Deep Learning 2020 **(Due: [18-11-2020])**

Submission Assignment #2

Instructor: Jakub Tomczak *Name:* Jens van Holland*, Netid:* jhd660

In this report the answers to questions 1-9 are given directly, question 10 is written down in a small report structure.

1 Question answers

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.*

Firstly, one starts with the scalar function and let the outcome of $f(X, Y)$ be S,

$$f(X_{ij}, Y_{ij}) = \frac{X_{ij}}{Y_{ij}} (= S_{ij})$$

Next, for X^{∇}_{ij}

$$\begin{aligned}
\frac{\partial Loss}{\partial X_{ij}} &= \sum_{k,l} \frac{\partial Loss}{\partial S_{kl}} \cdot \frac{\partial S_{kl}}{\partial X_{ij}} \\
&= \sum_{k,l} S^{\nabla}_{kl} \cdot \frac{\partial\left(\frac{X_{kl}}{Y_{kl}}\right)}{\partial X_{ij}} \\
&= S^{\nabla}_{ij} \cdot \frac{\partial\left(\frac{X_{ij}}{Y_{ij}}\right)}{\partial X_{ij}} \\
&= S^{\nabla}_{ij} \cdot \frac{1}{Y_{ij}}
\end{aligned}$$

Thus,

$$X^{\nabla} = S^{\nabla} \frac{1}{Y}$$

(element wise)

Next, Y^{∇}_{ij},

$$\begin{aligned}
\frac{\partial Loss}{\partial Y_{ij}} &= \sum_{k,l} \frac{\partial Loss}{\partial S_{kl}} \cdot \frac{\partial S_{kl}}{\partial Y_{ij}} \\
&= \sum_{k,l} S^{\nabla}_{kl} \cdot \frac{\partial\left(\frac{X_{kl}}{Y_{kl}}\right)}{\partial Y_{ij}} \\
&= S^{\nabla}_{ij} \cdot \frac{\partial\left(\frac{X_{ij}}{Y_{ij}}\right)}{\partial Y_{ij}} \\
&= S^{\nabla}_{ij} \cdot \frac{X_{ij}}{Y^2_{ij}}
\end{aligned}$$

Thus,

$$Y^{\nabla} = S^{\nabla}\frac{X}{Y^2}$$
$$\text{(element wise)}$$

Question 2 *Let f be a scalar-to-scalar function. 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.*

Let the outcome of $F(X)$ be S, for the scalar derivative,

$$\begin{aligned} X^{\nabla}_{ij} &= \frac{\partial Loss}{\partial X_{ij}} \\ &= \sum_{k,l} S^{\nabla}_{kl} \cdot \frac{\partial\big(f(X_{kl})\big)}{\partial X_{ij}} \\ &= S^{\nabla}_{ij} \cdot \frac{\partial\big(f(X_{ij})\big)}{\partial X_{ij}} \\ &= S^{\nabla}_{ij} \cdot f'(X_{ij}) \end{aligned}$$

Thus,

$$X^{\nabla} = S^{\nabla} \cdot F'(X)$$
$$\text{(element wise)}$$

Question 3 *Let matrix W be the weights of an MLP layer with f input 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.*

Let the matrix operation that computes the layer outputs be, $K = W \cdot X$.

Next,

$$\begin{aligned} X^{\nabla}_{ij} &= \frac{\partial Loss}{\partial X_{ij}} \\ &= \sum_{m,n} K^{\nabla}_{mn} \cdot \frac{\partial \sum_f \big[W_{mf} \cdot X_{fn}\big]}{\partial X_{ij}} \\ &= \sum_{m,n,f} K^{\nabla}_{mn} \cdot \frac{\partial\big[W_{mf} \cdot X_{fn}\big]}{\partial X_{ij}} \\ &= \sum_{m} K^{\nabla}_{mj} \cdot W_{mi} \cdot \frac{\partial X_{ij}}{\partial X_{ij}} \\ &= \sum_{m} K^{\nabla}_{mj} \cdot W_{mi} \\ &= \sum_{m} K^{\nabla}_{mj} \cdot W^{T}_{im} \end{aligned}$$

Thus,

$$X^{\nabla} = K^{\nabla}\dot{W}^{T}$$

This is not an element wise operation but a dot product.

