Foundations of Machine Learning: Week 2: Perceptrons

Professor Christopher Yau
christopher.yau@manchester.ac.uk

October 30, 2020

Foundations of Machine Learning: Week 2: Perceptrons

2020-10-30

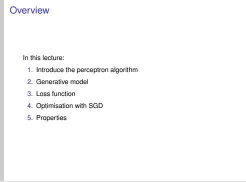
Foundations of Machine Learning:
Week 2: Perceptrons

Professor Christopher Yau
christopher.yau@manchester.ac.uk
October 30, 2020

Overview

In this lecture:

1. Introduce the perceptron algorithm
2. Generative model
3. Loss function
4. Optimisation with SGD
5. Properties


Overview

In this lecture:

1. Introduce the perceptron algorithm
2. Generative model
3. Loss function
4. Optimisation with SGD
5. Properties

In this lecture, we will use the theory we have established to construct a **perceptron** algorithm.
This is a very simple deterministic binary classification algorithm but whose construction has features that still underpin modern machine learning techniques.
I will talk about the generative models that underlie the perceptron and the loss function we need to optimise using stochastic gradient descent to train the classifier.
I will then talk about some of the properties.

Linear classifiers

Linear classifiers use a weighted (**w**) linear function of the inputs and a bias term w_0:

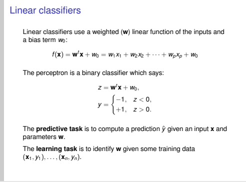
$$f(\mathbf{x}) = \mathbf{w}^t\mathbf{x} + w_0 = w_1x_1 + w_2x_2 + \cdots + w_px_p + w_0$$

The perceptron is a binary classifier which says:

$$z = \mathbf{w}^t\mathbf{x} + w_0,$$
$$y = \begin{cases} -1, & z < 0, \\ +1, & z > 0. \end{cases}$$

The **predictive task** is to compute a prediction $\hat{y}$ given an input **x** and parameters **w**.

The **learning task** is to identify **w** given some training data $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$.


Linear classifiers

Linear classifiers use a weighted (**w**) linear function of the inputs and a bias term w_0:

$$f(\mathbf{x}) = \mathbf{w}^t\mathbf{x} + w_0 = w_1x_1 + w_2x_2 + \cdots + w_px_p + w_0$$

The perceptron is a binary classifier which says:

$$z = \mathbf{w}^t\mathbf{x} + w_0,$$
$$y = \begin{cases} -1, & z < 0, \\ +1, & z > 0. \end{cases}$$

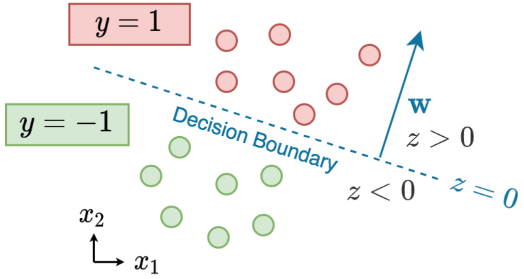
The **predictive task** is to compute a prediction $\hat{y}$ given an input **x** and parameters **w**.

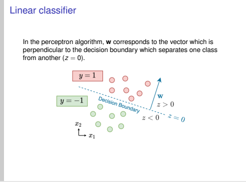
The **learning task** is to identify **w** given some training data $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$.

First, the perceptron is known as a linear classifier.
What this means is that the inputs **x** are scaled by some parameters or coefficients **w** and added together possibly with a bias or intercept term w_0 added on too.
The perceptron uses that weighted linear combination to form a latent variable z and then gives an output classification y which is based on the sign of z.
So if after computing the $\mathbf{w}^t\mathbf{x}$ we get a positive value of z then we assign the predicted output to be $y = 1$ else, if z is negative, we set $y = -1$.
There are two tasks we can perform with a perceptron, we can make a prediction if we are given an input **x** and some parameters $(\mathbf{w}, w_0)$.
Alternatively, and what we are principally concerned with in this lecture, is the learning task which is to identify the **w** that give rise to a data set of exemplar input and output pairs.

