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)

January 31, 2020

- **Delivery deadline: Friday, February 14, 2020, by 23:59.**
- **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 single PDF file to blackboard.
- You are required to use python3 to finish the programming assignments.
- 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. Furthermore, 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.

With this assignment, we provide you starter code for the programming tasks. You can download this from:
https://github.com/hukkelas/TDT4265-StarterCode.

To set up your environment, follow the guide in the Github repo:
https://github.com/hukkelas/TDT4265-StarterCode/blob/master/python_setup_instructions.md

Recommended Readings

1. Neural Networks and Deep Learning: Chapter 1
2. 3Blue1Brown: What is Backpropagation Really Doing?
3. 3Blue1Brown: Backpropagation Calculus

Delivery

We ask you to follow these guidelines:

- **Report:** Deliver your answers as a **single PDF file**. Include all tasks in the report, and mark it clearly with the task you are answering (Task 1.1, Task1.2, Task 2.1a etc).
- **Plots in report:** 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 is easy for us to see which graphs correspond to the train, validation and test set.
- **Source code:** Upload your code as a zip file. In the assignment starter code, we have included a script (`create_submission_zip.py`) to create your delivery zip. **Please use this**, as this will structure the zipfile as we expect. (Run this from the same folder as all the python files).

  To use the script, simply run: `python3 create_submission_zip.py`
- **Upload to blackboard:** Upload the ZIP file with your source code and the report to blackboard before the delivery deadline.

Late Delivery & Other Penalties

We have a strict policy on late deliveries. Any delivery submitted after the deadline will be limited in maximum score which you can achieve on the assignment. See "Course Information" on blacbkoard for more information.

Also, if you fail to follow our delivery guidelines we will **subtract 0.2 points** from your final score. This includes if you do not deliver your report as a PDF file, if you don't deliver your code as .ZIP file, or if you don't use the `create_submission_zip.py` to generate your .zip file.

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 two layers: an input layer, a hidden layer, and an output layer [1].

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 $z_j = \sum_{i=0}^{d} w_{ji}x_i$. f represents the hidden unit activation function (sigmoid in our case), and d is the dimensionality of the input. We write the activation of output unit k as $\hat{y}_k = f(z_k)$, 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 $f'(z_j)$.

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.

Changes from last assignment: In this assignment we are not going to normalize our cost function $C^n(w)$ by K [2]. Therefore, our cost function will be:

$$C(w) = -\frac{1}{N}\sum_{n=1}^{N}\sum_{k=1}^{K} y_k^n \ln(\hat{y}_k^n) \tag{1}$$

This change will apply for every task. If you forget to include this change in your code, you need to scale your learning rate by a factor of K as the gradients will be much smaller!

[1] Note that we only count the layers with actual learnable parameters (hidden layer and output layer).

[2] The normalization by K in the last assignment was not necessary, as you noticed that the gradient of the cost function was not dependent on the number of classes.

Task 1a: Backpropagation (0.75 points)

In the previous assignment you derived the gradient descent update rule for the weights w_{kj} of the output layer:

$$w_{kj} := w_{kj} - \alpha \frac{\partial C}{\partial w_{kj}} = w_{kj} - \alpha \delta_k a_j, \tag{2}$$

where $\delta_k = \frac{\partial C}{\partial z_k}$, and ":=" means assignment.

For the weights of the hidden layer, the gradient descent rule with learning rate α is:

$$w_{ji} := w_{ji} - \alpha \frac{\partial C}{\partial w_{ji}}, \tag{3}$$

Your task is to rewrite eq. (3) such that it can be written as a recursive update rule that can be applied without computing $\frac{\partial C}{\partial w_{ji}}$ directly. In order to do this, use the definition $\delta_j = \frac{\partial C}{\partial z_j}$.

The resulting expression should be similar to eq. (2). To avoid too many superscripts, assume that there is only one data sample, $N = 1$. Note that we are not assuming any particular form for the error, or any particular activation function.

Again, your job is, starting from this:

$$w_{ji} := w_{ji} - \alpha \frac{\partial C}{\partial w_{ji}}, \tag{4}$$

rewrite the update rule $\alpha \frac{\partial C}{\partial w_{ji}}$ using the definition of δ_j. Show that you get the expression,

$$w_{ji} := w_{ji} - \alpha \delta_j x_i, \tag{5}$$

and show that $\delta_j = f'(z_j) \sum_k w_{kj} \delta_k$.

Hint 1: A good starting point is to try rewriting $\alpha \frac{\partial C}{\partial w_{ji}}$ using the chain rule.

Hint 2: From the previous assignment, we know that $\delta_k = \frac{\partial C}{\partial z_k} = -(y_k - \hat{y}_k)$.

Task 1b: 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.

We expect you to clearly define the shape of each vector/matrix in your calculation.

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

Task 2: Softmax Regression with Backpropagation (2.0 points)

In this task, we will perform a 10-way classification on the digits in the MNIST dataset with a 2-layer neural network. The network should consist of an input layer, a hidden layer and an output layer.

We expect you to keep/re-implement the following functions from the last assignment:

- Implementation of `one_hot_encode` and `cross_entropy_loss` (include these in task2a.py). In your `cross_entropy_loss` function, **remember to remove** the normalization factor K.
- Early stopping in the training loop. This is not required; however, early stopping might enable you to stop training early and save computation time.

