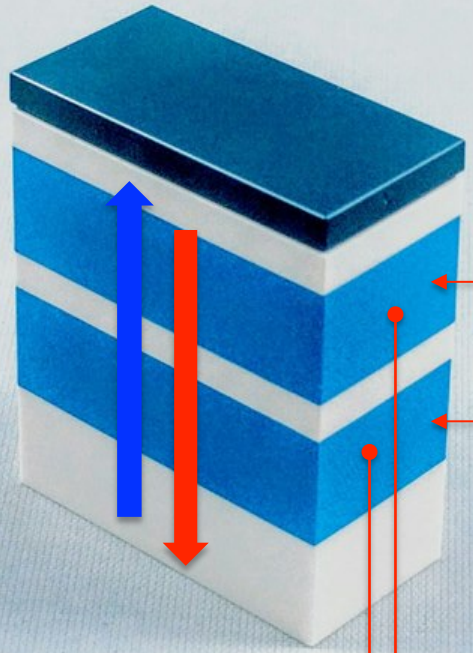
# Building your Neural Network from Scratch

Omar U. Florez
Research Scientist, Intel Labs (USA)
https://omar-florez.github.io/scratch_mlp/

$$h_2 = sigmoid(z_2)$$
$$z_2 = h_1 W_2$$
$$h_1 = sigmoid(z_1)$$
$$z_1 = X W_1$$

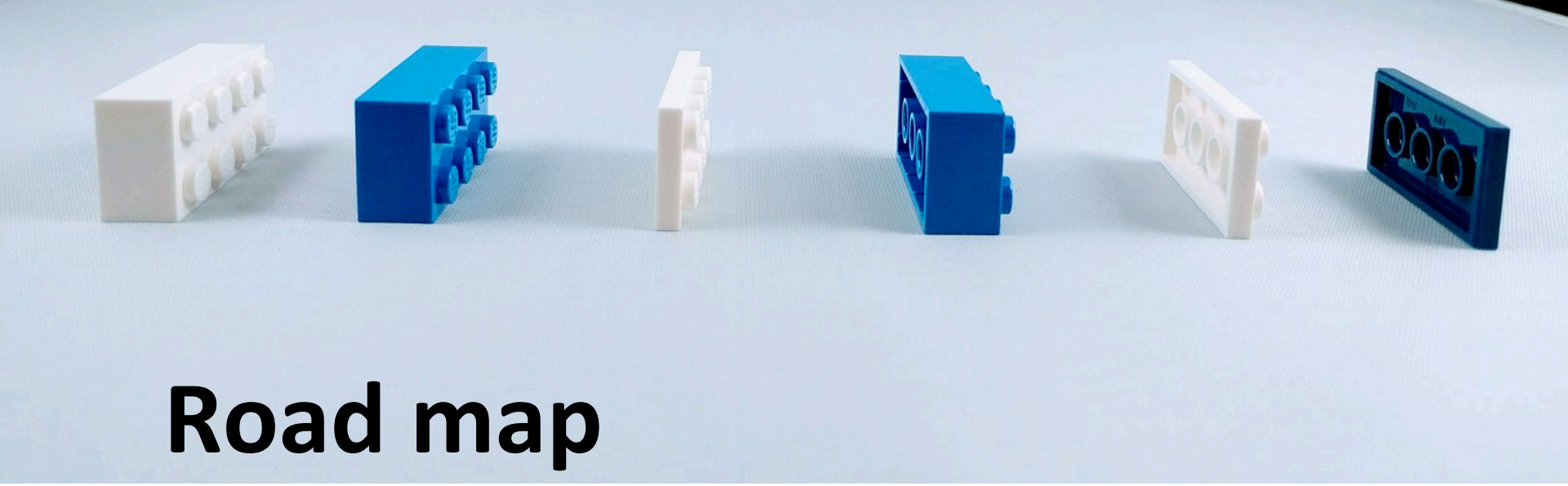


$$\frac{dLoss}{dW_2} = \frac{dLoss}{dh_2}\frac{dh_2}{dz_2}\frac{dz_2}{dW_2}$$

$$\frac{dLoss}{dW_1} = \frac{dLoss}{dh_1}\frac{dh_1}{dz_1}\frac{dz_1}{dW_1}$$

$$W_2 \mathrel{+}= -\alpha \frac{dLoss}{dW_2}$$

$$W_1 \mathrel{+}= -\alpha \frac{dLoss}{dW_1}$$



# Road map

1. Introduction
2. Error quantization (Loss function)
3. Backpropagation
    1. Forward step
    2. Backward step
4. Demo

https://omar-florez.github.io/scratch_mlp/

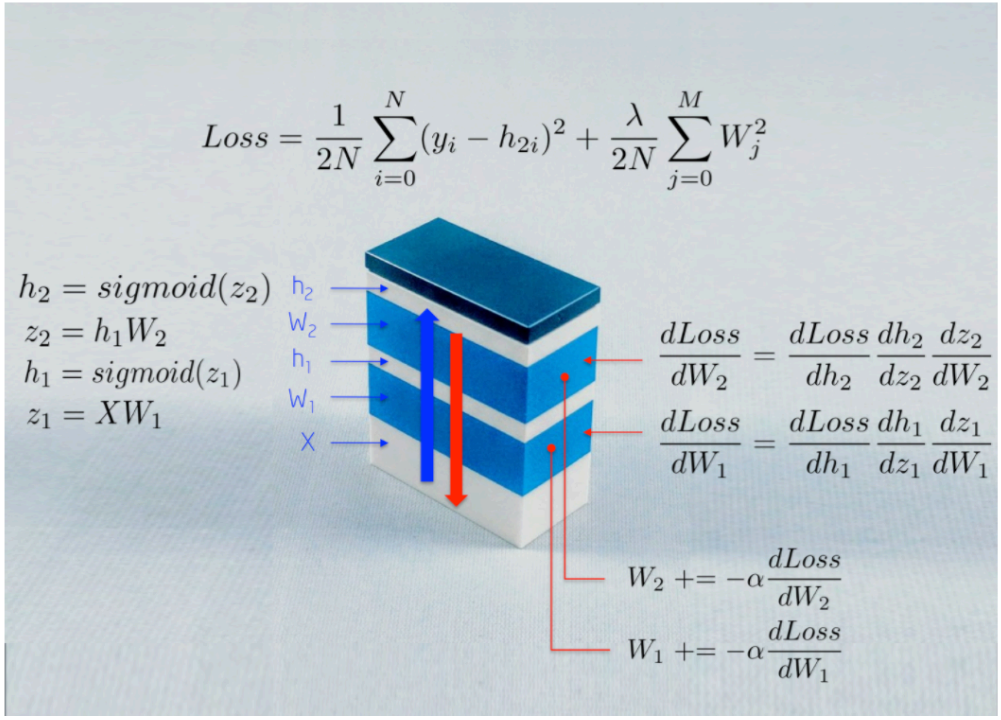
# One LEGO at a time: Explaining the Math of How Neural Networks Learn

> A **neural network** is a clever arrangement of linear and non-linear modules. When we choose and connect them wisely, we have a powerful tool to approximate any mathematical function. For example one that **separates classes with a non-linear decision boundary**.

A topic that is not always explained in depth, despite of its intuitive and modular nature, is the **backpropagation technique** responsible for updating trainable parameters. Let's build a neural network from scratch to see the internal functioning of a neural network using **LEGO pieces as a modular analogy**, one brick at a time.

Code implementing this can be found in this repository: https://github.com/omar-florez/scratch_mlp

## Neural Networks as a Composition of Pieces



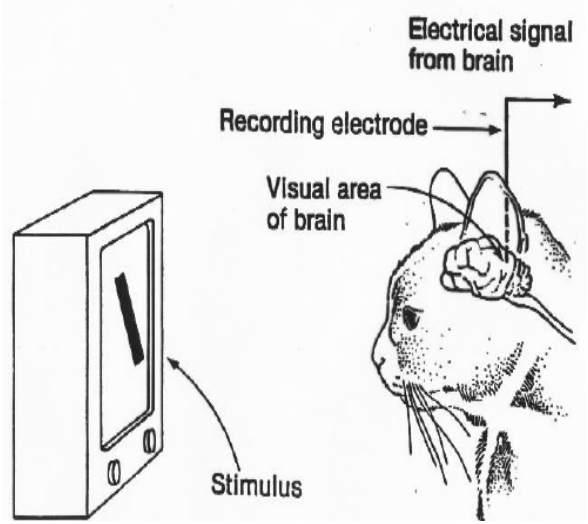
# Introduction

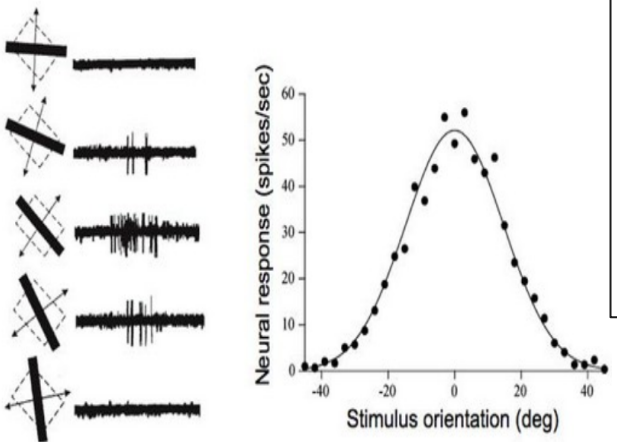
# Introduction

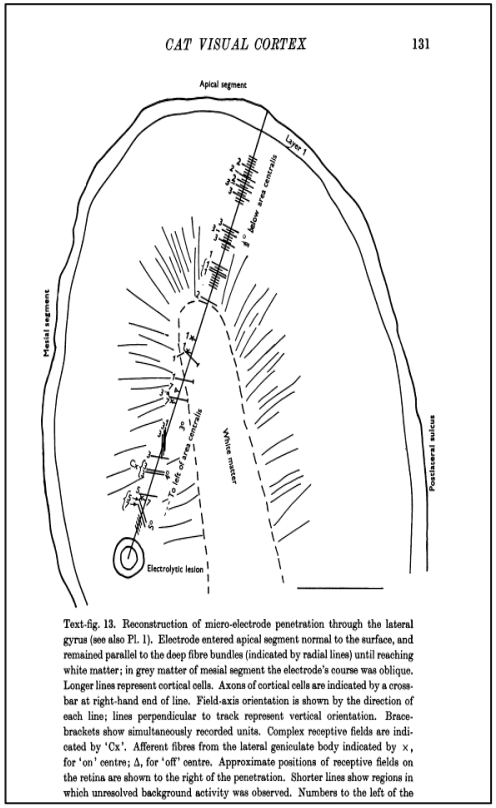
- Supervised learning and neural networks
  - What is an artificial neural network?
  - What is supervised learning?
  - Where is memory stored in neural network?
  - How does a neural network know it's working correctly?
  - What is the Math behind training a neural network?
  - Build our own neural network from scratch

