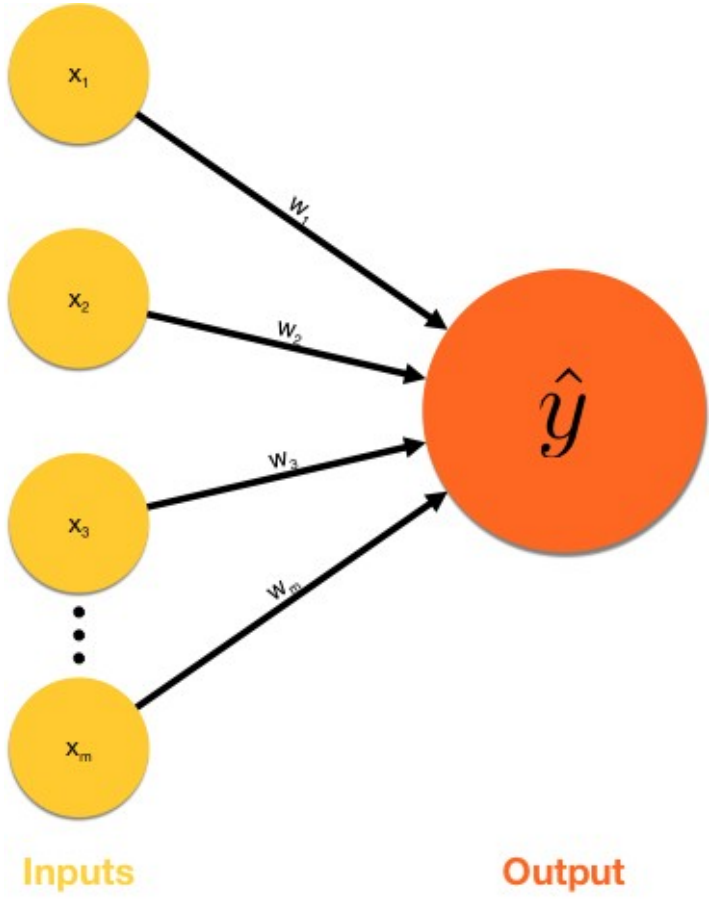
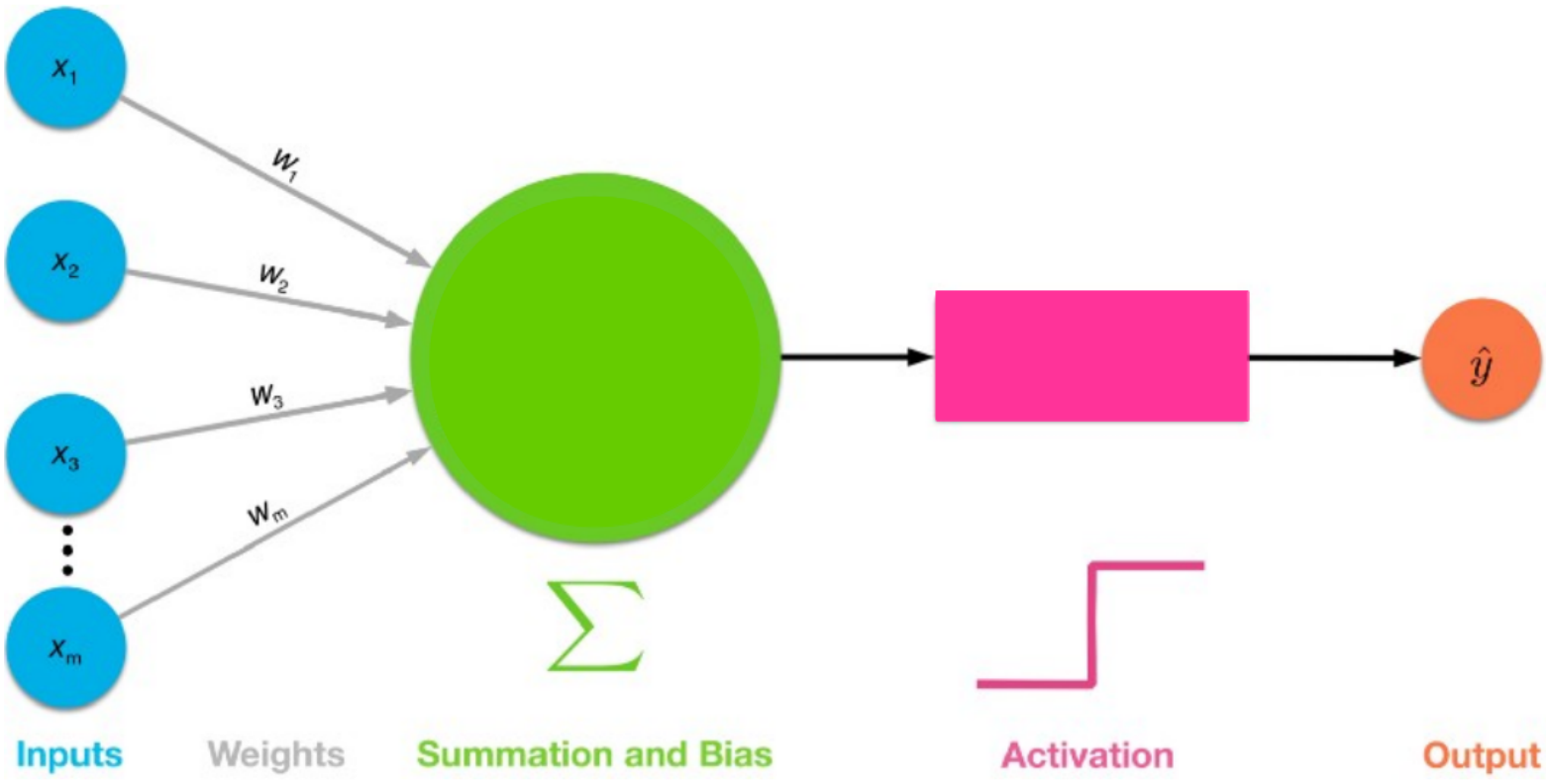
The Formal Neuron: 1943 [MP43]

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



The Formal Neuron: 1943 [MP43]

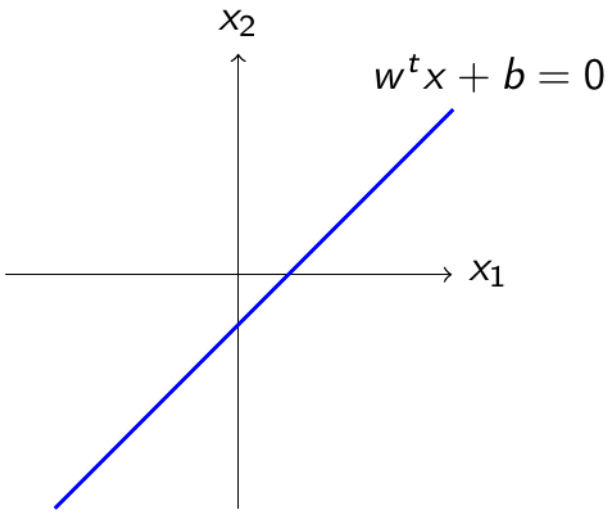
- Mapping from $\mathbf{x}$ to $\hat{y}$:
  1. Linear (affine) mapping: $s = \mathbf{w}^\top \mathbf{x} + b$
  2. Non-linear activation function: f: $\hat{y} = f(s)$



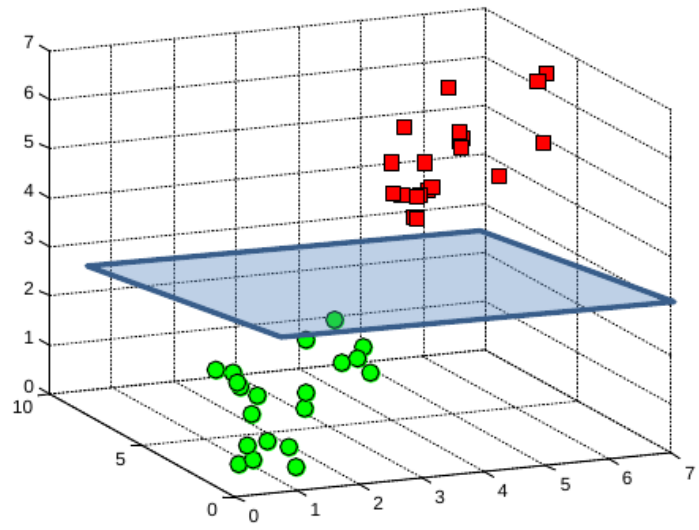
The Formal Neuron: Linear Mapping

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

2D hyperplane: line

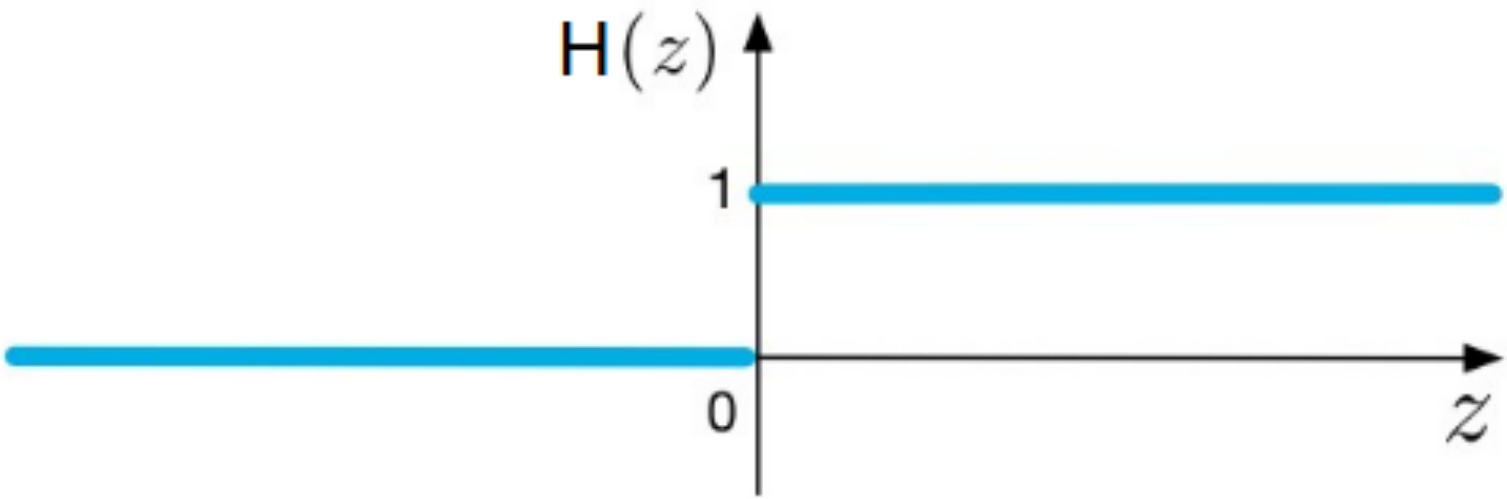


3D hyperplane: plane

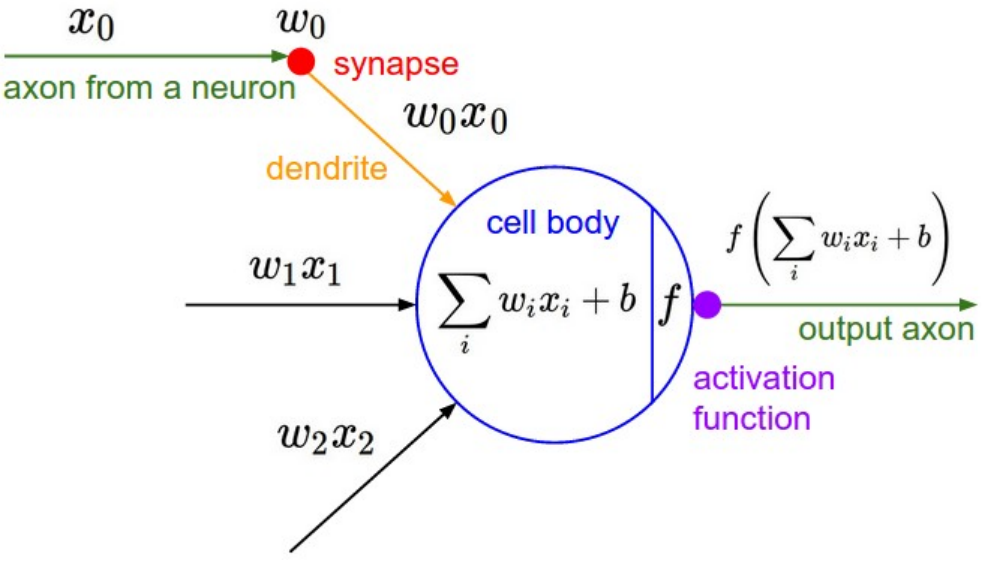


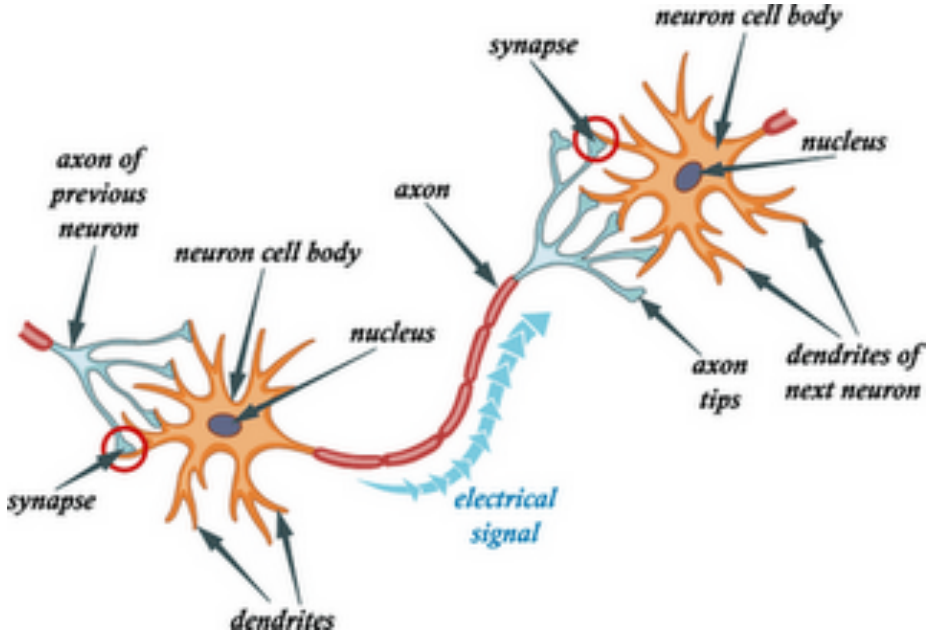
The Formal Neuron: Activation Function

- $\hat{y} = f(\mathbf{w}^\top \mathbf{x} + b)$, 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

