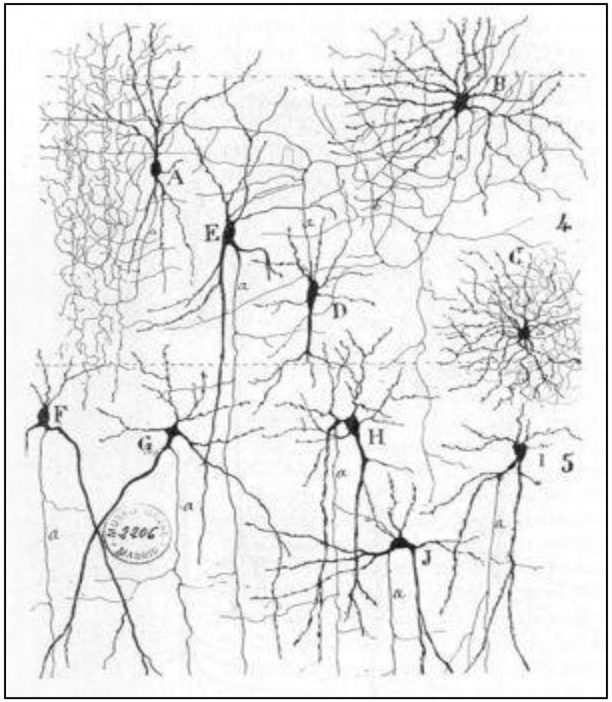
An Introduction to Neural Networks

The origin: To make a computer "think" and "learn", maybe we should start by trying to understand how the human brain think and learn.



Connectionist model: Complicated functions can be described by a set of interconnected simple units.

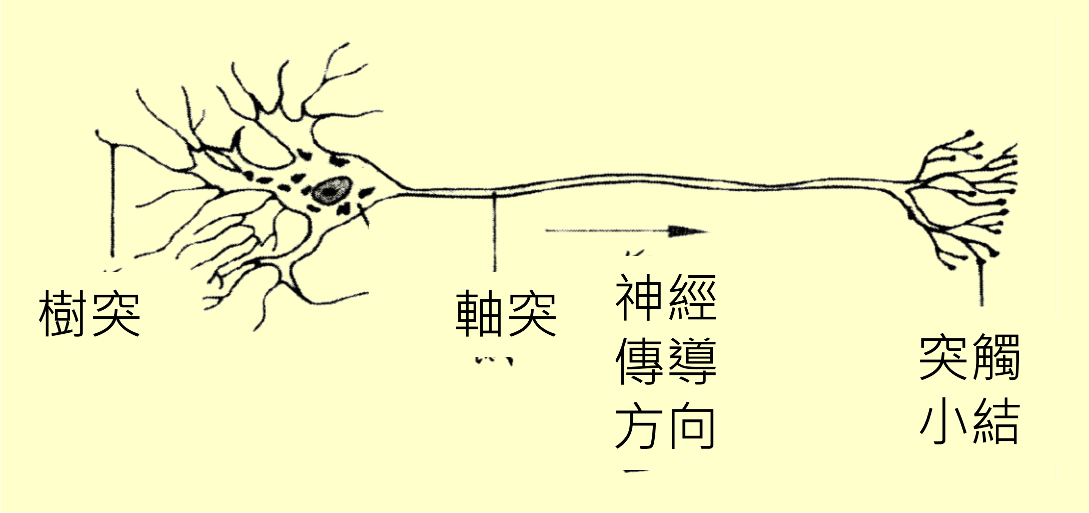
From "Texture of the Nervous System of Man and the Vertebrates" by Santiago Ramón y Cajal. The figure illustrates the diversity of neuronal morphologies in the auditory cortex.

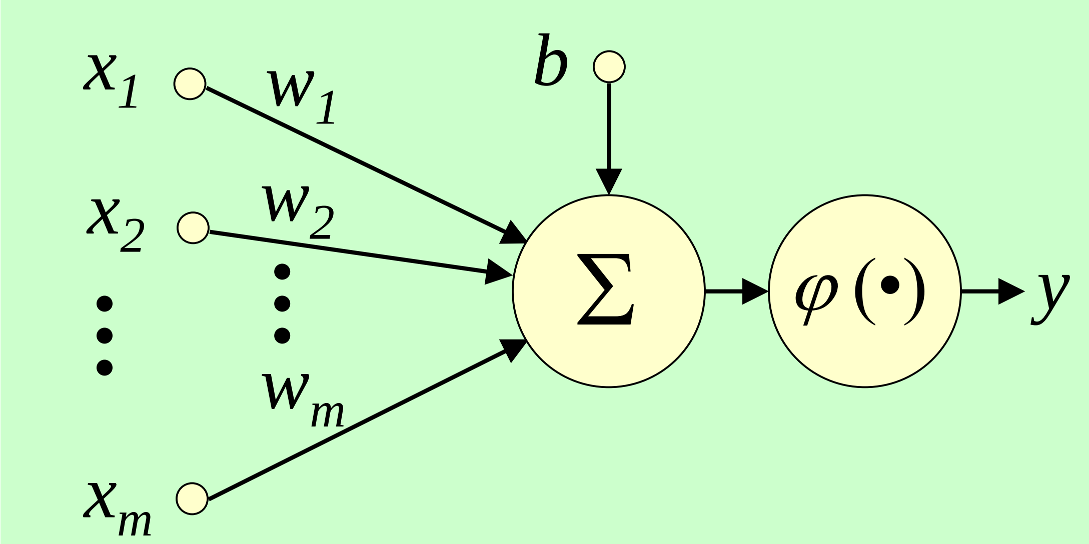
This is a model that spans cognitive psychology, neuroscience, philosophy, and computer science.

Biological vs. Artificial Neurons

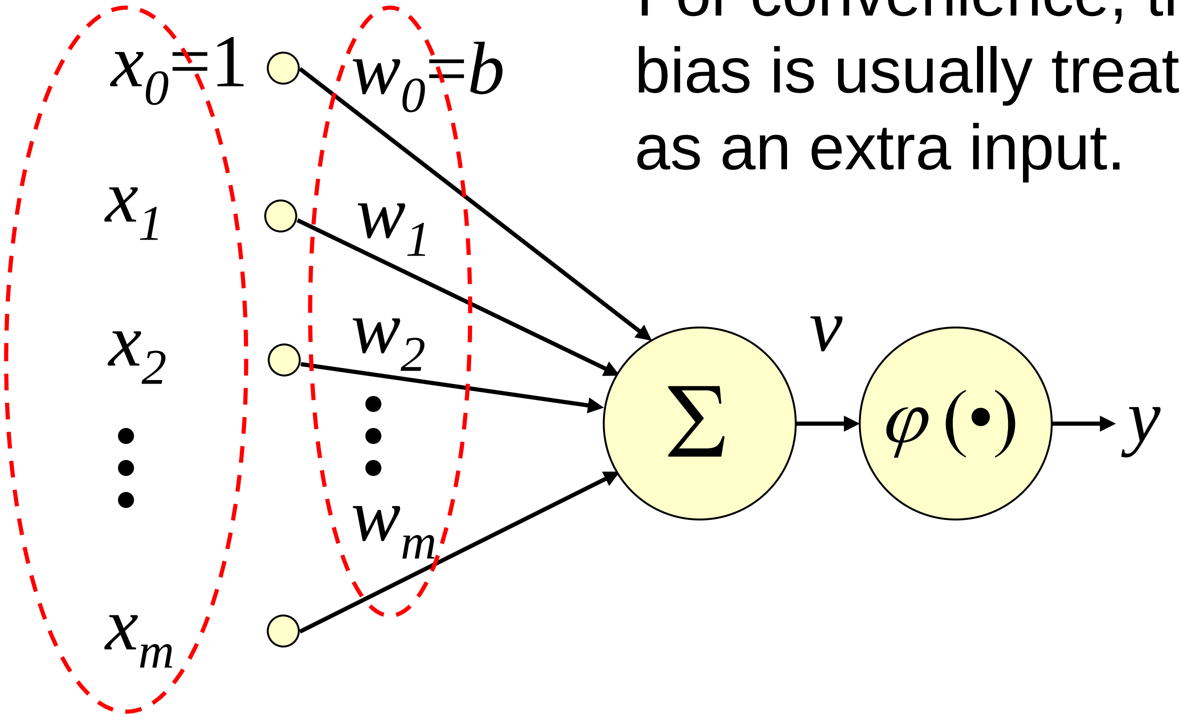
Important components:

