Report of MOOC

Deep Learning

Guangyue CHEN

2019/4/5

Contents

1 Introduction

Deep Learning is a subfield of machine learning concerned with algorithms inspired by the structure and function of the brain called artificial neural networks. It has been applied to fields including computer vision, speech recognition, natural language processing, audio recognition(etc.) where they have produced results comparable to and in some cases superior to human experts.

The basic technical idea of deep learning have been around for decades, but with the revolution of the algorithm , the increment of the data and the more powerful computation, deep learning is playing a more and more inportant role in this world.

1.1 Deep Learning - Coursera

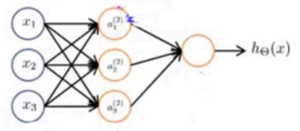
This Stanford online courses is taught by Andrew Ng who is one of the mostly influencing professors in this domain. In this class, I learned about not only the theoretical underpinnings of neural networks and different methods of Optimization , but also gain the practical experience to quickly and powerfully apply these techniques to new problems.

1.2 Course content

The course has a total of 19 weeks and I finished the first 6. It introduces us from simple Logistic regression algorithms, then gradually deepen to Neural Networks and its regularization and optimization, which also involves some derivations of mathematical formula. It has the test every weak. What's more, it has also one project per week to practice the knowledge that we learned. In general, it is divided into the following sections.

2 Neural Networks and Deep Learning

Neural Network



$a_i^{(j)} =$ "activation" of unit i in layer j

$\Theta^{(j)} =$ matrix of weights controlling function mapping from layer j to layer $j+1$

$$
\begin{aligned}
a_1^{(2)} &= g(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3) \\
a_2^{(2)} &= g(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3) \\
a_3^{(2)} &= g(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3) \\
h_\Theta(x) = a_1^{(3)} &= g(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)})
\end{aligned}
$$

Figure4:Neural Networks Model

In the **Neural Networks** model, there are 3 types of layer: input layer, hidden layer, output layer. Each layer content n nodes. After the modelization, we can compute our activation nodes by using a matrix of parameters. We apply each row of the parameters to our inputs to obtain the value for one activation node. It is used usually in supervised learning and are creating tremendous economic value.

And deep learning is a Neural Network consisting of a hierarchy of layers, whereby each layer transforms the input data into more abstract representations. These series of layers, between input and output, identify the input features and create a series of new features based on the data, just as our brain. In deep learning the more layers a network has, the higher the level of features it will learn. The output layer combines all these features and makes a prediction.

Recent years, computer vision has also made huge strides in the last several years, mostly due to deep learning. For image applications we'll often use convolution on neural networks, often abbreviated CNN. And so for sequence data (like audios), we often use an RNN, a recurrent neural network.

2.1 Neural Networks Basics and Logistic Regression

2.1.1 Vectorization

In the deep learning in practice, we often train on relatively large data sets, because that's when deep learning algorithms perform better. So it's important to make sure the code run very quickly. And it will be much faster if you vectorize your code. So in this course we often try a lot to avoid the 'for' loop.

Here we have one coding practice for Python photo data structure and Python basics for vectorizing computation.

