jjoy6_Assignment_4

April 20, 2025

Assignment_4

[ ]:
```
import os, pathlib, shutil, random
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.layers import TextVectorization
import tensorflow as tf
```

Downloading IMBD data

[ ]:
```
!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  18.0M      0  0:00:04  0:00:04 --:--:-- 18.0M
```

Preparing Data

[ ]:
```
batch_size = 32
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 = 5000
    val_files = files[-num_val_samples:]
    for fname in val_files:
        shutil.move(train_dir / category / fname,
                    val_dir / category / fname)
```

Function to change training size

[ ]:
```
def make_subset(subset_name, new_subset_name, sample_size):
    for category in ("neg", "pos"):
        dir = base_dir / new_subset_name / category
```

```
        src_dir = base_dir / subset_name / category
        os.makedirs(dir, exist_ok=True)
        fnames = [f for f in os.listdir(src_dir)]

        #sampled_files = random.sample(fnames, sample_size)
        sampled_files = fnames[:sample_size]
        for sampled_files in sampled_files:
            shutil.copyfile(src=src_dir / sampled_files,
                            dst=dir / sampled_files)
```

Making subsets of various training sizes

[ ]:
```
"""
make_subset("train", "train100", 50)
train_ds = keras.utils.text_dataset_from_directory(
    "aclImdb/train100", batch_size=batch_size
)
"""

"""
make_subset("train", "train200", 100)
train_ds = keras.utils.text_dataset_from_directory(
    "aclImdb/train200", batch_size=batch_size
)
"""
"""
make_subset("train", "train300", 150)
train_ds = keras.utils.text_dataset_from_directory(
    "aclImdb/train300", batch_size=batch_size
)
"""

"""
make_subset("train", "train7500", 7500)
train_ds = keras.utils.text_dataset_from_directory(
    "aclImdb/train7500", batch_size=batch_size
)
"""

make_subset("train", "train7500", 25)
train_ds = keras.utils.text_dataset_from_directory(
    "aclImdb/train7500", batch_size=batch_size
)
```

```
Found 50 files belonging to 2 classes.
```

Splitting Data into training, validation, and testing

[ ]:
```
val_ds = keras.utils.text_dataset_from_directory(
    "aclImdb/val", batch_size=batch_size
)
test_ds = keras.utils.text_dataset_from_directory(
    "aclImdb/test", batch_size=batch_size
)
text_only_train_ds = train_ds.map(lambda x, y: x)
```

```
Found 10000 files belonging to 2 classes.
Found 25000 files belonging to 2 classes.
```

Preparing text data Text standardization Text splitting (tokenization) Vocabulary indexing Using the TextVectorization layer

[ ]:
```
#Two approaches for representing groups of words: Sets and sequences
#Preparing the IMDB movie reviews data

#Processing words as a set: The bag-of-words approach
#Single words (unigrams) with binary encoding
#Preprocessing our datasets with a TextVectorization layer
tokenSize = 2771

text_vectorization = TextVectorization(
    max_tokens=10000,
    output_mode="multi_hot",
)

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

binary_1gram_train_ds = train_ds.map(
    lambda x, y: (text_vectorization(x), y),
    num_parallel_calls=4)
binary_1gram_val_ds = val_ds.map(
    lambda x, y: (text_vectorization(x), y),
    num_parallel_calls=4)
binary_1gram_test_ds = test_ds.map(
    lambda x, y: (text_vectorization(x), y),
    num_parallel_calls=4)

#Inspecting the output of our binary unigram dataset

for inputs, targets in binary_1gram_train_ds:
    print("inputs.shape:", inputs.shape)
    print("inputs.dtype:", inputs.dtype)
    print("targets.shape:", targets.shape)
    print("targets.dtype:", targets.dtype)
    print("inputs[0]:", inputs[0])
```

```
    print("targets[0]:", targets[0])
    break

#Our model-building utility

#With so little training data I do not have enough for 10000 tokens?

def get_model(max_tokens=tokenSize, hidden_dim=16):
    inputs = keras.Input(shape=(max_tokens,))
    x = layers.Dense(hidden_dim, activation="relu")(inputs)
    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"])
    return model

#Training and testing the binary unigram model

model = get_model()
model.summary()
callbacks = [
    keras.callbacks.ModelCheckpoint("binary_1gram{epoch:02d}.keras",
                                    save_best_only=True)
]

model.fit(binary_1gram_train_ds.cache(),
          validation_data=binary_1gram_val_ds.cache(),
          epochs=10,
          callbacks=callbacks)
```

```
inputs.shape: (32, 2771)
inputs.dtype: <dtype: 'int64'>
targets.shape: (32,)
targets.dtype: <dtype: 'int32'>
inputs[0]: tf.Tensor([0 1 1 … 0 0 0], shape=(2771,), dtype=int64)
targets[0]: tf.Tensor(1, shape=(), dtype=int32)
```

```
Model: "functional_1"
```

Layer (type)	Output Shape	Param #
input_layer_1 (InputLayer)	(None, 2771)	0
dense_2 (Dense)	(None, 16)	44,352

```
 dropout_1 (Dropout)             (None, 16)                    0

 dense_3 (Dense)                 (None, 1)                     17


Total params: 44,369 (173.32 KB)

Trainable params: 44,369 (173.32 KB)

Non-trainable params: 0 (0.00 B)

Epoch 1/10
2/2             4s 2s/step -
accuracy: 0.7433 - loss: 0.6375 - val_accuracy: 0.5227 - val_loss: 0.6907
Epoch 2/10
2/2             1s 643ms/step -
accuracy: 0.7462 - loss: 0.5791 - val_accuracy: 0.5419 - val_loss: 0.6839
Epoch 3/10
2/2             1s 621ms/step -
accuracy: 0.8175 - loss: 0.4892 - val_accuracy: 0.5405 - val_loss: 0.6845
Epoch 4/10
2/2             1s 649ms/step -
accuracy: 0.8175 - loss: 0.4740 - val_accuracy: 0.5811 - val_loss: 0.6715
Epoch 5/10
2/2             1s 610ms/step -
accuracy: 0.9525 - loss: 0.3956 - val_accuracy: 0.5797 - val_loss: 0.6691
Epoch 6/10
2/2             1s 602ms/step -
accuracy: 0.9629 - loss: 0.3485 - val_accuracy: 0.5790 - val_loss: 0.6695
Epoch 7/10
2/2             1s 622ms/step -
accuracy: 0.9525 - loss: 0.3240 - val_accuracy: 0.5884 - val_loss: 0.6650
Epoch 8/10
2/2             1s 629ms/step -
accuracy: 0.9629 - loss: 0.3042 - val_accuracy: 0.6031 - val_loss: 0.6587
Epoch 9/10
2/2             1s 656ms/step -
accuracy: 1.0000 - loss: 0.2517 - val_accuracy: 0.6110 - val_loss: 0.6561
Epoch 10/10
2/2             1s 631ms/step -
accuracy: 0.9762 - loss: 0.2649 - val_accuracy: 0.6196 - val_loss: 0.6520
```

