OUTLINE

I. Introduction
II. Background

III. Fitting a Model
   • How to minimize a function?
   • Backpropagation
   • Improved Gradient Descent
   • The PyTorch Framework

IV. Supervised Learning
V. Unsupervised Learning
VI. Fantastic DNNs: How to choose them, how to train them
OUTLINE

I. Introduction
II. Background
III. Fitting a Model
IV. Supervised Learning
V. Unsupervised Learning
VI. Fantastic DNNs: How to choose them, how to train them
OUTLINE

I. Introduction
II. Background
III. Fitting a Model
IV. Supervised Learning
   • Regression
   • How to Choose the Loss?
   • Detection & Classification
   • Over and Underfitting
V. Unsupervised Learning
VI. Fantastic DNNs: How to choose them, how to train them
IV. Supervised Learning

Supervised Learning

Labeled training data

\[ \mathcal{T} = \{(x_t, y_t)\}_{t=1}^{T} \]

Input \rightarrow Labels/Targets

\[ [x_1, x_2, \ldots, x_T] \rightarrow [y_1, y_2, \ldots, y_T] \]
IV. Supervised Learning

Supervised Learning

Labeled training data

\[ \mathcal{T} = \{ (x_t, y_t) \}_{t=1}^{T} \]

Input

\[ [x_1, x_2, \ldots, x_T] \]

Labels/Targets

\[ [y_1, y_2, \ldots, y_T] \]

Training

Learned Model

\[ f_\theta \]
IV. Supervised Learning

**Overview**

Supervised Learning

- **Labeled training data**
  \[ \mathcal{T} = \{ (x_t, y_t) \}_{t=1}^{T} \]
  - Input: \([x_1, x_2, \ldots, x_T]\)
  - Labels/Targets: \([y_1, y_2, \ldots, y_T]\)

- **Training**
  - Test: \(\tilde{x}\)

- **Learned Model**
  \(f_\theta\)

- **Estimate**
  \(\tilde{y}\)
## IV. Supervised Learning

### Overview

#### Supervised Learning

Labeled training data

\[ T = \{(x_t, y_t)\}_{t=1}^{T} \]

- **Input**
- **Labels/Targets**

**Training**

- **Test**
- **Estimate**

1. **Discrete case**: («one-hot»)
   - Classification
   - Ex. application: *dog breed*
   - Example: *German Shepherd*

\[ y = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \]
IV. Supervised Learning

Supervised Learning

Labeled training data

\[ \mathcal{T} = \{(x_t, y_t)\}_{t=1}^T \]

Input

Labels/Targets

Test

\[ \tilde{x} \]

Training

Learned Model

\[ f_\theta \]

Estimate

\[ \tilde{y} \]

1. **Discrete case**: («one-hot»)
   - Classification
   - Ex. application: *dog breed*
     
     \[ y = \begin{pmatrix} 1 \end{pmatrix} \]
     
     ![German Shepherd](image)

2. **Continuous case**:
   - Regression
   - Ex. application: *head pose*
     
     ![Head Pose](image)
IV. Supervised Learning

Supervised Learning

Labeled training data

\[ \mathcal{T} = \{(x_t, y_t)\}_{t=1}^T \]

- **Input**
- **Labels/Targets**

Test

\[ \tilde{x} \]

Training

Learned Model

\[ f_\theta \]

Estimate

\[ \tilde{y} \]

1. **Discrete case**: («one-hot»)
   - Classification
   - Ex. application: *dog breed*
     - \[ y = \begin{bmatrix} 0 \ 1 \ 0 \end{bmatrix} \]
     - 13: German Shepherd

2. **Continuous case**:
   - Regression
   - Ex. application: *head pose*
     - \[ y = \begin{bmatrix} 0.1 \\ 0.6 \\ 0.3 \end{bmatrix} \]

3. **Sparse case**:
   - Multi-label classification
   - Ex. application: *image labelling*
     - \[ y = \begin{bmatrix} 0.1 \\ 0.6 \\ 0.3 \end{bmatrix} \]
     - man
     - palm tree
     - phone
IV. Supervised Learning ▶ Regression

Generalizing linear regression

- Training set: \( \mathcal{T} = \{ (x_t, y_t) \}_{t=1}^{T} \)
- Models: \( f_{\theta}(x) = ax + b \)
- Parameters: \( \theta = [a, b]^T \in \mathbb{R}^2 \)
- Total Loss: \( g(\theta) = L(f_{\theta}, \mathcal{T}) = \frac{1}{T} \sum_{t=1}^{T} (f_{\theta}(x_t) - y_t)^2 \)
IV. Supervised Learning

Regression

Generalizing linear regression

- Training set: $\mathcal{T} = \{(x_t, y_t)\}_{t=1}^{T}$
- Models: $f_\theta(x) = ax + b$
- Parameters: $\theta = [a, b]^\top \in \mathbb{R}^2$
- Total Loss: $g(\theta) = L(f_\theta, \mathcal{T}) = \frac{1}{T} \sum_{t=1}^{T} (f_\theta(x_t) - y_t)^2$
Generalizing linear regression

- Training set: $\mathcal{T} = \{(x_t, y_t)\}_{t=1}^T$
- Models: $f_\theta(x) = ax + b$
- Parameters: $\theta = [a, b]^T \in \mathbb{R}^2$
- Total Loss: $g(\theta) = L(f_\theta, \mathcal{T}) = \frac{1}{T} \sum_{t=1}^T (f_\theta(x_t) - y_t)^2$
## Generalizing linear regression

- **Training set:** $\mathcal{T} = \{(x_t, y_t)\}_{t=1}^T$
- **Models:** $f_\theta(x) = ax + b$
- **Parameters:** $\theta = [a, b]^\top \in \mathbb{R}^2$
- **Total Loss:**
  $$g(\theta) = L(f_\theta, \mathcal{T}) = \frac{1}{T} \sum_{t=1}^T (f_\theta(x_t) - y_t)^2$$

$$y = a_0 x + b_0$$
IV. Supervised Learning

Generalizing linear regression

\[ y = a_0 x + b_0 \]

