AN INTRODUCTION TO NEURAL NETWORKS AND DEEP LEARNING

A guest lecture for BMEN Mathematical Modeling

TULANE UNIVERSITY

ALEXEJ GOSSMANN

APRIL 26, 2018

Quick remark before we get to the theoretical part of this presentation...

LIVE DEMOS

This lecture contains several live demos. Why?

- DL is largely an empirical research field. 👩‍💻
- Sometimes DL in actions seems like magic! 🧙‍♀️
- DL is not my research field — I finally have an excuse to spend some time figuring out the common DL libraries! 👯‍♂️

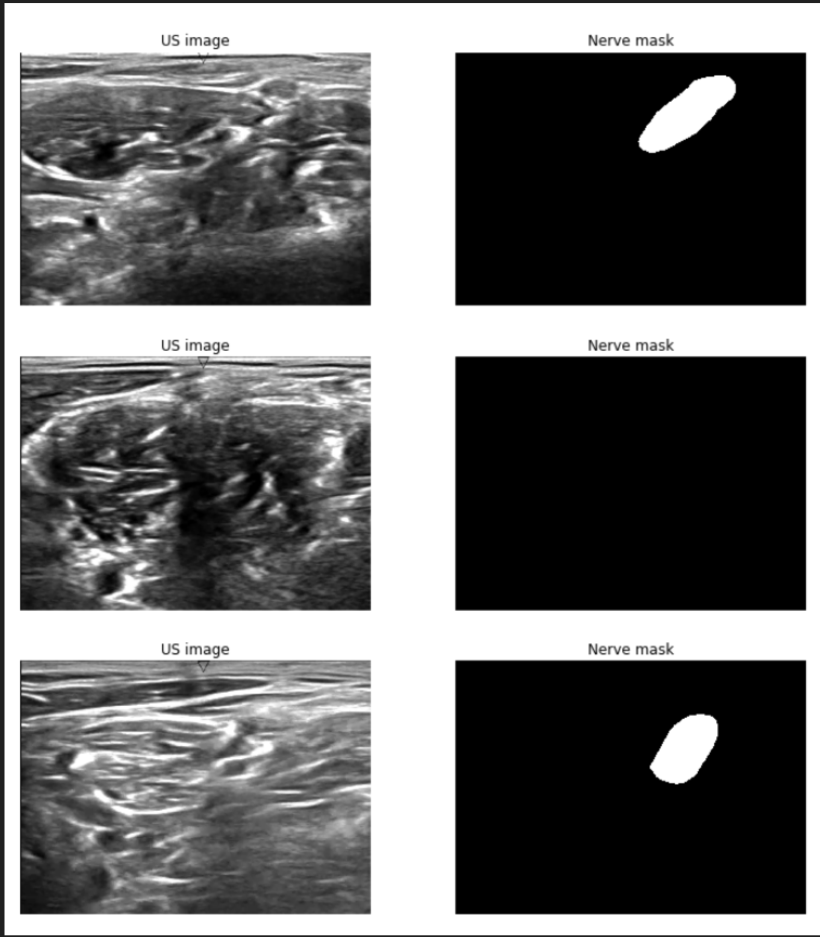
LIVE DEMOS

Main example:

ULTRASOUND NERVE SEGMENTATION

- **Task: Identify nerve structures in ultrasound images of the neck.**
- Needed for effective and accurate insertion of a patient's pain management catheter.

ULTRASOUND NERVE SEGMENTATION



Task: Predict the nerve mask for an input ultrasound image.

5635 training images, 5508 test images.

420×580 resolution.

~47% images don't have a mask.

Data from a $100,000 competition held in 2016: https://www.kaggle.com/c/ultrasound-nerve-segmentation

I will live-execute the following code (more on the hardware & software requirements later):

https://github.com/agisga/ultrasound-nerve-segmentation/blob/master/jupyter/U-net_improved.ipynb

The code borrows from: [1], [2].

Hopefully, by the end of this lecture you will understand all of the components of this deep learning model.

ARTIFICIAL NEURAL NETWORKS AND DEEP LEARNING

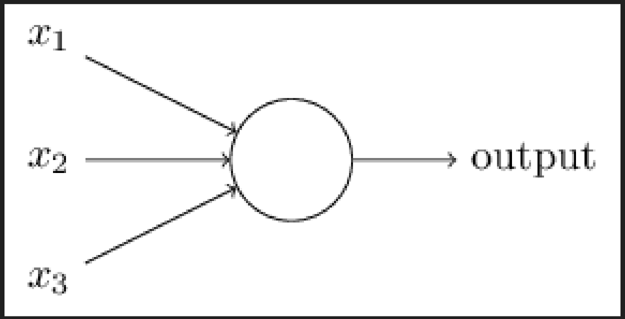
The theoretical part of this presentation (including most figures) is largely based on:

Michael A. Nielsen, "Neural Networks and Deep Learning", Determination Press, 2015 (licensed under a Creative Commons Attribution-NonCommercial 3.0 Unported License)

PERCEPTRONS

- Developed in the 50s and 60s by Frank Rosenblatt.
- Inspired by earlier work by Warren McCulloch and Walter Pitts (cf., NAUTILUS. The Man Who Tried to Redeem the World with Logic.).

Weighing multiple input factors to make a decision according to a decision threshold.



Binary inputs $x_1, x_2, \ldots \in \{0, 1\}$, and binary output.

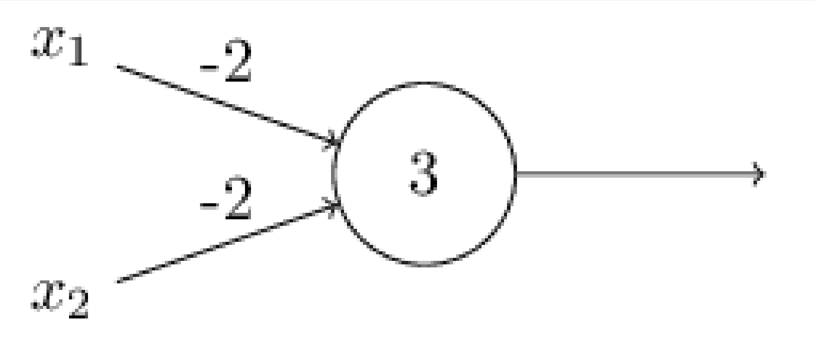
$$\text{output} = \begin{cases} 0 & \text{if } \sum_i w_i x_i \leq \text{threshold} \\ 1 & \text{if } \sum_i w_i x_i > \text{threshold} \end{cases} \tag{1}$$

BETTER NOTATION

$$\mathbf{x} \mapsto \begin{cases} 0 & \text{if } \mathbf{w}^T\mathbf{x} + b \leq 0 \\ 1 & \text{if } \mathbf{w}^T\mathbf{x} + b > 0 \end{cases} \tag{2}$$

- Input vector $\mathbf{x} \in \{0, 1\}^p$,
- Vector of weights $\mathbf{w} \in \mathbb{R}^p$,
- A *bias* term $b \in \mathbb{R}$,
- An output in $\{0, 1\}$.