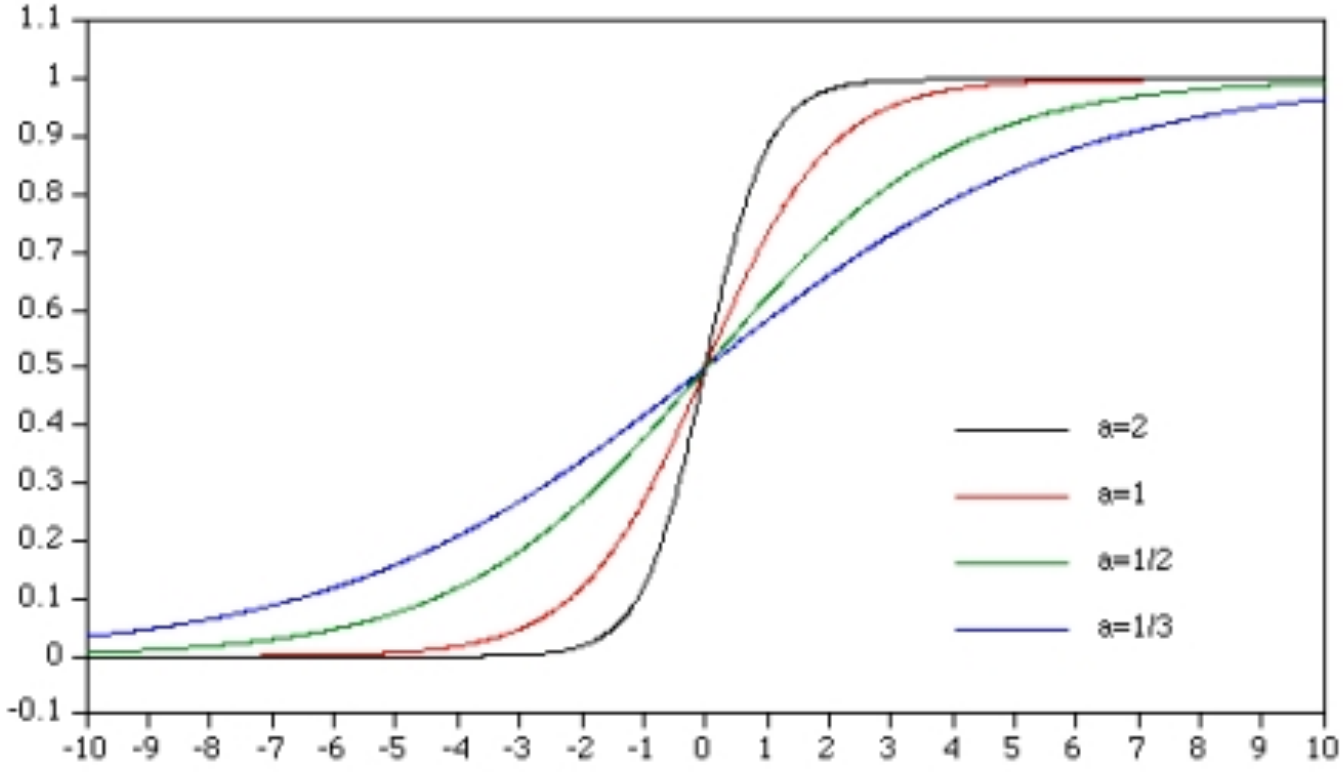




- Formal neuron, step activation H: $\hat{y} = H(\mathbf{w}^\top \mathbf{x} + b)$
  - $\hat{y} = 1$ (activated) $\Leftrightarrow \mathbf{w}^\top \mathbf{x} \geq -b$
  - $\hat{y} = 0$ (unactivated) $\Leftrightarrow \mathbf{w}^\top \mathbf{x} < -b$
- **Biological Neurons:** output activated $\Leftrightarrow$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

Sigmoid Activation Function

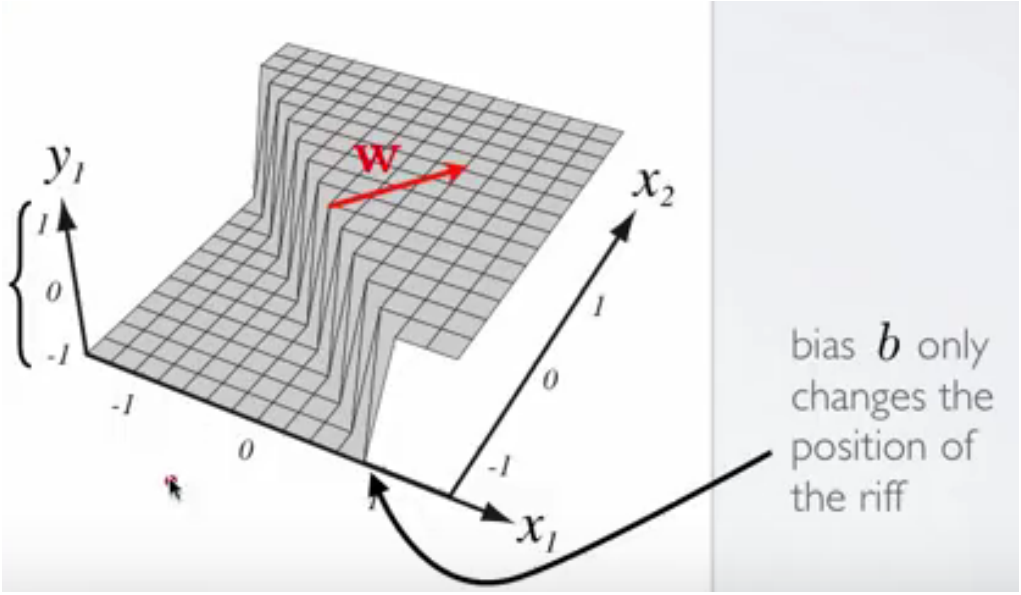
- Neuron output $\hat{y} = f(\mathbf{w}^\top \mathbf{x} + b)$, f activation function
- Sigmoid: $\sigma(z) = (1 + e^{-az})^{-1}$



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

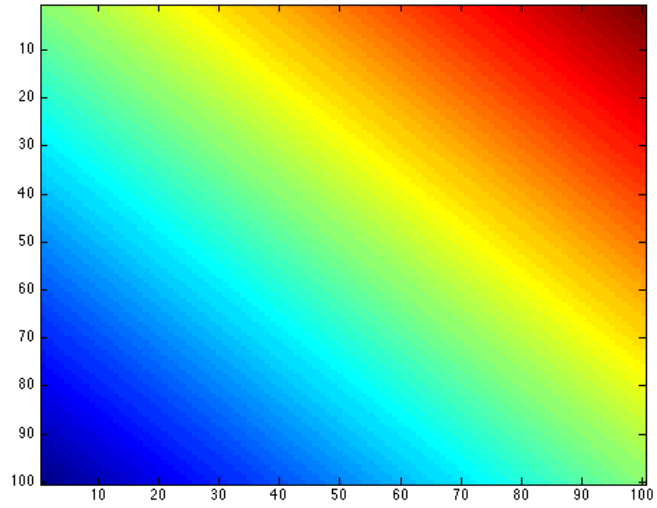
The Formal neuron: Application to Binary Classification

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



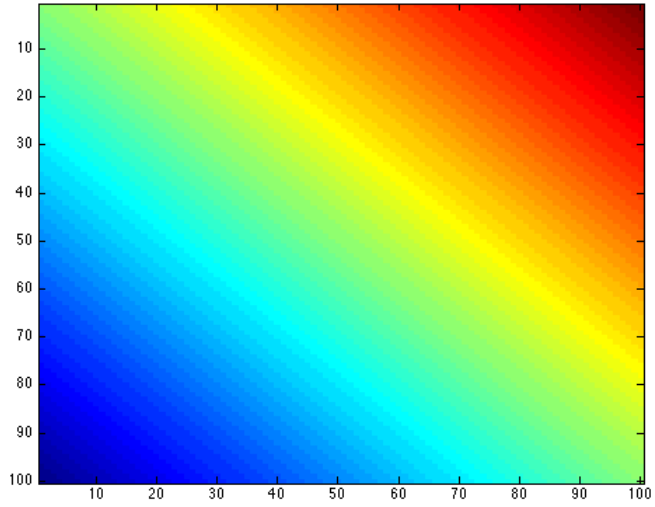
The Formal neuron: Toy Example for Binary Classification

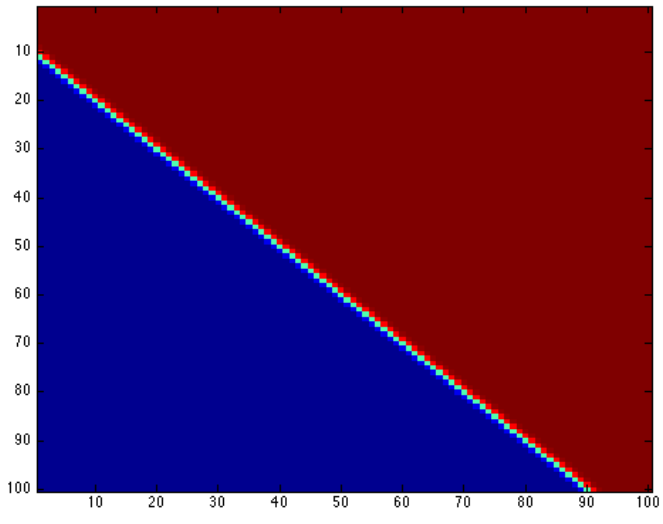
- 2d example: $m = 2$, $\mathbf{x} = \{x_1, x_2\} \in [-5; 5] \times [-5; 5]$
- Linear mapping: $\mathbf{w} = [1; 1]$ and $b = -2$
- Result of linear mapping : $s = \mathbf{w}^\top \mathbf{x} + b$

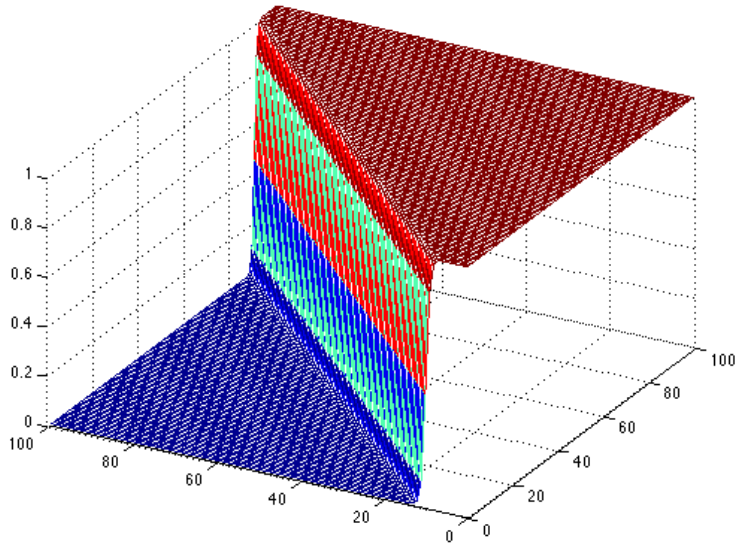


The Formal neuron: Toy Example for Binary Classification

- 2d example: $m = 2$, $\mathbf{x} = \{x_1, x_2\} \in [-5; 5] \times [-5; 5]$
- Linear mapping: $\mathbf{w} = [1; 1]$ and $b = -2$
- Result of linear mapping : $s = \mathbf{w}^\top \mathbf{x} + b$
- Sigmoid activation function: $\hat{y} = \left(1 + e^{-a(\mathbf{w}^\top \mathbf{x} + b)}\right)^{-1}$, $a = 10$

