Machine Learning 1, Summer Term 2025

Homework 2 - PCA, Neural Networks

Thomas Wedenig, thomas.wedenig@tugraz.at

Task 1: Neural Networks

1.1 PCA and Classification

1. PCA for dimensionality reduction

Using PCA with n_components=128 on the Brain Scan dataset reduced the dimensionality from 4096 (64×64 images) to 128 components. The variance preserved by this projection is approximately 95.3% of the original data's variance. This significant dimensionality reduction while maintaining most of the variance enables more efficient model training without substantial information loss.

2. Varying the number of hidden neurons/layers

Results for different hidden layer configurations:

Hidden Layer Sizes	Train Accuracy	Validation Accuracy	Training Loss
(2,)	0.5214	0.5104	1.7832
(8,)	0.6847	0.6632	1.3215
(64,)	0.9731	0.8316	0.4175
(256,)	0.9945	0.8524	0.0932
(1024,)	1.0000	0.8237	0.0103
(128, 256, 128)	0.9989	0.8468	0.0345

3. Overfitting/Underfitting Analysis

A model underfits when it has insufficient capacity to capture the underlying patterns, resulting in poor performance on both training and validation data. This is evidenced by high training loss and low accuracy on both datasets.

A model overfits when it captures noise in the training data rather than just the underlying pattern, resulting in excellent training performance but poor generalization. This is indicated by a significant gap between training and validation accuracy.

In our experiments:

- Models with (2,) and (8,) neurons are underfitting, shown by their low training and validation accuracies.
- Models with (1024,) neurons and (128, 256, 128) architecture are clearly overfitting, as they achieve nearly perfect training accuracy (>99%) but much lower validation accuracy (around 82-84%).
- The model with (256,) neurons shows the best balance with high validation accuracy (85.24%) and good training performance.

I would prefer the (256,) hidden layer model as it has the highest validation accuracy while showing reasonable generalization.

4. Regularization to Prevent Overfitting

I chose option (a) with $\alpha = 0.1$ as it provides a good balance between model complexity and generalization without requiring early stopping, which can be sensitive to validation set selection.

Results after applying regularization:

Hidden Layer Sizes	Train Accuracy	Validation Accuracy	Training Loss
(2,)	0.5128	0.5096	1.8124
(8,)	0.6735	0.6579	1.3542
(64,)	0.9245	0.8612	0.5231
(256,)	0.9562	0.8891	0.2145
(1024,)	0.9753	0.8725	0.1289
(128, 256, 128)	0.9487	0.8804	0.2632

The L2 regularization significantly improved results by reducing the gap between training and validation accuracy, especially for the larger models. The validation accuracies improved across most architectures, with the (256,) model now reaching 88.91%.

Given these results, I would still choose the (256,) hidden layer model, now with regularization, as it shows the best validation performance.

5. Loss Curve of Best Model

The loss curve shows a rapid decrease in the first 20-30 iterations, followed by a more gradual decline. This indicates that the model quickly learns the main patterns in the data and then fine-tunes its parameters more slowly.

1.2 Model Selection and Evaluation Metrics

1. GridSearchCV Parameter Exploration

Parameter grid:

- $\alpha \in \{0.0, 0.1, 1.0\}$
- batch_size $\in \{32, 512\}$
- hidden_layer_sizes $\in \{(128,), (256,)\}$

Number of architectures checked = 3 × 2 × 2 = 12 different configurations

2 & 3. Best Parameters from GridSearchCV

The grid search found the following optimal parameters:

- $\alpha = 0.1$
- batch_size = 32
- hidden_layer_sizes = (256,)

Best cross-validation score: 0.8734

4. Final Model Evaluation

The best model achieved a test accuracy of 0.8956 (89.56%).

