Recurrent Neural Networks for Natural Language Processing

BATH SPA UNIVERSITY

Data Analytics and Machine Learning

Overview

10:00 – 12:00

LECTURE

12:00 – 1:50

LAB SESSION

1:50 – 2:00

SESSION WRAP UP

Lesson Objectives

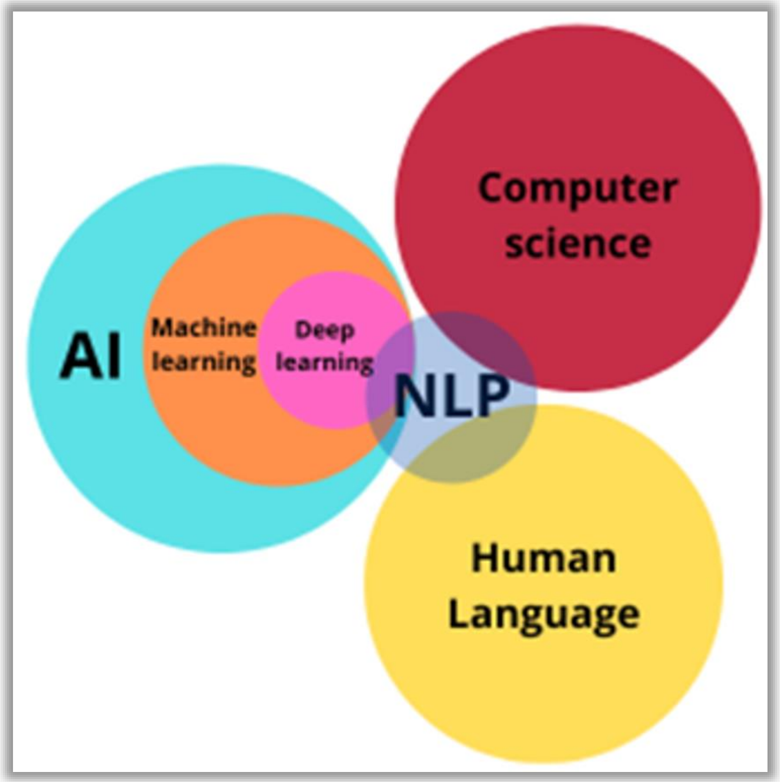
- Understand how machine learning models handle non-numeric data
- Understand which scenarios these might be applied to
- Work through a practical example

Keyword	Description
Machine Learning	A type of artificial intelligence that enables computers to learn from data and make decisions or predictions without being explicitly programmed.
Classification	This is a type of supervised learning task where the goal is to predict a categorical target variable.
Hyperparameter	These are parameters of the learning algorithm itself, not derived from the data, that need to be set before training the model.
Neural Network	A set of algorithms modelled after the human brain, designed to recognize patterns. They interpret sensory data through a kind of machine perception, labelling or clustering raw input.
Recurrent Neural Networks (RNN)	A type of artificial neural network designed to process sequential data. RNNs take information from prior inputs to influence the current input and output.
Long Short-Term Memory (LSTM)	A subtype of RNN that is designed to remember past information while forgetting irrelevant parts, making it effective for tasks involving sequential data.
Gated Recurrent Unit (GRU)	A type of RNN that operates similarly to LSTM but has fewer parameters, making it computationally more efficient.
Text Classification	A machine learning technique that assigns predefined categories to open-ended text, helping to organize and structure any kind of text data.
Early Stopping	A form of regularization used in machine learning to prevent overfitting. It allows training to stop once the model’s performance stops improving on a hold-out validation dataset.
Optimizer	A function or algorithm that adjusts the attributes of a neural network, such as weights and learning rates, to minimize the loss function and improve the model’s performance.

Introduction to Natural Language Processing

NLP - Natural Language Processing

- A branch of computer science
- Integration of computing and human language
  - Computational linguistics
  - Statistical Computer Science
  - Machine learning
  - Deep learning models
- Facilitates understanding of full meaning including intent and sentiment of speaker or writer



Natural Language Processing (NLP)

- Translate text
- Summarise large volumes of text
- Respond to spoken commands
  - Voice commanded GPS
  - Digital Assistants
  - Speech-to-text dictation
  - Customer service chatbots

- Increasingly being used in enterprise solutions
  - Streamline business operations
  - Simplify mission-critical processes

NLP – Human Language difficulties

- Human language is filled with **ambiguities**
  - Hard to create software that accurately determines:
    - Intended meaning
    - Context
    - Emotion
- **Irregularities** take humans years to learn
  - Must be taught to natural language-driven applications from the start for them to be useful

- **Challenges** include
  - Homonyms
  - Homophones
  - Sarcasm
  - Idiom
  - Metaphors
  - Grammar
  - Usage exceptions
    - Used in a different manner to their definition
  - Sentence structure variations e.g. to avoid monotony or to provide emphasis

BATH SPA UNIVERSITY

Note:

Usage exceptions: words used in a different manner to their definition
Sentence Structure: E.g,. To avoid monotony and providing emphasis

