AIML428

- Thursday Presentations
  - Nanda, Ella, Hadas, Huixin(Joy), Ye li, Guangyong
  - Submit your pptx file using our submission system
- Today
  - Implement CNN for text classification
- Project
  - Step1; due this Friday
  - You may follow one of the online tutorials
  - Submit the source files online
    - Do not include the dataset
    - Use README file to explain
      - where the data is,
      - how to run your file
      - A very brief summary on what you have done

Python for CNN with word embedding

- A simplified example is attached at schedule page
- Sequence model: preserving word order
- Train word embedding as part of a deep learning model
- Use pre-trained word embedding
- Built CNN
- Train CNN: compile, fit
- Discussions

Prepare data: Sequential model

- Sequence model: preserving word order

- Creating a tokenizer object, tokenizes the texts into words
- Each word is transferred to its integer ID
  - Creates a vocabulary using all words or the top K tokens (e.g. 20000)
- Transforming text documents to sequence of word IDs and pad them
  - Converts the tokens into sequence vectors
  - Pads the sequences to a fixed sequence length

Design the model

- Embedding layer
- Convolutional layer
- Pooling layer
- Dense layer
- Output layer (Dense layer)

Embedding with random initial weights

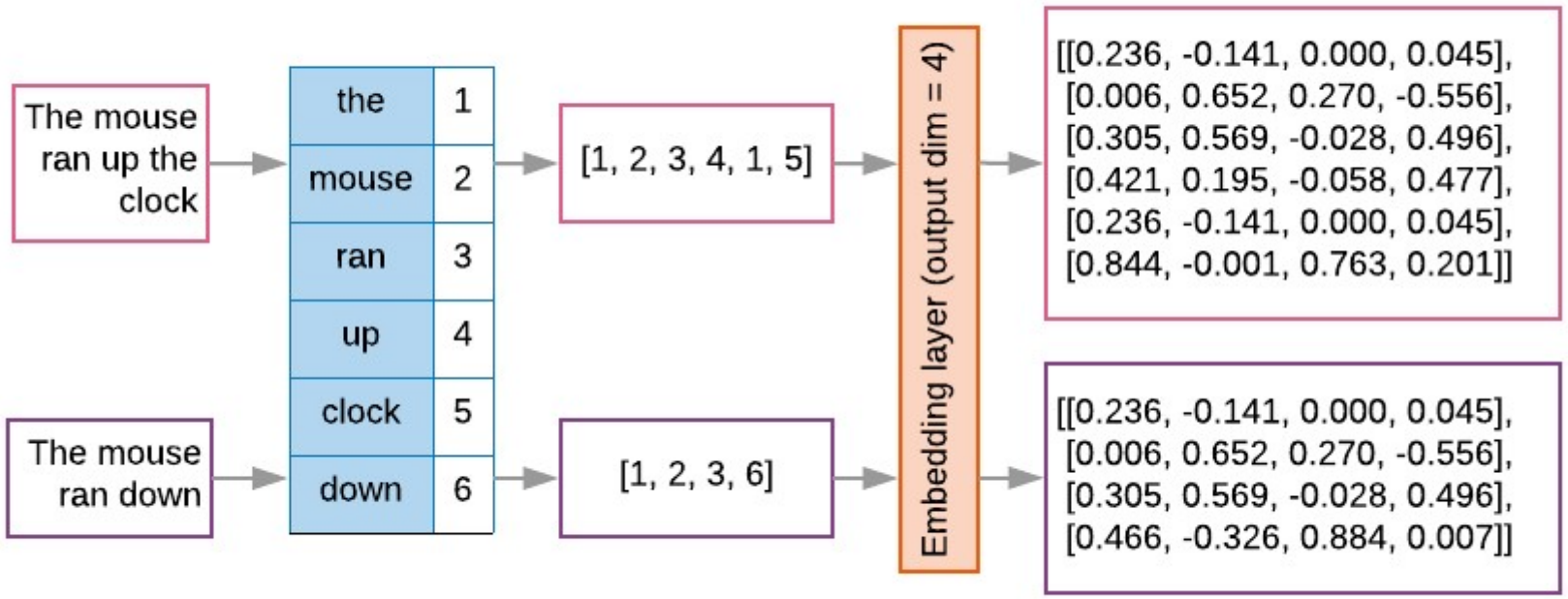
- a word embedding can be learned as part of a deep learning model. This can be a slower approach, but tailors the model to a specific training dataset.
- The Embedding layer is initialized with random weights and will learn an embedding for all of the words in the training dataset.
- input_dim: This is the size of the vocabulary in the text data.
- output_dim: This is the size of the vector space in which words will be embedded.
- input_length: This is the length of input sequences, as you would define for any input layer of a Keras model.

