IFT6135 Assignment1, Programming part

Xiao Fan (20086722)
Zhibin Lu (20091078)

February 19, 2018

Problem 1

Initialization

We use this setting for training : number of hidden units in the first and second layer $h^1 = 600$, $h^2 = 300$, total number of parameters of 654310 which is within the range of $[0.5M, 1.0M]$. Function *relu* is used as activation function, *learning rate* is 0.02, *mini-batch size* is 64.

Figure 1 plots the loss on training set as a function of number of epochs with different initialization methods.

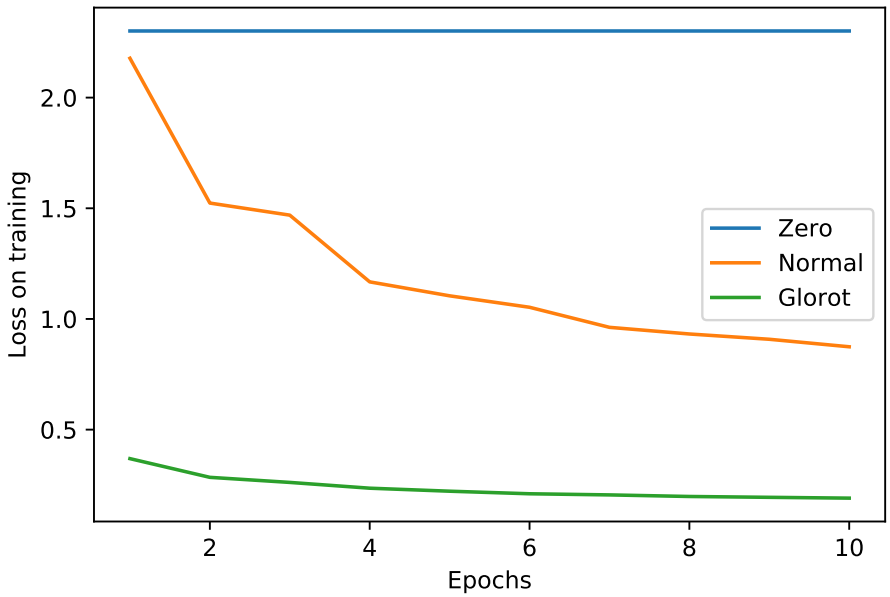


Figure 1: Losses against the training time of different initialization methods.

With zero initialization, weights are not updated during the training, the loss keeps the same. It's a bad initialization method. Glorot initialization is more efficient than normal initialization. The model with Glorot initialization begins with a much smaller loss. Even after 10 epochs, the loss with normal initialization is still larger than that with Glorot initialization at the beginning of training. Glorot initialization gives the model a good point to start.

Learning Curves

Number of hidden units in the first and second layer $h^1 = 600$, $h^2 = 100$, total number of parameters is 532110. Function *relu* is used as activation function, *learning rate* is 0.02, *mini-batch size* is 64. With this setting, the validation accuracy after 100 epochs is 98.02%. If using *learning rate* of 0.06, we can get an validation accuracy larger than 97% (97.29%) just after 10 epochs.

Figure 2 shows the accuracy on both training and validation sets at the end of each epoch. Accuracy on training and validation set after 100 epochs is 99.88% and 98.02% respectively.

