Deep Neural Network
Introduction and Example

Christoph Rhein, SAP
November, 2017

Agenda

Architecture

Artificial Neurons

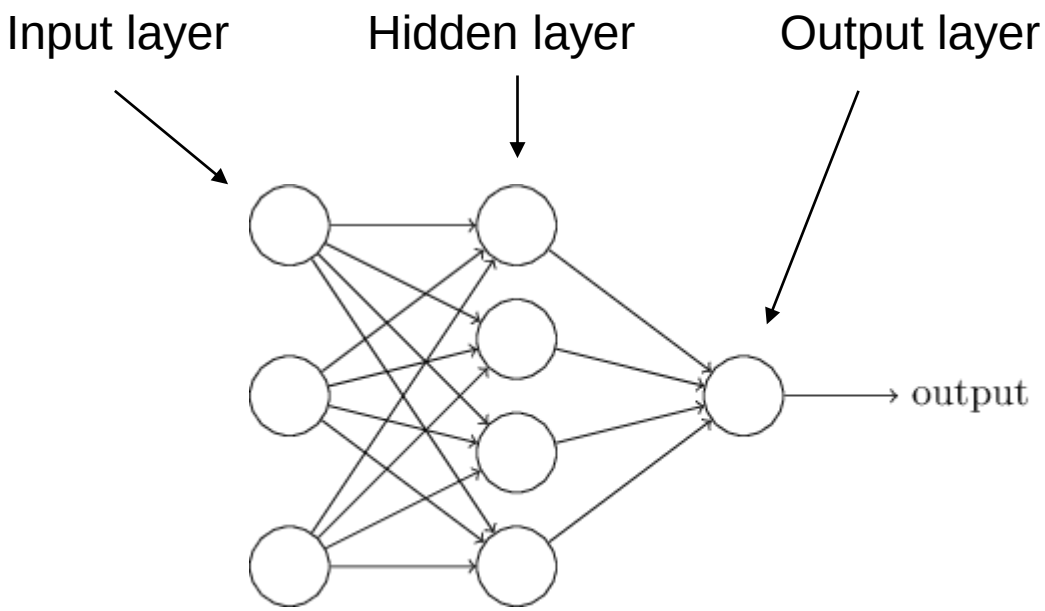
- Perceptron
- Sigmoid Neuron

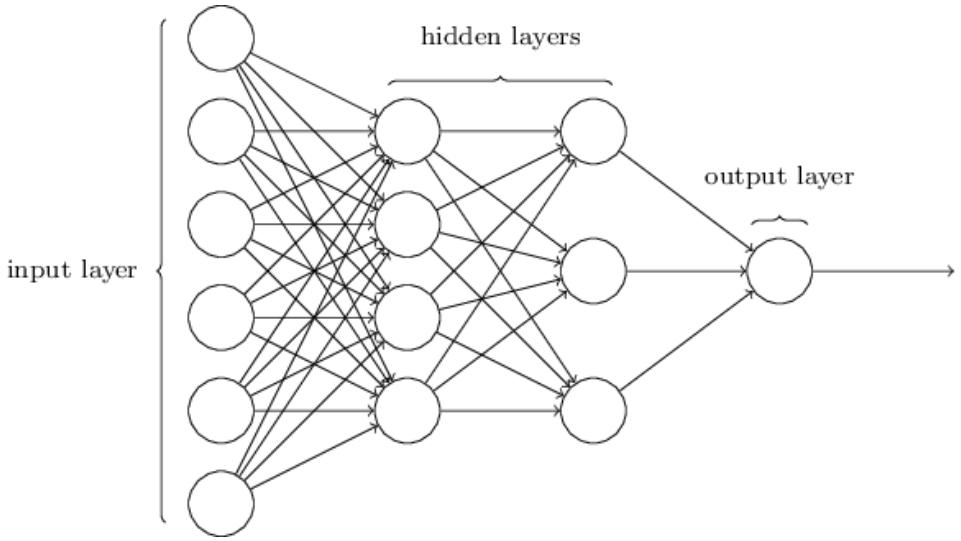
Network Training

- Cost Function
- Back Propagation

Demo

Neural Network - Architecture





Hyperparameters: Establish the structure and behavior of a network but are not updated by the training.

