19CSE456 Neural Network and Deep Learning Laboratory

List of Experiments

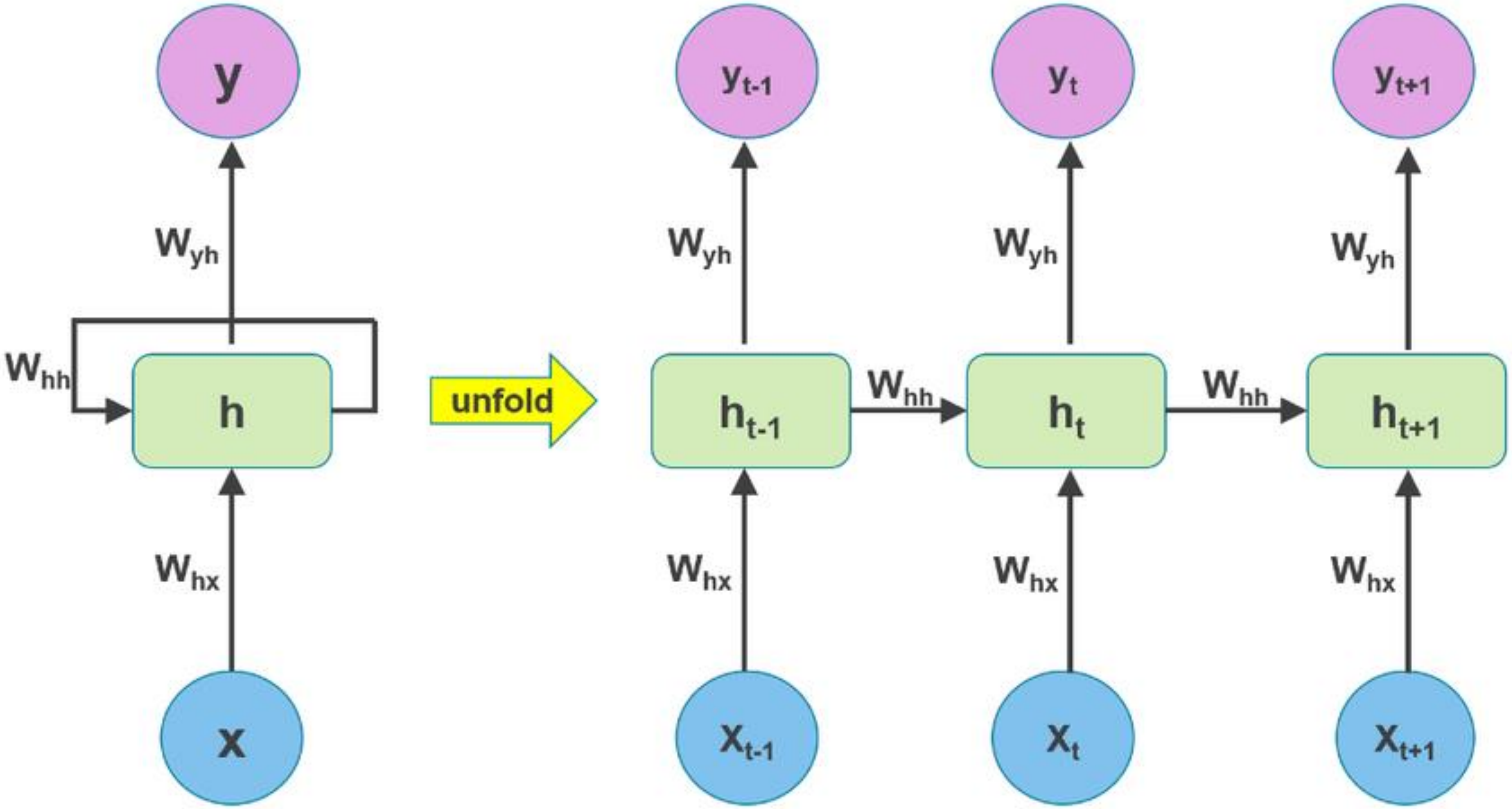
Week #	Experiment Title
1	Introduction to the lab and Implementation of a simple Perceptron (Hardcoding)
2	Implementation of Perceptron for Logic Gates (Hardcoding, Sklearn, TF)
3	Implementation of Multilayer Perceptron for XOR Gate and other classification problems with ML toy datasets (Hardcoding & TF)
4	Implementation of MLP for Image Classification with MNIST dataset (Hardcoding & TF)
5	Activation Functions, Loss Functions, Optimizers (Hardcoding & TF)
6	Lab Evaluation 1 (based on topics covered from w1 to w5)
7	Convolution Neural Networks for Toy Datasets (MNIST & CIFAR)
8	Convolution Neural Networks for Image Classification (Oxford Pets, Tiny ImageNet, etc.)
9	Recurrent Neural Networks for Sentiment Analysis with IMDB Movie Reviews
10	Long Short Term Memory for Stock Prices (Yahoo Finance API)

