Software Development Practical

Computer Vision & Deep Learning

Deep Learning

Neurons

Neuron

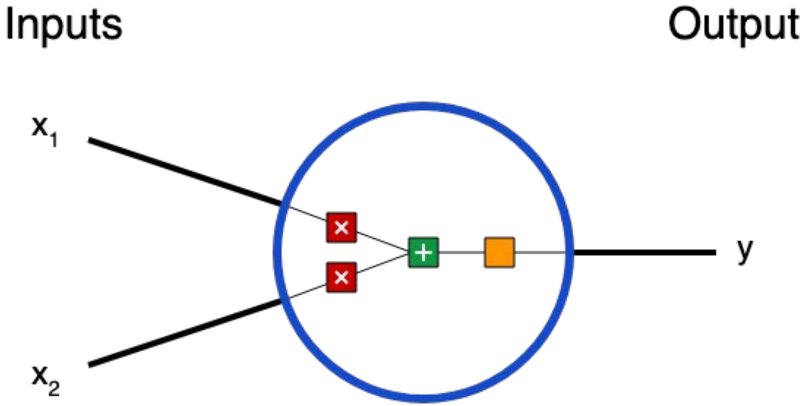
Assume we have a 2D data point with x1 and x2 coordinate.

Three things happen in a neuron.

1. Multiply each input by a weight

$$x_1 \rightarrow x_1 * w_1$$

$$x_2 \rightarrow x_2 * w_2$$



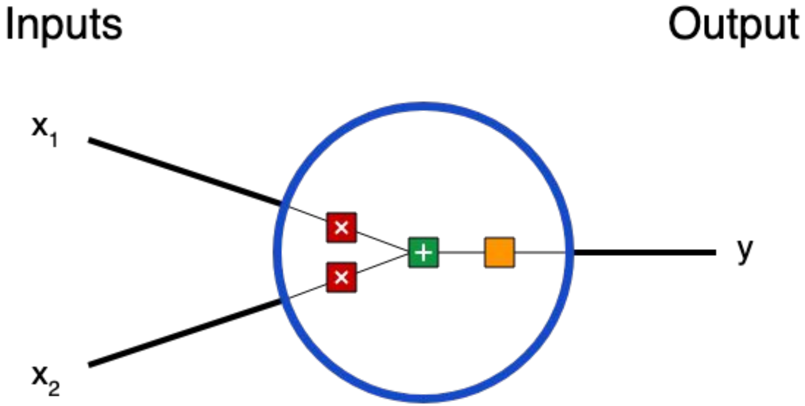
Neuron

Assume we have a 2D data point with x1 and x2 coordinate.

Three things happen in a neuron.

1. Multiply each input by a weight
2. Add weighted inputs and bias

$$(x_1 * w_1) + (x_2 * w_2) + b$$



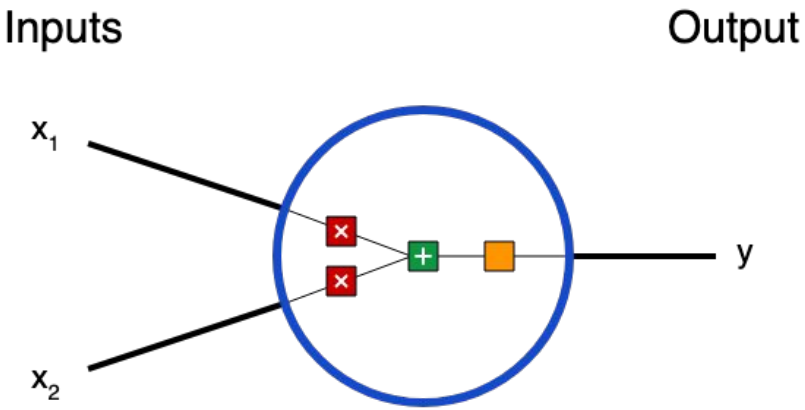
Neuron

Assume we have a 2D data point with x1 and x2 coordinate.

Three things happen in a neuron.

