Assignment 1

Luis Vitor Zerkowski - 14895730

November 13, 2023

4 MLP Backpropagation

Question 1 (Linear Module)

Since we are finding the closed form expressions for some derivatives in terms of $\frac{\partial L}{\partial \boldsymbol{Y}}$, considering an object of interest $\boldsymbol{O}$, we have $\frac{\partial L}{\partial \boldsymbol{O}} = \frac{\partial L}{\partial \boldsymbol{Y}} \frac{\partial \boldsymbol{Y}}{\partial \boldsymbol{O}}$ and we only need to compute $\frac{\partial \boldsymbol{Y}}{\partial \boldsymbol{O}}$. So we proceed for the computations of each one of the different objects:

1.a

We have $\frac{\partial L}{\partial \boldsymbol{W}} = \frac{\partial L}{\partial \boldsymbol{Y}} \frac{\partial \boldsymbol{Y}}{\partial \boldsymbol{W}}$ and we compute $\frac{\partial \boldsymbol{Y}}{\partial \boldsymbol{W}}$:

$$\frac{\partial \boldsymbol{Y}}{\partial \boldsymbol{W}} = \frac{\partial \left(\boldsymbol{X}\boldsymbol{W}^T + \boldsymbol{B}\right)}{\partial \boldsymbol{W}} =$$

By the sum rule:

$$= \frac{\partial \boldsymbol{X}\boldsymbol{W}^T}{\partial \boldsymbol{W}} + \frac{\partial \boldsymbol{B}}{\partial \boldsymbol{W}} \qquad \text{(I)}$$

Since $\boldsymbol{B}$ doesn't depend on $\boldsymbol{W}$, it can be treated as a constant and thus $\frac{\partial \boldsymbol{B}}{\partial \boldsymbol{W}} = 0$. Now we proceed to compute $\frac{\partial \boldsymbol{X}\boldsymbol{W}^T}{\partial \boldsymbol{W}}$ using index notation:

$$\left[\frac{\partial \boldsymbol{X}\boldsymbol{W}^T}{\partial \boldsymbol{W}}\right]_{ij} = \frac{\partial \boldsymbol{X}\boldsymbol{W}^T}{\partial W_{ij}}$$

Now we compute the above derivative for each one of the entries

$$\left[\boldsymbol{X}\boldsymbol{W}^T\right]_{pq} = \sum_{r=1}^{M} X_{pr} W^T_{rq} = \sum_{r=1}^{M} X_{pr} W_{qr} \qquad \text{(II)}$$

with $p \in \{1, ..., S\}$ and $q \in \{1, ..., N\}$. So we have:

$$\frac{\partial}{\partial W_{ij}} \sum_{r=1}^{M} X_{pr} W_{qr} =$$

By the sum rule and considering X_{pr} a constant w.r.t W_{ij}:

$$= \sum_{r=1}^{M} X_{pr} \frac{\partial W_{qr}}{\partial W_{ij}} = \sum_{r=1}^{M} X_{pr} \delta_{qi} \delta_{rj} = \delta_{qi} X_{pj}$$

So for each W_{ij}, the derivative of $\boldsymbol{X}\boldsymbol{W}^T$ is given by a matrix $\boldsymbol{D}_W \in \mathbb{R}^{S \times N}$ for which its i-th column is given by the j-th column of $\boldsymbol{X}$ and zeros anywhere else. Now we can go back to our original computation:

$$\frac{\partial L}{\partial W_{ij}} = \frac{\partial L}{\partial \boldsymbol{Y}} \frac{\partial \boldsymbol{Y}}{\partial W_{ij}} = \sum_{p,q} \frac{\partial L}{\partial Y_{pq}} \frac{\partial Y_{pq}}{\partial W_{ij}} =$$

$$= \sum_{p,q} \frac{\partial L}{\partial Y_{pq}} \delta_{qi} X_{pj} = \sum_{p} \frac{\partial L}{\partial Y_{pi}} X_{pj}$$

And for the matrix closed form solution we have:

$$\frac{\partial L}{\partial \boldsymbol{W}} = \left(\frac{\partial L}{\partial \boldsymbol{Y}}\right)^T \boldsymbol{X} \in \mathbb{R}^{N \times M}$$

