UNIVERSITÀ
DEGLI STUDI
DI BERGAMO

Dipartimento
di Ingegneria Gestionale,
dell'Informazione e della Produzione

Lesson 9.

Neural networks

DATA SCIENCE AND AUTOMATION COURSE

MASTER DEGREE SMART TECHNOLOGY ENGINEERING

TEACHER
Mirko Mazzoleni

PLACE
University of Bergamo

Outline

1. Neural networks: basic concepts
2. Neural networks: learning
3. Tips & tricks
4. Deep learning

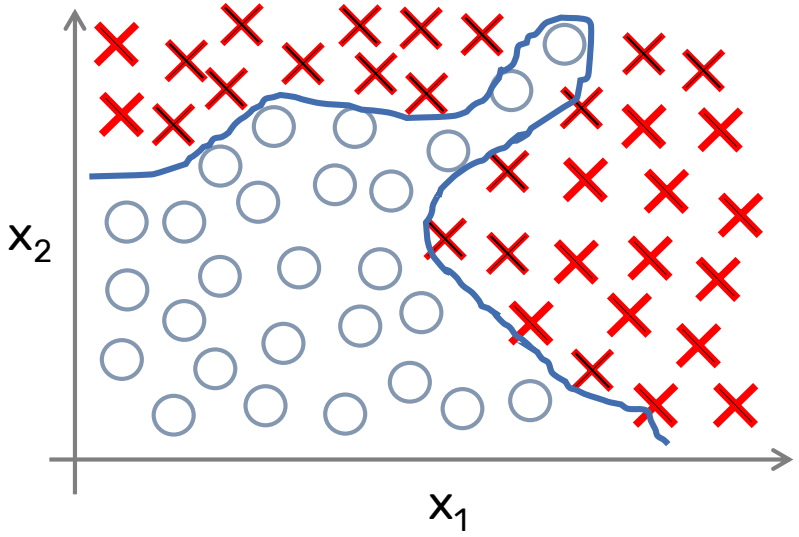
Outline

1. **Neural networks: basic concepts**
2. Neural networks: learning
3. Tips & tricks
4. Deep learning

Nonlinear hypothesis

Linear methods can be used to **learn nonlinear function**

- the important thing is that the model is **linear in the parameters**



Logistic regression

$$h_{\boldsymbol{\theta}}(\boldsymbol{x}) = s(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 + \theta_4 x_1^2 x_2 + \theta_5 x_1^3 x_2 + \cdots)$$

- Logistic regression in **OK if we have a 2-dim feature space** (x_1, x_2), so $q = d - 1 = 2$. We can compute all the quadratic terms $O(q^2) \approx \frac{q^2}{2}$
- If we have 100 features, we end up with about ≈ 5000 features
- This is **not a good way** to learn nonlinear functions

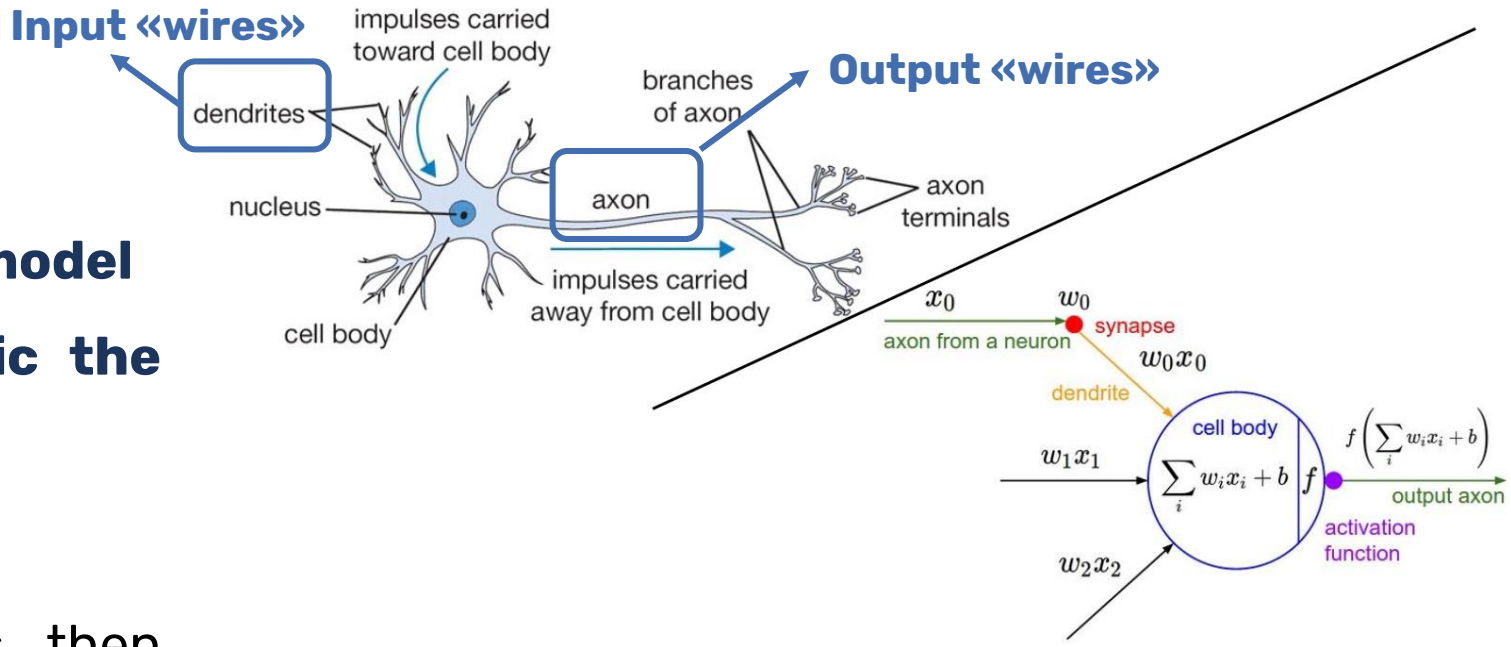
Neural networks

A Neural Network (NN) is a **nonlinear model**

- NN originate from trying to **mimic the brain**

- Very widely used in 80s and 90s, then their popularity diminished in late 90s

- Today they are **state-of-the-art** technique for many applications (vision, audio, text)



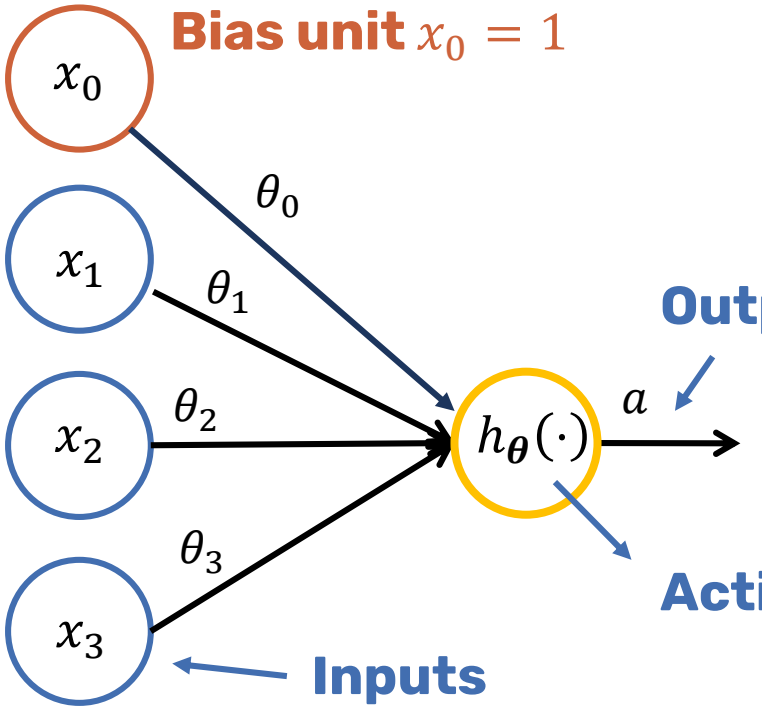
- If a neuron wants to send a message, it sends an electric impulse to other neurons

- Which, in turn, do some computation with the received input to send its messages

