Week02 TRANSCRIPCIÓN Clase Synchronic 02 Advanced Machine Learning Methods

Jan 17th, 2025

OK, let's switch to English now.

Good afternoon, everyone.

Nice to see you today.

So the idea today and the idea for all the sessions is going to be-- there will be some sessions where when I have something like a more structural thing, because I want to show something like a paper or something like that.

But most of the sessions are going to be like these kind of office hours.

And as I mentioned last time, so an opportunity to exchange ideas and questions.

So I would like to start by asking if there's any question from last week.

Or any comment, anything you would like to say? - I have a question.

I have a question. - Yes, please. - Yeah, I mean, it might sound a bit obvious, but I would like to have more-- that concept more clear.

So I saw in the video that when we have a problem with many classes, you create the final amount of neurons are related with the classes, right? - OK. - So my question is, what's the difference between creating a neuron for each class and just between-- and in the other hand, just using one neuron as the output and trying to minimize the loss function that is computed by the softmax function?

Because once you use the softmax, you have the probability of each class.

So therefore, you can use the loss function to reduce that just using one neuron.

So I would like to know what's the difference between taking just one neuron and many other neurons for it? - Let me see if I understood correctly your question.

Because for instance, what you're saying makes a lot of sense when you have two classes, for instance. - Yeah. - In the case of two classes, you have a choice, actually, of actually-- of only having one neuron.

In the case of having only two classes, like in a binary problem, the cross entropy reduces to a particular form, which is the binary cross entropy-- well, the loss function reduces to binary cross entropy, which is a particular variation of cross entropy.

And in that case, instead of applying softmax, you usually apply sigmoid.

And then with one neuron, usually, if you have-- if the probability is above 0.5, then you say it's a 1.

Depending on how you define the problem, usually it's a 1.

And if you have it below 0.5, then you will have a 0.

And that's usually how you solve it.

Alternatively, you could have two classes, two neurons.

And then you apply softmax, and then that will give you the probability of having the probability for each of the classes, even if it's only two.

And then you use softmax, and you use exactly the same approach as explained in the video.

So in that sense, when you have only two classes, you can do so.

When you have a multi-class problem, then it's a little bit trickier because when you have a multi-class problem, not necessarily, but what you are after is-- I mean, you have, for example, five classes, and in the end of the day, the loss function, it's only minus the natural logarithm of the prediction by your model of y hat, the prediction of the neuron that has the-- that should be the highest output.

It's not necessarily highest because it's definitely what you're trying to compute.

So in that sense, your question is why do you not have only one neuron for the output of the-- Yeah, because I saw that when you use multi-class problems and you use the softmax function at the end, you will end up with a vector of the probabilities.

And maybe that's my error.

That's my misunderstanding that you end up with the probability of each of the classes in one vector for one class.

Let's say, like when you said for the cat, so you have one vector with the probability of each class.

And you just take into account the probability of the cat if you're focusing on the neuron of the cat.

Right?

So my question is-- But in the vector, you have the probabilities for each of the classes.

Yeah, of course.

That's my question.

You have one neuron with the probability of each of the classes, but you just take into account-- like you just focus on one of those probabilities, right?

OK, you are focusing on the one-- because you know somehow-- I mean, the fact that you need a neuron for each of the classes is because you're doing supervised learning.

In a way, it's not the only reason, it's one of the reasons you could argue that is because you're doing supervised learning.

So in the end of the day, the loss function will be only minus natural logarithm of the predicted class by the model.

OK?

Yeah, so my question is, but when you have that vector, you also have the probabilities of the other classes.

So my question is, what's the improvement of accuracy?

Just focusing on the specific class that you're trying to focus, and like that, instead of taking just one vector and trying to reduce the loss function using the result of the final vector of just one neuron?

Because that neuron is also giving you the probability of the rest of the classes.

That neuron is also giving you the probability of the rest of the classes?

Yeah.

Like if you have 35 classes, you have to say, and you have the output of one class, how could you know the probabilities of the other 34 classes?

Through the self-max function, because with that, you have the whole vector and the vector all the observations.

But for that, exactly, you have the vector of all the classes.

I'm probably not following your question well.

Because in the end of the day, you need the-- when you have a multi-class problem, and I'm talking about more than two classes, that vector giving you the probability for each of the classes?

I mean, that might not-- I mean, you could argue that having all the probabilities, my-- I don't know.

Depending on your problem, you may need them.

You may not.

You may argue that probably you don't need them for your problem.

But you definitely need them in order to compute the gradient and to compute the loss and start doing the back probe.

I know how to explain my question.

I sent the chat the animation class.

I'm looking at the image, yeah?

OK, so if you see that image, you would see that you have-- the result is three vectors.

And each of those vectors have a result of each of the classes.

Right?

Yeah, the first vector is not-- the three vectors-- and I'm talking about the column vectors-- is not for each of the classes.

I mean, the column neurons-- let's talk about the cat only, for instance.

OK.

The column of the cat.

I mean, you have three neurons, one neuron per class.

But the whole vector is just for the image corresponding to the cat.

The second column would be for the image corresponding to the dog.

And the third one would be for the-- not for the class, but for the input.

In this case, it happens to be like cat, dog, and whatever.

But imagine you have another dog, a different dog.

That is also-- that you should be classified as a dog.

I mean, this is supposed to be like a husky or something like that.

If you had like a chihuahua, for instance, then you had a different dog.

But you would be expecting that it is the neuron in the second position, the one that activates.

So the columns are giving you vectors for the input.

Like, is the probability of that image corresponding to each of the classes?

That's what the positions in the vector give you.

Like, the green is going to be the cat, the orange is going to be the neuron for the dog, and the gray is going to be the neuron for the bird.

Yeah, so that's actually my question.

So you end up having-- like, you can just end up having like just one vector that gives you like the likelihood of each of the classes for one specific class.

But I mean, maybe this might be confusing because-- It means that the columns are not vectors for the class.

The columns are vectors for the input, for any given input.

Like, in this case, it's only three inputs, but you can have like-- I mean, there might be a variety of cats, a variety of dogs.

I mean, you will have-- in this case, you have three classes, but the number of inputs can be a lot.