1.b

We have $\frac{\partial L}{\partial \boldsymbol{b}} = \frac{\partial L}{\partial \boldsymbol{Y}} \frac{\partial \boldsymbol{Y}}{\partial \boldsymbol{b}}$ and we compute $\frac{\partial \boldsymbol{Y}}{\partial \boldsymbol{b}}$. Since the operations are pretty much the same up until equation (I) from **1.a**, we start from it:

$$\frac{\partial \boldsymbol{Y}}{\partial \boldsymbol{b}} = \frac{\partial \boldsymbol{X}\boldsymbol{W}^T}{\partial \boldsymbol{b}} + \frac{\partial \boldsymbol{B}}{\partial \boldsymbol{b}}$$

Since $\boldsymbol{X}\boldsymbol{W}^T$ doesn't depend on $\boldsymbol{b}$, we treat it as a constant and proceed to compute $\frac{\partial \boldsymbol{B}}{\partial \boldsymbol{b}}$:

$$\left[\frac{\partial \boldsymbol{B}}{\partial \boldsymbol{b}}\right]_i = \frac{\partial \boldsymbol{B}}{\partial b_i}$$

Computing the derivative for each one of the entries we have:

$$\frac{\partial B_{pq}}{\partial b_i} = \frac{\partial b_q}{\partial b_i} = \delta_{qi}$$

So for each b_i, the derivative of $\boldsymbol{B}$ is given by a matrix $\boldsymbol{D}_b \in \mathbb{R}^{S \times N}$ for which the i-th column is a column of ones and the rest of the matrix' entries are zero. Now we can go back to our original computation:

$$\frac{\partial L}{\partial b_i} = \frac{\partial L}{\partial \boldsymbol{Y}} \frac{\partial \boldsymbol{Y}}{\partial b_i} = \sum_{p,q} \frac{\partial L}{\partial Y_{pq}} \frac{\partial Y_{pq}}{\partial b_i} =$$

$$= \sum_{p,q} \frac{\partial L}{\partial Y_{pq}} \delta_{qi} = \sum_{p} \frac{\partial L}{\partial Y_{pi}}$$

Now for the matrix closed form solution we make use of a row vector $\mathbf{1}^T \in \mathbb{R}^{1 \times S}$ for which all the entries are ones and we have:

$$\frac{\partial L}{\partial \boldsymbol{b}} = \mathbf{1}^T \frac{\partial L}{\partial \boldsymbol{Y}} \in \mathbb{R}^{1 \times N}$$

1.c

We have $\frac{\partial L}{\partial \boldsymbol{X}} = \frac{\partial L}{\partial \boldsymbol{Y}} \frac{\partial \boldsymbol{Y}}{\partial \boldsymbol{X}}$ and we compute $\frac{\partial \boldsymbol{Y}}{\partial \boldsymbol{X}}$. Since the operations are pretty much the same up until equation (II) from **1.a**, we start from it:

$$\frac{\partial}{\partial X_{ij}} \sum_{r=1}^{M} X_{pr} W_{qr} =$$

By the sum rule and considering W_{qr} a constant w.r.t X_{ij}:

$$= \sum_{r=1}^{M} \frac{\partial X_{pr}}{\partial X_{ij}} W_{qr} = \sum_{r=1}^{M} \delta_{pi} \delta_{rj} W_{qr} = \delta_{pi} W_{qj}$$

So for each X_{ij}, the derivative of $\boldsymbol{X}\boldsymbol{W}^T$ is given by a matrix $\boldsymbol{D}_X \in \mathbb{R}^{S \times N}$ for which its i-th row is given by the transpose of the j-th column of $\boldsymbol{W}$ and zeros anywhere else. Now we can go back to our original computation:

$$\frac{\partial L}{\partial X_{ij}} = \frac{\partial L}{\partial \boldsymbol{Y}} \frac{\partial \boldsymbol{Y}}{\partial X_{ij}} = \sum_{p,q} \frac{\partial L}{\partial Y_{pq}} \frac{\partial Y_{pq}}{\partial X_{ij}} =$$

$$= \sum_{p,q} \frac{\partial L}{\partial Y_{pq}} \delta_{pi} W_{qj} = \sum_{q} \frac{\partial L}{\partial Y_{iq}} W_{qj}$$

And for the matrix closed form solution we have:

$$\frac{\partial L}{\partial \boldsymbol{X}} = \frac{\partial L}{\partial \boldsymbol{Y}} \boldsymbol{W} \in \mathbb{R}^{S \times M}$$

