Deep Learning: Overview and basics

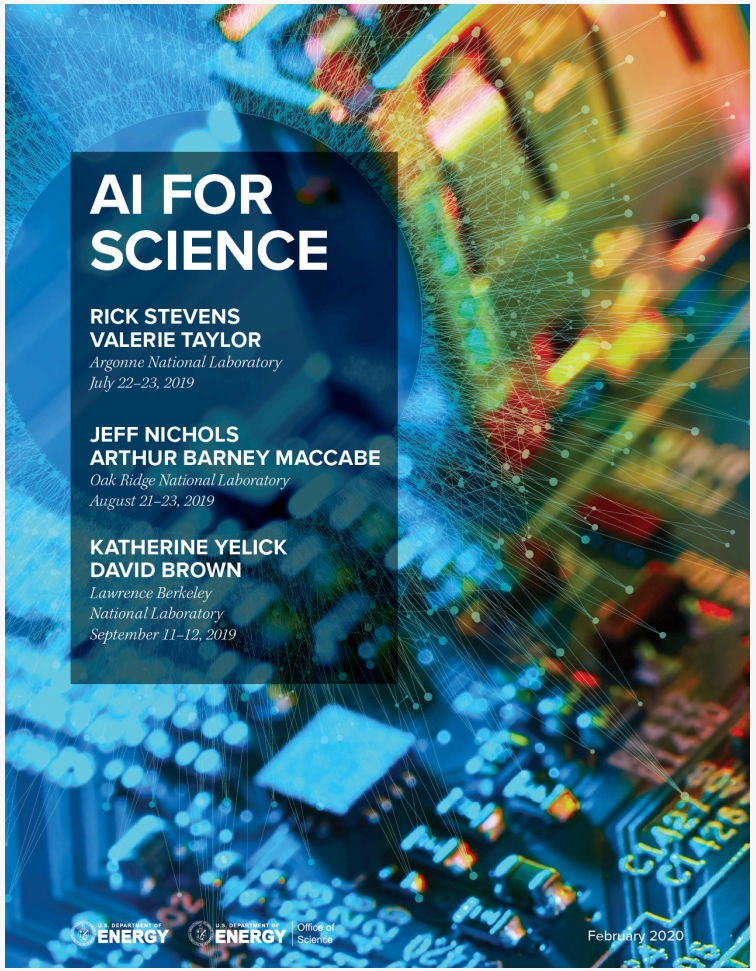
ATPESC 2020
Friday 2020-03-19

Kyle Gerard Felker
Postdoctoral Appointee, Aurora Early Science Program
Argonne Leadership Computing Facility
felker@anl.gov

Outline

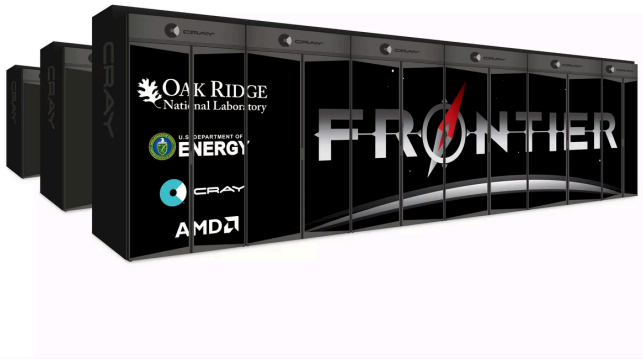
- High-level introduction to deep learning (25 min)
- Hands-on exercises (20 min)

AI for Science



NERSC: Perlmutter
AMD Epyc + NVIDIA A100s
(2020)

ALCF: Aurora
Intel Xeon + Xe GPUs
(2021)

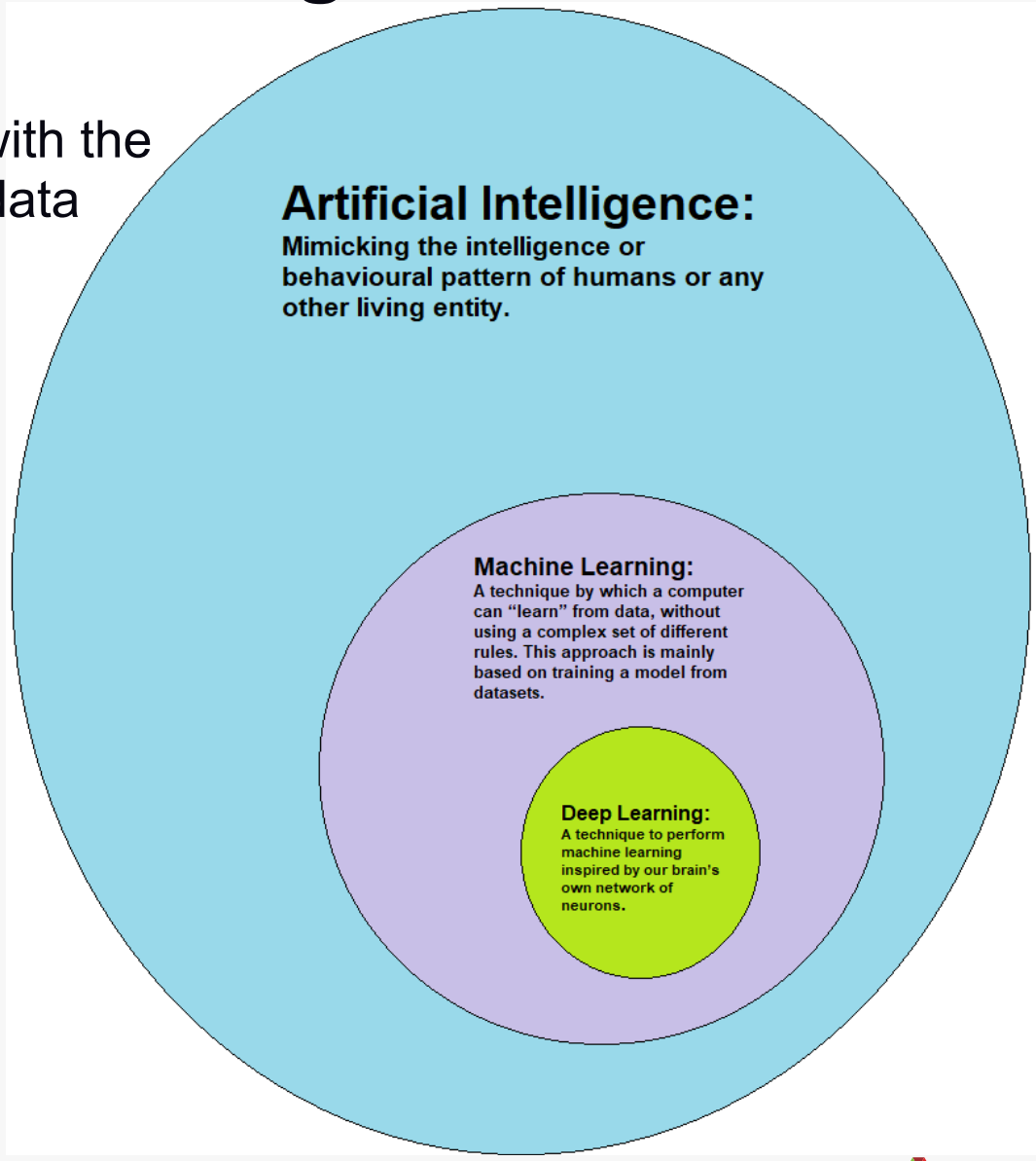


NERSC: Perlmutter
AMD Epyc + GPUs
(2021)

AI, machine learning, and deep learning

"Machine Learning is a subfield of artificial intelligence with the goal of developing algorithms capable of learning from data automatically."

Mehta, et al (2019)
"A high-bias, low-variance introduction to Machine Learning for physicists"



https://en.wikipedia.org/wiki/Deep_learning

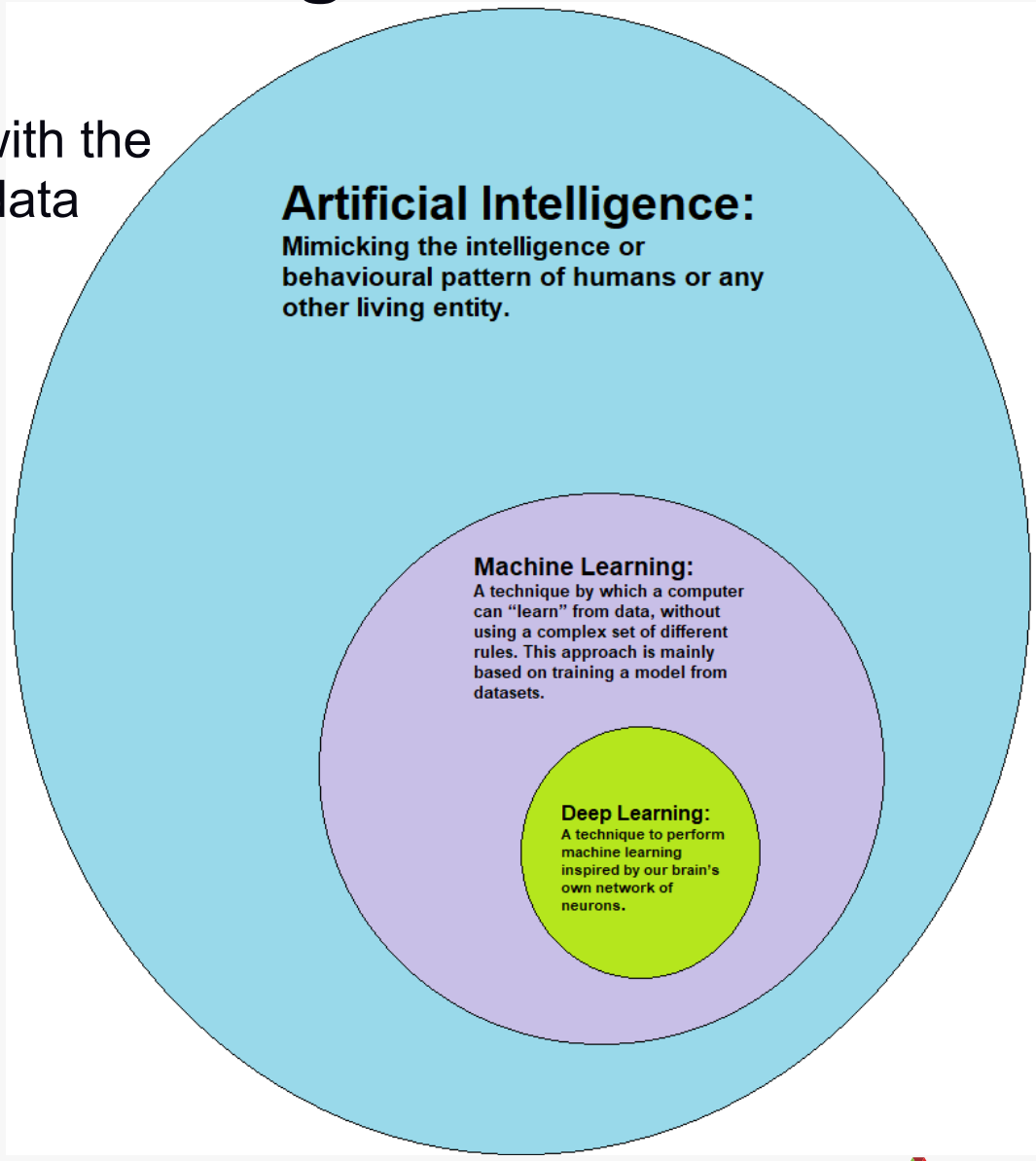
AI, machine learning, and deep learning

"Machine Learning is a subfield of artificial intelligence with the goal of developing algorithms capable of learning from data automatically."

Mehta, et al (2019)
"A high-bias, low-variance introduction to Machine Learning for physicists"

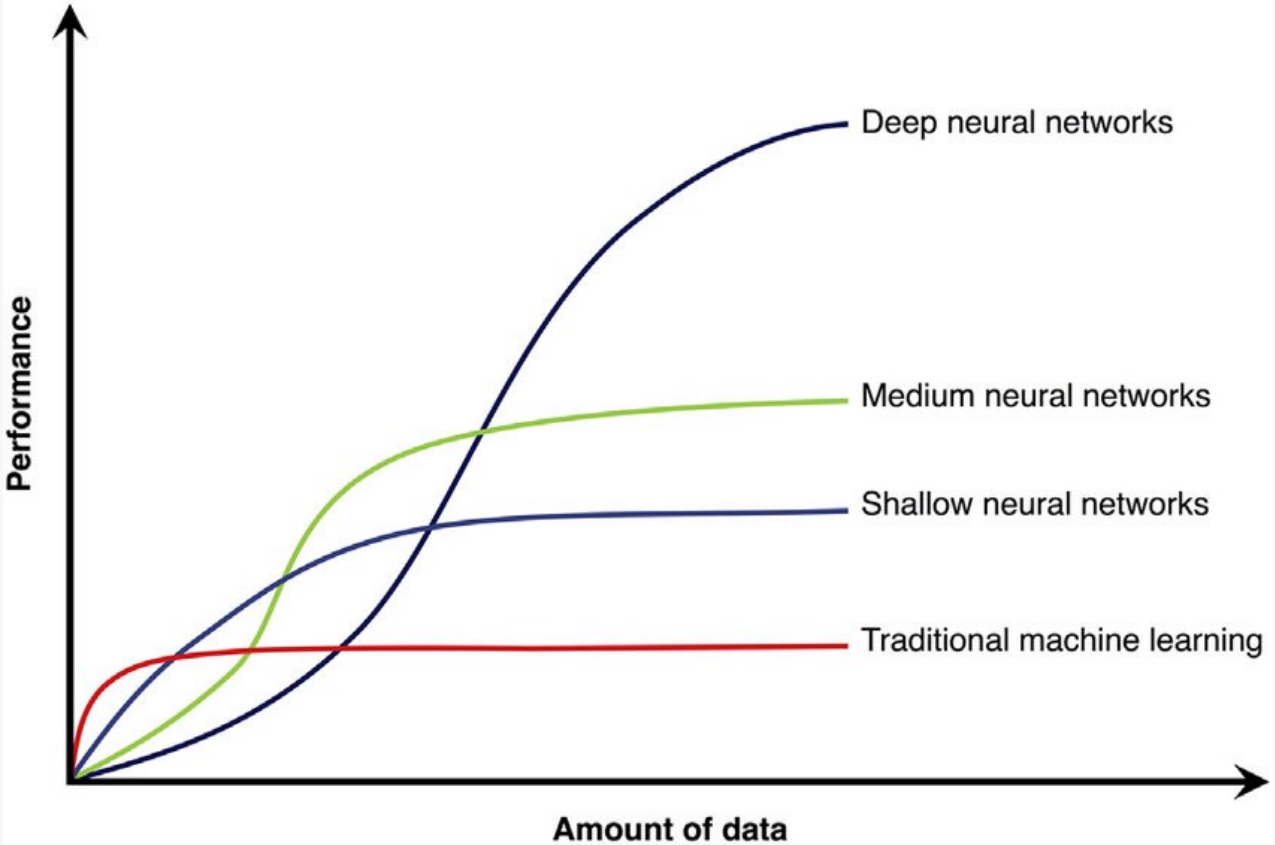
Machine learning algorithms:

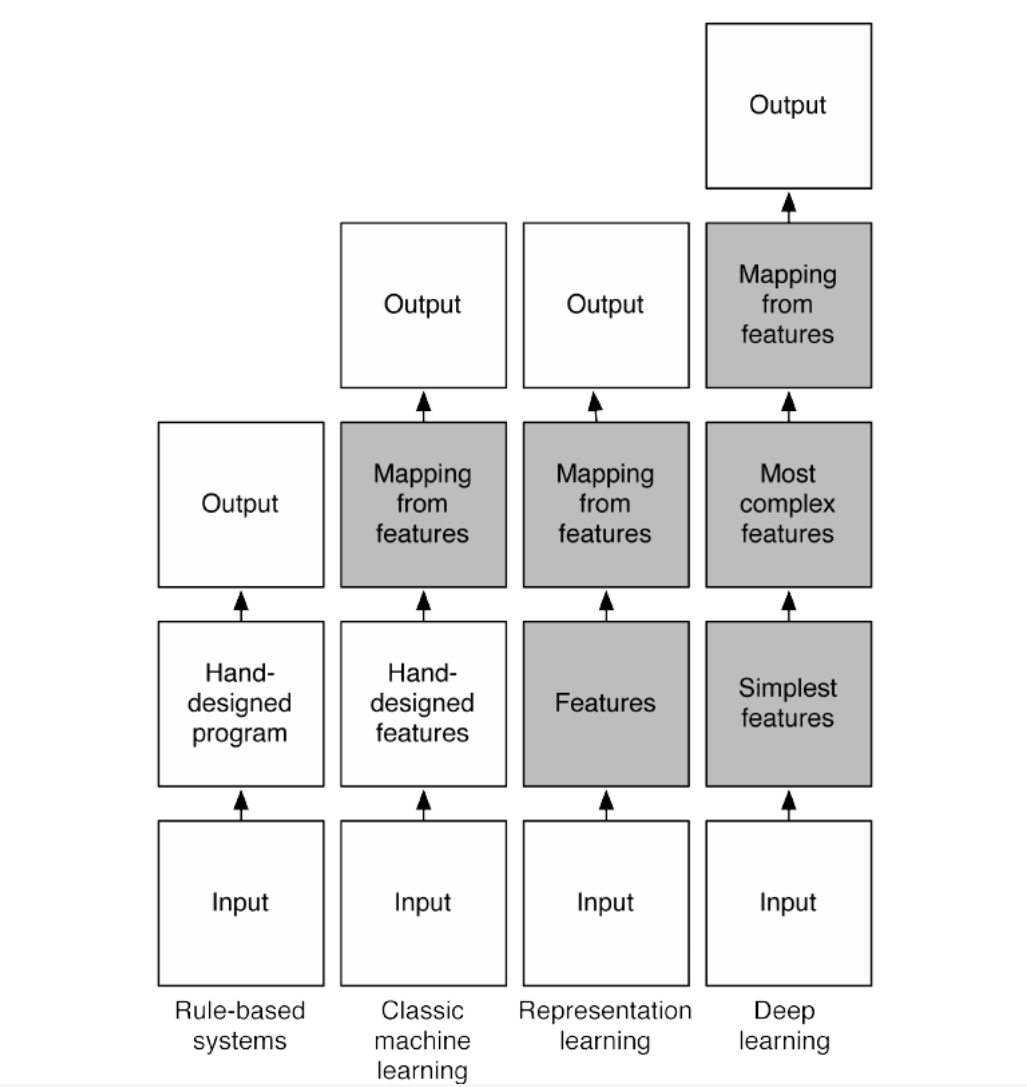
- can improve their accuracy given more data
- are capable of "teaching themselves"
- gain knowledge that is not programmed with an explicit set of rules



https://en.wikipedia.org/wiki/Deep_learning

Motivations for deep learning

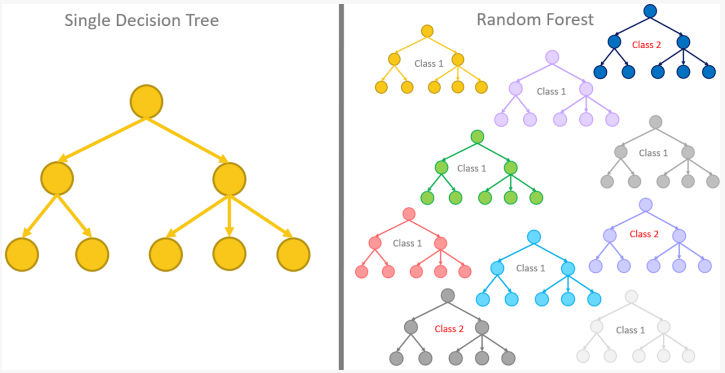




Categories of learning problems or paradigms

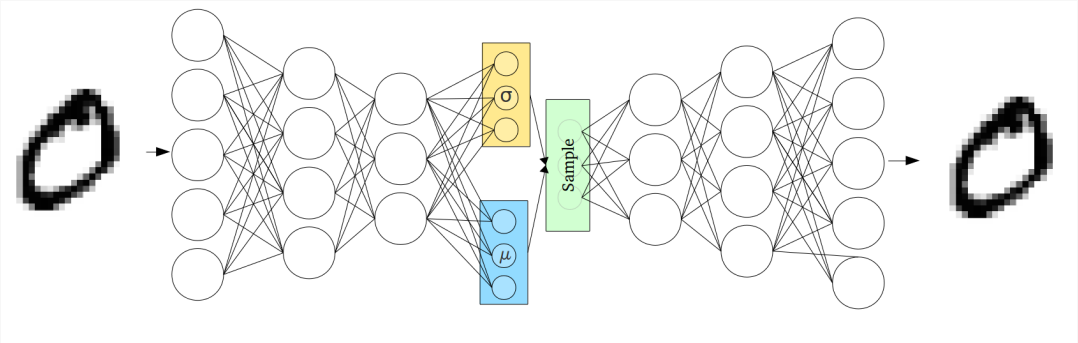
- Supervised learning (**this talk**)
  - Regression: output variable is continuous
  - Classification: output variable is discrete (categorical)
- Unsupervised learning (**Corey and Romit's talk**)
  - Clustering
  - Association
- Semi-supervised learning
  - *Mostly* unlabeled data
- Reinforcement learning

