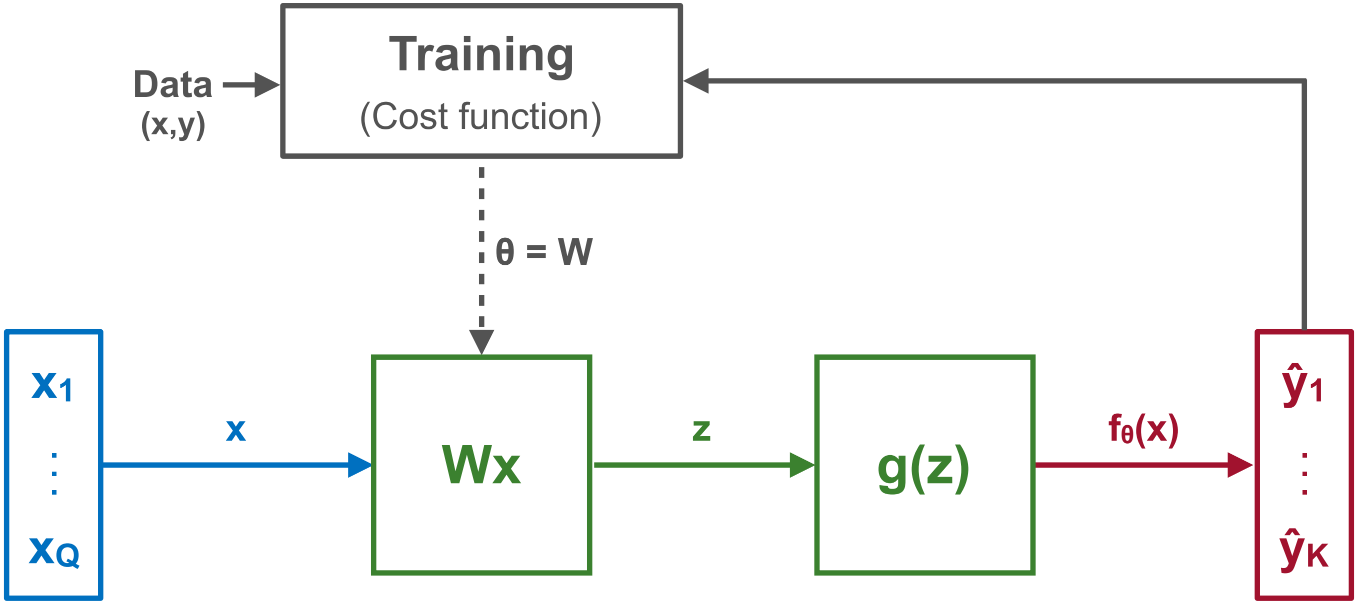
Linear models

Regression
Classification
Ways for improvement

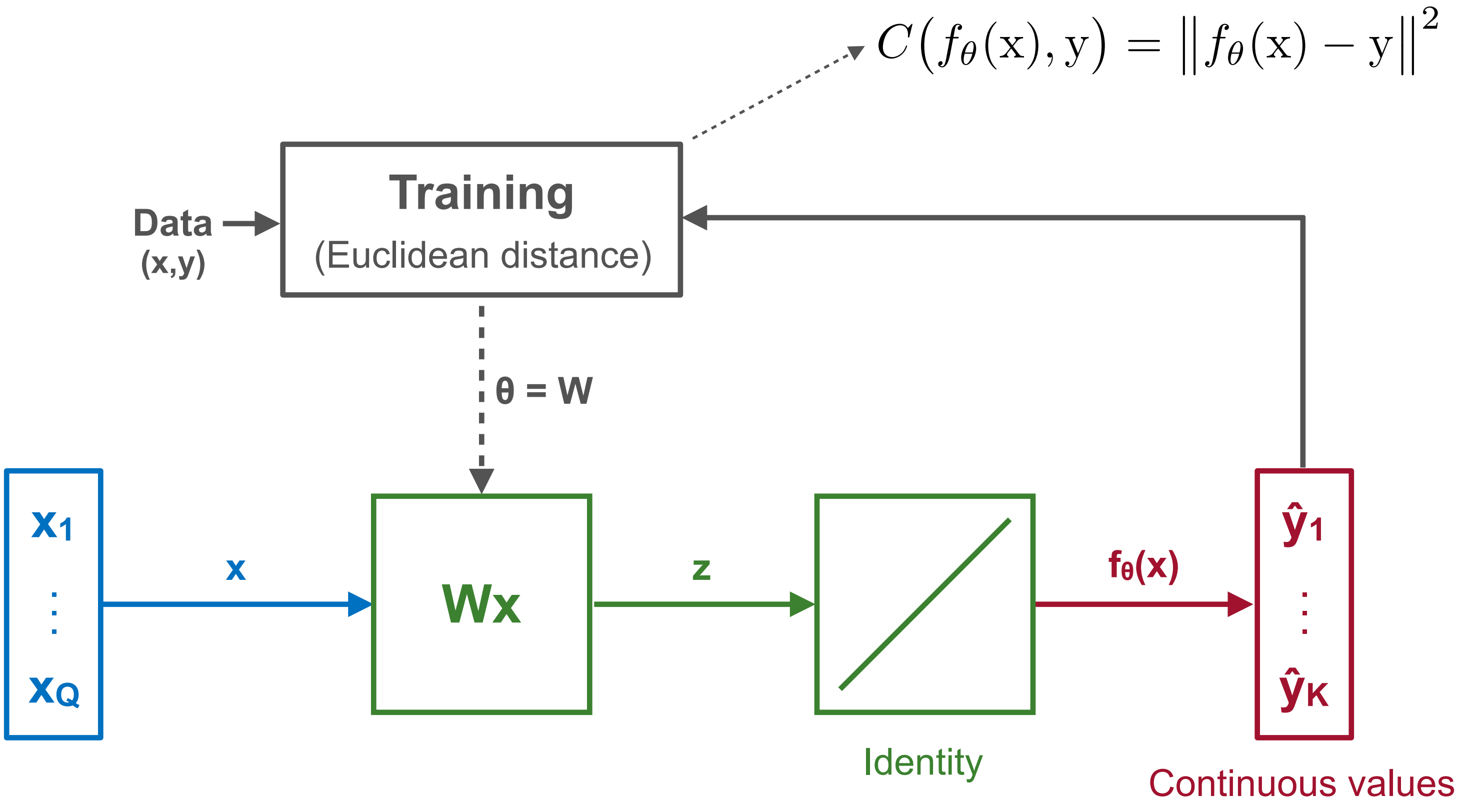
Linear models (1/3)

- So far, we have focused on **linear models**



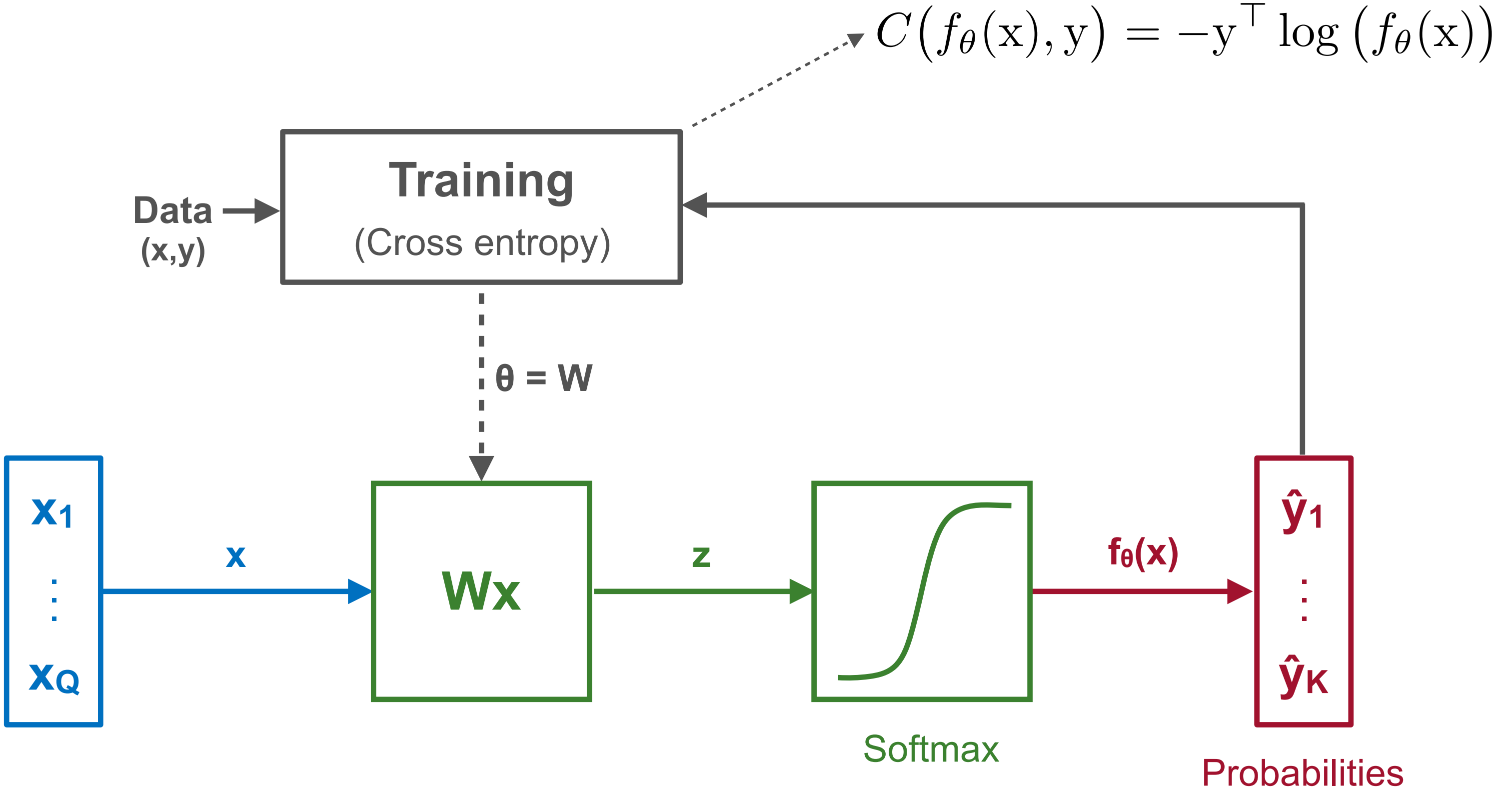
Linear models (2/3)

- Linear model for **regression**



Linear models (3/3)

- Linear model for **classification**

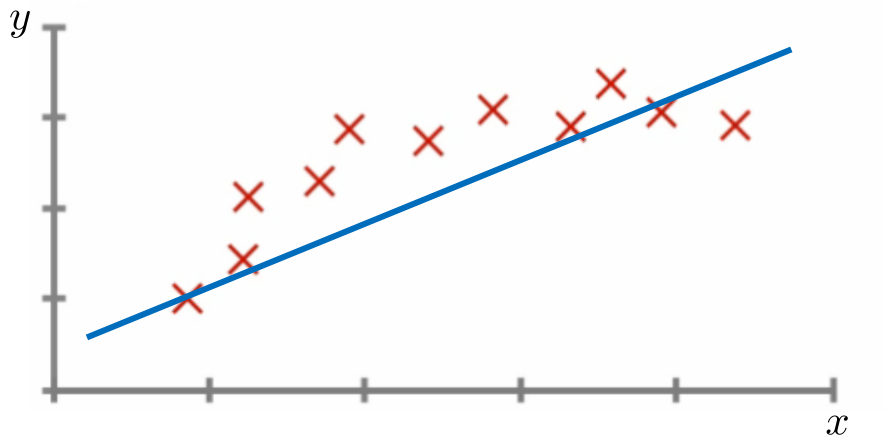


$$C\big(f_\theta(\mathrm{x}),\mathrm{y}\big) = -\mathrm{y}^\top \log\big(f_\theta(\mathrm{x})\big)$$

Nonlinear models (1/2)

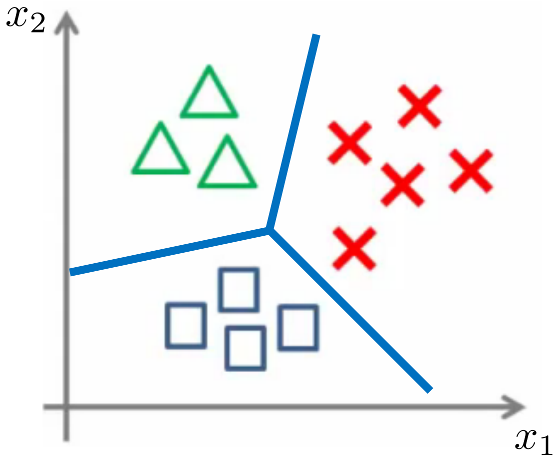
- Linear models may be overly simplistic
  - *Good choice when* ***Q >> N*** *(more features than examples)*
  - *Prone to under-fitting when* ***Q << N*** *(large dataset)*

Regression



(Here, the input space is 1-dimensional)

