Deep Learning in Data Science, DD2424

Short report on lab assignment 3 bonus
k-layer neural network with multiple outputs

Hamid Dimyati

1. Introduction

The focus of this report is to implement several methods to achieve higher accuracy on the same model architecture (*k*-layer neural networks) and datasets as the main report on lab assignment 3. Some methods that will be presented in this report cover hyperparameter random search (lambda), increasing number of layers, hidden layers size, and applying dropout.

The tool used for this bonus assignment is also `python 3.7.3`, along with several packages including `numpy 1.16.4`, `pandas 0.25.3`, and `matplotlib 3.1.0`.

2. Results and discussion

2.1. Hyperparameter random search

The value of lambda will be further fine-tuned in this experiment. Previously, the optimal value of lambda gained in the main task of the assignment is 0.00440015, which generates 53.72% accuracy on testing data. In this time, the similar coarse-to-fine random search with more iteration will be executed. In the coarse search, I define a broader list of lambda value to be tested:

$$\begin{bmatrix} 10^{-100}, 10^{-95}, 10^{-90}, 10^{-85}, 10^{-80}, 10^{-75}, 10^{-70}, 10^{-65}, 10^{-60}, 10^{-55}, 10^{-50}, 10^{-45}, \\ 10^{-40}, 10^{-35}, 10^{-30}, 10^{-25}, 10^{-20}, 10^{-15}, 10^{-10}, 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1} \end{bmatrix}$$

along with batch size 100, step size 2,250, and for 2 cycles learning. Below are the top three lambdas during the coarse search for the 3-layer neural networks with batch normalization. However, I only use lambda 10^{-2} and 10^{-3} as the limit for the fine search.

Table 1. Accuracy for the top three lambdas during coarse search

Lambda	Validation Accuracy
10^{-2}	54.28%
10^{-55}	53.26%
10^{-3}	53.22%

$$l = l_{min} + (l_{max} - l_{min}) * rand(1,1); \qquad (1)$$
$$lambda = 10^{l}$$

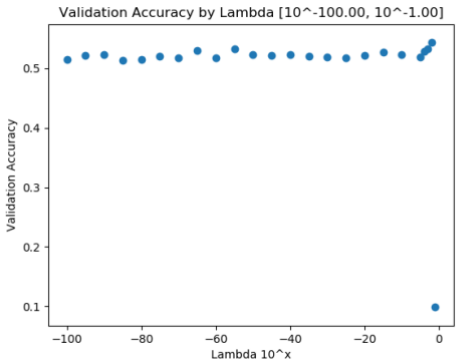

Figure 1. A coarse search of the best lambda with a broader range

In the fine search, I also use 2 cycles learning and iteratively generate 15 random lambda values within the range of $[10^{-3}, 10^{-2}]$ using equation (1), selecting the top five lambda values based on its validation accuracy, update the range based on the best lambda until it converges, as the process could be seen in Table 2 and Figure 2 below. After three iterations, the best lambda gained from this fine search is 0.00632131 or $10^{-2.199}$, which achieves accuracy almost 55.5% on validation datasets.

Table 2. The best lambda obtained from coarse-to-fine random search

Search	**Iteration**	**Range of lambda**	**Top five lambda**
Coarse	0	$[10^{-100}, 10^{-1}]$	[0.01, 1e-55, 0.001]
Fine	1	$[10^{-3}, 10^{-2}]$	[0.00514, 0.00926, 0.00542]
	2	$[10^{-2.28}, 10^{-2.03}]$	[0.00801, 0.00803, 0.00634]
	3	$[10^{-2.26}, 10^{-2.10}]$	[0.00632, 0.00618, 0.00740]