[ ]: <keras.src.callbacks.history.History at 0x79a620b4fed0>

Testing binary_1gram

[ ]:
```
model = keras.models.load_model("binary_1gram10.keras")
print(f"Test acc: {model.evaluate(binary_1gram_test_ds)[1]:.3f}")
```

```
782/782              2s 2ms/step -
accuracy: 0.6212 - loss: 0.6524
Test acc: 0.622
```

Bigrams

[ ]:
```
#Bigrams with binary encoding
#Configuring the TextVectorization layer to return bigrams

text_vectorization = TextVectorization(
    ngrams=2,
    max_tokens=tokenSize,
    output_mode="multi_hot",
)

#Training and testing the binary bigram model

text_vectorization.adapt(text_only_train_ds)
binary_2gram_train_ds = train_ds.map(
    lambda x, y: (text_vectorization(x), y),
    num_parallel_calls=4)
binary_2gram_val_ds = val_ds.map(
    lambda x, y: (text_vectorization(x), y),
    num_parallel_calls=4)
binary_2gram_test_ds = test_ds.map(
    lambda x, y: (text_vectorization(x), y),
    num_parallel_calls=4)

model = get_model()
model.summary()
callbacks = [
    keras.callbacks.ModelCheckpoint("binary_2gram{epoch:02d}.keras",
                                    save_best_only=True)
]
model.fit(binary_2gram_train_ds.cache(),
          validation_data=binary_2gram_val_ds.cache(),
          epochs=10,
          callbacks=callbacks)
```

```
Model: "functional_2"
```

Layer (type)	Output Shape	Param #
input_layer_2 (InputLayer)	(None, 2771)	0

```
 dense_4 (Dense)                 (None, 16)                    44,352

 dropout_2 (Dropout)             (None, 16)                         0

 dense_5 (Dense)                 (None, 1)                         17


 Total params: 44,369 (173.32 KB)

 Trainable params: 44,369 (173.32 KB)

 Non-trainable params: 0 (0.00 B)

Epoch 1/10
2/2             3s 2s/step -
accuracy: 0.4288 - loss: 0.7595 - val_accuracy: 0.5139 - val_loss: 0.6963
Epoch 2/10
2/2             1s 625ms/step -
accuracy: 0.7167 - loss: 0.6158 - val_accuracy: 0.5357 - val_loss: 0.6887
Epoch 3/10
2/2             1s 634ms/step -
accuracy: 0.8071 - loss: 0.4952 - val_accuracy: 0.5422 - val_loss: 0.6864
Epoch 4/10
2/2             1s 634ms/step -
accuracy: 0.9287 - loss: 0.4509 - val_accuracy: 0.5653 - val_loss: 0.6785
Epoch 5/10
2/2             1s 634ms/step -
accuracy: 0.9525 - loss: 0.3550 - val_accuracy: 0.5754 - val_loss: 0.6746
Epoch 6/10
2/2             1s 614ms/step -
accuracy: 1.0000 - loss: 0.2999 - val_accuracy: 0.5740 - val_loss: 0.6750
Epoch 7/10
2/2             1s 642ms/step -
accuracy: 0.9021 - loss: 0.3184 - val_accuracy: 0.5998 - val_loss: 0.6656
Epoch 8/10
2/2             1s 637ms/step -
accuracy: 0.9287 - loss: 0.2567 - val_accuracy: 0.6067 - val_loss: 0.6620
Epoch 9/10
2/2             1s 635ms/step -
accuracy: 0.9525 - loss: 0.2514 - val_accuracy: 0.6130 - val_loss: 0.6538
Epoch 10/10
2/2             1s 618ms/step -
accuracy: 1.0000 - loss: 0.1964 - val_accuracy: 0.6183 - val_loss: 0.6545
```

[ ]:
```
<keras.src.callbacks.history.History at 0x79a61c2daf50>
```

Testing Bigram

[ ]:
```
model = keras.models.load_model("binary_2gram09.keras")
print(f"Test acc: {model.evaluate(binary_2gram_test_ds)[1]:.3f}")
```

```
782/782             2s 2ms/step -
accuracy: 0.6206 - loss: 0.6535
Test acc: 0.618
```

Bigram using TF-IDF encoding

[ ]:
```
#Configuring TextVectorization to return TF-IDF-weighted outputs

text_vectorization = TextVectorization(
    ngrams=2,
    max_tokens=tokenSize,
    output_mode="tf_idf",
)

#Training and testing the TF-IDF bigram model

text_vectorization.adapt(text_only_train_ds)

tfidf_2gram_train_ds = train_ds.map(
    lambda x, y: (text_vectorization(x), y),
    num_parallel_calls=4)
tfidf_2gram_val_ds = val_ds.map(
    lambda x, y: (text_vectorization(x), y),
    num_parallel_calls=4)
tfidf_2gram_test_ds = test_ds.map(
    lambda x, y: (text_vectorization(x), y),
    num_parallel_calls=4)

model = get_model()
model.summary()

callbacks = [
    keras.callbacks.ModelCheckpoint("tfidf_2gram{epoch:02d}.keras",
                                    save_best_only=True)
]
model.fit(tfidf_2gram_train_ds.cache(),
          validation_data=tfidf_2gram_val_ds.cache(),
          epochs=10,
          callbacks=callbacks)
```

```
Model: "functional_3"
```

Layer (type)	Output Shape	Param #
input_layer_3 (InputLayer)	(None, 2771)	0
dense_6 (Dense)	(None, 16)	44,352
dropout_3 (Dropout)	(None, 16)	0
dense_7 (Dense)	(None, 1)	17

Total params: 44,369 (173.32 KB)

Trainable params: 44,369 (173.32 KB)

Non-trainable params: 0 (0.00 B)

```
Epoch 1/10
2/2             3s 2s/step -
accuracy: 0.5104 - loss: 8.9538 - val_accuracy: 0.5000 - val_loss: 6.9353
Epoch 2/10
2/2             1s 608ms/step -
accuracy: 0.5504 - loss: 4.0754 - val_accuracy: 0.5000 - val_loss: 4.8370
Epoch 3/10
2/2             1s 601ms/step -
accuracy: 0.5446 - loss: 4.3268 - val_accuracy: 0.5001 - val_loss: 3.3717
Epoch 4/10
2/2             1s 609ms/step -
accuracy: 0.6662 - loss: 1.6967 - val_accuracy: 0.5003 - val_loss: 2.7141
Epoch 5/10
2/2             1s 610ms/step -
accuracy: 0.5504 - loss: 2.3754 - val_accuracy: 0.5004 - val_loss: 2.5636
Epoch 6/10
2/2             1s 605ms/step -
accuracy: 0.7404 - loss: 0.7364 - val_accuracy: 0.5012 - val_loss: 2.2393
Epoch 7/10
2/2             1s 599ms/step -
accuracy: 0.6542 - loss: 1.1727 - val_accuracy: 0.5029 - val_loss: 1.9939
Epoch 8/10
2/2             1s 604ms/step -
accuracy: 0.7508 - loss: 0.5280 - val_accuracy: 0.5057 - val_loss: 1.8550
Epoch 9/10
2/2             1s 590ms/step -
accuracy: 0.7937 - loss: 0.5456 - val_accuracy: 0.5038 - val_loss: 1.9760
Epoch 10/10
```

```
2/2             1s 591ms/step -
accuracy: 0.8442 - loss: 0.4268 - val_accuracy: 0.5028 - val_loss: 2.2357
```