Neural networks

Neuron model: **logistic unit**

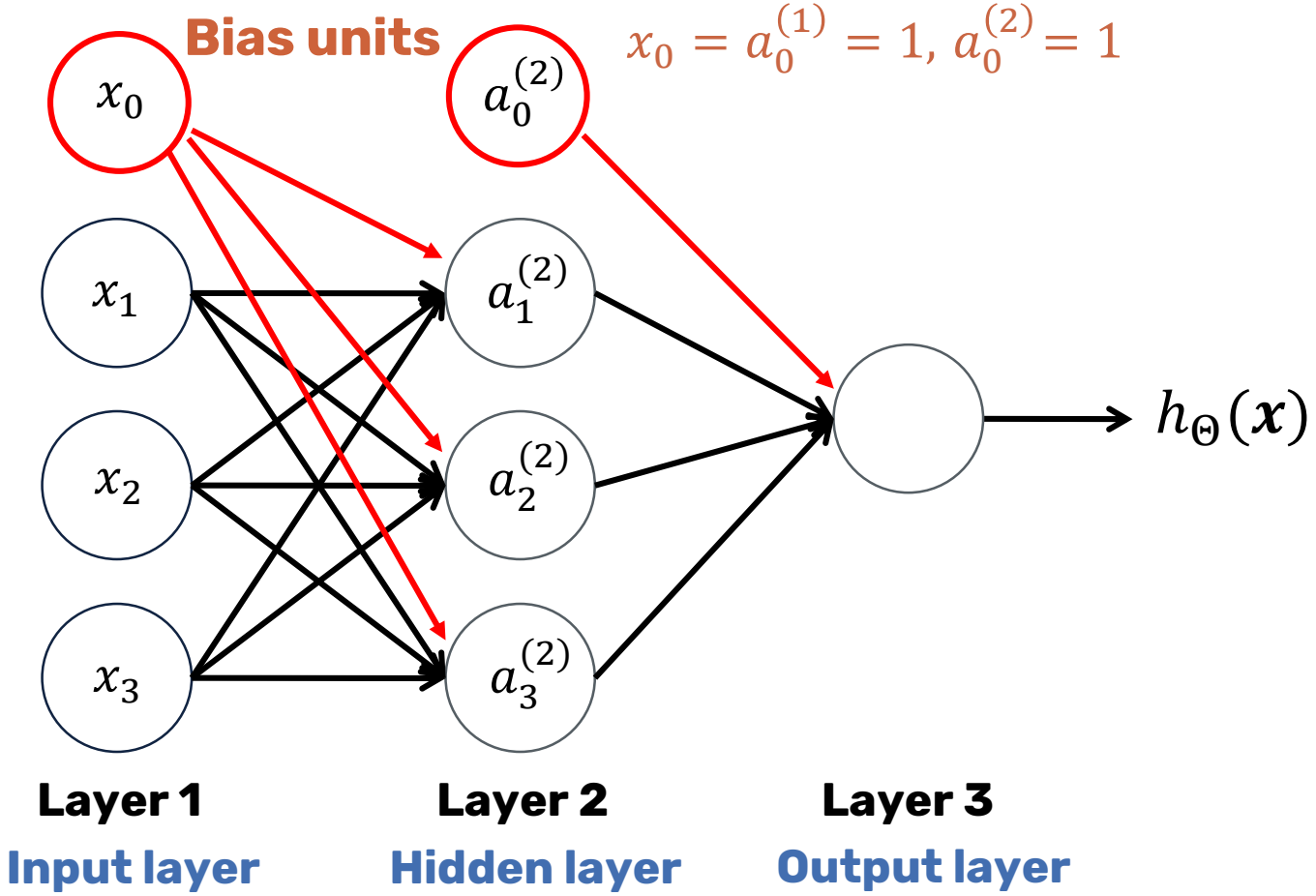
$$\boldsymbol{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad \boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix} \qquad a\text{: activation}$$



$$h_{\boldsymbol{\theta}}(\boldsymbol{x}) = s(\boldsymbol{x}^T\boldsymbol{\theta}) = \frac{1}{1 + e^{-\boldsymbol{x}^T\boldsymbol{\theta}}} = a$$

Neural networks

Neural networks can be seen as **combination of logistic units**

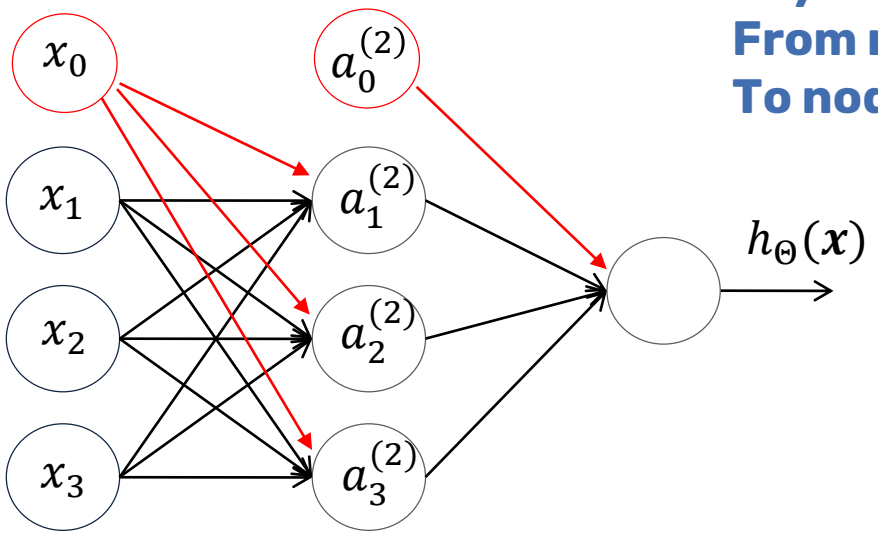


- Values in the **hidden layer** are not observed in the training set
- The network computes its **own features**

$$h_\Theta(x) = s\left(\theta_0 a_0^{(2)} + \theta_1 a_1^{(2)} + \theta_2 a_2^{(2)} + \theta_3 a_3^{(2)}\right)$$

- It is like logistic regression, but it uses $a_0^{(2)}, \dots, a_3^{(2)}$ instead of $x_0, \dots, x_3$

Neural networks



3 units *(do not consider bias units)*

3 units *(do not consider bias units)*

Layer 1
From node 0
To node 1

$a_i^{(l)}$ = "activation" of unit i in layer l

$\Theta^{(l)}$ = matrix of weights controlling function mapping from layer l to layer $l+1$

$$a_1^{(2)} = s\left(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3\right)$$

$$a_2^{(2)} = s\left(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3\right)$$

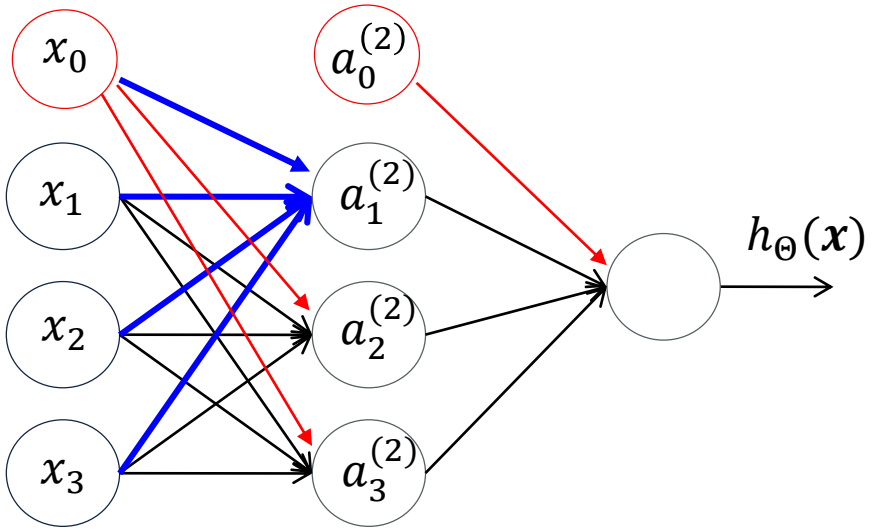
$$a_3^{(2)} = s\left(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3\right)$$

$$h_\Theta(\boldsymbol{x}) = a_1^{(3)} = s\left(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)}\right)$$

If network has w_l units in layer l, w_{l+1} units in layer $l+1$, then $\Theta^{(j)}$ will be of dimension $w_{l+1} \times (w_l + 1)$

In our example: $\Theta^{(1)} \in \mathbb{R}^{3\times4}$ and $\Theta^{(2)} \in \mathbb{R}^{1\times4}$

Neural networks



3 units *(do not consider bias units)*

3 units *(do not consider bias units)*

$a_i^{(l)}$ = "activation" of unit i in layer l

$\Theta^{(l)}$ = matrix of weights controlling function mapping from layer l to layer $l+1$

$$a_1^{(2)} = s\left(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3\right)$$

$$a_2^{(2)} = s\left(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3\right)$$

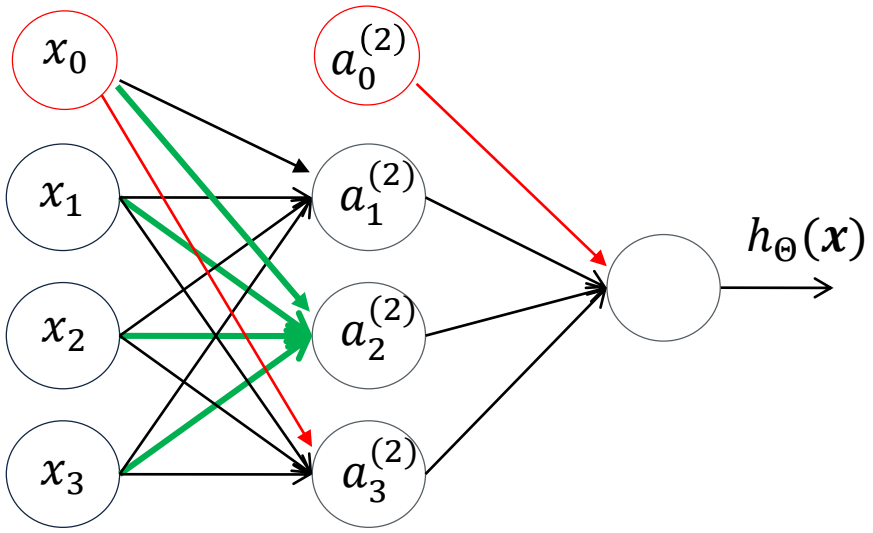
$$a_3^{(2)} = s\left(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3\right)$$

$$h_\Theta(\boldsymbol{x}) = a_1^{(3)} = s\left(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)}\right)$$

