Deep Learning Specialization by Andrew Ng — 21 Lessons Learned

Ryan Shrott Oct 25, 2017 · 10 min read

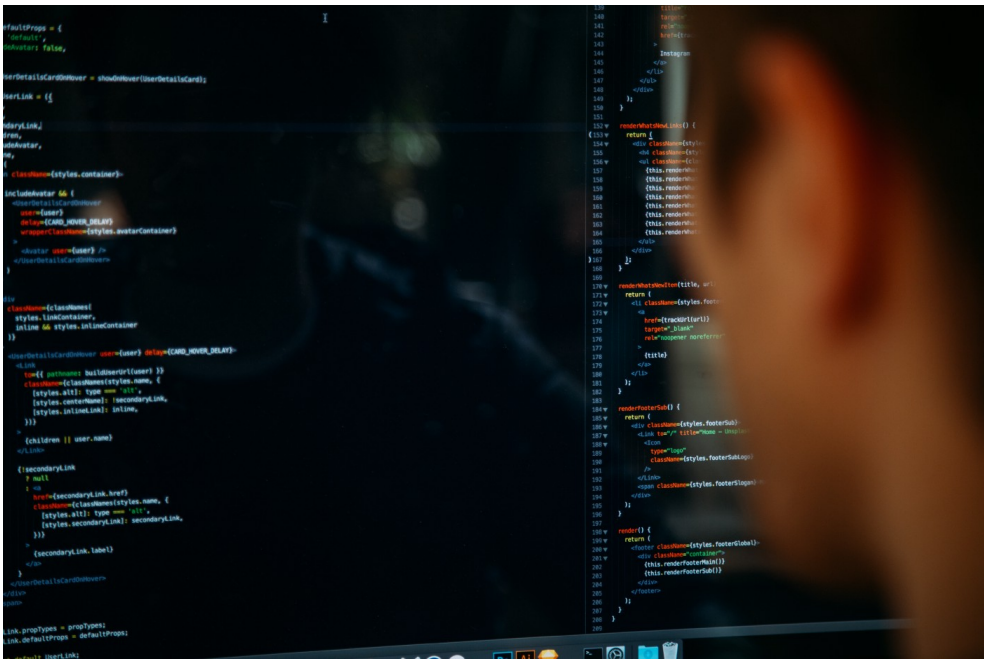


Photo by Charles Deluvio on Unsplash

I recently completed all available material (as of October 25, 2017) for Andrew Ng's new deep learning course on Coursera. There are currently 3 courses available in the specialization:

1. Neural Networks and Deep Learning
2. Improving Deep Neural Networks: Hyperparamater tuning, Regularization and Optimization
3. Structuring Machine Learning Projects

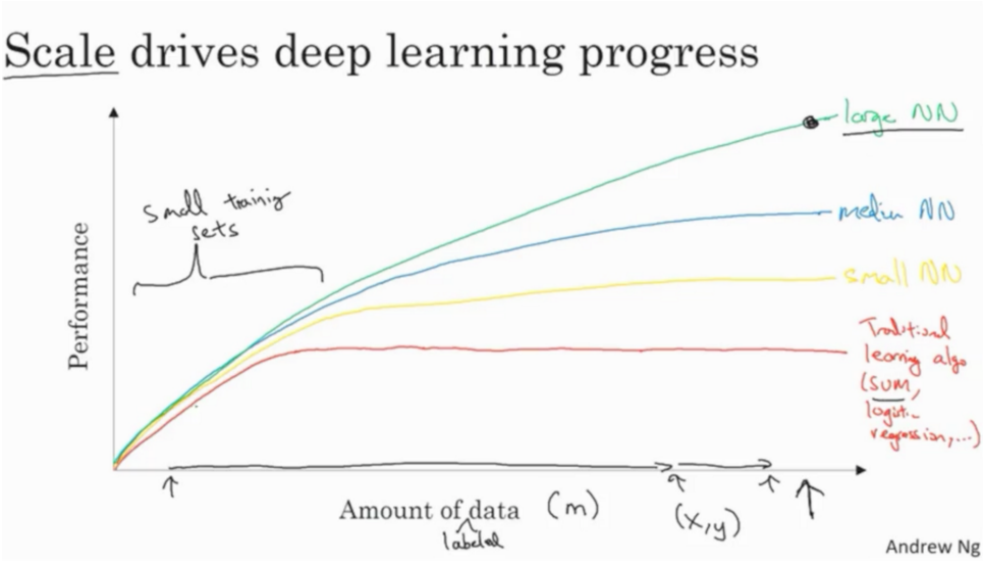
I found all 3 courses extremely useful and learned an incredible amount of practical knowledge from the instructor, Andrew Ng. Ng does an excellent job of filtering out the buzzwords and explaining the concepts in a clear and concise manner. For example, Ng makes it clear that supervised deep learning is nothing more than a multidimensional curve fitting procedure

and that any other representational understandings, such as the common reference to the human biological nervous system, are loose at best.

