SUMMARY OF COURSERA COURSE

IMPROVING DEEP NETWORK: HYPERPARAMETER TUNING, REGULARIZATION AND OPTIMIZATION (3 WEEKS)

RATINGS: 4/5

WEEK 1 – PRACTICAL ASPECTS OF DEEP LEARNING

UNIT 1: Train/dev/test sets

- Practical ways to make our deep learning work better
- Applied ML is a highly iterative process (idea > code > experiment)
- You have to make lot of choices for your hyper parameters
- DL applications: NLP. Computer vision, speech, structured data
- Intuitions from one domain don't translate to another domain most times
- How efficient is your iterative process?
- Data: train set, dev set, test set
- Most split the data as 70% train set, 30% test set
- In Big Data era where data size ranges from 1,000,000 our "dev set" can be about 10,000 (1%)
- Set up your data into TRAIN, DEV and TEST sets
- Make sure the DEV and TEST set come from the same distribution
- It might be okay not to have a TEST set, a DEV set alone might be sufficient

UNIT 2: Bias/Variance

- High Bias > Under fitting
- High Variance > Over fitting
- Train set error and Dev set error is used to get a feel of the bias and variance
- "High Bias and High Variance" is the worst of both worlds

UNIT 3: Basic Recipe for Machine Learning

- Check for High Bias (training data performance) > Bigger Network/More Layer/NN architecture search
- Bigger network mostly reduces bias without hurting variance, more data most times reduces variance without hurting bias
- Training big neural network is computationally expensive
- Using regularization reduces variances, it might affect the bias a little bit but with a big NN it is negligible

UNIT 4: Clarification about Upcoming Regularization Video

UNIT 5: Regularization

- Regularization helps to deal with high variance problems
- We usually add a “regularization parameter” to our cost function to implement regularization
- “l2 regularization” is the most common, also called weight decay
- For NN we use “Frobenius Norm” with our L2 regularization

UNIT 6: Why Regularization reduces overfitting

- Because regularization tries to reduce the “W” parameters reducing the complexity of the model
- Remember to add the regularization term when plotting the Cost Function vs No of Iteration graph

UNIT 7: Dropout Regularization

- Dropout regularization is another powerful regularization technique
- It drop outs different set of nodes (neurons) in the hidden layers, helps to simplify the NN model
- Keep probs: amount of nodes to keep (probability)
- “Inverted Dropout” the most used implementation of dropout regularization
- No drop outs at test time so outputs will not vary

UNIT 8: Clarification about Upcoming Understanding Dropout Video

- Dimension of w_1 = 7x3, dimension of w_3 = 3x7

UNIT 9: Understanding Dropout

- “Cannot rely on any feature, so have to spread out weights”, which shrinks the weight
- Dropout can be proved to work similarly to L2 regularization
- You can apply drop out to input layer, not very common in practice
- Earlier applications of drop out were used in Computer Vision
- Disadvantage of Drop out is that we don’t have a well-defined J (cost function)

UNIT 10: Other Regularization Methods

- Getting more training data can be expensive
- We can flip our images horizontally, randomly zoom/distortions our images to increase data size
- “Data Augumentation” can be used as a regularization technique by flipping, zooming and randomly distorting our images
- “Early Stopping”: helps to stop iterations early there by reducing the weight size
- Early stopping combines both the task of optimizing cost function and Check for overfitting this is a disadvantage
- Optimize Cost function > Check for Overfitting = Orthagonalization Technique

- He prefers to just use “L2 regularization”

UNIT 11: Clarification about upcoming Normalizing Inputs Video

- Vector “x” should be divided by σ

UNIT 12: Normalizing Inputs

- Normalizing the input helps to speed up our training
- You want your train and test to go through the same transformation process
- If we normalize our data, our cost function will look more symmetric and our cost function will be easier to optimize
- It never hurts to normalize our input features

UNIT 13: Vanishing/Exploding Gradients

- One of the problem of training NN especially deep NN is data vanishing and exploding gradients, that is your derivatives or slope can sometimes get either too big or too small
- If W > 1: in deep NN the weight increase exponentially
- If W < 1: the weight decrease exponentially