Linear classifier

In the perceptron algorithm, **w** corresponds to the vector which is perpendicular to the decision boundary which separates one class from another ($z = 0$).





To illustrate, if we have two classes of data, the perceptron tries to find a separating plane, known as a decision boundary which divides the input space between the two classes of output.
Along this decision boundary, the latent variable $z = 0$ but on one side $z > 0$ and the other $z < 0$ and this determines what the output classification would be.
The weights or parameters **w** correspond to a vector which points in a direction perpendicular to the separating plane.
Therefore by optimising **w** we are directly optimising the decision boundary.

Generative Process

A **generative process** describes the way in which data is generated.

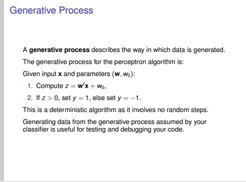
The generative process for the perceptron algorithm is:

Given input $\mathbf{x}$ and parameters $(\mathbf{w}, w_0)$:

1. Compute $z = \mathbf{w}^t\mathbf{x} + w_0$,
2. If $z > 0$, set $y = 1$, else set $y = -1$.

This is a deterministic algorithm as it involves no random steps.

Generating data from the generative process assumed by your classifier is useful for testing and debugging your code.



Generative Process

A **generative process** describes the way in which data is generated.
The generative process for the perceptron algorithm is:
Given input $\mathbf{x}$ and parameters $(\mathbf{w}, w_0)$:
1. Compute $z = \mathbf{w}^t\mathbf{x} + w_0$,
2. If $z > 0$, set $y = 1$, else set $y = -1$.
This is a deterministic algorithm as it involves no random steps.
Generating data from the generative process assumed by your classifier is useful for testing and debugging your code.

Many classifiers make an assumption about how data is generated. If these assumptions match the truth then that classifier will be optimal for the problem. We can encode this in an algorithm and for the perceptron, the previous slide tells us that the generative process is as follows:
Given input $\mathbf{x}$ and parameters $(\mathbf{w}, w_0)$. Then compute $z = \mathbf{w}^t\mathbf{x} + w_0$. If $z > 0$, set $y = 1$, otherwise set $y = -1$.
This is a deterministic algorithm because the output depends **not** depend on any random steps.
So if I can generate some inputs $\mathbf{x}$, I can use this procedure to find the matching outputs y which is in line with how the perceptron wants to work.
Generating data from the generative process assumed by your classifier is useful for testing and debugging your code because we know it should perform well.
The data generating mechanism for real data is unknown so not useful for testing and debugging because we would not know if our approach would ever work.

Loss function

If the prediction $\hat{y} = \mathrm{sign}(z)$ is correct then:

$$y \cdot \underbrace{(\mathbf{w}^t\mathbf{x} + w_0)}_{z} > 0,$$

else

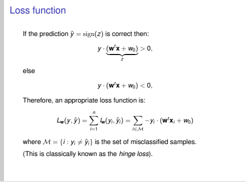
$$y \cdot (\mathbf{w}^t\mathbf{x} + w_0) < 0,$$

Therefore, an appropriate loss function is:

$$L_{\mathbf{w}}(y, \hat{y}) = \sum_{i=1}^{n} l_{\mathbf{w}}(y_i, \hat{y}_i) = \sum_{i \in \mathcal{M}} -y_i \cdot (\mathbf{w}^t\mathbf{x}_i + w_0)$$

where $\mathcal{M} = \{i : y_i \neq \hat{y}_i\}$ is the set of misclassified samples.

(This is classically known as the *hinge loss*).


