My Notes on Deep Learning Simplified

Neural Nets

- Structure of neural network is like any other network.
- It's a series of interconnected nodes, called neurons and edges that join them together.
- Main function is to receive a set of inputs perform progressively complex calculations and use output to solve a problem.
- Classification is one application. Classification is categorizing a group of objects while only using some basic data features that describe them.
- Example of classifiers - logistic regression, support machine vectors (SVM), naive bales, neural networks.
- Classifier starts with data as inputs processes it through the hidden layers and at the output give a confidence score.
- Neural networks consists of input layers, hidden layers and output layers. Hidden layers are in between the input and output layers.
- Neural nets can be viewed as the result of spinning classifiers together in a layered web because each node in the hidden and output layers has its own classifier.
- The series of events starting the firing from input -> hidden layers -> output is called forward propagation (prop).
- Layered web of perceptrons greatly improve accuracy of individual perceptrons firing.
- Each set of inputs is influenced by its own set of weights and biases. That's why when you fire through same layers you get a different result.
- Each edge has a unique weight and each node has a unique bias. The prediction accuracy of a neural net depends on its weights and biases.
- We want the accuracy to be high, want the neural net to predict a value as close to the actual output as possible every time.
- The process of improving a neural nets accuracy is called training.

- 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. (Cost = Generated Output - Actual Output).
- The point of training is to make that cost as small as possible across millions of training examples.
- To do this we 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.

Why Use Deep Learning

- Deep learning has the ability to recognize incredibly complex patterns. Especially neural networks. As patterns get more complex neural networks outperform all competition.
- Computers have always been good with repetitive calculations given detailed instructions but they've been historically bad at pattern recognition.
- In terms of pattern complexity, if you only need to recognize simple patterns use a method live SVM or logistic regression.
- But as your data grows to tens or more inputs, neural nets start to outperform.
- And as the patterns get even more complex, neural networks with a small number of layers can become unusable. The reason is because the number of nodes required in each layer grows exponentially, with the number of possible patterns in the data. This makes training way to expensive (time) and the accuracy suffers.
- So for more intricate patterns (images) the only practical choice is a deep net.
- Deep nets are able to break the complex patterns down into a series of simpler patterns.
- Example: detecting a human face. A deep net would first uses the edges to detect different parts of the face (the lips, nose, eyes, ears) and would then combine the results together to form the whole face.
- This feature of using simpler patterns as building blocks for complex patterns are what give deep nets an advantage.

- The accuracy have become more and more impressive.
- Deep nets were inspired by the structure of our own brain. We decipher patterns similarly to a deep net. We detect complex patterns by first detecting and combining the simple ones.
- Only downside is that deep nets take a lot longer to train. But with recent advances in computing (GPUs) we can train them faster then ever.

Which Deep Net to Use?

- First step is to figure out if you're trying to build a classifier or find patterns in your data.
- Unsupervised learning is trying to find patterns in unlabeled data. If that is your aim, your best methods are to use a restricted boltzmann machine (RBM) or autoencoders.
- Supervised learning is trying to build a classifier using labeled data. There you have many different options to chose from depending on application.
- For text processing tasks like sentiment analysis, parsing and named entity recognition, use a recursive neural tensor network (RNTN) or recurrent net. For any language model that operates on the character level us a recurrent net.
- For image recognition use a deep belief network (DBN) or convolutional net.
- For object recognition use a convolutional net or RNTN.
- For speech recognition use a recurrent net.
- In general, deep belief nets (DBNs) and multi-layered perceptrons (MLPs) with rectified linear units (RELU) are good choices for classifications.
- For time series analysis it's best to use a recurrent net.
- The reason it's taken so long to implement deep nets is because they are incredibly difficult to train.

