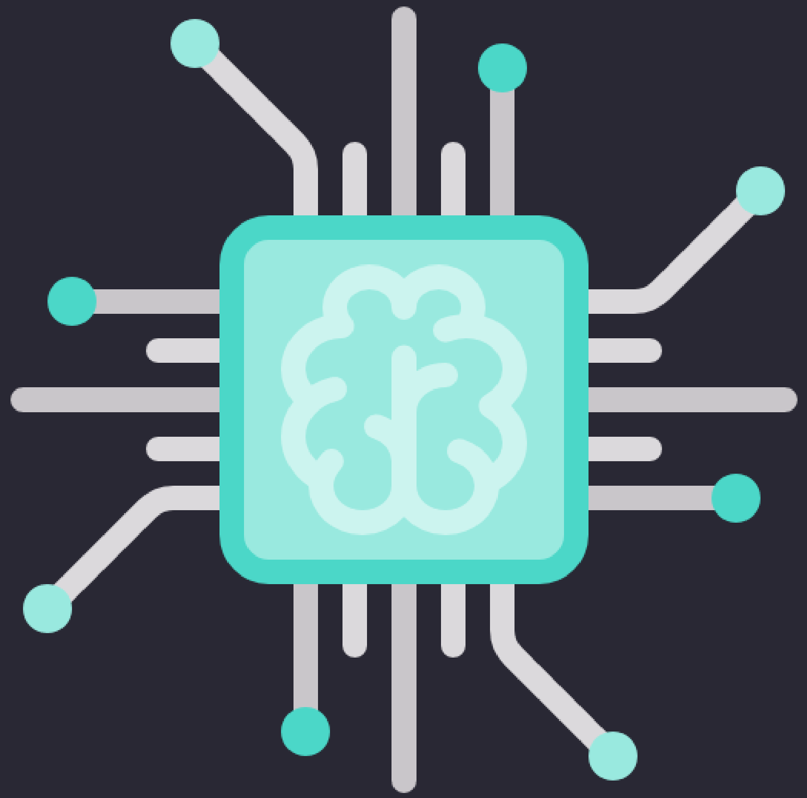
Deep Dissection

Understanding the ins and outs of Neural Networks

by Diogo Pinto



The road ahead

The road ahead

Motivation

- Why deep learning
- Playground

The road ahead

Motivation

- Why deep learning
- Playground

Fundamentals

- Derivatives & Chain rule
- Perceptron
- Sigmoid function & Cross-entropy

The road ahead

Motivation

- Why deep learning
- Playground

Fundamentals

- Derivatives & Chain rule
- Perceptron
- Sigmoid function & Cross-entropy

The Internals

- Forward propagation
- Error computation
- Back propagation
- Parameters update

The road ahead

Motivation

- Why deep learning
- Playground

Fundamentals

- Derivatives & Chain rule
- Perceptron
- Sigmoid function & Cross-entropy

The Internals

- Forward propagation
- Error computation
- Back propagation
- Parameters update

Coffee Break


The road ahead

Motivation

- Why deep learning
- Playground

Fundamentals

- Derivatives & Chain rule
- Perceptron
- Sigmoid function & Cross-entropy

The Internals

- Forward propagation
- Error computation
- Back propagation
- Parameters update

Coffee Break

Implementation

- Binary Classification – Census Income dataset

Who are you people? 👀

Who are you people? 👀

- Who is used to work with **Python**?

Who are you people? 👀

- Who is used to work with **Python**?
- Who already did something **close to this**?

Who are you people? 👀

- Who is used to work with **Python**?

- Who already did something **close to this**?

- Who likes **matrices and letters** in the place of numbers?

Who are you people? 👀

- Who is used to work with **Python**?
- Who already did something **close to this**?
- Who likes **matrices and letters** in the place of numbers?
- Who had a good night of **sleep**?

Assumptions

Assumptions

- A bit of **Algebra** and **Calculus**

Assumptions

- A bit of **Algebra** and **Calculus**
- Introductory level of **Machine Learning** knowledge

Assumptions

- A bit of **Algebra** and **Calculus**
- Introductory level of **Machine Learning** knowledge

Difficulty can be calibrated, give feedback!

Motivation

Why am I here?

"Don't limit your challenges, challenge your limits"

Unknown

The trick is in the representation…

The trick is in the representation…

- Machine learning is traditionally feature engineering intensive

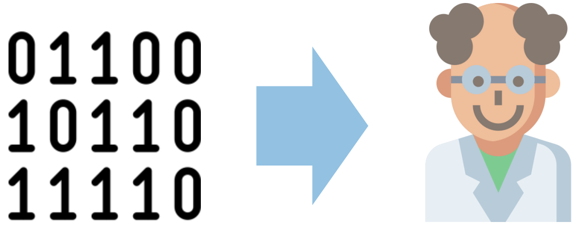
The trick is in the representation…

- Machine learning is traditionally **feature engineering** intensive

01100
10110
11110

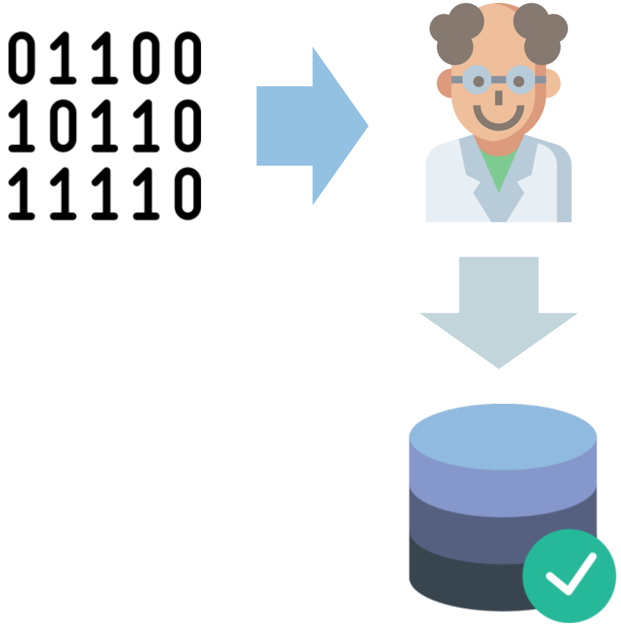
The trick is in the representation…

- Machine learning is traditionally feature engineering intensive



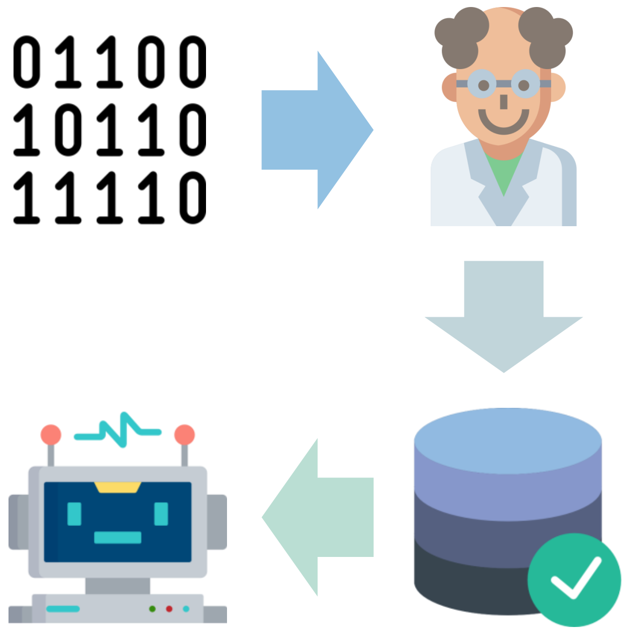
The trick is in the representation…

- Machine learning is traditionally feature engineering intensive



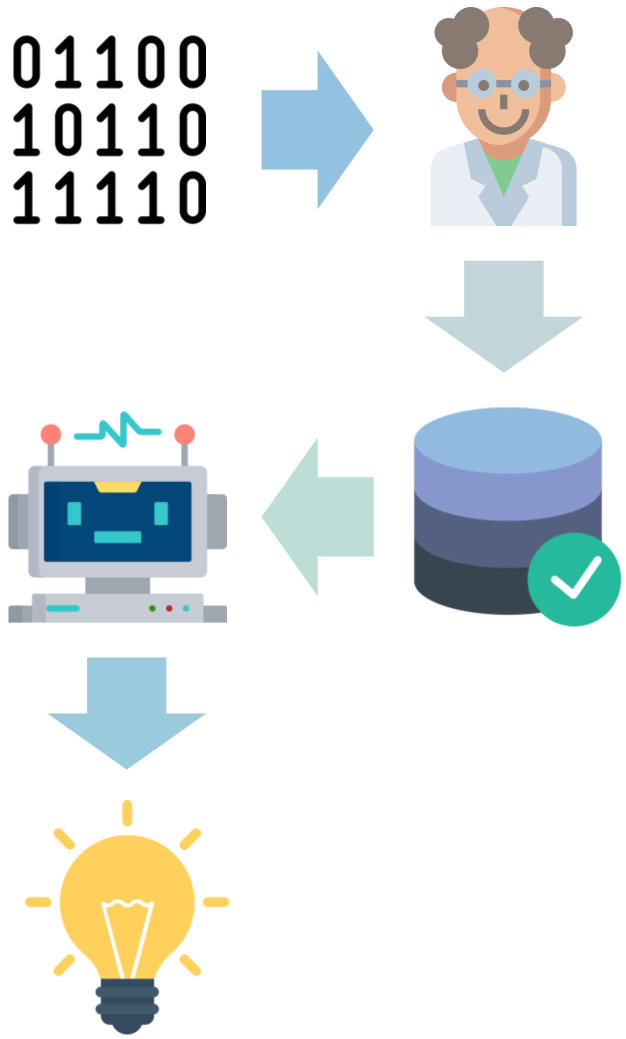
The trick is in the representation…

- Machine learning is traditionally **feature engineering** intensive



The trick is in the representation…

- Machine learning is traditionally **feature engineering** intensive

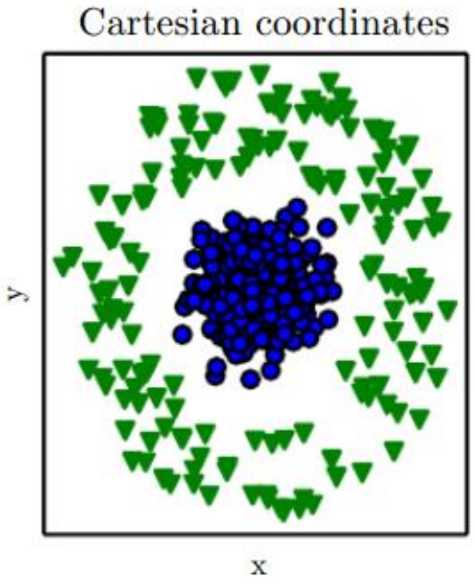


The trick is in the representation…

- Machine learning is traditionally feature engineering intensive
  - Identifying the **sources of influence**

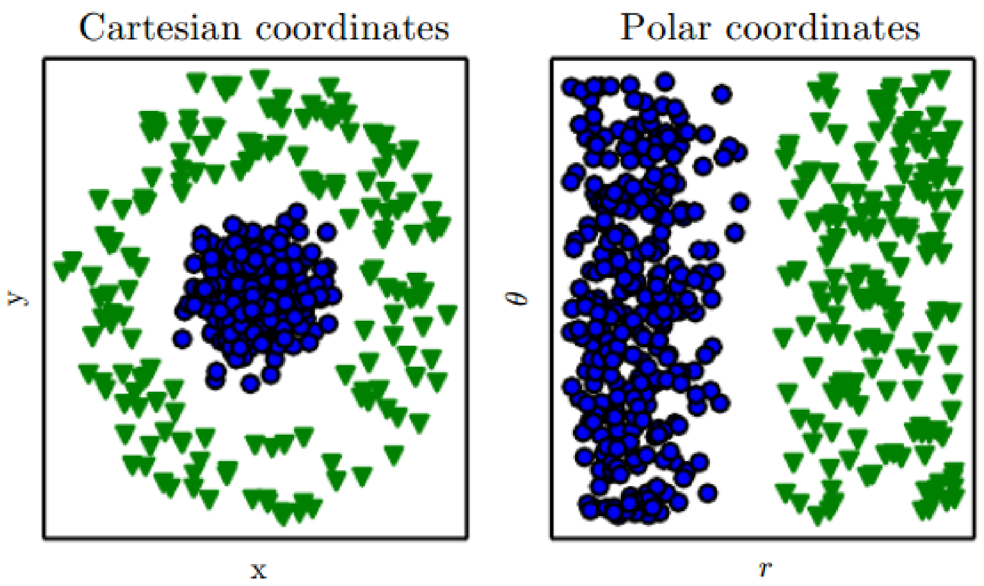
The trick is in the representation…

- Machine learning is traditionally feature engineering intensive
  - Identifying the **sources of influence**



