DLE Course 2024/2025

1. Recap of Machine Learning, Multi-Layer Perceptron	Lab 1: **Double Descent**
2. Backpropagation	Seminar 1
3. Convolutional Neural Networks	Lab 2: **Backpropagation, Computational Graph**
4. Training Deep Models	Seminar 2
5. Regularization Methods for NNs	Lab 3: **From Scratch: Initialization & regularization**
6. Stochastic Gradient Descent (SGD)	Seminar 3
7. Adversarial Patterns, Robust Learning Approaches	Lab 4: **CNN Fine-Tuning, Visualization & Adversarial Patterns**
8. Adaptive SGD Methods	Seminar 4
9. Learning Representations I: Word Vectors, Metric Learning	Lab 5: **Metric Learning**
10. Learning Representations II: Unsupervised Learning, VAE	Lab 6: **VAEs**
11. Graph Neural Networks	Seminar 5
12. Self-Attention, Transformers	Lab 7: **GNNs / Transformers**
13. TBA	Seminar 6

- Practical labs: implementation of selected methods (Python/PyTorch),
- Theoretical labs: solving theoretical assignments
  assignments are published in advance, you are expected to present/discuss solutions
- More details at the lab

Deep Learning (BEV033DLE)
Lecture 1.
Recap of ML, Multi-Layer Perceptron

Czech Technical University in Prague

- Machine Learning
  - Regression and classification (logistic, multinomial logistic models)
  - Test and training losses
  - Maximum likelihood
  - Bias-Variance trade-off
  - Generalization and overfitting
- Artificial Neuron, MLP, Perceptron
  - Universal approximation and capacity

Shallow Learning

Regression

- Assume y statistically depends on x in nature according to $p^*(y|x)$

  (functional dependence corrupted by noise, uncontrolled effects of other inputs)

  e.g. $y = f^*(x) + \varepsilon, \quad \varepsilon \sim \mathcal{N}(0, \sigma^2)$

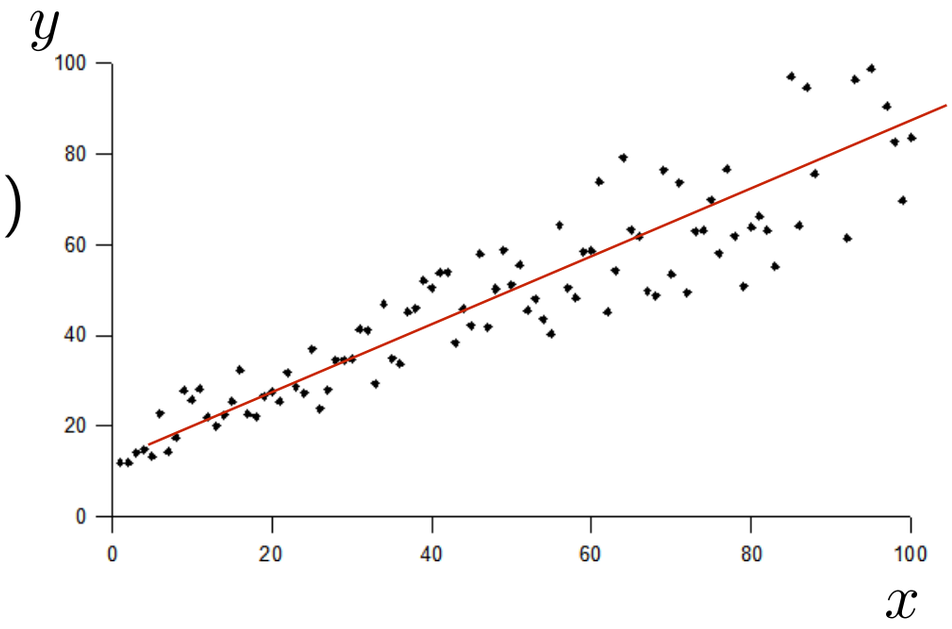
- Training (abstractly):
  - Training set $\mathcal{T} = \{x_i, y_i\}_{i=1}^N$ i.i.d. from p^*

    (x_i are given somehow and y_i is drawn from $p^*(y|x_i)$)

    ⇩ Training Algorithm

    Predictor $y = f(x)$

    e.g. $y = w^\mathsf{T} x + b$ — linear model



- **Test loss** — how well are we predicting?
  - Could be an application-specific loss (e.g. IoU in detection)
  - MSE loss (default choice, mathematical convenience):

    For a test input x, $\mathcal{L}_{\text{MSE}}(x) = \mathbb{E}_{y \sim p^*(y \mid x)}[(y - f(x))^2]$,

    then can take a sum over x where we want to predict (not essential here)
  - Performance of the predictor, not of the algorithm, it depends on $\mathcal{T}$

