TRAINING NEURAL NETWORKS

Shingchern D. You

Reading assignment

- You are strongly recommended to read the lecture notes of CS231n
- I can only extract some key points to present here (of course with some personal prejudices)

Some points in training NN

- Data augmentation
- Data pre-processing (normalization)
- Shuffle
- Weight initialization
- Batch normalization
- Regularization and learning rate
- Activation function
- Performance plots
- Class imbalance
- Ensemble

Data augmentation

- Data augmentation means to artificially generate additional training samples so we have more training data
- Why large size of training data is preferable
- In systems of equations, if we have more variables than equations, many possible solutions exist, and some of them can be bad
- More training data (better) prevents overfitting

Data augmentation

- There is no standard method to perform data augmentation
- In the case of audio signal, we can
  - Add noise
  - Raise/lower pitch of the audio (spectral scaling)
  - Increase/decrease tempo (temporal scaling)
- In the case of hand-written characters, we may add distorted characters

Data augmentation

- Apple ML lab reported a method to use two neural networks to generate artificial “facial” pictures
- One network is used to generate fake pictures
- The second network tries to identify if a fake picture is presented or not
- Both competing with each other
- In fact, it is a form of generative adversarial network (GAN)

Data augmentation

- One word of warning: If the performance (accuracy) increases, it is likely because the augmented data have some similarity to the test dataset
- If the augmented data are not related to the test data, accuracy may not improve
- In the end, no one knows whether we have done data augmentation correctly or not

Data pre-processing

- Usually make data zero mean by treating each feature of the sample points as outcomes from iid RV
- For example, the Iris dataset has four features. Therefore, we compute the mean value of each feature based on the training dataset
- If necessary, we may also perform normalization for values in each feature
- One widely setting is to ensure values are in [-1,1]

Data pre-processing

- Note: in the above argument, one pixel in a picture is one feature
- Therefore, computing "mean value" is to find the average gray level of pixels in the same location across all pictures
- Sometimes, we have another type of normalization: across one picture. Used especially if a particular picture is over-exposed (or vice versa)

Data pre-processing

- Data pre-processing may also include some sort of dimensionality reduction (as well as whiting), such as using PCA
- If the number of training data is large enough, we prefer to use “raw” data as input to neural networks
- Inputting raw data allow the network to learn classification as well as feature extraction

Shuffle

- Shuffle means to randomize the presentation order of training data for each epoch
- Shuffle is crucial for some neural network architectures, such as convolutional LSTM
- Thus, it is recommended to perform data shuffling during training
- Why shuffle is useful? One explanation is related to paths of gradient search

Weight initialization

- The neural network will NOT work if all weights are initially set to zero (why?)
- Initializing all weights with uniform distribution between (-1,1) is not good if the "fan-in" of a node is large
- Textbook suggested using small random numbers
- Another suggestion is to use N(0,1) * sqrt(2/n), where n is the number of fan-in (especially for ReLU units)

Weight initialization

- Why large initial weights are no good for a node with large number of inputs?
- Chances are that the node is dead from beginning (ReLU can easily “die”)

Batch normalization

- Batch normalization alleviates the headache of properly initializing weights
- In the implementation, applying this technique usually amounts to insert the BatchNorm layer immediately after fully connected layers (or convolutional layers, as we'll soon see), and before non-linear functions
- Perform normalization across all samples within the same mini-batch for each node

Batch normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Batch normalization

