Multiclass Perceptron

The Perceptron learner may addess multiclass scenarios by:

Multiclass Perceptron

The Perceptron learner may addess multiclass scenarios by:

1. **One-VS-Rest** scheme:
   - $\rightarrow$ Requires $|\mathcal{W}|$ Perceptrons
   - $\rightarrow$ Perceptron with the largest activation determines $\hat{\omega}$

Multiclass Perceptron

The Perceptron learner may addess multiclass scenarios by:

1. **One-VS-Rest** scheme:
   - → Requires $|\mathcal{W}|$ Perceptrons
   - → Perceptron with the largest activation determines $\hat{\omega}$
2. **One-VS-One** scheme:
   - → Requires $^{|\mathcal{W}|\cdot(|\mathcal{W}|-1)}/_{2}$ Perceptrons
   - → Class $\hat{\omega}$ is estimated by voting

Multiclass Perceptron

The Perceptron learner may addess multiclass scenarios by:

1. **One-VS-Rest** scheme:
   - $\rightarrow$ Requires $|\mathcal{W}|$ Perceptrons
   - $\rightarrow$ Perceptron with the largest activation determines $\hat{\omega}$
2. **One-VS-One** scheme:
   - $\rightarrow$ Requires $^{|\mathcal{W}|\cdot(|\mathcal{W}|-1)}/_{2}$ Perceptrons
   - $\rightarrow$ Class $\hat{\omega}$ is estimated by voting
3. **Multiple weight** vectors:
   - $\rightarrow$ Requires only one Perceptron
   - $\rightarrow$ Weight matrix with a weight vector per class: $\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots, \boldsymbol{\theta}_{|\mathcal{W}|}$
   - $\rightarrow$ Only the weight vector associated to the class is updated while training
   - $\rightarrow$ Weight vector with the largest activation determines $\hat{\omega}$

Outline

1. Linear models
   - Binary
   - Multiclass
   - Parameter estimation
2. Perceptron
   - Introduction
   - Training
   - Limitations
   - Multiclass Perceptron
3. Multi-layer Perceptron
   - Introduction
   - Structure
   - Training
   - Training dynamics and regularization

Introduction to MLP

- Natural evolution of the Perceptron learner

Introduction to MLP

- Natural evolution of the Perceptron learner
- Stack of Perceptron units $\Rightarrow$ approximating non-linear functions

Introduction to MLP

- Natural evolution of the Perceptron learner
- Stack of Perceptron units $\Rightarrow$ approximating non-linear functions
- Constitutes a particular case of Feedforward Neural Networks
  - $\rightarrow$ Information flows in one direction $\Rightarrow$ input to output

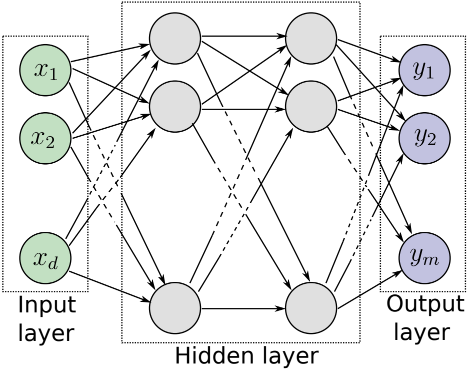
Introduction to MLP

- Natural evolution of the Perceptron learner
- Stack of Perceptron units $\Rightarrow$ approximating non-linear functions
- Constitutes a particular case of Feedforward Neural Networks
  - $\rightarrow$ Information flows in one direction $\Rightarrow$ input to output

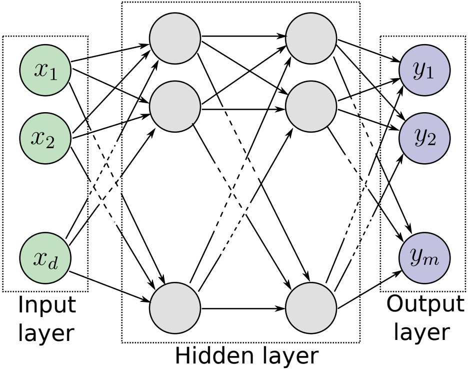
- **Universal Approximation Theorem**:

  A feedforward neural network with at least one hidden layer, using a nonlinear activation, can approximate any continuous function to any desired degree of accuracy, given sufficient neurons in the hidden layer

Structure



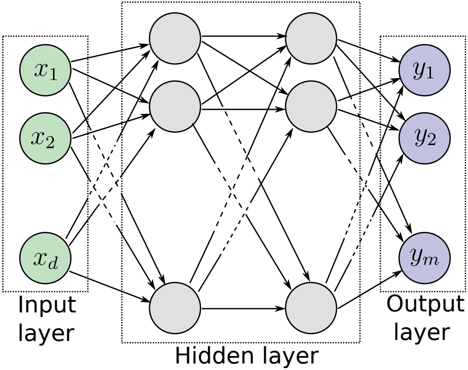
Structure



Input

- Representation space $\mathbb{R}^d$
  $\Rightarrow$ d neurons

Structure



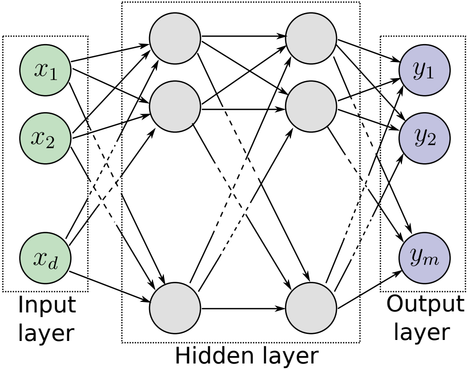
Input

- Representation space $\mathbb{R}^d$
  $\Rightarrow$ *d* neurons

Hidden

- Non-linear activations
- Fully connected
- Configuration depends on the complexity of the function

Structure



Input	Hidden	Output
- Representation space $\mathbb{R}^d$ $\Rightarrow$ d neurons	- Non-linear activations - Fully connected - Configuration depends on the complexity of the function	- Configuration depends on the task

Input layer

- Entrance point to the model for datum $\mathbf{x} \in \mathbb{R}$
- As many neurons as the dimensionality of $\mathbf{x} \Rightarrow d$ neurons
- There may be additional neurons $\Rightarrow$ additional biases

Hidden layer

- Sequence of (vertical) stacks of Perceptron units:
  - → I stacks
  - → J_i neurons at the i-th stack ($1 \leq i \leq I$)

Hidden layer

- Sequence of (vertical) stacks of Perceptron units:
  - $\rightarrow$ I stacks
  - $\rightarrow$ J_i neurons at the i-th stack $(1 \leq i \leq I)$

- Each stack may depict a different number of Perceptron units

Hidden layer

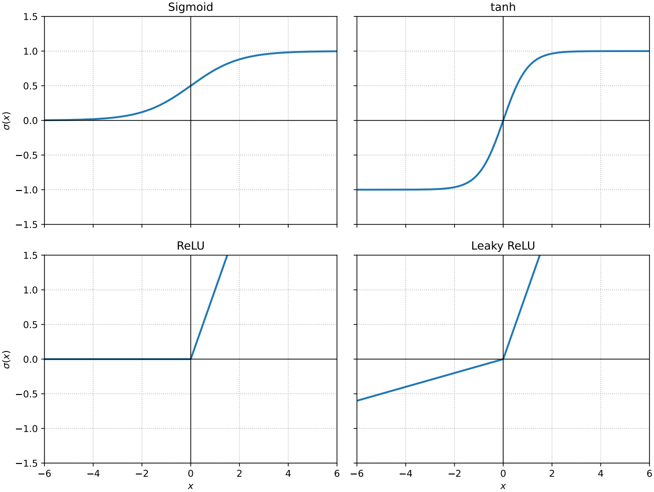
- Sequence of (vertical) stacks of Perceptron units:
  - → I stacks
  - → J_i neurons at the i-th stack $(1 \leq i \leq I)$

- Each stack may depict a different number of Perceptron units

- In general, non-linear activation functions:

$$y_{h_{i,j}} = \sigma\left(\mathbf{h}_{i,j}^{T} \cdot \mathbf{y}_{h_{i-1}}\right)$$

Activation functions



Activation functions

- Sigmoid:

$$\sigma(x) = \frac{e^x}{1 + e^x}$$

- Hyperbolic tangent (tanh):

$$\sigma(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

- Rectified Linear Unit (ReLU):

$$\sigma(x) = \max\{0, x\}$$

- Leaky Rectified Linear Unit (Leaky ReLU):

$$\sigma(x) = \begin{cases} x \text{ if } x \geq 0 \\ \alpha \cdot x \text{ if } x < 0 \end{cases} \qquad \text{with } \alpha \in [0.1, 0.3]$$