So every one of the columns, it would be the-- is not-- in this example, I mean, it's very simple.

But what you're assuming is that you are passing three images at the same time, one image per class.

That doesn't have to be the case.

You could have to cut three cats or two birds and a cat, whatever.

I mean, in this case, it's three images at the same time.

You are processing the three images simultaneously.

And every one of the columns is the output, the model's output, for every one of the input images, not for the classes.

The classes are given just the-- the classes-- the probability for the class for every input image is the neuron, is the color-coded cell to call it like that.

The columns don't represent classes, they represent inputs.

Yeah, right.

OK.

I'm trying to process that information.

For instance, imagine that you didn't have the-- like in the things that is within the parentheses, within the softmax.

Let's say that you only have the first one, the cat, right?

So if you only have the cat column, the output would be only the first column, the column that has 0.39, 0.228, and 0.33.

That would be if you only have-- and then that output would be representing your probability for that input image.

I mean, the green cell will give you that the output is 39% cat, then 28% dog, and then 33% bird.

So that would be the probabilities for that input image of the cat.

If you didn't pass the cat, but you passed the second image, you're the dog, then the output would be say that in the softmax-- the input to the softmax function was only the middle column, the one with the dog.

Then the output would be only one column, where you will have one probability.

I mean, the probability for that image to belong to the cat, dog, and the gray cell would be the bird.

So the vector is giving you-- the column vector is giving you the probabilities.

Let me get my item too.

Yeah, I mean, that's a bit-- like that's close to my question, because in that case, if you use just one observation, you will see that you have the probability of each of the classes in the vector, right?

So once you use all the observations, and you compute that vector, and then you compute the loss function, and take into account the probability of just one vector-- so in that case, you're just having just one output in your-- because I didn't-- yeah.

Probably.

Let me see if I can just share this.

Because if you can-- if you have only one image-- and actually, the number of images doesn't matter in the end, I mean, because it will be just more columns in this sense.

But if you had only one image, say the bird, you only have the bird, then to compute the loss, in the end of the day, you are going to use only the output corresponding to the neuron corresponding to the loss.

So indeed, it's only one value.

It's going to be minus the natural logarithm of the-- if it's the bird of the output corresponding to that neuron of the bird, the gray one.

Because you know it's a bird.

But if you're passing an image that corresponds to a dog, then it will be minus the natural logarithm of the neuron corresponding to the dog.

And so-- You're right.

So in that sense, I mean, having the different classes allow you to compute the loss.

And by computing the loss and just minus the natural logarithm of the output corresponding to the neuron, you should be getting-- I mean, that should have the maximum probability.

That's supervising the model.

That's telling the model what's the correct class.

Yeah, I understand.

By seeing this at this moment, my question was about the-- for having just one neuron.

But seeing this, the whole process of the calculus should not have any sense.

Because if you don't have three neurons, you can't even have a vector of three outputs.

So in that case, my question doesn't have any sense.

So I think that I understood the concept.

OK, OK.

Yeah, if there's any other question, you can send me an email.

And we can arrange a quick meet.

OK, sure.

Thank you.

Oh, thanks for the question.

Is there any other question, any other comment from last session, from the videos, from anyone who has started the homework?

You will see that there is a little bit of overlap in the videos.

Sometimes it might feel-- well, for some of you, depending on-- I understand that some of you have very busy jobs.

And so sometimes it will see that there is a lot of videos to watch.

But then you will see that next week, most likely, the last two or three videos might be the same videos.

So that's just like to give you a bit of breathing room.

Some videos are going to be like the last videos of one week.

Might be the first videos of the next week.

I mean, if you watch them the second week, then you don't have to watch them on the third week.

There will be a little bit of overlap.

Has anyone tried running the 1A notebook?

But we can do next week, because you still have next week to hit a notebook.

So I will show you that I was running it.

I have run in-- probably this is important.

Let me show my screen.

To run-- show you the 1A.

You will see that it's very similar.

Well, actually explaining one of the latest videos.

Anyone just confirmed that you can see my screen?

Probably.

Yes.

This week.

Yes.

OK, you will see that in the notebook.

I don't have these cells.

This might be-- if you are running the notebook in collab, you might be very familiar with these.

That's why I don't mention it.

And not everyone runs it in collab.

So what if you do run it in collab?

We will collab.

You will see-- you will need to share the drive.

And this line here also allows you to-- this is tricky.

And I think it's not very intuitive in collab.

Because the fact that you are-- that the notebook is in a given directory-- for instance, I have the notebook in this directory.

Up to here, probably most of the people will have the same path.

Because that's the default path for Jupyter notebooks in collab.

You might have changed it.

That's fine.

But some of you will have this path for sure.

Then I have the DC55033, which is the code of the class.

And then I have the 2025 version of the code.

So doing this allows you to-- the notebook is running in my notebook.

This notebook is running in that-- it's here, right?

So I have the 1a, 1b, the file, and the get\_images.py.

The information was provided.

So you would expect that just by having the notebook in the same directory with your files and your images, you would be having everything like you have it locally, right?

And that's not the case.

So it is until-- if you run this cell, a cell like this, it allows you to actually be-- to pretend that you-- not pretend, but to actually work as if it was a local directory.

Like, this is already mounted.

I do this.

And then if I run a current like this, you will see that actually the path I am-- my current path is exactly this one.

But if you run the notebook and you don't run something similar to this cell, there might be a few variations of this.

This is the one I like because it's like the simplest I can find so that everything seems to be local.

So if you don't run that, you will see that the current path-- like, I don't know if I can run this.

So I'll see.

Like, if I run only this one, it's to ask me if it's already mounted.

I'm not totally sure if this is going to be like-- it's still here.

But if you close everything, like, you stop the session and you go out and you come back, you will see that the path is not the same where your notebook is.

So this helps for that.

As I said, you might be aware of it.

But on common boat, still, probably worth mentioning, including this line where this is the path where you are working.

That's if you are working in Colab.

If you are working locally, then you should do it in your path.

If you are using some other host service, like I mentioned last time, Lightning AI.

Lightning AI has-- Lightning AI is a studio that's an interesting offer with a few hours of GPU.