1. Multiply each input by a weight
2. Add weighted inputs and bias
3. Pass sum through activation function

$$y = f(x_1 * w_1 + x_2 * w_2 + b)$$



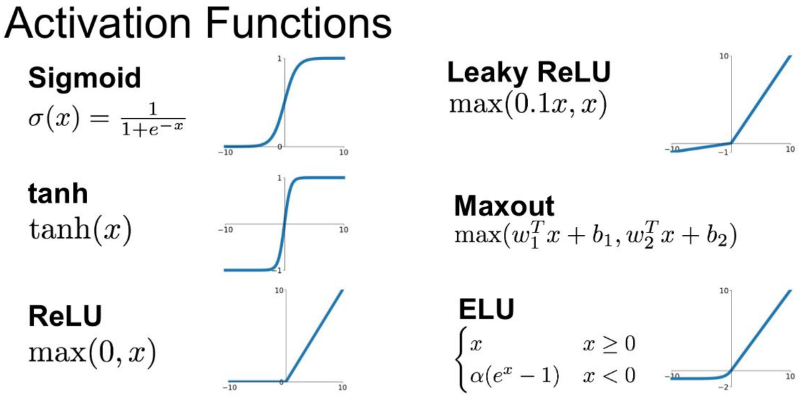
Activation Function

⇒ non-linearity

important ones:

- ReLU → [0, max(x)]
- Sigmoid → [0, 1]
- Softmax → [0, 1] & sum = 1

$$s\left(x_i\right) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$



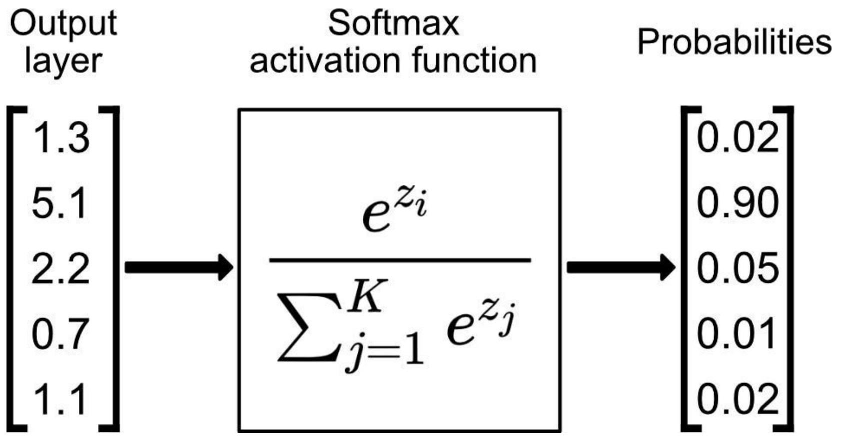
https://medium.com/@shrutijadon/survey-on-activation-functions-for-deep-learning-9689331ba092

Activation Function

Softmax

⇒ convert a vector of real-valued functions into probabilities

⇒ important for e.g. classification

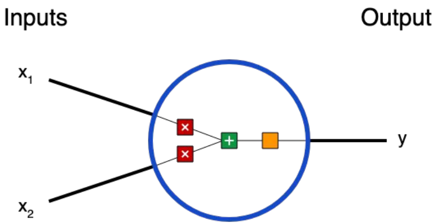


Example

Three things happen in a neuron.

1. Multiply each input by a weight
2. Add weighted inputs and bias
3. Pass sum through activation function

$$y = f(x_1 * w_1 + x_2 * w_2 + b)$$



We have our weight vector and bias:

w = [0, 1] ⇒ w1 = 0; w2 = 1

b = 4

We have our input:

x = [2, 3]

With the dot product we can write

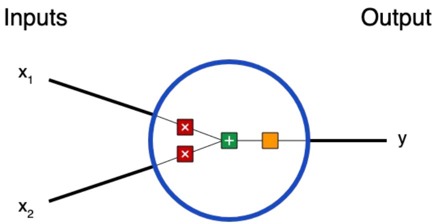
$$(w \cdot x) + b = ((w_1 * x_1) + (w_2 * x_2)) + b$$

Example

Three things happen in a neuron.

1. Multiply each input by a weight
2. Add weighted inputs and bias
3. Pass sum through activation function

$$y = f(x_1 * w_1 + x_2 * w_2 + b)$$



We have our weight vector and bias:

w = [0, 1] ⇒ w1 = 0; w2 = 1

b = 4

We have our input:

x = [2, 3]

With the dot product we can write

$$\begin{aligned}(w \cdot x) + b &= ((w_1 * x_1) + (w_2 * x_2)) + b \\ &= 0 * 2 + 1 * 3 + 4 \\ &= 7\end{aligned}$$

Programming 1

Please code the neuron we just described using python.

You have 20 minutes.

1. Implement the sigmoid function

$$\sigma(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{1+e^x} = 1 - \sigma(-x).$$

2. Plot the sigmoid function for the range -6 to 6
3. Implement a Neuron class

$$y = f(x_1 * w_1 + x_2 * w_2 + b)$$

4. Test it with **w** = [0, 1], **b** = 4 and **x** = [2, 3]

```python
import numpy as np

def sigmoid(x):
  # TODO

class Neuron:
  def __init__(self, weights, bias):
    # TODO

  def feedforward(self, inputs):
    # TODO
    # Weight inputs, add bias, & use activation function
```

Programming 1 - Solution

Please code the neuron we just described using python.

