OUTLINE

1. ~~Introduction~~
2. ~~Background~~
3. **Fitting a Model**
   - How to minimize a function?
   - Backpropagation
   - Improved Gradient Descent
   - The PyTorch Framework
4. Supervised Learning
5. Unsupervised Learning
6. Fantastic DNNs: How to choose them, how to train them

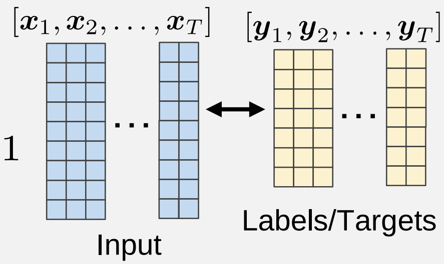
OUTLINE

OUTLINE

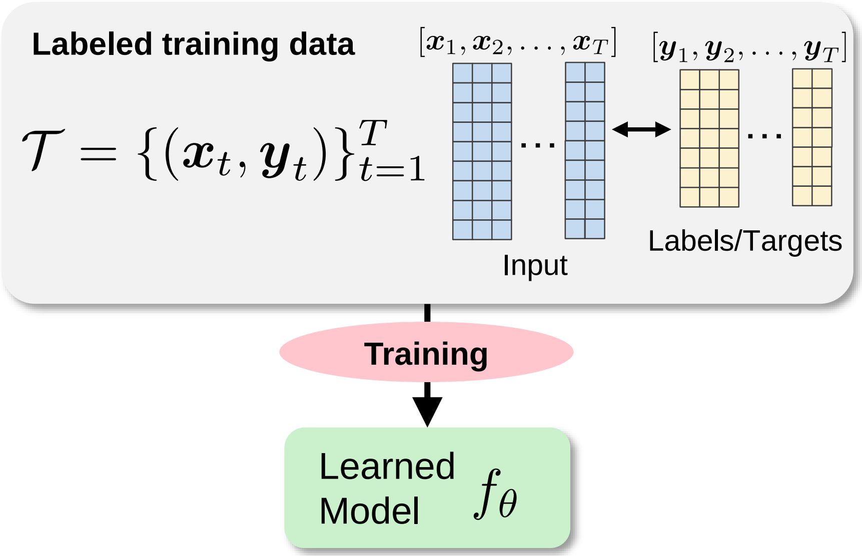
Supervised Learning

Labeled training data

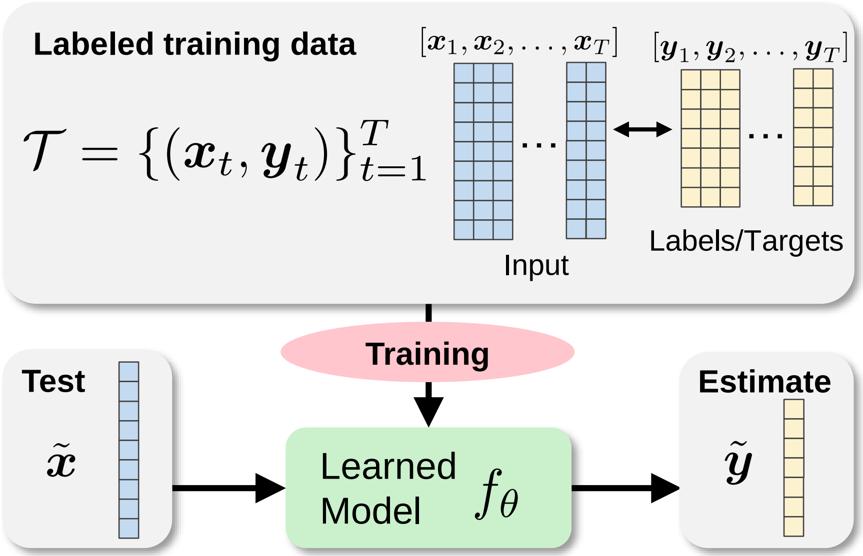
$$\mathcal{T} = \{(\boldsymbol{x}_t, \boldsymbol{y}_t)\}_{t=1}^{T}$$



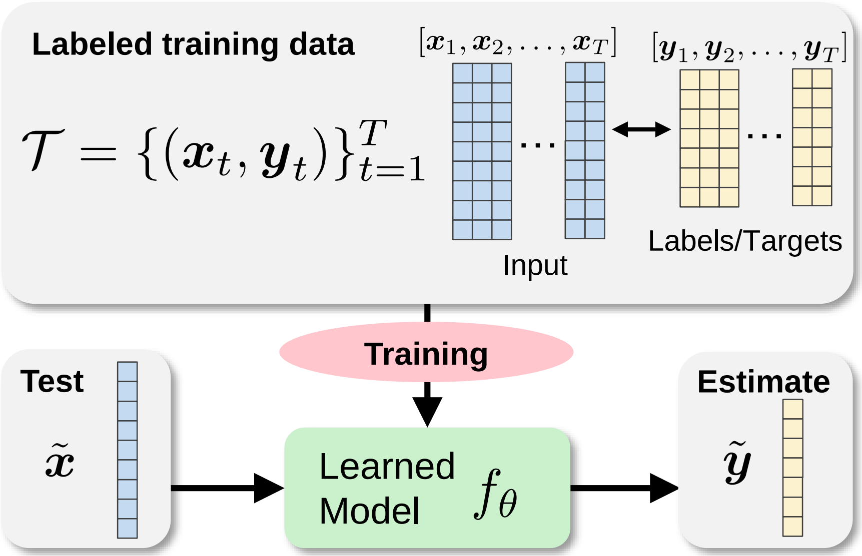
Supervised Learning



Supervised Learning



Supervised Learning



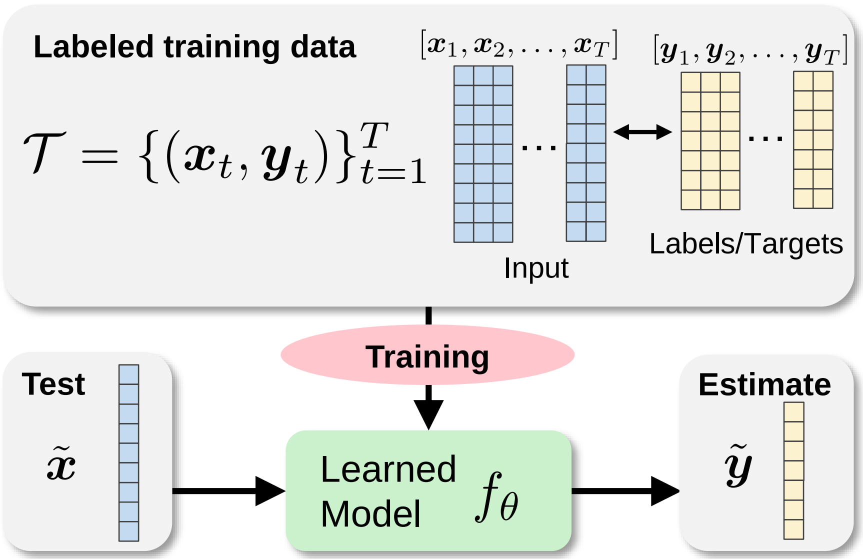
1. Discrete case: («one-hot»)
- Classification

Ex. application: *dog breed*

$\boldsymbol{y} =$

13: German Shepherd

Supervised Learning



1. Discrete case: («one-hot»)
- Classification

Ex. application: *dog breed*

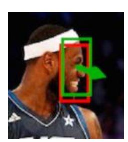
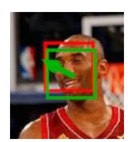
$\boldsymbol{y} =$ 1


13: German Shepherd

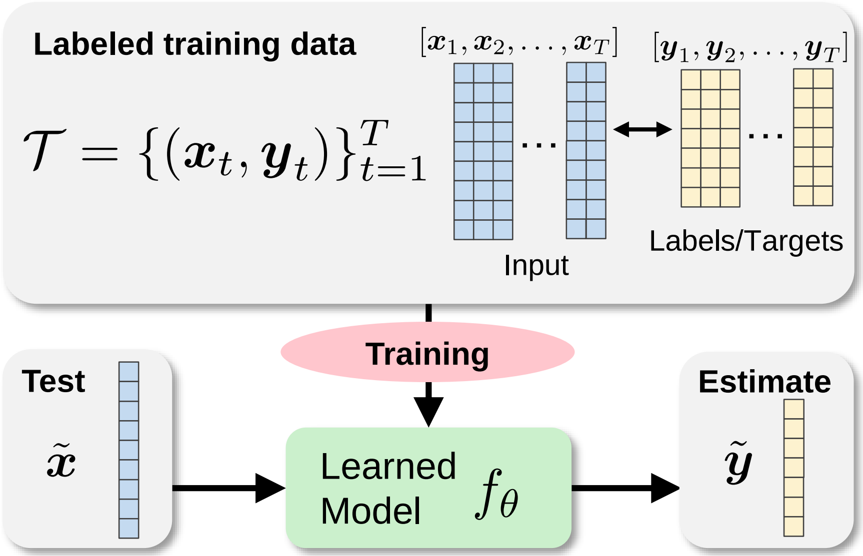
2. Continuous case:
- Regression

Ex. application: *head pose*

$\boldsymbol{y} =$

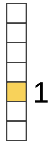



Supervised Learning



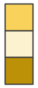
1. Discrete case: («one-hot»)
- Classification

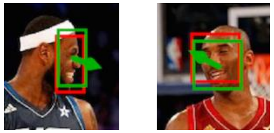
Ex. application: *dog breed*

$\boldsymbol{y} =$ 

2. Continuous case:
- Regression

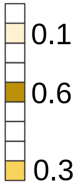
Ex. application: *head pose*

$\boldsymbol{y} =$ 

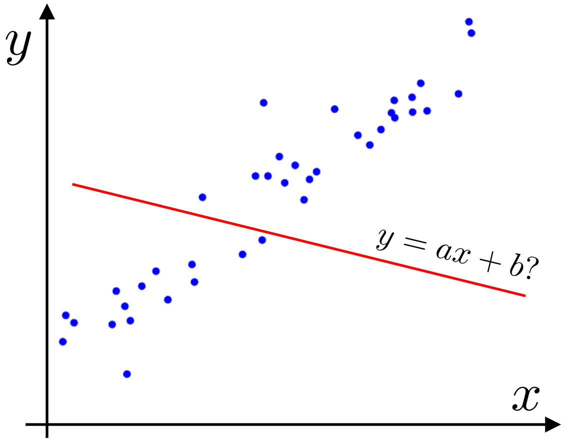


3. Sparse case:
- Multi-label classification

Ex. application: *image labelling*

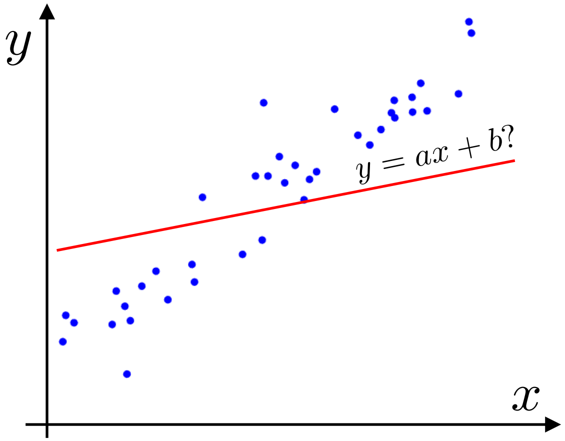
$\boldsymbol{y} =$ 

Generalizing linear regression



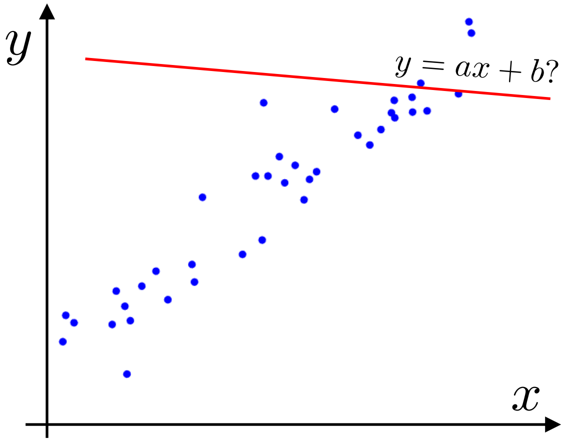
- Training set: $\mathcal{T} = \{(x_t, y_t)\}_{t=1}^{T}$
- Models: $f_{\boldsymbol{\theta}}(x) = ax + b$
- Parameters: $\boldsymbol{\theta} = [a, b]^\top \in \mathbb{R}^2$
- Total Loss: $g(\boldsymbol{\theta}) = L(f_{\boldsymbol{\theta}}, \mathcal{T}) = \frac{1}{T}\sum_{t=1}^{T}(f_{\boldsymbol{\theta}}(x_t)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

Generalizing linear regression



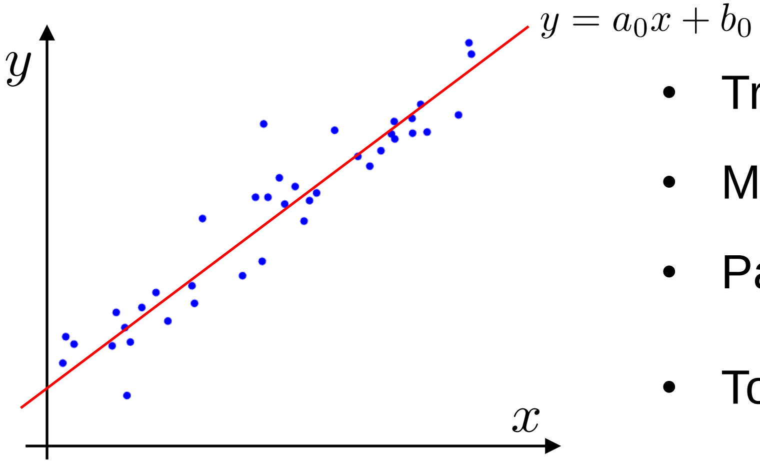
- Training set: $\mathcal{T} = \{(x_t, y_t)\}_{t=1}^{T}$
- Models: $f_{\boldsymbol{\theta}}(x) = ax + b$
- Parameters: $\boldsymbol{\theta} = [a, b]^\top \in \mathbb{R}^2$
- Total Loss: $g(\boldsymbol{\theta}) = L(f_{\boldsymbol{\theta}}, \mathcal{T}) = \frac{1}{T}\sum_{t=1}^{T}(f_{\boldsymbol{\theta}}(x_t)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

Generalizing linear regression



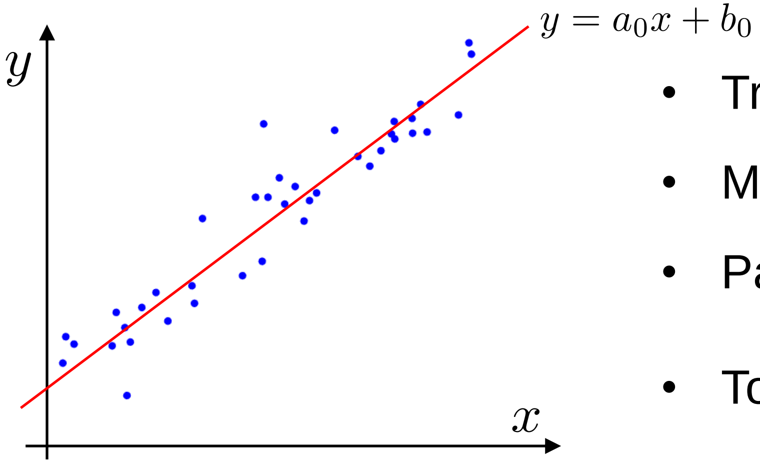
- Training set: $\mathcal{T} = \{(x_t, y_t)\}_{t=1}^{T}$
- Models: $f_{\boldsymbol{\theta}}(x) = ax + b$
- Parameters: $\boldsymbol{\theta} = [a, b]^\top \in \mathbb{R}^2$
- Total Loss: $g(\boldsymbol{\theta}) = L(f_{\boldsymbol{\theta}}, \mathcal{T}) = \dfrac{1}{T}\sum_{t=1}^{T}(f_{\boldsymbol{\theta}}(x_t)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

