fast.ai: Deep Learning Part 1 (v2) (2018)

7 Video lesson transcripts based on youtube.com autogenerated subtitles.

Discussion forum: http://forums.fast.ai/.

Version 0.1 (2018-06-28) (proofreading progress: 0%).

If you'd like to contribute with proofreading, please see:
https://github.com/stas00/fastai-transcript/.

Credits: Jeremy Howard (fast.ai Teacher/Founder), Eric Perbos-Brinck (Video Timelines), Ottokar Tilk (Punctuator), Stas Bekman (Transcript making).

Table of Contents

Table of Contents

Lesson 1: Recognizing cats and dogs

Outline

We learn today how to classify dogs from cats. Rather than understanding the mathematical details of how this works, we start by learning the nuts and bolts of how to get the computer to complete the task, using 'fine-tuning', perhaps the most important skill for any deep learning practitioner. In a later lesson we'll learn about how fine-tuning actually works "behind the scenes".

An important point discussed is how the data for this lesson needs to be structured. This is the most important step for you to complete—if your data is not structured correctly you will not be able to train any models.

Video Timelines and Transcript

1. 00:00:01

- **Welcome to Part 1, Version 2 of "Practical Deep Learning for Coders",**
- **Check the Fastai community for help on setting up your system on "forums.fast.ai"**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Hi everybody welcome to practical, deep learning for coders. This is part one of our two-part course. I'm presenting this from the data Institute in San Francisco will be doing seven lessons in this part of the course most of them will be about a couple of hours long. This first one may be a little bit shorter, practical, deep learning for coders is all about getting you up and running, with deep learning in practice, getting world-class results and it's a really coding focused approach, as the name suggests, but we're not going to dumb it down By the end of the course, you all have learned all of the theory in details that are necessary to rebuild all of the world-class results, we're learning about from scratch. Now I should mention that our videos are hosted on YouTube, that we strongly recommend watching them via our website, at course fast a I, although they're exactly the same videos, the important thing about watching them through our website is that you'll get all of the information you Need about kind of updates to libraries, my own locations, further information, frequently asked questions and so forth. So, if you're currently on YouTube watching this, why don't you switch over to crosstalk fast at AI now and start watching through there and make sure you read all of the material on the page before you start just to make sure that you've got everything you need? The other thing to mention is that there is a really great strong community at forums faster.

I, from time to time you will find that you get stuck, you may get stuck very early on. You may not get stuck for quite a while, but at some point you might get stuck with understanding why something works the way it does, or there may be some computer problem that you have

or so forth, on forums don't fast at AI. There are thousands of other learners talking about every lesson and lots of other topics. Besides, it's the most active, deep learning community on the internet by far so definitely register there.

2. 00:02:11

- **The “Top-Down” approach to study, vs the “Bottom-Up”,**
- **Why you want a nVidia GPU (Graphic Processing Unit = a video card) for Deep Learning**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

And start getting involved, you'll get a lot more out of this course. If you do that, so we're going to start by doing some coding. This is an approach. We're going to be talking about in the moment called the top-down approach to study. But let's learn it by doing it, so let's go ahead and try and actually train a neural network. Now, in order to train a neural network, you almost certainly want a GPU GPU of is a graphics processing, a graphics, processing unit. It's the things that companies use to help you play games better. They let your computer render the game much more quickly than your CPU. Okay, we'll be talking about them more shortly, but for now I'm going to show you how you can get access to a GPU. Specifically, you're going to need an NVIDIA GPU, because only NVIDIA GPUs support something called cooter cooter is the language and framework that nearly all deep learning, libraries and practitioners use to do their work. Obviously, it's not ideal that we're stuck with one particular vendors cards and over time we hope to see more competition in this base. But for now we do need an NVIDIA GPU. Your laptop almost certainly doesn't have one unless you specifically went out of your way to buy like a gaming laptop. So almost certainly you will need to rent one, and the good news is that renting access paying by the second or a GPU based computer is pretty easy and pretty cheap. I'm going to show you a couple of options.

The first option, I'll show you, which is probably the easiest, is called Crestle. If you go to kress allcom and click on

3. 00:04:11

- **Use crestle.com if you don’t have a PC with a GPU.**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Sign up or if you've been there before sign-in, you will find yourself at this screen, which has a big button. That says start jupiter and another switch called enable GP you. So if we make sure that is set to true the Nabal GPU is on and we click start Jupiter and we click start Jupiter. It's going to launch us into something called Jupiter notebook, Jupiter notebook in a recent survey of tens of thousands of data. Scientists was rated as the third most important tool in the data scientist toolbox. It's really important that you get to learn it well, and all of our courses will be run through Jupiter. Yes, Rachel, you have a question or a comment. I just wanted to point out that you get, I believe, 10 3 hours. So if you wanted to tricresyl out yeah, he might have changed that recently to less hours, but you can check the FAQ or the pricing, but you certainly get some three hours. The pricing varies because this is actually runs on top

of Amazon Web Services. So at the moment, it's 60 cents an hour. The nice thing is, though, that you can always turn it turn it on. You know start your Jupiter without the CP, without the GPU running and pay you a tenth of that price, which is pretty cool, so Jupiter notebook is something we'll be doing all of this course in and so to get started here. We're going to find our particular course so we're go to courses and would go to fast, a o2 and there they are.

Things have been moving around a little bit, so it may be in a different spot for you when you look at this and we'll make sure all the information current information is on the website. Now. Having said that, that's you know. The crystal approach is, you know, as you can see, it's basically instant and and easy, but if you've got you know an extra hour,

4. 00:06:11

- **Use paperspace.com instead of crestle.com, for faster and cheaper GPU computing. Technical hints to make it work with a Jupyter Notebook.**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Or so to get going and even better option is something called paper space paper space unlike Chris or doesn't run on top of Amazon. They have their own machines and if I click on so here's here's paper space. And so, if I click on new machine, I can pick which one of the three data centers to use so pick that one closest to you so I'll say a West Coast and then I'll say Linux and I'll say Ubuntu 16. And then it says, choose machine and you can see there's various different machines. I can choose from and pay-by-the-hour. So this is pretty cool for 40 cents an hour, so it's cheaper than cresol. I get a machine, that's actually going to be much faster than Crescent 67. Our machine or for 65 cents an hour way way way faster, all right, so I'm going to actually show you how to get started with the paper space approach, because that actually is going to do everything from scratch. You may find, if you trailers, do the 65 cents an hour, one that it may require you to contact paper space to say like. Why do you want it and it's just an anti fraud, the thing so, if you say faster AI there, then they'll quickly get you up and running. So I'm going to use the cheapest one here, 40 cents an hour. You can pick how much draw it. You want and note that you pay for a month of storage as soon as you start the machine up.

Alright, so don't start and stop lots of machines, because each time you pay for that month of storage, I think the 250 gig $ 7 a month option is pretty good, but you only need 50 gig. So if you're trying to minimize the price you can go there, the only other thing you need to do is turn on public IP, so that we can actually log into this, and we can turn off auto snapshot to save the money or not having backups all Right, so if you then click on create your paper space about a minute later, you will find that your machine will pop up. Here is my Ubuntu 1604 machine. If you check your email, you will find that they have emailed you a password, so you can copy that and you can go to your machine and enter your password now to paste the password you would press ctrl shift V or on Mac against Apple shift V. So it's slightly different to normal pasting or of course you can just type it in, and here we are now we can make a little bit more room here by clicking on those little arrows. There can zoom in a little bit, and so, as you can see, we've got like a terminal, that's sitting inside our browser, it's just kind of quite a handy way to do it so now we need to configure this further course and so the way you can Figure it for the course is you type curl, HTTP colon, slash, slash files, dot, fastai, slash setup, slash paper, space, type, okay, and so that's then, going to run a script

which is going to set up all of the crudo drivers, special Python, Reaper, Python distribution.

We use called anaconda all of the libraries or other courses and the data we use for the first part of the course ok, so that takes an hour or so and when it's finished, running you'll need to reboot your computer. So to reboot, not your own computer, but your pages paste computer, and so to do that. You can just click on this little circular restart machine button, ok and when it comes back up, you'll be ready to go. So what you'll find is if you've now got an anaconda three directory. That's where your python is. You've got a data directory which contains the data for the first part of this course. First lesson, which is their dogs and cats and you've got a fastai directory, and that contains everything for this course. So what you should do is CD fastai and from time to time you should go get Paul and that will just make sure that all of your fastai stuff is up-to-date and also from time to time. You might want to just check that your Python libraries are up-to-date, and so you can type Condor and update to do that. Alright, so make sure that you've CD into fastai and then you can type Jupiter notebook all right there. It is so we now have a Jupiter notebook servant running and we want to connect to that, and so you can see here it says copy paste this URL into your browser when you connect.

So if you double click on it, then that will actually that will actually copy it for you, then you can go and paste it, but you need to change this local host to be the paper space IP address. So if you click on the little arrows to go smaller, you can see the IP addresses here so I'll, just copy that and paste it where it used to say local host okay. So it's now HTTP and then my IP and then everything else a copied before and so there it is so this is the faster I get repo and our courses are all in courses and in there the deep learning part one is DL one and in there You will find lesson one by Mb, ipython

5. 00:12:30

- **Start with Jupyter Notebook lesson1.ipynb 'Dogs vs Cats'**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Notebook so here we are ready to go depending whether you're using Crestle or paper space or something else. If you check course it's not fast today, I will keep putting additional videos and links to information about how to set up other. You know good Jupiter, notebook providers as well, so to run a cell in Jupiter notebook, you select the cell and you hold down shift and press Enter or if you've got the tool bar showing you can just click on the little Run button. So you'll notice that some cells contain code and some contain text and some contain pictures and some contain videos, so this environment basically has you know it's a it's a way that we can give you access to and a way to run experiments and to kind of Tell you what's going on so pictures. This is why it's like a super popular tool in data science, a data science is kind of all about running experiments. Really so, let's go ahead and click run and you'll see that cell turn into a star the one turn into a star for a moment and then it finished running okay. So let's try the next one. This time, instead of using the toolbar, I'm going to hold down shift and press ENTER, and you can see again it turn into a star and then it said to so. If I hold down shift and keep pressing enter, it just keeps running each so right. So I can put anything I like, for example, one plus one is two.

So what we're going to do is we're going to yes Rachel. This is just a side note, but I wanted

to point out that we're using Python 3 here, yes, thank you, pythons are still using Python, mmhmm yeah um and it is important to switch to Python 3. You know now well for Class A I you require it, but you know increasingly, a lot of libraries are removing support for Python thanks Rachel. Now it mentions here that you can download the data set for this lesson from this location if you're using crystal or the paper space script that we just used to set up that this will already be and made available for you. Okay, if you're not you'll, need to wget it as now, Crestle is quite a bit slower than paper space and also it there are some particular things. It doesn't support that we really need, and so there are a couple of extra steps if you're using cresol, you have to run two more cells right, so you can see these are commented out there quite hashes at the start. So if you remove the hashes from these and run these two additional cells that just runs the stuff that the stuff that you only need for crystal I'm using paper space, so I'm not going to run it. Ok, so inside our data. So we set up this path to data, slash dogs, cats, that's pre, set up for you and so inside there you can see here. I can use an exclamation mark to basically say I don't want to run Python, but I want to run bash right.

I want to run shell, so this runs a bash command and the bit inside the curly brackets actually refers, however, to a python variable so inserts that python variable into the batch command. So here's the contents of our folder there's a training set and a validation set. If you're not familiar with the idea of training, sets and validation sets, it would be a very good idea to check out our practical machine learning course, which tells you a lot about this kind of stuff. If, like yeah the basics of how to setup and run machine learning projects more generally, would you recommend that people take that course before this one actually a lot of students? Who would you know, as they went through these, as said they'll they've liked doing them together? So you can kind of check it out and see the machine learning course yeah. They cover with some similar stuff, but all in different directions. So people have done both seen, you know say they find it. They they each support each other. I wouldn't say it's a prerequisite, but you know if I do, if I say something like hey, this is the training set and this is a validation set and you're going. I don't know what that means, at least Google, but do a quick read. You know, because we're assuming that you know the very basics of kind of what machine learning is and does to some extent - and I have a whole blog post on this topic as well.

Okay and we'll make sure that you link to that from Pastor day night and as we just wanted to say in general with fasting, our philosophy is to kind of learn things on an as-needed basis, yeah, exactly don't try and learn everything that you think you might Need first, otherwise, you'll never get around elite learning the stuff. You actually want to learn exactly that shows up in deep learning. I think particularly a lot yes, okay, so in our validation, folder, there's a cat's folder and a dog's folder, and then inside the validation cats. Folder is a whole bunch of JPEGs. The reason that it's set up like this is that this is kind of the most common standard approach for how image classification, datasets, shared and provided, and the idea is that each folder tells you the label. So there's each of these images is labeled cats and each of the images and the dogs folder is labelled dogs, okay. This is how chaos works as well, for example. So this is a pretty standard way to share image classification files, so we can have a look. So if you go plot dot a.m. show, we can see an example of the first of the cats. If you haven't seen this before this is a Python 3.6 format string. So you can google for that. If you haven't seen it, it's a very convenient way to do string formatting and we use it a lot. So there's no cat but we're going to mainly be interested in the underlying data that makes up that cat.

So, specifically, it's an image whose shape that is the dimensions of the array is 198 by 1/7. 9X3. Is it's a three dimensional array plus a quarter rank three tensor and here are the first

four rows and four columns of that image. So, as you can see, each of those cells has three items in it, and this is the red, green and blue pixel values between naught and 255. So here's a little subset of what a picture actually looks like inside your computer. So that's that that's will be. Our idea is to take these kinds of numbers and use them to predict whether those kinds of numbers represent a cat or a dog based on looking at lots of pictures of cats and dogs. So that's a pretty hard thing to do and at the point in time when this this this data set actually comes from a caracal competition, the dogs versus cats, caracal competition and when it was released in, I think it was 2012. The state of the art was 80 % accuracy, so computers weren't really able to at all accurately recognize dogs versus cats. So let's go ahead and train

6. 00:20:20

- **Our first model: quick start.**
- **Running our first Deep Learning model with the 'resnet34' architecture, epoch, accuracy on validation set.**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

A model so here are the three lines of code necessary to train a model, and so let's go ahead and run it. So I'll click on this on the cell, I'll press shift enter and then we'll wait a couple of seconds for it to pop up and there it goes okay and it's training, and so I've asked it to do three epochs. So that means it's going to look at every image three times in total or look at the entire set of images three times. That's what we mean by an epoch and as we do it's going to print out the accuracy is: that's lasted for three numbers. It prints out on the validation set. Okay, the first three numbers we'll talk about later. In short, they're, the value of the loss function, which is in this case the cross-entropy loss for the training set and the validation set, and then right at the start. Here is the epoch number, so you can see it's getting about 90 % see and it took 17 seconds. So you can see. We've come a long way since 2012 and in fact, even in the competition, this actually would have won the caracal competition of that time. The best in the caracal competition was 98.9 and we're getting about 99 %. So this play surprised you that we're getting a you know: Kaggle winning as of 20 and of 2012 early 2013 kaggle winning image classifier in 17 seconds that and three lines of code, and I think that's because, like a lot of people assume that deep learning takes A huge amount of time and lots of resources and lots of data and as you'll learn in this course that in general, rule isn't true.

One of the ways we've made it much simpler is that this code is written on top of a library we built imaginatively called fastai. The faster a library is basically a library which takes all of the best practices approaches that we can find, and so each time a paper comes out, you know we that looks interesting. We test it out if it works well for a variety of data sets and we can figure out how to tune it. We implement it in fastai and so faster. I kind of curates all this stuff and packages up for you and much of the time, but most the time kind of automatically figures out the best way to handle things so the first day our library is why we were able to do this in just three Lines of code and the reason that we were able to make the faster I library work so well is because it interns it's on top of something called pytorch, which is a really flexible, deep learning and machine learning and GPU computation library written by Facebook. Most people are more familiar with tensorflow than pytorch, because google markets that pretty heavily but most of the top researchers I know nowadays at least the ones that are at Google - have switched across to pytorch. Yes, Rachel and we'll be covering some

pie torts later in the course yeah. It's I mean one of the things that hopefully you'll really like about last AI, is that it's really flexible, that you can use all these kind of curated best practices as much as little as you want, and so really easy to hook in at any point and Write your own data augmentation write, your own loss function, write your own network architecture whatever, and so we'll do all of those things in this course. So

7. 00:24:11

- **"Analyzing results: looking at pictures" in lesson1.ipynb**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

What does this model look like? Well, what we can do is we can take a look at. So what are the? What is the validation set, dependent variable? The Y look like, and it's just a bunch of zeros and ones, okay, so the zeros. If we look at data dot classes, the zeros represent cats, the ones represent dogs you'll, see here, there's basically two objects. I'm working with one is an object, called data which contains the validation and training data and another one is the object, called learn which contains the model right so anytime. You want to find something out about the data. We can look inside data, so we're going to get predictions through our validation set, and so to do that we can call learned predict, and so you can see here the first ten predictions and what it's giving you as a prediction for dog and a prediction for Cat now, the way PI, Torche, generally works and therefore fastai also works - is that most models return the log of the predictions rather than the probabilities themselves, we'll learn why that is later in the course. So for now recognize that to get your probabilities, you have to get e to the power of you'll. See here we're using numpy NP is none play if you're not familiar with numpy. That is one of the things that we assume that you have some familiarity with. So be sure to check out the material on cost. I passed at AI to learn the basics of number.

It's the way that handles all of the fast numerical programming array computation that kind of thing okay, so we can get the probabilities using that using MP, dot X and there's a few functions here that you can look at yourself if you're interested, that's just some flooding Functions that we'll use - and so we can now plot some random, correct images, and so here are some images that it's correct about. Okay, and so remember, one is a dog, so anything greater than 0.5 is dog and zero is a cat. So this is what 10 to the negative 5. Obviously a cat here are some which are incorrect. Alright, so you can see that some of these, which it thinks are incorrect, obviously are just the you know, images that shouldn't be there at all, but clearly this one, which it called a a dog, is not at all a dog. So there are some obvious mistakes. We can also take a look at which cats is it. The most confident are cats which dogs are the most doglike, the most confident dogs, perhaps more. Interestingly, we can also see which cats is that the most confident are actually dogs, so which ones it is it? The most wrong about and same thing for the ones the dog said, it really thinks of cats. And again some of these are just pretty weird. I guess there is a dog in there. Yes, Rachel, I see suit. You want to see more about why you would want to look at your data, yeah sure so yeah.

So finally, I just mentioned the last one we've got here is to see which ones have the probability closest to 0.5. So these are the ones that the model knows. It doesn't really know what to do with, and some of these it's not surprising so yeah I mean this is kind of like always. The first thing I do after I build a model is to try to find a way to like visualize what it's built, because if I want to make the model better, then I need to take advantage of the

things that's doing well and fix the things. That's doing badly, and so in this case and off this is the case. I've learned something about the data set itself, which is that there are some things that are in here, that probably shouldn't be, but I've also like it's also clear that this model has room to improve like to me. That's pretty obviously a dog, but one thing I'm suspicious about here is this: image is very kind of fat and short and, as we all learn the way these algorithms work is it kind of grabs, a square piece at a time. So this rather makes me suspicious that we're going to need to use something called data augmentation that we'll learn about learn about later to handle this properly. Okay. So that's it right! We've now built we've now built an image classifier and something that you should try now is to grab some data yourself. Some pictures of two or more different types of thing put them in different folders and run the same three lines of code on them.

Okay and you'll find that it will work for that as well as long as that. They are pictures of things like the kinds of things that people normally take photos of right. So if their microscope microscope pictures or pathology pictures or CT scans or something, this won't work very well as well learn about later. There are some other things we didn't need to do to make that work, but for things that look like normal photos, these you can run exactly the same three lines of code and just point your path variable somewhere else to get your own image classifier. So, for example, one student took those three lines of code downloaded for Google Images, ten examples of pictures of people playing cricket, ten examples of people playing baseball and build a classifier of those images which was new perfectly correct. The same student actually also tried downloading seven pictures of Canadian currency, seven pictures of American currency and again, in that case the model was a hundred percent accurate. So you can just go to Google Images if you like and download a few things of a few different classes and see see what works and tell us on.

8. 00:30:45

- **Revisiting Jeremy & Rachel's approach of "Top-Down vs Bottom-Up" teaching philosophy, in details.**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

The forum, both your successes and your failures, so what we just did was to train a neural network, but we didn't first of all, tell you what a neural network is or what training means or anything. Why is that? Well, this is the start of our top-down approach to learning, and basically the idea is that, unlike the way math and technical subjects, I usually talk where you learn every little element piece by piece and you don't actually get to put them all together and build your Own image, classifier until third year of graduate school, our approach is to say from the start: hey, let's show you how to train an image classifier and you can start doing stuff and then gradually we dig deeper and deeper and deeper, and so the idea is that Throughout the course you're going to see like new problems that we want to solve so, for example, in the next lesson, we'll look at well what, if we're, not looking at normal kinds of photos, but we're looking at satellite images and we'll see why it is that This approach that we're learning today doesn't quite work as well, and what things do we have to change, and so we'll learn enough about the theory to understand why that happens, and then we'll learn about the libraries and how we can change change things with the libraries To make that work better, and so during the course we are gradually going to learn to solve more and more problems as we do so we all need to learn more and more parts of the library more and more

bits of the theory until by the end, We're actually going to learn how to create a world plus neural net architecture from scratch and our own training loop from scratch and so were actually built everything ourselves.

So that's the general approach, yes, Rachel and sometimes also call this the whole game, which is inspired by harvard professor david perkins yeah, and so the idea with the whole game is like. This is more like how you would learn, baseball or music with baseball. You would get taken to a ball game, you would learn what baseball is, you would start playing it, and it would only be years later that you might learn about the physics of how curveball works, for example. Well, with music, we put a instrument in your hand, and you start banging the drum or hitting the xylophone, and it's not until years later, that you learn about the circle of fifths and understand how to construct a cadence example so yeah, so that this is kind Of the approach we're using it's very inspired by David Perkins and other writers of Education, so what that does mean is to take advantage of this as we peel back the layers, we want you to keep

9. 00:33:45

- **Explaining the "Course Structure" of Fastai, with a slide showing its 8 steps.**
- **Looking at Computer Vision, then Structured Data (or Time Series) with the Kaggle Rossmann Grocery Sales competition, then NLP (Natural Language Processing), then Collaborative Filtering for Recommendation Systems, then Computer Vision again with ResNet.**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Like looking under the hood yourself as well like experiment a lot now, because this is a very code driven approach, so here's basically what happens right. We start out looking today at convolutional neural networks for images and then in a couple of lessons, we'll start to look at how to use neural nets, to look at structured data and then look at language data and then look at recommendation system data. And then we kind of then take all of those depths and we go backwards through them in reverse order. So now you know by the end of that fourth piece: you will know by the end of lesson four: how to create a world-class image. Classifier, a world-class structured data analysis program, world-class language, classifier, broad class recommendation system and then we're going to go back over all of them again and learn in depth about like well. What exactly did it do and how to do work, and how do we change things around and use it in different situations for for recommendation systems, structured data images and then, finally, back to language? So that's how it's going to work. So what that kind of means is that most students find that they tend to watch the videos two or three times but not like watch lesson, one two or three times and listen to two or three times in verse and three three times but like they do. The whole thing into end lessons one through seven and then go back and start lesson one again, that's an approach which a lot of people find when they want to go back and understand all the details enough.

That can work pretty well. So I would say you know, aim to get through to the end of lesson. Seven, you know as as quickly as you can, rather than aiming to fully understand every detail from this data. So basically, the plan is that, in today's lesson, you'll learn in as few lines of code as possible with as few details as possible. How do you actually build an image classifier with deep learning to do this to, in this case, say: hey here are some pictures of dogs as

opposed to pictures of cats. Then we're going to learn how to look at different kinds of images, and particularly we're going to look at images of from satellites and we're going to say for a satellite image. What kinds of things might you be seeing in that image and there could be multiple things that we're looking at so a multi-label classification problem from there we'll move to something which is perhaps the most widely applicable for the most people, which is looking at what we Call structured data so data about data that kind of comes from databases or spreadsheets. So we're going to specifically look at this data set of predicting sales, the number of things that are sold at different stores on different dates, based on different holidays and and so on and so forth, and so we're going to be doing its sales forecasting exercise.

After that, we're going to look at language and we're going to figure out what this person thinks about the movie is on be given and will be able to figure out how to create. Just like we create image classifiers for any kind of image will learn to create in NLP classifiers, to classify any kind of language in lots of different ways. Then we'll look at something called collaborative filtering, which is used mainly for recommendation systems. We're going to be looking at this data set that showed four different people for different movies. What rating did they give it, and here are some of the movies, and so this is maybe an easier way to think about it is there are lots of different users and lots of different movies and then for each one? We can look up for each user. How much they liked that movie and the goal will be, of course, to predict for user movie combinations we haven't seen before. Are they likely to enjoy that movie or not? And that's the really common approach used for like deciding what stuff to put on your homepage? When somebody's visiting you know what book might they want to read or what film might they want to see or so forth from there? We could have then dig back into language a bit more and we're going to look at actually we're gon na look at the writings of Nietzsche, the philosopher and learn how to create our own Nietzsche philosophy from scratch character by character.

So this here, perhaps that every life values a blood of intercourse when it senses there is unscrupulous who's very right sense to impulse love is not actually Nietzsche. That's actually like some character by character, generated text that we built with this recurrent neural network and then, finally, we're going to loop all the way back to computer vision. Again, we're going to learn how not just to recognize cats from dogs how to actually find like where the cat is with this kind of hate map, and we're also going to learn how to write our own architectures from scratch. So this is an example of a resonate, which is the kind of network that we are using in today's lesson for computer vision and so we'll actually end up building the network and the training loop from scratch and so they're. Basically, the the steps that we're going to be taking from here and at each step we're going to be getting into increasing amounts of detail about how to actually do these things yourself. So we've actually heard that from our students of past courses about what they've found and one of the things that we've heard, a lot of students say is that there's been too much time on theory and research and not enough time running the code and even after We tell people about this morning where they still come to the end of the course not and say I wish I had taken more seriously that advice, which is to keep running code, so these are actual quotes from our forum. In retrospect, I should have spent the majority of my time on the actual code and the notebooks see what goes in see what comes out now.

This idea that you can create world-class models in a code, first approach, learning what you need as you go, is very different to a lot of the advice. You're read out there, such as this person on the forum hacker news who claimed that the best way to become an m/l engineer is

to learn all of math and C and C++ learn. Parallel programming learn ml algorithms, implement them yourself using plain C and finally start doing ml. So we would say, if you want to become an effective practitioner, do exactly the opposite of of this. Yes, Rachel yeah, I'm just highlighting that this is. We think this is bad advice, and this can be very discouraging for a lot of people to come across this yeah yeah. It's it's it's it's! You know. We now have thousands and more tens of thousands of people that have done this course and have lots and lots of examples of people who are now running. Research, labs or Google brain residence, or you know, have created patents based on deep learning and so forth. Who have done it by doing this course, so the top-down approach works super well now one thing to mention is like we've: we've now already learned how you can actually train a world-class image classifier in 17 seconds. I should mention by the way the first time you run that code. There are two things it has to do that take more than 17 seconds. One is that it downloads a pre trained model from the internet, so you'll see the first time you run it it'll, say downloading model, so that takes a minute or two also the first time you run it.

It pre computes and caches some of the intermediate information that it needs and that takes about a minute and a half as well. So if the first time you run it, it takes three or four minutes to download and pre compute stuff, that's normal. If you run it again, you should find it takes 20 seconds or so so image classifiers. You know you may not feel like you need to recognize cats versus dogs very often on a computer. You can probably do it yourself pretty well, but what's interestingly interesting, is that these image classification, algorithms, are really useful for lots and lots of things, for example alphago, which became which beat the go world champion. The way it worked was to use something at its heart that looked almost exactly like our dogs, vs. cats, image classification, algorithm. It looked at thousands and thousands of go boards, and at for each one, there was a label saying whether that go board ended up being the winning or the losing player, and so it learnt basically an image classification that was able to look at a go board And figure out whether it was a good group or a bad code board, and that's really the key most important step in playing Gowell is to know which, which move is better. Another example is one of our earlier students who actually got a couple of patterns for this work.

Looked at anti-fraud, he had lots of all of his customers mouths movements because they they provided kind of these user tracking software to help avoid fraud, and so he took the the mouse paths basically of the users on his customers. Websites turn them into pictures of where their mouse moved and how quickly it moved and then built a image classifier that took those images as input and as output. It was was that a fraudulent transaction or not and turned

10. 00:44:11

- **What is Deep Learning ? A kind of Machine Learning.**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Out to go, you know really great results for his company, so image classifiers are like much more flexible than you might imagine so so this is how you know some of the ways you can use deep learning, specifically for image recognition and it's worth understanding that deep Learning is not, you know, just a word. That means the same thing as machine learning right like what is it that we're actually doing here when we're doing deep learning. Instead, deep

learning is a kind of machine learning, so machine learning was invented by this guy Arthur Samuels, who was pretty amazing in the late 50s. He got this IBM mainframe to play checkers better than he can and the way he did it was. He invented machine learning. He got the mainframe to play against itself lots of times and figure out which kinds of things led to victories and which kinds of things didn't and use that to kind of almost write. Its own program and Arthur Arthur Samuels actually said in 1962 that he thought that one day the vast majority of computer software would be written using this machine learning approach rather than written by hand by writing the loops and so forth by hand. So I guess that hasn't happened yet, but it seems to be in the process of happening. I think one of the reasons it didn't happen for a long time is because traditional machine learning actually was very difficult and very knowledge and time intensive.

So, for example, here's something called the computational, pathologist or C path from backwater and II back and II back back when he was at Stanford, he's now moved on to somewhere on the East Coast, no Harvard, I think, and what he did was he took these pathology Slides of breast cancer biopsies right and he worked with lots of pathologists to come up with ideas about what kinds of patterns or features might be associated with so long-term survival versus versus dying quickly. Basically, and so he came up with these ideas like well, they came up with these ideas, like relationship between epithelial, nuclear neighbors, relationship between epithelial and stromal objects and so forth, and so they came up with all of these ideas of features. These are just a few of the hundreds that they thought of and then lots of smart computer programmers wrote specialist algorithms to to calculate all these different features and then those those features were passed into a logistic regression to predict survival and it ended up working very Well - and it ended up that the survival predictions were more accurate than pathologists own survival, predictions work and so machine learning can work really well. But the point here is that this was a an approach that took lots of domain experts and computer experts.

Many years of work to actually to build this thing right, so we really want something something better, and so, specifically, I'm going to show you something which, rather than being a very specific function with all this very domain-specific, feature engineering, we're going to try and create an Infinitely flexible function, a function that could solve any problem right. It would solve any problem if only you set the parameters of that function correctly, and so then we need or purpose way of setting the parameters of that function, and we would need that to be fast and scalable right now. If we had something that had these three things, then you wouldn't need to do this incredibly time and domain knowledge intensive approach anymore. Instead, we can learn all of those things with this with this algorithm. So, as you might have guessed, the algorithm in question, which has these three properties, is called deep learning or it's not an algorithm, then maybe we will call it a class of algorithms. Let's look at each of these three things in turn, so the underlying function that deep learning uses is something called the neural network. Now the neural network we're going to learn all about it and implemented ourselves from scratch later on in the course, but for now all you need to know about it is that it consists of a number of simple linear layers, interspersed with a number of simple nonlinear Layers and when you in dispersed these layers in this way,

11. 00:49:11

- **The Universal Approximation Theorem, and examples used by Google corporation.**

You get something called the universal approximation theorem and the universal approximation theorem says that this kind of function can solve any given problem to arbitrarily close accuracy. As long as you add enough parameters, so it's actually provably shown to be an infinitely flexible function. Okay, so now we need some way to fit the parameters so that this infinitely flexible neural network solves some specific problem, and so the way we do, that is using a technique that probably most of you will have come across before at some stage called gradient descent And with gradient descent, we basically say: okay well for the different parameters. We have how: how good are they at solving my problem and, let's figure out a slightly better set of parameters and a slightly better set of parameters and j6v follow down the the surface of the loss function downwards. It's kind of like a marble going down, find the minimum and, as you can see here, depending on where you start, you end up in different places. These things a court local minima. Now, interestingly, it turns out that for neural networks, particularly in particular, there aren't actually multiple different local minima, there's, basically just there's, basically just one right or think of it. Another way there are different parts of the space which are all equally good, so gradient. Descent, therefore turns out to be actually an excellent way to solve this problem of fitting parameters to neural networks.

The problem is, though, that we need to do it in a reasonable amount of time, and it's really only thanks to GPUs that that's become possible, so GPUs. This shows over the last few years, how many gigaflops per second can you get out of a GPU? That's the red and green versus a CPU. That's the blue right, and this is on a log scale. So you can see that, generally speaking, the GPUs are about 10 times faster than the CPUs, and, what's really interesting is that nowadays, not only is the Titan X about 10 times faster than the e5 to $ 6.99 CPU, but the Titan X well actually better one To look at would be the GTX 1080i GPU costs about 700 bucks, whereas the CPU, which is 10 times slower costs over $ 4,000. So GPUs turn out to be able to solve these neural network parameter fitting problems incredibly quickly and also incredibly cheaply. So they've been absolutely key in bringing these three pieces together, then there's one more piece which is I mentioned, that these neural network, so you can intersperse multiple sets of linear and then nonlinear layers. In the particular example, that's drawn here, there's actually only one what we call hidden layer, one layer in the middle and something that we learned in the last few years is that these kinds of neural networks, although they do support the universal approximation theorem, they can solve Any given problem arbitrarily closely, they require an exponentially increasing number of parameters to do so, so they don't actually solve the fast and scalable for even reasonable size problems.

But we've since discovered that, if you create add multiple hidden layers, then you get super linear scaling. So you can add a few more hidden layers to get multiplicatively more accuracy, 2ma duplicative, lis, more complex problems, and that is where it becomes called deep learning. So deep learning means a neural network with multiple hidden layers. So when you put all this together, there's actually really amazing what happens? Google started investing in deep learning. In 2012, they actually hired Geoffrey Hinton who's kind of the father of deep learning and his top student Alex Bogusky, and they started trying to build a team that team became known as Google brain and because things with these three properties are so incredibly powerful. And so incredibly flexible, you can actually see over time how many projects at Google use deep learning. My graph here only goes up through a bit over a year ago, but it's I know it's been continuing to grow exponentially since then as well, and so what

you see now is around Google that deep learning is used in like every part of the business, and So it's really interesting to see how the this kind of simple idea that we can solve machine learning problems using a an algorithm that has these properties when a big company invests heavily in actually making that happen. You see this incredible growth in how much it's used.

So, for example, if you use the inbox by Google software, then when you receive an email from somebody, it will often tell you here are some replies that I could send for you, and so it's actually using deep learning here to read the original email and to Generate some suggested replies and so like this is a really great example of the kind of stuff that previously just wasn't possible. Another great example would be: Microsoft is also a little bit more recently invested heavily in deep learning, and so now you can use Skype. You can speaking to it in English and ask it at the other end to translate it in real time to Chinese or Spanish and then, when they talk back to you in Chinese or Spanish, Scott will in real-time translated the speech in in their language into English Speech in real-time and again, this is an example of stuff which we can only do thanks to deep learning and something is really interesting to think about how deep learning can be combined with human expertise. So here's an example of low drawing something just sketching it out and then using a program called neural doodle. This is from a couple of years ago then say: please, take that sketch and render it in the style of an artist and so here's the picture that have been created, rendering it, as you know, impressionist painting, and I think this is a really great example of How you can use deep learning to help combine human expertise and what computers are good at so I, a few years ago decided to try this myself like what would happen if I took think learning and tried to use it to solve a really important problem, and So the problem I picked was diagnosing lung cancer.

It turns out, if you can find lung nodules earlier, there's a 10 times higher probability of survival. So it's a really important problem to solve, so I got together with three other people. None of us had any medical background and we grabbed a data set of CT scans. We used to compilation or neural network much like the dogs vs. cats, one we trained at the start of today's lesson to try and predict which CT scans had malignant tumors in them, and we ended up after a couple of months with something with a much lower False negative rate and a much lower false positive rate than a panel with four radiologists, and we went on to build this in a start-up in just into a company called analytic, which has really become pretty successful. And since that time, the idea of using deep learning for medical imaging has become hugely popular and is

12. 00:58:11

- **More examples using Deep Learning, as shown in the PowerPoint from Jeremy course in ML1 (Machine Learning 1)**
- **What is actually going on in a Deep Learning model, with convolutional network.**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Being used all around the world, so what I've generally noticed is that you know the vast majority of of kind of things that people do in the world currently aren't using deep learning and then each time somebody says: oh, let's try using deep learning to improve performance At this thing they nearly always get fantastic results and then suddenly everybody in that industry starts using it as well. So there's just lots and lots of opportunities here at this

particular time to use deep learning to help with all kinds of different stuff. So I've jotted down a few ideas here. These are all things which I know you can use deep learning for right now to get good results from, and you know, are things which people spend a lot of money on or have a lot of. You know important business opportunities, there's lots more as well that these are some examples of things that maybe your company, you could think about applying deep learning for so, let's talk about what's actually going on what actually happened when we trained that deep learning model earlier and So, as I briefly mentioned, the thing we created is something called a convolutional neural network or CNN, and the key piece of a convolutional neural network is the convolution. So here's a great example from our website. I've got the URL up here, explained visually, it's called, and the explained visually website has an example of a convolution kind of.

In fact, this over here in the bottom left is a very zoomed in picture of somebody's face, and over here on, the right is an example of using a convolution on that image. You can see here. This particular thing is obviously finding edges the edges of his head about top and bottom edges in particular. Now, how is it doing that? Well, if we look at each of these are all three by three areas. This is moving over it's taking each three by three area of pixels, and here are the pixel values right for each thing in that 3x3 area and it's multiplying each one of those 3 by 3 pixels by each one of these 3 by 3 kernel values. In a convolution, this specific set of 9 values is called a kernel. It doesn't have to be 9, it could be 4 by 4 or 5 by 5 or 3 by 2 or whatever right, in this case, it's a 3 by 3 kernel and in fact, a deep learning. Nearly all of our kernels are 3 by 3. So in this case the kernel is 1. 1 /, 1, 2. 1. So we take each of the black through white pixel values and we multiply, as you can see, each of them by the corresponding value in the kernel and then we add them all together. And so, if you do that for every 3x3 area, you end up with the values you see over here. On the right hand, side, okay, so very low values become black, very high values become white, and so you can see when we're at an edge where it's black at the bottom and white at the top. We're obviously going to get higher numbers over here and vice versa. Okay, so that's a convolution! So, as you can see, it is a linear operation and so based on that definition of a neural net I described before this can be a layer in our neural network.

It is a simple linear operation and we're going to look much more at convolutions later, including building a little spreadsheet that implements them ourselves.

13. 01:02:11

- **Adding a Non-Linear Layer to our model, sigmoid or ReLu (rectified linear unit), SGD (Stochastic Gradient Descent)**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

So the next thing we're going to do is we're going to add a nonlinear layer. So a non-linearity as it's called, is something which takes an input value and turns it into some different value in a nonlinear way, and you can see this orange picture here is an example of a nonlinear function. Specifically, this is something called a sigmoid, and so a sigmoid is something that has this kind of S shape, and this is what we used to use as our nonlinearities in neural networks. A lot actually nowadays we're nearly entirely use. Something else called a rally or rectified linear unit, a rail. U is simply take any negative numbers and replace them with 0 and leave any positive numbers as they are so, in other words, in code that would be y equals max

X, comma, 0, so max X, comma 0 simply says, replace the negatives with 0. Now, regardless of whether you use a sigmoid or a RAL, you or something else, the key point about taking this combination of a linear layer, followed by a element-wise nonlinear function, is that it allows us to create arbitrarily complex shapes, as you see in the bottom right And the reason, why is that - and this is all from Michael Nelson's, neural networks and deep learning com - really fantastic, interactive book as you change the values of your linear functions, it basically allows you to kind of like build these arbitrarily tall or thin blocks and then Combine those blocks together, and this is actually the essence of the universal approximation theorem.

This idea that when you have a linear layer feeding into a non-linearity, you can actually create these arbitrarily complex shapes. So this is the key idea behind why neural networks can solve any computable problem. So then we need a way as we described to actually set these parameters. So it's all very well, knowing that we can move three, a meters around manually to try to create different shapes, but we have some specific shape. We want how do we get to that shape, and so, as we've discussed earlier, the basic idea is to use something called gradient descent. This is an extract from a notebook actually one of the fastai lessons, and it shows actually an example of using gradient descent to solve a simple linear regression problem, but I can show you the basic idea. Let's say you were just you had a simple quadratic. All right - and so you are trying to find the minimum of this quadratic and so in order to find the minimum you start out by randomly picking some point all right. So we say: okay, let's pick, let's pick here and so you go up there and you calculate the value of your quadratic at that point. So what you now want to do is try to find a slightly better point. So what you could do is you can move a little bit to the left and a little bit to the right to find out which direction is down and what you'll find out is that moving a little bit to the left decreases the value of the function. So that looks good right and so, in other words, we're calculating the derivative.

There's a function at that point right, so that tells you which way is down it's the gradient, and so now that we know that going to the left is down. We can take a small step in that direction to create a new point, and then we can repeat the process and say: okay, which way is down now, and we can now take another step and another step and another step, another step, another step. Okay and each time we're getting closer and closer, so the basic approach here is to say: okay, we start we're at some point. We've got some value X, which is our current guess right, that's at time, step n! So then, our new guest at time, step n plus 1, is just equal to our previous guess, plus the derivative right times some small number, because we want to take a small step. We need to pick a small number, because if we picked a big number right, then we say: okay, we know we want to go to the left. Let's jump a big long way to the left. We could go all the way over here and we actually end up worse all right and then we do it again now we're even worse again right. So if you have too high a step size, you can actually end up with divergence rather than convergence. So this number here we're going to be talking about it - a lot during this course and we're going to be writing all this stuff out in code from scratch ourselves. But this number here is called the learning rate. Okay, so you can see here.

This is an example of basically starting out with some random line and then using gradient descent to gradually make the line better and better and better. So what happens when you combine these ideas right, the convolution, the non-linearity and gradient descent, because they're all tiny small, simple little things? It doesn't sound that exciting. But if you have enough of these kernels right with enough layers, something really interesting happens and

14. 01:08:20

- **A paper on "Visualizing and Understanding Convolutional Networks", implementation on 'lesson1.ipynb', 'cyclical learning rates' with Fastai library as "lr_find" or learning rate finder.**
- **Why it starts training a model but stops before 100%: use Learner Schedule Finder.**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

We can actually draw them, so here's the so. This is a really interesting paper by Matt, Siler and Rob Fergus and what they did a few years ago was they figured out how to basically draw a picture of what each layer in a deep learning net Network learned, and so they showed that layer. One of the network here are nine examples of convolutional filters from layer, one of a trained network, and they found that some of the filters kind of learnt these diagonal lines or simple dat or grid patterns. Some of them learnt these simple gradients right and so for each of these filters they show nine examples of little pieces of actual photos which activate that filter quite highly right, so you can see layer, one these learnt, or ever these these are learnt using gradient descent. These filters were not programmed, they will learnt using gradient descent, so in other words, we were learning these nine numbers, so layer two then was going to take these as inputs and combine them together and so layer two had you know. This is like my in kind of attempts to draw one of the examples of the filters in layer two they're pretty hard to draw, but what you can do is say if the each filter, what are examples of little bits of images that activated them and you Can see by layer two we've got basically something that's being activated nearly entirely by little bits of sunset.

Some things, that's being activated by circular objects, something that's being activated by repeating horizontal lines, something that's being activated by corners, so you can see how we're basically combining layer. One features together. So if we combine those features together and again, these are all convolutional filters. Won't through gradient descent by the third layer, it's actually learn to recognize the presence of text. Another filter has learnt to recognize the presence of petals. Another filter has learnt to recognize the presence of human faces right. So just three layers is enough to get some pretty rich behavior so, but by the time we get to layer, five we've got something that can recognize the eyeballs of insects and birds and something that can recognize unicycle wheels. Alright. So so this is kind of where we start with something incredibly simple right, but if we use it as a big enough scale, thanks to the universal approximation theorem and the use of multiple hidden layers in deep learning, we actually get the very, very rich capabilities so That is what we used when we actually trained our little dog vs cat recognizer. Okay. So let's talk more about this dog vs cat recognizer, so we've learnt the idea of like we can look at the pictures that come out of the other end to see what the model is. Classifying well like, as I find badly or which ones it's unsure about, but let's talk about like this key thing I mentioned, which is the learning rate, so I mentioned we have to set this thing.

I just caught it L before the learning rate and you might have noticed, there's a couple of numbers, these kind of magic numbers here, the first one is the learning rate right. So this number is: how much do you want to multiply the gradient by when you're, taking each step in your gradient descent? We already talked about why you wouldn't want it to be too high right, but probably also it's obvious to see why you wouldn't want it to be too low. If you had

it too low, you would take like a little step and you'd be a little bit closer and little bits too little step little step, and it would take lots and lots and lots of steps, and it would take too long so setting this number. Well is actually really important and for the longest time this was driving deep learning, researchers crazy, because they didn't really know a good way to set this reliably. So the good news is last year a researcher came up with an approach to quite reliably set the learning rate. Unfortunately, almost nobody noticed so almost no deep learning, researchers I know about actually are aware of this approach, but it's incredibly successful and it's incredibly simple and I'll show you the idea right. It's built into the fastai library as something called @ lr find or the learning rate finder, and it comes from this paper. I was actually 2015 paper. Sorry, cyclic or learning rates for training, neural networks by a terrific researcher called Leslie Smith and I'll, show you Leslie's idea.

So Leslie's ideas started out with the same basic idea that we've seen before, which is, if we're going to optimize something pick. Some random point take its gradient all right and then, specifically, he said, take a tiny, tiny step like tiny step, so a learning rate of like 10 e next, seven all right and then do it again again, but each time increase the learning rate like double it. So then we try like to wean X, 7, 14 X, 7, 18 X, 7, 10 in X, 6 right and so gradually your steps are getting bigger and bigger right, and so you can see what's going to happen, it's gon na like start doing almost nothing right And it's going to then: suddenly the loss function is going to improve very quickly right, but then it's going to step even further again and then even further again, all right. Let's draw the rest of that line to be clear, all right and so suddenly it's then going to shoot off and get much worse right. So the idea, then, is to go back and say: okay at what point did we see like the best improvement? So here we've got our best improvement right, and so I would say: ok, let's use that learning rate right. So, in other words, if we were to plot the learning rate over time, it was increasing like so alright, and so what we then want to do is we want to plot the learning rate against the loss right. So when I say the loss, I basically mean like how accurate is the model. How close? In this case, the loss would be.

How far away is the predictive prediction from the from the goal? Okay, and so if we plotted the learning rate against the loss, we'd say like okay: initially, it didn't do very much right for small learning rates and then it suddenly improved a lot and then it suddenly got a lot worse. So that's the basic idea and so we'd be looking for the point where this graph is dropping quickly right, we're not looking for its minimum point, we're not saying like where was it the lowest, because that could actually be the point where it's just jumped too far. We want at what point: was it dropping the fastest? So if you go so, if you create your learn object in the same way that we did before we'll be learning more about this these details shortly. If you then call LR find method on that, you'll see that it'll start training a model like it did before, but it'll generally stop before it gets to 100 %. Okay, because if it notices that the loss is getting a lot worse, then it'll stop automatically that's what you can see here. It stopped at 84 %, and so then you can call one said that gets you the learning rate scheduler, that's the object, which actually does this learning rate finding and that object has a plot learning rate function, and so you can see here over by iteration. You can see the learning rate alright, so you can see each step. The learning rate is getting bigger and bigger. You can do it this way. We can see it's increasing exponentially.

Another way that Leslie Smith, the researcher suggests, is to do it linearly. So I'm actually currently researching with both of these approaches to see which works best. Recently, I've been mainly using exponential, but I'm starting to look more using Linea at the moment, and so if we then call shed but plot that does the plot that I just described down here. Learning

rate versus loss all right and so we're looking for the highest learning rate we can find where the loss is still improving, clearly well right, and so in this case I would say 10 to the negative 2x that 10 to the negative 1. It's not improving right 10 to the negative 3. It is also improving, but I'm trying to find the highest learning rate I can - or it's still clearly improving so I'd say 10 to the negative 2 okay. So you might have noticed that when we ran our model before we had 10 to the negative to 0.01. So that's why we picked that learning rate. So, there's really only one other number that we have to pick, and that was this number three and so that number three controlled. How many epochs did we run so an epoch means going through our entire data set of images and using each each time. We do a bunch of they're called mini batches. We grab like 64 images at a time and use them to try to improve the model a little bit using gradient descent right and using all of the images once is called one epoch, and so at the end of each epoch, we print out the accuracy and Validation and training loss at the end of the epoch, so the question of how many epochs should be run is kind of the one other question that you need to answer to run these three lines of code, and the answer really to me is like, as many As you like, what you might find happen is, if you run it for too long, the accuracy you'll start getting worse all right and we'll learn about that.

Why later it's something called overfitting right, so you can run it for a while run lots of epochs. Once you see it getting worse, you know how many epochs you can run and the other thing that might happen is if you've got like a really big model or a lot lots and lots of data. Maybe it takes so long, you don't have time, and so you just run enough epochs that fit into the time you have available. So the number of epochs you run. You know that's a pretty easy thing to set. So there are the only two numbers you're gon na have to see it, and so the goal this week will be to make sure that you can run not only these three lines of code on the data that I provided, but to run it on a set Of images that you either have on your computer or that you get from work well that you download from Google and like try to get a sense of like which kinds of images this is seem to work well, for which ones doesn't it work? Well, for what kind of learning rates do you need for different kinds of images? How many epochs do you need? How does the number of the learning rate change the accuracy you get and so forth? Like really experiment, and then you know try to get a sense of like what's inside this data object, you know what are the y-values look like? What are these places mean if you're not familiar with numpy, you know really practice a lot with numpy so that

15. 01:21:30

- **Why you need to use Numpy and Pandas libraries with Jupyter Notebook: hit 'TAB' for more info, or "Shift-TAB" once or twice or thrice (three times) to bring up the documentation for the code.**
- **Enter '?' before the function, or '??' to look at the code in details.**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

By the time you come back for the next lesson. You know we're going to be digging into a lot more detail, and so you'll really feel ready to do that now. One thing that's really important to be able to do. That is that you need to really know how to work with numpy, the faster, a library and so forth, and so I want to show you some tricks in Jupiter notebook to make that much easier. So one trick to be aware of is, if you can't quite remember how to spell something right. So if you're not quite sure what the method you want is you can always hit tab and you'll get a list of methods that start with that letter right and so that's a quick way to

find things. If you then can't remember what the arguments are to a method hit shift tab, all right, so hitting shift tab tells you the arguments to the method so shift. Tab is like one of the most helpful things I know so, let's take in P, X, P shift tab, and so now you might be wondering like okay. Well, what does this function do and how does it work? If you press shift tab twice, then it actually brings up the documentation, shows you what the parameters are and shows you what it returns and gives you examples. Okay, if you press it three times, then it actually pops up a whole little separate window with that information. Okay, so shift tab is super helpful. One way to grab that window straight away is, if you just put question mark at the start, then it just brings up that little documentation window now. The other thing to be aware of is increasingly during this course we're going to be looking at the actual source code of fastai itself and learning how it's built and why it's built.

That way, it's really helpful to look at source code. In order to you know, understand what you can do and how you can do it. So if you, for example, wanted to look at the source code for learned, I predict you can just put two question marks. Okay, and you can see it's popped up. The source code right, and so it's just a single line of code, you're very often find that fastai methods, like they they're, designed to never be more than about half a screen full of code and they're, often under six lines. So you can see this case. It's calling predicted with tags, so we could then get the source code for that in the same way, okay and then that's calling a function called predicted with tags, so we could get that documentation for that in the same way, and then so here yeah and then Finally, that's what it does it either rates through a data, loader gets the predictions and then passes them back and so forth. Okay, so question mark question. Mark is how to get source code, but the single question mark is how to get documentation and shift-tab is how to bring up parameters or press it more times to get the docs. So that's really helpful.

16. 01:24:40

- **Using the 'H' shortcut in Jupyter Notebook, to see the Keyboard Shortcuts.**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Another really helpful thing to know about is how to use Jupiter notebook well, and the button that you want to know is H. If you press H, it will bring up the keyboard shortcuts palette, and so now you can see exactly what Jupiter notebook can do and how to do it. I personally find all of these functions useful, so I generally tell students to try and learn four or five different keyboard shortcuts a day. Try them out see what they do see, how they work, and then you can try practicing in that session and one very important thing to remember when you're finished, with your work for the day go back to a paper space and click on that little button, which Stops and starts the machine. So after it's stopped you'll see it says, connection closed and you'll see it's off. If you leave it running, you'll be charged for it same thing with Crestle be sure to go to your cresol instance and stop it. You can't just turn your computer off or

17. 01:25:40

- **Don't forget to turn off your session in Crestle or Paperspace, or you end up being charged.**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Close the browser you actually have to stop an increase or or in paper space and don't forget to do that or you'll end up being charged until you finally do remember. Okay, so I think that's all the information that you need to get started. Please remember about the forum's okay, if you get stuck at any point check them out, but before you do make sure you read the information on course dot fast at AI for each lesson right, because that is going to tell you about like things that have changed. Okay, so if there's been some change, witch Cupid, a notebook provider we suggest using or how to set up paper space or anything like that and that'll all be on course, doc. Bastard, AI, okay, thanks very much for watching and look forward to seeing you in the next lesson.

Lesson 2: Convolutional neural networks

Outline

You will learn more about image classification, covering several core deep learning concepts that are necessary to get good performance: what a learning rate is and how to choose a good one, how to vary your learning rate over time, how to improve your model with data augmentation (including test-time augmentation). We also share practical tips (such as training on smaller images), an 8-step process to train a world-class image classifier, and more information on your hardware setup (including crestle, paperspace, and AWS as options).

Video Timelines and Transcript

1. 00:01:01

- **Lesson 1 review, image classifier,**
- **PATH structure for training, learning rate,**
- **what are the four columns of numbers in "A Jupyter Widget"**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Okay, so welcome back to deep learning lesson. 2. Last week we got to the point where we had successfully trained a pretty accurate image classifier, and so just to remind you about how we did that, can you guys see? Okay, I think the actually we can turn the phone once all right. Can you guys all see the screen? Okay, we can to adjust these ones. Can we some pictures all into darkness? But if that works then, is that okay, that's better, isn't it yeah? Can I dream the other two and maybe that one as well? Oh, but that one? Oh that's great! Sorry! I don't know your renders. Oh okay, great! That's better! Isn't it me so just to remind you the way that we built this image classifier was we used a small amount of code, basically three lines of code and these three lines of code pointed at a particular path which already had some data in it, and so The key thing for this to know how to train this model was that this path, which was data dogs, cats, and had to have a particular structure, which is that it had a train, folder and a valid folder and in each of those trained and valid folders. There was a cats folder in the dogs folder and if the cats on the docs folders was a bunch of images of cats and votes, but this is like a pretty standard. It's one of two main structures that are used to say here is the data that I want you to train an image model from so I know some of you during the week went away and tried different data sets where you had folders with different sets of Images and in credit, your own image, classifiers and generally, that seems to be working pretty well from what I can see on the forums so to make it clear at this point.

This is everything you need to get started. So if you create your own folders with different sets of images, you know a few hundred or a few thousand at each folder and run the same three lines of code. That will give you an image, classifier and you'll. Be able to see this third column tells you how accurate is so. We looked at some kind of simple visualizations to see

like what was it uncertain about what was it wrong about and so forth, and that's always a really good idea, and then we learned about the one key number you have to pick. So this is this number here is the one key number is 0.01, and this is called the learning rate, and so I wanted to go over this again and we'll learn about the theory behind what this is during the rest of the course in quite a lot Of detail, and for now I just wanted to talk about the practice. Yes, you know they cannot see you in the medium turnout. I just turned it around. You tell us about the other three numbers being bad. We did these three here we're going to talk about the other other ones shortly. So the main one we're going to look at for now is is the last column, which is the accuracy. The first column, as you can see, is the epoch number. So this tells us how many times has it been through the entire dataset trying to learn a better classifier and in the next two columns is, what's called the loss which we'll be learning about either later today or next week.

The first point is the loss on the training set. These are the images that we're looking at in order to try to make a better classifier, and the second is the loss of the validation set. These are the images that we're not looking at and we're training, but we're just sitting on the side to see how accurate we are so we'll learn about littering loss in accuracy later. Okay, so so we've got the epoch number. The training loss is the second column. The validation loss is the third column, and the accuracy is the fourth column you, okay, so the basic idea of the loading

2. 00:04:45

- **What is a Learning Rate (LR), LR Finder, mini-batch, 'learn.sched.plot_lr()' & 'learn.sched.plot()', ADAM optimizer intro**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Rate, so the basic idea of the learning rate is it's the thing that's going to decide: how quickly do we zoom do we kind of hone in on the solution, and so I find that a good way to think about this is to think about like well. What if we were trying to fit to a function that looks something like this right, we're trying to say: okay, where's, where abouts is the minimum point. This is basically what we do when we do deep learning. Is we try to find the minimum point of a function now? Our function happens to have millions or hundreds of millions of parameters, but it works the same basic way, and so when we look at it, you know we can immediately see that the lowest point is here. But how would you do that? If you are a computer algorithm and what we do is we we start out at some point at random. So you pick say here and we have a look and we say: okay, what's the what's the loss or the error at this point, and we say: what's the gradient, in other words, which way is up and which way is down, and it tells us that down Is going to be in that direction, and it also tells us how fast is it going down which is at this point is going down pretty quickly, and so then we take a step in the direction that's down and the distance we travel is going to be Proportional to the gradient sort of, unfortunately, how steep it is.

The idea is, if it's deeper, then we're probably further away. That's the general idea right and so specifically what we do is we take the gradient, which is how steep is it at this point and we multiply it by some number and that number is called the learning rate okay. So if we pick a number that is very small, then we're guaranteed that we're going to go a little bit closer and a little bit closer and a little bit closer each time right. But it's going to take us a very long time to eventually get to the bottom. If we dig a number, that's very big. We could actually step too

far could go in the right direction, but we could step all the way over to here right, as result of which we end up further away than we started, and we could oscillate and get worse and worse. So if you start training a neural net and you find that your accuracy or your loss is like spitting off into infinity almost certainly your learning rates too high, so in a sense learning rate too low is, is a better problem to have because you're going to Have to wait a long time, but wouldn't it be nice if there was a way to figure out like what's the best learning rate, something where you could kind of go quickly, go like Bom, Bom, Bom right, and so that's why we use this thing. Called a learning rate finder and what the learning rate finder does is it tries each each time it looks at another.

Remember the mini-batch, how many batches a few images that we look at each time so that we're using the parallel processing power of the GPU effectively. We look generally at around 64 128 images at a time for each mini batch, which is labeled here. As an iteration, we gradually increase the learning rate, multiplicatively increase the learning rate. We started really really tiny learning rates to make sure that we don't start at something too high and we gradually increase it. And so the idea is that eventually, the learning rate will be so big that the loss will start getting worse. And so what we're going to do, then, is we're a look at the plot of learning rate against loss right. So when the learning rates tiny it increases slowly, then it's that's. Where increase a bit faster and then eventually it starts not increasing as quickly and in fact it starts getting worse right so clearly here and make sure you're. You want to be familiar with this scientific notation, okay, so ten to the negative one is 0.1 10 to 50 or is 1 10 to the negative 2 is 0.001, and when we write this in Python, we'll generally write it like this, rather than writing 10 to The negative 1 or 10 to the negative 2 we'll just write 1, a neg 1 or 1 e neg. Okay. I mean the same thing: you're going to see that all the time and remember that equals 0.1. Oh point: O one: okay, so don't be confused by this text that it prints out here this this loss here is the the final loss at the very at the end of it's not of any interest right so ignore this.

This is only interesting when we're doing regular trading. That's not interesting for the learning rate finder. The thing that's interesting for the learning rate finder is this loan shed plot and specifically we're not looking for the point where it's the lowest back to the point where it's the lowest, it's actually not getting better anymore. So that's to higher learning rate. So I generally look to see like where is it the lowest and then I go back like one for magnitude, so one enoch two would be a pretty good choice: yeah, okay! So that's why you saw when we ran our fit here. We picked 0.01 right, which is one a neg two, so important point to make here is like this. This is the one key number that we've learnt to adjust and if you just adjust this number at nothing else, most of the time you're going to be able to get pretty good results, and this is like a very different message to what you would hear or See in any textbook or any video or any course, because up until now, there's been like dozens and dozens of these they're called hyper parameters. Dozens and dozens of hyper parameters to set and they've been thought of as highly sensitive and difficult to set so inside. The fastai library, we kind of do all that stuff for you as much as we can and during the course we're going to learn that there are some more. We can quake to get slightly better results.

But it's kind of like it's kind of in a funny situation here, because for those of you that haven't done anything learning before is kind of like oh, this is that's all there is to it. This is very easy and then, when you talk to people outside this class, they'll be like deep learning so difficult as someone to say it's a real art form, and so that's why there's this as is difference right and so that the truth is that the learning Rate really is the key thing to set, and this ability to use this to figure out how to set it. Well, though, the paper is now probably 18 months old.

Almost nobody knows about this paper. It was from a guy who's, not from a famous research labs. So most people kind of ignored it and, in fact even this particular technique was one subpart of a paper that was about something else. So again, this idea of like this is how you can set the learning rate. Really nobody outside this classroom. Just about knows about it, obviously the guy who wrote it Leslie Smith knows about it yeah. So it's a good thing to tell your colleagues about is like here is actually a great way to set the learning rate and there's even been papers. Caught like one of the famous papers is called no more pesky learning rates, which actually is a less effective technique than this one. But this idea that, like setting learning rates, is, is very difficult and, thirdly, is has been true for most of the kind of deep learning history. So here's the trick right go.

Look at this plot find kind of the lowest to go back about a multiple of ten and try that all right - and if that doesn't quite work, you can always try. You know going back another multiple ten, but this is always worked for me so far once why does this learning rate this method, work versus something else like momentum base or what's like the advantages, a disadvantage with just learning rate rate like technique? We're just feels. That's a great question, so we're going to learn during this course about a number of ways of improving gradient percent, like you mentioned momentum and atom and so forth. This is orthogonal. In fact. So one of the things the faster a library tries to do is figure out the right, gradient descent version and, in fact, behind the scenes, this is actually using something called atom. And so this technique is telling us. This is the best learning rate to use. Given what I thought other tweaks you're using in this case, the atom optimizer. So it's not that there's some compromise between this and some other approaches who sits on top of those approaches, and you still have to set the learning rate when you use with other approaches. So we're trying to find the best kind of optimizer to use for a problem that you still have to set the learning rate, and this is how we can do it. And, in fact, this idea of using this technique on top of more advanced optimizers.

Like Adam. Might haven't even seen mentioned in a paper before so I think this is like a I mean it's not a huge breakthrough, it seems obvious, but nobody else seems to tried it. So, as you can see, it was well when we use optimizers like Adam ditched, Harvick adaptive learning rate so, and he said this learning rate is Italy, initial learning rate because it changes during the people. So we're going to be learning about things like Adam the details about it later in the class, but the basic answer is no, even with even the Adam that there actually is a learning rate, it's just being it's being basically divided by the the gradient, the average Previous gradient and also the recent summer, Squared's of gradients, so there's still like a number called the learning rate there. There isn't a even these, so called dynamic learning rate methods still have unlearning rate. Okay, so the

3. 00:15:00

- **How to improve your model with more data,**
- **avoid overfitting, use different data augmentation 'aug_tfms='**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Most important thing that you can do to make your model better and is to give it more data. So the challenge that happens is that these models have hundreds of millions of parameters and, if you train them for a while, they start to do what's called overfitting. And so overfitting

means that they're going to start to see. Like the specific details of the images you're. Giving them, rather than the more general learning that can transfer across to the validation set, so the best thing we can do to avoid overfitting is to find more data now. Obviously, one way to do that would just be to collect more data from where you're getting it from or label more data, but a really easy way that we should always do is to use something called data augmentation. So they don't open tuition. Is one of these things that's key in many courses, it's not even mentioned at all, or if it is it's kind of like an advanced topic right at the end, but actually it's like the most important thing that you can do to make a better model. Okay, and so it's built into the faster you library, to make it very easy to do, and so we're going to look at the details of the code shortly. But the basic idea is that, as in our initial code, we had a line that said image classified data from parts and we passed in the path to our data and for transforms. We passed in basically the sizing, the architecture. We'll look at this in more detail.

Shortly, we just add one more parameter, which is what kind of data augmentation do you want to do and so to understand data augmentation? It's may be easiest to look at some pictures of data augmentation. So what I've done here again we'll look at the code in more detail later, but the basic idea is: oh, I've run I've built a data class like multiple times, I'm going to do it six times and each time I'm going to plot the same catch and You can see that what happens is that this cap here is further over to the left. This one here is further over to the right, and this one here is fit horizontally and so forth. So data augmentation different types of image, you're going to want different types of data, augmentation right. So, for example, if you were trying to recognize letters and digits, you wouldn't want to flip horizontally because, like it's actually has a different meaning, whereas on the other hand, if you're looking at photos of cats and dogs, you probably don't want to fit vertically, because cats Aren't generally upside down all right, where else, if you're looking at there's a current Kaggle competition, which is recognizing icebergs in satellite images, you probably do want to fit them upside down, because it's really matter which area around the iceberg or the satellite was right. So one of the examples of the transform sets we have is transforms sidon.

So, in other words, if you have photos that are like generally taken from the side, which generally means you want to be able to flip them horizontally, but not vertically, this is going to give you all the transforms you need for that, so it'll flip them sideways. Rotate them by small amounts, but not too much and slightly bury their contrast and brightness and slightly zoom in and out a little bit and move them around a little. So each time it's a slightly different fight, billionaires we're getting a

4. 00:18:30

- **More questions on using Learning Rate Finder**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Couple of questions from people about you, explaining, in the reason why you don't think the minimum of the loss curve yeah, but it's like the higher rate, so yeah and also could you people understand if this works for every CNN for CNN every minute. There's a running right: fine done yeah, exactly yeah, okay, great um! Could you put your hand up if there's a spare seat next to you? So there was a question about the learning rate finder about. Why do we use the learning rate? That's less than the lowest point, and so the reason why is to understand what's going on with this learning rate finder? So, let's go back to our picture here

like how do we figure out what learning rate to use right, and so what we're going to do is we're going to take steps and each time we're going to double the learning rate, so kind of double the amount By which we multiply the grander gradient, so, in other words, would go tiny step slightly, bigger, slightly bigger, slightly bigger, slightly bigger, slightly bigger, slightly bigger okay, and so the question is of the purpose of this is not to find the minimum. The purpose of this is to figure out what learning rate is allowing us to decrease quickly right. So the point at which the loss was lowest here is actually there right, but that learning rate actually looks like it's probably too high.

It's going to just jump like probably backwards and forwards, okay, so, instead what we do is we go back to the point where the learning rates quickly are giving us a quick increase in the loss. So here is so here is the actual learning rate increasing every single time. We look at a new mini batch, so mini-batch reiteration versus learning right and then here is learning rate versus loss. So here's that point at the bottom, where is now already too high? Okay and so here's the point where we go back a little bit and it's increasing nice and quickly we're going to learn about something called stochastic gradient descent with restarts shortly, where we're going to see like in a sense, you might want to go back to 1 Enoch 3, where it's actually even steeper still, and maybe we would actually find this book actually learn even quicker. You could try it, but we're going to see later why actually using a higher number is going to give us better generalization. So for now, let's put that aside, do you mean higher learning rate? When you say I know, I mean higher letting retina say higher yeah yeah I mean I am learning rate so as we increase the iterations from the learning rate finder, the learning rate is going up. This is iterations versus learning, ready. Okay, so as we do that as the learning rate increases - and we plot it here - the loss Goes Down, and here we get to the point where the learning rate is too high and at that point the most is nowgetting worse.

