Montag, Mai 19, 2025

Machine Learning 1, Summer Term 2025
Homework 1
Linear and Logistic Regression. Gradient Descent

Florian Scherer (12232933)
Linda Weiss (12108205)

Group Number: **g234**

1. **Neural Networks**
   1.1. PCA and Classification
      1.1.1. PCA for dimensionality reduction:

We reduced the dimensionality of the images using Principal Component Analysis (PCA), from originally 4096 features (64×64 pixels) down to 128 principal components. This was done to lower the computational cost for further processing and classification. While some information was inevitably lost in the process, the reduction preserves a substantial portion of the original data variance.
Below is the explained variance ratio for the 128 components:

```
0.16991706 0.044101   0.02962424 0.02682274 0.02540656 0.02113097
0.01558895 0.01162894 0.01053646 0.01002552 0.00896351 0.00861704
0.00825115 0.00783183 0.00686022 0.00662703 0.00646524 0.00567873
0.00550447 0.00535055 0.00525278 0.00502491 0.0048957  0.00475512
0.00473872 0.00442663 0.00427543 0.00415443 0.00410627 0.00383421
0.00371381 0.00370212 0.00356338 0.00345434 0.0033411  0.0032258
0.00318531 0.00308367 0.0030031  0.00293571 0.00285468 0.0027824
0.00273499 0.00264394 0.00258478 0.00256952 0.00253122 0.00249728
0.00242292 0.00240947 0.0023447  0.00232764 0.00229756 0.00227071
0.00218905 0.00214714 0.00209658 0.0020755  0.00204274 0.00203064
0.00200034 0.00198065 0.00194665 0.00192235 0.00191358 0.00188776
0.00187045 0.00182697 0.00181251 0.00177695 0.00175999 0.0017149
0.00169586 0.00168596 0.00167373 0.00165799 0.00163876 0.00162591
0.00160299 0.00157934 0.00157204 0.00156302 0.00153686 0.00150923
0.00150045 0.0014917  0.00148048 0.00146624 0.001453   0.00143867
0.00141202 0.00139908 0.00138966 0.00135919 0.00135531 0.00135458
0.00133131 0.0013166  0.00131402 0.00129704 0.00128422 0.00126928
0.00126022 0.00125416 0.001243   0.0012294  0.00121754 0.0012069
0.00119475 0.00118916 0.00118097 0.00117497 0.00115762 0.00115249
0.00113832 0.00113279 0.00111883 0.00111556 0.00111422 0.00110776
0.00109065 0.00107714 0.00107466 0.00106729 0.0010447  0.00103606
0.0010282  0.0010121
```

The cumulative explained variance is approximately 65.77 %, meaning that 128 components retain most of the essential structure of the data. Although not lossless, this trade-off is considered acceptable for the classification tasks ahead.

1.1.2. Varying the number of hidden neurons / layers

After reducing the input dimensionality using PCA, we trained several neural network models with different hidden layer sizes. The goal was to find the architecture that generalizes best to unseen data. We evaluated each model based on two criteria: First, validation accuracy, as the primary measure of generalization performance. Second, the gap between training and validation accuracy, to assess the degree of overfitting.
We selected the model that achieved the highest validation accuracy while maintaining a reasonable balance between learning performance and generalization. We additionally allowed a small margin of tolerance when comparing models: a new model could be selected even if it overfits slightly more than the current best one, as long as it achieved higher validation accuracy overall.

Following all possible layer models:
Hidden layer: (2,), Train accuracy: 0.6250, Validation accuracy: 0.5188
Training loss: 0.8973
Hidden layer: (8,), Train accuracy: 0.8430, Validation accuracy: 0.7094
Training loss: 0.4779
Hidden layer: (64,), Train accuracy: 0.9938, Validation accuracy: 0.7469
Training loss: 0.0953
Hidden layer: (128,), Train accuracy: 0.9992, Validation accuracy: 0.7562
Training loss: 0.0381
Hidden layer: (256,), Train accuracy: 0.9992, Validation accuracy: 0.7656
Training loss: 0.0169
Hidden layer: (1024,), Train accuracy: 1.0000, Validation accuracy: 0.7562
Training loss: 0.0045
Hidden layer: (128, 256, 128), Train accuracy: 1.0000, Validation accuracy: 0.7312
Training loss: 0.0041