Regression: Bias-Variance Decomposition

- How well the Algorithm performs?
  - In expectation over $\mathcal{T}$:

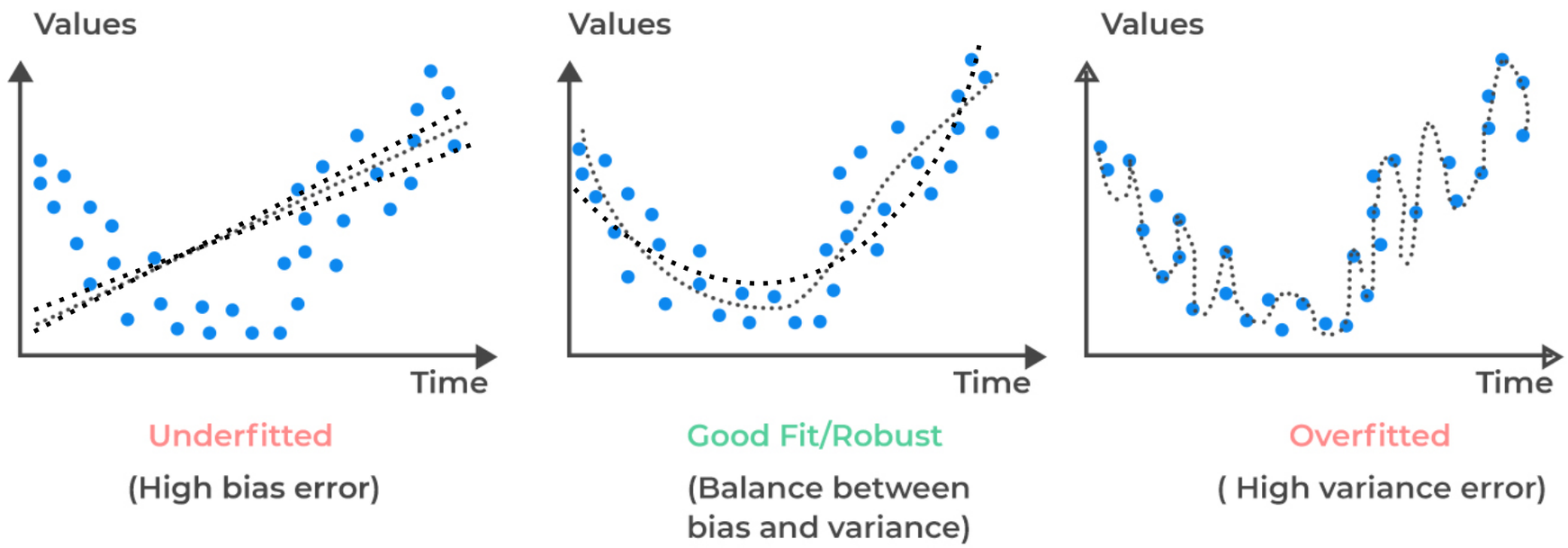
$$\mathbb{E}_{\mathcal{T}}[\mathcal{L}_{\text{MSE}}(x)] = \mathbb{E}_{\mathcal{T}}\left[\mathbb{E}_{y\sim p^*(y\,|\,x)}[(y - f_{\mathcal{T}}(x))^2]\right]$$

$$= \underbrace{\mathbb{E}_{y\sim p^*(y|x)}[(y - f^*(x))^2]}_{\text{Irreducible data variance}} + \underbrace{(f^*(x) - \bar{f}(x))^2}_{\text{Learning squared bias}} + \underbrace{\mathbb{E}_{\mathcal{T}}[(\bar{f}(x) - f_{\mathcal{T}}(x))^2]}_{\text{Learning variance}}$$