Loss function

If the prediction $\hat{y} = \mathrm{sign}(z)$ is correct then:

$$y \cdot \underbrace{(\mathbf{w}^t\mathbf{x} + w_0)}_{z} > 0,$$

else

$$y \cdot (\mathbf{w}^t\mathbf{x} + w_0) < 0,$$

Therefore, an appropriate loss function is:

$$L_{\mathbf{w}}(y, \hat{y}) = \sum_{i=1}^{n} l_{\mathbf{w}}(y_i, \hat{y}_i) = \sum_{i \in \mathcal{M}} -y_i \cdot (\mathbf{w}^t\mathbf{x}_i + w_0)$$

where $\mathcal{M} = \{i : y_i \neq \hat{y}_i\}$ is the set of misclassified samples.

(This is classically known as the *hinge loss*).

Now we know how the perceptron classifies and what it assumes about how the data is generated, lets build our loss function so that we can train a perceptron to perform a binary classification task.
First, what happens when the perceptron makes a mistake?
We might notice that if the perceptron makes a correct classification then y times z will be greater than zero but if the classification is incorrect than $y \times z$ is always negative.
Therefore one possible loss function is to add up all those negative values for the misclassified samples and use that total as our loss.
This is known as the hinge loss. Note we multiply by -1 because we want to do loss minimisation later.

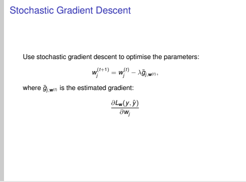
Stochastic Gradient Descent

Use stochastic gradient descent to optimise the parameters:

$$w_j^{(t+1)} = w_j^{(t)} - \lambda \tilde{g}_{j,\mathbf{w}^{(t)}},$$

where $\tilde{g}_{j,\mathbf{w}^{(t)}}$ is the estimated gradient:

$$\frac{\partial L_{\mathbf{w}}(y, \hat{y})}{\partial w_j}$$



Stochastic Gradient Descent

Use stochastic gradient descent to optimise the parameters:

$$w_j^{(t+1)} = w_j^{(t)} - \lambda \tilde{g}_{j,\mathbf{w}^{(t)}},$$

where $\tilde{g}_{j,\mathbf{w}^{(t)}}$ is the estimated gradient:

$$\frac{\partial L_{\mathbf{w}}(y, \hat{y})}{\partial w_j}$$

We now want to use stochastic gradient descent to minimise this loss function with respect to the parameters **w**.
Recall the form of the stochastic gradient descent update where we take the current parameter and add a small step in the direction of an approximate gradient estimate.

Stochastic Gradient Descent

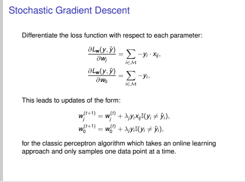
Differentiate the loss function with respect to each parameter:

$$\frac{\partial L_{\mathbf{w}}(y, \hat{y})}{\partial w_j} = \sum_{i \in \mathcal{M}} -y_i \cdot x_{ij},$$
$$\frac{\partial L_{\mathbf{w}}(y, \hat{y})}{\partial w_0} = \sum_{i \in \mathcal{M}} -y_i,$$

This leads to updates of the form:

$$w_j^{(t+1)} = w_j^{(t)} + \lambda_j y_i x_{ij} \mathbb{I}(y_i \neq \hat{y}_i),$$
$$w_0^{(t+1)} = w_0^{(t)} + \lambda_j y_i \mathbb{I}(y_i \neq \hat{y}_i),$$

for the classic perceptron algorithm which takes an online learning approach and only samples one data point at a time.


