Deep Learning 2020 **(Due: 6 nov 2020)**

Submission Assignment #1

Instructor: Jakub Tomczak *Name:* Mick IJzer, *Netid:* 2552611

The purpose of this assignment is to create a multilayer perceptron and its learning algorithm (stochastic gradient descent) without the use of machine learning libraries such as Keras or Tensorflow. Two neural networks will be build, one by using only scalars and one with tensors.

1 Question answers

Question 1 Gradients with respect to the weights are needed to be able to train a neural network. But to be able to compute those, the derivatives of the final activation function and loss function are needed as well. As final activation function the softmax (see Equation 1.1) is commonly used. A softmax function transforms its inputs to a probability distribution over the classes (i.e. the output of the softmax sums to 1). By using the coefficient rule (see Equation 1.2) and some rewriting (see Equation 1.3) we can compute the derivative of the softmax with respect to o_j. By following the same steps we can also compute the derivative with respect to o_i, which can be seen in Equation 1.4.

$$y_i = \frac{exp(o_i)}{\sum_j exp(o_j)} \tag{1.1}$$

$$\frac{\delta y_i}{\delta o_j} = \frac{\sum_j exp(o_j) * 0 - exp(o_i) * exp(o_j)}{(\sum_j exp(o_j))^2} \tag{1.2}$$

$$= -\frac{exp(o_i)}{\sum_j exp(o_j)} * \frac{exp(o_j)}{\sum_j exp(o_j)} = -y_i * y_j \tag{1.3}$$

$$\frac{\delta y_i}{\delta o_i} = y_i * (1 - y_i) \tag{1.4}$$

A loss function that is widely used for classification tasks is the cross entropy loss (see Equation 1.5). In the function the loss is summed over the possible classes, where y_i is the predicted probability and t_i is an indicator whether that class is true. The derivative with respect to y_i is given in Equation 1.6.

$$loss = E = -\sum_i t_i * log(y_i) \tag{1.5}$$

$$\frac{\delta E}{\delta y_i} = -\frac{t_i}{y_i} \tag{1.6}$$

What is interesting is that the combination of the derivative of the softmax and the derivative of the cross entropy loss results in a really nice gradient of the loss with respect to o_i (see Equation 1.7 and Equation 1.8). This derivative is the predicted probability of a class minus the indicator of whether that class is the true class or not.

$$\frac{\delta E}{\delta o_i} = \sum_k \frac{\delta E}{\delta y_k}\frac{\delta y_k}{\delta o_i} = \frac{\delta E}{\delta y_i}\frac{\delta y_i}{\delta o_i} - \sum_{k \neq i} \frac{\delta E}{\delta y_k}\frac{\delta y_k}{\delta o_i} \tag{1.7}$$

$$= -t_i * (1 - y_i) + \sum_{k \neq i} t_k * y_i = -t_i + \sum_k t_k * y_i = y_i - t_i \tag{1.8}$$

Question 2 The 'scalar' neural network has two inputs, one fully connected hidden layer with three nodes and sigmoid activation, and another fully connected output layer with two outputs and softmax activation. The goal of this network is classification. For this question the inputs and weights are given, after which one forward pass and one backward pass have to be performed. A code snippet of the implementation of the cost function and backward pass can be seen in Section 6. The derivatives of the first set of weights, the first bias, the second set of weights, and the second bias (i.e. dW1, db1, dW2, db2 in the code snipped) are shown in Table 1. The loss at the end of the forward pass is $loss = 0.693$ if the cross entropy loss function in Equation 1.5 is used.

Weight set	**Weights**	**Derivatives**
W1	[[1, 1, 1], [-1, -1, -1]]	[[0.0, 0.0, 0.0], [-0.0, -0.0, -0.0]]
b1	[0, 0, 0]	[0.0, 0.0, 0.0]
W2	[[1, 1], [-1, -1], [-1, -1]]	[[-0.440, 0.440], [-0.440, 0.440], [-0.440, 0.440]]
b2	[0, 0]	[-0.5, 0.5]

Table 1: Weights and their derivatives.