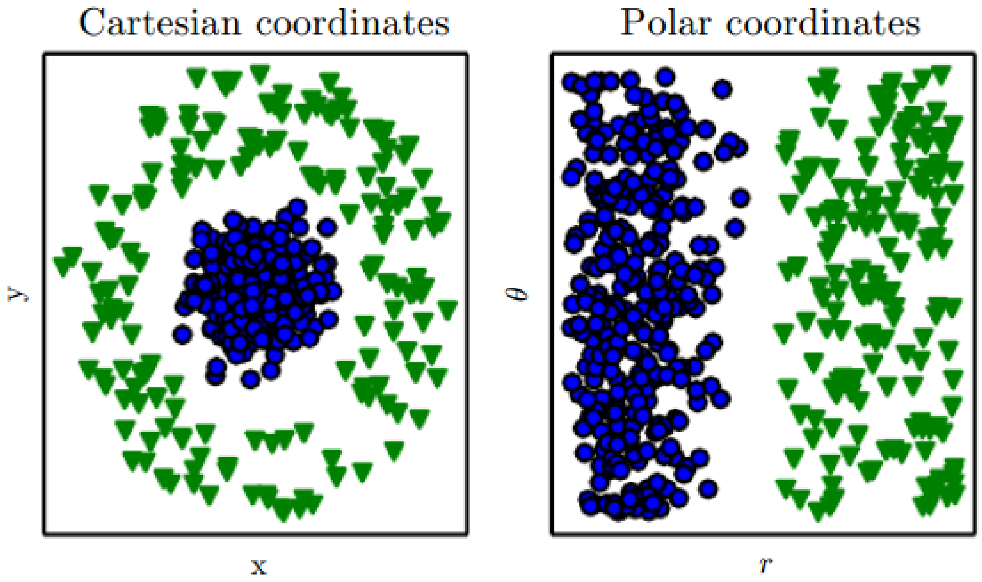
The trick is in the representation…

- Machine learning is traditionally feature engineering intensive
  - Identifying the **sources of influence**



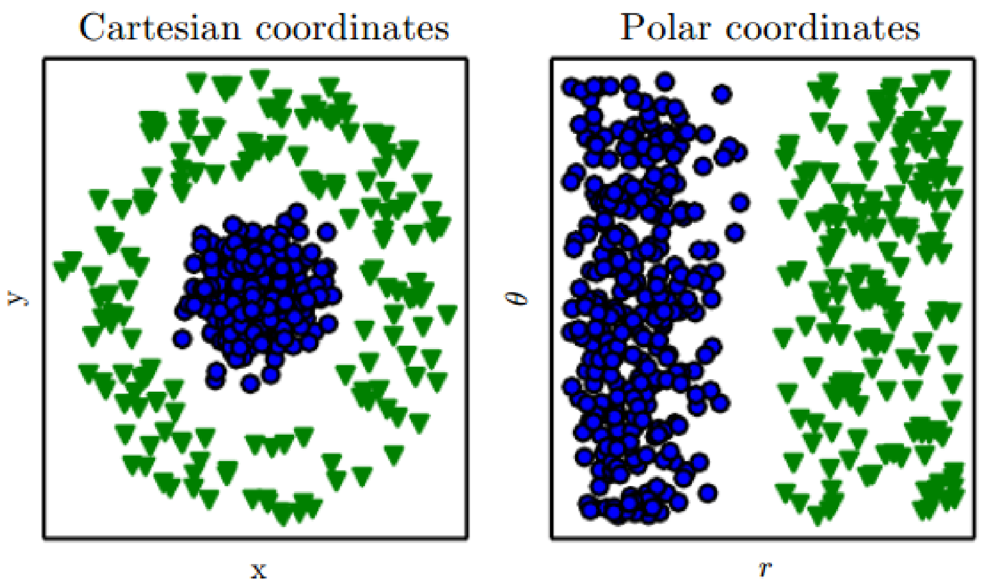
The trick is in the representation…

- Machine learning is traditionally feature engineering intensive
  - Identifying the **sources of influence**
  - Disentangle them



The trick is in the representation…

- Machine learning is traditionally feature engineering intensive
  - Identifying the **sources of influence**
  - Disentangle them
  - Discard op the unnecessary ones

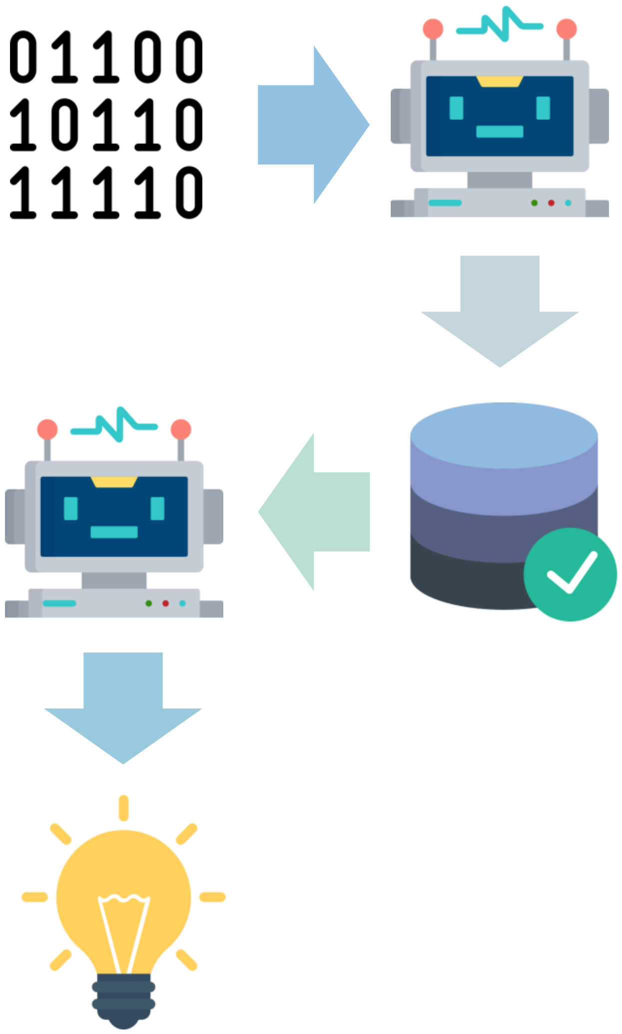


The trick is in the representation…

- Machine learning is traditionally feature engineering intensive
  - Identifying the **sources of influence**
  - Disentangle them
  - Discard op the unnecessary ones
- Representation learning seeks to automate this process

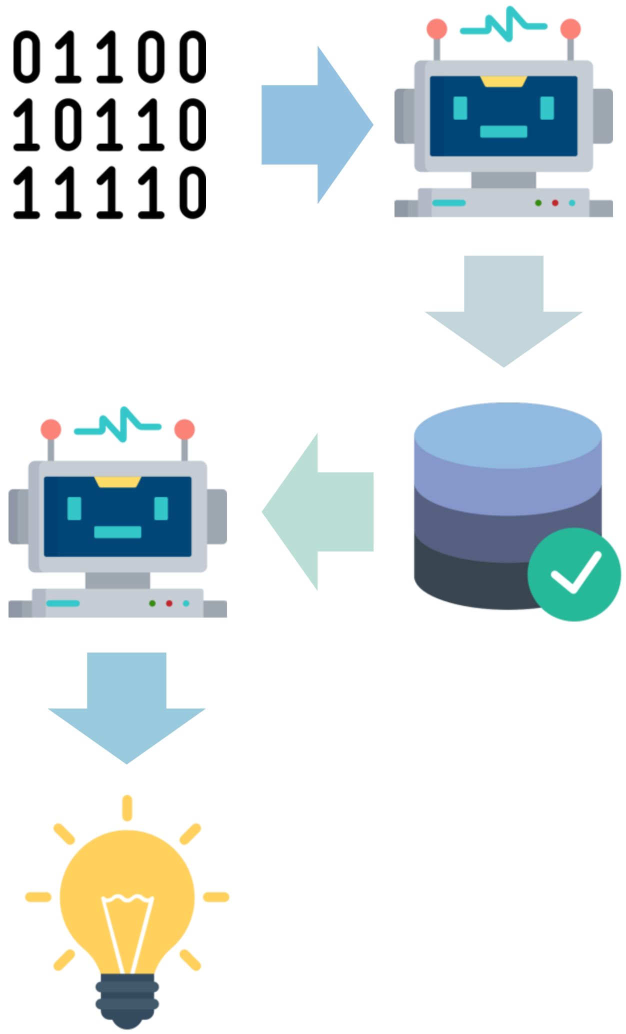
The trick is in the representation…

- Machine learning is traditionally **feature engineering** intensive
  - Identifying the **sources of influence**
  - Disentangle them
  - Discard op the unnecessary ones
- **Representation learning** seeks to automate this process



The trick is in the representation…

- Machine learning is traditionally **feature engineering** intensive
  - Identifying the **sources of influence**
  - Disentangle them
  - Discard op the unnecessary ones
- **Representation learning** seeks to automate this process
  - Learning **high-level concepts** can be as hard as the original problem



The trick is in the representation…

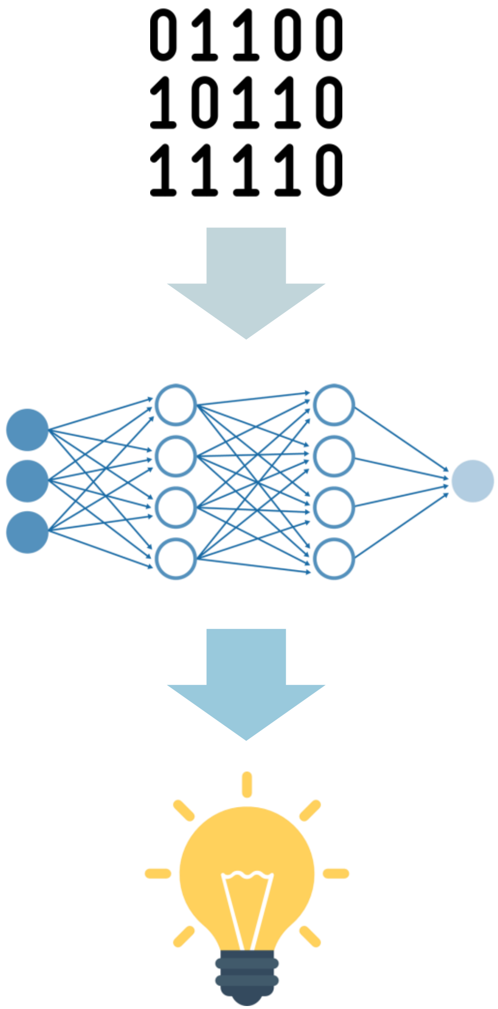
- Machine learning is traditionally feature engineering intensive
  - Identifying the **sources of influence**
  - Disentangle them
  - Discard op the unnecessary ones

- Representation learning seeks to automate this process
  - Learning **high-level concepts** can be as hard as the original problem