Next,

$$\begin{aligned}
W_{ij}^{\nabla} &= \frac{\partial Loss}{\partial W_{ij}} \\
&= \sum_{m,n} K_{mn}^{\nabla} \cdot \frac{\partial \sum_l \left[W_{mf} \cdot X_{fn}\right]}{\partial W_{ij}} \\
&= \sum_{m,n,f} K_{mn}^{\nabla} \cdot \frac{\partial \left[W_{mf} \cdot X_{fn}\right]}{\partial W_{ij}} \\
&= \sum_{n} K_{in}^{\nabla} \cdot X_{jn} \cdot \frac{\partial W_{ij}}{\partial W_{ij}} \\
&= \sum_{n} K_{in}^{\nabla} \cdot X_{jn} \\
&= \sum_{n} K_{in}^{\nabla} \cdot X_{nj}^{T}
\end{aligned}$$

Due to shapes, one has to switch the elements[0],

$$= \sum_{n} X_{nj}^{T} \cdot K_{in}^{\nabla}$$

Thus,

$$W^{\nabla} = X^{T} \cdot K^{\nabla}$$

This is not an element wise operation but a dot product.

[0]I am not really sure about this

Question 4 *Let f(x) = Y be a function that takes a vector x , and returns the matrix Y consisting of 16 columns that are all equal to x . Work out the backward of f. (This may seem like a contrived example, but it's actually an instance of broadcasting).*

Firstly, $f(x) = [x^{(0)}, x^{(1)}, ., x^{(l)} .., x^{(14)}, x^{(15)}] = Y$, where (l) denotes the l'th column.

Next,

$$\begin{aligned}
x_i^{\nabla} &= \frac{\partial Loss}{\partial x_i} \\
&= \sum_{kl} Y_{kl}^{\nabla} \cdot \frac{\partial Y_{kl}}{\partial x_i} \\
&= \sum_{l} Y_{il}^{\nabla} \cdot \frac{\partial Y_{il}}{\partial x_i}
\end{aligned}$$

Since each element in a row of Y contains the same value, one can say $Y_{il} = x_i$

.

Using this yields,

$$\begin{aligned}
x_i^{\nabla} &= \sum_{l} Y_{il}^{\nabla} \cdot \frac{\partial x_i}{\partial x_i} \\
&= \sum_{l} Y_{il}^{\nabla}
\end{aligned}$$

Thus,

$$x^{\nabla} = \sum_{l} Y_l^{\nabla}$$

Each element of x^{∇} is the sum of each row of the backward of Y

Question 5 *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?*

The code snippet below shows how a and b are initialized and how c is computed.

```
a = vg.TensorNode(np.random.randn(2,2))
b = vg.TensorNode(np.random.randn(2,2))
c = a+b
```

1. c.value contains the element wise sum of Tensornode a and Tensornode b.
2. c.source refers to the location of the OpNode object where it was computed.
3. c.source.inputs refers to the Tensornodes (a and b). In this case it takes the 0'th element, which is Tensornode a.
4. a.grad refers to the gradient of a, which is always initialized with zero('s).

Question 6

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?*

The answer of the first question is the Op object, the OpNode is defined by the operation it does which is an Op.

