Predicting Hand-written Digits using Backpropagation

Yue WU

yw9998

CS 391L

2019
April

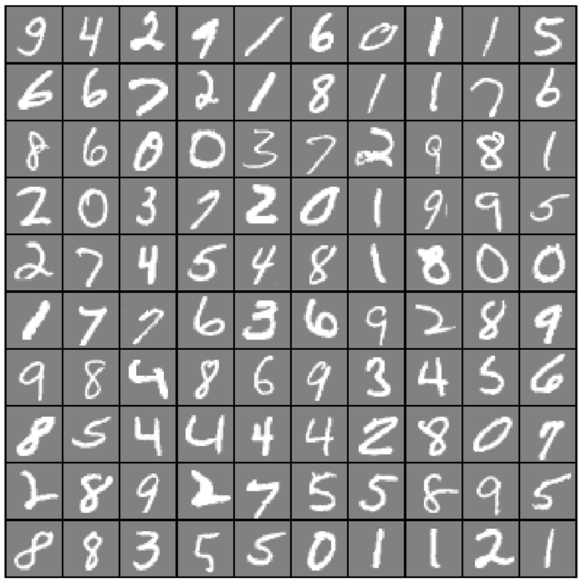
1 Introduction

This homework aims at implementing the backpropagation algorithm to detect hand-written digits automatically. We will use a certain amount (ntrain) of training images included in a training set of 60000 images to derive an algorithm that can detect the digits shown in a test set of 10000 images. Each image contains 784 pixels.

2 Methods

For each image, values for the pixels are stored in a matrix named $X(ntrain - 784)$, and the numbers of the group that each image belongs to are stored in a vector named Y. The numbers from ”0” to ”9” stored in Y are called "labels". Figure 1 shows some of the handwritten digits. In this homework, we mapped ”0” to ”10” in order to make things more compatible with MATLAB indexing where there is no zero index. We then expanded the Y vector to a matrix ry containing only 0s and 1s, where the position of 1s indicates the index:

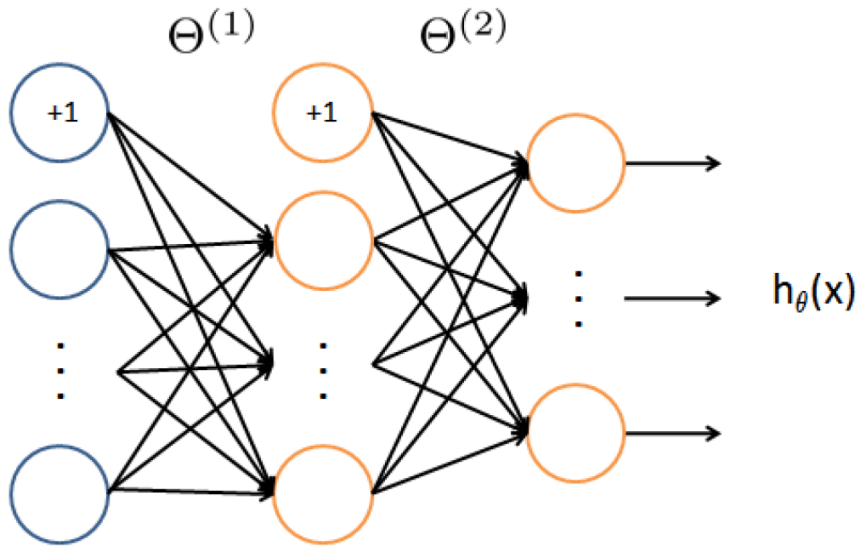
$$ry = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ ... & ... & ... & ... & ... & ... & ... & ... & ... & ... \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{1}$$



(a)

Figure 1: Handwritten digits.

