Deep Neural Networks

Jean Kossaifi

Machine Learning

- **Machine Learning**: **learning** (parameters of a model) from **data**

- **Supervised Learning**: dataset of samples (**features)** and **labels:** given a sample, predict the corresponding label — class (**classification**) or continuous value (**regression**)

Machine Learning

- **Machine Learning**: **learning** (parameters of a model) from **data**

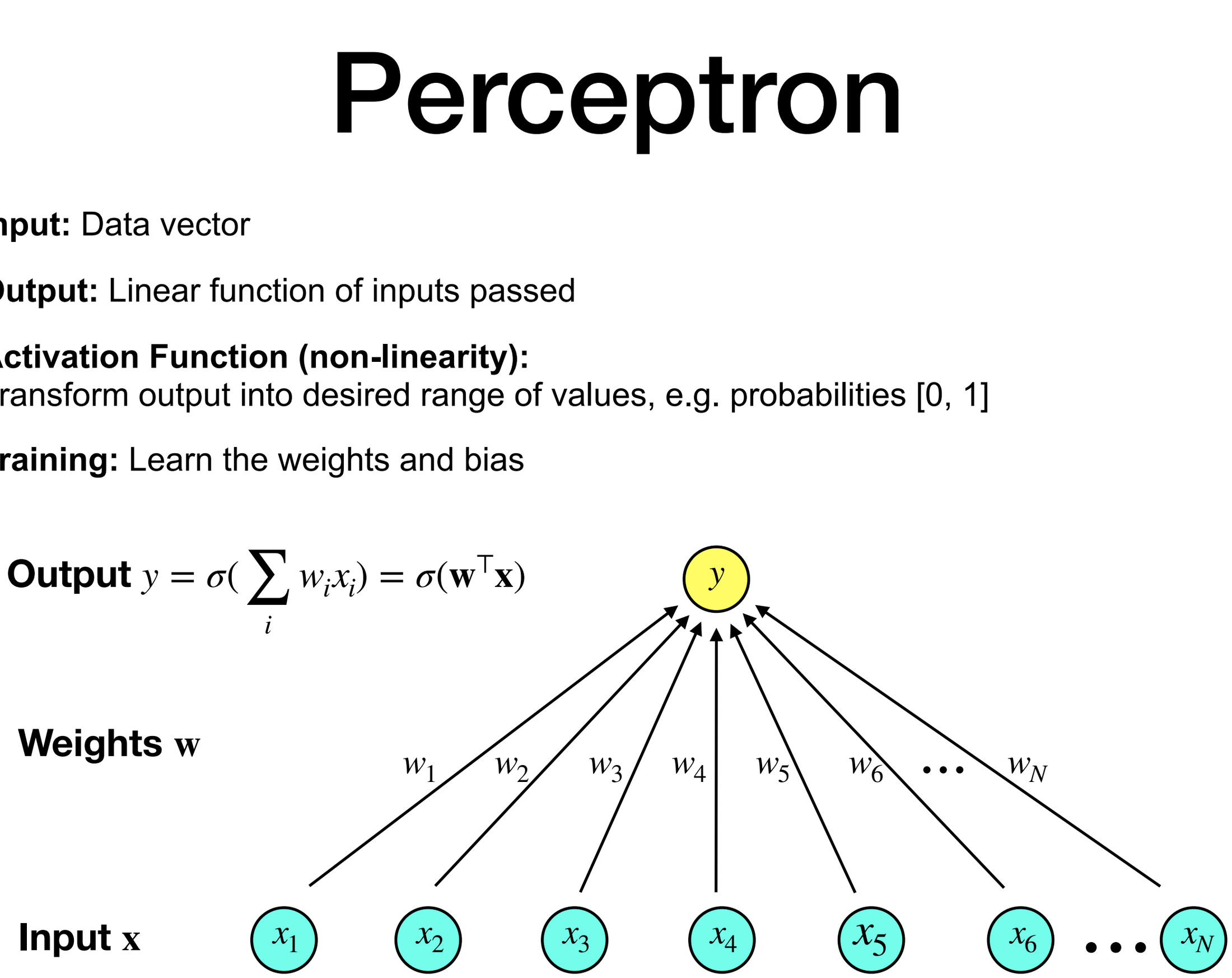
- **Supervised Learning**: dataset of samples (**features)** and **labels:** given a sample, predict the label class (**classification**) or continuous value (**regression**)

- **Generalise** to new, **unseen data** (test data)
  Failure to do so is called **overfitting**

