Artificial Intelligence II
Assignment 1 Report

Andreas - Theologos Spanopoulos (sdi1700146@di.uoa.gr)

November 13, 2020

Exercise 1

1. Equation (4.5) can be trivially proven as follows. Since point $\mathbf{x}$ lies on the decision surface, then it follows that

   $$y(\mathbf{x}) = 0 \Leftrightarrow \mathbf{w}^T\mathbf{x} + w_0 = 0 \Leftrightarrow \mathbf{w}^T\mathbf{x} = -w_0$$

   Dividing both sides by $||\mathbf{w}||$ yields

   $$\frac{\mathbf{w}^T\mathbf{x}}{||\mathbf{w}||} = -\frac{w_0}{||\mathbf{w}||}$$

   which is what we wanted to prove. ■

2. Equation (4.6) is a bit harder to prove. First we have to recall the Orthogonal Decomposition Theorem which states that if W is a subspace of $\mathbb{R}^n$, then any vector $\mathbf{x}$ can be "decomposed" in a sum of two vectors $\mathbf{x}_\perp$ and $\mathbf{z}$, where $\mathbf{x}_\perp$ is the orthogonal projection of $\mathbf{x}$ on W and $\mathbf{z}$ is a vector orthogonal to $\mathbf{x}_\perp$.

   In our case, the subspace W is defined as

   $$W = \{\mathbf{x} \in \mathbb{R}^n \mid y(\mathbf{x}) = 0\}$$

   And therefore $\mathbf{x}$ can be expressed as

   $$\mathbf{x} = \mathbf{x}_\perp + \mathbf{z}$$

   So to prove equation (4.6), we just need to show that

   $$\mathbf{z} = r\frac{\mathbf{w}}{||\mathbf{w}||} \tag{1}$$

   From the big paragraph in page 181 of the "Pattern Recognition and Machine Learning" textbook, we know that $\mathbf{w}$ is orthogonal to every vector lying within the decision surface, which makes it parallel to $\mathbf{z}$.

   Therefore, $\mathbf{z}$ can be expressed as

   $$\mathbf{z} = \alpha\,\mathbf{w} \tag{2}$$

   for some arbitrary α.

Also, we know that the norm of $\mathbf{z}$, that is, $||\mathbf{z}||$, is equal to r (from the definition of r). Thus, we can write

$$||\mathbf{z}|| = r \;\Leftrightarrow\; ||\alpha\,\mathbf{w}|| = r \;\Leftrightarrow\; \alpha||\mathbf{w}|| = r \;\Leftrightarrow\; \alpha = \frac{r}{||\mathbf{w}||}$$

which if substituted with equations (2) and (1), yields

$$\mathbf{x} = \mathbf{x}_{\perp} + r\,\frac{\mathbf{w}}{||\mathbf{w}||}$$

$\blacksquare$

Exercise 2

Let's start by defining our variables in a readable matrix form:

$$\mathbf{x} = \begin{bmatrix} x_1 & x_2 & \cdots & x_n \end{bmatrix}$$

and

$$\mathbf{W} = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1m} \\ w_{21} & w_{22} & \cdots & w_{2m} \\ \vdots & \vdots & \ddots & \cdots \\ w_{n1} & w_{n2} & \cdots & w_{nm} \end{bmatrix}$$

Then, $\mathbf{z}$ becomes

$$\mathbf{z} = \mathbf{xW} = \begin{bmatrix} \sum_{i=1}^{n} x_i w_{i1} & \sum_{i=1}^{n} x_i w_{i2} & \cdots & \sum_{i=1}^{n} x_i w_{im} \end{bmatrix}$$

and so we have that

$$\frac{\partial \mathbf{z}}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial \sum_{i=1}^{n} x_i w_{i1}}{\partial x_1} & \frac{\partial \sum_{i=1}^{n} x_i w_{i2}}{\partial x_1} & \cdots & \frac{\partial \sum_{i=1}^{n} x_i w_{im}}{\partial x_1} \\ \frac{\partial \sum_{i=1}^{n} x_i w_{i1}}{\partial x_2} & \frac{\partial \sum_{i=1}^{n} x_i w_{i2}}{\partial x_2} & \cdots & \frac{\partial \sum_{i=1}^{n} x_i w_{im}}{\partial x_2} \\ \vdots & \vdots & \ddots & \cdots \\ \frac{\partial \sum_{i=1}^{n} x_i w_{i1}}{\partial x_n} & \frac{\partial \sum_{i=1}^{n} x_i w_{i2}}{\partial x_n} & \cdots & \frac{\partial \sum_{i=1}^{n} x_i w_{im}}{\partial x_n} \end{bmatrix}$$

which is (obviously) equal to

$$\begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1m} \\ w_{21} & w_{22} & \cdots & w_{2m} \\ \vdots & \vdots & \ddots & \cdots \\ w_{n1} & w_{n2} & \cdots & w_{nm} \end{bmatrix} = \mathbf{W}$$