Examples of algorithms



Supervised: decision tree and random forests

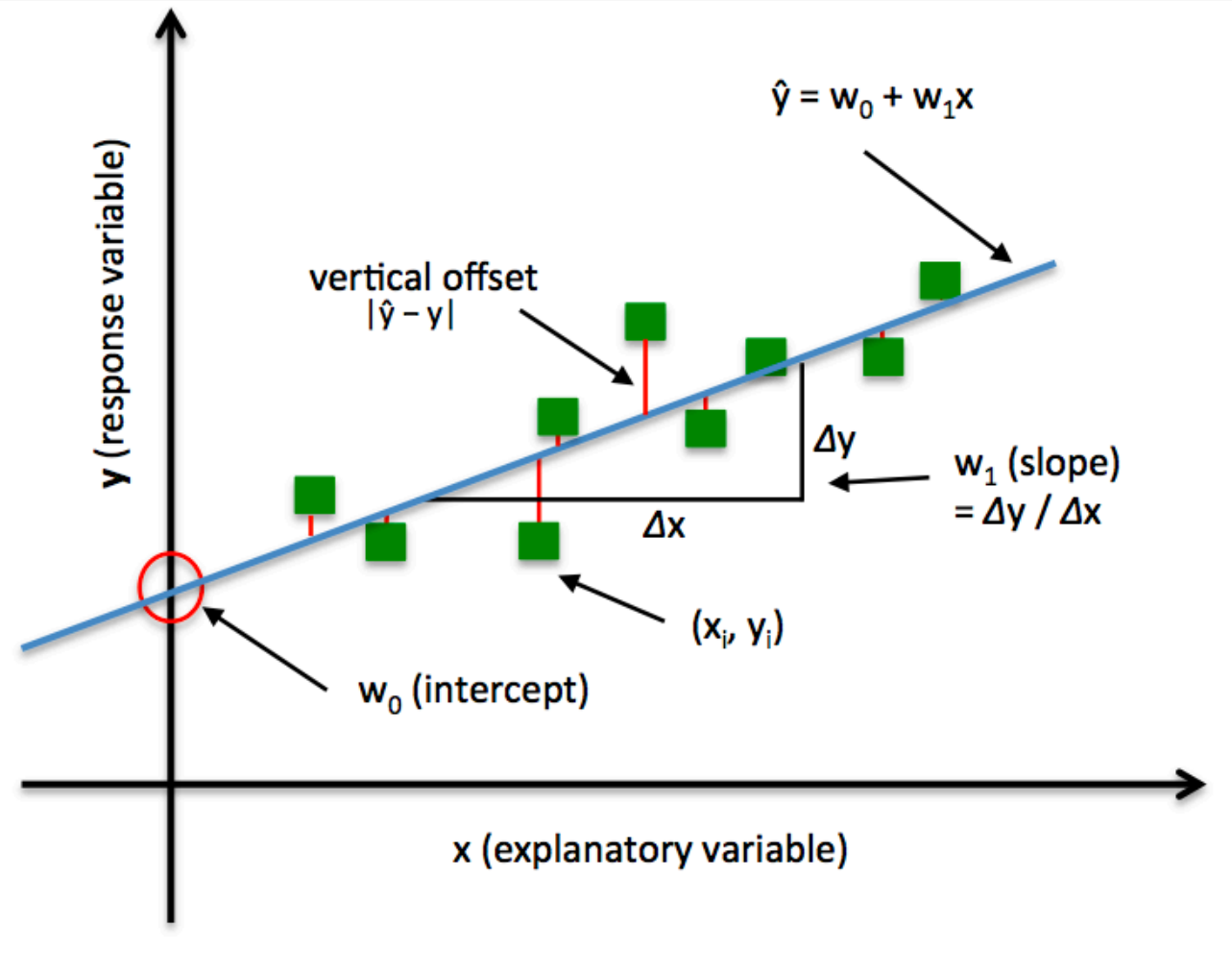
https://towardsdatascience.com/from-a-single-decision-tree-to-a-random-forest-b9523be65147



Unsupervised: Variational Autoencoder (VAE)

https://towardsdatascience.com/intuitively-understanding-variational-autoencoders-1bfe67eb5daf

Linear regression: ordinary least squares (OLS)



https://sebastianraschka.com/faq/docs/closed-form-vs-gd.html

Minimizes the Mean Squared Error (MSE), or quadratic loss:

$$J = \frac{1}{m} \sum_{i=1}^{m} (\hat{y}_i - y_i)^2$$

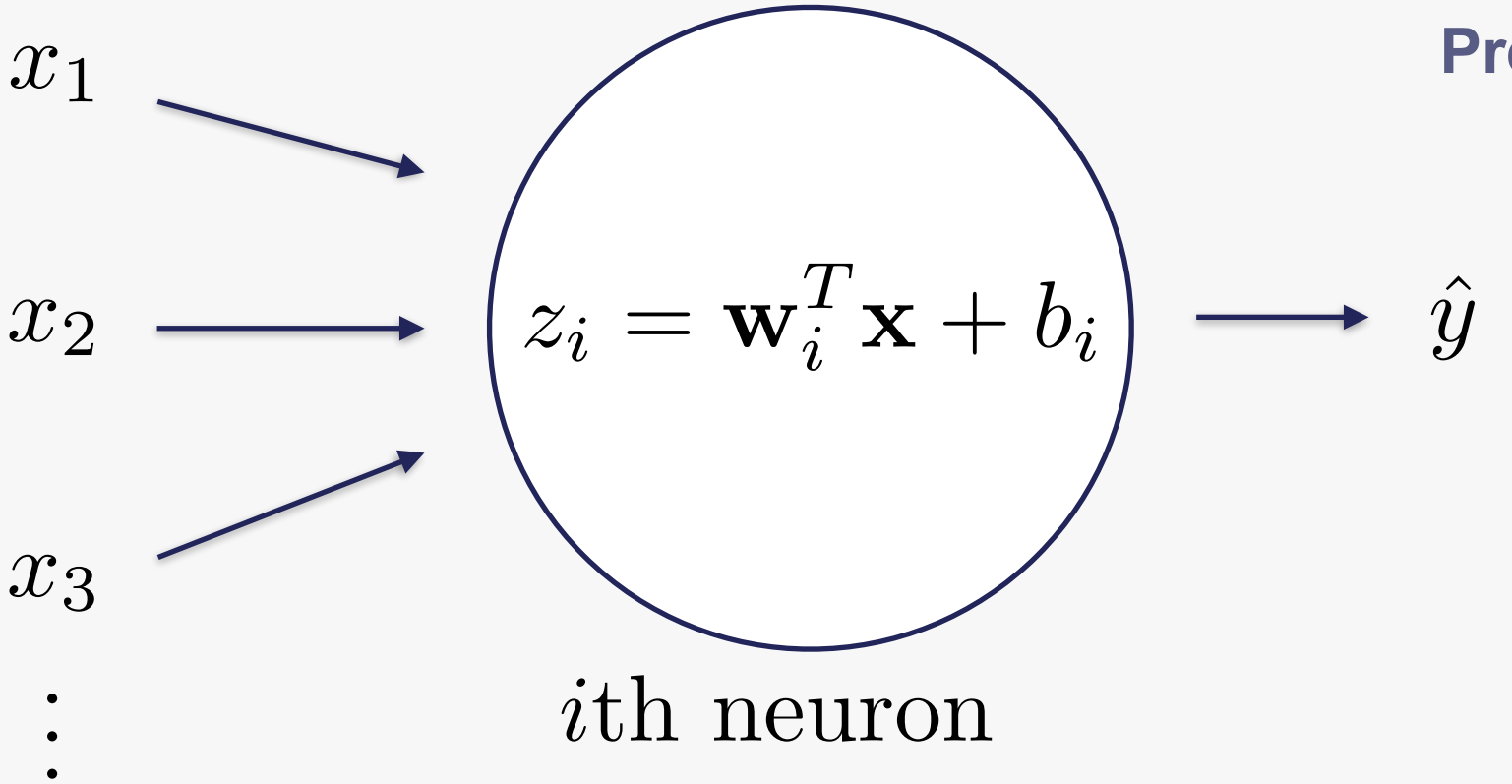
Closed-form solution exists; one-step procedure: **normal equation**

$$\mathbf{w}_i = \left(\mathbf{X}^T \mathbf{X}\right)^{-1} \mathbf{X}^T \mathbf{y}$$

Alternative solutions may be required or preferred in some cases

Neuron (linear regression)

Input features $\mathbf{x} \in \mathbb{R}^{n \times 1}$



Prediction (inference)

