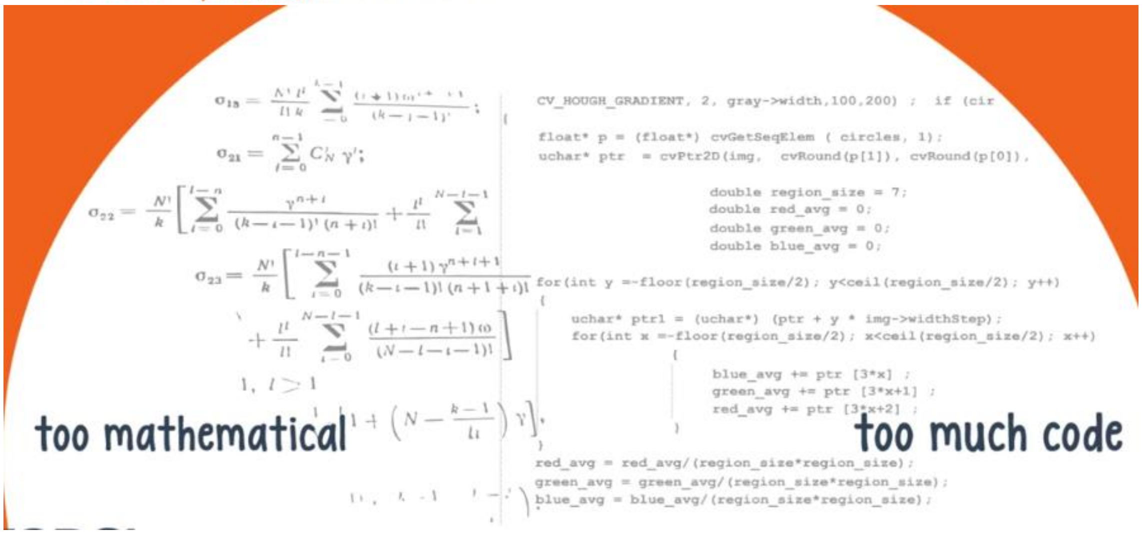
MODULE 1 – INTRODUCTION TO DEEP LEARNING

INTRODUCTION

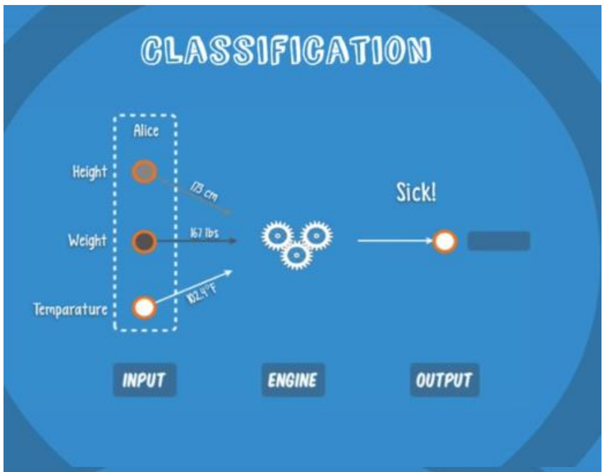


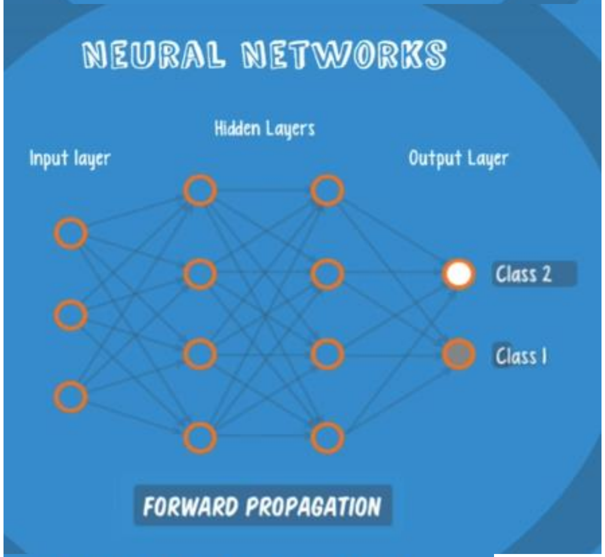
```
CV_HOUGH_GRADIENT, 2, gray->width,100,200) ;  if (cir

float* p = (float*) cvGetSeqElem ( circles, 1);
uchar* ptr  = cvPtr2D(img,  cvRound(p[1]), cvRound(p[0]),

                double region_size = 7;
                double red_avg = 0;
                double green_avg = 0;
                double blue_avg = 0;

for(int y =-floor(region_size/2); y<ceil(region_size/2); y++)
{
    uchar* ptr1 = (uchar*) (ptr + y * img->widthStep);
    for(int x =-floor(region_size/2); x<ceil(region_size/2); x++)
        {
            blue_avg += ptr [3*x] ;
            green_avg += ptr [3*x+1] ;
            red_avg += ptr [3*x+2] ;
        }
}
red_avg = red_avg/(region_size*region_size);
green_avg = green_avg/(region_size*region_size);
blue_avg = blue_avg/(region_size*region_size);
```

art of transcript. Skip to the end.
If you're like most beginners, trying to learn about Deep Learning feels like taking a drink from a firehose
you're hit with too much complicated info too quickly, and most of it ends up seeping out of your mind
If you're tired of all that, then you're gonna love the series I've created for you!
My goal is to simplify everything so that you know just enough to make sense out of all those technical details
If you've ever tried to look into Deep Learning in the past,
you probably immediately came across terms like Deep Belief Nets

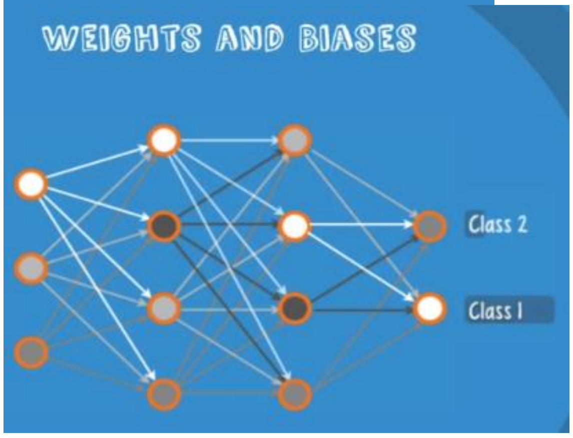
Convolutional Nets, Backpropagation, non-linearity, Image recognition, and so on
Or maybe you came across the big Deep Learning researchers like Andrew Ng, Geoff Hinton, Yann LeCun, Yoshua Bengio, Andrej Karpathy
If you follow tech news you may have even heard about Deep Learning in big companies
Google buying DeepMind for 400 million dollars,
Apple and its self-driving Car
nVidia and its GPUs
and Toyota's billion dollar AI research investment.
But there's one thing that's always hard to find:
an explanation of what Deep Learning really is
in simple language that anyone can understand
Videos on the topic are usually either too mathematical
have too much code
or are so confusingly high level and out of reach that they might as well be 100,000 feet up in the air
In this series, I'm going to explain Deep Learning to you without scaring you away with all that math and code
It's not that the technical side of Deep Learning is bad.
In fact, if you want to go far in this field, you'll need to learn about it at some point.
But if you are like me, you probably just want to skip to the point where Deep Learning is no longer scary
and everything just makes sense.
I know it sounds intimidating since there's so much information, but that's why I'm here to help!
At the very least, I want to get you to the point where you know how to take advantage of all the
great Deep Learning software and libraries that are available.
If you've ever struggled with finding clear information on Deep Learning,
please comment and let me know your thoughts!
Over the next several videos, I wanna bring you along step by step
until you know just enough where everything starts to make sense.
You won't know everything about the field, but you'll have a better idea
of what there is to learn and where to go next if you're interested in learning more.
We'll start with some basic concepts about Deep learning.
We'll touch on the different kinds of models and some ideas for choosing between them.
And don't worry – like I promised, we'll skip the math and go straight to the intuition.
Later, you'll learn about some different use cases for Deep Learning.
Then after that, we'll get to the practical stuff -
first you'll see some platforms that allow you to build your own deep nets,
and then you'll learn about software libraries you can use for your own personal apps.
YouTube is a great channel for these lessons because communication doesn't have to be one way.
If you ever feel that I'm being unclear or there's anything you'd like to add,
feel free to leave a comment and contribute.
The other viewers and I all want to hear from you!
End of transcript. Skip to the start.

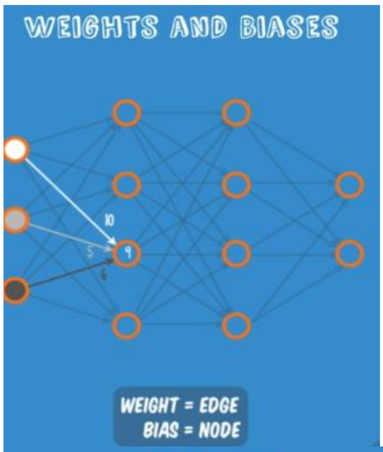
NEURAL NETWORK

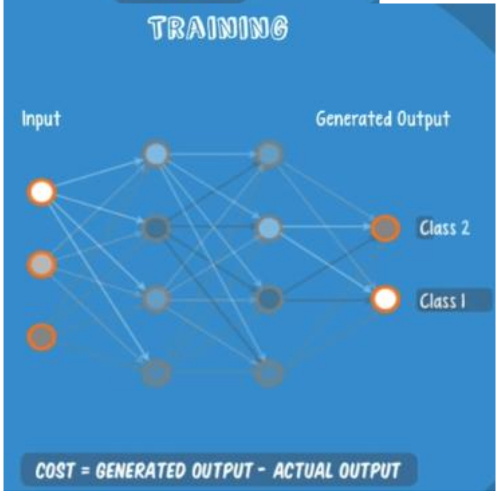


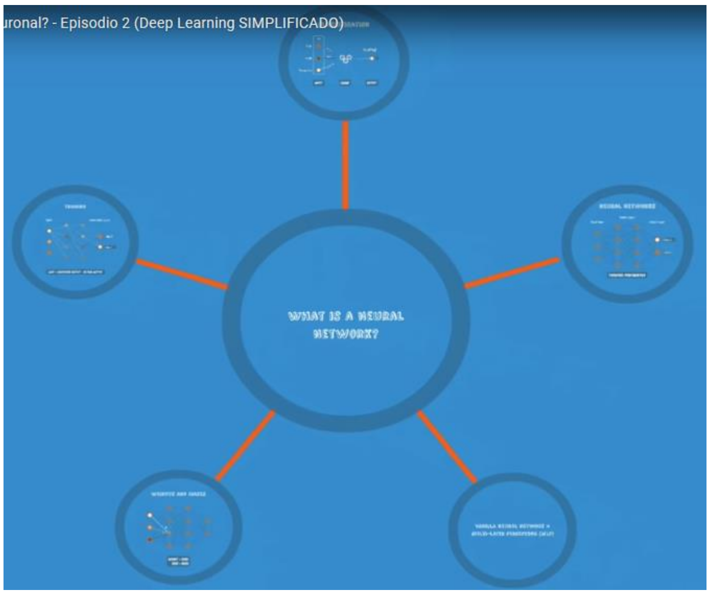


VANILLA NEURAL NETWORK =
MULTI-LAYER PERCEPTRON (MLP)







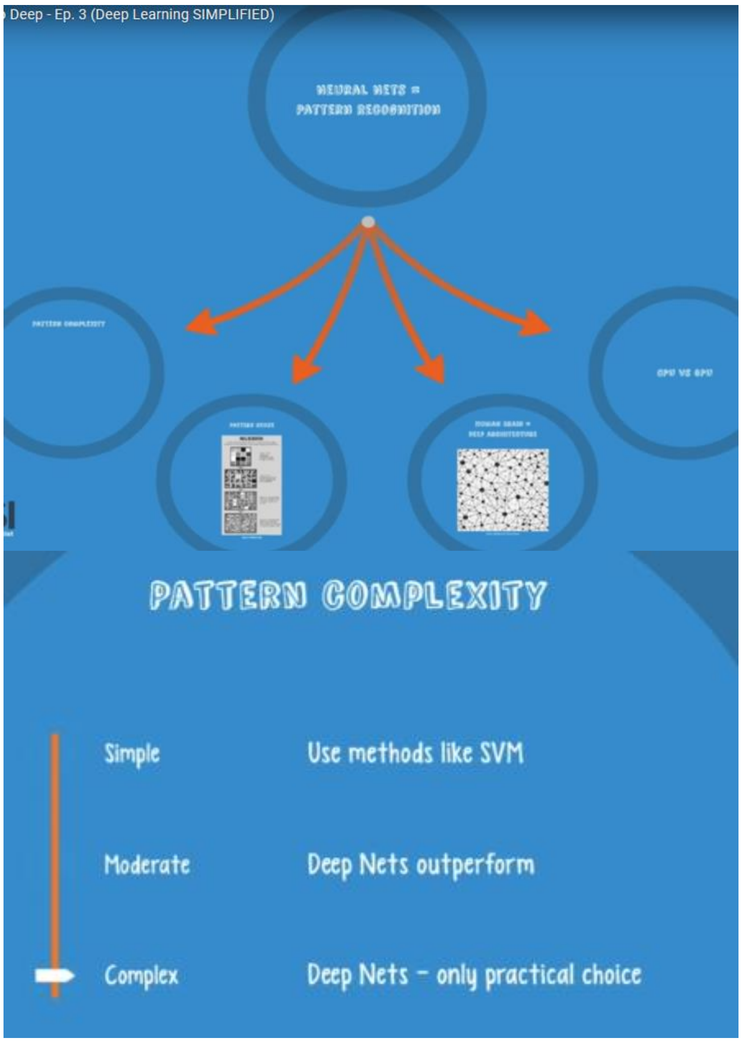

uronal? - Episodio 2 (Deep Learning SIMPLIFICADO)
WHAT IS A NEURAL NETWORK?

ranscript. Skip to the end.
If you've been ignoring neural nets cuz you think they're too hard to understand or you think you don't need them…boy do I have a treat for you!
In this video you'll learn about neural nets without any of the math or code – just an intro to what they are and how they work.
My hope is that you'll get an idea for why they're such an important tool.
Let's get started.

The first thing you need to know is that deep learning is about neural networks.
The structure of a neural network is like any other kind of network;
there is an interconnected web of nodes, which are called neurons,
and the edges that join them together.
A neural network's main function is to receive a set of inputs,
perform progressively complex calculations,
and then use the output to solve a problem.
Neural networks are used for lots of different applications,
but in this series we will focus on classification.
If you wanna learn about neural nets in a bit more detail, including the math,
my two favourite resources are Michael Nielsen's book, and Andrew Ng's class.
Before we talk more about neural networks, I'm gonna give you a quick overview of the problem of classification.
Classification is the process of categorizing a group of objects,
while only using some basic data features that describe them.
There are lots of classifiers available today -
like Logistic Regression, Support Vector Machines, Naive Bayes, and of course, neural networks.
The firing of a classifier, or activation as its commonly called, produces a score.
For example, say you needed to predict if a patient is sick or healthy,
and all you have are their height, weight, and body temperature.
The classifier would receive this data about the patient, process it, and fire out a confidence score.
A high score would mean a high confidence that the patient is sick, and a low score would suggest that they are healthy.
Neural nets are used for classification tasks where an object can fall
into one of at least two different categories.
Unlike other networks like a social network,
a neural network is highly structured and comes in layers.
The first layer is the input layer,
the final layer is the output layer,
and all layers in between are referred to as hidden layers.
A neural net can be viewed as the result of spinning classifiers together in a layered web.
This is because each node in the hidden and output layers has its own classifier.
Take that node for example -
it gets its inputs from the input layer, and activates.
Its score is then passed on as input to the next hidden layer for further activation.
So,
let's see how this plays out end to end across the entire network.
A set of inputs is passed to the first hidden layer,
the activations from that layer are passed to the next layer and so on,
until you reach the output layer,
where the results of the classification are determined by the scores at each node.
This happens for each set of inputs.
Here's another one...
like so.
This series of events starting from the input where each activation is sent to the next layer,
and then the next, all the way to the output,
is known as forward propagation, or forward prop.
Forward prop is a neural net's way of classifying a set of inputs.
Have you wanted to learn more about neural nets?
Please comment and let me know your thoughts?