The specialization only requires basic linear algebra knowledge and basic programming knowledge in Python. In my opinion, however, you should also know vector calculus to understand the inner workings of the optimization procedure. If you don't care about the inner workings and only care about gaining a high level understanding you could potentially skip the Calculus videos.

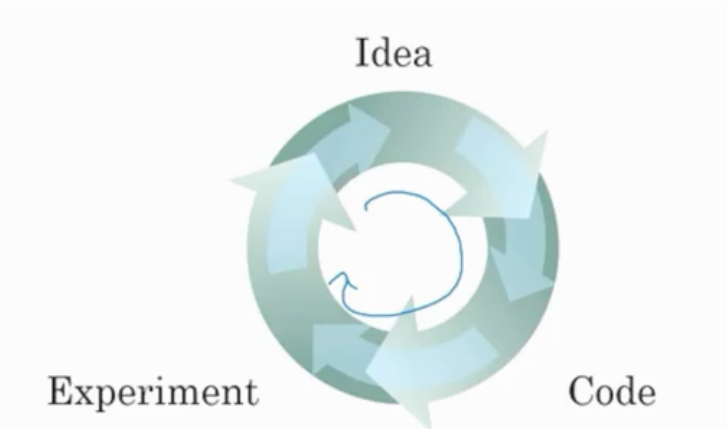
Lesson 1: Why Deep Learning is taking off?

90% of all data was collected in the past 2 years. Deep neural networks (DNN's) are capable of taking advantage of a very large amount of data. As a result, DNN's can dominate smaller networks and traditional learning algorithms.



Shows how scale drives performance in DNN's

Furthermore, there have been a number of algorithmic innovations which have allowed DNN's to train much faster. For example, switching from a sigmoid activation function to a RELU activation function has had a massive impact on optimization procedures such as gradient descent. These algorithmic improvements have allowed researchers to iterate throughout the IDEA -> EXPERIMENT -> CODE cycle much more quickly, leading to even more innovation.



Deep Learning Development Cycle

Lesson 2: Vectorization in Deep Learning

Before taking this course, I was not aware that a neural network could be implemented without any explicit for loops (except over the layers). Ng does an excellent job at conveying the importance of a vectorized code design in Python. The homework assignments provide you with a boilerplate vectorized code design which you could easily transfer to your own application.

Lesson 3: Deep Understanding of DNN's

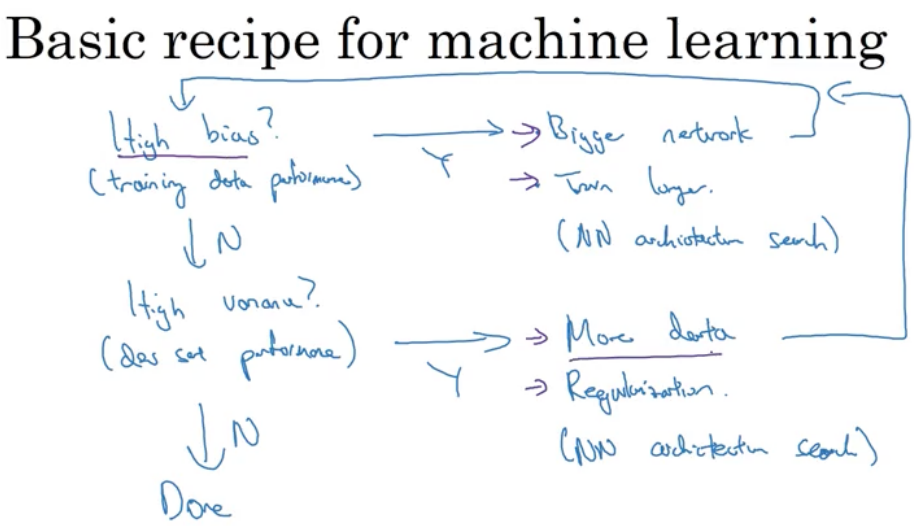
The first course actually gets you to implement the forward and backward propagation steps in numpy from scratch. By doing this, I have gained a much deeper understanding of the inner workings of higher level frameworks such as TensorFlow and Keras. Ng explains the idea behind a computation graph which has allowed me to understand how TensorFlow seems to perform "magical optimization".