Chosen model: Train accuracy: 0.9992, Validation accuracy: 0.7656
Training loss: 0.0169

1.1.3. Recognising Overfitting and Underfitting

Identifying overfitting and underfitting requires some experience and intuition, as there are no strict thresholds. The only way to detect them is by interpreting the model's accuracy in relation to the specific context.
Overfitting can typically be recognized when the training accuracy is significantly higher than the validation or test accuracy. In our case, we observed overfitting in the networks with hidden layer configurations of (64,), (128,), (256,), (1024,), and (128, 256, 128). Some of them, like (128, 256, 128), showed a gap of ~27 %, while others, like (256,), showed ~23 %. Although the latter still overfits, it achieved the best validation performance, so it was chosen despite the gap.

Underfitting, in contrast, occurs when the model is unable to learn from the training data properly. This results in both low training and validation accuracies. The model with (2,) hidden units is a good example, achieving only ~63 % training accuracy and ~52 % validation accuracy. Even though the gap between training and validation (~11 %) is smaller than the overfitting models mentioned above, its overall performance was worse.
A milder case of underfitting was seen with the (8,) model. It had a gap of ~14 %, which seems acceptable, but the resulting validation accuracy was still lower than that of the slightly overfitting (256,) model.
In summary, a model that overfits slightly may still outperform an underfitting one in terms of actual classification performance.

Despite the slight overfitting, the model with (256,) was preferred because it achieved the highest validation accuracy overall. This indicates that slight overfitting is acceptable when it leads to better generalization performance than underfitting or weaker models.

1.1.4. Overfitting

To prevent overfitting, the MLPClassifier in scikit-learn provides two built-in strategies: regularization and early stopping.

Regularization limits the size of the weights inside the neural network. This encourages the model to focus on learning the patterns in the data rather than memorizing individual training examples. → It reduces model complexity for generalization.

Early stopping, on the other hand, monitors the validation performance during training. Once the model stops improving on the validation set for several iterations, training is halted automatically. This prevents the model from continuing to learn and helps avoid overfitting – while still giving it enough time to learn meaningful patterns (prevent Underfitting).

```
alpha = 0.1
Hidden layer: (2,), Train accuracy: 0.6273, Validation accuracy: 0.5188
Training loss: 0.9011
Hidden layer: (8,), Train accuracy: 0.8414, Validation accuracy: 0.7031
Training loss: 0.4917
Hidden layer: (64,), Train accuracy: 0.9922, Validation accuracy: 0.7438
Training loss: 0.1589
Hidden layer: (128,), Train accuracy: 0.9984, Validation accuracy: 0.7438
Training loss: 0.1146
Hidden layer: (256,), Train accuracy: 0.9984, Validation accuracy: 0.7719
Training loss: 0.0987
Hidden layer: (1024,), Train accuracy: 1.0000, Validation accuracy: 0.7812
Training loss: 0.0763
Hidden layer: (128, 256, 128), Train accuracy: 1.0000, Validation accuracy:
0.7438
Training loss: 0.0777
```

```
early_stopping = True
Hidden layer: (2,), Train accuracy: 0.4523, Validation accuracy: 0.4031
Training loss: 1.1224
Hidden layer: (8,), Train accuracy: 0.7695, Validation accuracy: 0.6594
Training loss: 0.6293
Hidden layer: (64,), Train accuracy: 0.8055, Validation accuracy: 0.6844
Training loss: 0.4790
Hidden layer: (128,), Train accuracy: 0.9141, Validation accuracy: 0.7375
Training loss: 0.1895
Hidden layer: (256,), Train accuracy: 0.9203, Validation accuracy: 0.7188
Training loss: 0.1711
Hidden layer: (1024,), Train accuracy: 0.8352, Validation accuracy: 0.7469
Training loss: 0.1988
Hidden layer: (128, 256, 128), Train accuracy: 0.9414, Validation accuracy: 0.7281
Training loss: 0.0201
```

