TDT4265: Computer Vision and Deep Learning

Assignment 2

Håkon Hukkelås
hakon.hukkelas@ntnu.no
Department of Computer Science
Norwegian University of Science and Technology (NTNU)

February 1, 2019

- **Delivery deadline: February 14, 2018, by 2359.**
- **This project count towards 6% of your final grade.**
- You can work on your own or in groups of up to 2 people.
- Upload your code as a single ZIP file.
- Upload your report as a PDF file to blackboard. Please, **deliver the PDF file outside of the ZIP file.**
- You can choose what language and frameworks you use. Numpy is allowed to use with python, but any deep learning framework is not allowed in this assignment (Pytorch/tensorflow, matlab DL kit etc). If you're in doubt if a framework is allowed, ask a TA!
- We recommend you to use Python with numpy in this assignment.
- The delivered code is taken into account with the evaluation. Ensure your code is well documented and as readable as possible.

Introduction

In the previous assignment, you implemented a single-layer neural network to classify MNIST digits with softmax regression. In this assignment, we will extend this work to a multi-layer neural network. You will derive update rules for hidden layers by using backpropagation of the cost function. Further, you will experiment with several well-known "tricks of the trade" to improve your network in both accuracy and learning speed. Finally, you will experiment with different network topologies, testing different number of hidden units and number of hidden layers.

1 Softmax regression with backpropagation (1 point)

For multi-class classification on the MNIST dataset, you previously used softmax regression with cross entropy error as the objective function to train a single-layer neural network. Now, we will extend these derivations to work with multi-layer neural networks. We will extend the network by adding a hidden layer between the input and output, that consists of J units with the sigmoid activation function. This network will have three layers: an input layer, a hidden layer, and an output layer.

Notation: We use index k to represent a node in the output layer, index j to represent a node in the hidden layer, and index i to represent an input unit, i.e. x_i. Hence, the weight from node i in the input layer to node j in the hidden layer is w_{ji}. Similarly, for node j in the hidden layer to node k in the output layer is w_{kj}. We will write the activation of hidden unit j as $a_j = f(z_j)$, where $z_j = \sum_{i=0}^{d} w_{ji}x_i$, and f represents the hidden unit activation function (sigmoid in our case). We write the activation of output unit k as $y_k = f(z_k)$, where f represents the output unit activation function (softmax in our case). Note that we use the same symbol f for the hidden and output activation function, even though they are different. However, which f we mean should be clear from the context. This notation enables us to write the slope of the hidden activation function as $f'(z_j)$.

Note: We have slightly adjusted the notation from the previous assignment to keep a consistent notation between the lectures, assignments and the Nielson book. The main differences from the previous assignments are: z and a switched places, and we use the symbol C for the error/cost function instead of E.

Now, we will derive the update rule for the weights in both the hidden layer and the output layer. Since we are using the bias trick (as we did in assignment 1), you can ignore this in your calculations. The rule for updating the weights in the output layer w_{kj} is the same as you found in assignment 1; you do not need to derive this again.

Task 1.1: Backpropagation (0.75 points)

Derive the update rule for w_{ji} (the weights of the hidden layer) using learning rate α, starting with the gradient descent rule. To avoid too many superscripts, assume that there is only one data sample, $N = 1$. For w_{kj} the rule is:

$$w_{kj} := w_{kj} - \alpha \frac{\partial C}{\partial w_{kj}} = w_{kj} - \alpha \delta_k a_j, \tag{1}$$

where $\delta_k = \frac{\partial C}{\partial z_k}$, and ":=" means assignment. Note here we are not assuming any particular form for the error, or any particular activation function.