- Deep learning tackles this heads on

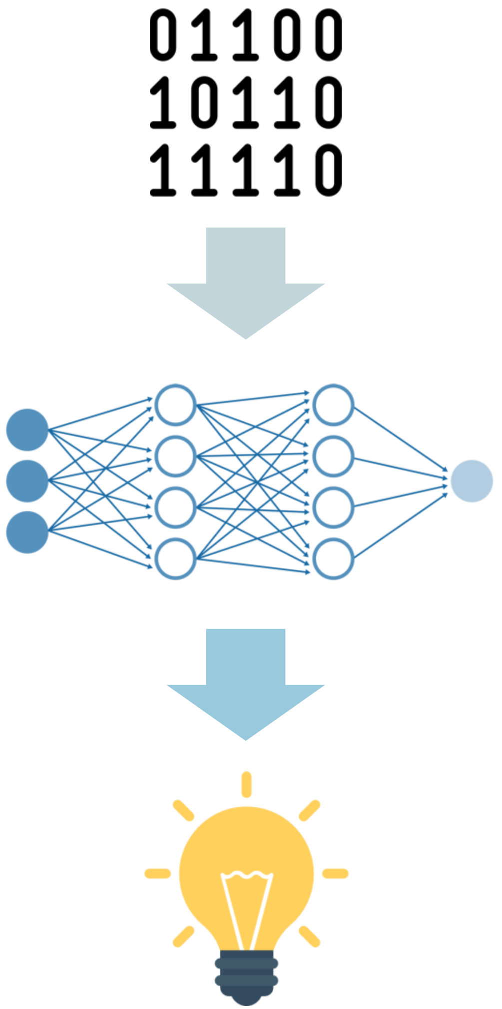
The trick is in the representation…

- Machine learning is traditionally feature engineering intensive
  - Identifying the **sources of influence**
  - Disentangle them
  - Discard op the unnecessary ones
- Representation learning seeks to automate this process
  - Learning **high-level concepts** can be as hard as the original problem
- Deep learning tackles this heads on



The trick is in the representation…

- Machine learning is traditionally feature engineering intensive
  - Identifying the **sources of influence**
  - Disentangle them
  - Discard op the unnecessary ones
- Representation learning seeks to automate this process
  - Learning **high-level concepts** can be as hard as the original problem
- Deep learning tackles this heads on
  - **Hierarchical combination** of simpler concepts into more complex ones



A quick visual demo…

[Neural Networks Playground]

Fundamentals

What do I need to know?

"You can't build a great building on a weak foundation."