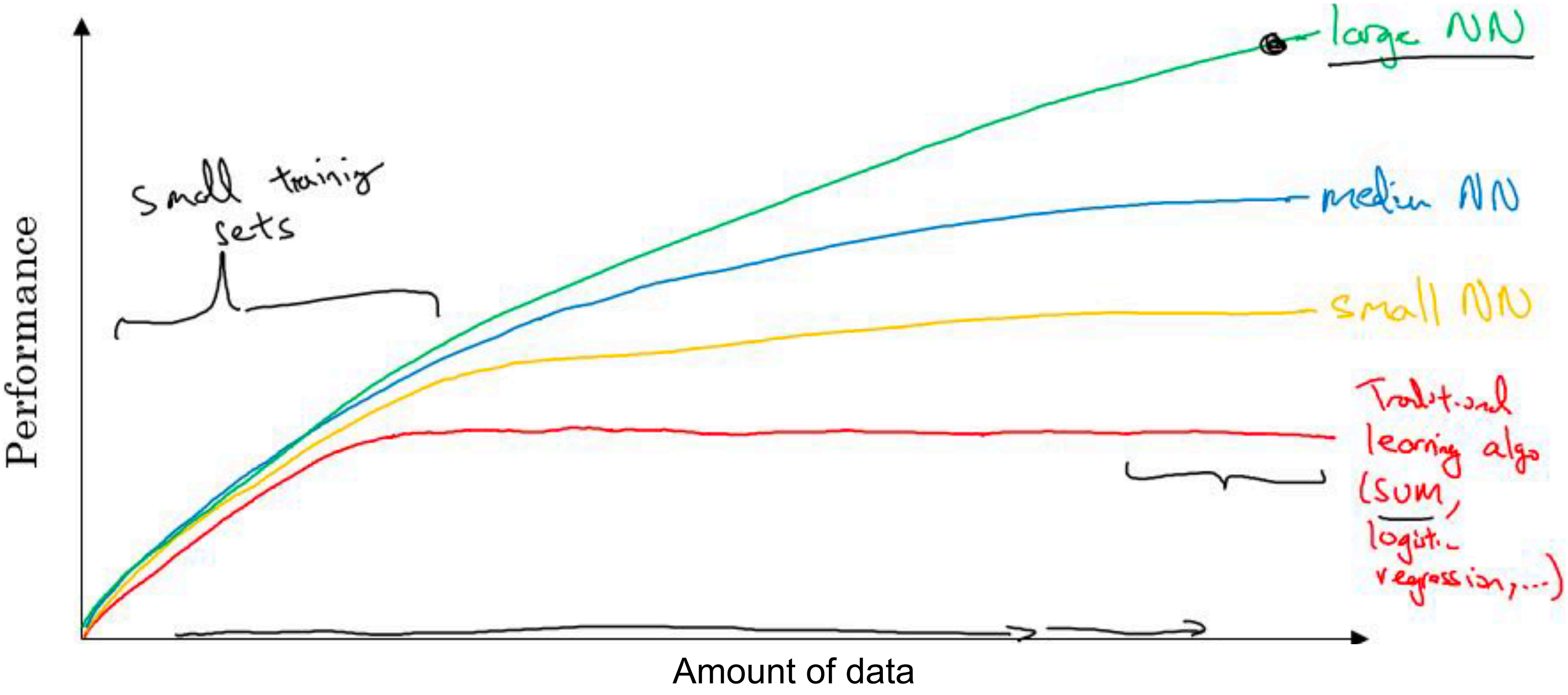
Classification



(Here, the input space is 2-dimensional)

Nonlinear models (2/2)

- How to get **better performance** out of machine learning?
  - *Use "more complex" nonlinear models*
  - *Use much more data for training*

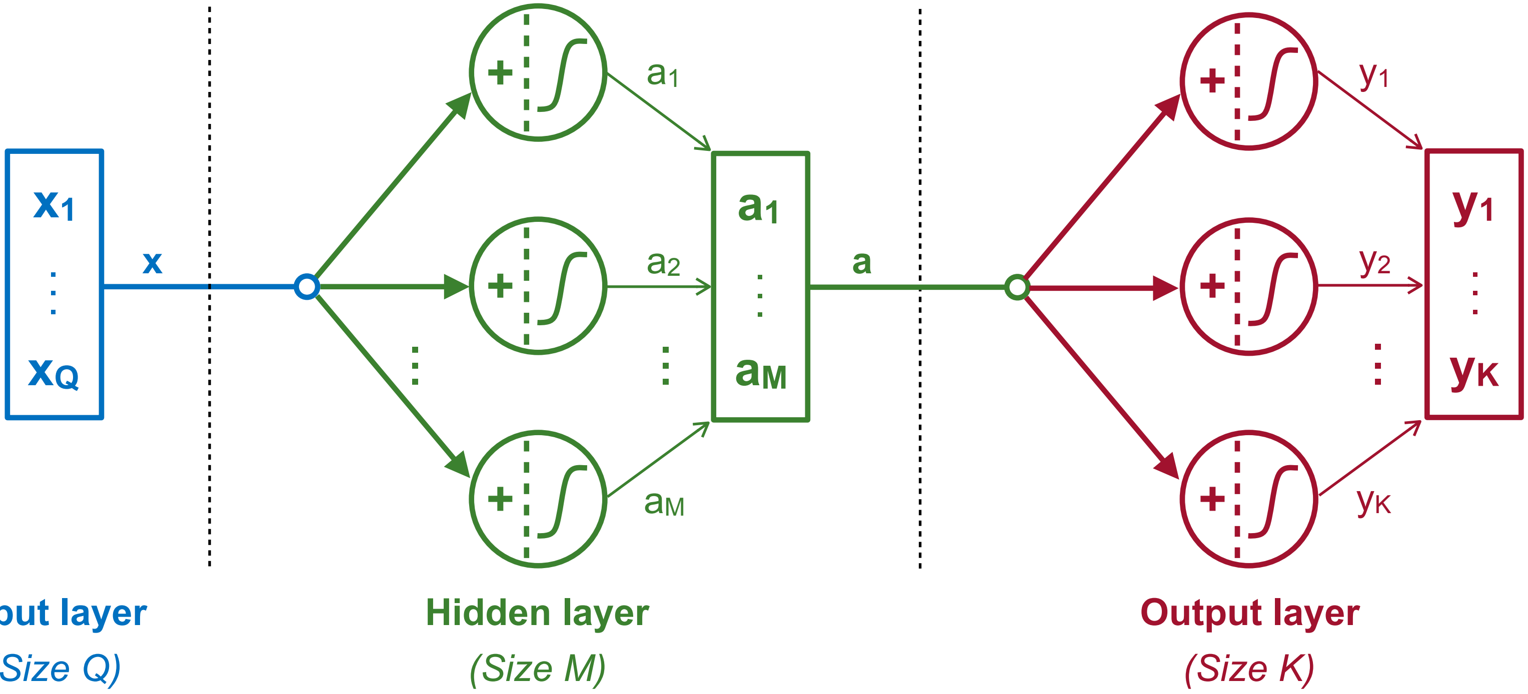


Two-layer neural networks

Hidden layer
Output layer
Forward propagation

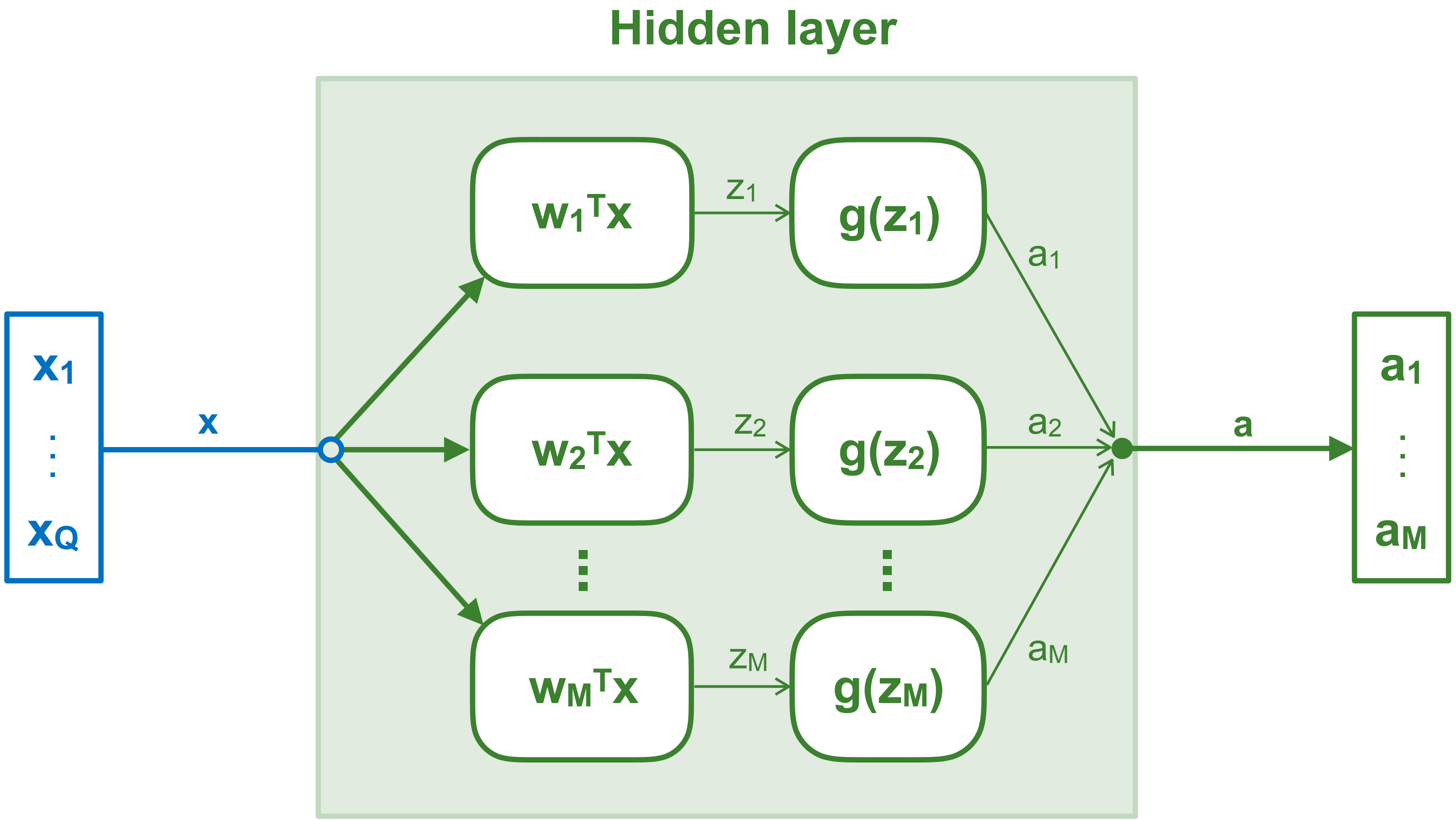
Two-layer neural networks

- A **neural network** consists of units organized in layers
  - ***Feed-forward*** ➔ *Special case when connections don't form cycles*
  - ***Two-layer network*** ➔ *The simplest instance of a feed-forward network*



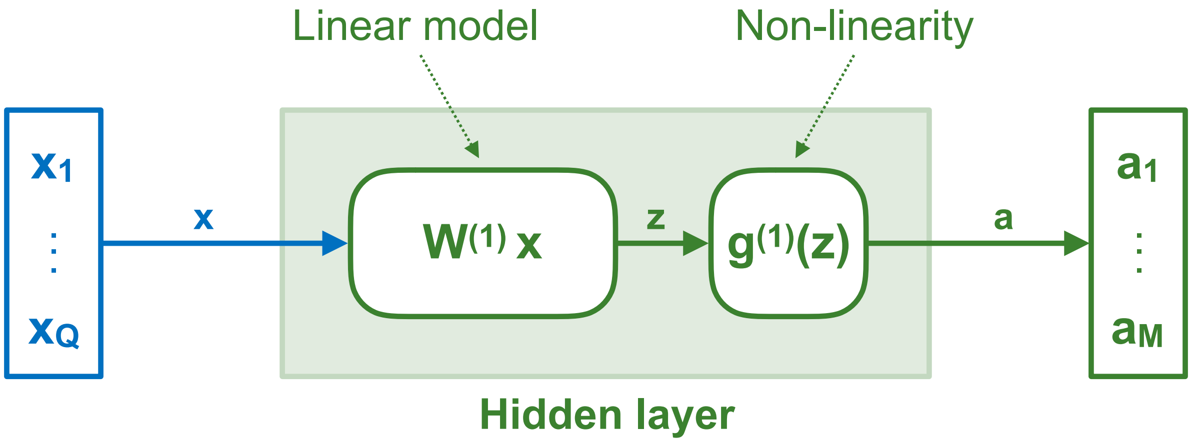
Hidden layer (1/3)

- The **hidden layer** is formed by many “parallel” units



Hidden layer (2/3)

- **Hidden layer** ➔ Linear model $\mathbf{W}^{(1)}$ + Non-linearity $\mathbf{g}^{(1)}$

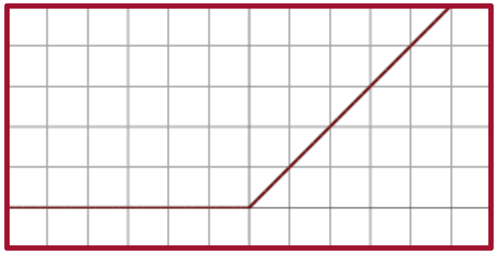


$$\mathrm{x} = \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_Q \end{bmatrix} \qquad \mathrm{W}^{(1)} = \begin{bmatrix} -\mathrm{w}_1^\top - \\ \vdots \\ -\mathrm{w}_M^\top - \end{bmatrix} \qquad \mathrm{g}^{(1)}(\mathrm{z}) = \begin{bmatrix} 1 \\ g(z_1) \\ \vdots \\ g(z_M) \end{bmatrix}$$

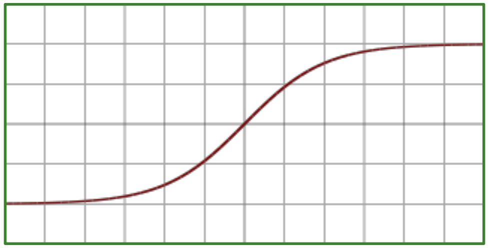
$$\boxed{\mathrm{a} = \mathrm{g}^{(1)}\left(\mathrm{W}^{(1)}\mathrm{x}\right)}$$

