Deep Learning

Practical Exercises - Week 13

Recurrent Neural Network

ICAI

FS 2024

1 Topic

In this lab session we will take a look at recurrent neural networks (RNN), more precisely at gated RNNs such as LSTMs (Long-Short-Term-Memory) and GRUs (Gated Recurrent Units). Gated RNNs are the most commonly used type of recurrent neural networks. RNNs are typically used to process sequential data. To better understand recurrent neural networks and in particular LSTMs and GRUs, check out this very good explanation.

2 Recurrent Neural Network / GRU

In this week's lab we will apply a bidirectional RNN (GRU) to a binary text classification problem. Specifically, we will train a model that classifies movie reviews into positive and negative, using the IBM movie review dataset. If you are not familiar with bidirectional RNNs, see Chapter 10.3 of the textbook.

First, a detailed tutorial will show you how to build an RNN for text classification. Next, you will try to build, train and test your own RNN-based text classifier.

2.1 Prepare Data

The Python module `data.py` contains the parts of the program which load and prepare the datasets. First, the entire dataset is downloaded and split it into a train, valid, test, and unsupervised subset. The unsupervised dataset is a subset of movie reviews which is not labeled. We will need this subset to initialize the text-to-number encoder later.

```
VALID_SPLIT = 0.1

datasetset, info = tfds.load("imdb_reviews", with_info=True, as_supervised=True)

valid_size = int(info.splits["train"].num_examples * VALID_SPLIT)

self._valid_dataset = dataset["train"].take(valid_size)
self._train_dataset = dataset["train"].skip(valid_size)
self._test_dataset = dataset["test"]
self._unsupervised_dataset = dataset["unsupervised"]
```

Let's take a look at an example of a movie review:

```
example = next(self._train_dataset.as_numpy_iterator())
print("text:", example[0])
print("label:", example[1])
```

```
text: b"Basically, this movie is one of those rare movies you either hate ..."
label: 1
```

Next, we shuffle the training dataset, create mini-batches and prefetch elements. The `prefetch` method allows later elements to be prepared while the current element is being processed. `batch` and `prefetch` are applied to all datasets, although here it is only shown for the training dataset.

```
MINI_BATCH_SIZE = 64
self._train_dataset = self._train_dataset.shuffle(10000)
self._train_dataset = self._train_dataset.batch(MINI_BATCH_SIZE)
self._train_dataset = self._train_dataset.prefetch(tf.data.AUTOTUNE)
```

2.2 RNN Model

Now that the datasets are prepared, we need to define the model we want to train. The model definition is provided in the Python module `model.py`

Each element of our dataset consists of a string of words (one movie review per element). However, neural networks do not know how to process letters, strings or sentences. Therefore, we first have to convert each word into a vector using a TextVectorizer in combination with an Embedding layer.

- The TextVectorizer simply converts the text to a sequence of token indices (basically a lookup table which maps from a word to an integer).
- An Embedding layer stores one vector per word. When called, it converts the sequences of word indices to sequences of vectors. These vectors are trainable. After training (with a large enough dataset), words with similar meanings result in similar vectors. If you want to know more about embedding layers, see here.

The above two layers of our model convert each word to a 64-dimensional vector. For this purpose we use the following TensorFlow classes / methods:

```
def __init__(self, vocabulary):
    self.enc_1 = TextVectorization(max_tokens=1000)
    self.enc_1.adapt(vocabulary)
    self.emb_1 = Embedding(
        input_dim=len(self.enc_1.get_vocabulary()), output_dim=64, mask_zero=True
    )


def call(self, x, training=False):
    t_enc_1 = self.enc_1(x)
    t_emb_1 = self.emb_1(t_enc_1)
```

As you can see in the code snippet above, the vocabulary is passed to the `TextVectorization` object by calling the `adapt` method. The `TextVectorization` object then maps each word to a specific integer. To complete the movie review classifier, we add a bidirectional RNN on top of these preprocessing layers. The concatenated final states of the two RNNs are then used as input to a fully-connected neural network classifier with output size 1. Figure 1 shows an illustration of the network architecture.

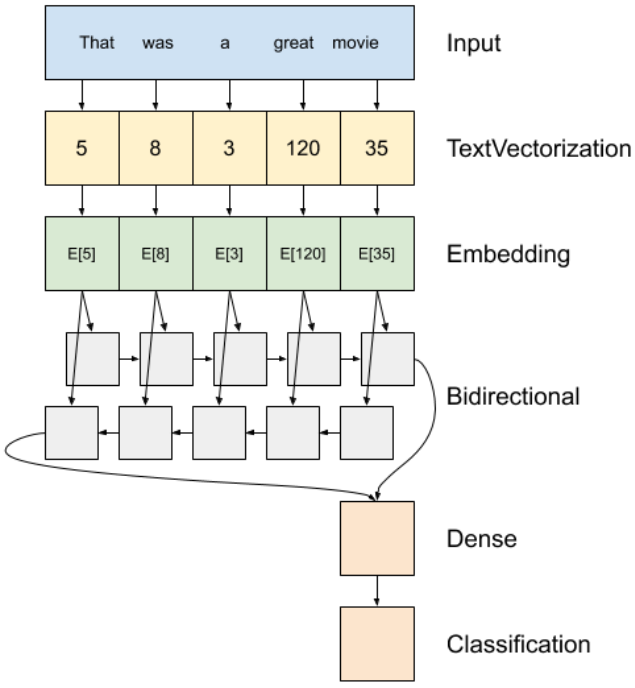

Figure 1: Network architecture.

2.3 Training

The training, evaluation and testing process as well as the preparation of the datasets and instantiation of the model is already prepared in the script `main.py`.

Implementing and invoking the training and validation programs is much easier with the built-in methods `model.fit()`, `model.compile()` and `model.evaluate()`. Remember, in previous labs we had to do our own implementation

of the training and validation programs. However, if the training process or the loss function become more complex (e.g. GANs), these built-in methods are inappropriate and cannot be used.

3 Exercise

Now it is your task to complete the RNN which is described in subsection 2.2 and illustrated in Figure 1. Please complete the following tasks:

- Instantiate and call a Bidirectional layer and pass a GRU layer as Layer instance.
- Instantiate and call two Dense layers. Use a ReLU activation function for the hidden layer and no activation function for the output function (sigmoid is not necessary since `from_logits` is set to `True`).
- Train your model for a few epochs and then test it with your own movie review.
- Try to improve the performance by stacking more Bidirectional layers or Dense layers.