A (Wild) Introduction to Deep Learning

Léonard Boussioux & Angelos Koulouras

15.S60, 2023

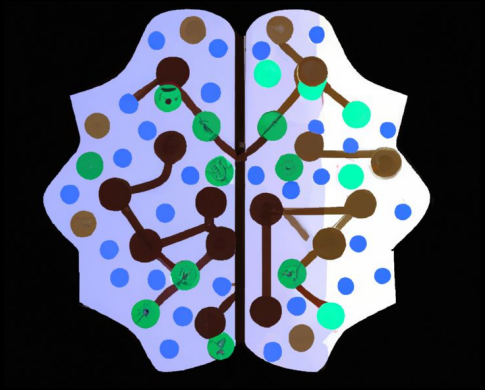


Image by DALLE

Plan

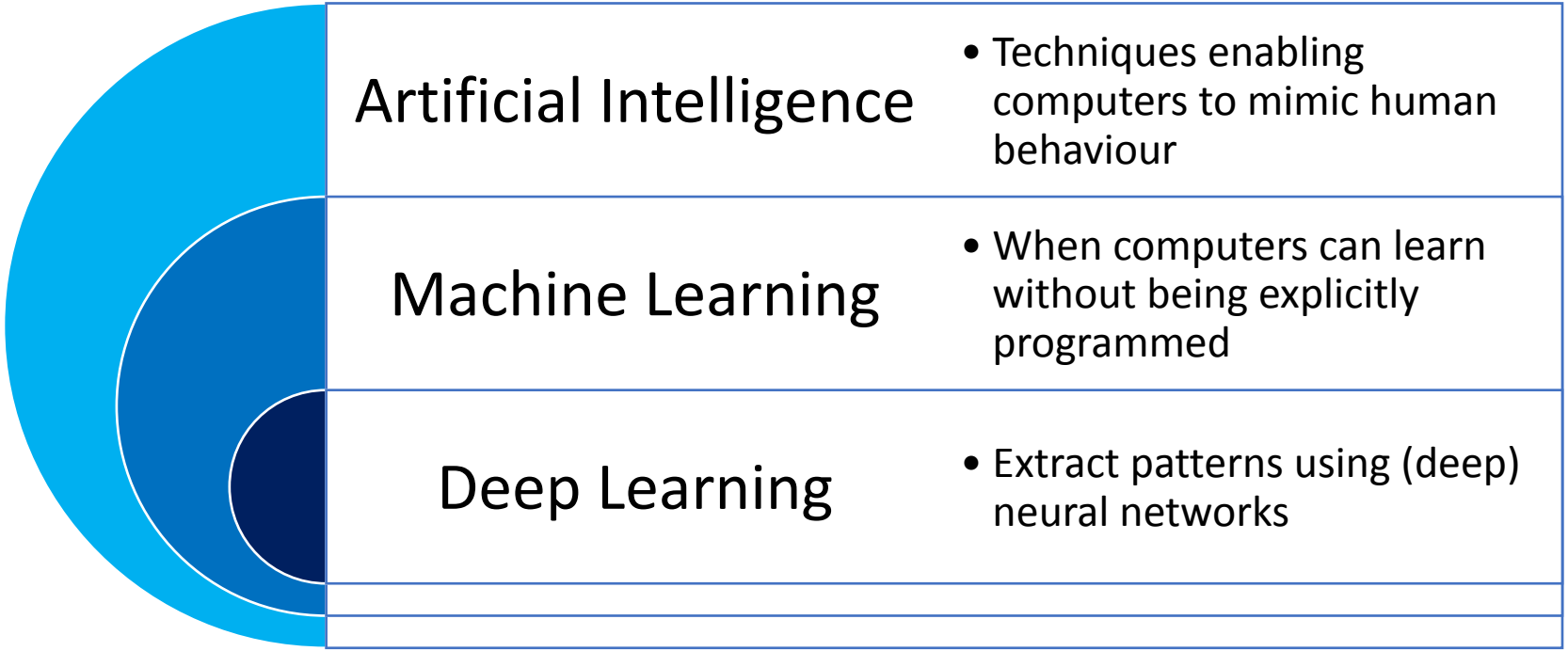
Concepts

- Perceptrons
- Gradient Descent & Training
- Convolutional Neural Networks

Code

- TensorFlow

What is Deep Learning?