The first neural nets were born out of the need to address the inaccuracy of an early classifier, the perceptron.
It was shown that by using a layered web of perceptrons,
the accuracy of predictions could be improved.
As a result, this new breed of neural nets was called a Multi-Layer Perceptron or MLP.
Since then, the nodes inside neural nets have replaced perceptrons with more powerful classifiers,
but the name MLP has stuck.
Here's forward prop again.
Each node has the same classifier, and none of them fire randomly;
if you repeat an input, you get the same output.
So if every node in the hidden layer received the same input,
why didn't they all fire out the same value?
The reason is that each set of inputs is modified by unique weights and biases.
For example, for that node,
the first input is modified by a weight of 10,
the second by 5, the third by 6 and then a bias of 9 is added on top.
Each edge has a unique weight, and each node has a unique bias.
This means that the combination used for each activation is also unique,
which explains why the nodes fire differently.
You may have guessed that the prediction accuracy of a neural net depends on its weights and biases.
We want that accuracy to be high,
meaning we want the neural net to predict a value that is as close to the actual output as possible,
every single time.
The process of improving a neural net's accuracy is called training,
just like with other machine learning methods.
Here's that forward prop again -
to train the net, the output from forward prop is compared to the output that is known to be correct,
and the cost is the difference of the two.
The point of training is to make that cost as small as possible, across millions of training examples.
To do this, the net tweaks the weights and biases step by step
until the prediction closely matches the correct output.
Once trained well, a neural net has the potential to make accurate predictions each time.
This is a neural net in a nutshell.
At this point you might be wondering;
why create and train a web of classifiers for a task like classification,
when an individual classifier can do the job quite well?
The answer involves the problem of pattern complexity, which we will see in the next video.
End of transcript. Skip to the start.

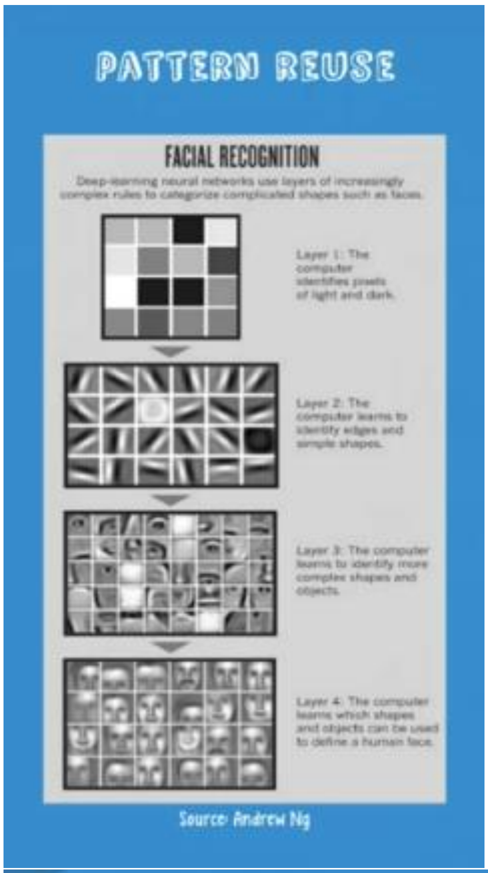
THREE REASONS TO GO DEEP


Deep - Ep. 3 (Deep Learning SIMPLIFIED)

NEURAL NETS = PATTERN RECOGNITION
PATTERN COMPLEXITY
HUMAN BRAIN = DEEP ARCHITECTURE
GPU VS CPU

PATTERN COMPLEXITY

Simple	Use methods like SVM
Moderate	Deep Nets outperform
Complex	Deep Nets – only practical choice



PATTERN REUSE

FACIAL RECOGNITION

Deep-learning neural networks use layers of increasingly complex rules to categorize complicated shapes such as faces.

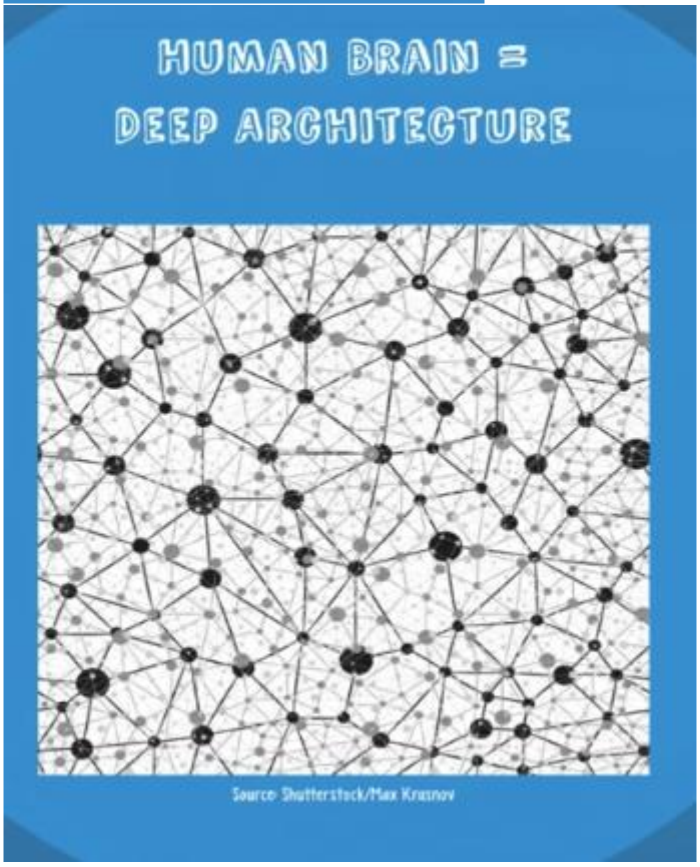
Layer 1: The computer identifies pixels of light and dark.

Layer 2: The computer learns to identify edges and simple shapes.

Layer 3: The computer learns to identify more complex shapes and objects.

Layer 4: The computer learns which shapes and objects can be used to define a human face.

Source: Andrew Ng

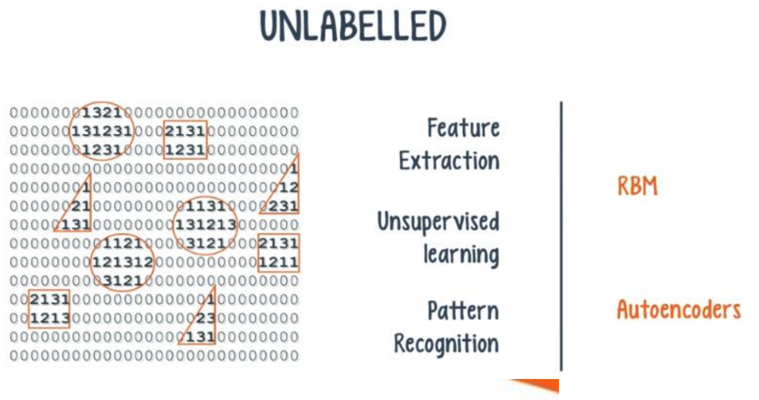


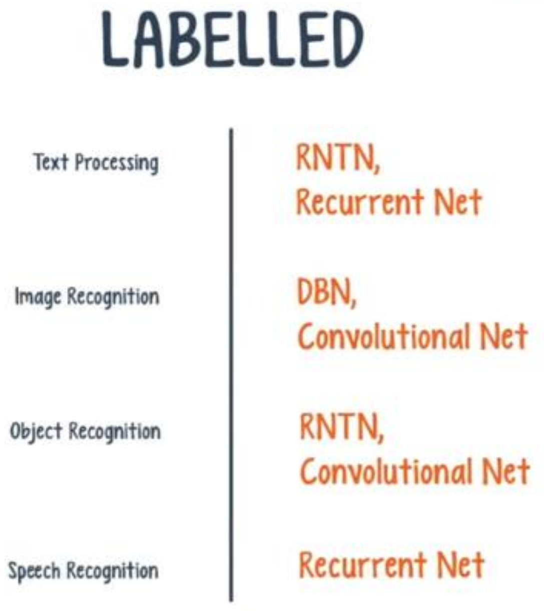
HUMAN BRAIN = DEEP ARCHITECTURE

Source: Shutterstock/Max Krasnov

of transcript. Skip to the end.
If you want your computer to recognize VERY complex patterns – then trust me on this – you really need to start using neural networks. When the patterns get really complex, neural nets start to outperform all of their competition. Plus, GPUs can train them faster than ever before! Let's take a look.
Neural nets truly have the potential to revolutionize the field of Artificial Intelligence. We all know that computers are very good with repetitive calculations and detailed instructions, but they've historically been bad at recognizing patterns. Thanks to deep learning, this is all about to change.
If you only need to analyze simple patterns, a basic classification tool like an SVM or Logistic Regression is typically good enough. But when your data has 10s of different inputs or more, neural nets start to win out over the other methods. Still, as the patterns get even more complex, neural networks with a small number of layers can become unusable. The reason is that the number of nodes required in each layer grows exponentially with the number of possible patterns in the data. Eventually training becomes way too expensive and the
accuracy starts to suffer. So for an intricate pattern – like an image of a human face, for example – basic classification engines and shallow neural nets simply aren't good enough – the only practical choice is a deep net.
Have you ever run into a wall when trying to work with highly complex data? Please comment and let me know your thoughts.
But what enables a deep net to recognize these complex patterns? The key is that deep nets are able to break the complex patterns down into a series of simpler patterns. For example, let's say that a net had to decide whether or not an image contained a human face. A deep net would first use edges to detect different parts of the face – the lips, nose, eyes, ears, and so on – and would then combine the results together to form the whole face.
This important feature – using simpler patterns as building blocks to detect complex patterns – is what gives deep nets their strength. The accuracy of these nets has become very impressive – in fact, a deep net from google recently beat a human at a pattern recognition challenge.
It's not surprising that deep nets were inspired by the structure of our own human brains. Even in the early days of neural networks, researches wanted to link a large number of perceptrons together in a layered web – an idea which helped improve their accuracy.
It is believed that our brains have a very deep architecture and that we decipher patterns just like a deep net – we detect complex patterns by first detecting, and combining, the simple ones.
There is one downside to all of this – deep nets take much longer to train. The good news is that recent advances in computing have really reduced the amount of time it takes to properly train a net. High performance GPUs can finish training a complex net in under a week, when fast CPUs may have taken weeks or even months.
Before we talk more about the various Deep Learning models, we're going to briefly discuss which types of deep nets are suitable for different machine learning tasks. That's coming up in the next video.
End of transcript. Skip to the start.

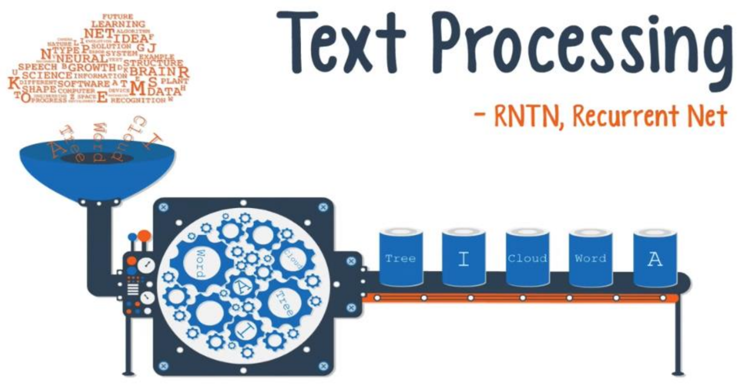
YOUR CHOICE OF DEEP NET



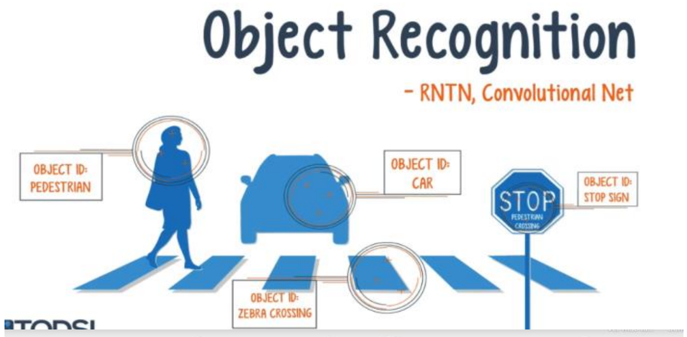


LABELLED

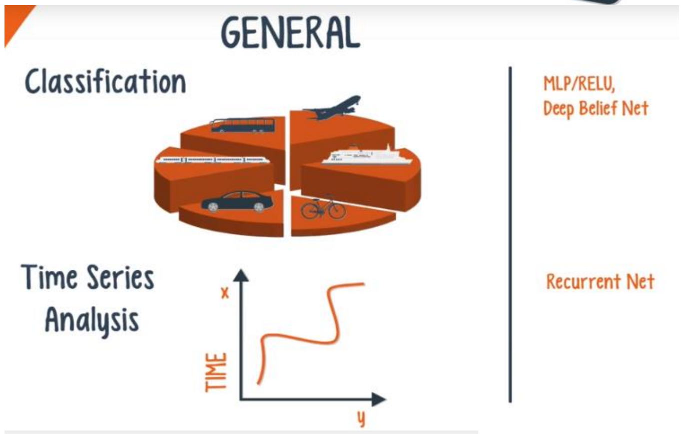
Text Processing	RNTN, Recurrent Net
Image Recognition	DBN, Convolutional Net
Object Recognition	RNTN, Convolutional Net
Speech Recognition	Recurrent Net

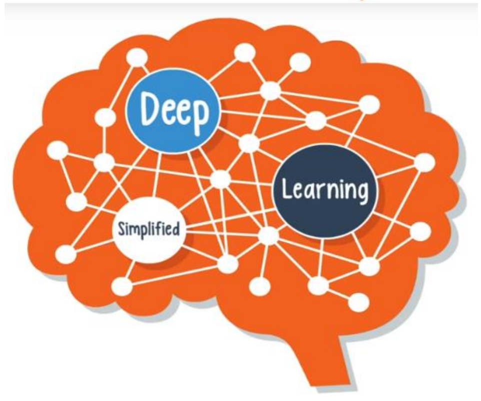


cat
Image Recognition
- DBN, Convolutional Net


Object Recognition
- RNTN, Convolutional Net
OBJECT ID: PEDESTRIAN
OBJECT ID: CAR
STOP
OBJECT ID: STOP SIGN
OBJECT ID: ZEBRA CROSSING

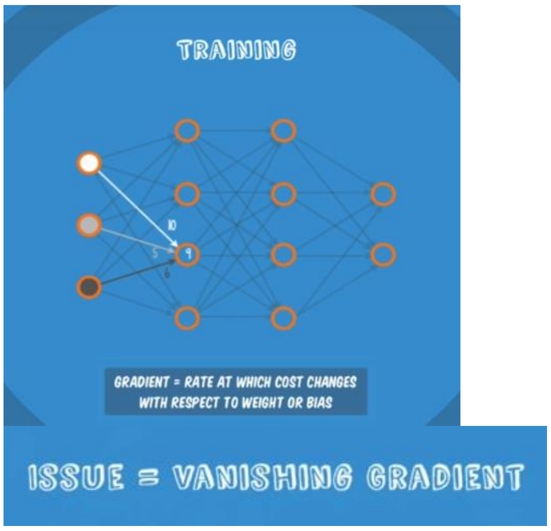
Speech Recognition
DEEP LEARNING
DEEP LEARNING...

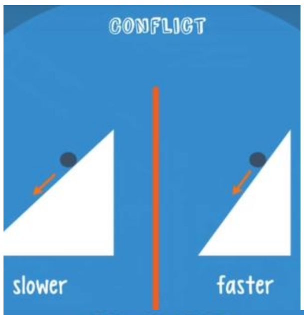

GENERAL
Classification
MLP/RELU, Deep Belief Net
Time Series Analysis
x
TIME
y
Recurrent Net

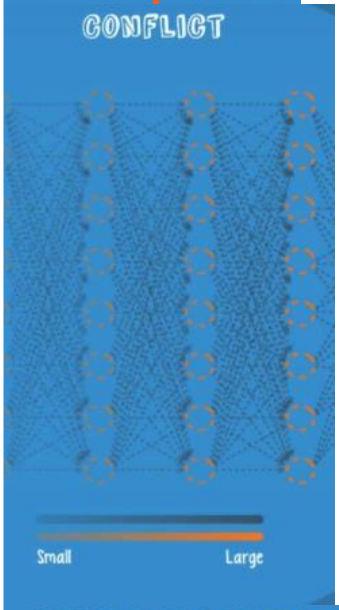

Deep
Learning
Simplified

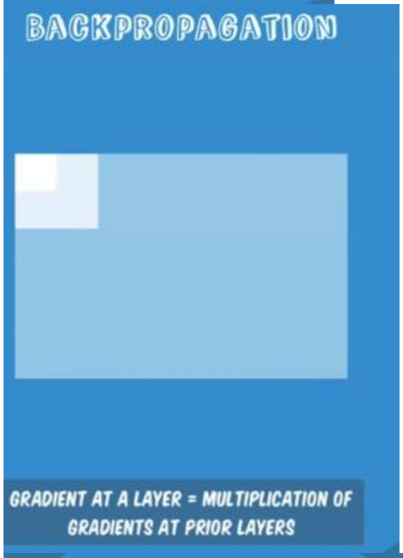
Now if you're like me, starting a Deep Learning project sounds really exciting. But when it comes to picking the right kind of net to use, well, things can get a little confusing.
You first need to decide if you're trying to build a classifier or if you're trying to find patterns in your data. Beyond that, I'll try to help by giving you some general guidelines.
Before we get started, I want to give you a bit of a heads up. I'm going to be using some terminology that may sound a little scary right now, but don't worry. I'll cover all these terms in detail in the upcoming videos.
If you're interested in unsupervised learning – that is, you want to extract patterns from a set of unlabelled data – then your best bet is to use either a Restricted Boltzmann Machine, or an autoencoder.
What type of projects would you need to use a Deep Net for? Please comment and let me know your thoughts.
If you have labeled data for supervised learning and you want to build a classifier, you have several different options depending on your application.
For text processing tasks like sentiment analysis, parsing, and named entity recognition – use a Recurrent Net or a Recursive Neural Tensor Network, which we'll refer to as an RNTN.
For any language model that operates on the character level, use a Recurrent Net.
For image recognition, use a Deep Belief Network or a Convolutional Net.
For object recognition, use a Convolutional Net or an RNTN.
Finally, for speech recognition, use a Recurrent Net.
In general, Deep Belief Networks and Multilayer Perceptrons with rectified linear units – also known as RELU – are both good choices for classification. For time series analysis, it's best to use a Recurrent Net.
Deep Nets are the current state of the art in pattern recognition, but it's worth noting that neural nets have been around for decades. So you might be wondering: why did it take almost 50 years for Deep Nets to come on to the scene? Well, as it turns out, Deep Nets are very hard to train, which we will see in the next video.
End of transcript. Skip to the start.

AN OLD PROBLEM

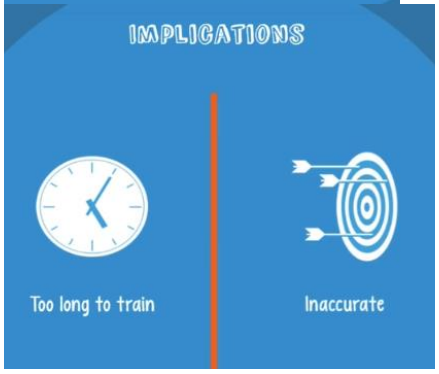

TRAINING
GRADIENT = RATE AT WHICH COST CHANGES WITH RESPECT TO WEIGHT OR BIAS
ISSUE = VANISHING GRADIENT

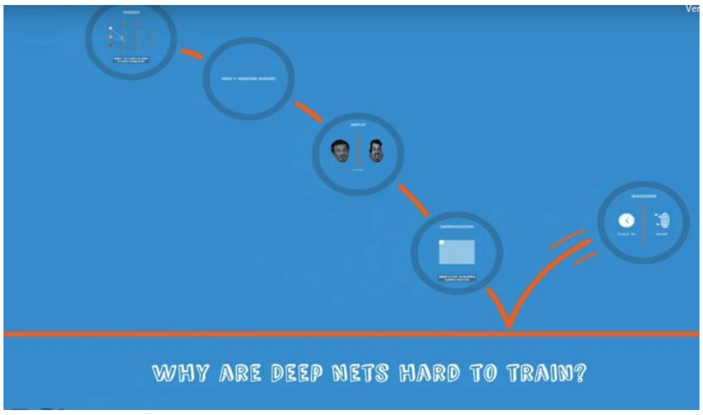






GRADIENT AT A LAYER = MULTIPLICATION OF GRADIENTS AT PRIOR LAYERS



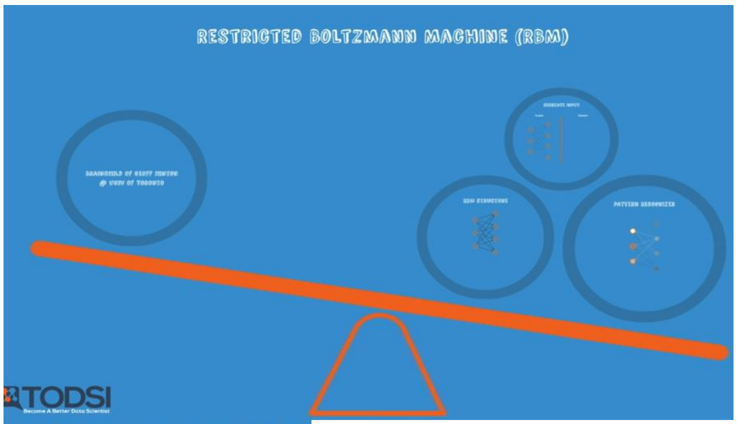


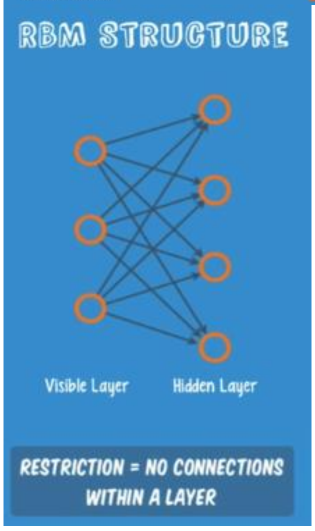
rt of transcript. Skip to the end.
So now you're probably thinking – wow, deep nets are really great! But why did it
take so long for them to become popular? Well as it turns out, when you try to train them
with a method called backpropagation, you run into a fundamental problem called the
vanishing gradient, or sometimes the exploding gradient. When that happens, training takes
too long and the accuracy really suffers. Let's take a closer look.
When you're training a neural net, you're constantly calculating a cost value. The cost
is typically the difference between the net's predicted output and the actual output from
a set of labelled training data. The cost is then lowered by making slight adjustments
to the weights and biases over and over throughout the training process, until the lowest
possible
value is obtained. Here is that forward prop again; and here are the example weights and
biases. The training process utilizes something called a gradient, which measures the rate
at which the cost will change with respect to a change in a weight or a bias.
Deep architectures are your best and sometimes your only choice for complex machine
learning
problems such as facial recognition. But up until 2006, there was no way to accurately
train deep nets due to a fundamental problem with the training process: the vanishing
gradient.
Let's think of a gradient like a slope, and the training process like a rock rolling
down that slope. A rock will roll quickly down a steep slope but will barely move at
all on a flat surface. The same is true with the gradient of a deep net. When the gradient
is large, the net will train quickly. When the gradient is small, the net will train
slowly. Here's that deep net again. And here is how the gradient could potentially vanish
or decay back through the net. As you can see, the gradients are much smaller in the
earlier layers. As a result, the early layers of the network are the slowest to train. But
this is a fundamental problem! The early layers are responsible for detecting the simple
patterns
and the building blocks – when it came to facial recognition, the early layers detected
the edges which were combined to form facial features later in the network. And if the
early layers get it wrong, the result built up by the net will be wrong as well. It could
mean that instead of a face like this, your net looks for this.
The process used for training a neural net is called back-propagation or back-prop. We
saw before that forward prop starts with the inputs and works forward; back-prop does the
reverse, calculating the gradient from right to left. For example, here are 5 gradients,
4 weight and 1 bias. It starts with the left and works back through the layers, like so.
Each time it calculates a gradient, it uses all the previous gradients up to that point.
So, lets start with that node. That edge uses the gradient at that node. And the next. So
far things are simple. As you keep going back, things get a bit more complex - that one for

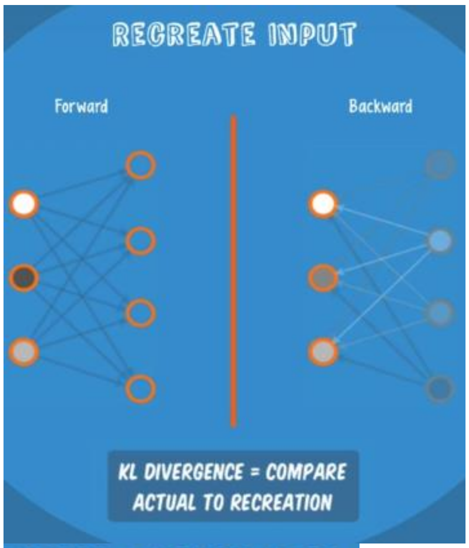
example uses a lot of gradients, even though this is a relatively simple net. If your net
gets larger and deeper, like this one, it gets even worse. But why is that? Well, a
gradient at any point is the product of the previous gradients up to that point. And the
product of two numbers between 0 and 1 gives you a smaller number. Say this rectangle is
a one. Also, say there are two gradients - a fourth - like that - and a third. If you multiply
them, you get a fourth of a third which is a twelfth. A fourth of a twelfth is a forty-eighth.
You can see that numbers keep getting smaller the more you multiply.
Have you ever had this issue while training a neural network with backpropagation? If
so, please comment and let me know your thoughts.
As a result of all this, backprop ends up taking a lot of time to train the net, and
the accuracy is often very low.
Up until 2006, deep nets were still underperforming shallow nets and other machine learning
algorithms.
But everything changed after three breakthrough papers published by Hinton, Lecun, and
Bengio
in 2006 and 2007. In the next video, we'll begin taking a closer look at these breakthroughs,
starting with the Restricted Boltzmann Machine.
End of transcript. Skip to the start

MODULE 2- DEEP LEARNING MODELS

RESTRICTED BOLZMANN MACHINESS (RBM)


RESTRICTED BOLTZMANN MACHINE (RBM)
BRAINCHILD OF GEOFF HINTON @ UNIV OF TORONTO
RBM STRUCTURE
PATTERN RECOGNIZER
TODSI



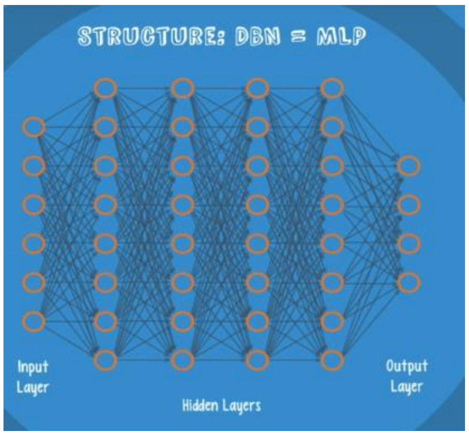


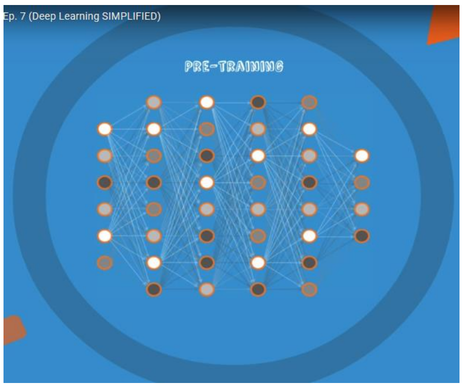
PATTERN RECOGNIZER
AUTOENCODER

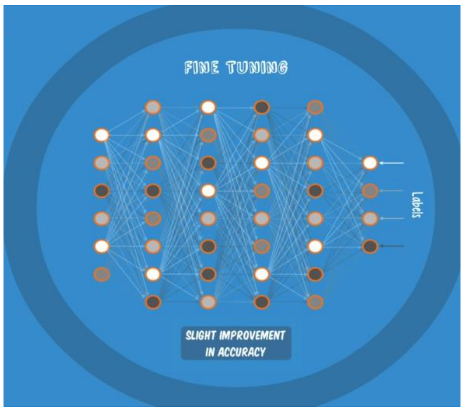
Start of transcript. Skip to the end.
So what was it that allowed researchers to overcome the vanishing gradient problem? The answer to this question has two parts, the first of which involves a Restricted Boltzmann Machine. This is a method that can automatically find patterns in our data by reconstructing the input. Sounds complicated, but bear with me. We'll take a closer look.
Geoff Hinton at the University of Toronto was one of the first researchers to devise a breakthrough idea for training deep nets.His approach led to the creation of the Restricted Boltzmann Machine, also known as the RBM. Because of his pioneering work he's often referred to as one of the father's of deep learning.
An RBM is a shallow, two-layer net; the first layer is known as the visible layer and the second is called the hidden layer. Each node in the visible layer is connected to every node in the hidden layer. An RBM is considered "restricted" because no two nodes in the same layer share a connection.
An RBM is the mathematical equivalent of a two-way translator – in the forward pass, an RBM takes the inputs and translates them into a set of numbers that encode the inputs. In the backward pass, it takes this set of numbers and translates them back to form the re-constructed inputs. A well-trained net will be able to perform the backwards translation with a high degree of accuracy. In both steps, the weights and biases have a very important role. They allow the RBM to decipher the interrelationships among the input features, and they also help
the RBM decide which input features are the most important when detecting patterns.

Through several forward and backward passes, an RBM is trained to reconstruct the input data. Three steps are repeated over and over through the training process:
a) With a forward pass, every input is combined with an individual weight and one overall bias, and the result is passed to the hidden layer which may or may not activate. Here's how it flows for the whole net. b) Next, in a backward pass, each activation is combined with an individual weight and an overall bias, and the result is passed to the visible layer for reconstruction. Here's how it flows back.
c) At the visible layer, the reconstruction is compared against the original input to determine the quality of the result.
RBMs use a measure called KL Divergence for step c); steps a) thru c) are repeated with varying weights and biases until the input and the re-construction are as close as possible.
Have you ever had to train an RBM in one of your own machine learning projects? Please comment and tell me about your experiences.
An interesting aspect of an RBM is that the data does not need to be labelled. This turns out to be very important for real-world data sets like photos, videos, voices, and sensor data – all of which tend to be unlabelled. Rather than having people manually label the data and introduce errors, an RBM automatically sorts through the data, and by properly adjusting
the weights and biases, an RBM is able to extract the important features and reconstruct the input. An important note is that an RBM is actually making decisions about which input features are important and how they should be combined to form patterns. In other words, an RBM is part of a family of feature extractor neural nets, which are all designed to recognize inherent patterns in data. These nets are also called autoencoders, because in a way, they have to encode their own structure.
So we saw that an RBM can extract features, but how does that help with the vanishing gradient? We'll get to the second part of our answer in the next video when we take a look at the Deep Belief Net.
End of transcript. Skip to the start.

DEEP BELIEF NETS (DBN)



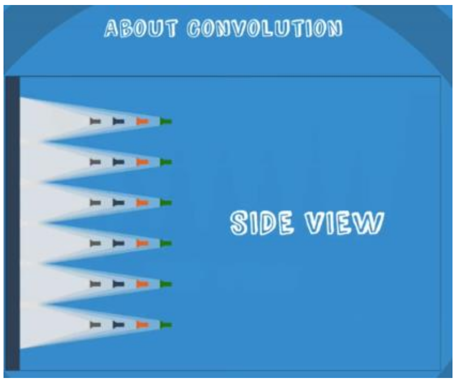


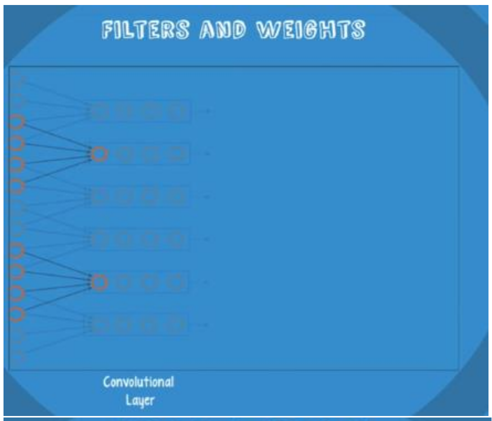


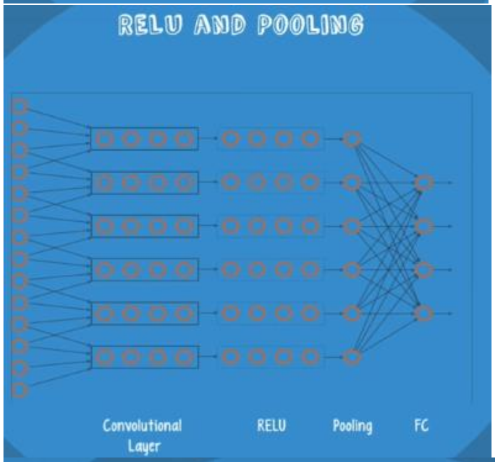
tart of transcript. Skip to the end.
Okay, so an RBM can extract features and reconstruct inputs…but how exactly does that help with
the vanishing gradient? By combining RBMs together and introducing a clever training method, we obtain a powerful new model that finally solves our problem. Let's now take a look at a Deep Belief Network.
Just like the RBM, Deep Belief Nets were also conceived by Geoff Hinton as an alternative to backpropagation. Because of his accomplishments, he was hired for image recognition work at
Google, where a large-scale DBN project is currently believed to be in development.
In terms of network structure, a DBN is identical to an MLP. But when it comes to training, they are entirely different. In fact, the difference in training methods is the key factor that enables DBNs to outperform their shallow counterparts.

