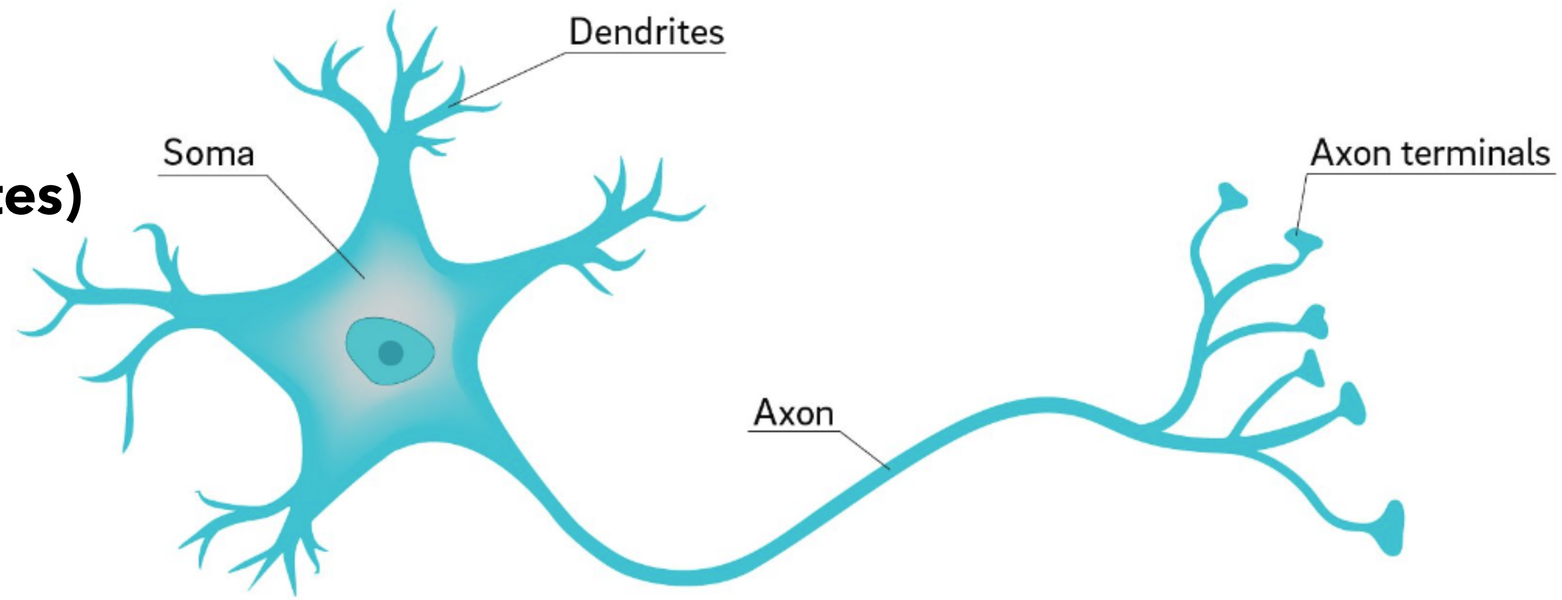
FULLY CONNECTED NEURAL NETWORKS

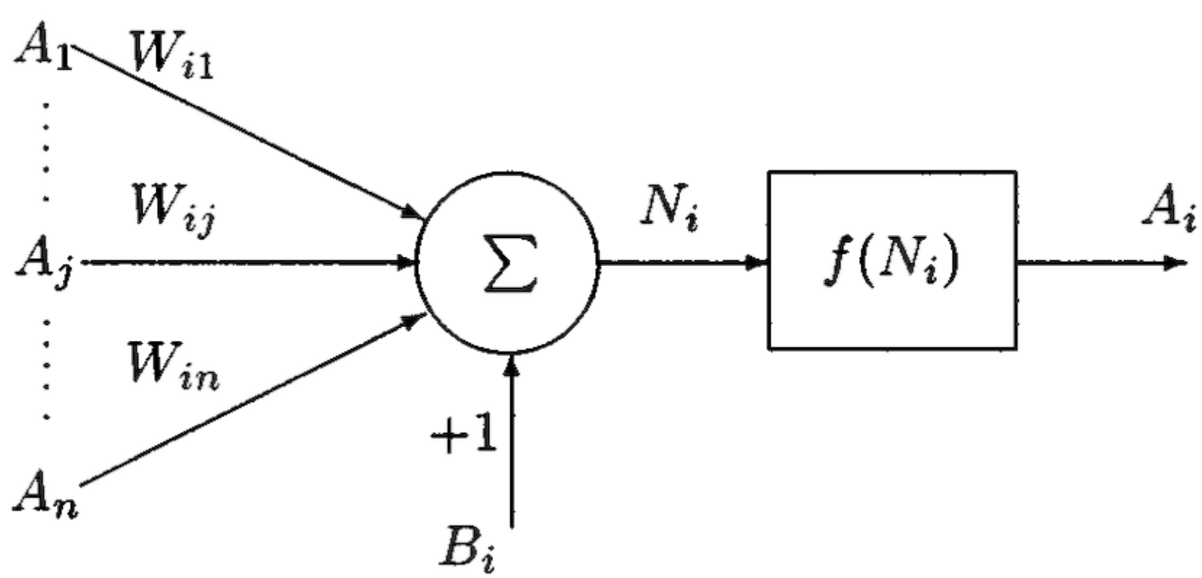
DEEPLEA17EM

MOTIVATION - A NEURON

- **Biology:**
  - **input from other neurons (dentrites)**
  - **summing them (soma)**
  - **based on the inputs firing**
  - **output to the axon**
- **Model:**
  - **inputs (x)**
