Assignment 3 - Exercise 1 - Text Classification with Recurrent Neural Networks

Contributor: Dimitris Vougioukos - f3352411

The exercise 1 focuses on building an RNN 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 1

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 is 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 RNN classifiers, we implemented classes and functions for each step of training, evaluation, and testing, as described below.

Class - RNNClassifier

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

> **Function - Constructor - init**
>
> This function initializes the RNN classifier by setting up the embedding layer, RNN model, self-attention MLP, and 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.

- Configures the RNN model with the specified type (RNN, LSTM, GRU), hidden dimension, number of layers, and bidirectionality.
- Defines a self-attention MLP for computing attention scores.
- Defines a 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.
- If the MLP type is "mlp head", 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 RNN classifier, processing input reviews and producing classification outputs.

The function follows these steps:

- Retrieves word embeddings for the input reviews.
- Passes the embeddings through the RNN to obtain hidden states.
- Computes attention scores for all final hidden states using the self-attention MLP.
- Applies a padding mask to prevent attention to padding tokens.
- Normalizes attention scores using the softmax function.
- Computes a weighted sum of hidden states based on the attention scores for all the input reviews.
- Passes the resulting feature representation through the MLP head to produce the final classification output.

Function - train_rnn_classifier

This function trains a RNN 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 RNN 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 RNN classifiers using different configurations (number of RNN layers, pre-embeddings, 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 RNN model, trained it using the `train_rnn_classifier` function, and evaluated it on the training, validation, and test sets using the `evaluate` function, utilizing the best performing checkpoint for each classifier.

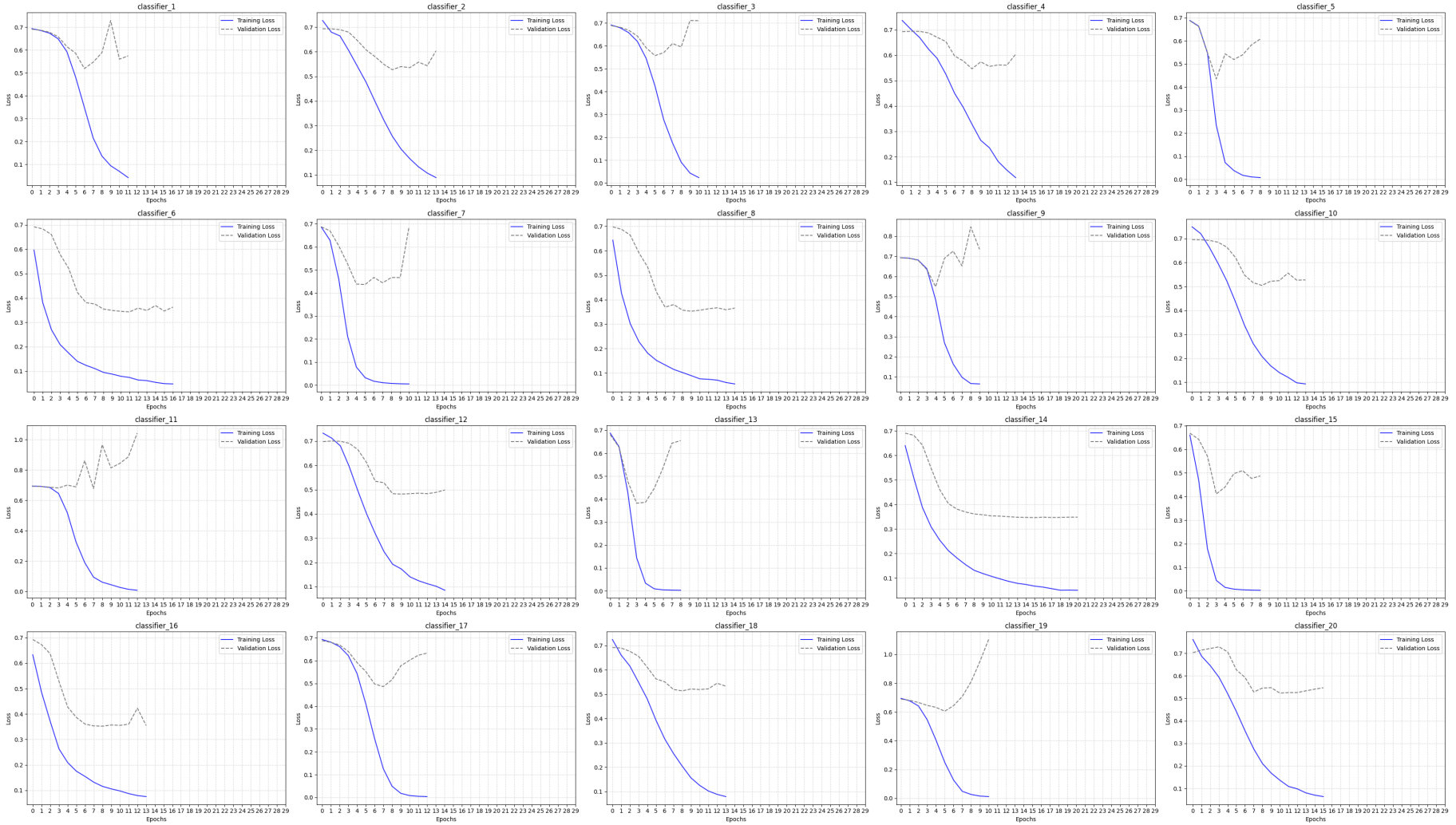
Below are the configurations tested for all classifiers.