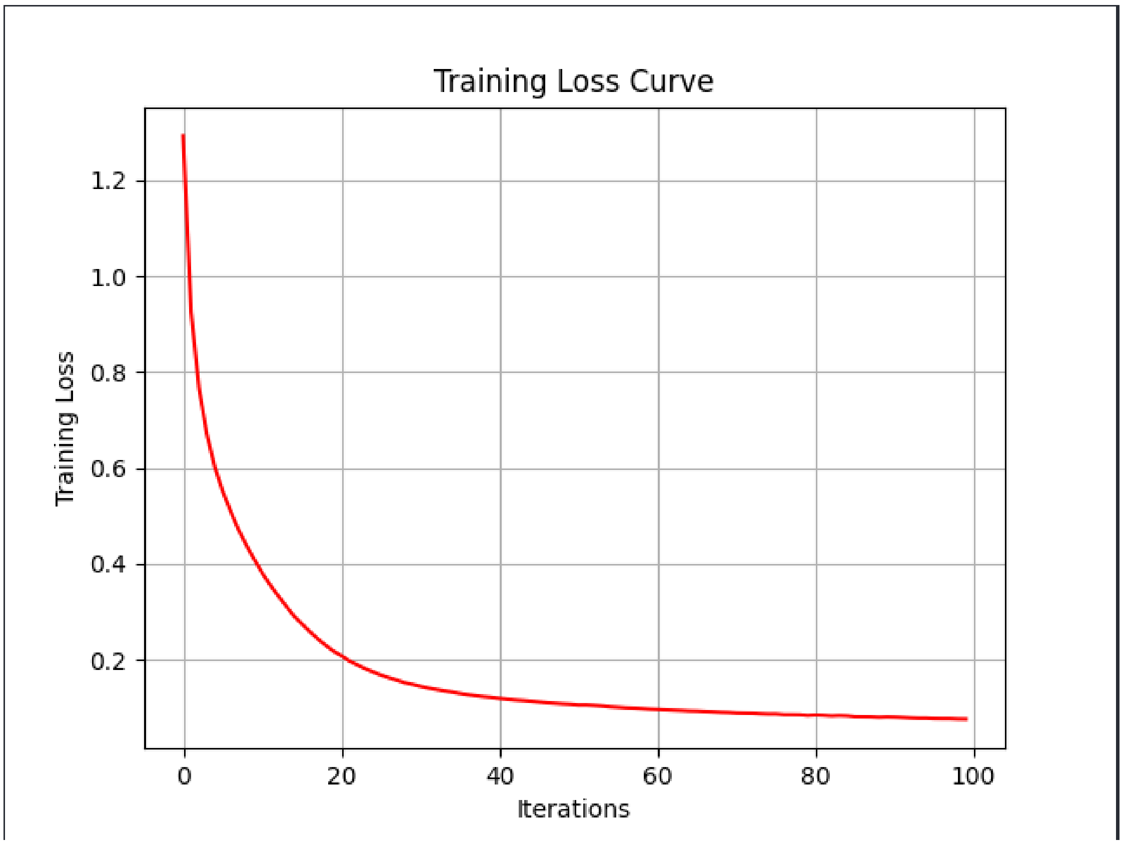
```
alpha = 0.1, early_stopping = True
Hidden layer: (2,), Train accuracy: 0.4398, Validation accuracy: 0.3937
Training loss: 1.1430
Hidden layer: (8,), Train accuracy: 0.7672, Validation accuracy: 0.6625
Training loss: 0.6385
Hidden layer: (64,), Train accuracy: 0.8102, Validation accuracy: 0.6813
Training loss: 0.4980
Hidden layer: (128,), Train accuracy: 0.9102, Validation accuracy: 0.7406
Training loss: 0.2516
Hidden layer: (256,), Train accuracy: 0.9187, Validation accuracy: 0.7219
Training loss: 0.2393
Hidden layer: (1024,), Train accuracy: 0.8344, Validation accuracy: 0.7469
Training loss: 0.2797
Hidden layer: (128, 256, 128), Train accuracy: 0.9219, Validation accuracy: 0.7281
Training loss: 0.1548
```

```
Chosen model: Train: 1.0000, Validation: 0.7812, Loss: 0.0763,
Best model case: a
```

Yes, we did choose another model, as we found a model which has both better validation accuracy and a better gap between training and validation accuracy which we achieved by introducing a weight threshold, regularization, of alpha = 0.1.
Our new model is now better at predicting the validation data and has a smaller gap, naming less overfitting.