Because I asked the question because you were just indicating that you know, even though the minimum was at 10 to the minus 1, you were gon na. You suggest that we should choose 10 to the minus 2, but now you're saying I mean we should go back. The other way higher, so I didn't mean to say that I'm sorry if I said something backwards, I want to go back down to the lower learning rate. So possibly I said a higher when I meant higher into this lower OS. Do you know I'm learning right? Okay, thanks yep last class, is said that the local, all the local minima are the same, and this graph also shows the same. Is that is this something that was observed or is the logic theory behind it? That's not what this graph is showing. This graph is simply showing that there's a point where, if we increase the learning rate more, then it stops getting better than actually starts getting worse. The idea that all local minima are the same is a totally separate issue and it's actually something else, we'll see a picture of shortly. So let's come back to that Jeremy. Do we have to find the base learning rate every time we are going to run a poke third time, we're running on a poke and a pop? So how many times should I run this like? Let me write find my training. That's a great question unit um. I certainly run it once when I start later on in this class we're going to learn about unfreezing layers and after I unfreeze layers, I sometimes run it again.

If I do something to like change the thing, I'm training or change the way, I'm training it. You may want to run it again, basically, or you know, if you particularly if you've changed something about how you train, like unfreezing layers, which we're gon na soon learn about and you're finding the other training is unstable or too slow. Well again, you can run it again, there's never any harm in running it. It doesn't take very long. That's great question. Okay, so back to

5. 00:24:10

- **Back to Data Augmentation (DA),**
- **'tfms=' and 'precompute=True', visual examples of Layer detection and activation in pre-trained**
- **networks like ImageNet. Difference between your own computer or AWS, and Crestle.**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Data augmentation, so if we add to a when we run this little transforms from model function, we pass in orientation transforms. We can pass in the main to a transform side on or transforms top down later on, we'll learn about creating your own custom, transform lists as well, but for now because we're taking pictures from the side, cats and dogs will say, transform side on and now each Time we look at an image, it's going to be zoomed in or out a little bit moved around a little bit rotated a little bit, possibly flipped, okay, and so what this does is it's not exactly creating new data, but as far as the convolutional neural net Is concerned, it's a different way of looking at this thing, and it actually therefore allows it to learn how to recognize cats or dogs from somewhat different angles right. So when we do data orientation, we're basically trying to say based on our domain knowledge here here are different ways that we can mess with this image that we know still make it the same image you know and that we could expect that you might actually see That kind of image in the real world, so what we can do now is when we call this from parts function which we'll learn more about shortly. We can now pass in this set of transforms, which actually have these augmentations in now. So that's going to we're going to start from scratch here we do a fit and initially the augmentations actually don't do anything and the reason initially they don't do.

Anything is because we've got here, something that says: precompute equals true we're going to come back to these lots of times, but basically what this is doing is. Do you remember this picture? We saw where we learn each different layer has these activations that basically look for it or anything from the middle of flowers to eyeballs of birds or whatever right, and so literally, what happens is that the the later layers of this convolutional neural network have these things Called activations and activation literally it's a number. An activation is a number that says this feature like eyeball of bird is in this location, with this level of confidence with its probability right, and so we're going to see a lot of this later. But what we can do is we can say all right. Well, in this we've got a pre trained network. Remember and a pre trained network is one where it's already learned to recognize certain things. In this case, it's learnt to recognize the one and a half million images in the imagenet dataset, and so what we could do is we could take the the second last layer. So the one which is like got all of the information necessary to figure out what kind of thing a thing is and we can save those activations. So basically saving things saying you know, there's this level of eyeball nurse here in this level of dogs facing us here or in this level of fluffy, ear there and so forth, and so we save for every image these activations and that we call them the pre Computed activations, and so the idea is now that when we want to create a new classifier which can basically take advantage of these pre computed applications, we can just very quickly train when all the details there shortly, we can very quickly train a simple linear model based On those - and so that's what happens when we say pre-compute equals true, and that's why you may have noticed this week, the first time that you run a model, a new model.

It takes a minute or two. Where else you saw when I ran it, it took like five or ten seconds, took you a minute or two, and that's because it had to pre-compute these activations and just has to do that once if you're, using like your own computer or AWS, it just has To do it once ever, if you're using Crestle, it actually has to do it once every single time you rerun press all because press or uses are just for these pre computed activations. It uses a special that all had a scratch space that disappears each time. You restart your press, or instance, so other than the special case of cresol generally speak. He does have to run at once ever for a data set okay. So the issue with that is that since we pre computed for each image, you know how much does it have an EI here and how much does it have a lizard's eyeball there and so forth? That means that data augmentations don't work right. In other words, even though we're trying to show at a different version of the cat each time, we've pre computed the activations for a particular version of that cat. So in

6. 00:29:10

- **Why use 'learn.precompute=False' for Data Augmentation, impact on Accuracy / Train Loss / Validation Loss**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Order to use data augmentation, we just have to go and learn. Pre compute equals false okay, and then we can run a few more APIs right, and so you can see here that, as we run more a Potts, the accuracy isn't particularly getting better. That's the bad news. The good news is that you can see that the train loss practices like the way of measuring the error of this model. Although that's getting better the errors going down, the validation error isn't going down, and but we're not overfitting and overfitting would mean that the training loss is much lower than the validation loss and we're going to talk about that. A lot during this course. But the general idea here is, if you're doing much better job on the training set, then you are on the validation set. That means your models, not generalize, so we're not at that point, which is good, but we're not really improving. So we're going to have to figure out how to deal

7. 00:30:15

- **Why use 'cycle_len=1', learning rate annealing,**
- **cosine annealing, Stochastic Gradient Descent (SGD) with Restart approach, Ensemble; "Jeremy's superpower"**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

With that before we do, I want to show you one other cool trick. I've added here cycle length equals one, and this is another really interesting idea. Here's the basic idea cycle length equals one enables a recent fairly recent discovery and deep learning called stochastic gradient descent with restarts and the basic idea is this as you as you get closer and closer as you get closer and closer to the right spot right now. Getting closer and closer, I may want to start to decrease my learning rate right because, as I get closer, I'm kind of like oh I'm, pretty close down. So, let's, let's slow down my steps to try to get executive, the right spot right, and so, as we do more iterations, our learning rate perhaps should actually go down right because, as we go along we're getting closer and closer to where we want to be - and We want to like get

exactly to the right spot. Okay, so the idea of decreasing the learning rate, as you train, is called learning rate annealing and it's it's very, very common, very, very popular everybody uses it basically all the time. The most common kind of learning rate annealing is really horrendously hacky. It's basically that researchers like pick a learning rate that seems to work for a while and then when it stops learning well, they drop it down by about 10 times, and then they keep learning a bit more until it doesn't seem to be improving and they drop It down by another ten times, that's what most academic research papers and most people in industry do so this would be like stepwise annealing very manual.

Very annoying. A better approach is simply to pick some kind of functional form like a line. It turns out that a really good functional form is one half of the cosine curve right and the reason. Why is that for a while when you're, not very close, you kind of have a really high learning rate, and that is you do get close. You kind of quickly drop down and do a few iterations with a really low learning rate, and so this is called cosine annealing. So to those of you who haven't done trigonometry for a while, cosine basically looks something like this right. So we've picked one little half piece: okay, so we're going to use cosine annealing, but here's the thing when you're in a very high dimensional space right near we're only able to show three dimensions right, but in reality we've got hundreds of millions of dimensions. We've got lots of different, fairly flat points there. No, not the actual local minima, but they're fairly flat points, all of which are pretty good right, but they might differ in a really interesting way, which is that some of those flat points? Let me show you: let's imagine: we've got a surface that looks something like this right now. Imagine that, where you kind of random guest started here and our initial therefore kind of learning rate annealing schedule got us down to here now.

Indeed, that's a pretty nice low error right, but it probably doesn't generalize very well, which is to say if we use a different data set where things are just kind of slightly different in one of these directions. Suddenly is a terrible solution right where else over here is basically equally good in terms of loss right, but it rather suggests that, if you move, if you have slightly different data, sets that are slightly moved in different directions, it's still going to be good right. So, in other words, we would expect this solution here is probably going to generalize better than this by key one. So here's what we do is we've got like a bunch of different low bits. Right then, our standard loading rate, annealing approach will start of go down here or downhill downhill, downhill downhill to one spot right, but what we could do instead is use a learning rate schedule. That looks like this, which is to say, we do a cosign annealing and then suddenly jump up again into a cosign, annealing and then jump up again and so each time we jump up. It means that if they're going to spiky bit and then we subtly increase the learning rate - and it jumps now all the way over to here and so then we kind of learning right in your learning right near death down to here, and then we jump up Again to a high learning rate: oh and it stays here right, so in other words, each time we jump up the learning rate.

That means that if it's in a nasty spiky part of the surface, it's going to hop out of the spiky part, and hopefully, if we do that enough times, it'll eventually find a nice smooth Bowl. Could you get the same effect by running multiple iterations through the different ground of my starting point, so that eventually, you explore all possible minimize yeah. So, in fact, that that's a great question and before this approach, which is called stochastic gradient, descent with restarts was, was created. That's exactly what people used to do. They used to create these things called ensembles, where they would basically relearn a whole new model ten times in the hope that one of them's like, but it ended up being better, and so the cool thing about this decosta gradient descent with restarts is that the model Once we're in a reasonably

good spot, each time we jump up the learning rate, it doesn't restart, it actually hangs out in this nice part part of the space and then keeps getting better. So, interestingly, it turns out that this approach, where we do this a bunch of separate cosine annealing steps, we end up with a better result as then, if we just randomly try it a few different starting points. So it's a super neat trick and it's a fairly recent development, but and again almost nobody's heard of it.

But I found like it's now, like my superpower like using this, along with the learning rate finder like I, can get better results than nearly anybody like in a casual competition. You know in the first week or two I can like jump in. It's been an arrow to and back I've got a fantastically good result, and so this is why I didn't pick the point where it's got the steepest slope. I actually trying to pick something kind of aggressively high. It's still getting down, but maybe like getting to the point where it's nearly too high, not because I want to make sure, because that's because, when we do this, stochastic gradient descent with restarts this ten to the negative two represents the a highest number that it uses. So it goes up to ten to the negative two and then goes down and then up to ten negative two and down. So if I use to lower learning rate, it's not going to jump to a different part of the function. So I have a few questions, but the first one is: how many times do you change your learning rate? You want to work, we don't change the learning rate, all three, how many times? Okay. So, in terms of this part here, where it's going down, we change the learning rate, every single mini all right and then the number of times we reset it is set by the cycle, length, parameter and so 1 means reset it up to every epoch. So if I had to there, it would reset it up to every to epochs and, interestingly this, this point that when we do the learning rate and kneeling that we actually change it, every single batch it turns out to be really critical to making this work, and It again is very different to what nearly everybody in industry in academia has done before.

What do you get a chance? Could you explain recompute? It was true because it's still yeah we're going to come back to that multiple times in this course. So the way this course has been a work is we're going to like do a really high-level version of each thing and then we're going to like come back to it in two or three lessons and then come back to it. At the end of the course and each time we're going to see like more of the math more of the code and get a deeper view, okay - and we can talk about it - also in the forums during the week. Our main goal is to generalize and we don't want to get those like narrow, demas yeah. That's a it's a very short summary. This method. Are we keeping track off to minimize and averaging them? Ah, that's that's another level of sophistication and indeed you can see there's something here called snapshot ensemble, so we're not doing it in the code right now. But yes, if you wanted to make us generalize even better, you can save the weights here and here and here and then take the average ishes. But for now we're just going to pick the last one. If you want to skip ahead, if you want to skip ahead, there's a parameter called cycle, safe name, which you can add as well as cycle them, and that will save a set of weights at the end of every learning rate cycle. And then you can ensemble them.

Ok, so we've

8. 00:40:35

- **Save your model weights with ‘learn.save()’ & ‘learn.load()’, the folders ‘tmp’ & ‘models’**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Got a pretty decent model here. Ninety nine point: three percent accuracy and we've gone through of you know a few steps that is taken. You know a minute or two to run, and so from time to time I tend to save my weight. So if you go, learn, dot, save and then pass in a file name, it's going to go ahead and save that for you later on. If you go, learn load you'll be straight back to where you came from okay. So it's a good idea to do that from time to time. This is a good time to mention what happens when you do this when you go, learn dot save when you create precomputed activations. Another thing we learn about soon when you create resized images. These are all creating various temporary files, okay, and so what happens is if we go to data and we go to dogs cats. This is my data. Folder and you'll see there's a folder here called TMP or, and so this is automatically created and all of my pre computed activations end up in here. I mention this because, if, if things are, if you're getting weird errors, that might be because you've got some, Oh pre computed activations like we're only half completed or are in some way incompatible with what you're doing so. You can always go ahead and just delete this TMP, this temporary directory and see if that causes your error, to go away. This is the faster I equivalent of turning it off and then on again.

You'll also see there's a directory called models, and that's where all of these, when you say dot, save with a model. That's where that's going to go. Actually it reminds me when the stochastic gradient descent with restarts paper came out. I saw a tweet that was, somebody was like, Oh to make your deep learning work, better turn it off and then on again question it. So, if I want to see, I want to retrain my model fuselage again. Do I just do everything the 10 folder? If you want, if you want to train your model,

9. 00:42:45

- **Question on training a model "from scratch"**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

From scratch, there's generally no reason to delete the pre computed activations, because the pre computed activations are without any training. That's what the pre trained model created with the with the weights that you downloaded off the internet, the only yeah I mean. The only reason you want to delete the pre computed activations is that there was some error caused by like half creating them and crashing or some something like that, as you change the size of your import change, different architectures and so forth, they all create different sets Of activations with different file names, so you don't generally you shouldn't have to worry about it. If you want to start training again from scratch, all you have to do is create a new learn object. So, each time you go like conch learner, dot, pre-trained. That creates a new object with with new sets.

10. 00:43:45

- **Fine-tuning and differential learning rate,**
- **'learn.unfreeze()', 'lr=np.array()', 'learn.fit(lr, 3, cycle_len=1, cycle_mult=2)'**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Of weights fever train from okay, so before our break, we'll finish off by talking about about

fine-tuning and differential learning rates, and so so far everything we've done has not changed any of these free trained filters right. So we've used a pre trained model. That already knows how to find at the early stages, edges, ingredients and then corners and curves, and then repeating patterns and bits of text and eventually eyeballs right. We have not retrained any of those activations any of those features. Well, specifically, any of those weights in the convolutional kernels, all we've done is: we've learnt some new layers that we've added. On top of these things, we've learned how to mix and match these pre-trained features. Now, obviously, it may turn out that your pictures have you know different kinds of eyeballs or faces or if you're, using different kinds of images like satellite images, totally different kinds of features altogether right. So if you're like training to recognize icebergs, you'll, probably want to go all the way back and learn, you know all the way back to kind of different combinations of these simple gradients and edges. In our cases, dogs, vs. cats, we're going to have some minor differences, but we still may find it's helpful to slightly tune some of these later layers, as well. So to tell the learner that we now want to start actually changing the convolutional filters themselves.

We simply say unfreeze okay, so a frozen layer is a layer which is not trained is not updated. Okay, so unfreeze unfreezes all of the layers. Now, when you think about it, it's pretty obvious that layer, one right which is like a diagonal edge or a gradient. Probably doesn't need to change by much if at all, right from the 1 and a half million images on image net, it probably already is figured out pretty well how to find like edges of gradients. It probably already knows also like which kind of corners to look for and how to find which kinds of curves and so forth, so, in other words, these early layers probably need little. If any learning, where else these later ones are much more likely to need more learning - and this is universally true, regardless of whether you're looking for satellite images of rainforests or icebergs or whether you're looking for cats versus dogs right, so what we do is we create An array of learning rates where we say okay: these are the learning rates to use for our additional layers that we've added on top. These are the learning rates to use in the middle few layers, and these are the learning rates to use for the first few layers. So these are the ones for the layers that represent like very basic, geometric features. These are the ones that are used to for the more complex kind of sophisticated convolutional features, and these are the ones that are used for the features that we've added and went from stretch right.

So you can create a array of learning rates and then, when we called up fit and pass an array of learning rates, it's now going to use those different learning rates for different parts of the model. This is not something that we've like invented, but I'd also say it's like it's, so not that common, that it doesn't even have a name as far as I know so we're going to call it differential learning rates if it actually has a name or indeed, if Somebody's actually written a paper specifically talking about it. I don't know, there's a great researcher, called Jason, your Sinskey, who who did write a paper about the kind of the idea that you might want different learning rates and showing why? But I don't think any other libraries support it and yeah. I don't know of a name for it. Having said that, though, this ability to like unfreeze and then use these differential learning rates, I found it's like the secret to taking a pretty good model and putting it into an awesome model, so just to clarify, so you have three numbers there: okay, three hyper parameters: The first one is the photo late model, so the mall that are late layers, the so with the it's. Your answer is many, many right and they're kind of in groups and we're going to learn about the architecture. This is called a ResNet for residual network. It kind of has ResNet blocks, and so what we're doing is we're grouping the blocks into three groups, and so this one is actually.

This first number is for the earliest layers, yeah they're ones closest to the pixels that represent like corners and edges and gradients. But why why do you well? I thought those layers are frozen at first, so yeah right, so we just said unfreeze the streets. Also, we so yeah, I'm freezing them because you have kind of partially trained, although lately we've trained, we've trained our added layers. Yes, now you are, we training the Oh step exactly obviously so it waits and the learning rate is particularly small for the early layers. That's right because you just find a want to find food yeah yeah. We probably don't want to change them at all, but you know if it does need to. Then it can thanks no problem so using the differential in rates a little different from like grid search. There's, no similarity to grid search, so grid search is where we're trying to find the best hyper parameter for something. So, for example, you could kind of think of the learning rate finder as a really sophisticated grid search, which is like trying lots and lots of learning rates to find which one is best, but this has nothing to do with that. This is actually for the entire training. From now on, it's actually going to use a different learning rate for each layer, and so I was wondering so you give a pre train model. Then you have to use the same input dimensions right because I was thinking.

Okay, let's say you have this big: they use like big machines to train these things and you want to take advantage of it. How would you go about you know? You have like images that are like bigger than the ones that they used or we're going to be talking about sizes later. But the short answer is that with this library and the modern architectures were using, we can use any size we like so did I mean do we need? Can we at least just a specific layer? We can we're not doing it yet. But if you wanted to, you can learn dot, freeze, underscore two and pass into layer number much to my surprise, or at least initial my surprise. It turns out. I almost never need to do that. I almost never find it helpful and I think it's because we're using differential learning rates, the the optimizer can kind of learn just as much as it needs to so yeah. It's a little data like very little data yeah. It still doesn't seem to help the one place I have found it helpful is, if I'm using like a really big memory, intensive model and I'm like running out of GPU crazy having the the less layers you unfreeze, the less memory it takes and the less time It takes so there's that kind of practical aspect. So to me she'll say I asked the question right. Can I just like unfreezes specific layer? No, you. You can only unfreeze layers from layer n onwards. You could probably delve inside the library in phase one phase, one layer, but I don't know why you would okay.

So I'm really excited to be showing you guys this stuff, because it's like it's something. We've been kind of researching all year, it's figuring out how to train state of the art models and we've kind of found these like tiny number of tricks, and so once we do that, we now go learn about fit right and you can see. Look at this. We get right up to that 99.5 % accuracy, which is crazy. There's one other trick. You might see here that, as well as using stochastic gradient descent with restarts a cycle length equals one. We've done three cycles so earlier on. I lied to you. I said this is this: is the number of epochs it's actually the number of cyclists right. So if you said cycle length equals two, it would do three cycles of each of two epochs or do six, because so here I've said two three cycles. Yet somehow it's done seven epochs and the reason why is I've got one last trick to show you which is cycle mult equals two and to tell you what that does I'm simply going to draw you a picture you? The picture, if I go learn Dutch share top plot learning rate there. It is now you can see what cycle mode equals to is doing. Okay, it's it's doubling the length of the cycle after each cycle, and so in the paper that introduced this stochastic gradient descent. With restarts the researcher kind of said, hey, this is something that seems to sometimes work pretty well, and I've certainly found that often to be the case.

So, basically, intuitively speaking, if your cycle length is too short right, then it's kind of starts going down to find a good spot and then it pops out - and it goes to a try and photographs button pops out it never actually gets to find a good spot Right so earlier on, you want it to do that, because it's trying to find the bit that's like smoother but then later on, you want it to fight, do more, exploring and then more exploring right. So that's why this cycle mole equals two thing often seems to be a pretty good approach right, so suddenly we're introducing more and more hyper parameters. Having told you that there aren't that many, but the reason is that, like you, can really get away with just taking a good learning rate, but then adding these extra tweaks really helps get that extra level up without any effort right and so in practice. I find this kind of three cycles, starting at 1 mode, equals 2 works very, very often to get a pretty decent model if it does doesn't, then often I'll just do 3 cycles of length 2 with no molt, okay, there's kind of like two things that seem To work a lot and there's not too much fiddling, I find necessary and, as I say, even even if you just if you use this line every time, I'd be surprised if you didn't get a reasonable result. So a question here: why does a smoother services correlate to more

11. 00:55:30

- **Advanced questions: “why do smoother services correlate to more generalized networks ?” and more.**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Generalize networks, so it's kind of this some this intuitive explanation. I try to just kill the whole thing I try to give back here, which is that if you've got something spiky right and so what this, what this x-axis is showing is like how? How good is this at recognizing dogs versus cats, as you change, this particular parameter right, and so so something to be generalizable. That means that we wanted to work when we give it when we give it a slightly different data set, and so a slightly different data set may have a slightly different relationship between this parameter and how caddy versus dog it is. It may instead look a little bit like this right, so in other words, if we end up at this point right, then it's not going to do a good job on this slightly different data set. For else. If we end up on this point, it's still going to do a good job on this data set okay. So that's what psychomotor equals doing? Okay, so we've got one last thing before we going to take a break, which is we're now going to take this model, which has 99.5 percent accuracy and we're going to try and make it better still. And what we're going to do is we're not actually going to change the model at all right, but instead we're going to look back at the original virtual visualization. We did where we looked at some of our incorrect pictures.

Now, what I've done is I've printed out? The whole of these incorrect pictures, but the key thing to realize is that particularly in fact, when we do the the validation set, all of our inputs to our model, all the time have to be square right, and the reason for that is, it's kind of a Minor technical detail, but basically the GPU doesn't go very quickly if you have like different dimensions for different images, because it needs seems to be consistent so that every part of the GPU can do the same thing, and I think this is probably fixable, but it now. That's the state of the technology we have so our validation set when we actually say for this particular thing is it's a dog what we actually do to make it square, as we just pick out the square in the middle right, so we would take off its Two edges, and so we take the whole height and then as much of the middle as we can, and so you can see in this case we wouldn't actually see this dog's

head right, so I think the reason this was actually not correctly classified was because the Validation set only got to see the body and the body doesn't look particularly doglike or cat-like. It's not at all, punctual what it is. So what we're going to do when we calculate the predictions for our validation set, is we're going to use something called test time.

Augmentation and what this means is that every time we decide is this cat or a dog not in the training but after we've trained the model is we're going to actually take four random data augmentations and remember the data augmentations move around and zoom in and out And flip okay, so we're going to take four of them at random and we're going to take the original and augmented sent a cropped image and we're going to do a prediction for all of those and then we're going to take the average of those predictions. So I'm going to say is this a cat? Is this a cat? Is this a cat? Is this a cat, but, and so hopefully in one of those random ones? We actually make sure that the face is there zoomed in by a similar amount to other dogs faces at sea, and it's rotated by the amount that it expects to see it and so forth, and so do that. All we have to do is just call tt8. Tta stands for Test time, augmentation this term of like what a? What do we call up when we're making predictions from up from a model we've trained? Sometimes it's called inference time. Sometimes it's called test time. Everybody since have a different name, so TTA, and so when we do that we go learn, TTA check the accuracy and lo and behold we're now at ninety nine point, six five percent, which is kind of crazy where's, our green box, but for every park we are Only showing one type of augmentation or for particular image right so when we are training back here, we're not doing any TTA right, so TTA is not like you could and sometimes like I've written libraries where, after a cheap up, I run TTA to see how.

Well, it's going, but that's not what's happening here. I trained the whole thing with training time organization which doesn't have a special name, because that's what we mean when we say data augmentation. We need training, time augmentation. So here every time we showed a picture, we were randomly changing it a little bit so each epoch, each of these seven epochs. It was seen slightly different versions of the picture. Having done that, we now have a fully trained model. We then said: okay, let's look at the validation set, so TTA by default, uses the validation set and said: okay, what are your predictions of? Which ones are cats and which ones are dogs, and it did 4 predictions with different random orientations, plus one on the organ. Under Augmented version, average them all together and that's what we got and that's what we can't clear the accurate. So is there a high probability of having sample in TTA that was not shown in doing trained yeah. Actually, every data, augmented for image is, is unique because the rotation could be like point. Zero. Three four degrees and zoom could be 1.0 one sixty five. So every time it's slightly different, no problem was behind you. What's your might not use white padding or something like that, just one of your white padding, like just you know, put like a white water around? Oh padding's. Not yes! So, like there's lots of different types of a better orientation you can do, and so one of the things you can do is to add a border around it.

Basically adding a border around it in my experiments doesn't doesn't help it doesn't make it any less cat-like. It's not the convolutional neural network doesn't seem to find it very interesting. Basically, something that I do do we'll see later is. I do something called reflection, padding, which is where I add some borders that are the outside just reflected. It's a way to kind of make some bigger images works well with satellite imagery in particular, but yeah in general. I don't do I have a lot of padding. Instead, I do a bit of zooming, it's kind of follow-up to that last one, but rather than cropping just at white space, because when you crop you lose the dog's face. But if you added white space you wouldn't yeah. So that's that's

where the kind of the reflection padding or the zooming or whatever can help. So there are ways in the faster. You know, library, when you do custom transforms of of making that happen, I find that it kind of depends on the image size. You know, but, generally speaking, it seems that using TTA plus data augmentation, the best thing to do is to try to use this larger image as possible, and so, if you kind of crop the thing down and put white borders on top and bottom, it's now quite A lot smaller and so to make it as big as it was before.

You now have to use more GPU, and if you're going to use more that multi figure, you could have zoomed in and used a bigger image. So in my playing around that doesn't seem to be generally as successful. There is a little interest on the topic of how do the domain tation in older than images. Indeed at least not images, um yeah, um, no one seems to know I actually um. I asked some of my friends in the natural language processing community about this we'll get to natural language processing. In a couple of lessons you know it seems like it'd, be really helpful. There's been a few example. I carry very few number examples of people where papers would like try replacing synonyms, for instance, but on the whole and understanding of like appropriate data. Augmentation for non image domains is under-researched in under under developed. The question was: could couldn't we just use a sliding window to generate on the images so in that dog? Thank you. Couldn't we generate three parts of it? Wouldn't that be better yeah PTI you mean just just in general, when you're creating your so training time. I would say no that wouldn't be better, because we're not gon na get as much variation. You know we want to have it like, like one degree off five, you know five degrees off ten pixels up like lots of slightly different versions, and so, if you just have three standard ways, then you're not giving it as many different ways of looking at the Data for testing augmentation having fixed cropped locations - I think, probably, would be better - and I just haven't gotten around to writing that. Yet I have a version in an old library.

I think having fixed cropped locations plus random contrast, brightness rotation changes might be better. The reason I've got around to it yet is because in my testing it didn't seem to help him practice very much and it made the code a lot more complicated. So you

12. 01:05:30

- **“Is the Fast.ai library used in this course, on top of PyTorch, open-source ?” and why Fast.ai switched from Keras+TensorFlow to PyTorch, creating a high-level library on top.**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Know it's kind of it's an interesting question. I just wanted all of this last AI api's that you are using. Is it yeah? That's a great question, so the faster you go, libraries open source and let's talk about it, a bit more generally, because you know it's like the fact that the fact that we're using this library is kind of interesting and unusual and it sits on top of something Called a torch right, so pytorch is a fairly recent development and it's kind of I've noticed all the researchers that I respect pretty much are now using high torch. I found in part two of last year's course that a lot of the cutting-edge stuff I wanted to teach I couldn't do it in chaos and tensorflow, which is what we used to teach with, and so I had to switch the course to pay torch halfway through Part two: the problem was that pytorch isn't very easy to use. You have to write your own training loop from scratch. I basically write everything from scratch or the stuff you see inside the class. They are library we would have had to written it. You know to

learn, and so it really makes it very hard to learn deep learning when you have to write hundreds of lines of code to do anything. So so we decided to create a library on top of pytorch, because we, you know this. Our mission is to teach world class big morning, so we wanted to show you like here's how you can be the best in the world at doing it, and we found that a lot of the world class stuff we needed to show really needed pytorch or At least with pytorch, it was far easier and but then PI thought itself just wasn't suitable as a first thing to teach with for new for new deep learning practitioners.

So we built this library on up of pytorch, initially heavily influenced by chaos, which is what we taught last year and but then we realized, we could actually make things much much much easier than care us. So in care us. If you look back at last year's course notes, you'll find that all of the code is two to three times longer and there's lots more opportunities for stakes, because there's just a lot of things you have to get right, so we ended up kind of building this. This this library, in order to make it easier to get into deep learning, but also easier to get state-of-the-art results and then, over the last year, as we started developing on top of that, we started discovering that by using this library, it made us so much more Productive that we actually started kind of developing you, state-of-the-art results and new methods ourselves, and we started realizing that there's a whole bunch of like papers that have kind of been ignored or lost, which, when you use them, it could like automate or semi-automated stuff. Like learning read, finder, that's not in any other library, so so it kind of got to the point where now not only is kind of fastai lets us do things easier, much easier than any other approach, but at the same time it actually has a lot More kind of sophisticated stuff behind the scenes than anything else, so so it's kind of an interesting mix so yeah.

So we've released this library like at this stage, it's like very early version and so through this course. By the end of this course, I hope, as a group, you know we will all a lot of people are already helping, have developed it into something. That's you know really pretty stable and rock-solid and yeah. Anybody can then can use it to build your own models under an open-source license. As you can see it's available on github behind the scenes, it's it's creating play, torch models and so apply torch models can then be exported into various different formats. Having said that, like a lot of folks like issue, if you want to do something on a mobile phone, for example, you're probably going to need to use tensorflow and so later on. In this course we're going to show like how some of the things that we're doing in the past AI library, you can do in chaos and cancel flow. So you can going to get a sense of what the different libraries look like and, generally speaking, the simple stuff is like it'll. Take you a small number of days to learn to do it and care us in tensorflow versus fastai and high torch and the more complex stuff. Often this won't be possible so that, like, if you needed to be intensive flow, you're, just kind of simplify it off in a little bit. But you know, I think the more important thing to realize is every year the kind of the libraries that are available and which ones are the best totally changes.

So, like the main thing, I hope that you get out of this course is an understanding of the concepts like here's, how you find a learning rate. Here's why differential learning rates are important? Is they do learn where the kneeling? You know here's what stochastic gradient a second's restarts does so on and so forth, because you know by the time we do this course again. Next year you know the library situations and the difference the king. That's a question of that. I was wondering if you've had an opinion on pyro, which is ubers new release. I haven't looked at it, no I'm very interested in probabilistic programming and it's really cool that's built on top of paper. So one of the things we'll learn about in this course is we'll see that pytorch is much more than just a deep learning library. It actually lets us write arbitrary

gpu-accelerated algorithms from scratch, which we're actually going to do and pyro is a great example of what people are now doing with might watch outside of the deep level. Great. Ok, let's take a eight-minute break and we'll come back at 7:55. So ninety nine point: six five

PAUSE

13. 01:11:45

- **Classification matrix 'plot_confusion_matrix()'**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Percent accuracy, what does that mean? So in classification, when we do classification and machine learning, the really simple way to look at the result of a classification is, what's called the confusion matrix. This is not just deep learning, but in any kind of classifier machine learning where we say okay, what was the actual truth? There were thousand cats and a thousand dogs out of the thousand actual cats. How many did we predict were cats? This is obviously in the validation step. This is the images that we didn't use to train with. It turns out. There were nine hundred ninety-eight cats that we actually predicted as cats and two that we got wrong. Okay and then for dogs. There were nine hundred ninety-five that we predicted were dogs and then five that we got wrong and so often these confusion matrices can be helpful, particularly if you've got like four or five classes. You're trying to predict to see like which group you having the most trouble with and you can see it uses color coding to tell you you know to highlight the large. The large bits you've got to hope that the diagonal is the highlighted section. So now that we've retrained the model, it can be quite helpful now that's better to actually look back and see like okay, which ones in particular were incorrect, and we can see here there were actually only two incorrect cats.

It prints out four by default, so you can actually see these two actually less than 0.5, so they weren't they weren't wrong. Okay, so it's actually. These two were wrong. Cats and this one isn't obviously a cat at all. This one is, but it looks like it's got a lot of weird artifacts and you can't see its eyeballs at all so and then here are the how many dogs, where they're all wrong there were five wrong dogs. Here are four of them. That's not! Obviously, a dog that looks like a mistake that looks like a mistake that one, I guess doesn't

14. 01:13:45

- **Easy 8-steps to train a world-class image classifier**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Have enough information that I guess it's a mistake, so so we've done a pretty good job here of creating a good classifier. I would based on entering a lot of capital competitions and comparing results. I've done two various research papers. I can tell you it's a state of the art classifier. It's it's right up there with the best in the world, we're going to make it a little bit better in a moment, but here in the basic steps right. So if you want to create a world-class image classifier, the steps that we just went through was that we started our week's term data augmentation on by saying oil transforms, equals and you either say sidon or top-down,

depending on what you're doing start with pre compute equals. True find a decent learning, eight. We then train just like it one or two epochs, which that takes a few seconds as we got through compute equals true, then we turn off pre compute, which allows us to use data augmentation to do another two or three epochs. Generally, with cycle length equals one, then I unfreeze all them, as I then set the earlier layers to be like either somewhere between a 3 times 2 10 times mobile learning rate in the previous. So in this case I did 10 times right. So it's like this was my learning rate that I found from the learning rate finer than I went 10 times smaller and then 10 times smaller as a rule of thumb like knowing that you're, starting with a pre trained imagenet model.

If you know, if you can see that the things that you're now trying to classify a pretty similar the kinds of things in imagenet ie pictures of normal objects in normal environments, you probably want about a 10x difference, because you want those earlier layers, like you, think That the earlier layers are probably very good already, but also, if you're doing something like satellite imagery or medical imaging, which is not at all like image net, then you probably want to be training those earlier layers, a lot more, so you might have like. Oh just a 3/8 difference all right, so that's like one change that I make is to try to make it out of 10x or 3x. Yes, so then, after unfreezing, you can now call LR find again, but at Nike didn't in this case, but like once you've unfrozen all the layers, you've turned on differential learning rates. You can then call a lot of fine again right, and so you can then check like oh does it still look like the same point I had last time is about right. Something to note is that if you call LR find having set differential learning rates, the thing that's actually going to print out is the learning rate of the last layers right, because you've got three different learning rates, so it's actually showing you the last layer. So then yeah then I trained the full

15. 01:16:30

- **New demo with Dog_Breeds_Identification competition on Kaggle, download/import data from Kaggle with 'kaggle-cli', using CSV files with Pandas. 'pd.read_csv()', 'df.pivot_table()', 'val_idxs = get_cv_idxs()'**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Network with cycle more equals two and it'll either it starts with the fitting or I run out of time right so like. Let me show you all right, so let's do this again, a totally different data set. So this morning I noticed that some of you on the forums were playing around with this playground. Kegel competition very similar, called dog breed identification, so the dog breed identification cat will challenge is one where you don't actually have to decide which ones are cats and which ones the dogs, they're all dogs. But you have to decide what kind of dog it is, but there are 120 different breeds of dogs. Okay, so you know. Obviously this could be like different types of cells and pathology, slides. It could be different kinds of cancers in CT scans. It could be different kinds of icebergs and satellite images, whatever right as long as you've got some kind of labeled images. So I want to show you what I did this morning, so it took me about an hour basically to go in to end from something I'd. Never seen before so I downloaded the data from kaggle and I'll show you how to do that shortly. But the short answer is: there's something called cable CLI, which is a github project you can search for and if you read the docs to basically run cagey download, provide the competition name and it will grab all the data for you to your crystal or Amazon or Whatever instance I put in my data folder and I then went LS, and I saw that it's a little bit different to our previous data set.

It's not that there's a train folder, which has a separate folder for each kind of dog, but instead of tonette there was a CSV file and the CSV file. I read it in with pandas, so pandas is the thing we use in python to do structured data analysis like csv files, so he picked pandas. We called pd, that's pretty much universal PDR. Htsb reads in the csv file. We can then take a look at it and you can see that basically, it had like some kind of identifier and then the debris right, so this is like a different way. This is the second main way that people kind of give you image labels. One is to put different images into different folders. The second is generally to give you as some kind of file like a CSV file, to tell you here's the image name and here's the label. Okay. So what I then did was I used pandas again to create a pivot table which basically groups it up just to see how many of each breed there were, and I sorted them, and so I saw okay they've got like about a hundred some of the more Common breeds and some of the less common breeds they've got like 60 or so okay. Altogether, there are 120 rows and I've been 120 different breeds represented. Okay, so I'm going to go through the steps right so enable data augmentation so to enable data augmentation. When we call this transforms from model, you just pass in and all transformers. In this case, I chose side on again. These are pictures of dots and stuff, so this side on photos.

I we're talking about maqsuum as more detail later, but maximum basically says when you do the data augmentation. We like zoom into it by up to one point one times: okay, so but randomly between one, the original image size and one point one points. So it's not always cropping out in the middle or an edge, but it could be cropping out a smaller part. Okay. So having done that, the key step now is to graphically going from paths. So previously we went from paths, and that tells it that the the names of the folders are the names of the labels. We go from CSV and we pass in the CSV file that contains the letters so we're passing in the path that contains all of the data, the name of the folder that contains the training data, the CSV that contains the labels. We need to also tell it where the test set is, if you want to submit to cattle later talk more about that next week. Now this time the previous data set, we had had actually separated a validation set out into a separate folder right, but in this case you'll see that there is not a separate folder called validation right. So we want to be able to track how good our performance is low, so we're going to have to separate some of the images out to put it into a validation set okay, so I do that at random and so up here you can see how it Basically opened up the CSV file, turned it into a list of rows and then taken the length of that minus one, because there's a header at the top right.

And so that's the number of rows in the CSV file, which must be the number of images that we have and then this is a fastai thing get cross-validation indexes now we'll talk about cross-validation later. But basically, if you call this and pass in a number, it's going to return to you by default a random twenty percent of the rows who uses your validation set and you can pass in parameters to get different amounts right. So this is now going to grab twenty percent of the data and say all right. This is the this is the indexes the numbers of the files which we're going to use as a validation set. Okay. So now that we've got that, in fact, let's kind of run this, so you can see what that looks like so well, indexes is just a big bunch of numbers, okay and so an is 10,000 right, and so we have about twenty percent of those is going To be in a validation set, so when we call from CSV we can pass in a parameter which is talent which indexes to treat us a validation set, and so that's passed in those indexes. One thing that's a little bit tricky here is that the file names actually have. I checked, they actually have a dot jpg on the end, and these obviously don't have a dot jpg. So you can pass in when you call from CSV you can pass in a suffix. It says that the labels don't actually contain the full file names. You need to add this to them.

Okay, so that's basically all I need to do to set up my data and, as a lot of Europe noticed during the week inside that data object, you can actually get access to the data set like what the training data set by same train. Yes and inside train des is a whole bunch of things, including the file names. Okay, so train desktop file. Names contains all of the file names of everything in the training set, and so here's like one file name, okay, so here's an example of one file name. So I can now go ahead and open that file and take a look at it. That's the next thing I did was to try and understand what my file my dataset looks like, and it found an adorable puppy, so that was very nice, so feeling good about this. I also want to know like how big of these files right, like how big are the images, because that's a key issue, if they're huge and then I have to think really carefully about how to deal with huge images, that's really challenging if they're tiny. Well, that's also challenging. Most of imagenet models are trained on either 224 by 224 or 299 by 299 images. So anytime, you have images in that kind of range. That's that's really hopeful you're, probably not going to have to do too much different in this case. The first image I looked at was about the right size, so I'm thinking of pretty hopeful.

So what I did then, is: I created a dictionary comprehension now, if you don't know about list, comprehensions and dictionary comprehensions in Python, go study them they're. The most useful thing super handy you can see the basic idea here is that are going through all of the files and then putting a dictionary that map's the name of the file to the size of that file. Again, this is a handy, little Python feature which I'll, let you think learn about during the week. If you don't know about it, which is zip and using a special star, notation, is never to take this dictionary and turn it into the rows and the columns, and so I can now turn those into num pay, arrays and like okay. Here are the first five rows sizes for each of my images, and then matplotlib is something you want to be very familiar with. If you do any kind of data science or machine learning in python matplotlib, we always refer to as PLT as if this is a histogram. And so I got a histogram of the how high how many rows there are in each image. So you can see here. I'm kind of getting a sense before I start doing any modeling. I kind of need to know what I'm modeling with and I can see. Some of the images are going to be like 2500 3000 pixels high, but most of them seem to be around 500. So, given it so few of them were bigger than a thousand.

I use standard numpy slicing to just grab those at a smaller than a thousand and histogram that just to zoom in a little bit - and I can see here all right - it looks like yet. The vast majority are around 500, and so this actually also prints out the histogram, so I can actually go through and I can see here for four thousand five hundred of them are about 450, okay, so I get about that seems about anywhere so generally, how many Images should we get in the validation set is always a 20 %, so the size of the validation set like using 20 % is fine. Unless you kind of feeling like my data is my data sets really small. I'm not sure. That's enough, you know like if you've got basically think of it. This way, if you train like the same model multiple times and you're, getting very different validation set results and your validation sets kind of small but smaller than a thousand or so then it's going to be quite hard to interpret how well you're doing now. This is particularly true like if you're like, if you care about the third decimal place of accuracy and you've, got like a thousand things in your validation set. Then you bring about like a single image. Changing class is changing. You know it's what you're looking at. So it's, it really depends on my cow, accurate. You have much difference you care about.

I would say in general, like at the point where you care about difference between like out of 0.01 and 0.02, like the second decimal place, you want that to represent like 10 or 20 roads.

You know like changing the class of that 10 or 20 rows. Then that's something you can be pretty confident of so like most of the time you know give them the data sizes we normally have 20 percent seems to work fine, but yeah. It's it's it's kind of a. It depends a lot on specifically what you're doing and what you care about, and it's not it's not a deep learning, specific question either you know so those who are interested in this kind of thing we're going to look into it. A lot more detail in our machine learning course, which will also be available online. Ok, so I did the same thing for the columns just to make sure that these aren't like super wide and I've got similar results and checked in and again found. They're kind of like 4 or 500 seem to be about the average size so based on all of that, I kind of thought. Ok, this looks like a pretty normal kind of image data set that I can probably use pretty normal kinds of models on. I was also particularly encouraged to see that when I looked at the that the dog like takes up most of the frame right, so I'm not too worried about like cropping problems.

You know if the if the dog was just like a tiny little piece of one little corner, that I'd be thinking about doing different. You know maybe zooming in a lot more or something like a medical imaging that happens a lot like often the tumor or the cell. Whatever is like one tiny piece and there's much more complex, so yeah based on all that and this morning I kind of

16. 01:29:15

- **Dog_Breeds initial model, image_size = 64,**
- **CUDA Out Of Memory (OOM) error**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Thought like okay, this looks pretty standard, so I I went ahead and created a little function called get data that basically had my normal two lines of code in it, but I made it so I could passed in a size and a batch size. The reason for this is that when I start working with new data set, I want everything to go super fast, and so, if I use small images, it's going to go super fast, so I actually started out with size equals 64 just to create some super small Images that just go like a second to run through and see how it later on, I started using some big images and some and some also some bigger architectures, at which point I started running out of GPU memory. So I started getting these errors saying CUDA out of memory error when you get a CUDA out of memory error. The first thing you need to do is to go kernel restart. Once you get a code, an out of memory error on your GPU. You can't really recover from it right. Doesn't matter what you do, you know you have to go restart and once I've restarted. I then just changed my batch size to something smaller. So when you call create your data object, you can pass in a batch size parameter. Okay and like i normally use 64 until i hit something that says out of memory and then i'll just have it, and if i still get out of memory I was hobbit again, okay.

So that's where I created this to allow me to like start making my size as bigger as I looked into it more and you know, as I started running out of memory to decrease my batch size. So at this point you know I went through this. A couple of iterations, but I basically found everything, was working fine. So once it's working fine set size 2 to 24 and I created my you know: pre-compute equals true first time I did that it took a minute to create the precomputed activations and then it ran through this in about 4 or 5 seconds, and you can see. I was getting eighty-three percent accuracy. Now. Remember, accuracy means it's it's exactly right, and so

it's predicting out of a hundred and twenty categories. It's predicting exactly right. So when you see something with two classes, is you know 80 % accurate versus something with 120 classes? Is 80 % accurate, they're, very different levels? You know so when I saw like eighty-three percent accuracy with just a pre computed, classify and OData augmentation, though I'm freezing anything else across 120 classes of the oh. This looks good right, so um, then I just kind of kept going throughout at all standard process right. So then I turn precompute off. Okay and cycle length equals one, and I started doing a few more cycles. Few more epochs so remember an epoch is one pass through the data and a cycle is, however, many epochs, you said is in a cycle. It's one, it's the learning rate going from the top that you asked for all the way down.

So since here cycle length equals one a cycle in an epoch at the same okay, so I did. I tried a few epochs. I did actually do the learning rate finder and I found one in a two again looked. Fine. It often looks fine and I found it kind of kept improving, so

17. 01:32:45

- **Undocumented Pro-Tip from Jeremy: train on a small size, then use 'learn.set_data()' with a larger data set (like 299 over 224 pixels)**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

I tried five epochs and I found my accuracy getting better so then I saved that and I tried something which we haven't looked at before, but it's kind of cool. If you train something on a smaller size, you can then actually call learned, set data and pass in a larger size data set and that's gon na take your model. However, it's trained so far and it's going to let you can in you to train on on larger images, and I tell you something amazing. This actually is another way you can get state-of-the-art results and I've never seen this written in any paper or discussed anywhere. As far as I know, this is a new insight. Basically, I've got a pre trained model which, in this case I've trained a few epochs with the size of 224 by 224, and I'm now going to do a few more air pops with the size of 299. By 299, now I've gotten very little data cut out by deep learning standards, only about 10,000 images right so with a 224 by 224. I kind of built this these final layers to try to find things that work well to 24, but to 24. But I go to 299 by 299. I basically, if I over fit before I'm definitely not going to over fit now might have changed the size of my images, they're kind of like totally different but like conceptually they're, still picked. The same kinds of pictures are the same kinds of things, so I found this trick of like starting training on small images for a few a box and then switching to bigger images and continuing training is an amazingly effective way to avoid overfitting.

And it's like it's. So easy and so obvious, I don't understand why it's never been written about before. Maybe it's in some paper somewhere and I haven't found it, but it's I haven't seen it. Would it be possible to do the same thing on using? Let's take a resort our disposal to feed a different size yeah, I think so like as long as you use one of these more modern architectures, what we call fully convolutional architectures, which means not vgg and you'll, see we don't use vgg in this course, because it Doesn't have this property, but most of the architectures developed in the last couple of years can handle pretty much arbitrary sizes yeah be worth trying yeah. I think it ought to work okay, so I call get data again. Remember get data is the just a little function that I created back up here. Right get data is just this little function. That's, oh, I just passed a different size to it, and so I call freeze just to make sure that, but everything

so the last layer is frozen. I mean it actually already was at its point that really doing a thing, and you can see now with free compute off I've now got that data augmentation working. So I kind of run a few more a pox, and what I notice here is that the loss to my training set and the loss of my validation set, my validation set loss is a lot lower than my training set. This is still just training. The last layer, so what this is telling me is I'm under fitting right and so from under fitting.

It means this cycle length equals one is too short, it means it's like finding something better popped with popping out and it's like never getting a chance to zoom. In properly so then I'd set cycle.

18. 01:36:15

- **Using Test Time Augmentation ('learn.TTA()') again**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Mod equals two to give it more time so, like the first time is one epoch. The second one is two epochs. The third one is for epochs and you can see now. The validation, train and training are about the same okay. So that's kind of thinking yeah. This is this is about the right track, and so then I tried using test time augmentation to see if that gets any better still didn't, actually help a hell of a lot just a tiny bit and just kind of at this point I think here this is Nearly done so, I just did it like. You know one more cycle of two to see if it got any better and it did get a little bit better and then I'm like okay, that looks pretty good. I've got a validation, set lost 0.199, and so your Lotus here, actually, you haven't tried unfreezing the reason why I was going to try to unfreezing and training more it didn't get any better, and so the reason for this clearly is that this data set is so Similar the image net that the training that convolutional layers actually doesn't help in the slightest and actually, when I loaded up into it, it turns out that this competition is actually using a subset of improve image net. So, that's okay, so that if we check this out point one nine, nine against the leaderboard, this is only a playground competition. So it's not like the best of here, but you know it's still interesting. It gets us somewhere around ten thrillers, okay and, in fact, we're competing against.

I noticed other the fastai student. This is a fastai student. These people up here, I know they actually posted that they cheated. They actually went. You downloaded the original images and train to that so, and this is why this is a playground, competition. They call it it's not it's not real right. You know it's just to allow us to try things out, but you can basically see out of two hundred and something people where you know we're getting some very good results without doing anything remotely interesting or clever, and we haven't even used the whole data set you're Going to use to eighty percent of it like to get a better result, I would go back and remove that validation set and just rerun the same steps and then submit that exact. Let's just use it under percent of the data. I have three questions. The first one is like that class in this case is very, it's not balanced instead, unbalanced, like it's, not totally balanced, but it's not bad right. It's like between sixty and a hundred like it's it's it's! It's not unbalanced enough that I would give it a second thought: okay, yeah! Let's get to that later in this course, and don't let me forget right, the short answer is that there was a recent list. The paper came out about two or three weeks ago on this, and it said the best way to deal with very unbalanced data sets is to basically make copies of the rare cases yeah. My second question is: I want to pin down a difference between creation.

He read was, and so you have these two options right. So when you beginning I did an optimization use that pre computed. It was true by not using layers right right, so it's not only they frozen their pre computed, so the data augmentation doesn't do anything at that point right before you outcries everything. What as examples you and IV only you only on freeze, so we're going to learn more about the details as we look into the the math and stuff in coming lessons. But basically what happened was we started with a pre trained network right which was kind of finding activations that had these kind of rich features and we were adding, then we add a couple of layers on the end of it, which which start out random, and so With fries equals, with with everything frozen and indeed with pre compute equals. True, all we're learning is told is those couple of layers that we've added and so with pre compute equals. True, we actually pretty calculate like how much does this image have something that looks like this? A ball one looks like this face and so forth, and therefore data augmentation doesn't do anything with pre compute equals true, because you know we're actually showing exactly the same activations. Each time we can then set pre compute equals false, which means it's still only training, those last two layers that we added it's still frozen, but data augmentations now working because it's actually going through and recalculating all of the activations from scratch and then, finally, when we Unfreeze, that's actually saying: okay now you can go ahead and change all of these earlier convolutional filters.

So well you just so the only reason to have pre compute equals true. Is it's just much faster? So it's like it is it's about. You know ten or more times faster, so particularly if you're working with like quite a large data set, you know it can save quite a bit of time, but it's never. There's no. Like companies like accuracy reason ever to use pre computed calls true. It's just a it's just a shortcut. It's also like quite handy. If you're like throwing together a quick model, you know it can take a few seconds to create my last question, which I think you answer is I don't like your suggestions to build a model. You have this aged yeah. What, if would you like? We just wanted one initial setting without these like checking after each I mean if you want it like. If your question is like, is there some shorter version of this? That's like a bit quicker and easier. I could like to lead a few things here. Okay, I think this is a kind of a minimal version to get you a very good result, which is like don't worry about pre compute equals true, because that's just saving a little bit of time. You know so so I still suggest use LR find at the start to find a good learning rate by default. Everything is frozen from the start, so then you can just go ahead and run two or three epochs or cyclic Nichols one unfreeze and then train the rest of the network with differential learning rates.

So it's basically three steps: learning rate, finder trained frozen network with cycle methods, one and then trained unfrozen network, with differential learning rates and cycle molecules too. So, like that's something you could turn into, I guess five or six lines of code at all. I think it's a question provide your own mix book by reusing the batch size. Does the only at better speed of training yeah pretty much so each batch and again we're going to see like all this stuff about precomputing batch sizes. We dig into the details of the algorithms it's going to make a lot more sense intuitively, but basically, if you're, showing it less images each time, then it's calculating the gradient with less images, which means it's less accurate, which means like knowing which direction to go and How far to go in that direction is less accurate, so, as you make the batch size, smaller you're, basically making it kind of more volatile, it's kind of like it kind of impacts, the optimal learning rate that you would need to use, but in practice, where only You know I generally find only dividing with the batch size by like 2 or 4. It doesn't seem to change things very much. Should I reveals the learning rate of quality me if you, if you change the batch size by much, you can rerun the learning rate.

Finder to see if it's changed, if I match, but it it I was in for only generally looking at like a power of 10, it probably is not going to change the it's, not that you can't, because plus back there, this is sort of a conceptual basic Questions we're going back to the previous night, where you should put in the thought behind sorry yeah. This is well one for conceptual, so a basic question: we've actually really slide where you should what the different layers were doing. Yes from this slide, I understand right. The meaning of sync, the third column relative to the fourth column, is that what you're interpreting what the layer is doing, based on what the image is actually yeah? So we're going to look at this in more detail, so these these gray ones basically say this is kind of what the filter looks like. So, on the first layer you can see exactly what the filter looks like, because the input to it of pixels right. So you can absolutely say, and remember we looked at what a convolutional kernel was like. Was that three by three thing? So this look like there's seven by seven kernels, you can say this is actually what it looks like, but later on it's combined. You know the the the input to it are themselves activations which are combinations of activations relation to activations.

So you can't draw it, but there's clever technique that I learned focus created which allowed them to say this is kind of what the filters tended to look like on average alright. So this is kind of what the photos look like and then here is specific examples of patches of image which activated that filter highly. So yet the pictures are the ones that I kind of find more useful because it tells you this kernel is kind of a mini cycle. We all find right. How do we know? That's it well, we'll come back well, we may come back to that. If not in this part in the next part, that probably a part two actually because this paper this paper uses to create these things, this paper uses something called a deconvolution which I'm pretty sure we won't do in this part, but we will do it in part. Two, so if you're interested check out the paper, it's it's in the notebook has a link to it: xyler in Fergus, it's a very clever technique and not terribly intuitive um right. So so you mentioned that it was good that the dog took up the full picture and it would have been a problem if it was kind of like off in one of the corners in really tiny. Well, what would you, what would you technique have been to try to make that work, something that we'll learn about in part two, but basically there's a technique that allows you to to kind of figure out, roughly which parts of an image and most likely to have The interesting things in them and then you can like crop out those bits if you're interested in learning about it.

We did cover it briefly in lesson, seven of part one, but I'm going to actually do it properly in part. Two of this course, because I didn't really cover it thoroughly enough - maybe we'll find time to have a quick look at it, but we'll see I know your Nets.

19. 01:48:10

- **How to improve a model/notebook on Dog_Breeds: increase the image size and use a better architecture.**
- **ResneXt (with an X) compared to Resnet. Warning for GPU users: the X version can 2-4 times memory, thus need to reduce Batch_Size to avoid OOM error**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Written some of the code that we need already. Ah, so once I have something like this notebook, that's basically working, I can immediately make it better by doing two things. Assuming that the size image I was using is smaller than the average size of the image that

we've been given. I can increase the size and, as I showed before, with the dog breeds, you can actually increase it during training. The other thing I can do is to create is to use a better architecture. Now an architect we're going to talk a lot in this course about architectures, but basically there are different ways of putting together like what size convolutional filters and how are they connected to each other and so forth, and different architectures have different like numbers of layers and Sizes of kernels and number of filters and so forth, and so there are some the one that we've been using. Resnet 34 is a great starting point and often a good finishing point, because it's like it's pretty. It doesn't have too many parameters. Often it works pretty. Well, with small amounts of data as we've seen and so forth, but there's actually an architecture that I really like called not res net but res next, which was actually the second-place winner in last year's image: net competition and like ResNet, you can put a number after The res next to say like how big it is and like my next step after resume 34 is always res next 50 now you'll find res next 50 takes like can take like twice as long as ResNet 34 that can take like 2 to 4 times as Much memory as retina 34, so what I wanted to do was I wanted to rerun that previous notebook with res next and increasing the image size to turn on a node.

So here I just said: architecture equals res next, 50 size equals 299, and then I found that I had to take the batch size all the way back to 28 to get it to fit. My GPU is 11 gig. If you're using AWS or cresol. I think they're, like 12 gigs, they might be a bit higher, but this is what I found I had to do so then I this is literally a copy of the previous notebook, so you can actually go file, make a copy right and then rerun it with With these different parameters, and so I deleted some of the pros and some of the expiratory stuff to see you know basically, I said everything else is the same, all the same steps as before. There's my in fact, you can kind of see what this minimum service desk looks like. I didn't need to worry about learning rate finder, so I just left it as is so transforms. Data equals loan equals bit pre computed false feet with cycle integrals. One and freeze differential learning rates bits and more - and you can see here - I didn't do the cycle mop thing, because I found like now that I'm using a bigger architecture, it's got more parameters. It was overfitting pretty quickly so, rather than like cycle length equals one. Never finding the right spot, it actually did find the right spot, and if I used longer cycle legs, I found that my validation error was higher than my training error. It was over there so check us out, though, by using these, you know three steps. I got plus TTA 99.75. So what does that mean? That means I have one incorrect dog for incorrect cats and when we look at the pictures of them, my incorrect dog has a cat.

Now this one is not a either. This one is not either so. I've actually got one mistake and then my incorrect dog is teeth right. So like we're at a point where we're now able to train a classifier, that's so good that it has like basically one's dead right, and so when people say like we have superhuman image performance. Now this is kind of what they're talking about right. So did you actually, when I looked at the dog breed one I did this morning I was like it was. It was getting the dog breeds much better than I ever could so like hits this. This is what we can get to if you use a really modern architect like redneck, and this suddenly took out a tall way and remember, don't like 20 minutes to Train, so that's kind of where we're up to so. If you want to do

20. 01:53:00

- **Quick test on Amazon Satellite imagery competition on Kaggle, with multi-labels**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Satellite imagery instead right, then it's the same thing and in fact the the planet. Satellite data sets already oh and Chris or if you're, using Chris, or you can jump straight there right and I just went into this data stash planet and I can do exactly the same thing right. I can image classifier from CSV right and you can see these three lines are actually exactly the same as my dog breed lines. You know how big, how many lines are in the file grab. My validation indexes this get data, as you can see, it's identical, except I've changed side on to top down the satellite images about top down, so I can fit them vertically and they still make sense right, and so you can see here I'm doing this trick round. Back to size, equals 64 and train a little bit first learning rate find on right and, interestingly, in this case you can see it. I want really high learning rates. I don't know what it is about this particular data set. This is true, but it's clearly I can use super high learning rate, so I use a lot here at a point too, and so I've trained for a while differential learning rates right and so remember - I said like if the data sets very different to image net. I probably want to train those middle layers a lot more, so I'm using divided by three rather than divided by ten, all right, the other than that is the same thing cycle, Nauticals all right and then I just kind of keep an eye on it.

So you can actually plot the loss if you go and learned up shared a plot loss. You can see here that here's the first cycle is. The second cycle is the third cycle right, so you can see, it gets better. Pops out gets better pops out if better pops out and each time it finds something better than the last time then set the size up to 128 and just repeat exactly the last few steps and then set up to 256. Repeat the last two steps and then do TTA and if you submit this - and this gets about 30th place in this competition, so these basic steps work super well this this thing where I went all the way back to a size of 64. I wouldn't do that. If I was doing like dogs and cats or dog breeds because like this is so small that, if if the thing I was working on is very similar to imagenet, I would kind of destroy those imagenet weights like 64 by 64, is so small. But in this case the satellite imagery data, it's so different to imagenet um, you know I really found that it worked pretty well start right back to these tiny images. It really helped me to avoid overfitting and interestingly, using this kind of approach. I actually found that even with using only 128 by 128, I was getting like much better cackled results than really everybody on the leader board and when I say 30th place, this is a very recent competition right, and so I find like in the last year, like A lot of people have got a lot better at computer vision, and so the people in the top 50 in this competition were generally ensemble in dozens of models. Lots of people on a team, lots of pre-processing, specific satellite data and so forth so like to be able to get xxx using this totally standard technique is pretty cool.

Alright. So now that we've got to this point right, we've got through two lessons. If you're still here, then

21. 01:56:30

- **Back to your hardware deep learning setup: Crestle vs Paperspace, and AWS who gave approx $200,000 of computing credits to Fast.ai Part1 V2.**
- **More tips on setting up your AWS system as a Fast.ai student, Amazon Machine Image (AMI), 'p2.xlarge',**
- **'aws key pair', 'ssh-keygen', 'id_rsa.pub', 'import key pair', 'git pull', 'conda env update', and how to shut down your $0.90 a minute with 'Instance State => Stop'**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Hopefully, you're thinking, okay, this is actually pretty useful. I want to do more, in which case Crestle might not be where you want to stay the issues with Crestle. I mean it's, it's it's pretty handy, it's pretty cheap and something we haven't talked about. Much is paper. Space is another great choice. By the way paper space are short, they're going to be releasing kress or like instant Drupal notebooks, unfortunately, they're not ready quite yet, but they do have an ability to. Basically, they have the best price performance relationship right now and they you can SSH into them and use them so they're also a great choice, and, probably by the time this, the MOOC will probably have a separate lesson showing you how to set up set up paper Space because there they're likely to be a great option, but at some point you're probably going to want to look at AWS a couple of reasons why the first is, as you all know, by now, amazon have been kind enough to donate about $ 200,000 worth of Compute time to this course, so I want to say thank you very much to Amazon. We've all been given credit, so everybody this year. So thanks very much hey, don't worry, we're so sorry, you're sure in the MOOC. We didn't get it for you, but everybody here is like AWS credits for everybody, so um, but you can get even if you're, not here in person, you can get AWS credits from lots of places. Github has a student pack, Google for github student pack.

That's like 150 bucks worth of credits. Aws educate can get credits these our office students, so there's lots of places you can get started on AWS, pretty much everybody everybody. A lot of the people that you might work with will be using AWS, because it's like super flexible right now AWS has the fastest available GPUs. You can get in the cloud. They're p3s they're kind of expensive at three bucks an hour. But if you've got like a model where you've done all the steps before you're thinking, this is looking pretty good. You know for 6 bucks you could get a p3 for 2 hours and run turbo speed right um. We didn't start with AWS because well hey it's like twice as expensive as Chris Hall for the cheapest GPU and being a Texan setup right. But I wanted to kind of go through and show you how to get your AWS setup and so we're going to be going slightly over time to do that. But I want to show you a very quick place. I feel prettier if you have to, but I want to show you very quickly how you can get your AWS setup right from scratch. So, basically, you have to go to consult on AWS, but amazon.com and it'll take you to the console right, and so you can follow along on the video with this quickly from here. You have to go to AC. This is where you set up your instances and so from ec2. You need to do what's called launching an instance, so launching an instance means you're, basically creating a computer right now creating a computer on Amazon. So I say launch instance, and what we've done is we've created a fastai.

It's got an amo and ami is like a template for how your computer's going to begin. So if you've got a community, a Mis and type in fastai you'll see that there's one there called fastai part: 1 version 2 for the p2. Ok, so I'm going to select that and then we need to say what kind of computer do you want, and so I can say I want a GPU compute computer and then I can say I want a p2 x large. This is the cheapest reasonably effective for deep learning instance type they have and then I can say launch and then I can say launch, and so at this point they asked you to choose a key pair right now. If you don't have a key pair, you have to create one right so to create a key pair. You need to open your terminal. If you don't have a terminal, if you've got a Mac or Linux box, you've definitely got one if you've got Windows. Hopefully, you've got Ubuntu. If you don't already have Ubuntu setup, you can go to the Windows, Store and click on Ubuntu right, we'll get it from the Windows Store. So from there. You basically go SSH caged in and that will create like a special password for your computer, to be able to log in to Amazon, and then you just hit enter three times. Okay and that's going to create for you, your key. You can use to get into Amazon alright. So then, what I do is I copy that key somewhere that I know where it is so it'll be in the dot SSH folder, it's called IDRs a dub, and

so I'm going to copy it to my hard drive.

So if you're in a macro and Linux it'll already be in an easy to find place, it'll be in your SSH folder that in documents so from there back in AWS, you have to tell it that you've created this key. So you can go to key pairs and you say import key pair and you just browse to that file that you just created there. It is, I say, import okay, so if you've ever used SSH before you've already got the key pair, you don't have to do those depths if you've used AWS before you've already imported it. You don't have to do that step. Maybe haven't done any of those things. You have to do both steps so now I can go ahead and launch my instance community. I am eyes search last day I select launch, and so now it asks me what's where's your key pair and I can choose that one that I just grabbed okay. So this is going to go ahead and create a new computer for me to log into, and you can see here it says the following have been initiated, and so, if I click on that, it'll show me this new computer that I've created okay, so it'll be Able to log into it I need to know its IP address, so here it is the IP address there. Okay, so I can copy that and that's the IP address of my computer so to get to this computer, I need to SSH to it so SSH into a computer means connecting to that computer so that it's like you're typing in that computer.

So I type SSH and they username, for this instance - is always Ubuntu right and then I can paste in that IP address and then there's one more thing I have to do, which is. I have to connect up the jupiter notebook on that instance to the jupiter notebook on my machine and so to do that, there's just a particular flag that i said. Okay, we can talk about it on the forums as to exactly what it does, but you just type l.a today date: localhost, 8, 8. 8. 8. Ok, so like once, you've done it once you can like save that as an alias and type. In the same thing, every time, so we can check here we can see it says that it's running so we should be able to now hit enter first time ever which sit reconnect to it. It does checks. This is okay, I'll, say yes, and then that goes ahead and SSH is in so this ami is all set up for you. Alright, so you'll find that the very first time you log in it takes a few extra seconds because it just kind of is getting everything set up. But once it's logged in you'll see there that there's a directory called fastai and the fastai directory contains our fastai repo that contains all the notebooks or the code, etc. So I can just go CD faster. All right. First thing you do when you get in is to make sure it's updated, so you just go git pull right and that updates to make sure that your repo is the same as the most recent video. And so, as you can see there, we go. Let's make sure it's got all the most recent code.

The second thing you should do is type Condor and update. You can just do this, maybe once a month or so, and that makes sure that the libraries there are all the most recent libraries I'm not going to run that. So it takes a couple of minutes. Okay and then the last step is to type particular notebook. Okay, so this is going to go ahead and launch the triplet notebook server on this machine again, the first time I do it the first time you do everything on AWS. It just takes like a minute or two and then once you've done it in the future. We just as fast as running it locally. Basically, okay, so you can see it's going ahead and firing out the notebook, and so what's going to happen. Is that because, when we SSH into it, we said to both connect our notebook port to the remote notebook port. We're just going to be able to use this locally, so I see he says here copy paste this URL, so I'm going to grab that URL and I'm going to paste it into my browser and that's it okay. So this notebook is now actually not running on my machine. It's actually running on AWS, okay, using the AWS GPU. It's got a lot of memory, it's not the fastest around, but it's not terrible. You can always fire up a p3. If you want something. That's super fast. This is costing me ninety cents a minute okay. So when you're finished, please don't forget to shut it down right so to shut it down. You can right-

click on it and say, instance. Date. Stop. Okay, we've got five hundred bucks of credit.