You have 20 minutes.

1. Implement the sigmoid function

$$\sigma(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{1+e^x} = 1 - \sigma(-x).$$

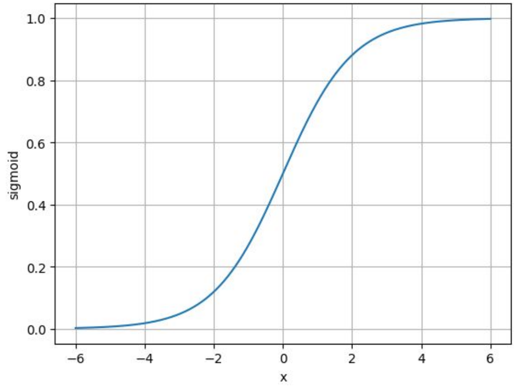
2. Plot the sigmoid function for the range -6 to 6
3. Implement a Neuron class

$$y = f(x_1 * w_1 + x_2 * w_2 + b)$$

4. Test it with **w** = [0, 1], **b** = 4 and **x** = [2, 3]

```
import numpy as np

def sigmoid(x):
  # Our activation function: f(x) = 1 / (1 + e^(-x))
  return 1 / (1 + np.exp(-x))
```



Programming 1 - Solution

Please code the neuron we just described using python.

You have 20 minutes.

1. Implement the sigmoid function

$$\sigma(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{1+e^x} = 1 - \sigma(-x).$$

2. Plot the sigmoid function for the range -6 to 6
3. Implement a Neuron class

$$y = f(x_1 * w_1 + x_2 * w_2 + b)$$

4. Test it with **w** = [0, 1], **b** = 4 and **x** = [2, 3]

```
class Neuron:
  def __init__(self, weights, bias):
    self.weights = weights
    self.bias = bias

  def feedforward(self, inputs):
    # Weight inputs, add bias, & use activation function
    total = np.dot(self.weights, inputs) + self.bias
    return sigmoid(total)
```

Programming 1 - Solution

Please code the neuron we just described using python.

You have 20 minutes.

1. Implement the sigmoid function

$$\sigma(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{1+e^x} = 1 - \sigma(-x).$$

2. Plot the sigmoid function for the range -6 to 6
3. Implement a Neuron class

$$y = f(x_1 * w_1 + x_2 * w_2 + b)$$

4. Test it with **w** = [0, 1], **b** = 4 and **x** = [2, 3]

```
weights = np.array([0, 1]) # w1 = 0, w2 = 1
bias = 4                   # b = 4
n = Neuron(weights, bias)

x = np.array([2, 3])       # x1 = 2, x2 = 3
print(n.feedforward(x))    # 0.9990889488055994
```

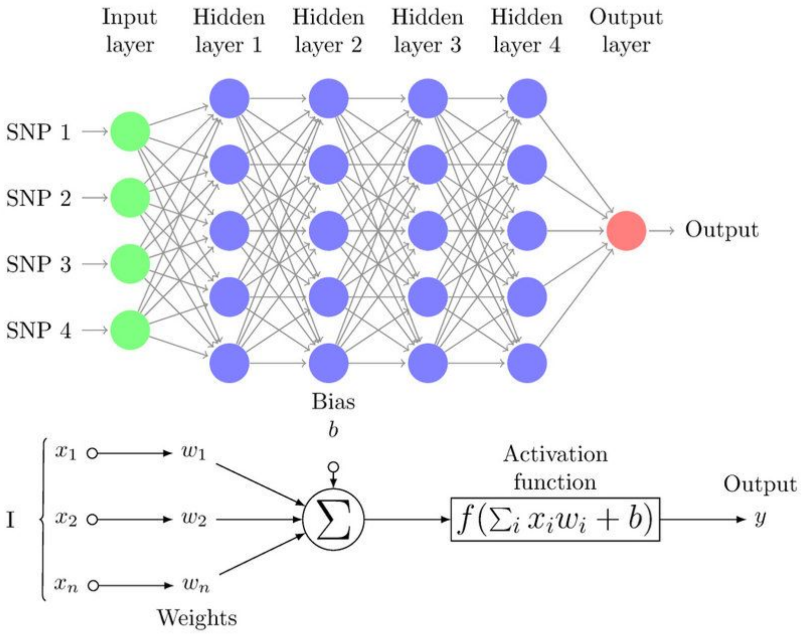
Deep Learning

Multilayer Perceptrons

Neural Networks

Neural networks are nothing more than multiple neurons connected together.

Back-propagation of loss is used for computing gradients for optimization.