Generalizing linear regression



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

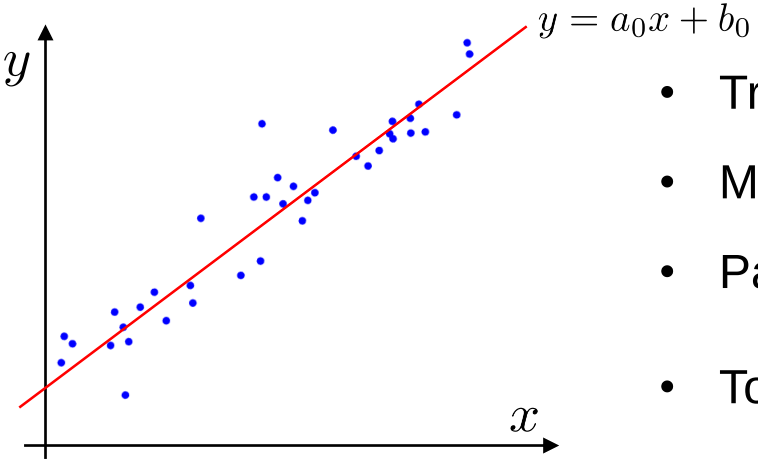
Generalizing linear regression



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

$$g(\boldsymbol{\theta}) = \frac{1}{T}\sum_{t=1}^{T}\Bigg(\underbrace{[x_t, 1]^\top}_{\boldsymbol{w}_t \in \mathbb{R}^2}\begin{bmatrix} a \\ b \end{bmatrix} - y_t\Bigg)^2 = \frac{1}{T}\|\mathbf{W}\boldsymbol{\theta} - \boldsymbol{y}\|_2^2, \quad \nabla_\theta g(\boldsymbol{\theta}_0) = \mathbf{0} \Rightarrow \boldsymbol{\theta}_0 = (\mathbf{W}^\top\mathbf{W})^{-1}\mathbf{W}^\top\boldsymbol{y}$$

Generalizing linear regression

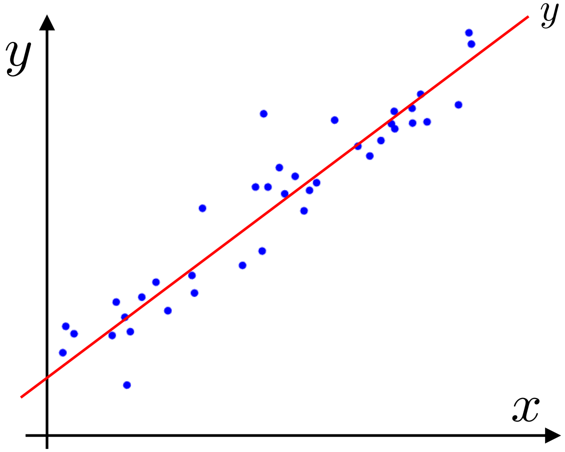


- Training set: $\mathcal{T} = \{(x_t, y_t)\}_{t=1}^{T}$
- Models: $f_{\boldsymbol{\theta}}(x) = ax + b$
- Parameters: $\boldsymbol{\theta} = [a, b]^\top \in \mathbb{R}^2$
- Total Loss: $g(\boldsymbol{\theta}) = L(f_{\boldsymbol{\theta}}, \mathcal{T}) = \frac{1}{T}\sum_{t=1}^{T}(f_{\boldsymbol{\theta}}(x_t)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

$$g(\boldsymbol{\theta}) = \frac{1}{T}\sum_{t=1}^{T}\Bigg(\underbrace{[x_t, 1]^\top}_{\boldsymbol{w}_t \in \mathbb{R}^2}\begin{bmatrix} a \\ b \end{bmatrix} - y_t\Bigg)^2 = \frac{1}{T}\|\mathbf{W}\boldsymbol{\theta} - \boldsymbol{y}\|_2^2, \quad \nabla_\theta g(\boldsymbol{\theta}_0) = \mathbf{0} \Rightarrow \boldsymbol{\theta}_0 = (\mathbf{W}^\top\mathbf{W})^{-1}\mathbf{W}^\top\boldsymbol{y}$$

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

Generalizing linear regression



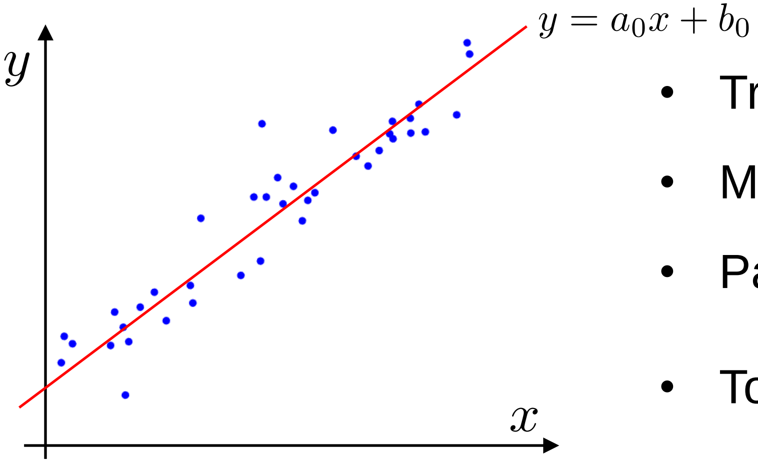
- Training set: $\mathcal{T} = \{(x_t, y_t)\}_{t=1}^{T}$
- Models: $f_{\boldsymbol{\theta}}(x) = ax + b$
- Parameters: $\boldsymbol{\theta} = [a, b]^\top \in \mathbb{R}^2$
- Total Loss: $g(\boldsymbol{\theta}) = L(f_{\boldsymbol{\theta}}, \mathcal{T}) = \frac{1}{T}\sum_{t=1}^{T}(f_{\boldsymbol{\theta}}(x_t)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

$$g(\boldsymbol{\theta}) = \frac{1}{T}\sum_{t=1}^{T}\Big(\underbrace{[x_t, 1]^\top}_{\boldsymbol{w}_t \in \mathbb{R}^2}\begin{bmatrix} a \\ b \end{bmatrix} - y_t\Big)^2 = \frac{1}{T}\|\mathbf{W}\boldsymbol{\theta} - \boldsymbol{y}\|_2^2, \quad \nabla_\theta g(\boldsymbol{\theta}_0) = \mathbf{0} \Rightarrow \boldsymbol{\theta}_0 = (\mathbf{W}^\top\mathbf{W})^{-1}\mathbf{W}^\top\boldsymbol{y}$$

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

$$g(\boldsymbol{\theta}) = \frac{1}{T}\sum_{t=1}^{T}\|\boldsymbol{A}\boldsymbol{x}_t + \boldsymbol{b} - \boldsymbol{y}_t\|_2^2,$$

Generalizing linear regression



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

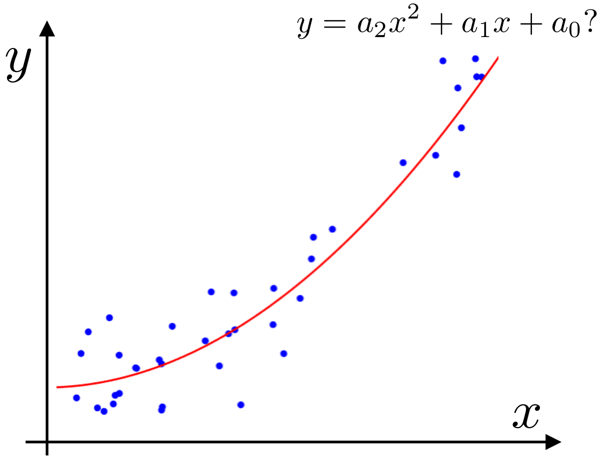
$$g(\boldsymbol{\theta}) = \frac{1}{T}\sum_{t=1}^{T}\Bigg(\underbrace{[x_t, 1]^\top}_{\boldsymbol{w}_t \in \mathbb{R}^2}\begin{bmatrix} a \\ b \end{bmatrix} - y_t\Bigg)^2 = \frac{1}{T}\|\mathbf{W}\boldsymbol{\theta} - \boldsymbol{y}\|_2^2, \quad \nabla_\theta g(\boldsymbol{\theta}_0) = \mathbf{0} \Rightarrow \boldsymbol{\theta}_0 = (\mathbf{W}^\top\mathbf{W})^{-1}\mathbf{W}^\top\boldsymbol{y}$$

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

$$g(\boldsymbol{\theta}) = \frac{1}{T}\sum_{t=1}^{T}\|\boldsymbol{A}\boldsymbol{x}_t + \boldsymbol{b} - \boldsymbol{y}_t\|_2^2, \quad \nabla_\theta g(\boldsymbol{\theta}_0) = \mathbf{0} \Rightarrow \begin{bmatrix} \hat{\boldsymbol{a}}_{n,0} \\ b_{n,0} \end{bmatrix} = (\mathbf{W}^\top\mathbf{W})^{-1}\mathbf{W}^\top\boldsymbol{y}_n, \ \forall n$$

$$\mathbf{W} = \begin{bmatrix} \boldsymbol{x}_1^\top, & 1 \\ \vdots & \vdots \\ \boldsymbol{x}_T^\top, & 1 \end{bmatrix} \in \mathbb{R}^{T\times(D+1)}$$

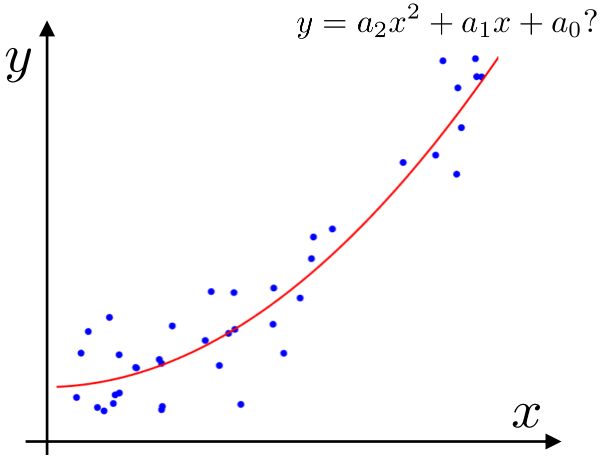
Generalizing linear regression



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

→ What about polynomial regression?

Generalizing linear regression

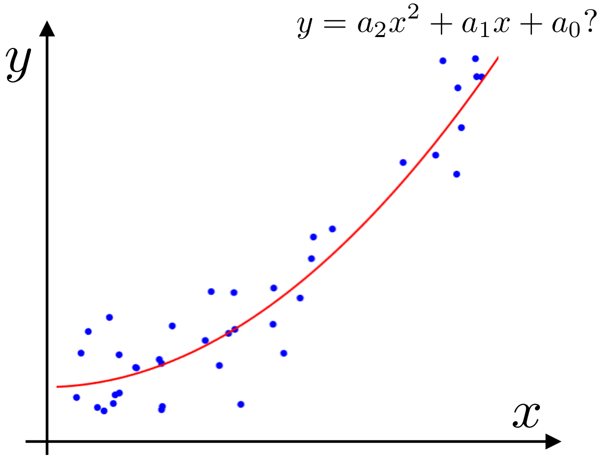


- Training set: $\mathcal{T} = \{(x_t, y_t)\}_{t=1}^T$
- Models: $f_{\boldsymbol{\theta}}(x) = a_2x^2 + a_1x + a_0$
- Parameters: $\boldsymbol{\theta} = [a_0, a_1, a_2]^\top \in \mathbb{R}^3$
- Total Loss: $g(\boldsymbol{\theta}) = L(f_{\boldsymbol{\theta}}, \mathcal{T}) = \frac{1}{T}\sum_{t=1}^T (f_{\boldsymbol{\theta}}(x_t)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

$\rightarrow$ 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



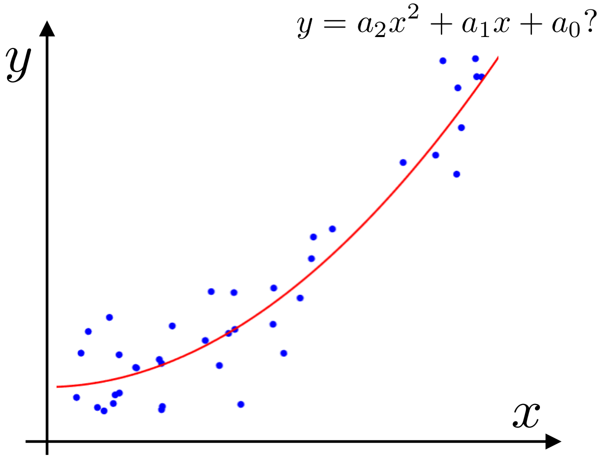
- Training set: $\mathcal{T} = \{(x_t, y_t)\}_{t=1}^{T}$
- Models: $f_{\boldsymbol{\theta}}(x) = a_2x^2 + a_1x + a_0$
- Parameters: $\boldsymbol{\theta} = [a_0, a_1, a_2]^\top \in \mathbb{R}^3$
- Total Loss: $g(\boldsymbol{\theta}) = L(f_{\boldsymbol{\theta}}, \mathcal{T}) = \frac{1}{T}\sum_{t=1}^{T}(f_{\boldsymbol{\theta}}(x_t)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

→ 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^{nd} degree **multivariate** polynomial regression $(\boldsymbol{y} \in \mathbb{R}^N, \boldsymbol{x} \in \mathbb{R}^D)$:

$$y_n = \sum_{i=1}^{D}\sum_{j=i}^{D} a_{ij}^{(n)} x_i x_j + \sum_{i=1}^{D} a_i^{(n)} x_i + a_0^{(n)} \Rightarrow y_n = [a_0^{(n)}, a_1^{(n)}, \dots, a_{11}^{(n)}, a_{12}^{(n)} \dots, a_{DD}^{(n)}] \times \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_1^2 \\ x_1x_2 \\ \vdots \\ x_D^2 \end{bmatrix}$$

Generalizing linear regression



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

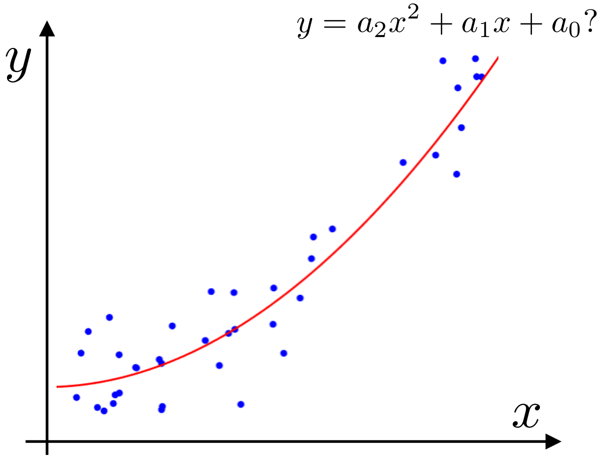
→ What about polynomial regression?

- 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 $(\boldsymbol{y} \in \mathbb{R}^N, \boldsymbol{x} \in \mathbb{R}^D)$:

$$y_n = \sum_{i=1}^{D}\sum_{j=i}^{D} a_{ij}^{(n)} x_i x_j + \sum_{i=1}^{D} a_i^{(n)} x_i + a_0^{(n)} \Rightarrow y_n = [a_0^{(n)}, a_1^{(n)}, \dots, a_{11}^{(n)}, a_{12}^{(n)} \dots, a_{DD}^{(n)}] \times \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_1^2 \\ x_1x_2 \\ \vdots \\ x_D^2 \end{bmatrix}$$

→How many parameters?