- Training set: \( \mathcal{T} = \{ (x_t, y_t) \}_{t=1}^{T} \)
- Models: \( f_{\theta}(x) = ax + b \)
- Parameters: \( \theta = [a, b]^T \in \mathbb{R}^2 \)
- Total Loss: 
  \[
  g(\theta) = L(f_{\theta}, \mathcal{T}) = \frac{1}{T} \sum_{t=1}^{T} (f_{\theta}(x_t) - y_t)^2
  \]

\[
  g(\theta) = \frac{1}{T} \sum_{t=1}^{T} \left( [x_t, 1]^T \begin{bmatrix} a \\ b \end{bmatrix} - y_t \right)^2 = \frac{1}{T} \| W\theta - y \|_2^2, \quad \nabla_\theta g(\theta_0) = 0 \Rightarrow \theta_0 = (W^T W)^{-1} W^T y
\]
IV. Supervised Learning ▶ Regression

Generalizing linear regression

\[ y = a_0 x + b_0 \]

- Training set: \( \mathcal{T} = \{(x_t, y_t)\}_{t=1}^T \)
- Models: \( f_\theta(x) = ax + b \)
- Parameters: \( \theta = [a, b]^\top \in \mathbb{R}^2 \)
- Total Loss: \( g(\theta) = L(f_\theta, \mathcal{T}) = \frac{1}{T} \sum_{t=1}^T (f_\theta(x_t) - y_t)^2 \)

\[
g(\theta) = \frac{1}{T} \sum_{t=1}^T \left( \begin{bmatrix} x_t, 1 \end{bmatrix}^\top \begin{bmatrix} a \\ b \end{bmatrix} - y_t \right)^2 = \frac{1}{T} \| W_\theta - y \|_2^2, \quad \nabla_\theta g(\theta_0) = 0 \Rightarrow \theta_0 = (W^\top W)^{-1} W^\top y
\]

\[ \rightarrow \text{This generalizes to } y = f_\theta(x) = Ax + b, \quad \theta = (A, b) \in \mathbb{R}^{N \times D} \times \mathbb{R}^N : \]
IV. Supervised Learning

Regression

Generalizing linear regression

\[ y = a_0 x + b_0 \]

- Training set: \( \mathcal{T} = \{(x_t, y_t)\}_{t=1}^T \)
- Models: \( f_\theta(x) = ax + b \)
- Parameters: \( \theta = [a, b]^T \in \mathbb{R}^2 \)
- Total Loss: \( g(\theta) = L(f_\theta, \mathcal{T}) = \frac{1}{T} \sum_{t=1}^T (f_\theta(x_t) - y_t)^2 \)

\[ g(\theta) = \frac{1}{T} \sum_{t=1}^T \left( [x_t, 1] \begin{bmatrix} a \\ b \end{bmatrix} - y_t \right)^2 = \frac{1}{T} \|W\theta - y\|_2^2, \quad \nabla_\theta g(\theta_0) = 0 \Rightarrow \theta_0 = (W^T W)^{-1} W^T y \]

→ This generalizes to \( y = f_\theta(x) = Ax + b, \quad \theta = (A, b) \in \mathbb{R}^{N \times D} \times \mathbb{R}^N \):

\[ g(\theta) = \frac{1}{T} \sum_{t=1}^T \|Ax_t + b - y_t\|_2^2, \]
IV. Supervised Learning

Regression

Generalizing linear regression

\[ y = a_0 x + b_0 \]

- Training set: \( T = \{ (x_t, y_t) \}_{t=1}^T \)
- Models: \( f_\theta(x) = ax + b \)
- Parameters: \( \theta = [a, b]^T \in \mathbb{R}^2 \)
- Total Loss: \( g(\theta) = L(f_\theta, T) = \frac{1}{T} \sum_{t=1}^T (f_\theta(x_t) - y_t)^2 \)

\[
g(\theta) = \frac{1}{T} \sum_{t=1}^T \left( \begin{bmatrix} x_t \ 1 \end{bmatrix}^T \begin{bmatrix} a \\ b \end{bmatrix} - y_t \right)^2 = \frac{1}{T} \| W\theta - y \|_2^2, \quad \nabla_\theta g(\theta_0) = 0 \Rightarrow \theta_0 = (W^T W)^{-1} W^T y
\]

→ This generalizes to \( y = f_\theta(x) = Ax + b, \quad \theta = (A, b) \in \mathbb{R}^{N \times D} \times \mathbb{R}^N \):

\[
g(\theta) = \frac{1}{T} \sum_{t=1}^T \| Ax_t + b - y_t \|_2^2, \quad \nabla_\theta g(\theta_0) = 0 \Rightarrow \begin{bmatrix} \hat{a}_{n,0} \\ b_{n,0} \end{bmatrix} = (W^T W)^{-1} W^T y_n, \quad \forall n
\]

\[
\begin{bmatrix} x_1^T, 1 \\ \vdots \\ x_T^T, 1 \end{bmatrix} \in \mathbb{R}^{T \times (D+1)}
\]
### Generalizing linear regression

- **Training set:** $\mathcal{T} = \{(x_t, y_t)\}_{t=1}^T$
- **Models:** $f_\theta(x) = a_2 x^2 + a_1 x + a_0$
- **Parameters:** $\theta = [a_0, a_1, a_2]^T \in \mathbb{R}^3$
- **Total Loss:** $g(\theta) = L(f_\theta, \mathcal{T}) = \frac{1}{T} \sum_{t=1}^{T} (f_\theta(x_t) - y_t)^2$

→ What about polynomial regression?
Generalizing linear regression

- Training set: $\mathcal{T} = \{(x_t, y_t)\}_{t=1}^T$
- Models: $f_\theta(x) = a_2 x^2 + a_1 x + a_0$
- Parameters: $\theta = [a_0, a_1, a_2]^\top \in \mathbb{R}^3$
- Total Loss: $g(\theta) = L(f_\theta, \mathcal{T}) = \frac{1}{T} \sum_{t=1}^T (f_\theta(x_t) - y_t)^2$

→ What about polynomial regression?
- Convertible to the same problem using the lifting: $y = [a_0, a_1, a_2] \times \begin{bmatrix} 1 \\ x \\ x^2 \end{bmatrix}$
Generalizing linear regression

- Training set: $\mathcal{T} = \{(x_t, y_t)\}_{t=1}^T$
- Models: $f_\theta(x) = a_2x^2 + a_1x + a_0$
- Parameters: $\theta = [a_0, a_1, a_2]^\top \in \mathbb{R}^3$
- Total Loss: $g(\theta) = L(f_\theta, \mathcal{T}) = \frac{1}{T} \sum_{t=1}^T (f_\theta(x_t) - y_t)^2$

