October 23, 2017 Lecture 12, Oct. 16, 2017

Classification - Neural Networks

Lecturer: Justin Domke *Scribe: Annamalai Natarajan*

1 Summary

In the last lecture we discussed back propagation for neural networks in detail. In this lecture we will look at one examples of using a neural network with error back propagation to train a neural network to recognize hand written digits. Following this we will wrap up classification by discussing loss functions in the context of classification.

2 Neural Networks - Recap

Consider the neural network (NN) in Figure 2.1 to recognize hand written digits like in Figure 2.2. The input layer has 784 nodes corresponding to the pixels in the input image (28×28), the hidden layer has 72 nodes and the output layer has 10 nodes corresponding to the number of classes *i.e. ten digits*. Given an input image this NN predicts the output (forward pass) as,

$$f(x) = V\sigma(Wx) \tag{2.1}$$

$$\sigma(s) = \tanh(s) \tag{2.2}$$

Where, $f(x) \in \mathbf{R^{10}}$ is a vector of length 10 for the hand written digit recognition task, $W \in \mathbf{R}^{72\times784}$ and $V \in \mathbf{R}^{10\times72}$. There are two reasons for a NN to output a vector of real numbers versus an integer corresponding to the target class.

1. Gradient descent type algorithms (which a vast majority of NN utilize) require real number outputs for back propagation algorithm to work

2. The vector representation is flexible enough to capture uncertainty in the NN predictions by placing mass (real numbers) on two or more outputs *e.g., a digit that looks a 0 and 6 will result in an output vector with non-zero mass on both 0 and 6*

We plot the weights for the W, V matrices and output $f(x)$ in Figures 2.3 and 2.4 respectively. The light colors indicate higher values (desired).

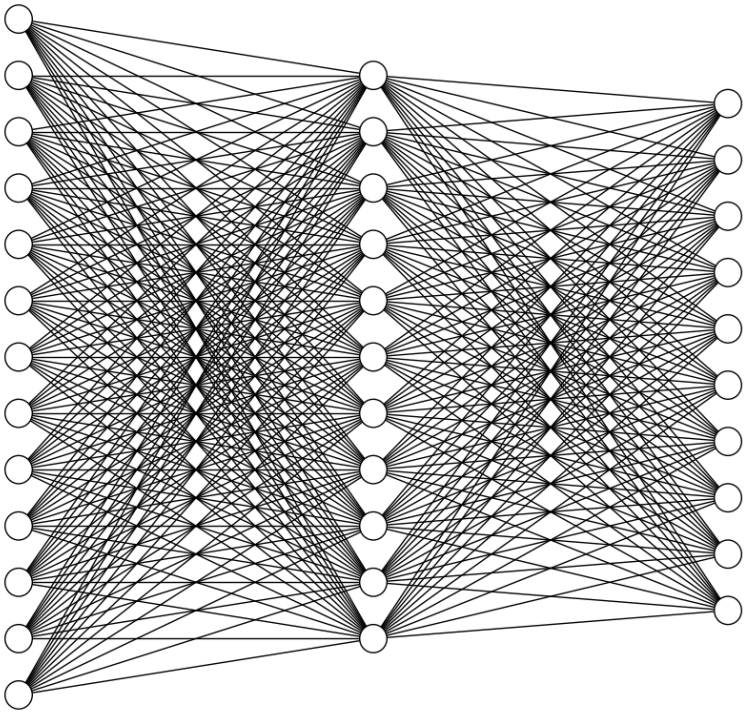

Figure 2.1: Sample neural network to recognize hand written digits. The input layer (left) has 784 nodes, the hidden layer (middle) has 72 nodes and the output layer (right) has 10 nodes

