CS 182/282A: Designing, Visualizing and Understanding Deep Neural Networks **Spring 2019**

Section 1: Introduction to Learning

Notes by: David Chan

1.1 Course Logistics

- The goal of the sections/discussions is to provide useful supplemental information to the main lecture
- There will be a mix of practical skills discussions and theoretical discussion
- We're considering moving the sections around in time/location: Please fill out the form `https://bit.ly/2B5srpu` to update your preferences.

1.2 Neural Network Basics

An artificial neural network (ANN) is a computational model, which was originally inspired by the way that neurons in the human brain processes information. The basic unit of computation in a simple ANN is the **neuron**, also called **nodes** or **units**. These units receive **inputs** from other nodes, or from inputs to the network, and they compute a function of the inputs, to produce **outputs**. The simplest neuron is the "Inner Product" neuron, given in Figure 1.1. This neuron computes the output function $f : \mathbb{R}^n \to \mathbb{R}^k$:

$$f(x) = a(Wx + b) \tag{1.1}$$

where $W \in \mathbb{R}^{k \times n}$ is a **weight** matrix, $b \in \mathbb{R}^k$ is what we call a **bias** vector, and $a : \mathbb{R}^k \to \mathbb{R}^k$ is an **activation function**.

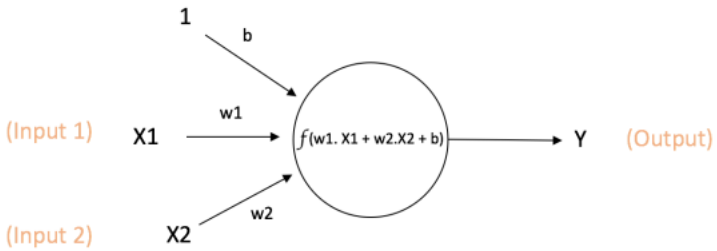

Figure 1.1: A pictorial example of the fully connected neuron. The neuron takes scalar inputs x_1 and x_2, as well as the bias b and computes an output Y. Figure from [3].

Originally a was the *sigmoid* function,

$$a(x) = \frac{1}{1+e^{-x}} \tag{1.2}$$

however there are a number of activation functions that we will explore though the remainder of the class. Figure 1.2 gives a few example functions which we will explore.

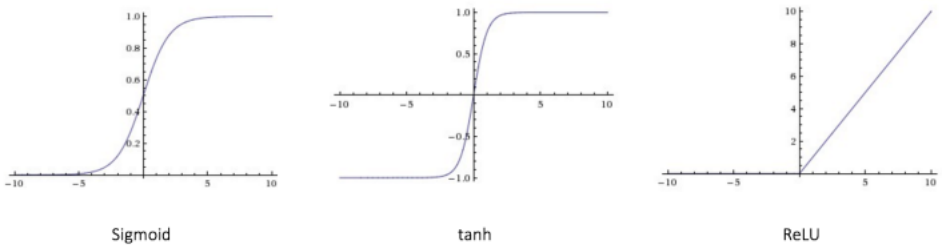

Figure 1.2: Some example activation functions that we will be exploring in this class. Figure from [3].

Notice that in this neuron formulation that the output of the neuron function is entirely controlled by the weight matrix W and the bias vector b. The goal of **training** a neural network is to find sufficient W and b vectors so that the neural network approximates the function which we are trying to learn.

1.2.1 More Complex Networks of Neurons

While the neuron seems simple, we can expand the ways that we connect the inputs and outputs of the neurons together. The classic multi-layer perceptron is given in Figure 1.3. In this model we "stack" neurons together, connecting the output of neurons to the inputs of others. This can help us to approximate more complex functions of the input than a single neuron would allow for. Note that since the computation of outputs are row/column matrix multiplications as we mentioned in the previous section, the computation in these copmlex networks reduce to Matrix-matrix multiplications that GPUs are good at.

