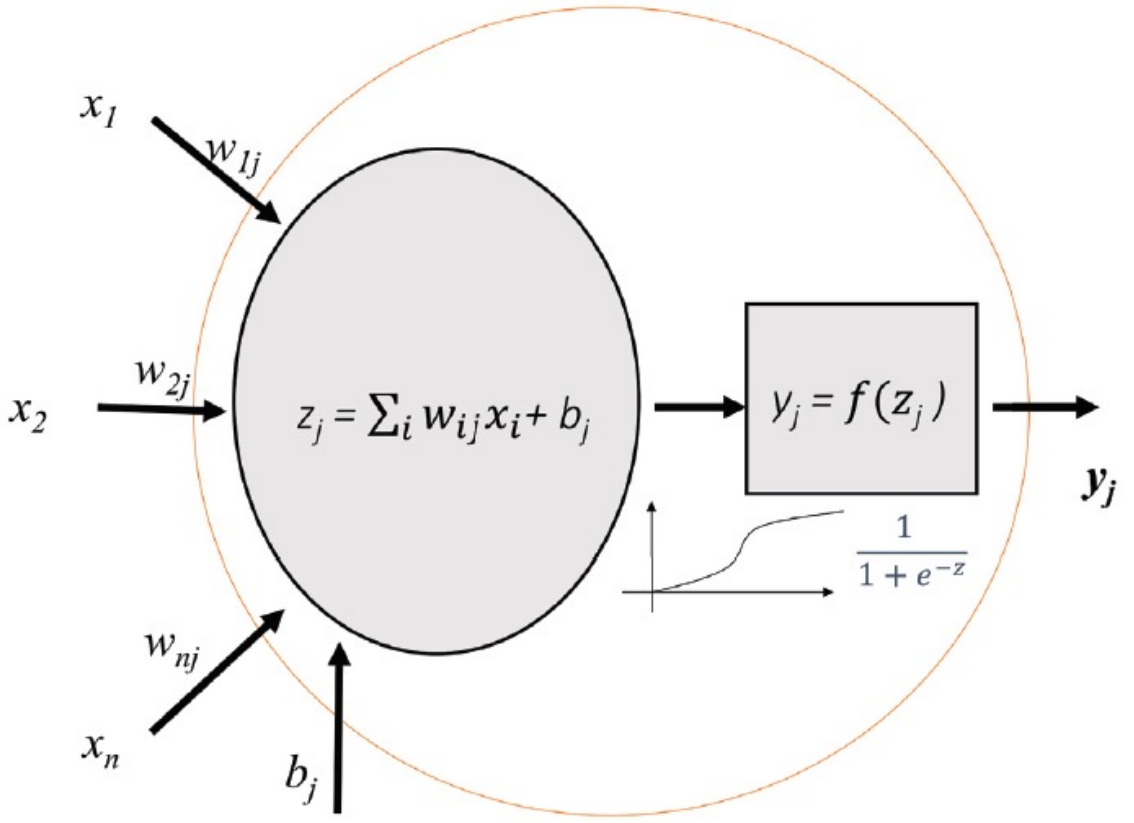
Some basics about the learning process

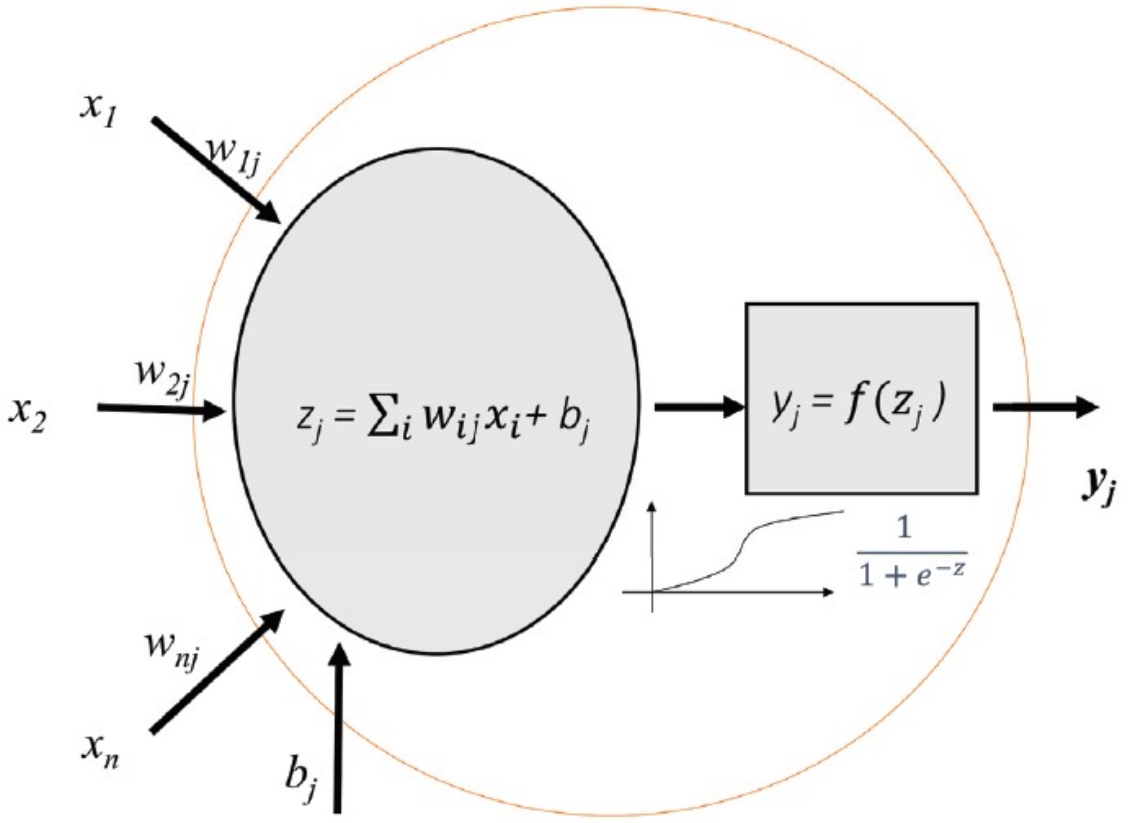
Jordi TORRES.AI

Neuron



A neural network is formed by neurons connected to each other; in turn, each connection of one neural network is associated with a weight that dictates the importance of this relationship in the neuron when multiplied by the input value.

Neuron



Each neuron has an activation function that defines the output of the neuron.

The activation function is used to introduce non-linearity in the network connections.

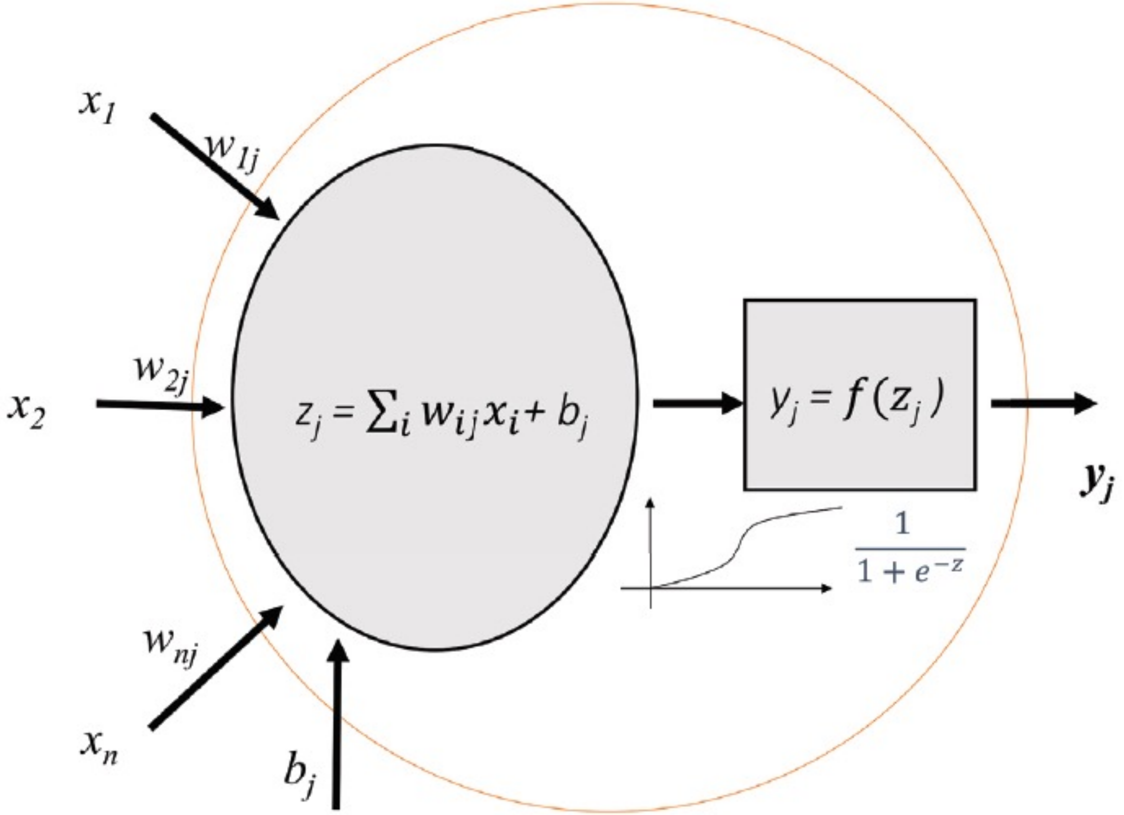
Learning process:

- **Adjust the biases and weights** of the connections between neurons

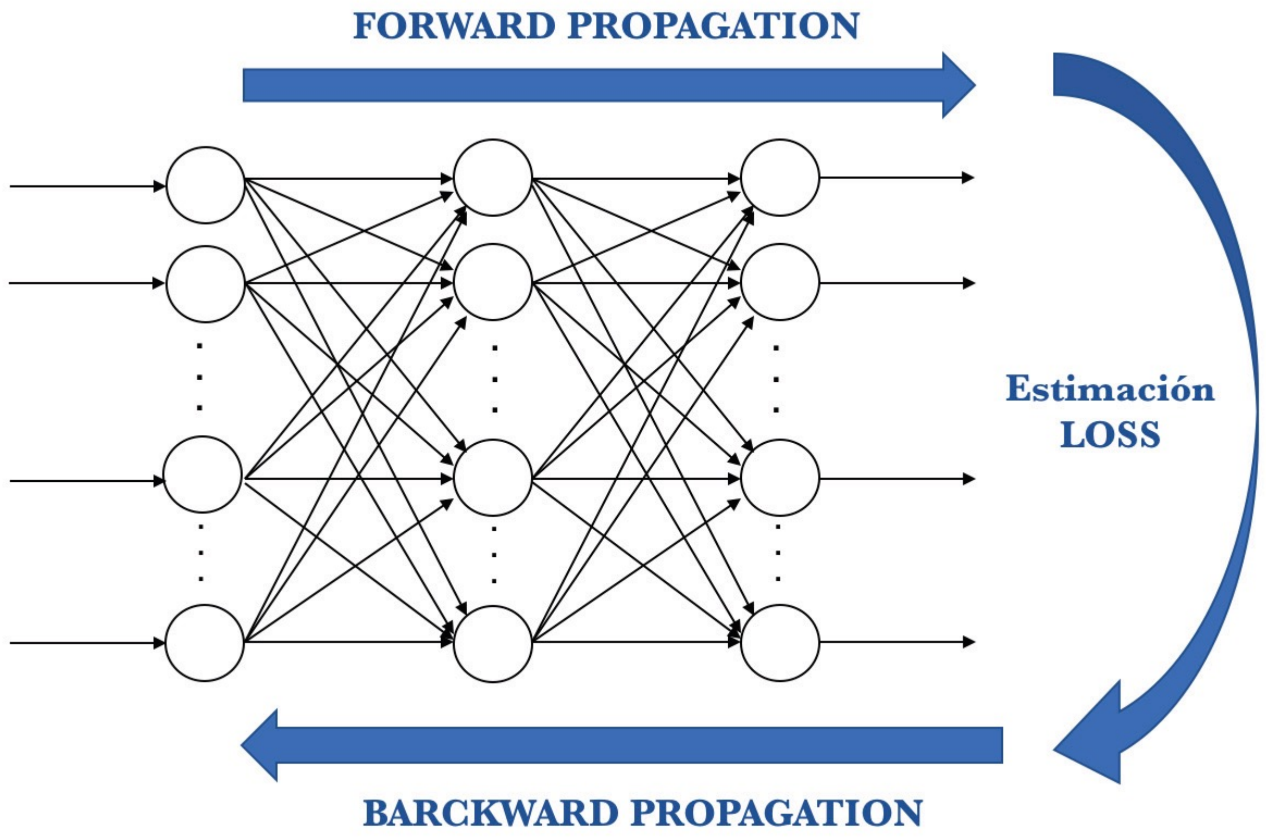
Objective:
Learn the values of the network parameters
(weights w_{ij} and biases b_j)

