POLITECNICO
MILANO 1863

Hardware Architectures for Embedded and Edge AI

Prof Manuel Roveri – manuel.roveri@polimi.it
Massimo Pavan – massimo.pavan@polimi.it

Exercise session 2 – Tensorflow and CNNs

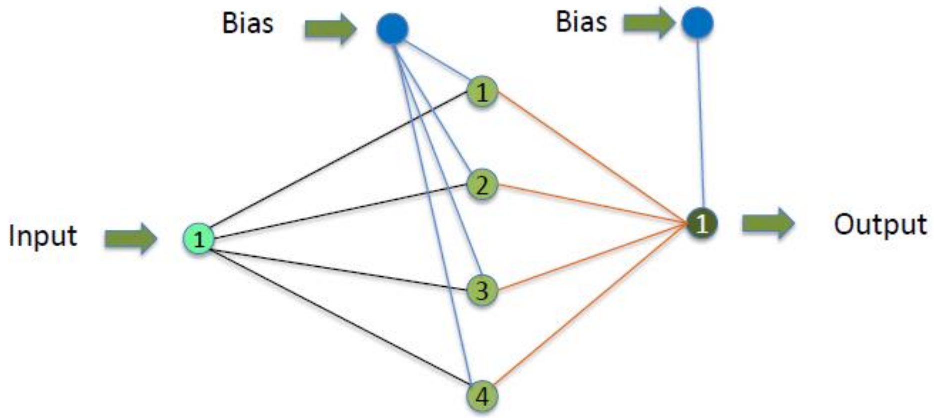
Which types and tasks of Machine Learning will we address?

- Supervised Learning
  - The largest, most mature, most widely used sub-field of machine learning
  - Training data set including desired outputs: $D = \{<x; t>\}$ from some unknown function f
  - Find: A good approximation of f that generalizes well on test data
  - Input variables x are also called features, attributes
  - Output variables t are also called targets, labels
    - If t is discrete: **classification**
    - if t is continuous: **regression**

- (a tiny example of) **Unsupervised Learning**
  - The goal is to learn the representation

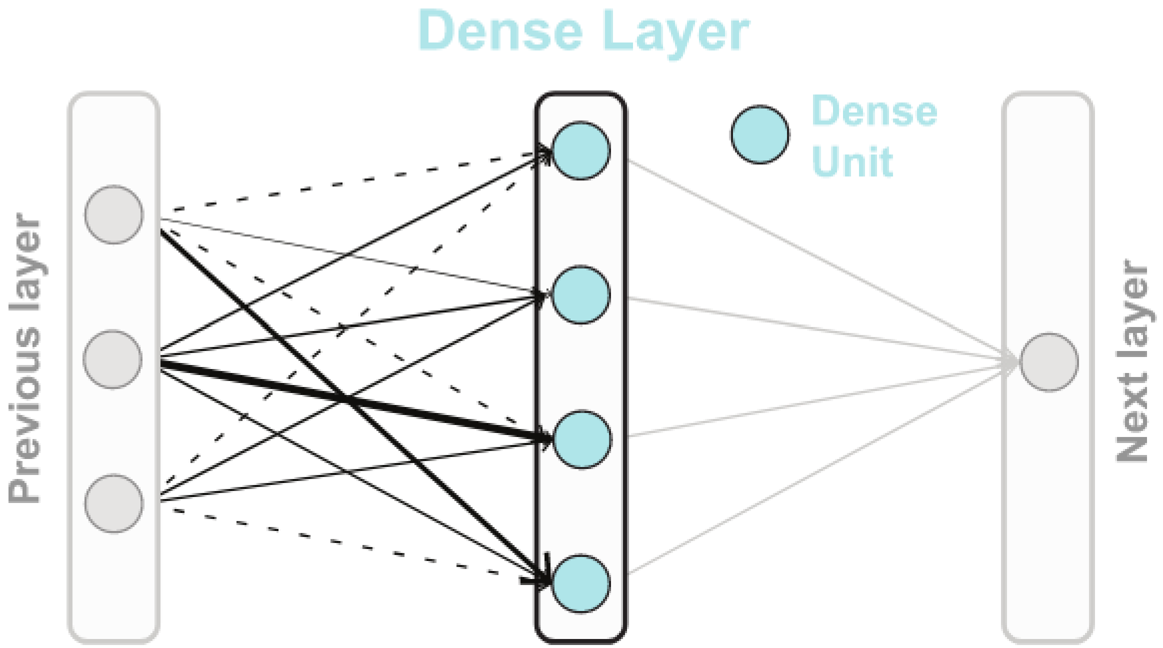
Deep learning

- Neural networks are among the most powerful models in machine learning nowadays
- It basically consists in matrix and vector multiplications and summing. The input is multipied by the **learned** weights of the network and summed to the bias
- But how exactly are those network composed?
- How exactly are the weights learned?



