Course: Deep Learning *Coordinator:* Jakub Tomczak

Submission Assignment 2

Name: Chi Him Ng, *Student ID:* 2748786

Please provide (concise) answers to the questions below. If you don't know an answer, please leave it blank. If necessary, please provide a (relevant) code snippet. If relevant, please remember to support your claims with data/figures.

Question 1

Let f(X, Y) = X / Y for two matrices X and Y (where the division is element-wise). Derive the backward for X and for Y. Show the derivation.

Answer f(x,y) = X/Y

$$X_{ij}^{\nabla} = \sum_{kl} \frac{\delta l}{\delta C_{kl}} \frac{\delta C_{kl}}{\delta X_{ij}} = \sum_{kl} C_{kl}^{\nabla} \frac{\delta C_{kl}}{\delta X_{ij}} = \sum_{kl} C_{kl}^{\nabla} \frac{\delta [\frac{X}{Y}]_{kl}}{\delta X_{ij}} = \sum_{kl} C_{kl}^{\nabla} \frac{1}{y_{kl}} * 1 = C^{\nabla} \frac{1}{Y} \tag{0.1}$$

The other derivative follows the same steps, except for the last one where the derivative will go over the other variable, this means the outcome will be:

$$X^{\nabla} = -C^{\nabla} \frac{X}{Y^2} \tag{0.2}$$

Question 2

Let f be a scalar-to-scalar function $f : \mathbb{R} \to \mathbb{R}$. Let F(X) be a tensor-to-tensor function that applies f element-wise (For a concrete example think of the sigmoid function from the lectures). Show that whatever f is, the backward of F is the element-wise application of f' applied to the elements of X, multiplied (element-wise) by the gradient of the loss with respect to the outputs.

Answer Since the backward of F is f', we can assume X = C * f'(x)

$$X_{ij}^{\nabla} = \sum_{kl} \frac{\delta l}{\delta C_{kl}} \frac{\delta C_{kl}}{\delta X_{ij}} = \sum_{kl} C_{kl}^{\nabla} \frac{\delta C_{kl}}{\delta X_{ij}} = \sum_{kl} C_{kl}^{\nabla} \frac{\delta [f(x)]_{kl}}{\delta X_{ij}} = C_{ij}^{\nabla} f'(x)_{ij} \tag{0.3}$$

Thus, we can say:

$$X^{\nabla} = C^{\nabla} f'(x) \tag{0.4}$$

Question 3

Let matrix **W** *be the weights of an MLP layer with f input nodes and m output nodes, with no bias and no nonlinearity, and let* **X** *be an n-by-f batch of n inputs with f features each. Which matrix operation computes the layer outputs? Work out the backward for this operation, providing gradients for both* **W** *and* **X**.

Answer C = X * W , furthermore: x(n,f), w(f,m) and c(n,m).

$$\begin{aligned} X_{ij}^{\nabla} &= \sum_{nm} \frac{\delta l}{\delta C_{nw}} \frac{\delta C_{nw}}{\delta X_{nf}} \\ &= \sum_{nm} C_{nm}^{\nabla} \frac{\delta C_{nm}}{\delta X_{nf}} \\ &= \sum_{nm} C_{nm}^{\nabla} \frac{\delta [X * W]_{nm}}{\delta X_{nf}} \\ &= \sum_{nmi} C_{nm}^{\nabla} \frac{\delta X_{in} W_{im}}{\delta X_{nf}} \\ &= \sum_{nm} C_{nm}^{\nabla} \frac{\delta X_{nf} W_{im}}{\delta X_{nf}} \\ &= \sum_{nm} C_{nm}^{\nabla} W_{fm}^{T} \end{aligned} \tag{0.5}$$

$$X^{\nabla} = C^{\nabla} W^{T} \tag{0.6}$$

Question 4

Let $f(\mathbf{x}) = \mathbf{Y}$ be a function that takes a vector $\mathbf{x}$, and returns the matrix $\mathbf{Y}$ consisting of 16 columns that are all equal to $\mathbf{x}$. Work out the backward of f. (This may seem like a contrived example, but it's actually an instance of broadcasting).