  - **weighted sum of inputs + bias (b)**
  - **activation function (g)**
  - **output/activation: a = g(w*x + b)**



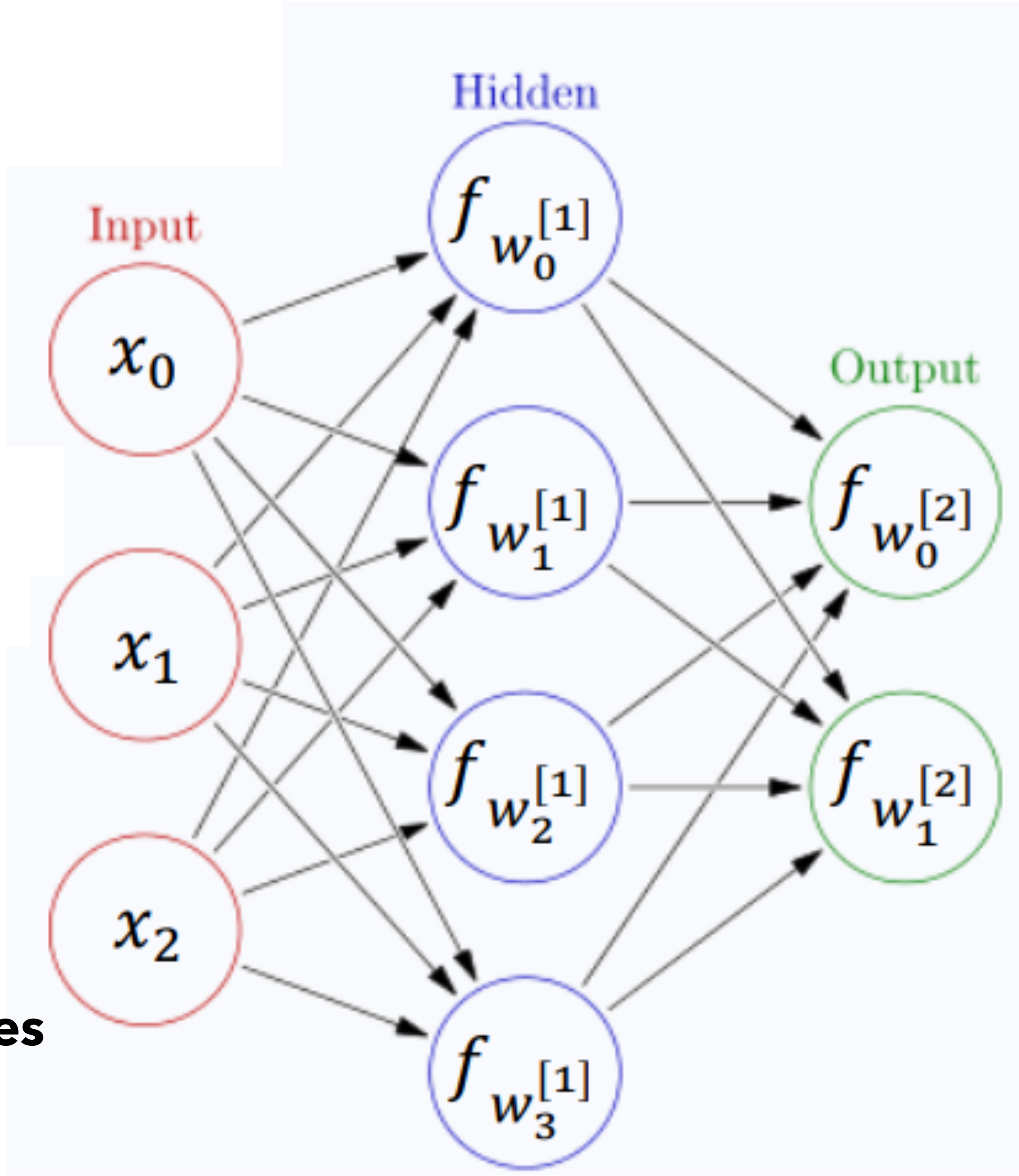
Credit: David Baillot/ UC San Diego



Yang et el, 2000. Constraint Satisfaction Adaptive Neural Network and Heuristics Combined Approaches for Generalized Job-Shop Scheduling