List of Experiments contd.

Week #	Experiment Title
11	Implementation of Autoencoders and Denoising Autoencoders (MNIST/CIFAR)
12	Boltzmann Machines (MNIST/CIFAR)
13	Restricted Boltzmann Machines (MNIST/CIFAR)
14	Hopfield Neural Networks (MNIST/CIFAR)
15	Lab Evaluation 2 (based on CNN, RNN, LSTM, and AEs)
16	Case Study Review (Phase 1)
17	Case Study Review (Phase 1)

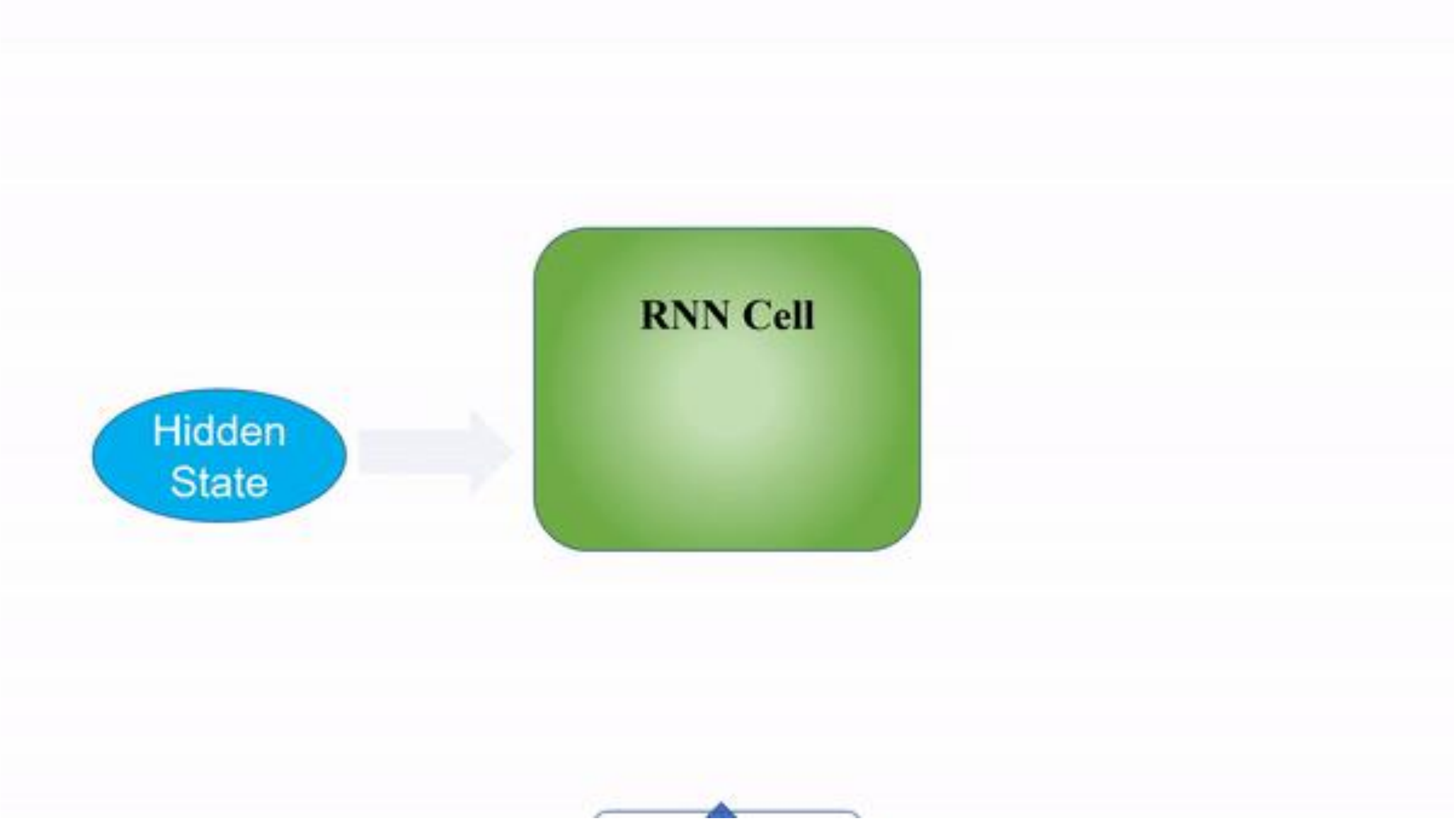
Recurrent Neural Network

- A type of artificial neural network designed for sequential data processing
- Unlike traditional feedforward neural networks, RNNs have connections that form directed cycles, allowing them to maintain a "memory" of previous inputs



Recurrent Neural Network

- A type of artificial neural network designed for sequential data processing
- Unlike traditional feedforward neural networks, RNNs have connections that form directed cycles, allowing them to maintain a "memory" of previous inputs



Sentiment Labelled Sentences Dataset

This a collection of sentences labeled with positive or negative sentiment. It was created for the paper "From Group to Individual Labels using Deep Features" by Dimitrios Kotzias, Misha Denil, Nando de Freitas, and Padhraic Smyth, published in 2015

```
So there is no way for me to plug it in here in the US unless I go by a converter.	0
Good case, Excellent value.	1
Great for the jawbone.	1
Tied to charger for conversations lasting more than 45 minutes.MAJOR PROBLEMS!!	0
The mic is great.	1
I have to jiggle the plug to get it to line up right to get decent volume.	0
If you have several dozen or several hundred contacts, then imagine the fun of sending each of them one by one.	0
If you are Razr owner...you must have this!	1
Needless to say, I wasted my money.	0
What a waste of money and time!.	0
And the sound quality is great.	1