Gordon B. Hinckley

It’s Calculus Time 🎉

It’s Calculus Time

- Derivatives

It's Calculus Time

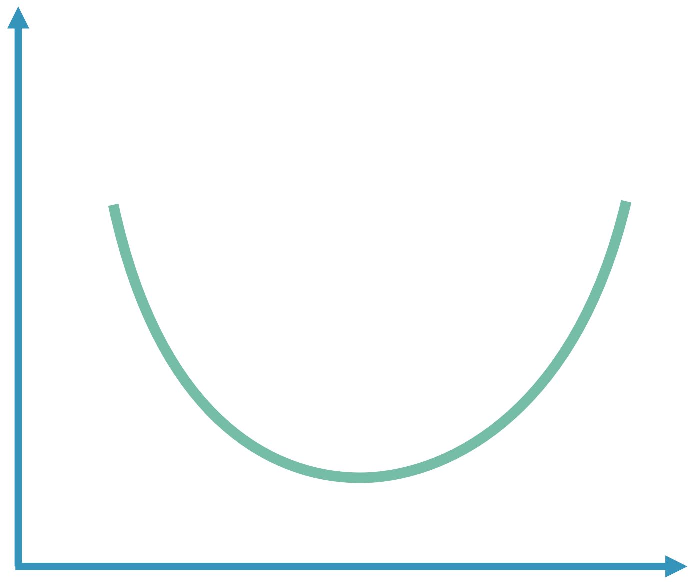
- Derivatives
  - $y = f(x)$

It's Calculus Time

- Derivatives
  - $y = f(x)$
  - How a small change in ***x*** changes ***y***

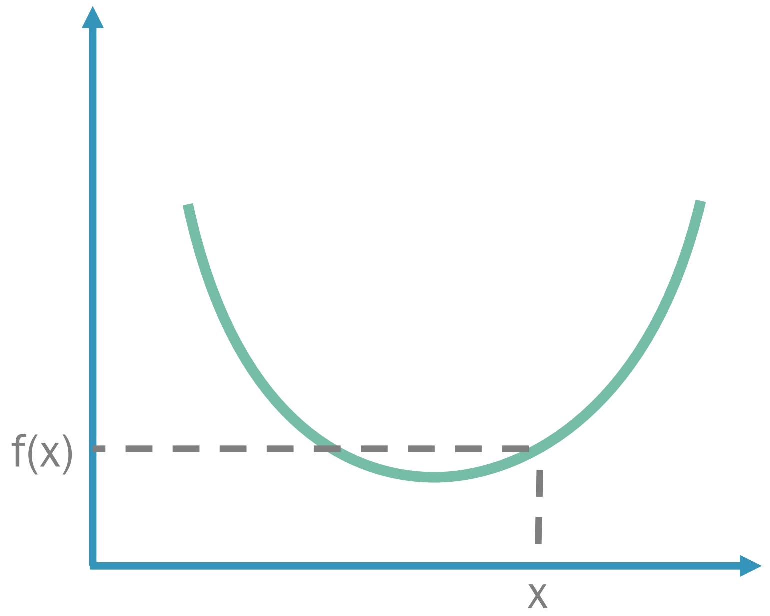
It’s Calculus Time

- Derivatives
  - $y = f(x)$
  - How a small change in ***x*** changes ***y***



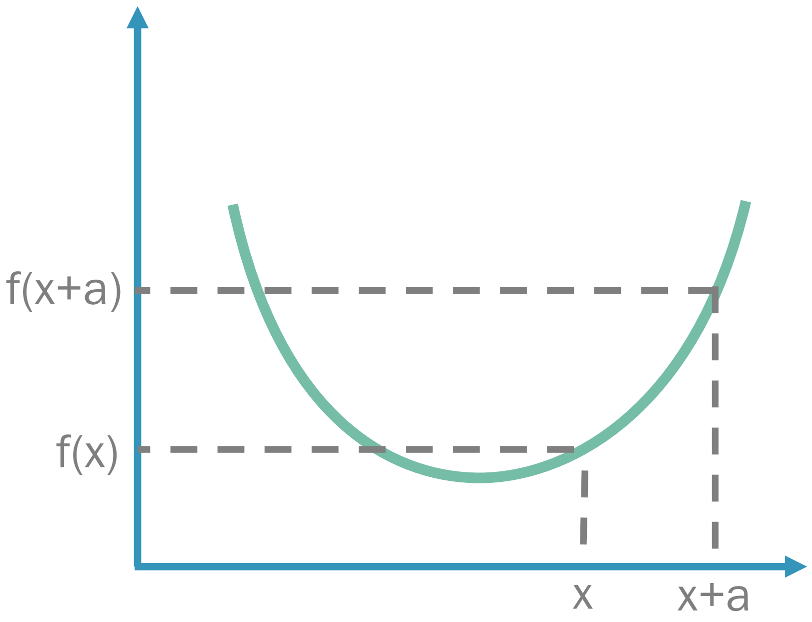
It's Calculus Time 🎉

- Derivatives
  - $y = f(x)$
  - How a small change in ***x*** changes ***y***



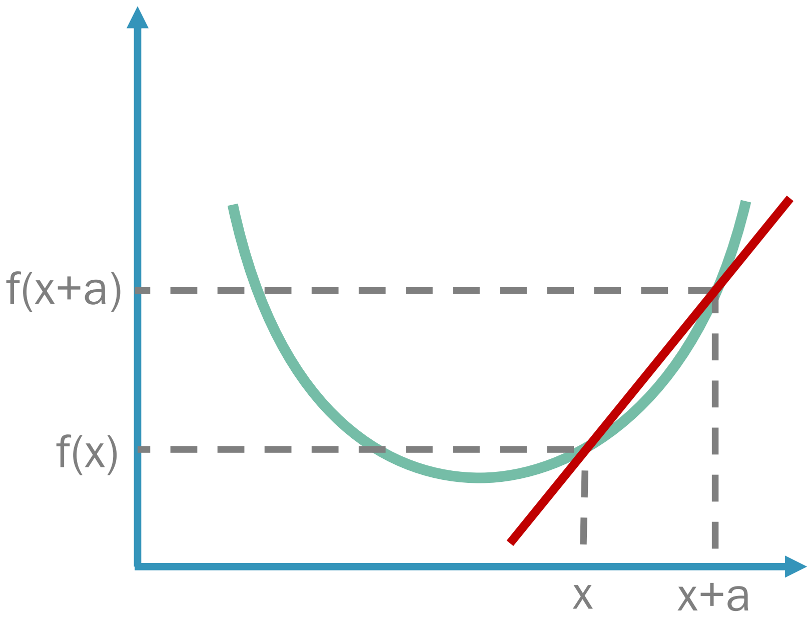
It's Calculus Time

- Derivatives
  - $y = f(x)$
  - How a small change in ***x*** changes ***y***



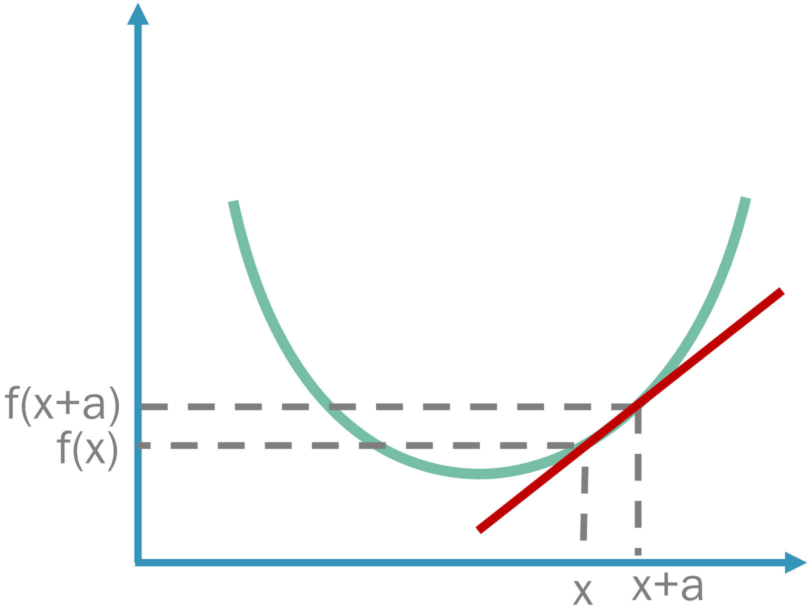
It's Calculus Time 🎉

- Derivatives
  - $y = f(x)$
  - How a small change in ***x*** changes ***y***



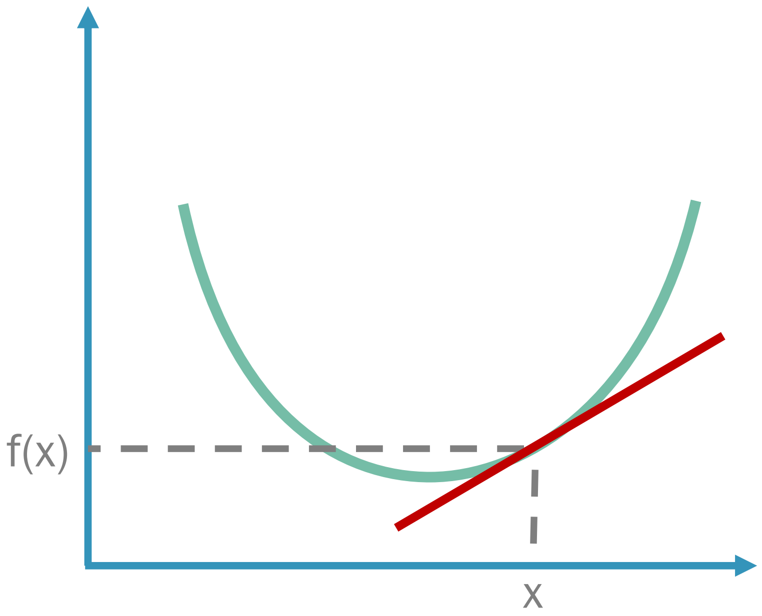
It's Calculus Time 🎉

- Derivatives
  - $y = f(x)$
  - How a small change in *x* changes *y*



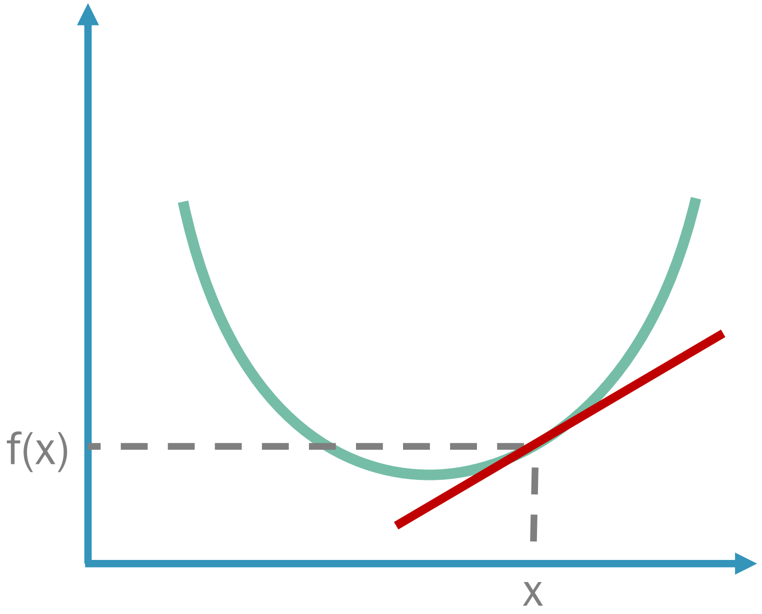
It's Calculus Time

- Derivatives
  - $y = f(x)$
  - How a small change in ***x*** changes ***y***



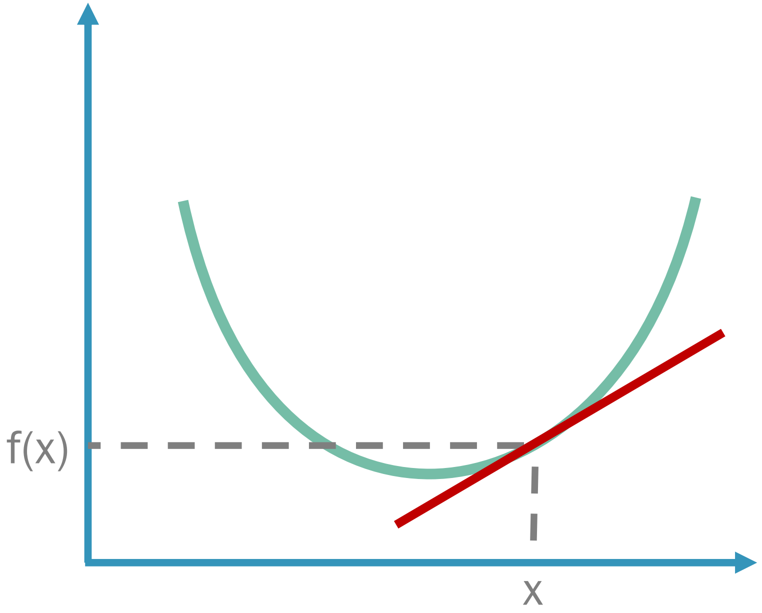
It’s Calculus Time 🎉

- Derivatives
  - $y = f(x)$
  - How a small change in ***x*** changes ***y***
  - $\lim_{a \to 0} \frac{f(x+a)-f(x)}{(x+a)-x}$ or $\frac{dy}{dx}$



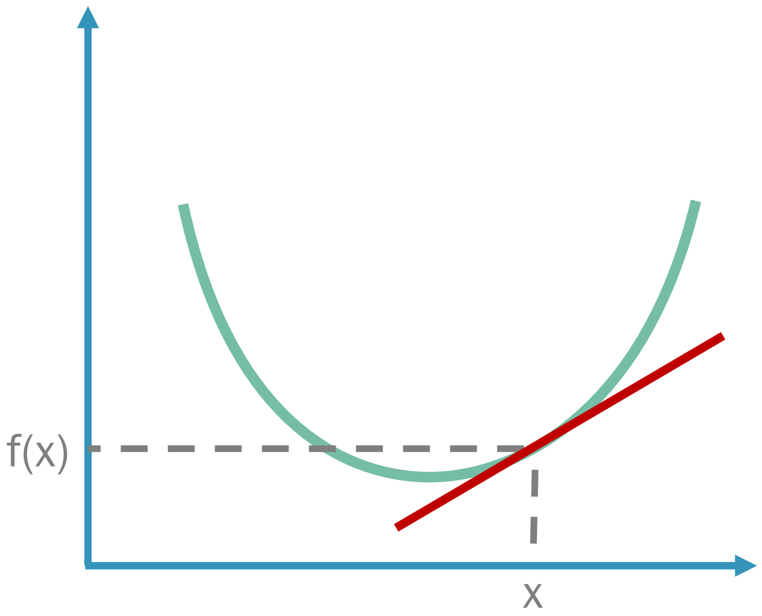
It's Calculus Time

- Derivatives
  - $y = f(x)$
  - How a small change in ***x*** changes ***y***
  - $\lim_{a \to 0} \frac{f(x+a) - f(x)}{a}$ or $\frac{dy}{dx}$



It’s Calculus Time

- Derivatives
  - $y = f(x)$
  - How a small change in ***x*** changes ***y***
  - $\lim_{a \to 0} \frac{f(x+a)-f(x)}{a}$ or $\frac{dy}{dx}$
  - **Slope** of the tangent line

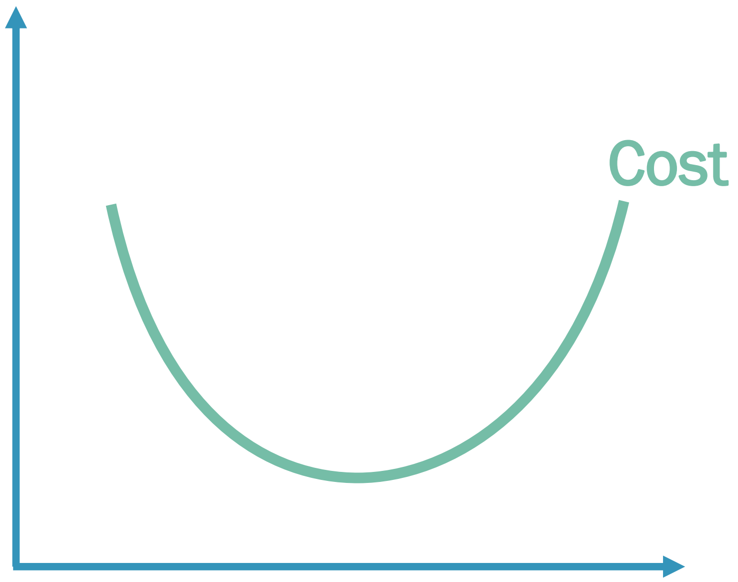


It’s Calculus Time

- Derivatives
  - $y = f(x)$
  - How a small change in ***x*** changes ***y***
  - $\lim_{a \to 0} \frac{f(x+a)-f(x)}{a}$ or $\frac{dy}{dx}$
  - **Slope** of the tangent line
- Gradient descent

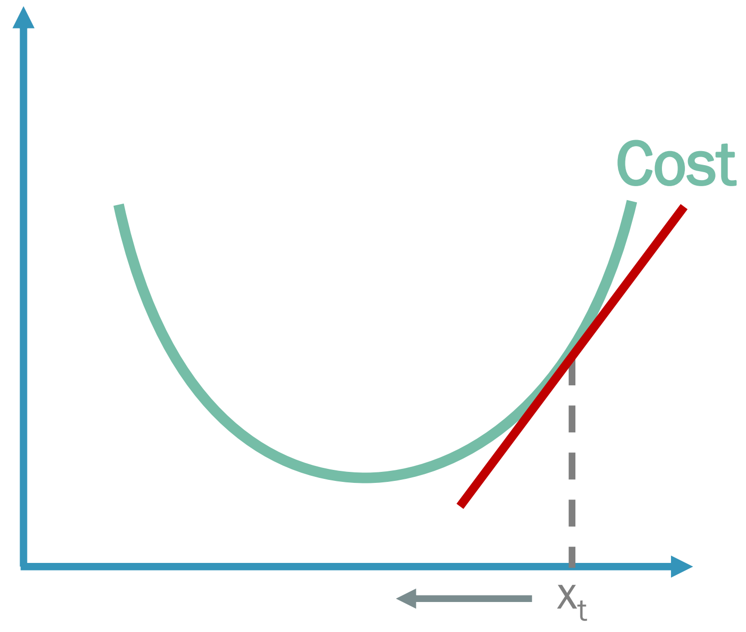
It's Calculus Time 🎉

- Derivatives
  - $y = f(x)$
  - How a small change in ***x*** changes ***y***
  - $\lim_{a \to 0} \frac{f(x+a)-f(x)}{a}$ or $\frac{dy}{dx}$
  - **Slope** of the tangent line
- Gradient descent
  - Minimize a function by **subtracting derivative** at the point



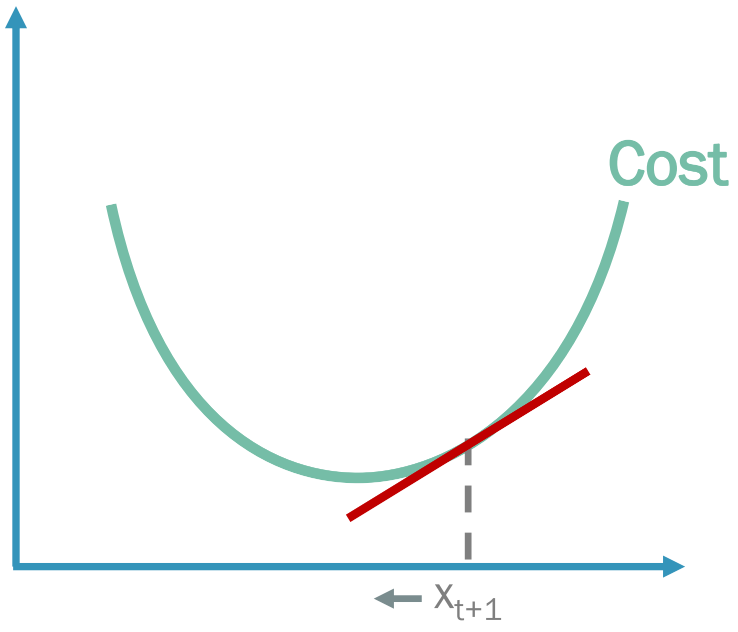
It's Calculus Time

- Derivatives
  - $y = f(x)$
  - How a small change in ***x*** changes ***y***
  - $\lim_{a \to 0} \frac{f(x+a) - f(x)}{a}$ or $\frac{dy}{dx}$
  - **Slope** of the tangent line
- Gradient descent
  - Minimize a function by **subtracting derivative** at the point



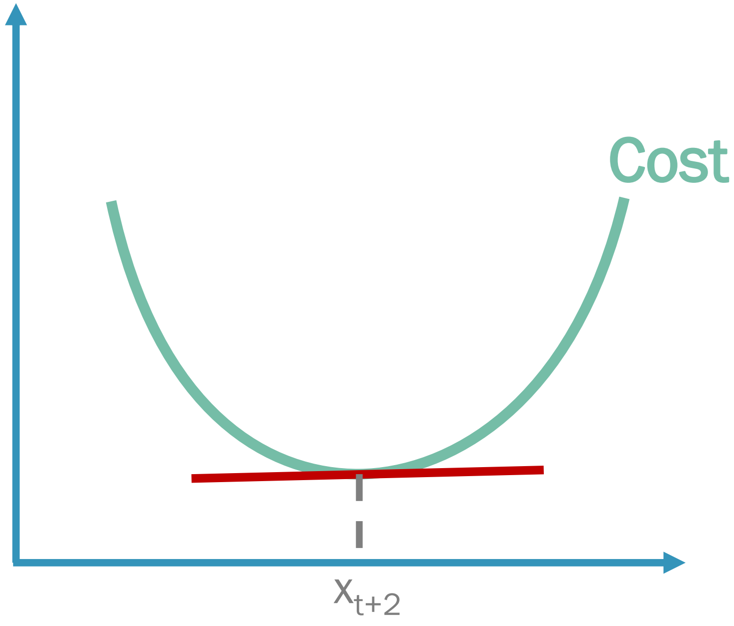
It's Calculus Time 🎉

- Derivatives
  - $y = f(x)$
  - How a small change in ***x*** changes ***y***
  - $\lim_{a \to 0} \frac{f(x+a)-f(x)}{a}$ or $\frac{dy}{dx}$
  - **Slope** of the tangent line
- Gradient descent
  - Minimize a function by **subtracting derivative** at the point