For this, we don't need GPU, by the way.

I mean, we are going to be using GPU until the next homework when we start using PyTorch.

And yeah, the code is explained in the videos, the final two videos that are going to be also for next week.

They explain-- I will just go very quickly with the videos.

The videos that-- sorry, the explanation of the notebook, not in a lot of detail because it's going to be from what you will see in the videos, but just want to mention that we are going to be working with MNIST.

MNIST has been now widely called the Hello World of Deep Learning.

It's probably the most famous data set in deep learning.

It's the one that I was mentioning last time, that Jan Lykun in the late '80s is one of his contributions from that time.

I mean, these handwritten digits codes because convolutional neural networks were the ones that were used to recognize digits in the post service in the US and also for ATM machines or ATMs to read the checks, the numbers in the checks.

So this is by today's standards.

This is a very simple and like a toy data set.

But I think it's still nice to do.

There have been variations.

Like there is the fashion MNIST, which is like very basically the same kind of images, very tiny images, 28 by 28 pixels, or they are of clothing like fashion.

So this is the data set.

Let me see if I can find the-- this is the-- I think this is the original website from Jan Lykun's website.

Jan Lykun probably cannot access it.

It's not loading.

But well, anyway, so the MNIST row that you are going to see is the data set that you would have downloaded from this website, from the original website of MNIST.

For some reason now they have like password protected the data set.

So well, I still have it.

So you can still get it like in many places, because it's a very common.

You can even use it like in PyTorch.

PyTorch will have it reloaded if you wanted to load it from PyTorch.

You can also preload it from Kaggle.

There might be other places.

So what I still-- the one you're going to be using is like the original in the MNIST website, is the website that is not loading.

And this code that we are giving in these get images, is just a very simple Python code.

You can have a look at it.

It's just how to preprocess the data so that you get them from these serialized Python files to basically non-py vectors or non-py arrays.

And we are doing a little bit of reshaping.

You will see that there is 50,000 images for the training set.

And you will have 10,000 images for the test set.

So basically what we are doing is-- sorry, 60,000 for training and 10,000 for test.

So what we are doing just in this part is just to take the 50,000 images, the first 50,000 for training, the last 10,000 of those trainings of that training set for validation.

And then we have the test set for the 10,000 images.

Recall that one of the things that we are looking for in the homework is that you create nice comments for the functions.

You don't have to comment line by line.

That's probably not needed.

When you think it's needed, then please do so.

But at least comment per cell.

I mean, that is something that we're expecting.

And well, then we do a normalization of the data that is common in any machine learning application.

You don't think the number is just-- I think it's a very good practice.

You might have already seen this in other machine learning classes.

But looking at your data and getting to know your data is really important.

In this case, you know the data and it's very simple.

But still, it's a very good idea.

There is this library that is a new company.

The conference is 51, which is-- they have a lot of tools to visualize your data and actually do annotations on the data and analyze the data.

It's called 51.

Yeah, it's interesting.

I mean, it's probably for-- not probably, it's definitely for more 51 library.

Boxer 51, yeah.

This is the library.

They have a very nice tool to actually do annotations and then check your annotations and then check for-- you can plot the distributions of your images.

And actually, you can select every single dot in there.

In that, it will show you dots of the distribution of your images.

And then there might be out layers.

And then you can click at the dot and it will show you the image.

And it might show that actually that image is noise.

That's like a nice tool to clean your data.

So all this is just to support the idea that actually looking at your data, getting to know your data, is really important.

If you have millions of images, it's going to be impossible to see them all.

But at least you should have a good idea, a good idea about your data.

So this is yours.

And this is what you're going to be implementing.

I mean, this is going to be from scratch.

So the other bit is explain this code in detail.

So I don't want to go into a lot of detail.

It was very general level of abstraction.

We create mini batches.

Have you already familiar with the concept of mini batches?

Like in the example that we-- sorry, yes, say that?

No, in my case, no.

Yeah, what we're going to say, like for instance, I'm sure-- can you see the image now?

The image that we were talking about before in the class?

I just put it in my screen, but I don't know if you can see it.

The one for the five?

Yeah, the image that we were just talking about, the image of the softmax that says softmax.

And we have the three images.

No, we cannot see it.

You cannot see it?

OK, let me see it now.

Maybe I was sharing of the best stuff, but only the app.

We were seeing the collab notebook.

Only the collab notebook.

Yeah.

This should be in the collab notebook.

And now can you see the figure?

Yes.

Yeah.

Yeah, OK.

But I'm sure that the collab notebook before disappeared.

We saw it for a moment, but it just disappeared.

I think it's in the chat.

I know why, because I think it's from Zoom.

Yeah, I think it's in the chat for anyone who's looking for it.

Yeah, the image that's sent before.

Yeah, that image.

What I was going to say is the concept of mini-batch.

I know why you cannot see it, because I would have to download it.

Let me download it, and then I will open it.

Because it's from Zoom, so I cannot share.

Zoom doesn't let you sharing something that belongs to Zoom.

Let me save it, and then I should be able to open it.

I just want to show you-- Just to illustrate the concept of the mini-batch.

Sorry.

You can share what you are using in Zoom if you change the setting.

I believe it's in general settings.

I will look for it, but I believe it is in share screen.

OK.

Then you can choose-- I will look for it and let you know.

OK, thank you so much.

In the meantime, I think you can see it now, right?

Yes.

Yeah.

Yeah, yeah, yeah.

OK, yeah.

OK, what I was saying is in this case, in this example, what we are doing is the multiplication of three images with the matrix, right?

And since it is a matrix multiplication, and ideally we are using something that parallelizes the computations, like GPUs are ideal for that.

And then that's why we have CUDA and all-- we have all these libraries.

So the multiplication, the matrix multiplication-- because we have now the matrix with the weights, the parameters, the things that the network is going to learn.

And then we have a matrix where every column is an example.

So in this case, we would say that we are using a mini-batch of three images.

This means that we are passing to the model three images at the same time.

Say, for example, if we had 1,000 images in our data set, just a very simple number, we had 1,000 images, we could let the neural network see only one image at a time.

And in that case, we would say, OK, one image, then one iteration, another iteration, and then another image.