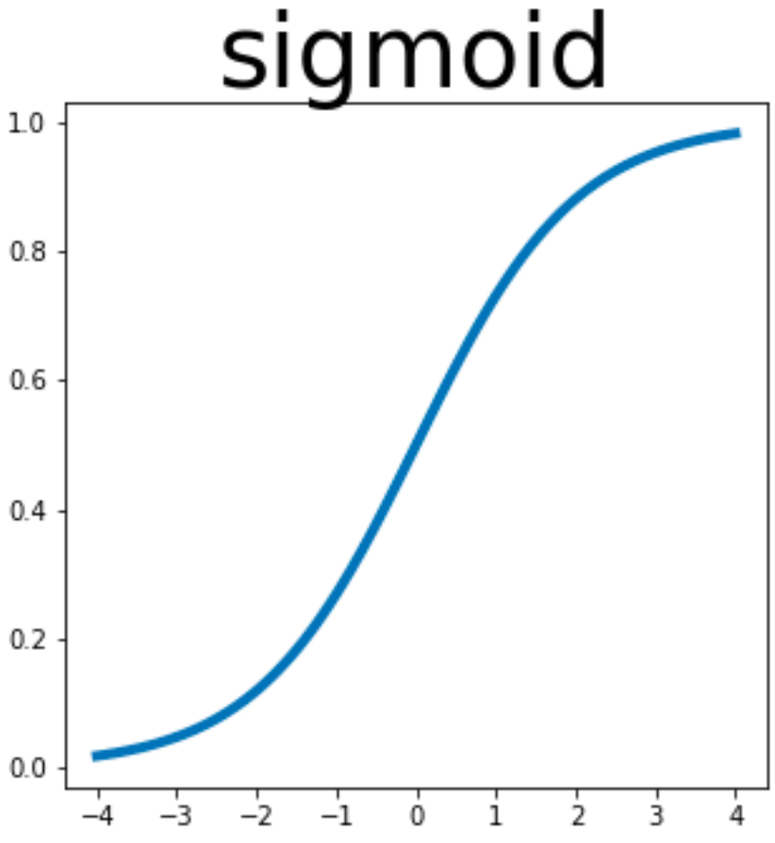
- Always divide your dataset into **disjoint training** and **testing** sets.