Example: **NAND Gate.**

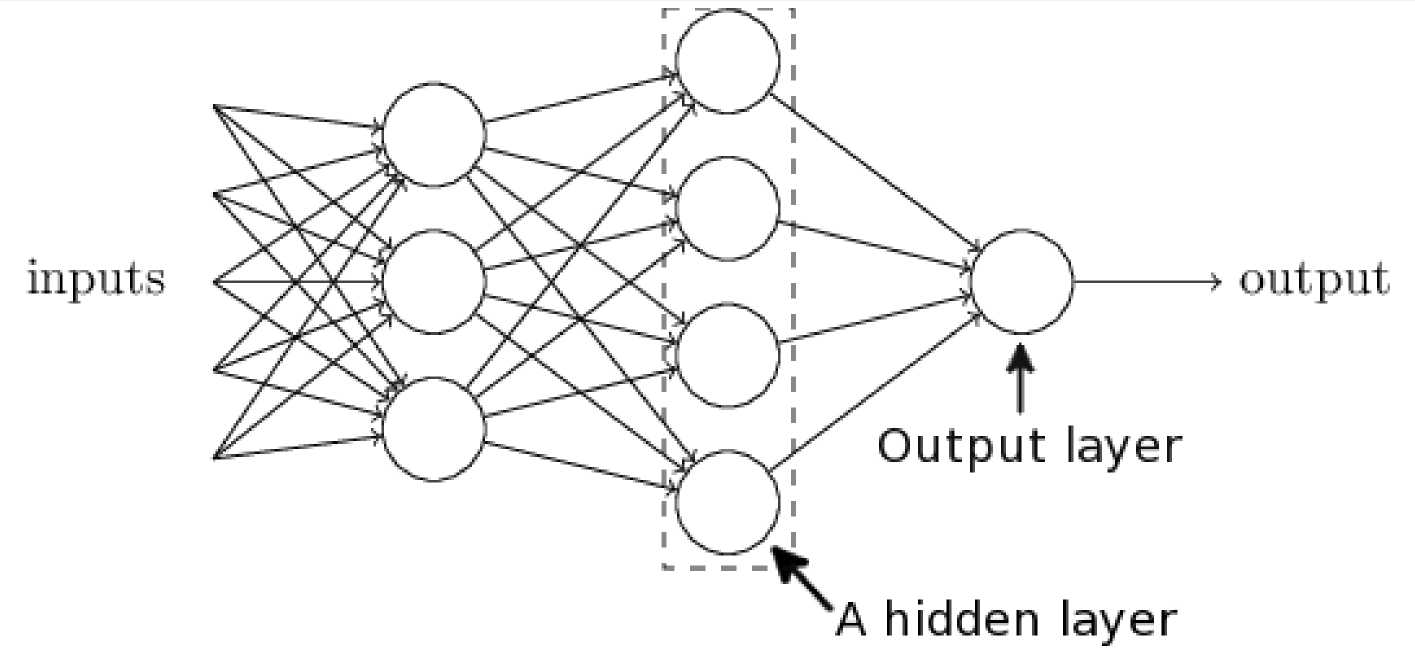


$$\text{output} = \begin{cases} 0 & \text{if } (-2x_1 - 2x_2 + 3) \leq 0 \\ 1 & \text{if } (-2x_1 - 2x_2 + 3) > 0 \end{cases} \tag{3}$$

That is, $00 \mapsto 1$, $01 \mapsto 1$, $10 \mapsto 1$, $11 \mapsto 0$.

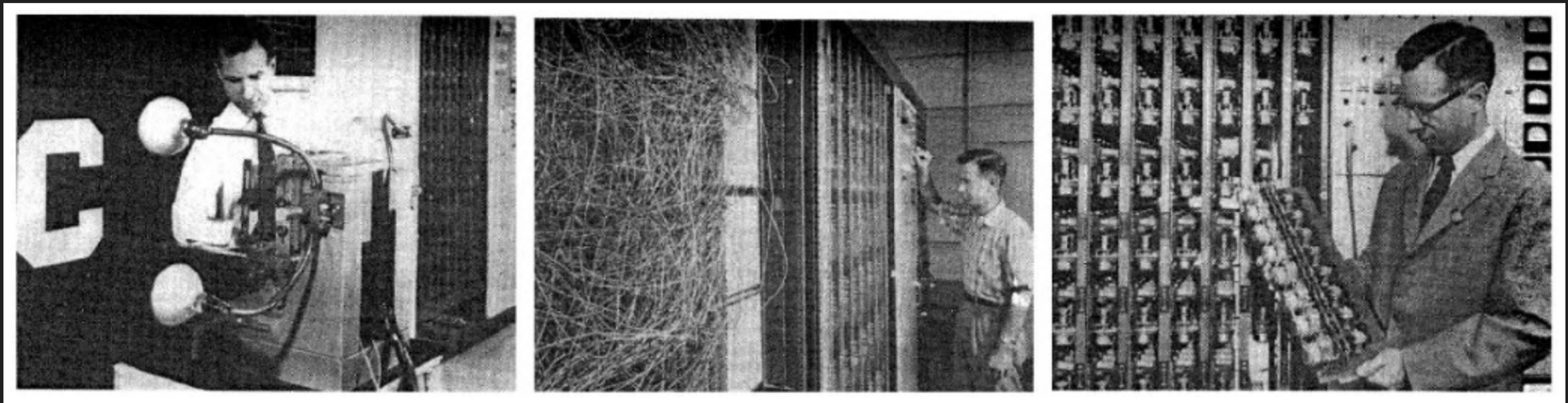
(NAND gate is universal for computation. Analogy: a network of perceptrons and a computational circuit.)

A NETWORK OF PERCEPTRONS



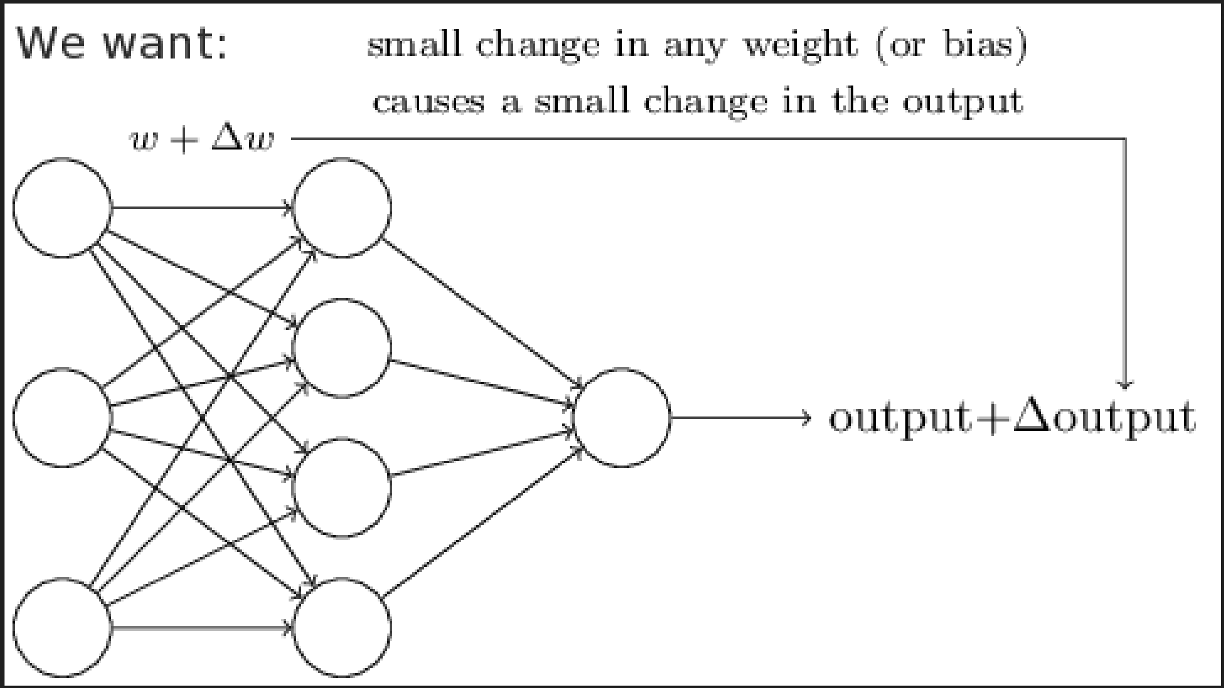
Hidden layer: Making a decision at a more abstract level by weighing up the results of the first layer.

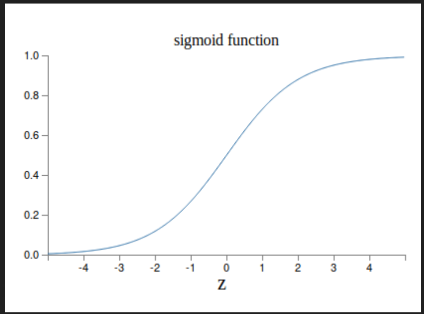
THE “MARK 1 PERCEPTRON” (CORNELL, 1957)



Array of 400 photocells, connected to the “neurons”.
The weights (w_i) and biases (b) are potentiometers.