Exercise 3

Again, let's define the variables in their readable matrix form:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \ \mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix}$$

Also, the sigmoid (or logistic) activation function $\sigma(x)$ is defined as:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

which composed with $\mathbf{x}^T\mathbf{w}$ gives

$$\sigma(\mathbf{x}^T\mathbf{w}) = \frac{1}{1 + e^{-\mathbf{x}^T\mathbf{w}}} = \frac{1}{1 + e^{-\sum\limits_{i=1}^{n} x_i w_i}}$$

Thus, the gradient of $\hat{y} = \sigma(\mathbf{x}^T\mathbf{w})$ w.r.t. w can be calculated as

$$\frac{\partial \hat{y}}{\partial \mathbf{w}} = \begin{bmatrix} \frac{\partial \hat{y}}{\partial w_1} & \frac{\partial \hat{y}}{\partial w_2} & \cdots & \frac{\partial \hat{y}}{\partial w_n} \end{bmatrix}^T$$

The gradient of $\hat{y}$ w.r.t. any weight w_k, for $k = 1, 2, .., n$ can be computed as follows:

$$\frac{\partial \hat{y}}{\partial w_k} = \frac{\partial \frac{1}{1+e^{-\sum_{i=1}^{n} x_i w_i}}}{\partial w_k} = \frac{x_k\, e^{-\sum_{i=1}^{n} x_i w_i}}{(1 + e^{-\sum_{i=1}^{n} x_i w_i})^2} = \frac{x_k}{1 + e^{-\sum_{i=1}^{n} x_i w_i}} \left(\frac{1 + e^{-\sum_{i=1}^{n} x_i w_i} - 1}{1 + e^{-\sum_{i=1}^{n} x_i w_i}} \right)$$

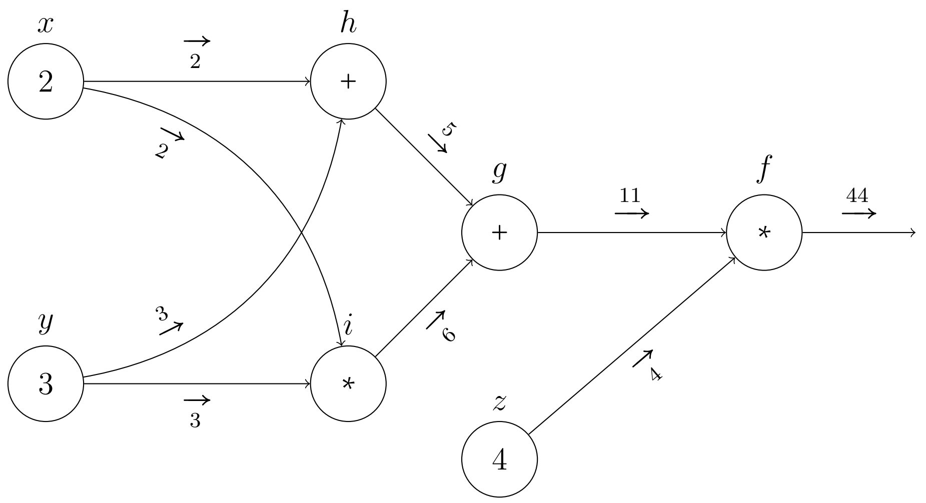
$$= x_k \left(\frac{1}{1 + e^{-\sum_{i=1}^{n} x_i w_i}} \right) \left(1 - \frac{1}{1 + e^{-\sum_{i=1}^{n} x_i w_i}} \right) = x_k\, \hat{y}\, (1 - \hat{y})$$

which makes the gradient of $\hat{y}$ be equal to:

$$\frac{\partial \hat{y}}{\partial \mathbf{w}} = \begin{bmatrix} x_1\, \hat{y}\, (1 - \hat{y}) \\ x_2\, \hat{y}\, (1 - \hat{y}) \\ \cdots \\ x_n\, \hat{y}\, (1 - \hat{y}) \end{bmatrix} = \hat{y}(1 - \hat{y}) \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \hat{y}(1 - \hat{y})\mathbf{x}$$