Question 1 (Activation Module)

1.d

Analogously to the previous exercises, we have $\frac{\partial L}{\partial \boldsymbol{X}} = \frac{\partial L}{\partial \boldsymbol{Y}} \frac{\partial \boldsymbol{Y}}{\partial \boldsymbol{X}}$ and we compute $\frac{\partial \boldsymbol{Y}}{\partial \boldsymbol{X}}$. So we start by computing the derivative of Y_{pq} w.r.t X_{ij}:

$$\frac{\partial Y_{pq}}{\partial X_{ij}} = \frac{\partial h\left(X_{pq}\right)}{\partial X_{ij}} =$$

By the chain rule:

$$= \frac{\partial h\left(X_{pq}\right)}{\partial X_{pq}} \frac{\partial X_{pq}}{\partial X_{ij}} = h'\left(X_{pq}\right) \delta_{pi} \delta_{qj}$$

Now we can go back to our original computation:

$$\frac{\partial L}{\partial X_{ij}} = \frac{\partial L}{\partial \boldsymbol{Y}} \frac{\partial \boldsymbol{Y}}{\partial X_{ij}} = \sum_{p,q} \frac{\partial L}{\partial Y_{pq}} \frac{\partial Y_{pq}}{\partial X_{ij}} =$$

$$= \sum_{p,q} \frac{\partial L}{\partial Y_{pq}} h'\left(X_{pq}\right) \delta_{pi} \delta_{qj} = \frac{\partial L}{\partial Y_{ij}} h'\left(X_{ij}\right)$$

And for the matrix closed form solution we have:

$$\frac{\partial L}{\partial \boldsymbol{X}} = \frac{\partial L}{\partial \boldsymbol{Y}} \circ h'\left(\boldsymbol{X}\right)$$

With $h'\left(\boldsymbol{X}\right)$ denoting the derivative of the activation function h evaluated element-wise on $\boldsymbol{X}$.

Question 2 (Softmax and Loss Modules)

2.a

We start by expanding the expression:

$$\frac{\partial L}{\partial \boldsymbol{Z}} = \boldsymbol{Y} \circ \left(\frac{\partial L}{\partial \boldsymbol{Y}} - \left(\frac{\partial L}{\partial \boldsymbol{Y}} \circ \boldsymbol{Y} \right) \mathbf{1}\mathbf{1}^T \right) = \boldsymbol{Y} \circ \frac{\partial L}{\partial \boldsymbol{Y}} - \boldsymbol{Y} \circ \left(\frac{\partial L}{\partial \boldsymbol{Y}} \circ \boldsymbol{Y} \right) \mathbf{1}\mathbf{1}^T =$$

Now using $\frac{\partial L}{\partial \boldsymbol{Y}} = -\frac{1}{S} \frac{\boldsymbol{T}}{\boldsymbol{Y}}$, we have:

$$= -\frac{1}{S} \left(\boldsymbol{Y} \circ \frac{\boldsymbol{T}}{\boldsymbol{Y}} \right) + \frac{1}{S} \left(\boldsymbol{Y} \circ \left(\frac{\boldsymbol{T}}{\boldsymbol{Y}} \circ \boldsymbol{Y} \right) \mathbf{1}\mathbf{1}^T \right) = \frac{1}{S} \left(\left(\boldsymbol{Y} \circ \boldsymbol{T} \right) \mathbf{1}\mathbf{1}^T - \boldsymbol{T} \right)$$

So we have $\alpha = \frac{1}{S} \in \mathbb{R}^+$ and $\boldsymbol{M} = \left(\left(\boldsymbol{Y} \circ \boldsymbol{T} \right) \mathbf{1}\mathbf{1}^T - \boldsymbol{T} \right) \in \mathbb{R}^{S \times C}$.

Question 2 (Residuals)

2.b

By using a residual connection, we add a constraint to N_2. Since we are summing the matrix $\boldsymbol{X}$ to the output of the module, we need to make sure that the shape of the outputs of the neurons in the second linear layer match the shape of $\boldsymbol{X}$, hence $\mathbb{R}^{S \times F}$. For that to be true, we need to have $N_2 = F$, the number of output features on the second linear layer.

As for the first linear layer, since the size of the samples S is preserved throughout the whole module, regardless of the number of output features on the first linear layer, we don't need to add any constraints to N_1.