# Learning to see

- The neural basis of Visual Perception (1959)
- **Nearby cells** in visual cortex corresponded to **nearby regions** in the **visual field** (topological mapping function)




Text-fig. 13. Reconstruction of micro-electrode penetration through the lateral gyrus (see also Pl. 1). Electrode entered apical segment normal to the surface, and remained parallel to the deep fibre bundles (indicated by radial lines) until reaching white matter; in grey matter of mesial segment the electrode's course was oblique. Longer lines represent cortical cells. Axons of cortical cells are indicated by a cross-bar at right-hand end of line. Field-axis orientation is shown by the direction of each line; lines perpendicular to track represent vertical orientation. Brace-brackets show simultaneously recorded units. Complex receptive fields are indicated by 'Cx'. Afferent fibres from the lateral geniculate body indicated by ×, for 'on' centre; Δ, for 'off' centre. Approximate positions of receptive fields on the retina are shown to the right of the penetration. Shorter lines show regions in which unresolved background activity was observed. Numbers to the left of the



J. Physiol. (1959) 148, 574–591

## RECEPTIVE FIELDS OF SINGLE NEURONES IN THE CAT'S STRIATE CORTEX

By D. H. HUBEL* and T. N. WIESEL*
*From the Wilmer Institute, The Johns Hopkins Hospital and University, Baltimore, Maryland, U.S.A.*

(*Received* 22 *April* 1959)

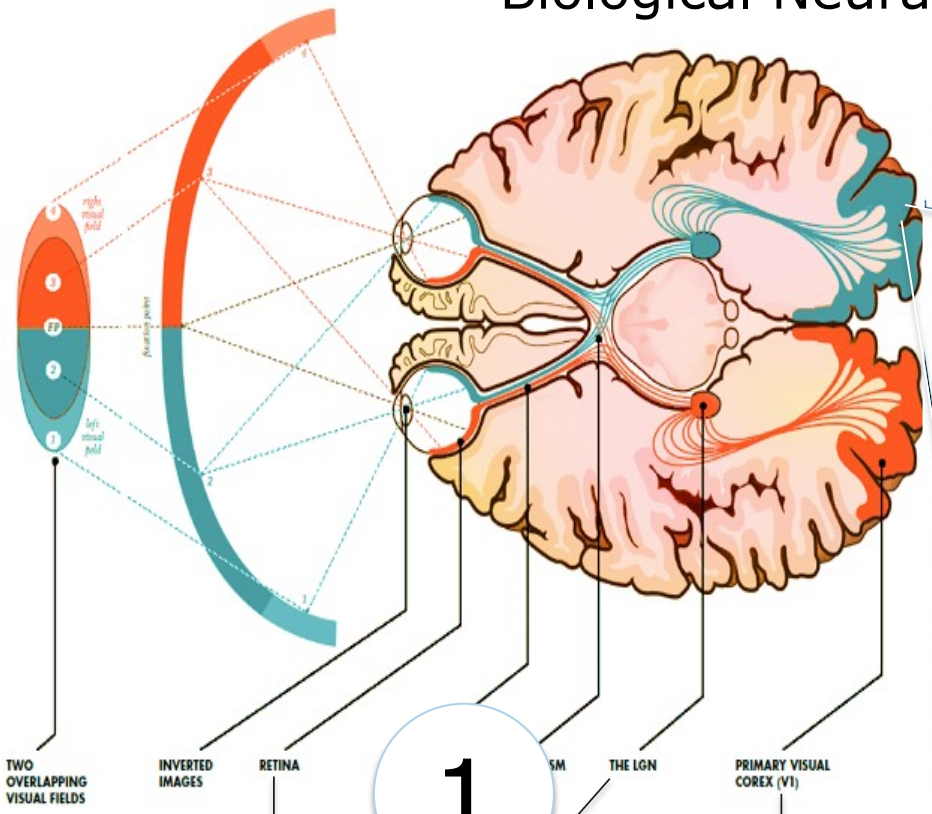
In the central nervous system the visual pathway from retina to striate cortex provides an opportunity to observe and compare single unit responses at several distinct levels. Patterns of light stimuli most effective in influencing units at one level may no longer be the most effective at the next. From differences in responses at successive stages in the pathway one may hope to gain some understanding of the part each stage plays in visual perception.

By shining small spots of light on the light-adapted cat retina Kuffler (1953) showed that ganglion cells have concentric receptive fields, with an 'on' centre and an 'off' periphery, or vice versa. The 'on' and 'off' areas within a receptive field were found to be mutually antagonistic, and a spot restricted to the centre of the field was more effective than one covering the whole receptive field (Barlow, FitzHugh & Kuffler, 1957). In the freely moving light-adapted cat it was found that the great majority of cortical cells studied gave little or no response to light stimuli covering most of the animal's visual field, whereas small spots shone in a restricted retinal region often evoked brisk responses (Hubel, 1959). A moving spot of light often produced stronger responses than a stationary one, and sometimes a moving spot gave more activation for one direction than for the opposite.

The present investigation, made in acute preparations, includes a study of receptive fields of cells in the cat's striate cortex. Receptive fields of the cells considered in this paper were divided into separate excitatory and inhibitory ('on' and 'off') areas. In this respect they resembled retinal ganglion-cell receptive fields. However, the shape and arrangement of excitatory and inhibitory areas differed strikingly from the concentric pattern found in retinal ganglion cells. An attempt was made to correlate responses to moving stimuli

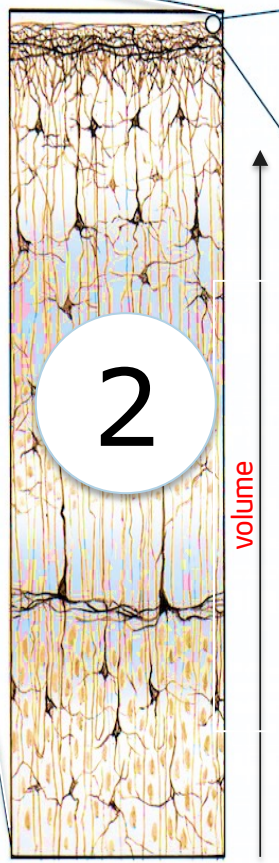
* Present address, Harvard Medical School, 25 Shattuck St., Boston 15, Massachusetts.

# Biological Neural Networks




1

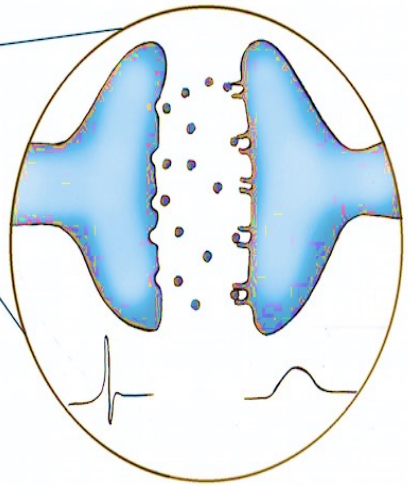
Visual cortex is divided into layers

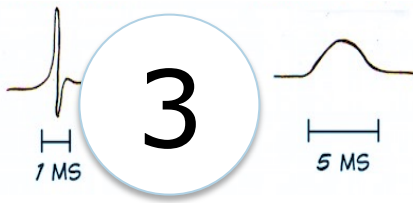


2

volume

THE UPPER LAYERS ARE VERY DENSE WITH SYNAPSES, WHERE EXCITATORY POST SYNAPTIC POTENTIALS CAN OCCUR.

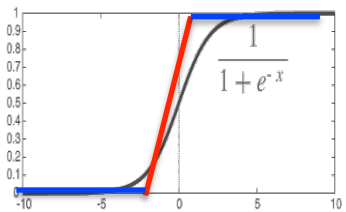





3

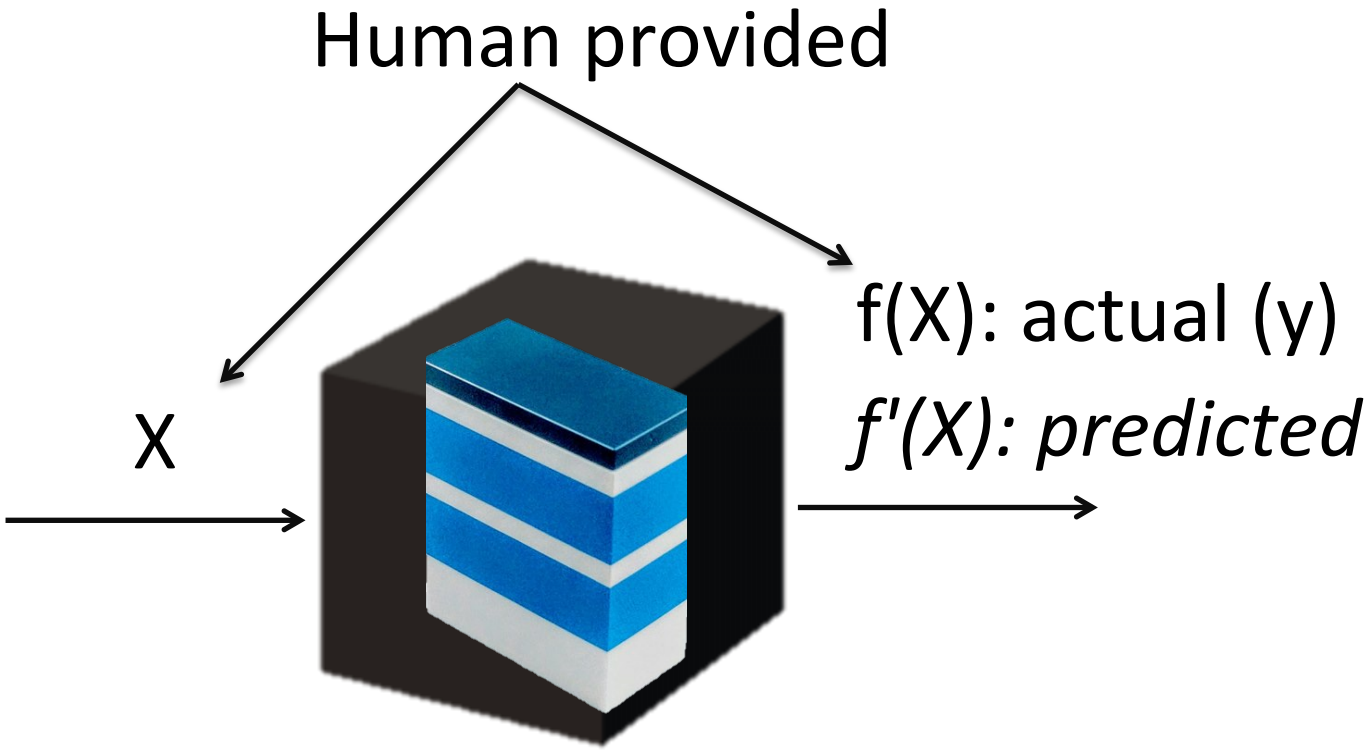
Simple Complex Simple Complex

Activation function (sigmoid)



