Deep learning - Regularization

Prateek Gupta

June 2015

1 Introduction

1. Improve generalization error (error rate observed on Out of Sample Data or validation data)

2. Putting extra constraints that are designed to 'not help' fit the training set

3. Trade off increased bias for reduced variance

$$MSE(\tilde{\theta}) = Variance(\tilde{\theta}) + (E[\tilde{\theta}] - \theta)^2 + \epsilon^2$$

where, $\tilde{\theta}$ is an estimator of θ coming from update equations or solution of optimization procedure. Variability in $\tilde{\theta}$ is because of randomness in data and bias is due to model mismatch.

Well known bias-variance trade off- As complexity of the model is increased model mismatch(bias) is decreased while variance in the prediction is increased because of randomness in training inputs.

4. Deep Learning systems have large number of parameters and therefore have high capacity(complexity) to fit relatively complicated functions.To address this problem either have huge dataset or carefully regularize parameters. Regularization plays a major rule in improving generalization error for such systems by limiting the domain of parameters. (induces bias and decreases variance because parameters are restricted to a domain)

2 Classical Regularization

Mathematically, the training objective function becomes:

$$\tilde{J}(\theta; X, y) = J(\theta; X, y) + \alpha\Omega(\theta)$$

where, $\Omega(\theta)$ is some function of θ such that each θ_i contributes positively such that optimization is penalized by the parameters.

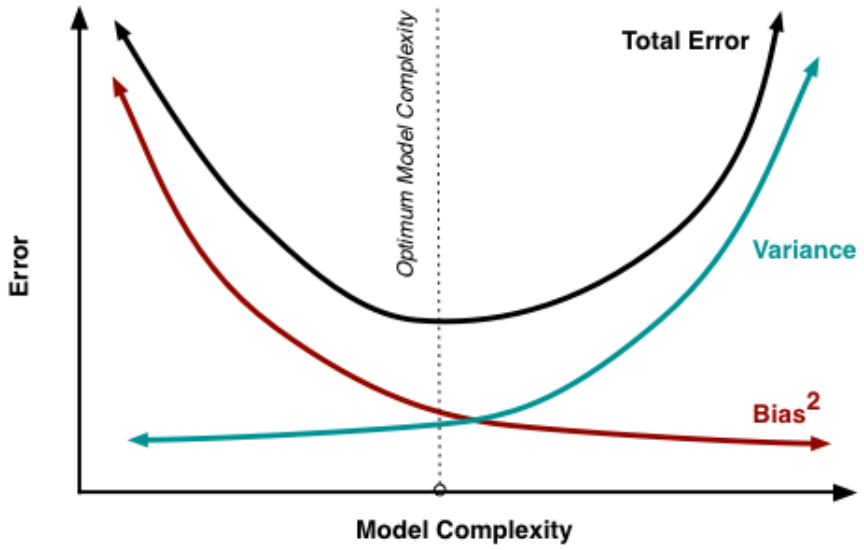

Figure 1: Bias Variance Decomposition

2.1 L^2 Regularization

$$\Omega(\theta) = \frac{1}{2}||w||_2^2 = \frac{1}{2}(w_1^2 + w_2^2 + ... + w_n^2)$$

- constant terms are not penalized because it is multiplied by a constant(i.e. 1). Thus, regularizing bias does not reduce variance by much and instead can increase bias a lot (underfitting)
- What happens mathematically?
  Denoting H by Hessian Matrix of J calculated at empirical optimal solution of J i.e. w*,

$$\nabla_w \tilde{J}(w; X, y) = \nabla_w J(w; X, y) + \alpha \nabla_w \Omega(w)$$
$$\nabla_w \tilde{J}(w; X, y) = H(w - w^*) + \alpha w$$
$$\alpha = (H + I\alpha)^{-1} H w^*$$

Eigen decomposition of H yields $H = Q\Lambda Q^T$ Thus,

$$Q^T w = (\Lambda + I\alpha)^{-1} \Lambda Q^T w^*$$

Looking in the same basis we find that original solution is shrunk by a factor of $\frac{\lambda_i}{\lambda_i + \alpha}$

  - $\lambda_i << \alpha$ implies high shrinkage i.e. gradient change is small so the parameters are shrunk to small values

– $\lambda_i >> \alpha$ implies no shrinkage i.e. wherever the gradient change is huge parameters are not affected