Hidden layer (3/3)

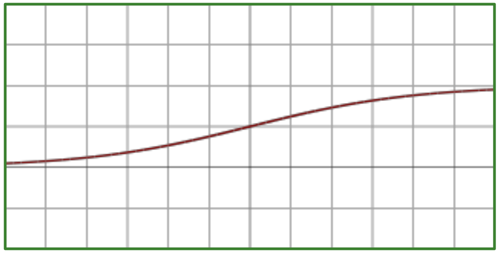
- Different choices for function **$g^{(1)}$** are possible



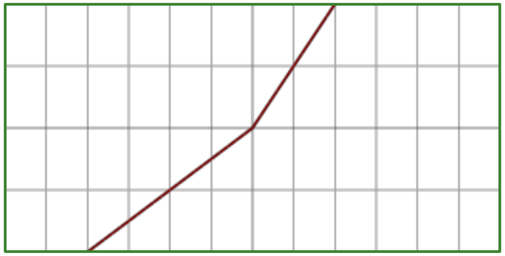
ReLU



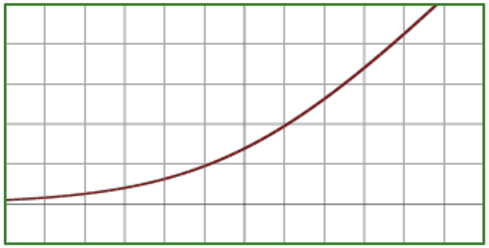
Tanh



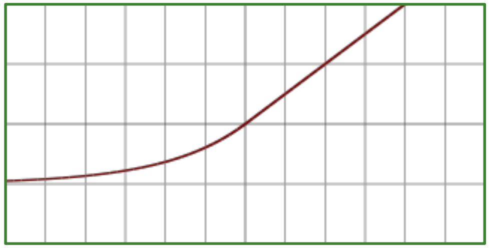
Sigmoid



Leaky ReLu



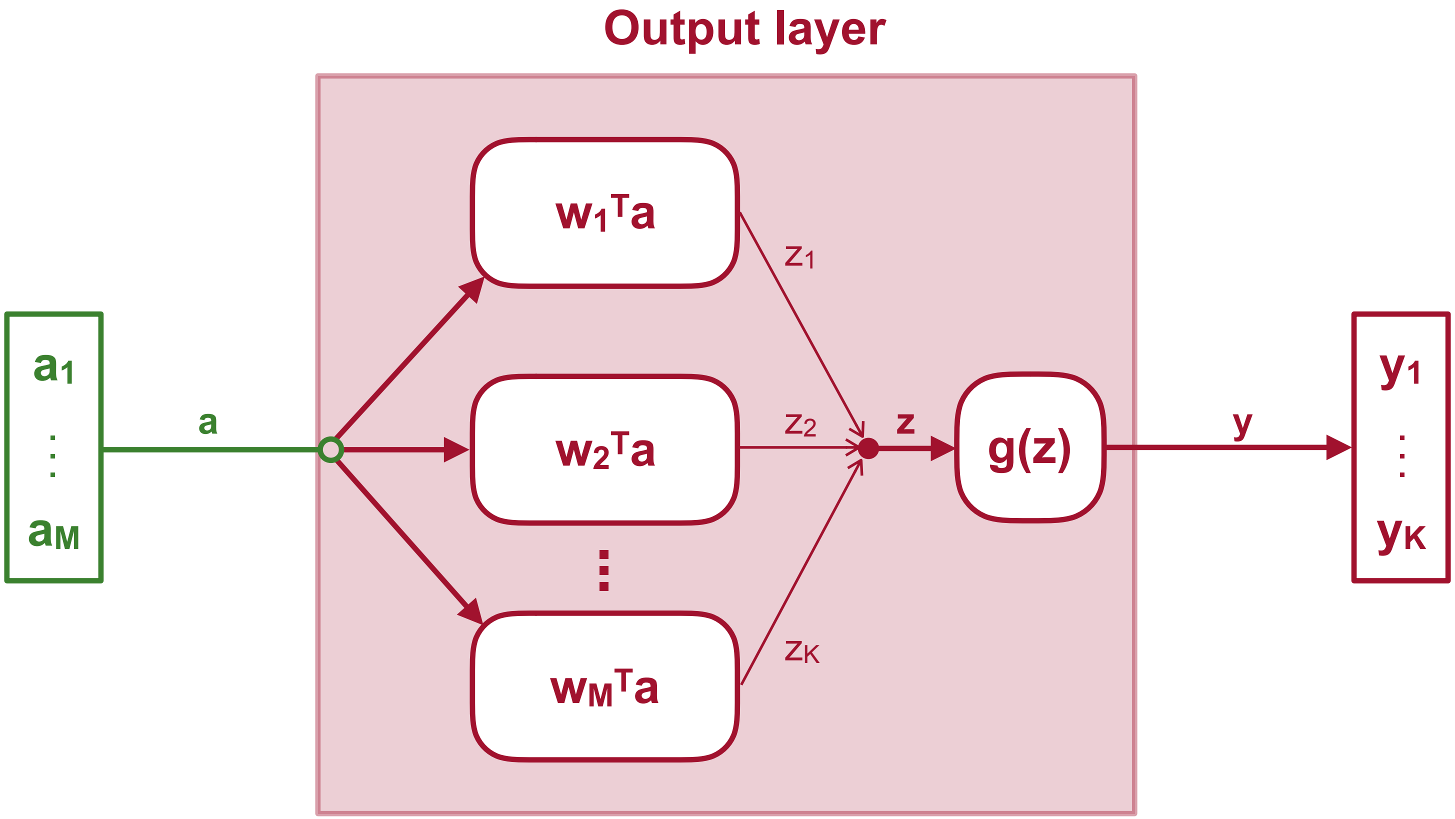
Logistic



Exp-linear

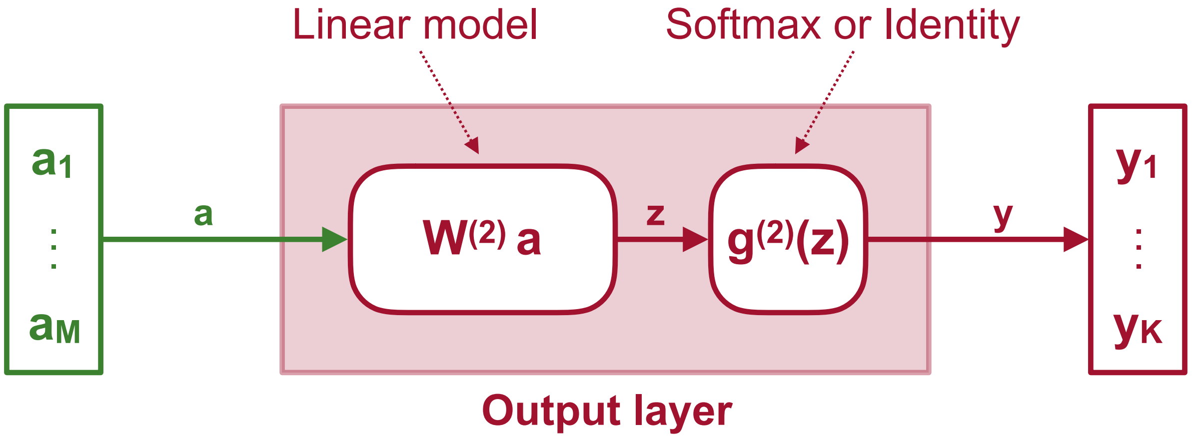
Output layer (1/2)

- The **output layer** is either a **regressor** or a **classifier**



Output layer (2/2)

- **Output layer** ➔ Linear model $\mathbf{W}^{(2)}$ + Softmax/Identity $\mathbf{g}^{(2)}$



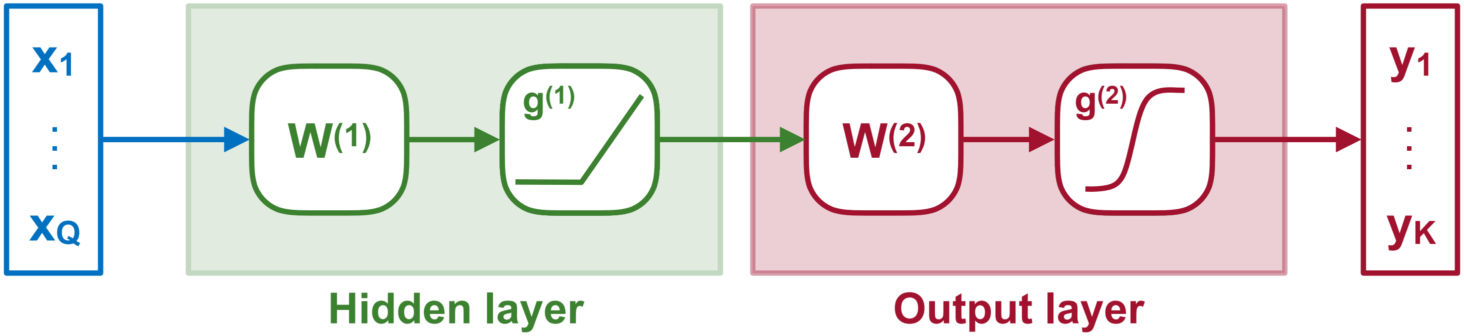
$$W^{(2)} = \begin{bmatrix} -\, w_1^\top \, - \\ \vdots \\ -\, w_K^\top \, - \end{bmatrix} \qquad g^{(2)}(z) = \begin{bmatrix} \sigma_1(z) \\ \vdots \\ \sigma_K(z) \end{bmatrix}$$

$$\boxed{y = g^{(2)}\big(W^{(2)} a\big)}$$

Forward propagation (1/2)

- Neural network with **2 layers**
  - ***Hidden layer*** ➔ *The input is transformed into (learned) features*
  - ***Output layer*** ➔ *The features are used for regression or classification*

$$f_\theta(\mathrm{x}) = \mathrm{g}^{(2)}\big(\mathrm{W}^{(2)}\mathrm{g}^{(1)}\big(\mathrm{W}^{(1)}\mathrm{x}\big)\big)$$



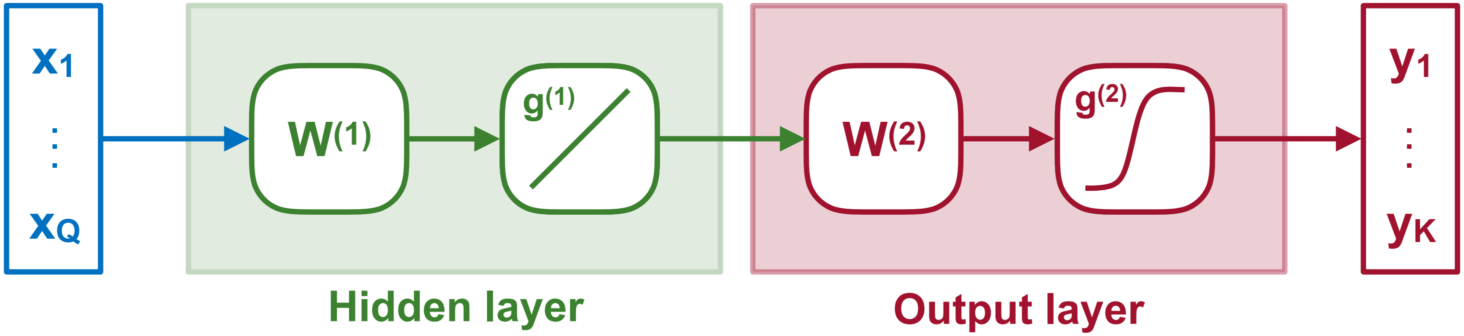
Forward propagation (2/2)

- The **non-linearity** of **g$^{(1)}$** is essential for neural networks
  - *Otherwise, the network behaves like a* ***linear regressor/classifier***

$$g^{(1)}(z) = z \quad \Rightarrow \quad f_\theta(x) = g^{(2)}\big(W^{(2)}W^{(1)}x\big) = g^{(2)}\big(Wx\big) \quad \longrightarrow \quad \text{linear}$$

This behaves like a linear model !!!

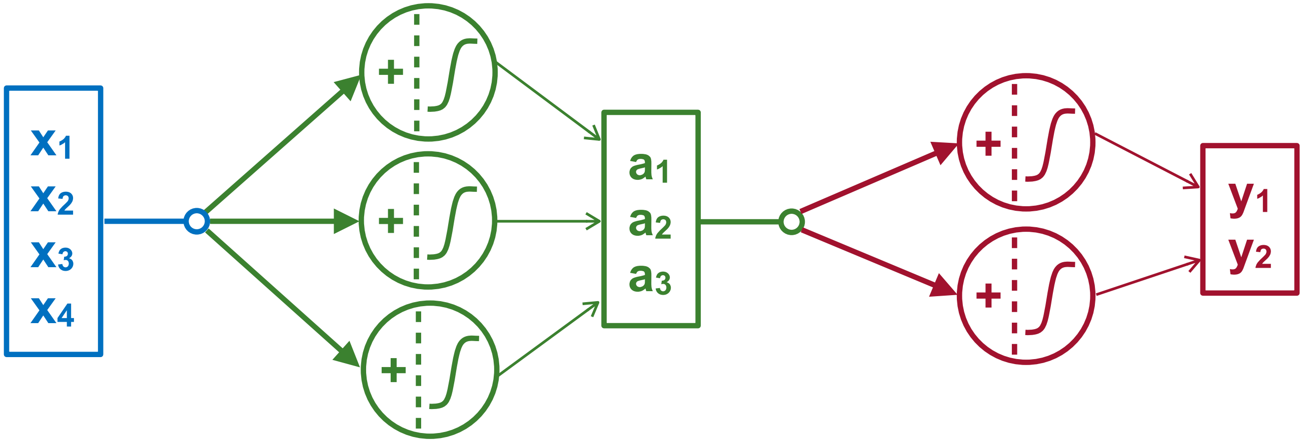
(with more parameters than strictly necessary)



Quiz

- Consider the network shown below.