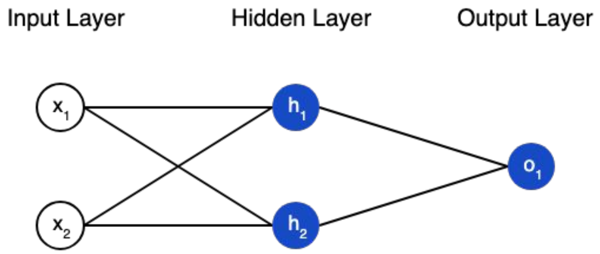


Neural Networks

Example

Assume we have a network with all weights having the same weights **w** = [0, 1] and same bias **b = 0**, let h1, h2, o2 denote the outputs of the neurons they represent.

We call it h because its a *hidden layer.*



E.g. input **x** = [2, 3]:

$$\begin{aligned} h_1 = h_2 &= f(w \cdot x + b) \\ &= f((0 * 2) + (1 * 3) + 0) \\ &= f(3) \\ &= 0.9526 \end{aligned}$$

$$\begin{aligned} o_1 &= f(w \cdot [h_1, h_2] + b) \\ &= f((0 * h_1) + (1 * h_2) + 0) \\ &= f(0.9526) \\ &= \boxed{0.7216} \end{aligned}$$

Programming 2

Please code the neural network we just described using python.

The neural network has

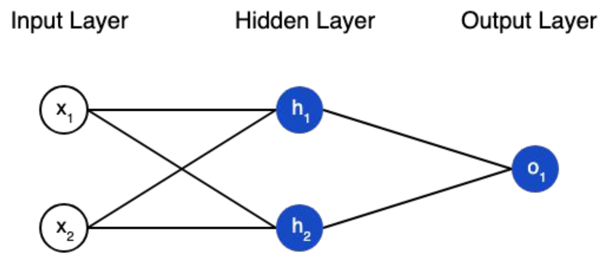
- 2 inputs
- a hidden layer with 2 neurons (h1, h2)
- an output layer with 1 neuron (o2)

Each neuron has the same weights and bias **w** = [0, 1] and **b** = 0.

Test the neural network with **x** = [2, 3]

You have 20 minutes.

```python
class NeuralNetwork:
  def __init__(self):
    # TODO
  def feedforward(self, x):
    # TODO
```



Programming 2 - Solution

Please code the neural network we just described using python.

The neural network has

- 2 inputs
- a hidden layer with 2 neurons (h1, h2)
- an output layer with 1 neuron (o2)

Each neuron has the same weights and bias **w** = [0, 1] and **b** = 0.

Test the neural network with **x** = [2, 3]

```python
class NeuralNetwork:
  def __init__(self):
    weights = np.array([0, 1])
    bias = 0

    self.h1 = Neuron(weights, bias)
    self.h2 = Neuron(weights, bias)
    self.o1 = Neuron(weights, bias)

  def feedforward(self, x):
    out_h1 = self.h1.feedforward(x)
    out_h2 = self.h2.feedforward(x)
    out_o1 = self.o1.feedforward(np.array([out_h1, out_h2]))
    return out_o1
```

Programming 2 - Solution

Please code the neural network we just described using python.

The neural network has

- 2 inputs
- a hidden layer with 2 neurons (h1, h2)
- an output layer with 1 neuron (o2)

Each neuron has the same weights and bias **w** = [0, 1] and **b** = 0.

Test the neural network with **x** = [2, 3]

```
network = NeuralNetwork()
x = np.array([2, 3])
print(network.feedforward(x)) # 0.7216325609518421
```

Deep Learning

Training a Neural Network

Machine Learning Terms

Input

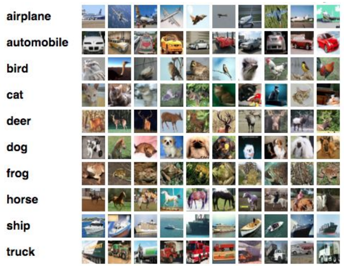
- Data points that we want to process
- e.g. Images, Audio, etc

Output / Labels

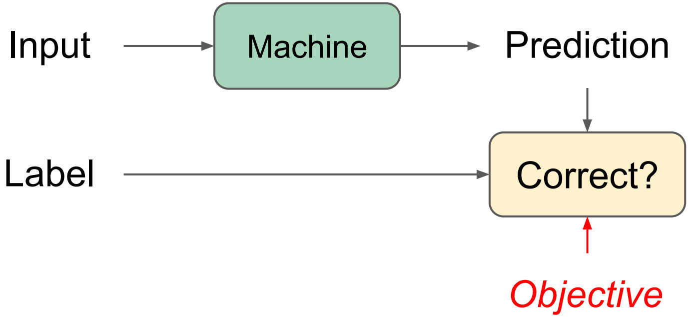
- Corresponding value / label for each datapoint
- e.g. temperature or “car”