In 1958 The New York Times reported the perceptron to be “the embryo of an electronic computer that [the Navy] expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence.” (Mikel Olazaran (1996) *A Sociological Study of the Official History of the Perceptrons Controversy*)



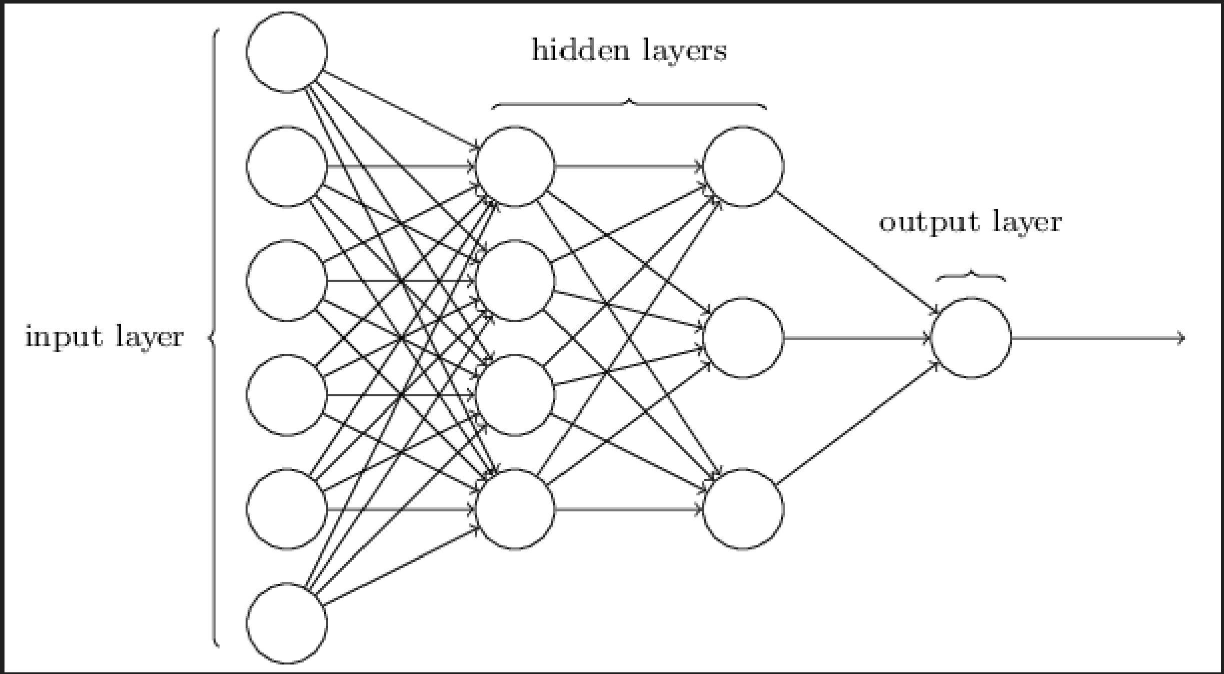


Use the sigmoid function as the *activation function*.

The sigmoid neuron:

$$\mathbf{x} \mapsto \frac{1}{1 + \exp(-\mathbf{w}^T\mathbf{x} - b)}$$

MULTILAYER PERCEPTRON (MLP)



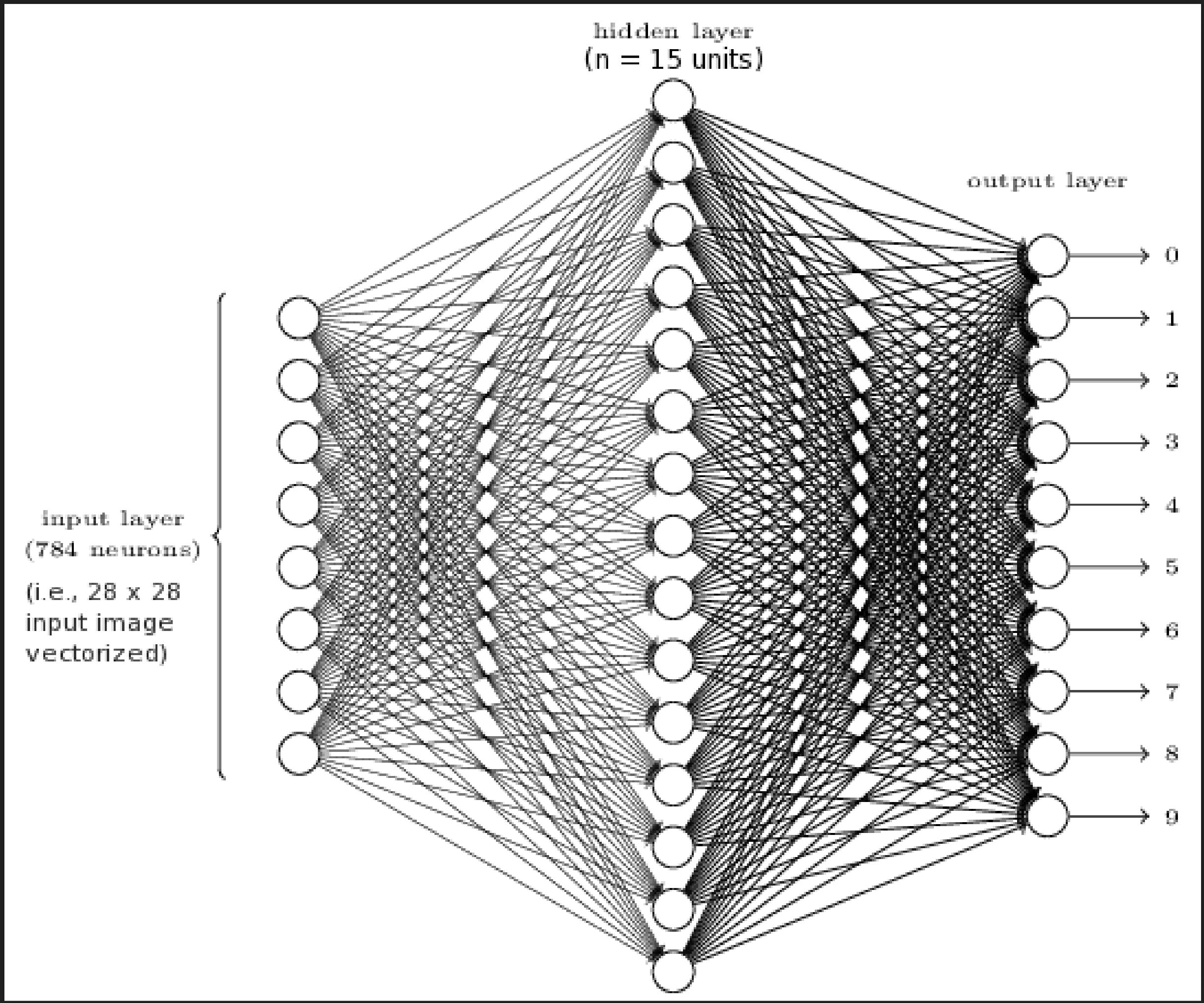
- Made up of sigmoid neurons, not perceptrons — a misnomer?
- *Feedforward* neural network — no loops allowed.

UNIVERSALITY THEOREM

> *Neural networks can be used to approximate any continuous function to any desired precision.*

(among other sources, see Chapter 4 in Michael Nielsen's book, or George Cybenko (1989) Approximation by superpositions of a sigmoidal function).

MLP FOR RECOGNITION OF HANDWRITTEN DIGITS



Let's see how well it works in a live demonstration!

LIVE DEMONSTRATIONS

State-of-the-art neural network models require a modern GPU (or GPUs).

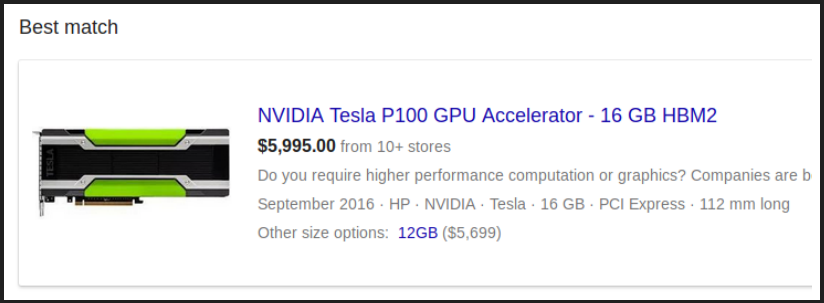
In this talk I use the following setup on a Google Cloud virtual machine.