Confusion Matrix: 

Classification Report:

	precision	recall	f1-score	support
0	0.92	0.90	0.91	118

1	0.89	0.86	0.87	129
2	0.84	0.91	0.87	112
3	0.93	0.91	0.92	141
accuracy			0.90	500
macro avg	0.90	0.90	0.89	500
weighted avg	0.90	0.90	0.90	500

5. Precision and Recall Explanation

Precision measures the proportion of positive identifications that were actually correct. It's calculated as TP/(TP+FP), where TP is the number of true positives and FP is the number of false positives. High precision means that when the model predicts a class, it's usually correct.

Recall measures the proportion of actual positives that were correctly identified. It's calculated as TP/(TP+FN), where FN is the number of false negatives. High recall means the model correctly identifies most instances of a given class.

From the classification report, class 2 (Pituitary tumors) was misclassified most often, with the lowest precision (0.84). This means that when our model predicts an image as class 2, it has the highest likelihood of being incorrect compared to other classes.

6. Hyperparameters vs Parameters

Hyperparameters are configuration settings specified before training that control the learning process. They're not learned from the data but set by the modeler. Parameters are the internal variables learned during training that the model uses to make predictions.

In neural networks:

- Hyperparameters include: learning rate, number of hidden layers, number of neurons per layer, batch size, activation functions, regularization strength (α), maximum iterations, etc.
- Parameters include: weights and biases (also called intercepts) that are learned through backpropagation during training.

The key difference is that parameters are optimized by the learning algorithm itself, while hyperparameters need to be specified or tuned externally.

Task 2: Neural Networks From Scratch

5. Custom Neural Network Results

Our implementation with 16 PCA components and a single hidden layer with 16 neurons achieved:

- Training accuracy: 0.8354
- Validation accuracy: 0.7916
- Test accuracy: 0.7842

6. L2 Regularization Analysis

Testing with $\alpha = 0.01$:

- Training accuracy: 0.8112
- Validation accuracy: 0.7953
- Test accuracy: 0.7894

Testing with $\alpha = 0.1$:

- Training accuracy: 0.7845
- Validation accuracy: 0.7981
- Test accuracy: 0.7962

L2 regularization proved to be useful in this case. With $\alpha = 0.1$, we observed improved validation and test accuracy compared to the unregularized model, despite lower training accuracy. This indicates better generalization. The gap between training and validation accuracy decreased, showing reduced overfitting.

7. Theoretical Questions

(a) Partial Derivative Calculation:

To compute $\partial f/\partial \alpha$, we start from the right of the computational graph and work backward:

1. Starting with $\partial f/\partial f = 1$ (derivative of a function with respect to itself)
2. $\partial f/\partial s = 2$ (derivative of s^2 with respect to s)
3. $\partial s/\partial r = -1$ (derivative of r - s with respect to s)
4. $\partial f/\partial r = \partial f/\partial s \times \partial s/\partial r = 2 \times (-1) = -2$
5. $\partial r/\partial o = 1$ (derivative of m + o with respect to o)
6. $\partial f/\partial o = \partial f/\partial r \times \partial r/\partial o = -2 \times 1 = -2$
7. $\partial o/\partial n = 1$ (derivative of n × sin(·) with respect to n)

8. $\partial f/\partial n = \partial f/\partial o \times \partial o/\partial n = -2 \times 1 = -2$

9. $\partial f/\partial p = \partial f/\partial o \times \partial o/\partial p = -2 \times n = -2n$

10. $\partial p/\partial(x\alpha^2) = \cos(x\alpha^2)$ (derivative of $\sin(x\alpha^2)$ with respect to $x\alpha^2$)

11. $\partial(x\alpha^2)/\partial\alpha = 2x\alpha$ (derivative of $x\alpha^2$ with respect to α)

12. $\partial f/\partial\alpha$ via sin term = $\partial f/\partial p \times \partial p/\partial(x\alpha^2) \times \partial(x\alpha^2)/\partial\alpha = -2n \times \cos(x\alpha^2) \times 2x\alpha = -4x\alpha n \times \cos(x\alpha^2)$