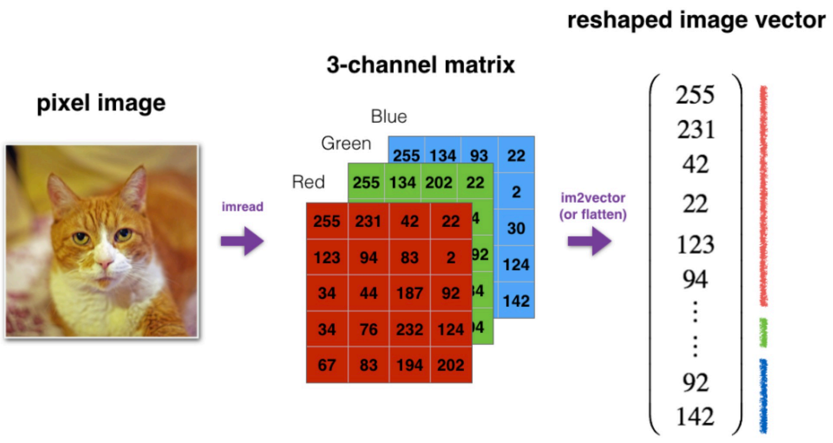


Figure1: Photo and Vectorization

2.1.2 Logistic Regression As A Neural Network

In the logistic regression model, through the feature extraction, we first select the characteristic variable x which affects the estimated variable y, and then through our training set and learning algorithm, we can get a hypothesis h. Here h can be expressed as:

$$h_\theta(x) = b + w_1 * x_1 + w_2 * x_2 + ...$$

And then dicide the decision boundary.

$$h_\theta(x) \geq 0.5, y = 1.$$

$$h_\theta(x) < 0.5, y = 0.$$

Considering y equals 0 or 1, our cost function for logistic regression looks like:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}log(h_\theta(x^{(i)})) + (1 - y^{(i)})log(1 - h_\theta(x^{(i)}))]$$

Furthermore, with vectoration, we get:

$$Z = W^T X + b$$

$$A = \sigma(Z) \geq 0.5$$

With vectorizing Logistic Regression's Gradient:

$$dZ = A - Y$$

$$dW = \frac{1}{m} X dZ^T$$

$$db = \frac{1}{m} np.sum(dZ)$$

$$W = W - \alpha(learning\ rate)\ dW$$

$$b = b - \alpha db$$

So we can finish one iteration. So here we have one project, we build a Logistic Regression, using a Neural Network mindset. And we can see that Logistic Regression is actually a very simple Neural Network.

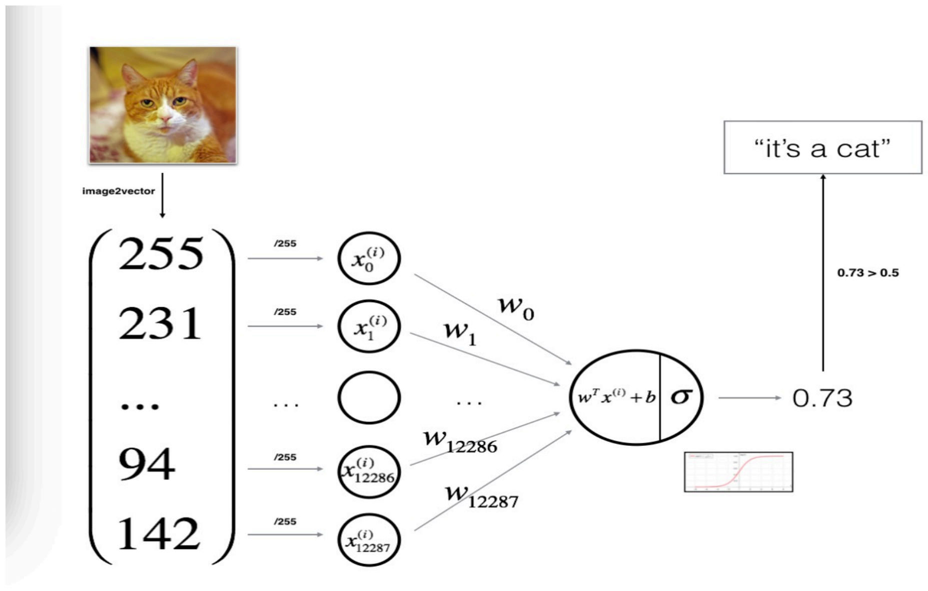
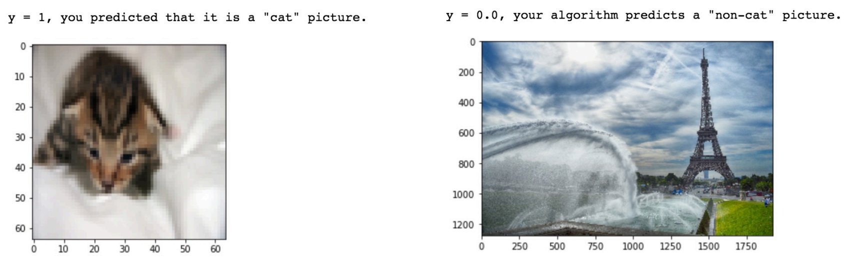


