Deep Learning and TensorFlow 2

Machine Learning

Introduction to Deep Learning

- Supervised Learning with Deep Neural Networks

Input(x)	Output(y)	Application	Model
Home features	Price	Real Estate	Fully- Connected Neural Nets
Ad, user info	Click on ad?(0/1)	Online Advertising	Fully- Connected Neural Nets
Image	Object (1,…,1000)	Photo tagging	Convolutional Neural Nets
Audio	Text transcript	Speech recognition	Recurrent Neural Nets
Korean	English	Machine translation	Recurrent Neural Nets

https://cs230.stanford.edu/

Introduction to Deep Learning

- Supervised Learning with Deep Neural Networks

Structured Data

Size	#bedrooms	…	Price(1000$)
2104	3	…	400
1600	3	…	330
2400	3	…	369
…	…	…	…
3000	4	…	540

Unstructured Data

Image

Audio

This shirt is very flattering to all due to …

Text

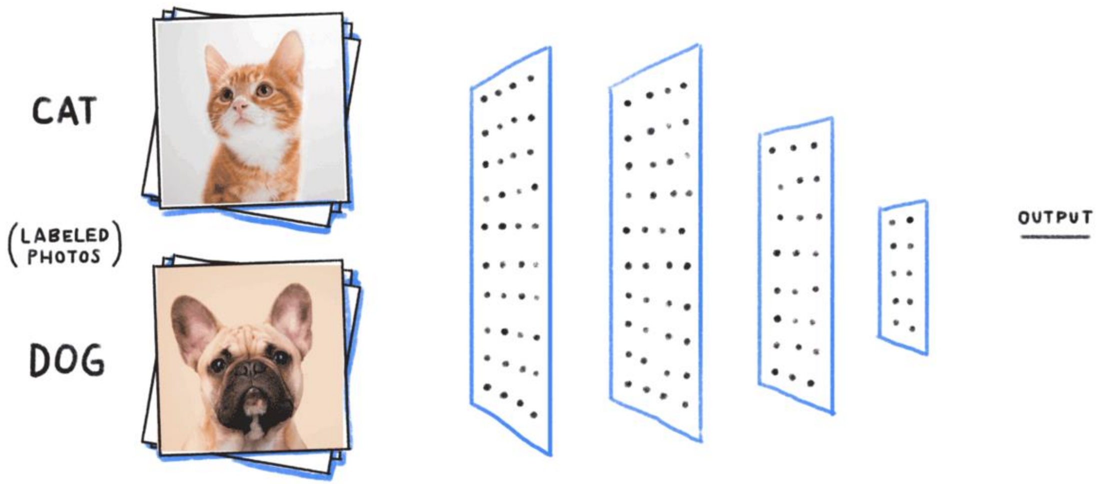
Video

Contents

- Convolutional Neural Networks
- Recurrent Neural Networks

Convolutional Neural Networks

- Convolutional Neural Networks : for Analyzing Visual Data
  - Smaller number of connection
  - Weight sharing
  - Detect features at different positions in a image



https://becominghuman.ai/building-an-image-classifier-using-deep-learning-in-python-totally-from-a-beginners-perspective-be8dbaf22dd8

Convolutional Neural Networks

- Image Classification with CNNs
  - Load Fashion-MNIST Dataset and Preprocessing

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt

fashion_mnist = tf.keras.datasets.fashion_mnist
(x_train, y_train),(x_test, y_test) = fashion_mnist.load_data()

x_train, x_test = x_train / 255.0, x_test / 255.0
num_classes = 10

x_train = x_train.reshape(x_train.shape[0], 28, 28, 1)
x_train.shape                    → (60000, 28, 28, 1)
x_test = x_test.reshape(x_test.shape[0], 28, 28, 1)
x_test.shape                     → (10000, 28, 28, 1)
y_train = y_train.reshape(y_train.shape[0], 1)
y_train.shape                    → (60000, 1)
y_test = y_test.reshape(y_test.shape[0], 1)
y_test.shape                     → (10000, 1)

y_train                          → array([[9],
                                          [0],
                                          [0],
                                          ...,
                                          [3],
                                          [0],
                                          [5]], dtype=uint8)
```

Convolutional Neural Networks

- Image Classification with CNNs
  - Build the CNN Model

```
base_model = tf.keras.models.Sequential([
     layers.Conv2D(32, kernel_size=(3, 3), padding = "same",
                             input_shape = (28, 28, 1),
                             activation="relu"),
     layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
     layers.MaxPool2D(pool_size=2),

     layers.Flatten(),
     layers.Dense(128, activation="relu"),
     layers.Dense(10, activation="softmax")
])
```

```
base_model.compile(loss='categorical_crossentropy',
                   optimizer='adam',
                   metrics=['accuracy'])
```

Convolutional Neural Networks

- Image Classification with CNNs
  - Train the model

```
base_history = base_model.fit(x_train, y_train, epochs=10,
validation_data=(X_test, y_test))
```

```
Epoch 1/10
60000/60000 [==============================] - 9s 151us/step - loss: 0.3607 - acc: 0.8718
Epoch 2/10
60000/60000 [==============================] - 9s 142us/step - loss: 0.2224 - acc: 0.9172
Epoch 3/10
60000/60000 [==============================] - 9s 143us/step - loss: 0.1684 - acc: 0.9377
Epoch 4/10
60000/60000 [==============================] - 8s 138us/step - loss: 0.1264 - acc: 0.9533
Epoch 5/10
60000/60000 [==============================] - 8s 139us/step - loss: 0.0928 - acc: 0.9657
Epoch 6/10
60000/60000 [==============================] - 8s 140us/step - loss: 0.0667 - acc: 0.9756
Epoch 7/10
60000/60000 [==============================] - 8s 138us/step - loss: 0.0494 - acc: 0.9822
Epoch 8/10
60000/60000 [==============================] - 8s 139us/step - loss: 0.0361 - acc: 0.9869
Epoch 9/10
60000/60000 [==============================] - 8s 139us/step - loss: 0.0288 - acc: 0.9896
Epoch 10/10
60000/60000 [==============================] - 8s 138us/step - loss: 0.0255 - acc: 0.9908