$f^*(x) = \mathbb{E}_{y\sim p^*(y\,|\,x)}[y]$ — true mean

$\bar{f}(x) = \mathbb{E}_{\mathcal{T}}[f(x)]$ — average predictor

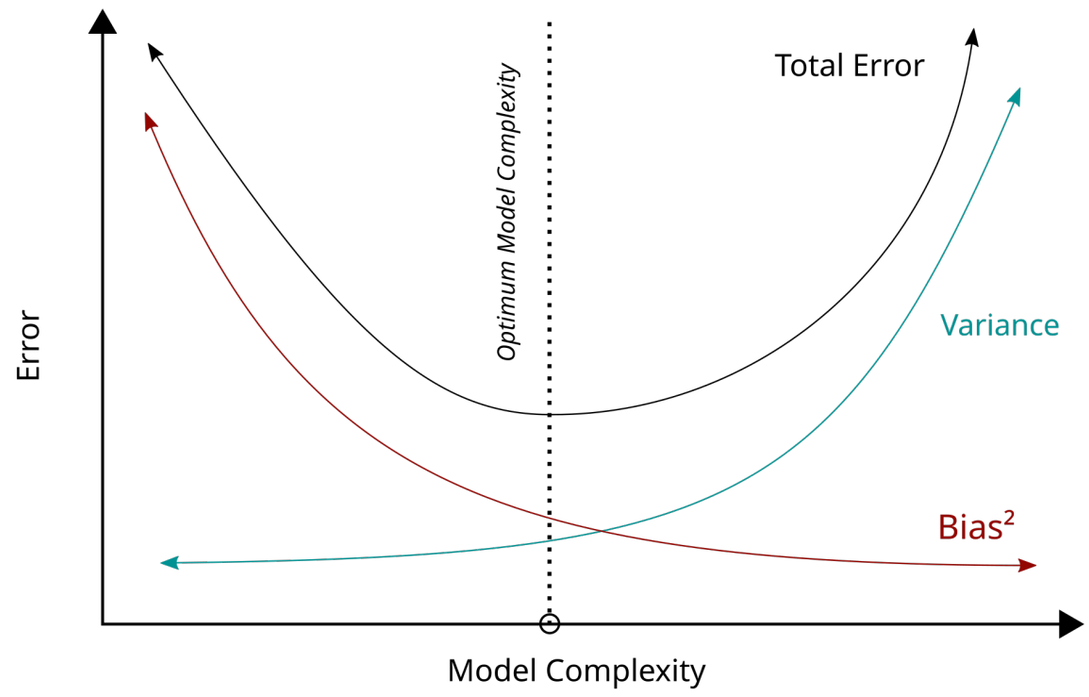
Classical Illustration:



Model + learning induce a preferred solution (inductive bias) — it may or may not be good for the data at hand

Regression: Bias-Variance Decomposition

- Classical paradigm:
  - Too simple models have high bias
  - Too complex models will overfit to the noise and as a result will have high variance



- But:
  - Bias depends on the model, learning method and the true dependence
  - Variance depends on the model, learning method and data distribution, but can be reduced with more training samples

=> Can regularize instead of complexity control
/design the learning to use complexity adaptively

Regression: Training

Which **training criterion** to minimize?

- 1) same loss we used to measure the test performance
  - MSE loss: $\min_\theta \frac{1}{N}\sum_{i=1}^{N}(y_i - f_\theta(x_i))^2$, f_θ — parametric predictor
  - More difficult with IoU loss
- 2) Likelihood of a **Probabilistic Predictor**
  - $p_\theta(y\,|\,x)$ — parametric conditional distribution
  - Maximum Likelihood training: $\max_\theta p_\theta(\mathcal{T}) = \max_\theta \prod_i p_\theta(y_i\,|\,x_i)$

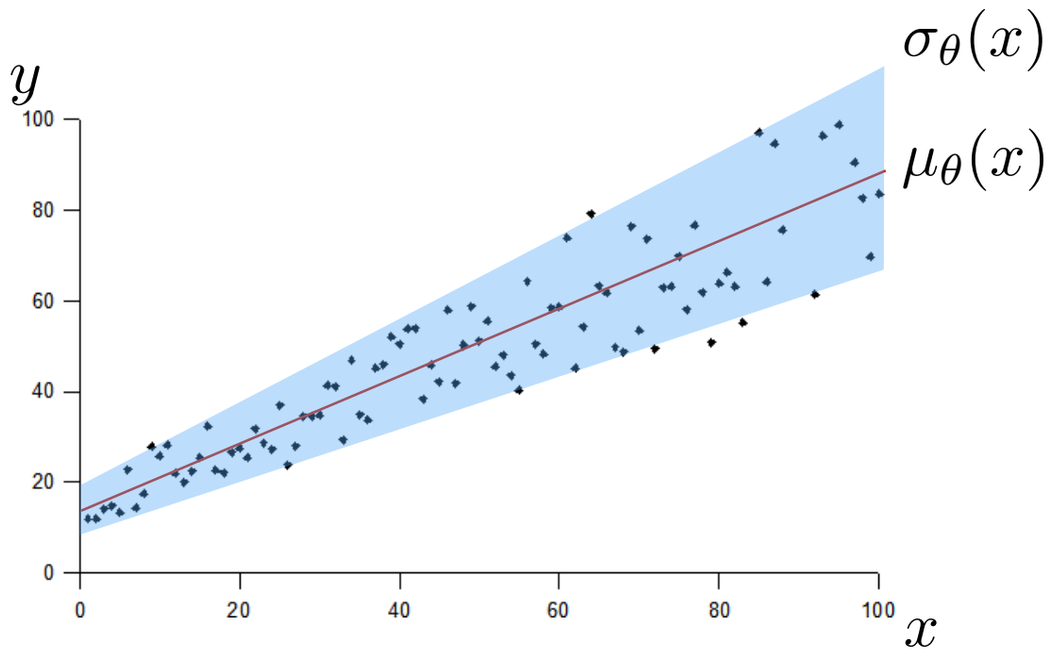
    e.g., for $p_\theta(y\,|\,x) = p_{\mathcal{N}}(y; \mu_\theta(x), \sigma^2_\theta(x))$ ML: $\min_\theta \left[\frac{1}{N}\sum_i \frac{(y_i - \mu_\theta(x_i))^2}{2\sigma^2_\theta(x)} + \log\sigma_\theta(x)\right]$