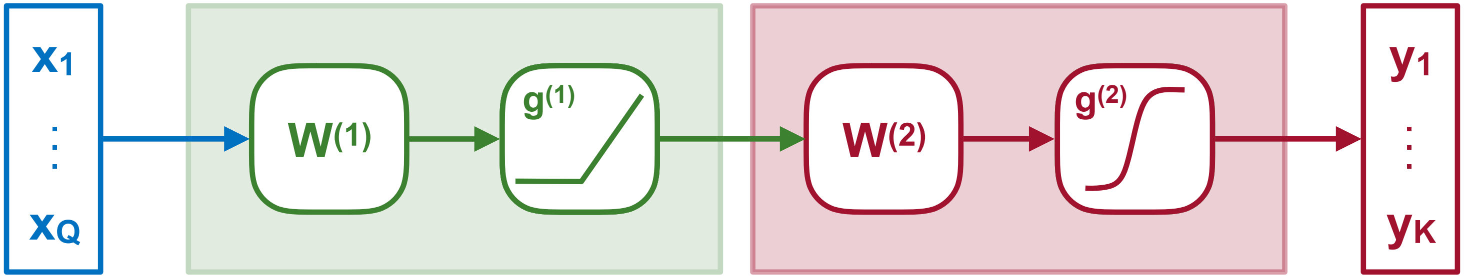
1) *What is the size of matrix $W^{(1)}$?*
2) *What is the size of vector $z^{(1)}$?*
3) *What is the size of matrix $W^{(2)}$?*
4) *What is the size of matrix $z^{(2)}$?*



What we have seen so far…

- Architecture of **shallow neural networks**
  - ***Hidden layer*** ➔ *Feature learning*
  - ***Output layer*** ➔ *Regression or classification*
- Function $\mathbf{g}^{(1)}$ must be non-linear ➔ **ReLU**

$$f_\theta(\mathrm{x}) = \mathrm{g}^{(2)}\big(\mathrm{W}^{(2)}\mathrm{g}^{(1)}\big(\mathrm{W}^{(1)}\mathrm{x}\big)\big)$$



Multilayer neural networks

Multilayer neural networks

Forward propagation

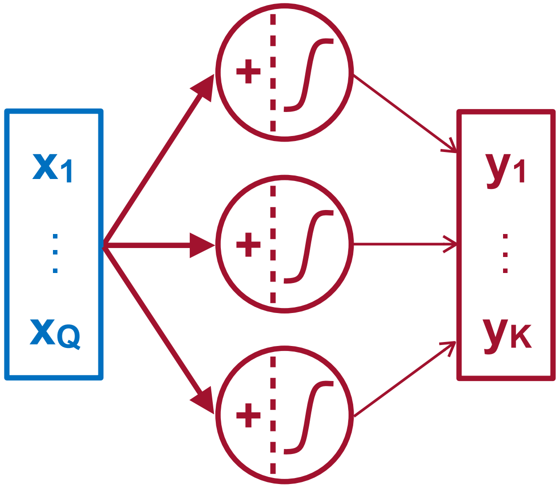
Why deep models?

Multilayer neural networks (1/2)

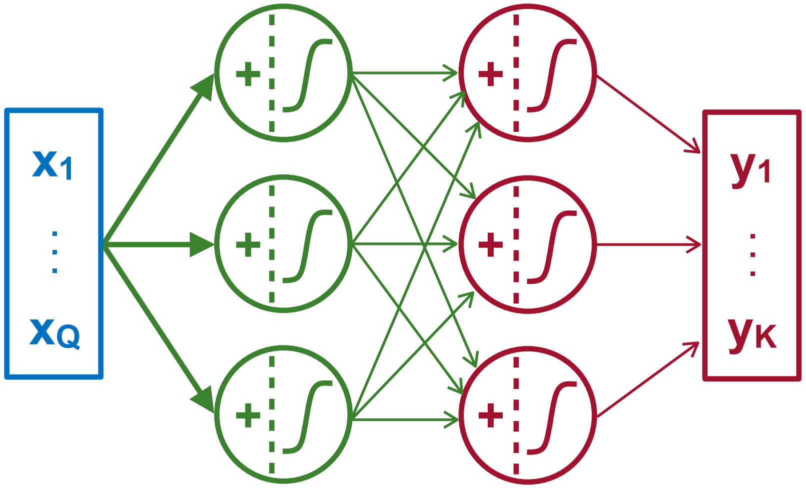
- So far, we have focused on **shallow networks**

1-layer network

(Logistic regression)

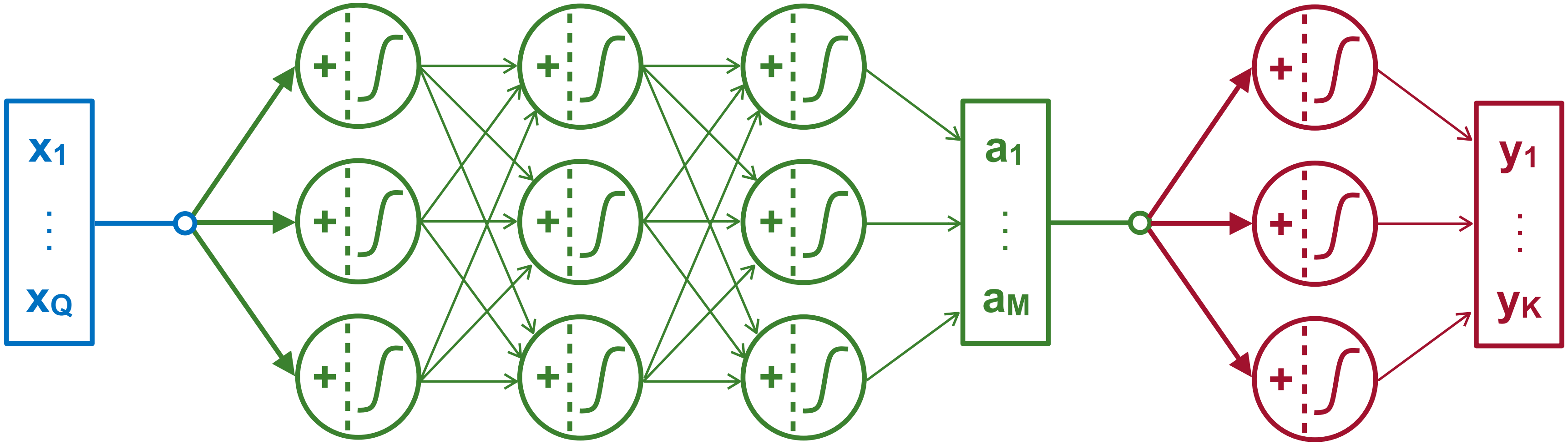


2-layer network



Multilayer neural networks (2/2)

- A **multilayer network** contains many hidden layers



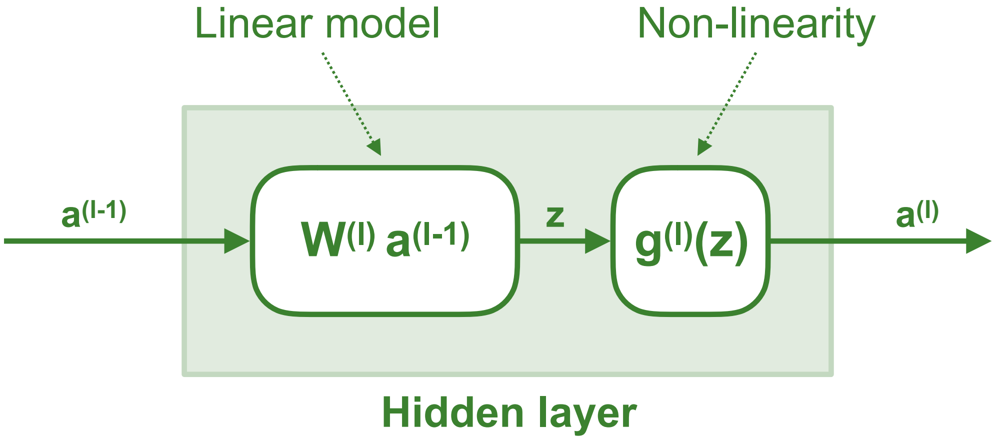
Input layer
(Size M_0=Q)

Hidden layers
(Sizes $M_1,\dots,M_{L-1}$)

Output layer
(Size M_L=K)

Forward propagation (1/2)

- Each **hidden layer** has a similar structure



$$W^{(\ell)} = \begin{bmatrix} -\, w_1^\top \,- \\ \vdots \\ -\, w_{M_\ell}^\top \,- \end{bmatrix} \qquad g^{(\ell)}(z) = \begin{bmatrix} 1 \\ g(z_1) \\ \vdots \\ g(z_{M_\ell}) \end{bmatrix}$$

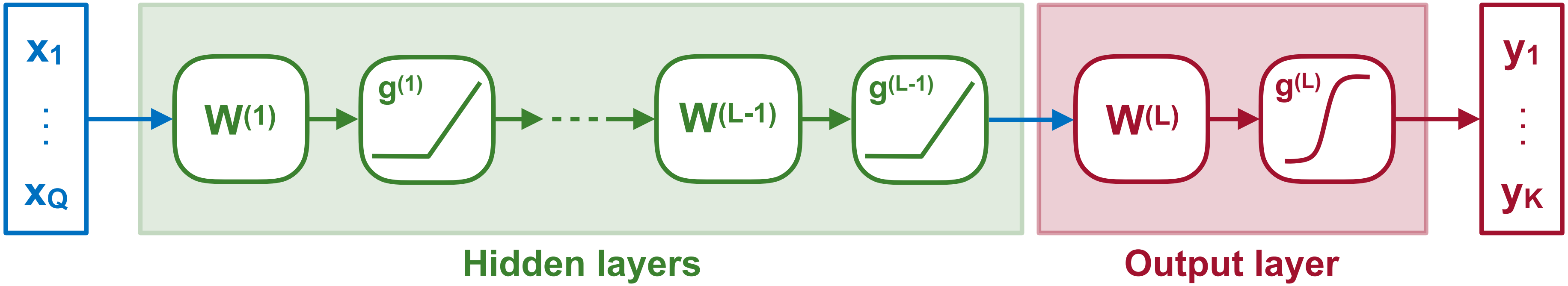
$$\boxed{a^{(\ell)} = g^{(\ell)}\left(W^{(\ell)} a^{(\ell-1)}\right)}$$

$$a^{(0)} = x$$
$$y = a^{(L)}$$

Forward propagation (2/2)

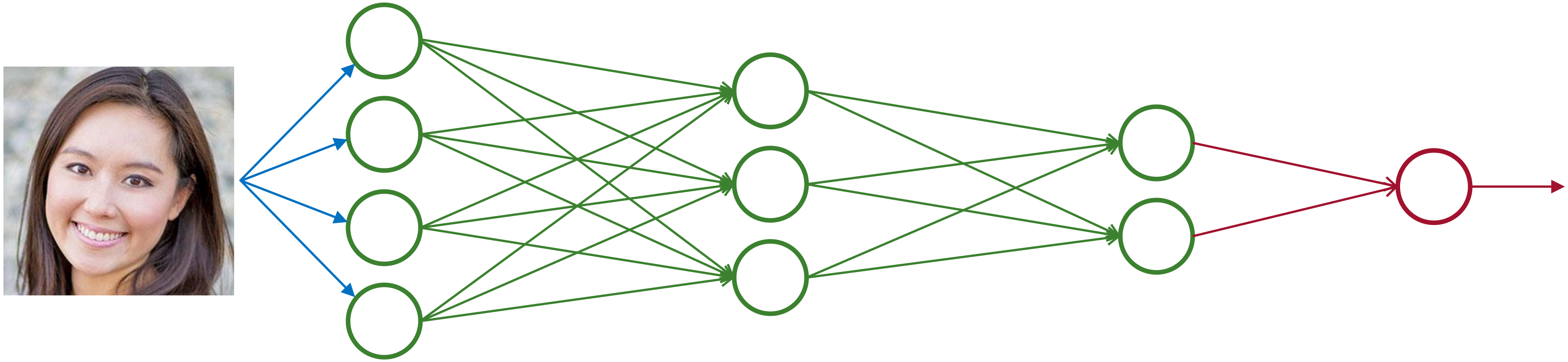
- Neural network with **multiple layers**
  - ***Hidden layers*** ➔ *The input is transformed into (learned) features*
  - ***Output layer*** ➔ *The features are used for regression or classification*

