Assignment 5 - Exercise 2 - Text Classification with Convolutional Neural Networks

Contributor: Dimitris Vougioukos - f3352411

The exercise 2 focuses on building a CNN classifier for sentiment analysis on an existing dataset and comparing its performance with baseline classifiers.

The following chapters will outline the steps taken and the design decisions made to achieve the desired results.

Exercise 2

At the beginning of the exercise, we chose to download and use the `movie_reviews` dataset from the `NLTK` library, which contains various movie reviews. We examined its structure and identified two file types, `neg` (negative review) and `pos` (positive review), representing the distinct classes under which the reviews are categorized. Specifically, we observed that the dataset contains 1.000 reviews categorized under the `neg` class and another 1.000 reviews under the `pos` class. This defines a binary classification problem.

After that, we extracted the raw text of each review along with its corresponding class/label and stored them into two separate lists for further processing. During the preprocessing phase, we first applied the `sent_tokenize` function from the `NLTK` library to segment the reviews into sentences. Then, we used the `word_tokenize` function from the same library to tokenize the sentences. We filtered out stopwords, lemmatized the remaining tokens to their base forms, and combined them to create the final processed reviews. We observed that the average length of the reviews was 342 tokens.

Next, we used the `train_test_split` function from the `sklearn` library to divide the dataset into three subsets: training, validation, and test sets, with the following sizes.

Subset	Size - Number of Reviews
Training	1120
Validation	480
Test	400

The final average length of the training reviews was 341 tokens, which is used to maintain a specific length for all reviews that pass through the neural network, as we will see below.

The next step involved constructing a vocabulary of the 10.000 most frequent tokens from the training dataset, along with two additional tokens for padding (PAD) and unknown (UNK), each of which was assigned a specific index. Using this vocabulary, we then created the corresponding embedding matrix based on the pre-trained `word2vec-google-news-300` model. This table serves as a mapping between the tokens and their embeddings for the neural network. The padding token was assigned a zero vector, while the unknown token, as well as any token without a corresponding Word2Vec embedding, was assigned the average Word2Vec embedding.

We then represented each review by the indices of its tokens from the constructed vocabulary, using a function described below.

Function - encode_reviews

This function encodes a set of tokenized reviews into numerical representations based on a predefined vocabulary while maintaining a fixed length for each review.

The function follows these steps:

- Iterates through each review.
- For each token in the review:
  - Checks if the token exists in the provided vocabulary and assigns its corresponding index.
  - If the token is not in the vocabulary, assigns the index for the unknown token which is 1.
- Adjusts the length of the encoded review:
  - If shorter than a predefined length, pads with zeros to match the required length.
  - If longer than a predefined length, truncates by keeping tokens from both the beginning and end of the review.
- Returns the encoded reviews.

To build CNN classifiers, we implemented classes and functions for each step of training, evaluation, and testing, as described below.

Class - CNNClassifier

This class defines a neural network (CNN) for binary classification and consists of three functions.

Function - Constructor - init

This function initializes the CNN classifier by setting up the embedding layer, convolutional layers, max-pooling, and the MLP head.

The function follows these steps:

- Initializes the embedding layer:
  - If a pre-trained embedding matrix is provided, loads it and optionally freezes it.
  - Otherwise, initializes a random embedding layer.
- Adjusts the input dimensionality, if needed, using a linear layer.
- Defines multiple convolutional layers with different filter sizes:
  - Each convolutional layer has multiple stacked layers with ReLU activation function and residual connection.
- Adds a max-pooling layer to extract the most important features from the convolutional outputs (features maps).
- Defines the MLP head for producing the final classification output.

Function - define_mlp

This function constructs a Multi-Layer Perceptron (MLP) with optional dropout, batch normalization, and layer normalization.

The function follows these steps:

- Initializes a list of layers.
- Iterates over the specified hidden layers:
  - Adds a fully connected layer.
  - Applies a ReLU activation function.
  - Optionally applies dropout, batch normalization, and layer normalization.
- Adds an output layer.
- Applies a Sigmoid activation function to the output.
- Returns the constructed MLP as a sequential module.

Function - forward

This function defines the forward pass of the CNN classifier, processing input text sequences and producing classification outputs.

The function follows these steps:

- Retrieves word embeddings for the input text.
- If necessary, adjusts the input/embedding dimensionality.
- Transposes the input/embeddings to match the expected input format for convolutional layers.
- Passes the embeddings through multiple convolutional layers:
  - Applies stacked layers with residual connections.
  - Uses ReLU activation function at each step.
- Applies max-pooling to extract key features from convolutional outputs.
- Concatenates feature vectors from different convolutional layers.
- Passes the final extracted feature vector through the MLP head.
- Returns the final classification output.