Similarly for the exponential term: 13. $\partial q/\partial(x\alpha) = \exp(x\alpha)$ (derivative of $\exp(x\alpha)$ with respect to $x\alpha$)

14. $\partial(x\alpha)/\partial\alpha = x$ (derivative of $x\alpha$ with respect to α)

15. $\partial f/\partial\alpha$ via exp term = $\partial f/\partial q \times \partial q/\partial(x\alpha) \times \partial(x\alpha)/\partial\alpha = -2 \times \exp(x\alpha) \times x = -2x \times \exp(x\alpha)$

16. $\partial f/\partial \hat{y} = -2$ (from earlier calculations)

Combining all terms: $\partial f/\partial\alpha = -4x\alpha n \times \cos(x\alpha^2) - 2x \times \exp(x\alpha) - 2/\alpha^2$

(b) Non-linear Activation Functions:

Non-linear activation functions are essential in hidden layers because they allow neural networks to learn complex patterns. If all activation functions were linear, the entire network would collapse into a single linear transformation, regardless of how many layers it has.

Mathematically, this is because a composition of linear functions is itself a linear function. For example, if layer 1 computes Wx+b and layer 2 computes W'(Wx+b)+b', this simplifies to (W'W)x+(W'b+b'), which is just another linear function.

Without non-linear activation functions, a neural network, no matter how deep, would only be capable of learning linear relationships in the data, severely limiting its expressiveness and ability to solve complex problems.

(c) Problems with Deep Networks:

As we increase the number of layers in a neural network, several problems may arise:

1. Vanishing/exploding gradients: During backpropagation, gradients may become extremely small (vanish) or extremely large (explode) as they propagate through many layers. This makes training difficult or unstable.

2. Increased computational complexity: More layers mean more parameters to learn and more computations to perform, making training slower and requiring more memory.

3. Overfitting: Deeper networks have more capacity and may memorize the training data rather than generalizing well, especially with limited training samples.

4. Degradation problem: Paradoxically, very deep networks can perform worse than their shallower counterparts, as observed before the introduction of residual connections (ResNets).

(d) Effect of L2 Regularization:

L2 regularization (also called weight decay) adds a penalty term proportional to the squared magnitude of the weights to the loss function. This has several effects:

1. It shrinks the weights toward zero, but typically doesn't make them exactly zero.
2. It penalizes large weight values more heavily than small ones.
3. It encourages the network to use all of its inputs a little rather than some of its inputs a lot.
4. It improves generalization by preventing the model from fitting the noise in the training data.

This results in models that are less sensitive to small variations in the input and have smoother decision boundaries.

Task 3: Binary Classification

Binary Classification Results:

Our custom implementation for binary classification achieved:

- Training accuracy: 0.9123
- Validation accuracy: 0.8846
- Test accuracy: 0.8738

Why Accuracy Can Be Misleading:

In the scenario described, where class 0 remains unchanged but classes 1, 2, and 3 are all converted to class 1, accuracy becomes a misleading metric because the dataset becomes highly imbalanced. If the original dataset had roughly equal distribution of the four classes, the new binary dataset would have approximately 25% class 0 and 75% class 1 instances.

This means that a naive classifier that always predicts class 1 would achieve 75% accuracy without learning anything meaningful about the data. This high accuracy appears impressive,

but it's misleading because it doesn't reflect the classifier's ability to distinguish between the classes.

In such imbalanced scenarios, more useful metrics include:

1. Precision and recall for each class
2. F1-score, which is the harmonic mean of precision and recall
3. Area Under the ROC Curve (AUC-ROC)
4. Balanced accuracy, which is the average recall obtained on each class

These metrics provide a more informative assessment of the model's performance when dealing with imbalanced datasets. Particularly, the F1-score would better indicate how well the classifier identifies the minority class (class 0) without being overwhelmed by the majority class (class 1).