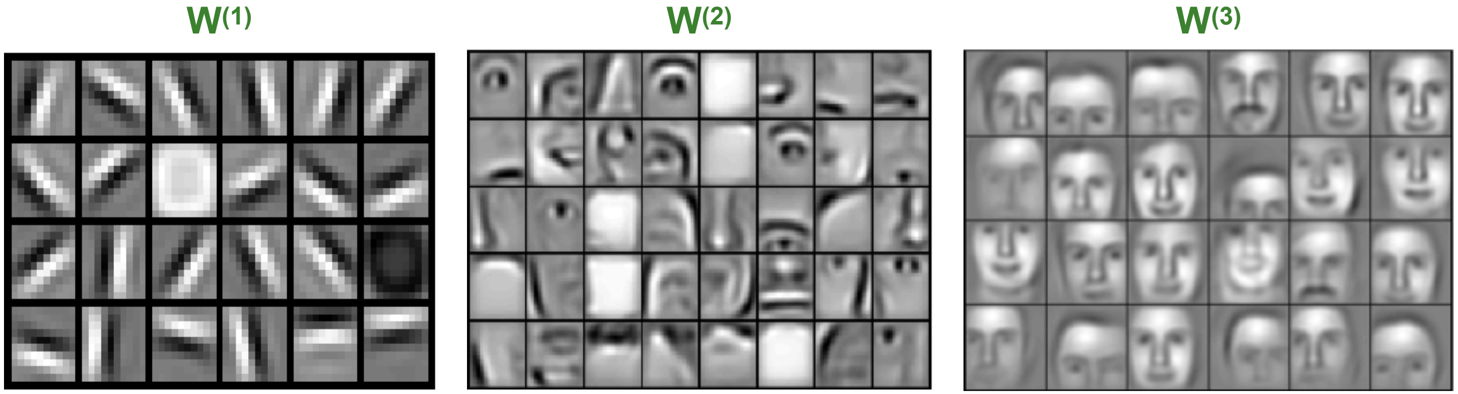
$$f_\theta(\mathrm{x}) = \mathrm{g}^{(L)}\left(\mathrm{W}^{(L)} \dots \mathrm{g}^{(2)}\left(\mathrm{W}^{(2)}\mathrm{g}^{(1)}\left(\mathrm{W}^{(1)}\mathrm{x}\right)\right)\right)$$



Why neural networks? (1/2)

- Neural networks can learn a **hierarchical representation**

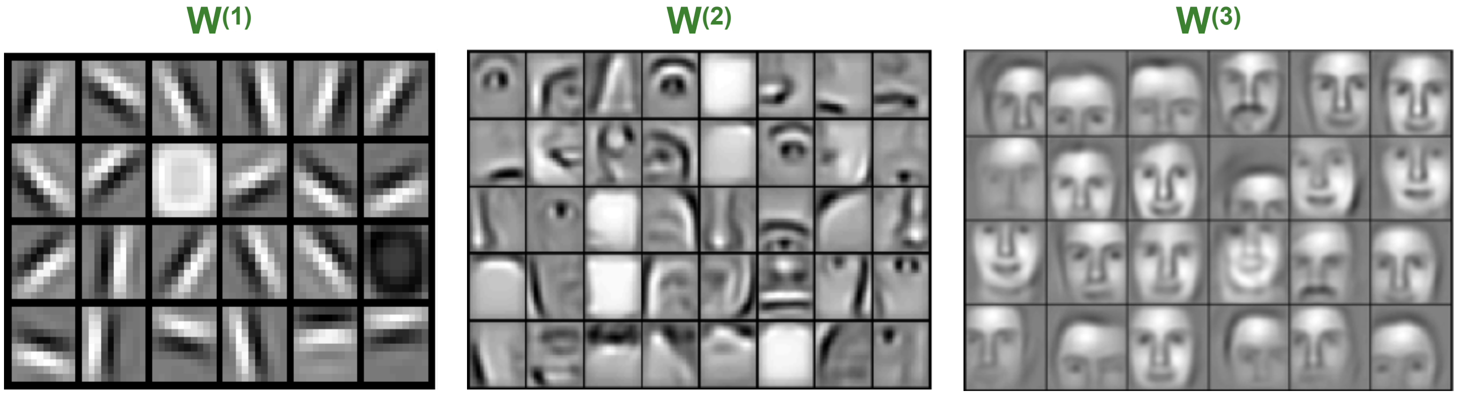




Visualization of hidden unis

Why neural networks? (2/2)

- Neural networks can learn **hierarchical representations**
  - ***First layer*** ➔ *Localization of edges in the input images*
  - ***Second layer*** ➔ *Grouping of edges into shapes (e.g., eyes, noses, …)*
  - ***Third layer*** ➔ *Formation of full objects (e.g., faces)*
  - ***Fourth layer*** ➔ *Object classification (e.g., face detection)*

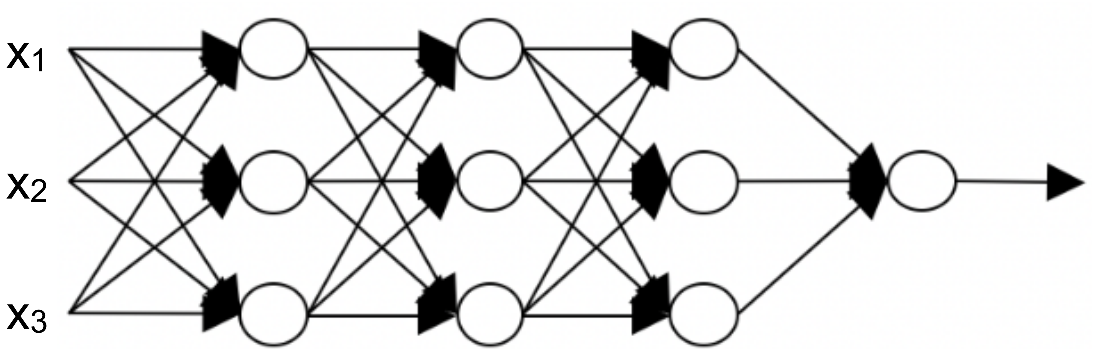


Visualization of hidden unis

Quiz

- How many layers does the following network have?

1) *The number of layers is 4, whereof 3 are hidden layers.*
2) *The number of layers is 3, whereof 3 are hidden layers.*
3) *The number of layers is 4, whereof 4 are hidden layers.*
4) *The number of layers is 5, whereof 4 are hidden layers.*

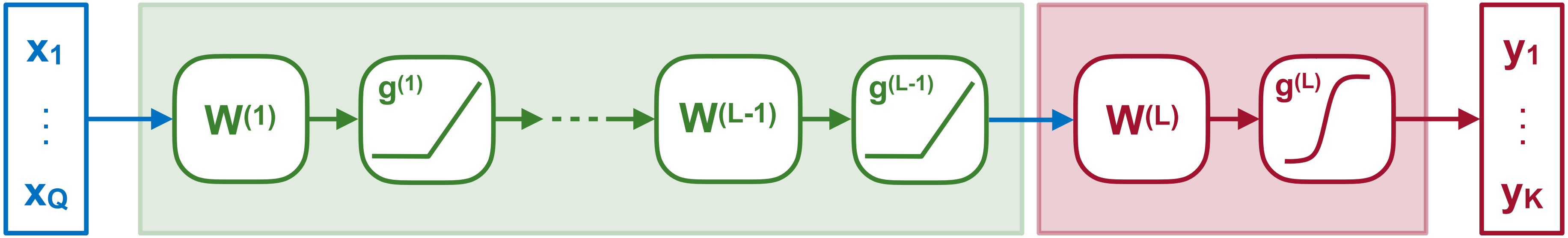


What we have seen so far…

- Architecture of **multilayer neural networks**
  - ***Hidden layers*** ➔ *Hierarchical feature learning*
  - ***Output layer*** ➔ *Regression or classification*

- Functions **$g^{(1)},\ldots,g^{(L-1)}$** must be non-linear ➔ **ReLU**

$$f_\theta(\mathrm{x}) = \mathrm{g}^{(L)}\left(\mathrm{W}^{(L)} \ldots \mathrm{g}^{(2)}\left(\mathrm{W}^{(2)}\mathrm{g}^{(1)}\left(\mathrm{W}^{(1)}\mathrm{x}\right)\right)\right)$$



Neural network training

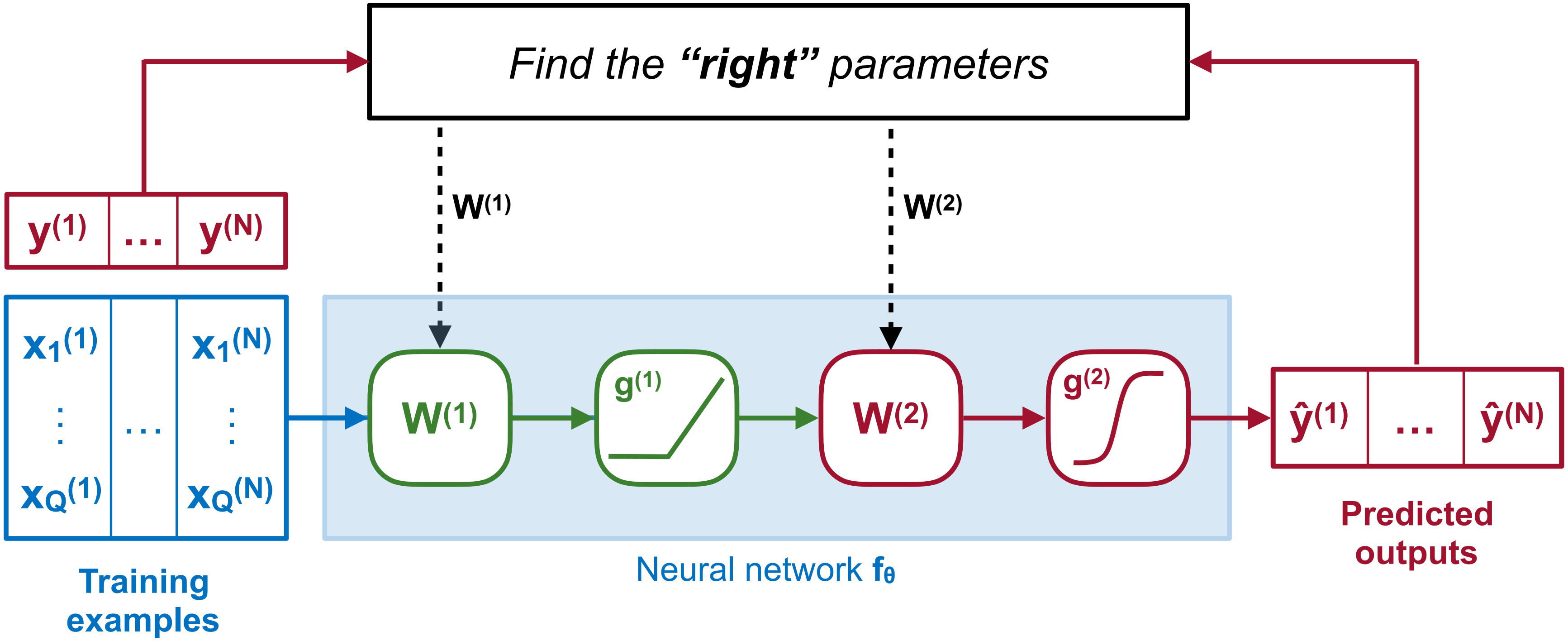
Cost function

Practical advice

Hyper-parameters

Cost function for neural networks (1/3)

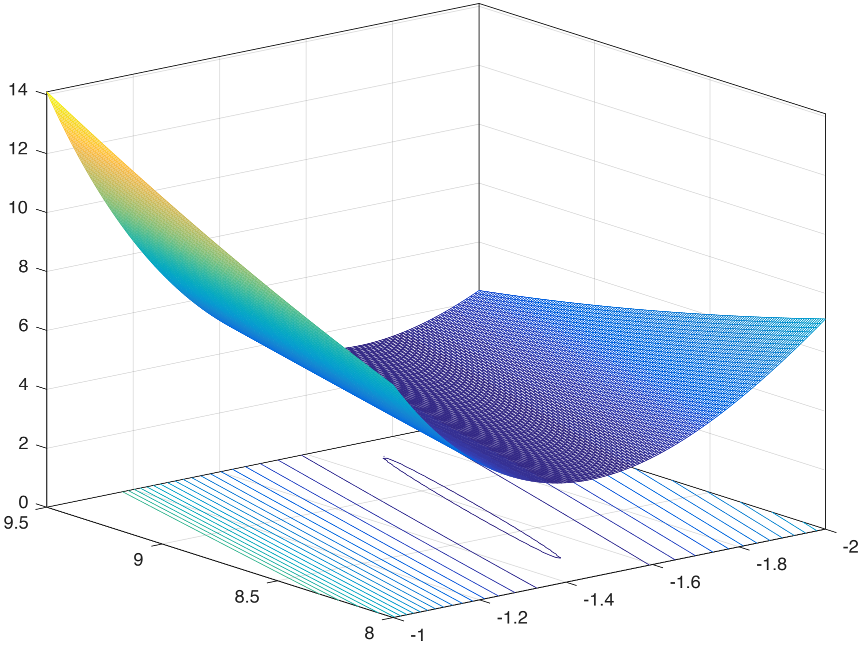
- Our goal is to **learn** the prediction $\mathbf{f}_\theta$ from training data
  - *This amounts to finding the "right values" for parameters* $\boldsymbol{\theta} = (\mathbf{W}^{(1)}, \mathbf{W}^{(2)})$



Cost function for neural networks (2/3)

- How to choose the **“right values”** for parameters **θ** ?
  - *We select **θ** such that the **model f_θ is fitted** to the training data*

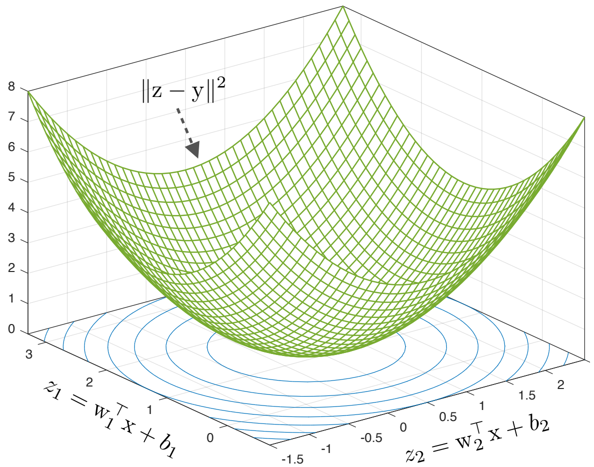
$$\widehat{\theta} = \arg\min_{\theta} \sum_{n=1}^{N} \underbrace{C}_{\text{Cost function}}\Big(\underbrace{f_\theta(\mathrm{x}^{(n)})}_{\text{Prediction}}, \underbrace{\mathrm{y}^{(n)}}_{\text{Output}}\Big)$$