The answer of the second question is the code snippet below (commented code for reference).

```
    #def forward(context, a, b):
    #    assert a.shape == b.shape, f'Arrays not the same sizes ({a.shape} {b.shape}).'
        return a + b
```

The answer of the third question is that the opnode has not computed any values and does not know in which Tensornode it needs to store the values yet. The code below shows in which line the OpNode is connected to the output nodes.

```
        #outputs = [TensorNode(value=output, source=opnode) for output in outputs_raw]
        opnode.outputs = outputs
```

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 Op s being called, which do the actual computation. In which line of the code does this happen?*

The code snippet below shows where the backward functions are called.

```
        # -- the gradient of the loss node is 1, with the same shape as the loss node itself
        if self.source is not None:
            self.source.backward()
```

Question 8 *core.py contains the three main Op s, with some provided in ops.py. Pick one of the Ops in ops.py, which isn't an element-wise operation, and show that the implementation is correct.*

The operation that is chosen is the sum operation. I denote the following,

$$F(X) = \sum_i \sum_j X_{ij} = o$$

The backward is,

$$\begin{aligned}
X_{ij}^{\nabla} &= \frac{\partial Loss}{\partial X_{ij}} \\
&= o^{\nabla} \cdot \frac{\partial F(X)}{\partial X_{ij}} \\
&= o^{\nabla} \cdot \frac{\partial \sum_k \sum_l X_{kl}}{\partial X_{ij}} \\
&= o^{\nabla} \cdot \frac{\partial X_{ij}}{\partial X_{ij}} \\
&= o^{\nabla}
\end{aligned}$$

This means for each element X_{ij}^{∇} the backward is o^{∇}.

The vector operation would be,

$$X^{\nabla} = o^{\nabla} \cdot J$$

Where J is a matrix with the same shape as X but all elements are 1.

The code snippet below shows the same. The forward is the summation of all elements (F(X)), the backward is a matrix with the same size as the input (X) where all the elements are the "go" value (gradient up to F(X)). This is the same as explained above.

```
class Sum(Op):
    """
    Op that sums all elements in a tensor together, returning a scalar value.
    """

    @staticmethod
    def forward(context, x):

        context['xsize'] = x.shape
        return np.asarray(x.sum())

    @staticmethod
    def backward(context, go):
        assert go.shape == () # go should be scalar

        xsize = context['xsize']

        return np.full(shape=xsize, fill_value=go)
```

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.*

As can be seen in figure 1, the MLP with the ReLU activation function performs quite well compared to the MLP with the sigmoid activation. Also, it seems like the MLP with the ReLU activation doe not increase the performance of the model after the first epoch. While the MLP with the sigmoid increases the performance until the fifth epoch. The MLP with the ReLU activation also outperforms the MLP with the sigmoid activation after every epoch.