ith neuron

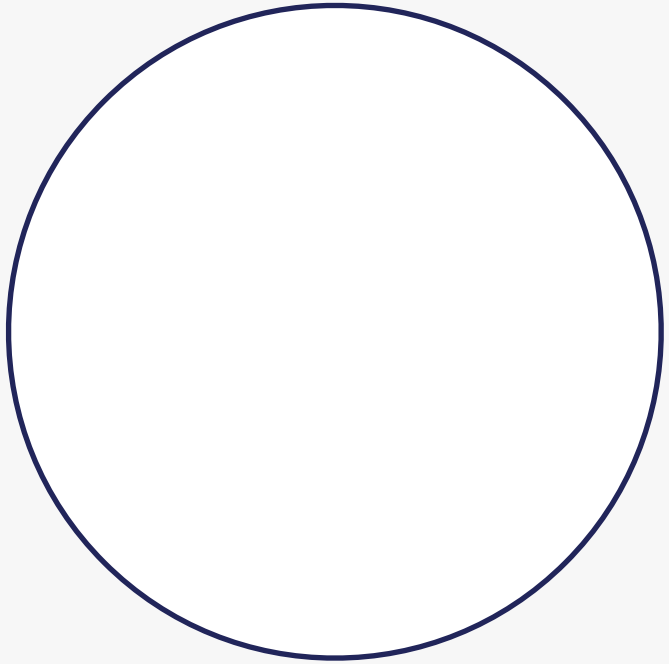
$z_i, b_i \in \mathbb{R}$

$\mathbf{w} \in \mathbb{R}^{n \times 1}$

*notation varies

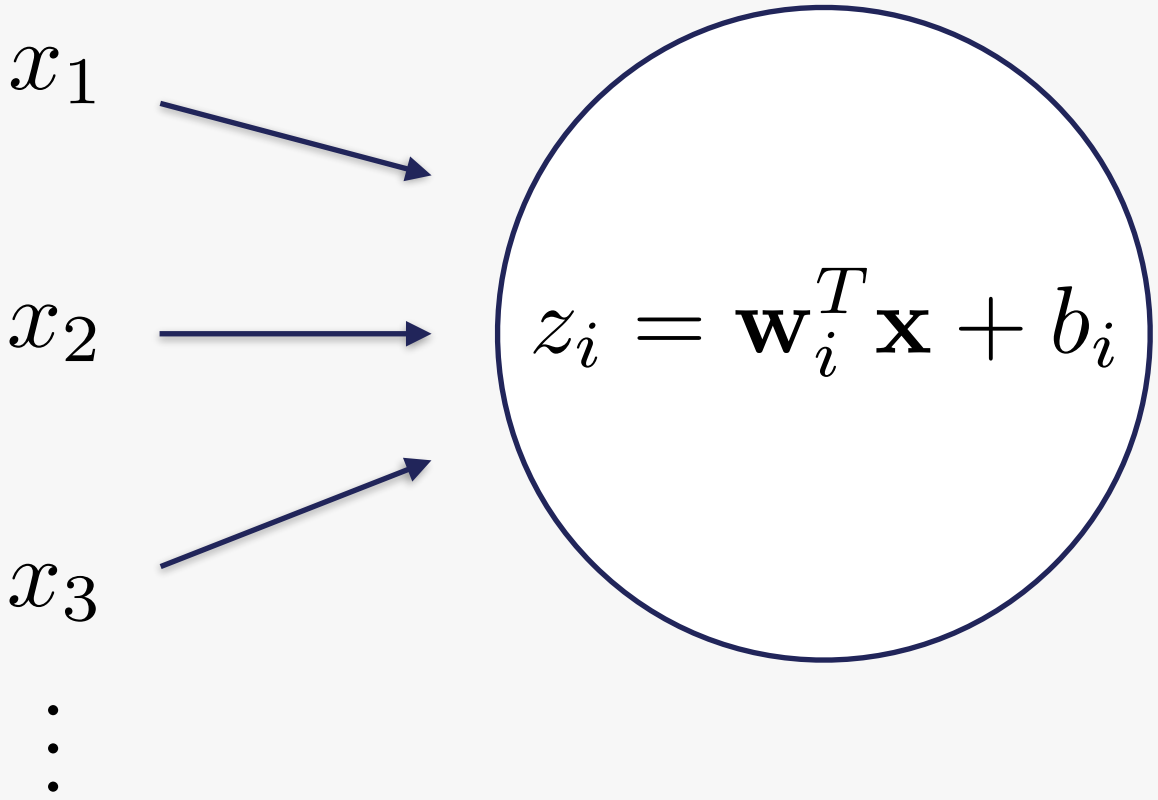
Forward pass: compute the loss

Given single labeled training example, $\left(\mathbf{x}^{(m)}, y^{(m)}\right)$



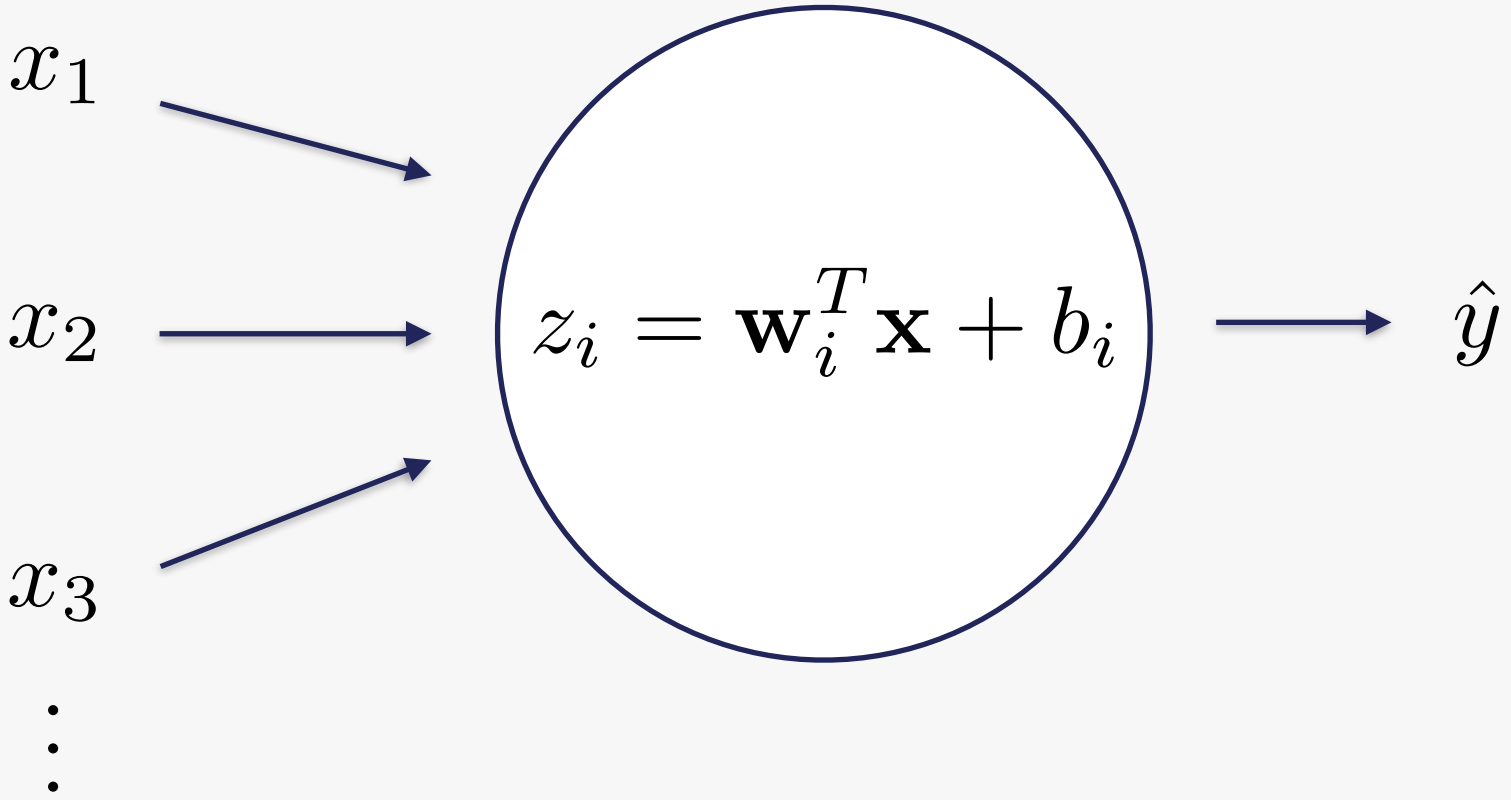
Forward pass: compute the loss

Given single labeled training example, $\left(\mathbf{x}^{(m)}, y^{(m)}\right)$



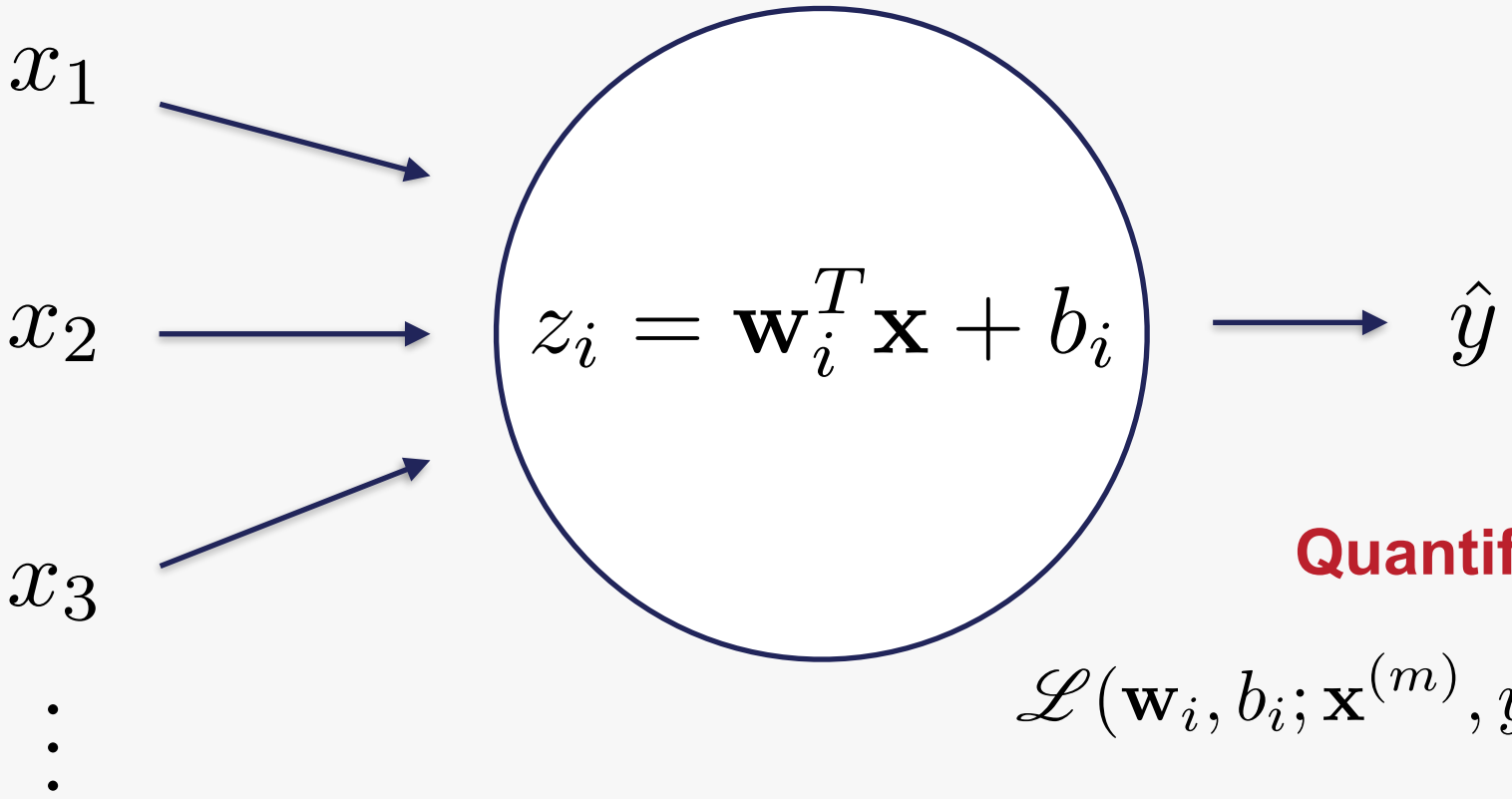
Forward pass: compute the loss

Given single labeled training example, $\left(\mathbf{x}^{(m)}, y^{(m)}\right)$



Forward pass: compute the loss

Given single labeled training example, $\left(\mathbf{x}^{(m)}, y^{(m)}\right)$



Quantify error (loss)

$$\mathscr{L}(\mathbf{w}_i, b_i; \mathbf{x}^{(m)}, y^{(m)}) = \left(\hat{y} - y^{(m)}\right)^2$$

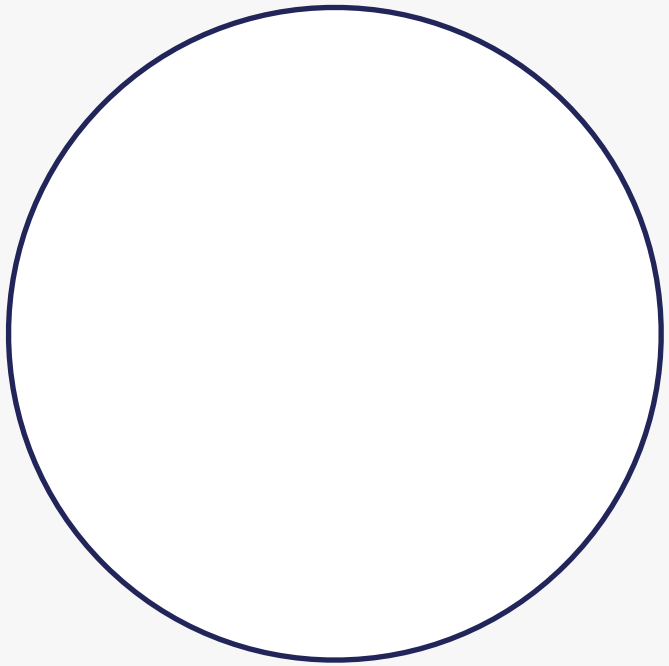
Backward pass: compute the derivatives

Given single labeled training example, $\left(\mathbf{x}^{(m)}, y^{(m)}\right)$

$$\mathscr{L}(\mathbf{w}_i, b_i; \mathbf{x}^{(m)}, y^{(m)}) = \left(\hat{y} - y^{(m)}\right)^2$$

Backward pass: compute the derivatives

Given single labeled training example, $\left(\mathbf{x}^{(m)}, y^{(m)}\right)$



$$\mathscr{L}(\mathbf{w}_i, b_i; \mathbf{x}^{(m)}, y^{(m)}) = \left(\hat{y} - y^{(m)}\right)^2$$

$$\frac{\partial \mathscr{L}}{\partial z_i}$$

Backward pass: compute the derivatives

Given single labeled training example, $\left(\mathbf{x}^{(m)}, y^{(m)}\right)$

$$\frac{\partial \mathscr{L}}{\partial \mathbf{w}_i}$$

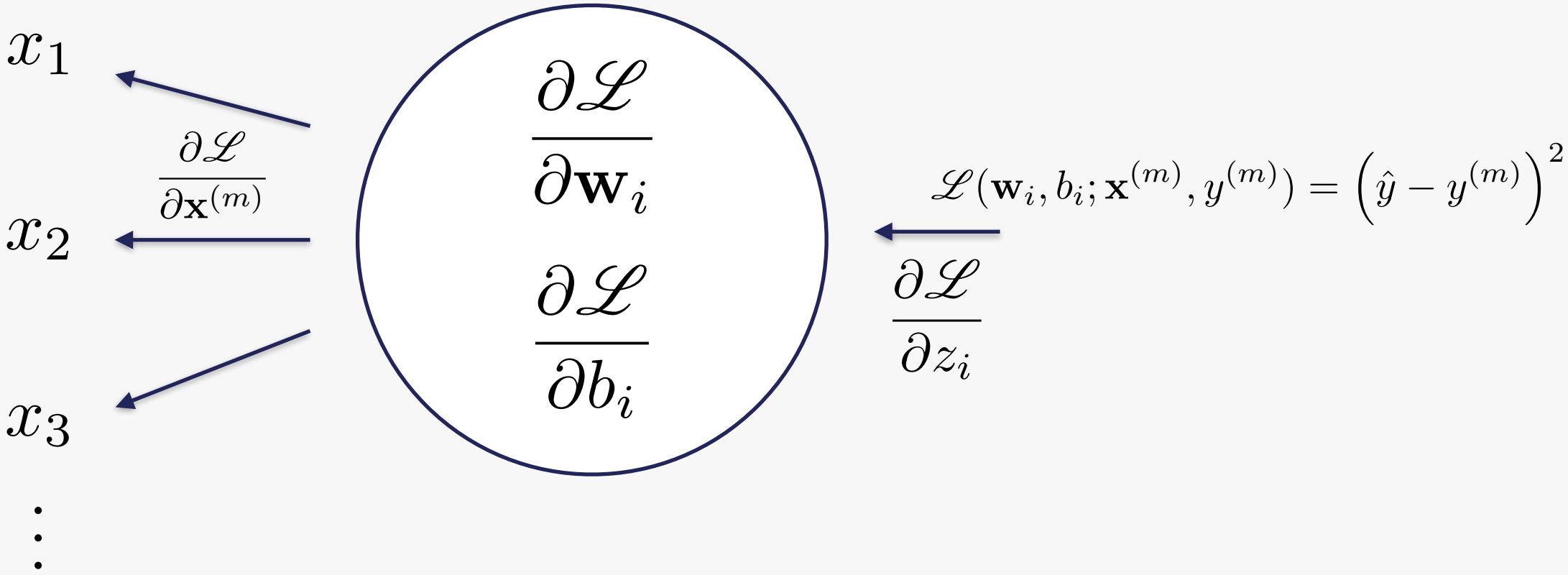
$$\frac{\partial \mathscr{L}}{\partial b_i}$$

$$\mathscr{L}(\mathbf{w}_i, b_i; \mathbf{x}^{(m)}, y^{(m)}) = \left(\hat{y} - y^{(m)}\right)^2$$

$$\frac{\partial \mathscr{L}}{\partial z_i}$$

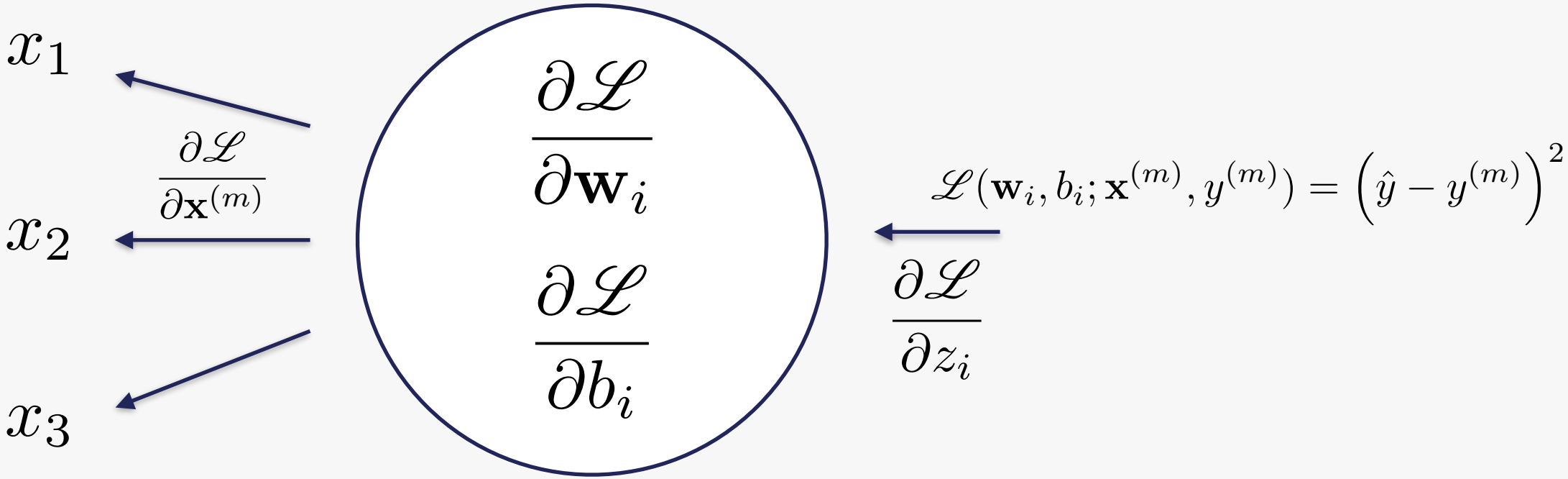
Backward pass: compute the derivatives

Given single labeled training example, $\left(\mathbf{x}^{(m)}, y^{(m)}\right)$



Backward pass: compute the derivatives

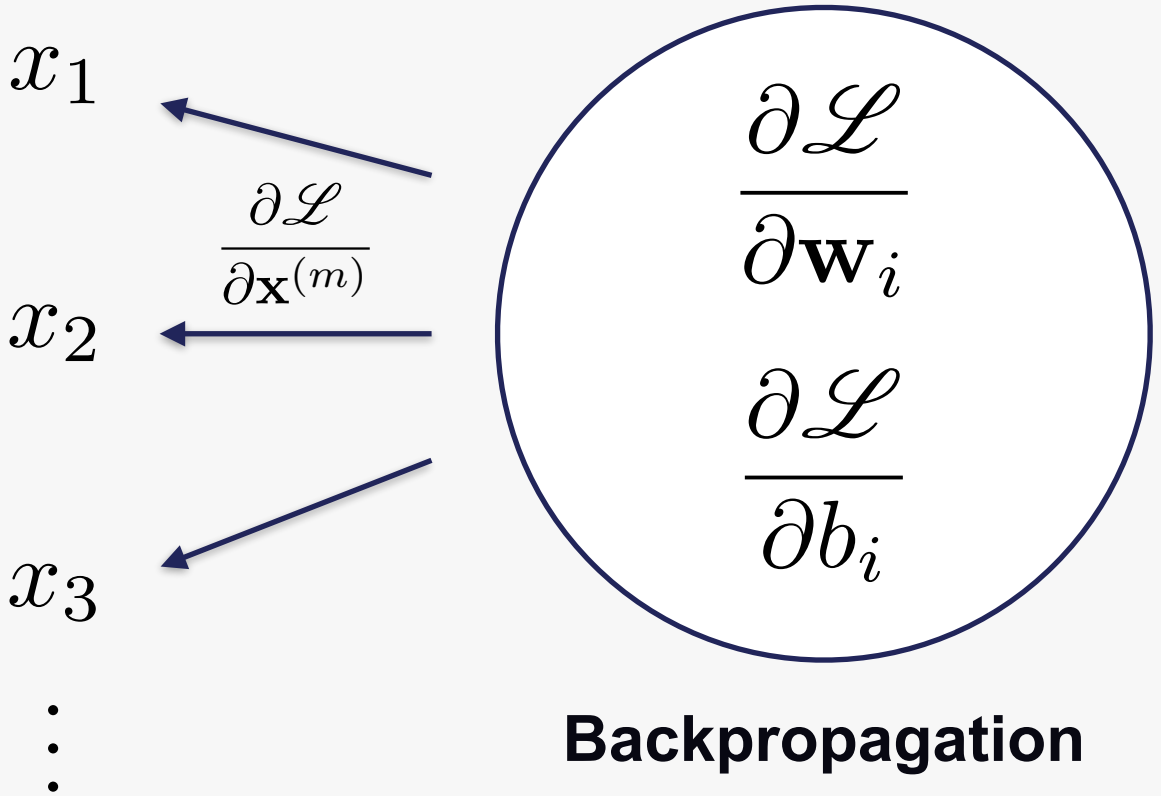
Given single labeled training example, $\left(\mathbf{x}^{(m)}, y^{(m)}\right)$



Backpropagation
=
chain rule

Backward pass: compute the derivatives

Given single labeled training example, $\left(\mathbf{x}^{(m)}, y^{(m)}\right)$



$$\mathscr{L}(\mathbf{w}_i, b_i; \mathbf{x}^{(m)}, y^{(m)}) = \left(\hat{y} - y^{(m)}\right)^2$$

$$\frac{\partial \mathscr{L}}{\partial z_i}$$

Backpropagation
=
chain rule

$$\frac{\partial \mathscr{L}}{\partial b_i} = \frac{\partial \mathscr{L}}{\partial z_i}\frac{\partial z_i}{\partial b_i}$$

$$\frac{\partial \mathscr{L}}{\partial \mathbf{w}_i} = \frac{\partial \mathscr{L}}{\partial z_i}\frac{\partial z_i}{\partial \mathbf{w}_i}$$

Backward pass: compute the derivatives

Given single labeled training example, $\left(\mathbf{x}^{(m)}, y^{(m)}\right)$

x_1

x_2

x_3

$\vdots$

$$\frac{\partial \mathscr{L}}{\partial \mathbf{x}^{(m)}}$$

$$\frac{\partial \mathscr{L}}{\partial \mathbf{w}_i}$$

$$\frac{\partial \mathscr{L}}{\partial b_i}$$

$$\mathscr{L}(\mathbf{w}_i, b_i; \mathbf{x}^{(m)}, y^{(m)}) = \left(\hat{y} - y^{(m)}\right)^2$$

$$\frac{\partial \mathscr{L}}{\partial z_i}$$

Backpropagation
=
chain rule

$$\frac{\partial \mathscr{L}}{\partial b_i} = \frac{\partial \mathscr{L}}{\partial z_i}\frac{\partial z_i}{\partial b_i}$$

$$\frac{\partial \mathscr{L}}{\partial \mathbf{w}_i} = \frac{\partial \mathscr{L}}{\partial z_i}\frac{\partial z_i}{\partial \mathbf{w}_i}$$

$$\frac{\partial \mathscr{L}}{\partial z_i} = 2\left(z_i - y\right)$$

$$\frac{\partial z_i}{\partial b_i} = 1$$

$$\frac{\partial z}{\partial \mathbf{w}_i} = \mathbf{x}^{(m)}$$

Training neuron weights with gradient descent

Let $\theta \equiv \{\mathbf{w}_i, b_i\}$ trainable parameters

Cost: average loss over training examples

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \mathscr{L}(\theta; \mathbf{x}^{(m)}, y^{(m)})$$

- Initialize θ randomly
- For N epochs
  - For each batch of training examples $\{(x_0, y_0), \ldots, (x_b, y_b)\}$:
    - compute loss gradient: $\frac{\partial J(\theta)}{\partial \theta} = \frac{1}{B}\sum_{i=1}^{N} \frac{\partial J^i(\theta)}{\partial \theta}$
    - update θ with update rule:
      $\theta := \theta - \eta \frac{\partial J(\theta)}{\partial \theta}$