→ What about polynomial regression?
- Convertible to the same problem using the lifting: $y = [a_0, a_1, a_2] \times \begin{bmatrix} 1 \\ x \\ x^2 \end{bmatrix}$
- More generally, 2nd degree multivariate polynomial regression ($y \in \mathbb{R}^N$, $x \in \mathbb{R}^D$):

$$y_n = \sum_{i=1}^D \sum_{j=i}^D a_{ij}^{(n)} x_i x_j + \sum_{i=1}^D a_i^{(n)} x_i + a_0^{(n)} \Rightarrow y_n = [a_0^{(n)}, a_1^{(n)}, \ldots, a_{11}^{(n)}, a_{12}^{(n)} \ldots, a_{DD}^{(n)}] \times \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_1^2 \\ \vdots \\ x_D^2 \end{bmatrix}$$
IV. Supervised Learning

Regression

Generalizing linear regression

$y = a_2 x^2 + a_1 x + a_0$?

- Training set: $\mathcal{T} = \{(x_t, y_t)\}_{t=1}^T$
- Models: $f_\theta(x) = a_2 x^2 + a_1 x + a_0$
- Parameters: $\theta = [a_0, a_1, a_2]^T \in \mathbb{R}^3$
- Total Loss: $g(\theta) = L(f_\theta, \mathcal{T}) = \frac{1}{T} \sum_{t=1}^T (f_\theta(x_t) - y_t)^2$

→ What about polynomial regression?
- Convertible to the same problem using the lifting: $y = [a_0, a_1, a_2] \times \begin{bmatrix} 1 \\ x \\ x^2 \end{bmatrix}$
- More generally, 2nd degree multivariate polynomial regression $(y \in \mathbb{R}^N, x \in \mathbb{R}^D)$:

$$y_n = \sum_{i=1}^D \sum_{j=i}^D a_{ij}^{(n)} x_i x_j + \sum_{i=1}^D a_i^{(n)} x_i + a_0^{(n)} \Rightarrow y_n = [a_0^{(n)}, a_1^{(n)}, \ldots, a_{11}^{(n)}, a_{12}^{(n)}, \ldots, a_{DD}^{(n)}] \times \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_1^2 \\ \vdots \\ x_2^2 \end{bmatrix}$$

→ How many parameters?
IV. Supervised Learning

Regression

Generalizing linear regression

\[ y = a_2 x^2 + a_1 x + a_0 ? \]

- Training set: \( \mathcal{T} = \{(x_t, y_t)\}^T_{t=1} \)
- Models: \( f_\theta(x) = a_2 x^2 + a_1 x + a_0 \)
- Parameters: \( \theta = [a_0, a_1, a_2]^T \in \mathbb{R}^3 \)
- Total Loss: \( g(\theta) = L(f_\theta, \mathcal{T}) = \frac{1}{T} \sum_{t=1}^{T} (f_\theta(x_t) - y_t)^2 \)

→ What about polynomial regression?
- Convertible to the same problem using the lifting: \( y = [a_0, a_1, a_2] \times \begin{bmatrix} 1 \\ x \\ x^2 \end{bmatrix} \)
- More generally, 2\text{nd} degree multivariate polynomial regression \( (y \in \mathbb{R}^N, x \in \mathbb{R}^D) \):

\[
y_n = \sum_{i=1}^{D} \sum_{j=i}^{D} a_{ij}^{(n)} x_i x_j + \sum_{i=1}^{D} a_i^{(n)} x_i + a_0^{(n)} \Rightarrow y_n = [a_0^{(n)}, a_1^{(n)}, \ldots, a_{11}^{(n)}, a_{12}^{(n)}, \ldots, a_{DD}^{(n)}] \times \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_1^2 \\ \vdots \\ x_D^2 \end{bmatrix}
\]
→ How many parameters? \( O(ND^2) \)
Generalizing linear regression

- $K^{\text{th}}$ degree multivariate polynomial regression:

$$y_n = \sum_{i_1=1}^{D} \sum_{i_2=i_1}^{D} \cdots \sum_{i_K=i_{K-1}}^{D} a_{i_1i_2\ldots i_K}^{(n)} x_{i_1} x_{i_2} \cdots x_{i_K} + \cdots + \sum_{i_1=1}^{D} \sum_{i_2=i_1}^{D} a_{i_1i_2}^{(n)} x_{i_1} x_{i_2} + \sum_{i_1=1}^{D} a_{i_1}^{(n)} x_{i_1} + a_0^{(n)}$$

$\rightarrow$ How many parameters? $\mathcal{O}(ND^K)$
IV. Supervised Learning ▶ Regression

Generalizing linear regression

- $K^{th}$ degree multivariate polynomial regression:

$$y_n = \sum_{i_1=1}^{D} \sum_{i_2=i_1}^{D} \cdots \sum_{i_K=i_{K-1}}^{D} a_{i_1i_2\ldots i_K}^{(n)} x_{i_1} x_{i_2} \cdots x_{i_K} + \cdots + \sum_{i_1=1}^{D} \sum_{i_2=i_1}^{D} a_{i_1i_2}^{(n)} x_{i_1} x_{i_2} + \sum_{i_1=1}^{D} a_{i_1}^{(n)} x_{i_1} + a_0^{(n)}$$

→ How many parameters? $O(ND^K)$

- In contrast:

![Diagram of a neural network with $K$ layers](image)

→ How many parameters?
Generalizing linear regression

- $K^{th}$ degree multivariate polynomial regression:

$$y_n = \sum_{i_1=1}^{D} \sum_{i_2=i_1}^{D} \cdots \sum_{i_K=i_{K-1}}^{D} a_{i_1i_2\ldots i_K}^{(n)} x_{i_1} x_{i_2} \cdots x_{i_K} + \cdots + \sum_{i_1=1}^{D} \sum_{i_2=i_1}^{D} a_{i_1i_2}^{(n)} x_{i_1} x_{i_2} + \sum_{i_1=1}^{D} a_{i_1}^{(n)} x_{i_1} + a_0^{(n)}$$

→How many parameters? $\mathcal{O}(ND^K)$

- In contrast:

$K$ layers

$\begin{align*}
\mathbf{x} & \in \mathbb{R}^D \\
& \vdots \\
& N \quad N \quad N \quad N \\
\mathbf{y} & \in \mathbb{R}^N
\end{align*}$

→How many parameters? $\mathcal{O}(DN + (K - 1)N^2)$
How to choose the loss?