If network has w_l units in layer l, w_{l+1} units in layer $l+1$, then $\Theta^{(j)}$ will be of dimension $w_{l+1} \times (w_l + 1)$

In our example: $\Theta^{(1)} \in \mathbb{R}^{3\times 4}$ and $\Theta^{(2)} \in \mathbb{R}^{1\times 4}$

Neural networks



3 units *(do not consider bias units)*

3 units *(do not consider bias units)*

$a_i^{(l)}$ = "activation" of unit i in layer l

$\Theta^{(l)}$ = matrix of weights controlling function mapping from layer l to layer $l+1$

$$a_1^{(2)} = s\left(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3\right)$$

$$a_2^{(2)} = s\left(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3\right)$$

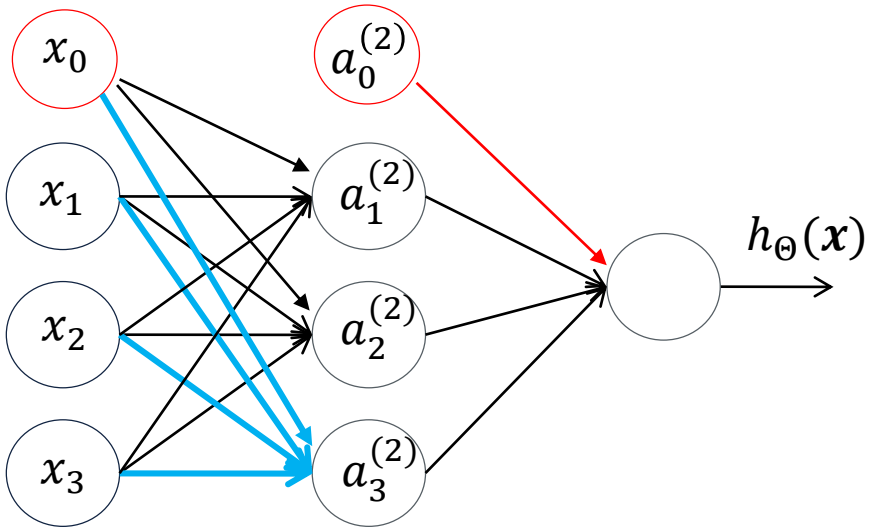
$$a_3^{(2)} = s\left(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3\right)$$

$$h_\Theta(\boldsymbol{x}) = a_1^{(3)} = s\left(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)}\right)$$

If network has w_l units in layer l, w_{l+1} units in layer $l+1$, then $\Theta^{(j)}$ will be of dimension $w_{l+1} \times (w_l + 1)$

In our example: $\Theta^{(1)} \in \mathbb{R}^{3\times 4}$ and $\Theta^{(2)} \in \mathbb{R}^{1\times 4}$

Neural networks



3 units *(do not consider bias units)*

3 units *(do not consider bias units)*

$a_i^{(l)}$ = "activation" of unit i in layer l

$\Theta^{(l)}$ = matrix of weights controlling function mapping from layer l to layer $l+1$

$$a_1^{(2)} = s\left(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3\right)$$

$$a_2^{(2)} = s\left(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3\right)$$

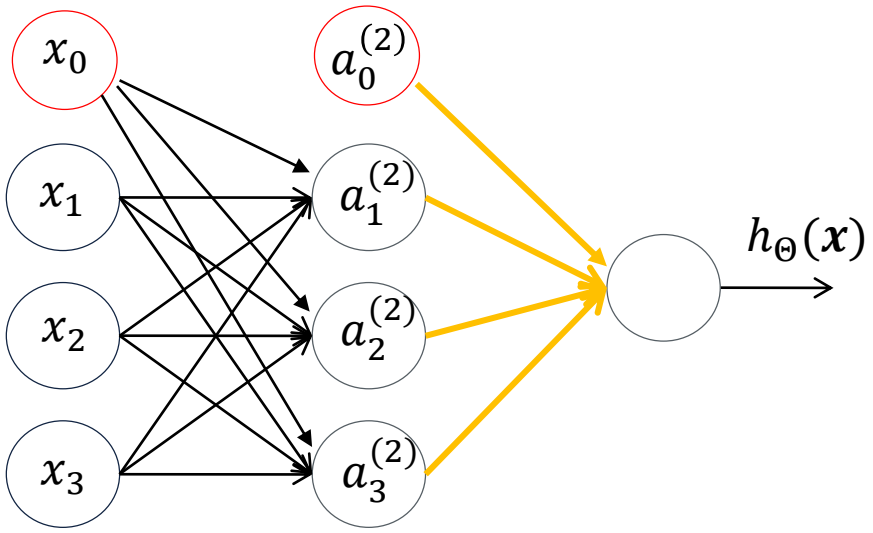
$$a_3^{(2)} = s\left(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3\right)$$

$$h_\Theta(\boldsymbol{x}) = a_1^{(3)} = s\left(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)}\right)$$

If network has w_l units in layer l, w_{l+1} units in layer $l+1$, then $\Theta^{(j)}$ will be of dimension $w_{l+1} \times (w_l + 1)$

In our example: $\Theta^{(1)} \in \mathbb{R}^{3\times 4}$ and $\Theta^{(2)} \in \mathbb{R}^{1\times 4}$

Neural networks



3 units *(do not consider bias units)*

3 units *(do not consider bias units)*

$a_i^{(l)}$ = "activation" of unit i in layer l

$\Theta^{(l)}$ = matrix of weights controlling function mapping from layer l to layer $l+1$

$$a_1^{(2)} = s\left(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3\right)$$

$$a_2^{(2)} = s\left(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3\right)$$

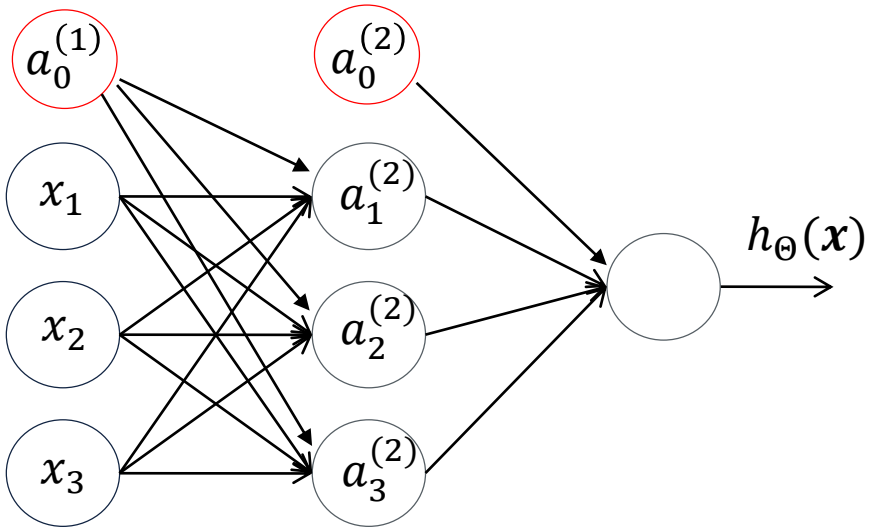
$$a_3^{(2)} = s\left(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3\right)$$

$$h_\Theta(\boldsymbol{x}) = a_1^{(3)} = s\left(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)}\right)$$

If network has w_l units in layer l, w_{l+1} units in layer $l+1$, then $\Theta^{(l)}$ will be of dimension $w_{l+1} \times (w_l + 1)$

In our example: $\Theta^{(1)} \in \mathbb{R}^{3\times 4}$ and $\Theta^{(2)} \in \mathbb{R}^{1\times 4}$

Forward computation



$$a_1^{(2)} = s\left(\overbrace{\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3}^{z_1^{(2)}}\right) \rightarrow a_1^{(2)} = s\left(z_1^{(2)}\right)$$

$$a_2^{(2)} = s\left(\overbrace{\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3}^{z_2^{(2)}}\right) \rightarrow a_2^{(2)} = s\left(z_2^{(2)}\right)$$

$$a_3^{(2)} = s\left(\overbrace{\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3}^{z_3^{(2)}}\right) \rightarrow a_3^{(2)} = s\left(z_3^{(2)}\right)$$

$$h_\Theta(\boldsymbol{x}) = a_1^{(3)} = s\left(\overbrace{\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)}}^{z_1^{(3)}}\right)$$

$$\boldsymbol{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad \boldsymbol{z}^{(2)} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{bmatrix} \quad \boldsymbol{a}^{(2)} = \begin{bmatrix} a_1^{(2)} \\ a_2^{(2)} \\ a_3^{(2)} \end{bmatrix}$$

$$\begin{bmatrix} \cdots \\ \ddots \\ \cdots \end{bmatrix} \begin{bmatrix} \vdots \end{bmatrix}$$

$$\boldsymbol{z}^{(2)} = \Theta^{(1)}\boldsymbol{x} \qquad \boldsymbol{z}^{(3)} = \Theta^{(2)}\boldsymbol{a}^{(2)}$$

$$\boldsymbol{a}^{(2)} = s\left(\boldsymbol{z}^{(2)}\right) \rightarrow \boldsymbol{a}^{(2)} \equiv \begin{bmatrix} a_0^{(2)} & \boldsymbol{a}^{(2)} \end{bmatrix}^T \qquad h_\Theta(\boldsymbol{x}) = \boldsymbol{a}^{(3)} = s\left(\boldsymbol{z}^{(3)}\right)$$