Why are Deep Nets Hard to Train

- When we try to train them using a method called back propagation you run into a fundamental problem called the vanishing gradient sometimes called the exploding gradient.
- When vanishing gradient occurs, training takes too long and the accuracy suffers greatly.
- When training a neural net, you are constantly calculating a cost value. Again the cost is the difference between the nets predicated output and the actual output from a set of labeled 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. This is known as forward prop.
- The training process utilizes something called a gradient which measures the rate at which a cost changes with respect to a change in a weight or bias.
- Deep architectures are your best and sometimes only choice for complex machine learning problems like facial recognition. But until 2006 there was no way to train deep nets because of a fundamental problem with the process known as the vanishing gradient.
- Think of a gradient as similar to a slope and the training process like a ball going down the slope. The steeper the slope the faster the ball goes down. The more flatter the slope, the slower the ball goes down. The same logic applies to the gradient of a deep net. When the gradient is large the net will train quickly and when it's small it will train much more slowly.
- The gradient starts to decay or vanish as you go back through the net. The gradients are much smaller in the earlier layers. As a result the early layers are the slowest to train. But this is a fundamental problem because the early layers are responsible for detecting the simple patterns and the building blocks (example: in facial recognition, the early layers detected the edges which were combined to form facial features which were later in the network).
- If the early layers get it wrong, the results built up by the net will be wrong.
- The process used for training a neural net is called back propagation (prop). Forward prop starts with the inputs and moves forward, back prop does the reverse. It calculates the gradient from right to left.

- In back prop, each time it calculates a gradient it uses all the previous gradients up to that point. But as you keep going back, things get much more complex. One node can use many different gradients throughout the net.
- The larger and deeper the net, the more complex it becomes and the more gradients each node has. This is because a gradient at any point is the product of the previous gradients up to that point. And the products (multiplication) of two numbers between zero and one, gives you a smaller number.
- Example say you have a rectangle which is one. Also say there are two gradients, one that is a 1/4 and one that is 1/3. Multiply them and you get 1/12. Then multiply that by 1/4 and you get 1/48. Numbers keep getting smaller the more you multiply.
- Because of all of this back prop ends up taking too long to train and the accuracy is often very low. Up until 2006, deep nets were underperforming shallow nets and machine learning algorithms but that changed with the restricted boltzmann machine.

Restricted Boltzmann Machine (RBM)

- Part of what allowed researchers to overcome the vanishing gradient problem.
- This is a method that can automatically find patterns in our data by reconstructing the input.
- It was the brainchild of Geoff Hinton at the University of Toronto, referred to as one of the fathers of deep learning.
- An RBM is a shallow two layer net, the first layer is known as the visible layer and the second layer is known as the hidden layer.
- Each node in the visible layer is connected every layer 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 backwards pass, it takes this set of numbers and translates them back to form the reconstructed 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 inter-relationships among the input features and they also allow the RBM to determine which input features are the most important when detecting patterns.
- Through several forwards and backwards passes, an RBM is trained to reconstruct the input data.
- Three steps are repeated over and over through the training process.
- Step 1: With a forward pass, every input is combined with an individual weight and one overall bias and result is passed to the hidden layer which may or may not activate.
- Step 2: Next in a backwards pass, each activation is combined with an individual weight and an overall bias and the result is passed to the visible layer for reconstruction.
- Step 3: At the visible layer the reconstruction is compared against the original input to compare the quality of the result.
- RBMs use a measure called KL Divergence to compare actual result to reconstructed result.
- Steps 1-3 are repeated with varying weights and biases until the input an the reconstruction are as close as possible.
- An interesting aspect of an RBM is that the data does not need to be labeled.
- This turns out to be very important for real world data sets such as photos, videos, voices and census data. All of which tend to be unlabeled. Rather then have 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 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 auto encoders because they have to encode their own structure.
- So how does an RBM being able to extract features help with the vanishing gradient? This pertains to the deep belief net.

Deep Belief Nets

- By combining RBMs together and introducing a clever training method, we obtain a powerful new model that finally solves our problem of the vanishing gradient. Which is a deep belief network (DBN)
- Also brainchild of Geoff Hinton at the University of Toronto. Conceived as an alternative to back prop.
- In terms of network structure a DBN is identical to an MLP (multi-layered perceptron) but when it comes to training they are entirely different.
- The difference in training methods is key factor that enables DBNs to outperform their shallow counterparts.
- A DBN can be viewed as a stack of RBMs (restricted boltzmann machines) where the hidden layer of one RBM is the visible layer of the one above it.
- A DBN is trained as follows: The first RBM is trained to reconstruct it's input as accurately as possible. 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. 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. (Example: facial recognition, early layers detect edges and later layers would use those results to form features).
- A DBN on the other hand works globally by fine tuning the entire input in succession as the model slowly improves. (Example: a camera lens slowly focusing on a picture).
- The reason a DBN works so well is because a stack of RBMs will outperform a single unit. Just like a multi-layered 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 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 a net's perception of the patterns

and often a small increase in the total accuracy. Fortunately the set of labeled data can be small relative to the original data set which is extremely helpful in real world applications.

