```
!curl -O https://ai.stanford.edu/~amaas/data/sentiment/aclImdb_v1.tar.gz
!tar -xf aclImdb_v1.tar.gz
!rm -r aclImdb/train/unsup
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 80.2M  100 80.2M    0     0  6571k      0  0:00:12  0:00:12 --:--:-- 13.8M
```

DATA PREPARATION

Preparing an integer sequence dataset

A model built on one-hot encoded vector sequences

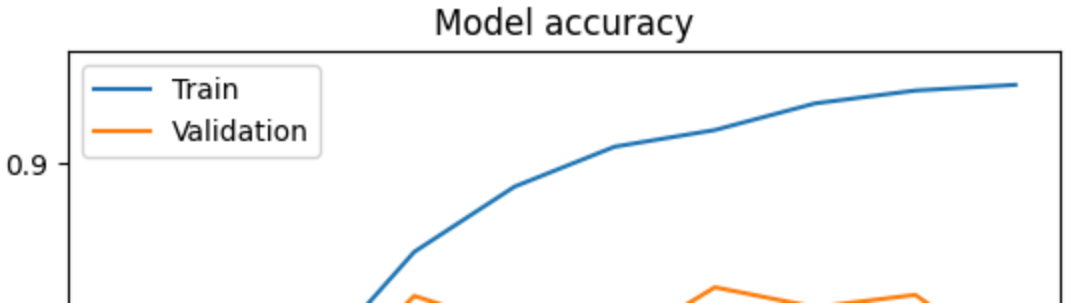
Training a basic model

```
import os, pathlib, shutil, random
from tensorflow import keras

batch_size = 32
max_length = 150
max_tokens = 10000
num_train_samples = 100
num_val_samples = 10000

base_dir = pathlib.Path("aclImdb")
val_dir = base_dir / "val"
train_dir = base_dir / "train"

for category in ("neg", "pos"):
    os.makedirs(val_dir / category, exist_ok=True)
    files = os.listdir(train_dir / category)
    random.Random(1337).shuffle(files)
    num_val_samples_cat = int(num_val_samples/2)
    val_files = files[-num_val_samples_cat:]
    for fname in val_files:
        shutil.move(train_dir / category / fname,
                    val_dir / category / fname)

train_ds = keras.preprocessing.text_dataset_from_directory(
    "aclImdb/train",
    batch_size=batch_size,
    validation_split=0.2,
    subset='training',
    seed=1337)

train_ds = train_ds.take(num_train_samples)

val_ds = keras.preprocessing.text_dataset_from_directory(
    "aclImdb/train",
    batch_size=batch_size,
    validation_split=0.2,
    subset='validation',
    seed=1337)

val_ds = val_ds.take(num_val_samples)

test_ds = keras.preprocessing.text_dataset_from_directory(
    "aclImdb/test", batch_size=batch_size)

text_only_train_ds = train_ds.map(lambda x, y: x)

from tensorflow.keras import layers

text_vectorization = layers.TextVectorization(
    max_tokens=max_tokens,
    output_mode="int",
```

```
    output_sequence_length=max_length,
)

text_vectorization.adapt(text_only_train_ds)

int_train_ds = train_ds.map(lambda x, y: (text_vectorization(x), y),
                            num_parallel_calls=4)

int_val_ds = val_ds.map(lambda x, y: (text_vectorization(x), y),
                        num_parallel_calls=4)

int_test_ds = test_ds.map(lambda x, y: (text_vectorization(x), y),
                          num_parallel_calls=4)

import tensorflow as tf

inputs = keras.Input(shape=(None,), dtype="int64")
embedded = tf.one_hot(inputs, depth=max_tokens)
x = layers.Bidirectional(layers.LSTM(32))(embedded)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs, outputs)

model.compile(
    optimizer="rmsprop",
    loss="binary_crossentropy",
    metrics=["accuracy"]
)

model.summary()

callbacks = [
    keras.callbacks.ModelCheckpoint("one_hot_bidir_lstm.keras", save_best_only=True)
]

history = model.fit(
    int_train_ds,
    validation_data=int_val_ds,
    epochs=10,
    callbacks=callbacks
)

model = keras.models.load_model("one_hot_bidir_lstm.keras")
print(f"Test acc: {model.evaluate(int_test_ds)[1]:.3f}")
import matplotlib.pyplot as plt

# Plot training & validation accuracy values
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
```

```
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()

# Plot training & validation loss values
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```