- Inputs
- Synaptic weights
- Bias
- Summing function
- Activation function
- Output





Important Notations



For convenience, the bias is usually treated as an extra input.

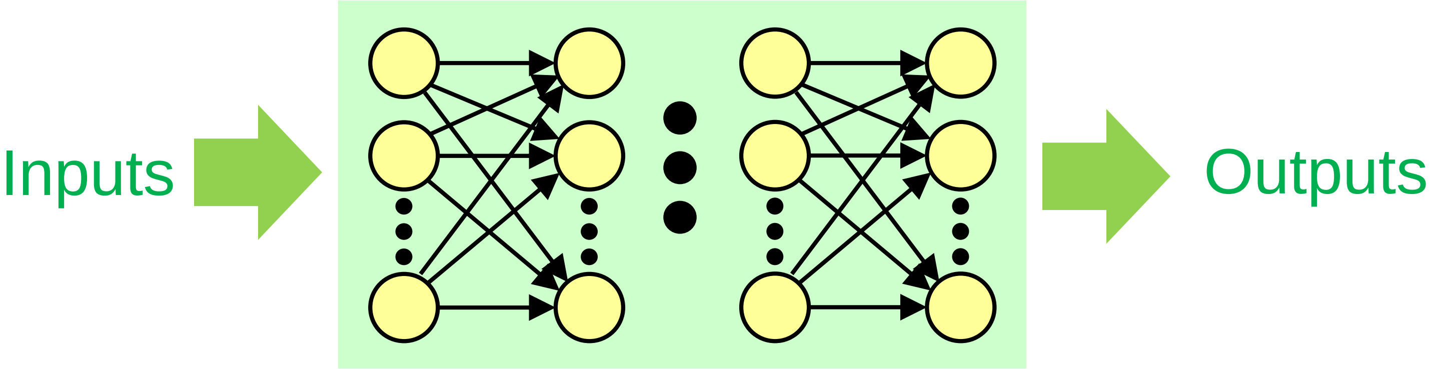
$\boldsymbol{x}$: The (column) vector of all the inputs

$\boldsymbol{w}$: The (column) vector of all the synaptic weights of a neuron

The combined input: $v = \boldsymbol{w}^T \boldsymbol{x}$

Neural Network Architectures

An artificial neural network is made of a directed graph of a set of neurons:



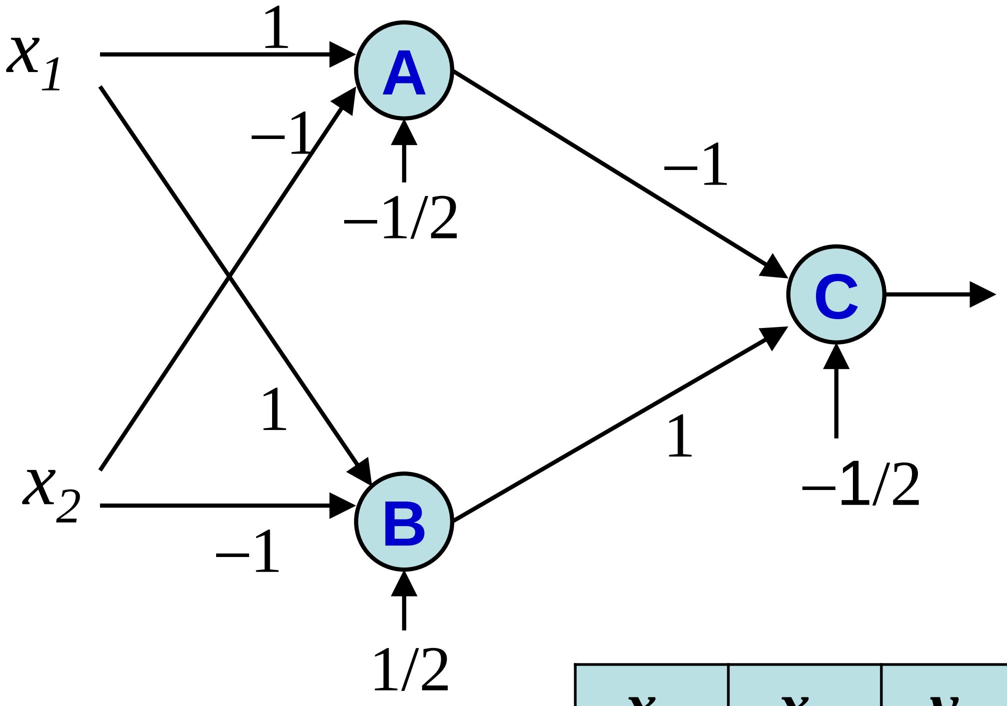
Inputs

Outputs

Two main categories of neural network architectures:

- Feedforward NNs: Signals flow in one direction only.
- Recurrent NNs: Both feedforward and feedback links ➔ Have the effect of "state memory" ➔ More suitable for processing sequence/dynamic data. (Examples: language, speech, handwriting, etc.)

