The perceptron algorithm consists of one of the simplest types of neural network architectures. It is a single-layer neural network with a single neuron. The neuron is a linear combination of the input variables, with a bias term, and then passed through an activation function. The activation function is used to determine the output of the neuron. The algorithm itself is a supervised learning algorithm, meaning that it requires labeled training data to train the model. The algorithm is iterative, hence it'll continue to update the weights until the model converges. The learning rate is a hyperparameter that controls the step size of the weight updates. The activation function is a hyperparameter that determines the output of the neuron. The learning rule is as follows:

$$w^{new} \leftarrow w^{old} + \eta \cdot (z - \hat{z}(x)) \cdot x, \quad \hat{z}(x) = f(net(x)), \quad net(x) = \sum_{i=1}^{n} w_i \cdot x^{(i)} + w_0$$

where w_i is the weight for the ith input variable, η is the learning rate, z is the true label, $\hat{z}$ is the predicted label, and x_i is the ith input variable. The learning rule is applied for each training example, and an epoch is a single pass through the entire training set.

1. Considering the following linearly separable training data:

	y_1	y_2	y_3	z
x_1	0	0	0	-1
x_2	0	2	1	1
x_3	1	1	1	1
x_4	1	-1	0	-1

Given the perceptron learning algorithm with a learning rate $\eta = 1$, sign activation and all weights initialized to one (including the bias):

(a) Considering y_1 and y_2, apply the algorithm until convergence. Draw the separation hyperplane.

(b) Considering all input variables, apply one epoch of the algorithm. Do weights change for an additional epoch?

(c) Identify the perceptron output for $x_{new} = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T$.

(d) What happens if we replace the sign function with the step function? Specifically, how would you change η to ensure the same results?

(a) As per the question statement, we're working with $\eta = 1$ and sign activation:

$$\hat{z} = f(net) = \begin{cases} 1 & net \geq 0 \\ -1 & net < 0 \end{cases}$$

Moreover, we're considering weights (and the bias, w_0) initialized at one. Performing epochs following the $\{x_1, \cdots, x_4\}$ order, we get the following weight updates:

$$\begin{aligned}
\hat{z}(x_1) &= f(net(x_1)) = f(1 + 1 \cdot 0 + 1 \cdot 0) = f(1) = 1 \\
\hat{z}(x_2) &= f(net(x_2)) = f(1 + 1 \cdot 0 + 1 \cdot 2) = f(3) = 1 \\
\hat{z}(x_3) &= f(net(x_3)) = f(1 + 1 \cdot 1 + 1 \cdot 1) = f(3) = 1 \\
\hat{z}(x_4) &= f(net(x_4)) = f(1 + 1 \cdot 1 + 1 \cdot -1) = f(1) = 1
\end{aligned}$$

$$x_1: \quad w \leftarrow \begin{bmatrix}1\\1\\1\end{bmatrix} + 1 \cdot (-1-1) \cdot \begin{bmatrix}1 & 0 & 0\end{bmatrix} = \begin{bmatrix}-1\\1\\1\end{bmatrix}$$
$$x_2: \quad w \leftarrow \begin{bmatrix}-1\\1\\1\end{bmatrix} + 1 \cdot (1-1) \cdot \begin{bmatrix}1 & 0 & 2\end{bmatrix} = \begin{bmatrix}-1\\1\\1\end{bmatrix}$$
$$x_3: \quad w \leftarrow \begin{bmatrix}-1\\1\\1\end{bmatrix} + 1 \cdot (1-1) \cdot \begin{bmatrix}1 & 1 & 1\end{bmatrix} = \begin{bmatrix}-1\\1\\1\end{bmatrix}$$
$$x_4: \quad w \leftarrow \begin{bmatrix}-1\\1\\1\end{bmatrix} + 1 \cdot (-1-1) \cdot \begin{bmatrix}1 & 1 & -1\end{bmatrix} = \begin{bmatrix}-1\\1\\1\end{bmatrix}$$

After entering a new epoch, we'd update the weights again, this time for x_1; since such an update wouldn't lead to an actual update on the weight matrix, that'd make a full pass on the training set without changes, and, as such, the algorithm would converge. Therefore, the regression hyperplane is given by:

$$-1 + x_1 + x_2 = 0 \leftrightarrow x_2 = 1 - x_1$$

(b) Here, we apply the same algorithm as before, but now considering all input variables.

$$\hat{z}(x_1) = f(net(x_1)) = f(1 + 1 \cdot 0 + 1 \cdot 0 + 1 \cdot 0) = f(1) = 1$$
$$\hat{z}(x_2) = f(net(x_2)) = f(1 + 1 \cdot 0 + 1 \cdot 2 + 1 \cdot 1) = f(4) = 1$$
$$\hat{z}(x_3) = f(net(x_3)) = f(1 + 1 \cdot 1 + 1 \cdot 1 + 1 \cdot 1) = f(4) = 1$$
$$\hat{z}(x_4) = f(net(x_4)) = f(1 + 1 \cdot 1 + 1 \cdot -1 + 1 \cdot 0) = f(2) = 1$$