IV. Supervised Learning

► How to Choose the Loss?

How to choose the loss?

$W^1, b^1$ $W^2, b^2$ $W^K, b^K$

$\sigma$

$\mathbf{a}$

$x^1$ $x^2$ $x^K$

$\mathbf{x}_t$

$y_t$

$y$ (age, height) $\in \mathbb{R}^2$

$\mathbf{l}$
How to choose the loss?

A general principled approach is to use the network to model $p(y|x)$.
IV. Supervised Learning

How to choose the loss?

A general principled approach is to use the network to model $p(y|x)$.

1) Choose a **simple** family of **parameterized** probabilistic distributions over the domain of $y$ and $x^K$, i.e., $\mathcal{P} = \{\tilde{p}_\lambda(y)\}_{\lambda \in \Lambda}$.
How to choose the loss?

A general principled approach is to use the network to model $p(y|x)$.

1) Choose a **simple** family of parameterized probabilistic distributions over the domain of $y$ and $x^K$, i.e., $\mathcal{P} = \{\tilde{p}_\lambda(y)\}_{\lambda \in \Lambda}$.

Ex: $\tilde{p}_\mu(y) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{\|y - \mu\|_2^2}{2}\right)$, $\tilde{p}_b(y) = \begin{cases} b \in [0, 1] & \text{for } y = 1 \\ 1 - b & \text{for } y = 0 \end{cases}$

(Gaussian) (Bernouilli)
How to choose the loss?

A general principled approach is to use the network to model $p(y|x)$.

1) Choose a **simple** family of **parameterized** probabilistic distributions over the domain of $y$ and $x^K$, i.e., $\mathcal{P} = \{\tilde{p}_\lambda(y)\}_{\lambda \in \Lambda}$.

   \[
   \text{Ex: } \tilde{p}_\mu(y) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{\|y - \mu\|_2^2}{2}\right), \quad \tilde{p}_b(y) = \begin{cases} 
   b \in [0, 1] & \text{for } y = 1 \\
   1 - b & \text{for } y = 0
   \end{cases}
   \]

   (Gaussian) (Bernoulli)

2) We then model the conditional probability as: $p_\theta(y|x) = \tilde{p}_{\lambda = \text{dnn}_\theta(x)}(y)$
How to choose the loss?

A general principled approach is to use the network to model \( p(y|x) \).

1) Choose a **simple** family of **parameterized** probabilistic distributions over the domain of \( y \) and \( x^K \), i.e., \( \mathcal{P} = \{ \tilde{p}_\lambda(y) \}_{\lambda \in \Lambda} \).

   \[
   \text{Ex: } \tilde{p}_\mu(y) = \frac{1}{\sqrt{2\pi}} \exp \left( -\frac{\|y - \mu\|^2_2}{2} \right), \quad \tilde{p}_b(y) = \begin{cases} 
   b \in [0, 1] & \text{for } y = 1 \\
   1 - b & \text{for } y = 0
   \end{cases} 
   \]

   (Gaussian) (Bernouilli)

2) We then model the conditional probability as: \( p_\theta(y|x) = \tilde{p}_{\text{dnn}_\theta(x)}(y) \)

3) We want to find \( \theta \) that maximizes the **likelihood** over the training set:

   \[
   \hat{\theta} = \arg\max_\theta \prod_{t=1}^T p_\theta(y_t|x_t)
   \]
How to choose the loss?

A general principled approach is to use the network to model $p(y|x)$.

1) Choose a **simple** family of **parameterized** probabilistic distributions over the domain of $y$ and $x^K$, i.e., $\mathcal{P} = \{\tilde{p}_\lambda(y)\}_{\lambda \in \Lambda}$.

   $$Ex: \quad \tilde{p}_\mu(y) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{\|y - \mu\|_2^2}{2}\right), \quad \tilde{p}_b(y) = \begin{cases} b \in [0, 1] & \text{for } y = 1 \\ 1 - b & \text{for } y = 0 \end{cases}$$

   (Gaussian)  
   (Bernoulli)

2) We then model the conditional probability as: $p_\theta(y|x) = \tilde{p}_\lambda = \text{dnn}_\theta(x)(y)$

3) We want to find $\theta$ that maximizes the **likelihood** over the training set:

   $$\hat{\theta} = \arg\max_\theta \prod_{t=1}^T p_\theta(y_t|x_t)$$

**Note:** we assume the training set examples are **independent**.
IV. Supervised Learning

How to choose the loss?

4) Usually, we define the **total loss** as the **negative log-likelihood**:

\[
L(dnn_\theta, \mathcal{T}) = -\log \prod_{t=1}^{T} p_\theta(y_t|x_t)
\]
How to choose the loss?

4) Usually, we define the **total loss** as the **negative log-likelihood**:

\[
L(dnn_\theta, \mathcal{T}) = -\log \prod_{t=1}^{T} p_\theta(y_t|x_t) = \sum_{t=1}^{T} -\log p_\theta(y_t|x_t)
\]
IV. Supervised Learning

How to choose the loss?

4) Usually, we define the **total loss** as the **negative log-likelihood**:

\[
L(\text{dnn}_\theta, \mathcal{T}) = - \log \prod_{t=1}^{T} p_\theta(y_t | x_t) = \sum_{t=1}^{T} - \log p_\theta(y_t | x_t)
\]

\[
= \sum_{t=1}^{T} - \log \tilde{p}_{\text{dnn}_\theta}(x_t)(y_t)
\]
How to choose the loss?

4) Usually, we define the total loss as the negative log-likelihood:

\[
L(dnn_{\theta}, \mathcal{T}) = -\log \prod_{t=1}^{T} p_{\theta}(y_t|x_t) = \sum_{t=1}^{T} -\log p_{\theta}(y_t|x_t)
\]

\[
= \sum_{t=1}^{T} -\log \tilde{p}_{dnn_{\theta}}(x_t)(y_t) = \sum_{t=1}^{T} -\log \tilde{p}_{x^K}(y_t)
\]
IV. Supervised Learning

How to choose the loss?