NLP - Tasks

Speech recognition

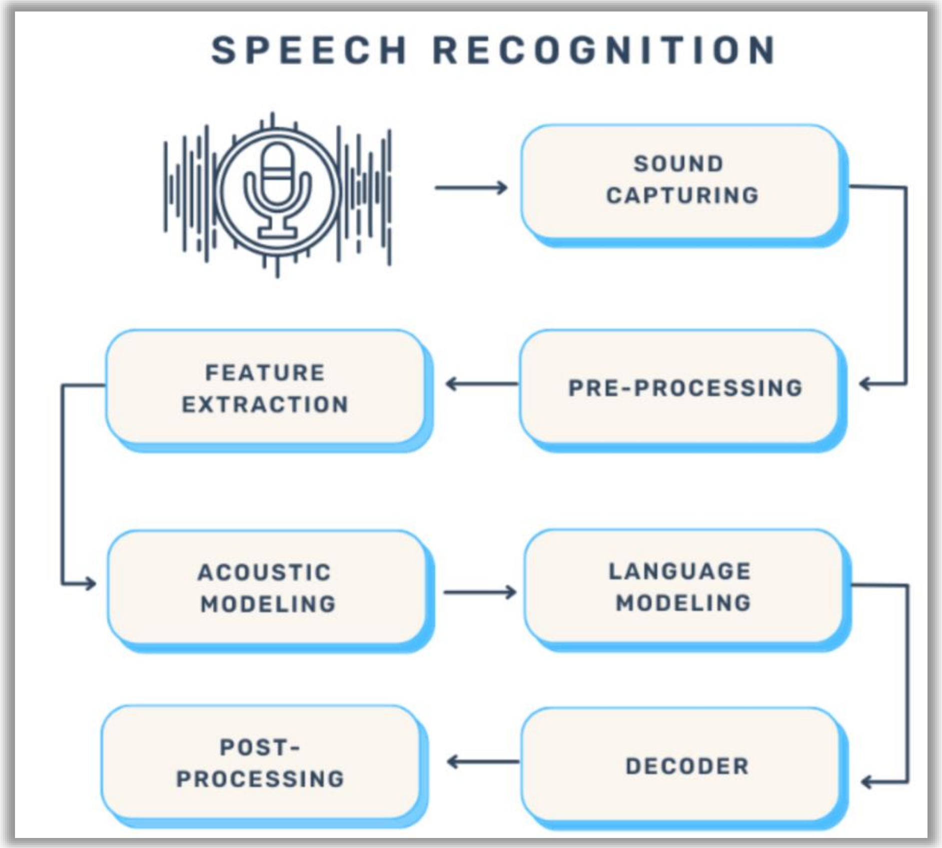
- Converts voice data into text data
    - **Implementation**
        - Machine learning algorithms to interpret human speech
    - **Example**
        - Voice-operated GPS systems

Part of speech tagging

- Determines speech of word based on its use and context
    - **Implementation**
        - Uses linguistic rules and statistical models
    - **Example**
        - Identifies ‘make’ as a verb ‘I can make a paper plane’

Word sense disambiguation

- Selects the meaning of a word based on context
    - **Implementation**
        - Uses semantic analysis algorithms
    - **Example**
        - Distinguishes the meaning of ‘make’ in ‘make the grade’
            - vs. ‘make a bet’



NLP - Tasks

Named entity recognition (NEM)
- Identifies words or phrases
    - As useful entities
    - **Implementation**
        - Machine learning models
            - Trained on annotated data
    - **Example**
        - Identifies ‘Kentucky’ as a location
        - Or ‘Fred’ as a man’s name

Co-reference resolution
- Identifies if two words refer to the same entity
    - **Implementation**
        - Uses rule-based methods
            - And machine learning models
    - **Example**
        - Determines that ‘she’ refers to ‘Mary’
            - In a given context