- So to recap the benefits to a deep belief network: A DBN only needs a small labeled data set, which is important for real life 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.

Convolutional Neural Net (CNN)

- Dominated the machine vision space in recent years.
- They're so influential it's made deep learning one of the hottest topics in AI.
- But they can be tricky to understand.
- Pioneered by Yann Lacun at NYU.
- A convolutional net has been the go to solution of machine vision projects in the last few years.
- There are many component layers to CNNs.
- The first component is the convolutional layer.
- Example: Imagine you have a wall that represents a digital image, also imagine that you have a series of flashlights shining at the wall, creating a group of overlapping circles. There are 8 flashlights in each row and 6 rows in total. The purpose of these flashlights is to seek out certain patterns in the image, like an edge or color contrast. 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 is called a filter, which is able to determine if the given pattern occurs in this image and in what regions. Let's start at 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 shining at a wall all looking for a different pattern, so this particular convolutional layer is an (8x6x4) 3d 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. Think of it as the process of filtering through the image for a specific pattern. One

important note is that the weights and biases of this layer effect 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. But in a convolutional layer, neurons perform this convolution operation. Unlike the nets we've seen thus far where every neuron in a layer is connected to every neuron in an adjacent layer. A CNN has the flashlight structure. Each neuron is only connected to the input neuron 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 rectified linear units (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 which fires like in other nets. The activation used is called rectified linear units (RELU).
- CNNs are trained using back prop so once again the vanishing gradient can potentially still be an 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 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.

- So to recap, a typical deep CNN has three layers, a convolutional layer, a RELU layer and a pooling layer. 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 labeled data for training which can be challenging to obtain in a real world application.

Recurrent Neural Networks (RNN)

- If the patterns in your data change over time, your best network to use a recurrent neural network.
- This model has a simple structure with a built in feedback loop allowing it to act as a forecasting engine.
- RNNs are the brainchild of Jurgen Schmidhuber and Sepp Hochreiter. Their applications are extremely versatile ranging from speech recognition to driverless cars.
- All the nets covered so far have been feed forward neural networks. In a feed forward neural network signals only flow in one direction from input to output, one layer at a time.
- In a recurrent neural 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.
- Think of this process as a passage through time. Imagine four time steps. Time 0,1, 2, 3. Start at time = 0, at time = 1 the net takes the output of time from time = 0 and sends it back into the net along with the next input. The net repeats this for time = 2 and time = 3 and so on.
- Unlike feed forward nets, a recurrent net can receive a sequence of values as inputs and it can also produce a sequence of values as outputs.
- The ability to operate with sequences opens up these nets with a wide variety of applications.

- 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 then that can statistically forecast the demand and supply chain planning.
- Like we've seen with our previous deep learning models, by stacking RNNs on top of each other, you can form a net more capable of complex output then a single RNN working alone.
- Typically an RNN is an extremely difficult net to train. Since these nets use back propagation, 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 feed forward network.
- So training an RNN for a hundred time steps is like a hundred layer feed forward 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 which 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 gated recurrent unit (GRU) and long short-term memory (LSTM).
- Besides Gating there are a few other techniques such as Gradient clipping and Steeper gates and Better Optimizers.
- When it comes to training a recurrent net, GPUs are an obvious choice over ordinary CPUs. GPUs trained RNNs up to 250 times faster in tests.
- So under what circumstances would you use a RNN over a feed forward net? Feed forward net outputs one values which in many ways is a class or a prediction. Recurrent net is suited for time series data where an output can be the next value or the next several values in a sequence. So the answer depends on whether the application calls for classification/regression (Feed forward) or forecasting (RNN).

Auto Encoders

- Auto encoders are incredibly useful when trying to figure out the underlying structure of a data set such as having access to the most important data features gives you a lot of flexibility when you start applying labels.
- A Restricted boltzmann machine (RBM) is a very popular example of an auto encoder. There are other types of auto encoders such as de-noising and contractive.
- An auto encoder is a neural net that takes a set of typically unlabeled 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.
- Auto encoders are typically very shallow and are usually comprised of an input layer, output layer and hidden layer. An RBM is an example of an auto encoder with only two layers.
- In terms of the forward pass, there are two steps: The encoding and 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.
- Auto encoders are trained with Back propagation using a metric called Loss. As opposed to cost, lost measures the amount of information that was lost when the net tried to reconstruct the input. A net with a small loss value would produce reconstructions that would look very similar to the originals.
- Not all of these nets are shallow, in fact deep auto encoders are extremely useful tools for dimensionality reduction.
- Example: Consider an image containing a 28 X 28 grid of pixels. A neural net would need to process over 750 input values for just one image. Doing this across millions of images would waste significant amounts of memory and processing time. A deep auto encoder would encode this image into an impressive 30 numbers and still main 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 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 pattern.
- Deep auto encoders perform better at dimensionality reduction then their predecessor principle component analysis (PCA).