He was very impressed when going from the original battery to the extended battery.	1
If the two were seperated by a mere 5+ ft I started to notice excessive static and garbled sound from the headset.	0
```

- The sentences come from three different websites: IMDb, Amazon, and Yelp
- Each source provides 500 positive and 500 negative sentences, making a total of 3,000 sentences

Recurrent Neural Network

Load and Preprocess the Data → Text Preprocessing → Model Building → Model Training → Model Evaluation

```
# 1. Data Loading and Preprocessing
def load_data(filepath):
   # Read the data - tab separated
   data = pd.read_csv(filepath, sep='\t', names=['text', 'label'])
   return data

# Load all three datasets and combine them
amazon = load_data('/content/sample_data/amazon_cells_labelled.txt')
yelp = load_data('/content/sample_data/yelp_labelled.txt')
imdb = load_data('/content/sample_data/imdb_labelled.txt')
df = pd.concat([amazon, yelp, imdb], ignore_index=True)
df.head()
```

This code snippet is for loading and preprocessing text data from three different datasets (Amazon, Yelp, and IMDb)

Recurrent Neural Network

Load and Preprocess the Data → **Text Preprocessing** → Model Building → Model Training → Model Evaluation

```python
# 2. Text Preprocessing
max_words = 5000
max_len = 100
tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(df['text'])
sequences = tokenizer.texts_to_sequences(df['text'])
X = pad_sequences(sequences, maxlen=max_len)
y = df['label'].values
# Print preprocessing info
print("\nPreprocessing Info:")
print(f"Vocabulary size: {len(tokenizer.word_index) + 1}")
print(f"Sequence length: {max_len}")
print(f"Total samples: {len(X)}")
```

The maximum number of words to keep based on word frequency. Only the top

The maximum length of sequences.
Longer lengths – truncation
Shorter lengths - padding