A Simple Feedforward NN



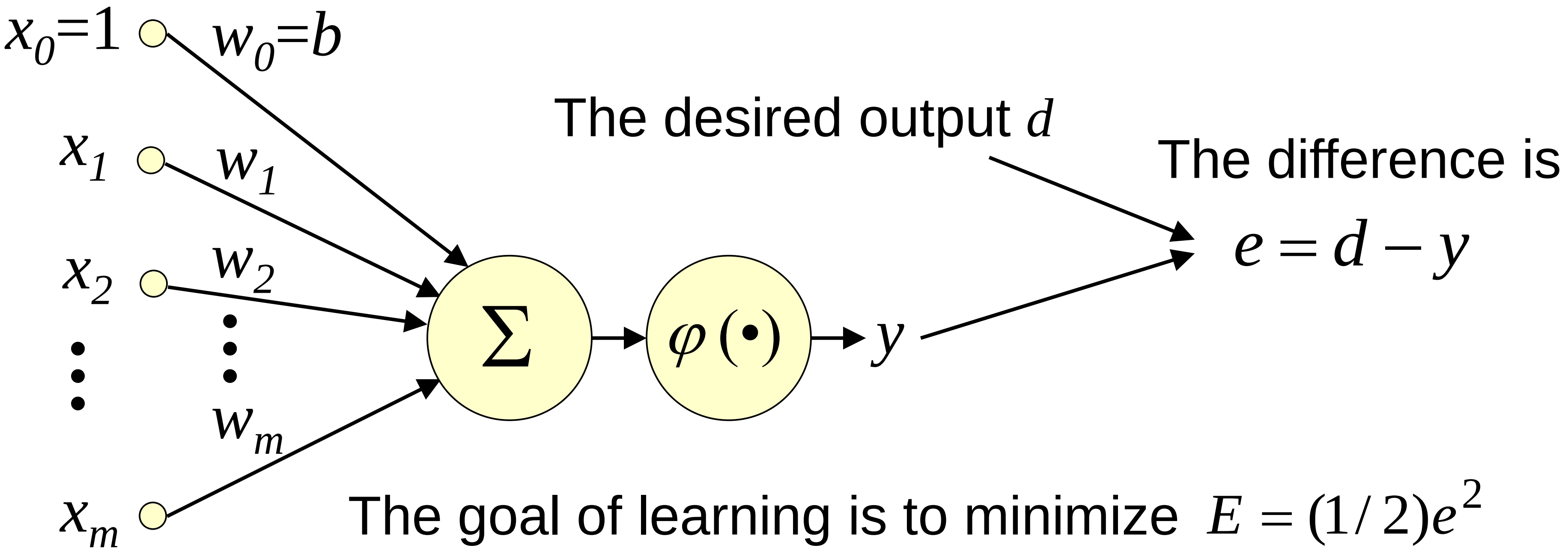
Activation function:

$$\varphi(v) = \begin{cases} 1 & \text{for } v > 0 \\ 0 & \text{otherwise} \end{cases}$$

x_1	x_2	v_A	y_A	v_B	y_B	v_C	y_C
0	0	–1/2	0	1/2	1	1/2	1
1	0	1/2	1	3/2	1	–1/2	0
0	1	–3/2	0	–1/2	0	–1/2	0
1	1	–1/2	0	1/2	1	1/2	1

Error-Correction Learning

This is an example with a single neuron:



The desired output d

The difference is $e = d - y$

The goal of learning is to minimize $E = (1/2)e^2$ by adjusting $\boldsymbol{w}$. ($\boldsymbol{w}$ is the vector of all the weights.)

This is a typical optimization problem.

In a typical 2-class classification problem, we just use two different values as the desired outputs. (Most commonly 0 and 1, or -1 and 1)

Error-Correction Learning: Example

Assume a linear activation function such that

$$y = \varphi(v) = v = \mathbf{w}^T \mathbf{x}$$

The error is then $E = (1/2)e^2 = (1/2)(d - \mathbf{w}^T \mathbf{x})^2$

The gradient of the error relative to $\mathbf{w}$ is $\dfrac{\partial E}{\partial \mathbf{w}} = -e\,\mathbf{x}$

Gradient-descent learning:

$$\mathbf{w}(t+1) = \mathbf{w}(t) - \underset{\uparrow}{\boxed{\eta}} \frac{\partial E}{\partial \mathbf{w}} = \mathbf{w}(t) + \eta e\,\mathbf{x}$$

Learning rate: The step size during local search.

Perceptron Algorithm

Specifically designed for classification problems, this famous **perceptron algorithm** only updates the weights when there are classification errors:

$$\boldsymbol{w}(t+1) = \begin{cases} \boldsymbol{w}(t) + \eta_t \boldsymbol{x}(t) & \text{for } \boldsymbol{w}(t)^T \boldsymbol{x}(t) \leq 0 \text{ and } \boldsymbol{x}(t) \in \omega_1 \\ \boldsymbol{w}(t) - \eta_t \boldsymbol{x}(t) & \text{for } \boldsymbol{w}(t)^T \boldsymbol{x}(t) \geq 0 \text{ and } \boldsymbol{x}(t) \in \omega_2 \\ \boldsymbol{w}(t) & \text{otherwise} \end{cases}$$

with learning rate $\eta_t > 0$

Its success brought up a hype for neural networks because it can **learn** a classification task from training data.

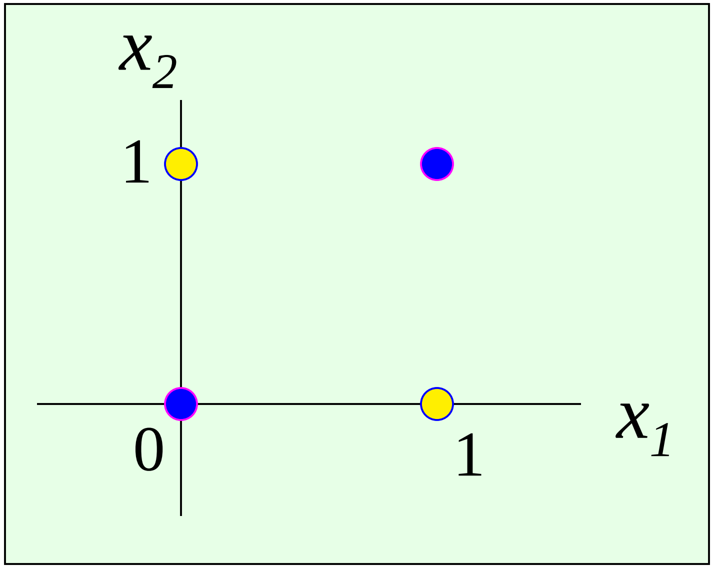
The XOR Problem

However, with a single neuron, the perceptron algorithm can not handle cases where the classes are not linearly separable.

➔ People gave up on neural networks.

An important example of a very simple problem that is not linearly separable.

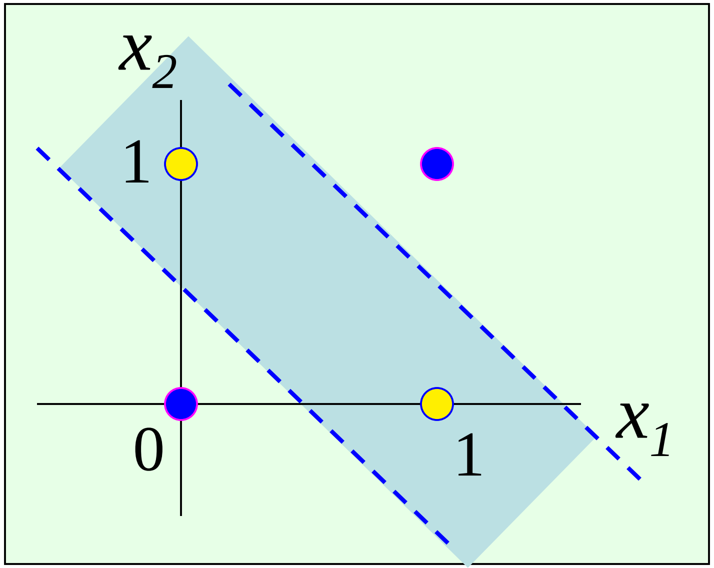
input (***x***)	output (*y*)
(0, 0)	0 (class 2)
(0, 1)	1 (class 1)
(1, 0)	1 (class 1)
(1, 1)	0 (class 2)



The feature space of the XOR problem.