Recursive Neural Tensor Nets (RNTN)

- Useful when trying to discover the hierarchical structure of a set of data such as the parsed trees of a group of sentences. RNTN perform better then feed forward and recurrent nets.
- They are the brain child of Richard Socher @ Metamind
- The purpose of these nets was to analyze data that had a hierarchical structure. Originally designed for sentiment analysis where sentiment of a sentence depends not just on its component words but on the order on which they're syntactically grouped.
- The structure is as follows: An RNTN has three basic components. A parent group, called the root. A child group which we'll call the leaves (leaf). Each group is simply a collection of neurons where the number of neurons depends on the complexity of the input data. The root is connected to all sets of leaves but each leaf is not connected to each other. Technically speaking the three components form 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.
- The RNTN structure may seem simple but just like the recurrent net, the complexity comes from the way in which data moves throughout the network. In the case of a RNTN, this process is recursive.
- Example of recursion: Take a sentence, "The car is fast" and at step one you feed the first two words into leaf groups one and two respectively. The leaf groups don't actually receive the words but a vector (ordered set of numbers) representation of the words. 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 two vectors (represented by The and Car) move across the net to the root which fires out two values the class and 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 then a single word and the second leaf 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 ("The, Car, Is). This process continues until all the inputs are used up and the net has a parsed tree with every single word included. This is a simplified example of a RNTN and illustrates the main idea but in a practical application we typically counter more complex recursive processes.

Rather then 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.

- There can be different structures for the how to parse the words of a sentence for example. 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 of 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. In our example, “The car” is labeled a noun phrase and “is fast” is labeled as a verb phrase. It then works its way up and adds a special label that signifies the beginning of the parse structure.
- RNTN are trained with back propagation by comparing the predicted sentence structure with the proper sentence structure obtained from a set of labeled training data. Once trained the net will give a higher score to structures that are more similar to the parsed trees that it saw in training.
- RNTN are used in natural language processing for both syntactic parsing and sentiment analysis.
- RNTN is also used to parse images typically when an image contains a scene with many different components.

Deep Learning Use Cases

- Machine vision is one of the biggest applications of machine learning.
- Image search systems use deep learning for image classification and automatic tagging which allows the images to be accessible by a standard search query.
- Deep nets are also used to recognize objects within images, which allow for images to be searchable based on the objects within them.
- Other uses of deep learning include image and video parsing. Video recognition systems are important tools for driverless cars, remote robots and theft detection.
- While it may not be completely a part of machine vision, speech recognition got a powerful boost with the introduction of deep nets.

- Deep net parsers can be used to extract relations and facts from text as well as automatically translate text from other languages.
- These nets are extremely useful in sentiment analysis applications and can be used as a part of movie ratings and new product introductions.
- 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. Such as cancer detection or even drug discovery. In radiology they can help determine tumor malignancy through data from various mri's and ct scans.
- In finance, deep nets can help make buy and sell predictions based on market data streams, portfolio allocations and risk profiles. Depending on how they are trained, they are useful for short term and long term trading forecasting.
- In digital advertising, deep nets can help 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 how to optimally bid for ad space on a given webpage.
- 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 most susceptible to fraud.
- In marketing and sales, deep nets are used to gather and analyze customer information in order to determine the best upsetting strategies.

Deep Net Platforms

- Deep learning platforms come in two different forms: software platforms and full platforms.
- A platform is a set of tools that other people can build on top of. For example, think of the application that can be built off of the tools provided by iOS and android or windows and macOS.
- 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.
- Some advantages and disadvantages to using a platform vs 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. But you are constrained by the platform's selection of deep nets as well as the configuration options. But if you're looking to quickly deploy a deep net, a platform is the best way to go.
- A software library is a set of function 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 but you can code your own with the right deep net library. The obvious downside to libraries is the coding experience required to use then, but if you need the flexibility, they are a great resource.