ONE WON'T BE ENOUGH... NEURAL NETWORK

- **fully connected neural network with one hidden layer**
- **input: the data itself**
- **output: the prediction**
- **hidden: everything between**



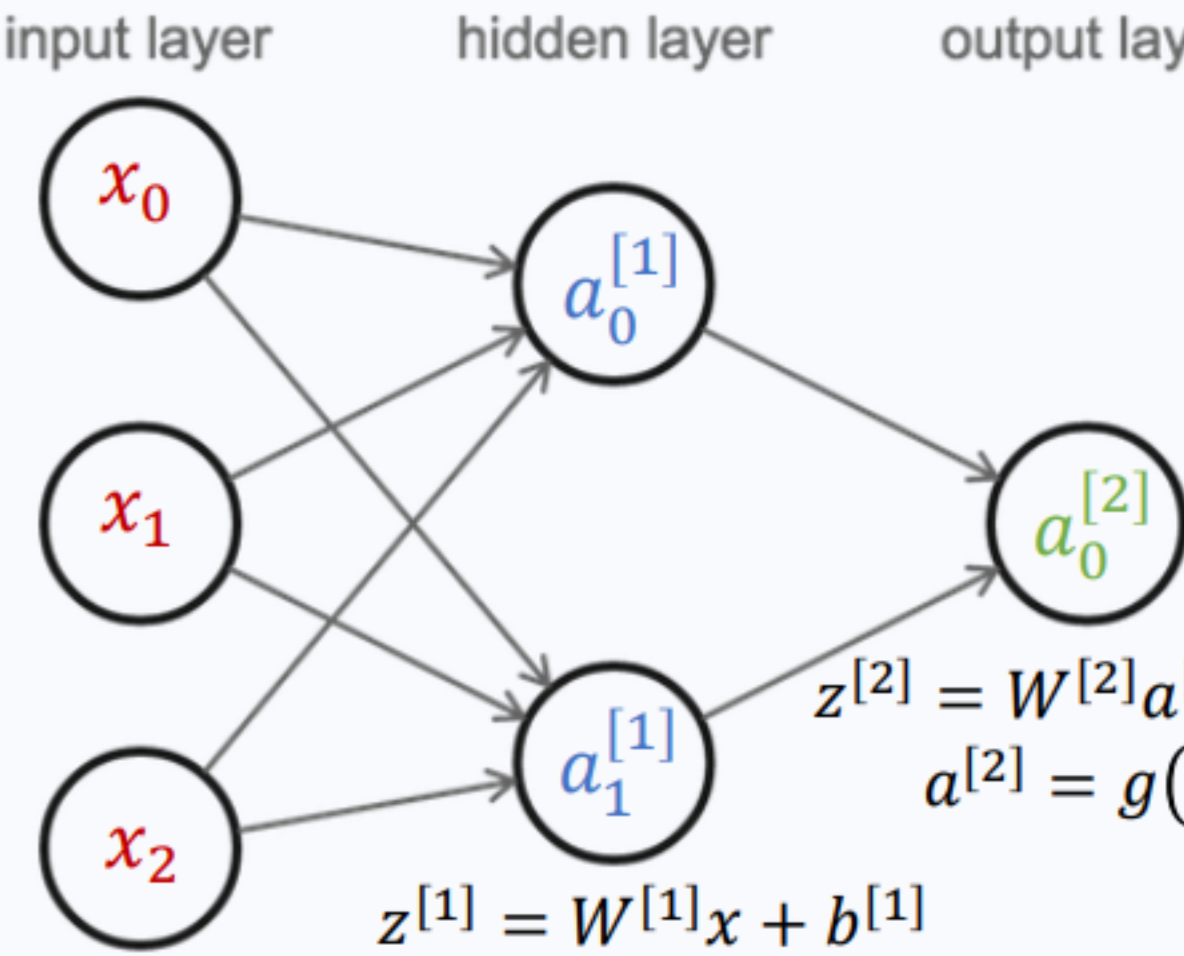
- **activation functions**
  - **ReLU max(0, x) -- all inner layer**
  - **softmax -- converting the last layer into probabilities**

