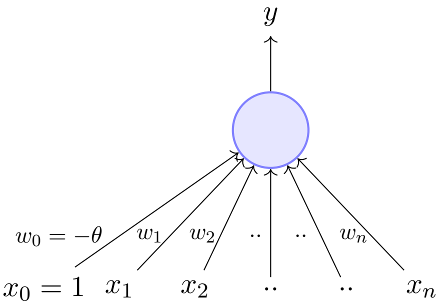
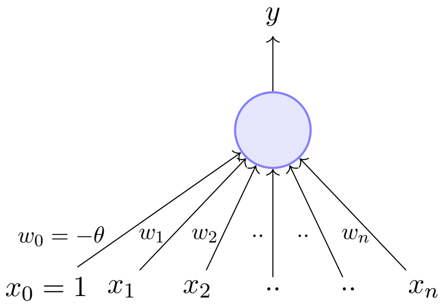
Module 3.2: A typical Supervised Machine Learning Setup

Sigmoid (logistic) Neuron



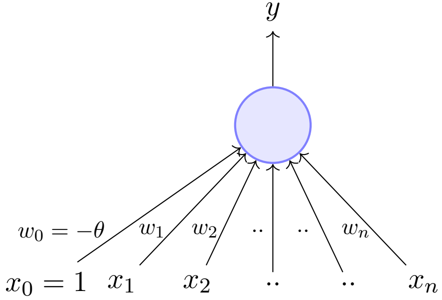
- What next ?

Sigmoid (logistic) Neuron

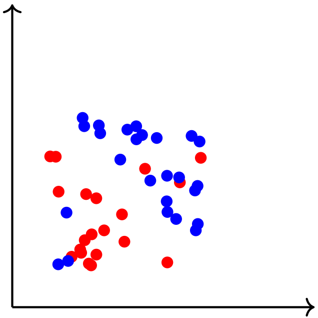


- What next ?
- Well, just as we had an algorithm for learning the weights of a perceptron, we also need a way of learning the weights of a sigmoid neuron

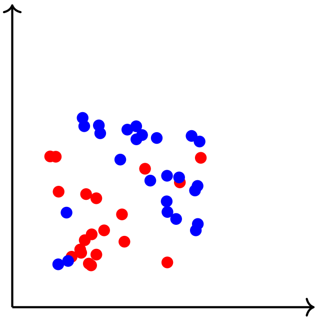
Sigmoid (logistic) Neuron



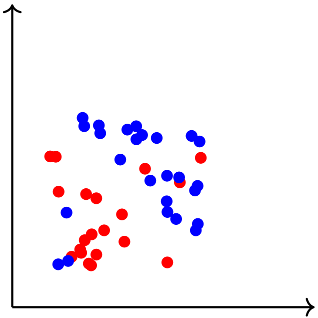
- What next ?
- Well, just as we had an algorithm for learning the weights of a perceptron, we also need a way of learning the weights of a sigmoid neuron
- Before we see such an algorithm we will revisit the concept of **error**



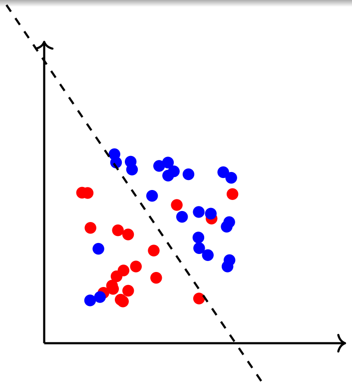
- Earlier we mentioned that a single perceptron cannot deal with this data because it is not linearly separable



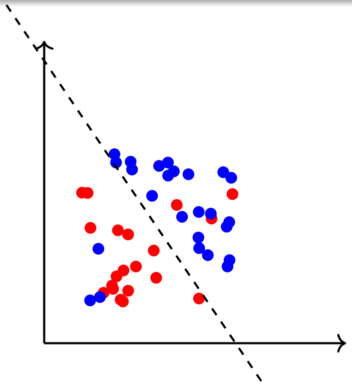
- Earlier we mentioned that a single perceptron cannot deal with this data because it is not linearly separable
- What does "cannot deal with" mean?



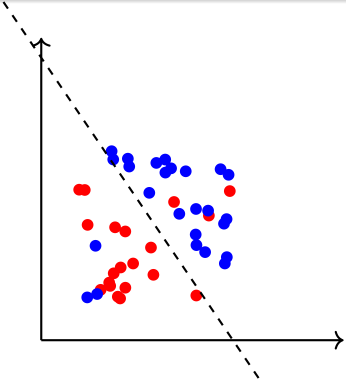
- Earlier we mentioned that a single perceptron cannot deal with this data because it is not linearly separable
- What does "cannot deal with" mean?
- What would happen if we use a perceptron model to classify this data ?