    (coincides with MSE loss when σ_θ is constant)

    — consistent, asymptotically efficient
  - Predictor: $f(x) = \mathbb{E}_{y\sim p_\theta(y|x)}[y] = \mu_\theta(x)$,

    — mean function of $p_\theta(y|x)$

    $\sigma_\theta(x)$ — scatter estimate (uncertainty)
- 3) Robust models, *etc*.



Classification

- Nature: $p^*(y, x)$, $y \in \mathcal{Y}$ — finite set
  - **Binary classification:** $\mathcal{Y} = \{-1, 1\}$
- **Classifier**: $x \mapsto s(x) \in R$ — **score**, *e.g.* $s(x) = w^\mathsf{T} x + b$

$$y = f(x) = \text{sign}(s(x))$$

- Test 0-1 loss (error rate): $\mathcal{L} = \mathbb{E}_{(x,y)\sim p^*}[\![y \neq f(x)]\!] \approx \frac{1}{N}\sum_i [\![y_i \neq f(x_i)]\!]$

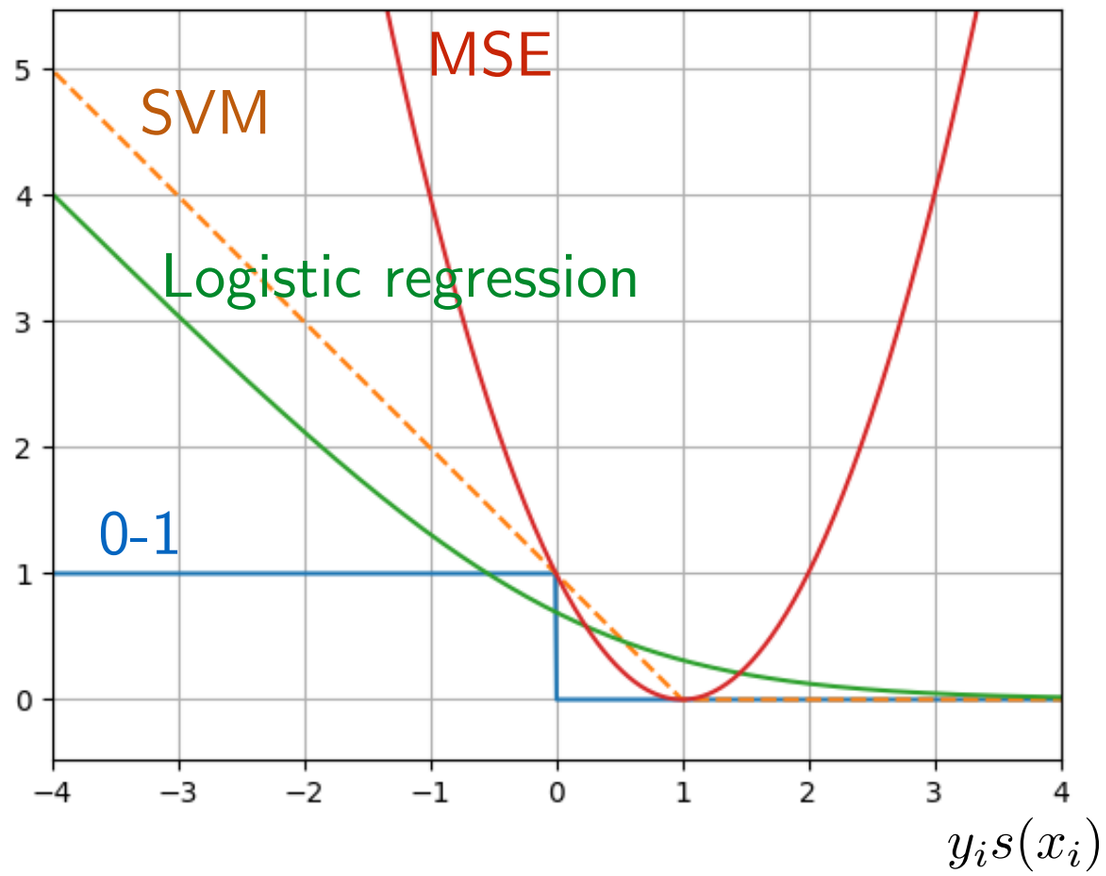
  (general decision making: $f(x) \in \mathcal{D} \neq \mathcal{Y}$, $l(y, d)$, Risk $\mathcal{R} = \mathbb{E}_{(x,y)\sim p^*}[l(y, f(x))]$)
- **Training criterion?**
  - θ – vector of parameters to train
  - 0-1 loss (error rate) is not differentiable
  - A loss function of the score, *e.g.*

    MSE: $\mathcal{L}_{\text{MSE}}(\theta) = \frac{1}{N}\sum_i (y_i - s(x_i))^2$,

    ML approach: Logistic regression

    ...