Generalizing linear regression



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

→ 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 $(\boldsymbol{y} \in \mathbb{R}^N, \boldsymbol{x} \in \mathbb{R}^D)$:

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

→*How many parameters?* $\mathcal{O}(ND^2)$

Generalizing linear regression

- K[th] degree multivariate polynomial regression:

$$y_n = \sum_{i_1=1}^{D} \sum_{i_2=i_1}^{D} \cdots \sum_{i_K=i_{K-1}}^{D} a^{(n)}_{i_1 i_2 \dots i_K} x_{i_1} x_{i_2} \dots x_{i_K} + \cdots + \sum_{i_1=1}^{D} \sum_{i_2=i_1}^{D} a^{(n)}_{i_1 i_2} x_{i_1} x_{i_2} + \sum_{i_1=1}^{D} a^{(n)}_{i_1} x_{i_1} + a^{(n)}_0$$

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

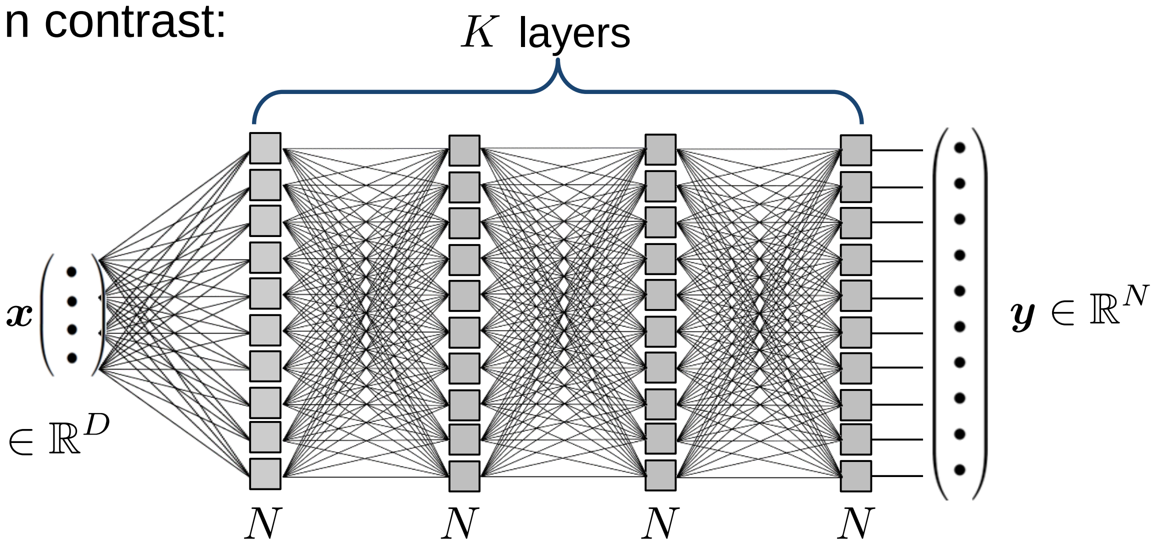
Generalizing linear regression

- Kth degree multivariate polynomial regression:

$$y_n = \sum_{i_1=1}^{D} \sum_{i_2=i_1}^{D} \cdots \sum_{i_K=i_{K-1}}^{D} a^{(n)}_{i_1 i_2 \ldots i_K} x_{i_1} x_{i_2} \ldots x_{i_K} + \cdots + \sum_{i_1=1}^{D} \sum_{i_2=i_1}^{D} a^{(n)}_{i_1 i_2} x_{i_1} x_{i_2} + \sum_{i_1=1}^{D} a^{(n)}_{i_1} x_{i_1} + a^{(n)}_0$$

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

- In contrast:



→How many parameters?

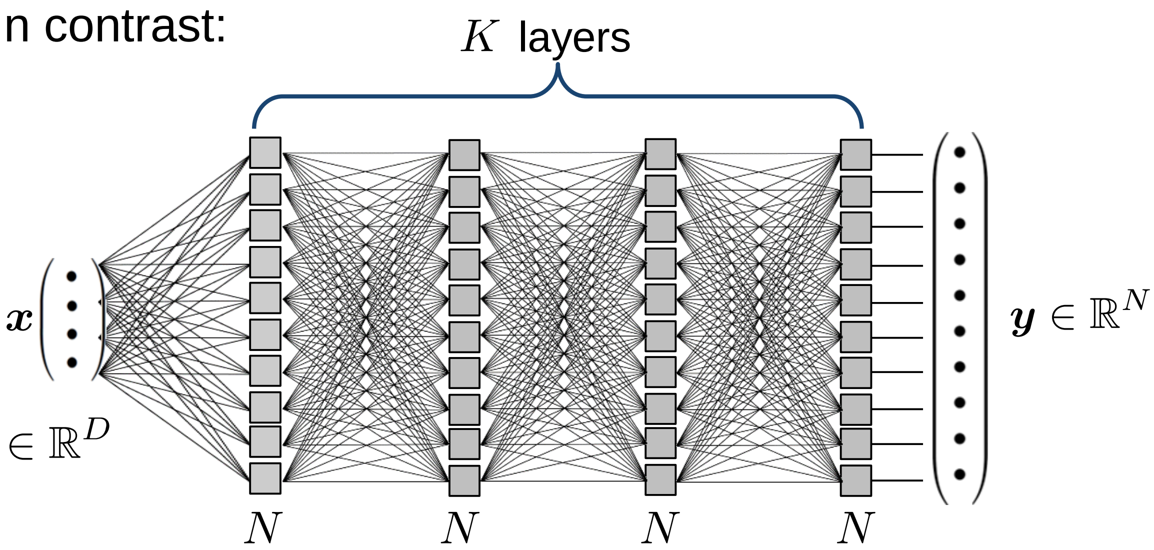
Generalizing linear regression

- K^{th} degree multivariate polynomial regression:

$$y_n = \sum_{i_1=1}^{D}\sum_{i_2=i_1}^{D}\cdots\sum_{i_K=i_{K-1}}^{D} a^{(n)}_{i_1 i_2 \dots i_K} x_{i_1} x_{i_2}\dots x_{i_K} + \dots + \sum_{i_1=1}^{D}\sum_{i_2=i_1}^{D} a^{(n)}_{i_1 i_2} x_{i_1} x_{i_2} + \sum_{i_1=1}^{D} a^{(n)}_{i_1} x_{i_1} + a^{(n)}_0$$

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

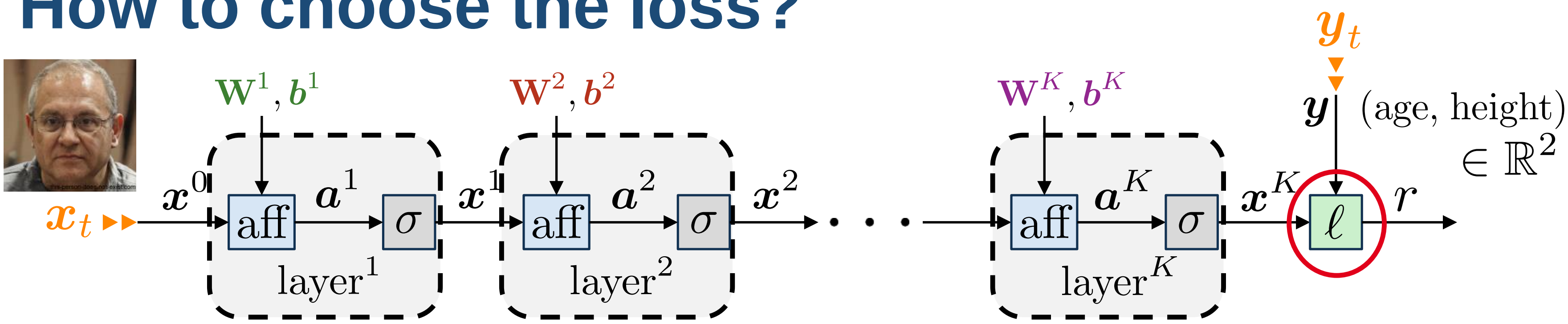
- In contrast:



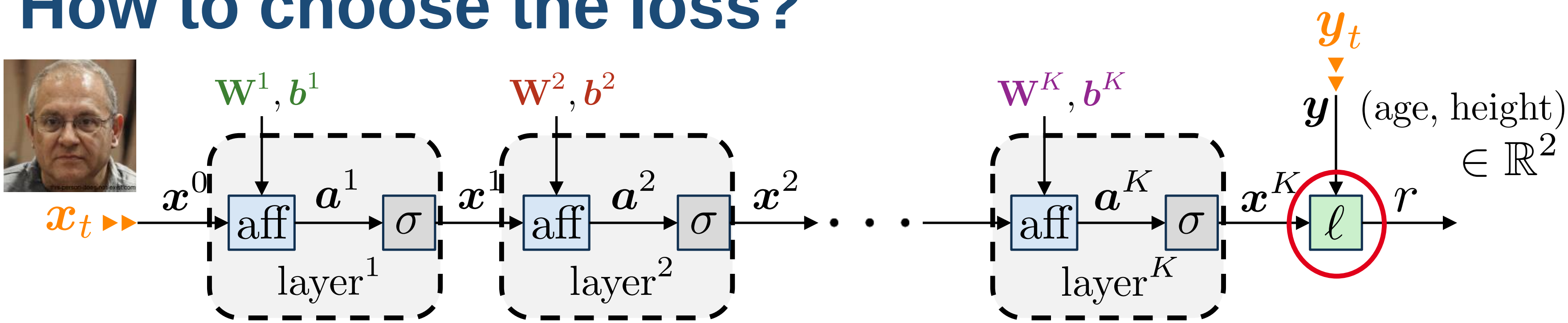
→How many parameters?

$$\mathcal{O}(DN + (K-1)N^2)$$

How to choose the loss?

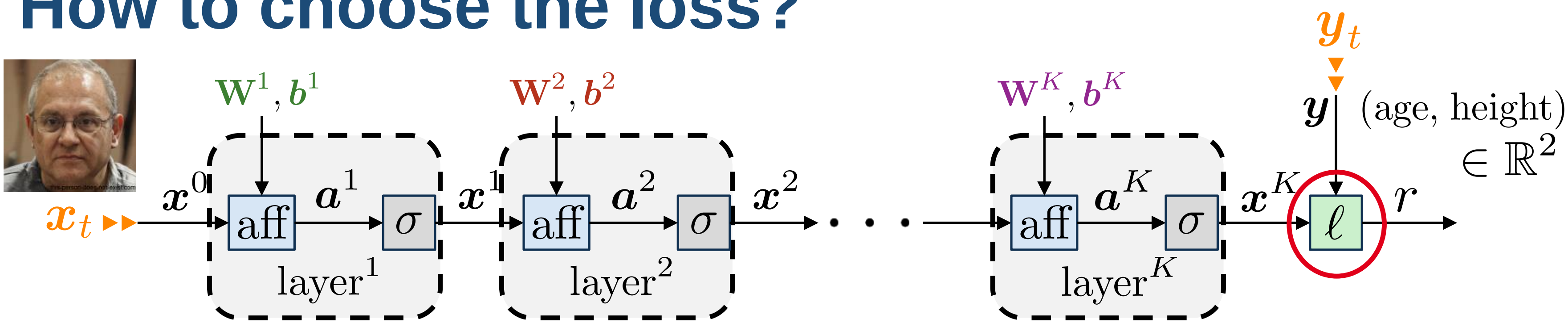


How to choose the loss?



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

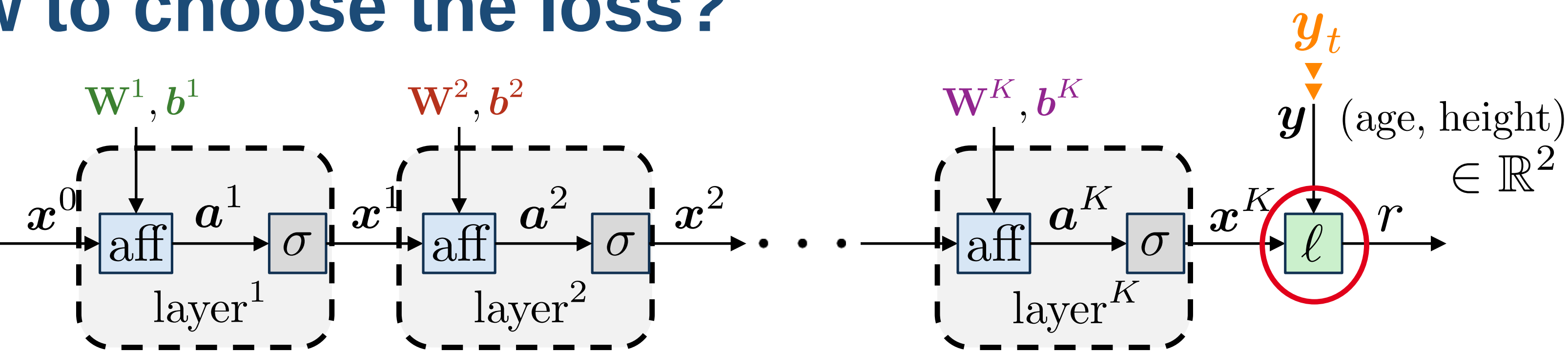
How to choose the loss?



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

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

How to choose the loss?



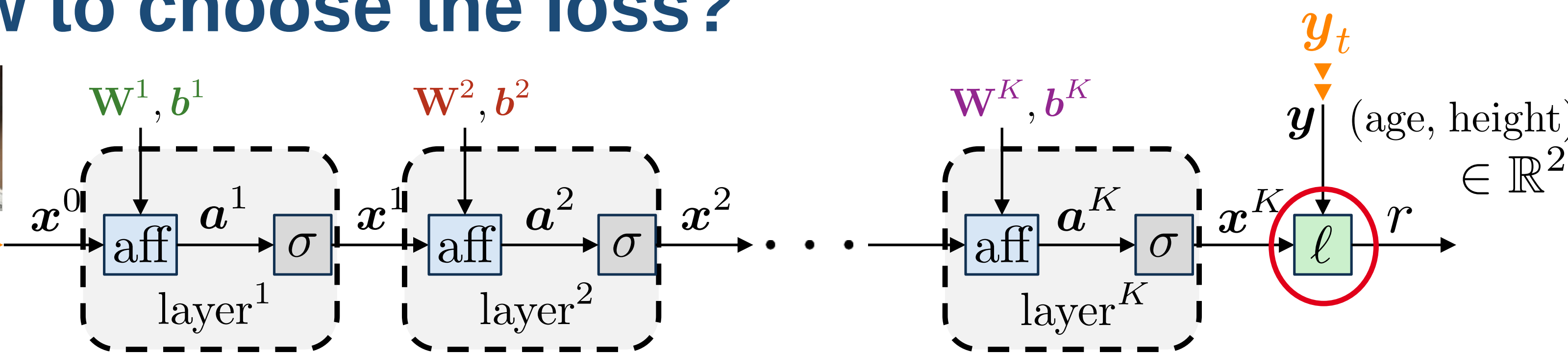
A general principled approach is to use the network to model $p(\boldsymbol{y}|\boldsymbol{x})$.

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

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

(Gaussian) (Bernouilli)

How to choose the loss?



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

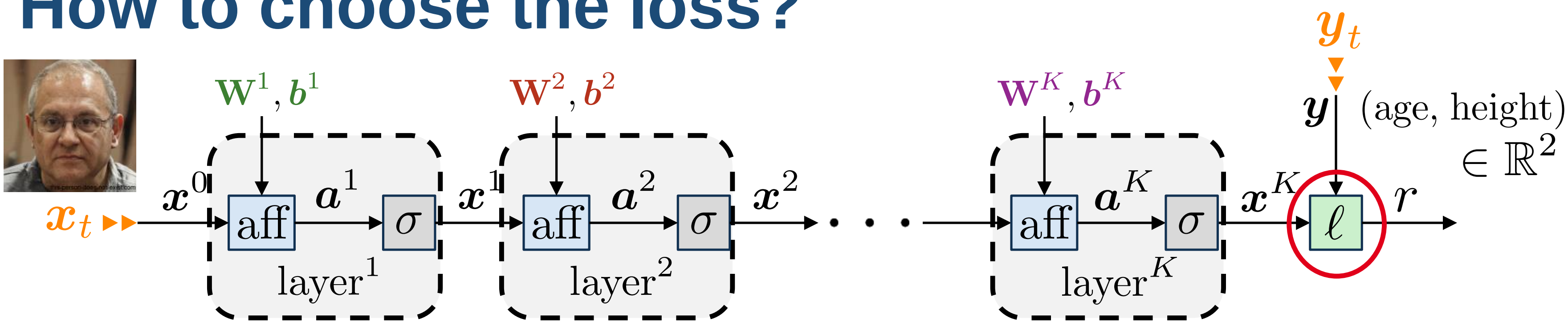
1) Choose a **simple** family of **parameterized** probabilistic distributions over the domain of $\boldsymbol{y}$ and $\boldsymbol{x}^K$, i.e., $\mathcal{P} = \{\tilde{p}_\lambda(\boldsymbol{y})\}_{\lambda \in \Lambda}$.