Figure2: Mathematical expression of the logistic algorithm

This project is a simple image-recognition algorithm that can correctly classify pictures as cat or non-cat. Which use Logistic Regression with Neural Network minset.



y = 1, you predicted that it is a "cat" picture.
y = 0.0, your algorithm predicts a "non-cat" picture.

Figure3: Resaults for algorithm

2.2 Shallow Neural Network

The third week, we study the most important algorithm for Neural Network and apply it into a shallow Neural Network.

2.2.1 Forward and Backpropagation

Neural Network with just one hidden layer is called Shallow Neural Network. With Vectorization, we compte the prediction of y by Forward propagation.($\sigma()$:activation functions)

$$a^{[1]} = \sigma(W^{[1]}x + b^{[1]})$$
$$\hat{y} = \sigma(W^{[2]}a^{[1]} + b^{[2]})$$

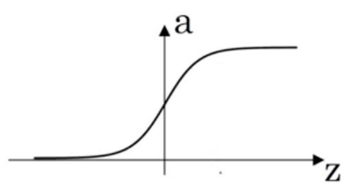
"Backpropagation" is neural-network terminology for minimizing our cost function, just like what we were doing with gradient descent in logistic regression.

$$dz^{[2]} = \hat{y} - y$$
$$dW^{[2]} = \frac{1}{m}dz^{[2]}a^{[1]^T}$$
$$db^{[2]} = \frac{1}{m}np.sum(dz^{[2]}, axis = 1, keepdims = True)$$
$$dz^{[1]} = W^{[2]^T}dz^{[2]} * \sigma^{[1]'}(z^{[1]})$$
$$dW^{[1]} = \frac{1}{m}dz^{[1]}x^{^T}$$
$$db^{[1]} = \frac{1}{m}np.sum(dz^{[1]}, axis = 1, keepdims = True)$$

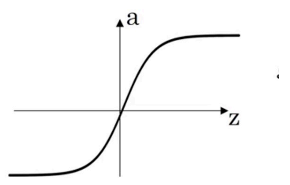
2.2.2 activation functions

When we build a neural network, we should choose what activation function to use in the hidden layers. Because if we don't have an activation function, then no matter how many layers our neural network has, all it's doing is just computing a linear activation function, which is really bad.

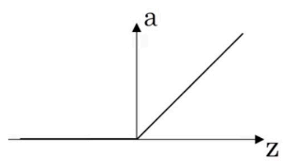
Here are four common activation functions, we usually use Relu for the hidden layers and Sigmoid or tanh for the outout layer.



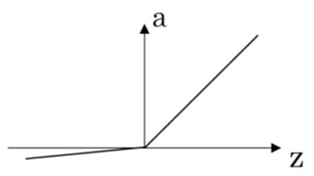
(a) $sigmoid: \ \sigma(z) = \frac{1}{1+e^{-z}}$



(b) Tanh:$tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$



(c) ReLU:$max(0, z)$



(d) leakyReLU:$max(0.01z, z)$

2.2.3 Planar data classification

And then it's week 3 programming assignment. It's the first time to build a neural network, which has just one hidden layer. Then we compare neural network and logistic regression. We can see a big difference these two models.