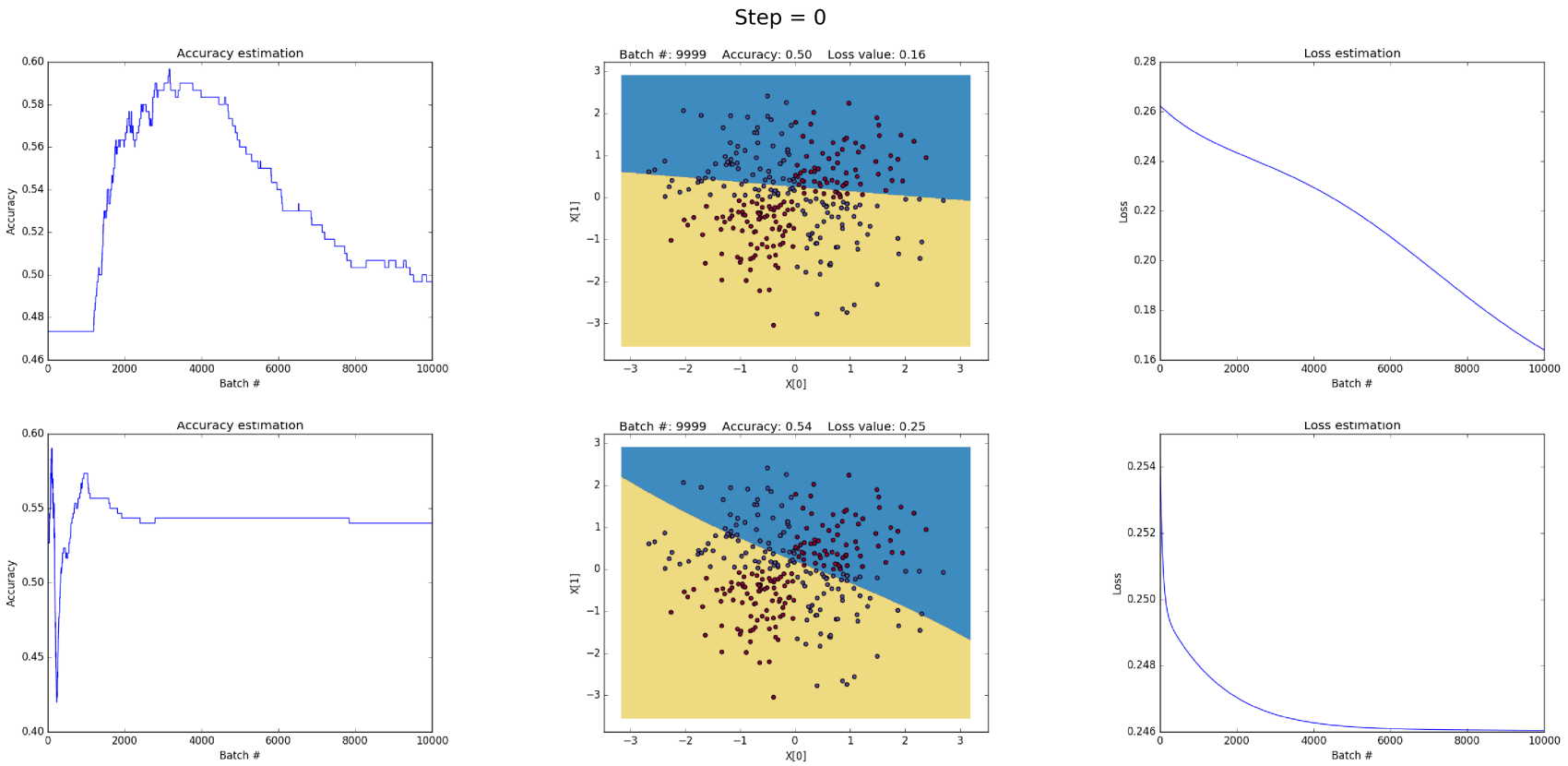
# Introduction

- Neural Network: y = f(x)



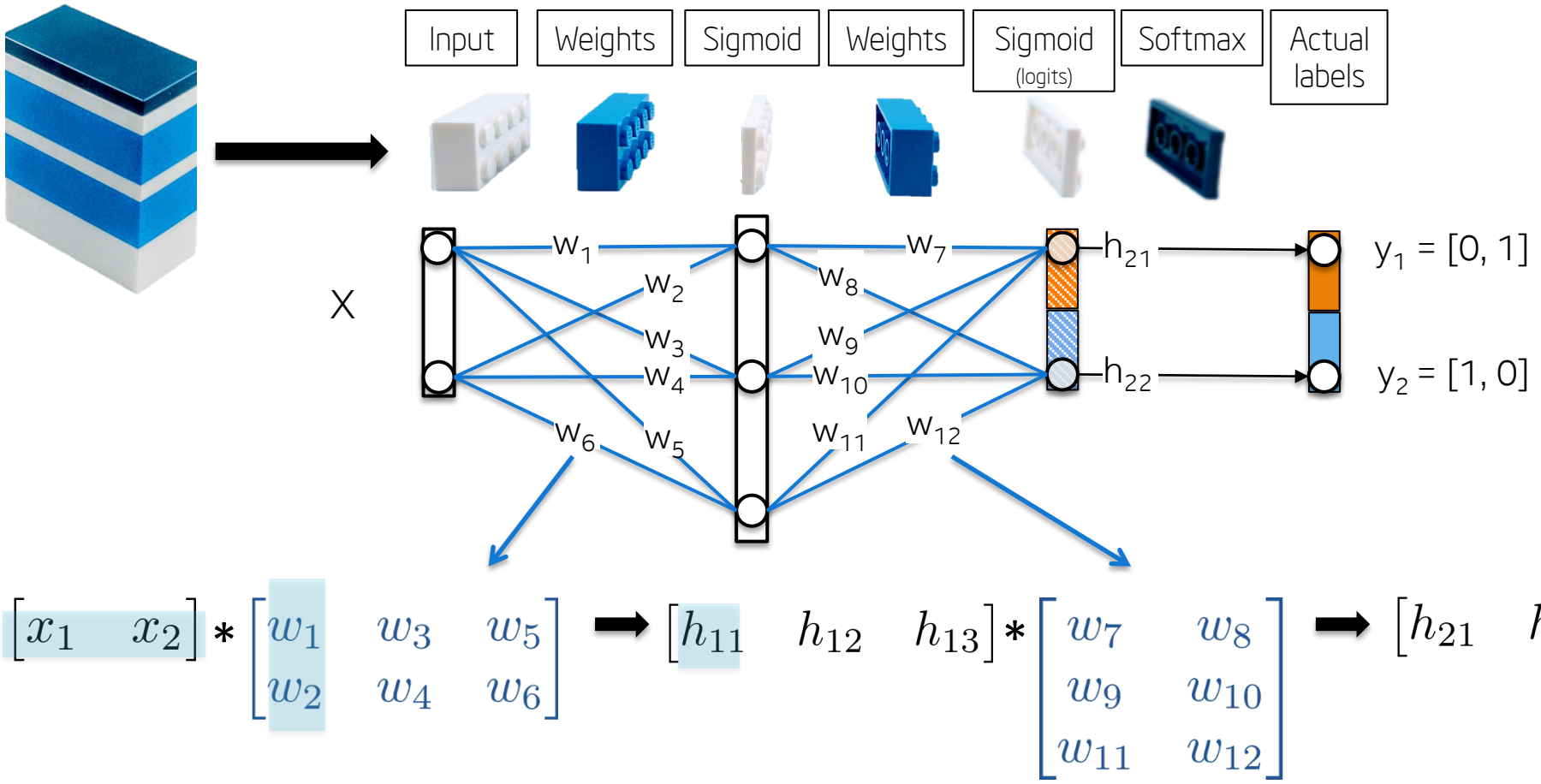
# Introduction

- A neural network is a clever arrangement of linear and non-linear **modules** that when we **connect** and **train** them wisely, it becomes a powerful tool to approximate any mathematical function that separates classes with a non-linear decision boundary