For each task we have set the hyperparameters (learning rate and batch size) that should work fine for these tasks. If you decide to change them, please state it in your report.

Input Normalization: Input normalization is a crucial part of optimizing neural networks efficiently. The convergence is usually faster if the average of each input variable over the training set is close to zero [LeCun et al., 2012]. Section 4.3 in [LeCun et al., 2012] explains in detail the effect of input normalization and presents an extreme case of what can happen if you don't normalize your input data. A simple, yet efficient, way to normalize your images is

$$X_{norm} = \frac{X - \mu}{\sigma}, \tag{6}$$

where μ and σ is the mean pixel value and the standard deviation over the whole training set, respectively.

For your report, please:

(a) [0.3*pt*] Before implementing our gradient descent training loop, we will implement a couple of essential functions. Implement task 2a and task 2b in the file `task2a.py`.

Find a mean and standard deviation value from the whole training set. Then, implement a function that pre-processes our images in the function `pre_process_images`. This should normalize our images with the input normalization trick shown in Equation 6, and it should apply the bias trick.

Note that you should use the same mean and standard deviation value when you normalize your training set, validation set, and test set!

(b) [1*pt*] Implement a function that performs the forward pass through our softmax model. This should compute $\hat{y}$. Implement this in the function `forward`.

Implement a function that performs the backward pass through our two-layer neural network. Implement this in the function `backward`. The backward pass computes the gradient for each parameter in our network (for both the weights in the hidden layer and the output layer).

We have included a couple of simple tests to help you debug your code.

(c) [0.4*pt*] **The rest of the task 2 subtasks should be implemented in `task2.py`.**

Implement softmax regression with mini-batch gradient descent for a single layer neural network. The network should consist of a single hidden layer with 64 hidden units, and an output layer with 10 outputs.

Initialize the weight (before any training) to randomly sampled weights between [-1, 1] [3].

In your report, include a plot of the training and validation loss and accuracy over training. Have the number of gradient steps on the x-axis, and the loss/accuracy on the y-axis. Use the `ylim` function to zoom in on the graph.

[3] You can use the function `np.random.uniform(-1, 1, (785, 64))` to get a weight with shape $[785, 64]$ sampled from a random uniform distribution.

(d) [0.3*pt*] How many parameters are there in the network defined in task 2c? [4]

[4] Number of parameters = number of weights + number of biases

Task 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.

Note that you can create a new file for this task (and task 4), for example `task3.py`, if you want to separate your code from task 2.

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

(b) [0.5*pt*] 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*pt*] 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) [0.35*pt*] Implement momentum to your gradient update step. Use momentum with an μ of 0.9. Note that you will need to reduce your learning rate when applying momentum. For our experiments, a learning rate of 0.02 worked fine.

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

Note that we expect you to comment on convergence speed, generalization/overfitting, and final accuracy/validation loss of the model.

We recommend you to include a plot of the loss detailing the improvements for each addition. For example, as shown in Figure 1.

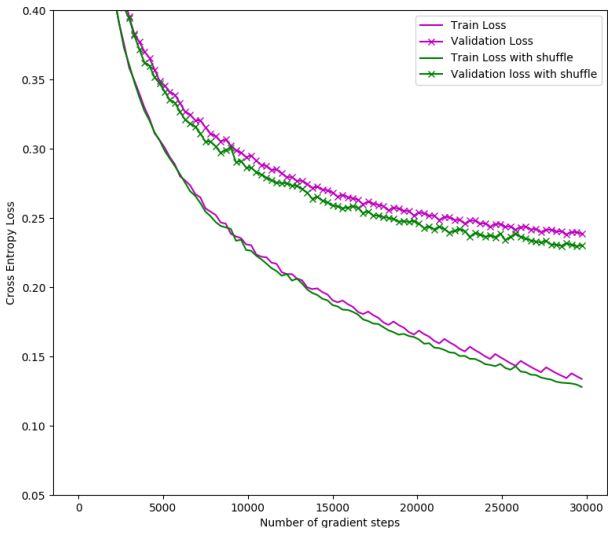

Figure 1: Cross entropy loss for training the model from task2 with and without shuffling the training examples.

Task 4: Experiment with network topology (1.5 points)

Start with your final network from Task 3. Now, we will consider how the network topology changes the performance.

In you report, please answer the following:

(a) [0.2*pt*] Set the number of hidden units to 16. What do you observe if the number of hidden units is too small?

(b) [0.2*pt*] Try doubling the number of hidden units. What do you observe if the number is too large?

(c) [0.8*pt*] Generalize your implementation of your softmax model to handle a variable number of hidden layers.

You can modify your model from task 2a. The variable `neurons_per_layer` specifies the number of layers (length of the list) and the number of units per layer.

To test your implementation you can run the code in `task4c.py`, where we test your model on a network with 2 hidden layers.

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!)

(d) [0.3*pt*] Create a new model with two hidden layers of equal size. The model should have approximately the same number of parameters as the network from task 3 [5].

In your report, state the number of parameters there are in your network from task 3, and the number of parameters there are in your new network with multiple hidden layers. Also, state the number of hidden units you use for the new network.

Train your new model (you can use the same hyperparameters as before). Plot the training, and validation loss over training. Repeat this plot for the accuracy.

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

Bonus (0.2 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!

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

[5] Remember, number of parameters = number of weights + number of biases

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.