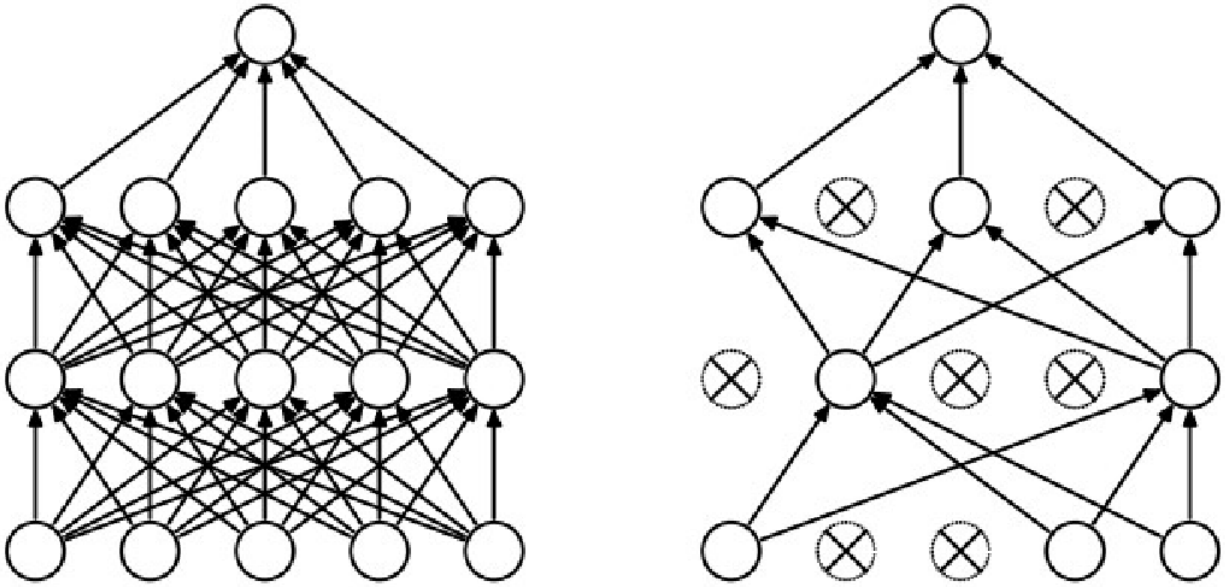
- During batch normalization, parameters γ and β are learned
- Applied to a state-of-the-art image classification model, Batch Normalization achieves the same accuracy with **14 times fewer training steps**, and beats the original model by a significant margin
- You may want to read the original paper published by Google

Regularization

- L1 norm (Lasso)
- L2 norm
- Max norm
- Dropout

Dropout

- Randomly drop some nodes (i.e., set to zero)
- Train only the retained portion of network (weights)



(a) Standard Neural Net

(b) After applying dropout.

Dropout

- Usually perform one random selection per mini-batch
- Typically $p = 0.5$ (dropping rate = 50%)
- During testing (prediction), no dropout is used
- Therefore, more nodes and weights in layer k are used to compute the output in layer $(k+1)$
- Consequently, we need to **scale all weights by the same factor (i.e., p) during testing**

Dropout

- We show that the dropout regularizer is first-order equivalent to an L2 regularizer applied after scaling the features by an estimate of the inverse diagonal Fisher information matrix – From paper Dropout Training as Adaptive Regularization

Regularization

- Practical recommendations:
  - L2 norm + dropout $p = 0.5$
  - Use global regularization (not per layer one)
  - No regularization on bias weights

Learning rate

- In a first attempt, we may use a fixed learning rate (eg. 0.1) throughout training. Observe the validation accuracy, then try reducing (or increasing) learning rate by 2 (or even 5) to see if any improvement occurs
- Alternatively, it might be usually helpful to **anneal** the learning rate over time in training deep networks

Learning rate

- Approaches to anneal learning rate
  - Step decay: reducing the rate by 2 every, say, 5 epochs
  - Exponential decay
  - (1/t) decay
- Step decay seems a little bit better
- Another possibility is to use adaptive learning rate (adapt to per weight learning rate)

Newton's method

- A second-order learning method is Newton's method

$$\boldsymbol{w}_{(k+1)} \leftarrow \boldsymbol{w}_{(k)} - [Hf(\boldsymbol{w})]^{-1} \nabla f(\boldsymbol{w})$$

where $[Hf(\boldsymbol{w})]$ is the Hessian matrix, 2nd partial derivative of function

- Newton's method is fast, but requires matrix inverse (not plausible for a big network)
- In a sense, the momentum is an approximation of the Hessian matrix

Adaptive learning rate

- Basic idea: Weights that receive high gradients will have their effective learning rate reduced, while weights that receive small or infrequent updates will have their effective learning rate increased
- Some used methods
  - Adagrad
  - RMSprop (a leaky Adagrad)
  - Adam (like RMSprop with momentum)

Adaptive learning rate

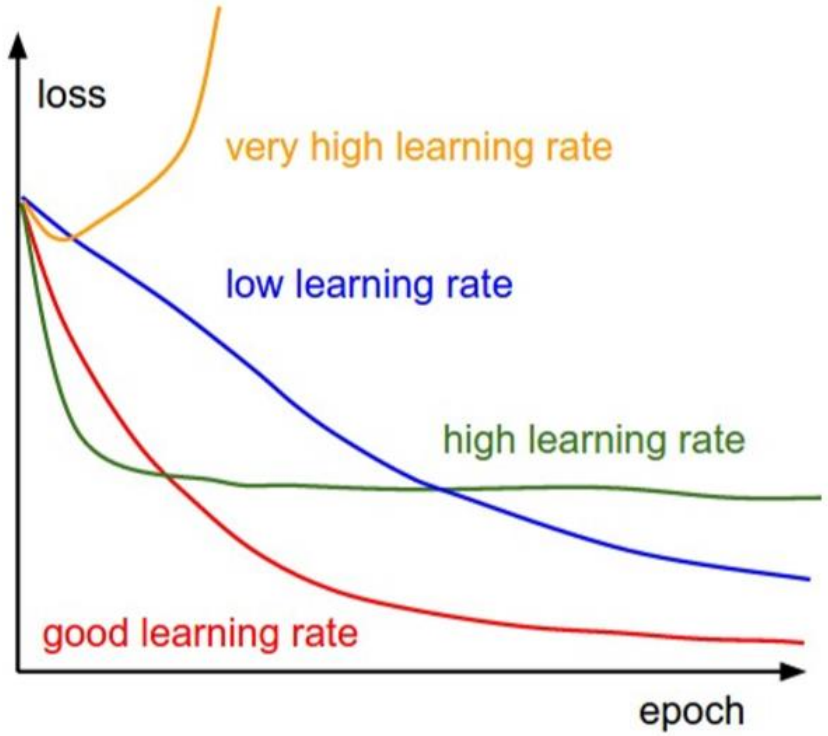
- Currently, Adam is used as a default learning algorithm (recommended in CS231n)
- Although we find that sometimes Adam is not good when compared with **AdaDelta** (another approximation method to Newton's method)
- There are still some hyperparameters to set in Adam (despite the name "adaptive")

Activation and loss function

- We mentioned this part before
- Usually ReLU or leaky ReLU is preferred (than the Sigmoid)
- Cross-entropy loss function is the default setting in most training

