1. **Why is it generally preferable to use a Logistic Regression classifier rather than a classical Perceptron (i.e., a single layer of linear threshold units trained using the Perceptron training algorithm)? How can you tweak a Perceptron to make it equivalent to a Logistic Regression classifier?**

**A classical Perceptron will converge only if the dataset is linearly separable, and it won't be able to estimate class probabilities. In contrast, a Logistic Regression classifier will converge to a good solution even if the dataset is not linearly separable, and it will output class probabilities.  
If you change the Perceptron's activation function to the logistic activation function (or the softmax activation function if there are multiple neurons), and if you train it using Gradient Descent, then it becomes equivalent to a Logistic Regression classifier.**

1. **Why was the logistic activation function a key ingredient in training the first MLPs?**

**Because the derivative of the logistic function is always nonzero, so Gradient Descent can always roll down the slope. When the activation function is a step function, Gradient Descent cannot move, as there is no slope at all.  
The backpropagation algorithm may be used with other activation functions, instead of the logistic  
function.**

1. **Name three popular activation functions. Can you draw them?**

* **Logistic/logit/sigmoid**
* **step/threshold**
* **ReLU**
* **hyperbolic tangent tanh**
* **piecewise linear**
* **ELU**

1. **Suppose you have an MLP composed of one input layer with 10 passthrough neurons, followed by one hidden layer with 50 artificial neurons, and finally one output layer with 3 artificial neurons. All artificial neurons use the ReLU activation function.**
   * **What is the shape of the input matrix X?**

**The shape of X will be (m x 10) where m is the batch size. The logic of a passthrough neuron is that it should have no effect at all on the input layer. Therefore, there must be one neuron for each input feature that allows the input feature to pass through to the next layer without altering its value in any way.**

* + **What about the shape of the hidden layer’s weight vector W*h*, and the shape of its bias vector b*h*?**

**Since the hidden layer has 50 neurons and there are 10 features, its weight vector will have the shape of 10 x 50. Because each of the 10 features will need to be multiplied by a weight which is connected to each of the 50 hidden layer neurons.**

**The bias vector will have a length of 50. This is because one bias unit is added to the output of the passthrough layer. That unit is then multiplied once for each of the 50 artificial neurons.**

* + **What is the shape of the output layer’s weight vector W*o*, and its bias vector b*o*?**

**The output layer has 3 neurons. The input to the output layer is the output of the hidden layer.The output of the hidden layer has 50 neurons, therefore the shape of the output layer's weight vector is 50 x 3.**

**The bias vector will have a length of 3 because one bias unit is added to the output of the hidden layer. That unit is then multiplied once for each of the 3 neurons in the output layer.**

* + **What is the shape of the network’s output matrix Y?**

**The shape of the output matrix is going to be m x 3, where m is the batch size, and 3 because each example will compute a probability that it belongs to one of three classes.**

* + **Write the equation that computes the network’s output matrix Y as a function of X, W*h*, b*h*, W*o* and b*o*.**

**Y = (X \* Wh + bh) \* (Wo + bo)**

**When adding a bias vector to a matrix it is added to every single row. This is called broadcasting.**

1. **How many neurons do you need in the output layer if you want to classify email into spam or ham? What activation function should you use in the output layer? If instead you want to tackle MNIST, how many neurons do you need in the output layer, using what activation function?**

**Email classification is a binary classification problem, so you would only need one neuron in the output layer. This neuron would indicate the probability that the email is spam or ham. You'd most likely use the sigmoid activation function in the output layer.**

**For the MNIST problem you would need 10 output neurons in the final layer, one for each digit. You would then replace the logistic function with the softmax function which can output one probability per class per digit.**

**Predicting housing prices is a linear regression problem. You'd only need one output neuron in the final layer. You wouldn't need to use an activation function at all.**

1. **What is backpropagation and how does it work? What is the difference between backpropagation and reverse-mode autodiff?**

**Backpropagation is an algorithm used to train neural networks. It first computes the gradients of the cost function with regards to every model parameter then it performs a gradient descent step using these gradients.**

**This backpropagation step is performed until the model parameters converge to values that hopefully minimize the cost function.**

**Backpropagation refers to the whole process of training a neural network. Reverse-mode autodiff is a technique to compute the gradients efficiently. It is used by the backprop algorithm.**

1. **Can you list all the hyperparameters you can tweak in an MLP? If the MLP overfits the training data, how could you tweak these hyperparameters to try to solve the problem?**

**In general, the hyperparameters of a neural network you can adjust are the number of hidden layers, the number of neurons in each hidden layer, and the activation function used by each neuron.**

**For binary classification, use the logistic activation function. For a multi-class problem, use softmax. For a linear regression problem, don't use an activation function.**

**Some simple ways to try and solve overfitting are reducing the number of hidden layers or the number of neurons.**

1. **Train a deep MLP on the MNIST dataset and see if you can get over 98% precision. Try adding all the bells and whistles (i.e., save checkpoints, restore the last checkpoint in case of an interruption, add summaries, plot learning curves using TensorBoard, and so on).**

Here is an code that trains a deep MLP on the MNIST dataset using Keras. It includes the implementation of the optimal learning rate search, saving checkpoints, early stopping, and plotting learning curves using TensorBoard.