When applying backpropagation (Figure 2), we first set the parameters (Θs) in the model to some random small values. Then we do feedforward and calculate the cost and gradient for the neural network. We can check if the learning process is converging by checking if the cost is generally decreasing after each iteration, and we can update values for the parameters with the cost and gradient for each parameter that we just calculated. We will explain the procedures in detail in the following paragraph, and we will use a model with one input layer, two hidden layers, and one output layer as an example. In this homework, we first initialize the parameters $\Theta^{(1)}$, $\Theta^{(2)}$, and $\Theta^{(3)}$ to be some random small values. Then, we do



(a)

Figure 2: Neural network model.

feedforward to calculate the predicted Y values using the following equations:

$$a^{(1)} = X \tag{2}$$

$$z^{(2)} = a^{(1)}\Theta^{(1)\prime} \tag{3}$$

$$a^{(2)} = actfunction(z^{(2)}) \tag{4}$$

$$z^{(3)} = a^{(2)}\Theta^{(2)\prime} \tag{5}$$

$$a^{(3)} = actfunction(z^{(3)}) \tag{6}$$

$$z^{(4)} = a^{(3)}\Theta^{(3)\prime} \tag{7}$$

$$a^{(4)} = sigmoid(z^{(3)}) \tag{8}$$

where,

$$sigmoid(z^{(3)}) = \frac{1}{1+e^{-z^{(3)}}} \tag{9}$$

the $a^{(4)}$ showing here is the prediction we calculated using the current parameters and training samples. The "actfunction" shown here is the activation function of our choice. We will compare the performance of four activation functions: sigmoid, tanh, ReLU and SeLU. Next, we compare $a^{(4)}$ with ry and compute the costs. We applied two different loss functions to compute the cost: the cross entropy loss (log loss) function (Equation 10) and the hinge loss function (Equation 11).

$$\mathcal{L} = -ry * log(a^{(4)}) - (1 - ry) * log(1 - a^{(4)}) \tag{10}$$

$$\mathcal{L} = max(0, 1 - ry * a^{(4)}) \tag{11}$$

Then, the regularized cost can be calculated by:

$$\begin{aligned} J(\theta) = {} & \frac{1}{ntrain} \sum_{i=1}^{ntrain} \sum_{k=1}^{K} \left[-y_k^{(i)} log((h_\theta(x^{(i)}))_k) - (1 - y_k^{(i)}) log(1 - (h_\theta(x^{(i)}))_k) \right] \\ & + \frac{\lambda}{2 * ntrain} \left[\sum_{j=1}^{nfirst} \sum_{k=1}^{ninput} (\Theta_{j,k}^{(1)})^2 + \sum_{j=1}^{nsecond} \sum_{k=1}^{nfirst} (\Theta_{j,k}^{(2)})^2 + \sum_{j=1}^{noutput} \sum_{k=1}^{nsecond} (\Theta_{j,k}^{(3)})^2 \right] \end{aligned} \tag{12}$$

where *ntrain* is the number of training samples, *ninput* is the number of input parameters (which equals to 784 in this case), *nfirst* is the number of parameters in the first layer, *nsecond* is the number of parameters in the second layer, and *noutput* is the number of output parameters (which equals to 10 in this case).

Now we have the cost, which is a value to determine how far away our prediction is from the actual value. We still need the gradients for each parameter to update the parameters and improve the model using stochastic gradient descent (SGD). The gradient is calculated using the following series of equations:

$$(Cross\ Entropy)\ \delta^{(4)} = a^{(4)} - ry \tag{13}$$

$$(Hinge\ Loss)\ \delta^{(4)} = -ry * a^{(4)} * (1 - a^{(4)}) \tag{14}$$

$$\delta^{(3)} = \delta^{(4)} * \Theta^{(3)} * actfuncgrad(z^{(3)}) \tag{15}$$

$$\delta^{(2)} = \delta^{(3)} * \Theta^{(2)} * actfuncgrad(z^{(2)}) \tag{16}$$

$$\Delta^{(3)} = \delta^{(4)\prime} * a^{(3)} \tag{17}$$

$$\Delta^{(2)} = \delta^{(3)\prime} * a^{(2)} \tag{18}$$

$$\Delta^{(1)} = \delta^{(2)\prime} * a^{(1)} \tag{19}$$

Then, the gradients are:

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = \frac{1}{m} \Delta_{ij}^{(l)} + \frac{\lambda}{m} \Theta_{ij}^{(l)} \tag{20}$$

We update the parameters using:

$$\Theta^{(l)} = \Theta^{(l)} - \alpha \frac{\partial \mathcal{L}}{\Theta^{(l)}} \tag{21}$$

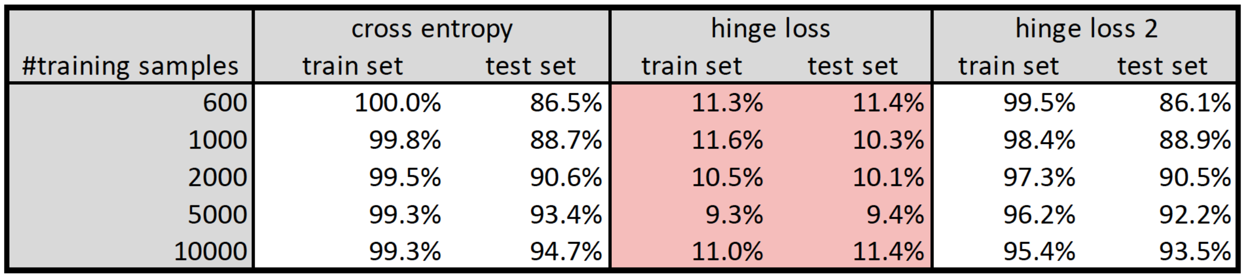
where α is the learning rate.

The updated parameters are then put back into the model and calculate new predictions, costs, and gradients until the cost finally reaches a pre-set threshold.

3 Results

3.1 Comparing loss functions

In this experiment, we used a model with two hidden layers to explore the difference in performance between two categorical loss functions: cross entropy loss function (Equation 10) and hinge loss function (Equation 11). The input layer contains 784 units, the first hidden layer contains 50 units, the second hidden layer contains 30 units, and the output layer contains 10 units which represent the ten labels. We used the sigmoid activation function, and we set the threshold for the cost to be 0.1, the learning rate α to be 0.1. To accelerate the procedure, we applied mini-batch gradient descent and set the size of the batches to 200. We set the regularization parameter λ to be 0 (no regularization). The results are shown in the following form:



	cross entropy		hinge loss		hinge loss 2	
#training samples	train set	test set	train set	test set	train set	test set
600	100.0%	86.5%	11.3%	11.4%	99.5%	86.1%
1000	99.8%	88.7%	11.6%	10.3%	98.4%	88.9%
2000	99.5%	90.6%	10.5%	10.1%	97.3%	90.5%
5000	99.3%	93.4%	9.3%	9.4%	96.2%	92.2%
10000	99.3%	94.7%	11.0%	11.4%	95.4%	93.5%

(a)

Figure 3: Accuracies of prediction when applying two categorical loss functions: cross entropy loss function and hinge loss function.

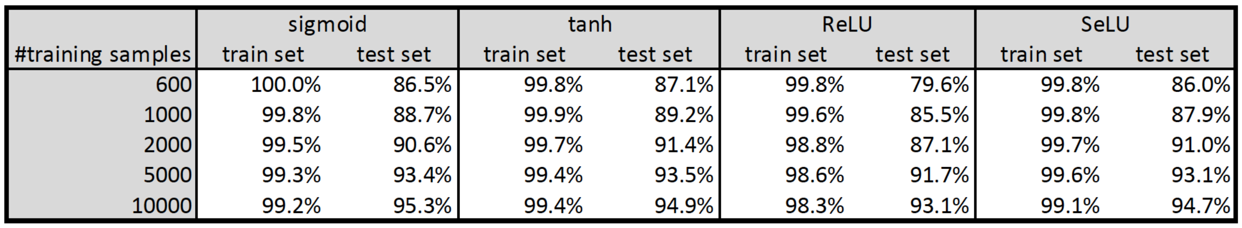
The two columns under "cross entropy" shows prediction accuracies for both training set and testing set for different numbers of training samples using cross entropy loss function.