This creates an index of words and their frequencies

The text data is converted into sequences of integers

The sequences are padded (or truncated) to ensure they all have the same length

Visualizing the Tokenizer's Output

```
Word Index:
the: 1
and: 2
i: 3
a: 4
is: 5
it: 6
to: 7
this: 8
of: 9
was: 10
in: 11
for: 12
not: 13
that: 14
with: 15
my: 16
very: 17
good: 18
on: 19
great: 20
you: 21
but: 22
have: 23
movie: 24
are: 25
as: 26
so: 27
phone: 28
film: 29
be: 30
all: 31
```

```
really: 49
there: 50
they: 51
we: 52
well: 53
out: 54
has: 55
would: 56
about: 57
no: 58
or: 59
your: 60
only: 61
by: 62
best: 63
don't: 64
even: 65
here: 66
ever: 67
up: 68
also: 69
will: 70
back: 71
me: 72
when: 73
more: 74
than: 75
quality: 76
go: 77
what: 78
love: 79
he: 80
i've: 81
```

```
bagels: 1799
dine: 1800
rarely: 1801
curry: 1802
bathrooms: 1803
decorated: 1804
middle: 1805
greeted: 1806
highlights: 1807
joint: 1808
caught: 1809
judging: 1810
overcooked: 1811
charcoal: 1812
decided: 1813
dirt: 1814
gyros: 1815
valley: 1816
readers: 1817
bowl: 1818
disrespected: 1819
lived: 1820
stepped: 1821
gold: 1822
puree: 1823
corn: 1824
bug: 1825
we're: 1826
friend: 1827
shower: 1828
bisque: 1829
filet: 1830
pepper: 1831
```

```
portrayals: 4051
detailing: 4052
loyalty: 4053
treachery: 4054
melville: 4055
manages: 4056
transcend: 4057
limitations: 4058
indie: 4059
continually: 4060
subverting: 4061
emerge: 4062
intense: 4063
crocdodile: 4064
believed: 4065
crocs: 4066
swamp: 4067
christopher: 4068
eccleston: 4069
tardis: 4070
continuation: 4071
succeeded: 4072
here's: 4073
pi: 4074
witticisms: 4075
bob: 4076
rise: 4077
finale: 4078
kieslowski: 4079
amaze: 4080
colours: 4081
flag: 4082
connections: 4083
```

```
frost: 5238
bonuses: 5239
fest: 5240
spoiled: 5241
brat: 5242
babysitting: 5243
sundays: 5244
march: 5245
judith: 5246
cutie: 5247
confidence: 5248
riot: 5249
hugo: 5250
weaving: 5251
obsessed: 5252
gay: 5253
estate: 5254
salesman: 5255
clients': 5256
houses: 5257
trysts: 5258
flaming: 5259
darren: 5260
hollander: 5261
flowed: 5262
bonding: 5263
hoot: 5264
n: 5265
jessice: 5266
clothes: 5267
virtue: 5268
regrettable: 5269
exceptionally: 5270
one's: 5271
```

