Entreposage et fouille de données (STA211)
Neural Networks and Deep Learning

Nicolas Thome
Conservatoire Nationnal des Arts et Métiers (CNAM)
Laboratoire CEDRIC - équipe Vertigo

le cnam

Cédric

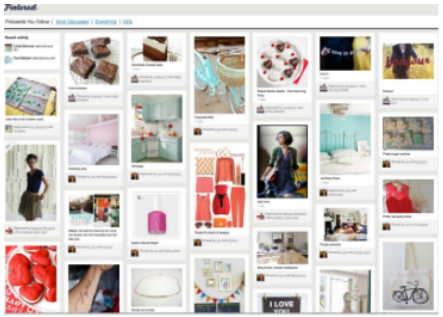
Outline

Context

Big Data

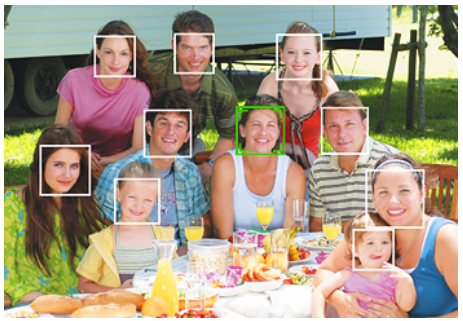
- Superabundance of data: images, videos, audio, text, use traces, *etc*

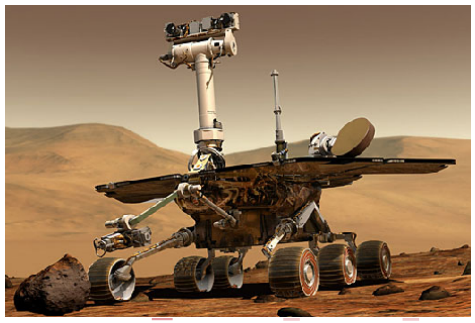
BBC: 2.4M videos


Facebook: 350B images
1B each day


100M monitoring cameras

- Obvious need to access, search, or classify these data: **Recognition**
- Huge number of applications: mobile visual search, robotics, autonomous driving, augmented reality, medical imaging *etc*
- Leading track in major ML/CV conferences during the last decade





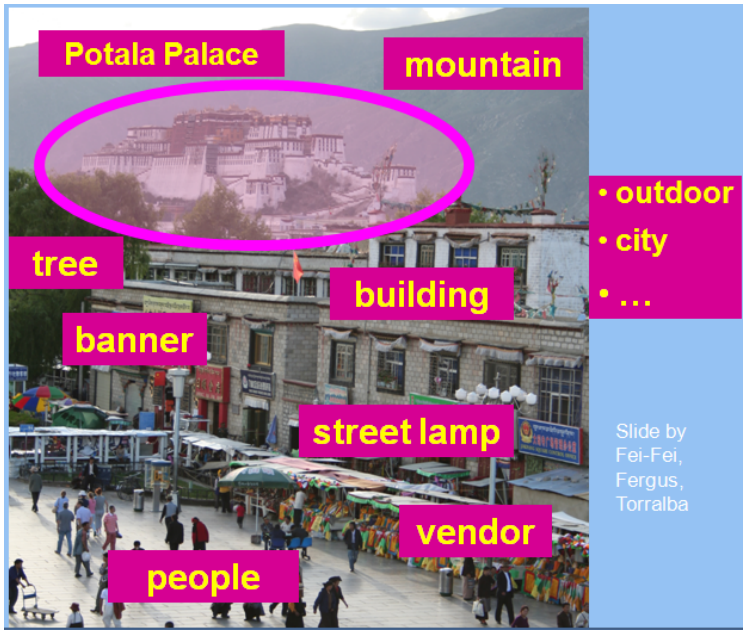
Recognition and classification

- Classification : assign a given data to a given set of pre-defined classes
- Recognition much more general than classification, *e.g.*
  - Ranking for document indexing
  - Localization, segmentation for image understanding
  - Sequence prediction for text, speech, audio, *etc*
- Many tasks can be cast as classification problems
  $\Rightarrow$ **importance of classification**

Focus on Visual Recognition: Perceiving Visual World

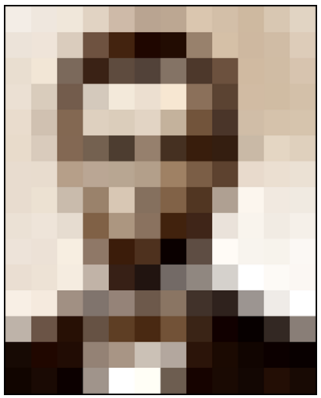
- Visual Recognition: archetype of low-level signal understanding
- Supposed to be a master class problem in the early 80's
- Certainly the most impacted topic by deep learning

- Scene categorization
- Object localization
- Context & Attribute recognition
- Rough 3D layout, depth ordering
- Rich description of scene, *e.g.* sentences



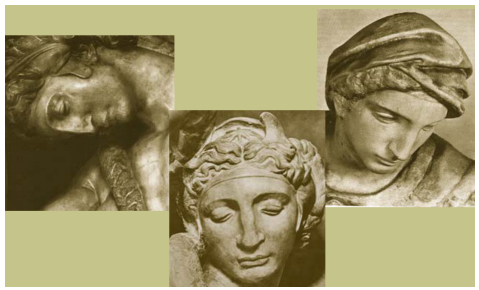
Recognition of low-level signals

Challenge: filling the semantic gap



What we perceive *vs*
What a computer sees

243	239	240	225	206	185	188	218	211	206	216	225
242	239	218	110	67	31	34	152	213	206	208	221
243	242	123	58	94	82	132	77	108	208	208	215
235	217	115	212	243	236	247	139	91	209	208	211
233	208	131	222	219	226	196	114	74	208	213	214
232	217	131	116	77	150	69	56	52	201	228	223
232	232	182	186	184	179	159	123	93	232	235	235
232	236	201	154	216	133	129	81	175	252	241	240
235	238	230	128	172	138	65	63	234	249	241	245
237	236	247	143	59	78	10	94	255	248	247	251
234	237	245	193	55	33	115	144	213	255	253	251
248	245	161	128	149	109	138	65	47	156	239	255
190	107	39	102	94	73	114	58	17	7	51	137
23	32	33	148	168	203	179	43	27	17	12	8
17	26	12	160	255	255	109	22	26	19	35	24

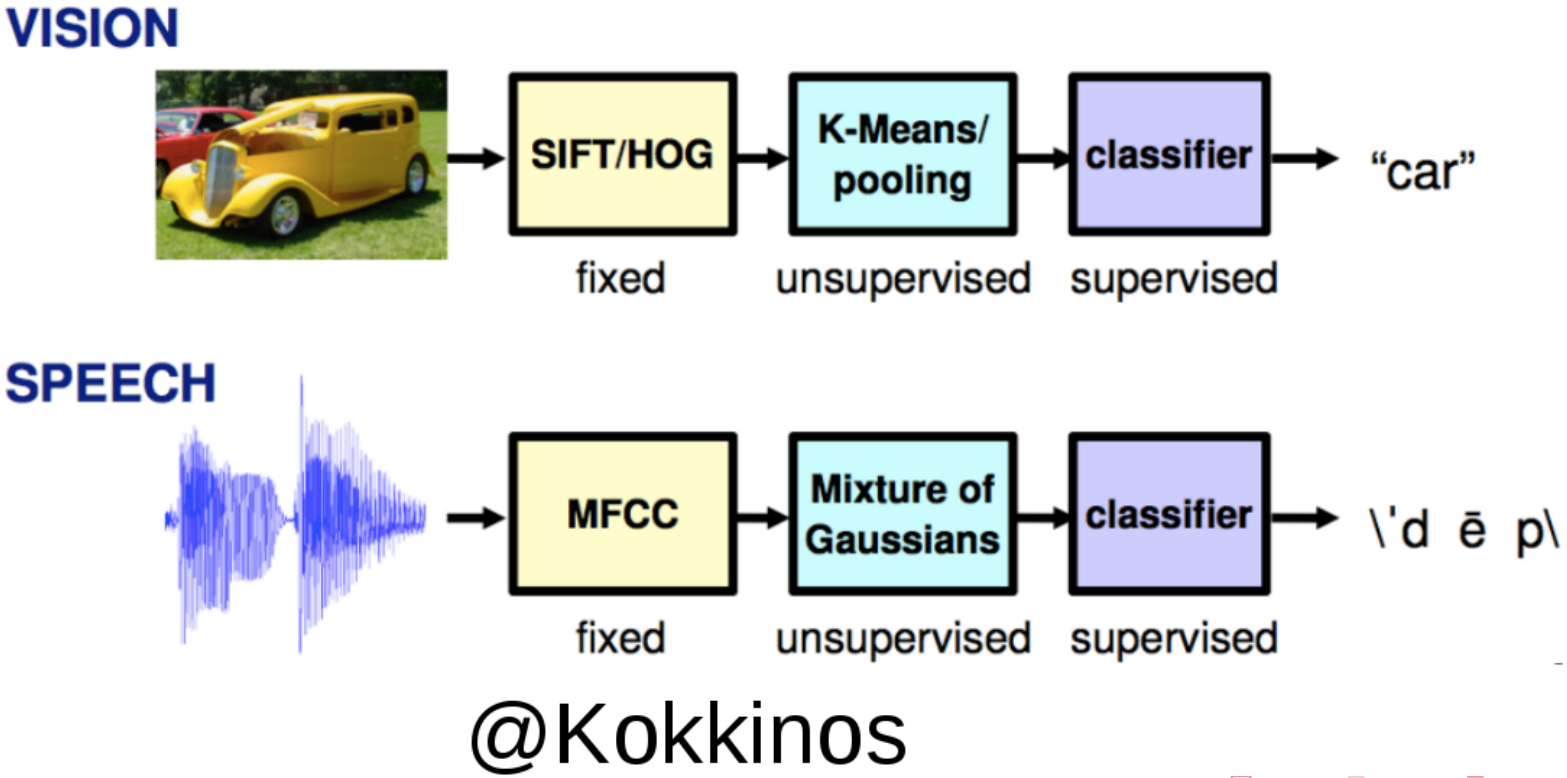


- Illumination variations
- View-point variations
- Deformable objects
- intra-class variance
- *etc*

⇒ **How to design "good" intermediate representation ?**

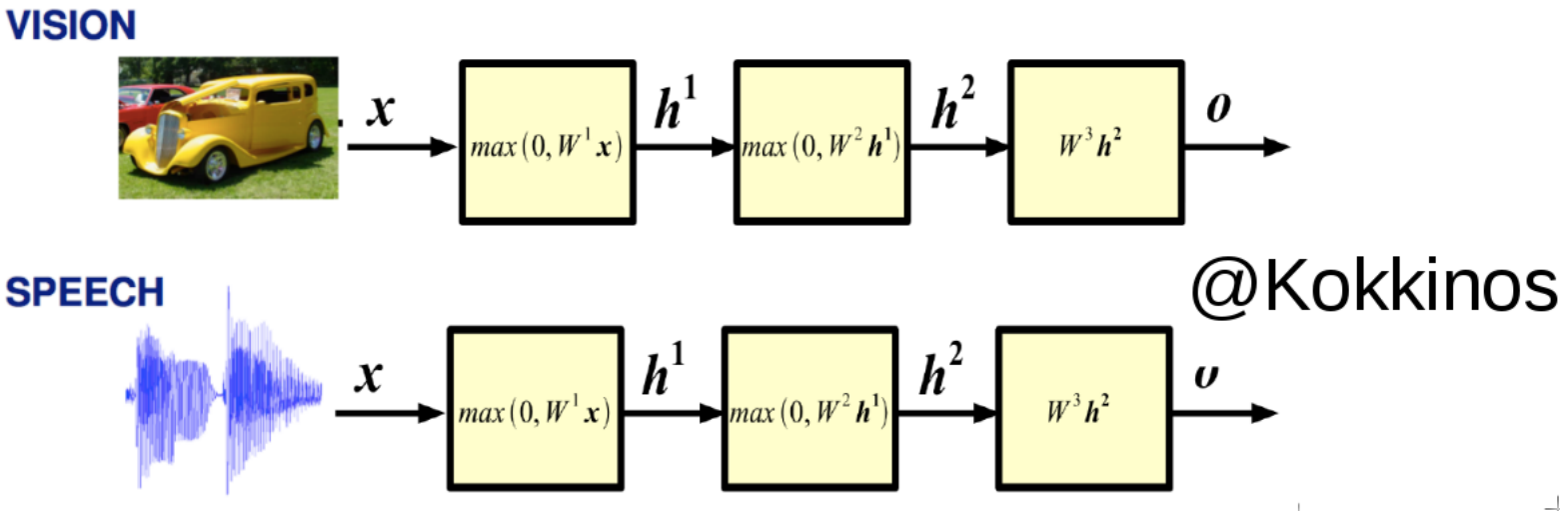
Deep Learning (DL) & Recognition of low-level signals

- DL: breakthrough for the recognition of low-level signal data
- Before DL: handcrafted intermediate representations for each task
  - ⊖ Needs expertise (PhD level) in each field
  - ⊖ Weak level of semantics in the representation



