Coding Project

Apostolopoulou Ioanna, Kofopoulos Lymberis Stamatis, Mourelatos Gerasimos

January 2024

Contents

1 Abstract

This project explores the development of a neural network classifier for text categorization, a fundamental task in machine learning. The classifier is designed to categorize relatively small texts into primary and secondary categories. The approach involves constructing a neural network that integrates various input features, including title, content, source, author, and publication date, to predict hierarchical news categories. The neural network employs a combination of Bidirectional Long Short-Term Memory (LSTM) layers and Dense layers, with dropout and batch normalization for regularization. The model is trained on a dataset comprising approximately 10,917 news articles collected in 2019, categorized manually into 17 primary and 109 secondary categories according to the NewsCodes Media Topic classification. Data augmentation techniques are applied to the training data to enhance the model's robustness and performance.

2 Methodology

2.1 Data preprocessing

For the data preprocessing for the neural network project is managed through the DataHandler class. Here's a detailed description:

1. We load the vocabulary for the categories from an excel file. This vocabulary includes all the categories for MediaTopic NewsCodes, provided by the International Press Telecommunications Council
(link for first category) (link for second category)

2. The categories for the articles are encoded using a StringLookup layer.

3. The articles are loaded from the CSV file, which is randomly shuffled and split into training and testing sets based on the specified percentage (default is 80% for training).

4. The input data are transformed into a suitable format for the neural network. The 'published' dates are converted to a numerical format (year, month, day, hour, minute, second), handling any missing values. The textual data (title, content, source, author) is converted into padded sequences of integers using the adapted vectorizer.

5. After that if augmentation is enabled, the training dataset undergoes augmentation, more specifically words in the title and the content that are in the same sentence are swapped and replaced with synonyms.

2.2 Neural network architecture

We used keras to develop model for the article classification. Here is the model architecture

1. Input Layers:
The model consists of five input layers, each corresponding to different features of the articles: title, content, source, author, and published date.

2. Embedding Layers:
Both the title and content inputs are passed through separate embedding layers. These layers transform the word indices into dense vectors of fixed size (output_dim=100), capturing word relationships. Dropout layers with a rate of 0.5 are applied to both embedding outputs to prevent overfitting

3. Bidirectional LSTM Layers:
The LSTM (Long Short-Term Memory) layers are used to process the title and the content text data. The model employs bidirectional LSTMs, allowing it to capture dependencies from both directions (forward and backward) of the sequence. The content is processed by a 40-unit bidirectional LSTM, and the title by a 10-unit bidirectional LSTM.

4. Dense and Batch Normalization Layers:
The model features a dense layer, followed by batch normalization and dropout. The dense layer combines features from the title's LSTM output, published date, source and author.

5. Output Layers:
The model has two output layers, each a dense layer with a softmax activation function.

6. Model Compilation:
The model uses the Adam optimizer and sparse categorical crossentropy as the loss function for both outputs. The model's performance is evaluated based on sparse categorical accuracy, and different loss weights are assigned to the two outputs (category_1=1, category_2=7.5).

7. Training:
The model was trained for 20 epochs with a batch size 128. We used 80% of the data for training and 20% for valuation. We also used 2 callbacks. The first one to reduce the learning rate on plateaus (after 2 epochs) and the second one is for early stopping (after 5 epochs).

3 Results

The training time for this model (trained in a CPU) was 4 hours, 26 minutes and 39 seconds and the inference time for 10 samples was 1.117 seconds. The model weights are shared on the google drive: https://drive.google.com/file/d/1FNyea1Q6zDYYZmANZl3lokEv16JTU4IM/view?usp=sharing

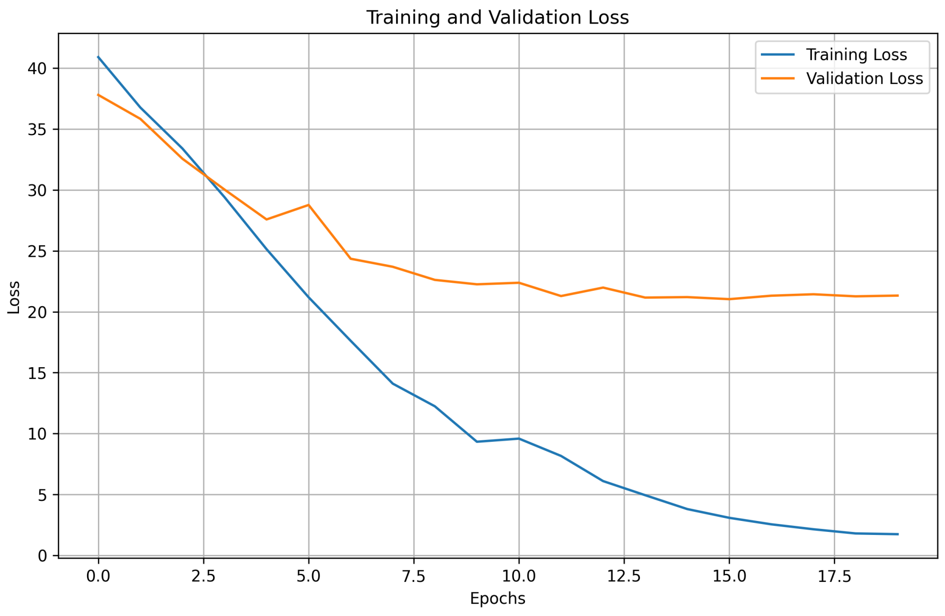

Figure 1: Training and valuation loss.

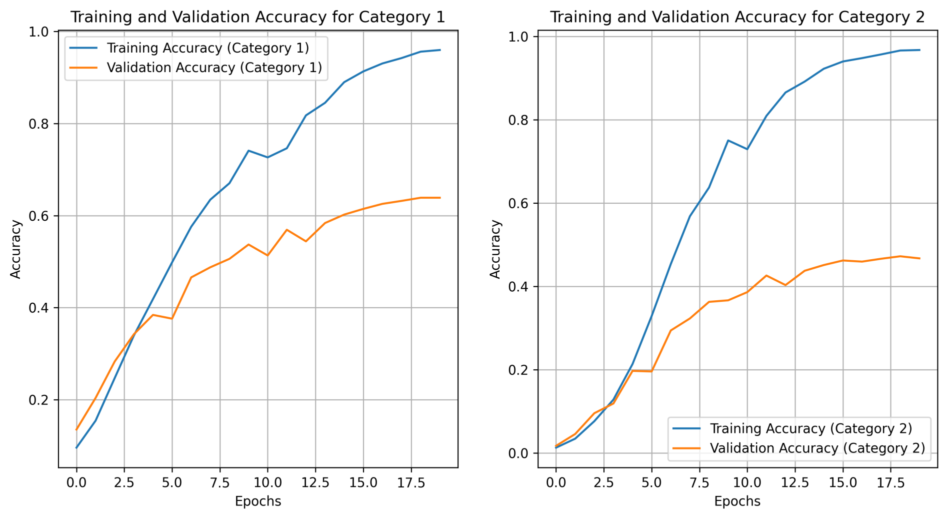

Figure 2: Training and valuation accuracy.