LSTM:

```
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

(x_train,y_train),(x_test,y_test)=imdb.load_data(num_words=10000)
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')])
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}") for the above code
```

MODIFICATIONS:
Add Dropout layer to reduce overfitting #Dropout(0.5),
Use bidirectional LSTM #Bidirectional(LSTM(64)),
Loss function -> mean_squared #loss='mean_squared_error',
Use different activation function and optimizer
#activation='relu' #optimizer='adagrad'
#activation='tanh' #optimizer='sgd'

Use different datasets

```
import tensorflow as tf
from tensorflow.keras.datasets import reuters
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense
import numpy as np

# Load and filter Reuters dataset for binary classification (class 0 vs
others)
(x_train, y_train), (x_test, y_test) =
reuters.load_data(num_words=10000)
y_train = np.where(y_train == 0, 1, 0)  # class 0 as positive, others as
negative
y_test = np.where(y_test == 0, 1, 0)
```

CNN:

```
import numpy as np from tensorflow import keras from
tensorflow.keras import layers num_classes=10
input_shape=(28,28,1)
(x_train,y_train),(x_test,y_test)=keras.datasets.mnist.load_data()
x_train=x_train.astype("float32")/255
x_test=x_test.astype("float32")/255
x_train=np.expand_dims(x_train,-1) x_test=np.expand_dims(x_test,-1)
y_train=keras.utils.to_categorical(y_train,num_classes)
y_test=keras.utils.to_categorical(y_test,num_classes)
model=keras.Sequential([
layers.Conv2D(32,kernel_size=3,activation="relu",input_shape=input_
shape), layers.MaxPooling2D(pool_size=2),
layers.Conv2D(32,kernel_size=3,activation="relu"),
layers.MaxPooling2D(pool_size=2), layers.Flatten(),
layers.Dropout(0.5), layers.Dense(num_classes,activation="softmax")
])
model.compile(loss="categorical_crossentropy",optimizer="adam",met
rics=["accuracy"]) model.summary()
model.fit(x_train,y_train,epochs=10,batch_size=128,validation_split=0.
1) loss,acc=model.evaluate(x_test,y_test) print(f"Test
loss:{loss:4f},Test acc:{acc:4f}")
```

MODIFICATION:

Add batch Normalization:

```
model = keras.Sequential([
    layers.Conv2D(32, kernel_size=3, activation='relu',
input_shape=input_shape),
    layers.BatchNormalization(),
    layers.MaxPooling2D(pool_size=2),

    layers.Conv2D(64, kernel_size=3, activation='relu'),
    layers.BatchNormalization(),
    layers.MaxPooling2D(pool_size=2),

    layers.Flatten(),
    layers.Dropout(0.5),

    layers.Dense(128),  # Optional hidden dense layer before output
    layers.BatchNormalization(),
    layers.Activation("relu"),

    layers.Dense(num_classes, activation="softmax")
])
```

Add third Convolution layer

```
model = keras.Sequential([
    layers.Conv2D(32, kernel_size=3, activation="relu",
input_shape=input_shape),
    layers.MaxPooling2D(pool_size=2),

    layers.Conv2D(64, kernel_size=3, activation="relu"),
    layers.MaxPooling2D(pool_size=2),

    layers.Conv2D(128, kernel_size=3, activation="relu"),  # Third Conv
layer
    layers.Flatten(),

    layers.Dropout(0.5),
    layers.Dense(num_classes, activation="softmax")
])
```

Add Early Stopping Callback:

```
y_test = keras.utils.to_categorical(y_test, num_classes)

# Define EarlyStopping callback
early_stop = EarlyStopping(monitor='val_loss', patience=3,
restore_best_weights=True)

# Build the model
model = keras.Sequential([
```

Add Image Shape Expansion:

```
x_train = x_train.astype("float32") / 255
```

```
x_test = x_test.astype("float32") / 255

# ✅ Expand grayscale images to include channel dimension (28, 28) → (28, 28, 1)
x_train = np.expand_dims(x_train, axis=-1)
x_test = np.expand_dims(x_test, axis=-1)
```

Use different dataset - cifar-10:

```
(x_train, y_train), (x_test, y_test) = keras.datasets.cifar10.load_data()
```

Q-learning:

```
import numpy as np
alpha=0.1
gamma =0.9
epsilon=0.1
episodes=1000
grid_size=5
num_actions=4
q_table=np.zeros((grid_size*grid_size,num_actions))

def state_to_index(state):
    return state[0]*grid_size+state[1]
def choose_action(state):
    if np.random.rand()<epsilon:
        return np.random.randint(num_actions)
    return np.argmax(q_table[state_to_index(state)])
def take_action(state,action):
    row,col=state
    if action==0 and row>0:row-=1
    elif action==1 and row<4:row+=1
    elif action==2 and col>0:col-=1
    elif action==3 and col<4:col+=1
    next_state=(row,col)
    reward=1 if next_state==(4,4) else -0.1
    done=next_state==(4,4)
    return next_state,reward,done

for ep in range(episodes):
    state=(0,0)
    while True:
        action=choose_action(state)
        next_state,reward,done=take_action(state,action)
        i,ni=state_to_index(state),state_to_index(next_state)

q_table[i,action]+=alpha*(reward+gamma*np.max(q_table[ni])-q_table[
i,action])
        state=next_state
        if done:
            break
    if (ep+1)%100==0:
        print(f"Episodes {ep+1} completed")

def test_policy():
    state=(0,0)
    path=[state]
    while state !=(4,4):
        action=np.argmax(q_table[state_to_index(state)])
        state, _, _ = take_action(state, action)
        path.append(state)
    return path
```

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

MODIFICATION:

To find possible paths from source to goal vice versa :