An iterative process of two phases:

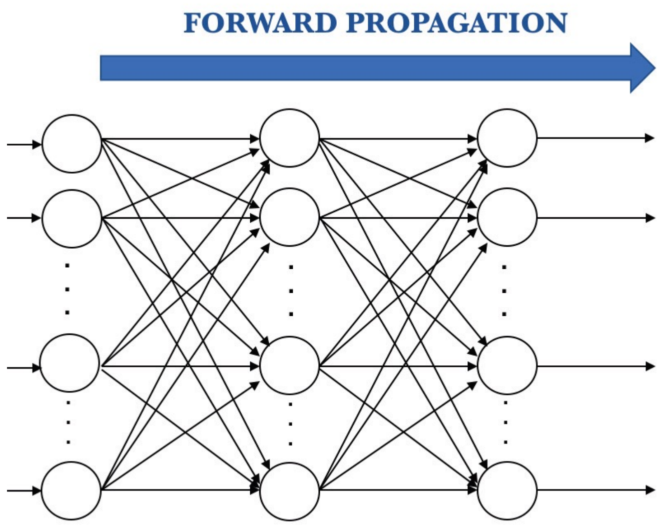
- Forwardpropagation.
- Backpropagation.



Learning process: overview

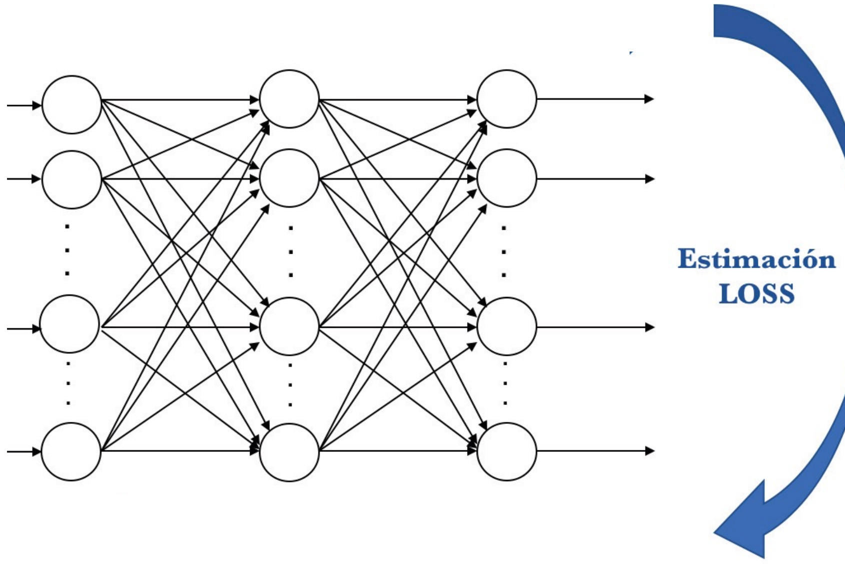


Learning process: Forwardpropagation



- The first phase, forwardpropagation, occurs when the training data crosses the entire neural network in order to calculate their predictions (labels).
  - That is, pass the input data through the network in such a way that all the neurons apply their transformation to the information they receive from the neurons of the previous layer and send it to the neurons of the next layer.
  - When data has crossed all the layers where all its neurons have made their calculations, the final layer will be reached, with a result of label prediction for those input examples.

Learning process: Loss function

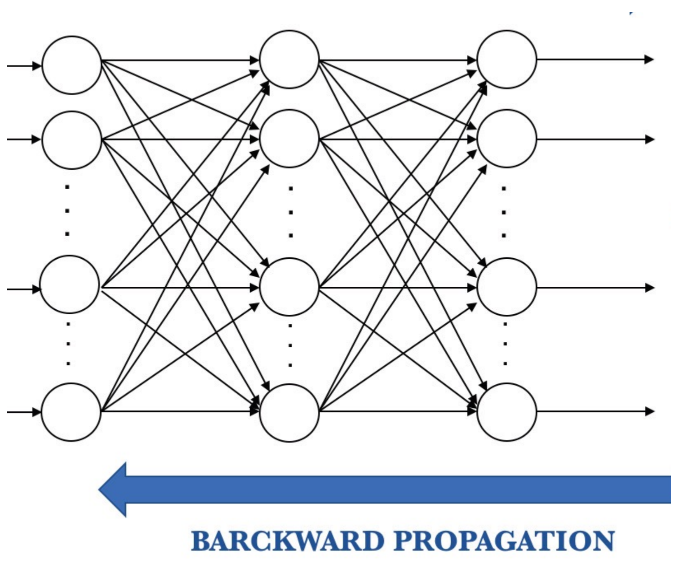


- Next, we will use a loss function to estimate the loss (or error) and to measure how good/bad our prediction result was in relation to the correct result.

(Remember that we are in a supervised learning environment and we have the label that tells us the expected value).

  - Ideally, we want our cost to be zero, that is, without divergence between estimated and expected value.
  - As the model is being trained, the weights of the interconnections of the neurons will be adjusted automatically until good predictions are obtained.

Learning process Backpropagation

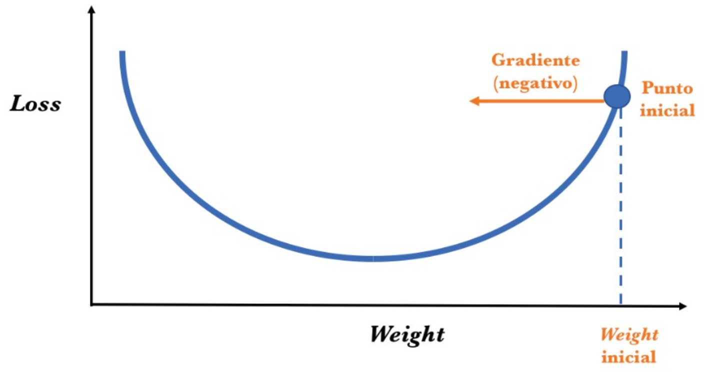


- Starting from the output layer, the calculated loss is propagated backward, to all the neurons in the hidden layer that contribute directly to the output.
  - However, the neurons of the hidden layer receive only a fraction of the total loss signal, based on the relative contribution that each neuron has contributed to the original output (indicated by the weight of the connexion).
  - This process is repeated, layer by layer, until all the neurons in the network have received a loss signal that describes their relative contribution to the total loss.

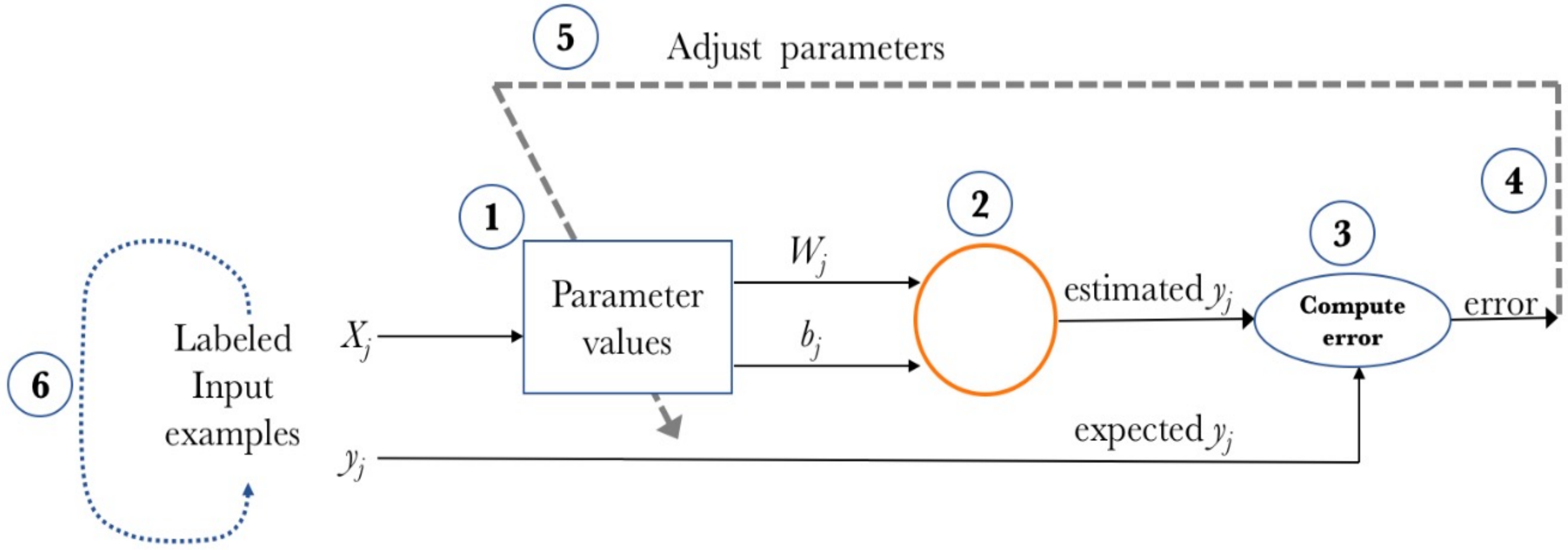
Adjust the weights: Gradient descent

- Once the loss information has been propagated to all the neurons, we proceed to adjust the weights.

- This technique changes the weights and biases in small increments, **with the help of the derivative (or gradient)** of the loss function, which allows us to see in which direction "to descend" towards the global minimum.
    - Generally, this is done in batches of data in the successive iterations of the set of all the data that we pass to the network in each iteration.



Suplementary: Learning algorithm overview

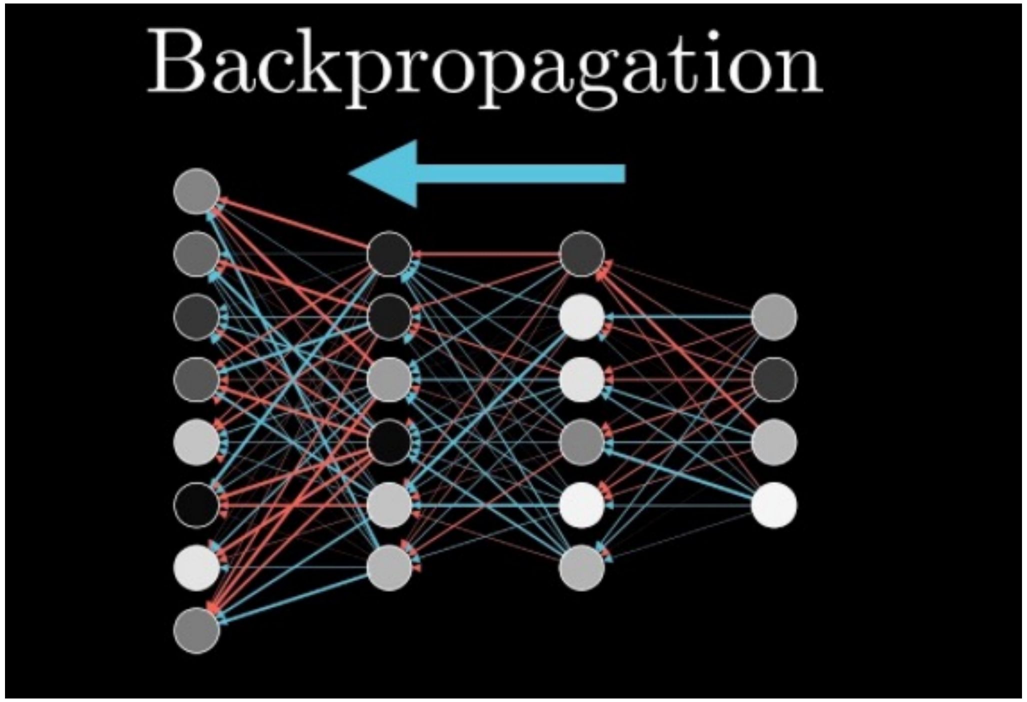


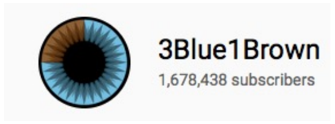
Suplementary: Learning algorithm overview