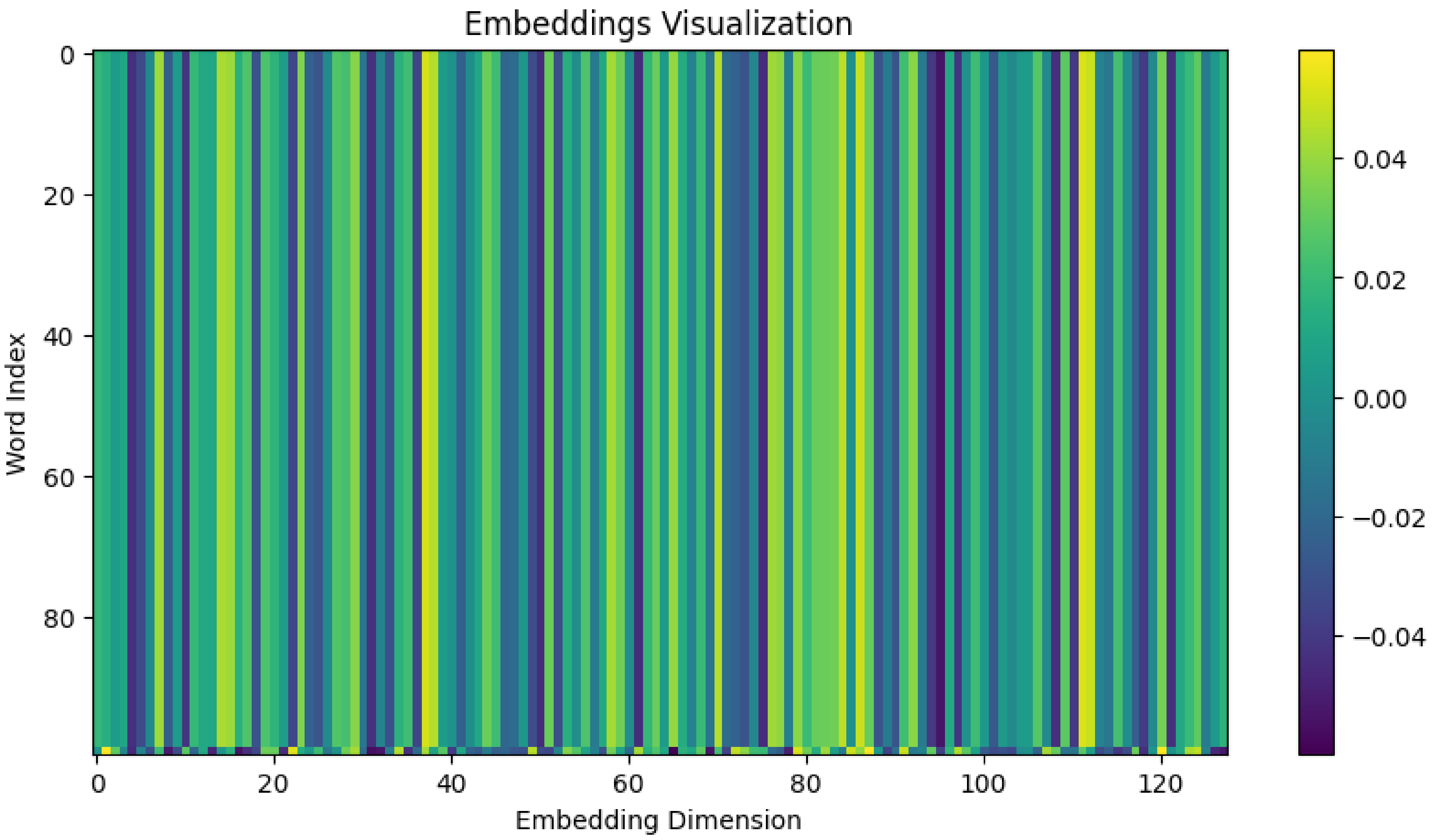
Recurrent Neural Network

Load and Preprocess the Data → Text Preprocessing → **Model Building** → Model Training → Model Evaluation