BATH SPA UNIVERSITY

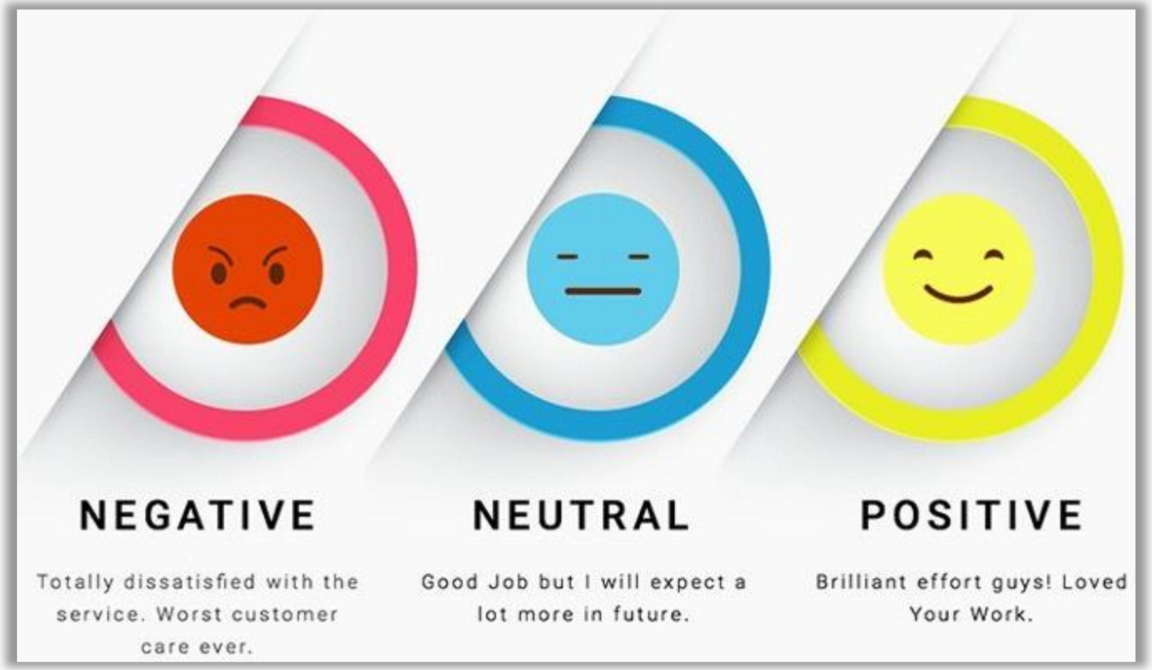
NLP - Tasks

Sentiment analysis

- Extracts subjective qualities like attitudes and emotions from text
  - **Implementation**
    - NLP Text analysis
    - Computational linguistics
  - **Example**
    - Determines user sentiment
      - From product reviews

Natural language generation

- Converts structured information
  - Into human language
  - **Implementation**
    - Uses
      - Templates
      - Rules
      - Machine learning models
  - **Example**
    - A weather app that generates a weather report
      - From meteorological data



NLP – Use Cases

Social media sentiment analysis

- Extracts attitudes and emotions
  - from social media
    - Posts
    - Responses
    - Reviews, etc

- Implementation
  - Natural language processing
  - Text analysis
  - Computational linguistics

- Example
  - Analysing customer sentiment
    - Towards products or promotions on social media platforms

Text summarisation

- Creates summaries
  - Of large volumes of digital text

- Implementation
  - Semantic reasoning
  - Natural language generation
  - To add to summaries
    - Useful context
    - Conclusions

- Example
  - Summarising news articles or research papers
    - For quick reading

NLP – Good Libraries

Hugging Face Transformers

- Pre-trained models for tasks on text, vision, audio

spaCy

- Supports tokenization and training for 60+ languages

Fairseq

- Train custom models
- Translation, summarisation, language modelling, etc

Jina

- Building scalable neural search applications

Gensim

- Topic modelling, document indexing, and similarity retrieval with large corpora

NLTK (Natural Language Toolkit)

- Platform for building Python programs to work with human language data

TextBlob

- A simple API for common NLP tasks
- Part-of-speech tagging, noun phrase extraction, sentiment analysis, classification, translation, etc.

CoreNLP

- Group of NLP Programs
- Tokenization, part-of-speech tagging, lemmatization, etc

Polyglot

- Perform different NLP operations

Scikit-learn

- Intuitive class methods and numerous algorithms
- To build machine learning models

Pattern

- Implementing Natural Language processing tasks
- Text Mining, NLP, and Machine Learning

NLP Sentiment Analysis and Classification
with TensorFlow

Data Pre-processing

- **Download the dataset**
  - From Kaggle
    - https://www.kaggle.com/datasets/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews
  - Place it in the project's home directory

- **Read the dataset**
  - Convert the sentiment column
    - To numeric values
    - For binary classification

    - Use np.where()
      - 'positive' sentiment = 1
      - 'negative' sentiment = 0

- **Convert the labels and reviews to NumPy arrays**
  - Pre-processing methods favour arrays
    - Over Pandas series

Use pandas to load the IMDB review dataset

```python
reviews = pd.read_csv('IMDB Dataset.csv')
reviews.head()
```

✓ 0.5s

	review	sentiment
0	One of the other reviewers has mentioned that ...	positive
1	A wonderful little production. The...	positive
2	I thought this was a wonderful way to spend ti...	positive
3	Basically there's a family where a little boy ...	negative
4	Petter Mattei's "Love in the Time of Money" is...	positive

Convert the sentiment column to numeric values for binary classification

Regard 'positive' sentiment as 1 and 'negative' sentiment as 0 using np.where()

```python
# Regard 'positive' sentiment as 1 and 'negative' sentiment as 0
reviews['sentiment'] = np.where(reviews['sentiment'] == 'positive', 1, 0)
```

✓ 0.0s

Convert the labels and reviews(sentences) to NumPy arrays

Note: Pre-processing methods favor arrays over pandas series

```python
# Convert the labels and reviews(sentences) to NumPy arrays
sentences = reviews['review'].to_numpy()
labels = reviews['sentiment'].to_numpy()
```