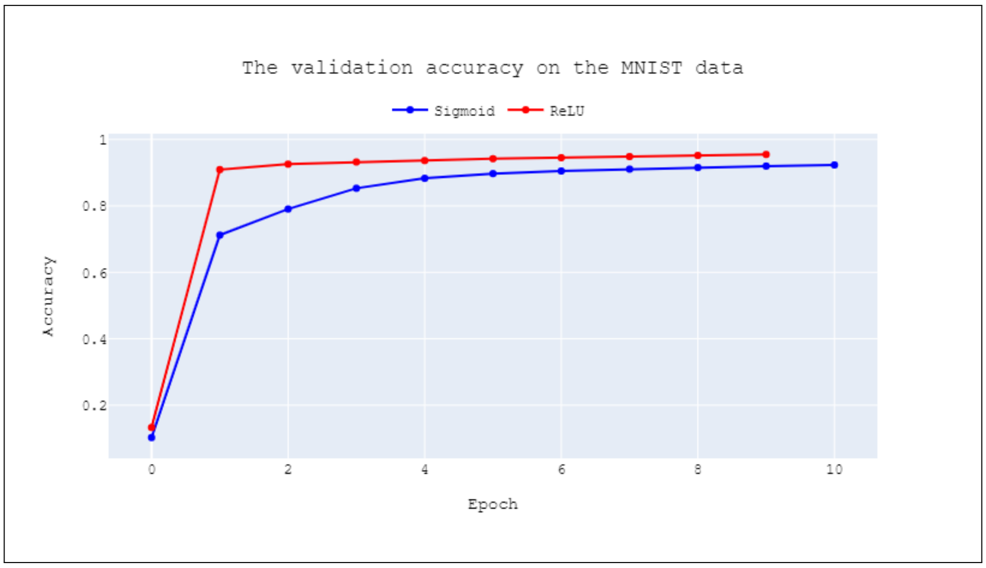

Figure 1: The validation accuracy on the MNIST data set for the MLP's with the sigmoid (blue) and ReLU (blue) hidden layer activation. Both MLP's are trained for 10 epochs with a learningrate of 0.000001 and a batchsize of 128.

The implementation of the ReLU op is shown in appendix A.

2 Problem statement

For this assignment, one has to build a classifier and train it on the CIFAR10 data set. The data set contains images of ten different objects, such as airplanes, birds, trucks, horses and a few other objects. The data set contains 55,000 images, there are 50,000 training samples and 5,000 test samples. Furthermore, this is a balanced data set, meaning that the each class has the same amount of images in the data set. Each image is represented by 32 x 32 pixels with RGB, e.g. the shape of an image is (32, 32, 3) and the shape of the full trainingset is (55,000, 32, 32, 3).

The first objective is to make a working model in PyTorch, the second objective is to maximize the accuracy meaning that the model will try to correctly predict the targets as much as possible.

3 Methodology

Firstly, the first image classifier that is used in this report, is from this Pytorch tutorial. The second classifier that is used is the LeNet-5 model (LeCun et al., 1998).
For training the model, three optimizers are used, namely: SGD, Adam and Adamax.
Next, the original training set is split into a training set and validation set with 45,000 and 5,000 observations, respectively. When the analysis is done, the chosen model will be retrained with the original training set. The final results will be based on the original test set.

4 Experiments

In this section the main analysis will be explained. Since each image contains a lot of values (32*32*3 = 3072), one does not choose a MLP due to the number of parameters for one hidden layer alone. So, a model that does paramater sharing, such as a convolutional model, is used.
The convolutional models that are used:

1. Conv(3,6,5) - ReLU - MaxPool(2,2) - Conv(6,16,5) - ReLU - FC(16*5*5, 120) - ReLU - FC(120, 84) - ReLU - FC(84, 10)

2. Conv(3,6,5) - Tanh - AvePool(2,2) - Conv(6,16,5) - Tanh - AvePool(2,2) - Conv(16, 120, 5) - Tanh - FC(120, 84) - Tanh - FC(84, 10)

The first model stated above is from the tutorial and the second model is the LeNet model[1] (LeCun et al., 1998).
For the first model, three different optimizers[2] are used:

- The SGD optimizer, with a learningrate of 0.001 and a momentum factor of 0.9.
- The Adam optimizer, with a learningrate of 0.0005.
- The Adamax optimizer, with a learningrate of 0.0001.

A batchsize of 64 is used during each training.

[1]Implementation based on Lewinson (2020)
[2]The learningrates are chosen by trial and error.