<tensorflow.python.keras.callbacks.History at 0x279dbfdacf8>
```

```
base_model.evaluate(x_train, y_train)
```

[0.019858361043675423, 0.9931833333333333]

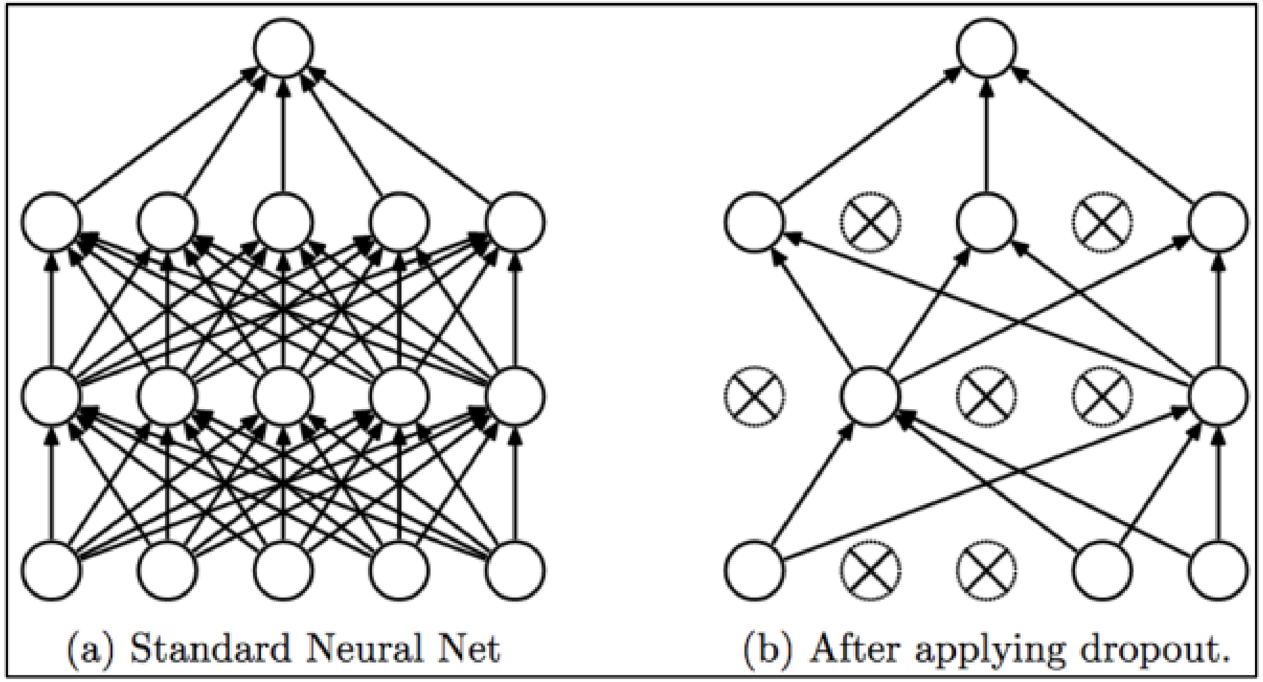
```
base_model.evaluate(x_test, y_test)
```

[0.42943426142530516, 0.9202]

loss *accuracy*

Convolutional Neural Networks

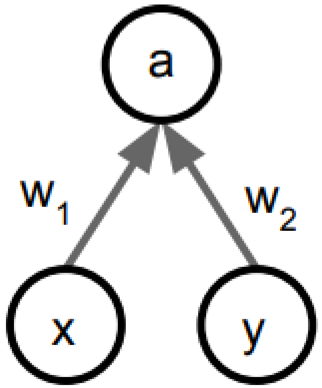
- Avoid the Overfitting, Dropout
  - In each forward pass, randomly set some neurons to zero
  - Probability of dropping is a hyperparameter; 0.5 is common



http://cs231n.stanford.edu/

Convolutional Neural Networks

- Avoid the Overfitting, Dropout
  - Consider a single neuron



  - Training phase
    - Dropout applying probability p (ex_ 0.5)

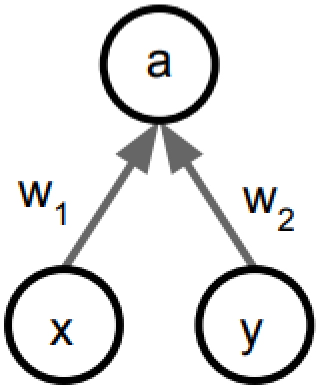
$$a = \frac{1}{4}(w_1x + w_2y) + \frac{1}{4}(w_1x + 0y) + \frac{1}{4}(0x + w_2y) + \frac{1}{4}(0x + 0y)$$
$$= \frac{1}{2}(w_1x + w_2y)$$

  - Test phase
    - No dropout

$$a = w_1x + w_2y$$

Convolutional Neural Networks

- Avoid the Overfitting, Dropout
  - Consider a single neuron



  - Training phase
    - Dropout applying probability p (ex_ 0.5)

$$a = \frac{1}{4}(w_1x + w_2y) + \frac{1}{4}(w_1x + 0y) + \frac{1}{4}(0x + w_2y) + \frac{1}{4}(0x + 0y)$$

$$= \frac{1}{2}(w_1x + w_2y)$$

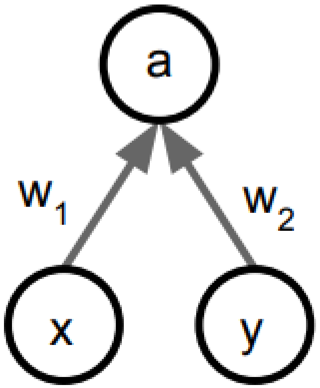
  - Test phase
    - No dropout

$$a = w_1x + w_2y \qquad \times p$$

  - So, we need to multiply p at test time for approximation!

Convolutional Neural Networks

- Avoid the Overfitting, Dropout
  - Consider a single neuron



  - Training phase
    - Dropout applying probability p (ex_ 0.5)

$$a = \frac{1}{4}(w_1 x + w_2 y) + \frac{1}{4}(w_1 x + 0y) + \frac{1}{4}(0x + w_2 y) + \frac{1}{4}(0x +$$

$$= \frac{1}{2}(w_1 x + w_2 y)$$

  - Test phase
    - No dropout

$$a = w_1 x + w_2 y$$

$\times 1/p$

  - But commonly, we multiply $1/p$ at training time for approximation!
  - This is called “Inverted Dropout”

Convolutional Neural Networks

- Image Classification with CNNs
  - The model with dropout regularization

```
dropout_model = tf.keras.models.Sequential([
     layers.Conv2D(32, kernel_size=(3, 3), padding = "same",
                           input_shape = (28, 28, 1),
                           activation="relu"),
     layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
     layers.MaxPool2D(pool_size=2),

     layers.Flatten(),
     layers.Dense(128, activation="relu"),
     layers.Dropout(0.5),
     layers.Dense(10, activation="softmax")
])

dropout_model.compile(loss='sparse_categorical_crossentropy',
                   optimizer='adam',
                   metrics=['accuracy'])
```

Convolutional Neural Networks

- Image Classification with CNNs
  - The model with dropout regularization

```
drop_history = dropout_model.fit(x_train, y_train,
epochs=10validation_data=(X_test, y_test)))
```