Neural Network - Architecture

Feedforward Neural Network: Output from one layer is the input of the next layer. No loops. Information is always fed forward, never back

Recurrent Neural Network: Feedback loops. The idea is that these models have neurons that fire for some limited duration of time before becoming quiescent. Their output can stimulate other neurons and so on. Over time we get a cascade of neurons firing. Loops are no problem since neuron’s output only affects its input at some later time, not instantaneously.

Neural Network - Architecture

Recurrent Neural Network: Long Short-Term Memory; good for language processing

Feedforward Neural Network: Convolutional Neural Network; good for image processing

Neural Network - Architecture

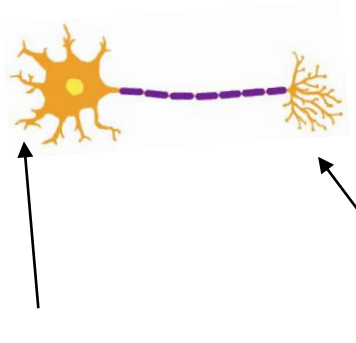
Supervised learning

The model is trained on a labeled dataset, so it can predict the outcome of out-of-sample data. Used for classification and regression.

Unsupervised learning

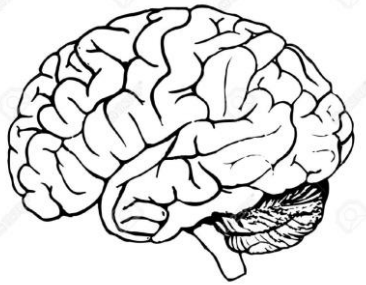
The model works on its own to discover information and draws conclusions on unlabeled data.

Neural Network - It's all about Neurons

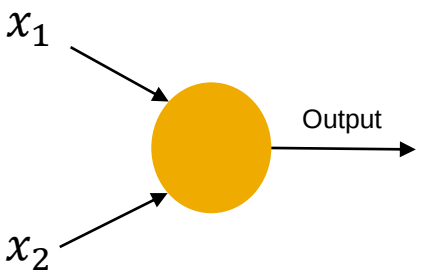


Dendrites Axon terminal

The human brain contains around 140 millions neurons with tens of billions connections



Artificial Neuron



Perceptron developed in 1950 and 1960 by Frank Rosenblatt inspired by earlier work from Warren McCulloch and Walter Pitts

binary input – binary output

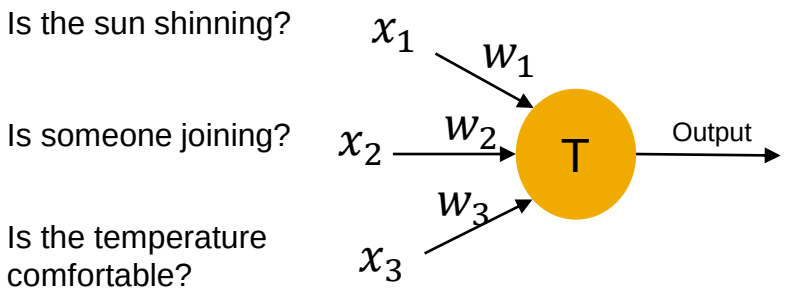
Neural Network - Perceptron

Example: Should I go for a run?

Weights: How important is each input?

Threshold: When does the neuron fire?