Function - train_cnn_classifier

This function trains a CNN classifier using a given dataset, optimizes its weights, and implements early stopping to prevent overfitting.

The function follows these steps:

- Initialize training settings
  - Uses `Binary Cross-Entropy Loss` for binary classification.
  - Optimizes using `Adam` optimizer with a specified learning rate.
  - Implements early stopping with a patience of 5 epochs.
- Training Loop:
  - Iterates over the mini-batches of the training dataset:
    - Resets the optimizer gradients.
    - Performs forward pass to get predictions.
    - Computes training loss.

        - Applies backpropagation to compute gradients.
        - Updates model weights.
    - Stores the average training loss for the epoch.
- Validation Loop:
    - Iterates over the mini-batches of the validation dataset:
        - Computes validation loss.
        - Stores the average validation loss for the epoch.
- Perform early stopping:
    - If the new validation loss is lower, saves the model and deletes the previous best model checkpoint.
    - If no improvement for 5 consecutive epochs, early stopping is triggered.
- Returns:
    - A list of training losses per epoch.
    - A list of validation losses per epoch.
    - The file path of the best model checkpoint.

Function - compute_classification_scores

This function is used to compute the classification metrics of a classifier, including precision, recall, f1-score, and precision-recall AUC.

The function follows these steps:

- Computes precision, recall, and f1-score for `neg` (negative) and `pos` (positive) classes and the macro-averaged values of them.
- Computes precision-recall AUC for `neg` (negative) and `pos` (positive) classes.
- Computes macro-averaged precision-recall AUC.
- Returns the classification scores.

Function - evaluate

This function evaluates a trained CNN classifier on a given dataset by computing classification metrics.

The function follows these steps:

- Sets the classifier to evaluation mode.
- Iterates over the mini-batches of the provided dataset:
    - Gets predicted probabilities from the classifier.
    - Converts probabilities to binary predictions using a 0.5 threshold.
- Compute classification scores using the `compute_classification_scores` function.
- Returns the classification scores.

After defining all necessary functions, we trained and evaluated multiple CNN classifiers using different configurations (pre-embeddings, filter size, number of convolutional layers, hidden layers, dropout, normalization etc.).