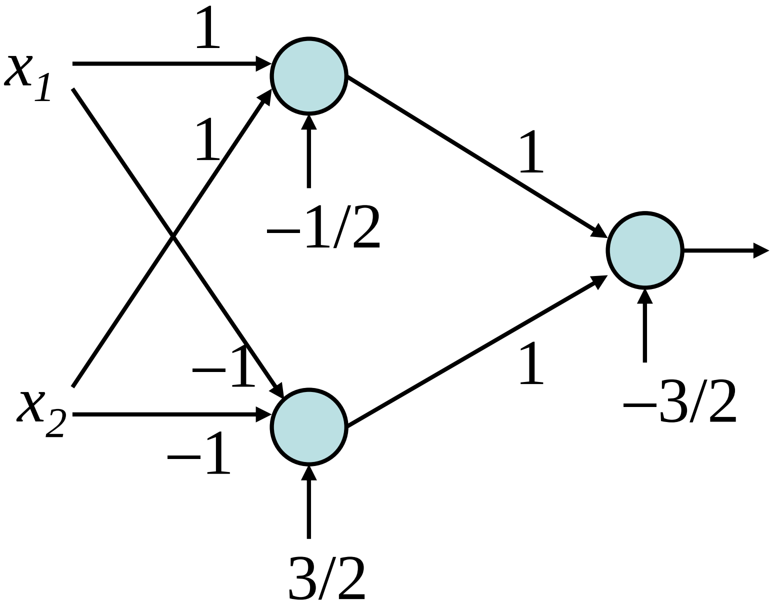
The XOR Problem

However, we can get exact classification if our decision boundary includes two lines:



We can AND the regions of OR and NAND operators as the region of XOR.

This results in a two-layer perceptron.



Hidden Layer

Output Layer

The problem: How to learn the weights here?

Credit Assignment Problem

There are many components of $\boldsymbol{w}$. The problem is, how do we know how much to change each component according to the current performance?

We know how to do this in the single-layer perceptron case.

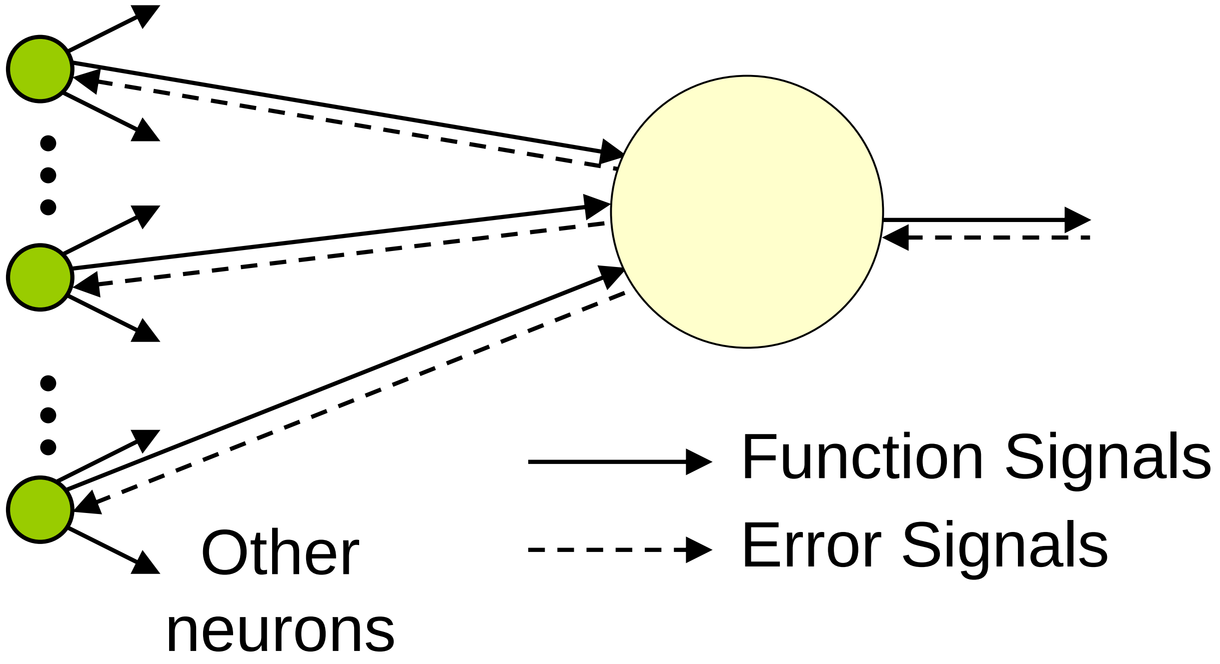
However, this is not trivial for MLP because it is too difficult to actually differentiate J relative to each component of $\boldsymbol{w}$ directly.

Back-Propagation

What we can do is layer-by-layer processing:

- Only update the weights from the neurons in the previous layer.
- Propagate the error to the individual neurons in the previous layer.

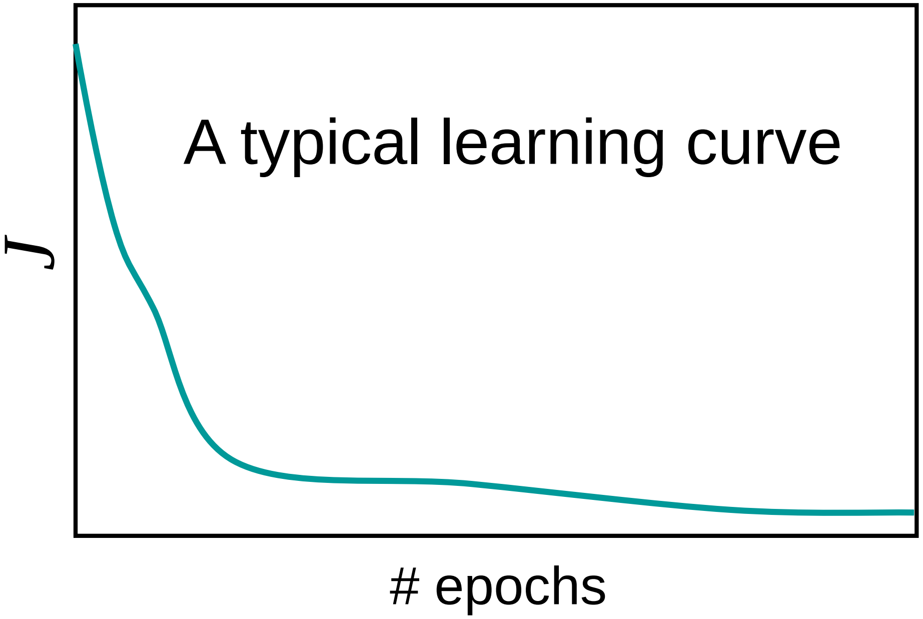
Signal Flow:



Learning Curve

Learning curve: The curve of J (the total error) versus the number of epochs during iterative training.

- An epoch used to represent a training iteration in which all the training samples are presented once.
- Nowadays, with the training set getting huge, one can define an epoch as the processing of a fixed amount of training data.