Deep Learning (DL) & Recognition of low-level signals

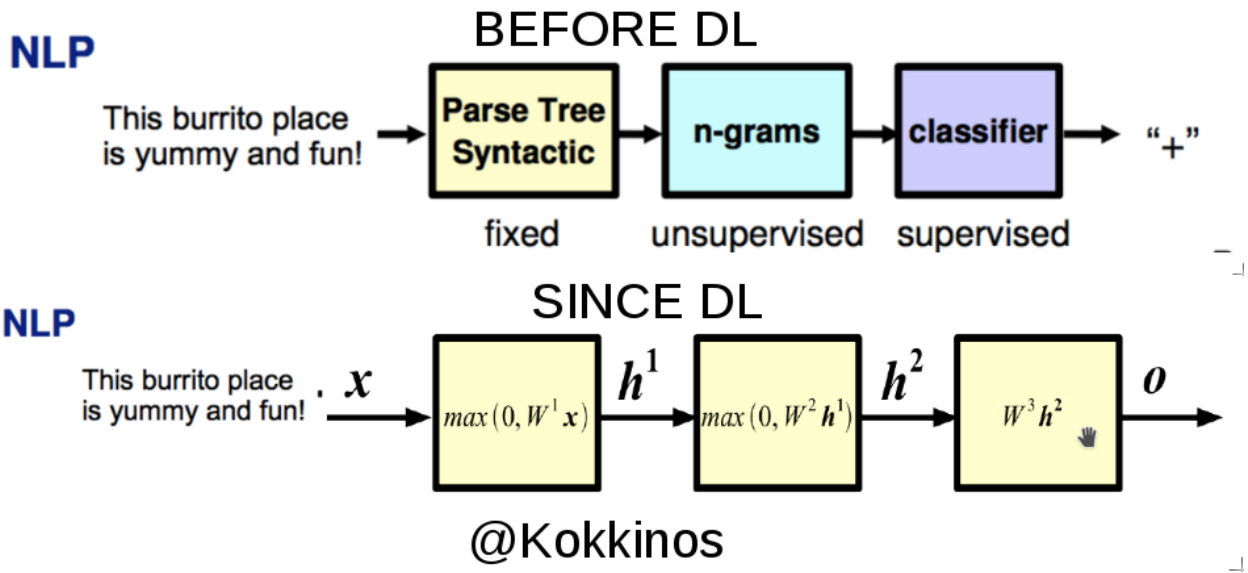
- DL: breakthrough for the recognition of low-level signal data
- Since DL: automatically **learning intermediate representations**
  - ⊕ Outstanding experimental performances >> handcrafted features
  - ⊕ Able to learn high level intermediate representations
  - ⊕ Common learning methodology ⇒ field independent, no expertise

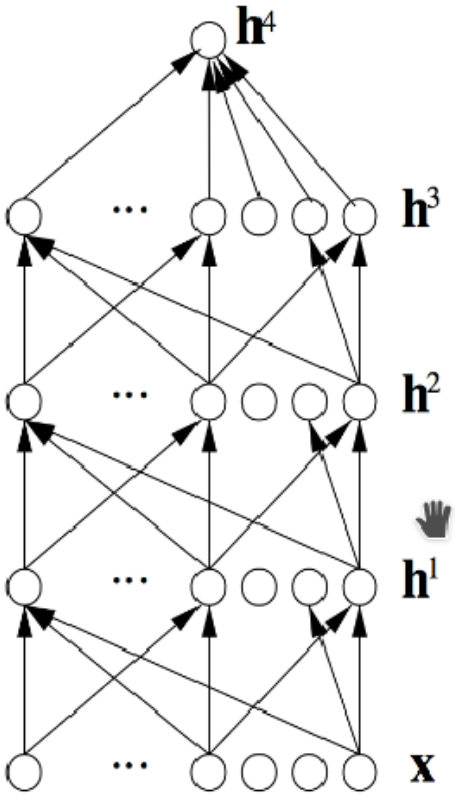


Deep Learning (DL) & Representation Learning

- DL: breakthrough for representation learning
  - Automatically learning intermediate levels of representation

- Ex: Natural language Processing (NLP)

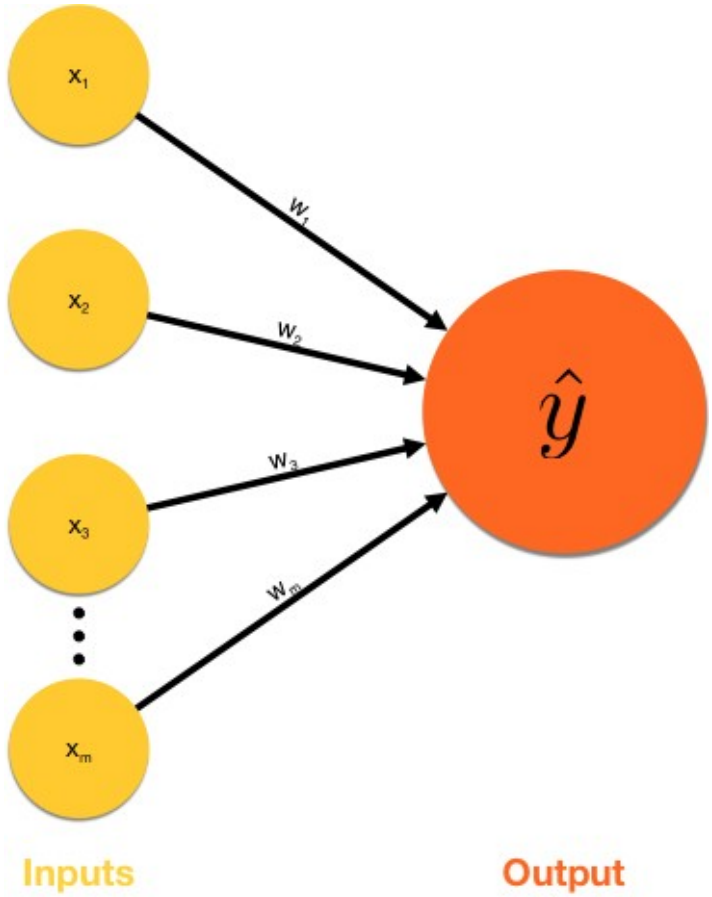




Outline

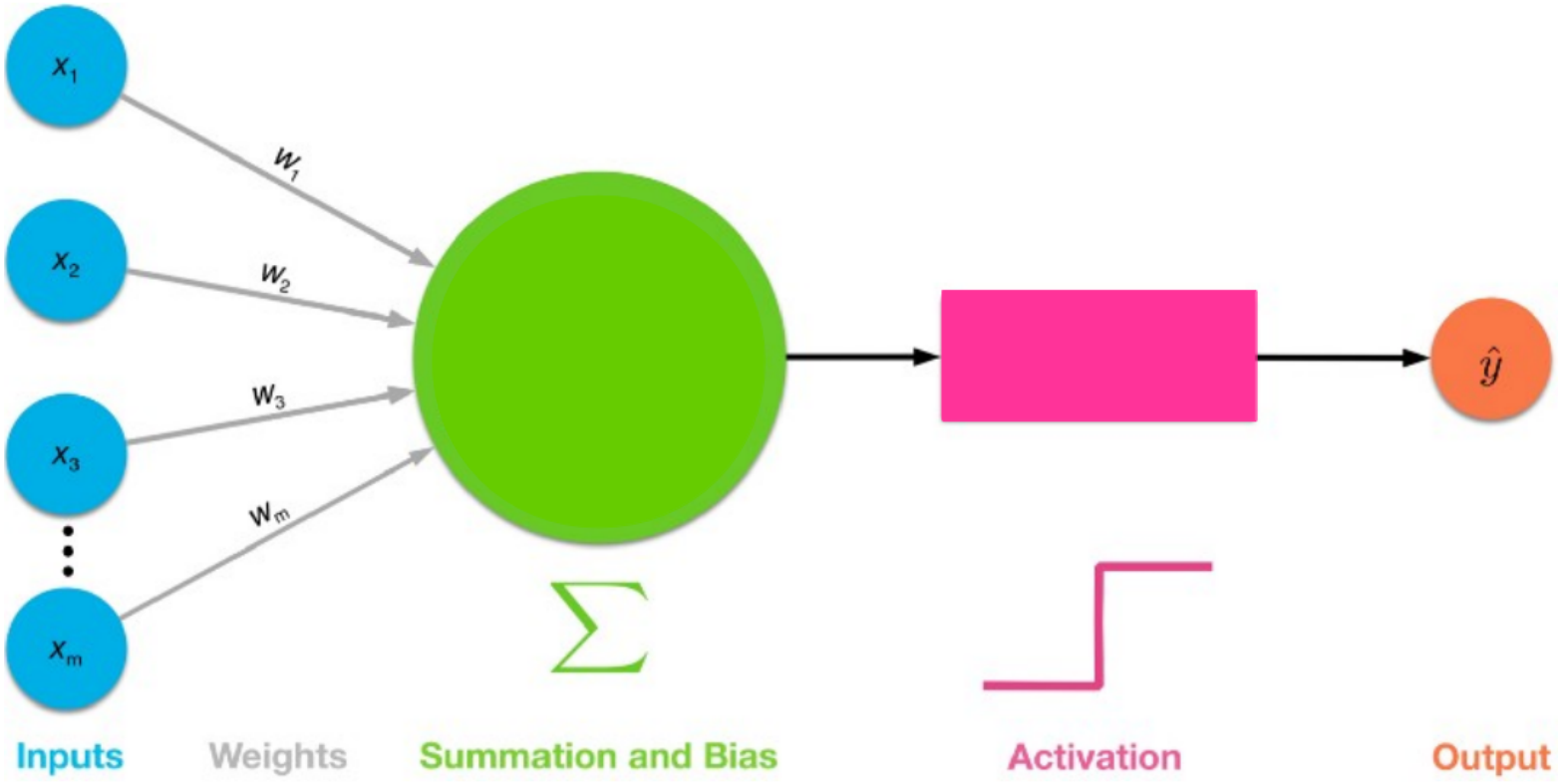
The Formal Neuron: 1943 [MP43]

- Basis of Neural Networks
- Input: vector $x \in \mathbb{R}^m$, *i.e.* $x = \{x_i\}_{i \in \{1,2,\ldots,m\}}$
- Neuron output $\hat{y} \in \mathbb{R}$: scalar



The Formal Neuron: 1943 [MP43]

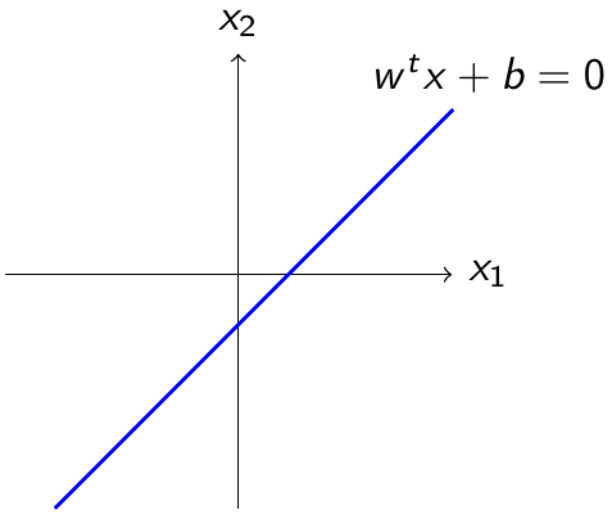
- Mapping from x to $\hat{y}$:
  1. Linear (affine) mapping: $s = \mathrm{w}^\top \mathrm{x} + b$
  2. Non-linear activation function: f: $\hat{y} = f(s)$



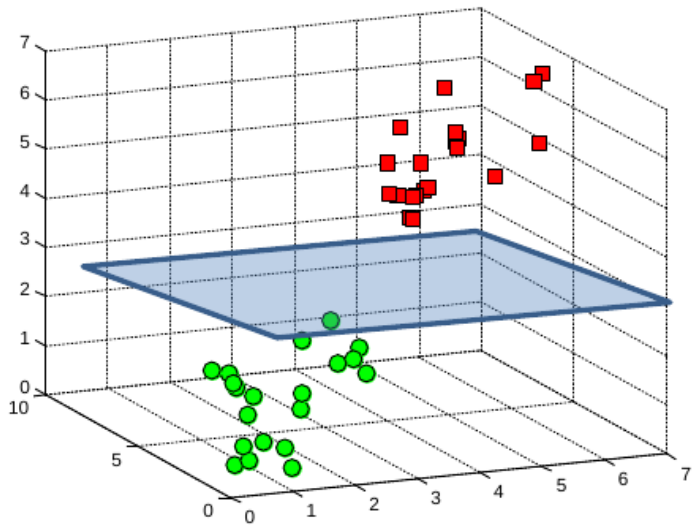
The Formal Neuron: Linear Mapping

- Linear (affine) mapping: $s = \mathrm{w}^\top \mathrm{x} + b = \sum_{i=1}^{m} w_i x_i + b$
  - w: normal vector to an hyperplane in $\mathbb{R}^m \Rightarrow$ **linear boundary**
  - b bias, shift the hyperplane position