1.1.5. Plotting loss curve

We received the hint that MLPClassifier stores some training information in its attributes.
Upon inspection, we found that the training loss history over iterations is the only such information available.
Therefore, we chose to plot only this loss curve, as no validation or accuracy curves are tracked by the classifier.We've gotten the hind that the MLPClassifier saves some information about the training in his attributes. And as the training loss history across iterations is the only thing it holds, we only printed this.



1.2. Model selection and Evalution Metrics

1.2.1. How many different architectures will be checked?

In total there are 60 different “fits” that will be checked.

These 60 are computed by:

- the number of combinations of the values inside the dictionary
  alpha1 with batch_size1 and hidden_layer_size1 or
  alpha1 with batch_size1 and hidden_layer_size2 and so on
- times the number of cross-validations (folds) per combination of 1.

1.2.2. Only code

1.2.3. What was the best parameter set that was found in this grid search?

The best parameter set was alpha = 1.0, batch_size = 512 and hidden_layer_size = (256,). It got a mean cross-validation score of 0.7637 and an explained variance ratio of:

```
[0.16991706 0.044101   0.02962424 0.02682274 0.02540656 0.02113097
 0.01558895 0.01162893 0.01053645 0.0100255  0.0089634  0.00861681
 0.00825089 0.00783065 0.00685771 0.00662568]
```

DATA 1.2.1 – 1.2.3:

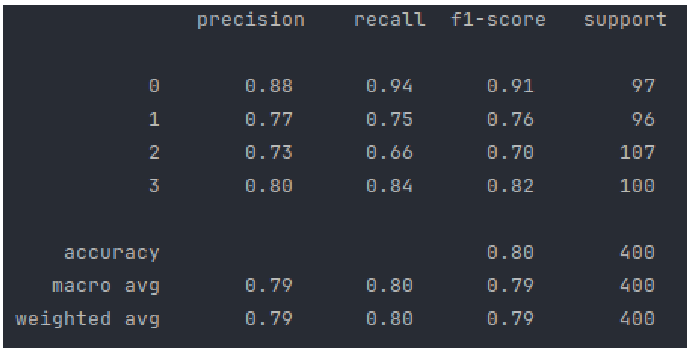
```
Fitting 5 folds for each of 12 candidates, totalling 60 fits
[CV 1/5] END alpha=0.0, batch_size=32, hidden_layer_sizes=(128,);, score=0.772 total time=   4.1s
[CV 2/5] END alpha=0.0, batch_size=32, hidden_layer_sizes=(128,);, score=0.778 total time=   4.1s
[CV 3/5] END alpha=0.0, batch_size=32, hidden_layer_sizes=(128,);, score=0.719 total time=   4.2s
[CV 4/5] END alpha=0.0, batch_size=32, hidden_layer_sizes=(128,);, score=0.784 total time=   3.5s
[CV 5/5] END alpha=0.0, batch_size=32, hidden_layer_sizes=(128,);, score=0.734 total time=   4.0s
[CV 1/5] END alpha=0.0, batch_size=32, hidden_layer_sizes=(256,);, score=0.750 total time=   6.8s
[CV 2/5] END alpha=0.0, batch_size=32, hidden_layer_sizes=(256,);, score=0.775 total time=   6.9s
[CV 3/5] END alpha=0.0, batch_size=32, hidden_layer_sizes=(256,);, score=0.725 total time=   6.2s
[CV 4/5] END alpha=0.0, batch_size=32, hidden_layer_sizes=(256,);, score=0.787 total time=   6.0s
[CV 5/5] END alpha=0.0, batch_size=32, hidden_layer_sizes=(256,);, score=0.722 total time=   6.1s
[CV 1/5] END alpha=0.0, batch_size=512, hidden_layer_sizes=(128,);, score=0.762 total time=   2.0s
[CV 2/5] END alpha=0.0, batch_size=512, hidden_layer_sizes=(128,);, score=0.775 total time=   0.8s
[CV 3/5] END alpha=0.0, batch_size=512, hidden_layer_sizes=(128,);, score=0.725 total time=   0.8s
[CV 4/5] END alpha=0.0, batch_size=512, hidden_layer_sizes=(128,);, score=0.781 total time=   1.3s
[CV 5/5] END alpha=0.0, batch_size=512, hidden_layer_sizes=(128,);, score=0.722 total time=   0.9s
[CV 1/5] END alpha=0.0, batch_size=512, hidden_layer_sizes=(256,);, score=0.759 total time=   1.5s
[CV 2/5] END alpha=0.0, batch_size=512, hidden_layer_sizes=(256,);, score=0.759 total time=   1.7s
[CV 3/5] END alpha=0.0, batch_size=512, hidden_layer_sizes=(256,);, score=0.731 total time=   1.6s
[CV 4/5] END alpha=0.0, batch_size=512, hidden_layer_sizes=(256,);, score=0.791 total time=   1.4s
[CV 5/5] END alpha=0.0, batch_size=512, hidden_layer_sizes=(256,);, score=0.716 total time=   1.4s
[CV 1/5] END alpha=0.1, batch_size=32, hidden_layer_sizes=(128,);, score=0.778 total time=   5.2s
[CV 2/5] END alpha=0.1, batch_size=32, hidden_layer_sizes=(128,);, score=0.772 total time=   5.3s
[CV 3/5] END alpha=0.1, batch_size=32, hidden_layer_sizes=(128,);, score=0.738 total time=   4.1s
[CV 4/5] END alpha=0.1, batch_size=32, hidden_layer_sizes=(128,);, score=0.784 total time=   5.0s
[CV 5/5] END alpha=0.1, batch_size=32, hidden_layer_sizes=(128,);, score=0.741 total time=   4.2s
[CV 1/5] END alpha=0.1, batch_size=32, hidden_layer_sizes=(256,);, score=0.766 total time=   5.6s
[CV 2/5] END alpha=0.1, batch_size=32, hidden_layer_sizes=(256,);, score=0.762 total time=   6.2s
[CV 3/5] END alpha=0.1, batch_size=32, hidden_layer_sizes=(256,);, score=0.738 total time=   5.6s
[CV 4/5] END alpha=0.1, batch_size=32, hidden_layer_sizes=(256,);, score=0.794 total time=   5.8s
[CV 5/5] END alpha=0.1, batch_size=32, hidden_layer_sizes=(256,);, score=0.747 total time=   5.4s
[CV 1/5] END alpha=0.1, batch_size=512, hidden_layer_sizes=(128,);, score=0.759 total time=   0.9s
[CV 2/5] END alpha=0.1, batch_size=512, hidden_layer_sizes=(128,);, score=0.769 total time=   0.8s
[CV 3/5] END alpha=0.1, batch_size=512, hidden_layer_sizes=(128,);, score=0.731 total time=   0.9s
[CV 4/5] END alpha=0.1, batch_size=512, hidden_layer_sizes=(128,);, score=0.784 total time=   1.0s
[CV 5/5] END alpha=0.1, batch_size=512, hidden_layer_sizes=(128,);, score=0.722 total time=   0.9s
[CV 1/5] END alpha=0.1, batch_size=512, hidden_layer_sizes=(256,);, score=0.772 total time=   1.7s
[CV 2/5] END alpha=0.1, batch_size=512, hidden_layer_sizes=(256,);, score=0.759 total time=   1.3s
[CV 3/5] END alpha=0.1, batch_size=512, hidden_layer_sizes=(256,);, score=0.728 total time=   1.8s
```

```
[CV 4/5] END alpha=0.1, batch_size=512, hidden_layer_sizes=(256,);, score=0.797 total time=   1.7s
[CV 5/5] END alpha=0.1, batch_size=512, hidden_layer_sizes=(256,);, score=0.716 total time=   2.5s
[CV 1/5] END alpha=1.0, batch_size=32, hidden_layer_sizes=(128,);, score=0.734 total time=   5.2s
[CV 2/5] END alpha=1.0, batch_size=32, hidden_layer_sizes=(128,);, score=0.719 total time=   4.0s
[CV 3/5] END alpha=1.0, batch_size=32, hidden_layer_sizes=(128,);, score=0.709 total time=   4.0s
[CV 4/5] END alpha=1.0, batch_size=32, hidden_layer_sizes=(128,);, score=0.769 total time=   3.2s
[CV 5/5] END alpha=1.0, batch_size=32, hidden_layer_sizes=(128,);, score=0.700 total time=   4.0s
[CV 1/5] END alpha=1.0, batch_size=32, hidden_layer_sizes=(256,);, score=0.728 total time=  10.2s
[CV 2/5] END alpha=1.0, batch_size=32, hidden_layer_sizes=(256,);, score=0.725 total time=   8.8s
[CV 3/5] END alpha=1.0, batch_size=32, hidden_layer_sizes=(256,);, score=0.703 total time=   6.2s
[CV 4/5] END alpha=1.0, batch_size=32, hidden_layer_sizes=(256,);, score=0.772 total time=   5.4s
[CV 5/5] END alpha=1.0, batch_size=32, hidden_layer_sizes=(256,);, score=0.731 total time=   9.7s
[CV 1/5] END alpha=1.0, batch_size=512, hidden_layer_sizes=(128,);, score=0.762 total time=   1.2s
[CV 2/5] END alpha=1.0, batch_size=512, hidden_layer_sizes=(128,);, score=0.766 total time=   0.9s
[CV 3/5] END alpha=1.0, batch_size=512, hidden_layer_sizes=(128,);, score=0.734 total time=   1.3s
[CV 4/5] END alpha=1.0, batch_size=512, hidden_layer_sizes=(128,);, score=0.806 total time=   1.4s
[CV 5/5] END alpha=1.0, batch_size=512, hidden_layer_sizes=(128,);, score=0.719 total time=   0.9s
[CV 1/5] END alpha=1.0, batch_size=512, hidden_layer_sizes=(256,);, score=0.772 total time=   1.5s
[CV 2/5] END alpha=1.0, batch_size=512, hidden_layer_sizes=(256,);, score=0.775 total time=   1.6s
[CV 3/5] END alpha=1.0, batch_size=512, hidden_layer_sizes=(256,);, score=0.728 total time=   4.2s
[CV 4/5] END alpha=1.0, batch_size=512, hidden_layer_sizes=(256,);, score=0.812 total time=   1.6s
[CV 5/5] END alpha=1.0, batch_size=512, hidden_layer_sizes=(256,);, score=0.731 total time=   1.4s
```