Again, your job is starting from this:

$$w_{ji} := w_{ji} - \alpha \frac{\partial C}{\partial w_{ji}}, \tag{2}$$

and the definition $\delta_j = \frac{\partial C}{\partial z_j}$, and arrive at this:

$$\alpha \frac{\partial C}{\partial w_{ji}} = \alpha \delta_j x_i, \tag{3}$$

where

$$\delta_j = f'(z_j) \sum_k w_{kj} \delta_k \tag{4}$$

Hint: From the previous assignment, we know that $\delta_k = -(t_k - y_k)$

Task 1.2: Vectorize computation (0.25 points)

The computation is much faster when you update all w_{ji} and w_{kj} at the same time, using matrix multiplications rather than for-loops. Please show the update rule for the weight matrix from the hidden layer to output layer and for the weight matrix from input layer to hidden layer, using matrix/vector notation.

Hint: If you're stuck on this task, take a look in Chapter 2 in Nielsen's book.

2 MNIST Classification (2 points)

Now, you will refer to your derivation from task 1 and implement a 3-layer neural network with an input layer, a hidden layer and a output layer. We will start with a hidden layer with 64 units.

(a) Read in the data from the MNIST database. You can use the loader function given in assignment 1. Remember to one-hot encode the target values. Use 10% of the training data for a validation set, giving you $54,000$ training examples, $6,000$ validation examples and $10,000$ test examples.

(b) The pixel values are in the range $[0, 255]$. Divide them by 127.5, and subtract by 1 so that they are in the range $[-1, 1]$.

(c) Use the softmax activation function for the output layer, and use the sigmoid function for the hidden layer. $a_j = f(z_j) = 1/(1 + exp(-z_j))$. This has the nice feature that $f'(z_j) = \frac{\partial a_j}{\partial z_j} = a_j(1 - a_j)$. Use 64 hidden units to start with.

(d) Check your code for computing the gradient using a small subset of data (a few examples). You can compute the gradient with respect to one weight by using numerical approximation:

$$\frac{\partial C^n}{\partial w_{ji}} = \frac{C^n(w_{ji} + \epsilon) - C^n(w_{ji} - \epsilon)}{2\epsilon}, \tag{5}$$

where ϵ is a small constant (e.g. 10^{-2}), and $C(w_{w_ij} + \epsilon)$ refers to the error on example x^n when weight w_{ji} is set to $w_{ji} + \epsilon$. The difference between the gradients should be within big-O of ϵ^2, so if $\epsilon = 10^{-2}$, your gradients should agree within $O(10^{-4})$. Check your gradient for a few weights between the input layer and the output layer for a few examples.

Hint: If your gradient approximation does not agree with your calculated gradient from backpropagation, there is something wrong with your code!

Remember to be consistent with your normalization of loss and gradient. If you divide your loss by the batch size, remember to do the same for your gradient!

(e) Using the update rule you obtained in Task 1, perform mini-batch gradient descent to learn a classifier that maps each input data to one of the labels. Start with initially random weights and a mini-batch size of 128. Use a validation set to stop training (early stopping). Calculate the error and accuracy after each epoch(holding the weights fixed!), for the training, validation and test set. Stop training when the error on the validation set goes up over 3 epochs. Preferably, save your weights after each epoch and select the final weights from when the validation loss was at its minimum.

In your report, please:

(a) *(1.3 pts)* Implement point (a-c, e). Describe your training procedure, such that a person reading it can replicate what you did. Shortly report your hyperparameters (learning rate, batch size, etc.).

(b) *(0.3 pts)* Shortly report your result for your numerical approximation check of your gradients. Include for both weights in the hidden layer and the output layer. You can select a number of weights from each layer, no need to check for every single one.

When you report this, you can report the maximum absolute difference between the calculated gradient and the approximation of the gradient for each weight matrix. Report the maximum absolute difference for the weight matrix in the input to hidden layer and hidden to output layer.

(c) *(0.4 pts)* Plot your training, validation and testing loss vs number of training iterations of gradient descent. Also, plot the accuracy for the three sets. By accuracy, we mean the percent correct. Again, if your classifier learns the task over a few epochs, calculate the loss and accuracy more often, such as every 1/10th epoch.