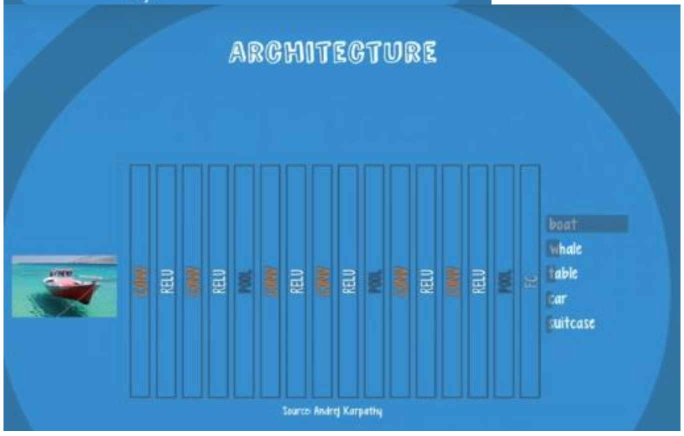
A deep belief network can be viewed as a stack of RBMs, where the hidden layer of one RBM is the visible layer of the one "above" it. A DBN is trained as follows:
a) The first RBM is trained to re-construct its input as accurately as possible
b) The hidden layer of the first RBM is treated as the visible layer for the second and the second RBM is trained using the outputs from the first RBM
c) This process is repeated until every layer in the network is trained
An important note about a DBN is that each RBM layer learns the entire input. In other kinds of models – like convolutional nets – early layers detect simple patterns and later layers recombine them. Like in our facial recognition example, the early layers would detect edges in the image, and later layers would use these results to form facial features. A DBN, on the other hand, works globally by fine tuning the entire input in succession as the model slowly improves – kind of like a camera lens slowly focusing a picture. The reason that a DBN works so well is highly technical, but suffice it to say that a stack of RBMs will outperform a single unit – just like a Multilayer perceptron was able to outperform
a single perceptron working alone.
After this initial training, the RBMs have created a model that can detect inherent patterns in the data. But we still don't know exactly what the patterns are called. To finish training, we need to introduce labels to the patterns and fine-tune the net with supervised learning. To do this, you need a very small set of labeled samples so that the features and patterns can be associated with a name. The weights and biases are altered slightly, resulting in a small change in the net's perception of the patterns, and often a small increase in the total accuracy. Fortunately, the set of labelled data can be small relative to the original data set, which as we've discussed, is extremely helpful in real-world applications.
Have you ever trained a Deep Belief Network before? If so, please comment and share your experiences.
So let's recap the benefits of a Deep Belief Network. We saw that a DBN only needs a small labelled data set, which is important for real-world applications. The training process can also be completed in a reasonable amount of time through the use of GPUs. And best of all, the resulting net will be very accurate compared to a shallow net, so we can finally see that a DBN is a solution to the vanishing gradient problem!
Now that we've covered the Deep Belief Net, we can begin to discuss a few other deep learning
models and see what kinds of problems they can solve. Let's jump over to the next video, and take a look at how a convolutional net could be trained to recognize different objects in an image.
End of transcript. Skip to the start.

CONVOLUTIONAL NETS (CNN)




FILTERS AND WEIGHTS
Convolutional Layer


RELU AND POOLING
Convolutional Layer
RELU
Pooling
FC


ARCHITECTURE
CONV
RELU
CONV
RELU
POOL
CONV
RELU
CONV
RELU
POOL
CONV
RELU
CONV
RELU
POOL
FC
boat
whale
table
car
suitcase
Source: Andrej Karpathy

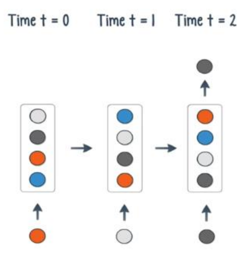
CONVOLUTIONAL NEURAL NET
Ver más ta...
BRAINCHILD OF YANN LECUN @NYU
ARCHITECTURE
RELU AND POOLING
ABOUT CONVOLUTION
FILTERS AND WEIGHTS

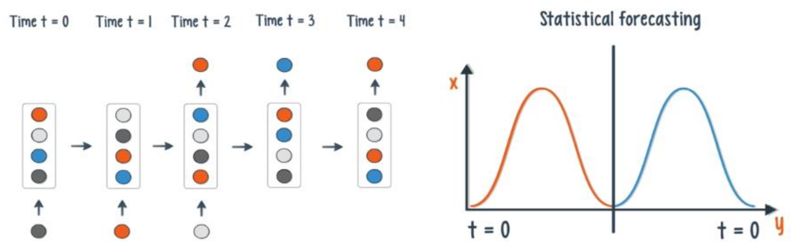
If there's one deep net that has completely dominated the machine vision space in recent years, it's certainly the convolutional neural net, or CNN. These nets are so influential that they've made Deep Learning one of the hottest topics in AI today. But they can be tricky to understand, so let's take a closer look and see how they work.
CNNs were pioneered by Yann Lecun of New York University, who also serves as the director of Facebook's AI group. It is currently believed that Facebook uses a CNN for its facial recognition
software.
A convolutional net has been the go to solution for machine vision projects in the last few years. Early in 2015, after a series of breakthroughs by Microsoft, Google, and Baidu, a machine
was able to beat a human at an object recognition challenge for the first time in the history of AI.
It's hard to mention a CNN without touching on the ImageNet challenge. ImageNet is a project that was inspired by the growing need for high-quality data in the image processing space. Every year, the top Deep Learning teams in the world compete with each other to create
the best possible object recognition software. Going back to 2012 when Geoff Hinton's team took first place in the challenge, every single winner has used a convolutional net as their model. This isn't surprising, since the error rate of image detection tasks has dropped significantly with CNNs, as seen in this image.
Have you ever struggled while trying to learn about CNNs? If so, please comment and share your experiences.
We'll keep our discussion of CNNs high level, but if you're inclined to learn about the math, be sure to check out Andrej Karpathy's amazing CS231n course notes on these nets.
There are many component layers to a CNN, and we will explain them one at a time. Let's start with an analogy that will help describe the first component, which is the "convolutional layer"
Imagine that we have a wall, which will represent a digital image. Also imagine that we have a series of flashlights shining at the wall, creating a group of overlapping circles. The purpose of these flashlights is to seek out a certain pattern in the image, like an edge or a color contrast for example. Each flashlight looks for the exact same pattern as all the others, but they all search in a different section of the image, defined by the fixed region created by the circle of light. When combined together, the flashlights form what's a called a filter. A filter is able to determine if the given pattern occurs in the image, and in what regions. What you see in this example is an 8x6 grid of lights, which is all considered to be one filter.
Now let's take a look from the top. In practice, flashlights from multiple different filters will all be shining at the same spots in parallel, simultaneously detecting a wide array of patterns.
In this example, we have four filters all shining at the wall, all looking for a different pattern. So this particular convolutional layer is an 8x6x4, 3-dimensionsal grid of these flashlights.
Now let's connect the dots of our explanation: - Why is it called a convolutional net? The net uses the technical operation of convolution to search for a particular pattern. While the exact definition of convolution is beyond the scope of this video, to keep things simple, just think of it as the process of filtering through the image for a specific pattern.
Although one important note is that the weights and biases of this layer affect how this operation
is performed: tweaking these numbers impacts the effectiveness of the filtering process.
- Each flashlight represents a neuron in the CNN. Typically, neurons in a layer activate or fire. On the other hand, in the convolutional layer, neurons perform this "convolution"

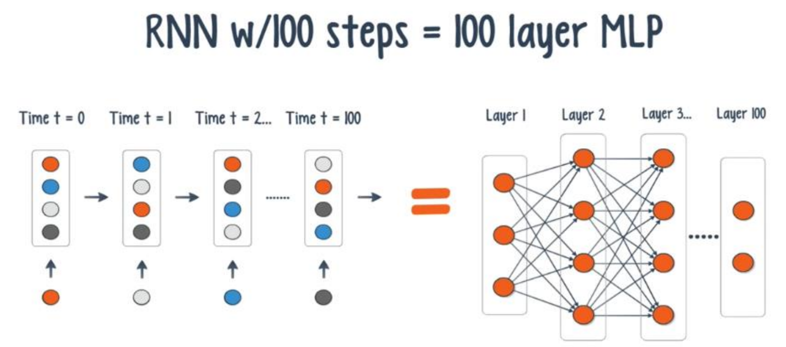
operation. We're going to draw a box around one set of flashlights to make things look
a bit more organized.
- Unlike the nets we've seen thus far where every neuron in a layer is connected to every
neuron in the adjacent layers, a CNN has the flashlight structure. Each neuron is only
connected to the input neurons it "shines" upon.
The neurons in a given filter share the same weight and bias parameters. This means that,
anywhere on the filter, a given neuron is connected to the same number of input neurons
and has the same weights and biases. This is what allows the filter to look for the
same pattern in different sections of the image. By arranging these neurons in the same
structure as the flashlight grid, we ensure that the entire image is scanned.
The next two layers that follow are RELU and pooling, both of which help to build up the
simple patterns discovered by the convolutional layer. Each node in the convolutional layer
is connected to a node that fires like in other nets. The activation used is called
RELU, or rectified linear unit. CNNs are trained using backpropagation, so the vanishing
gradient
is once again a potential issue. For reasons that depend on the mathematical definition
of RELU, the gradient is held more or less constant at every layer of the net. So the
RELU activation allows the net to be properly trained, without harmful slowdowns in the
crucial early layers.
The pooling layer is used for dimensionality reduction. CNNs tile multiple instances of
convolutional layers and RELU layers together in a sequence, in order to build more and
more complex patterns. The problem with this is that the number of possible patterns
becomes
exceedingly large. By introducing pooling layers, we ensure that the net focuses on
only the most relevant patterns discovered by convolution and RELU. This helps limit
both the memory and processing requirements for running a CNN.
Together, these three layers can discover a host of complex patterns, but the net will
have no understanding of what these patterns mean. So a fully connected layer is attached
to the end of the net in order to equip the net with the ability to classify data samples.
Let’s recap the major components of a CNN. A typical deep CNN has three sets of layers
– a convolutional layer, RELU, and pooling layers – all of which are repeated several
times. These layers are followed by a few fully connected layers in order to support
classification. Since CNNs are such deep nets, they most likely need to be trained using
server resources with GPUs.
Despite the power of CNNs, these nets have one drawback. Since they are a supervised
learning method, they require a large set of labelled data for training, which can be
challenging to obtain in a real-world application. In the next video, we’ll shift our attention
to another important deep learning model – the Recurrent Net.
End of transcript. Skip to the start.

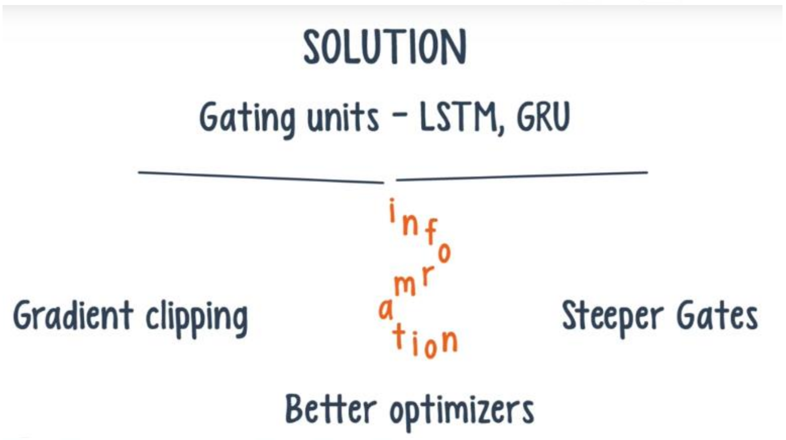
RECURRENT NETS (RNN)

Review
"It has been a while where I saw action combined with a decent story line..."
Market
"Oil prices were supposed to fall but not by this much. Low energy prices..."








SOLUTION

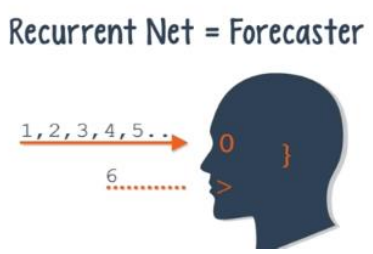
Gating units – LSTM, GRU

Gradient clipping

information

Steeper Gates

Better optimizers

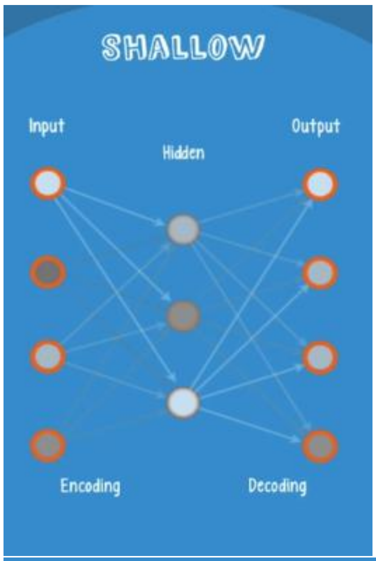


Start of transcript. Skip to the end.
What do you do if the patterns in your data change with time? In that case, your best bet is to use a recurrent neural network. This deep learning model has a simple structure with a built-in feedback loop, allowing it to act as a forecasting engine. Let's take a closer look.
]Recurrent neural networks, or RNNs, have a long history, but their recent popularity

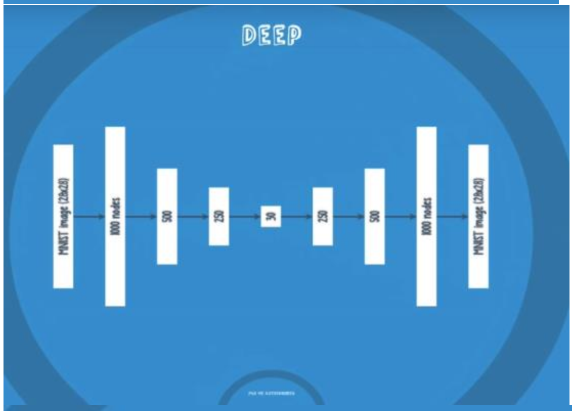
is mostly due to the works of Juergen Schmidhuber, Sepp Hochreiter, and Alex Graves. Their applications
are extremely versatile – ranging from speech recognition to driverless cars.
All the nets we've seen up to this point have been feedforward neural networks. In
a feedforward neural network, signals flow in only one direction from input to output,
one layer at a time. In a recurrent net, the output of a layer is added to the next input
and fed back into the same layer, which is typically the only layer in the entire network.
You can think of this process as a passage through time – shown here are 4 such time
steps. At t = 1, the net takes the output of time t = 0 and sends it back into the net
along with the next input. The net repeats this for t = 2, t = 3, and so on.
Unlike feedforward nets, a recurrent net can receive a sequence of values as input, and
it can also produce a sequence of values as output. The ability to operate with sequences
opens up these nets to a wide variety of applications. Here are a few examples. When the input is
singular and the output is a sequence, a potential application is image captioning. A sequence
of inputs with a single output can be used for document classification. When both the
input and output are sequences, these nets can classify videos frame by frame. If a time
delay is introduced, the net can statistically forecast the demand in supply chain planning.
Have you ever used an RNN for one of these applications? If so, please comment and share
your experiences.
Like we've seen with previous deep learning models, by stacking RNNs on top of each other,
you can form a net capable of more complex output than a single RNN working alone.
Typically, an RNN is an extremely difficult net to train. Since these nets use backpropagation,
we once again run into the problem of the vanishing gradient. Unfortunately, the vanishing
gradient is exponentially worse for an RNN. The reason for this is that each time step
is the equivalent of an entire layer in a feedforward network. So training an RNN for
100 time steps is like training a 100-layer feedforward net – this leads to exponentially
small gradients and a decay of information through time.
There are several ways to address this problem - the most popular of which is gating. Gating
is a technique that helps the net decide when to forget the current input, and when to remember
it for future time steps. The most popular gating types today are GRU and LSTM. Besides
gating, there are also a few other techniques like gradient clipping, steeper gates, and
better optimizers.
When it comes to training a recurrent net, GPUs are an obvious choice over an ordinary
CPU. This was validated by a research team at Indico, which uses these nets on text processing
tasks like sentiment analysis and helpfulness extraction. The team found that GPUs were
able to train the nets 250 times faster! That's the difference between one day of training,
and over eight months!
So under what circumstances would you use a recurrent net over a feedforward net? We
know that a feedforward net outputs one value, which in many cases was a class or a prediction.
A recurrent net is suited for time series data, where an output can be the next value
in a sequence, or the next several values. So the answer depends on whether the application
calls for classification, regression, or forecasting.
In the next video, we'll take a look at a family of deep learning models known as
the autoencoders.
End of transcript. Skip to the start.

