Introduction to Deep Learning

Graduate School Orléans Numérique

Lecture 1: from the neuron to a first classifier

Daniel Förster
UFR Sciences et Techniques
Université d'Orléans

Agenda

- What is deep learning, and why now?
- From the biological neuron to the perceptron
- Building a network: layers and activation functions
- Datasets and best practices
- Regression vs. classification: output activations, loss functions, metrics
- First real-world example: handwritten digits (MNIST)



`daniel-forster.fr/dl`

slides, problem sheets, notebooks, project ideas

Course organisation

Ten sessions; dates, times and rooms on the course website

	Session
1	**Lecture 1:** introduction, principles, datasets
2	Practical 1
3	**Lecture 2:** gradient descent, backpropagation
4	Practical 2, then **multiple-choice test 1**
5	**Lecture 3:** convolutional networks, autoencoders
6	Practical 3, then **multiple-choice test 2**
7	**Lecture 4:** outlook, advanced topics
8	Practical 4
9	Project work
10	Project presentations and discussion

Grading

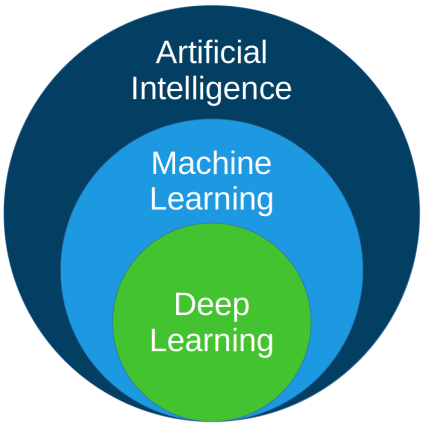
- Multiple-choice tests: 50 %
- Mini-project in groups of two or three: 50 % (dataset and task ideas on the website, or propose your own); presented in the last session

Practicals: Python with TensorFlow/Keras, notebooks provided

Textbooks (all recommended)

- Prince, *Understanding Deep Learning* (2023), free online
- Goodfellow, Bengio, Courville, *Deep Learning* (2016), free online
- Chollet, *Deep Learning with Python*, 2nd ed. (2021)

What is Deep Learning?



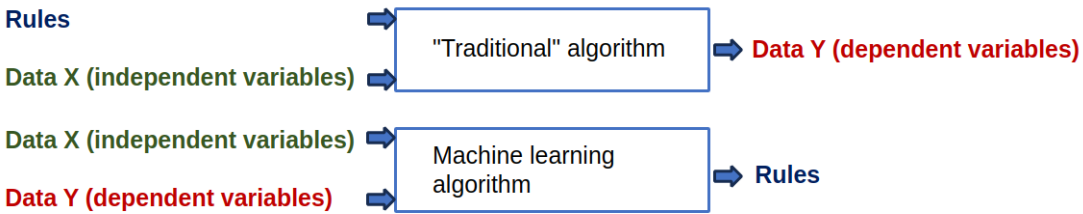
Artificial intelligence:
A program that can sense, reason, act, and adapt

Machine learning:
Algorithms that learn from data

Deep learning:
Machine learning using multi-layered artificial neural networks

Today's "generative AI" (large language models, image generation) is deep learning at very large scale.

Machine learning vs. traditional programming



- The "rules" (the model) are **learned from examples** instead of being written by hand
- Requires large amounts of labelled data
- Data can sometimes be obtained through **simulation** (common in science, where experiments are expensive)