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

(Gaussian) (Bernouilli)

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

How to choose the loss?



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

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

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

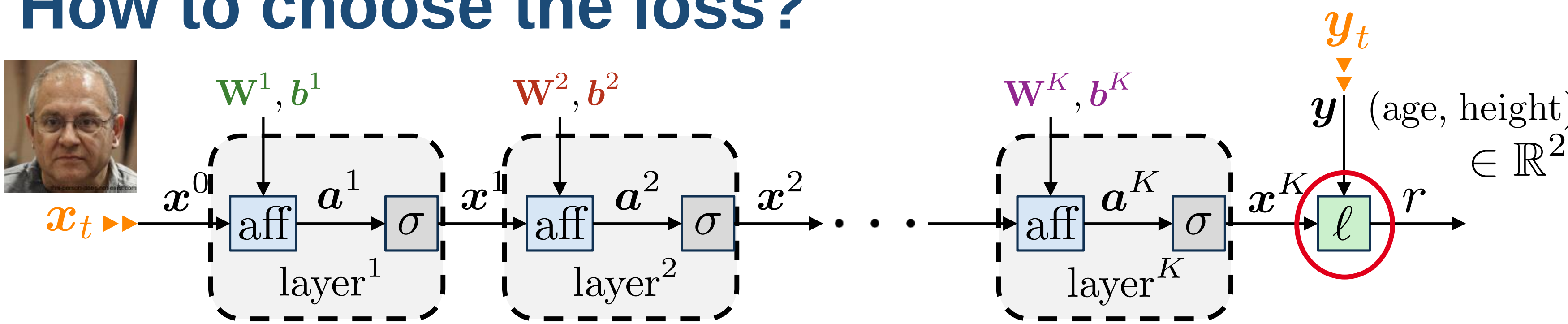
(Gaussian) (Bernouilli)

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

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

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \prod_{t=1}^{T} p_{\boldsymbol{\theta}}(\boldsymbol{y}_t|\boldsymbol{x}_t)$$

How to choose the loss?



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

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

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

(Gaussian) (Bernouilli)

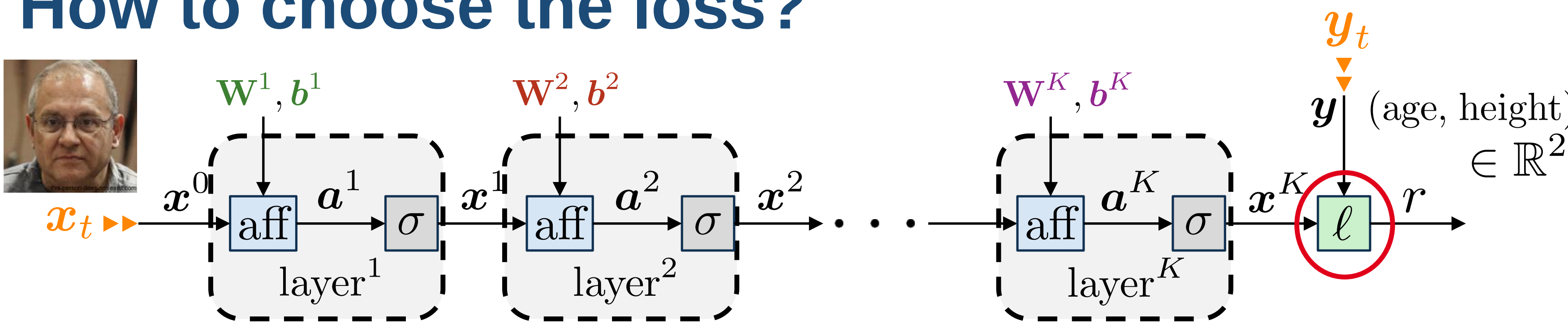
2) We then model the conditional probability as: $p_{\boldsymbol{\theta}}(\boldsymbol{y}|\boldsymbol{x}) = \tilde{p}_{\lambda=\mathrm{dnn}_{\boldsymbol{\theta}}(\boldsymbol{x})}(\boldsymbol{y})$

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

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \prod_{t=1}^{T} p_{\boldsymbol{\theta}}(\boldsymbol{y}_t|\boldsymbol{x}_t)$$

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

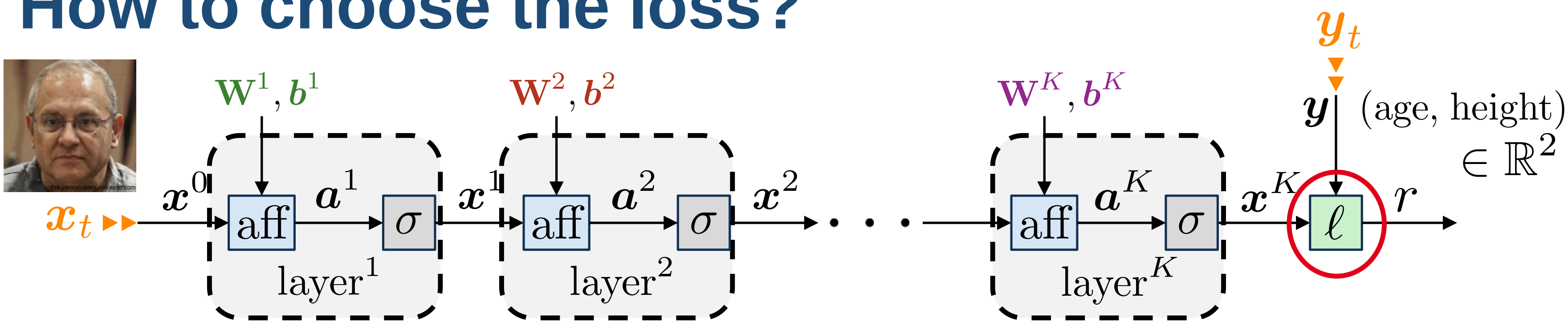
How to choose the loss?



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

$$L(\mathrm{dnn}_{\boldsymbol{\theta}}, \mathcal{T}) = -\log \prod_{t=1}^{T} p_{\boldsymbol{\theta}}(\boldsymbol{y}_t | \boldsymbol{x}_t)$$

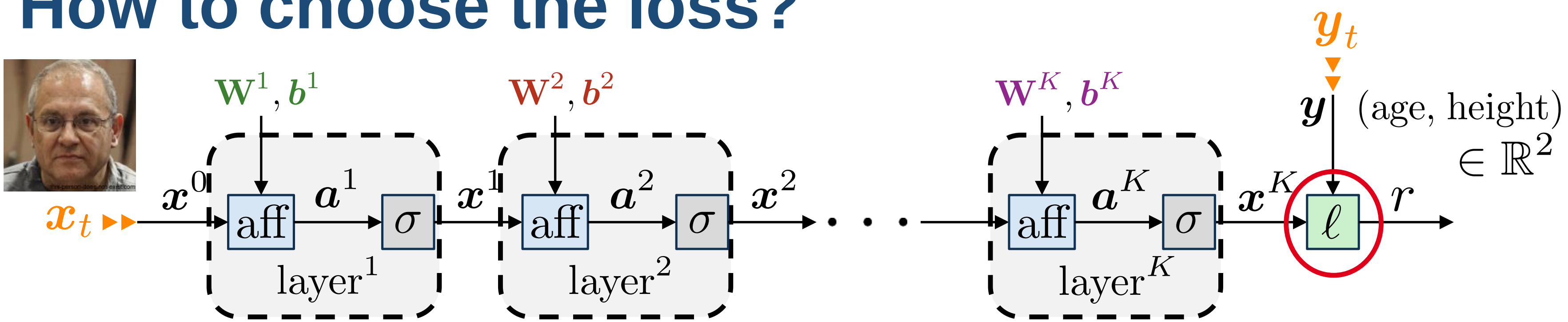
How to choose the loss?



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

$$L(\mathrm{dnn}_{\boldsymbol{\theta}}, \mathcal{T}) = -\log \prod_{t=1}^{T} p_{\boldsymbol{\theta}}(\boldsymbol{y}_t | \boldsymbol{x}_t) = \sum_{t=1}^{T} -\log p_{\boldsymbol{\theta}}(\boldsymbol{y}_t | \boldsymbol{x}_t)$$

How to choose the loss?

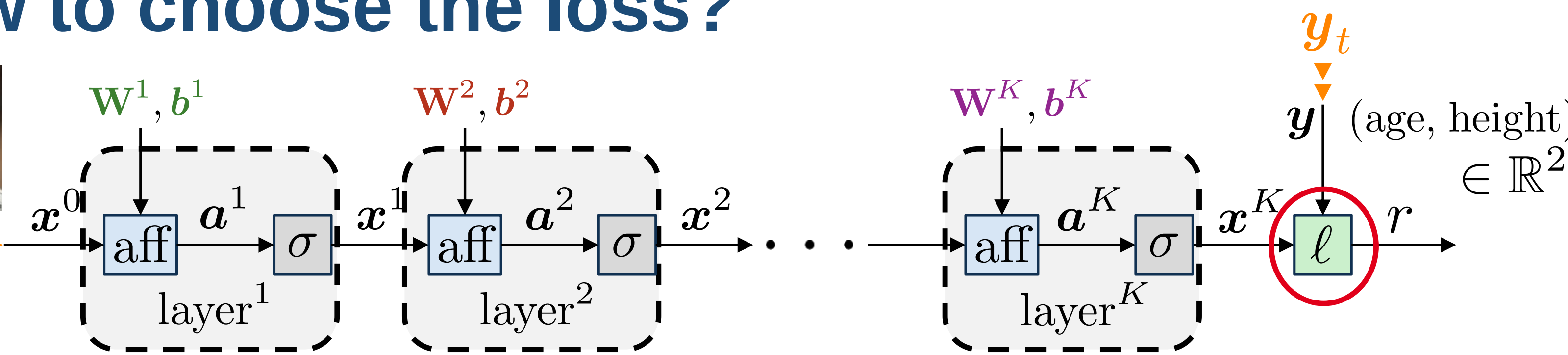


4) Usually, we define the **total loss** as the ***negative log-likelihood***:

$$L(\mathrm{dnn}_{\boldsymbol{\theta}}, \mathcal{T}) = -\log \prod_{t=1}^{T} p_{\boldsymbol{\theta}}(\boldsymbol{y}_t | \boldsymbol{x}_t) = \sum_{t=1}^{T} -\log p_{\boldsymbol{\theta}}(\boldsymbol{y}_t | \boldsymbol{x}_t)$$

$$= \sum_{t=1}^{T} -\log \tilde{p}_{\mathrm{dnn}_{\boldsymbol{\theta}}(\boldsymbol{x}_t)}(\boldsymbol{y}_t)$$

How to choose the loss?

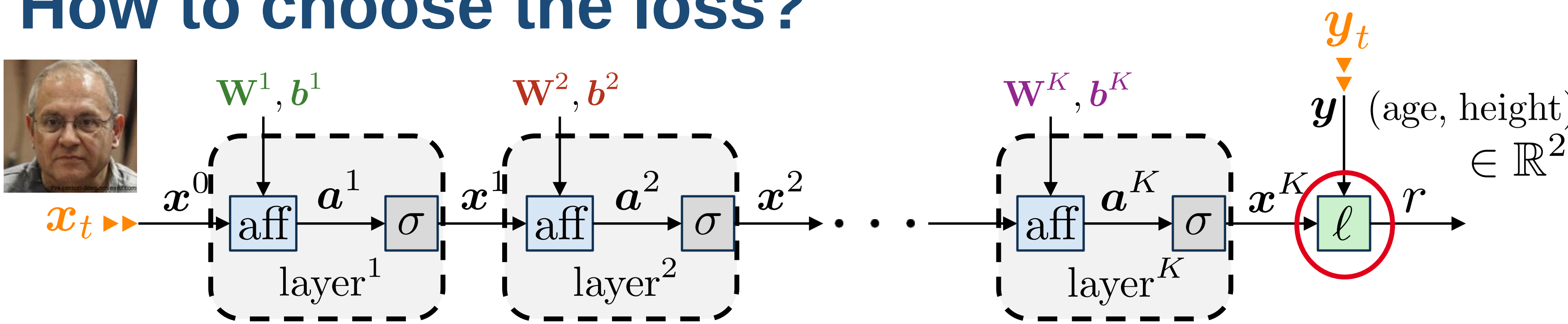


4) Usually, we define the **total loss** as the ***negative log-likelihood***:

$$L(\mathrm{dnn}_{\boldsymbol{\theta}}, \mathcal{T}) = -\log \prod_{t=1}^{T} p_{\boldsymbol{\theta}}(\boldsymbol{y}_t | \boldsymbol{x}_t) = \sum_{t=1}^{T} -\log p_{\boldsymbol{\theta}}(\boldsymbol{y}_t | \boldsymbol{x}_t)$$

$$= \sum_{t=1}^{T} -\log \tilde{p}_{\mathrm{dnn}_{\boldsymbol{\theta}}(\boldsymbol{x}_t)}(\boldsymbol{y}_t) = \sum_{t=1}^{T} -\log \tilde{p}_{\boldsymbol{x}_t^K}(\boldsymbol{y}_t)$$

How to choose the loss?



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

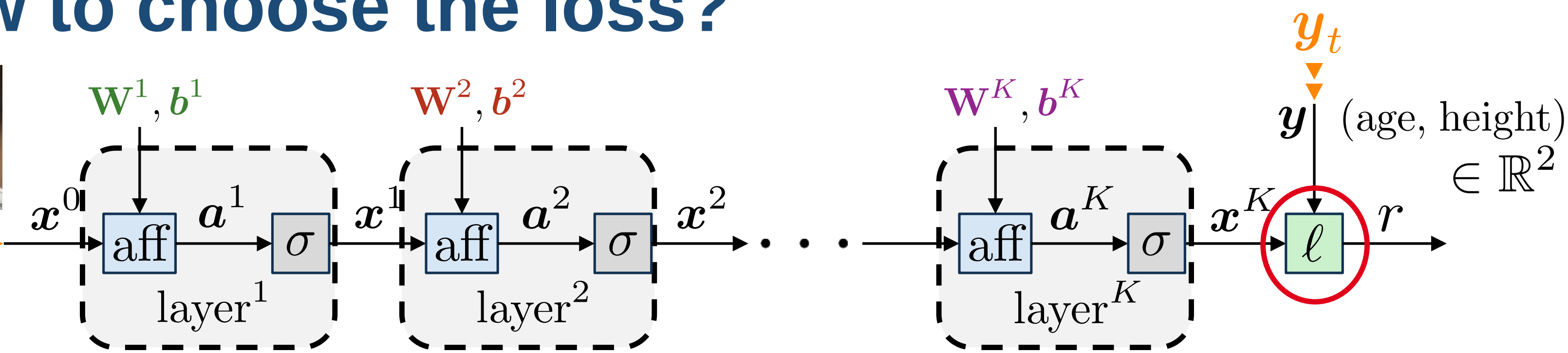
$$L(\mathrm{dnn}_{\boldsymbol{\theta}}, \mathcal{T}) = -\log \prod_{t=1}^{T} p_{\boldsymbol{\theta}}(\boldsymbol{y}_t | \boldsymbol{x}_t) = \sum_{t=1}^{T} -\log p_{\boldsymbol{\theta}}(\boldsymbol{y}_t | \boldsymbol{x}_t)$$

$$= \sum_{t=1}^{T} -\log \tilde{p}_{\mathrm{dnn}_{\boldsymbol{\theta}}(\boldsymbol{x}_t)}(\boldsymbol{y}_t) = \sum_{t=1}^{T} -\log \tilde{p}_{\boldsymbol{x}_t^K}(\boldsymbol{y}_t)$$

Example with the Gaussian distribution:

$$\ell(\boldsymbol{x}_t^K, \boldsymbol{y}_t) = -\log \tilde{p}_{\boldsymbol{x}_t^K}(\boldsymbol{y}_t) = -\log \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{\|\boldsymbol{x}_t^K - \boldsymbol{y}_t\|_2^2}{2}\right)$$

How to choose the loss?