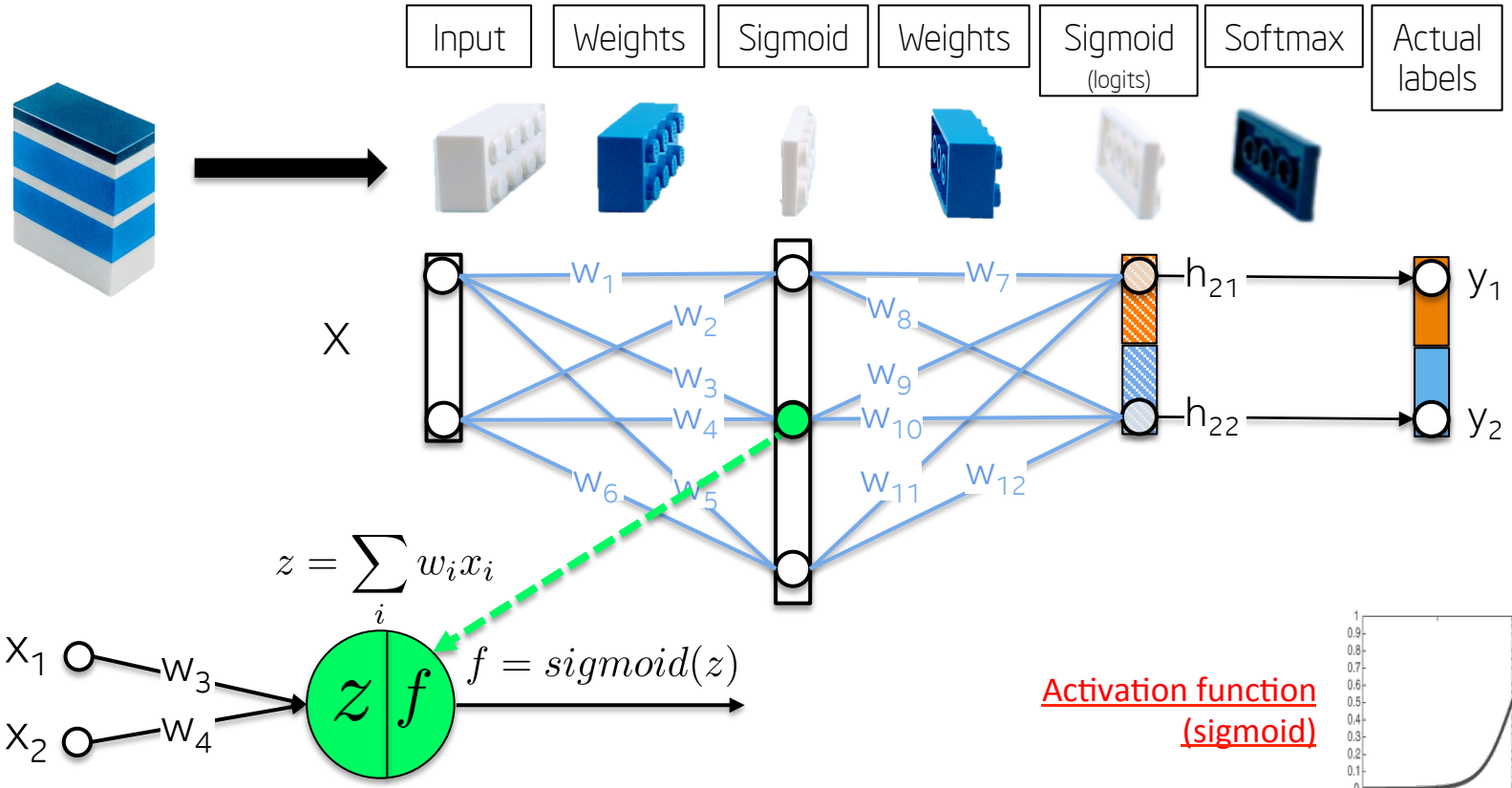
# Introduction

- Modules…
  - Each module is function with trainable parameters (weights)

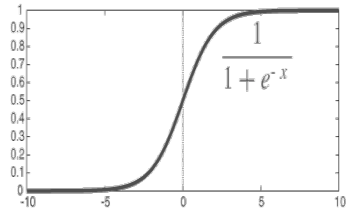


# Introduction

- Neural Networks
  - A neuron is a weighted linear combination of observations followed by an non-linear activation function

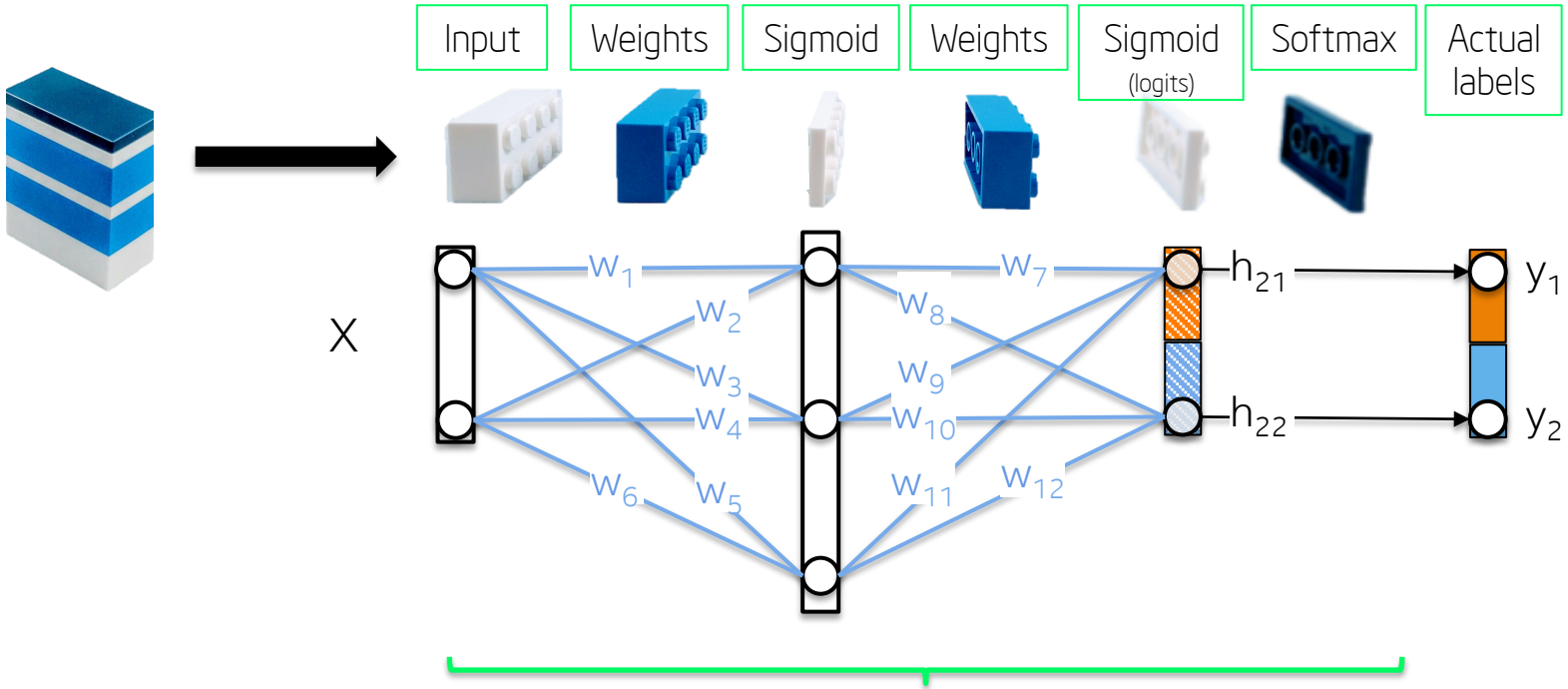



Activation function (sigmoid)



# Introduction

- Neural Networks
  - This is then a composition of functions




$$predicted = softmax(f(f(X, W_1), W_2))$$

# Recap

- We have learned:
  - Input is raw data stored in a matrix in which observations are rows and dimensions are columns
  - Weights map input to a different space just as linear kernels
  - To avoid numbers go out of range, we scale the values to 0-1 using a Sigmoid activation function inspired in neural synapsis
  - So far this operation is only a dot product and doesn't have the capacity to model non-linear interactions. This changes when we stack one hidden layer
  - A Softmax block converts log probabilities in the last layer into a muntinomial distribution of k states or neurons. This is known as inference
  - Next section we will see: how neural networks measure error during learning

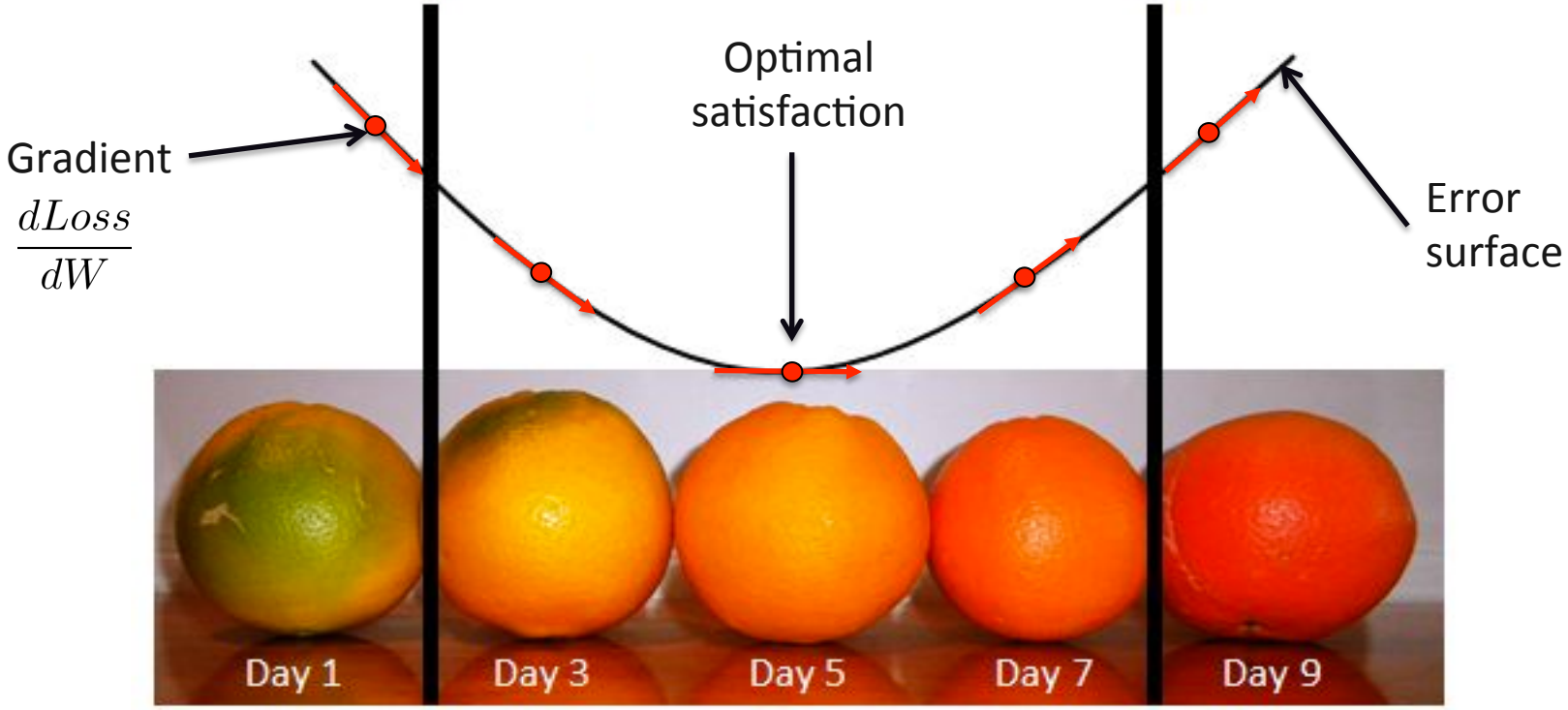
# Error Quantification (loss function)