Place of machine learning in science

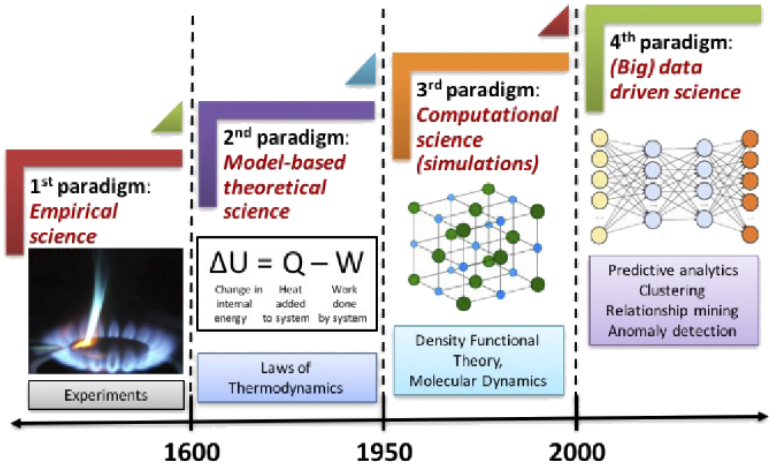


Image source: Agrawal & Choudhary, APL Mater. 4, 053208 (2016)

Deep learning history: 1943–2006

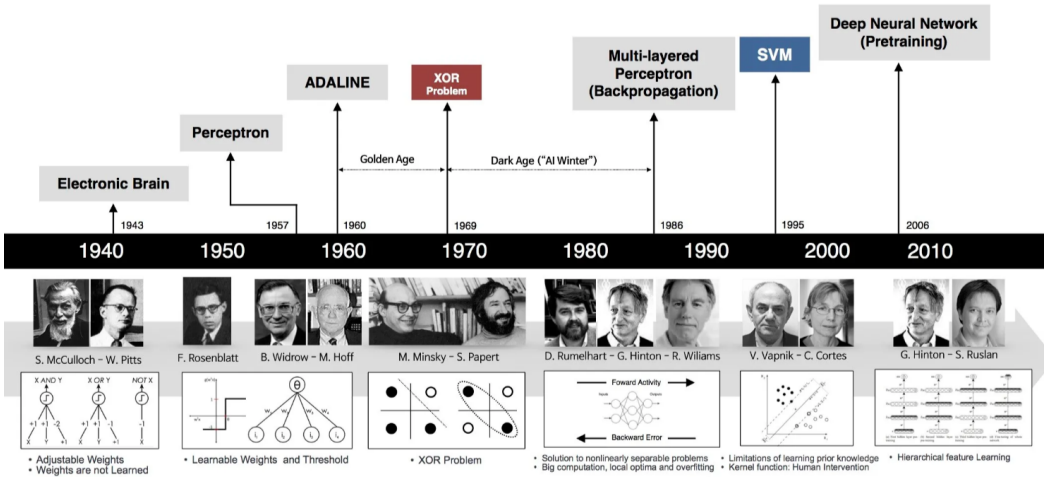
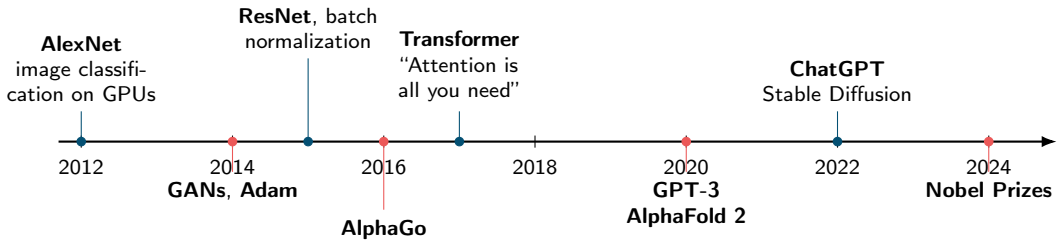


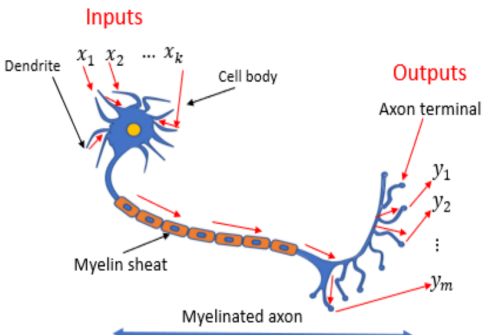
Image source: Int. J. Electr. Comput. Eng. 10, 538 (2020)

