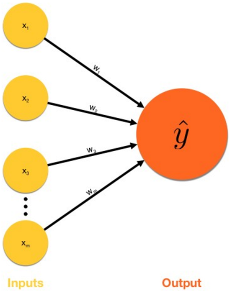
Outline

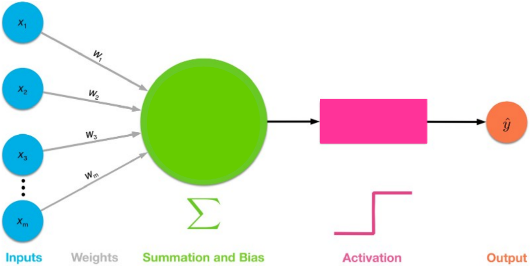
The Formal Neuron: 1943 [?]

- 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 [?]

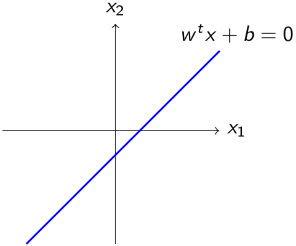
- Mapping from x to $\hat{y}$:
  1. Linear (affine) mapping: $s = \mathrm{w}^\top \mathrm{x} + b$
  2. Non-linear activation function: f: $\hat{y} = f(s)$



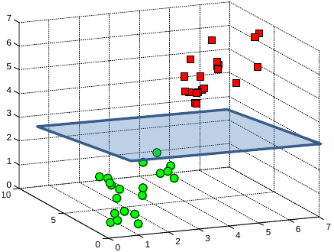
The Formal Neuron: Linear Mapping

- 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

<u>2D hyperplane:</u> line

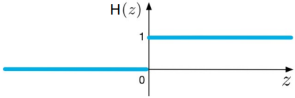


<u>3D hyperplane:</u> plane

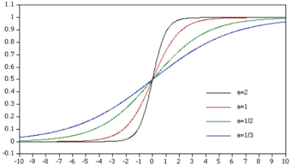


The Formal Neuron: Activation Function

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

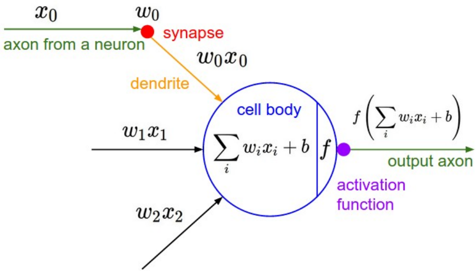


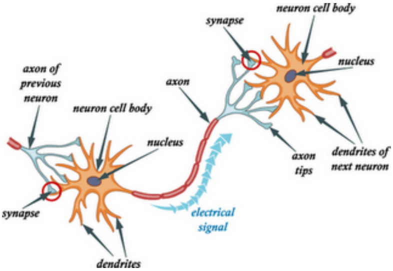
  - Popular f choices: sigmoid, tanh, ReLU, GELU, ...
  - Sigmoid: $\sigma(z) = (1 + e^{-az})^{-1}$



    - $a \uparrow$: 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

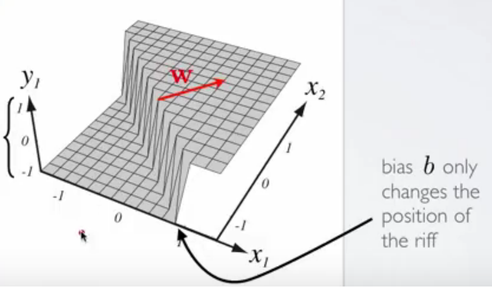
The Formal neuron: Application to Binary Classification

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

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

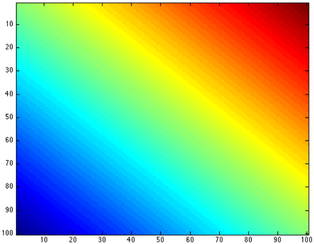
- Sigmoid: probabilistic interpretation $\Rightarrow \hat{y} \sim P(1|\mathbf{x})$
  - Input x classified as 1 if $P(1|\mathbf{x}) > 0.5 \Leftrightarrow \mathbf{w}^\top \mathbf{x} + b > 0$
  - Input x classified as 0 if $P(1|\mathbf{x}) < 0.5 \Leftrightarrow \mathbf{w}^\top \mathbf{x} + b < 0$

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



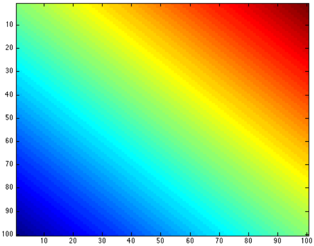
The Formal neuron: Toy Example for Binary Classification

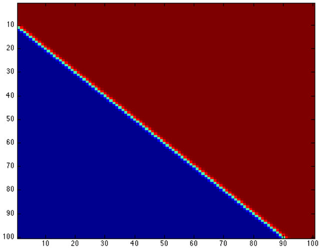
- 2d example: $m = 2$, $\mathrm{x} = \{x_1, x_2\} \in [-5; 5] \times [-5; 5]$
- Linear mapping: $\mathrm{w} = [1; 1]$ and $b = -2$
- Result of linear mapping : $s = \mathrm{w}^\top \mathrm{x} + b$

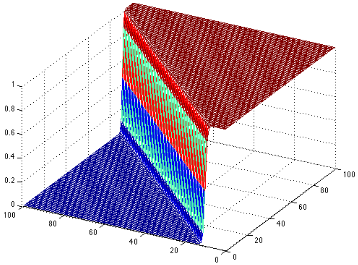


The Formal neuron: Toy Example for Binary Classification

- 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: $\hat{y} = \left(1 + e^{-a(\mathrm{w}^\top \mathrm{x} + b)}\right)^{-1}$, $a = 10$

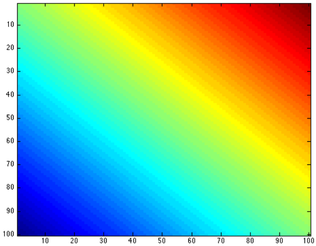


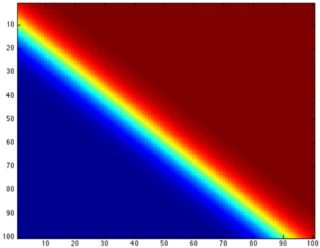


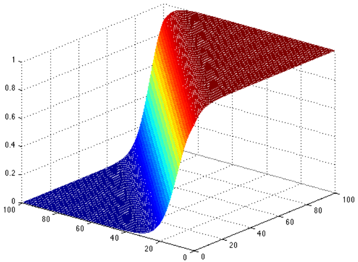


The Formal neuron: Toy Example for Binary Classification

- 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: $\hat{y} = \left(1 + e^{-a(\mathrm{w}^\top \mathrm{x} + b)}\right)^{-1}$, $a = 1$

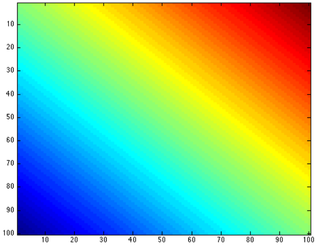


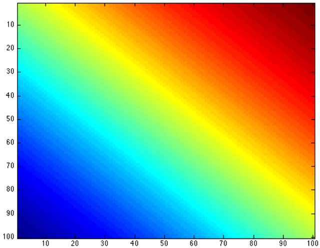


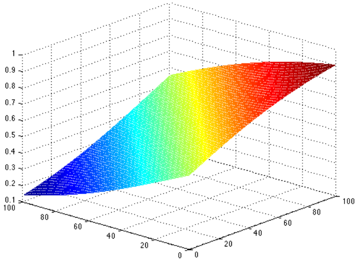


The Formal neuron: Toy Example for Binary Classification

- 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: $\hat{y} = \left(1 + e^{-a(\mathrm{w}^\top \mathrm{x} + b)}\right)^{-1}$, $a = 0.1$

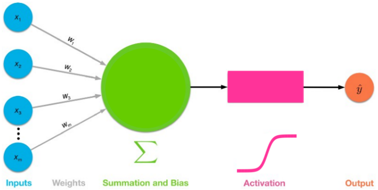


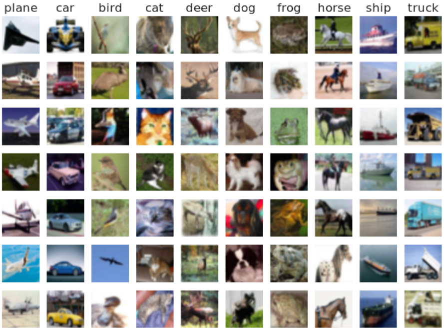




From Formal Neuron to Neural Networks

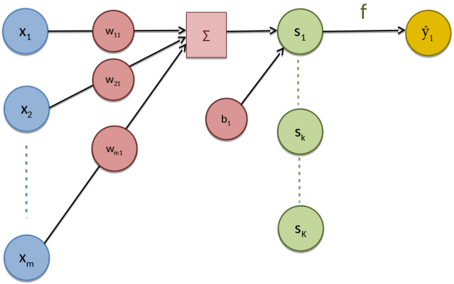
- Formal Neuron:
  1. A single scalar output
  2. Linear decision boundary for binary classification
- Single scalar output: limited for several tasks
  - Ex: multi-class classification, *e.g.* MNIST or CIFAR



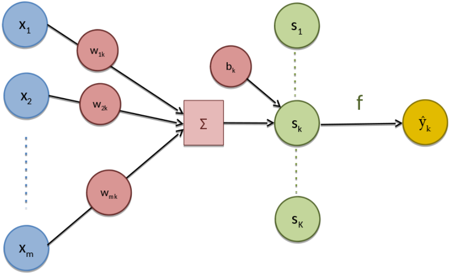


Perceptron and Multi-Class Classification

- Formal Neuron: limited to binary classification
- **Multi-Class Classification:** use several output neurons instead of a single one! $\Rightarrow$ **Perceptron**
- Input 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:
  - $\mathbf{w_1} = \{w_{11}, ..., w_{m1}\} \in \mathbb{R}^m$
  - $b_1 \in \mathbb{R}$



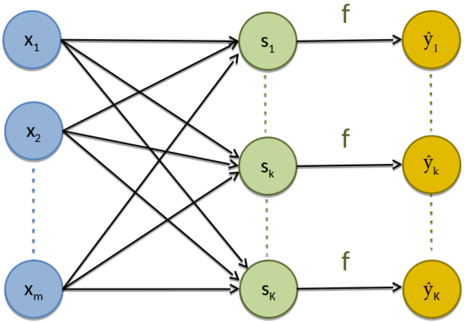
Perceptron and Multi-Class Classification



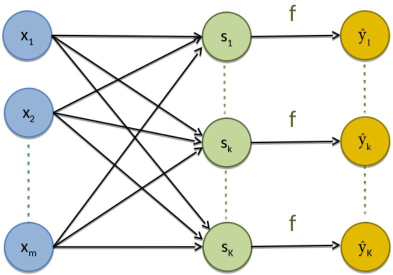
- Input x in $\mathbb{R}^m$
- Output neuron $\hat{y}_k$ is a formal neuron:
  - Linear (affine) mapping: $s_k = \mathrm{w_k}^\top \mathrm{x} + b_k$
  - Non-linear activation function: f: $\hat{y}_k = f(s_k)$
- Linear mapping parameters:
  - $\mathrm{w_k} = \{w_{1k}, ..., w_{mk}\} \in \mathbb{R}^m$
 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

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 ~ 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
  - $\hat{y}_k \sim P(k|\mathrm{x}, \mathrm{W})$

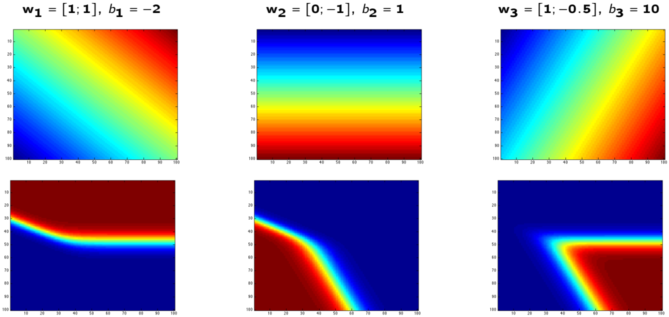
$\Rightarrow$ **Logistic Regression (LR) Model!**

2d Toy Example for Multi-Class Classification

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

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

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

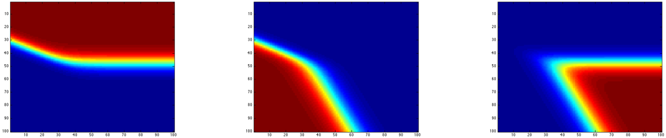


2d Toy Example for Multi-Class Classification

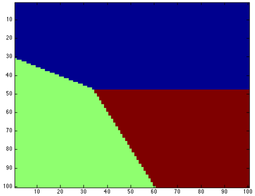
- $\mathrm{x} = \{x_1, x_2\} \in [-5; 5] \times [-5; 5]$, $\hat{y}$: 3 outputs (classes)

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

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



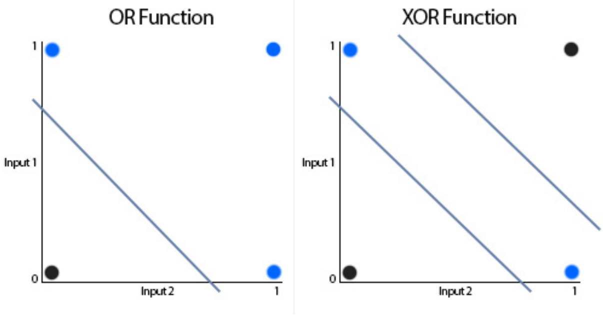
Class Prediction:
$k^* = \arg\max_k P(k|\mathrm{x}, \mathrm{W})$



Beyond Linear Classification

X-OR Problem

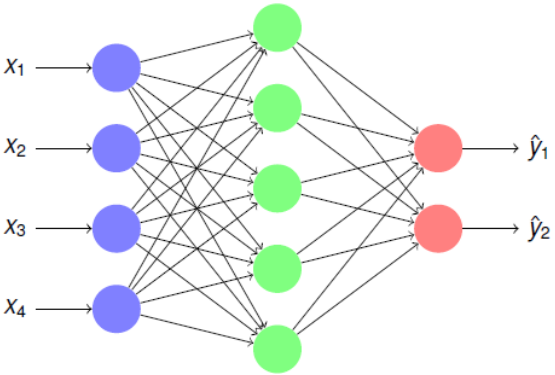
- Logistic Regression (LR): NN with 1 input layer & 1 output layer
- LR: limited to linear decision boundaries
- **X-OR:** NOT 1 and 2 OR NOT 2 AND 1
  - **X-OR: Non linear decision function**



Beyond Linear Classification

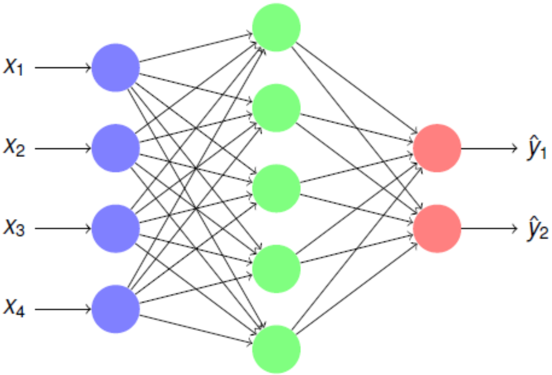
- LR: limited to linear boundaries
- **Solution:** add a layer!

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



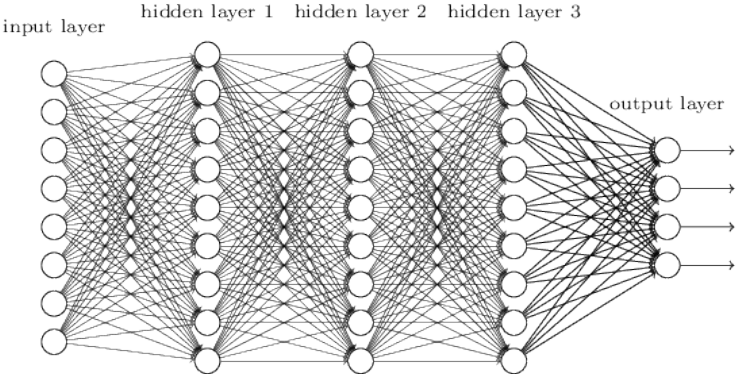
Multi-Layer Perceptron

- **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 $\hat{y}$:
- $h = f(xW_1 + b_1)$
  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!



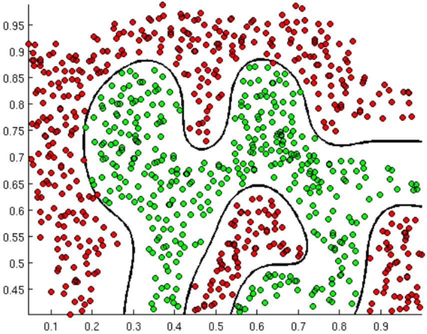
Deep Neural Networks

- 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**

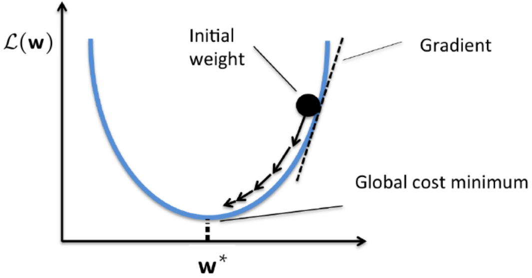
Outline

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 $\mathcal{L}(w)$

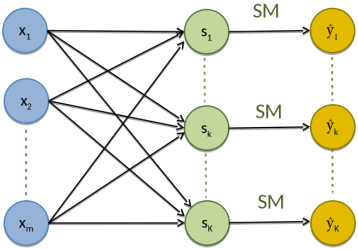
MLP Training

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



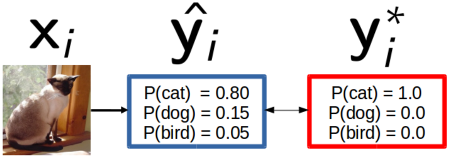
Supervised Learning: Multi-Class Classification

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



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

Logistic Regression Training Formulation



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

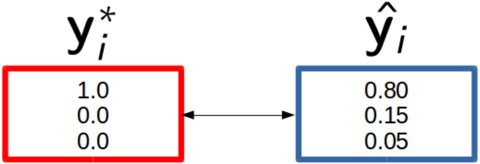
$$y^*_{c,i} = \begin{cases} 1 & \text{if c is the groud truth class for } x_i \\ 0 & \text{otherwise} \end{cases}$$

Logistic Regression Training Formulation

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



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

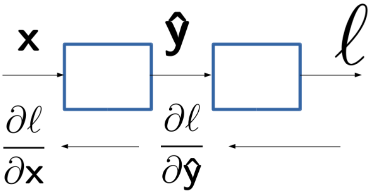
Logistic Regression Training

- $\mathcal{L}_{CE}(\mathrm{W}, \mathrm{b}) = \frac{1}{N} \sum_{i=1}^{N} \ell_{CE}(\hat{\mathrm{y}}_i, \mathrm{y}_i^*) = -\frac{1}{N} \sum_{i=1}^{N} log(\hat{y}_{c^*,i})$
- 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: $\boxed{\frac{\partial \ell_{CE}}{\partial W} = \frac{\partial \ell_{CE}}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial s_i} \frac{\partial s_i}{\partial W}}$