

```
#1
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import SGD, Adam, RMSprop
import numpy as np

# Generate dummy data
X = np.random.rand(1000, 10)
y = np.random.randint(0, 2, (1000, 1))

# Define different activation functions and optimizer classes
activations = ['relu', 'sigmoid', 'tanh']
optimizer_classes = [SGD, Adam, RMSprop]

# Training small models with different combinations
for act in activations:
    for opt_class in optimizer_classes:
        optimizer = opt_class() # Create a new optimizer instance here
        print(f"Training model with activation={act} and optimizer={optimizer.__class__.__name__}")
        model = Sequential([
            Dense(32, input_shape=(10,)), activation=act),
            Dense(1, activation='sigmoid') # Output layer for binary classification
        ])
        model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accuracy'])
        model.fit(X, y, epochs=5, batch_size=32, verbose=0)
        loss, acc = model.evaluate(X, y, verbose=0)
        print(f"Accuracy: {acc:.4f}\n")

↩ Training model with activation=relu and optimizer=SGD
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` arg
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Accuracy: 0.4850

Training model with activation=relu and optimizer=Adam
Accuracy: 0.5360

Training model with activation=relu and optimizer=RMSprop
Accuracy: 0.5470

Training model with activation=sigmoid and optimizer=SGD
Accuracy: 0.4980

Training model with activation=sigmoid and optimizer=Adam
Accuracy: 0.5200

Training model with activation=sigmoid and optimizer=RMSprop
Accuracy: 0.4860

Training model with activation=tanh and optimizer=SGD
Accuracy: 0.5320

Training model with activation=tanh and optimizer=Adam
Accuracy: 0.5090

Training model with activation=tanh and optimizer=RMSprop
Accuracy: 0.4870
```

```
#3
import cv2
import numpy as np
import matplotlib.pyplot as plt
def apply_filter(image, kernel):
    return cv2.filter2D(image, -1, kernel)
def main():
    image = cv2.imread('istockphoto-945104978-612x612.jpg', cv2.IMREAD_COLOR_RGB)

    # if image is None:
    #     print("Error: Could not load image")
    # return
    edge_detection = np.array([[[-1,-1,-1],[-1,8,-1],[-1,-1,-1]]])

    sharpening = np.array([[[-1,-1,1],[-1,5,-1],[0,-1,0]])
    box_blur = np.ones((3,3), np.float32)/9.0

    edge_detected_image = apply_filter(image, edge_detection)
    sharpened_image = apply_filter(image, sharpening)
    blurred_image = apply_filter(image, box_blur)

    titles = ['Original', 'Edge Detection', 'Sharpening', 'Blurring']
    images = [image, edge_detected_image, sharpened_image, blurred_image]
```

```

images = [image, edge_detected_image, sharpened_image, blurred_image]

plt.figure(figsize=(10,5))
for i in range(4):
    plt.subplot(1,4,i+1)
    plt.imshow(images[i], cmap='gray')
    plt.title(titles[i])
    plt.axis('off')
plt.show()

if __name__ == "__main__":
    main()

#4
import tensorflow as tf

# Define a simple CNN model
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(10, activation='softmax')
])

# Load and preprocess the MNIST dataset
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
x_train = x_train.reshape(-1, 28, 28, 1).astype('float32') / 255.0
x_test = x_test.reshape(-1, 28, 28, 1).astype('float32') / 255.0
y_train = tf.keras.utils.to_categorical(y_train, num_classes=10)
y_test = tf.keras.utils.to_categorical(y_test, num_classes=10)

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

# Train the model (for a small number of epochs for brevity)
model.fit(x_train, y_train, epochs=2, batch_size=32, verbose=0)

# Evaluate the model
loss, accuracy = model.evaluate(x_test, y_test, verbose=0)
print(f"Test Loss: {loss:.4f}")
print(f"Test Accuracy: {accuracy:.4f}")

Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 — 0s 0us/step
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/
    super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Test Loss: 0.0696
Test Accuracy: 0.9768

#5
import tensorflow as tf; import numpy as np

# Synthetic data: (samples, time steps, features)
X = np.random.rand(1, 100, 5)
y = np.sin(np.sum(X, axis=2))[:, :, np.newaxis]


# LSTM model
lstm_model = tf.keras.models.Sequential([
    tf.keras.layers.LSTM(32, return_sequences=True, input_shape=(100, 5)),
    tf.keras.layers.Dense(1)
])

# GRU model
gru_model = tf.keras.models.Sequential([
    tf.keras.layers.GRU(32, return_sequences=True, input_shape=(100, 5)),
    tf.keras.layers.Dense(1)
])

# Compile
lstm_model.compile(optimizer='adam', loss='mse')
gru_model.compile(optimizer='adam', loss='mse')

# Summaries
lstm_model.summary()
gru_model.summary()