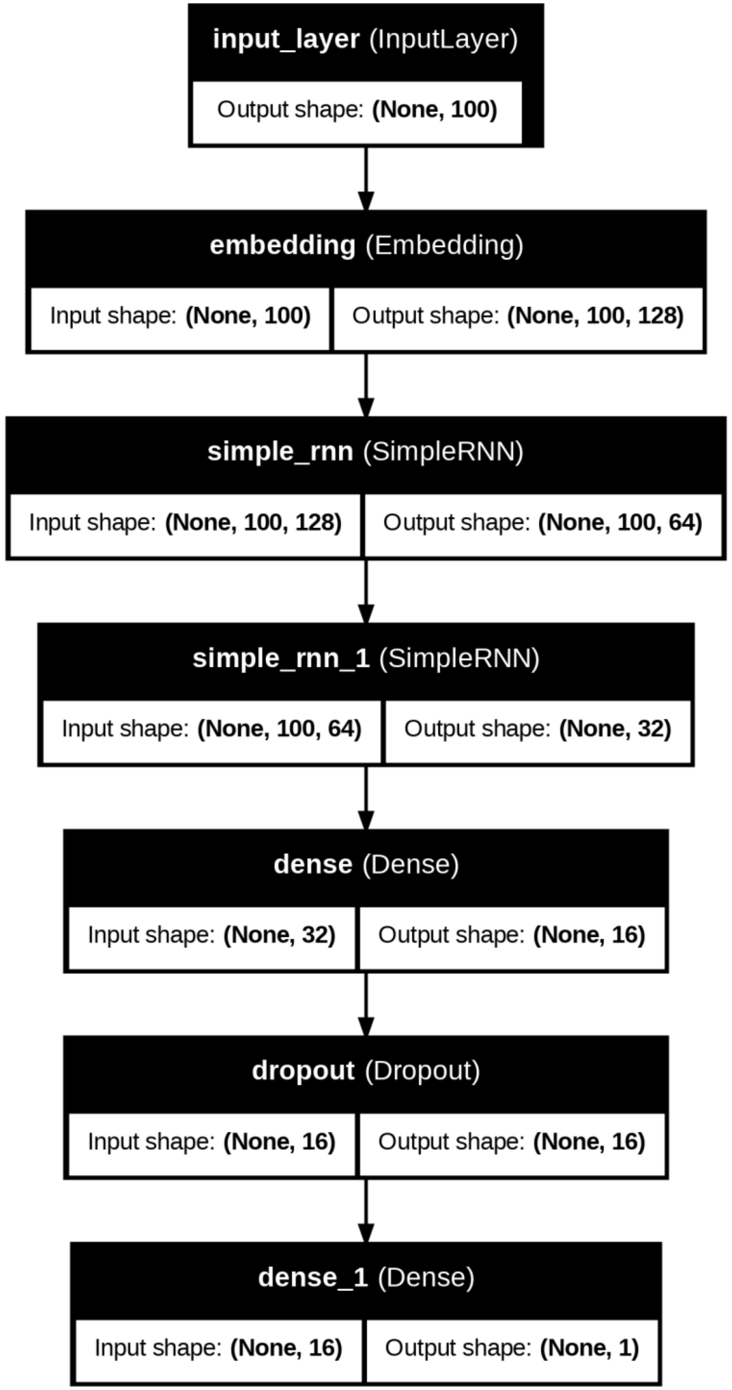
```
# 3. Build the RNN Model using functional API
vocab_size = min(len(tokenizer.word_index) + 1, max_words)

# Create model with explicit input shape
inputs = tf.keras.Input(shape=(max_len,))
x = tf.keras.layers.Embedding(vocab_size, 128)(inputs)
x = tf.keras.layers.SimpleRNN(64, return_sequences=True)(x)
x = tf.keras.layers.SimpleRNN(32)(x)
x = tf.keras.layers.Dense(16, activation='relu')(x)
x = tf.keras.layers.Dropout(0.5)(x)
outputs = tf.keras.layers.Dense(1, activation='sigmoid')(x)

model = tf.keras.Model(inputs=inputs, outputs=outputs)
```

- The vocab_size is set to the smaller of len(tokenizer.word_index) + 1 and max_words
- an input layer with a shape corresponding to max_len
- This layer maps the input integers to dense vectors of fixed size (128 dimensions)
- This layer is a Simple Recurrent Neural Network (RNN) with 64 units
- This is fully connected (Dense) layer with 16 units
- This fully connected (Dense) layer has a single unit

Visualizing Word Embedding



Model Visualization



Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 100)	0
embedding (Embedding)	(None, 100, 128)	640,000
simple_rnn (SimpleRNN)	(None, 100, 64)	12,352
simple_rnn_1 (SimpleRNN)	(None, 32)	3,104
dense (Dense)	(None, 16)	528
dropout (Dropout)	(None, 16)	0
dense_1 (Dense)	(None, 1)	17

```
Total params: 656,001 (2.50 MB)
Trainable params: 656,001 (2.50 MB)
Non-trainable params: 0 (0.00 B)
```

Recurrent Neural Network

Load and Preprocess the Data → Text Preprocessing → **Model Building** → Model Training → Model Evaluation

```
# Compile the model
model.compile(optimizer='adam',
        loss='binary_crossentropy',
        metrics=['accuracy'])
```

This line tells the model to:

- Use the adam optimizer.
- Minimize the binary_crossentropy loss function during training.
- Track the accuracy metric during training and evaluation.