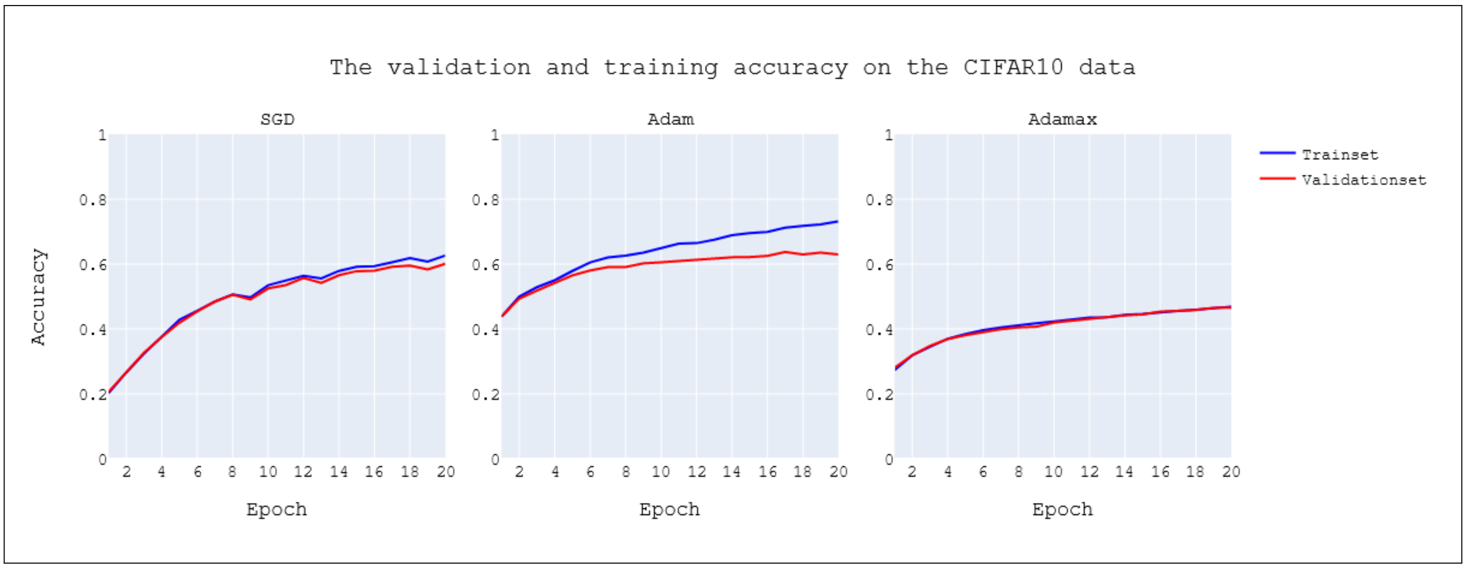

Figure 2: The validation and training accuracy on the CIFAR10 dataset. Using the first stated architecture, with the SGD optimizer (left), Adam optimizer (center) and Adamax optimizer (right).

Based on figure 2, it seems like the model with the SGD optimizer has a smooth learning curve, The Adam optimizer looks promising but the validation and training accuracy seem to be diverging, indicating overfitting. The Adamax has a very smooth learning curve, but the results are not that impressive.
For the second architecture, the same optimizers and learningrates are used. Also, for these runs is the batchsize set to 64.