Assuming that you put your code down in the spreadsheet one thing I forgot to do the first time. I showed you this by the way I said, make sure you choose a p2. The second time I went through, I didn't choose p2 by mistake. So, just don't forget: choose gpq compute P. Do you have a question my Bernice it's an hour. Thank you, 90 cents an hour. It also costs, like I don't know three or four bucks a month for the storage as well. Thanks for checking that all right see you next week, sorry we're a bit over

Lesson 3: Improving your image classifier

Outline

We explain convolutional networks from several different angles: the theory, a video visualization, and an Excel demo. You'll see how to use deep learning for structured/tabular data, such as time-series sales data.

We also teach a couple of key bits of math that you really need for deep learning: exponentiation and the logarithm. You will learn how to download data to your deep learning server, how to submit to a Kaggle competition, and key differences between PyTorch and Keras/TensorFlow.

Video Timelines and Transcript

1. 00:00:05

- **Cool guides & posts made by Fast.ai classmates**
- **tmux, summary of lesson 2, learning rate finder, guide to Pytorch, learning rate vs batch size,**
- **decoding ResNet architecture, beginner's forum**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Welcome back everybody, , I'm sure you've noticed, but there's been a lot of cool activity on the forum this week, and one of the things that's been really great to see is that a lot of you have started creating really helpful materials both for your Classmates to better understand stuff, and also for you to better understand stuff by trying to teach what you've learned. I just wanted to highlight a few. I've actually posted to the wiki thread of a few of these, but there's there's lots more. Russian has posted a whole bunch of nice introductory tutorials, so, for example, if you're having any trouble getting connected with AWS she's got a whole step-by-step. How to go about logging in and getting everything working, which, i think is a really terrific thing, and so it's a kind of thing that, if you are writing some notes for yourself to remind you how to do it, you may as well post them for others To do it as well and by using a markdown file like this, and it's actually good practice if you haven't used github before, if you put it up on github, everybody can now use it or, of course you can just put it in the forum. So more advanced a thing that Reshma wrote up about is she noticed that I like using Tmax, which is a handy little thing which lets me lets me basically have a window I'll show you so as soon as I log into my computer.

If I run Tmax you'll see that all of my windows pop straight up basically - and I can like continue running stuff in the background - and I can like I've - got them over here and I can kind of zoom into it or I can move over to the Top, which is here so Jupiter, colonel running and so forth. So if that sounds interesting, Reshma has a tutorial here on how you can use

two maps and it's actually got a whole bunch of stuff in her github. So that's that's really cool I built, among has written a very nice kind of summary, basically of our last lesson, which kind of covers. What are the key things we did, and why did we do them? So if you are a kind of wondering like how does it fit together, I think this is a really helpful summary like what, if those couple of hours look like, if we summarize it all into a page or two, I also really like Pavel has dad kind Of done a deep dive on the learning rate finder, which is a topic that a lot of you have been interested in learning more about, particularly those of you who have done deep learning before. I realized that this is like a solution to a problem that you've been having for a long time and haven't seen before, and so it's kind of something which hasn't really been blogged about before. So this is the first I've seen it's logged about. So when I put this on Twitter, a link to pebbles post, it's been shared, you know hundreds of times it's been really really popular and viewed many thousands of times.

So that's some great content. Radec has posted lots of cool stuff. I really like this practitioners guide to apply torch, which again this is more for more advanced students, but it's like digging into people who have never used pytorch before but know a bit about numerical programming in general. And it's a quick introduction to how high torch is different and then there's been some interesting little bits of research like what's the relationship between learning rate and batch sites. So one of the students actually asked me this before class, and I said oh well, one of the other students has written an analysis of exactly that. So what he's done is basically looked through and tried different batch sizes and different learning rates and tried to see how they seemed to relate together, and these are all like cool experiments, which you know you can try yourself. I predict again he's written something again, a kind of a research into this question. I made a claim that the the stochastic gradient descent with restarts finds more generalizable parts of the function surface, because they're kind of flatter and he's been trying to figure out. Is there a way to measure that more directly, not quite successful yet? But a really interesting piece of research got some introductions to convolutional neural networks and then something that we'll be learning about towards the end of this course.

But I'm sure you've noticed we're using something called ResNet and a nonce. Aha, actually posted a pretty impressive analysis of like watts arrest net, and why is it interesting and this one's actually being very already shared very widely around the internet? I've seen also so some more advanced students who are interested in jumping ahead can look at that, and uphill Tamang also has done something similar, so lots of yeah lots of stuff going on on the forums, I'm sure you've also noticed. We have a beginner forum now specifically, for you know, asking questions which you know. It is always the case that there are no dumb questions, but when there's lots of people around you talking about advanced topics, it might not feel that way. So hopefully, the beginners forum is just a less intimidating space and if there are more advanced student who can help answer those questions, please do but remember when you do answer those questions, try to answer in a way that's friendly to people that maybe you know have No more than a year of programming experience, you haven't done any machine learning before so you know, I hope, other

2. 00:05:45

- **Where we go from here**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

People in the class feel like you can contribute as well and just remember all of the people we just looked at or many of them, I believe, have never hosted anything to the internet before right I mean you, don't have to be a particular kind of person To be allowed to block something you can just jot down your notes, throw it up there, and one handy thing is: if you just put it on the forum and you're, not quite sure of some of the details, then then you know you have an opportunity to Get feedback and say like: oh well, that's not quite how that works. You know. Actually it works this way instead or oh, that's a really interesting insight. Have you thought about taking this further and so forth? So what we've done so far is a kind of an injury, an introduction as a just as a practitioner to convolutional neural networks for images, and we haven't really talked much at all about the theory or why they work or the math of them. But on the other hand, what we have done is seen how to build a model which actually works exceptionally well. Compact world-class level models and we'll kind of review a little bit of that today and then also today, we're going to dig in a little quite a lot more actually into the underlying theory of like what is a. What is a CNN? What's a convolution, how does this work and then we're going to kind of go through this this cycle, where we're going to dig we're going to do a little intro into a whole bunch of application areas using neural nets for structured data, so kind of like logistics Or forecasting, or you know, financial data or that kind of thing, and then looking at language applications and LP applications using recurrent neural Nets and then collaborative filtering for recommendations and systems, and so these will all be like similar to what we've done for cnn's. For images.

Would be like here's how you can get a state-of-the-art result without digging into the theory, but but knowing how to actually make a work and then we're kind of going to go back through those almost in reverse order. So then, we're going to dig right into collaborative filtering in a lot of detail and see how how to write the code underneath and how the math works underneath and then we're going to do the same thing for the structured data analysis. We're going to do the same thing for comp nets, for images and, finally, an in depth deep dive into apparent neural networks. So that's

3. 00:08:20

- **How to complete last week assignement “Dog breeds detection”**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Kind of where we're okay! So, let's start by doing a little bit of a review, and I want to also provide a bit more detail on some on some steps that we only briefly slipped over. So I want to make sure that we're all able to complete kind of last week's assignment, which was that the dog breeze I mean to basically apply what you've learned with another data set, and I thought the easiest one to do with me. The dog breeds cattle competition, and so I want to make sure everybody has everything you need to do this right now so, and the

4. 00:08:55

- **How to download data from Kaggle (Kaggle CLI) or anywhere else**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

First thing is to make sure that you know how to download data, and so there's there's two main places at the moment. We're kind of downloading data from one is from cattle and the other is from like anywhere else and so I'll. First of all, do the the casual version so to download from cattle we use something called cattle CLI, which is gear and to install what I think it's already in the system will shake yeah, so it should already be in your environment. But to make sure one thing that happens is because this is downloading from the cattle website through experience rating every time cap will change us the website it breaks so anytime. You try to use it and if the cattles websites changed recent, when you'll need to make sure you get the most recent version, so you can always go to pip, install cable CL I upgrade, and so that'll just make sure that you've got the latest version of Of it and everything that it depends on okay, and so then, having done that, you can follow the instructions. Actually I think Reshma was kind enough to they go. There's a cable CLI. You know, like everything you need to know, can be under Reshma's github, so basically to do that at the next step you go kg download and then you provide your username with you. You provide your password with P and then see it.

The competition name and a lot of people in the forum is being confused about what to enter here, and so the key thing to note is that, when you're at a capital, competition after the /c, there's a specific name - planet, understanding, etcetera right, that's the name! You need okay, the other thing you'll need to make sure is that you've on your own computer have attempted to click download, at least once because when you do ask you to accept the rules, if you've forgotten, to do that, kg download will give you a hint It'll say: oh, it looks like you might have forgotten the rules. If you log into cattle with like a Google account like anything other than a username password, this won't work so you'll need to click forgot password on Kaggle and get them to send you a normal password. So that's the cattle version right, and so, when you do that, you end up with a whole folder created for you with all of that competition and data in it. So a couple of reasons you might want to not use that the first years that you're using a data set that's not on cattle. The second is that you don't want all of the data sets in a cattle competition, for example the planet, competition that we've been looking at a little bit. We'll look at again today has data in two formats TIFF and JPEG. The TIFF is 19 gigabytes and the JPEG is 600 megabytes.

5. 00:12:05

- **Cool tip to download only the files you need: using CulrWget**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

So you probably don't want to download both so I'll. Show you a really cool kit which actually somebody on the forum taught me, I think, was one of the young MSN students here at USF. There's a Chrome extension cord curl wget so you can just search for a curl wget, and then you install it by just clicking on installed and having an extension before and then from now on. Every time you try to download something so I'll, try and download this file and I'll just go ahead and cancel it right, and now you see this little yellow button. That's added up here: there's a whole command here right, so I can copy that and paste it into my window and hit go and it's there cuz okay. So what that does is like all of your cookies and headers, and everything else needed to download that file is like say so. This is not just useful for downloading data. It's also useful. If you like, trying to download some. I don't know TV show or something anything where you're it's hidden behind a login or something you can.

You can grab it, and actually that is very useful for data science, because quite often we want to analyze things like videos on our on our consoles. So this is a good trick. Alright, so there's two ways to get the data, so then, having

6. 00:13:35

- **Dogs vs Cats example**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Got the data you then need to build your model right, so what I tend to do like you'll notice that I tend to assume that the data is in a directory called data. That's a subdirectory of wherever your notebook is right. Now you don't necessarily actually want to put your data there. You might want to put it directly in your home directory or you might wan na put it on another drive or whatever. So what I do is, if you look inside my courses, do one folder you'll see that data is actually a symbolic link to a different drive. Alright, so you can put it anywhere you like, and then you can just add a symbolic link or you can just put it there directly. It's up to you if you haven't used symlinks before they're, like aliases or shortcuts on the mac or windows very handy and there's some threads on the forum about how to use them. If you want help with that, that, for example, is also how we actually have the fastai modules available from the same place as our notebooks, it's just a symlink to where they come from anytime. You want to see like where things actually point to in Linux. You can just use the L flag listing a directory and that'll show you where the symlinks exist still lost I'll, show you which scenes the directories so forth. Okay, so one thing which may be a little unclear based on what we've done so far, is like how little code you actually need to do this end to it. So what I've got here is is in a single window.

Is an entire end-to-end process to get a state-of-the-art result? Put cats versus dogs all right, I've only step. I've skipped is the bit where we've downloaded it from title and then, where we unzip it all right. So these are literally all the steps, and so we import our libraries and actually, if you import this one Kampf loner that basically imports everything else. So that's that we need to tell at the path of where things are the size that we want the batch size that we want right so then, and we're going to learn a lot more about what these do very shortly. But basically we say how do we want to transform our data, so we want to transform it in a way, that's suitable to this particular kind of model, and it assumes that the photos aside on photos and that we're going to zoom in up to ten percent. Each time we say that we want to get some data based on paths, and so remember this is this idea that there's a path called cats and the path called dogs and they're inside a path called train and a path called valid. Note that you can always overwrite these with other things. So if your things are in different folders, you could either rename them or you can see here, there's like a train name and a bowel name. You can always pick something else here. Also notice there's a test name, so if you want to submit some into cattle, you'll need to fill in the name, the name of the folder, where the test sentence and obviously those those won't be labeled.

So then we create a model from a pre-training model. It's from a resonant 50 model using this data, and then we call fit and remember by default that has all of the layers, but the last few frozen and again we'll learn a lot more

7. 00:17:15

- **What means “Precompute = True” and “learn.bn_freeze”**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

About what that means - and so that's that's what that does so that that took two and a half minutes notice. Here, I didn't say, pre-compute equals true again, there's been some confusion on the forums about like what that means. It's only a is only something that makes it a little faster for this first step right, so you can always skip it and if you're at all confused about it or it's causing you any problems, just leave it off right because it's just a it's just a Shortcut which caches some of the intermediate steps that don't have to be recalculating each time and remember that when we are using pre computed, activations data or augmentation doesn't work right. So, even if you ask for a data augmentation, if you've got pre compute equals true, it doesn't actually do any data augmentation, because it's using the cached non augmented activations. So in this case to keep this as simple as possible, I have no pre computed anything going on, so I do three cycles of length one and then I can then unfreeze. So it's now going to train the whole thing something we haven't seen before and we'll learn about in the second half is called B and freeze. For now. All you need to know is that, if you're using a model like a bigger, deeper model like ResNet, 50 or rez next 101 on a data set, that's very very similar to imagenet like these cat sandbox data set sort of words. It's like sidon photos of standard objects.

You know of a similar size to image turn and money somewhere between 200 and 500 pixels. You should probably add this line when you unfreeze for those of you that are more advanced. What it's doing is it's causing the batch normalization moving averages to not be updated, but in the second half of this course we're gon na learn all about. Why we do that. It's something that's not supported by any other library, but it turns out to be super important anyway. So we do one more epoch with training the whole network and then at the end we use test time augmentation to ensure that we get the best predictions we can, and that gives us ninety nine point, four. Five percent. So that's that's it right. So when you try a new data set they're, basically the minimum set of steps that you would need to follow. You'll notice. This is assuming I already know what learning wrote to use so you'd use the learning rate finder. For that. It's assuming that I know that the directory layout and so forth, so that's kind of a minimum set. Now one of the things that I wanted to make sure you had an understanding of how to do is how to use other libraries

8. 00:20:10

- **Intro & comparison to Keras with TensorFlow**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Other than fastai - and so I feel like the best thing to to look at, is to look at care us because care us is a library just like fastai sits on top of pytorch care. Us sits on top of actually a whole variety of different backends. It fits mainly people nowadays use it with tensorflow there's, also an MX net version, there's also a Microsoft CNT K version. So what I've got? If you do a git pull you'll see that there's a something called care us less than one where I've attempted to replicate at least parts of lesson, one in care us just to give you a sense of how that works. I'm not going to talk more about batch two norm, freeze now other than to say, if you're, using

something which has got a number larger than 34 at the end, so like ResNet, 50 or res next 101 and you're trading. A data set that has that is very similar to image net. So it's like normal photos of normal sizes. Where the thing of interest takes up most of the frame, then you probably should at the end fries true after unfreeze, if in doubt try trading it with and then try trading it without more advanced students will certainly talk about it on the forums this week and We will be talking about the details of it in the second half of the course when we come back to our CNN in death section in the second last lesson, so with care us again, we import a bunch of stuff and remember.

I mentioned that this idea that you've got a thing called train and a thing called valid and inside that you've got a thing called dogs and things called cats is a standard way of providing image, labelled images, so Karis does that too right? So it's going to tell it where the training set and the validation set are twice what batch size to used now you'll notice in Karis. We need much much much more code to do the same thing. More importantly, each part of that code has many many many more things you have to set and if you set them wrong, everything breaks right so I'll give you a summary of what they are so you're. Basically, rather than creating a single data object in chaos, we first of all have to define something called a data generator to say kind of generate the data, and so a data generator. We basically have to say what kind of data augmentation we want to do, and we also we actually have to say what kind of normalization do we want to do so we're else with fastai. We just say whatever ResNet 50 requires just do that. For me, please, we actually have to kind of know a little bit about. What's expected of us. Generally speaking, copying and pasting cos code from the internet is a good way to make sure you've got the right, the right stuff to make that work and again it doesn't have a kind of a standard set of like here. The best data augmentation parameters to use for photos, so you know I've copied and pasted all of this from the Kaos documentation.

So I don't know if it's I don't think it's the best set to use it all, but it's the set that they're using in their docks. So having said this is how I want to generate data so horizontally fit. Sometimes you know zoom, sometimes sheer. Sometimes we then create a generator from that by taking that data generator and saying I want to generate images by looking from a directory, we pass in the directory, which is of the same directory structure that fastai users and you'll, see there's some overlaps with kind Of how fastai works here, you tell it what size images you want to create. You tell at what batch size you want in your mini batches and then there's something you not to worry about too much, but basically, if you're just got two possible outcomes, you would generally say binary here. If you've got multiple possible outcomes would say categorical, yeah. So we've only got cats or dogs, so it's binary. So an example of like where things get a little more complex. Is you have to do the same thing for the validation set? So it's up to you to create a data generator that doesn't have data augmentation because, obviously for the validation set unless you're using t/ta. That's going to start things up you also when you train you randomly reorder the images so that they're always shown in different orders to make it more random, but with a validation.

It's vital that you don't do that, because if you shuffle the validation set, you then can't track how well you're doing it's in a different order for the labels. That's a basically, these are the kind of steps you have to do every time with care us. So again, the reason I was using ResNet 50 before is chaos doesn't have ResNet 34. Unfortunately, so I just wanted to compare like with Mike so we're going to use resident 50 here there isn't the same idea with chaos of saying, like constructor model that is suitable for this data set for me, so you have to do it by hand right. So the way you do it is to basically say this is my base model, and then you have to construct on top of that manually. The layers that you want to add, and so by the end of

this course you'll understand a way. It is that these particular three layers, other layers, that we add so having done that in chaos, you basically say: okay, this is my model and then again there isn't like a concept to it like automatically freezing things or an API for that. So you just have to allow look through the layers that you want to freeze and call dot. Trainable equals false on them. In Karis, there's a concept we don't have in fastai or play a torch of compiling a model. So, basically, once your model is ready to use, you have to compile it passing in what kind of optimizer to use what kind of loss to look for about metric so again with fastai.

You don't have to pass this in because we know what loss is the write loss to use. You can always override it, but for a particular model we give you good defaults, okay, so having done all that rather than calling fit, you call generator passing in those two generators that you saw earlier: the Train generator in the validation generator for reasons. I don't quite understand. Chaos expects you to also tell it how many batches there are per epoch, so the number of batches is a quarter the size of the generator divided by the batch size. You can tell it how many epochs, just like in fastai. You can say how many processes or how many workers to use for pre-processing , unlike fastai, the default in chaos, is basically not to use any so to get good speed. You're going to make sure you include this, and so that's basically enough to start fine tuning the last layers. So, as you can see, I got to a validation, accuracy of 95 %, but, as you can also see something really weird happened. We're after one it was like 49 and then it was 69 and then 95. I don't know why these are so low, that's not normal. I may have there may be a bug and chaos. They may be a bug. In my code. I reached out on Twitter to see if anybody could figure it out, but they couldn't. I guess this is one of the challenges with using something like this is one of the reasons I wanted to use fast.

Ai, for this course is it's much harder to screw things up, so I don't know if I screwed something up or somebody else did yes, you know this is you've, seen the chance to float back end yeah yeah, and if you want to run this to try It out yourself, you just can just go: pip install tensorflow, GPU, Kerris, okay, because it's not part of the faster I environment by default. But that should be all you need to do to get that working. So then there isn't a concept of like layer, groups or differential learning rates or partial unfreezing or whatever. So you have to decide like I had to print out all of the layers and decide manually how many I wanted to fine-tune. So I decided to fine-tune everything from a layer 140 onwards. So that's why I just looked through like this after you change that you have to recompile the model and then after that I then ran another step. And again I don't know what happened here. The accuracy of the training set stayed about the same, but the validation set totally fill in the hole, and I mean the main thing to notice. Even if we put aside the validation set, we're getting, I mean, I guess the main thing is there's a hell of a lot more code here which is kind of annoying, but also the performance is very different. So where else is here even on the training set? We're getting like 97 % after four epochs that took a total of about eight minutes. You know over here we had 99.5 % on the validation set and it ran a lot faster.

So I was like four or five minutes right so, depending on what you do, particularly if you end up wanting to deploy stuff to

9. 00:30:10

- **Porting PyTorch fast.ai library to Keras+TensorFlow project**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Mobile devices at the moment, the kind of pytorch on mobile situation is very early, so you may find yourself wanting to use tensorflow or you may work for a company, that's kind of settled on tensorflow. So if you need to convert something like redo, something you've learnt here intensive flow, you probably want to do it with care us, but just recognize. You know it's got to take a bit more work to get there and by default it's much harder to get. I mean I to get the same state of the out results you get the faster I you'd have to like replicate all of the state-of-the-art algorithms that are in first a nice. So it's hard to get the same level of results, but you can see the basic ideas are similar. Okay and it's certainly it's certainly possible. You know like there's nothing I'm doing in fastai that, like would be impossible, but, like you'd, have to implement stochastic gradient percent with restarts, you would have to implement differential learning rates. You would have to implement batch norm, freezing which you probably don't want to do. I know and well that's not quite true, I think somewhat one person, at least on the forum is attempting to create a chaos, compatible version of or a tensorflow compatible version of fastai, which I think I hope will get there. I actually spoke to Google about this a few weeks ago and they're very interested in getting faster. I ported to tensorflow, so maybe by the time you're looking at this on the mooc.

Maybe that will exist. I certainly hope so. We will see hey wait. So Karis is Karis. Intensive flow was certainly not you know that difficult to handle, and so I don't think you should worry. If you're told you have to learn them after this course. For some reason, even let me take you a couple of days, I'm sure. So that's kind of most of the stuff. You would need to kind of complete this. This kind of assignment from last week, which was like try to do everything you've seen already but on the dog of reinstated, said just to remind you that kind of last few minutes of last week's lesson. I show you how to do much of that, including like

10. 00:32:30

- **Create a submission to Kaggle**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

How I actually explored the data to find out like what the classes were and how big the images were and stuff like that right. So if you've forgotten that or didn't quite follow at all last week, check out the video from last week to see one thing that we didn't talk about is how do you actually submit to Carol? So how do you actually get predictions, so I just wanted to show you that last piece as well and on the wiki thread this week. I've already put a little image of this to show you these days. But if you go to the kaggle website for every competition, there's a section called evaluation and they tell you what it's a bit, and so I just copied and pasted these two lines from from there. And so it says we're expected to submit a file where the first line contains the the work, the word ID and then a comma separated list of all of the possible dog breeds. And then every line after that will contain the idea itself, followed by all the probabilities of all the different dog breeds. So how do you create that so recognize that inside our data object, there's a dot classes which has got in alphabetical order? All of the four other classes, and then so it's got all of the different classes and then inside data dot test data set just yes, you can also see there's all the file names so and just to remind you, dogs and cats.

Sorry, cats dog breeds was not provided in the kind of care, our style format where the dogs and cats from different folders, but instead it was provided as a CSV file of labels right. So

when you get a CSV file of labels, you use image classifier data from CSV, rather than image classifier data from parts there isn't an equivalent in care us so you'll, see like on the cattle forums. People share scripts for how to convert it to a care. Our style folders, but in our case we don't have to we just go image: classifier data from CSV passing in that CSV file, and so the CSV file will, you know, has automatically told the data. You know what the masses are and then also we can see from the folder of test images, what the file names of those are. So, with those two pieces of information we're ready to go. So I always think it's a good idea to use TTA, as you saw with that dogs and cats example. Just now. It can really improve things, particularly when your model is less good, so I can say, learn dot, t ta and if you pass in yeah, if you pass in is test equals true, then it's going to give you predictions on the test set rather than the validation Set okay and now, obviously, we can't nowget an accuracy or anything because by definition we don't know the labels for the test set right. So by default, most high-touch models.

Give you back the log of the predictions. So then we just have to go X. Of that to get back out probabilities, so in this case the test set had ten thousand three hundred and fifty seven images in it and there are 120 possible breeds all right. So we get back a matrix of of that size, and so we now need to turn that into something that looks like this, and so the easiest way to do that is with pandas. If you're not familiar with pandas. There's lots of information online about it or check out the machine learning course intro to machine learning that we have where we do lots of stuff with pandas. But basically we can describe PD data frame and pass in that matrix and then we can say the names of the columns are equal to data duck classes and then finally, we can insert a new column at position. 0 called ID that contains the file names, but you'll notice that the file names contain five letters at the end. I start we don't want and four letters at the end we don't want, so I just subset in like so right. So at that point I've got a data frame that looks like this, which is what we want, so you can now call a data frame data, so social cues dated DF not des. Let's fix it now, data frame. Okay, so you can now call data frame to CSV and quite often you'll find these files actually get quite big. So it's a good idea to say compression equals gzip and that'll, zip it up on the server for you and that's going to create a zipped up.

Csv file on the server on wherever you're running is Jupiter notebook. So you need apps that you now need to get that back to your computer, so you can upload it or you can use carol CLA, so you can type kgs submit and do it that way. I generally download it to my computer so like how often back to this lab double check. It all looks, ok, so to do that. There's a cool little theme called file link and if you run file link with a path on your server, it gives you back a URL which you can click on and it will download that file from the server onto your computer. So if I click on that now I can go ahead and save it, and then I can see in my downloads there it is here's my submission file, they want to open their yeah and, as you can see, it's exactly what I asked for there's my ID And 120 different dog breeds and then here's my first row containing the file name and the 120 different probabilities. Okay, so then you can go ahead and submit that to cattle through their through their regular form, and so this is also a good way. You can see. We've now got a good way of both grabbing any file off the internet and getting a to our AWS instance or paper space or whatever, by using the cool little extension in chrome and we've also got a way of grabbing stuff off our server easily. Those of you that are more command line oriented.

You can also use SCP, of course, but I kind of like doing everything through the notebook all

11. 00:39:30

- **Making an individual prediction on a single file**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Right one other question I had during the week was like what, if I want to just get a single, a single file that I want to, you know, get a prediction for so, for example, you know. Maybe I want to get this first file from my validation set, so there's its name, so you can always look at a file just by calling image, dot open that just uses regular imaging library, and so what you can do is there's actually I'll show you the Shortest version, you can just call learn, predict array passing in your your image. Okay, now the image needs to have been transformed, so you've seen, transform Trent transforms from model before normally we just put put it all in one variable, but actually behind the scenes it was returning to things it was returning. Training transforms and validation transforms, so I can actually split them apart, and so here you can see, I'm actually applying example. My training transforms or probably more likely that would apply, validation transforms. That gives me back an array containing the image. The transformed image, which I can then past everything that gets passed to or returned from bottles is generally assumed to be a mini batch right. It's generally assumed to be a bunch of images. So we'll talk more about some numpy tricks later, but basically, in this case we only have one image, so we have to turn that into a mini batch of images.

So, in other words, we need to create a tensor that basically is not just rows by columns by channels, but it's number of image by rows by columns by channels and as one image. So it's basically becomes a 4 dimensional tensor. So, there's a cool little trick in numpy that if you index into an array with none that basically adds additional unit access to the start. So it turns it from an image into a mini batch of one images. And so that's why we had to do that. So if you basically find you're trying to do things with a single image with any kind of pytorch or fastai thing, this is just something you might. You might find. It says, like expecting. Four dimensions only got three; it probably means that or if you get back a return value from something that has like some weird first access, that's probably why it's probably giving you like back a mini batch. Okay, and so we'll learn a lot more about this. But it's just something:

12. 00:42:15

- **The theory behind Convolutional Networks, and Otavio Good demo (Word Lens)**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

To be aware of okay, so that's kind of everything you need to do in practice. So now we're going to kind of get into a little bit of theory, what's actually going on behind the scenes with these convolutional neural networks, and you might remember it back in Lesson one. We actually saw our first little bit of theory, which we stole from this fantastic websites, a toaster dot, IO, either explained visually, and we learnt that a that. A convolution is something where we basically have a little matrix in deep learning, nearly always three by three. A little matrix that we basically multiply every element of that matrix by every element of a three by three section of an image add them all together to get the result of that convolution at one point all right now, let's see how that all gets turned together To create these, these various layers that we saw in the the Zeiler and burgers paper and to do that again, I'm going to steal off somebody who's, much smarter than I am we're going to steal from a guy called Ottavia good

Ottavia. Oh good was the guy who created Word Lens, which nowadays is part of Google Translate. If I'm Google Translate you've ever like done that thing, where you point your camera at something at something which has any kind of foreign language on it and in real-time it overlays it with the translation. That was a views company that built that, and so Tokyo was kind enough to share this fantastic video.

He created he's at Google now and I want to kind of step you through works. I think it explains really really well what's going on and then after we look at the video we're going to see how to implement the whole a whole sequence of kintyre set of layers of convolution on your network in Microsoft, Excel so with you're, a visual learner Or a spreadsheet learner, hopefully you'll be able to understand all this okay, so we're going to start with an image and something that we're going to do later in the course is we're going to learn to nice digits. So we'll do it like end to end we'll. Do the whole thing, so this is pretty similar, so we're going to try and recognize in this case letters so here's an A which, obviously it's actually a grid of numbers right and so there's the creative numbers, and so what we do is we take our first Convolutional filter, so we're assuming this is all this is assuming that these are already learned right and you can see this point. It's got wiped down the right-hand side right and black down the left, so it's like zero, zero, zero, maybe negative. 1 negative 1 negative. 1. 0, 0, 0, 1, 1, 1, and so we're taking each 3x3 part of the image and multiplying it by that 3x3 matrix, not as a matrix product that an element-wise product. And so you can see what happens is everywhere, where the the white edge is matching the edge of the a and the black edge.

Isn't we're getting green we're getting a positive and everywhere, where it's the opposite, we're getting a negative we're getting a red right, and so that's the first filter creating the first that the result of the first kernel right and so here's a new kernel. This one is: it's, got a white stripe along the top right, so we literally scan through every 3x3 part of the matrix multiplying those 3 bits of the a the neighbors of the a by the 9 bits as a filter to find out whether it's red or Green and how red or green it is ok, and so this is assuming we had two filters. One was a bottom edge, one was a left edge and you can see here the top edge - not surprisingly it's red here, so a bottom edge was red here and green here, the right edge right here in green here and then in the next step. We add a non-linearity, ok, the rectified linear unit, which literally means strongly the negatives. So here the Reds all gone: okay, so here's layer 1, the input, here's layup to the result of 2 convolutional filters, here's layer 3, which is which is throw away all of the red stuff and that's called a rectified linear unit and then layer 4 is something Called a max pull on a layer 4, we replace every 2 by 2 part of this grid and we replace it with its maximum mat. So it basically makes it half the size. It's basically the same thing but half the size, and then we can go through and do exactly the same thing.

We can have some new filter three by three filter that we put through each of the two results of the previous layer. Okay and again, we can throw away the red bits right so get rid of all the negatives, so we just keep the positives. That's called applying a rectified linear unit and that gets us to our next layer of this convolutional neural network. So you can see that by you know at this layer back here it was kind of very interpretive. All it's like we've either got bottom edges or left edges, but then the next layer was combining the results of convolutions. So it's starting to become a lot less clear, like intuitively, what's happening, but it's doing the same thing and then we do another max pull right. So we replace every 2x2 or 3x3 section with a single digit. So here this 2x2, it's all black. So we replaced it with a black right and then we go and we take that and we we compare it to basically a kind of a template of what we would expect to see if it was an a it was a B, but the see it was D, give it an E and we see how closely it matches and we can do it in exactly the same way. We can multiply every one of the

values in this four by eight matrix with every one of the four by eight in this one, and this one and this one - and we add - we just add them together to say like how often does it match versus how Often does it not match and then that could be converted to give us a percentage probability that this is a no.

So in this case this particular template matched well with a so notice, we're not doing an each training here right. This is how it would work if we have a pre trained model all right, so when we download a pre trained imagenet model off the internet and isn't on an image without any changing to it. This is what's happening or if we take a model that you've trained and you're, applying it to some test set or for some new image. This is what it's doing all right as it's basically taking it through it, buying a convolution to each layer to each multiple convolutional filters to each layer and then doing the rectified linear unit, so throw away the negatives and then do the max pull and then repeat That a bunch of times - and so then we can do it with a new letter, A or letter B or whatever, and keep going through that process right. So, as you can see, that's far nice, the visualization thing and I could have created because I'm not at a vo. So thanks to him for sharing this with us, because it's totally awesome he actually. This is not done by hand. He actually wrote a piece of computer software to actually do these

13. 00:49:45

- **ConvNet demo with Excel,**
- **filter, Hidden layer, Maxpool, Dense weights, Fully-Connected layer**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Convolutions, this is actually being actually being done, dynamically, which is pretty cool, so I'm more of a spreadsheet guy. Personally, I'm a simple person. So here is the same thing now in spreadsheet. All right and so you'll find this in the github repo. So you can either get clone the repo to your own computer, open up the spreadsheet or you can just go to github.com, slash, /, ji and click on this. It's it's inside. If you go to our repo and just go to courses as usual, go to deal 1 as usual, you'll see, there's an Excel section there, okay, and so he lay all that, so you can just download them by clicking them or you can clone the whole repo And we're looking at cognitive example, convolution example all right, so you can see. I have here an input right. So in this case the input is the number 7. So I grab this from a dataset called m-must, MN ist, which we'll be looking at in a lot of detail, and I just took one of those digits at random and I put it into Excel, and so you can see every Hextall is actually just a number Between 9 1, okay, very often actually it'll be a bite between Norton 255 or sometimes it might be a float between naught and 1. It doesn't really matter by the time it gets to pytorch, we're generally dealing with floats. So we, if one of the steps we often will take, will be to convert it to a number between naught 1. So then, you can see I've just use conditional formatting in Excel to kind of make the higher numbers more red. So you can clearly see that this is a red.

This is a 7, but but it's just a bunch of numbers that have been imported into Excel. Okay, so here's our input so remember what at a via did was he then applied two filters right with different shapes, so here I've created a filter which is designed to detect top edges. So this is a 3 by 3 filter. Okay and I've got ones along the top zeroes in the middle minus ones at the bottom right. So let's take a look at an example. That's here right, and so, if I hit that you can see here highlighted this is the 3 by 3 part of the input that this particular thing is calculating

right. So here you can see, it's got. 1 1 1 are all being multiplied by 1 and point 1. 0 0 are all being multiplied by negative 1. Okay, so in other words, all the positive bits are getting a lot of positive. The negative bits are getting nearly nothing at all. So we end up with a high number okay, where else on the other side of this bit of the 7 right, you can see how you know this is basically zeros here or perhaps more interestingly, on the top of it. Okay, here we've got high numbers at the top, but we've also got high numbers at the bottom, which are negating it ok, so you can see that the only place that we end up activating is where we're actually at an edge. So, in this case this here this number 3, this is called an activation.

Ok, so when I say an activation, I mean ah at number, a number that is calculated and it is calculated by taking some numbers from the input and applying some kind of linear operation, in this case, a convolutional kernel to calculate an output right, you'll notice, that Other than going inputs multiplied by kernel and summing it together right. So here's my some and here's my x then take that and I go max of 0 comma that and so that's my rectified linear unit. So it sounds very fancy rectified linear unit, but what they actually mean is open up, Excel and type equals max 0, comma C. Ok, that's all about! Then you'll see people in the biz so to say value a so ral. You means rectified. Linear unit means max 0 comma thing and I'm not like simplifying it. I really mean it like when I say like, if I'm simplifying, I always say so, I'm simplifying, but if I'm not saying I'm simplifying, that's the entirety, okay, so a rectified linear unit in its entirety is this and a convolution in its entirety is? Is this okay? So a single layer of a convolutional neural network is being implemented in its entirety here in Excel okay, and so you can see what it's done is it's deleted pretty much the vertical edges and highlighted the horizontal edges. So again, this is assuming that our network is trained and that, at the end of training it a created a convolutional filter with these specific line numbers in - and so here is a second convolutional filter.

It's just a different line numbers now: pytorch doesn't store them as two separate nine digit arrays. It stores it as a tensor right. Remember, a tensor just means an array with more dimensions. Okay, you can use the word array as well. It's the same thing, but in pytorch they always use the word tensor, so I'm going to say cancer, okay. So it's just a tensor with an additional axis which allows us to stack each of these filters together. Right, filter and kernel. Pretty much mean the same thing: yeah right. It refers to one of these three by three matrices or one of these three by three slices of a three dimensional tensor. So if I take this one - and here I've literally just copied the formulas in Excel from above. Okay - and so you can see this, one is now finding a vertebra which, as we would expect okay, so we've now created one layer right. This here is a layer them specifically we'd, say it's a hidden layer which is it's not an input layer and it's not an output layer. So everything else is a hidden layer. Okay, and this particular hidden layer has is a size two on this dimension right because it has two filters right, two kernels, so what happens next? Well, let's do another one! Okay, so, as we kind of go along, things can multiply a little bit in complexity right because my next filter is going to have to contain two of these three by threes, because I'm gon na have to say how do I want to bring Adam? I want to write these three things and at the same time, how do I want to wait the corresponding three things down here right because in pytorch, this is going to be.

This whole thing here is going to be stored as a multi-dimensional tensor right. So you shouldn't really think of this now as two 3x3 kernels, but one two by three by three eternal okay, so to calculate this value here, I've got the sum product of all of that plus the sum product of scroll down all of that. Okay, and so the top ones are being multiplied by this part of the kernel, and the bottom ones have been multiplied by this part of the kernel and so over time. You want to start to get very comfortable with the idea of these, like higher dimensional

linear combinations right like it's it's harder to draw it on the screen, like I had to put one above the other, but conceptually just stuck it in your mind like this. That's really how you want to think right and actually Geoffrey Hinton in his original 2012 neural Nets. Coursera class has a tip, which is how all computer scientists deal with like very high dimensional spaces, which is that they basically just visualize the two-dimensional space and then say, like twelve dimensions, really fast, and they had lots of tires. So that's it right. We can see two dimensions on the screen and then you're just going to try to trust that you can have more dimensions like the Const. It's just you know, there's there's nothing different about them, and so you can see in Excel, you know Excel, doesn't have the ability to handle three-dimensional tensors. So I had to like say: okay, take this two-dimensional dot product.

Add on this two-dimensional dot product right, but if there was some kind of 3d Excel, I could have to stand that in a single line, all right and then again apply max 0, comma, otherwise known as rectified linear unit, otherwise known as value okay. So here is my second layer, and so when people create different architectures write, an architecture means like how big is your kernel at layer 1? How many filters are in your kernel at layer 1, so here I've got a 3 by 3 where's number 1 and a 3 by 3 there's number 2. So like this architecture, I've created starts off with 2 3 by 3 convolutional kernels and then my second layer has another two kernels of size: 2 by 3 by 3. So there's the first one and then down here, here's a second 2 by 3 by 3 kernel. Okay, and so remember, one of these specific any one of these numbers is an activation okay. So this activation is being calculated from these three things here and other 3 things up there and we're using these this 2 by 3 by 3 kernel, okay, and so what tends to happen? Is people generally give names to their layers? So I say: okay, let's call this layer here, con 1 and this layer here and this and this layer here con all right. So that's you know, but generally you'll just see that, like when you print out a summary of a network, every layer will have some kind of name. Okay, and so then what happens next well part of the architecture is like.

Do you have some max pooling where bounces up Matt spalling happen, so in this architecture, we're inventing we're going to next step is do max fully. Okay, Matt spooling is a little hard to kind of show in Excel, but we've got it so max pooling. If I do a two by two max pooling it's going to have the resolution both height and width, so you can see here that I've replaced these four numbers with the maximum of those four numbers right. And so because I'm having the resolution, it only makes sense to actually have something: every two cells - okay, so you can see here the way I've got kind of the same looking shape as I had back here: okay, but it's now half the resolution so for placed Every two by two with its max and you'll notice, like it's not every possible two by two I skip over from here. So this is like starting at beat Hugh and then the next one starts at BS right, so they're like non-overlapping. That's why it's decreasing the resolution? Okay, so anybody who's comfortable with spreadsheets. You know you can open this and have a look, and so after our max pooling, there's a number of different things we could do next and I'm going to show you a kind of classic old style approach nowadays, in fact, what generally happens nowadays is we do A max pool where we kind of like max across the entire size right but on older architectures and also on all the structured data stuff. We do.

We actually do something called a fully connected layer and so here's a fully connected layer. I'm going to take every single one of these activations and I've got to give every single one of them or weight right, and so then, I'm going to take over here here is the sum product of every one of the activations by every one of the weights. For both of the two levels of my three-dimensional tensor right - and so this is called a fully connected layer notice, it's

different to a convolution - I'm not going through a few at a time right, but I'm creating a really big weight matrix right so rather than having A couple of little 3x3 kernels, my weight matrix, is now as big as the entire input, and so, as you can imagine, architectures that make heavy use of fully convolutional layers can have a lot of weights, which means they can have trouble with overfitting, and they can Also be slow and so you're going to see a lot, an architecture called vgg because it was the first kind of successful, deeper architecture. It has up to 19 layers and vgg actually contains a fully connected layer with 4096 weights connected to a hidden layer with 4,000. Sorry, 4096 activations connected to a hidden layer with 4096 activations, so you've got like 4096 by 4096 x, remember or apply it by the number of kind of kernels that we've calculated so in vgg.

There's this, I think it's like 300 million weights, of which something like 250 million of them, are in these fully connected layers, so we'll learn later on in the course about how we can kind of avoid using these big, fully connected layers and behind the scenes. All the stuff that you've seen us using like ResNet and res next, none of them use very large, fully connected layers. You know you had a question. Sorry yeah come on um. So could you tell us more about, for example, if we had like three channels for the input, what would be the shape yeah these filters right? So that's a great question. So if we have 3 channels of input, it would look exactly like conv one right cons. One kind of has two channels right, and so you can see with cons one. We had two channels so therefore our filters had to have like two channels per filter, and so you could like imagine that this input didn't exist. You know - and actually this was the airport alright. So when you have a multi-channel input, it just means that your filters look like this and so images often full color. They have three red, green and blue. Sometimes they also have an alpha Channel. So, however, many you have that's how many inputs you need - and so something which I know Jeanette was playing with recently, was like using a full color image net model in medical imaging for something called bone age calculations which has a single channel and so what she Did was basically take the the input, the the single channel input and make three copies of it.

So you end up with basically like one two three versions of the same thing, which is like it's kind of a small idea like it's kind of redundant information that we don't quite want. But it does mean that then, if you had a something that expected a three channel, convolutional filter, you can use it right and so at the moment, there's a cable competition for iceberg detection using a some funky satellite specific data format that has two channels. So here's how you could do that you could either copy one of those two channels into the third channel or I think what people in Carroll are doing is to take the average of the two again. It's not ideal, but it's a way that you can use. Pre-Trained networks - yeah I've done a lot of fiddling around like that. You can also actually I've actually done things where I wanted to use a three channel image net Network on four channel data. I had a satellite data where the fourth channel was near-infrared, and so basically, I added an extra kind of level to my convolutional kernels that were all zeros and so basically like started off by ignoring the near-infrared band, and so what happens? It basically and you'll see this next week is that, rather than having these like carefully trained filters, when you're actually training something from scratch, we're actually going to start with random numbers. That's actually what we do.

We actually start with random numbers, and then we use this thing called stochastic gradient descent, which we've kind of seen conceptually to slightly improve those random numbers.

14. 01:08:30

- **ConvNet demo with Excel (continued)**
- **output, probabilities adding to 1, activation function, Softmax**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

To make them less random and we basically do that again and again and again: okay, great, let's take a seven minute break and we'll come back at 7:50, all right! So what happens next, so we've got as far as doing a fully connected layer right. So we had our the results of our max pooling layer got fed to a fully connected layer and he might notice those of you that remember your linear algebra, the fully connected layer is actually doing a classic traditional matrix product. Okay. So it's basically just going through each pair in turn, multiplying them together and then adding them up to do a matrix product now in practice. If we want to calculate which one of the ten digits we're looking at their single number, we've calculated isn't enough, we would actually calculate ten numbers, so what we will have is, rather than just having one set of fully connected weights like this and I say, set Because remember, there's like a whole 3d kind of tensor of them, we would actually need ten of those right, so you can see that these tensors start to get a little bit high, dimensional right, and so this is where my patients we're doing it. Next cell ran out, but imagine that I had done this ten times. I could now have ten different numbers or being calculated yeah using exactly the same process. Right we'll just be ten of these fully connected to by m-by-n erased. Basically, and so then we would have ten numbers being spat out.

So what happens next so next up we can open up a different Excel worksheet entropy example: dot XLS that's got two different worksheets. One of them is called soft mass and what happens here? Sorry. I've changed domains rather than predicting whether it's the number from one not to nine, I'm going to predict whether something is a cat, a dog, a plane of Fisher Building. Okay, so out of our that fully connected layer, we've got this case. We'd have five numbers and notice at this point. There's no rail, you! Okay! In the last layer, there's no rail, you, okay, so I can have negatives. So I want to turn these five numbers H into a probability. I want to turn it into a probability from naught to one that it's a cat, that's a dog, there's a plane that it's a fish that it's a building, and I want those probabilities to have a couple of characteristics. First is that each of them should be between zero and one, and the second is that this state together should add up to one right. It's definitely one of these five things. Okay, so to do that, we use a different kind of activation function. What's an activation function, an activation function is a function that is applied to activations so, for example, max 0 comma. Something is a function that I applied to an activation, so an activation function always takes in one number and spits out one number so max of 0. Comma X takes in a number X and spits out some different number value of s.

That's all an activation function is, and if you remember back to that PowerPoint we saw in Lesson one. Each of our layers was just a linear function and then, after every layer we said we needed some non-linearity act as if you stack a bunch of linear layers together right then all you end up with is a linear layer. Okay, so somebody's talking can. Can you not a slow just acting? Thank you if you stack a number of linear functions together, you just end up with a linear function and nobody does any cool, deep learning with displaying your functions right, but remember. We also learnt that by stacking linear functions with between each one, a non-linearity we could create like arbitrarily complex shapes, and so the non-linearity that we're using after every hidden layer is a rally rectified linear unit. A non-linearity is an activation function. An activation function is a non-linearity in with in deep way, obviously there's lots

of other nonlinearities and in the world, but in deep learning. This is what we mean, so an activation function is any function that takes some activation in as a single number and spits out. Some new activation like max of 0 comma, so I'm now going to tell you about a different activation function. It's slightly more complicated than value, but not too much. It's called soft max soft max only ever occurs in the final layer at the very end and the reason.

Why is that soft max always spits out numbers as an activation function that always spits out a number between Norton 1 and it always spits out a bunch of numbers that add to 1. So a soft max gives us what we want right in theory, this isn't strictly necessary right, like we could ask our neural net to learn a set of kernels which have you know which which give probabilities that line up as closely as possible with what we want, But in general, with deep learning, if you can construct your architecture so that the desired characteristics are as easy to express as possible, you'll end up with better models like they'll, learn more quickly with less parameters. So in this case, we know that our probabilities should end up being between 9 1. We know that they should end up adding to 1. So if we construct an activation function, which always has those features, then we're going to make our neural network do a better job, it's gon na make it easier for it. It doesn't have to learn to do those things because it all happen automatically. Okay,

15. 01:15:30

- **The mathematics you really need to understand for Deep Learning**
- **Exponentiation & Logarithm**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

So, in order to make this work, we first of all have to get rid of all of the negatives right like we can't have negative probabilities so to make things not being negative. One way we could do. It is just go into the pair of right. So here you can see. My first step is to go X of the previous one right, and I think I've mentioned this before, but of all the math that you just need to be super familiar with to do deep learning. The one you really need is logarithms and asks write, all of deep learning and all of machine learning. They appear all the time right. So, for example, you absolutely need to know that log of x times y equals log of X, plus log of Y, all right and like not just know that that's a formula that exists but have a sense of like what does that mean? Why is that interesting? Oh, I can turn multiplications into additions that could be really handy right and therefore log of x over y equals log of X minus log of Y. Again, that's going to come in pretty handy, you know, rather than dividing. I can just subtract things right and also remember that if I've got log of x equals y, then that means a to the y equals x, in other words, log log and E to the for the inverse of each other. Okay, again, you just you need to really really understand these things and like so, if you, if you haven't spent much time with logs and X, for a while, try plotting them in Excel or a notebook, have a sense of what shape they are, how they combine Together just make sure you're really comfortable with them, so we're using it here right, we're using it here.

So one of the things that we know is a to the power of something is positive. Okay, so that's great the other thing, you'll notice, about e to the power of something is because it's a power numbers that are slightly bigger than other numbers, like four, is a little bit bigger than 2.8. When you go e to the power of it really accentuates that difference, okay, so we're going to take advantage of both of these features for the purpose of deep learning. Okay, so we take

our the results of this fully connected layer. We go e to the power of for each of them and then we're gon na yeah and then we're going to add them up. Okay, so here is the sum of e to the power of so then here we're going to take e to the power of divided by the sum of e to the power of so, if you take all of these things divided by their sum, then by definition, All of those things must add up to 1 and furthermore, since we're dividing by their sum, they must always vary between 0 and 1 because they were always positive. Alright and that's it. So that's what softmax is ok, so I've got this kind of doing random numbers. Each time right - and so you can see like as I as I look through my softmax generally - has quite a few things that are so close to zero that they round down to zero. And you know, maybe one thing: that's nearly 1 right and the reason for that is what we just talked about, that is, with the X just having one number a bit bigger than the others tends to like push it out further right.

So, even though my inputs, here around on numbers between negative 5 and 5 right, my outputs from the softmax, don't really look that random at all, in the sense that they tend to have one big number and a bunch of small numbers. And now that's what we want right. We want to say like in terms of like. Is this a cat, a dog, a plane, a fish or a building? We really want it to say like it's, it's that you know it's. It's a dog or it's a plane not like I don't know. Okay, so softmax has lots of these cool properties right. It's going to return a probability that adds up to 1 and it's going to tend to want to pick one thing particularly strongly. Okay. So that's soft mess your net. Could you pass actually bust me up? We? How would we do something that, as let's say, you have any

16. 01:20:30

- **Multi-label classification with Amazon Satellite competition**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Imaging you want to count in categorize, I was like cat and the dog or like has multiple things, but what kind of function will we try to use so happens. We're going to do that right now, so so hope you think about why we might want to do that and so runways, where you might want to do, that is to do multi-label classification, so we're looking now at listen to image models and specifically we're going to Take a look at the planet: competition, satellite, imaging competition. Now the satellite imaging competition has some similarities to stuff we've seen before right. So before we've seen cat versus dog and these images are a cat or a dog they're - not Maya, they're, not both right, but the satellite imaging competition has stayed as images that look like this and in fact, every single one of the images is classified by whether There's four kinds of weather, one of which is haze and another of which is clear. In addition to which there is a list of features that may be present, including agriculture, which is like some some cleared area used for agriculture, primary, which means primary rainforest and water, which means a river or a creek, so here is a clear day satellite image, showing Some agriculture, some primary rainforest and some water features and here's one which is in haze and is entirely primary rainforest.

So in this case, we're going to want to be able to show we're going to predict multiple things, and so softmax wouldn't be good, because softmax doesn't like predicting multiple things and like. I would definitely recommend anthropomorphizing your activation functions right. They have personalities, okay and the personality of the softmax. Is it wants to pick a thing and people forget this all the time I've seen many people even well-regarded researchers in

famous academic papers, using like soft maps for multi-label classification. It happens all the time right and it's kind of ridiculous, because they're not understanding the personality of their activation function. So for multi classification, where each sample can belong to one or more classes. We have to change a few things, but here's the good news in fastai. We don't have to change anything right. So fastai will look at the labels in the CSV and if there is more than one label ever for any item, it will automatically switch into like multi-label mode. So I'm going to show you how it works behind the scenes, but the good news is you: don't actually have to care, it happens anywhere. So if you have multi label images, multi label objects. You obviously can't use the classic Kerris style approach where things are in folders, because something can't conveniently be in multiple folders at the same time. Right, so that's why we, you basically have to use the from CSV approach right.

So if we look at an example actually I'll show you, I tend to take you through it right, so we can say: okay, this is the CSV file containing our labels. This looks exactly the same as I did before, but rather than side on its top down alright and top down I've mentioned before that can do our vertical flips. It actually does more than that. There's actually eight possible symmetries for a square, which is, it can be rotated through 90, 180, 270 or 0 degrees, and for each of those it can be flipped. And if you think about it, for awhile you'll realize that that's a complete enumeration of everything that you can do in terms of symmetries to a square, so they're called it's called the dihedral group of eight. So if you see in the code there's actually a transform or dihedral, that's why it's called that, so this transforms will basically do the full set of eight symmetric, dihedral, rotations and flips, plus everything which we can do to dogs and cats. You know small clinical rotations. A little bit of zooming a little bit of contrast and brightness adjustment, so these images are a size 256 by 256. So I just create a little function here to let me quickly grab, you know: data loader of any size, so here's a 256 by 256 once you've got a data object inside it. We've already seen that there's things called Val D s test D s, train D s: there are things that you can just index into and grab a particular image.

So you just use square brackets. 0. You'll also see that all of those things have a DL. That's a data loader so des is data set. Dl is data motor. These are concepts from PI watch. So if you, Google, pytorch data set or pipe watch data loader, you can basically see what it means, but the basic idea is a data set. Gives you a single image or a single object back a data loader gives you back a mini batch and specifically, it gives you back a transformed mini. So that's why, when we create our data object, we can pass in num workers and transforms it's like how many processes do you want to use? What transforms do you want, and so, with with a data loader, you can't ask for an individual image. You can only get back at a mini batch and you can't get back a particular mini batch. You can only get back the next mini, so something reverses look through grabbing a mini batch at a time and so in Python. The thing that does that is called a generator right or an iterator. This slightly different versions are the same thing so to turn a data loader into an iterator. You use the standard Python function, cordetta, that's a Python function, just a regular part of the Python basic language that returns you an iterator and an iterator is something that takes. You can pass the static, give pass it to the standard, Python function or statement next, and that just says give me another batch from this iterator so we're. Basically, this is one of the things I really like about pytorch.

Is it really leverages modern, pythons kind of stuff you know in in tensorflow they invent their whole new world earth ways of doing things, and so it's kind of more in a sense, it's more like cross-platform, but in another sense like it's, not a good fit to Any platform, so it's nice, if you, if you know Python well, pytorch comes very naturally. If you don't know Python well, pytorches are good reason to learn Python. Well, a pytorch, your module neural

network module is a standard Python bus, for example. So any work you put into learning Python better will pay off with paid watch. So here I am using standard Python, iterators and next to grab my next mini batch from the validation sets data loader and that's going to return two things. It's going to return the images in the mini batch and the labels of the mini, so standard Python approach. I can pull them apart like so, and so here is one mini batch of labels, and so not surprisingly, since I said that my batch size, let's go ahead and find it Oh actually, it's the batch size by default is 64, so I didn't pass in a Batch size, and so just remember, shift tab to see like what are the things you can pass and what are the defaults so by default, my batch size is 64, so I've got that something of size 64 by 17. So there are 17 of the possible classes. Right so, let's take a look at the zeroth set of labels, so the zeroth images labels. So I can zip again standard Python things.

It takes two lists and combines it. So you get the zero theme from the first list as you're asking for the second list and the first thing for the first first, this first thing from the second list and so forth. So I can zip them together and that way I can find out for the zeroth image and the validation set is agriculture. It's clear, its primary rainforest, its slash-and-burn, its water? Okay. So, as you can see here, this is a MOLLE label. You see here's a way to do multi-label classification, so, by the same token, right, if we go back to our single label classification, it's a cat dog playing official building behind the scenes we haven't actually looked at it, but behind the scenes fastai imply torch are Turning our labels into something called one hot encoded labels, and so if it was actually a dog than the actual values would be like that right. So these are like the actuals okay. So do you remember at the very end of at AV, o --'s video? He showed how, like the template, had to match to one of the like five ABCDE templates, and so what it's actually doing is it's. Comparing when I said it's basically doing a dot product, it's actually a fully connected layer at the end right that calculates an output activation that goes through a soft Max and then the soft max is compared to the one hot encoded label right.

So if it was a dog, there would be a one here and then we take the difference between the actuals and the softmax activation is to say and add those add up those differences to say how much error is there essentially we're skipping over something called a Loss function that we'll learn about next week, but essentially we're basically doing that now, if it's one hot encoded like there's only one thing which have a 1 in it, then actually storing it as 0 1 0 0 0 is terribly inefficient. Right like we could basically say what are the index of each of these things right, so we can say it's like 0, 1, 2, 3, 4, like so right, and so, rather than storing it is 0. 1. 0. 0. 0. We actually just store the index value right. So if you look at the the Y values for the cats and dogs, competition or the dog breeds competition, you won't actually see a big lists of ones and zeros like this you'll see a single integer right, which is like what what class index is it right And internally inside pipe arch, it will actually turn that into a one hot encoded vector but, like you will literally never see it. Okay and and pytorch has different loss functions where you basically say this thing's won. This thing is one hot encoder door. This thing is not, and it uses different bus functions, that's all hidden by the faster I library right so like you, don't have to worry about it, but is.

But the the cool thing to realize is that this approach for multi-label encoding with these ones and zeros behind the scenes, the exact same thing happens for single level classification. Does it make sense to change the beginners of the sigmoid of the softmax function by changing the base? No because when you change the more math log base, a of B equals log B over log a so changing the base is just a linear scaling and linear scaling is something which the neural net can with very easily

17. 01:33:35

- **Example of improving a “washed-out” image**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Good question: okay, so here is that image right here is the image with slash-and-burn water, etc, etc. One of the things to notice here is like when I first displayed this image, it was so washed out. I really couldn't see it right, but remember images now. You know we know, images are just matrices of numbers, and so you can see here, I just said times 1.4 just to make it more visible right so, like now that you kind of it's, the kind of thing I want you to get familiar with is the Idea that this stuff you're dealing with they're just matrices of numbers and you can fiddle around with them, so if you're looking at something like guys a bit washed out, you can just multiply it by something to brighten it up a bit okay. So here we can see. I guess this is the slash-and-burn. Here's, the river, that's the water, here's, the primary rainforest, maybe that's the agriculture so forth. Okay! So so you know with all that background, how do we actually use this exactly the same way as everything we've done before right? So you know size and - and the interesting thing about playing around with this planet competition - is that these images are not at all like image there, and I would guess that the vast majority is of stuff that the vast majority of you do, involving convolutional neural Nets. Won't actually be anything like image net, you know, it'll be it'll, be medical, imaging it'll, be like classifying different kinds of steel, tube or figuring out whether a world you know is going to break or not or or looking at satellite images, or you know whatever right.

So it's it's good to experiment with stuff like this planet, competition to get a sense of kind of what you want to do, and so you'll see here I start out by resizing my data to 64 by 64. It starts out at 256 by 256 right now. I wouldn't want to do this for the cats and dogs competition because it cats end on competition. We start with a pre trained imagenet Network. It's it's nearly isn't. It starts off nearly perfect right. So if we resized everything to 64 by 64 and then retrained the whole set regular, it we'd basically destroy the weights that are already pre trained to be very good. Remember, imagenet, most imagenet models are trained at either 224 by 224 or $ 2.99 by 299. All right, so, if we like retrain them at 64 by 64, we're going to we're going to kill it. On the other hand, there's nothing in image net. That looks anything like this. You know, there's no satellite images, so the only useful bits of the image net Network for us are kind of layers like this one. You know finding edges and gradients, and this one you know finding kind of textures and repeating patterns, and maybe these ones are kind of finding more complex textures, but that's probably about it right. So so, in other words, you know, starting out by training. Very small images works pretty well when you're using stuff like satellites. So in this case I started right back at 64 by 64 grab.

Some data built my model found out what learning rate to use and, interestingly, it turned out to be quite high. It seems that, because, like it's so unlike imagenet, I needed to do quite a bit more fitting with just that last layer before it started to flatten out

18. 01:37:30

- **Seting different learning rates for different layers**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to

improve - remove this note once proofread)

Then I unfreeze dit and again, this is the difference to image net like datasets is my learning rate in the initial layer. I set 2/9 the middle layers, I said 2/3, where else for stuff like it's like image net, I had a multiple of 10. Each of those you know again the idea being that that earlier layers, probably and not as close to what they need to be compared to the like dances again unfreeze train for a while, and you can kind of see here. You know there's cycle one there's cycle, there's cycle three and then I kind of increased double the size with my images fit for a while and freeze fit for a while double the size of the images again fit for a while. I'm freeze for a while and then add TTA, and so, as I mentioned last time, we looked at this. This process ends up. You knowgetting us about 30th place in this competition, which is really cool because people, you know a lot of very, very smart people. Just a few months ago worked very very hard on this competition. A couple of

19. 01:38:45

- **'data.resize()' for speed-up, and 'metrics=[f2]' or 'fbeta_score' metric**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Things people have asked about, one is: what is this data dot resize do so a couple of different pieces here. The first is that when we say back here, what transforms do we apply and here's our transforms? We actually pass in a size right. So one of the things that that one of the things that data loaded does is to resize the images like on-demand every time it sees them. It's got nothing to do with that dot, resize method right. So this is this is the thing that happens at the end like whatever's passed in before it hits out that before our data loader spits it out, it's going to resize it to this size. If the initial input is like a thousand by a thousand reading, that JPEG and resizing it to 64 by 64 turns out to actually take more time than training the content, that's for each batch all right! So basically all resize does, is it says: hey I'm not going to be using any images bigger than size times 1.3, so just grow through once and create new JPEGs of this size right and they're rectangular right so new JPEGs, where the smallest edges of this size And again, it's like you never have to do this, there's no reason to ever use it. If you don't want to it's just a speed-up okay, but if you've got really big images coming in it saves you a lot of time and you'll often see on like Carol kernels or forum posts or whatever people will have like bash script.

Stuff, like that, like loop through and resize images to save time, you never have to do that right. Just you can just say: dot, resize and it'll just create you know, once-off it'll go through and create that if it's already there and it'll use the criticized ones for you, okay, so it's just it's just a speed up. Convenience function, no more! Okay! So for those of you that are kind of past dog breeds, I would be looking at planet next. You know like try it like play around with with trying to get a sense of like how can you get this as an accurate model? One thing to mention, and I'm not really going to go into it in details - there's nothing to do with deep learning, particularly, is that I'm using a different metric. I didn't use metrics equals accuracy, but I said metrics equals f2. Remember from last week that confusion matrix that, like two by two, you know correct incorrect for each of dogs and cats. There's a lot of different ways. You could turn that confusion matrix into a score. You know: do you care more about false negatives, or do you care more about false positives and how do you weight them and how do you combine them together, right, there's, a basic there's! Basically, a function called F

beta where the beta says: how much do you weight, false negatives versus false positives and so f. 2 is f beta with beta equals 2, and it's basically, as particular way of weighting, false negatives and false positives and the reason we use it is because cattle told us that planet who are running this competition wanted to use this particular F beta metric.

The important thing for you to know is that you can create custom metrics. So in this case you can see here it says from Planet import, f2 and really I've got this here so that you can see how to do it right. So if you look inside courses, DL 1, you can see there's something called planet py right. And so, if I look at planet, py you'll see there's a function. There called f2 right and so f2 simply calls F beta score from psychic or side PI and patent. Where it came from and does a couple little tweets that are particularly important, but the important thing is like you can write any metric. You like right as long as it takes in set of predictions and a set of targets and they're both going to be numpy. Arrays one dimensional non pyros and then you return back a number okay, and so as long as you put a function that takes two vectors and returns at number, you can call it as a metric. And so then, when we said see here, learn, metrics equals and then past in that array, which just contains a single function. F2, then it's just going to be printed out after every for you, okay, so in general, like the the faster I library, everything is customizable. So kind of the idea is that everything is everything is kind of gives you what you might want by default, but also everything can be changed as well. Yes, you know, um.

We have a little confusion about the difference between multi-label and a single label. Uh-Huh the vanish as an example in which compared like similarly the example they just show us ah activation function, yeah. So so I'm so sorry, I said I'd do that. Then I didn't so. The activation the output activation function for a single label. Classification is softmax, but all the reasons that we talked today, but if we were trying to predict something that was like 0 0, 1, 1 0, then softmax would be a terrible choice because it's very hard to come up with something where both of these are high. In fact, it's impossible

20. 01:45:10

- **'sigmoid' activation for multi-label**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Because they have to add up to 1, so the closest they could be would be point 5. So for multi-label classification, our activation function is called sigmoid, ok and again, the faster library does this automatically for you. If it notices, you have a multi label problem, and it does that by checking your data tip to see if anything has more than one label applied to it, and so sigmoid is a function which is equal to it's basically, the same thing, except rather than we Never add up all of these X, but instead we just take this X when we say it's just equal to it, divided by one plus it, and so the nice thing about that is that now, like multiple things can be high at once, right and so generally, Then, if something is less than zero, its sigmoid is going to be less than 0.5. If it's greater than zero is signal, it's going to be greater than 0.5, and so the important thing to know about a sigmoid function is that its shape is something which asymptotes at the top to one and asymptotes drew asymptotes at the bottom to zero. And so, therefore, it's a good thing to model a probability with anybody who has done any logistic regression will be familiar with. This is what we do in logistic regression, so it kind of appears everywhere in machine learning and you'll see that kind of a sigmoid and a softmax

they're very close to each other conceptually.

But this is what we want is our activation function for multi-label, and this is what we want: the single label and again, fastai. Does it all for you? There was a question over here. Yes, I

21. 01:47:30

- **Question on "Training only the last layers, not the initial freeze/frozen ones from ImageNet models"**
- **'learn.unfreeze()' advanced discussion**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Have a question about the initial training that you do? If I understand correctly, you have we have frozen the the premium model and you only need initially try to train the latest playwright right, but from the other hand we said that only the initial layer. So, let's last probably, the first layer is like important to us and the other two are more like features that are, you must not related, and we then apply in this case. What that they, the lie, is a very important, but the pre-trained weights in them aren't so it's the later layers that we really want to train the most so earlier layers likely to be like already closer to what we want. Okay, so you started with the latest one, and then you go right. So if you go back to our quick dogs and cats right when we create a model from pre train from a pre train model, it returns something where all of the convolutional layers are frozen and some randomly set fully connected layers. We add to the end, our unfrozen, and so when we go fit at first it just trains, the randomly set, a randomly initialized, fully connected letters right and if something is like really close to imagenet, that's often all we need, but because the early early layers are Already good at finding edges, gradients repeating patterns for ears and dogs heads you know, so then, when we unfreeze, we set the learning rates for the early layers to be really low, because we don't want to change the mesh for us the later ones we set them To be higher, where else for satellite data right, this is no longer true.

You know the early layers are still like better than the later layers, but we still probably need to change them quite a bit. So that's right. This learning rate is nine times smaller than the final learning rate, rather than a thousand times smaller. The final loan rate, okay, you play with with the weights of the layers yeah normally most of the stuff. You see online. If they talk about this at all, they'll talk about unfreezing, different subsets of layers, and indeed we do unfreeze our randomly generated runs. But what I found is, although the first layer library you can type learn dot, freeze too and just freeze a subset of layers. This approach of using differential learning rates seems to be like more flexible to the point that I never find myself. I'm freezing subsets of layers that I would expect you to start with that with a different cell, the different learning rates, rather than trying to learn the last layer. So the reason okay. So you could skip this training just the last layers and just go straight to differential learning rates, but you probably don't want to, and the reason you probably don't want to is that there's a difference. The convolutional layers all contain pre-trained weights, so they're like they're, not random, for things that are close to imagenet they're, actually really good for things that are not close to imagenet they're better than that. All of our fully connected layers, however, are totally random.