Figure 2.2: Example hand written digits

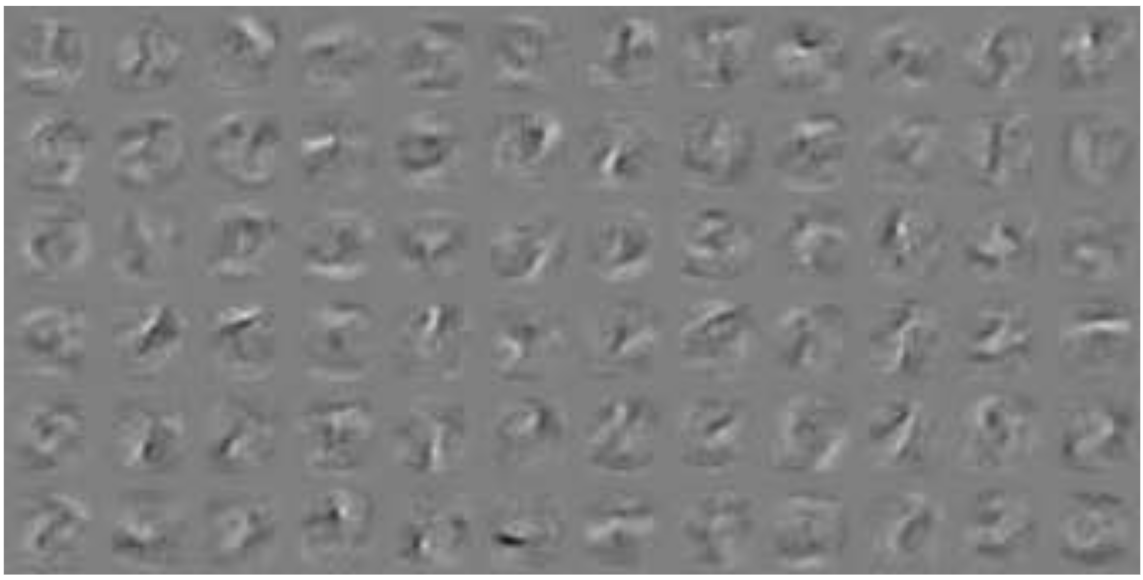

Figure 2.3: Visualizing W matrix. Each subplot is a 28×28 matrix and there are 72 of them corresponding to 72 hidden units. Visualizing the weights are interesting for image recognition problems since the weights directly point to the learning going on under the hood. For hand written digits the W matrix learns the contours in hand written digits.

3 Classification - Loss Functions

In this section we discuss various loss functions in the context of classification problems.

3.1 0-1 Loss

Consider a binary classification problem where $f(x)$ is the prediction made by the classification model for input x and $y \in -1, +1$. The 0-1 loss is given by,

$$L(y, f(x)) = I[yf(x) \leq 0] \tag{3.1}$$

The 0-1 loss is plotted in Figure 3.1.

Cons:

- Hard to optimize
- Difficult to regularize
- Insensitive to scaling
- Does not generalize very well

3.2 Hinge Loss

The hinge loss is given by,

$$L(y, f(x)) = max\{0, 1 - yf(x)\} \tag{3.2}$$

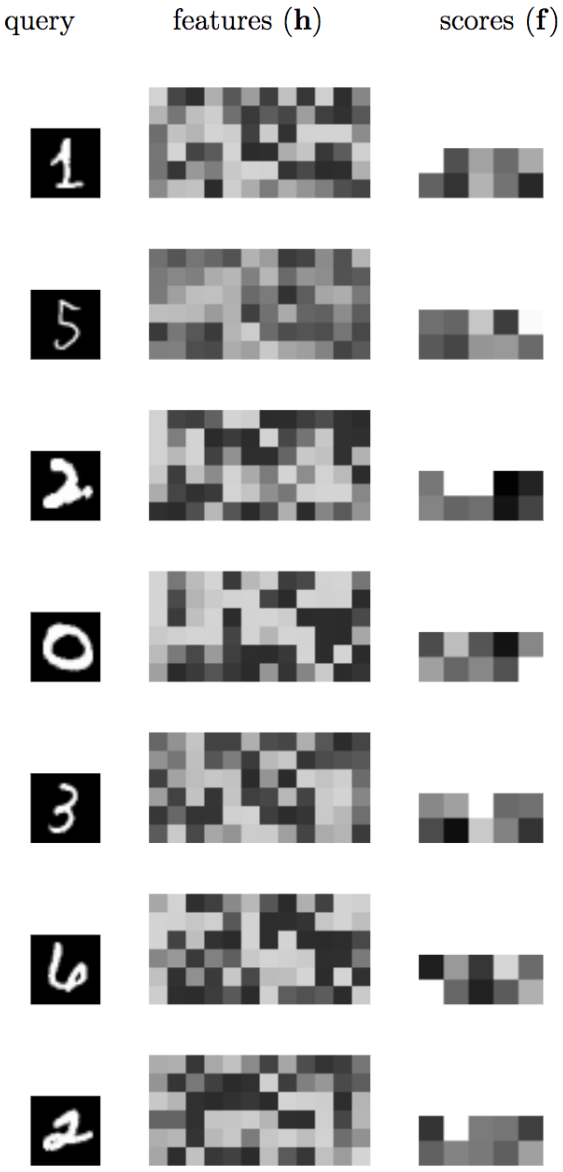

Figure 2.4: Visualizing V matrix and $f(x)$ vector. Each subplot (middle column) is a vector of length 72 and there are ten of them. The outputs $f(x)$ are shown in the extreme right.