2.c

By adding the residual connection, we are also adding a factor to the backward flow of the network. $\frac{\partial L}{\partial \boldsymbol{X}}$ thus is now influenced by both the flow coming from the output features $\frac{\partial L}{\partial \boldsymbol{Y}}$ and the flow coming directly from the residual connection. For computing this last term coming, because of the sum rule for derivatives, we can focus only on the addition of the residual connection:

$$\left[\frac{\partial \boldsymbol{X}}{\partial \boldsymbol{X}}\right]_{ij} = \frac{\partial \boldsymbol{X}}{\partial X_{ij}} \implies \frac{\partial X_{pq}}{\partial X_{ij}} = \delta_{pi}\delta_{qj}$$

Going back to the original computation but only focusing on the addition by the residual connection:

$$\sum_{p,q} \frac{\partial L}{\partial P_{pq}} \frac{\partial X_{pq}}{\partial X_{ij}} = \sum_{p,q} \frac{\partial L}{\partial P_{pq}} \delta_{pi}\delta_{qj} = \frac{\partial L}{\partial P_{ij}}$$

So the value of $\frac{\partial L}{\partial \boldsymbol{X}}$ is changed by an addition factor of $\frac{\partial L}{\partial \boldsymbol{P}}$.

2.d

One of the main ideas of this type of connections is trying to help with the vanishing gradient problem, a huge problem when training networks. By adding another term to the gradients, we make sure that their values stay higher and hence numerically more stable, enabling the network to properly propagate the gradient throughout the whole architecture and thus effectively updating the parameters for learning.

Question 3 (Numpy Neural Network)

Figure 1 shows the loss curve for the training process on the training data for each batch on each epoch. Figure 2, in turn, show the accuracy on the validation dataset after each epoch. It's important to notice that all these results were obtained by testing the network on the default parameters, as requested.

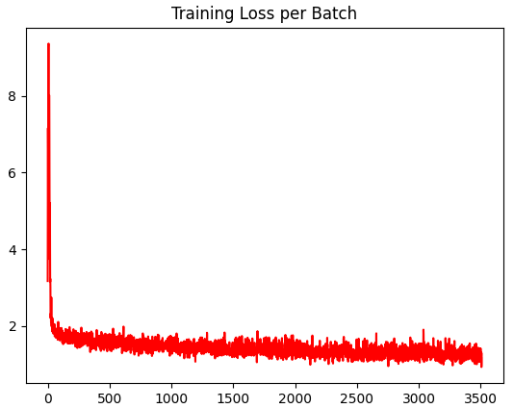

Figure 1: Training loss for each batch on each epoch.

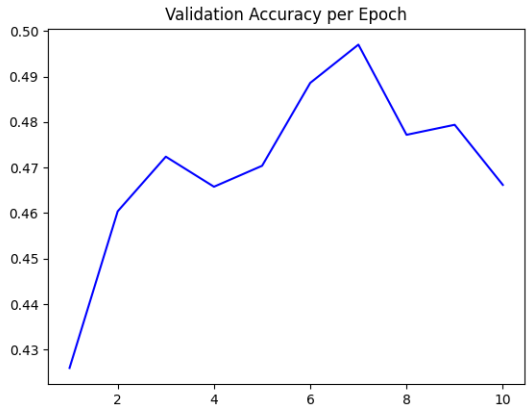

Figure 2: Validation accuracy after each epoch.

For the best model, chosen in terms of the validation accuracy, the test accuracy was 49.41%.

5 Pytorch MLP

Question 4 (Pytorch Neural Network)

Figure 3 shows the loss curve for the training process on the training data for each batch on each epoch. Figure 4, in turn, show the accuracy on the validation dataset after each epoch. It's important

to notice that all these results were obtained by testing the network on the default parameters, as requested.

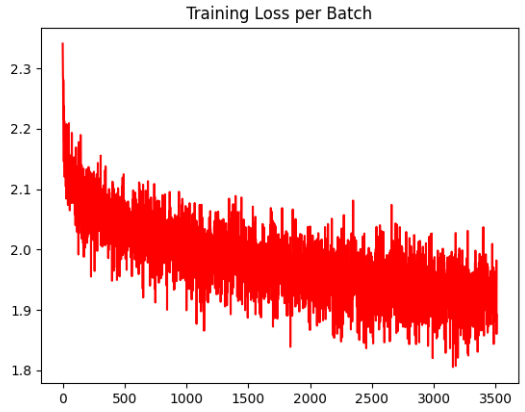

Figure 3: Training loss for each batch on each epoch.