The two columns under "hinge loss" shows results obtained when applying hinge loss function in both calculations of cost and of gradients. The two columns under "hinge loss 2" shows results when the hinge loss function is used only when applying cost, and we still used the cross entropy loss function when calculating gradients.

Our experiment shows that hinge loss uses much fewer iterations for the cost to reach the threshold, however, the accuracies are very low, or in other words, all predictions failed and the training was failed. This is because hinge loss found out that simply predicting all "1"s can help it reach the threshold easily, which causes problems when calculating the gradients. This is not the case for cross entropy because predicting all "1"s will make the cost become infinity. Therefore, when we used the cross entropy loss function to calculate the gradients while keep using hinge loss function to check the cost, we can still arrive at pretty good results. Besides, training set accuracy is becoming lower as the number of training samples increases, because the number of units is not enough to "overfit" all data, while testing set accuracy increases as the number of training samples increases.

3.2 Comparing activation functions

Next, we used the same model as before, fixed the loss function to be cross entropy loss function, and compare the performance between four different activation functions: sigmoid, tanh, ReLU and SeLU (Equation 22 - Equation 25). Results are shown in the form:



	sigmoid		tanh		ReLU		SeLU	
#training samples	train set	test set	train set	test set	train set	test set	train set	test set
600	100.0%	86.5%	99.8%	87.1%	99.8%	79.6%	99.8%	86.0%
1000	99.8%	88.7%	99.9%	89.2%	99.6%	85.5%	99.8%	87.9%
2000	99.5%	90.6%	99.7%	91.4%	98.8%	87.1%	99.7%	91.0%
5000	99.3%	93.4%	99.4%	93.5%	98.6%	91.7%	99.6%	93.1%
10000	99.2%	95.3%	99.4%	94.9%	98.3%	93.1%	99.1%	94.7%

(a)

Figure 4: Accuracies of prediction when applying four activation functions: sigmoid, tanh, ReLU and SeLU.

$$(sigmoid) \quad g(z) = \frac{1}{1+e^{-z}} \tag{22}$$

$$(tanh) \quad g(z) = \frac{e^z - e^{-z}}{x^z + x^{-z}} \tag{23}$$

$$(ReLU) \quad g(z) = max(z, 0) \tag{24}$$

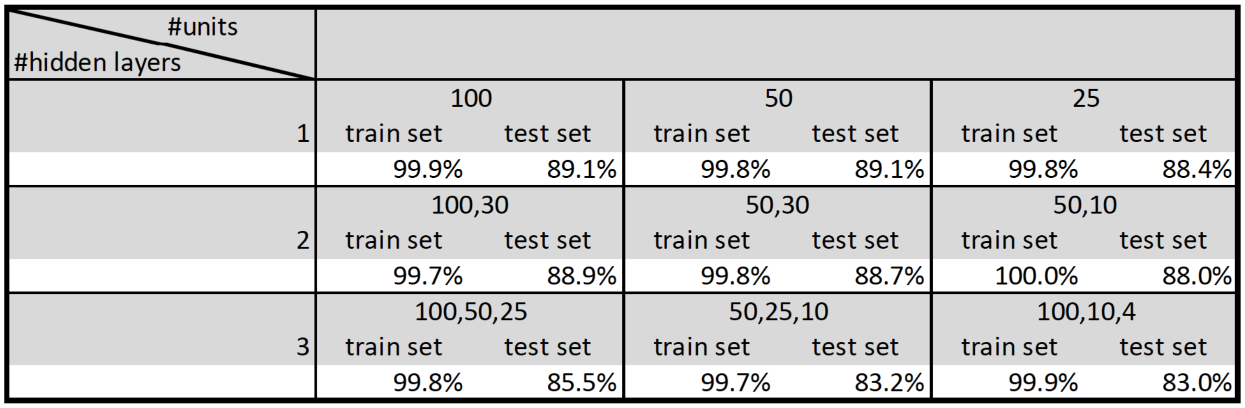
$$(SeLU) \quad g(z) = \lambda \begin{cases} x & if\ x > 0 \\ \alpha e^x - \alpha & if\ x \leq 0 \end{cases} \tag{25}$$