✓ 0.0s

BATH SPA UNIVERSITY

Data Pre-processing

- **Train/Test Split**

- Split the dataset (train/test split) before any pre-processing
- 75:25 split

- Dataset is 50,000 reviews
  - Training model using 37500 reviews
  - Testing model accuracy using the unseen 12500 reviews

Split the dataset into training and test instances before any pre-processing

Use a 75:25 split for training and testing data, respectively

Dataset is 50,000 reviews

Training LSTM model using 37500 reviews
Testing model accuracy using the unseen 12500 reviews

```python
X_train, X_test, y_train, y_test = train_test_split(sentences, labels, test_size=0.25)
print("Training Data Input Shape: ", X_train.shape)
print("Training Data Output Shape: ", y_train.shape)
print("Testing Data Input Shape: ", X_test.shape)
print("Testing Data Output Shape: ", y_test.shape)
```

✓ 0.0s

```
Training Data Input Shape:  (37500,)
Training Data Output Shape:  (37500,)
Testing Data Input Shape:  (12500,)
Testing Data Output Shape:  (12500,)
```

BATH SPA UNIVERSITY

Data Pre-processing

Tokenisation on the entire text corpus

- Includes all the training data reviews

Convert textual data

- Reviews
  - Into numeric values
  - To build a mathematical model

- Specify **vocabulary size**
  - Tokenisation of training data
- Consider the **first 10000 words**
  - Based on frequency
    - In the training data
- Specify **oov_tok**
  - As <OOV>
  - Replaces any unknown word in the text corpus

```
# Set the vocabulary size to 10000
vocab_size = 10000
# Set the out-of-vocabulary token to "<OOV>"
oov_tok = "<OOV>"
# Initialise the tokenizer with the specified vocabulary size and OOV token
tokenizer = Tokenizer(num_words=vocab_size, oov_token=oov_tok)
```

Data Pre-processing

Tokenise sentences

- Into a set of **individual words** during tokenisation

- Calculate statistical features for each word
- **word_counts**
  - Dictionary of words with word count in the entire text corpus
- **word_docs**
  - Dictionary of words depicting the number of documents in the text corpus containing a specific word
- **word_index**
  - A unique index assigned to a dictionary of words
- **document_count**
  - Number of documents used for fitting the tokenizer

```python
# Fit the tokenizer on the training data
tokenizer.fit_on_texts(X_train)

# Print the number of documents used for fitting the tokenizer
print("Number of Documents: ", tokenizer.document_count)

# Print the number of words in the tokenizer
print("Number of Words: ", tokenizer.num_words)
```

✓ 4.9s

```
Number of Documents:  37500
Number of Words:  10000
```

BATH SPA UNIVERSITY

Data Pre-processing

Hyperparameters for tokeniser

- Fit the hyperparameters for Tokenizer() on the training data
  - fit_on_texts()

- Visualise:
  - the count of each word in the overall dictionary
  - The number of documents containing a specific word

- Convert each textual review into a numerical sequence using the fitted tokenizer

```
# Fit the tokenizer on the training data
tokenizer.fit_on_texts(X_train)

# Print the number of documents used for fitting the tokenizer
print("Number of Documents: ", tokenizer.document_count)

# Print the number of words in the tokenizer
print("Number of Words: ", tokenizer.num_words)
```
✓ 4.9s

```
Number of Documents:  37500
Number of Words:  10000
```

```
# Print the word counts in the tokenizer
tokenizer.word_counts
```
✓ 0.0s

```
OrderedDict([('once', 3425),
             ('upon', 1354),
             ('a', 242241),
             ('time', 18773),
             ('there', 23618),
             ('was', 71739),
             ('science', 790),
             ('fiction', 727),
             ('author', 331),
             ('named', 1151),
             ('h', 344),
             ('beam', 25),
             ('piper', 108),
             ('who', 30325),
             ('wrote', 821),
             ('classic', 2652),
             ('book', 3474),
             ('little', 9190),
             ('fuzzy', 83),
             ('which', 17549),
             ('about', 25496),
             ('man', 8248),
             ('discovering', 129),
             ('race', 589),
             ('of', 216196),
...
             ('heels', 87),
             ('private', 397),
             ('detective', 626),
             ('yarn', 48),
             ...])
```

```
# Print the word documents in the tokenizer
tokenizer.word_docs
```
✓ 0.0s

```
defaultdict(int,
            {'where': 7088,
             'man': 5707,
             'about': 15482,
             "who's": 895,
             'ewoks': 21,
             'adorable': 137,
             'for': 26690,
             'fuzzy': 78,
             'is': 33473,
             'race': 439,
             'to': 35195,
             'org': 17,
             'this': 33988,
             "today's": 360,
             'free': 946,
             'blatant': 156,
             'science': 600,
             'before': 5347,
             'died': 712,
             'mr': 1424,
             'and': 36218,
             'project': 669,
             'upon': 1232,
             'take': 4453,
...
             'priceless': 134,
             'mark': 793,
             'strongest': 106,
             'producing': 164,
             ...})
```