```
Epoch 1/10
60000/60000 [==============================] - 9s 147us/step - loss: 0.4540 - acc: 0.8393
Epoch 2/10
60000/60000 [==============================] - 8s 139us/step - loss: 0.2944 - acc: 0.8937
Epoch 3/10
60000/60000 [==============================] - 8s 139us/step - loss: 0.2476 - acc: 0.9098
Epoch 4/10
60000/60000 [==============================] - 8s 139us/step - loss: 0.2151 - acc: 0.9218
Epoch 5/10
60000/60000 [==============================] - 8s 138us/step - loss: 0.1936 - acc: 0.9286
Epoch 6/10
60000/60000 [==============================] - 8s 139us/step - loss: 0.1668 - acc: 0.9381
Epoch 7/10
60000/60000 [==============================] - 8s 138us/step - loss: 0.1505 - acc: 0.9434
Epoch 8/10
60000/60000 [==============================] - 8s 139us/step - loss: 0.1363 - acc: 0.9485
Epoch 9/10
60000/60000 [==============================] - 8s 138us/step - loss: 0.1235 - acc: 0.9527
Epoch 10/10
60000/60000 [==============================] - 8s 140us/step - loss: 0.1123 - acc: 0.9571

<tensorflow.python.keras.callbacks.History at 0x2770408a7f0>
```

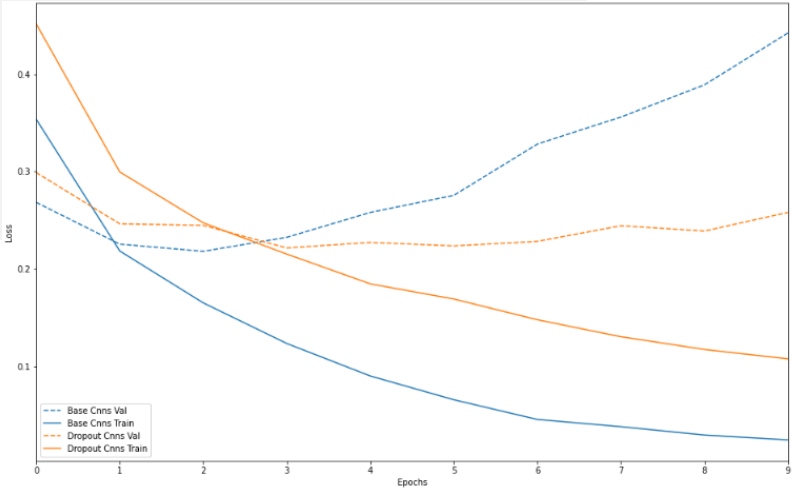
```
dropout_model.evaluate(x_train, y_train)  →  [0.05887163372437159, 0.9790333333333333]

dropout_model.evaluate(x_test, y_test)  →  [0.2695904264975339, 0.9225]
```

loss *accuracy*

Convolutional Neural Networks

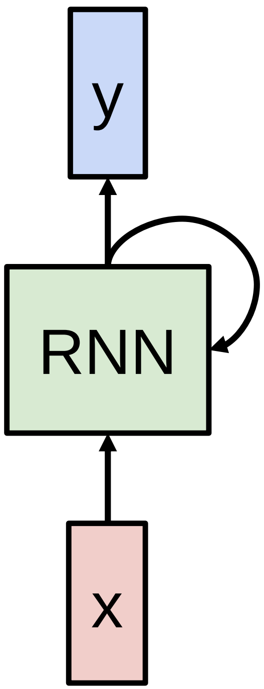
- Image Classification with CNNs
  - Plotting the learning curve

```
def plot_history(histories, key='loss'):
      plt.figure(figsize=(16,10))
      for name, history in histories:
            val = plt.plot(history.epoch, history.history['val_'+key],
            '--', label=name.title()+' Val')
            plt.plot(history.epoch, history.history[key],
            color=val[0].get_color(),
            label=name.title()+' Train')

      plt.xlabel('Epochs')
      plt.ylabel(key.replace('_',' ').title())
      plt.legend()
      plt.xlim([0,max(history.epoch)])