1.2.4. Reporting and choosing a model

We’ve chosen the model provided by GridSearchCV, even though the model we’ve found in 1.1.4 has a better validation accuracy (1.2: ~0.76 vs 1.1.4: ~0.78). This is because the model of 1.1.4 achieved this score in a **single** only split whereas the model of GridSearchCV got a **mean** score. A high score in a single split could therefore just be luck with the train and validation set. Whereas a mean score shows a good learning across various set combinations resulting in a stronger model.

With this we’ve achieved a final accuracy of 0.7950 on the test set!
With the information from the class report, we can see how the model performed for each class:



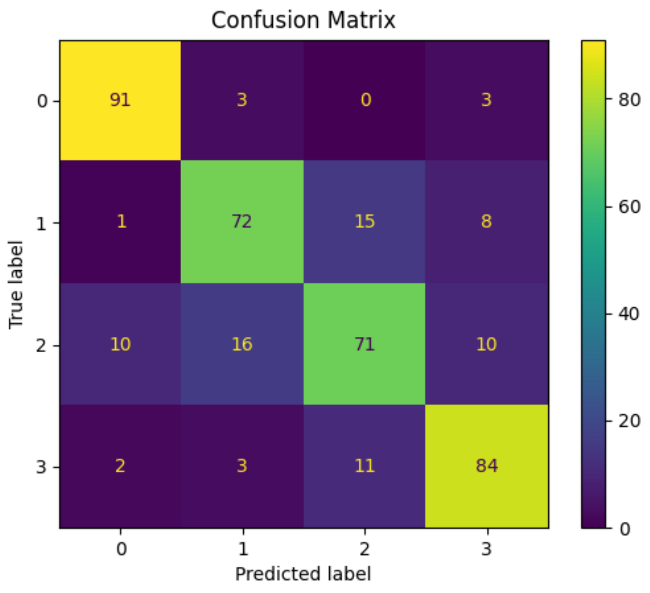
```
              precision    recall  f1-score   support

           0       0.88      0.94      0.91        97
           1       0.77      0.75      0.76        96
           2       0.73      0.66      0.70       107
           3       0.80      0.84      0.82       100

    accuracy                           0.80       400
   macro avg       0.79      0.80      0.79       400
weighted avg       0.79      0.80      0.79       400
```

This tells us for example, that only 73% of the predictions of class 2 were correct (**precision**), 66% of the pictures that are really in class 2 were correctly recognized (**recall**) and got a mean of precision and recall (**f1-score**) of 70% and that there are 107 real class 2 pictures (**support**).
The **accuracy** tells us how many pictures we predicted correct overall, **macro avg** tells us the average of all classes while all classes are worth the same and **weighted avg** tells us the average weighted by the number of pictures by class (for example: class 2 got the most pictures therefore it is weighted more in the average).