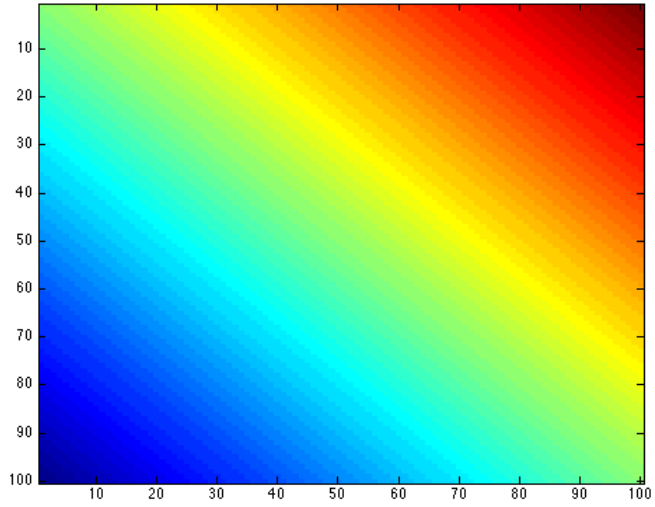


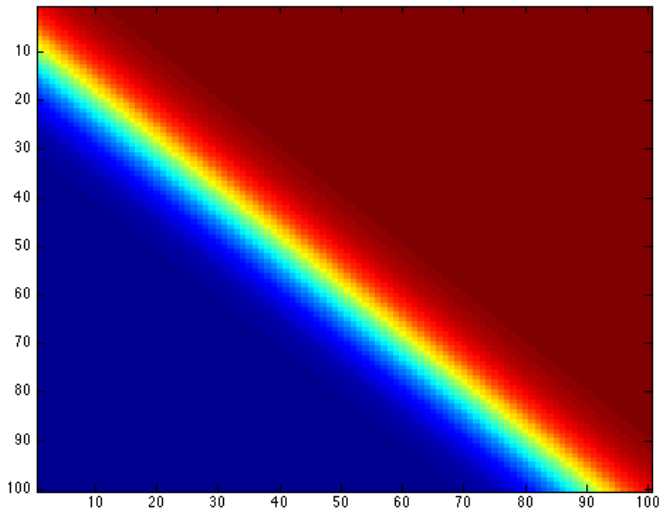


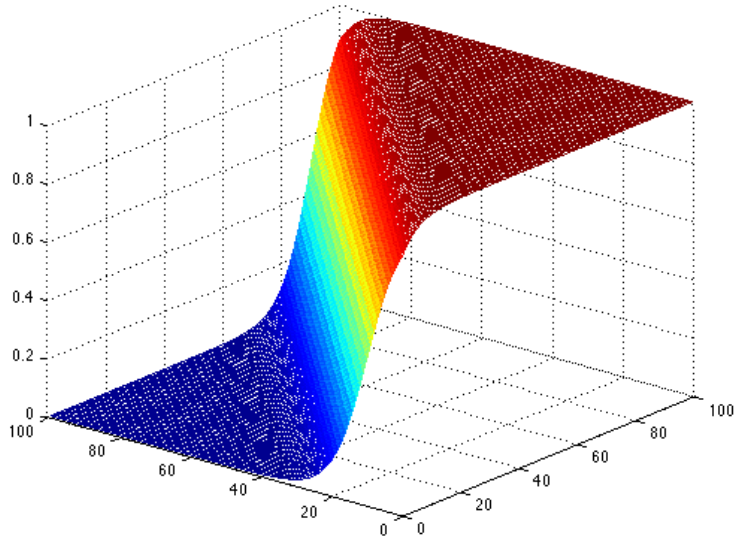


The Formal neuron: Toy Example for Binary Classification

- 2d example: $m = 2$, $\mathbf{x} = \{x_1, x_2\} \in [-5; 5] \times [-5; 5]$
- Linear mapping: $\mathbf{w} = [1; 1]$ and $b = -2$
- Result of linear mapping : $s = \mathbf{w}^\top \mathbf{x} + b$
- Sigmoid activation function: $\hat{y} = \left(1 + e^{-a(\mathbf{w}^\top \mathbf{x} + b)}\right)^{-1}$, $a = 1$

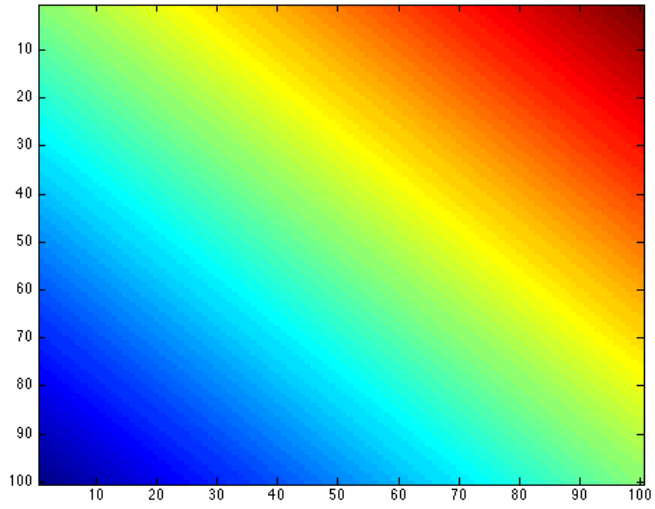


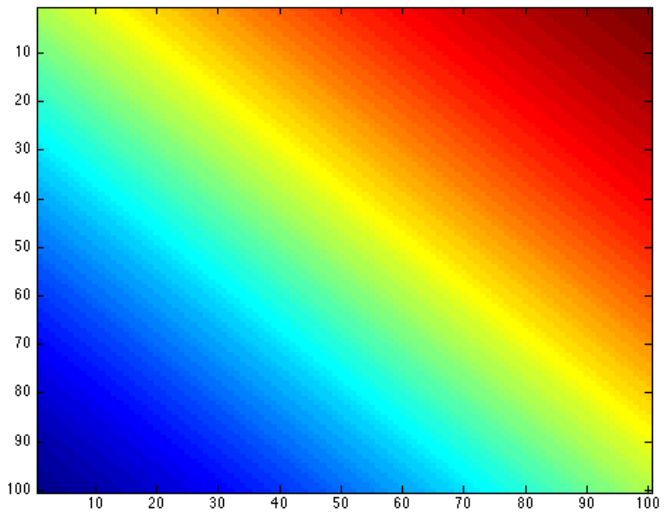


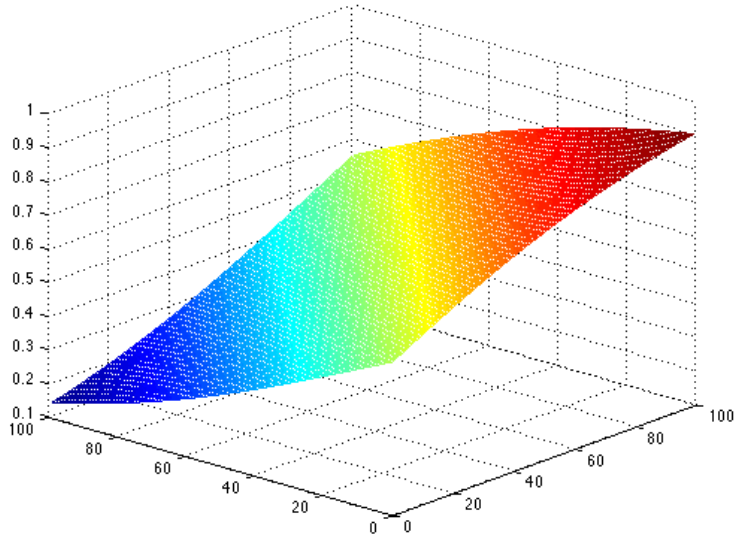


The Formal neuron: Toy Example for Binary Classification

- 2d example: $m = 2$, $\mathbf{x} = \{x_1, x_2\} \in [-5; 5] \times [-5; 5]$
- Linear mapping: $\mathbf{w} = [1; 1]$ and $b = -2$
- Result of linear mapping : $s = \mathbf{w}^\top \mathbf{x} + b$
- Sigmoid activation function: $\hat{y} = \left(1 + e^{-a(\mathbf{w}^\top \mathbf{x} + b)}\right)^{-1}$, $a = 0.1$

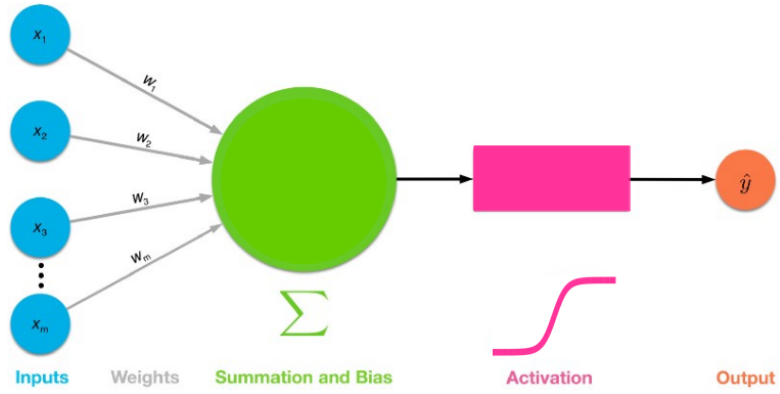


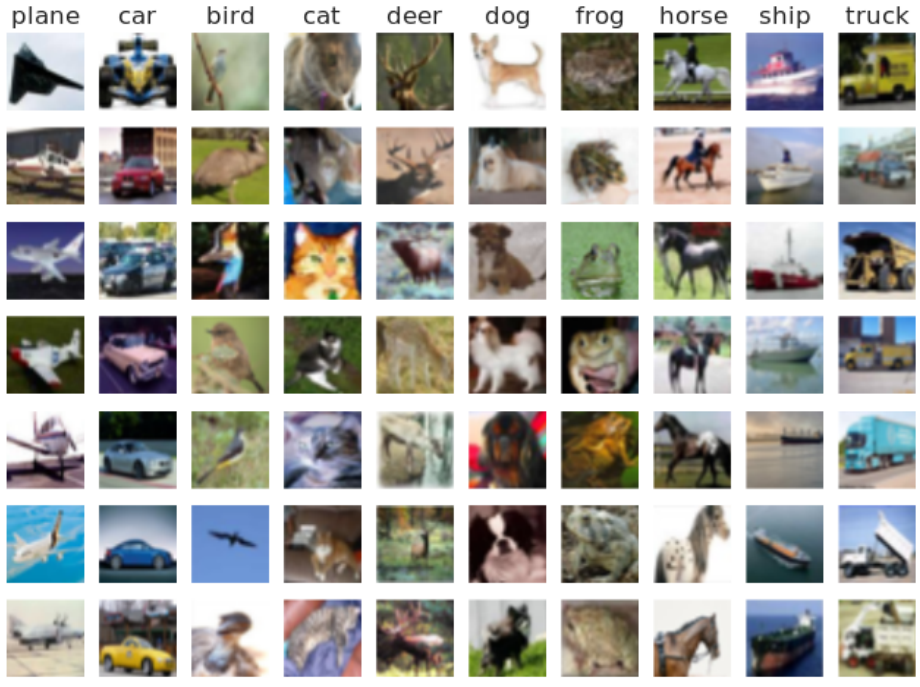




From Formal Neuron to Neural Networks

- Formal Neuron:
  1. A single scalar output
  2. Linear decision boundary for binary classification
- Single scalar output: limited for several tasks
  - Ex: multi-class classification, *e.g.* MNIST or CIFAR

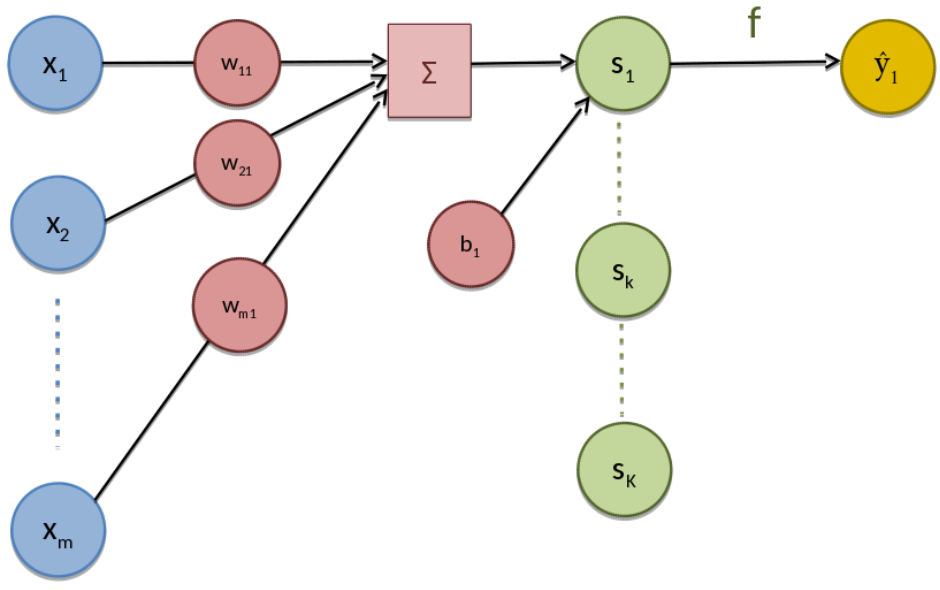




Perceptron and Multi-Class Classification

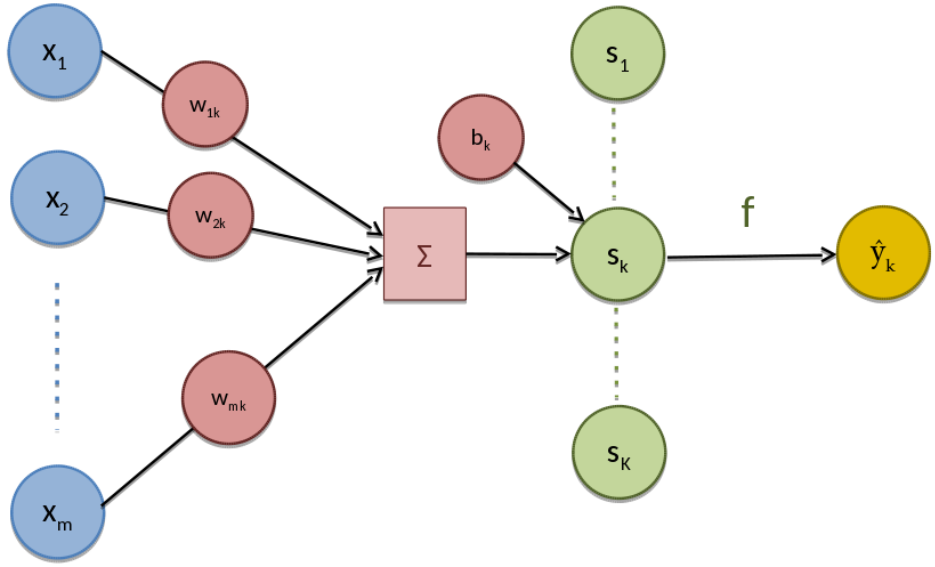
- Formal Neuron: limited to binary classification
- **Multi-Class Classification:** use several output neurons instead of a single one !
  $\Rightarrow$ **Perceptron**

