Deep Learning Lab – Code Cheat Sheet

1. Deep Neural Network starting with Linear Regression

1a) Single Variable

```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow import keras
from tensorflow.keras import layers

X = np.arange(0, 100, dtype=float)
y = 3 * X + 7 + np.random.randn(100) * 5

model = keras.Sequential([layers.Dense(1, input_shape=(1,))])
model.compile(optimizer=keras.optimizers.Adam(0.1), loss='mse')
model.fit(X, y, epochs=200, verbose=0)

print("Weights:", model.layers[0].get_weights())
print("Predict x=50:", model.predict(np.array([50.0])))

plt.scatter(X, y)
plt.plot(X, model.predict(X), color='r')
plt.show()
```

1b) Multiple Variables

```
import numpy as np
from tensorflow import keras
from tensorflow.keras import layers
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

X = np.random.rand(500, 3) * 10
y = 2*X[:, 0] + 3*X[:, 1] - 1.5*X[:, 2] + 5 + np.random.randn(500)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

```
model = keras.Sequential([
    layers.Dense(64, activation='relu', input_shape=(3,)),
    layers.Dense(32, activation='relu'),
    layers.Dense(1)
])
model.compile(optimizer='adam', loss='mse', metrics=['mae'])
model.fit(X_train, y_train, epochs=100, validation_split=0.2, verbose=0)

print("Test [loss, mae]:", model.evaluate(X_test, y_test, verbose=0))
print("Sample predictions:", model.predict(X_test[:5]).flatten())
```

2. Speech, Text and Video Conversion

2a) Speech into Text

```
# pip install SpeechRecognition pyaudio
import speech_recognition as sr

r = sr.Recognizer()
with sr.Microphone() as source:
    print("Speak now...")
    r.adjust_for_ambient_noise(source)
    audio = r.listen(source)

try:
    print("You said:", r.recognize_google(audio))
except sr.UnknownValueError:
    print("Could not understand audio")
except sr.RequestError as e:
    print("API error:", e)

# From an audio file instead of the microphone:
# with sr.AudioFile('audio.wav') as source:
#     audio = r.record(source)
```

2b) Text into Speech

```
# pip install gTTS
from gtts import gTTS
import os

text = "Hello, welcome to the deep learning lab"
gTTS(text=text, lang='en', slow=False).save("output.mp3")
os.system("start output.mp3")   # Windows; use 'afplay' (Mac) or 'mpg321'
(Linux)

# Offline alternative:
# import pyttsx3
# e = pyttsx3.init(); e.say(text); e.runAndWait()
```

2c) Video into Frames

```
# pip install opencv-python
import cv2, os

os.makedirs("frames", exist_ok=True)
cap = cv2.VideoCapture("video.mp4")
count = 0
while True:
    ret, frame = cap.read()
    if not ret:
        break
    cv2.imwrite(f"frames/frame_{count:04d}.jpg", frame)
    count += 1
cap.release()
print("Total frames saved:", count)
```

3. Feed Forward Neural Network for Logic Gates

```
import numpy as np
from tensorflow import keras
from tensorflow.keras import layers

X = np.array([[0,0],[0,1],[1,0],[1,1]], dtype=float)
gates = {
    "AND":  np.array([0,0,0,1]),
```

```
    "OR":   np.array([0,1,1,1]),
    "NAND": np.array([1,1,1,0]),
    "NOR":  np.array([1,0,0,0]),
    "XOR":  np.array([0,1,1,0]),
    "XNOR": np.array([1,0,0,1]),
}

for name, y in gates.items():
    model = keras.Sequential([
        layers.Dense(8, activation='relu', input_shape=(2,)),
        layers.Dense(1, activation='sigmoid')
    ])
    model.compile(optimizer=keras.optimizers.Adam(0.05),
loss='binary_crossentropy', metrics=['accuracy'])
    model.fit(X, y, epochs=500, verbose=0)
    pred = (model.predict(X, verbose=0) > 0.5).astype(int).flatten()
    print(f"{name}: predicted={pred}, expected={y}")
```

4. Character Recognition

4a) CNN

```
from tensorflow import keras
from tensorflow.keras import layers

(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()
x_train = x_train.reshape(-1, 28, 28, 1) / 255.0
x_test = x_test.reshape(-1, 28, 28, 1) / 255.0

model = keras.Sequential([
    layers.Conv2D(32, (3,3), activation='relu', input_shape=(28,28,1)),
    layers.MaxPooling2D(2),
    layers.Conv2D(64, (3,3), activation='relu'),
    layers.MaxPooling2D(2),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dropout(0.3),
    layers.Dense(10, activation='softmax')
])
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
```