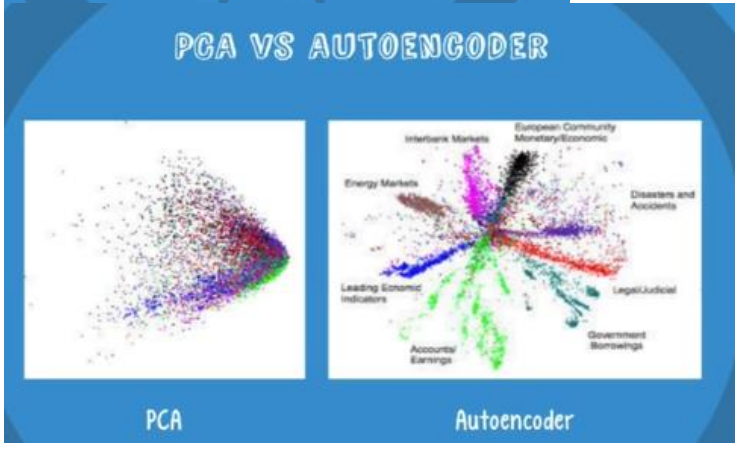
MODULE 3 – ADDITIONAL DEEP LEARNING MODELS

AUTOENCODERS



BACKPROPAGATION WITH LOSS

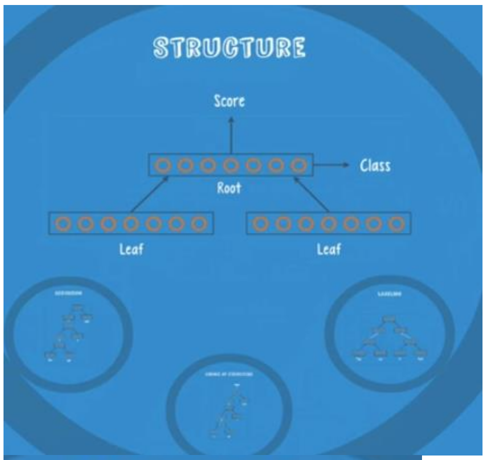


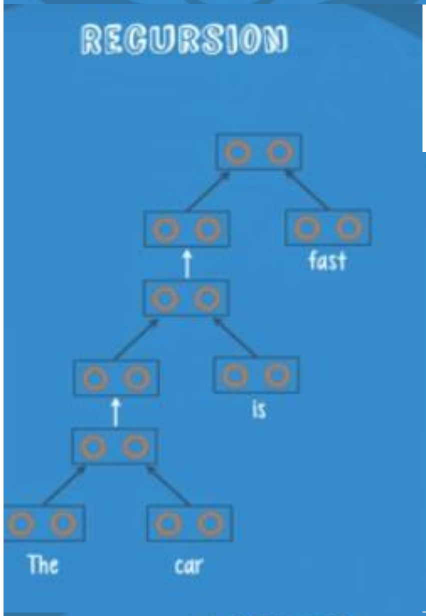


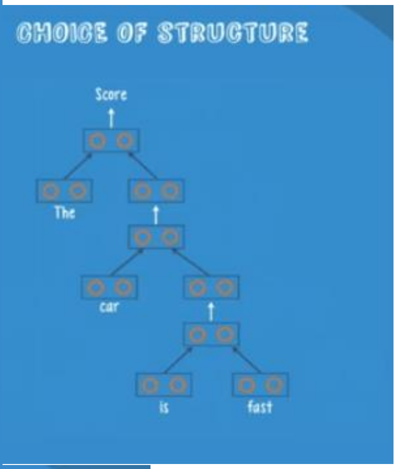
Start of transcript. Skip to the end.
There are times when it's extremely useful to figure out the underlying structure of a data set. Having access to the most important data features gives you a lot of flexibility when you start applying labels. Autoencoders are an important family of neural networks that are well-suited for this task. Let's take a look.
In a previous video we looked at the Restricted Boltzmann Machine, which is a very popular

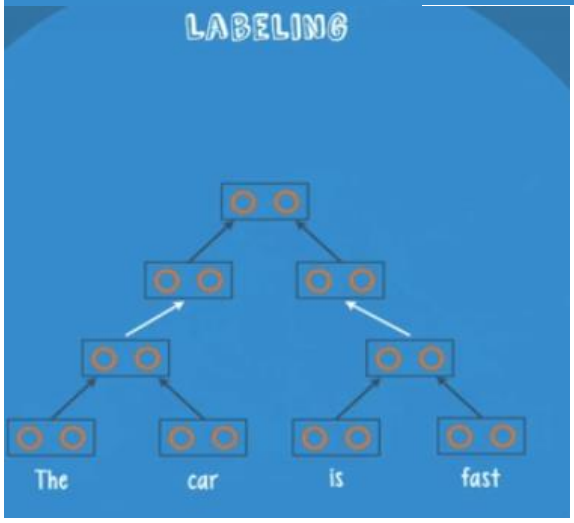
example of an autoencoder. But there are other types of autoencoders like denoising and contractive,
just to name a few. Just like an RBM, an autoencoder is a neural net that takes a set of typically unlabelled inputs, and after encoding them, tries to reconstruct them as accurately as possible. As a result of this, the net must decide which of the data features are the most important, essentially acting as a feature extraction engine.
Autoencoders are typically very shallow, and are usually comprised of an input layer, an output layer and a hidden layer. An RBM is an example of an autoencoder with only two layers. Here is a forward pass that ends with a reconstruction of the input. There are two steps - the encoding and the decoding. Typically, the same weights that are used to encode a feature in the hidden layer are used to reconstruct an image in the output layer.
Autoencoders are trained with backpropagation, using a metric called "loss". As opposed to "cost", loss measures the amount of information that was lost when the net tried to reconstruct the input. A net with a small loss value will produce reconstructions that look very similar to the originals.
Not all of these nets are shallow. In fact, deep autoencoders are extremely useful tools for dimensionality reduction. Consider an image containing a 28x28 grid of pixels. A neural net would need to process over 750 input values just for one image – doing this across millions of images would waste significant amounts of memory and processing time. A deep autoencoder could encode this image into an impressive 30 numbers, and still maintain information about the key image features. When decoding the output, the net acts like
a two-way translator. In this example, a well-trained net could translate these 30 encoded numbers
back into a reconstruction that looks similar to the original image. Certain types of nets also introduce random noise to the encoding-decoding process, which has been shown to improve the
robustness of the resulting patterns.
Have you ever needed to use an autoencoder to reduce the dimensionality of your data?
If so, please comment and share your experiences.
Deep autoencoders perform better at dimensionality reduction than their predecessor, principal
component analysis, or PCA. Below is a comparison of two letter codes for news stories of different
topics – generated by both a deep autoencoder and a PCA. Labels were added to the picture for illustrative purposes.
In the next video, we'll take a look at Recursive Neural Tensor Nets or RNTNs
End of transcript. Skip to the start.

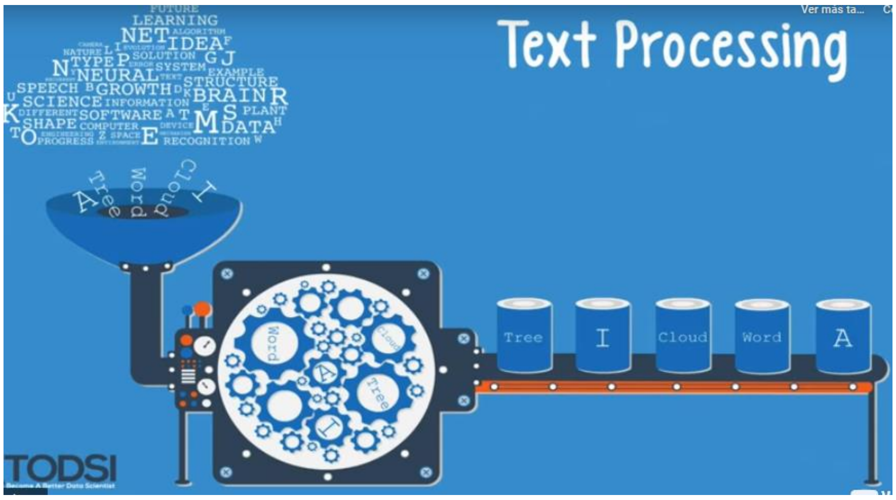
RECURSIVE NEURAL TENSOR NETS (RNTN)







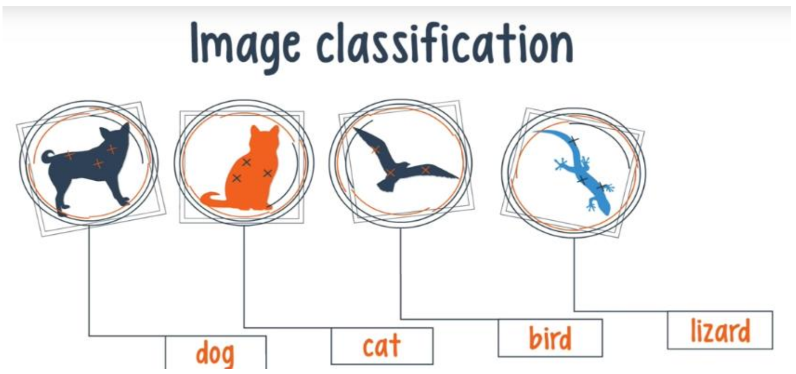


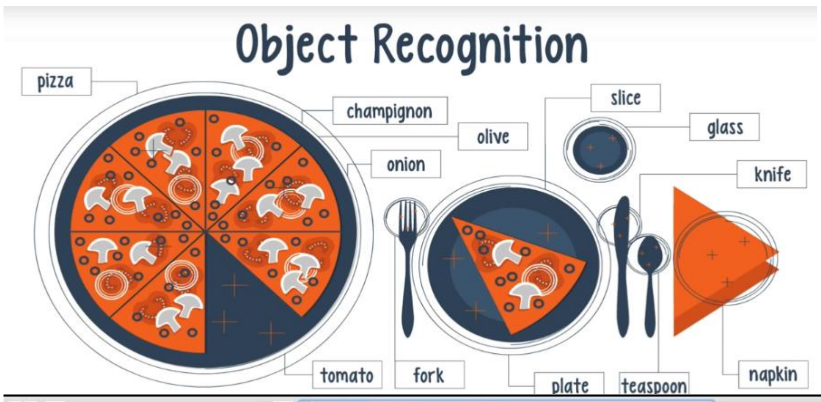


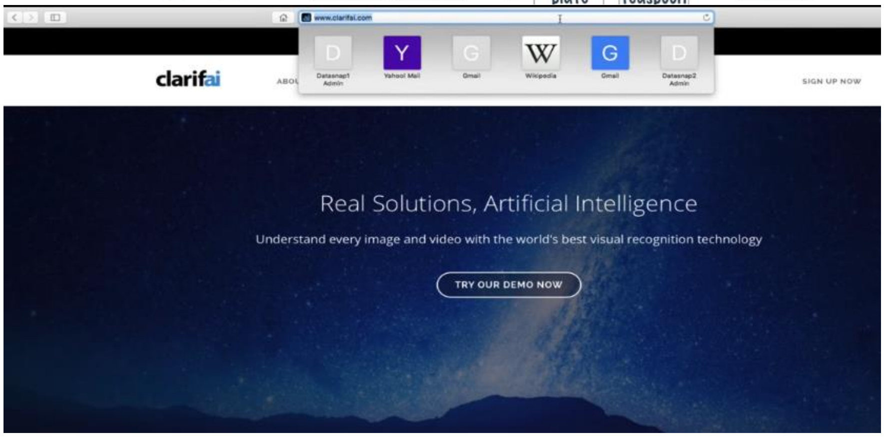
Start of transcript. Skip to the end.
Sometimes it’s useful to discover the hierarchical structure of a set of data, such as the parse trees of a group of sentences. In these cases, Recursive Neural Tensor Networks, or RNTNs, are a better fit than feedforward or recurrent nets. Let’s take a closer look and see why.
RNTNs were conceived by Richard Socher of MetaMind as part of his PhD thesis at Stanford.
The purpose of these nets was to analyze data that had a hierarchical structure. Originally, they were designed for sentiment analysis, where the sentiment of a sentence depends not just on its component words, but on the order in which they’re syntactically grouped.
So let’s take a look at the structure of an RNTN. An RNTN has three basic components – a parent group, which we’ll call the root, and the child groups, which we’ll call the leaves. Each group is simply a collection of neurons, where the number of neurons depends
on the complexity of the input data. As you can see, the root is connected to both leaves, but the leaves are not connected to each other. Technically speaking, the three components form what’s called a binary tree. In general, the leaf groups receive input, and the root group uses a classifier to fire out a class and a score. We’ll get to the significance of these two values in a moment. An RNTN’s structure may seem simple, but just like a recurrent net, the complexity comes from the way in which data moves throughout the network.
In the case of an RNTN, this process is recursive.
To see how this recursion works, let’s take a look at an example. Let’s feed an English sentence into the net, and receive the sentence’s parse tree as output. At step one, we feed the first two words into leaf groups one and two, respectively. As a practical note, the leaf groups do not actually receive the words per say, but rather vector representations of the words. A vector is just an ordered set of numbers, and it’s been shown that these nets work best with very specific vector representations – particularly, good results are achieved when the numbers in the two vectors encode the similarities between the two words,
when compared to other words in the vocabulary. The exact details of this process are beyond the scope of this video.
The two vectors move across the net to the root, which processes them and fires out two values – the class and the score. The score represents the quality of the current parse, and the class represents an encoding of a structure in the current parse. This is the point where the net starts the recursion. At the next step, the first leaf group now receives the current parse, rather than a single word. The second leaf group receives the next word in the sentence. At this point, the root group would output the score of a parse that is three words long. This continues until all the inputs are used up, and the

net has a parse tree with every single word included.
This simplified example illustrates the main idea behind an RNTN; but in a practical application, we typically encounter more complex recursive processes. Rather than use the next word in the sentence for the second leaf group, an RNTN would try all of the next words, and eventually, vectors that represent entire sub-parses. By doing this at every step of the recursive process, the net is able to analyze – and score – every possible syntactic parse.
Have you ever had to work with data where the underlying patterns were hierarchical? Please comment and let us know what you learned.
Shown here are three possible parse trees for the same sentence. To pick the best one, the net relies on the score value produced by the root group. By using this score to select the best substructure at each step of the recursive process, the net will produce the highest-scoring parse as its final output.
Once the net has the final structure, it backtracks through the parse tree in order to figure out the right grammatical label for each part of the sentence. Here, it does that one first and labels it as a noun phrase. Then it works on this, and you get a verb phrase. It then works its way up, and when it reaches the top, it adds a special label that signifies the beginning of the parse structure.
RNTNs are trained with backpropagation by comparing the predicted sentence structure with the proper sentence structure obtained from a set of labelled training data. Once trained, the net will give a higher score to structures that are more similar to the parse trees that it saw in training.
RNTNs are used in natural language processing for both syntactic parsing and sentiment analysis.
They are also used to parse images, typically when an image contains a scene with many different
components.
In the next video, we'll take a closer look at the many applications of Deep Learning.
End of transcript. Skip to the start.

USES CASES

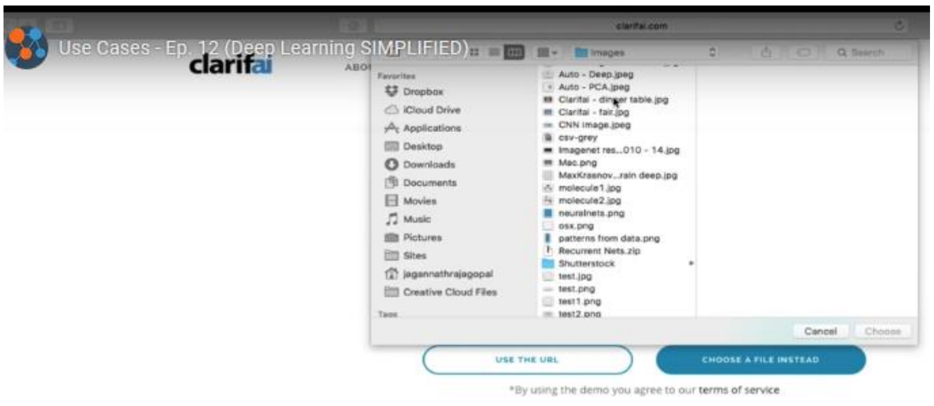




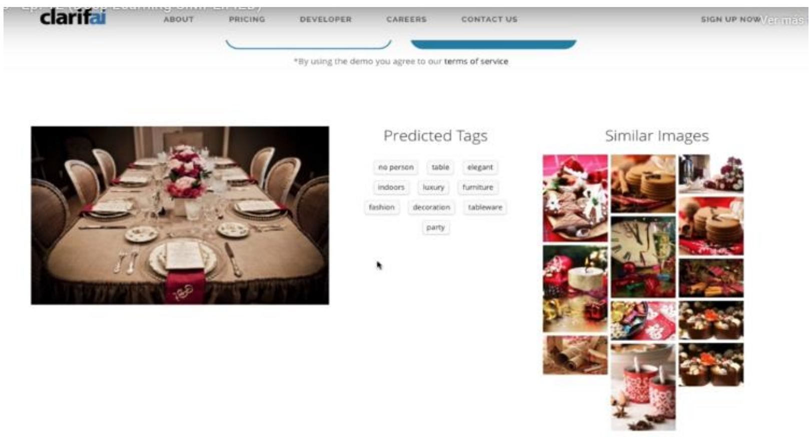


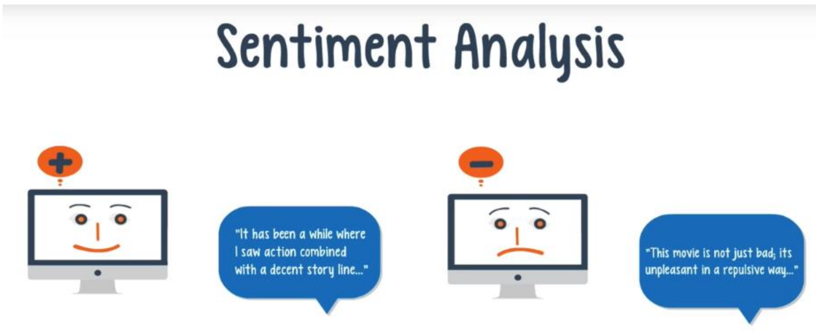
Real Solutions, Artificial Intelligence

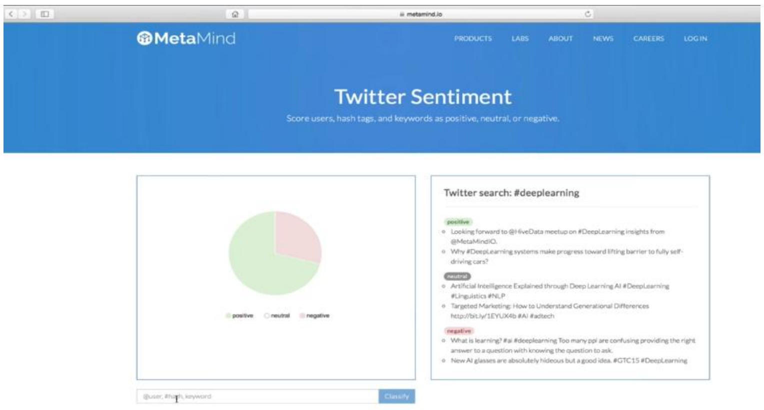
Understand every image and video with the world's best visual recognition technology



*By using the demo you agree to our terms of service




Sentiment Analysis
"It has been a while where I saw action combined with a decent story line..."
"This movie is not just bad; its unpleasant in a repulsive way..."