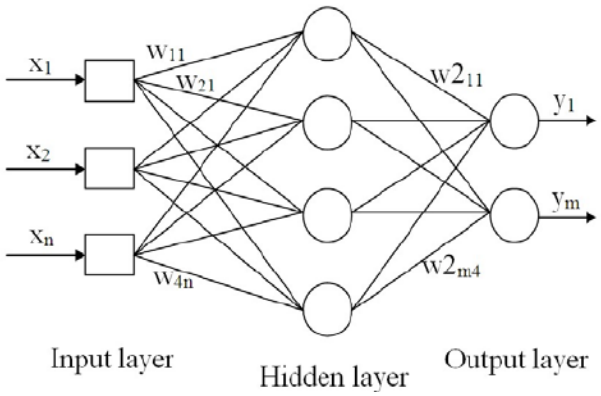

Figure 1.3: A diagram of a multi-layer perception. Figure from [4].

We can also think about adding neurons that perform more complex functions, such as convolutions and pooling (which will lead to Convolutional Neural Networks), or even thinking about augmenting the neurons with temporal features (which will lead to Recurrent Neural Networks).

1.3 Function Approximation & Risk Functions

There is a lot of hype surrounding deep neural networks, but at their core they are just ways of learning functions. In the case of classification, we are trying to learn $p(y|x)$, the probability of some class y given the input features x. In the case of regression, it's a similar continuous response variable. In the case of generative models, we are trying to learn to approximate a whole distribution. In all of the cases, we are trying to find an estimator $\hat{y}$ of a true distribution y.

The way that we find this estimator $\hat{y}$ is by adjusting the weights and biases in the network, often called the **parameters** of the network in order to minimize the distance between the estimated distribution $\hat{y}$ and the true distribution y. But how do we specify this distance?

One of the most common distance metrics, or **Risk functions** for evaluating the quality of an estimator is the mean-squared error:

$$MSE(\hat{y}) = \mathbb{E}_y[(y - \hat{y})^2] \tag{1.3}$$

This is the expected squared deviation of the estimator from the true distribution (over the true distribution).

Because we don't have access to the true distribution y, we cannot directly optimize this objective, however, we minimize the **empirical risk**, where the true distribution y is replaced by the **empirical distribution** (Samples from the true distribution y). We will see echos of the MSE risk function throughout the material presented in the class.

1.3.1 Bias-Variance Tradeoff

Even though we can't directly optimize the MSE, we can still look at the MSE equation to better understand sources of error in our estimation. In particular, we can derive the **bias-variance** decomposition of the MSE, and relate it to **over-fitting** and **under-fitting** in the network.

A simple of the bias-variance decomposition is given below:

$$\begin{aligned}
MSE(\hat{y}) &= \mathbb{E}_y[(\hat{y} - y)^2] && (1.4)\\
&= \mathbb{E}_y[\hat{y}^2 - 2y\hat{y} + y^2] && (1.5)\\
&= \mathbb{E}_y[\hat{y}^2] - 2y\mathbb{E}_y[\hat{y}] + \mathbb{E}_y[y^2] && (1.6)\\
&= Var(\hat{y}) + \mathbb{E}_y[\hat{y}]^2 - 2y\mathbb{E}_y[\hat{y}] + y^2 && (1.7)\\
&= Var(\hat{y}) + (\mathbb{E}_y[\hat{y}] - y)^2 && (1.8)\\
&= Var(\hat{y}) + Bias(\hat{y})^2 && (1.9)
\end{aligned}$$

We can thus see that the MSE estimator is exactly to the variance of the estimator plus the square of it's bias. But what this mean?

The bias of an estimator is equal to $Bias(\hat{y}) = \mathbb{E}_y[\hat{y} - y]$, that is, how much does the expected value of the estimator differ from the true distribution. An **unbiased** estimator is one where $\mathbb{E}_y[\hat{y}] = y$. The variance of

a estimator is how much the estimator varies, that is $Var(\hat{y}) = \mathbb{E}_y[(\hat{y} - \mathbb{E}[\hat{y}])^2]$, or how much the estimator differs from the expected value of the estimator on average.

The best estimator, thus, has low bias, and low variance. So why don't we always use an unbiased estimator? Sometimes, we might want to introduce a little bit of bias if it significantly decreases the variance.