1. **Start with some values (often random) for the network weights and biases.**
2. **Take a set of input examples and perform the forwardpropagation process passing them through the network to get their prediction.**
3. **Compare these obtained predictions with the expected labels and calculate the loss using the loss fuction.**
4. **Perform the backpropagation process to propagate this loss, to each and every one of the parameters, that make up the model of the neural network.**
5. **Use this propagated information to update the parameters of the neural network with the gradient descent. This reduces the total loss and obtains a better model.**
6. **Continue iterating on steps 2 to 5 until we consider that we have a good model (we will see later when we should stop).**

Suplementary

Watch the video:





https://www.youtube.com/watch?v=Ilg3gGewQ5U&list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi&index=4&t=25s

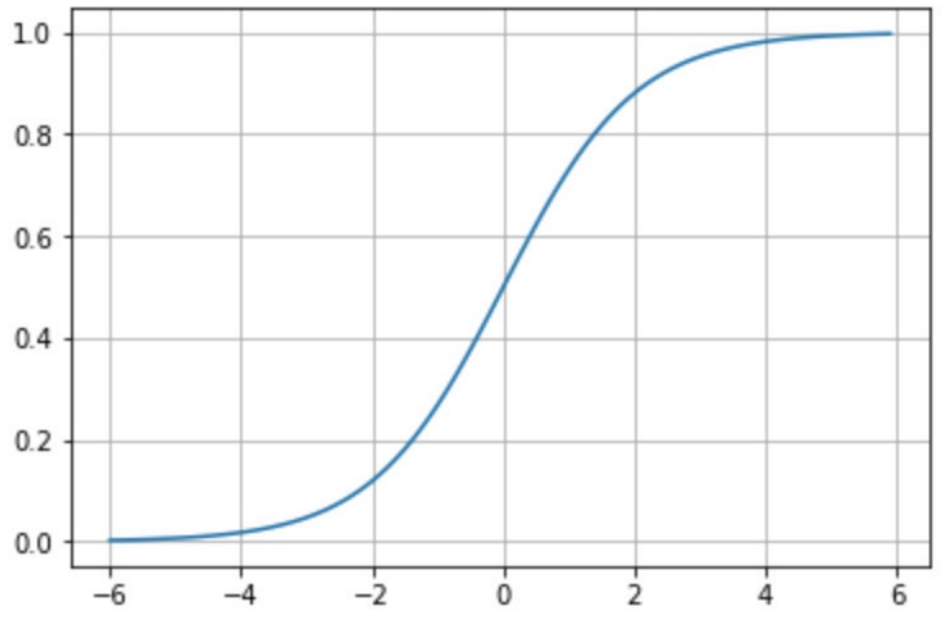
Activation Function

Jordi TORRES.AI

Activation Function: Sigmoid

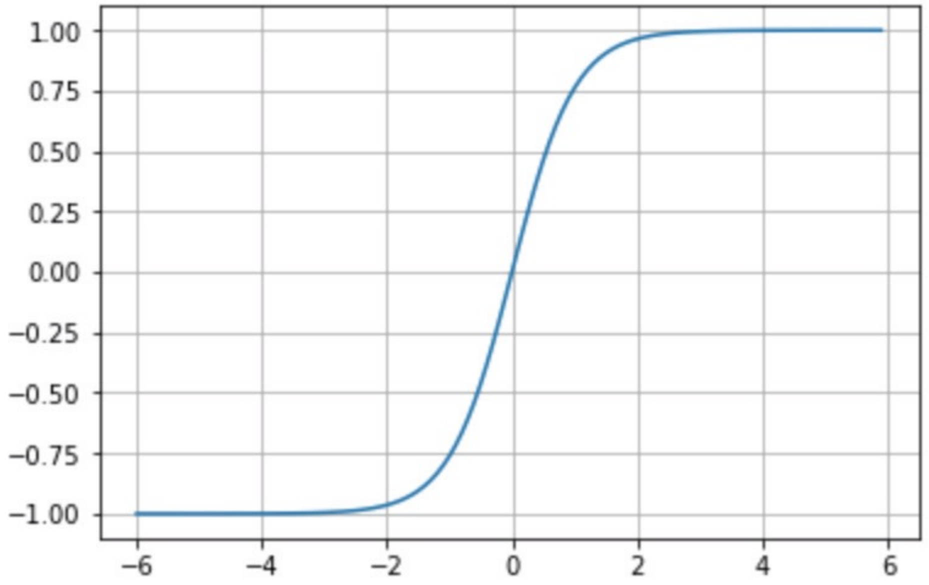
The sigmoid function was introduced previously and allows us to reduce extreme or atypical values without eliminating them.

A sigmoid function converts independent variables of almost infinite range into simple probabilities between 0 and 1. Most of the output will be very close to the extremes of 0 or 1, as we have already commented.



Activation Function: Tanh

In the same way that the tangent represents a relation between the opposite side and the adjacent side of a right-angled triangle, tanh represents the relation between the hyperbolic sine and the hyperbolic cosine: tanh(x)=sinh (x)/cosh(x).

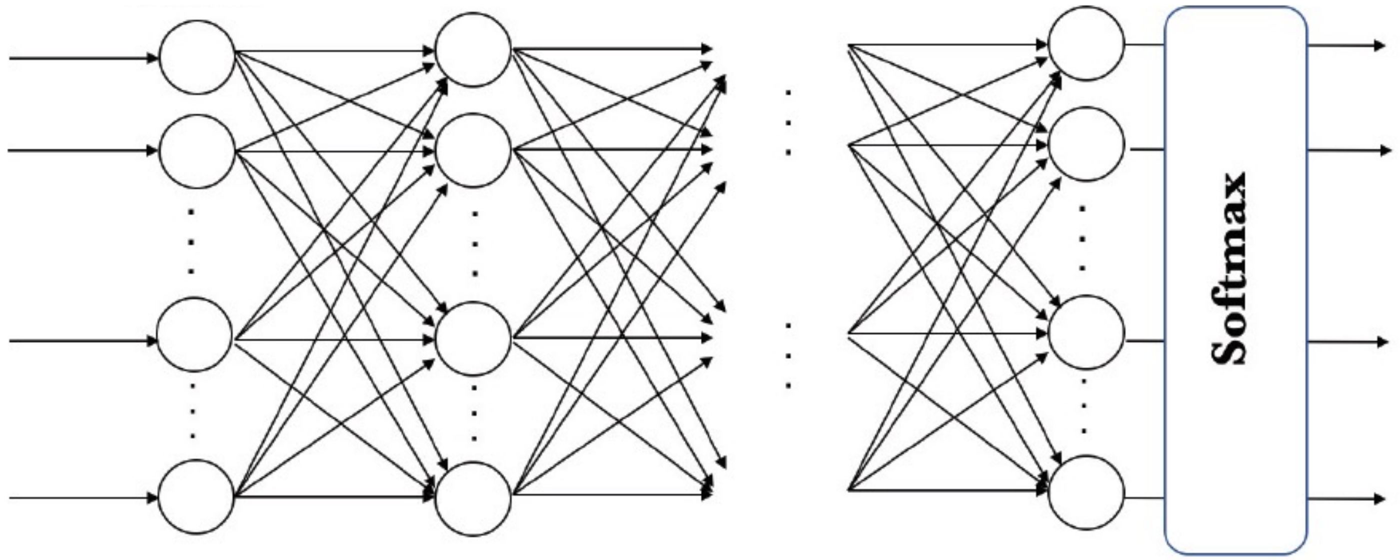


Unlike the sigmoid function, the normalized range of tanh is between -1 to 1, which is the input that suits some neural networks.

The advantage of tanh is that you can also deal more easily with negative numbers.

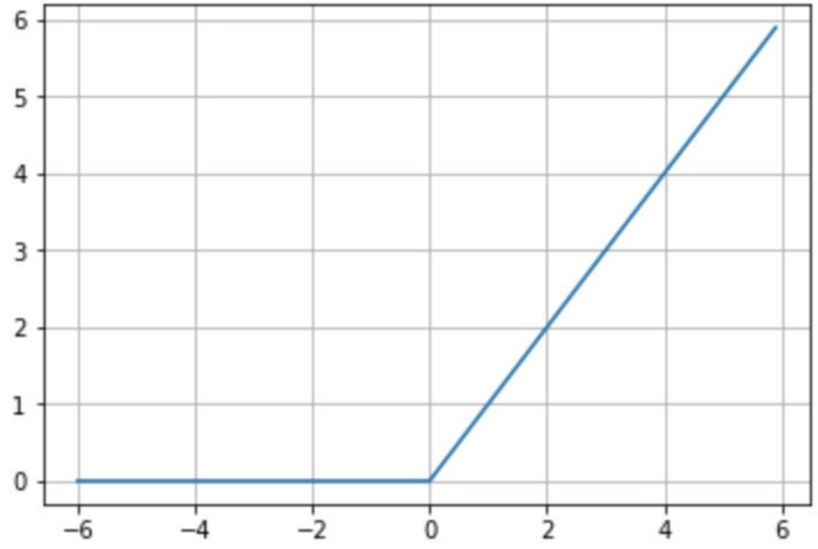
Activation Function: Softmax

- The Softmax activation function was already presented as a way to generalize the logistic regression.
- The softmax activation function returns the probability distribution on mutually exclusive output classes.
- Softmax will often be found in the output layer of a classifier.



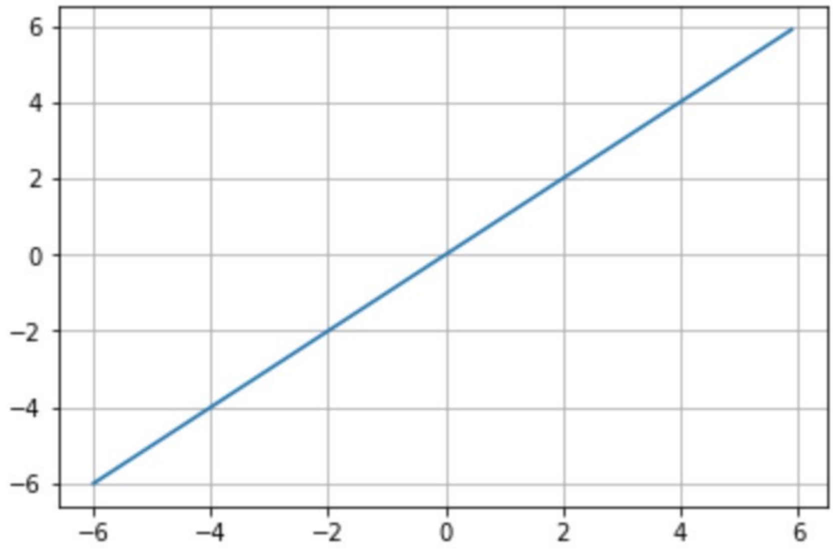
Activation Function: ReLU

- The activation function Rectified linear unit (ReLU) is a very interesting transformation that activates a single node if the input is above a certain threshold.
  - While the input has a value below zero, the output will be zero, but when the input rises, the output is a linear relationship with the input variable of the form f (x)=max (0,x) .
  - The ReLUs activation function **has proven to be the best option in many different cases.**



Activation Function: Linear

- The linear activation function is basically the identity function, which in practical terms, means that the signal does not change. Sometimes, we can see this activation function in the input layer of a neural network.



Learning process configuration

Jordi TORRES.AI

Backpropagation

In summary, we can see backpropagation as a method to alter the weights and biases of the neural network in the right direction.

> Start first by calculating the loss term and then, the weights+biases are adjusted in reverse order, with an optimization algorithm, taking into account this calculated loss.