So therefore, you would always want to make the fully connected weights better than random

by training them a bit first, because otherwise, if you go straight to unfreeze, then you're actually going to be like fiddling around of those early early can early layer weights when the later Ones are still random. That's probably not what you want. I think that's another question here any possible. So when we unfreeze what are the things we're trying to change there? Will it change the Colonel's themselves that that's always what SGD does yeah? So the only thing? What training means is setting these numbers right and these numbers and these numbers the weights, so the weights are the weights of the fully connected layers and the weights in those kernels and the convolutions. So that's what training means it's and we'll learn about how to do it with SGD, but training literally is setting those numbers. These numbers, on the other hand, are activations, they're, calculated they're, calculated from the weights and the previous layers activations or amounts of questions. So you can lift it up higher and speak badly. So in your example of a cheerleader set of that English example. So you start with very small size existed for yeah, so does it literally mean you know? The model takes a small area from the entire image that is 64 bytes.

So how do we get that 64 by 64 depends on the transforms by default, our transform takes the smallest edge and recites the whole thing out samples it. So the smallest edge is societics. T4 and then it takes a Center crop of that. Okay, although when we're using data augmentation, it actually takes a randomly chosen prop ie the case where the image ties to multiple objects. Don't in this case like would it be possible that you would just lose the other things that they try to predict yeah, which is why data augmentation is important so by by and particularly their test time. Augmentation is going to be particularly important because you would you wouldn't want to. You know that there may be a artisanal mine out in the corner which, if you take a center crop, you you don't see so data augmentation becomes very important yeah. So when we talk on their tributaries, are he receiver up to that's, not really what a model choice? Delton, that's a great point. That's not the loss function, yeah right. The loss function is something we'll be learning about next week and it uses cross, entropy or otherwise known as like negative log likelihood. The metric is just this thing, that's printed, so we can see what's going on just next to that. So, in the context of my deep pass, modeling cannot change data.

Does it trading? It also have to be multiplied. So can I train on just like images of pure cats and dogs and expect it at prediction time to predict? If I give it a picture of both having cat eye on it over I've never tried that and I've never seen an example of something that needed it. I guess conceptually there's no reason it wouldn't work, but it's kind of out there and you still use a sigmoid. You would have to make sure you're using a sigmoid loss function. So, in this case, faster eyes default would not work because by default first day I would say your training data knitter has both a cat and the dog, so you would have to override the loss function when you use the differential learning rates. Those three learning rates: do they just kind of spread evenly across the layers. Yeah, we'll talk more about this later in the course, but I mean the faster I library there's a concept of layer groups so in something like a resonant 50. You know there's hundreds of layers and I think it you don't want to write down hundreds of learning rates, so I've basically decided for you how to split them and the the last one always refers just to the fully connected layers that we've randomly initialized and edit To the end, and then these ones are split generally about halfway through. Basically, I've tried to make it so that these you know these ones are kind of the ones which you hardly want to change at all, and these are the ones you might want to change.

A little bit - and I don't think we're covered in the course. But if you're interested we can

22. 01:56:30

- **Visualize your model with 'learn.summary()', shows 'OrderedDict()'**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Talk about in the forum, there are ways you can override this behavior to define your own layer groups. If you want to, and is there any way to visualize the model easily or like don't dump the layers of the model yeah, absolutely you can, let's make sure we've got one here. Okay, so if you just type learn, it doesn't tell you much at all, but what you can do is go learn summary and that spits out, basically everything, there's all the letters, and so you can see in this case. These are the names I mentioned. How they look up names right, so the first layer is called con 2 d 1, and it's going to take as input. This is useful to actually look at it's taking 64 by 64 images, which is what we told it we're going to transform things to. This is three channels. High torch, like most things, have channels at the end, would say 64 by 64 by 3, ply torch music to the front, so it's 3 by 64 by 64. That's because it turns out that some of the GPU computations run faster when it in that order. Okay, but that happens, all behind-the-scenes automatic plays a part of that transformation. Stuff, that's kind of all done automatically is to do that. Minus one means, however, however big the batch size is in care us, they use the number. They use a special number, none in pile types that used minus one. So this is a four dimensional mini batch. The number of elements in the amount of images in the mini batch is dynamic. You can change that.

The number of channels is three number, which is a 64 by 64, okay, and so then you can basically see that this particular convolutional kernel apparently has 64 kernels in it, and it's also having we haven't talked about this, but convolutions can have something called a stride That is like Matt pullin: it changes the size, so it's returning a 32 by 32, 564 kernel, tenza and so on and so forth. So that's summary and we'll learn all about that's doing in detail on in the second half of the course one where I clicked in my own data set and I try to use the in as a really small data set. These currencies from Google Images and I tried to do a learning rate, find and then the plot and it just it, gave me some numbers which I didn't understand and the learning rate font yeah and then the plot was empty. So yeah I mean let's, let's talk about that on the forum, but basically the learning rate finder is going to go through a mini batch at a time if you've got a tiny data set, there's just not enough mini batches, so the trick is to make your Mini bit make your batch size really small like try making it like four okay. They were great questions. It's not nothing online to add. You know they were great questions. We've got a little bit.

23. 01:59:45

- **Working with Structured Data "Corporacion Favorita Grocery Sales Forecasting"**
- **Based on the Rossman Stores competitition**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Past, where I hope to, but let's let's quickly talk about structured data, so we can start thinking about it for next week. So this is really weird right to me. There's basically, two types of data set. We use in machine learning, there's a type of data like audio images, natural language text, where all of the all of the things inside an object like all of the pixels inside an image are all

the same kind of thing. They're, all pixels or they're. All apertures of a waveform or they're all words. I call this kind of data unstructured and then there's data sets like a profit and loss statement or the information about a Facebook user where each column is like structurally quite different. You know. One thing is representing like how many page views last month: another one is their sex. Another one is what zip code they're in, and I call this structure there. That particular terminology is not unusual, like lots of people use that terminology, but lots of people, don't there's. No particularly agreed-upon terminology, so when I say structured data, I'm referring to kind of columnar data, as you might find in a database or a spreadsheet, where different columns represent different kinds of things, and each row represents an observation, and so structured data is probably what most Of you are analyzing most of the time.

Funnily enough, you know, academics in the deep learning world don't really give a about structured data, because it's pretty hard to get published in fancy conference proceedings if you're like if you've got a better logistics model. You know it's the thing that makes the world goes round, it's a thing that makes everybody you know and efficiency and make stuff work, but it's largely ignored. Sadly, so we're not going to ignore it because we're a practical, deep learning and cackled doesn't ignore it either because people put prize money up on Kaggle to solve real-world problems. So there are some great capable competitions. We can look at there's one running right now, which is the grocery sales forecasting competition for Ecuador's largest chain. It's always a little. I've got to be a little careful about how much I show you about currently running competitions, because I don't want to you know, help you cheat, but it so happens. There was a competition a year or two ago for one of Germany's magistrate chains, which is almost identical, so I'm going to show you how to do that, so that was called the Rossman stores data, and so I would suggest you know first of all, try practicing What we're learning on Russman right, but then see if you can get it working on on grocery, because currently on the leaderboard, no one seems to basically know what they're doing in the groceries competition.

If you look at the leaderboard, the let's see here, yeah these ones around 5 to 9 v 3o are people that are literally finding like group averages and submitting those. I know because they're kernels that they're using so you know the basically the people around 20th place are not actually doing any machine learning so yeah, let's see if we can improve things so you'll see, there's a less than 3 rossmann notebook sure you get pull. Ok, in fact, you know just reminder: you know before you start working, get pull in you're, faster, a repo and from time to time, Condor and update for you guys during the in-person course the Condor and update you should do it more often, because we're kind of Changing things a little bit, um folks in the MOOC you know more like once a month should be fine. So anyway, I just just changed this a little bit so make sure you get Paul to get lesson 3 Rossman and there's a couple of new libraries here. One is fast: AI dot structure, faster, guided structured, contain stuff, which is actually not at all high torch specific, and we actually use that in the machine learning course as well for doing random forests with no tie torch at all. I mentioned that because you can use that particular library without any of the other parts of fastai, so that can be handy and then we're also going to use faster column data, which is basically some stuff that allows us to do fast, a type a torch Stuff with columnar structured data for structured data, we need to use pandas a lot.

Anybody who's used. Our data frames will be very familiar with pandas. Pandas is basically an attempt to kind of replicate data friends in Python. You know, and a bit more, if you're not entirely familiar with pandas there's a great book, , which I think I might have mentioned

before for data analysis by Wes McKinney, there's a new addition that just came out a couple of weeks ago, obviously being By the pandas author, its

24. 02:05:30

- **Book: Python for Data Analysis, by Wes McKinney**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Coverage of pandas is excellent, but it also covers numpy, scipy, matplotlib, scikit-learn and Jupiter, really well, okay, and so I'm kind of going to assume that you know your way around these libraries to some extent also there was the workshop we did before they started and there's A video of that online, where we kind of have a brief mention of all of those tools. Structured data is generally shared as CSV files. It was no different in this competition as you'll see, there's a hyperlink to the rustman data set here right now. If you look at the bottom of my screen, you'll see this goes to file start faster day.i, because this doesn't require any login or anything to grab this data set it's as simple as right. Clicking copy link address head over to wherever you want it and just type wget and the URL okay. So that's because you know it's it's not behind a login or anything, so you can grab the grab it from there and you can always read a CSV file with just pandas. Don't read CSV now, in this particular case, there's a lot of pre-processing that we do and what I've actually done here is I've. I've actually stolen the entire pipeline from the third place, winner, roster, okay, so they made all their data they're really great. You know they better get hub available with everything that we need and I've ported it all across and simplified it and tried to make it pretty easy to understand.

This course is about deep learning, not about data processing, so I'm not going to go through it, but we will be going through it in the machine learning course in some detail because feature engineering is really important. So if you're interested, you know check out the machine learning course for that. I will, however, show you kind of what it looks like so once we read the CSV Xin, you can see basically what's there, so the key one is for a particular store. We have the we have the date and we have the sales for that particular store. We know whether that thing is on promo or not. We know the number of customers at that particular store had we know whether that date was a school holiday. We also know what kind of store it is, so this is pretty common right. You'll often get datasets where there's some column with like just some kind of code. We don't really know what the code means and most of the time I find it doesn't matter what it means like. Normally you get given a data dictionary when you start on a project and obviously, if you're working on an internal project, you can ask the people at your company. What does this column mean? I kind of stay away from learning too much about it. I prefer to like to see what the data says. First, there's something about what kind of product are we selling in this particular row and then there's information about like how far away is the nearest competitor? How long have they been open for? How long is the promo being on for for each store? We can find out what state it's in for each state we can find at the name of the state.

This is in Germany and, interestingly, they were allowed to download any data external data they wanted in this competition, just very common as long as you share it with everybody else, and so some folks tried downloading data from Google Trends. I'm not sure exactly what it was that they would check in the trend off, but we have this information from Google Trends. Somebody downloaded the weather for every day in Germany, every state and yeah.

That's about it right, so you can get a data frame summary with pandas, which kind of lets you see how many observations and means and standard deviations again. I don't do a hell of a lot with that early on, but it's nice to note there. So what we do you know this is called a relational data set. A relational data set is one where there's quite a few tables. We have to join together. It's very easy to do that in pandas. There's a thing called merge, so great little function to do that, and so I just started joining everything together: joining the weather or the Google Trends stores. Yeah. That's about everything I guess you'll see, there's one thing that I'm using from the FASTA, a library which is called add date part. We talked about this a lot in the machine learning course, but basically this is going to take a date and pull out of a bunch of columns day of week is at the start of a quarter month of year, so on and so forth and add them All in to the dataset okay, so this is all standard pre-processing all right, so we join everything together we fiddle around with some of the dates a little bit.

Some of them are in month in year format. We turn it into date, format. We spend a lot of time trying to take information about, for example, holidays and add a column for like how long until the next holiday. How long has it been since the last holiday ditto for promos so on and so forth? Okay, so we do all that and at the very end we

25. 02:11:50

- **We save the dataframe with 'Joined.to_feather()' from Pandas, use 'df = pd.read_feather()' to load.**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Basically save a big structured data file that contains all that stuff, something that those of you that use pandas may not be aware of is that there's a very cool new format called feather, which you can save a pandas data frame into this feather format. It's kind of pretty much takes it as it sits in RAM and dumps it to the disk, and so it's like really really really fast. The reason that you need to know this is because the ecuadorian grocery competition is on now has 350 million records, so you will care about how long things take a talk. I believe about six seconds for me to save three hundred and fifty million records to feather format. So that's pretty cool so at the end of all that I'd save it as a feather format and for the rest of this discussion. I'm just going to take it as given that we've got this nicely. Pre-Processed feature engineered file and I can just go read better okay, but for you to play along at home, you will have to run those previous cells. Oh, except the see these ones have commented out, you don't have to run those because the file that you download from files, doc bastard AI, has already done that for you, okay, all right, so we basically have all these columns, so it basically is going to tell Us you know how many of this thing was sold on this date at this store, and so the goal of this competition is to find out how many things will be sold for each store for

26. 02:13:30

- **Split Rossman columns in two types: categorical vs continuous**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Each type of thing in the future: okay, and so that's, basically, what we're going to be trying to do, and so here's an example of what some of the data looks like and so next week we're going to see how to go through these steps. But basically, what we're going to learn is we're going to learn to split the columns into two types. Some columns were going to treat as categorical, which is to say, store ID one and store ID, I'm not numerically related to each other they're categories, right we're going to treat day of week like that, Monday and Tuesday day, zero and day, one not numerically related to Each other, where else distance in kilometers to the nearest competitor, that's a number that we're going to treat numerically right. So, in other words, the categorical variables we basically are going to one how to encode them. You can think of it as one hot encoding on where the continuous variables we're going to be feeding into fully connected layers, just as is okay, so what we'll be doing is we'll be basically creating a validation set and you'll. See like a lot of these are start to look familiar. This is the same function. We used on planet and dog breeds to create a validation set, there's some stuff that you haven't seen before, where we're going to basically, rather than saying image, data dot from CSV, we're going to say, columnar data from data frame right.

So you can see like the basic API concepts, will be the same but they're a little different right, but just like before we're going to get a learner and we're going to go, lr find to find our best learning rate and then we're going to go dot. Fit with a metric with a cycle length, okay, so the basic sequence, who's going to end up, looking, hopefully very familiar, okay, so we're out of time. So what I suggest you do this week is like try to enter as many capital image competitions as possible like like try to really get this feel for, like cycling learning rates plotting things. You know that that post, I showed you at the start of class today that kind of took you through lesson, one like really go through that on as many image datasets as you can, you just feel really comfortable with it right because you want to get to The point where next week, when we start talking about structured data, that this idea of like how learners kind of work and data works and data loaders and data sets and looking at pictures should be really, you know intuitive all right. Good luck see you next week, [Applause,]

Lesson 4: Structured, time series, & language models

Outline

We complete our work from the previous lesson on tabular/structured, time-series data, and learn about how to avoid overfitting by using dropout regularization. We then introduce natural language processing with recurrent neural networks, and start work on a language model.

Video Timelines and Transcript

1. 00:00:04

- **More cool guides & posts made by Fast.ai classmates**
- **"Improving the way we work with learning rate", “Cyclical Learning Rate technique”,**
- **“Exploring Stochastic Gradient Descent with Restarts (SGDR)”, “Transfer Learning using differential learning rates”, “Getting Computers to see better than Humans”**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Okay, hi everybody! Welcome back. Let's see you all here, it's been another busy week of deep learning lots of cool things going on and like last week. I would have to highlight a few really interesting articles that some of some of you folks have have written. Vitaliy wrote one of the best articles I've seen for a while, I think, actually talking about differential learning rates and stochastic gradient descent with restarts be sure to check it out. If you can, because what he's done, I feel, like he's done a great job of kind of positioning, it a place that you can get a lot out of it. You know, regardless of your background, but for those who want to go further. He's also got links to like the academic papers. It came from and kind of rests of showing examples of all of all the things he's talking about, and I think it's a it's a particularly nicely done article so good kind of role model for technical communication. One of the things I've liked about you know seeing people post these post these articles during the week is the discussion on the forums have also been like really great. There's been a lot of a lot of people helping out like explaining things. You know, which you know, maybe those parts of the post period where people have said actually that's not quite how it works, and people have learnt new things that way.

People have come up with new ideas as a result as well. These discussions of stochastic gradient descent with restarts and cyclic or learning rates just being a few of them actually Anand sahar has written another great post talking about a similar, similar topic and why it works so well and again: lots of great pictures and references to papers And most importantly,

perhaps code showing how it actually works. Mark Hoffman covered the same topic at kind of a nice introductory level. I think really really kind of clear intuition. Many Cantor talk specifically about differential learning rates and why it's interesting and again providing some nice context to people not familiar with transfer learning you're, not going right back to saying like or what is transfer learning. Why is that interesting? And, given that why good differential learning rates be helpful and then one thing I particularly liked about arjen's article was that he talked not just about the technology that we're looking at, but also talked about some of the implications, particularly from a commercial point of view. So thinking about like based on some of the things we've learned about so far, what are some of the implications that that has you know in real life and lots of background lots of pictures and then discussing some of the yeah some of the implications.

So there's been lots of great stuff online and thanks to

2. 00:03:04

- **Where we go from here: Lesson 3 -> 4 -> 5**
- **Structured Data Deep Learning, Natural Language Processing (NLP), Recommendation Systems**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Everybody for all the great work that you've been doing as we talked about last week, if you're kind of vaguely wondering about writing something but you're feeling a bit intimidated about it, because you've never really written a technical post before just jump in you know it's it's! It's it's a really welcoming and encouraging group I think, to to work with so we're going to have a kind of an interesting lesson today, which is we're going to cover a whole lot of different applications. So we've we've spent quite a lot of time on computer vision and today we're going to try if we can to get through three totally different areas: structured learning. So looking at kind of how you look at so we're going to start out looking at structured learning or structured data learning, by which I mean building models on top of things, look more like database tables, so kind of columns of different types of data. There might be financial or geographical or whatever we're going to look at using deep learning for language, natural language processing and we're going to look at using deep learning for recommendation systems. And so we're going to cover these at a very high level. And the focus will be on here's: how to use the software to do it more. Then here is what's going on behind the scenes and then the next three lessons we'll be digging into the details of what's been going on behind the scenes, and also coming back to looking at a lot of the details of computer vision that we've kind of skipped Over so far, so the focus today is really on like how do you actually do these applications and we'll kind of talk briefly about some of the concepts involved before we do? I did want to talk about one key new concept, which is

3. 00:05:04

- **Dropout discussion with “Dog_Breeds”,**
- **looking at a sequential model’s layers with ‘learn’, Linear activation, ReLu, LogSoftmax**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to

Dropout - and you might have seen dropout mentioned a bunch of times already and got there got the impression that this is something important and indeed it is so look at dropout. I'm going to look at the dog breeds current cable competition. That's going on and what I've done is I've gone ahead and I've created a pre train network as per usual, and I've passed in pre compute equals true, and so that's going to pre compute, the activations that come out of the last convolutional layer remember an activation Is just a number it's a number just to remind you. An activation like here is one activation. It's a number, and specifically the activations, are calculated based on some weights, also called parameters that make up kernels or filters, and they get applied to the previous layers activations. But it could well be the inputs or they could themselves be the results of other calculations. Okay, so when we say activation, keep remembering we're talking about a number that's being calculated, so we've pre compute some activations and then what we do is we put on top of that a bunch of additional, initially randomly generated fully connected layers, so we're just going to Do some matrix multiplications on top of these, just like in our Excel worksheet at the very end, we had this matrix that we just did a matrix multiplication, but so what you can actually do is, if you just type the name of your loner object, you can Actually see what's in it, you can see the layers in it.

So when I was previously been skipping over a little bit about, are we add a few layers to the end? These are actually the layers of yet we're going to do batch norm. In the last lesson, so don't worry about that. For now a linear layer simply means a matrix multiply. Okay, so this is a matrix which has a 1024 rows and 512 columns, and so in other words, it's going to take in 1024 activations and spit out 512 activations. Then we have a rail unit which remember is just replace the negatives with 0 we'll skip over the batch norm. We'll come back drop out, then we have a second linear layer that takes those 512 activations from the previous linear layer and puts them through a new matrix multiply: 5 12 by 120. It spits out a new 120 activations and then finally put that through soft mats and for those of you that don't remember, softmax. We looked at that last year last week. It's this idea that we basically just take the the activation. Let's say the dog go e to the power of that and then divide that into the sum of e to the power of all the intermissions. So that was the thing that adds up to one all of them add up to one and each one individually is between 0 and 1. Ok, so that's that's what we added on top and that's the thing when we have pre computed calls. True, that's the thing we trained, so I wanted to talk about what this dropout is and what this key is, because it's a really important thing that we get to choose so a dropout layer with P equals 0.5 literally. Does this we go over to our spreadsheet and let's pick any layer with some activations and let's say: ok, I'm going to apply dropout with a P of 0.5 to con true what that means is I go through and with a 50 % chance. I pick a cell right pick an activation, so I kept like half of them randomly and I delete them.

Okay, that's that's! What dropout is right, so it's so the P equals 0.5 means. What's the probability of deleting that cell all right. So when I delete those cells, if you have a log like look at the output, it doesn't actually change by very much at all just a little bit, particularly because remember it's getting through a Mac spalling layer right. So it's only going to change it at all. If it was actually the maximum in that group of four and furthermore it's just one piece of you know if it's going into a convolution rather than into a max Paul, is just one piece of that that filter. So, interestingly, the idea of like randomly throwing away half of the activations in a layer has a really interesting result, and one important thing to mention is each mini batch. We throw away a different random half of activations earlier, and so what it

means is it forces it to not over fit right. In other words, if there's some particular activation, that's really learnt just that exact, that exact dog or that exact cat right then, when that gets dropped out the whole thing now isn't going to work as well. It's not going to recognize that image right, so it has to. In order for this to work, it has to try and find a representation that that actually continues to work even as random half of the activations get thrown away every time all right.

So it's a it's it's, I guess about four years old, now, three or four years old and it's been absolutely critical in making modern, deep learning work and the reason why is it really just about solve the problem of generalization for us before drop out came along? If you try to train a model with lots of parameters and you were overfitting and you already tried all the imitation you could and you already had as much data as you could you, there were some other things you could try, but to a large degree you Were kind of stuck okay, and so then Geoffrey Hinton and his colleagues came up with this. This dropout idea that was loosely inspired by the way the brain works and also loosely inspired by Geoffrey Hinton's experience in bank teller, Hugh's, apparently and yeah. Somehow they came up with this amazing idea of like hey: let's, let's try throwing things away at random, and so, as you could imagine, if your P was like point O one then you're throwing away 1 % of your activations for that layer at random. It's not gon na randomly change things up very much at all, so it's not really going to protect you from overfitting much at all. On the other hand, if your pain was 0.99, then that would be like going through the whole thing and throwing away nearly everything right and that would be very hard for it to overfit.

So that would be great for generalization, but it's also going to kill your accuracy, so this is kind of play off between high p-values generalized well, but will decrease your training accuracy and low p-values will generalize less well. That will give you a less good training accuracy. So for those of you that have been wondering, why is it that particularly early in training, my validation losses better than my training losses, but which seems otherwise, really surprising? Hopefully, some of you have been wondering why that is because on a data set that it never gets to see, you wouldn't expect the losses to ever be that's better and the reason why is because, when we look at the validation set, we turn off dropout right. So, in other words, when you're doing inference, when you're trying to say is this or cat, or is this a dog we certainly don't want to be, including random drop out there right we want to be using the best model. We can okay. So that's why? Early in training in particular, actually see that our validation, accuracy and loss tends to be better if we're using dropout. Okay. So yes, you know you have to do anything to accommodate for the fact that you are throwing away. Some that's a great question, so we don't that pytorch does so pytorch behind the scenes. Does two things: if you say P equals point five. It throws away half of the activations, but it also doubles all the activations that are already there.

So when average, the kind of the average activation doesn't change, which is pretty pretty neat trick so yeah you don't have to worry about it. Basically, it's it's done for you. So if we say so, you can pass in peas. This is the this. Is the p value for all of the added layers to say with fastai? What dropout do you want on each of the layers in these these added layers? It won't change. The dropout in the pre trained network, like the hope, is that that's already been pretty trained with some appropriate level of dropout. We don't change it. Put on these layers that we add you can say how much, and so you can see here as a T's equals 0.5, so my first dropout has 0.5. My second dropout has 0.5. I remember coming to the input of this was the output of the last convolutional layer of pre-trained network and we go over it and we actually throw away half of that before you can start go through our linear layer, throw away

the negatives throw away half the Result of that go through another linear layer and then pass it to our softness for minor numerical precision region reasons. It turns out to be better to take the log of the softmax, then softmax directly, and that's why you'll have noticed that when you actually get predictions out of our models, you always have to go npx both the predictions, but again the details as to. Why aren't important? So if we want to try removing dropout, we could go.

Peas, equals zero, all right and you'll see where else before we started with the point: seven six accuracy in the first epoch. Now you could have point eight accuracy in the first debug. Alright, so by not doing drop out, our first teapot worked better, not surprisingly, because we're not throwing anything away but by the third epoch. Here we had eighty four point eight, and here we have eighty four point one. So it started out better and ended up worse. So, even after three epochs, you can already see where master for overfitting right we've got point three loss on the train and point five loss on the validation yep. And so, if you look now, you can see in the resulting model. There's no drop out at all. So if the P is zero, we don't even add it to the model. Another thing to mention is you might have noticed that what we've been doing is we've been adding two linear layers right in our additional layers. You don't have to do that by the way. There's actually a parameter called extra fully connected layers that you can basically pass a list of. How long do you want or how big do you want, each of the additional fully connected layers to be, and so by default? Well, you need to have at least one right, because you need something that takes the output of the convolutional layer, which, in this case is of size thousand twenty-four and turns it into the number of classes you have.

Cats versus dogs would be two dog breeds would be 120 planet satellite, seventeen, whatever that's. You always need one linear layer, at least, and you can't pick how big that is, that's defined by your problem, but you can choose what the other size is or if it happens at all. So if we were to pass in an empty list - and now we're saying don't add any additional mini layers, just the one that we have to have right so here, we've got P - is equals zero extra fully connected layers is empty. This is like the minimum possible kind of top model we can put on top and again like if we do that, you can see above we actually end up with, in this case, a reasonably good result, because we're not training it for very long, and this particular Pre-Trained Network is really well suited to this particular problem yesterday so Jeremy. What kind of piece should we were using by default? So the one that's there by default for the

4. 00:18:04

- **Question: “What kind of ‘p’ to use for Dropout as default”, overfitting, underfitting, ‘xtra_fc=’**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

First layer is 0.25 and for the second layer is 0.5. That seems to work pretty well for most things right, so like it's, it's it, you don't necessarily need to change it at all. Basically, if you find it's overfitting just start bumping it up so try. First of all, setting it to 0.5 that'll set them both to 0.5. If it still overfitting a lot, try 0.7, like you, can you can narrow down and like it's, not that many numbers change right and if you're under fitting, then you can try and making it lower. It's unlikely. You would need to make it much lower because, like even in these dogs versus cats situations, you know we don't see they have to make it lower. So it's more likely to be increasing at about 0.6 0.7, but you can fiddle around. I find these the ones that are there

by defaults in work pretty well most of the time so one place I actually did increase. This was in the dog breeds one I did set it them both to 0.5, when I used a bigger model, so like ResNet 34 has less parameters, so it doesn't over fit as much. But then, when I started bumping pumping it up to like a resonate 50, which has a lot more parameters and noticed it started overfitting. So then I also increased my drop out. So as you use like bigger models, you'll often need to add more drama. Can you pass it over there, please, you know if we set B to 0.5, roughly what percentage is it 50 %? We say, RP, pasta? Is there a particular way in which you can determine if the data is being all fitted yeah? You can see that the like here, you can see that the training error is a loss, is much lower than the validation list.

You can't tell if it's like to over fitted like zero overfitting is not generally optimal, like the only way to find that out is remember. The only thing you're trying to do is to get this number low right, the validation loss number low. So, in the end, you kind of have to play around with a few different things and see which thing ends up getting the validation loss low, but you kind of get a feel overtime for your particular problem. What does overfitting? What does too much River fitting? Look like great so so that's dropout and we're going to be using that a lot and remember it's there by default service here. Another question: oh so I have two questions. So one is so when it says the dropout rate is 0.5. It does it like. You know I delete each cell with a probability of 0.5 or does it just pick 50 % randomly I mean I know both effectively. Is the 4-month yeah? Okay, okay, a second question is: why does the average activation matter? Well, it matters because the remember, if you look the Excel spreadsheet, that the result of this cell, for example, is equal to these nine multiplied by each of these nine right and add it up. So if we deleted half of these, then that would also cause this number to half, which would cause like everything else after that to change, and so, if you change what it means you know like you, then you're changing something that used to say, like Oh fluffy Ears are fluffy if this is greater than point six now, it's only fluffy if it's greater than point three like we're changing the meaning of everything, so you here is to delete things without changing. Where are you using a linear activation for one of the earlier activations? Why are we using when you yeah why that particular activation, because that's what this set of layers is so we've with the the pre-trained network is or is the convolutional net work and that's pre computed, so we don't see it.

So what that spits out is it's a vector, so the only choice we have is to use linear layers at this point. Okay, can we have different level of dropout by layer? Yes, absolutely how to do that great. So so you can absolutely have different dropout by layer and that's why this is actually called peas, so you could pass in an array here. So if I went 0, comma 0.2 for example, and then extra fully connecting it, I might add - 512 right, then that's going to be 0 drop out before the first of them and point to drop out before the second of them. Yes, requests, and I must admit I don't have a great intuition even after doing this, for a few years for like when should earlier or later layers have different amounts of dropping out. It's still something I kind of play with, and I can't quite find rules of thumb. So if some of you come up with some good rules of thumb, I'd love to hear about them. I think if in doubt you can use the same drop out and every fully connected layer. The other thing you can try is often

5. 00:23:45

- **Question: “Why monitor the Loss / LogLoss vs Accuracy”**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

People only put drop out on the very last linear layer, so there'd be the two things to try so Jeremy. Why do you monitor the log loss? The loss instead of the accuracy going up? Well, because the loss is the only thing that we can see for both the validation set in the training set, so it's nice to be able to compare them. Also, as we learn about later, the loss is the thing that we're actually optimizing. So it's it's kind of a little more. It's a little easier to monitor that and understand what that means. Can you pass it over there so with the drop out we're kind of adding some random noise every iteration right, you know. So that means that we don't do as much learning yeah that's right, so it doesn't seem to impact the learning rate enough thrive ever noticed it - I I would say you're, probably right in theory it might, but not enough that it's ever affected me. Okay. So let's talk about

6. 00:25:04

- **Looking at Structured and Time Series data with Rossmann Kaggle competition, categorical & continuous variables, '.astype('category')'**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

This structured data problem and so to remind you, we were looking at kegels rossmann competition, which is a German chain of supermarkets. I believe - and you can find this in lesson - three Russman and the main data set is the one where we were looking to say at a particular store. How much did they sell? Okay and there's a few big key piece of information? One is what was the date another was: were they open? Did they have a promotion on? Was it a holiday in that state and was it a holiday? As for school, a state holiday, there wasn't a school holiday yeah, and then we had some more information about stores like what for this store, what kind of stuff did they tend to sell? What kind of store are they how far away the competition and so forth? So, with the data set like this there's really two main kinds of column: there's columns that we think of as categorical. They have a number of levels, so the assortment column is categorical and it has levels such as a B and C. Where else something like competition distance, we will call continuous. It has a number attached to it where differences or ratios, even if that number have some kind of meaning, and so we need to deal with these two things quite differently. Okay, so anybody who's done any machine. Learning of any kind will be familiar with using continuous columns if you've done any linear regression.

For example, you can just like modify them by parameters, for instance categorical columns, we're going to have to think about a little bit more we're not going to go through the data cleaning, we're going to assume that that's a feature, Engineering we're going to assume all that's Been done, and so basically, at the end of that we have a list of columns and the in this case I didn't do any of the thinking around the feature, engineering or dedicating myself. This is all directly from the third-place winners of this competition, and so they came up with all of these different columns that they found useful and so you'll notice. The list here is a list of the things that we're going to treat as categorical variables numbers like year, a month and day, although we could treat them as continuous like they, the different you know, differences between 2000 and 2003 is meaningful. We don't have to right and you'll see shortly how how categorical variables are treated, but basically, if we decide to make something, a categorical variable, what we're telling our neural net down the track is that for every different level of say year, you know 2000. 2001. 2002. You can treat it totally differently where else, if we say it's continuous, its have to come up with some kind of like function, some kind of smooth ish

function right and so often, even for things like a year that actually are continuous, but they don't actually have Many distinct levels: it often works better to treat it as categorical, so another good example, day of week, right so like day of week between naught & amp 6, it's a number and it means something motifs between 3 & amp, 5 is two days and has meaning, But if you think about like how word sales in a strawberry buy a day of week, it could well be that, like you know, Saturdays and Sundays, are over here, and Fridays are over here and Wednesdays. Are over here, like each day is going to behave.

Kind of qualitatively differently right so by saying this is the categorical variable as you'll see we're going to let the neural-net do that right. So this thing where we get where we say which are continuous in which a categorical to some extent, this is the modeling decision. You get to make now if something is coded in your data is like a B and C, or you know, Jeremy, and you knit or whatever you actually you're going to have to call that categorical right. There's no way to treat that directly as a continuous variable. On the other hand, if it starts out as a continuous variable like age or day of week, you get to decide whether you want to treat it as continuous or categorical. Okay, so summarize, if it's categorical and data it's going to have to be categorical in the model, if it's continuous in the data, you get to pick whether to make it continuous or categorical in the model. So in this case again, what I just did, whatever the third-place winners of this competition did, these are the ones that they decided to use as categorical. These were the ones they decided to use as continuous, and you can see that basically, the continuous ones are all of the ones which are actual floating-point numbers like competition. Distance actually has a decimal place to it.

Right and temperature actually has a decimal place to it. So these would be very hard to make categorical because they have many many levels right like if it's like five digits of floating-point, then potentially there will be as many levels as there are, as there are roads and by the way, the word we use to say How many levels are in a category we use the word cardinality right. So if you see me say cardinality example, the cardinality of the day of week variable is 7 because there are 7 different days of the week. Do you have a heuristic for one to have been continuous variables, or do you ever in variables? I don't ever been continuous variables, so yeah. So one thing we could do with like max temperature is group it into nought to 10 10 to 20 20 to 30 and then call that categorical. Interestingly, a paper just came out last week in which a group of researchers found that sometimes bidding can be helpful, but it literally came out in the last week and until that time I haven't seen anything in deep learning. Saying that so I haven't. I haven't looked at it myself until this week. I would have said it's a bad idea now. I have to think differently. I guess maybe it is sometimes so if you're using year as a category, what happens when you run the model of a year? It's never seen so your training will get there yeah.

The short answer is: it will be treated as an unknown category and so pandas, which is the underlying data frame. Thinking we're using with categories as a special category called unknown and if it stays a category it hasn't seen before it gets treated as unknown so for AB deep learning model unknown will just be another category. If our data set training, the data set, doesn't have a category and test has unknown. How will it did you know just paper, this unknown category? It's still predict, it will predict something right like it will just have the value 0 barn scenes and if there's been any unknowns of any kind in the training set, then it off learnt a way to predict unknown. If it hasn't it's going to have some random vector - and so that's a interesting detail around training that we probably want to talk about in this part of the course, but we can certainly talk about on the forum. Okay, so we've got our categorical and continuous variable lists defined. In this case there was eight hundred thousand rows, so

eight hundred thousand dates basically by Storz, and so you can now take all of these columns. Look through each one and replace it in the data frame where the version where you say, take it and change its type to category okay, and so that just that just a pandas things. So I'm not going to teach you pandas, there's plenty of books so particularly with McKinney's books.

Book on python for data analysis is great, but hopefully it's intuitive as to what's going on, even if you haven't seen the specific syntax before so we're going to turn that column into a categorical column and then for the continuous variables we're going to make them all. 32-Bit floating-point and for the reason for that is that pipe torch expects everything to be 32-bit, floating-point. Okay, so, like some of these include like 1 0 things like, I can't see them straight away, but anyway, so much yeah like was there. A promo was, was a holiday and so that'll become the floating point: values 1 and 0, for instance. Ok, so I try to do as much of my work as possible on small data sets for when I'm working with images that generally means resizing the images to like 64 by 64 or 128 by 128. We can't do that with structured data, so instead I tend to take a sample. So I randomly pick a few rows, so I start running with a sample and I can use exactly the same thing that we've seen before for getting a validation set. We can use the same way to get some random random row numbers to use in a random sample okay, so this is just a bunch of random numbers and then okay, so that's going to be a size 150,000 rather than 800 40,000, and so my data that Before I go any further, it basically looks like this. You can see I've got some boolean x' here. I've got some integers here of various different scales. Here's my year 2014 and I've got some letters here.

So, even though I said, please call that a pandas category pandas still displays that in the notebook as strings right, it's just stored in internally differently. So then

7. 00:35:50

- **fastai library 'proc_df()', 'yl = np.log(y)', missing values, 'train_ratio', 'val_idx'. "How (and why) to create a good validation set" post by Rachel**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

The first day, our library has a special little function, called processed data frame and process data frame takes a data frame, and you tell it what's my dependent variable right and it does a few different things. The first thing is, it's pulled out that dependent variable and puts it into a separate variable, okay and deletes it from the original data frame. So DF now does not have the sales column in where else Y just contains a sales column. Something else that it does is it does scaling, so neural nets really like to have the input data to all be somewhere around zero, with a standard deviation of somewhere around one all right. So we can always take our data and subtract the mean and divide by the standard deviation to make that happen. So that's what do see a littles. True, that's and it actually returns a special object which keeps track of what mean and standard deviation did it use for that normalizing. So you can then do the same thing to the test set later. It also handles missing values, so missing values and categorical variables just become the ID 0 and then all the other categories become 1. 2. 3. 4. 5. 4, that categorical variable for continuous variables. It replaces the missing value with the median and creates a new column. That's a boolean and just says: is this missing or not, and I'm gon na skip over this pretty quickly because we talked about this in detail the machine learning course? Okay. So if you've got any questions about this part, that would be a good

place to go.

It's nothing deep learning specific there, so you can see afterwards year 2014, for example, has become year, two okay, because these categorical variables have all been replaced with with contiguous integers starting at zero, and the reason for that is later on we're going to be putting them Into a matrix right - and so we wouldn't want the matrix to be 2014 rows long when it could just be two rows one there. So that's the basic idea there and you'll see that the AC, for example, has been replaced in the same way with one and three okay. So we now have a data frame which does not contain the dependent variable and where everything is a number okay. And so that's that that's where we need to get to to do deep learning and all of the stage about that. As I said, we talked about in detail in the machine learning course nothing deep learning specific about any of it. This is exactly what we throw into our random forests as well. So another thing we talk about a lot in the machine learning core, of course, is validation, sets in this case. We need to predict the next two weeks of sales right. It's not like pick a random set of sales, but we have to pick the next two weeks of sales. That was what the cattle competition folks told us to do, and therefore I'm going to create a validation set, which is the last two weeks of my training, set right to try and make it as similar to the test set as possible, and we just posted actually Rachel wrote this thing last week about creating validation sets.

So if you go too fast at AI, you can check it out, we'll put that in the lesson wiki as well, but it's basically a summary of a recent machine learning lesson that we did. The videos are available for that as well, and this is kind of a written, a written summary of it, okay, so yeah so Rachel, and I spend a lot of time thinking about kind of you know. How do you need to think about validation, sets and training sets and test sets and so forth, and that's all there, but again

8. 00:39:45

- **RMSPE: Root Mean Square Percentage Error,**
- **create ModelData object, 'md = ColumnarModelData.from_data_frame()'**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Nothing deep learning specific, so, let's get straight to the deep learning action. Okay, so in this particular competition, as always with any competition or any kind of machine learning project, you really need to make sure you have a strong understanding of your metric. How are you going to be judged here, and in this case you know Carol makes it easy. They tell us how we're going to be judged, and so we're going to be judged on the roots mean squared percentage error right, so we're gon na say like. Oh, you predicted three. It was actually three point three, so you were can sent out and then we're gon na average, all those percents right and remember. I warned you that you are gon na need to make sure you know logarithms really well right, and so in this case from you know, we're basically being saying your prediction divided by the actual. The mean of that right is the thing that we care about, and so we don't have a metric in play. Torch called root mean squared percent error. We could actually easily create it by the way. If you look at the source code, you'll see like it's, you know a line of code, but easiest deal would be to realize that that, if you have that right, then you could replace a with like log of a dash and be with like log of B Dash and then you can replace that whole thing with a subtraction, that's just the rule of loaves right, and so, if you

don't know that rule then don't make sure you go look it up, because it's super helpful, but it means in this case all we need To do is to take the log of our data, which I actually did earlier in this notebook, and when you take the log of the data, getting the root mean squared error will actually get you.

There means great percent error for free okay, but then, when we want to like print out our it means percent error, we actually have to go e ^ it again right and then we can actually return the percent difference. So that's all that's going on here. It's again not really deep learning specific at all, so here we finally get to the deep learning alright. So, as per usual, like you'll, see everything we look at today looks exactly the same as everything we've looked at so far, which is first, we create a model data object, something that has a validation, set, training set and optional test set built into it. From that we will get a learner, we will then optionally called learner. Dot LR find real, then called learner, dot, fetch it'll, be all the same parameters and everything that you've seen many times before. Okay, so the difference, though, is obviously we're not going to go image. Classify a data dot from CSV or dot from paths, we need to get some different kind of model data and so for stuff, that is in rows and columns. We use columnar model data, but this will return an object with basically the same API that you're familiar with, and rather than from paths or from CSV. This is from data frame. Okay, so this gets passed a few things. The path here is just used for it to know where, should it store like model files or stuff like that right? This is just basically saying: where do you want to store anything that you saved later? This is the list of the indexes of the rows that we want to put in the validation set we created earlier.

Here's our data frame, okay and then look here's. This is where we did the log right. So I took the the Y that came out of property F, our dependent variable. I logged it and I call that yl all right. So we tell it when we create our model data, we need to tell it that's our dependent variable. Okay. So so far, we've got most of the stuff from the validation set, which is what's our independent variables, how dependent variables and then we have to tell it which things do we want traded as categorical right because remember by this time, everything's a number right. So it could do the whole things it's continuous. It would just be totally meaningless right, so we need to tell it which things do we want to treat as categories, and so here we just pass in that list of names that we used before. Okay and then a bunch of the parameters are the same as the ones you're used to. For example, you can set the batch size yeah. So after we do that, we've got a little standard model. Data object, but there's a trained DL attribute. There's a Val DL attribute a trained es attribute of LDS attribute. It's got a length, it's got all the stuff exactly like it did in all of our image. Based data objects. Okay, so now we need to create the the model or create the learner and so to skip ahead a little bit. We're basically going to pass in something that looks pretty familiar.

We're going to be passing thing from our model from our model data create a learner that is suitable for it and will basically be passing in a few other bits of information which will include how much dropout to use at the very start. How many? How many activations

9. 00:45:30

- **'md.get_learner(emb_szs,…)', embeddings**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

To have in each layer how much dropout to use at the later layers, but then there's a couple of extra things that we need to learn about, and specifically it's this thing called embeddings. So this is really the key new concept. We have to learn about all right, so all we're doing basically is we're going to take our let's forget about categorical variables for a moment and just think about the continuous variables for our continuous variables. All we're going to do is we're going to grab them all. Okay, so for our continuous variables, we're basically going to say like okay, here's a big list of all of our continuous variables like the minimum temperature and the maximum temperature and the distance to the nearest competitor and so forth. Right and so here's just a bunch of floating-point numbers, and so basically what the neuron that's going to do is going to take that that 1d array, or or vector or to be very DL like rank one tensor or means the same thing. Okay, so we're going to take our egg one tensor and let's put it through a matrix multiplication. So let's say this has got like I don't know 20 continuous variables and then we can put it through a matrix which must have 20 rows. That's how matrix multiplication works and then we can decide how many columns we want right. So maybe we decided 100 right and so that matrix model captions going to spit out a new length. 100 rank 1 tensor.

Okay, that's that's what that's! What a linear! That's? What a matrix product does and that's the definition of a linear layer, indeed what okay and so then the next thing we do is we can put that through a rail you right, which means we throw away the negatives okay, and now we can put that through Another matrix product, okay, so this is going to have to have a hundred rows by definition and we can have as many columns as we like, and so let's say, maybe this was the last layer. So the next thing we're trying to do is to predict sales. So there's just one value: we're trying to predict for sales, so we could put it through a matrix product that just had one column and that's going to spit out a single number all right. So that's like that's kind of like a one layer neural net. If you like now in practice, you know we wouldn't make it one layer, so we would actually have leg. You know maybe we'd have 50 here, and so then that gives us a 50 long vector and then maybe we then put that into our final 50 by one and that's if it's out a single number and one reason I would have to change that there was To point out, you know rally, you would never put rally you in the last layer. I could never want to throw away the negatives because that the softmax - let's go back to the softness, the soft max needs negatives in it, because it's the negatives that are the things that allow it to create low probabilities. That's minor detail, but it's useful to remember.

Okay, so basically, so, basically, a simple view of a fully connected euro net is something that takes in as an input a rank, one tensor. It's bits: it's through a linear layer, an activation layer, another linear layer, softmax and that's the output, okay, and so we could obviously decide to add more linear layers. We could decide, maybe to add, dropout all right, so these are some of the decisions that we need. We get to make right, but we there's not that much. We can do right, there's not much really crazy architecture stuff to do so when we come back to image models later in the course we're going to learn about all the weird things that go on and like resonates and inception networks, and but in these fully connected Networks, they're, really pretty simple they're, just in dispersed linear layers, that is matrix products and activation functions like value and a soft mix at the edge, and if it's not classification, which actually ours is not classification in this case we're trying to predict sales, there isn't even A soft mix right: we don't want it to be between 0 and 1. Ok, so we can just throw away the last activation altogether. If

10. 00:50:40

- **Dealing with categorical variables**
- **like 'day-of-week' (Rossmann cont.), embedding matrices, 'cat_sz', 'emb_szs', Pinterest, Instacart**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

We have time we can talk about a slight trick. We can do there, but for now we can think of it that way. So that was all assuming that everything was continuous right. But what about categorical right? So we've got like day of week right and we're going to treat it as categorical practice like Saturday, Sunday Monday, that should be 6 ready. Okay, how do we feed that in? Because I want to find a way of getting that in so that we still end up with a wreck. One tends to refloat, and so the trick is this. We create a new little matrix of with seven rows and as many columns as we choose right. So, let's pick four all right, so here's our seven rows and four columns right and basically what we do is, let's add our categorical variables to the end. So let's say the first row was Sunday right. Then what we do is we do a lookup into this matrix. We say: oh here's sunday, we do and look up into here and we grab this row and so this matrix we basically fill with floating-point numbers. So we're going to end up grabbing little subset of for floating-point numbers at Sunday's particular for floating-point numbers, and so that way we convert Sunday into a rank 1 tensor of for floating-point numbers and initially those four numbers are random. All right - and in fact this whole thing - we initially start out random, okay, but then we're going to put that through our neural net right. So we basically then take those four numbers and we remove sunday. Instead, we add our four numbers on here right, so we've turned our categorical thing into a floating-point vector, and so now we can just put that throughout neural net just like before and at the very end we found out the loss and then we can figure out Which direction is down and do gradient descent in that direction, and eventually that will find its way back to this little list of four numbers and it'll, say: okay, those random numbers, weren't very good.

This one needs to go up a bit that one is to go up a bit, that one is to go down a bit that one is to go up a bit and so will actually update our original those four numbers in that match and we'll do this Again and again and again, and so this this matrix will stop looking random and it will start looking more and more like, like the exact four numbers that happen to work best for Sunday. The exact four numbers that happen to work best for Friday and so forth, and so in other words, this matrix, is just another bunch of weights in our neural net. All right, and so matrices of this type are called embedding matrices. So an embedding matrix is something where we start out with an integer between zero and the maximum number of levels of that category. We literally index into a matrix to find a particular row. So if it was, the level was one we take the first row. We grab that road and we append it to all of our continuous variables, and so we now have a new vector of continuous variables and when we can do the same thing, so let's say zip code right. So we could like have an embedding matrix. Let's say there are 5,000 zip codes. It would be 5,000 rows long as wide as we decide. Maybe it's 50 wide and so we'd say: ok, here's 9, 4. 0. 0. 3. That zip code is index number 4. You know matrix, ordered out and we'd find the fourth row regret those 50 numbers and append those on to our big vector and then everything after that is just the same. We just put it through our linear layer, a linear layer, whatever.

What are those 4 numbers represent? That's a great question and we'll learn more about that when we look at collaborative filtering, but now they represent no more or no less than any other parameter in a neural net. You know they're just they're, just parameters that we're

learning that happen to end up giving us a good loss. We will discover later that these particular parameters, often, however, are human interpretive. All and quote can quite interesting, but that's a side effect of them. It's not fundamental. They're, just for random numbers for now that we're that we're learning or sets of four random numbers to have a good heuristic for at the dimensionality of embedding matrix. So why four here, I sure do so. What I first of all did was I made a little list of every categorical variable and its cardinality, okay, so they're they allow so there's a hundred and there's a thousand plus different stores, apparently in Rothman's Network. There are eight days of the week. That's because there are seven days of the week, plus one left over for unknown, even if there were no missing values in the original data. I always still set aside one just in case there's a missing or an unknown, or something different in the test set again for years, but there's actually three plus room for an unknown and so forth right.

So what I do my rule of thumb is this: take the cardinality with the variable divide it by two, but don't make it bigger than 50 okay. So these are my embedding matrices, so my store matrix, so there has to have a thousand one hundred and sixteen rows cuz. I need to look up right to find his store number three and then it's been a return back a rank, one tensor of length. Fifty day of week, it's going to look up into which one of the eight and returning the thing of length four. So what you typically build on embedding metrics for each categorical feature: yes yeah! So that's what I've done here. So I've said for see in categorical variables see how many categories there are and then for each of those things create one of these. And then this is called embedding sizes. And then you may have noticed that. That's actually the first thing that we pass to get learner, and so that tells it for every categorical variable. That's the embedding matrix to use for that variable. That is behind you, listen yes, traffic aggression! So, besides our random initialization and there are other ways to actually initialize, embedding yes or no there's two ways: one is random. The other is pre-trained and we'll probably talk about pre-trained more later in the course. But the basic idea, though, is if somebody else at Rossmann had already trained a neural net. Just like you, you would use a pre trained net from imagenet to look at pictures of cats and dogs.

If somebody else is pre-trained a network to predict cheese sales in ruspin, you may as well start with their embedding matrix of stores to predict liquor sales in Rossmann, and this is what happens, for example, at Pinterest and Institute. They both use this technique. Instacart uses it for routing their shoppers, Pinterest uses it for deciding what to display on a web page when you go there and they have embedding matrices of products in instigates case of stores that get shared in the organization. So people don't have to train you once so for the embedding sighs. Why wouldn't you just use like open hot scheme and just well? What is the advantage of doing this they're supposed to just? Do it well good question? So so we could easily, as you point out, have instead of passing in these four numbers record instead of passed in seven numbers, all zeroes, but one of them is one, and that also is a list of floats and that would totally work and that's how, generally Speaking, categorical variables have been used in statistics for many years. It's called dummy variables. The problem is that in that case, the concept of sundae could only ever be associated with a single floating-point number right, and so it basically gets this kind of linear behavior. It says, like sunday is more or less of a single thing: yeah worth noticing directions. It's saying like now.

Sunday is a concept in four dimensional space right, and so what we tend to find happen is that these embedding vectors tend to get these kind of rich semantic concepts. So, for example, if it turns out that weekends kind of have a different behavior you'll tend to see that Saturday and Sunday will have like some particular number higher or more likely. It turns out

that certain days of the week are associated with higher sales of certain kinds of goods that you kind of can't go without. I don't know like gas or milk see where else there might be. Other products like like wine, for example, like wine, that tend to be associated with like the days before weekends or holidays right. So there might be kind of a column which is like to what extent is this day of the week kind of associated with people going out? You know so basically yeah by by having this higher dimensionality dektor, rather than just a single number. It gives the deep Learning Network a chance to learn these rich representations, and so this idea of an embedding is actually what's called a distributed representation, it's kind of the fun most fundamental concept of neural networks. This is the idea that a concept in a neural network has a kind of a a high dimensional representation, and often it can be hard to interpret because the idea is like each of these numbers in this vector doesn't even have to have just one meaning.

You know it could mean one thing if this is low and that one's high and something else if that one's high and that one's low, because it's going through this kind of rich nonlinear function right, and so it's this. It's this rich representation that allows it to learn such such such interesting relationships, I'm kind of oh another, question sure I'll speak louder. So are there he's in a meeting, so I get the the fundamental of be like the word vector were to Vic, vector algebra even run on this thing: are the embedding suited suitable for certain types of variables like or are these only suitable, for there are different Categories that that the embeddings are suitable for an embedding is suitable for any categorical variable. Okay, so so the only thing it it can't really work well at all. Four would be something that is too high cardinality, so I'm like, in other words we had like whatever it was six hundred thousand rows. If you had a variable with six hundred thousand levels, that's just not a useful categorical variable. You could packetize it. I guess but yeah in general, like you, can see here that the the third place getters in this competition really decided that everything that was not too high cardinality, they put them all as categorical, very and I think that's a good rule of thumb. You know if you can make a categorical variable you may as well, because that way it can learn this rich distributed representation.

Where else, if you leave it as continuous, you know, the most it can do is to kind of try and find a know. A single functional form that fits it well after question, so you were saying that you are kind of increasing the dimension, but actually in most cases we will use a one holding column which has even a bigger dimension that so in a way you are also reducing, But in the most reach, I think that's very good yeah it like yes, you know you can figure this one hot encoding which actually is high dimensional, but it's not meaningfully high dimensional, because everything set one is easy right. I'm saying that also because even this will reduce the amount of memory and things like this, that you have to write, you're, absolutely right and, and so we may as well go ahead and actually destroyed like what's going on with the matrix algebra. Behind the scenes see this: if this doesn't quite make sense, you can kind of skip over it, but for some people I know this really helps if we started out with something saying this is Sunday right. We could represent this as a one, hot, encoded, vector right and so Sunday, you know maybe was position here, so that would be a 1 and then the rest of zeros, okay and then we've got our embedding matrix right with eight rows and in this case four Columns, one way to think of this actually is a matrix product right, so I said you could think of this as like.

Looking up the number one you know and finding like its index in the array, but if you think about it, that's actually identical to doing a matrix product between a one-pot, encoded vector and the embedding matrix like you're, going to go zero times this row. One times this row zero times this row, and so it's like a one hot, embedding matrix product is identical to during

the lookup, and so some people in the bad old days actually implemented embedding matrices by doing a one, hot encoding and then a matrix product. And in fact, a lot of like machine learning methods still kind of do that. But, as you know, that was kind of alluding to it's. That's terribly inefficient. So all of the modern libraries implement this as taking take an integer and do a lookup into an array. But the nice thing about realizing that is actually a matrix product mathematically is. It makes it more obvious how the gradients are going to flow. So when we do stochastic gradient descent, it's we can think of it as just another linear layer. Okay, does it say: that's like somewhat minor detail, but hopefully for some of you it helps. Could you touch on using dates and times this category course and how that affects seasonality? Yeah, absolutely that's. A great question did I cover dates. It all remember no! Okay. So I cover dates in a

11. 01:07:10

- **Improving Date fields with 'add_datepart', and final results & questions on Rossmann, step-by-step summary of Jeremy's approach**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Lot of detail in the machine learning course, but it's worth briefly mentioning here, there's a fastai function called add date. Part which takes a data frame and column in that column. Name needs to be a date. It removes unless you squat drop, equals false. It optionally removes the column from the data frame and replaces it with lots of column, representing all of the useful information about that date like day of week, day of month month of year year, is at the start of the quarter. Is at the end of the quarter, basically everything that pandas gives us, and so that way we end up when we look at our list of features where you can see them here, right, yeah month week, data etc. So these all get created for us by a date pad. So we end up with you know this eight long embedding matrix. So I guess eight rows by four column embedding matrix for day of week and conceptually that allows us or allows our model to create some. Pretty interesting time series models all right like it can, if there's something that has a seven day period cycle, that kind of goes up on Mondays and down on Wednesdays, but only for dairy and only in Berlin. It can totally do that, but it has all the information it needs to do that. So this turns out to be a really fantastic way to deal with time series. So I'm really glad you asked the question. You just need to make sure that that the the cycle indicator in your time series exists as a column.

So if you didn't have a column there called day of week, it would be very, very difficult for the neural network to somehow learn to do like a divide, mod, seven and then somehow look that up in an omitting matrix like it not impossible but really hard. It would use lots of computation, wouldn't do it very well. So an example of the kind of thing that you need to think about might be holidays. For example, you know - or if you are doing something in you know sales of beverages in San Francisco. You probably want a list of like when weather that when is the ball game on at AT & amp T Park, because that's going to impact how many people that are drinking beer in Soma right. So you need to make sure that the kind of the basic indicators or or periodicity x' or whatever there and your data and as long as they are the neuron it's going to learn to use them. So I'm kind of trying to skip over some of the non deep learning parts alright. So the key thing here is that we've got our model data that came from the data frame. We tell it how big to make the embedding matrices. We also have to tailor of the columns in that data frame, how many of those categorical variables or how

many of them are continuous variables, so the actual parameter is number of continuous variables, so you can here, you can see we just pass in how many columns Are there how many categorical variables are there? So then, that way, the the neural net knows how to create something that puts the continuous variables over here and the categorical variables over there. The embedding matrix has its own drop out alright.

So this is the dropout to apply to the embedding matrix. This is the number of activations in the first linear player, the number of activations in the second linear layer, the dropout in the first linear player, the drop out for the second linear layer. This bit, we won't worry about for now, and then, finally, is how many outputs do we want to create okay, so this is the output of the last mini layer and obviously it's one, because we want to predict a single number, which is sales okay. So, after that, we now have a learner where we can call LR find and we get the standard looking shape and we can say what amount do we want to use and we can then go ahead and start training using exactly the same API we've seen before. So this is all identical you can pass in, I'm not sure if you've seen this before custom metrics. What this does is, it just says: please print out a number at the end of every epoch. By calling this function. This is a function we defined a little bit earlier, which was the root means bread percentage error, first of all, going either the power of our sales, because our sales were originally logged. So this doesn't change the training at all it just it's just something to print out, so we trained that for a while, and you know, we've got some benefits that the original people that built this don't have. Specifically. We've got things like cyclic, all muscle, learning rate, stochastic, gradient descent with restarts, and so it's actually interesting to have a look and compare, although our validation set isn't identical to the test set, it's very similar, it's a two-week period that is at the end of the Training data, and so our numbers should be similar, and if we look at what we get point, oh nine, seven and compare that to the leaderboard public leaderboard, you can see we're kind of sort of look in the top.

Actually, that's interesting. There is a big difference between the public and private leaderboard. It would have, it would have been right at the top of the private leaderboard, but only in the top thirty or forty on the public leaderboards. So not quite sure, but you can see like we're. Certainly, in the top end of this competition, I actually tried running the third place to get his code and their final result was over a point one. So I actually think that we're trippy compared to private leaderboard, but I'm not sure so anyway, so you can see they're, basically there's a technique for dealing with time series and structured data, and you know, interestingly, the group that that used this technique. They actually wrote a paper about it, that's linked in this notebook when you compare it to the folks that won this competition and came second, they did the other folks did way more feature. Engineering like the winners of this competition, were actually subject matter. Experts in logistics, sales forecasting, and so they had their own code to create lots and lots of features and talking to the folks at Pinterest who built their very similar model for recommendations for Pinterest. They say the same thing, which is that when they switched from gradient boosting machines to deep learning, they did like way way way less feature engineering.

It was a much much simpler model and requires much less maintenance, and so this is like one of the big benefits of using this approach to deep learning. You can get state of the at results, but with a lot less work. Yes, are you using any time series in any of these fits indirectly absolutely using what we just saw? We have a day of week, month of year, all that stuff, our columns and most of them are being treated as categories, so we're building a distributed representation of January we're building a distributed representation of Sunday we're building a distributed representation of Christmas. So we're not using any plastic time

series techniques. All we're doing is true, fully connected layers in a neural net, better metrics, that's what exactly exactly yeah, so the embedding matrix is able to deal with this stuff like day of week, periodicity and so forth. In a way, richer way than any standard time series technique, I've ever come across one last question: the matrix in the earlier models we did CNN did not pass it during the fig. We passed it when the data was when we got the data, so we're not passing anything to fit just the learning rate and the number of cycles. In this case, we're passing in metrics is not a printout some extra stuff. There is a difference in the we're calling data get learner.

So, with the imaging approach, we just go learner, dot, trained and pass at the data, but in for these kinds of models, in fact, for a lot of the models, the model that we build depends on the data. In this case, we actually need to know like what embedding matrices do. We have and stuff like that. So in this case it's actually the data object that creates the learner so yeah it is. It is a bit upside down to what we've seen before yeah. So just to summarize - or maybe I'm confused so in this case, what we are doing is that we have some kind of structured data did feature engineering. We got some columnar database or something embedding matrix for the categorical variables, so the continuous we just put them straight feature engineering, yeah, then, to map it to deep learning. I just have to figure out which one I can great question. So yes, exactly if you want to use this on your own data set step, one is list the categorical variable names list. The continuous variable names put it in a data frame. Pandas dataframe step two is to create a list of which row indexes. Do you want, in your validation, set step? Three is to call this line of code using this, except like these exact you can just copy and paste it step. Four is to create your list of how big you want each embedding matrix to be and then step 5 is to call get loner. You can use these exact parameters to start with and if it over fits or under fits, you can fiddle with them and then the final step is to call fit so yeah.

Almost all of this code will be nearly identical. Have a couple of questions. One is: how is data element ation can be used in this case and the second one is why, whatever dropouts doing in here, okay, so data augmentation - I have no idea - I mean that's a really interesting question. I think it's got ta be domain-specific. I've never seen any paper or anybody in industry, doing data augmentation with structured data and deep blow, so I don't think it can be done. I just haven't seen it done. What is dropout doing exactly the same as before. So at each point we have the output of each of these. Linear layers is just a rank. One tensor and so dropout is going to go ahead and say: let's throw away half of the activations, and the very first dropout imbedding drop out literally goes through the embedding matrix and says: let's throw away half the activations, that's it: okay, let's

12. 01:20:10

- **More discussion on using Fast.ai library for Structured Data.**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Take a break and let's come back at five past eight: okay, thanks everybody! So now we're gon na move into something equally exciting. Actually, before I do, I just been sure that I had a good question during the break, which was what's the downside like like almost no one's using this. Why not and - and basically I think the answer is like, as we discussed before. No one in academia almost is working on this because it's not something that people really publish on and as a result, there haven't been really great examples where people could look

at and say: oh here's a technique that works well. So, let's have our company implement it, but perhaps equally importantly, until now, with this fastai library, there hasn't been any way to do it. Conveniently if you wanted to implement one of these models, you had to write all the custom code yourself. Where else now, as we discussed it's, you know sick, it's basically a six step process you know involving about you know not much more than six lines of code. So the reason I mentioned this is to say, like I think there are a lot of big commercial and scientific opportunities to use this to solve problems that previously haven't been solved very well before so, like I'll, be really interested to hear. If some of you try this out, you know maybe on like old cattle competitions you might find like.

Oh, I would have won this if I'd use this technique, that would be interesting or if you've got some data set. You work with at work without some kind of model that you've been doing to the GBM or a random forest. Does this help? You know the thing I I'm still somewhat new to this. I've been doing this for basically, since the start of the year was when I started working on these structured, deep learning models, so I haven't had enough opportunity to know where might it fail? It's worked for nearly everything I've tried it with so far, but yeah. I think this class is the first time that there's going to be like more than half a dozen people fulfilled who actually are working on this. So I think you know, as a group, we're gon na hopefully learn a lot and build some interesting things, and this would be a great thing if you're thinking of writing a post about something or here's, an area that there's a couple of that. There's a poster in staccato about what they did. Pinterest has a an O'reilly, a a video about what they did. That's about it and there's two academic papers both about Carroll competition victories one from Yoshi, Joshua Ben geo and his group. They won a taxi destination forecasting. Competition and then also the one linked for this rossmann competition, so yeah

13. 01:23:30

- **Intro to Natural Language Processing (NLP)**
- **notebook 'lang_model-arxiv.ipynb'**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

There's some background on that. Alright, so language, natural language processing is the area which is kind of like the most up-and-coming area moaning, it's kind of like two or three years behind computer vision in deep learning. It was kind of like the the second area that deep learning started getting really popular in and you know, computer vision got to the point where it was like clear state of the art for most computer vision, things. Maybe in like 2014, you know and in some things in like 2012 in NLP, we're still at the point where, for a lot of things, deep learning is now the state of the art, but not quite everything, but as you'll see the state of kind of the Software and some of the concepts is much less mature than it is for computer vision, so in general, none of the stuff we talked about after computer vision is going to be as like settled as the computer vision and stuff was so NLP. One of the interesting things is in the last few months, some of the good ideas from computer vision have started to spread into NLP for the first time and we've seen some really big advances. So a lot of the stuff you'll see in NLP is is pretty new, so I'm going to start with a particular kind of NLP problem and one of the things refined in NLP is like. There are particular problems you can solve and they have particular names and so there's a particular kind of problem in NLP called language modeling and language.

Modeling has a very specific definition. That means build a model. We're given a few words of a sentence. Can you predict what the next word is going to be so, if you're, using your mobile phone and you're typing away and you press space and then it says like this is what the next word might be like SwiftKey? Does this like really well and SwiftKey, actually uses deep learning for this? That's that's a language model. Okay, so it has a very specific meaning. When we say language modeling, we mean a model that can predict the next word of a sentence. So let me give you an example: I downloaded about 18 months worth of papers from archive, so for those of you that don't know what archive is the most popular preprint server in this community and various others, and has you know lots of academic papers? And so I grabbed the abstracts and the topics for each and so here's an example. So the category of this particular paper, what computer CSMA is computer science and networking and then the summary let the abstract of the paper they're, seeing the exploitation of mm-wave bands is one of the key enabler for 5g mobile bla bla bla, okay. So here's like an example piece of text from my language model, so I trained a language model on this archived data set that I downloaded and then I built a simple little test, which basically you would pass it. Some like priming text, so you'd say like.

Oh imagine you started reading a document that said category is computer science, networking and the summary is algorithms that and then I said, please write an archive abstract. So it said that if it's networking algorithms that use the same network as a single node, I'm not able to achieve the same performance as a traditional network based routing algorithms. In this paper we propose a novel routing scheme, but okay, so it it's learnt by reading archive papers that somebody who is playing algorithms, that where the word cat CSM ie came before it is going to talk like this and remember it started out not knowing English. At all right, it actually started out with an embedding matrix for every word in English, that was random. Okay, and by reading lots of archive papers, it weren't what kind of words followed others. So then, I tried what, if we said: cat computer science, computer vision, summary algorithms that use the same data to perform image specification are increasingly being used to proves the performance of image classification, algorithms, and this paper. We propose a novel method for image specification using a deeper convolutional neural network parentheses CNN. So you can see like it's kind of like almost the same sentence as back here, but things have just changed into this world of computer vision rather than networking.

So I tried something else which is like okay category computer vision and I created the world's shortest ever abstract that words and then I said title on, and the title of this is going to be on that performance. Object. Learning for image classification in OS is end of string. So that's like end of title. What if it is networking summary algorithms title on the performance of wireless networks as opposed to towards computer vision towards a new approach to image specification networking towards then you approach to the analysis of wireless networks. So, like I find this mind-blowing right, I started out with some random matrices, which had like literally no no pre-trade anything. I fed at 18 months worth of archived articles and it learnt not only how to write English pretty well but also, after you say, something's a convolutional neural network. You should then use parentheses to say what it's called and, furthermore, that the kinds of things people talk could say: create algorithms for in computer vision are performing image. Classification and in networking are achieving the same performance as traditional network based routing algorithms. So, like a language model is, can be like incredibly deep and subtle right and so we're going to try and build that, but actually not because we care about this at all. We're going to build it because we're going to try and create a pre-trained model.