Output layer

- Each output is a real-valued function: $y_j(\mathbf{x};\boldsymbol{\theta}) \in \mathbb{R}$ with $i \leq j \leq m$

Output layer

- Each output is a real-valued function: $y_j(\mathbf{x}; \boldsymbol{\theta}) \in \mathbb{R}$ with $i \leq j \leq m$

- Regression tasks:
  - $\rightarrow$ One regressor per output unit

Output layer

- Each output is a real-valued function: $y_j(\mathbf{x};\boldsymbol{\theta}) \in \mathbb{R}$ with $i \leq j \leq m$
- Regression tasks:
  $\rightarrow$ One regressor per output unit
- Classification tasks ($|\mathcal{W}|$ classes):
  $\rightarrow$ One unit per class $\Rightarrow m = |\mathcal{C}|$

Output layer

- Each output is a real-valued function: $y_j(\mathbf{x}; \boldsymbol{\theta}) \in \mathbb{R}$ with $i \leq j \leq m$

- Regression tasks:
  - $\rightarrow$ One regressor per output unit

- Classification tasks ($|\mathcal{W}|$ classes):
  - $\rightarrow$ One unit per class $\Rightarrow m = |\mathcal{C}|$
  - $\rightarrow$ Highest score is the estimated class $\Rightarrow \hat{\omega} = \arg\max_{j \in \{1,\dots,m\}} y_j$

Output layer

- Each output is a real-valued function: $y_j(\mathbf{x};\boldsymbol{\theta}) \in \mathbb{R}$ with $i \leq j \leq m$

- Regression tasks:
  - $\rightarrow$ One regressor per output unit

- Classification tasks ($|\mathcal{W}|$ classes):
  - $\rightarrow$ One unit per class $\Rightarrow m = |\mathcal{C}|$
  - $\rightarrow$ Highest score is the estimated class $\Rightarrow \hat{\omega} = \arg\max_{j \in \{1,\ldots,m\}} y_j$
  - $\rightarrow$ Softmax activation as estimated probability:

$$\hat{P}(\omega|\mathbf{x}) = \frac{e^{y_\omega}}{\sum_{j=1}^{|\mathcal{W}|} e^{y_j}} \in [0,1]$$

Output layer

- Each output is a real-valued function: $y_j(\mathbf{x};\boldsymbol{\theta}) \in \mathbb{R}$ with $i \leq j \leq m$

- Regression tasks:
  → One regressor per output unit

- Classification tasks ($|\mathcal{W}|$ classes):
  → One unit per class $\Rightarrow m = |\mathcal{C}|$
  → Highest score is the estimated class $\Rightarrow \hat{\omega} = \arg\max_{j\in\{1,\ldots,m\}} y_j$
  → Softmax activation as estimated probability:

$$\hat{P}(\omega|\mathbf{x}) = \frac{e^{y_\omega}}{\sum_{j=1}^{|\mathcal{W}|} e^{y_j}} \in [0,1]$$

  → Binary case ($|\mathcal{C}| = 2$) may be modeled with a single neuron:

$$\hat{\omega} = \begin{cases} \omega_1 & \text{if } y \geq 0 \\ \omega_2 & \text{if } y < 0 \end{cases}$$

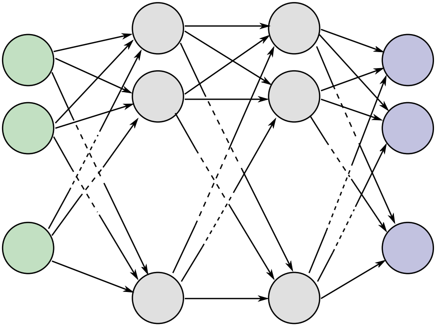
Contextualization

- Training is done following the *forward-backward* propagation

Contextualization

- Training is done following the *forward-backward* propagation

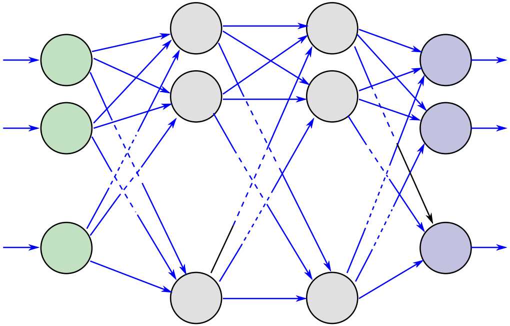
- **Steps**:



Contextualization

- Training is done following the *forward-backward* propagation

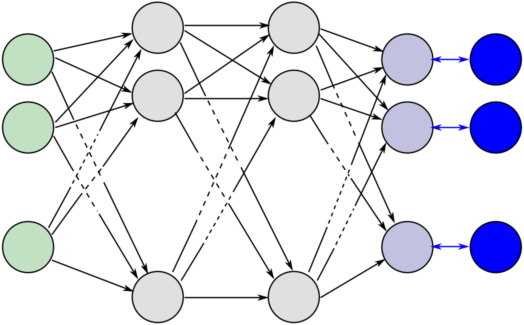
- **Steps**:
  1. Forward pass



Contextualization

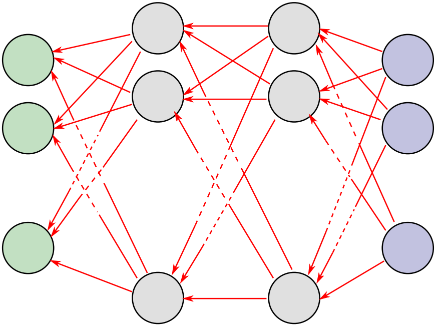
- Training is done following the *forward-backward* propagation

- **Steps**:
  1. Forward pass
  2. Loss computation



Contextualization

- Training is done following the *forward-backward* propagation

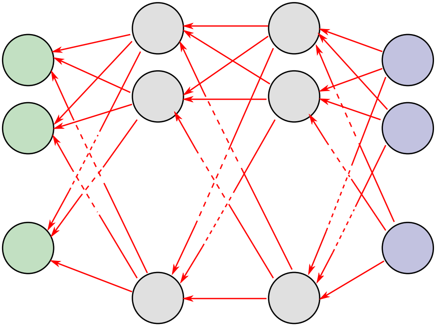


- **Steps**:
  1. Forward pass
  2. Loss computation
  3. Backward pass
  4. Weight update

Contextualization

- Training is done following the *forward-backward* propagation

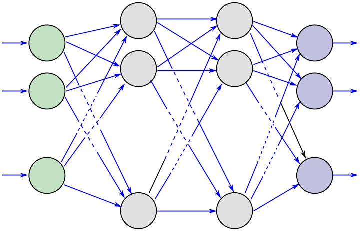
- **Steps**:
    1. Forward pass
    2. Loss computation
    3. Backward pass
    4. Weight update



- The process is done until a convergence criterion is reached

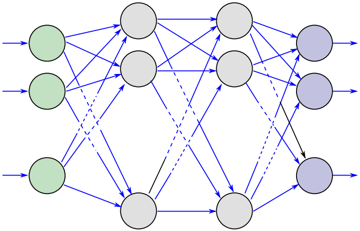
Forward-backward training

1. **Forward pass:**
   - Datum **x** goes from input to output
   - Weights $\boldsymbol{\theta}$ modify **x** across the network

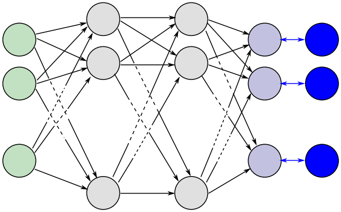


Forward-backward training

1. **Forward pass:**
   - Datum $\mathbf{x}$ goes from input to output
   - Weights $\boldsymbol{\theta}$ modify $\mathbf{x}$ across the network

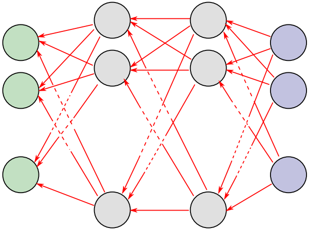


2. **Loss computation:**
   - Estimation $\hat{\mathbf{y}}$ is compared to reference $\mathbf{y}$
   - Depends on the task:
     - $\rightarrow$ Regression: MAE, MSE
     - $\rightarrow$ Classification: Categorical Cross Entropy