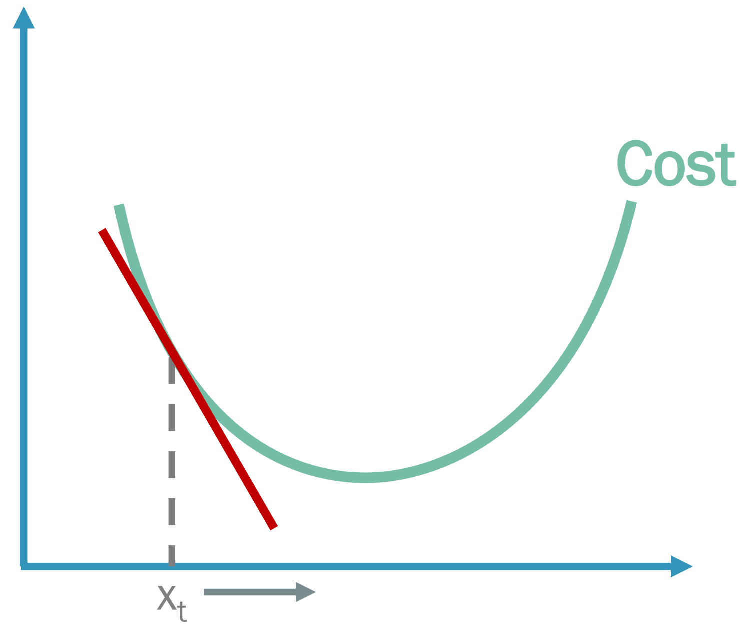
It's Calculus Time

- Derivatives
  - $y = f(x)$
  - How a small change in ***x*** changes ***y***
  - $\lim_{a \to 0} \frac{f(x+a) - f(x)}{a}$ or $\frac{dy}{dx}$
  - **Slope** of the tangent line
- Gradient descent
  - Minimize a function by **subtracting derivative** at the point



It's Calculus Time

- Derivatives
  - $y = f(x)$
  - How a small change in ***x*** changes ***y***
  - $\lim_{a \to 0} \frac{f(x+a) - f(x)}{a}$ or $\frac{dy}{dx}$
  - **Slope** of the tangent line
- Gradient descent
  - Minimize a function by **subtracting derivative** at the point



It's Calculus Time

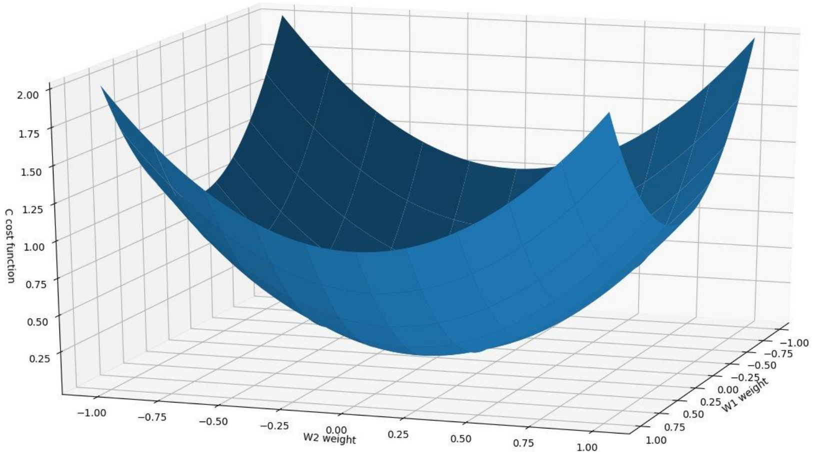
- Derivatives
  - $y = f(x)$
  - How a small change in ***x*** changes ***y***
  - $\lim_{a \to 0} \frac{f(x+a) - f(x)}{a}$ or $\frac{dy}{dx}$
  - **Slope** of the tangent line
- Gradient descent
  - Minimize a function by **subtracting derivative** at the point
- Partial Derivative

It's Calculus Time

- Derivatives
  - $y = f(x)$
  - How a small change in ***x*** changes ***y***
  - $\lim_{a \to 0} \frac{f(x+a)-f(x)}{a}$ or $\frac{dy}{dx}$
  - **Slope** of the tangent line
- Gradient descent
  - Minimize a function by **subtracting derivative** at the point
- Partial Derivative
  - $y = f(x_1, x_2, \dots, x_k)$

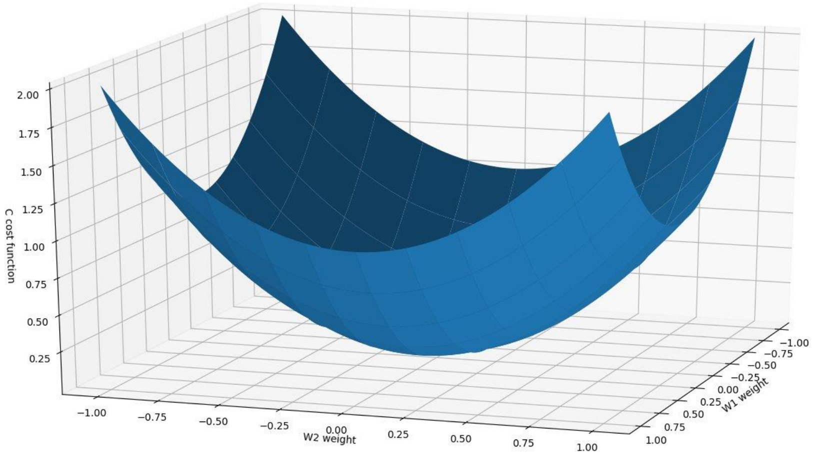
It's Calculus Time

- Derivatives
  - $y = f(x)$
  - How a small change in ***x*** changes ***y***
  - $\lim_{a \to 0} \frac{f(x+a) - f(x)}{a}$ or $\frac{dy}{dx}$
  - **Slope** of the tangent line
- Gradient descent
  - Minimize a function by **subtracting derivative** at the point
- Partial Derivative
  - $y = f(x_1, x_2, \ldots, x_k)$



It’s Calculus Time

- Derivatives
  - $y = f(x)$
  - How a small change in ***x*** changes ***y***
  - $\lim_{a \to 0} \frac{f(x+a)-f(x)}{a}$ or $\frac{dy}{dx}$
  - **Slope** of the tangent line
- Gradient descent
  - Minimize a function by **subtracting derivative** at the point
- Partial Derivative
  - $y = f(x_1, x_2, \dots, x_k)$
  - How a small change in $\boldsymbol{x_i}$ changes **y**, keeping all $\boldsymbol{x_{k \neq i}}$ constant



It's Calculus Time

- Derivatives
  - $y = f(x)$
  - How a small change in ***x*** changes ***y***
  - $\lim_{a \to 0} \frac{f(x+a) - f(x)}{a}$ or $\frac{dy}{dx}$
  - **Slope** of the tangent line
- Gradient descent
  - Minimize a function by **subtracting derivative** at the point
- Partial Derivative
  - $y = f(x_1, x_2, \ldots, x_k)$
  - How a small change in $\boldsymbol{x_i}$ changes **y**, keeping all $\boldsymbol{x_{k \neq i}}$ constant
  - $\frac{\partial y}{\partial x}$ instead of $\frac{dy}{dx}$

The chain rule of derivatives

The chain rule of derivatives

$$\frac{d}{dx} g\big(f(x)\big) = \frac{dg}{df}\frac{df}{dx}$$

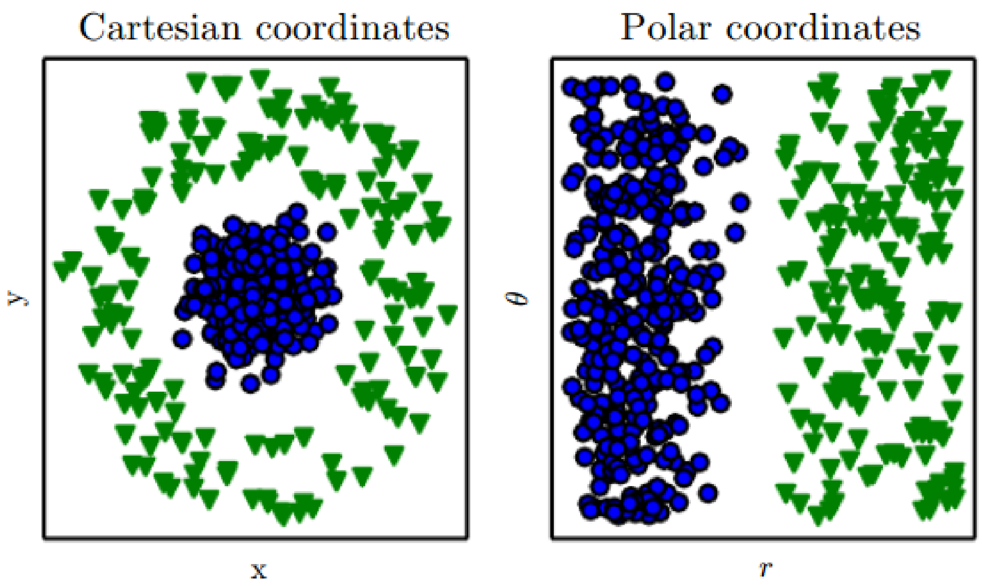
Perceptron

Perceptron

- Linear binary classifier

Perceptron

- Linear binary classifier
  - Linearly separable data



Perceptron

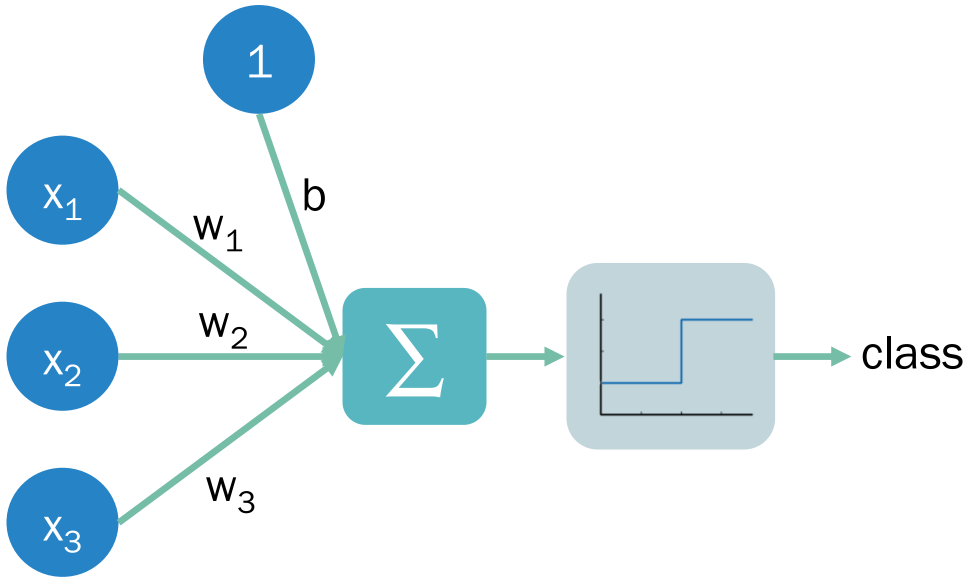
- Linear binary classifier
  - Linearly separable data
  - One out of two classes

Perceptron

- Linear binary classifier
  - Linearly separable data
  - One out of two classes
  - Supervised learning