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

$$L(\mathrm{dnn}_{\boldsymbol{\theta}}, \mathcal{T}) = -\log \prod_{t=1}^{T} p_{\boldsymbol{\theta}}(\boldsymbol{y}_t|\boldsymbol{x}_t) = \sum_{t=1}^{T} -\log p_{\boldsymbol{\theta}}(\boldsymbol{y}_t|\boldsymbol{x}_t)$$

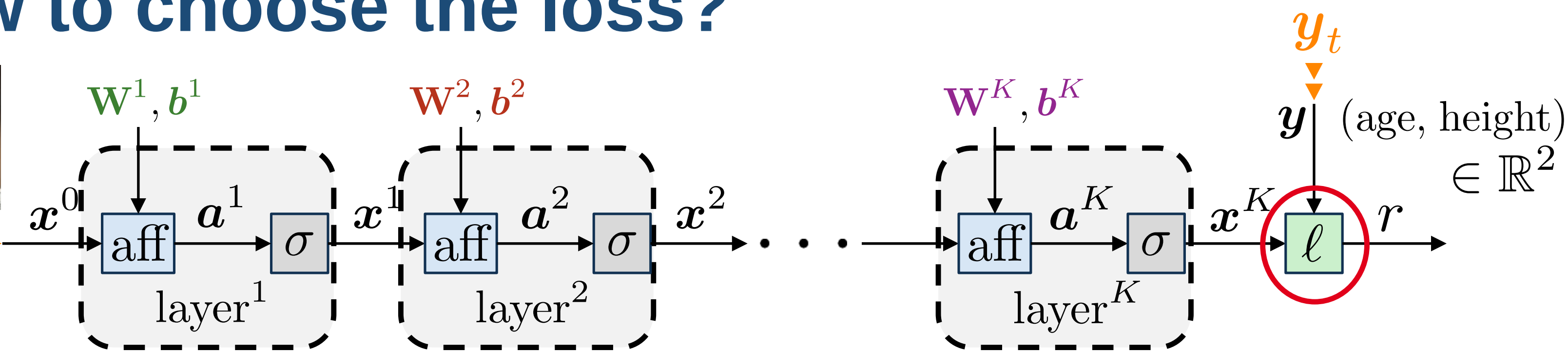
$$= \sum_{t=1}^{T} -\log \tilde{p}_{\mathrm{dnn}_{\boldsymbol{\theta}}(\boldsymbol{x}_t)}(\boldsymbol{y}_t) = \sum_{t=1}^{T} -\log \tilde{p}_{\boldsymbol{x}_t^K}(\boldsymbol{y}_t)$$

Example with the Gaussian distribution:

$$\ell(\boldsymbol{x}_t^K, \boldsymbol{y}_t) = -\log \tilde{p}_{\boldsymbol{x}_t^K}(\boldsymbol{y}_t) = -\log \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{\|\boldsymbol{x}_t^K - \boldsymbol{y}_t\|_2^2}{2}\right) \stackrel{c}{=} \frac{1}{2}\|\boldsymbol{x}_t^K - \boldsymbol{y}_t\|_2^2$$

$\rightarrow$ We recover the **L2 loss**!

How to choose the loss?



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

$$L(\text{dnn}_{\boldsymbol{\theta}}, \mathcal{T}) = -\log \prod_{t=1}^{T} p_{\boldsymbol{\theta}}(\boldsymbol{y}_t | \boldsymbol{x}_t) = \sum_{t=1}^{T} -\log p_{\boldsymbol{\theta}}(\boldsymbol{y}_t | \boldsymbol{x}_t)$$

$$= \sum_{t=1}^{T} -\log \tilde{p}_{\text{dnn}_{\boldsymbol{\theta}}(\boldsymbol{x}_t)}(\boldsymbol{y}_t) = \sum_{t=1}^{T} -\log \tilde{p}_{\boldsymbol{x}_t^K}(\boldsymbol{y}_t)$$

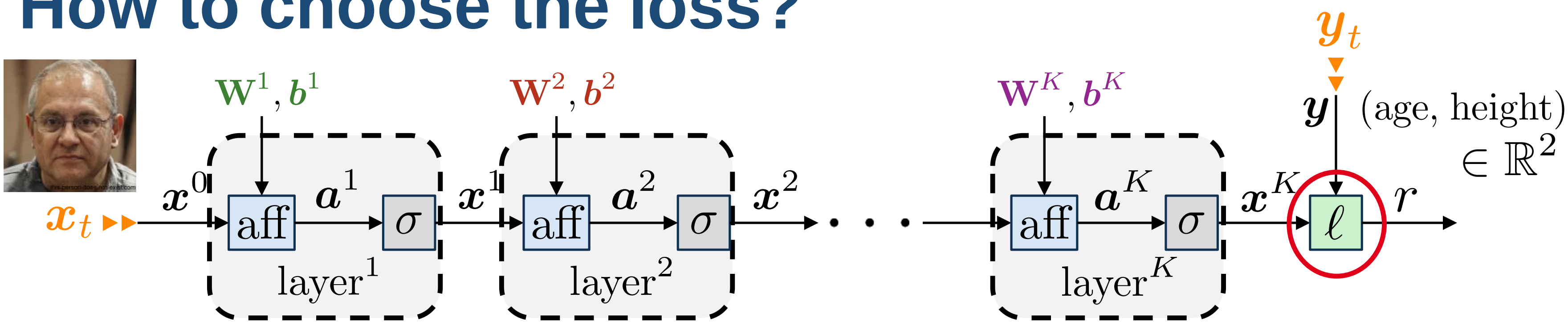
Example with the Gaussian distribution:

$$\ell(\boldsymbol{x}_t^K, \boldsymbol{y}_t) = -\log \tilde{p}_{\boldsymbol{x}_t^K}(\boldsymbol{y}_t) = -\log \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{\|\boldsymbol{x}_t^K - \boldsymbol{y}_t\|_2^2}{2}\right) \stackrel{c}{=} \frac{1}{2}\|\boldsymbol{x}_t^K - \boldsymbol{y}_t\|_2^2$$

→ We recover the **L2 loss**!

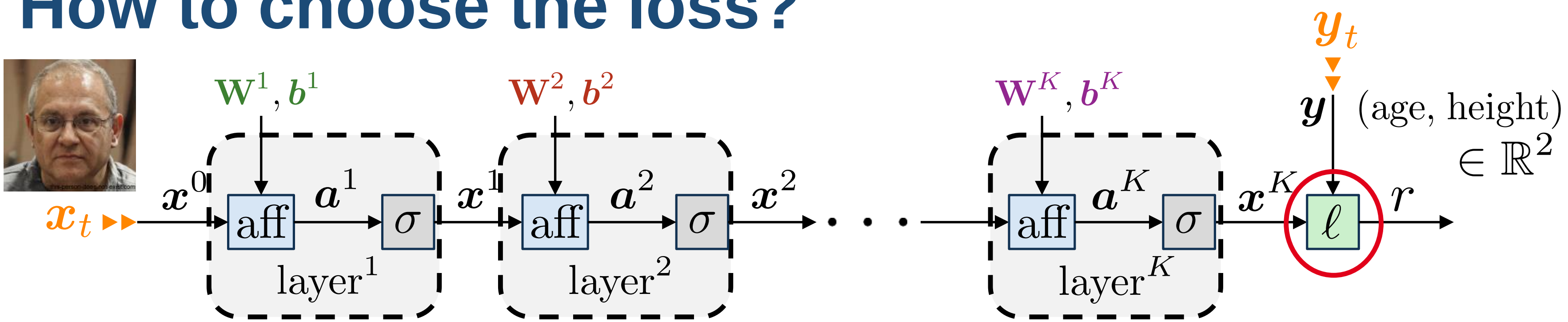
Using the L2 loss is equivalent to assuming the network will make i.i.d Gaussian errors.

How to choose the loss?



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

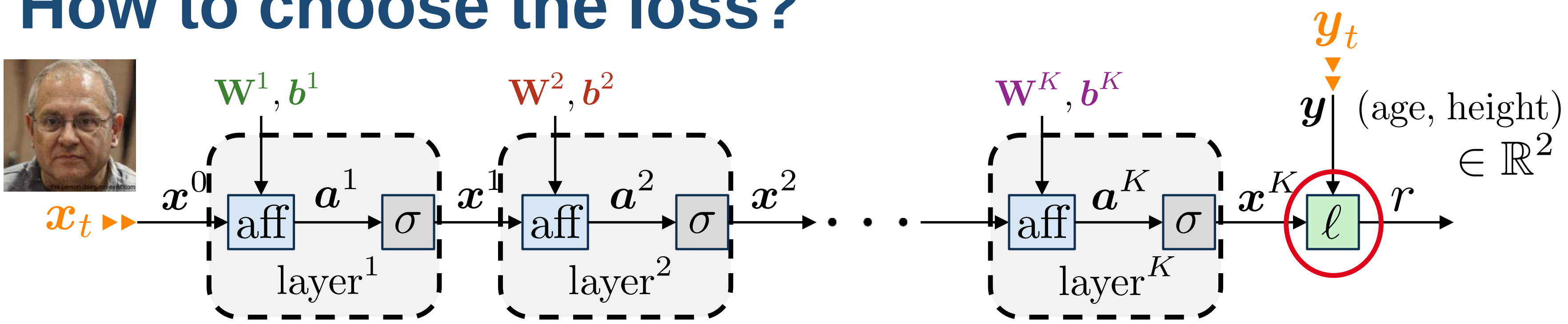
How to choose the loss?



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?



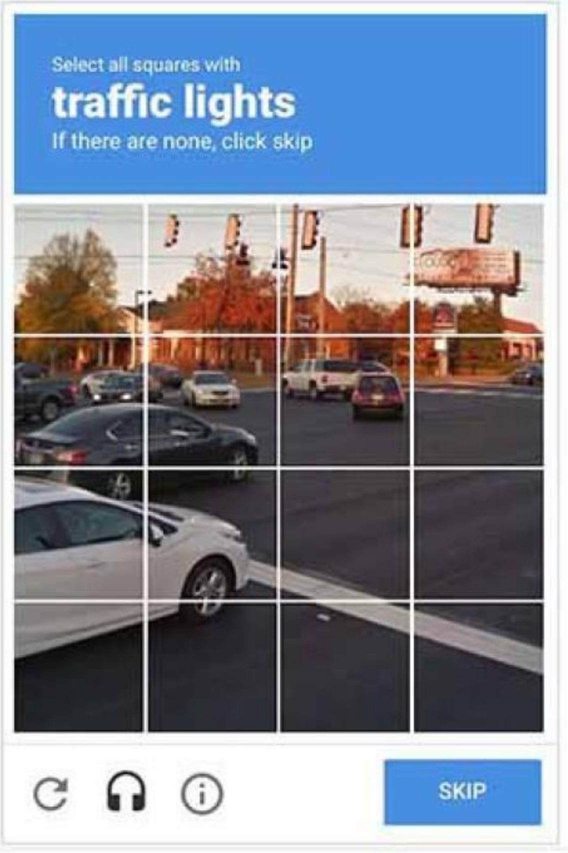
The approach is very general and can be used with a variety of parameterized probability distributions.

- 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*** (≈uncertainty) of the network output:

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

Detection

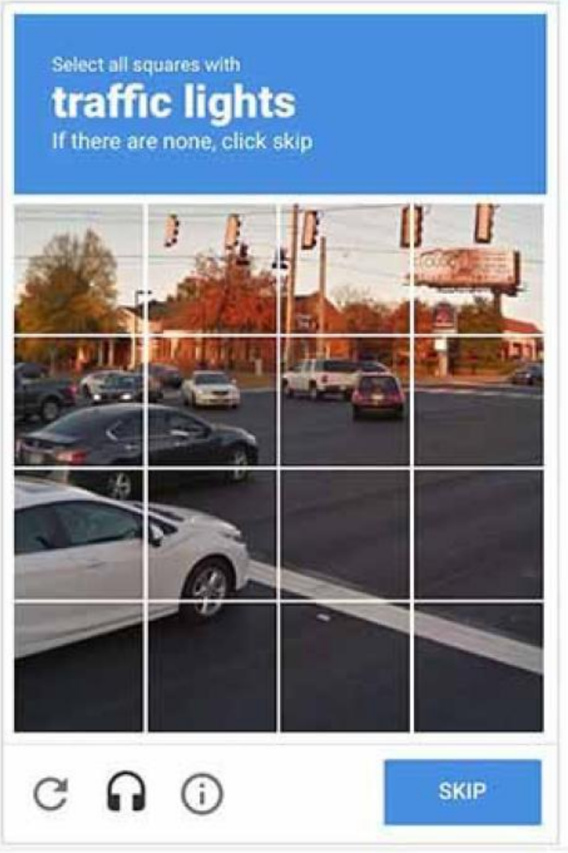
Example: Captcha $\boldsymbol{x}_t \in \mathbb{R}^D, \; y_t \in \{0, 1\}$



Detection

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

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

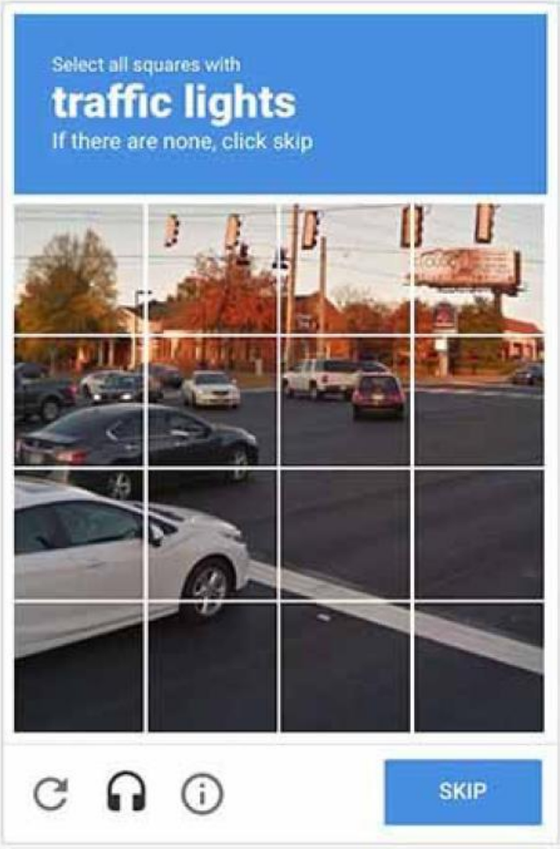


Detection

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

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

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



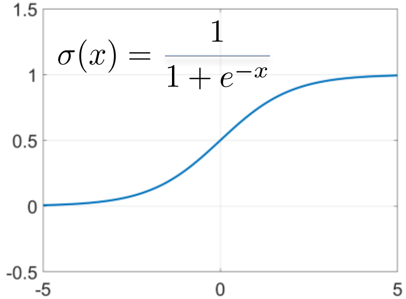
Detection

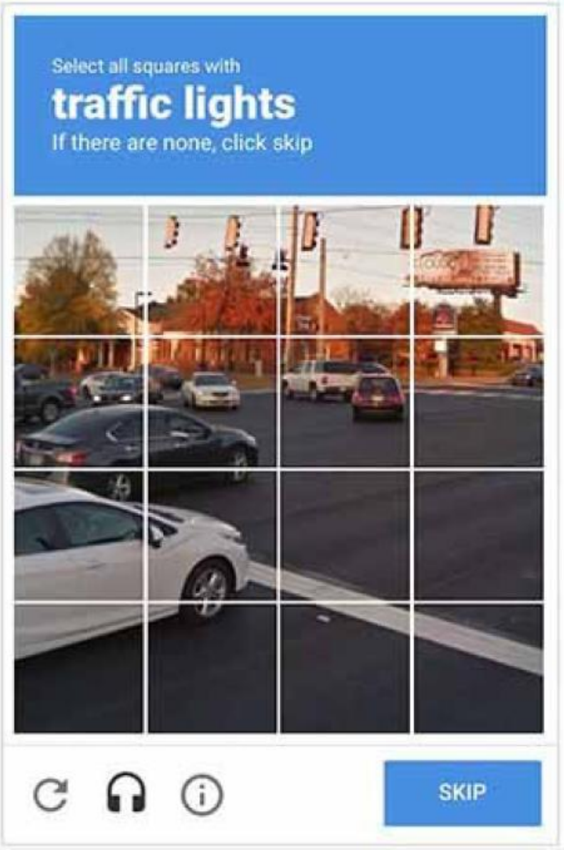
Example: Captcha $\boldsymbol{x}_t \in \mathbb{R}^D, \ y_t \in \{0, 1\}$