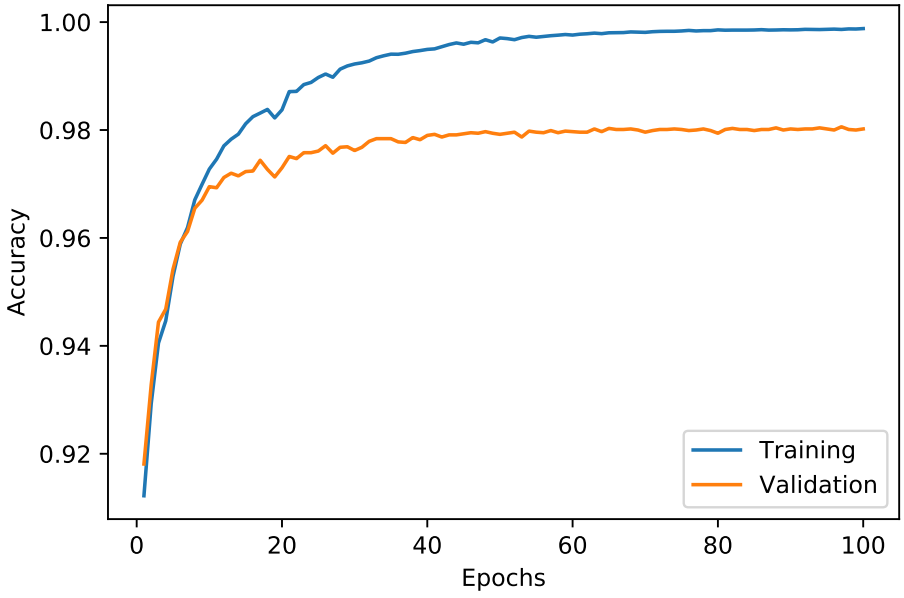


Figure 2: Accuracy on training and validation sets with 100 epochs.

We now double the model capacity with $h^1 = 800$, $h^2 = 400$, total number of parameters is 952410. Accuracy on training and validation set after 100 epochs is 99.9% and 97.95% respectively. Figure 3 shows the accuracy on training and validation sets with the double capacity model.

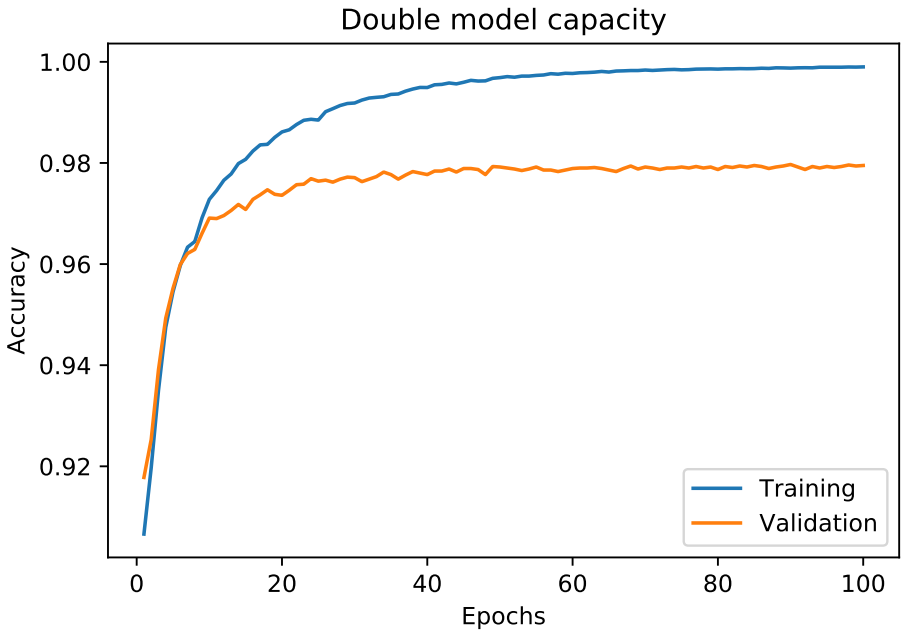


Figure 3: Accuracy on training and validation sets with double capacity model.

Table 1 contains the accuracy on training and validation set after 100 epochs for both models. As seen in the two figures above, for both models, the training accuracy is increasing when training time increases. The difference between training and validation set is large which indicates that both models are overfitting. Double capacity model has a higher accuracy on training set, it has less bias compared to original model. However, the difference of accuracy between training and validation set is larger for double capacity model, which means the variance is larger. Model with more capacity can decrease the bias but will increase the variance at the same time, so the generalization error may increase or decrease. Both bias and variance should be considered to obtain a smallest generalization error.

	accuracy on training set	accuracy on validation set	**difference**
original model	99.88	98.02	1.86
double capacity	99.9	97.95	1.95

Table 1: Accuracy on training and validation set after 100 epochs and their difference for both original model and double capacity model.

Training Set Sizes, Generalization Gap, and Standard Error

We use this setting in this part of experimentation : number of hidden units in the first and second layer $h^1 = 600$, $h^2 = 100$, total number of parameters is 532110, *relu* is used as activation function, *learning rate* is 0.02, *mini-batch size* is 64.

Table 2 contains the generalization gap of each run for subset size of 500, 1000, 2500, 5000, 50000 and their average and standard error.