Output: 0 if $\Sigma_j w_j x_j \leq \text{threshold}$
1 if $\Sigma_j w_j x_i > \text{threshold}$



No = 0
Yes = 1

$w_1 = 6$
$w_2 = 2$
$w_3 = 3$
T = 5

Sunny day, alone and chilly: $x_1 = 1$ $x_2 = 0$ $x_3 = 0$

Output: 1x6 + 0x2 + 0x3 > 5 => run

Changing **weights** and **threshold** changes my **willingness** to go for a run.

Neural Network - Perceptron

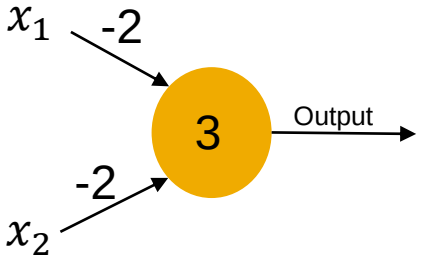
A perceptron can be described as a method for weighing evidence to make decisions.

Simplified notation: $\sum_j w_j x_j > T \equiv w \times x + b$

Output = $\begin{cases} 0 \text{ if } w \times x + b \leq 0 \\ 1 \text{ if } w \times x + b > 0 \end{cases}$

B = Bias

Another way perceptrons can be used is to compute elementary logical functions we usually think of as underlying computation (functions such as AND, OR and NAND).



Input 0;0 produces output 1: (0 x (-2) + 0 x (-2) + 3 = 3 (3 > 0 => 1)

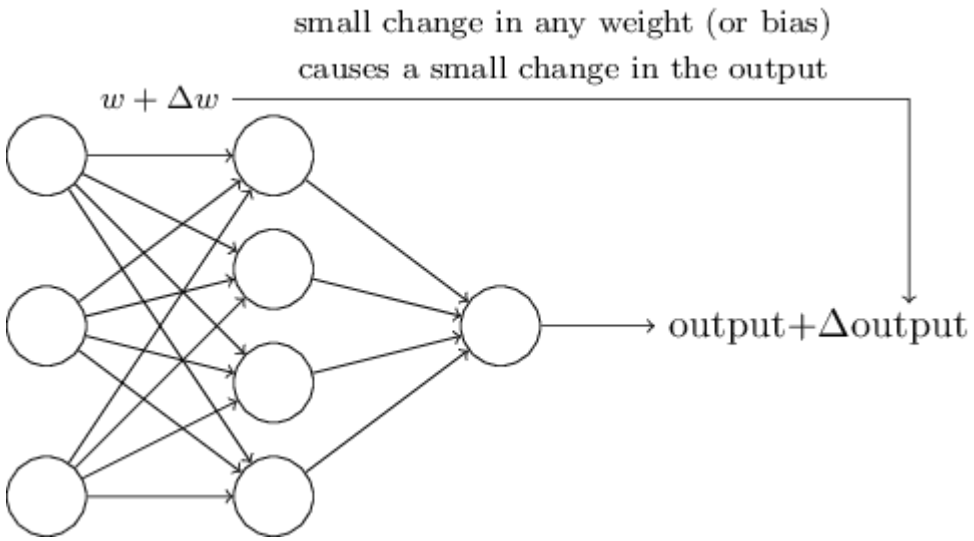
0	0	1
0	1	1
1	0	1
1	1	0

NAND

Neural Network – From Perceptrons to Sigmoid Neurons

Percetprons are universal for computations.
What's the big news?

We can devise **learning algorithms** which can automatically tune the **weights** and **biases** of a network of artificial neurons.

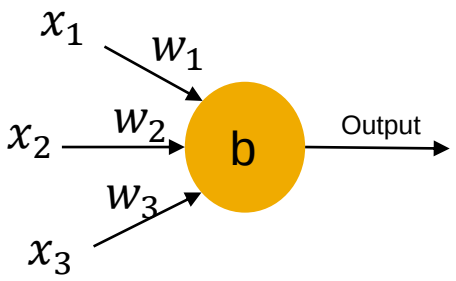


Example: We have raw pixel data from an image of a digit. The network should learn to read the digit correctly by adjusting weights and biases. The network might mistakenly classify an image as '8' when it should be a '9'. We could figure out how to make small changes to the weights and biases so the network gets closer to classify the image as '9'. Repeating this over and over again the network would be learning.
Unfortunately this is not happening if our network contains perceptrons.

