Radovan Simic 12313464
Vasil Dakov 12307157

1 Computing Gradients and Automatic Differention

1 and 2: The missing values in the tables are $N_I = 2$ and $N_O = 5$. The learnable parameters are $\mathbf{W}^0 \in \mathbb{R}^{12\times 2}$, $\mathbf{W}_1 \in \mathbb{R}^{5\times 12}$, $\mathbf{b}^{(0)} \in \mathbb{R}^{12}$ and $\mathbf{b}^{(1)} \in \mathbb{R}^5$. In $\mathbf{W}^0$ we have 24 learnable parameters, in $\mathbf{W}^{(1)}$ we have 60, in $\mathbf{b}^{(0)}$ we have 12 and in $\mathbf{b}^{(1)}$ we have 5, giving us a total of 101 learnable parameters.

3: With the convention given in the assignment, we know that $\mathbf{z}^{(0)} = \mathbf{a}^{(0)} = \mathbf{x}$ and we know:

$$\mathbf{z}^{(1)} = \mathbf{h}^{(1)} + \mathbf{b}^{(0)} = \mathbf{W}^{(0)}\mathbf{x} + \mathbf{b}^{(0)} \;\;\&\;\; \mathbf{a}^{(1)} = \text{softplus}(\mathbf{z}^{(1)})$$

$$\mathbf{z}^{(2)} = \mathbf{h}^{(2)} + \mathbf{b}^{(1)} = \mathbf{W}^{(1)}\mathbf{a}^{(1)} + \mathbf{b}^{(1)} \;\;\&\;\; \mathbf{a}^{(2)} = \text{softmax}(\mathbf{z}^{(2)})$$

Additionally we know the single sample and whole sample set loss and are defined as follow:

$$l(\mathbf{a}^2, \mathbf{y}^{OH}) = -\sum_i y_i^{OH} * \ln(a_i^2)$$

$$\mathcal{L} = \frac{1}{S}\sum_s l(\mathbf{a}^{(2)s}, \mathbf{y}^{OH})$$

Where the $\mathbf{z}^{(1)}, \mathbf{a}^{(1)} \in R^{12}$, $\mathbf{z}^{(2)}, \mathbf{a}^{(2)}, \mathbf{y}^{OH} \in R^5$ and $\mathbf{y}^{OH}$ is the one hot encoding of the label y and x is the input in $\mathbb{R}^2$. Additionally, the soft plus and softmax functions are applied element-wise and calculated with the formulas given in the assignment. Each sample has its loss l calculated for it, and to get the final loss $\mathcal{L}$, they are all averaged.

4: The computational graph can be seen on Figure 1. We can see here how each variable is calculated using matrix multiplication (*) and addition (+) with the softplus and softmax functions as activation. The variable l is the cross-entropy loss of the input $\mathbf{x}$ and the one-hot encoded class label $\mathbf{y}^{OH}$. This loss is then accounted for in the final loss $\mathcal{L}$ by averaging all of the losses in the sample.

This graph is only meant to represent a single forward pass of a single instance for a single sample and the calculation of the cross-entropy loss for it. The variable $\mathcal{L}$ was added to show how the output value l is added to the averaging of $\mathcal{L}$ which will be minimez. The actual calculation is done S times for the S samples in the dataset in the forward and backward pass accordingly.

5: The idea of training a neural network is nothing more than minimizing the final loss $\mathcal{L}$ with in this case the steepest gradient descent method. For that, we need the gradient of each learnable parameter $\theta = \{\mathbf{W^{(0)}}, \mathbf{b^{(0)}}, \mathbf{W^{(1)}}, \mathbf{b^{(1)}}\}$ in there current values. For any parameter $p \in \theta$ the gradient of $\mathcal{L}$ with respect of p is:

$$\frac{\partial L}{\partial p} = \frac{1}{S}\sum_s \frac{\partial l(\mathbf{a}^{(2)s}, \mathbf{y}^{OH})}{\partial p}$$

which is just the average gradient of the losses for all sample points. To calculate a single gradient we use the backward pass for a specific forward pass formally known as the reverse automatic differing. The backward pass is the combination of the chain rule with the values of the forward pass values used to calculate the derivatives faster. It examines how much the tuning of which parameters affects the final output of the network. We first derive the chain rule aspect as:

$$\delta_1 = \frac{\partial l}{\partial \mathbf{a}^{(2)}} \cdot \frac{\partial \mathbf{a}^{(2)}}{\partial \mathbf{z}^{(2)}} \in R^5$$

$$\delta_2 = \delta_1 \cdot \frac{\partial \mathbf{z}^{(2)}}{\partial \mathbf{h}^{(2)}} \cdot \frac{\partial \mathbf{h}^{(2)}}{\partial \mathbf{a}^{(1)}} \cdot \frac{\partial \mathbf{a}^{(1)}}{\partial \mathbf{z}^{(1)}} \in R^{12}$$

$$\frac{\partial l}{\partial \mathbf{b}^{(1)}} = \delta_1 \cdot \frac{\partial \mathbf{z}^{(2)}}{\partial \mathbf{b}^{(1)}} \in R^5$$

$$\frac{\partial l}{\partial \mathbf{W}^{(1)}} = \delta_1 \cdot \frac{\partial \mathbf{z}^{(2)}}{\partial \mathbf{h}^{(2)}} \cdot \frac{\partial \mathbf{h}^{(2)}}{\partial \mathbf{W}^{(1)}} \in \mathbb{R}^{5\times 12}$$

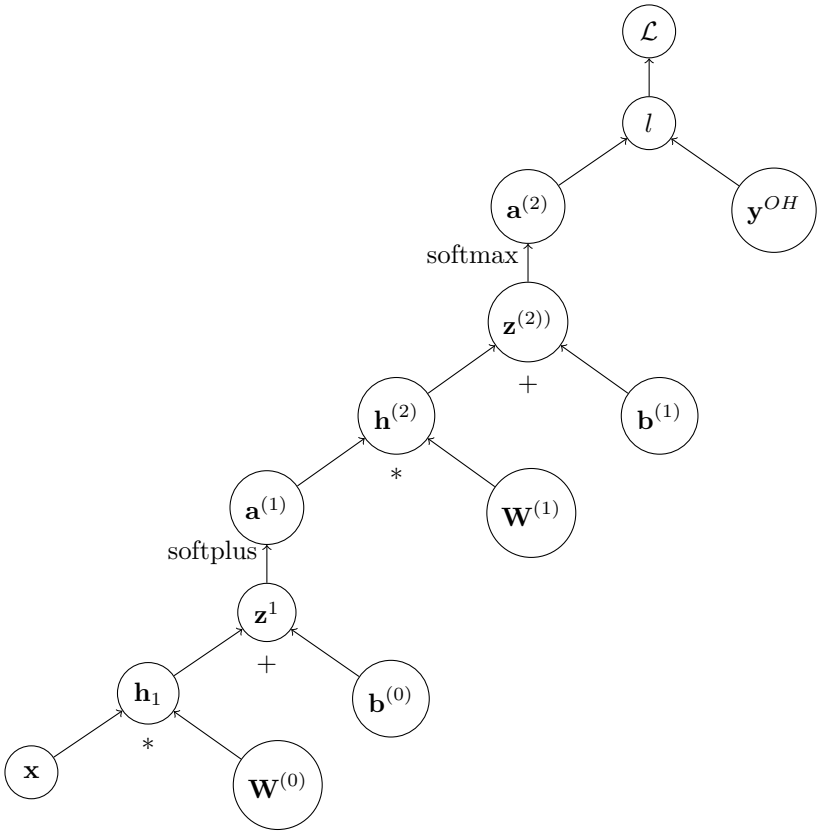

Figure 1: The computational graph

$$\frac{\partial l}{\partial \mathbf{b}^{(0)}} = \delta_2 \cdot \frac{\partial \mathbf{z}^{(1)}}{\partial \mathbf{b}^{(0)}} \in \mathbb{R}^{12}$$
$$\frac{\partial l}{\partial \mathbf{W}^{(0)}} = \delta_2 \cdot \frac{\partial \mathbf{z}^{(1)}}{\partial \mathbf{h}^{(1)}} \cdot \frac{\partial \mathbf{h}^{(1)}}{\partial \mathbf{W}^{(0)}} \in \mathbb{R}^{12\times 2}$$

