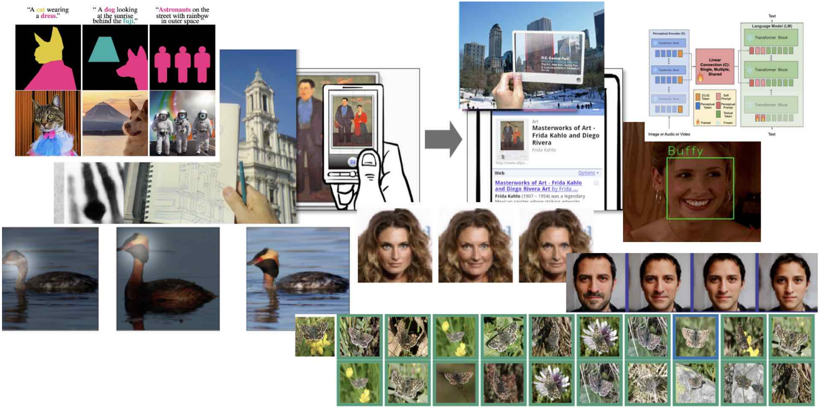


COURS RDFIA deep Image

https://cord.isir.upmc.fr/teaching-rdfia/

Matthieu Cord
Sorbonne University

Course Outline

1. Computer Vision and ML basics: Visual (local) feature detection and description, Bag of Word Image representation, Linear classification (SVM)
2. Introduction to Neural Networks (NNs)
3. Machine Learning basics (2): Risk, Classification, Datasets, benchmarks and evaluation
4. Neural Nets for Image Classification
5. Vision Transformers
6. Transfer learning and domain adaptation
7. Segmentation and Detection
8. Generative models with GANs
9. Generative models with diffusion
10. Large VL models: CLIP, StableDiffusion, Flamingo
11. Control (to be checked) -- Explainable AI, Fairness

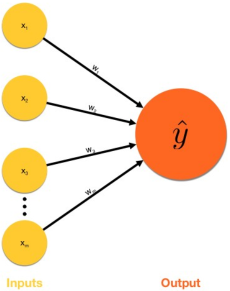
12/13 Bayesian deep learning

14 Robustness

Evaluations: Control (30%) + Practicals (3 reports; 70%)
can be modified by 10% between the 2 evaluations

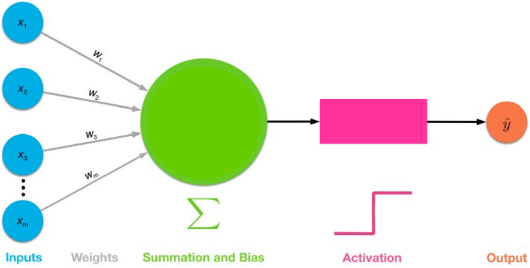
The Formal Neuron: 1943 [MP43]

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



The Formal Neuron: 1943 [MP43]

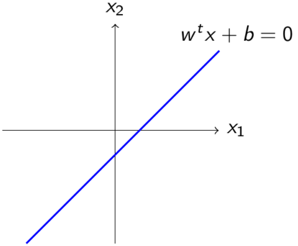
- Mapping from x to $\hat{y}$:
  1. Linear (affine) mapping: $s = \mathrm{w}^\top \mathrm{x} + b$
  2. Non-linear activation function: f: $\hat{y} = f(s)$



