2695 Introduction to Machine Learning
Masters Program in Economics, Finance and Management

8a Video Lecture

INTRODUCTION TO NEURAL NETWORKS

Modeling a simple decision

Should I go for a run? (binary variable y: yes/no)

1. Is the weather good? x_1
2. Do I have energy? x_2
3. Do I have time? x_3

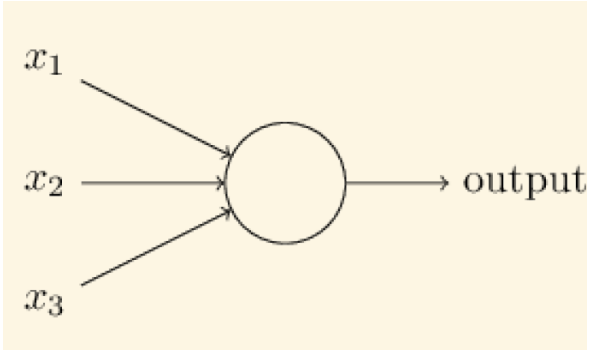
Binary variables $\mathbf{x_i}$: 0 is for answer 'no' and 1 for answer 'yes'.

Feature weights w_i: how important each feature is to the decision

$$y = \begin{cases} 0, & x_1 \cdot w_1 + x_2 \cdot w_2 + x_3 \cdot w_3 < \text{threshold} \\ 1, & x_1 \cdot w_1 + x_2 \cdot w_2 + x_3 \cdot w_3 \geq \text{threshold} \end{cases}$$

Material from: http://euler.stat.yale.edu/~tba3/stat665/

Perceptron



Dot product between the vector of all binary inputs (*x*), a vector of weights (*w*), and change the threshold to the negative bias (*b*):

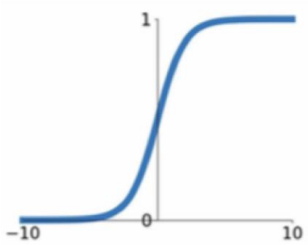
$$y = \begin{cases} 0, & x \cdot w + b < 0 \\ 1, & x \cdot w + b \geq 0 \end{cases}$$

Material from: http://euler.stat.yale.edu/~tba3/stat665/

From binary to continuous output

- Perceptron: small change in the input values can cause a large change in the output because each node only has two possible states: 0 or 1.
- Introduce non-linearity, example, sigmoid function

$$\sigma(x \cdot w + b) = \frac{1}{1 + e^{-(x \cdot w + b)}}$$



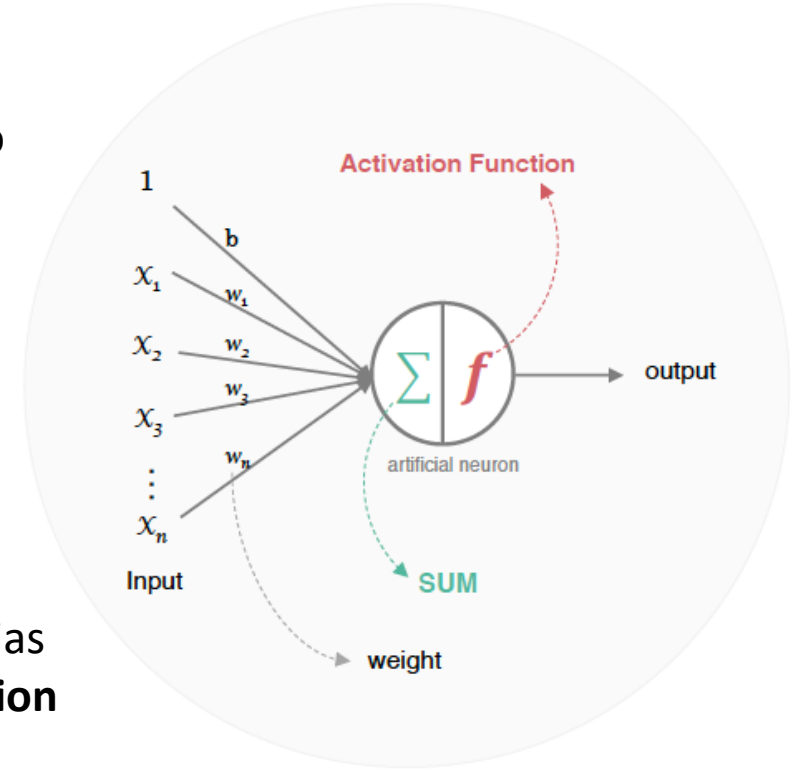
Sigmoid

- The choice of what function to use to go from *x w* + *b* to an output is called the **activation function.**
- There are many different activation functions used.

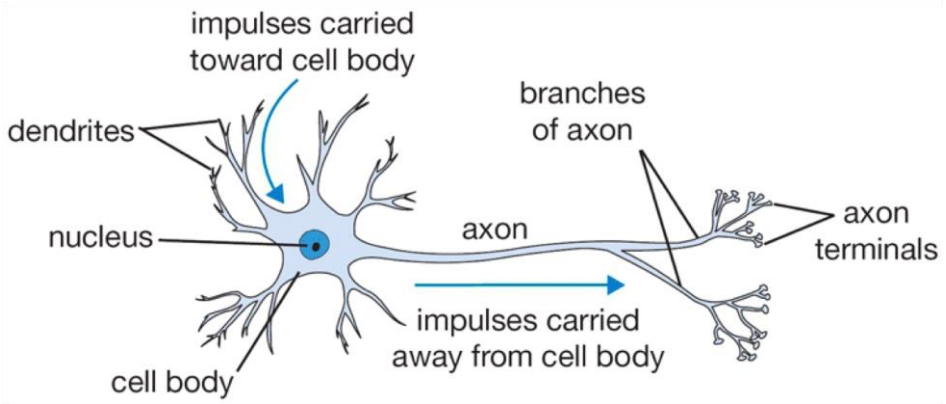
Structure of artificial neuron

- The artificial neuron receives one or more inputs and sums them to produce an output.
- Each input has some weight associated with it.

Weighted sum of input, plus the bias term, is passed through an **activation function**



Artificial neuron



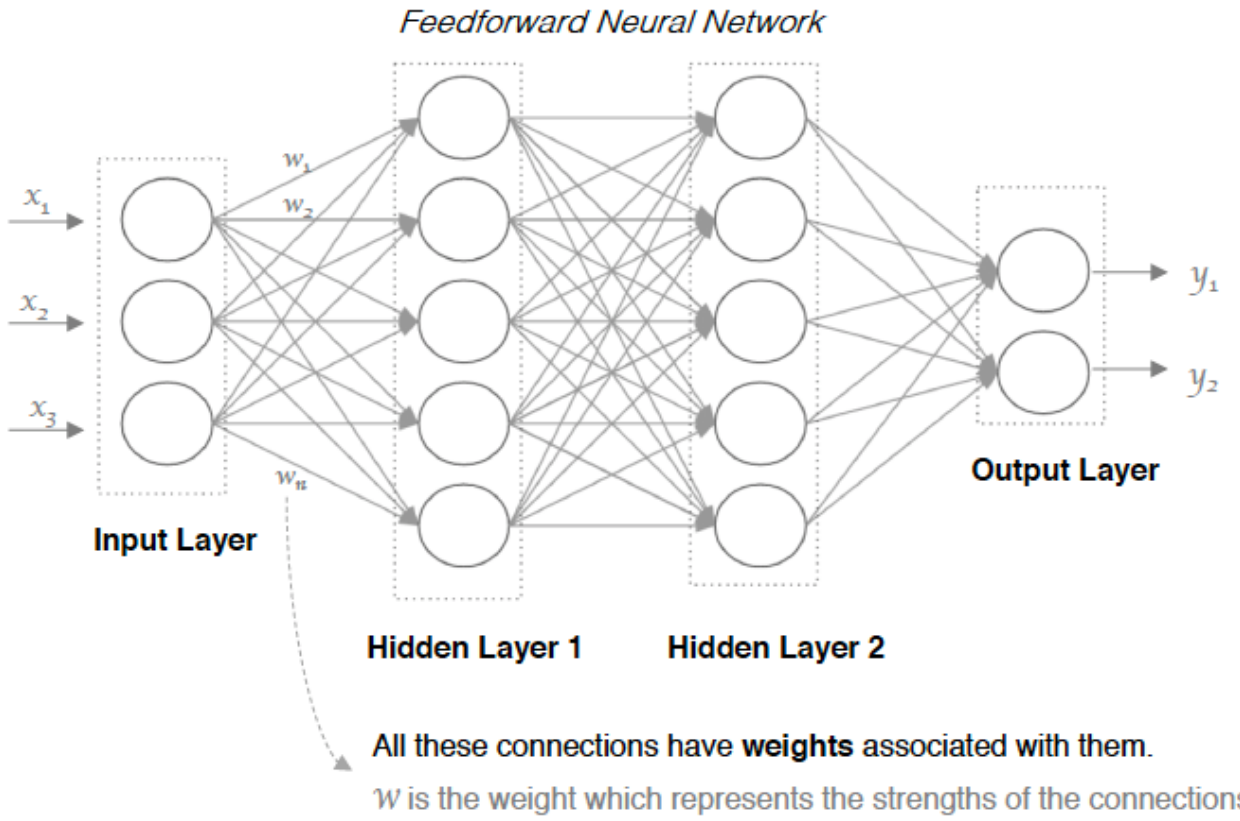
Biological neuron

Feedforward Neural Network

- **Artificial neural network:** network of artificial neurons (nodes).
- **Feedforward neural network:** artificial neural network where connections between the nodes do not form a cycle.
- Fully connected network: all the neurons, in one layer are connected to the neurons in the next layer.
- The network is trained by iteratively modifying the weights so that given inputs map to the correct response.

Input Layer

- Takes the features from the dataset
- Each node in this layer represents one feature from the input data

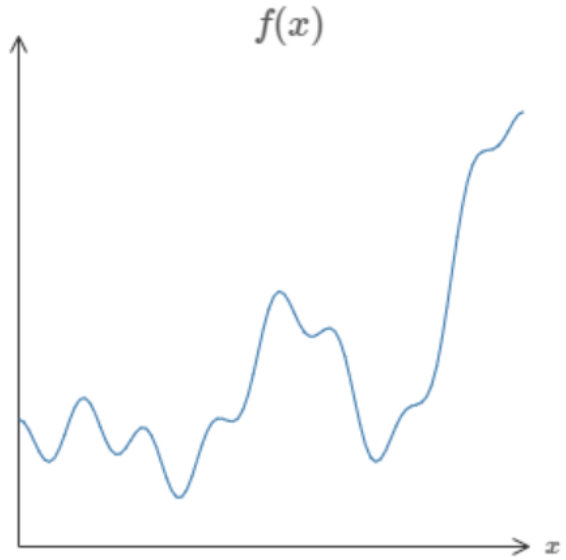


Output Layer

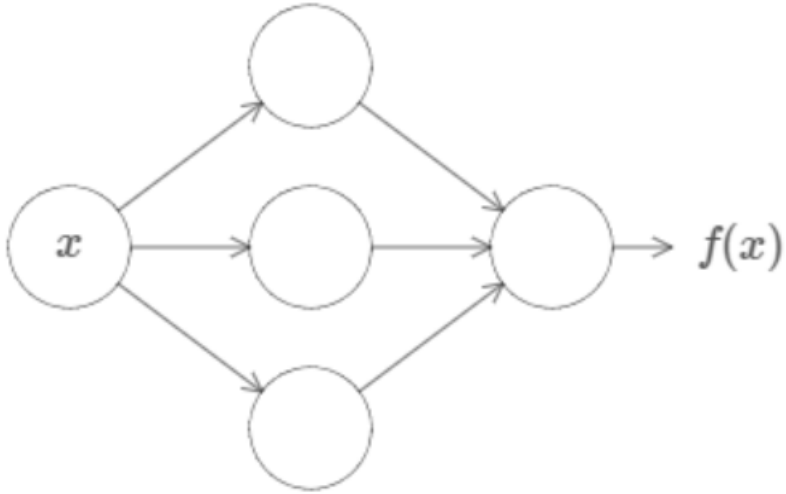
- Produces the final prediction

Why are neural networks so popular?