And then we would need 1,000 iterations to complete the whole data set.

Does that make sense?

Yes.

So that means we will be needing 1,000 batches, for instance?

No.

That only means that we will need 1,000 iterations.

OK.

To complete the whole data set.

OK.

Right.

When we do that, when we complete the whole data set-- and that's another important concept-- we say that we complete an epoch.

When you hear the word epoch, some people call it epochah.

Like in Spanish.

I don't think that's a problem.

But whenever you hear the word epoch, that means that you saw the entire data set once.

So say that you have 1,000 images, and you are passing one image at a time.

In this case, it would be like passing only the cat, only one cat, and then only the dog, and then only the bird.

So you complete like that.

One at a time, you complete the 1,000 images, you complete one epoch.

And for that epoch, you required 1,000 iterations.

OK.

Now, you have a very powerful GPU.

Say that you have-- OK, I have a lot of memory in my GPU, like really, really a lot of VRAM.

And so I can pass the 1,000 images at the same time.

That will be the other extreme.

Right?

So I would say, OK, I will pass the 1,000 images, my entire data set at the same time.

And then so for that in that case, you will have only one iteration, because you are passing the 1,000 images.

So you will need one iteration to complete one epoch.

OK.

In the first case, the one that you are passing one image at a time, you would say that your mini batch size is one image at a time.

In the latter case, the other extreme, it would be that your mini batch is of size 1,000.

Now, usually, passing one image at a time is not bad in terms of learning, because it has a lot of noise.

It's not good in terms of efficiency, because you will be doing everything sequential.

And you have a very nice GPU.

So you would say, OK, you have a very nice GPU, you could pass the 1,000 images.

Say that you can.

I mean, usually, you cannot pass your whole data set in one go.

But say that you could.

That's also not a very good idea, because it helps the training process to have more iterations within every epoch.

So what you do in the practice is to have a compromise between having one image at a time and having the entire data set at a time.

Say that instead of having one or 1,000, we say, OK, let's pass 100 images at a time.

So you will say that your mini batch is of 100, of size 100.

And then you will need 10 iterations to complete the whole data set, 10 iterations to complete one epoch.

So that's what this function that is called creating mini batches is doing.

OK, just got a question there.

So you're saying that tuning the epochs and the batch sizes depends on the infrastructure we will work on?

Yes and no.

I mean, the simplest way to say it is that, yes.

OK, let me tell you why, yes.

Doing one image at a time is extremely inefficient, extremely, extremely inefficient.

Because you will be-- I mean, if you have 1,000 images, you will have to pass one, and you will have to wait.

Every iteration is sequential.

You cannot-- well, depending on how you program, but in general, iterations tend to be sequential, depending on how you are computing that.

But in general, you could say that iterations are sequential.

So you will need to wait for 1,000 iterations.

But you may have capacity in your infrastructure to run several images at a time.

So in that sense, it helps to have-- to pass several images at a time, and that's the size of the mini batch.

This MB size is-- say that I'm going to pass 100 images, as I said in this example, then that will be the size of your mini batch 100.

So that will help for efficiency, because now you will have only 10 iterations, and so only 10 sequential steps.

And that's the reason why it helps.

I mean, it has to do with your infrastructure.

Now, having said that-- and this is probably a more subtle point and a little bit more advanced-- is that if you had enough resources to pass more images, more data in one single step, to make your mini batch way larger, that still is not a very good idea.

Even when it will help for efficiency in the time, it will take to train.

It's been shown that mini batches that are above 64 elements tend not to lead to very good results in terms of the quality of the training in your model.

So it helps to have a small mini batches, but you cannot have them that small, because it's going to be extremely, extremely slow.

So you need to get a compromise.

So that's why you have-- that's why it depends on infrastructure, but it's not the only reason.

Is that OK?

Yeah, but do you-- so the rule of thumb is like 65 elements in one batch?

Yeah, I mean, usually-- and this is also something that has to do with history a little bit-- is that usually you use powers of 2, because it used to be the case that the framework's very optimized to work with powers of 2.

So you would say 4, 8, 16, 32, 64, and so on.

So say that you pass 64.

Empirically, the optimal sizes for most applications and also this is depending on the application.

But for most applications, assuming you have enough computational resources, is 32, 64, no more than that.

Then to be heuristically-- there's a paper for that.

I could share it in the announcements that shows that mini batches larger than 64 are not recommended.

Another caveat is that I said powers of 2, and I said that's a little bit of history, because also in 2021, '22, something like that, there was a paper that showed that that rule of thumb of using powers of 2 for the mini batches probably had some sense in the past, but now it's not relevant.

I mean, you could have 100.

I mean, you could have 50 or numbers that are not powers of 2.

So what I'm still used to using powers of 2.

So that's why I say 32, 64.

So in general, you would say, OK, try 32 if you have enough computational resources for that.

Like for a problem, like the one you are going to be addressing here, that is really simple.

You can have larger mini batches, and it will work fine.

But still, when you are working with harder problems, you will probably not want to use such a large mini batch.

Also, problems like generating images, like diffusion models, the ones that draw the pretty pictures of-- I don't know.

Please draw me-- I don't have enough.

Well, I had a picture of a dog wearing a hockey suit.

But I don't have enough.

But still, you say, OK, a dog playing hockey, right?

And it generates the pretty picture.

Those kinds of models are really computationally expensive.

And usually mini batches tend to be like 2, 4, no more than that.

So it depends on the problem.

But in general, for classification problems, you may try with 32.

That's a good number.

OK, got it.

So one last question.

Regarding the number of epochs that the model will use, having a larger number of epochs will turn on overfeeding or underfeeding or something like that?

It might.

I mean, the number of-- if you have a very large number of epochs, maybe overfeeding is more likely than underfeeding.

A smaller number of epochs might be more prone to underfeeding.

But still, that's very dependent on the problem.

And that's a really good question and a really subtle point and a really difficult thing to give a general answer.

The best advice I could think of is to-- if you are training your model, every so often plot the loss function.

More than the accuracy, the loss function.

If you see that the loss function is still going down, like it hasn't totally plateaued, then there might be sense in doing more epochs.

That's why you look at the loss function.

