# Assignment of Neural Network a simple perception

## Theory Questions

### 1. What is Deep Learning, and how is it connected to Artificial Intelligence?

Deep Learning is a subset of Machine Learning, which itself is a subfield of Artificial Intelligence (AI). It involves neural networks with many layers (hence "deep") that learn hierarchical representations of data.

- **AI** → Broad field aiming to create intelligent machines.
- **Machine Learning** → Algorithms that learn from data.
- **Deep Learning** → Neural networks that learn complex patterns using multiple layers.

### 2. What is a Neural Network, and what are the different types?

A Neural Network is a computational model inspired by the human brain. It consists of interconnected layers of neurons (nodes) that process information.

**Types of Neural Networks:**

- **Feedforward Neural Network (FNN)** – basic structure; no cycles.
- **Convolutional Neural Network (CNN)** – ideal for image data.
- **Recurrent Neural Network (RNN)** – used for sequence data (text, time-series).
- **Generative Adversarial Network (GAN)** – two networks competing (generator vs discriminator).
- **Transformer Networks** – designed for NLP tasks, powering models like GPT.

### 3. What is the Mathematical Structure of a Neural Network?

A single-layer neuron performs:

$$y=f(\sum = wi * xi + b )$$

Where:

- Xi : input
- Wi : weight
- b : bias
- f : activation function
- y : output

For multilayer networks, this is repeated layer-wise, forming a composition of functions.

**4. What is an Activation Function, and why is it Essential?**

An activation function introduces non-linearity into the network, allowing it to model complex relationships.

Without activation functions, a neural network would behave like a linear regression model regardless of how many layers it has.

**5. Common Activation Functions**

- **Sigmoid**: f(x) =  1 / 1+e^-x
- **Tanh**: f(x)=tanh(x)
- **ReLU (Rectified Linear Unit)**: f(x)=max(0,x)
- **Leaky ReLU**: Allows small gradients when input is negative.
- **Softmax**: Converts outputs into probability distribution (used in classification).

**6. What is a Multilayer Neural Network?**

Also called a Multilayer Perceptron (MLP), it consists of:

- An input layer

- One or more hidden layers
- An output layer

Each layer has multiple neurons and is fully connected to the next.

**7. What is a Loss Function, and why is it Crucial?**

The loss function measures how well the model's predictions match the actual labels.

- It provides a quantitative value to optimize during training.
- Lower loss → better performance.

**8. Common Types of Loss Functions**

- **Mean Squared Error (MSE)** – regression tasks
- **Cross-Entropy Loss** – classification tasks
- **Hinge Loss** – used in SVMs
- **Huber Loss** – combines MSE and MAE

**9. How Does a Neural Network Learn?**

1. **Forward propagation**: Pass input through the network to get prediction.
2. **Loss calculation**: Compute the loss.
3. **Backward propagation**: Compute gradients using backpropagation.
4. **Weight update**: Adjust weights using an optimizer.

**🧰 10. What is an Optimizer and Why is it Necessary?**

An optimizer updates the network's weights to minimize the loss function using gradients.

Without an optimizer, the model would not learn from the loss.

**11. Common Optimizers**

- SGD (Stochastic Gradient Descent)
- Momentum
- Adam (Adaptive Moment Estimation)
- RMSprop
- Adagrad

Adam is widely used due to its adaptive learning rate and efficiency.

**12. Forward and Backward Propagation**

- **Forward Propagation**: Input → hidden layers → output → compute prediction.
- **Backward Propagation**: Compute loss → calculate gradients using chain rule → update weights.

**13. What is Weight Initialization, and How Does it Impact Training?**

Weight initialization sets the initial values of weights before training.

Good initialization:

- Prevents vanishing/exploding gradients
- Speeds up convergence

Common methods:

- Xavier Initialization
- He Initialization

**14. What is the Vanishing Gradient Problem?**

In deep networks, gradients can become very small, especially with sigmoid/tanh. This causes earlier layers to learn slowly or not at all.

---

**15. What is the Exploding Gradient Problem?**

When gradients become too large, causing unstable updates and numerical overflow. Common in RNNs and very deep networks.