- **Universality theorem**: a neural network with a single hidden layer is capable of approximating any continuous function.



No matter what the function f(x) is, there is guaranteed to be a neural network such that for every possible input, x, outputs the value f(x) (or some close approximation).

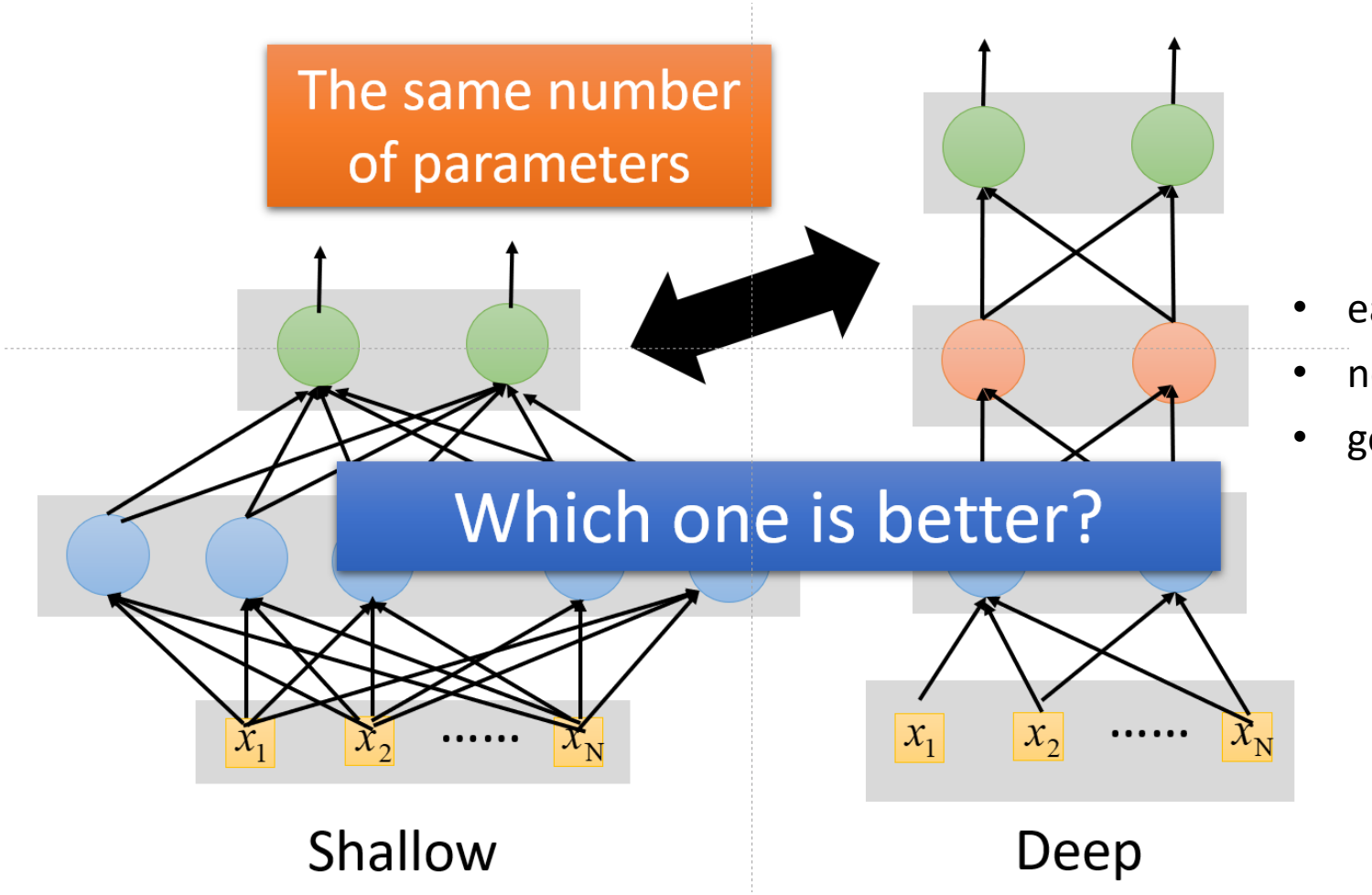


- Note: just because we know a neural network exists that can solve a very complex problem, that does not mean we have good techniques for constructing such a network.

http://neuralnetworksanddeeplearning.com/chap4.html

Why deep neural networks and not shallow wide networks?

- **Deep network** means it has many hidden layers.



- easier to train
- needs fewer parameters
- generalizes better

Training neural networks

- **Cost function** (**loss function**): measures how well the network predicts outputs given input.
- The goal is to then **find a set of weights and biases that minimizes the cost**.
- There are many different types of cost functions, example squared error loss:

$$C(w,b) = \frac{1}{2n}\sum_i (y_i - \widehat{y}(x_i))^2$$

- **Gradient descent**

$$w_{k+1} = w_k - \eta \cdot \nabla_w C$$
$$b_{k+1} = b_k - \eta \cdot \nabla_b C$$

η is the **learning rate.**

Stochastic gradient descent

- Total cost is a sum of costs associated for each data point i.

$$C = \frac{1}{n}\sum_i C_i \qquad C(w,b) = \frac{1}{2n}\sum_i (y_i - \hat{y}(x_i))^2 \qquad C_i(w,b) = \frac{1}{2}\ (y_i - \hat{y}_i)^2$$

- Weight update: $w_{k+1} = w_k - \frac{\eta}{n}\sum_{i=1}^{n} \nabla C_i\,(w_k)$ (Computationally expensive)
- Partition the input data into disjoint groups of m data points: $M_1; M_2, \ldots, M_{n/m}$.
- More efficient: Weight updates, by estimating the gradient using only a small subset of the entire training set:

$$\begin{aligned} w_{k+1} &= w_k - \frac{\eta}{m}\sum_{i\in M_1} \nabla C_i \\ w_{k+2} &= w_{k+1} - \frac{\eta}{m}\sum_{i\in M_2} \nabla C_i \\ &\vdots \\ w_{k+n/m+1} &= w_{k+n/m} - \frac{\eta}{m}\sum_{i\in M_{n/m}} \nabla C_i \end{aligned}$$

Each set M_j is called a **mini-batch.**

Going through the entire dataset as above is called an **epoch.**

Backpropagation and weight update

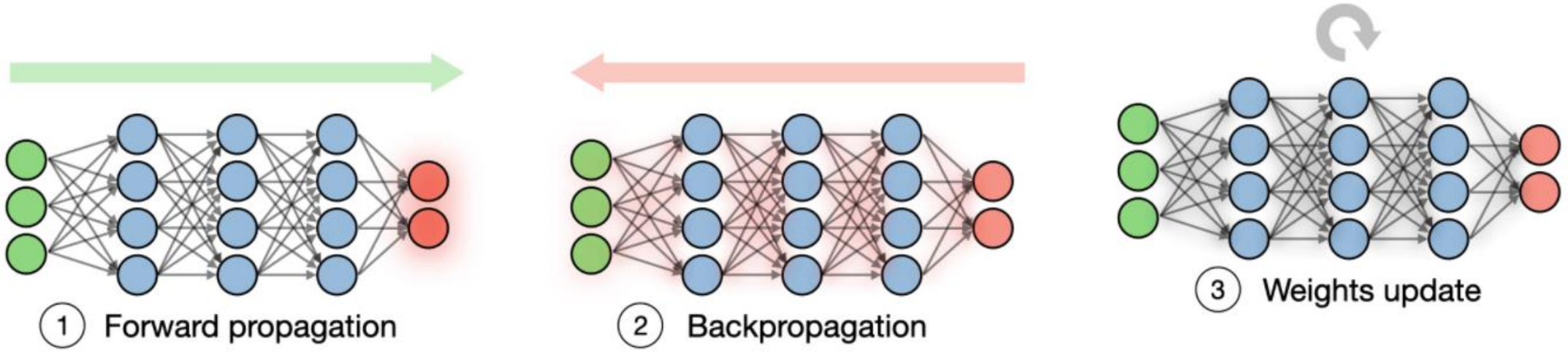
- Weights are updates as follows:

Step 1: Take a mini-batch of training data and perform forward propagation to compute the cost (C).

Step 2: Backpropagate the cost to get the gradient with respect to each weight.