2D hyperplane: line

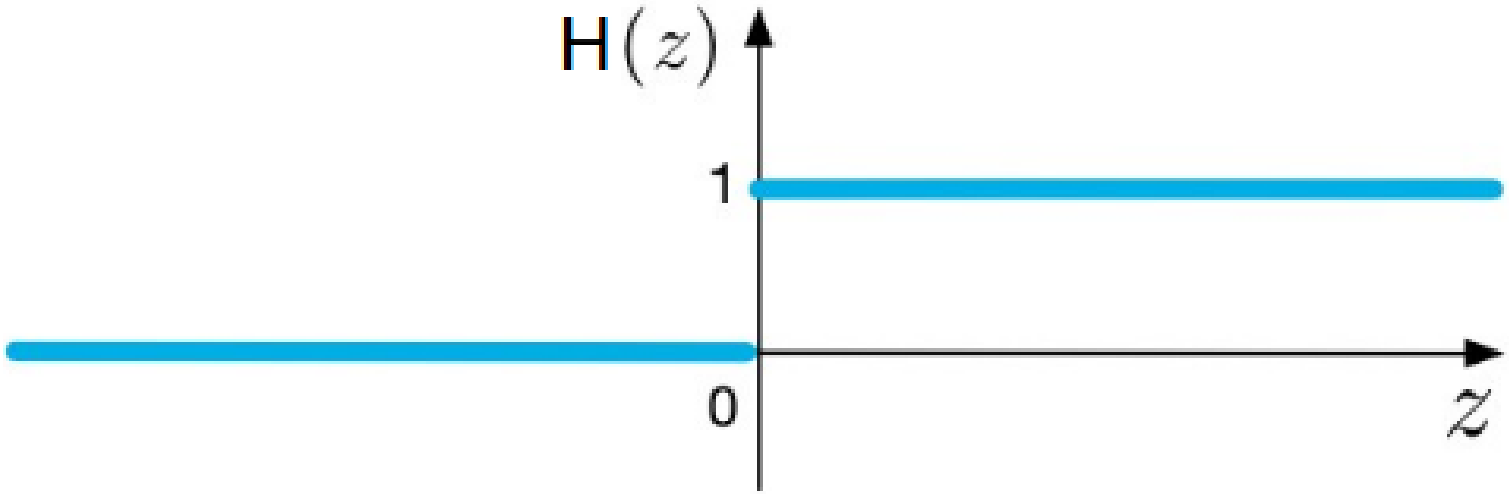


3D hyperplane: plane

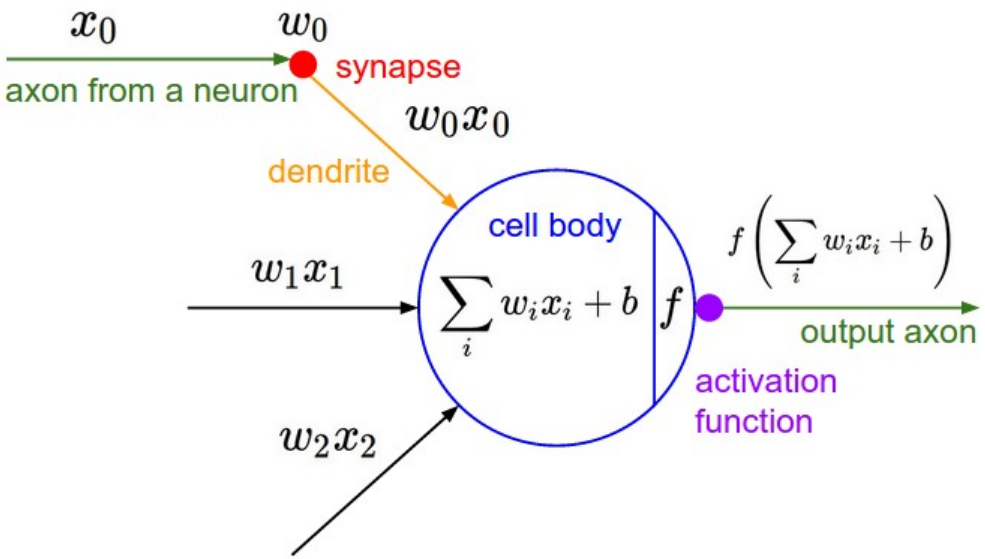


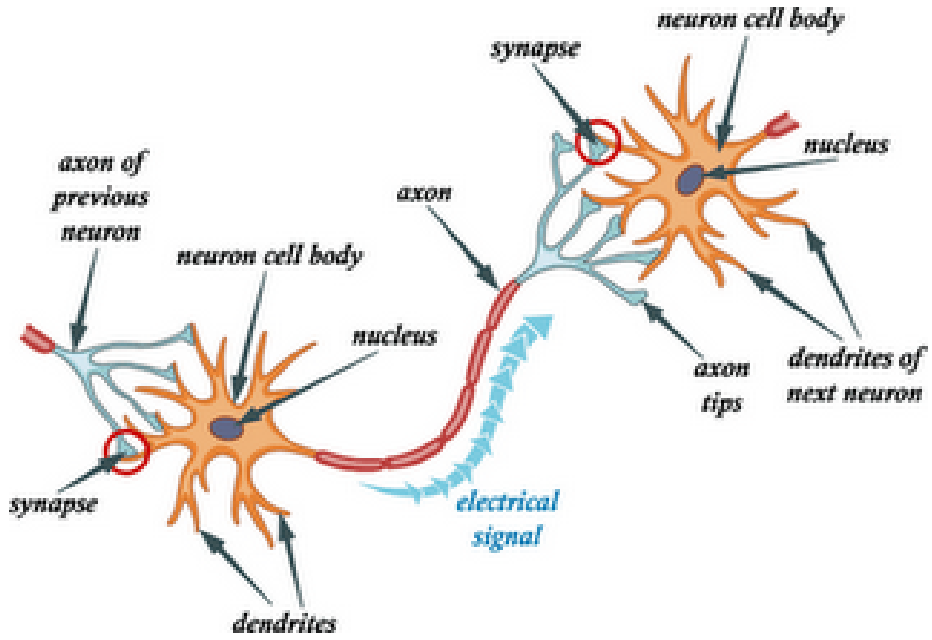
The Formal Neuron: Activation Function

- $\hat{y} = f(\mathrm{w}^\top \mathrm{x} + b)$, f activation function
  - Popular f choices: step, sigmoid, tanh
- Step (Heaviside) function: $H(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{otherwise} \end{cases}$



Step function: Connection to Biological Neurons

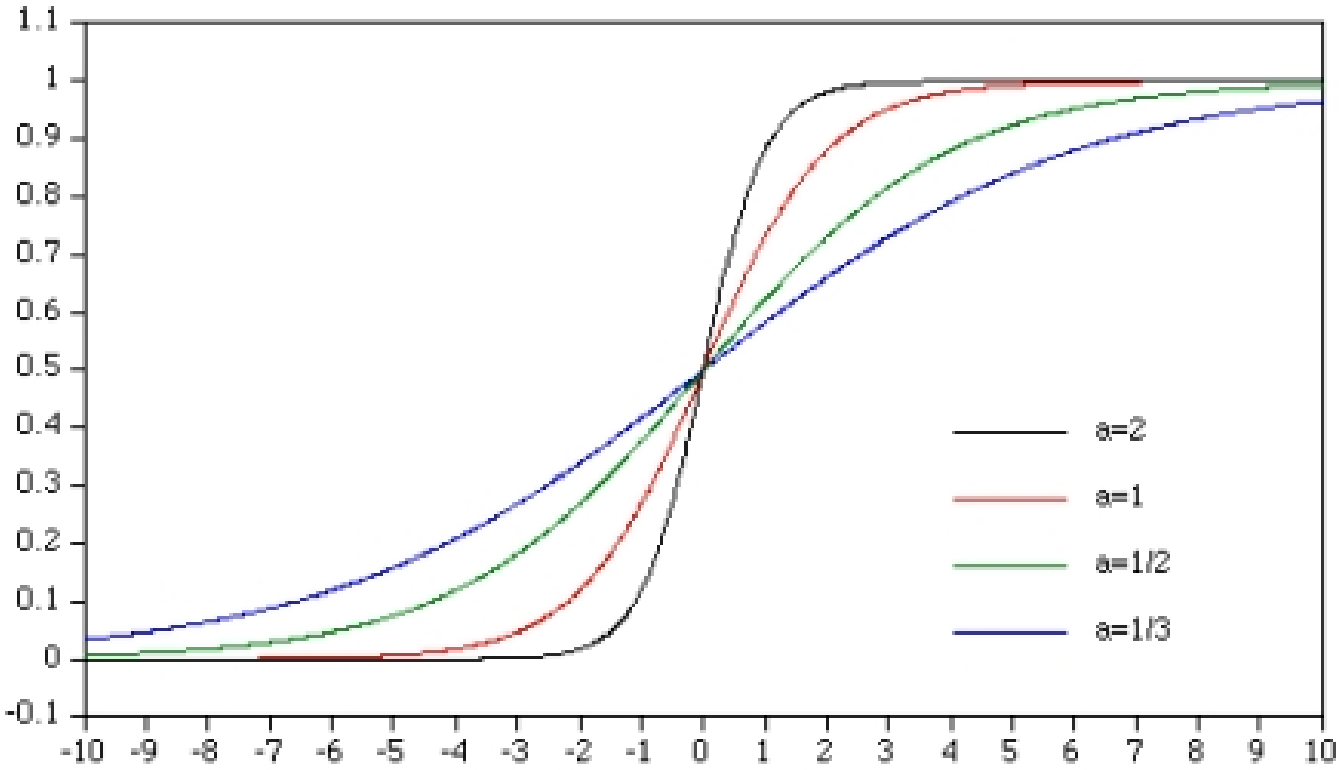




- Formal neuron, step activation H: $\hat{y} = H(\mathrm{w}^\top \mathrm{x} + b)$
  - $\hat{y} = 1$ (activated) $\Leftrightarrow \mathrm{w}^\top \mathrm{x} \geq -b$
  - $\hat{y} = 0$ (unactivated) $\Leftrightarrow \mathrm{w}^\top \mathrm{x} < -b$
- **Biological Neurons:** output activated $\Leftrightarrow$ input weighted by synaptic weight $\geq$ threshold

Sigmoid Activation Function

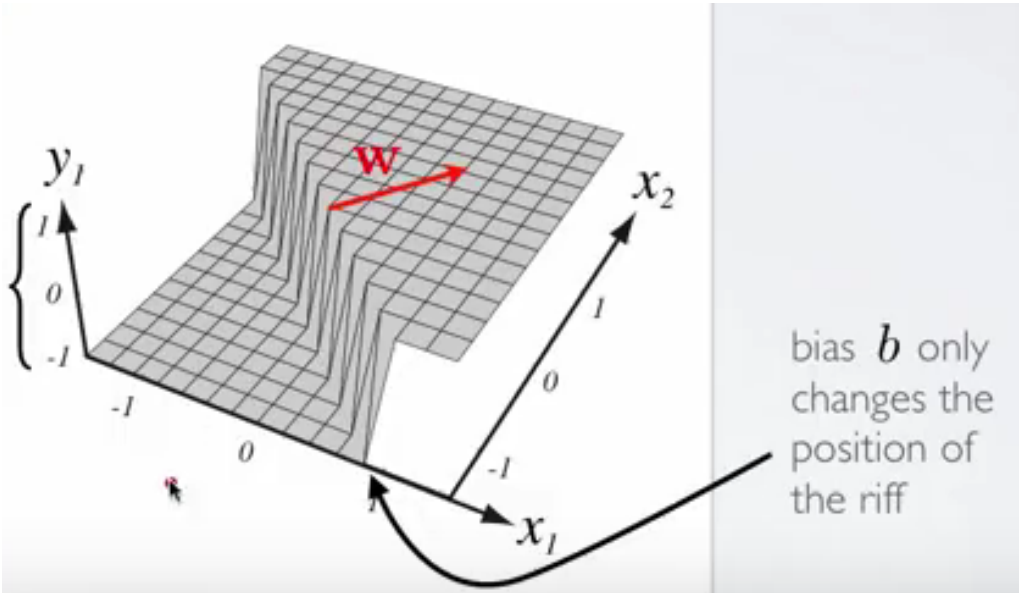
- Neuron output $\hat{y} = f(\mathrm{w}^\top \mathrm{x} + b)$, f activation function
- Sigmoid: $\sigma(z) = (1 + e^{-az})^{-1}$



- $a \uparrow$: more similar to step function (step: $a \to \infty$)
- Sigmoid: linear and saturating regimes

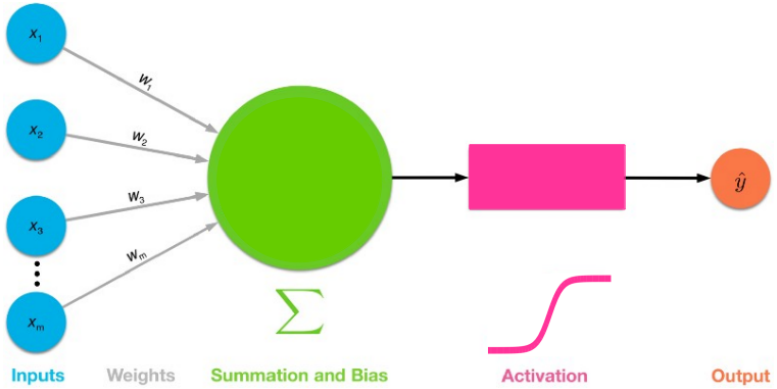
The Formal neuron: Application to Binary Classification

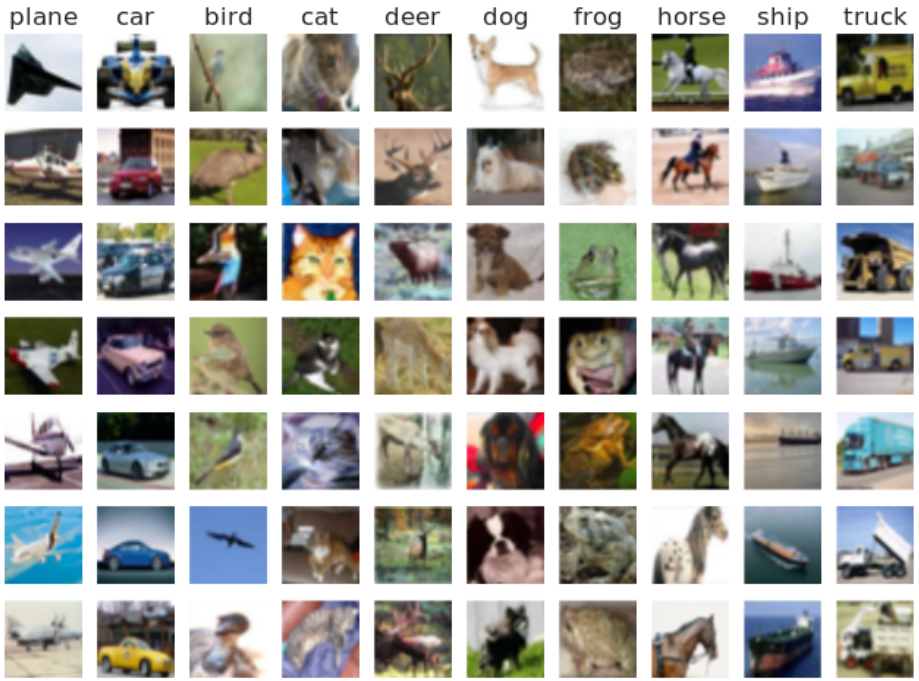
- Binary Classification: label input x as belonging to class 1 or 0
- Neuron output with sigmoid: $\hat{y} = \frac{1}{1+e^{-a(\mathrm{w}^\top \mathrm{x}+b)}}$
- Sigmoid: probabilistic interpretation $\Rightarrow \hat{y} \sim P(1/\mathrm{x})$
  - Input x classified as 1 if $P(1/\mathrm{x}) > 0.5 \Leftrightarrow \mathrm{w}^\top \mathrm{x} + b > 0$
  - Input x classified as 0 if $P(1/\mathrm{x}) < 0.5 \Leftrightarrow \mathrm{w}^\top \mathrm{x} + b < 0$
    $\Rightarrow \mathrm{sign}(\mathrm{w}^\top \mathrm{x} + b)$: linear boundary decision in input space !