Where traditionally we would calculate each derivative step by step, but for convenience we wrote a few jumps with respect to our computational graph. We then find each of the needed partial derivatives:

$$\frac{\partial l}{\partial \mathbf{a}^{(2)}}_k = -\sum_i y_i^{OH} * \frac{\partial \ln(a_i^{(2)})}{\partial \mathbf{a}_k^{(2)}} = -y_k^{OH} * \frac{1}{\mathbf{a}_k^{(2)}} \rightarrow \frac{\partial l}{\partial \mathbf{a}^{(2)}} - 1 \cdot diag(a_2)^{-1} * y^{OH} \in \mathbb{R}^5$$

$$\frac{\partial \mathbf{a}^{(2)}}{\partial \mathbf{z}^{(2)}}_{i,j} = \frac{\partial a_i^{(2)}}{\partial z_j^{(2)}} = \frac{\partial}{\partial z_j^{(2)}} \frac{e^{z_i^{(2)}}}{\sum_{k=0}^{N_O} e^{z_k^{(2)}}} = \mathbb{I}(i=j)\left(\frac{e^{z_i^{(2)}}}{\sum_{k=1}^{N_0} e^{z_k^{(2)}}} - \frac{(e^{z_i^{(2)}})^2}{(\sum_{k=1}^{N_0} e^{z_k^{(2)}})^2}\right) + \mathbb{I}(i\neq j)\left(-\frac{e^{z_i^{(2)}} e^{z_j^{(2)}}}{\sum_{k=1}^{N_0} e^{z_k^{(2)}}}\right) =$$
$$= \mathbb{I}(i=j)\left(a_i^{(2)} - (a_i^{(2)})^2\right) + \mathbb{I}(i\neq j)\left(-a_i^{(2)} a_j^{(2)}\right) = a_i^{(2)} \cdot (\mathbb{I}(i=j) - a_j^{(2)}) \in \mathbb{R}$$

$$\frac{\partial \mathbf{a}^{(2)}}{\partial \mathbf{z}^{(2)}} = diag(a_2) \cdot (Id - 1 \cdot a_2^T) \in \mathbb{R}^{5\times 5} (\text{Derived from } \frac{\partial \mathbf{a}^{(2)}}{\partial \mathbf{z}^{(2)}}_{i,j})$$
$$(\frac{\partial \mathbf{a}^{(1)}}{\partial \mathbf{z}^{(1)}})_{i,j} = \frac{\partial a_i^{(1)}}{\partial z_j^{(1)}} = \frac{\partial}{\partial z_j^{(1)}} \ln(1 + e^{z_i^{(1)}}) = \mathbb{I}(i=j) \cdot \frac{e^{z_i^{(1)}}}{1 + e^{z_i^{(1)}}} \in \mathbb{R}$$
$$\frac{\partial \mathbf{a}^{(1)}}{\partial \mathbf{z}^{(1)}} = diag(\text{dsoftplus}(z_1)) \in \mathbb{R}^{12\times 12} (\text{Derived from } \frac{\partial \mathbf{a}^{(1)}}{\partial \mathbf{z}^{(1)}}_{i,j})$$

where $diag(v)$ is the diagonal matrix with the vector elements on the diagonal, Id is the corresponding identity matrix, and dsoftplus is the elementwise derivative of the softplus function.

From the notes of the practicals we have the following derivatives:

$$\frac{\partial \mathbf{h}^{(1)}}{\partial \mathbf{W}^{(0)}} = x^T \in R^{1\times 2}$$

$$\frac{\partial \mathbf{h}^{(2)}}{\partial \mathbf{W}^{(1)}} = a_1^T \in R^{1\times 12}$$

$$\frac{\partial \mathbf{h}^{(2)}}{\partial \mathbf{a}^{(1)}} = \mathbf{W}^{(1)} \in R^{5\times 12}$$

$$\frac{\partial \mathbf{z}^{(2)}}{\partial \mathbf{h}^{(2)}} = Id \in R^{5\times 5}$$

$$\frac{\partial \mathbf{z}^{(2)}}{\partial \mathbf{b}^{(1)}} = Id \in R^{5\times 5}$$

$$\frac{\partial \mathbf{z}^{(1)}}{\partial \mathbf{h}^{(1)}} = Id \in R^{12\times 12}$$

$$\frac{\partial \mathbf{z}^{(1)}}{\partial \mathbf{b}^{(0)}} = Id \in R^{12\times 5}$$