If you see the loss function-- because ideally, your loss function should be something that is not like-- Like a logarithm?

Shouldn't be that.

I mean, that would be like, wow.

Like if you have a loss function like this, can you see the draw?

Yes.

Yeah, if it's like that, that's very ideal.

I mean, that might be really unlikely to get something like that in a real problem.

Occasionally, it might happen.

You should expect something more like this.

And this is a good loss function.

OK.

I mean, the trend is down, but there will be noise.

Because you usually plot the loss function per mini patch.

And every mini patch has different data.

And the data might be-- your model may like some kind of data more than other kind of data.

And so that's why sometimes it will be a little go up and then down and down.

But you would expect that the trend is like that.

Right?

It's going down.

Yeah, you might see some things that-- if you see something, this is what you don't want to see.

And it goes like that.

That is what you don't want to see.

When training LLMs like these large language models, especially in the first ones, there were some papers that were saying, training for longer helps.

And actually, there is a paper called Chinchilla.

Well, Chinchilla is not the official name of the paper, but the name of the model is Chinchilla.

So the paper is now the Chinchilla paper.

And they show that the larger your data set and the larger the training time, you get better results.

And there are-- it's just that it's difficult to follow.

But there are some papers that show that if your training function goes like this.

And then it goes a little bit like that.

If you continue training, it might go probably like this.

And that's good.

But this is when you're talking about really large data sets with really large models.

And we are talking like this time period might be one week.

So it's not something that we are going to be-- at least for the class.

Most people don't worry about this kind of things, even in real applications.

Because most people, even in industry, don't train for that long.

Most of the cases, you don't train for weeks.

I mean, LLMs, they do train them for weeks.

But most people wouldn't.

So something like this one is kind of a good loss function.

If you get something like this, OK, that's probably very, very easy data.

You should probably get something a little bit more noisy.

But to know the number of epochs in total, that's a tricky part.

That's why also you keep looking at your loss function often.

And also, if you see that you achieve something like this, say something like this, you're going like this, this is going good, good, good, good, good, good.

And at some point, it gets like this.

That you see that your loss function kind of plateau.

Like it reached a flat level.

This might be an indication that you are not going to get better performance.

And there are techniques called early stopping that actually check for this kind of behavior.

And you can check for the loss.

And if your loss function is not-- is kind of reaching this kind of plateau, you can adjust the percentage level of change that you are willing to keep looking for.

And the percentage of change in the loss function, I mean, if you say, OK, if he's not changing more than 2% and he's going for, I don't know, 10, 20 epochs, then probably I can stop the training.

And that's going to be my best result with the settings I have.

I currently have.

Early stopping.

Sorry, stopping.

Really good point.

I mean, not a unique answer.

And not a unique answer.

You will see that most of the things to train the models are not unique.

There are things that you actually have to train and test.

So that's what this function is doing, creating the mini batches.

Yeah.

Professor?

May it be the case that if my loss function got up plateau, could that be an indicator that this function is not the correct one, or this model is not the correct one for the type of data I have?

It depends.

Because for instance, your loss function might be like that.

I mean, you can have a loss function and then you plateau like here.

And the number of loss function is not that important.

You might have a 0.7.

And you would like to have a 0.1.

But that number is very relative to your data and to your problem.

But at the same time, then you have the accuracy function, or the good performance function.

And the accuracy function is something like this.

And you are plateauing here.

You are reaching a plateau.

But your accuracy is like 95%.

And for your problem, say that for your problem is 95 is really good, then you're happy.

And then you say, it doesn't matter that it plateau.

You stop the training.

You do early stopping.

And you're happy with your 95% accuracy.

And that's good.

If it plateaus, if it does like this and then it plateaus, but then your accuracy is like, I don't know, 77%.

And 77 is not a good result for you.

Then you have to think about changing the model or doing something else.

Because the result is not good.

So the loss function by itself is not enough.

You need to have more data, more information.

But maybe you have a very difficult problem.

I mean, there are problems where you don't have-- we're having-- when having an accuracy that is just like, say it's two classes, but it's such a hard problem, that having 60% accuracy is really good.

So if in that kind of problem you are getting 77, then that's really good.

So it depends on the problem.

OK.

Thanks.

Thank you for the question.

These are really not very simple questions to answer, because they depend a lot on the problem you're solving.

So that was the creating a batches class.

Then you have the-- it's in the video.

It's very well explained.

Well, it's better explained in the video with more detail.

But this is just-- are you familiar with Python classes?

I have a question just before that.

Yeah, I just want to clarify one aspect.

In the example of the video, when you use the mini batch, that mini batch correspond like one observation to each of the classes.

I just would like to understand how those calculations change when you use different observations of the same class doing the process of optimization.

In the end of the day, it doesn't matter.

Because when you select your mini batch, say if you see the function, there is this shuffle.

You have, say, 1,000 images, that simple example.

And say that you have 10 classes.

So you have 100 images of every class.

Because you usually shuffle your data.

You randomize it.

When you select, say, your mini batches of 100 images, you select 100 images.

But there will be a variation of classes within those 100 images.

So when you pass them, say that if it was like-- in expectation, you will have like 10 images per class.

If it was 10 classes.

Well, that's not going to happen.

So in expectation, it would be like that.

But in reality, you will have 20 images of one class, 25 of dogs, and so on.

So when you pass all those images to your model, you will calculate a loss for every one of the images.

The model, the computation, will happen in parallel.

Because it's a matrix multiplication.

It's what we were explaining here.

In this case, it's only three images of different classes.

You might have 100 images where there are-- obviously, there are going to be several images for each of the classes.

And then you will have all those for every one of the images, and those for every one of the samples, which you will then take the average, which is probably the next slide in the video.

So and then you will take the average of all the losses for the individual images.

And that will be the cost that you are going to try to minimize, the average of the losses of all the images in the mini batch.

It doesn't matter what class they belong to.

Oh, OK.

Yeah, clear.

Thank you.

No worries.

Good question.

Thank you.

So what I was going to say is that this looks like we are not doing anything, but it's kind of important for the implementation of the model.

This is just creating our class that is going to be called non-py tensor, non-py MP tensor.

