**MA690: Advanced Topics in Data Analysis and Computation** Novemeber 16

# Neural Nets: An Introduction

*Prof: Xiuyuan Cheng* *Scribe: Chris Arora*

## 1 Overview

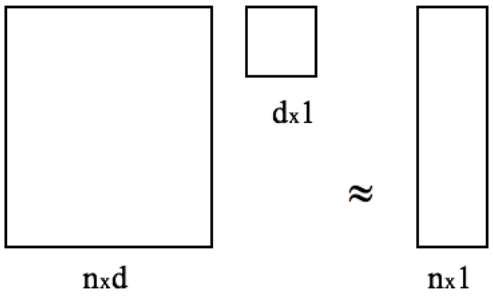
These notes will finish up our discussion on neural network approximation, discuss different methods for regularization, dive deeper into methods for adding noise, and break down different optimization techniques for neural nets.

## 2 Approximation (continued)

Recall for functions $f(x)$ on $\mathbb{R}^d$ we defined a Fourier constant $C_f = \int_{\mathbb{R}^d} |\omega|_1 |\hat{f}(\omega)| d\omega$ to measure the accuracy of an approximation $f_n(x) \in \mathbb{R}^d$ to the target function $f(x) \in \mathbb{R}$ in terms of the $L_2(\mu, \mathbf{B})$ norm, where $\mu$ is a probability measure and $\mathbf{B}$ is a bounded set.
$\|f(x) - f_n(x)\| = \int_{\mathbf{B}} |f(x) - f_n(x)|^2 \mu(dx)$, for any arbitrary $\mu$ and there is no explicit dependence on the dimension except for the constant, $C_f$. Given any arbitrary sigmoidal function, $\phi$, any arbitrary target function $f$ with $C_f < \infty$, and any arbitrary $\mu$, then for any neural network of size $m \geq 1$, we can define a neural network of the form $\|f(x) - f_n(x)\| \leq \frac{C_f}{\sqrt{m}}$ [Barron '93].

## 3 Ways of Regularization

1. **Add a penalty of a parameter**
   Regularization is a way to reduce overfitting and can be achieved by adding a complex penalty to the loss function. Without regularization we want to solve $\text{argmin}_\theta \; L(\theta)$, and with regularization we want to solve $\text{argmin}_\theta \; L(\theta) + \lambda\Omega(\theta)$. The $\lambda\Omega(\theta)$ term is designed to control the complexity of our $\theta$ and the $\Omega(\theta)$ function is some function along the form of $\Omega(\theta) = \|\theta\|_2^2, \Omega(\theta) = \|\theta\|_1$. $L_1$ norm regularization creates sparsity of the coefficients, which in turn reduces the complexity of the model to a handful of active parameters, while the $L_2$ norm leads to smaller weights. One remark about $\lambda\Omega(\theta)$ is that it is representative of a Bayesian prior. In order to really understand this let's take a closer look at a linear regression example.




The $nxd$ block represents $x$, the $dx1$ block represents $\beta$, the $nx1$ block represents $y$, and $x\beta \approx y$. The data is of the form $\{x, y\}$, $x = \left\{x_i|_1^n, x_i \in \mathbb{R}^d\right\}$, $y = \{y_i|_1^n, y_i \in \mathbb{R}\}$.
Framing this within the context of our loss and omega function, we have

$$L(\theta) = \|x\theta - y\|_2^2 \text{ and } \Omega(\theta) = \|\theta\|_1 = \Sigma_{j=1}^d |\theta_j|$$

If we look at one sample from $y|x, \theta \sim N(x^T\theta, \sigma^2)$, where $y = x^T\theta + \epsilon$, and $\epsilon \sim N(0, \sigma^2)$ then

$$p_\theta(y|x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(x^T\theta - y)^2}{2\sigma^2}}$$

Taking the log of both sides we arrive at

$$\log p_\theta(y|x) = constant - \frac{1}{2\sigma^2}|x^T\theta - y|^2$$

Taking n samples (iid):

$$\prod_{i=1}^n \log p_\theta(y_i|x_i) = constant - \frac{1}{2\sigma^2}\sum_{i=1}^n |x_i^T\theta - y_i|^2$$

Where $\frac{1}{2\sigma^2}\sum_{i=1}^n |x_i^T\theta - y_i|^2$ is equivalent to $\|\mathbf{X}\theta - \vec{y}\|_2^2$. Extending this one step further,

$$p(\{x, y\}, \theta) = p(\theta)p(\{x, y\}|\theta)$$

Taking the log of both sides again,

$$\log p(\{x, y\}, \theta) = \log p(\theta) + \log p(\{x, y\}|\theta)$$

$$\text{Prior of } \theta\text{: } \theta \sim N(0, \sigma^2 I)$$

This leads to

$$p(\theta) \propto e^{\frac{-\|\theta\|^2}{\sigma^2}}$$

Which shows,

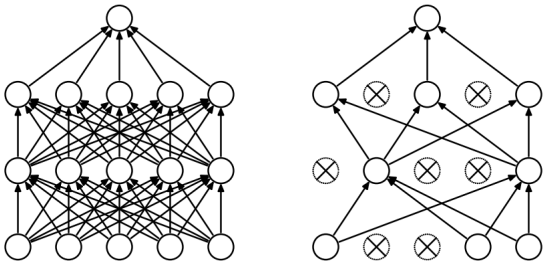
$$\log p \to \Omega(\theta) = \|\theta\|_2^2$$