Twitter Sentiment

Score users, hash tags, and keywords as positive, neutral, or negative.

Twitter search: #deeplearning

positive

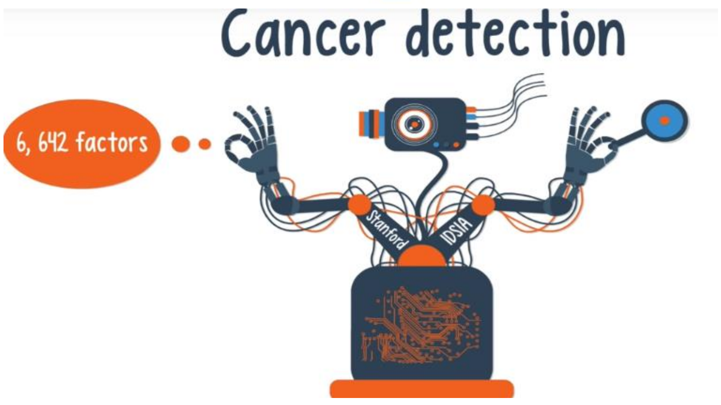
- Looking forward to @HiveData meetup on #DeepLearning insights from @MetaMindIO.
- Why #DeepLearning systems make progress toward lifting barrier to fully self-driving cars?

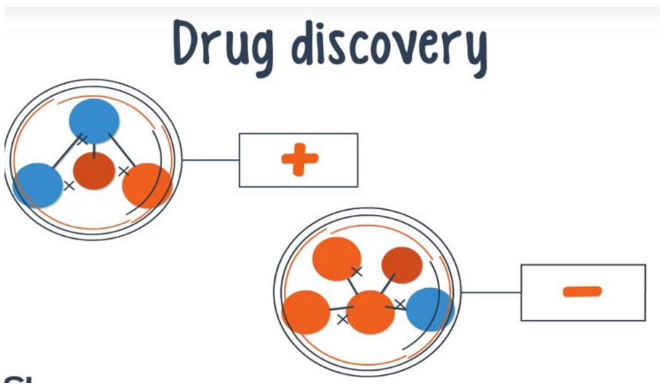
neutral

- Artificial Intelligence Explained through Deep Learning AI #DeepLearning #Linguistics #NLP
- Targeted Marketing: How to Understand Generational Differences http://bit.ly/1EYUX4b #AI #adtech

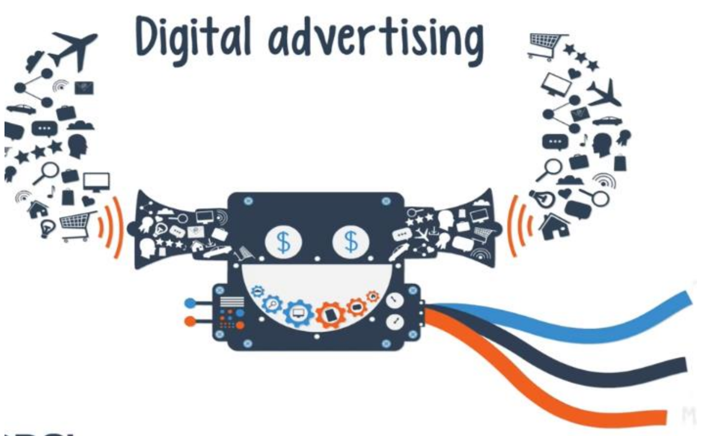
negative

- What is learning? #ai #deeplearning Too many ppl are confusing providing the right answer to a question with knowing the question to ask.
- New AI glasses are absolutely hideous but a good idea. #GTC15 #DeepLearning




Drug discovery
+
−

Finance
DWJI 17,499.10
SP500 2,025.51
NASDAQ 4,976.9
AAPL 107.71
GOOG 750.06
TSLA 234.24
SELL
SELL
BUY
AAPL
GOOG
TSLA


Digital advertising

Fraud detection

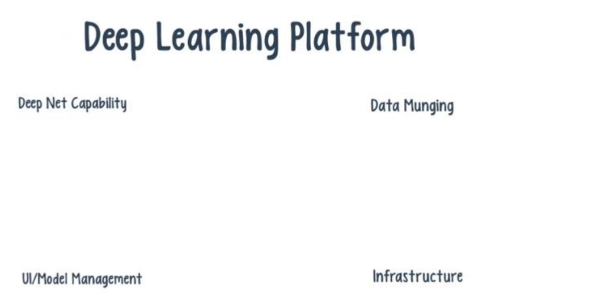
Customer intel
UPSELL
CROSS-SELL

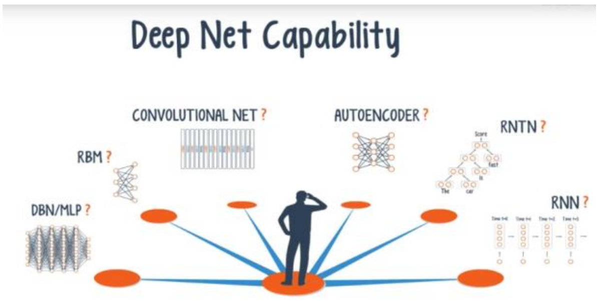
of transcript. Skip to the end.
There are so many important use cases for Deep Learning, that it's impossible to produce an exhaustive list. Deep Learning is just getting started, and new applications pop up all the time. Let's take a look at some of the biggest ones today.
At this point, it should be no surprise that machine vision is one of the biggest applications of deep learning. Image search systems use deep learning for image classification and automatic tagging, which allows the images to be accessible through a standard search query. Companies like Facebook use deep nets to scan pictures for faces at many different angles, and then label the face with the proper name. Deep nets are also used to recognize objects within images which allow for images to be searchable based on the objects within them. Let's look at an example application – Clarifai.
Let's load Clarifai in a browser. Here is the URL, which you'll also find in the video description below. Clarifai is an app that uses a convolutional net to recognize things and concepts in a digital image. Let's take a look. Right in the middle of the page you have the demo button. Lets click that.
It takes you to part of the webpage where you have the demo. You have two choices - a) either choose a URL where the image is located, or b) load the digital image yourselves if you have it on file. I'm going with choice b) - loading an image; I am in the right folder now and am going to select the first one.
When I select an image, it wants me to go through a verification process. In this case, it wants me to select all the squares that have a gift box, so I'm gonna go through and do that. This changes every time btw - you can have different tests.
Its come back and you see the predictions. Firstly, it says there's no person, it expected to find a person but there weren't any so it identified that as a pattern for this one, which is cool! The other predictions are "tableware", "indoors", "party", "fashion" etc. So this is the list of tags its associated with this image.
If you scroll down, it shows a list of example images and the items in them. Like the first one with a coffee and croissant, which I think is cool. If you go to the one with the concert, its tagged it pretty accurately with "concert", "band", "singer" etc. You also get similar images.
I'm going to pick another one, this time of a county fair. Again it goes through the same verification process - this time it wants me to pick images with cars. Ok - it came back and gave me some tags. It recognized a Ferris wheel, and though carousel is only partly visible to the left, it still picked it out! It also picked out the word "fun".
Also, the images it suggested as similar are accurate - they are virtually identical to the one I picked. Further, it presents the same example images as the last time.
So there you have it, a demo of object recognition using Clarifai.
Other uses of deep learning include image and video parsing. Video recognition systems are important tools for driverless cars, remote robots, and theft detection. And while not exactly a part of machine vision, the speech recognition field got a powerful boost from the introduction of deep nets.
Deep Net parsers can be used to extract relations and facts from text, as well as automatically translate text to other languages. These nets are extremely useful in sentiment analysis applications, and can be used as part of movie ratings and new product intros. Here is a quick demo of Metamind - an RNTN that performs sentiment analysis.
Let's load Metamind in a browser. Here is the URL, which you'll also find in the video description below. Metamind is an app by Richard Socher that uses an RNTN for twitter sentiment,
amongst other things.
You can search by user name, or keyword or hashtag. I'm going to search by hash tag.
My first one's #coffee.

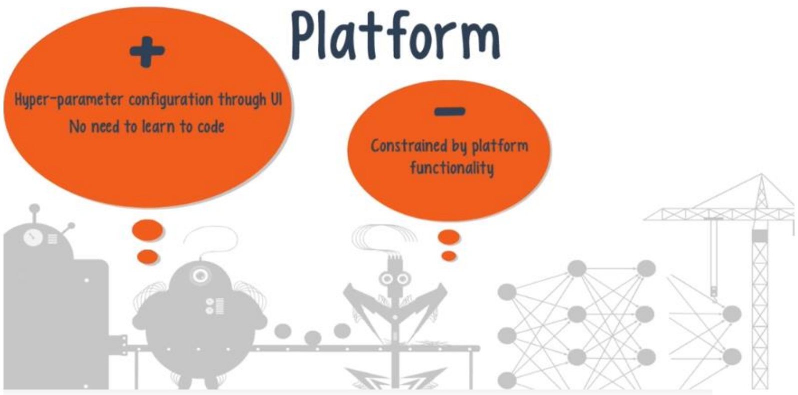
When you click "Classify", it first downloads the tweets which takes a little time. It then comes back and displays you two things. On the left, it shows you a pie chart of the 3 different sentiments - positive, negative and neutral. For most searches, you'll get lots of neutral comments which is natural, but here you have more positive comments than negative - 206 vs 41, which I think is good :-)
On the right, it also lists some example comments classified as positive, neutral and negative.
Let's search a different one - #holidays. Not surprisingly, you find a ton more positive comments about the holidays. In this case, if you look at the example, even the negative ones are light-hearted.
So there you have it, a demo of twitter sentiment analysis using Metamind.
Even recurrent nets have found uses in character-level text processing and document classification.
Deep nets are now beginning to thrive in the medical field. A Stanford team used deep learning
to identify over 6000 factors that help predict the chances of a cancer patient surviving.
Researchers from IDSIA in Switzerland created a deep net model to identify invasive breast cancer cells. Beyond this, deep nets are even used for drug discovery. In 2012, Merck hosted the Molecular Activity challenge on Kaggle in order to predict the biological activities of different drug molecules based solely on chemical structure. As a brief mention, this challenge was won by George Dahl of the University of Toronto, who led a team by the name of
'gggg'. But one crucial application of deep nets is radiology. Convolutional nets can help detect anomalies like tumors and cancers through the use of data from MRI, fMRI, EKG, and CT scans.
In the field of finance, deep nets can help make buy and sell predictions based on market data streams, portfolio allocations, and risk profiles. Depending on how they're trained, they're useful for both short term trading and long term investing. In digital advertising, deep nets are used to segment users by purchase history in order to offer relevant and personalized
ads in real time. Based on historical ad price data and other factors, deep nets can learn to optimally bid for ad space on a given web page. In fraud detection, deep nets use multiple data sources to flag a transaction as fraudulent in real time. They can also determine which products and markets are typically the most susceptible to fraud. In marketing and sales, deep nets are used to gather and analyze customer information, in order to determine the best
upselling strategies. In agriculture, deep nets use satellite feeds and sensor data to identify problematic environmental conditions.
Which of these deep learning applications appeals to you the most? Please comment and share your thoughts.
In the next video, we'll take a look at the main ideas behind a Deep Learning Platform.
End of transcript. Skip to the start.

MODULE 4 – DEEP LEARNING PLATFORMS AND LIBRARIES

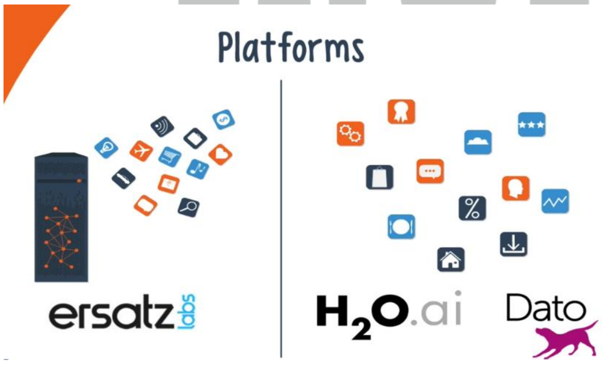
DEEP NET PLATAFORM






Platform
+
Hyper-parameter configuration through UI
No need to learn to code
−
Constrained by platform functionality

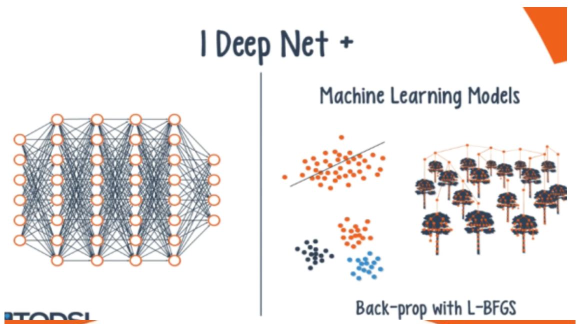
Library
+
Increased flexibility and options
−
Need to know code

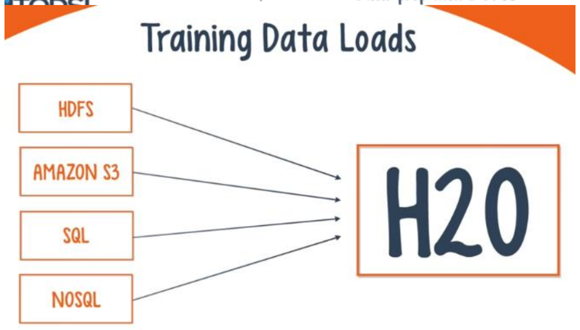


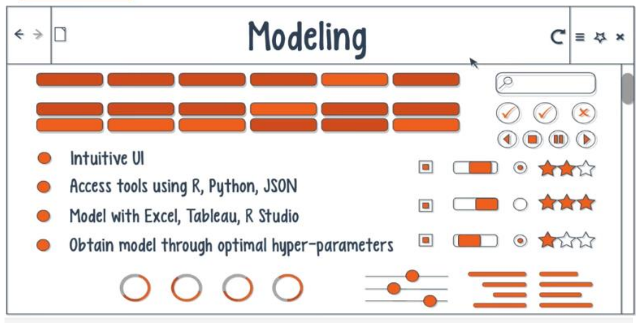
art of transcript. Skip to the end.

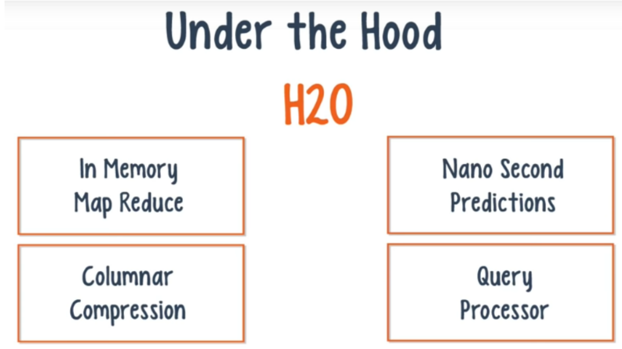
At this point, you're probably eager to use a deep net in one of your own applications.
If that's the case, let me introduce you to a few platforms that allow you to utilize
these nets without the hassle of building one yourself. Deep Learning platforms come
in two different forms – software platforms and full platforms. Let's see how they work.
A platform is a set of tools that other people can build on top of. For example, think of
the applications that can be built off of the tools provided by iOS and Android, Windows
and MacOS, IBM Websphere, and even Oracle BEA.
So a Deep Learning platform provides a set of tools and an interface for building custom
deep nets. Typically, they provide a user with a selection of deep nets to choose from,
along with the ability to integrate data from different sources, manipulate data, and manage
models through a UI. Some platforms also help with performance if a net needs to be trained
with a large data set.
Have you ever used a Deep Learning platform in one of your own projects? Please comment
and share your experiences.
Later in the series, we will introduce you to software libraries that will help you code
your own deep nets. There are some advantages and disadvantages of using a platform vs.
using a software library. A platform is an out-of-the-box application that lets you configure
a deep net's hyper-parameters through an intuitive UI; with a platform, you don't
need to know anything about coding in order to use the tools. The downside is that you
are constrained by the platform's selection of deep nets as well as the configuration
options. But for anyone looking to quickly deploy a deep net, a platform is the best
way to go.
A software library is a set of functions and modules that you can call through your own
code in order to perform certain tasks. Deep net libraries give you a lot of extra flexibility
with net selection and hyper-parameter configuration. For example, there aren't many
platforms
that let you build a Recursive Neural Tensor Net, but you can code your own with the right
deep net library! The obvious downside to libraries is the coding experience required
to use them, but if you need the flexibility, they really are a great resource.
Soon we'll take a look at Ersatz Labs, a dedicated Deep Learning platform that handles
all the technical issues like code, deployments, and performance – and allows the user to
go straight to modelling. We'll also look at two machine learning software platforms
called H2O, and GraphLab Create, both of which offer Deep Learning tools. Note that since
the latter two are software platforms and not full platforms like Ersatz, they would
need to be installed on your own personal hardware.
Coming up next, we'll first see an overview of Ersatz Labs.
End of transcript. Skip to the start.