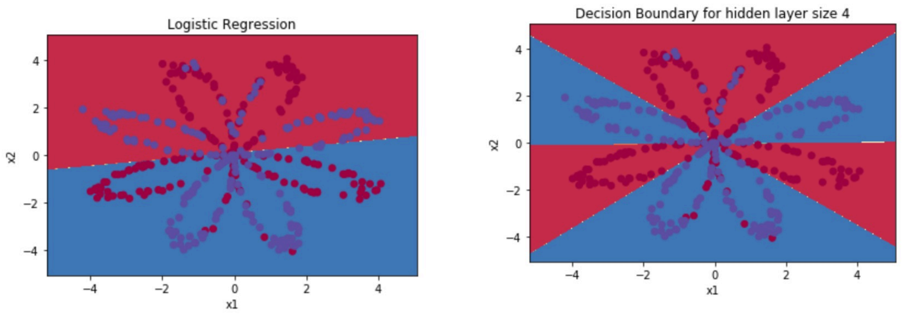


Figure4: Resaults for data classification

2.3 Deep Neural Networks

For implementing the Deep Neural Net with L layers, we need to use one activation function that replicates the previous layer L—1 times(usually RELU activation function), then follows with another one activation function(usually SIGMOID).

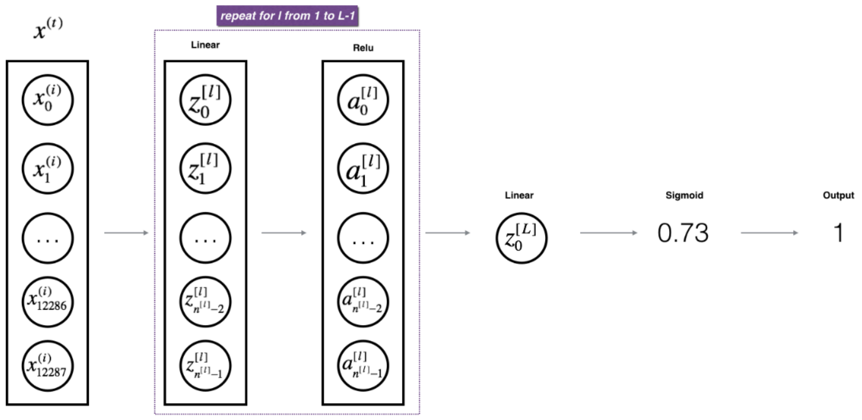


Figure5: L-layers Neural Net model

So for the forward propagation, we use the equations bellow:

$$A^{[l]} = \sigma(Z^{[l]}) = \sigma(W^{[l]}A^{[l-1]} + b^{[l]})$$

And the cost function become to:

$$-\frac{1}{m}\sum_{i=1}^{m}(y^{(i)}log(a^{[L](i)}) + (1-y^{(i)})log(1-a^{[L](i)}))$$

So for the back propagation, we need to update the parameters from L layer to the first layer:

$$dW^{[l]} = \frac{1}{m}dZ^{[l]}A^{[l-1]T}$$

$$db^{[l]} = \frac{1}{m}\sum_{i=1}^{m}dZ^{[l](i)}$$

$$dA^{[l-1]} = W^{[l]T}dZ^{[l]}$$

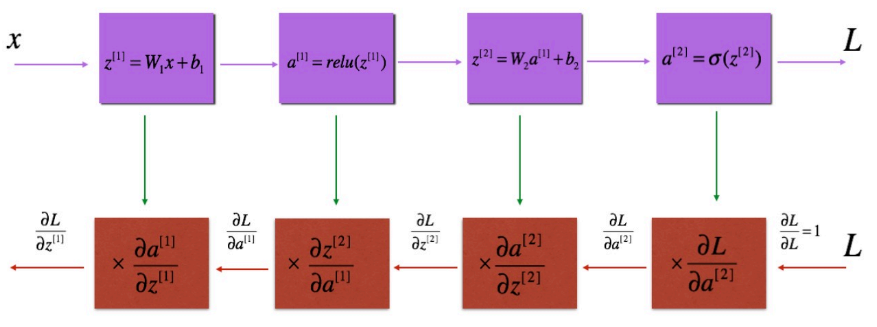

x
$z^{[1]} = W_1 x + b_1$
$a^{[1]} = relu(z^{[1]})$
$z^{[2]} = W_2 a^{[1]} + b_2$
$a^{[2]} = \sigma(z^{[2]})$
L
$\frac{\partial L}{\partial z^{[1]}}$
$\times \frac{\partial a^{[1]}}{\partial z^{[1]}}$
$\frac{\partial L}{\partial a^{[1]}}$
$\times \frac{\partial z^{[2]}}{\partial a^{[1]}}$
$\frac{\partial L}{\partial z^{[2]}}$
$\times \frac{\partial a^{[2]}}{\partial z^{[2]}}$
$\frac{\partial L}{\partial a^{[2]}}$
$\times \frac{\partial L}{\partial a^{[2]}}$
$\frac{\partial L}{\partial L} = 1$
L

Figure6: Forward and Backward propagation for L-layers Neural Net model

Then we do the most important project of this course, we build a 4-layer neural networks with 12288, 20, 7 and 5 dimensions, and train it to the task of cat vs non-cat classification. And for the activation functions, we use ReLU function for the first 3 layer, and sigmoid function for the last layer.

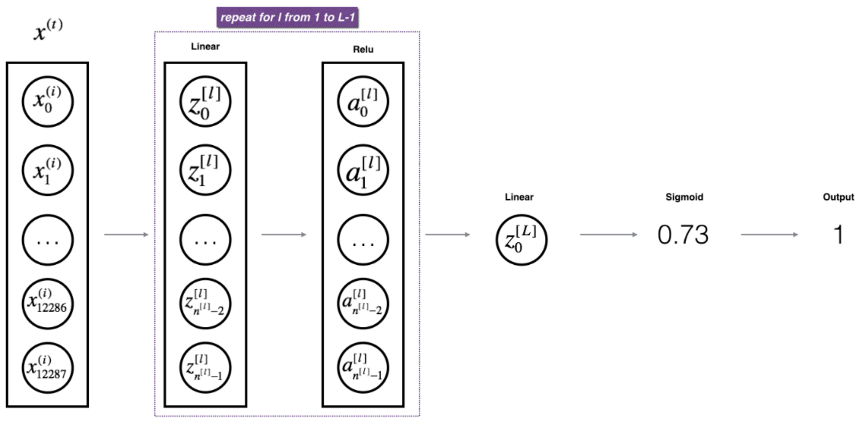


Figure7: Result of cat vs non-cat classification

To compare with the result of the logistic regression model, the accuracy increase from 68% to 80%.

3 Improving Deep Neural Networks

This part is important for getting deep learning to work better. This second course of the Deep Learning Specialization include initialization, L2 and dropout regularization, Batch normalization, and some optimization algorithms, such as mini-batch gradient descent, Momentum, RMSprop and Adam. I just finish two weaks of this course, there are the main points below.

3.1 Initialization

Training a neural network requires specifying an initial value of the weights. A well chosen initialization method will help learning to speed up the convergence of gradient descent and increase the odds of gradient descent converging to a lower training error.

With the different ways to initialize the parameters, we can see that a big difference:

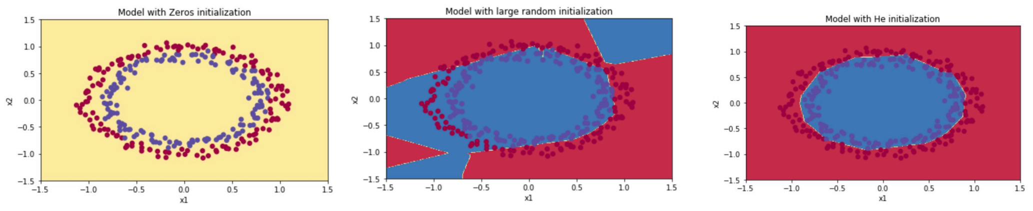


Figure8: Various Initialization

For this case, if we initialize parameters as 0, the model we get will be very bad. With random initialization, the result is better. And with He method(Random initialization multiply with $\sqrt{\frac{2}{dimension\ of\ previous\ layer}}$), we have the best model.