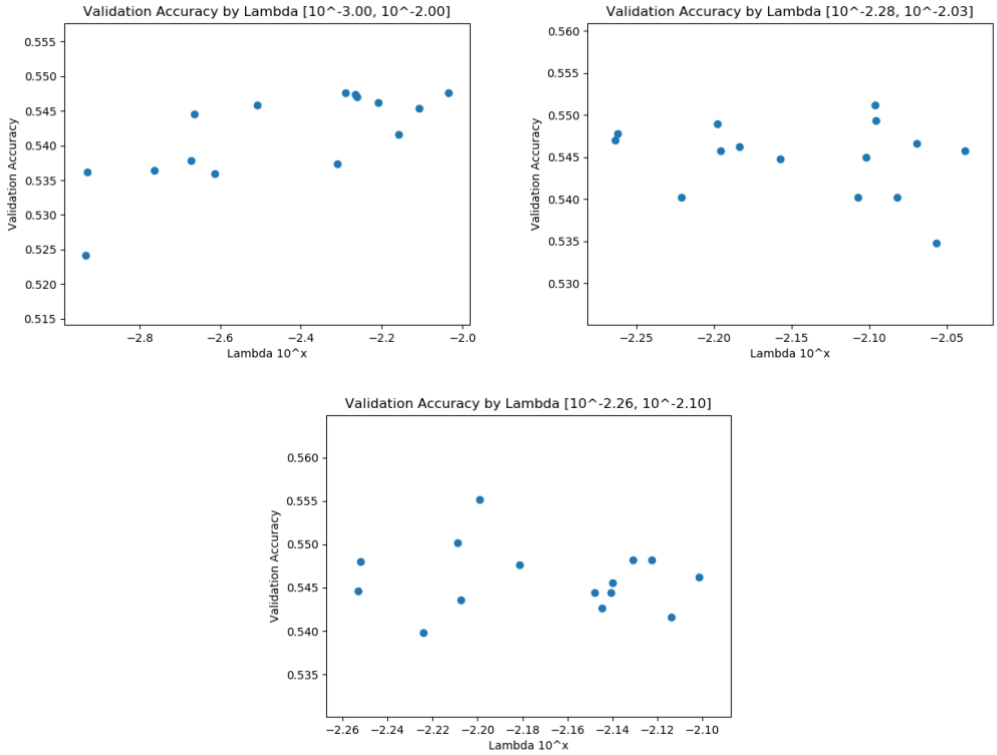

Figure 2. A fine search of the best lambda in the first iteration (top left), the second iteration (top right) and the third iteration (bottom)

Using the best lambda value mentioned above, I got 62.12% accuracy on training data, 54.12% accuracy on validation data, and 53.87% accuracy on testing data. It could improve even only a little bit.

2.2. Effect of the number of hidden layers

Another idea to improve accuracy is increasing the number of hidden layers. I varied the possible number of hidden layers to be tested on the data: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] with the size of 50 nodes in each layer. Figure 3 explains how the effect of increasing the number of hidden layers to the model accuracy. As the number of hidden layers goes up, the accuracy could improve significantly until 4 hidden layers or 5-layer neural networks, and then the accuracy starts to decrease. The highest accuracy is 63.98% accuracy on training data, 55.40% accuracy on validation data, and 54.39% accuracy on testing data. The next effort to increase the accuracy is to fine-tune the hidden size of each layer.

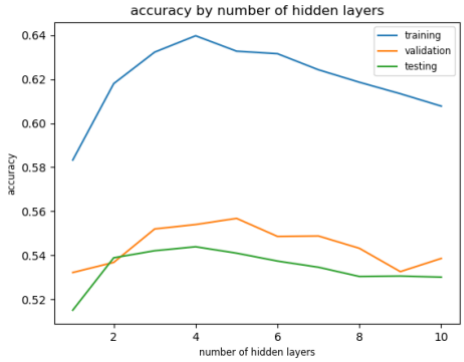

Figure 3. Accuracy of the training, validation and testing datasets as a function of the number of hidden layers

2.3. Hidden size tuning

From 2.2, the best performing neural networks are to use 4 hidden layers. Its performance could be obviously increased by randomly setting the hidden size. I limit the range for generating the number of hidden nodes from 50 to 200 and randomly generate 20 different combinations. The results based on its validation accuracy is presented in Figure 4 and Table 3.

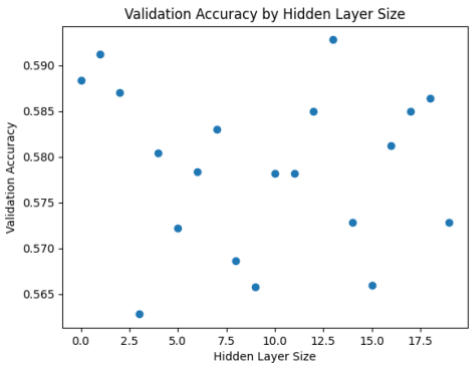

Figure 4. Accuracy of the validation datasets for each combination of hidden sizes