- Input $\mathbf{x}$ in $\mathbb{R}^m$
- Output neuron $\hat{y_1}$ is a formal neuron:
  - Linear (affine) mapping: $s_1 = \mathbf{w_1}^\top \mathbf{x} + b_1$
  - Non-linear activation function: f: $\hat{y_1} = f(s_1)$
- Linear mapping parameters:
  - $\mathbf{w_1} = \{w_{11}, \ldots, w_{m1}\} \in \mathbb{R}^m$
  - $b_1 \in \mathbb{R}$



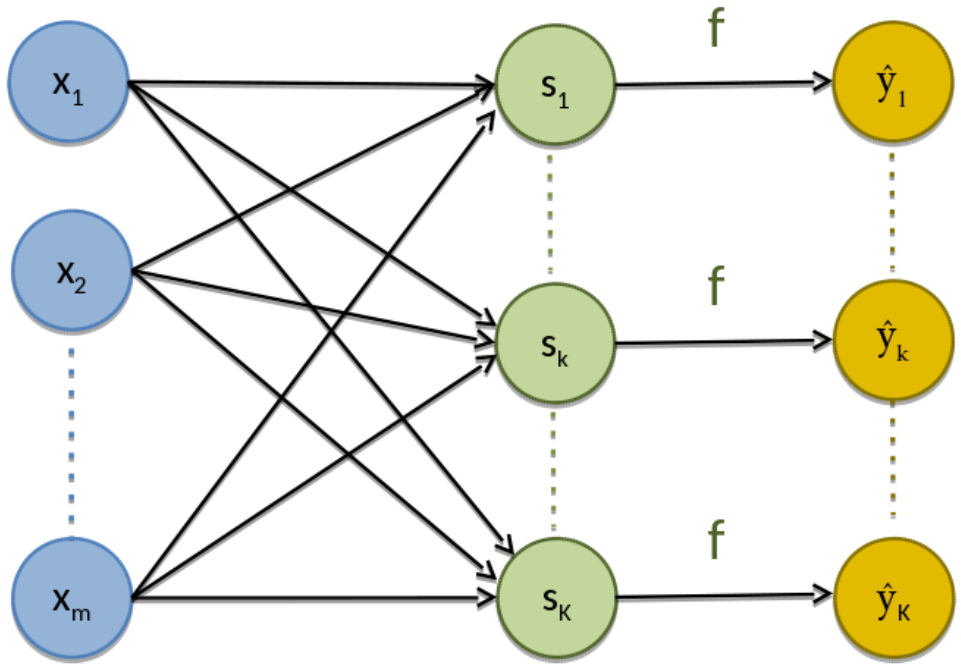
Perceptron and Multi-Class Classification

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

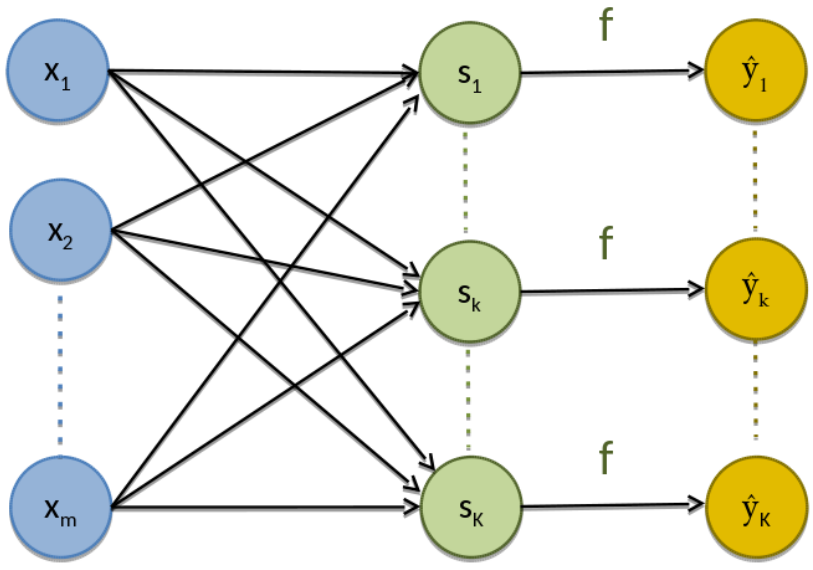


Perceptron and Multi-Class Classification

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



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
  - $\hat{y}_k \sim P(k/\mathbf{x}, \mathbf{w})$

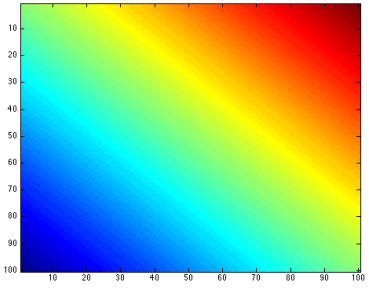
$\Rightarrow$ **Logistic Regression (LR) Model !**

2d Toy Example for Multi-Class Classification

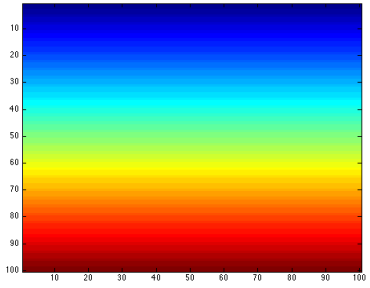
- $\mathbf{x} = \{x_1, x_2\} \in [-5; 5] \times [-5; 5]$, $\hat{y}$: 3 outputs (classes)

Linear mapping for each class:
$\mathbf{s}_k = \mathbf{w_k}^\top \mathbf{x} + b_k$

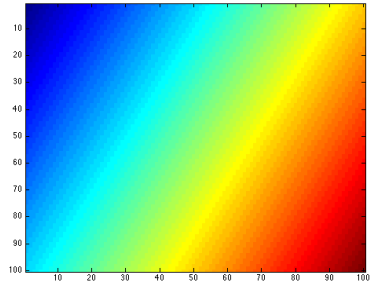
$\mathbf{w_1} = [\mathbf{1}; \mathbf{1}]$, $b_\mathbf{1} = -\mathbf{2}$



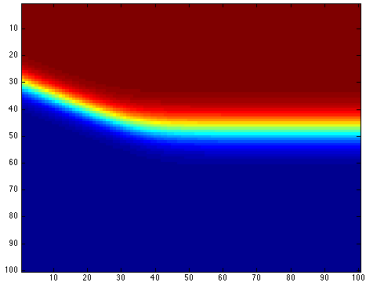
$\mathbf{w_2} = [\mathbf{0}; -\mathbf{1}]$, $b_\mathbf{2} = \mathbf{1}$

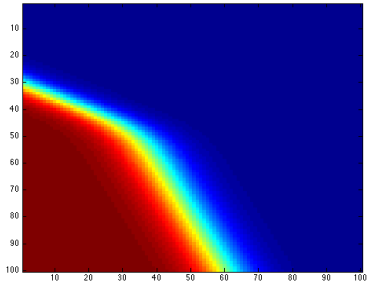


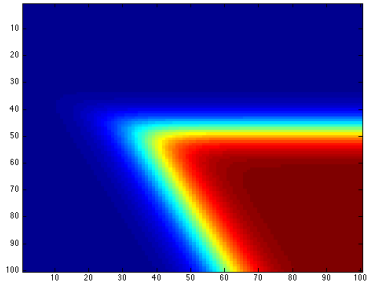
$\mathbf{w_3} = [\mathbf{1}; -\mathbf{0.5}]$, $b_\mathbf{3} = \mathbf{10}$



Soft-max output:
$P(k/\mathbf{x}, \mathbf{W})$





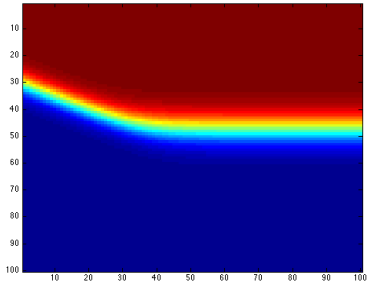


2d Toy Example for Multi-Class Classification

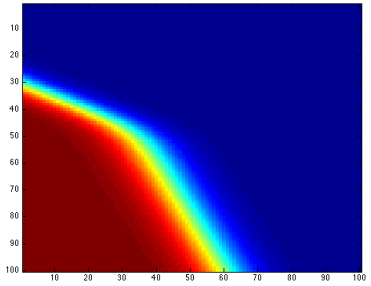
- $\mathbf{x} = \{x_1, x_2\} \in [-5; 5] \times [-5; 5]$, $\hat{y}$: 3 outputs (classes)

Soft-max output:
$P(k/\mathbf{x}, \mathbf{W})$

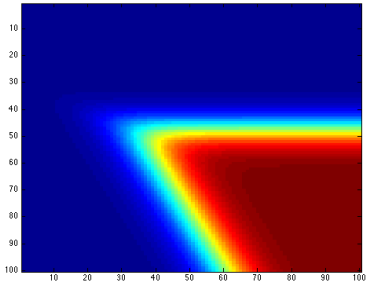
$\mathbf{w_1} = [\mathbf{1}; \mathbf{1}]$, $b_\mathbf{1} = \mathbf{-2}$



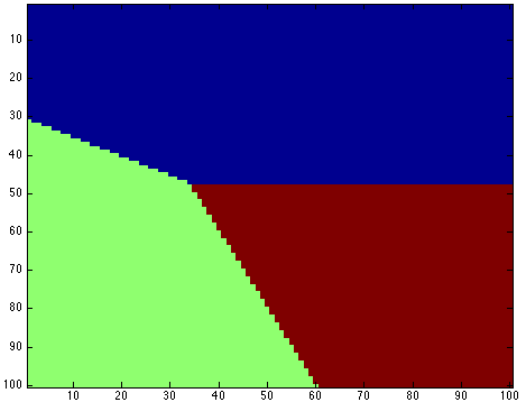
$\mathbf{w_2} = [\mathbf{0}; \mathbf{-1}]$, $b_\mathbf{2} = \mathbf{1}$



$\mathbf{w_3} = [\mathbf{1}; \mathbf{-0.5}]$, $b_\mathbf{3} = \mathbf{10}$



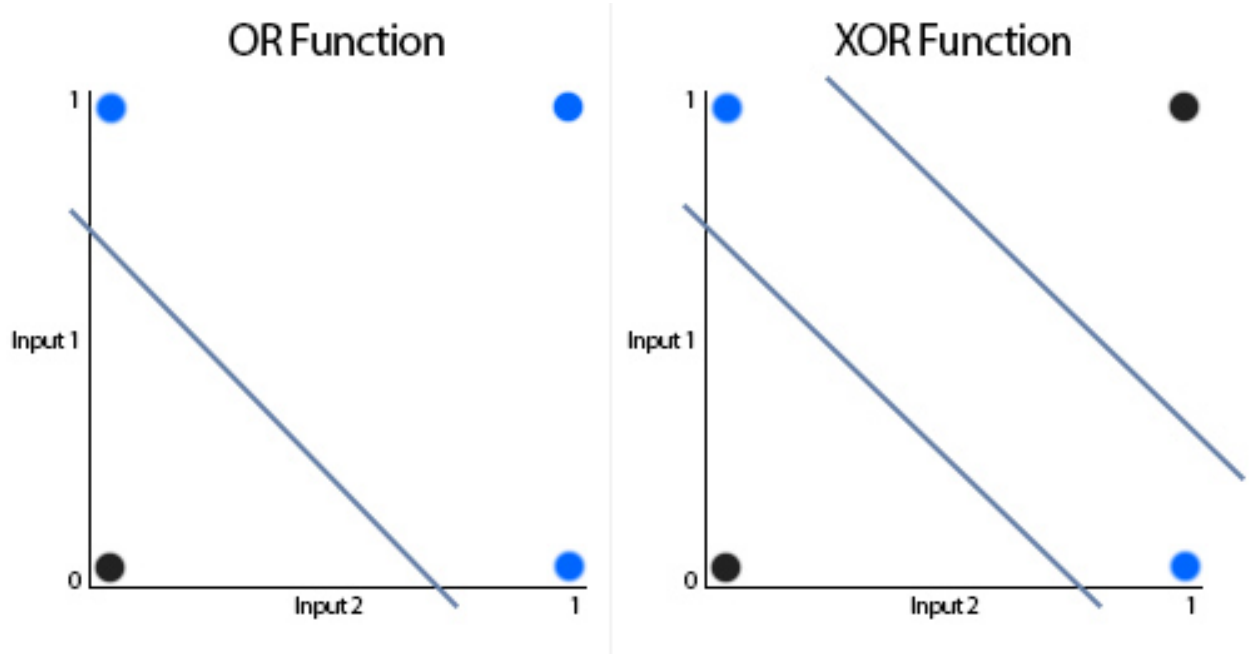
Class Prediction:
$k^* = \arg\max_k P(k/\mathbf{x}, \mathbf{W})$