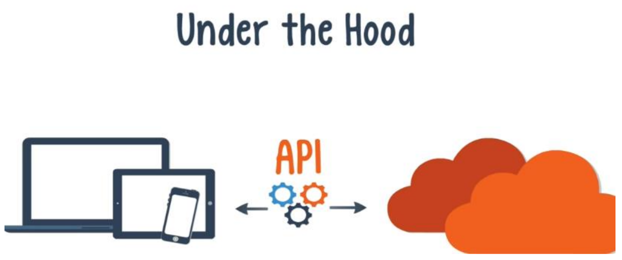
H2O.ai






Modeling
Intuitive UI
Access tools using R, Python, JSON
Model with Excel, Tableau, R Studio
Obtain model through optimal hyper-parameters

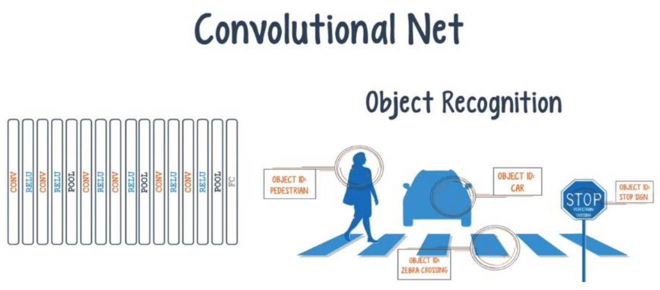


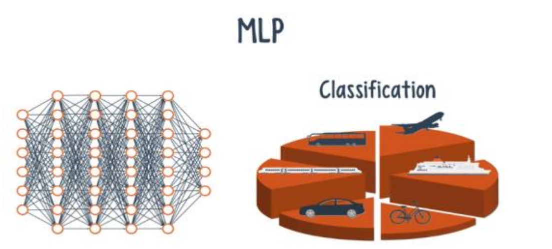


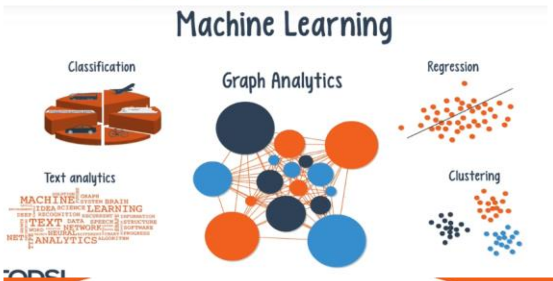
If you need to run different learning models alongside a deep net – perhaps as an ensemble – then H2O.ai might be a good choice. This software platform offers one deep net – the multilayer perceptron – and a few other machine learning algorithms. Let's take a look.

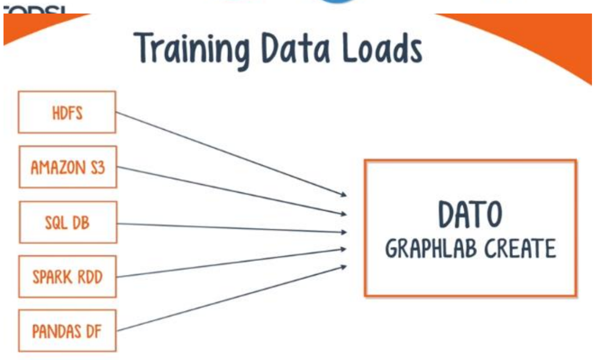
H2O started out as an open-source machine learning platform, with deep nets being a recent addition. Besides a set of machine learning algorithms, the platform offers several useful features, such as data pre-processing.
Currently, the only deep net supported by H2O is the multilayer perceptron. Despite this, the platform has sophisticated data munging capabilities as well as an intuitive model management UI. The other supported machine learning models include a GLM, a Distributed
Random Forest, K-Means clustering, a gradient boosting machine that uses an ensemble of decision trees, a cox proportional hazard, and a Naïve Bayes classifier. Training with backpropagation is performed by means of the L-BFGS algorithm.
Thankfully, H2O comes with built-in integration tools for platforms like HDFS, Amazon S3, SQL, and NoSQL.
While the platform has an intuitive UI, you can access the tools through a familiar programming
environment like R, Python, JSON, and several others. You can even model and analyze data with Tableau, Microsoft Excel, and R Studio. H2O also offers ensemble training, which allows you to obtain a model with the optimal set of hyper-parameters.
Unlike Ersatz, H2O is provided as a downloadable software package, which you'll need to deploy
and manage on your own hardware infrastructure. To help with this, the platform provides an in-memory map-reduce capability, distributed parallel processing, and columnar compression. These additions should help speed up your training significantly. The creators' hope was that search and analytics would be interactive, but this does not apply to training; instead, they hoped that the trained nets could make predictions about new data on the fly, perhaps even in nano-seconds. As one final note, it's not clear whether or not GPU support is built-in to the platform at this time. Although, you can access your deep net programmatically through an API.
H2O offers a lot of interesting features, but the website can be a bit confusing to navigate. There are a lot of different options and it may take some time to figure out how to use the platform to meet your needs. Still, if your interested in building your own deep net, it's worth checking out.
If you have any experience using the H2O.ai platform, please comment and share your experiences.
We'd love to hear from you.
Next up, we'll take a look at Dato GraphLab Create.
End of transcript. Skip to the start.

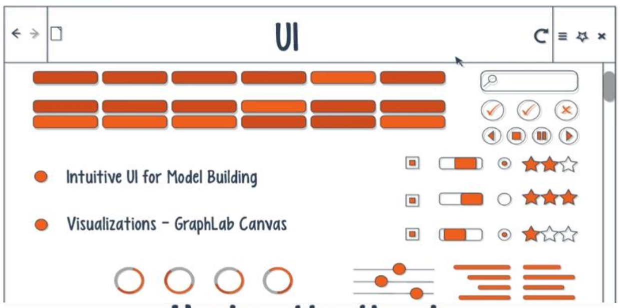
DATO GRAPHLAB

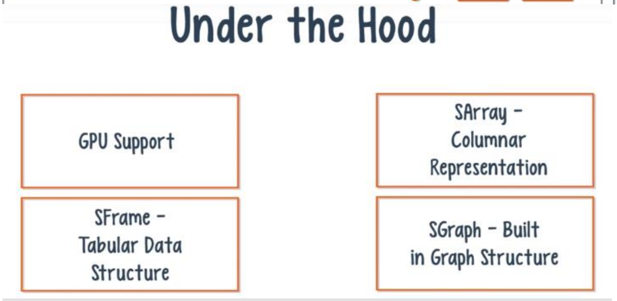


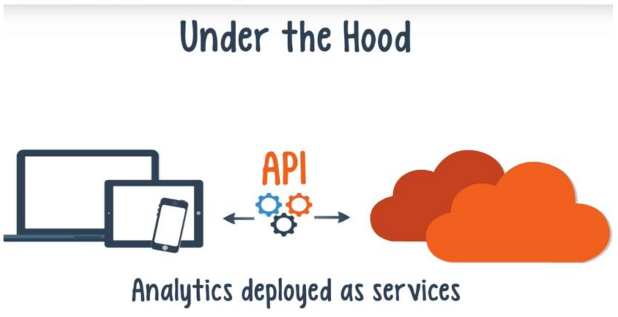

MLP
Classification


Machine Learning
Classification
Graph Analytics
Regression
Text analytics
Clustering


Training Data Loads
HDFS
AMAZON S3
SQL DB
SPARK RDD
PANDAS DF
DATO
GRAPHLAB CREATE

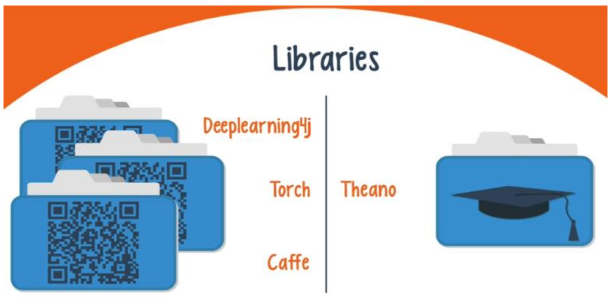

UI
Intuitive UI for Model Building
Visualizations – GraphLab Canvas


Under the Hood
GPU Support
SArray – Columnar Representation
SFrame – Tabular Data Structure
SGraph – Built in Graph Structure


Under the Hood
API
Analytics deployed as services

If your deep learning project requires graph analytics and other important algorithms,
then Dato GraphLab Create might be a good choice. GraphLab is a software platform that
offers two deep nets and a whole host of machine learning and graph algorithms. Let's take
a closer look.
GraphLab offers you two different types of deep nets, depending on the nature of your
input data. If you supply GraphLab with image data, the default selection will be a
convolutional
net. With any other kind of data, the default is a multilayer perceptron. In addition to
deep nets, the platform has several built-in machine learning algorithms including text
analytics, a recommender, classification, regression, and clustering. And as the name
suggests, they also provide Graph Analytics tools, which is unique among deep net platforms.
Under what circumstances would you use a graph in your deep learning projects? Please
comment
and let me know your thoughts.
Just like the H2O platform, GraphLab provides a great set of data munging features. It provides
built-in integration for SQL databases, Hadoop, Spark, Amazon S3, Pandas data frames, and
many others.
GraphLab also offers an intuitive UI for model management. In addition to this, GraphLab
Canvas allows you to create sophisticated visualizations of your model's results.
GraphLab Create is provided as a downloadable software package, so it will need to be
deployed
and managed on your own hardware infrastructure.
GraphLab offers three different types of built-in storage, all of which are open source. There
is SArray, which is a columnar representation. Then there is SFrame, a tabular storage model.
And finally there is SGraph, the graph model. According to the platform's website, these
tools are designed to handle terabytes of data analysis at interactive speeds.
An important note is that the GraphLab Create platform supports the use of GPUs. As we've
previously seen, this feature is becoming increasingly important for Deep Learning
applications.
You can use the GraphLab models to build different types of predictive analytics tools, which
can then be set up as services. These services can be accessed programmatically through an
API on your computer or mobile device.
The information on GraphLab's website is a bit limited, so you may have to sift through
the API's documentation in order to see if the platform is a good fit for your project.
Up next, we'll start looking into Deep Learning Libraries.
End of transcript. Skip to the start.

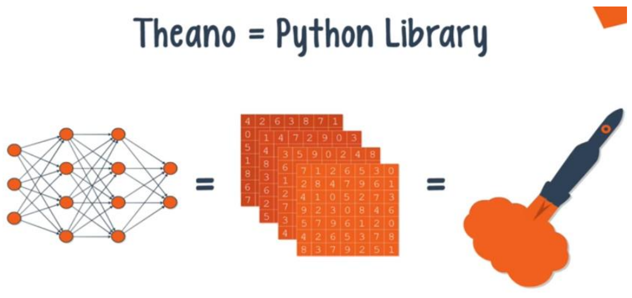
DEEP LEARINING LIBRARY

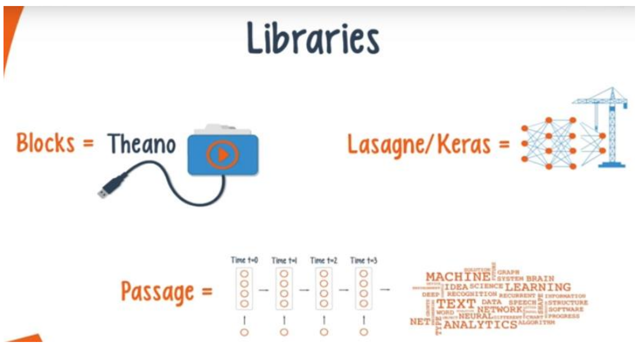


If you're coding a deep neural network, using a Deep Learning software library is
a sure-fire way to simplify the development process. Rather than re-invent the wheel,

you can take advantage of well-tested code that was created by experts in the field.
Let’s take a closer look.
If you aren’t a developer or if you’re unfamiliar with the term, a library is a premade
set of functions and modules that you can call through your own programs. Libraries
are typically created by highly-qualified software teams, and popular libraries are
regularly maintained. Many libraries are open-source, and are surrounded by big communities that
provide support and contribute to the codebase. Deep Learning has plenty of great libraries
available, several of which were created by key people in the field.
Have you ever tried to code your own deep net? Did you use a library to help simplify
the process? Please comment and share your thoughts.
If you’re building a commercial app that requires the use of a deep net, your best
bet is to use a commercial-grade library like deeplearning4j, Torch, or Caffe. For educational
or scientific projects, you should use a library like Theano. Another notable library is deepmat,
and there are many others.
In the next video, we’ll start by looking at the Theano library.
End of transcript. Skip to the start.

THEANO

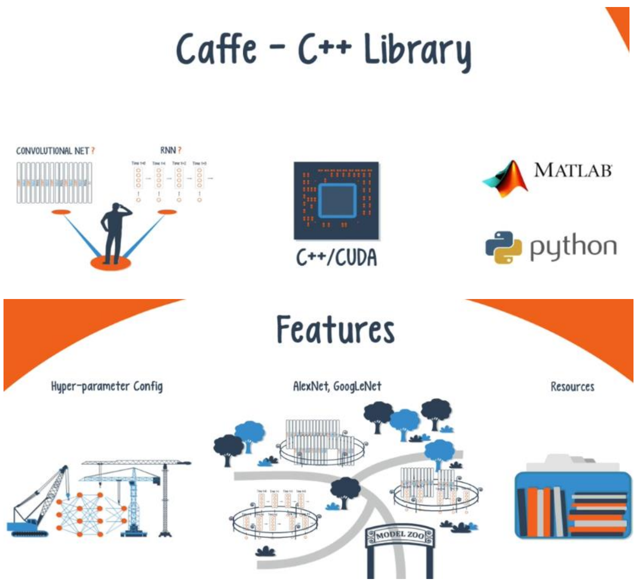


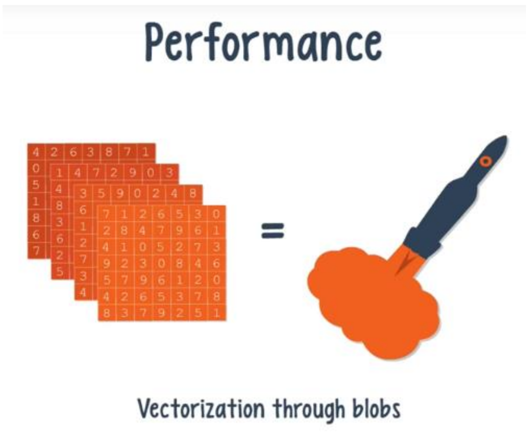


If you’re ready to start coding your own deep net, then you should take a look at the
Theano library. Theano provides an important set of functions for building deep nets that
will train quickly on your machine. Let’s take a look at what the library offers.
Theano was created by the Machine Learning group at the University of Montreal. The head
of the group, Yoshua Bengio, is one of the pioneers behind Deep Nets.
Theano is a Python library that lets you define and evaluate mathematical expressions with
vectors and matrices, which are rectangular arrays of numbers. Technically speaking, both
neural nets and input data can be represented as matrices, and all the standard net operations
can be redefined as matrix calculations. This is extremely important since computers can
perform matrix operations very quickly, especially when you use a library like Theano to process
multiple matrix values in parallel. So if you build a neural net with this underlying

structure, you could potentially use just a single machine with a GPU to train enormous
nets in a reasonable time window.
Keep in mind that if you use Theano, you will have to build a deep net from the ground up.
The library does not provide complete functionality for creating a specific type of deep net.
Instead, you'll need to code every aspect of a net, like the model, the layers, the
activation, the training method, and any special methods for preventing overfitting. The good
news is that Theano allows you to build your implementation atop a set of vectorized
functions,
providing you with a highly efficient, optimized solution.
Do you have any experience coding a deep net with the Theano library? Please comment and
share your thoughts.
There are many other libraries that extend the functionality of Theano. For example,
the Blocks platform provides a wrapper for each Theano function, allowing you to access
the functions with parameters. The Lasagne package allows you to build a net on top of
Theano by providing the net's hyper-parameters layer by layer. Keras is another library with
a minimalist design that allows you to easily build a net layer by layer, train it, and
run it. Niche libraries like Passage are suited for text analysis applications that require
a recurrent net.
Currently, Theano provides no support for distributed, multi-node implementations. So
for example, training a net in a Hadoop cluster is not possible with Theano at this time.
Next up, we'll take a look at Deeplearning4j.
End of transcript. Skip to the start.