And that is inheriting from the non-py array object and nothing else.

But the fact that we create as a class, our own class, allows us to modify it.

And that's the reason we are doing this.

We can now modify the non-py array class with our own class.

We cannot modify the non-py array class.

So that's the reason.

And then we will be implementing objects, I mean, objects for the linear operations, for the linear layers, for the activation layers.

And then this is like a class that is going to be a wrapper that is going to allow us to add these classes.

Objects or instantiations, instances from these classes, from the linear.

The linear class is just going to do the matrix multiplication for a linear layer.

So that's the only thing that's going to do in the linear class.

I call it linear because actually PyTorch calls the class linear.

And you will see that implementation here.

So we will have our linear class, our ReLU class, which is our activation function.

And then the sequential layer class, which we'll receive objects from the other classes to build our model.

Then what we're going to do-- well, this is just for the-- this is a very naive implementation of the softmax function.

But still, for this example, it works.

If you were going to implement a more robust one, you would have to be very, very careful about floating point errors.

Like, there might be overflowing of the operations.

Very easy.

So you have to be-- you have to take care of that doing some more robust implementation of the softmax operation.

But for this example, it works.

If you want to read a little bit more about the more robust implementation of softmax, please feel free to do.

It's basically just using exponents properties to do operations, not just like the formula, but actually use it with properties of exponentiation so that you don't get this-- you don't fall into this kind of overflowing possibility.

Well, let's say this is a softmax.

Then this is the train function.

You will see that in any deep learning model-- and this includes a very simple neural network, such as this one, or a very fancy one, big one, like LLMs.

You will have-- the training process is not going to change.

You will define how many epochs you are going to train for.

This is the four.

For that, when you are talking about LLMs, for instance, this Chinchilla paper that I was talking about before, Chinchilla paper, they show that you need 20 times the number of parameters.

The data set-- let me say this because I didn't say it properly.

When you're training with text like LLMs, like ChatDTP, Gemini, and all these fancy LLMs, your training set instead of images is going to be text.

Every group of words-- sorry, group of characters-- let's say a word is called a token.

It's not exactly a word.

It's a little bit smaller than a word.

We will see this later on.

But what I was saying is that then if you're training with the whole of Wikipedia, with the whole of the internet, there will be a given number of tokens, words in your data set.

And the Chinchilla paper shows that if you have a model with a given number of parameters, you will see parameters.

If you haven't seen the video, you will see it this week.

And then also next week, you will continue learning.

The parameters are like in these images.

If the parameters in this very small time in neural network would be the values in this matrix W and the biases.

These are the parameters.

And these are the things that we adjust during the training process.

In an LLM, like ChatGPT-- I think this was the-- no, this was GPT3, the model before ChatGPT.

That was the number of parameters in that model was 175 billion with a B.

So it was 175 billion parameters.

That's a lot.

And the Chinchilla paper that came after-- after-- totally sure.

But what this paper shows that for the model to train properly, the data set should be at least 20 times the number of parameters in the model.

So we are talking about-- and in this sense, the data set means the number of tokens in the data set.

So we're talking about data sets that are trillions.

So trillions of tokens.

That's massive.

Like that's really massive.

So probably when we think about the number, it doesn't sound like a lot, but it's really massive.

So what I was saying is that in those cases, the number of epochs to train the model might be just 2, 3, 4, 5 epochs.

And that might take a lot of time.

It might be only one epoch, like going through the model through the whole data set only once.

Like so that's another-- that's related to the number of epochs.

That's why I wanted to say that.

In this case, in our case, we can select 10, 20 epochs.

That's going to be plenty, more or less.

So to train any model, we will select a number of epochs and run the number of epochs.

Then these four is going to be loading the mini batches.

In the example we have been saying, if you had 1,000 images and your mini batches were 100 images in each mini batch, then you will be generating your mini batches of 100.

And so these four will allow you to complete the 10 iterations, complete the epoch.

And then you will repeat and repeat again and repeat that process.

So those two loops, whether you have them in a single loop that multiplies these two numbers, or you have them like we have it here in two loops, which is the more common one, you will have them.

And then you will have to do three things, these three things that is run your model with the data in the mini batch, which is this instruction.

Run in the model.

These will calculate the output for each of the classes.

Compute the cost function or the loss function, however you want to call it.

That allows you to see how well your model is doing.

Compute the function.

Then compute the gradients-- actually, it's four things-- compute the gradients that are going to allow you to update the parameters and then update the parameter.

These four steps, you might have more fancest of in between, but at least those four steps you have to have.

Run the model, compute the loss, compute the gradient, update parameter, and so on.

And you will see these explaining the tutorials of this week if you haven't had a chance to see.

So that's the process of training a neural network.

Whatever neural network it is, it might take longer.

It might take two weeks, three weeks.

It might take like five minutes for this example.

But still, the process is similar.

And so we train the model.

We calculate the accuracy.

This is just like, you're going to calculate how many of the answers are correct.

And then this that I have here is actually-- this line is creating the model using the sequential layer class.

And then the sequential layer class is receiving linear objects, relu objects, linear objects, relu objects, and linear objects.

That's creating the model.

When we see PyTorch, we will see that it's actually very similar.

PyTorch works very similar to this.

That's the reason we are creating the model like this, like in classes, like this object oriented style.

This is kind of the way frameworks work.

Then we decide the mini batch size of 512.

I told you this is a very simple data set.

So you could even go larger if you wanted to.

Don't worry about that.

It's larger than 64.

For this one, it's not really critical.

Learning rate and then the number of epochs in this case is 20.

And then we train the model.

And then this is something I was training.

I mean, this was just like very quick training, and it got like 97% accuracy.

Is there any question?

What I was-- [INAUDIBLE] Hi.

I do have a question.

Just this.

I've been hearing that when you model a neural network, that you actually have to consider that power of 2 that you were talking about earlier for the number of neurons in each layer.

So I was going to ask if there's any truth to that.

And yeah, at first, yeah.

Yeah, probably there was at some point.

But this is one of those things that also really don't matter right now.

These kind of things don't matter.

I mean, it's probably a good stylistic practice, maybe, because it looks kind of nice to have powers of-- at least for me.