Beyond Linear Classification

X-OR Problem

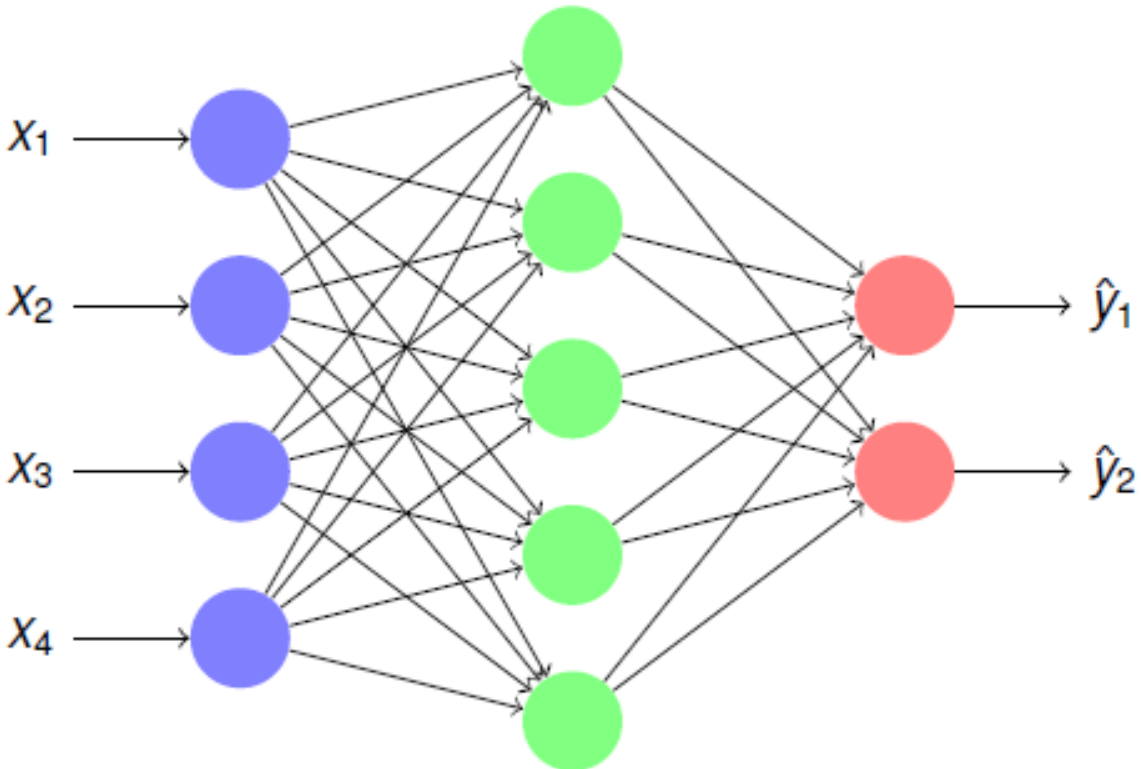
- Logistic Regression (LR): NN with 1 input layer & 1 output layer
- LR: limited to linear decision boundaries
- **X-OR:** NOT 1 and 2 OR NOT 2 AND 1
  - **X-OR: Non linear decision function**



Beyond Linear Classification

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

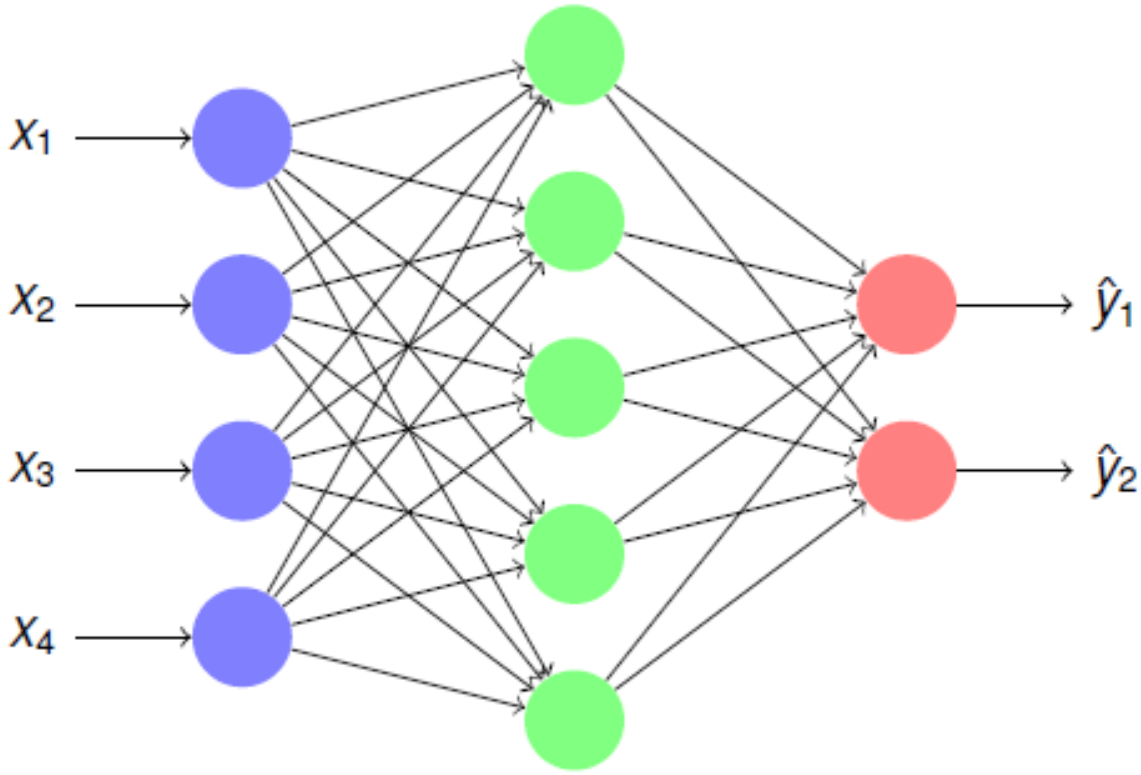
- Input $\mathbf{x}$ in $\mathbb{R}^m$, *e.g.* $m = 4$
- Output $\hat{\mathbf{y}}$ in $\mathbb{R}^K$ (K # classes), *e.g.* $K = 2$
- **Hidden layer h in $\mathbb{R}^L$**



Multi-Layer Perceptron

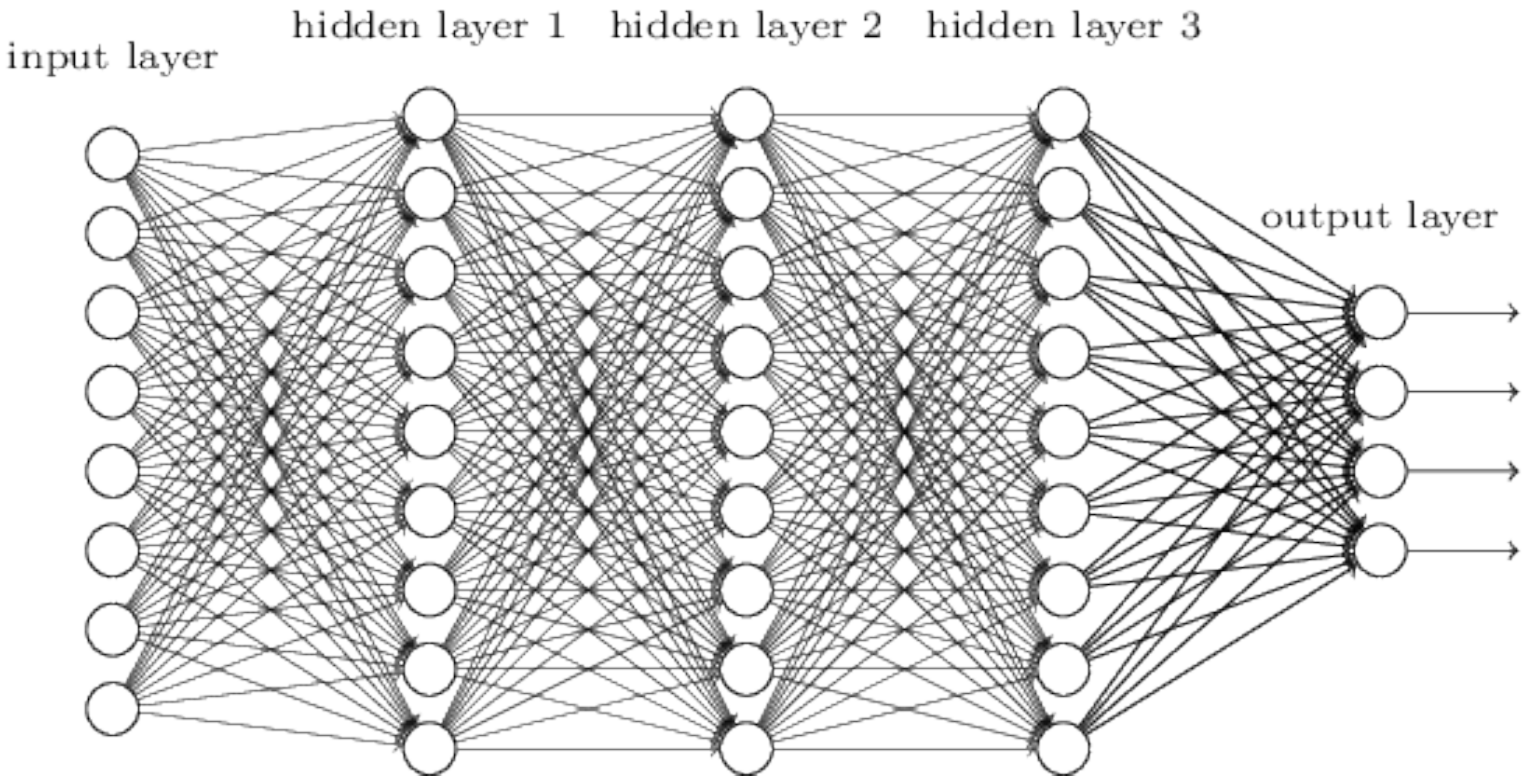
- **Hidden layer h**: **x** projection to a new space $\mathbb{R}^L$
- Neural Net with ≥ 1 hidden layer: **Multi-Layer Perceptron (MLP)**

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



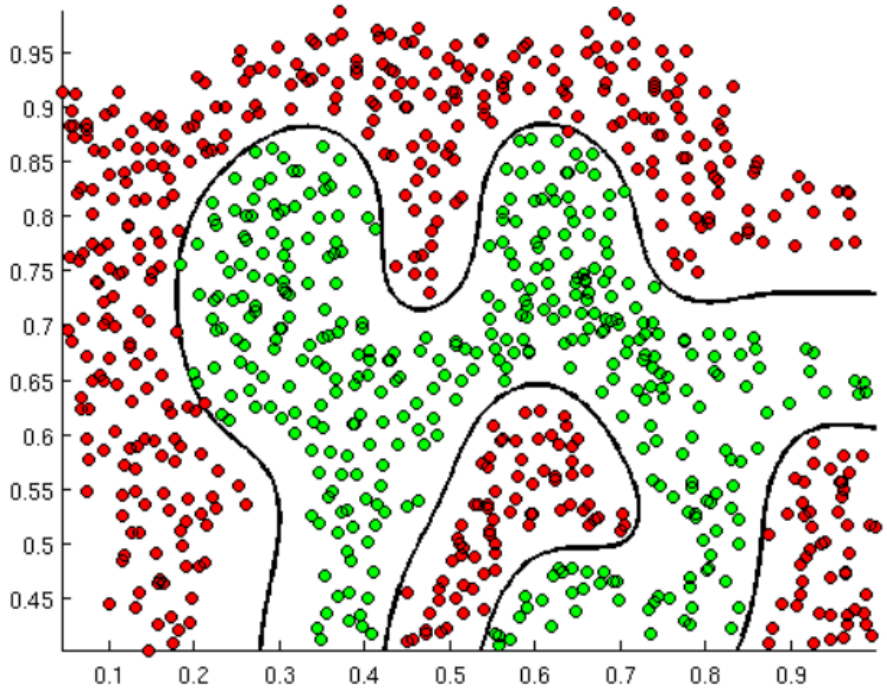
Deep Neural Networks

- Adding more hidden layers: Deep Neural Networks (DNN) $\Rightarrow$ **Basis of Deep Learning**
- Each layer $\mathbf{h}^{l}$ projects layer $\mathbf{h}^{l-1}$ 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

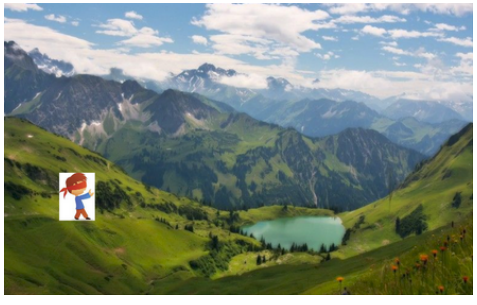


- Visualize prediction from fixed model parameters
- Reverse problem: **Supervised Learning**

Outline

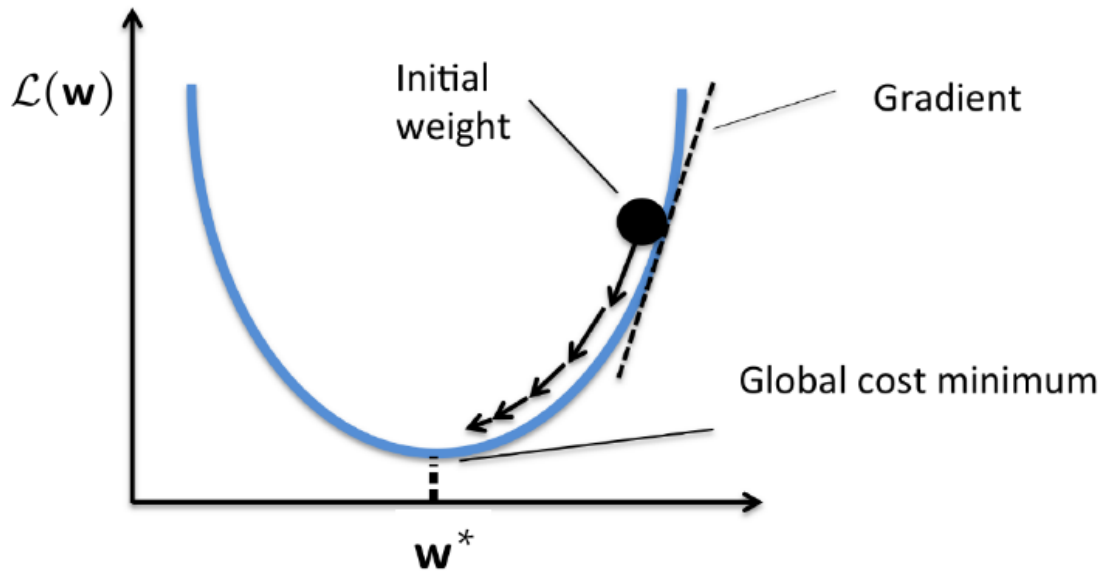
Training Multi-Layer Perceptron (MLP)