Recurrent Neural Network

Load and Preprocess the Data → Text Preprocessing → Model Building → **Model Training** → Model Evaluation

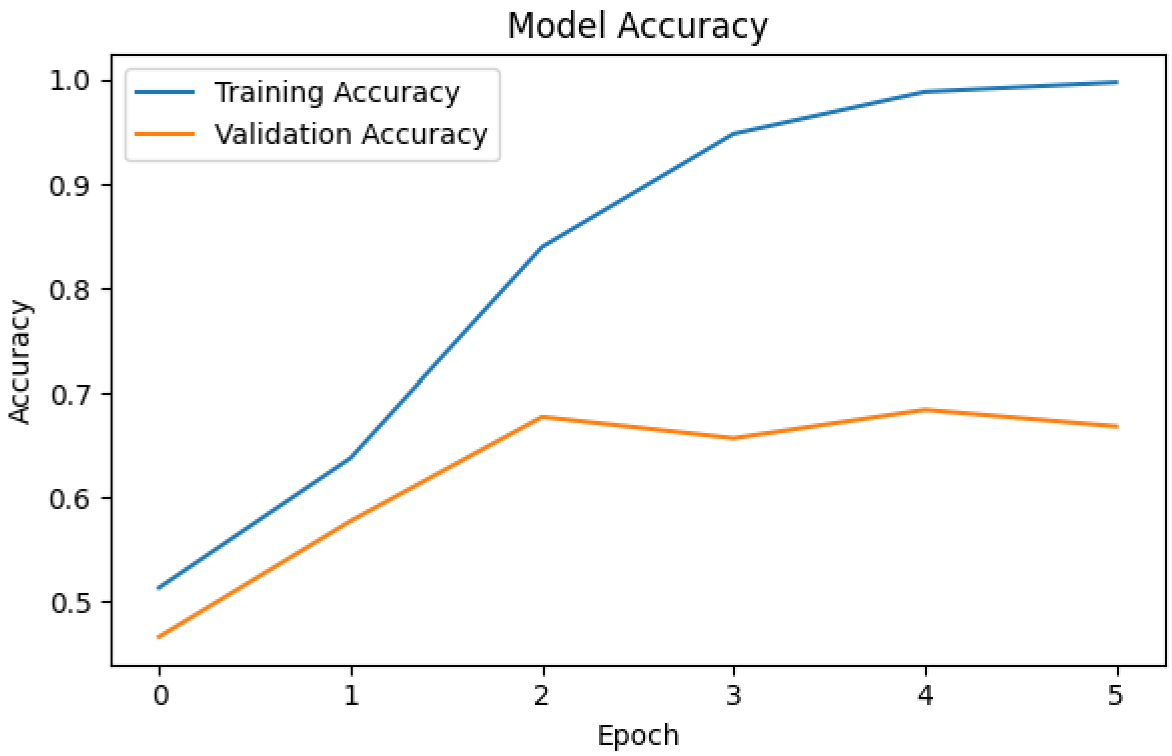
```
# 4. Train the Model
early_stopping = EarlyStopping(
    monitor='val_loss',
    patience=3,
    restore_best_weights=True
)

history = model.fit(
    X_train, y_train,
    epochs=10,
    batch_size=32,
    validation_split=0.2,
    callbacks=[early_stopping]
)
```