BATH SPA UNIVERSITY

Data Pre-processing

Convert training data reviews

- Convert each review in the training data
  - Into a numerical sequence
    - For further training purposes

- Note
  - Each review has different lengths of words
    - Will produce diverse numeric sequence lengths

```
# Convert the tokenised training data into sequences
train_sequences = tokenizer.texts_to_sequences(X_train)

# Print the first sequence
print(train_sequences[0])
```

✓ 3.1s

```
[281, 684, 4, 56, 47, 14, 4, 1092, 1166, 2266, 792, 2194, 1, 5385, 37, 1060, 4, 361, 278, 792,
```

Data Pre-processing

Limit Sequence Lengths
  - To a constant value for each review

- Set a nominal **sequence length**
  - 200 for each review

- **Truncate** numerical sequences
  - Lengths greater than 200

- **Pad** sequences
  - Lengths smaller than 200
    - With zeros

- Set the **sequence padding** for numerical sequences
  - Of textual reviews

- **Repeat** the same pre-processing steps
  - For **test data**
    - after training data complete

- **Complete pre-processing of the textual reviews**
  - Tokenisation, sequence conversion, and padding

```
# Convert the tokenised training data into sequences
train_sequences = tokenizer.texts_to_sequences(X_train)

# Print the first sequence
print(train_sequences[0])
```

✓ 3.1s

```
[281, 684, 4, 56, 47, 14, 4, 1092, 1166, 2266, 792, 2194, 1, 5385, 37, 1060, 4, 361, 278, 792, 120, 6531, 61, 14,
```

```
sequence_length = 200
train_padded = pad_sequences(train_sequences, maxlen=sequence_length, padding='post', truncating='post')
```

✓ 0.3s

```
# Convert the tokenised test data into sequences
test_sequences = tokenizer.texts_to_sequences(X_test)

# Pad the sequences to ensure uniform length, truncating longer ones and padding shorter ones with zeros
test_padded = pad_sequences(test_sequences, maxlen=sequence_length, padding='post', truncating='post')
```

Building an RNN using TensorFlow

Initiate a new Sequential model

- This model serves as a linear stack of layers
  - In the neural network
- Each layers output
  - Is the input for the next layer
  - Last layer outputs
    - Prediction label
- Use this model to embed the layers of the LSTM (Long Short-Term Memory) network
- The LSTM layers can be added to this model

```
# Convert the tokenised training data into sequences
train_sequences = tokenizer.texts_to_sequences(X_train)

# Print the first sequence
print(train_sequences[0])
```

✓ 3.1s

```
[281, 684, 4, 56, 47, 14, 4, 1092, 1166, 2266, 792, 2194, 1, 5385, 37, 1060, 4, 361, 278, 792,
```

Building an RNN using TensorFlow

Add an embedding layer to the model

- This layer converts each word
  - Into a dense vector
  - Of embedding dimensions
    - Hyperparameters of the layer

- Set **vocabulary size** and **sequence length**
  - For each review

- **Add a Bidirectional() layer**

- **Add a LSTM layer to the model**
  - Set a unit size in the LSTM layer

```python
# Set the embedding dimension to 16
embedding_dim = 16
# Add an Embedding layer to the model
model.add(Embedding(vocab_size, embedding_dim, input_length=sequence_length))

# Set the LSTM output to 32
lstm_out = 32
# Add a Bidirectional LSTM layer to the model
model.add(Bidirectional(LSTM(lstm_out)))
# Add two Dense layers to the model with 'relu' and 'sigmoid' activation functions respectively
model.add(Dense(10, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
# Compile the model with binary crossentropy loss function, adam optimizer, and accuracy metrics
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
# Print a summary of the model
model.summary()
```

✓ 0.6s

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 200, 16)           160000

 bidirectional (Bidirection  (None, 64)                12544
 al)

 dense (Dense)               (None, 10)                650

 dense_1 (Dense)             (None, 1)                 11

=================================================================
Total params: 173205 (676.58 KB)
Trainable params: 173205 (676.58 KB)
Non-trainable params: 0 (0.00 Byte)
```

Note: Bidirectional LSTM remembers output from:
- Past to future
- And from future to past
- More robust models for time series analysis

Building an RNN using TensorFlow

- **Add two Dense layers to the model**
  - Specify activation functions
- **Add a fully connected layer**
  - 10 units
  - ‘relu’ activation
- **Add an output layer**
  - 1 unit
  - ‘sigmoid’ activation
- output layer
  - Outputs probability input belongs to
    - 1 (positive) using the sigmoid filter

```python
# Set the embedding dimension to 16
embedding_dim = 16
# Add an Embedding layer to the model
model.add(Embedding(vocab_size, embedding_dim, input_length=sequence_length))