Issues in Training

There are many issues in designing and training a NN:

- Stopping criteria
- Non-linearity (through activation functions)
- Speed of convergence:
  - Normalization (data normalization, batch normalization)
  - Learning rate adjustment
  - Momentum
- Generalization (performance for data not used for training)
  - Stochastic training / dropout
  - Regularization
  - Early stopping

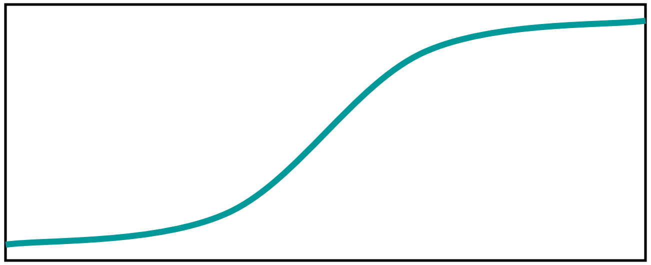
Stopping Criteria

- Limit the number of epochs.
- Check for signs of convergence:
  - Stop when the cost function J is lower than a given threshold.
  - Stop when the gradient of the cost function $\partial J / \partial \mathbf{w}$ is lower than a given threshold.
  - Stop when the reduction of J over a given number of epochs is below a given ratio.
- Early stopping (discussed later).

Nonlinear Activation Functions

For a neuron, the activation function is the only place to be nonlinear. Typical activations are:

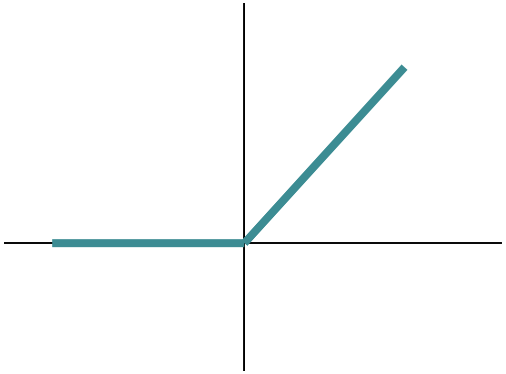
- Sigmoid/logistic and tanh:
  - The old standard
  - Continuously differential everywhere
  - Problem: Vanishing gradients



$$f(v) = \frac{1}{1 + \exp(-av)}$$

$$f(v) = \tanh(av)$$

- RELU (rectified linear units):
  - The new standard for deep NNs
  - Some variants
  - Nonlinear with no vanishing gradients



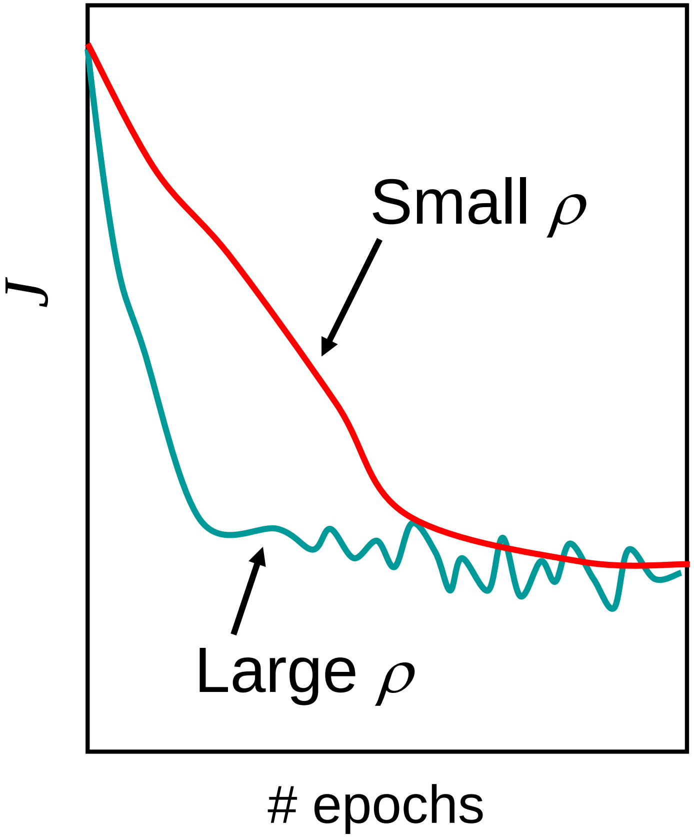
Learning Rate Adjustments

Large learning rate:

- Faster training (reaches the local maxima of J in fewer epochs)
- More oscillation

Small learning rate:

- Slower training
- Less oscillation (smoother approach to the local maxima of J)



A typical approach is to start with a larger learning rate (e.g., 0.1; smaller for DNNs) and gradually decrease it.

Momentum

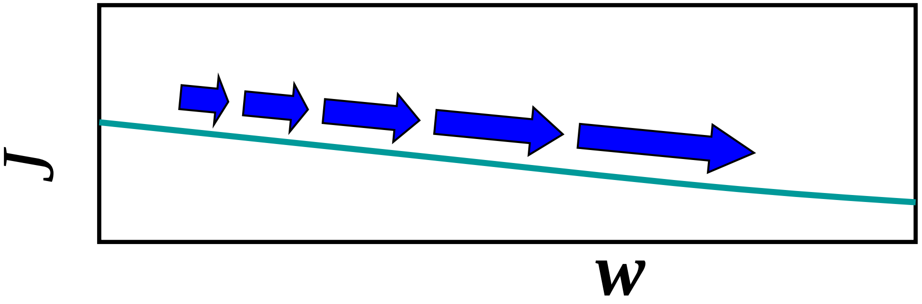
Momentum term is a popular technique for speeding up training and avoiding oscillation.

$0 \le \alpha < 1$

$$\mathbf{w}(t+1)=\mathbf{w}(t)-\rho\frac{\partial J}{\partial \mathbf{w}}+\alpha[\mathbf{w}(t)-\mathbf{w}(t-1)]$$

momentum term

The effect of momentum in a plateau region:



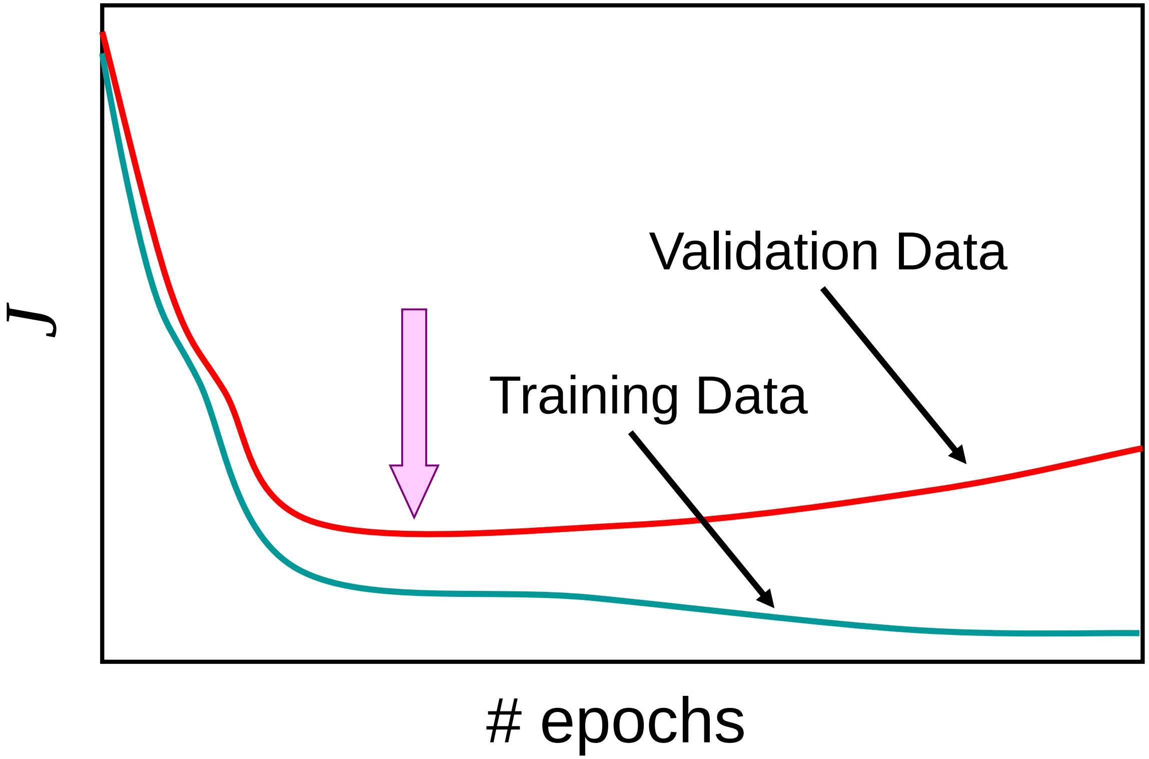
Effects of momentum:

- Acceleration: The update is larger when the gradients are in similar directions.
- Stabilization: The update is smaller when the gradients have opposite directions.

Early Stopping

Validation data: labeled data not used directly for training, but for validating the generalization ability of a classifier and the selection of classifier parameters.

We can stop the training at the minimum of J for the validation set.



Typical learning curves for the training and validation sets.

Deep Neural Networks

- Before DNNs became popular, most applications use networks with just one or two hidden layers.
- Several reasons:
    - The complexity and danger of overfitting for larger networks.
    - Theoretically, more layers do not increase the *expressive power* (the ability of modeling a given mapping) of a neural network. (Universal Approximator Theorem)

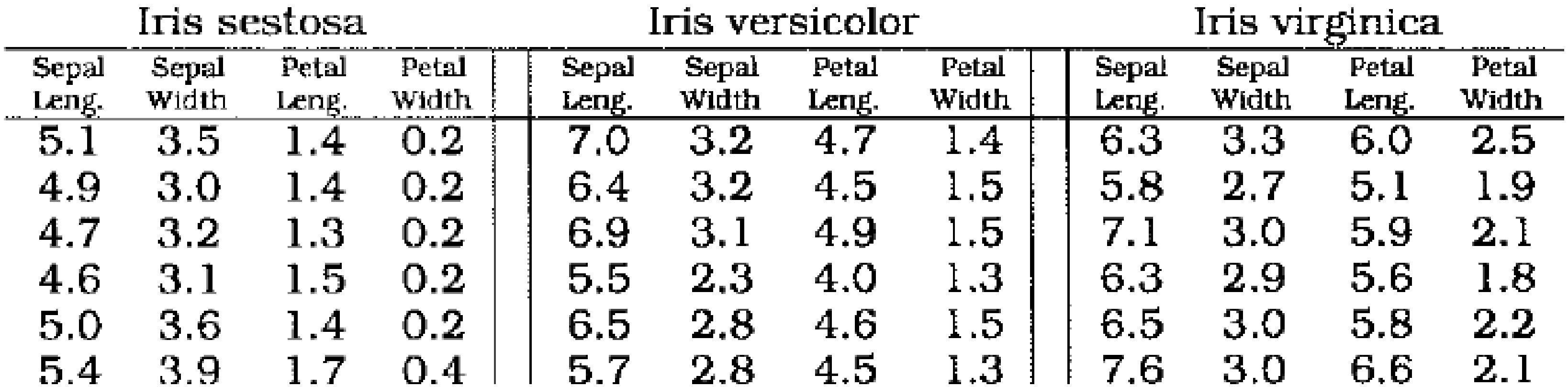
A Look at Features

- To understand why DNNs, especially CNNs (convolutional neural networks), become popular, we need to consider a very important part of supervised learning: **Features**.
- In general, we represent a sample with a set of features (or attributes; can be numerical or categorical) for a given problem.

A Look at Features

Using the very famous Iris dataset as an example:

- Data: Each sample represents an iris flower.
- Objective: Classify each sample into one of three iris varieties.
- Four features for each flower: sepal length, sepal width, petal length, petal width.



Iris sestosa				Iris versicolor				Iris virginica			
Sepal Leng.	Sepal Width	Petal Leng.	Petal Width	Sepal Leng.	Sepal Width	Petal Leng.	Petal Width	Sepal Leng.	Sepal Width	Petal Leng.	Petal Width
5.1	3.5	1.4	0.2	7.0	3.2	4.7	1.4	6.3	3.3	6.0	2.5
4.9	3.0	1.4	0.2	6.4	3.2	4.5	1.5	5.8	2.7	5.1	1.9
4.7	3.2	1.3	0.2	6.9	3.1	4.9	1.5	7.1	3.0	5.9	2.1
4.6	3.1	1.5	0.2	5.5	2.3	4.0	1.3	6.3	2.9	5.6	1.8
5.0	3.6	1.4	0.2	6.5	2.8	4.6	1.5	6.5	3.0	5.8	2.2
5.4	3.9	1.7	0.4	5.7	2.8	4.5	1.3	7.6	3.0	6.6	2.1

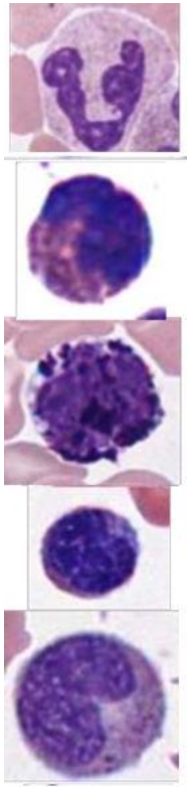
and more …

A Look at Features

- Some features are from direct measurements, or are otherwise straightforward.
  - Features for Iris
  - A person's age, gender, etc.
- For many problems, the "raw" features are difficult to use. Examples:
  - Images
  - Speech; audio signals
  - Trajectory, such as online handwriting recognition
  - Text
  - ...