Outline

1. Neural networks: basic concepts

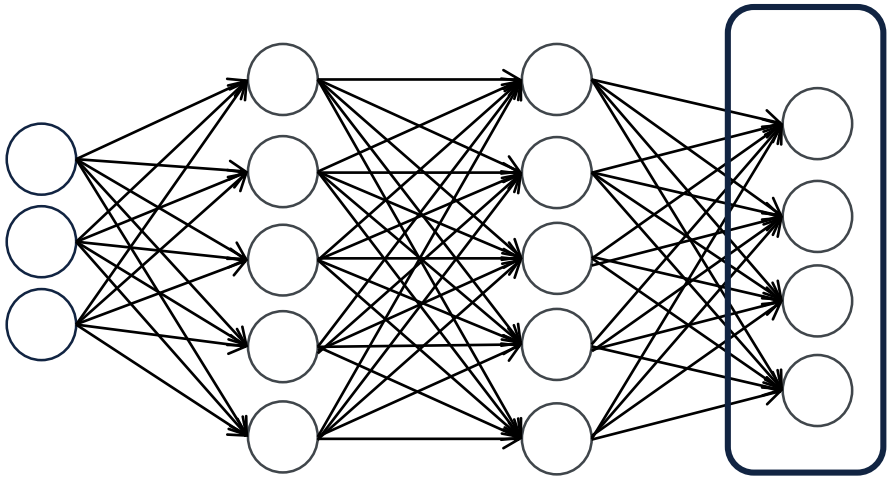
2. **Neural networks: learning**

3. Tips & tricks

4. Deep learning

Multi-class classification

- Neural networks can deal straightforwardly with **multiple outputs**
- Suppose we want to classify in **4 classes**
- Then, we want that the following to hold:



$h_\Theta(\boldsymbol{x}) \in \mathbb{R}^4$

When class 1

$$h_\Theta(\boldsymbol{x}) \approx \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

When class 2

$$h_\Theta(\boldsymbol{x}) \approx \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

When class 3

$$h_\Theta(\boldsymbol{x}) \approx \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

When class 4

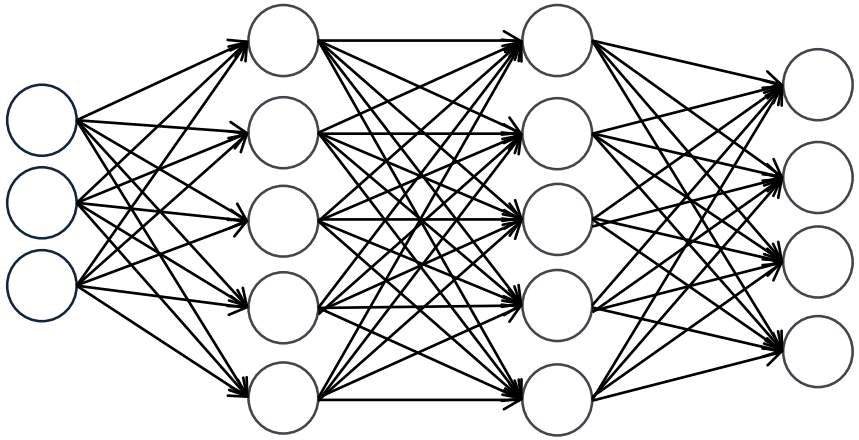
$$h_\Theta(\boldsymbol{x}) \approx \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

Neural networks for classification

$\mathcal{D} = \{(\boldsymbol{x}(1), y(1)), \dots, (\boldsymbol{x}(N), y(N))\}$

$L =$ total number of layers in network

$w_l =$ number of units (not counting bias unit) in layer l



Binary classification

$y = 0$ or $y = 1$

1 output unit

Multiclass classification (C classes)

$y \in \mathbb{R}^C$ $\quad \begin{bmatrix}1\\0\\0\\0\end{bmatrix} \quad \begin{bmatrix}0\\1\\0\\0\end{bmatrix} \quad \begin{bmatrix}0\\0\\1\\0\end{bmatrix} \quad \begin{bmatrix}0\\0\\0\\1\end{bmatrix}$

C output units

Cost function for classification

Additional regularization term

Logistic regression

$$J(\boldsymbol{\theta}) = -\sum_{i=1}^{N}\Big(y(i)\ln h_{\boldsymbol{\theta}}\big(\boldsymbol{x}(i)\big) + \big(1-y(i)\big)\ln\big[1-h_{\boldsymbol{\theta}}\big(\boldsymbol{x}(i)\big)\big]\Big) + \frac{\lambda}{2}\sum_{j=1}^{d-1}\theta_j^2$$

Neural network

$$h_\Theta(\boldsymbol{x}) \in \mathbb{R}^C \qquad \{h_\Theta(\boldsymbol{x})\}_c = c\text{-th output}$$

$$J(\Theta) = -\sum_{i=1}^{N}\sum_{c=1}^{C}\Big(\{y\}_c(i)\ln\{h_\Theta\big(\boldsymbol{x}(i)\big)\}_c + \big(1-\{y\}_c(i)\big)\ln\Big[1-\{h_\Theta\big(\boldsymbol{x}(i)\big)\}_c\Big]\Big) + \frac{\lambda}{2}\sum_{l=1}^{L-1}\sum_{i=1}^{w_l}\sum_{j=1}^{w_l+1}\Big(\Theta_{ji}^{(l)}\Big)^2$$

Gradient computation

Given the cost function $J(\Theta)$, we now have to minimize it. Need to compute:

- $J(\Theta)$: cost function
- $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$: gradient with respect to the network parameters (weights), $\Theta_{ij}^{(l)} \in \mathbb{R}$

Neural networks models use an algorithm called **Backpropagation**

Backpropagation is an algorithm to compute gradients. It relies on the **chain rule** of derivatives

Outline

1. Neural networks: basic concepts

2. Neural networks: learning

3. **Tips & tricks**

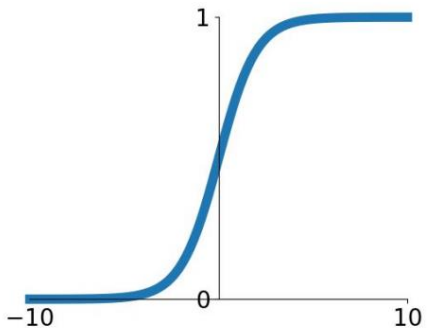
4. Deep learning

Activation functions

There are different (nonlinear) activation functions that can be used. The following can be used in the **hidden layers**. In the **output layer**, we used **sigmoid** for classification and a **linear** function for regression. In this case, the cost function can be the squared error.

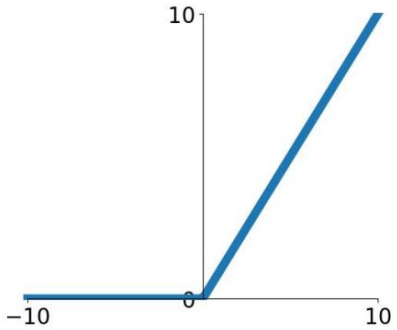
Sigmoid

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



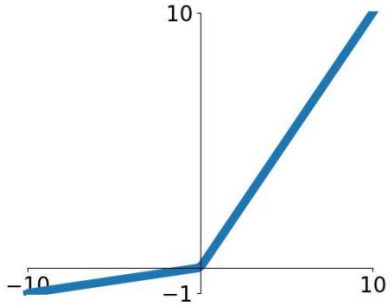
ReLU

$$y(x) = max(0, x)$$



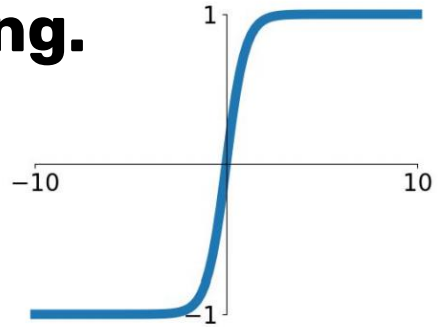
Leaky ReLU

$$y(x) = max(0.1x, x)$$



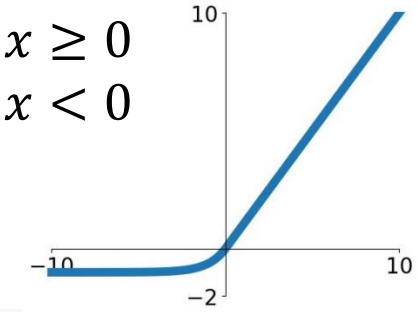
Hyperbolic tang.

$$y(x) = tanh(x)$$



ELU

$$y(x) = \begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Weights random initialization

It is not correct to inizialize all the parameters to zero. It turns out that in this way the network learns the same feature for each neuron, since $\frac{\partial J(\Theta)}{\partial \Theta_{ij}^{(l)}}$ is the same for each weight

- A proper initialization is mandatory to alleviate the **vanishing** or **exploding gradient** probems [13]

Common heuristics: generate weights from a Gaussian distribution of proper dimension, and then multiply for a number that depends on the network dimension and activation