In Keras the *compile()* method allows us to specify the components involved in the learning process:

```
model.compile(loss='categorical_crossentropy',
              optimizer='sgd',
              metrics=['accuracy'])
```

Arguments of *compile()* method: loss

- A loss function is one of the parameters required to quantify how close a particular neural network is to our ideal, during the learning process.
    - In the Keras manual, we can find all types of loss functions available.

  https://keras.io/losses/

- The choice of the best function of loss lies in understanding what type of error is or is not acceptable for the particular problema.

```
model.compile(loss='categorical_crossentropy',
              optimizer='sgd',
              metrics=['accuracy'])
```

Available losses

Note that all losses are available both via a class handle and via a function handle. The class handles enable you to pass configuration arguments to the constructor (e.g. `loss_fn = CategoricalCrossentropy(from_logits=True)`), and they perform reduction by default when used in a standalone way (see details below).

Probabilistic losses

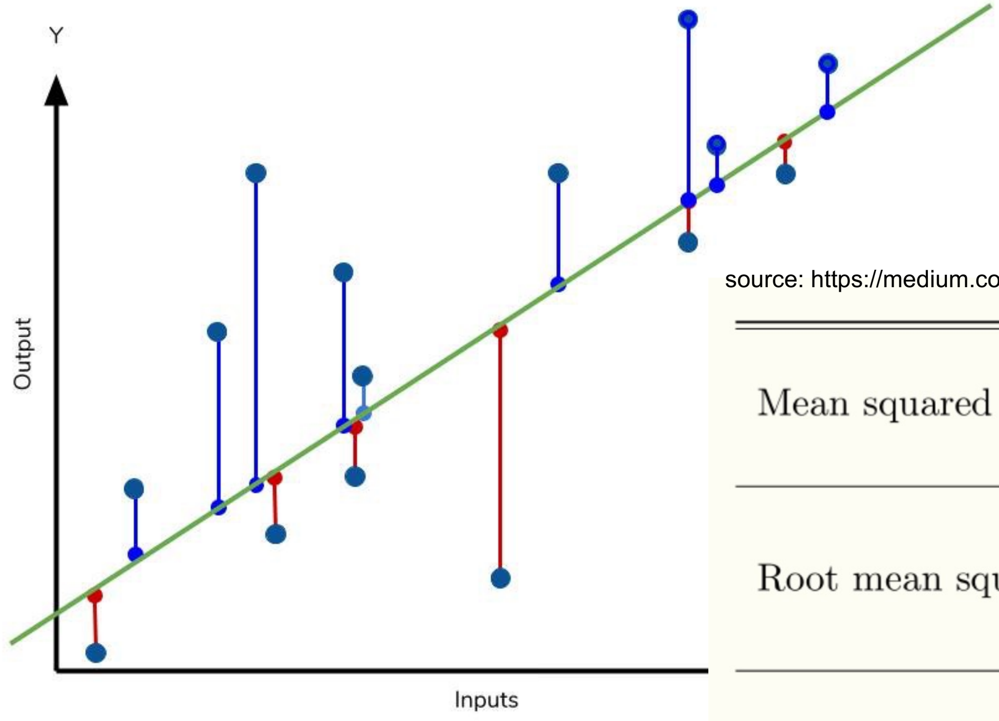
- BinaryCrossentropy class
- CategoricalCrossentropy class
- SparseCategoricalCrossentropy class
- Poisson class
- binary_crossentropy function
- categorical_crossentropy function
- sparse_categorical_crossentropy function
- poisson function
- KLDivergence class
- kl_divergence function

Regression losses

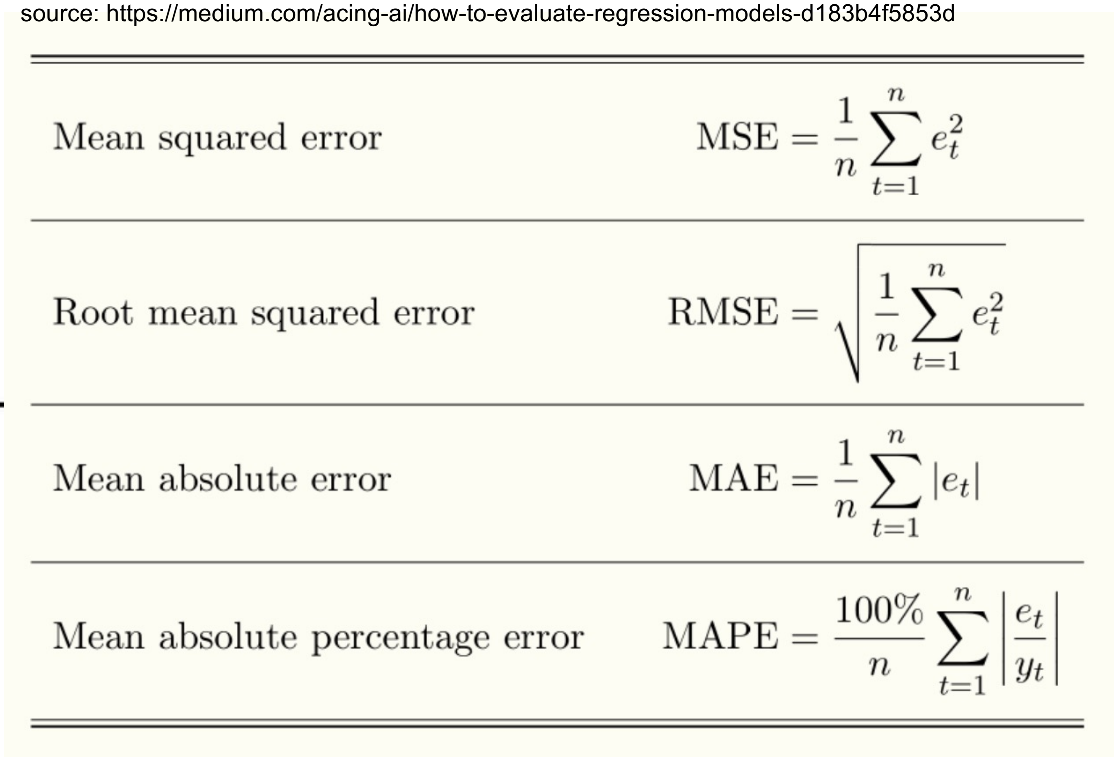
- MeanSquaredError class
- MeanAbsoluteError class
- MeanAbsolutePercentageError class
- MeanSquaredLogarithmicError class
- CosineSimilarity class
- mean_squared_error function
- mean_absolute_error function
- mean_absolute_percentage_error function
- mean_squared_logarithmic_error function
- cosine_similarity function
- Huber class
- huber function
- LogCosh class
- log_cosh function

Hinge losses for "maximum-margin" classification

- Hinge class
- SquaredHinge class
- CategoricalHinge class
- hinge function
- squared_hinge function
- categorical_hinge function



source: https://medium.com/acing-ai/how-to-evaluate-regression-models-d183b4f5853d



Mean squared error	$\text{MSE} = \frac{1}{n}\sum_{t=1}^{n} e_t^2$
Root mean squared error	$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{t=1}^{n} e_t^2}$
Mean absolute error	$\text{MAE} = \frac{1}{n}\sum_{t=1}^{n} \lvert e_t \rvert$
Mean absolute percentage error	$\text{MAPE} = \frac{100\%}{n}\sum_{t=1}^{n} \left\lvert \frac{e_t}{y_t} \right\rvert$

Jordi TORRES.AI

Arguments of *compile()* method: optimizers

- In general, we can see the learning process as a global optimization problem where the parameters (weights and biases) must be adjusted in such a way that the loss function presented above is minimized.
  - In most cases, this optimization cannot be solved analytically, but in general, it can be approached effectively with iterative optimization algorithms or optimizers.
  - Keras currently has different optimizers that can be used:
    - SGD, RMSprop, Adagrad, Adadelta, Adam, Adamax, Nadam.

  → https://keras.io/optimizers/

Stochastic Gradient Descent (SGD)

We have not yet talked about how often the values of the parameters are adjusted.

- **Stochastic gradient descent:** After each entry example (**online learning)**
- **Batch gradient descent:** After each iteration on the whole set of training examples (**batch learning)**

(*) The literature indicates that we usually get better results with online learning, but there are reasons that justify batch learning because many optimization techniques will only work with it.

- For this reason most applications of SGD actually use a minibatch of several samples (**mini-batch learning)** : The values of the parameters are adjusted after a sample of examples of the training set.

SGD in Keras/TensorFlow

- We divide the data into several batches.
- Then we take the first batch, calculate the gradient of its loss and update the parameters; this would follow successively until the last batch.
- Now, in a single pass through all the input data, only a number of steps have been made, equal to the number of batches.

```
model.compile(loss='categorical_crossentropy',
              optimizer='sgd',
              metrics=['accuracy'])
```

```
model.fit(X_train, y_train, epochs=10, batch_size=100)
```

Hyperparameters

Jordi TORRES.AI

Parameters vs hyperparameters

- **Parameter:** A variable of a model that the **DL system trains on its own**. For example, weights are parameters whose values the DL system gradually learns through successive training iterations.

- **Hyperparameters:** The **"knobs"** that you tweak during successive runs of training a model.

Many hyperparameters:

- **Epochs**
- **Batch size**
- Learning rate
- Learning rate decay
- Momentum
- . . .

- Number of layers
- Number of neurons per layer
- . . .

Epochs

- An epoch is **defined as a single training iteration of all batches** in both forward and back propagation. This means 1 epoch is a single forward and backward pass of the entire input dataset.
- The number of epochs you would use to train your network can be chosen by you.
  - It’s highly likely that more epochs would show higher accuracy of the network, however, it would also take longer for the network to converge.
  - Also you must be aware that if the number of epochs is too high, the network might be overfitted.

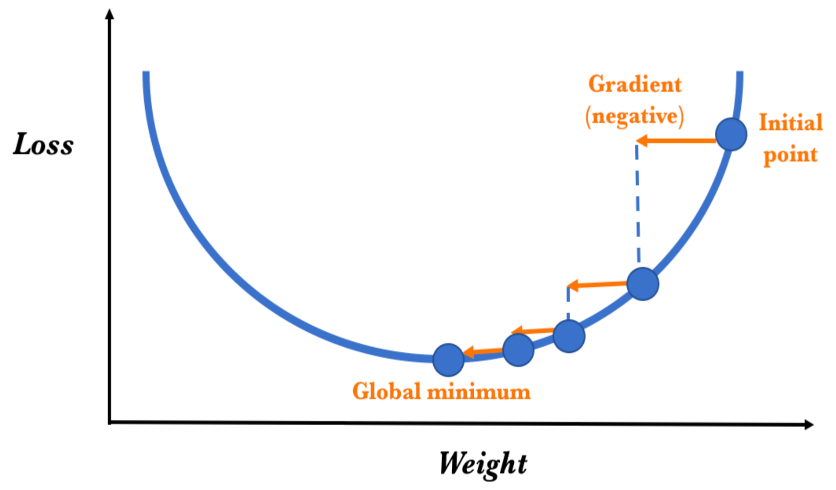
Batch size

- The number of examples in a batch. The set of examples used in one single update of a model's weights during training.
- Indicated in the `fit()` method
- The optimal size will depend on many factors, including the memory capacity of the computer that we use to do the calculations.

Learning rate

- A scalar used to train a model via gradient descent. During each iteration, the gradient descent algorithm multiplies the learning rate by the gradient.

- In simple terms, the rate at which we descend towards the minima of the cost function is the learning rate. We should choose the learning rate very carefully since it should be neither so large that the optimal solution is missed and nor so low that it takes forever for the network to converge.