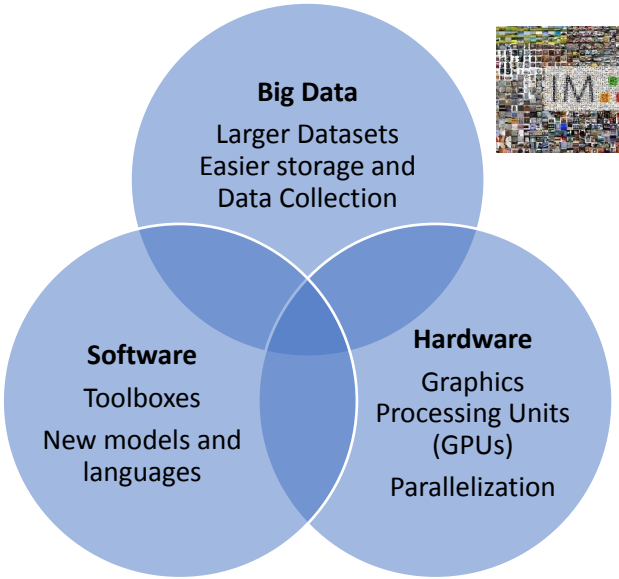


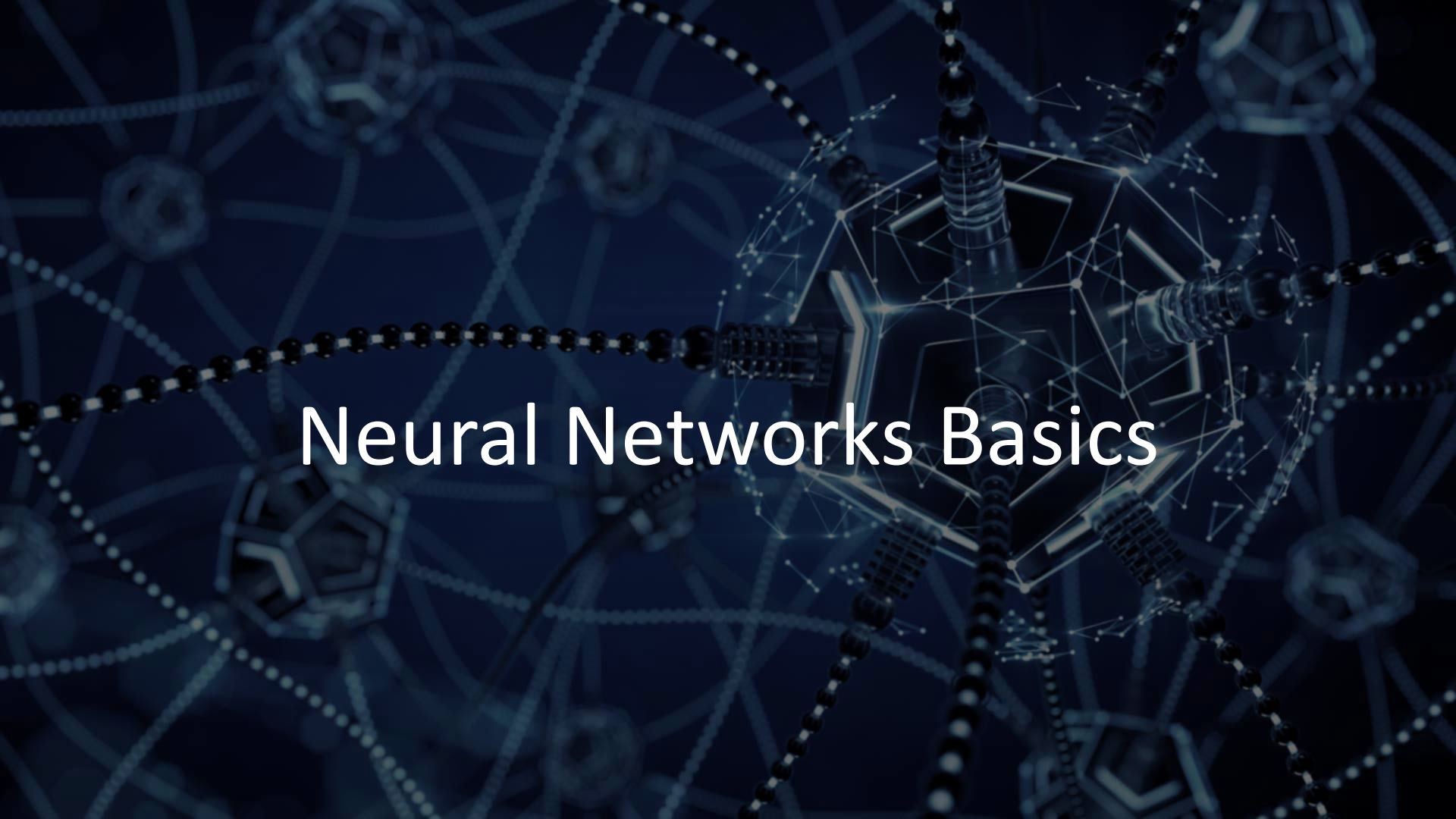
What is the history behind the current AI boom?