4) Usually, we define the total loss as the negative log-likelihood:

\[
L(dnn_\theta, T) = -\log \prod_{t=1}^{T} p_\theta(y_t|x_t) = \sum_{t=1}^{T} - \log p_\theta(y_t|x_t)
\]

\[
= \sum_{t=1}^{T} - \log \tilde{p}_{dnn_\theta}(x_t)(y_t) = \sum_{t=1}^{T} - \log \tilde{p}_{x^K_t}(y_t)
\]

Example with the Gaussian distribution:

\[
\ell(x^K_t, y_t) = - \log \tilde{p}_{x^K_t}(y_t) = - \log \frac{1}{\sqrt{2\pi}} \exp \left( - \frac{||x^K_t - y_t||^2}{2} \right)
\]
How to choose the loss?

4) Usually, we define the **total loss** as the **negative log-likelihood**:

\[ L(dnn_\theta, T) = -\log \prod_{t=1}^{T} p_\theta(y_t|x_t) = -\log p_\theta(y_t|x_t) \]

\[ = \sum_{t=1}^{T} -\log \tilde{p}_{dnn_\theta}(x_t)(y_t) = \sum_{t=1}^{T} -\log \tilde{p}_{x^K_t}(y_t) \]

Example with the Gaussian distribution:

\[ \ell(x^K_t, y_t) = -\log \tilde{p}_{x^K_t}(y_t) = -\log \frac{1}{\sqrt{2\pi}} \exp \left( -\frac{\|x^K_t - y_t\|^2}{2} \right) = \frac{1}{2} \|x^K_t - y_t\|^2 \]

→ We recover the **L2 loss**!
How to choose the loss?

4) Usually, we define the **total loss** as the **negative log-likelihood**:

\[
L(dnn_\theta, T) = -\log \prod_{t=1}^{T} p_\theta(y_t | x_t) = \sum_{t=1}^{T} -\log p_\theta(y_t | x_t) \\
= \sum_{t=1}^{T} -\log \tilde{p}_{dnn_\theta}(x_t)(y_t) = \sum_{t=1}^{T} -\log \tilde{p}_{x^K_t}(y_t)
\]

Example with the Gaussian distribution:

\[
\ell(x^K_t, y_t) = -\log \tilde{p}_{x^K_t}(y_t) = -\log \frac{1}{\sqrt{2\pi}} \exp \left( -\frac{\|x^K_t - y_t\|^2}{2} \right) = \frac{1}{2} \|x^K_t - y_t\|^2
\]

→ We recover the **L2 loss**!

Using the L2 loss is equivalent to assuming the network will make i.i.d Gaussian errors.
How to choose the loss?

The approach is very general and can be used with a variety of parameterized probability distributions.
How to choose the loss?

The approach is very general and can be used with a variety of parameterized probability distributions.

- For $y \geq 0$ we can use an exponential distribution $\tilde{p}_\lambda(y) = \lambda \exp(-\lambda y)$.
How to choose the loss?

The approach is very general and can be used with a variety of parameterized probability distributions.

- For $y \geq 0$ we can use an exponential distribution $\tilde{p}_\lambda(y) = \lambda \exp(-\lambda y)$

- We can use this approach to not only estimate the mean but also the variance ($\approx$uncertainty) of the network output:

$$\tilde{p}_{\mu,\sigma^2}(y) = \frac{1}{\sqrt{2\pi\sigma^2N}} \exp\left(-\frac{\|y - \mu\|^2_2}{2\sigma^2}\right), \quad x^K \equiv [\mu, \sigma^2]$$
Detection

Example: Captcha \( x_t \in \mathbb{R}^D, \ y_t \in \{0, 1\} \)
Detection

Example: Captcha \( x_t \in \mathbb{R}^D, \ y_t \in \{0, 1\} \)

- Let’s use the same principle to design our loss.
Detection

Example: Captcha \( \mathbf{x}_t \in \mathbb{R}^D, \ y_t \in \{0, 1\} \)

- Let’s use the same principle to design our loss.
- We can use a Bernouilli distribution:

\[
\tilde{p}_b(y) = \begin{cases} 
    b & \text{for } y = 1 \\
    1 - b & \text{for } y = 0 
\end{cases}
\]
Detection

Example: Captcha $\mathbf{x}_t \in \mathbb{R}^D$, $y_t \in \{0, 1\}$

- Let’s use the same principle to design our loss.
- We can use a Bernouilli distribution:

$$\tilde{p}_b(y) = \begin{cases} b & \text{for } y = 1 \\ 1 - b & \text{for } y = 0 \end{cases}$$

- Note that $b \in [0, 1]$, hence we need to constrain the output of the network in this interval => we use a sigmoid function at the output:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$
IV. Supervised Learning

Detection

Example: Captcha $x_t \in \mathbb{R}^D$, $y_t \in \{0, 1\}$

- Let’s use the same principle to design our loss.
- We can use a Bernoulli distribution:
  \[
  \tilde{p}_b(y) = \begin{cases} 
  b \in [0, 1] & \text{for } y = 1 \\
  1 - b & \text{for } y = 0
  \end{cases}
  \]

- Note that $b \in [0, 1]$, hence we need to constrain the output of the network in this interval $\Rightarrow$ we use a sigmoid function at the output:

- Using the maximum likelihood approach with this distribution, we obtain the following loss:
  \[
  \ell(x^K_t, y_t) = -\log \tilde{p}_{x^K_t}(y_t) = -y_t \log x^K_t - (1 - y_t) \log (1 - x^K_t),
  \]
  \[
  = \text{the Binary Cross-Entropy}.
  \]
IV. Supervised Learning  ▶ Detection & Classification

Classification

• This generalizes to **multi-class classification**

\[ \mathbf{x}_t \in \mathbb{R}^D, \; y_t \in \{1, 2, \ldots, N\} \]

Ex: ImageNet (1000 classes)
Classification

- This generalizes to **multi-class classification**

\[ x_t \in \mathbb{R}^D, \ y_t \in \{1, 2, \ldots, N\} \]

- It is convenient to represent the output as a **“one-hot”** vector:

\[ y_t = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \rightarrow \text{index of the class} \]

Ex: ImageNet (1000 classes)
Classification

- This generalizes to multi-class classification

\[ x_t \in \mathbb{R}^D, \ y_t \in \{1, 2, \ldots, N\} \]

- It is convenient to represent the output as a "one-hot" vector:

\[
\begin{bmatrix}
0 \\
0 \\
\vdots \\
0 \\
1 \\
0 \\
\vdots \\
0
\end{bmatrix}
\rightarrow \text{index of the class}
\]

- We use a categorical distribution:

\[
\tilde{p}_b(y) = \begin{cases} 
  b_1 \in [0, 1] & \text{for } y_1 = 1 \\
  b_2 \in [0, 1] & \text{for } y_2 = 1 \\
  \vdots \\
  b_N \in [0, 1] & \text{for } y_N = 1 
\end{cases}, \quad \text{with } \sum_n b_n = 1
\]

Ex: ImageNet (1000 classes)
Classification

- This generalizes to **multi-class classification**
  \[ x_t \in \mathbb{R}^D, \ y_t \in \{1, 2, \ldots, N\} \]
- It is convenient to represent the output as a **“one-hot”** vector:
  \[
  y_t = \begin{bmatrix}
  0 \\
  \vdots \\
  0 \\
  1 \\
  0 \\
  \vdots \\
  0
  \end{bmatrix} \rightarrow \text{index of the class}
  \]
- We use a **categorical distribution**:
  \[
  \tilde{p}_b(y) = \begin{cases}
  b_1 \in [0, 1] & \text{for } y_1 = 1 \\
  b_2 \in [0, 1] & \text{for } y_2 = 1 \\
  \vdots & \\
  b_N \in [0, 1] & \text{for } y_N = 1 \\
  \end{cases}, \quad \text{with } \sum_n b_n = 1
  \]
Classification

- The **Soft-Max** activation function:

\[
x^K = \sigma(-a^K) = \frac{1}{\sum_{n'=1}^{N} \exp(a^K_{n'})} \begin{bmatrix}
\exp(a^K_1) \\
\exp(a^K_2) \\
\vdots \\
\exp(a^K_N)
\end{bmatrix}
\]
IV. Supervised Learning

Classification

• The *Soft-Max* activation function:

\[
x^K = \sigma(-a^K) = \frac{1}{\sum_{n'=1}^{N} \exp(a_{n'}^K)} \begin{bmatrix} \exp(a_1^K) \\ \exp(a_2^K) \\ \vdots \\ \exp(a_N^K) \end{bmatrix}
\]

• Can be viewed as a generalization of the sigmoid
IV. Supervised Learning

Classification

- The **Soft-Max** activation function:

\[
x^K = \sigma(-a^K) = \frac{1}{\sum_{n'=1}^{N} \exp(a^K_{n'})} \begin{bmatrix}
\exp(a^K_1) \\
\exp(a^K_2) \\
\vdots \\
\exp(a^K_N)
\end{bmatrix}
\]

- Can be viewed as a generalization of the sigmoid
- Approximates the **max** function, in the sense that if one value is much larger than the others, we obtain a 1-hot vector at that value
IV. Supervised Learning

Classification

- The **Soft-Max** activation function:
  \[
  x^K = \sigma(-a^K) = \frac{1}{\sum_{n'=1}^{N} \exp(a^K_{n'})} \begin{bmatrix}
  \exp(a^K_1) \\
  \exp(a^K_2) \\
  \vdots \\
  \exp(a^K_N)
  \end{bmatrix}
  \]

- Can be viewed as a generalization of the sigmoid

- Approximates the **max** function, in the sense that if one value is much larger than the others, we obtain a 1-hot vector at that value

- Using the maximum likelihood approach with a categorical distribution yields the (generalized) **cross entropy loss**:

  \[
  \ell(x^K_t, y_t) = -\log \tilde{p}_{x^K_t}(y_t) = \sum_{n=1}^{N} -y_{t,n} \log x^K_{t,n}
  \]
Classification

- The **Soft-Max** activation function:
  \[
  x^K = \sigma(-a^K) = \frac{1}{\sum_{n'=1}^{N} \exp(a^K_{n'})} \begin{bmatrix}
  \exp(a^K_1) \\
  \exp(a^K_2) \\
  \vdots \\
  \exp(a^K_N)
\end{bmatrix}
  \]

- Can be viewed as a generalization of the sigmoid

- Approximates the **max** function, in the sense that if one value is much larger than the others, we obtain a 1-hot vector at that value

- Using the maximum likelihood approach with a categorical distribution yields the (generalized) **cross entropy loss**:
  \[
  \ell(x^K_t, y_t) = -\log \tilde{p}_{x^K_t}(y_t) = \sum_{n=1}^{N} -y_{t,n} \log x^K_{t,n}
  \]

**Multi-Label Classification**
IV. Supervised Learning

Classification

- The **Soft-Max** activation function:
  
  \[ x^K = \sigma(-a^K) = \frac{1}{\sum_{n'=1}^{N} \exp(a_{n'}^K)} \begin{bmatrix} \exp(a_1^K) \\ \exp(a_2^K) \\ \vdots \\ \exp(a_N^K) \end{bmatrix} \]

- Can be viewed as a generalization of the sigmoid

- Approximates the **max** function, in the sense that if one value is much larger than the others, we obtain a 1-hot vector at that value

- Using the maximum likelihood approach with a categorical distribution yields the (generalized) **cross entropy loss**:
  
  \[ \ell(x^K_t, y_t) = -\log \tilde{p}_x^K(y_t) = \sum_{n=1}^{N} y_{t,n} \log x^K_{t,n} \]

Multi-Label Classification

- Can be done by statistically aggregating **multiple binary detectors**
Classification

• The **Soft-Max** activation function:

\[
x^K = \sigma(-a^K) = \frac{1}{\sum_{n'=1}^{N} \exp(a^K_{n'})} \begin{bmatrix} \exp(a^K_1) \\ \exp(a^K_2) \\ \vdots \\ \exp(a^K_N) \end{bmatrix}
\]

• Can be viewed as a generalization of the sigmoid

• Approximates the **max** function, in the sense that if one value is much larger than the others, we obtain a 1-hot vector at that value

• Using the maximum likelihood approach with a categorical distribution yields the (generalized) **cross entropy loss**:

\[
\ell(x^K_t, y_t) = -\log \tilde{p}_t x^K_t(y_t) = \sum_{n=1}^{N} -y_{t,n} \log x^K_{t,n}
\]

**Multi-Label Classification**

• Can be done by statistically aggregating **multiple binary detectors**

• Falls in the category of **ensemble methods**
**Scenario:**

- Imagine we have a training dataset $\mathcal{T}$ containing 10,000 images with labels (supervised learning)
Scenario:

- Imagine we have a training dataset \( \mathcal{T} \) containing 10,000 images with labels (supervised learning)
- Using the backpropagation algorithm, we train a DNN to perform the multi-class classification task
Scenario:

- Imagine we have a training dataset $\mathcal{T}$ containing 10,000 images with labels (supervised learning).