How to use pre-trained word embeddings

- Each word index gets mapped to a dense vector of real values representing that word's location in semantic space

- Loading the pre-trained word embeddings
  - Dictionary:
    - Key: word
    - Value: vector
- Create a mapping of token and their respective embeddings
  - Each word ID, maps to a vector

The Input: example



Embedding layer Parameters

- **vocab_size as input_dim:** how many unique words (or top K) in the vocabulary
- **Embedding dimensions as output_dim**: The number of dimensions we want to use to represent word embeddings—i.e., the size of each word vector. Recommended values: 50–300. If you use pre-trained embeddings, it depends on which file you use.
- **weights:** the mapping between word Id to their embeddings
- input_length is the maximum length of documents.
- Trainable: whether you want to change these weights
  - Default is trainable, normally trainable is better

Summary on Embeddings

- Can add a embedding layer without the weights, so the word vector weights are learned as part of the network, but can be slow
- Can use pre-trained
- Can train a standalone word embedding using local data set, save and use later, more efficient

Convolutional layer

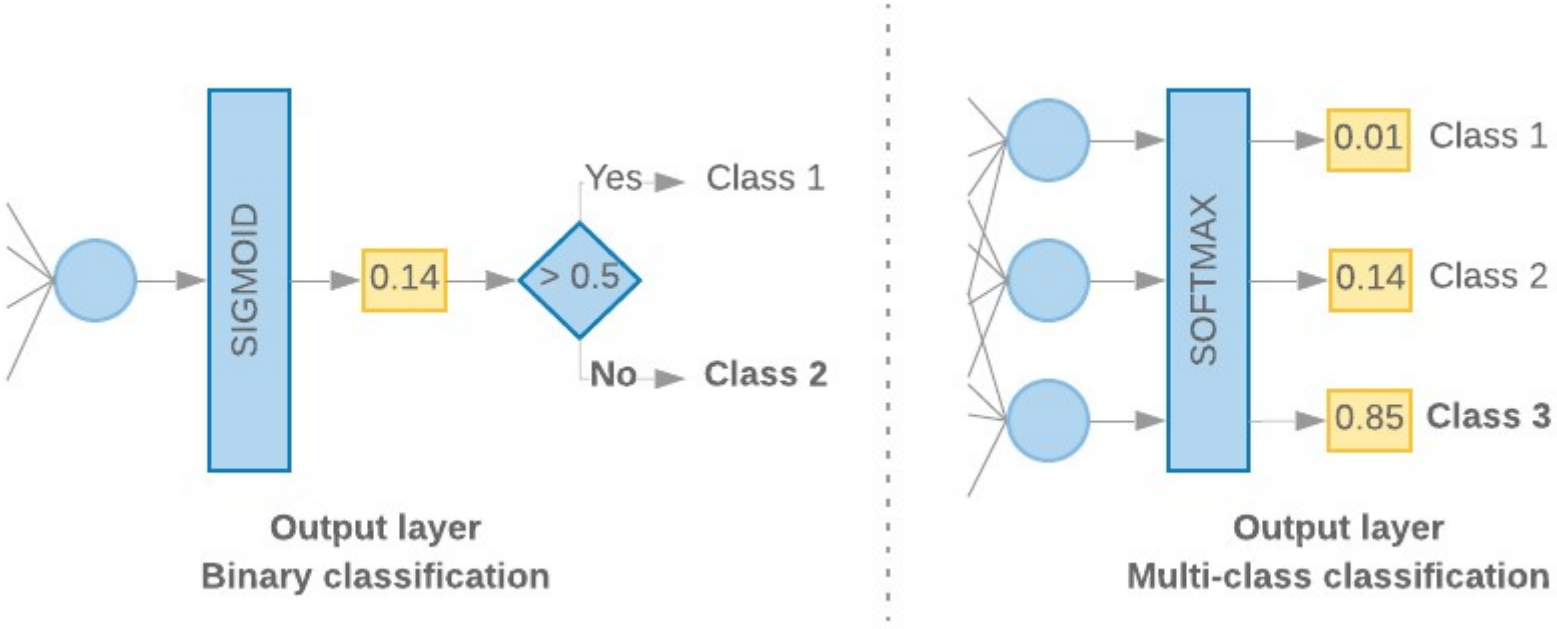
- 1D
- Number of filters
- Kernel size: The size of the convolution window. Recommended values: 3 or 5