[ ]:
```
<keras.src.callbacks.history.History at 0x79a5fea17bd0>
```

Testing Bigram with TF-IDF

[ ]:
```
model = keras.models.load_model("tfidf_2gram08.keras")
print(f"Test acc: {model.evaluate(tfidf_2gram_test_ds)[1]:.3f}")
```

```
782/782             2s 2ms/step -
accuracy: 0.5039 - loss: 1.7967
Test acc: 0.505
```

Sequence model with one hot encoding

[ ]:
```
#Preparing integer sequence datasets

max_length = 150
max_tokens = 10000
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)

#A sequence model built on one-hot encoded vector sequences


inputs = keras.Input(shape=(None,), dtype="int64")
#embedded = tf.keras.layers.Lambda(lambda x: tf.one_hot(x, depth=max_tokens),␣
 ↪output_shape=(None, max_tokens))(inputs)
embedded = keras.ops.one_hot(inputs, num_classes=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()

#Training a first basic sequence model

callbacks = [
    keras.callbacks.ModelCheckpoint("one_hot_bidir_lstm{epoch:02d}.keras",
                                    save_best_only=True)
]
model.fit(int_train_ds, validation_data=int_val_ds, epochs=10,␣
 ↪callbacks=callbacks)


#model.evaluate(int_test_ds)
```

Model: "functional_4"

Layer (type)	Output Shape	Param #
input_layer_4 (InputLayer)	(None, None)	0
one_hot (OneHot)	(None, None, 10000)	0
bidirectional (Bidirectional)	(None, 64)	2,568,448
dropout_4 (Dropout)	(None, 64)	0
dense_8 (Dense)	(None, 1)	65

Total params: 2,568,513 (9.80 MB)

Trainable params: 2,568,513 (9.80 MB)

Non-trainable params: 0 (0.00 B)

```
Epoch 1/10
2/2             6s 3s/step -
accuracy: 0.4838 - loss: 0.6939 - val_accuracy: 0.5077 - val_loss: 0.6930
Epoch 2/10
```

```
2/2             3s 3s/step -
accuracy: 0.6721 - loss: 0.6911 - val_accuracy: 0.5150 - val_loss: 0.6929
Epoch 3/10
2/2             3s 3s/step -
accuracy: 0.8042 - loss: 0.6866 - val_accuracy: 0.5178 - val_loss: 0.6929
Epoch 4/10
2/2             3s 3s/step -
accuracy: 0.7596 - loss: 0.6836 - val_accuracy: 0.5203 - val_loss: 0.6927
Epoch 5/10
2/2             3s 3s/step -
accuracy: 0.8754 - loss: 0.6809 - val_accuracy: 0.5182 - val_loss: 0.6927
Epoch 6/10
2/2             3s 3s/step -
accuracy: 0.8546 - loss: 0.6758 - val_accuracy: 0.5188 - val_loss: 0.6926
Epoch 7/10
2/2             3s 3s/step -
accuracy: 0.9392 - loss: 0.6705 - val_accuracy: 0.5203 - val_loss: 0.6924
Epoch 8/10
2/2             3s 3s/step -
accuracy: 0.9125 - loss: 0.6665 - val_accuracy: 0.5213 - val_loss: 0.6922
Epoch 9/10
2/2             3s 3s/step -
accuracy: 0.9021 - loss: 0.6633 - val_accuracy: 0.5245 - val_loss: 0.6920
Epoch 10/10
2/2             3s 3s/step -
accuracy: 0.9258 - loss: 0.6540 - val_accuracy: 0.5293 - val_loss: 0.6917
```

[ ]:
```
<keras.src.callbacks.history.History at 0x79a5d39a9ed0>
```

Testing Sequence with one hot encoding

[ ]:
```
model = keras.models.load_model("one_hot_bidir_lstm10.keras", safe_mode=False)

print(f"Test acc: {model.evaluate(int_test_ds)[1]:.3f}")
```

```
782/782             7s 8ms/step -
accuracy: 0.5265 - loss: 0.6919
Test acc: 0.526
```

Embedding Layer from scratch

[ ]:
```
#Understanding word embeddings
#Learning word embeddings with the Embedding layer
#Instantiating an Embedding layer

embedding_layer = layers.Embedding(input_dim=max_tokens, output_dim=256)

#Model that uses an Embedding layer trained from scratch
```

```
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{epoch:02d}.keras",
                                    save_best_only=True)
]
model.fit(int_train_ds, validation_data=int_val_ds, epochs=10,␣
 ↪callbacks=callbacks)
```

Model: "functional_5"

Layer (type)	Output Shape	Param #
input_layer_5 (InputLayer)	(None, None)	0
embedding_1 (Embedding)	(None, None, 256)	2,560,000
bidirectional_1 (Bidirectional)	(None, 64)	73,984
dropout_5 (Dropout)	(None, 64)	0
dense_9 (Dense)	(None, 1)	65

Total params: 2,634,049 (10.05 MB)

Trainable params: 2,634,049 (10.05 MB)

Non-trainable params: 0 (0.00 B)

```
Epoch 1/10
2/2             5s 2s/step -
accuracy: 0.5846 - loss: 0.6915 - val_accuracy: 0.5284 - val_loss: 0.6924
Epoch 2/10
```

```
2/2             2s 2s/step -
accuracy: 0.6662 - loss: 0.6811 - val_accuracy: 0.5212 - val_loss: 0.6920
Epoch 3/10
2/2             2s 2s/step -
accuracy: 0.8042 - loss: 0.6602 - val_accuracy: 0.5233 - val_loss: 0.6917
Epoch 4/10
2/2             2s 2s/step -
accuracy: 0.7538 - loss: 0.6463 - val_accuracy: 0.5250 - val_loss: 0.6912
Epoch 5/10
2/2             2s 2s/step -
accuracy: 0.8012 - loss: 0.6292 - val_accuracy: 0.5247 - val_loss: 0.6904
Epoch 6/10
2/2             2s 2s/step -
accuracy: 0.9496 - loss: 0.6007 - val_accuracy: 0.5275 - val_loss: 0.6900
Epoch 7/10
2/2             2s 2s/step -
accuracy: 0.9125 - loss: 0.5760 - val_accuracy: 0.5364 - val_loss: 0.6889
Epoch 8/10
2/2             2s 2s/step -
accuracy: 0.9333 - loss: 0.5403 - val_accuracy: 0.5296 - val_loss: 0.7019
Epoch 9/10
2/2             2s 2s/step -
accuracy: 0.8650 - loss: 0.4850 - val_accuracy: 0.5588 - val_loss: 0.7010
Epoch 10/10
2/2             2s 2s/step -
accuracy: 0.8696 - loss: 0.4584 - val_accuracy: 0.5485 - val_loss: 0.6943
```