3.2 Regularization

Sometimes, our model could get the overfitting issus, which means the model has a low training error but perform bad for the test dataset. Here we learn different regularization methods include L2 and dropout regularizatin.

L2 adds "squared magnitude" of coefficient as penalty term to the cost function.

$$Cost\ function = Cost\ function + \frac{1}{2}\frac{\lambda}{m}W^2$$

And we also need to implement the changes in backward propagation to take into account regularization.

$$dW = dW + \frac{d}{dW}(\frac{1}{2}\frac{\lambda}{m}W^2) = dW + \frac{\lambda}{m}W$$

Finally, dropout is a widely used regularization technique that is specific to deep learning. It randomly shuts down some neurons in each iteration. So for each neuron unit, it will become less sensitive to the activation. So it used to fix the overfitting problem.

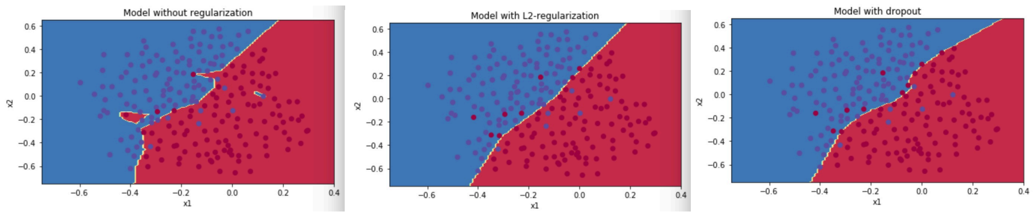


Figure8: Various Initialization

3.3 Optimization

Mini-batch gradient descent

Nowadays, big data is a very porpular domain, but Deep Learning does not work best in a this regime. When we neural networks on a huge data set we find that it's really slow. But with mini-batch gradient descent algorithm, we can speed up the efficiency.

The main idea is that split up your training set into smaller sets which are called mini-batches. For each mini-batche, the model could complete once learning iteration. So for one epoch(Traverse once dataset), it can take 5,000 gradient descent steps rather than 1 gradient descent steps in general batche gradient descent. So when we have a huge training set, mini-batch gradient descent runs much faster than batch gradient descent.

Momentum algorithm

Momentum compute an exponentially weighted average of the gradients, and then use that gradient to update the weights.

$$v_{dW} = \beta v_{dW} + (1-\beta)dW$$

$$v_{db} = \beta v_{db} + (1-\beta)db$$

$$W = W - \alpha v_{dW}, b = b - \alpha v_{db}$$

So when we do the gradient descend, it will gain momentum. So it could speed up learning for some parameters, at the same time slow down some others. So we can reach the convergence point quickly.

RMSprop

RMSprop stands for root mean square prop. It's idea is similar with Momentum algorithm except it adds an exponentially weighted average of the squares of the derivatives.

$$S_{dW} = \beta S_{dW} + (1-\beta) dW^2$$

$$S_{db} = \beta S_{db} + (1-\beta) db^2$$

$$W = W - \alpha \frac{dW}{\sqrt{S_{dW}}}, b = b - \alpha \frac{db}{\sqrt{S_{db}}}$$

Adam optimization algorithm

Adam optimization algorithm is basically taking momentum and RMSprop and putting them together. Adam is one of those rare algorithms that has really stood up, and has been shown to work well across a wide range of deep learning architectures.

$$v_{dW} = \beta v_{dW} + (1-\beta) dW,\ v_{db} = \beta v_{db} + (1-\beta) db$$

$$S_{dW} = \beta S_{dW} + (1-\beta) dW^2,\ S_{db} = \beta S_{db} + (1-\beta) db^2$$

$$W = W - \alpha \frac{v_{dW}}{\sqrt{S_{dW}} + \epsilon}, b = b - \alpha \frac{v_{db}}{\sqrt{S_{db}} + \epsilon}$$

