```
#Deep Learning:
import tensorflow as tf
from tensorflow.keras.datasets import imdb
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense
# Load top 10,000 words from IMDB dataset
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=10000)
# Pad sequences to the same length (200 words)
x_train = pad_sequences(x_train, maxlen=200)
x_test = pad_sequences(x_test, maxlen=200)
model = Sequential([
    Embedding(input_dim=10000, output_dim=64, input_length=200),
    LSTM(64),
    Dense(1, activation='sigmoid')  # Output: 0 (neg) or 1 (pos)
])
model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])
model.fit(x_train, y_train, epochs=3, batch_size=64, validation_split=0.2)
loss, acc = model.evaluate(x_test, y_test)
print(f"Test Accuracy: {acc:.2f}")
```

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

# Load and preprocess MNIST
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()
x_train, x_test = x_train[..., np.newaxis] / 255.0, x_test[..., np.newaxis] / 255.0
y_train = keras.utils.to_categorical(y_train, 10)
y_test = keras.utils.to_categorical(y_test, 10)

# Build model
model = keras.Sequential([
    layers.Conv2D(32, 3, activation='relu', input_shape=(28, 28, 1)),
    layers.MaxPooling2D(2),
    layers.Conv2D(64, 3, activation='relu'),
    layers.MaxPooling2D(2),
    layers.Flatten(),
    layers.Dropout(0.5),
    layers.Dense(10, activation='softmax')
])
model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])

# Train and evaluate
model.fit(x_train, y_train, batch_size=128, epochs=10, validation_split=0.1)
loss, acc = model.evaluate(x_test, y_test)
print(f"Test loss: {loss:.4f}, Test accuracy: {acc:.4f}")
```

```
##Q Learning:
import numpy as np

# Parameters
alpha, gamma, epsilon, episodes = 0.1, 0.9, 0.1, 1000
size, actions = 5, 4
q = np.zeros((size * size, actions))

# Helpers
def idx(r, c): return r * size + c
def act(state):
    return np.random.randint(actions) if np.random.rand() < epsilon else
np.argmax(q[idx(*state)])
def step(state, a):
    r, c = state
    if a == 0 and r > 0: r -= 1
    if a == 1 and r < 4: r += 1
    if a == 2 and c > 0: c -= 1
    if a == 3 and c < 4: c += 1
    next = (r, c)
    return next, (1 if next == (4, 4) else -0.1), next == (4, 4)

# Training
for ep in range(episodes):
    s = (0, 0)
    while True:
        a = act(s)
        ns, r, d = step(s, a)
        q[idx(*s), a] += alpha * (r + gamma * np.max(q[idx(*ns)]) - q[idx(*s), a])
        s = ns
        if d: break
    if (ep + 1) % 100 == 0: print(f"Episode {ep+1} complete")

# Test
def test():
    s, path = (0, 0), [(0, 0)]
    while s != (4, 4):
        s, *_ = step(s, np.argmax(q[idx(*s)]))
        path.append(s)
    return path

print("Learned path to goal:", test())
```