Stochastic Gradient Descent

Differentiate the loss function with respect to each parameter:

$$\frac{\partial L_{\mathbf{w}}(y, \hat{y})}{\partial w_j} = \sum_{i \in \mathcal{M}} -y_i \cdot x_{ij},$$
$$\frac{\partial L_{\mathbf{w}}(y, \hat{y})}{\partial w_0} = \sum_{i \in \mathcal{M}} -y_i,$$

This leads to updates of the form:

$$w_j^{(t+1)} = w_j^{(t)} + \lambda_j y_i x_{ij} \mathbb{I}(y_i \neq \hat{y}_i),$$
$$w_0^{(t+1)} = w_0^{(t)} + \lambda_j y_i \mathbb{I}(y_i \neq \hat{y}_i),$$

for the classic perceptron algorithm which takes an online learning approach and only samples one data point at a time.

If we differentiate the loss function with respect to our parameters, we find that the derivatives are actually simple functions of the inputs **x** and/or the outputs, summed over all currently misclassified samples.
For the classic perceptron, we use a purely online learning approach and actually only sample one data point at a time to estimate the gradient rather than using the total overall misclassified samples.
That is, we randomly pick one misclassified sample, and add x_{ij} to w_j scaled by a learning rate λ to get an updated parameter estimate for w_j. In the case of the bias term w_0, we add +1 or -1 depending on the value of the true output.
The update can also be expressed in vector form which saves us from having to enumerate each element of **w** .

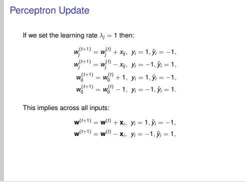
Perceptron Update

If we set the learning rate $\lambda_j = 1$ then:

$$\begin{aligned} w_j^{(t+1)} &= w_j^{(t)} + x_{ij},\ \ y_i = 1, \hat{y}_i = -1, \\ w_j^{(t+1)} &= w_j^{(t)} - x_{ij},\ \ y_i = -1, \hat{y}_i = 1, \\ w_0^{(t+1)} &= w_0^{(t)} + 1,\ \ y_i = 1, \hat{y}_i = -1, \\ w_0^{(t+1)} &= w_0^{(t)} - 1,\ \ y_i = -1, \hat{y}_i = 1. \end{aligned}$$

This implies across all inputs:

$$\begin{aligned} \mathbf{w}^{(t+1)} &= \mathbf{w}^{(t)} + \mathbf{x}_i,\ \ y_i = 1, \hat{y}_i = -1, \\ \mathbf{w}^{(t+1)} &= \mathbf{w}^{(t)} - \mathbf{x}_i,\ \ y_i = -1, \hat{y}_i = 1, \end{aligned}$$



Perceptron Update

If we set the learning rate $\lambda_j = 1$ then:

$$\begin{aligned} w_j^{(t+1)} &= w_j^{(t)} + x_{ij},\ \ y_i = 1, \hat{y}_i = -1, \\ w_j^{(t+1)} &= w_j^{(t)} - x_{ij},\ \ y_i = -1, \hat{y}_i = 1, \\ w_0^{(t+1)} &= w_0^{(t)} + 1,\ \ y_i = 1, \hat{y}_i = -1, \\ w_0^{(t+1)} &= w_0^{(t)} - 1,\ \ y_i = -1, \hat{y}_i = 1. \end{aligned}$$

This implies across all inputs:

$$\begin{aligned} \mathbf{w}^{(t+1)} &= \mathbf{w}^{(t)} + \mathbf{x}_i,\ \ y_i = 1, \hat{y}_i = -1, \\ \mathbf{w}^{(t+1)} &= \mathbf{w}^{(t)} - \mathbf{x}_i,\ \ y_i = -1, \hat{y}_i = 1, \end{aligned}$$

If we set the learning rate to one then updates have a very simple form were we increment **w** by an amount equal to the input in the direction to correct the misclassification.
So, for example, if the true output is $y = 1$ but we estimate $\hat{y} = -1$, then the updated w_j is given by the current w_j plus x_{ij}.
However, if the true output was $y = -1$ and we estimate $\hat{y} = 1$ then the updated w_j is w_j minus x_{ij}.