[ ]:
```
<keras.src.callbacks.history.History at 0x79a5a1927ed0>
```

Testing embedding layer from scratch

[ ]:
```
model = keras.models.load_model("embeddings_bidir_gru07.keras")
print(f"Test acc: {model.evaluate(int_test_ds)[1]:.3f}")
```

```
782/782             5s 6ms/step -
accuracy: 0.5337 - loss: 0.6906
Test acc: 0.532
```

Adding masking

[ ]:
```
#Understanding padding and masking
#Using an Embedding layer with masking enabled

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

callbacks = [
    keras.callbacks.ModelCheckpoint("embeddings_bidir_lstm_with_masking{epoch:
 ↪02d}.keras",
                                    save_best_only=True)
]
model.fit(int_train_ds, validation_data=int_val_ds, epochs=10,␣
 ↪callbacks=callbacks)
```

Model: "functional_6"

Layer (type)	Output Shape	Param #	Connected to
input_layer_6 (InputLayer)	(None, None)	0	-
embedding_2 (Embedding)	(None, None, 256)	2,560,000	input_layer_6[0]…
not_equal (NotEqual)	(None, None)	0	input_layer_6[0]…
bidirectional_2 (Bidirectional)	(None, 64)	73,984	embedding_2[0][0]… not_equal[0][0]
dropout_6 (Dropout)	(None, 64)	0	bidirectional_2[…
dense_10 (Dense)	(None, 1)	65	dropout_6[0][0]

Total params: 2,634,049 (10.05 MB)

Trainable params: 2,634,049 (10.05 MB)

Non-trainable params: 0 (0.00 B)

```
Epoch 1/10
2/2             5s 3s/step -
```

```
accuracy: 0.4392 - loss: 0.6946 - val_accuracy: 0.5046 - val_loss: 0.6930
Epoch 2/10
2/2             2s 2s/step -
accuracy: 0.7300 - loss: 0.6804 - val_accuracy: 0.5081 - val_loss: 0.6927
Epoch 3/10
2/2             2s 2s/step -
accuracy: 0.8308 - loss: 0.6675 - val_accuracy: 0.5120 - val_loss: 0.6924
Epoch 4/10
2/2             2s 2s/step -
accuracy: 0.9762 - loss: 0.6501 - val_accuracy: 0.5122 - val_loss: 0.6922
Epoch 5/10
2/2             2s 2s/step -
accuracy: 0.9525 - loss: 0.6289 - val_accuracy: 0.5191 - val_loss: 0.6919
Epoch 6/10
2/2             2s 2s/step -
accuracy: 0.9525 - loss: 0.6128 - val_accuracy: 0.5172 - val_loss: 0.6917
Epoch 7/10
2/2             2s 2s/step -
accuracy: 0.9867 - loss: 0.5893 - val_accuracy: 0.5223 - val_loss: 0.6915
Epoch 8/10
2/2             2s 2s/step -
accuracy: 1.0000 - loss: 0.5442 - val_accuracy: 0.5241 - val_loss: 0.6914
Epoch 9/10
2/2             2s 2s/step -
accuracy: 1.0000 - loss: 0.5091 - val_accuracy: 0.5346 - val_loss: 0.6914
Epoch 10/10
2/2             2s 2s/step -
accuracy: 1.0000 - loss: 0.4572 - val_accuracy: 0.5298 - val_loss: 0.6943
```

[ ]:
```
<keras.src.callbacks.history.History at 0x79a5341ffed0>
```

Testing model that uses masking and padding

[ ]:
```
model = keras.models.load_model("embeddings_bidir_lstm_with_masking09.keras")
print(f"Test acc: {model.evaluate(int_test_ds)[1]:.3f}")
```

```
782/782             6s 6ms/step -
accuracy: 0.5211 - loss: 0.6927
Test acc: 0.525
```

Downloading GloVe word-embeddings

[ ]:
```
#Using pretrained word embeddings
!wget http://nlp.stanford.edu/data/glove.6B.zip
!unzip -q glove.6B.zip
```

```
--2025-04-20 15:09:46--  http://nlp.stanford.edu/data/glove.6B.zip
Resolving nlp.stanford.edu (nlp.stanford.edu)… 171.64.67.140
Connecting to nlp.stanford.edu (nlp.stanford.edu)|171.64.67.140|:80…
```

```
connected.
HTTP request sent, awaiting response… 302 Found
Location: https://nlp.stanford.edu/data/glove.6B.zip [following]
--2025-04-20 15:09:46--  https://nlp.stanford.edu/data/glove.6B.zip
Connecting to nlp.stanford.edu (nlp.stanford.edu)|171.64.67.140|:443…
connected.
HTTP request sent, awaiting response… 301 Moved Permanently
Location: https://downloads.cs.stanford.edu/nlp/data/glove.6B.zip [following]
--2025-04-20 15:09:46--  https://downloads.cs.stanford.edu/nlp/data/glove.6B.zip
Resolving downloads.cs.stanford.edu (downloads.cs.stanford.edu)… 171.64.64.22
Connecting to downloads.cs.stanford.edu
(downloads.cs.stanford.edu)|171.64.64.22|:443… connected.
HTTP request sent, awaiting response… 200 OK
Length: 862182613 (822M) [application/zip]
Saving to: ‘glove.6B.zip’

glove.6B.zip        100%[===================>] 822.24M  5.00MB/s    in 2m 39s

2025-04-20 15:12:25 (5.18 MB/s) - ‘glove.6B.zip’ saved [862182613/862182613]
```

Preparing GloVe embeddings for use

[ ]:
```python
#Parsing the GloVe word-embeddings file

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.")

#Preparing the GloVe word-embeddings matrix

embedding_dim = 100

vocabulary = text_vectorization.get_vocabulary()
word_index = dict(zip(vocabulary, range(len(vocabulary))))

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,
)
```

```
Found 400000 word vectors.
```

Using Pretrained embedding

[ ]:
```
#Model that uses a pretrained Embedding layer

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{epoch:02d}.
 ↪keras",
                                    save_best_only=True)
]
model.fit(int_train_ds, validation_data=int_val_ds, epochs=10,␣
 ↪callbacks=callbacks)
```

```
Model: "functional_8"
```