- Input $\mathbf{x}$, output $\mathbf{y}$
- A parametrized (w) model $\mathbf{x} \Rightarrow \mathbf{y}$: $f_{\mathbf{w}}(\mathbf{x}_i) = \hat{\mathbf{y}}_i$
- Supervised context:
  - Training set $\mathcal{A} = \left\{(\mathbf{x}_i, \mathbf{y}_i^*)\right\}_{i \in \{1,2,\ldots,N\}}$
  - Loss function $\ell(\hat{\mathbf{y}}_i, \mathbf{y}_i^*)$ for each annotated pair $(\mathbf{x}_i, \mathbf{y}_i^*)$
  - Goal: Minimizing average loss $\mathcal{L}$ over training set: $\mathcal{L}(w) = \frac{1}{N} \sum_{i=1}^{N} \ell(\hat{\mathbf{y}}_i, \mathbf{y}_i^*)$
- Assumptions: parameters $\mathbf{w} \in \mathbb{R}^d$ continuous, $\mathcal{L}$ differentiable
- Gradient $\nabla_{\mathbf{w}} = \frac{\partial \mathcal{L}}{\partial \mathbf{w}}$: steepest direction to decrease loss $\mathcal{L}(w)$



MLP Training

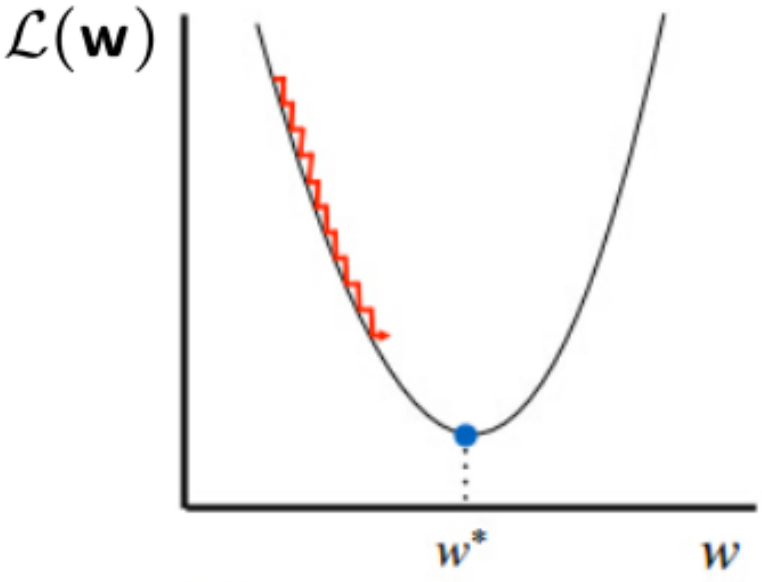
- Gradient descent algorithm:
  - Initialyze parameters $\mathbf{w}$
  - Update: $\boxed{\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \frac{\partial \mathcal{L}}{\partial \mathbf{w}}}$
  - Until convergence, *e.g.* $||\nabla_{\mathbf{w}}||^2 \approx 0$



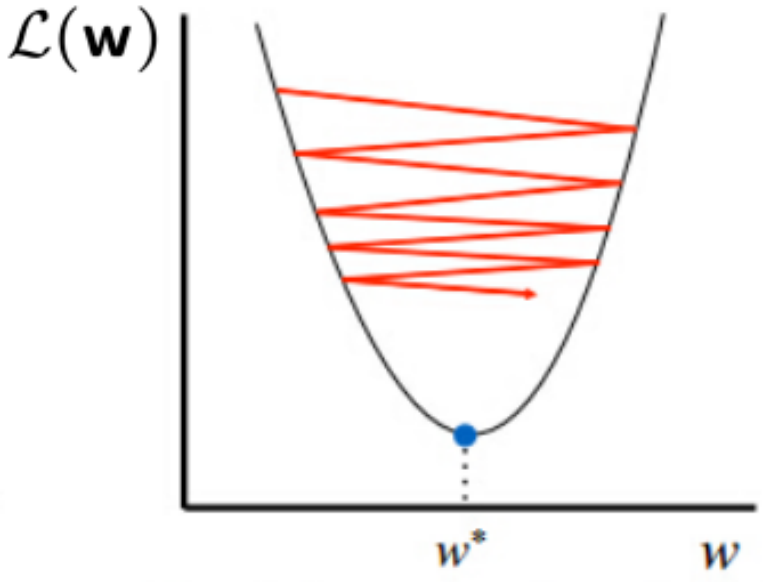
Gradient Descent

Update rule: $\boxed{\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \frac{\partial \mathcal{L}}{\partial \mathbf{w}}}$ η **learning rate**

- **Convergence ensured ?** $\Rightarrow$ provided a "well chosen" learning rate η



Too small: converge very slowly

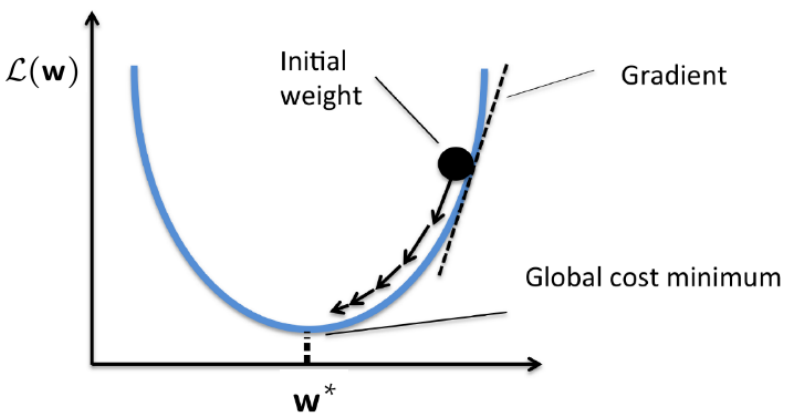


Too big: overshoot and even diverge

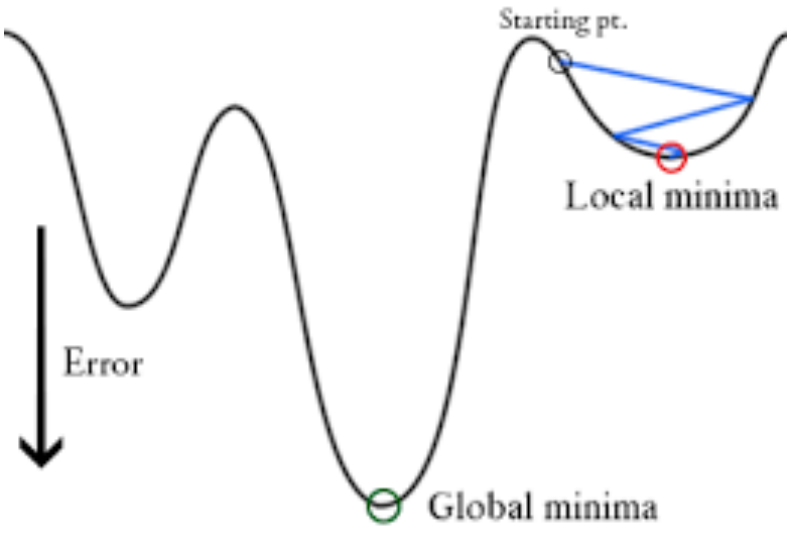
Gradient Descent

Update rule: $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \frac{\partial \mathcal{L}}{\partial \mathbf{w}}$

- **Global minimum ?**
  $\Rightarrow$ **convex** a) *vs* **non convex** b) loss $\mathcal{L}(\mathbf{w})$



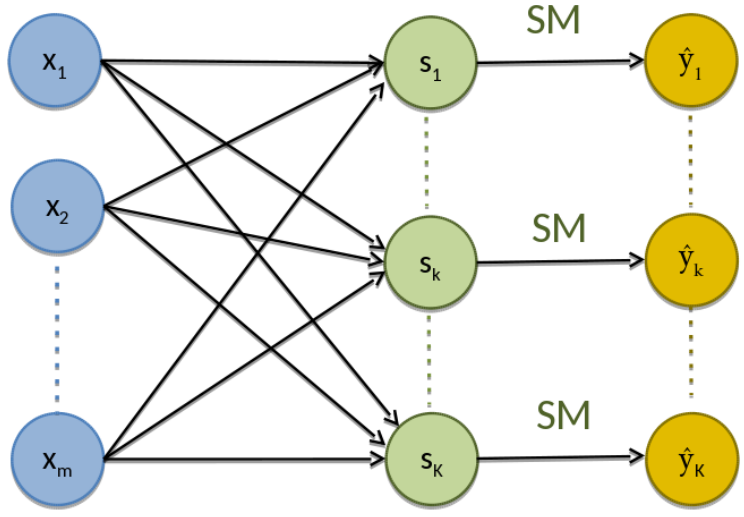
a) Convex function



a) Non convex function

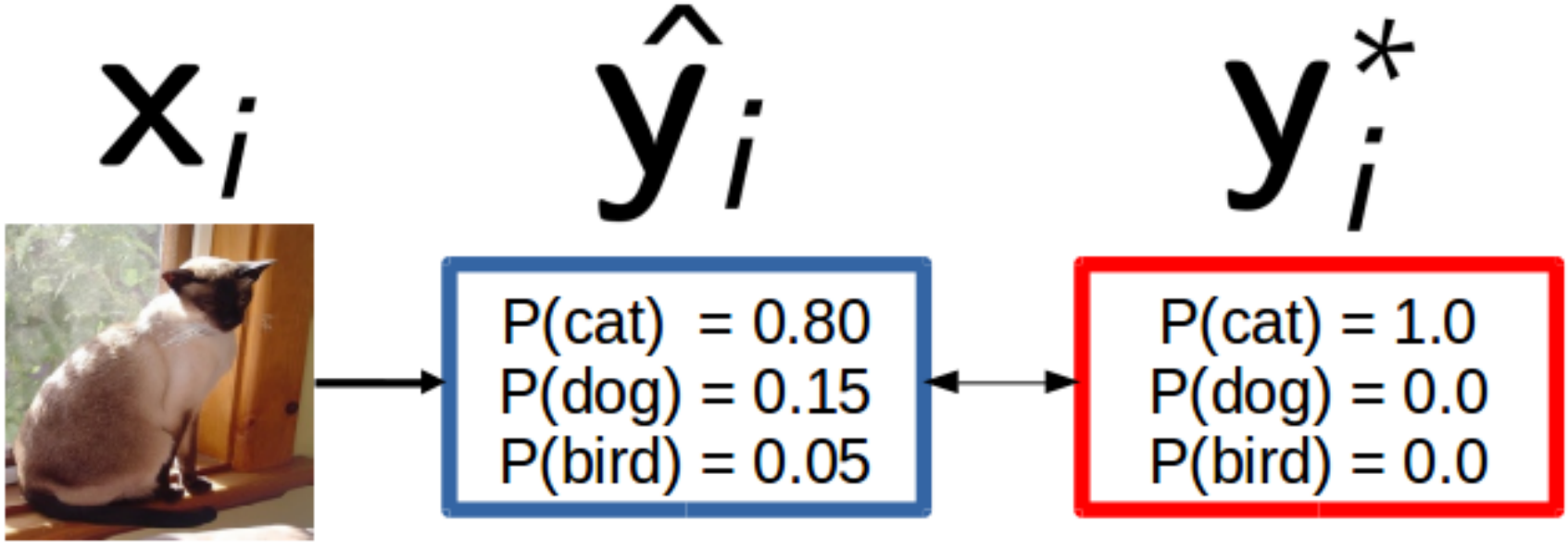
Supervised Learning: Multi-Class Classification

- Logistic Regression for multi-class classification
- $\mathbf{s}_i = \mathbf{x}_i\mathbf{W} + \mathbf{b}$
- Soft-Max (SM): $\hat{\mathbf{y}}_k \sim P(k/\mathbf{x_i}, \mathbf{W}, \mathbf{b}) = \frac{e^{\mathbf{s}_k}}{\sum_{k'=1}^{K} e^{\mathbf{s}_{k'}}}$
- Supervised loss function: $\mathcal{L}(\mathbf{W}, \mathbf{b}) = \frac{1}{N} \sum_{i=1}^{N} \ell(\hat{\mathbf{y}}_i, \mathbf{y}_i^*)$



1. $\mathbf{y} \in \{1; 2; ...; K\}$
2. $\hat{\mathbf{y}}_i = \arg\max_k P(k/\mathbf{x}_i, \mathbf{W}, \mathbf{b})$
3. $\ell_{0/1}(\hat{\mathbf{y}}_i, \mathbf{y}_i^*) = \begin{cases} 1 & \text{if } \hat{\mathbf{y}}_i \neq \mathbf{y}_i^* \\ 0 & \text{otherwise} \end{cases}$: **0/1 loss**

Logistic Regression Training Formulation



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

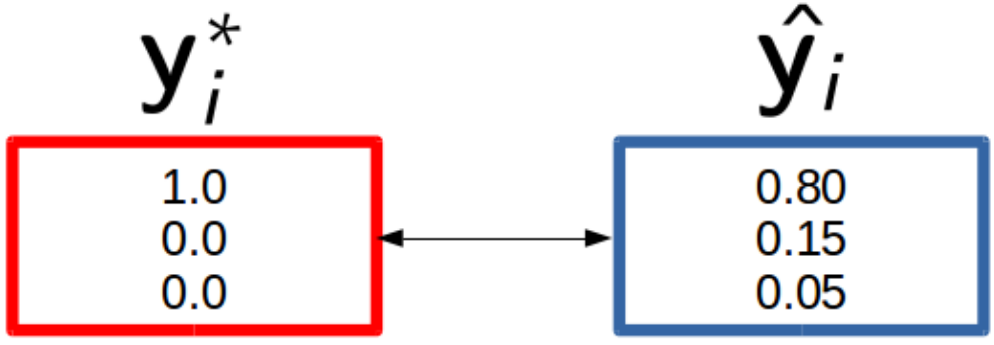
$$y_{c,i}^* = \begin{cases} 1 & \text{if c is the groud truth class for } \mathbf{x}_i \\ 0 & \text{otherwise} \end{cases}$$