$$x_1: \quad w \leftarrow \begin{bmatrix}1\\1\\1\\1\end{bmatrix} + 1 \cdot (-1-1) \cdot \begin{bmatrix}1 & 0 & 0 & 0\end{bmatrix} = \begin{bmatrix}-1\\1\\1\\1\end{bmatrix}$$
$$x_2: \quad w \leftarrow \begin{bmatrix}-1\\1\\1\\1\end{bmatrix} + 1 \cdot (1-1) \cdot \begin{bmatrix}1 & 0 & 2 & 1\end{bmatrix} = \begin{bmatrix}-1\\1\\1\\1\end{bmatrix}$$
$$x_3: \quad w \leftarrow \begin{bmatrix}-1\\1\\1\\1\end{bmatrix} + 1 \cdot (1-1) \cdot \begin{bmatrix}1 & 1 & 1 & 1\end{bmatrix} = \begin{bmatrix}-1\\1\\1\\1\end{bmatrix}$$
$$x_4: \quad w \leftarrow \begin{bmatrix}-1\\1\\1\\1\end{bmatrix} + 1 \cdot (-1-1) \cdot \begin{bmatrix}1 & 1 & -1 & 0\end{bmatrix} = \begin{bmatrix}-1\\1\\1\\1\end{bmatrix}$$

Once again, we'd enter a new epoch and update the weights considering the first sample, but since such an update wouldn't lead to an actual update on the weight matrix, that'd make a full pass on the training set without changes, and, as such, the algorithm would converge. **An additional epoch wouldn't, therefore, change the weights**.

(c) As we've mentioned before, the perceptron's output is given by the activation function - here, the sign of the net input. Therefore, the perceptron's output is given by:

$$\hat{z} = f(net) = \begin{cases} 1 & net \geq 0 \\ -1 & net < 0 \end{cases}$$

Here, considering the weights computed in the previous question, we'll have the following:

$$net(x_{new}) = -1 + 1 \cdot 0 + 1 \cdot 0 + 1 \cdot 1 = 0$$

As we know, $f(0) = 1$, and, as such, the perceptron would classify the new sample as belonging to the binary class 1.

(d) As we know, the step function is rather similar to the sign function, defined as follows:

$$f(net) = \begin{cases} 1 & net \geq 0 \\ 0 & net < 0 \end{cases}$$

Let's consider the following learning rules (the first one considers the activation function as the sign function, and the second one considers the activation function as the step function):

$$\begin{aligned} &\text{Sign function:} \quad w \leftarrow w + \eta \cdot (z - sign(net)) \cdot x \\ &\text{Step function:} \quad w \leftarrow w + \eta \cdot (z - step(net)) \cdot x \end{aligned}$$

We can note, of course, that the only differing term between both rules is $(z - \hat{z})$; therefore, to make it so that both rules are equivalent, we'd have to make sure that the step function's output is correctly balanced with the sign function's output, utilizing a scalar factor (be it τ) for such purpose, effectively altering the learning rate to be $\eta_{sign} = \tau \cdot \eta_{step}$. Let's try to find τ:

$$z - sign(net) = \tau(z - step(net))$$

We know that $z \in \{-1, 1\}$, and, as such, we can write each side of the equation's interval as follows:

$$[-1, 1] - [-1, 1] = \tau([-1, 1] - [0, 1]) \leftrightarrow [-2, 2] = \tau[-1, 1]$$

As such, if we want an equal learning rule utilizing both step and sign functions, we'd have to set $\tau = 2$, and, as such, get the following learning rule:

$$w \leftarrow w + 2\eta_{sign} \cdot (z - step(net)) \cdot x \quad \leftrightarrow \quad w \leftarrow w + \eta_{sign} \cdot (z - sign(net)) \cdot x$$

2. Let us consider the following activation function:

$$\hat{z}(x, w) = \frac{1}{1 + e^{-2wx}}$$

Consider also the half sum of squared errors as the loss function:

$$E(w) = 1/2 \sum_{i=1}^{N} (z_i - \hat{z}(x_i, w))^2$$

(a) Determine the gradient descent learning rule for this unit.

(b) Compute the first gradient descent update, assuming an initialization of all ones.

(c) Compute the first stochastic gradient descent update assuming an initialization of all ones.

(a)

(b)

(c)

3. Let us consider the following activation function:

$$\hat{z}(x, w) = \frac{1}{1 + e^{-wx}}$$

Here, we'll be using the cross-entropy loss function:

$$E(w) = -\sum_{i=1}^{N} z_i \log \hat{z}(x_i, w) + (1 - z_i) \log(1 - \hat{z}(x_i, w))$$

(a) Determine the gradient descent learning rule for this unit.

(b) Compute the first gradient descent update, assuming an initialization of all ones.

(c) Compute the first stochastic gradient descent update assuming an initialization of all ones.

(a)

(b)

(c)

4. Consider now the activation function described in the previous exercise, paired with the half sum of squared errors loss function.

(a) Determine the gradient descent learning rule for this unit.

(b) Compute the stochastic gradient descent update for input $x_{new} = \begin{bmatrix} 1 & 1 \end{bmatrix}^T$, $z_{new} = 0$, with initial weights $w = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix}^T$ and learning rate $\eta = 2$.

(a)

(b)

5. Consider the sum squared and cross-entropy loss functions. Any stands out? What changes when one changes the loss function?