Layer (type)	Output Shape	Param #	Connected to
input_layer_8 (InputLayer)	(None, None)	0	-
embedding_3 (Embedding)	(None, None, 100)	1,000,000	input_layer_8[0]…
not_equal_4 (NotEqual)	(None, None)	0	input_layer_8[0]…

```
 bidirectional_4       (None, 64)              34,048   embedding_3[1][0…
 (Bidirectional)                                        not_equal_4[0][0]

 dropout_8 (Dropout)   (None, 64)                   0   bidirectional_4[…

 dense_12 (Dense)      (None, 1)                   65   dropout_8[0][0]


 Total params: 1,034,113 (3.94 MB)

 Trainable params: 34,113 (133.25 KB)

 Non-trainable params: 1,000,000 (3.81 MB)

Epoch 1/10
2/2             5s 4s/step -
accuracy: 0.4704 - loss: 0.7567 - val_accuracy: 0.5021 - val_loss: 0.6971
Epoch 2/10
2/2             4s 4s/step -
accuracy: 0.5208 - loss: 0.6717 - val_accuracy: 0.5211 - val_loss: 0.6945
Epoch 3/10
2/2             2s 2s/step -
accuracy: 0.4971 - loss: 0.6807 - val_accuracy: 0.5295 - val_loss: 0.6946
Epoch 4/10
2/2             3s 3s/step -
accuracy: 0.5742 - loss: 0.6760 - val_accuracy: 0.5379 - val_loss: 0.6934
Epoch 5/10
2/2             2s 2s/step -
accuracy: 0.8250 - loss: 0.5773 - val_accuracy: 0.5254 - val_loss: 0.6940
Epoch 6/10
2/2             2s 2s/step -
accuracy: 0.6587 - loss: 0.6281 - val_accuracy: 0.5355 - val_loss: 0.6945
Epoch 7/10
2/2             2s 2s/step -
accuracy: 0.6217 - loss: 0.6263 - val_accuracy: 0.5271 - val_loss: 0.6965
Epoch 8/10
2/2             2s 2s/step -
accuracy: 0.7300 - loss: 0.5876 - val_accuracy: 0.5322 - val_loss: 0.6969
Epoch 9/10
2/2             2s 2s/step -
accuracy: 0.7746 - loss: 0.5828 - val_accuracy: 0.5221 - val_loss: 0.7026
Epoch 10/10
2/2             2s 2s/step -
accuracy: 0.7804 - loss: 0.5587 - val_accuracy: 0.5278 - val_loss: 0.7020
```

[ ]: <keras.src.callbacks.history.History at 0x79a5bff79290>

Testing pretrained embedding

[ ]:
```
model = keras.models.load_model("glove_embeddings_sequence_model04.keras")
print(f"Test acc: {model.evaluate(int_test_ds)[1]:.3f}")
```

```
782/782             6s 6ms/step -
accuracy: 0.5350 - loss: 0.6948
Test acc: 0.535
```

Transformer

[ ]:
```
#Vectorizing the data

max_length = 150
max_tokens = 10000
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)
#Transformer encoder implemented as a subclassed Layer

class TransformerEncoder(layers.Layer):
    def __init__(self, embed_dim, dense_dim, num_heads, **kwargs):
        super().__init__(**kwargs)
        self.embed_dim = embed_dim
        self.dense_dim = dense_dim
        self.num_heads = num_heads
        self.attention = layers.MultiHeadAttention(
            num_heads=num_heads, key_dim=embed_dim)
        self.dense_proj = keras.Sequential(
            [layers.Dense(dense_dim, activation="relu"),
             layers.Dense(embed_dim),]
        )
```

```
        self.layernorm_1 = layers.LayerNormalization()
        self.layernorm_2 = layers.LayerNormalization()

    def call(self, inputs, mask=None):
        if mask is not None:
            mask = mask[:, tf.newaxis, :]
        attention_output = self.attention(
            inputs, inputs, attention_mask=mask)
        proj_input = self.layernorm_1(inputs + attention_output)
        proj_output = self.dense_proj(proj_input)
        return self.layernorm_2(proj_input + proj_output)

    def get_config(self):
        config = super().get_config()
        config.update({
            "embed_dim": self.embed_dim,
            "num_heads": self.num_heads,
            "dense_dim": self.dense_dim,
        })
        return config
#Using the Transformer encoder for text classification

vocab_size = 10000
embed_dim = 256
num_heads = 2
dense_dim = 32

inputs = keras.Input(shape=(None,), dtype="int64")
#x = embedding_layer(inputs)
x = layers.Embedding(vocab_size, embed_dim)(inputs)
x = TransformerEncoder(embed_dim, dense_dim, num_heads)(x)
x = layers.GlobalMaxPooling1D()(x)
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()
#Training and evaluating the Transformer encoder based model

callbacks = [
    keras.callbacks.ModelCheckpoint("transformer_encoder{epoch:02d}.keras",
                                    save_best_only=True)
]
model.fit(int_train_ds, validation_data=int_val_ds, epochs=20,␣
 ↪callbacks=callbacks)
```

```
Model: "functional_8"
```