A small change the weights or bias of any single perceptron in the network can sometimes cause the output of that perceptron to completely flip (say from 0 to 1). That flip might change the rest of the network completely. While the '9' might be classified now correctly the behavior of the network on all the other images is likely to have completely changed in some hard to control way.

Neural Network – From Perceptrons to Sigmoid Neurons

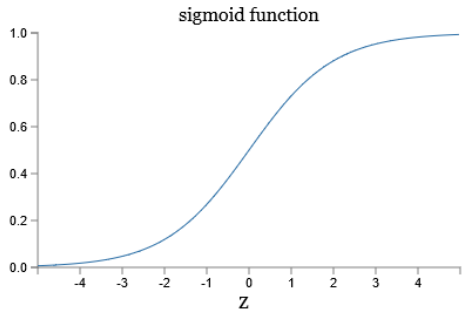
Sigmoid neurons are similar to perceptrons but modified so that **small changes** to their **weights and bias** cause only a **small change** in their **output.**



Input is **between** 0 and 1
Output is $\sigma(w \times x + b)$; also between 0 and 1

σ is called the sigmoid function

$$\sigma(z) \equiv \frac{1}{1 + e^{-z}}$$



By using the sigmoid function we get a smoothed out perceptron.

Neural Network – Sigmoid Neurons

The smoothness of σ means that small changes Δw_j in the weights and Δb in the bias will produce a small change $\Delta output$ in the output of the neuron.

$$\Delta output \approx \sum_j \frac{\partial output}{\partial w_j} \Delta w_j + \frac{\partial output}{\partial b} \Delta b$$

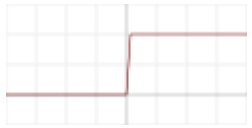
Don’t panic if you are not comfortable with partial derivatives!

The expression means: $\Delta output$ is a linear function of the changes Δw_j and Δb in the weights and bias.

The **activation function** in a sigmoid neuron is the sigmoid function.

Neural Network – Activation Functions

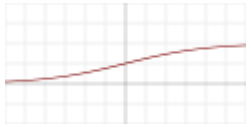
Binary step



$$f(x) = \begin{cases} 0 \; for \; x < 0 \\ 1 for \; x \geq 0 \end{cases} \qquad f'(x) = \begin{cases} 0 \; for \; x \neq 0 \\ ? \; for \; x = 0 \end{cases}$$

{0,1}

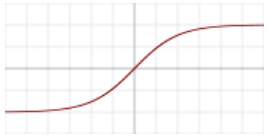
Sigmoid function (a.k.a. soft step)



$$f(x) = \frac{1}{1 + e^{-z}} \qquad f'(x) = f(x)(1 - f(x))$$

(0,1)

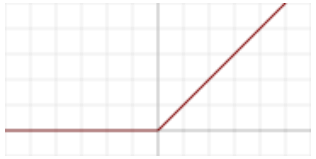
TanH



$$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1 \qquad f'(x) = 1 - f(x)^2$$

(-1,1)

Rectified linear unit (ReLU)



$$f(x) = \begin{cases} 0 \; for \; x < 0 \\ x \; for \; x \geq 0 \end{cases} \qquad f'(x) = \begin{cases} 0 \; for \; x < 0 \\ 1 \; for \; x \geq 0 \end{cases}$$

[0,∞]

Neural Network – Example

Predict our test score based on hours of sleep the night before and hours studied.