The types of layer: Dense

- A set of units composed by weights and Bias
- Each of the value composing the output of the previous layer is multiplied to the weights of the dense
- The output for each value are then summed along with the bias
- Input is usually 1D
- Each layer is usually followed by a (non-linear) activation function

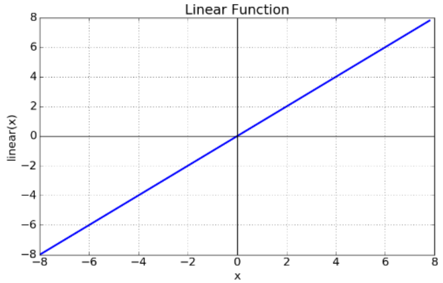


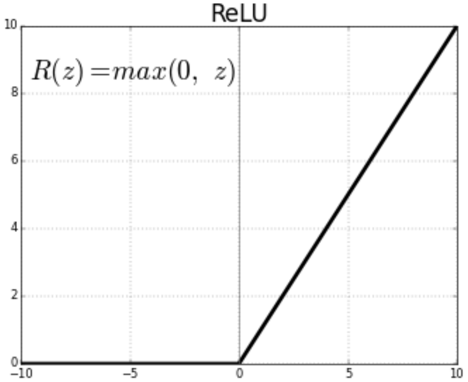
(non-linear) Activation functions

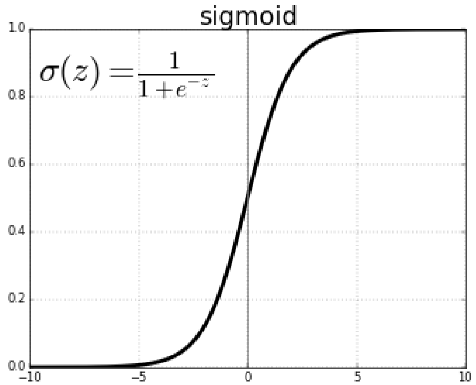
- Without non-linear activations, the network would just be a linear transformation of the input:
    - This would limit a lot the representative power of the model
- Of particular relevance, the activation of the final layer directly modify the output of the network, making it interpretable.
    - For example in case of classification, the output of the n-th neuron of the final layer is interpreted as the probability of the n-th class, and for this reason it must be taken back to values between 0 and 1.

An incomplete list of possible Activation functions

- Linear:
  - Not a proper activation function, just identity
  - Often used as output for regression problems
- Relu:
  - Usually used in the hidden layers
  - Clip to 0 any value smaller than 0
- Sigmoid:
  - clip values between 0 and 1, and is differentiable
  - Usually used as output for binary classification
- Softmax:
  - As sigmoid, but used for multiclass classification