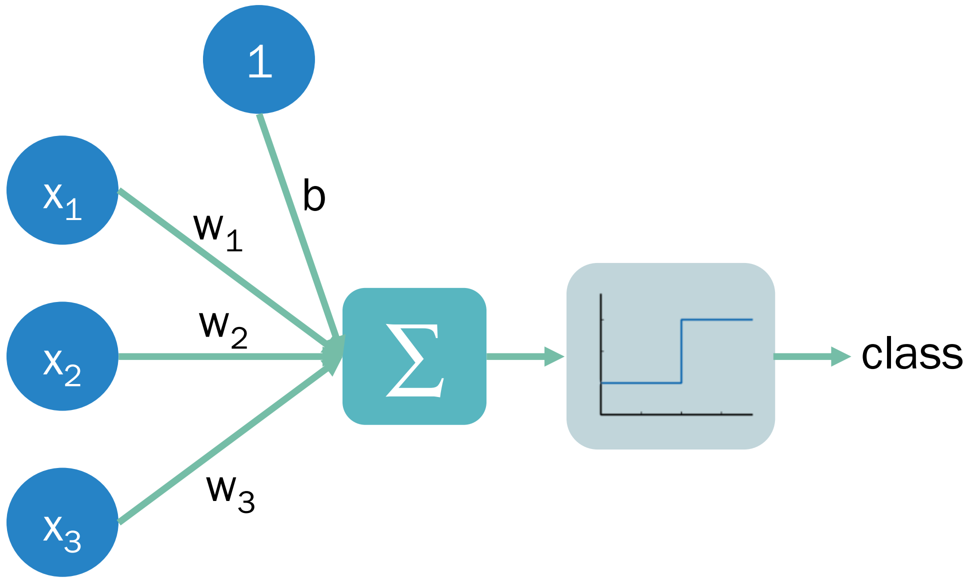
Perceptron

- Linear binary classifier
  - Linearly separable data
  - One out of two classes
  - Supervised learning
- **Components**



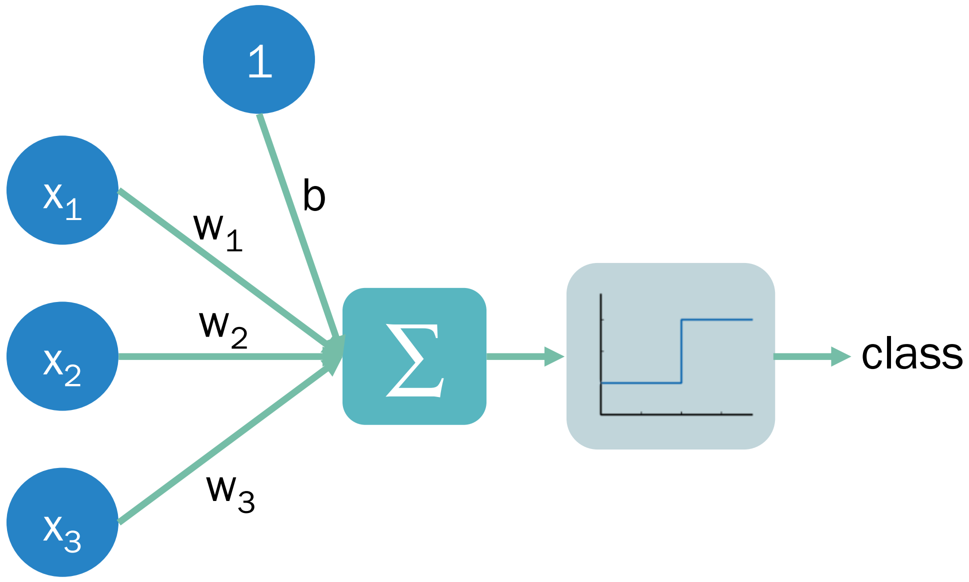
Perceptron

- Linear binary classifier
  - Linearly separable data
  - One out of two classes
  - Supervised learning
- **Components**
  - Input values



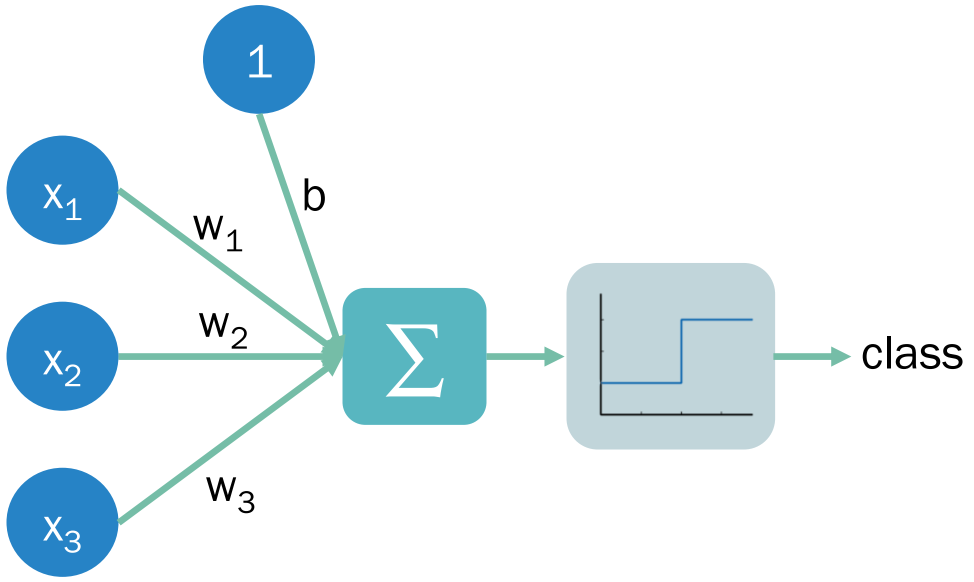
Perceptron

- Linear binary classifier
  - Linearly separable data
  - One out of two classes
  - Supervised learning
- **Components**
  - Input values
  - Weights and Biases



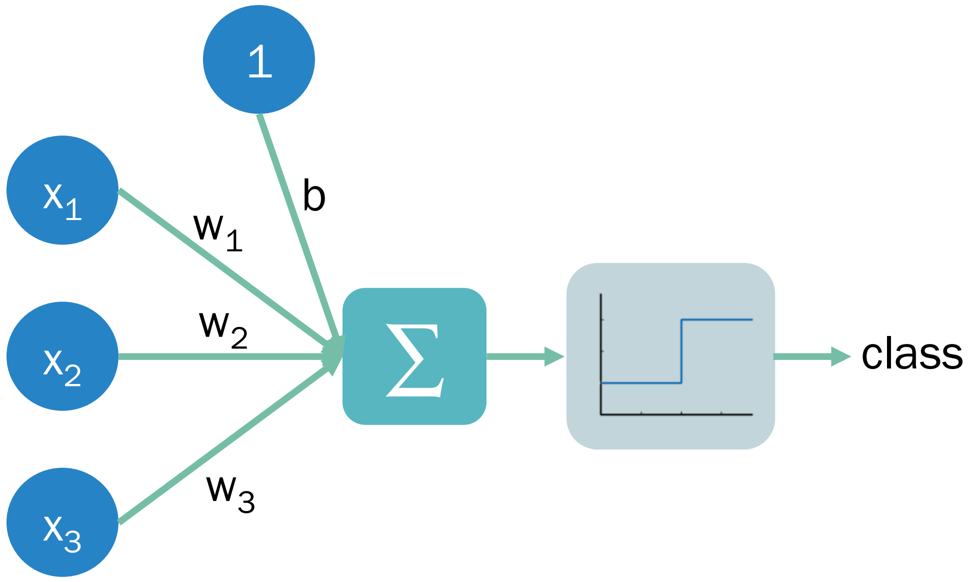
Perceptron

- Linear binary classifier
  - Linearly separable data
  - One out of two classes
  - Supervised learning
- **Components**
  - Input values
  - Weights and Biases
  - Sum



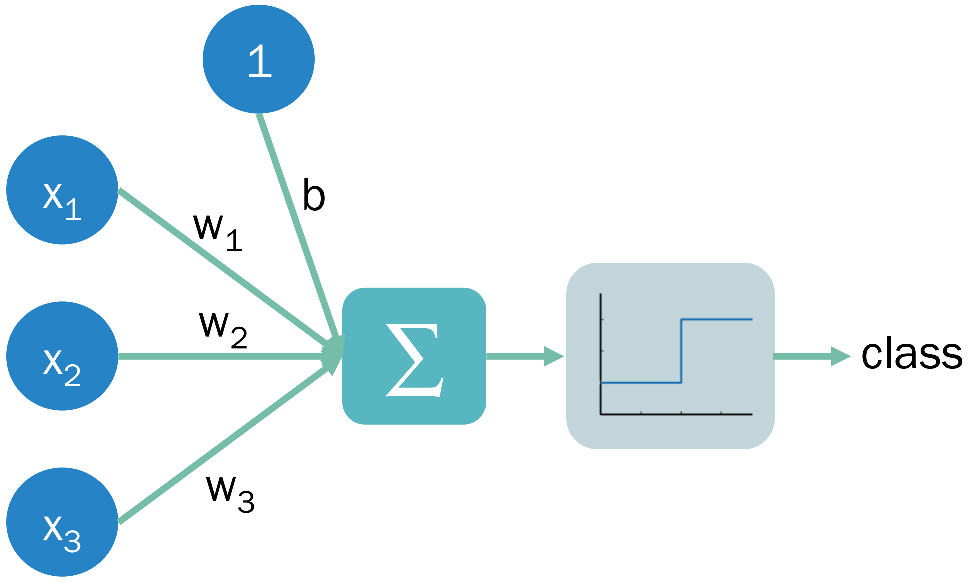
Perceptron

- Linear binary classifier
  - Linearly separable data
  - One out of two classes
  - Supervised learning
- **Components**
  - Input values
  - Weights and Biases
  - Sum
  - Activation function



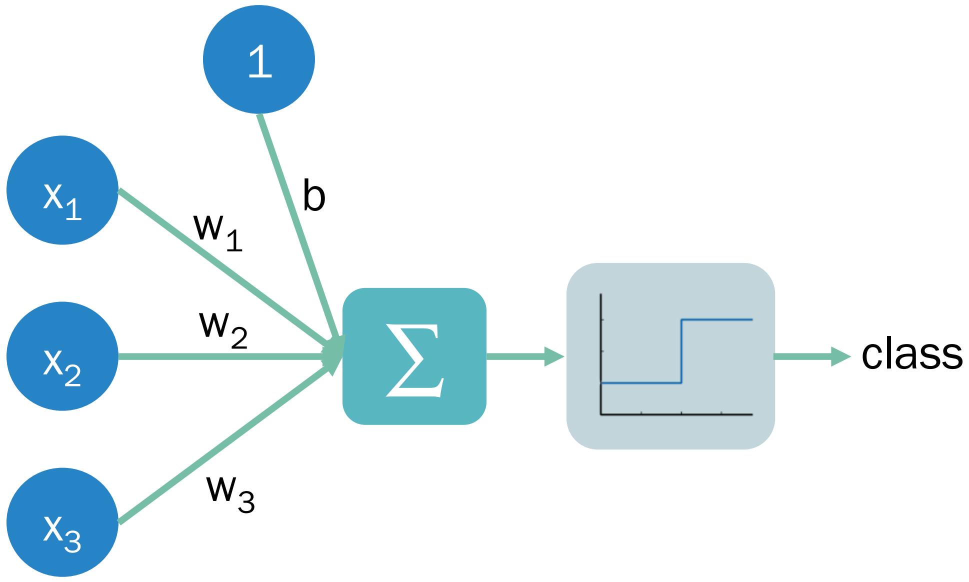
Perceptron

- Linear binary classifier
  - Linearly separable data
  - One out of two classes
  - Supervised learning
- **Components**
  - Input values
  - Weights and Biases
  - Sum
  - Activation function
  - Weight update from errors *(training)*