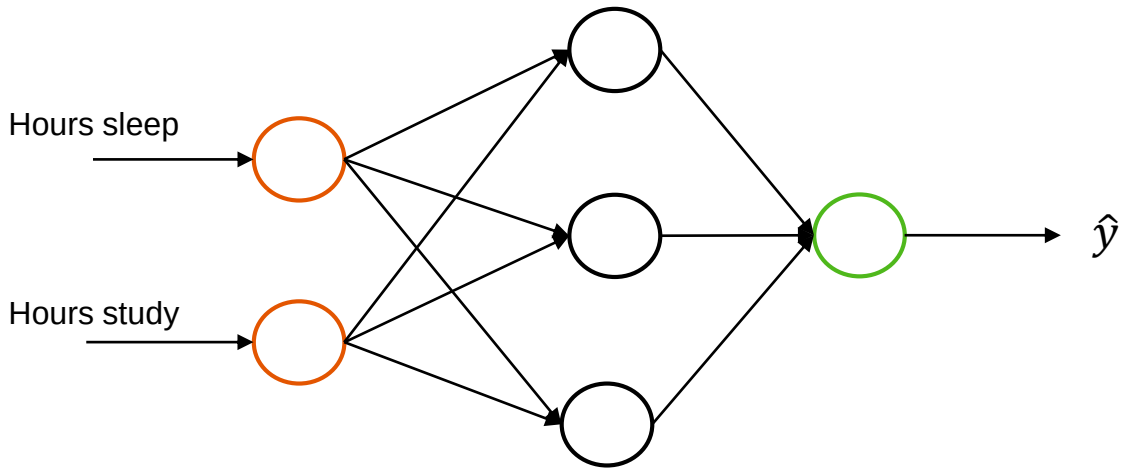
Sleep (h) , Study (h) : Score

(3 , 5)	75
(5 , 1)	82
(10, 2)	93
(8 , 2)	???

Training data

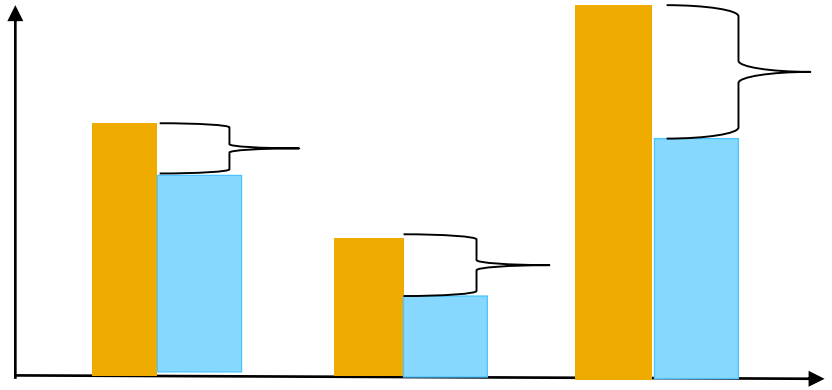
Supervised Regression Problem

Neural Network – Example



Neural Network – Training the network

Right now our model does a pretty bad job in predicting our test score. To improve our model we need to quantity how wrong our predictions are.



Test data

Prediction

Cost function

J = costs = $e_1^2 + e_2^2 + e_3^2$

J = $\sum \frac{1}{2} e^2 = \sum \frac{1}{2} (y - \hat{y})^2$

Training the network = minimizing a cost function

Neural Network – Minimizing the cost function

What are the influential factors of the cost function?

- Our example data
- weights

We can’t change our test data therefore we can only change the weights in order to minimize the error.

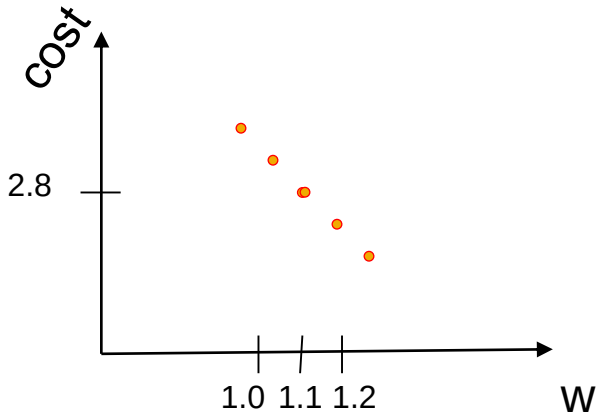
Challenge: Find the combination of weights that minimize the error.