From Formal Neuron to Neural Networks

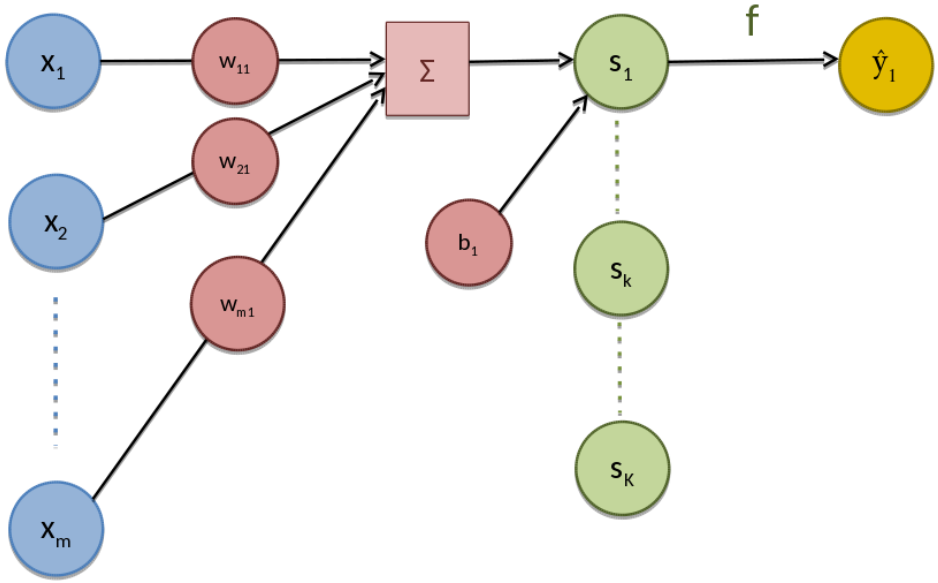
- Formal Neuron:
  1. A single scalar output
  2. Linear decision boundary for binary classification
- Scalar output: limited for several tasks
  - Ex: multi-class classification





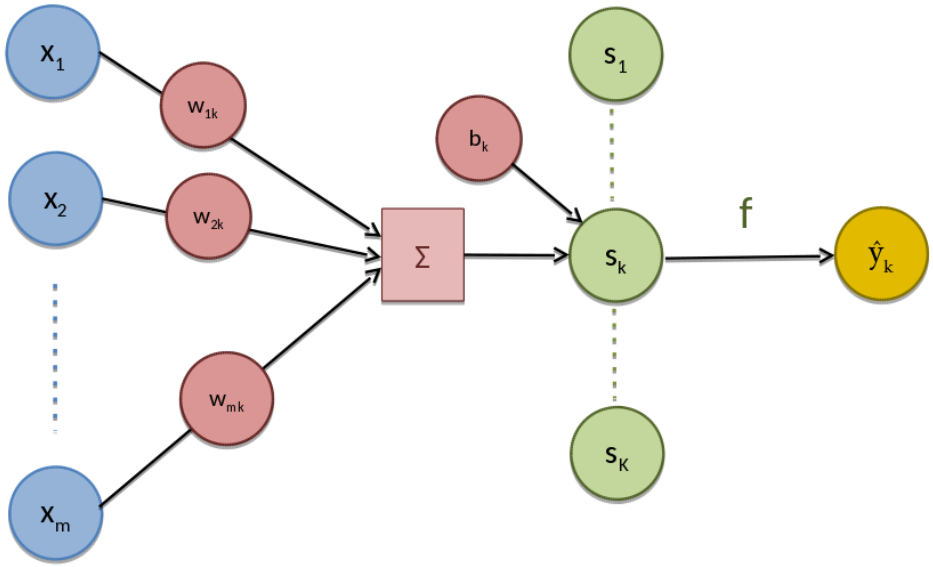
Perceptron and Multi-Class Classification

- Formal Neuron: limited to binary classification
- **Multi-Class Classification:** use several output neurons instead of a single one ! $\Rightarrow$ **Perceptron**
- Input x in $\mathbb{R}^m$
- Output neuron $\hat{y}_1$ is a formal neuron:
  - Linear (affine) mapping: $s_1 = w_1{}^\top x + b_1$
  - Non-linear activation function: f: $\hat{y}_1 = f(s_1)$
- Linear mapping parameters:
  - $w_1 = \{w_{11}, ..., w_{m1}\} \in \mathbb{R}^m$
  - $b_1 \in \mathbb{R}$



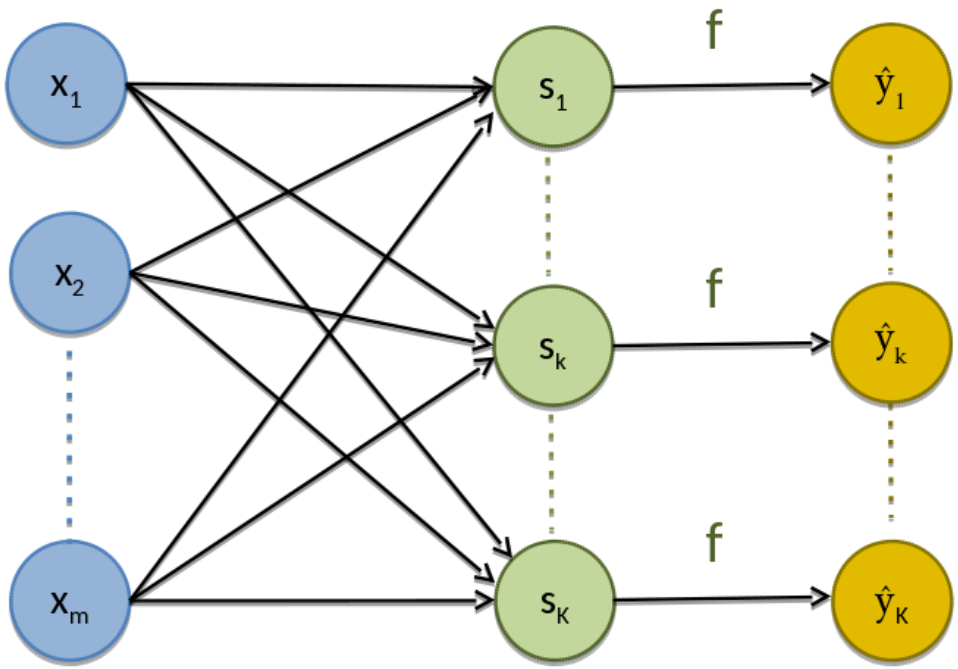
Perceptron and Multi-Class Classification

- Input x in $\mathbb{R}^m$
- Output neuron $\hat{y}_k$ is a formal neuron:
  - Linear (affine) mapping: $s_k = \mathrm{w_k}^\top \mathrm{x} + b_k$
  - Non-linear activation function: f: $\hat{y}_k = f(s_k)$
- Linear mapping parameters:
  - $\mathrm{w_k} = \{w_{1k}, ..., w_{mk}\} \in \mathbb{R}^m$
  - $b_k \in \mathbb{R}$



Perceptron and Multi-Class Classification

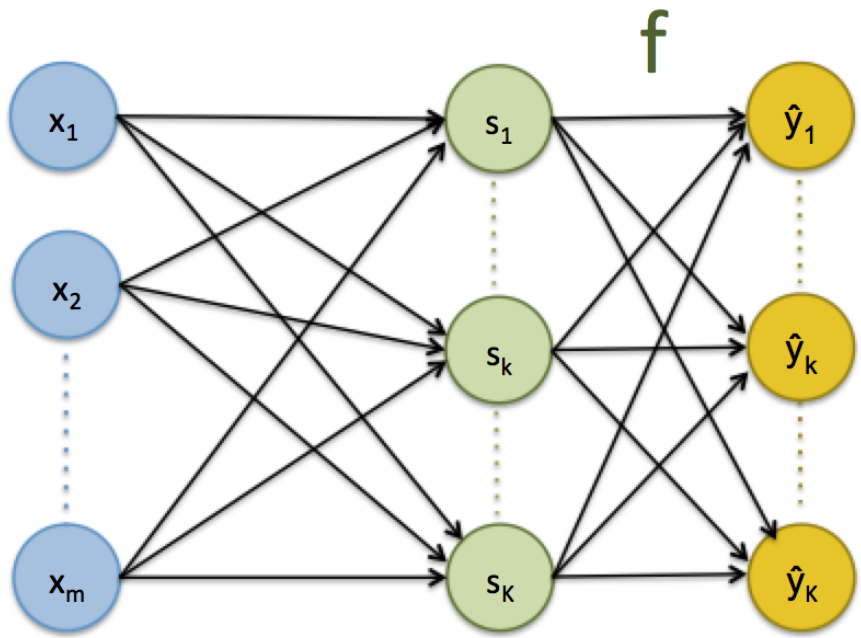
- Input x in $\mathbb{R}^m$ $(1 \times m)$, output $\hat{y}$: concatenation of K formal neurons
- Linear (affine) mapping $\sim$ matrix multiplication: $s = xW + b$
  - W matrix of size $m \times K$ - columns are w_k
  - b: bias vector - size $1 \times K$
- Element-wise non-linear activation: $\hat{y} = f(s)$



Perceptron and Multi-Class Classification

- **Soft-max Activation**:

$$\hat{y}_k = f(s_k) = \frac{e^{s_k}}{\sum_{k'=1}^{K} e^{s_{k'}}}$$



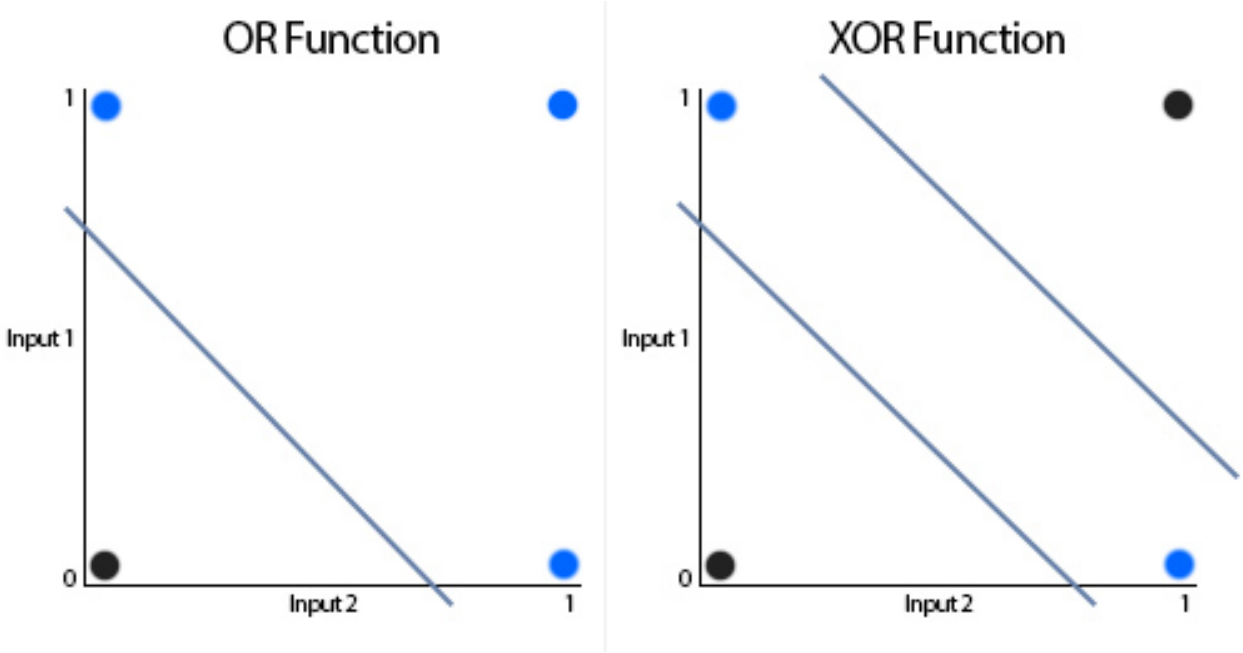
- **Probabilistic interpretation for multi-class classification**:
  - Each output neuron $\Leftrightarrow$ class
  - $\hat{y}_k \sim P(k/\mathrm{x}, \mathrm{w})$