plot_history([('Base CNNs', base_history),
('Dropout CNNs', drop_history)])
```



Recurrent Neural Networks

- Recurrent Neural Networks : for analyzing sequential Data
  - Sequential data : language, video, stock price, weather, …
  - We can process a sequence of vectors **x** (x_1, x_2, x_3, …)
    by applying a **recurrence formula** at every time step



$$y_t = W_{hy} h_t$$

$$h_t = f_W(h_{t-1}, x_t)$$
$$= \tanh(W_{hh} h_{t-1} + W_{xh} x_t)$$

$$x_t$$

http://cs231n.stanford.edu

Recurrent Neural Networks

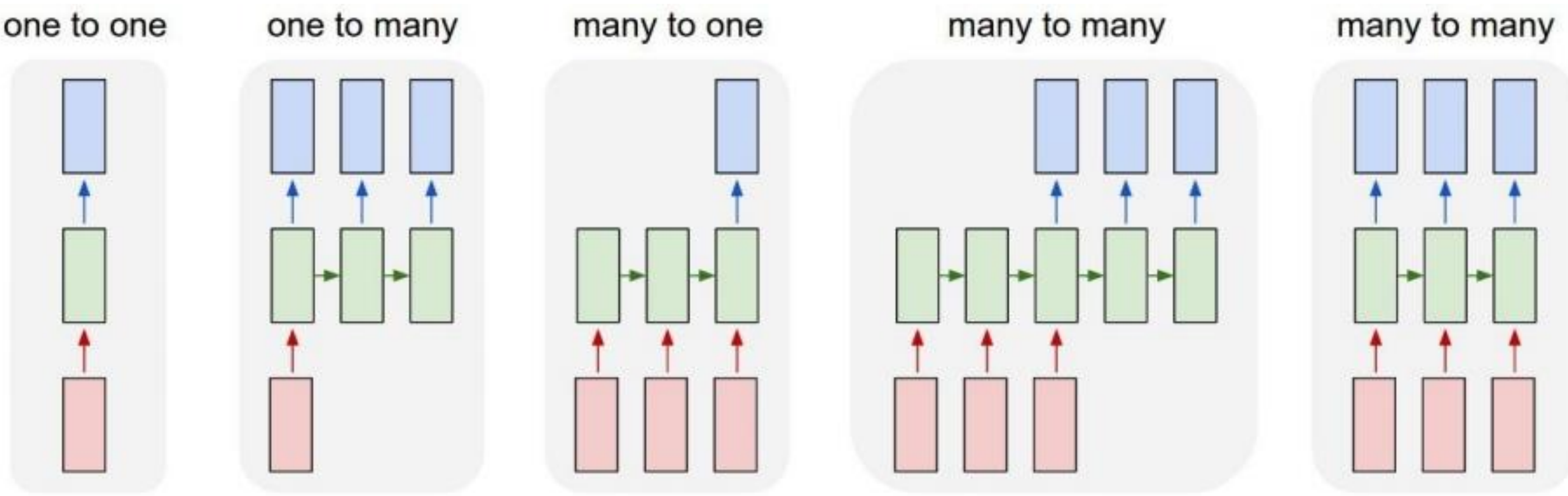
- Various types of input-output relations

"Two dogs play on the grass"

POS/NEG

"I love you"

Seq. of labels

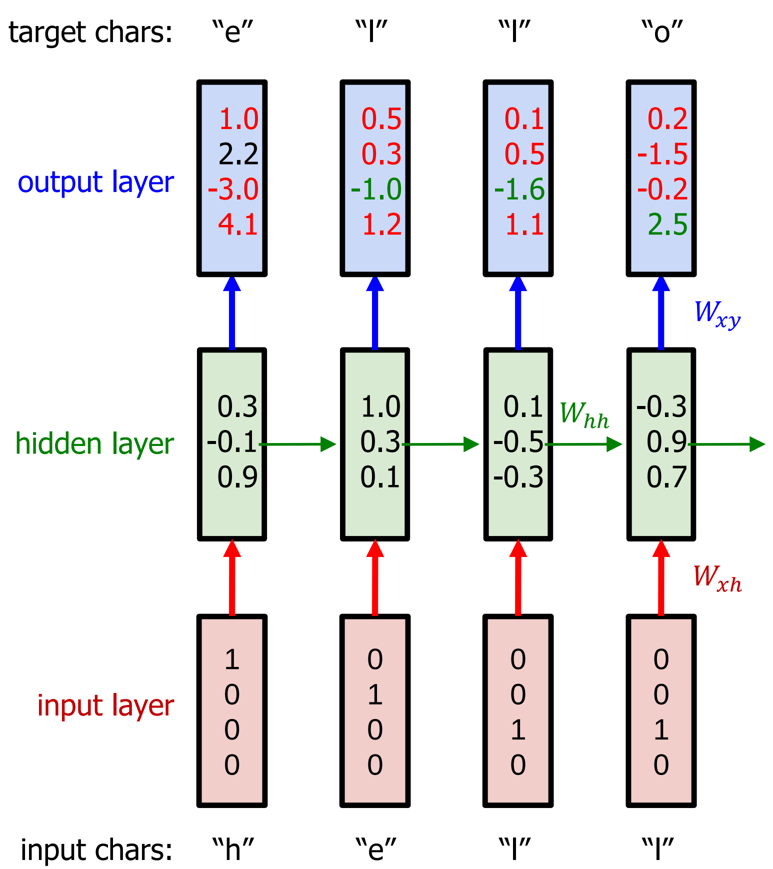


"This is a wonderful movie"

"Ich liebe dich"

Text Generation using RNN

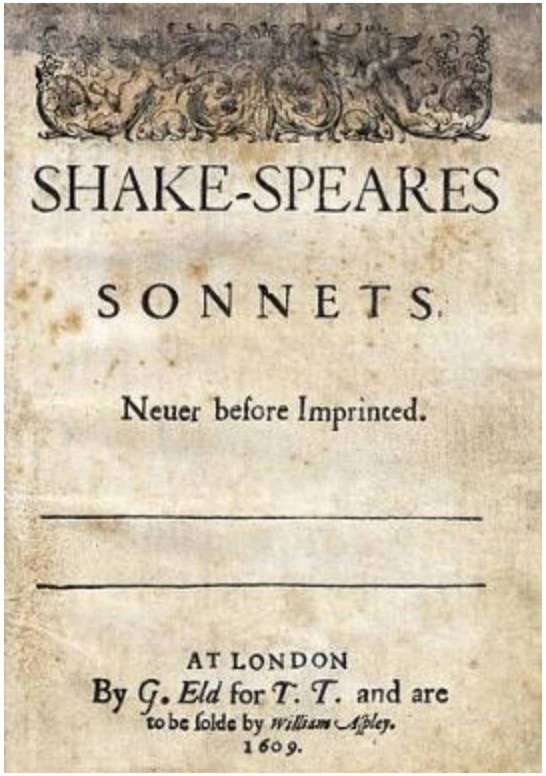
- Example :
  Learning Character-level Language Model
  - Vocabulary : [h, e, l, o]
  - Example training sequence : "hello"
  - $h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$
  - $\hat{y}_t = \text{Softmax}(W_{hy}h_t)$



http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture10.pdf

Recurrent Neural Networks

- Implementing simple RNN with TensorFlow
  - Dataset : Shakespeare's sonnets (Shakespeare.txt)




```
First Citizen:
Before we proceed any further, hear me speak.

All:
Speak, speak.

First Citizen:
You are all resolved rather to die than to famish?

All:
Resolved. resolved.

First Citizen:
First, you know Caius Marcius is chief enemy to the people.
```

Recurrent Neural Networks

- Character-level Language Model with RNNs
  - Load and preprocess the Shakespeare dataset

```
import tensorflow as tf
import numpy as np
import os
import time

# 1. Download the Shakespeare's Sonnet dataset
path_to_file = tf.keras.utils.get_file('shakespeare.txt',
'https://storage.googleapis.com/download.tensorflow.org/data/shakespeare.txt')

# Load whole text file as a string, then decode.
text = open(path_to_file, 'rb').read().decode(encoding='utf-8')

# length of text is the number of characters in it
print ('Length of text: {} characters'.format(len(text)))
```

```
Downloading data from https://storage.googleapis.com/download.tensorflow.org/data/shakespeare.txt
1122304/1115394 [==============================] - 0s 0us/step
Length of text: 1115394 characters
```

```
# We'll use the subset
text = text[:14592]
len(text)
```

```
14592
```

Recurrent Neural Networks

- Character-level Language Model with RNNs
  - Load and preprocess the Shakespeare dataset

```
# Take a look first 250 characters
print(text[:250])
```

```
First Citizen:
Before we proceed any further, hear me speak.

All:
Speak, speak.

First Citizen:
You are all resolved rather to die than to famish?

All:
Resolved. resolved.