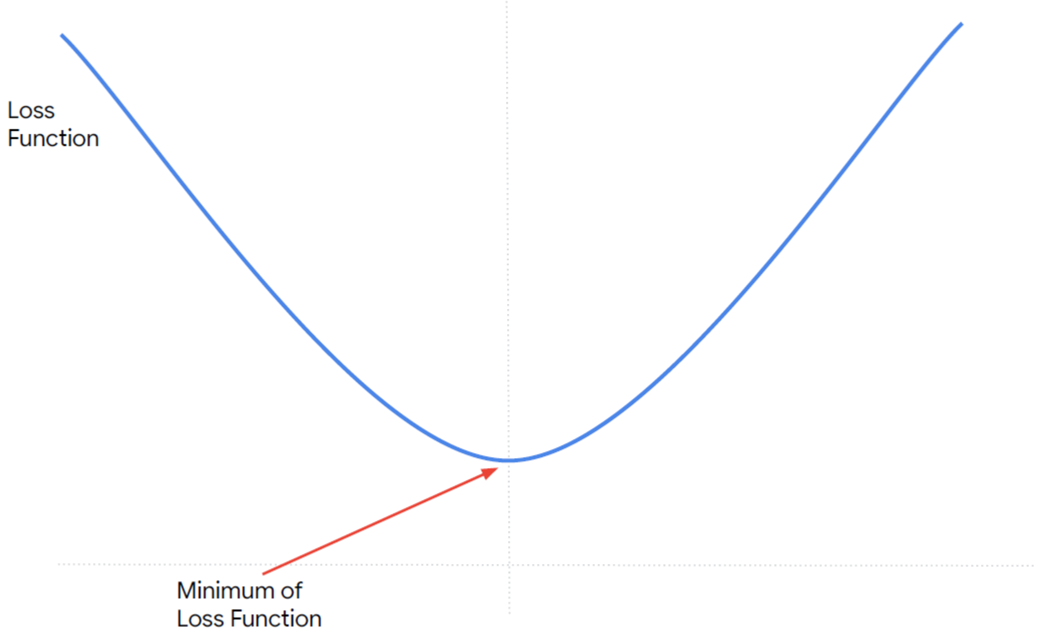






Learning: the loss function

- The weights are initially randomly initialized
- Compute the output starting on your training input data
- Compute an appropriate *loss function*:
  - How far is the computed output from the label (actual output)?
- Use the *loss* to estimate how my weights should be updated in order to obtain a better prediction (output)
- The goal is to find the minimum of this loss function (or to get close to it)



An incomplete list of possible loss function

Called *X* the input, *t* the target label and *Y = f(X)* the output computed by the neural network, having *N* training inputs:

- For regression tasks:
  - Mean squared error:
    - $MSE = \frac{1}{N}\sum_{n=0}^{N}(Y_n - T_n)^2$
- For classification taks:
  - Binary Cross-entropy (for binary classification):
    - $BCE = -\frac{1}{N}\sum_{n=0}^{N}(Y_n \log(T_n) + (1 - Y_n)\log(1 - T_n))$
  - Categorical Cross-entropy (for K class):
    - $CCE = -\frac{1}{N}\sum_{n=0}^{N}\sum_{k=0}^{K}\left(Y_n^k \log(T_n^k)\right)$ * CCE requires one hot encoded labels, but some implementations let you use also spare representation of the targets (Sparse CCE)

An incomplete list of possible optimizers

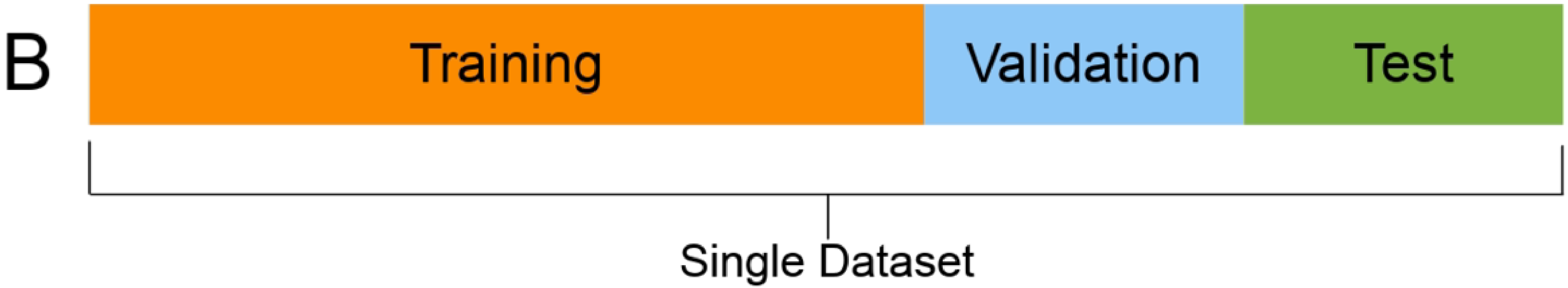
Optimizers manage how the gradient is used to update the weights of the network

- Stochastic gradient descent:
  - Tf.keras.optimizers.SGD()
- Rmsprop
  - Tf.keras.optimizers.RMSprop()
- Adam
  - Tf.keras.optimizers.Adam()

All the listed optimizer requires you to specify a Learning rate ***Lr*** (in case of RMSprop and Adam it's adapted over time)