Table 3. Accuracy of different hidden layers size

No	Hidden size	Accuracy
1	[188, 156, 159, 129]	58.84%
2	[171, 121, 88, 120]	59.12%
3	[179, 189, 124, 145]	58.70%
4	[53, 67, 67, 126]	56.28%
5	[154, 78, 72, 58]	58.04%
6	[60, 122, 82, 162]	57.22%
7	[103, 111, 95, 101]	57.84%
8	[166, 75, 111, 148]	58.30%
9	[93, 156, 106, 132]	56.86%
10	[71, 183, 195, 172]	56.58%
11	[117, 126, 181, 169]	57.82%
12	[101, 187, 63, 151]	57.82%
13	[178, 84, 130, 91]	58.50%
14	**[195, 194, 129, 92]**	**59.28%**
15	[95, 68, 172, 130]	57.28%
16	[101, 79, 107, 116]	56.60%
17	[91, 184, 140, 59]	58.12%
18	[189, 96, 151, 99]	58.50%
19	[121, 196, 164, 74]	58.64%
20	[95, 182, 191, 86]	57.28%

The highest validation accuracy is gained from a 5-layer neural network with the hidden size of [195, 194, 129, 92]. Using this setting, I got 75.72% accuracy on training data, 59.28% accuracy on validation data, and 57.22% accuracy on testing data. This clearly suffers from overfitting problem, and then a dropout would be applied to the network.

2.4. Dropout

The idea behind this technique is to reduce a high number of hidden nodes by dropping out some random nodes based on a certain probability threshold in every iteration. Thus, there will be different models learned in every iteration, and the error would be the average of errors from all different models. Therefore, this dropout technique could also be viewed as an approximation to the ensemble method. Using cyclical learning rate $[1e-5, 1e-1]$, lambda 0.00632131, batch size 100, step size 2,250, 20 epochs, 2 cycles learning, and the probability of each node to be included as much 0.85, I got 65.64% accuracy on training data, 57.74% accuracy on validation data, and 56.43% accuracy on testing data. However, considering both the testing accuracy and the regularization power, the best probability to keep the nodes in the dropout process is 0.75, as seen in Figure 5, which tells us that the higher the probability, the higher the gap between training and testing accuracy.

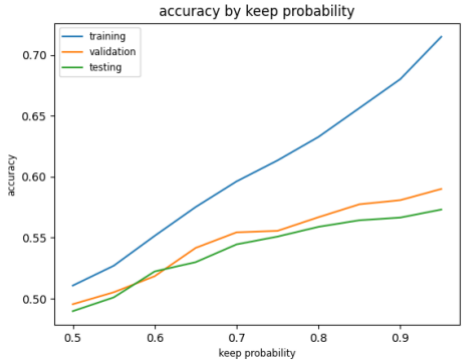

Figure 5. Accuracy of the training, validation and testing datasets as a function of the probability to keep the hidden nodes

By changing the probability into 0.75, I got 61.34% accuracy on training data, 55.56% accuracy on validation data, and 55.09% accuracy on testing data. Figure 6 depicts the cross-entropy loss value, cost value, and the accuracy for each step of the 5-layer network with hidden size [195, 194, 129, 92], applying batch normalization, cyclical learning rate, and dropout.

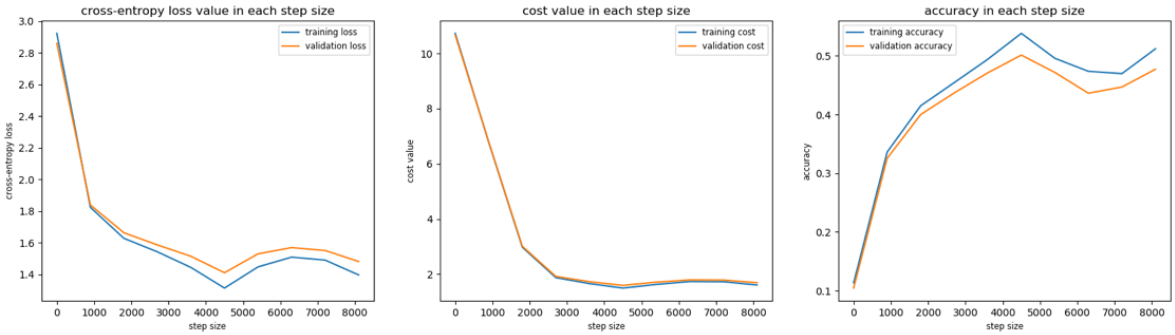

Figure 6. Cross-entropy loss (left), cost value (middle), and accuracy (right) as a function of the step size from the final 5-layer networks

3. Final remarks

From this assignment, it could be learned that random search could be used for hyperparameter tuning as an alternative to the grid search. Besides, adding more hidden layers and fine-tuning the hidden size could also potentially increase the accuracy with the cost of regularization drop due to overfitting. However, the dropout technique could come as a solution to the problem. The best performance I get is 61.34% accuracy on training data, 55.56% accuracy on validation data, and 55.09% accuracy on testing data.