Ονοματεπώνυμο: Παντελεήμων Μαλέκας
A.M: 1115201600268

1.

a) For the equation (4.5)

First we need to find the length of the projection of the x vector on the w vector.
This is given by $||proj_w x|| = ||x|| * \cos(x,w) = ||x|| * \frac{w*x}{||w||*||x||} = \frac{w*x}{||w||}$

The numerator is the same for $w^T x$. So we have: $\frac{wT*x}{||w||}$

Now since we have y = 0, this gives us $w^T x + w_0 = 0 \Leftrightarrow w^T x = -w_0$

So now we replace the numerator in the projection formula and we have: $\frac{-w0}{||w||}$

This gives us: $\frac{wT*x}{||w||} = \frac{-w0}{||w||}$

b) For the equation (4.6)

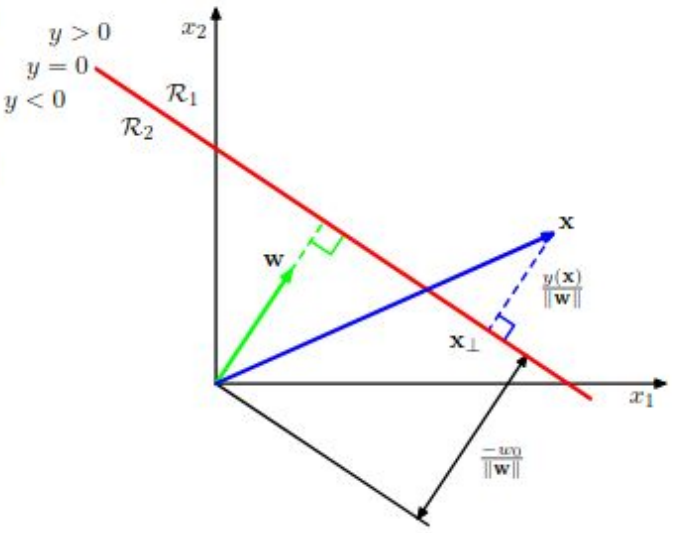
According to the theorem of Orthogonal Decomposition we have that: if **W** is a subspace of $\mathbf{R^n}$, then each vector **y** in $\mathbf{R^n}$ can be written uniquely in the form: **y** = $\hat{y}$ + **z**, where $\hat{y}$ is in **W** and **z** is in **W** ⊥ (meaning an orthogonal complement to **W**).

Also, geometrically, $\hat{y}$ is the orthogonal projection of **y** onto the subspace **W** and **z** is a vector orthogonal to $\hat{y}$ **.**

In our case we have: x = x ⊥ + r $\frac{w}{||w||}$.

First, we know that x ⊥ is the orthogonal projection of x onto the decision surface.
Second, we also know that r is equal to $\frac{y(x)}{||w||}$ as it is given in equation (4.7).
This distance is orthogonal to x ⊥ as we can see in figure 4.1

Figure 4.1 Illustration of the geometry of a linear discriminant function in two dimensions. The decision surface, shown in red, is perpendicular to $\mathbf{w}$, and its displacement from the origin is controlled by the bias parameter w_0. Also, the signed orthogonal distance of a general point $\mathbf{x}$ from the decision surface is given by $y(\mathbf{x})/\|\mathbf{w}\|$.



So, we have that x can can be written as the sum of x ⊥ and r (which is also multiplied by $\frac{w}{\|w\|}$, which doesn't affect the equation), which is what we wanted to prove.

2. The partial derivative $\frac{\partial z}{\partial x}$ of **z** = **xW** is equal to: **W**

Explanation: The dimensions of the **z** vector are 1 x m.
The i-th element of **z** is given by : $z_i = \sum_{k=1}^{n} x_k w_{ik}$
We also have the following partial derivative: $\frac{\partial zi}{\partial xj} = w_{ij}$ for every i = 1, 2, …, m and j = 1, 2, …, n.
So we have: $\frac{\partial z}{\partial x}$ = **W.**

3. The partial derivative $\frac{\partial \hat{y}}{\partial w}$ of $\hat{y}$ = σ($x^T w$) is equal to: $e^{-wx} x / (1+e^{-wx})^2$