Training set, validation set and test set

- It's always good to keep part of your data to evaluate the performance of your algorithm
- **Training set** is composed of the data used for properly training the algorithm, or the data used to minimize the loss
- In the validation set there are the data used to choose which is the best among the algorithm that I'm training (e.g.: N layers net vs N+1 layers, net trained for 50 epochs vs 150 epochs). **This results are still biased**!
- With the test set, we establish the performance of the algorithm on data never seen before by the algorithms

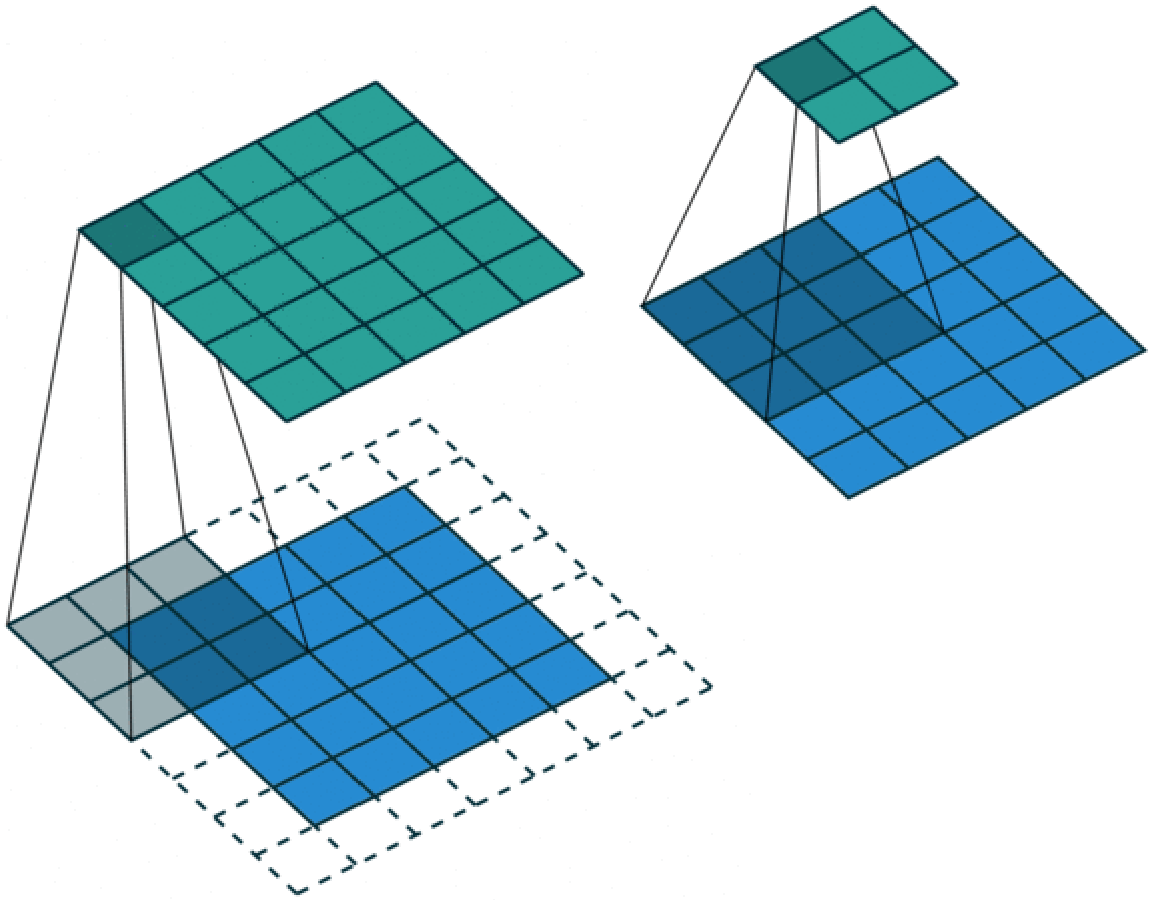


Colab

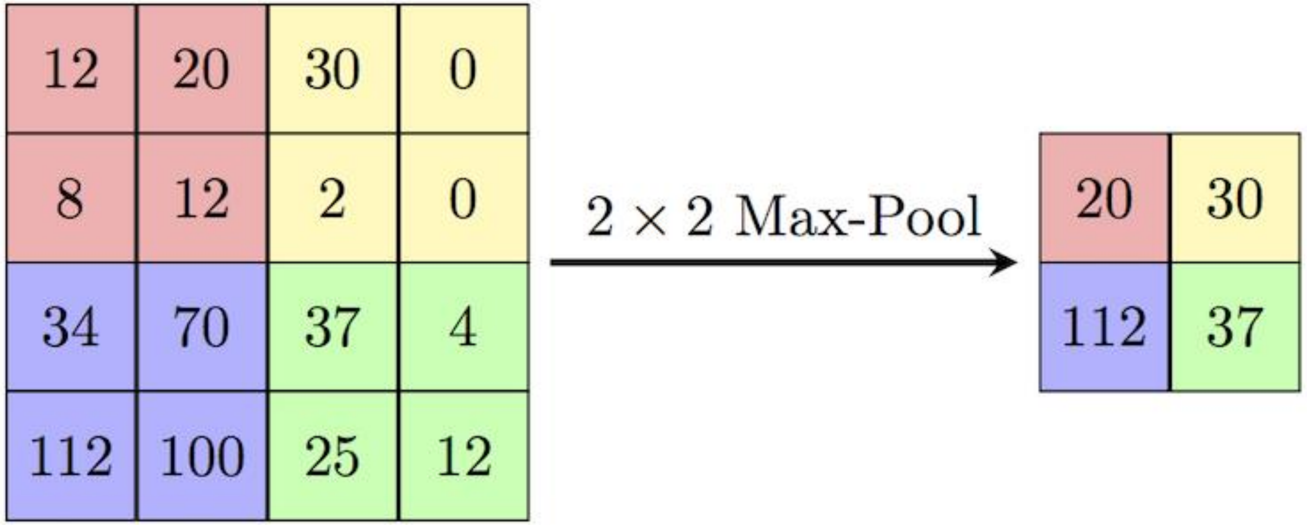
https://oreil.ly/NN6Mj

The types of layer: 2D convolutions

- Input is 3D (height, width, channels)
- As for dense layers, there are weights and Bias
- Characterized by the dimension of the kernel
- Additional relevant parameters:
  - Stride
  - Padding
  - # Filters
- For each filter, it outputs a 2D matrix. The output is consequently 3-dimensional