```

 /usr/local/lib/python3.11/dist-packages/keras/src/layers/rnn/rnn.py:200: UserWarning: Do not pass an `input\_shape`/`input\_dim` argument to `super().__init__()` (`**kwargs`)  
 super().\_\_init\_\_(\*\*kwargs)  
 Model: "sequential\_10"

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 100, 32)	4,864
dense_19 (Dense)	(None, 100, 1)	33

Total params: 4,897 (19.13 KB)  
 Trainable params: 4,897 (19.13 KB)  
 Non-trainable params: 0 (0.00 B)  
 Model: "sequential\_11"

Layer (type)	Output Shape	Param #
gru (GRU)	(None, 100, 32)	3,744
dense_20 (Dense)	(None, 100, 1)	33

Total params: 3,777 (14.75 KB)  
 Trainable params: 3,777 (14.75 KB)  
 Non-trainable params: 0 (0.00 B)

```
#6
import tensorflow as tf


# Vocabulary and embedding size
vocab_size = 10000
embedding_dim = 100
sequence_length = 50

# LSTM model for word embeddings
lstm_model = tf.keras.models.Sequential([
    tf.keras.layers.Embedding(vocab_size, embedding_dim, input_length=sequence_length),
    tf.keras.layers.LSTM(128),
    tf.keras.layers.Dense(vocab_size, activation='softmax') # Example: predicting next word
])

# GRU model for word embeddings
gru_model = tf.keras.models.Sequential([
    tf.keras.layers.Embedding(vocab_size, embedding_dim, input_length=sequence_length),
    tf.keras.layers.GRU(128),
    tf.keras.layers.Dense(vocab_size, activation='softmax') # Example: predicting next word
])

# Compile models
lstm_model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
gru_model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Model summaries
lstm_model.summary()
gru_model.summary()
```

 /usr/local/lib/python3.11/dist-packages/keras/src/layers/core/embedding.py:90: UserWarning: Argument `input\_length` is deprecated. Use `input_shape[1]` instead.  
 warnings.warn(  
 Model: "sequential\_12"

Layer (type)	Output Shape	Param #
embedding (Embedding)	?	0 (unbuilt)
lstm_1 (LSTM)	?	0 (unbuilt)
dense_21 (Dense)	?	0 (unbuilt)

Total params: 0 (0.00 B)  
 Trainable params: 0 (0.00 B)  
 Non-trainable params: 0 (0.00 B)  
 Model: "sequential\_13"

Layer (type)	Output Shape	Param #
embedding_1 (Embedding)	?	0 (unbuilt)
gru_1 (GRU)	?	0 (unbuilt)
dense_22 (Dense)	?	0 (unbuilt)