Step 3: Use the gradients to update the weights of the network.

$$w_{k+1} = w_k - \eta \cdot \nabla_w C$$
$$b_{k+1} = b_k - \eta \cdot \nabla_b C$$



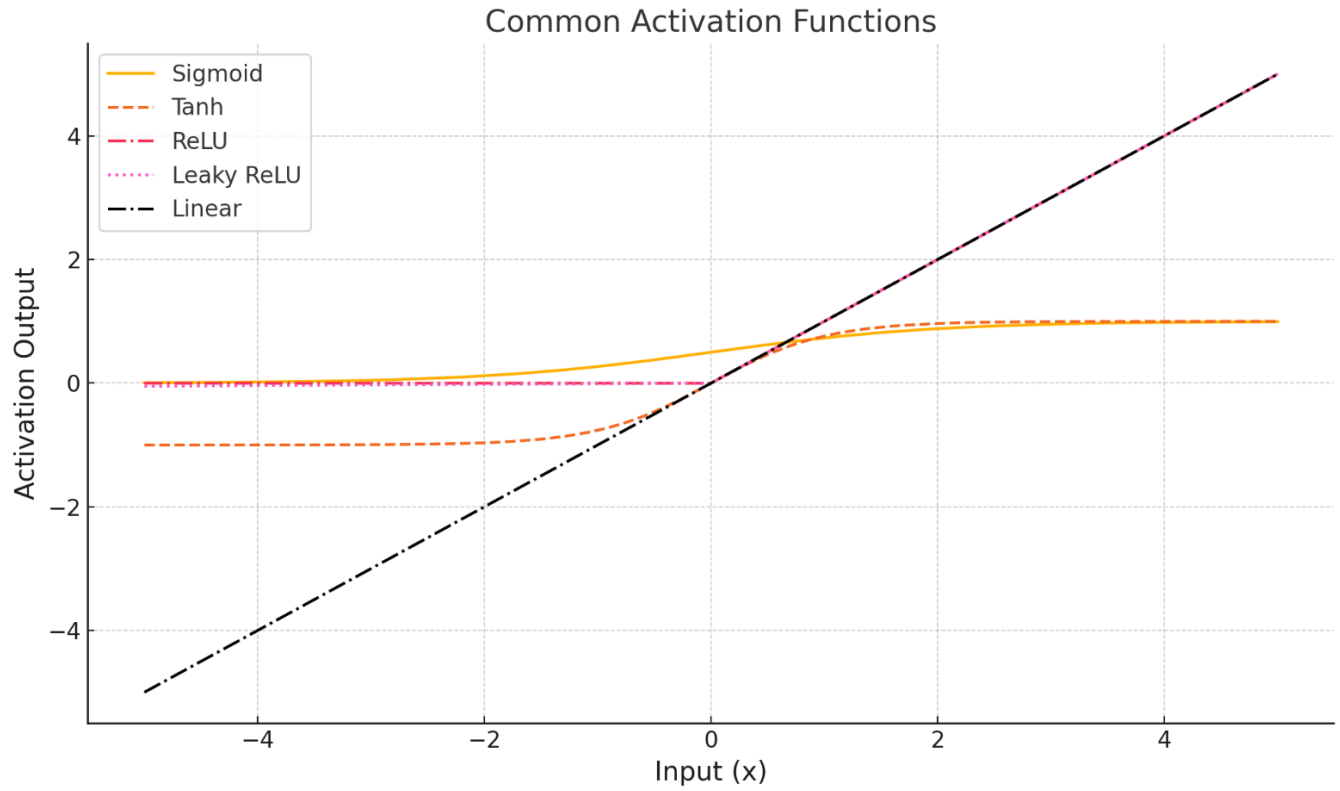
Neural network architecture

Even for a basic Neural Network, there are many design decisions to make:

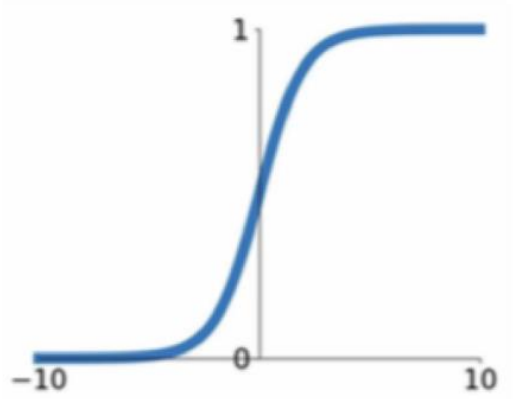
- Number of hidden layers (depth)
- Number of neurons per hidden layer (width)
- Type of activation function
- Choice of cost function
- Hyperparameters: learning rate, batch size, number of epochs, regularization methods...
- …

Activation functions

- Function to use to go from the weighted sum of the inputs to an output**.**
- Performs a non-linear transformation to the input making it capable to learn and perform more complex tasks.



Sigmoid and tanh activation function



Sigmoid

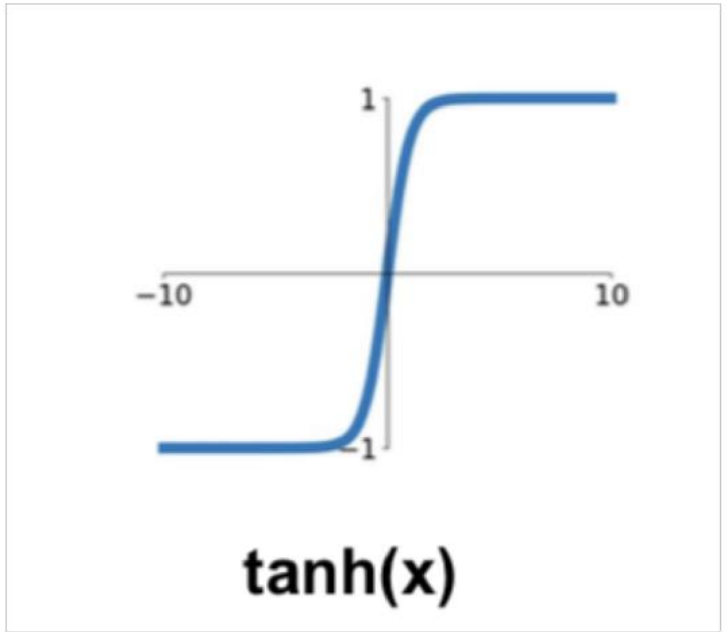
$$f(x) = \sigma(x) = \frac{1}{1 + e^{(-\alpha \cdot x)}}$$

$$f'(x) = \sigma'(x) = \alpha \cdot \sigma(x) \cdot (1 - \sigma(x))$$

Range: 0 to 1

Historically popular, recently not used as much:

- sigmoid outputs are not zero-centered
- exponential is a bit expensive to compute
- vanishing gradients problem



Hyperbolic tangent:

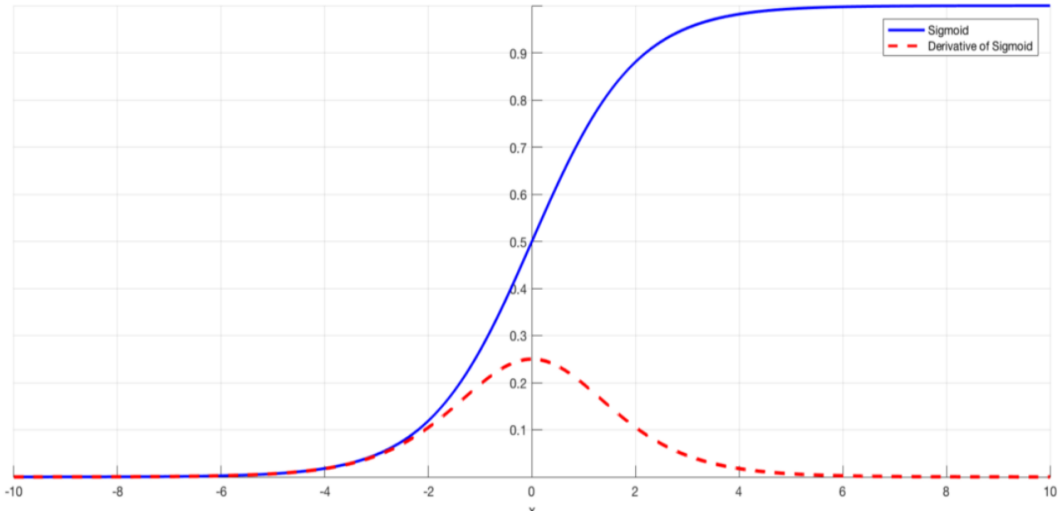
$$\tanh x = \frac{\sinh x}{\cosh x} = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{e^{2x} - 1}{e^{2x} + 1}.$$

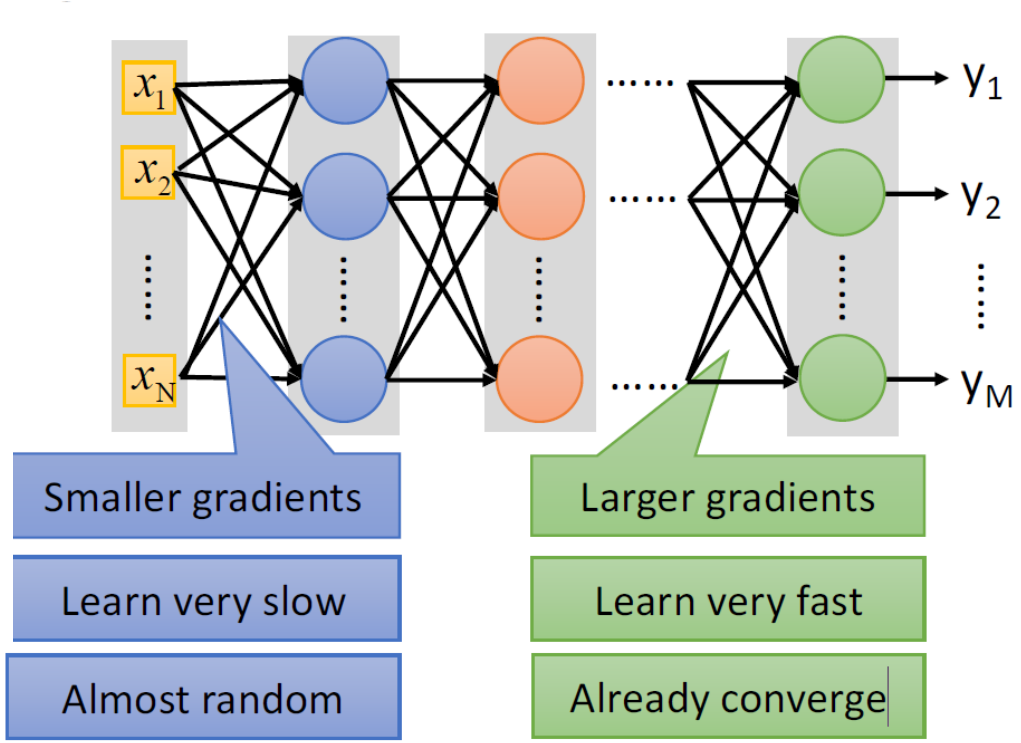
Range between -1 and 1:

- Outputs are zero centered
- Vanishing gradient problem

Vanishing gradient problem

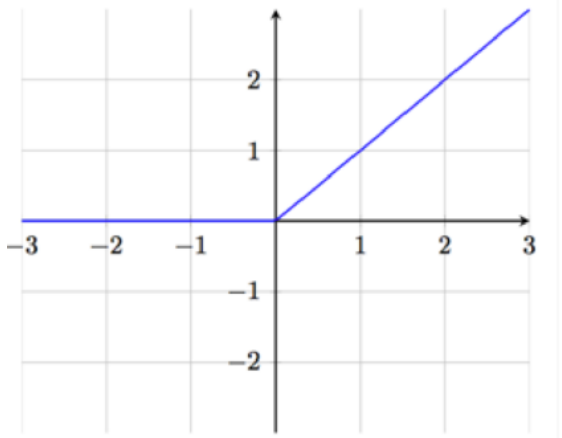
- The derivatives of each layer are multiplied down the network, from the final layer to the initial, to compute the derivatives of the initial layers.
- When *n* hidden layers use an activation like the sigmoid function, *n* small derivatives are multiplied together.
- The gradient decreases as we propagate down to the initial layers and the weights.
- A small gradient means that the weights and biases of the initial layers will not be updated effectively with each iteration.