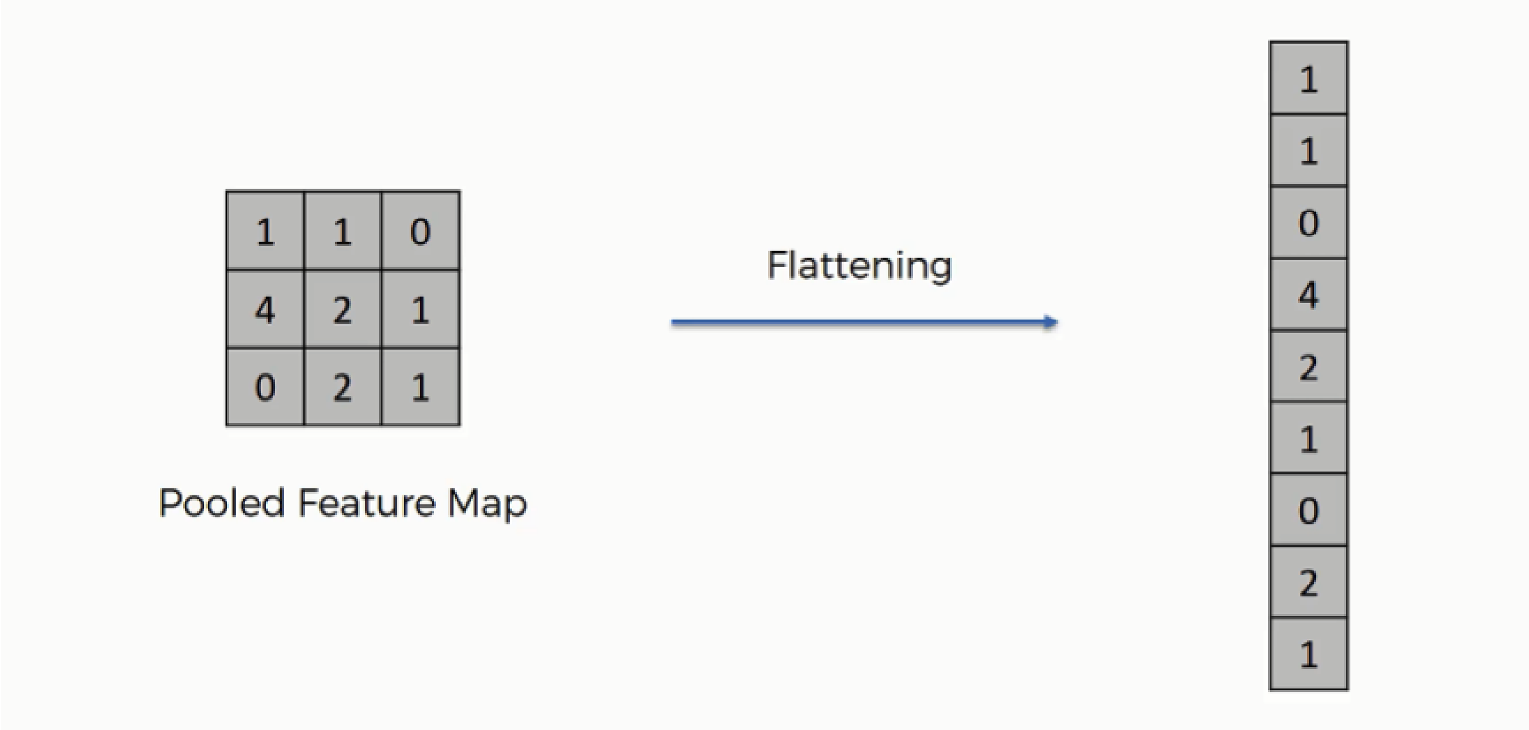


The types of layer: Max Pooling



- Given a dimension for the pooling filter (e.g. 2x2), take the maximum value in the input for each area covered by the filter
- Main goal is to reduce the dimension of the input
- Additional parameters include stride and padding