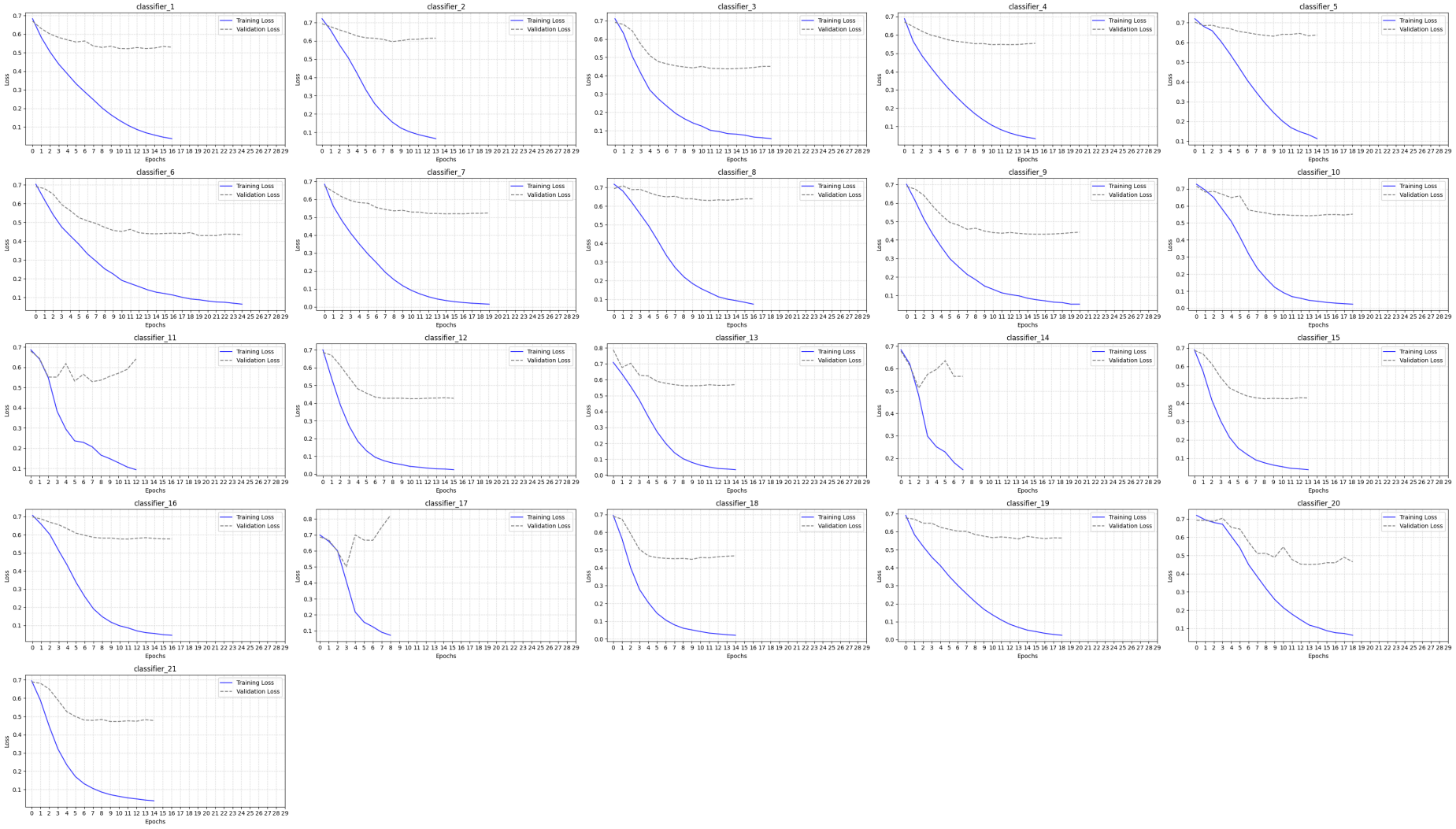
First, we converted the training, validation, and test data into `PyTorch` tensors and loaded them into `DataLoaders` for batch processing. For each classifier configuration, we initialized a CNN model, trained it using the `train_cnn_classifier` function, and evaluated it on the training, validation, and test sets using the `evaluate` function, utilizing the best performing checkpoint for each classifier.

Below are the configurations tested for all classifiers.

Name	Pre-Embeddings	Freeze	Filter Size	Filters	Convolutional Layers	MLP Head - Hidden Layers	MLP Head - Dropout	MLP Head - Batch Normalization	MLP Head - Layer Normalization
Classifier_1	False	True	[2]	64	2	None	0	False	False
Classifier_2	False	True	[2]	64	3	[32]	0.3	True	False
Classifier_3	True	True	[2]	64	3	[32]	0.3	True	False
Classifier_4	False	True	[3]	64	2	None	0	False	False
Classifier_5	False	True	[3]	64	3	[32]	0.3	True	False
Classifier_6	True	True	[3]	64	3	[32]	0.3	True	False
Classifier_7	False	True	[4]	64	2	None	0	False	False
Classifier_8	False	True	[4]	64	3	[32]	0.3	True	False
Classifier_9	True	True	[4]	64	3	[32]	0.3	True	False
Classifier_10	False	True	[2, 3]	64	2	[64]	0.3	True	False
Classifier_11	True	True	[2, 3]	64	3	None	0	False	False
Classifier_12	True	False	[2, 3]	64	3	[64]	0.3	True	False
Classifier_13	False	True	[2, 4]	64	2	[64]	0.3	True	False
Classifier_14	True	True	[2, 4]	64	3	None	0	False	False
Classifier_15	True	False	[2, 4]	64	3	[64]	0.3	True	False
Classifier_16	False	True	[3, 4]	64	2	[64]	0.3	True	False
Classifier_17	True	True	[3, 4]	64	3	None	0	False	False
Classifier_18	True	True	[3, 4]	64	3	[64]	0.3	True	False
Classifier_19	False	True	[2, 3, 4]	32	3	None	0	False	False
Classifier_20	True	True	[2, 3, 4]	32	3	[48, 24]	0.3	True	False
Classifier_21	True	False	[2, 3, 4]	32	3	[48]	0.3	True	False

For all CNN classifiers, we tracked the training and validation losses across all epochs (30) during training process and recorded the classification scores for the training, validation, and test datasets.

After collecting all the loss values, we visualized the corresponding curves, as shown below.