```
Found 15000 files belonging to 2 classes.
Using 12000 files for training.
Found 15000 files belonging to 2 classes.
Using 3000 files for validation.
Found 25000 files belonging to 2 classes.
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, None)]            0

 tf.one_hot (TFOpLambda)     (None, None, 10000)       0

 bidirectional (Bidirection  (None, 64)                2568448
 al)

 dropout (Dropout)           (None, 64)                0

 dense (Dense)               (None, 1)                 65

=================================================================
Total params: 2568513 (9.80 MB)
Trainable params: 2568513 (9.80 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/10
100/100 [==============================] - 13s 71ms/step - loss: 0.6928 - accuracy: 0
Epoch 2/10
100/100 [==============================] - 5s 54ms/step - loss: 0.6533 - accuracy: 0.6
Epoch 3/10
100/100 [==============================] - 5s 53ms/step - loss: 0.5227 - accuracy: 0.7
Epoch 4/10
100/100 [==============================] - 5s 52ms/step - loss: 0.4056 - accuracy: 0.8
Epoch 5/10
100/100 [==============================] - 5s 51ms/step - loss: 0.3243 - accuracy: 0.8
Epoch 6/10
100/100 [==============================] - 5s 53ms/step - loss: 0.2731 - accuracy: 0.9
Epoch 7/10
100/100 [==============================] - 5s 54ms/step - loss: 0.2325 - accuracy: 0.9
Epoch 8/10
100/100 [==============================] - 5s 53ms/step - loss: 0.1891 - accuracy: 0.9
Epoch 9/10
100/100 [==============================] - 5s 53ms/step - loss: 0.1631 - accuracy: 0.9
Epoch 10/10
100/100 [==============================] - 5s 50ms/step - loss: 0.1779 - accuracy: 0.9
782/782 [==============================] - 17s 20ms/step - loss: 0.4144 - accuracy: 0
Test acc: 0.814
```

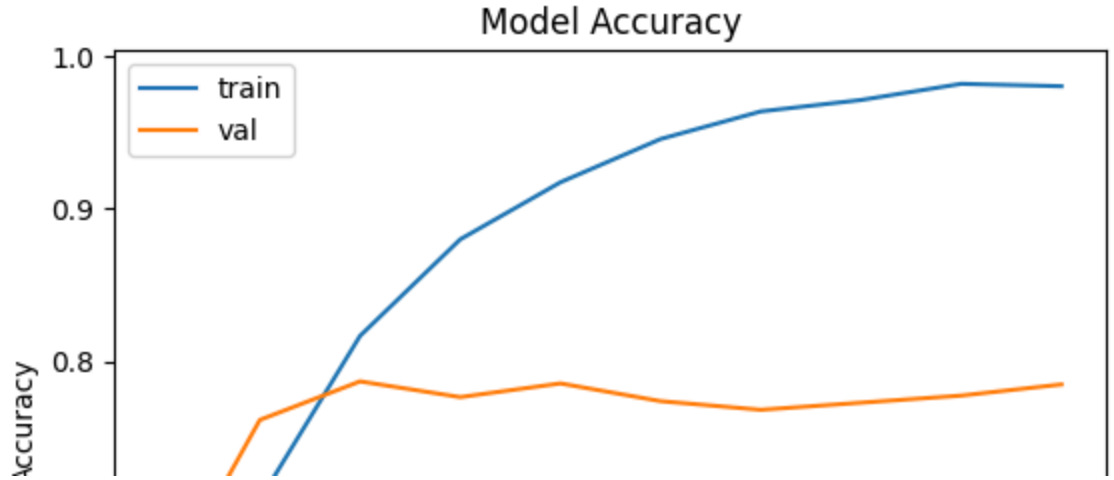


Instantiating an layer

Model that uses an Embedding layer trained from scratch

```
import matplotlib.pyplot as plt

inputs = keras.Input(shape=(None,), dtype="int64")
embedded = layers.Embedding(input_dim=max_tokens, output_dim=256)(inputs)
x = layers.Bidirectional(layers.LSTM(32))(embedded)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs, outputs)
model.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
model.summary()

callbacks = [
    keras.callbacks.ModelCheckpoint("embeddings_bidir_gru.keras",
                                    save_best_only=True)
]

history = model.fit(int_train_ds, validation_data=int_val_ds, epochs=10, callbacks=callbacks

model = keras.models.load_model("embeddings_bidir_gru.keras")
print(f"Test acc: {model.evaluate(int_test_ds)[1]:.3f}")