Objective

- Feedback signal
- Measures whether algorithm does a good job



https://www.analyticsvidhya.com/blog/2020/02/learn-image-classification-cnn-convolutional-neural-networks-3-datasets/



Machine Learning Terms

Input

- Data points that we want to process
- e.g. Images, Audio, etc

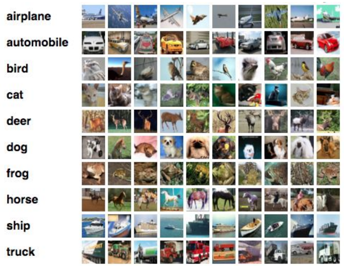
Output / Labels

- Corresponding value / label for each datapoint
- e.g. temperature or “car”

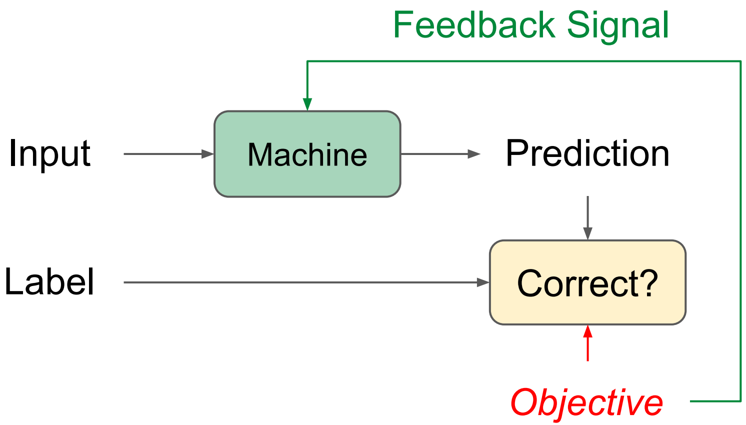
Objective

- Feedback signal
- Measures whether algorithm does a good job

⇒ Used to change the parameters of our model



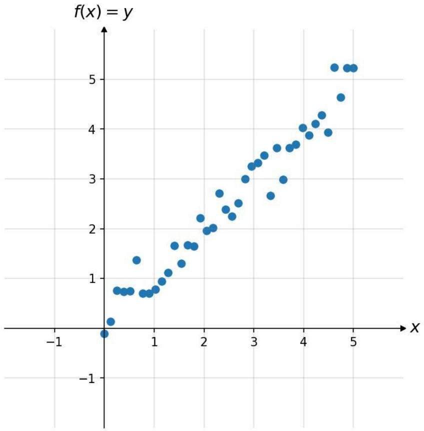
https://www.analyticsvidhya.com/blog/2020/02/learn-image-classification-cnn-convolutional-neural-networks-3-datasets/



Objective Function

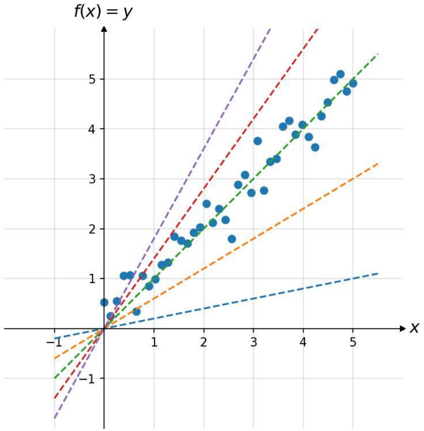
data

pairs of (x, y) values



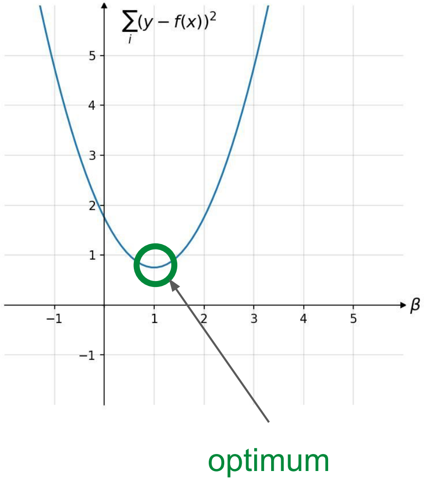
model

$$f_\beta(x) = y = \beta x$$



cost function / objective

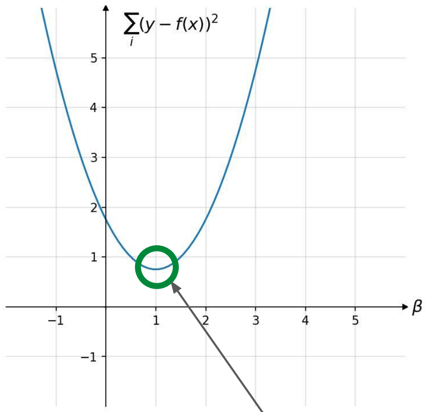
$$1/n \sum_i^n (y - f_\beta(x))^2$$



Objective Function / Loss

⇒ Quantifies how “good” our model is

- must be differentiable w.r.t. to parameters
- outputs a scalar
- better predictions = lower loss