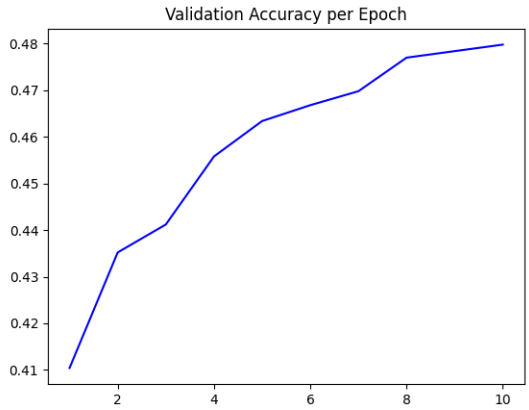

Figure 4: Validation accuracy after each epoch.

For the best model, chosen in terms of the validation accuracy, the test accuracy was 47.85%.

Comparing the results obtained using `Pytorch` with the results obtained using `Numpy` on our toy problem, we can easily observe some similarities and some differences. Both models, given the same default parameters, reach very similar numerical metrics: training loss going up and down around $[1.8, 2.0]$, best validation accuracies around $47\% - 49\%$ and test accuracies for the best model around $47\% - 49\%$. Even though we got similar final results, it's also pretty clear from the images that the training proces using `Pytorch` is much more stable, so we don't see the training loss going really high on the first batches of the first epoch before going down, and neither the validation accuracy going up and down a lot, but rather an always increasing curve.

Question 5 (Optimization)

5.a

Considering f a continously differentiable invertible function, we start by approximating it using the second order Taylor expansion. For that, let's take a generic local minimum point $\boldsymbol{p}$ and let's try to approximate the function on a generic point $\boldsymbol{x}$ around it. So we have:

$$f(\boldsymbol{x}) \approx f(\boldsymbol{p}) + \nabla f(\boldsymbol{p})(\boldsymbol{x} - \boldsymbol{p}) + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{p})^T H_f(\boldsymbol{p})(\boldsymbol{x} - \boldsymbol{p})$$

But since we are working with a local minimum, we know that $\nabla f(\boldsymbol{p}) = \boldsymbol{0}$, which leads to:

$$f(\boldsymbol{x}) \approx f(\boldsymbol{p}) + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{p})^T H_f(\boldsymbol{p})(\boldsymbol{x} - \boldsymbol{p})$$

Since we know more specifically that we are dealing with a strictly local minimum, any point $\boldsymbol{x}$ on the neighborhood of $\boldsymbol{p}$ will lead to $f(\boldsymbol{x}) > f(\boldsymbol{p})$. We then have:

$$\begin{aligned} &\frac{1}{2}(\boldsymbol{x} - \boldsymbol{p})^T H_f(\boldsymbol{p})(\boldsymbol{x} - \boldsymbol{p}) \approx f(\boldsymbol{x}) - f(\boldsymbol{p}) \implies \\ &\implies \frac{1}{2}(\boldsymbol{x} - \boldsymbol{p})^T H_f(\boldsymbol{p})(\boldsymbol{x} - \boldsymbol{p}) > 0 \implies \\ &\implies (\boldsymbol{x} - \boldsymbol{p})^T H_f(\boldsymbol{p})(\boldsymbol{x} - \boldsymbol{p}) > 0 \end{aligned}$$

Since $\boldsymbol{x}$ is a generic point around the strictly local minimum, we can simply rewrite $\boldsymbol{x} - \boldsymbol{p}$ as a new point $\boldsymbol{v}$, leading to $\boldsymbol{v}^T H_f(\boldsymbol{p})\boldsymbol{v} > 0$ and thus indicating that the Hessian in $\boldsymbol{p}$ is a positive definite

matrix. From this point on, we just need to prove that the eigenvalues of a positive definite matrix are all positive. So let's take a generic eigenvector $\boldsymbol{u}$ of the Hessian with the eigenvalue λ and by the definition of eigenvectors and eigenvalues we have:

$$H_f(\boldsymbol{p})\boldsymbol{u} = \lambda\boldsymbol{u} \implies \boldsymbol{u}^T H_f(\boldsymbol{p})\boldsymbol{u} = \boldsymbol{u}^T\lambda\boldsymbol{u}$$