Answer f(x) = Y

$$X_{ij}^{\nabla} = \sum_{kl} \frac{\delta l}{\delta C_{kl}} \frac{\delta C_{kl}}{\delta X_{ij}} = \sum_{kl} C_{kl}^{\nabla} \frac{\delta C_{kl}}{\delta X_{ij}} = \sum_{kl} C_{kl}^{\nabla} \frac{\delta X_{kl}}{\delta X_{ij}} = C_{ij}^{\nabla} * 1 \tag{0.7}$$

Thus, we can say

$$X^{\nabla} = C^{\nabla} \tag{0.8}$$

Question 5

Open an ipython session or a jupyter notebook in the same directory as the README.md file, and import the library with import vugrad as vg. Also do import numpy as np.

Create a TensorNode with x = vg.TensorNode(np.random.randn(2, 2))

Answer the following questions (in words, tell us what these class members mean, don't just copy/paste their values).

1) What does c.value contain?
2) What does c.source refer to?
3) What does c.source.inputs[0].value refer to?
4) What does a.grad refer to? What is its current value?

Answer c.value contains the value for the results of a + b.

c.source to the object which is referred to. In this case the node.

c.source.inputs[0].value refers to the first node used for the input, in this case it refers to a.

a.grad refers to the backward operation from the outcome to a.

Question 6

You will find the implementation of TensorNode and OpNode in the file vugrad/core.py. Read the code and answer the following questions
1) An OpNode is defined by its inputs, its outputs and the specific operation it represents (i.e. summation, multiplication). What kind of object defines this operation?
2) In the computation graph of question 5, we ultimately added one numpy array to another (albeit wrapped in a lot of other code). In which line of code is the actual addition performed?
3) When an OpNode is created, its inputs are immediately set, together with a reference to the op that is being computed. The pointer to the output node(s) is left None at first. Why is this? In which line is the OpNode connected to the output nodes?

1) In class opnode as self.op, in the class op this is defined as cls.
2) In the class op, function def do_forward()
outputs_raw = cls.forward(context, *inputs_raw, **kwargs)
3) The output is already calculatend as outputs_raw, thus we can just fill in the calculated value afterwards in opnode.outputs = outputs. The variable has not been given a value, probably to clearly indicate that the value is missing or not valid and that it must be filled in.

Question 7

When we have a complete computation graph, resulting in a TensorNode called loss, containing a single scalar value, we start backpropagation by calling
`loss.backward()`
Ultimately, this leads to the backward() functions of the relevant Ops being called, which do the actual computation. In which line of the code does this happen?

Answer if self.visits == self.numparents or start:
if self.source is not None:
self.source.backward()

Question 8

core.py contains the three main Ops, with some more provided in ops.py. Choose one of the ops Normalize, Expand, Select, Squeeze or Unsqueeze, and show that the implementation is correct. That is, for the given forward, derive the backward, and show that it matches what is implemented.

Answer

Question 9

The current network uses a Sigmoid nonlinearity on the hidden layer. Create an Op for a ReLU nonlinearity (details in the last part of the lecture). Retrain the network. Compare the validation accuracy of the Sigmoid and the ReLU versions.

Answer For the comparison, I used the standard parameters that were given by the assignment. Only exception being learning rate, this being 0.001. This was chosen because the prompt to run the file also used 0.001. Figure 1 and 2 show the accuracy and loss. Both have an accuracy that can be deemed as good. Figure 1 shows that Sigmoid converges faster then ReLu. Furthermore, there is a clear difference when looking at the loss. Lastly, ReLu has an inconsisnent pattern when compared to Sigmoid. The model that uses sigmoid has an constant improvement, whereas the ReLu model has several slopes and descents in the graph.

Question 10

Change the network architecture (and other aspects of the model) and show how the training behavior changes for the MNIST data. Here are some ideas (but feel free to try something else). Try adding more layers to the MLP, or widening the network (more nodes in the hidden layer).
- Add a momentum term to the gradient descent. This is discussed in lecture 4.