Getting started with CNN

Relevant Deep Learning Models

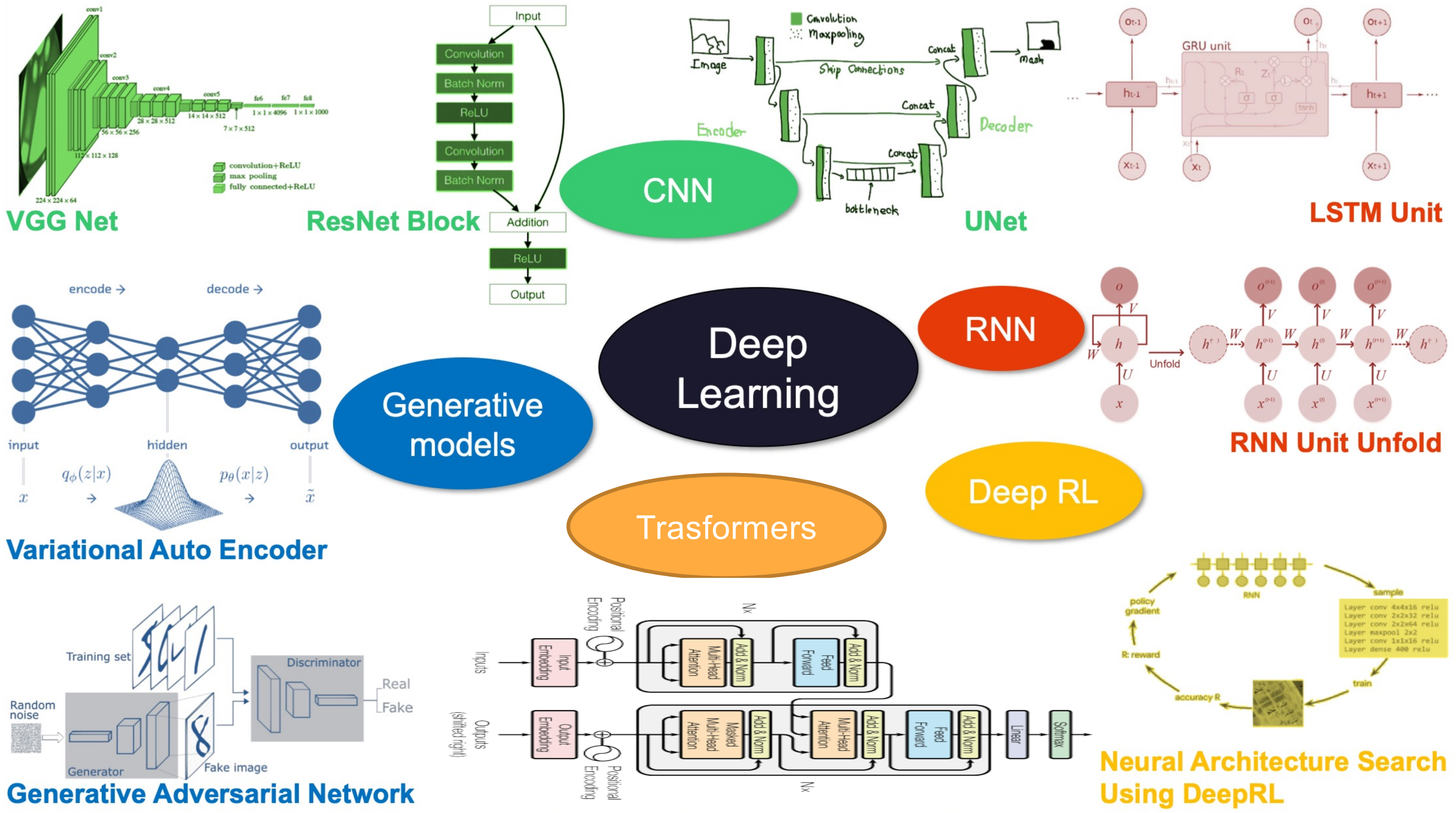


Image source:
https://arxiv.org/pdf/2006.10027v1.pdf

Convolutional Neural Networks

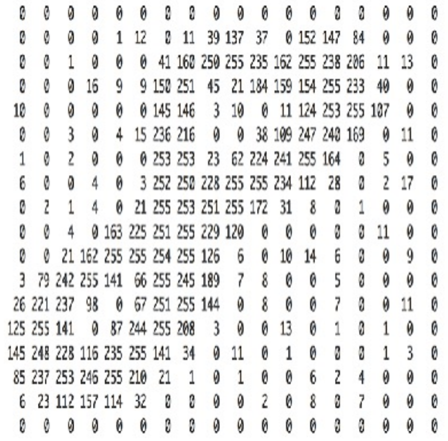
- Acronyms: CNNs or ConvNets

- A category of Neural Networks that have proven very effective in areas such as image recognition and classification.

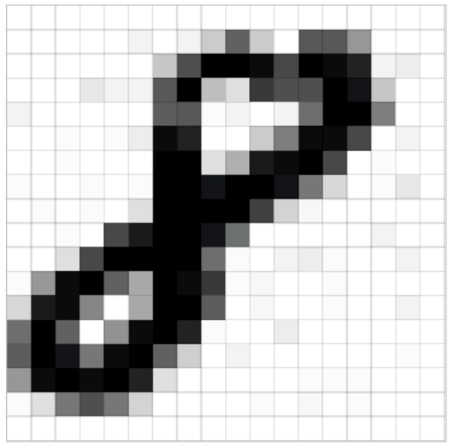
- Popularity: ConvNets have been successful in identifying faces, objects and traffic signs apart from powering vision in robots and self driving cars.

Image: matrix of pixel values

- Un CNN: An explicit assumption that the inputs are images.



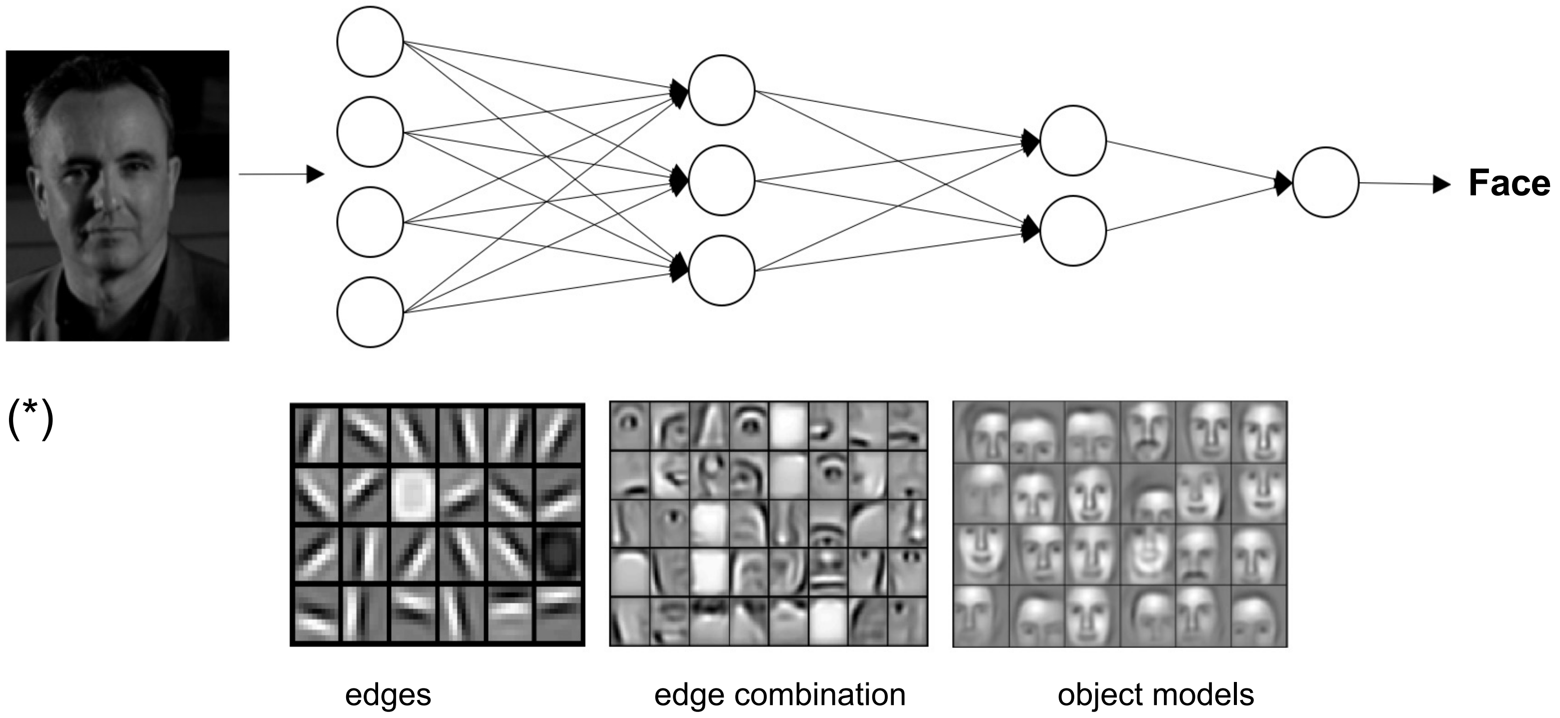
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	12	0	11	39	137	37	0	152	147	84	0	0	0
0	0	1	0	0	0	41	160	250	255	235	162	255	238	206	11	13	0
0	0	0	16	9	9	150	251	45	21	184	159	154	255	233	40	0	0
10	0	0	0	0	0	145	146	3	10	0	11	124	253	255	107	0	0
0	0	3	0	4	15	236	216	0	0	38	109	247	240	169	0	11	0
1	0	2	0	0	0	253	253	23	62	224	241	255	164	0	5	0	0
6	0	0	4	0	3	252	250	228	255	255	234	112	28	0	2	17	0
0	2	1	4	0	21	255	253	251	255	172	31	8	0	1	0	0	0
0	0	4	0	163	225	251	255	229	120	0	0	0	0	0	11	0	0
0	0	21	162	255	255	254	255	126	6	0	10	14	6	0	0	9	0
3	79	242	255	141	66	255	245	189	7	8	0	0	5	0	0	0	0
26	221	237	98	0	67	251	255	144	0	8	0	0	7	0	0	11	0
125	255	141	0	87	244	255	208	3	0	0	13	0	1	0	1	0	0
145	248	228	116	235	255	141	34	0	11	0	1	0	0	0	1	3	0
85	237	253	246	255	210	21	1	0	1	0	0	6	2	4	0	0	0
6	23	112	157	114	32	0	0	0	0	2	0	8	0	7	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0



- **Channel**
    - is a conventional term used to refer to a certain component of an image.
    - For an RGB color image → 3 channels

Convolutional Neural Networks

- Intuitive Explanation of CNN.



Basic components of a CNN

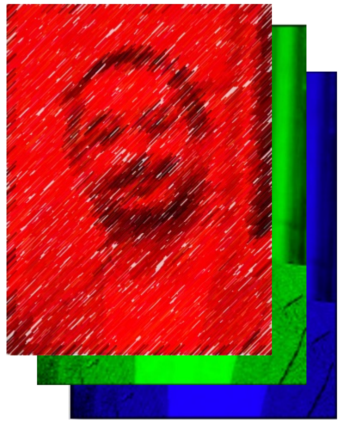
- **The convolution operation**
    - Intuitively, we could say that the main purpose of a convolutional layer is to detect features or visual features in images.
    - Convolutional layers can learn spatial hierarchies of patterns by preserving spatial relationships.
- **The pooling operation**
    - Accompanies the convolutional layers.
    - Simplifies the information collected by the convolutional layer and create a condensed version of the information contained in them.
- **Classification (Fully Connected Layer)**

Basic components of a CNN

- **The convolution operation**
    - Intuitively, we could say that the main purpose of a convolutional layer is to detect features or visual features in images.
    - Convolutional layers can learn spatial hierarchies of patterns by preserving spatial relationships.
- **The pooling operation**
    - Accompanies the convolutional layers.
    - Simplifies the information collected by the convolutional layer and create a condensed version of the information contained in them.
- **Classification (Fully Connected Layer)**

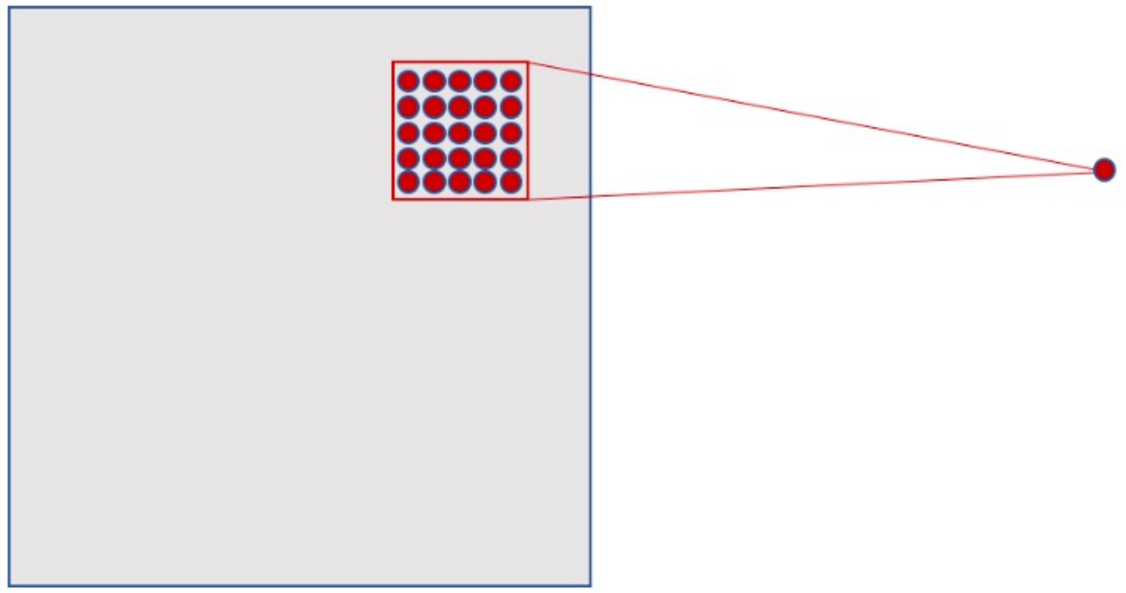
The convolution operation

- In general, the convolution layers operate on 3D tensors, called feature maps, with two spatial axes of height and width, as well as a channel axis (also called depth).
    - For an RGB color image, the dimension of the depth axis is 3, since the image has three channels: red, green and blue.
    - For a black and white image, such as the MNIST, the depth axis dimension is 1 (grey level).