Instructions to reproduce the same exact setup can be found at

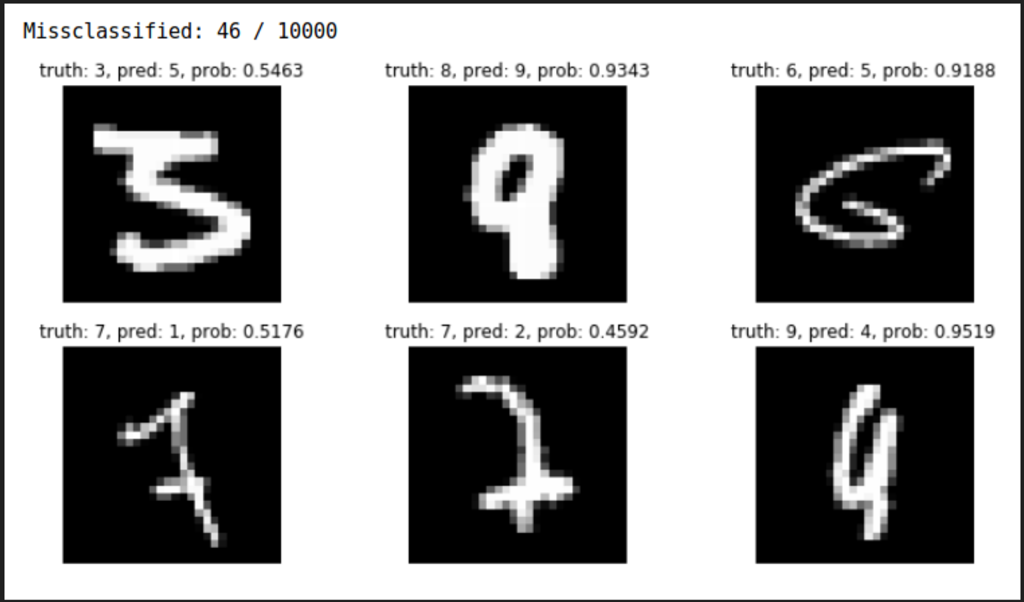
https://github.com/agisga/coding_notes/blob/master/goo

Components:

- NVIDIA Tesla P100 GPU rented on Google Cloud (about $1.19 per hour), and GPU drivers on linux
- CUDA and cuDNN libraries for numerical computing on the GPU.
- TensorFlow, with Keras as frontend.
- Python, SciPy, NumPy, Jupyter Notebook, ...



CLASSIFICATION OF HANDWRITTEN DIGITS



I will execute the following code:

https://github.com/agisga/coding_notes/blob/master/Keras/MNIST_functional_API.ipynb

STOCHASTIC GRADIENT DESCENT (SGD)

How do we find the optimal weights and biases?

Cost function:

$$C = \frac{1}{2n} \sum_{i=1}^{n} \| y(\mathbf{x}_i) - a(\mathbf{x}_i) \|_2^2,$$

where $a(\mathbf{x}_i)$ is the output of the NN and $y(\mathbf{x}_i)$ is the desired output for the input $\mathbf{x}_i$.

GRADIENT DESCENT

- $\begin{bmatrix} \mathbf{w} + \Delta\mathbf{w} \\ b + \Delta b \end{bmatrix} \rightsquigarrow C + \Delta C.$
- $\Delta C \approx \nabla C \cdot \begin{bmatrix} \Delta\mathbf{w} \\ \Delta b \end{bmatrix}.$
- We want: $\Delta C < 0.$
- Update: $\begin{bmatrix} \mathbf{w} \\ b \end{bmatrix} \leftarrow \begin{bmatrix} \mathbf{w} \\ b \end{bmatrix} - \eta \nabla C.$
- $\eta > 0$ is called the *learning rate*.

STOCHASTIC GRADIENT DESCENT (SGD)

$$C = \frac{1}{2n}\sum_{i=1}^{n}\|y(\mathbf{x}_i) - a(\mathbf{x}_i)\|_2^2 = \frac{1}{n}\sum_{i=1}^{n} C_i,$$

where $C_i := (y(\mathbf{x}_i) - a(\mathbf{x}_i))^2/2.$

Randomly choose a subset $\mathbf{x}_{i_1}, \ldots, \mathbf{x}_{i_m}$ of size $m \ll n.$

Then, $\nabla C \approx \frac{1}{m}\sum_{j=1}^{m} \nabla C_{i_j},$

The set $\{\mathbf{x}_{i_1}, \ldots, \mathbf{x}_{i_m}\}$ is called a *mini-batch*.

STOCHASTIC GRADIENT DESCENT (SGD)

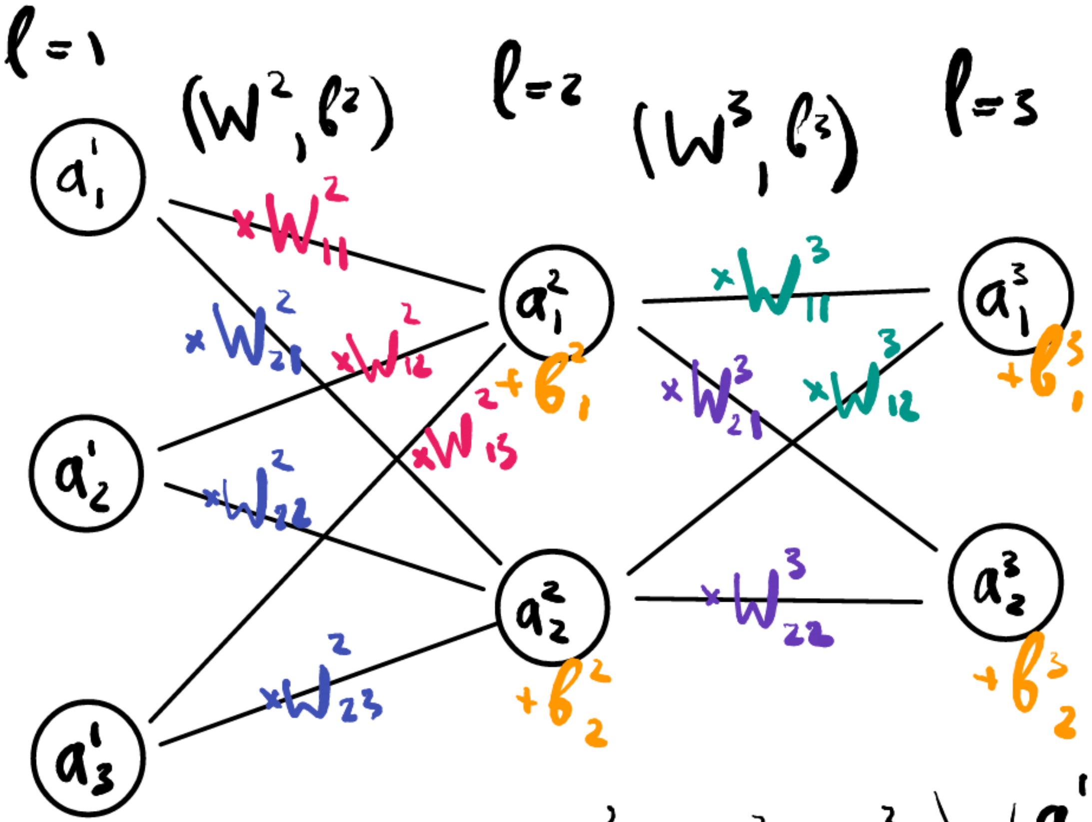
1. Pick a mini-batch size $m \ll n$.
2. Randomly (without replacement) choose a mini-batch $\mathbf{x}_{i_1}, \ldots, \mathbf{x}_{i_m}$.
3. Update all weights and biases:
$$\begin{bmatrix} \mathbf{w} \\ b \end{bmatrix} \leftarrow \begin{bmatrix} \mathbf{w} \\ b \end{bmatrix} - \frac{\eta}{m} \sum_{j=1}^{m} \nabla C_{i_j}.$$
4. Repeat 2-3.
5. An *epoch* is completed, when every input $\mathbf{x}_i$ has been used.

STOCHASTIC GRADIENT DESCENT (SGD)

Now we only need to figure out how to differentiate C_{i_j} with respect to every weight and every bias in the neural network...🤔

THE BACKPROPAGATION ALGORITHM

- 1986 paper by Rumelhart, Hinton, and Williams.
- The “workhorse” of deep learning.



$$a^2 = \begin{pmatrix} a^2_1 \\ a^2_2 \end{pmatrix} = \sigma\left[\begin{pmatrix} W^2_{11} & W^2_{12} & W^2_{13} \\ W^2_{21} & W^2_{22} & W^2_{23} \end{pmatrix} \cdot \begin{pmatrix} a^1_1 \\ a^1_2 \\ a^1_3 \end{pmatrix} + \begin{pmatrix} b^2_1 \\ b^2_2 \end{pmatrix}\right]$$

$$a^2 = \sigma\left(W^2 \cdot a^1 + b^2\right) = \sigma(z^2), \qquad a^3 = \sigma\left(W^3 \cdot a^2 + b^3\right) = \sigma(z^3).$$

MORE BETTER NOTATION

- Layers $l = 1, 2, \ldots, L$.
- $\mathbf{a}^1 :=$ input layer (as a vector).
- $\mathbf{a}^l :=$ activations of layer l (as a vector).
- $W^l :=$ matrix of weights at layer l.
- $\mathbf{b}^l :=$ vector of biases at layer l.
- $\mathbf{z}^l := W^l \mathbf{a}^{l-1} + \mathbf{b}^l$.

$$\Rightarrow \mathbf{a}^l = \sigma(W^l \mathbf{a}^{l-1} + \mathbf{b}^l) = \sigma(\mathbf{z}^l)$$

CHAIN RULE!