Logistic Regression Training Formulation

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

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

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



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

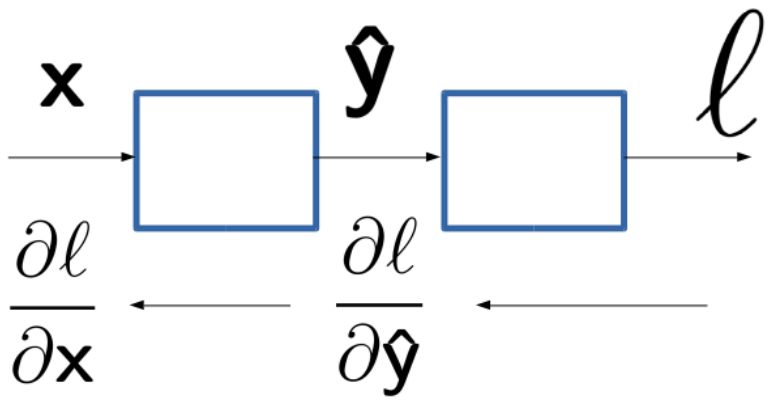
Logistic Regression Training

- $\mathcal{L}_{CE}(\mathbf{W}, \mathbf{b}) = \frac{1}{N} \sum_{i=1}^{N} \ell_{CE}(\hat{\mathbf{y}}_i, \mathbf{y}_i^*) = -\frac{1}{N} \sum_{i=1}^{N} log(\hat{y}_{c^*,i})$
- ℓ_{CE} smooth convex upper bound of $\ell_{0/1}$
  $\Rightarrow$ **gradient descent optimization**
- Gradient descent: $\mathbf{W}^{(t+1)} = \mathbf{W}^{(t)} - \eta \frac{\partial \mathcal{L}_{CE}}{\partial \mathbf{W}}$ $\quad (\mathbf{b}^{(t+1)} = \mathbf{b}^{(t)} - \eta \frac{\partial \mathcal{L}_{CE}}{\partial \mathbf{b}})$
- **MAIN CHALLENGE: computing** $\frac{\partial \mathcal{L}_{CE}}{\partial \mathbf{W}} = \frac{1}{N} \sum_{i=1}^{N} \frac{\partial \ell_{CE}}{\partial \mathbf{W}}$?

  $\Rightarrow$ **Key Property:** chain rule $\frac{\partial \mathbf{x}}{\partial z} = \frac{\partial \mathbf{x}}{\partial \mathbf{y}} \frac{\partial \mathbf{y}}{\partial z}$

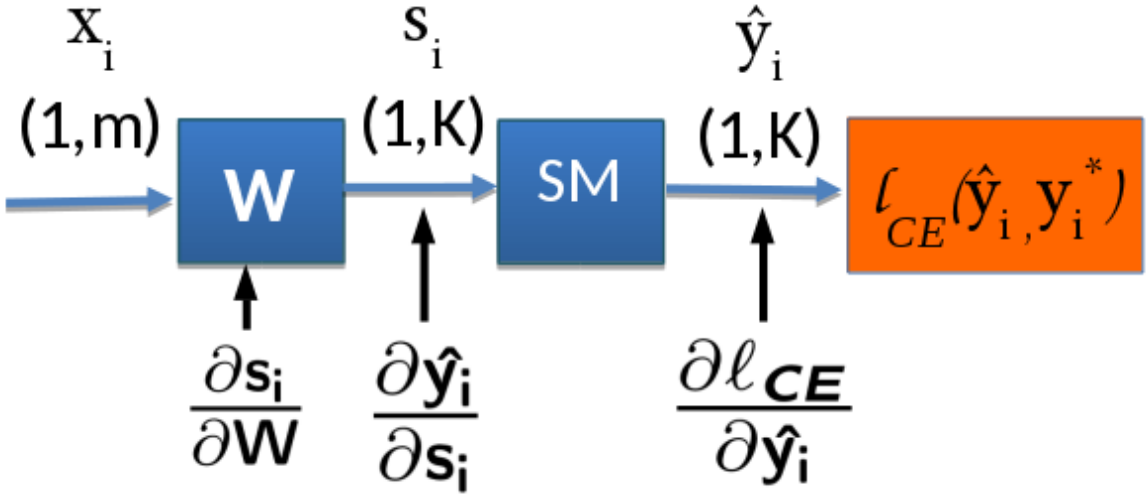
  $\Rightarrow$ **Backpropagation of gradient error!**

Chain Rule



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

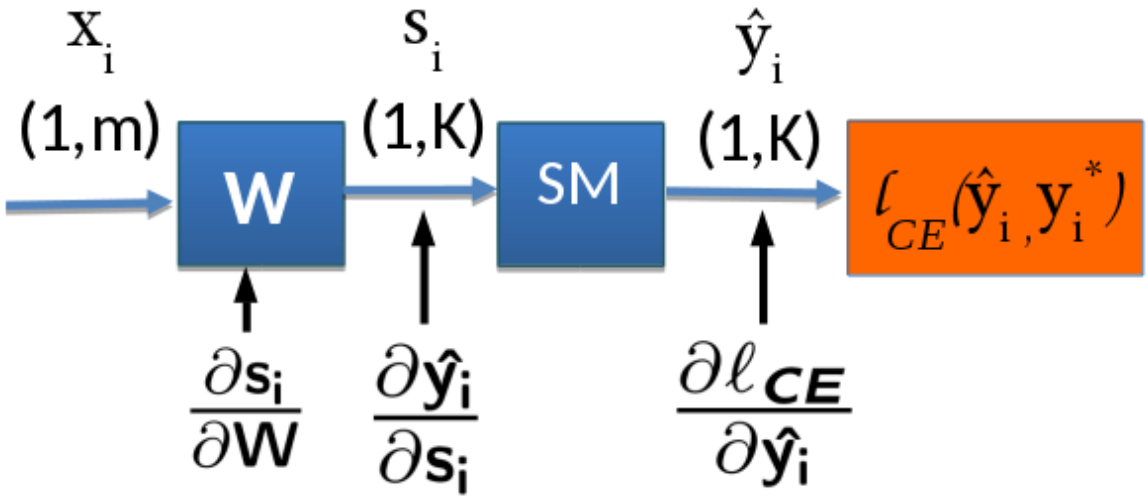
- Logistic regression: $\frac{\partial \ell_{CE}}{\partial \mathbf{W}} = \frac{\partial \ell_{CE}}{\partial \hat{\mathbf{y}}_i} \frac{\partial \hat{\mathbf{y}}_i}{\partial \mathbf{s}_i} \frac{\partial \mathbf{s}_i}{\partial \mathbf{W}}$



Logistic Regression Training: Backpropagation

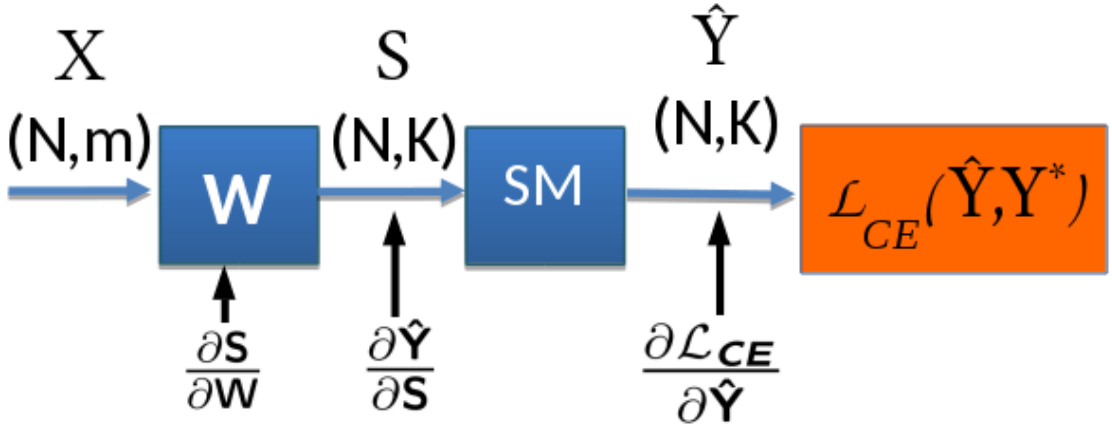
$$\frac{\partial \ell_{CE}}{\partial \mathbf{W}} = \frac{\partial \ell_{CE}}{\partial \hat{\mathbf{y}}_\mathbf{i}} \frac{\partial \hat{\mathbf{y}}_\mathbf{i}}{\partial \mathbf{s}_\mathbf{i}} \frac{\partial \mathbf{s}_\mathbf{i}}{\partial \mathbf{W}}, \ \ell_{CE}(\hat{\mathbf{y}}_i, \mathbf{y}_i^*) =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

- $\frac{\partial \ell_{CE}}{\partial \hat{\mathbf{y}}_\mathbf{i}} = \frac{-1}{\hat{y}_{c^*,i}} = \frac{-1}{\hat{\mathbf{y}}_\mathbf{i}} \odot \delta_{\mathbf{c},\mathbf{c}^*}$
- $\frac{\partial \ell_{CE}}{\partial \mathbf{s}_\mathbf{i}} = \hat{\mathbf{y}}_\mathbf{i} - \mathbf{y}_\mathbf{i}^* = \delta_\mathbf{i}^\mathbf{y}$
- $\frac{\partial \ell_{CE}}{\partial \mathbf{W}} = \mathbf{x}_\mathbf{i}{}^T \delta_\mathbf{i}^\mathbf{y}$



Logistic Regression Training: Backpropagation

- Whole dataset: data matrix $\mathbf{X}$ $(N \times m)$, label matrix $\hat{\mathbf{Y}}$, $\mathbf{Y}^*$ $(N \times K)$
- $\mathcal{L}_{CE}(\mathbf{W}, \mathbf{b}) = -\frac{1}{N} \sum_{i=1}^{N} log(\hat{y}_{c^*,i})$, $\frac{\partial \mathcal{L}_{CE}}{\partial \mathbf{W}} = \frac{\partial \mathcal{L}_{CE}}{\partial \hat{\mathbf{Y}}} \frac{\partial \hat{\mathbf{Y}}}{\partial \mathbf{S}} \frac{\partial \mathbf{S}}{\partial \mathbf{W}}$

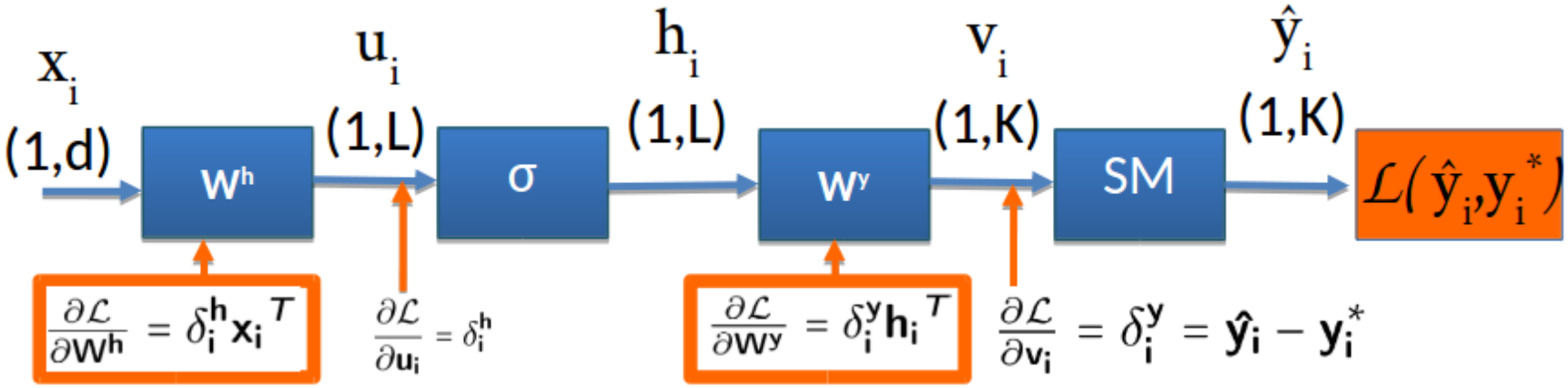


- $\boxed{\frac{\partial \mathcal{L}_{CE}}{\partial \mathbf{s}} = \hat{\mathbf{Y}} - \mathbf{Y}^* = \Delta^y}$
- $\boxed{\frac{\partial \mathcal{L}_{CE}}{\partial \mathbf{W}} = \mathbf{X}^T \Delta^y}$

Perceptron Training: Backpropagation

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

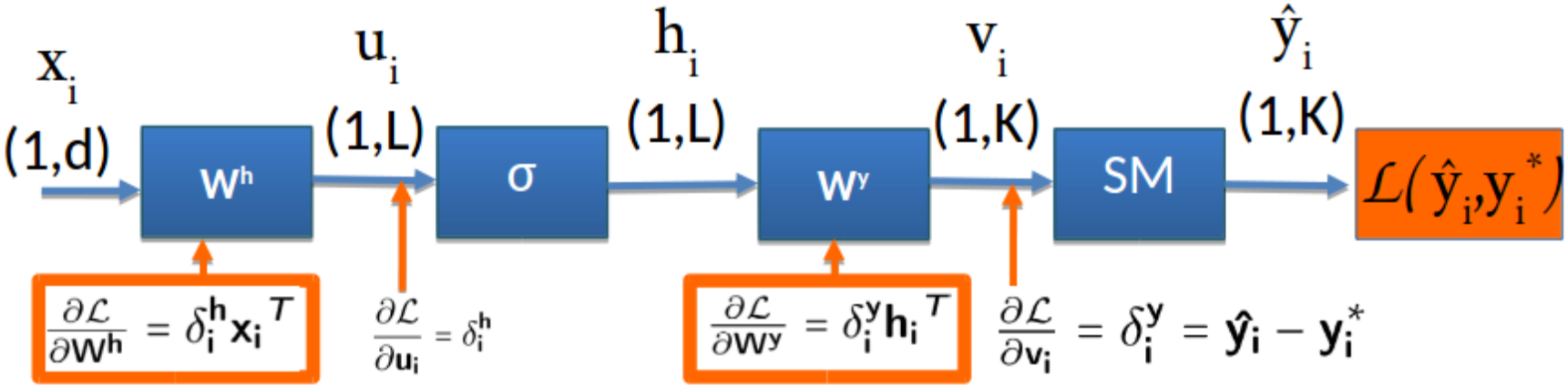
  $\Rightarrow$ computing $\boxed{\frac{\partial \mathcal{L}_{CE}}{\partial \mathbf{W}^y} = \frac{1}{N}\sum_{i=1}^{N} \frac{\partial \ell_{CE}}{\partial \mathbf{W}^y}}$ and $\boxed{\frac{\partial \mathcal{L}_{CE}}{\partial \mathbf{W}^h} = \frac{1}{N}\sum_{i=1}^{N} \frac{\partial \ell_{CE}}{\partial \mathbf{W}^h}}$