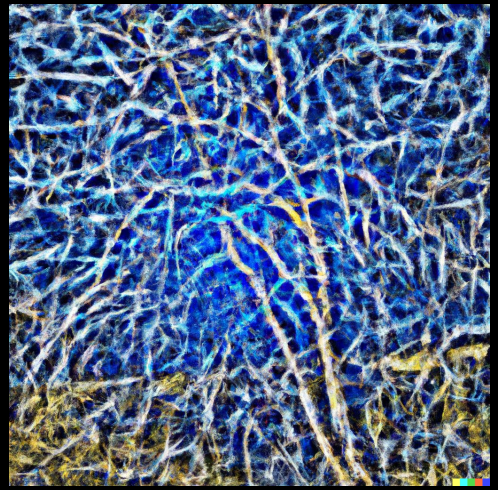
Learnable weights

Multi-Layer Perceptron

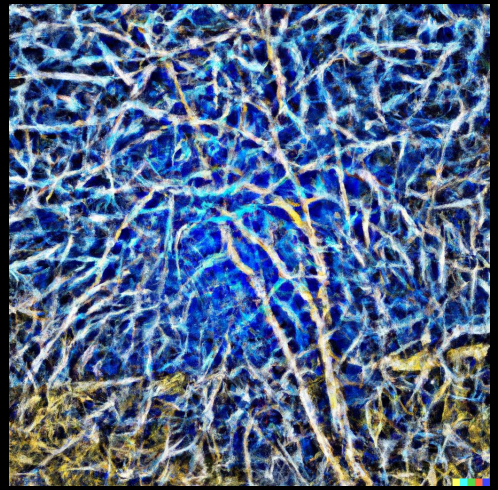
Digit Recognition




Neural Networks Basics



Images by DALLE



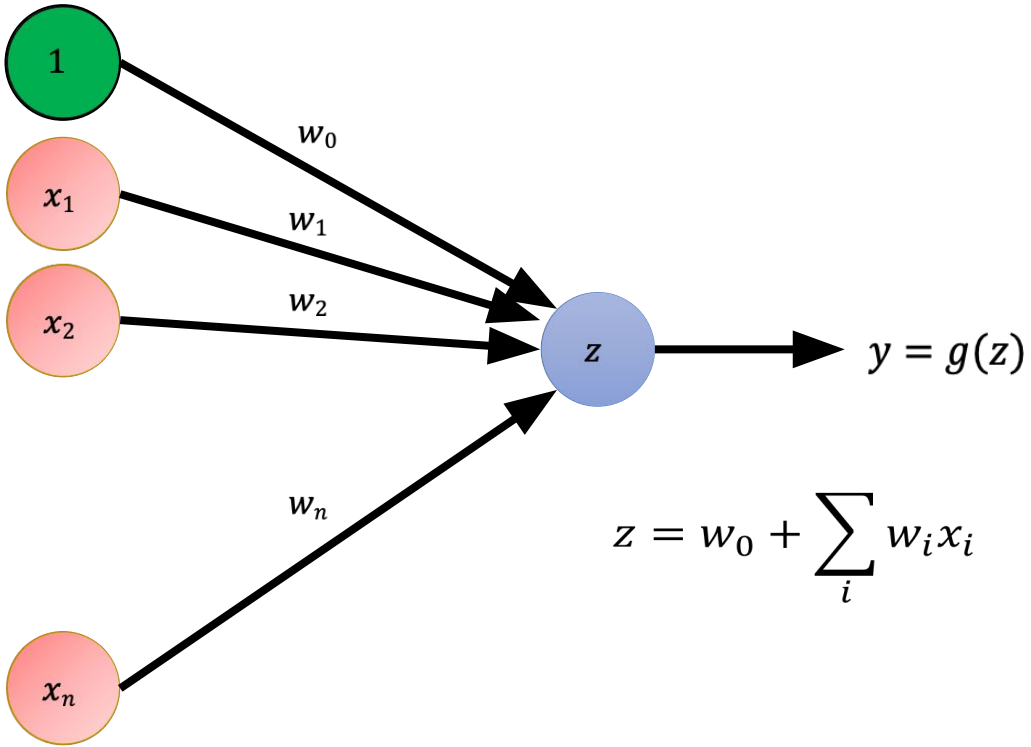
"A neural network"
impressionist style

"A neural network"
abstract pencil with
watercolor

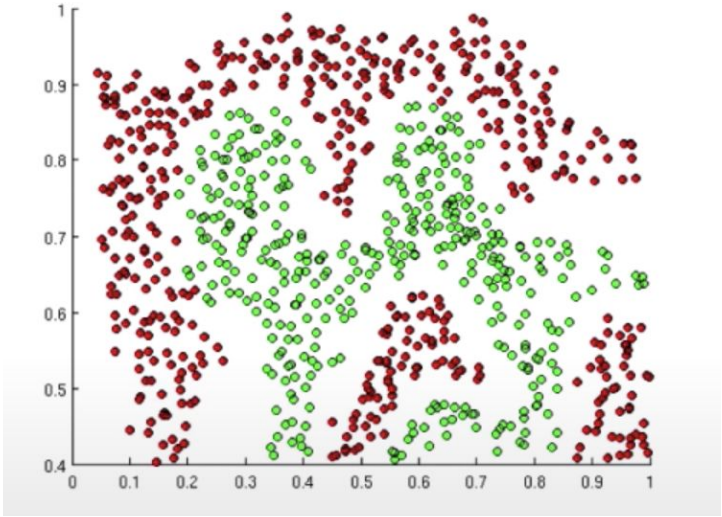
"A neural network"
by Matisse

Images by DALLE

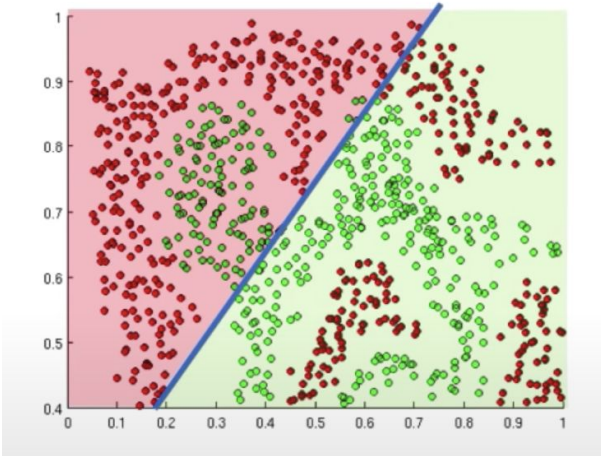
A simplified perceptron



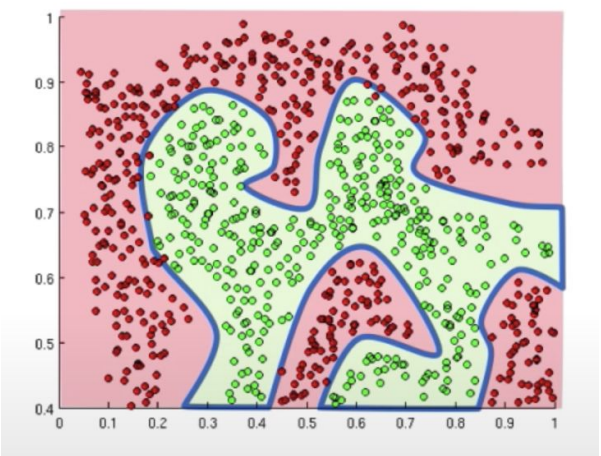
Classification Task



Activation functions

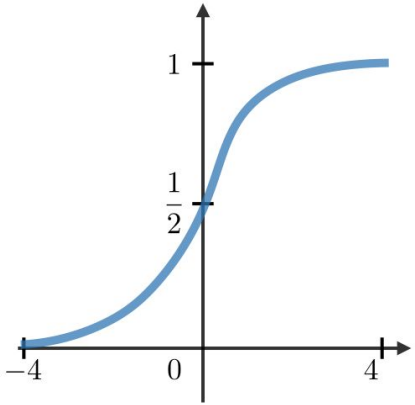
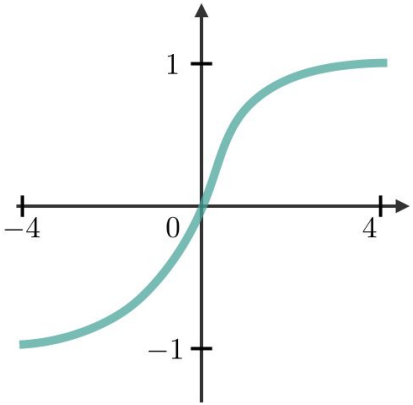
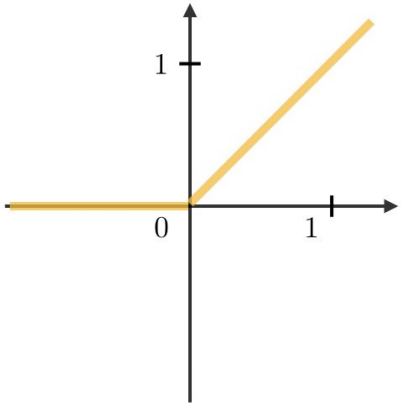