But since λ is a scalar, we can simply do $\boldsymbol{u}^T\lambda\boldsymbol{u} = \lambda\boldsymbol{u}^T\boldsymbol{u}$ to get:

$$\implies \boldsymbol{u}^T H_f(\boldsymbol{p})\boldsymbol{u} = \lambda\boldsymbol{u}^T\boldsymbol{u}$$

Finally, we know that $\boldsymbol{u}^T\boldsymbol{u} > 0$ since the inner product will be given by the sum of squares of the components of the vector and $\boldsymbol{u}$ cannot be the zero vector. We also know, from our previous computations, that $H_f(\boldsymbol{p})$ is positive definite, which means lambda is given by the division of two positive numbers. This leads to $\lambda = \frac{\boldsymbol{u}^T H_f(\boldsymbol{p})\boldsymbol{u}}{\boldsymbol{u}^T\boldsymbol{u}} > 0$ and hence it is shown that all the eigenvalues of the Hessian in a strictly local minimum are positive.

5.b

Intuitively a saddle point can be characterized by a point for which some features have local maximum and some features have local minimum. If all the features had maximums, $\boldsymbol{p}$ would be a local maximum and conversely if all the features had minimums $\boldsymbol{p}$ would be a local minimum. So considering a generic critical point, in which all the features have maximums or minimums for a point, it is quite natural to understand that maximums and minimums are much more rare than saddle points, since they require all the features to agree, either all maximums or all minimums.

Putting a number to it, imagine a critical point $\boldsymbol{p}$ on a D dimensional space. Since we are dealing with a critical point, we know that all features have either local maximum or local minimum values on $\boldsymbol{p}$. The point is a maximum with probability $\frac{\text{All maximum}}{\text{All combinations}} = \frac{1}{2^D}$ and a minimum with the same probability $\frac{\text{All minimum}}{\text{All combinations}} = \frac{1}{2^D}$. The probability of $\boldsymbol{p}$ being a saddle point thus is given by $1 - P(\text{Maximum}) - P(\text{Minimum}) = 1 - \frac{1}{2^{D-1}}$, exponentially higher than the probabilities of $\boldsymbol{p}$ being a local maximum or local minimum for high values of D.

There's is also a fun fact if you think about the probabilities. When $D = 1$, the probability of $\boldsymbol{p}$ being a maximum is $\frac{1}{2}$ and being a minimum is also $\frac{1}{2}$. This indicates that $\boldsymbol{p}$ is a saddle point with probability zero, which makes total sense because we don't have saddle points for data with only one feature - we can think of it as the feature always agreeing with itself on a maximum or a minimum.

5.c

The update formula of gradient descent for a specific point (SGD) is given by: $\boldsymbol{w} = \boldsymbol{w} - \eta\nabla f(\boldsymbol{p})$. But we know that a saddle point is a critical point for the function, which means $\nabla f(\boldsymbol{p}) = \boldsymbol{0}$. This leads to $\boldsymbol{w} = \boldsymbol{w} - \boldsymbol{0}$, so no update on the weights getting our model stuck on the saddle point.

This situation is not ideal because we know that on a saddle point there exists at least one feature for which that point represents a local maximum, which means we still have plenty of space for optimization. But since the gradient on that point is given by the zero vector, we never really update the weights.

Question 6 (F1 Score)

6.a

1. **Precision:** It's used to indicate the percentage of positive predictions that are actually right - how sure your model is about positive predictions. This metric is taken into consideration when you don't want to make wrongful predictions for the positive class, so you want to make sure that when you eventually predicts positive, the data really belongs to the positive class.

   An example of application for this metric would be biometrically identifying if a person can enter a secure building. In this case, you want to be sure that only authorized people can enter the building (TP), and you really don't want unauthorized people to be able to enter (FP). If authorized people get blocked sometimes (FN), it doesn't matter that much because they can simply call for help and they will manually check the person's credentials.

2. **Recall:** It's use to indicate the percentage of positive predictions over all the positive samples - how much of the positive class your model can capture. This metric is taken into consideration when you want to make sure you don't miss any positive examples, so it's fine for the model to predict false positives as long as it doesn't predict many false negatives.