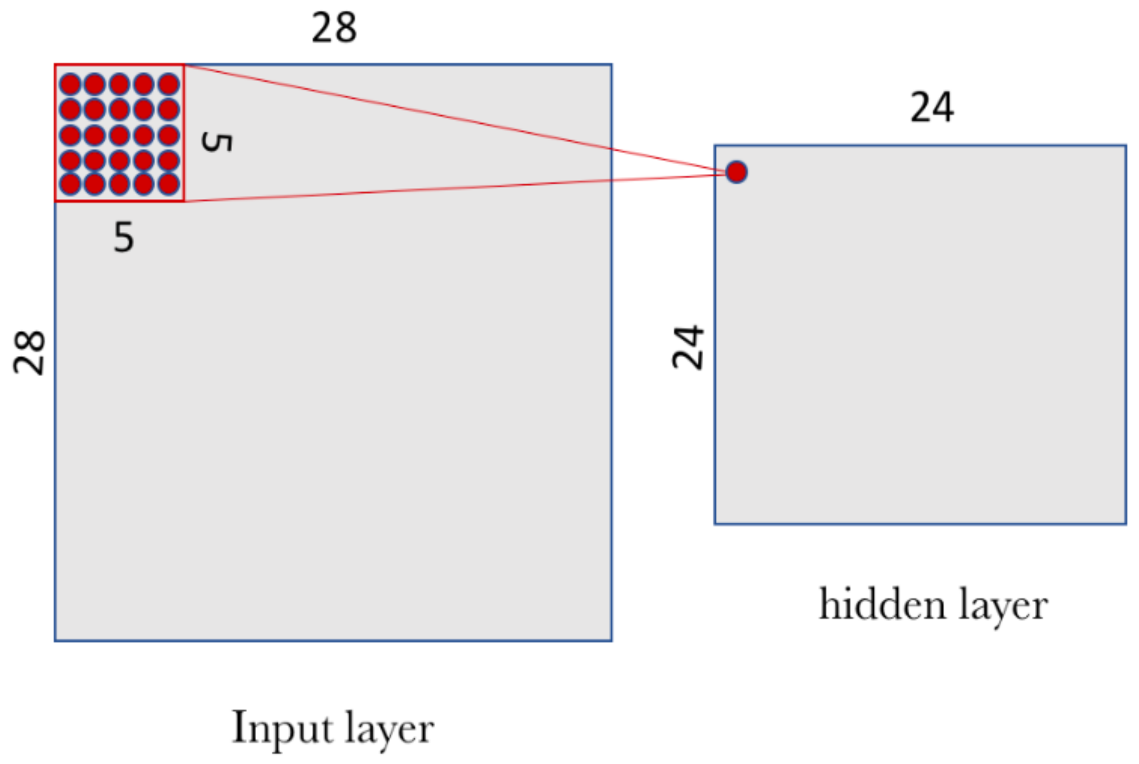


The convolution operation

- In CNN not all the neurons of a layer are connected with all the neurons of the next layer as in the case of fully connected neural networks; it is done by using only small localized areas of the space of input neurons.

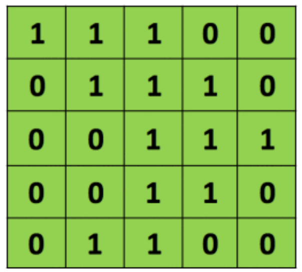


The convolution operation

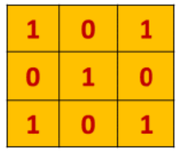


- Sliding window
- Use the same filter (the same W matrix of weights and the same bias b) for all the neurons in the next layer

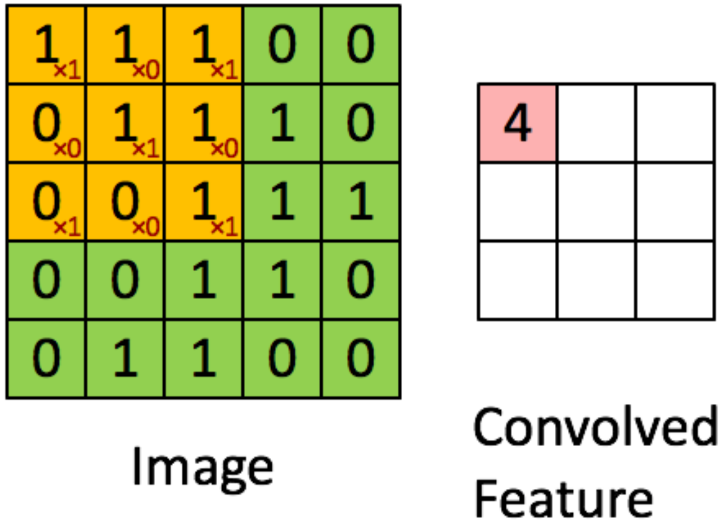
The convolution operation visual example



1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0



1	0	1
0	1	0
1	0	1



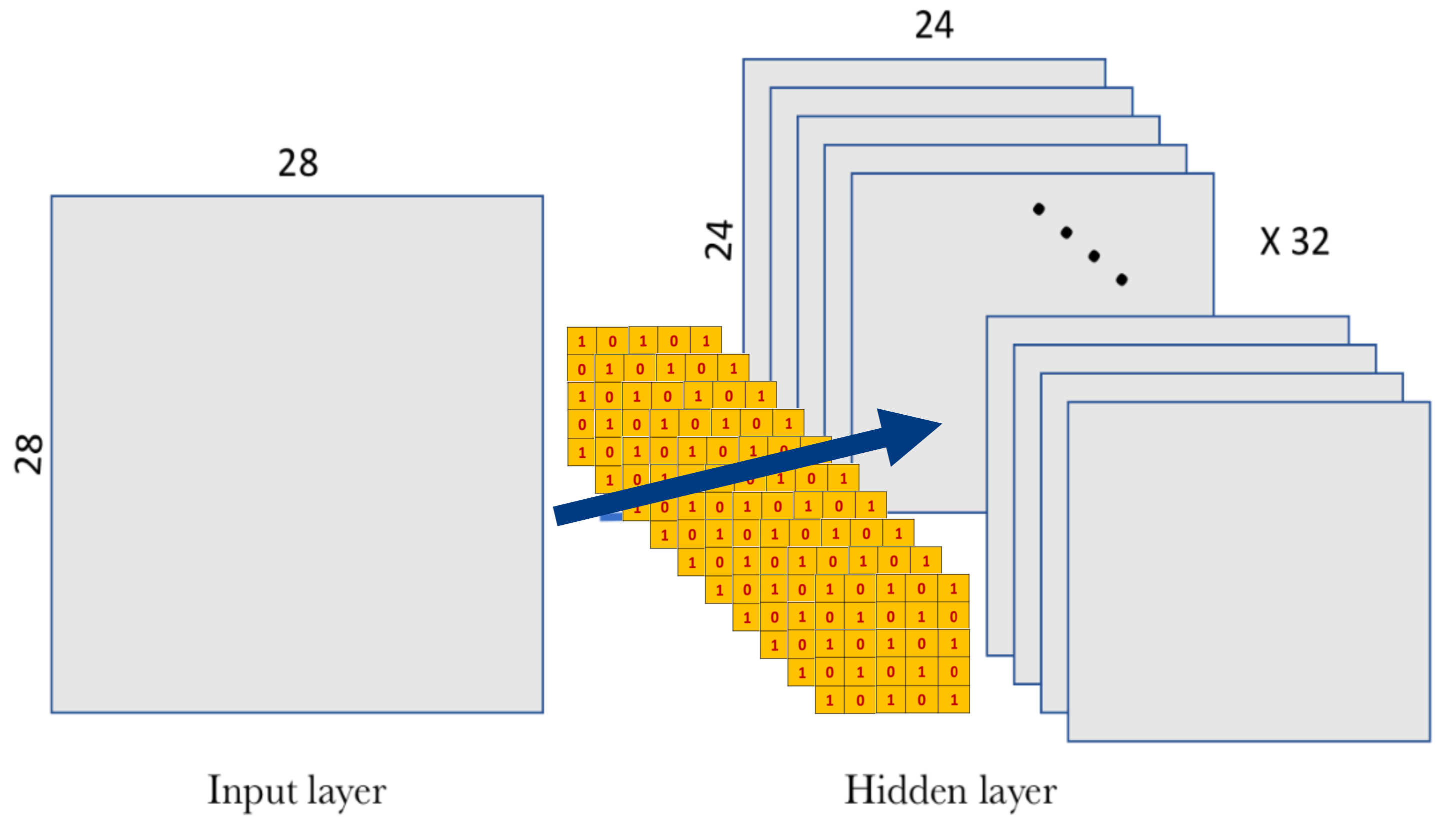
Source: http://deeplearning.stanford.edu/wiki/index.php/Feature_extraction_using_convolution

The convolution operation

- In CNN terminology, the 3×3 matrix is called a '**filter**' or 'kernel' or 'feature detector'
- and the matrix formed by sliding the filter over the image and computing the dot product is called the 'Convolved Feature' or 'Activation Map' or the '**Feature Map**'
- It is important to note that filters acts as feature detectors from the original input image.

The convolution operation

- Many filters (one for each feature that we want to detect)



The convolution operation

- In practice, a CNN *learns* the values of these filters on its own during the training process

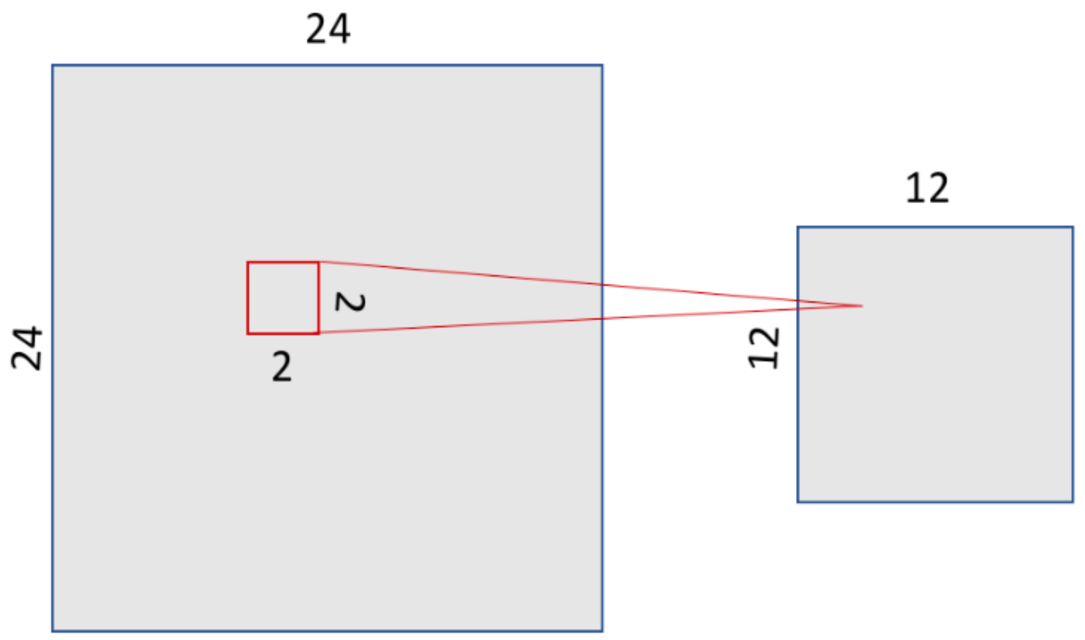
- Reminder: we still need to specify hyperparameters such as
  - number of filters,
  - filter size,
  - architecture of the network
  - etc.