Total params: 0 (0.00 B)

```
#8
import tensorflow as tf
from transformers import TFBertModel, BertTokenizer
```

```

from transformers import BertModel, BertTokenizer
import matplotlib.pyplot as plt
import numpy as np

# Load tokenizer and model
tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")
model = TFBertModel.from_pretrained("bert-base-uncased", output_attentions=True)

# Input text
sentence = "Hugging Face makes NLP easy"
inputs = tokenizer(sentence, return_tensors="tf")

# Get outputs (includes attention weights)
outputs = model(**inputs)
attentions = outputs.attentions # Tuple of (layer, batch, head, seq_len, seq_len)

# Visualize attention from layer 0, head 0
attn = attentions[0][0, 0].numpy() # shape: (seq_len, seq_len)

# Plot attention heatmap
tokens = tokenizer.tokenize(sentence)
tokens = ['[CLS]'] + tokens + ['[SEP]']
plt.figure(figsize=(6, 6))
plt.imshow(attn, cmap='viridis')
plt.xticks(range(len(tokens)), tokens, rotation=90)
plt.yticks(range(len(tokens)), tokens)
plt.colorbar()
plt.title("BERT Attention (Layer 0, Head 0)")
plt.tight_layout()
plt.show()

```

⚠ /usr/local/lib/python3.11/dist-packages/huggingface\_hub/utils/\_auth.py:94: UserWarning:  
The secret `HF\_TOKEN` does not exist in your Colab secrets.  
To authenticate with the Hugging Face Hub, create a token in your settings tab (<https://huggingface.co/settings/tokens>), set it as :  
You will be able to reuse this secret in all of your notebooks.  
Please note that authentication is recommended but still optional to access public models or datasets.

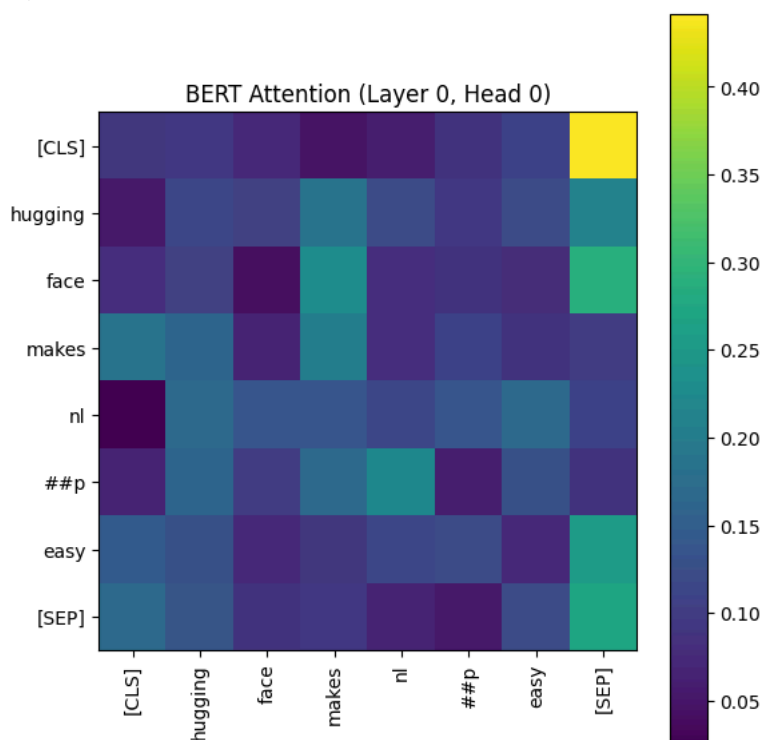
```

warnings.warn(
tokenizer_config.json: 100% 48.0/48.0 [00:00<00:00, 3.83kB/s]
vocab.txt: 100% 232k/232k [00:00<00:00, 5.32MB/s]
tokenizer.json: 100% 466k/466k [00:00<00:00, 24.1MB/s]
config.json: 100% 570/570 [00:00<00:00, 47.7kB/s]

```

Xet Storage is enabled for this repo, but the 'hf\_xet' package is not installed. Falling back to regular HTTP download. For better performance, install the 'hf\_xet' package.  
WARNING:huggingface\_hub.file\_download:Xet Storage is enabled for this repo, but the 'hf\_xet' package is not installed. Falling back to regular HTTP download.  
model.safetensors: 100% 440M/440M [00:08<00:00, 47.4MB/s]