First Citizen:
First, you know Caius Marcius is chief enemy to the people.
```

```
# The unique characters in the file
vocab = sorted(set(text))
print ('{} unique characters'.format(len(vocab)))
```

```
58 unique characters
```

Recurrent Neural Networks

- Character-level Language Model with RNNs
  - Load and preprocess the Shakespeare dataset

```
# 2. Vectorize the text
# Creating a mapping from unique characters to indices, and vice versa
char2idx = {u:i for i, u in enumerate(vocab)}
idx2char = np.array(vocab)

# Convert the characters to the indices
text_as_int = np.array([char2idx[c] for c in text])

# Show how the first 13 characters from the text are mapped to integers
print ('{} ---- characters mapped to int ---- > {}'.format(repr(text[:13]),
text_as_int[:13]))
```

```
'First Citizen' ---- characters mapped to int ---- > [18 47 56 57 58  1 15 47 58 47 64 43 52]
```

Recurrent Neural Networks

- Character-level Language Model with RNNs
  - Load and preprocess the Shakespeare dataset

```
# 3. Creating training task
# The maximum length sentence we want for a single input in characters
seq_length = 100

# Create training examples / targets
char_dataset = tf.data.Dataset.from_tensor_slices(text_as_int)

for i in char_dataset.take(5):
    print(idx2char[i.numpy()])
```

```
F
i
r
s
t
```

```
# Check the shape : char_dataset
np.array(list(char_dataset.as_numpy_iterator())).shape
```

```
(14592,)
```

Recurrent Neural Networks

- Character-level Language Model with RNNs
  - Load and preprocess the Shakespeare dataset

```
# 'batch' method convert these individual characters to sequences of the desired
size
sequences = char_dataset.batch(seq_length+1, drop_remainder=True)

for item in sequences.take(5):
     print(repr(''.join(idx2char[item.numpy()])))
```

```
'First Citizen:₩nBefore we proceed any further, hear me speak.₩n₩nAll:₩nSpeak, speak.₩n₩nFirst Citizen
'are all resolved rather to die than to famish?₩n₩nAll:₩nResolved. resolved.₩n₩nFirst Citizen:₩nFirst,
"now Caius Marcius is chief enemy to the people.₩n₩nAll:₩nWe know't, we know't.₩n₩nFirst Citizen:₩nLet
"ll him, and we'll have corn at our own price.₩nIs't a verdict?₩n₩nAll:₩nNo more talking on't; let it l
'one: away, away!₩n₩nSecond Citizen:₩nOne word, good citizens.₩n₩nFirst Citizen:₩nWe are accounted poo
```

```
# Check the shape : sequences
np.array(list(sequences.as_numpy_iterator())).shape
```

```
(144, 101)
```

Recurrent Neural Networks

- Character-level Language Model with RNNs
  - Load and preprocess the Shakespeare dataset

```
# map_func
def split_input_target(chunk):
     # input text is shifted to form the target text
     input_text = chunk[:-1]
     target_text = chunk[1:]
     return input_text, target_text

# 'map' method lets us easily apply a simple function to each batch
dataset = sequences.map(split_input_target)

# Print the examples
for input_ex, target_ex in dataset.take(1):
     print ('Input : ', repr(''.join(idx2char[input_ex.numpy()])))
     print ('Target :', repr(''.join(idx2char[target_ex.numpy()])))
```

```
Input :  'First Citizen:₩nBefore we proceed any further, hear me speak.₩n₩nAll:₩nSpeak, speak.₩n₩nFirs
Target : 'irst Citizen:₩nBefore we proceed any further, hear me speak.₩n₩nAll:₩nSpeak, speak.₩n₩nFirst
```

```
# Check the shape : dataset - (1)
np.array(list(dataset.as_numpy_iterator())).shape
```

Recurrent Neural Networks

- Character-level Language Model with RNNs
  - Load and preprocess the Shakespeare dataset

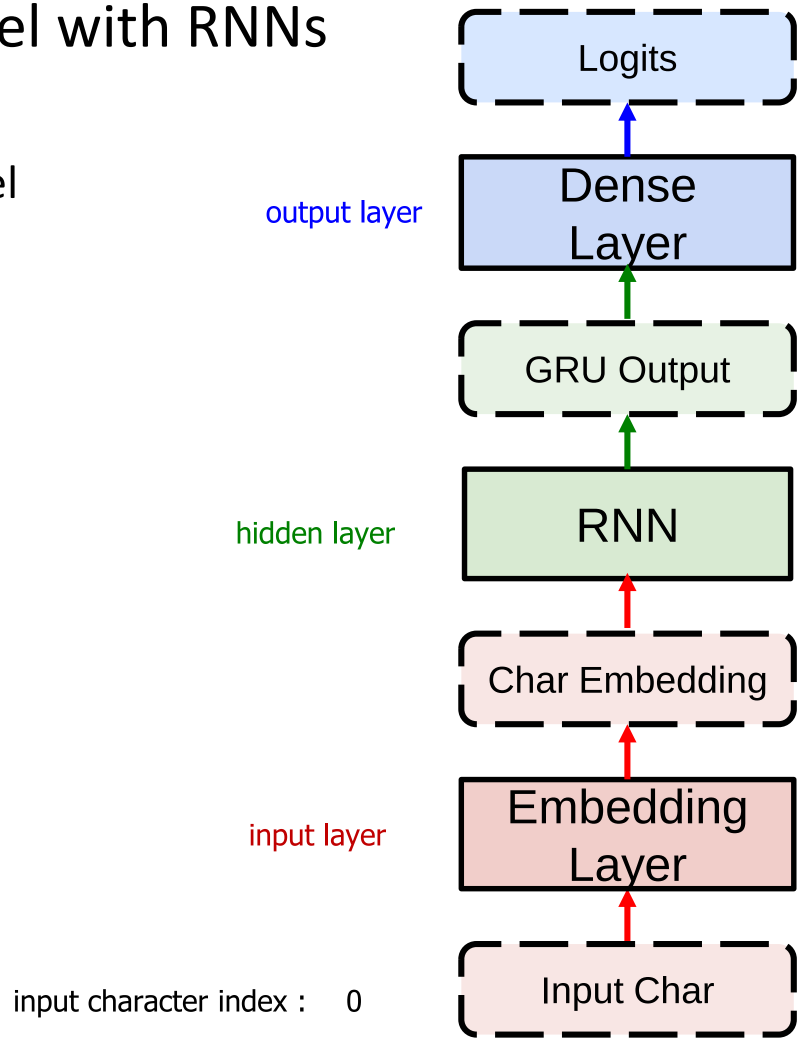
```
# 4. Create training batches
# Batch size
BATCH_SIZE = 16

# Buffer size to shuffle the dataset
BUFFER_SIZE = 100

dataset = dataset.shuffle(BUFFER_SIZE).batch(BATCH_SIZE, drop_remainder=True)
```

```
# Check the shape : dataset - (2)
np.array(list(dataset.as_numpy_iterator())).shape
```

```
(9, 2, 16, 100)
```

Recurrent Neural Networks

- Character-level Language Model with RNNs
  - Build the model
  - 3 layers are used to define this model
    1. tf.keras.layers.Embedding
    2. tf.keras.layers.RNN
    3. tf.keras.layers.Dense



Recurrent Neural Networks

- Character-level Language Model with RNNs
  - Build the model