Subset size	1st run	2nd run	3rd run	4th run	5th run	**Average**	**Std**
500	0.1264	0.1087	0.1198	0.0771	0.0971	0.1058	0.017482
1000	0.0808	0.0925	0.0824	0.082	0.0827	0.0841	0.004259
2500	0.0585	0.0566	0.0593	0.0605	0.0561	0.0582	0.001647
5000	0.0479	0.0542	0.0499	0.055	0.0444	0.0503	0.003950
50000	0.02428	0.02382	0.02356	0.02318	0.02408	0.0238	0.000387

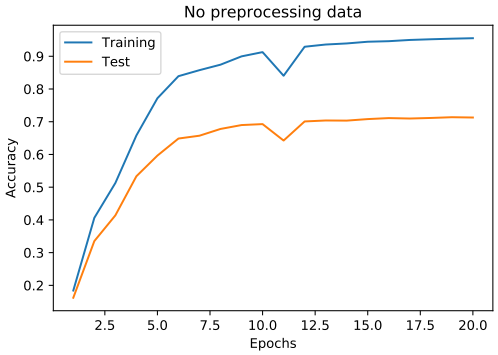
Table 2: Average and standard error of generalization gap for subset size of 500, 1000, 2500, 5000, 50000.

Generalization gap ($r^{(\text{train})} - r^{(\text{test})}$) is an estimate of the variance. When the subset size increases, the average generalization gap decreases which means the trained model has less variance. In practice, more training examples help to reduce both bias and variance which reduces overfitting of model. The decreasing standard error in the table indicates that the trained model is more robust, we have more confidence on the estimate of variance. As a conclusion, with larger training set, we are more confident to have a smaller variance and thus a smaller generalization error.

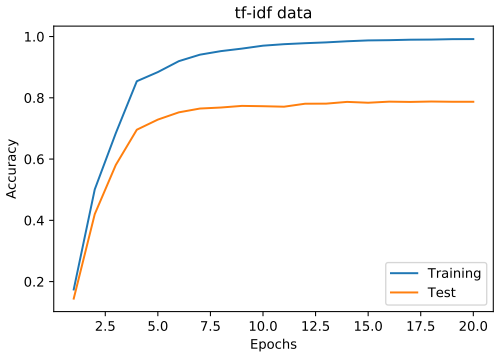
Problem 2

Building the model

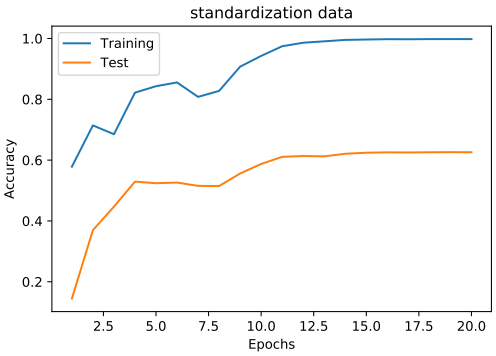
We use 64 as *mini-batch size*. For no preprocessing, *learning rate* is 0.01. For tf-idf preprocessing, *learning rate* is 0.09. For standardization preprocessing, *learning rate* is 0.05. The transformer of tf-idf and of standardization are both trained with training set and are then applied to training and test set. Momentum is set to 0 for vanilla SGD. Figure 4 contains the plot of accuracy on training and test set as a function of training time for different preprocessing methods.



(a) No preprocessing



(b) tf-idf



(c) Standardization

Figure 4: Accuracy on the training and test set for 20 epochs for no-preprocessing, tf-idf and standardization preprocessing.

1. Briefly discuss the results

With no preprocessing, accuracy on test set is about 71% after 20 epochs with *learning rate* of 0.01. The model is overfitting with high accuracy on training set. Using tf-idf preprocessing with *learning rate* of 0.09, we observe a better accuracy on test set, about 79%. Tf-idf helps the model learn better. With standardization, accuracy on test set is worse, only about 63%, even though training set has a high accuracy. We observe huge and increasing loss when *learning rate* is larger than 0.05, but the test accuracy is higher compared to that with a small *learning rate*. This can be explained by the fact the accuracy and loss are two different functions, so it is possible to get high accuracy with increasing loss. Another possible explanation of loss overflow is that some incorrect points contribute very large loss while most of examples are correctly classified. In general, standardization preprocessing method deteriorates the performance of model.

standardization准确率低的原因是，能区分的特征被压缩，缩小至0–1范围。
textbook有类似结论

2. Answer the following points

(a) We could not use the same learning rate for all the models. Input data has different scale after different preprocessing procedures. Without preprocessing, the input

values are term frequency which are 0 or positive integers. With tf-idf preprocessing, word counts are scaled to positive values between 0 and 1. With standardization, input values can be negative and are centered around 0 with variance of 1. Input data has a different distribution after transformation, as a result, different learning rate should be used.