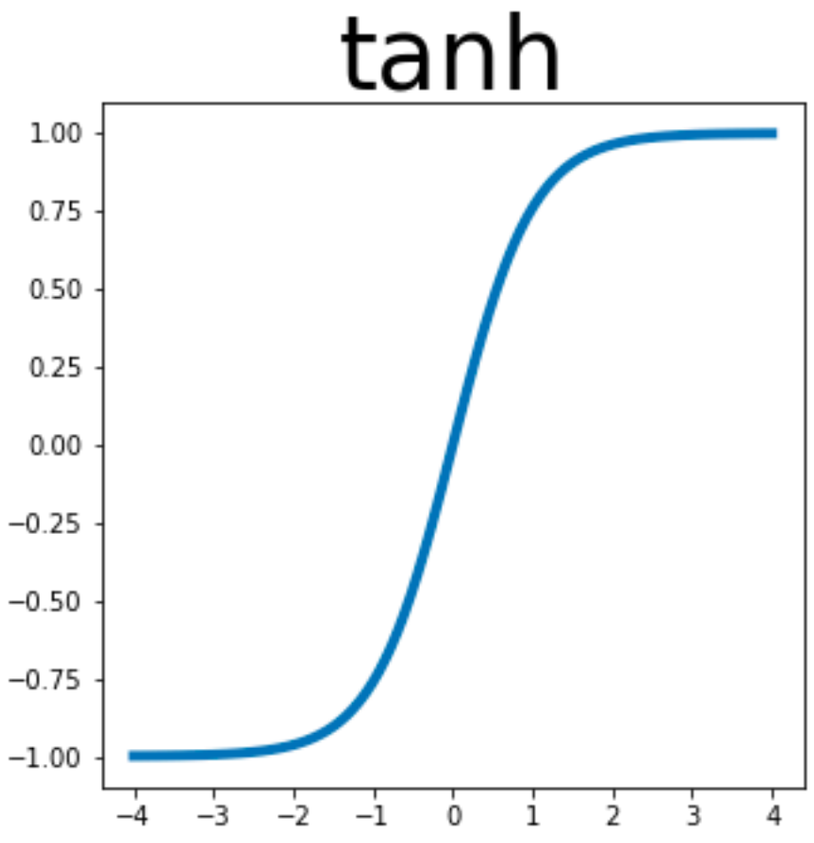
Perceptron

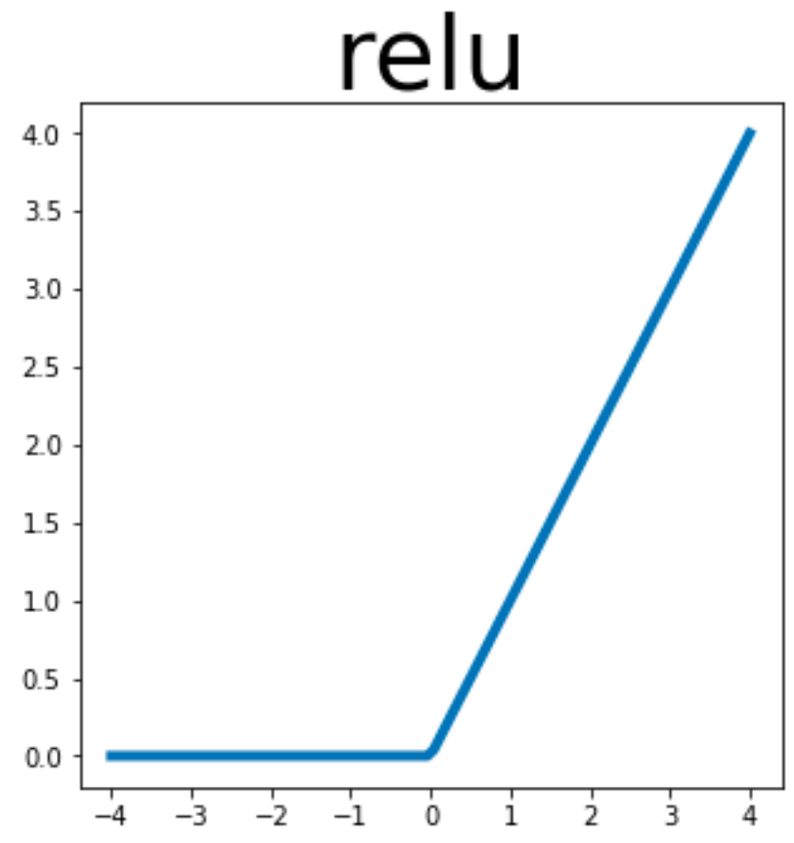
- **Input:** Data vector
- **Output:** Linear function of inputs passed
- **Activation Function (non-linearity):**
  Transform output into desired range of values, e.g. probabilities [0, 1]
- **Training:** Learn the weights and bias