Training neuron weights with gradient descent

Let $\theta \equiv \{\mathbf{w}_i, b_i\}$ trainable parameters

Cost: average loss over training examples

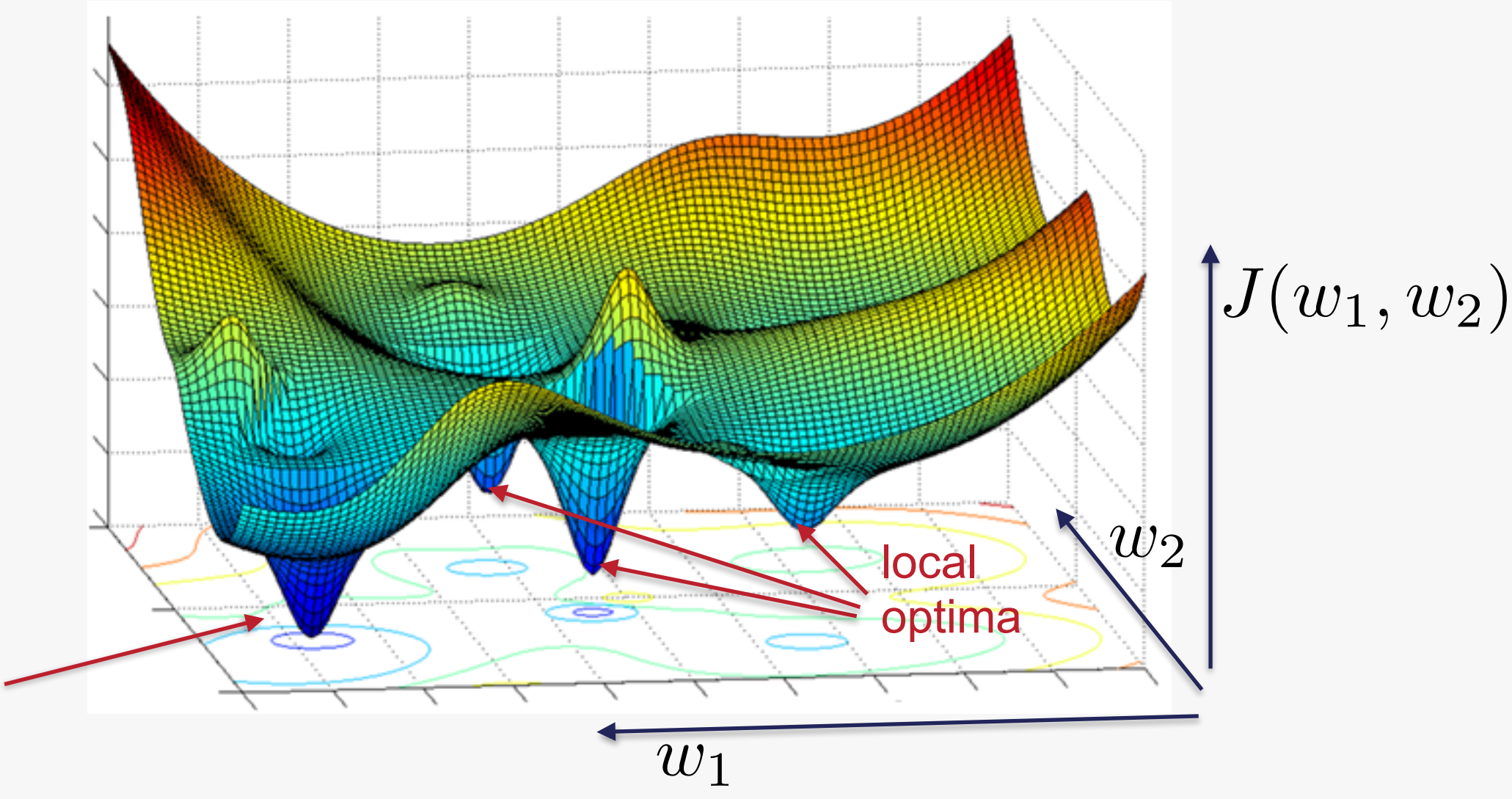
$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \mathscr{L}(\theta; \mathbf{x}^{(m)}, y^{(m)})$$

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta)$$

$\eta \equiv$ learning rate

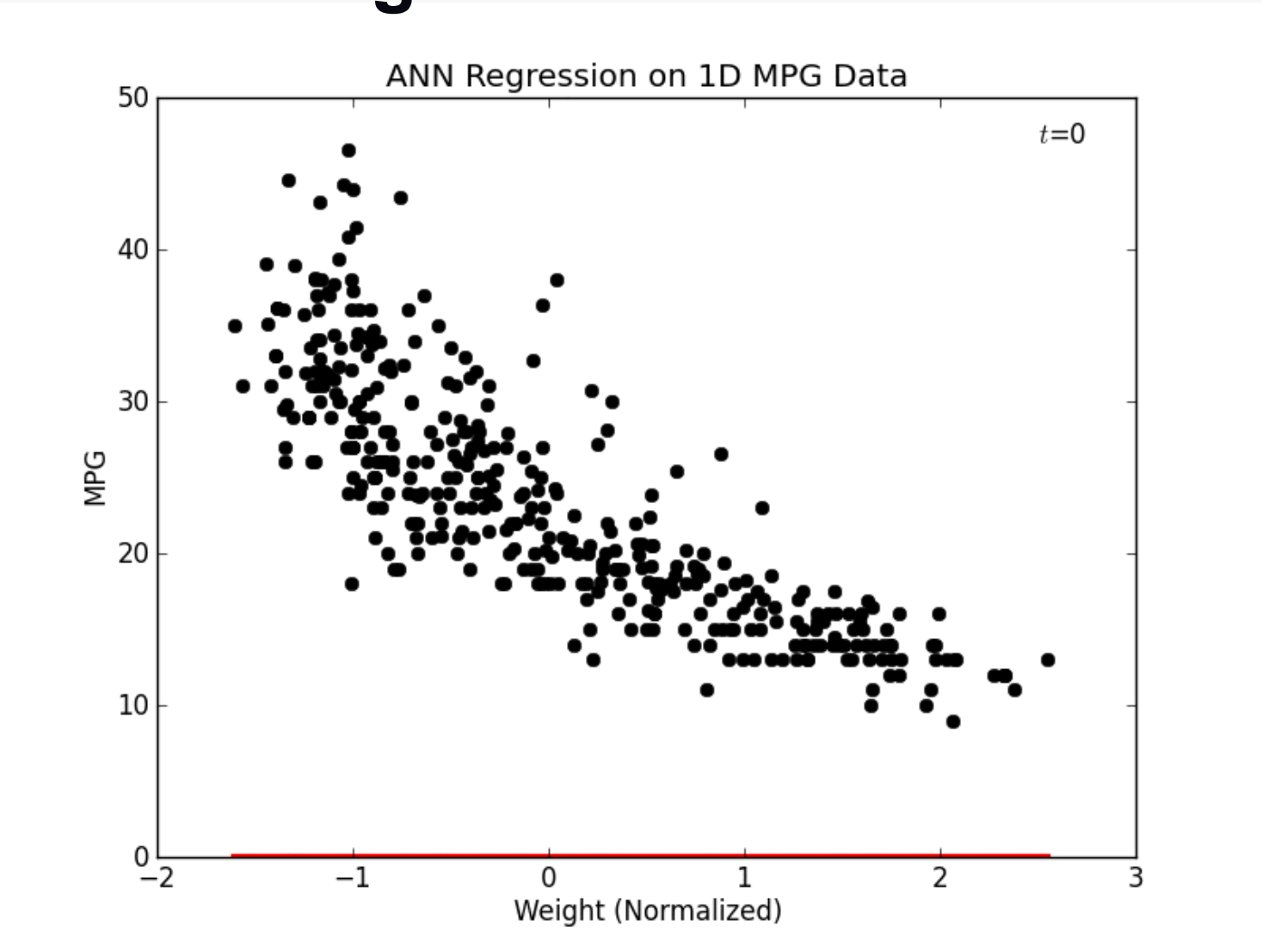
- Initialize θ randomly
- For N epochs
  - For each batch of training examples $\{(x_0, y_0), \ldots, (x_b, y_b)\}$:
    - compute loss gradient: $\frac{\partial J(\theta)}{\partial \theta} = \frac{1}{B}\sum_{i=1}^{N} \frac{\partial J^i(\theta)}{\partial \theta}$
    - update θ with update rule: $\theta := \theta - \eta \frac{\partial J(\theta)}{\partial \theta}$

Global vs. local minima



http://firsttimeprogrammer.blogspot.com/2014/09/multivariable-gradient-descent.html

OLS solved with gradient descent

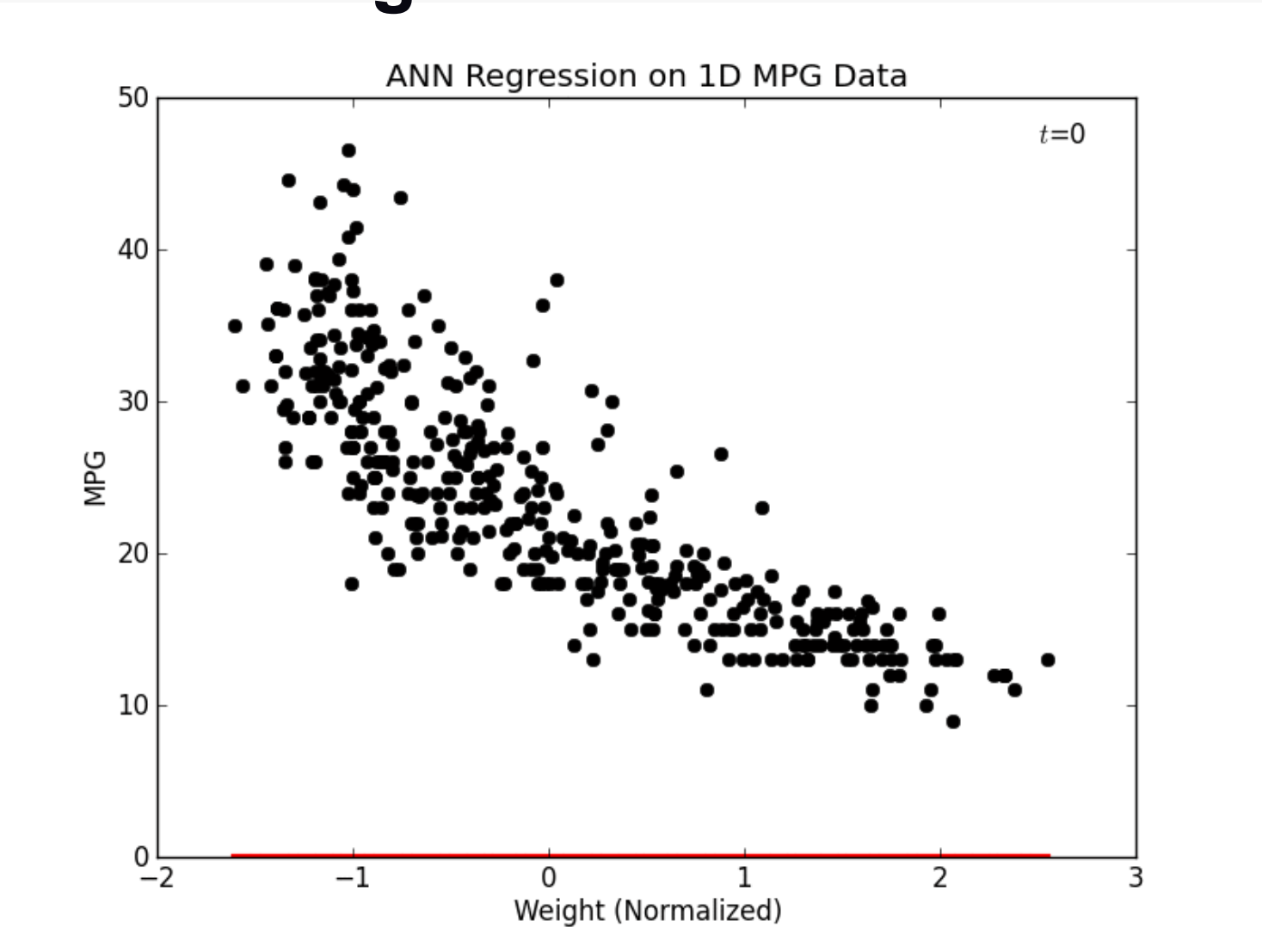


http://www.briandolhansky.com/blog/artificial-neural-networks-linear-regression-part-1

t = epoch

= neuron sees **all** samples in the training set

OLS solved with gradient descent

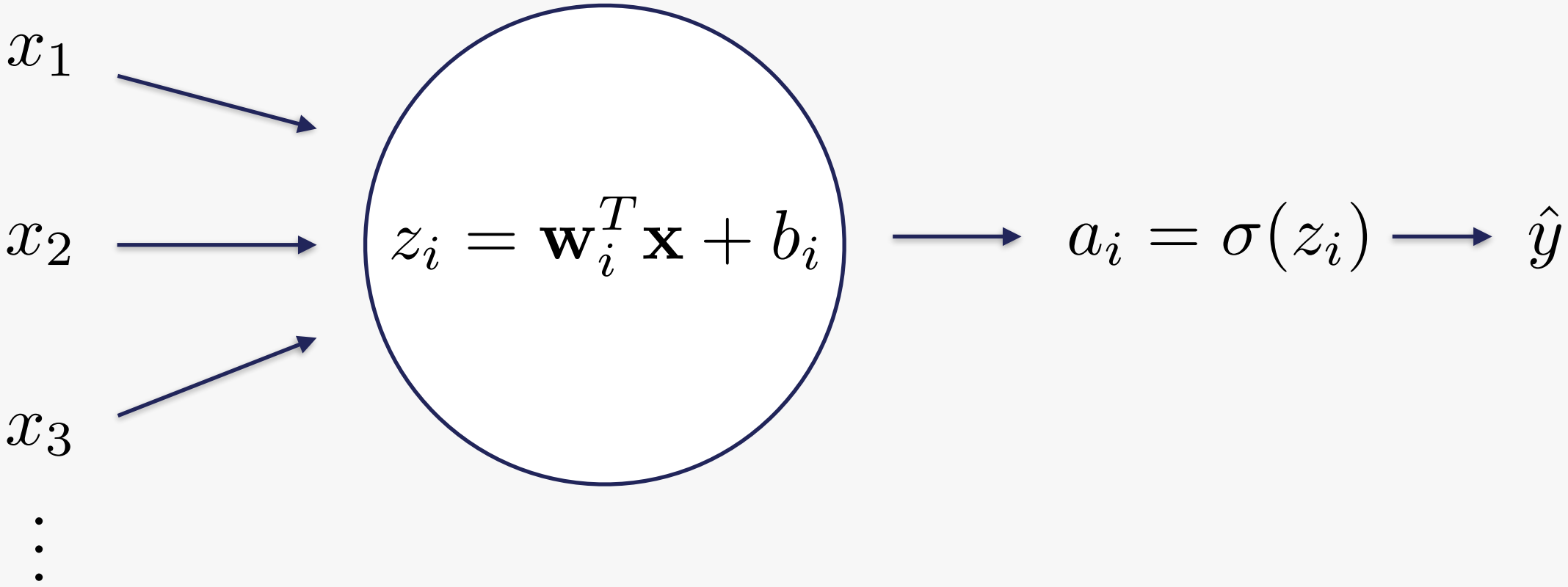


http://www.briandolhansky.com/blog/artificial-neural-networks-linear-regression-part-1

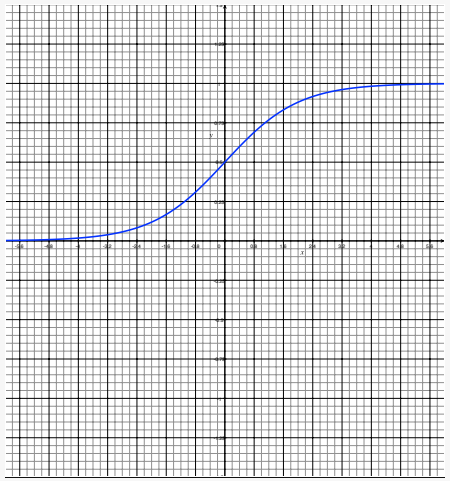
t = epoch

= neuron sees **all** samples in the training set

Neuron (generally)



Neuron (generally)



$$x_1$$

$$x_2$$

$$x_3$$

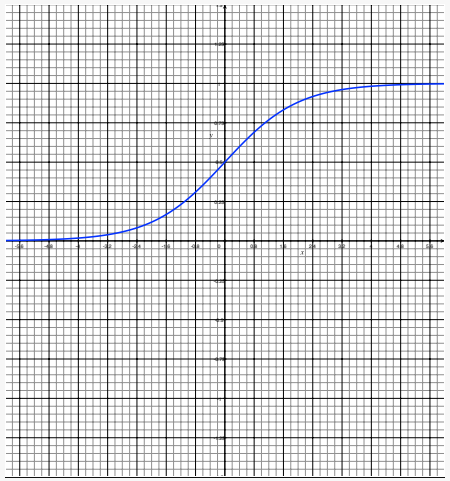
$$\vdots$$

$$z_i = \mathbf{w}_i^T \mathbf{x} + b_i \longrightarrow a_i = \sigma(z_i) \longrightarrow \hat{y}$$

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

= sigmoid = logistic function

Neuron (generally)



$$x_1$$

$$x_2$$

$$x_3$$

$$\vdots$$

$$z_i = \mathbf{w}_i^T \mathbf{x} + b_i \longrightarrow a_i = \sigma(z_i) \longrightarrow \hat{y}$$

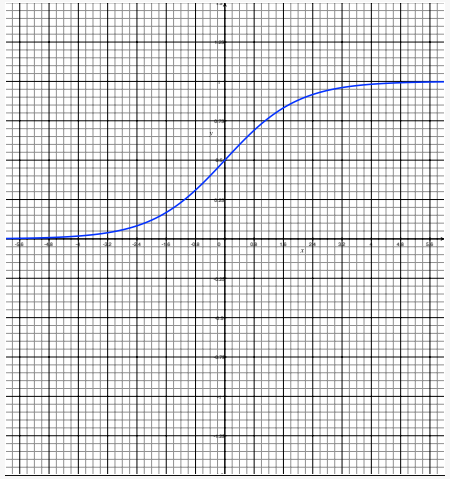
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

= sigmoid = logistic function

<u>Binary cross-entropy loss function:</u>

$$\mathscr{L}(\hat{y}, y) = -\left(y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})\right)$$

Neuron (generally)



$$x_1$$

$$x_2$$

$$x_3$$

$$\vdots$$

$$z_i = \mathbf{w}_i^T \mathbf{x} + b_i \longrightarrow a_i = \sigma(z_i) \longrightarrow \hat{y}$$

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

= sigmoid = logistic function

= Logistic regression

<u>Binary cross-entropy loss function:</u>

$$\mathscr{L}(\hat{y}, y) = -\left(y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})\right)$$