Bias and variance can be summarized by the following graphic:

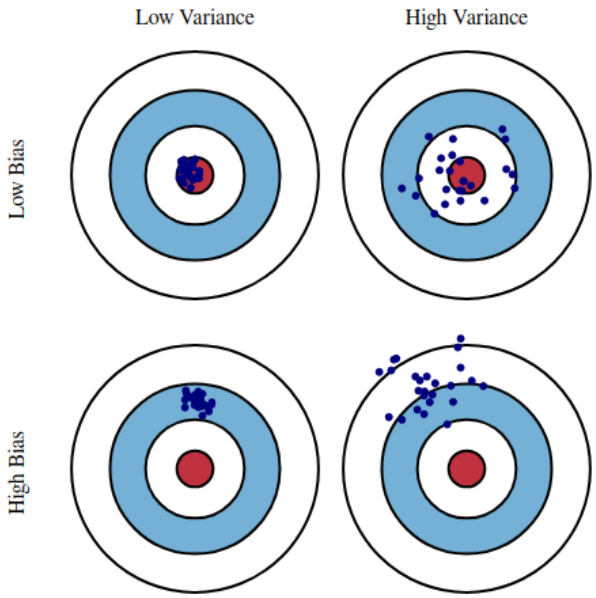

Figure 1.4: A visual explanation of the bias and variance. Figure from [1].

Exercise: Is the Nearest Neighbor estimator unbiased? How about a k-Nearest Neighbor estimator?

1.3.2 Bias and Variance in Neural Networks

There are two terms that often come up when we discuss bias and variance in neural networks. The first term is **Overfitting**. Overfitting occurs when a statistical model or machine learning algorithm captures the noise of the data. Intuitively, overfitting occurs when the model or the algorithm fits the data too well. Specifically, overfitting occurs if the model or algorithm shows low bias but high variance. Overfitting is often a result of an excessively complicated model, and it can be prevented by fitting multiple models and using validation or cross-validation to compare their predictive accuracies on test data [2].

The other term that often comes up is **underfitting**. Underfitting occurs when a statistical model or machine learning algorithm cannot capture the underlying trend of the data. Intuitively, underfitting occurs when the model or the algorithm does not fit the data well enough. Specifically, underfitting occurs if the model or algorithm shows low variance but high bias [2].

1.4 Deriving Logistic Regression

In class, we discussed the basics of generative and discriminative classifiers, as well as showed an introduction to logistic regression. Recall that Logistic regression is a method for predicting a binary target value $y \in \{0, 1\}$ from an m-dimensional binary input vector $x \in \{0, 1\}^m$. In class, we derived the function

$$f(x) = \frac{1}{1 + exp(-w^T x + b)} \tag{1.10}$$

Where does this function come from? Why do we pick the logistic function from all of the possible nonlinear functions? Our goal is to show that this model can be derived from the perspective of a Naive Bayes model with class-conditional Gaussians.

The Naive Bayes model is exemplified by a generative process for the data:

1. We choose a class $Y = 0$ or $Y = 1$ depending on a Bernoulli random process
2. We sample the class features $\{X_1, \ldots, X_m\}$ from sampling m unique gaussian distributions which have different parameters depending on the value of Y.

If we follow this process and we make some conditional independence assumptions, we find that we get a joint probability for any sample (x, y) in our dataset is:

$$P(x, y|\theta) = p(y|\pi) \prod_{j=1}^{m} p(x_j|y, \theta_j) \tag{1.11}$$

Where θ is a set of parameters. More formally, let $Y \in 0, 1$ be a Bernoulli random variable with parameter π. Thus for a dataset element in our distribution:

$$p(y|\pi) = \pi^y (1 - \pi)^{1-y} \tag{1.12}$$

Given the conditional independence assumption expressed by the Naive Bayes model, the probability $p(x|y)$ will factor into a product over conditional probabilities $p(x_j|y)$.

For $Y = 0$, let each X_j have a Gaussian distribution:

$$P(x_j|Y = 0, \theta_j) = \frac{1}{(2\pi\sigma_j^2)^{1/2}} exp\left\{-\frac{1}{2\sigma_j^2}(x_j - \mu_{0j})^2\right\} \tag{1.13}$$

where μ_{0j} is the j'th component of the mean vector for the class $Y = 0$. For the class $Y = 1$, we have similarly:

$$P(x_j|Y = 1, \theta_j) = \frac{1}{(2\pi\sigma_j^2)^{1/2}} exp\left\{-\frac{1}{2\sigma_j^2}(x_j - \mu_{1j})^2\right\} \tag{1.14}$$

Notice that we assume that the variances remain the same between both distributions - while they can vary along the component distribution, they are not allowed to vary between classes. Thus, we have $\theta_j = (\mu_{0j}, \mu_{1j}, \sigma_j^2)$.