Rectified Linear Unit (ReLU) and Leaky Rectified Linear Unit

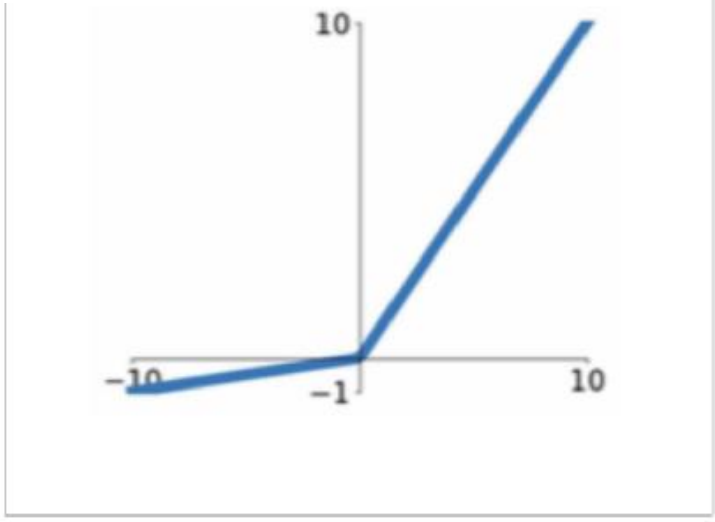
$f(x) = max(0,x)$



ReLu function

- Very computationally efficient
- Converges much faster than sigmoid/tanh in practice
- But: Not zero-centered output
- When x<0, the node “dies”

$f(x) = \max(\alpha x, x)$



- Computationally efficient
- Converges much faster than sigmoid/tanh in practice
- For x<0, the node will not die
- Not zero-centered output

SoftMax activation function

- Squashes the outputs between 0 and 1, just like a Logistic/Sigmoid function.
- Used in the output layer for multi-class classification
- Converts a vector of raw prediction scores into probabilities that sum up to 1, the class with the highest probability is selected

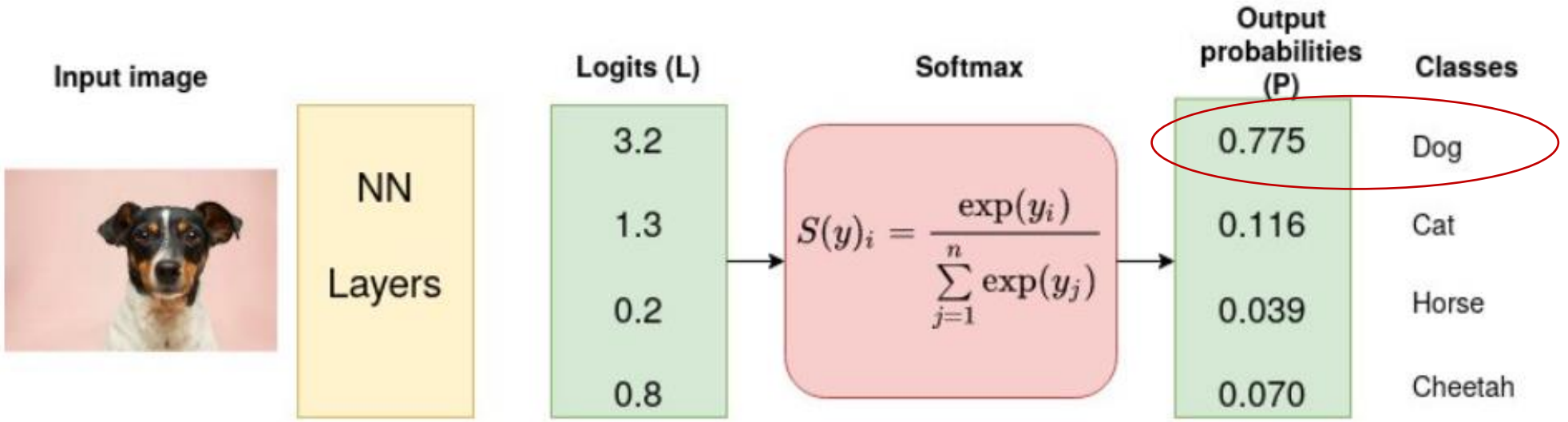
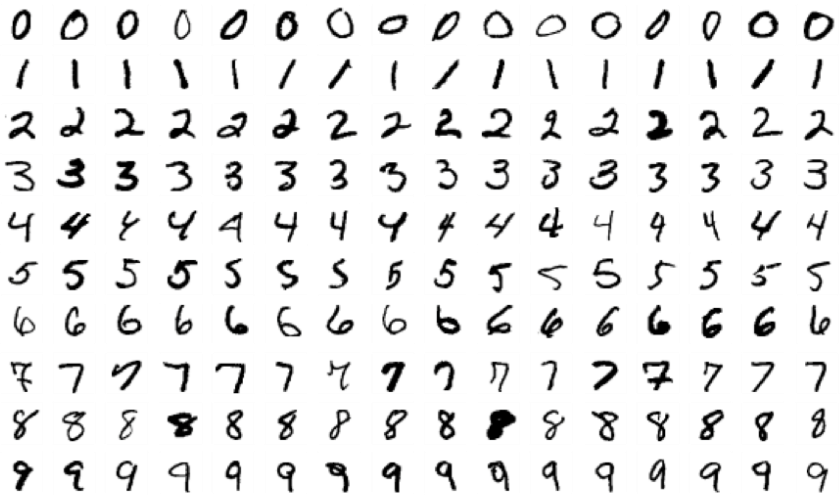


Image from: https://towardsdatascience.com/softmax-activation-function-how-it-actually-works-d292d335bd78

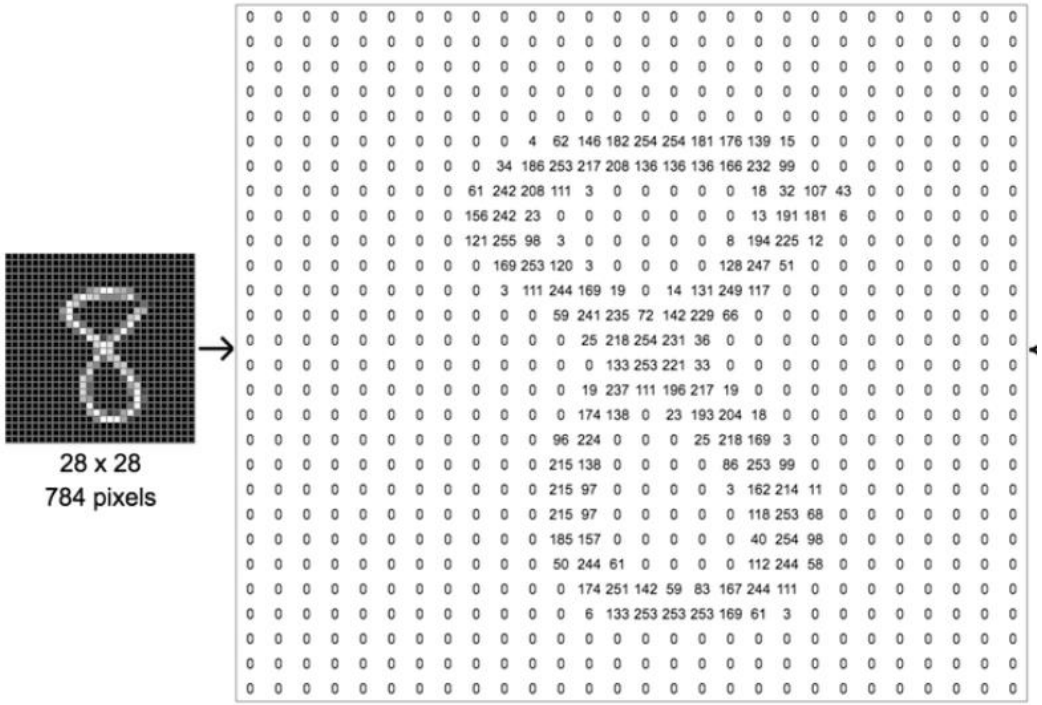
Neural Network example: MNIST classification

- The MNIST database (Modified National Institute of Standards and Technology database): large dataset of handwritten digits.
- Each image is 28 pixels by 28 pixels which has been flattened into 1-D array of size 784.
- The network would have:
  - 784 input neurons: one for each pixel
  - 10 output units, one for each digit 0 to 9: input an image of a number 8, the output unit corresponding to the digit 8 would be activated.

Handwritten digits

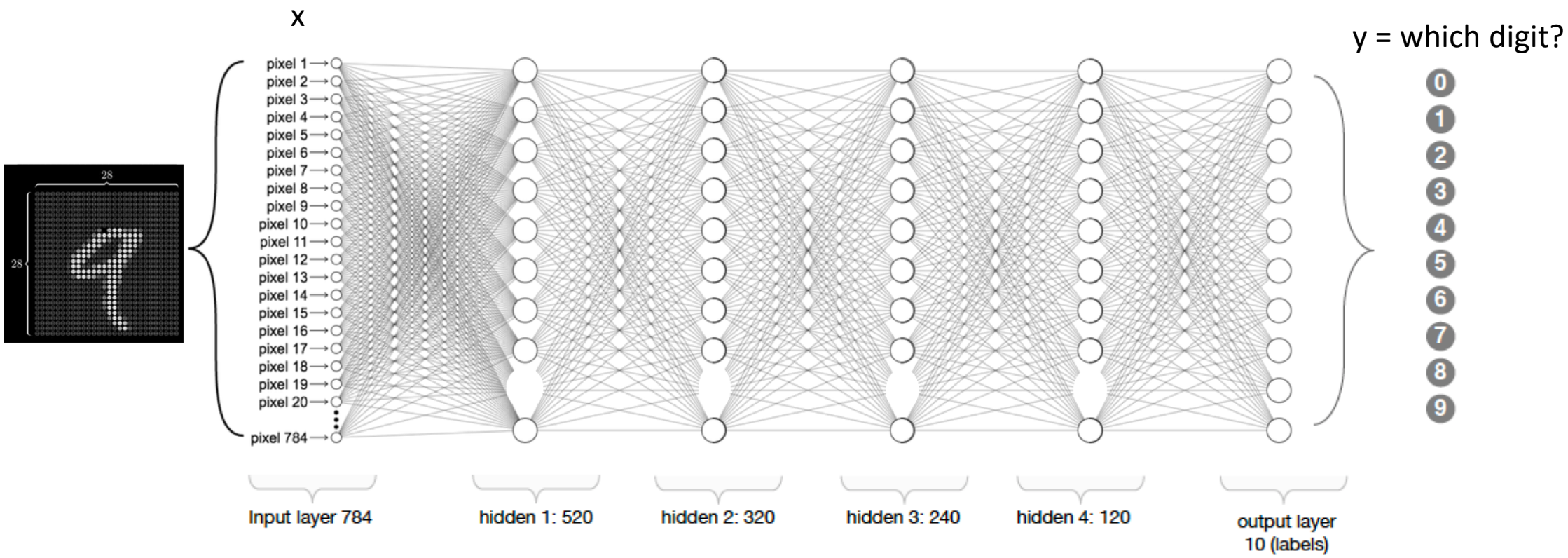


collection of 784-dimensional vectors.