Solutions:

- Gradient clipping
- Better weight initialization
- Using ReLU instead of sigmoid/tanh

# 1 Practical Questions

## 1.1 1) How do you create a simple perceptron for basic binary classification ?

[3]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import warnings
warnings.filterwarnings('ignore')

model = Sequential([
    Dense(1, input_shape=(2,), activation='sigmoid')  # 1 neuron, 2 features
])

model.compile(optimizer='sgd', loss='binary_crossentropy', metrics=['accuracy'])
```

## 1.2 2) How can you build a neural network with one hidden layer using Keras ?

[5]:
```
model = Sequential([
    Dense(8, input_shape=(4,), activation='relu'),  # hidden layer with 8␣
 ↪neurons
    Dense(1, activation='sigmoid')  # output layer for binary classification
])

model.compile(optimizer='adam', loss='binary_crossentropy',␣
 ↪metrics=['accuracy'])
```

## 1.3 3) How do you initialize weights using the Xavier (Glorot) initialization method in Keras ?

[7]:
```
from tensorflow.keras.initializers import GlorotUniform

model = Sequential([
    Dense(8, input_shape=(4,), activation='relu',␣
 ↪kernel_initializer=GlorotUniform()),
    Dense(1, activation='sigmoid')
])
```

## 1.4 4) How can you apply different activation functions in a neural network in Keras?

[9]:
```
model = Sequential([
    Dense(8, input_shape=(4,), activation='tanh'),       # Tanh activation
    Dense(4, activation='relu'),                          # ReLU activation
    Dense(1, activation='sigmoid')                        # Sigmoid for binary␣
 ↪output
])
```

## 1.5 5) How do you add dropout to a neural network model to prevent overfitting?

[11]:
```
from tensorflow.keras.layers import Dropout

model = Sequential([
    Dense(128, input_shape=(100,), activation='relu'),
    Dropout(0.5),  # Drop 50% neurons during training
    Dense(1, activation='sigmoid')
])
```

## 1.6 6) How do you manually implement forward propagation in a simple neural network?

[13]:
```
import numpy as np

# Input features and weights
X = np.array([[0.5, 0.2]])
weights = np.array([[0.1], [0.4]])
bias = 0.3

# Activation (Sigmoid)
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

z = np.dot(X, weights) + bias
output = sigmoid(z)

print("Output:", output)
```

```
Output: [[0.60587367]]
```

## 1.7 7) How do you add batch normalization to a neural network model in Keras?

[15]:
```
from tensorflow.keras.layers import BatchNormalization

model = Sequential([
    Dense(64, input_shape=(10,), activation='relu'),
```

```
    BatchNormalization(),
    Dense(1, activation='sigmoid')
])
```

### 1.8 8) How can you visualize the training process with accuracy and loss curves?

[17]:
```
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers import Dense
import matplotlib.pyplot as plt

# 1. Generate data
X, y = make_classification(n_samples=1000, n_features=2, n_informative=2,
                           n_redundant=0, random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# 2. Build model
model = Sequential([
    Dense(10, input_dim=2, activation='relu'),
    Dense(1, activation='sigmoid')  # Binary classification
])

# 3. Compile the model (must be done before training)
model.compile(optimizer='adam', loss='binary_crossentropy',␣
 ↪metrics=['accuracy'])

# 4. Train the model
history = model.fit(X_train, y_train, epochs=20, validation_data=(X_val, y_val))

# 5. Plot loss and accuracy
plt.figure(figsize = (8,3))
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Val Loss')
plt.plot(history.history['accuracy'], label='Train Acc')
plt.plot(history.history['val_accuracy'], label='Val Acc')
plt.xlabel('Epochs')
plt.ylabel('Value')
plt.title('Training History')
plt.legend()
plt.show()
```

```
Epoch 1/20
25/25              2s 25ms/step -
accuracy: 0.8355 - loss: 0.4886 - val_accuracy: 0.8600 - val_loss: 0.4464
Epoch 2/20
```