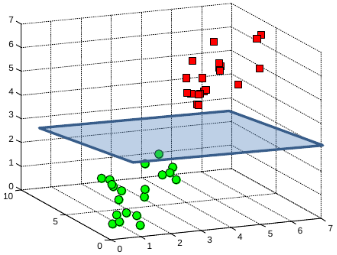
The Formal Neuron: Linear Mapping

- 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

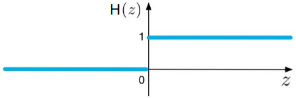


3D hyperplane: plane

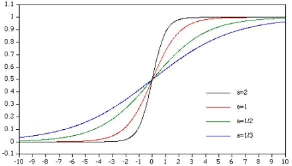


The Formal Neuron: Activation Function

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

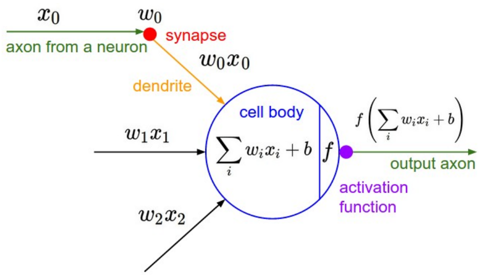


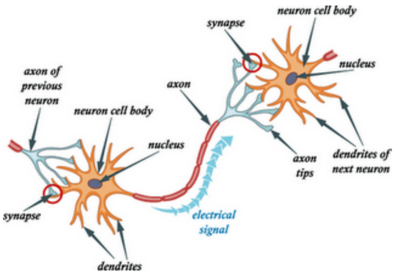
  - Popular f choices: sigmoid, tanh, ReLU, GELU, ...
  - Sigmoid: $\sigma(z) = (1 + e^{-az})^{-1}$



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

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

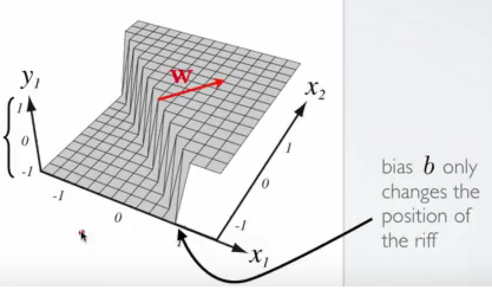
The Formal neuron: Application to Binary Classification

- Binary Classification: label input x as belonging to class 1 or 0
- Neuron output with sigmoid:

$$\hat{y} = \frac{1}{1 + e^{-a(\mathbf{w}^\top \mathbf{x} + b)}}$$

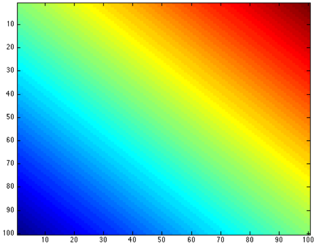
- Sigmoid: probabilistic interpretation $\Rightarrow \hat{y} \sim P(1|\mathbf{x})$
  - Input x classified as 1 if $P(1|\mathbf{x}) > 0.5 \Leftrightarrow \mathbf{w}^\top \mathbf{x} + b > 0$
  - Input x classified as 0 if $P(1|\mathbf{x}) < 0.5 \Leftrightarrow \mathbf{w}^\top \mathbf{x} + b < 0$

  $\Rightarrow \text{sign}(\mathbf{w}^\top \mathbf{x} + b)$: linear boundary decision in input space !



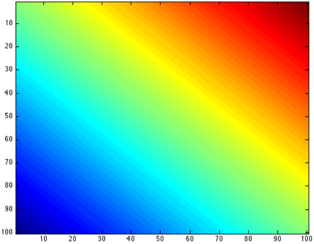
The Formal neuron: Toy Example for Binary Classification

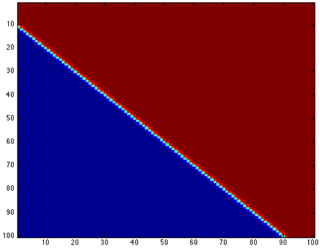
- 2d example: $m = 2$, $\mathrm{x} = \{x_1, x_2\} \in [-5; 5] \times [-5; 5]$
- Linear mapping: $\mathrm{w} = [1; 1]$ and $b = -2$
- Result of linear mapping : $s = \mathrm{w}^\top \mathrm{x} + b$