# Plot the training and validation accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()

# Plot the training and validation loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```

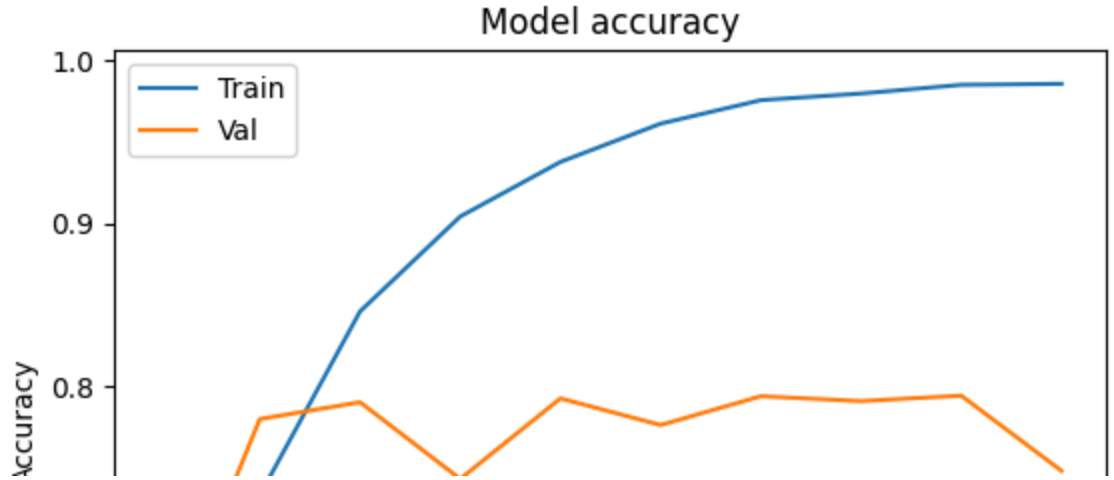
```
Model: "model_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, None)]            0

 embedding (Embedding)       (None, None, 256)         2560000

 bidirectional_1 (Bidirecti  (None, 64)                73984
 onal)

 dropout_1 (Dropout)         (None, 64)                0

 dense_1 (Dense)             (None, 1)                 65

=================================================================
Total params: 2634049 (10.05 MB)
Trainable params: 2634049 (10.05 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/10
100/100 [==============================] - 12s 88ms/step - loss: 0.6854 - accuracy: 0
Epoch 2/10
100/100 [==============================] - 5s 49ms/step - loss: 0.5761 - accuracy: 0.7
Epoch 3/10
100/100 [==============================] - 5s 47ms/step - loss: 0.4338 - accuracy: 0.8
Epoch 4/10
100/100 [==============================] - 3s 32ms/step - loss: 0.3287 - accuracy: 0.8
Epoch 5/10
100/100 [==============================] - 3s 32ms/step - loss: 0.2281 - accuracy: 0.9
Epoch 6/10
100/100 [==============================] - 3s 31ms/step - loss: 0.1676 - accuracy: 0.9
Epoch 7/10
100/100 [==============================] - 3s 34ms/step - loss: 0.1221 - accuracy: 0.9
Epoch 8/10
100/100 [==============================] - 3s 29ms/step - loss: 0.0911 - accuracy: 0.9
Epoch 9/10
100/100 [==============================] - 3s 25ms/step - loss: 0.0623 - accuracy: 0.9
Epoch 10/10
100/100 [==============================] - 2s 24ms/step - loss: 0.0646 - accuracy: 0.9
782/782 [==============================] - 8s 8ms/step - loss: 0.4728 - accuracy: 0.78
Test acc: 0.782
```