ReLU

$$\Theta^{(l)} = \text{randn}(w_l, w_{l-1}) \cdot \sqrt{\frac{2}{w_{l-1}}}$$

ReLU units exibit less vanishing or exploding gradients problems

Tanh

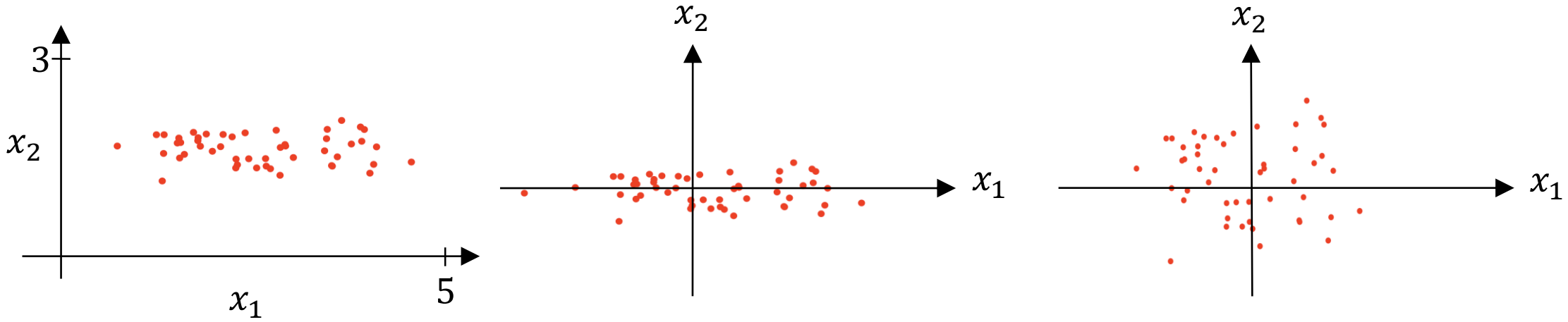
$$\Theta^{(l)} = \text{randn}(w_l, w_{l-1}) \cdot \sqrt{\frac{1}{w_{l-1}}}$$

$$\Theta^{(l)} = \text{randn}(w_l, w_{l-1}) \cdot \sqrt{\frac{2}{w_{l-1} + w_l}}$$

Xavier initialization

Input normalization

It is often useful to **normalize** the inputs to the neural net (and in general to all optimization algorithms) in order to lead to faster convergence of the gradient descent



Subtract mean **Normalize variance**

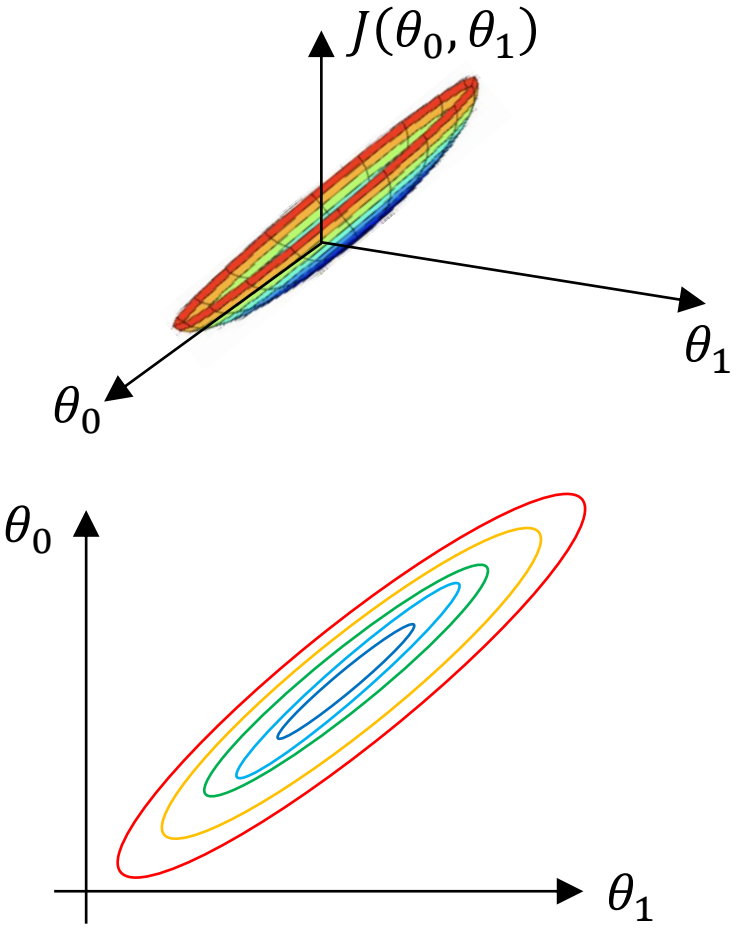
$$\hat{\mu}_j = \frac{1}{N}\sum_{i=1}^{N} x_j(i) \qquad \hat{\sigma}_j^2 = \frac{1}{N}\sum_{i=1}^{N}\left(x_j(i) - \mu_j\right)^2$$

$$x_j(i) = \frac{x_j(i) - \hat{\mu}_j}{\sqrt{\hat{\sigma}_j^2}}$$

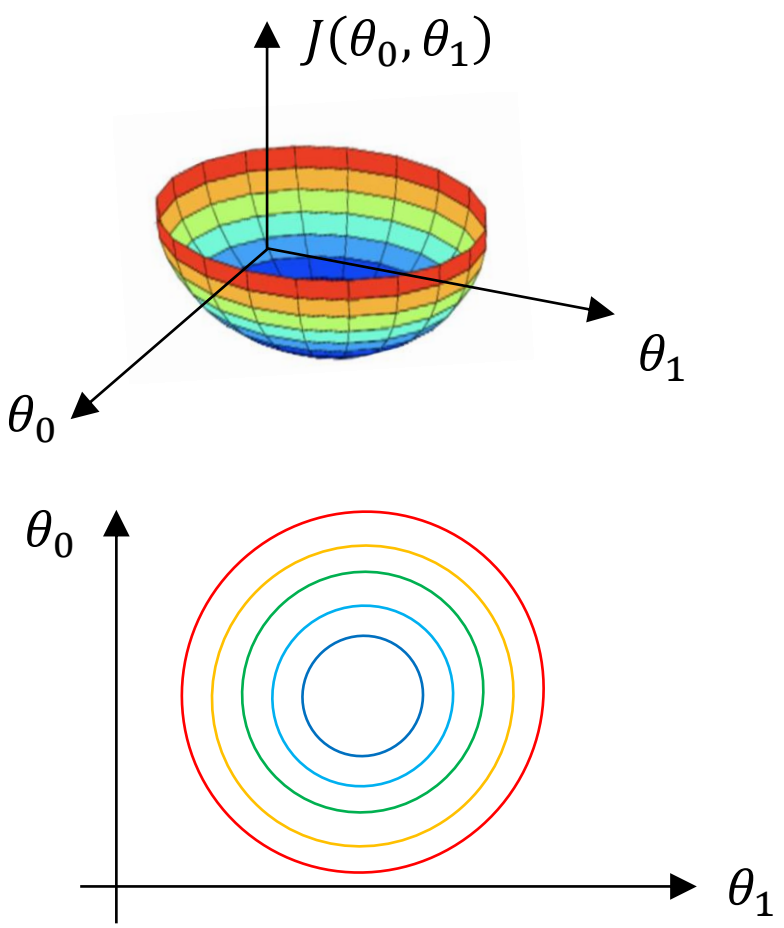
Normalize new data using THE SAME MEAN AND THE SAME VARIANCE of the training data

Input normalization

Unnormalized



Normalized



Mini batch gradient descent

Nowdays, neural networks are not trained by computing the full gradient, that requires all examples

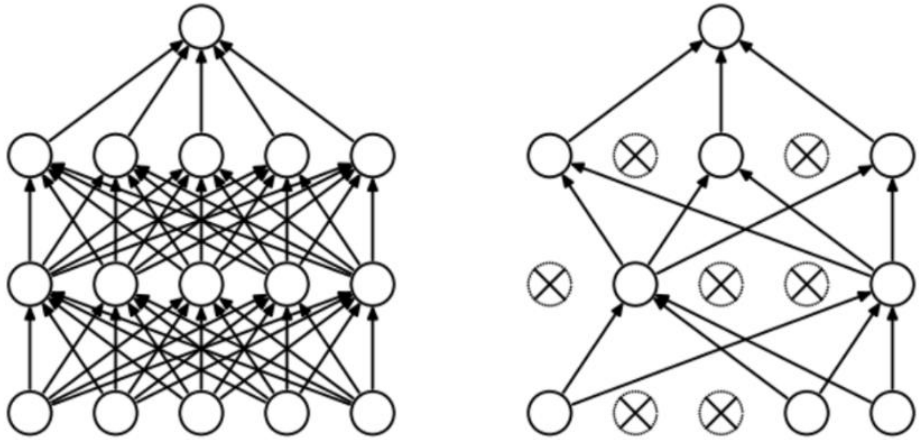
- The **stochastic gradient descent** does a gradient descent step each time a new data is available
  - ✓ This is both **faster** to compute and also leads to improved **regularization** properties. However, the gradient based on only one sample tends to be very noisy

- For this reason, modern implementations allow to specify a **batch** of data that will be used to compute the gradient. This is called **mini-batch gradient descent**