Deep learning history: 2012–today



- **2024 Nobel Prize in Physics:** J. Hopfield and G. Hinton, "for foundational discoveries and inventions that enable machine learning with artificial neural networks"
- **2024 Nobel Prize in Chemistry:** D. Baker, D. Hassabis and J. Jumper, computational protein design and protein structure prediction (AlphaFold)
- Drivers of the last decade: **data** (internet, sensors, simulations), **compute** (GPUs), **open software** (TensorFlow, PyTorch)

Biological neuron



$\rightarrow$ Inspiration for artificial neural networks: many inputs (dendrites), one output (axon), a threshold

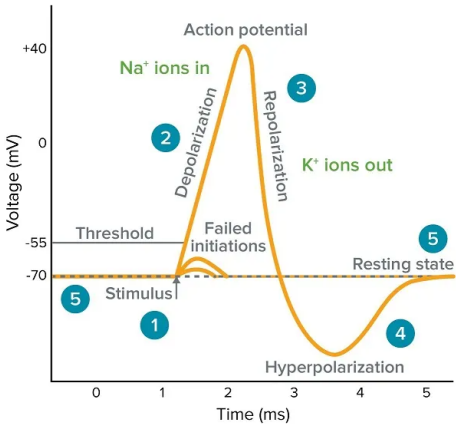
Learning in the brain: synapses change their strength with activity ("cells that fire together wire together"). The rule is **local**: a synapse only sees its two neurons, plus global reward signals such as dopamine.

Backpropagation (lecture 2) is not what the brain does, but it is what works.

Sample efficiency: a child learns what a cat is from a handful of examples; a network needs thousands of labelled images.

Image source: Artif. Life Robotics 27, 2022

Action potentials



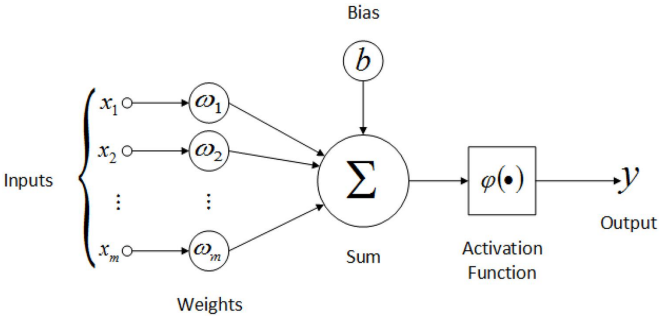
All-or-none principle: an action potential is either generated or not, according to a voltage threshold

Frequency of action potentials encodes signal intensity

How deep is the brain? Recognising an object takes about 150 ms (Thorpe et al. 1996); one stage of neurons (integrate, fire, transmit) takes about 10 ms $\rightarrow$ at most 10–15 sequential stages. Fast vision is a feed-forward network about 10 layers deep (AlexNet: 8).

Image source: Molecular Devices

Anatomy of an artificial neuron



Output y:

$$y = \phi\left(\sum_{i=1}^{m} x_i w_i + b\right)$$

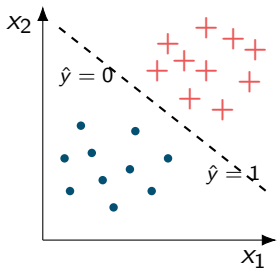
with m inputs x_i, "weights" w_i, "bias" b, and "activation function" ϕ.

Perceptron (Rosenblatt, 1957): the activation function is the Heaviside step function

$$\Theta(z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases} \quad \rightarrow \text{ the neuron "fires" or not, like the biological neuron}$$

Image source: Medium

Perceptron in 2D: a linear classifier



Output: $\hat{y} = \Theta\left(w_1 x_1 + w_2 x_2 + b\right)$

Decision boundary $w_1 x_1 + w_2 x_2 + b = 0$, i.e. the line $x_2 = -\frac{w_1}{w_2} x_1 - \frac{b}{w_2}$

How can w_1, w_2, b be obtained?