Famous MNIST dataset for handwritten digits recognition

Neural Network for digit classification



Choosing number of layers and number of neurons

Number of layers

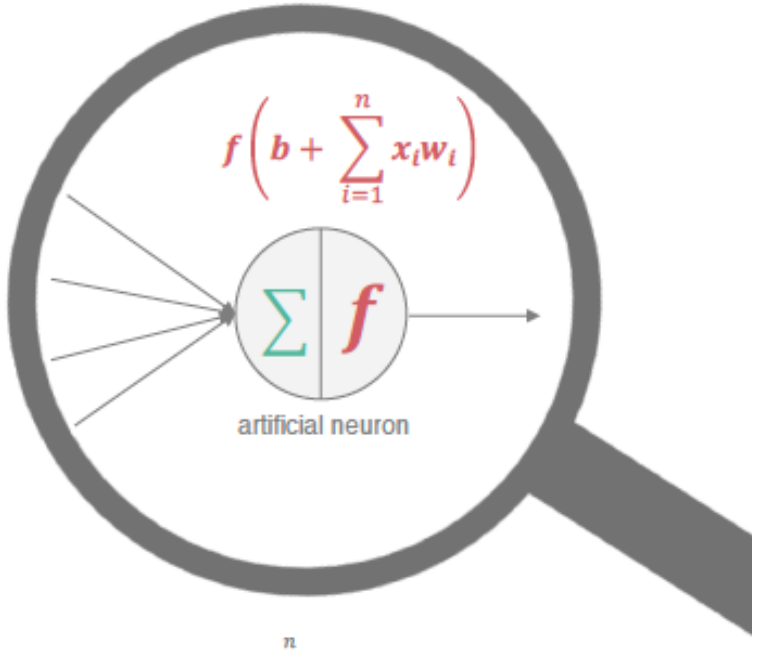
- start with one or two hidden layers
- for more complex problems: gradually increase the number of layers until we start overfitting the training dataset (check performance on the validation dataset)
- very complex problems: 10s or 100s of hidden layers, but network not trained from scratch, use pretrained one

Number of neurons

- the number of neurons in the input and output layers is determined by the type of input and output a given task requires
- previously common: use fewer and fewer nodes at each layer
- nowadays used: same number of neurons at each layer
- gradually increase the number of neurons until the network starts overfitting
- generally increasing the number of layers tends to be more useful than increasing the number of neurons per layer
- start with more layers and neurons and use **early stopping**: interrupt training when it measures no progress on the validation set for a predefined number of epochs

Choosing activation function

- How fast the network learns
- How well the network generalizes



$$z = b + \sum_{i=1}^{n} x_i w_i$$

$$f(z) = f(b + \sum_{i=1}^{n} x_i w_i)$$

Hidden Layer

- By Default, use **ReLu** function
- In case that ReLu doesn't provide good results, then try other activations like Leaky Relu
- Do not use **Sigmoid and Tanh** function for deep networks due to the vanishing gradient problem.

Output Layer

- Use Logistic or SoftMax function for classification problem
- Use Linear or ReLu function for regression problem, if specific range sigmoid or tanh

Common cost functions

Measures how far off the model's predictions are from the true labels and It provides a single number that the model tries to minimize during training

Regression:

- Same cost as Linear Regression: **Mean squared error**
- Many outliers: Mean absolute error

Classification:

- Same cost as Logistic Regression: **Cross-entropy (log loss)**
  - cost high: estimated probability close to 0 for a positive instance or close to 1 for a negative one
  - cost close to 0: estimated probability is close to 0 for a negative instance or close to 1 for a positive one

Typical architecture

REGRESSION

Hyperparameter	Typical Value
# input neurons	One per input feature (e.g., 28 x 28 = 784 for MNIST)
# hidden layers	Depends on the problem. Typically 1 to 5.
# neurons per hidden layer	Depends on the problem. Typically 10 to 100.
# output neurons	1 per prediction dimension
Hidden activation	ReLU (or SELU, see Chapter 11)
Output activation	None or ReLU/Softplus (if positive outputs) or Logistic/Tanh (if bounded outputs)
Loss function	MSE or MAE/Huber (if outliers)

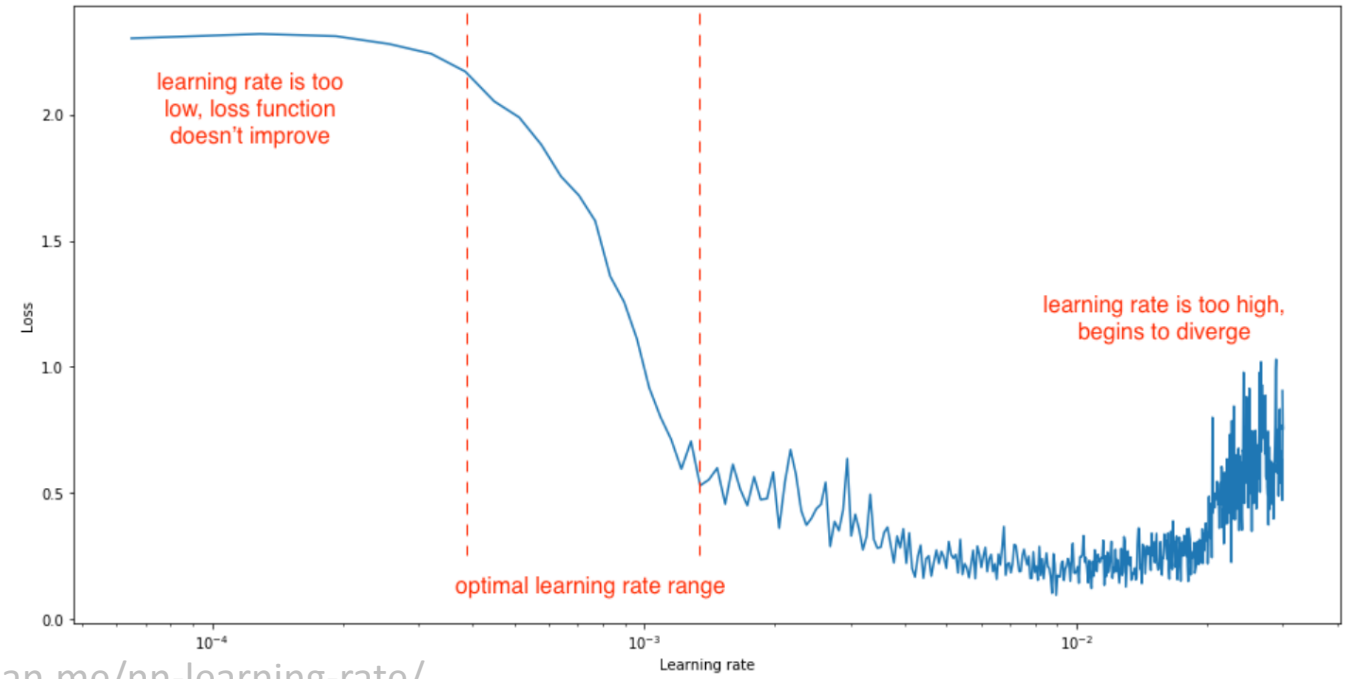
CLASSIFICATION

Hyperparameter	Binary classification	Multilabel binary classification	Multiclass classification
Input and hidden layers	Same as regression	Same as regression	Same as regression
# output neurons	1	1 per label	1 per class
Output layer activation	Logistic	Logistic	Softmax
Loss function	Cross-Entropy	Cross-Entropy	Cross-Entropy

Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition, Aurélien Géron

Learning rate

- Hyperparameter that **directly affects how fast or slow the model learns**
- **Learning rate annealing**: start with a relatively high learning rate and then gradually lower the learning rate during training.
  - Step decay: learning rate is reduced by some percentage after a set number of training epochs.
- Given computational budget: err on the side of slower decay and train for a longer time.
- **Adaptive leaning rate:** Adaptive Moment estimation (Adam), adapts learning rate per parameter



https://www.jeremyjordan.me/nn-learning-rate/

Weight initialization

Random initialization: Gaussian with zero mean and standard deviation 1

- Works for small networks, but causes problems for deeper networks:
  - Almost all neurons completely saturated, either -1 and 1.
  - Gradients will be all zero.

- **Xavier initialization:** Weights from a Gaussian distribution with:
  - zero mean
  - variance of 1/N (N specifies the number of nodes from the previous layer that are incoming to this given node)
  - Not the best choice for ReLU

- **He initialization:** Weights from a Gaussian distribution with:
  - zero mean
  - variance of 2/N
  - Works well for ReLU

Regularization

- As the size of neural networks grow, the number of weights and biases can quickly become quite large.
- Neural networks often have millions of weights to learn (Large Language models have billions).
- Regularization:
  - Early stopping
  - Add a penalty term to the cost function.

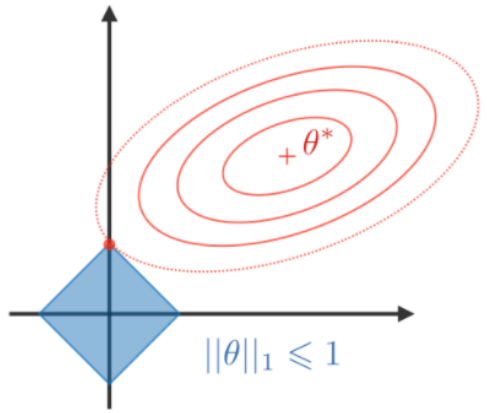
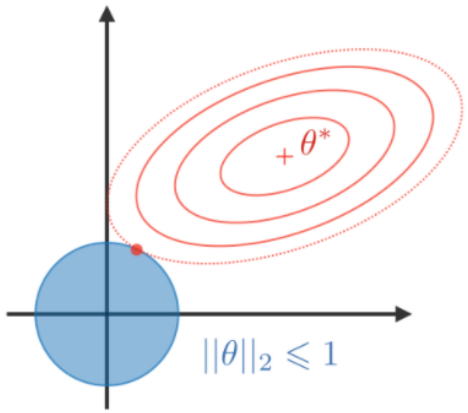
Common choices are the L2-norm:

$$C = C_0 + \lambda \sum_i w_i^2$$

and the L1-norm:

$$C = C_0 + \lambda \sum_i |w_i|.$$

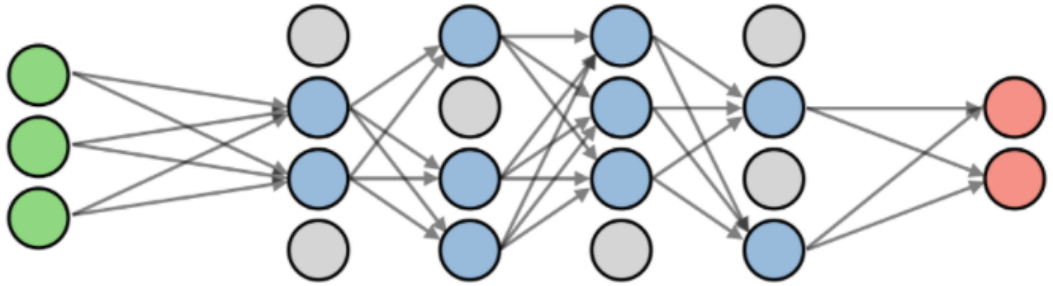
where C_0 is the unregularized cost.