# Set the LSTM output to 32
lstm_out = 32
# Add a Bidirectional LSTM layer to the model
model.add(Bidirectional(LSTM(lstm_out)))
# Add two Dense layers to the model with 'relu' and 'sigmoid' activation functions respectively
model.add(Dense(10, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
# Compile the model with binary crossentropy loss function, adam optimizer, and accuracy metrics
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
# Print a summary of the model
model.summary()
```

✓ 0.6s

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 200, 16)           160000

 bidirectional (Bidirection  (None, 64)                12544
 al)

 dense (Dense)               (None, 10)                650

 dense_1 (Dense)             (None, 1)                 11

=================================================================
Total params: 173205 (676.58 KB)
Trainable params: 173205 (676.58 KB)
Non-trainable params: 0 (0.00 Byte)
```

Building an RNN using TensorFlow

- **Compile the model**
  - Optimises the binary_crossentropy
    - During training
- ‘adam’ **optimiser**
  - Minimises loss value
    - By tweaking the weights
    - During the training phase
  - Tries to find the global minima
    - For the loss value
  - Across all the local minima
- **‘accuracy’** of the model
  - Reported for each training batch/epoch
    - Gauge the convergence
      - Of the neural network
- **Visualise** the summary of the LSTM model

```
# Set the embedding dimension to 16
embedding_dim = 16
# Add an Embedding layer to the model
model.add(Embedding(vocab_size, embedding_dim, input_length=sequence_length))

# Set the LSTM output to 32
lstm_out = 32
# Add a Bidirectional LSTM layer to the model
model.add(Bidirectional(LSTM(lstm_out)))
# Add two Dense layers to the model with 'relu' and 'sigmoid' activation functions respectively
model.add(Dense(10, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
# Compile the model with binary crossentropy loss function, adam optimizer, and accuracy metrics
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
# Print a summary of the model
model.summary()
```

✓ 0.6s

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 200, 16)           160000

 bidirectional (Bidirection  (None, 64)                12544
 al)

 dense (Dense)               (None, 10)                650

 dense_1 (Dense)             (None, 1)                 11

=================================================================
Total params: 173205 (676.58 KB)
Trainable params: 173205 (676.58 KB)
Non-trainable params: 0 (0.00 Byte)
```

BATH SPA UNIVERSITY

Building an RNN using TensorFlow – EarlyStopping()

- **EarlyStopping()**
  - Halts model training
    - After the model fails to minimise
    - The validation loss value
      - After a set number of epochs

- Helps **avoid overfitting** the model
  - On the training data

- **ModelCheckpoint()**
  - Monitor the loss after each epoch
  - Save the best model
    - In terms of validation loss

```python
# Set the checkpoint file path to the current working directory
checkpoint_filepath = os.getcwd()

# Create a ModelCheckpoint callback that saves the best model in terms of validation loss
model_checkpoint_callback = ModelCheckpoint(filepath=checkpoint_filepath,
                                            save_weights_only=False,
                                            monitor='val_loss',
                                            mode='min',
                                            save_best_only=True)

# Create a list of callbacks including EarlyStopping and ModelCheckpoint
callbacks = [EarlyStopping(patience=2), model_checkpoint_callback]
```

BATH SPA UNIVERSITY

Building an RNN using TensorFlow – Fit Model

- **Fit the model**
  - Set the number of epochs
    - Network trained this amount
  - Number of epochs needed
    - Often unknown
    - Requires an educated guess
      - And tweaking
  - Maximum of 10 epochs
    - In example

- **Set the validation data**
  - Monitor the loss on the validation dataset

- **Halt the training**
  - If the validation loss is not minimised
    - For two consecutive epochs
  - Specified in the callback

  - Model training may halt before reaching 10 epochs
    - If the validation loss does not improve

```
# Fit the model on the training data for a maximum of 10 epochs
# monitoring the loss on the validation dataset
history = model.fit(train_padded,
                    y_train, epochs=10,
                    validation_data=(test_padded,
                                     y_test),
                    callbacks=callbacks)
```

BATH SPA UNIVERSITY

Building an RNN using TensorFlow – Model Accuracy

- Model training halted
  - After 5 epochs
    - Loss did not improve
      - After Epoch 3
- Model parameters
  - Saved in 'history' variable
- Achieved 86% validation accuracy
  - On the IMDB review dataset
  - By training a simple bidirectional LSTM network
- Accuracy could be improved
  - Using back-to-back LSTM layers
  - Or using increased word dictionary

```
history = model.fit(train_padded,
                    y_train,
                    epochs=10,
                    validation_data=(test_padded,
                                     y_test),
                    callbacks=callbacks)
```

✓ 6m 44.8s