Nonlinear activation functions

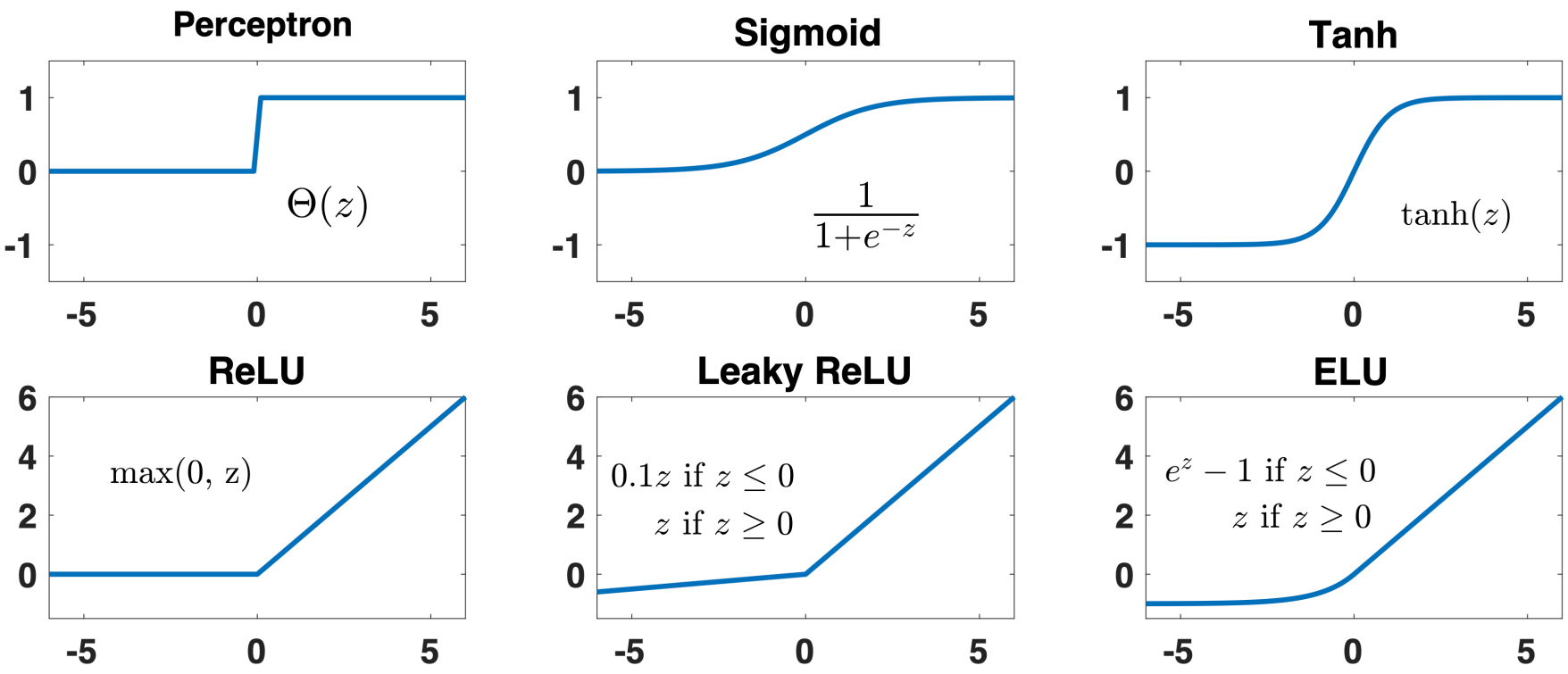
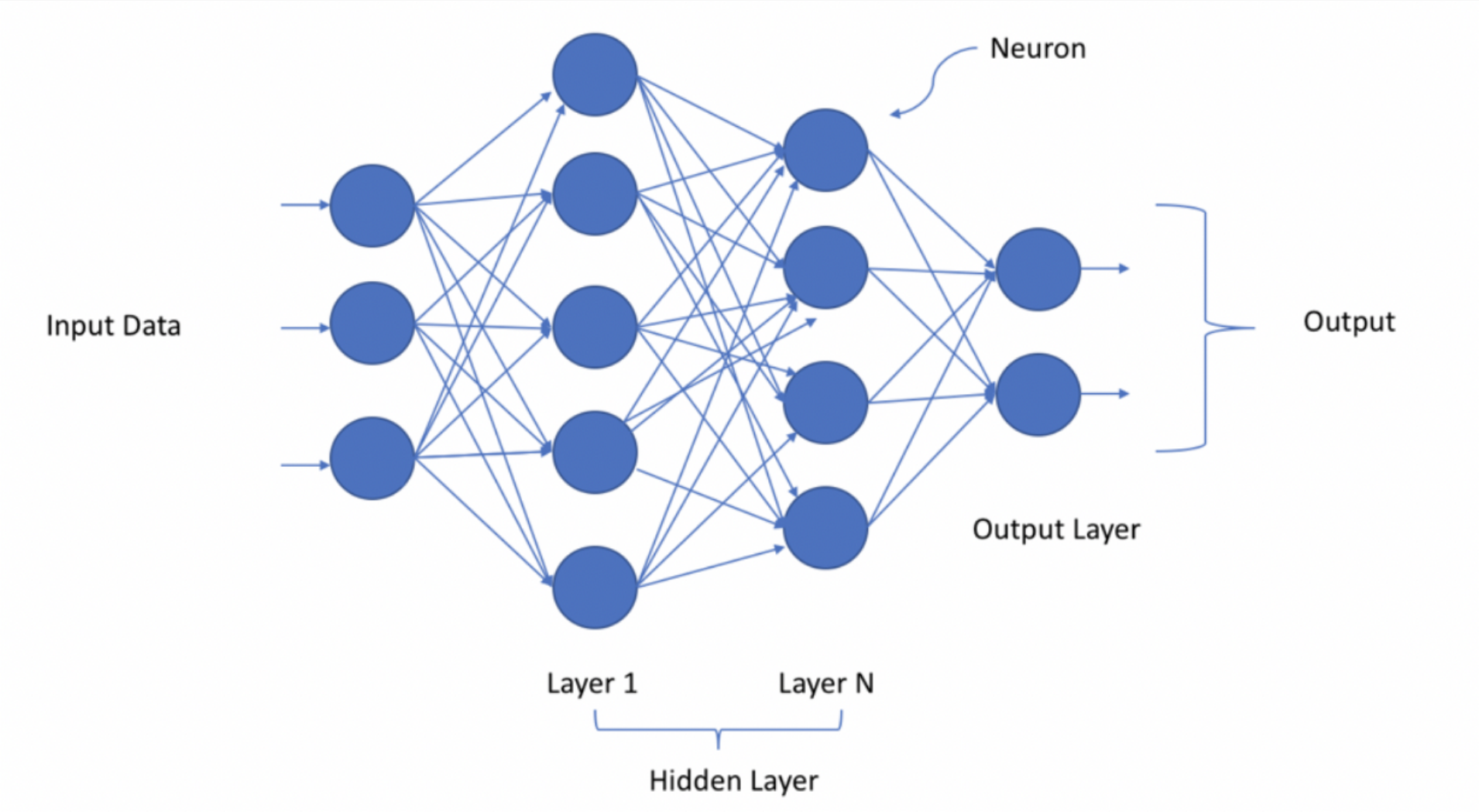


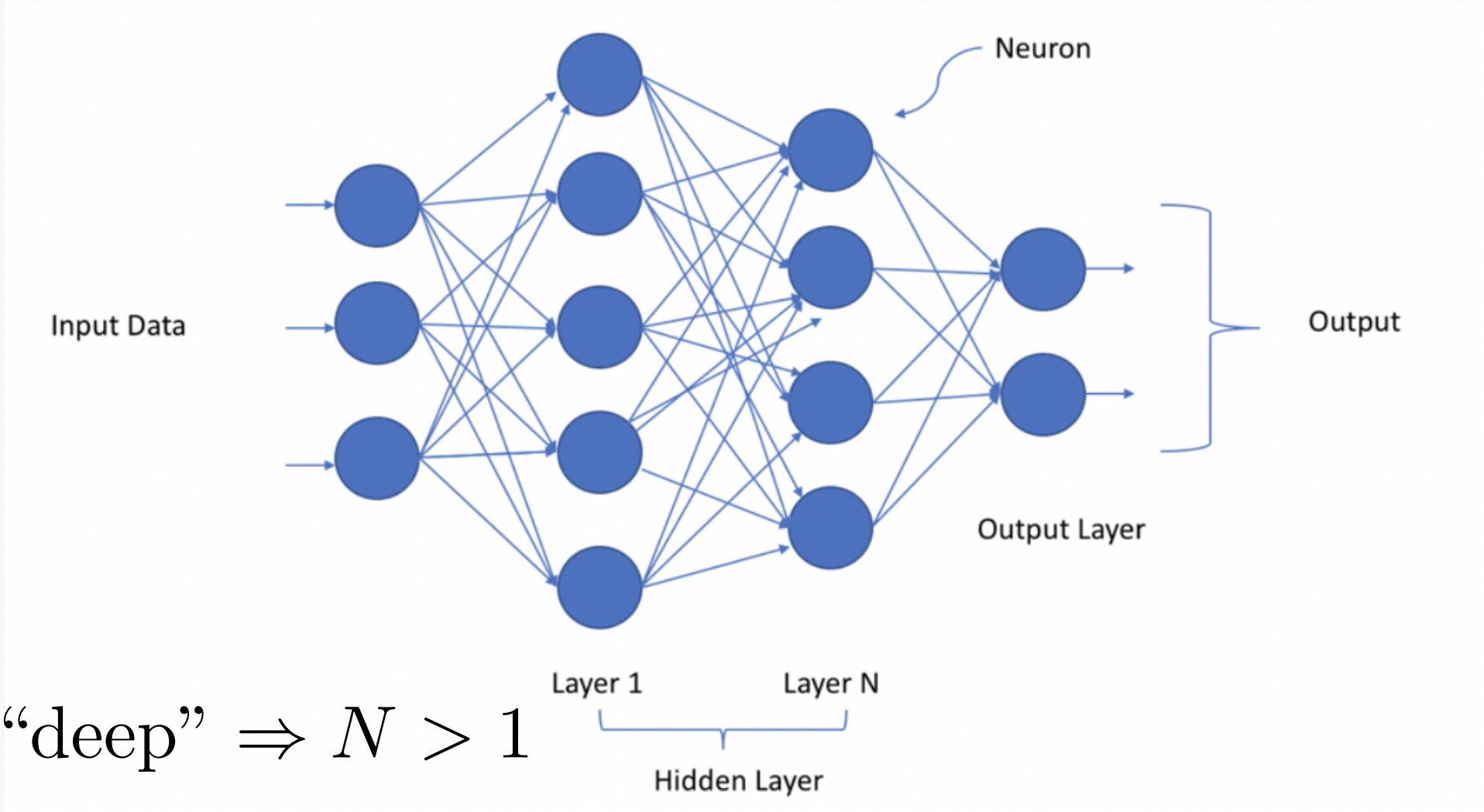
FIG. 36 **Possible non-linear activation functions for neurons.** In modern DNNs, it has become common to use non-linear functions that do not saturate for large inputs (bottom row) rather than saturating functions (top row).

Deep neural network (DNN)



https://towardsdatascience.com/a-laymans-guide-to-deep-neural-networks-ddcea24847fb

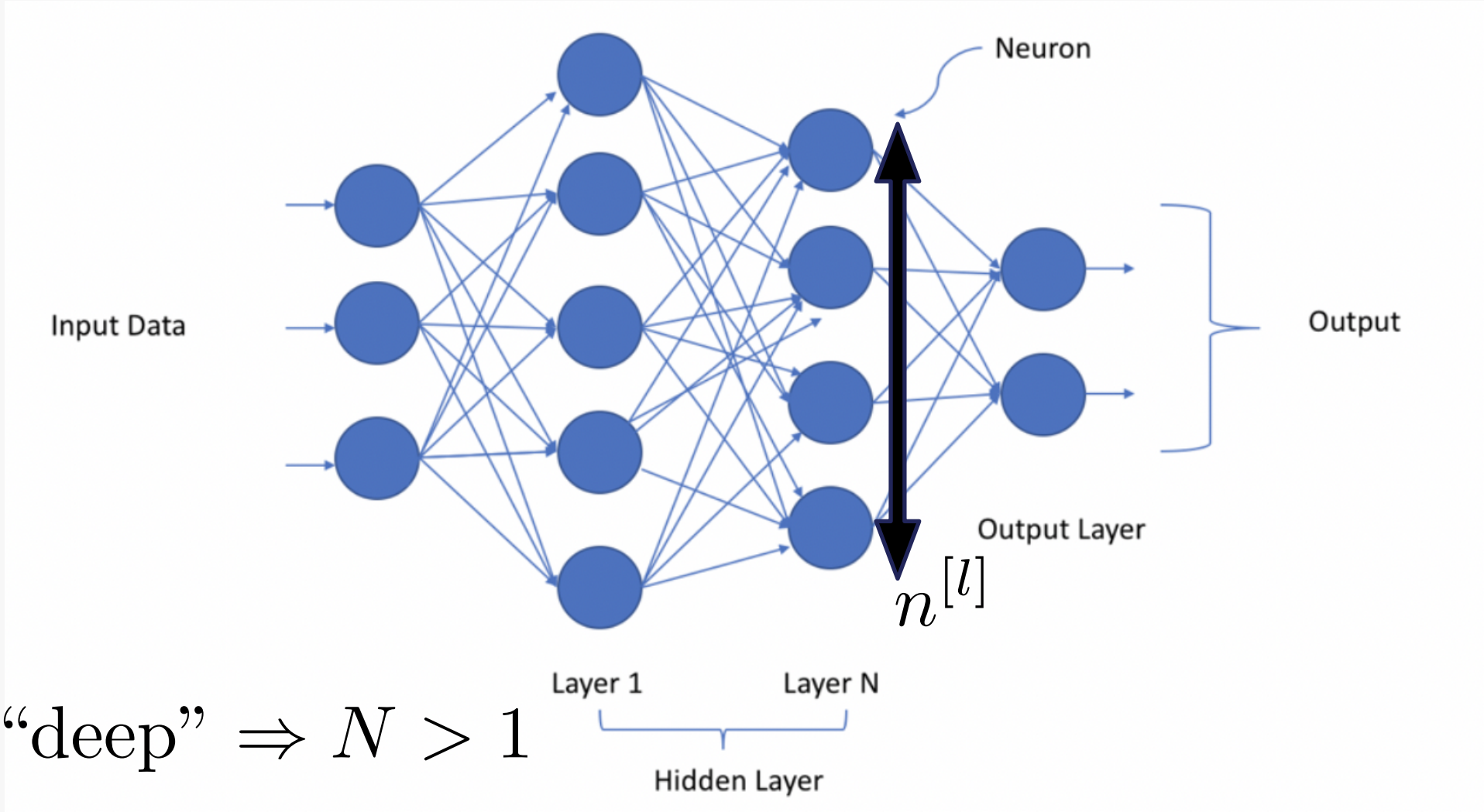
Deep neural network (DNN)



$$\text{“deep”} \Rightarrow N > 1$$

https://towardsdatascience.com/a-laymans-guide-to-deep-neural-networks-ddcea24847fb

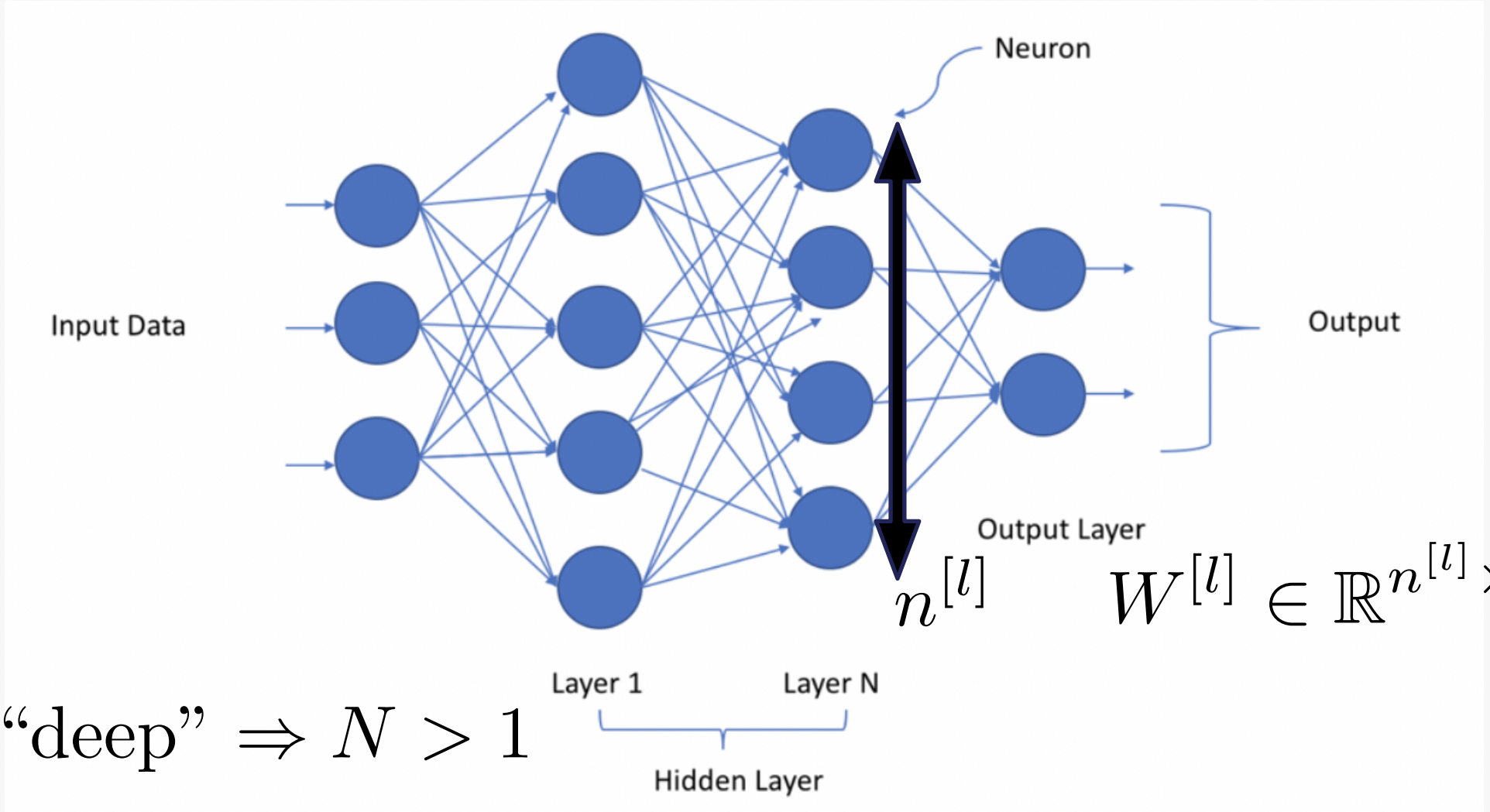
Deep neural network (DNN)



$$\text{“deep”} \Rightarrow N > 1$$

https://towardsdatascience.com/a-laymans-guide-to-deep-neural-networks-ddcea24847fb

Deep neural network (DNN)