Activation Functions







Multi-Layer Perceptron

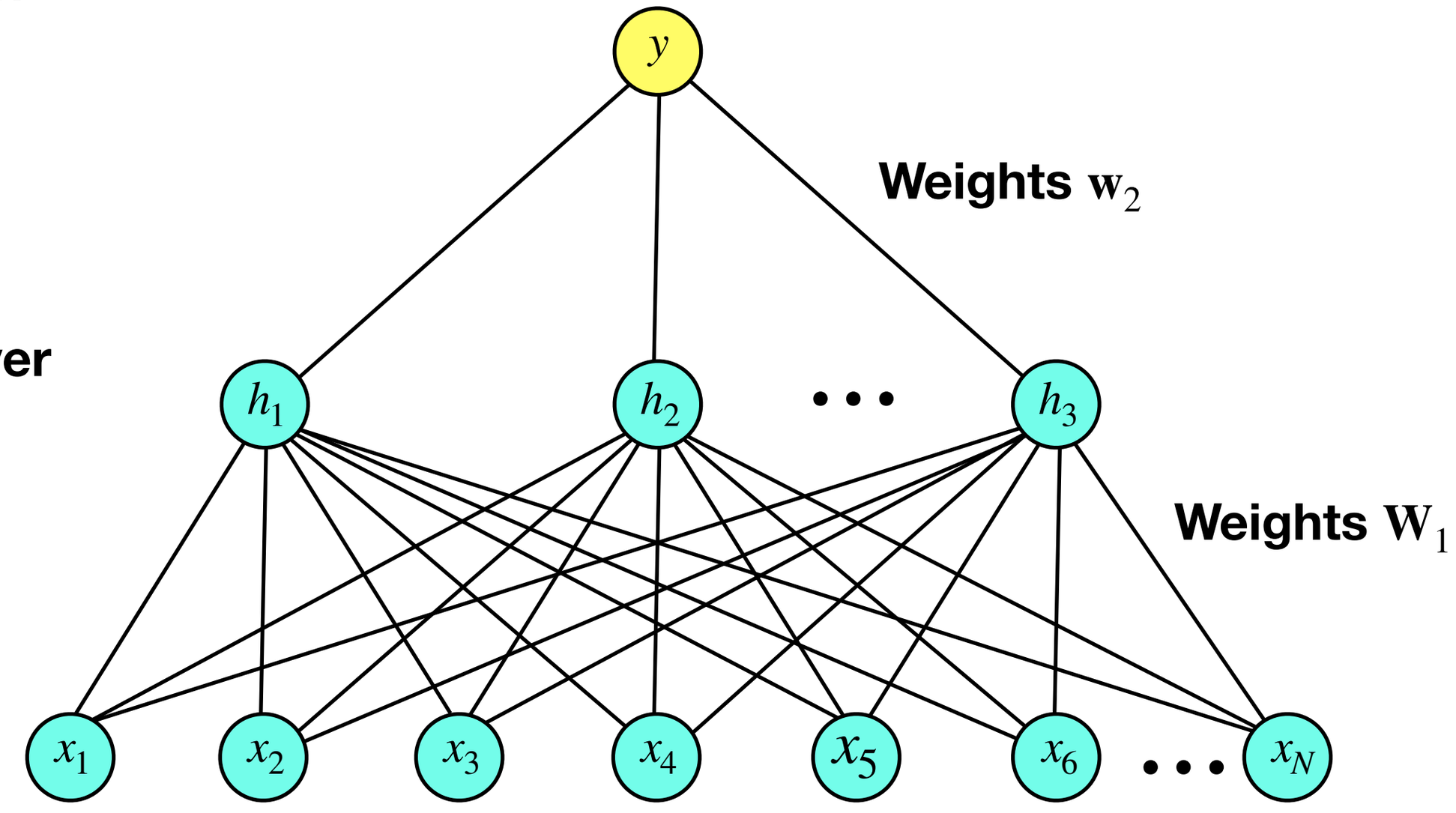
Output layer

$$y = \sigma(\mathbf{w}_2^\top \mathbf{h})$$

Hidden Layer

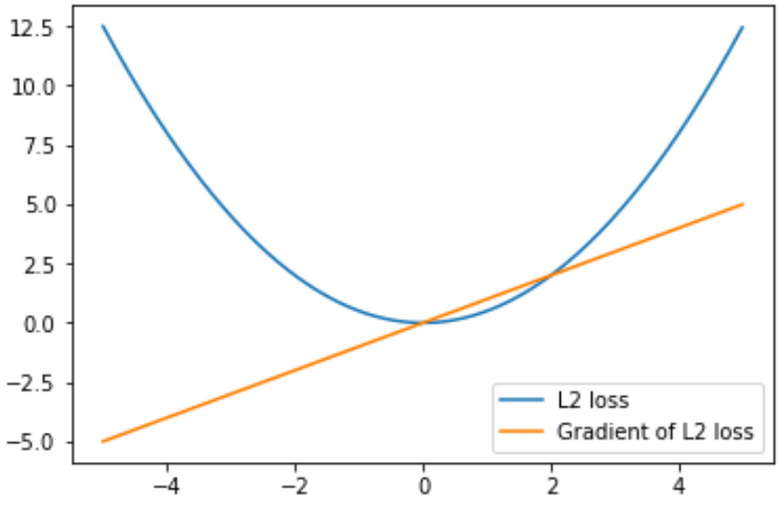
$$h_i = \sigma(\mathbf{W}_1 \mathbf{x})$$

Input $\mathbf{x}$



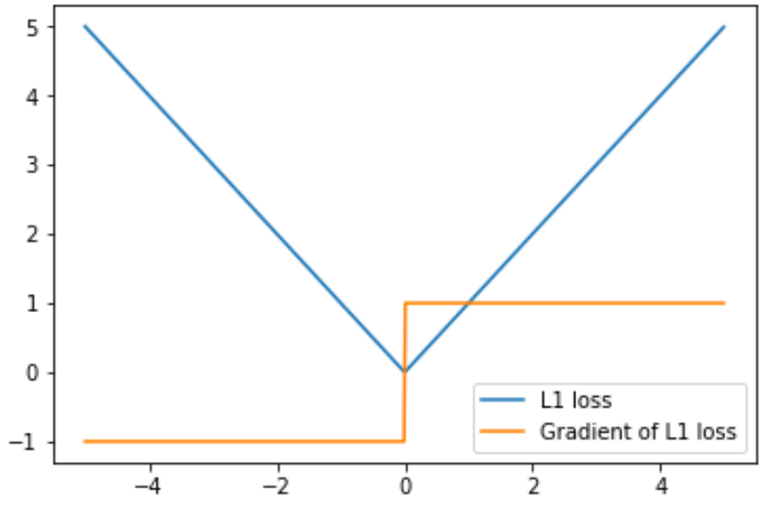
Losses

- How far is our prediction f(x) from the target label y?



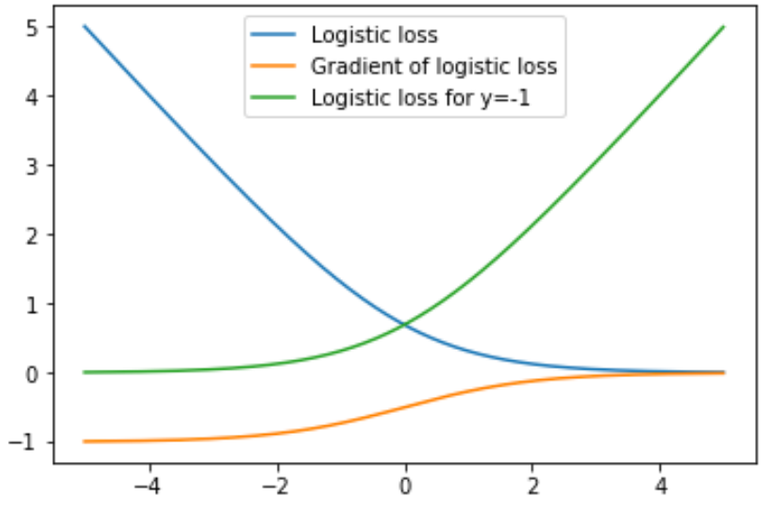
L2-loss

$$\frac{1}{2}\|f(x)-y\|^2$$



L1-loss

$$|f(x)-y|$$



Logistic loss

$$\log\left(1+\exp\left(-yf(x)\right)\right)$$

Training the Model

- **Loss:** $\mathscr{L}(\hat{y}_i, y_i)$

- **Layer representation:** $f_i(x_i, w_i)$
  each layer performs a transformation