Exercise 4

Let's first start by doing Forward Propagation in the Computational Graph:



Now let's formalize this in a more strict methamatical notation:

1. $h = x + y$
2. $i = x \times y$
3. $g = h + i$, which expands to $g = (x + y) + (x \times y)$
4. $f = g \times z$, which expands to $f = z \times (x + y) + z \times x \times y$

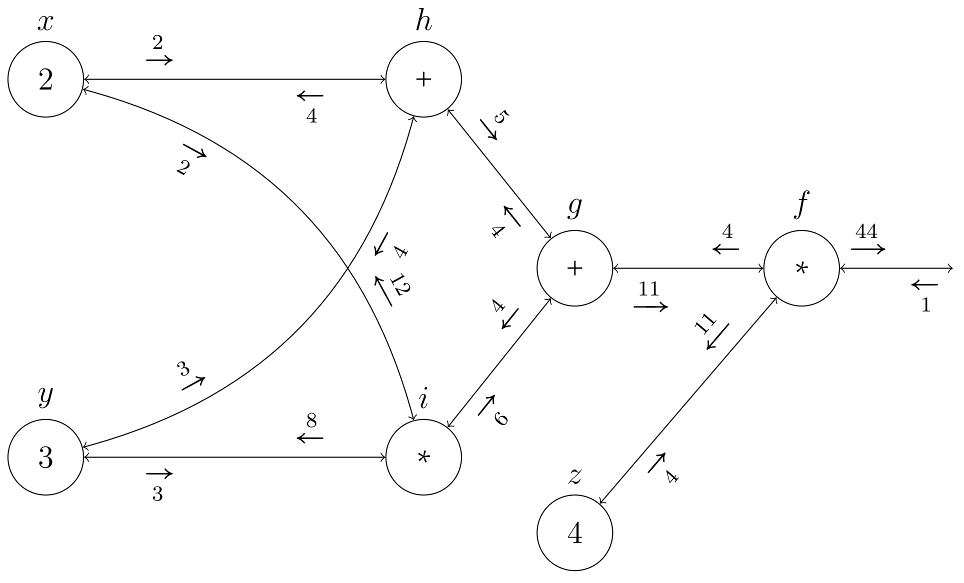
This formalization allows us to compute the local gradients as follows:

1. For the variable h:
   - $\frac{\partial h}{\partial x} = 1, \quad \frac{\partial h}{\partial y} = 1$
2. For the variable i:
   - $\frac{\partial i}{\partial x} = y = 3, \quad \frac{\partial i}{\partial y} = x = 2$
3. For the variable g:
   - $\frac{\partial g}{\partial h} = 1, \quad \frac{\partial g}{\partial i} = 1$
4. For the variable f:
   - $\frac{\partial f}{\partial g} = z, \quad \frac{\partial f}{\partial z} = g$

And now we can use the chain rule in order to compute all the derivatives of the graph as follows (assuming that the derivative of f is 1):

1. $f\_\text{to}\_z = \frac{\partial f}{\partial f} \times \frac{\partial f}{\partial z} = 1 \times 11 = 11, \quad f\_\text{to}\_g = \frac{\partial f}{\partial f} \times \frac{\partial f}{\partial g} = 1 \times 4 = 4$
2. $f\_\text{to}\_h = \frac{\partial f}{\partial g} \times \frac{\partial g}{\partial h} = 4 \times 1 = 4, \quad f\_\text{to}\_i = \frac{\partial f}{\partial g} \times \frac{\partial g}{\partial i} = 4 \times 1 = 4$
3. $f\_\text{to}\_h\_\text{to}\_x = \frac{\partial f}{\partial h} \times \frac{\partial h}{\partial x} = 4 \times 1 = 4, \quad f\_\text{to}\_i\_\text{to}\_x = \frac{\partial f}{\partial i} \times \frac{\partial i}{\partial x} = 4 \times 3 = 12$
4. $f\_\text{to}\_h\_\text{to}\_y = \frac{\partial f}{\partial h} \times \frac{\partial h}{\partial y} = 4 \times 1 = 4, \quad f\_\text{to}\_i\_\text{to}\_y = \frac{\partial f}{\partial i} \times \frac{\partial i}{\partial y} = 4 \times 2 = 8$

which in turn gives us the complete computational graph:



Hope you appreciate the above computational graph as it took me a whole day to learn how the tikz library works :)

Exercise 5

I will be thrifty in this report, as most of the explanation is done inside the Colab Notebook. I have implemented 2 models:

1. A Neural Network that uses tfidf to extract features from the Dataset.
2. A Neural Network that uses the pretrained GloVe embeddings.