Name	Type	State Dimension	Pre-Embeddings	Freeze	Layers	Attention-Hidden Layers	Attention-Drop	Attention-Batch Norm	Attention-Layer Norm	Head-Hidden Layers	Head-Drop	Head-Batch Norm	Head-Layer Norm
Classifier_1	RNN	32	False	True	2	None	0	False	False	None	0	False	False
Classifier_2	RNN	32	False	True	2	None	0	False	False	[32,16]	0.2	True	False
Classifier_3	RNN	32	False	True	2	[32,16]	0.2	True	False	None	0	False	False
Classifier_4	RNN	32	False	True	2	[32,16]	0.2	True	False	[32,16]	0.2	True	False
Classifier_5	LSTM	32	True	True	2	None	0	False	False	None	0	False	False
Classifier_6	LSTM	32	True	True	2	None	0	False	False	[32,16]	0.2	True	False
Classifier_7	LSTM	32	True	True	2	[32,16]	0.2	True	False	None	0	False	False
Classifier_8	LSTM	32	True	True	2	[32,16]	0.2	True	False	[32,16]	0.2	True	False
Classifier_9	LSTM	32	False	True	3	None	0	False	False	None	0	False	False

Name	Type	State Dimension	Pre-Embeddings	Freeze	Layers	Attention-Hidden Layers	Attention-Drop	Attention-Batch Norm	Attention-Layer Norm	Head-Hidden Layers	Head-Drop	Head-Batch Norm	Head-Layer Norm
Classifier_10	LSTM	32	False	True	3	None	0	False	False	[32,16]	0.2	True	False
Classifier_11	LSTM	32	False	True	3	[32,16]	0.2	True	False	None	0	False	False
Classifier_12	LSTM	32	False	True	3	[32,16]	0.2	True	False	[32,16]	0.2	True	False
Classifier_13	GRU	32	True	True	3	None	0	False	False	None	0	False	False
Classifier_14	GRU	32	True	True	3	None	0	False	False	[32,16]	0.2	True	False
Classifier_15	GRU	32	True	True	3	[32,16]	0.2	True	False	None	0	False	False
Classifier_16	GRU	32	True	True	3	[32,16]	0.2	True	False	[32,16]	0.2	True	False
Classifier_17	GRU	32	False	True	2	None	0	False	False	None	0	False	False
Classifier_18	GRU	32	False	True	2	None	0	False	False	[32,16]	0.2	True	False
Classifier_19	GRU	32	False	True	2	[32,16]	0.2	True	False	None	0	False	False
Classifier_20	GRU	32	False	True	2	[32,16]	0.2	True	False	[32,16]	0.2	True	False

For all RNN 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 6, 8, 13, 14, and 16, with their loss curves decreasing during training until the validation loss begins to rise, triggering early stopping. Among these five, classifier_6 achieved the lowest validation loss, approximately 0.343 at epoch 12. Therefore, we selected it for comparison with the baseline classifiers, as detailed below.

Classification Scores across all Classifiers

Training Dataset

Class	Metric	RNN Classifier	MLP Classifier	k-NN Classifier	Majority Classifier
neg	Precision	1.0	0.9251	0.8237	0.0

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

Validation Dataset

Class	Metric	RNN Classifier	MLP Classifier	k-NN Classifier	Majority Classifier
neg	Precision	0.8692	0.8435	0.7094	0.0
	Recall	0.8583	0.8083	0.7833	0.0
	F1-Score	0.8637	0.8255	0.7446	0.0
	Precision-Recall AUC Score	0.9304	0.9209	0.8141	0.75
pos	Precision	0.8601	0.816	0.7581	0.5
	Recall	0.8708	0.85	0.6792	1.0
	F1-Score	0.8654	0.8327	0.7165	0.6667
	Precision-Recall AUC Score	0.9093	0.9041	0.7891	0.75
macro avg	Precision	0.8646	0.8297	0.7338	0.25
	Recall	0.8646	0.8292	0.7312	0.5

Class	Metric	RNN Classifier	MLP Classifier	k-NN Classifier	Majority Classifier
	F1-Score	0.8646	0.8291	0.7305	0.3333
	Precision-Recall AUC Score	0.9198	0.9125	0.8016	0.75

Test Dataset

Class	Metric	RNN Classifier	MLP Classifier	k-NN Classifier	Majority Classifier
neg	Precision	0.8511	0.8514	0.7062	0.0
	Recall	0.8	0.745	0.745	0.0
	F1-Score	0.8247	0.7947	0.7251	0.0
	Precision-Recall AUC Score	0.9143	0.9011	0.7804	0.75
pos	Precision	0.8113	0.7733	0.7302	0.5
	Recall	0.86	0.87	0.69	1.0
	F1-Score	0.835	0.8188	0.7095	0.6667
	Precision-Recall AUC Score	0.9135	0.8849	0.7638	0.75
macro avg	Precision	0.8312	0.8124	0.7182	0.25
	Recall	0.83	0.8075	0.7175	0.5
	F1-Score	0.8298	0.8067	0.7173	0.3333
	Precision-Recall AUC Score	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.
- The MLP classifier performs well but lags slightly behind the RNN Classifier.
- The k-NN classifier struggles with generalization and shows significantly lower scores across all metrics, suggesting that it may not be well-suited for text classification tasks requiring contextual understanding.
- The Majority classifier is ineffective, as expected, since it does not differentiate between classes and results in poor metrics.