What we're actually going to try and do is take IMDB movie reviews and figure out whether they're, positive or negative. So if you think about it, this is a lot like cats vs. dogs, that's a classification algorithm, but rather than an image, we're going to have the text of a review. So I'd really like to use a pre-trained Network like I would at least my connect to start with a network that knows how to read English right, and so my view was like okay to know how to read English means you should be able to like predict The next word of a sentence, so what if we pre train a language model and then use that pre-trained language model and then just like in computer vision? Stick some new layers on the end and ask it instead of predicting the next word in the sentence. Instead predict whether something is positive or negative. So when I started working on this, this was actually a new idea. Unfortunately, in the last couple of months I've been doing it. You know a few people have actually couple people started publishing this, and so this has moved from being a totally new idea to being a you know, somewhat new idea. So so this idea of creating a language model making that

14. 01:31:15

- **Creating a Language Model with IMDB dataset**
- **notebook 'lesson4-imdb.ipynb'**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

The pre-trained model for a classification model is what we're going to learn to do now, and so the idea is we're really kind of trying to leverage exactly what we learnt in our computer vision work, which is: how do we do fine tuning to create powerful classification Models, yes, you know, so why don't you

15. 01:31:34

- **Question: "So why don't you think that doing just directly what you want to do doesn't work better?" (referring to the pre-training of a language model before predicting whether a review is positive or negative)**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Think that doing just directly, what you want to do doesn't work better well because it doesn't just turns out, it doesn't empirically and the reason it doesn't is a number of things. First of all, as we know, fine-tuning a pre-trained network is really powerful right. So if we can get it to learn some related tasks first, then we can use all that information to try and help it on the second task. The other reason is IMDB movie reviews. You know up to a thousand words long, they're, pretty big, and so after reading a thousand words knowing nothing about how English is structured or even what the concept of a word is or punctuation or whatever at the end of this thousand integers. You know they end up being integers. All you get is a 1 or a 0 positive or negative, and so trying to like learn the entire structure of English and then how it expresses positive and negative sentiments from a single number is just too much to expect. So, by building a language model first, we can try to build a neural network, that kind of understands the English of movie reviews, and then we hope that some of the things that's learnt about are going to be useful in deciding whether something's a positive or a Negative nutrition. That's a great question! You can pass that thanks! Is this similar to

16. 01:33:09

- **Question: "Is this similar to the char-rnn by karpathy?"**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

The car RNN yeah - this is somewhat similar to our Olympic apathy, so the famous car, as in CH AR AR and in try to predict the next letter, given a number of previous letters, language models generally work at a word level. They don't have to and doing things that a word level turns out to be can be quite a bit more powerful and we're going to focus on word level. Modeling in this course. To what extent are these generated words actually copies of what it found in the in the training data set, or are these completely random things that it actually learned, and how do we know how to distinguish between those two yeah? I mean these are awkward questions. The the words are definitely words, we've seen before the work, because it's not at a character level, so it can only give us the word it seen before the sentences there's a number of kind of rigorous ways of doing it. But I think the easiest is to get a sense of like well. Here are two like different categories, where it's kind of created very similar concepts, but mixing them up in just the right way like it. It would be very hard to to do what we've seen here just by like speeding back things at scene before, but you could of course, actually go back and check. You know. Have you seen that sentence before or like a stream distance? Have you seen a similar sentence before, in this case? Oh and of course, another way to do it is the length, most importantly, when we train the language model as we'll see we'll have a validation set, and so we're trying to predict the next word of something.

That's never seen before, and so, if it's good at doing that, it should be good at generating text. In this case, the purpose, the purpose is not to generate text. That was just a fun example, and so I'm not really gon na study that too much. But you know you're during the week turtley can like you, can totally build your you know Great American Novel generator or whatever. There are actually some tricks to to using language models, to generate text that I'm not using here, they're, pretty simple. We can talk about them on the forum if you like, but my focus is actually on classification, so I think that's the thing which is incredibly powerful. Like text classification, I don't know you're a hedge fund. You want to like read every article as soon as it comes out through writers or Twitter or whatever, and immediately identify things which in the past have caused. You know massive market drops, that's a classification model or you want to recognize all of the customer service queries which tend to be associated with people who who leave your you know who cancel their contracts in the next month's. That's a classification problem so like it's a really powerful kind of thing for data journalism, Activision that activism more promise so forth, right, like I'm, trying to class documents into whether they're part of legal discovery or not part of legal discovery. Okay, so you get the idea.

So in terms of stuff we're importing we're importing a few new things here, one of the bunch of things we're importing is torch text torch text is PI, torches like LP library and so fast. Ai is designed to work hand in hand with torch text as you'll, see and then there's a few text specific sub bits of faster, fastai that we'll be using. So we're going to be working with the IMDB large movie review data set. It's very very well studied in academia. You know lots and lots of people over the years have studied this. Data set fifty thousand reviews highly polarized reviews, either positive or negative. Each one has been classified by sentiment, okay, so we're going to try our first of all. However, to create a language model, so we're

going to ignore the sentiment entirely, all right so just like the dogs and cats Cree trainer model to do one thing and then fine tune it to do something else, because this kind of idea in NLP is is so So so new there's basically no models you can download for this, so we're going to have to create their own right. So having downloaded the data, you can use the link here we do the usual stuff of saying the path to training and validation path and, as you can see, it looks pretty pretty traditional compared to a vision. There's a directory of training, there's a directory of tests. We don't actually have separate test and validation in this case and just like in envision.

The training directory has a bunch of files in it in this case, not representing images but representing movie reviews. So we could cat one of those files, and here we learn about the classic zombie garand movie. I have to say, with a name like zombie, gedan and an atom bomb on the front cover. I was expecting a flat-out chop-socky fun coup rent it. If you want to get stoned on a Friday night and laugh with your buddies, don't rent it if you're an uptight, weenie or want a zombie movie or lots of fresh eating, I think I'm going to enjoy zombie getting so alright. So we've learned something today all right, so we can just use standard UNIX stuff to see like how many words are in the data set, so the training set we've got seventeen and a half million words test set. We've got 5.6 million words, so he is. These are, this is IMDB, so IMDB is random people. This is not New York Times listed review. As far as I know, okay, so before we can do anything,

17. 01:39:30

- **Tokenize: splitting a sentence into an array of tokens**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

With text we have to turn it into a list of tokens. Token is basically like a word right, so we're going to try and turn this eventually into a list of numbers. So the first step is to turn it into a list of words. That's called tokenization in NLP NLP has a huge lot of jargon that will we'll learn over time. One thing: that's a bit tricky, though, when we're doing tokenization is here: I've, I've, tokenized that review and then joined it back up with spaces and you'll. See here that wasn't has become two tokens which makes perfect sense. Right was, is two things right: dot dot dot has become one token right. Where else lots of exclamation marks has become lots of tokens, so, like a good tokenizer, will do a good job of recognizing like pieces of it in your sentence. Each separate piece of punctuation will be separated and each part of a multi-part word will be separated as appropriate. So Spacey is, I think, it's an Australian develop piece of software. Actually, that does lots of you know, P stuff. It's got the best tokenizer, I know, and so past AI is designed to work well with the Spacey tokenizer, as is torch text, so here's an example of tokenization alright. So what we do with torch text is we basically have to start out by creating something called a field, and a field is a definition of how to pre-process some text and so here's an example with the definition of a field.

It says I want to lowercase a text and I want to tokenize it with the function called Spacey tokenize. Okay, so it hasn't done anything yet we're just telling you when we do do something. This is what to do and so that we're going to store that description of what to do in a thing called capital text, and so this is this is none of this, but this is not fastai specific at all. This is part of torch text. You can go to the torch text. Website read the docs, there's not lots of Doc's. Yet this is all very, very new, so probably the best information you'll find about it is

in this lesson, but there's some more information on this site all right. So what we can now do is go ahead and create the usual fastai model data object, okay and so to create the model data object. We have to provide a few bits of information, but you have to say: what's the training set so the path to the text, files, the validation set and the test set in this case. Just to keep things simple, I don't have a separate validation and test set. So I'm going to pass in the validation set for both of those two things right so now we can create our model data object as per usual. The first thing you give it as a path. The second thing we give it is the torch text field, definition of how to pre-process that text. The third thing we give it is the dictionary or the list of all of the files. We have trained validation tests.

As per usual, we can pass in a batch size and then we've got a special special couple of extra things here. One is very commonly used in NLP minimum frequency. What this says is in a moment we're going to be replacing every one of these words with an integer which basically will be a unique index for every word, and this basically says if there are any words that occur less than 10 times just call it unknown. Right, don't think of it as a word, but we'll see that indeed more detail in a moment. We're going to see this in more detail as well. Be PTT stands for back prop through time, and this is where we define how long a sentence will we stick on the GPU at once, so we're going to break them up in this case we're going to break them up into sentences of 70 tokens or less On the whole, so we're going to see all this in a moment:

18. 01:43:45

- **Build a vocabulary 'TEXT.vocab' with 'dill/pickle'; 'next(iter(md.trn_dl))'**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Okay, so after building our model data object right, what it actually does is it's going to fill this text field with an additional attribute called vocab, and this is a really important and LP concept. I'm sorry there's so many NLP concepts we just have to throw at you kind of quickly, but we'll see them a few times right. The vocab is the vocabulary, and the vocabulary in NLP has a very specific meaning. It is what is the list of unique words that appear in this text, so every one of them is going to get a unique in this. So let's take a look right here. Is text vocab dot I to s this dancer. This is all torch text, not faster hide text, upper cap, dot, int 2 string Maps, the integer 0 to unknown the integer 1, the padding unit 2 to desert, then in comma dot and of 2 and so forth right. So this is the first 12 elements of the array of the vocab from the IMDB movie review and it's been sorted by frequency and except for the first two special ones. So, for example, we can then go backwards. S2I string to int here is the it's in position: 0, 1, 2, so stream to end the is 2. So the vocab lets us take a word and map it to an integer or take an integer and that a tour word right, and so that means that we can then take the first 12 tokens, for example, of our text and turn them into twelve inch. So, for example, here is of the agency 7 2, and here you can see 7/2 right, so we're going to be working in this form.

Did you have a question deputy plus that back there you know? Is it a common tyranny stemming or limit izing? Not really? No generally tokenization is is what we want like with a language model. We, you know to keep it as general as possible. We want to know what's coming next and so like whether its future tense or past tense or plural or seem to learn like we. Don't really know which things are going to be interesting in which ant, so it seems that

it's generally best to kind of leave it alone as much as possible, be the short answer. You know. Having said that, as I say, this is all pretty new, so if there are some particular areas that some researcher maybe is already discovered that some other kinds of pre-processing you're helpful, you know I wouldn't be surprised not to know about it, so we Abdullah, we know You don't natural language is in context. Important context is very important, so individual words, no, no we're not looking worth this. Is this look. This is, I just don't get some of the big premises of this like they're there in order yeah. So, just because we replaced I with the number 12, these are still in that order. Yeah, there is a different way of dealing with natural language called a bag of words and bag of words. You through throw away the order in the context and in the machine learning course we'll be learning about working with bag of words, representation, z'. But my belief is that they are no longer useful or in the verge of becoming no longer useful, we're starting to learn.

How to use deep learning to use context properly now, but it's kind of for the first time it's really like only in the last few months right so I mentioned that we've got two numbers batch size and B PTT back crop through time, so this is kind Of subtle, so we've got some big long piece of text. Okay, so we've got some big long piece of text. You know here's our sentence, it's a bunch of words right and actually what happens in a language model is even though we have lots of movie reviews. They actually all get concatenated together into one big block of text right, so it's basically predict the next word in this huge long thing, which is all of the IMDB movie reviews concatenated together. So this thing is you know what do we say? It was like tens of millions of words long and so what we do is we split it up into batches first right, so these, like aerial, spits into batches right, and so, if we said we want a batch size of 64, we actually break the whatever. It was sixty million words into 64 sections right and then we take each one of the 64 sections and we move it like underneath the previous one. I didn't do a great job of that all right move it underneath. So we end up with a matrix, which is you sixty-four? Actually, I think we've moved them across twice, so it's actually, I think, just transpose it. We end up with the matrix. It's like 64 columns wide and the length. Let's say the original was 64 million right.

Then the length is like 10 million long right, so each of these represents one sixty-fourth with our entire IMDB. Refused it all right, and so that's our starting point. So then, what we do is we then grab a little chunk of this at a time and those chunk lengths are approximately equal to be PTT, which I think we had equal to 70. So he basically grab a little 70 long section and that's the first thing. We check into our GPU that's a batch right, so a batch is always of length of width, 64 or batch size and each bit is a sequence of length up to 70. So let me show you all right so here, if I go, take my train data loader. I know if you folks have tried playing with this yet, but you can take any data, loader wrap it with inner turn it into an iterator and then call next on it to grab a batch of data. Just as if you were a neural net, you get exactly what the neuron that gets and you can see here. We get back a 75 by 64 sensor right. So it's 64 wide right - and I said it's approximately 70 high and but not exactly and that's actually kind of interesting, a really neat trick that torch text does. Is they randomly change the backprop through time number every time, so each epoch it's getting slightly different bits of text. This is kind of like in computer vision. We randomly shuffle the images we can't randomly shuffle the words right because we needed to be in the right order, so instead we randomly move their breakpoints a little bit okay, so this is the equivalent. So in other words, this this here is of length 75 right. There's a there's, an ellipsis in the middle, and that represents the first 75 words of the first review right.

Where else this 75 here represents the first 75 words of this of the second of the 64 segments.

Let's it have to go in, like 10 million words to find that one right and so here's the first 75 words of the last of those 64 segments. Okay, and so then, what we have down here is the next sequence right so 51, there's 51. 6. 1. 5 there's 6. 1. 5. 25 is 25 right, and in this case it actually is of the same size. It's also 75 plus 64, but for minor technical reasons, it's being flattened out into a single vector, but basically it's exactly the same at this matrix. But it's just moved down by one because we're trying to predict the next word right so that all happens for us right. If we ask for - and this is the first date I know, if you ask for a language model data object, then it's going to create these batches of batch size width by B, PTT height bits of our language corpus, along with the same thing shuffled along by One word right, and so we're always going to try and predict the next word. Yes, so why don't you, instead of just arbitrarily choosing 64? Why don't you choose like, like 64, is a large number, maybe like stood by sentences and make it a large number and then padded with zero or something? If you, you know, so that you actually have a one full sentence per line? Basically, wouldn't that make more sense, not really because remember we're using columns right, so each of our columns is of length about 10 million right. So, although it's true that those columns aren't always exactly finishing on a full stop, there's so damn long, we don't care because they're, like 10 million won and we're trying to also line, contains multiple cents incentive column contains more costumes and sorry yeah it's of length about 10 million - and it contains many many many many many sentences because remember the first thing we did was take all thing and split it into 64 groups - okay, great so um.

I found this. You know pertaining to this question this thing about like. What's in this language model, matrix a little mind-bending for quite a while, so don't worry if it takes a while, and you have to ask a thousand questions on the forum. That's fine right, but go back and listen to what I just said in this lecture again. Go back to that bit where I showed you splitting it up to 64 and moving them around and try it with some sentences in Excel or something and see if you can do a better job of explaining it than I did. Okay, because this is like how torch text works and then what fastai adds on is this idea of, like kind of how to build a a language model out of it? Well, they'll? Actually, a lot of that stolen from torch text as well like there's some times where torch text starts and fastai ends is, or vice versa, trees a little saddle. They really work closely together. Okay, so now that we have a model data object that can feed us batches, we can go ahead and create a model right, and so, in this case we're going to create an embedding matrix and our vocab. We can see how big a vocab was. Let's have a look back here, so we can see here in the model data object. There are four thousand six hundred and two kind of pieces that we're going to go through. That's basically equal to the number of the total length of everything divided by batch size times B PTT and this one I wanted to show you NT. I've got the definition up here. Number of unique tokens.

Nt is the number of tokens. That's the size of our vocab, so we've got three thirty, four thousand nine hundred and forty five unique words and notice the unique words that had to appear at least ten times. Okay, because otherwise they've been replaced with the length of the data set, is one because, as far as the language model is concerned, there's only one thing, which is the whole corpus all right and then that thing has I hear it is twenty point: six million words. You know right so those thirty four thousand hundred and forty five things are used to create an embedding matrix of number of rows is equal to thirty, four, nine, four, five right, and so the first one represents UNK. The second one represents pad the third one was dot. The fourth one was comma, with one under sketching, was there and so forth right, and so each one of these gets an embedding vector. So this is literally identical to what we did before the brick right. This is a categorical variable, it's just a very high cardinality, categorical variable and furthermore, it's the only variable right. This is pretty standard in NLP. You have

a variable, which is a word right. You have a single categorical variable single column. Basically, and it's it's of thirty. Four thousand nine hundred forty five cardinality categorical variable and so we're going to create an embedding matrix for it.

So M size is the size of the omitting vector 200. Okay. So that's going to be length 200, a lot bigger than our previous embedding vectors, not surprising, because a word has a lot more nuance to it than the concept of Sunday right or Russ means Berlin's door or whatever right. So it's generally an embedding size. For a word will be somewhere between about 50 and about 600 okay, so I've kind of done some in the middle. We then have to say, as per usual, how many activations do you want in your layers, so we're going to use 500 and then how many layers do you want in your neural net, we're going to use three okay? This is a minor technical detail. It turns out that we're going to learn later about the atom optimizer that basically the defaults for it. Don't work very well with these kinds of models, so you just have to change some of these. You know basically any time you're doing NLP. You should probably include this mine because it works pretty well. So having done that, we can now again take our model data object and grab a model out of it, and we can pass in a few different things. What optimization function do we want? How big an embedding do we want how many hidden activate how many activations number of Hitler, how many layers and how much drop out of many different kinds? So this language model, we're going to use, is a very recent development called AWD LS TM by Steven marady who's, a NLP research based in San Francisco, and his main contribution really was to show like how to put drop out all over the place in in these Nlp models, so we're not going to worry now we'll do this in the last lecture is worrying about like what all that like.

What is the architecture, and what are all these dropouts for now just know is the same as per usual. If you try to build an NLP model and draw underfitting and decrease all of these dropouts, if you're overfitting then increase all of these dropouts in roughly this ratio? Okay, that's that's my rule of thumb and again this is such a recent paper. Nobody else is working on this model anyway, so there's not a lot of guidance, but I found this. These ratios worked well, it's what Stephens been using as well there's another kind of way we can avoid overfitting that we'll talk about in the last class again for now, this one actually works totally reliably. So all of your NLP models probably want this particular line of code, and then this point we're going to talk about at the end last lecture as well. You can always improve this. Basically, what it says is when you do when you look at your gradients, and you multiply them by the learning rate, and you decide how much to update you or weights by this is clip them, like literally like sit like don't let them be more than 0.3 And this is quite a cool little trick right because, like if you're learning rates pretty high and you kind of don't want to get in that situation, we talked about where you're kind of got this kind of thing. Where you go, you know, rather than little snippets, that little step instead, you go like Oh, too big. Oh too, big right with gradient clipping.

It kind of goes this far and it's like, oh, my goodness, I'm going too far I'll stop and that's basically what gradient flipping does so anyway. So these are a bunch of parameters. The details, don't matter too much right now. You can just deal these and then we can go ahead and call fit with exactly the same parameters as usual, so Jeremy um there are all this other work. Embedding things like like worked vague and glow. So I have two questions about that. One is how are those different from these and the second question: why don't you initialize them with one of those yeah? So so, basically, that's a great question. So basically, people have pre trained these embedding matrices before to do various other tasks. They're not called pre-trained models. They're, just a pre trained, embedding matrix and you can

download them and as unit says they have names like word to Veck and love and they're, literally just a matrix, there's, no reason we couldn't download them really it's just like kind of. I found that building a whole pre-trained model in this way didn't seem to benefit much if at all, from using pre trained word vectors, where else using a whole pre-trained language model made a much bigger difference. So I can remember what a big those of you who saw word Tyvek. It made a big splash when it came out.

I mean I'm finding this technique of pre-trained language models seems much more powerful basically, but I think we can combine both to make them a little better still what what is the model that you have used like? How can I know that architecture of the model so we'll be learning about the model architecture? In the last lesson, and for now it's a recurrent neural network using something called an LS TN long, short-term memory? Okay. So so, if they had lots of details that we're skipping over - but you know you can do all this without any of those details, we go ahead and fit the model. I found that this language model took quite a while to fit so I kind of like ran it for a while noticed it was still under fitting safer. It was up to ran it. A bit more longer cycle length saved it again. It still was kind of under fitting you know, run it again and kind of finally got to the point where it's like kind of honestly, I kind of ran out of patience, so I just like saved it at that point and I did the same kind of Tests that we looked at before so I was like - oh, it wasn't quite expecting, but I realized it anyway, the best and the most like. Okay, let's see how that goes, the best performance with one movie were, I say: okay, it looks like the language models working pretty well, so I've pre-trained a language model, and so now I want to use it fine tune it to do. Classification sentiment classification now, obviously, if I'm gon na use a pre trained model, I need to use exactly the same vocab but the the word the still needs to map for the number two so that I can look up the vector that right.

So that's why I first of all load back up my my field object the thing with the vocab in right now, in this case, if I ran it straight afterwards, this is unnecessary, it's already in memory, but this means I can come back to this later right. In a new session, basically, I can then go ahead and say: okay, I've never got one more field right. In addition to my field, which represents the reviews, I've also got a field which represents the label. Okay and the details are too important here now this time I need to not treat the whole thing as one big piece of text, but every review is separate because each one has a different sentiment attached to it, but it so happens that torch text already has A data set that does that for IMDB, so I just used IMDB built into torch text. So basically, once we've done all that we end up with something where we can like grab for a particular example, or you can grab its label positive and here's. Some of the text - this is another great Tom, Berenger movie, all right, so this is all not nothing faster. I specific here we'll come back to it in the last lecture, but torch text Docs can help understand. What's going on, all you need to know is that once you've used this special tox torch text, thing called splits to grab a Spitz object. You can passed it straight into faster, a text data from splits and that basically converts a torch text object into a fastai object.

We can train on so as soon as you've done that you can just go ahead and say get model right and that gets us our learner and then we can load into it. The pre trained model, the language model right, and so we can now take that pre-trained language model and use the stuff that we're kind of familiar with right. So we can make sure all that you know all its at the last layers frozen training a bit unfreeze. It train it a bit and the nice thing is once you've got a pre trained language model. It actually trains super fast. You can see here it's like a couple of minutes for epoch and it only took me to get my is my best one here and he took me like 10 a box, so it's like 20 minutes to train this bit. It's really fast and I ended up with 94.5 %, so how

gone is 94.5 %? Well, it so happens that actually one of Steven verities colleagues James Bradbury, recently created a paper looking at the state at like they tried to create a new state of the art for a bunch of NLP things, and one of the things that looked at was IMDB And they actually have here a list of the current world's best for IMDB and even with stuff that is highly specialized for sentiment analysis. The best anybody had previously come up with 94.1, so in other words, this technique getting 94.5, it's literally better than anybody has created in the world before as far as we know, or as far as James Bradbury knows so so when I say like, there are big Opportunities to use this I mean like this is a technique that nobody else currently has access to which you know you could, like you know, whatever iBM has in what CERN or whatever any big company has.

You know that they're advertising, unless they have some secret sauce, that they're not publishing, which they don't right, because people get you know if they have a better thing, they publish it. Then you now have access to a better text. Classification method, then, has ever existed before. So I really hope that you know you can try this out and see how you go. There may be some things it works really well on and others that it doesn't work as well, and I don't know I think, this kind of sweet spot here that we had about 25,000. You know short to medium size documents if you don't have at least that much text, it may be hard to train a different language model. But, having said that, there's a lot more. We do here right and we won't be able to do it in part. 1 of this course, but in part 2, that, for example, we could start like training language models that look at like you know, lots and lots of medical journals, and then we could like make a downloadable medical language model that then anybody could use to like fine Tune on like a prostate cancer, subset of medical literature, for instance like there's so much, we could do it's kind of exciting, and then you know to the next point. We could also combine this with like pre-trained word vectors, it's like even without trying that hard.

Like you know, we even without use like we could have pre-trained a Wikipedia, say corpus language model and then fine-tuned it into a IMDB language model and then fine tune that into an IBM. Imdb sentiment analysis model and we would have got something better than this. So like this, I really think this is the tip of the iceberg, and I was talking there's a really fantastic researcher called Sebastian Reuter, who is basically the only NLP researcher. I know who's been really really writing a lot about pre-training and fine tuning and transfer learning and NLP, and I was asking him like: why isn't this happening more and his view was it's because there isn't the software to make it easy? You know so I'm actually going to share this lecture with with him tomorrow, because you know it feels like there's. You know,

- The rest of the video covers the ins and outs of the notebook ‘lesson4-imdb’, don’t forget to use ‘J’ and ‘L’ for 10 sec backward/forward on YouTube videos.

19. 02:11:30

- **Intro to Lesson 5: Collaborative Filtering with Movielens**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Hopefully, gon na be a lot of stuff coming out now that we're making it really easy to do this. Ok we're kind of out of time. So what I'll do is I'll quickly look at collaborative filtering introduction and then we'll finish it next time, but collaborative filtering. There's very very

little new to learn. We've basically learned everything. We're gon na need, so collaborative filtering will will cover this quite quickly next week and then we're going to do a really deep dive into collaborative filtering next week, where we're going to learn about like we're. Actually, going to from scratch, learn how to do mr. plastic gradient descent, how to create loss functions, how they work exactly and then we'll grow from there and will gradually build back up to really deeply understand. What's going on in the structured models and then what's going on in confidence and then finally, what's going on in recurrent neural networks and hopefully we'll be able to build them all from scratch? Ok, so this is kind of a gon na be really important. This movie lens data set because we've got a user to learn a lot of like really foundational theory and kind of math behind it, so the movie lens data set. This is basically what it looks like it contains: a bunch of ratings. It says user number one watched movie number 31 and they gave it a rating of two and a half at this particular time, and then they watched movie 102 nine and they gave it a rating of three and they watched reading one one's really one one.

Seven two and they gave it a rating before okay and so forth. Okay, so this is the ratings table. This is really the only one that matters and our goal will be for some use that we haven't seen before so for some user movie combination we haven't seen before we have to predict if they'll like it right, and so this is how recommendation systems are built. This is how like Amazon, besides what books, to recommend how Netflix decides, what movies to recommend and so forth, to make it more interesting, we'll also actually download a list of movies, so each movie we're actually gon na, have the title, and so for that question earlier About like, what's actually going to be in these embedding matrices, how do we interpret them? We're actually going to be able to look and see how that's working. So, basically, this is kind of like what we're creating. This is kind of crosstab of users by movies, alright and so feel free to look ahead during the week. You'll see basically, as per usual collaborative data set from CSP model data. Docket learner learn it and we're done and don't be surprised to hear when we then take that and we can kick the benchmarks, it seems to be better than the benchmarks where you look at so that'll, basically be it and then next week we'll have a deep Dive and we'll see how to actually build this from scratch.

Alright see you next week, thank you. [Applause,],

Lesson 5: Collaborative filtering; Inside the training loop

Outline

You will learn about collaborative filtering through the example of making movie recommendations, and talk about key developments that occurred during the Netflix prize.

We will dig into some lower level details of deep learning: what happens inside the training loop, how optimizers like momentum and Adam work, and regularization using weight decay. You will learn how to think spatially about math concepts like the 'chain rule', 'jacobian', and 'hessian'.

Video Timelines and Transcript

1. 00:00:01

- **Review of students articles and works**
- **"Structured Deep Learning" for structured data using Entity Embeddings,**
- **"Fun with small image data-sets (part 2)" with unfreezing layers and downloading images from Google,**
- **"How do we train neural networks" technical writing with detailed walk-through,**
- **"Plant Seedlings Kaggle competition"**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Welcome back so we had a busy lesson last week and I was really thrilled to see. Actually, one of our masters, students here at USF, actually actually took what we learned took, what we learned with structured, deep learning and turned it into a blog post which, as I suspected, has been incredibly popular because it's just something people didn't know about, and so it Actually ended up getting picked up by the towards data science publication, which I quite liked actually, if you're interested in keeping up with. What's going on a data science, it's quite good, medium publication, and so Karen talked about structured, deep learning and basically introduced. You know the the the basic ideas that we learned about last week and it got picked up quite quite widely. One of the one of the things I was pleased to say actually sebastian ruder, who actually mentioned in last week's class. As being one of my favorite researchers tweeted it and then somebody from stitch fix said: oh yeah we've actually been doing that for ages, which is kind of cute. I I kind of know that this is happening in industry. A lot and I've been telling people. This is happening in industry, a lot but nobody's been talking about it and now the Karen's kind of published a blog saying, hey check out this cool thing and they all stitch fixes like yeah we're doing that already. So so that's been great great to see, and I think there's still a lot more that can be dug into with this structured people.

Learning stuff you know to build on top of Karen's post would be that maybe like experiment with some different datasets, maybe find some old, careful competitions and see like there's some competitions that you could now win with this, or some which doesn't work for would be equally Interesting and also like just experimenting a bit with different amounts of dropout, different layer sizes. You know, because nobody much is written about this. I don't think there's been any blog posts about this before that I've seen anywhere there's a lot of unexplored territory. So I think, there's a lot we could. We could build on top of here and there's definitely a lot of interest as well. One person on Twitter saying this is what I've been looking for for ages. Another thing which I was pleased to see is Nikki or who we saw his cricket versus baseball predictor, as well as his a currency predictor after less than one went on to download something a bit bigger, which was to download a couple of hundred of images of Actors - and he manually went through and checked which well, I think, first of all, he like used Google to try and find ones with glasses and ones. Without then, he manually went through and checked that that they put in the right spot, and this was a good example of one where vanilla ResNet didn't do so well with just the last layer, and so what Nikhil did was he went through and tried on freezing The layers and using differential learning rates and got up to a hundred percent accuracy, and the thing I like about these things that Nikhil was doing is the way he's he's not downloading a kegel data set he's like deciding on a problem that he's going to try And solve he's going from scratch from google and he's actually got a link here even to the suggested way to help you download images from Google. So I think this is great and actually gave a talk just this afternoon at singularity University to an executive team of one of the world's largest telecommunications companies and actually show them this post.

Because the folks there were telling me that that all the vendors that come to them and tell them they need, like millions of images and huge data. Centers will have hardware, and you know they have to buy a special software that only these vendors can provide. And I said like actually, this person has been doing it, of course, for three weeks now and look at what he's just done with a computer that cost him 60 cents an hour and they were like. They were so happy to hear that, like okay they're, you know this actually is in the reach of normal people. I'm assuming Nikhil is a normal person I haven't actually and if your proudly abnormal Nicole, I apologize, I actually went and actually had a look at his cricket classifier, and I was really pleased to see that his code actually is the exact same code that were used In Lesson one I was hoping that would be the case. You know the only thing he changed was the number of epochs. I guess so. This idea that we can take those four lines of code and reuse it to do other things. That's definitely turned out to be true, and so these are good things to show like it yeah your organization, if you're anything like the executives of this big company I spoke to today, there'll be a certain amount of like not to surprise but almost like pushback like If this was true, somebody does it all that message. She said if this was true, somebody would have told us so like.

Why isn't everybody doing this already so we'd like it? I think you might have to actually show them. You know, maybe you can build your own there's. Some internal data you've got at work or something like here. It is, you know, didn't cost me anything. It's all finished fiddly or badly. I don't know how to pronounce his name correctly, has done another very nice post on just an introductory post on how we train neural networks, and I've wanted to point this one out as being like. I think this is one of the participants in this course who has got a particular knack for technical communication, and I think we can all learn from you know from his post about about good technical writing. What I really like particularly, is that he he assumes almost nothing like he has a kind of a very chatty tone and describes everything, but he also assumes that the reader is intelligent. But you know so like he's not afraid to kind of

say here's a paper or here's an equation or or whatever, but then he's going to go through and tell you exactly what that equation means. So it's kind of like this nice mix of like writing for respectfully for an intelligent audience, but also not assuming any particular background knowledge. So then, I made the mistake earlier this week of posting a picture of my first placing on the Carroll seedlings competition, at which point five other fastai students posted their pictures of them pass over the next few days.

So this is the current leaderboard for the cattle plant seedlings competition. I believe the product top six are all fastai students or in the worst of those teachers, and so I think this is like a really Oh James is just passed. He was first. This is a really good example of like what you can do, but this is trying to think it was like a small number of thousands of images, and most of the images were only were less than a hundred pixels by a hundred pixels and yet week. You know, I bet my approach was basically to say: let's just run through the notebook, we have pretty much default, took the I don't know an hour and I'm, I think the other students doing a little bit more than that, but not a lot more and basically What this is saying is yeah these these techniques work pretty reliably to a point where people that aren't using the fast - I know, libraries, you know literally really struggling. Let's just pick off. These are fastaid. A students might have to go down quite a way. So I thought that was very interesting and really really cool. So

2. 00:07:45

- **Starting the 2nd half of the course: what's next ?**
- **MovieLens dataset: build an effective collaborative filtering model from scratch**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Today we are going to start what I would kind of call like the second half of this course, so the first half of this course is being like getting through like these are the applications that we can use this for here's kind of the code you have To write, here's a fairly high level ish description of what it's doing and we're kind of we're kind of done for that bit and what we're now going to do is go in reverse we're going to go back over all of those exact same things again, but This time we're going to dig into the detail of every one and we're going to look inside the source code of the first idea library to see what it's doing and try to replicate that. So in a students like there's not going to be a lot more best practices to show you, like I've kind of shown you the best best practices. I know, but I feel like for us to now build on top of those to debug those models to come back to part two, where we're going to kind of try out some new things. You know it really helps to understand. What's going on behind the scenes? Okay, so the goal here today is we're going to try and create a pretty effective collaborative filtering model almost entirely from scratch. So we'll use the kind of we'll use pytorch as a automatic differentiation tool and there's a GPU programming tool and not very much else. We'll try not to use its neural net features we'll try not to use fastai library anymore than necessary. So that's the goal.

So let's go back and you know we only very quickly know collaborative filtering last time. So, let's, let's go back and have a look at collaborative filtering, and so we're going to look at this movie lens data set. So the movie lens data set basically is a list of ratings. It's got a bunch of different users that are represented by some ID and a bunch of movies that are represented by some ID and rating. It also has a timestamp. I haven't actually ever tried to use this. I guess

this is just like what what time did that person read that movie? So that's all we're going to use for modelling is three columns. User ID movie, ID and rating, and so thinking of that in kind of structured data terms, user ID and movie ID would be categorical variables. We have two of them, and rating would be a with the independent variable, we're not going to use this for modeling, but we can use it for looking at stuff later. We can grab a list of the names of the movies as well and reproduce. This genre information I haven't tried to be interested if, during the week, anybody tries it and finds it helpful. My guess is, you might not find it helpful, we'll see. So in order to kind of look at this better. I just grabbed the users that have watched the most movies and the movies that have been the most watched and made a crosstab of it right.

So this is exactly the same data, but it's a subset and now, rather than being user movie rating, we've got user movie rating, and so some users haven't watched some of these movies. That's why some of these okay, then I copied that into Excel and you'll, see. There's a thing called collab your XLS. If you don't see it there now I'll make sure I put it there back tomorrow, and here is where I've copied that table okay. So as I go through this like setup of the problem and kind of how its described and stuff, if you're ever feeling lost, feel free to ask either directly or through the forum, if you ask through the forum and somebody answers there, I want you to answer It here, but if somebody else asks a question you would like answered, of course, just like it and your network keep an eye out for that, because kind of that's we're digging in to the details of what's going on behind the scenes, it's kind of important that At each stage you feel like okay, I can see

3. 00:12:15

- **Why a matrix factorization and not a neural net ?**
- **Using Excel solver for Gradient Descent 'GRG Nonlinear'**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

What's going on okay, so we can actually not going to build a neural net to start with. Instead we're going to do something called a matrix, factorization, the reason we're not going to build a neural net to start with is that it so happens. There's a really really simple kind of way of solving these kinds of problems which I'm going to show you, and so, if I scroll down I've, basically what I've got here is the same, the same thing, but this time these are my predictions rather than my actuals And I'm going to show you how I created these predictions? Okay, so here my actuals right here, my predictions and then down here we have our score, which is the sum of the different squared average square root? Okay, so this is. I are MSE down here. Okay, so on average we're randomly initialized model is out by 2.8. So let me show you what this model is and I'm going to show you by saying: how do we guess how much user ID number 14 likes movie ID number 27 and the prediction here? This is just at this stage. This is still random is 0.9 1. So how we calculate 0.9 1 - and the answer is we're taking it as this vector here, dot product with this vector here so dot product means 0.71 times, 0.1, 9 plus 0.8. 1 times. Point 6, 3 plus point 7, volt plus point 3, 1 and so forth and in you know, linear algebra speak because one of them is a column and one of them is a row. This is the same as a matrix product, so you can see here.

I've used the Excel fashion, matrix multiplier and that's my prediction. Having said that, if the original rating doesn't exist at all, then I'm just going to set this to 0 right because, like there's,

no error in predicting something that hasn't happened. Okay, so what I'm going to do is I'm basically going to say alright, everyone of my right rate, my predictions is not going to be a neural net. It's going to be a single matrix multiplication all right now, the matrix multiplication that it's doing is basically in practice is between like this matrix and this matrix right. So each one of these is a single part of that, so I randomly initialize these. These are just random numbers that I've just pasted in here, so I've basically started off with two random matrices, and I've said, let's assume, for the time being, that every rating can be represented as the matrix product of those two. So then in Excel. You can actually do a gradient descent. You have to go to your options to the add-ins section and check the box to say turn it on and once you do, you'll see, there's something there called solver and if I go solver it says: okay, what's your objective function and you just choose the cell? So in this case we chose the cell that contains that repeats, grade error, and then it says: okay, what do you want to change? And you can see here, we've selected this matrix and this matrix, and so it's going to do a gradient descent for us by changing these matrices to try and in this case minimize this min minimize this Excel so right, GRG nonlinear is a gradient just yet so I'll say solve and you'll see it starts at 2.8 and then down here you'll see that numbers drain down.

It's not actually showing us what it's doing, but we can see that the numbers going down. So this has kind of got a near or nettie feel to it in that we're doing like a matrix product and we're doing a gradient descent. But we don't have a nonlinear layer and we don't have a second linear layer on top of that. So we don't get to call this deep learning so things where people do like deep learning, each things where they have kind of matrix products and gradient descents. But it's not deep. People tend to just call that shallow learning. Okay, so we're doing this chattering yeah all right, so I'm just going to go ahead and press escape to stop it because I'm sick of waiting - and so you can see - we've now got down to the 0.39 all right. So, for example, it guessed that movie 72 for sorry movie, 27 for user. Seventy two would get 4.4 for rating 2772 and actually got a four ready. So you can see like it's it's it's doing something quite useful. So why is it doing something quite useful? I mean something to note here is the number of things we're trying to predict here? Is there's 225 of them right and the number of things we're using to predict is that times two so hundred and fifty of them? So it's not like we can just exactly fit. We actually have to do some kind of machine learning here.

So basically, what this is saying is that there does seem to be some way of making predictions in this way, and so for those of you that have done some linear algebra, and this is actually a matrix decomposition normally in linear algebra. You would do this using a analytical technique or using some techniques that are specifically designed for this purpose, but the nice thing is that we can use gradient descent to solve pretty much everything, including this. I don't like to so much think of it from a linear algebra point of view, though I like to think of it from an insured point of view, which is this, let's say movie sorry, let's say movie id 27 is Lord of the Rings part 1, and, Let's say move and so let's say we're trying to make that prediction for user 2072. Are they going to like Lord of the Rings, part 1, and so conceptually that particular movie? Maybe there's like this 4, so there's 5 numbers here and we could say like well what, if the first one was like, how much is it sci-fi and fantasy, and the second one is like how recent a movie and how much special effects is there? You know, and the one at the top might be like how dialogue-driven is it right, like let's say those kind of five, these five numbers represented particular things about the movie, and so, if that was the case, then we could have the same five numbers for the User saying like ok, how much does the use of like sci-fi fantasy? How much does the user, like modern, modern, CGI, driven movies? How much does this give us a like dialogue, different movies, and so, if you then took that cross-

product, you would expect to have a good model right would expect to have a reasonable reading now the problem is, we don't have this information for each user? We don't have the information for each movie, so we're just going to like assume that this is a reasonable kind of way of thinking about this system and, let's, unless stochastic gradient, descent, try and find these models right.

So so, in other words, these these factors, we call these things, factors these factors and we call them factors because you can multiply them together to create this, not they're factors and how many addresses these factors we call them latent factors because they're not actually. This is not actually a vector that we've like named and understood and like entered in manually, we've kind of assumed that we can think of movie ratings. This way, we've assumed that we can think of them as a dot product of some particular features about a movie and some particular features of to look what users, like those kinds of movies right and then we've used gradient descent to just say: okay, try and find Some numbers that work, so that's that's, basically the technique right and it's kind of the end and the entirety is in this printing right. So that is collaborative filtering using what we call probabilistic matrix factorization and, as you can see, the whole thing is easy to do in an excel spreadsheet and the entirety of it really is this single thing, which is a single matrix multiplication plus randomly initializing? If it would be better to cap this to 0 and 5 - maybe yes yeah and we're gon na do that later. Right, there's a whole lot of stuff. We can do to improve this. This is like our simple as possible, starting point all right, so so what we're going to do now is we're going to try and implement this in Python and run it on the whole data set.

Another question is: how do you figure out how many you know how it's clear: how long are the matrix five yeah yeah, so something to think about, given that this is like movie 49 right and we're looking at a rating for movie 49. Think about this. This is actually at embedding matrix, and so this length is actually the size of the embedding matrix. I'm not saying this is an analogy. I'm saying it literally. This is literally an embedding mattress. We could have a one hot encoding where 72, where a one is in the 72nd position, and so we'd like to look it up and it would return this list of five numbers. So the question is actually: how do we decide on the dimensionality of our embedding vectors and the answer to that question is we have no idea, we have to try a few things and see what was the underlying concept? Is you need to pick an embedding dimensionality, which is enough to reflect the kind of true complexity of this causal system, but not so big that you have too many parameters that it could take forever to Tehran or even

4. 00:23:15

- **What are the negative values for ‘movieid’ & ‘userid’, and more student questions**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

With regularization in my overfit, so what does it mean when the factor is negative, that the factor being negative in the movie case would mean like? This is not dialogue-driven? In fact, it's like the opposite dialogue here is terrible. A negative for the user would be like. I actually dislike modern CGI movies, so it's not from zero to whatever it's the range of score it'd be negative. This is a range of score, even like no net Maxim. No there's no constraints at all here. These are just standard. Embedding matrices questions. So first question is: why do

what why can we trust this embeddings because, like if you take a number six, it can be expressed as 1 into 6 or like 6 into 1 or 22 3 & amp 3 into 2? All so you're saying like we could like reorder these higher. Hardly the value itself might be different as long as the product is something well, but you see we're using gradient descent to find the best numbers so like once, we've found a good minimum. The idea is like yeah, there are other numbers, but they don't give you as good an objective value and, of course we should be checking that on a validation set really which we'll be doing in the Python version. Okay - and the second question is when we have a new movie or a new user to be a 30 trainer model. That is a really good question and there isn't a straightforward answer to that time.

Permitting will come back, but basically you would need to have like a kind of a new user model or a new movie model that you would use initially and then over time. Yes, you would then have to retrain the model so, like I don't know if they still do it, but Netflix used to have this thing that when you were first on boarded on Netflix, it would say like what movies do you like and you'd have to go Through and let's say a bunch of movies you like, and it would then my train is moral. Just find the nearest movie yeah, you could use nearest neighbors for sure, but the thing is initially, at least in this case we have no columns to describe a movie. So if you had something about like the movies genre release date, who was in it or something you could have some kind of non collaborative filtering model and that's kind of what I meant a new movie model. You have to have some some kind of

5. 00:26:00

- **Collaborative filtering notebook, 'n_factors=', 'CollabFilterDataset.from_csv'**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Predictors, okay, so a lot of this is going to look familiar and and the way I'm going to do. This is again it's kind of this top-down approach, we're going to start using a few features of pytorch and fastai, and gradually we're going to redo it a few times in a few different ways, kind of doing a little bit deeper, each time um regardless. We do need a validation set, so we can use our standard cross-validation indexes approach to grab a random set of ID's. This is something called weight decay which we'll talk about later in the course for those of you that have done some machine learning. It's l2 regularization. Basically - and this is where we choose - how big a embedding matrix do we want okay? So again, you know: here's where we get our model data object from CSV passing in that ratings file, which remember looks like that. Okay, so you'll see like stuff tends to look pretty familiar after a while, and then you just have to pass in the. What are your rows effectively, what are your columns effectively and what are your values effectively? Alright, so any any collaborative filtering recommendation system approach, there's basically a concept of like you know a user and an item now they might not be users and items like if you're doing the Ecuadorian, groceries, competition.

There are stores and items and you're trying to predict how many things are you going to sell at this store of this type, but, generally speaking, just this idea of like you've got a couple of kind of high cardinality, categorical variables and something that you're measuring and You're kind of conceptualizing and saying okay, we could predict the rating. We can predict the value by doing this this dot. For that, interestingly, this is kind of relevant to that that last question or suggestion an identical way to think about this. What I've expressed this is to say

when we're deciding whether user 72 will like movie twenty-seven, it's basically saying which other users liked movies that 72 liked and which other movies were liked by people like you, user 72. It turns out that these are basically two ways of saying the exact same thing. So basically, what collaborative filtering is doing you know kind of conceptually is to say okay, this movie and this user, which other movies are similar to it in terms of like similar people enjoyed them and which people are similar to this person based on people that, like The same kind of movies, so that's kind of the underlying structure at any time. There's an underlying structure like this. That kind of collaborative filtering approach is likely to be useful. Okay, so so you yeah so there's basically two parts. The two bits of your thing that you're factoring and then the the value of the dependent variable.

So as per usual, we can take our model data and ask for a learner from it and we need to tell it what size of any matrix to use. How many sorry, what validation set index is to use what batch size to use and what optimizer to use and we're going to be talking more about optimizes? Surely we want to Adam today, Adam next week or the week after, and then we can go ahead and say fit alright and it all looks pretty similar interest is usually interestingly, I only had to do three pops, like this kind of model, seem to Train super Quickly, you can use the learning rate finder as per usual. All the stuff you're familiar with will work fine, and that was it so this talk, you know about two seconds: the Train there's no free trained anything's here this is from random from scratch. Okay, so this is our validation set and we can compare it. We have. This is a mean squared error, not a root mean squared error, so we can take a square root. So with that last time I ran it was point, seven, seven, six and that's 0.88 and there's some benchmarks available for this data set and when I scrolled through and found the bench the best benchmark. I could find here from this recommendation system specific library. They had point nine one, so we've got a better loss in two seconds already. So that's good! So, that's basically how you can do collaborative filtering with the faster I library without thinking too much, but so now we're going to dig in and try and rebuild that we'll try and get to the point that we're getting something around 0.7. Seven point seven eight from scratch, but if you want to do this yourself at home, you know without worry about the detail.

That's you know those three lines of code: here's what you need! Okay, so we can get the predictions in the usual way and you know we could. For example, plot SNS is Seabourn, see one's a really great flooding library. It sits on top of matplotlib, it actually leverages matplotlib. So anything you learn about matplotlib will help you with SIBO, and it's got a few like nice. Little plots like this joint plot here is: I'm doing predictions against against actuals. So these are my actual season. My predictions and you can kind of see the the shape here is that, as we predict higher numbers, they actually are higher numbers, and you can also see the histogram of the predictions and a histogram of the ashes. So that's kind of floating. That is to show you another interesting visualization. Would you please explain the n factors why it's set to 50? It's set to 50 because I tried a few things in the world. It's the dimensionality of the embedding images or to think for it. Another way it's like how you know, rather than five, it's fit Jeremy. I have a question about suppose that your recommendation system is more implicit, so you have zeros or ones instead of just actual numbers right. So basically, we would then need to use a classifier instead of regresa. I have to sample the negative or something like that. So if you don't have it, which is up once, let's say like just kind of implicit feedback - oh I'm not sure we'll get to that.

One in this class, but what I will say is like in the case that you just doing classification rather than regression. We haven't actually built that in the library, yet maybe somebody this week that wants to try adding it. It would only be a small number of lines of code. You basically

have to change the activation function, to be a sigmoid, and you would have to change the criterion or the loss function to be cross-entropy rather than rmse, and that will give you a classifier rather than a regresar. How those are the only things you'll have to change, so hopefully, somebody this week won't take up that challenge and by the time we come back next week. We've all have that working. Ok, so I said that we're basically doing a dot product right or you know a dot product is kind of the vector version I guess of this matrix product, so we're basically doing

6. 00:34:05

- **Dot Product example in PyTorch, module 'DotProduct()'**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Each of these things times each of these things and then add it together. So let's just have a look at how we do that in Python, so we can create a tensor in pytorch. Just using this little capital. T thing you can just say: that's the first day I version the full version is torch dot from I'm pie or something, but I've got to set up, so you can possibly pass in even a list of lists, so this is going to create a torch tensor With one two three four and then here's a torch tensor with two to ten ten. Ok, so here are two more chances, I didn't say doc, CUDA so they're, not on the GPU they're sitting on the CPU, just FYI. We can multiply them together, right and so anytime. You have a mathematical operator between tensors in numpy or pipe torch. It will do element wise assuming that they're the same dimensionality, which they are they're both to about two okay, and so here, we've got 2 by 2. Is 4 3 by 10 is 30 and so forth? Ok, so there's a a times B. So, if you think about basically what we want to do here is we want to take ok, so I've got 1 times. 2 is 2 2 times. 2 is 4, 2 plus 4 is 6, and so that is actually the dot product between 1 2 & amp, 2 4, and then here, we've got 3 by 10 is 34 by 40. Sorry, 4 by 10 is 40, 30 and 40 and 70 so in other words a times B, dot some along the first dimension. So that's summing up the columns, in other words, across a row.

Okay, this thing here is doing the dot product of each of these rows with each of these rows, so it makes sense - and obviously we could do that with you know some kind of matrix multiplication approach, but I'm trying to really do things with this little special Case stuff as possible: ok, so that's what we're going to use for our dot products from now on. So basically all we need to do now is remember. We have the data we have is not in that crosstab format, so in excel. We've got it in this crosstab format, but we've got it here in this listed format: here's our movie rating user movie revenue. So conceptually we want to be like looking up this user into our embedding matrix to find their 50 factors looking up that movie to find their 50 factors and then take the dot product of those two 50 long vectors. So, let's do that to do it, we're going to build a layer, our own custom, neural net layer? That's not right! So the the the more generic vocabulary we call. This is we're going to build a high torch module. Okay, so a pytorch module is a very specific thing. It's something that you can use as a layer and a neural net. Once you've created your own height watch module, you can throw it into a mirror on it and a module works by assuming we've already got once a cordon model.

You can pass in some things in parentheses and it will calculate it right so, assuming that we already have a modular product, we can instantiate it like so to create our product object, and we can basically now treat that like a function right. But the thing is it's not just a function because we'll be able to do things like take derivatives of it stack them up together into a big

stack of neural network layers, blah blah blah. So it's basically a function that we can kind of compose. Very conveniently so here, how do we define a module which, as you can see here, returns a dot product? Well, we have to create a Python class, and so, if you haven't done oo before you're going to have to learn because all my torch modules are written in Python oo and that's one of the things I really like about pytorch - is that it doesn't reinvent Totally new ways of doing things by tensorflow does all the time in pytorch that you know really tend to use pythonic ways to do things. So in this case, how do you create? You know some kind of new behavior? You create a Python plus, it's so Jeremy suppose that you have a lot of data, not just a little bit of data you can have in memory. Will you be able to use fossae I to solve glory filtering? Yes, absolutely it's! It uses mini-batch stochastic gradient descent, which does have a batch at a time. The this particular version is going to create a pandas data frame and pandas data frame has to live in memory.

Having said that, you can get easily 512 gig, you know instances on Amazon so like if you had a CSV that was bigger than 512 gig. You know that would be impressive. If that did happen, I guess you would have to instead save that as a be calls array and create a slightly different version that reads from a because array just streaming in or maybe from a desk data frame, which also so it would be easy to do. I don't think I've seen real-world situations where you have 512 gigabyte collaborative filtering matrices, but yeah. We can do it. Okay now this is pytorch specific. This next bit is that when you define like the actual work to be done, which is here return user times movie dot, some you have to put it in a special method called forward. Okay - and this is this idea that, like it's very likely you're on that right in a neural net, the thing where you calculate the next set of activations is called the the forward pass and so that's doing a forward calculation. The gradients is called the backward calculation. We don't have to do that because pytorch calculates that automatically, so we just have to define forward. So we create a new class. We define forward, and here we write in our definition of dot product. Ok, so that's it so now that we've created this class definition, we can instantiate our model right and we can call our model and get back the numbers be expected.

Okay, so that's it! That's how we create a custom, pytorch layer, and if you compare that to like any other library around pretty much, this is way easier. Basically, I guess because we're leveraging what's already in person, so let's go ahead and now create a more complex module and we're going to basically do the same thing. We've got to have a forward again we're going to have our users x, movies, dot sum, but we're

7. 00:41:45

- **Class 'EmbeddingDot()'**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Going to do one more thing before hand, which is we're going to create two embedding matrices and then we're going to look up our users and our movies in those inventing matrices. So let's go through and and do that. So the first thing to realize is that the uses, the user IDs and the movie IDs may not be contiguous. You know like they're, maybe they start at a million and go to a million in 1000. So right. So if we just used those IDs directly to look up into an embedding matrix, we would have to create an embedding matrix of size, 1 million 1000 right, which we don't want to do so. The first thing I do is to get a list of the unique user

IDs, and then I create a mapping from every user ID to a contiguous integer. This thing I've done here where I've created a dictionary which maps from every unique thing to a unique index is well worth studying during the week because, like it's is super super handy, it's something you very very often have to do in all kinds of machine learning. All right - and so I won't go through it here - it's easy enough to figure out if you can't figure it out, just ask on the forum anyway. So once we've got the mapping from user to a contiguous index, we then can say: let's now replace the user ID column with that contiguous index right, so pandas dot apply applies an arbitrary function and python lambda is how you create an anonymous function on the fly And this anonymous function simply returns the NS through the same thing for movies, and so after that we now have the same ratings table we had before, but our IDs have been mapped to contiguous integers.

Therefore, they're things that we can look up into an embedding matrix. So let's get the count of our users in our movies and let's now go ahead and try and create our Python version of this okay. So earlier on, when we created our simplest possible pytorch module, there was no like state. We didn't need a constructor, because we weren't like saying how many users are there or how many movies are there or how many factors do we want or whatever right anytime. We want to do something like this, where we're passing in and saying, we want to construct our module with this number of users and this number of movies. Then we need a constructor for our class and you create a constructor in Python by defining a dunder init underscore underscore init underscore underscore yet special name, so this just creates a constructor, and if you haven't done over before you wanted to do some study during the Week, but it's pretty simple idea: this is just the thing that when we create this object, this is what gets wrong. Okay, again special python thing. When you create your own constructor, you have to call the parent class constructor and if you want to have all of the cool behavior of a PI porch module, you get that by inheriting from an end, module neural net module. Okay. So, basically, by inheriting here and calling the superclass constructor, we now have a fully functioning pytorch layer.

Okay, so now we have to give it some behavior, and so we give us some behavior by storing some things in it all right. So here we're going to create something called self dot you users, and that is going to be an embedding layer. A number of rows is an user's number of columns is in factors. So that is exactly this right. The number of rows is M users. Number of columns is inventors and then we'll have to do the same thing for movies. Okay, so that's going to go ahead and create these two randomly initialized arrays. However, when you randomly initialize over an array, it's important to randomly initialize it to a reasonable set of numbers like a reasonable scale right. If we randomly initialize them from like naught to a million, then we would start out - and you know these things would start out. Being like, you know, billions and billions of writing, and that's going to be very hard to do gradient descent on. So I just kind of manually figured here like okay about what size numbers are going to give me about the right readiness, and so we don't. We know we did ratings between about normal five. So if we start out with stuff between about naught and 0.05, then we're going to get ratings of about the right level. You can easily enough, like that, calculate that in in neural nets, there are standard algorithms for basically doing doing that calculation.

8. 00:47:05

- **Kaiming He Initialization (via DeepGrid),**
- **sticking an underscore '_' in PyTorch, 'ColumnarModelData.from_data_frame()', 'optim.SGD()'**

And the basic the key algorithm is something called initialization from climbing her, and the basic idea is that you take the yeah. You basically set the weights equal to a normal distribution with a standard deviation, which is basically inversely proportional to the number of things in the previous layer and so in our previous layer. So in this case, we basically if you basically take that nor to 0.05 and multiply it by the fact that you've got 40 things with a 40 or 50 things coming out of it. 50 50 things coming out of it and then you're going to get something about the right size. pytorch has already has like her initialization class they're like we don't in normally in real life, have to think about this. We can just call the existing initialization functions, but we're trying to do this all like from scratch here: okay without any special stuff going on so there's quite a bit of pytorch, notation here, so self dot. U we've already set to an instance of the embedding class. It has a dot weight attribute which contains the actual the actual embed images, so that contains this. The actual embedding matrix is not a tensor. It's a variable. A variable is exactly the same as a tensor. In other words, it supports the exact same operations as a tensor, but it also does automatic differentiation. That's all a variable is basically to pull the tensor out of a variable. You get its data attribute, okay, so this is so.

This is now the tensor of the weight matrix of the self dot you're inventing, and then something that's really handy to know is that all of the tensor functions in pytorch. You can stick an underscore at the end, and that means do it in place all right, so this is say, create a random, uniform, random number of an appropriate size for this tensor and don't return it but actually fill in that matrix unless okay. So that's a super handy thing to know about. I mean it wouldn't be rocket science. Otherwise we would have to have gone , okay, here's the non in-place version. That's what saves us! Some typing saves us some screen noise. That's all! Okay! So now we've got our randomly initialized, embedding weight, matrices and so now the forward, I'm actually going to use the same columnar model data that we used for Russman, and so it's actually going to be passed, both categorical variables and continuous variables. And in this case there are no continuous variables, so I'm just going to grab the 0th column out of the categorical variables and call it users and the first column and call it movies. Okay. So I'm just kind of too lazy to create my own. I've lots to do about too lazy out that we do have a special class with this, but I'm trying to avoid creating a special class so just going to leverage this columnar model data plus okay. So we can basically grab our user and movies mini-batches right and remember.

This is not a single user in a single movie. This is going to be a whole mini batch of them. We can now look up that mini batch of users in our embedding matrix. U and the movies in are embedding matrix. Okay, so this is like exactly the same is just doing an array lookup to grab the user ID numbered value, but we're doing that a whole mini batch at a time right, and so it's because pytorch can do a whole mini batch at a time with Pretty much everything that we can get really easy speed up. We don't have to write any loops on the whole to do everything through our mini batch and in fact, if you do ever loop through your mini batch manually, you don't get GPU acceleration. That's really important to know right, so you never want to loop. Have a for loop going through your mini batch. You always want to do things in this kind of like whole mini batch at a time, but pretty much everything imply torch. Does things are holding events at a time, so you shouldn't have to worry about it and then here's our product just like before right so having to find that I'm now going to go ahead and say alright, my X values is everything except the rating and the Timestamp in my writings table, my Y is

my rating and then I can just say: okay, let's grab a model data from a data frame using that X and that Y - and here is our list of categorical variables: okay and then so, let's now instantiate that pytorch object.

Alright, so we've now created that from scratch, and then the next thing we need to do is to create an optimizer. So this is part of pytorch. The only fastai thing here is this line right because that's like, I don't think showing you how to build data sets and data load is interesting enough. Really, we might do that in part two of the course and it's actually so straightforward. Like a lot of, you are already doing it on the forums. So I'm not going to show you that in this part, but if you're interested feel free to talk on the forums about it, but I'm just going to basically take the thing that feeds us. Data is a given particularly cuz. These things are so flexible right. You know if you've got stuff enough data frame, you can just use this, you don't have to rewrite it. So that's the only fastai thing we're using. So this is a pytorch thing, and so optiom is the thing and pytorch that gives us an optimizer will be learning about that very shortly. So it's actually the thing. That's going to update our weights. pytorch, calls them the parameters of the model. So earlier on, we set model, equals embedding dot, blah blah right and because embedding dot derives from NN module. We get all of the pytorch module behavior and one of the things we got for free is the ability to say got parameters. So that's pretty.

That's pretty any right, that's the thing that basically is going to automatically give us a list of all of the weights in our model that have to be updated, and so that's what gets passed to the optimizer. We also passed the optimized at the learning rate, the weight decay which we'll talk about later and momentum that we'll talk about later. Okay, one other thing that I'm not going to do right now, but we will do later, is to write a training loop. So the training loop is a thing that lives for each mini batch and updates the weight to subtract the gradient times. The moment there's a function in fastai, which is the training loop, and it's it's pretty simple here. It is right for a POC in epochs. This is just the thing that shows a progress bar so ignore this for X, comma Y, in my training data loader calculate the loss print out the lots you know in a progress bar call any callbacks you have and at the end call the call the metrics On the validation, alright, so there's there's just eh Apoc go through each mini batch and do one step of optimizer step is basically going to take advantage of this optimizer, but we'll be writing that from scratch shortly. So this is notice we're not using a learner. Okay, we're just using a hi book module, so this this fit thing, although it's passed to a part of fastai, it's like lower down the layers of abstraction. Now this is the thing that takes a regular high torch model.

So if you ever want to like skip as much faster eye stuff as possible, like you've, got some high torch model, you've got some code on the internet. You basically want to run it that you don't want to write your own training loop. Then this is. This. Is what you want to do? You want to call fast, a high speed version, and so what you'll find is like the library is designed so that you can kind of dig in at any layer abstraction you like right and so at this layer of abstraction you're. Not going to get things like stochastic gradient descent with restarts you're not going to get like differential learning rates like all that stuff. That's in the learner like you could do it, but you'd have to write it all about by hand yourself. Alright and that's the downside of kind of going down to this level of abstraction, the upside is that, as you saw, the code for this is very simple: it's just a simple training loop. It takes a standard 5 torch model, so this is like this is a good thing for us to use here. We can, we just call it, and it looks exactly like what we used to see all right. We got our validation and training loss for the 3 e bus now you'll notice that we wanted something around 0.76, so we're not there. So, in other words, the the the default fastai collaborative, dory rhythm is doing something smarter

than this.

So we're going to try and do that, one thing that we can do since we're calling our you know this lower level fifth function, there's no learning rate and kneeling. We could do our own learning rate annealing, so you can hear it see here. There's a first day I function called set learning rates, you can pass in a standard, height, watch optimizer and pass in your new learning rate and then call fit again, and so this is how we can let manually do a learning rate schedule, and so you can See, we've got a little bit better 1.13, where you still got a long way to go okay, so I think what we might do is we might have a seven minute break and then we're going to come back and try and improve this core of it. For those who are

- Pause

9. 00:58:30

- **'fit()' in 'model.py' walk-through**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Interested somebody was asking me the break for a kind of a quick walkthrough, so this is totally optional. But if you go into the first day, I library there's a model py file and that's where fit is which we're just looking at, which goes through each epoch. In epochs and then goes through each x and y in the mini batch, and then it calls this step function. So the step function is here and you can see the key thing is it calculates the output from the model, the models for M right, and so, if you remember our dot product, we didn't actually call model dot forward. We just called model parentheses and that's because the N n dot module automatically. You know when you call it as if it's a function, it passes it along to forward okay. So, that's that's what that's doing there right and then the rest of this world will learn about shortly, which is basically doing the the loss function and the backward pass. Okay. So, for those who are interested, that's that's kind of gets you a bit of a sense of how the cone it's structured. If you want to look at it and, as I say like the the faster I code is designed to both be world-class performance, but also pretty easy to read so like feel free, like take a look at it and if you want to know what's going on. Just ask on the forums and if you you know, if you think, is anything that could be clearer, let us know because yeah the code is definitely now we're going to be digging into the code or in law.

Okay, so

10. 01:00:30

- **Improving the MovieLens model in Excel again,**
- **adding a constant for movies and users called "a bias"**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Let's try and improve this a little bit and, let's start off by improving it in Excel, so you might have noticed here that we've kind of got the idea that use a 72. You know like sci-fi, modern movies, with special effects. You know whatever and movie number 27 is sci-fi and that

special effects so much dialogue, but we're missing an important case which is like user 72 is pretty enthusiastic on the hall and on average rates things higher and Highland. You know and movie 27. You know it's just a popular movie, you know it's just on average its higher. So what would really like is to add a constant for the user and a constant for the movie and remember in neural network terms. We call that a bias, that's what we want to add a bias, so we could easily do that and if we go into the bias tab here, we've got the same data as before, and we've got the same latent factors as before, and I've just got one Extra row here and one extra column here - and you won't be surprised here - that we now take these same matrix multiplication as before, and we add in that - and we add in that - okay - so that's bias so other than that we've got exactly the same loss function Over here and so just like before, we can now go ahead and solve that, and now our changing variables include the bias and we can say solve, and if we leave that for a little while it will come to a better result than we had before.

Okay, so that's the first thing: we're

11. 01:02:30

- **Function 'get_emb(ni, nf)' and Class 'EmbeddingDotBias(nn.Module)', '.squeeze()' for broadcasting in PyTorch**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Going to do to improve our model and there's really very little show just to make the code a bit shorter. I have to find a function called get embedding which takes a number of inputs and a number of factors. So the number of rows and the embedding matrix, Nomos they're both medications creates the embedding and then randomly initializes it. I don't know why I'm doing negative to positive here and it zeroed. Last time, honestly, it doesn't matter much as long as it's in the right ballpark and then we return that initialized emitting. So now we need not just our users by factors which are Chuck into you, our movies, by factors which I've shocked into M. But we also need users by one which will put into UV user bias and movies by one which will put into the movie bias okay. So this is just doing a list. Comprehension going through each of the tuples create an embedding for each of them and putting them into these things. Okay, so now our forward is exactly the same as before: u times M sub, I mean this is actually a little confusing because we're doing it into two steps. Maybe they make it a bit easier. Let's pull this out. Put it up here, put this in parentheses! Okay, so maybe that looks a little bit more familiar all right, you times, n dot, some that's the same dot product and then here it is going to add in our user, pious and our movie bus dot squeeze is the pytorch thing that adds an additional Unit axis, that's not going to make any sense if you haven't done broadcasting before I'm not going to do a broadcasting in this course, because we've already done it and we're doing it in the machine learning course.

But basically in in short, broadcasting is what happens when you do something like this? Where um is a matrix, you be self-taught, you, the users is a is a vector. How do you add a vector to a matrix and basically, what it does is it duplicates the vector so that it makes it the same size as the matrix and the particular way, whether it duplicates it across columns or down rows or how it does? It is called broadcasting the broadcasting rules are the same as numpy Pytor didn't actually used to support broadcasting, so I was actually the guy who first added broadcasting to pytorch using an ugly hack and then the pipe or authors did an

awesome job of supporting it. Actually, inside the language, so now you can use the same broadcasting operations in five torches non-player. If you haven't dealt with this before, it's really important to learn it because, like it's, it's kind of the most important fundamental way to do computations quickly in the high-end paid warship. It's the thing that lets you not have to do loops. How could you imagine here if I had to look through every row of this matrix and add each did you know this vector to every row? It would be slow, the you know a lot more code and the idea of broadcasting. It actually goes all the way back to APL, which was a language designed in the 50s by an extraordinary guy called Ken Iverson, yeah APL was originally designed or written out as a new type of mathematical notation.