Layer L (last layer):

$$\frac{\partial C}{\partial W_{ij}^{L}} = \nabla_{\mathbf{a}^L} C \cdot \frac{\partial \sigma(\mathbf{z}^L)}{\partial \mathbf{z}^L} \cdot \frac{\partial \mathbf{z}^L}{\partial W_{ij}^{L}}$$

$$=: \delta^L \cdot \frac{\partial \mathbf{z}^L}{\partial W_{ij}^{L}} = \delta^L \cdot \mathbf{a}_j^{L-1}$$

Layer L-1:

$$\frac{\partial C}{\partial W_{ij}^{L-1}} = \nabla_{\mathbf{a}^L} C \cdot \frac{\partial \sigma(\mathbf{z}^L)}{\partial \mathbf{z}^L} \cdot \frac{\partial \mathbf{z}^L}{\partial \mathbf{z}^{L-1}} \cdot \frac{\partial \mathbf{z}^{L-1}}{\partial W_{ij}^{L-1}}$$

$$= \delta^L \cdot W^L \cdot \frac{\partial \sigma(\mathbf{z}^{L-1})}{\partial \mathbf{z}^{L-1}} \cdot \frac{\partial \mathbf{z}^{L-1}}{\partial W_{ij}^{L-1}}$$

$$=: \delta^{L-1} \cdot \frac{\partial \mathbf{z}^{L-1}}{\partial W_{ij}^{L-1}} = \delta^{L-1} \cdot \mathbf{a}_j^{L-2}$$

Likewise, any other **layer l:** $\frac{\partial C}{\partial W_{ij}^{l}} = \delta^l \cdot \mathbf{a}_j^{l-1}$,
where δ^l is determined by δ^{l+1}, W^{l+1}, and $\partial \sigma(\mathbf{z}^l)/\partial \mathbf{z}^l$.

THE BACKPROPAGATION ALGORITHM

1. **Input x:** Set the corresponding activation a^1 for the input layer.

2. **Feedforward:** For each $l = 2, 3, \ldots, L$ compute $z^l = w^l a^{l-1} + b^l$ and $a^l = \sigma(z^l)$.

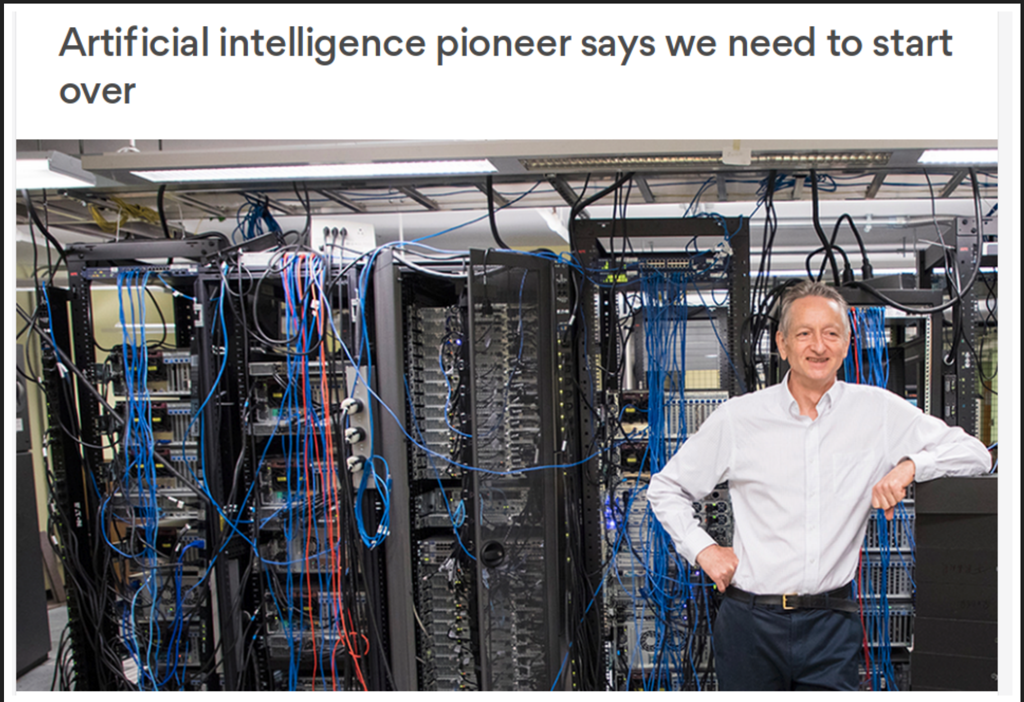
3. **Output error δ^L:** Compute the vector $\delta^L = \nabla_a C \odot \sigma'(z^L)$.

4. **Backpropagate the error:** For each $l = L-1, L-2, \ldots, 2$ compute $\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$.

5. **Output:** The gradient of the cost function is given by $\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j$ and $\frac{\partial C}{\partial b^l_j} = \delta^l_j$.

BACKPROPAGATION — ASIDE:

https://www.axios.com/artificial-intelligence-pioneer-says-we-need-to-start-over-1513305524-f619efbd-9db0-4947-a9b2-7a4c310a28fe.html



BACKPROPAGATION — ASIDE:

Hinton (...) is now "deeply suspicious" of back-propagation (...). "My view is throw it all away and start again," he said.

(...)

"Max Planck said, 'Science progresses one funeral at a time.' The future depends on some graduate student who is deeply suspicious of everything I have said."

IMPROVING THE WAY NN WORK

There are a lot of techniques, tricks, and best practices (some based on theory, some on empirical trial and error). Here is a small selection.

Least squares loss: Learning slowdown b/c ∇C depends on $\sigma'(z)$ (≈ 0 for $|z| > 5$).

A BETTER CHOICE OF COST FNCT.

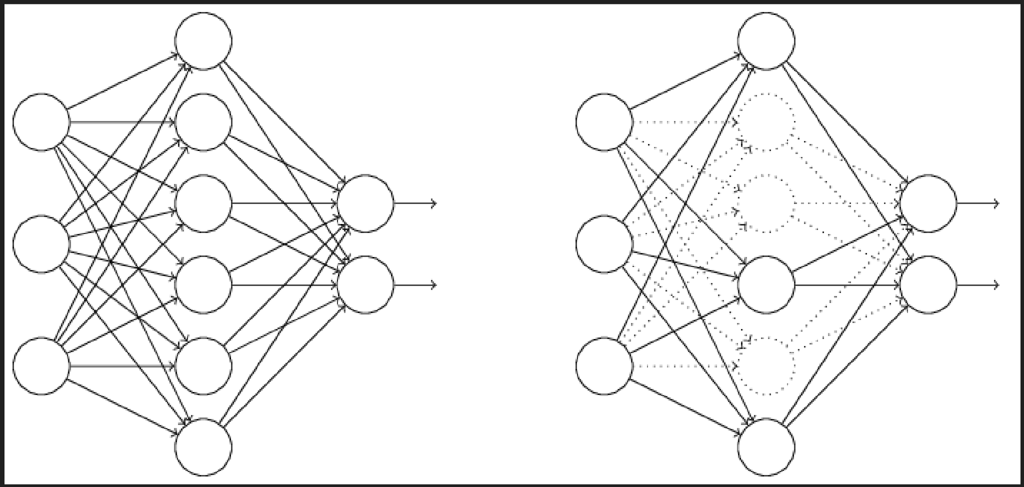
Categorical cross-entropy:

$$C = -\frac{1}{n} \sum_x [y \ln(a) + (1 - y) \ln(1 - a)].$$

REGULARIZATION

Reduce over-fitting to the training data.

- L2: minimize $C + \frac{\lambda}{2n} \sum_w w^2$.
- L1: minimize $C + \frac{\lambda}{n} \sum_w |w|$.
- Dropout: Within every iteration of backprop, randomly and temporarily delete some neurons.