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

  - **we need non-linear activation function! why?**

ONE WON'T BE ENOUGH... NEURAL NETWORK

- **notation: x - input, a - activations, g - activation function (usually ReLU for the hidden layers)**



$$z^{[2]} = W^{[2]} a^{[1]} + b^{[2]}$$
$$a^{[2]} = g(z^{[2]})$$

$$z^{[1]} = W^{[1]} x + b^{[1]}$$
$$a^{[1]} = g(z^{[1]})$$

$$z_0^{[1]} = w_{0,0}^{[1]} x_0 + w_{0,1}^{[1]} x_1 + w_{0,2}^{[1]} x_2 + b_0^{[1]}$$
$$a_0^{[1]} = g\left(z_0^{[1]}\right)$$

$$z_1^{[1]} = w_{1,0}^{[1]} x_0 + w_{1,1}^{[1]} x_1 + w_{1,2}^{[1]} x_2 + b_0^{[1]}$$
$$a_1^{[1]} = g\left(z_1^{[1]}\right)$$

$$z_0^{[2]} = w_{0,0}^{[2]} a_0 + w_{0,1}^{[2]} a_1 + b_0^{[2]}$$
$$a_0^{[2]} = g\left(z_0^{[2]}\right)$$

<u>Vectorization</u>: $x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix}$, $a^{[1]} = \begin{bmatrix} a_0^{[1]} \\ a_1^{[1]} \end{bmatrix}$, $W^{[1]} = \begin{bmatrix} w_{0,0}^{[1]} & w_{0,1}^{[1]} & w_{0,2}^{[1]} \\ w_{1,0}^{[1]} & w_{1,1}^{[1]} & w_{1,2}^{[1]} \end{bmatrix}$, $b^{[1]} = \begin{bmatrix} b_0^{[1]} \\ b_1^{[1]} \end{bmatrix}$

- **but how will we get the <u>w</u> and <u>b</u> parameters?**

LOSS FUNCTION

- it measures how accurate we are, we want to minimize it!
- depends on the predictions and the true labels
  - actually depends on the w, b parameters and the true labels
- differentiable
- need to set based on the problem itself
- popular ones:
  - mean absolute error (MAE) - double the error, double the loss
  - mean squared error (MSE) - double the error, multiply the loss by 4
  - cross-entropy loss $-\frac{1}{M}\sum_i y_i \cdot \log(y_{pred_i})$
    - penalty on being confidently wrong
  - each represents a different task
- The w and b parameters are obtained by minimizing the loss function on the training set!

GRADIENT DESCENT

- Problem:

$$\underset{W,b}{\operatorname{argmin}}\, L(W,b)$$

- One solution:

$$\frac{\partial L}{\partial W} = 0, \qquad \frac{\partial L}{\partial b} = 0$$

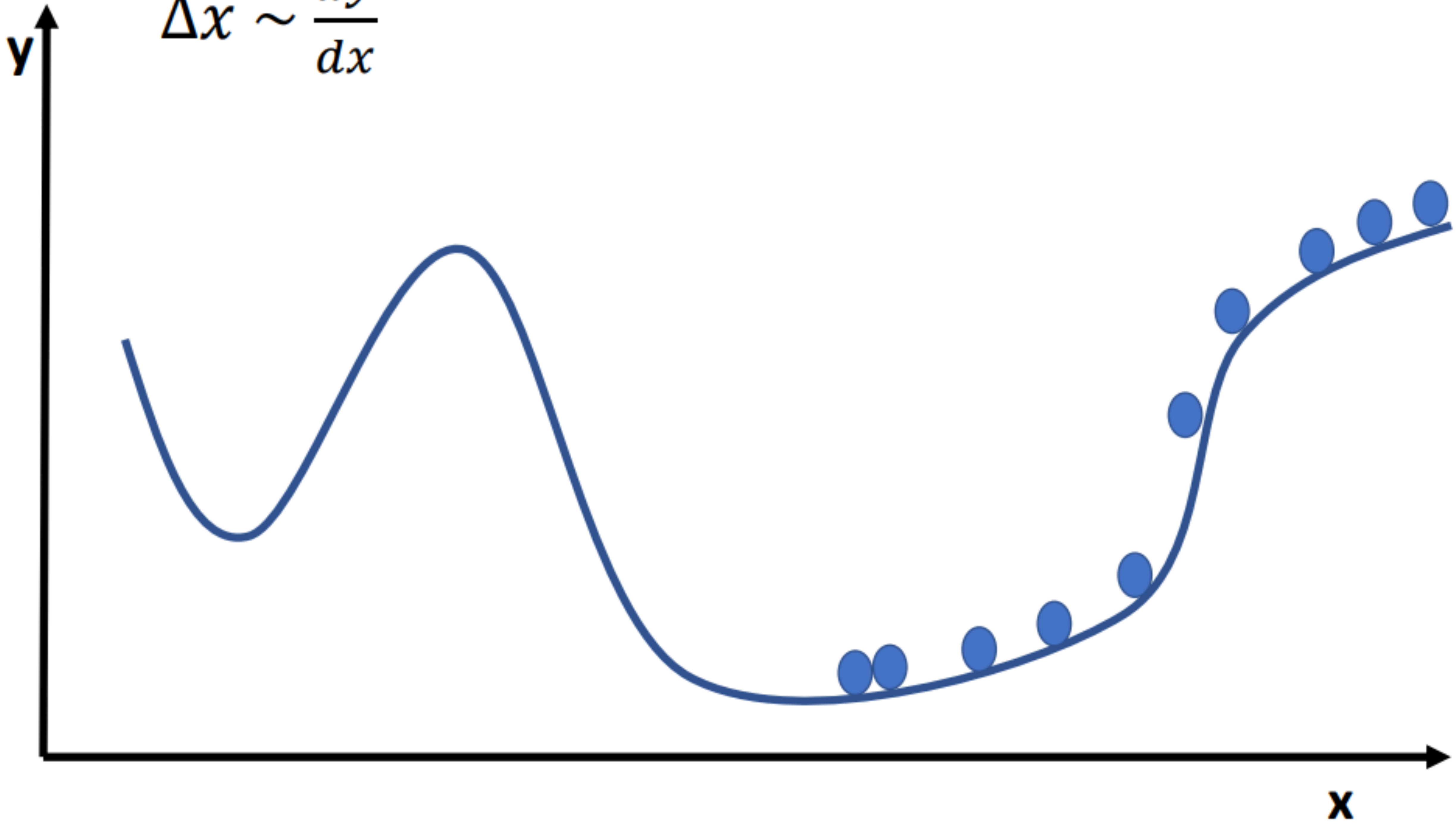
- Problem: too complicated for neural networks
- Solution: gradient descent