Lesson 4: Why Deep Representations?

Ng gives an intuitive understanding of the layering aspect of DNN's. For example, in face detection he explains that earlier layers are used to group together edges in the face and then later layers use these edges to form parts of faces (i.e. nose, eyes, mouth etc.) and then further layers are used to put the parts together and identify the person. He also explains the idea of circuit theory which basically says that there exists functions which would require an exponential number of hidden units to fit the data in a shallow network. The exponential problem could be alleviated simply by adding a finite number of additional layers.

Lesson 5: Tools for addressing Bias and Variance

Ng explains the steps a researcher would take to identify and fix issues related to bias and variance problems. The picture he draws gives a systematic approach to addressing these issues.

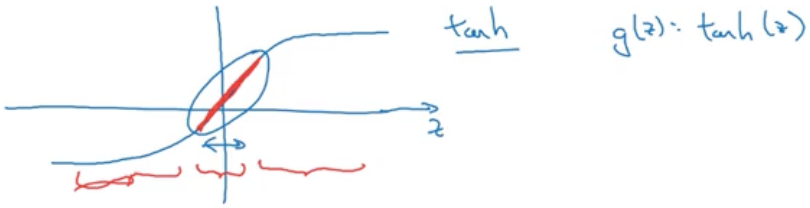


Cookbook for addressing bias and variance problems

He also addresses the commonly quoted “tradeoff” between bias and variance. He explains that in the modern deep learning era we have tools to address each problem separately so that the tradeoff no longer exists.

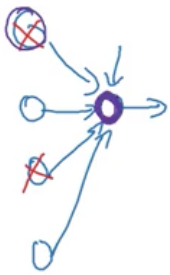
Lesson 6: Intuition for Regularization

Why does a penalization term added to the cost function reduce variance effects? The intuition I had before taking the course was that it forced the weight matrices to be closer to zero producing a more “linear” function. Ng gave another interpretation involving the tanh activation function. The idea is that smaller weight matrices produce smaller outputs which centralizes the outputs around the linear section of the tanh function.



tanh activation function

He also gave an interesting intuitive explanation for dropout. Prior to taking the course I thought that dropout is basically killing random neurons on each iteration so it’s as if we are working with a smaller network, which is more linear. His intuition is to look at life from the perspective of a single neuron.



Perspective from a single neuron

Since dropout is randomly killing connections, the neuron is incentivized to spread it’s weights out more evenly among its parents. By spreading out the weights, it tends to have the effect of shrinking the squared norm of the weights. He also explains that dropout is nothing more than an adaptive form of L2 regularization and that both methods have similar effects.

Lesson 7: Why normalization works?

Ng demonstrates why normalization tends to improve the speed of the optimization procedure by drawing contour plots. He explicitly goes through an example of iterating through a gradient descent example on a normalized and non-normalized contour plot.

Lesson 8: The importance of initialization

Ng shows that poor initialization of parameters can lead to vanishing or exploding gradients. He demonstrates several procedure to combat these issues. The basic idea is to ensure that each layer's weight matrices has a variance of approximately 1. He also discusses Xavier initialization for tanh activation function.

Lesson 9: Why mini-batch gradient descent is used?

Using contour plots, Ng explains the tradeoff between smaller and larger mini-batch sizes. The basic idea is that a larger size becomes to slow per iteration, while a smaller size allows you to make progress faster but cannot make the same guarantees regarding convergence. The best approach is do something in between which allows you to make progress faster than processing the whole dataset at once, while also taking advantage of vectorization techniques.

Lesson 10: Intuitive understanding of advanced optimization techniques

Ng explains how techniques such as momentum and RMSprop allow gradient descent to dampen it's path toward the minimum. He also gives an excellent physical explanation of the process with a ball rolling down a hill. He ties the methods together to explain the famous Adam optimization procedure.

Lesson 11: Basic backend TensorFlow understanding

Ng explains how to implement a neural network using TensorFlow and also explains some of the backend procedures which are used in the optimization procedure. One of the homework exercises encourages you to implement dropout and L2 regularization using TensorFlow. This further strengthened my understanding of the backend processes.

Lesson 12: Orthogonalization

Ng discusses the importance of orthogonalization in machine learning strategy. The basic idea is that you would like to implement controls that only affect a single component of your algorithms performance at a time. For example, to address bias problems you could use a bigger network or more robust optimization techniques. You would like these controls to only affect bias and not other issues such as poor generalization. An example of a control which lacks orthogonalization is stopping your optimization procedure early (early stopping). This is because it simultaneously affects the bias and variance of your model.