Q: Why not just try all the possible combinations of all weights and see when the error is the smallest?

A: Curse of Dimensionality

Neural Network – Minimizing the cost function

Evaluate the cost function for a specific value of w.

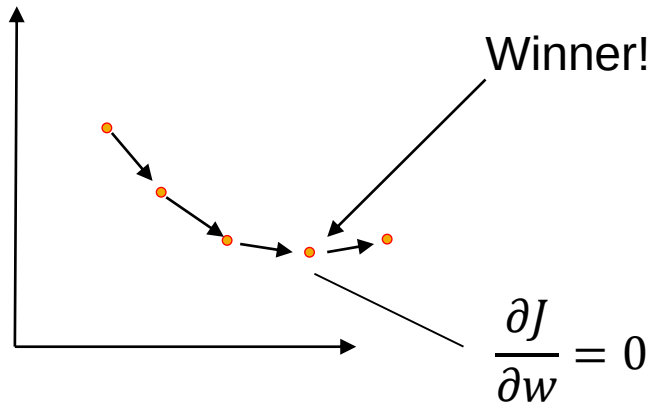


Which way is downhill? => lower the cost

We can test the cost function immediately with values left and right to our test point and see which one is smaller

Numerical Gradient Estimation

Neural Network – Gradient Descent



Advantage:

- We know in which direction we have to move
- We can take iteratively steps and stopping when the cost stops getting smaller

Neural Network – Gradient Descent

(1) $z^{(2)} = x \times w^{(1)}$

(2) $a^{(2)} = f\left(z^{(2)}\right)$

(3) $z^{(3)} = a^{(2)} \times w^{(2)}$

(4) $\hat{y} = f\left(z^{(3)}\right)$

(5) $J = \sum \frac{1}{2}(y - \hat{y})^2$

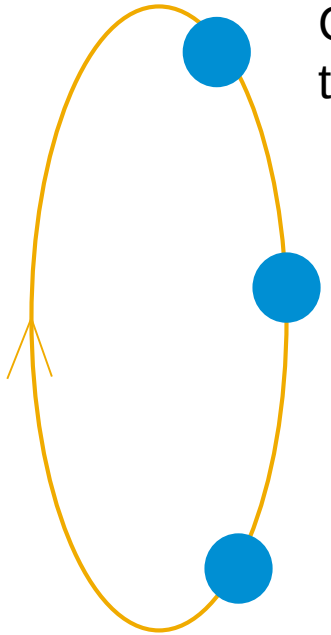
$$J = \sum \frac{1}{2}\left(y - f\left(f\left(x \times w^{(1)}\right) \times w^{(2)}\right)\right)^2$$

We want to know how J is changing (uphill/downhill) with respect to w.

$\frac{\partial J}{\partial w}$ = partial derivative, the change in error when I change a specific weight

Neural Network – Backpropagation

Is an algorithm calculating the gradients (partial derivative) with respect to all weights in a network and then change the weights in the direction of the gradient at a learning rate.