Numerical Example

Q: Given $\mathbf{w} = (w_1, w_2) = (0.5, 0.3)$, $w_0 = 1$ and a training sample $x = (-1, 1)$, $y = -1$, what is the perceptron update for $\mathbf{w}$, w_0?

Solution:

1. Compute the predicted output given the current parameters:

$$z = w_1 x_1 + w_2 x_2 + w_0 = -1(0.5) + 1(0.3) + 1 = +0.8$$

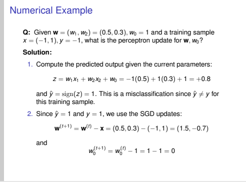
   and $\hat{y} = \mathrm{sign}(z) = 1$. This is a misclassification since $\hat{y} \neq y$ for this training sample.

2. Since $\hat{y} = 1$ and $y = 1$, we use the SGD updates:

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \mathbf{x} = (0.5, 0.3) - (-1, 1) = (1.5, -0.7)$$

   and

$$w_0^{(t+1)} = w_0^{(t)} - 1 = 1 - 1 = 0$$


Numerical Example
Q: Given $\mathbf{w} = (w_1, w_2) = (0.5, 0.3)$, $w_0 = 1$ and a training sample $x = (-1, 1)$, $y = -1$, what is the perceptron update for $\mathbf{w}$, w_0?
Solution:
1. Compute the predicted output given the current parameters:
$z = w_1 x_1 + w_2 x_2 + w_0 = -1(0.5) + 1(0.3) + 1 = +0.8$
and $\hat{y} = \mathrm{sign}(z) = 1$. This is a misclassification since $\hat{y} \neq y$ for this training sample.
2. Since $\hat{y} = 1$ and $y = 1$, we use the SGD updates:
$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \mathbf{x} = (0.5, 0.3) - (-1, 1) = (1.5, -0.7)$
and
$w_0^{(t+1)} = w_0^{(t)} - 1 = 1 - 1 = 0$

Lets consider a numerical example:
Suppose we have weights $\mathbf{w} = (0.5, 0.3)$ and a bias $w_0 = 1$. We are given a training sample with input $\mathbf{x} = (-1, 1)$ and $y = -1$.
What is the perceptron update?
First, we need to compute the predicted output given the current parameters, so we compute z and taking the weighted linear average of the inputs $\mathbf{x}$ given $\mathbf{w}$ and the bias w_0. In this case this gives us $z = 0.8$. The predicted output $\hat{y}$ is the sign of z so the prediction is 1.
This is a misclassification as the true output is -1.
Since this is an error, we need to update the perceptron parameters. The previous slide showed we should subtract the input from $\mathbf{w}$ and decrease the bias w_0 by 1 to get new parameters.

Numerical Example

Q: Given $\mathbf{w} = (w_1, w_2) = (0.5, 0.3)$, $w_0 = 1$ and a training sample $x = (-1, 1), y = -1$, what is the perceptron update for $\mathbf{w}$?

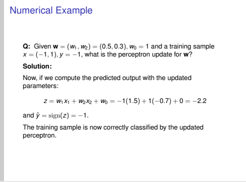
Solution:

Now, if we compute the predicted output with the updated parameters:

$$z = w_1x_1 + w_2x_2 + w_0 = -1(1.5) + 1(-0.7) + 0 = -2.2$$

and $\hat{y} = \mathrm{sign}(z) = -1$.

The training sample is now correctly classified by the updated perceptron.


Numerical Example

Q: Given $\mathbf{w} = (w_1, w_2) = (0.5, 0.3)$, $w_0 = 1$ and a training sample $x = (-1, 1), y = -1$, what is the perceptron update for $\mathbf{w}$?

Solution:

Now, if we compute the predicted output with the updated parameters:

$$z = w_1x_1 + w_2x_2 + w_0 = -1(1.5) + 1(-0.7) + 0 = -2.2$$

and $\hat{y} = \mathrm{sign}(z) = -1$.

The training sample is now correctly classified by the updated perceptron.