A Look at Features

- For problems where it is impractical to use the "raw features", we need to find ways to extract meaningful and manageable "derived features" from the raw features, and then use these derived features for our classification / regression tasks.
- Example:



size (area)
ratio of nuclei
shape features

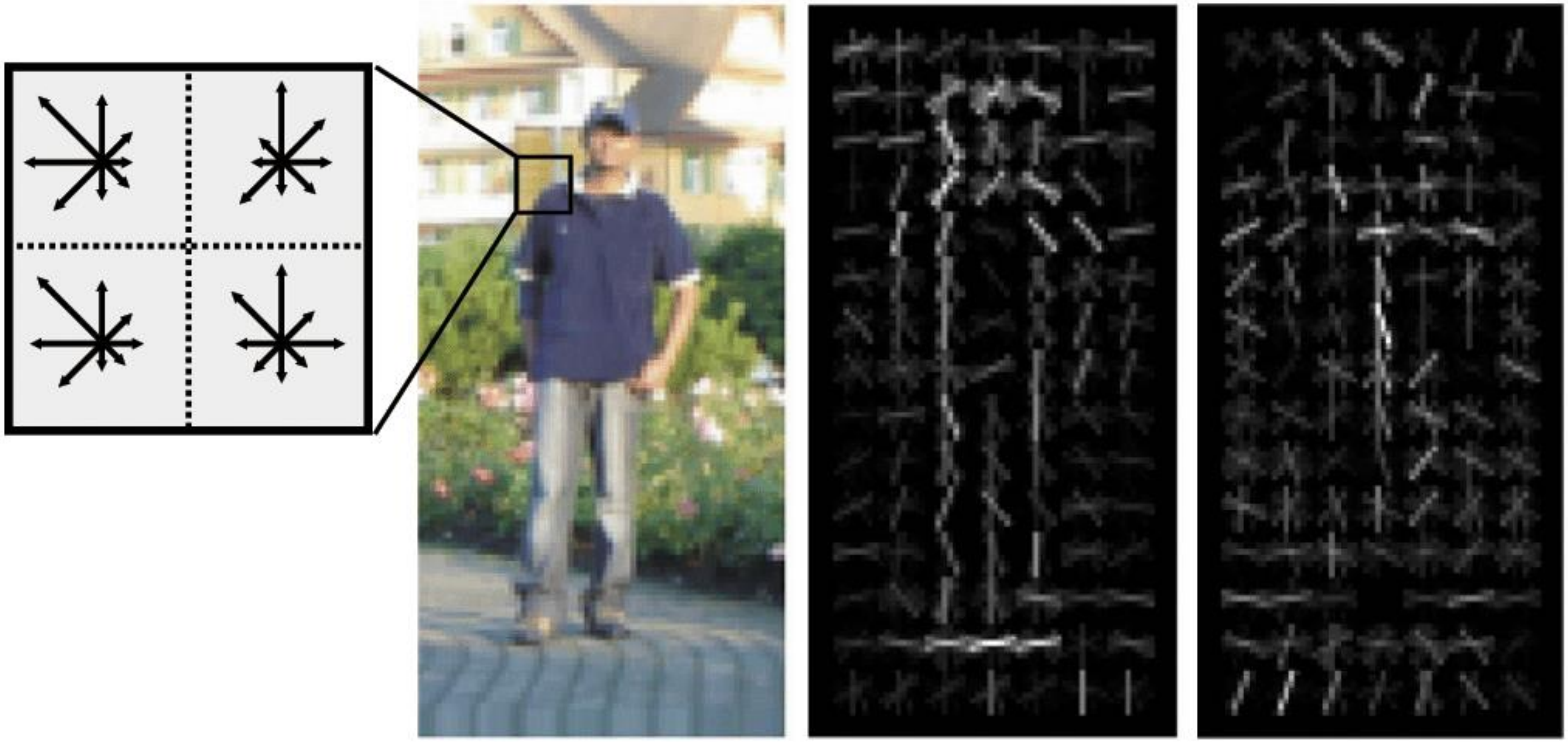
- moments
- etc.

texture features

- DCT
- etc.

A Look at Features

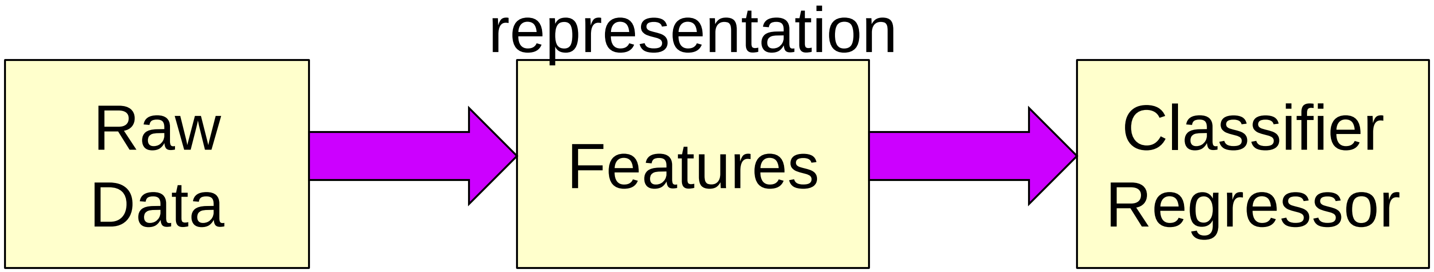
- Another classic example: The HOG (histogram of oriented gradients) features for pedestrian detection. This was the state-of-the-art before CNNs became practical.



http://doi.ieeecomputersociety.org/10.1109/TPAMI.2009.122

A Look at Features

- The standard approach:



- For each type of data, there are numerous types of "derived features".
- Good features are usually more important than good classifiers/regressors for solving a particular problem. Examples:
  - Haar features for face detection
  - HOG for pedestrian detection
  - Bag of Words for text classification

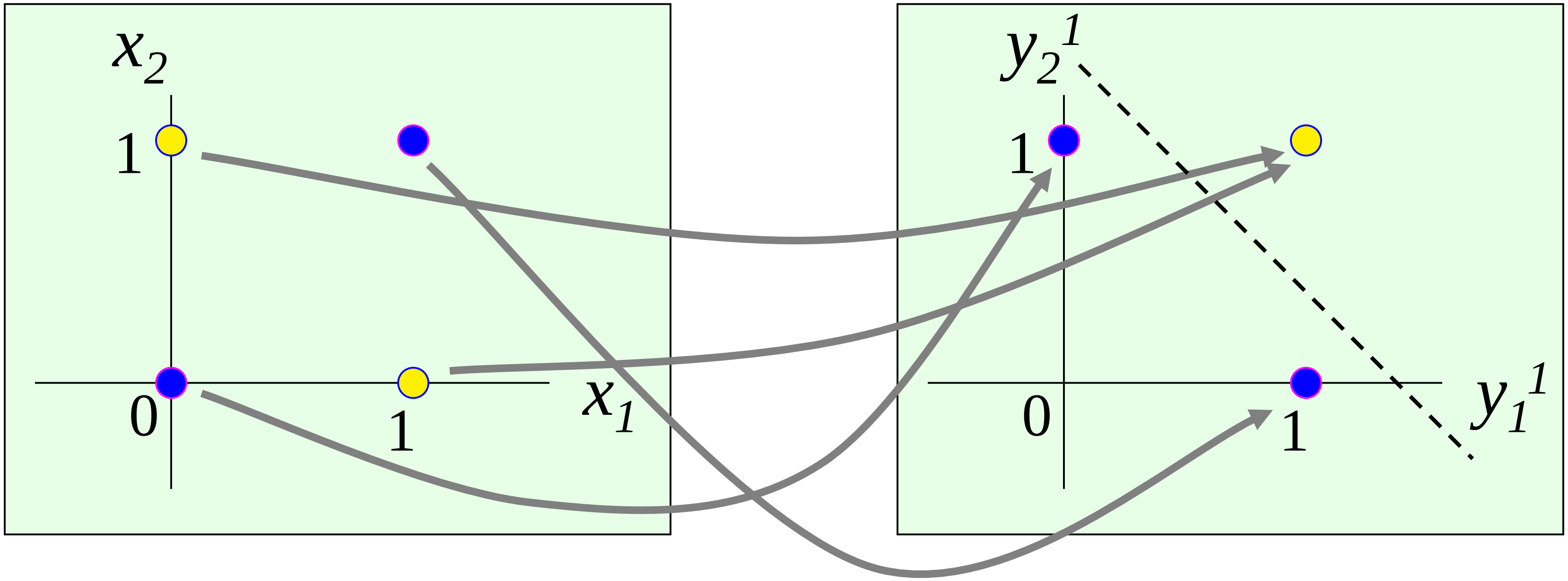
A Look at Features

- However, even good hand-crafted features are never good enough.
  - They can not capture all the useful information.
  - They can only represent the low-level information (easier to define and compute) well.
  - It is very difficult to design features for high-level and semantically rich information.
    - To understand this, consider the HOG features for pedestrian detection. They do not provide us with information regarding whether and where the head, arms, or legs are detected.