# Loss function

Human provided

Inference Likelihood = Pr(class|trained network)

$$Loss = Error = \sum \frac{1}{2}(actual - predicted)^2$$

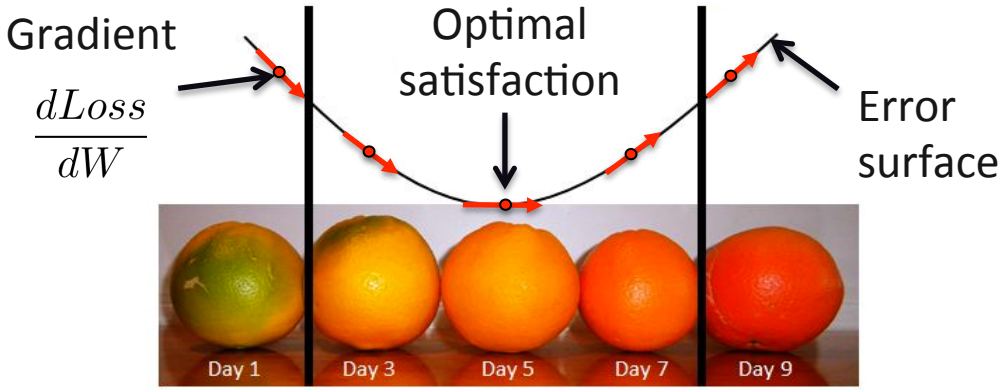


# Loss function

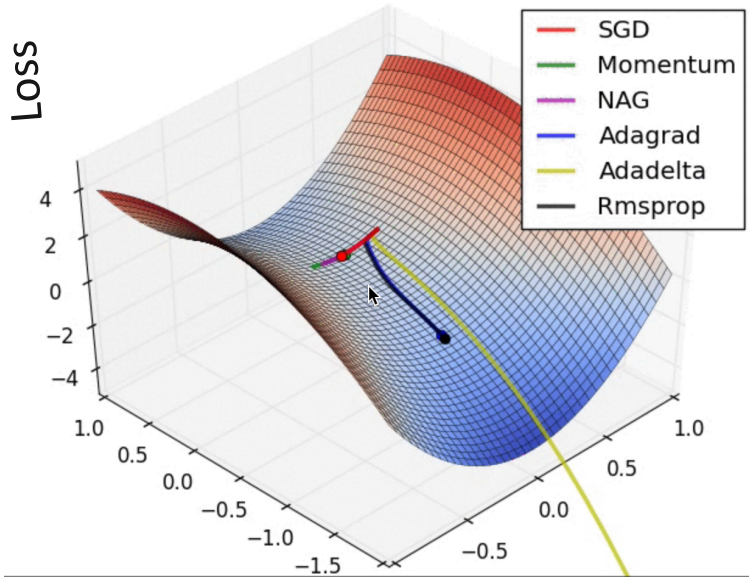
Human provided

Inference
Likelihood = Pr(class|trained network)

$$Loss = Error = \sum \frac{1}{2}(actual - predicted)^2$$




## Optimization Technique



# Recap

- We have learned:
  - A loss function is the way how a neural network knows whether it's working well or not
  - A gradient or first derivative of the loss function with respect to a trainable parameter will point in the direction of the greatest decrease of the loss function, that is minimizing it. And that is cool
  - In a convex function, we have a global minimum. Real life problem are rarely convex, so there are many comparable sub-optimal solutions. This is one of the reasons why a neural network takes so much time to train
  - There are multiple optimization methods including Stochastic Gradient Descend. Community is converging to use adaptive gradients techniques such as ADAM for training complex networks
  - Next section we will see: how neural networks learn also know as backpropagation

# Backpropagation algorithm

# Backpropagation

- We will talk important concepts in this section
  - How to update the weights of a neural network to minimize the loss function
  - Chain rule
  - Automatic differentiation
  - Math behind training

# Backpropagation

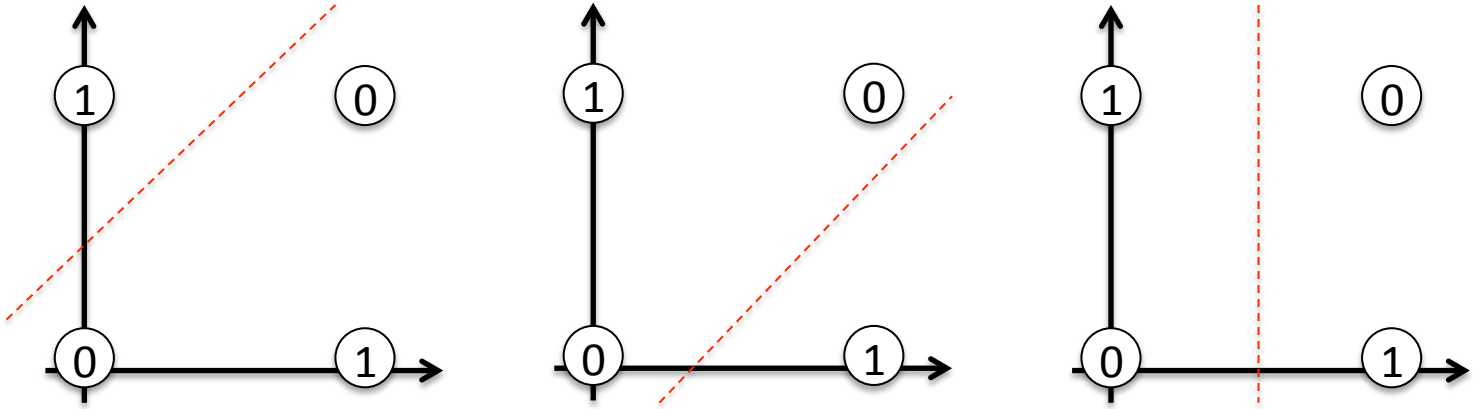
- Goal is to compute the first derivative of the loss function with respect to each of its trainable parameters (weights)
- Backprop is important because this technique is responsible for updating the weights of a neural network during a stage called training
- Automatic differentiation engines (PyTorch, Tensorflow, Caffe, etc.) hide low level implementations in favor of reducing bugs and enabling abstraction. But this could a trap as well

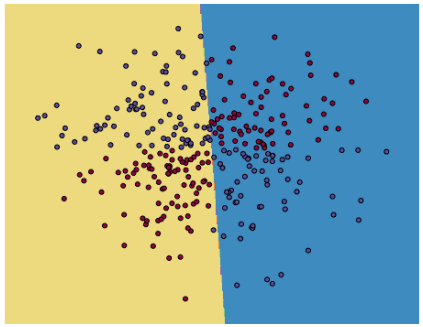
# Backpropagation

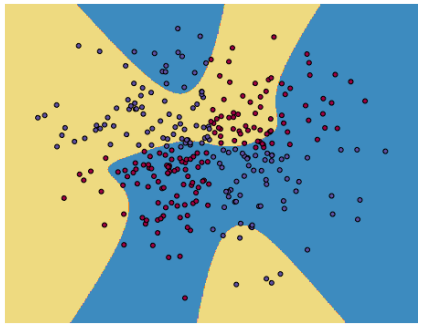
- Our neural network is intended to learn the XOR function. The election of this non-linear function is by no means random chance. Without backpropagation it would be hard to learn to separate classes with a straight line, a major class of problems that we find in real life.

XOR($x_1$>0, $x_2$>0) = Y

| $x_1$ | $x_2$ | Y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |



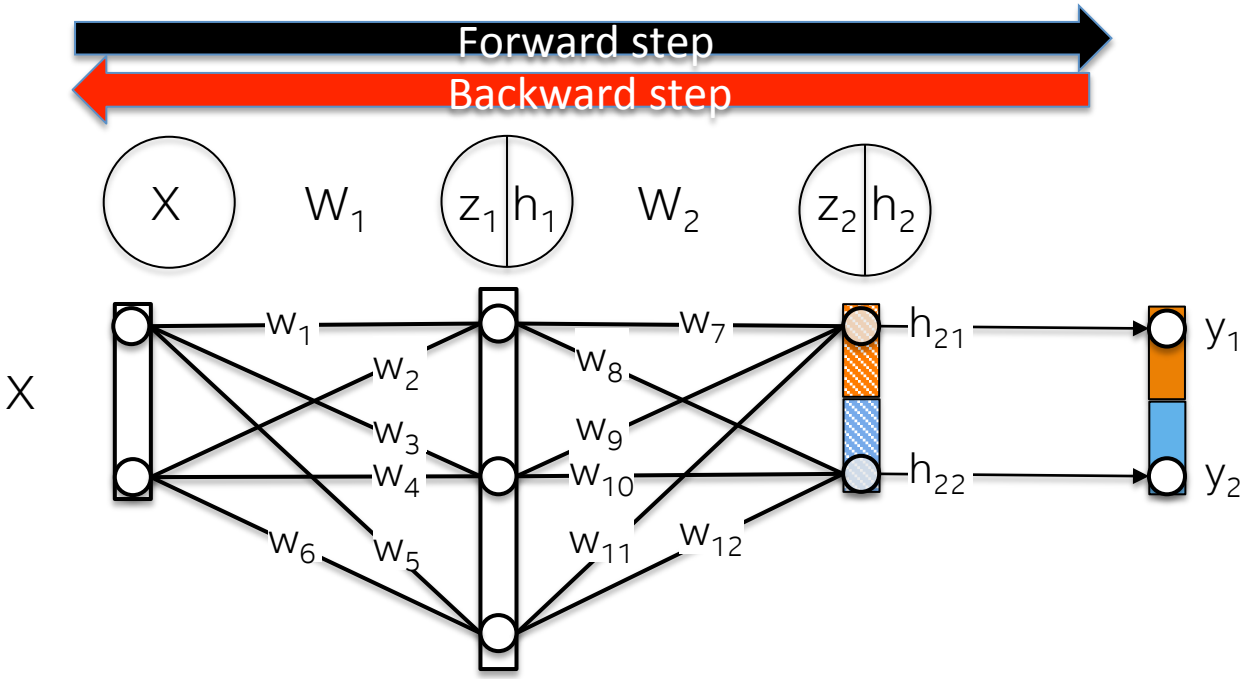





# Backpropagation

- Forward step:

$$z_1 = XW_1$$
$$h_1 = sigmoid(z_1)$$
$$z_2 = h_1 W_2$$
$$h_2 = sigmoid(z_2)$$



# Backpropagation

- Forward step:

$$z_1 = XW_1$$
$$h_1 = sigmoid(z_1)$$
$$z_2 = h_1 W_2$$
$$h_2 = sigmoid(z_2)$$

- Backward step:

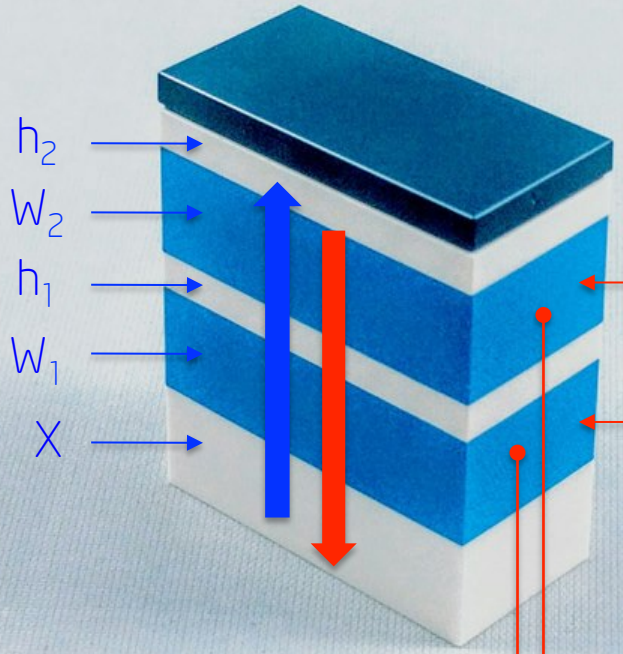
$$Loss = \frac{1}{2N}\sum_{i=0}^{N}(y_i - h_{2i})^2 + \frac{\lambda}{2N}\sum_{j=0}^{M} W_j^2$$

$$\frac{dLoss}{dW_2} = \frac{dLoss}{dh_2}\frac{dh_2}{dz_2}\frac{dz_2}{dW_2} \qquad W_2 \mathrel{+}= -\alpha\frac{dLoss}{dW_2}$$

$$\frac{dLoss}{dW_1} = \frac{dLoss}{dh_1}\frac{dh_1}{dz_1}\frac{dz_1}{dW_1} \qquad W_1 \mathrel{+}= -\alpha\frac{dLoss}{dW_1}$$

$$Loss = \frac{1}{2N}\sum_{i=0}^{N}(y_i - h_{2i})^2 + \frac{\lambda}{2N}\sum_{j=0}^{M} W_j^2$$

$$h_2 = sigmoid(z_2)$$
$$z_2 = h_1 W_2$$
$$h_1 = sigmoid(z_1)$$
$$z_1 = X W_1$$



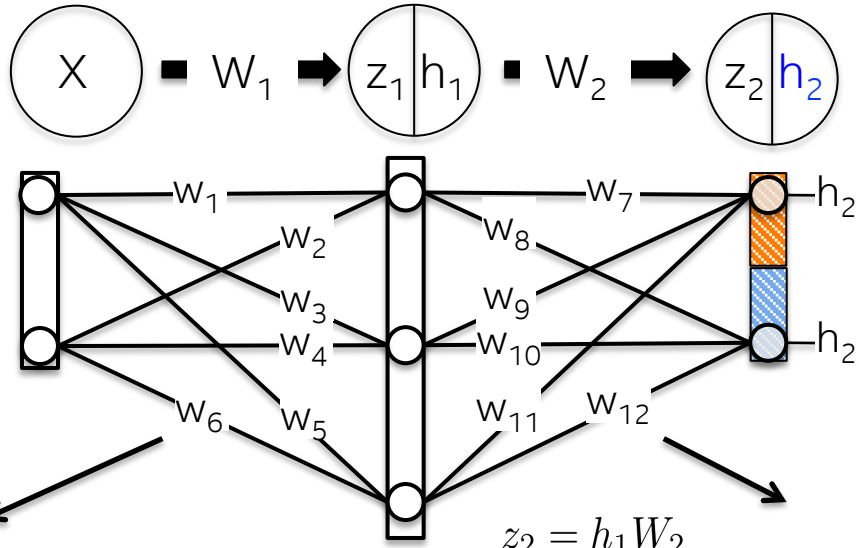

$$\frac{dLoss}{dW_2} = \frac{dLoss}{dh_2}\frac{dh_2}{dz_2}\frac{dz_2}{dW_2}$$

$$\frac{dLoss}{dW_1} = \frac{dLoss}{dh_1}\frac{dh_1}{dz_1}\frac{dz_1}{dW_1}$$

$$W_2 \mathrel{+}= -\alpha\frac{dLoss}{dW_2}$$

$$W_1 \mathrel{+}= -\alpha\frac{dLoss}{dW_1}$$

# Forward step: log probabilities $h_2$




$$X = [1.76405235 \quad 0.40015721]$$

$$Y = [1, 0]$$

$$W_1 = \begin{bmatrix} -0.67746455 & 0.37769425 & -0.07365253 \\ -0.53663519 & 0.60920496 & 0.21149622 \end{bmatrix}$$

$$z_2 = h_1 W_2$$

$$z_2 = [0.19626223 \quad 0.71301043 \quad 0.48867814] \begin{bmatrix} -0.41491732 & -0.30992816 \\ -0.31583913 & -0.27706468 \\ -0.44701662 & 0.15231059 \end{bmatrix}$$

$$z_2 = [-0.52507645 \quad -0.18394635]$$

$$z_1 = XW_1$$

$$z_1 = [1.76405235 \quad 0.40015721] \begin{bmatrix} -0.67746455 & 0.37769425 & -0.07365253 \\ -0.53663519 & 0.60920496 & 0.21149622 \end{bmatrix}$$

$$z_1 = [-1.40982136 \quad 0.91005018 \quad -0.04529517]$$

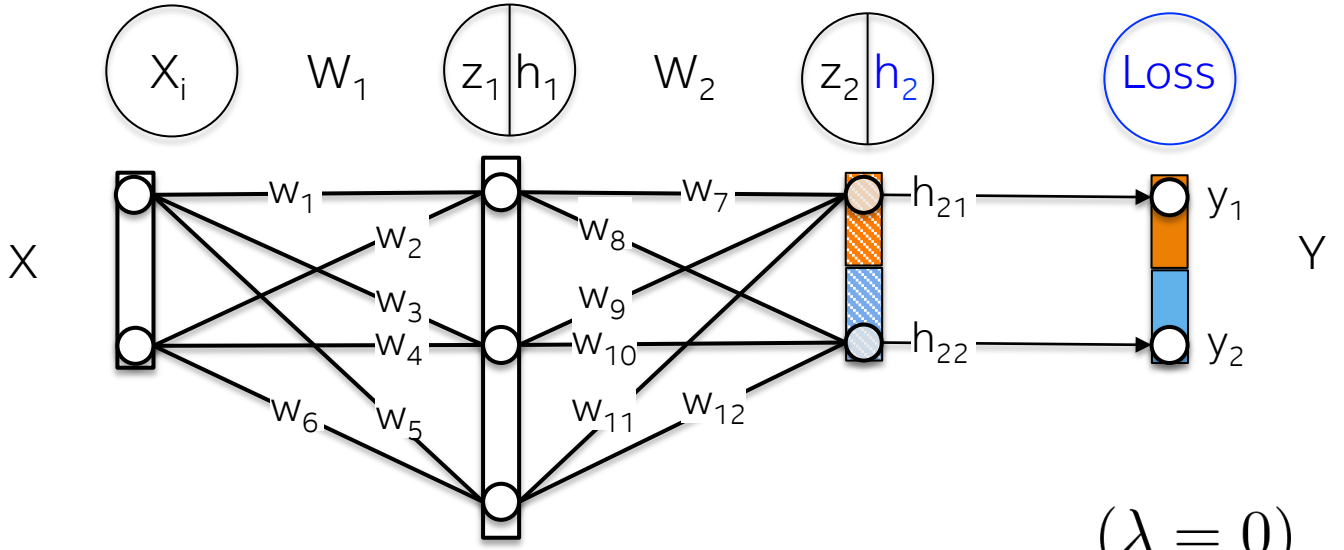
$$h_1 = sigmoid(z_1)$$

$$h_1 = [0.19626223 \quad 0.71301043 \quad 0.48867814]$$

$$h_2 = sigmoid(z_2)$$

$$h_2 = [0.37166596 \quad 0.45414264]$$

# Backward step: loss function




$$z_1 = XW_1$$

$$h_1 = sigmoid(z_1)$$

$$z_2 = h_1 W_2$$

$$h_2 = sigmoid(z_2)$$

$$Loss = \frac{1}{2N}\sum_{i=0}^{N}(y_i - h_{2i})^2$$

$$(\lambda = 0)$$

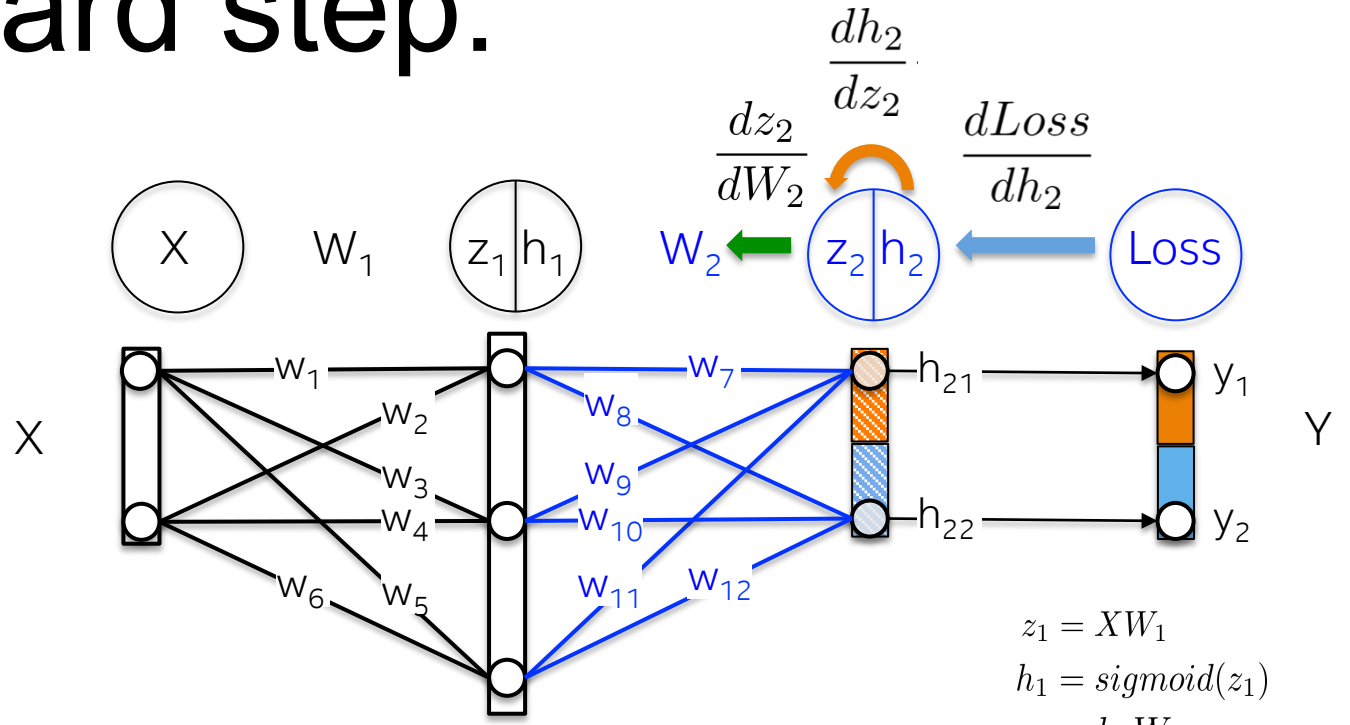
$$Loss = \frac{1}{2N}\sum_{i=0}^{N}(y - h_2)^2 + \frac{\lambda}{2N}\sum_{j=0}^{M} W_j^2$$