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

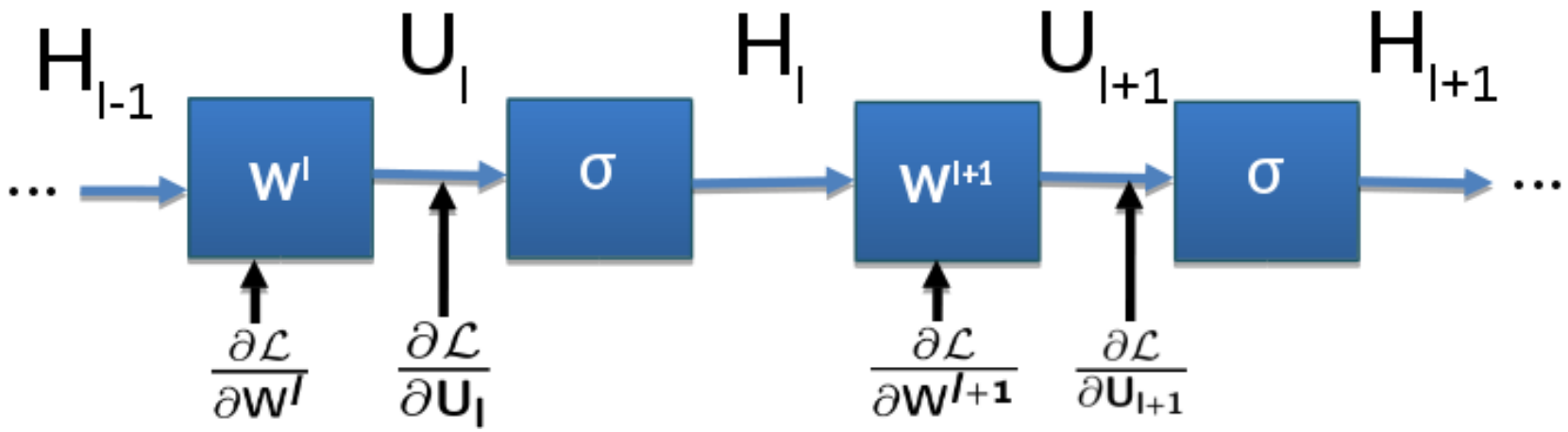
Perceptron Training: Backpropagation

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



Deep Neural Network Training: Backpropagation

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



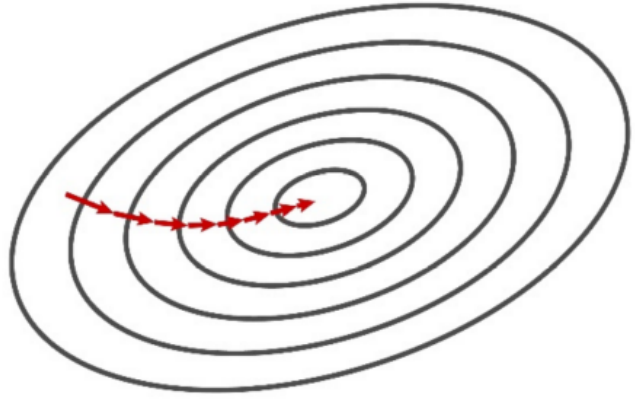
Neural Network Training: Optimization Issues

- Classification loss over training set (vectorized **w**, **b** ignored):

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

- Gradient descent optimization:

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



- Gradient $\nabla_{\mathbf{w}}^{(t)} = \frac{1}{N}\sum_{i=1}^{N}\frac{\partial\ell_{CE}(\hat{\mathbf{y}}_i,\mathbf{y}_i^*)}{\partial\mathbf{w}}\left(\mathbf{w}^{(t)}\right)$ linearly scales wrt:
  - **w** dimension
  - Training set size

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

Stochastic Gradient Descent

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

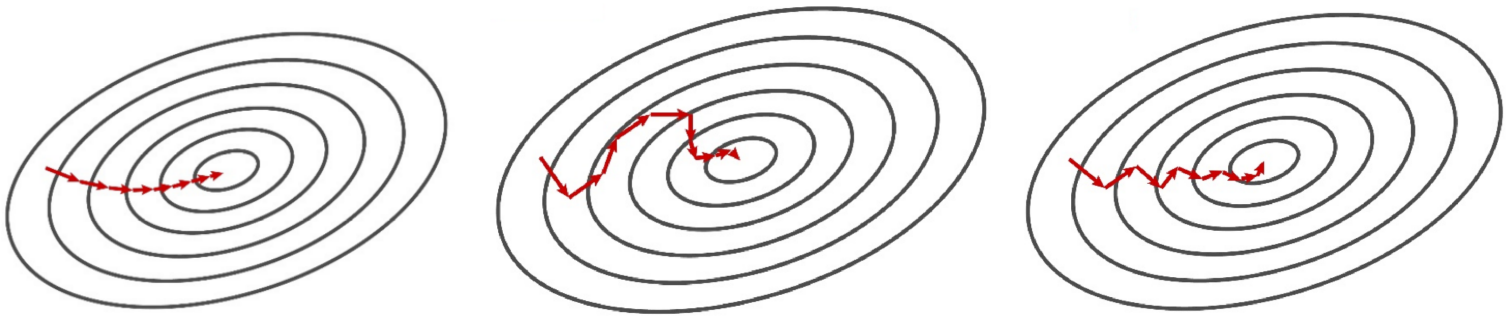
  $\Rightarrow$ **Stochastic Gradient Descent (SGD)**

  - Use a single example (online):

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

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

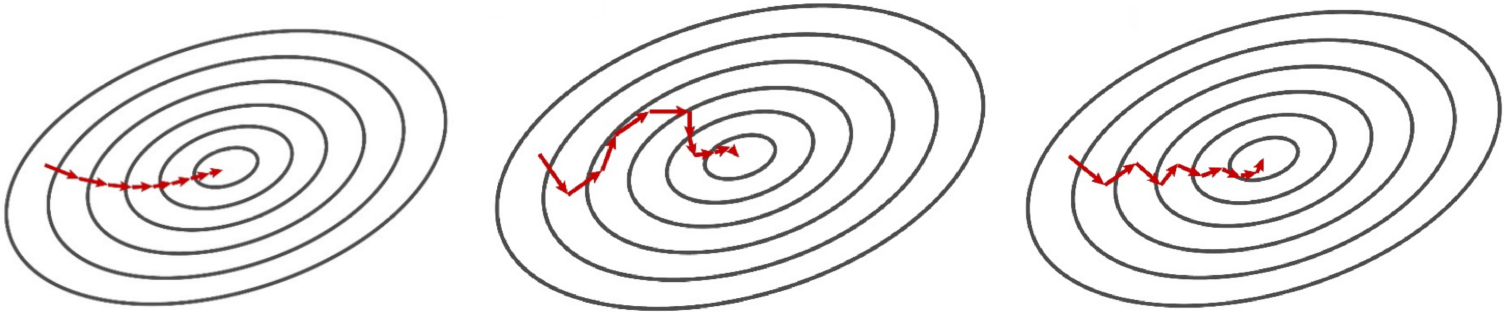
$$\nabla_{\mathbf{w}}^{(t)} \approx \frac{1}{B} \sum_{i=1}^{B} \frac{\partial \ell_{CE}(\hat{\mathbf{y}}_i, \mathbf{y}_i^*)}{\partial \mathbf{w}} \left(\mathbf{w}^{(t)}\right)$$



Full gradient | SGD (online) | SGD (mini-batch)

Stochastic Gradient Descent

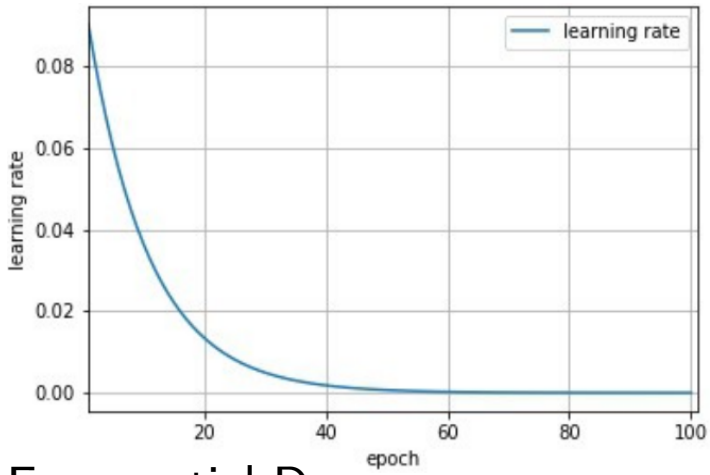
- **SGD: approximation of the true Gradient $\nabla_{\mathbf{w}}$!**
  - Noisy gradient can lead to bad direction, increase loss
  - **BUT:** much more parameter updates: online $\times N$, mini-batch $\times \frac{N}{B}$
  - **Faster convergence**, at the core of Deep Learning for large scale datasets



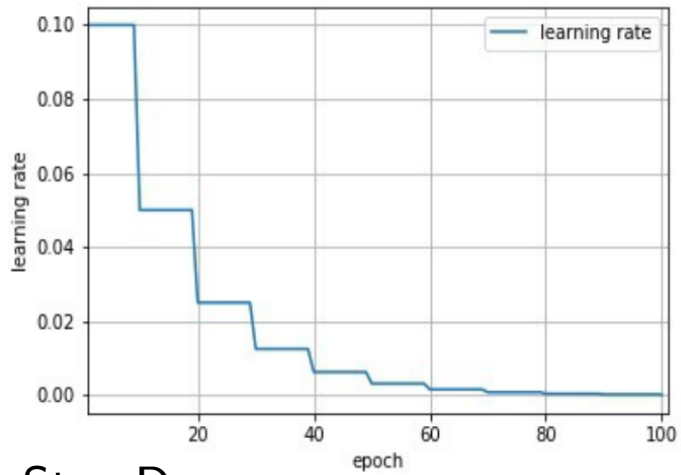
Full gradient | SGD (online) | SGD (mini-batch)

Optimization: Learning Rate Decay

- Gradient descent optimization: $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \nabla_{\mathbf{w}}^{(t)}$
- 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 $\eta_t = \eta_0 \cdot r^{\frac{t}{t_u}}$ ...



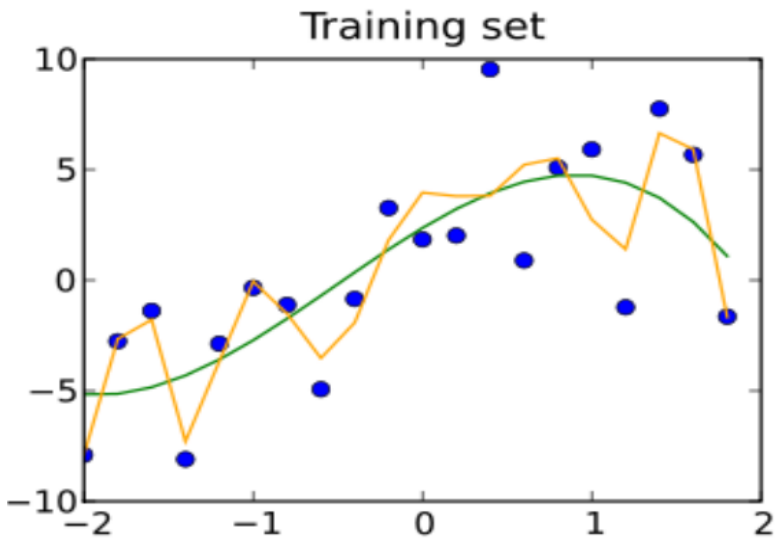
Exponential Decay ($\eta_0 = 0.1$, $\lambda = 0.1s$)



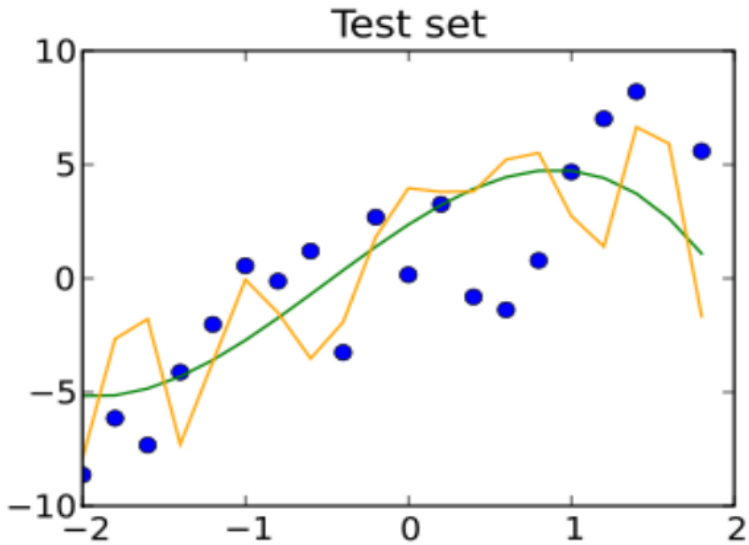
Step Decay ($\eta_0 = 0.1$, $r = 0.5$, $t_u = 10$)

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*** $R(\mathbf{w})$ in training objective:

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

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