Cost function for neural networks (3/3)

- Euclidean distance for **regression**

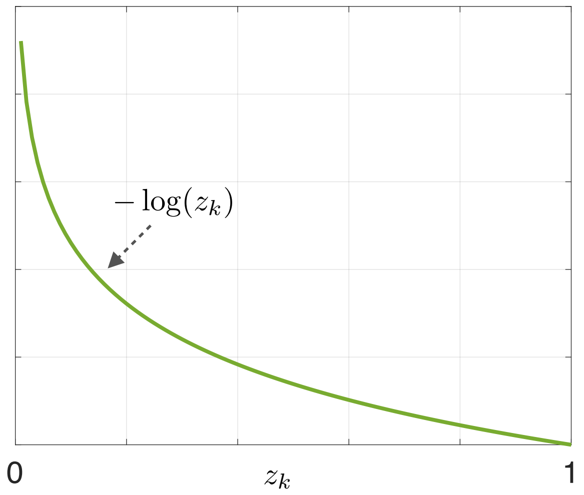
$$C\big(f_\theta(\mathrm{x}), \mathrm{y}\big) = \big\|f_\theta(\mathrm{x}) - \mathrm{y}\big\|^2$$



- Cross-entropy for **classification**

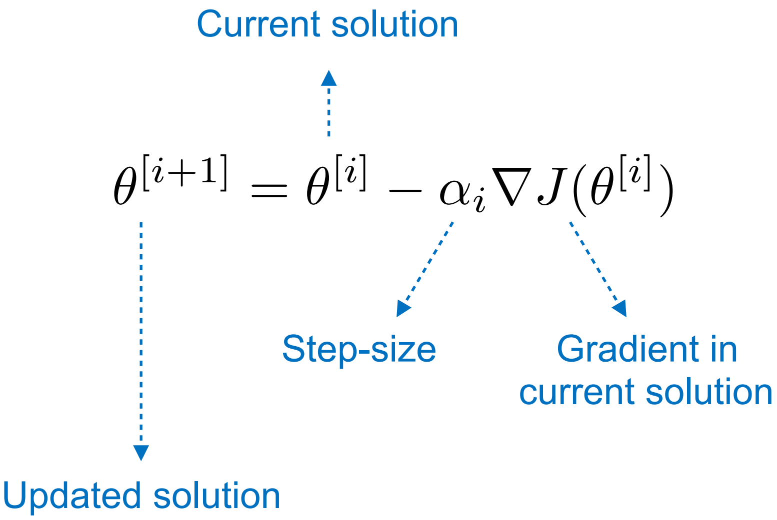
$$C\big(f_\theta(\mathrm{x}), \mathrm{y}\big) = -\mathrm{y}^\top \log\big(f_\theta(\mathrm{x})\big)$$

One-hot encoding



Gradient descent (1/2)

- How to **minimize the cost J(θ)** on the training set ?
    - *We find the optimal* ***θ*** *through* ***gradient descent***



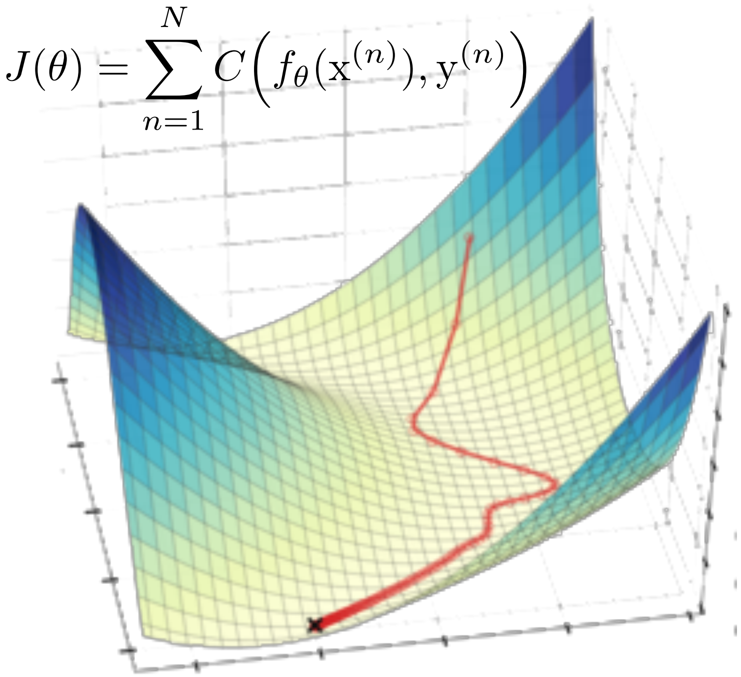
Current solution

$$\theta^{[i+1]} = \theta^{[i]} - \alpha_i \nabla J(\theta^{[i]})$$

Step-size

Gradient in current solution

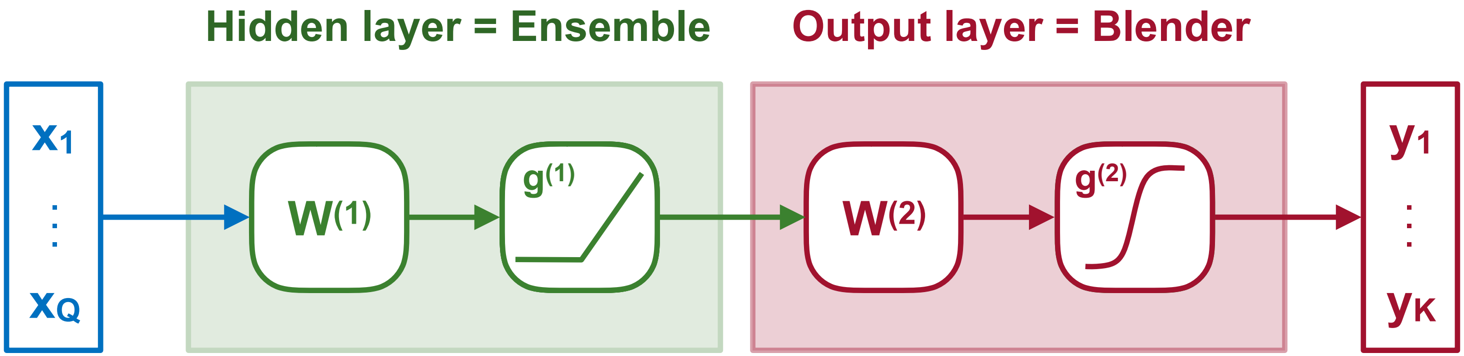
Updated solution



$$J(\theta) = \sum_{n=1}^{N} C\Big(f_\theta(\mathrm{x}^{(n)}), \mathrm{y}^{(n)}\Big)$$

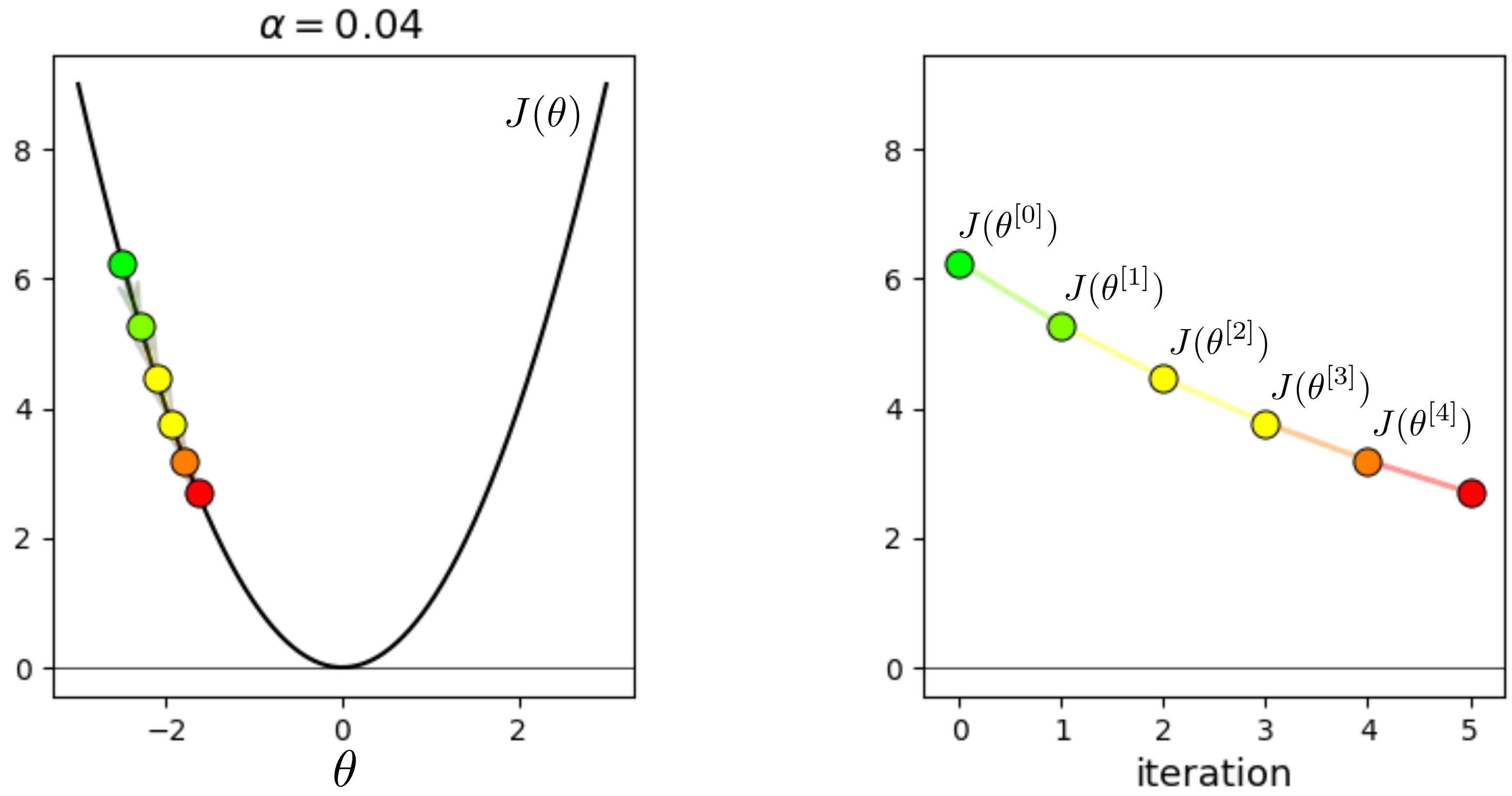
Gradient descent (2/2)

- The neural network parameters are **randomly initialized**
  - *If the parameters were initialized to zero, each neuron in the hidden layer would perform the same computation…*
  - *… so even after multiple iterations of gradient descent, each neuron in the layer would be computing the same thing as other neurons.*
  - ***Recall*** ➜ *Random initialization introduces **diversity** in the ensemble.*



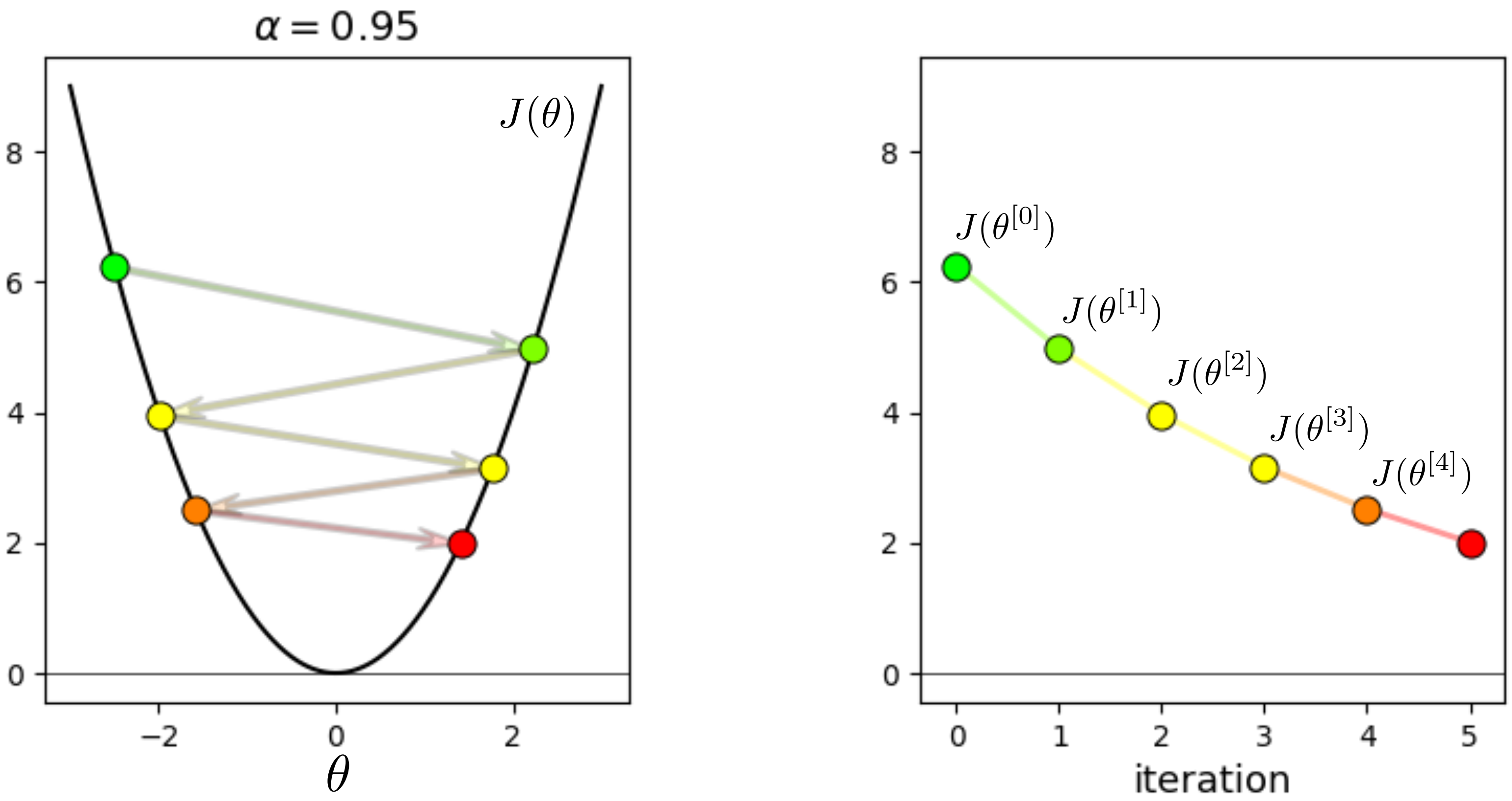
Step-size (1/4)