To visualize this data, we've computed and plotted a confusion matrix. It shows us what class a picture should be (True label / y-axis), what class our model predicted it in (Predicted label / x-axis) and how often it was predicted in a specific class comparing to the true class (Numbers inside the "boxes"). These numbers also show us, that the pictures of the True Label class 2 were miss-predicted the most (16 times) as class 1. Closely followed by the pictures of True Label class 1 as class 2 (15 times). The colors are a heat map of the accuracy for visualization purposes.



1.2.5. Recall vs. precision – misclassified ranking winner

Simply said, **precision** measures how often the model's *predictions were correct* (i.e., how many predicted labels were actually true), whereas **recall** measures how many *actual instances of a class were correctly identified*.
A more detailed explanation is given in the answer to 1.2.4.

The class most often misclassified by our model was **class 2** (see 1.2.4).

1.2.6. Difference between hyperparameters and parameters of a model

Hyperparameters are the parameters *we're able to control as a user.* Those parameters that we give the model before it starts and that change the acting and the structure of the model. Examples are the number of hidden layers, the batch size, the number of iterations…

Parameters of the model are the pendant to the Hyperparameters. These are the parameters that the model *learns by itself*. It represents the knowledge of the model. Examples are weights and biases of the neurons.

2. Neural Networks from Scratch

2.1. Only code
2.2. Only code
2.3. Only code
2.4. Only code

2.5. <u>Own MLP Classifier Accuracy</u>

After training, the final accuracy scores for our own MLP classifier are as follows:

• Train acc:	0.7289
• Test acc:	0.6875
• Validation acc:	0.6725

2.6. <u>Is L2 regularization worth it?</u>

To check if L2 regularization of our network is worth it in this case, we set alpha to a none zero value and check the train, test and validation accuracies again.

Chosen alpha values:

- **α = 0.005**

o Train acc:	0.7344
o Test acc:	0.6906
o Validation acc:	0.6725

Train and test accuracy slightly increased with slightly bigger alpha, and validation accuracy stayed the same.

- **α = 0.1**

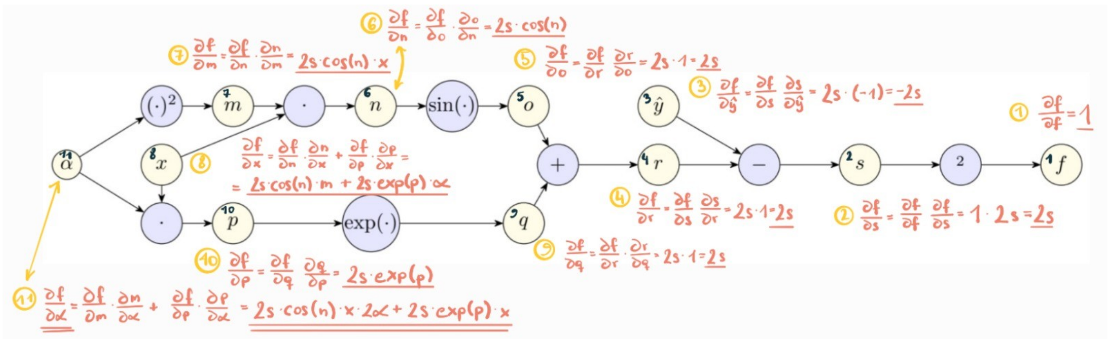
o Train acc:	0.7422
o Test acc:	0.7000
o Validation acc:	0.6800

All accuracy values slightly increased with increased alpha.

Both chosen values of alpha led to improved test accuracy, with α = 0.1 yielding the best overall results. This suggests that L2 regularization is useful in this case, helping the model perform slightly better across all datasets.