Another point is in this context $\lambda\|\theta\|_2^2$ creates a weight decaying effect, which has been shown to not have great performance for some deep learning problems. The $L_2$ penalty term creates a weight decay effect, but this has been shown to have little regularizing effect on simplifying the given function, since the function is not changed by the scale of the weights.

2. Adding Noise

2A. **Dropout**

In each training epoch, for each neuron $h_k$, randomly ($\sim 0.5 = p$) decide that $h_k$ does not participate in the prediction model in computing the loss, and update $\theta$ accordingly. Applying dropout to a neural network is synonymous to sampling a smaller thinned network from it. There are $2^n$ possible neural networks for a network with $n$ units. Training a neural network with dropout is like training $2^n$ smaller networks with weight sharing between the networks. For testing, a single neural network can be used and the weights of the network can be scaled by the dropout probability, $p$. This is like performing model averaging on all possible $2^n$ networks.



(a) Standard Neural Net (b) After applying dropout.

Figure 1: Dropout Neural Net Model. **Left**: A standard neural net with 2 hidden layers. **Right**: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

Source: [Srivastava, Hinton, Krizhevsky, Sutskever, Salakhutdinov '14]

Without dropout ($p = 1$) we have

$$y = \sum_{k=1}^{m} c_k \sigma(a_k^T x + b_k) + Const, \text{ where } \sigma(a_k^T x + b_k) = h_k$$

And with dropout ($p < 1$) we have

$$y = \sum_{k=1}^{m} c_k \cdot h_k \cdot \xi_k + Const$$

Where the $\xi_k$ are iid Bernoulli, such that $\xi_k = 1$ with probability $p$ and $\xi_k = 0$ with probability $1 - p$.

2B. **Denoising Autoencoder**

The motivation behind the denoising autoencoder is that encoders with more hidden layers than inputs can learn the identity function, which essentially renders it useless. Denoising autoencoders are the stochastic equivalent of autoencoders. They attempt to reduce the ability of the encoder to learn the identity function by introducing noise, with more noise being added for smaller datasets and less for larger.

2C. **Ensemble Bayesian Model Averaging**

The motivation behind Bayesian Model Averaging is to model uncertainty in parameters to combat overfitting. To achieve this we have

$$p(\theta|D) = p(D\theta)p(\theta)$$

where the equation above represents the posterior being the product of the conditional and the prior. In order to average over all possible models we take

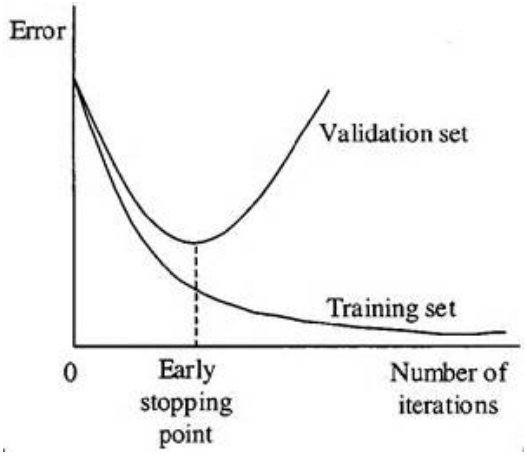
$$p(\theta|D) = \int_{\theta} \theta p(\theta|D)p(\theta)$$

2D. **SGD + Bayesian sampling (SG-MCMC)**
Adding appropriate amounts of noise in order to sample from the posterior. For mroe info on this refer to "Bayesian Learning via Stochastic Gradient Langevin Dynamics".

# 4 Optimization Techniques

1. **SGD + variance reduction**
The motivation behind SGD + variance reduction is due to the inherent variance, stochastic gradient descent has slow convergence asymptotically. If we can reduce this variance while maintaining a high learning rate, then the expected converge will be faster than plain SGD. This is true both in theory and in practice. One common method for reducing variance in gradient descent is control variates, for more info on control variates see "Variance Reduction for Stochastic Gradient Optimization".

2. **Early Stopping**




The idea behind early stopping is to monitor the performance of a model on a held out validation set after each epoch. After each epoch or $n$ epochs we test our model on the validation set and store the best model, after we start seeing no to little performance gained, we stop training and return the best results. Early stopping works well for back propagation neural networks with excess capacity. We can train large, deep nets in similar times to smaller nets using early stopping.