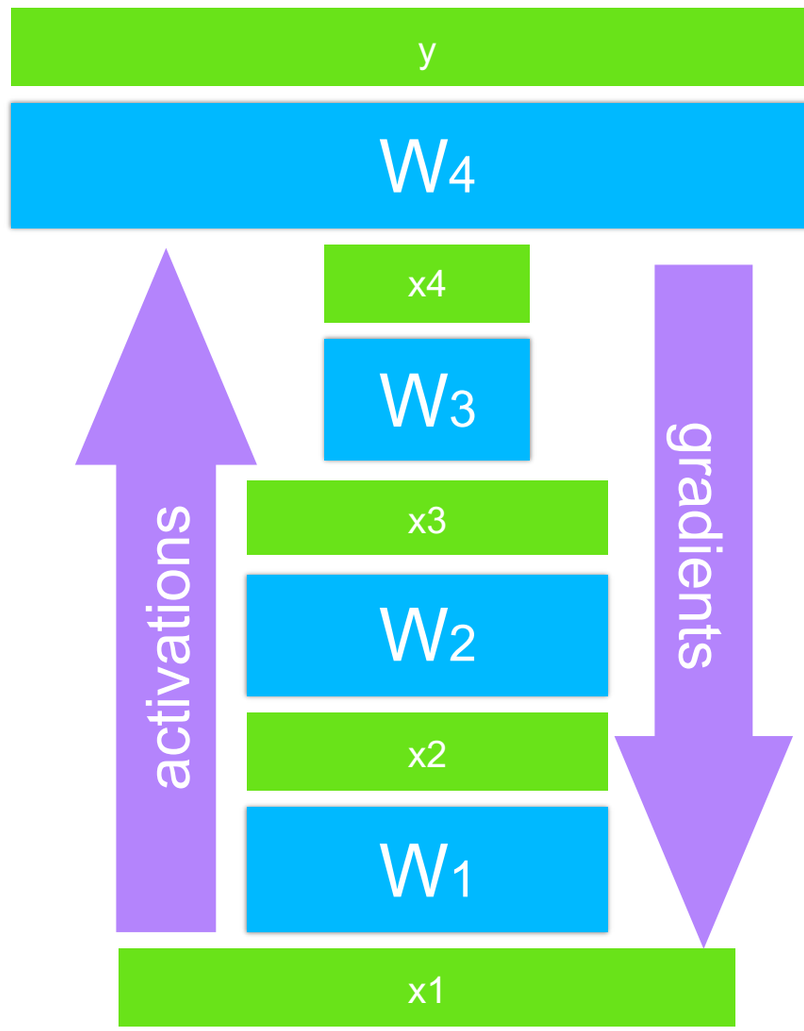
- **Gradient back-propagation (chain rule):**

$$\frac{\partial g \circ f}{\partial x} = \frac{\partial g}{\partial f}\frac{\partial f}{\partial x}$$

Next layer

Current layer

- **Update rule:** $\mathbf{W}_i \leftarrow \mathbf{W}_i - \eta \frac{\partial \mathscr{L}(\hat{y}_i, y_i)}{\mathbf{W}_i}$



Deep Convolutional Neural Networks

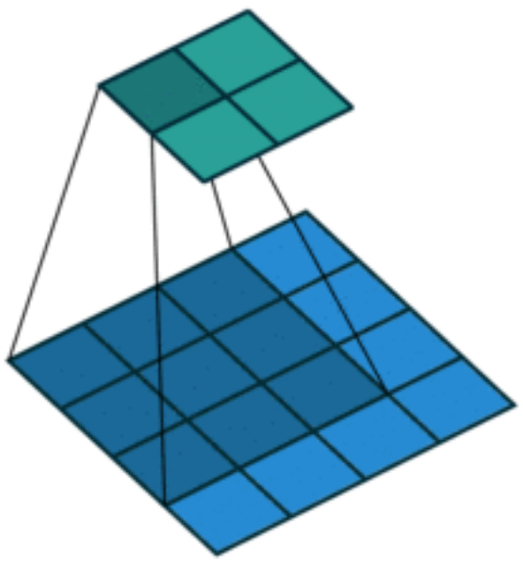
- Why not simply use the Multi-Layer Perceptron ?