He has this great essay called notation as a tool for thought, and the idea was that, like really good, notation could actually make you think of better things and part of that notation. Is this idea of broadcasting I'm incredibly enthusiastic about it and we're gon na use? It plenty so either watch the machine, learning lesson or you know: google numpy broadcasting for information anyway. So basically it works reasonably intuitively we can add on. We can add the vectors to the matrix, all right.

12. 01:06:45

- **Squeashing the ratings between 1 and 5, with Sigmoid function**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Having done that, we're now going to do one more trick, which is, I think it was your net asked earlier about. Could we squish the ratings to be between one and five, and the answer is we could right and specifically what we could do is we could put it through a sigmoid function? All right so remind you. A sigmoid function looks like that right, and this is that's one. Okay, we could put it through a secret function, so we could take like four point: nine six and put it through a sigmoid function and like that you know that's kind of high, so it kind of be over here somewhere right and then we could multiply that Sigmoid, like the result of that by five, for example, all right - and in this case we want it to be between one and five right. So maybe we would multiply it by four and add one instance. That's the basic idea, and so here is that trick. We take the result, so the result is basically the the thing that comes straight out of the dot product, plus the addition of the biases and put it through a sigmoid function. Now in pytorch. Basically, all of the functions you can do to tensors are available inside this thing called capital F, and this is like totally standard in pytorch. It's actually called torch and or functional, but everybody, including all of the pipe torch, Doc's import, torch, start and end, are functional as capital F, all right, so capital, F, dot, sigmoid means a function called sigmoid that is coming from tortures, functional module right, and so that's Going to apply a sigmoid function for the result, so I squish them all between zero and one using that nice little shape, and then I can multiply that by five minus one plus four right and then add on one and that's gon na give me plumbing between One and five okay, so like there's, no need to do this, I could comment it out and it'll still work right, but now it has to come up with a set of calculations that are always between one and five right.

Where else, if I leave this in then it's like makes it really easy. It's basically like. Oh, if you think this is a really good movie, just calculate a really high number. It's a really crappy movies, low number and I'll make sure it's in the right regions. So, even though this is a neural network, it's still a good example of this kind of like, if you're doing any kind of

parameter, fitting, try and make it so that the thing that you want your function to return, it's like it's easy for it to return That, okay, so that's why we do that that function squishing. So we call this embedding dot bias, so we can create that in the same way as before you'll see here, I'm calling dr. to put it on the GPU because we're not using any learner stuff, normally it'll all happen for you, but we have to manually, say Put it on the GPU: this is the same as before, create our optimizer fit exactly the same as before, and these numbers are looking good, all right and again, we'll do a little change to our learning rate, learning rate schedule and we're down to 0.8. So we're actually pretty close, pretty close. So that's the key steps - and this is how this is, how most collaborative filtering is done and unit reminded me of an important point, which is that this is not strictly speaking a matrix factorization, because strictly a matrix, factorization would take that matrix by that matrix to Create this matrix and remembering anywhere that this is empty like here or here, we're putting in a zero right we're saying if the original was empty, put in a zero right now, normally you can't do that with normal matrix, factorization normal matrix factorization.

It creates the whole matrix, and so it was a real problem actually when people used to try and use traditional linear algebra for this, because when you have these sparse matrices like in practice, this matrix is not doesn't have many gaps because we picked the users that Watch the most movies and the movies that are the most watched. But if you look at the whole matrix, it's it's mainly empty and so traditional techniques treated empty is zero and so, like you basically have to predict a zero as if the fact that I haven't watched a movie means I don't like the movie. That's gives terrible answers, so this probabilistic, matrix factorization approach takes advantage of the fact that our data structure actually looks like this rather than that cross tab right, and so it's only calculating the loss for the user ID movie ID combinations that actually appear that's its. If you like, use red a1 movie, I think 102 9 should be 3. It's actually three and a half sauce is 0.5. Like there's nothing here, that's ever going to calculate a prediction or a loss for a

13. 01:12:30

- **What happened in the Netflix prize, looking at 'column_data.py' module and 'get_learner()'**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

User movie combination that doesn't appear in this table by definition, the only stuff that we can appear in a mini batch is what's in this table. Okay and like a lot of this happened, interestingly enough, actually in the Netflix price, so before the Netflix prize came along. There's probabilistic matrix factorization, it had actually already been invented, but nobody noticed all right and then, in the first year of the Netflix price, someone wrote this like really really famous blog post, where they basically said like hey check this out. Incredibly, simple technique works incredibly well when suddenly, all the net fix leaderboard entries - and so that's quite a few years ago now - and this is like now - every collaborative filtering approach. Does this? Not every collaborative filtering approach adds this sigmoid thing by the way. It's not like rocket science. This is this is not like the NLP thing we saw last week, which is like hey. This is a new state-of-the-art like this is, you know, not particularly uncommon, but there are still people that don't do this. It definitely helps a lot. I have to have this, and so actually you know what we could do is maybe now's a good time to have a look at the definition of this right. So the column data module contains all these definitions and we can

now compare this to the thing we originally used, which was whatever came out of collaborative data set all right.

So let's go to collab filter data set here it is and we called get learner all right, so we can go down to get Elena and that created a collab filter. Learner passing in the model from get model is get model, so created an embedding bias, and so here is embedding drop bias, and you can see here here. It is like it's the same thing. There's the embedding for each of the things. Here's our forward that does the you times, I dot some plus plus sigmoid. So in fact we have just actually rebuilt. What's in the past, our library, literally okay, it's a little shorter and easier because we're taking advantage of the fact that there's a special collaborative filtering data set. So we can actually we're getting past in the users and the items and we don't have to pull them out of cat Kant's, but other than that. This is exactly the same. So hopefully you can see like the faster you have. Ivory is not some inscrutable code containing concepts. You can never understand. We've actually just built up this entire thing from scratch ourselves, and so why did we get 0.76 rather than 0.8? You know, I I think it's simply because we used stochastic gradient descent with restarts or the cycle multiplier and an atom optimizer. You know like a few little training chase, some I'm looking at this and thinking that is.

We could totally improve this small, but maybe looking at the date and doing some tricks with the date, because this this is kind of a just, a regular kind of smaller, no way yeah. You can add more features, yeah exactly exactly so like now that you've seen this you could now you know, even if you didn't have embedding dot bias in a notebook that you've written yourself through some other model, that's in fastai. You could look at it in faster and be like, oh, that does most of the things that I'd want to do, but it doesn't deal with time, and so you could just go. Oh okay, let's grab it copy it. You know pop it into my notebook and, let's create you, know the better version all right and then you can start playing that and you can now create your own model class from the open source code here and so yeah. Your that's mentioning a couple things. We could do we could try and incorporate in time stamp. So we could assume that maybe well maybe there's just like some for a particular user over time users tend to get more or less positive about movies. Also remember, there was the list of genres for each movie. Maybe we could incorporate that so one problem is it's a little bit difficult to incorporate that stuff into this? Embedding bias.

14. 01:17:15

- **Creating a Neural Net version “of all this”, using the ‘movielens_emb’ tab in our Excel file, the “Mini net” section in ‘lesson5-movielens.ipynb’**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Model because it's kind of it's pretty custom right, so what we're going to do next is we're going to try to create a neural net version of this hey. So the basic idea here is we're going to take exactly the same thing as we had before. Here's our list of users right and here is Erin Bates, alright and here's our list of movies, and here is our embedded right and so, as you can see, I've just kind of transposed the movie ones so that so that they're all in the same orientation - and Here is our user movie rating but D cross tab, okay, so in the original format. So each row is a user movie rating okay. So the first thing I do is, I need to replace user 14 with that users contiguous in this right, and so I can do that in Excel using this match.

That basically says what you know: how far down this list do you have to go, and it said user 14 was the first thing in that list: okay, user 29 was the second thing in that list, so forth, okay. So this is the same as that thing that we did in our Python code, where we basically created a dictionary to master. So now we can for this particular user movie rating combination. We can look up the appropriate embedding right, and so you can see here what it's doing is it's saying all right, let's basically offset from the start of this list and the number of rows we're going to go down is equal to the user index and the Number of columns we're going to go across is one two three four or five okay, and so you can see what it does.

Is it creates point one nine point, six, three point three one here. It is point one, nine point: okay, so so this is literally modern. Embedding this but remember this is exactly the same as doing a one hot encoding right, because if instead this was a vector containing one zero, zero, zero, zero right - and we multiplied that by this matrix, then the only row it's going to return would be the first One okay, so so it's really useful to remember that embedding actually just is a matrix product. The only reason it exists. The only reason it exists is because this is an optimization. You know this, let's pipe or to know like okay. This is just a matrix multiply, but I guarantee you that you know this thing is one hard encoded. Therefore you don't have to actually do the matrix multiply. You can just do a directory of that. Okay, so that's literally all an embedding is. Is it is a computational performance thing for a particular kind of matrix multiplier all right, so that looks up that uses user and then we can look up that users movie all right. So here is movie ID movie ID four one, seven, which apparently is indexed number. Fourteen here it is here, so it should have been point. Seven, five point: four: seven! Yes, it is point, seven five point, plus it okay, so we've now got the user embedding and the movie embedding, and rather than doing a dot product of those two okay, which is what we do normally.

Instead, what? If we concatenate the two together into a single vector of length 10 and then feed that into a neural net, okay and so anytime, we've got you know a tensor of import activations or in this case a tensor of. Actually, this is a tensor of output activations. This is coming out of an embedding layer. We can chuck it in a neural net because neural Nets, we now know, can calculate anything - okay, including, hopefully collaborative filtering. So let's try that so here is our embedding net. So this time I have not bothered to create a separate bias, because instead the linear layer in pytorch already has a bias in it all right. So when we go NN Linea right, that's kind of draw this out. So we've got our! U matrix right, and this is the number of users - and this is the number of factors right and we've got our M matrix that so here's our number of movies and here's our again number of factors. Okay and so remember, we look up a single user. We look up a single movie and let's grab them and concatenate them together. Okay, so here's like the user part, here's the movie part and then let's put that through a matrix product right, so that number of rows here is going to have to be the number of users plus the number of movies right. Because that's how long that is and then the number of columns can be anything we want, because we're going to take that so in this case we're going to pick 10. Apparently. So it's pick 10 and then we're going to stick that through a rail you and then stick that through another matrix, which obviously needs to be of size 10. Here and then the number of columns is a size 1 because we want to predict a single rating.

Okay, and so that's our kind of flow chart of what's going on right, it is a standard, I'm called a one, hidden layer, neural net. It depends how you think of it like there's kind of an embedding layer, but because is linear, and this is linear. The two together is really one linear layer right this just a computational convenience, so it's really got one hidden layer because

it's got one layer before this nonlinear activation. So, in order to create a linear layer with some number of rows and some number of columns you just go in and on in the machine learning class this week we learnt how to create a linear layer from scratch by creating our own weight matrix and our Own biases, so if you want to check that out, you couldn't do so there right, but it's the same basic technique, we've already seen, so we create our embeddings. We create our two linear layers, that's all the stuff that we need to start with. You know really, if I wanted to make this more general, I would have had another parameter here called like num hidden, you know equals equals 10 and then this would be a parameter, and then you could like more easily play around with different numbers of activations. So when we say like okay in this layer, I'm going to create a layer with this many activations, all I mean assuming it's a fully connected layer is my linear layer has how many columns in its weight matrix that's how many activations it creates all right.

So we grab our users and movies, we put them through our embedding matrix and then we concatenate them together. Okay, so torch cat concatenate them together on the first dimension, so in other words, we concatenate the columns together to create longer rows. Okay, so that's concatenating! On dimension, one drop out we'll come back to her in a moment. We've got that briefly. So then, having done that, we'll put it through that linear layer, we had we'll do our value and you'll notice. That value is again inside our capital F and end up optional right. It's just a function, so remember, activation function are basically things that take one activation in and spit one activation out in this case, taking something that can have negatives or positives and truncate the negatives. To zero, that's what well you does and then here's a sigmoid. So that's that that is now a genuine neural network. I don't know if I get to call it deep. It's only got one hidden layer, but it's definitely a neural network all right, and so we can now construct it. We can put it on the GPU, you can create an optimizer for it and we can fit it now. You'll notice, there's one other thing: I've been passing to fit, which is what loss function? Are we trying to minimize okay? This is the mean squared error loss and again it's inside F, okay, pretty much all the functions are inside it, okay.

So one of the things that you have to pass fit is something saying like: how do you score it's what counts as good or bad, so it should. I mean now that we have a real neural net. Do we have to use the same number of embeddings for users and that's a great question? You don't know absolutely right, you don't and so, like we've got a lot of benefits here right, because if we, you know think about, you know we're grabbing a user embedding or concatenating it with a movie embedding, which maybe is like some different size, but then also, Perhaps we looked up the genre of the movie and, like you know, there's actually a embedding matrix of like number of genres by I don't know three or something and so like. We could then concatenate like a genre embedding and then maybe the timestamp is in here as a continuous number right, and so then that whole thing we can then feed into you know and you're on it all right and then at the end, remember a final non-linearity Was a sigmoid right, so we can now recognize that thing we did where we did sigmoid x max reading vote min reading, + blah blah blah is actually just another nonlinear activation function. Alright remember in our last layer we use generally different kinds of activation functions. So, as we said, we don't need any activation function at all right. We could just do that right, but by not having any nonlinear activation function, we're just making it harder. So that's why we put the sigmoid in there as well.

Okay, so we can then fit it in the usual way and there we go. You know, interestingly, we actually got a better score than we did with our this model, so I'll be interesting to try training. This with stochastic gradient descent with restarts and see if it's actually better, you know,

maybe you can play around with the number of hidden layers and the drop out and whatever else and see if you can come up with, you know, get a better answer than point. Seven: six ish, okay, so so general. So this is like if you were going deep into collaborative filtering at your workplace, whatever this wouldn't be a bad way to go. I could like I'd start out with, like oh okay: here's like a flat footed, dataset 30. In first day I get learner, there's you know not much, I can send it. Basically number of factors is about the only thing that I pass in. I can learn for a while. Maybe try a few different approaches and then you're like okay, there's like that's how I go if I use the defaults okay, how do I make it better and then I'd be like dig into the code and seeing like okay? Well, what? If Jeremy actually do here? This is actually what I want you know, and so one of the nice things about the neural net approach is that you know, as unit mentioned, we can have different numbers of embeddings. We can choose how many hidden and we can also choose, drop now right. So so what we're actually doing is we haven't just got real you that we're also going like okay, let's, let's delete a few things at random. Alright, let's drop out.

So in this case we were deleting after the first linear layer, 75 % of them all right and then after the second one in like 75 % of them, so we can add a whole lot of regularization. Yes, so you know this. It kind of feels like the this, this embedding net. You know you could you could change this again, we could like have it so that we can pass into the constructor. Well, if you're gon na make it look as much as possible like what we had before. We could surpass him. Peace, peace equals 0.75. Oh I'm not sure this is the best API, but it's not terrible. Probably what, since we've only got exactly two layers, we could say: p1 equals 0.75 v p2 v, and so then this will be P 1. This will be Peter, you know where we go and like. If you wanted to go further, you could make it look more like our structured data learner. You could actually have a thing this number of hidden, you know, maybe you could make a list, and so then, rather than creating exactly one hidden layer and one output layer, this could be a little loop that creates and hidden miners each one of the size you Want so like this is all stuff you can play with during the hearing the week if you want to - and I feel like if you've got like a much smaller collaborative children data set, you know, maybe you need like more regularization or whatever it's a much bigger One, maybe more layers would help. I don't know you know.

Iii haven't seen much discussion of this kind of neural network approach to collaborative filtering, but I'm not a collaborative filtering expert. So maybe it's maybe it's around, but that'd be

15. 01:33:15

- **What is happening inside the “Training Loop”, what the optimizer ‘optim.SGD()’ and ‘momentum=’ do, spreadsheet ‘graddesc.xlsm’ basic tab**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Interesting thing to try, so the next thing I wanted to do was to talk about the training loop, so what's actually happening inside the training loop, so at the moment we're basically passing off the actual updating of the weights to pytorches optimizer. But what I'm going to do is like understand what that optimizer is, is actually good and we're. Also. I also want to understand what this Momentum's him he's doing. So you'll find we have a spreadsheet called grab disk gradient descent, and it's kind of designed to be read left to right. Sorry right to left worksheet

was so the rightmost worksheet. Is some data right and we're going to implement gradient descent in Excel, because obviously everybody wants to do deep learning in it? Selman we've done collaborative filtering in Excel, we've done convolutions in Excel, so now we need SGD in Excel, so we can replace once and for all okay. So let's start by creating some data right and so here's you know, here's some independent. You know I've got one column of X's, you know and one column of wise and these are actually directly linearly related. So this is this is random right and this one here is equal to x, times, 2 plus 30. Ok, so let's try and use Excel to take that data and try and learn those parameters. Okay, that's going to be able. So, let's start with the most basic version of SGD, and so the first thing I'm going to do is I'm going to run a macro.

So you can see what this looks like so I'll hit run and it does five eight bucks under another five: eight bucks, another five, eight bucks, okay, so the first one was pretty terrible. It's hard to see so I'll just delete that first, one get better scaling. Alright, so you can see it actually, it's pretty constantly improving the loss. All right. This is the loss per pot all right. So how do we do that? So, let's reset it. So here is my X's and my y's, and what I do is I start out by assuming some intercept and some slope right. So this is my randomly initialized weights, so I have randomly initialized them both to one. You could pick a different random number if you like, but I promise that I randomly picked the number one twice there you go. It was a random number between one and one. So here is my intercept and slope. I'm just going to copy them over here right. So you can literally see this is just equal see. One here is equals c2 okay, so I'm gon na start with my very first row of data x equals 14 y equals 58 and my goal is to come up after I look at this piece of data. I want to come up with a slightly better intercept and a slightly better slope. Okay, so to do that, I need to first of all, basically figure out which direction is is down. In other words, if I make my intercept a little bit higher or a little bit lower, would it make my error a little bit better or a little bit worse? So, let's start out by calculating the error so to calculate the error.

The first thing we need is a prediction, so the prediction is equal to the interest at plus x times, so that is our zero hidden layer, neural network, okay, and so here is our era. It's equal to our prediction, our actual squared. So we could like play around with this. I don't want my error to be 18-49. I'd like it to be lower. So what if we set the intercepts to one point, one 18-49 goes to 1840. Okay, so a higher intercept would be better. Okay. What about the slope to increase that it goes from 1849 to 1730. Okay, a higher slope would be better as well, not surprising, because we know actually that there should be 30 into so one way to figure that out, you know encode, and this protein is to do literally. What I just did is to add a little bit to the intercept and the slope and see what happens and that's called finding the derivative through finite differencing right and so let's go ahead and do that. So here is the value of my error. If I add 0.01 to my intercept all right, so it's c4 plus 0.01 and then I just put that into my Lydian function and then I subtract, my actual all squared all right and so that causes my arrow to go down a bit. That's our increasing! My is that increasing will see for increasing the intercept. A little bit has caused my arrow to go down. So what's the derivative? Well, the derivative is equal to how much the dependent variable changed by divided by how much the independent variable changed by all right and so there it is right.

Our dependent variable changed by that that right and our independent variable we changed by 0.01. So there is the estimated value of the error dB. So remember when people talking about derivatives right, this is this is all they're doing, is they're saying what's this value, but as we make this number smaller and smaller and smaller and smaller as it as limits to zero, I'm not mad enough to think in terms of Like derivatives and integrals and stuff like that, so

whatever I think about this, I always think about you know an actual like plus 0.01 and divided by 0.01, because, like I just find that easier, just like I'd ever think about probability density functions, I always think about Actual probabilities of that toss, a coin something happens three times, so I always think like remember it's. It's totally fair to do this, because a computer is discrete. It's not continuous, like a computer, can't do anything infinitely small anyway right, so it's actually got to be calculating things at some level of precision right and our brains kind of need that as well. So this is like my version of Jeffery Clinton's like to visualize things in more than two dimensions. You just like, say: 12 dimensions really quickly well, visualizing in two dimensions. This is my equivalent. You know to to think about derivatives, just think about division and like, although all the mathematicians say no, you can't do that. You actually can like, if you think of DX dy is being literally.

You know change in X over changing Y, like the division actually like the calculations, do work like all the time. So, okay. So, let's do the same thing now with changing my slope by a little bit and so here's the same thing right and so you can see both of these are negative. Okay, so that's saying if I increase my intercept, my loss goes down.

16. 01:41:15

- **“You don’t need to learn how to calculate derivates & integrals, but you need to learn how to think about the spatially”, the ‘chain rule’, ‘jacobian’ & ‘hessian’**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

If I increase my slope, my loss goes down right, and so my derivative of my error, with respect to my slope, is, is actually pretty high and that's not surprising because it's actually you know the constant term is just being added. Where else as slope is being multiplied by 40, okay now find that differencing is all very well and good, but there's a big problem with finite difference, seeing in Hyden no spaces and the problem is this right, and this is like you - don't need to learn how To calculate derivatives or integrals, but you need to learn how to think about them spatially, right and so remember. We have some vector very high dimensional vector. It's got like a million items in it right and it's going through some weight, matrix right of size, like 1 million by size, a hundred thousand or whatever, and it's spitting out something of size. I hundred thousand, and so you need to realize, like there isn't like a gradient yeah, but it's like for every one of these things in this vector right, there's a gradient in every direction. You know in every part of the output right, so it actually has not a single gradient number, not even a gradient vector, but a gradient matrix right, and so this this is a lot to calculate right. I would literally have to like add a little bit to this and see what happens to all of these add a little bit to this see what happens to all of these right to fill in one column of this at a time.

So that's going to be horrendously slow, like that. So that's! Why, like, if you're ever thinking like how we can just do this with finite differencing, just remember like okay, we we're dealing in the with these very high dimensional vectors, where you know this. This kind of matrix, calculus, like all the concepts, are identical, but when you actually draw it out like this, you suddenly realize like okay for each number. I could change, there's a whole bunch of numbers that impacts - and I have this whole matrix of things to compute right, and so your gradient calculations can take up a lot of memory and they can take up a lot of time. So we want to find some way to do this more quickly, okay, and it's definitely well worth like

spending time kind of studying these ideas of, like you know, the idea of like the gradients like look up things like Jacobian and Hessian they're, the things that you Want to search for just that, unfortunately, people normally write about them with. You know: lots of great letters and bla bla bla right, but there are some. There are some nice. You know intuitive explanations out there and hopefully you can share them on the forum. If you find them, because this is stuff, you really need to really need to understand in here. You know, because you're trying to train something - and it's not working properly and like later on we'll learn how to like look inside pytorch to like actually get the values of the gradients, and you need to know like okay. Well, how would I like what the gradients you know? What would I consider unusual? Like you know, these are the things that turn you into a really awesome. Deep learning practitioner is when you can like debug your problems by like grabbing the gradients and doing histograms of them and like knowing you know that you could like plot that all each layer, my average gradients getting worse or you know, bigger, okay, so the trick to Doing this more quickly is to do it, analytically, rather than through finite differencing, and so analytically is.

Basically, there is a list. You probably all learned it at high school. There is a literally a list of rules that for every mathematical function, there's a like this is the derivative of that function. So you probably remember a few of them, for example x, squared it's alright, and so we actually have here an x squared. So here is our two x right now, the one that I actually want you to know is not any of the individual rules, but I want you to know the chain rule right, which you've got some function of some function of something. Why is this important? I don't know, that's a linear layer, that's a rally right and then we can kind of keep going backwards, map, etc. Right. A neural net is just a function of a function of a function of a function where the innermost is. You know it's basically linear rally. Your linear rally your dot, linear, sigmoid or soft mass all right, and so it's a function of a function of a function and so therefore to calculate the derivative of the weights in your model. The loss of your model with respect to the weights of your model, you're going to need to use the chain rule and specifically, whatever layer. It is that you're up to like. I want to calculate the derivative here and got a need to use. All of these. All of these ones, because that's all that's that's the function, that's being applied right and that's why they call this back propagation because the value of the derivative of that is equal to that derivative.

Now, basically, you can do it like this. You can say: let's call you is this right: let's call that you all right, then it's simply equal to the derivative of that times. Derivative of that right. You just multiply them together, and so that's what back propagation is like it's not that back propagation is a new thing for you to learn. It's not a new algorithm. It is literally take the derivative of every one of your layers and multiply them all together. So like it doesn't deserve a new name right apply. The chain rule to my layers does not deserve a new name, but it gets one because us neural networks folk really need to seem as clever as possible. It's really important that everybody else thinks we are way outside of their capabilities. So the fact that you're here means that we've failed, because you guys somehow think that you're capable right. So remember. It's really important when you talk to other people that you say backpropagation and rectified linear unit, rather than like multiply the layers, gradients or replace negatives with zeros. Okay. So so here we go so here is so I've just gone ahead and grabbed the derivative. Unfortunately, there is no automatic differentiation in Excel yet so I did the alternative, which is to paste the formula into Wolfram Alpha and got back the derivative, so there's the first derivative and there's the second derivative analytically. We only have one layer in this infinite.

Finally, small neural network, so we don't have to worry about the chain rule and we should see that this analytical derivative is pretty close to our estimated derivative from the find out differencing. And indeed it is right, and we should see that these ones are pretty similar. As well, and indeed they are right, and if you're you know back when I implemented my own neural Nets 20 years ago, I you know had to actually calculate the derivatives, and so I always would write like had something that would check the derivatives using finite difference. In and so for those poor people that they'd have to write these things by hand, you'll still see that they have like a finite differencing checkout. So if you ever do have to implement a derivative by hand, please make sure that you have a finite differencing checker. So that you can test it, alright, so there's no derivatives. So we know that if we increase B, then we're going to get a slightly better loss. So, let's increase B by a bit. How much should we increase it by well, we'll increase it by some more for this, so the motor-pod we're going to choose is called a learning rate and so here's our learning rate. So here's one enoch 4. Ok, so our new value is equal to whatever it was before our derivative times, our learning rate. Okay, so we've gone from one to one point or one, and then a we've done the same thing. So it's gone from one to one point, one two. So this is a special kind of mini batch.

It's a mini batch of size, one okay, so we call this online grading. Does it just means mini batch of size one? So then we can go into the next one. Next is 86. Why is 202 right? This is my intercept and slope copied across from the last row. Okay, so here's my new wire prediction: here's my new era here are my derivatives. Here are my new, a and B okay, so we keep doing that for every mini batch of one. Until eventually, we run out at the end of the new pocket, okay and so then, at the end of an epoch, we would grab our intercept and slope and paste them back over here as our new values there we are, and we can now continue again all Right so we're now, starting with pops today's either in the wrong spot. It should be pasted special, transpose values, all right. Okay, so there's a new intercept, there's any slope. Possibly I got that the wrong way around, but anyway you get the idea and then we continue. Okay, so I recorded the world's tiniest macro, which literally just copies the final slope and puts it into the new slope copies. The final intercept put the new intercept and does that five times and after each time it grabs the root, mean squared error and pastes it into the next spare area. And that is attached to this Run button. And so that's going to go ahead and do that. Five times, okay, so that's stochastic, gradient descent and, if so so, to turn this into a CNN all right, you would just replace this error function right and therefore this prediction with the output of that convolutional example spreadsheet, okay, and that then, would be in CNN being Trained with with SGD okay, now the problem is that you'll see when I run this, it's kind of going very slowly right.

We know that we need to get to a slope of two and an intercept of thirty, and you can kind of see it. This rate, it's going to take a very long time right and specifically, it's like it.

17. 01:53:45

- **Spreadsheet 'Momentum' tab**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Keeps going the same direction, so it's like come on! Take a hint, that's a good direction, so they come on. Take a hint, that's a good direction. Please keep doing that, but more is called momentum right so on our next spreadsheet, we're going to implement momentum. Okay, so

what momentum does is the same thing and what to simplify this spreadsheet? I've removed the finite difference cause okay other than that. This is just the same right. So it's true! What our X is our wise, A's and B's predictions. Our error is now over here: okay and here's, our derivatives, okay, our new calculation. For this particular row, our new calculation here for our new a term just like before is - is equal to whatever a was before. Okay. Now, this time I'm not taking the derivative, but I'm income other number times the loan rate. So what's this other number? Okay, so this other number is equal to the derivative times. What's this k, 1.02 plus 0.98 times the thing just above it? Okay, so this is a linear interpolation between this rows derivative for this mini-batches derivative and whatever direction we went last time right so, in other words, keep going the same direction as you were before. Right then update it, a little bit right and so in our rich. In our Python, just before we had a momentum of 0.9 okay, so you can see what tends to happen is that our negative kind of gets more and more negative right all the way up to like 2,000, where else, with our standard, SGD approach, a derivatives are Kind of all over the place right, sometimes there's 700 something negative, 7 positive. 100. You know so this is basically saying like yeah if you've been going down for quite a while, keep doing that until.

Finally, here it's like okay, that's that seems to be far enough. So that's being less and less and less negative, all right! Mister, we start being positive again, so you can kind of see why it's called momentum. It's like once. You start traveling in a particular direction for a particular weight. You're kind of the wheel start spinning and then once the gradient turns around the other way, it's like Oh slow down. We've got this kind of event, um and then finally turn back around right. So when we do it this way, all right, we can do exactly the same thing right and after five iterations we're at 89, where else before after five iterations we're at 104 right and after a few more, let's go, maybe 15, okay, so get this 102. For us here it's going right, so it's it's! It's a bit better! It's not hips better! You can still see like these numbers - they're, not zipping along right, but it's definitely an improvement and it also gives us something else to tune which is nice like. So if this is kind of a well-behaved error, surface right, in other words like, although it might be bumpy along the way, there's kind of some overall direction, like imagine, you're going down a hill right and there's like bumps - oh alright, so the mobile more momentum. You got going to skipping over the tops right, so we could say like okay, let's increase our beater up to 0.98 right and see if that like allows us to train a little faster and whoa. Look at that suddenly, what's going to okay, so one nice thing about things like momentum, is it's like another parameter that you can choose to try and make your model train better in practice? Basically, everybody does this every like you look at any like image: net winner or whatever they all use momentum, okay, and so back over here when we said here's SGD that basically means use the basic tab of our Excel spreadsheet.

But then momentum equals 0.9 means. Add in put a point nine over here, okay and so that that's kind of your like default starting point. So let's keep going and talk about Adam. So

18. 01:59:05

- **Spreasheet 'Adam' tab**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Adam is something which I actually was not right earlier on. In this course I said: we've been using Adam by default, we actually haven't, we've actually been. I noticed our we've actually

been using SGD with momentum by default, and the reason is that Adam has had much faster as you'll see it's much much faster to learn with, but there's been some problems, which is people who haven't been getting quite as good. Like final answers with Adam as they have with std with momentum, and that's why you'll see like all the you know, image net winning solutions and so forth, and all the academic papers always use SGD with momentum and I'll Adam seems to be a particular problem in Nlp people really haven't got Adam working at all. Well, the good news is this was I built. It looks like this was solved two weeks ago it. Basically it turned out that the way people were dealing with a combination of weight. Decay in Adam had a nasty kind of bargainer, basically, and that's that's kind of carried through to every single library and one of our students, and then Sahara has actually just completed a prototype of adding. Is this new version of Adam has called Adam W into fastai and he's confirmed that he's getting much faster, both the faster performance and also the better accuracy? So hopefully, we'll have this Adam W in faster, ideally before next week, we'll see how we go very very soon so so it is worth telling you about about Adam. So, let's talk about it.

It's actually incredibly simple, but again, you know make sure you make it sound, really complicated when you tell people so that you can so here's the same spreadsheet again right and here's our randomly selected a and B again somehow it's still one, here's a prediction: here's our Derivatives, okay, so now how we count letting on you hey, you could immediately see it's looking pretty hopeful because, even by like row, ten we're like we're seeing the numbers move a lot more right, so this is looking pretty encouraging. So how are we calculating this? It's equal to our previous value, with B minus j h, we're gon na have to find out what that is times our learning rate divided by the square root of LH okay, so I'm gon na have to dig it and see. What's going on, one thing to notice here is that my learning rate is way higher than it used to be, but then we're dividing it by this big number. Okay. So, let's start out by looking and seeing what this day-out thing is: okay, j8 is identical to what we had before. J8 is equal to the linear interpolation of the derivative and the previous direction. Okay, so that was easy, so one part of atom is to use momentum in the way we just defined it. Okay, the second piece was to divide by square root: L 8. What is that square root? L? 8. Okay is another linear interpolation of something and something else, and specifically it's a linear, interpolation of F, 8 squared okay, it's a linear interpolation of the derivative squared, along with the derivative squared last time; okay, so in other words, we've got two pieces of momentum going on Here one is calculating the momentum version of the gradient.

The other is calculating the momentum version of the gradient squared, and we often refer to this idea as a exponentially weighted moving average. In other words, it's basically equal to the average of this one and the last one in the last one in the last one that we're like multiplicatively decreasing the previous ones right because we're multiplying it by 0.9 times what 999. And so you actually see that, for instance, in the faster I code, if you look at fish, we don't just calculate the average loss right, because what I actually want. We certainly don't just report the loss for every mini match because that just bounces around so much so instead I say, average loss is equal to whatever the average loss was last time times: 0.98 plus the loss this time times 0.02 right. So, in other words, the faster you library, the thing that it's actually when you do, like the learning rate, finder or plot loss, it's actually showing you the exponentially weighted moving average of the loss. Okay. So it's like a really handy concept. It appears quite a lot right. The other in handy concept know about is this idea of like you've got two numbers: one of them is multiplied by some value. The other is multiplied by one minus that value. So this is a linear interpolation of two values. You'll see it all the time and for some reason, deep learning people nearly always use the value alpha when they do this.

So like keep an eye out if you're reading a paper or something - and you see like alpha times - bla bla, bla, bla, bla, plus one minus alpha times, some other bla bla bla bla right immediately like when people read papers. None of us like read every thing in the equation.

We look at it, we go. Oh linear, interpolation, right - and I said something I was just talking to Rachel about yesterday - is like whether we could start trying to find like a a new way of writing papers where we literally refactor them right, like it'd, be so much better to have written like Linear interpolate bla bla, bla bla bla right, because then you don't have to have that pattern. Recognition right, but until we convince the world to change how they write papers. This is what you have to do. Is you have to look? You know know what to look for right and once you do suddenly the huge page with formulas that at all, like you often notice like, for example, the two things in here like they might be totally identical. But this might be a time T, and this might be at like time, t minus y or something right like it's very often, these big ugly formulas turn out to be really really simple, if only they had ripped out them. Okay. So what are we doing with this gradient squared? So what we were doing with the gradient squared is, we were taking the square root and then we were adjusting the learning rate by dividing the learning rate by that okay, so gradient squared is always positive right and we're taking the exponentially waiting move moving average of A bunch of things that are always positive and then we're taking the square root of that right. So when is this number going to be high, it's going to be particularly high.

If there's like one big, you know if the gradients got a lot of variation. That's! Oh there's a high variance of gradient, then this G squared thing is going to be a really high number for us. If it's like a constant amount right, it's going to be smaller. That cuz, when you add things that are squared the squared slight jump out much bigger for us, if there wasn't, if there wasn't much change, it's not going to be as big. So basically, this number at the bottom here is going to be high. If our Brady -- nt is changing a lot now, what do you want to do if you've got something which is like first negative and then positive and then small and then high right? Well, you probably want to be more careful right. You probably don't want to take a big step, because you can't really trust it right. So when the when the variance of the gradient is high, we're going to divide our learning rate by a big number, we also found learning rate is very similar kind of size. All the time, then, we probably feel pretty good about the step, so we're dividing it by a small amount yeah, and so this is called an adaptive learning rate yeah and, like a lot of people, have this confusion about atom. I've seen it on the forum actually like there's some kind of adaptive learning rate where somehow you like, setting different learning rates for different layers or something it's like. No, not really right.

All we're doing is we're just saying, like this, keep track of the average of the squares of the gradients and use that to adjust the learning rate, so there's still one learning rate. Okay, in this case, it's one okay, but effectively every parameter at every epoch is being kind of like getting a bigger jump if the learning rate, if the gradients been pretty constant for that wait and a smaller jump. Otherwise, okay and that's Adam - that's the entirety of Adam in in Excel right. So there's now no reason at all why you can't train imagenet in Excel, because you've got you've got access to all of the pieces you need, and so let's try this out run. Okay, that's not bad right five and we straight up to twenty nine and two right. So the difference between, like you know, standard SGD and this is is huge and basically that you know the key difference was that it figured out that we need to be. You know moving this number much faster, okay and so, and so it do, and so you can see, we've now got like two different parameters. One is kind of momentum for the gradient piece. The other is the momentum for the gradient squared piece and there I think they're called like, I think, there's just a couple of the beta. I

think when you, when you want to change it in PI tortes, is, I think, what beta, which is just a couple of two numbers. You can change Jeremy, so so you set the yeah.

I think I understand this concept of you know one day when a gradient is, it goes up and down then you're not really sure which direction should should go, so you should kind of slow things down. Therefore, you subtract that gradient from the learning rate, so, but how do you implement? How far do you go? I guess maybe I miss something early on you. Do you set a number somewhere, we divide yeah, we divide the learning rate divided by the square root of the moving average gradient squared. So that's where we use it. Oh I'm sorry, can you be a little more sure? So d2 is the learning rate, which is one yeah m27, is our moving average of the squared gradients. So we just go D 2, divided by square root and preserve. That's it. Okay thanks. I have one question yeah, so the new method that you just mentioned, which is in the process of getting implemented in yes, how different is it from here? Okay, let's do that, so to understand Adam W. We have to understand, wait, okay and maybe we'll learn more about that later. Let's see how we go now with great okay, so the idea is that when you have lots and lots of parameters like we do with you know most of the neural Nets. We train. You very often have like more parameters and data points, or you know, like regularization, becomes important and we've learnt how to avoid overfitting by using dropout right, which

19. 02:12:01

- **Beyond Dropout: 'Weight-decay' or L2 regularization**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Randomly deletes some activations in the hope, that's going to learn some kind of more resilient set of weights. There's another kind of Ritter ization. We can use called weight, decay or l2 regularization and it's actually comes kind of as a kind of classic statistical technique, and the idea is that we take our loss function right. So we take out like arrow, squared loss function and we add an additional piece to it. Let's add weight decay right now. The additional piece we add is to basically add the square of the weights, so we'd say, plus, B, squared plus a squared okay. That is now wait, 2 K or L tree regularization, and so the idea is that now the the loss function wants to keep the weight small, because increasing the weights makes the loss worse, and so it's only going to increase the weights if the loss improves by More than the amount of that penalty and, in fact to make this weight to get to proper weight decay, we then need some multiplier yeah right. So, if you remember back in our here, we said weight. Decay equals W d5e, neg, 4, okay, so to actually use the same way to K, I would have to multiply by 0.005 all right, so that's actually now the same weight. Okay. So if you have a really high weight decay that it's going to set all the parameters to zero, so it'll never over fit right because it can't set any parameter to anything.

And so, as you gradually decrease the weight decay, a few more weights can actually be used right, but the ones that don't help much it's still going to leave at zero or close to zero right. So that's what that's, what weight decay is is is literally to change the loss function to a D in this sum of squares of weights times some parameter, some hyper parameter. I should say the problem is that if you put that into the loss function as I have here, then it ends up in the moving average of gradients and the moving average of Squared's of gradients for atom right, and so basically we end up when there's a Lot of variation, we end up decreasing the amount

of weight decay and, if there's very little variation, we end up increasing the amount of weight decay. So we end up basically saying penalize parameters. You know weights that are really hi unless their gradient varies a lot, which is never what we intended right. That's just not not the plan at all, so the trick with Adam W is we basically remove weight decay from here? So it's not in the last function. It's not in the G, not in the G squared, and we move it so that instead it's it's. It's added directly to the when we update with the learning rate it's out of there instead, so in other words it would be, we would put the weight decay or I should a gradient of the weight decay in here when we calculate the new, a mu V. So it never ends up in our G M G squared.

So that was like a super fast description, which will probably only make sense if you listen to a three or four times on the video and then talk about it on the forum yeah. But if you're interested, let me know - and we can also look at Ann Ann's code - that's implemented. Yes and you know the the idea of using weight decay. Is it's a really helpful regularizer, because it's basically this way that we can kind of stay like you know? Please don't increase any of the weight values unless the you know, improvement in the loss is worth it, and so, generally speaking, pretty much all state of the art models have both dropout and weight decay, and I don't claim to know like how to set each one And how much of H to use to say, like you, it's worth trying both to go back to the idea of embeddings? Is there any way to interpret the final to reduce it? Embeddings, like absolutely we're gon na look at that next week, I've it's super fun! It turns out that you know we'll learn what some of the worst movies of all time. It's Letham, it's that John Travolta Scientology once my battleship earth or something I think that was like the worst movie of all time. According to our beds, to many recommendations for scaling the l2 penalty, or is that kind of based on how how wide the notes are? How many notes about III have no suggestion at all, like I, I kind of look for like papers or cackle competitions or whatever similar and try to set up. Frankly, the same, it seems like in a particular area like computer vision, object, recognition, it's like somewhere between one in neck, four or one in egg five seems to work.

You know actually, in the Adam W paper, the authors point out that, with this new approach it actually becomes like it seems to be much more stable as to what the right way to K amounts are so hopefully now, when we start playing with it, we'll be Able to have some definitive recommendations by the time we get to part two all right. Well, that's nine o'clock! So this week you know practice the thing that you're least familiar with. So if it's like jacobians and Hessians read about those. If it's broadcasting read about those, if it's understanding python ooo read about that, you know try to implement your own custom layers, read the faster higher layers you know and and talk on the forum about anything that you find weird or confusing. Alright see you next week,

Lesson 6: Interpreting embeddings; RNNs from scratch

Outline

Today is a very busy lesson! We first learn how to interpret the collaborative filtering embeddings we created last week, and use that knowledge to answer the question: “what is the worst movie of all time?”

Then we cover what is perhaps the most practically important topic in the whole course: how to use deep learning to model “structured data” such as database tables and spreadsheets, as well as time series. It turns out that not only is deep learning often the most accurate modeling approach for this tasks, it can be the easiest approach to develop too.

We close out the lesson with an introduction to recurrent neural networks (RNNs), and use an RNN to write a new philosophical treatise…

Video Timelines and Transcript

1. 00:00:10

- **Review of articles and works**
- **"Optimization for Deep Learning Highlights in 2017" by Sebastian Ruder,**
- **“Implementation of AdamW/SGDW paper in Fastai”,**
- **“Improving the way we work with learning rate”,**
- **“The Cyclical Learning Rate technique”**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Welcome back less than six, so this is our penultimate lesson and believe it or not a couple of weeks ago in Lesson four I mentioned, I was going to share that lesson with this terrific. You know P researcher Sebastian Reuter, which I did and he he said he loved it and he's gone on to yesterday released this new post. He called optimization for deep learning highlights in 2017, in which he covered, basically everything that we talked about in that lesson and with some very nice shout outs to some of the work that some of the students here have done, including when he talked about this separation Of the separation of weight decay from the momentum term, and so he actually mentions here the opportunities in terms of improved kind of software decoupling. This allows and actually links to the commits from an answer hah actually showing how to implement this in fastai. So, first a eyes code is actually being used as a bit of a role model. Now he then covers some of these learning rate tuning techniques that we've talked about, and this is the SGD our schedule. It looks a bit different to what you're used to seeing this is on a log curve. This is the way that they show it on the paper and for more information again links to two blog posts, one from vitaly about this topic

and and again, ananza ha is blog post on this topic.

So it's great to see that some of the work from faster, our students is already getting noticed and picked up and shared, and this blog post went on to get on the front page of hacker news. So that's pretty cool and hopefully more and more of this work or be picked up on sisters.

2. 00:02:10

- **Review of last week "Deep Dive into Collaborative Filtering" with MovieLens, analyzing our model, 'movie bias', '@property', 'self.models.model', 'learn.models', 'CollabFilterModel', 'get_layer_groups(self)', 'lesson5-movielens.ipynb'**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Released publicly so last week we were kind of doing a deep dive into collaborative filtering and, let's remind ourselves of kind of what our final model looked like. So, in the end, we kind of ended up rebuilding the model, that's actually in the first, a a library where we had an embedding. So we had this little get embedding function that grabbed an embedding and randomly initialize the weights for the users and for the items. That's the kind of generic term. In our case the items are movies and the bias for the users, the bias for the items, and we had n factors embedding size for each for each one. Of course, the biases just had a single one, and then we grabbed the users and item in weddings. Multiply them together. Summed it up each row and add it on the bias terms pop that through a sigmoid, to put it into the range that we wanted. So that was our model and one of you asked if we can kind of interpret this information in some way, and I promised this week we would see how to do that. So, let's take a look, so we're going to start with the model we built here, where we just used that fastai library, collaborative data set from CSP and then that get learner, and then we fitted it in three epochs 19 seconds. We've got a pretty good result, so what we can now do is to analyze that model.

So you may remember right back when we started, we read in the movies CSV file, but that's just a mapping from the ID of the movie to the name of the movie and so we're just going to use that for display purposes. So we can see what we're doing, because not all of us have watched every movie, I'm just going to limit this to the top 500, most populous or 3,000 most popular movies. So we might have more chance of recognizing the movies we're looking at and then I'll go ahead and change it from the movie IDs from movie lens to those unique IDs that we're using the contiguous IDs, because that's what a model has alright so inside the learn. Object that we create inside alona, we can always grab the pytorch model itself, just by saying, learn: model okay and like I'm, going to kind of show you more and more of the code at the moment. So, let's take a look at the definition of model, and so a model is a property. So if you haven't seen a property before a property is just something in Python which looks like a method when you define it that you can call it without parentheses, as we do here, alright, and so it kind of looks when you call it like it's a Regular attribute, but it looks like when you define it like it's a method, so every time you call it, it actually runs this code, okay, and so in this case it's just a shortcut to grab something called dot models model.

So you may be interested to know what that looks like learn about models, and so this is there's a fastai model type is a very thin wrapper for pite watch models. So we could take a

look at this code filter model and see what that is. It's only one line of code: okay and yeah - we'll talk more about these in part two right, but basically that there's this very thin wrapper and the main thing one of the main things that fast i out does is we have this concept of layer groups where Basically, when you say here, though, different learning rates and they're going to apply two different sets of layers and that's something that's not in paid watch. So when you say I want to use this pytorch model, all this with one thing we have to do, which is to say like okay, one hour later, groups yeah, so the details aren't terribly important, but in general, if you want to create a little wrapper For some other pipe watch model, you could just write something like this, so to get to get inside that to grab the actual pytorch model itself, its models, dot model, that's the pytorch model and then the learn object has a shortcut to that. Okay. So we're going to set m to be the pytorch model, and so, when you print out a pipe watch model, it prints it out. Basically, by listing out all of the layers that you created in the constructor, it's quite it's quite nifty. Actually, when you kind of think about the way, this works thanks to kind of some very handy stuff in Python, we're actually able to use standard, oh wow, to kind of define these modules in these layers and they basically automatically kind of register themselves with pipe which So back in our embedding bias, we just had a bunch of things where we said: okay, each of these things are equal to these things, and then it automatically knows how to represent that.

So you can see. There's the name. Is you, and so the name is just literally whatever we called it, yeah you and then the definition is it's this kind of layer? Okay, so that's our height watch model, so we can look inside that basically use that. So if we say m dot I be then that's referring to the embedding layer for an item which is the bias layer. So an item bias in this case is the movie bias, so each move either a 9000 of them has a single bias element. Okay, now the really nice thing about high torch layers and models is that they all look the same. They basically got to use them, you call them as if they were action, so we can go m.i.b, parenthesis right and that basically says I want you to return. The value of that layer and that layer could be a full-on model right so to actually get a prediction from a play: torch model you just, I would go m and pass in my variable, okay, and so in this case my B and pass in my top Movie indexes now models remember layers. They require variables, not tensors, because it needs to keep track of the derivatives. Okay, and so we use this capital V to turn the tensor into a variable and was just announced this week that pytorch 0.4, which is the version after the one. That's just about to be released is going to get rid of variables and will actually be able to use tensors directly to keep track of derivatives.

So, if you're watching this on the MOOC and you're looking at point four, then you'll probably notice that the code doesn't have this V unit anymore, and so that would be pretty exciting when that happens. But for now we have to remember if we're going to pass something into a model to turn it into a variable. First and remember: a variable has a strict superset of the API of a tensor, so anything you can do to a tensor. You can do to a variable and it up will take its log or whatever okay, so that's going to return a variable which consists of going through each of these movie IDs, putting it through this embedding layer to get its bias. Okay and that's going to return a variable. Let's take a look so before I press shift down to here. You can have a think about what I'm going to have. I've got a list of 3,000 movies going in turning into variable, putting it through this embedding layer. So just have a think about what we expect to come out: okay and we have a variable of size 3,000 by one. Hopefully that doesn't surprise you. We had 3000 movies that we are looking up each one hadn't had a one, long, embedding, okay, so there's our three thousand one you'll notice, it's a variable, just not surprising, because we fed it a variable, so we've got a variable back and it's a variable.

That's on the GPU right doc, CUDA, okay, so we have a little shortcut in fastai, because we we very often when I take variables, turn them into tensors and move them back to the CPU. So we can play with them more easily. So two NP is is two numpy okay, and that does all of those things and it works regardless of whether it's a tensor or a variable. It works, regardless of whether it's on the CPU or GPU it'll end up giving you a a numpy array from that. Okay, so if we do that, that gives us exactly the same thing as we just looked at, but now in numpy form. Okay, so that's a super handy thing to use when you're playing around with pytorch. My approach to things is: I try to use numpy for everything, except when I explicit and

3. 00:12:10

- **Jeremy: "I try to use Numpy for everything, except when I need to run it on GPU, or derivatives",**
- **Question: "Bring the model from GPU to CPU into production ?", move the model to CPU with 'm.cpu()', 'load_model(m, p)', back to GPU with 'm.cuda()', 'zip()' function in Python**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

You need something to run on the GPU or I need its derivatives right, in which case I use pytorch because, like none part like I kind of find none PI's, often easier to work with it's been around many years longer than pytorch. So you know and lots of things like the Python, imaging library, OpenCV and lots and lots of stuff like pandas. It works with numpy. So my approach is kind of like do as much as I can in num pile and finally, when I'm ready to do something on the GPU or take its derivative to pytorch and then as soon as I can. I put it back in vampire and you'll see that the fastai library really works. This way, like all the transformations and stuff happen in lamb pie, which is different to most high torch computer vision, libraries, which tend to do it all as much as possible in pytorch. I try to do as much as possible in non pipe. So let's say we wanted to transfer build a model in the GPU with the GPU and train it. Then we want to bring this to production. So would we call to numpy on the model itself or would we have to iterate through all the different layers and then call to NP yeah good question, so it's very likely that you want to do inference on a cpu rather than a GPU? It's it's more scalable, you don't have to worry about putting things in batches, you know and so forth. So you can move a model onto the cpu just by typing m dot, CPU, and that model is now on the cpu.

And so, therefore, you can also then put your variable on the CPU by doing exactly the same thing, so you can say like so now. Having said that, if you're, if you'll serve, it doesn't have a GPU or CUDA GPU, you don't have to do this because it won't put it on the GPU at all. So if for inferencing on the server, if you're running it on, you know some t2 instance or something it'll, work fine and will run on the on the cpu, automatically quick follow-up. And if we train the model on the GPU, and then we save those embeddings and the weights, would we have to do anything special to load? You know you won't. We have something. Well, it kind of depends how much of faster I you're using so I'll. Show you how you can do that in case. You have to do it manually, one of the students figure this out, which is really handy when we there's a load model function and you'll see what it does, but it does torch dot load. Is it basically? This is like some magic incantation that, like normally, it has to load it onto the same GPU or saved on, but this will like load it into what it was, what it is available. So there's a Andy discovery thanks for the great questions and to put that back on the GPU I'll

need to say doc, CUDA and now there we go. I can run it again, okay, so it's really important to know about the zip function in Python, which iterates through a number of lists at the same time. So in this case, I want to grab each movie along with its bias term, so that I can just pop it into our list of tuples.

So if I just go zip like that, that's going to iterate through each movie ID and each bias term, and so then I can use that in a list comprehension to grab the name of each movie along with its place.

4. 00:16:10

- **Sort the movies, John Travolta Scientology worst movie of all time “Battlefield Earth”, ‘key=itemgetter()jj’, ‘key=lambda’**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Okay, so having done that, I can then sort - and so here are - I told you that John John Travolta Scientology movie at the most negative of the quiet by a lot. If this was a cable competition, Battlefield Earth would have like won by miles or this seven seven, seven, ninety six, so here's the worst movie of all time according to IMDB and like it's interesting when you think about what this means right, because this is like a Much more authentic way to find out how bad this movie is because, like some people are just more negative about movies right and like it more of them watch your movie, like you, know, highly critical audience. They're gon na read it badly. So if you take an average, it's not quite fair right, and so what this is? You know what this is doing is saying once we, you know, remove the fact that different people have different overall, positive or negative experiences and different people watch different kinds of movies, and we correct for all that this is the worst movie of all time. So that's a good thing to know, so this is how we can yeah look inside our our model and and interpret the bias vectors you'll see here, I've sorted by the zeroth element of each tuple by using a lambda. Originally, I used this special item ghetto. This is part of pythons operator library, and this creates a function that returns the zeroth element of something in order to save time, and then I actually realize that the lambda is only one more character to write.

Then the item get us, so maybe we don't need to know this after all, so yeah really useful to make sure you know how to write lambdas in Python. So this is this is a function, okay and so sort. The sort is going to call this function. Every time it decides like is this thing higher or lower than that other thing, and this fact this is going to return the zeroth element. Okay, so here's the same thing and item get a format, and here is the reverse and Shawshank Redemption right at the top. I'll definitely agree with that: Godfather usual suspects yeah. These are

5. 00:18:30

- **Embedding interpration, using ‘PCA’ from ‘sklearn.decomposition’ for Linear Algebra**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

All pretty great movies, twelve Angry Men. Absolutely so there you go, there's how we can look at the base. So then the second piece to look at would be the the embeddings. How can we look at the embeddings, so we can do the same thing so remember. I was the item embeddings rather than IV. With the item bias, we can pass in our list of movies as a variable turn it into numpy and here's our movie embedding so for each of the 3,000. Most popular movies here are its 50 embeddings. So it's very hard unless you're Geoffrey Hinton, to visualize a 50 dimensional space. So what we'll do is we'll turn it into a three dimensional space. So we can compress high dimensional spaces down into lower dimensional spaces, using lots of different techniques. Perhaps one of the most common and popular is called PCA. Pca stands for principle, components, analysis, it's a linear technique, but when your techniques generally work fine for this kind of embedding I'm not going to teach you about PCA now, but I will say in Rachel's, computation or linear algebra class, which you can get to you from. First, at AI, we cover PCA in detail and it's a really important technique. It. Actually, it turns out to be almost identical to something called singular value, decomposition which is a type of matrix decomposition which actually does turn up in deep learning a little bit from time to time, it's kind of somewhat worth knowing if you were going to dig more Into linear algebra, you know SPD and PCA, along with eigenvalues and eigenvectors, which are all slightly different versions.

Is this kind of the same thing or all worth knowing, but for now just know that you can grab PCA from SK, learn to calm position, say how much you want to reduce the dimensionality too. So I want to find three components, and what this is going to do is it's going to find three linear combinations of the 50 dimensions which capture as much as the variation as possible? Badar is different to each other as possible. Okay, so we would call this a lower rank approximation of our matrix all right. So then we can grab the components, so that's going to be their three dimensions, and so once we've done that, we've now got three by three thousand, and so we can now take a look at the first of them and we'll do the same thing of using Zip to look at each one along with its movie, and so here's the thing right. We we don't know ahead of time. What this PCA thing is. It's just, it's just a bunch of latent factors. You know it's. It's kind of the the main axis in this space of latent factors, and so what we can do is we can look at it and see if we can figure out what it's about right. So, given that police academy for is high up here, along with water world, where else Fargo Pulp Fiction and God further a high up here - I'm gon na guess that a high value is not going to represent like critically acclaimed movies or serious watching. So I kind of like all this yeah okay. I call this easy.

What she is serious all right, but like this is kind of how you have to interpret your embeddings is like take a look at what they seem to be showing and decide what you think it means. So this is the kind of the the principal axis in this set of embedding, so we can look at the next one. So do the same thing and look at the the first index one embedding this one's a little bit harder to kind of figure out. What's going on, but with things like Mulholland Drive and Purple Rose of Cairo, these look more kind of dialog II, kind of ones, or else things like Lord of the Rings in the Latin and Star Wars. These book more like kind of modern, CGI II kind of ones, so you could kind of imagine that on that pair of dimensions, it probably represents a lot of you know differences between how people read movies. You know some people, like you, know: purple rise of Cairo type movies. You know Woody Allen, kind of classic and some people like these. You know big Hollywood spectacles, some people, presumably like police academy, for more than they like Fargo, so yeah. So I'm like you, can kind of get the idea of what's happened. It's it's done a you know through a model which was you know, for a model which was literally multiply, two things together and Adam hop. It's

learnt quite a lot. You know which is kind of cool, so that's what we can do with with that and then we could. We could plot them if we wanted to.

I just grabbed a small subset to plot on those first, two asses all right. So that's that so I wanted to next kind of dig in a layer deeper into what actually happens when we say fit alright. So when we said

6. 00:24:15

- **Looking at the "Rossmann Retail / Store" Kaggle competition with the 'Entity Embeddings of Categorical Variables' paper.**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Learn fit: what's it doing, for something like the store model? Is it a way to interpret the embeddings for something like this? The rustman one? Yes, yeah, we'll see that in a moment. Well, let's jump straight there, what the hell, okay, so so for the rustman. How much are we going to sell at each store on each date model we? This is from the paper gore and burke on it. So it's a great paper by the way well worth you know like pretty accessible. I think any of you would, at this point, be able to at least get the gist of it. If you know, and much of the detail as well, particularly as you've also done the machine learning course, and they actually make this point in the paper. This is in the paper that the equivalent of what they call entity embedding layers, so an embedding of a categorical variable is identical to a one hot encoding, followed by a matrix multiply. That's why they're basically saying if you've got three embeddings, that's the same as doing three one hot encodings putting each through one through a matrix, multiply and then put that through a a dense layer. Well, what pytorch would call a linear, oh yeah, right one of the nice things here is because this is kind of like well. They thought it was. The first paper is actually the second, I think paper to show the idea of using categorical embeddings for this kind of data set. They really go to clean too. Quite a lot of detail to you know right back to the the detailed stuff that we learnt about.

So it's kind of a second, you know a second cat of thinking about what embeddings are doing so one of the interesting things that they did was they said: okay, after we've trained a neural net with these embeddings. What else could we do with it? So they got a winning result with a neural network where the entity meetings, but then they said hey, you know what we could take those empty embeddings and replace each categorical variable with the learnt entity embeddings and then feed that into a GBM right. So, in other words like, rather than passing into the GBM, a one modern coded version or an ordinal version, let's actually replace the categorical variable with its embedding for the appropriate level for that row right. So it's actually a way of create. You know feature engineering and so the main average percent error, without that for gbms, I'm using just 100 codings, was 0.15, but with that it was 0.11 that random forests, without that was point one six with that 0.108, nearly as good as the neural net right. So this is kind of an interesting technique, because what it means is in your organization, you can train a neural net that has an embedding of stores and an embedding of product types and an embedding of I don't know whatever kind of high cardinality or even medium Cardinality categorical variables you have and then everybody else in the organization can now like chuck those into their.

You know JVM or random forest or whatever, and I'm use them, and what this is saying is

they won't get. In fact, you can even use K nearest neighbors with this technique and get nearly as good a result right. So this is a good way of kind of giving the power of neural nets to everybody in your organization without having them do the faster idea of learning course. First, you know they can just use whatever SK, learn or R or whatever that they're used to and like those those embeddings could literally be in a database table. Because if you think about an embedding is just an index lookup right, which is the same as an inner join in SQL right. So if you've got a table on each product along with its embedding vector, then you can literally do in a joint. And now you have every row in your table along with its product, embedding vector. So that's a really. This is. This is a really useful idea and gbm's and random forests learn a lot quicker than neural nets. Do all right! So that's like, even if you do know how to train your on its. This is still potentially quite handy. So here's what happened when they took the various different states of Germany and plotted the first two principal components of their embedding vectors, and they basically here is where they were in that 2d space and wacken lee enough.

I've circled in red three cities and i've circled here: the three cities in Germany and here I've circled in purple, so blue here at the blue: here's the green here's, the green. So it's actually drawn a map of Germany, even though it never was told anything about how far these states are away from each other or the very concept of geography didn't exist, so that's pretty crazy, so that was from there paper. So I went ahead and looked well. Here's another thing. I think this is also from their paper. They took every pair of places and they looked at how far away they are on a map versus how far away are they in embedding space and they've got this beautiful correlation. Alright, so again it kind of. Apparently, you know it's doors that are near by each other, physically, have similar characteristics in terms of when people buy more or less stuff from them. So I looked at the same thing four days of the week right. So here's an embedding of the days of the week from our model and I just kind of joined up Monday, Tuesday, Wednesday, Tuesday, Thursday, Friday Saturday Sunday. I did the same thing for the months of the year. All right again, you can see you know: here's! Here's winter here's summer so yeah, I think, like visualize embeddings, can be interesting like it's good to like. First of all check you can see things you would expect to see you know and then you could like try and see.

Like maybe things you didn't expect to see, so you could try all kinds of clusterings or or whatever - and this is not something which has been widely studied at all right. So I'm not going to tell you what the limitations are of this technique or whatever. Oh yes, so I've heard of other ways to generate embeddings like skip grams uh-huh wondering if you could say is there one better than the other using your own Network, sir skip grams, so screwed grams is quite specific to NLP right so, like I'm, not sure. If we'll cover it in this course, but basically the the approach to original kind of word to vac approach to generating embeddings was to say you know what we actually don't have. We don't actually have our labelled data set. You know they said all we have is like google books, and so they have an unsupervised, learning problem, unlabeled problem, and so the best way, in my opinion, to turn an unlabeled problem into a labelled problem is to kind of invent some labels and so what they Did in the word to vet case was they said: okay, here's a sentence with 11 words in it right and then they said: okay, let's delete the middle word and replace it for the random word, and so you know originally it said cat and they say no. Let's replace that with justice all right so before it said the cute little cat sat on the fuzzy mat and now it says the cute little justice sat on the fuzzy man right and what they do.

Is they do that, so they have one sentence where they keep exactly as is, and then they make a

copy of it and they do the replacement. And so then they have a label where they say it's a one. If it was unchanged, it was the original and zero, otherwise, okay, and so basically, then you now have something you can build a machine learning model on, and so they went and build a machine learning model on this. So the model was like try and find the effect sentences, not because they were interested in a fake sentence binder, but because, as a result, they now have embeddings that, just like we discussed you can now use for other purposes, and that became word to vet. Now it turns out that if you do this as just a kind of a effectively like a single matrix multiply, rather than make it a deep neural net, you can train this super quickly, and so that's basically what they did with they'd met there, though they kind Of decided we're going to make a pretty crappy model like a shallow learning model rather than a deep model. You know with the downside, it's a less powerful model, but a number of upsides. The first thing we can train it on a really large data set and then also really importantly, we're going to end up with embeddings, which have really very linear characteristics. So we can like add them together and subtract them and stuff like that. Okay, so that so there's a lot of stuff, we can learn about there from like for other types of embedding like categorical embeddings and specifically, if we want categorical embeddings, which we can kind of draw nicely and expect them to us to be able to add and Subtract them and behave linearly. You know, probably if we want to use them in k-nearest, neighbors and stuff, we should probably use shallow learning.

If we want something, that's going to be more predictive, we probably want to use a neural net and so actually an NLP. I'm really pushing the idea that we need to move past word to backhand glove, these linear based methods, because it turns out that those embeddings are way less predictive than embeddings learnt from models, and so the language model that we learned about, which ended up getting a State-Of-The-Art on sentiment, analysis didn't used a lot more work to vet that instead we pre trained a deep, recurrent neural network and we ended up with not just a pre trained word vectors, but a for pre-trained model. So it looks like Duke, creates embeddings for entities. We need like a dummy task, not necessarily a dummy task like in this case. We had a real task right, so we created the embeddings for Rossmann by trying to predict store sales. You only need this isn't just in this, isn't just for learning embeddings for learning. Any kind of feature space you either need label data or you need to invent some kind of fake task. So does that task matter like if I choose a task and train and lettings? If I choose another task and train and lettings like which one is it's a great question, and it's not something - that's been studied nearly enough right, I'm not sure that many people even quite understand that when they say unsupervised learning now about nowadays, they almost nearly always Mean fake tasks, labeled learning and so the idea of like what makes a good fake task. I don't know that I've seen a paper on that right that intuitively you know, we need something where the kinds of relationships it's going to learn likely to be the kinds of relationships that you probably care about right. So, for example, in in computer vision, one kind of fake task people use is to say, like let's take some images and use some kind of like unreal and unreasonable data.

Augmentation like like recolor them too much or whatever, and then we'll ask the neural net to like predict, which one was the Augmented, which one was not. You admitted yeah. So it's I think, it's a fascinating area, one which you know would be really interesting for people to you know maybe some of the students here they're looking to further. It's like take some interesting semi-supervised tour, unsupervised, datasets and try and come up with some like more clever fake tasks and see like does it matter. You know how much does it matter in general like if you can't come up with a fake task that you think seems great, I would say, use it use the best. You can it's an often surprising how how little you need like the ultimately

crappy fake task. Is called the auto encoder and the auto encoder is the thing which which one the claims prediction - competition that just finished on cattle. They had lots of examples of insurance policies where we knew this was how much was claimed and then lots of examples of insurance policies where I guess there must have been still still open. We didn't yet know how much they claimed right and so what they did was they said, okay, so for all of the ones. So, let's basically start off by grabbing every policy right and we'll take a single policy and we'll put it through a neural net right and we'll try and have it reconstruct itself.

But in these intermediate layers, and at least one of those intermediate layers will make sure there's less activations and there were inputs. So let's say if there was a hundred variables on the insurance policy. You know we'll have something in the middle that only has like twenty activations all right, and so, when you basically are saying hey reconstruct your own input like it's, not a different kind of model doesn't require any special code. It's literally just passing. You can use any standard, pipe torch or fastai learner. You just say my output equals my input right and that's that's like the the most uncreated. You know invented task you can create and that's called an autoencoder and it works surprisingly. Well, in fact, to the point that it literally just won a cackle competition, they took the features that it learnt and chucked it into another neural net and yeah and one you know. Maybe if we have enough students taking an interest in this, then you know we'll be able to cover covered unsupervised learning in more detail in in part two specially, given this cattle have a win. I think this may be related to the previous question when training language models is the language model example trained on the archive data. Is that useful at all in the movie great question you know I was just talking to Sebastian about this question read about this this week and we thought would try and do some research on this in January. It's it's again. It's not well done.