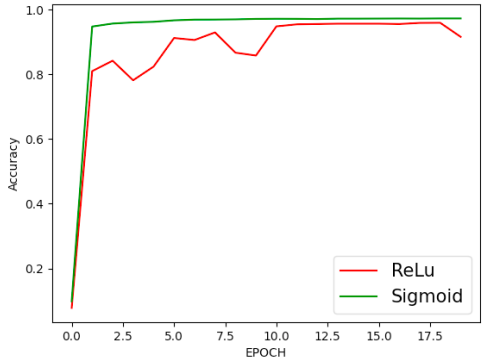

Figure 1: Accuracy with ReLu and Sigmoid on MNIST

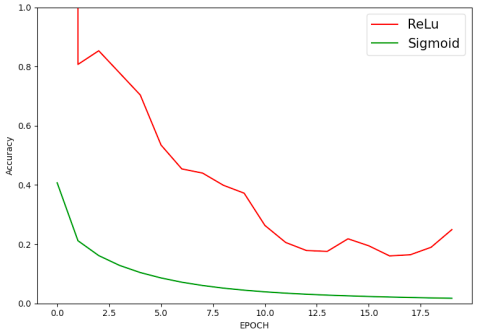

Figure 2: Loss with ReLu and Sigmoid on MNIST

- Try adding a residual connection between layers. These are also discussed in lecture 4.
- It is often said that good initialization is the key to neural network performance. What happens if you replace the Glorot initialization (used in the Linear module) by something else? If you initialize to all 0s or sample from a standard normal distribution, do you see a drop in performance?
- If your computer is too slow to do this quickly enough on the MNIST data, you can also work with the synthetic data, but it may be too simple to show the benefit of things like residual connections. In that case, just report what you find.

Answer To improve the model, I have decided to change the weight initialization. Since this step is important and can effect the outcome, especially when using neural network models like in this assignment. In modules.py, I edited the initialization to: w = np.random.randn(output_size, input_size) * np.sqrt(2.0 / input_size). Figure 3 and 4 show the accuracy and loss from this new model. ReLu has a clear improvement. The model converges faster and now has an identical accuracy as the sigmoid model. Furthermore, the loss has declined. Lastly, the improvement is more constant when comparing it to the old model.

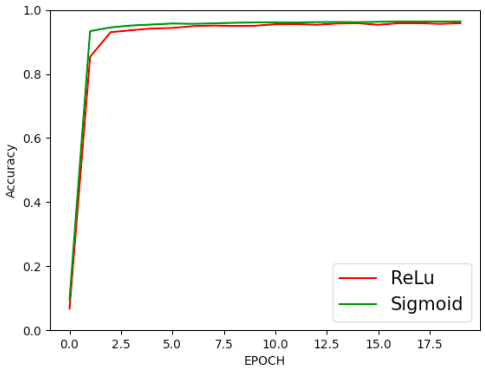

Figure 3: Accuracy with ReLu and Sigmoid on MNIST with different weight initialization

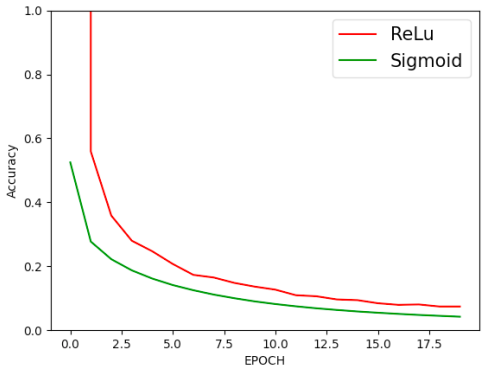

Figure 4: Loss with ReLu and Sigmoid on MNIST with different weight initialization

Question 11

Install pytorch using the installation instructions on its main page. Follow the Pytorch 60-minute blitz. When you've built a classifier, play around with the hyperparameters (learning rate, batch size, nr of epochs, etc) and see what the best accuracies are that you can achieve. Report your hyperparameters and your results.