optimum

Mean Squared Error

$$MSE(y, \hat{y}) = \frac{1}{n} \sum_{i=1}^{n} (y - \hat{y})^2$$

Ground truth

Prediction

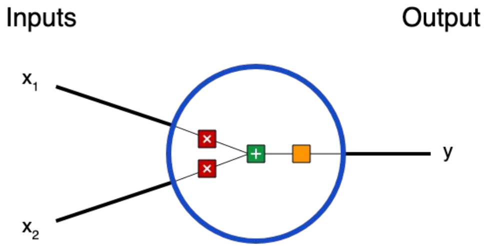
Objective Function / Loss

⇒ Quantifies how "good" our model is

Example for a single sample

Our neural network: $h(\mathbf{x}; \mathbf{w}, b) = \sigma(\mathbf{w}^T \mathbf{x} + b)$

The loss function:
$$\begin{aligned} L(\mathbf{x}, y; \mathbf{w}, b) &= (y - \hat{y})^2 \\ &= (y - h(\mathbf{x}; \mathbf{w}, b))^2 \\ &= (y - \sigma(\mathbf{w}^T \mathbf{x} + b))^2 \\ &= (y - \sigma(w_1 \cdot x_1 + w_2 \cdot x_2 + b))^2 \end{aligned}$$



⇒ How much do we need to tweak **w** and **b** to decrease L?

Partial Derivative

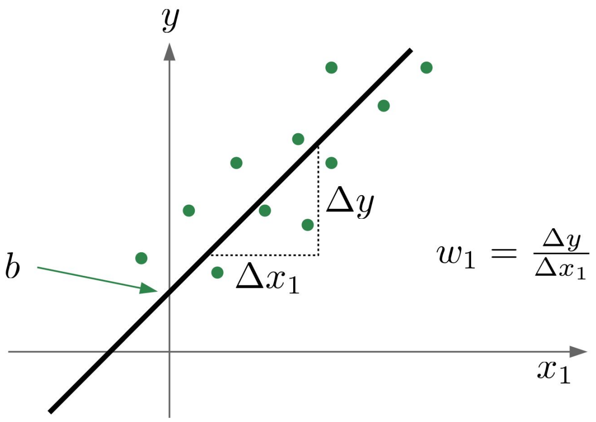
How would L change if we change w1?

⇒ **partial derivative** $\frac{\partial L}{\partial w_1}$

How can we get the partial derivative?

$$L(\mathbf{x}, y; \mathbf{w}, b) = (y - \sigma(w_1 \cdot x_1 + w_2 \cdot x_2 + b))^2$$

⇒ **Chain rule**



Chain Rule

The derivative of the composition of two functions *f* and *g* can be expressed as

$$h'(x) = f'(g(x)) \cdot g'(x)$$

or equivalently

$$h' = (f \circ g)' = (f' \circ g) \cdot g'.$$

Example

$$h(x) = (2x^3 + 3x)^2$$

$$\text{with} \quad u(x) := 2x^3 + 3x$$

$$\boxed{\begin{aligned} h(u) &= u^2 \\ \frac{\partial h(u)}{\partial u} &= 2u \end{aligned}}$$

$$\boxed{\frac{\partial u(x)}{\partial x} = 6x^2 + 3}$$

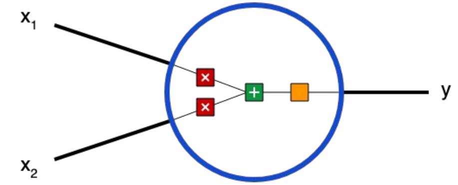
$$\begin{aligned} \frac{\partial h}{\partial x} &= \frac{\partial h}{\partial u} \cdot \frac{\partial u}{\partial x} \\ &= \boxed{2 \cdot (2x^3 + 3x)} \cdot \boxed{(6x^2 + 3)} \end{aligned}$$

Chain Rule

For our example

$$L(\mathbf{x}, y; \mathbf{w}, b) = (y - \sigma(w_1 \cdot x_1 + w_2 \cdot x_2 + b))^2$$

$$g := w_1 \cdot x_1 + w_2 \cdot x_2 + b$$
$$\hat{y} := \sigma(g)$$
$$L := (y - \hat{y})^2$$



$$L(\mathbf{x}, y; \mathbf{w}, b) = (y - \sigma(\underbrace{w_1 \cdot x_1 + w_2 \cdot x_2 + b}_{g}))^2$$
$$= (y - \underbrace{\sigma(g)}_{\hat{y}})^2$$
$$= (y - \hat{y})^2$$