We know that in computer vision, it's shockingly effective to train on cats and dogs and use that fruit train network to do lung cancer diagnosis and CT scans in the NLP world. Nobody much seems to have tried this, the NLP research as I've spoken to other than Sebastian about this assume that it wouldn't work and they generally haven't bother trying. I think it would work great so so, since we're talking about

7. 00:41:02

- **“Rossmann” Data Cleaning / Feature Engineering, using a Test set properly, Create Features (check the Machine Learning “ML1” course for details), ‘apply_cats’ instead of ‘train_cats’, ‘pred_test = m.predict(True)’, result on Kaggle Public Leaderboard vs Private Leaderboard with a poor Validation Set. Example: Statoil/Iceberg challenge/competition.**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Ruspin I just mentioned during the week. I was interested to see like how good this solution actually actually was, because I noticed that on the public leader board, it didn't look like it was going to be that great, and I also thought it'd be good to see. Like what does it actually take to use a test set properly with this kind of structured data? So if you have a look at ruspin now, I've pushed some changes that actually run the test set through as well, and so you can get a sense of how to do this so you'll see. Basically, every line appears twice one for tests and one-foot one for train when we get there yeah test train test trains, history. Obviously, you could do this on a lot fewer lines of code by putting all of the steps into a method and then

pass either the train data set. Well, the test data set up dataframe to it. In this case, i wanted to kind of put for teaching purposes, you'd, be able to see step and to experiment to see what each step looks like, but you could certainly simplify this code so yeah. So we do this for every data frame and then some of these you can see I kind of lived through the data frame in joined and the joint test right training. Just this whole thing about the durations. I basically put two lines here: one that said data frame equals train columns, one that says data frame equals test columns, and so my you know basically ideas you'd run this line first and then you would skip the next one and you'd run everything beneath it and Then you'd go back and run this line and then run everything believe it. So some people on the forum were asking how come this code wasn't working this week, which is a good reminder that the code is not designed to be code that you always run top to bottom without thinking right, you're meant to like think like what is this Code here should I be running it right now: okay, and so like the early lessons, I tried to make it so you can run it top to bottom, but increasingly as we go along, I kind of make it more and more that, like you, actually have to Think about what's going on so Jimmy you're, talking about shadow learning and deep learning, could you define that a bit better by sure? I'm learning, I think I just mean anything that doesn't have a hidden layer, so something that's like a dot product matrix multiplier.

Basically, okay, so so we end up with a training and a test version, and then everything else is basically the same. One thing to note on a lot of these details of this: we cover in the machine learning course by the way, because it's not really deep, learning specific so check that out. If you're, just in the details, I should mention you know, we use apply cats rather than train cats to make sure the test set and the training set have the same categorical codes and that they join too. We also need to make sure that we keep track of the mapper. This is the thing which basically says: what's the mean and standard deviation of each continuous column and then apply that same method test set, and so when we do all that, that's basically it then the rest is easy. We just have to pass you in the test. Data frame in the usual way when we create our model data object and there's no changes through all here. We trained it in the same way and then once we finish training it, we can then call predict, as per usual, passing in true to say this is the test set rather than the validation set and pass that off to cattle, and so it was really interesting Because this was my submission, it got a public score of 103, which would put us in about 300, and some things place which looks awful right and our private score of 107 need a board private.

Here's about fifth right so like if you're competing in a cable competition - and you don't haven't thoughtfully, created a validation set of your own and you're, relying on publicly the board feedback. This could totally happen to you, but the other way around you'll be like. Oh I'm. In the top ten I'm doing great and then oh, for example, at the moment, the ice Berg's competition recognizing icebergs, a very large percentage of the public leaderboard set, is synthetically generated data augmentation data like totally meaningless, and so your validation set is going to be much More helpful and the public leaderboard feedback right so yeah be very careful. So our final score here is kind of within statistical noise of the actual third-place getters. So I'm pretty confident that we've we've captured their approach, and so that's that's pretty interesting. Something to mention. There's a nice kernel about the rustman, I quite a few nice kernels actually, but you can go back and see like, particularly if you're doing the groceries competition go and have a look at the Rossmann kernels, because actually quite a few of them, a higher quality than The ones for the Ecuadorian groceries competition, one of them, for example, showed how, on four particular stores like straw, eighty five, the sales for non Sundays and the sale for Sunday's looked very different.

Where else there are some other stores where the sales on Sunday don't look any

8. 00:47:10

- **A mistake made by Rossmann 3rd winner, more on the Rossmann model.**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Different and it can kind of like get a sense of why you need these kind of interactions. The one I particularly wanted to point out is the one. I think I briefly mentioned that the third-place winners, whose approach we used, they didn't notice - is this one? And here's a really cool visualization here you can see that the store this store is closed right and just after, oh, my god, we run a we run out of eggs and just before, oh my god go and get the milk before the store closes. Alright and here again closed bang right, so this third-place winner actually deleted all of the closed store rows before they started doing any analysis. Right so remember how we talked about like don't touch your data unless you, first of all analyze, to see whether that thing you're doing is actually okay, no assumptions right. So, in this case I am sure, like I haven't, tried it, but I'm sure they would have one otherwise right because like well, though there weren't actually any store closures to my knowledge in the test set period. The problem is that their model was trying to fit to these like really extreme things and so and because it wasn't able to do it very well, it was gon na end up getting a little bit confused. It's not gon na break the model, but it's definitely gon na harm it, because it's kind of trying to do computations to fit something which it literally doesn't have the data for your neck.

Can you pass that back there all right, so that Russman model again like it's nice to kind of look inside to see what's actually going on right and so that Russman model? I want to make sure you kind of know how to find your way around the code, so you can answer these questions for yourself, so it's inside columnar model data. Now um. We started out by kind of saying: hey if you want to look at the code for something you couldn't like a question mark question mark like this and oh okay, I need to I haven't, got this reading, but you can use question mark question mark to get The source code for something right, but obviously like that's, not really a great way, because often you look at that source code and it turns out. You need to look at something else right and so for those of you that haven't done much coding, you might not be aware that almost certainly the editor you're using probably has the ability to both open up stuff directly off SSH and to navigate through it. So you can jump straight from place to place right so want to show you what I mean. So if I were to find columnar model data - and I have to be using vim here - I can basically say tag columnar model data and it will jump straight to the definition of that plus right. And so then, I notice here that, like oh, it's actually building up a data loader, that's interesting.

If I get control right square bracket, it'll jump to the definition of the thing that was under my cursor and after I finished reading it for a while. I can hit ctrl T to jump back up to where I came from right and you kind of get the idea right or if I want to find it for usage of this in this file of columnar model data, I can hit star to jump to the Next place, it's new used, you know and so forth. Alright, so in this case, get learner was the thing which actually got the model and we want to find out what kind of model it is, and apparently it uses a I'm not using collaborative filtering are. We were using columnar model data, sorry columnar model data, okay learner, which users - and so here you can see mixed input model is the pytorch model, and then it wraps it in the structured learner, which is the the first day. I learn a type which

wraps the data and the model together. So if we want to see the definition of this actual pytorch model, I can go to control right square bracket to see it right, and so here is the model right and nearly all of this we can now understand right. So we got past, we got past a list of embedding sizes in the mixed model that we saw. Does it always expect categorical and continuous together? Yes, it does and the the model data behind the scenes if there are no none of the other type, it creates a column of ones or zeros or something okay. So if it is null, it can still work.

Yeah yeah yeah it's kind of ugly and hacky and will you know hopefully improve it, but yeah? You can pass in an empty list of categorical or continuous variables to the model data and it will basically yeah it'll, basically pass an unused column of zeros. To avoid things breaking and I'm I'm leaving fixing some of these slightly hacky edge cases, because height or 0.4, as well as you're, getting rid of variables. They're going to also add rank 0 tensors, which is to say, if you grab a single thing out of. Like a rent, 110, sir, rather than getting back at a number which is like qualitatively different you're, actually going to get back like a tensor that just happens to have no rank now it

9. 00:53:20

- **"How to write something that is different than Fastai library"**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Turns out that a lot of this kind of codes gon na be actually easier to write, then so, and for now it's it's a little bit more happier than it needs to be Jeremy. You talk about this a little bit before, where maybe it's a good time at some points, talk about how can we write something that is slightly different for worries in the library yeah? I think we'll cover that a little bit next week that I'm mainly going to do that in part to like Pat who's going to cover quite a lot of stuff. One of the main things were cover in part. Two is what it called generative models, so things where the output is a whole sentence or a whole image. But you know I also dig into like powder really either customize the first day I library or use it on more custom models. But if we have time we'll touch on it a little bit next week, okay, so the the learner we were passing in a list of embedding sizes and, as you can see, that embedding sizes list was literally just the number of rows and the number of columns In each embedding right and the number of code rose was just coming from literally how many stores are there in the store category, for example, and the number of columns was just a quarter that divided by two and a maximum of 50. So that thing that list of tuples was coming in, and so you can see here how we use it right, we go through each of those tuples grab the number of categories and the size of the embedding and construct an embedding all right, and so that's a That's a list right, one minor thing, height or specific thing we haven't talked about before is for it to be able to like register.

Remember how we kind of said, like it registers your parameters. It registers your your layers like someone we like listed the model. It actually printed out the Novation varying an age bias. It can't do that if they're hidden inside a list right, they have to be like a there, have to be a an actual n n dot module subclass. So there's a special thing called an N n dot module list which takes a list and it basically says I want you to register. Everything in here has been part of this model. Okay, so it's just a minor tweak, so yeah. So our mixed input model has a list of embeddings and then I do the same thing for a list of linear layers right. So when I said

here, 1000 comma 500, this was saying how many activations I wanted featured. My lineal is okay, and so here I just go through that list and create a linear layer that goes from this size to the next size. Okay, so you can see like how easy it is to kind of construct your own, not just your own model, but a kind of a model which you can pass parameters to have a constructed on the fly dynamically and that's normal talk about next week. This is initialization, we've mentioned climbing her initialization before, and we mentioned it last week and then drop out same thing right. We have here a list of how much drop out to apply to each layer right so again. Here it's just like go through each thing.

In that list and create a drop out layer for it, okay, so this constructor, we understand everything in it except for batch norm, which we don't have to worry about for now, so that's the constructor, and so then the forward also, you know all stuff we're aware Of go through each of those embedding layers that we just saw and remember, we've just treated like as a function, so call it with the ithe categorical variable and then concatenate them all together, put that through drop out and then go through each one of our linear Layers and call it apply relia to it, apply, dropout and then finally apply the final linear layer and the final linear layer has this as its size, which is here right size, one there's a single unit, sales, okay. So we're kind of getting to the point where oh and then, of course, at the end, if this I mentioned would come back to this, if you passed in a Y underscore range parameter, then we're going to do the thing we just learned about last week, which Is to use a sigmoid right, and this is a cool little trick to make you're not just to make your collaborative filtering better. But in this case my basic idea was you know: sales are going to be greater than zero and probably less than the largest sale. They've ever had so I just pass in that as Y range, and so we do a sigmoid and multiply with the sigmoid by the range that I passed it all right. And so hopefully we can find that here yeah here it is right.

So I actually said: hey, maybe the range is between zero and you know the highest x. One point two: you know cuz, maybe maybe the next two weeks we have one bigger, but this is kind of like again try to make it a little bit easier for it to give us the kind of results that it thinks is right. So, like increasingly, you know, I'd love your wall to kind of try to not treat these learners and models as black boxes, but to feel like you now have the information you need to look inside them and remember. You could then copy and paste this plus paste it into a cell in duple, notebook and start fiddling with it to create your own versions. Okay, I think what I might do is we might take a bit of a early break because we've got a lot to cover and I want to do it all in one big go. So, let's take a let's take a break until 7:45 and then we're going to come back and talk about recurrent neural networks, all right,

- PAUSE

10. 00:59:55

- **More into SGD with 'lesson6-sgd.ipynb' notebook, a Linear Regression problem with continuous outputs. 'a*x+b' & mean squared error (MSE) loss function with 'y_hat'**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

So we're going to talk about Aaron ends before we do. We've got to kind of dig a little bit deeper into SGD because I just want to make sure everybody's totally comfortable with with

SGD, and so what we're going to look at is we're going to look at lesson: six SGD notebook and we're going to look at A really simple example of using SGD to learn y equals ax, plus B, and so what we're going to do here is we're going to create, like the simplest possible model y equals ax, plus, B, okay, and then we're going to generate some random data. That looks like so so: here's our X and here's our Y. We want to predict Y from X and we passed in 3 & amp 8 as our a and B, so we're going to kind of try and recover that right, and so the idea is that if we can solve something like this, which has two parameters, we Can use the same technique to solve? We can use the same technique to solve something with a hundred million parameters right without any changes at all. So, in order to find a and a B that fits this, we need a loss function and this is a regression problem because we have a continuous output so for continuous output regression, we tend to use, mean squared error, all right and obviously all of this stuff. There's there's implementations in non pious implementations in flight or we're just doing stuff by hand. So you can see all the steps right.

So there's MSE, okay y hat is we often call our predictions Y hat mitis y squared mean there's, I meant whatever okay. So, for example, if we had ten and five where a and B then there's our mean square R, squared error three point: two: five: okay, so if we've got an A and a B and we've got an x and a y, then our mean square error. Loss is just the mean squared error of our linear. That's our predictions and our way. Okay! So there's a last four ten five X Y, all right! So that's a loss function right, and so when we talk about combining linear layers and loss functions and optionally nonlinear layers, this is all we're doing right is we're putting a function inside a function. Yeah, that's that's all like. I know people draw these clever, looking dots and lines all over the screen when they're saying this is what a neural network is, but it's just it's just a function of a function of a function. Okay, so here we've got a prediction: function: being a linear layer, followed by a loss, function being MSE, and now we can say like oh well, let's just define this as MSA Lost's and we'll use that in the future. Okay, so there's our loss function, which incorporates our prediction function: okay, so let's

11. 01:02:55

- **Gradient Descent implemented in PyTorch, 'loss.backward()', '.grad.data.zero_()' in 'optim.sgd' class**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Generate 10,000 items or thick data and let's show them in two variables, so we can use them with pytorch, because Jeremy doesn't like taking derivatives so we're going to use pytorch for that and let's create random, wait for a and B. So a single random number and we want the gradients of these to be calculated as we start computing with them, because these are the actual things we need to update in our SGD okay, so here's our a and B 0.029 0.111 all right. So let's pick a learning rate, okay and let let's do 10,000 epochs of SGD. In fact, this isn't really SGD. It's not stochastic gradient. It said this is actually full gradient descent we're going to each each loop is going to look at all of the data. Okay, stochastic gradient descent would be looking at a subset each time so to do gradient descent. We basically calculate loss right so remember: we've started out with a random a and B okay, and so this is going to compute some amount of loss and then it's nice from time to time. So one way of saying from time to time is: if the epoch number mod, a thousand is zero right, so every thousand epochs just print out the loss. So you have it. Do it? Okay! So now that we've computed the loss, we can compute our gradients right, and so you just remember this thing here is both a number, a

single number, that is our lost, something we can print, but it's also a variable because we passed variables into it and therefore It also has a method type backward, which means calculate the gradients of everything that we asked it to.

Everything where we said requires radical is true. Okay, so at this point we now have a dot grad property inside a and inside P, and here they are here - is that grant grad property? Okay. So now that we've calculated the gradients for a and B, we can update them by saying a is equal to whatever it used to be the learning rate times. The gradient, okay, dot data, because a is a variable and a variable contains a tensor and it's dot. Data property - and we again this is going to disappear in height, which point four, but for now it's actually the ten so that we need to update okay, so update the tensor inside here, with whatever it used to be the learning rate times. The gradient. Okay and that's basically it all right, that's basically all gradient. Descent is okay, so it's it's as simple, as we claimed there's one extra step in pytorch, which is that you might have like multiple different loss functions or like lots of lots of output layers, all contributing to the gradient, and you like to have to add Them all together, and so, if you've got multiple loss functions, you could be calling loss, stop backward on each of them and what it does is an ad sit to the gradients right, and so you have to tell it when to set the gradients back to zero. Okay, so that's where you just go: okay, set a to zero and gradients in set B gradients to zero, okay, and so this is wrapped up inside the you know: op TMS, JD class right so when we say up Tim dot, SGD - and we just say you Know dot step, it's just doing these for us.

So when we say dot, zero gradients is just doing this force and this underscore here every pretty much every function. That applies to a tensor in pytorch. If you stick an underscore on the end, it means do it in place. Okay, so this is actually going to not return a bunch of zeros, but it's going to change this in place to be a bunch of zeros. So that's basically it we can look at the same thing without pytorch, which

12. 01:07:05

- **Gradient Descent with Numpy**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Means we actually do have to do some calculus. So if we generate some fake data again, we're just going to create 50 data points this time just to make this fast and easy to look at, and so let's create a function called update right, we're just going to use numpy no pytorch, okay. So our predictions is equal to again linear and in this case we actually gon na calculate the derivatives. So the derivative of the square of the loss is just two times and then the derivative is the vector. A is just that you can confirm that yourself. If you want to - and so here our we're going to update a minus equals learning rate times the derivative of loss with respect to a and for B, it's learning rate times derivative with respect to B, okay, and so what we can do, let's just run all This so just for fun, rather than looping through manually, we can use the map flop, matplotlib, func, animation, command, to run the animate function, a bunch of times, and the animate function is going to run 30 epochs and at the end of each epoch, it's going to Print out on the plot, where the line currently is - and that creates this at all movie okay, so you can actually see that the line moving at a place right. So if you want to play around with like understanding how high torque gradients actually work step-by-step, here's like the world's simplest at all example.

Okay - and you know it's kind of like it's kind of weird to say like that's: that's it like when you're optimizing a hundred million parameters in a neural net, it's doing the same thing, but it it actually is alright. You can actually look at the pytorch code and see it's this. Is it right, there's no trick? Well, we load a couple of minor tricks last time, which was like momentum and atom right that, if you could do it in Excel, you can do it invite them so. Okay,

13. 01:09:15

- **RNNs with 'lesson6-rnn.ipynb' notebook with Nietzsche, Swiftkey post on smartphone keyboard powered by Neural Networks**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

So let's do talk about our lens, so we're now in less than six hour and in notebook and we're going to study Nietzsche, as you should so Nietzsche says. Supposing that truth is a woman? What, then I love this? Apparently all philosophers have failed to understand women. So apparently, at the point that Nietzsche was alive, there was no female philosophers, or at least those that were around didn't understand women either so anyway. So this is the philosopher. Apparently, we've chosen to study it. Leech is actually much less worse than people think he is, but it's a different era. I guess alright, so we're going to learn to write philosophy like Nietzsche and so we're going to do it one character at a time. So this is like the language model that we did in Lesson, four, where we did it a word at the time, but this time we're going to do a character at a time, and so the main thing I'm going to try and convince you is an RNN. Is no different to anything you've already learned, okay and so to show you that going to build it from plain pytorch layers, all of which are extremely familiar already: okay, and eventually we're going to use something really complex, which is a for loop. Okay. So that's when we're going to make a really sophisticated, so the basic idea of our n ends is that you want to keep track of the main thing.

Is you want to keep track of kind of state over long term dependencies? So, for example, if you're trying to model something like this kind of template language right then at the end of your percent comment: blue percent, you need a percent common end percent right, and so somehow your model needs to keep track of the fact that it's like Inside a comment over all of these different characters: right and so this is this idea of state. It's kind of memory right - and this is quite a difficult thing to do with, like just a calm, confident it turns out actually to be possible, but it's it's. You know a little bit tricky where elsewhere, as an iron in it turns out to be pretty straightforward all right. So these are the basic ideas. If you want the stateful representation, where you kind of keeping track of like where are we now have memory, have long term dependencies and potentially even have variable length sequences? These are all difficult things to do with confidence, they're very straightforward, with arid ends. So, for example, SwiftKey a year or so ago, did a blog post about how they had a new language model where they

14. 01:12:05

- **a Basic NN with single hidden layer (rectangle, arrow, circle, triangle), by Jeremy,**
- **Image CNN with single dense hidden layer.**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to

Basically, this is from their blog post. We basically said like, of course, this is what their neural net looks like. Somehow they always look like this on the internet. You know, you've got a bunch of words and it's basically going to take your particular words in their particular orders and try and figure out what the next words going to be, which is to say they built a language model. They actually have a pretty good language model if you've used SwiftKey, they seem to do better predictions than anybody else still. Another cool example was andre. Capaci a couple of years ago showed that he could use character. Level are a 10 to actually create an entire latex document, so he didn't actually tell it in any way what life looks like he just passed. The some may tech text, like this and said, generate more low text text and it literally started writing something which means about as much to me as most math papers. Do this, okay, so we're gon na start with something: that's not an RN and I'm going to introduce Jeremy's patented neural network, notation involving boxes circles and triangles. So let me explain: what's going on as a rectangle is an input, an arrow is a layer as a circle. In fact, every square is a bunch of activate, so every shape is a bunch of activations right. The rectangle is the input activations. The circle is a hidden activations and a triangle is an output. Activations and arrow is a layer, operation right or possibly more than one all right.

So here my rectangle is an input of number of rows. Equal a batch size and number of columns equal to the number of number of inputs number of variables, all right, and so my first arrow. My first operation is going to represent a matrix product, followed by our Lu and that's going to generate a set of activation. Remember activations, like an activation, is a number that an activation is a number, a number that's being calculated by a value or a matrix product or whatever it's a number right. So this circle here represents a matrix of activations. All of the numbers that come out when we take the inputs, we do a matrix product, followed by a value, so we started with batch size, byte number of inputs, and so after we do this matrix operation, we now have batch size by you know. Whatever the number of columns in our matrix product was by number of hidden units, okay, and so if we now take these activations, but it's the matrix and we put it through another operation in this case another matrix product and the softmax, we get a triangle. That's our output, activations, another matrix of activations and again number of roses. Batch size number of columns number is equal to the number of classes. Again, however, many columns our matrix in this matrix product head. So that's a that's a neuro net right.

That's our basic kind of one hidden layer, neural net and if you haven't written one of these from scratch, try it you know and in fact, in lessons nine ten and eleven of the machine learning course we do this right. We create one of these from scratch. So if you're not quite sure how to do it, you can check out the machine learning costs yeah in general. The machine learning cost is much more like building stuff up from the foundations. Where else this course is much more like best practices, kind of top-down all right, so if we were doing like a cognate with a single, dense, hidden layer, our input would be equal to actually number yeah. That's very implied, watch number of channels by height by width, right and notice that here batch size appeared every time. So I'm not gon na I'm not gon na write it anymore. Okay, so I've removed the batch size. Also the activation function. It's always basically value or something similar for all the hidden layers and softmax at the end for classification. So I'm not going to write that either okay, so I'm kind of edge picture, I'm going to simplify it a little bit alright. So I'm not gon na mention batch size. It's still there we're not going to mention real you or softmax, but it's still there. So here's our input - and so in this case rather than a matrix product, will do a convolution, let's drive to convolution, so we'll skip over every second one

or could be a convolution, followed by a mac spool in either case.

We end up with something which is replaced number of channels with number of filters right, and we have now height, divided by two and width divided by 2. Okay, and then we can flatten that out. Somehow we'll talk next week about the main way. We do that nowadays, which is basically to do something called an adaptive max pooling where we basically get an average across the height and the width and turn that into a vector anyway. Somehow we flatten it out into a vector we can do a matrix product or a couple of matrix products. We actually tend to do in fastai, so that'll be our fully connected layer with some number of activations final matrix product give us some number of classes. Okay, so this is our basic component, remembering rectangles input circle is hidden. Triangle is output, all other shapes represent a tensor of activations. All of the arrows represent a operation or lay operation all right. So now that's going to jump to the one, the first one that we're going to actually try to try to create for NLP and we're going to basically do exactly the same thing as here right and we're going to try and predict the third character. In a three character: sequence, based on the previous two characters, so our input and again remember: we've removed the batch size dimension. We're not saying that we're still here, okay, and also here, I've removed the names of the layer operations entirely. Okay, just keeping simplifying things.

So, for example, our first import would be the first character of each string in our mini batch. Okay and assuming this is one hot encoded, then the width is just. However many items there are in the vocabulary: how many unique characters could we have? Okay, we probably won't really one hot encoder will feed it in as an integer and pretend it's one hot encoded by using an embedding layer which is mathematically, identical, okay, and then we that's going to give us some activations, which we can stick through a fully connected Layer - okay, so we we put that through. If we click through a fully connected layer to get some activations, we can then put that another fully connected layer and now we're going to bring in the input of character to alright. So the character to input will be exactly the same dimensionality as the character one input, and we now need to somehow combine these two arrows together. So we could just add them up, for instance, right because remember this arrow here represents a matrix product, so this matrix product is going to spit out the same dimensionality as this matrix product. So we could just add them up to create these activations, and so now we can put that through another matrix product and of course, remember all these metrics products have a RAL you as well, and this final one will have a softmax instead to create our predicted Set of characters right, so it's a standard, you know two hidden layer.

I guess it's actually three matrix products neural net. This first one is coming through an embedding layer. The only difference is that we're also got a second input coming in here that we're just adding in right, but it's kind of conceptually identical. So let's let's implement that for Nietzsche all right, so I'm not going to use torch text. I'm gon na try not to use almost any fastai, so we can see it all kind of again from raw right. So here's the first 400 characters of the collected works. Let's grab a set of all of the letters that we see there and sort them. Okay and so a set creates all the unique letters, so we've got 85 unique letters in our vocab. Let's pop up, it's nice to put an empty kind of a null or some some kind of padding character in there for padding. So we're gon na put a parenting character at the start. Right, and so here is what our vocab looks like. Okay, so so Kars is our bouquet. So, as per usual, we want some way to map every character, to a unique ID and every unique ID to a character. And so now we can just go through our collected works of niche and grab the index of each one of those characters. So now we've just turned it into this right. So rather than quote PR e, we now have 40 42, 29, okay. So so that's basically the first step and just to confirm. We can now take

each of those indexes and turn them back into characters and join them together and yeah there.

It is okay, so from now on we're just going to work with this IDX list, the list of character members in the connected works of Nietzsche. Yes, so, Jeremy, why are we doing like a model of characters and not a model of words? I just thought it seemed simpler. You know with a vocab of 80-ish items, we can kind of see it better character, level models turn out to be potentially quite useful. In a number of situations, but we'll cover that in part two, the short answer is like you generally want to combine both the word level model and a connect character. Level model like if you're doing, say, translation, it's a great way to deal with unknown. Like unusual words, rather than treating it as unknown anytime, you see a word you haven't seen before. You could use a character level model for that and there's actually something in between the two quarter: byte pair and coding, vpe, which basically looks at at all. Engrams of characters, but we'll cover all that in part two. If you want to look at it right now, then part two of the existing course

15. 01:23:25

- **Three char model, question on 'in1, in2, in3' dimensions**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Already has this stuff taught and part two of the version 1 of this course, although the NLP stuff is in flight which, by the way so you'll understand it straight away, it was actually the thing that inspired us to move to piped watch because trying to do It in chaos turned out to be a nightmare all right, so let's create the inputs to this. We're actually going to do something slightly different. What I said we're actually going to. I predict the fourth character that actually this the fifth character using the first four. So the index four character using the index zero one two and three okay, so it was exactly the same thing, but with just a couple more layers. So that means that we need a list of the zeroth first, second and third characters. That's why I'm just cutting every character from the start from the one from two from three skipping over three at a time? Okay, so hmm! This is I, I said this wrong, so we're going to predict the third character, the fourth character from the third for the first story. Okay, the fourth character is history, all right, so our inputs will be these three lists right, so we can just use n P dot stack to pop them together, all right, so here's the zero one and two characters that are going to feed into a model and Then here is the next character in the list, so, for example, X, 1, X, 2, X, 3 and Y all right, so you can see. For example, we start off the first.

The very first item would be 40, 42 and 29 right, so that's characters naught 1 and 2 and then we'd be predicting 30. That's the fourth character, which is the start of the next row, all right. So then, 30. 25. 27. We need to predict 29, which is the start of next row and so forth, so we're always using three characters to predict the fourth. So there are 200,000 of these that we're going to try and model right. So we're going to build this, which means we need to decide how many activations, so I'm going to use 256 okay and we need to decide how big our embeddings are going to be, and so I decided to use 42 so about half the number of characters. I have - and you can play around these, so you can come up with better numbers. It's just a kind of experimental and now we're going to build our model. Now I'm gon na change my model slightly, and so here is the the full version so predicting character for using characters. 1 2 & amp 3. As you can see, it's the same picture as a previous page, but I put

some very important coloured arrows here. All the arrows of the same color are going to use the same matrix, the same weight matrix right, so all of our input embeddings are going to use the same matrix all of our layers that go from one layer to the next they're going to use the Same orange arrow weight matrix and then our output will have its own matrix. So we're going to have one two three weight, matrices right and the idea here is the reason.

I'm not gon na have a separate one, but every everything here is that, like. Why would kind of semantically a carrot to have a different meaning, depending if it's the first or the second or the third item, in a sequence like it's not like we're, even starting every sequence, at the start of a sentence we're just arbitrarily chopped it into groups Of three right, so you would expect these to all have the same kind of conceptual mapping and ditto like when we're moving from claritin or character, one you know to kind of say, build up some state here. Why would that be any different kind of operation to moving from character, wonder character to so that's the basic idea. So let's create a three character model and so we're going to create one linear layer for our Green Arrow, one linear layer, fat, orange arrow and one linear layer for our blue arrow and then also one embedding okay. So the embedding is going to bring in something with size, whatever it was 84, I think vocab size and spit out something with an factors in the embedding. Well then put that through a linear layer, and then we've got our hidden layers before the output layer. So when we call forward they're going to be passing in one two, three characters, so if each one will stick it through an embedding we'll, stick it through a linear layer and we'll stick it through a value just to do it the character, one character and character.

Three: okay, then I'm going to create this circle of activations here: okay and that matrix I'm going to call H right, and so it's going to be equal to my input, activations. Okay, after going through the value and the linear layer and the embedding right and then I'm going to apply this l hidden so the orange arrow and that's going to get me to here. Okay! So that's what this layer here does and then to get to the next one. I need to reply the same thing and it apply the orange arrow to that okay, but I also have to add in this second input right so take my second input and add in okay, my previous layer, your neck. Could you pass it back three rows? I don't really see how these dimensions are the same from eight and in2 from literature which, from yeah okay, let's go through. So, let's figure out the dimensions together, so self dot E is gon na be of length, 42, okay and then it's gon na go through L in I'm just gon na make it of size, n, hidden, okay, and so then we're going to pass that which is Now size n hidden through this, which is also going to return something of size, n, hidden, okay. So it's a really important to notice that this is square. This is a square weight matrix okay, so we now know that this is of size n hidden into it's. Going to be exactly the same size as in one was, which is n hidden, so we can now sum together. Two sets of activations, both the size n, hidden passing it into here and again it returns something of size, n hidden.

So basically, the trick was to make this a square matrix and to make sure that it's square matrix was the same size as the output of this hidden. Well, thanks for the great question, can you pass that out to you now? Jeremy is summing. The only thing people can do in these cases I'll come back to that in a moment. That's great point: okay, um! I don't like it when I have like three bits of code that look identical and then three bits of codes that look nearly identical but aren't quiet because it's harder to refactor. So I'm going to put a make H into a bunch of zeros so that I can then put H here, and these are now identical. Okay, so that the hugely complex trick that we're going to do very shortly is to replace these three things. With a for loop, okay and it's going to loop through one two and three: that's that's going to be the for loop or actually zero, one, two! Okay! At that point, we'll be able to call it a recurrent neural network, okay, so just to skip ahead a little bit alright, so we create that that model

make sure I've run all these, so we can actually run this thing. Okay, so we can now just use the same columnar model data class that we've used before and if we use from arrays, then it's basically it's going to spit back the exact arrays. We gave it right. So if we pass, if we stack together those three arrays, then it's going to feed us those three things back to our forward method.

So if you want to like play around with training models using like you know as roar approach as possible, but without writing lots of boilerplate, this is kind of how to do it. Here's column, Namit model data from arrays and then, if you pass in whatever you pass in here, right you're going to get back here, okay, so I've passed in three things, which means I'm going to get sent three things: okay, so that's how that works! Batch size! 512, because this is you know, this data is tiny, so I can use a bigger batch size, so I'm not using really much faster, i stuff at all, I'm using fastai stuff, just to save me fiddling around with data loaders and data sets and stuff. But I'm actually going to create a standard ply torch model, I'm not going to create a loner okay. So this is a standard paper model and because I'm using ply towards that means, I have to remember to write CUDA okay, let's tick it on the GPU. So here is how we can look inside at what's going on right, so we can say it er MD train data loader to grab the iterator to iterate through the training set. We can then call next on that to grab a mini batch and that's going to return all of our X's and why tensor? And so we can then take a look at you know: here's our X's, for example, all right, and so you would expect have a think about what you would expect for this length three, not surprisingly, because these are the three things: okay and so then XS 0, Not surprisingly, okay is of length 512 and it's not actually one hot encoded, because we use an embedding to pretend it is okay. And so then we can use a model as if it's a function.

Okay, by passing to it the variable eyes, version of our tensors and so have a think about what you would expect to be returned here. Okay, so not surprisingly, we had a mini batch of 512, so we still have 5 12 and then 85 is the probability of each of the possible vocab items and of course, we've got the log of them because that's kind of what we do in pytorch. Okay, you can see here the softmax alright, so that's how you can look inside alright, so you can see here how to do everything really very much by hand. So we can create an optimizer again using standard pipe torch, so with pytorch, when you use a plate or optimizer, you have to pass in a list of the things to optimize, and so, if you call m dot parameters that will return that list for you And then we can fit and there it goes okay, and so we don't have learning rate finders and sttr and all that stuff, because we're not using a learner so we'll have to manually do learning rate annealing so set the learning rate a little bit lower and Fit again, okay, and so now we can write a little function to to test this thing out. Okay, so here's something called getnext where

16. 01:36:05

- **Test model with 'get_next(inp)',**
- **Let's create our first RNN, why use the same weight matrices ?**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

We can pass in three characters like why full top space right, and so I can then go through and turn that into a tensor with capital T of an array of the character index for each character in that list. So basically turn those into the integers turn. Those into variables pass that to our

model right and then we can do an Arg max on that to grab which character number is it, and in order to do stuff in none pile, and I use two NP to turn that variable into a lumpy array. Right and then I can return that character and so, for example, a capital T, because what it thinks would be reasonable. After seeing why space that seems like a very reasonable way to start a sentence if it was ppl a that sounds reasonable space, th, a that's bouncer e, small, a and D space. That sounds reasonable, so it seems to reflect created something sensible. Alright, so you know, the important thing to note here is our character. Model is a totally standard, fully connected model right. The only slightly interesting thing we did was to kind of do this addition of each of the inputs one at a time. Okay, but there's nothing new conceptually here, we're training it in the usual way, all right, let's now create an errand in so an iron in is when we do exactly the same thing that we did here all right, but I could draw this more simply by saying You know what, if we've got a green arrow going to a circle, let's not draw a green arrow, go into a circle again and again and again so, let's just draw it like this green arrow, going to a circle right and rather than drawing an orange arrow Going to a circle, let's just draw it like this okay, so this is the same picture, exactly the same picture as this one right, and so you just have to say how many times to go around this circle right.

So in this case, if we were to predict character, number n from characters, one through n minus one, then we can take the character. One input get some activations feed that to some new activations that go through remember orange is the hidden to hidden weight, matrix right and each time, we'll also bring in the next character of input through its embeddings okay, so that picture and that picture. I have two ways of writing the same thing, but this one is more flexible because, rather than me having to say hey, let's do it for H. I don't have to draw eight circles right. I can just say I'll, just repeat this, so I could simplify this a little bit further by saying you know what, rather than having this thing as a special case, let's actually start out with a bunch of zeros right and then, let's have all of our characters. Inside here yes yeah, so I was wondering if you can explain it be better. Why are you reusing those why you think oh they're, the same yeah? Where are you you kind of seem to be reusing? The same same weight, matrices weight, matrices yeah? Maybe this is kind of similar to what we did in convolution, your Nets like, if somehow no, I don't think so, at least not that I can see. So the idea is just kind of semantically speaking like this arrow here. This this arrow here is saying, take a character of import and represented, as some says, some set of features right, and this arrow is saying the same thing, take some character and represent as a set of features, and so is this one okay so like.

Why would the three be represented with different weight matrices, because it's all doing the same thing right and this orange arrow is saying kind of transition from character, 0 state to character 1 state 2 characters to state again it's it's the same thing it's like. Why would the transition from character 0 to 1 be different to character from transition from one or two, so the idea is like, but is to like say hey if if it's doing the same conceptual thing, let's use the exact same white matrix. My comment on convolution neural networks is that a filter, or so this apply to multiple places. I think something like a convolution is almost like: a kind of a special dot product with shared weights. Yeah. No, that's. Okay, that's very good point and in fact one of our students actually wrote a good blog post about that last year. We should dig that up. Okay, I totally see where you're coming from, and I totally agree with you all right. So, let's, let's implement this version so this time we're going to do eight as eight sees okay and so let's create a list of every eighth character from zero through seven and then our outputs will be the next character, and so we can stack them together and So now we've got six hundred thousand by eight, so here's an example so, for example, after this series of eight characters right, so this is characters north through eight. This is characters one through nine.

This is two through ten.

These are all overlapping, okay, so after characters one north through eight, this is going to be the next one, okay and then, after these characters, this will be the next one all right. So you can see that this one here has 43. Is its Y value right because after those the next one will be 43? Okay? So so this is the first eight characters. This is two through nine, three through ten and so forth right. So these are overlapping groups of eight characters and then this is the the next one. Okay, so let's create that model. Okay, so again we use from arrays to create a model data class and so you'll see here we have exactly the same code as we had before. There's our embedding Linea hidden output. These are literally identical, okay and then we've replaced our value of the linear input of the embedding with something that's inside a loop, okay and then we've replaced the cell hidden thing, okay, also inside the loop. I just realize didn't mentioned last time. The use of the hyperbolic tan, hyperbolic tan, looks like this okay. So it's just a sigmoid, that's offset right and it's very common to use a hyperbolic tan inside this trend. This state to state transition because it kind of stops it from flying off too high or too low. You know it's nicely controlled back in the old days we used to use hyperbolic tanh or the equivalent sigmoid a lot as most of our activation functions.

Nowadays, we tend to use value, but in these hidden state to here in the hidden state transition weight matrices, we still tend to use hyperbolic tanh. Quite a lot so you'll see I've done that also yeah hyperbolic tanh okay. So this is exactly the same as before, but I've just replaced it with a Pollard and then here's my output. Yes, you know so a does. He have to do anything with convergence. These networks yeah we'll talk about that a little bit over time. Let's, let's, let's come back to that, though, for now we're not really going to do anything special at all. You know recognizing. This is just a standard fully connected Network. You know, interestingly, it's quite a deep one right like because this is actually this that we've got. Eight of these things now we've now got a deep, eight layer Network, which is why units starting suggest we should be concerned. As you know, as we get deeper and deeper networks, they can be harder and harder to train, but let's try training this all right. So when it goes as before, we've got a batch size of 512 we're using Adam and where it goes so we will sit there watching it. So we can then set the learning rate down back to 20 neg 3. We can fit it again and yeah. It's actually, it seems to be training, fun, okay, but we're gon na try something else, which is we're going to use this a trick that your net rather hinted at before, which is maybe we shouldn't be adding these things together, and so the reason you might want To be feeling a little uncomfortable about adding these things together is that the input state and the hidden state are kind of qualitatively different kinds of things right.

The input state is the is the encoding of this character. For us H represents the encoding of the series of characters so far, and so adding them together is kind of potentially going to lose information. So I think what your net was going to prefer that we might do is maybe to concatenate these instead of adding them. So it sound good to you, you know she's, not it okay, so let's now make a copy of the previous cell all the same right rather than using plus, let's use cat okay. Now, if we can cat, then we need to make sure now that our input layer is not from n fac-2 hidden, which is what we had before, but because we're concatenated it needs to be in fact, plus and hidden to end hidden, okay, and so now that's Going to make all the dimensions work nicely, so this now is of size, n fact, plus and hidden. This now makes it back to size n hidden again, okay and then this is putting it through the same square matrix as before. So it's still a size n here, okay, so this is like a good design. Heuristic, if you're designing an architecture is if you've got different types of information that you want to combine. You generally want concatenate it. Okay, you know adding things together, even if

they're the same shape is losing information, okay and so once you've concatenated things together, you can always convert it back down to a fixed size by just tracking it through a matrix product.

Okay. So that's what we've done here again, it's the same thing, but now we're concatenating instead, and so we can fit that and so last time we got one point: seven two, this time you go at one point: six six! So it's not setting the world on fire, but it's an improvement and the improvements of it. Okay, so we can now test that with get next and so now we can pass in eight things right. So it's no before those let's go to a part of that sounds good as well so Queens, and that sounds good too. All

17. 01:48:45

- **RNN with PyTorch, question: "What the hidden state represents ?"**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Right so great, so that's enough manual, hackery, let's see if pytorch couldn't do some of this for us, and so basically what pytorch will do for us is. It will write this loop automatically, okay and it will create these linear input layers automatically, okay and so to ask it to do that. We can use the n n, dot, R and n plus so here's the exact same thing in less code, by taking advantage of height choice and again, I'm not using a conceptual analogy to say player torches doing something like it. I'm saying play torch is doing it now. This is just the code. You just saw wrapped up a little bit, reflect it a little bit for your convenience right, so where we say we now want to create an era. Ten call our it n. Then what this does is it does that for live now, notice that our for loop needed a starting point. You remember why? Right because otherwise our for loop didn't quite work, we couldn't quite refactor it out and because this is exactly the same, this needs our starting point to, and so let's give it a starting point, and so you have to pass in your initial hidden State for reasons That will become apparent later on. It turns out to be quite useful to be able to get back that here in the state. At the end, and just like we could here, we could actually keep track of the hidden state.

We get back to things we get back, both the output and the hidden state right, so we pass in the input in the hidden State when we get back the output and the hidden state. Yes, could you remind us what the hint state represents? The hidden state is H, so it's the it's, the orange circle, ellipse of activations, okay, and so it is of size, 256. Okay, all right, so we can okay, there's one other thing too to know which is in our case, we were replacing H with a new hidden state. The one minor difference in pytorch is they append the new hidden state to a list or to a tensor which gets bigger and bigger, so they actually give you back all of the hidden states. So, in other words, rather than just giving you back the final ellipse, they give you back all the ellipses stacked on top of each other, and so because we just want the final one. I was got indexed into it with minus one here, okay, other than that. This is the same code as before, put that through our output layer to get the correct, vocab size, and then we can train that alright, so you can see here, I can do it manually. I can create some hidden state. I can pass it to that area and I can see the stuff I get back. You'll see that the dimensionality of H, it's actually a rank 3 tensor. Where else in my version it was a. Let's see it was a rank. Two tensor, okay and the difference is here: we've got just a unit axis at the front, we'll learn more about why that is later, but basically it turns out.

You can have a second R and n that goes backwards, alright, one that goes forwards, one that goes backwards from the idea is neck, and then it's going to be better at finding relationships that kind of go backwards. That's quite a bi-directional eridan. Also, it turns out, you can have an error in feed to an iron in that's got a multi-layer eridan. So basically, if you have those things, you need an additional access on your tensor to keep track of those additional layers of hidden state, but for now we'll always have a one yeah and we'll always also get back a one at the end. Okay, so if we go ahead and fit this now, let's actually trade it for a bit longer. Okay, so last time we only kind of did a couple of epochs this time, we're due for a pox. What have we sit at one in egg three and then we'll do another to epochs at one in egg four, and so we've now got our lost down to one point: five, so getting better and better. So here's our get next again. Okay - and you know - let's just it was the same thing, so what we can now do is we can look through like forty times calling get next each time and then each time will replace our input by removing the first character and adding the thing that we Just predicted, and so that way we can like feed in a new set of eight characters that get them again and again, and so that way, we'll call that get next in so here are 40 characters that we've generated.

So we started out with four th OS. So we got four those of the same. The same the same. You can probably guess what happens if you can't predicting the same the same all right so it's you know it's doing. Okay, we we now have something which you know, we've basically built from scratch, and then we've said: here's how high torture effected it for us. So if you want to like, have an interesting little homework assignment this week, try to write your own version of an RNN plus all right like try to like literally like create your like. You know Jeremy's aren't in and then like type in here, Jeremy's aren't in or in your case maybe your name's, not Jeremy, which is okay too, and then get it to run. Writing your implementation, that's fast from scratch. Without looking at the piped water source code, you know like basically it's just a case of like going up and seeing what we did back here right and like make sure you get the same answers and confirm that you do so. That's kind of a good little test, simply simple at all assignment, but I think you'll feel really good when you seem like. Oh I've, just reimplemented an end alone in alright. So I'm going to do one other thing when I switched from this one when I've moved the car one input inside the dotted line right, this dotted rectangle represents the thing I'm repeating. I also watch the triangle, the output. I moved that inside as well.

Now that's a big difference, because now what I've actually done is I'm actually saying spit out an output after every one of these circles, so spit out an output here and here and here alright. So, in other words, if I have a three character input, I'm going to spit out a three character: output, I'm saying half the character 1. This will be next after character to this, be next after character. 3. This will be next, so again, nothing different and again this you know if you want to go a bit further with the assignment. You could write this by hand as well, but basically, what we're saying is in the for loop would be saying, like you know, results equals some empty list right and then would be going through and rather than returning that we're instead be saying, you know, results dot, Append that right and then like return whatever torch, dot, stat, something like that right that it made me right in my question. So now you know we now have like every step. We've created an output okay, so which is basically this picture, and so the reason was lots of reasons. That's interesting, but I think the main reason right now that's interesting is that you probably noticed this. This approach to dealing with our data seems terribly inefficient, like we're grabbing the first eight right, but then this next set all, but one of them overlap the previous one right. So we're kind of like recalculating the exact set of embeddings.

Seven out of eight of them are going to be exact, same embeddings, right, exact, same transitions, it kind of

18. 01:57:55

- **Multi-output model**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Seems weird to like do all this calculation to just predict one thing and then go back and recalculate seven out of eight of them and add one more to the end to calculate the next thing all right. So the basic idea, then, is to say: well, let's not do it that way, instead, let's taking non overlapping sets of characters, all right so like so here is our first eight characters here is the next day. Characters here are the next day characters so like. If you read this top left to bottom right, that would be the whole nature right, and so then, if these are the first eight characters then offset this by one starting here, that's a list of outputs right. So after we see characters zero through seven, we should predict characters 1 through 8. The XS so after 40 should come 42 as it did after 42 should come 29 as it did. Okay, and so now that can be our inputs and labels for that model, and so it shouldn't be any more or less accurate. It should just be the same right pretty much, but it should allow us to do it more efficiently. So let's try that all right. So I mentioned last time that we had a minus 1 index here, because we just wanted to grab the last triangle. Okay, so in this case we're going to grab all the triangles. So this this is actually the way it end on. Rnn creates things we we only kept the last one, but this time we're going to keep all of them. So we've made one change, which is to remove that minus one other than that.

This is the exact same code as before. Okay, so, but there's nothing much to show you here I mean, except of course, at this time, if we look at the labels, it's now 512 by eight factors we're trying to predict eight things every time through. So there is one complexity here, which is that we want to use the negative log likelihood loss function as before, right, but the ligand, if lost likelihood, loss function just like our MSE expects to receive to rank one tensors actually with the mini-batch access to rank two Tensors all right so two to mini-batches of vectors problem is that we've got eight-time steps. You know it characters in an RNN. We call it a time step right. We have eight time steps and then for each one we have 84 probabilities. We have the probability for every single one of those eight times deaths, and then we have that for each of our 512 items in the mini batch. So we have a rank 3 tensor, not a rank two tensor um. So that means that the negative log likelihood loss function is going to spit out an error. Now, frankly, I think this is kind of dumb. You know, I think it would be better if pytorch had written the loss functions in such a way that they didn't care at all about rank and they just applied it to whatever rank you gave it, but for now at least it does care about rick. But the nice thing is, I get to show you how to write a custom loss function. Okay, so we're going to create a special negative log likelihood loss function for sequences, okay, and so it's going to take an input in the target and it's got a call.

F, dot negative log likelihood lost so the pipe launched one all right, but what we're going to do is we're going to flatten our input and we're going to flatten our targets right and so, and it turns out these are going to be the first two axes That I have to be transposed so the way pytorch handles are and end data by default is the first axis is the sequence length in this case eight right, so the sequence length of an R and n is how many times deaths. So we have eight

characters, so a sequence length of eight. The second axis is the batch size and then, as would expect, the third axis is the actual hidden state itself. Okay. So this is going to be eight by 512 by n, hidden, which I think was 256 yeah okay, so we can grab the size and unpack it into each of these sequence, length batch size and I'm hidden now, target mighty dot size is 512 by 8. Where else this one here was 8 by 512, so to make them match we're going to have to transpose the first two axis: okay, pytorch, when you do something like transpose, doesn't generally actually shuffle the memory order, but instead it just kind of Keeps some internal metadata to say like hey, you should treat this as if it's transposed and some things in pytorch will give you an error if you try and use it when it has these like this internal state, and I basically say error, this tensor is Not contiguous, if you ever see that error at the word contiguous after it and it goes away, so I don't know they can't do that for you apparently.

So in this particular case I got that error, so I wrote the code contiguous after it, okay and so then. Finally, we need to flatten it out into a single vector, and so we can just go a dot view, which is the same as non PI: dot reshape and minus one means as long as it needs to be okay and then the input again. We also reshape that right, but remember the input. Sorry, the the the predictions also have this axis of length. 84. All of the predicted probabilities. Okay, so so here's a custom. These are custom lost function. That's it right! So if you ever want to play around with your own loss functions, you can just do that like so and then pass that to fit okay. So it's important to remember that Fitch. Is this like lowest level, fastai abstraction, that's --! It's that this is the thing that implements the training, look, okay and so like you're, the stuff you pass it in is all standard pytorch stuff, except for this, this is our model data object. This is the thing that wraps up the test set.

19. 02:05:55

- **Question on 'sequence length vs batch size'**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

The training set and the validation set to get that okay, your neck. Could you pass that back? So when we pull the triangle into the replicator structure right, so the the first n minus one iterations of the sequence length, we don't see the whole sequence length yeah. So does that mean that the batch size should be much bigger so that be careful? You don't mean that size, you main sequence length right, because the batch size is like some firing. Yeah. Okay! So yes, yes, if you have a short sequence length like eight yeah, the first character has nothing to go on. It starts with an empty hidden state of zeros, okay. So what we're going to start with next week is we're going to learn how to avoid that problem right, and so it's a really insightful question or concern right, and but if you think about it, the basic idea is: why should we reset this to zero? Every time you know like, if we can kind of line up these mini batches somehow so that the next mini batch joins up correctly. It represents like the next letter in leaches works. Then we'd want to move this up into the constructor right and then like pass that here and then store it here right and now we're not resetting the hidden state each time we're actually we're actually keeping the hidden state from call to call, and so the only Time that it would be failing to benefit from learning state would be like literally at the very start of the document.

So that's where, but that's where we're going to try and ahead next week. I feel like this lesson. Every time I've got a punch line coming. Somebody asks me a question where I have

to like: do the punch line ahead of time. Okay, so we can fit that and we can fit that, and I want to show you something interesting, and this is coming to the punch line that another punch line that you net try to spoil, which is when we're you know remember. This is just doing a loop right, applying the same matrix multiply again and again, if that matrix multiply tends to increase the activations each time then effectively we're doing that to the power of eight right. So it's going to like to shoot off really high or if it's decreasing it a little bit each time, that's going to shoot off really low. So this is what we call a gradient explosion right, and so we really want to make sure that the initial H naught H the initial. But if we call it, the initial L hidden that we create is is

20. 02:09:15

- **The Identity Matrix (init!), a paper from Geoffrey Hinton "A Simple Way to Initialize Recurrent Networks of Rectified Linear Units"**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Like oversize, that's not going to cause our activations on average to increase or decrease right and there's actually a very nice matrix. That does exactly that called the identity matrix, so the identity matrix for those that don't quite remember their linear algebra is this. This would be a size, 3 identity, matrix all right, and so the trick about an identity matrix is anything times. An identity. Matrix is itself right, and so, therefore you could multiply it by this again and again and again and again and still end up with itself right, so there's no gradient explosion, so what we could do is instead of using whatever the default random in it is for This matrix we could instead, after we create our errand in, is we can go into that Erol in right and notice. This right, we can go m, dot, RN, n right and if we now go like so, we can get the docs for m dot, R and M right and as well as the arguments for constructing it, it also tells you the inputs and outputs for calling the Layer - and it also tells you the attributes, and so it tells you there's something called weight, H H and these are the learn about hidden to hidden weights. That's that square matrix right. So after we've constructed our M, we can just go in and say all right, m dot, R and n dot weight, h, HL, dot data, that's the tensor dot copy underscore in place torch. I that is I, for identity. In case you are wondering.

So this is an identity matrix of size n hidden, so this both puts into this weight matrix and returns the identity matrix, and so this was like. Actually, a Geoffrey Hinton paper was like hey, you know, after it was it's 2015, so after recurrent neural Nets have been around for decades. Here's like hey gang. Maybe we should just use the identity matrix to initialize this and like it actually turns out to work really. Well, and so that was a 2015 paper believe it or not, from the father of neural networks - and so here is the here is our implementation of his paper, and this is an important thing to know right when very famous people, like Geoffrey Hinton, write a paper. Sometimes in entire implementation of that paper looks like one line of code. Okay, so let's do it before we got point six one, two, five, seven we'll fit it with exactly the same parameters, and now we get 0.5 1 and in fact, can keep training 0.50. So like this tweak really really really helped. Okay, now one of the nice things about this tweak was before I could only use a learning rate of one in x3 before it started going crazy, but after identity matrix I found I could use one in egg too, because it's you know it's better behaved weight. Initialization I found I could use a higher learning rate. Okay and honestly, these things, you know increasingly we're trying to incorporate into the defaults in first day.

I you know you don't necessarily personally need to actually know them, but you know at this point we're still at a point where you know most things in most libraries, most of the time don't have great defaults, it's good to know all these little tricks. It's also nice to know if you want to improve something, what kind of tricks people have used elsewhere, because you can often borrow them yourself all right. Well, that's the end of the lesson today and so next week we will look at this idea of a stateful RNN, that's going to keep this hidden state around and then we're going to go back to looking at language models again and then. Finally, we're going to go all the way back to computer vision and learn about things like rez nets and batch norm and all the tricks that were in figured out in cats versus dogs, see you, then [Applause,]

Lesson 7: Resnets from scratch

Outline

We finish off our recurrent neural network from scratch implementation from last week, and introduce the GRU and LSTM cells to allow training of long sequences with RNNs.

Then we complete this part of the course with a return to computer vision, where we implement the powerful resnet architecture and batch normalization layer from scratch. Congratulations on completing the course! Be sure to let us know on the forums about what projects you have built, and what you're planning next…

Video Timelines and Transcript

1. 00:03:04

- **Review of last week lesson on RNNs,**
- **Part 1, what to expect in Part 2 (start date: 19/03/2018)**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Okay, so last class of part one. I guess the theme of part one is classification and regression with deep wading and specifically it's about identifying and learning the best practices for classification and regression, and we started out with the kind of here are three lines of code to do: image, classification and gradually, we've Been for the first four lessons within kind of going through NLP, structured data, collaborative filtering and kind of understanding, some of the key pieces and, most importantly, understanding. You know how to actually make these things work well in practice, and then the last three lessons are then kind of going back over all of those topics, in kind of reverse order to understand more detail about what was going on and understanding what the code looks Like behind the scenes and wanting to kind of write them from scratch, part two of the course we'll move from a focus on classification and regression, which is kind of predicting a thing like a number or or at most a small number of things. Like a small number of labels and we'll focus more on generative, modeling, generative modeling means predicting kind of lots of things, for example, creating a sentence such as in Ural, translation or image, captioning or question-answering, while creating an image such as in style, transfer, super-resolution segmentation and So forth and then in part, two it'll move away from being just here are some best practices.

You know established best practices, either through people that are written papers or through research. That last day is done, and it kind of got convinced that these are best practices to some stuff, which would be a little bit more speculative. You know some stuff, which is maybe recent papers that haven't been fully tested yet and sometimes in part. Two, like pickles, will come out in the middle of the course and will change direction with the course and study that paper, because it's just you know interesting, and so, if you're interested in kind

of learning a bit more about how to read a paper and how To implement it from scratch and so forth, then that's another good reason to do part two. It still doesn't assume any particular math background, but it does beyond kind of high school, but but it does assume that you're prepared to spend time, like you know, digging through the notation and understanding it and converting it to code and so forth. All right so we're we're up to is is our intent at the moment, and I think one of the issues I find most with teaching iron ends is trying to ensure that people understand they're, not in any way different or unusual or magical they're they're. Just a standard fully connected Network, and so let's go back to the standard, fully connected Network which looks like this right, so to remind you, the arrows represent one or more layer operations. Generally speaking, a linear, followed by a nonlinear function.

In this case, their matrix modifications, followed by real new raw or fan and the arrows of the same color represent the same, exactly the same weight matrix being used, and so one thing which was just slightly different from previous fully connected networks, we've seen, is that we Have an input coming in at the not just at the first layer, but also for the second layer and also at the third layer, and we tried a couple of approaches. One was concatenating the inputs and one was adding the airport's okay. But there was nothing at all conceptually different about this, so that code looked like this. We had a model where we basically defined the the three arrows colors. We had as three different weight: matrices, okay and by using the linear. We got actually both the weight matrix and the bias vector wrapped up for free for us, and then we went through and we did each of our embeddings put it through our first linear layer, and then we did each of our. We call them hiddens. Being the orange orange areas and in order to avoid the fact that there's no orange arrow coming into the first one, we decided to kind of invent an empty matrix and that way every one of these rows about the same right. And so then we did exactly the same thing except we used to loop just to refactor the cutter okay. So it's just. It was just a code refactoring.

There was no change of anything conceptually and since we were doing a refactoring, we took advantage of that to increase the number of characters to eight, because I was too lazy to type 8 when the alias, but I'm quite happy to change the loop in that stage. Yeah, so this now looked through this exact same thing, but we had eight of these rather than three. So then we refactored that again by taking advantage of an end errand in which basically puts that loop together for us and keeps track of the this.h as it goes along for us and so by. Using that we were able to replace the loop with a single call, and so again that's just a refactoring doing exactly the same thing. Okay, so then we looked at something which was mainly designed to save some training time, which was previously we had if we had a big piece of text right. So we've got like a movie review, but we were basically splitting it up into eight character, segments and we'd grab like segment number one and use that to predict the next character right. But in order to make sure that we kind of used all of the data we didn't just put it up like that, we actually said like okay, here's our whole thing, let's grab the first will be to grab this section. The second will be to grab that section in that section, then that section and each time would predict predicting the next one character, a lot okay, and so you know I was bit concerned that that seems pretty wasteful because, like as we calculate this section, nearly all Of it overlaps with the previous section, okay, so instead what we did was we said all right.

Well, what if we actually did split it into non-overlapping pieces right - and we said all right - let's grab this section here and use it to predict every one of the characters, one along right and then let's grab this section here and use it to predict every one Of the characters went along so after we look at the first character in we try to predict the second character and then now, if we look at the second character, we try to predict the third character and so okay, and

so that's where you've got to and Then what if you perceptive folks, asked a really interesting question or expressed their concern, which was hey after we got through the first? The first point here after we got through the first point here, we kind of withdrew away our H, activations and started a new one which meant that when it was trying to use character, one to predict character, it's got nothing to go on. You know it hasn't built, it's only built, it's only done one linear layer, and so that seems like a problem which indeed it is okay.

2. 00:08:48

- **Building the RNN model with 'self.init_hidden(bs)' and 'self.h', the "back prop through time (BPTT)" approach**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

So we're going to do the obvious thing, which is: let's not throw away H. Okay, so let's not throw away that that matrix at all so in code. The big problem is here, but every time we call forward so in other words, every time we do a new mini-batch we're creating our our hidden state right, which remember, is the orange circles. Okay, we're resetting it back to a bunch of zeros, and so, as we go to the next non-overlapping section, we're saying forget everything that's come before, but in fact the whole point is we know exactly where we are we're at the end of the previous section and About to start the new next contiguous section, so let's not throw it away so instead the idea would be to cut this out right, move it up to here: okay, store it away in self and then kind of keep updating it right now. So we're going to do that and there's going to be some minor details to get right. So, let's start by looking at the model, so here's the model, it's it's nearly identical and okay, here's the model, it's nearly identical, but I've got as expected. One more line in my constructor, where I call something called init hidden and as expected in it hidden sets self dot H to be a bunch of zeros. Okay, so that's entirely unsurprising and then, as you can see, our R and n now takes in self garage and it as before, spits out our new hidden, activations and so now the trick is to now store that away inside self dot, H and so here's wrinkle Number one, if you think about it, if I was to simply do it like like that right and now I train this on a document.

That's I don't know a million words million characters. Long then, the size of this unrolled are a 10 is has a million circles here, and so that's fine going forwards right there. When I finally get to the end - and I say, here's my character and actually remember we're doing multi output now so multi output looks like this right or if we were to draw the unrolled version of multi output, we would have a triangle coming off at every Point: okay, so the problem is that then, when we do back propagation, we're calculating, you know how much does the error at character? One impact the final answer: how much does the error character to impact the final answer and so forth, and so we need to go back through and say like how do we have to update our weights based on all of those you know errors, and so, if There are our million characters. My unrolled R and N is a million layers long. I have a 1 million layer fully connected Network. All right and like I didn't - have to write the million layers because I have for loop and the for loops hidden away behind that. You know the self dot, R and n, but it's still there right, we so so this is actually a 1 million layer fully connected Network, and so the problem with that is it's going to be very memory intensive, because in order to do the chain rule, I Have to be able to multiply at every step, like you know, after a few times she acts right and so, like I've got.

That means I have to remember that those values you the value of every set of layers, so I'm gon na have to remember all those million layers and I'm going to do - have to have to do a million multiplications and I'm going to have to do that. Every batch okay, so that would be bad so to avoid that we basically say all right well from time to time. I want you to forget your history, okay, so we can still remember the state right, which is to remember, like what's the values in our hidden matrix right, but we can remember the state without remembering everything about how we got there. So there's a little function called repackage variable, which literally is just this right. It just simply says: grab the tensor out of it right because remember, the tensor itself doesn't have any concept of history right and create a new variable out of that, and so this variables going to have the same value, but no no history of operations and therefore, When it tries to back propagate it all it'll stop there. So, basically, what we're going to do, then, is we're going to call this in our forward. So that means it's going to do add characters it's going to back propagate through eight layers. It's going to keep track of the actual values in our hidden state, but it's going to throw away at the end of those eight, it's its history of operations. So this is this approach.

It's called back prop through time, and you know when you read about it online people make it sound like like a different algorithm or some big insight or something, but it's it's not at all right. It's just saying: hey after our for loop, you know just throw away your your history operations and start afresh, so we're keeping our hidden state but we're not keeping our hidden States history. Okay, so that's that's! Wrinkle number one! That's what this repackage bar is doing, and so what do you see? Bp, BP TT, that's referring to that crop through time and you might remember. We saw that in our original errand in Lesson we had a variable called BP t t equals 70, and so when we set that they're actually saying how many layers backprop through another good reason not to back crop through too many layers is, if you have any Kind of gradient instability like gradient explosion or gradients, banishing you know too many more of the more layers you have, the harder, the network s to Train so smaller and less resilient on the other hand, and longer value for VP TT means that you're able to explicitly Capture a longer kind of memory, more state, okay, so that's a that's something that you get to tune when you create your area, all right, wrinkle number, two is: how are we going to put the data into this right like it's all, very well? The way I described it just now where we said you know we could do this, and we can first of all look at this section.

Then this section in this section, but we're going to do a mini batch at a time right. We want to do a bunch at a time, so, in other words, we want to say: let's do it like this, so mini-batch number one would say: let's look at this section and predict that section and at the same time, in parallel, let's look at this totally Different section and predict this and at the same time, in parallel, let's look at this totally different section and predict this right, and so then, because remember in our in our hidden state, we have a vector of hidden state for everything in our mini batch right. So it's going to keep track of at the end of this is going to be a you know, a vector here, a vector here, a vector here, and then we can move across to the next one and say okay, so this part of the mini batch use. This to predict that and use this to predict that and use this to predict that right, so you can see that we're moving, that we've got like a number of totally separate bits of our text that we're moving through in parallel right. So hopefully this is going to ring a few bells for you, because what happened was

3. 00:17:50

- **Creating mini-batches, “split in 64 equal size chunks” not “split in chunks of size 64”, questions on data augmentation and choosing a BPTT size, PyTorch QRNN**

Was back when we started looking at torch texture, the first time we started talking about how it creates these mini batches? And I said what happened: was we took our whole big long document consisting of like you know the entire works of nature or all of the IMDB reviews, concatenated together or whatever, and a lot of a lot of you? Not surprisingly, because this really said this is really weird at first, a lot of you didn't quite hear what I said correctly. What I said was we split this into 64, equal sized chunks and a lot of your brains when Jeremy just said, we split this into chunks of size 64, but that's not what Theresa Jeremy said. We split it into 64: equal sized chunks right. So if this whole thing was length, 64 million right, which would be a reasonable sized corpus, not an unusual size corpus, then each of our 64 chunks would have been of length 1 million right, and so then what we did was. We talked the first chunk of 1 million and we put it here and then we took the second chunk of 1 million and we put it here - the third chunk of 1 million. We put it here and so forth to create 64 chunks and then H. Mini-Batch consisted of us going - let's split this down here and here and here and each of these is of size, BP te T, which I think we had something like 70 right, and so what happened was we said. Alright, let's look at our first mini batch.

Is all of these right, so we do all of those at once and predict everything accrue off set by one and then at the end of that first mini batch. We went to the second chunk right and used each one of these to predict the next one. Offset by one ok, so that's that's why we did that slightly weird thing right is that we wanted to have a bunch of things we can look through in parallel, each of which, like hopefully, a far enough away from each other. You know that we don't have to worry about the fact that you know the truth. Is this starting? The start of this million characters was actually in the middle of a sentence, but you know who cares right because it's you know that only happens once every million characters honey. I was wondering if you could talk a little bit more about augmentation for this kind of data set and how to data augmentation of this kind of data said yeah. No, I can't because I don't. I really know a good way. It's one of the things I'm going to be studying between now and part two. There have been some recent developments, particularly something we talked about. The machine learning course - and I think, we've refinished in here, which was somebody for a recent careful competition, won it by doing data augmentation by randomly inserting parts of different rows basic, something like that may be useful here and I've seen it.

I've seen some papers that do something like that, but yeah I haven't seen any kind of recent ish state-of-the-art new NLP papers that that are doing this kind of data orientation. So it's something we're planning to work on, so it's generally how the issues be PTT. So there's a couple of things to think about when you pick your be PTT. The first is that you'll note that the the matrix size for a mini batch has a B PTT, the the TT by batch size. So one issue is your GPU Ram needs to be able to fit that by your embedding matrix racks. Every one of these is going to have B of length embedding length plus all of the hidden state. So one thing is to you know: if you get a cruder out of memory error, you need to reduce one of those if you're, finding your training is very unstable, like your loss is shooting off to LAN. Suddenly, then, you could try to decreasing your B PTT because you've got less layers to gradient explode through it's too slow. You could try decreasing your B PTT because it's going to kind of do one of those steps at a time like that for loop can't be paralyzed. Well, I say that there's a recent thing called QR, an N which is will hopefully talk about in part two which kind of does paralyze it, but the versions we're looking at, don't paralyze it. So there would be the main issues. I think before look at performance.

Look at memory and look at stability and try and find a number: that's you know as high as you can make it, but all of those things work for you, okay, so trying to get. Although that chunking and lining up and anything to work is more code than I want to write so for this section we're going to go back and use torched s together, okay, so when you're using AP is like fastai

4. 00:23:41

- **Using the data formats for your API, changing your data format vs creating a new dataset class, 'data.Field()'**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

And torch text, which in these case these two API, is a desire to, or at least from the first day I site, designed to work together, you often have a choice which is like okay. This API has a number of methods that expect the data in this kind of format, and you can either change your data to fit that format or you can write your own data set subclass to handle the format that your data is already in. I've noticed on the forum. A lot of you are spending a lot of time. Writing your own dataset classes, whereas I am way lazier than you and I spend my time instead changing my data to fit the data set classes. I have like. I that's fine and if you realize, like oh there's, a kind of a format of data that me and other people are likely to be seen quite often, and it's not in the first day our library, then by all means, write the data set subclass it submitted As a PR and then everybody can benefit, you know, but

5. 00:24:45

- **How to create Nietzsche training/validation data**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

In this case, I just kind of thought I want to have some Nietzsche data fed into torch text. I'm just going to put it in the format that watch text kind of already support, so torch text already has, or at least the first day I wrap her around Twitter text already has something where you can have a training path and a validation path, and you Know one or more text files in each path containing a bunch of stuff, that's concatenated together for your language model. So, in this case, all I did was I made a copy of my nature file copied it into training, made another copy, stuck it into the validation and then in one of the you know in the training set. I did I deleted the last twenty percent of rows and in the validation set I deleted all, except for the last one Center for us, and I was done right, so I found this that in this case I found that easier than writing a custom dataset class. The other benefit of doing it. That way was that I felt like it was more realistic to have a validation set that wasn't a random shuffled set of rows of text. That was like a totally separate part of the corpus because I feel like in practice, you're very often going to be saying. Like oh, I've got, I don't know these books or these authors I'm learning from, and then I want to apply it to these different books and these different authors, you know, so I felt like for getting a more realistic validation of my nietzsche model.