Neural Networks as Feature Extractors

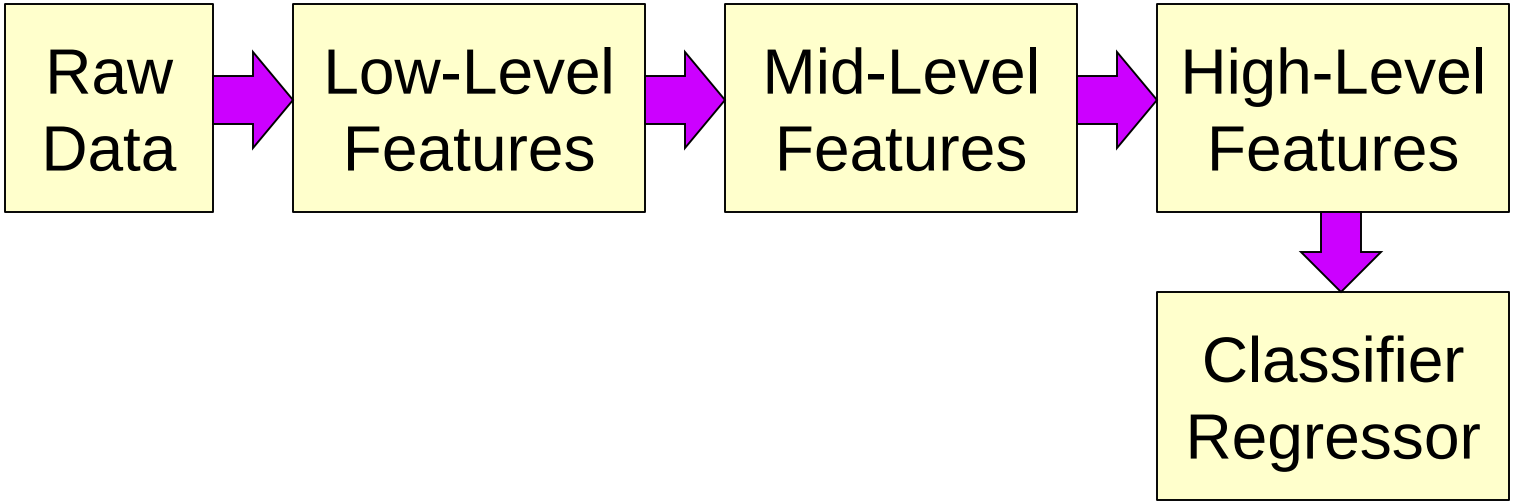
It is well known that the outputs of the hidden neurons can be considered as derived features that make the task easier.

Here is the mapping by the two hidden neurons of our NN for the XOR problem:



Now the two classes are linearly separable, and can be classified by the output neuron.

Neural Networks as Feature Extractors



Raw Data

Low-Level Features

Mid-Level Features

High-Level Features

Classifier Regressor

This is how our brain processes visual inputs.

Yes, this is quite DEEP.

Convolutional Neural Networks

Let's assume the use of image data here:

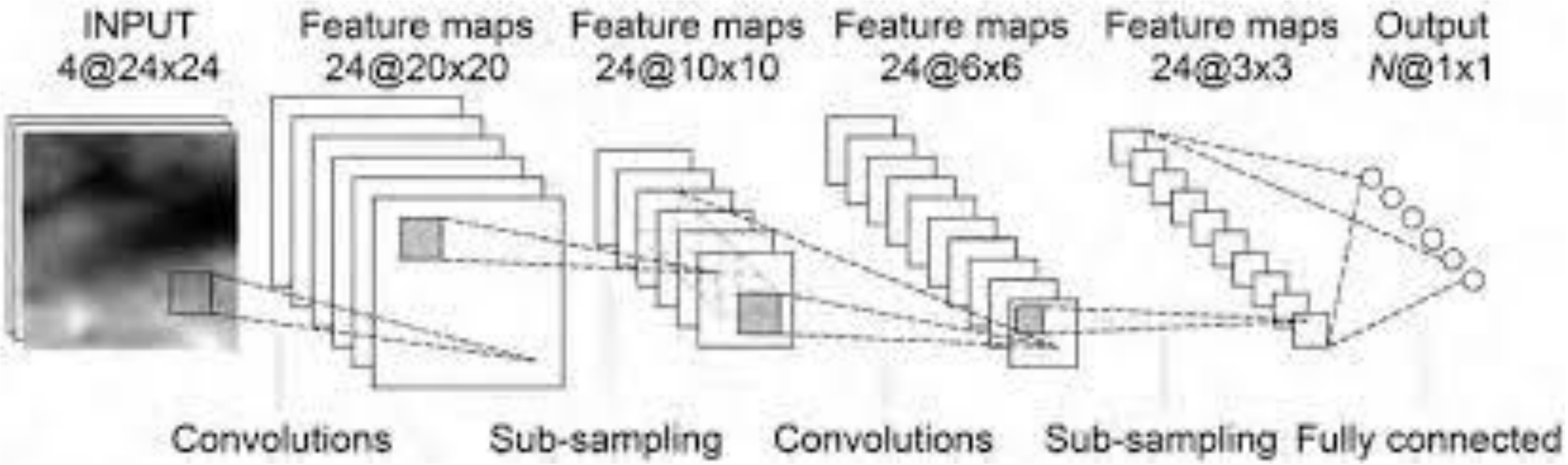
- The "convolution" operation: Compute a feature for a pixel by applying a "weighted sum" of the pixel values in its neighborhood.
- The weights represent a "mask" or "kernel".
- The same kernel is applied to all the pixels. (Example: Gradient is one of the most common kernel.)
    - This is sometimes called weight sharing, and it helps to limit the overall network complexity even if we have numerous nodes (to represent the results for all the pixels.)
- There can be many kernels used ➔ many features.
- The kernel weights can be trained. (Yes, they are the weights of the CNN.)

Convolutional Neural Networks

- The convolution operation generates the features.
    - They are arranged in a grid, just like the original image. So the outputs from a whole layer is sometimes called a feature map.
- No hand-crafting of features needed; the learning process optimizes the kernel weights for the given task.
- The convolutional layers can be stacked to form feature maps of different levels:

Convolutional Neural Networks

A small (but typical) CNN example for image classification:



- The last part (fully connected layer) is actually the classifier. It is just like a traditional multi-layer perceptron.
- The CNN can also work with other classifiers (e.g., SVMs), but using FC layers is common as the training can be done end-to-end.