Deep Learning Libraries

- A useful way to simplify the development process. Rather then re-inventing the wheel, you can take advantage of well-tested code that was created by experts in the field.
- A library is a pre-made 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 surrounded by big communities that provide support and contribute to the codebase. Deep learning has plenty of great libraries, several of which were created by key people in the field.
- 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 Then. Another notable library is deepmat.

Theano

- Theano provides an important set of functions for building deep nets that will train quickly on your machine.
- 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.
- But if you use Theano, you will have to build a deep net from the ground up. The library does not provide the 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.
- There are also 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. For example, training a net in Hadoop cluster is not possible with Theano at this time.

Overfitting

- When it comes to fitting a model to our data, our best bet is to follow the "Goldilocks" principle, we don't want to under fit our data and produce a model with poor accuracy

but we also don't want to overfit our data and produce a model that can't generalize to new samples. We want our model to be "just right".

- Underfitting is failing to give sufficient weight to important data features when distinguishing between different classes. To fix the problem of underfitting we can increase the model's precision by adding more features that help differentiate the samples.
- Overfitting is a model that analyzes every conceivable attribute in intricate ways but eventually we run the risk of developing a large set of arbitrary patterns that describe the training set very well but don't generalize to new samples of data. Models that overfit the data have failed to identify the true patterns that accurately differentiate the classes of the data set.
- So what causes overfitting to occur when training a neural network? Typically, overfitting occurs when the set of input features is far too big or the network architecture is too complex for the problem - in other words, there are more hidden layers and nodes per layer than are needed.
- In either case, overfitting will manifest itself in the weight and biases of the network. During training, the neural network assigns weight to each feature, which determines both their importance and the ways they will be recombined.
- When overfitting, the model will assign weights to features that are not needed and will add unnecessary complexity to the patterns.
- Overfitting is a very common problem across many different data science methods.
- One popular way to prevent overfitting is by splitting up the data into three sets - the training set, the test set and cross validation set. Along with parameter averaging, this method ensures that the model is not too dependent on any particular subset of the overall data set.
- For neural networks in particular, a common method is regularization. There are a few different types, such as L1 and L2 but each follows the same general principle. The model is penalized for having weights and biases that are too large
- Another method is Max Norm constraints, which directly adds a size limit to the weights and biases.
- A completely different approach is dropout, which randomly switches off certain neurons in the network preventing the model from becoming too dependent on a set of neurons and its associated weights.

- While these methods are all applied broadly across the model and don't systematically search for problem patterns or problem weights and biases, they have proven to be effective at reducing or preventing the problem of overfitting.

TensorFlow

- Built by Google and useful for build commercial grade apps with Python.
- TensorFlow is based on the concept of a computational graph, in a computational graph, does 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.
- The output from one operation or set of operations is then fed as an input into the next.
- TensorFlow adopts several useful features from Theano such as auto differentiation, shared and symbolic variables and common sub-expression elimination.
- Although open-sourced, it has comprehensive and informative documentation.
- Different types of deep nets can be built using TensorFlow.
- Supports data parallelism, 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.
- Also supports model parallelism, where different portions of the model are trained on different devices in parallel. For example, you could use model parallelism to train staked RNNs by deploying each RNN on a different device.
- Includes tools to visualize performance of the network at different levels as well as view different summary-level metrics and changes over time throughout the training process.

Metrics for Deep Learning

- Error is a straightforward measurement that captures the proportion of data points classified incorrectly - typically from the test data set. The calculation is simply the

number of incorrect classifications made by the net divided by the total number of classifications.

- Error has a significant drawback as a performance measurement, especially when the data points are skewed towards one class over another.
- The problem arises when we measure error globally across the set of all classes, rather than taking a granular view of the models performance at the class-level.
- Example: Two class classification problem, where a data point is either considered positive or negative. If the model makes a positive classification, there are two possibilities - the model could be correct, in which case we have a true positive or the model could be incorrect in which case we have a false positive. True negatives and false negatives are defined similarly when the model makes a negative prediction. If the number of occurrences of these four values are placed in a two by two grid, we form a valuable tool called a confusion matrix.
- Each square in the matrix is a placeholder for a value, so the values are not related to any particular square's visual size. Using this matrix, we can devise new measurements that overcome the issues of the error metric. We can do this by asking two related questions about the model's performance. We first look at the positives in the data set and ask, "What percentage of these positives was the model correctly able to identify?" This metric is know as recall, expressed as the number of true positives divided by the total number of positives in the data set. We can also look at the data and ask, "What percentage of these predictions were actually positive?" This metric is known as precision, expressed as the number of true positives divided by the number of data points classified as positive. These two questions sound similar but spoken another way, we want to know, "How many of the positive data points was the model able to "recall" and how "precise" were the actual predictions?" Also keep in mind that precision and recall are defined by the negative data points as well.
- We can use hybrid measures in order to achieve a balance between precision and recall. One of these measures, the F1 score, is calculated as the harmonic mean of precision and recall, which outperforms the standard arithmetic mean in the presence of outliers. The harmonic mean is also an improvement when the features have values between 0 and 1, something which comes in handy during the process of feature scaling.