$$\text{repeat} \quad \left\{ \begin{aligned} W &= W - \alpha \frac{\partial L}{\partial W} \\ b &= b - \alpha \frac{\partial L}{\partial b} \end{aligned} \right\}$$

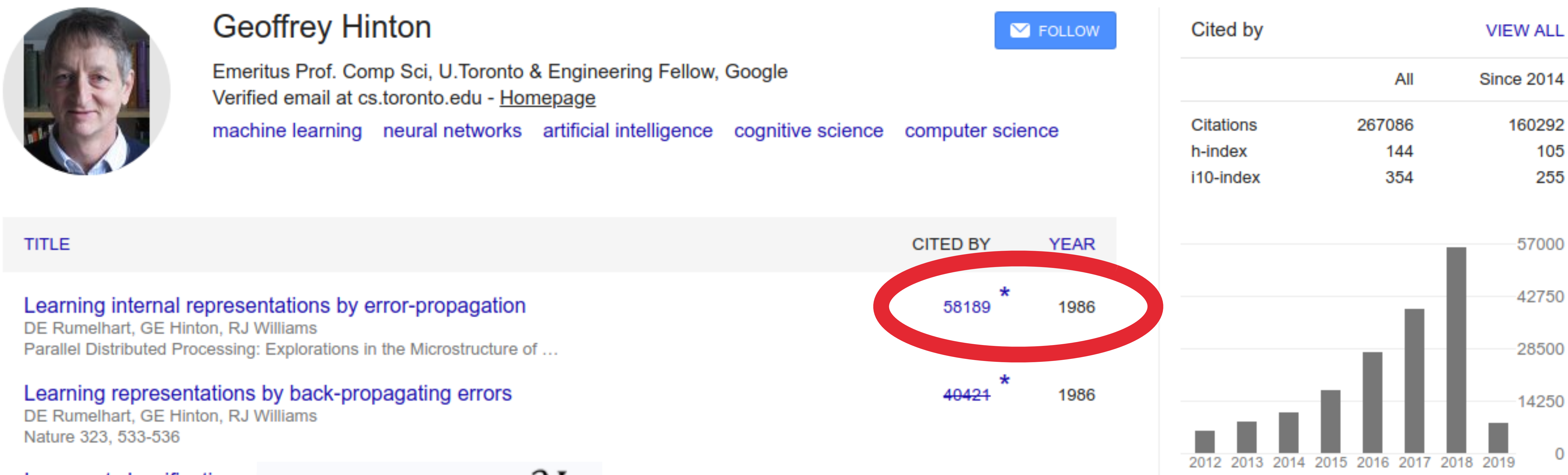
learning rate

Step size in x is proportional to the derivate.

$$\Delta x \sim \frac{dy}{dx}$$



BACKPROPAGATION



- **an update:** $$W = W - \alpha \frac{\partial L}{\partial W}$$
- **we will have a few million W parameters**
  - **it is slow to calculate it million times from ground. but they are not independent!**
- **NN is actually a function composition --> chain rule**

BACKPROPAGATION

- **activations are known**
- **derivate of the activation functions is easy (eq for ReLU it is simply 0 or 1)**

$$a_i = g(z_i) = g(w_{ij}a_j + b_i)$$

$$\frac{\partial L}{\partial w_{ij}} = \frac{\partial L}{\partial a_i}\frac{\partial a_i}{\partial z_i}\frac{\partial z_i}{\partial w_{ij}} = \frac{\partial L}{\partial a_i} g'(z_i) a_j$$