Training set: inputs (x_1^i, x_2^i) with labels $y^i \in \{0, 1\}$

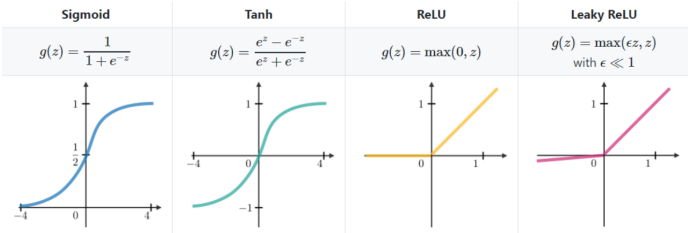
Perceptron learning rule: for each example i, compute $\hat{y}^i$ and update

$$w_j \leftarrow w_j + \eta \left(y^i - \hat{y}^i\right) x_j^i$$
$$b \leftarrow b + \eta \left(y^i - \hat{y}^i\right)$$

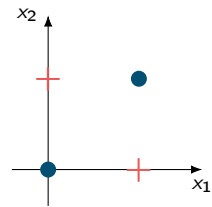
where η is the "learning rate". Correct prediction $\rightarrow$ no change; wrong prediction $\rightarrow$ the boundary moves towards the misclassified point.

Limitation: a single perceptron can only separate classes by a straight line (hyperplane).

Activation functions: why we need nonlinearity



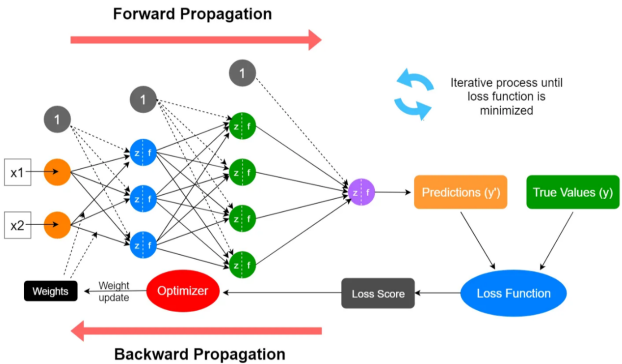
- Smooth (or piecewise-linear) replacements for the step function: differentiable, which is needed for training (next lecture)
- **ReLU** is the default choice for hidden layers today
- Without a nonlinear ϕ, stacking layers is useless: $W_2(W_1 x + b_1) + b_2$ is still linear in x



`xor`: output 1 if exactly one input is 1. No straight line separates the two classes (Minsky & Papert, 1969): a single perceptron cannot learn it, a network with one hidden layer can.

Image source: Stanford CS229 cheatsheet

Building a network: layers



Dense (fully connected) layer: each of the n_{out} neurons sees all n_{in} inputs

$$\mathrm{y} = \phi\left(W\mathrm{x} + \mathrm{b}\right), \qquad W \in \mathbb{R}^{n_{\text{out}} \times n_{\text{in}}}$$

- Trainable parameters per layer: $n_{\text{in}}\, n_{\text{out}} + n_{\text{out}}$
- **Input layer** $\rightarrow$ **hidden layers** $\rightarrow$ **output layer**; "deep" = several hidden layers
- **Forward propagation:** compute the prediction layer by layer
- **Training:** a **loss function** compares prediction and true value; the weights are then adjusted (backpropagation, next lecture)

Image source: Medium

Dataset

Examples from which the system learns: for each input x, there is an output (or label) y. Both input and output are usually vectors (or matrices); the output can be a single number as well.