- **Case 1 ➜** Slow convergence when **α** is "too small"



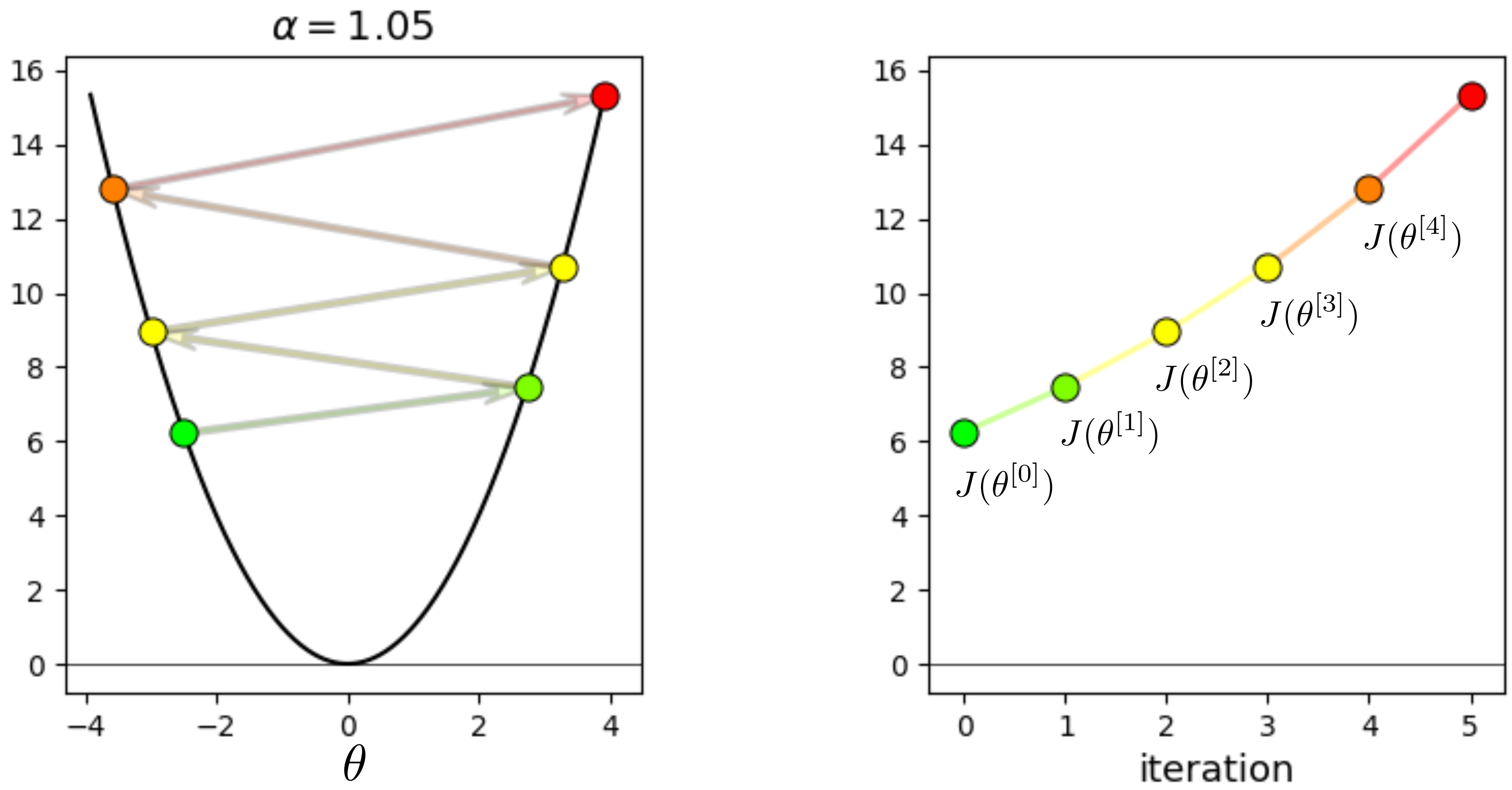
Step-size (2/4)

- **Case 2 ➜** Slow convergence when **α** is "too big"



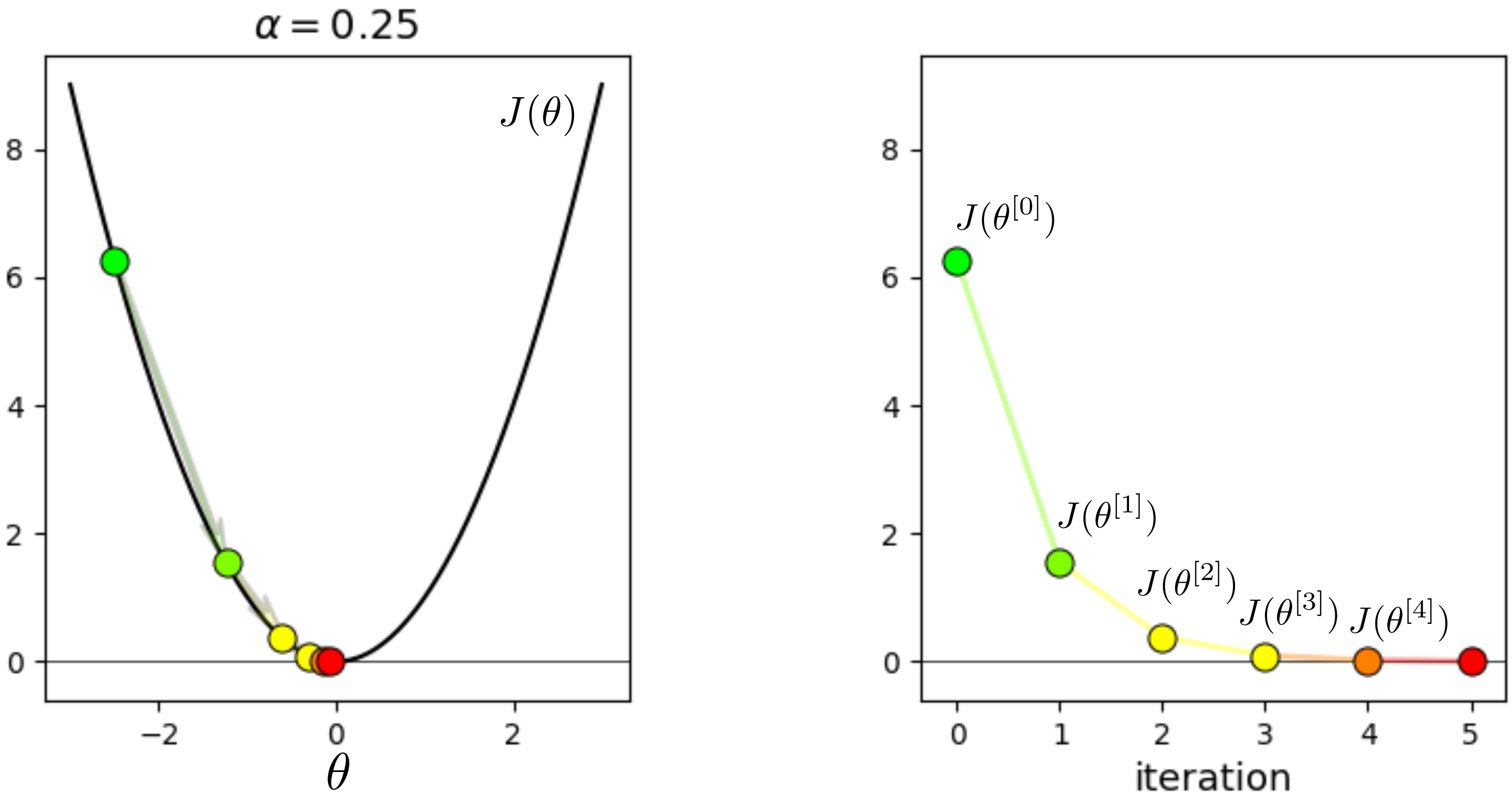
Step-size (3/4)

- **Case 3 ➜** Divergence when **α** is “way too big”



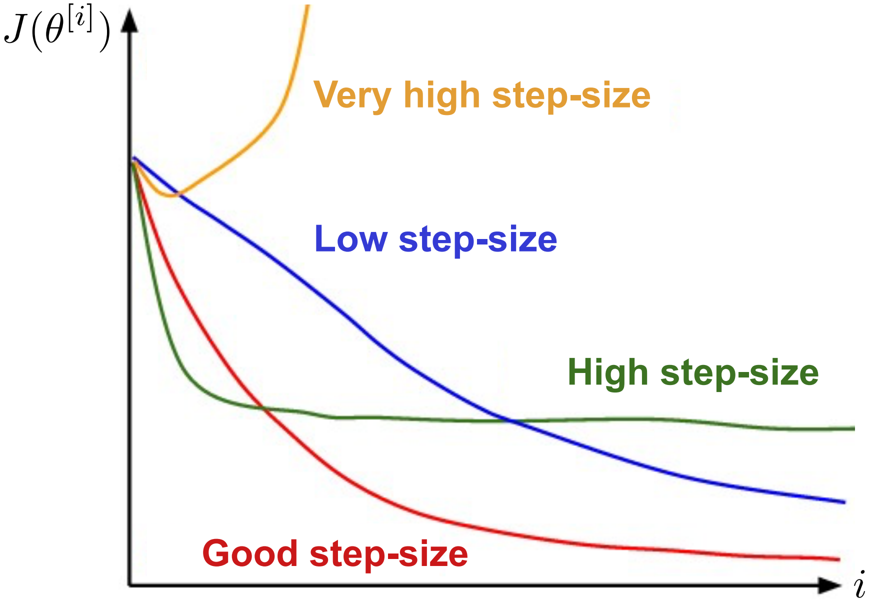
Step-size (4/4)

- **Case 4 ➜** Fast convergence when **α** is "just right"



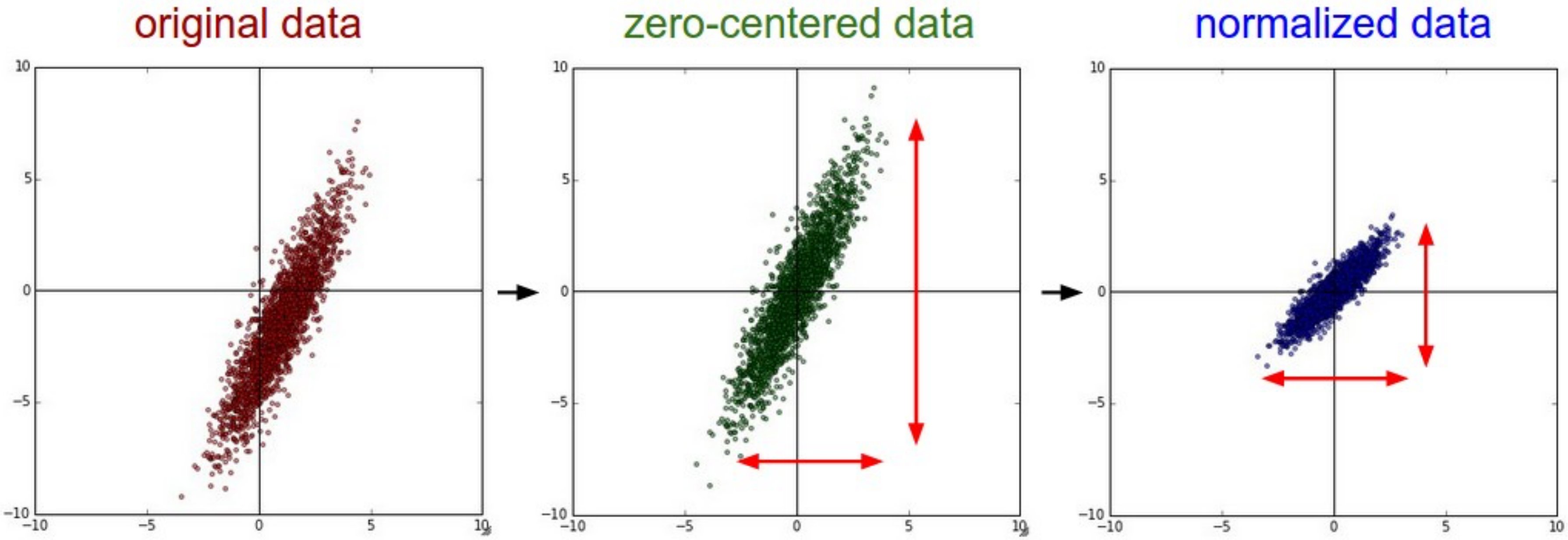
Practical advice (1/2)

- **Advice ➜** Track the cost function during training
  - *Compute **$J(\theta^{[i]})$** at each iteration **i** and save/plot its value*
  - *The shape of **$J(\theta^{[0]}),\ldots,J(\theta^{[i]})$** will tell you about the **step-size***



Practical advice (2/2)

- **Advice ➜** Normalize data at the network's input
  1) *Subtract the mean across every individual feature in the data*
  2) *Divide each feature by its standard deviation (after mean subtraction)*



Quiz

- In order to train a neural network with gradient descent, suppose you have initialized its parameters to be zero. Which of the following statements is true?