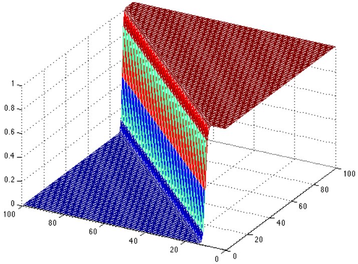


The Formal neuron: Toy Example for Binary Classification

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

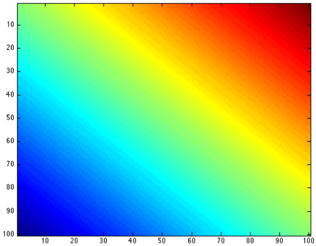


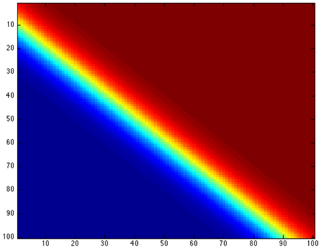


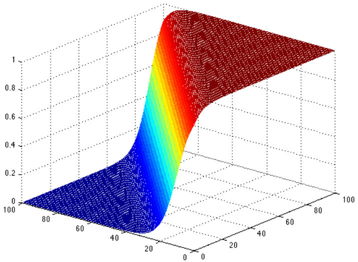


The Formal neuron: Toy Example for Binary Classification

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

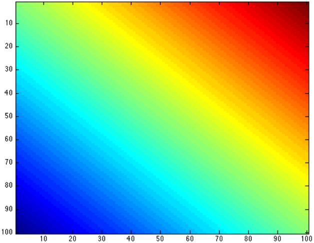


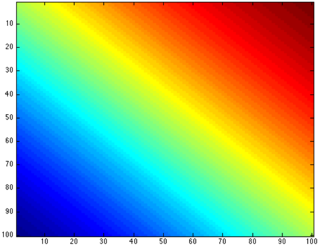


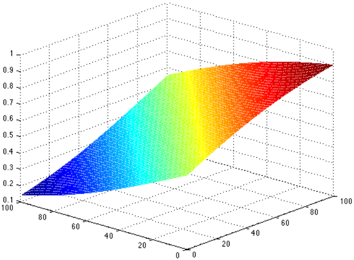


The Formal neuron: Toy Example for Binary Classification

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

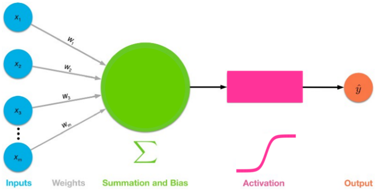


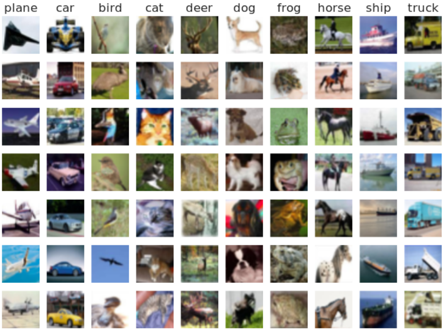




From Formal Neuron to Neural Networks

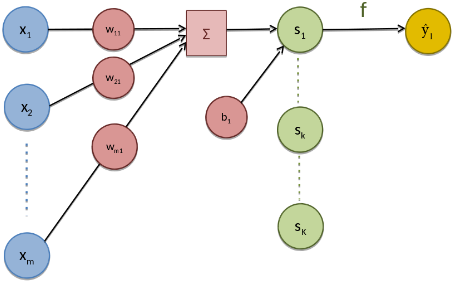
- Formal Neuron:
  1. A single scalar output
  2. Linear decision boundary for binary classification
- Single scalar output: limited for several tasks
  - Ex: multi-class classification, *e.g.* MNIST or CIFAR





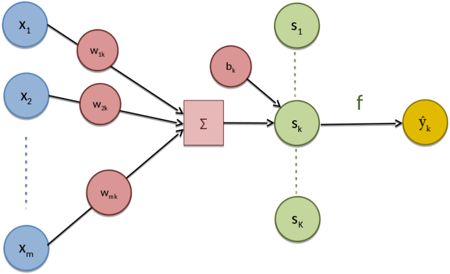
Perceptron and Multi-Class Classification

- 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:
  - $\mathbf{w_1} = \{w_{11}, \ldots, w_{m1}\} \in \mathbb{R}^m$
  - $b_1 \in \mathbb{R}$



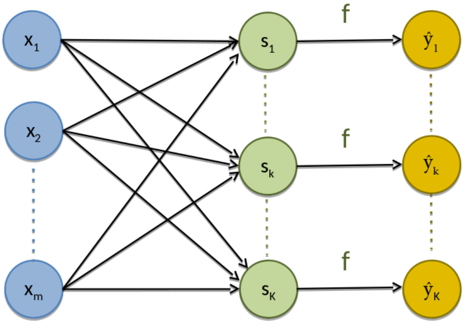
Perceptron and Multi-Class Classification

- Input 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: $\mathrm{s} = \mathrm{xW} + \mathrm{b}$
  - W matrix of size $m \times K$ - columns are $\mathrm{w_k}$
  - b: bias vector - size $1 \times K$
- Element-wise non-linear activation: $\hat{\mathrm{y}} = f(\mathrm{s})$