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial g} \cdot \frac{\partial g}{\partial w_1}$$

$$\boxed{\frac{\partial L}{\partial \hat{y}} = -2 \cdot (y - \hat{y})}$$

$$\boxed{\frac{\partial \hat{y}}{\partial g} = \sigma(g) \cdot (1 - \sigma(g))}$$

$$\boxed{\frac{\partial g}{\partial w_1} = x_1}$$

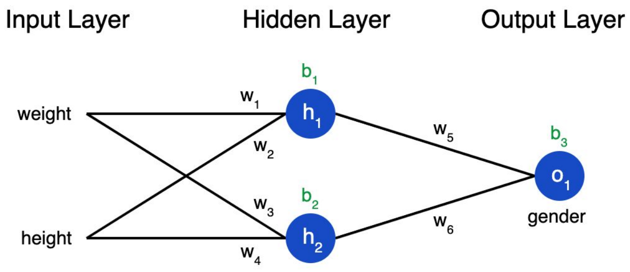
Chain Rule

More complicated example

$$L(w_1, w_2, w_3, w_4, w_5, w_6, b_1, b_2, b_3)$$

$$\frac{\partial L}{\partial w_1} = \boxed{\frac{\partial L}{\partial \hat{y}}} \cdot \frac{\partial \hat{y}}{\partial w_1}$$

$$\boxed{\frac{\partial L}{\partial \hat{y}} = \frac{\partial (1-\hat{y})^2}{\partial \hat{y}} = -2(y-\hat{y})}$$



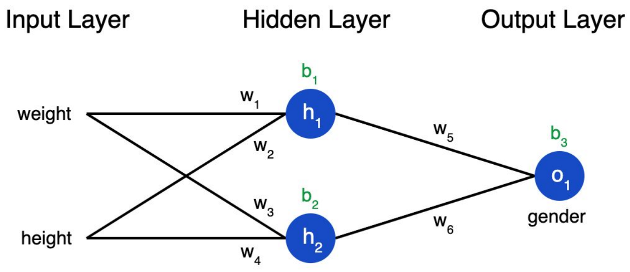
Chain Rule

More complicated example

$$L(w_1, w_2, w_3, w_4, w_5, w_6, b_1, b_2, b_3)$$

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial \hat{y}} \cdot \boxed{\frac{\partial \hat{y}}{\partial w_1}}$$

$$\hat{y} = o_1 = f(w_5 \cdot h_1 + w_6 \cdot h_2 + b_3)$$



since w1 only affects h1 we can write

$$\boxed{\frac{\partial \hat{y}}{\partial w_1} = \frac{\partial \hat{y}}{\partial h_1} \cdot \frac{\partial h_1}{\partial w_1}} \longrightarrow \boxed{\frac{\partial \hat{y}}{\partial h_1} = w_5 \cdot f'(w_5 \cdot h_1 + w_6 \cdot h_2 + b_3)}$$

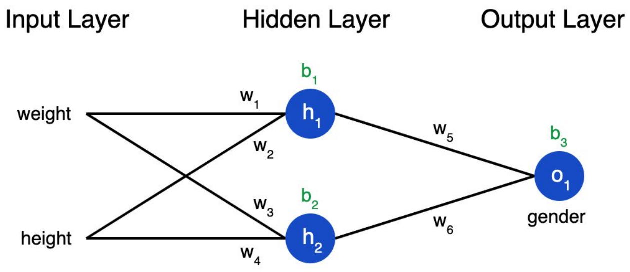
Chain Rule

More complicated example

$$L(w_1, w_2, w_3, w_4, w_5, w_6, b_1, b_2, b_3)$$

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial \hat{y}} \cdot \boxed{\frac{\partial \hat{y}}{\partial w_1}}$$

$$\hat{y} = o_1 = f(w_5 \cdot h_1 + w_6 \cdot h_2 + b_3)$$



since w1 only affects h1 we can write

$$\boxed{\frac{\partial \hat{y}}{\partial w_1} = \frac{\partial \hat{y}}{\partial h_1} \cdot \frac{\partial h_1}{\partial w_1}}$$