- Using the backpropagation algorithm, we train a DNN to perform the multi-class classification task.

- We get nearly **perfect results** on these 10,000 images, e.g., 99.9% of correct classification.
Scenario:

- Imagine we have a training dataset $\mathcal{T}$ containing 10,000 images with labels (supervised learning)

- Using the backpropagation algorithm, we train a DNN to perform the multi-class classification task

- We get nearly **perfect results** on these 10,000 images, e.g., 99.9% of correct classification

- However, when we run the DNN on new images, the results are **awful**, i.e., close to random.
IV. Supervised Learning

Scenario:

• Imagine we have a training dataset \( \mathcal{T} \) containing 10,000 images with labels (supervised learning)

• Using the backpropagation algorithm, we train a DNN to perform the multi-class classification task

• We get nearly **perfect results** on these 10,000 images, e.g., 99.9% of correct classification

• However, when we run the DNN on new images, the results are **awful**, i.e., close to random.

*What’s going on?*
Overfitting

• Our algorithm is guilty of overfitting \((\text{sur-apprentissage})\)
Overfitting

• Our algorithm is guilty of overfitting (*sur-apprentissage*)

• Instead of learning general features to classify the images, it learned by heart all the images in our training dataset!
Overfitting

- Our algorithm is guilty of **overfitting** (*sur-apprentissage*)

- Instead of learning general features to classify the images, it **learned by heart** all the images in our training dataset!

- Remember that we often have **millions** or **billions** or parameters in a deep model. Hence, it has the **capacity** to **store/encode** large amount of data
Overfitting

- Our algorithm is guilty of **overfitting** (*sur-apprentissage*)

- Instead of learning general features to classify the images, it **learned by heart** all the images in our training dataset!

- Remember that we often have **millions** or **billions** or parameters in a deep model. Hence, it has the **capacity** to **store/encode** large amount of data

- This may even happen for models of relatively small capacity, if the **amount of training data** is insufficient.
Overfitting

- Ex: polynomial regression
Overfitting

- Ex: polynomial regression

- Ex: binary classification
Overfitting

• In supervised learning, we are *not only interested* in a model that works perfectly on our training set. *We already have the answers anyway*, by definition of a (supervised) training set!
Overfitting

- In supervised learning, we are **not only interested** in a model that works perfectly on our training set. **We already have the answers anyway**, by definition of a (supervised) training set!
- We want a model that **generalizes** to **unseen** data.
Overfitting

- In supervised learning, we are **not only interested** in a model that works perfectly on our training set. We already have the answers **anyway**, by definition of a (supervised) training set!

- We want a model that **generalizes** to unseen data

- Solution: We split our dataset in **3**:

  ![Dataset Diagram]

  - TRAINING
  - VALIDATION
  - TEST
Overfitting

• In supervised learning, we are not only interested in a model that works perfectly on our training set. We already have the answers anyway, by definition of a (supervised) training set!

• We want a model that generalizes to unseen data

• Solution: We split our dataset in 3:

  Dataset
  
  | TRAINING | VALIDATION | TEST |

• These 3 subsets must be perfectly disjoint and all representative of the data.
Overfitting

- In supervised learning, we are not only interested in a model that works perfectly on our training set. **We already have the answers anyway,** by definition of a (supervised) training set!

- We want a model that **generalizes** to **unseen** data

- Solution: We split our dataset in **3:**
  - **TRAINING**
  - **VALIDATION**
  - **TEST**

- These 3 subsets must be **perfectly disjoint** and all **representative** of the data.

- To achieve this, the split is done **at random.**
IV. Supervised Learning

Overfitting

Dataset

TRAINING

VALIDATION

TEST
Overfitting

- This separation is **absolutely essential** for any supervised machine learning algorithm to reliably work.
IV. Supervised Learning

Overfitting

- This separation is **absolutely essential** for any supervised machine learning algorithm to reliably work.
- The model parameters are only optimized over the **training set**.

![Dataset](image)
Overfitting

- This separation is **absolutely essential** for any supervised machine learning algorithm to reliably work.
- The model parameters are only optimized over the **training set**.
- We only use the validation set to:
  - At each training step, verify that the model is making progress on that set (possibly using another performance measure than the loss) ➞ If not: we **stop**.
  - Tune hyperparameters (e.g. gradient steps), compare different families of models.
IV. Supervised Learning

Overfitting

Dataset

<table>
<thead>
<tr>
<th>TRAINING</th>
<th>VALIDATION</th>
<th>TEST</th>
</tr>
</thead>
</table>

• This separation is **absolutely essential** for any supervised machine learning algorithm to reliably work.

• The model parameters are only optimized over the **training set**.

• We only use the validation set to:
  • At each training step, verify that the model is making progress on that set (possibly using another performance measure than the loss) => If not: we **stop**.
  • Tune hyperparameters (e.g. gradient steps), compare different families of models.

• Looking at the test set if **forbidden** in any of those steps (**inverse crime**).
Overfitting

- We can detect overfitting by tracking the total loss over the training iterations / epochs:

**GOOD.** The validation loss is only slightly less good than the training one and does not increase

**NOT GOOD.** The validation loss increases: we are starting to learn by heart the training set!
Some vocabulary

- **Capacity**: flexibility of a model. It often (but not necessarily!) correlates with the number of parameters of the model.

- **Hyper-parameter**: a parameter of a model that is not trained (specified before training).

- **Model selection**: process of choosing the best hyperparameters on the validation set.

- **Underfitting**: state of model which could improve generalization with more training or more capacity.

- **Overfitting**: state of model which could improve generalization with less training or less capacity.
IV. Supervised Learning

► Over and Underfitting

Overfitting vs. Underfitting

- Training vs. Validation

underfitting  overfitting

training time or capacity
IV. Supervised Learning

► Over and Underfitting

**Quizz**

- If capacity increases:
  - training error will ?
  - validation error will ?

- If training time increases:
  - training error will ?
  - validation error will ?

- If training set size increases:
  - generalization error will ?
  - difference between the training and generalization error will ?
IV. Supervised Learning

Quizz

- If capacity increases:
  - training error will ?
  - validation error will ?
  - decrease

- If training time increases:
  - training error will ?
  - validation error will ?