- We can also examine precision and recall graphically by plotting these values and examining the area under the curve. The model that maximizes this area will typically be the best performing.
- We can extend these concepts to classification problems with more than two classes. The confusion matrix will be larger but the definitions for Precision and Recall remain the same as in the case of two-class classification. The difference is that now a data point can be misclassified in multiple ways, so false positives and false negatives must be summed over all the possible misclassification pairs. So, for multi-class classification, precision is the ratio of the number of true positives over all points classified as positive. And Recall is the ratio of the number of true positives over the total number of positives.

Deep Net Performance

- GPUs are a powerful tool for training deep nets and nearly every software library supports them but there are also some alternative to improve performance.
- CPU's are impractical for large scale deep nets because they are too slow and they work in serial not parallel.
- One trick we can use is to implement deep nets using vector. Vector algebra - like addition, dot products and transposes - are all operations that can be performed in parallel.
- Though the use of a parallel implementation, deep nets can be trained orders of magnitude faster. Parallelism implemented at the hardware level is known as parallel processing and parallelism at the software level is know as parallel programming.
- Parallel processing can be broken down into two general categories - shared memory and distributed computing.
- Shared memory options: First, the GPU, unlike a CPU where the number of built in cores is typically in the single or double digits, GPUs implement 100s sometimes even 1000s of cores. Each GPU core is versatile and capable of general purpose parallel computing. Any task that can be implemented in parallel can be performed on a GPU. With regards to deep nets, the most popular application for GPUs is the training process. The deep learning community provides great support of GPUs through libraries, implementations and a vibrant ecosystem fostered by nVidia. Despite all their advantages, GPUs do come with one big drawback. Their versatility

and general purpose design leads to extremely high power consumption. This becomes a significant issue for large scale deep nets, like the ones used by the tech giants.

- One alternative to the GPU is the Field Programmable Gate Array or FPGA. FPGAs are highly configurable and were originally used to electrical engineers to build mock-ups of different chip designs. That way the engineers could test different solutions to a given problem, without actually having to design a chip each time. FPGAs allow you to tweak the chips function at the lowest level which is the logic gate. So an FPGA can be tailored specifically for a deep net application allowing them to consume less power than a GPU. But there's an additional benefit since FGPAs can be used to run a deep net model and generate predictions. This would come in handy if, for example, you needed to run a large-scale convolutional net across 1000s of images per second. So FPGAs are a great tool but their great strength, that is their configurability can also be somewhat of a weakness. To properly setup and configure an FPGA, an engineer would need highly specialized knowledge in digital and integrated circuit design.
- Another option is the Application Specific Integrated Circuit, ASIC. ASICs are highly specialized with designs build in at the hardware and integrated circuit level. Once built they will perform very well at the tasks they were designed for but are generally unusable at any other tasks. Compared to GPUs and FPGA, ASICs tend to have the lowest power consumption requirements. There are several deep learning ASICs such as the Google Tensor.
- Aside from shared memory, parallelism can also be implemented using distributed computing. Generally speaking the three options for distributed computing are data parallelism, model parallelism and pipeline parallelism.
- Data parallelism allows you to train different subsets of the data on different nodes in a cluster for each training pass. This is followed by parameter averaging and replacement across the cluster.
- Pipeline parallelism works like an assembly line. Generally, there will be a number of jobs to be completed, each of which can be broken up into independent tasks. Each task for a given job will be dedicated to a worker, ensuring that each worker is relatively well-utilized. When a worker finishes its task, it can move on to a task for another job down the line, even if the other workers are still working on the current job.