Basic components of a CNN

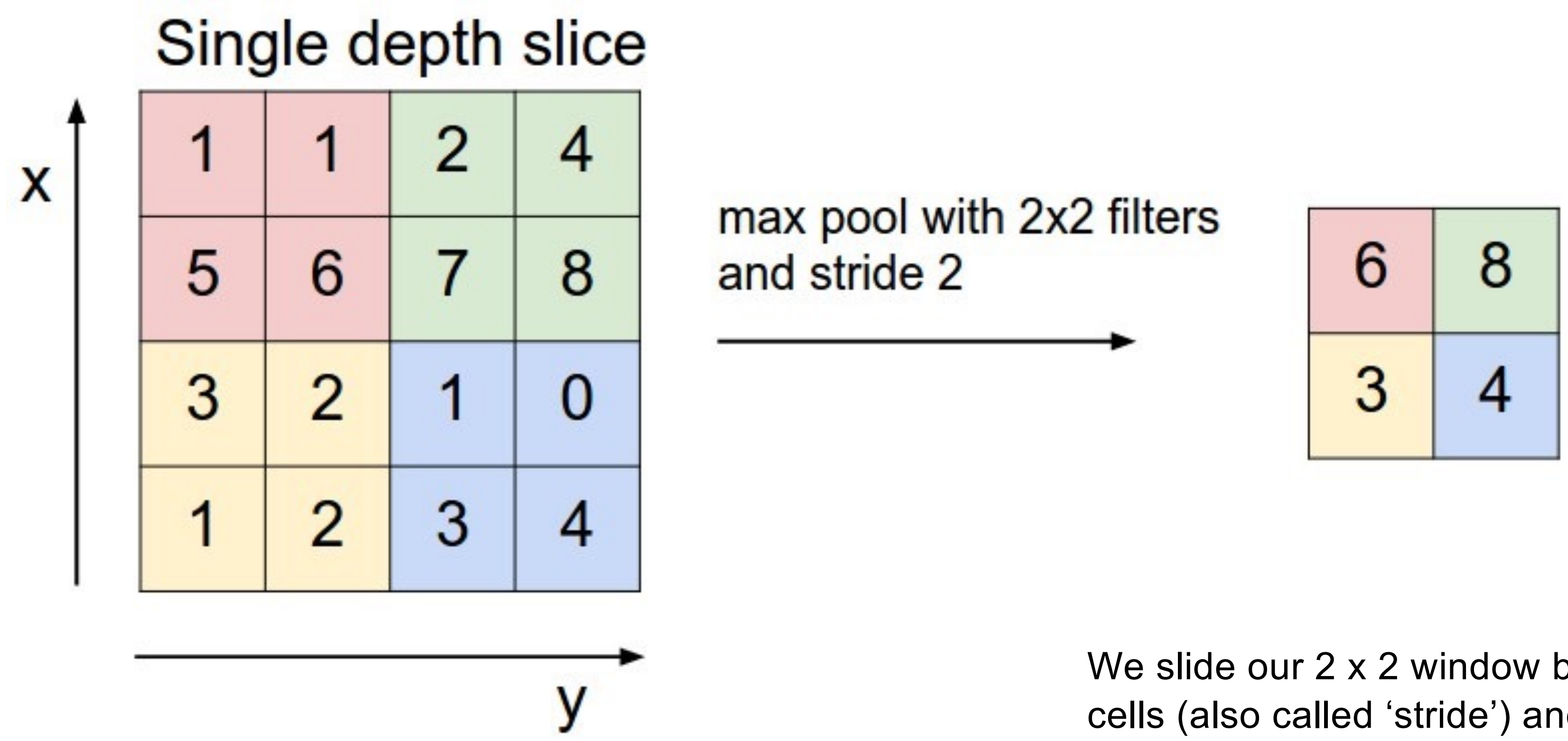
- **The convolution operation**
    - Intuitively, we could say that the main purpose of a convolutional layer is to detect features or visual features in images.
    - Convolutional layers can learn spatial hierarchies of patterns by preserving spatial relationships.
- **The pooling operation**
    - Accompanies the convolutional layers.
    - Simplifies the information collected by the convolutional layer and create a condensed version of the information contained in them.
- **Classification (Fully Connected Layer)**

The pooling operation

- Accompanies the convolution layer
- Simplifies the information collected by the convolutional layer and creates a condensed version of the information:
    - max-pooling
    - average-pooling



The pooling operation

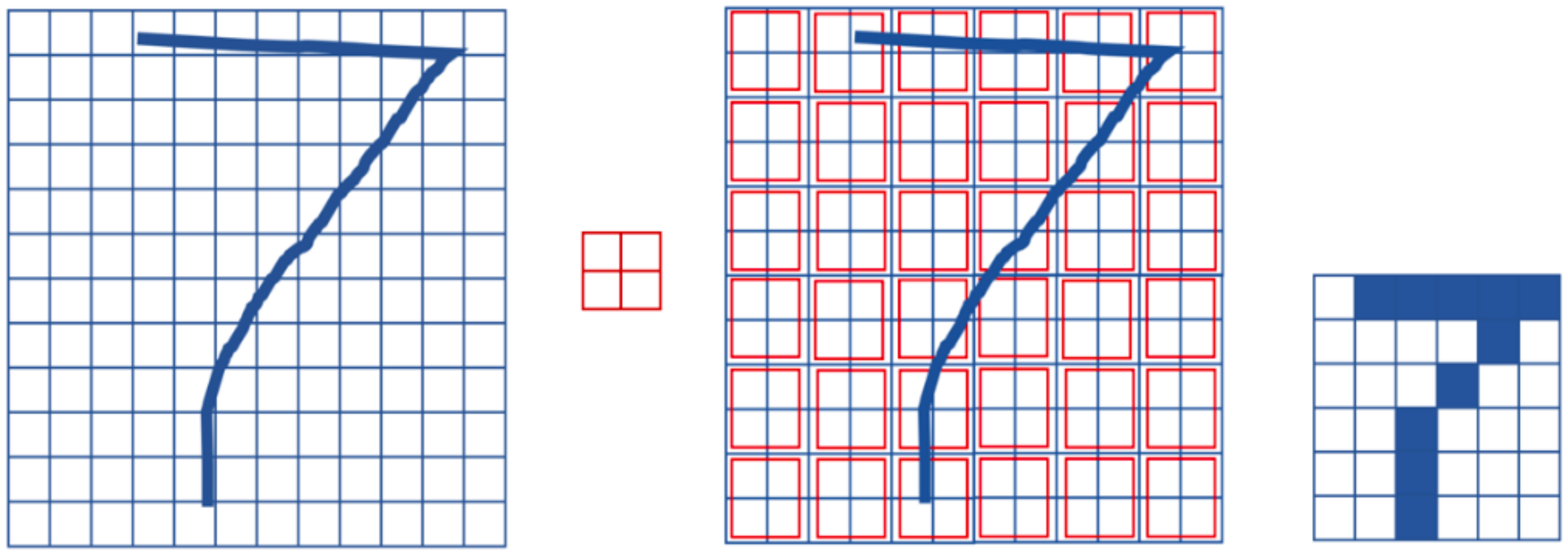


Source: http://cs231n.github.io/convolutional-networks/

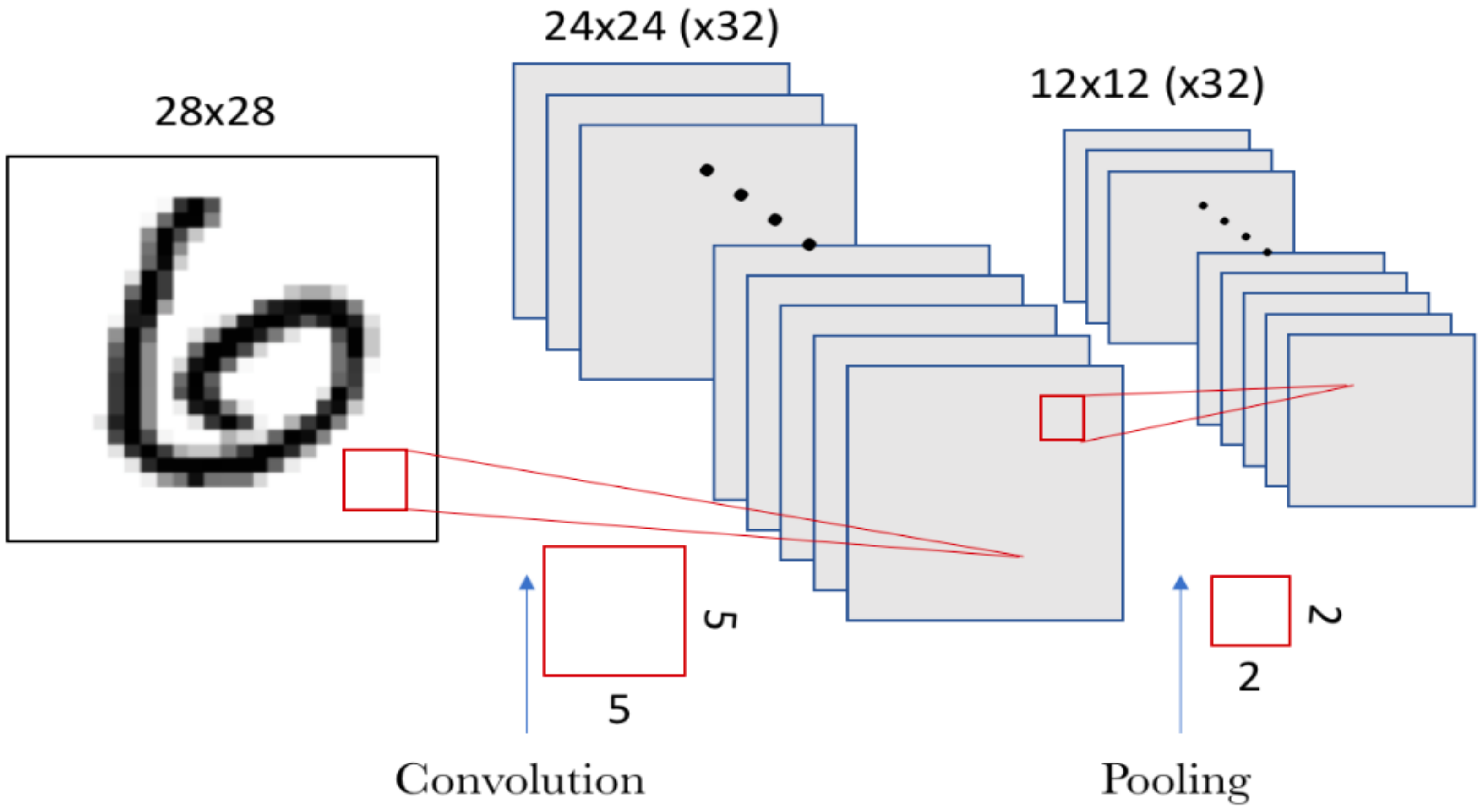
We slide our 2 x 2 window by 2 cells (also called 'stride') and take the maximum value in each region.