```
metrics=['accuracy'])
model.fit(x_train, y_train, epochs=5, batch_size=128, validation_split=0.1)

print("Test [loss, acc]:", model.evaluate(x_test, y_test))
```

4b) RNN

```
from tensorflow import keras
from tensorflow.keras import layers

(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0   # each image = 28
timesteps x 28 features

model = keras.Sequential([
    layers.SimpleRNN(128, input_shape=(28, 28)),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')
])
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
model.fit(x_train, y_train, epochs=5, batch_size=128, validation_split=0.1)

print("Test [loss, acc]:", model.evaluate(x_test, y_test))
```

5. Image Caption Prediction

5a) LSTM (CNN features + LSTM decoder)

```
import numpy as np
from tensorflow.keras.applications import InceptionV3
from tensorflow.keras.applications.inception_v3 import preprocess_input
from tensorflow.keras.preprocessing import image
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense, LSTM, Embedding, Dropout,
add

# ---- Feature extractor ----
```

```
cnn = InceptionV3(weights='imagenet')
extractor = Model(cnn.input, cnn.layers[-2].output)   # 2048-d

def get_features(path):
    img = image.load_img(path, target_size=(299, 299))
    x = preprocess_input(np.expand_dims(image.img_to_array(img), 0))
    return extractor.predict(x, verbose=0)[0]

# ---- Sample data (replace with Flickr8k) ----
data = {"img1.jpg": "startseq a dog runs on grass endseq",
        "img2.jpg": "startseq a man rides a horse endseq"}
features = {k: get_features(k) for k in data}

tok = Tokenizer()
tok.fit_on_texts(data.values())
vocab = len(tok.word_index) + 1
max_len = max(len(c.split()) for c in data.values())

X1, X2, y = [], [], []
for k, cap in data.items():
    seq = tok.texts_to_sequences([cap])[0]
    for i in range(1, len(seq)):
        X1.append(features[k])
        X2.append(pad_sequences([seq[:i]], maxlen=max_len)[0])
        y.append(to_categorical(seq[i], vocab))
X1, X2, y = np.array(X1), np.array(X2), np.array(y)

# ---- Model ----
i1 = Input(shape=(2048,)); f = Dense(256, activation='relu')(Dropout(0.5)
(i1))
i2 = Input(shape=(max_len,)); s = LSTM(256)(Dropout(0.5)(Embedding(vocab,
256, mask_zero=True)(i2)))
out = Dense(vocab, activation='softmax')(Dense(256, activation='relu')
(add([f, s])))
model = Model([i1, i2], out)
model.compile(loss='categorical_crossentropy', optimizer='adam')
model.fit([X1, X2], y, epochs=30, verbose=0)

# ---- Generate caption ----
def caption(path):
    feat = get_features(path).reshape(1, -1)
    text = 'startseq'
    for _ in range(max_len):
```

```
        seq = pad_sequences(tok.texts_to_sequences([text]), maxlen=max_len)
        word = tok.index_word.get(np.argmax(model.predict([feat, seq],
verbose=0)))
        if word is None: break
        text += ' ' + word
        if word == 'endseq': break
    return text.replace('startseq', '').replace('endseq', '').strip()

print(caption("img1.jpg"))
```

5b) CNN (CNN features + dense decoder, no LSTM)

```
import numpy as np
from tensorflow.keras.applications import InceptionV3
from tensorflow.keras.applications.inception_v3 import preprocess_input
from tensorflow.keras.preprocessing import image
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense, Embedding, Flatten,
Dropout, concatenate

cnn = InceptionV3(weights='imagenet')
extractor = Model(cnn.input, cnn.layers[-2].output)

def get_features(path):
    img = image.load_img(path, target_size=(299, 299))
    x = preprocess_input(np.expand_dims(image.img_to_array(img), 0))
    return extractor.predict(x, verbose=0)[0]

data = {"img1.jpg": "startseq a dog runs on grass endseq",
        "img2.jpg": "startseq a man rides a horse endseq"}
features = {k: get_features(k) for k in data}

tok = Tokenizer(); tok.fit_on_texts(data.values())
vocab = len(tok.word_index) + 1
max_len = max(len(c.split()) for c in data.values())

X1, X2, y = [], [], []
for k, cap in data.items():
    seq = tok.texts_to_sequences([cap])[0]
```