Answer As the figures show, a learning rate of 0.001 and 0.0001 did not make much difference, however setting an even lower learning rate resulted in bad results. Furthermore, i tried varying the batch sizes, here

a better difference could be seen. A batch size of 100 appeared to perform better. However to conclude this we need to try experiment with a bigger batch size, as can be seen 200 also performs well. A batch size of 300 or larger could also perform well. With such a big dataset not much can actually be said about the optimal parameters, for this more combinations of parameters need to be tested to draw a conclusion.

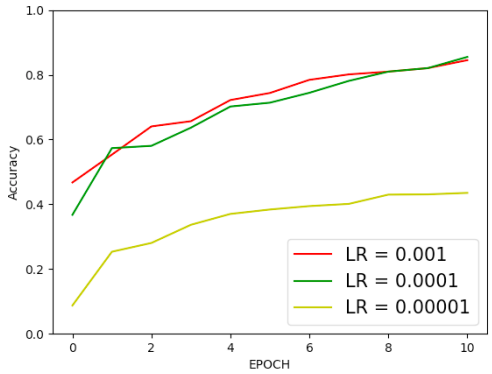

Figure 5: Training accuracy with different learning rates

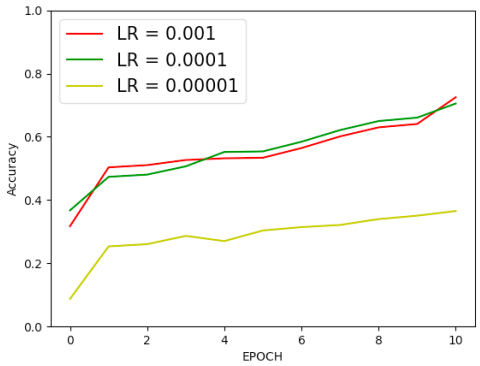

Figure 6: Validation accuracy with different learning rates

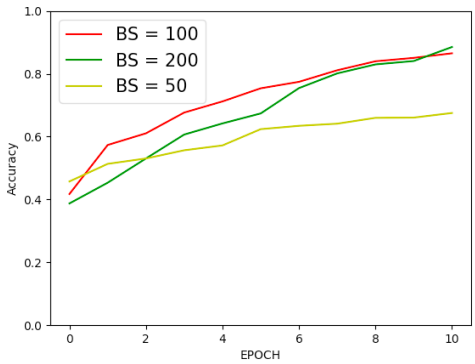

Figure 7: Training accuracy with different batch-sizes

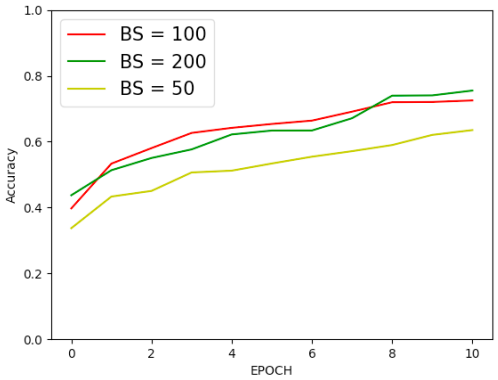

Figure 8: Validation accuracy with different batch-sizes

Question 12

Change some other aspects of the training and report the results. For instance, the package torch.optim contains other optimizers than SGD which you could try. There are also other loss functions. You could even look into some tricks for improving the network architecture, like batch normalization or residual connections. We haven't discussed these in the lectures yet, but there are plenty of resources available online. Don't worry too much about getting a positive result, just report what you tried and what you found.

Answer I tried to change the loss function to mean squared error (MSE), for the parameters i used the best values from the previous question. This did not improve the accuracy and the loss was higher when comparing the two. It was expected that this choice would led to a worse outcome, since it is not guarenteed that MSE will minimize the cost funciton in a classification problem.

References
Indicate papers/books you used for the assignment. References are unlimited. I suggest to use `bibtex` and add sources to `literature.bib`. An example citation would be Eiben et al. (2003) for the running text or otherwise (Eiben et al., 2003).

References

Eiben, A. E., Smith, J. E., et al. (2003). *Introduction to evolutionary computing*, volume 53. Springer.

Appendix

The TAs may look at what you put here, **but they're not obliged to**. This is a good place for, for instance, extra code snippets, additional plots, hyperparameter details, etc.