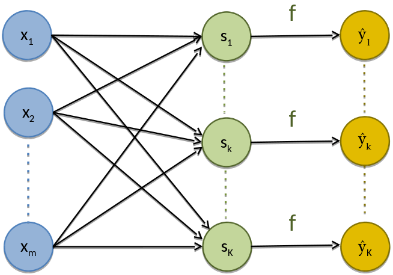


Perceptron and Multi-Class Classification

- **Soft-max Activation**:

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

- Note that $f(s_k)$ depends on the other $s_{k'}$, the arrow is a functional link
- **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!**

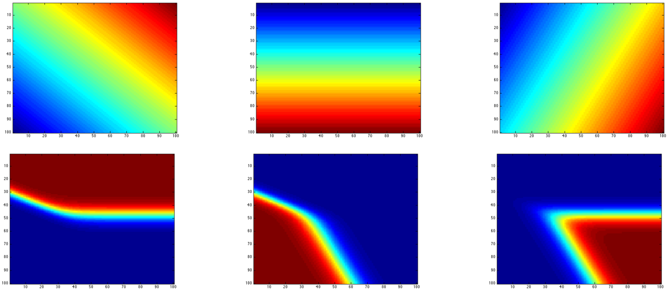
2d Toy Example for Multi-Class Classification

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

$\mathbf{w_1} = [\mathbf{1}; \mathbf{1}]$, $b_\mathbf{1} = -\mathbf{2}$ $\quad$ $\mathbf{w_2} = [\mathbf{0}; -\mathbf{1}]$, $b_\mathbf{2} = \mathbf{1}$ $\quad$ $\mathbf{w_3} = [\mathbf{1}; -\mathbf{0.5}]$, $b_\mathbf{3} = \mathbf{10}$

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

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

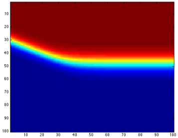


2d Toy Example for Multi-Class Classification

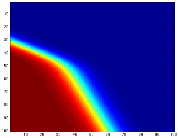
- $\mathrm{x} = \{x_1, x_2\} \in [-5; 5] \times [-5; 5]$, $\hat{y}$: 3 outputs (classes)

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

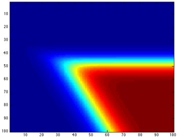
$\mathbf{w_1} = [\mathbf{1}; \mathbf{1}],\ b_{\mathbf{1}} = -\mathbf{2}$



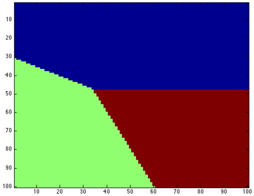
$\mathbf{w_2} = [\mathbf{0}; -\mathbf{1}],\ b_{\mathbf{2}} = \mathbf{1}$



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



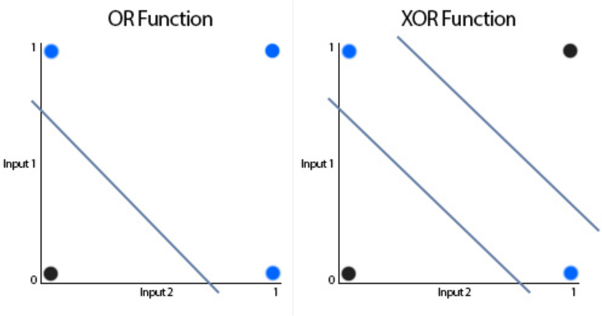
Class Prediction:
$k^* = \arg\max_k P(k|\mathrm{x}, \mathrm{W})$



Beyond Linear Classification

X-OR Problem

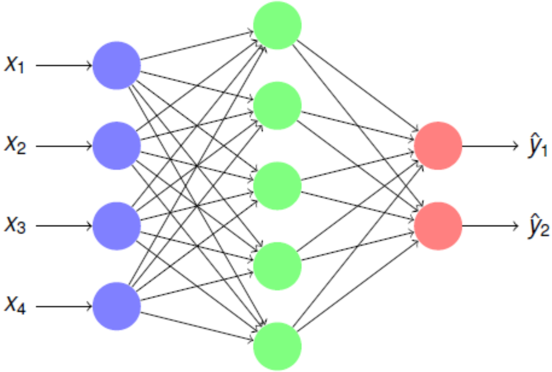
- Logistic Regression (LR): NN with 1 input layer & 1 output layer
- LR: limited to linear decision boundaries
- **X-OR:** NOT 1 and 2 OR NOT 2 AND 1
  - **X-OR: Non linear decision function**



