Improving Deep Neural Networks: Hyperparameter tuning, Regularization and Optimization

- 1. Practical Aspects of Deep Learning
  - DL is a highly iterative process, you have a lot of things and hyperparameters to take care of
  - Setting up data in train/dev/test sets
    - Ckt: 70/30 or 60/20/20
      - small dataset? these ratios will be okay
    - Because dl has a lot of data, 95/5 will also work
      - depends on the size of data
    - Make sure dev and test set come from the same distribution
  - Bias/Variance
    - 2D data: we can plot the line and see if bias or variance is there
    - What is Optimal/Baye's error?
      - High dims data?
        - Training error - baye's error small, train - dev error big?
          - Overfitting
            - High variance
        - Training error - baye's error big, train-dev error small?
          - Underfitting
            - High bias
        - Training error - baye's error big, train-dev error big?
          - Worst of both worlds
            - High bias and variance
        - Training error - baye's error small, train-dev error small?
          - 'Best of both worlds
            - Low bias low variance
        - The optimal error will define the bias or variance or both problem.
  - Solving problem of bias and variance when not low
    - Basic recipe for machine learning
      - After training initial model
      - 1. Ask if the model has high bias?
        - try bigger network
        - almost always helps
        - train longer
        - try different NN architecture
        - more training data will not help here
      - 2. Ask if the model has high variance?
        - get more data
        - regularization
        - more appropriate NN architecture
    - Dealing with high variance problem
      - Regularizing your neural network
        - L1 regularization
          - W may be sparse, avoid
        - L2 regularization
          - in NN application it becomes Forbenius Norm
          - in backprob, "weight decay" happens
        - Dropout Regularization
          - Inverted Dropout (Most Common)
          - Cost function not clearly defined
          - Use only in the training time
        - Data Augmentation
          - Getting more data is tough
          - Augment the data
            - Flip
            - Rotate
            - Crop
            - Skew, etc.
        - Early Stopping
          - Orthogonalization
            - Downside of early stopping
          - lambda is very large, weights become smaller, Z cores in smaller range of values and hence NN becomes much simpler
          - Why does regularization help with overfitting?
  - Setting up your optimization problem
    - Normalizing Inputs
      - 1. Subtract the mean (1/m)*np.sum(x^(i))
      - 2. Normalize the variance (x = x / sigma^2) (Sigma = (1/m) * np.sum(x^(** 2))
      - Use same mean and variance to normalise the test set
      - The cost function looks elongated when not normalized and hence gradient descent becomes tough to perform
      - Input features are on different scales, normalization becomes very important
      - anyway always normalize
    - Vanishing and exploding gradients
      - Weight initialization to tackle this problem
        - Random initialization
        - Xavier Initialization
          - If tanh activation
        - He initialization
          - If relu activation
            - np.sqrt(np.divide(2, n^[l-1]) = n^l[l]
    - Gradient checking
      - In backprob we use this to check if it is working correctly or not
        - Helps find bugs in Backprop
        - Helps save time
        - Find limit ---> euclidean distance
        - Checks distance between gradient calculated manually and by the backprop.
      - Practical Tips
        - Don't use in training, only to debug
        - If algo fale grad check, look at components to Try to identify bugs
        - Remember regularization
        - keep_prop = 1.0
          - doesn't work with dropout
        - run at random initialization, perhaps again after some training