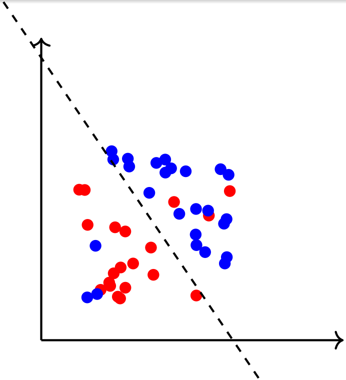
- Earlier we mentioned that a single perceptron cannot deal with this data because it is not linearly separable
- What does "cannot deal with" mean?
- What would happen if we use a perceptron model to classify this data ?
- We would probably end up with a line like this ...



- Earlier we mentioned that a single perceptron cannot deal with this data because it is not linearly separable
- What does "cannot deal with" mean?
- What would happen if we use a perceptron model to classify this data ?
- We would probably end up with a line like this ...
- This line doesn't seem to be too bad



- Earlier we mentioned that a single perceptron cannot deal with this data because it is not linearly separable
- What does "cannot deal with" mean?
- What would happen if we use a perceptron model to classify this data ?
- We would probably end up with a line like this ...
- This line doesn't seem to be too bad
- Sure, it misclassifies 3 blue points and 3 red points but we could live with this error in **most** real world applications



- Earlier we mentioned that a single perceptron cannot deal with this data because it is not linearly separable
- What does "cannot deal with" mean?
- What would happen if we use a perceptron model to classify this data ?
- We would probably end up with a line like this ...
- This line doesn't seem to be too bad
- Sure, it misclassifies 3 blue points and 3 red points but we could live with this error in **most** real world applications
- From now on, we will accept that it is hard to drive the error to 0 in most cases and will instead aim to reach the minimum possible error

This brings us to a typical machine learning setup which has the following components...

This brings us to a typical machine learning setup which has the following components...

- **Data:** $\{x_i, y_i\}_{i=1}^{n}$

This brings us to a typical machine learning setup which has the following components...

- **Data:** $\{x_i, y_i\}_{i=1}^n$
- **Model:** Our approximation of the relation between x and y. For example,

This brings us to a typical machine learning setup which has the following components...

- **Data:** $\{x_i, y_i\}_{i=1}^{n}$
- **Model:** Our approximation of the relation between x and y. For example,

$$\hat{y} = \frac{1}{1 + e^{-(w^T x)}}$$

This brings us to a typical machine learning setup which has the following components...

- **Data:** $\{x_i, y_i\}_{i=1}^{n}$
- **Model:** Our approximation of the relation between x and y. For example,

$$\hat{y} = \frac{1}{1 + e^{-(w^T x)}}$$

$$or \quad \hat{y} = w^T x$$

This brings us to a typical machine learning setup which has the following components...

- **Data:** $\{x_i, y_i\}_{i=1}^{n}$
- **Model:** Our approximation of the relation between x and y. For example,

$$\hat{y} = \frac{1}{1 + e^{-(w^T x)}}$$

$$or \quad \hat{y} = w^T x$$

$$or \quad \hat{y} = \mathbf{x^T w x}$$

This brings us to a typical machine learning setup which has the following components...

- **Data:** $\{x_i, y_i\}_{i=1}^{n}$
- **Model:** Our approximation of the relation between x and y. For example,

$$\hat{y} = \frac{1}{1 + e^{-(w^T x)}}$$

$$or \quad \hat{y} = w^T x$$

$$or \quad \hat{y} = \mathbf{x^T w x}$$

or just about any function

This brings us to a typical machine learning setup which has the following components...

- **Data:** $\{x_i, y_i\}_{i=1}^{n}$
- **Model:** Our approximation of the relation between x and y. For example,

$$\hat{y} = \frac{1}{1 + e^{-(w^T x)}}$$

$$or \quad \hat{y} = w^T x$$

$$or \quad \hat{y} = \mathbf{x^T w x}$$

  or just about any function

- **Parameters:** In all the above cases, w is a parameter which needs to be learned from the data

This brings us to a typical machine learning setup which has the following components...

- **Data:** $\{x_i, y_i\}_{i=1}^{n}$
- **Model:** Our approximation of the relation between x and y. For example,

$$\hat{y} = \frac{1}{1 + e^{-(w^T x)}}$$

$$or \quad \hat{y} = w^T x$$

$$or \quad \hat{y} = \mathbf{x^T w x}$$

  or just about any function

- **Parameters:** In all the above cases, w is a parameter which needs to be learned from the data
- **Learning algorithm:** An algorithm for learning the parameters (w) of the model (for example, perceptron learning algorithm, gradient descent, etc.)