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

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

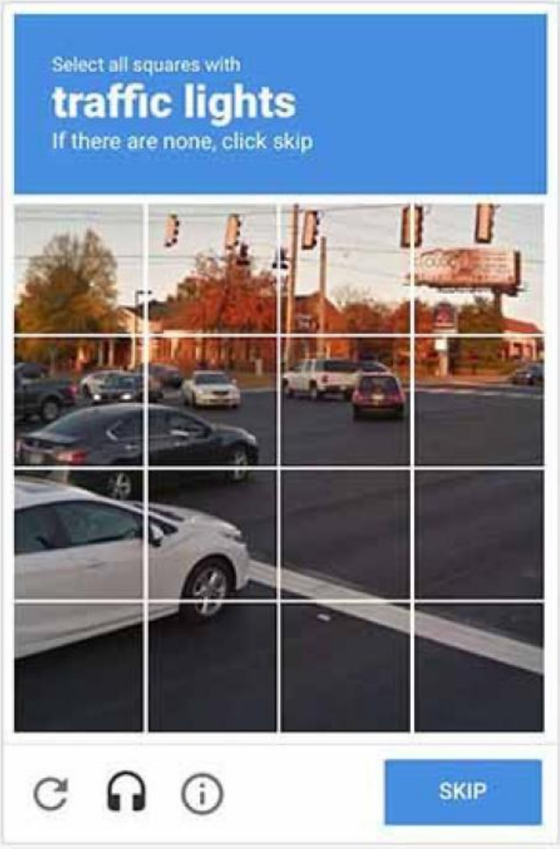
- Note that $b \in [0,1]$, hence we need to constrain the output of the network in this interval => we use a **sigmoid function** at the output:





Detection

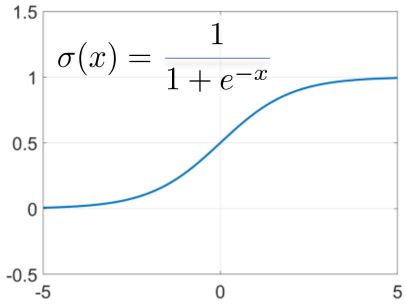
Example: Captcha $\boldsymbol{x}_t \in \mathbb{R}^D,\ y_t \in \{0,1\}$



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

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

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



- Using the ***maximum likelihood*** approach with this distribution, we obtain the following loss:

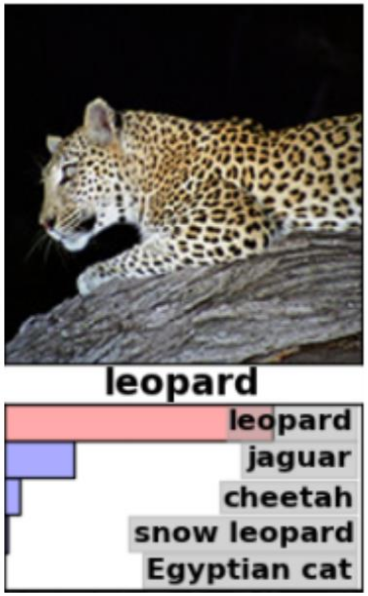
$$\ell(x_t^K, y_t) = -\log \tilde{p}_{x_t^K}(y_t) = -y_t \log x_t^K - (1-y_t)\log(1-x_t^K),$$

= the ***Binary Cross-Entropy***.

Classification

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

$$\boldsymbol{x}_t \in \mathbb{R}^D,\ y_t \in \{1, 2, \ldots, N\}$$



Ex: ImageNet (1000 classes)

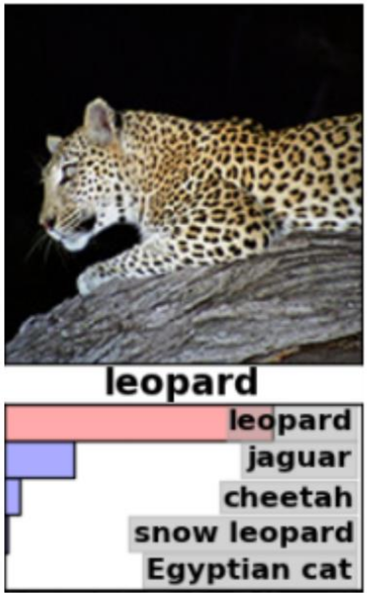
Classification

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

$$\boldsymbol{x}_t \in \mathbb{R}^D,\; y_t \in \{1, 2, \ldots, N\}$$

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

$$\boldsymbol{y}_t = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \rightarrow \text{index of the class}$$



Ex: ImageNet
(1000 classes)

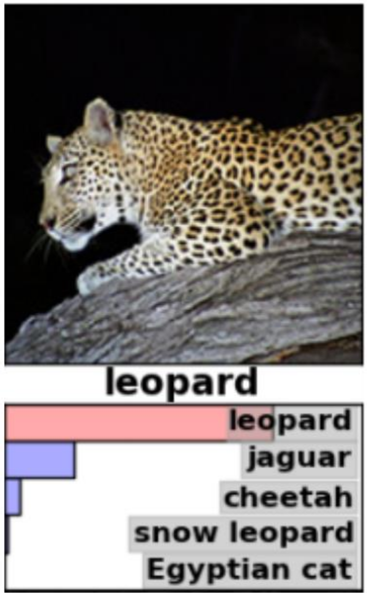
Classification

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

$$\boldsymbol{x}_t \in \mathbb{R}^D, \; y_t \in \{1, 2, \dots, N\}$$

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

$$\boldsymbol{y}_t = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \rightarrow \text{index of the class}$$



Ex: ImageNet
(1000 classes)

- We use a ***categorical distribution***:

$$\tilde{p}_{\boldsymbol{b}}(\boldsymbol{y}) = \begin{cases} b_1 \in [0,1] \text{ for } y_1 = 1 \\ b_2 \in [0,1] \text{ for } y_2 = 1 \\ \vdots \\ b_N \in [0,1] \text{ for } y_N = 1 \end{cases}, \text{ with } \textstyle\sum_n b_n = 1$$

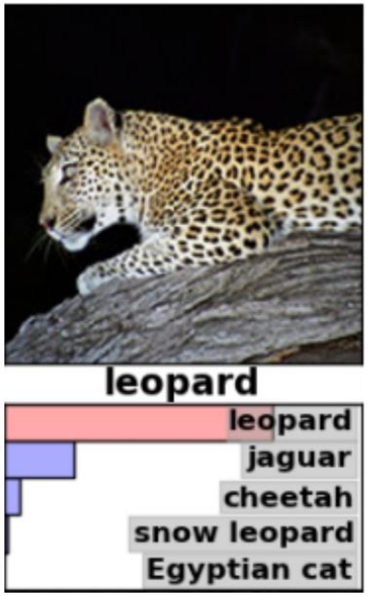
Classification

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

$$\boldsymbol{x}_t \in \mathbb{R}^D, \; y_t \in \{1, 2, \dots, N\}$$

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

$$\boldsymbol{y}_t = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \rightarrow \text{index of the class}$$



Ex: ImageNet (1000 classes)

- We use a ***categorical distribution***:

How to enforce this constraints at the network output?

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

Classification

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

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

Classification

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

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

- Can be viewed as a generalization of the sigmoid

Classification

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

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

- 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

Classification

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

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

- 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**:

$$\ell(\boldsymbol{x}_t^K, \boldsymbol{y}_t) = -\log \tilde{p}_{\boldsymbol{x}_t^K}(\boldsymbol{y}_t) = \sum_{n=1}^{N} -y_{t,n} \log x_{t,n}^K$$

Classification

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

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

- 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**:

$$\ell(\boldsymbol{x}_t^K, \boldsymbol{y}_t) = -\log \tilde{p}_{\boldsymbol{x}_t^K}(\boldsymbol{y}_t) = \sum_{n=1}^{N} -y_{t,n} \log x_{t,n}^K$$

Multi-Label Classification

Classification

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

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

- 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**:

$$\ell(\boldsymbol{x}_t^K, \boldsymbol{y}_t) = -\log \tilde{p}_{\boldsymbol{x}_t^K}(\boldsymbol{y}_t) = \sum_{n=1}^{N} -y_{t,n} \log x_{t,n}^K$$

Multi-Label Classification

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

Classification

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

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

- 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**:

$$\ell(\boldsymbol{x}_t^K, \boldsymbol{y}_t) = -\log \tilde{p}_{\boldsymbol{x}_t^K}(\boldsymbol{y}_t) = \sum_{n=1}^{N} -y_{t,n} \log x_{t,n}^K$$

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.

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** (*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

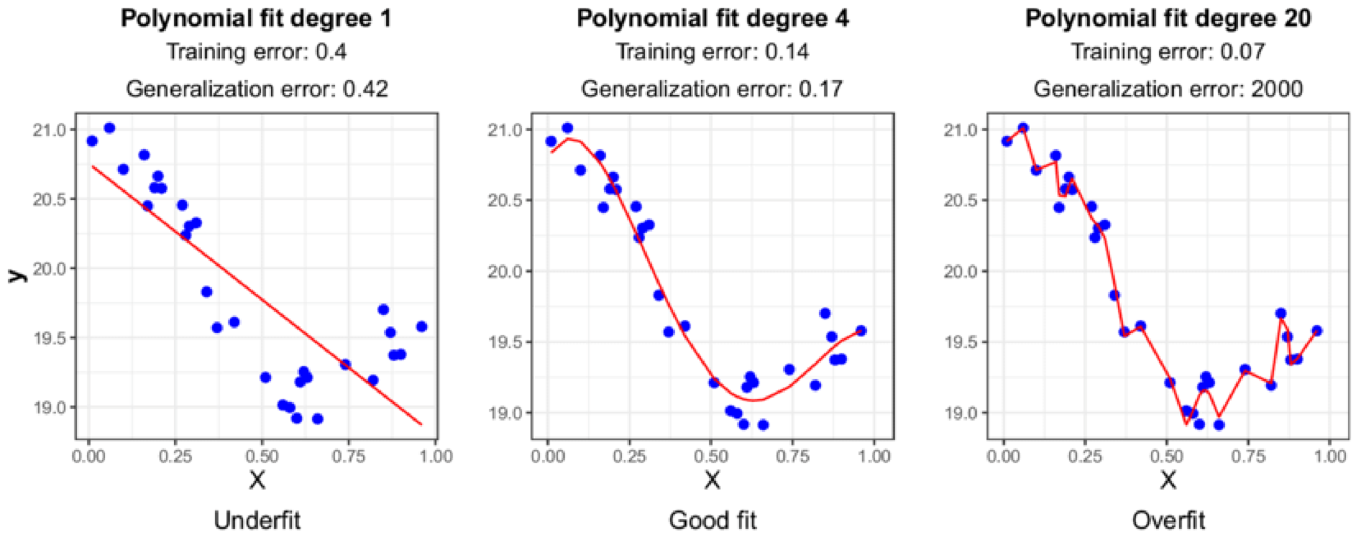
- Our algorithm is guilty of **overfitting** (*sur-apprentissage*)
- Instead of learning general features to classify the images, it **learned by heart** all the images in our training dataset!
- Remember that we often have **millions** or **billions** or parameters in a deep model. Hence, it has the **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.

