# I2ML :: CHEAT SHEET

The **I2ML**: Introduction to Machine Learning course offers an introductory and applied overview of "supervised" Machine Learning. It is organized as a digital lecture.

## NN Overview

**Deep learning** is a subfield of machine learning that uses neural networks with many layers to learn complex patterns in data.
**Representation learning:** DL automates feature engineering.

### Hidden Layers:

- Each layer adds degree of non-linearity that can learn more abstract representations.
- Output of hidden units serves as input for units in next layer.
- Too many hidden layers or too many units per layer cause overfitting.

## Perceptron

The **perceptron** is a single artificial neuron and basic computational unit of neural networks. It is restricted to learn only linear decision boundaries.
A neural network is built by combination of multiple perceptrons.
The perceptron is a weighted sum of input values, transformed by $\tau$:

$$f(\mathbf{x}) = \tau(w_1x_1 + \ldots + w_px_p + b) = \tau(\mathbf{w}^\top\mathbf{x} + b).$$

### Structure:

A neuron performs a 2-step computation:
- **Affine Transformation:** weighted sum of inputs plus bias: $z_{in} = w_1x_1 + \ldots + w_px_p + b$.
- **Non-linear Activation:** a non-linear transformation: $\tau(z_{in})$.

Weights $\mathbf{w}$ are connected to edges from the input layer.
For an explicit graphical representation, we do a simple trick:
- Add a constant feature to the inputs: $\tilde{\mathbf{x}} = (1, x_1, \ldots, x_p)^\top$
- Absorb the bias into the weight vector: $\tilde{\mathbf{w}} = (b, w_1, \ldots, w_p)^\top$

A **forward pass**: input vector being "fed" to neurons on the left followed by a sequence of computations performed from left to right.

### Activation function:

A single neuron represents different functions depending on the choice of activation function $\tau$.
- Identity function gives us the simple linear regression: $f(\mathbf{x}) = \tau(\mathbf{w}^\top\mathbf{x}) = \mathbf{w}^\top\mathbf{x}$
- Logistic function gives us the logistic regression: $f(\mathbf{x}) = \tau(\mathbf{w}^\top\mathbf{x}) = \frac{1}{1+\exp(-\mathbf{w}^\top\mathbf{x})}$

### Hypothesis space:

$$\mathcal{H} = \{f : \mathbb{R}^p \to \mathbb{R} | f(\mathbf{x}) = \tau(\sum_{j=1}^{p} w_jx_j + b), \mathbf{w} \in \mathbb{R}^p, b \in \mathbb{R}\}$$

If $\tau$ is the logistic sigmoid or identity function, $\mathcal{H}$ corresponds to the hpothesis space of logistic or linear regression, respectively.

### Optimization:

Minimize the empirical risk $\mathcal{R}_{\text{emp}} = \frac{1}{n}\sum_{i=1}^{n} L\left(y^{(i)}, f\left(\mathbf{x}^{(i)}\right)\right)$.
Regression: L2 loss. Binary classification: cross-entropy loss.
- For a single neuron and both choices of $\tau$, the loss function is convex.
- The global optimum can be found with an iterative algorithm like GD.
- A single neuron with logistic sigmoid function trained with the Bernoulli loss yields the same result as logistic regression when trained until convergence.

## Single hidden layer networks

**Single hidden layer networks** have a set of neurons in hidden layers and one or more output neurons. Multiple inputs are simultaneously fed to the network. Each neuron in the hidden layer performs a 2-step computation as a single neuron. The final output of the network is then calculated by another 2-step computation performed by the neuron in the output layer.
**Hidden Layer Activation Function:**
**ReLU:** $\tau(v) = \max(0, v)$. **Signoid:** $\tau(v) = \frac{1}{1+\exp(-v)}$.

### Multi-class Classification:

Multiple neurons in the output layer. Each neuron will represent a specific class. For $g$-class classification, $g$ output units: $\boldsymbol{f} = (f_1, \ldots, f_g)$.
$m$ hidden neurons $z_1, \ldots, z_m$ with $z_j = \sigma((W)_j^\top\mathbf{x}), j = 1, \ldots, m$.
$f_{in,k} = \boldsymbol{U}_k^\top\boldsymbol{z}, \boldsymbol{z} = (z_1, \ldots, z_m)\top, k = 1, \ldots, g$.

Apply a **softmax** activation function to the output layer. This gives us a probability distribution over g different possible classes:

$$f_{out,k} = \tau_k(f_{in,k}) = \frac{\exp(f_{in,k})}{\sum_{k'=1}^{g} f_{in,k'}}.$$

Derivative $\frac{\partial\tau(f_{in})}{\partial f_{in}} = \text{diag}(\tau(f_{in})) - \tau(f_{in})\tau(f_{in})^\top$.