UNIT 14: Weight Initialization for Deep Networks

- A partial solution to vanishing/exploding gradient is the careful choice of the random initialization of our weights
- Variance of W = 1/n useful for a tanh function, or 2/n for ReLU functions, it helps to reduce vanishing/exploding gradients
- “ReLU activation function” is the most common
- Reducing vanishing/exploding gradients help our Deep NN train more quickly

UNIT 15: Numerical Approximation of gradients

- “Gradient checking” is a test that help to make sure we are implementing our back propagation correctly
- Using two sided difference for computing our derivative is more accurate than using one sided difference

UNIT 16: Gradient Checking

- Gradient Checking helps to debug our back propagation implementation
- Take Ws and bs, reshape them into a big vector “theta”
- Take dWs and dbs reshape them into a big vector “dtheta”

UNIT 17: Gradient Checking Implementation Notes

- Don’t use grad check in training, only to debug
- If algorithm fails grad check, look at components to try to identify bug
- Remember regularization

- Doesn't work with drop out
- Run at random initialization, perhaps again after some training

UNIT 18: Quiz

UNIT 19: Programming Assignment 1 – Initialization

UNIT 20: Programming Assignment 2 – Regularization

UNIT 21: Programming Assignment 3 – Gradient Checking

UNIT 22: Yoshua Bengio Interview

- He worked on "Word Embedding"
- Many different ideas to try out in unsupervised learning, it leaves a wide field of possibilities
- More still to be done in AI, most AI are still working at the superficial level
- Experiments to better understand the workings of current deep NN
- Deep Learning Engineer and Deep Learning Researcher
- Reading others people code, understanding what you are doing, write your codes
- Within 6 months you can start working well in Deep Learning Research

WEEK 2 – OPTIMIZATION ALGORITHMS

UNIT 1: Mini-batch gradient descent

- Learn about optimization algorithms that will enable us to train your NN much faster
- The size of our dataset affects how fast vectorization occurs
- "Batch Gradient Descent": you process your entire training set at the same time
- "Mini-batch GD": you process a single mini-batch of the training set at the same time
- "1 epoch" means a single pass through a training set
- Mini-batch works better than batch gradient descent for large data sets

UNIT 2: Understanding Mini-batch gradient descent

- The graph of Cost versus No of mini-batch trends downward but it is noisy
- You have to choose the size of your mini-batch
- Stochastic GD: every example is a mini-batch, similar to setting mini-batch size to 1, you lose all your speed up from vectorization
- Choose a mini-batch that is not too big or too small (64,128, 256, 512)
- If the training set is small (< 2000) just use "Batch GD"
- Make sure mini-batch fit in CPU/GPU memory

UNIT 3: Exponentially Weighted Averages

UNIT 4: Understanding Exponentially Weighted Averages

- It is a key component for several optimization algorithms
- It takes very little memory

UNIT 5: Bias Correction in exponentially weighted averages

- Bias correction makes computation of the weighted average more accurate, it is really needed at the initial stage of the EWA

UNIT 6: Gradient Descent with Momentum

- Momentum or GD with momentum works faster with default GD
- Momentum works with both Batch and Mini-batch GD
- Most common value for "β" is 0.9, people don't usually use "bias correction" when implementing Momentum

UNIT 7: RMSprop

- "RMSprop": Root Mean Square Prop, it can speed up GD
- It dampens out the oscillation in Gradient Descent

UNIT 8: Clarification about upcoming Adam Optimization

UNIT 9: Adam Optimization algorithm

- Adam Optimization basically combines Momentum and RMSprop
- People usually just tune the "learning rate" leaving the others as default
- "Adam": Adaptive Moment estimation

UNIT 10: Clarification about Learning rate decay video

UNIT 11: Learning rate decay

- Reducing learning rate over time might help speed up the learning algorithm (learning rate decay)
- "1 epoch": means one pass through the data
- Manual decay
- Learning decay is one of the last things he will try