2.7. Answers

a)


⑥ $\frac{\partial f}{\partial n} = \frac{\partial f}{\partial o} \cdot \frac{\partial o}{\partial n} = \underline{2s \cdot \cos(n)}$
⑦ $\frac{\partial f}{\partial m} = \frac{\partial f}{\partial n} \cdot \frac{\partial n}{\partial m} = \underline{2s \cdot \cos(n) \cdot x}$
⑤ $\frac{\partial f}{\partial o} = \frac{\partial f}{\partial r} \frac{\partial r}{\partial o} = 2s \cdot 1 = \underline{2s}$
③ $\frac{\partial f}{\partial \hat{y}} = \frac{\partial f}{\partial s} \frac{\partial s}{\partial \hat{y}} = 2s \cdot (-1) = \underline{-2s}$
① $\frac{\partial f}{\partial f} = \underline{1}$
$(\cdot)^2$
m
$\cdot$
n
$\sin(\cdot)$
o
$\hat{y}$
α
x
⑧ $\frac{\partial f}{\partial x} = \frac{\partial f}{\partial n} \cdot \frac{\partial n}{\partial x} + \frac{\partial f}{\partial p} \cdot \frac{\partial p}{\partial x} = \underline{2s \cdot \cos(n) \cdot m + 2s \cdot \exp(p) \cdot \alpha}$
$+$
r
$-$
s
2
f
p
$\exp(\cdot)$
q
④ $\frac{\partial f}{\partial r} = \frac{\partial f}{\partial s} \frac{\partial s}{\partial r} = 2s \cdot 1 = \underline{2s}$
② $\frac{\partial f}{\partial s} = \frac{\partial f}{\partial f} \frac{\partial f}{\partial s} = 1 \cdot 2s = \underline{2s}$
⑩ $\frac{\partial f}{\partial p} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial p} = \underline{2s \cdot \exp(p)}$
⑨ $\frac{\partial f}{\partial q} = \frac{\partial f}{\partial r} \cdot \frac{\partial r}{\partial q} = 2s \cdot 1 = \underline{2s}$
⑪ $\underline{\underline{\frac{\partial f}{\partial \alpha}}} = \frac{\partial f}{\partial m} \cdot \frac{\partial m}{\partial \alpha} + \frac{\partial f}{\partial p} \cdot \frac{\partial p}{\partial \alpha} = \underline{\underline{2s \cdot \cos(n) \cdot x \cdot 2\alpha + 2s \cdot \exp(p) \cdot x}}$

b) The activation function in the hidden layers must be nonlinear because a neural network made of only linear activation functions is equivalent to just a single linear unit. That means we couldn’t take advantage of having multiple layers. A nonlinear activation makes it possible for the network to learn more complex patterns.

c) When we add more layers to a neural network, it can become harder to train. This was a common problem in older networks, which is why deep neural networks were not used much in the past. Up until 2006, it was said that neural networks with more than two layers could not be trained reliably.

d) L2 regularization adjusts the weights by making them smaller, which leads to simpler functions. This can make the model less likely to overfit the training data.

The answers to these questions are based on the lecture slides from Lecture05 and Lecture07.

3. Binary Classification

3.1. Implement sigmoid and binary_cross_entropy_loss

sigmoid():

implemented the sigmoid function by using the formular

sigma(z) = 1 / 1 + e^(-z)

This returns us a scalar with it's value between 0 and 1.

binary_cross_entropy_loss():

implented the bce_loss function using the formular

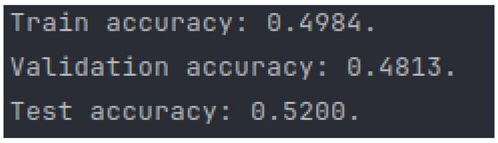
bce = -[y_true * log(prop) + (1 – y_true) * log(1 – prob)]

which measures the error between prob and the true value y_true and returns the loss.

3.2. Training on binary data

We trained our own MLP model for binary classification using the provided setup: PCA with 16 components, a single hidden layer with 16 neurons, and 5 training epochs.

The resulting model clearly underfits the data:



```
Train accuracy: 0.4984.
Validation accuracy: 0.4813.
Test accuracy: 0.5200.
```

This performance is close to random guessing and suggests that the model lacks to learn patterns.

3.3. Theory Question

No, the model is basically just lucky guessing, as he tried to make a binary distribution (by allowing only 2 classes) but their chances aren't 50:50. As he split it up in A = {0} and B = {1,2,3} the distribution is 25:75 in reality, which means, that if the model guest guesses “Class B” every time, it is right 75% of the time but didn't learn anything. Meaning that just the accuracy can be pretty misleading.

In general, I would prefer keeping the original 4-class problem.