- **for the final layer the derivate is easy**

$$\frac{\partial L}{\partial a_i} = \sum_{l \in L} \frac{\partial L}{\partial a_l}\frac{\partial a_l}{\partial z_l}\frac{\partial z_l}{\partial a_i} = \sum_{l \in L} \frac{\partial L}{\partial a_l} g'(z_l) w_{li}$$

  - **for the previous layers we can propagate the derivation back from the later layers**
- **So training a neural network is actually:**
  - **Forward pass: calculating the activations (the final layer's activation is the prediction)**
  - **Backward pass: calculating the gradients + updating the weights accordingly.**

A FULLY CONNECTED NEURAL NETWORK

- **universal approximation theorem** http://neuralnetworksanddeeplearning.com/chap4.html
  - **a fully connected neural network can approximate any function, it it has enough neurons**

$\mathrm{x} \in \mathbb{R}^N, \mathrm{y} \in \mathbb{R}^K$, neural network: $\mathbb{R}^N \to \mathbb{R}^K$

$$z^{[1]} = W^{[1]}x + b^{[1]}, \qquad W\colon n^{[1]} \times N, \qquad b\colon n^{[1]} \times 1$$
$$a^{[1]} = g\left(z^{[1]}\right)$$

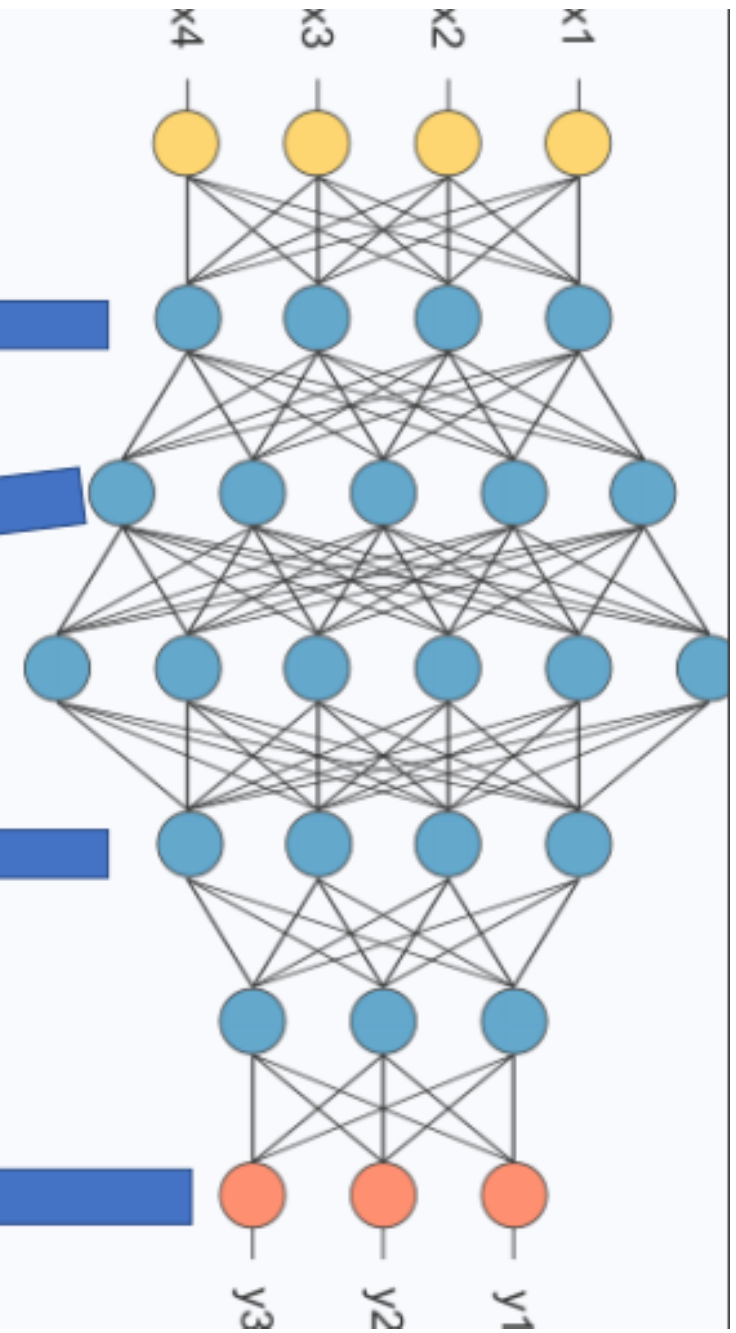
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}, \qquad W\colon n^{[2]} \times n^{[1]}, \qquad b\colon n^{[2]} \times 1$$
$$a^{[2]} = g\left(z^{[2]}\right)$$