Beyond Linear Classification

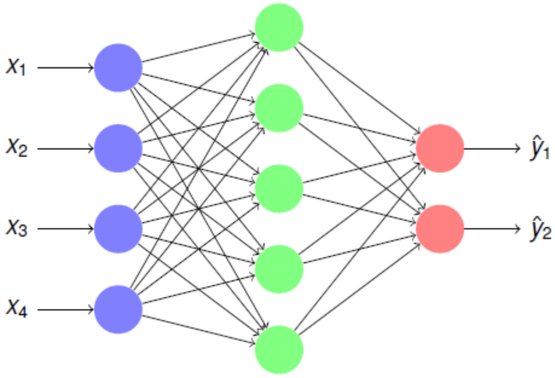
- LR: limited to linear boundaries
- **Solution:** add a layer!

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



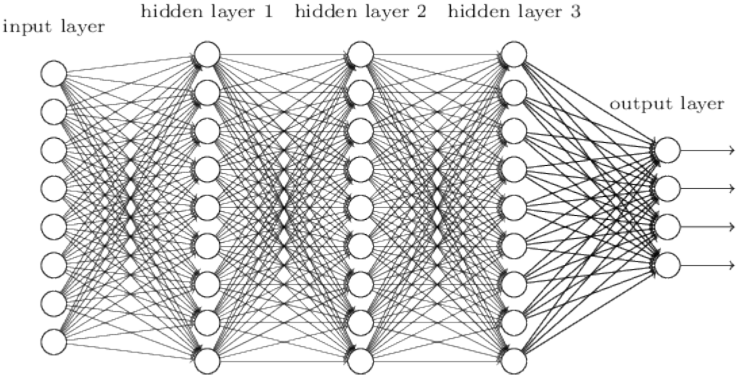
Multi-Layer Perceptron

- **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 $\hat{y}$:
- $h = f(xW_1 + b_1)$
  f non-linear activation,
  $s = hW_2 + b_2$
  $\hat{y} = SoftMax(s)$
- **Mapping from** x **to** $\hat{y}$**: non-linear boundary!**
  $\Rightarrow$ **non-linear activation** f **crucial!**



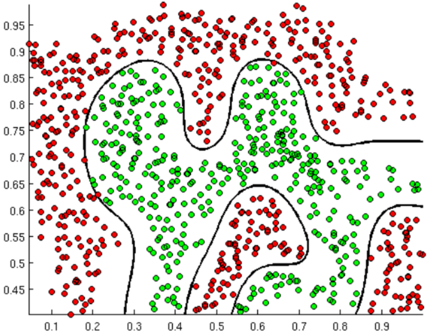
Deep Neural Networks

- Adding more hidden layers: Deep Neural Networks (DNN) $\Rightarrow$ **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**

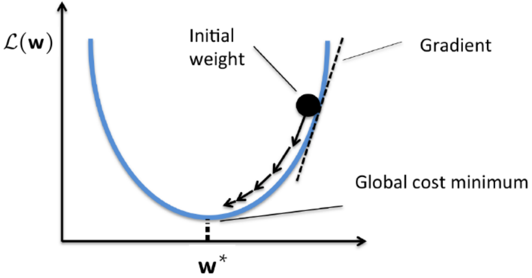
Outline

Training Multi-Layer Perceptron (MLP)

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

MLP Training

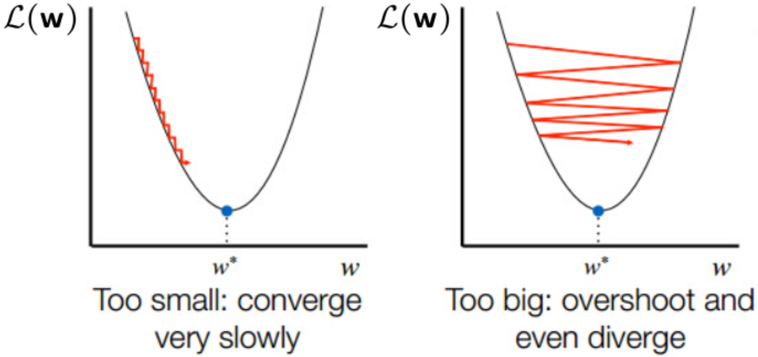
- Gradient descent algorithm:
  - Initialyze parameters w
  - Update: $\boxed{w^{(t+1)} = w^{(t)} - \eta \frac{\partial \mathcal{L}}{\partial w}}$
  - Until convergence, *e.g.* $||\nabla_w||^2 \approx 0$