(b) If $\epsilon = 0$, it could incur the error of "division by 0" when doing standardization. Words that exist only in test set will have 0 frequency in training set, which makes $\sigma_{j_{train}} = 0$. Or if words have equal frequency in each document, they will also have $\sigma_{j_{train}} = 0$. As a result, when doing standardization, they will have error of dividing by 0. Besides modifying ϵ, when a word has same frequency for each document, we can assign 1 as standardization result or 0 if frequencies are all 0.

(c) Tf-idf is a more sophisticated term-weighting schema. Compared to basic word count, it considers not only the word frequency in each document, but also the total word frequency in corpus. The word that appears more frequently in the whole corpus has less relevance of specific document and as a result should have less weight. Tf-idf enables to give more weight to words that are more meaningful to the document which helps the model learn better.

Variance in training

Figure 5 is the loss on the training set for the first 5000 updates with mini-batch size of 1 and of 100. We set *learning rate* to 0.15.

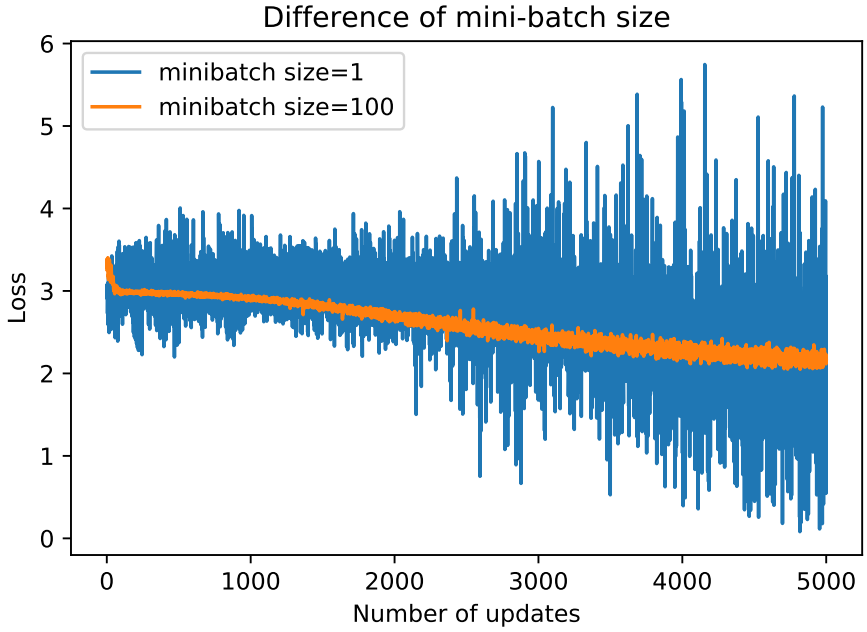


Figure 5: Loss on the training set for the first 5000 updates with mini-batch size of 1 and of 100.

With mini-batch size of 1, the loss on the training set fluctuate heavily, while with mini-batch size of 100, the variance of loss is much smaller. With more updates, the loss tends to decrease but the variance of loss is increasing.

1. The gradient used at each update is the average of gradient evaluated at each example in the mini-batch, that is $g = \frac{1}{n}\sum_{i=1}^{n} g_i$, where n is the mini-batch size. So $\text{Var}[g] = \frac{1}{n}\text{Var}[g_i]$, the variance decreases as a function of mini-batch size. When all samples are used, we calculate the exact gradient. That's why the loss with mini-batch size of 100 has smaller variance. Using single point gradient as the gradient approximation brings more noise which makes the training process unstable.

2. Momentum can help to reduce the variance during training. It corrects the current gradient using past information. The momentum term increases the directions whose gradients keep in the same directions and decreases the directions whose gradients change direction. It helps SGD accelerate in relevant direction and dampen oscillation. Other technique like SAG(stochastic average gradient) can also help. The main idea is to memorize the gradients at each update and use them for subsequent updates. At each step, it calculates a new gradient at one example and uses the average of this gradient and $n-1$ previous gradients for the update. It sacrifices the memory to obtain an accelerated optimizer.