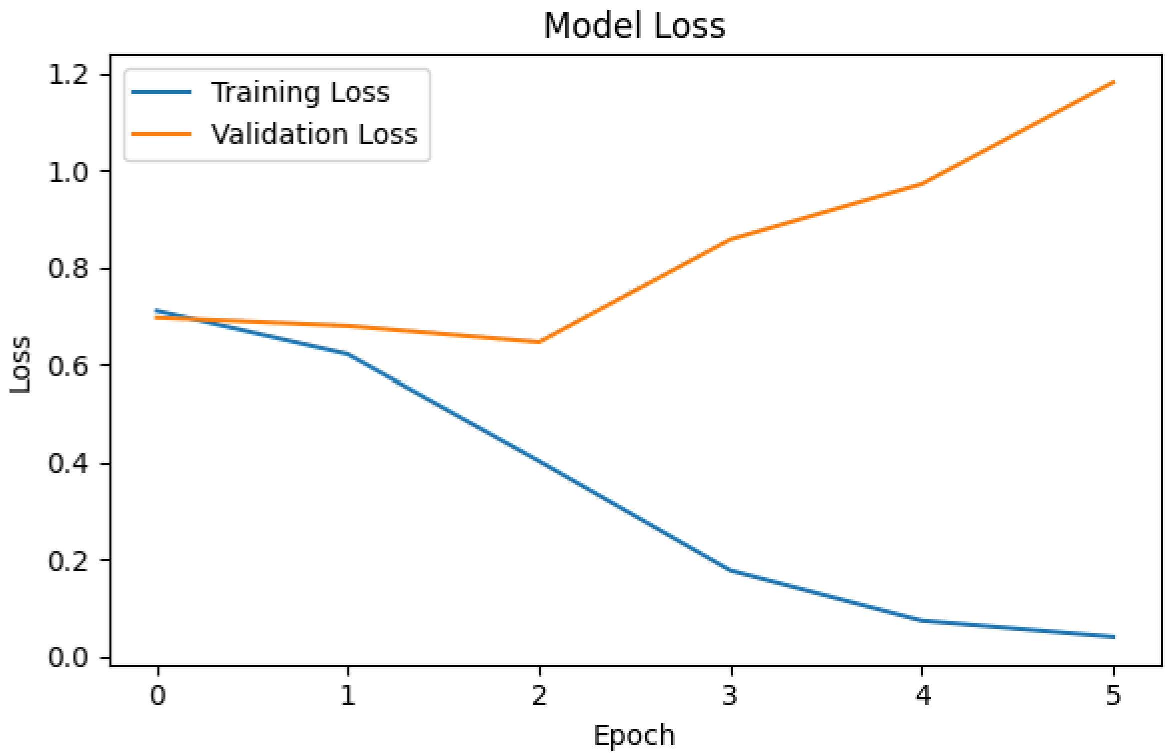
This is a callback function that monitors the performance of the model during training and stops training early if the performance on the validation set does not improve

The fit method returns a history object, which contains details about the training process, including the loss and accuracy on both the training and validation sets for each epoch.

Recurrent Neural Network


```
55/55 ━━━━━━━━━━━━━━━━━━━━ 1s 17ms/step - accuracy: 0.8122 - loss: 0.4432 - val_accuracy: 0.6773 - val_loss: 0.6456
Epoch 4/10
55/55 ━━━━━━━━━━━━━━━━━━━━ 1s 17ms/step - accuracy: 0.9482 - loss: 0.1869 - val_accuracy: 0.6568 - val_loss: 0.8572
Epoch 5/10
55/55 ━━━━━━━━━━━━━━━━━━━━ 1s 17ms/step - accuracy: 0.9884 - loss: 0.0798 - val_accuracy: 0.6841 - val_loss: 0.9715
Epoch 6/10
55/55 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.9964 - loss: 0.0450 - val_accuracy: 0.6682 - val_loss: 1.1810
```





Making Prediction using the Trained RNN

```
# Make Predictions
def predict_sentiment(text):
    sequence = tokenizer.texts_to_sequences([text])
    padded = pad_sequences(sequence, maxlen=max_len)
    prediction = model.predict(padded)[0][0]

    return {
        'text': text,
        'sentiment': 'Positive' if prediction > 0.5 else 'Negative',
        'confidence': float(prediction if prediction > 0.5 else 1 - prediction)
    }
```

- This line uses the trained model to predict the sentiment of the padded sequence.
- The model.predict method returns a prediction for each sample in the batch.

Week 8 Exercises

1.SMS Spam Detection using RNN

Objective: To build an end-to-end Recurrent Neural Network (RNN) model to classify SMS messages as "ham" (legitimate) or "spam.“

Dataset: The dataset used in this lab is the SMS Spam Collection dataset, which contains SMS messages labeled as either "ham" or "spam.“

Tasks:

1. Data Loading and Exploration
2. Text Preprocessing
3. Build the RNN Model
4. Train the Model
5. Evaluate the Model
6. Make Predictions