CAFFE

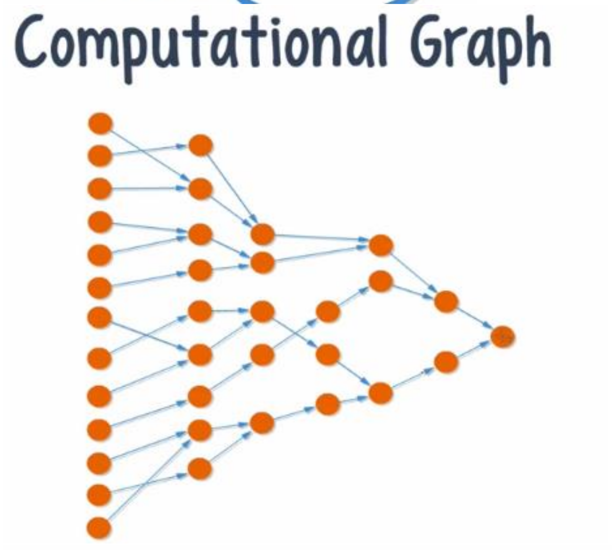




f transcript. Skip to the end.
If you need a library for a machine vision or a forecasting application, then Caffe might
be a good choice. This library lets you build your own deep nets with a sophisticated set
of layer configuration options. You can even access premade nets that were uploaded to
a community website. Let's take a look.
The Caffe Deep Learning Library was created by Google's Yangqing Jia, who won an ImageNet
Challenge in 2014.
Caffe was originally designed for machine vision tasks, so it's well-suited for convolutional
nets. However, recent versions of the library provide support for speech and text,
reinforcement
learning, and recurrent nets for sequence processing. Since the library is written in
C++ with CUDA, applications can easily switch between a CPU and a GPU as needed. Matlab
and Python interfaces are also available for Caffe.
With Caffe, you can build a deep net by configuring its hyper-parameters. In fact, the layer
configuration
options are very sophisticated. You can create a net with many different types of layers,
such as a vision layer, a loss layer, an activation layer, and a few others. So each layer can
perform a different function or take on a different role. This flexibility allows you
to develop extremely complex deep nets for your application. Caffe is supported by a
large community where users can contribute their own deep net to a repository known as
the "Model Zoo". AlexNet and GoogleNet are two popular user-made nets available to
the community. There are also a few educational resources like demos and slides, so if you're
going to use Caffe, it's a great place to start.
Caffe vectorizes input data through a special data representation called a "blob". A
"blob" is a type of array that speeds up data analysis and provides synchronization
capabilities between a CPU and a GPU.
Have you ever used the Caffe library in one of your own Deep Net projects? Please comment
and share your experiences.
End of transcript. Skip to the start.

TENSORFLOW

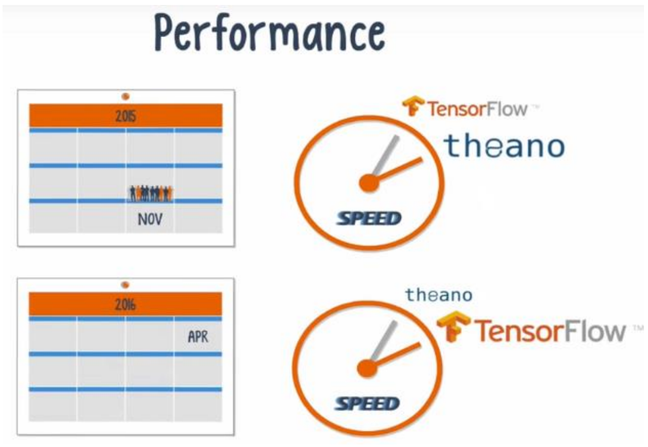
TENSORFLOW @ GOOGLE
DistBelief v2 = TensorFlow
Research at Google

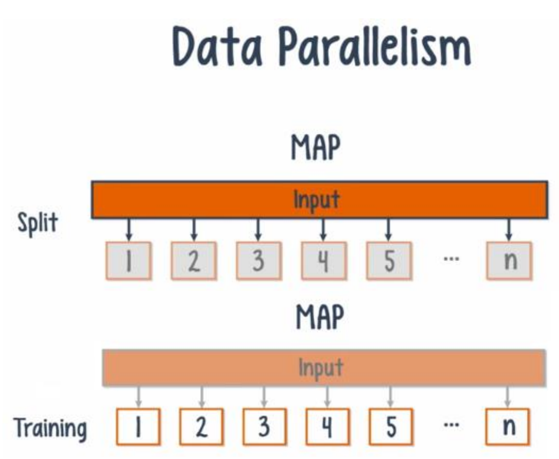

Computational Graph

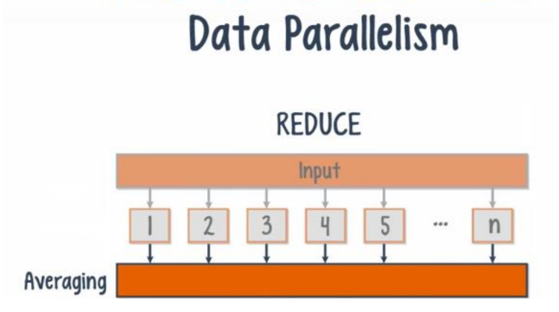
"Sub-expressions"
"Variables"
"Differentiation"
TensorFlow
theano

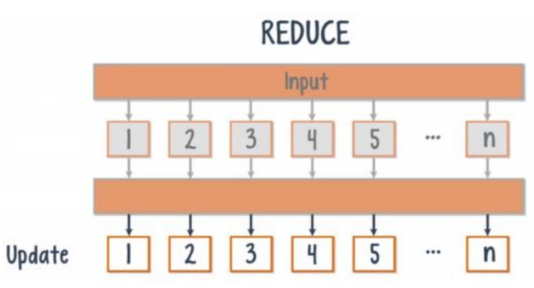
TensorFlow
Lua
Go
JavaScript
R

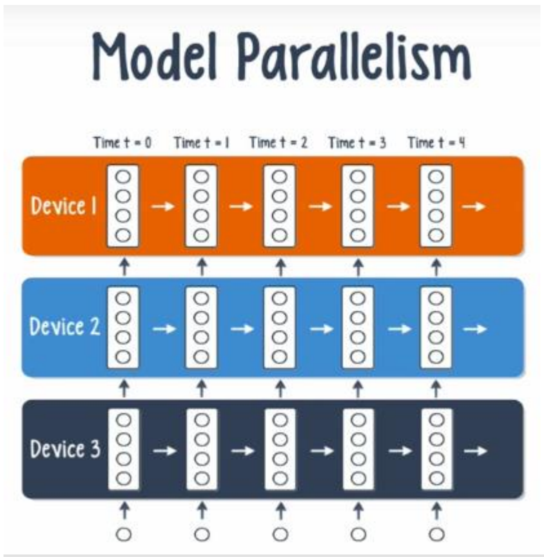
TensorFlow
Ruby
A PROGRAMMER'S BEST FRIEND
somatic

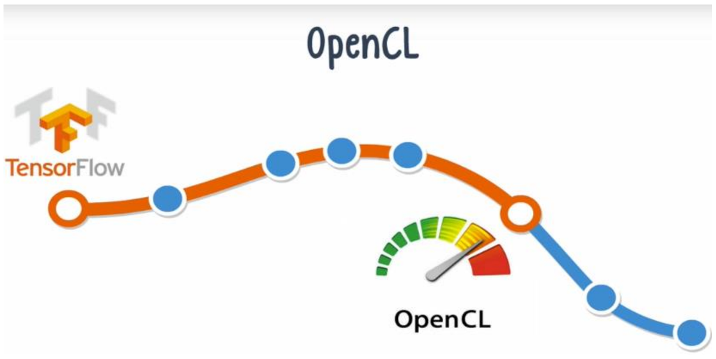

Performance
2015
NOV
TensorFlow
theano
SPEED
2016
APR
theano
TensorFlow
SPEED

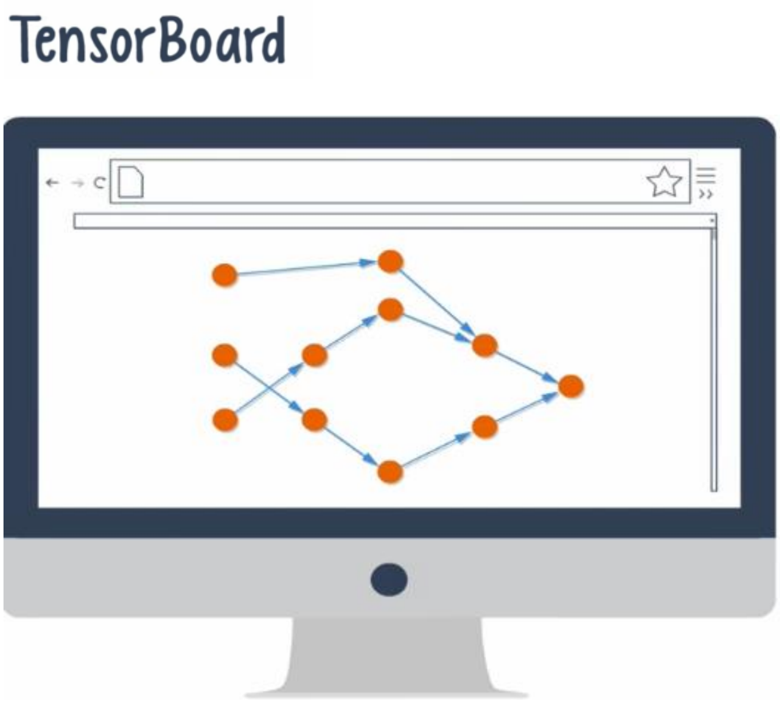

Data Parallelism
MAP
Split
Input
1 2 3 4 5 ... n
MAP
Input
Training
1 2 3 4 5 ... n

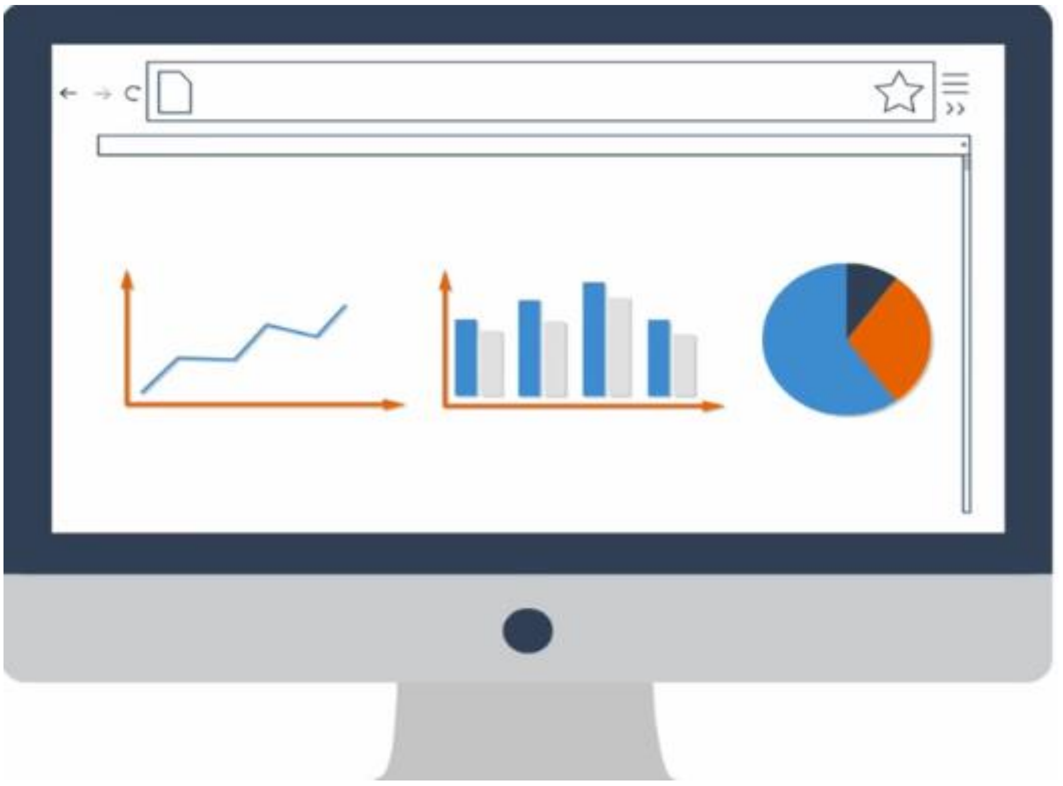

Data Parallelism
REDUCE
Input
1 2 3 4 5 ... n
Averaging


REDUCE
Input
1 2 3 4 5 ... n
Update
1 2 3 4 5 ... n


Model Parallelism
Time t = 0
Time t = 1
Time t = 2
Time t = 3
Time t = 4
Device 1
Device 2
Device 3


OpenCL
TensorFlow
OpenCL


TensorBoard



rt of transcript. Skip to the end.
Google's TensorFlow library is a great choice for building commercial-grade deep learning applications using Python. There's a lot of hype surrounding TensorFlow, so in this video we're going to take a closer look and shed some light on the library's various features.
TensorFlow grew out of an earlier Google library called “DistBelief”, which is a proprietary deep net library developed as part of the Google Brain project. The project team's vision was to build a system that simplified the deployment of large-scale machine learning models onto a variety of different hardware setups – anything from a smart phone, to single servers, to systems consisting of 100s of machines with 1000s of GPUs. In essence, this library would improve the portability of machine learning so that research models could be more easily applied to commercial-grade applications. Even though TensorFlow is only
6 months old, it's currently the most popular Machine Learning Library on GitHub.
Much like the Theano library, TensorFlow is based on the concept of a computational graph.
In a computational graph, nodes represent either persistent data or a math operation, and edges represent the flow of data between nodes. The data that flows through these edges is a multi-dimensional array known as a tensor, hence the library's name, “TensorFlow”.
The output from one operation or set of operations is then fed as an input into the next. Even though TensorFlow was designed to support neural networks, it can support any domain where computation can be modelled as a data flow graph. TensorFlow also adopts several useful features from Theano such as auto differentiation, shared and symbolic variables, and common
sub-expression elimination. And for an open source library, it has comprehensive and informative
documentation, in addition to a free massive open online course on Udacity as of March 2016.
Different types of deep nets can be built using TensorFlow, although there is currently

no support for hyper-parameter configuration. TensorFlow has a RoadMap that details some upcoming features, and while hyper-parameter configuration is mentioned, there is no specific timeline for this feature's implementation. For now, TensorFlow users have to work with an additional library called Keras if this flexibility is required. Right now TensorFlow has a “no-nonsense” interface for C++, and the team hopes that the community will develop more language interfaces through SWIG, an open-source tool for connecting programs and libraries. Recently, Jason Toy of Somatic announced the release of a SWIG interface to Ruby for the summer of 2016.
You may have noticed that TensorFlow and Theano are very similar in many ways, but there are
a few key differences. When TensorFlow was released in November of 2015, its compile and run times were several orders of magnitude slower than Theano, which was still a reported
issue as recent as March 2016.
The TensorFlow community – both the development team and outside contributors – have worked
hard to combat these performance issues. Soumith Chintala of Facebook regularly publishes updates
on the performance of different libraries on GitHub; an update in April of 2016 showed that TensorFlow performed reasonably well in the ImageNet category, with no Theano-based libraries listed in the analysis.
Another improvement over Theano comes in the form of parallelism. As we saw in a prior video, Deeplearning4j supports the training of a machine learning model on a distributed framework through the use of a two-step procedure called Iterative Map-Reduce. The underlying
concept, known as data parallelism, is implemented in a recent release known as distributed TensorFlow
– v0.8. Data parallelism allows you to train different subsets of the data on different nodes in a cluster for each training pass, followed by parameter averaging and replacement across the cluster.
v0.8 also implements model parallelism, where different portions of the model are trained on different devices in parallel. For example, you could use model parallelism to train stacked RNNs by deploying each RNN on a different device.
Even though most Deep Learning Libraries support CUDA, very few support OpenCL, a fast-rising
standard for GPU computing. In response to a top community issue currently open on this topic, the TensorFlow team has added OpenCL support to the RoadMap.
A nice feature is TensorBoard, a visualization tool for network architecture and performance. The tool allows you to zoom in and visualize different levels of the network, as well as view different summary-level metrics and changes over time throughout the training process.
TensorFlow has achieved significant results in the world of deep learning in a very short amount of time. If this trend continues, it is on track to become the premier library for building deep nets. In the next video, we’ll take a look at Metrics for Deep Learning.
End of transcript. Skip to the start.