Both models are feedforward Neural Networks. To be honest there is nothing extremely special about them. Their architectures can be found in the Notebook. I will go briefly over the different hyperparameters that I hand-tuned, in order to see what changes took place.

NN with tfidf as input

- Hyperparameters

  1. Number of Hidden Layers

     Adding up to 3 Hidden Layers improved the models performance. After the third layer not much was changing. The Validation Loss and the Validation F1-score remained steady.

  2. Neurons in Fully Connected Layers

     Increasing the number of neurons, in this case, did not affect a lot the performance of the model. Instead it slowed down training (especially if many neurons were placed in the first hidden layer). Decreasing on the other hand (up until 64-32 in the 1st and 2nd layers) let to decrease in performance, but faster training.

  3. Activation Functions

     Let's start by stating that every model in the end uses a sigmoid activation function as we have a binary classification problem. After that, between the Linear Layers, the following activations were tried:

     – ReLU
     – SeLU
     – LeakyReLU ($\alpha = 15$)

To be honest not much difference was shown when changing activation functions. I also tried combining these functions (e.g. 1 layer with ReLU and the next with SeLU and vice versa). Small improvements were shown when using SeLU.

4. Dropout

   As expected, adding dropout (with probability $p = 0.15$) helped in generalization as it increased validation F1-score by 0.02.

5. Batch Normalization

   Batch Normalization increased a bit the performance of the model, specifically it helped in decreasing the loss down by 0.02.

6. Loss Function

   Since we are dealing with a Binary Classification problem, I used the Binary Crossentropy loss. It words good. I didn't experiment further with this, as I did not have much time.

7. Optimizer

   SGD, Adam and RMSprop were tried. Adam outperformed all of them, but it slowed down the training time a bit. I used RMSprop in the end just for the sake of difference with the second model.

8. Learning Rate

   Optimal value for learning rate was found around 5×10^{-4}. Values higher than that usually led to divergence, and values lower than that led to slow learning.

9. Weight Decay

   Optimal value for the weight decay (ridge regularization penalty, L2) was found around 10^{-4}. Higher values made the model underfit, and lower values let to gradual overfitting.

10. Batch Size

    It's hard to define an optimal value for the batch size, but by the looks of it, 32 (as Yann LeCun suggests) works out very well, though it makes training a bit slow. Also, since we are using batch normalization, having large batches introduces bias to the learning procedure, as the weights learn to rely on the normalization that takes place during training. So using a small batch size, in our case, actually helps the model generalize.

- Comparison with model from Assignment 1

  The model from Assignment 1 seems to be performing better than this one (it had 0.80 F1-score), but it was trained in the whole Dataset, rather than 1/3 which is the case here. The Logistic Regression model is a basically a shallow Neural Network, so that's why is performs ok. This model could not perform as good as the logistic Regressor, because the logistic regressor could make use of sparse matrices in order to speed up training and do more epochs. Still, both models seem to be performing nicely, but this problem can better be faced with RNNs, and there we will get the highest score.

NN with pretrained GloVe Embeddings

- Hyperparameters

  1. Number of Hidden Layers

     Again, adding up to 2 Hidden Layers improved the models performance. After the third layer, the model started to overfit a lot.

  2. Neurons in Fully Connected Layers

     Increasing the number of neurons, in general, helped improve the performance of the model. But after 512 neurons for the first layer, then training became way too slow.

  3. Activation Functions

     Pretty much the same as with the first model. Sigmoid for the last layer and in between the following activations were tested:

     – ReLU
     – SeLU
     – LeakyReLU ($\alpha = 15$)

     Again, pretty much the same results were shown.

  4. Dropout

     As expected, again, adding dropout (with probability $p = 0.3$) helped in generalizing, as it increased validation F1-score by 0.01.

5. Batch Normalization

   Batch Normalization increased a bit the performance of the model, specifically it helped in decreasing the loss again by 0.02.

6. Loss Function

   Again, Binary Crossentropy Loss was used for this Network. Next assignment I will experiment with Huber Loss, promise.

7. Optimizer

   Adam and RMSprop were tried. Adam outperformed RMSprop, but it was so slow (especially in the grid search), so I ended up using RMSprop again.