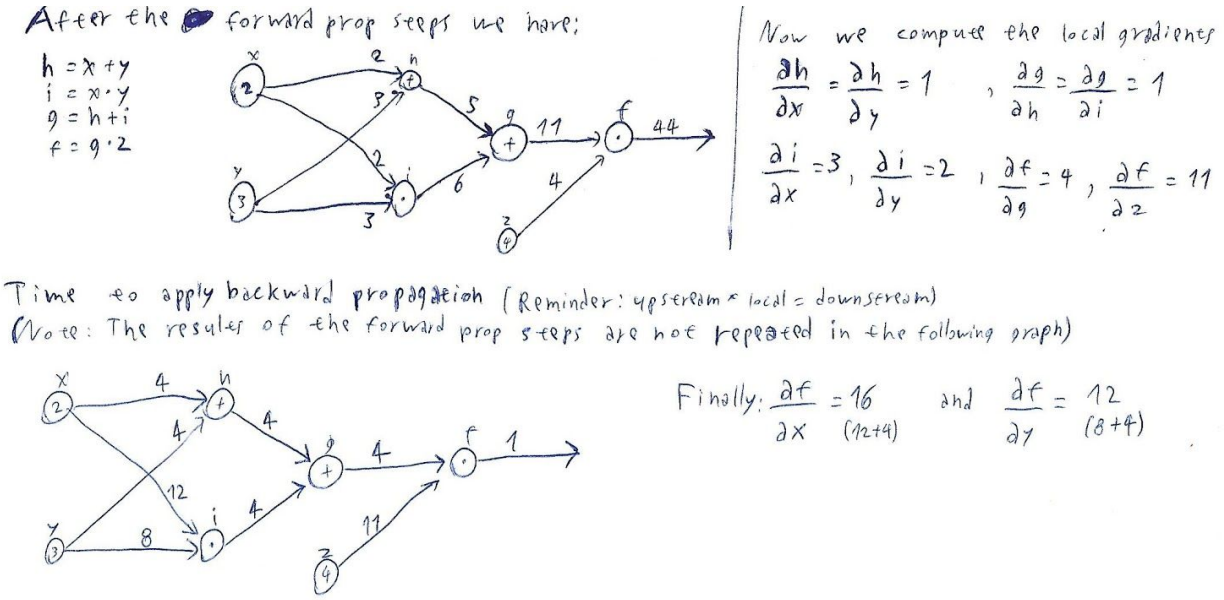
Explanation: Let f(w) = 1/g(w) = 1 / $1+e^{h(w)}$ = 1 / $1+e^{-wx}$ (which is the sigmoid function)

By the chain rule we have: f'(w) = -g'(w) / $(g(w))^2$, g'(w) = $e^{h(w)}$h'(w) and h'(w) = -x

So we have: f'(w) = - g'(w) / $(g(w))^2$ = - ($e^{h(w)}$h'(w)) / $(g(w))^2$ = $e^{-wx} x / (1+e^{-wx})^2$

(Note: The inverse symbol (T) is skipped because it can't be formatted properly in Google Docs)

4.



5. For the fifth question, three Feed Forward models were defined. A TF-IDF model, a GloVe model and an One-Hot model. Each model has different parameters.

The GloVe and One-Hot models are for the most part about the same. First, the text is pre-processed and then, stemming and lemmatization is applied. Then, the data is prepared with the use of torchtext. During the building of the vocabulary, a Twitter GloVe file is used for the GloVe model, while no additional vector is used in the One-Hot model. After preparing the data and creating batches, the model is created and trained, showing the training+test loss and the test accuracy of each epoch. After the training, the final precision, recall and F1 scores are printed.

The TF-IDF model is also the same for the most part. The main difference is that TorchText was not used in this model. Instead, the TF-IDF vectors are simply converted in PyTorch tensor format. This model was also the one chosen. More on that later.

Also, no fine tuning was performed, since it wasn't necessary.

Now, here we have a table that compares the precision, recall and F1 scores of the three models.