LASSO	Ridge
• Shrinks coefficients to 0 • Good for variable selection	Makes coefficients smaller
	
$\ldots + \lambda \|\|\theta\|\|_1$ $\lambda \in \mathbb{R}$	$\ldots + \lambda \|\|\theta\|\|_2^2$ $\lambda \in \mathbb{R}$

Dropout

- The output of a randomly chosen subset of the neurons is temporarily set to zero during the training of a given mini-batch.
- Impact: neurons cannot overly adapt to the output from prior layers as these are not always present.
- Wide-spread adoption massive empirical evidence to its usefulness.
- Significantly improves training speed.
- Hyperparameter: probability of dropping
- Only active during training, not during inference

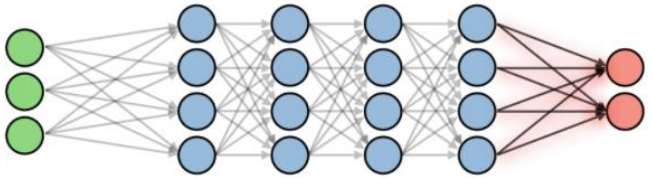
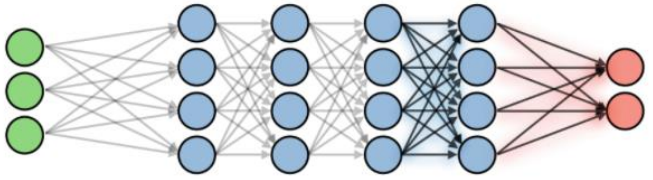
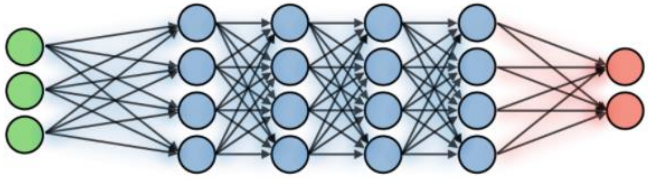
❒ **Dropout** — Dropout is a technique used in neural networks to prevent overfitting the training data by dropping out neurons with probability $p > 0$. It forces the model to avoid relying too much on particular sets of features.



Remark: most deep learning frameworks parametrize dropout through the 'keep' parameter $1 - p$.

Transfer Learning

- Training a deep learning model requires a lot of data and a lot of time.
- Transfer learning: use some already trained neural network that accomplishes a similar task to the one we are trying to tackle and then just reuse the lower layers of this network.
- Take advantage of pre-trained weights on huge datasets that took days/weeks to train and leverage it towards our use case.
- It will speed up training considerably and it will also require much less training data.

Training size	Illustration	Explanation
Small		Freezes all layers, trains weights on softmax
Medium		Freezes most layers, trains weights on last layers and softmax
Large		Trains weights on layers and softmax by initializing weights on pre-trained ones

8b Video Lecture

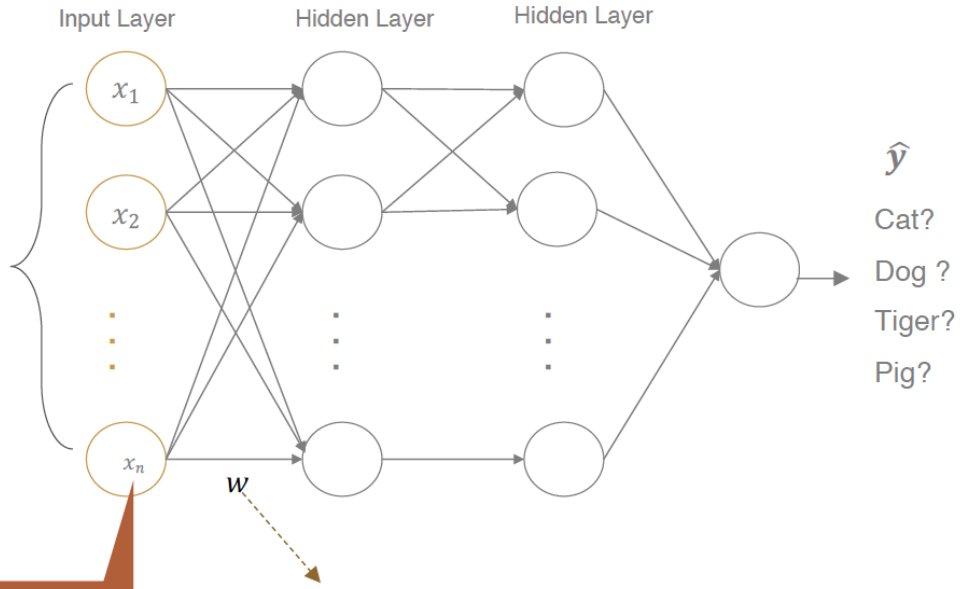
CONVOLUTIONAL NEURAL NETWORKS

Traditional neural network for image classification

Although a regular deep neural network could work on small images using today's computing power, it doesn't work well for large images because of the **huge number of parameters** it requires.

Image is
1000 × 1000 pixels with
3 color channels (RGB),
then, this results in a large
number of parameters.

1000 × 1000 × 3 = 3,000,000

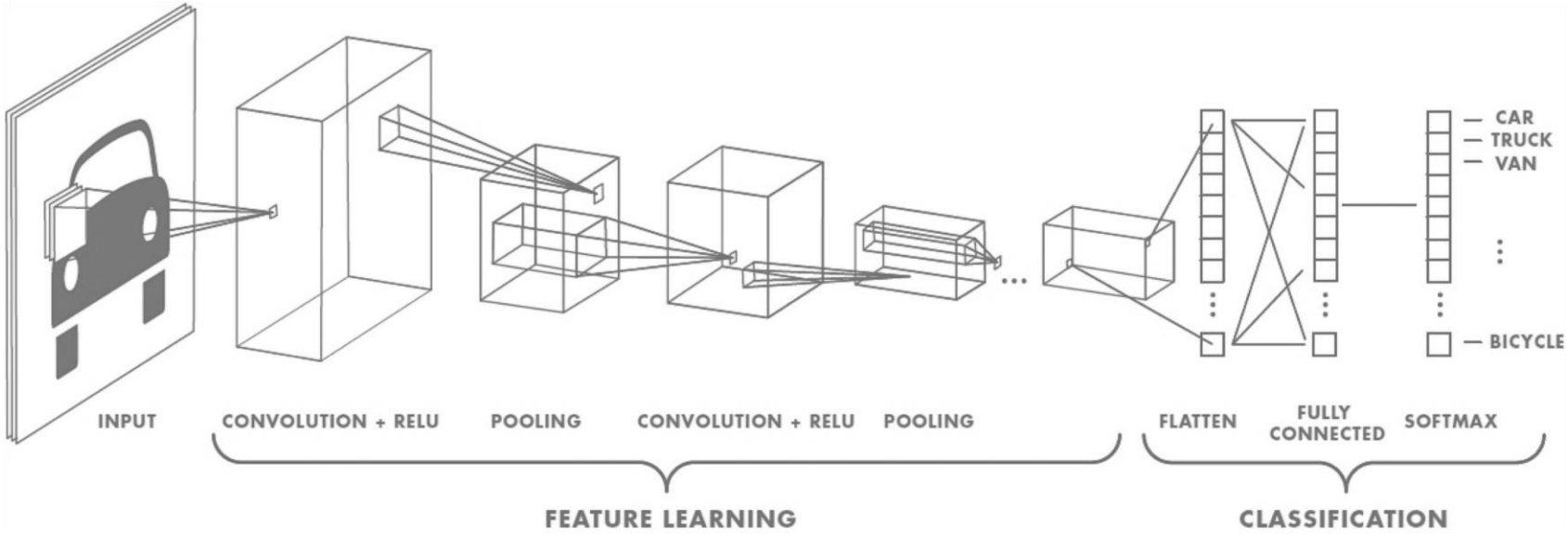


3 millions of input features

If the 1st hidden layer has 1,000 hidden units, then the total number of **weights** will be **3 billions** (i.e. 3,000,000 * 1000 = 3,000,000,000). And that's just the first layer.

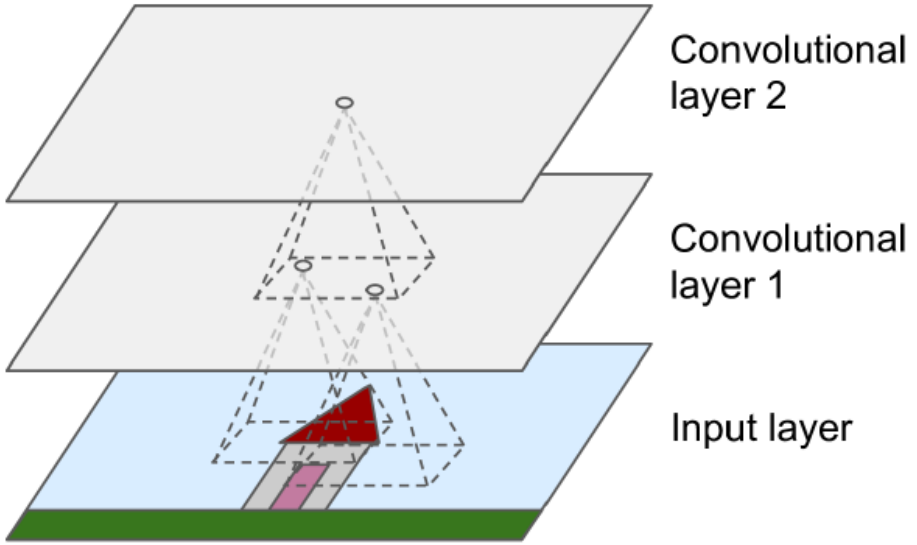
Architecture of Convolutional Neural Networks

- There are three main types of layers to build CNN architectures:
  - **Convolutional Layer**
  - **Pooling Layer**
  - **Fully-Connected Layer** (FC, exactly as used in regular Neural Networks)
- CNN model a combination of two components: **feature learning part** and **classification part**.
- After learning features in many layers, the architecture of a CNN shifts to classification.
- The final layer of the CNN architecture uses a classification layer such as SoftMax to provide the classification output.