Forward-backward training

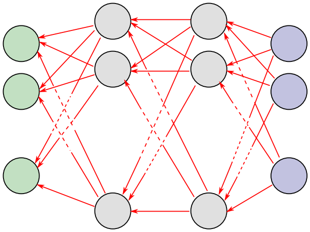
3. **Backward pass** (backpropagation) **:**
   - Gradient of the loss function with respect to $\boldsymbol{\theta}$
   - Chain rule for the derivatives of the (nested) equation
   - Gradients are propagated from output to input



Forward-backward training

3. Backward pass (backpropagation) **:**

- Gradient of the loss function with respect to $\boldsymbol{\theta}$
- Chain rule for the derivatives of the (nested) equation
- Gradients are propagated from output to input



4. Weight update:

- Update parameters $\boldsymbol{\theta}$ (weights and biases)
- Optimization methods:
  - $\rightarrow$ Stochastic Gradient Descent
  - $\rightarrow$ Momentum, RMSProp, Adam, ...

Classification scenarios

- Number of neurons must match labels $\Rightarrow m = |\mathcal{W}|$

Classification scenarios

- Number of neurons must match labels $\Rightarrow m = |\mathcal{W}|$
- Consider a case of $m = 3$:

Classification scenarios

- Number of neurons must match labels $\Rightarrow m = |\mathcal{W}|$
- Consider a case of $m = 3$:

Ideal case

Class A: $[1, 0, 0]$
Class B: $[0, 1, 0]$
Class C: $[0, 0, 1]$

Classification scenarios

- Number of neurons must match labels $\Rightarrow m = |\mathcal{W}|$
- Consider a case of $m = 3$:

Ideal case	Real case
Class A: $[1, 0, 0]$	Class A: $[0.67, 0.10, 0.23]$
Class B: $[0, 1, 0]$	Class B: $[0.05, 0.80, 0.15]$
Class C: $[0, 0, 1]$	Class C: $[0.25, 0.25, 0.50]$

Classification scenarios

- Number of neurons must match labels $\Rightarrow m = |\mathcal{W}|$
- Consider a case of $m = 3$:

Ideal case	Real case
Class A: $[1, 0, 0]$	Class A: $[0.67, 0.10, 0.23]$
Class B: $[0, 1, 0]$	Class B: $[0.05, 0.80, 0.15]$
Class C: $[0, 0, 1]$	Class C: $[0.25, 0.25, 0.50]$

- **Training** $\Rightarrow$ reference data must be adequated $\Rightarrow$ one-hot encoding
  $\rightarrow$ Mapping function: $\left[\omega_1, \ldots, \omega_{|\mathcal{W}|}\right] \rightarrow [0, 1]^{|\mathcal{W}|}$

Classification scenarios

- Number of neurons must match labels $\Rightarrow m = |\mathcal{W}|$
- Consider a case of $m = 3$:

Ideal case	Real case
Class A: $[1, 0, 0]$	Class A: $[0.67, 0.10, 0.23]$
Class B: $[0, 1, 0]$	Class B: $[0.05, 0.80, 0.15]$
Class C: $[0, 0, 1]$	Class C: $[0.25, 0.25, 0.50]$

- **Training** $\Rightarrow$ reference data must be adequated $\Rightarrow$ one-hot encoding
  - $\rightarrow$ Mapping function: $\left[\omega_1, \ldots, \omega_{|\mathcal{W}|}\right] \rightarrow [0, 1]^{|\mathcal{W}|}$
- **Categorical Cross Entropy** $\Rightarrow$ Typical loss function for classification

$$\mathcal{L}\left(\mathbf{y}, \hat{\mathbf{y}}\right) = -\sum_{j=1}^{|\mathcal{W}|} y_j \cdot \log\left(\hat{y}_j\right)$$

Classification scheme

Consider the following example:

Classification scheme

Consider the following example:

- Feature space $\mathbb{R}^d$ with $d = 2$ dimensions

Classification scheme

Consider the following example:

- Feature space $\mathbb{R}^d$ with $d = 2$ dimensions
- Three possible categories: $\mathcal{W} = \{\omega_1, \omega_2, \omega_3\}$

Classification scheme

Consider the following example:

- Feature space $\mathbb{R}^d$ with $d = 2$ dimensions
- Three possible categories: $\mathcal{W} = \{\omega_1, \omega_2, \omega_3\}$
- Test element $\mathbf{x} = [0.4, -2.2]$ with $\omega = \omega_2$