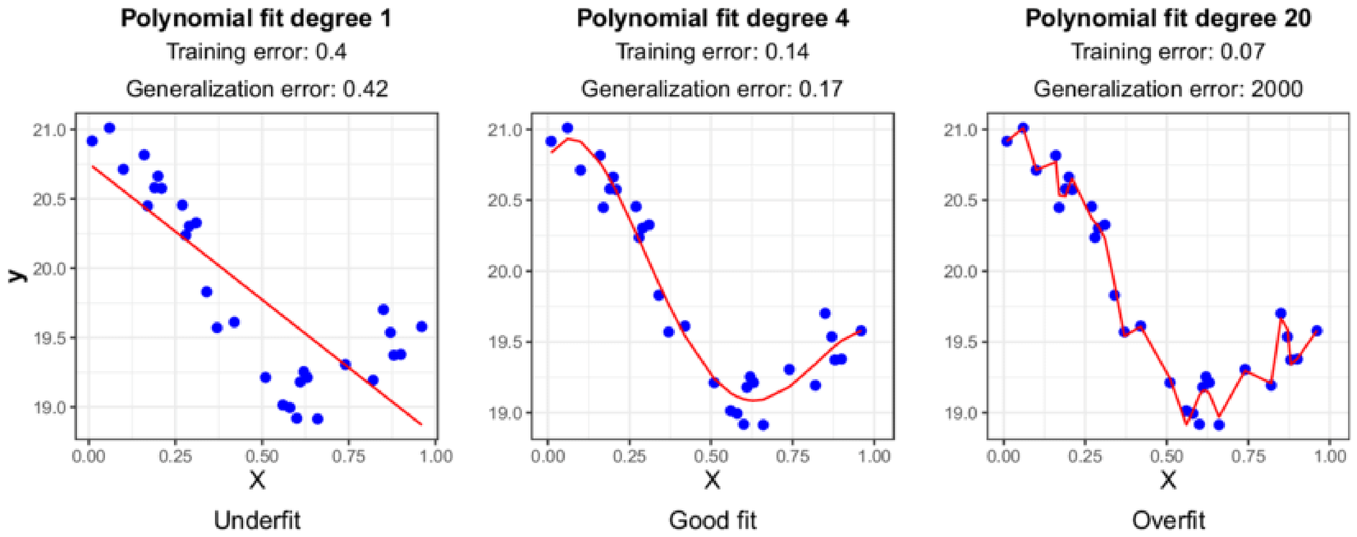
Overfitting

- Ex: polynomial regression

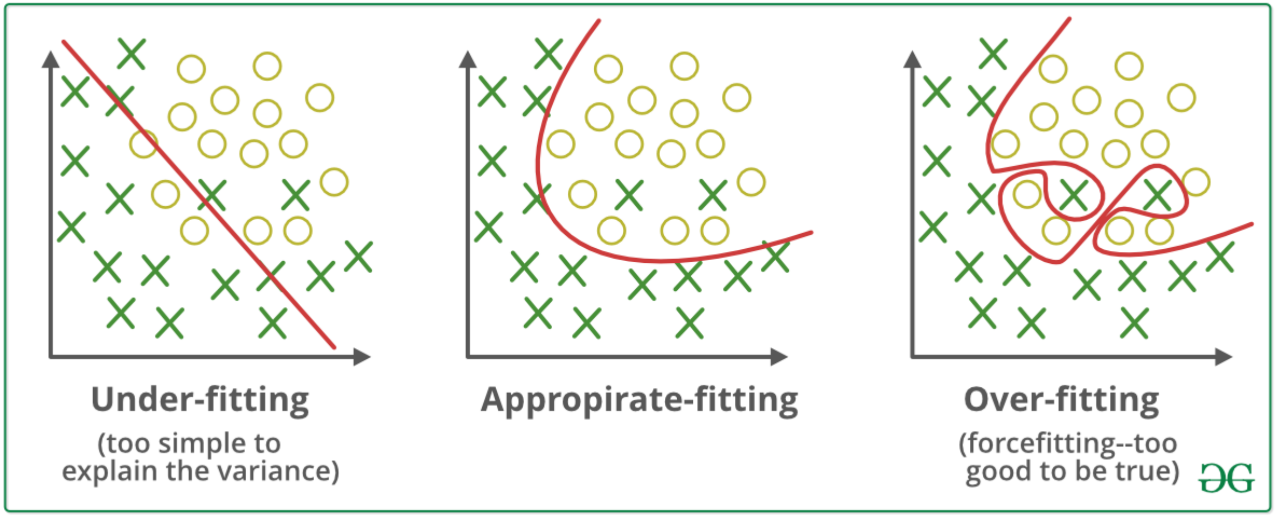


Overfitting

- Ex: polynomial regression



- Ex: binary classification



Overfitting

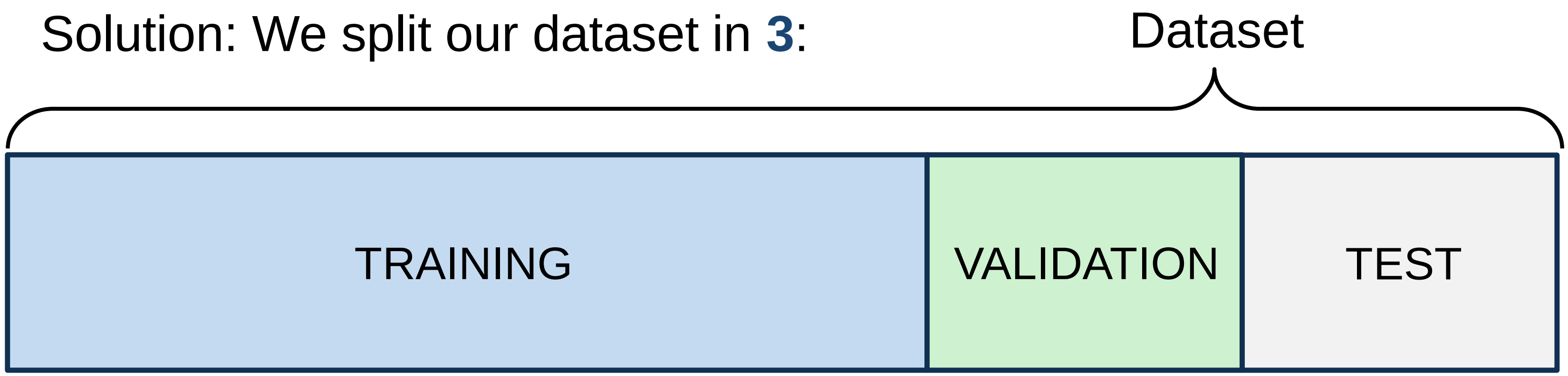
- In supervised learning, we are **not only interested** in a model that works perfectly on our training set. **We already have the answers anyway**, by definition of a (supervised) training set !

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

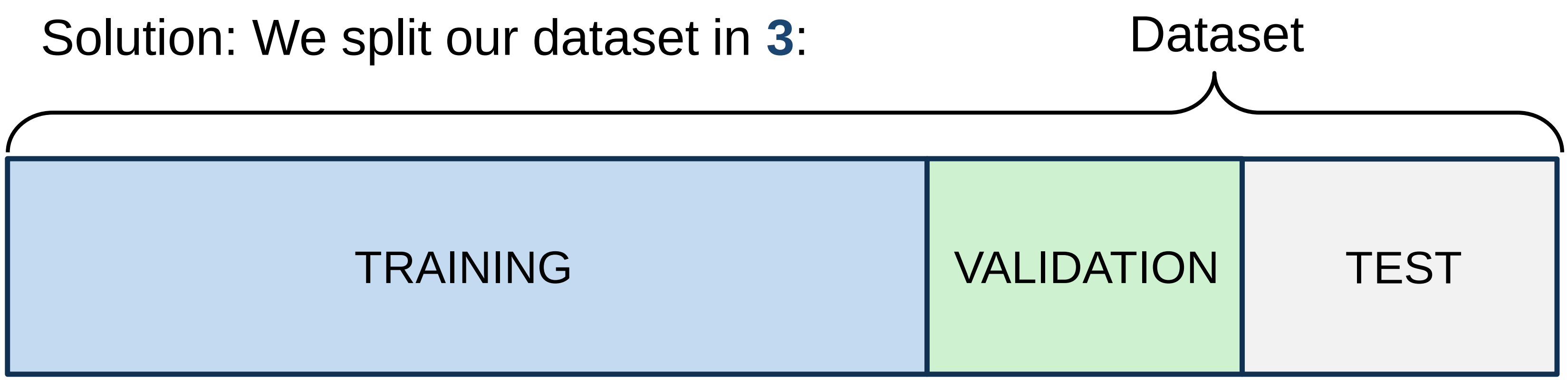
Overfitting

- 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**:



Overfitting

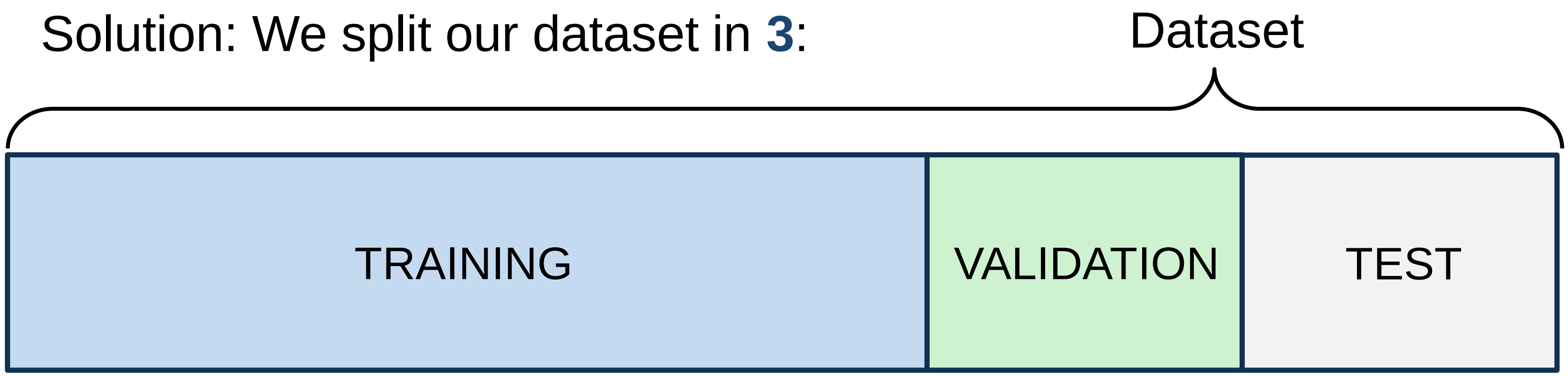
- In supervised learning, we are **not only interested** in a model that works perfectly on our training set. **We already have the answers anyway**, by definition of a (supervised) training set !
- We want a model that **generalizes** to **unseen** data
- Solution: We split our dataset in **3**:



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

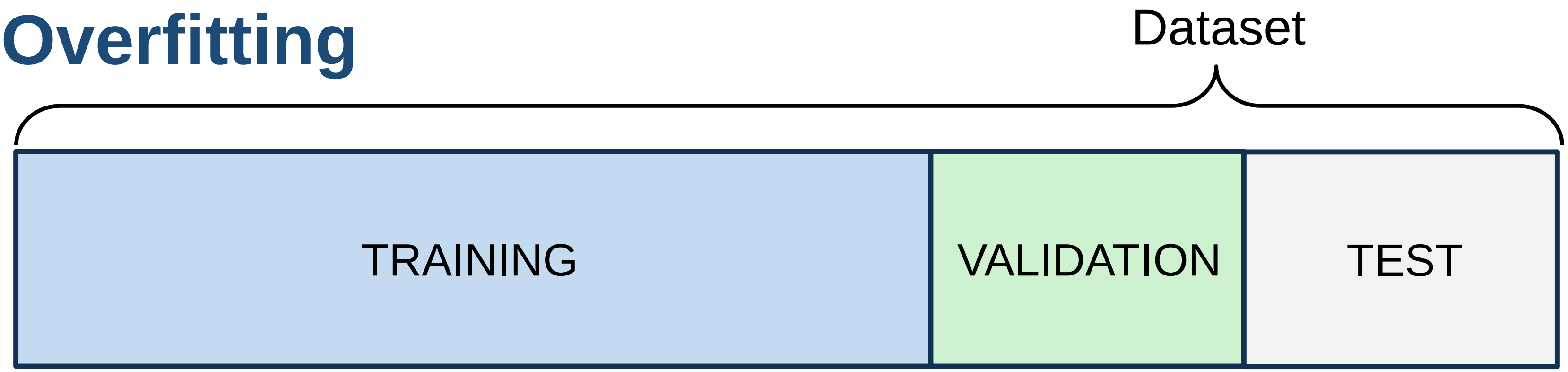
Overfitting

- 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**:

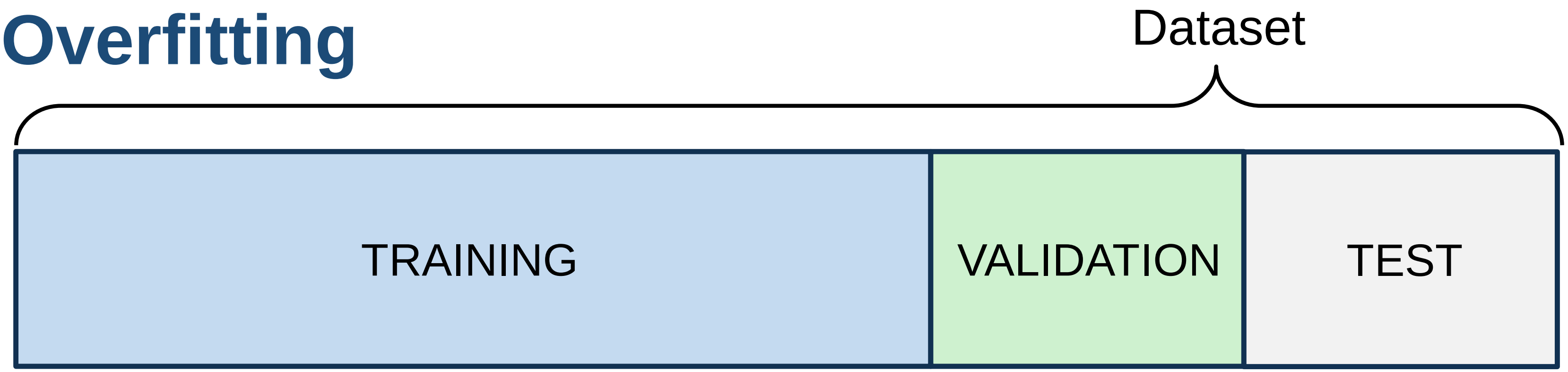


- These 3 subsets must be **perfectly disjoint** and all **representative** of the data.
- To achieve this, the split is done **at random**.

Overfitting

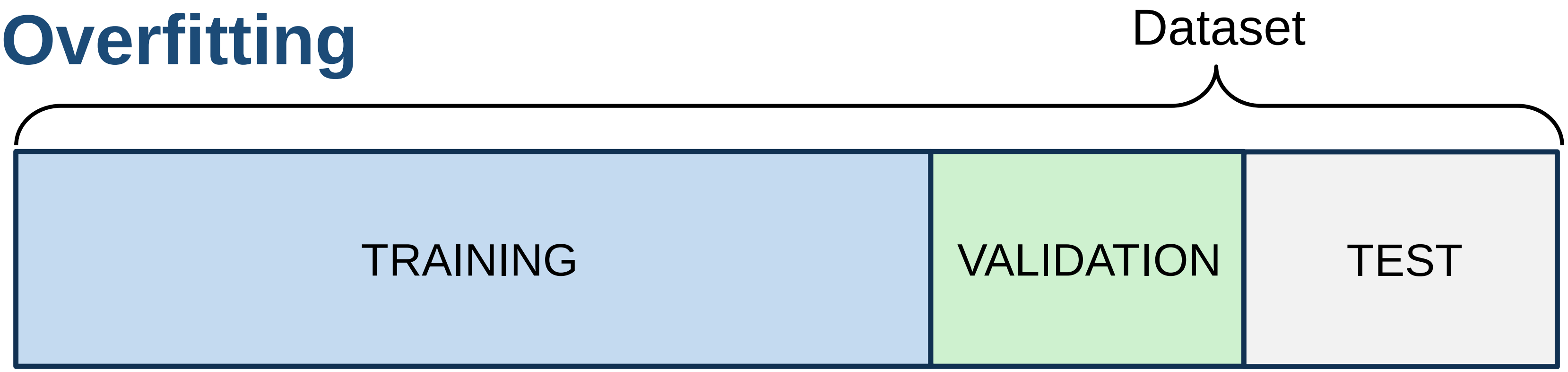


Overfitting



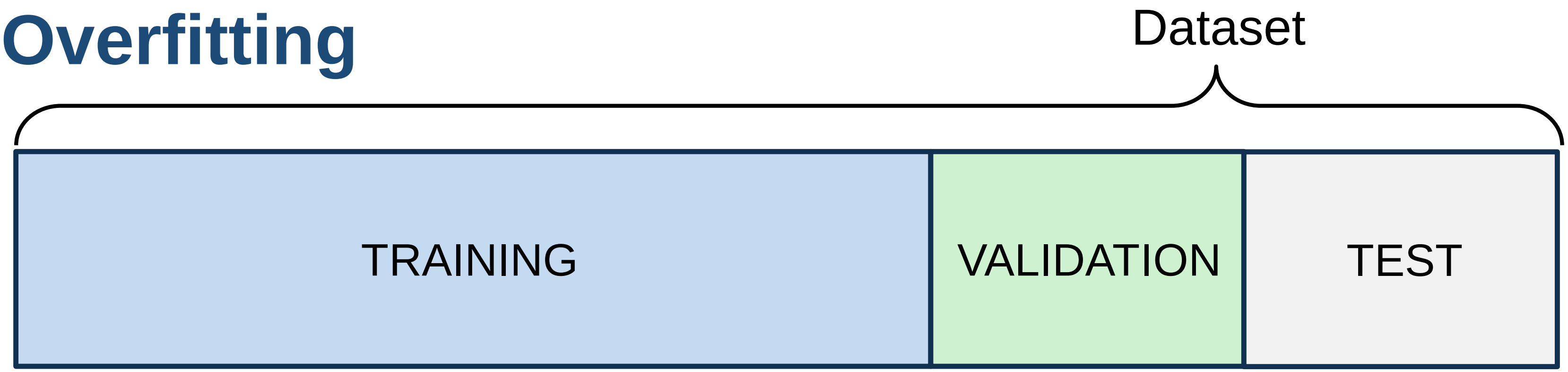
- This separation is **absolutely essential** for any supervised machine learning algorithm to reliably work

Overfitting



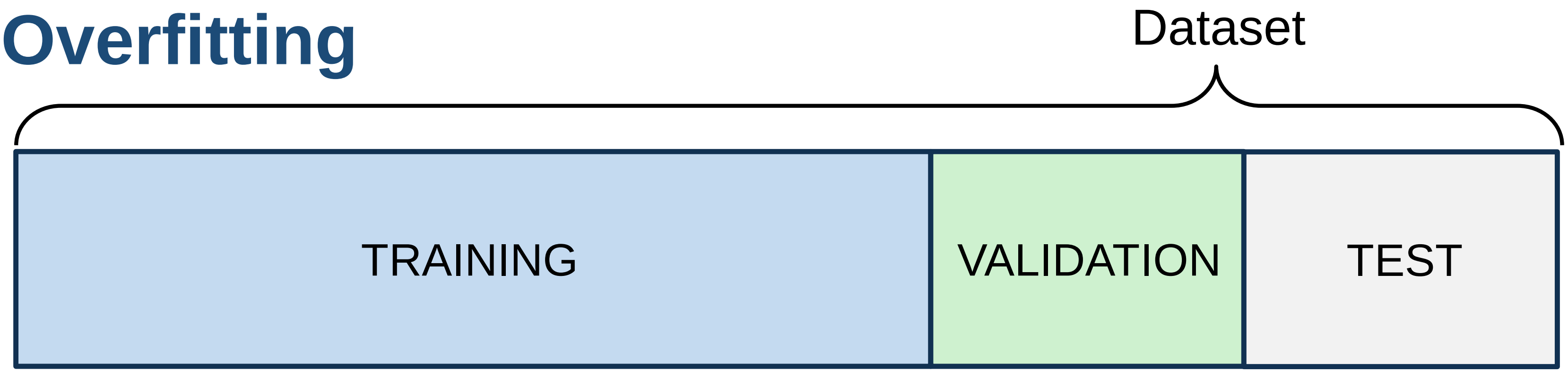
- This separation is **absolutely essential** for any supervised machine learning algorithm to reliably work
- The model parameters are only optimized over the **training set**