$\Rightarrow$ **Logistic Regression (LR) Model !**

Beyond Linear Classification

X-OR Problem

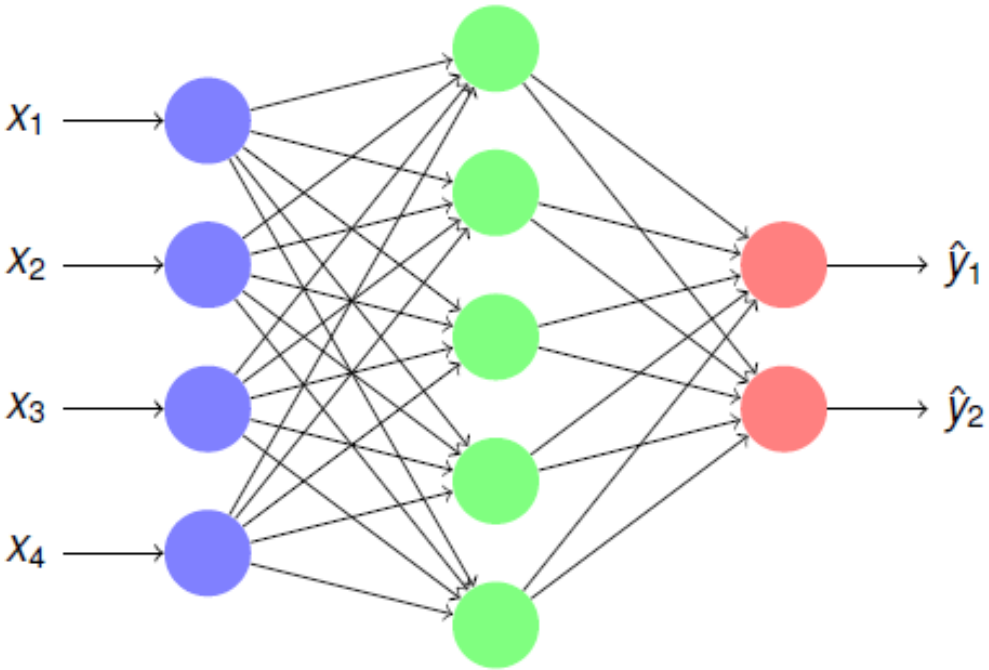
- Logistic Regression (LR): NN with 1 input layer & 1 output layer
- LR: limited to linear decision boundaries
- **X-OR:** NOT 1 and 2 OR NOT 2 AND 1
  - **X-OR: Non linear decision function**



Beyond Linear Classification

- LR: limited to linear boundaries
- **Solution:** add a layer !

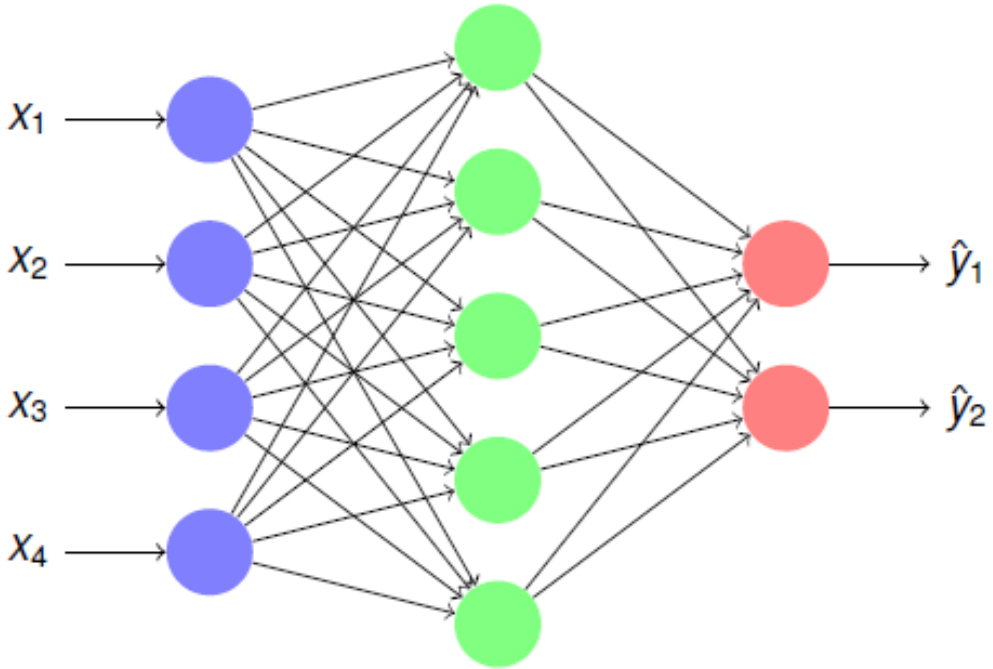
- Input x in $\mathbb{R}^m$, *e.g.* $m = 4$
- Output $\hat{y}$ in $\mathbb{R}^K$ (K # classes), *e.g.* $K = 2$
- **Hidden layer h in $\mathbb{R}^L$**



Multi-Layer Perceptron

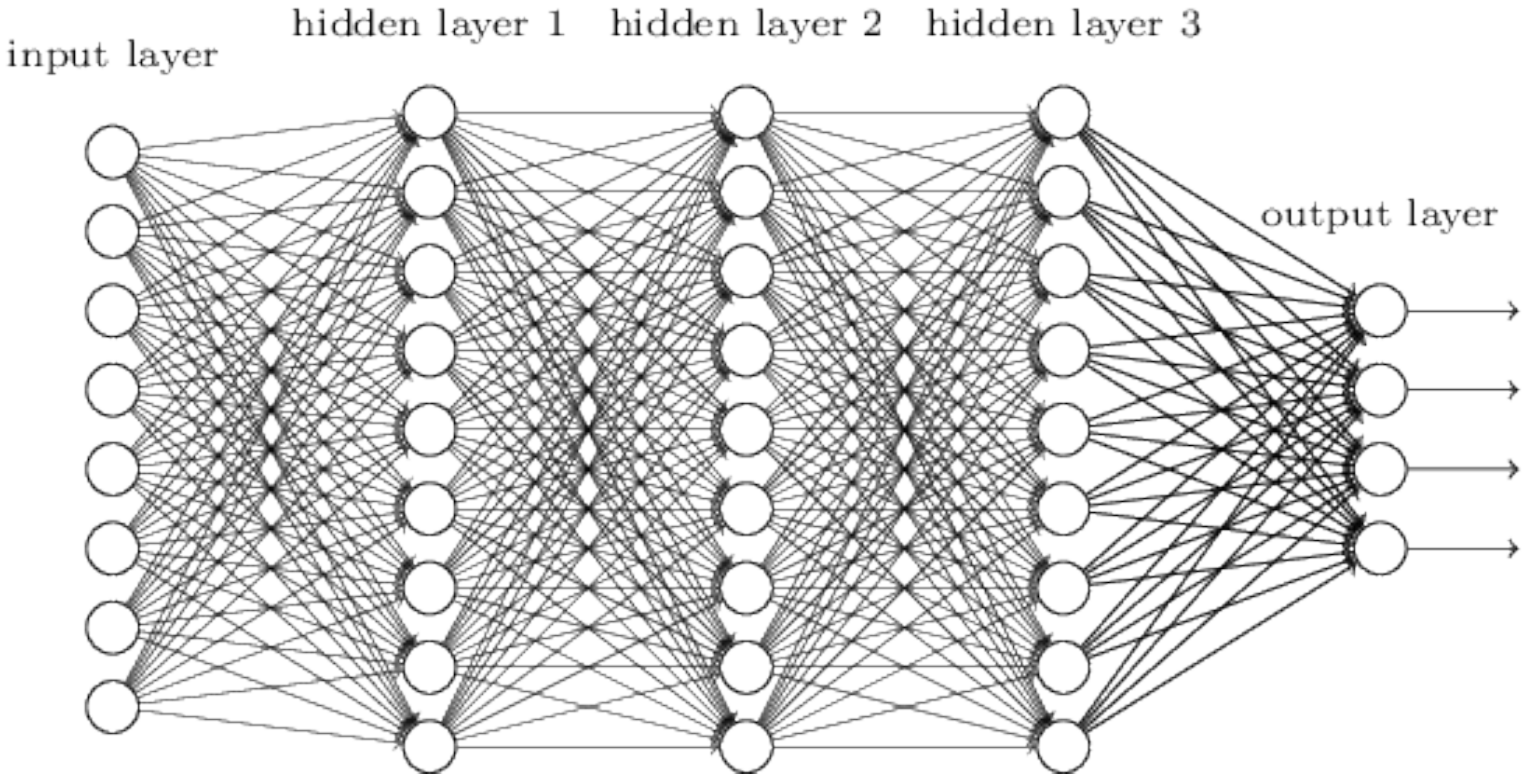
- **Hidden layer** h: x projection to a new space $\mathbb{R}^L$
- Neural Net with ≥ 1 hidden layer: **Multi-Layer Perceptron (MLP)**

- h: intermediate representations of x for classification $\hat{y}$: $h = f(xW + b)$
- **Mapping from x to $\hat{y}$: non-linear boundary ! $\Rightarrow$ activation f crucial!**



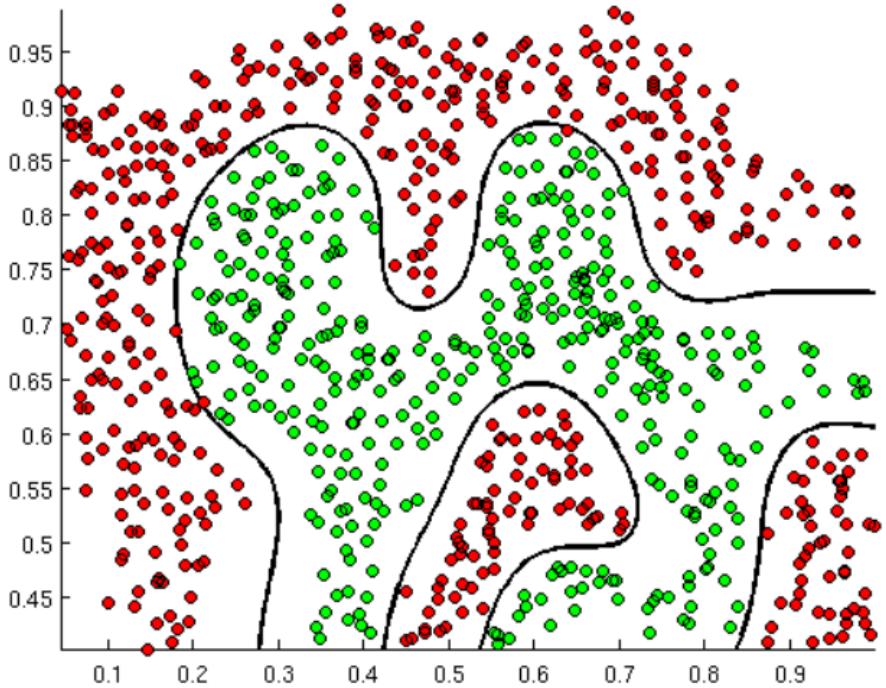
Deep Neural Networks

- Adding more hidden layers: Deep Neural Networks (DNN) ⇒ **Basis of Deep Learning**
- Each layer h^l projects layer h^{l-1} into a new space
- Gradually learning intermediate representations useful for the task



Conclusion

- Deep Neural Networks: applicable to classification problems with non-linear decision boundaries



- Visualize prediction from fixed model parameters
- Reverse problem: **Supervised Learning**

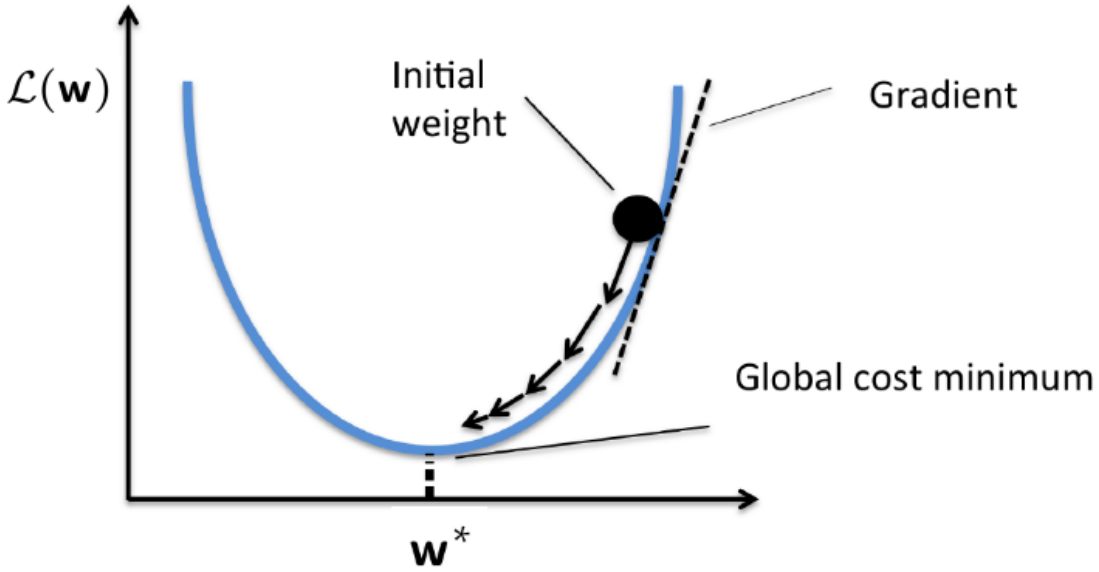
Outline

Training Multi-Layer Perceptron (MLP)