Layer (type)	Output Shape	Param #
input_layer_7 (InputLayer)	(None, None)	0
embedding_4 (Embedding)	(None, None, 256)	2,560,000
transformer_encoder (TransformerEncoder)	(None, None, 256)	543,776
global_max_pooling1d (GlobalMaxPooling1D)	(None, 256)	0
dropout_8 (Dropout)	(None, 256)	0
dense_12 (Dense)	(None, 1)	257

```
Total params: 3,104,033 (11.84 MB)

Trainable params: 3,104,033 (11.84 MB)

Non-trainable params: 0 (0.00 B)

Epoch 1/20
469/469             58s 73ms/step -
accuracy: 0.6475 - loss: 0.7964 - val_accuracy: 0.8120 - val_loss: 0.4147
Epoch 2/20
469/469             3s 7ms/step -
accuracy: 0.8030 - loss: 0.4326 - val_accuracy: 0.8229 - val_loss: 0.3852
Epoch 3/20
469/469             3s 6ms/step -
accuracy: 0.8286 - loss: 0.3797 - val_accuracy: 0.8091 - val_loss: 0.4239
Epoch 4/20
469/469             3s 6ms/step -
accuracy: 0.8492 - loss: 0.3507 - val_accuracy: 0.8278 - val_loss: 0.3877
Epoch 5/20
469/469             3s 6ms/step -
accuracy: 0.8657 - loss: 0.3165 - val_accuracy: 0.8345 - val_loss: 0.3720
Epoch 6/20
469/469             3s 6ms/step -
accuracy: 0.8755 - loss: 0.2929 - val_accuracy: 0.8362 - val_loss: 0.3863
Epoch 7/20
```

```
469/469             3s 6ms/step -
accuracy: 0.8917 - loss: 0.2650 - val_accuracy: 0.8420 - val_loss: 0.3838
Epoch 8/20
469/469             3s 6ms/step -
accuracy: 0.9113 - loss: 0.2274 - val_accuracy: 0.8426 - val_loss: 0.3781
Epoch 9/20
469/469             3s 6ms/step -
accuracy: 0.9202 - loss: 0.2067 - val_accuracy: 0.8419 - val_loss: 0.4218
Epoch 10/20
469/469             3s 6ms/step -
accuracy: 0.9310 - loss: 0.1814 - val_accuracy: 0.8311 - val_loss: 0.4467
Epoch 11/20
469/469             3s 6ms/step -
accuracy: 0.9416 - loss: 0.1550 - val_accuracy: 0.8303 - val_loss: 0.4882
Epoch 12/20
469/469             3s 6ms/step -
accuracy: 0.9524 - loss: 0.1311 - val_accuracy: 0.8369 - val_loss: 0.5257
Epoch 13/20
469/469             3s 6ms/step -
accuracy: 0.9631 - loss: 0.1078 - val_accuracy: 0.8255 - val_loss: 0.6154
Epoch 14/20
469/469             3s 6ms/step -
accuracy: 0.9691 - loss: 0.0911 - val_accuracy: 0.8333 - val_loss: 0.6085
Epoch 15/20
469/469             3s 6ms/step -
accuracy: 0.9724 - loss: 0.0800 - val_accuracy: 0.8277 - val_loss: 0.6675
Epoch 16/20
469/469             3s 7ms/step -
accuracy: 0.9806 - loss: 0.0577 - val_accuracy: 0.8333 - val_loss: 0.7185
Epoch 17/20
469/469             3s 6ms/step -
accuracy: 0.9818 - loss: 0.0531 - val_accuracy: 0.8314 - val_loss: 0.7806
Epoch 18/20
469/469             3s 6ms/step -
accuracy: 0.9837 - loss: 0.0485 - val_accuracy: 0.8269 - val_loss: 0.8679
Epoch 19/20
469/469             3s 6ms/step -
accuracy: 0.9876 - loss: 0.0377 - val_accuracy: 0.8244 - val_loss: 0.9400
Epoch 20/20
469/469             3s 6ms/step -
accuracy: 0.9887 - loss: 0.0319 - val_accuracy: 0.8272 - val_loss: 0.9225
```

[ ]: `<keras.src.callbacks.history.History at 0x7adb1ff70610>`

Testing transformer

[ ]:
```
model = keras.models.load_model(
    "transformer_encoder05.keras",
    custom_objects={"TransformerEncoder": TransformerEncoder})
print(f"Test acc: {model.evaluate(int_test_ds)[1]:.3f}")
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/layer.py:393:
UserWarning: `build()` was called on layer 'transformer_encoder', however the
layer does not have a `build()` method implemented and it looks like it has
unbuilt state. This will cause the layer to be marked as built, despite not
being actually built, which may cause failures down the line. Make sure to
implement a proper `build()` method.
  warnings.warn(
782/782            8s 9ms/step -
accuracy: 0.8304 - loss: 0.3765
Test acc: 0.830
```

Transformer with positional embedding

[ ]:
```
#Using positional encoding to re-inject order information
#Implementing positional embedding as a subclassed layer

class PositionalEmbedding(layers.Layer):
    def __init__(self, sequence_length, input_dim, output_dim, **kwargs):
        super().__init__(**kwargs)
        self.token_embeddings = layers.Embedding(
            input_dim=input_dim, output_dim=output_dim)
        self.position_embeddings = layers.Embedding(
            input_dim=sequence_length, output_dim=output_dim)
        self.sequence_length = sequence_length
        self.input_dim = input_dim
        self.output_dim = output_dim

    def call(self, inputs):
        length = tf.shape(inputs)[-1]
        positions = tf.range(start=0, limit=length, delta=1)
        embedded_tokens = self.token_embeddings(inputs)
        embedded_positions = self.position_embeddings(positions)
        return embedded_tokens + embedded_positions

    def compute_mask(self, inputs, mask=None):
        return keras.ops.not_equal(inputs, 0)

    def get_config(self):
        config = super().get_config()
        config.update({
            "output_dim": self.output_dim,
            "sequence_length": self.sequence_length,
```

```
            "input_dim": self.input_dim,
        })
        return config
#Putting it all together: A text-classification Transformer
#Combining the Transformer encoder with positional embedding

vocab_size = 10000
sequence_length = 150
embed_dim = 256
num_heads = 2
dense_dim = 32

inputs = keras.Input(shape=(None,), dtype="int64")
x = PositionalEmbedding(sequence_length, vocab_size, embed_dim)(inputs)
x = TransformerEncoder(embed_dim, dense_dim, num_heads)(x)
x = layers.GlobalMaxPooling1D()(x)
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("full_transformer_encoder{epoch:02d}.keras",
                                    save_best_only=True)
]
model.fit(int_train_ds, validation_data=int_val_ds, epochs=20,␣
 ↪callbacks=callbacks)
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/layer.py:938:
UserWarning: Layer 'transformer_encoder_1' (of type TransformerEncoder) was
passed an input with a mask attached to it. However, this layer does not support
masking and will therefore destroy the mask information. Downstream layers will
not see the mask.
  warnings.warn(
```

Model: "functional_11"