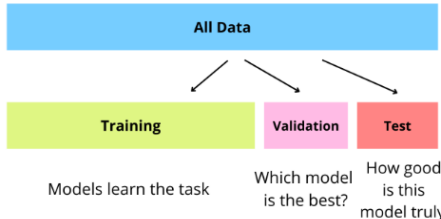
The data set should be:

- representative of the problem, diverse
- balanced: a similar number of examples per class, uniform coverage of the input range, no duplicates
- outlier-free
- normalized (e.g. input and output values between zero and one, or zero mean and unit variance)
- accurate (label noise limits the achievable accuracy)
- sufficiently large

Sometimes it is possible to **simulate** data, making it easier to increase the size and quality of the data set.

Training, validation, and test sets

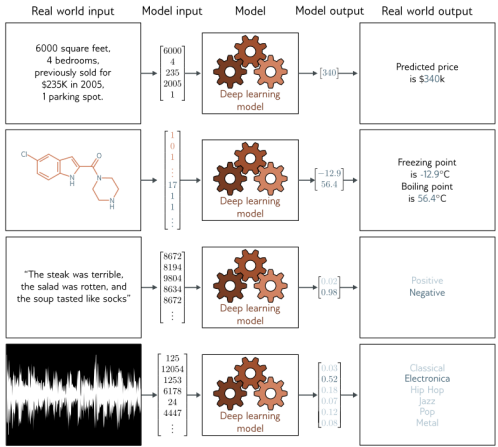
Dividing a dataset into subsets to train, tune, and evaluate the deep learning model



- Common split ratios are 60-20-20 or 70-15-15 for training-validation-test
- **Shuffle** before splitting; fit any preprocessing (e.g. normalization) on the training set only
- **Data leakage** happens when information from the validation/test sets unintentionally gets used during training $\rightarrow$ overly optimistic performance estimate

Image source: Medium

Supervised learning: two families of problems



Supervised learning: learn a mapping from inputs to outputs from labelled examples (x, y)

Regression: the output is a continuous quantity
(price, temperature, particle size, formation energy, . . .)

Classification: the output is one of a finite set of categories
(spam / not spam, which digit, which crystal structure, . . .)

The choice determines the **output activation** and the **loss function**.

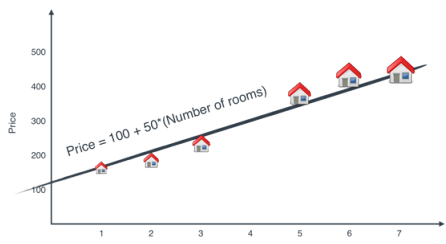
Image source: S. Prince, Understanding Deep Learning

Regression

- **Objective**: predict continuous numerical values from input features
- **Examples**:
  - house price from size, location, number of bedrooms
  - melting temperature or band gap of a material from its composition
  - particle size from a scattering curve
- **Model representation**:

$$\hat{y} = f(x; \theta)$$

where x is the input data, θ represents the model parameters (all weights and biases), and $\hat{y}$ is the predicted output.



Output activation functions for regression

The activation of the *last* layer must match the range of the quantity to predict:

- **Linear**:

$$f(z) = z$$

  Default choice, output values are unbounded.

- **Sigmoid**:

$$f(z) = \frac{1}{1 + e^{-z}}$$

  Output values constrained between 0 and 1 (e.g. a fraction or a normalized quantity).

- **ReLU** $f(z) = \max(0, z)$ or **softplus** $f(z) = \log(1 + e^z)$: output is non-negative. Softplus is smooth; ReLU as an output layer can get "stuck" at zero.

Loss functions in regression

- **Mean Squared Error (MSE)**:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

  Average squared difference between true and predicted values. Large errors are penalized strongly.

- **Mean Absolute Error (MAE)**:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

  Average absolute difference. More robust to outliers.

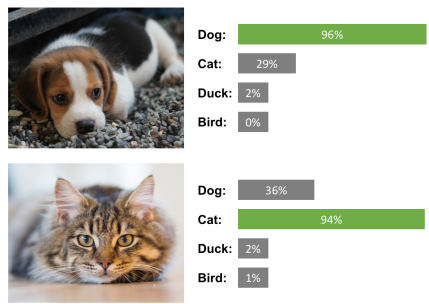
- **Purpose**: the loss is the number that training tries to minimize by adjusting the parameters θ.