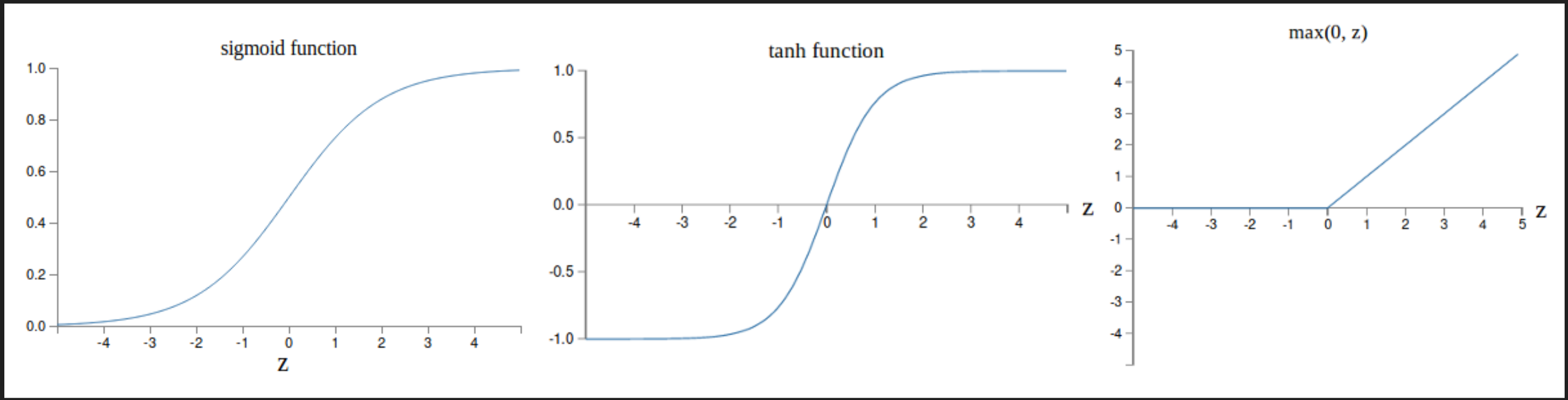


VARIATIONS OF SGD

- Hessian technique
- Momentum-based GD
- Many more: An overview of gradient descent optimization algorithms (a blog post by Sebastian Ruder)

ACTIVATION FUNCTIONS

- Sigmoid: $\frac{1}{1+\exp(-\mathbf{w}^T\mathbf{x}-b)}$
- Hyperbolic tan: $\tanh(\mathbf{w}^T\mathbf{x}+b)$
- Rectifier linear unit (ReLU): $\max(0, \mathbf{w}^T\mathbf{x}+b)$.
- Softmax. E.g., in MNIST last layer: $a_j^L = \frac{\exp(z_j^L)}{\sum_{k=0}^{9}\exp(z_k^L)}$ can be interpreted as "probability" that input image shows digit j.

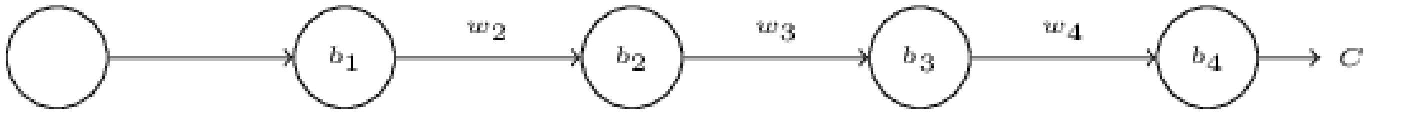


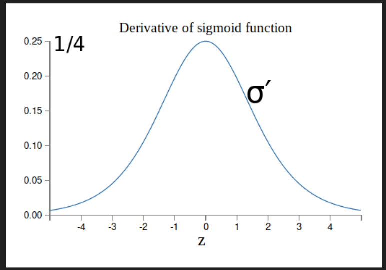
Let's try these techniques on the handwritten digit recognition example.

BACKPROPAGATION — THE VANISHING GRADIENT PROBLEM IN DEEP NN

(unrelated to Hinton's remarks on a previous slide)

$$\frac{\partial C}{\partial b_1} = \sigma'(z_1) \times w_2 \times \sigma'(z_2) \times w_3 \times \sigma'(z_3) \times w_4 \times \sigma'(z_4) \times \frac{\partial C}{\partial a_4}$$





With standard initialization $|w_j| < 1$, as so, $|w_j \sigma'(z_j)| < 1/4$. $\Rightarrow$ Vanishing gradient in the earlier layers of a deep model $\Rightarrow$ The earlier layers "learn" much slower than later layers.

Likewise: exploding gradient when all $|w_j \sigma'(z_j)| \gg 1$.

DEEP LEARNING

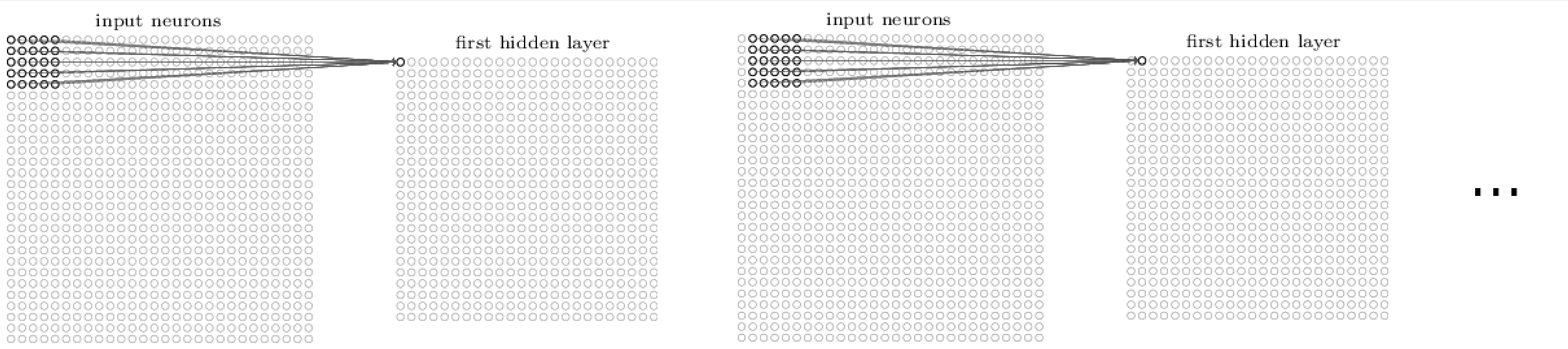
CONVOLUTIONAL NEURAL NETWORK

Deep convolutional network is the most widely used type of deep nearal network.

Modern CNN: LeCun, Bottou, Bengio, Haffner (1998).

CONVOLUTION

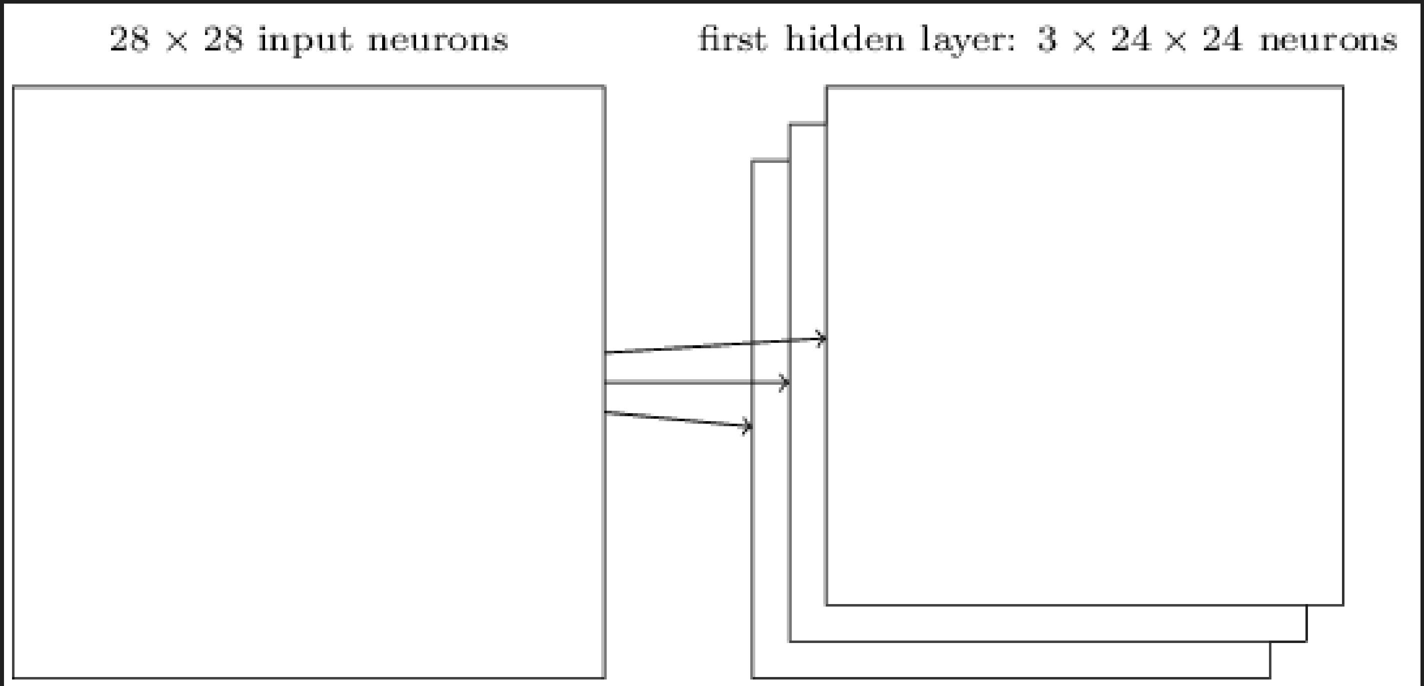
Each neuron in the hidden layer is connected to only $5 \times 5 = 25$ input activations (pixels).