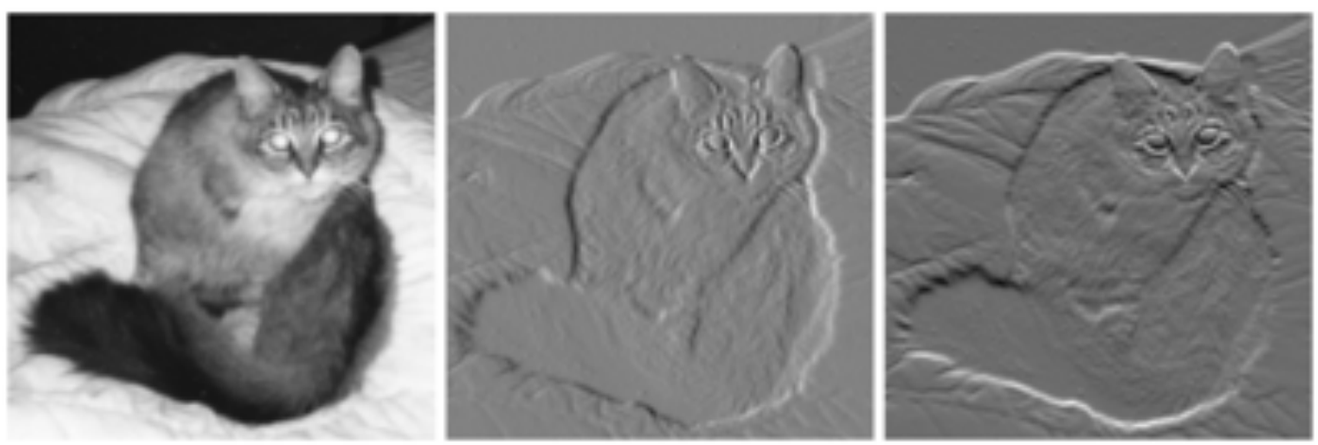
- MLP ignore multi-dimensional structure (e.g. spatial) in the data.

- Number of parameters:
  Imagine classifying images in various classes
  input = RGB images of size 64 x 64 x 3
  1000 hidden units in the first layer
  => = 3 x 64 x 64 x 1000 >12 millions parameters!!!