Classification

- **Objective**: assign input data to predefined categories or classes
- **Types**:
  - **Binary classification**: two classes (e.g., spam vs. not spam)
  - **Multi-class classification**: more than two classes (e.g., digit recognition, 10 classes)
- **Model representation**:

$$P(y \mid x) = f(x; \theta)$$

The network outputs a **probability** for each class y given the input x; the predicted class is the most probable one.



One-hot encoding

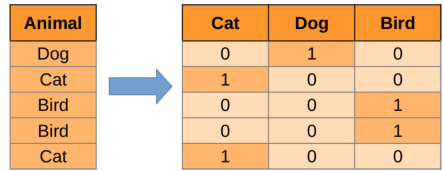
- **Definition**: converts categorical class labels into binary vectors of length K (number of classes)
- **Example**:

$$\text{Classes} : \{\text{Cat}, \text{Dog}, \text{Bird}\}$$

$$\text{Cat} \to [1, 0, 0], \quad \text{Dog} \to [0, 1, 0],$$

$$\text{Bird} \to [0, 0, 1]$$

- **Purpose**: the label gets the same shape as the network output (one probability per class); no artificial order between classes
- Libraries also accept integer labels directly ("sparse" losses, see below)



Output activation functions for classification

- **Sigmoid** (binary classification, one output):

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

  Maps any real number into $(0, 1)$: the probability of the positive class.

- **Softmax** (multi-class classification, K outputs):

$$\text{softmax}(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

  Produces a probability distribution over K classes:
  - each output is between 0 and 1
  - the outputs sum to 1
  - the largest z_j gets the largest probability (a "soft" maximum)

  For $K = 2$, softmax reduces to the sigmoid.

Surprise, entropy, cross-entropy

Surprise of an outcome with probability p: $I = -\log p$. Certain events carry no information, rare events a lot ($\log_2$: bits, ln: nats).

Entropy: average surprise under the true distribution p, $H(p) = -\sum_k p_k \log p_k$, the shortest possible average code length (Shannon, 1948). One of 10 equally likely digits: $\log 10$, i.e. 3.3 bits or 2.3 nats.

Cross-entropy: average surprise when the truth is p but we predict with a model q,

$$H(p, q) = -\sum_k p_k \log q_k \;\geq\; H(p),$$

equal only if $q = p$: minimising it pulls the model's probabilities towards the true ones.

One-hot label on class c: $H(p, \hat{y}) = -\log \hat{y}_c$, the loss on the next slide.

Prediction is compression

- A good predictor codes cheaply: an outcome costs $-\log_2 q$ bits, so the loss counts the bits the model needs to describe the data.
- Shannon (1951) estimated the entropy of English at about **1 bit per letter** by having a person guess the next letter of a text. Against 4.7 bits for 27 equally likely symbols, English is about 75 % redundant.
- **Large language models** are trained with exactly this loss: predict the next token, minimise the cross-entropy. A large model compresses text to under a tenth of its size, gzip to about a third.
- "Intelligence is compression": to understand data is to be able to predict it.

Loss functions in classification

- **Binary cross-entropy**:

$$\mathcal{L} = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)\right]$$

- **Categorical cross-entropy** ($y_{i,k}$ one-hot encoded):

$$\mathcal{L} = -\frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{K} y_{i,k}\log(\hat{y}_{i,k}) = -\frac{1}{n}\sum_{i=1}^{n}\log(\hat{y}_{i,\,c_i})$$

  Only the predicted probability of the *correct* class c_i matters. With integer labels c_i instead of one-hot vectors, the same loss is called **sparse** categorical cross-entropy.

- **Interpretation**: the model is pushed to assign a high probability to the correct class ($\hat{y}_{i,c_i} \to 1 \Rightarrow \mathcal{L} \to 0$). Sanity check: an untrained network that predicts $1/K$ for every class gives $\mathcal{L} = \log K$, the entropy of a uniform guess, i.e. $\log 10 \approx 2.30$ for 10 classes.