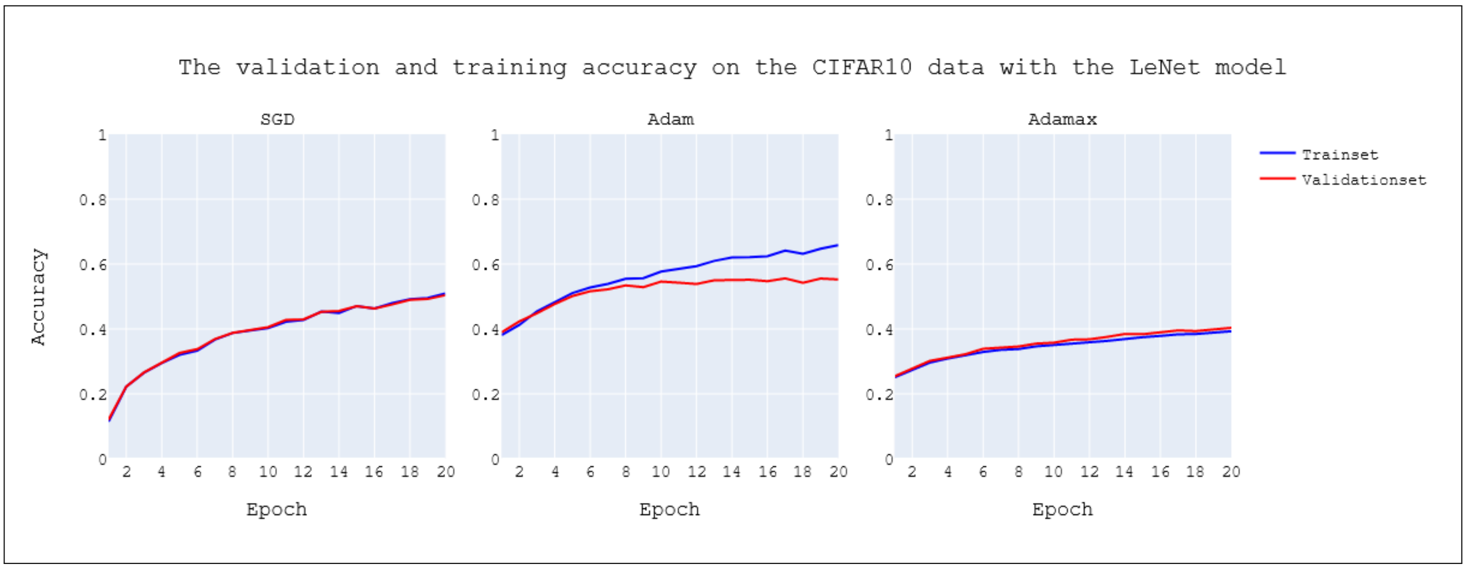

Figure 3: The validation and training accuracy on the CIFAR10 dataset. With the LeNet architecture, using the SGD optimizer (left), Adam optimizer (center) and Adamax optimizer (right).

As can be seen in figure 3, the model trained with the SGD and Adamax optimizer has a smooth learning curve and there is no indication of overfitting. The model trained with the Adam optimizer seems to work best in this case, but the training and validation accuracy seem to be diverging.

(Question 10)
Based on figure 2 and 3, the final run is done with the first stated architecture and the Adam optimizer. The batchsize is set to 64, the learningrate to 0.0005 and the model will be trained for 10 epochs. The number of epochs is based on the fact that the validation accuracy does not improve that much after 10 epochs.
The final results are shown in table 1, the model performs somewhat better on the training set compared to the test set, which indicates a bit over overfitting.

Dataset	Final accuracy
Training	0.664
Test	0.6032

Table 1: The results of the last epoch (10) of the final run on the CIFAR10 dataset.

5 Results and discussion

In this report, two different architectures are compared, namely: the ”tutorial” model and the LeNet-5 model (LeCun et al., 1998). Three different optimizers are used for these two architectures. It is shown in figure 2 and 3 that the Adam optimizer performs better than the SGD and Adamax optimizers. The goal of this assignment question is to make a working model in PyTorch and get an acceptable final run accuracy. The accuracy on the final run is not the best one can get, but it is better than guessing the object in an image.
I believe, due to the training time of the models and computing power of my laptop, this is not the optimal results by far. One could use some other tricks or other architectures to get a better accuracy.
Furthermore, it would be interesting to see a confusion matrix or a classification report[3] of this model. Based on that information, one could finetune the model better.

References

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Lewinson, E. (2020). Implementing yann lecun’s lenet-5 in pytorch. [Online; posted 09-05-2020].

A ReLU implementation

```
# Made in ops.py
class ReLU(Op):
"""
Op for ReLU activation function
"""
    @staticmethod
    def forward(context, input):

        relux =  np.maximum(0,input)
        context['relux'] = relux # store the relu of x for the backward pass
        return relux

    @staticmethod
    def backward(context, goutput):
        relux = context['relux'] # retrieve the relu of x
        goutput[relux<=0] = 0    # gradient values or 0 (since ReLU derivative is 0 or 1)
        return goutput
```

```
 # Made in functions.py
def relu(x):
    """
    Applies relu.
    :param x: A matrix:
    :return: A matrix of the same size as X, with 0 or X value.
    """
    return ReLU.do_forward(x)
```

[3]Due to the time given for this assignment, I did not include a confusion matrix or classification report.