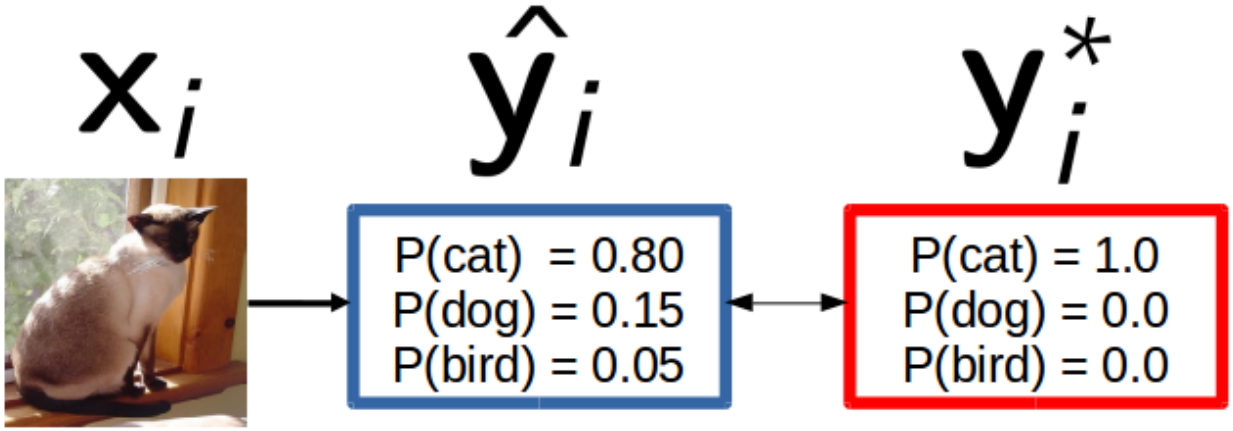
- Input x, output y
- A parametrized model $x \Rightarrow y$: $f_w(x_i) = \hat{y}_i$
- Supervised context: training set $\mathcal{A} = \{(x_i, y_i^*)\}_{i \in \{1,2,\ldots,N\}}$
  - A loss function $\ell(\hat{y}_i, y_i^*)$ for each annotated pair (x_i, y_i^*)
- Assumptions: parameters $w \in \mathbb{R}^d$ continuous, $\mathcal{L}$ differentiable
- Gradient $\nabla_w = \frac{\partial \mathcal{L}}{\partial w}$: steepest direction to decrease loss $\mathcal{L}$

MLP Training

- Gradient descent algorithm:
  - Initialyze parameters w
  - Update: $\boxed{\mathrm{w}^{(t+1)} = \mathrm{w}^{(t)} - \eta \frac{\partial \mathcal{L}}{\partial \mathrm{w}}}$
  - Until convergence, *e.g.* $\|\nabla_{\mathrm{w}}\|^2 \approx 0$



MLP Training: loss function



- **Input x_i, ground truth output supervision y_i^***
- One hot-encoding for y_i^*:

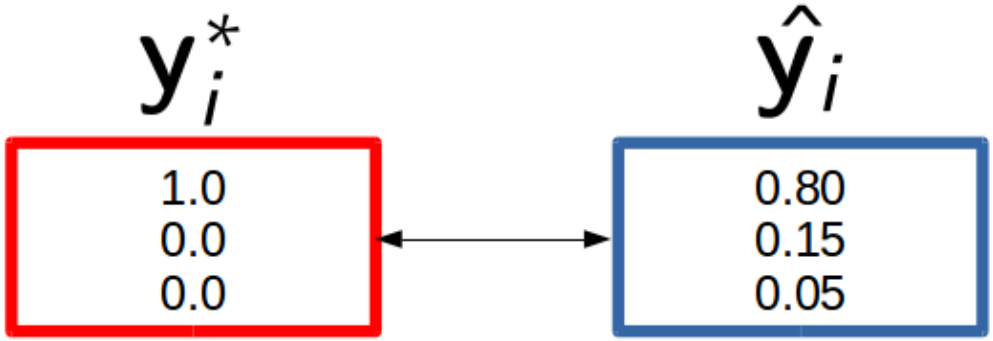
$$y^*_{c,i} = \begin{cases} 1 & \text{if c is the groud truth class for } x_i \\ 0 & \text{otherwise} \end{cases}$$

MLP Training

- Loss function: multi-class Cross-Entropy (CE) ℓ_{CE}
- ℓ_{CE}: Kullback-Leiber divergence between y_i^* and $\hat{y}_i$

$$\ell_{CE}(\hat{y}_i, y_i^*) = KL(y_i^*, \hat{y}_i) = -\sum_{c=1}^{K} y_{c,i}^* log(\hat{y}_{c,i}) = -log(\hat{y}_{c^*,i})$$

- KL asymmetric: $KL(\hat{y}_i, y_i^*) \neq KL(y_i^*, \hat{y}_i)$

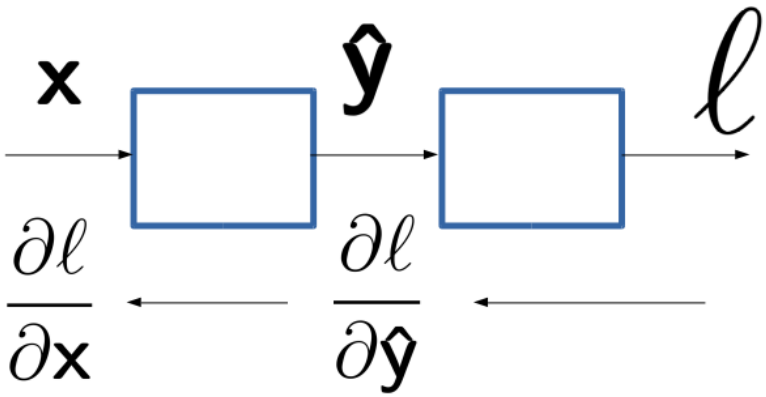


$$KL(y_i^*, \hat{y}_i) = -log(\hat{y}_{c^*,i}) = -log(0.8) \approx 0.22$$

MLP Training: Backpropagation

- $\mathcal{L}_{CE}(\mathrm{W}, \mathrm{b}) = \frac{1}{N} \sum_{i=1}^{N} \ell_{CE}(\hat{\mathrm{y}}_i, \mathrm{y}_i^*) = -\frac{1}{N} \sum_{i=1}^{N} log(\hat{y}_{c^*,i})$
- ℓ_{CE} smooth convex upper bound of $\ell_{0/1}$
  $\Rightarrow$ **gradient descent optimization**
- Gradient descent: $\mathrm{W}^{(t+1)} = \mathrm{W}^{(t)} - \eta \frac{\partial \mathcal{L}_{CE}}{\partial \mathrm{W}}$
  $(\mathrm{b}^{(t+1)} = \mathrm{b}^{(t)} - \eta \frac{\partial \mathcal{L}_{CE}}{\partial \mathrm{b}})$
- Computing $\frac{\partial \mathcal{L}_{CE}}{\partial \mathrm{W}} = \frac{1}{N} \sum_{i=1}^{N} \frac{\partial \ell_{CE}}{\partial \mathrm{W}}$?
  $\Rightarrow$ **Backpropagation of gradient error!**
  $\Rightarrow$ **Key Property:** chain rule $\frac{\partial \mathrm{x}}{\partial z} = \frac{\partial \mathrm{x}}{\partial \mathrm{y}} \frac{\partial \mathrm{y}}{\partial z}$

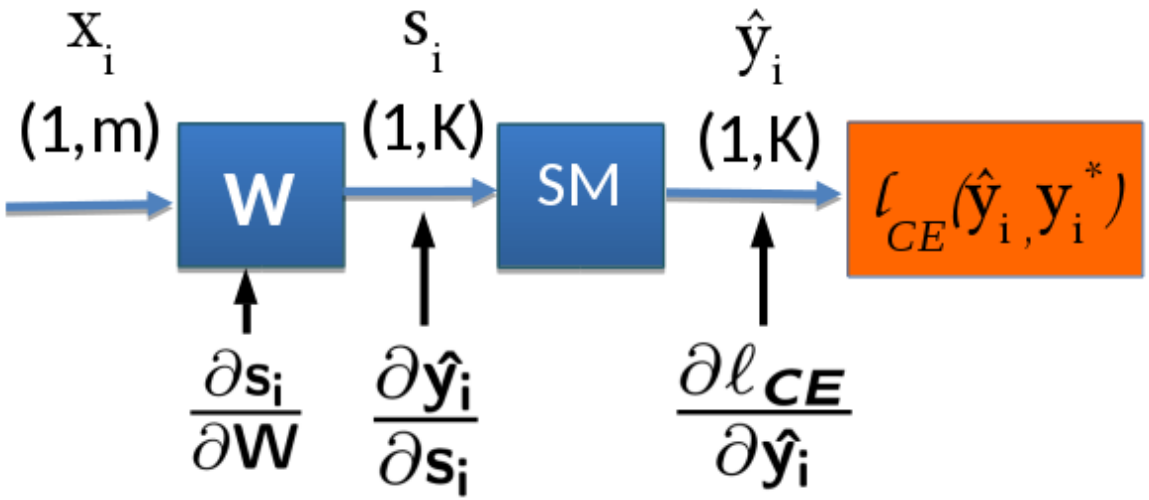
Chain Rule



$$\frac{\partial \ell}{\partial x} = \frac{\partial \ell}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial x}$$

- Logistic regression:

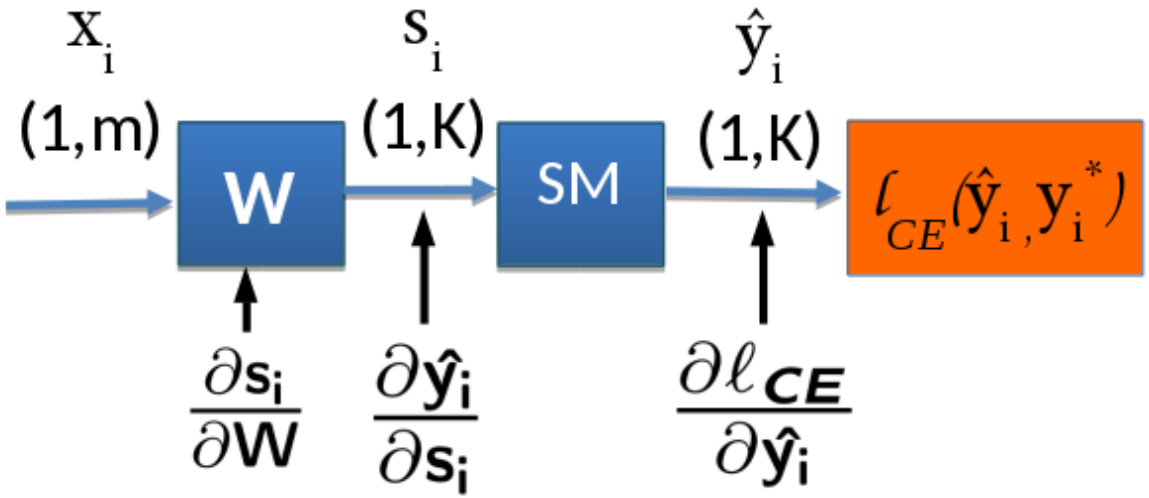
$$\frac{\partial \ell_{CE}}{\partial W} = \frac{\partial \ell_{CE}}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial s_i} \frac{\partial s_i}{\partial W}$$



Logistic Regression Training: Backpropagation

$\frac{\partial \ell_{CE}}{\partial W} = \frac{\partial \ell_{CE}}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial s_i} \frac{\partial s_i}{\partial W}$, $\ell_{CE}(\hat{y}_i, y_i^*) = -log(\hat{y}_{c^*,i}) \Rightarrow$ Update for 1 example:

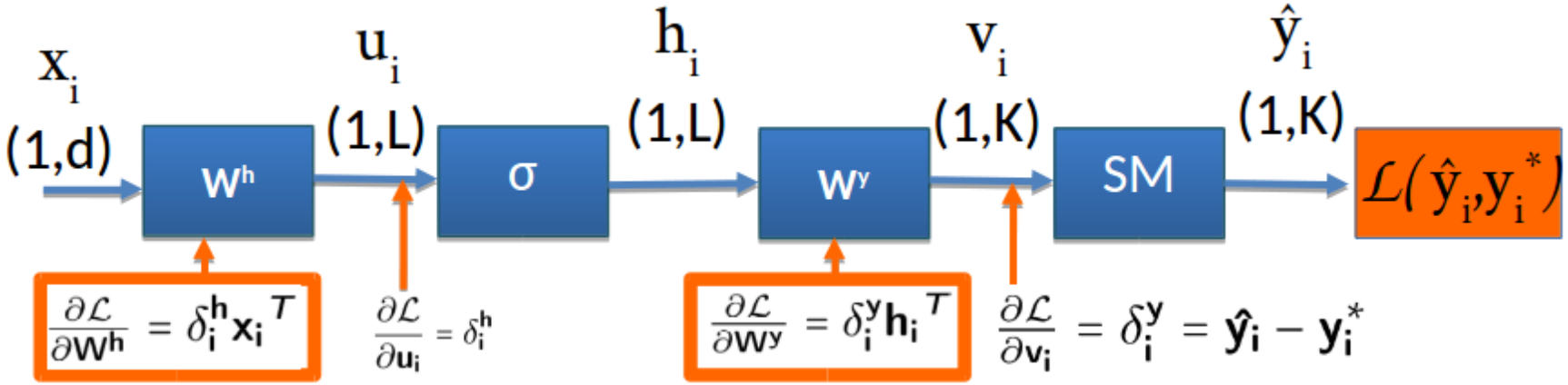
- $\frac{\partial \ell_{CE}}{\partial \hat{y}_i} = \frac{-1}{\hat{y}_{c^*,i}} = \frac{-1}{\hat{y}_i} \odot \delta_{c,c^*}$
- $\frac{\partial \ell_{CE}}{\partial s_i} = \hat{y}_i - y_i^* = \delta_i^y$
- $\frac{\partial \ell_{CE}}{\partial W} = x_i{}^T \delta_i^y$