If I see a mini batch that is like 55, I would cringe internally.

But that's because I'm kind of-- my OCD would be the one that-- but probably is not needed.

Probably the same applied for the number of neurons in a layer.

You could go with different numbers that are not powers of 2.

It might have been the case at some point.

OK, great.

Thanks.

And the next one was from basically like the same interpretation.

I thought you would likely have a lower number of neurons per each layer as your model goes.

So I don't know if that's true also.

Like at the first layer, you will have, I don't know, 200.

And then 100 and then 50.

Like if that's a natural-- like if that's mean to something, there's any truth to that?

That's interesting.

To the best of my knowledge, in this kind of neural networks, multi-layer perceptions, like linear layer, linear layer, it doesn't matter.

There might be some studies that show something related to that, but I'm not aware of that.

To the best of my knowledge, it doesn't matter.

What matters-- I mean, the relationship that actually matters is that it is better to spread the number of parameters in length than in depth.

What does this mean?

That if you're going to have 1 million parameters in the neural network, it is better to have 1 million parameters spread in 10 layers than 1 million parameters in only one layer.

I mean, the results are going to be way better if you have more layers, if you are going to keep the same number of parameters.

Now, if you-- I'm not aware of that of the reducing dimension in the number of neurons per layer.

There might be something about it, but not to my knowledge.

OK, so, D'Ana, you're saying that it's kind of better to have a long architecture that having a lot of neurons per layer?

And you know how it depends on data and the problem and everything, but-- Of course. --as a general rule?

In general, better to have depth than width.

OK, great.

In general, it leads to better results.

There's actually a paper about that.

Oh, I would love to see the video.

Let me look for it.

OK, so I mentioned another paper that I was going to-- let me write it down because-- Yeah, I think it was the one from the power of 2, the size of batch.

Yeah, the mini batch size.

Yeah, let me write it.

Take a note of this.

Thank you.

Yeah, very cool about it.

[AUDIO OUT] Yeah, I'll do it.

Yeah.

Anything else?

Another question?

Yep, I got a question.

Yes, please.

So the LLMs right now are classified in the number of parameters that the companies used to train the model, right?

So larger number of parameters will cause a better model or what's the-- I am not sure if that affirmation is true.

That's what you would expect, at least.

Like the LLMA versions, like the open models from Meta.

LLMA.

I mean, LLMA has been amazing because actually, you can see that in general, you would say, OK, I'm talking about this LLMA 2.

I don't remember the version of LLMA, but it doesn't matter.

There is a LLMA that you would say LLMA 1 billion, LLMA 3 billion, LLMA 7 billion, LLMA 405 billion parameters.

Something like that.

I think that would be LLMA 3.

In general, you will see better performance in the 105 billion than in the 3 billion, by far.

I mean, that's expected, and that's how it works.

Having said that, I mean, and in that sense, the number of parameters, the more parameters, the better the performance, in general.

Like economists say, like, setter is parables, like everything else is equal, more parameters, better performance.

Now, you see that from generation to generation.

You can see that probably-- and this is not exactly the model, but this is the general trend.

Like, maybe LLMA 3-- like, there is LLMA 3.3 now.

Let's say LLMA 3.2, because LLMA 3.3, I think, LLMA 3.2, 3 billion parameters, might have better performance than LLMA 2, 70 billion parameters, even when the number of parameters, which actually shows that the things they do during training actually matter.

I mean, there are changes.

Because the issue with LLMA is that there are open weights, but not open everything else.

So we don't know much more about the training.

We don't know much more about the construction of the training process and the model.

We only know, like, we have the weights open.

So we can use the model.

That's how it was trained.

OK, just I was confused, because last week, I read a news about there is some company that is simulating GPT with less parameters.

So I was confused at that point.

That might be totally true.

I mean, and it's similar to what I'm saying.

I mean, like this company, I don't know what company it is.

It might be some startup or something.

But it might be the case that today someone can release a new model that has, say, chat GPT-- no, chat GPT, I don't recall the number of parameters.

But GPT 3 had 175 billion of parameters.

LLMA 3.3, that is only 70 billion parameters, might perform better than GPT 3, which is not chat GPT.

It's the one before chat GPT.

The original chat GPT was 3.5.

Now they are in 0.3 or whatever it is.

So it might be the case that actually a smaller model, because of whatever they did in the training, because of whatever data set they used, because of whatever they did, works better than out of the box GPT 3.

That's an older model.

That's what I'm trying to say.

So there are improvements.

I mean, the more parameters in general, the better.

But there might be cases when training or things that they do during the training process, improved models and smaller models might be better than previous models that were larger.

You would expect that if you have the same version of the model, like, say, LLMA-- and I say LLMA because it's the one I'm familiar with, but there might be-- you call whatever model it is.

But I say LLMA 3.2, say, you would expect that LLMA 3.2 with 70 billion parameters works better than LLMA 3.2 with 3 billion parameters.

That would be expected.

OK, so it might be the case that a smaller model is better than one of the GPTs in the past that were larger.

Because of-- I mean, the evolution is not only in the size.

I mean, the size is really important, but there is also improvements in the training and things that they don't learn now.

Like, LLMA is open, but it's only weight open.

Open weights.

It's not totally open so that we can see exactly how they go to the result.

OK.

And let's see some example.

When I took a LLMA model and I retrained that model with my own data, I am increasing the size of the training set.

And with those weights, I am able to make an emphasis to my data, right?

Absolutely, yeah.

Then what is the difference between doing that and making our retrieval augmented generation?

Yeah.

OK.

So when you are doing-- we are getting a little bit out of the scope, but let me answer briefly.

I'll try to do briefly.

When you are training, say that you have LLMA 3.2 with 3 billion parameters.

The process that you are saying is fine-tuning the model to your data.

You can actually fine-tune the model with your data.

You generate-- you would have to read the specifications of the model and how you prepare the data set.

And you might include your data-- pardon?

To train LLMA 3.2, 3 billion parameters with your data.

I mean, you already got it retrained in a lot of data.

That's what you call a retrained model.

So that retrained, you do fine-tuning on your data.

And then you fine-tune it on your data.

And you might find your data or whatever specific task that you want your model to carry out.