The types of layer: Flatten

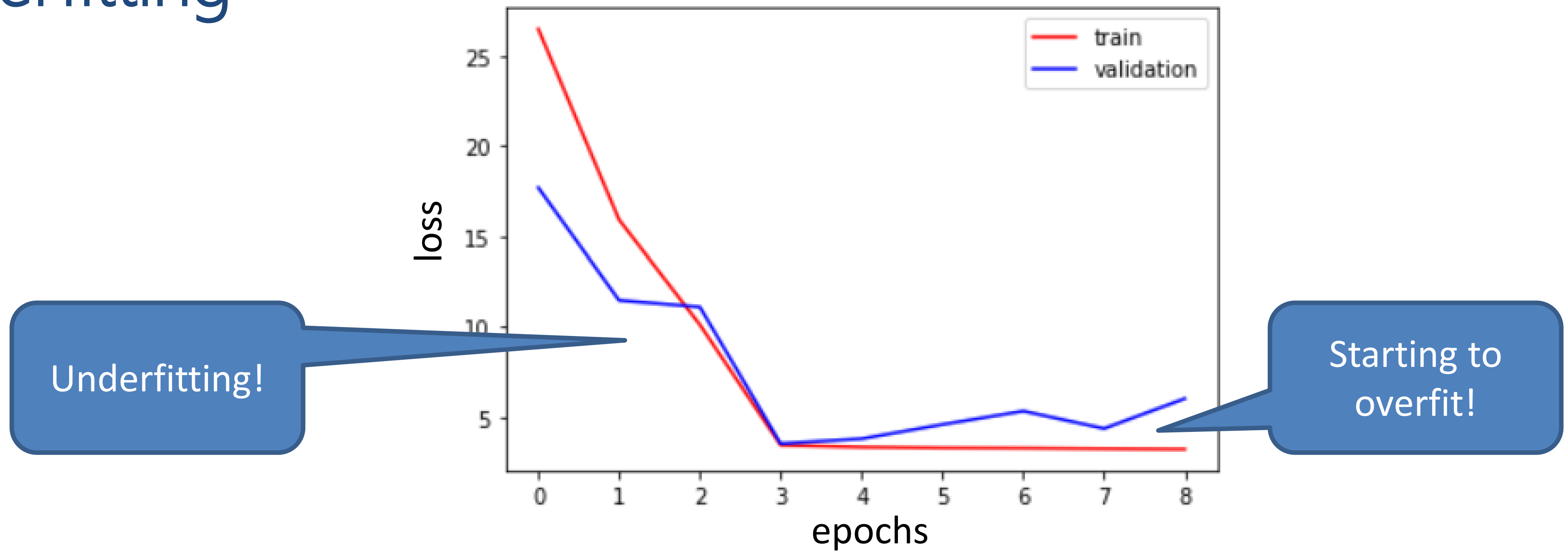


- Take a 2D/3D input and transform it into a vector
- That's all

Colab

https://colab.research.google.com/github/tinyMLx/colabs/blob/master/2-3-5-FashionMNISTConvolutions.ipynb#scrollTo=C0tFgT1MMKi6

Overfitting



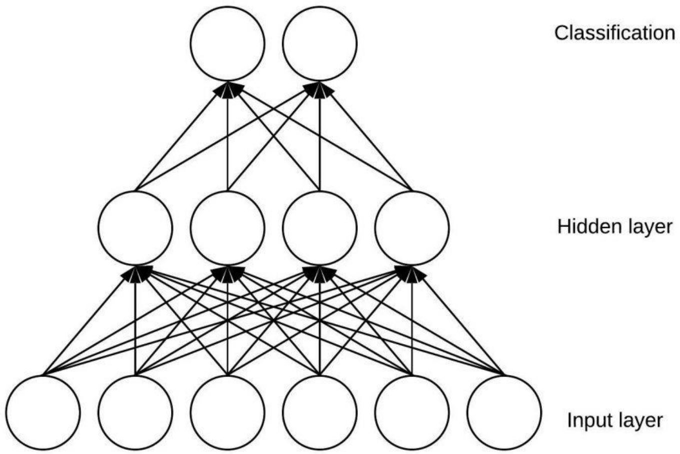
- ML Algorithms could perfectly learn the training set, but if they fail in generalizing to new, unseen data they are useless
- As a general rule, the goal of training is to obtain the lowest value possible for the validation (and consequently testing) loss or performance metric
- When the training loss is continuing to get lower with epochs, but the validation loss is starting to rise – that's a sign that our algorithm are starting to ***overfit***.

How to deal with overfitting

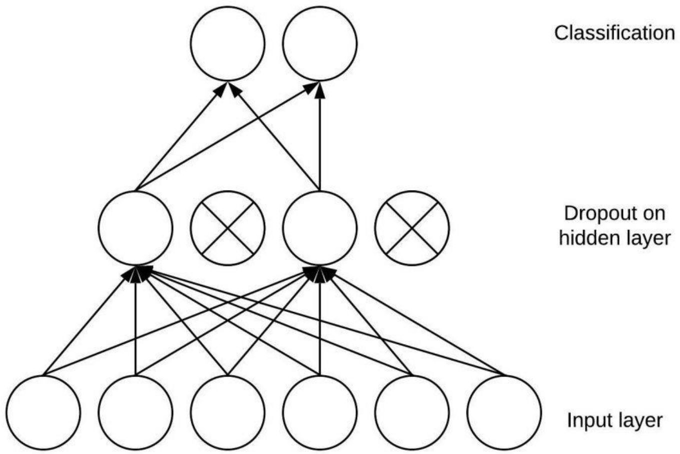
- Collect new data, enlarging the training set
- Perform some sort of Data augmentation
- Add a dropout layer
- Use other type of regularization

The types of layer: Dropout

- Active during training, switched off during inference
- Works with inputs of any dimensions
- «switch off» a given percentage of the nodes in a network while training on a batch of data.
- By making use of less nodes, the network improve its generalization capabilities



Without Dropout



With Dropout

Colab

https://colab.research.google.com/github/tinyMLx/colabs/blob/master/2-3-9-AssignmentQuestion.ipynb

Appendix

Credits and reference

- The colab examples are taken from the book and from the online course:

- “TinyML: Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers”, Daniel Situnayake, Pete Warden, O'Reilly Media, Inc.
- Online course:
  - https://www.edx.org/professional-certificate/harvardx-tiny-machine-learning
- A lot more material on TinyML:
  - http://tinyml.seas.harvard.edu/

- Pictures are mostly from the web and from the material cited
- Definitions of Machine Learning in the first slide are taken from the slide of the course Machine Learning at Politecnico di Milano, held by M. Restelli