2.2 L^1 Regularization

$$\Omega(\theta) = ||w||_1 = |w_1| + |w_2| + ... + |w_n|$$
$$\tilde{J}(w; X, y) = \frac{1}{2}\gamma_i(w_i - w_i^*)^2 + \beta sign(w_i)$$
$$w_i = sign(w_i)max(|w_i^*| - \frac{\beta}{\gamma_i}, 0)$$

- regularization contribution to the gradient no longer scales linearly with w , instead it is a constant factor with sign equal to sign(w)
- $w_i^* \leq \frac{\beta}{\gamma_i} \Rightarrow w_i = 0$. Small w_i^* are forced to 0 inducing sparsity
- large w_i^* are just shifted by $\frac{\beta}{\gamma_i}$

3 Regularization with Explicit Constraints

Optimization procedure viewed as Lagrange objective function implying that penalties can be interpreted as constraints. Thus,

$$L(\theta, \alpha; X, y) = J(\theta; X, y) + \alpha(\Omega(\theta) - k)$$
$$\theta^* = \min_{\theta} \max_{\alpha, \alpha \geq 0} L(\theta, \alpha)$$
$$\theta^* = \min_{\theta} L(\theta, \alpha^*) = \min_{\theta} J(\theta; X, y) + \alpha^* \Omega(\theta)$$

Where, α^* is a function of $J(\theta)$ and k such that it increases when $||\theta||_p > k$ and decreases when $||\theta||_p < k$

- α is large $\Rightarrow$ k is small
- α is small $\Rightarrow$ k is large

Advantages of using explicit constraints

- Time : In stochastic gradient descent (SGD) we can stop whenever the constraints are satisfied instead of seaching for α that corresponds to k
- Preventing local minima: Penalties can cause non-convex optimization procedure to get stuck in local minima corresponding to small θ

  Neural Network ends up having lot of dead units i.e. those units do not contribute much to the behavior of function because weights going in and coming out of those nodes are all very small. When training with a penalty on the norm of the weights these configurations can be locally optimal even if it is possible to significantly reduce J by making weights larger

4 Under-Constrained Problems

Many algorithms require X^TX to be invertible. This is not always the case as in when number of input features are more than number of observations or when there is no variation in one direction. We end up inverting $(X^TX + \alpha I)$ instead.

5 Dataset Augmentation

- To generalize better, train on more and more data
- generate fake data either by transformation of inputs (images -rotation, flip, etc) or adding noise to inputs
- Neural Networks are not found to robust against noise. Thus, to improve robustness of NN
  - train with noise applied to inputs
  - apply noise to hidden units (dataset augmentation at multiple levels of abstraction)

6 Bagging

- Reduce generalization error by combining several models: Model Averaging by ensemble strategy
- Generalization error is reduced when errors from different models are uncorrelated which can be enforced by making models with different assumptions, hyperparameters or random start
- Assuming there are k models and ϵ_i is the error on an observation by model i, $E[\epsilon_i\epsilon_j] = c, E[\epsilon_i^2] = \nu$ then

$$e_i = \frac{1}{k}\sum_i \epsilon_i$$

$$E[e_i^2] = E[(\frac{1}{k}\sum_i \epsilon_i)^2] = \frac{1}{k^2}E[\sum_i(\epsilon_i^2 + \sum_{i\neq j}\epsilon_i\epsilon_j))] = \frac{\nu}{k} + \frac{k-1}{k}c$$

  Thus, when errors are uncorrelated, $c = 0$ thus,

$$E[e_i^2] = \frac{\nu}{k}$$

- Procedure
  - construct k datasets (each has same number of observations)
  - train k models

- Neural Networks have large number of parameters thereby reaching large number of solutions. These solutions results in practically uncorrelated errors because of random initialization, random selection of mini-batches and different hyperparameters. Thus Bagging in case of NN will help in reducing generalization error

7 Early Stopping

- Most commonly used in deep learning: Effective and simple
- also a hyperparameter selection algorithm
- no change in underlying procedure, objective function or set of parameter values
- requirement of validation set
- instead of looking for a local minimum we run the algorithm until the error on the validation set has not improved for some amount of time
- Classification problem has a loss function which differs entirely from its performance metric. This mismatch problem is solved by early stopping procedure as performance metric on validation set can be used instead of using same loss function on validation set.