2 Problem statement

As stated in the introduction, the purpose of this assignment is to create a multilayer perceptron and its learning algorithm without the use of machine learning libraries such as Keras or Tensorflow. To achieve this goal two neural networks will be build, one by using only scalars and one with tensors. The 'scalar' network will be used for classification based on a synthetic (SYNTH) dataset. The SYNTH dataset has two features and 2 output classes, and the decision boundary of the classes is an elipse in the feature space. The 'tensor' network will be trained on the MNIST dataset. MNIST is a well known dataset consisting of 28x28 pixel images of handwritten numbers. The goal of using the MNIST dataset is to be able to correctly classify the images with respect to which number is written.

Because both neural networks will be used for classification tasks the final activation function and the loss-function are the same, namely softmax and cross-entropy loss respectively. Lastly, the weights in both network will be updated using stochastic gradient descent. If all of these aspects are correctly implemented the two models will learn, which will result in a decrease of the cross-entropy loss over the epochs. Besides the cross entropy loss the f1 score, accuracy, and a confusion matrix will be used to assess the performance of the 'tensor' network.

3 Methodology

The training of the networks, as well as the computation of the training and validation loss are schematically shown in pseudocode in Algorithm 1. First, the bias weights are set to zero and the other weights are randomly sampled from a normal distribution with parameters $\mu = 0, std = 0.5$. Then, for each epoch, every instance in the training set is passed through the network. For every instance the loss and gradients are computed, after which the weights of the network are updated according to the learning rate. Lastly, at the end of each epoch both the training set and validation set are passed through the network to compute the training loss and validation loss.

4 Experiments

The goal of the 'scalar' network was to create a neural network that was able to learn without using any python packages except the standard math module. Therefore, the experimental setup for this network is quite simple. The training set of the SYNTH dataset is passed through the network after which the weights are updated, then the training and validation loss are computed. This process is repeated for 15 epochs. Furthermore, the entirety of training the network is also repeated 5 times because of the effects that the random initialization of the weights and randomized order of training instances can have on the learning process. The result should be a loss-curve that reliably goes down over the epochs.

For the 'tensor' network a training, validation, and test set are used. The experimental setup is similar to the 'scalar' network in a way that the network is trained on the training set for several epochs, and is also

Algorithm 1: General algorithm of the Neural Network

```
Result: A Trained Neural Network
Initialize(Weights);
for each epoch do
    for each instance, target in train do
        activation = forward pass(weights, instance);
        instance loss = loss(activation, target);
        gradients = backward pass(weights, instance loss);
        weights = update weights(weights, gradients);
    end
    train predictions = forward pass(weights, x_train);
    training loss = loss(train predictions, y_train);
    val predictions = forward pass(weights, x_val);
    validation loss = loss(val predictions, y_val);
end
```

trained multiple times to combat random effects. The experimental setup slightly differs because the effect of different learning rates is studied using the training and validation set. This means that for each learning rate the experiment is repeated multiple times to obtain reliable performance measures. Based on the performances one learning rate is selected and used to train a network on the complete training and validation set, after which the performance of the test set is evaluated. Table 2 schematically shows the (experimental) setup for both networks. Note that both networks only have one single hidden layer.

	'Scalar' Network	**'Tensor' Network**
Dataset	SYNTH	MNIST
Training Instances	60.000	55.000
Validation Instances	10.000	5.000
Test Instances	-	10.000
Input Shape	2	784
Hidden Nodes	3	300
Output Shape	2	10
Weight Initialization	$\mu = 0, std = 0.5$	$\mu = 0, std = 0.5$
Epochs	15	5
Repetitions	5	3
Learning Rate	0.0001	[0.05, 0.01, 0.001]
Performance Metrics	X-Entropy Loss	X-Entropy Loss, Accuracy, F1, Confusion Matrix

Table 2: Experimental Setup for the Scalar and Tensor Networks.

5 Results and discussion

The mean and standard deviation of the loss of the different repetitions of the 'scalar' network are shown in Figure 1. Note that the mean and standard deviation of the training and validation datasets are nearly equal for all epochs. This means that there is no significant underfitting or overfitting going on. Furthermore, it can be seen that the only significant variations between the repetitions occur for the epochs between epoch 3 and epoch 8. An explanation of this observation could be that depending on the initialization of the weights the network sometimes needs time to find the path to the global minimum, while other initializations are easier to start from. Even when it takes some epochs to see a significant reduction in loss, the loss did converge for all repetitions. Overall, it can be concluded that the 'scalar' network is able to minimize the cross-entropy loss by using gradient descent.