  1) *Each neuron in the hidden layer will perform the same computation. So even after multiple iterations of gradient descent, each neuron in the layer will be computing the same thing as other neurons.*

  2) *Each neuron in the hidden layer will perform the same computation in the first iteration. But after one iteration of gradient descent, they will learn to compute different things, because we have "broken symmetry".*

  3) *Each neuron in the hidden layer will compute the same thing, but the neurons in the output layer will compute different things.*

  4) *The neurons in the hidden layer will perform different computations from each other even in the first iteration; their parameters will thus keep evolving in their own way.*

What we have seen so far…

- Neural networks are trained with gradient descent

$$\theta^{[i+1]} = \theta^{[i]} - \alpha_i \nabla J(\theta^{[i]})$$

- Practical advice for learning

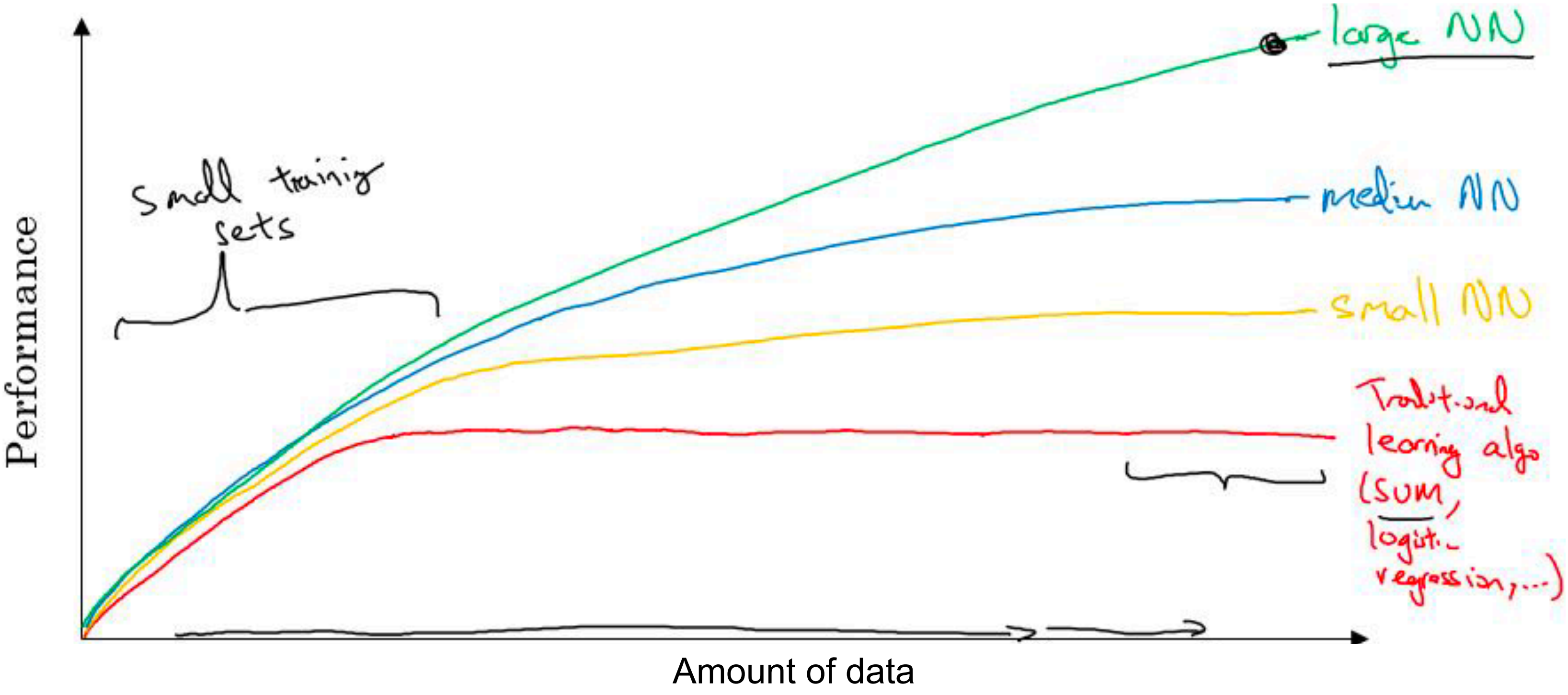
1) *Data normalization* → *Speed up the optimization*
2) *Cost tracking* → *Just for checking the convergence*
3) *Choice of the step-size* → *Help converging to better minima*
4) *Random initialization* → *Otherwise the network won’t learn*

Conclusion

Quest for nonlinear models
Multilayer networks
Hierarchical representation
Why are neural networks successful?

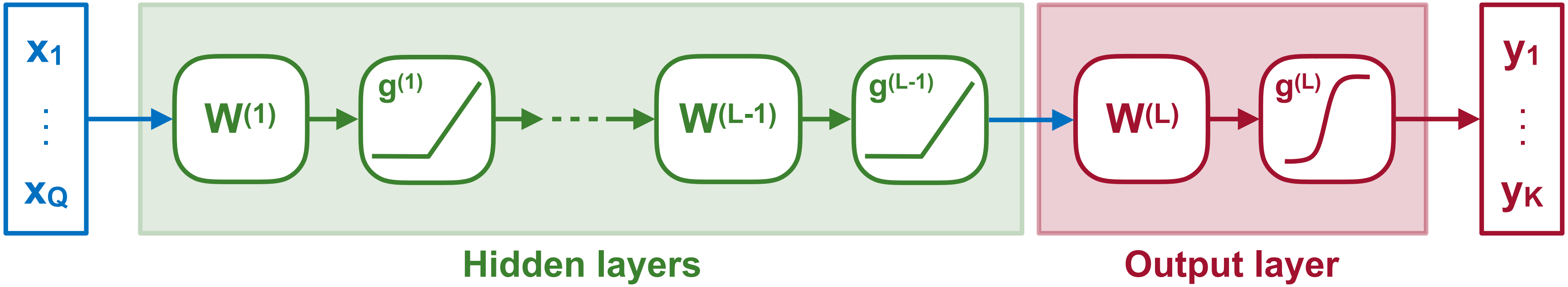
Quest for nonlinear models

- How to get better performance out of machine learning?
  - *Use "more complex" nonlinear models*
  - *Use much more data for training*



Multilayer networks

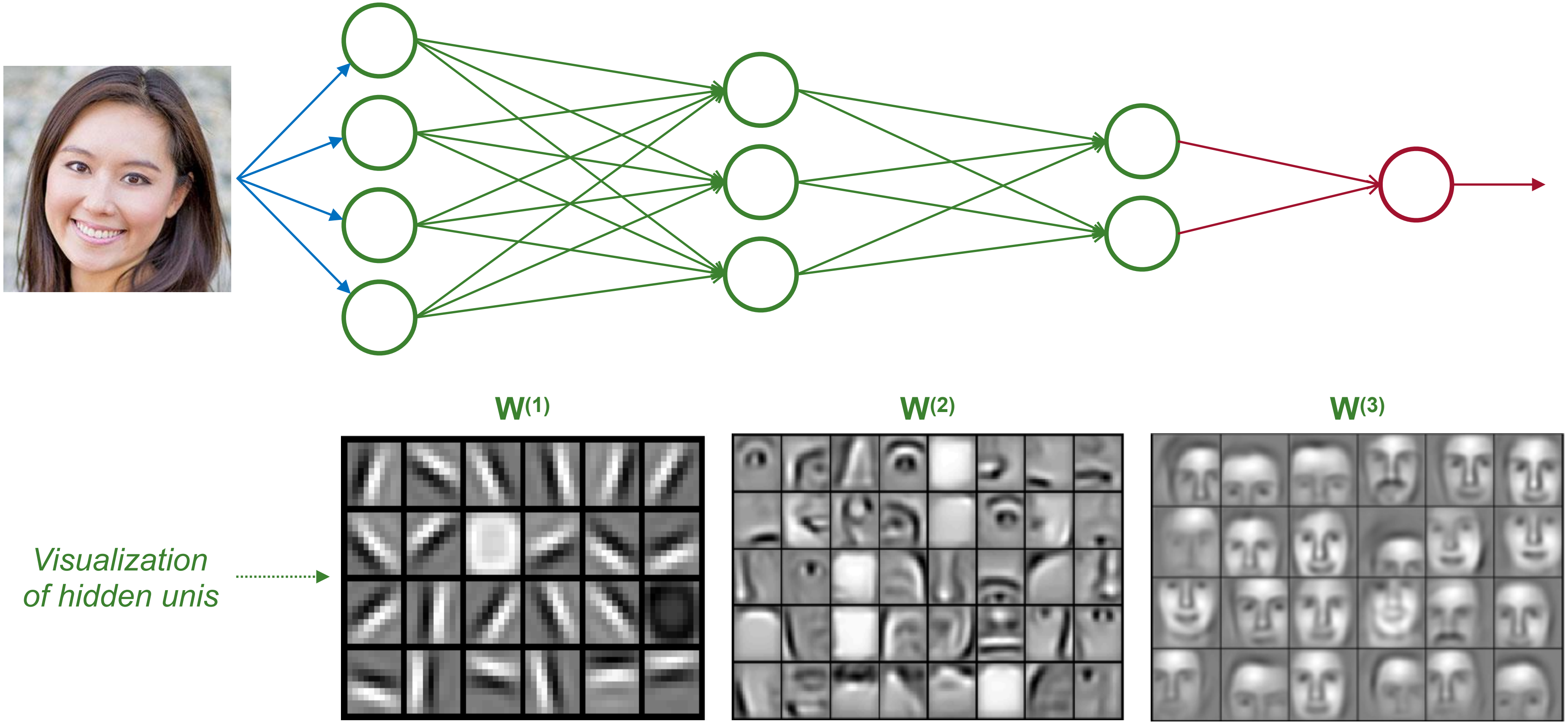
- Architecture of a multilayer neural network



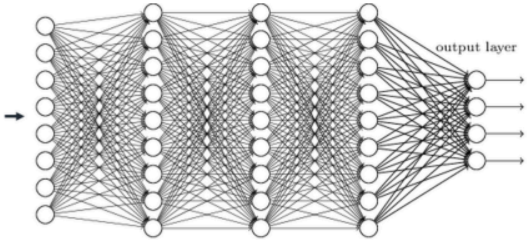
$$f_\theta(\mathrm{x}) = \mathrm{g}^{(L)}\Big(\mathrm{W}^{(L)} \dots \mathrm{g}^{(2)}\Big(\mathrm{W}^{(2)}\mathrm{g}^{(1)}\Big(\mathrm{W}^{(1)}\mathrm{x}\Big)\Big)\Big)$$

Hierarchical representation

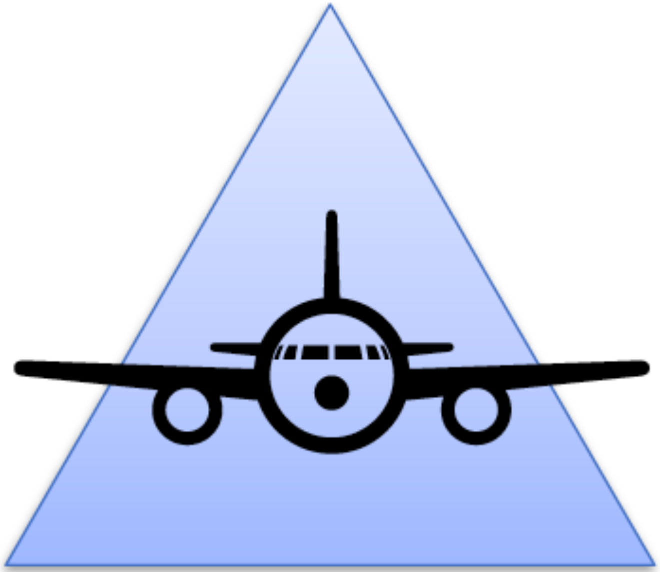
- Multilayer networks can learn a hierarchical representation



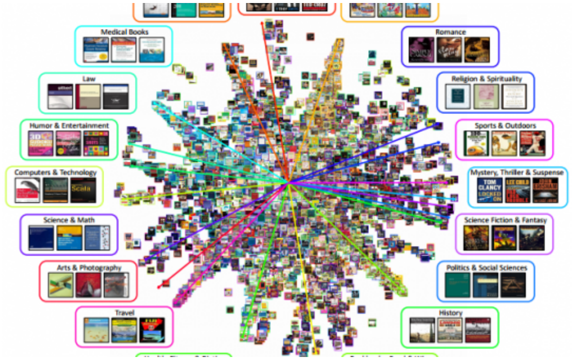
Why are neural networks successful?



High capacity models



Computing power



Lots of training data