Dropout

Dropout is a **regularization technique** specifically designed for neural networks [12]

- It consists into (probabilistically) turn off (drop) random neurons of the network during the training phase
- These units are not considered during forward and backward passes
- It is like the are performing an averaging of less complex models



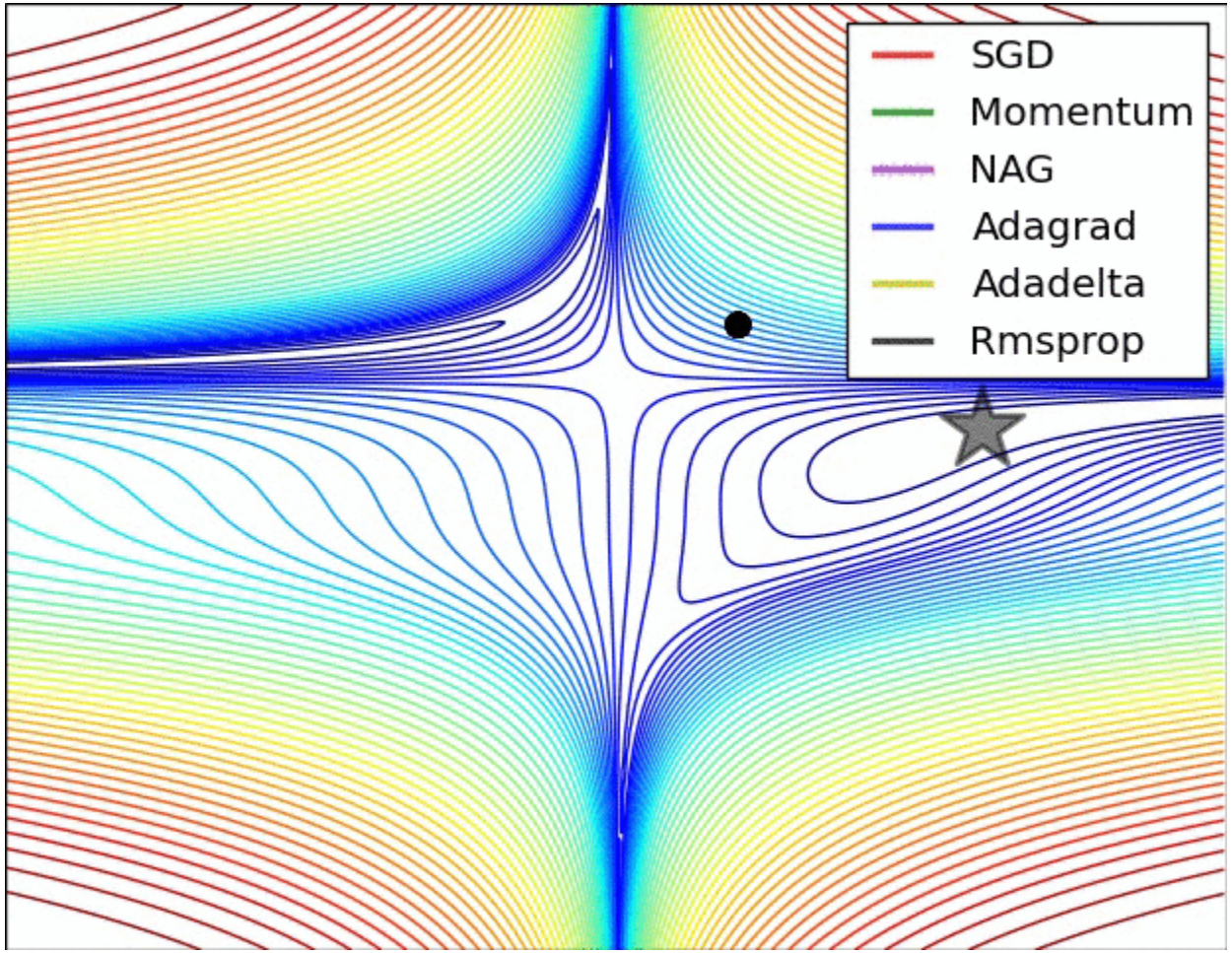
(a) Standard Neural Net (b) After applying dropout.

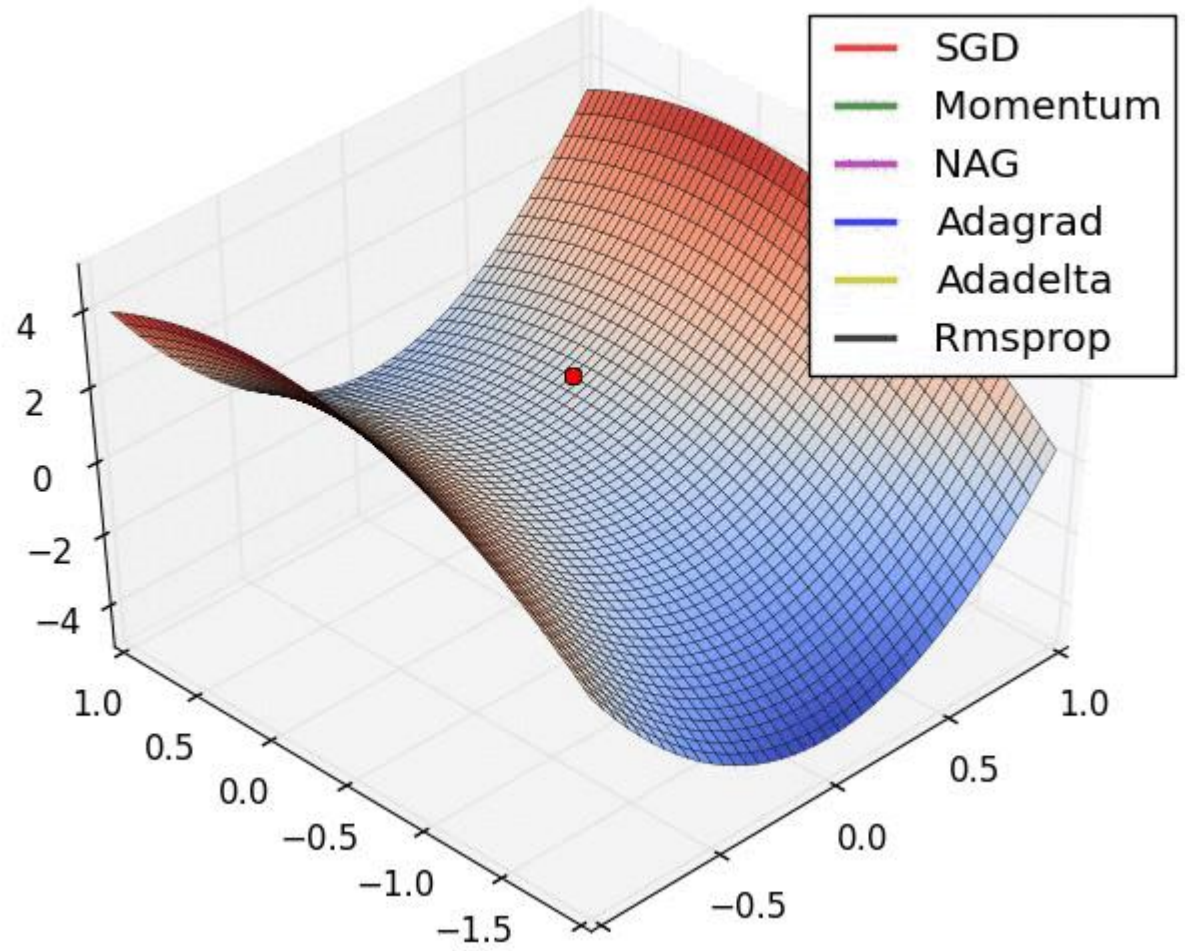
Dropout forces a neural network to learn more robust features that are useful in conjunction with many different random subsets of the other neurons.

Other tricks and heuristics

There exists different tricks that can be used to improve the learning of the neural net:

- **Batch normalization:** normalization step performed on every batch of data
- **Learning rate decay:** decrease the learning rate progressively as we reach towards the minimum
- **Advanced optimization schemes:** Momentum, RMSprop, ADAM





https://towardsdatascience.com/types-of-optimization-algorithms-used-in-neural-networks-and-ways-to-optimize-gradient-95ae5d39529f

Stochastic Gradient Descent (SGD) is the slowest

SGD gets stuck in saddle point

Outline

1. Neural networks: basic concepts

2. Neural networks: learning

3. Tips & tricks

4. **Deep learning**

Deep learning

The rise of available **data**, computational **power** and research **innovations** (such as the improvements in optimization just reviewed) allowed to train **more complex models**

- Neural networks with many layers and particular architectures were named **deep learning models**
- They became state-of-art in many tasks, most often for the more **unstructed** ones (computer vision, Natural Language Processing,…)
- In the next lessons we will talk about **Convolutional Neural Networks**, a particular type of deep architecture used for image analysis

Deep learning and bias-variance tradeoff

In principle, deep learning models require a **lot of data**, since they are very complex (remember the VC analysis)