3.4 Tool and Language

Jupiter iPython

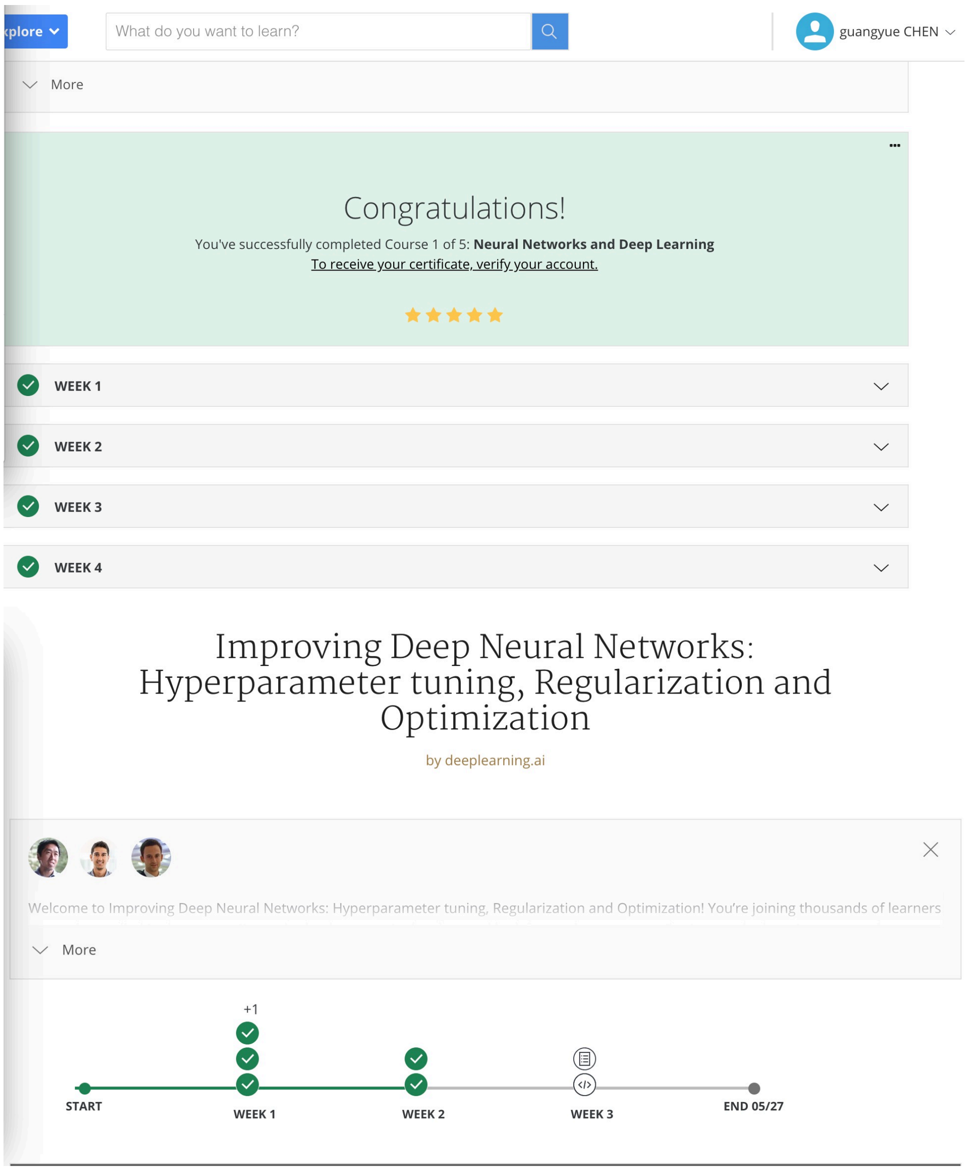
This course use Jupiter iPython notebook for the programming exercises, it has the interactive command shell nature which is very useful for learning quickly. We use the package 'numpy' and 'math' for computing, 'scipy.io' and 'sklearn' for the datas, and 'matplotlib.pyplot' for show the result.