This brings us to a typical machine learning setup which has the following components...

- **Data:** $\{x_i, y_i\}_{i=1}^n$
- **Model:** Our approximation of the relation between x and y. For example,

$$\hat{y} = \frac{1}{1 + e^{-(w^T x)}}$$
$$or \quad \hat{y} = w^T x$$
$$or \quad \hat{y} = \mathbf{x^T w x}$$

  or just about any function
- **Parameters:** In all the above cases, w is a parameter which needs to be learned from the data
- **Learning algorithm:** An algorithm for learning the parameters (w) of the model (for example, perceptron learning algorithm, gradient descent, etc.)
- **Objective/Loss/Error function:** To guide the learning algorithm

This brings us to a typical machine learning setup which has the following components...

- **Data:** $\{x_i, y_i\}_{i=1}^n$
- **Model:** Our approximation of the relation between x and y. For example,

$$\hat{y} = \frac{1}{1 + e^{-(w^T x)}}$$
$$or \quad \hat{y} = w^T x$$
$$or \quad \hat{y} = \mathbf{x^T w x}$$

  or just about any function
- **Parameters:** In all the above cases, w is a parameter which needs to be learned from the data
- **Learning algorithm:** An algorithm for learning the parameters (w) of the model (for example, perceptron learning algorithm, gradient descent, etc.)
- **Objective/Loss/Error function:** To guide the learning algorithm - the learning algorithm should aim to minimize the loss function

As an illustration, consider our movie example

As an illustration, consider our movie example

- **Data:** $\{x_i = movie, y_i = like/dislike\}_{i=1}^{n}$

As an illustration, consider our movie example

- **Data:** $\{x_i = movie, y_i = like/dislike\}_{i=1}^{n}$
- **Model:** Our approximation of the relation between x and y (the probability of liking a movie).

As an illustration, consider our movie example

- **Data:** $\{x_i = movie, y_i = like/dislike\}_{i=1}^{n}$
- **Model:** Our approximation of the relation between x and y (the probability of liking a movie).

$$\hat{y} = \frac{1}{1 + e^{-(w^T x)}}$$

As an illustration, consider our movie example

- **Data:** $\{x_i = movie, y_i = like/dislike\}_{i=1}^{n}$
- **Model:** Our approximation of the relation between x and y (the probability of liking a movie).

$$\hat{y} = \frac{1}{1 + e^{-(w^T x)}}$$

- **Parameter:** w

As an illustration, consider our movie example

- **Data:** $\{x_i = movie, y_i = like/dislike\}_{i=1}^{n}$
- **Model:** Our approximation of the relation between x and y (the probability of liking a movie).

$$\hat{y} = \frac{1}{1 + e^{-(w^T x)}}$$

- **Parameter:** w
- **Learning algorithm:** Gradient Descent [we will see soon]

As an illustration, consider our movie example

- **Data:** $\{x_i = movie, y_i = like/dislike\}_{i=1}^{n}$
- **Model:** Our approximation of the relation between x and y (the probability of liking a movie).

$$\hat{y} = \frac{1}{1 + e^{-(w^T x)}}$$

- **Parameter:** w
- **Learning algorithm:** Gradient Descent [we will see soon]
- **Objective/Loss/Error function:**

As an illustration, consider our movie example

- **Data:** $\{x_i = movie, y_i = like/dislike\}_{i=1}^{n}$
- **Model:** Our approximation of the relation between x and y (the probability of liking a movie).

$$\hat{y} = \frac{1}{1 + e^{-(w^T x)}}$$

- **Parameter:** w
- **Learning algorithm:** Gradient Descent [we will see soon]
- **Objective/Loss/Error function:** One possibility is

$$\mathscr{L}(w) = \sum_{i=1}^{n} (\hat{y}_i - y_i)^2$$

As an illustration, consider our movie example

- **Data:** $\{x_i = movie, y_i = like/dislike\}_{i=1}^{n}$
- **Model:** Our approximation of the relation between x and y (the probability of liking a movie).

$$\hat{y} = \frac{1}{1 + e^{-(w^T x)}}$$

- **Parameter:** w
- **Learning algorithm:** Gradient Descent [we will see soon]
- **Objective/Loss/Error function:** One possibility is

$$\mathscr{L}(w) = \sum_{i=1}^{n} (\hat{y}_i - y_i)^2$$

The learning algorithm should aim to find a w which minimizes the above function (squared error between y and $\hat{y}$)