- Parallel Programing by designing algorithms with parallelism in mind will allow you to take full advantage of the parallelism capabilities of the hardware.
- There are many ways to parallel-ize your code. One method (Model) is to decompose your data model into chunks, where each chunk is needed to perform an instance of a task. By organizing your data in this manner, each row can be used as an input in parallel. Another method (Algorithm) is to identify tasks that have dependencies and place them into a single group. By creating multiple groups that have no dependencies on one another, you can process the final job in parallel by dividing up the groups. Another method (Implementation) is to implement threads and processes that handle different tasks or task groups. This method can be performed independently but the performance benefits can be significant when combined with the second method.

Text Analytics

- One of the goals of Big Data Analytics is to make the most out of the mountain of unstructured textual data that we have access to. Standard Natural Language Processing techniques are okay but deep learning can truly revolutionize the field of text analytics.
- Natural Language Processing (NLP) is a big field that includes methods like Lemmatization, Named Entity Recognition, Part of Speech Tagging, Syntactic Parsing, Fact Extraction, Sentiment Analysis, Machine Translation and many others. These methods typically rely on a “language model”, a model that estimates the likelihood of seeing a particular component in a body of natural language.
- One example is the trigram model, which attempts to calculate the probability of seeing a specific sequence of three words in a natural language corpus. All of these methods perform well in practice but each one has some kind of limitation.
- Language is subjective and ambiguous - sometimes the same word can mean something different depending on the context and sometimes synonyms can have subtly different meanings depending on the way they are used. It can also be difficult to add new words to an existing language model, so NLP often requires a lot of manual curation. This added labor comes at the cost of variable quality and consistency.
- Deep learning is an important tool that overcomes some of the limitations of NLP. The fundamental difference between deep learning and traditional methods is the use

of vectors. To represent a word, a deep learning model might use the “one-hot” vector implementation, which has its roots in digital circuit design. In this form, each word is represented as a vector, whose length is the size of the entire vocabulary. Every value in the vector is set to 0, except for the index that maps to the word which is set to 1. Not surprisingly this can get out of hand if the vocabulary size is large. Large vectors can really slow down the processing time, as we’ve seen before.

- Dimensionality reduction is one application for which one-hot vectors are used. Take the “continuous bag of words” model for instance. Suppose we have some word, which we’ll call “w” surrounded by a fixed set of words called the “context”. In this model, the context is used as a set of features to predict what “w” might be, in a type of “fill in the blanks” application. A shallow three-layered neural net is used for this task, with the input layer containing one-hot vectors for the context words and the output layer firing the predicted target word, “w”.
- The “skip-gram” model is the reverse of the continuous bag of words model, since it allows you to estimate the context of words, given a target word. Since only the target is used as input, rather than a set of context words, the input to the hidden layer will much smaller. As a result, we can use the hidden layer’s activation to represent the target word, rather than use the long one-hot vector form. Basically, we can reduce the input vector’s dimensionality. Many popular tools online can do this for you.
- Syntactic parsing is an old problem that received a big boost for the use of Recursive Neural Tensor Networks (RNTNs). The RNTN consists of a root node connected to two leaf nodes. Two words are passed as input to the leaf nodes, with each leaf receiving its own word. The leaf nodes pass these words to the root, which processes them and essentially produces an intermediate parse, along with a score. This parse is fed into the leaf, along with the next word in the sentence. This process is recursive and it continues until every word in the sentence has been fed into the net. In practice, the recursive process is much more complex, since the net needs to analyze all possible sub-parses during the recursive step, rather than just using the next word. But by doing so, it is able to analyze and score every possible syntactic parse.
- Deep nets are also useful for machine translation. A recurrent net can take in a sequence of inputs along with a time delay and produce a sequence of outputs. A properly trained recurrent net can learn the inherent syntactic and semantic relationships of multiple different languages. So when the net is fed a sequence of

words in one language, it knows how to generate the appropriate sequence of words in the other language.

- Sentiment Extraction - sentiment, like syntax is hierarchical in nature and that a single word in the right place can change the entire meaning of the sentence. A well trained RNTN is capable of interpreting the deep semantic structure of this sentence in order to make the correct prediction.

