Training a 2 layer Neural Network using SVD-generated weights

Hyemin Gu

Department of Mathematics Ewha Womans University

Objectives

To train a 2 layered neural network for MNIST handwritten digit classification in a low cost:

- by adjusting the scales of weights and
- by initializing the input layer's weight dependent to the given data using SVD.

Preliminaries

MNIST dataset is composed of 60,000 training samples and 10,000 test samples of gray-scale handwritten digit images from 0 to 9 and their labels. An original image is of 28×28 where each pixel value varies from 0 to 255. When feeding the images as an input of the feed-forward neural network, we vectorize each image, center it at 0, and l_2-normalize by 1. Then we concatenate all the vectors of training/test samples in a row and define it as **Input matrix $\mathbf{X^0}$**. So we have X^0_{train} of $784 \times 60,000$ and X^0_{test} of $784 \times 10,000$. Using one-hot encoding, we define a **Target matrix Y** of $10 \times samplesize$ where $Y_{ij} = I\{\text{label}_j = i-1\}$.

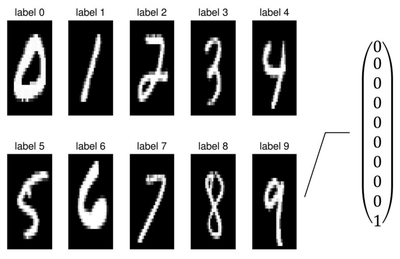

Figure 1: MNIST digit images and a target

The purpose of our 2 layered neural network is to find a set of **weights and biases $\{\mathbf{W^l}, \mathbf{b^l}\}_{\mathbf{l=1,2}}$** where $h_{\{W,b\}_{l=1,2}}(X^0)$ approximates Y with the least error.

From layer $l = 1$ to 2, we calculate

$$z^l = W^l * x^{l-1} + b^l,$$
$$x^l = f(z^l) = \frac{1}{1 + exp(-z^l)}. \tag{1}$$

f is an **(sigmoid) activation function** having the property $f' = f * (1 - f)$. Then we put $h_{\{W,b\}_{l=1,2}}(x^0) = x^2$.

To estimate the error between x^2 and y, we define a **loss function L** called **Cross Entropy(CE)**. Given a pair (x^0, y), $L\left(\{W^l, b^l\}_{l=1,2}\right)$ is defined as

$$-\sum_{i=1}^{s_2} \left(y_i \log x_i^2 + (1 - y_i) \log\left(1 - x_i^2\right)\right). \tag{2}$$

We train $\{W^l, b^l\}_{l=1,2}$ by Gradient descent method. Using **back propagation algorithm**, $\frac{\partial L}{\partial W^l_{ij}}$ and $\frac{\partial L}{\partial W^l_{ij}}$ are formulated as below:

$$\begin{cases} \frac{\partial}{\partial W^l_{ij}} L\left(\{W^l, b^l\}_{l=1,2}\right) = \frac{\partial L}{\partial z^l_i} x^{l-1}_j & \text{, for } l = 2, 1, \\ \frac{\partial}{\partial b^l_i} L\left(\{W^l, b^l\}_{l=1,2}\right) = \frac{\partial L}{\partial z^l_i} & \text{, for } l = 2, 1. \end{cases} \tag{3}$$

$$\frac{\partial L}{\partial z^l_i} = \begin{cases} x^l_i - y_i & \text{, if } l = 2, \\ \sum_{j=1}^{s_{l+1}} W^{l+1}_{ji} \frac{\partial L}{\partial z^{l+1}_j} * \left(x^l_i(1 - x^l_i)\right) & \text{, if } l = 1. \end{cases} \tag{4}$$

Inspiration

We initialize W^l_{ij} to be mean 0, and sharing the constant variance among layers $l = 1, 2$.[1] And let the bias terms $b^1_k = 0$, and $b^2_k = -\frac{1}{2}\sum_j W^2_{kj} \approx 0$. Then we can estimate the **expectation of the norm square of z^1**.

❶If W^{l+1} and x^l are independent,

$$E\left[\left\|z^1\right\|^2\right] = s_1 Var\left[W^1\right] E\left[\left\|x^0\right\|^2\right], \tag{5}$$

❷if W^{l+1} and x^l are the same,

$$\left\|z^1\right\|^2 = s_0 s_1 Var\left[W^1\right] \left\|x^0\right\|^2. \tag{6}$$

When we fix the Frobenius norm of weights while training-(*), 1 and 2 are the minimum and maximum quantity of $\|z^1\|^2$, respectively. So, we set the **dependency ratio between W^1 and x^0** as

$$dep = \frac{\|z^1\|^2_{\text{obs}} - s_1 \|W^1\|^2_{\text{Fro}} / (s_0 s_1)}{s_0 s_1 \|W^1\|^2_{\text{Fro}} / (s_0 s_1)}. \tag{7}$$

Note that we use $Var\left[W^1\right] = \|W^1\|^2_{\text{Fro}} / (s_0 s_1)$, if $E\left[W^1\right] = 0$.