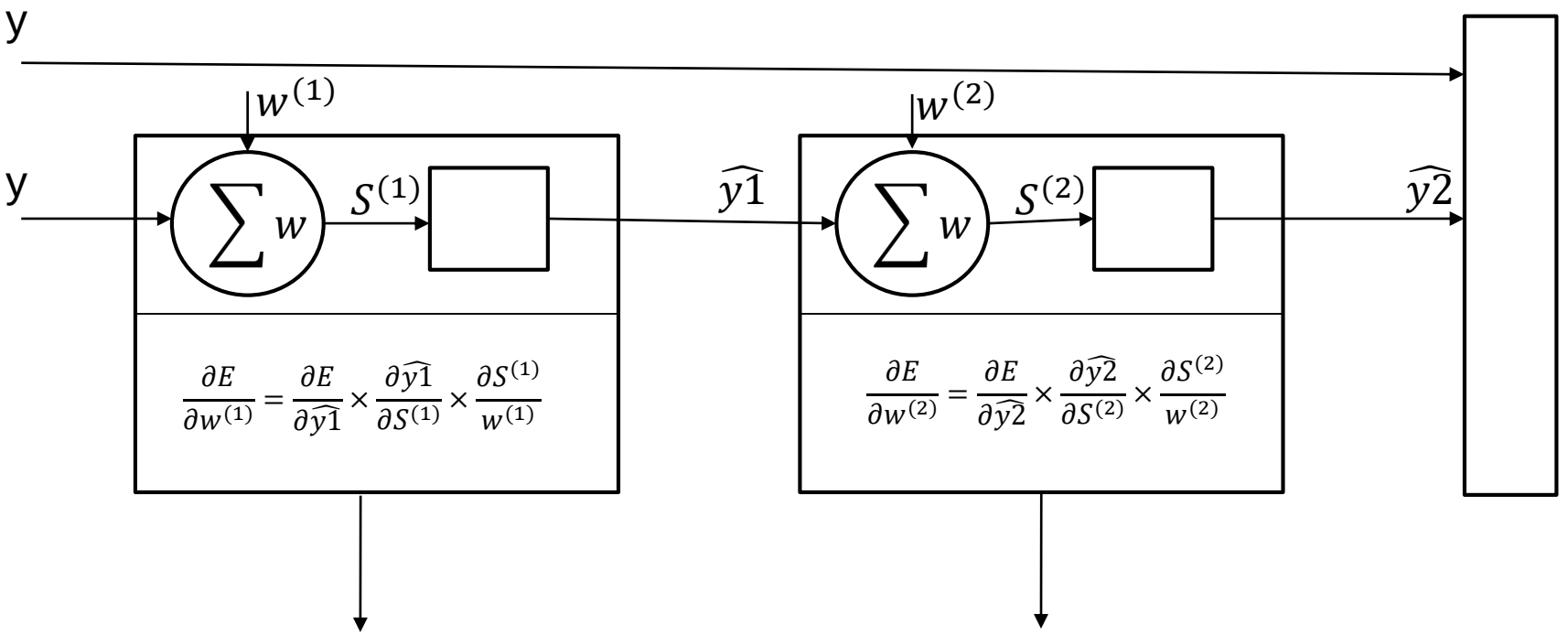


Compute the error E over the dataset

Compute the gradient with respect to weights

Move the weights in the direction of the gradient at the learning rate

Neural Network – Backpropagation



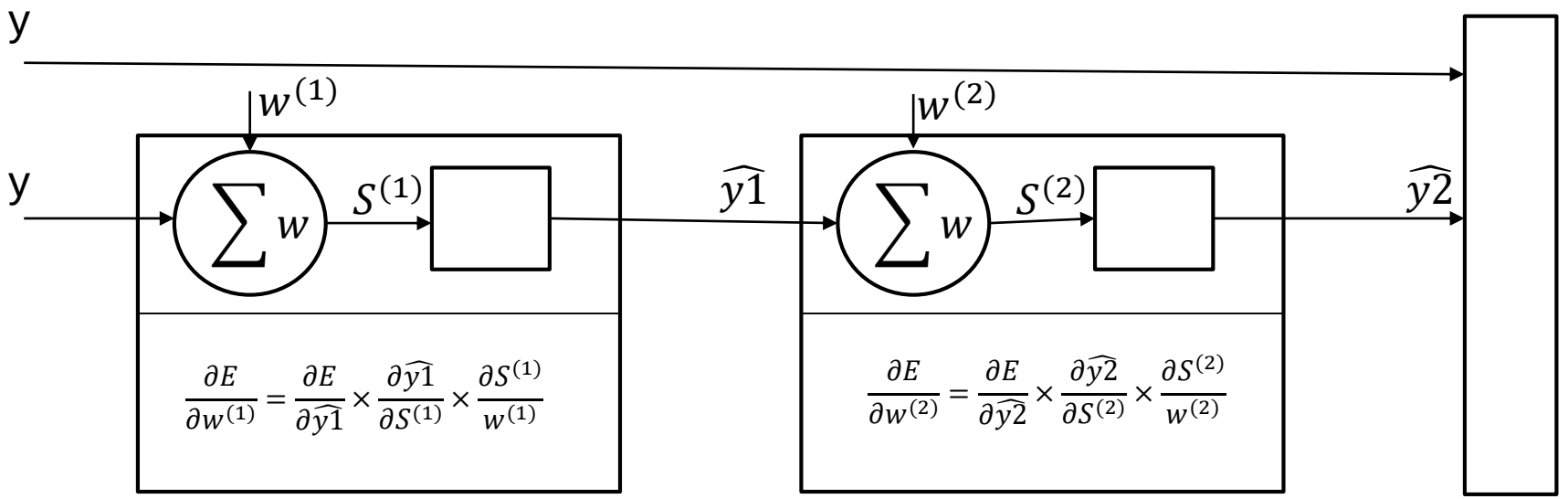
$$E=\frac{1}{2}(y-\widehat{y2})^2$$

$$\frac{\partial E}{\partial w^{(1)}} \qquad \frac{\partial E}{\partial w^{(2)}}$$

$$\alpha = learning\ rate$$

$$\text{New } w^{(1)} = w^{(1)} - \propto \times \frac{\partial E}{\partial w^{(1)}} \qquad \text{New } w^{(2)} = w^{(2)} - \propto \times \frac{\partial E}{\partial w^{(2)}}$$

Neural Network – Backpropagation



$$E=\frac{1}{2}(y-\widehat{y2})^2$$

$$\frac{\partial E}{\partial w^{(1)}} = \frac{\partial E}{\partial \widehat{y1}} \times \frac{\partial \widehat{y1}}{\partial S^{(1)}} \times \frac{\partial S^{(1)}}{w^{(1)}}$$

$$\frac{\partial E}{\partial w^{(1)}} = \frac{\partial E}{\partial \widehat{y2}} \times \frac{\partial \widehat{y2}}{\partial S^{(2)}} \times \frac{\partial S^{(2)}}{w^{(1)}} \times (\widehat{y1})(\widehat{y1}-1) \times y$$

$$\frac{\partial E}{\partial w^{(2)}} = \frac{\partial E}{\partial \widehat{y2}} \times \frac{\partial \widehat{y2}}{\partial S^{(2)}} \times \frac{\partial S^{(2)}}{w^{(2)}}$$

$$\frac{\partial E}{\partial w^{(2)}} = (y-\widehat{y2}) \times (\widehat{y2}) \times (\widehat{y2}-1) \times \widehat{y1}$$

Neural Network – Recap

Training a network is minimizing a cost function

Brute force is not possible

The intelligence lies in the weights/biases

The design of a deep neural network is a lot of trail and error

Google and Andrew Ng's lab at Stanford:
Their network comprised millions of neurons and 1 billion connection weights. They trained it on a dataset of 10 million 200x200 pixel RGB images to learn 20,000 object categories. The training simulation ran for three days on a cluster of 1,000 servers totaling 16,000 CPU cores.

Thank you.

Contact information:

Christoph Rhein
Dev Expert
SAP Labs, Washington DC
(202)312-3500

SAP Run Simple

Partner logo