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

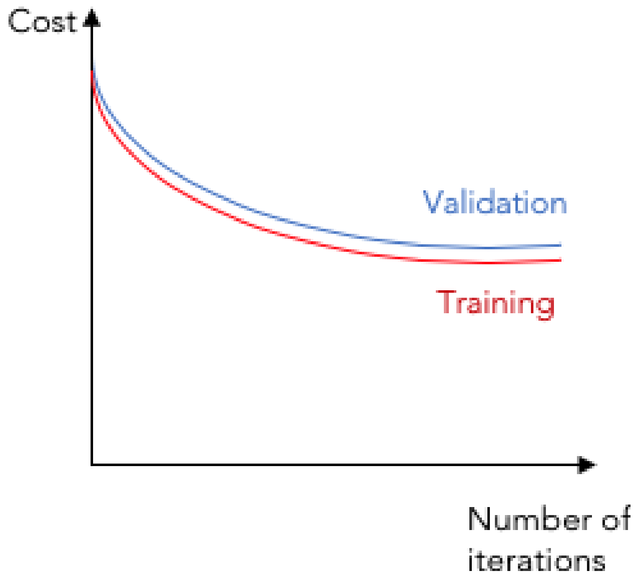
Overfitting

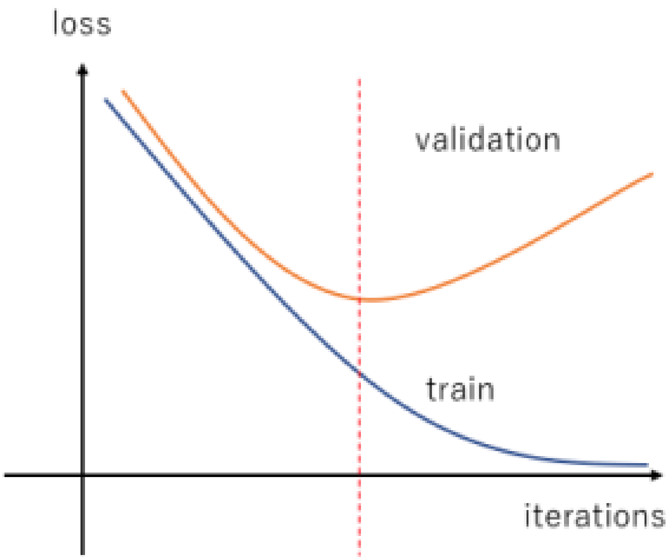


- This separation is **absolutely essential** for any supervised machine learning algorithm to reliably work
- The model parameters are only optimized over the **training set**
- We only use the validation set to:
  - At each training step, verify that the model is making progress on that set (possibly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

Overfitting vs. 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 ?

Quizz

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

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 ?

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 ?

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 ?

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 ?

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

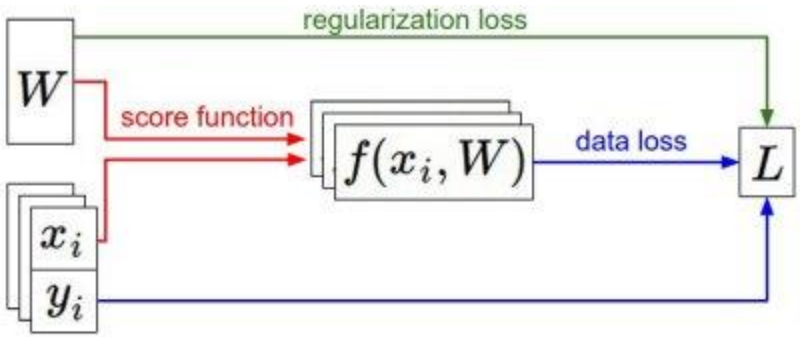
1) Regularization

Techniques to reduce overfitting

1) Regularization

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

$$L(\mathrm{dnn}_{\boldsymbol{\theta}}, \mathcal{T}) = \frac{1}{T}\sum_{t=1}^{T} \ell\left(\mathrm{dnn}_{\boldsymbol{\theta}}(\boldsymbol{x}_t), \boldsymbol{y}_t\right) + \lambda\mathcal{R}(\boldsymbol{\theta})$$

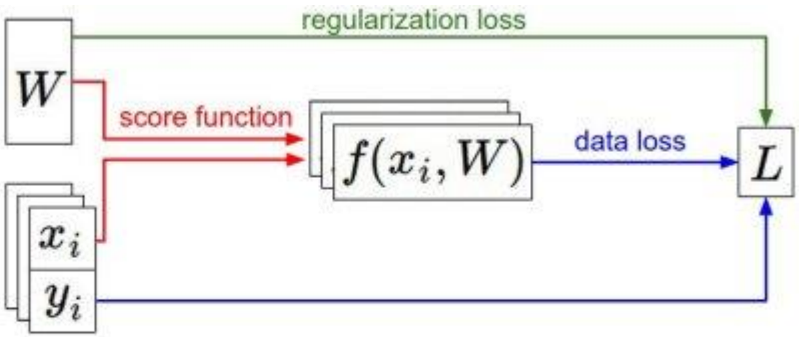


Techniques to reduce overfitting

1) Regularization

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

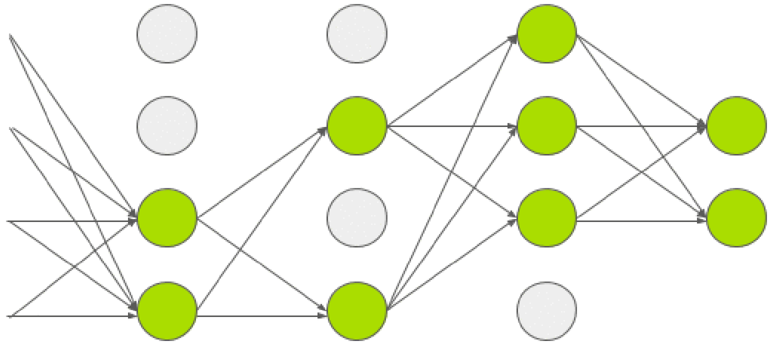
$$L(\mathrm{dnn}_{\boldsymbol{\theta}}, \mathcal{T}) = \frac{1}{T}\sum_{t=1}^{T} \ell\left(\mathrm{dnn}_{\boldsymbol{\theta}}(\boldsymbol{x}_t), \boldsymbol{y}_t\right) + \lambda \mathcal{R}(\boldsymbol{\theta})$$



- 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

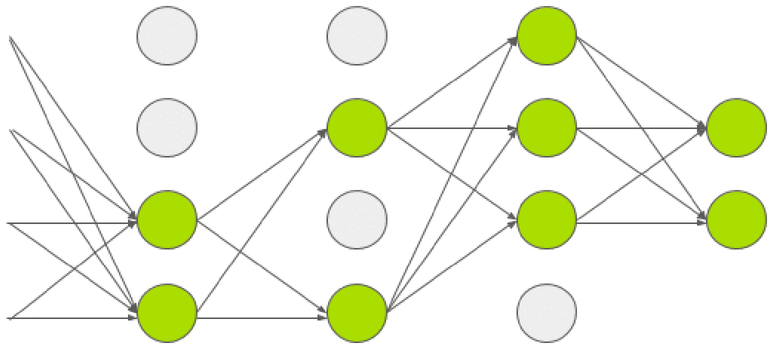
2) Dropout



- Idea: “cripple” the neural network by removing hidden units stochastically

Techniques to reduce overfitting

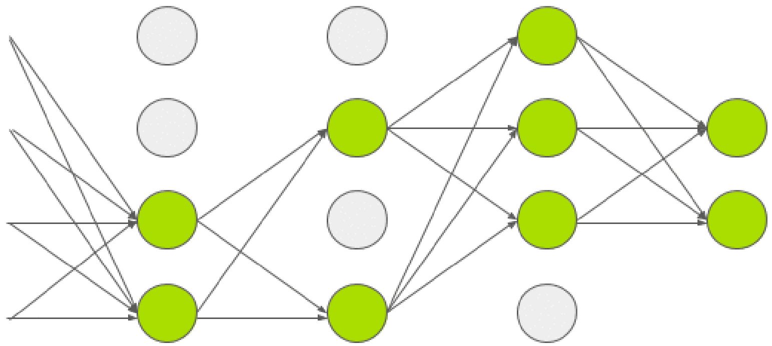
2) Dropout



- 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

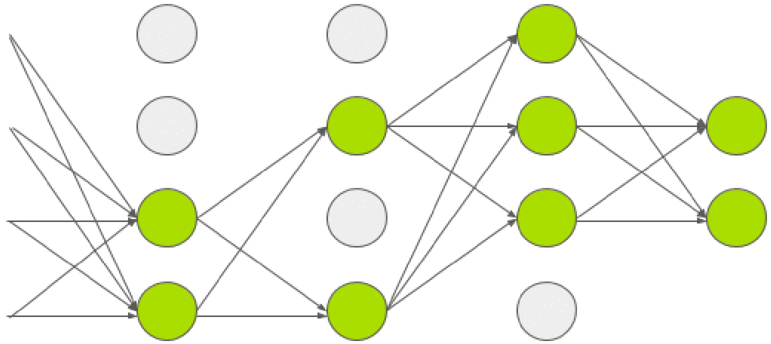
2) Dropout



- 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

Techniques to reduce overfitting

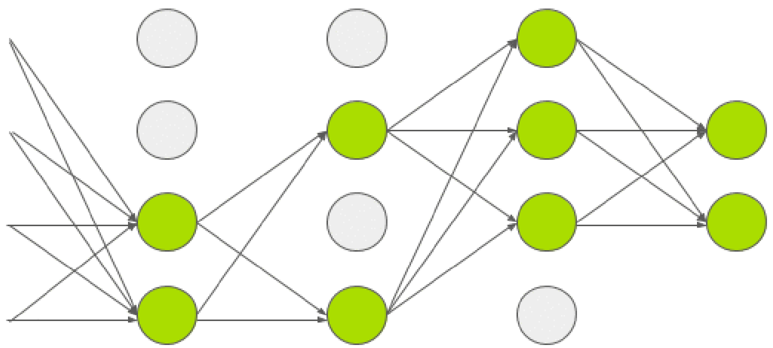
2) Dropout



- 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

Techniques to reduce overfitting

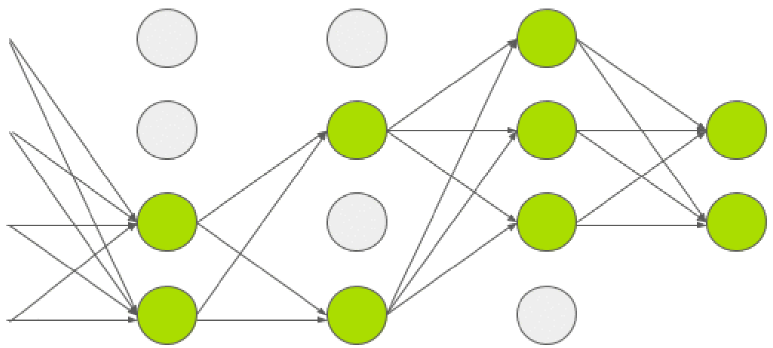
2) Dropout



- 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

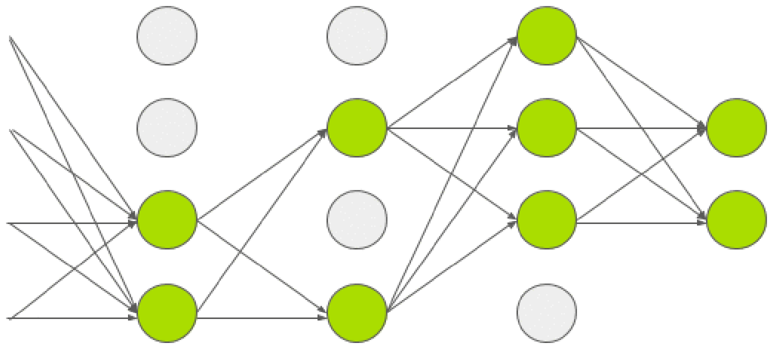
2) Dropout



- 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

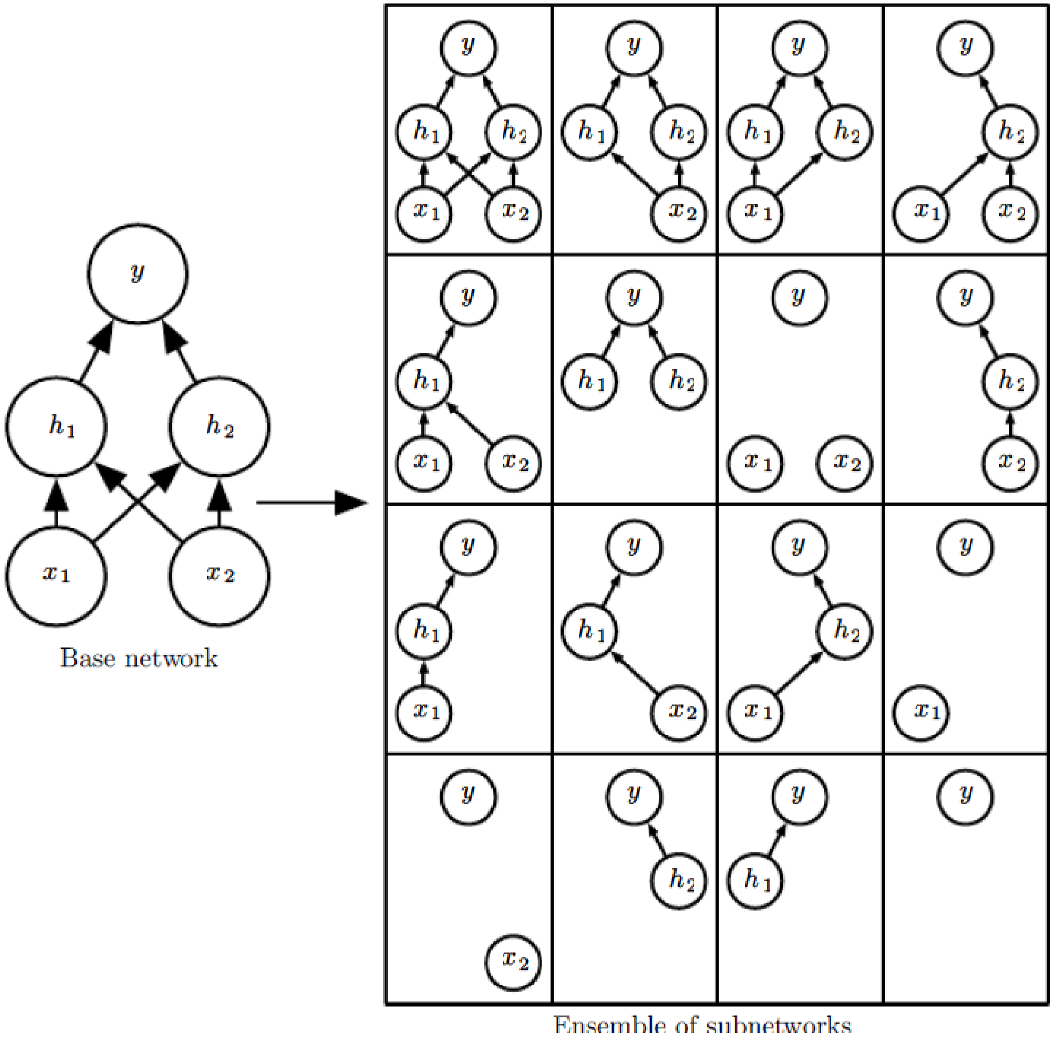
2) Dropout



- 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.

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

Techniques to reduce overfitting

3) Data Augmentation