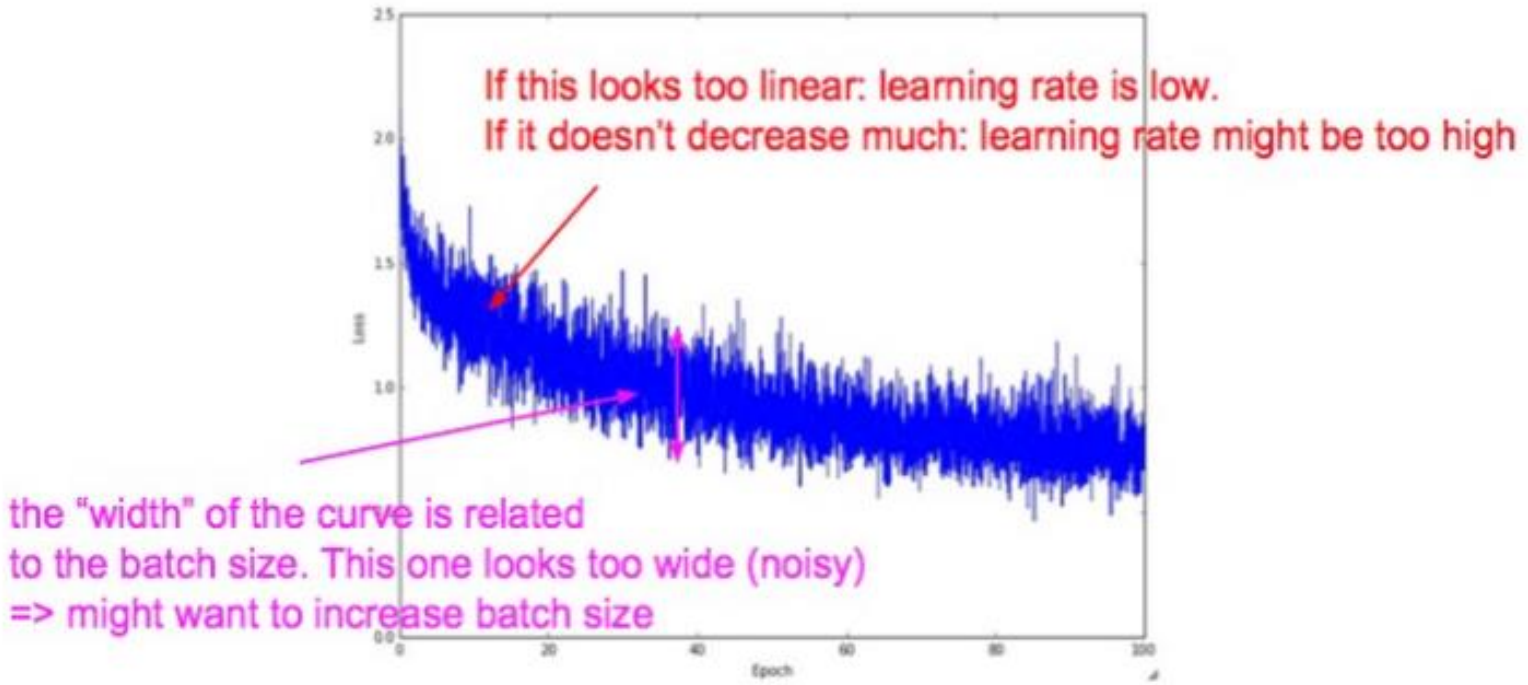
Performance plot

- We may gain some knowledge by plotting the loss function (as well as validation accuracy)



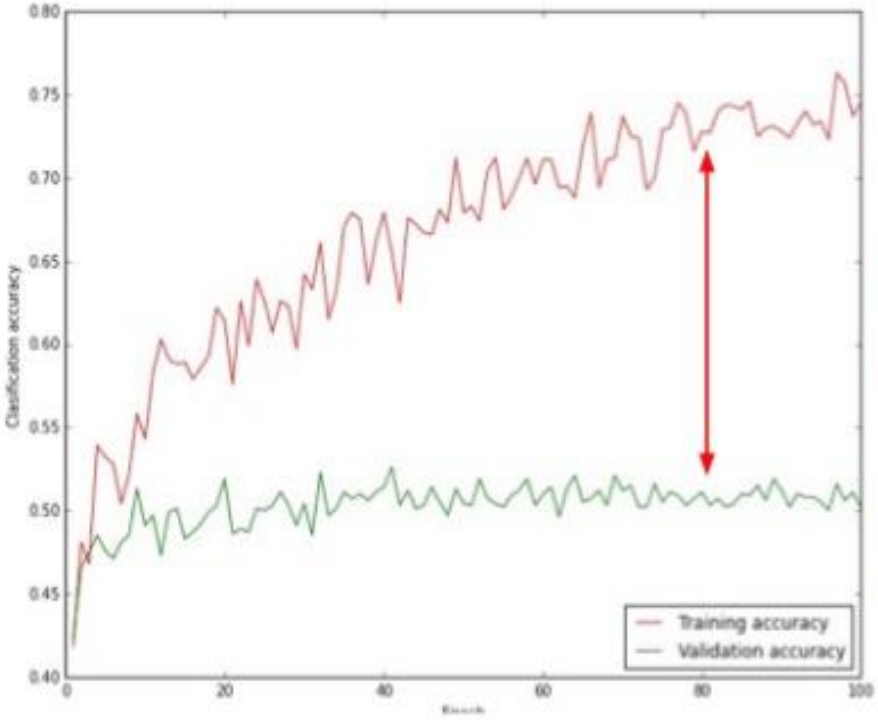
Performance plot

- Actual plot of loss function (source: http://lamda.nju.edu.cn/weixs/project/CNNTricks/CNNTricks.html)



Performance plot

- Accuracy between validation and training



big gap = overfitting
=> increase regularization strengt

no gap
=> increase model capacity

Class imbalance

- It is important to check the number of training samples in each class
- Different number of samples in each training class is called class imbalance
- For SVM, the classifier strongly favors the class with more samples (though adjustable)
- In the neural network case, the situation is not that obvious
- However, this problem deserves some attention

Ensemble

- If we want to further improve the classification accuracy, we may use ensemble technique
- There are several methods to perform ensemble classifications
  - Same model with different initialization weights
  - Train several networks with same architecture using partial training samples
  - Train several networks with different architecture using all training samples

Ensemble

- In our experience, we tried ensemble with different models
- The results seem good (better than any single model)
- Even less accurate classifiers can contribute to the overall improvement (e.g., 93% with 90%, 88% and 80% classifiers)