I should use like a whole separate piece of the text, so in this case it's the last. You know 20 % of the rows if the corpus, so I haven't, created this for you right intentionally, because you

know this is the kind of stuff I want to do. Practicing is making sure that you're familiar enough comfortable enough with with batch or whatever, that you can create these and that you understand what they need to look like and so forth. So in this case you can see I've now got. You know a train and a validation here, and then I could yeah okay, so you can see. I've literally just got one file in it because it's a fire when you're doing a language model, ie predicting the next character or predicting the next word. You don't really need separate files. It's fine if you do have separate files, but they just get capped native together anyway. Alright, so that's my source data, and so here is you know the same lines of code that we've seen before and let's go over them again. So it's a couple of lessons ago right so in torch text we create this thing called a field and the field initially is just a description of how to go about pre-processing the test. Okay, now you in this case, I'm gon na, say: hey lowercase it. You know cuz, I don't mean now, I think about it. There's no particular reason to have done this lower case upper case would work fine too, and then how do I talk maser, and so you might remember last time we used a tokenization function, which kind of largely spit on white space and try to do some flavor Things with punctuation right and that gave us a word model in this case.

I want to character model, so I actually want every character put into a separate token. So I could just use the function list in Python because list in Python does that? Okay, so this is where you can kind of see like understanding how libraries like torch text and fast ar e are designed to be extended, can make your life a lot easier right. So when you realize that very often both of these libraries kind of expect you to pass a function, that does something and then you realize like. Oh, I can write any function. I like all right okay, so this is now going to mean that each mini batch is going to contain a list of characters, and so here's where we get to define all our different parameters and so to make it the same as previous sections of this notebook. I'm going to use the same batch size, the same number of characters then they're going to rename it to their PT t. Since we know what that means, the number of the size of the embedding and the size of our hidden state. Okay, remembering that size of our hidden state simply means going all the way back to the start, right and hidden simply means the size of the state that's created by each of those orange arrows. So it's the size of each of those circles, yeah, okay. So having done that, we can then create a little dictionary saying: what's our training, validation and test set in this case, I don't have a separate test set, so I just use the same thing and then I can say all right.

I want a language model data subclass with model data, I'm going to grab it from text files, and this is my path, and this is my field which I defined earlier, and these are my files - and these are my hyper parameters - min fracks not going to do Anything actually in this case, because there's not, I don't think, there's going to be any character that appears less than three times: that's probably redundant. Okay, so at the end of that it says, there's going to be 963 batches to go through, and so, if you think about it, that should be equal to the number of tokens divided by the batch size divided by B PTT because that's like the size of Each of those rectangles you'll find that in practice, it's not exactly that and the reason it's not exactly that is that the authors of torch text did something pretty smart, which I think we've briefly mentioned this before they said. Okay, we can't shuffle the data like with images we'd like to shuffle the order, so every time we see them in a different order, so there's a bit more random listed. We can't shuffle because we need to be contiguous, but what we could do is randomize. The length of you know basically, randomize be PTT a little bit each time, and so that's what pytorch? Does it's not always going to give us exactly 8 characters? Long 5 % of the time it'll actually cut it enough, and then it's going to add on a small little standard deviation. You know to make it slightly bigger or smaller than for white okay.

So it's going to be slightly different to eight on average. Yes, just to make sure is it going to be constant per Vinny watch yeah yeah, exactly that's right, so a mini batch you know has to kind of it needs to do a matrix multiplication and the mini batch size has to remain constant because we've got this Age, weight matrix that has to you know, has to line up in size with the size of the mini batch yeah. But the number you know the sequence length can can change their problem. Okay, so that's why we have 963, that's so the length of a data loader is how many mini batches in this case. It's so do it approximate okay number of tokens is how many unique things are in the vocabulary and remember after we run this line. Text now does not just contain in a description of what we want, but it also contains an extra attribute called vocab right, which contains stuff like a list of all of the unique items in the vocabulary and a reverse mapping from each item to its number. Okay. So that text object is now an important thing to keep out all right. So let's now try this. So we do. We started out by looking at the class, so the class is exactly the same as the class we've had before. The only key difference is to call in at hidden which calls sets out. So H is not a variable anymore. It's now an attribute itself. That H is a variable containing a bunch of zeros.

Now I've been shown that that size remains constant H time, but unfortunately, when I said that I lied to you and the way that I lied to you is that the very last mini batch will be shorter. Okay, the very last mini batch is actually going to have less than 60 well, it might be exactly the right size if it so happens that this data set is exactly divisible by B PTT times patch size, but it probably isn't so. The last batch will probably has a little bit less okay, and so that's why I do a little check here. That says, let's check that the batch size inside self, dot, H, right and so self dot H is going to be the height. Sorry, the height is going to be the number of activations and the width is going to be the mini batch size. Okay, check that that's equal to the actual sequence length. Sorry, the actual batch size length that we've received okay and, if they're not the same, then set it back to 0's again, okay, so this is just a minor little wrinkle that basically, at the end of each epoch, it's going to do like a little mini mini Batch right, and so then, as soon as it starts the next epoch, it's going to see that they're not the same again and it'll reinitialize it to the correct full batch size. Okay, so that's why you know if you're wondering there's an inert hidden, not just in the constructor but also inside forward it's to handle this kind of end of each epoch, start of each epoch.

Difference: okay, not an important point by any means, but potentially confusing. When you see it, okay, so the last wrinkle, the last

6. 00:35:43

- **Dealing with PyTorch not accepting a “Rank 3 Tensor”, only Rank 2 or 4, ‘F.log_softmax()’**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Wrinkle is something which i think is something that slightly sucks about pytorch, and maybe somebody can be nice enough to try and fix it with a PR if anybody feels like it, which is that the loss functions such as softmax, I'm not happy receiving a rank. 3 tensor remember a rank. 3 tensor is just another way of saying dimension three right: okay, there's no particular reason they ought to not be happy receiving a rank. 3 tensor, you know, like somebody, could write some code to say, hey a wreck. Three tensor is, probably you know a sequence length by batch size by you know, results thing, and so you should just do it for each of the two initial

axis, but no one's done that, and so it expects it to be a rank. Two tensor, funnily enough. It can handle write to or rec for, they're, not right through yeah, so we've got so. We've got a rank. Two tensor containing you know for each time period. I can't remember which way around the the y axes are, but whatever for each time period for each batch, we've got our predictions. Okay and then we've got our our actuals for each time period. For each batch, we've got our predictions and we've got our actuals. Okay, and so we just want to check whether they're the same and so at an ideal world, our lost function, a loss function, would check.

You know, item 1, 1, then item 1, 2 and then item 1 3. But since that hasn't been written, we just have to flatten them both out okay, and we can literally just flatten them out, put rose, rose, and so that's why here I have to use dot view. Okay and so dot view says, the number of columns will be equal to the size of the vocab, because remember we're going to end up with a prediction. You know a probability for each letter and then the number of rows is. However, big is necessary which will be equal to batch size times B, PTT, okay, and then you may be wondering where I do that. That's so that's where the predictions you may be wondering where I do that for the target, and the answer is torch text knows that the target needs to look like that. So torch text has already done that for us. Okay, so torch text automatically changes the target to be flattened out, as you might actually remember. If you go back to lesson 4, when we actually looked at a mini batch, that's bad out of torch text. We did. We noticed actually that it was flattened - and I said, we'll learn about why later and so later is now all right. Okay, so they're the three wrinkles get rid of the history ooh. I guess for wrinkles recreate the hidden state if the batch size changes flatten out and then you just torch text to create mini batches that line up nicely. So once we do those things we can then create our model, create our optimizer with that models, parameters and fit it.

One thing to be careful of here is that softmax now, as of pytorch 0.3, requires that we pass in a number here saying which access do we want to do the softmax over? So at this point this is a three-dimensional tensor right, and so we want to do the softmax over the final axis right. So when i say which axis do we do the softmax over? Remember we divide by there we go e to the X. I divided by the sum of e to the X I so it's saying, which axis do we sum over so which access we want to sum to one, and so in this case clearly we want to do it over the last axis, because the last axis is The one that contains the probability per letter of the alphabet - and we want all of those probabilities to sum to one okay, so therefore, to run this notebook you're going to need pytorch 0.3, which just came out this week. Ok! So if you're doing this on the milk you're, fine, I'm sure you've got at least a 0.3 or later ok, where else the students here, if you just go Conda and update it, will automatically update you to 0.3. The really great news is that 0.3, although it does not yet officially support windows, it does in practice. I successfully installed 0.3 from Condor yesterday by typing. Condor install torch, pytorch in Windows are then attempted to use the entirety of lesson 1 and every single part worked. So I actually ran it on this very laptop. So, for those who are interested in doing deep, burning on their laptop can definitely recommend the new surface book.

The new surface book 15-inch has a gtx, 1066 gig GPU in it, and i was getting and it was running about 3 times slower than my 1080 TI, which i think means it's about the same speed as an AW sp2, the instance and, as you can see, It's also a nice convertible tablet that you can write on and it's thin and light, and so it's like I've never seen a such a good, deep learning boss. Also, I successfully installed Linux on eart and all of the faster I staff worked on Linux as well. So really good option if you're interested in a laptop that can run deep learning, stuff, . Alright. So that's that's going to be aware of with this team equals minus 1. So then we can

go ahead and construct this and we can call fit and yeah we're basically going to get pretty similar results to what we got the ball. Alright. So then we can go a bit further without RNN by just kind of unpacking it a bit more, and so this is now again exactly the same thing gives exactly the same answers, but I have removed the cold air in it. So I've got rid of this self to iron in okay, and so this is just something I won't spend time on it, but you can check it out so instead, I've now to find iron in as R and n cell and I've copied and pasted the code. Above don't run it, this is just for your reference from pytorch.

This is this: is the color, the definition of eridan, so in pytorch, and I want you to see that you can now read pytorch source code and understand it not only that you'll recognize it as being something we've done before it's a matrix modification of The weights by the inputs plus biases, so f, dot, linear, simply does a matrix product, followed by an addition right and interestingly you'll see they do not concatenate the the input bit and the hidden bit. They sum them together, which is our first approach and I'm, as I said, you can do either neither one's right or wrong. But it's interesting to see this is the definition yeah

7. 00:44:05

- **Question on 'F.tanh()', tanh activation function,**
- **replacing the 'RNNCell' by 'GRUCell'**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Yes, you know can give some insight about what are they using that particular activation function: firm, yeah yeah. I think you might have briefly covered this last week, but very happy to do it again. If I did basically fan that's positive 1 and negative 1 then looks like that. So in other words, it's a sigmoid function, double the height minus one. Naturally they're they're equal. So it's it's! It's a nice function in that it's forcing it to be. You know no smaller than minus one, no bigger than plus one and since we're multiplying by this white matrix again and again and again and again, we might worry that our Lu, because it's unbounded might have more of a gradient explosion problem. That's basically the theory. Having said that, you can actually ask pytorch for an RNN cell, which uses a different non-linearity, so you can see by default it uses. Then you can ask for a value as well, but yeah most people seem to pretty much. Everybody still seems to use them as far as I can tell so. You can basically see here. This is all the same, except now I've got an errand in cell, which means now I need to put my for loop back alright, and you can see every time I call my my little linear function. I just obtained the result onto my list. Okay and at the end, the result is that all stacked up together, okay, so like just trying to show you how nothing inside pytorches is mysterious right, you should find you get. Basically the fact you I found.

I got exactly the same answer from this as the previous one. Okay, in practice, you would never write it like this, but what you may well find in practice is that somebody will come up with like a new kind of eridan cell or a different way of kind of keeping track of things over time or a different way Of doing regularization, and so inside posterize code, you will find that we we do this exactly this. Basically, we have this by hand, because we use some regularization approaches that are supported by pytorch, all right. So then another thing I'm not going to spend much time on, but I'll. Mention briefly, is that nobody really uses this hour an insult in practice and the reason we don't use that eridan so in practice is even though the fan is here, you do tend to find gradient. Explosions are still a problem, and so we have to use

pretty low learning rates to get these to train and pretty small values for B PTT to get them to Train. So

8. 00:47:15

- **Intro to GRU cell (RNNCell has gradient explosion problem - i.e. you need to use low learning rate and small BPTT)**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

What we do instead is we replace the eridan cell with something like this. This is called a GI: u cell and a GI? U so here it is, has a picture of it and there's the equations for it. So basically I'll show you both quickly but we'll talk about it much more in part, two we've got our input. Okay and our input normally goes straight in gets multiplied by a weight matrix to create our new activations. That's not what happens and then we've cost. We also, we add it to the existing activations. That's not what happens here in this case. Our input goes into this H, tilde temporary thing, and it doesn't just get added to our activations, their previous activations, that our previous activations get multiplied by this value. R and R stands for reset: it's a reset gate. Okay, and how do we calculate that this? This value goes between Norton one right in our reset gate? Well, the answer is: it's simply equal to a matrix product between some weight matrix and the concatenation of our previous hidden state and our new input. In other words, this is a little one, hidden layer, neural net and in particular it's a one, hidden layer, neural net, because we're then put it through the sigmoid function when you seasick, but one of the things I hate about mathematical notation is symbols are overloaded. A lot right, sometimes, when you see Sigma, that means standard deviation when you see it next to a parenthesis like this, it means the sigmoid function.

Okay, so in other words that okay, which looks like that okay. So this is like a little mini neural net, with no hidden layers or to think of it. Another way, it's like a little logistic regression. Okay - and this is - I mentioned this briefly, because it's going to come up a lot in part two, and so it's a good thing to like start learning about it's. This idea that, like in the very learning itself, you can have like little mini neural nets inside your neural nets right and so this little mini neural net is going to be used to decide how much of my hidden state am. I going to remember right, and so it might learn that Oh in this particular situation forget everything you know. For example, oh there's a full-stop, you know hey when you see a full-stop, you should throw away nearly all of your hidden state. That is probably something it would learn and that that's very easy for it to learn using this little mini neuron there, okay, and so that goes through to create my new hidden state, along with the input and then there's a second thing. That happens, which is there's this gate here called Z and what Z says is all right. You've got your some amount of your previous hidden state, plus your new input right and it's going to go through to create your new state and I'm going to.

Let you decide to what degree do you use this new input version of your hidden state and to what degree where you just leave the hidden state the same as before? So this thing here is called the update gate right, and so it's got two choices. It can make the first-years to throw away some hidden state when deciding how much to incorporate that versus my new input and how much to update my hidden state versus just leave it exactly the same. And the equation, hopefully, is going to look pretty similar to you, which is check this out here. Remember how I said you want to start to recognize some some common ways of looking at

things. Well, here I have a 1 something by a thing and a something without the 1 by a thing which remember is a linear interpolation right, so, in other words, the value of Z is going to decide to what degree do I have keep the previous hidden state And to what degree do I use the new hidden state right? So that's why they draw it here, as this kind of like it's not actually a switch, but like you can put it in any position. You can be like. Oh, it's here or it's here or it's here to decide how much that'll do ok, so so they're, basically the equations. It's a it's a little mini neural net, with its own weight matrix to decide how much to update little mini neural net with its own weight.

Matrix to decide how much to reset and then that's used to do an interpolation between the two hidden states, so that's called giu gated recurrent Network there's the definition from the pytorch source code. They have some slight optimizations here that, if you're interested in, we can talk about them on the forum, but it's exactly the same for we just saw, and so if you go and NDI you that it uses this same code, but it replaces the iron in so With this cell, okay and as a result, rather than having something where we needed, where we were getting a 1.54 we're nowgetting down to one point 400 and we can keep training even more get right down to one point: three: six: okay, so in practice a Giu or very nearly equivalently, we'll see in a moment in lsdm in practice what pretty much everybody always uses. So the art he and HT are ultimately scalars after they go through the sigmoid, but there are Clyde element-wise. Is that correct, mm-hmm yeah, although of course one for each mini batch? But yes, the scaler yeah, okay, great thanks and on the excellent colas

9. 00:53:40

- **Long Short Term Memory (LSTM), 'LayerOptimizer()', Cosine Annealing 'CosAnneal()'**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Blog Chris Ellis blog there's an understanding, LS TM networks post, which you can read all about this in much more detail if you're, interested and also the other one I was dealing from here - is a wild ml, also have a good blog post on this there's. Somebody wants to be helpful, feel free to put them in the lesson: wiki. Okay, so then putting it all together. I'm now going to replace my GI, you with Nellis TM, I'm not going to bother showing you the soul for this. It's very similar to GI! U but the LS TM has one more piece of state in it called the sell state, not just the hidden state. So if you do use an LS, TM you're now inside you're in it hidden, have to return a couple of matrices, they're exactly the same size as the hidden state, but you just have to return the tupple okay, the details, don't matter too much, but we can Talk about it during the week, if you're interested, you know, when you pass in you still pass in self dot, H still returns. A new value of H is to repackage it in the usual way. So this code is identical to the code before one thing. I've done, though, is I've had a drop out inside my air and in which you can do with the height watch, R and n function. So that's going to do drop out after a time step and I've doubled the size of my hidden layer.

Since I've now added point five drop out, and so my hope was that this would make it be able to learn more but be more resilient as it does so so then I wanted to show you how to take advantage of a little bit more fastai Magic without using the layer class, and so I'm going to show you how to use Cobra and specifically we're going to do s GDR without without using the learner class. Ok. So to do that, we create our model again, just a standard, pytorch

model. Ok and this time, rather than going remember the usual pipe torch approach is: opt equals up to M, naught atom and you pass in the parameters and a learning rate. I'm not going to do that. I'm going to use the fastai layer, optimizer class, which takes my opt-in class constructor from pytorch. It takes my model, it takes my learning rate and optionally takes weight. Decay, ok, and so this class is tiny. It doesn't do very much at all. The key reason it exists is to do differential learning rates and differential weight decay right, but the reason we need to use it is that all of the mechanics inside fastai assumes that you have one of these right. So if you want to use like callbacks or SPDR or whatever in code, where you're not using the learner class, then you need to use rather than saying you know: opt equals off to m dot, atom and here's my parameters, you instead say lay optimizer, okay, so That gives us a layer, optimizer object and, if you're interested basically behind the scenes, you can now grab a dot, opt property which actually gives you the optimizer.

But you don't have to worry about that itself, but that's basically what happens behind the scenes? The key thing we can now do is that we can now when we call fit, we can pass in that optimizer and we can also pass in some callbacks and specifically we're going to use the cosine annealing callback, okay, and so the cosine annealing callback requires a Layer optimizer object right, and so what this is going to do is it's going to do cosine annealing by changing the learning right inside this object? Okay, so the details aren't terribly important. We can talk about them on the forum. It's really the concept I wanted to get across here right, which is that now that we've done this, we can say all right create a cosine and kneeling callback which is going to update the learning rates in this layer. Optimizer, the length of an epoch is equal to this here right. How many mini batches are there in an epoch? Well, it's whatever the length of this data. Loader is okay, so because it's going to be it's going to be doing the cosine annealing, it needs to know how often to reset okay and then you can pass in the cycle more in the usual way, and then we can even save our model automatically. Like you remember how there was that cycle saved name parameter that we can pass to learn not fit. This is what it does behind the scenes behind the scenes. It sets an on cycle, end callback, and so here I have to find that callback as being something that saves my model.

Okay, so there's quite a lot of cool stuff that you can do with callbacks. Callbacks are basically things where you can define like at the start of training or at the start of an epoch or at the side of a batch or at the end of training or at the end of an epoch or at the end of a batch. Please call this club okay and so we've written some for you, including SGD R, which is the cosine annealing callback and then so how I recently wrote a new callback to implement the new approach to decoupled weight decay. We use callbacks to draw those little graphs of the loss over time, so there's lots of cool stuff. You can do with callbacks. So in this case, by passing in that callback, we're getting as JDR and that's able to get us down to one point three one here and then we can train a little bit more and eventually get down to one point: two five, and so we can now Test that out - and so if we passed in a few characters of text we get not surprisingly and a after four. Thus, let's do then 400, and now we have our own Nietzsche, so Nietzsche tends to start his sections with a number and a dot. So 2, 9 3, perhaps that every life, the values of blood have intercourse when it senses there is unscrupulous his very rights and still impulse love. Ok, so I mean it's slightly less clear than Nietzsche normally, but it gets the tone right.

Ok - and it's actually quite interesting like if to play around with training these character, based language models, to like run this at different levels of loss, to get a sense of like what does it look like, like you really notice that this is like 1.25 and, like That's slightly worse, like 1.3, this looks like total junk. You know, there's like punctuation in random places, and you

know nothing makes sense and, like you start to realize that the difference between you know, Nietzsche and random. Junk is not that far in kind of language model terms, and so, if you train this for a little bit longer, you'll suddenly find like. Oh it's it's it's making more and more sense. Okay. So if you are playing around with NLP stuff, particularly generative stuff like this and you're like there is also like kind of okay, but not great, don't be disheartened, because that means you're, actually very, very nearly there. You know the the difference between like something which is starting to create something which almost vaguely looks English, if you squint and something that's, actually a very good generation. It's it's not it's not far in most motion tests. Okay, great! So, let's take a

10. 01:01:47

- **Pause**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Five-Minute break we'll come back at 7:45 and we're going to go back to computer vision.

11. 01:01:57

- **Back to Computer Vision with CIFAR 10 and 'lesson7-cifar10.ipynb' notebook, Why study research on CIFAR 10 vs ImageNet vs MNIST ?**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Okay, so now become full circle back to vision, so now we're looking at less than seven so far ten that book, you might have heard of so far ten. It's a really well-known data set in academia and it's partly it's well known. It's actually pretty old by you know: computer vision standards well before image net was around. There was sci-fi 10. You might wonder why we're going to be looking at such an old data set, and actually I think, small data sets - are much more interesting than image net because, like most of the time, you're likely to be working with stuff, with a small number of thousands of Images, rather than one and a half million images, some of you will work at one and a half million images, but most of you won't right so learning how to use these kind of data sets, I think, is much more interesting, often also a lot of the Stuff we're looking at like in medical, imaging we're looking at like the specific area where there's a lung, nodule you're, probably looking at like 32 by 32 pixels at most as being the area where that lung nodule actually exists right and so sci-fi 10 is small. Both in terms of it doesn't have many images, and the images are very small, and so therefore, I think this is like it's been in. A lot of ways is much more challenging. Then something like image net and in some ways it's much more interesting right and also, most importantly, you can run stuff much more quickly on earth.

So it's much better to test out your algorithms with something you can run quickly and they're still challenging. And so I hear a lot of researchers complain about like how they can't afford to study all the different versions that their algorithm properly, because it's too expensive and they're doing that on imagenet. So, like it's literally a week of you know expensive CP GPU work for every study they do and like. I don't understand why you would do that kind of study on imagenet doesn't make sense yeah, and so this has been a particularly you know.

There's been a particular a lot of kind of debate about this this week, because I'm really interesting researcher named Ali raha me nips this week gave a talk. A really great talk about kind of the need for rigor in experiments in deep learning, and you know he felt like there's a lack of rigor and I've talked to him about it quite a bit since that time and I'm not sure we yet quite understand each Other as to where we're coming from, but but we have very similar kinds of concerns, which is basically people aren't doing carefully tuned carefully thought about experiments, but instead they kind of throw lots of GPUs or lots of data and consider that a day. And so this idea of like saying like well, you know it's my data statement. It's my algorithm meant to be good.

That's moral imagers at small data sets well, if so, let's study on sci-fi 10 revin, studying it on imagenet and then do more studies of different versions of the algorithm and learning different bits on and off, understand which parts are actually important and so forth. People also complain a lot about amnesty, which we've talked about looked at before and I would say the same thing about em this right, which is like, if you're, actually trying to understand rich parts of your algorithm, make a difference and why using m. Mr? That kind of study is a very good idea and all these people who complain about em NIST, I think they're, just showing off they're saying like oh, I work at Google and I have you know a part of TP use and I have $ 100,000 a week Of time just being on it no worries, but I don't know, I think, that's all it is. You know it's just signalling, rather than actually academically rigorous, okay. So I'm so far ten you can download from here, and this person is very kindly made it available in image form. If you, google, for size 510 you'll find us a much less convenient form. So please use this one. It's already in the exact form you need once you download it, you can use it in the usual way. So here's a list of the classes that are there now you'll see here. I've created this thing called that's normally.

When we've been using pre trained models, we have been seeing transforms from model and that's actually created the necessary transforms to convert our data set into a normalized data set based on the means and standard deviations of each channel in the original model. That was trained in our case, though this time we got a trainer model from scratch, so we have no such thing, so we actually need to tell it the mean and standard deviation of our data to normalize it. Okay - and so in this case I haven't included the code here to do it. You should try and try this yourself to confirm that you can do this and understand where it comes from, but this is just the mean channel and the standard deviation per channel of all of the images. Alright, so we're going to try and create a model from scratch, and so the first thing we need is some transformations, so for so far 10 people generally do data augmentation of simply flipping randomly horizontally. So here's how we can create a specific list of augmentations to use and then they also tend to add a little bit of padding black padding around the edge and then randomly pick a 32 by 32 spot from within that pattern image. So if you add the pad parameter to any of the fast, a a transform creators, it'll it'll do that for you, okay, and so in this case, I'm just going to add 4 pixels around each size, and so now that I've got my transforms. I can go ahead and create my image classifier data from paths in the usual way.

Okay, I'm going to use a batch size of 256 because these are pretty small. So it's going to, let me do a little bit more at a time. So here's what the data looks like so, for example, here's a boat and just to show you how tough this is. What's that? Okay, it is it's a not chicken prog-rock. So I guess it's this big thing. Whatever the thing is called there's your frog, okay, so so these are the kinds of things that we want to look at. So I'm going to

12. 01:08:54

- **Looking at a Fully Connected Model, based on a notebook from student 'Kerem Turgutlu', then a CNN model (with Excel demo)**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Start out so our student, Karen, we saw one of his posts earlier in this course. He he made this really cool log book which shows how different optimizers works. There we go. So Karen made this really cool notebook. I think it was maybe last week in which he showed how to create various different optimizers from scratch, so this is kind of like the Excel thing I had, but this is the Python version of momentum and Adam and Nesterov and Atta grad all written from scratch. It is very cool one of the nice things he did was. He showed a tiny little general-purpose, fully connected Network generator, so we're going to start with his so so he called that simple net. So are we so here's a simple class which has a list of fully connected layers? Okay, whenever you create a list of layers in pytorch, you have to wrap it in an end module list just to tailpipe torch, to like register these as attributes. And so then we just go ahead and flatten the data that comes in because it's fully connected layers and then go through each layer and call that linear layer do the value to it and at the end, do a soft mess. Okay, so there's a really simple approach, and so we can now take that model. And now I'm going to show you how to step up one level of the API higher rather than calling the fit function.

We're going to create a learn object, but we're going to create a learn, object from a custom model. And so we can do that by say: we want a convolutional learner, we want to create it from a model and from some data and the model is this one. So this is just a general height watch model and this is a model data object of the usual kind and that will return a loaner. So this is a bit easier than what we just saw with the RNN. We don't have to fiddle around with lair, optimizers and cosine and kneeling and callbacks, and whatever this is now a loaner that we can do all the usual stuff with that we can do it with any model that we created. Okay. So if we just go, learn that'll go ahead and print it out; okay, so you can see. We've got three thousand and seventy two features coming in, because you've got 32 by 32 pixels by three channels: okay, and then, we've got 40 features coming out of the first layer, that's going to go into the second layer. Ten features coming out because we've got the ten so far ten categories. Okay, you can call dot summary to see that a little bit more detail, we can do LR find we can plot that and we can then go fetch and we can use cycle length and so forth. Okay, so with a simple how many hidden layers do we have one hidden layer right, one getting layer, one output, layer, one hidden layer model with, and here we can see the number of parameters we have is that over 120,000 okay, we get a 47 percent accuracy.

So not great all right, so let's kind of try and improve it right, and so the goal here is we're going to try and eventually replicate the basic kind of architecture of a resonator okay. So that's where we're going to try and get to hear it specially built up to a resonant, so the first step is to replace our fully connected model with a convolutional model. Okay, so, to remind you, so to remind you, a fully connected layer is simply doing a dot product right. So if we had like all of these data points and all of these weights right, then we basically do a sum product of all of those together right. In other words, it's a matrix multiply right. Then that's a fully connected layer, okay, and so we need the white matrix is going to take, contain an item for every every element of the input for every element of the output. Okay. So that's why we have here a pretty big weight matrix and so that's why we had, despite the fact that we have such a crappy accuracy. We have a lot of parameters because in this very first layer we've got 3072 coming in and for T coming out. So that gives us three thousand times forty parameters,

and so we end up not using them very efficiently, because we're basically saying every single pixel in the input has a different weight. And, of course, what we really want to do is kind of find groups of three by three pixels that have particular patterns to them.

Okay, and remember, we call that a convolution okay, so a convolution looks like so. We have like little 3x3 section of our image and a corresponding 3x3 set of filters right or our filter with a three by three kernel, and we just do a sum product of just that three by three by that three by three okay and then we do That for every single part of our image right, and so when we do that across the whole image, that's called a convolution and remember in this case we actually had multiple filters right, so the result of that convolution actually had multiple. It was a tensor with an additional third dimension to it effectively. So let's take exactly the same code that we had before, but we're going to replace n n dot linear with NN com2 D. Okay. Now what I want to do in this case, though, is each time I have a layer I want to the next layer, smaller and so the way I did that in my Excel example, was I used max Pauling right, so max Pauling took every 2x2 section and Replaced it with this maximum value right nowadays, we don't use that kind of max bowling much at all. Instead, nowadays, what we tend to do is do what's called a stride to convolution. Let's drag to convolution, rather than saying, let's go through every single 3x3. It says. Let's go through every second 3x3 so, rather than moving this three by three one to the right, we move it two to the right and then when we get to the end of the row rather than moving one row down, we move two rows down.

Okay, so that's called a stride to convolution, and so it's tried to convolution has the same kind of effect as a max pooling, which is you end up having the resolution in each dimension, so we can ask for that by saying stroud equals to okay. We can say we wanted to be three by three by saying kernel, size and then the first term parameters are exactly the same as nn, but linear they're, the number of features coming in and the number of features coming out. Okay, so we create a multiple list of those layers and then at the very end of that. So in this case I'm going to say, okay, I've got three channels coming in the first one layer will come out with 20, then a at 40 and then 80. So if we look at the summary we're going to start with a 32 by 32, we're going to spit out of 15 by 15 and then a 7 by 7 and then a 3 by 3 right. And so what do we do now to get that down? To a prediction of one of 10 classes, what we do is we do something called adaptive max pooling, and this is what is pretty standard now for state-of-the-art algorithms. Is that the very last layer we do a max pool, but rather than doing like a 2 go to next Paul, we say like it doesn't. Have you to bow to could have been 3x3, which is like replace every three by three pixels with its maximum could have been four by four adaptive backs. Paul is where you say: I'm not going to tell you how big an area to pull, but instead I'm going to tell you how big a resolution to create right.

So, if I said, for example, I think my input here is like 28 by 28 right. If I said, do a 14 by 14 adaptive max Paul, that would be the same as a 2 by 2 max Paul, because in other that's saying, please create a 14 by 14 output. If I said, do a 2 by 2 adaptive max Paul right, then that would be the same as saying do a 14 by 14 max Paul, and so what we pretty much always do in modern cnn's. Is we make our per northmet layer a 1 by 1 adaptive Max Paul so in other words, find the single largest cell and use that as our new activation right, and so once we've got that we've now got a 1 by 1 tensor right, we're actually 1 By 1 by number of features tensor, so we can then, on top of that, go view X, dot view X, dot, size, comma, minus 1, and actually there are no other dimensions to this. Basically right. So this is going to return a matrix of mini-batch by number of features, and so then we can feed that into a linear layer with, however many classes, we need right, so you can see here. The last thing I pass in is

how many classes am I trying to predict and that's what's going to be used to create that last layer, so it goes through every convolutional layer? Does a convolution does arel you? Does an adaptive max pool this dot view just gets rid of those trailing unit, backsies one comma, one axis, which is not necessary. That allows us to fit that into our final linear layer that spits out something of size C, which here is ten, so you can now see how it works.

It goes 32 to 15 to 7 by 7 to 3 by 3. The adaptive next pull makes it 80 by 1 by 1 right and then our dot view makes it just a mini batch size by 80 and then finally, a linear layer which makes it from 80 to 10, which is what we wanted. Okay, so that's our like. Most basic, you would call this a fully convolutional network, so a fully convolutional network is something where every layer is convolutional, except for the very last. So again, we can now go Li dot, find and now, in this case, when I did ll find it went through the entire data set and we're still getting better, and so, in other words, even a the default final learning rate rises, 10, and even at that Point it was still like pretty much getting better, so you can always override the final learning rate by saying n del R equals that and that will get it just to get it to try morphine's, okay, and so here is the learning rate finder, and so I Picked 10 to the minus 1 trained that for a while and that's looking pretty good. So I try to put the cycle length of 1 and it's starting to flatten out at about 60 % right. So you can see here the number of elements. The number of parameters I have here are 500 7000 28,000 about 30,000 right. So I have about a quarter of the number of routers that my accuracy has gone up from 47 % to 60 % right and the time per epoch.

Here is under 30 seconds, and here also so the time period box about the same, and that's not surprising because when you use small, simple architectures most of the time is the memory transfer. The actual time during the compute is is trivial. Okay, so I'm going to refactor

13. 01:21:54

- **Refactored the model with new class 'ConvLayer()' and 'padding'**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

This slightly because I want to try and put less stuff inside my forward and so calling RAL you every time you know it doesn't seem ideal. So I'm going to create a new class called conf lair. Okay and the conf lair class is going to contain a convolution with a kernel size of three and a stride of two one thing I'm going to do now is I'm going to add padding. Did you notice here? The first layer went from 32 by 32 to 15 by 15, not 16 by 16, and the reason for that is that, at the very edge of your convolution right here see how this first convolution like there isn't a convolution where the middle is. The top left. Point right because there's like nothing outside it, where else, if we had put a row of zeros at the top and a row of zeros at the edge of each column, we now could go all the way to the edge. Alright, so pad equals 1 adds that little layer of zeros around the edge for us, ok, and so this way we're going to make sure that we go 32 by 32 to 16 by 16 to 8 by 8. It doesn't matter too much when you've got these bigger layers, but by the time you get down to like say, 4 by 4. You really don't want to throw away a whole piece right, so padding becomes important so by refactoring it to put this with its defaults. Here and then in the forward, I put the value in here as well. It makes by confident you know a little bit smaller and you know more to the point. It's going to be easier for me to make sure that everything is correct in the future.

By always using this common player class, ok, so now you know not only how to create your own neural network model, but how to create your own neural network layer. So here now I can use conf layer right - and this is such a cool thing about pytorch is a layer definition and a neural network definition are literally identical. Okay, they both have a constructor and a forward, and so anytime you've got the lair. You can use it as a neural net anytime. You have a neural net, you can use it as a lair okay. So this is now the exact same thing as we had before. One difference is. I now have padding okay and another thing just to show you. You can do things differently back here, my max pool I did as as an object likely. I use the class n n dot adaptive max pool, and I stuck it in this attribute and then I called it, but this actually doesn't have any state. There's no weights inside max pooling, so I can actually do it with a little bit less code by calling it as a function right. So everything that you can do as a class. You can also do as a function. It's inside this capital F, which is n n dot functional okay. So this should be a tiny bit better, because this time I've got the padding. I didn't trade it for as long to actually check. So, let's skip over that all right. So one issue here is that, in the end this is having I, when I tried to add more layers, I had travel training it. Okay and the reason I was having trouble training it is it was you know.

If I used larger learning rates, it would go off to NI N and if I use smaller learning rates that are kind of takes forever and doesn't really have a chance to explore properly. So it wasn't resilient. So to make my

14. 01:25:40

- **Using Batch Normalization (BatchNorm) to make the model more resilient, 'BnLayer()' and 'ConvBnNet()'**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Model more resilient, I'm going to use something called batch normalization which literally everybody calls bachelor and bachelorettes a couple of years old now and it's been pretty transformative since it came along because it suddenly makes it really easy to train deeper networks. Alright, so the network I'm going to create is going to have more layers right. I've got one two three four five convolutional layers plus a fully connected layer right so like back in the old days. That would be considered a pretty deep network and we considered pretty hard to train nowadays. Super simple thanks to vaginal now to use batch norm. You can just write in end on that to learn about it, we're going to write it from scratch. Okay, so the basic idea of batch norm is that we've got some vector of activations anytime I draw a vector of activations. Obviously I mean you can repeat it for the mini batch. So I pretend it's a mini batch with one. So we've got some veteran activations and it's coming into some layer right, so so probably some convolutional matrix multiplication and then something comes out. The other side. So imagine this. This is just a matrix multiply, which was like I don't know, say it was a identity matrix right then, every time I'd multiply it by that across lots and lots of layers.

My activations are not getting bigger, they're, not getting smaller they're, not changing at all. Okay! That's all fine right, but imagine if it was actually like 2 2, 2 right, and so if every one of my weight, matrices or filters was like that, then my activations are doubling each time right and so suddenly I've got as exponential growth and that in deep Models, that's going to be a disaster right because my gradients are exploding at an exponential rate, and so the challenge you have is that it's, it's very unlikely. Unless you try carefully to deal with it, that

your matrices, your weight, matrices on average, are not going to cause your activations to keep getting smaller and smaller, or keep getting bigger and bigger right. You have to kind of carefully control things to make sure that they stay. You know at a reasonable size. You want to, you, know, keep them at a reasonable scale. So we start things off with zero, mean standard deviation, one by normalizing. The inputs really like to do is to normalize every layer, not just the inputs, all right and so, okay, fine, let's do that right. So here I've created a BN layer which is exactly like my Kampf layer. It's got my common 2d with my stride. My padding right, I do my condom, I value right and then I calculate the mean of each channel or of each filter and the standard deviation of each channel or each filter. And then I subtract the means and divide by the standard deviations right.

So now I don't actually need to normalize my input at all, because it's actually going to do it automatically right, it's normalizing it per channel or and for later layers its normalizing it per filter. So it turns out that's not enough right, because SGD is bloody-minded right, and so, if sgt decided that it what's the weight matrix to be. You know like so where that matrix is something which is going to. You know, increase the values overall repeatedly then trying to divide it by the subtract, domains and divide by the standard deviations just means the next mini-batch. It's going to try and do it again and they were try and do it again, it'll try and do it again. So it turns out that this actually doesn't help like it literally does nothing, because SGD is just going to go ahead and undo it. The next mini batch. So what we do is we create a new multiplier for each channel and a new added value for each Channel literally just and we just start them out as the addition and addition is just a bunch of zeros so for the first layer, three zeros and the Multiplier for the first layer is just three ones. Okay, so number of filters for the first layer is just three, and so we then, like basically undo exactly what we just did or potentially we undo them right. So by saying this is an addenda parameter that tells pytorch you're allowed to learn these as weights right so initially it says: okay, so check the means divided by the standard deviations multiplied by one add on zero.

Okay, that's fine! Nothing much happened there, but what it turns out is that now, rather than like, if it wants to kind of scale the layer up, it doesn't have to scale up every single value in the matrix. It can just scale up this single trio of numbers self dot M. If it wants to shift it all Apple down, a bit doesn't have to shift the entire weight matrix. They can just shift this trio of numbers self dot a so I will say this, I'm at this talk I mentioned at nips Alley. Rahim ease talk about rigor. He actually pointed to this paper this batch norm paper as being a particularly useful, particularly interesting paper, where a lot of people, don't necessarily we quite quite know why it was right, and so, if you're thinking like okay subtracting out the means and then adding some learned Weights of exactly the same rank and size sounds like a weird thing to do. There are a lot of people that feel the same way right. So at the moment, I think the best is I can say like intuitively is what's going on here. Is that we're normalizing the data and then we're saying you can then shift it and scale it using far fewer parameters than would have been necessary. If I was asking you to actually shift and scale the entire set of convolutional filters right, that's the kind of basic intuition, more importantly, in practice, what this does is it adds.

Is it basically allows us to increase our learning rates and it increases the resilience of training and allows us to add more layers? So once I added a PN layer rather than a common flower, I found I was able to add more layers to my model and it still trained effectively. Generally, are we worried about anything that maybe we are divided by something very small or anything like that? Once we do this, probably I think in the pie chart version, it would probably be divided by itself, dudes plus Epsilon or something yeah. This worked fine for me, but yeah.

That is definitely something to think about. If you were trying to make this more reliable, I mentioned poplar, so the self dot m and self dot a getting it's getting updated through back propagation as well yeah. So, by putting like saying it's an N n dot parameter, that's how we flag to pytorch! To learn it through that probe exactly right. The other interesting thing it turns out the batch norm does. Is it regularizes? In other words, you can often decrease or remove, drop out or decrease or remove weight? Okay, when you use batch normal and the reason why is, if you think about it, each mini batch is going to have a different mean and a different standard deviation to the previous mini batch. So these things keep changing and because they keep changing it's kind of changing the meaning of the filters in this subtle way, and so it's adding a regularization effect, because it's noise that when you add noise of any kind, it regularizes your model, all right.

I'm actually cheating a little bit here in the real version of batch norm. You don't just use this batches mean and standard deviation, but instead you take an exponentially weighted moving average standard deviation and - and so if you wanted to exercise to try a during the week, that would be a good thing to try. But I will point out something very important here, which is, if self-training, when we are doing our training loop. This will be true when it's being applied to the training set and it will be false when it's being applied to the validation set, and this is really important, because, when you're going through the validation set, you do not want to be changing the meaning of the Model, okay, so this is this really important idea is that there are some types of layer that are actually sensitive to what the mode of the of the network is, whether it's in training mode or as plight, which calls it evaluation mode, or we might say a Test mode right and actually we actually had a bug, a

15. 01:36:02

- **Previous bug in 'Mini net' in 'lesson5-movielens.ipynb', and many questions on BatchNorm, Lesson 7 Cifar10, AI/DL researchers vs practioners, 'Yann Lecun' & 'Ali Rahimi talk at NIPS 2017' rigor/rigueur/theory/experiment.**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Couple of weeks ago, when we did our mini net for movie lens the collaborative filtering, we actually had F dot dropout in our forward pass without protecting it with a F self training, F, dot dropout, as a result of which we were actually doing dropout in the Validation piece, as well as the training piece, which obviously isn't what you want: okay, so I've actually gone back and fixed this by changing it to using n n dot dropout and n n dot. Dropout has already been written for us to check whether it's being used in training mode or not that or alternatively, I could have added a if self dot training before I use the dropout yeah. Okay. So it's important to think about that. You know any and the main, the main true or pretty much the only two built-in two pytorch, where this happens is dropout and, and so interestingly, this is also a key difference in fastai, which no other library does is that these means and standard deviations Get updated in training mode in every other library as soon as you basically say, I'm training, regardless even of whether that layer is set to trainable or not, and it turns out that with a pre trained network. That's a terrible idea! If you have a pre trained network for specific values of those means and standard deviations in batch norm, if you change them, it changes the meaning of those pre trained layers right and so in fastai, always by default. It won't touch those means and standard deviations.

If your layer is frozen, okay as soon as you, I'm freezing it'll start updating them unless you've set won't, be and freeze. True, if you set learned up being freeze true, it says: never touch these met, means and standard deviations, and you know I've found in practice that that often seems to work a lot better for pre-trained models, particularly if you're working with data that's quite similar to what The pre trained model was trained with you know. As you look like, I did a lot of work. Did you say sorry, like quite a lot of code here well you're doing more work than you would normally do essentially you're calculating all these aggregates as you go through each each each layer? Yes, wouldn't this mean you're training like your epoch time like now, this is like super fast like if you think about what a cone has to do. A cones has to go through every 3x3. You know with a stride and do this multiplication and then addition like that is a lot more work than simply calculating the per channel mean. So this is so and that's a little bit of time, but it's it's less time intensive than the convolution. Would it be like right after, like decomposition, yeah, we'll talk about that in a moment? So at the moment we have it after the rally and in the original batch norm paper. I will that's where they put it. So this this idea of something called an ablation study and an ablation study is something where you basically try kind of turning on and off different pieces of your model to see like which bits make which impacts and one of the things that wasn't done in the Original batch norm paper was any kind of really effective, ablation study and one of the things therefore, that was missing was this question, which you just asked, which is like: where do you put the vaginal before the early year after the earlier whatever? And so since that time, you know that oversight has caused a lot of problems because it turned out the original paper didn't actually put it in the best spot, and so then other people since then have now figured that out and they're like every time.

I show people code where it's actually in the spot, that turns out to be better people, always say your bedrooms in the wrong spot, and I have to go back and say no, I know that's what the paper said. What they're doing now. That's what I thought, and so it's kind of causes confusion. So there's there's been a lot of question about that. So a little bit of a higher-level question, so we started out with cipher data. Yes, it's the basic reasoning that you use a smaller data set to quickly train a new model, and then you take it the same model and you're using much much much bigger data set to get a higher accuracy level. Is that the basic? Maybe so, if you want to you know, if you had a large data set or if you were like interested in the question of like how good is this technique on a large data set, then, yes, what you just said would be what I would do. I would do lots of testing on a small data set, which I had already discovered, had the same kinds of properties as my larger data set, and therefore my conclusions would likely carry forward and then our test them at the end. Having said that, personally, I matched be more interested in actually studying small datasets for their own sake, because I find most people I speak to in the real world. Don't have a million images they have.

You know somewhere between about two thousand, and twenty thousand images seems to be much more common, so I'm very you know very interested in having fewer rows, because I think it's more valuable in factors I'm also pretty interested in small images, not just for the rest. You mentioned, which is it allows me to test things out more quickly, but also, as I mentioned before. Often a small part of an image actually turns out to be what you're interested in that's. Certainly true in in medicine, I have two questions. The first is on what you mentioned in terms of small datasets, particular middle medical. Imaging you've, you've heard of, I guess, is it vicarious to start up in the specialization and one-shot learning, so your opinions on that, and then this second being this is related to. I guess Ali's talk at nips, so it was, I don't say its controversial, but like young laoco[illegible]n, there was like a really, I guess,

controversial thread attacking you in terms of what you're talking about as a baseline of theory, just not keeping up with practice, and so I mean I guess I was siding with the on where's all he actually, he tweeted at me quite a bit trying to defend like he wasn't attacking yawn at all, but in fact he was you know trying to support him, but I just kind of feel like A lot of theory, as as you go, is just sort of at it. They even it's hard to keep up whether then you know no archive from on Draco party to keep up.

But if the theory isn't keeping up, but industry is the one that's actually sitting in the standard, then doesn't that mean that you know people who are actual practitioners are the ones like young lacunae are publishing the theory that are keeping up to date or is like Academic research institutions are actually behind, so I don't have any comments on the vicarious papers because I haven't read them, I'm not aware of any of them have as actually showing you know better results than the papers, but I think they've come a long way in the Last twelve, so that might be wrong. Yeah yeah, I viewed the discussion between yarn lacunae and a lyric. Jimmy is very interesting because they're, both smart people who have interesting things to say. Unfortunately, a lot of people talk, Ally's talk as meaning something which he says it didn't mean and when I listened to his talk, I'm not sure he didn't actually made it at the time, but he clearly doesn't mean it now, which is he's. He's now said many times he didn't. He was not talking about theory. He was not saying, we need more theory at all. Actually, he thinks we need more experiments, and so specifically he's he's also now saying he wished. He hadn't used the word rigour which I also wish, because rigour is it's kind of meaningless and everybody can kind of say when he says rigor. He means the specific thing I study you know.

So a lots of people have kind of taken his talk as being like. Oh yes, this proves that nobody else should work in neural networks unless they are experts at the one thing, I'm an expert in so yeah. So I'm going to catch up with him and talk about more about this in January and hopefully we'll pick up some more stuff out together. But basically what would we can clearly agree on, and I think you and also agrees on is careful. Experiments are important. Just doing things on massive amounts of data using massive amounts of TP use or GP users, not interesting of itself, and we should instead try to design experiments that give us the maximum amount of insight into what's going on so Jeremy. Is it a good statement to say something like so drop out and bash norm are very different? Things drop out is the realization technique. Bash norm has maybe some realization effect, but it's actually just about convergence of the optimization method. Yeah, yeah and - and I would further say like I can't see any reason not to use pattern, or there are versions of batch norm that, in certain situations, turned out not to work so well. That people have figured out ways around that for nearly every one of those situations now, so I would always seek to find a way to use batch norm. It may be a little harder in our own ends, at least, but even there there are ways of doing batch norm in our attenders as well.

So you know, try, try and always use batch norm on every layer if you can and the question that somebody asked is, does it mean I have to? I can stop normalize in my data yeah yeah. It does, although do it anyway, because it's not at all hard to do it, and at least that way the people using your data - I don't know they kind of know how you've normalized it, and particularly with these issues around a lot of libraries. In my opinion, at least warm not my opinion, my experiments don't deal with batch norm correctly for pre-trained models. Just remember that when somebody starts retraining, those averages and stuff are going to change for your data set, and so if your new data set has very different input averages, it could really cause a lot of problems so so yeah. I went through a period where I actually stopped normalizing my data and you know things kind of worked, but it's probably not worth it.

Okay, so so the rest of this is identical right. All I've done is I've changed, conf layer to BN layer, but I've done one more thing, which is I'm kind of trying to get closer and closer to modern approaches which I've added a single convolutional layer at the start, with a bigger kernel, size and a stride Of one, why have I done that? So the basic idea is that I want my first layer to kind of have a richer input right so before my first layer had an input of just three, because there's just three channels right. But if I start with my image right and and I kind of take a bigger area - few different color - I kind of take a bigger area right and I do a convolution using that bigger area.

In this case, I'm doing five by five right. Then that kind of allows me to try and find more interesting, richer features in that 5x5 area, and so then I spin out a bigger output. This case, I spit out a filter size good about ten five by five filters, and so the idea is like pretty much. Every state of the art convolutional architecture now starts out with a single con flare, with like a five by five or seven by seven or sometimes even like 11, by 11 convolution, with like quite a few filters, you know something like you know. Thirty two filters coming out and it's just just a way of kind of trying to and like because I use the straight of one and the padding of kernel size minus one over two. It means that my outputs going to be exactly the same size as my input, but just got more filters now. This is just a good way of trying to create a richer starting point for my sequence of convolutional layers. Okay, so that's the basic theory of why I've added this single convolution, which I just do once at the start and then I just go through all my layers and then I do my adaptive max pooling and my final classifier okay. So it's a minor tweak, but it helps right and so you'll see. Now I kind of can go for a Moodle. I have 60 % and after a couple is 45 %. Now, after a couple, that's 57 % and after a few more I'm out for 68 % okay, so you can see it's. You know the the batch norm and you know tiny bit the conveyor at the start, it's helping now, what's more, you can see.

This is still increasing right, so that's looking pretty encouraging okay! So, given that

16. 01:50:51

- **'Deep BatchNorm'**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

This is looking pretty good. An obvious thing to try might be to see is to try increasing the depth of the model, and now I can't just add more of most dried to layers because remember how it half the size of the image each time, I'm basically down to two by two At the end right, so I can't add much more so what I did instead was. I said: okay, here's my original layers, so you must trade. Two layers for everyone also create a straight one layer. So astrayed one layer doesn't change the size, and so now I'm saying zip my stride, two layers and my stride, one layers together and so first of all do the straight too and then do the straight one. So this is now actually twice as deep okay. So this is so. This is now twice as deep, but I end up with the exact same. You know two by two that I had before, and so, if I try this, you know here after one two three four epochs is at sixty-five percent. After one two three epochs, I'm still at 65 %, it hasn't helped right, and so the reason it hasn't helped is I'm now too deep, even for batch norm, two handlers. Now my depth is now one two three four five times two is ten: eleven kampf 112. Okay, so 12 layers deep, it's possible to train a standard, confident 12 layers deep, but it starts to get difficult to do it properly right and it certainly doesn't seem to be really helping much if at all, so that's where I'm instead going to

17. 01:52:43

- **Replace the model with ResNet, class 'ResnetLayer()', using 'boosting'**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Replace this with a ResNet all right, so a rest net is our final stage. What a resin it does is, I'm going to replace our BN layer right, I'm going to inherit from BN layer and replace our forward with that and that's it. Everything else is going to be identical, except now I'm going to do like way. Lots of layers I'm going to make it four times deeper right and it's going to Train beautifully just because of that. So why does that help so much so this is called a ResNet block and, as you can see, I'm saying that's not what I meant to do. I'm saying my predictions equals my input plus some function. You know, in this case a convolution of my input. Alright, that's that's! That's what I've written here and so I'm now going to shuffle that around a little bit and I'm going to say I'm going to say f of X, equals y minus X. Ok, so there's the same thing shuffled around right! That's my prediction! Within the previous layer right, and so what this is then doing, is it's trying to fit a function to the difference between these two right, and so the difference is actually the residual. So if this is what I'm trying to calculate my actual Y value - and this is the thing that I've most recently calculated, then the difference between the two is basically the error in terms of what I've calculated so far, and so this is therefore saying that okay, Try to find a set of convolutional weights that attempts to fill in the the amount I was off by so.

In other words, if we let's clear this out, if we have some inputs coming in right and then we have this function, which is basically trying to predict the error, it's like how much are we off by right, and then we add that on so we basically Add on this additional, like prediction of how much will be wrong by and then we add on another prediction of how much were we wrong by that time and add on another prediction of how much we wrong by that time, then that each time we're kind of Zooming in getting closer and closer to our correct answer and each time we're saying like okay, we've got to a certain point, but we still got an error. You've still got a residual, so let's try and create a model that just predicts that residual and add that on to our previous model and then let's build another model that predicts the residual and add that on to our previous model. And if we keep doing that again and again, we should get closer and closer to our answer, and this is based on a theory called boosting which people that have done some machine learning will certainly come across right. And so basically, the trick here is that by specifying that, as being the thing that we're trying to calculate, then we kind of get boosting for free right. It's like because we couldn't just juggle that around to show that actually it's just calculating a model on the wrist Jill. So that's kind of amazing and you know it totally works.

As you can see here, I've now got my standard batch norm. Layer, okay, which is something which is going to reduce my size by two, because it's got the stride too, and then I've got a resident layers, dried one and another resident layer, astride one right and sorry. I think I said that was four of these is actually three of these, so this is now three times deeper. I zipped through all of those and so I've now got a function of a function of a function, so three layers per group and then my con at the start and my linear at the end. So this is now three times bigger than my original and if I fit it you can see, it's just keeps going up and up and up and up, I keep fitting it more he's going up and up and it's still going up when I kind of got Bored, okay, so the rest net has been a really important development and it's allowed us to create

these really deep networks right now. The full risk net does not quite look the way. I've described it here. The full res net doesn't just have one convolution right, but it actually has two convolutions right. So the way people normally draw resident blocks is they normally say. You've got some input coming in to the layer. It goes through one convolution to convolutions and then gets added back to the original input right. That's the full version of a ResNet block.

In my case, I've just done

18. 01:58:38

- **'Bottleneck' layer with 'BnLayer()', 'ResNet 2' with 'Resnet2()', Skipping Connections.**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

One convolution, okay and then you'll see also in every block right one of them. It's actually the first one. Does he it's actually the first one here is not a resident block, but a standard convolution with a stride of two right. This is called a bottleneck layer right and the idea is. This is not a ResNet block. So from time to time, we actually change the geometry right, we're doing this trade to in resident. We don't actually use just a standard, convolutional layer, there's actually a different form of bottleneck, block that I'm not going to picture in this course. I'm going to teach you in part, two okay, but, as you can see, even this somewhat simplified version of a resin. It still works pretty well, and so we can make it a little bit bigger all right, and so here I've just increased all of my sizes. I have still got my three and also I've had it drop out right. So at this point I'm gon na say this is other than the minor simplification of ResNet. You know a reasonable approximation of a good starting point for a modern architecture - okay and so now, I've added in my point to drop out. I've increased the size here and if I train this, you know I can treat it for a while. It's going pretty well, I can then add in gta. At the end, eventually, I get 85 %, and you know this is at a point now, where, like literally, I wrote this whole notebook in like three hours right.

We can like create this thing in three hours, and this is like an accuracy that in kind of 2012-2013, was considered pretty much data the art for say, pocket right, so this is actually no. This is actually pretty damn good to get. You know nowadays, the most recent results are like 97 %. You know there are there's plenty of room. We can still improve, but they're all based on these techniques like there isn't really anything. You know when we start looking in in part to it like how to get this right up to state of the art. You'll see it's basically better approaches to data augmentation, better approaches to regularization some tweaks on ResNet, but it's all basically the circuit. Okay, so so is the residual training on the residual method. Is that only looks like it's a generic thing that can be applied non image problems? Oh great question, yeah? Yes, it is, but it's like being ignored everywhere else in NLP, something called the transformer architecture recently appeared and you know was shown to be the state of the art for translation and it's got like a simple resonance structure. You know first time, I've ever seen it in NLP. I haven't really seen anybody else take advantage of it yeah this general approach. We call these skip connections, this idea of like skipping over a layer and kind of doing an identity. It's yeah: it's been appearing a lot in computer vision and nobody else much seems to be using it.

Even though there's nothing computer vision specific about it. So I think it's a big opportunity. Okay, so final stage, I

19. 02:02:01

- **'lesson7-CAM.ipynb' notebook, an intro to Part #2 using 'Dogs v Cats'.**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Want to show you is how to use an extra feature of pytorch, to do something cool and it's going to be a kind of a segue into part. Two. It's going to be our first little hint as to what else we can build on these neural nets, and so - and it's also going to take us all the way back to lesson 1, which is we're going to do. Dogs and cats. Ok, so going all the way back to dogs and cats we're going to create a resin at 34; ok, so these different ResNet, 3450 101 they're they're, basically just different numbers of different sized blocks. It's like how many of these kind of pieces do you have before it bottleneck block and then how many of these sets of super blocks? Do you have right? That's all these different numbers mean. So if you look at the torch vision source code, you can actually see the definition of these different resonates. You'll see they're all just different parameters: right, ok, so we're going to use rest at 34, and so we're going to do this a little bit more by hand okay. So if this is my architecture, this is just the name of a function then I can call it to get that model right and then true, look at the definition is, do I want the pre-trained? So, in other words, is it going to load in the pre-trained imagenet weights? Okay, so M now contains a model, and so I can take a look at it like so.

Okay - and so you can see here what's going on right - is that inside here I've got my initial two deconvolution, and here is that kernel size of seven by seven, okay and interestingly, in this case, it actually starts out with a seven by seven strobe. Okay, there's the padding that we talked about to make sure that we don't lose the edges all right. There's our batch naught. Okay, there's our Lu! You get the idea right, Kong and then so here you can now see there's a layer that contains a bunch of blocks all right. So here's a block which contains a cons fetch norm rally you con Bethnal. You can't see it printed, but after this is where it does the addition all right so there's like a whole ResNet block and then another resident block and then another ResNet block okay and then you can see. Also, sometimes you see one where there's a stripe right. So here's actually one of these bottleneck layers. Okay, so you can kind of see how this is. This is structure so, in our case sorry, I skipped over this a little bit, but the approach that we ended up using for real you was to put it before our before our Bashan on which see what they do here. We've got fetch norm railing. You cons that no, I'm really okay, okay, so you can see the order that they're using it here. Okay and you'll find like there's two different versions of ResNet. In fact, there's three different versions of ResNet floating around the one which actually turns out to be the best. It's called the pre-act ResNet, which has a different ordering again, but you can look it up.

It's basically a different order of where the value and where the batch norm yeah okay, so we're going to start with a standard resident 34 and normally what we do is we need to now turn this into something that can predict dogs versus class right. So currently, the final layer has a thousand features, because imagenet has a thousand features right, so we need to get rid of this. So when you use confer owner from pre-trained in fastai, it actually deletes this layer for you, and it also deletes this layer and something that, as far as I know, is unique to fastai.

Is we replace this see? This is an average pooling layer of size, seven by seven right. So this is the basically the adaptive pooling layer, but whoever wrote this didn't know about adaptive pooling, so they manually said. Oh, I know it's meant to be seven by seven, so in fastai we replaced this with adaptive pooling, but we actually do both adaptive average pooling and adaptive max pooling and we then concatenate them two together which it's it is something we invented, but at the Same time we invented it. Somebody wrote a paper about it. So it's you know, we don't get any credit, but I think we're the only library that provides it, and certainly only one that does it by default. We're going to for the purpose of this exercise, though, we're going to do a simple version where we delete the last two layers so we'll grab.

All the children of the model will delete the last two layers and then instead we're going to add a convolution which just has two outputs right. I'll show you why, in a moment right then we're going to do our average pooling and then we're going to do soft mess? Okay, so that's a model which is going to have you'll see that there is no. This one has a fully connected layer at the end, this one does not have a fully connected layer yet, but if you think about it, this convolutional layer is going to be two filters only right and it's going to be two by seven by seven, and so Once we then do the average pooling it's going to end up being just two numbers that it produces. So this is a different way of producing just two numbers. I'm not going to say it's better, it's going to say it's different, okay, but there's a reason. We do it I'll show you the reason. We can now train this model in the usual way right, so we can say, transform stock model, image, classifier data from paths, and then we can use that Kampf learner from model data we just learnt about. I'm now going to freeze every single layer, except for that one - and this is the fourth last layer, so we'll say, freeze to minus four right, and so this is just training the last layer. Okay, so we get 99.1 percent accuracy so that you know this approaches.

Working fine and here's what we can do, though we can now do something.

20. 02:08:55

- **Class Activation Maps (CAM) of 'Dogs v Cats'.**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Called fast comm last activated Maps fast activation is what we're going to do. Is we're going to try to look at this particular cat and we're going to use a technique called class activation Maps where we take our model and we ask you which parts of this image turned out to be important and when we do this, it's going to Feed out this is got the picture, it's going to create alright, and so, as you can see here, it's found the cat. So how did it do that? Well, the way it did that will kind of work backwards is to produce this matrix now you'll see in this matrix, there's some pretty big numbers around about here which correspond to our cat. So what is this matrix? This matrix is simply or to the value of this feature. Matrix times this py vector the py vector, is simply equal to the predictions which in this case said I am 100 %, confident it's a cat right, so this is just equal to the value of. If I just call the model passing in our cat, this is our cat. That's an X, then we got our predictions right. So it's just the value of our predictions, so py is just the value of our predictions. What about feet? What's that equal to feet is equal to the values in this layer right, in other words, the value that comes out of the final in facts come out of this ladder coming out of the final convolutional layer right. So it's actually the seven by seven by two, and so you can see here, let's see feet, the shape of

features is two filters by seven by seven right.

So the idea is, if we multiply that vector by that tensor right, then it's going to end up grabbing all of the first channel, because that's a 1 and none of the second channel, because that's a 0 and so therefore it's going to return the value of The last convolutional layer for the for the section which lines up with being a cat, but if you think about it, this the first section lines up with being a cat. The second section lines up with being a dog. So if we multiply that tensor by that tensor, we end up with this matrix and this matrix is which parts most like a cat or to put it another way in our model. The only thing that happened after the convolutional layer was an average pooling layer. So the average pooling layer talked that 7x7 grid and said average. How much each part is cat Lake that answered my final value. My final prediction was the average cattiness there's the whole thing right, and so because it had to be able to average out these things. To get the average cattiness, that means I could then just take this matrix and resize it to be the same size as my original cat and just overlay it on top. You get this heat map right, so the way you can use this technique at home is to basically calculate this matrix right on some. Like really you've got some really big picture.

You can calculate this matrix on a quick, small, little cognate and then zoom into the bit that has the highest value and then rerun it just on that part that so it's like. Oh, this is the area that seems to be the most like a hat or most like a dog that zoom in to that bit right. So I skipped over that pretty quickly because we ran out of time and so we'll be learning more about these kind of approaches in part two, and we can talk about it more on the forum and hopefully you get the idea. The one thing that totally skipped over was: how do we actually ask for that particular layer, okay and I'll? Let you read about this during the week, but basically there's a thing called a hook, so we said we called save features, which is this little class that we wrote that goes register forward hook and basically a forward hook is a special pytorch thing that every Time it calculates a layer, it runs this function, it's like a callback. Basically, it's like a callback that happens every time. It calculates a layer, and so in this case it just saved the value of the particular layer that I was interested in okay, and so that way I was able to go inside here and grab those features out. Look after I was done okay, so I call save features that gives me my pork and then later on. I can just grab the value that I saved okay, so I skipped over that pretty quickly. But if you look in the pipe or docks they have some more information and help about that.

Yes, Jeremy, can you

21. 02:14:27

- **Questions to Jeremy: “Your journey into Deep Learning” and “How to keep up with important research for practioners”,**
- **“If you intend to come to Part 2, you are expected to master all the techniques in Part 1”, Jeremy’s advice to master Part 1 and help new students in the incoming MOOC version to be released in January 2018.**

(autogenerated subtitles follow, may contain gibberish/bad format - please proofread to improve - remove this note once proofread)

Spend five minutes talking about your journey into deep learning yeah and finally, how can we keep up with important research that is important to practice, sure yeah, so it's gon na that's good! I think I'll close more on the latter bit, which is like what now okay. So for those

of you are interested, you should aim to come back for part two. If you're aiming to come back for part two, how many people would like to come back to part two? Okay, that's not bad! I think almost everybody. So if you want to come back to part two be aware of this by that time, you're expected to have mastered all of the techniques we've learnt in part one and there's plenty of time between now and then okay, even if you haven't done much or any Ml before, but it does assume that you're going to be working, you know at the same level of intensity for now until then that you have been with practicing right so practicing. So, generally speaking, the people who did well in part two last year had watched each of the videos about three times right and some of the people. Actually, I knew had actually discovered. They learnt some of them off by heart by mistake. So they're like watching the video, is again is helpful and make sure you get to the point that you can recreate the notebooks without watching the videos, all right, and so other men make more interesting, obviously try and recreate them notebooks using different data sets.

You know and definitely then just keep up with the forum and you'll see people keep on posting more stuff about recent papers and recent advances and over the next couple of months, you'll find increasingly less and less of it seems weird mysterious and more and more of It makes perfect sense, and so it's a bit of a case with just thing. Staying tenacious, you know, there's always going to be stuff that you don't understand yet and but you'll be surprised. If you go back to lesson one and two now you'll be like. Oh, that's all trivial right, so you know that's kind of hopefully a bit of your learning journey and yeah. I think the main thing I've noticed is that people who succeed are the ones who just keep keep working at it. You know so not coming back here. Every Monday you're not going to have that forcing function. I've noticed the forum suddenly gets busy at 5:00 p.m. on a Monday. You know it's like, Oh course, is about to start, and suddenly these questions start coming in. So now that you don't have that forcing function, you know try and use some other technique to. You know give yourself that little kick. Maybe you can tell your partner at home. You know I'm going to try and produce something every Saturday for the next four weeks or I'm going to try and finish reading this paper or something you know anyway.

So I hope to see you all back in March and even I, regardless whether I do or don't it's been a really great pleasure to get to know you all and I hope to keep seeing on the forum thanks very much. [Applause,]