$n^{[l]}$

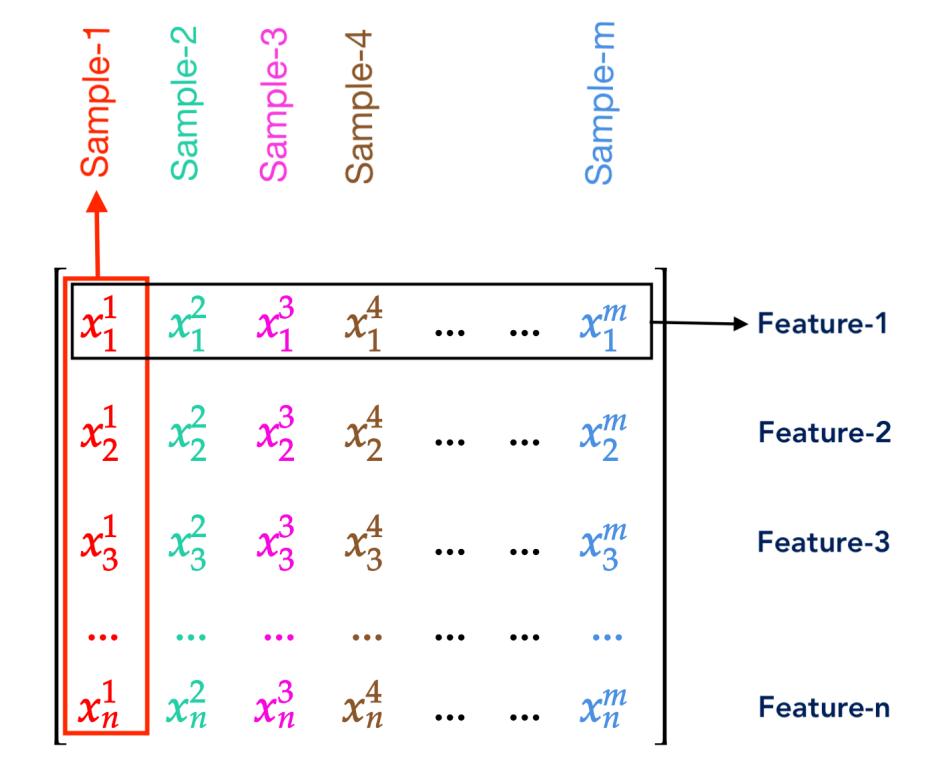
$W^{[l]} \in \mathbb{R}^{n^{[l]} \times n^{[l-1]}}$

$$\text{“deep”} \Rightarrow N > 1$$

https://towardsdatascience.com/a-laymans-guide-to-deep-neural-networks-ddcea24847fb

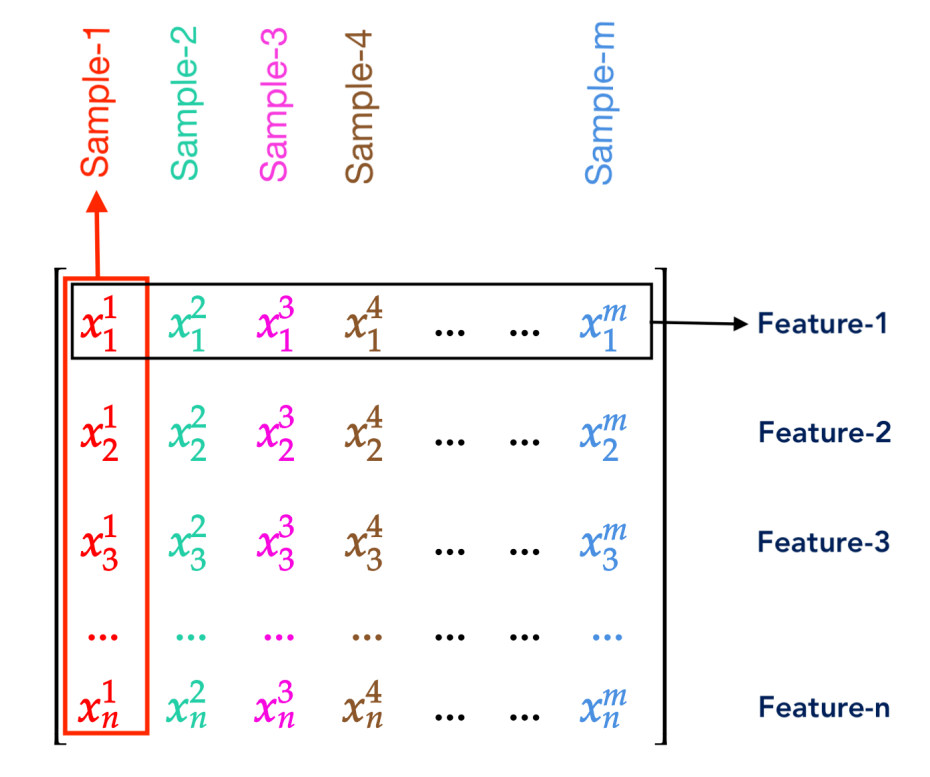
Vectorized notation: pass m examples at once

Start with horizontally-stacked input vectors $X \in \mathbb{R}^{n \times m}$



Vectorized notation: pass m examples at once

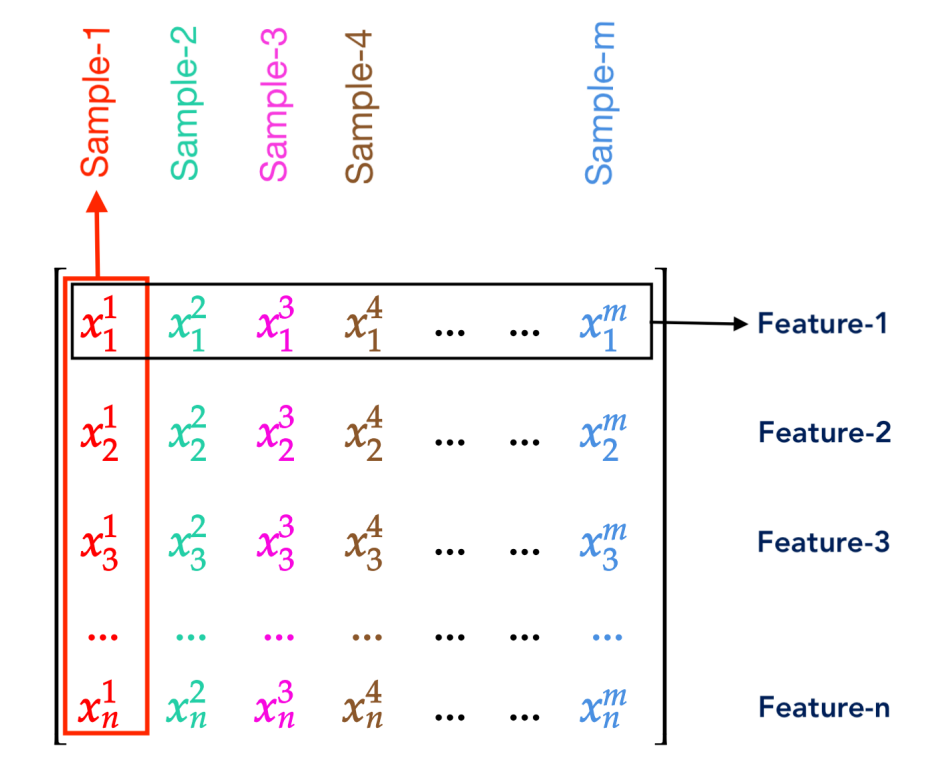
Start with horizontally-stacked input vectors $X \in \mathbb{R}^{n \times m}$



Vectorized forward propagation

Vectorized notation: pass m examples at once

Start with horizontally-stacked input vectors $X \in \mathbb{R}^{n \times m}$

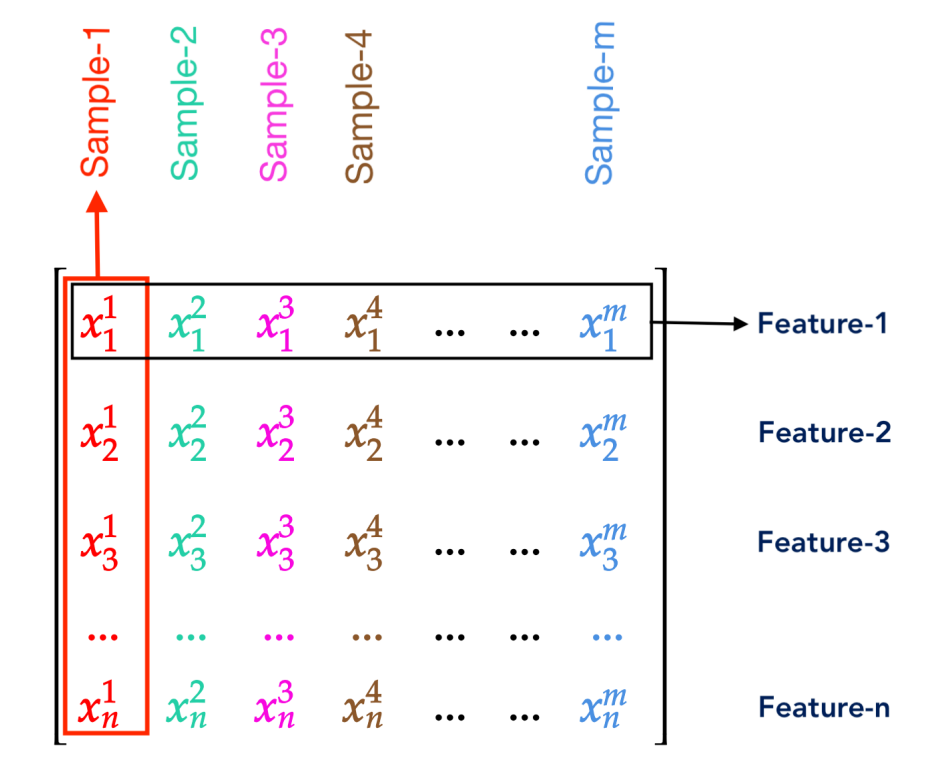


Vectorized forward propagation

$$Z^{[l]} = W^{[l]} A^{[l-1]} + B^{[l]}$$

Vectorized notation: pass m examples at once

Start with horizontally-stacked input vectors $X \in \mathbb{R}^{n \times m}$



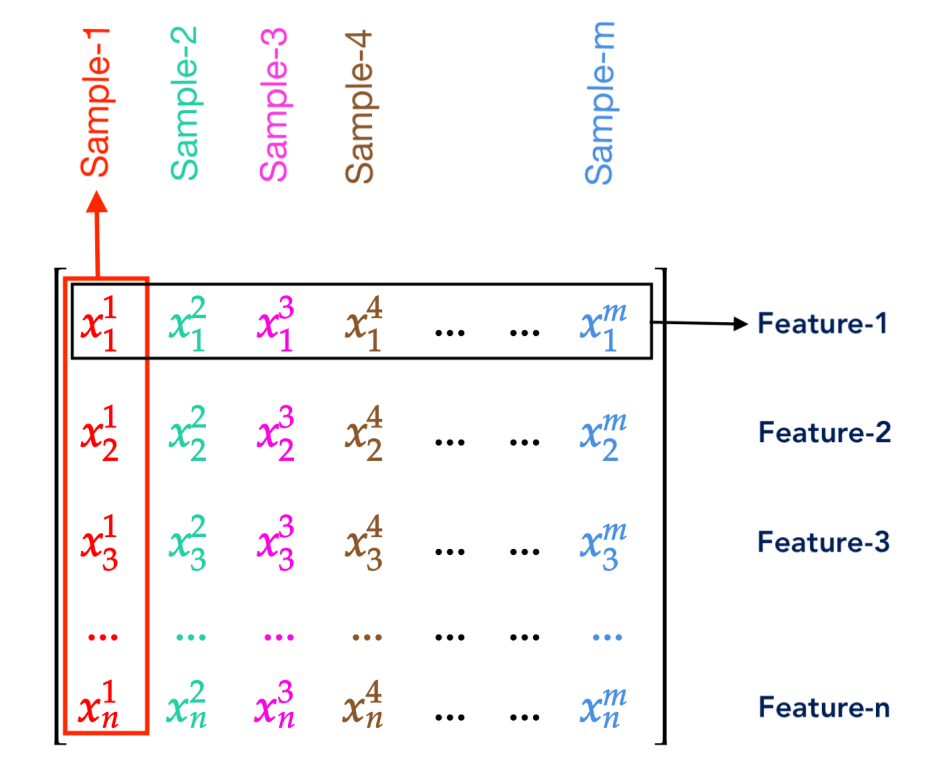
Vectorized forward propagation

$$Z^{[l]} = W^{[l]} A^{[l-1]} + B^{[l]}$$

$$A^{[l]} = \sigma\left(Z^{[l]}\right)$$

Vectorized notation: pass m examples at once

Start with horizontally-stacked input vectors $X \in \mathbb{R}^{n \times m}$



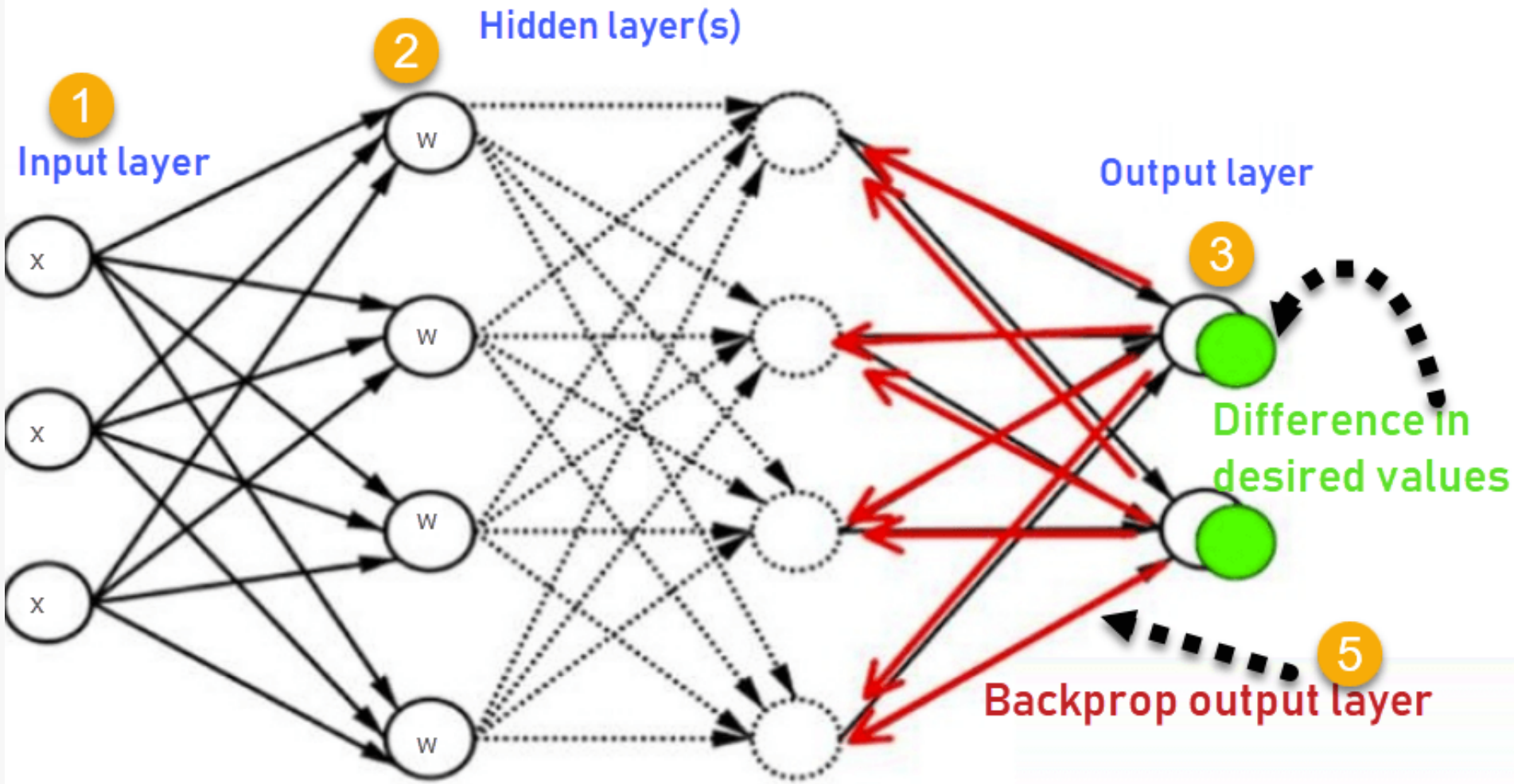
Vectorized forward propagation

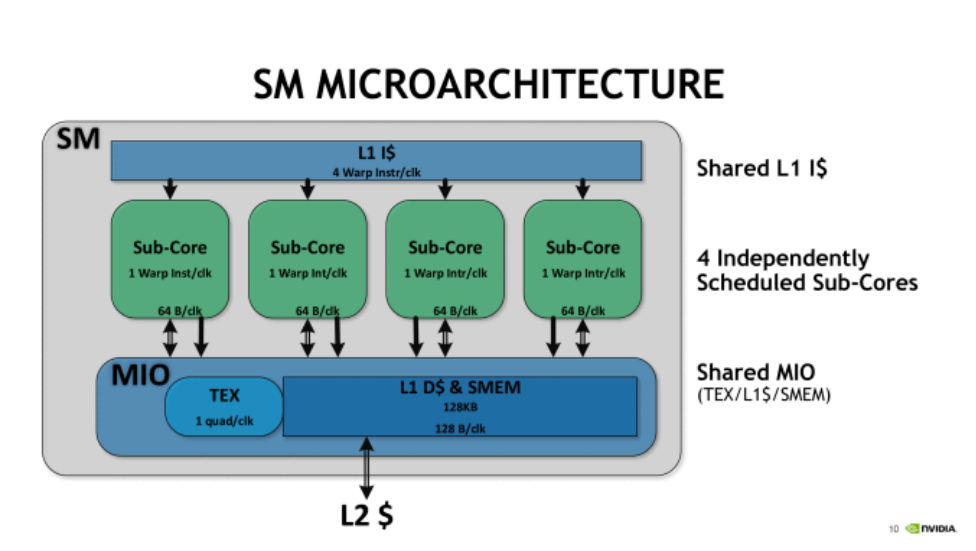
$$Z^{[l]} = W^{[l]} A^{[l-1]} + B^{[l]}$$

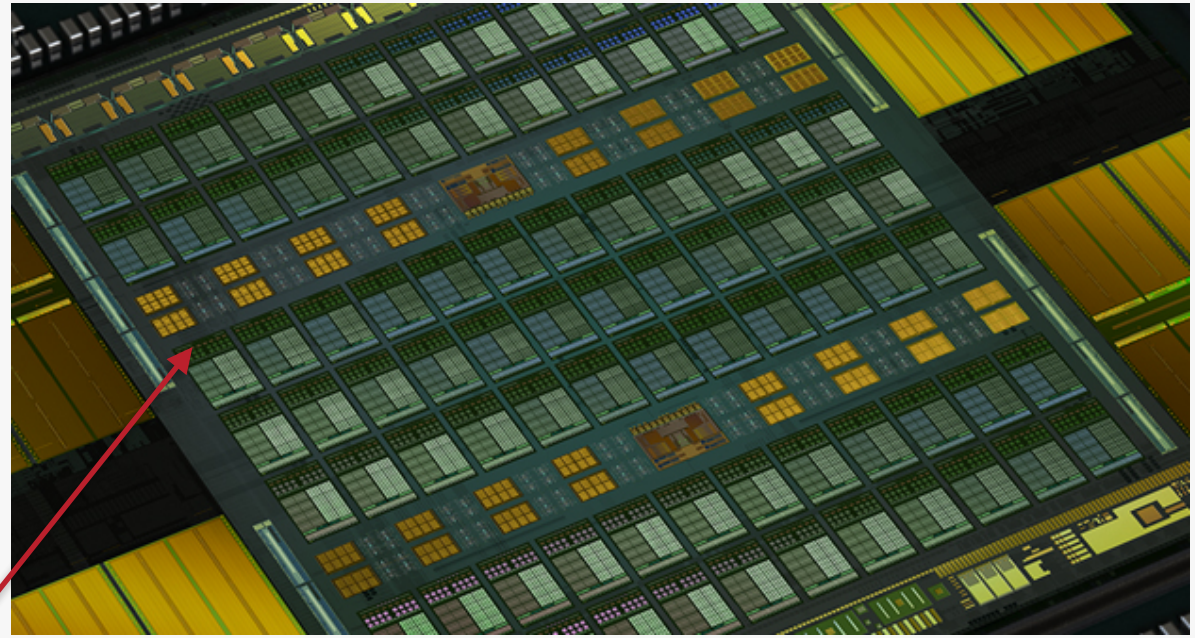
$$A^{[l]} = \sigma\left(Z^{[l]}\right)$$