- If training set size increases:
  - generalization error will ?
  - difference between the training and generalization error will ?
IV. Supervised Learning

Quizz

- If capacity increases:
  - training error will ? decrease
  - validation error will ? decrease or increase

- If training time increases:
  - training error will ?
  - validation error will ?

- If training set size increases:
  - generalization error will ?
  - difference between the training and generalization error will ?
IV. Supervised Learning

Quizz

• If capacity increases:
  • training error will ? decrease
  • validation error will ? decrease or increase

• If training time increases:
  • training error will ? decrease
  • validation error will ?

• If training set size increases:
  • generalization error will ?
  • difference between the training and generalization error will ?
IV. Supervised Learning

► Over and Underfitting

Quizz

- If capacity increases:
  - training error will ? decrease
  - validation error will ? decrease or increase

- If training time increases:
  - training error will ? decrease
  - validation error will ? decrease or increase

- If training set size increases:
  - generalization error will ?
  - difference between the training and generalization error will ?
IV. Supervised Learning

Over and Underfitting

Quizz

- If capacity increases:
  - training error will ? decrease
  - validation error will ? decrease or increase

- If training time increases:
  - training error will ? decrease
  - validation error will ? decrease or increase

- If training set size increases:
  - generalization error will ? decrease (or stay the same)
  - difference between the training and generalization error will ?
IV. Supervised Learning

Over and Underfitting

Quizz

- If capacity increases:
  - training error will ? decrease
  - validation error will ? decrease or increase

- If training time increases:
  - training error will ? decrease
  - validation error will ? decrease or increase

- If training set size increases:
  - generalization error will ? decrease (or stay the same)
  - difference between the training and generalization error will ? decrease
Techniques to reduce overfitting

1) Regularization
Techniques to reduce overfitting

1) Regularization

- We add to the total loss a term that depends directly on the parameters of the neural network:

\[
L(dnn_\theta, T) = \frac{1}{T} \sum_{t=1}^{T} \ell(dnn_\theta(x_t), y_t) + \lambda \mathcal{R}(\theta)
\]
IV. Supervised Learning

Techniques to reduce overfitting

1) Regularization

- We add to the total loss a term that depends directly on the parameters of the neural network:

\[
L(dnn_\theta, \mathcal{T}) = \frac{1}{T} \sum_{t=1}^{T} \ell(dnn_\theta(x_t), y_t) + \lambda R(\theta)
\]

- For example, we could add the **L2 norm** of the coefficients in the **weight matrices**, to avoid that they become very large (a common clue of overfitting)
Techniques to reduce overfitting

2) Dropout

- Idea: “cripple” the neural network by removing hidden units stochastically
IV. Supervised Learning  ➤ Over and Underfitting

Techniques to reduce overfitting

2) Dropout

- Idea: “cripple” the neural network by removing hidden units stochastically
- Each hidden unit is set to 0 with a certain probability at each gradient step
Techniques to reduce overfitting

2) Dropout

- Idea: “cripple” the neural network by removing hidden units stochastically
- Each hidden unit is set to 0 with a certain probability at each gradient step
- Hidden units cannot co-adapt to other units
IV. Supervised Learning

Techniques to reduce overfitting

2) Dropout

- Idea: “cripple” the neural network by removing hidden units stochastically
- Each hidden unit is set to 0 with a certain probability at each gradient step
- Hidden units cannot co-adapt to other units
- Hidden units must be more generally useful
IV. Supervised Learning

Techniques to reduce overfitting

2) Dropout

- Idea: “cripple” the neural network by removing hidden units stochastically.
- Each hidden unit is set to 0 with a certain probability at each gradient step.
- Hidden units cannot co-adapt to other units.
- Hidden units must be more generally useful.
- Dropout probability typically between 0.2 and 0.5.
Techniques to reduce overfitting

2) Dropout

- Idea: “cripple” the neural network by removing hidden units stochastically
- Each hidden unit is set to 0 with a certain probability at each gradient step
- Hidden units cannot co-adapt to other units
- Hidden units must be more generally useful
- Dropout probability typically between 0.2 and 0.5.
- At test time, replace the masks by their expectation (e.g., constant vector 0.5 if dropout probability is 0.5).
Techniques to reduce overfitting

2) Dropout

- Idea: “cripple” the neural network by removing hidden units stochastically.
- Each hidden unit is set to 0 with a certain probability at each gradient step.
- Hidden units cannot co-adapt to other units.
- Hidden units must be more generally useful.
- Dropout probability typically between 0.2 and 0.5.
- At test time, replace the masks by their expectation (e.g., constant vector 0.5 if dropout probability is 0.5).
- Can be viewed as averaging an exponential number of networks.
IV. Supervised Learning ▶ Over and Underfitting

Techniques to reduce overfitting

2) Dropout
Techniques to reduce overfitting

3) Data Augmentation
Techniques to reduce overfitting

3) Data Augmentation

- Increase the dataset size by applying **transformations** to the input examples that does **not** affect the output (or affect it in a predictable way)
Techniques to reduce overfitting

3) Data Augmentation

- Increase the dataset size by applying transformations to the input examples that does not affect the output (or affect it in a predictable way)

- Examples:
  - Crop an image, flip it, modify its brightness. Replace words by synonyms in sentences
  - Add noise to input signals, degrade their quality, remove imperceptible parts
Techniques to reduce overfitting

3) Data Augmentation

- Increase the dataset size by applying transformations to the input examples that does not affect the output (or affect it in a predictable way)

- Examples:
  - Crop an image, flip it, modify its brightness. Replace words by synonyms in sentences
  - Add noise to input signals, degrade their quality, remove imperceptible parts

- One may as well augment the data by using simulators: e.g. photorealistic 3D scenes, simulated acoustic scenes… or other generative machine learning models.
Techniques to reduce overfitting

3) Data Augmentation

- Increase the dataset size by applying transformations to the input examples that does not affect the output (or affect it in a predictable way)

- Examples:
  - Crop an image, flip it, modify its brightness. Replace words by synonyms in sentences
  - Add noise to input signals, degrade their quality, remove imperceptible parts

- One may as well augment the data by using simulators: e.g. photorealistic 3D scenes, simulated acoustic scenes….or other generative machine learning models.

- Data augmentation is often key for a ML method to work
IV. Supervised Learning

Techniques to reduce overfitting

3) Data Augmentation