Classification scheme

Consider the following example:

- Feature space $\mathbb{R}^d$ with $d = 2$ dimensions
- Three possible categories: $\mathcal{W} = \{\omega_1, \omega_2, \omega_3\}$
- Test element $\mathbf{x} = [0.4, -2.2]$ with $\omega = \omega_2 \Rightarrow \mathbf{y} = [0, 1, 0]$

Classification scheme

Consider the following example:

- Feature space $\mathbb{R}^d$ with $d = 2$ dimensions
- Three possible categories: $\mathcal{W} = \{\omega_1, \omega_2, \omega_3\}$
- Test element $\mathbf{x} = [0.4, -2.2]$ with $\omega = \omega_2 \Rightarrow \mathbf{y} = [0, 1, 0]$
- Predicted vector $\hat{\mathbf{y}} = [0.1, 0.7, 0.2]$

Classification scheme

Consider the following example:

- Feature space $\mathbb{R}^d$ with $d = 2$ dimensions
- Three possible categories: $\mathcal{W} = \{\omega_1, \omega_2, \omega_3\}$
- Test element $\mathbf{x} = [0.4, -2.2]$ with $\omega = \omega_2 \Rightarrow \mathbf{y} = [0, 1, 0]$
- Predicted vector $\hat{\mathbf{y}} = [0.1, 0.7, 0.2]$

The associated loss is:

$$\begin{aligned}\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}) &= -\sum_{j=1}^{3} y_j \cdot \log\left(\hat{y}_j\right) \\ &= -\left[0 \cdot \log(0.1) + 1 \cdot \log(0.7) + 0 \cdot \log(0.2)\right] = 0.357\end{aligned}$$

Practical considerations

MLPs (and other Neural models) depict remarkable limitations:

Practical considerations

MLPs (and other Neural models) depict remarkable limitations:

1. **Overfitting**: *Memorizing* instead of *learning*

Practical considerations

MLPs (and other Neural models) depict remarkable limitations:

1. **Overfitting**: *Memorizing* instead of *learning*
2. **Vanishing/exploding gradients**: Gradients become negligible/excessively large producing stagnation/oscillations

Practical considerations

To prevent overfitting $\Rightarrow$ regularization strategies:

Practical considerations

To prevent overfitting $\Rightarrow$ regularization strategies:

- **L1/L2 regularization**: Adding additional penalties during training

Practical considerations

To prevent overfitting ⇒ regularization strategies:

- **L1/L2 regularization**: Adding additional penalties during training
- **Dropout**: Randomly disconnecting neurons during training

Practical considerations

To prevent overfitting ⇒ regularization strategies:

- **L1/L2 regularization**: Adding additional penalties during training
- **Dropout**: Randomly disconnecting neurons during training
- **Early stopping**: Stop training as validation loss increases

Practical considerations

To prevent overfitting ⇒ regularization strategies:

- **L1/L2 regularization**: Adding additional penalties during training
- **Dropout**: Randomly disconnecting neurons during training
- **Early stopping**: Stop training as validation loss increases
- **Data augmentation**: Apply transformations on (train) data to artificially expand the size of the set

Practical considerations

To prevent vanishing/exploding gradients:

Practical considerations

To prevent vanishing/exploding gradients:

- **Non-linear activations**: (Leaky) ReLU palliate vanishing issue

Practical considerations

To prevent vanishing/exploding gradients:

- **Non-linear activations**: (Leaky) ReLU palliate vanishing issue
- **Initialization strategies**: Weight initialization policies

Practical considerations

To prevent vanishing/exploding gradients:

- **Non-linear activations**: (Leaky) ReLU palliate vanishing issue
- **Initialization strategies**: Weight initialization policies
- **Batch/Layer normalization**: Maintaining activations in stable ranges

Practical considerations

To prevent vanishing/exploding gradients:

- **Non-linear activations**: (Leaky) ReLU palliate vanishing issue
- **Initialization strategies**: Weight initialization policies
- **Batch/Layer normalization**: Maintaining activations in stable ranges
- **Gradient clipping**: Limiting gradient value above threshold

T5: Linear methods and Perceptron

Fundamentos del Aprendizaje Automático

Curso 2025/2026