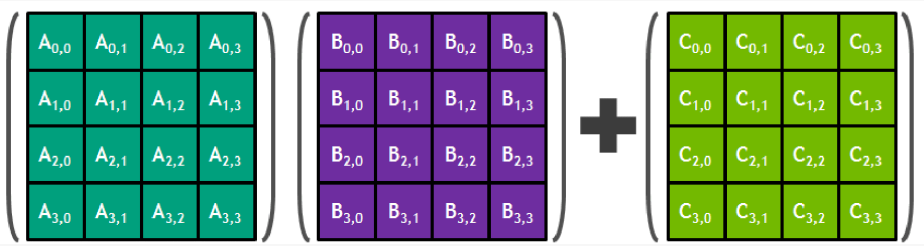
$$Z^{[l]}, A^{[l]} \in \mathbb{R}^{n^{[l]} \times m}$$

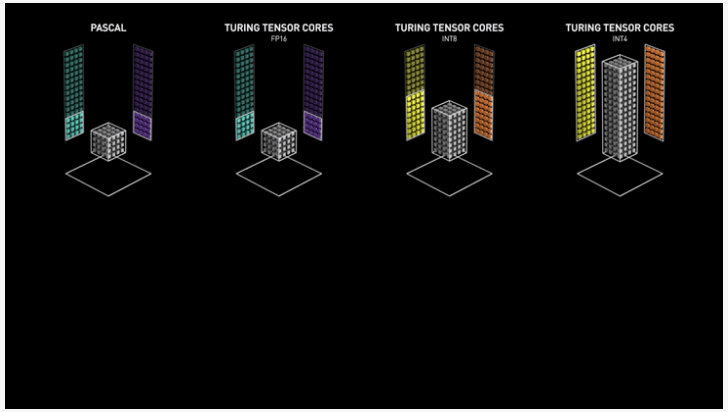
“Layer-by-layer” gradient descent through backpropagation

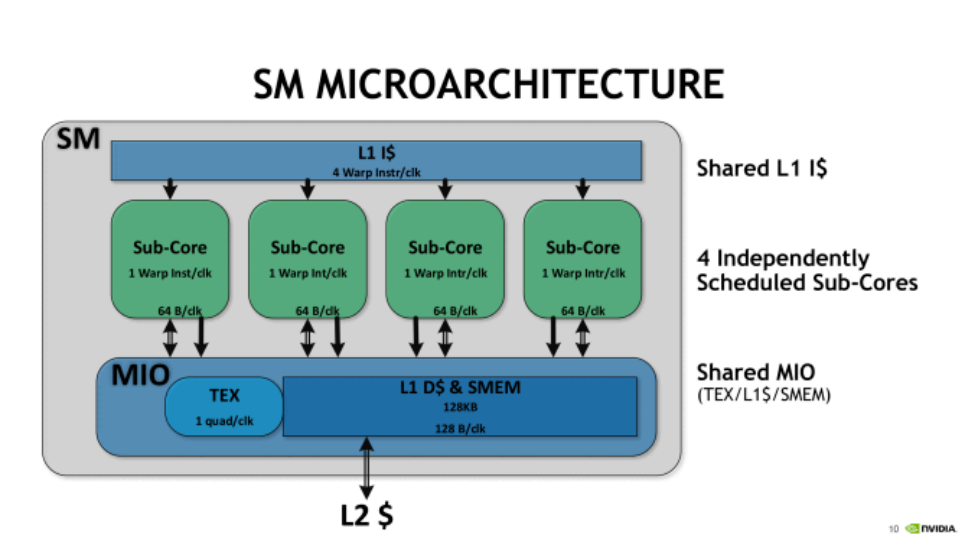


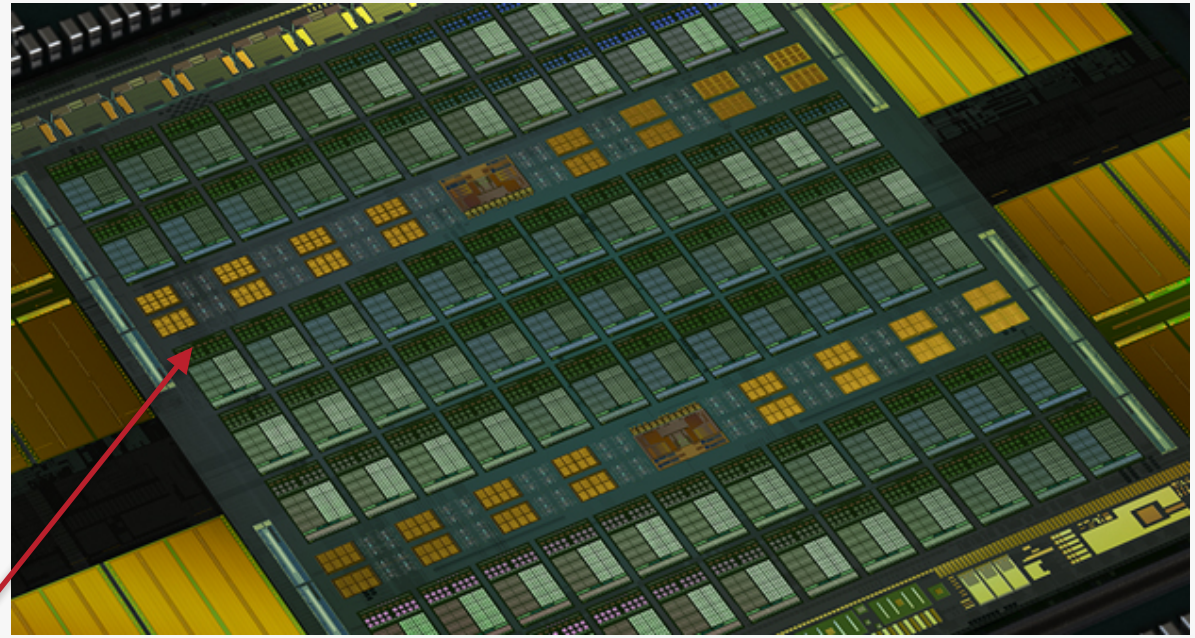

SM MICROARCHITECTURE
SM
L1 I$
4 Warp Instr/clk
Sub-Core
1 Warp Inst/clk
64 B/clk
Sub-Core
1 Warp Int/clk
64 B/clk
Sub-Core
1 Warp Intr/clk
64 B/clk
Sub-Core
1 Warp Intr/clk
64 B/clk
MIO
TEX
1 quad/clk
L1 D$ & SMEM
128KB
128 B/clk
L2 $
Shared L1 I$
4 Independently Scheduled Sub-Cores
Shared MIO
(TEX/L1$/SMEM)
10 NVIDIA

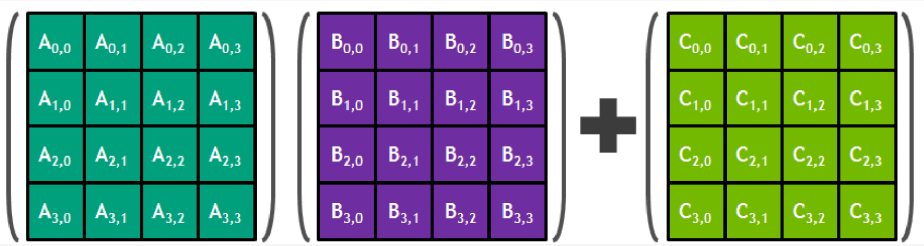


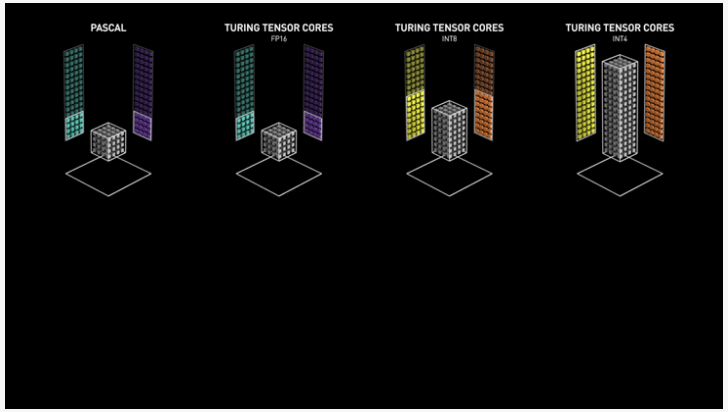

A0,0 A0,1 A0,2 A0,3
A1,0 A1,1 A1,2 A1,3
A2,0 A2,1 A2,2 A2,3
A3,0 A3,1 A3,2 A3,3
B0,0 B0,1 B0,2 B0,3
B1,0 B1,1 B1,2 B1,3
B2,0 B2,1 B2,2 B2,3
B3,0 B3,1 B3,2 B3,3
C0,0 C0,1 C0,2 C0,3
C1,0 C1,1 C1,2 C1,3
C2,0 C2,1 C2,2 C2,3
C3,0 C3,1 C3,2 C3,3


PASCAL
TURING TENSOR CORES
FP16
TURING TENSOR CORES
INT8
TURING TENSOR CORES
INT4

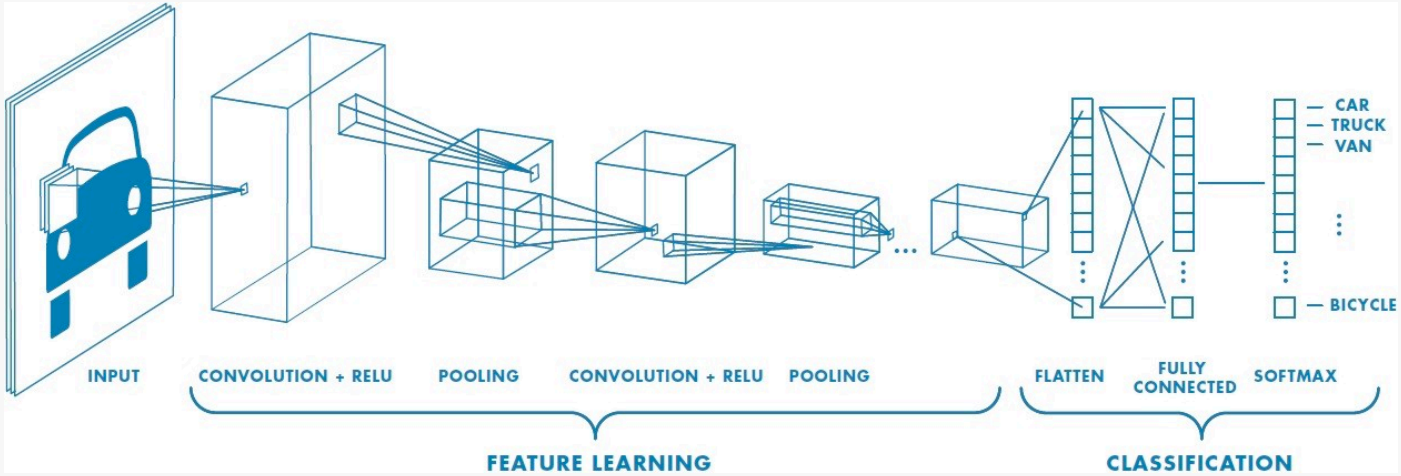

SM MICROARCHITECTURE
SM
L1 I$
4 Warp Instr/clk
Sub-Core
1 Warp Inst/clk
64 B/clk
Sub-Core
1 Warp Int/clk
64 B/clk
Sub-Core
1 Warp Intr/clk
64 B/clk
Sub-Core
1 Warp Intr/clk
64 B/clk
MIO
TEX
1 quad/clk
L1 D$ & SMEM
128KB
128 B/clk
L2 $
Shared L1 I$
4 Independently Scheduled Sub-Cores
Shared MIO
(TEX/L1$/SMEM)
10 NVIDIA

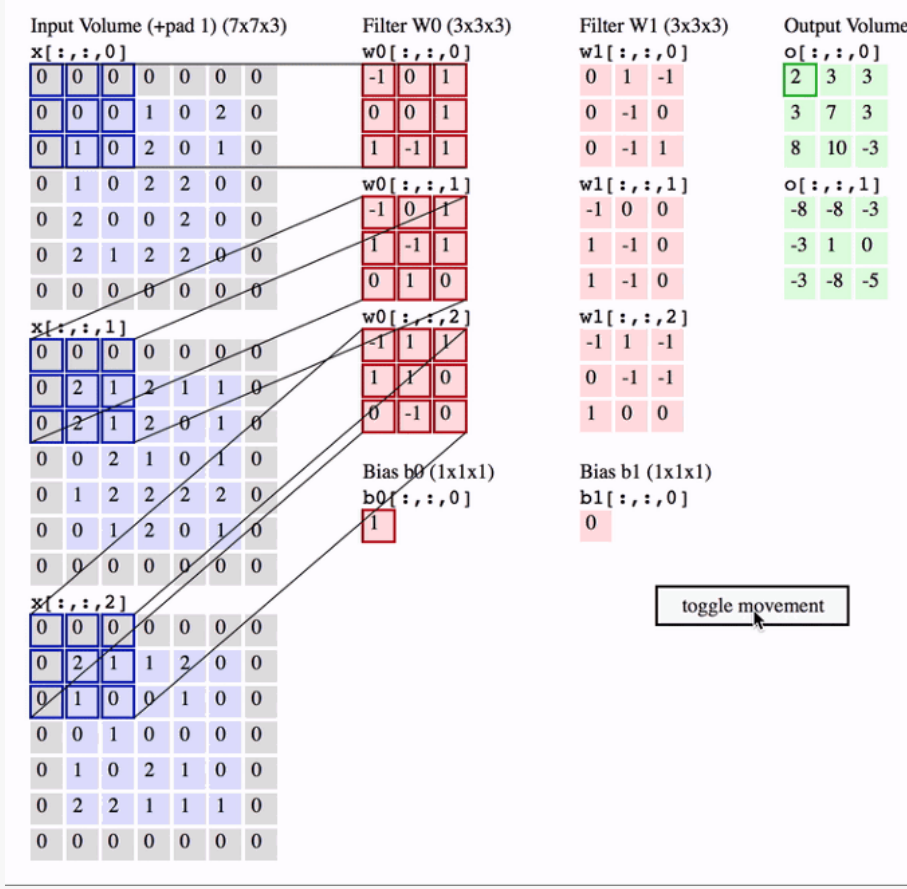


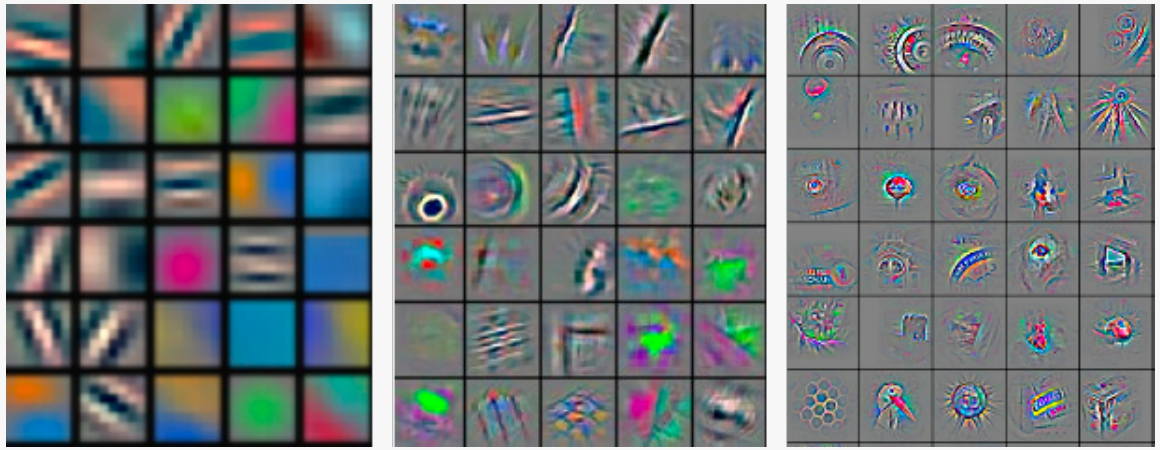

A0,0 A0,1 A0,2 A0,3
A1,0 A1,1 A1,2 A1,3
A2,0 A2,1 A2,2 A2,3
A3,0 A3,1 A3,2 A3,3
B0,0 B0,1 B0,2 B0,3
B1,0 B1,1 B1,2 B1,3
B2,0 B2,1 B2,2 B2,3
B3,0 B3,1 B3,2 B3,3
C0,0 C0,1 C0,2 C0,3
C1,0 C1,1 C1,2 C1,3
C2,0 C2,1 C2,2 C2,3
C3,0 C3,1 C3,2 C3,3