```
Epoch 1/10
1172/1172 [==============================] - 91s 74ms/step - loss: 0.6494 - accuracy: 0.6212 - val_loss: 0.5022 - val_accuracy: 0.7782
Epoch 2/10
1172/1172 [==============================] - 85s 72ms/step - loss: 0.4619 - accuracy: 0.7881 - val_loss: 0.3759 - val_accuracy: 0.8459
Epoch 3/10
1172/1172 [==============================] - 95s 81ms/step - loss: 0.3061 - accuracy: 0.8778 - val_loss: 0.3214 - val_accuracy: 0.8628
Epoch 4/10
1172/1172 [==============================] - 67s 57ms/step - loss: 0.2497 - accuracy: 0.9039 - val_loss: 0.3566 - val_accuracy: 0.8443
Epoch 5/10
1172/1172 [==============================] - 67s 57ms/step - loss: 0.2222 - accuracy: 0.9162 - val_loss: 0.3429 - val_accuracy: 0.8664
```

```
metrics_df = pd.DataFrame(history.history)
print(metrics_df)
```

✓ 0.0s

	loss	accuracy	val_loss	val_accuracy
0	0.649434	0.621227	0.502158	0.77824
1	0.461919	0.788107	0.375894	0.84592
2	0.306105	0.877813	0.321435	0.86280
3	0.249678	0.903947	0.356642	0.84432
4	0.222168	0.916160	0.342907	0.86640

BATH SPA UNIVERSITY

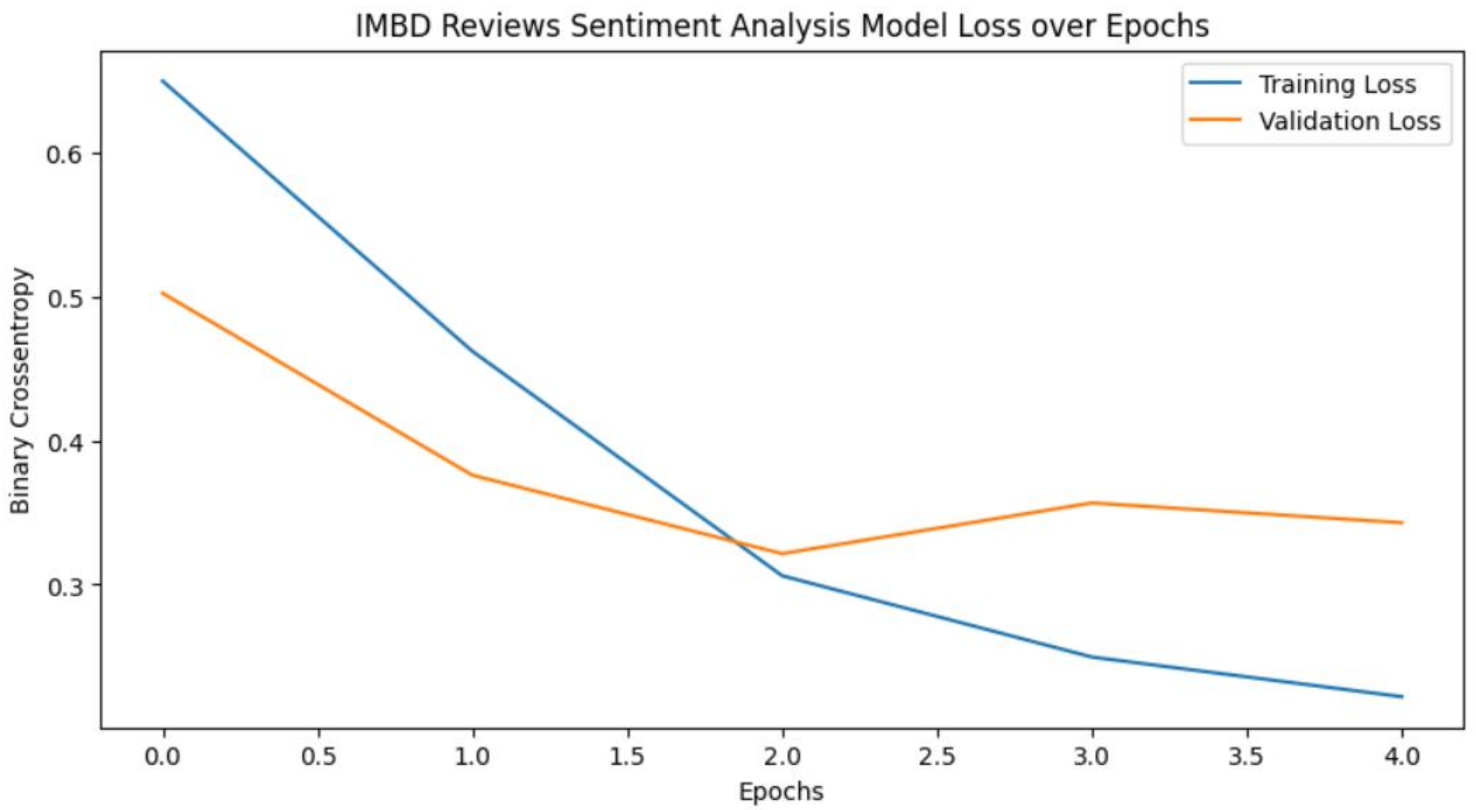
Building an RNN using TensorFlow – Model Loss

- Visualise the

- Training/testing data
  - Over the number of epochs
  - Using matplotlib