Now that we have updated parameters, lets check the updated prediction. We use the new parameters to compute z given the inputs $\mathbf{x}$ and this time $z = -2.2$ which means, taking the sign, the output is now predicted to be -1. This is the correct answer.
So the perceptron update has modified the parameters to now correctly classify the training sample.
In a real analysis, there would be multiple data points and we would cycle through these updating the parameters until the performance of the perceptron can be improved no further and the loss function is minimised.

Intuition

What is the update doing?

If we had previously predicted $\hat{y}_i = -1$ when $y_i = 1$ then the update would be:

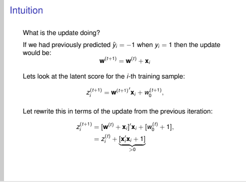
$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \mathbf{x}_i$$

Lets look at the latent score for the i-th training sample:

$$z_i^{(t+1)} = \mathbf{w}^{(t+1)\prime}\mathbf{x}_i + w_0^{(t+1)},$$

Let rewrite this in terms of the update from the previous iteration:

$$\begin{aligned} z_i^{(t+1)} &= [\mathbf{w}^{(t)} + \mathbf{x}_i]'\mathbf{x}_i + [w_0^{(t)} + 1], \\ &= z_i^{(t)} + \underbrace{[\mathbf{x}_i'\mathbf{x}_i + 1]}_{>0} \end{aligned}$$


Intuition

What is the update doing?

If we had previously predicted $\hat{y}_i = -1$ when $y_i = 1$ then the update would be:

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \mathbf{x}_i$$

Lets look at the latent score for the i-th training sample:

$$z_i^{(t+1)} = \mathbf{w}^{(t+1)\prime}\mathbf{x}_i + w_0^{(t+1)},$$

Let rewrite this in terms of the update from the previous iteration:

$$\begin{aligned} z_i^{(t+1)} &= [\mathbf{w}^{(t)} + \mathbf{x}_i]'\mathbf{x}_i + [w_0^{(t)} + 1], \\ &= z_i^{(t)} + \underbrace{[\mathbf{x}_i'\mathbf{x}_i + 1]}_{>0} \end{aligned}$$

We can ask ourselves, what is the update doing?
Suppose we predict -1 but the true output for a given input is actually 1.
Then the update we derived says we should add that input to the parameter.
Lets look at the latent score z. Remember z is the dot product of the current estimate of **w** and **x** plus a bias term.
If we rewrite the current parameter estimate in terms of the update from its last value, we can re-derive the current latent score $z_i^{(t+1)}$ in terms of its previous score $z_i^{(t)}$ and an additional term due to the update.
This additional term $\mathbf{x}_i'\mathbf{x}_i + 1$ is always positive valued. So the update always increases the latent score.

Intuition

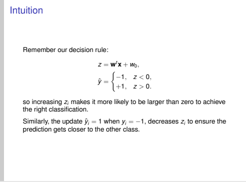
Remember our decision rule:

$$z = \mathbf{w}^t\mathbf{x} + w_0,$$

$$\hat{y} = \begin{cases} -1, & z < 0, \\ +1, & z > 0. \end{cases}$$

so increasing z_i makes it more likely to be larger than zero to achieve the right classification.

Similarly, the update $\hat{y}_i = 1$ when $y_i = -1$, decreases z_i to ensure the prediction gets closer to the other class.



Intuition

Remember our decision rule:

$$z = \mathbf{w}^t\mathbf{x} + w_0,$$

$$\hat{y} = \begin{cases} -1, & z < 0, \\ +1, & z > 0. \end{cases}$$

so increasing z_i makes it more likely to be larger than zero to achieve the right classification.

Similarly, the update $\hat{y}_i = 1$ when $y_i = -1$, decreases z_i to ensure the prediction gets closer to the other class.