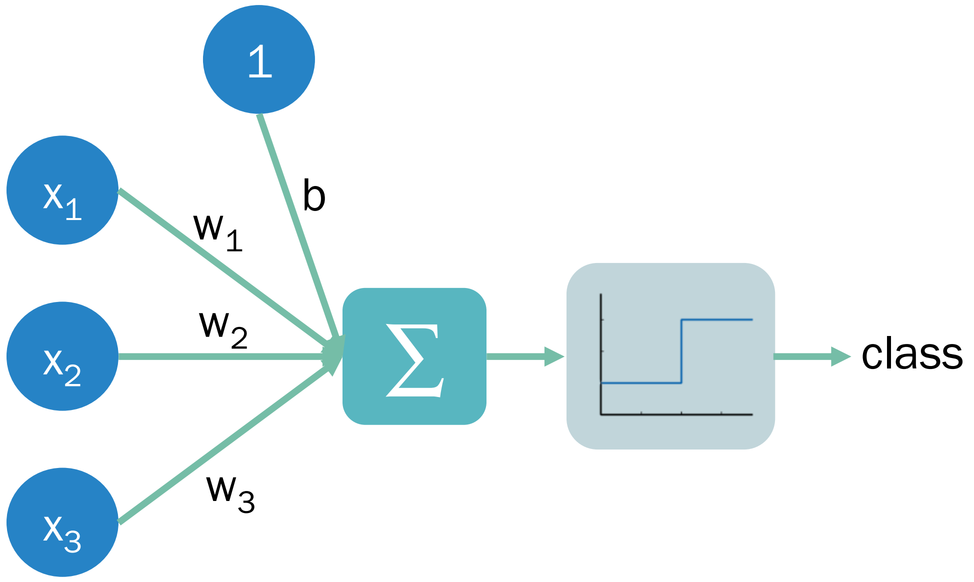
Perceptron

- Linear binary classifier
  - Linearly separable data
  - One out of two classes
  - Supervised learning
- **Components**
  - Input values
  - Weights and Biases
  - Sum
  - Activation function
  - Weight update from errors *(training)*
- **Steps**



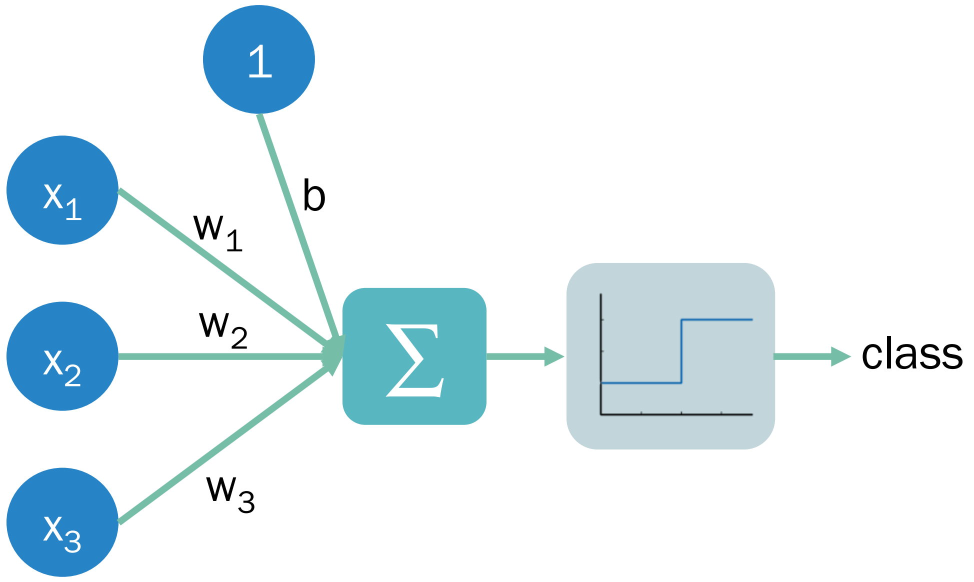
Perceptron

- Linear binary classifier
  - Linearly separable data
  - One out of two classes
  - Supervised learning
- Components
  - Input values
  - Weights and Biases
  - Sum
  - Activation function
  - Weight update from errors *(training)*
- Steps
  1. Multiply inputs by the weights



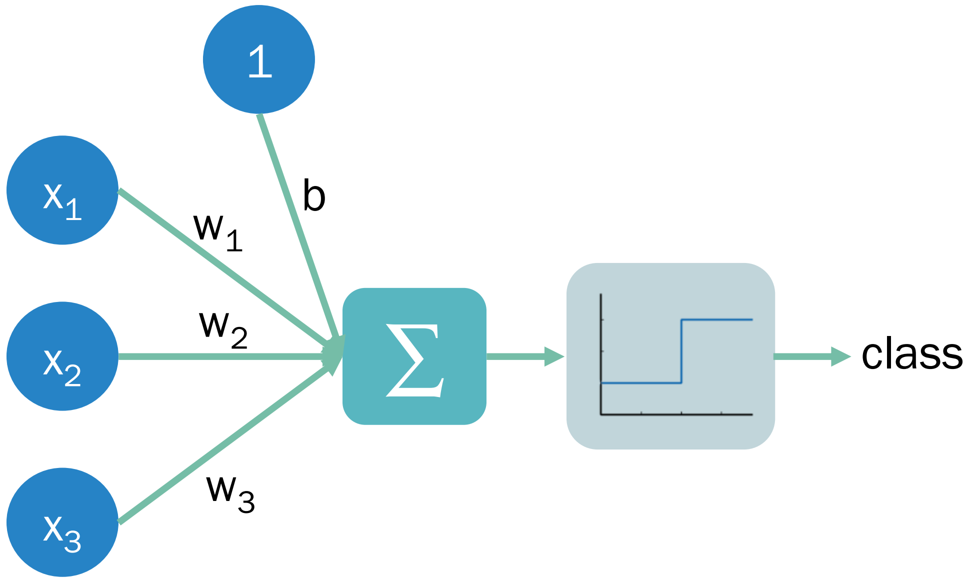
Perceptron

- Linear binary classifier
  - Linearly separable data
  - One out of two classes
  - Supervised learning
- **Components**
  - Input values
  - Weights and Biases
  - Sum
  - Activation function
  - Weight update from errors *(training)*
- **Steps**
  1. Multiply inputs by the weights
  2. Add up the values



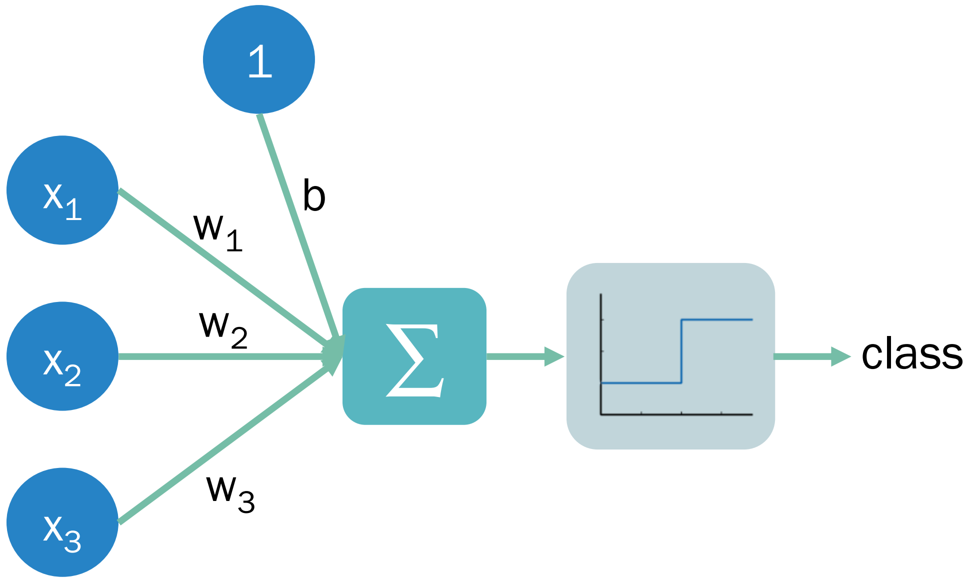
Perceptron

- **Linear binary classifier**
  - Linearly separable data
  - One out of two classes
  - Supervised learning
- **Components**
  - Input values
  - Weights and Biases
  - Sum
  - Activation function
  - Weight update from errors *(training)*
- **Steps**
  1. Multiply inputs by the weights
  2. Add up the values
  3. Apply the activation function



Perceptron

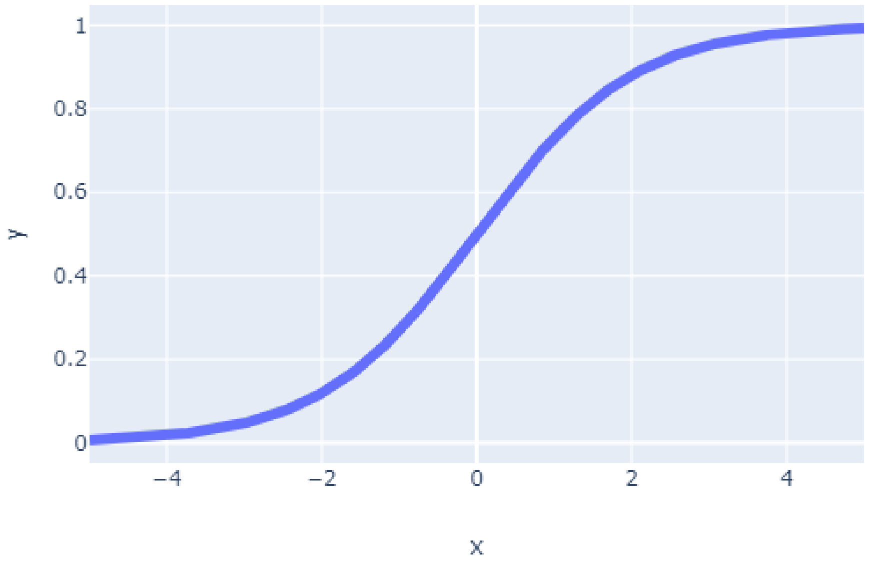
- **Linear binary classifier**
  - Linearly separable data
  - One out of two classes
  - Supervised learning
- **Components**
  - Input values
  - Weights and Biases
  - Sum
  - Activation function
  - Weight update from errors *(training)*
- **Steps**
  1. Multiply inputs by the weights
  2. Add up the values
  3. Apply the activation function
  4. Update the weights given error *(training)*



Step & Error functions

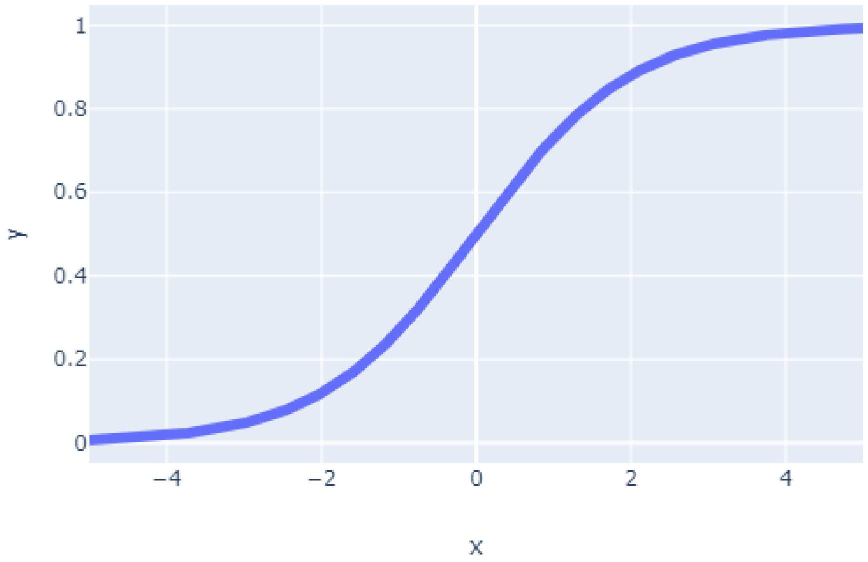
Step & Error functions

- Sigmoid function



Step & Error functions

- Sigmoid function
  - $\frac{1}{1+e^{-x}}$
  - Can be interpreted as probability



Step & Error functions

- Sigmoid function
  - $\frac{1}{1+e^{-x}}$
  - Can be interpreted as probability
- Cross-entropy

Step & Error functions

- Sigmoid function
  - $\frac{1}{1+e^{-x}}$
  - Can be interpreted as probability
- Cross-entropy
  - $-\sum_i (t_i \ln(y_i) + (1 - t_i)\ln(1 - y_i))$
  - Distance between two distributions

Step & Error functions

- Sigmoid function
  - $\frac{1}{1+e^{-x}}$
  - Can be interpreted as probability

- Cross-entropy
  - $-\sum_i (t_i \ln(y_i) + (1-t_i)\ln(1-y_i))$
  - Distance between two distributions
  - Rooted on information theory
    - Huffman coding

The Internals

Is it all a black box?