$$h_2 = \log p(x)$$

Label human provided

# Backward step:

$$\frac{dLoss}{dW_2}$$




$$z_1 = XW_1$$
$$h_1 = sigmoid(z_1)$$
$$z_2 = h_1 W_2$$
$$h_2 = sigmoid(z_2)$$
$$Loss = \frac{1}{2N}\sum_{i=0}^{N}(y_i - h_{2i})^2$$

Chain rule:

$$\frac{dLoss}{dW_2} = \frac{dLoss}{dh_2}\frac{dh_2}{dz_2}\frac{dz_2}{dW_2}$$

$$\frac{dLoss}{dh_2} = -(y - h_2) \qquad \frac{dh_2}{dz_2} = h_2(1 - h_2) \qquad \frac{dz_2}{dW_2} = h_1$$

- Visually, you can see below the components needed to compute derivative of the loss with respect to the weighs W2. Let's assume the regularization term is zero to simplify the equations. First, the derivative of the loss with respect to the values in the last layer of the network h2.

$$\frac{dLoss}{dW_2} = \frac{dLoss}{dh_2}\frac{dh_2}{dz_2}\frac{dz_2}{dW_2}$$

$$\frac{dLoss}{dh_2} = -(y - h_2) \qquad \frac{dh_2}{dz_2} = h_2(1 - h_2) \qquad \frac{dz_2}{dW_2} = h_1$$

$$\begin{aligned}
\frac{dLoss}{dh_2} &= -(y - h_2) \\
&= -(\begin{bmatrix}1 & 0\end{bmatrix} - \begin{bmatrix}0.37166596 & 0.45414264\end{bmatrix}) \\
&= \begin{bmatrix}-0.62833404 & 0.45414264\end{bmatrix}
\end{aligned}$$

$$\begin{aligned}
\frac{dh_2}{dz_2} &= h_2(1 - h_2) \\
&= \begin{bmatrix}0.37166596 & 0.454142644\end{bmatrix}(1 - \begin{bmatrix}0.37166596 & 0.454142644\end{bmatrix}) \\
&= \begin{bmatrix}-0.23353037 & 0.2478971\end{bmatrix}
\end{aligned}$$

$$\begin{aligned}
\frac{dz_2}{dW_2} &= h_1 \\
&= \begin{bmatrix}0.19626223 & 0.71301043 & 0.48867814\end{bmatrix}
\end{aligned}$$

- Visually, you can see below the components needed to compute derivative of the loss with respect to the weighs W2. Let's assume the regularization term is zero to simplify the equations. First, the derivative of the loss with respect to the values in the last layer of the network h2.

$$\frac{dLoss}{dW_2} = \frac{dLoss}{dh_2}\frac{dh_2}{dz_2}\frac{dz_2}{dW_2}$$

$$\frac{dLoss}{dh_2} = -(y - h_2) \qquad \frac{dh_2}{dz_2} = h_2(1 - h_2) \qquad \frac{dz_2}{dW_2} = h_1$$

$$\frac{dLoss}{dW_2} = \begin{bmatrix} 0.19626223 & 0.71301043 & 0.48867814 \end{bmatrix}^T \begin{bmatrix} -0.23353037 & 0.2478971 \end{bmatrix} \begin{bmatrix} -0.62833404 & 0.45414264 \end{bmatrix}$$

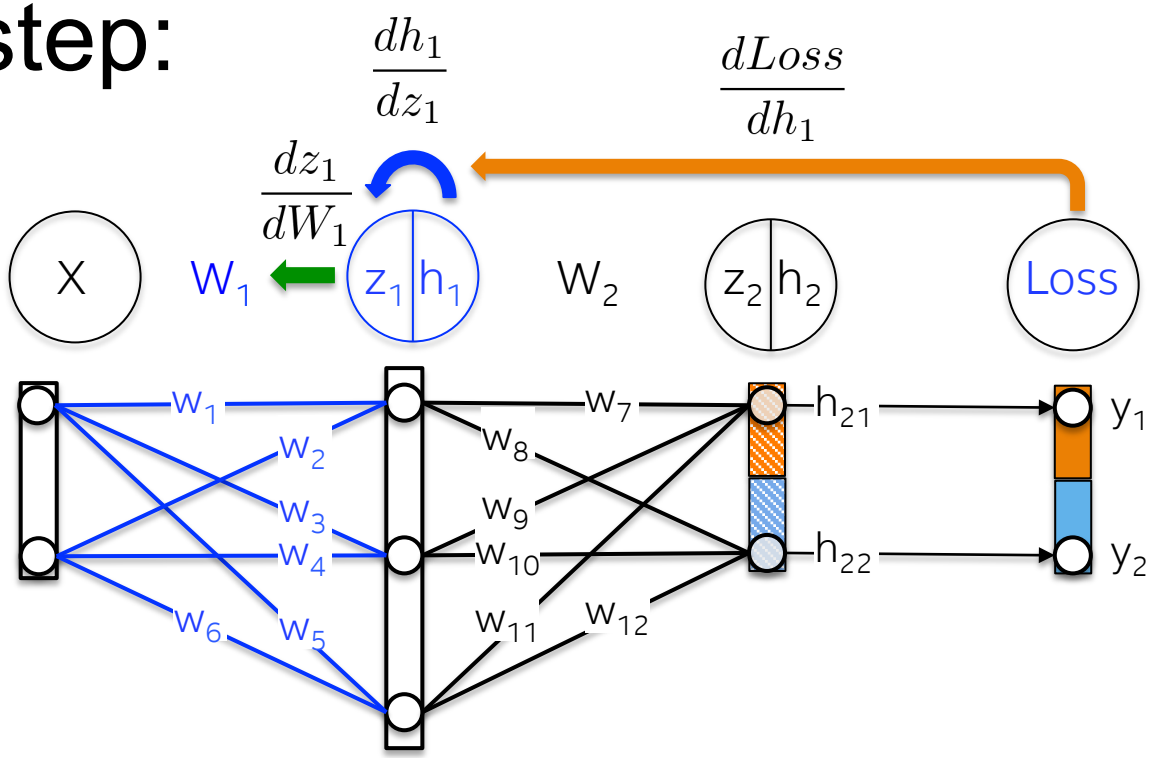
$$= \begin{bmatrix} -0.02879856 & 0.02209533 \\ -0.10462365 & 0.08027117 \\ -0.07170623 & 0.0550157 \end{bmatrix}$$

$$W_2^* = W_2 - \alpha \frac{dLoss}{dW_2} = \begin{bmatrix} -0.41491732 & -0.30992816 \\ -0.31583913 & -0.27706468 \\ -0.44701662 & 0.15231059 \end{bmatrix} - 0.001 \begin{bmatrix} -0.02879856 & 0.02209533 \\ -0.10462365 & 0.08027117 \\ -0.07170623 & 0.0550157 \end{bmatrix}$$

$$W_2^* = \begin{bmatrix} -0.41488852 & -0.30995026 \\ -0.31573451 & -0.27714495 \\ -0.44694491 & 0.15225557 \end{bmatrix}$$

# Backward step:

$$\frac{dLoss}{dW_1}$$




# Chain rule:

$$\frac{dLoss}{dW_1} = \frac{dLoss}{dh_1}\frac{dh_1}{dz_1}\frac{dz_1}{dW_1} \qquad \frac{dLoss}{dW_2} = \frac{dLoss}{dh_2}\frac{dh_2}{dz_2}\frac{dz_2}{dW_2}$$

$$\frac{dLoss}{dh_1} = \frac{dLoss}{dz2}\frac{dz2}{dh1}$$

- We can reuse components and gradients from previous computations given Neural Networks compositionality

$$\frac{dLoss}{dW_1} = \frac{dLoss}{dh_1}\frac{dh_1}{dz_1}\frac{dz_1}{dW_1} \qquad \frac{dLoss}{dW_2} = \frac{dLoss}{dh_2}\frac{dh_2}{dz_2}\frac{dz_2}{dW_2}$$

$$\frac{dLoss}{dh_1} = \frac{dLoss}{dz2}\frac{dz2}{dh1}$$

$$\frac{dLoss}{dh_1} = \frac{dLoss}{dz2}\frac{dz2}{dh1}$$

$$= [-(y - h_2)h_2(1 - h_2)][W_2]$$

$$= \begin{bmatrix} 0.23353037 & 0.2478971 \end{bmatrix} \begin{bmatrix} -0.41491732 & -0.30992816 \\ -0.31583913 & -0.27706468 \\ -0.44701662 & 0.15231059 \end{bmatrix}$$

$$= \begin{bmatrix} 0.02599101 & 0.01515256 & 0.08274025 \end{bmatrix}$$

$$\frac{dh_1}{dz_1} = h_1(1 - h_1)$$

$$= \begin{bmatrix} 0.19626223 & 0.71301043 & 0.48867814 \end{bmatrix} (1 - \begin{bmatrix} 0.19626223 & 0.71301043 & 0.48867814 \end{bmatrix}$$

$$= \begin{bmatrix} 0.15774337 & 0.20462656 & 0.24987182 \end{bmatrix}$$

$$\frac{dz_1}{dW_1} = X = \begin{bmatrix} 1.76405235 & 0.40015721 \end{bmatrix}$$