So the update used the misclassified sample to modify the weights so that the latent score z_i for the i-th sample increased from the t to $t+1$ iteration. If you recall the decision rule, $z > 0$ means an assignment to $y = 1$, since $y = 1$ is the true output here, the update brings the decision boundary closer to making this true.

Similarly, if the misclassification was the other way around, we would update and shift the decision boundary in the opposite direction to bring the value of z down.

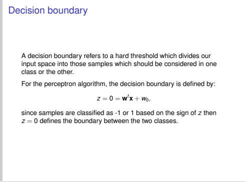
Decision boundary

A decision boundary refers to a hard threshold which divides our input space into those samples which should be considered in one class or the other.

For the perceptron algorithm, the decision boundary is defined by:

$$z = 0 = \mathbf{w}^t\mathbf{x} + w_0,$$

since samples are classified as -1 or 1 based on the sign of z then $z = 0$ defines the boundary between the two classes.


Decision boundary

A decision boundary refers to a hard threshold which divides our input space into those samples which should be considered in one class or the other.

For the perceptron algorithm, the decision boundary is defined by:

$$z = 0 = \mathbf{w}^t\mathbf{x} + w_0,$$

since samples are classified as -1 or 1 based on the sign of z then $z = 0$ defines the boundary between the two classes.

As previously noted, a decision boundary refers to a hard threshold which divides our input space into those samples which should be considered in one class or the other.
So once the perceptron is trained, we can define the shape of that decision boundary by setting the latent variable z to zero and determining which set of inputs $\mathbf{x}$ lie on one side of the boundary or the other.
In multiple dimensions, the input space would be divided in two by a simple plane described by these equations.

Decision boundary

For two inputs:

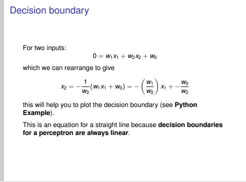
$$0 = w_1 x_1 + w_2 x_2 + w_0$$

which we can rearrange to give

$$x_2 = -\frac{1}{w_2}(w_1 x_1 + w_0) = -\left(\frac{w_1}{w_2}\right) x_1 + -\frac{w_0}{w_2}$$

this will help you to plot the decision boundary (see **Python Example**).

This is an equation for a straight line because **decision boundaries for a perceptron are always linear**.


Decision boundary

For two inputs:

$$0 = w_1 x_1 + w_2 x_2 + w_0$$

which we can rearrange to give

$$x_2 = -\frac{1}{w_2}(w_1 x_1 + w_0) = -\left(\frac{w_1}{w_2}\right) x_1 + -\frac{w_0}{w_2}$$

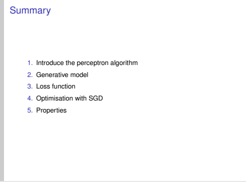
this will help you to plot the decision boundary (see **Python Example**).

This is an equation for a straight line because **decision boundaries for a perceptron are always linear**.

In 2 or 3 dimensions this can be visualised by solving for one set of x's in terms of the others but this is harder in more dimensions.
For two inputs, we can write x_2 in terms of x_1 and you will use this in the associated Python Example to plot decision boundaries.
Note that the equation for the decision boundary is of the form of a straight line.
This is because decision boundaries for the perceptrons are always linear.

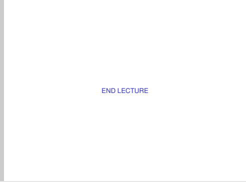
Summary

1. Introduce the perceptron algorithm
2. Generative model
3. Loss function
4. Optimisation with SGD
5. Properties



We have considered the derivation of the perceptron algorithm for binary classification.
I have told you the underlying data generating process assumed by the perceptron method and how we can then derive an appropriate loss function from it.
Classically, the training of the perceptron is done using online learning, so stochastic gradient descent using one sample at a time.
This leads to very simple updates for the algorithm and I have provided you with some intuition about how those updates work.

END LECTURE


END LECTURE