Linear activation functions -> linear decisions

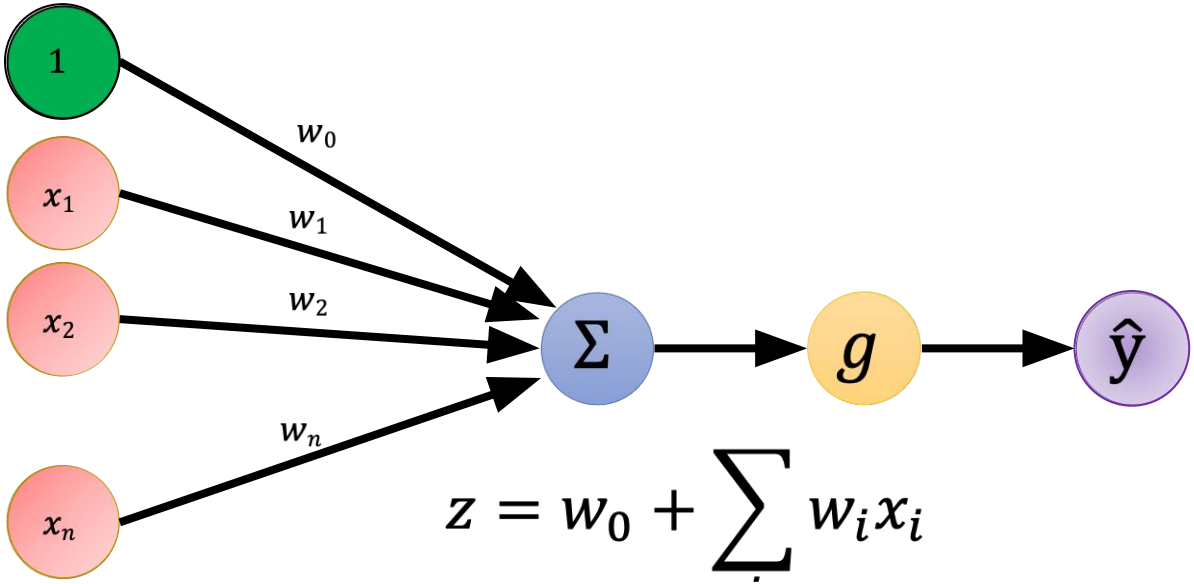


Non-linear activation -> approximate arbitrarily complex functions

Common Activation Functions

Sigmoid	Tanh	RELU
$g(z) = \frac{1}{1+e^{-z}}$	$g(z) = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$	$g(z) = \max(0, z)$
		

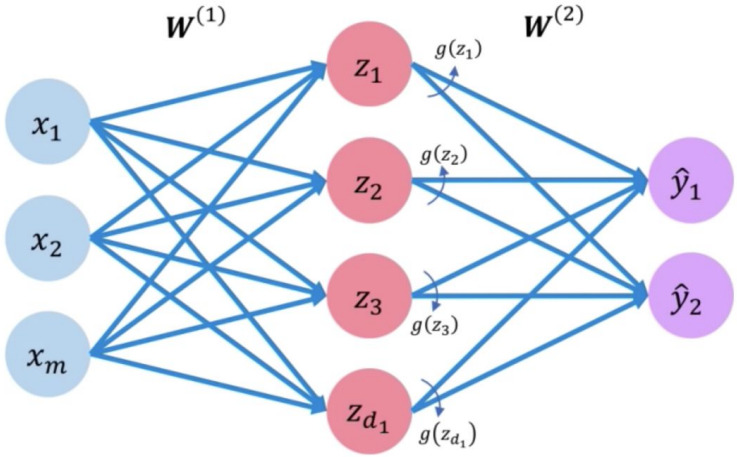
The perceptron: forward propagation



$$z = w_0 + \sum_i w_i x_i$$

$$\hat{y} = g(\sum_i w_i x_i)$$

Single Layer Neural Network

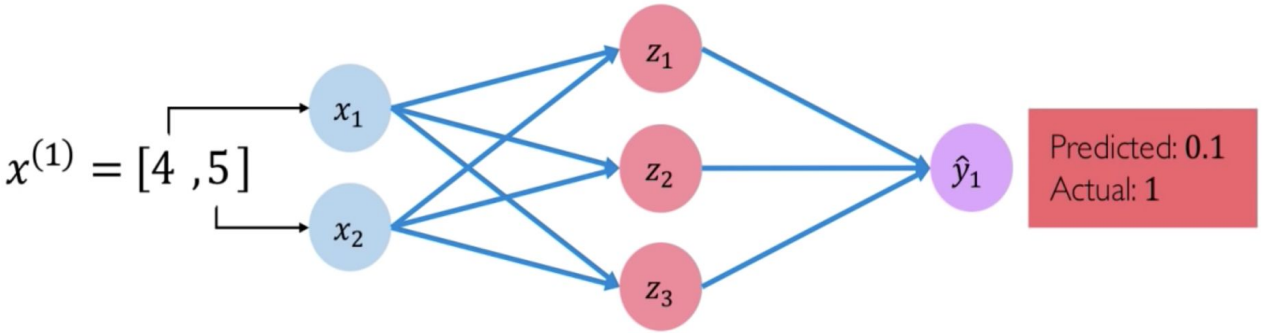


$$z_i = w_{0,i}^{(1)} + \sum_{j=1}^{m} x_j \, w_{j,i}^{(1)} \qquad \hat{y}_i = g\left(w_{0,i}^{(2)} + \sum_{j=1}^{d_1} z_j \, w_{j,i}^{(2)}\right)$$

Picture source: http://introtodeeplearning.com/

Loss function

The **loss** of our network measures the cost incurred from incorrect predictions.



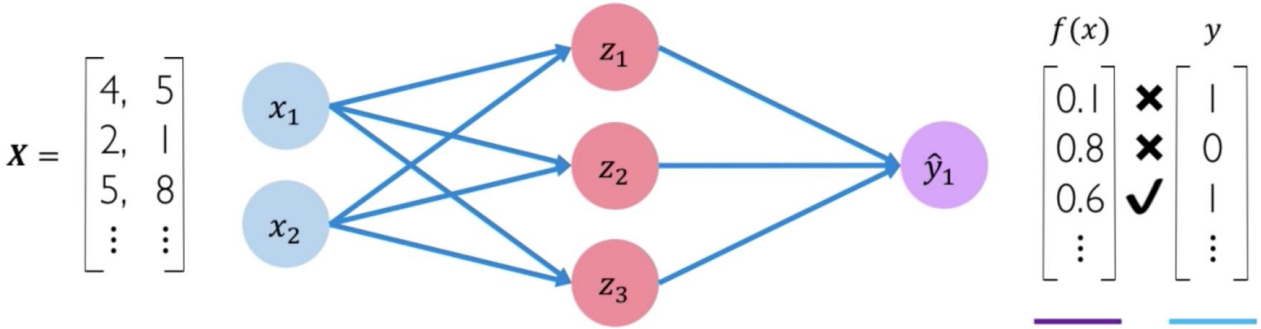
$$\mathcal{L}\left(f\left(x^{(i)}; \boldsymbol{W}\right), y^{(i)}\right)$$

Predicted Actual

Picture source: http://introtodeeplearning.com/

Empirical Loss

The **empirical loss** measures the total loss over our entire dataset.