PASCAL
TURING TENSOR CORES
FP16
TURING TENSOR CORES
INT8
TURING TENSOR CORES
INT4

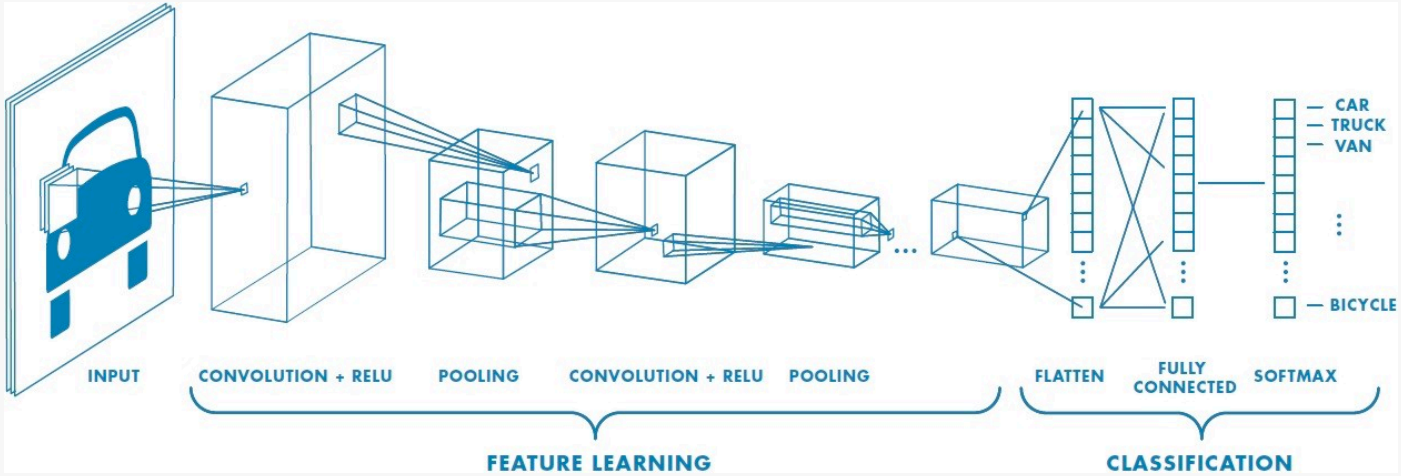
Convolutional layer

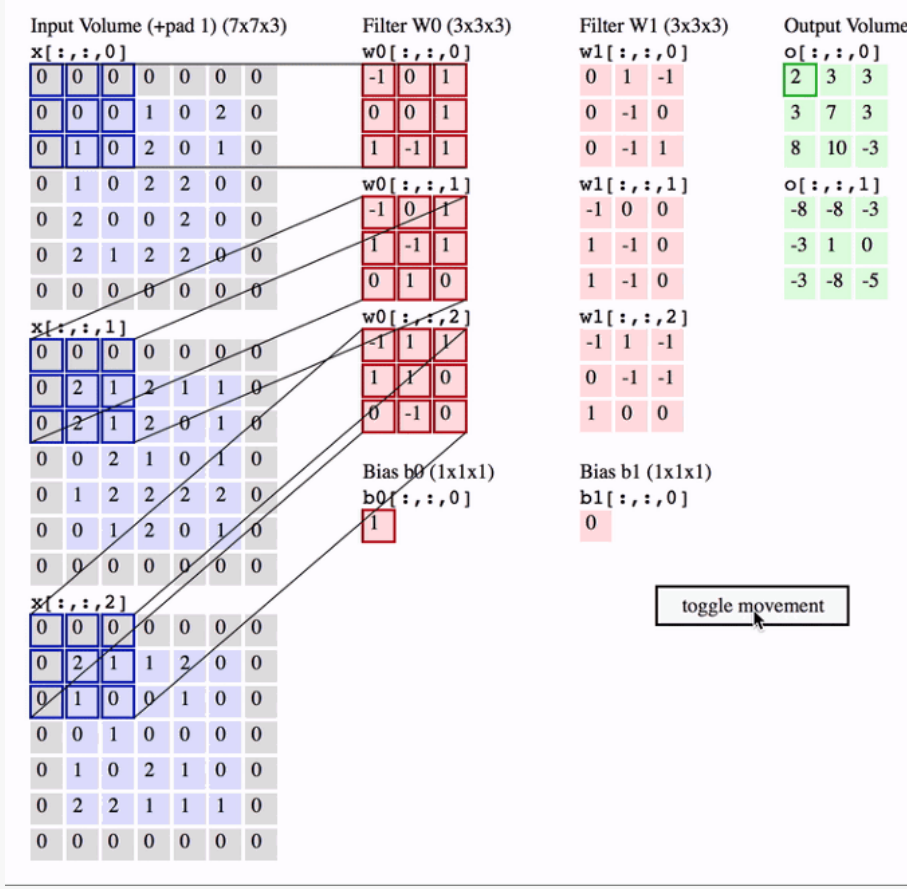


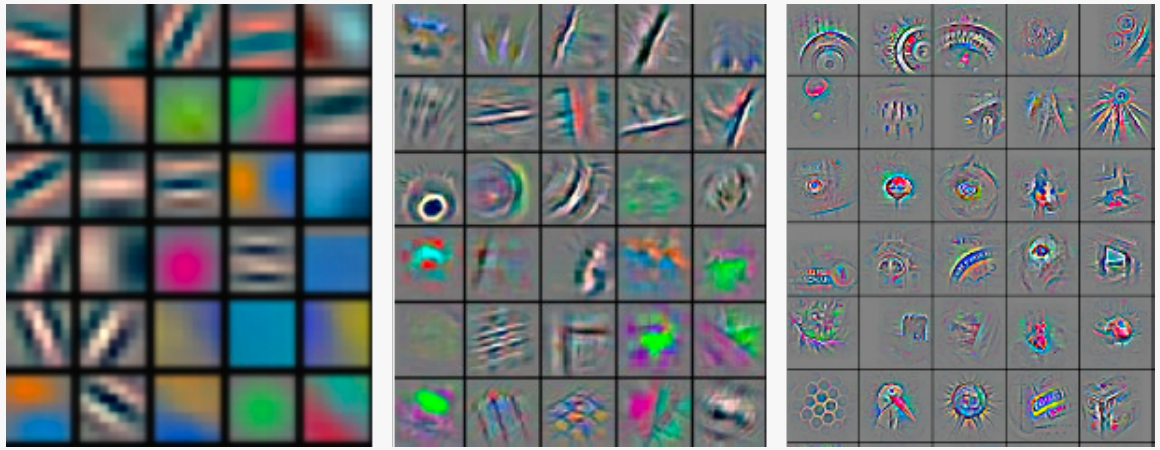




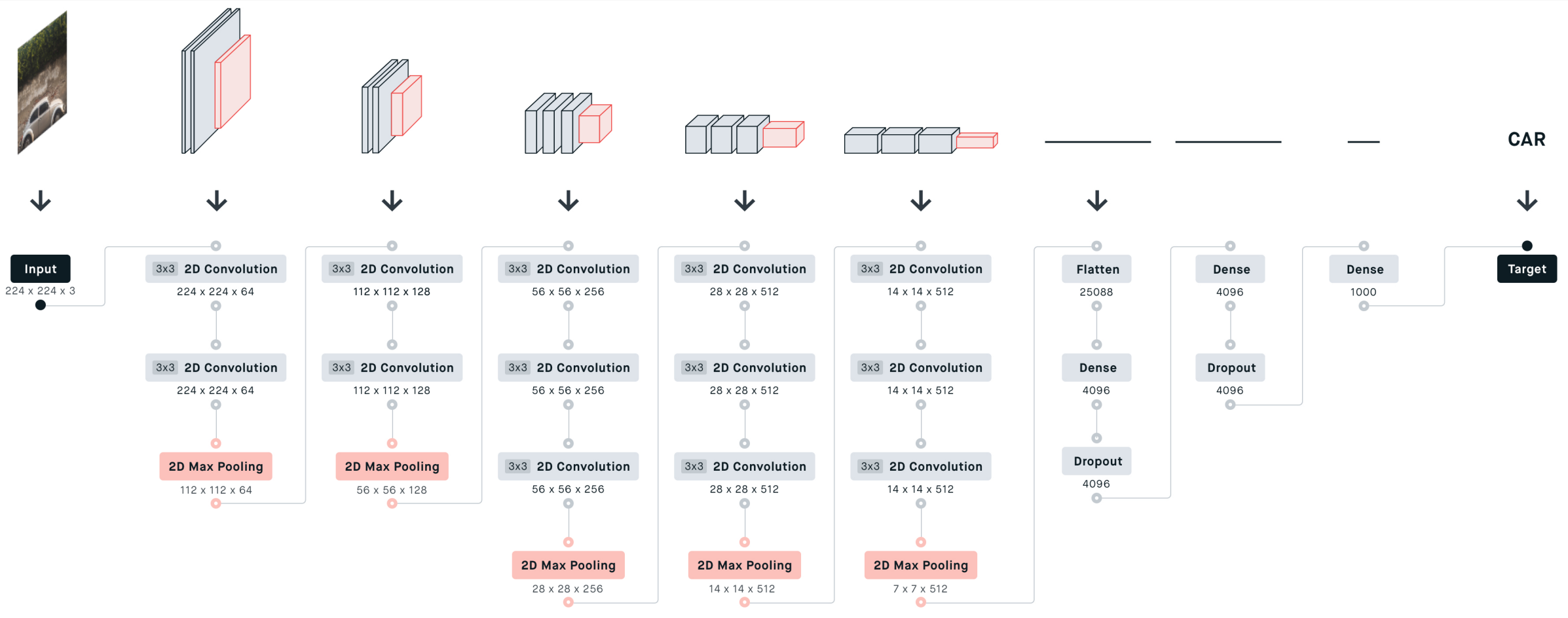
Convolutional layer



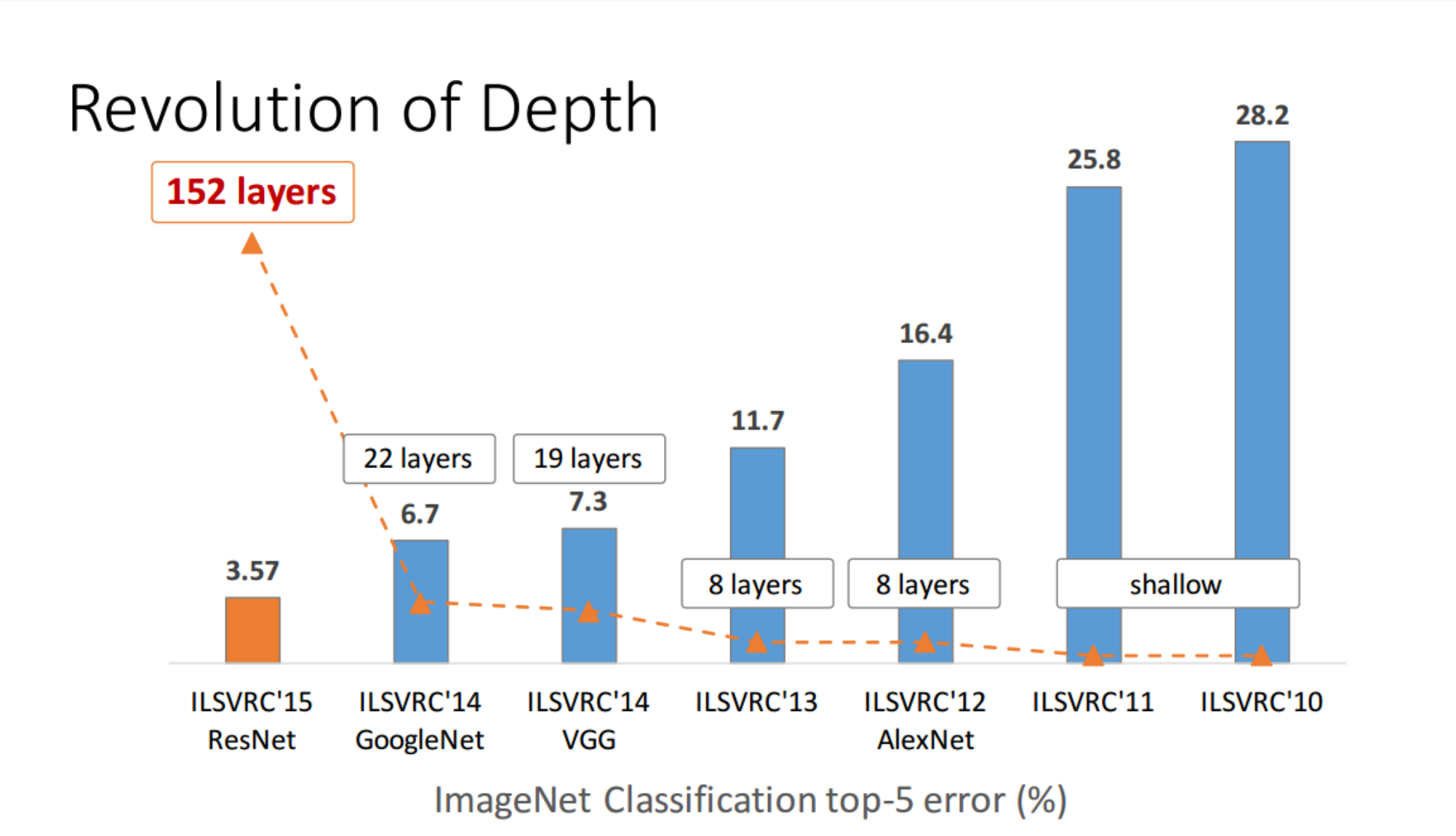




Convolutional neural network: VGG-16

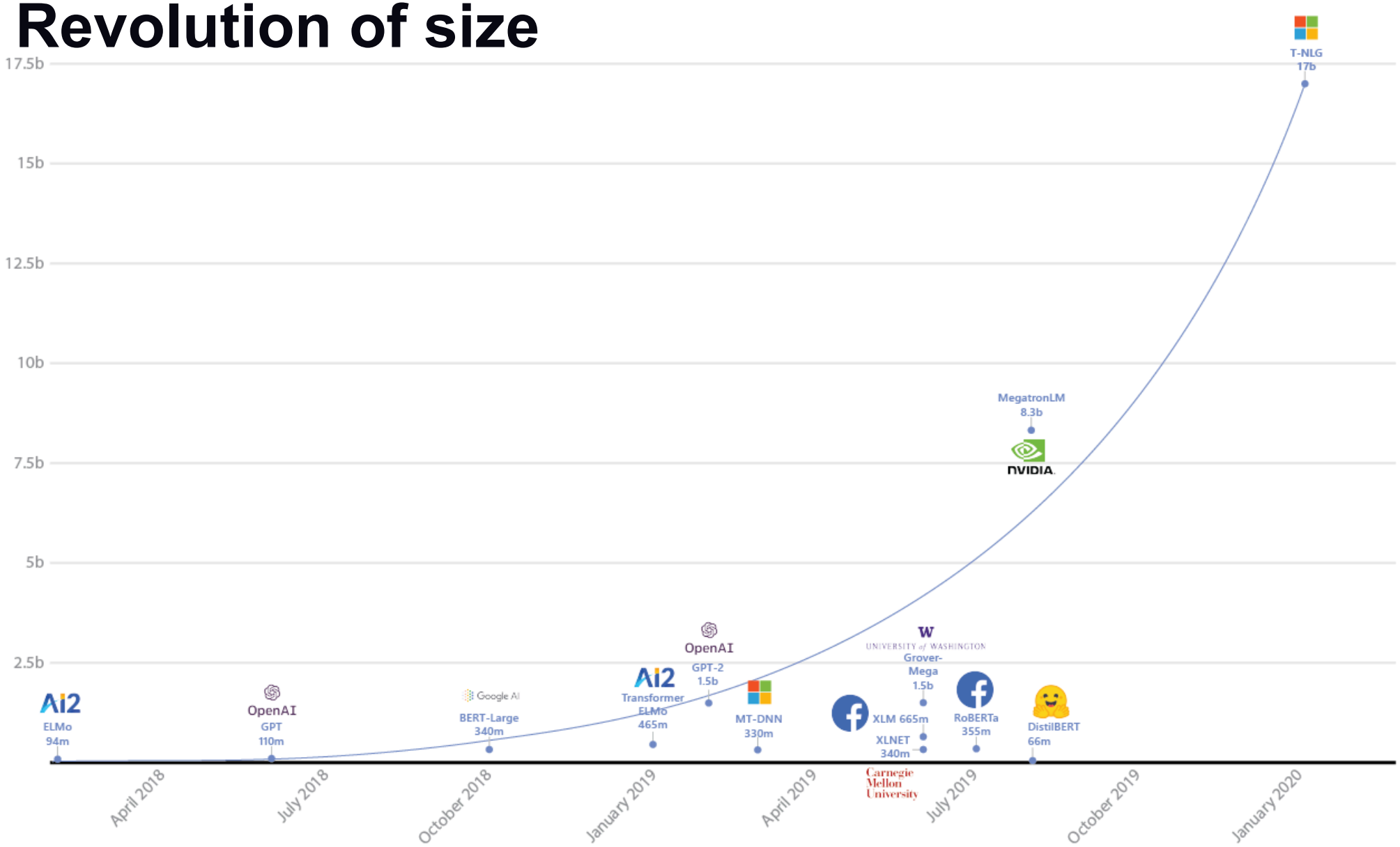


https://peltarion.com/



Kaiming He's ICML 2016 Tutorial

Revolution of size



https://www.microsoft.com/en-us/research/blog/turing-nlg-a-17-billion-parameter-language-model-by-microsoft/

Revolution of size

175b

* GPT-3

June 2020

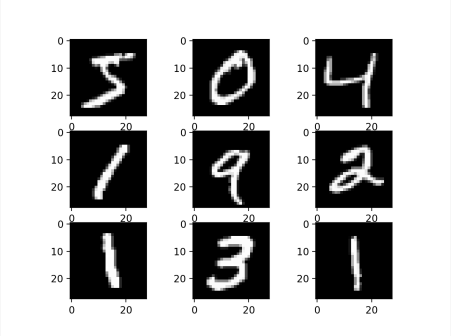
-
-
-

Software frameworks

TensorFlow

Keras

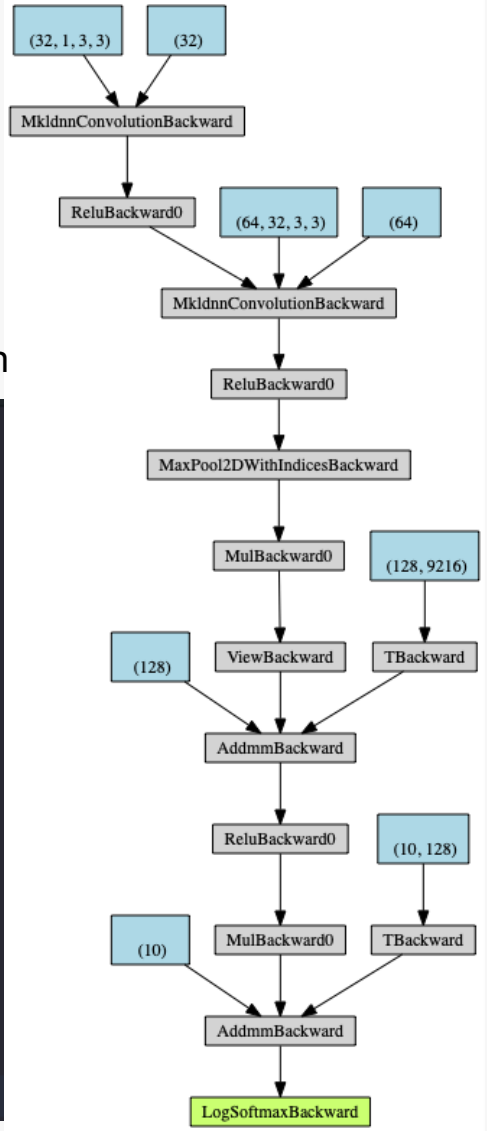
PyTorch



Example: PyTorch implementation of 4-layer CNN for MNIST classification

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
        self.conv2 = nn.Conv2d(10, 20, kernel_size=5)
        self.conv2_drop = nn.Dropout2d()
        self.fc1 = nn.Linear(320, 50)
        self.fc2 = nn.Linear(50, 10)

    def forward(self, x):
        x = F.relu(F.max_pool2d(self.conv1(x), 2))
        x = F.relu(F.max_pool2d(self.conv2_drop(self.conv2(x)), 2))
        x = x.view(-1, 320)
        x = F.relu(self.fc1(x))
        x = F.dropout(x, training=self.training)
        x = self.fc2(x)
        return F.log_softmax(x)
```



https://github.com/pytorch/examples/blob/master/mnist/main.py

Hands-on resources

Google Colaboratory

https://colab.research.google.com/

Tutorial Notebooks

https://github.com/argonne-lcf/ATPESC_MachineLearning/

Thank you!