   A classic example for which this metric can be used is the cancer detection one. If you have a model that predicts whether a patient has cancer or not, you want to make sure it doesn't miss any patients, so don't generate FNs. It is not ideal, but definitely less impactful if it predicts that a patient has cancer when the patient is actually healthy, so you don't bother generating a little bit more of FPs.

   Another example could be predicting the necessity of maintenance for an airplanes. Given some features of an airplane, you want to be sure an airplane that needs maintenance is classified accordingly (TP), and you really don't want broken airplanes to be flying (FN). If an airplane didn't need maintenance and you still gave it maintenance (FP), it doesn't matter that much.

3. **Accuracy:** It's a more generic metric that captures how much the model correctly predicts labels over all the samples. It's mostly used when there's no strong constraint on the classes, so errors for false positive or false negative are equally bad and the goal is just to find a model that performs well in general.

   Good examples for this metric are more arbitrary, so predicting whether an image is a picture of a dog or not, or predicting if a house will eventually be sold given its price, size and location.

6.b

Figure 5 shows the confusion matrix for CIFAR10 on the test set using the best model obtained - according to validation accuracy - of `Numpy` version MLP with default parameters.

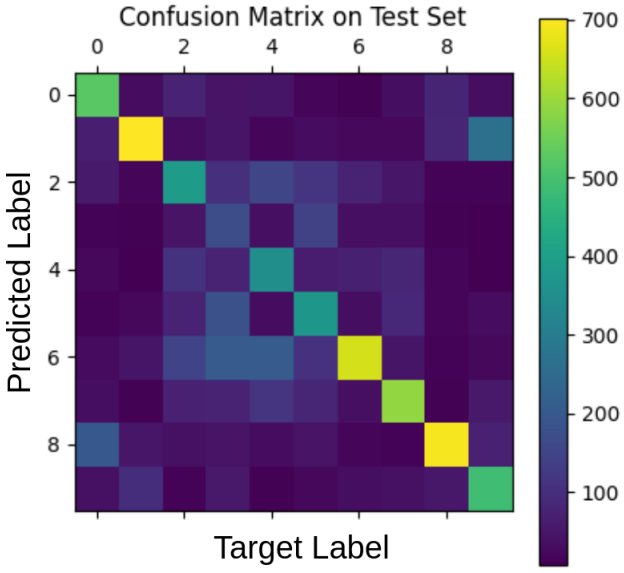

Figure 5: Confusion matrix for CIFAR10 on test set.

6.c

Based on the definition of F_β:

1. When beta is small, $\beta = 0.1$, we have $1 + \beta^2 \approx 1$ and $\beta^2\text{precision} \approx 0$, which means $F_\beta \approx \frac{\text{precision} \cdot \text{recall}}{\text{recall}} = \text{precision}$.
2. Conversely when beta is large, $\beta = 10$, we have $1 + \beta^2 \approx \beta^2$ and $\beta^2\text{precision} >> \text{recall}$, which means $F_\beta \approx \beta^2 \frac{\text{precision} \cdot \text{recall}}{\beta^2 \text{precision}} = \text{recall}$.

3. When has an intermediate value, $\beta = 1$, we have a balancing of precision and recall.

	Class 0	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Class 9
$\beta = 0.1$	**0.587**	0.547	**0.394**	**0.335**	**0.424**	**0.429**	0.444	0.547	0.578	**0.561**
$\beta = 1$	0.543	0.609	0.381	0.223	0.374	0.387	0.520	0.557	0.623	0.510
$\beta = 10$	0.524	**0.702**	0.385	0.177	0.347	0.372	**0.649**	**0.588**	**0.696**	0.483

Table 1: F_β scores for all the classes and all β values.

Analyzing table 1 we have the F_β scores for all the classes and all β values. We can observe that the largest value for every class is always either $\beta = 0.1$ (precision) or $\beta = 10$ (recall). This happens because $\beta = 1$ is a balancing metric in which we study the trade-off, hence one of the two metrics is going to push the value up and the other one push the value down. Naturally this balancing cannot be larger than the highest between precision and recall and neither smaller than the lowest between precision and recall.

So for better precision we have classes 0, 2, 3, 4, 5 and 9. For better recall, in turn, we have classes 1, 6, 7 and 8. This does not mean that these classes have high values for precision or recall. For the high values, we have only class 1 and class 8 with high recalls.