- The 25 weights and the bias parameter are shared (i.e., the same for each hidden neuron). $\Rightarrow$ Drastic reduction in the number of parameters.
- Each neuron looks for the same "feature", just at a different location.

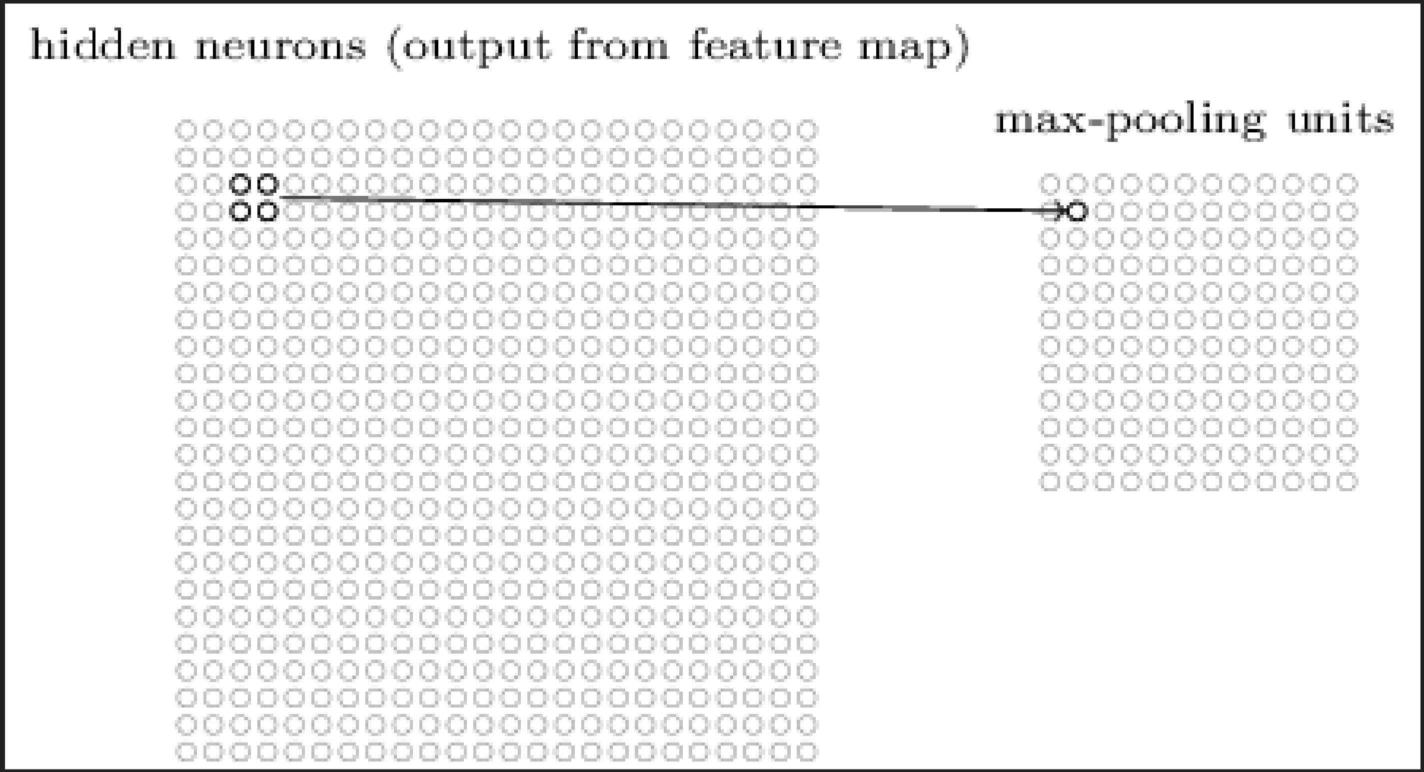
CONVOLUTIONAL LAYER

A complete convolutional layer uses several convolutional filters to produce several different feature maps.



POOLING LAYERS

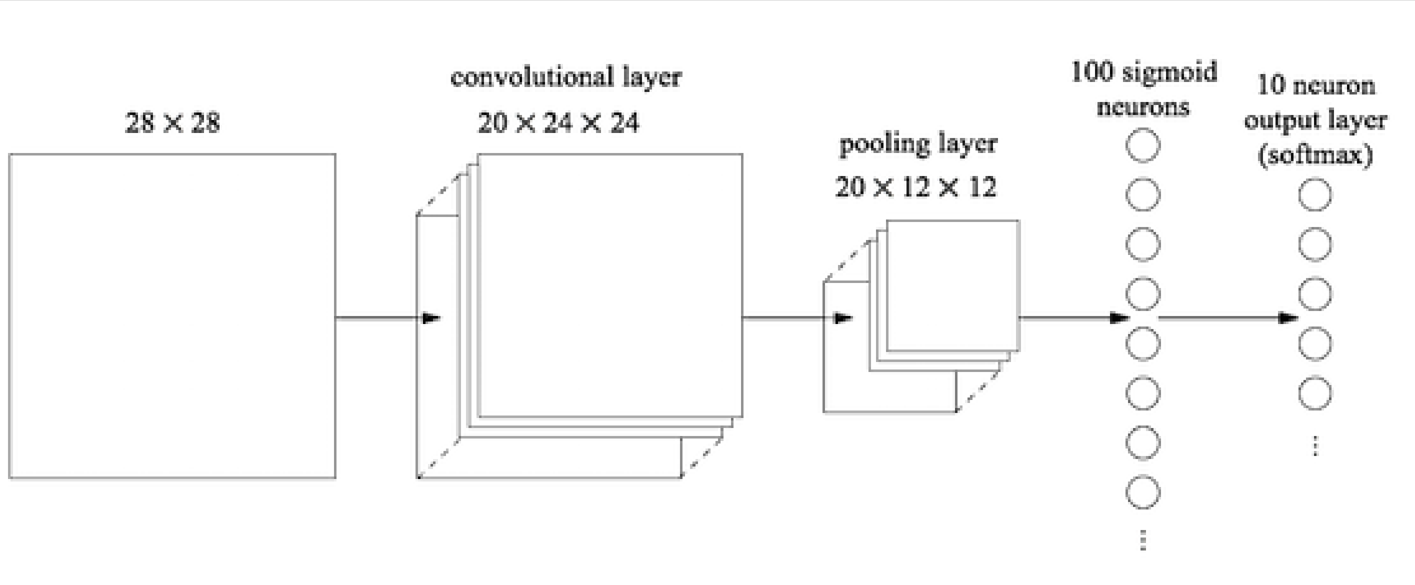
Most common: max-pooling



Other options: L2-pooling, average pooling.