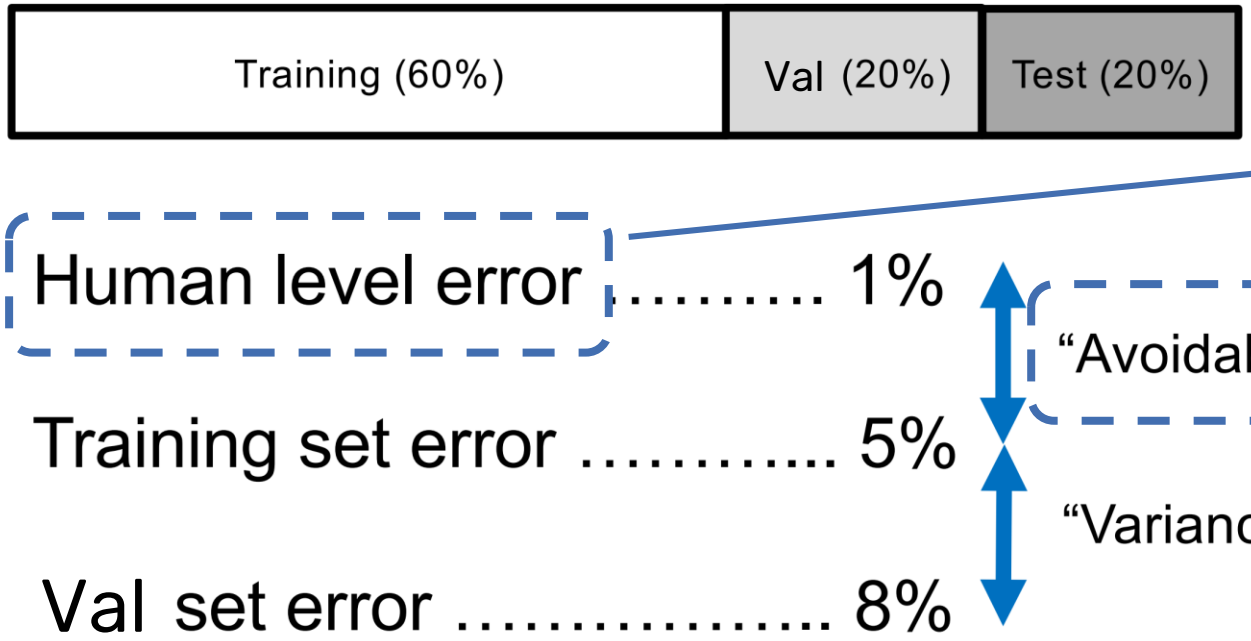
- Therefore, most deep learning projects have the train and test data that come **from different distributions** (and different applicative domain)

- On the first domain we have a lot of data and we train the model. But, the trained model needs to be **employed on the second** domain, for which we could have few examples

- The train/validation/test set paradigm needs to slightly change to reflect this **data mismatch problem** [14]

Traditional train/validation/test and bias-variance

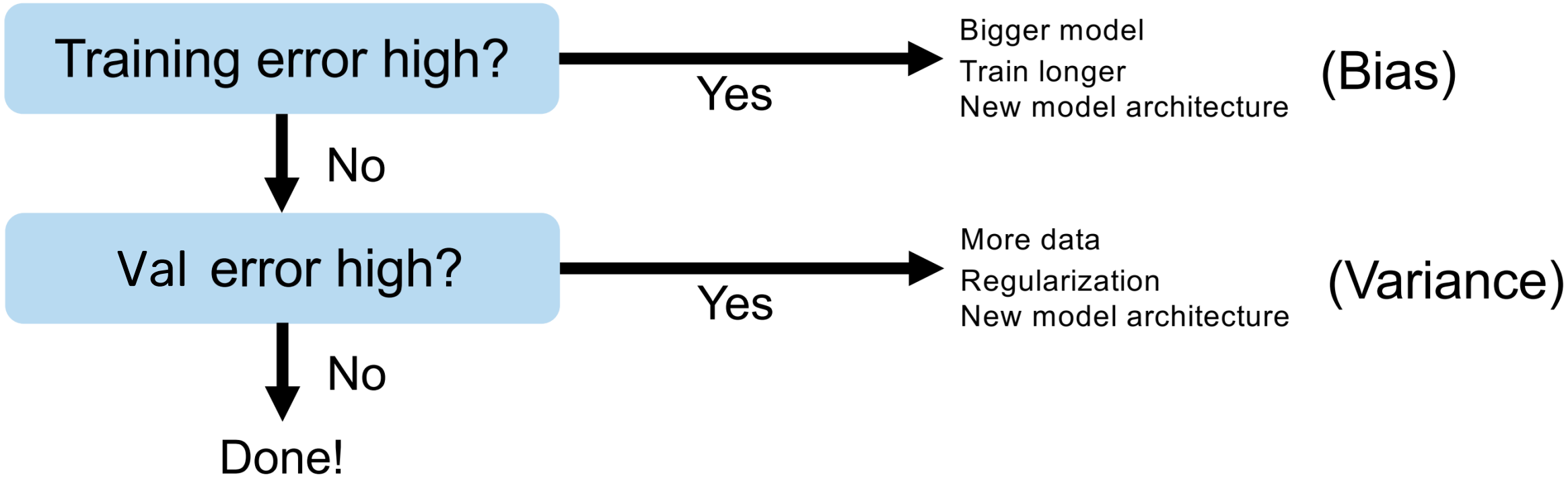
Say you want to build a human-level speech recognition system. You split your data into train/validation/test:



Human error is often an "upper bound" for performance: it is very difficult to do better than human (at least for image classification problems)

By knowing what the human-level performance is, it is possible to tell when a training set is performing well, too well or not well.

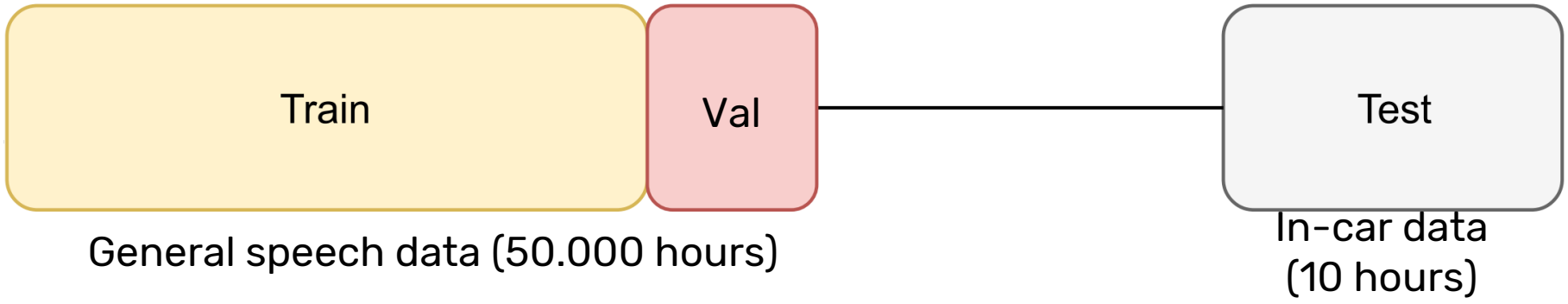
Basic recipe for machine learning



Different training and test set distributions

Say you want to build a **speech recognition** system for a new **in-car** rearview mirror product. You have 50.000 hours of general speech data, and 10 hours of in-car data. How do you split your data?

Bad way

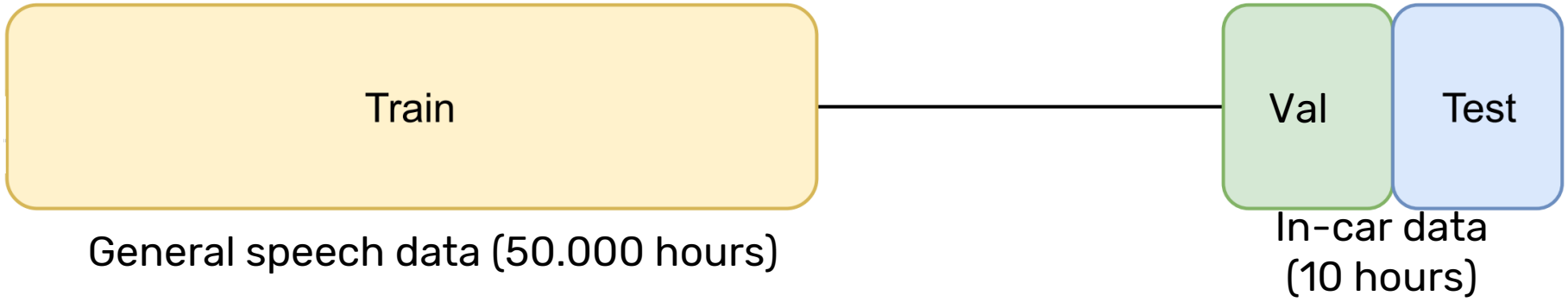


Having **mismatched val and test** distributions is not a good idea. Your team may spend months optimizing for dev set performance only to find it doesn't work well on the test set

Different training and test set distributions

Say you want to build a **speech recognition** system for a new **in-car** rearview mirror product. You have 50.000 hours of general speech data, and 10 hours of in-car data. How do you split your data?