Like say you wanted to help talk to appointments.

So you can actually fine-tune it to deal with this.

What you are doing there-- and you will see that this model of 3 billion parameters that is fine-tuned with your data, when prompted something about your data, is most likely going to do better than a LLMA 3.2, 7 billion parameter that was not fine-tuned to your data when prompted with your data.

So now you improve the performance of this small model.

But what you have done is, depending on how you do it-- because fine-tuning an LLMA, even if it's only 3 billion parameters, is tricky.

I mean, the expectation is like 24 times like if you wanted to fine-tune 3 billion parameters, 1 billion parameter would be like 24 gigabytes of BDRAM.

So you are talking about-- you would need a GPU that is about 80 gigabytes of BDRAM to load something like LLMA 3.2, 3 billion parameters and fine-tune it.

So that's probably not easy, right?

Because-- and the fine-tuning process is not going to be expensive.

It's not going to take time.

You are going to need resources.

And it's not going to be that easy.

Still, if you manage to do it, then it will work better than the LLMA.

Then there are other techniques that we might talk about later.

I don't know.

Is it an important-- We might?

Pardon?

And in those cases, should I ask myself if I am fine-tuning or over-fitting?

If I am downsizing the parameters?

A lot, I mean.

You would-- usually you don't worry about over-fitting when you are fine-tuning an LLMA.

Because you don't have enough capacity to over-fit.

Do that.

OK.

Yeah, usually, because in the end of the day, you want your model to learn your data, right?

So having said that, I mean, fine-tuning is not easy.

I mean, fine-tuning an entire model is not easy.

Because you need a lot of resources.

Even when-- like, I've been in talks with people from Google, and they say, like, what?

Because people from Stanford will complain.

Like, we don't have enough resources to do.

Like, we only have 100 GPUs in my cluster.

And they say, OK, then you can probably fine-tune the small models like that.

OK, a start from I be able to do it.

I mean, it's a lot of resources still for normal human beings.

Even fine-tuning, when it's supposed to be like cheap, is not that cheap.

So if you manage to fine-tune, good.

Really good.

There are other techniques that are parameter efficient fine-tuning, where you don't fine-tune the entire model.

You only fine-tune some of the layers, or some of the weights, or you are fine-tuning the tokens.

And that's more efficient.

And that you can do probably in one, two hours.

Now the question about rack-- now, if you fine-tune the model and manage to do it, then your model is going to learn the information from your data.

And so when you query your model about the data, your data, you will get answers about your data.

That's good.

Your model now has this information.

No disinformation.

That might not be always efficient.

So if you cannot fine-tune, doing rack helps you.

Because actually, what you allow the model is to actually search for the information in the documents.

And then the answer to the query is going to be not just answering from-- because I'm not totally fine-tuned to give the answer.

So I go and look for the data.

You have usually a special agent that will do search.

You have to-- there are other issues, like orchestrations and things.

How do you control the agent that is going to look for the data and all that?

But in general, what you do is send an agent, look for the data, and then retrieve that data, and then show the data.

And then it's a combination of the search.

And what your model knows about information.

So in that sense, rack might be a way to avoid fine-tuning and actually prevent kind of hallucinations and things like that.

OK, got it.

Thank you.

No worries.

Good question.

Also, yeah, we'll talk a little bit more in the last two or three weeks about these kind of topics.

Of course.

So I think that if there is no other question, guys, we are going to call it a day for today.

OK.

[INTERPOSING VOICES] Yes, please.

I was doing or reviewing the activity A1B for this week.

When I trained the model and everything, my accuracy was like 0.75 or 6%.

That's good enough?

That's good enough.

I don't know exactly what's there.

What's there?

In the rubric, it says what's the expectation of exponse, 70.

0.70, yeah.

That's good enough.

Yeah, that's what I was thinking.

That's good enough.

You will see that it's basically just what I usually advise you to do.

And this is an advice I give to everyone, is because activity 1B, you could do it as simple as copying from 1A, right?

Just changing a couple of lines.

My advice is always when you do that and you feel like, OK, this is pretty simple.

Try to do 1B, but not looking at the code from 1A.

Not looking at the word.

And that actually helps to see if you are actually building the whole understanding of how to join the sheet.

OK.

And I don't know if I was checking that we have to remove two letters.

That's because we don't have the images or because-- No, that's because-- Is the J and the Z.

And those because the J is like this.

It has movement.

And the Z has like this.

So you cannot capture the movement in the picture.

It's because of that.

In the ACL system, right?

Yeah, exactly.

All right.

And one other question.

In the rubric states that we need to enhance the view of the file, do we need to add colors and make it beautiful?

Or what's the mean?

Or what's the-- The view.

I don't think that.

I think it's more like-- because I don't remember exactly the comment that you're saying.

Let me check it quickly.

It says in the rubric activity-- OK.

This one, right?

Can you see?

Readability, maybe?

Yeah, that's-- The quality of markdown documentation include headers and any other media to make it-- the new-- Yeah.

More readable.

Yeah, yeah, yeah.

Yeah, it's actually like adding markdowns to the-- like you can add markdown cells that actually explain something.

Like my advice would be in the cells that have functions for classes include like doc strings that explains what are the inputs to the classes, what are the inputs to the functions, and what are the expected outputs.

So doc strings doing that.

Like if you see the documentation of Python, all the documentation of-- yeah, Python is a good example.

You will see these doc strings for the functions.

Something like that.

And then markdowns are like-- well, markdown cells that have like formatted comments or formatted text that you might consider relevant.

I mean, in the end of the day, try to think-- Probably one advice to say is to try to build a notebook as if it was a tutorial for someone to follow.

And that would be-- OK. --I guess.

I like the approach to get your more professional.

No, thank you for the question.

I think that clarifies for-- not only for you, but for many of-- Thank you.

OK.

All right, then.

Then thank you so much for joining today.

Now Friday is not easy.

So yeah, hopefully next week is going to be on Thursday.

This was because I was arriving yesterday at 7.30 PM.

It was complicated.

Not what?

Yeah.

Thank you, Shuman.

Professional.

Thank you.

Thank you.

Thank you.

Thanks.