$\vdots$

$$z^{[i]} = W^{[i]}a^{[i-1]} + b^{[i]}, \qquad W\colon n^{[i]} \times n^{[i-1]}, \qquad b\colon n^{[i]} \times 1$$
$$a^{[i]} = g\left(z^{[i]}\right)$$

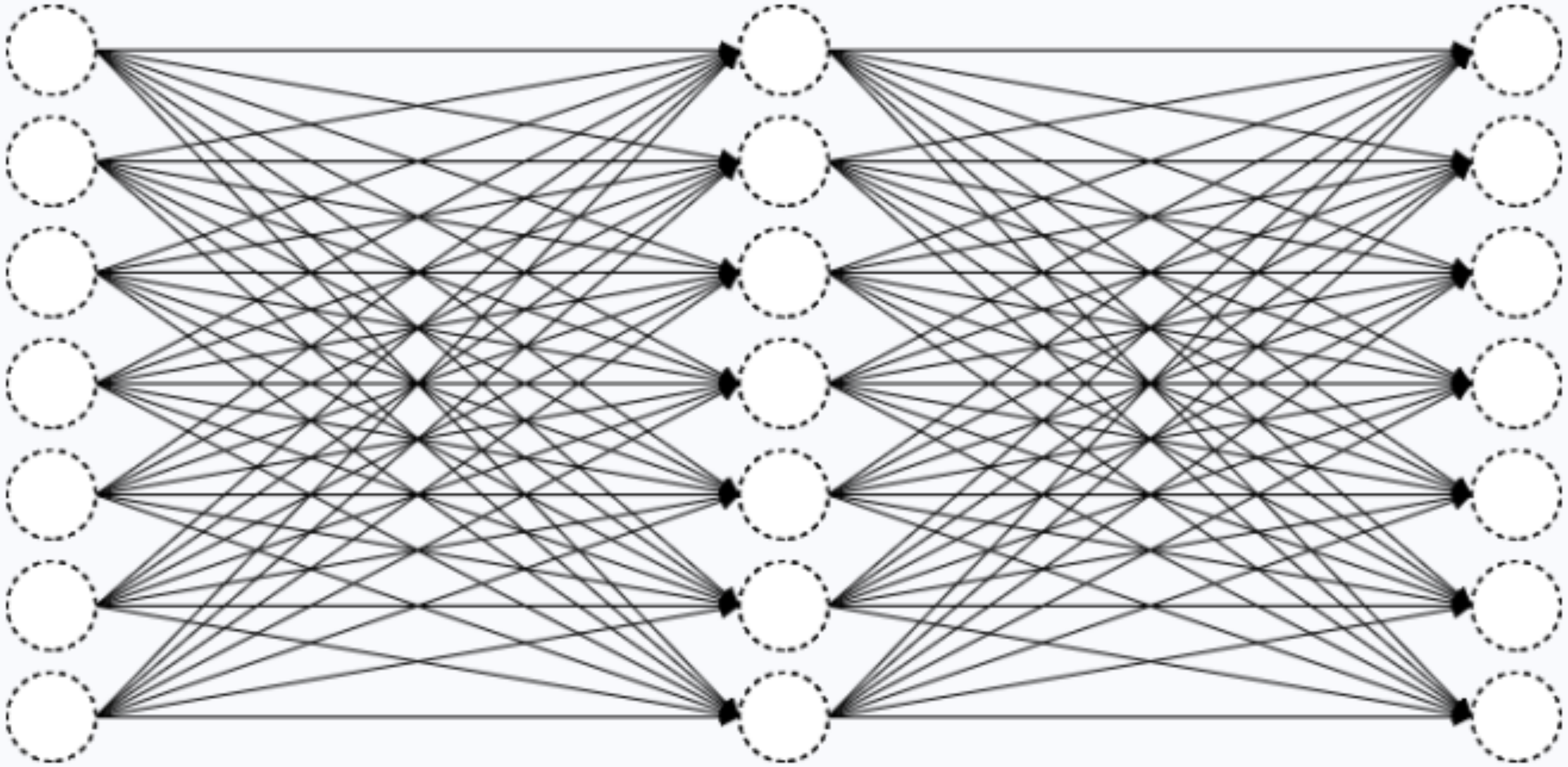
$\vdots$

$$z^{[L]} = W^{[L]}a^{[L-1]} + b^{[L]}, \qquad W\colon n^{[L]} \times n^{[L-1]}, \qquad b\colon n^{[L]} \times 1$$
$$y = a^{[L]} = g\left(z^{[L]}\right)$$