**import tensorflow as tf**

**from tensorflow import keras**

**from tensorflow.keras.datasets import mnist**

**from tensorflow.keras.models import Sequential**

**from tensorflow.keras.layers import Dense**

**from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping, TensorBoard**

**from tensorflow.keras.optimizers import Adam**

**import numpy as np**

**import matplotlib.pyplot as plt**

**# Load the MNIST dataset**

**(X\_train, y\_train), (X\_test, y\_test) = mnist.load\_data()**

**# Preprocess the data**

**X\_train = X\_train.reshape(-1, 28\*28).astype('float32') / 255.0**

**X\_test = X\_test.reshape(-1, 28\*28).astype('float32') / 255.0**

**# Define the deep MLP model**

**model = Sequential()**

**model.add(Dense(256, activation='relu', input\_shape=(28\*28,)))**

**model.add(Dense(128, activation='relu'))**

**model.add(Dense(10, activation='softmax'))**

**# Compile the model**

**model.compile(loss='sparse\_categorical\_crossentropy',**

**optimizer='adam',**

**metrics=['accuracy'])**

**# Define the learning rate search function**

**def find\_learning\_rate(model, X, y, min\_lr=1e-5, max\_lr=1e-2, num\_lr=100):**

**lr\_values = np.logspace(np.log10(min\_lr), np.log10(max\_lr), num=num\_lr)**

**losses = []**

**for lr in lr\_values:**

**model.optimizer.lr = lr**

**hist = model.fit(X, y, epochs=1, batch\_size=128, verbose=0)**

**loss = hist.history['loss'][0]**

**losses.append(loss)**

**return lr\_values, losses**

**# Find the optimal learning rate**

**lr\_values, losses = find\_learning\_rate(model, X\_train, y\_train)**

**# Plot the learning rate search results**

**plt.plot(lr\_values, losses)**

**plt.xscale('log')**

**plt.xlabel('Learning Rate')**

**plt.ylabel('Loss')**

**plt.title('Learning Rate Search')**

**plt.show()**

**# Find the point where the loss shoots up**

**optimal\_lr = lr\_values[np.argmin(losses)]**

**print('Optimal learning rate:', optimal\_lr)**

**# Define callbacks**

**checkpoint\_cb = ModelCheckpoint('mnist\_model.h5', save\_best\_only=True)**

**early\_stopping\_cb = EarlyStopping(patience=10, restore\_best\_weights=True)**

**tensorboard\_cb = TensorBoard(log\_dir='./logs', histogram\_freq=1)**

**# Train the model with optimal learning rate**

**model.optimizer.lr = optimal\_lr**

**history = model.fit(X\_train, y\_train, epochs=100, batch\_size=128,**

**validation\_data=(X\_test, y\_test),**

**callbacks=[checkpoint\_cb, early\_stopping\_cb, tensorboard\_cb])**

**# Evaluate the model on the test set**

**test\_loss, test\_accuracy = model.evaluate(X\_test, y\_test)**

**print('Test Loss:', test\_loss)**

**print('Test Accuracy:', test\_accuracy)**

**Explanation:**

1. **The necessary libraries and modules are imported, including TensorFlow, Keras, and other components needed for building and training the model**
2. **The MNIST dataset is loaded using the `mnist.load\_data()` function. The dataset is split into training and testing sets, consisting of images of handwritten digits and their corresponding labels.**
3. **The data is preprocessed by reshaping the input images and scaling the pixel values between 0 and 1. This is done to ensure that the input data is in the appropriate format for the deep MLP model.**
4. **A deep MLP model is defined using the Keras Sequential API. The model consists of input layers, hidden layers with activation functions, and an output layer with the softmax activation function for multiclass classification.**
5. **The model is compiled with the specified loss function, optimizer, and metrics. In this case, the sparse categorical cross-entropy loss is used since the labels are integers, and the Adam optimizer is used for training the model.**
6. **The code defines a function called `find\_learning\_rate()` that performs the learning rate search. It takes the model, input data, and labels as inputs. Inside the function, a range of learning rates is defined, and for each learning rate, the model is trained for one epoch using the specified learning rate. The loss value from each training epoch is recorded.**
7. **The learning rate search is performed by calling the `find\_learning\_rate()` function with the model, training data, and labels. The function returns the learning rate values and the corresponding loss values.**
8. **The learning rate search results are plotted using Matplotlib. The learning rate values are plotted on the x-axis, and the loss values are plotted on the y-axis. This plot helps identify the point where the loss starts to increase rapidly, indicating the optimal learning rate.**
9. **The optimal learning rate is determined as the learning rate value corresponding to the minimum loss. This value is then used to set the learning rate of the model's optimizer.**
10. **Checkpointing, early stopping, and TensorBoard callbacks are defined. The ModelCheckpoint callback saves the best model during training based on the validation loss. The EarlyStopping callback stops training if the validation loss does not improve for a specified number of epochs. The TensorBoard callback logs training statistics for visualization using TensorBoard.**
11. **The model is trained using the optimal learning rate. The training data, labels, and callbacks are provided to the `model.fit()` function. The model is trained for a specified number of epochs, with a batch size of 128.**
12. **After training, the model is evaluated on the test set using the `model.evaluate()` function. The test loss and accuracy are printed.**

**This code will train a deep MLP model on the MNIST dataset, search for the optimal learning rate, save the best model checkpoint, use early stopping to prevent overfitting, and plot learning curves using TensorBoard. The final test loss and accuracy will be printed after training and evaluation.**

**Make sure you have TensorFlow and Keras installed. You can also install additional packages like matplotlib and tensorboard using pip if you don't have them already.**

**Note: You need to have TensorFlow version 2.0 or higher to run this code, as it uses the Keras API integrated within TensorFlow.**

**the given code trains a deep MLP model on the MNIST dataset using Keras. It includes features such as searching for the optimal learning rate, saving checkpoints, using early stopping, and plotting learning curves using TensorBoard. The code preprocesses the data, defines the model architecture, compiles the model, performs a learning rate search, trains the model with the optimal learning rate, and evaluates its performance on the test set.**