Training loss functions



Classification: Logistic Regression

Logistic Regression

- Probabilistic Classifier: $p(y \,|\, x)$, $y \in \{-1, 1\}$
- Logistic regression model: $\log \dfrac{p(y{=}1 \,|\, x)}{p(y{=}-1 \,|\, x)} = s(x)$

  (score interpretability as log odds + math convenience)

  $\Rightarrow p(y = 1 \,|\, x) = \frac{1}{1+e^{-s}} =: \mathcal{S}(s)$ — logistic sigmoid function

  ($\mathcal{S}^{-1}(\pi) = \log \frac{\pi}{1-\pi}$ — logit function)

Maximum Likelihood:

- $\mathcal{L}(\theta) = -\sum_i \log p_\theta(y_i \,|\, x_i) = \sum_i \log(1 + e^{-y_i s_\theta(x_i)})$ — log. regression loss
- Classifier, maximum a posteriori (MAP): $y = \operatorname{argmax}_y p(y \,|\, x) = \operatorname{sign}(s(x))$

Multinomial (Logistic) Regression

- $\mathcal{Y}$ — finite set, $p(y \,|\, x)$ — categorical distribution
- $s(x) \in \mathbb{R}^{|\mathcal{Y}|}$ — vector of scores (score per class)
- Model: $\log \dfrac{p(y{=}k \,|\, x)}{p(y{=}l \,|\, x)} = s_k - s_l \quad \Rightarrow \quad p(y = k \,|\, x) = \frac{e^{s_k}}{\sum_j e^{s_j}} =: \operatorname{softmax}(s)_k$

  (same interpretation, independence of irrelevant alternatives)

Classification: Multinomial Regression

- **Multinomial (Logistic) Regression**
  - $\mathcal{Y}$ — finite set, $p(y \,|\, x)$ — categorical distribution
  - $s(x) \in \mathbb{R}^{|\mathcal{Y}|}$ — vector of scores (score per class)
  - Model: $\log \frac{p(y{=}k \,|\, x)}{p(y{=}l \,|\, x)} = s_k - s_l \quad \Rightarrow \quad p(y = k \,|\, x) = \frac{e^{s_k}}{\sum_j e^{s_j}} =: \mathrm{softmax}(s)_k$
- ML: $\mathcal{L}(\theta) = -\sum_i \log p(y_i \,|\, x_i) = \sum_i \underbrace{-\langle t_i, \mathrm{logsoftmax}(s(x_i)) \rangle}$

  $t_i = \mathrm{onehot}(\mathrm{y_i}) \quad$ *e.g.*, $t_i = \begin{pmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{pmatrix}$

  Cross-entropy of categorical distributions t_i and $\mathrm{softmax}(s(x_i))$
- Classifier
  - MAP: $\mathrm{argmax}_k \, p(y = y \,|\, x) = \mathrm{argmax}_k \, s(x)_k$ — highest score is most likely prediction

Empirical Risk and Generalization

- Risk vs. Empirical Risk
  - Predictor or classifier $f(x)$, validation set $\mathcal{T}$ of size N
  - Risk: $\mathcal{R}(f) = \mathbb{E}_{(x,y)\sim p^*}[l(y, f(x))] \approx \frac{1}{N}\sum_{(x,y)\in\mathcal{T}} l(y, f(x)) =: \mathcal{R}_\mathcal{T}(f)$

    How good is this approximation?

- Concentration Inequalities
  - $l(y, f(x))$ is random, sum of random numbers tends to Normal distribution.
  - Chebyshev's Inequality: $\mathbb{P}(|\mathcal{R}(f) - \mathcal{R}_\mathcal{T}(f)| > \varepsilon) < \dfrac{\mathbb{V}[l(y, f(x))]}{N\varepsilon^2}$
  - Heffding's Inequality: $\mathbb{P}(|\mathcal{R}(f) - \mathcal{R}_\mathcal{T}(f)| > \varepsilon) < 2\exp\Big\{-\dfrac{2N\varepsilon^2}{\Delta l^2}\Big\}$,

    where $\Delta l = l_{\max} - l_{\min}$ (if loss is bounded)

- What if $\mathcal{T}$ is the training set?
  - Consider choosing f from a set $\mathcal{F}$
  - Need a *uniform* bound: $\mathbb{P}(\exists f \in \mathcal{F} \;\; |\mathcal{R}(f) - \mathcal{R}_\mathcal{T}(f)| > \varepsilon) <?$

Empirical Risk and Generalization

- Guarantees for the risk while selecting f from a set $\mathcal{F}$?