Parameters

- **Number of layers in the model**: The number of layers in a neural network is an indicator of its complexity.
  - Too many layers will allow the model to learn too much information about the training data, causing overfitting.
  - Too few layers can limit the model's learning ability, causing underfitting.
  - For text classification datasets, normally we use 6 or 9 layers. In the example, we tried one, two, three layers.

- **Number of units per layer**: The units in a layer must hold the information for the transformation that a layer performs.
  - For the first layer, this is driven by the number of features.
  - In subsequent layers, the number of units depends on the choice of expanding or contracting the representation from the previous layer.
  - Try to minimize the information loss between layers. We tried unit values in the range [8, 16, 32, 64], and 32/64 units worked well.

The output: binary or multiple-class



the activation function of the last layer should be a sigmoid function, and the loss function used to train the model should be binary cross-entropy

the activation function of the last layer should be softmax, and the loss function used to train the model should be sparse categorical cross-entropy.

Softmax is a function that takes as input a vector of K real numbers, and normalizes it into a probability distribution consisting of K probabilities. All values are normalised to range [0, 1] and sum to one.

Train the model, learning parameters

- Metric: How to measure the performance of our model using a metric.
- Loss function: A function that is used to calculate a loss value that the training process then attempts to minimize by tuning the network weights.
- Optimizer: A function that decides how the network weights will be updated based on the output of the loss function.

Metric	accuracy
Loss function - binary classification	binary_crossentropy
Loss function - multi class classification	sparse_categorical_crossentropy
Optimizer	adam

Compile and fit

- In Keras, we can pass these learning parameters to a model using the compile method.

- Then use fit: training data, validation data, number of epochs, batch size, learning rate, verbose
  - Do not use test data in the learning process
  - Further split the training data into training and validation

  - Epochs: the number of times that the learning algorithm will work through the entire training dataset

  - Batch size: the number of samples to work through before updating the internal model parameters.
  - verbose: how much details you want to see

Learning rate

- Learning rate: This is the rate at which the neural network weights change between iterations.
  - A large learning rate may cause large swings in the weights, and we may never find their optimal values.
  - A low learning rate is good, but the model will take more iterations to converge.
  - It is a good idea to start low, say at 1e-4. If the training is very slow, increase this value. If your model is not learning, try decreasing learning rate.

Source: embeddings and CNN

- https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/#:~:text=2.-,Keras%20Embedding%20Layer,API%20also%20provided%20with%20Keras.
- A tutorial with examples and explanation
  - https://developers.google.com/machine-learning/guides/text-classification
- More code for pre-processing and train standalone word vectors.
  - https://machinelearningmastery.com/develop-word-embedding-model-predicting-movie-review-sentiment/
- More online tutorials on the assignment page