Better way

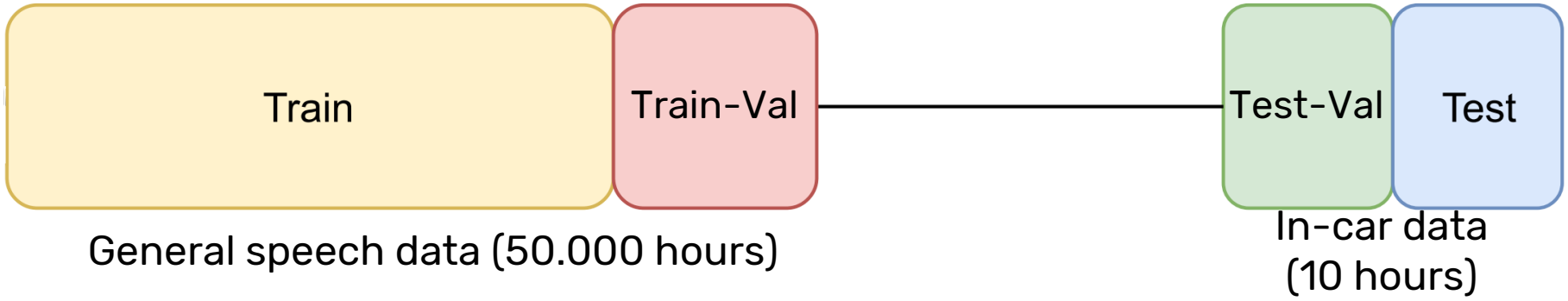


Hence, a smarter way of splitting the above dataset would be just like the second line of the diagram. In this way, the validation and test set are from the **same domain**

Different training and test set distributions

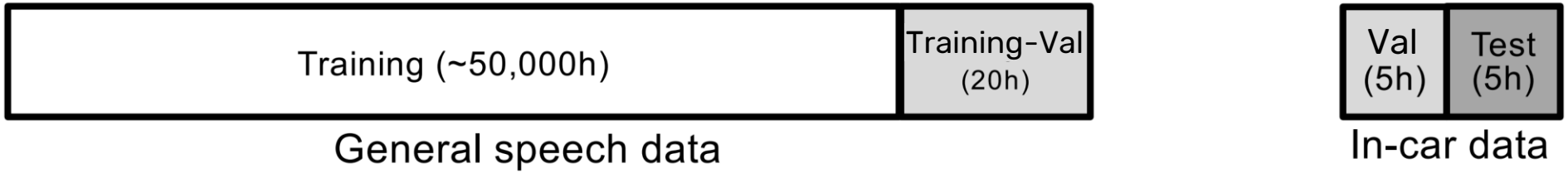
Say you want to build a **speech recognition** system for a new **in-car** rearview mirror product. You have 50.000 hours of general speech data, and 10 hours of in-car data. How do you split your data?

Best way



Create validation sets from **both data distributions**: a train-val and test-val set. In this way, any gap between the different errors can help you tackle the problem more clearly

Different training and test set distributions



Human level error 1%

Training error 1.1%

Training-val error 1.5%

Val set error 8%

Test set error 8.5%

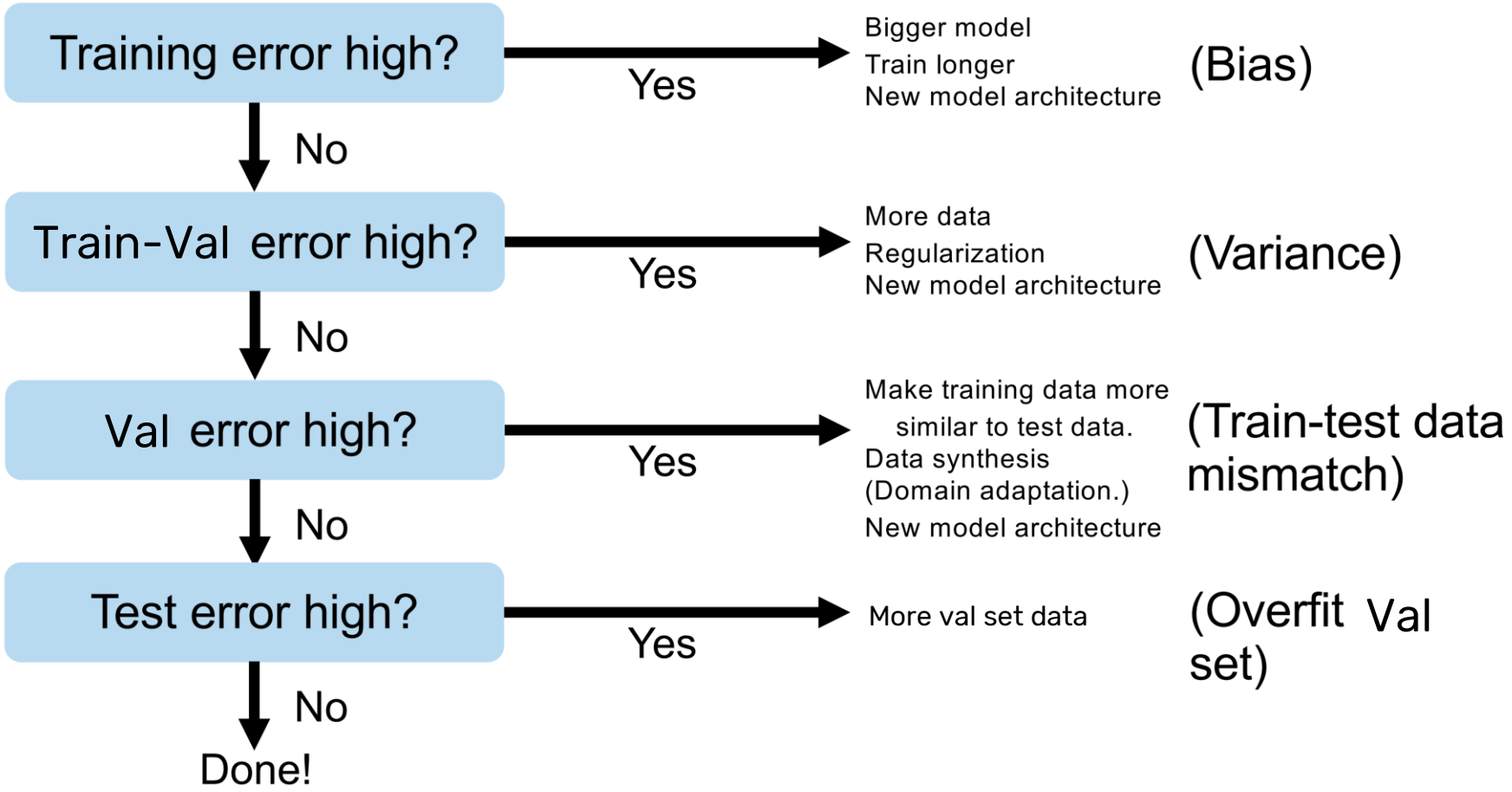
"Avoidable bias"

Overfitting of training

Data mismatch

Overfitting of Val set

New recipe for machine learning



More general formulation

	General speech data (50,000 hours)	In-car speech data (10 hours)
Performance of humans	Human-level error	(Carry out human evaluation to measure.)
Performance on examples you've trained on	Training error	(Insert some in-car data into training set to measure.)
Performance on examples you haven't trained on	Training-Val error	Val/Test error

"Avoidable bias"

"Variance"/degree of overfitting

Data mismatch

In this way we can compare performance in different settings, being able to better assess the model

References

1. Provost, Foster, and Tom Fawcett. "*Data Science for Business: What you need to know about data mining and data-analytic thinking*". O'Reilly Media, Inc., 2013.
2. Brynjolfsson, E., Hitt, L. M., and Kim, H. H. *"Strength in numbers: How does data driven decision making affect firm performance?"* Tech. rep., available at SSRN: http://ssrn.com/abstract=1819486, 2011.
3. Pyle, D. "*Data Preparation for Data Mining*". Morgan Kaufmann, 1999.
4. Kohavi, R., and Longbotham, R. *"Online experiments: Lessons learned"*. Computer, 40 (9), 103–105, 2007.
5. Abu-Mostafa, Yaser S., Malik Magdon-Ismail, and Hsuan-Tien Lin. *"Learning from data"*. AMLBook, 2012.
6. Andrew Ng. *"Machine learning"*. Coursera MOOC. (https://www.coursera.org/learn/machine-learning)
7. Domingos, Pedro. "*The Master Algorithm*". Penguin Books, 2016.
8. Christopher M. Bishop, "*Pattern recognition and machine learning*", *Springer-Verlag New York, 2006.*
9. Hastie, T., Tibshirani, R.,, Friedman, J. *"The Elements of Statistical Learning"*. New York, NY, USA: Springer New York Inc, 2001.
10. Tom Fawcett, *"An introduction to ROC analysis",* Pattern Recognition Letters, Volume 27, Issue 8, 2006, Pages 861-874, ISSN 0167-8655,
11. Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *"An Introduction to Statistical Learning: With Applications in R".* Springer Publishing Company, Incorporated, 2014.
12. Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. "*Dropout: a simple way to prevent neural networks from overfitting".* J. Mach. Learn. Res. 15, 1 (January 2014), 1929-1958.
13. Andrew Ng. *"Deep learning specialization"*. Coursera MOOC (https://www.coursera.org/specializations/deep-learning)
14. Andrew Ng, "*The nuts and bolts of deep learning*", https://media.nips.cc/Conferences/2016/Slides/6203-Slides.pdf