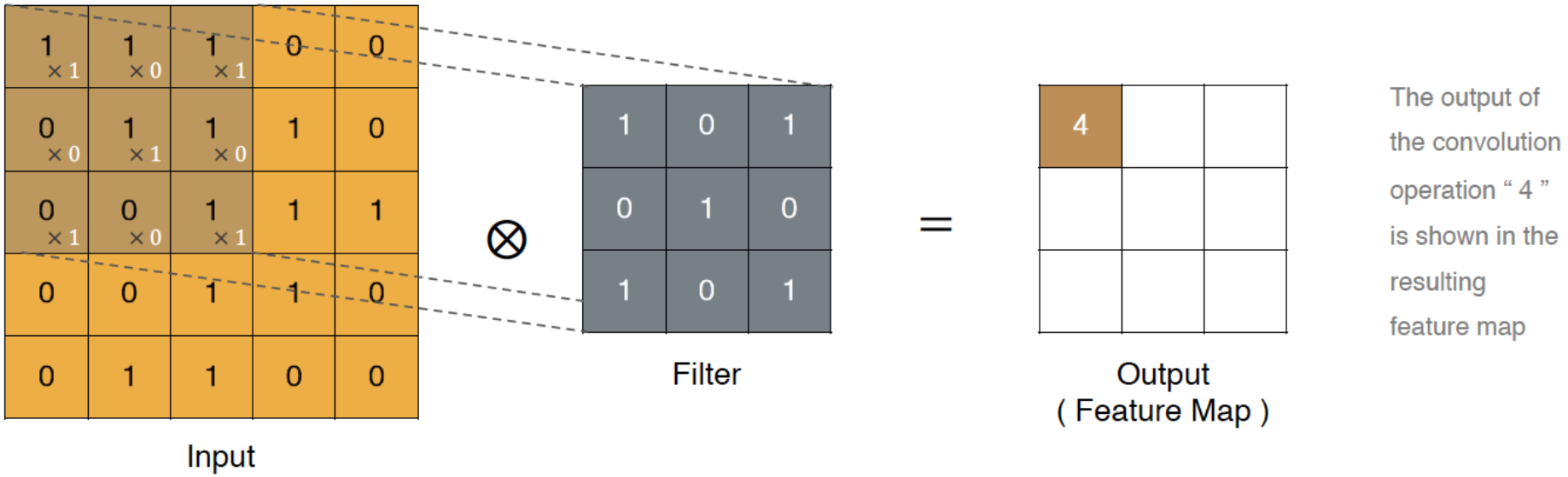
Convolutional layer

- Neurons in the first convolutional layer are not connected to every single pixel in the input image, but **only to pixels located within a small rectangle** (receptive field).
- Each neuron in the second convolutional layer is connected only to neurons located within a small rectangle in the previous layer.
- A neuron's set of weights is **filter** (**kernel**). Filter size: design choice, the values of weights learnt in training
- The convolution operation is performed by sliding the convolution filter over the input image: performing dot products between the filters (weights) and local regions of the input. The resulting output is called **feature map** or activation map.



How convolution works

- Convolution performs a dot product between two matrices: filter and the receptive field in the input image (small region to which a neuron is connected to)
- Stride: How many pixels does the convolutional filter move each time

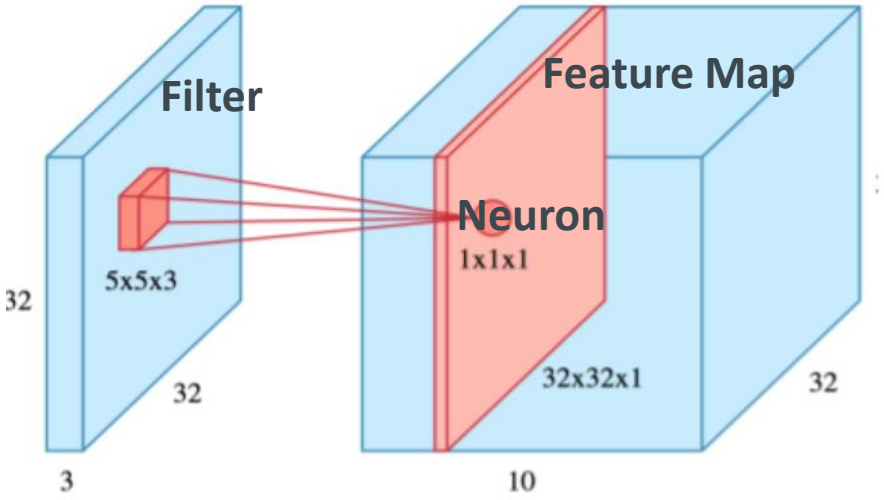


Slide the 3 x 3 filter over the input image, element-wise multiply, and sum the result.

1 x 1 + 1 x 0 + 1 x 1 + 0 x 0 + 1 x 1 + 1 x 0 + 0 x 1 + 0 x 0 + 1 x 1 = 4

Parameter sharing

- Convolution compresses an image or a previous layer into a feature map and one layer can have multiple filters, each outputs a feature map
- The weight and biases for a given neuron are only non-zero over a small, local region
- All neurons in a feature map share the same parameters: **parameter sharing**
  - dramatically reduces the number of parameters to learn
  - the same filter applied everywhere
- Feature maps are typically followed elementwise through an activation function such as ReLU.



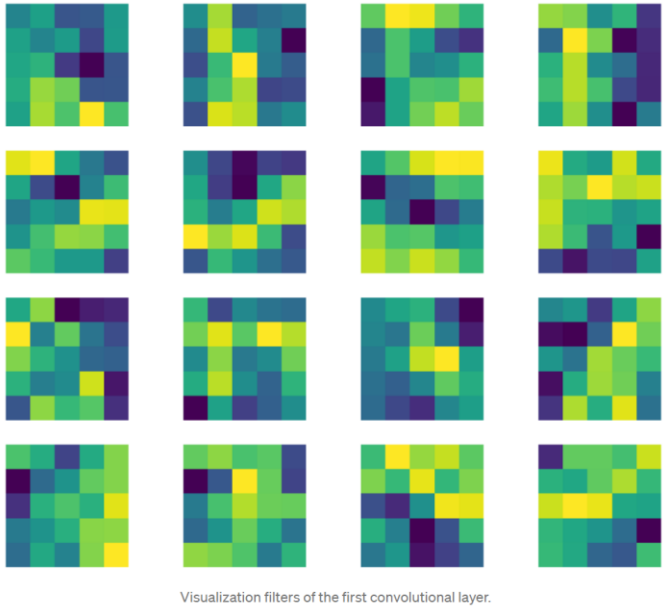
Note : 10 filters of size 5x5x3 are deployed

Example:

- Image of the size 32x32x3
- Filter size 5x5
- Each neuron will have non-zero weights to 5x5x3 region
- Total non-zero weights per neuron: 5*5*3=75 (+bias)

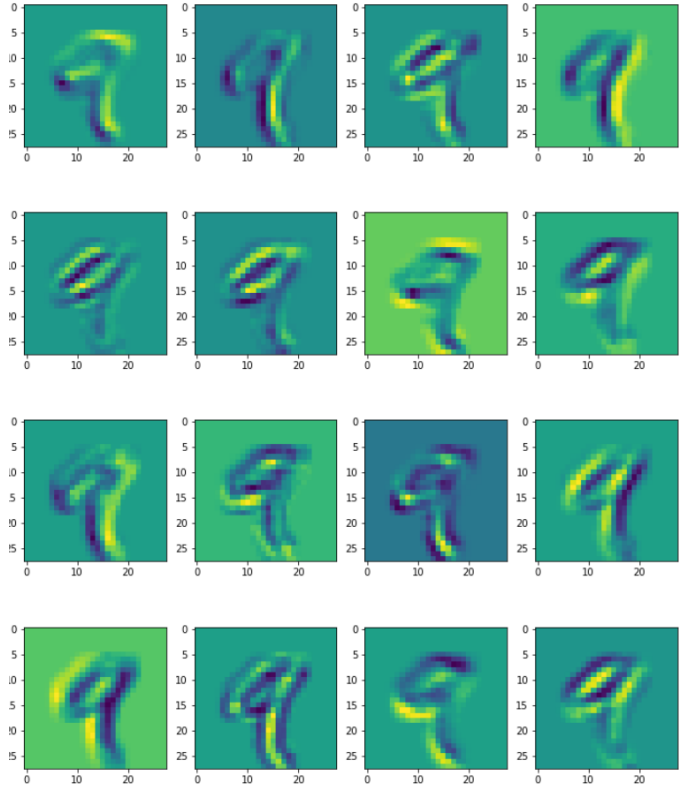
Feature maps

- Each feature map highlights where certain features are present in the input.
- The network learns during training what filters (features) are useful for the task
- We can visualize some filters and feature maps, but we cannot always exactly what every feature represents especially deeper in the network



Visualization filters of the first convolutional layer.

FILTERS

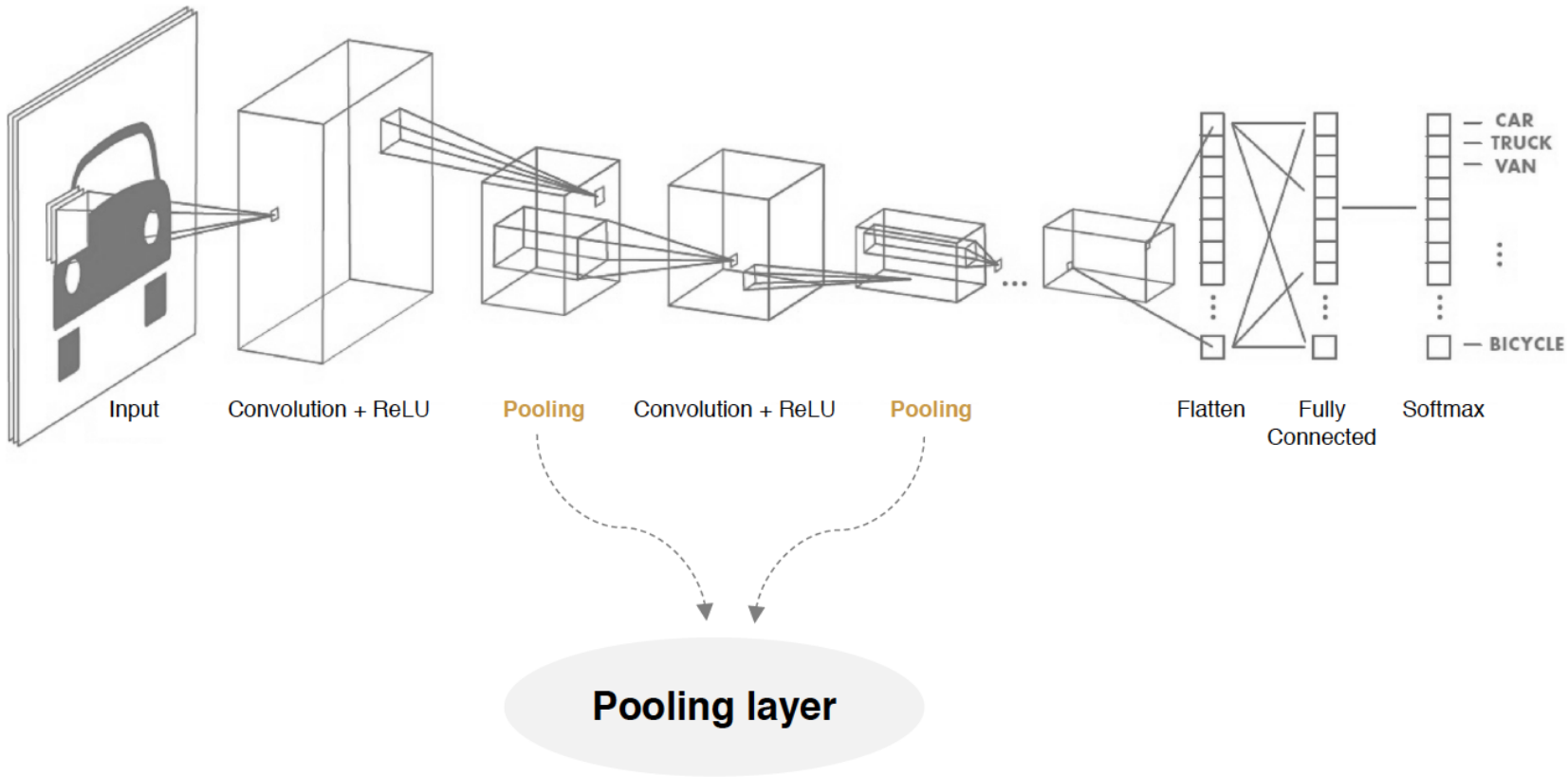


FEATURE MAPS

https://medium.com/dataseries/visualizing-the-feature-maps-and-filters-by-convolutional-neural-networks-e1462340518e