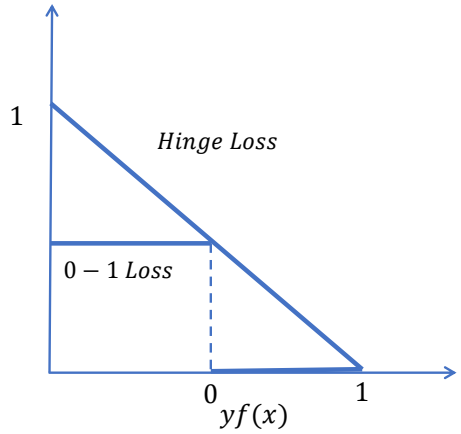

Figure 3.1: 0-1 and Hinge Loss

The hinge loss is plotted in Figure 3.1. Like we observe, The further the predictions are along the left tail the more penalization is incurred. This loss can be optimized efficiently when compared to 0-1 loss. The hinge loss is both convex (leads to single global solution; does not get struck in local optima) and continuous but not smooth since there is a kink at $yf(x) = 1$. Hence we cannot take the gradients of the hinge loss neither can the hinge loss be used for classification models that rely on gradients or gradient based algorithms.

3.3 Logistic Loss

We first discuss logistic loss for binary classification problems. The logistic loss is given by,

$$L(y, f(x)) = \log(1 + \exp(-yf(x))) \tag{3.3}$$

This is a continuous and smooth loss function. Consider the two extreme cases when $yf(x)$ is a large negative and positive value respectively,

$$\begin{aligned} L(y, f(x)) &= \log(1 + \exp(-yf(x))) && (3.4)\\ &= \log(1 + \exp(-(-1000))) && (3.5)\\ &= Inf && (3.6) \end{aligned}$$

$$\begin{aligned} L(y, f(x)) &= \log(1 + \exp(-yf(x))) && (3.7)\\ &= \log(1 + \exp(-(1000))) && (3.8)\\ &= 0.0 && (3.9) \end{aligned}$$

The log loss results in infinity and 0 respectively. Hence log loss penalizes data examples that have incorrect predictions.

Pros:

- Smooth function (no kinks)
- Very popular

Fitting a linear model using logistic loss is referred to as logistic regression, a misnomer, within the machine learning community.

3.4 Multivariate Logistic Loss

Consider a multi-class classification problem where $f(x)$, a vector, is the prediction made by the classification model for input x and $y \in 1, 2, 3, ..., k$. The multivariate logistic loss is given by,

$$L(y, f(x)) = \log(-f_y(x) + \log \sum_{k=1}^{K} \exp(-f_k(x))) \tag{3.10}$$

where, $f_k(x) = \beta_{k0} + \beta_k^T x$, $\beta_{k0} \in \mathbf{R}$, $\beta_k \in \mathbf{R^p}$ and p is the number of features. This can also be represented in matrix form as $f(x) = \beta_0 + \beta^T x$, $\beta_0 \in \mathbf{R^K}$, $\beta \in \mathbf{R^{p \times K}}$.

4 Probabilistic Interpretation of Log Loss

Consider a dataset, D, of N examples of the form $\{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$ which follows some unknown probability distribution. Using N examples we fit a distribution $P(y_i|x_i; \theta)$ which maps the features, x, to the outputs, y using model parameters θ. The question we pose here is how well does the fitted distribution match the data?

To answer this question we compute the conditional likelihood (*i.e. how likely are the y's given x's for a specific setting of* θ). This is computed as,

$$\prod_{i=1}^{N} p(y_i|x_i; \theta) \tag{4.1}$$

Note that we find one setting of θ for the entire dataset. This can also be represented as conditional log likelihood which is more numerically stable as,

$$\sum_{i=1}^{N} \log p(y_i|x_i; \theta) \tag{4.2}$$

Alternately the book represents log likelihood and conditional log likelihood as,

$$\prod_{i=1}^{N} p_{y_i}(x_i; \theta) \tag{4.3}$$

$$\sum_{i=1}^{N} \log p_{y_i}(x_i; \theta) \tag{4.4}$$

Typically the goal is to maximize conditional log likelihood or to minimize negative log likelihood. Since we are discussing log likelihood in the context of loss functions our goal is to minimize the negative log likelihood. This is denoted as,

$$\sum_{i=1}^{N} - \log p(y_i|x_i; \theta) \tag{4.5}$$

For a specific data example (x, y) the negative log likelihood is given by,

$$L(y, f(x)) = -\log(p(Y = y|X = x)) \tag{4.6}$$

$$p(Y = y|X = x) = \frac{\exp(f_y(x))}{\sum_{k=1}^{K} \exp(f_k(x))} \tag{4.7}$$

$$L(y, f(x)) = -\log\Big(\frac{\exp(f_y(x))}{\sum_{k=1}^{K} \exp(f_k(x))}\Big) \tag{4.8}$$

$$= -f_y(x) + \log \sum_{k=1}^{K} \exp(f_k(x)) \tag{4.9}$$

where, $f(x) = \beta_0 + \beta^T x$ and K is the maximum number of classes ($K = 2$ for binary; $K > 2$ for multi class problems).