The pooling operation

- The pooling maintains the spatial relationship

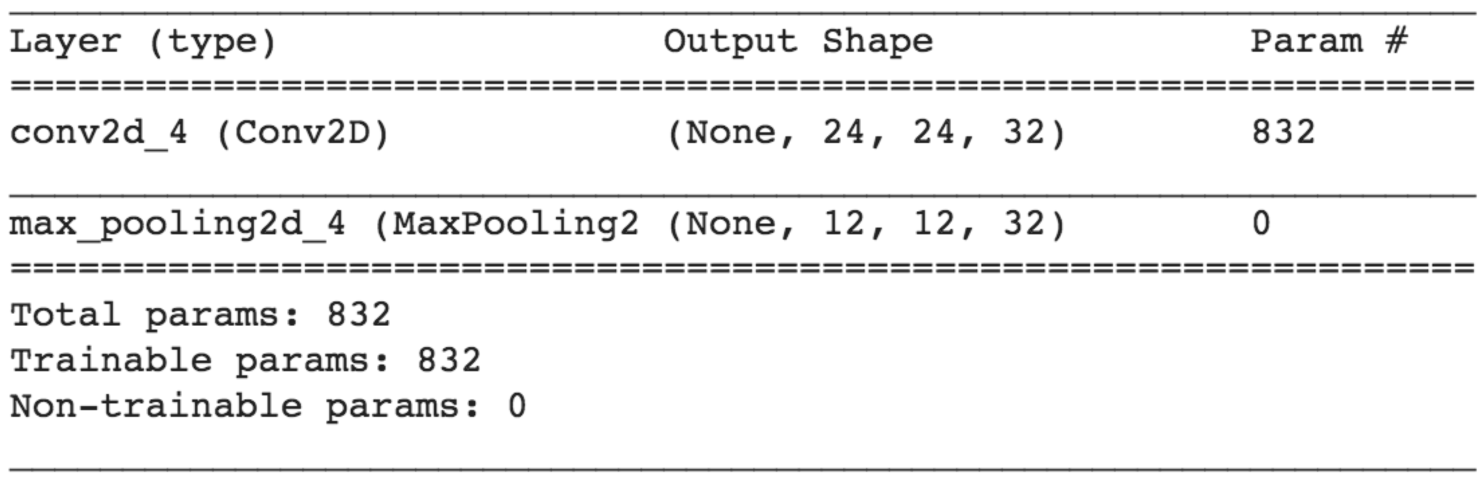


Convolutional+Pooling layers: Summary



Basic elements of a CNN in TensorFlow

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32, (5, 5), activation='relu', input_shape=(28, 28, 1)),
    tf.keras.layers.MaxPooling2D(2, 2),
])
```



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_4 (Conv2D)            (None, 24, 24, 32)        832
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 12, 12, 32)        0
=================================================================
Total params: 832
Trainable params: 832
Non-trainable params: 0
_________________________________________________________________
```

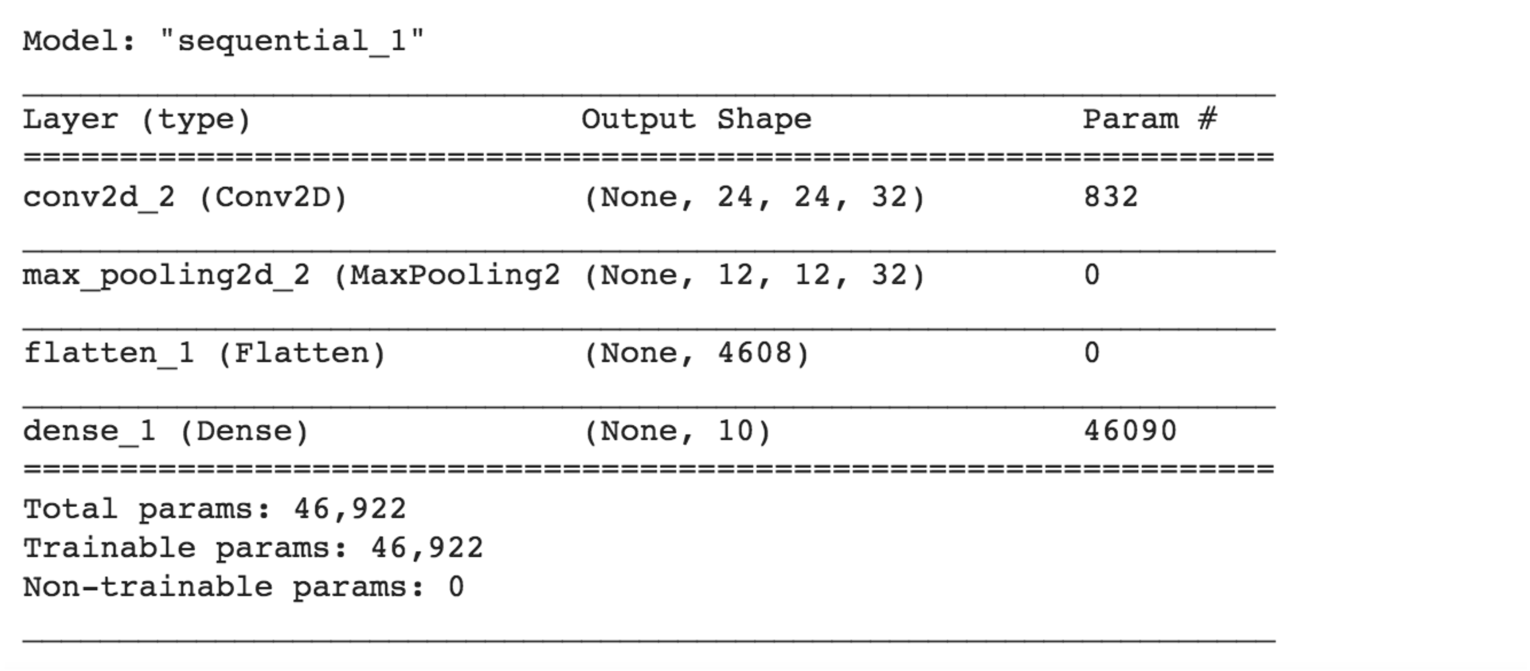
Basic components of a CNN

- **The convolution operation**
  - Intuitively, we could say that the main purpose of a convolutional layer is to detect features or visual features in images.
  - Convolutional layers can learn spatial hierarchies of patterns by preserving spatial relationships.
- **The pooling operation**
  - Accompanies the convolutional layers.
  - Simplifies the information collected by the convolutional layer and create a condensed version of the information contained in them.
- **Classification (Fully Connected Layer)**

Basic elements of a CNN in TensorFlow

```
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(10, activation='softmax')
```

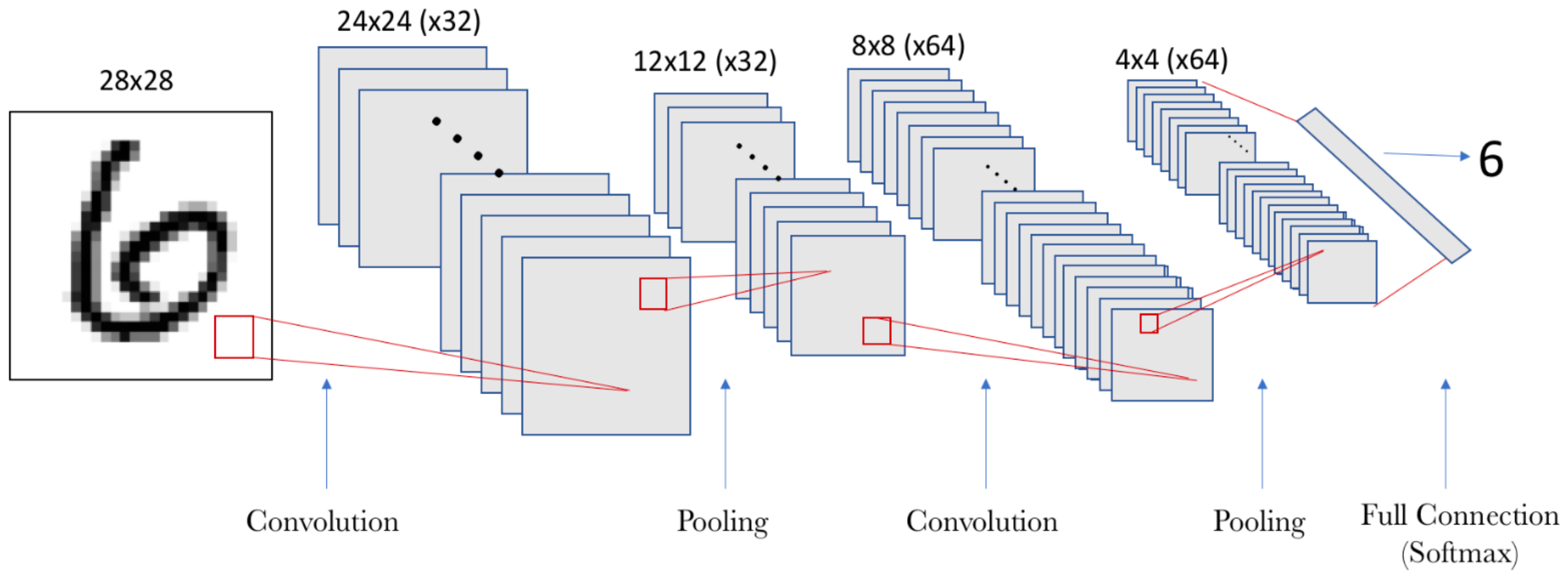
(*) Add a softmax



```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_2 (Conv2D)            (None, 24, 24, 32)        832
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 12, 12, 32)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 4608)              0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                46090
=================================================================
Total params: 46,922
Trainable params: 46,922
Non-trainable params: 0
_________________________________________________________________
```

Global picture: A simple CNN model in TensorFlow

- Visual representation:



Hyperparameters of the convolutional layer

- **Size and number of filters**
    - **The size of the window** (*window_height* × *window_width*) that keeps information about the spatial relationship of pixels are usually 3×3 or 5×5.
    - **The number of filters** (*output_depth*) indicates the number of features and is usually 32 or 64.

```
Conv2D(output_depth, (window_height, window_width))
```

Pf: Deep Learning Basics (using Colab)

- **Tasks included:**
  - task 7.1 – Set up your google colab environment
    task 7.2 – Execute the provided notebook step-by-step
    task 7.3 – Improve the accuracy of your model

part 1

  - task 8.1 – Improving a basic cnn model
    task 8.2 – Exporting the python script
    task 8.3 – Running your first natural network on a login node
    task 8.4 – Submitting your first gpu dl training with slurm
  - task 9.1 – Comparative implementation in pytorch and tensorflow

part 2

Main deep learning frameworks

- **2015 – TensorFlow (Google):**
  Introduced *static computation graphs* — powerful for production but less flexible for research.

- **2016 – PyTorch (Meta / Facebook AI Research):**
  Born from *Torch*, a Lua-based framework, and designed to bring deep learning closer to native Python.

- **Today:**
  PyTorch dominates academic and research use;
  TensorFlow remains strong in industrial and production pipelines.

Building a simple model in PyTorch vs TensorFlow

Concept	PyTorch	TensorFlow
Define model	Subclass nn.Module	Use tf.keras.Model
Forward pass	forward() method	call() method
Training loop	Fully controlled by user	Handled by model.fit()
Flexibility	High (research-friendly)	High-level abstraction (production)

Pf: Deep Learning Basics (using Colab)

- **Tasks included:**
  - task 7.1 – Set up your google colab environment
    task 7.2 – Execute the provided notebook step-by-step
    task 7.3 – Improve the accuracy of your model

    part 1

  - task 8.1 – Improving a basic cnn model
    task 8.2 – Exporting the python script
    task 8.3 – Running your first natural network on a login node
    task 8.4 – Submitting your first gpu dl training with slurm

    part 2

  - task 9.1 – Comparative implementation in pytorch and tensorflow

Pf: Deep Learning Basics (using Colab)

- **In class:**
  - This lab is mainly **pedagogical**, aimed at helping you understand the fundamental concepts of deep learning.
  - During lab class, the instructor will **assist each student individually** to consolidate understanding.
- **Deliverable & Evaluation:**
  - Upload a single PDF (per group) to the **racó@FIB** intranet.
  - This PDF can simply include **screenshots or short notes** showing that you have completed each task.
    The goal is to confirm that you worked through the exercises, not to produce a polished "report".
- **Evaluation day:**
  - A group will give a **brief and informal explanation** of their results.
  - "No preparation is required" — **a few comments or screenshots are enough**.
    The focus is on learning, not on presentation style or timing.