```
    for i in range(1, len(seq)):
        X1.append(features[k])
        X2.append(pad_sequences([seq[:i]], maxlen=max_len)[0])
        y.append(to_categorical(seq[i], vocab))
X1, X2, y = np.array(X1), np.array(X2), np.array(y)

i1 = Input(shape=(2048,)); f = Dense(256, activation='relu')(Dropout(0.5)
(i1))
i2 = Input(shape=(max_len,)); w = Flatten()(Embedding(vocab, 64)(i2))
w = Dense(256, activation='relu')(w)
out = Dense(vocab, activation='softmax')(Dense(256, activation='relu')
(concatenate([f, w])))
model = Model([i1, i2], out)
model.compile(loss='categorical_crossentropy', optimizer='adam')
model.fit([X1, X2], y, epochs=30, verbose=0)

text = 'startseq'
feat = get_features("img1.jpg").reshape(1, -1)
for _ in range(max_len):
    seq = pad_sequences(tok.texts_to_sequences([text]), maxlen=max_len)
    word = tok.index_word.get(np.argmax(model.predict([feat, seq],
verbose=0)))
    if word is None or word == 'endseq': break
    text += ' ' + word
print(text.replace('startseq', '').strip())
```

6. Autoencoder and GAN on MNIST

6a) Autoencoder

```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow import keras
from tensorflow.keras import layers

(x_train, _), (x_test, _) = keras.datasets.mnist.load_data()
x_train = x_train.reshape(-1, 784) / 255.0
x_test = x_test.reshape(-1, 784) / 255.0

inp = keras.Input(shape=(784,))
enc = layers.Dense(128, activation='relu')(inp)
```

```
enc = layers.Dense(32, activation='relu')(enc)
dec = layers.Dense(128, activation='relu')(enc)
dec = layers.Dense(784, activation='sigmoid')(dec)

autoencoder = keras.Model(inp, dec)
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
autoencoder.fit(x_train, x_train, epochs=20, batch_size=256, validation_data=
(x_test, x_test))

recon = autoencoder.predict(x_test[:10])
plt.figure(figsize=(20, 4))
for i in range(10):
    plt.subplot(2, 10, i+1);      plt.imshow(x_test[i].reshape(28,28),
cmap='gray'); plt.axis('off')
    plt.subplot(2, 10, i+11);     plt.imshow(recon[i].reshape(28,28),
cmap='gray'); plt.axis('off')
plt.show()
```

6b) GAN

```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow import keras
from tensorflow.keras import layers

(x_train, _), _ = keras.datasets.mnist.load_data()
x_train = (x_train.reshape(-1, 784).astype('float32') - 127.5) / 127.5   #
scale to [-1, 1]
latent_dim, batch = 100, 128

generator = keras.Sequential([
    layers.Dense(256, input_dim=latent_dim), layers.LeakyReLU(0.2),
    layers.Dense(512), layers.LeakyReLU(0.2),
    layers.Dense(1024), layers.LeakyReLU(0.2),
    layers.Dense(784, activation='tanh')
])
discriminator = keras.Sequential([
    layers.Dense(1024, input_dim=784), layers.LeakyReLU(0.2),
layers.Dropout(0.3),
    layers.Dense(512), layers.LeakyReLU(0.2), layers.Dropout(0.3),
    layers.Dense(256), layers.LeakyReLU(0.2),
    layers.Dense(1, activation='sigmoid')
```

```
])
discriminator.compile(optimizer=keras.optimizers.Adam(0.0002, 0.5),
loss='binary_crossentropy')

discriminator.trainable = False
gan_in = keras.Input(shape=(latent_dim,))
gan = keras.Model(gan_in, discriminator(generator(gan_in)))
gan.compile(optimizer=keras.optimizers.Adam(0.0002, 0.5),
loss='binary_crossentropy')

def show(epoch):
    imgs = generator.predict(np.random.normal(0, 1, (16, latent_dim)),
verbose=0) * 0.5 + 0.5
    plt.figure(figsize=(4, 4))
    for i in range(16):
        plt.subplot(4, 4, i+1); plt.imshow(imgs[i].reshape(28,28),
cmap='gray'); plt.axis('off')
    plt.suptitle(f"Epoch {epoch}"); plt.show()

for epoch in range(1, 5001):
    real = x_train[np.random.randint(0, len(x_train), batch)]
    fake = generator.predict(np.random.normal(0, 1, (batch, latent_dim)),
verbose=0)
    d_loss = discriminator.train_on_batch(np.vstack([real, fake]),
                                          np.vstack([np.ones((batch,1)),
np.zeros((batch,1))]))
    g_loss = gan.train_on_batch(np.random.normal(0, 1, (batch, latent_dim)),
np.ones((batch,1)))
    if epoch % 1000 == 0:
        print(f"{epoch} D:{d_loss:.3f} G:{g_loss:.3f}")
        show(epoch)
```