Layer (type)	Output Shape	Param #	Connected to
input_layer_10 (InputLayer)	(None, None)	0	-
positional_embeddi…	(None, None, 256)	2,598,400	input_layer_10[0…

```
 (PositionalEmbeddi…

 not_equal_4           (None, None)                0   input_layer_10[0…
 (NotEqual)

 transformer_encode…   (None, None, 256)      543,776   positional_embed…
 (TransformerEncode…                                    not_equal_4[0][0]

 global_max_pooling…   (None, 256)                 0   transformer_enco…
 (GlobalMaxPooling1…

 dropout_11            (None, 256)                 0   global_max_pooli…
 (Dropout)

 dense_17 (Dense)      (None, 1)                 257   dropout_11[0][0]


 Total params: 3,142,433 (11.99 MB)

 Trainable params: 3,142,433 (11.99 MB)

 Non-trainable params: 0 (0.00 B)

Epoch 1/20
469/469             30s 38ms/step -
accuracy: 0.6043 - loss: 0.8663 - val_accuracy: 0.8066 - val_loss: 0.4306
Epoch 2/20
469/469             3s 6ms/step -
accuracy: 0.7977 - loss: 0.4380 - val_accuracy: 0.8403 - val_loss: 0.3749
Epoch 3/20
469/469             3s 6ms/step -
accuracy: 0.8413 - loss: 0.3583 - val_accuracy: 0.8266 - val_loss: 0.4148
Epoch 4/20
469/469             3s 6ms/step -
accuracy: 0.8680 - loss: 0.3078 - val_accuracy: 0.7900 - val_loss: 0.5438
Epoch 5/20
469/469             3s 6ms/step -
accuracy: 0.8901 - loss: 0.2679 - val_accuracy: 0.8493 - val_loss: 0.4226
Epoch 6/20
469/469             3s 6ms/step -
accuracy: 0.9076 - loss: 0.2313 - val_accuracy: 0.8478 - val_loss: 0.4241
Epoch 7/20
469/469             3s 6ms/step -
accuracy: 0.9264 - loss: 0.1944 - val_accuracy: 0.8377 - val_loss: 0.5340
Epoch 8/20
469/469             3s 6ms/step -
```

```
accuracy: 0.9378 - loss: 0.1577 - val_accuracy: 0.8444 - val_loss: 0.5812
Epoch 9/20
469/469             3s 6ms/step -
accuracy: 0.9556 - loss: 0.1160 - val_accuracy: 0.8325 - val_loss: 0.6704
Epoch 10/20
469/469             3s 6ms/step -
accuracy: 0.9687 - loss: 0.0865 - val_accuracy: 0.8378 - val_loss: 0.7235
Epoch 11/20
469/469             3s 6ms/step -
accuracy: 0.9794 - loss: 0.0621 - val_accuracy: 0.8382 - val_loss: 0.8978
Epoch 12/20
469/469             3s 6ms/step -
accuracy: 0.9837 - loss: 0.0465 - val_accuracy: 0.8224 - val_loss: 0.8137
Epoch 13/20
469/469             3s 6ms/step -
accuracy: 0.9857 - loss: 0.0392 - val_accuracy: 0.8327 - val_loss: 0.9521
Epoch 14/20
469/469             3s 5ms/step -
accuracy: 0.9930 - loss: 0.0253 - val_accuracy: 0.8318 - val_loss: 1.1659
Epoch 15/20
469/469             3s 5ms/step -
accuracy: 0.9925 - loss: 0.0249 - val_accuracy: 0.8299 - val_loss: 1.1141
Epoch 16/20
469/469             3s 6ms/step -
accuracy: 0.9941 - loss: 0.0177 - val_accuracy: 0.8146 - val_loss: 1.5620
Epoch 17/20
469/469             2s 5ms/step -
accuracy: 0.9944 - loss: 0.0166 - val_accuracy: 0.8309 - val_loss: 1.1767
Epoch 18/20
469/469             2s 5ms/step -
accuracy: 0.9943 - loss: 0.0133 - val_accuracy: 0.8320 - val_loss: 1.2911
Epoch 19/20
469/469             3s 5ms/step -
accuracy: 0.9948 - loss: 0.0169 - val_accuracy: 0.8114 - val_loss: 1.4533
Epoch 20/20
469/469             3s 5ms/step -
accuracy: 0.9955 - loss: 0.0143 - val_accuracy: 0.8360 - val_loss: 1.5275
```

[ ]:
```
<keras.src.callbacks.history.History at 0x7ada181c6990>
```

Testing full transformer

[ ]:
```
model = keras.models.load_model(
    "full_transformer_encoder02.keras",
    custom_objects={"TransformerEncoder": TransformerEncoder,
                    "PositionalEmbedding": PositionalEmbedding})
print(f"Test acc: {model.evaluate(int_test_ds)[1]:.3f}")
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/layer.py:393:
UserWarning: `build()` was called on layer 'positional_embedding', however the
layer does not have a `build()` method implemented and it looks like it has
unbuilt state. This will cause the layer to be marked as built, despite not
being actually built, which may cause failures down the line. Make sure to
implement a proper `build()` method.
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/keras/src/layers/layer.py:393:
UserWarning: `build()` was called on layer 'transformer_encoder_1', however the
layer does not have a `build()` method implemented and it looks like it has
unbuilt state. This will cause the layer to be marked as built, despite not
being actually built, which may cause failures down the line. Make sure to
implement a proper `build()` method.
  warnings.warn(
782/782            6s 6ms/step -
accuracy: 0.8326 - loss: 0.3761
Test acc: 0.830
```

Transformer with pretrained embedding

[ ]:
```python
#Vectorizing the data

max_length = 150
max_tokens = 10000
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)
#Transformer encoder implemented as a subclassed Layer

class TransformerEncoder(layers.Layer):
    def __init__(self, embed_dim, dense_dim, num_heads, **kwargs):
        super().__init__(**kwargs)
        self.embed_dim = embed_dim
        self.dense_dim = dense_dim
```

```
        self.num_heads = num_heads
        self.attention = layers.MultiHeadAttention(
            num_heads=num_heads, key_dim=embed_dim)
        self.dense_proj = keras.Sequential(
            [layers.Dense(dense_dim, activation="relu"),
             layers.Dense(embed_dim),]
        )
        self.layernorm_1 = layers.LayerNormalization()
        self.layernorm_2 = layers.LayerNormalization()

    def call(self, inputs, mask=None):
        if mask is not None:
            mask = mask[:, tf.newaxis, :]
        attention_output = self.attention(
            inputs, inputs, attention_mask=mask)
        proj_input = self.layernorm_1(inputs + attention_output)
        proj_output = self.dense_proj(proj_input)
        return self.layernorm_2(proj_input + proj_output)

    def get_config(self):
        config = super().get_config()
        config.update({
            "embed_dim": self.embed_dim,
            "num_heads": self.num_heads,
            "dense_dim": self.dense_dim,
        })
        return config
#Using the Transformer encoder for text classification

vocab_size = 10000
embed_dim = 100
num_heads = 2
dense_dim = 32

inputs = keras.Input(shape=(None,), dtype="int64")
x = embedding_layer(inputs)
#x = layers.Embedding(vocab_size, embed_dim)(inputs)
x = TransformerEncoder(embed_dim, dense_dim, num_heads)(x)
x = layers.GlobalMaxPooling1D()(x)
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()
#Training and evaluating the Transformer encoder based model
```

```
callbacks = [
    keras.callbacks.ModelCheckpoint("transformer_encoder_pretrained{epoch:02d}.
 ↪keras",
                                    save_best_only=True)
]
model.fit(int_train_ds, validation_data=int_val_ds, epochs=20,␣
 ↪callbacks=callbacks)
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/layer.py:938:
UserWarning: Layer 'transformer_encoder_3' (of type TransformerEncoder) was
passed an input with a mask attached to it. However, this layer does not support
masking and will therefore destroy the mask information. Downstream layers will
not see the mask.
  warnings.warn(
```

Model: "functional_15"

Layer (type)	Output Shape	Param #	Connected to
input_layer_15 (InputLayer)	(None, None)	0	-
embedding_3 (Embedding)	(None, None, 100)	1,000,000	input_layer_15[0…
not_equal_7 (NotEqual)	(None, None)	0	input_layer_15[0…
transformer_encode… (TransformerEncode…	(None, None, 100)	87,632	embedding_3[2][0… not_equal_7[0][0]
global_max_pooling… (GlobalMaxPooling1…	(None, 100)	0	transformer_enco…
dropout_15 (Dropout)	(None, 100)	0	global_max_pooli…
dense_24 (Dense)	(None, 1)	101	dropout_15[0][0]