- 2. Optimization Algorithms
  - Gradient Descent
    - Analogy: "Gone Downhill"
  - Stochastic Gradient Descent
    - Batch Size = 1
    - Required 3 for loops to implement
      - 1. Over the number of iterations
      - 2. Over the m training examples
      - 3. Over the number of layers in the neural network
  - Mini-Batch Gradient Descent
    - Divide huge data into batches for training
      - 5 million data points? Divide by 1000, update parameters in GD after every mini batch is processed.
    - Choosing your mini batch size
      - Batch size = m (total training examples
        - Batch Gradient Descent
      - Batch size = 1
        - Stochastic gradient descent
      - mini-batch between 1 m
      - In practice
        - Why Mini-Batch GD is good
          - BGD takes too long
          - SGD: you lose the speed up from vectorization
          - MOGD
            - In practice gives faster learning
            - Vectorized implementation
            - Less waiting
        - make sure mini-batch size fits in memory of GPU or CPU
        - Shuffling and partitioning are two steps required to build mini-batches
      - Training Set Size
        - <= 2000
          - Use batch gradient descent
        - Bigger?
          - Mini-batch sizes of power 2^n (64, 128, 256, 512)
    - You may not converge everytime
      - Trend should be going down
      - Hyperparameter tuning
        - Learning rate for example
  - Exponentially Weighted Averages
    - Key equation
  - GD with Momentum
    - Almost always works faster than Batch GD
    - Use exponentially weighted average of gradients and use those gradients to update your weights
      - V(t) / 1-B^t : for removing initial bias in EWA
    - Smoothes out the steps of gradient descent
    - Value of Beta
      - larger the value, smoother the update
      - Common values
        - 0.8 - 0.999
        - Don't feel like tuning? 0.9 is mostly used
        - May need several attempts to find out the right value of beta for your model.
    - Tuning of alpha with beta is common
  - RMS Prop
    - S(dW) = beta * S(dw) + (1-beta)* dW^2 (element wise)
    - S(db) = beta * S(db) + (1-beta)* db^2 (element wise)
      - dW = dW / sqrt(S(dW))
      - db = db / sqrt(S(db))
  - Adam
    - Adam = Adaptive Moment Estimation
    - Mix of momentum and RMS Prop
    - One of the most effective optimization technique
    - parameters
      - alpha
      - beta1
      - beta2
      - apsilon
    - Advantages
      - Relatively low memory requirements
      - Usually works well even with little tuning of hyperparameters (except learning_rate)
  - Learning Rate Decay
    - By formula
  - The problem of local optima
    - Problem of plateaus
      - can be taken care of by different optimization algorithms

- 3. Hyperparameter tuning, Batch normalization and Programming frameworks
  - Hyperparameter Tuning
    - some hyperparameters are more important that others
      - learning rate is the most important
      - next maybe the momentum term
      - 3rd mini batch size, etc.
    - to find the right parameters, you gotta find them randomly
      - coarse to fine scheme
        - do random sampling
    - Use an appropriate scale to pick hyperparameters
      - Drastically different lower and upper limits?
        - use Log scale than linear scale
    - Exponentially weighted average hyperparameter
      - Can't be randomly chosen because very sensitive
    - In practice:
      - Pandas approach
        - Babysit one model
      - Caviar approach
        - Train multiple models simultaneously
      - Approach selection will depend on availability of computation power
    - Batch Normalization
      - Makes hyperparameters search easier
      - Makes NN more robust
        - Enables to train large NNs easily
      - Implementation
        - Normalize Z terms
          - z(i) = gamma * z(i)norm + beta
            - gamma and beta are learnable parameters using GD/Adam/RMSProp/Momentum
        - Instead of using unnormalized values you use normalized values of Z
        - batch norm is applied using mini batch gradient descent
        - Why BN works?
          - It makes weights deeper in the NN more robust to change
        - BN has a slight regularization effect though it should not be used as a regularization technique
        - BN at Test time
          - mu and sigma^2 are calculates using exponentially weighted averages across mini-batch
            - Subtopic 1
    - Softmax Regression
      - Generalization of Logistic Regression
      - Recognize 1 out of c classes
      - Algorithm
      - Training a softmax classifier
        - Changes in the label (make it no.of final classes x 1 vector)
        - Normally programming frameworks will take care of the complex operations
  - Deep learning Frameworks
    - Caffe / Caffe2
    - DL4J
    - CNTK
    - Keras
    - Lasagne
    - mxNet
    - Paddle-Paddle
    - Tensorflow
      - Dld in the assignment
    - Theano
    - Torch

XMind: ZEN | Trial Mode