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?**

* There are a few reasons why logistic regression is generally preferred over the classical perceptron.

First, logistic regression is a probabilistic model, which means that it can output not only the most likely class, but also the probability that an instance belongs to a particular class. This can be useful for evaluating the confidence of a model's predictions, and for comparing the relative likelihood of different classes.

Second, logistic regression can be regularized, which means that it can be trained to prefer simpler models that have fewer parameters. This can help to prevent overfitting and improve the generalization ability of the model.

Third, logistic regression can be trained using an iterative optimization algorithm such as gradient descent, which can be more efficient and numerically stable than the methods used to train the classical perceptron.

In summary, logistic regression is generally preferred over the classical perceptron because it is probabilistic, can be regularized, and can be trained using efficient optimization algorithms.

* A perceptron is a type of linear classifier, while logistic regression is a type of non-linear classifier. However, it is possible to modify the perceptron algorithm to make it equivalent to logistic regression.

One way to do this is to use a sigmoid activation function instead of a step function in the perceptron. The sigmoid function takes in a real-valued input and squashes it to a value between 0 and 1, which can be interpreted as a probability. Then, the output of the perceptron can be interpreted as the probability that a given input belongs to a certain class. This makes the perceptron algorithm similar to logistic regression, which also produces probabilities for each class.

Another way to make a perceptron equivalent to a logistic regression classifier is to use a cross-entropy loss function instead of a hinge loss function. The cross-entropy loss is commonly used in classification tasks, and is the default loss function for logistic regression. By using this loss function, the perceptron can be trained to minimize the difference between the predicted probabilities and the true probabilities, just like logistic regression.

Overall, by using a sigmoid activation function and a cross-entropy loss function, it is possible to modify a perceptron to make it equivalent to a logistic regression classifier.

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

The logistic activation function, also known as the sigmoid function, was a key ingredient in the training of the first multi-layer perceptrons (MLPs) because it helps to normalize the output of each neuron to a value between 0 and 1. This is useful in MLPs because it allows the model to represent the probability that an input belongs to a certain class, which is often the desired output in classification tasks. Additionally, the logistic function has a derivative that is easy to compute, which is useful for training the model using gradient descent.

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

Three popular activation functions are the logistic (sigmoid), hyperbolic tangent (tanh), and rectified linear unit (ReLU) functions.

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**?
   * What about the shape of the hidden layer’s weight vector **W***h*, and the shape of its bias vector **b***h*?
   * What is the shape of the output layer’s weight vector **W***o*, and its bias vector **b***o*?
   * What is the shape of the network’s output matrix **Y**?
   * 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*.
2. **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?**

If we want to classify email as spam or ham, we would need two neurons in the output layer, one for each class. In this case, it is common to use a logistic (sigmoid) activation function in the output layer, as it will output a value between 0 and 1 that can be interpreted as a probability.

If we want to tackle MNIST, a common approach is to use a one-hot encoding for the output classes, in which each output neuron corresponds to a different digit class. In this case, we would need 10 output neurons, one for each digit from 0 to 9. In this case, it is common to use a softmax activation function in the output layer, as it will normalize the outputs of the neurons to sum to 1, allowing them to be interpreted as probabilities.

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

Backpropagation is a method for training artificial neural networks. It is a way of computing the gradients of the loss function with respect to the weights of the network, which can be used to update the weights in order to minimize the loss. This process is also known as gradient descent.

Backpropagation works by starting at the final layer of the network, and then working backwards through the layers, using the chain rule to calculate the gradient of the loss function with respect to the weights at each layer. This is done by first calculating the gradient of the loss function with respect to the output of each layer, and then using this gradient to calculate the gradient with respect to the weights.

Reverse-mode autodiff is a method for calculating gradients that is similar to backpropagation. It also uses the chain rule to calculate the gradients, but it does so in a different way. Instead of working backwards through the layers of the network, reverse-mode autodiff starts at the inputs and works forwards through the layers, calculating the gradient at each step.

The main difference between backpropagation and reverse-mode autodiff is the direction in which the gradients are calculated. Backpropagation works backwards through the layers of the network, while reverse-mode autodiff works forwards. This difference can affect the efficiency of the two methods, with backpropagation being more efficient for networks with many layers, and reverse-mode autodiff being more efficient for networks with few layers.

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 an MLP (multi-layer perceptron), the following are some common hyperparameters that can be adjusted:**

The number of hidden layers

The number of neurons in each hidden layer

The activation function used in each hidden layer

The learning rate

The batch size

The number of epochs (i.e., the number of times the model is trained on the entire dataset)

If an MLP is overfitting the training data, one possible solution is to try using regularization. This can be achieved by adjusting the following hyperparameters:

The L1 or L2 regularization strength

The dropout rate (i.e., the probability of dropping out a given neuron during training)

Another possible solution is to try reducing the complexity of the model by reducing the number of hidden layers or the number of neurons in each hidden layer. This can help prevent the model from overfitting the training data.

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).