Let's now then calculate the posterior probability $P(Y = 1|x, \theta)$ for $x = (x_1, \ldots, x_m)$. The math is a bit ugly, but greatly simplified if we work with matrix notation. Thus let:

$$p(x|y = k, \theta) = \frac{1}{(2\pi)^{m/2}|\Sigma|^{1/2}} \exp\left\{-\frac{1}{2}(x - \mu_k)^T \Sigma^{-1}(x - \mu_k)\right\} \tag{1.15}$$

for both of the classes $k \in \{0, 1\}$. We have $\mu_k = (\mu_{k1}, \mu_{k2}, \ldots, \mu_{km})$ as the vector of means for the k'th Gaussian, and where $\Sigma = \text{diag}(\sigma_1^2, \sigma_2^2, \ldots, \sigma_m^2)$ is the diagonal covariance matrix. We thus have:

$$p(Y = 1|x, \theta) = \frac{p(x|y = 1, \theta)p(Y = 1|\pi)}{p(x|Y = 1, \theta)p(Y = 1|\pi) + p(x|Y = 0, \theta)p(Y = 0|\pi)} \tag{1.16}$$

$$= \frac{\pi \exp\{-\frac{1}{2}(x - \mu_1)^T \Sigma^{-1}(x - \mu_1)\}}{\pi \exp\{-\frac{1}{2}(x - \mu_1)^T \Sigma^{-1}(x - \mu_1)\} + (1 - \pi) \exp\{-\frac{1}{2}(x - \mu_0)^T \Sigma^{-1}(x - \mu_0)\}} \tag{1.17}$$

$$= \frac{1}{1 + \exp\{-\log \frac{\pi}{1-\pi} + \frac{1}{2}(x - \mu_1)^T \Sigma^{-1}(x - \mu_1) - \frac{1}{2}(x - \mu_0)^T \Sigma^{-1}(x - \mu_0)\}} \tag{1.18}$$

$$= \frac{1}{1 + \exp\{-(\mu_1 - \mu_0)^T \Sigma^{-1} x + \frac{1}{2}(\mu_1 - \mu_0)^T \Sigma^{-1}(\mu_1 + \mu_0) - \log \frac{\pi}{1-\pi}\}} \tag{1.19}$$

$$= \frac{1}{1 + \exp\{-\beta^T x + \gamma\}} \tag{1.20}$$

Where the final equation defines the parameters β and γ:

$$\beta \equiv \Sigma^{-1}(\mu_1 - \mu_0) \tag{1.21}$$

and

$$\gamma \equiv \frac{1}{2}(\mu_1 - \mu_0)^T \Sigma^{-1}(\mu_1 + \mu_0) - \log \frac{\pi}{1 - \pi} \tag{1.22}$$

We can recall that the formula presented in class uses the same parameters γ and β, however we make no probabilistic assumptions on the two parameters. Thus, we can see that the logistic function in logistic regression arises from some simple probabilistic assumptions given by the Naive Bayes classifier.

** This derivation is courtesy Dr. Michael I. Jordan (From our very own UC Berkeley, Not the Basketball player)

References

[1] *Bias and Variance*. URL: http://scott.fortmann-roe.com/docs/BiasVariance.html.

[2] *Machine Learning Lesson of the Day - Overfitting and Underfitting*. Mar. 2014. URL: https://chemicalstatistician.wordpress.com/2014/03/19/machine-learning-lesson-of-the-day-overfitting-and-underfitting/.

[3] Ujjwalkarn. *A Quick Introduction to Neural Networks*. Aug. 2016. URL: https://ujjwalkarn.me/2016/08/09/quick-intro-neural-networks/.

[4] Khursiah Zainal Mokhtar and Junita Mohamad-Saleh. "An Oil Fraction Neural Sensor Developed Using Electrical Capacitance Tomography Sensor Data". In: *Sensors (Basel, Switzerland)* 13 (Sept. 2013), pp. 11385–406. DOI: 10.3390/s130911385.