Perceptron Training: Backpropagation

- Perceptron *vs* Logistic Regression: adding hidden layer (sigmoid)
- **Goal:** Train parameters W^y and W^h (+bias) with Backpropagation

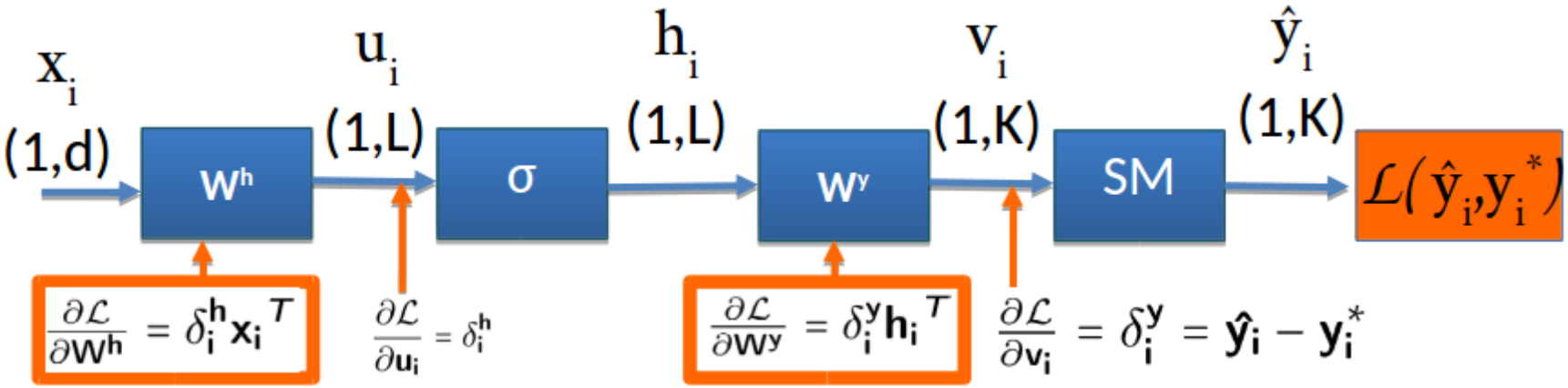
$\Rightarrow$ computing $\boxed{\frac{\partial \mathcal{L}_{CE}}{\partial W^y} = \frac{1}{N}\sum_{i=1}^{N} \frac{\partial \ell_{CE}}{\partial W^y}}$ and $\boxed{\frac{\partial \mathcal{L}_{CE}}{\partial W^h} = \frac{1}{N}\sum_{i=1}^{N} \frac{\partial \ell_{CE}}{\partial W^h}}$



- Last hidden layer $\sim$ Logistic Regression
- First hidden layer: $\frac{\partial \ell_{CE}}{\partial W^h} = x_i^T \frac{\partial \ell_{CE}}{\partial u_i} \Rightarrow$ **computing** $\frac{\partial \ell_{CE}}{\partial u_i} = \delta_i^h$

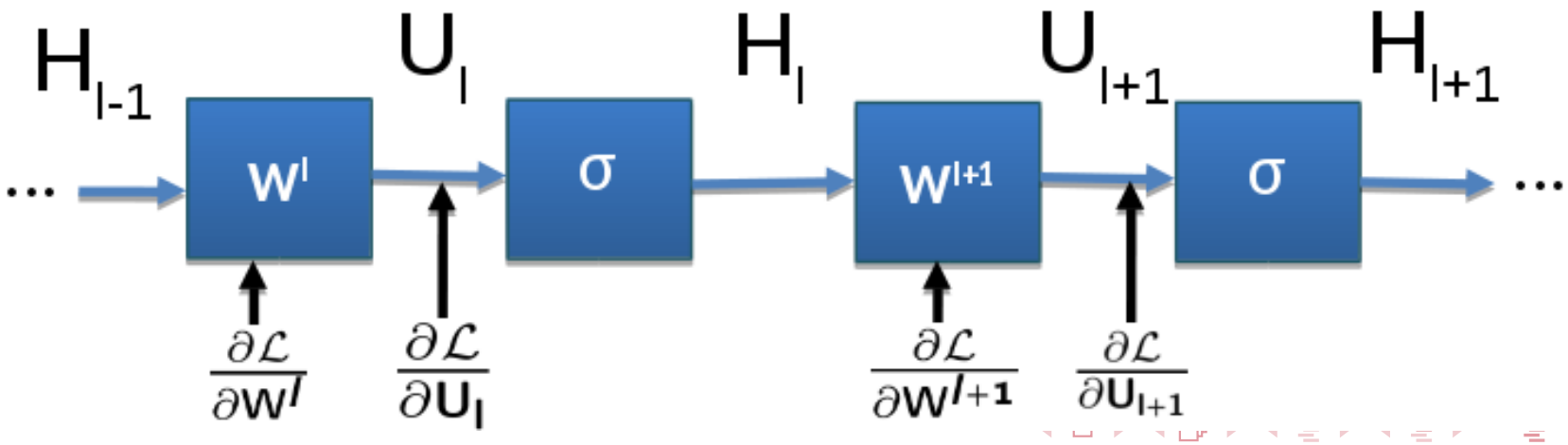
Perceptron Training: Backpropagation

- Computing $\frac{\partial \ell_{CE}}{\partial u_i} = \delta_i^h \Rightarrow$ use chain rule: $\frac{\partial \ell_{CE}}{\partial u_i} = \frac{\partial \ell_{CE}}{\partial v_i} \frac{\partial v_i}{\partial h_i} \frac{\partial h_i}{\partial u_i}$
- ... Leading to:
  $\frac{\partial \ell_{CE}}{\partial u_i} = \delta_i^h = \delta_i^{y\,T} W^y \odot \sigma'(h_i) = \delta_i^{y\,T} W^y \odot (h_i \odot (1 - h_i))$



Deep Neural Network Training: Backpropagation

- Multi-Layer Perceptron (MLP): adding more hidden layers
- Backpropagation update ~ Perceptron: **assuming** $\frac{\partial \mathcal{L}}{\partial U_{l+1}} = \Delta^{l+1}$ **known**
  - $\boxed{\frac{\partial \mathcal{L}}{\partial W^{l+1}} = H_l{}^T \Delta^{l+1}}$
  - $\boxed{\text{Computing } \frac{\partial \mathcal{L}}{\partial U_l} = \Delta^l}$ $(= \Delta^{l+1}{}^T W^{l+1} \odot H_l \odot (1 - H_l)$ sigmoid$)$
  - $\boxed{\frac{\partial \mathcal{L}}{\partial W^l} = H_{l-1}{}^T \Delta^{h_l}}$



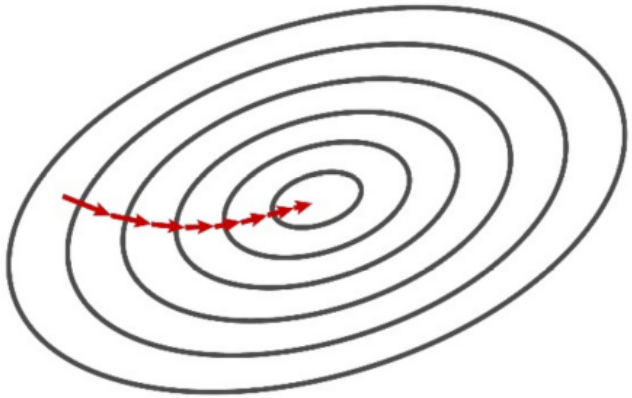
Neural Network Training: Optimization Issues

- Classification loss over training set (w):

$$\mathcal{L}_{CE}(\mathrm{w}) = \frac{1}{N}\sum_{i=1}^{N}\ell_{CE}(\hat{\mathrm{y}}_i, \mathrm{y}_i^*) = -\frac{1}{N}\sum_{i=1}^{N} log(\hat{y}_{c^*,i})$$

- Gradient descent optimization:

$$\mathrm{w}^{(t+1)} = \mathrm{w}^{(t)} - \eta\frac{\partial\mathcal{L}_{CE}}{\partial \mathrm{w}}\left(\mathrm{w}^{(t)}\right) = \mathrm{w}^{(t)} - \eta\nabla_{\mathrm{w}}^{(}$$



- Gradient $\nabla_{\mathrm{w}}^{(t)} = \frac{1}{N}\sum_{i=1}^{N}\frac{\partial\ell_{CE}(\hat{\mathrm{y}}_i,\mathrm{y}_i^*)}{\partial \mathrm{w}}\left(\mathrm{w}^{(t)}\right)$ linea..,

  scales wrt:
  - w dimension
  - Training set size

$\Rightarrow$ **Too slow even for moderate dimensionality & dataset size!**

Stochastic Gradient Descent

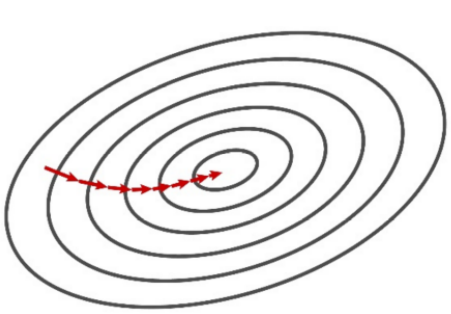
- **Solution:** approximate $\nabla_{\mathrm{w}}^{(t)} = \frac{1}{N} \sum_{i=1}^{N} \frac{\partial \ell_{CE}(\hat{\mathrm{y}}_i, \mathrm{y}_i^*)}{\partial \mathrm{w}} \left(\mathrm{w}^{(t)}\right)$ with subset

  $\Rightarrow$ **Stochastic Gradient Descent (SGD)**
  - Use a single example (online):

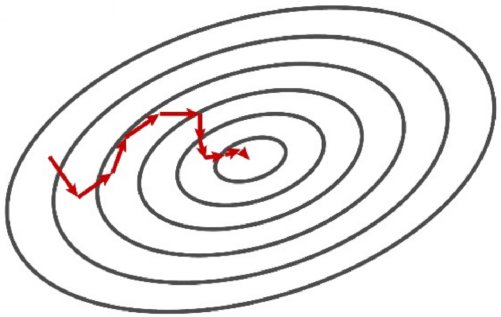
$$\nabla_{\mathrm{w}}^{(t)} \approx \frac{\partial \ell_{CE}(\hat{\mathrm{y}}_i, \mathrm{y}_i^*)}{\partial \mathrm{w}} \left(\mathrm{w}^{(t)}\right)$$

  - Mini-batch: use $B < N$ examples:

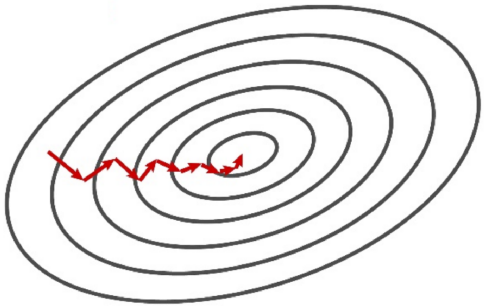
$$\nabla_{\mathrm{w}}^{(t)} \approx \frac{1}{B} \sum_{i=1}^{B} \frac{\partial \ell_{CE}(\hat{\mathrm{y}}_i, \mathrm{y}_i^*)}{\partial \mathrm{w}} \left(\mathrm{w}^{(t)}\right)$$



Full gradient



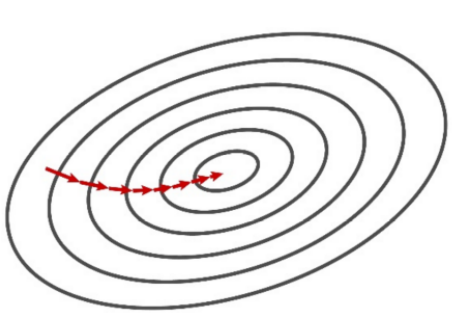
SGD (online)



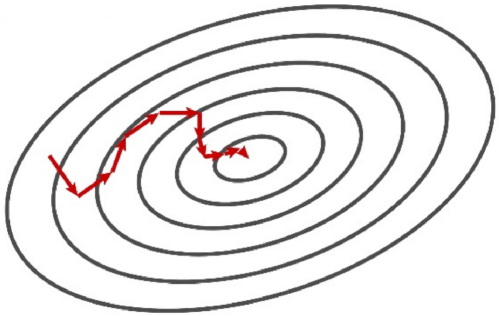
SGD (mini-batch)

Stochastic Gradient Descent

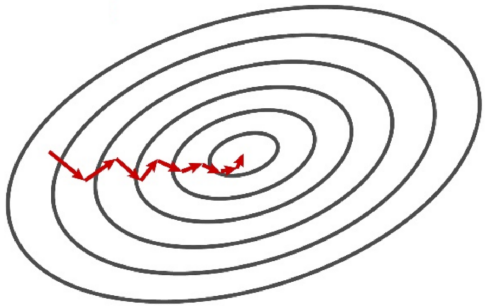
- **SGD: approximation of the true Gradient ∇_w !**
  - Noisy gradient can lead to bad direction, increase loss
  - **BUT:** much more parameter updates: online $\times N$, mini-batch $\times \frac{N}{B}$
  - **Faster convergence**, at the core of Deep Learning for large scale datasets