As shown in the plots above, the best-performing classifiers are 3, 6, 9, 12, 15, 18, 20 and 21, with their loss curves decreasing during training until the validation loss begins to rise, triggering early stopping. Among these eight, classifier_15 achieved the lowest validation loss, approximately 0.4240 at epoch 9. Therefore, we selected it for comparison with the baseline classifiers, as detailed below.

Classification Scores across all Classifiers

Training Dataset

Class	Metric	CNN Classifier	RNN Classifier	MLP Classifier	k-NN Classifier	Majority Classifier
neg	Precision	1.0	1.0	0.9251	0.8237	0.0
	Recall	1.0	1.0	0.8821	0.8429	0.0
	F1-Score	1.0	1.0	0.9031	0.8332	0.0
	Precision-Recall AUC Score	1.0	1.0	0.9658	0.9091	0.75
pos	Precision	1.0	1.0	0.8874	0.8391	0.5
	Recall	1.0	1.0	0.9286	0.8196	1.0
	F1-Score	1.0	1.0	0.9075	0.8293	0.6667
	Precision-Recall AUC Score	1.0	1.0	0.9698	0.9108	0.75
macro avg	Precision	1.0	1.0	0.9062	0.8314	0.25
	Recall	1.0	1.0	0.9054	0.8313	0.5
	F1-Score	1.0	1.0	0.9053	0.8312	0.3333
	Precision-Recall AUC Score	1.0	1.0	0.9678	0.91	0.75

Validation Dataset

Class	Metric	CNN Classifier	RNN Classifier	MLP Classifier	k-NN Classifier	Majority Classifier
neg	Precision	0.7897	0.8692	0.8435	0.7094	0.0
	Recall	0.8292	0.8583	0.8083	0.7833	0.0
	F1-Score	0.8089	0.8637	0.8255	0.7446	0.0
	Precision-Recall AUC Score	0.8826	0.9304	0.9209	0.8141	0.75
pos	Precision	0.8202	0.8601	0.816	0.7581	0.5
	Recall	0.7792	0.8708	0.85	0.6792	1.0
	F1-Score	0.7991	0.8654	0.8327	0.7165	0.6667
	Precision-Recall AUC Score	0.8906	0.9093	0.9041	0.7891	0.75
macro avg	Precision	0.8049	0.8646	0.8297	0.7338	0.25
	Recall	0.8042	0.8646	0.8292	0.7312	0.5
	F1-Score	0.804	0.8646	0.8291	0.7305	0.3333
	Precision-Recall AUC Score	0.8866	0.9198	0.9125	0.8016	0.75

Test Dataset

Class	Metric	CNN Classifier	RNN Classifier	MLP Classifier	k-NN Classifier	Majority Classifier
neg	Precision	0.8041	0.8511	0.8514	0.7062	0.0
	Recall	0.78	0.8	0.745	0.745	0.0
	F1-Score	0.7919	0.8247	0.7947	0.7251	0.0
	Precision-Recall AUC Score	0.875	0.9143	0.9011	0.7804	0.75
pos	Precision	0.7864	0.8113	0.7733	0.7302	0.5
	Recall	0.81	0.86	0.87	0.69	1.0
	F1-Score	0.798	0.835	0.8188	0.7095	0.6667
	Precision-Recall AUC Score	0.8668	0.9135	0.8849	0.7638	0.75
macro avg	Precision	0.7953	0.8312	0.8124	0.7182	0.25
	Recall	0.795	0.83	0.8075	0.7175	0.5
	F1-Score	0.795	0.8298	0.8067	0.7173	0.3333
	Precision-Recall AUC Score	0.8709	0.9139	0.893	0.7721	0.75

From the classification scores across all datasets, we reached the following conclusions:

- The RNN classifier is the best-performing model, consistently achieving the highest Precision, Recall, F1-Score, and AUC across all datasets. Its strong performance indicates that it effectively captures sequential dependencies in the text data, making it highly suitable for the current text classification problem.
- The MLP classifier outperforms the CNN classifier in the validation and test datasets across most metrics. While it does not reach the performance of the RNN, it generalizes well and proves to be a strong model for the current problem.
- The CNN classifier performs well in the training dataset but slightly underperforms compared to the MLP classifier in validation and test datasets, suggesting that it might be overfitting to the training data. Despite this, it remains a solid model for the current the problem.
- The k-NN classifier struggles significantly, with notably lower scores across all metrics. This indicates that it is not well-suited for the current problem, likely due to its inability to capture contextual relationships effectively.
- The Majority classifier is ineffective, as expected, since it simply predicts the majority class and does not differentiate between classes. This results in poor overall metrics.