Model 3 - An Embedding Layer with Masking enabled

```
import matplotlib.pyplot as plt

# Define the model architecture
inputs = keras.Input(shape=(None,), dtype="int64")
embedded = layers.Embedding(
    input_dim=max_tokens, output_dim=256, mask_zero=True)(inputs)
x = layers.Bidirectional(layers.LSTM(32))(embedded)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs, outputs)

# Compile the model
model.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
model.summary()

# Define the callbacks
callbacks = [
    keras.callbacks.ModelCheckpoint("embeddings_bidir_gru_with_masking.keras",
                                    save_best_only=True),
    keras.callbacks.History()
]

# Train the model
history = model.fit(int_train_ds, validation_data=int_val_ds, epochs=10, callbacks=callbacks

# Load the best model
model = keras.models.load_model("embeddings_bidir_gru_with_masking.keras")
print(f"Test acc: {model.evaluate(int_test_ds)[1]:.3f}")

# Plot the training and validation accuracy and loss
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper left')
plt.show()

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper left')
plt.show()
```

```
Model: "model_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_3 (InputLayer)        [(None, None)]            0

 embedding_1 (Embedding)     (None, None, 256)         2560000

 bidirectional_2 (Bidirecti  (None, 64)                73984
 onal)

 dropout_2 (Dropout)         (None, 64)                0

 dense_2 (Dense)             (None, 1)                 65

=================================================================
Total params: 2634049 (10.05 MB)
Trainable params: 2634049 (10.05 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/10
100/100 [==============================] - 19s 115ms/step - loss: 0.6767 - accuracy: 0
Epoch 2/10
100/100 [==============================] - 6s 60ms/step - loss: 0.5502 - accuracy: 0.7
Epoch 3/10
100/100 [==============================] - 5s 52ms/step - loss: 0.3818 - accuracy: 0.8
Epoch 4/10
100/100 [==============================] - 4s 44ms/step - loss: 0.2633 - accuracy: 0.9
Epoch 5/10
100/100 [==============================] - 4s 35ms/step - loss: 0.1782 - accuracy: 0.9
Epoch 6/10
100/100 [==============================] - 4s 43ms/step - loss: 0.1166 - accuracy: 0.9
Epoch 7/10
100/100 [==============================] - 3s 34ms/step - loss: 0.0749 - accuracy: 0.9
Epoch 8/10
100/100 [==============================] - 3s 32ms/step - loss: 0.0652 - accuracy: 0.9
Epoch 9/10
100/100 [==============================] - 4s 36ms/step - loss: 0.0456 - accuracy: 0.9
Epoch 10/10
100/100 [==============================] - 3s 30ms/step - loss: 0.0445 - accuracy: 0.9
782/782 [==============================] - 12s 9ms/step - loss: 0.4510 - accuracy: 0.7
Test acc: 0.789
```



Model 4 - Using Pretrained Word Embedding

```
!wget http://nlp.stanford.edu/data/glove.6B.zip
!unzip -q glove.6B.zip

import numpy as np
path_to_glove_file = "glove.6B.100d.txt"

embeddings_index = {}
with open(path_to_glove_file) as f:
    for line in f:
        word, coefs = line.split(maxsplit=1)
        coefs = np.fromstring(coefs, "f", sep=" ")
        embeddings_index[word] = coefs

print(f"Found {len(embeddings_index)} word vectors.")

embedding_dim = 100
max_tokens = 10000
max_len = 150
num_samples = 100
validation_samples = 10000

vocabulary = text_vectorization.get_vocabulary()
word_index = dict(zip(vocabulary, range(len(vocabulary))))
word_index = {k: v for k, v in word_index.items() if v < max_tokens}

embedding_matrix = np.zeros((max_tokens, embedding_dim))
for word, i in word_index.items():
    if i < max_tokens:
        embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector

embedding_layer = layers.Embedding(
    max_tokens,
    embedding_dim,
    embeddings_initializer=keras.initializers.Constant(embedding_matrix),
    trainable=False,
    mask_zero=True,
)

inputs = keras.Input(shape=(None,), dtype="int64")
embedded = embedding_layer(inputs)
x = layers.Bidirectional(layers.LSTM(32))(embedded)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs, outputs)
model.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
model.summary()
```

```
callbacks = [
    keras.callbacks.ModelCheckpoint("glove_embeddings_sequence_model.keras",
                                    save_best_only=True)
]


history = model.fit(int_train_ds.take(num_samples).cache(),
          validation_data=int_val_ds.take(validation_samples).cache(),
          epochs=10, callbacks=callbacks)

model = keras.models.load_model("glove_embeddings_sequence_model.keras")
_, test_acc = model.evaluate(int_test_ds.take(validation_samples))
print(f"Test acc: {test_acc:.3f}")

# Plot training and validation accuracy
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
epochs = range(1, len(acc) + 1)
plt.plot(epochs, acc, 'bo', label='Training accuracy')
plt.plot(epochs, val_acc, 'b', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

# Plot training and validation loss
loss = history.history['loss']
val_loss = history.history['val_loss']
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```