Metrics: accuracy and the confusion matrix

The **loss** is what the optimizer minimizes; a **metric** is what we report because it is easier to interpret.

- **Accuracy**: fraction of examples whose predicted class (largest probability) equals the true class
- Not differentiable and blind to confidence, hence unsuitable as a loss
- Misleading for **imbalanced** classes: if 99 % are "healthy", always predicting "healthy" gives 99 % accuracy
- **Confusion matrix**: which classes get mixed up with which? Row = true class, column = predicted class

	predicted		
true	3	5	8
3	**960**	30	20
5	25	**940**	27
8	18	22	**934**

Toy example: the diagonal counts correct predictions; off-diagonal entries reveal systematic confusions (3 vs. 5, 5 vs. 8).

Application: MNIST dataset

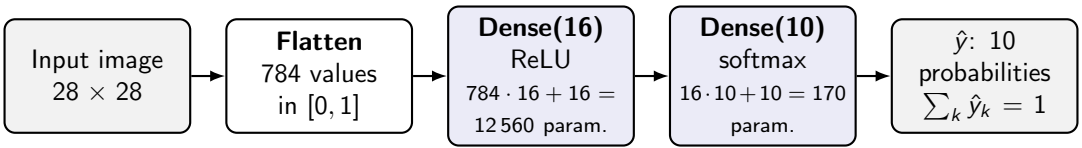


- 70 000 labelled images of handwritten digits (LeCun et al., 1998)
- 10 classes (0–9), about 7 000 images per class
- each image: 28×28 pixels, grayscale (0–255)
- classic benchmark: the "hello world" of deep learning

Objective: sort images into a finite number of predefined categories $\rightarrow$ **multi-class classification**

Image source: LeCun et al., Proc. IEEE 86, 2278 (1998)

Putting it together: a first network for MNIST



- **Input**: the image is reshaped into a vector, pixel values are scaled to $[0, 1]$
- **Hidden layer**: 16 neurons with ReLU activation (later: more neurons, more layers)
- **Output layer**: one neuron per class, softmax activation
- **Loss**: (sparse) categorical cross-entropy; **metric**: accuracy
- Compare: the previous slide's confusion matrix tells us *which* digits this network confuses

First attempt with TensorFlow/Keras

```
from tensorflow import keras
from tensorflow.keras import layers

model = keras.Sequential([
    layers.Input(shape=(28, 28)),
    layers.Flatten(),
    layers.Dense(16, activation="relu"),
    layers.Dense(10, activation="softmax"),
])
model.compile(
    optimizer="adam",
    loss="sparse_categorical_crossentropy",
    metrics=["accuracy"])
history = model.fit(
    x_train, y_train, epochs=20,
    validation_data=(x_val, y_val))
model.evaluate(x_test, y_test)
```

Things to consider:

- Images are reshaped into vectors
- Grayscale values scaled to $[0, 1]$
- Labels: integers 0–9 (sparse loss) or one-hot vectors (categorical loss)
- An **epoch** is one pass over the whole training set
- Watch training *and* validation loss (problem sheet 1)

→ **Demo**: notebook on the course website

Summary and outlook

Today:

- Deep learning = machine learning with multi-layered artificial neural networks; the model is learned from data
- A neuron computes $\phi(\sum_i w_i x_i + b)$; a network stacks layers of neurons with **nonlinear** activation functions
- The data set and its split into training / validation / test sets decide what can be learned and how honestly it is measured
- Regression vs. classification $\rightarrow$ output activation (linear / sigmoid / softmax) and loss (MSE / cross-entropy)

Open question: how are the weights and biases actually found?
$\rightarrow$ Lecture 2: gradient descent and backpropagation

Next session, practical 1: build, train and analyze the MNIST classifier of the last slides.
Problem sheet and notebook on `daniel-forster.fr/dl`