Also known as:
- Objective function
- Cost function
- Empirical Risk

$$J(W) = \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}\left(f\left(x^{(i)}; W\right), y^{(i)}\right)$$

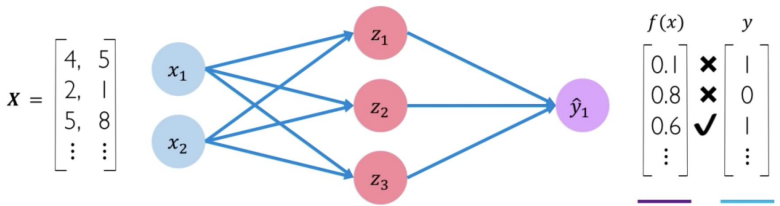
Predicted Actual

Picture source: http://introtodeeplearning.com/

Different loss functions

Binary Cross Entropy Loss

Cross entropy loss *can be used with models that output a probability between 0 and 1*

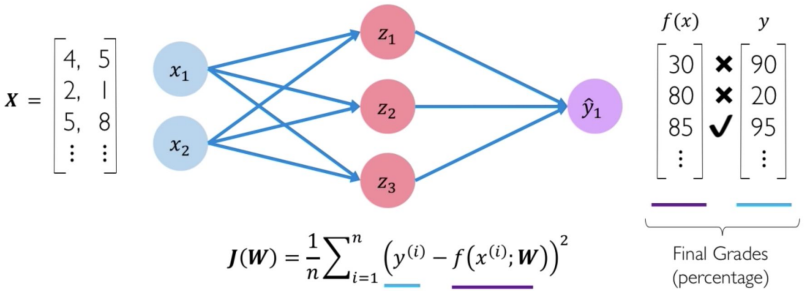


$$J(W) = \frac{1}{n}\sum_{i=1}^{n} \underbrace{y^{(i)}}_{\text{Actual}} \log \underbrace{\left(f(x^{(i)}; W)\right)}_{\text{Predicted}} + (1 - \underbrace{y^{(i)}}_{\text{Actual}}) \log \left(1 - \underbrace{f(x^{(i)}; W)}_{\text{Predicted}}\right)$$

```
loss = tf.reduce_mean( tf.nn.softmax_cross_entropy_with_logits(y, predicted) )
```

Mean Squared Error Loss

Mean squared error loss *can be used with regression models that output continuous real numbers*



$$J(W) = \frac{1}{n}\sum_{i=1}^{n} \left(\underbrace{y^{(i)}}_{\text{Actual}} - \underbrace{f(x^{(i)}; W)}_{\text{Predicted}}\right)^2$$

```
loss = tf.reduce_mean( tf.square(tf.subtract(y, predicted)) )
```

Picture source: http://introtodeeplearning.com/

How to optimize the loss function?

We need to find the network weights leading to the lowest loss.

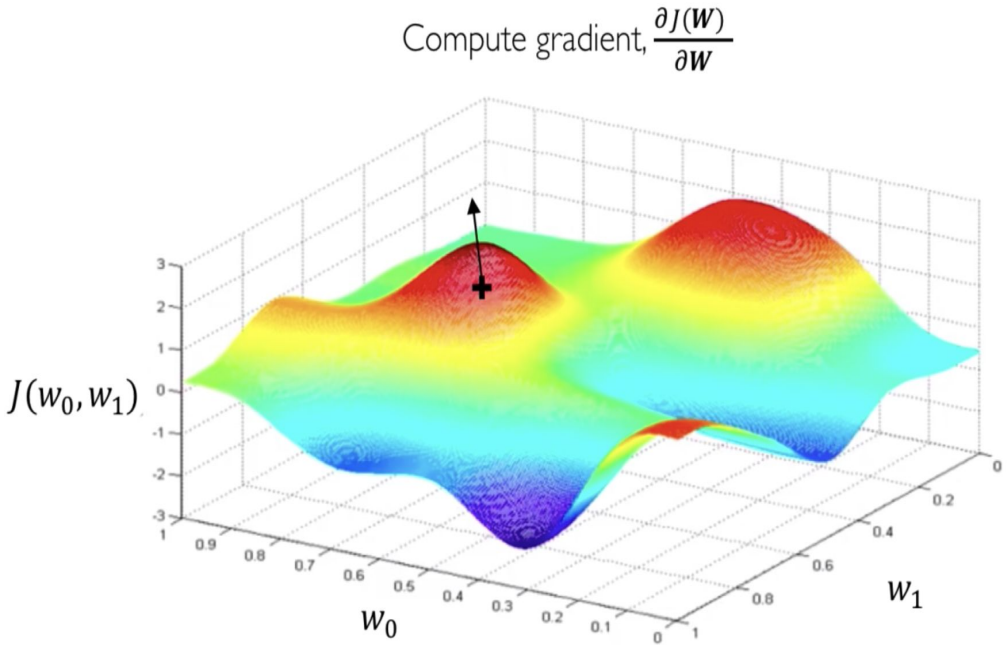
$$\boldsymbol{W}^* = \underset{\boldsymbol{W}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}\left(f\left(x^{(i)}; \boldsymbol{W}\right), y^{(i)}\right)$$

$$\boldsymbol{W}^* = \underset{\boldsymbol{W}}{\operatorname{argmin}} J(\boldsymbol{W})$$

Remember:

$$\boldsymbol{W} = \left\{\boldsymbol{W}^{(0)}, \boldsymbol{W}^{(1)}, \dots\right\}$$

But it looks like this…



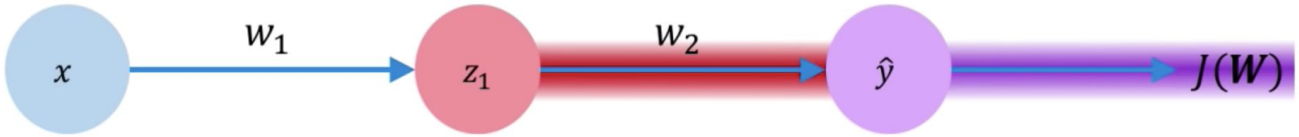
Picture source: http://introtodeeplearning.com/

Fortunately, there is Gradient Descent!

Algorithm

1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient, $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
4. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Return weights

And backpropagation with chain rule



$$\frac{\partial J(\boldsymbol{W})}{\partial w_2} = \frac{\partial J(\boldsymbol{W})}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial w_2}$$

Picture source: http://introtodeeplearning.com/

But how do you choose the learning rate?

Remember:

Optimization through gradient descent

$$\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$$

How can we set the learning rate?

Ideas?