  - Example: Hoeffding's inequality + Union Bound:

    $$\mathbb{P}(\exists f \in \mathcal{F} \;\; |\mathcal{R}(f) - \mathcal{R}_{\mathcal{T}}(f)| > \varepsilon) < 2|\mathcal{F}| \exp\Big\{ - \frac{2N\varepsilon^2}{\Delta l^2} \Big\}$$

  - For better interpretability, can be converted to the upper bound form:

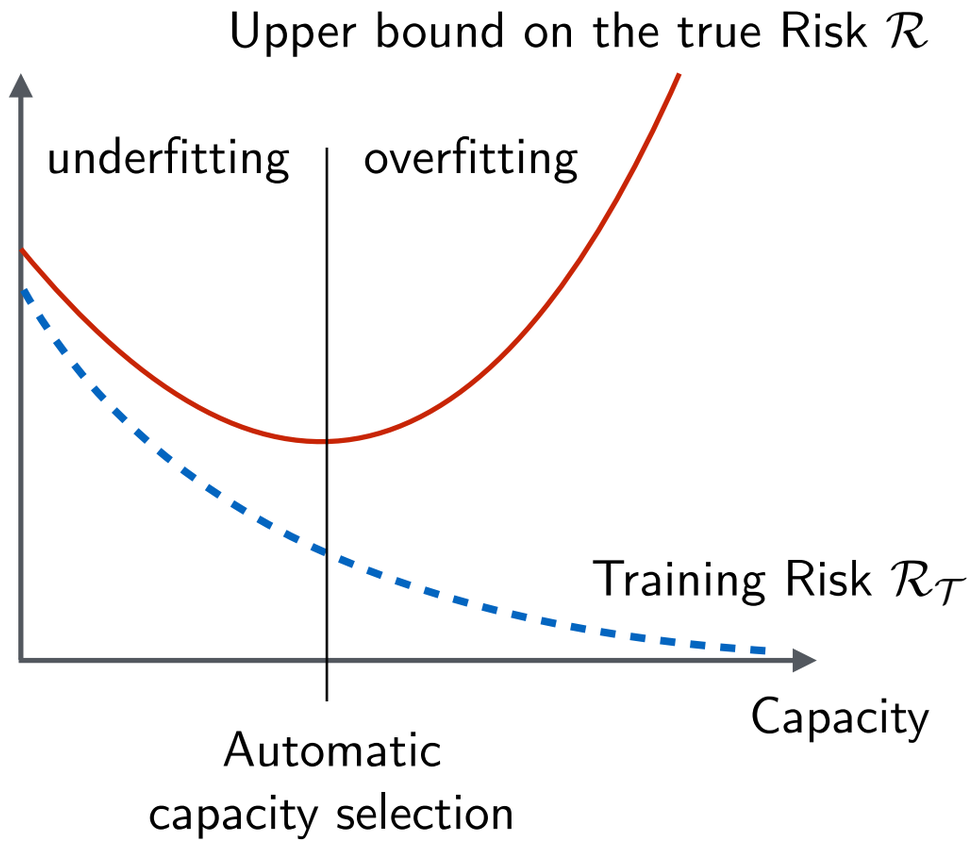
    $\mathcal{R}(f) \leq \mathcal{R}_{\mathcal{T}}(f) + B(N, \mathcal{F}, \alpha)$ with probability (confidence) α over random $\mathcal{T}$

Different bounds exist, $B(N, \mathcal{F}, \alpha)$
- typically decreases with N as $\frac{1}{\sqrt{N}}$
- quickly grows with $1 - \alpha$
- grows with "capacity" of $\mathcal{F}$
  (*e.g.* VC dimension, fat shattering dimension,...)