	TF-IDF model	GloVe model	One-Hot model
	Hidden layers: Two, with one unit each. Activation function: Tanh (both layers) Loss function: Cross Entropy Optimizer: Adam	Hidden layers: One, with one unit (with Embedding layer input) Activation function: ReLU Loss function: Cross Entropy Optimizer: Adamax	Hidden layers: One, with one unit (with Embedding layer input) Activation function: SELU Loss function: Cross Entropy Optimizer: AdamW
Precision	73.49%	77.58%	76.69%
Recall	73.46%	73.16%	76.03%
F1	73.47%	73.83%	76.11%

Regarding the choices of the parameters: Since we wanted simplistic models, I considered the use of one to two hidden layers to be good enough. One layer was used in the embedding layers because, well, one layer took enough time. The TF-IDF model was faster so I added another layer in that model.

The activation functions used were the ones presented in the lectures. I also tried using the Sigmoid function but it was discarded because it negatively affected the scores.

The Cross Entropy loss function was chosen simply because it was the only one that ‘would do’. I tried using some other loss functions such as BCE or MSE loss, but either the input was problematic or the scores were way off.

Finally, Adam and some variations of it were used as the optimizer. I also experimented with SGD, but not only did it take up too much time, the scores were off too.

Now regarding the scores, we can see the scores are more or less about the same, with the ones in the embedding models (GloVe and One-Hot) giving slightly better scores. Maybe with a few more epochs in the TF-IDF model we can get a few higher scores. It should be noted that the dataset was reduced in every model, so as a result we can't be 100% sure about the actual accuracy of the predictions. Also, the use of fine-tuning could probably increase the accuracies. Admittedly, the size of the dataset is quite huge, which causes a few problems during computation. Nevertheless, even with the subset that was used, we can see some rather good scores for the implemented models.

Finally, the TF-IDF model was chosen for the final commentary. Although the other models gave slightly better scores, the TF-IDF model could be trained faster so this gave an option to add more epochs, which as a result made the visualization of Loss vs Epochs a bit more interesting. Also, in the TF-IDF model the test accuracy actually increases with each epoch, which shows an interesting increase in the learning curve. The other models, however, already start off with a rather high accuracy in the test set.

A few comments about the plots:

In the ROC curve, we can see the expected curve for an AUC score of ~73%, which was also given from the other scores too.

In the Loss vs Epochs, we can see that the loss starts off quite high, but as the epochs progress the loss decreases. Which shows that the model actually learns better with each epoch. Interestingly, the training loss appears to be lower than the test loss after about 15 epochs. This might be a slight case of overfitting, or a different learning rate could give more consistent results.

Finally, here we have a comparison with the Logistic Regression model from HomeWork 1 (Note: 0 refers to the negative label, while 1 refers to the positive label).

	FF TF-IDF model	Logistic Regression
Precision	73.49%	76-73% (0-1)
Recall	73.46%	71-77% (0-1)
F1	73.47%	74-75% (0-1)

We can see that the scores are quite close (if not slightly better) to the ones given from the Feed Forward model. Given the fact the Feed Forward Model is quite simplistic, this should be

expected. Probably with the use of a few different hyper-parameters, we could achieve higher scores. But to get even higher scores, we'll probably need a more sophisticated model. To see a comparison of the two models, you can view the ReadMe file.

Note: It should be noted that the scores from the Logistic Regression are not the exact same from HomeWork 1. That's because the dataset could not be used in its entirety in this assignment.

Links that were used for the first question:
https://math.stackexchange.com/questions/1029153/deriving-the-normal-distance-from-the-origin-to-the-decision-surface

https://mathworld.wolfram.com/OrthogonalDecomposition.html

Links that were used for the second question:
https://atmos.washington.edu/~dennis/MatrixCalculus.pdf

Links that were used for the third question:
https://math.stackexchange.com/questions/2219891/parital-derivative-of-sigmoid-function-with-respect-to-theta

Links that were used for the fourth question:
https://eclass.uoa.gr/modules/document/file.php/DI517/%CE%94%CE%B9%CE%B1%CF%86%CE%AC%CE%BD%CE%B5%CE%B9%CE%B5%CF%82/4_cs224n-2020-lecture04-neuralnets.pdf

Links that were used for the fifth question:
https://www.deeplearningwizard.com/deep_learning/practical_pytorch/pytorch_feedforward_neuralnetwork/

https://github.com/donglinchen/text_classification/blob/master/model_pytorch.ipynb

https://github.com/bentrevett/pytorch-sentiment-analysis

https://towardsdatascience.com/deep-learning-for-nlp-with-pytorch-and-torchtext-4f92d69052f