```
25/25              0s 8ms/step -
accuracy: 0.8681 - loss: 0.4377 - val_accuracy: 0.8600 - val_loss: 0.4238
Epoch 3/20
25/25              0s 8ms/step -
accuracy: 0.8441 - loss: 0.4335 - val_accuracy: 0.8600 - val_loss: 0.4053
Epoch 4/20
25/25              0s 8ms/step -
accuracy: 0.8565 - loss: 0.4273 - val_accuracy: 0.8650 - val_loss: 0.3897
Epoch 5/20
25/25              0s 8ms/step -
accuracy: 0.8649 - loss: 0.4058 - val_accuracy: 0.8650 - val_loss: 0.3762
Epoch 6/20
25/25              0s 11ms/step -
accuracy: 0.8567 - loss: 0.4052 - val_accuracy: 0.8700 - val_loss: 0.3642
Epoch 7/20
25/25              0s 9ms/step -
accuracy: 0.8646 - loss: 0.3757 - val_accuracy: 0.8750 - val_loss: 0.3531
Epoch 8/20
25/25              0s 10ms/step -
accuracy: 0.8422 - loss: 0.4032 - val_accuracy: 0.8750 - val_loss: 0.3439
Epoch 9/20
25/25              0s 10ms/step -
accuracy: 0.8666 - loss: 0.3603 - val_accuracy: 0.8750 - val_loss: 0.3356
Epoch 10/20
25/25              0s 11ms/step -
accuracy: 0.8454 - loss: 0.3798 - val_accuracy: 0.8750 - val_loss: 0.3286
Epoch 11/20
25/25              0s 9ms/step -
accuracy: 0.8637 - loss: 0.3501 - val_accuracy: 0.8750 - val_loss: 0.3218
Epoch 12/20
25/25              0s 10ms/step -
accuracy: 0.8482 - loss: 0.3674 - val_accuracy: 0.8750 - val_loss: 0.3161
Epoch 13/20
25/25              0s 10ms/step -
accuracy: 0.8570 - loss: 0.3661 - val_accuracy: 0.8750 - val_loss: 0.3111
Epoch 14/20
25/25              0s 11ms/step -
accuracy: 0.8526 - loss: 0.3561 - val_accuracy: 0.8750 - val_loss: 0.3060
Epoch 15/20
25/25              0s 10ms/step -
accuracy: 0.8484 - loss: 0.3438 - val_accuracy: 0.8750 - val_loss: 0.3016
Epoch 16/20
25/25              0s 11ms/step -
accuracy: 0.8464 - loss: 0.3475 - val_accuracy: 0.8750 - val_loss: 0.2980
Epoch 17/20
25/25              0s 11ms/step -
accuracy: 0.8586 - loss: 0.3374 - val_accuracy: 0.8750 - val_loss: 0.2946
Epoch 18/20
```

```
25/25              0s 11ms/step -
accuracy: 0.8719 - loss: 0.3149 - val_accuracy: 0.8750 - val_loss: 0.2910
Epoch 19/20
25/25              0s 11ms/step -
accuracy: 0.8443 - loss: 0.3482 - val_accuracy: 0.8750 - val_loss: 0.2889
Epoch 20/20
25/25              0s 12ms/step -
accuracy: 0.8548 - loss: 0.3454 - val_accuracy: 0.8750 - val_loss: 0.2859
```

## 1.9 9) How can you use gradient clipping in Keras to control the gradient size and prevent exploding gradients?

[19]:
```
from tensorflow.keras.optimizers import Adam

optimizer = Adam(learning_rate=0.001, clipvalue=1.0)  # or clipnorm=1.0
model.compile(optimizer=optimizer, loss='binary_crossentropy',␣
 ↪metrics=['accuracy'])
```

## 1.10 10) How can you create a custom loss function in Keras?

[21]:
```
import tensorflow.keras.backend as K

def custom_mse(y_true, y_pred):
    return K.mean(K.square(y_pred - y_true))

model.compile(optimizer='adam', loss=custom_mse, metrics=['mae'])
```

## 1.11 11) How can you visualize the structure of a neural network model in Keras?

[23]:
```
from tensorflow.keras.utils import plot_model

plot_model(model, to_file='model.png', show_shapes=True, show_layer_names=True)
```

[23]:

[ ]:
```

```