- Motivates structured risk:

  Learning should minimize combination

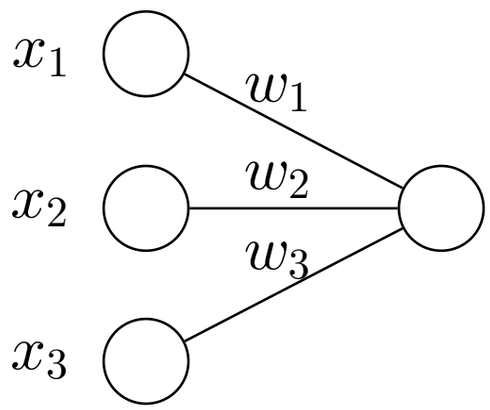
  of loss and complexity



Neural Networks

Artificial Neuron

Neuron: combination of a linear transform and a non-linear function



$a = w^\mathsf{T} x + b$ — pre-activation

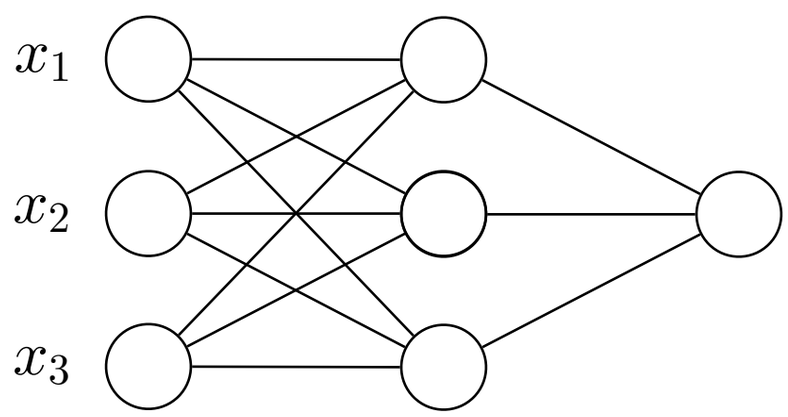
$y = \varphi(a)$ — activation (neuron output)

- McCulloch-Pitts (MCP) Neuron, 1943:
  - A computational model of the brain
  - Inputs: excitatory or inhibitory and firing or quiet = {-1,0,1}
  - Weights: present / absent connections = {0,1}

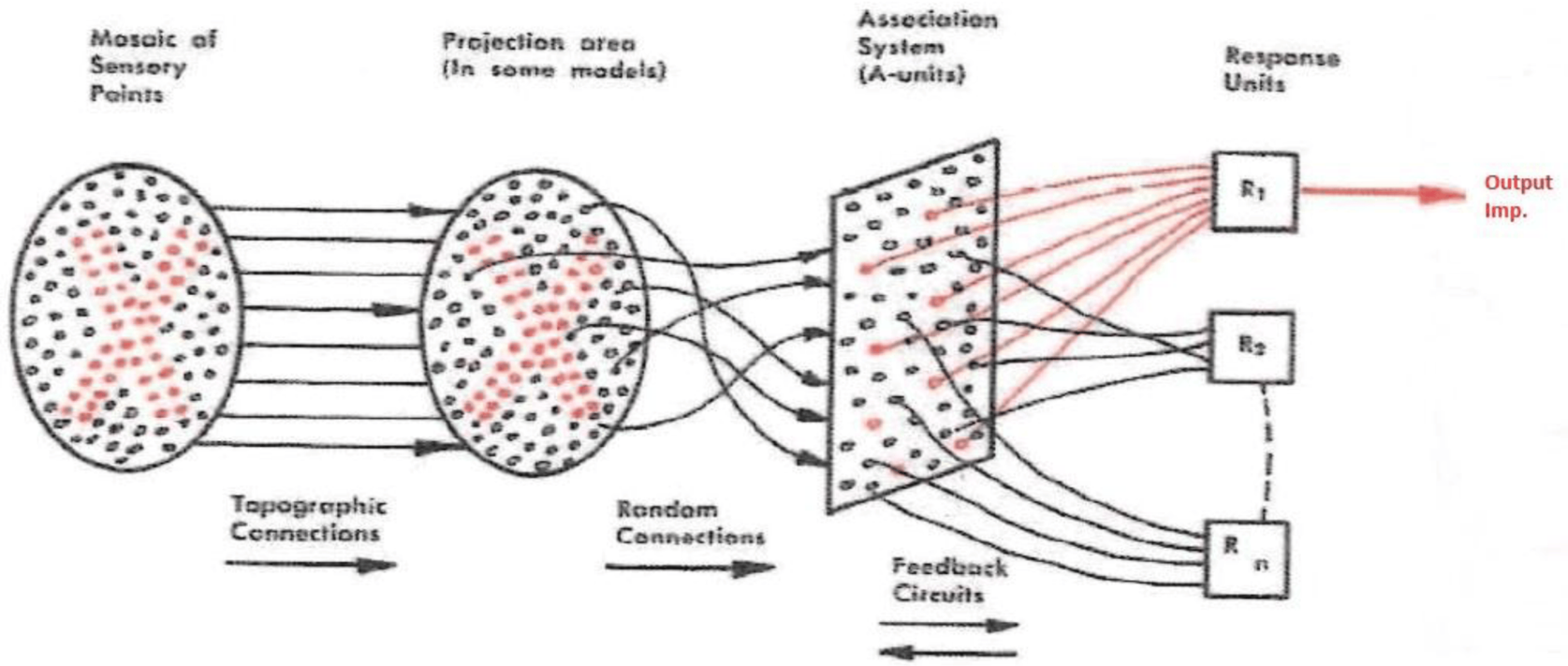
Interesting facts:

- A two layer network can implement any Boolean function (via CNF) — but this may require exponentially many neurons
- Any finite state machine can be simulated by a MCP neural network with discrete time and feedback connections

two-layer network



Perceptron



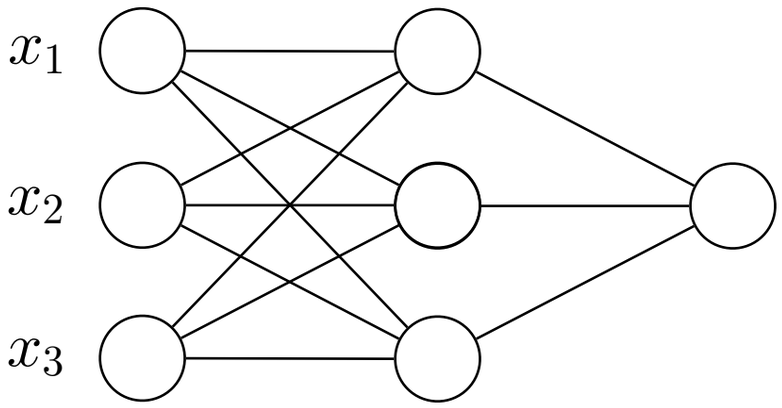
- Mark I Perceptron Machine (1958):
  - Input: 20x20 input (sensory) units
  - First layer: 512 hidden (association) units, randomly connected
  - Output layer: 8 units (response), adjustable weights
  - Sensory and association units have continuous response

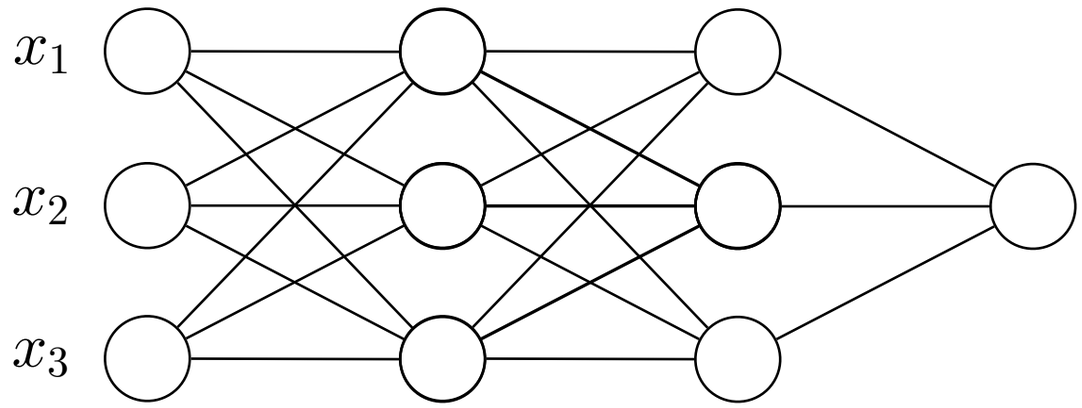
potentiometer

"Devices of this sort are expected ultimately to be capable of concept formation, language translation, collation of military intelligence, and the solution of problems through inductive logic."

F. Rosenblatt

Multi-Layer Perceptron





- **Theorem** (Cybenko, 1989) Every smooth function on $[0,1]^n$ can be approximated arbitrarily well by a network with sigmoid units and two layers. In other words, given a smooth function $f : [0,1]^n \to \mathcal{R}$ and an $\varepsilon > 0$, there is a sum

$$G(x) = \sum_{j=1}^{N} \alpha_j \mathcal{S}(w_j^T x + b_j)$$

s.t. $|f(x) - G(x)| \leqslant \varepsilon$ for all $x \in [0,1]^n$.

Remark:

- There are also "dual" universal approximation theorems that restrict the width of the network (i.e. number of units per layer) and allow arbitrary network depth.
- We limit the expressive power once we fix a network architecture.

Learning Neural Networks: Generalisation & Overfitting

✦ Zang et al. (2018): Understanding deep learning requires rethinking generalization

Experiment:

- CIFAR10 Dataset: $\sim 5 \cdot 10^4$ training images (32x32), 10 classes
- Network with $\sim 10^5$ parameters
- The network learned by SGD and additional regularization (data augmentation dropout, etc.)
- Archived $> 95\%$ test accuracy
- Capacity tests:
  - train with random labels
  - shuffle pixels, same in all images
  - shuffle pixels in each image
  - Random Gaussian pixels with statistics matching the image mean and variance