- Visually, you can see below the components needed to compute derivative of the loss with respect to the weighs W2. Let's assume the regularization term is zero to simplify the equations. First, the derivative of the loss with respect to the values in the last layer of the network h2.

$$\frac{dLoss}{dW_1} = \frac{dLoss}{dh_1}\frac{dh_1}{dz_1}\frac{dz_1}{dW_1}$$

$$\frac{dLoss}{dW_1} = \begin{bmatrix}1.76405235 & 0.40015721\end{bmatrix}^T \begin{bmatrix}0.02599101 & 0.01515256 & 0.08274025\end{bmatrix} \begin{bmatrix}0.15774337 & 0.20462656 & 0.24987182\end{bmatrix}$$

$$= \begin{bmatrix}0.00723246 & 0.00546965 & 0.03647082 \\ 0.00164061 & 0.00124073 & 0.00827303\end{bmatrix}$$

$$W_1^* = W_1 - \alpha\frac{dLoss}{dW_1} = \begin{bmatrix}-0.67746455 & 0.37769425 & -0.07365253 \\ -0.53663519 & 0.60920496 & 0.21149622\end{bmatrix} - 0.001\begin{bmatrix}0.00723246 & 0.00546965 & 0.03647082 \\ 0.00164061 & 0.00124073 & 0.00827303\end{bmatrix}$$

$$W_1^* = \begin{bmatrix}-0.67747178 & 0.37768878 & -0.073689 \\ -0.53663683 & 0.60920372 & 0.21148795\end{bmatrix}$$

- Finally, we are ready to update the weights W2 and W1 and complete one iteration during the training of the neural network:

$$W2 \mathrel{-}= \alpha \frac{dLoss}{dW2}$$

$$W1 \mathrel{-}= \alpha \frac{dLoss}{dW1}$$

# Recap

- We have learned:
  - A lot
  - The backpropagation algorithm is responsible for updating the weights of a neural network
  - This is a recursive algorithm, which can reuse previous computations of gradient and heavily relies on differentiable functions
  - Since these updates reduce the loss function, a network is 'learning' to react this way when seeing similar new patterns. A property called generalization
  - Next section we will see: a demo. I will walk you through code to see how a neural network is built from scratch including backprop using only numpy

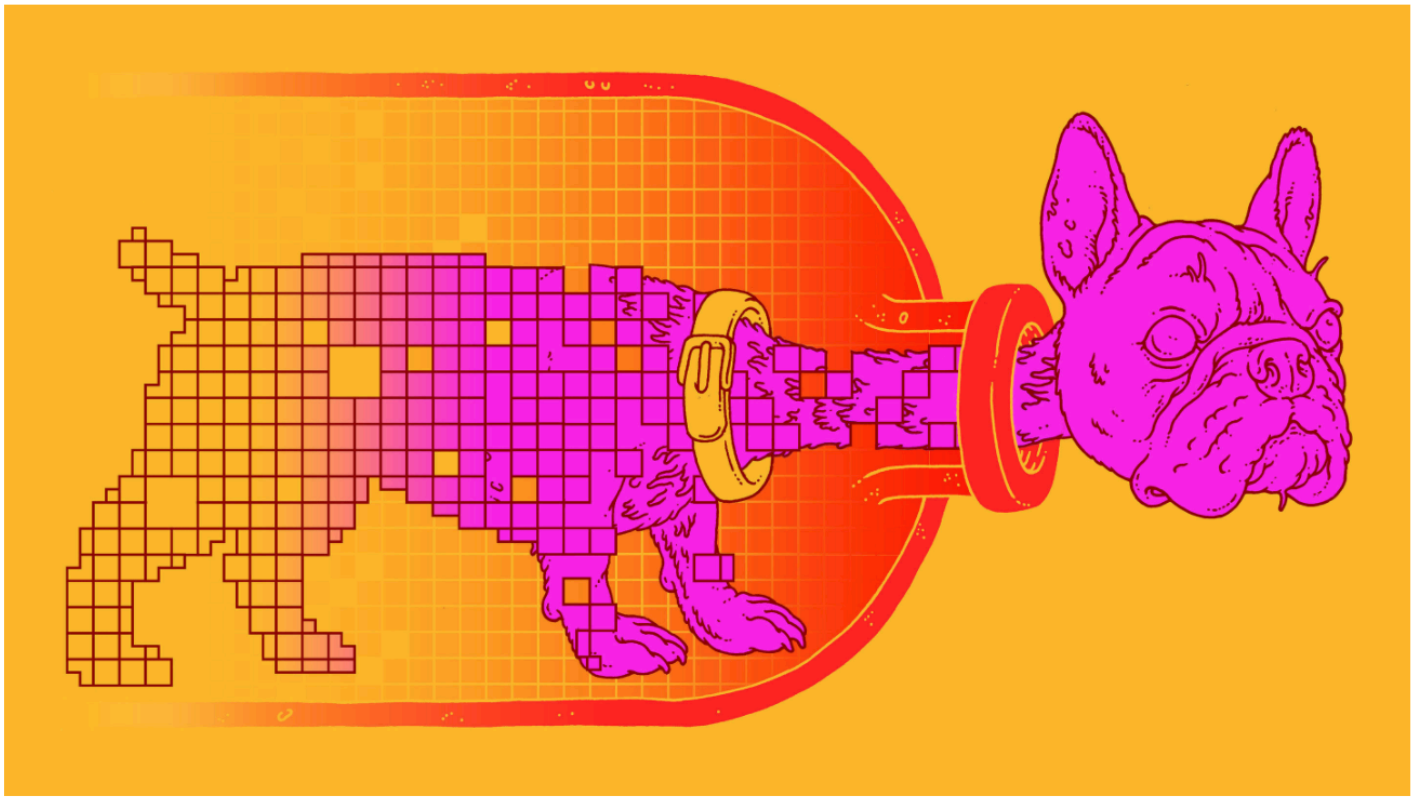
# Demo

SERIES WIRED TO LEARN: THE NEXT AI

# New Theory Cracks Open the Black Box of Deep Learning

29

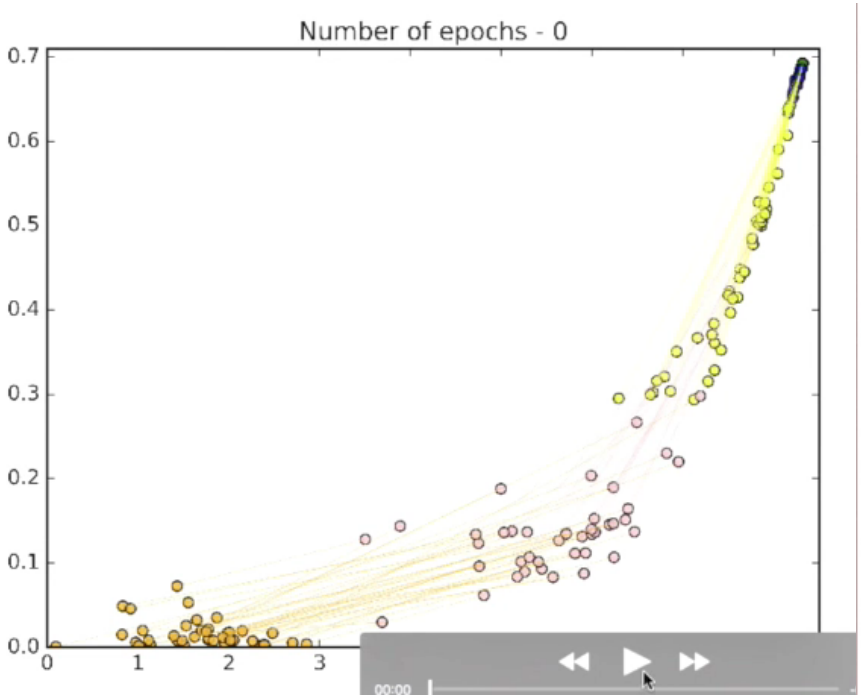
*A new idea called the "information bottleneck" is helping to explain the puzzling success of today's artificial-intelligence algorithms — and might also explain how human brains learn.*

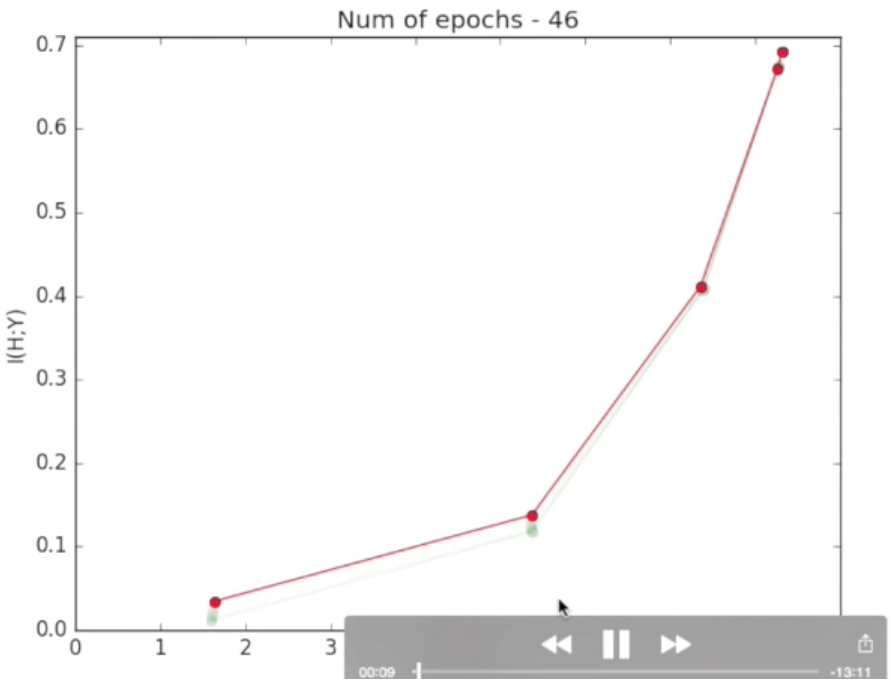


Eric Nyquist for Quanta Magazine

Even as machines known as "deep neural networks" have learned to converse, drive cars, beat video games and Go champions, dream, paint pictures and help make

Share this article


Number of epochs - 0
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0
1
2
3
00:00


Num of epochs - 46
I(H;Y)
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0
1
2
3
00:09
-13:11