- Learnable features

Convolutions



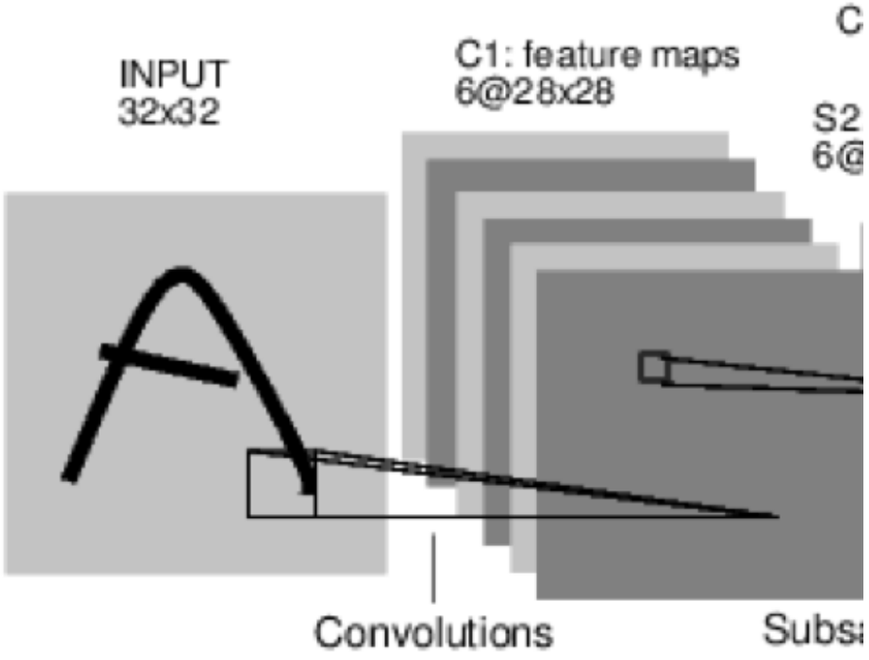


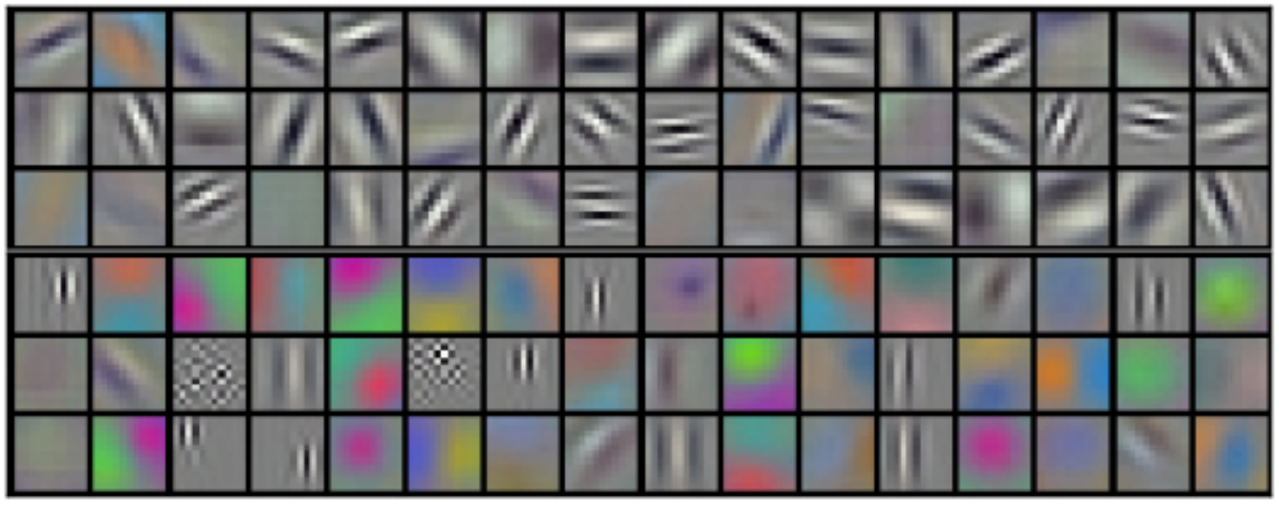
$$K = \begin{bmatrix} -1 \\ 0 \\ +1 \end{bmatrix}$$

- Convolve image with a filter
- Typically, features were hand-crafter (HOG, SIFT, LBP, …)
- DCNN allows learning the features

Convolutional Layers

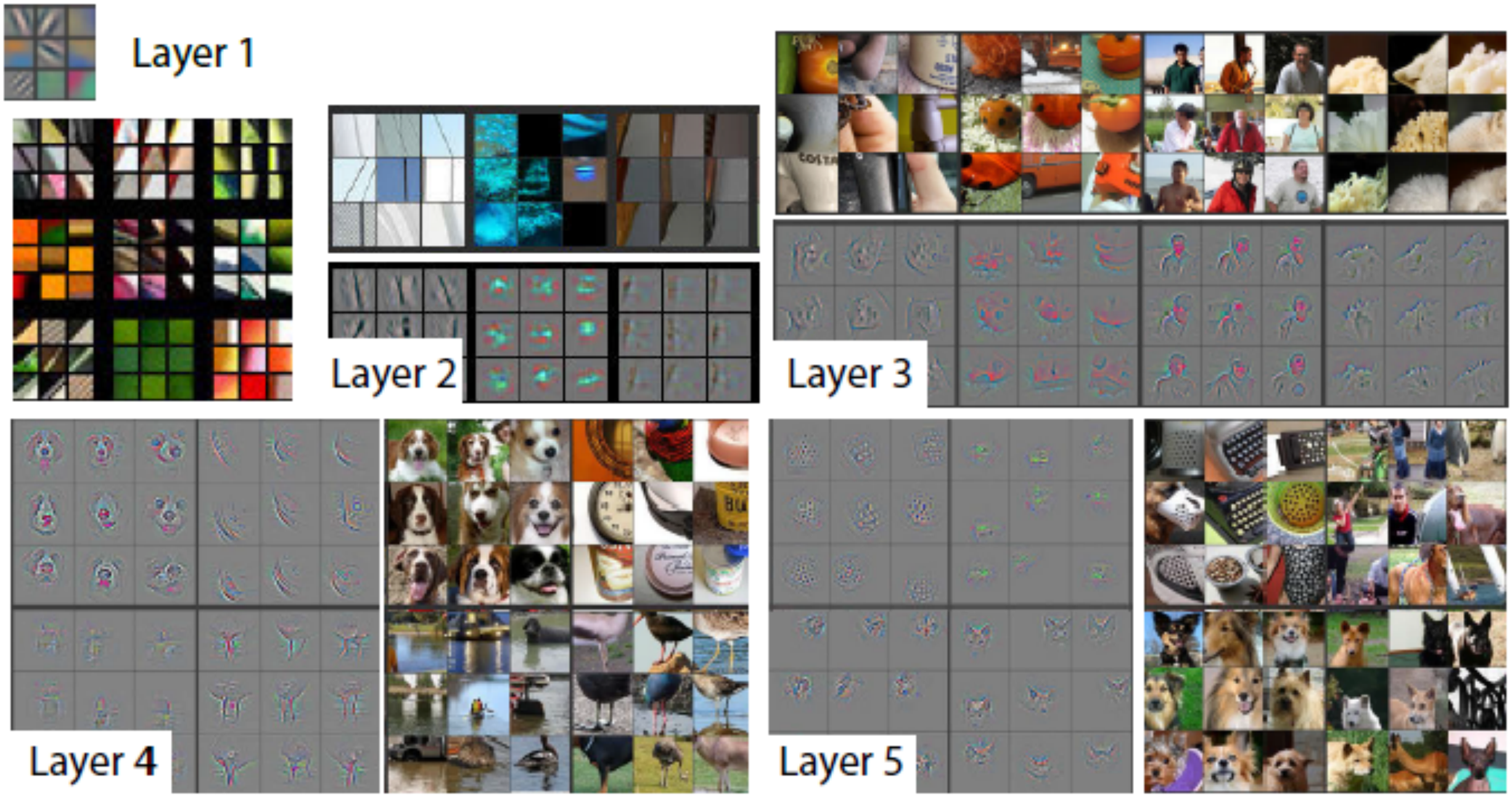
- Images have translation invariance (to some extent)
- Low level is mostly edge and feature detectors
- Usually via convolution (plus nonlinearity)