Gradient Descent

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

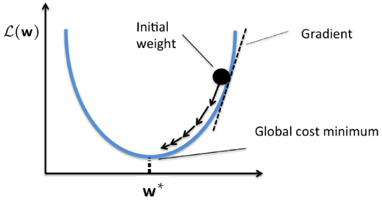
- **Convergence ensured ?** $\Rightarrow$ provided a "well chosen" learning rate η



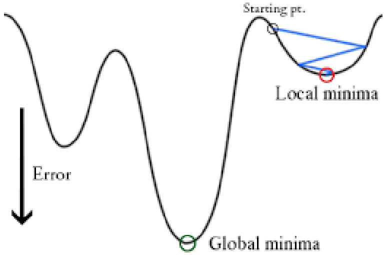
Gradient Descent

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

- **<u>Global minimum ?</u>**
  $\Rightarrow$ **convex** a) *vs* **non convex** b) loss $\mathcal{L}(\mathrm{w})$



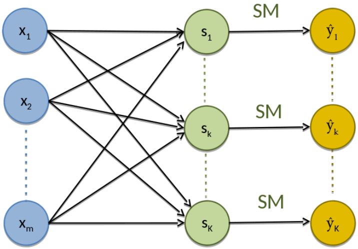
a) Convex function



a) Non convex function

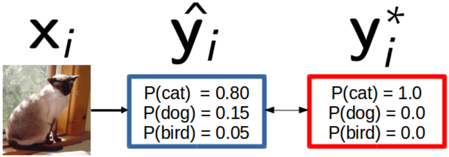
Supervised Learning: Multi-Class Classification

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



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

Logistic Regression Training Formulation



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

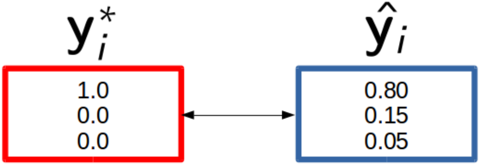
$$y_{c,i}^* = \begin{cases} 1 & \text{if c is the groud truth class for } x_i \\ 0 & \text{otherwise} \end{cases}$$

Logistic Regression Training Formulation

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

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

- ⚠ KL asymmetric: $KL(\hat{\mathrm{y}}_\mathrm{i}, \mathrm{y}_\mathrm{i}^*) \neq KL(\mathrm{y}_\mathrm{i}^*, \hat{\mathrm{y}}_\mathrm{i})$ ⚠



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

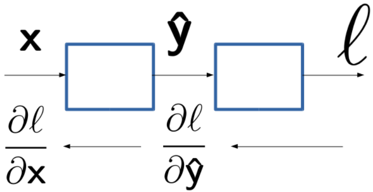
Logistic Regression Training

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

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

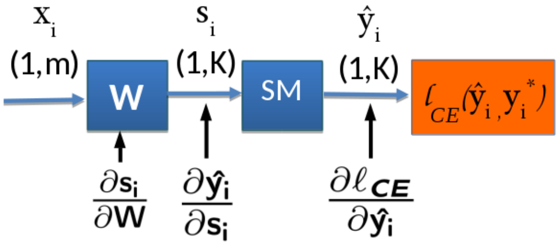
  $\Rightarrow$ **Backpropagation of gradient error!**

Chain Rule



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

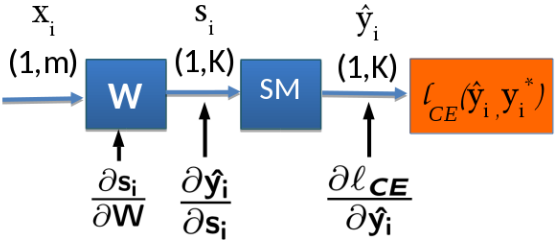
- Logistic regression: $\boxed{\frac{\partial \ell_{CE}}{\partial W} = \frac{\partial \ell_{CE}}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial s_i} \frac{\partial s_i}{\partial W}}$