4 Conclusion

After a period of study, I learned about the most effective deep learning techniques, and gain practice implementing them and getting them to work for myself. From this course, I find that there are still many interesting things during this domain, I will keep learning and make myself to be a better data scientist. To conclusion, I benefit a lot from this course.

5 Appendix

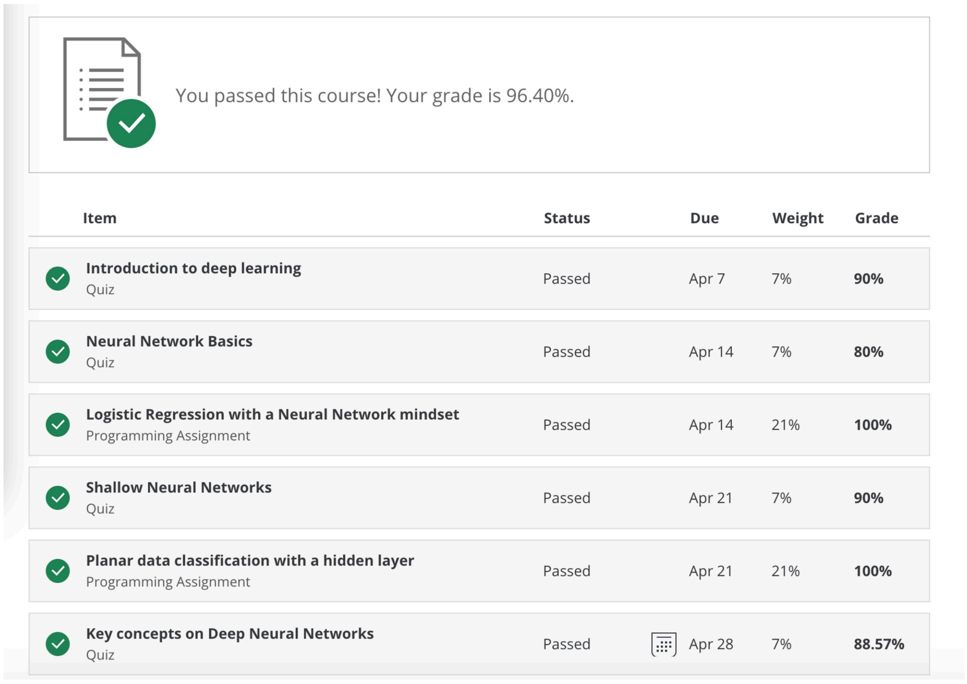
Course:https://www.coursera.org/specializations/deep-learning
After 6 weeks studying, I passed this course. My final resault is 96/100.



Congratulations!
You've successfully completed Course 1 of 5: **Neural Networks and Deep Learning**
To receive your certificate, verify your account.

Improving Deep Neural Networks: Hyperparameter tuning, Regularization and Optimization
by deeplearning.ai

Welcome to Improving Deep Neural Networks: Hyperparameter tuning, Regularization and Optimization! You're joining thousands of learners



You passed this course! Your grade is 96.40%.

	Item	Status	Due	Weight	Grade
✓	**Introduction to deep learning** Quiz	Passed	Apr 7	7%	**90%**
✓	**Neural Network Basics** Quiz	Passed	Apr 14	7%	**80%**
✓	**Logistic Regression with a Neural Network mindset** Programming Assignment	Passed	Apr 14	21%	**100%**
✓	**Shallow Neural Networks** Quiz	Passed	Apr 21	7%	**90%**
✓	**Planar data classification with a hidden layer** Programming Assignment	Passed	Apr 21	21%	**100%**
✓	**Key concepts on Deep Neural Networks** Quiz	Passed	Apr 28	7%	**88.57%**