8 Dropout

- inexpensive approximation to training and evaluating a bagged ensemble of exponentially many neural networks
- train the ensemble consisting of all sub-networks that can be formed by removing units from NN
- easily adapted to backpropagation
- Criple NN by removing hidden units stochastically
  - each hiddne unit is set to 0 with probability 0.5
  - use random binary mask
  - Prediciton : Masks are replaced by expectation. In single hidden layer, it is geometric mean of all models (i.e. all possible masks)

9 Multi-Task Learning

- improve generalization by pooling the examples arising out of several tasks
- Shared parameters improve statistical strength

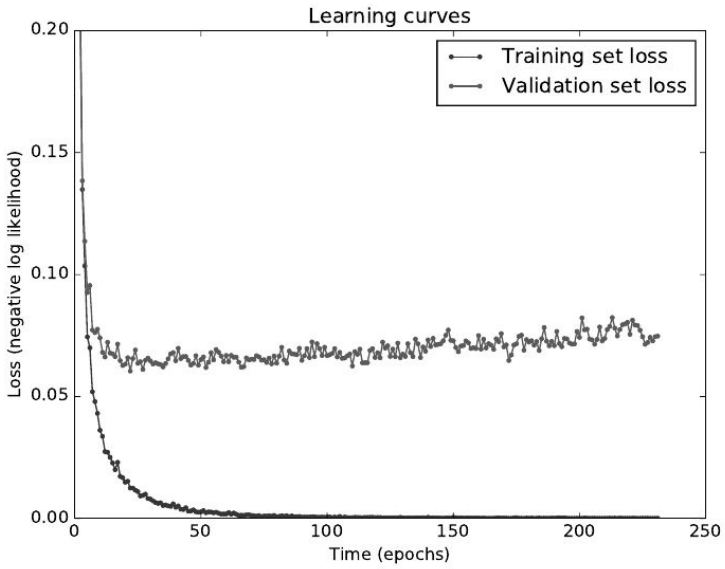

Figure 7.4: Learning curves showing how the negative log likelihood loss changes over time. In this example, we train a maxout network on MNIST, regularized with dropout. Observe that the training loss decreases consistently over time, but the validation set loss eventually begins to increase again.

Figure 2:

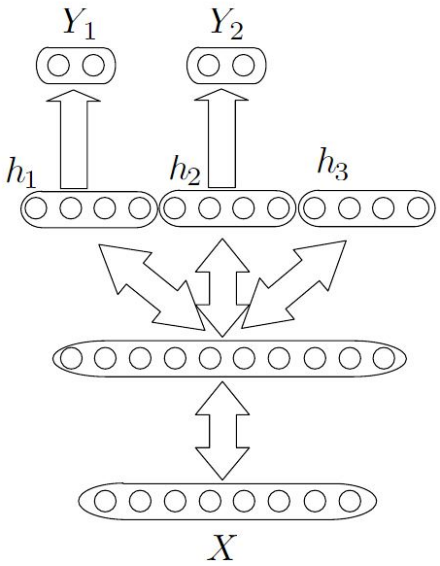

Figure 7.6: Multi-task learning can be cast in several ways in deep learning frameworks and this figure illustrates the common situation where the tasks share a common input but involve different target random variables. The lower layers of a deep network (whether it is supervised and feedforward or includes a generative component with downward arrows) can be shared across such tasks, while task-specific parameters can be learned on top of a shared representation (associated respectively with h_1 and h_2 in the figure). The underlying assumption is that there exist a common pool of factors that explain the variations in the input X, while each task is associated with a subset of these factors. In the figure, it is additionally assumed that top-level hidden units are specialized to each task, while some intermediate-level representation is shared across all tasks. Note that in the unsupervised learning context, it makes sense for some of the top-level factors to be associated with none of the output tasks (h_3): these are the factors that explain some of the input variations but are not relevant for these tasks.

Figure 3:

- In deep learning among the factors that explain the variations observed in the data associated with different tasks some are shared across 2 or more tasks
- Any model can be divided into two kinds of parameters:
  - Task-specific parameters - upper layers of NN (benefits from the examples of their tasks to achieve good generalization)
  - Generic parameters - shared across all the tasks (benefits from pooled data of all the tasks)