Logistic Regression Training: Backpropagation

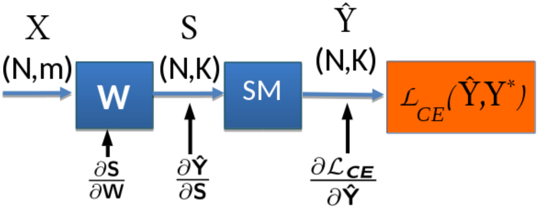
$$\frac{\partial \ell_{CE}}{\partial \mathbf{W}} = \frac{\partial \ell_{CE}}{\partial \hat{\mathbf{y}}_\mathbf{i}} \frac{\partial \hat{\mathbf{y}}_\mathbf{i}}{\partial \mathbf{s}_\mathbf{i}} \frac{\partial \mathbf{s}_\mathbf{i}}{\partial \mathbf{W}}, \ \ell_{CE}(\hat{\mathbf{y}}_i, \mathbf{y}_i^*) =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

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



Logistic Regression Training: Backpropagation

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

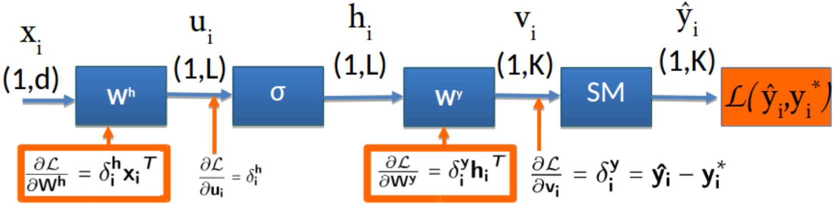


- $\boxed{\frac{\partial \mathcal{L}_{CE}}{\partial \mathbf{s}} = \hat{\mathrm{Y}} - \mathrm{Y}^* = \Delta^y}$
- $\boxed{\frac{\partial \mathcal{L}_{CE}}{\partial \mathrm{W}} = \mathrm{X}^T \Delta^y}$

Perceptron Training: Backpropagation

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

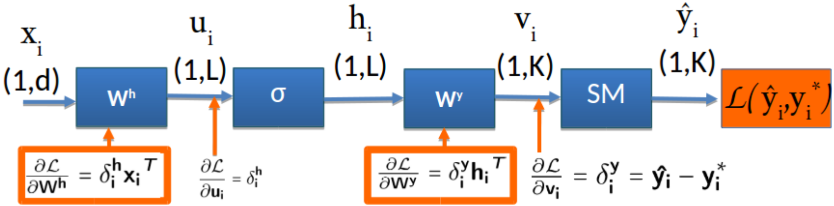
$\Rightarrow$ computing $\boxed{\frac{\partial \mathcal{L}_{CE}}{\partial W^y} = \frac{1}{N}\sum_{i=1}^{N} \frac{\partial \ell_{CE}}{\partial W^y}}$ and $\boxed{\frac{\partial \mathcal{L}_{CE}}{\partial W^h} = \frac{1}{N}\sum_{i=1}^{N} \frac{\partial \ell_{CE}}{\partial W^h}}$



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

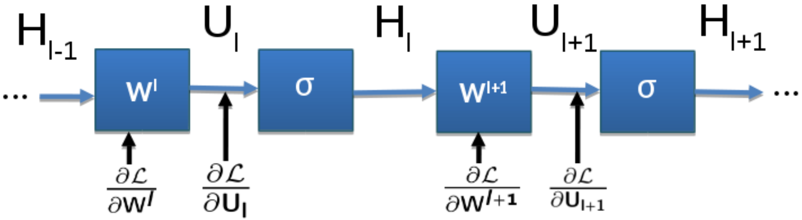
Perceptron Training: Backpropagation

- Computing $\frac{\partial \ell_{CE}}{\partial \mathbf{u_i}} = \delta_i^h \Rightarrow$ 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: $\frac{\partial \ell_{CE}}{\partial \mathbf{u_i}} = \delta_i^h = \delta_i^{y\,T} W^y \odot \sigma'(h_i) = \delta_i^{y\,T} W^y \odot (h_i \odot (1 - h_i))$



Deep Neural Network Training: Backpropagation

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



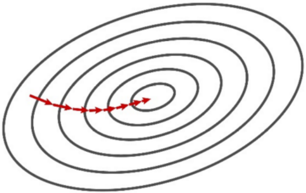
Neural Network Training: Optimization Issues