UNIT 12: The problem of local optima

- Local optima, Global Optima
- At high dimensional space it is less likely to experience bad local optimal
- Plateaus can make learning slow (optimization algorithm like Adam really help here)

UNIT 13: Quiz

UNIT 14: Programming Assignment – Optimization

UNIT 15: Yuanqing Lin Interview

- He is possibly the number 1 Deep Learning guy in China
- You can start with free Deep Learning Frameworks

WEEK 3 – HYPERPARAMETER TUNING, BATCH NORMALIZATION AND PROGRAMMING FRAMEWORKS

UNIT 1: Tuning Process

- A lot of hyper parameters to deal with in NN, but some are more important than the others
- "learning rate" > "momemtum rate, no of hidden units, mini-batch size"
- He advises to try random sampling (random combinations of hyper parameters) instead of using a grid
- "Coarse to fine": coarse sample the grid area> then focus on a smaller area of the grid area

UNIT 2: Using an appropriate scale to pick hyper parameters

- You should use log scale instead of linear scale to sample hyper parameters like learning rate, theta (exponentially weighted averages)

UNIT 3: Hyper parameters tuning in practice – Pandas vs Caviar

- Intuitions do get scale, re-evaluating our hyper parameters occasionally
- "Babysitting One Model": used mostly where the data is large, and there is limited computational resource "Pandas"
- "Training many models in parallel": if we have sufficient computational resource you can try a lot of different hyper parameters combinations "Caviar"

UNIT 4: Normalizing activations in a network

- Batch Normalization makes hyper parameter search easier and make NN more robust
- Can we normalize the hidden units?
- Normalizing "Z" is the most common approach
- Batch Normalization applies normalization to some values of the hidden units

UNIT 5: Fitting Batch Norm into a Neural Network

- In practice you don't have to implement everything from scratch
- In mini-batch you perform the batch norm for each mini-batches
- No need for the parameter "b" when doing Batch Normalizations
- Batch Normalization works with Momentum, RMSprop and Adam

UNIT 6: Why does Batch Norm Work

- Make weights in deeper networks more robust to changes
- Covariate Shift: if we map X > Y, if the variance of X changes we will need to retrain our model on this, batch norm reduces the covariate shift of hidden layer values by ensuring the mean and variance remain the same
- Batch Norm makes the deep NN more stable, it has a slight regularization effect
- He will not use Batch Norm as a regularization technique
- Use to normalize and speed up our learning

UNIT 7: Batch Norm at test time

- Batch norm processes your data one mini batch at a time, but the test time might need to process the examples one at a time
- For test time we use the lastest computed values of "σ" and "μ" abd the derived values of "β" and "gamma" from the training set

UNIT 8: Clarification about Upcoming Softmax Video

UNIT 9: Softmax Regression

- Softmax Regression: is a generalization of logistic regression capable of multiple class prediction
- Using softmax activation in the output layer helps to output probabilities for multiple class prediction

UNIT 10: Training a softmax classifier

- In "hardmax" the biggest value in the vector gets mapped to "1" and others to "0", but "softmax" returns the probabilities
- We will start using programming frameworks, in the frameworks you only have to focus on getting the Forward Prop right, it takes care of the back prop

UNIT 11: Deep Learning Frameworks

- It not always practical for you to implement everything from scratch
- But it helps to know the underlying principles
- Deep Learning Frameworks: caffe/caffe2, Keras, PaddlePaddle, Tensorflow, Torch, Lasagne, Theono
- Criteria for choosing frameworks: Ease of programming (development and deployment), Running speed, Truly open (open source with good governance)

UNIT 12: TensorFlow

- "import tensorflow as tf"
- By implement Forward Prop in Tensorflow it can automatically implement Back Prop
- "session = tf.Session" > "session.run(init)" > "session.run(w)"
- You can do a lot with one line of code in a Deep Learning Frameworks

UNIT 13: Quiz

UNIT 14: Programming Assignment – Tensorflow

- Tensorflow steps: “create tensors (variables)” > “write operations btw those tensors” > “initialize your tensors” > “create a session” > “run session”
- Construct Computation Graph > Run Computation Graph