```
# Length of the vocabulary in chars
vocab_size = len(vocab)
# The embedding dimension
embedding_dim = 128
# Number of RNN units
rnn_units = 256
```

```
def build_model(vocab_size, embedding_dim, rnn_units, batch_size):
	layers = tf.keras.layers
	model = tf.keras.Sequential([
		layers.Embedding(input_dim=vocab_size,
				output_dim=embedding_dim,
				batch_input_shape=[batch_size, None]
				),
		layers.SimpleRNN(rnn_units,
				return_sequences=True,
				stateful=True,
				recurrent_initializer='glorot_uniform' # Xavier
				Initialization
				),
		layers.Dense(vocab_size)
	])
	return model
```

Recurrent Neural Networks

- Character-level Language Model with RNNs
  - Build the model

```
# Build the model
model = build_model(
     vocab_size = vocab_size,
     embedding_dim=embedding_dim,
     rnn_units=rnn_units,
     batch_size=BATCH_SIZE)

# Check the model architecture
# Model can be run on inputs of any length
model.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (16, None, 128)           7424
_________________________________________________________________
simple_rnn (SimpleRNN)       (16, None, 256)           98560
_________________________________________________________________
dense_10 (Dense)             (16, None, 58)            14906
=================================================================
Total params: 120,890
Trainable params: 120,890
Non-trainable params: 0
_________________________________________________________________
```

Recurrent Neural Networks

- Character-level Language Model with RNNs
  - Try the model

```
# Check the shape of the output
for input_example_batch, target_example_batch in dataset.take(1):
     example_batch_predictions = model(input_example_batch)
print(example_batch_predictions.shape, "# (batch_size, sequence_length,
vocab_size)")
```

```
(64, 100, 65) # (batch_size, sequence_length, vocab_size)
```

Recurrent Neural Networks

- Character-level Language Model with RNNs
  - Try the model

```
# We need to sample from the output distribution, not to take the argmax of the distribution
sampled_indices = tf.random.categorical(example_batch_predictions[0], num_samples=1)
sampled_indices = tf.squeeze(sampled_indices,axis=-1).numpy()
```

```
array([29, 25, 20, 20, 45, 46, 22, 40, 40, 36, 25, 20, 51, 57, 10, 48, 11,
       50, 22,  5, 11, 10, 16,  9, 61, 36, 36, 63, 51, 22, 54, 63, 14, 25,
       64, 16, 59, 28, 53, 47, 15, 17, 35, 32, 16, 50, 58, 51,  0,  6,  4,
       28, 23, 27, 52, 59, 12,  9, 45, 53, 63, 13, 28, 52,  3, 19, 30, 64,
       38,  2, 44, 57, 48, 58, 38, 15, 42, 40, 35, 56,  0, 40, 10, 42, 19,
       43, 27, 26,  0, 31, 39,  6, 17, 24, 27, 25, 26, 21, 52, 56])
```

```
# Decode the predictions, the model shows poor performance
print("Input: \n", repr("".join(idx2char[input_example_batch[0]])))
print()
print("Next Char Predictions: \n", repr("".join(idx2char[sampled_indices ])))
```

```
Input:
 ':\nNow, by Saint Paul, this news is bad indeed.\nO, he hath kept an evil diet long,\nAnd overmuch consu'

Next Char Predictions:
 "QMHHghJbbXMHms:j;lJ';:D3wXXymJpyBMzDuPoiCEWTDltm\n,&PKOnu?3goyAPn$GRzZ!fsjtZCdbWr\nb:dGeON\nSa,ELOMNlnr"
```

Recurrent Neural Networks

- Character-level Language Model with RNNs
  - Train the model

```
# 1. Attach an optimizer, and a loss function
# define the loss function
def loss(labels, logits):
     # Because the output of model is logit,
     return tf.keras.losses.sparse_categorical_crossentropy(labels, logits,
     from_logits=True)

# Test the loss function
example_batch_loss = loss(target_example_batch, example_batch_predictions)
print("Prediction shape: ", example_batch_predictions.shape, " # (batch_size,
sequence_length, vocab_size)")
print("scalar_loss: ", example_batch_loss.numpy().mean())

# Configure the training procedure
model.compile(optimizer='adam', loss=loss)
```

```
Prediction shape:  (64, 100, 65)  # (batch_size, sequence_length, vocab_size)
scalar_loss:       4.172501
```

Recurrent Neural Networks

- Character-level Language Model with RNNs
  - Train the model

```
# 2. Configure the checkpoints
# `tf.keras.callbacks.ModelCheckpoint` : The callback function to save the model
checkpoint

# Directory where the model weights will be saved
ckpt_dir = './training_rnns_ckpts'

# Checkpoint name
ckpt_prefix = os.path.join(ckpt_dir, "ckpt_rnns_{epoch}")

# Callback function to save the model weights
ckpt_callback=tf.keras.callbacks.ModelCheckpoint(
filepath=ckpt_prefix,
save_weights_only=True)
```

Recurrent Neural Networks

- Character-level Language Model with RNNs
  - Train the model

```
# 3. Execute the training
EPOCHS=10
rnn_history = model.fit(dataset, epochs=EPOCHS, callbacks=[ckpt_callback])
```

```
Epoch 1/10
172/172 [==============================] - 20s 114ms/step - loss: 2.6591
Epoch 2/10
172/172 [==============================] - 20s 114ms/step - loss: 2.0140
Epoch 3/10
172/172 [==============================] - 20s 114ms/step - loss: 1.7984
Epoch 4/10
172/172 [==============================] - 20s 114ms/step - loss: 1.6711
Epoch 5/10
172/172 [==============================] - 20s 115ms/step - loss: 1.5867
Epoch 6/10
172/172 [==============================] - 20s 114ms/step - loss: 1.5296
Epoch 7/10
172/172 [==============================] - 20s 114ms/step - loss: 1.4854
Epoch 8/10
172/172 [==============================] - 20s 114ms/step - loss: 1.4529
Epoch 9/10
172/172 [==============================] - 20s 114ms/step - loss: 1.4287
Epoch 10/10
172/172 [==============================] - 20s 114ms/step - loss: 1.4027
```

Recurrent Neural Networks

- Character-level Language Model with RNNs
  - Generate text

```
# Check the latest checkpoint
tf.train.latest_checkpoint(ckpt_dir)
```

```
'./training_rnns_ckpts/ckpt_rnns_10'
```

```
# To run the model with one sample(not with batch_size of samples),
# We rebuild the model, and load the weights from the saved checkpoint.
model = build_model(vocab_size, embedding_dim, rnn_units, batch_size=1)
model.load_weights(tf.train.latest_checkpoint(ckpt_dir))
```