Total params: 1,087,733 (4.15 MB)

Trainable params: 87,733 (342.71 KB)

```
 Non-trainable params: 1,000,000 (3.81 MB)


Epoch 1/20
469/469             30s 37ms/step -
accuracy: 0.5931 - loss: 0.7832 - val_accuracy: 0.7715 - val_loss: 0.4730
Epoch 2/20
469/469             4s 8ms/step -
accuracy: 0.7905 - loss: 0.4596 - val_accuracy: 0.8086 - val_loss: 0.4129
Epoch 3/20
469/469             4s 8ms/step -
accuracy: 0.7987 - loss: 0.4379 - val_accuracy: 0.8150 - val_loss: 0.4082
Epoch 4/20
469/469             2s 4ms/step -
accuracy: 0.8067 - loss: 0.4272 - val_accuracy: 0.8011 - val_loss: 0.4234
Epoch 5/20
469/469             4s 8ms/step -
accuracy: 0.8108 - loss: 0.4162 - val_accuracy: 0.8236 - val_loss: 0.3912
Epoch 6/20
469/469             2s 5ms/step -
accuracy: 0.8161 - loss: 0.4112 - val_accuracy: 0.8226 - val_loss: 0.3984
Epoch 7/20
469/469             2s 4ms/step -
accuracy: 0.8198 - loss: 0.4045 - val_accuracy: 0.8149 - val_loss: 0.4058
Epoch 8/20
469/469             4s 9ms/step -
accuracy: 0.8199 - loss: 0.4025 - val_accuracy: 0.8211 - val_loss: 0.3901
Epoch 9/20
469/469             2s 4ms/step -
accuracy: 0.8229 - loss: 0.3954 - val_accuracy: 0.8161 - val_loss: 0.4064
Epoch 10/20
469/469             2s 4ms/step -
accuracy: 0.8281 - loss: 0.3888 - val_accuracy: 0.8184 - val_loss: 0.3979
Epoch 11/20
469/469             2s 4ms/step -
accuracy: 0.8334 - loss: 0.3820 - val_accuracy: 0.8164 - val_loss: 0.3982
Epoch 12/20
469/469             2s 5ms/step -
accuracy: 0.8335 - loss: 0.3762 - val_accuracy: 0.8168 - val_loss: 0.3978
Epoch 13/20
469/469             2s 4ms/step -
accuracy: 0.8358 - loss: 0.3713 - val_accuracy: 0.8058 - val_loss: 0.4348
Epoch 14/20
469/469             2s 4ms/step -
accuracy: 0.8422 - loss: 0.3634 - val_accuracy: 0.8209 - val_loss: 0.3988
Epoch 15/20
469/469             2s 4ms/step -
accuracy: 0.8414 - loss: 0.3596 - val_accuracy: 0.8008 - val_loss: 0.4477
```

```
Epoch 16/20
469/469             2s 4ms/step -
accuracy: 0.8465 - loss: 0.3578 - val_accuracy: 0.8143 - val_loss: 0.4074
Epoch 17/20
469/469             2s 4ms/step -
accuracy: 0.8473 - loss: 0.3485 - val_accuracy: 0.8148 - val_loss: 0.4166
Epoch 18/20
469/469             2s 4ms/step -
accuracy: 0.8512 - loss: 0.3418 - val_accuracy: 0.8066 - val_loss: 0.4387
Epoch 19/20
469/469             2s 4ms/step -
accuracy: 0.8578 - loss: 0.3361 - val_accuracy: 0.7989 - val_loss: 0.4434
Epoch 20/20
469/469             2s 4ms/step -
accuracy: 0.8570 - loss: 0.3327 - val_accuracy: 0.8068 - val_loss: 0.4399
```

[ ]:
```
<keras.src.callbacks.history.History at 0x7ada28702d50>
```

Testing transformer with pretrained embedding

[ ]:
```
model = keras.models.load_model(
    "transformer_encoder_pretrained08.keras",
    custom_objects={"TransformerEncoder": TransformerEncoder})
print(f"Test acc: {model.evaluate(int_test_ds)[1]:.3f}")
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/layer.py:393:
UserWarning: `build()` was called on layer 'transformer_encoder_3', however the
layer does not have a `build()` method implemented and it looks like it has
unbuilt state. This will cause the layer to be marked as built, despite not
being actually built, which may cause failures down the line. Make sure to
implement a proper `build()` method.
  warnings.warn(
782/782             5s 5ms/step -
accuracy: 0.8148 - loss: 0.3956
Test acc: 0.813
```

Full transformer with pretrained embedding

[ ]:
```
#Using positional encoding to re-inject order information
#Implementing positional embedding as a subclassed layer

class PositionalEmbedding(layers.Layer):
    def __init__(self, sequence_length, input_dim, output_dim, **kwargs):
        super().__init__(**kwargs)
        self.token_embeddings = layers.Embedding(
            input_dim=input_dim,
            output_dim=output_dim,
```

```
            embeddings_initializer=keras.initializers.
 ↪Constant(embedding_matrix),
            trainable=False,
            mask_zero=True,
            )
        self.position_embeddings = layers.Embedding(
            input_dim=sequence_length, output_dim=output_dim)
        self.sequence_length = sequence_length
        self.input_dim = input_dim
        self.output_dim = output_dim

    def call(self, inputs):
        length = tf.shape(inputs)[-1]
        positions = tf.range(start=0, limit=length, delta=1)
        embedded_tokens = self.token_embeddings(inputs)
        embedded_positions = self.position_embeddings(positions)
        return embedded_tokens + embedded_positions

    def compute_mask(self, inputs, mask=None):
        return keras.ops.not_equal(inputs, 0)

    def get_config(self):
        config = super().get_config()
        config.update({
            "output_dim": self.output_dim,
            "sequence_length": self.sequence_length,
            "input_dim": self.input_dim,
        })
        return config
#Putting it all together: A text-classification Transformer
#Combining the Transformer encoder with positional embedding

vocab_size = 10000
sequence_length = 150
embed_dim = 100
num_heads = 2
dense_dim = 32

inputs = keras.Input(shape=(None,), dtype="int64")
x = PositionalEmbedding(sequence_length, vocab_size, embed_dim)(inputs)
x = TransformerEncoder(embed_dim, dense_dim, num_heads)(x)
x = layers.GlobalMaxPooling1D()(x)
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("full_transformer_encoder_pretrained{epoch:
 ↪02d}.keras",
                                    save_best_only=True)
]
model.fit(int_train_ds, validation_data=int_val_ds, epochs=20,␣
 ↪callbacks=callbacks)
```

Testing full transformer with pretrained embedding

[ ]:
```
model = keras.models.load_model(
    "full_transformer_encoder_pretrained15.keras",
    custom_objects={"TransformerEncoder": TransformerEncoder,
                    "PositionalEmbedding": PositionalEmbedding})
print(f"Test acc: {model.evaluate(int_test_ds)[1]:.3f}")
```