The loss function for a softmax classifier is

$$L(y, f(\mathbf{x})) = \sum_{k=1}^{g}[y = k]\log\frac{\exp(f_{in,k})}{\sum_{k'=1}^{g} f_{in,k'}}, [y = k] = \begin{cases} 1, & \text{if } \quad y = k \\ o, & \text{otherwise} \end{cases}$$

This is equivalent to the cross-entropy loss when the label vector y is one-hot coded. There is no analytical solution.

## ML FNNs

We allow an arbitrary amount $l$ of hidden layers as multi-layer (ML).
**Feedforward neural networks (FNNs)**: inputs are passed through the network from left to right, no feedback-loops are allowed.

### Chain Structure:

$$f(\mathbf{x}) = \tau \circ \phi \circ \sigma^{(l)} \circ \phi^{(l)} \circ \sigma^{(l-1)} \circ \phi^{(l-1)} \circ \ldots \circ \sigma^{(1)} \circ \phi^{(1)}.$$

$\sigma^{(i)}$: $i$-th hidden layer activation function; $\phi^{(i)}$: weighted sum of $i$-th layer; $\tau$ and $\phi$: corresponding components of the output layer.
Each hidden layer has:
- An associated weight matrix $\boldsymbol{W}^{(i)}$, bias $\boldsymbol{b}^{(i)}$ and activations $\boldsymbol{z}^{(i)}$.
- $\boldsymbol{z}^{(i)} = \sigma^{(i)}(\phi^{(i)}) = \sigma^{(i)}(\boldsymbol{W}^{(i)\top}\boldsymbol{z}^{(i-1)} + \boldsymbol{b}^{(i)})$ where $z^{(0)} = \mathbf{x}$.

Without non-linear activations in the hidden layers, the network can only learn linear decision boundaries.

## Backpropagation

We would like to run ERM by GD on:

$$\mathcal{R}_{\text{emp}}(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} L\left(y^{(i)}, f\left(\mathbf{x}^{(i)} \mid \boldsymbol{\theta}\right)\right).$$

Backprop training of NNs runs in 2 alternating steps, for one $\mathbf{x}$:
- **Forward pass:** Inputs flow through model to outputs, then compute the observation loss..
- **Backward pass:** Loss flows backwards to update weights so error is reduced, as in GD.

### Example:

One hidden layer, logistic activations with L2 loss.
Forward pass:
$z_{i,in} = \boldsymbol{W}_i^\top\mathbf{x} + b_i, \quad z_{i,out} = \sigma(z_{i,in})$,
$f_{in} = \boldsymbol{u}^\top\boldsymbol{z} + c, \quad f_{out} = \tau(f_{in}), \quad L(y, f(\mathbf{x})) = \frac{1}{2}(y - f_{out})^2$.
Backward pass:
$\frac{\partial L(y,f(\mathbf{x}))}{\partial u_i} = \frac{\partial L(y,f(\mathbf{x}))}{\partial f_{out}}\frac{\partial f_{out}}{\partial f_{in}}\frac{\partial f_{in}}{\partial u_i}$
$\frac{\partial L(y,f(\mathbf{x}))}{\partial f_{out}} = -(y - f_{out}), \quad \frac{\partial f_{out}}{\partial f_{in}} = \sigma(f_{in})(1 - \sigma(f_{in})), \quad \frac{\partial f_{in}}{\partial u_i} = z_{i,out}$.
$u_i^{[new]} = u_i^{[old]} - \alpha\frac{\partial L(y,f(\mathbf{x}))}{\partial u_i}$.
$\frac{\partial L(y,f(\mathbf{x}))}{\partial W_{ij}} = \frac{\partial L(y,f(\mathbf{x}))}{\partial f_{out}}\frac{\partial f_{out}}{\partial f_{in}}\frac{\partial f_{in}}{\partial z_{j,out}}\frac{\partial z_{j,out}}{\partial z_{j,in}}\frac{\partial z_{j,in}}{\partial W_{ij}}$
$\frac{\partial f_{in}}{\partial z_{j,out}} = u_j, \quad \frac{\partial z_{j,out}}{\partial z_{j,in}} = \sigma(z_{j,in})(1 - \sigma(z_{j,in})), \quad \frac{\partial z_{j,in}}{\partial W_{ij}} = x_i$.
$W_{ij}^{[new]} = W_{ij}^{[old]} - \alpha\frac{\partial L(y,f(\mathbf{x}))}{\partial W_{ij}}$.