Adaptive Learning Rate

Learning rate is not fixed anymore, instead:

You can adapt it depending on:
- How large gradients are
- How fast the learning is happening (momentum)
- …

In Deep Learning, Adam is an extremely popular method and does all this automatically. But you still need to choose a base value (typically between 10^{-3}/5.10^{-5} but depends on the task and architecture.)

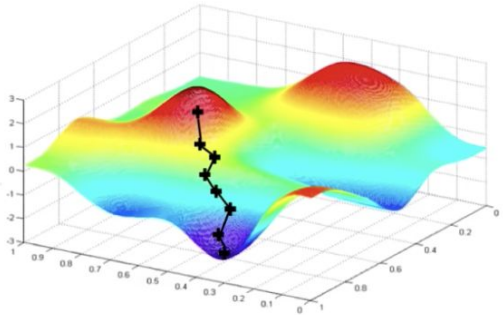
Computing gradients is expensive!

Gradient Descent

Algorithm

1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient, $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
4. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Return weights

Can be very **computationally intensive** to compute!



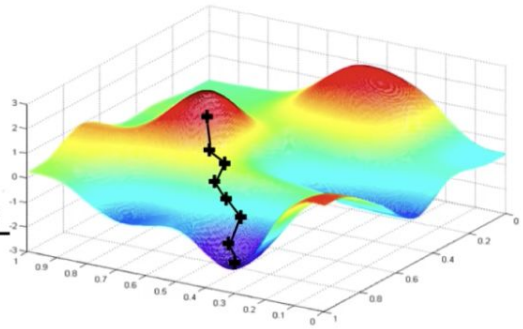
Picture source: http://introtodeeplearning.com/

Computing gradients is expensive!

Stochastic Gradient Descent

Algorithm

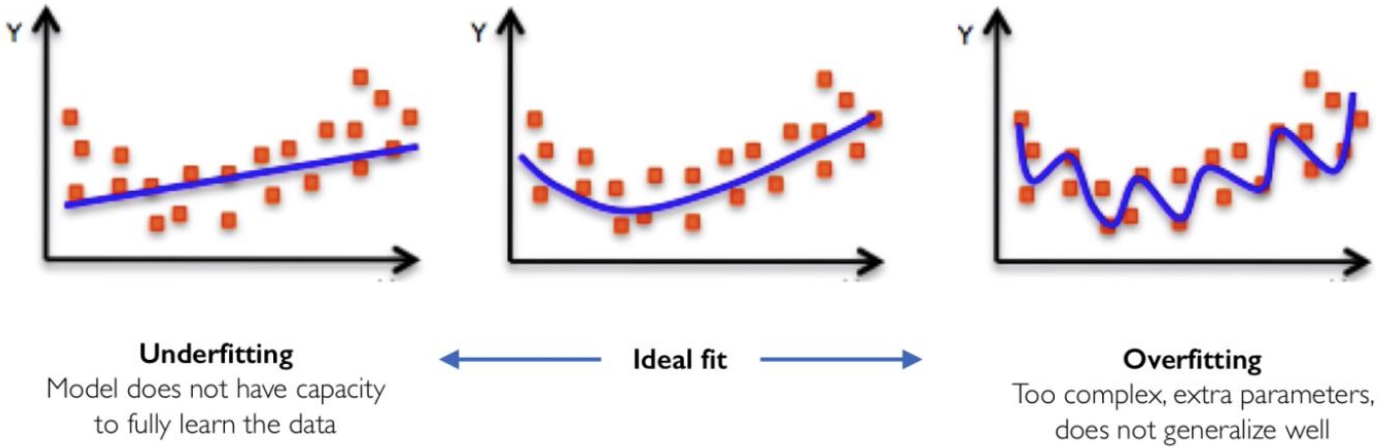
1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Pick batch of B data points
4. Compute gradient, $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}} = \frac{1}{B}\sum_{k=1}^{B} \frac{\partial J_k(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
6. Return weights



We use **mini-batches** while training allowing for a smoother convergence and larger learning rates.

Picture source: http://introtodeeplearning.com/

Another classic ML problem: overfitting



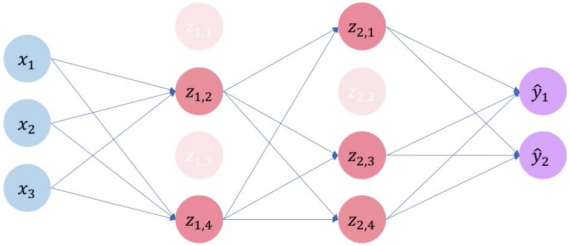
Picture source: http://introtodeeplearning.com/

Let’s overcome this! Ideas?

Regularization 1: Dropout

- During training, randomly set some activations to 0
  - Typically ‘drop’ 50% of activations in layer
  - Forces network to not rely on any 1 node

```
tf.keras.layers.Dropout(p=0.5)
```



Regularization 2: Early Stopping

- Stop training before we have a chance to overfit

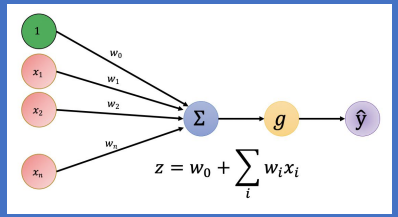
Regularization 3: L1 or L2 penalty on the weights