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

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

- Gradient descent optimization:

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



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

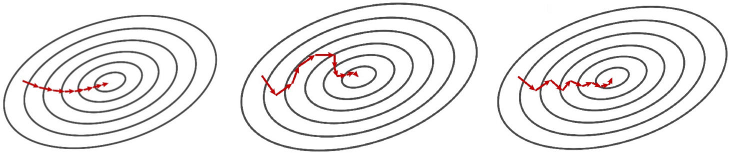
  $\Rightarrow$ **Stochastic Gradient Descent (SGD)**

  - 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:

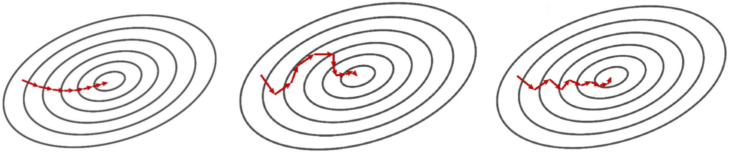
$$\nabla_{\mathrm{w}}^{(t)} \approx \frac{1}{B} \sum_{i=1}^{B} \frac{\partial \ell_{CE}(\hat{\mathrm{y}}_i, \mathrm{y}_i^*)}{\partial \mathrm{w}} \left(\mathrm{w}^{(t)}\right)$$



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

Stochastic Gradient Descent

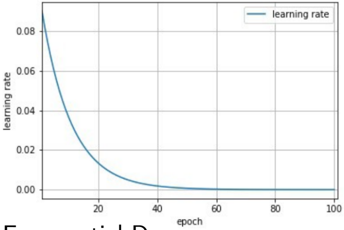
- **SGD: approximation of the true Gradient ∇_w !**
  - Noisy gradient can lead to bad direction, increase loss
  - **BUT:** much more parameter updates: online $\times N$, mini-batch $\times \frac{N}{B}$
  - **Faster convergence**, at the core of Deep Learning for large scale datasets



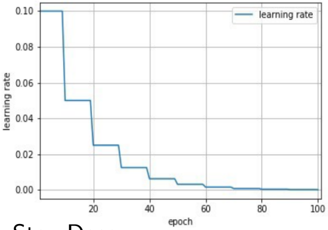
Full gradient　　SGD (online)　　SGD (mini-batch)

Optimization: Learning Rate Decay

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



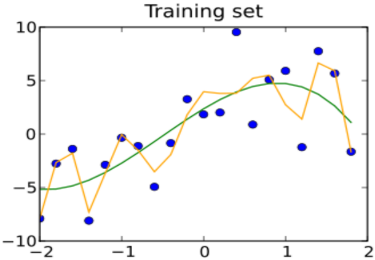
Exponential Decay $(\eta_0 = 0.1, \lambda = 0.1s)$

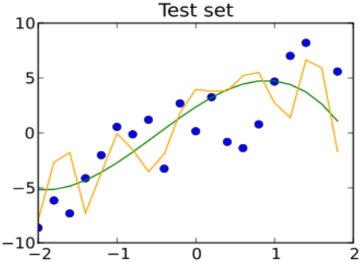


Step Decay $(\eta_0 = 0.1, r = 0.5, t_u = 10)$

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**





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

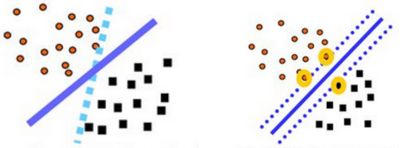
$\Rightarrow$ **Optimization $\neq$ Machine Learning!**
$\Rightarrow$ **Generalization / Overfitting**!

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*



L^2 **regularization**: interpretation

- "Smooth" interpretation of L^2 **regularization**, Cauchy-Schwarz:
$$\langle \mathrm{w}, (\mathrm{x} - \mathrm{x}') \rangle \leq ||\mathrm{w}||^2 ||\mathrm{x} - \mathrm{x}'||^2$$
- Controlling L^2 norm $||\mathrm{w}||^2$: "small" variation between inputs x and x'
$\Rightarrow$ small variation in neuron prediction $\langle \mathrm{w}, \mathrm{x} \rangle$ and $\langle \mathrm{w}, \mathrm{x}' \rangle$

$\Rightarrow$ **Supports simple, *i.e.* smoothly varying prediction models**

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:** $\Rightarrow$ improve generalization: estimate performances on a validation set