Configuring a Deep Net

- Each hyper-parameter of a deep net is an important design choice, in terms of both the architecture and the training approach.
- The first decision is the architecture, where you need to specify the types of layers for the deep net. Input layers are generally straightforward since they're typically constrained by the application and the data set. However, there are many different types of hidden layers to choose from, such as convolution, pooling, activation and loss, just to name a few.
- Convolution and Pooling often have their own sub-parameters for the number of filters and the type of pooling, respectively. Some nets can also have matrix operations built in if you need to change the shape of your data between layers.
- When deciding on an output you need to specify the cost function for the layer, which could be the sum of squares or even cross-entropy, a function that's often used for multi-class classification.
- After choosing the layers, you then need to determine the number of neurons to place in each layer. There are a few heuristics for this aspect of neural net design, two of which are growing and pruning.
- With growing, you start with a smaller-than-expected neuron count and you keep adding neurons throughout the training process until the cost is no longer affected.
- Pruning is the reverse process. With pruning, you start with an excess of neurons and you keep removing them until there is a cost difference.
- Growing and Pruning are valuable techniques but they're computationally expensive for large nets. If resources are limited, a better approach is to start with an excess of neurons and then use a regularizer to prevent overfitting.

- There are a variety of Activation functions to choose from, the most popular of which are the logistic unit or the Sigmoid…the Hyperbolic Tangent and the Rectified Linear Unit (RELU). RELU is one approach to fight the vanishing gradient problem when working with back-prop. These activations all come with their own variants.
- When you combine an over-configured net with a regularizer, we typically end up with a model that can generalize well to new data points. The reason is that a net with more neurons has more ways to combine its weights and biases. As a result, there are more possible "patterns" that the network can learn to recognize, so there's a better chance of finding some kind of weight and bias configuration with the lowest possible cost. Think of it like a rough cut technique - we error on the side of caution in the hopes that a good solution will exist and then we cut away the rough edges until we find something that works.
- If you need to build in recurrence, you will need carefully consider the size of the net's input memory. This will determine how much of the past input the net is required to remember. Gating units, like GRUs and LSTMs are designed specifically to help a net remember the different parts of an input sequence.
- The success of the training process depends heavily on the choice of parameters. Back-prop in particular is sensitive to the learning rate. If the rate is too large, the algorithm may simply skip over the point that minimizes overall cost. If the rate is too small the training process will take an unreasonable amount of time. A common practice is to use a rate that changes as the training progresses. Generally, the rate starts out large in the beginning to speed up training and then decreases gradually as training progresses. The net starts learning in a broad sense and then fine tunes its understanding. Some strategies for dynamically altering the learning rate include, Adagrad, RMSprop and Adadelta.
- Before a net can be trained, we need to assign starting values to the weights and biases in a process called initialization. The easiest solution is to simply randomize the values but since the quality of the initial weights has such a big effect on training times, this may not always be appropriate. An example of an advanced method of doing this is a DBN using an RBM to set the initial weights and biases before supervised fine-tuning takes place.
- If you're working with RNNs, you can use techniques like gradient clipping and steeper gates to speed up training. But generally speaking, if you've exhausted all the available options and your deep nets are still falling short of performance targets,

just raise the number of training epochs. This strategy will work for every type of deep net.

Reinforcement Learning

- The origins of reinforcement learning goes all the way back to AI, animal psychology and control theory.
- It involves an autonomous agent, like a person, animal, robot or deep net, learning to navigate an uncertain environment with the goal of maximizing a numerical reward.
- If you're building an autonomous agent, how would you model this? We know that the agent's actions will change the state of the environment, so a model would need to be able to take a state and an action as input and generate the maximum expected reward as output. But since that only gets you to the next state, you'll need to take into account the total expected reward for every action from the current till the end state. The way this works will be different for every application.
- Reinforcement learning isn't just a fancy way to say supervised learning. Supervised learning is all about making sense of the environment based on historical examples. But that isn't always the best way to do things.
- Reinforcement learning is all about the reward. You get points for your action. In the example of driving, you get points for staying in your lane, driving under the speed limit, signaling when you're supposed, etc. But you can also lose points for dangerous actions like tailgating and speeding. Your objective is to get the maximum number of points possible given the current state of the traffic on the road around you.
- Reinforcement learning emphasizes that an action results in a change of state, which is something a supervised learning model doesn't focus.
- Exploration vs exploitation, with reinforcement learning, an agent can explore the trade-off between exploration and exploitation and choose the path to the maximum expected reward.
- Reinforcement also falls into the broader umbrella of artificial intelligence. It involves topics like goal setting, planning and perception. And it can even form a bridge between AI and the engineering disciplines.