After finding all the derivatives we still need to make the chain rule multiplication make sense, given that this are matrix-vector multiplication this is relatively easy to do by matching the dimension and the results are:

$$\delta_1 = (\frac{\partial L}{\partial \mathbf{a}^{(2)}}^T \cdot \frac{\partial \mathbf{a}^{(2)}}{\partial \mathbf{z}^{(2)}})^T \in R^5$$

$$\delta_2 = (\delta_1^T \cdot \frac{\partial \mathbf{z}^{(2)}}{\partial \mathbf{h}^{(2)}} \cdot \frac{\partial \mathbf{h}^{(2)}}{\partial \mathbf{a}^{(1)}} \cdot \frac{\partial \mathbf{a}^{(1)}}{\partial \mathbf{z}^{(1)}})^T \in R^{12}$$

$$\frac{\partial l}{\partial \mathbf{b}^{(1)}} = (\delta_1^T \cdot \frac{\partial \mathbf{z}^{(2)}}{\partial \mathbf{b}^{(1)}})^T$$

$$\frac{\partial l}{\partial \mathbf{W}^{(1)}} = (\delta_1^T \cdot \frac{\partial \mathbf{z}^{(2)}}{\partial \mathbf{h}^{(2)}})^T \cdot \frac{\partial \mathbf{h}^{(2)}}{\partial \mathbf{W}^{(1)}}$$

$$\frac{\partial l}{\partial \mathbf{b}^{(0)}} = (\delta_2^T \cdot \frac{\partial \mathbf{z}^{(1)}}{\partial \mathbf{b}^{(0)}})^T$$

$$\frac{\partial l}{\partial \mathbf{W}^{(0)}} = (\delta_2^T \cdot \frac{\partial \mathbf{z}^{(1)}}{\partial \mathbf{h}^{(1)}})^T \cdot \frac{\partial \mathbf{h}^{(1)}}{\partial \mathbf{W}^{(0)}}$$

and all that is needed is to plug in the values. This can be seen in the backward function of the **main.py** file where we calculate the derivative of a single forward pass loss.

8. The gradient check was done by checking the gradient of a single loss function for the whole training set and can be seen in the check_gradient function. It returned that our analytically calculated gradients are correct.

2 Training the Neural Network

2. Different learning rates were tastet and selected. Namely $\alpha = 0.01, 0.1, 0.15, 0.25$. Out of them, the best-performing one was $\alpha = 0.1$. The criterion by which they were compared were the accuracy of the output achieved on the training set were final loss achieved, and stability of training. Learning rates above 0.1 had large fluctuations in their training loss, with it taking up and down and having a harder time converging. $\alpha = 0.01$ on the other hand was converging too slowly and achieved a higher loss. The 0.1 step size achieved the best of both worlds.

Different numbers of epochs were tested too. As the training was quick in the end we settled on 1000 epochs. Any more than that and the model curves started overfitting to our training set too much, and despite achieving higher performance on the training set, they were far less generalizable.

3. The plot can be found in the uploaded **figures.pdf** and shows a steady decline in the loss function. Given that the steepest gradient descent was used and that with later iterations our learning rate stays the same this is a good result. After training we ran all of the training data through the model and predicted the class by choosing the index of the largest element of the $\mathbf{a}^{(2)}$ vector without updating the parameters. The accuracy of the model on the training data was 65.75%.

4. The final parameters can be seen in the uploaded **model.json** file.

3 Inference

1. After training the network and finalizing the network parameters in θ, we took the test samples and passed them through our network without the backward pass, the predicted class was chosen as the index of the largest element of the $\mathbf{a}^{(2)}$ vector. The accuracy was 66% and the final error on the cross-entropy was 0.702.

2. and 3. The confusion matrix with the colour bar associated with the value of its elements can be seen in the uploaded **figures.pdf** file. As can be seen on it our model had a hard time distinguishing the instances in the first three classes, while it had practically no mistakes in Class 3 and Class 4. This makes due to the distribution of the classes in our feature space. Class 0 - Class 2 all have complex circular boundaries within each other, that are difficult to be made separable. It is simply a challenging case, where it is difficult for the networ to be exact, further reinforced by the fact that training data overlaps there too.

With more data, this may be improved.

3. It can also be seen **figures.pdf** file. Reinforcing the information from the confusion matrix, the boundaries between Class 3 and Class 4 and the rest are simple and clear, while the ones around the rest are more complex. As some of the training data overlaps for Classes 0-2, the decision boundaries are also a tad more complex, along with it being a complicated case. Despite all this, the fine-tuned model seems to do a good job in creating a general decision curve with reasonable accuracy.