```
plt.figure(figsize=(10,5))
plt.plot(metrics_df.index, metrics_df.loss)
plt.plot(metrics_df.index, metrics_df.val_loss)
plt.title('IMBD Reviews Sentiment Analysis Model Loss over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Binary Crossentropy')
plt.legend(['Training Loss', 'Validation Loss'])
plt.show()
```

✓ 0.2s

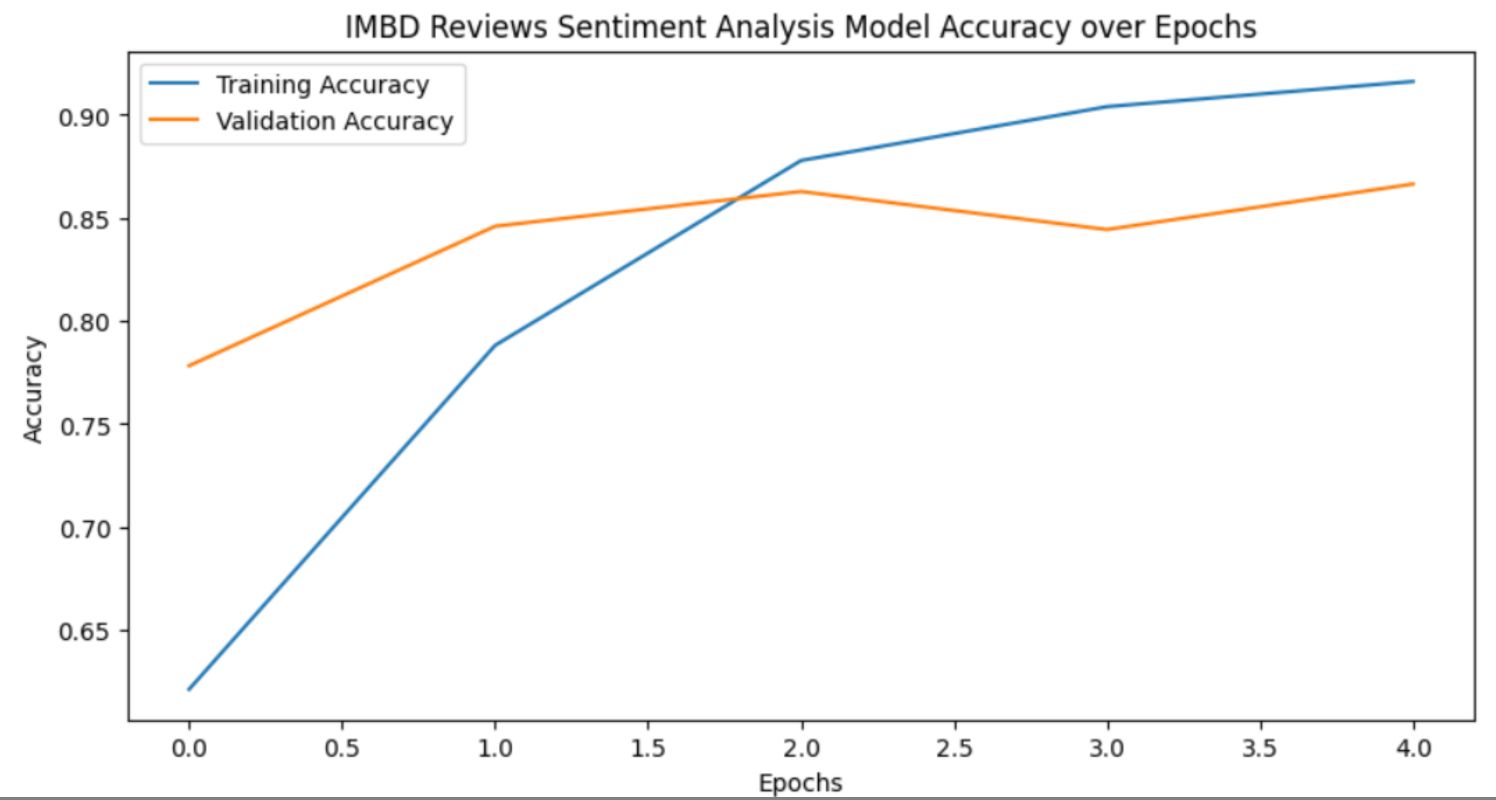


Building an RNN using TensorFlow – Model Accuracy

- Visualise the accuracy
- Training/testing data
  - Over the number of epochs
  - Using matplotlib

```python
plt.figure(figsize=(10,5))
plt.plot(metrics_df.index, metrics_df.accuracy)
plt.plot(metrics_df.index, metrics_df.val_accuracy)
plt.title('IMBD Reviews Sentiment Analysis Model Accuracy over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend(['Training Accuracy', 'Validation Accuracy'])
plt.show()
```

✓ 0.1s



BATH SPA UNIVERSITY

Session Review

- Understand how machine learning models handle non-numeric data
- Understand which scenarios these might be applied to
- Work through a practical example