Similar graphs are shown in Figure 2 for the 'tensor' network for each learning rate that was tested. As can be seen in the graphs the loss on the learning and validation set both become smaller over the epochs. The learning rate of 0.05 seems to be too high, since the validation starts to diverge from the training loss, which can indicate overfitting. Furthermore, due to the limited number of epochs, the learning rate of 0.001 is too low. Both the training and validation loss are still relatively high after 5 epochs compared to the other learning rates that were tested.

Therefore, the learning rate of 0.01 was selected to train the network on the complete training and validation set for 5 epochs. After which the performance was evaluated on the unseen test set. The cross entropy loss on the test set was 0.228, with an accuracy of 96.09%, and an F1 score of .96. From these performance measures the conclusion can be drawn that the network is able to correctly classify almost all images that are passed through the network. Lastly, because the performance is not perfect a confusion matrix was constructed to identify the numbers the network struggles with. In general errors occured for all numbers at similar rates. However, the images of the numbers '9' and '5' were incorrectly classified more often than the other numbers, 16.7% and 14.7% of the total number of wrongly classified instances were from those two classes respectively.

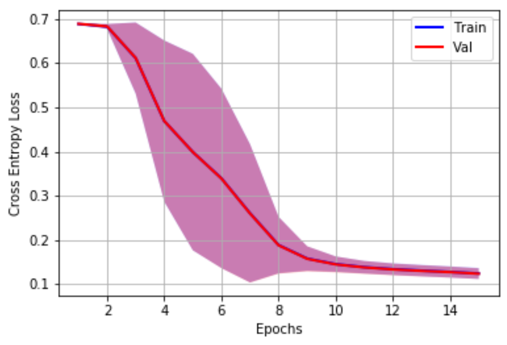

Figure 1: Loss on Synth over Time

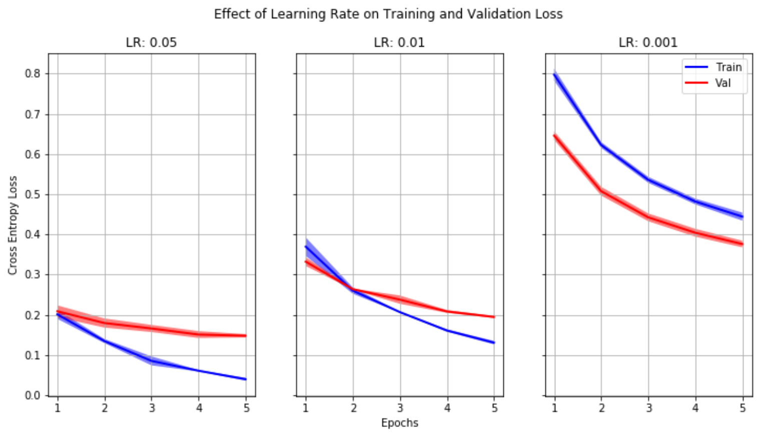

Figure 2: Loss on MNIST over Time

6 A code snippet

```
def compute_loss(a2, y):
    '''
    Computes the cross entropy loss given a prediction and true class.
    Also gives the derivative with respect to o_{j}
    '''

    y_ = [0, 0]
    y_[y] = 1

    loss = 0
    for i in range(len(a2)):
        if (a2[i] != 0) and (a2[i] != 1):
            loss -= y_[i]*math.log(a2[i])

    dz2 = [0, 0]
    for i in range(len(dz2)):  # 2
        dz2[i] = a2[i] - y_[i]

    return loss, dz2

def backward_pass(dz2, a1, w2, x):
    '''
    Computes the derivatives of the loss with respect to the weights.
    '''
    da1 = [0, 0, 0]
    dw2 = [[0, 0],
           [0, 0],
           [0, 0]]
    db2 = [0, 0]
    for i, dz in enumerate(dz2):
        db2[i] = dz
        for j in range(len(a1)):
            dw2[j][i] = dz*a1[j]
            da1[j] += dz*w2[j][i]

    dz1 = [0, 0, 0]
    for i in range(len(dz1)):
        dz1[i] = da1[i]*a1[i]*(1 - a1[i])

    dw1 = [[0, 0, 0],
           [0, 0, 0]]
    db1 = [0, 0, 0]
    for i in range(len(dw1[0])):
        db1[i] = dz1[i]
        for j in range(len(dw1)):
            dw1[j][i] = dz1[i]*x[j]

    return dw2, db2, dw1, db1
```