"For every action there is an equal and opposite reaction"

Newton's 30rd Law of Motion

Setup

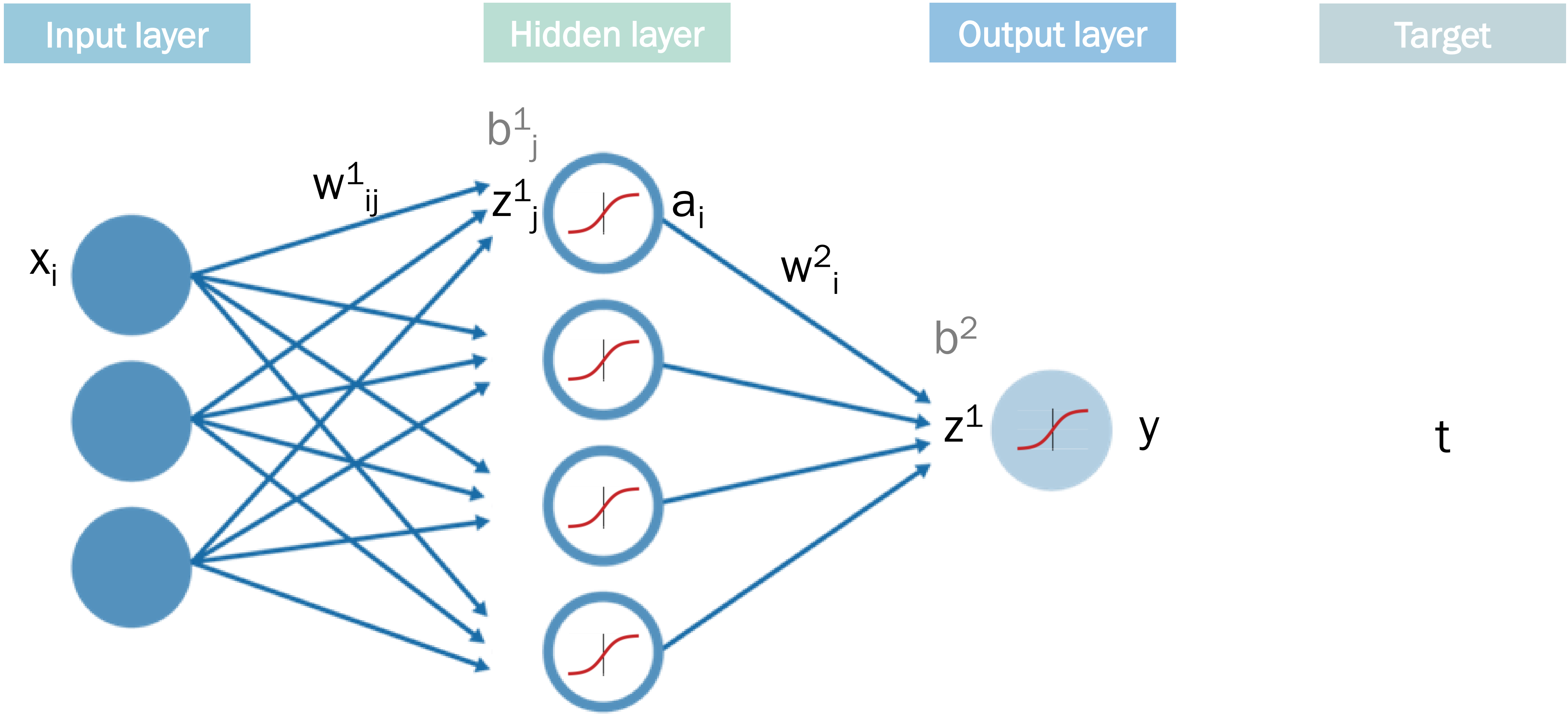
Single datum

Binary classification problem

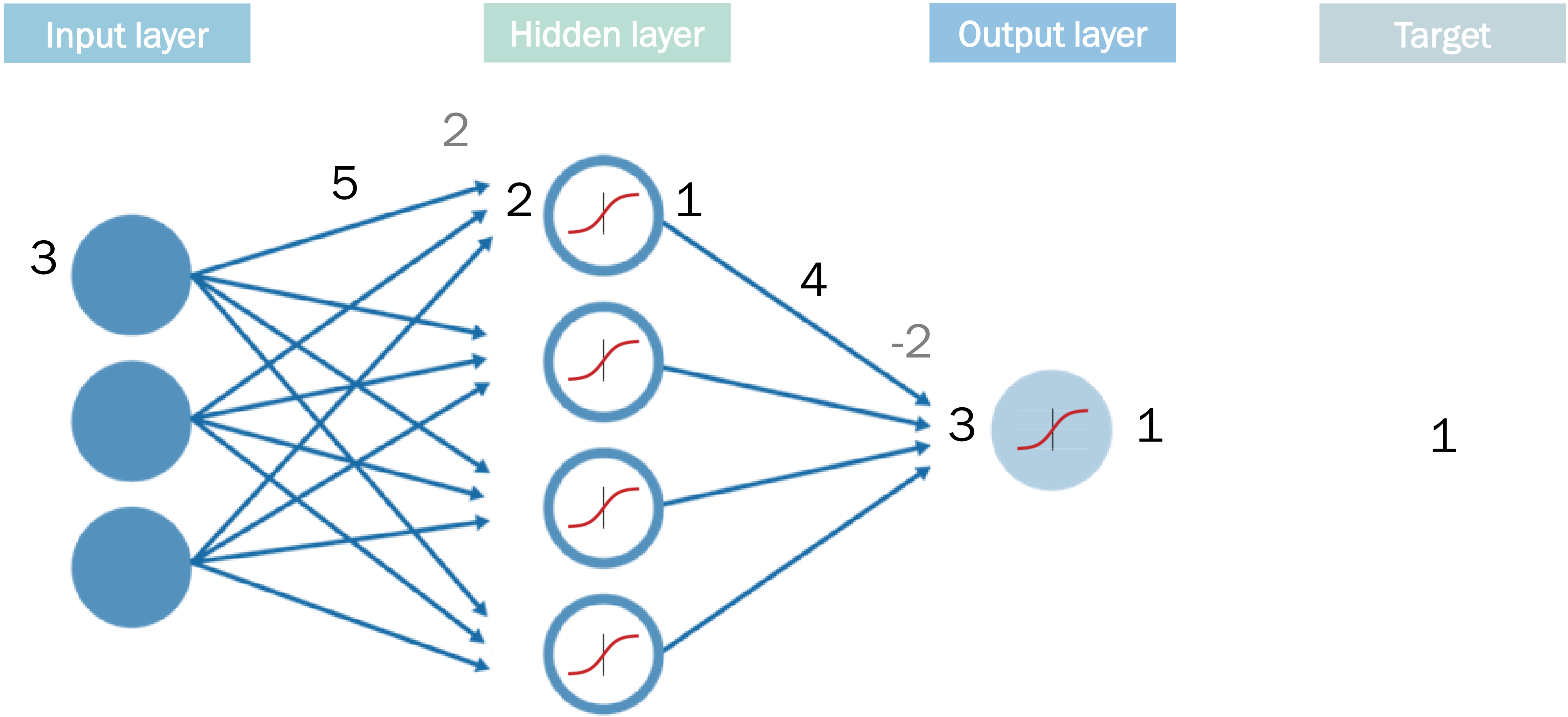
Objective:

$$\arg\min_{w,b} Error$$

Anatomy of a neural network



Anatomy of a neural network



Derivation Steps

Forward Propagation

Error Computation

Back Propagation

Parameters update

Implementation

Does it even work?

"Machines take me by surprise with great frequency."

Alan Turing

Just a tiny bit more of fundamentals 😅

- Dot product of two matrices

```python
import numpy as np

a = np.random.normal(size=(100, 50))
b = np.random.normal(size=(50, 100))

# dot product of `a` and `b`?
```

Just a tiny bit more of fundamentals 😅

- Dot product of two matrices
  - **Iterating** over the arrays
    - 247 ms

```python
import numpy as np

a = np.random.normal(size=(100, 50))
b = np.random.normal(size=(50, 100))

# dot product of `a` and `b`

c_slow = np.zeros((100, 100))

for i in range(a.shape[0]):
    for j in range(b.shape[1]):
        acc = 0
        for k in range(a.shape[1]):
            acc += a[i, k] * b[k, j]
        c_slow[i, j] = acc
```

Just a tiny bit more of fundamentals 😅

- Dot product of two matrices
  - **Iterating** over the arrays
    - 247 ms
  - Numpy's dot product
    - 110 μs

```
import numpy as np

a = np.random.normal(size=(100, 50))
b = np.random.normal(size=(50, 100))

# dot product of `a` and `b`

c = a.dot(b)
```

Just a tiny bit more of fundamentals 😅

- Dot product of two matrices
  - **Iterating** over the arrays
    - 247 ms
  - Numpy's dot product
    - 110 μs

- Vectorization – rewriting loops into efficient parallelizable operations

```python
import numpy as np

a = np.random.normal(size=(100, 50))
b = np.random.normal(size=(50, 100))

# dot product of `a` and `b`

c = a.dot(b)
```

Just a tiny bit more of fundamentals 😅

- Dot product of two matrices
  - **Iterating** over the arrays
    - 247 ms
  - Numpy's dot product
    - 110 μs

- Vectorization – rewriting loops into efficient parallelizable operations
  - "Closer-to-the-metal" computations

```
import numpy as np

a = np.random.normal(size=(100, 50))
b = np.random.normal(size=(50, 100))

# dot product of `a` and `b`

c = a.dot(b)
```

Just a tiny bit more of fundamentals 😅

- Dot product of two matrices
  - **Iterating** over the arrays
    - 247 ms
  - Numpy's dot product
    - 110 μs

- **Vectorization** – rewriting loops into efficient parallelizable operations
  - "Closer-to-the-metal" computations
  - Prone to parallelization

```python
import numpy as np

a = np.random.normal(size=(100, 50))
b = np.random.normal(size=(50, 100))

# dot product of `a` and `b`

c = a.dot(b)
```

Just a tiny bit more of fundamentals 😅

- Dot product of two matrices
  - **Iterating** over the arrays
    - 247 ms
  - Numpy’s dot product
    - 110 μs

- Vectorization – rewriting loops into efficient parallelizable operations
  - “Closer-to-the-metal” computations
  - Prone to parallelization
  - Cleaner code

```python
import numpy as np

a = np.random.normal(size=(100, 50))
b = np.random.normal(size=(50, 100))

# dot product of `a` and `b`

c = a.dot(b)
```

http://bit.ly/2GFMIJP


2016
#WorldofDance
NBC
BIONIC
VOTA
@SB NATION
writtensword.tumblr.com

Today I learned

What we did:

- Derived the math behind traditional neural networks for binary classification
- Implemented and evaluated them using Numpy

Potential paths of exploration:

- Regression
  - Mean squared error
- Multiclass
  - Softmax activation
  - Cross entropy
- Refactor the implementation
- …

Bibliography

- *Deep Learning* by Goodfellow et al. for images and inspiration
- Notes on Backpropagation
- Census income dataset from UCI Repository
- Visual Information Theory by Christopher Olah
- Neural Networks MOOC by Geoffrey Hinton
- What the Hell is a Perceptron by Sagar Sharma

Acknowledgements

- Icons made by Freepik, dDara, surang, Vectors Market, obtained from Flaticon

Thank you

[@diogojapinto](https://twitter.com/diogojapinto)

github.com/diogojapinto

pt.linkedin.com/in/diogojapinto