Picture source: http://introtodeeplearning.com/

Review

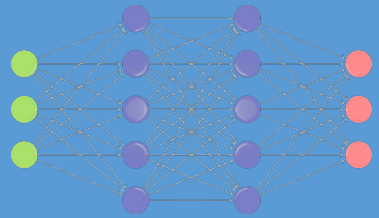
The Perceptron

- Structure
- Nonlinear activation functions



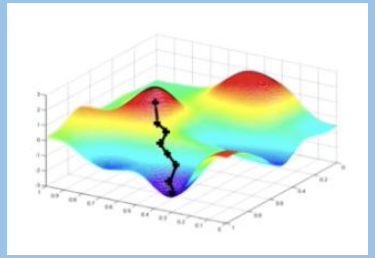
Neural Networks

- Stacking layers of perceptrons
- Backpropagation and gradient descent



Training in Practice

- Adaptive learning
- Batching
- Regularization

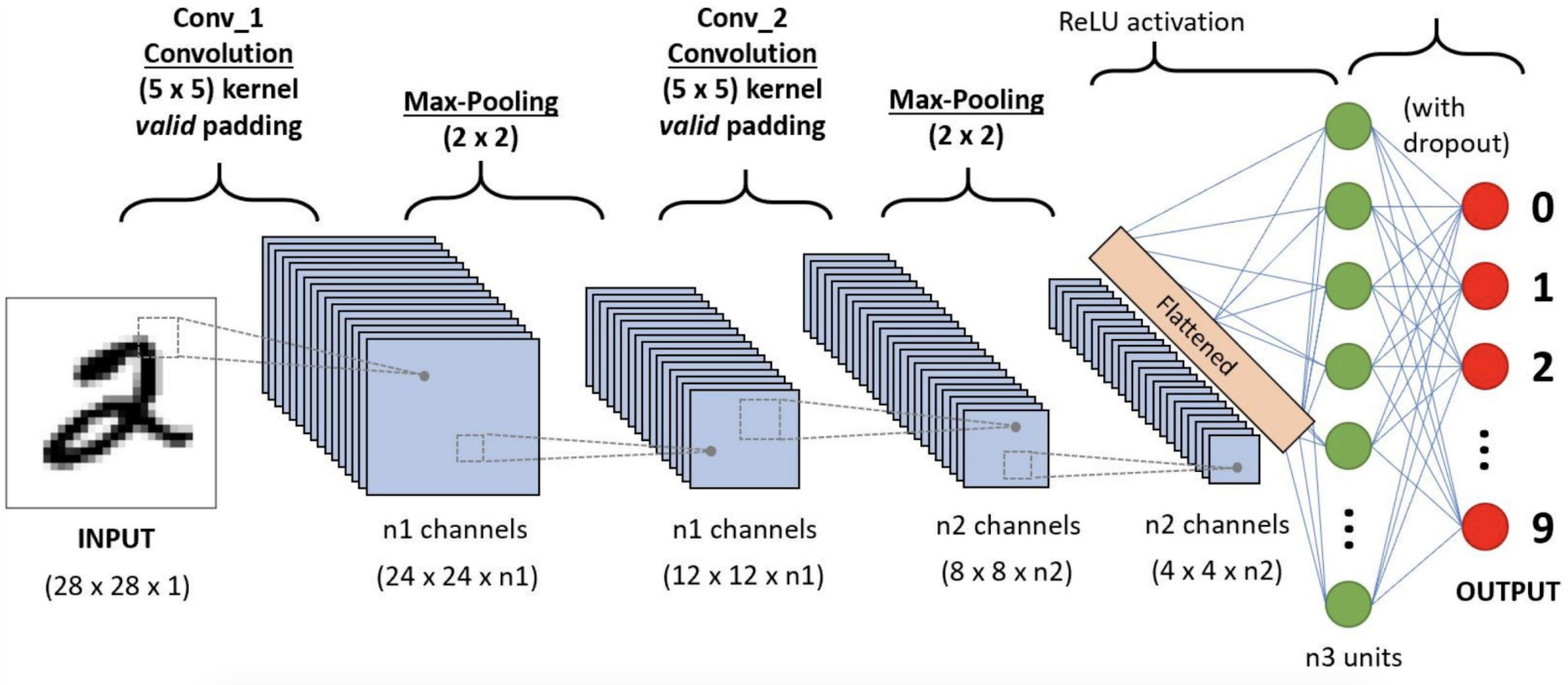


Please ask any questions!

Break

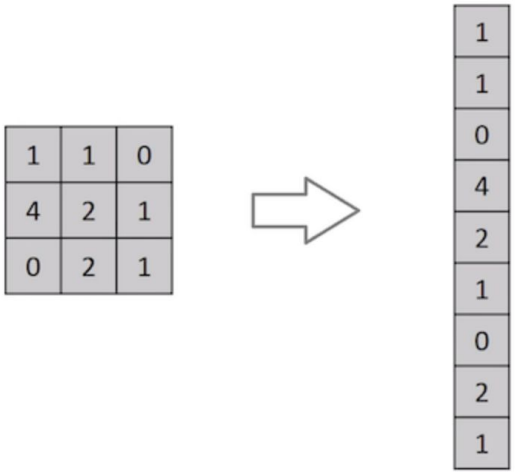
Convolutional Neural Networks

Let’s now deal with images!