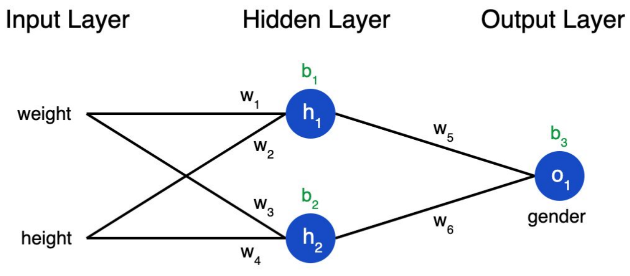
$$\boxed{\frac{\partial \hat{y}}{\partial h_1} = w_5 \cdot f'(w_5 \cdot h_1 + w_6 \cdot h_2 + b_3)}$$

$$\boxed{\frac{\partial h_1}{\partial w_1} = x_1 \cdot f'(w_1 \cdot x_1 + w_2 \cdot x_2 + b_1)}$$

Chain Rule

More complicated example

$$L(w_1, w_2, w_3, w_4, w_5, w_6, b_1, b_2, b_3)$$



$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial h_1} \cdot \frac{\partial h_1}{\partial w_1}$$

$$\boxed{\frac{\partial L}{\partial \hat{y}} = \frac{\partial (1-\hat{y})^2}{\partial \hat{y}} = -2(y-\hat{y})}$$

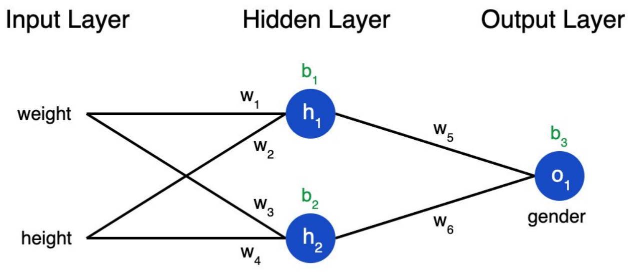
$$\boxed{\frac{\partial \hat{y}}{\partial h_1} = w_5 \cdot f'(w_5 \cdot h_1 + w_6 \cdot h_2 + b_3)}$$

$$\boxed{\frac{\partial h_1}{\partial w_1} = x_1 \cdot f'(w_1 \cdot x_1 + w_2 \cdot x_2 + b_1)}$$

⇒ **So for what is this good now?**

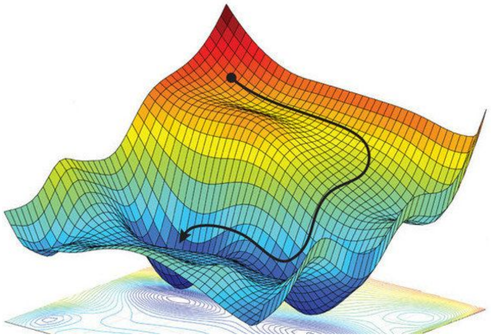
Gradient Descent

1. Initialize parameters at random
2. Compute error
3. Compute gradients w.r.t. parameters
4. Apply update rule
5. Repeat from step 2. until the error does not decrease anymore



Update Rule

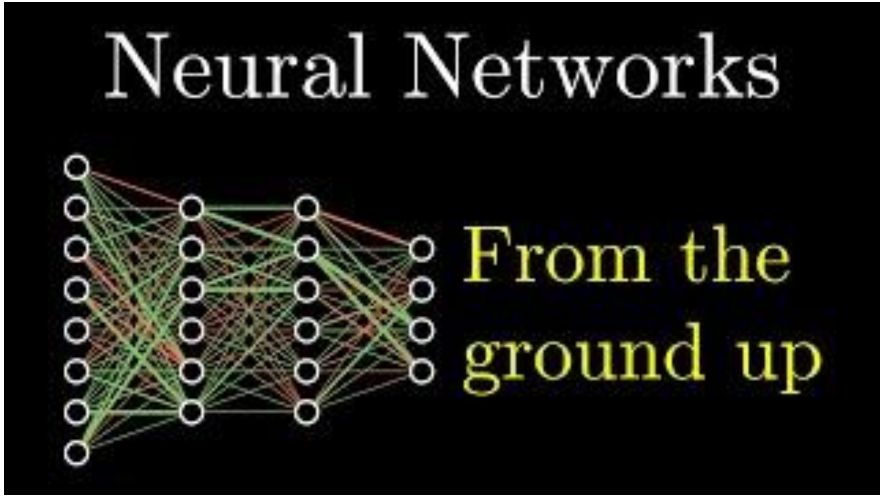
$$w_1 \leftarrow w_1 - \alpha \frac{\partial L}{\partial w_1}$$



https://www.researchgate.net/figure/Non-convex-optimization-We-utilize-stochastic-gradient-descent-to-find-a-local-optimum_fig1_325142728

3blue1brown

Really nice introduction video into neural networks…



Thanks for your Attention

Next Week: Dive deeper into Gradient Descent