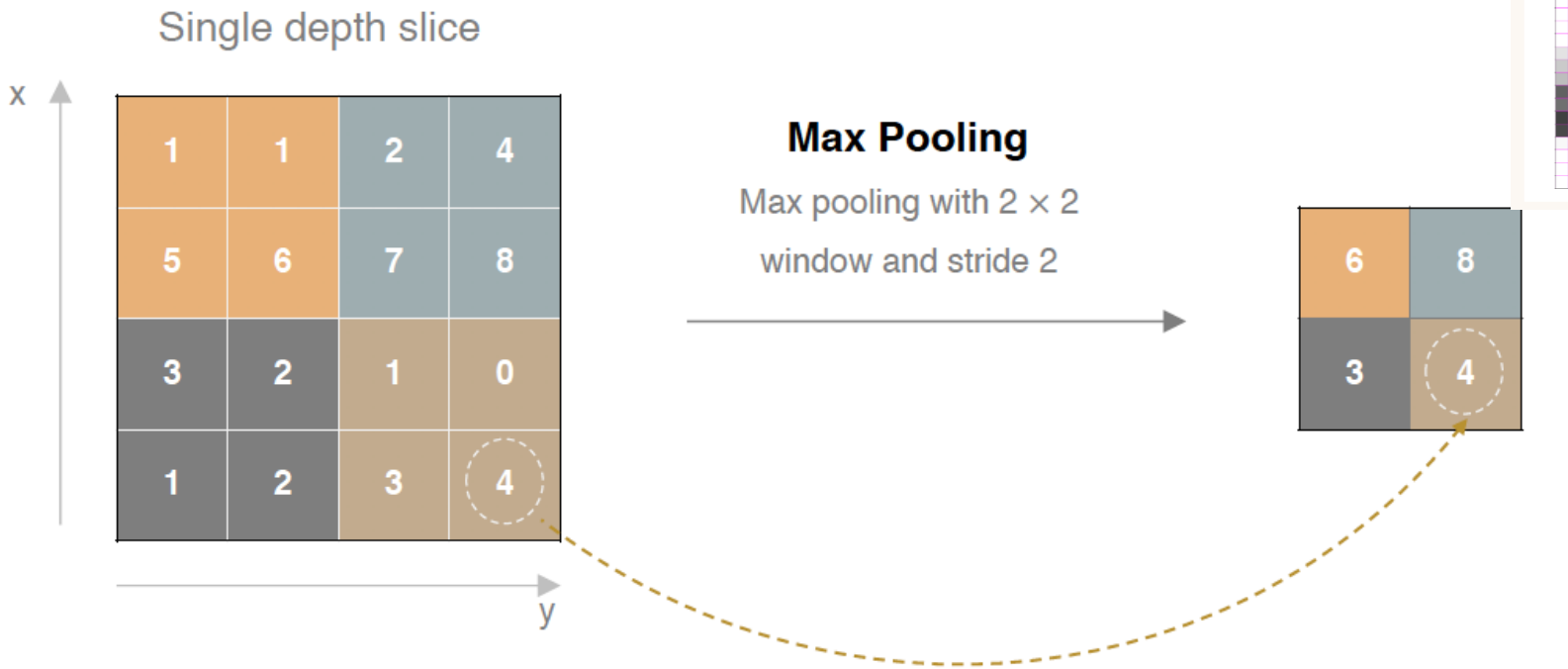
Pooling layer

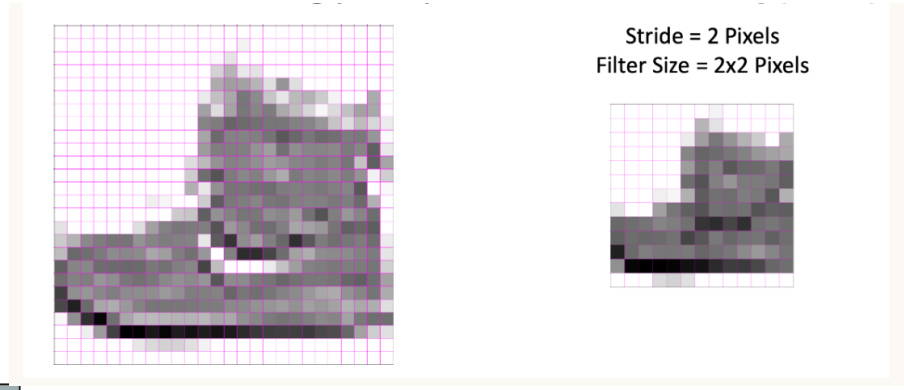
- Added between the convolutional layers.
- Reduce the dimensionality, the number of parameters and computation in the network, control overfitting.
- Increases robustness: Small changes in the input image do not change the pooled output much
- The most common type of pooling is max pooling which just takes the max value in the pooling window.



Max Pooling example

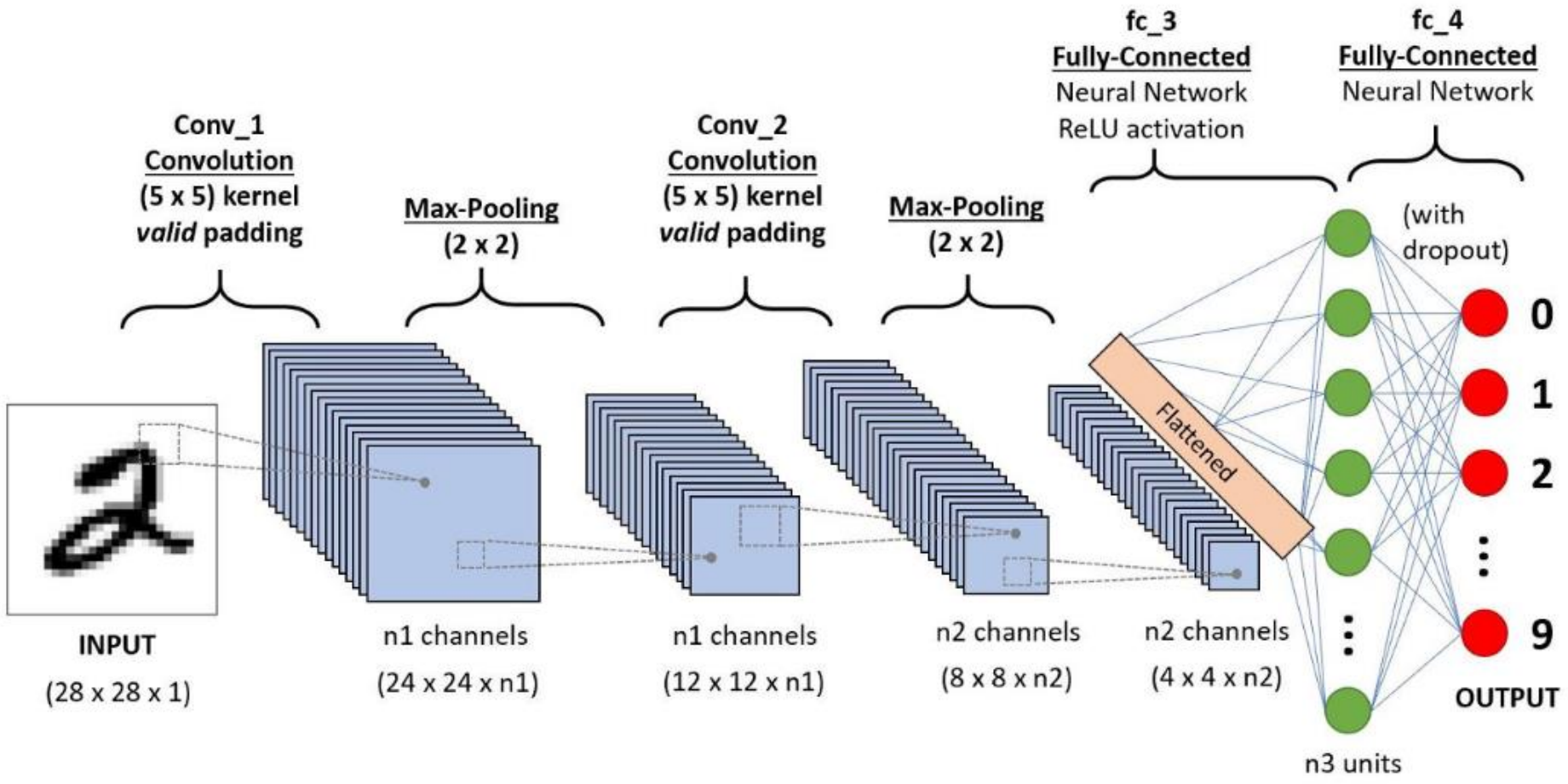
- Max pooling with 2 x 2 window.





- **In this example Max Pooling will halve the input in both dimensions.**
- By halving the height and the width, we reduced the number of weights to 1/4 of the input: millions of weights in CNN architectures, this reduction is significant.
- Improved statistical efficiency and reduced memory requirements for storing the parameters.

CNN architecture for MNIST digit classification



https://medium.com/@yash.4198/cnn-for-mnist-handwritten-dataset-cc94d61f6c94

CNN common architecture choices

- Convolutional layers, followed by ReLu, pooling layer, Convolutional with ReLu, Pooling…Fully Connected Layers
- Input layer: some typical values 32, 64, 96, 224, 384, 512 (divisible by 2)
- Convolutional layer:
  - Filter size: small, 3x3 or 5x5. Larger values 7x7 only in on the first convolutional layer that is looking at the input image
  - Number of filters (depth): depends, common practice to double after each pool layer
  - Filters not defined manually: learn during training (weights)
  - Stride small
- Pooling layer: max pooling 2x2
- Fully connected layer
  - Flatten
  - 1-2 dense layers
  - Dropout

- CNNs no longer state of the art for computer vision