8. Learning Rate

   Optimal value for learning rate was found around 10^{-4}. Values much higher than that usually led to divergence, and values lower than that led to convergence in local minima.

9. Weight Decay

   Optimal value for the weight decay (ridge regularization penalty, L2) was found around 10^{-3}. Higher values made the model underfit, and lower values let to gradual overfitting.

10. Batch Size

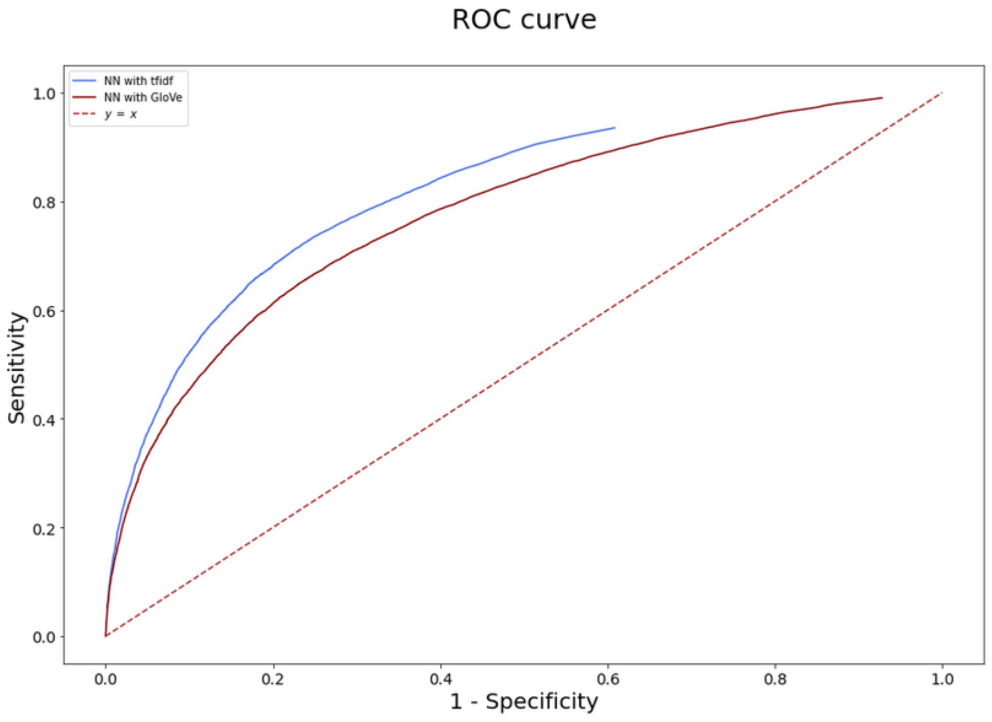
    For this Network, again a batch size of 32 was used. In the beggining I was using 512, but it was too unstable (the metrics would change drastically between each fold), so I switched back to 32.

- Comparison with model from Assignment 1

  Again, the logistic regressor from Assignment 1 performs better. This is expected, as the model in this assignment is very simple: take the word embeddings and concatenate them, then add fully connected layers. The model will not be able to interpret semantical relations, and the order of words which can make an impact in the sentiment. Of course, maybe a better implementation that mine could give WAY better results, but as suggested in Piazza, I didn't spend much time hand-tuning and searching for optimal hyperprameters to get the best possible performance.

ROC Curves

The ROC curves for both models can be seen in graph below (also inside the Notebook):



It's true that both curves should be intersecting $y = x$ at $x = 1$. They don't because the thresholds tried were from 0 up to 0.9999 with threshold step size of 0.0001. In order for them to touch the $y = x$ line, we would need to use a smaller threshold step. Still, this gives us the big picture of how both models perform. The area under a curve, is a metric of the performance of a model, and it is clear that the first model is performing better as it has a bigger area.

Best Model

Without doubt, but unexpectidely, the first model (NN with tfidf) performs better than the second one which uses GloVe word embeddings. The reasons for this were mostly explained above in the analysis of each model. But how can we conclude that the first model performs better? We have the following metrics, measured in the same test set:

1. The Loss of the first model is smaller than the loss of the second.
2. Accuracy of the first model is better than the accuracy of the second.
3. F1 score of the first model is better than the F1 score of the second.
4. The area under the ROC curve of the first model is bigger than that of the second.

There 4 metrics point us to the fact that the first model performs better.