Similar to before, training set accuracy is becoming lower as the number of training samples increases, because the number of units is not enough to "overfit" all data. And testing set accuracy increases as the number of training samples increases. Sigmoid, tanh, and SeLU have similar performance. ReLU is also good but the accuracy is slightly lower than the other three, although it requires fewer iterations and reaches the threshold faster. A potential problem with backpropagation is called "vanishing gradients". When the gradients are close to zero, the weights will not adjust and the whole process stops. ReLU was designed to overcome the problem because as long as the value is larger than zero, the gradient will be 1. However, a potential problem with ReLU is that when the changes in weights are high, the resulting value in the next iteration will be small, leading to the activation function stuck at the left side of the y-axis, where the gradient is zero. This is called "dying ReLU". This might be the reason for the low accuracy for ReLU in our experiment. On the other hand, SeLU, which is very similar to ReLU at the right side of the y-axis, is able to overcome the problem of "dying ReLU" while still has a low possibility of experiencing "vanishing gradients" because of internal normalization.

3.3 Comparing depth and width

Then, applying a similar model with the categorical loss function fixed to be cross entropy loss function and the activation function fixed to be sigmoid, we tried different numbers of depth (hidden layers) and width (units in each hidden layer) for 1000 training samples. The results are shown below:

We have "4" as the smallest number of units here because "4" is the least amount of units required to separate 10 categories. However, this makes it requires much more



#units / #hidden layers						
	100		50		25	
1	train set	test set	train set	test set	train set	test set
	99.9%	89.1%	99.8%	89.1%	99.8%	88.4%
	100,30		50,30		50,10	
2	train set	test set	train set	test set	train set	test set
	99.7%	88.9%	99.8%	88.7%	100.0%	88.0%
	100,50,25		50,25,10		100,10,4	
3	train set	test set	train set	test set	train set	test set
	99.8%	85.5%	99.7%	83.2%	99.9%	83.0%

(a)

Figure 5: Accuracies of prediction when using different numbers of layers and units.

iterations than other cases to reach the threshold and produce an acceptable prediction result, and the accuracy is still lower than the other two. We also found out that more units in each layer give higher testing set accuracy, while more layers do not necessarily lead to higher testing set accuracy, as the possibility of overfitting also increases as the number of layers increases. Therefore, selecting appropriate numbers of layers and units is important to achieving satisfying results.

4 Conclusion

In this homework, we have shown that for categorical loss functions, the cross entropy loss function is more suitable than hinge loss function in our case, as hinge loss tends to simply predict all "1"s so that it can reach the threshold easily, which causes problems when calculating the gradients. For activation functions, Sigmoid, tanh, and SeLU have similar performance. ReLU is also good but the accuracy is slightly lower than the other three. ReLU can overcome the problem of "vanishing gradients", and SeLU can deal with both "vanishing gradients" and "dying ReLU". Besides, Training set accuracy is becoming lower as the number of training samples increases because the number of units is not enough to "overfit" all data. While testing set accuracy increases as the number of training samples increases. We also found out that more units in each layer give higher testing set accuracy, while more layers do not necessarily lead to higher testing set accuracy. Therefore, it is important to select appropriate numbers of layers and units in order to achieve satisfying results.