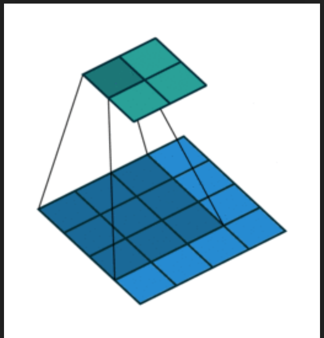
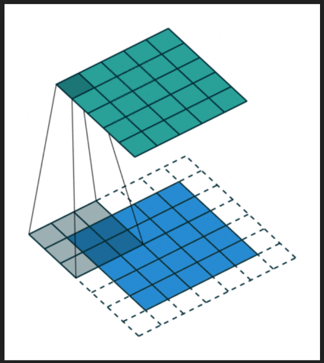
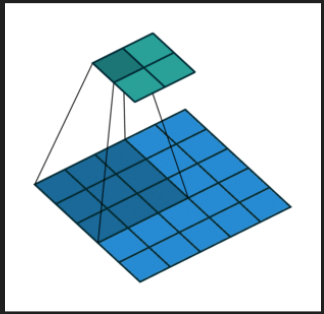
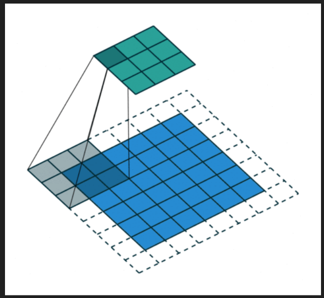
CONVOLUTIONAL NEURAL NETWORK

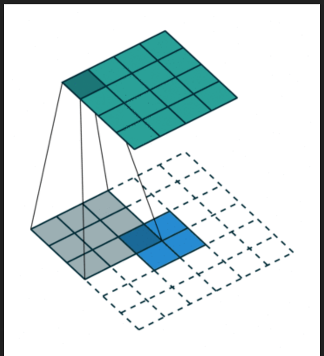
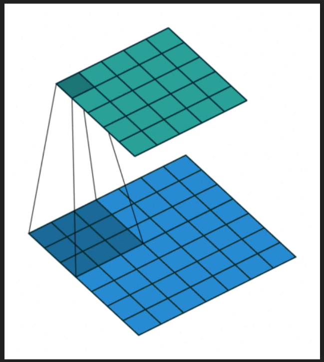
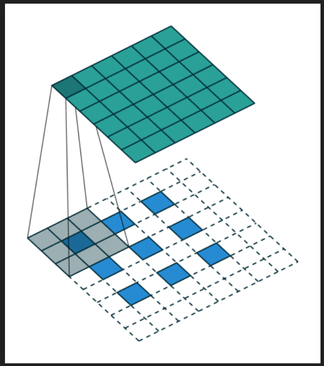
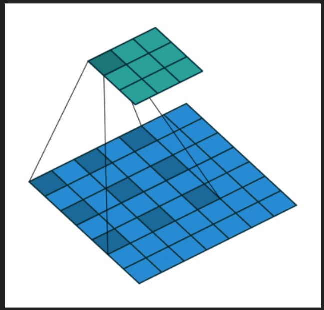
Putting it all together



MANY TYPES OF CONVOLUTIONS

Vincent Dumoulin, Francesco Visin - A guide to convolution arithmetic for deep learning

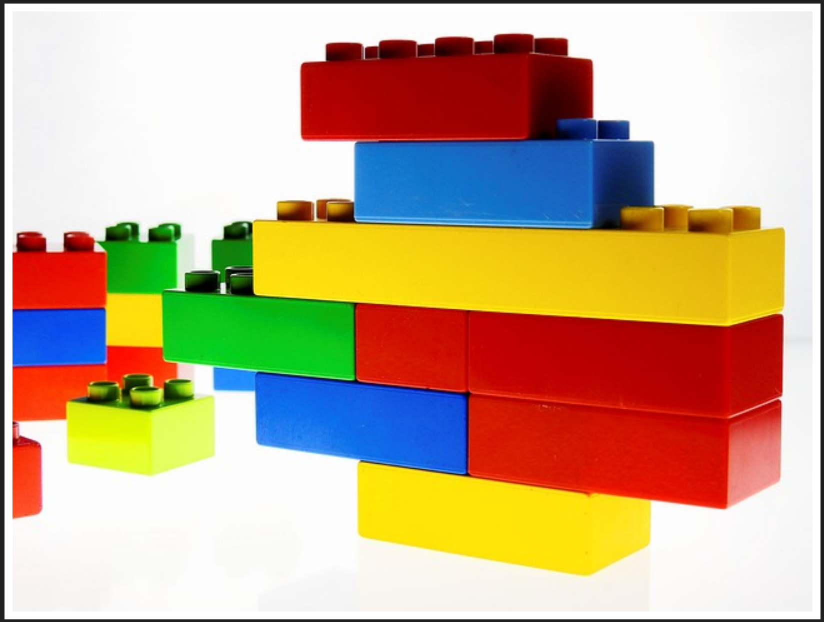





Animations source: https://github.com/vdumoulin/conv_arithmetic

CONVOLUTIONAL NEURAL NETWORK

Let's try it on the handwritten digit recognition example.

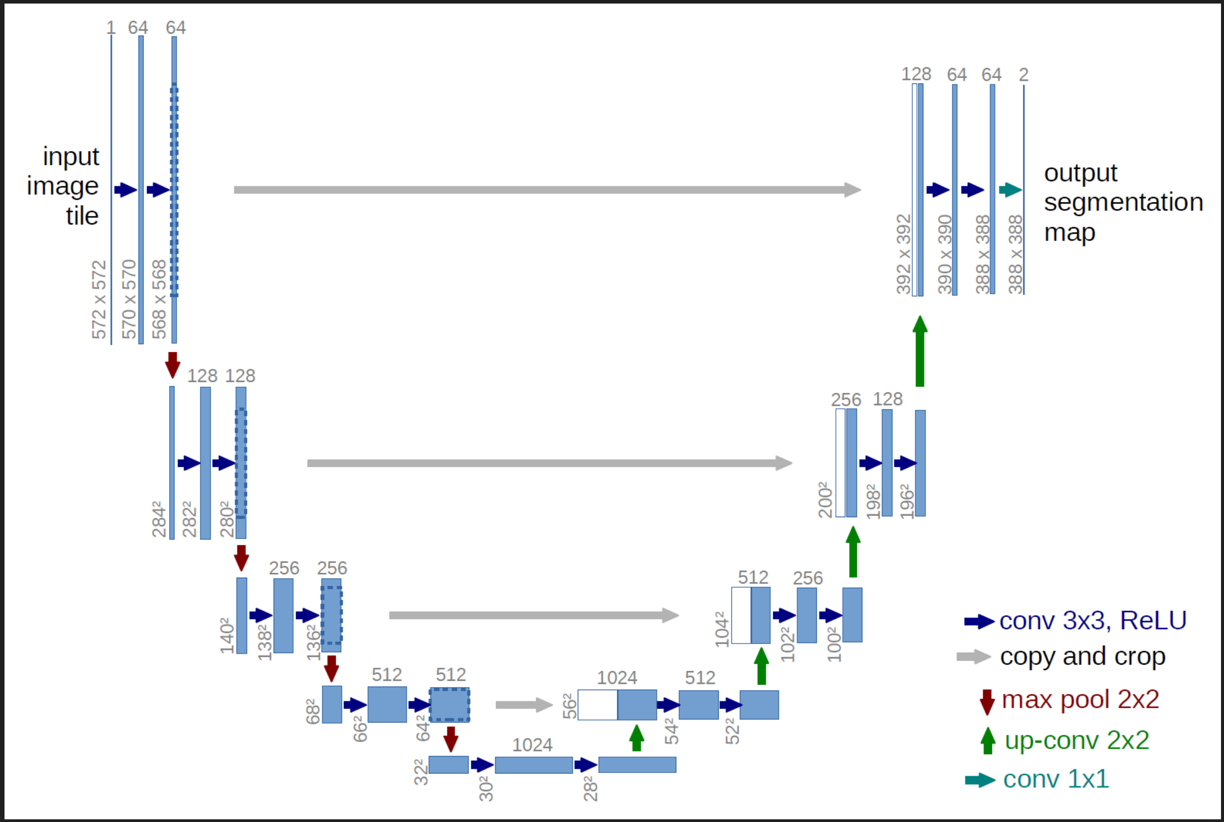
DEEP LEARNING



Modular nature.

Many building blocks.

SEGMENTATION DL ARCHITECTURES — U-NET



Ronneberger, Fischer, Brox. 2015. "U-Net: Convolutional Networks for Biomedical Image Segmentation." Medical Image Computing and Computer-Assisted Intervention (MICCAI), Springer, LNCS, Vol.9351: 234--241.

Back to the ultrasound image segmentation example.