Full gradient



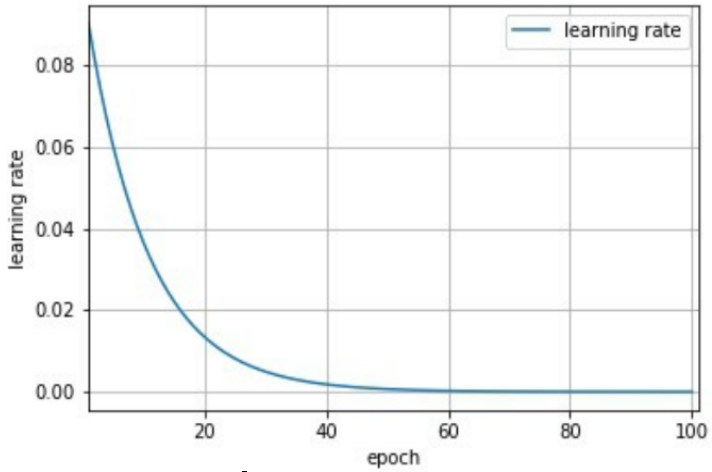
SGD (online)



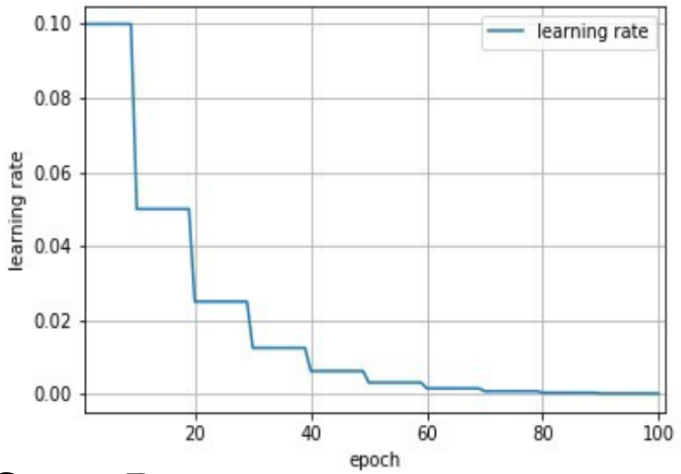
SGD (mini-batch)

Optimization: Learning Rate Decay

- Gradient descent optimization: $\mathrm{w}^{(t+1)} = \mathrm{w}^{(t)} - \eta \nabla_{\mathrm{w}}^{(t)}$
- η setup ? $\Rightarrow$ open question
- Learning Rate Decay: decrease η during training progress
  - Inverse (time-based) decay: $\eta_t = \frac{\eta_0}{1+r \cdot t}$, r decay rate
  - Exponential decay: $\eta_t = \eta_0 \cdot e^{-\lambda t}$
  - Step Decay $\eta_t = \eta_0 \cdot r^{\frac{t}{t_u}}$ ...



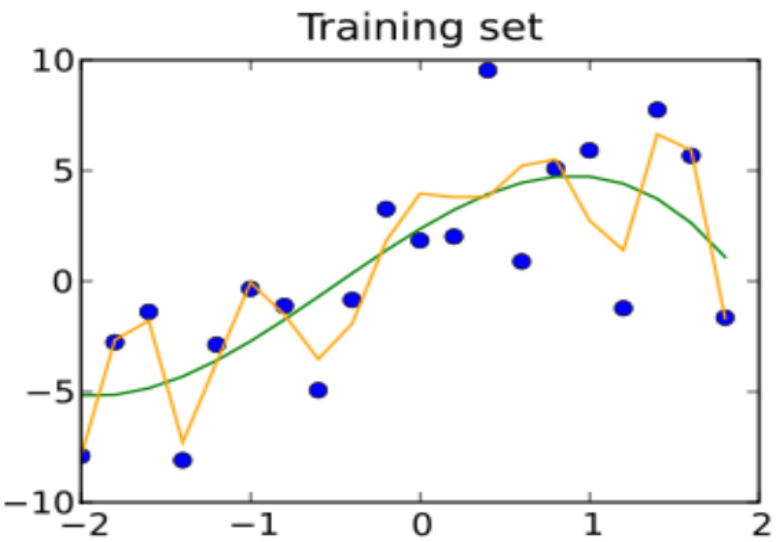
Exponential Decay ($\eta_0 = 0.1$, $\lambda = 0.1$s)

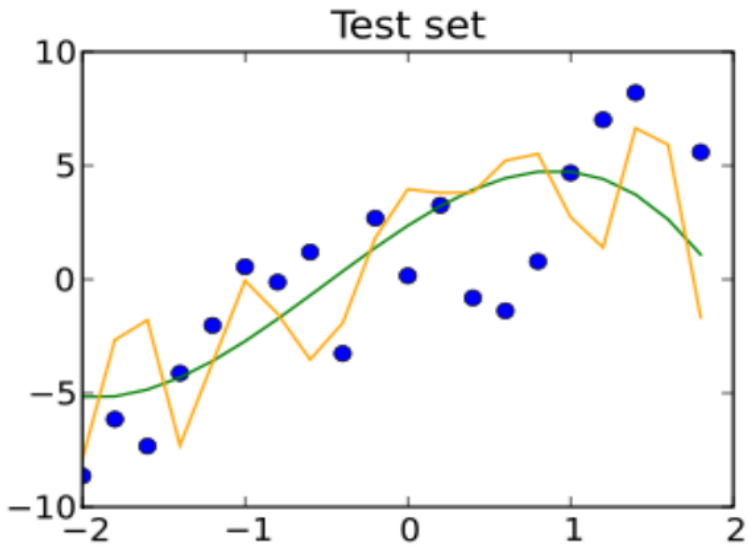


Step Decay ($\eta_0 = 0.1$, $r = 0.5$, $t_u = 10$)

Generalization and Overfitting

- **Learning:** minimizing classification loss $\mathcal{L}_{CE}$ over training set
  - Training set: sample representing data *vs* labels distributions
  - **Ultimate goal:** train a prediction function with low prediction error on the **true (unknown) data distribution**





$\mathcal{L}_{train} = 4$, $\mathcal{L}_{train} = 9$ $\mathcal{L}_{test} = 15$, $\mathcal{L}_{test} = 13$

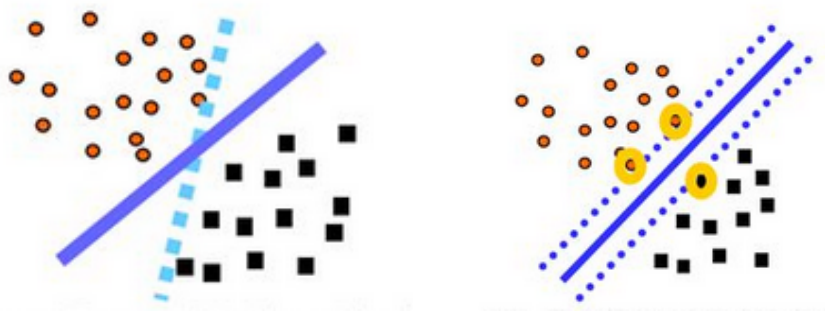
$\Rightarrow$ **Optimization $\neq$ Machine Learning!**
$\Rightarrow$ **Generalization / Overfitting!**

Regularization

- **Regularization:** improving generalization, *i.e.* test ($\neq$ *train*) performances
- Structural regularization: add ***Prior*** $R(\mathrm{w})$ in training objective:

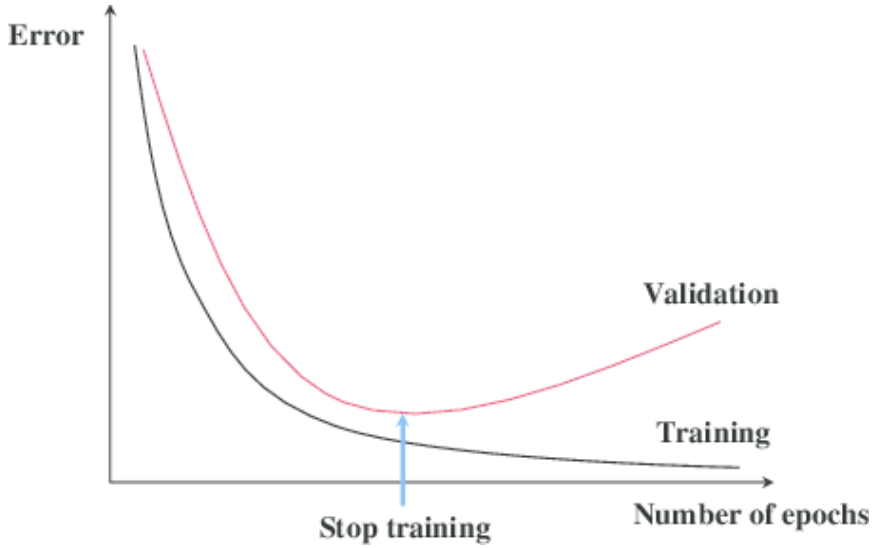
$$\mathcal{L}(\mathrm{w}) = \mathcal{L}_{CE}(\mathrm{w}) + \alpha R(\mathrm{w})$$

- L^2 **regularization: weight decay**, $R(\mathrm{w}) = ||\mathrm{w}||^2$
  - Commonly used in neural networks
  - Theoretical justifications, generalization bounds (SVM)
- Other possible $R(\mathrm{w})$: L^1 regularization, dropout, *etc*



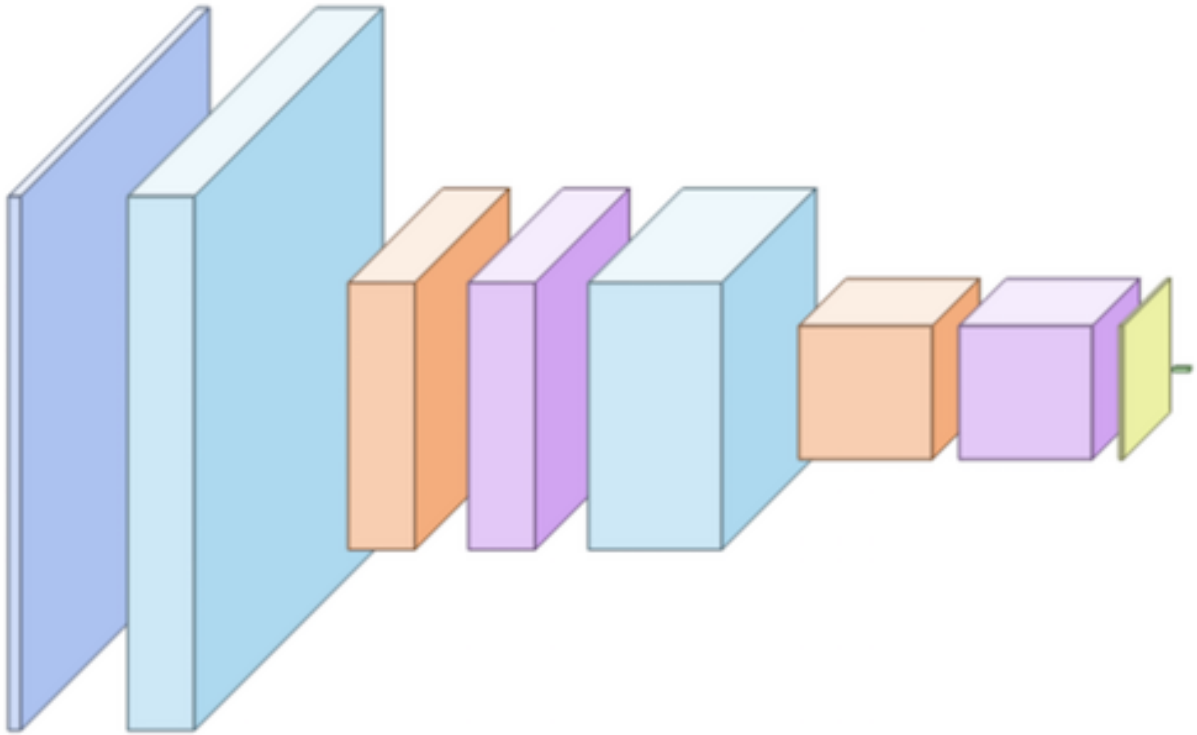
Regularization and hyper-parameters

- **Neural networks:** hyper-parameters to tune:
  - **Training parameters:** learning rate, weight decay, learning rate decay, # epochs, *etc*
  - **Architectural parameters:** number of layers, number neurones, non-linearity type, *etc*
- **Hyper-parameters tuning:** ⇒ improve generalization: estimate performances on a validation set



Neural networks: Conclusion

- Training issues at several levels: optimization, generalization, cross-validation
- **Limits of fully connected layers and Convolutional Neural Nets ? $\Rightarrow$ next course!**



References I


George Cybenko, *Approximation by superpositions of a sigmoidal function*, Mathematics of control, signals and systems **2** (1989), no. 4, 303–314.


Kunihiko Fukushima, *Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position*, Biological cybernetics **36** (1980), no. 4, 193–202.


Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel, *Backpropagation applied to handwritten zip code recognition*, Neural computation **1** (1989), no. 4, 541–551.


Warren S McCulloch and Walter Pitts, *A logical calculus of the ideas immanent in nervous activity*, The bulletin of mathematical biophysics **5** (1943), no. 4, 115–133.


F. Rosenblatt, *The perceptron: A probabilistic model for information storage and organization in the brain*, Psychological Review **65** (1958), no. 6, 386–408.


P. Werbos, *Beyond regression: New tools for prediction and analysis in the behavioral sciences*, Ph.D. thesis, Harvard University, 1974.