```
# Check the model summary
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (1, None, 256)            16640
_________________________________________________________________
simple_rnn_1 (SimpleRNN)     (1, None, 1024)           1311744
_________________________________________________________________
dense_5 (Dense)              (1, None, 65)             66625
=================================================================
Total params: 1,395,009
Trainable params: 1,395,009
Non-trainable params: 0
```

Recurrent Neural Networks

- Character-level Language Model with RNNs
  - Generate text – (1)

```
# The prediction loop
def generate_text(model, start_string):
     # Number of characters to generate
     n_generate = 1000

     # Converting start_strings to index (vectorizing)
     input_eval = [char2idx[s] for s in start_string]
     input_eval = tf.expand_dims(input_eval, 0)

     # Making the empty list to store results
     text_generated = []

     # Low temperatures -> more predictable text.
     # Higher temperatures -> more surprising text.
     temperature = 1
```

Recurrent Neural Networks

- Character-level Language Model with RNNs
  - Generate text – (2)

```
# Here batch size == 1
model.reset_states()
for i in range(n_generate):
     predictions = model(input_eval)
     # remove the batch dimension
     predictions = tf.squeeze(predictions, 0)

     # using a categorical distribution to predict the character
     predictions = predictions / temperature
     predicted_id = tf.random.categorical(predictions, num_samples=1)[-
     1,0].numpy()

     # Passing the predicted character as the next input to the model
     # along with the previous hidden state
     input_eval = tf.expand_dims([predicted_id], 0)

     text_generated.append(idx2char[predicted_id])

return (start_string + ''.join(text_generated))
```

Recurrent Neural Networks

- Character-level Language Model with RNNs
  - Generate text

```
# It shows poor performance
# Sometimes it prints out a series of meaningless characters
# -> Due to vanishing(or exploding) gradients problem
print(generate_text(model, start_string=u"ROMEO: "))
```

```
ROMEO: that out of nibus of murdinge with queen of Rommant;
And
nother mat,
What you do done,
Then lathing a foul add him
to then.

KINGS moursed
fellow to have this of the rosprest
Try father? For do it him, but more polighits of or thoughts are tears!

EDWARD IV:

KING RICHARD III:
Divire,
And so banamiced to the pleasing stands?
Is they actions
To free but in oherea
Brace of our a Plorce
Your foul offord stiful and at feer
```

```
ROMEO: QXEXEOVJUGQUXYZAQUGJUL&JUGZAUDJUKD3UK&XCKYZYXXGQUQUTZAQUFXEQVH
KXEZEX$NQXYOXPZYUXXY3UX$XX3$ZZUF3YRUXX33ZYZULY$$ZYZ3RYXUKQQULJZHZQUGQU
JUxJQULEXGZRUXEXXY&3ZAUKX3YOXFJGHXYQYAUCKZIUDUK$3YZYYXY$XK33$KX3Z3UKE
```

Recurrent Neural Networks

- Text Classification with RNNs
  - Import TensorFlow and other libraries

```
import tensorflow_datasets as tfds #conda install -c anaconda tensorflow-datasets
import tensorflow as tf
import matplotlib.pyplot as plt
```

```
Downloading and preparing dataset imdb_reviews/subwords8k/1.0.0 (download: 80.23 MiB, gen
Dl Completed...: 100% 1/1 [00:01<00:00, 1.58s/ url]
Dl Size...: 100% 80/80 [00:01<00:00, 51.59 MiB/s]

Shuffling and writing examples to /root/tensorflow_datasets/imdb_reviews/subwords8k/1.0.0.incompleteVY
83% 20794/25000 [00:00<00:00, 207937.22 examples/s]
Shuffling and writing examples to /root/tensorflow_datasets/imdb_reviews/subwords8k/1.0.0.incompleteVY
82% 20623/25000 [00:00<00:00, 206228.23 examples/s]
Shuffling and writing examples to /root/tensorflow_datasets/imdb_reviews/subwords8k/1.0.0.incompleteVY
22% 10793/50000 [00:00<00:00, 107929.59 examples/s]
Dataset imdb_reviews downloaded and prepared to /root/tensorflow_datasets/imdb_reviews/sul
```

Recurrent Neural Networks

- Text Classification with RNNs
  - Load and preprocess the IMDb dataset

```
# info object has the lookup table(encoder) of token and index
encoder = info.features['text'].encoder
print('Vocabulary size: {}'.format(encoder.vocab_size))
```

```
Vocabulary size: 8185
```

```
# Test the encoder
sample_string = 'Hello TensorFlow.'
encoded_string = encoder.encode(sample_string)
print('Encoded string is {}'.format(encoded_string))
original_string = encoder.decode(encoded_string)
print('The original string: "{}"'.format(original_string))
```

```
Encoded string is [4025, 222, 6307, 2327, 4043, 2120, 7975]
The original string: "Hello TensorFlow."
```

```
# Test the encoder
for index in encoded_string:
print('{} ----> {}'.format(index, encoder.decode([index])))
```

```
4025 ----> Hell
222 ----> o
6307 ----> Ten
2327 ----> sor
4043 ----> Fl
2120 ----> ow
7975 ----> .
```

Recurrent Neural Networks

- Text Classification with RNNs
  - Load and preprocess the IMDb dataset

```
# Creating training task
BUFFER_SIZE = 10000
BATCH_SIZE = 64

train_dataset = train_dataset.shuffle(BUFFER_SIZE)
train_dataset = train_dataset.padded_batch(BATCH_SIZE)

test_dataset = test_dataset.padded_batch(BATCH_SIZE)
```

Recurrent Neural Networks

- Text Classification with RNNs
  - Load and preprocess the IMDb dataset

```
# Check the shape of batches
for x, y in train_dataset.take(2):
print(x)
print(y)
print("-"*90)
```

```
tf.Tensor(
[[5700 1101   89 ...    0    0    0]
 [4074  370    2 ...    0    0    0]
 [ 135 7968    8 ...    0    0    0]
 ...
 [7969 2517 1104 ...    0    0    0]
 [  12  284  107 ...    0    0    0]
 [2883  798  968 ...    0    0    0]], shape=(64, 1119), dtype=int64)
tf.Tensor(
[0 0 0 1 1 0 1 1 0 0 0 0 0 1 1 1 1 0 1 1 0 0 1 0 0 0 0 0 1 1 0 1 1 0 0 0 0
 0 1 0 0 1 0 0 1 1 0 0 0 0 0 1 0 1 0 0 1 0 0 0 1 0 0 1], shape=(64,), dtype=int64)
------------------------------------------------------------------------------------------
tf.Tensor(
[[  69  279   35 ...    0    0    0]
 [1028  330  571 ...    0    0    0]
 [  12   31    7 ...    0    0    0]
 ...
 [ 407   41 2579 ...    0    0    0]
 [  12 2768  109 ...    0    0    0]
 [ 147 5562 6904 ...    0    0    0]], shape=(64, 1335), dtype=int64)
tf.Tensor(
[0 0 0 1 1 0 1 1 1 0 1 0 0 0 1 1 1 1 0 1 1 0 1 0 1 0 0 0 1 0 1 0 1 0 1 1 1
 1 1 1 0 1 1 1 0 0 0 0 0 1 0 0 0 0 0 0 1 1 0 0 0 0 0 0 1], shape=(64,), dtype=int64)
------------------------------------------------------------------------------------------
```

Recurrent Neural Networks

- Text Classification with RNNs
  - Build the model