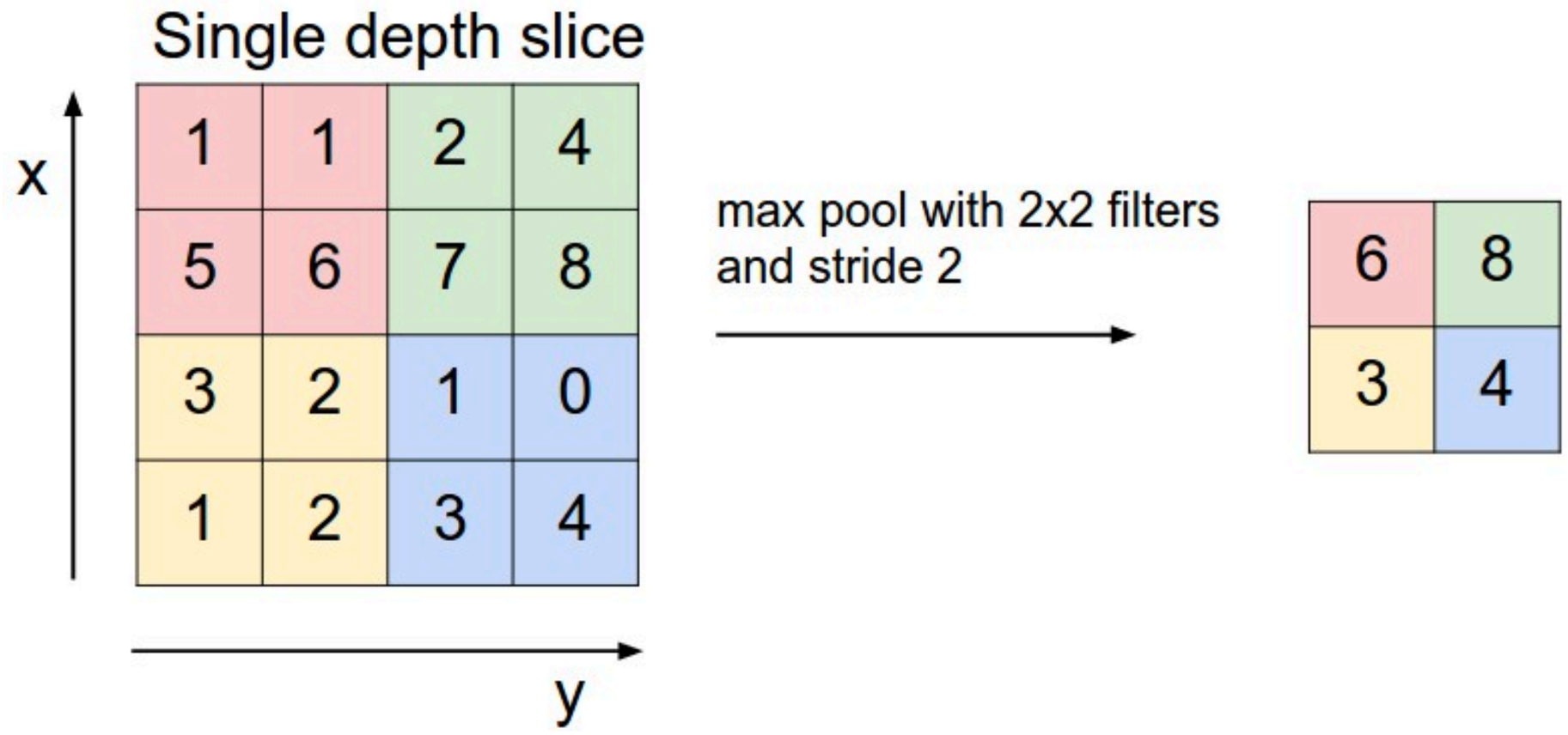
Convolutional Layers

Multilevel feature extractions from raw pixels to semantic meanings



Pooling: reduce dimension

- Subsampling: average over patches
- Max-Pooling: maximum over patch — better in practice



How to train your CNN

- Normalise data (zero mean over features, divide by std)
- BatchNorm: force activations to be normally distributed
- Regularise to prevent overfitting
- Dropout: randomly set neurons to 0

Try it for Yourself!

- Jupyter Notebook