By experiments, we have seen $E\left[W^1\right] = 0$ and this dependent ratio getting increased while training. By (*), this implies W^1 is getting similar to x^0 while training.

Then what if we put the input layer's weight much close to our given training data? In this idea, we put W^1 using the principal components from SVD on the training matrix.

Methods

Let X^0 be the whole training input matrix. By SVD, $X^0 = U\Sigma V^T$ where U, Σ are of 784×784, and V^T is of $784 \times 60,000$. Choose $s_1 \leq r = rank\left(X^0\right)$. Let $\hat{I}$ and $\hat{\Sigma}^{-1}$ be $s_1 \times 784$ matrices only containing the diagonal elements 1 and σ_{ii}^{-1}, respectively. And say $Z^1 = W^1 X^0$. Then we can think of 2 types of W^1 and their $E\left[\left\|z^1\right\|^2\right]$.

❶An orthogonal matrix multiplied by d_1

$$W^{1,\text{orth, whole}} = d_1 * \hat{I} U^T, \tag{8}$$
$$\|Z^1\|^2_{\text{Fro}} = d_1^2 \sum_{i=1}^{s_1} \sigma_{ii}^2. \quad \therefore d_1 = \sqrt{\frac{s_1(1+784*dep)Var[W^1]}{\sum_{i=1}^{s_1} \sigma_{ii}^2/60,000}}.$$

❷A scaled matrix multiplied by d_2

$$W^{1,\text{scale, whole}} = d_2 * \hat{\Sigma}^{-1} U^T, \tag{9}$$
$$\|Z^1\|^2_{\text{Fro}} = d_2^2 * s_1.$$
$$\therefore d_2 = \sqrt{60,000 * (1 + 784 * dep) Var\left[W^1\right]}.$$

Also, we can consider applying SVD on each label subset. We call them as $W^{1,\bullet,\text{each}}$.

Results

Experiments where $s_1 = 100$ and $Var\left[W^1\right] = 0.09$.

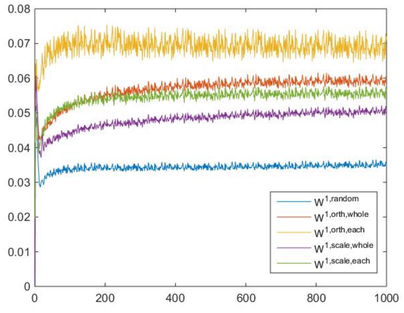

Figure 2: Dependency change while training

Table 1: Performance of each initialization after 5 epochs

Weight Type	Loss	Train Accuracy	Test Accuracy
$W^{1,\text{random}}$	0.71	91.67 %	90.98 %
$W^{1,\text{orth, whole}}$	0.63	93.00 %	91.39 %
$W^{1,\text{orth, each}}$	0.59	94.67 %	93.03 %
$W^{1,\text{scale, whole}}$	0.70	93.00 %	90.90 %
$W^{1,\text{scale, each}}$	0.62	94.00 %	92.73 %

Conclusion

In general, orthogonal matrix of each class showed the best performance. There was no improvement of scaling each row by singular values. But the performance was better as the initial dependency of weight matrix with input data gets bigger. By the way, the computational cost of applying SVD on a $m \times n$ matrix is $O(m^2 n)$ floating-point operations (flops), assuming that $m \leq n$. So, applying SVD on whole data set has the same computational cost with applying SVD on 10 divided subsets.

Further studies

We can consider

- assuming the linearity of f near $0(x = f(z) \approx \frac{1}{4}z + \frac{1}{2})$. And try to estimate the expectation of the norm square of x^1 and so on,
- fixing W^1, and so train a NN without updating W^1 and reduce computational cost.

References

[1] Xavier Glorot, Yoshua Bengio. *Understanding the difficulty of training deep feedforward neural networks*. Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, PMLR 9:249-256, 2010.

[2] Adam Gibson, Josh Patterson. *Deep Learning : A Practitioner's Approach*. O'Reilly Media, 2017.

[3] *Unsupervised Feature Learning and Deep Learning Tutorial* http://uflldl.stanford.edu/tutorial/

Contact Information

- E-mail : nicolegu6616@gmail.com

이화여자대학교 EWHA WOMANS UNIVERSITY