```
layers = tf.keras.layers
model = tf.keras.Sequential([
      layers.Embedding(encoder.vocab_size, 64),
      layers.Bidirectional(layers.LSTM(64)),
      layers.Dense(64, activation='relu'),
      layers.Dense(1)
])
```

```
model.compile(loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
optimizer=tf.keras.optimizers.Adam(1e-4),
metrics=['accuracy'])
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, None, 64)          523840
_________________________________________________________________
bidirectional (Bidirectional (None, 128)               66048
_________________________________________________________________
dense_2 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 65
=================================================================
Total params: 598,209
Trainable params: 598,209
Non-trainable params: 0
_________________________________________________________________
```

Recurrent Neural Networks

- Text Classification with RNNs
  - Train the model

```
history = model.fit(train_dataset, epochs=5,
validation_data=test_dataset,
validation_steps=30)
```

  - Evaluate the model

```
test_loss, test_acc = model.evaluate(test_dataset)
print('Test Loss: {}'.format(test_loss))
print('Test Accuracy: {}'.format(test_acc))
```

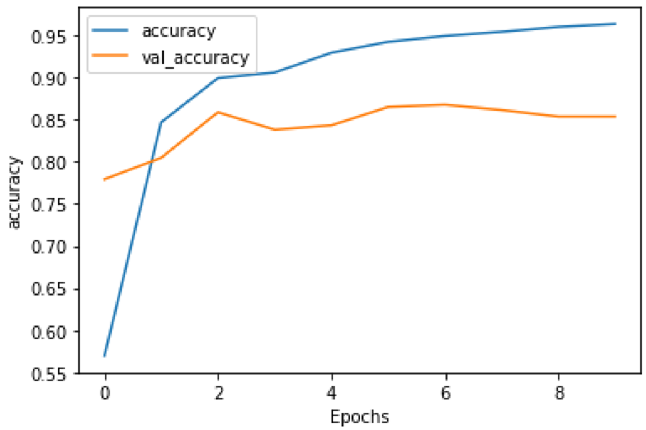
```
391/391 [==============================] - 17s 44ms/step - loss: 0.4383 - accuracy: 0.8537
Test Loss: 0.4383341670036316
Test Accuracy: 0.8536800146102905
```

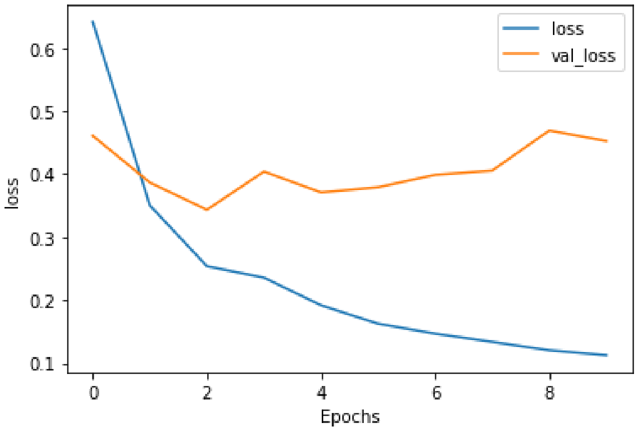
Recurrent Neural Networks

- Text Classification with RNNs
  - Evaluate the model

```
# helper function
def plot_graphs(history, metric):
      plt.plot(history.history[metric])
      plt.plot(history.history['val_'+metric], '')
      plt.xlabel("Epochs")
      plt.ylabel(metric)
      plt.legend([metric, 'val_'+metric])
      plt.show()
```

```
plot_graphs(history, 'accuracy')
plot_graphs(history, 'loss')
```





Recurrent Neural Networks

- Text Classification with RNNs
  - Evaluate the model

```
def sample_pred(text, model):
    encoded_text = encoder.encode(text)
    encoded_text = tf.cast(encoded_text, tf.float32)
    predictions = model.predict(tf.expand_dims(encoded_text, 0))
    prob = tf.sigmoid(predictions)[0][0].numpy()
    print('Prob : ', prob)
    if prob >= 0.5:
        return "Positive"
    else:
        return "Negative"
```

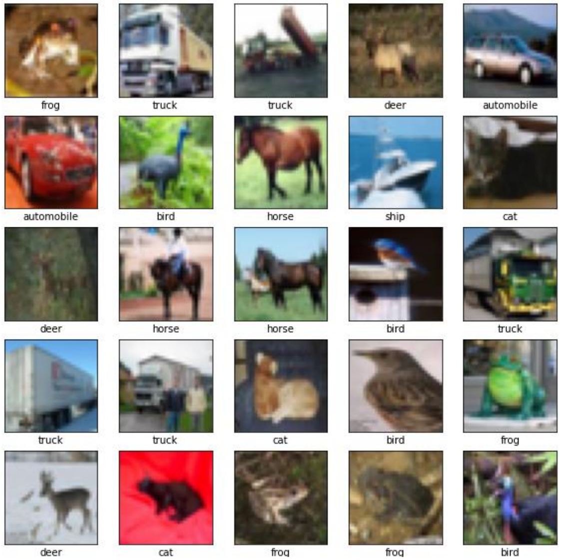
```
sample_pred_text = 'You should watch this movie, this movie is excellent'
sample_pred(sample_pred_text, model)
```

```
Prob :  0.79965454
'Positive'
```

Quiz 1 : Image Classification Model on the CIFAR-10

- Build the Convolutional Neural Networks
  - Build the model following the bellow model summary
  - Apply the dropout regularization to the model and compare the result
- Compare the performance of the model built last week

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 32, 32, 32)        896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 14, 14, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 7, 7, 64)          0
_________________________________________________________________
flatten (Flatten)            (None, 3136)              0
_________________________________________________________________
dense (Dense)                (None, 128)               401536
_________________________________________________________________
dense_1 (Dense)              (None, 10)                1290
=================================================================
Total params: 422,218
Trainable params: 422,218
Non-trainable params: 0
```



Quiz 2 : Character-level Language Model

- Build the Character-level Language Model with LSTM
- Compare the generated text to the one generated by RNNs