Some weights of the PyTorch model were not used when initializing the TF 2.0 model TFBertModel: ['cls.predictions.transform.dense.weight', 'cls.predictions.transform.dense.bias', 'cls.predictions.decoder.weight']. This is expected if you are initializing TFBertModel from a PyTorch model trained on another task or with another architecture (e.g. seq2seq). This IS NOT expected if you are initializing TFBertModel from a PyTorch model that you expect to be exactly identical (e.g. initializing TFBertModel with the same weights). All the weights of TFBertModel were initialized from the PyTorch model.  
If your task is similar to the task the model of the checkpoint was trained on, you can already use TFBertModel for predictions with the same weights.



```
#7
import tensorflow as tf
import numpy as np
from tensorflow.keras.layers import Input, Embedding, LSTM, Dense
from tensorflow.keras.models import Model

# Tiny dataset
eng = ["hello", "thank you"]
fre = ["bonjour", "merci"] # Removed <start>/<end> for simplicity

# Create character-level tokenizer (works better for tiny datasets)
eng_chars = set(''.join(eng))
fre_chars = set(''.join(fre))

eng_to_idx = {c:i+1 for i,c in enumerate(eng_chars)}
fre_to_idx = {c:i+1 for i,c in enumerate(fre_chars)}

# Convert sentences to sequences
max_len = max(len(s) for s in eng+fre)
eng_seq = np.array([[eng_to_idx[c] for c in s] + [0]*(max_len-len(s)) for s in eng])
fre_seq = np.array([[fre_to_idx[c] for c in s] + [0]*(max_len-len(s)) for s in fre])

# Simple seq2seq model
encoder_inputs = Input(shape=(max_len,))
enc_emb = Embedding(len(eng_to_idx)+1, 8)(encoder_inputs)
_, state_h, state_c = LSTM(16, return_state=True)(enc_emb)

decoder_inputs = Input(shape=(max_len,))
dec_emb = Embedding(len(fre_to_idx)+1, 8)(decoder_inputs)
decoder_outputs = LSTM(16, return_sequences=True)(dec_emb, initial_state=[state_h, state_c])
outputs = Dense(len(fre_to_idx)+1, activation='softmax')(decoder_outputs)

model = Model([encoder_inputs, decoder_inputs], outputs)
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy')

# Train (using teacher forcing)
model.fit([eng_seq, fre_seq], np.expand_dims(fre_seq, -1), epochs=100, verbose=0)

# Translation function
def translate(word):
    inp = np.array([[eng_to_idx.get(c, 0) for c in word] + [0]*(max_len-len(word))])
    pred = model.predict([inp, np.zeros((1, max_len))], verbose=0)[0]
    return ''.join([list(fre_to_idx.keys())[np.argmax(p)-1] for p in pred if np.argmax(p) > 0])

print(f"hello -> '{translate('hello')}'")
print(f"thank you -> '{translate('thank you')}'")
```

→ 'hello' -> 'oo'  
'thank you' -> ''

```
#2
# prompt: different weight initialization techniques in deep learning
# give full code in 15 lines there should be a graph in output

import matplotlib.pyplot as plt
import numpy as np

# Sample weight initialization techniques (replace with your actual techniques)
initializers = ['zeros', 'ones', 'random_normal', 'glorot_uniform']
losses = []

for initializer in initializers:
    # Simulate model training and record loss
    # Replace this with your actual model training logic
    loss = np.random.rand()
    losses.append(loss)

# Create the plot
plt.figure(figsize=(8, 6))
plt.bar(initializers, losses, color=['skyblue', 'salmon', 'lightgreen', 'gold'])
plt.xlabel("Weight Initialization Techniques")
plt.ylabel("Loss")
plt.title("Impact of Weight Initialization on Model Performance")
plt.show()
```