Lesson 13: Importance of a single number evaluation metric

Ng stresses the importance of choosing a single number evaluation metric to evaluate your algorithm. You should only change the evaluation metric later on in the model development process if your target changes. Ng gives an example of identifying pornographic photos in a cat classification application!

Lesson 14: Test/dev distributions

Always ensure that the dev and test sets have the same distribution. This ensures that your team is aiming at the correct target during the iteration process. This also means that if you decide to correct mislabeled data in your test set then you must also correct the mislabelled data in your development set.

Lesson 15: Dealing with different training and test/dev distributions

Ng gives reasons for why a team would be interested in not having the same distribution for the train and test/dev sets. The idea is that you want the evaluation metric to be computed on examples that you actually care about. For example, you may want to use examples that are not as relevant to your problem for training, but you would not want your algorithm to be evaluated against these examples. This allows your algorithm to be trained with much more data. It has been empirically shown that this approach will give you better performance in many cases. The downside is that you have different distributions for your train and test/dev sets. The solution is to leave out a small piece of your training set and determine the generalization capabilities of the training set alone. Then you could compare this error rate to the actual development error and compute a “data mismatch” metric. Ng then explains methods of addressing this data mismatch problem such as artificial data synthesis.

Lesson 16: Train/dev/test sizes

The guidelines for setting up the split of train/dev/test has changed dramatically during the deep learning era. Before taking the course, I was aware of the usual 60/20/20 split. Ng stresses that for a very large dataset, you should be using a split of about 98/1/1 or even 99/0.5/0.5. This is due to the fact that the dev and test sets only need to be large enough to ensure the confidence intervals provided by your team. If you are working with 10,000,000 training examples, then perhaps 100,000 examples (or 1% of the data) is large enough to guarantee certain confidence bounds on your dev and/or test set.

Lesson 17: Approximating Bayes optimal error

Ng explains how human level performance could be used as a proxy for Bayes error in some applications. For example, for tasks such as vision and audio recognition, human level error would be very close to Bayes error. This allows your team to quantify the amount of avoidable bias your model has. Without a benchmark such as Bayes error, it’s difficult to understand the variance and avoidable bias problems in your network.

Lesson 18: Error Analysis

Ng shows a somewhat obvious technique to dramatically increase the effectiveness of your algorithms performance using error analysis. The basic idea is to manually label your misclassified examples and to focus your efforts on the error which contributes the most to your misclassified data.

Image	Dog	Great Cats	Blurry	Instagram	Comments
1	✓			✓	Pitbull
2			✓	✓	
3		✓	✓		Rainy day at zoo
⋮	⋮	⋮	⋮		
% of total	8%	43%	61%	12%	

Cat Recognition App Error Analysis

For example, in the cat recognition Ng determines that blurry images contribute the most to errors. This sensitivity analysis allows you see how much your efforts are worth on reducing the total error. It may be the case that fixing blurry images is an extremely demanding task, while other errors are obvious and easy to fix. Both the sensitivity and approximate work would be factored into the decision making process.

Lesson 19: When to use transfer learning?

Transfer learning allows you to transfer knowledge from one model to another. For example, you could transfer image recognition knowledge from a cat recognition app to a radiology diagnosis. Implementing transfer learning involves retraining the last few layers of the network used for a similar application domain with much more data. The idea is that hidden units earlier in the network have a much broader application which is usually not specific to the exact task that you are using the network for. In summary, transfer learning works when both tasks have the same input features and when the task you are trying to learn from has much more data than the task you are trying to train.

Lesson 20: When to use multi-task learning?

Multi-task learning forces a single neural network to learn multiple tasks at the same time (as opposed to having a separate neural network for each task). Ng explains that the approach works well when the set of tasks could benefit from having shared lower-level features and when the amount of data you have for each task is similar in magnitude.

Lesson 21: When to use end-to-end deep learning?

End-to-end deep learning takes multiple stages of processing and combines them into a single neural network. This allows the data to speak for itself without the bias displayed by humans in hand engineering steps in the optimization procedure. To the contrary, this approach needs much more data and may exclude potentially hand designed components.

Conclusion

Ng's deep learning course has given me a foundational intuitive understanding of the deep learning model development process. The lessons I explained above only represent a subset of the materials presented in the course. After completing the course you will not become an expert in deep learning. My only complaint of the course is that the homework

assignments were too easy. I was not endorsed by deeplearning.ai for writing this article.

—

That's all folks — if you've made it this far, please comment below and add me on LinkedIn.

My Github is here.

Machine Learning | Deep Learning | Neural Networks | Artificial Intelligence | Towards Data Science