let's try it out! https://playground.tensorflow.org/

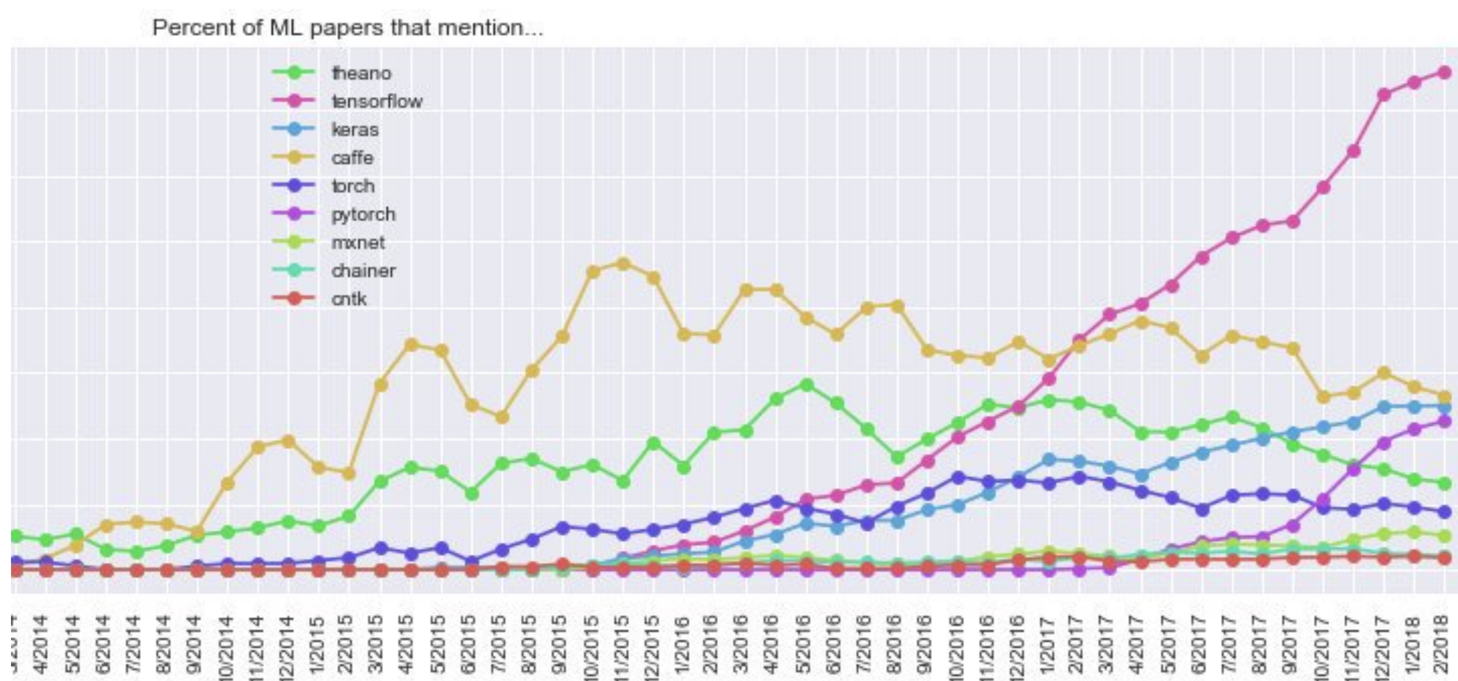
WHY IT WON'T WORK ON REAL-WORLD PHOTOS (ONE REASON OUT OF MANY...)



- Exploding parameter number:
  - 200x200 pixel input → 40000 input
  - $40000^2 + 40000 \approx 1.6 \cdot 10^9$ parameters per layer
  - float32: 4 byte/number → 6.4 GB/layer
  - color images have 3 color channels (RGB) → 57.6 GB/layer

OTHER TECHNICAL DETAILS - NOTATIONS

- **weight initialization**
- **epoch - looping over all the training data once**
- **batch - update the weights according to the average gradients coming from the same batch**
    - **usually 16 - 32 - 64 - 128 - 256 (depends on how much memory you have on the GPU)**



Andrej Karpathy, twitter

- **online training**
- **deep learning library**
    - **Keras / Tensorflow**
    - **Pytorch**
- **GPU vs CPU**
    - **the task is easily parallelizable -> usually GPU is at least 10 times faster**
- **Let's build our first neural network!**
    - **https://github.com/patbaa/physdl/blob/master/notebooks/03/fully_connected.ipynb**

SUMMARY

Jeremy Howard
@jeremyphoward

Követés

1. Multiply things together
2. Add them up
3. Replaces negatives with zeros
4. Return to step 1, a hundred times

> **Morgan Housel** @morganhousel
> "When you first study a field, it seems like you have to memorize a zillion things. You don't. What you need is to identify the 3-5 core principles that govern the field. The million things you thought you had to memorize are various combinations of the core principles." -J. Reed
>
> Hozzászóláslánc megjelenítése

22:11 - 2019. febr. 28.

91 retweet **571** kedvelés