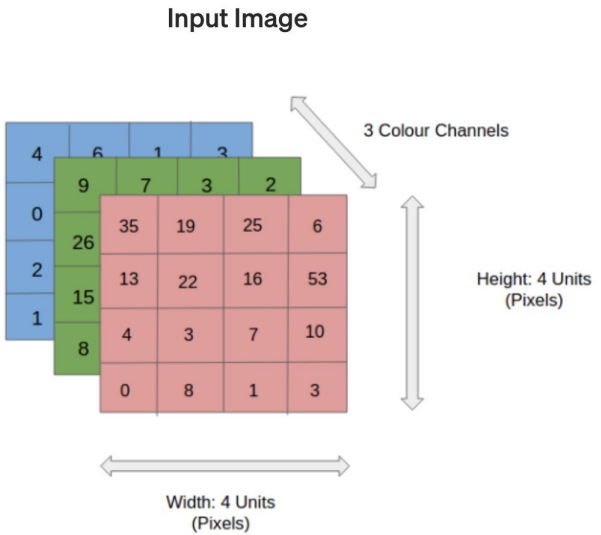


Images source:
https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

3-dimensional data

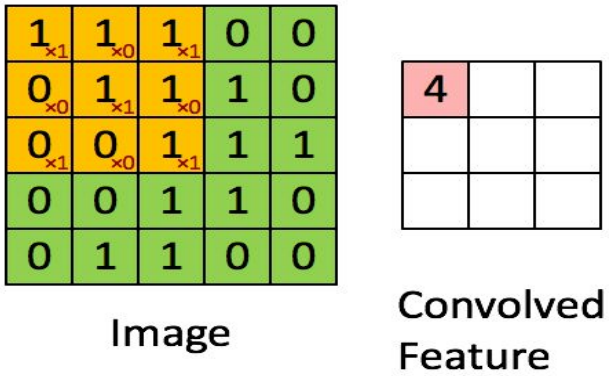


Flattening of a 3×3 image matrix into a 9×1 vector



Images source:
https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

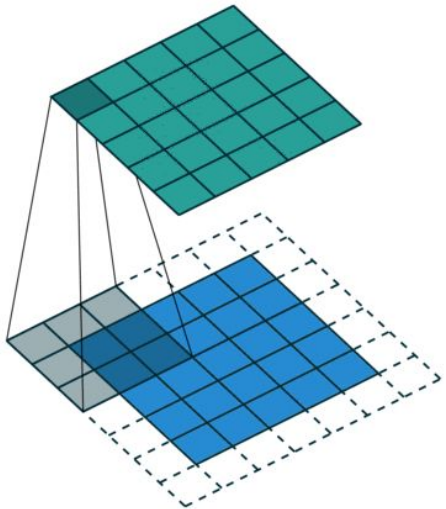
Convolution Operation



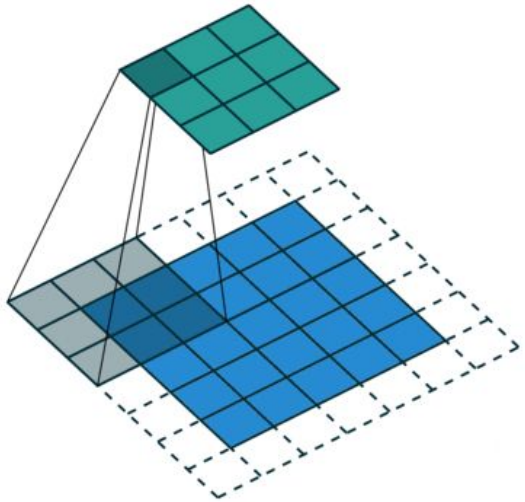
Images source:
https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

Convolution Operation

Padding: 1
Stride: 1

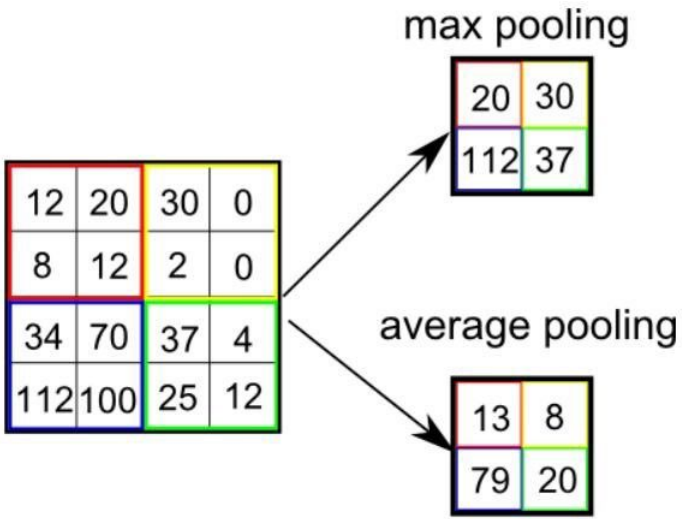


Padding: 1
Stride: 2



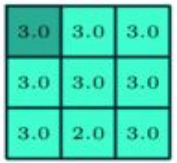
Images source:
https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

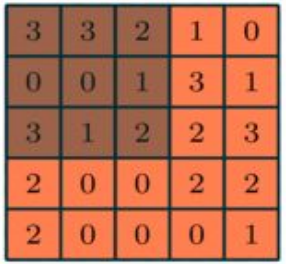
Pooling



Images source:
https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

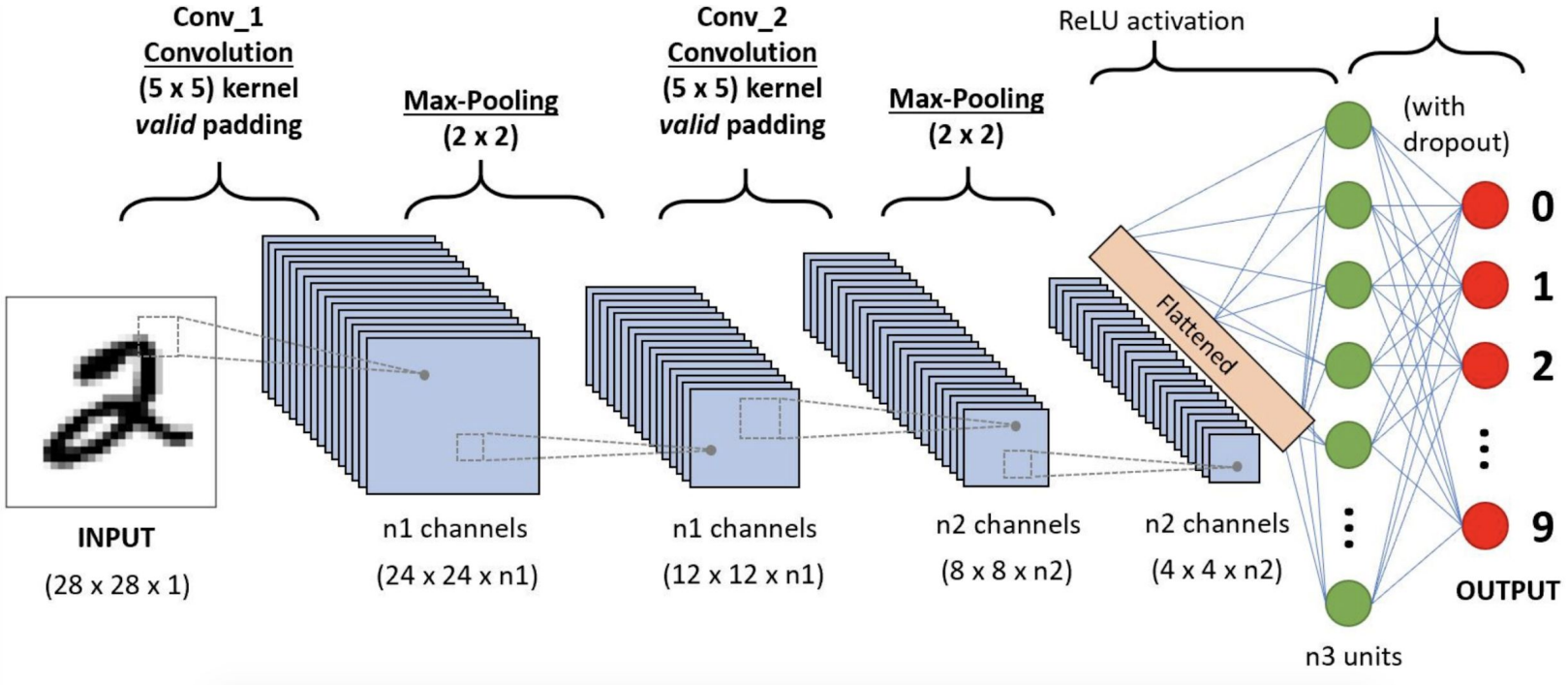
Pooling





Images source:
https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

Now we can deal with images!



Images source:

What kind of tasks can we perform?

- Classification, Regression
- Segmentation
- Prediction of next image
- Image generation
- …

Please ask any questions!

Thank you for your attention!