3 Adding the "Tricks of the Trade" (1.5 points)

Read the paper *Efficient Backprop*[LeCun et al., 2012] Section 4.1-4.7. The paper can be found with the assignment files. Implement the following ideas from the paper. Do these changes incrementally, i.e., report your results, then add another trick, and report your result again. This way you can observe what effect the different tricks improves the learning. Stick with the normalization previously explained, which is different from what he describes in Section 4.3. If you decide to implement this normalization, please state it in your report.

(a) *(0.3 pts)* If you didn't shuffle your training examples after each epoch, try that now.

(b) *(0.35 pts)* For the hidden layer, use the improved sigmoid in Section 4.4. Note that you will need to derive the slope of the activation function again when you are performing backpropagation.

(c) *(0.35 pts)* Initialize the input weights from a normal distribution, where each weight use a mean of 0 and a standard deviation of $1/\sqrt{\text{fan-in}}$. Fan-in is the number of inputs to the unit/neuron.

(d) *(0.5 pts)* Use momentum with an μ of 0.9. If you implemented nesterov momentum in the previous assignment, you can use this, but remember to state it in the report and report your result!

In your report, please: Shortly comment on the change in performance, which has hopefully improved with each addition, at least in term of learning speed.

4 Experiment with network topology (1.5 points)

Start with your final network from Task 4. Now, we will consider how the network topology changes the performance.

In you report, please answer the following:

(a) *(0.4 points)* Try halving the number of hidden units. What do you observe if the number of hidden units is too small?

(b) *(0.4 points)* Try doubling the number of hidden units. What do you observe if the number is too large?

(c) *(0.7 points)* Increase the number of hidden layers. Create a new architecture that uses two hidden layers of equal size and has approximately the same number of parameters as the network with one hidden layer. By that, we mean it should have the same total number of weights and biases.

In your report, state the number of parameters there is in your network from task 4, and the number of parameters there is in your new network with multiple hidden layers. Also, state the number of hidden units you use for the new network.

Plot the training, validation and testing loss over training. Repeat this plot for the accuracy.

How does the network with multiple hidden layers compare to the previous network?

Hint: When adding a new hidden layer, you can copy the update rule from the previous hidden layer. (This is the beautiful part of backpropagation!)

5 Bonus (0.5 points)

The maximum number of points in this assignment is 6 points and you can not exceed this score. The bonus is a chance for you to get max score on the assignment even though some of the previous tasks are incorrect.

In this bonus task, you are free to have a little fun and try different things of your own choosing to improve accuracy.

Things you can try out(but you are free to try out other things as well!):

1. Implement dropout for the hidden layer.
2. Implement data augmentation during training. Shifting the pixels a few pixels horizontal/vertical should improve your generalization significantly!
3. Use a different activation function for the hidden layer. ReLU, LeakyReLU or ELU might perform well in this task.

In your report, please: State the observed improvement in both accuracy and loss by implementing one (or more) improvements incrementally(much like task 3).

Delivery

Write a report detailing the work you have done. Include all tasks in the report, and mark it clearly with the task you are answering (Task 1.1, Task1.2, Task 2a etc). The report should be in a single PDF file.

For the plots in the report, ensure that they are large and easily readable. You might want to use the "ylim" function in the matplotlib package to "zoom" in on your plots. Label the different graphs such that it's easy for us to see which graphs corresponds to the train, validation and test set.

Include all the necessary code in a single ZIP file. Make sure that your code is well documented and easily readable. Do not include any unnecessary files or folders(do not include the MNIST dataset in your ZIP file).

If you want to deliver your report directly from a jupyter notebook, first ensure that all cells are exectued and has visible output. Export this as a PDF file and deliver both the exported PDF file to blackboard, and the code (as .ipynb or .py file(s)) in a ZIP file.

References

[LeCun et al., 2012] LeCun, Y. A., Bottou, L., Orr, G. B., and Müller, K.-R. (2012). Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer.