MASTER OF SCIENCE IN ENGINEERING

Week 4:
Model Selection, Back-Propagation

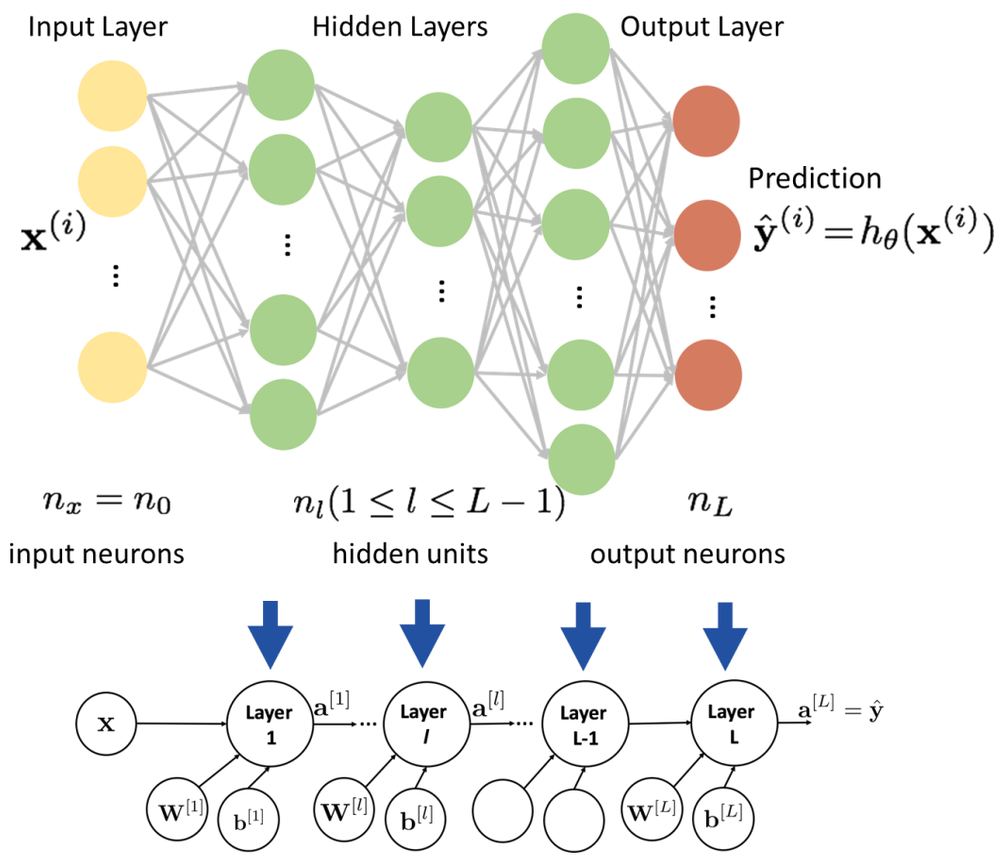
TSM_DeLearn

Andreas Fischer
Klaus Zahn

We are grateful to J. Hennebert and M. Melchior, that they provided their slides

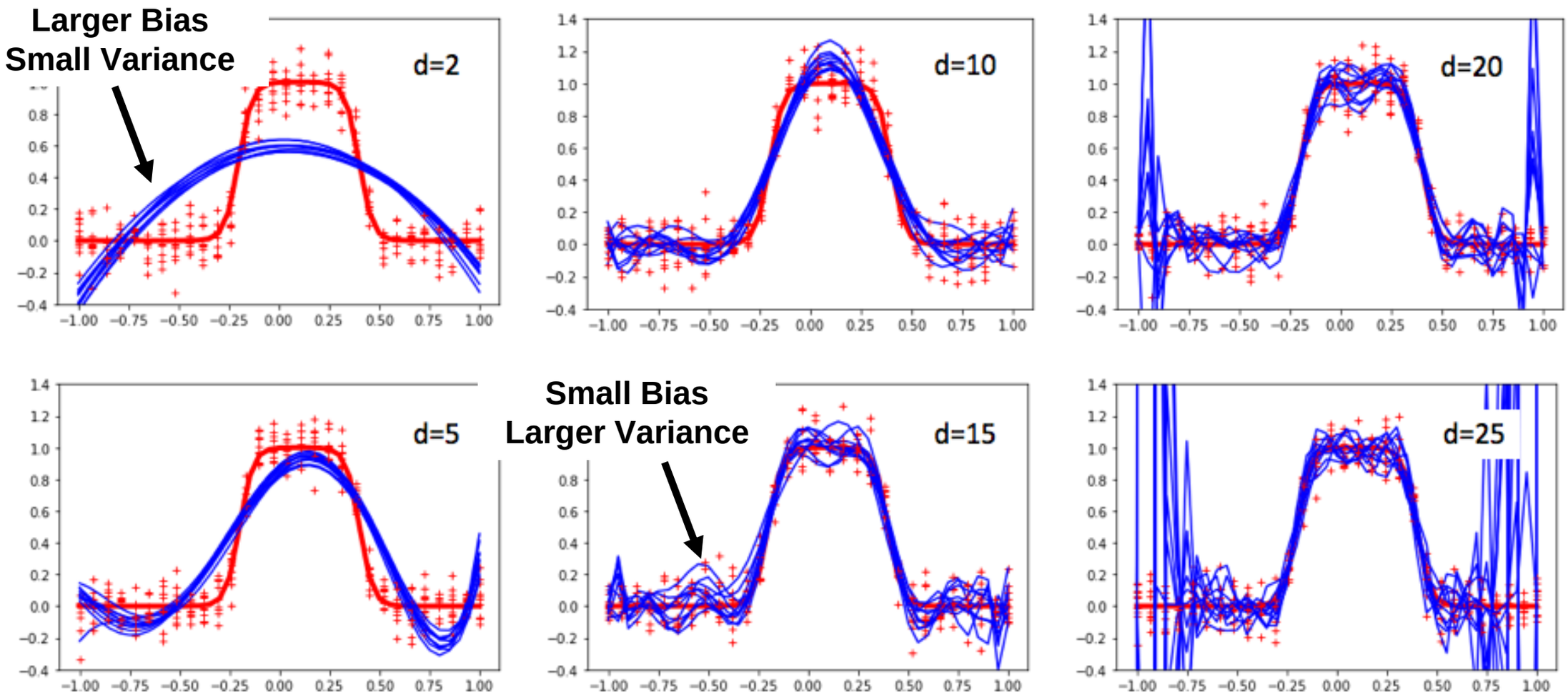
Plan for this week

- Learn the tools how to analyse and optimise generalisation error and overfitting :
  - Hyper-Parameter tuning and model selection
  - Performance measures
- Implement gradient descent for deep networks by exploiting the specific computational structure of neural network models and the rules of calculus to compute derivatives.

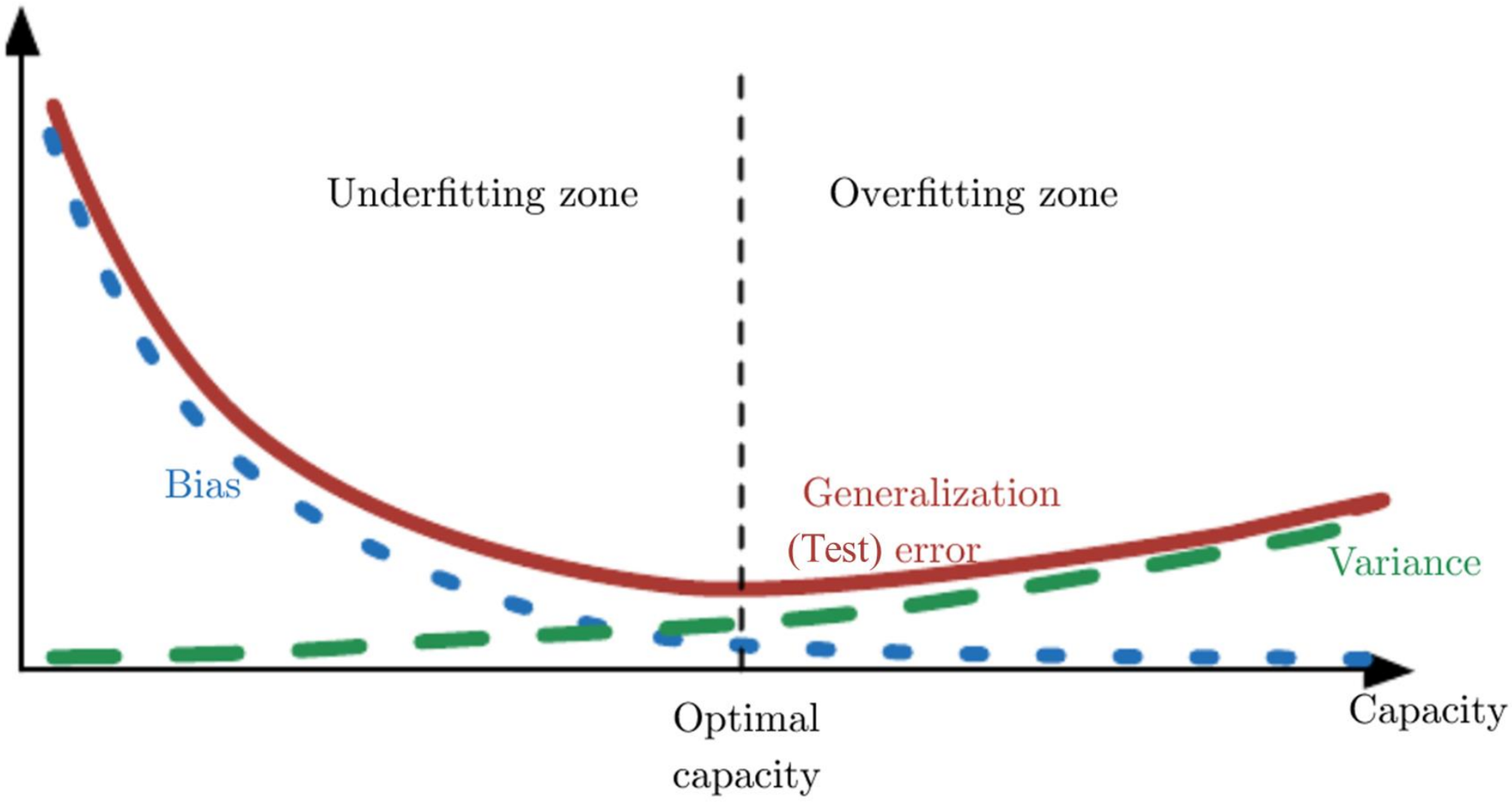


Illustrative Example

Independently generated training sets (with m=30 each

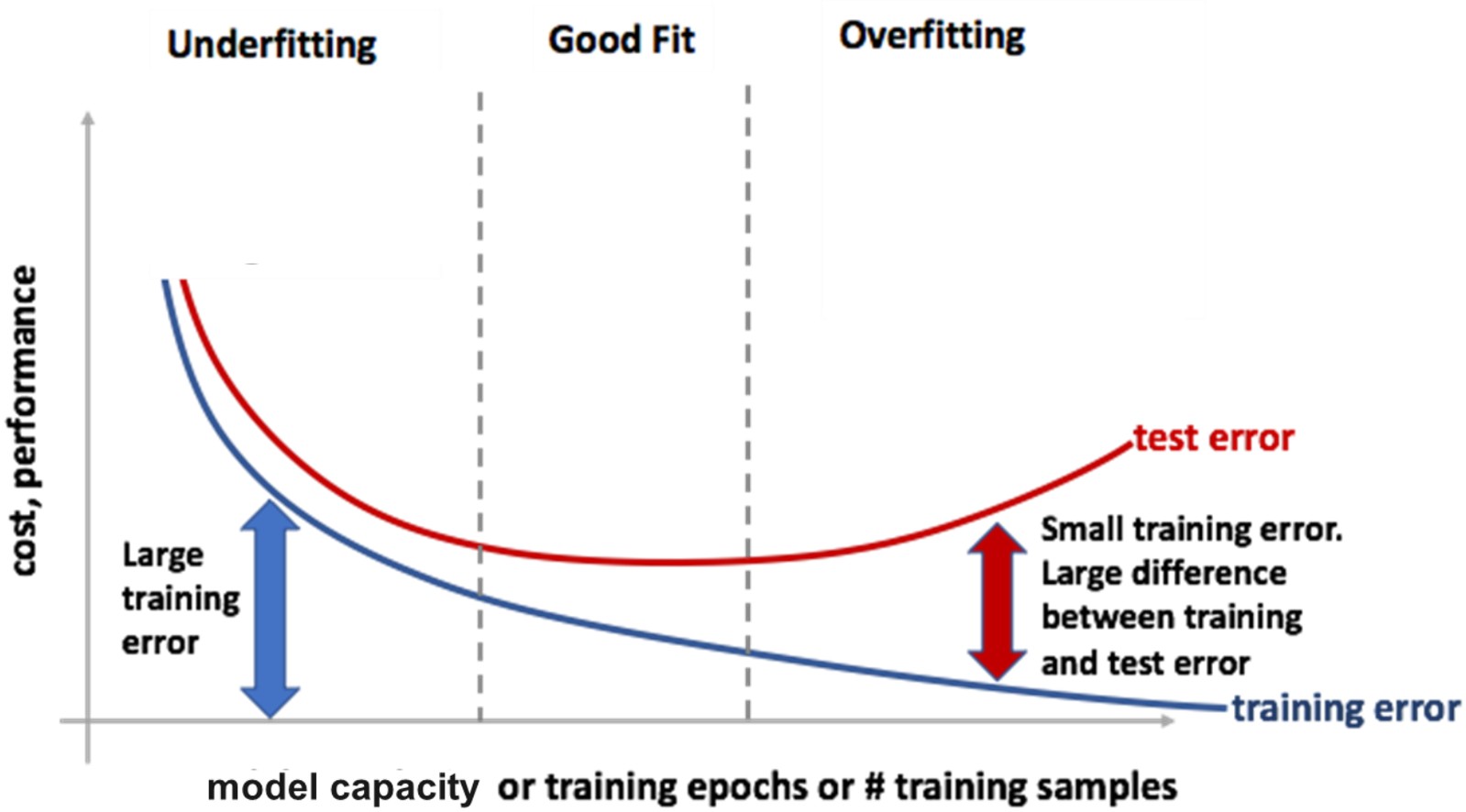


Bias-Variance Tradeoff



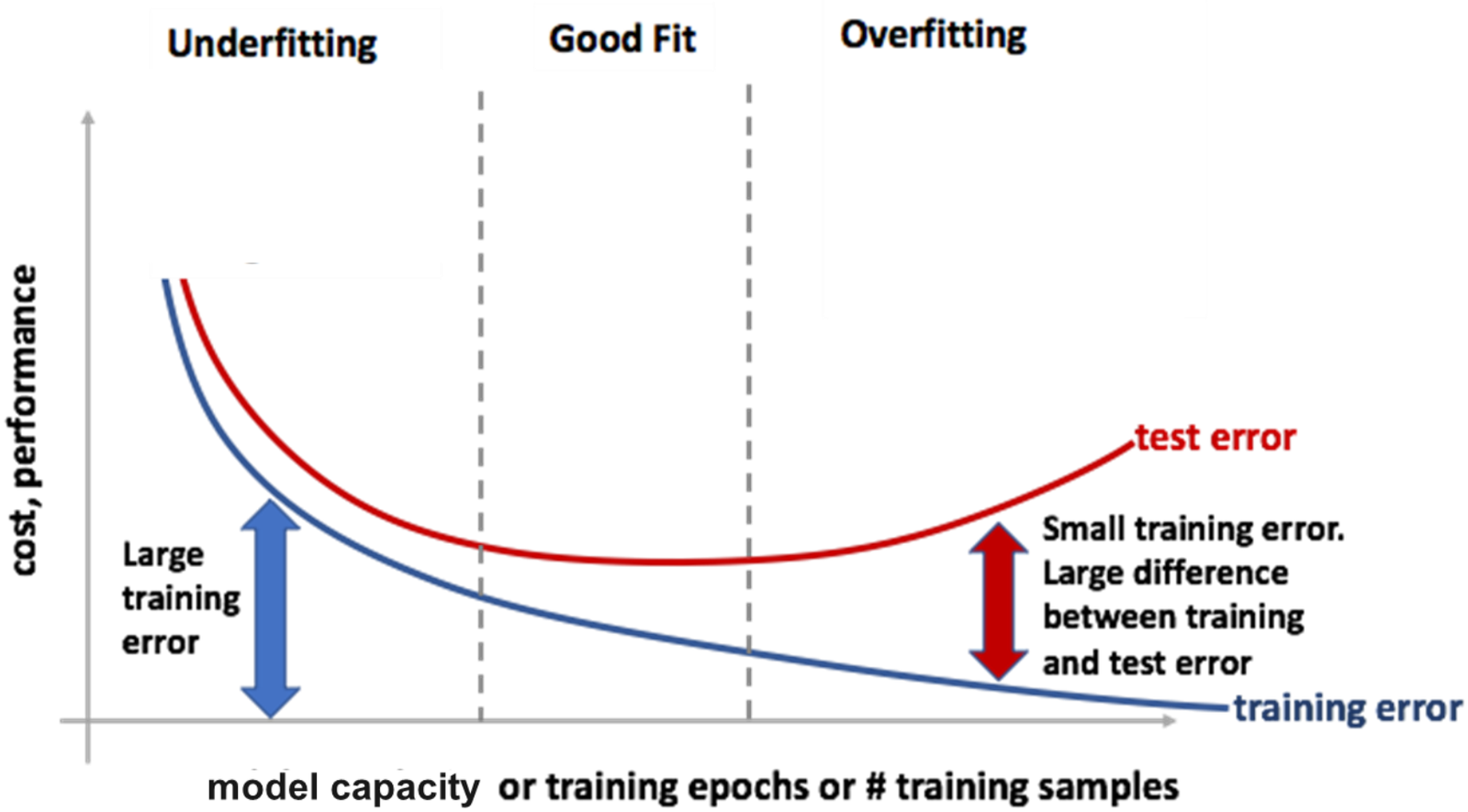
$$\text{MSE} = \mathbf{E}\left[\left(h_{\theta,D} - f\right)^2\right] = \text{bias}(h_\theta)^2 + \text{Var}(h_\theta) + \sigma^2$$

How to Examine Overfitting?



- for increasing model complexities
- for increasing training set sizes
- for increasing number of training epochs

How to Examine Overfitting?

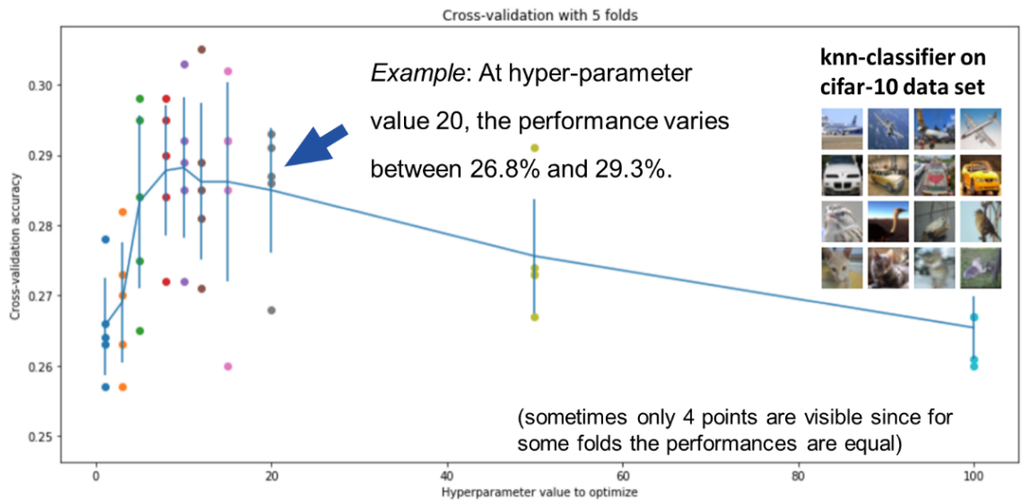


A machine learning algorithm to perform well should be able to make

1. training error (“bias error”) small
2. gap between training and test error (“variance error”) small
3. bias and variance error of comparable magnitude

Model Selection Process

Hyper-Parameter Tuning Cross-Validation



Model Selection

Goal

Select from a family of models the model with the best performance.

Problem

To achieve this goal we need to evaluate the performance for different models (e.g. models with different complexities). But to avoid overfitting on the test set we must not use information from the test set to determine the settings (“hyper-parameters”) of the model.

Solution

Further split the original training set into two subsets:

- Training Set: Used for training
- Validation Set: Used for evaluating the performance and for model selection (selecting hyper parameters)

Hyper-Parameters

Definition

A hyperparameter is a parameter of the learning algorithm itself or a higher-level property of the model that cannot be not determined by optimising the cost on the training set.

Examples

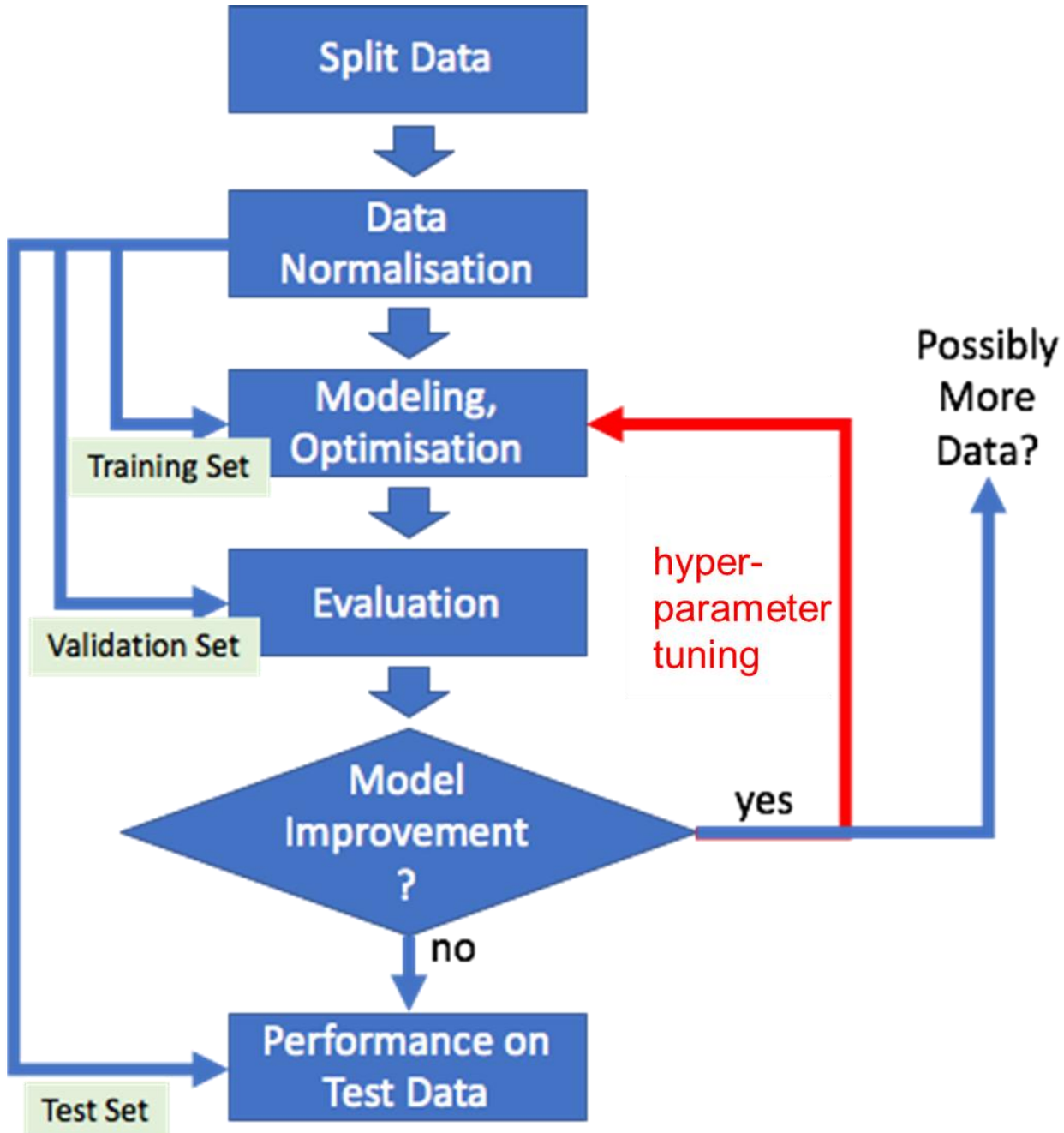
- Learning rate
- Batch size
- Activation function
- Number of hidden layers in a neural network
- Number of neurons in a layer of a neural network
- …etc.

Split Data in Training, Validation, Test Set

Split in three disjoint datasets:

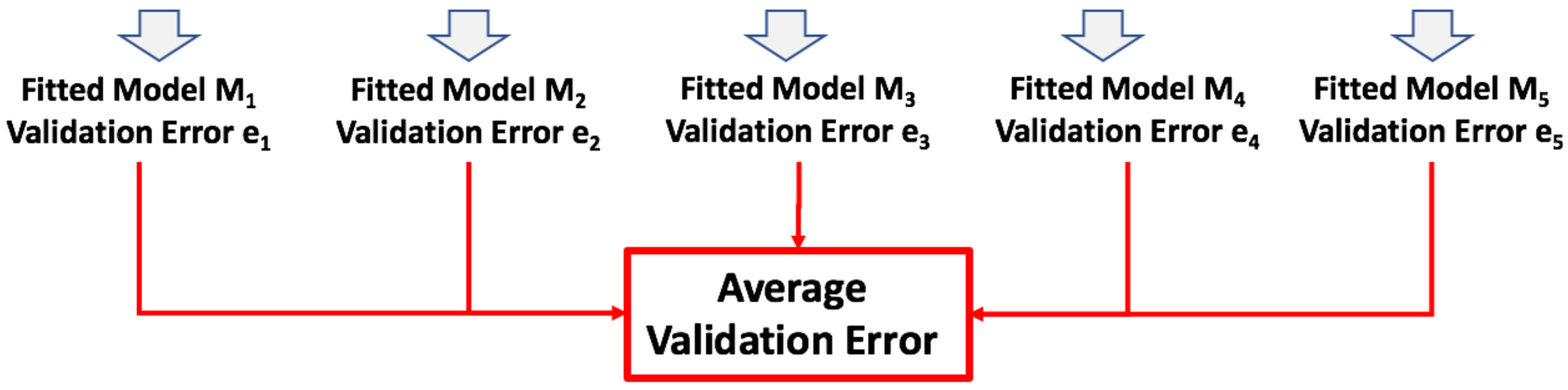
		typical split ratio small datasets	typical split ratio large datasets
Training Set	Subset data used to train the model (choose the parameters that lead to a small cost).	60%	98%
Validation Set	Subset data used to select models, e.g. by selecting the best *hyper-parameters* of the model.	20%	1%
Test Set	Subset data used to measure the performance of the finally selected model.	20%	1%

Standard Process for Machine Learning



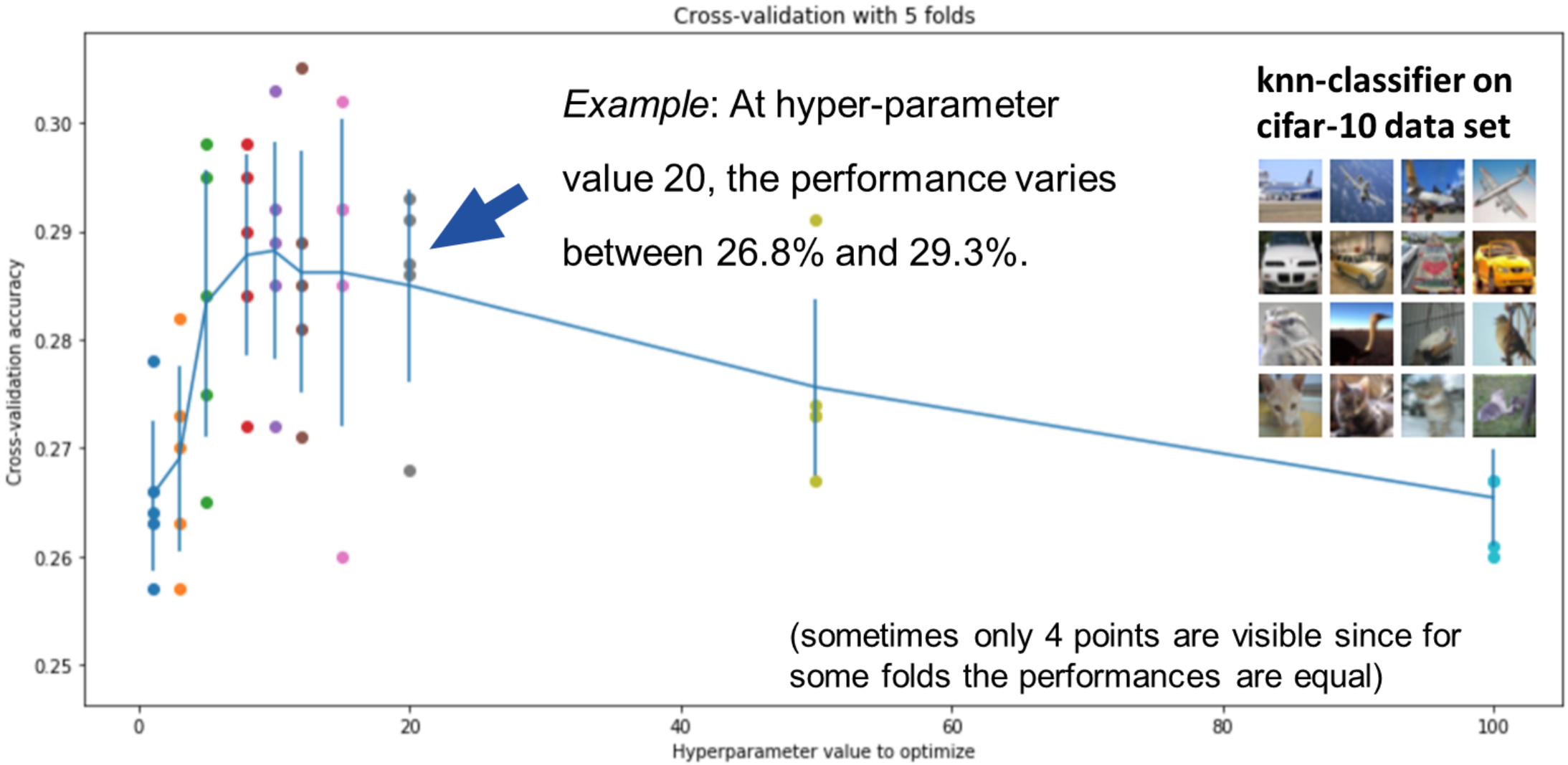
k-fold Cross-Validation (k=5)

Validation	Training	Training	Training	Training
Training	Validation	Training	Training	Training
Training	Training	Validation	Training	Training
Training	Training	Training	Validation	Training
Training	Training	Training	Training	Validation



In addition:
Spread in the validation error of the different folds gives a feeling about the confidence in the validation error estimate.

k-fold Cross-Validation (k=5)



Performance Measures

Confusion Matrix
Error Measures

Explain the terms precision and recall

Per Class Sensitivity (or Recall)

Class Sensitivity or **Recall**

(percentage of correct classification for a given class)

$$\frac{TP}{TP+FN}$$

What fraction of all positives is recognised by the system?

Example

Digit '5'

Actual \ Predicted	P	N	Total
P	1091	206	1297
N	105	12598	12703
Total	1196	12804	14000

Class sensitivity digit '5':

1091 / (1091 + 206) = 84.1%

0	1	2	3	4	5	6	7	8	9
96.8%	96.5%	90.0%	87.9%	93.7%	84.1%	95.8%	94.5%	90.8%	87.0%

Class Precision

Class Precision
(percentage of correct classification in the predicted outputs for a given class)

$$\frac{TP}{TP+FP}$$

What fraction of all positively classified is correctly classified?

Example Digit '5'

Actual	Predicted: P	Predicted: N	Total
P	1091	206	1297
N	105	12598	12703
Total	1196	12804	14000

Class precision for digit '5':

1091 / (1091 + 105) = 91.2%

0	1	2	3	4	5	6	7	8	9
95.7%	95.4%	93.6%	90.8%	89.5%	91.8%	93.7%	91.4%	83.9%	92.0%

Precision and recall curve

Consider a binary classifier with output $h_\theta(x)=\sigma(w\cdot x+b)$. Instead of using the decision rule $h_\theta(x) \geq$ *limit* for *"true"* with *limit* = 0.5 we may choose *limit* from the full interval [0,1] to obtain a so-called precision-recall-curve. Sketch a typical precision-recall-curve.

Precision and recall curve

Describe typical situations where you would like to obtain a high recall:

Describe typical situations where you would like to obtain a high precision:

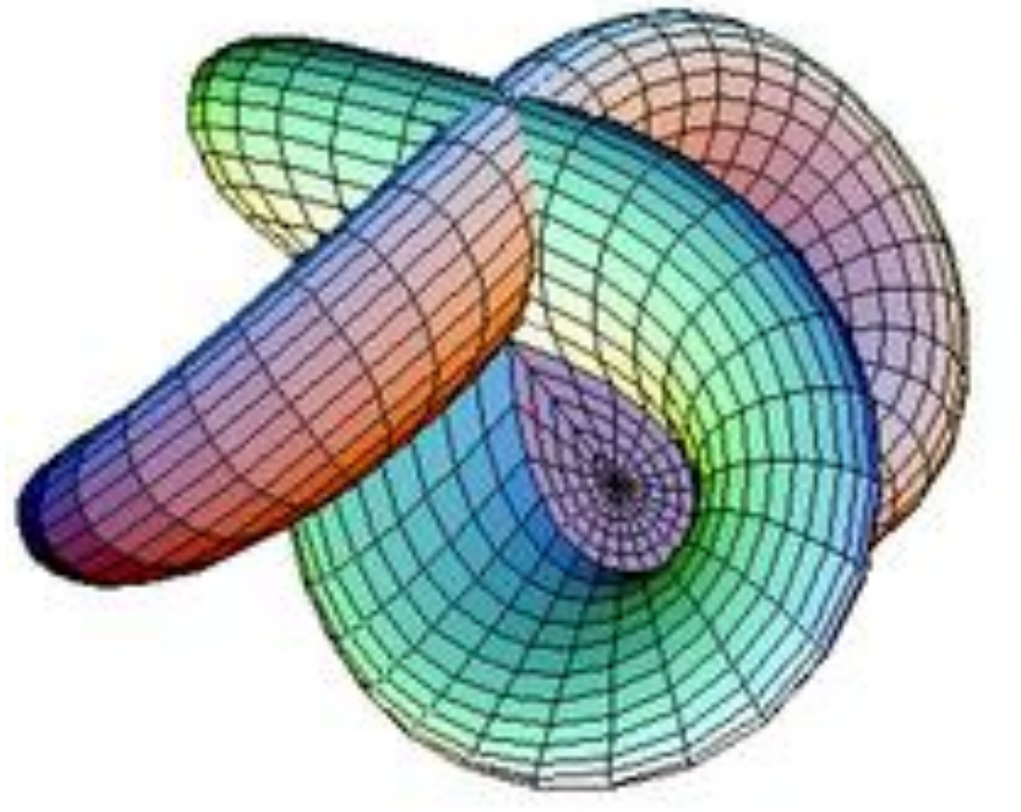
Precision and recall curve

When increasing the parameter limit in question 2. from 0 to 1, does

- the recall increase or decrease?
- the precision increase or decrease?
- the recall change monotonically?
- the precision change monotonically?

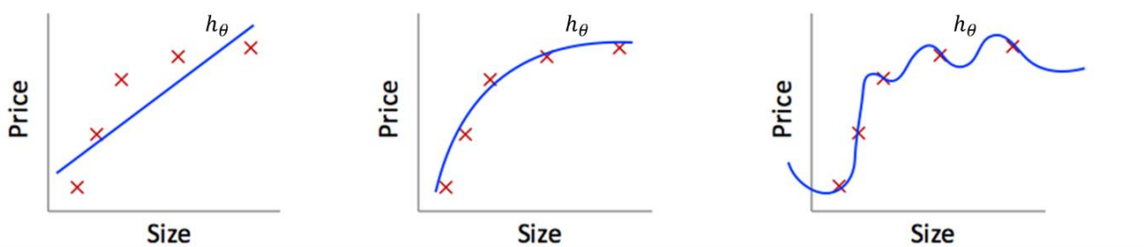
Deep Learning and Curse of Dimensionality

- Typical Dimensionality in Deep Learning Problems
- Hierarchy of Concepts



Local Smoothness Assumption in Classical ML

- The function to approximate does not vary much locally and stays roughly constant in small regions.
- This allows to generalise to variations in small regions from closely located training examples.
  A strategy adopted e.g., in clustering (k-means), k-nearest neighbours, decision trees, etc. and shallow neural nets.
- This implies that training examples are needed in all the regions of interest. If no or only few examples are available in a given region, no (confident) predictions can be made.



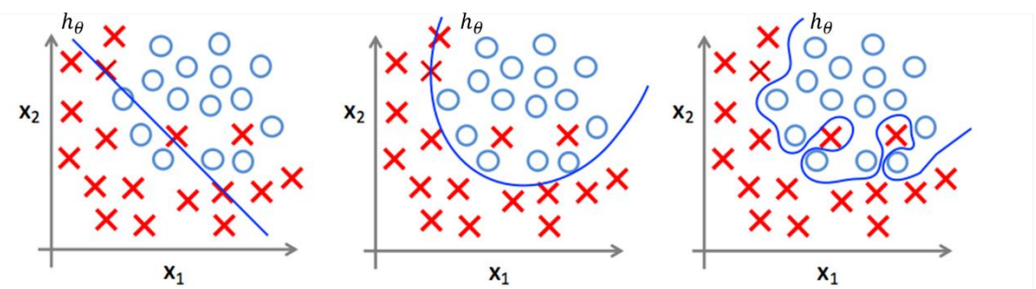
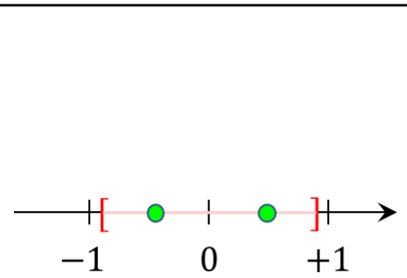
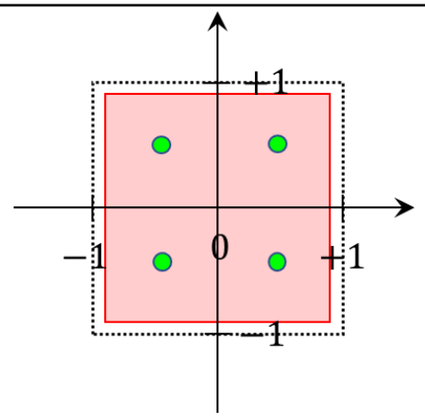
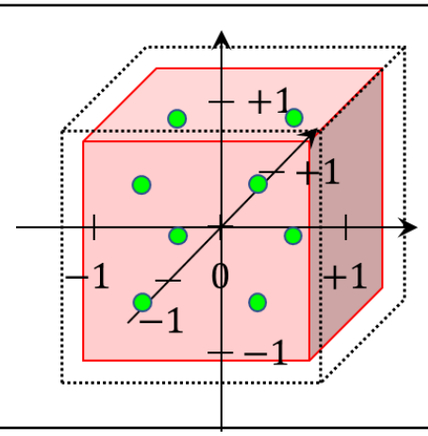


Illustration Curse of Dimensionality

$d = 1$	$d = 2$	$d = 3$	$d = 784$
			
$N = 2$	$N = 4$	$N = 8$	$N = 2^{784}$ $\approx 10^{236}$
$\frac{2}{n}$	$\frac{2}{\sqrt{n}}$	$\frac{2}{\sqrt[3]{n}}$	$\frac{2}{\sqrt[784]{n}} \approx^{1} 1.97$ [1]for $n = 70'000$
0.99	0.99^2	0.99^3	0.99^{784} ≈ 0.0004

For DL Local Smoothness Assumption fails because of Curse of Dimensionality?

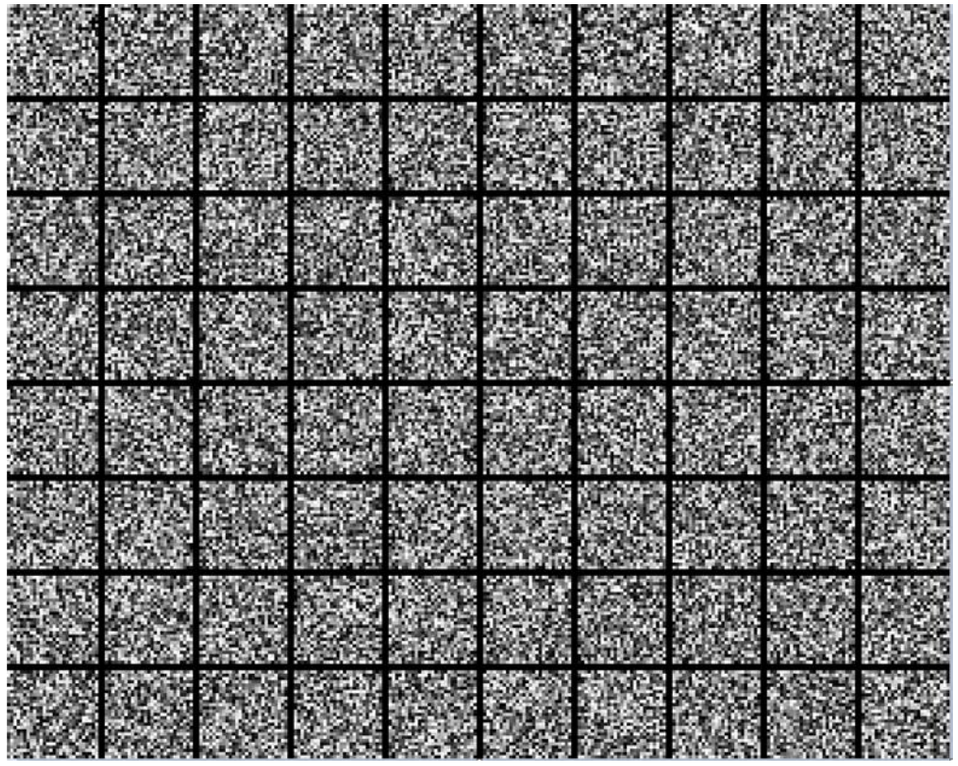
Coined by Richard E. Bellmann (1961)

Wikipedia (https://en.wikipedia.org/wiki/Curse_of_dimensionality):

> *"… when the dimensionality increases, the volume of the space increases so fast that the available data become sparse. This sparsity is problematic for any method that requires statistical significance. In order to obtain a statistically sound and reliable result, the amount of data needed to support the result often grows exponentially with the dimensionality."*

But why does DL work?

$\approx 10^{??}$

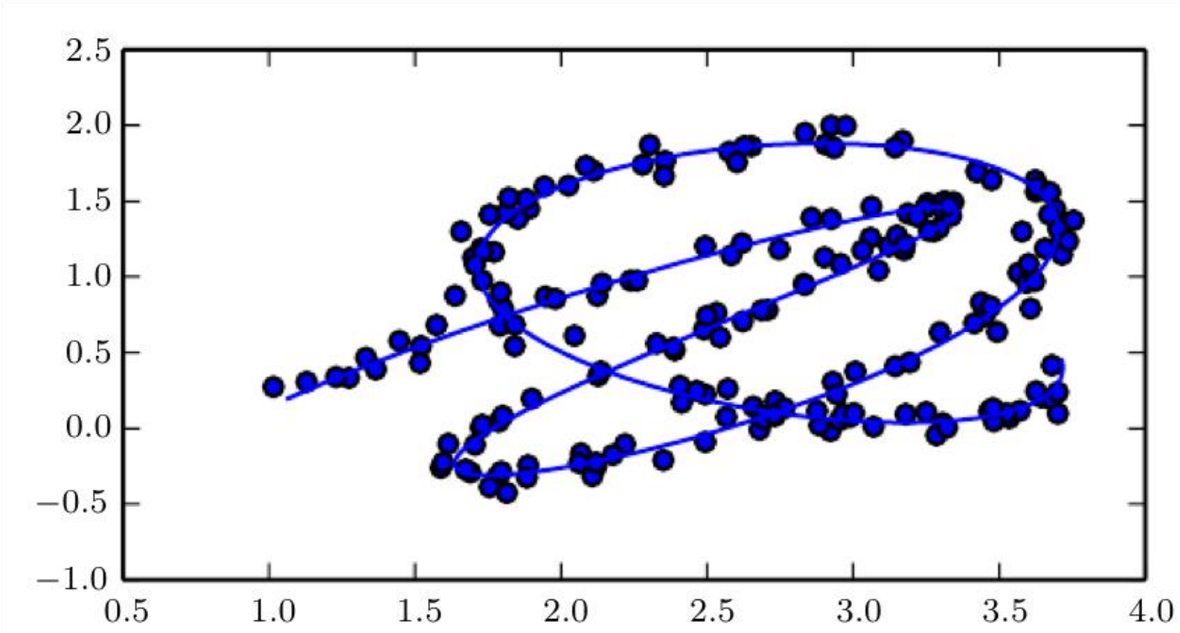


$256^{784} \approx 10^{1888}$

Typically, the patterns found in the data and its variational degrees of freedom (yet large!) are expected to be well represented on a (much) lower dimensional manifold.

Lower Dimensional Manifolds

Typically, the interesting information is concentrated on some lower dimensional manifold (with much less degrees of freedom).



Illustration

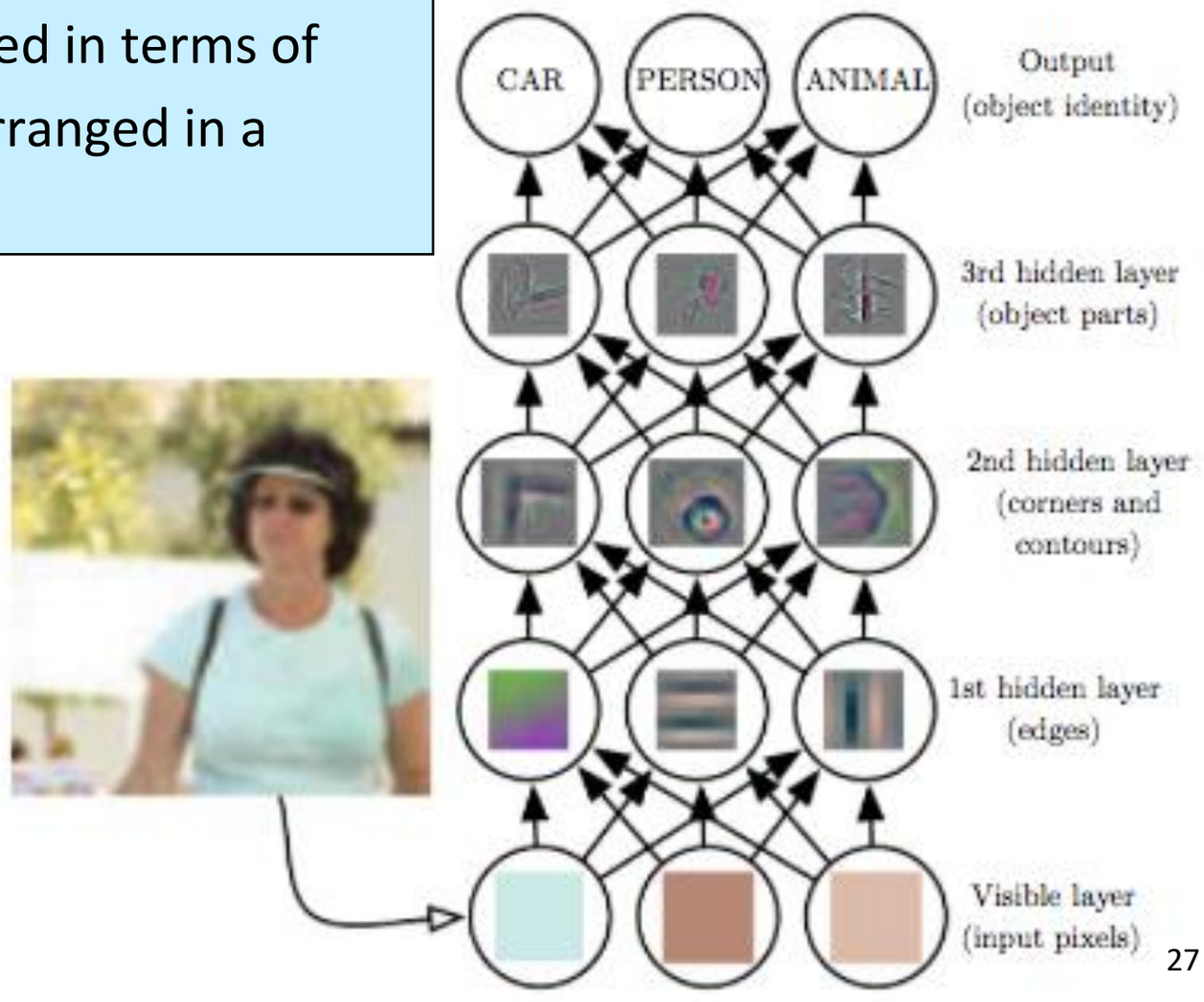
The data points lie near a lower-dimensional manifold (1d) embedded in the original input space (2d).

Local smoothness assumption may hold on the sub-manifold, but not in the original input space.

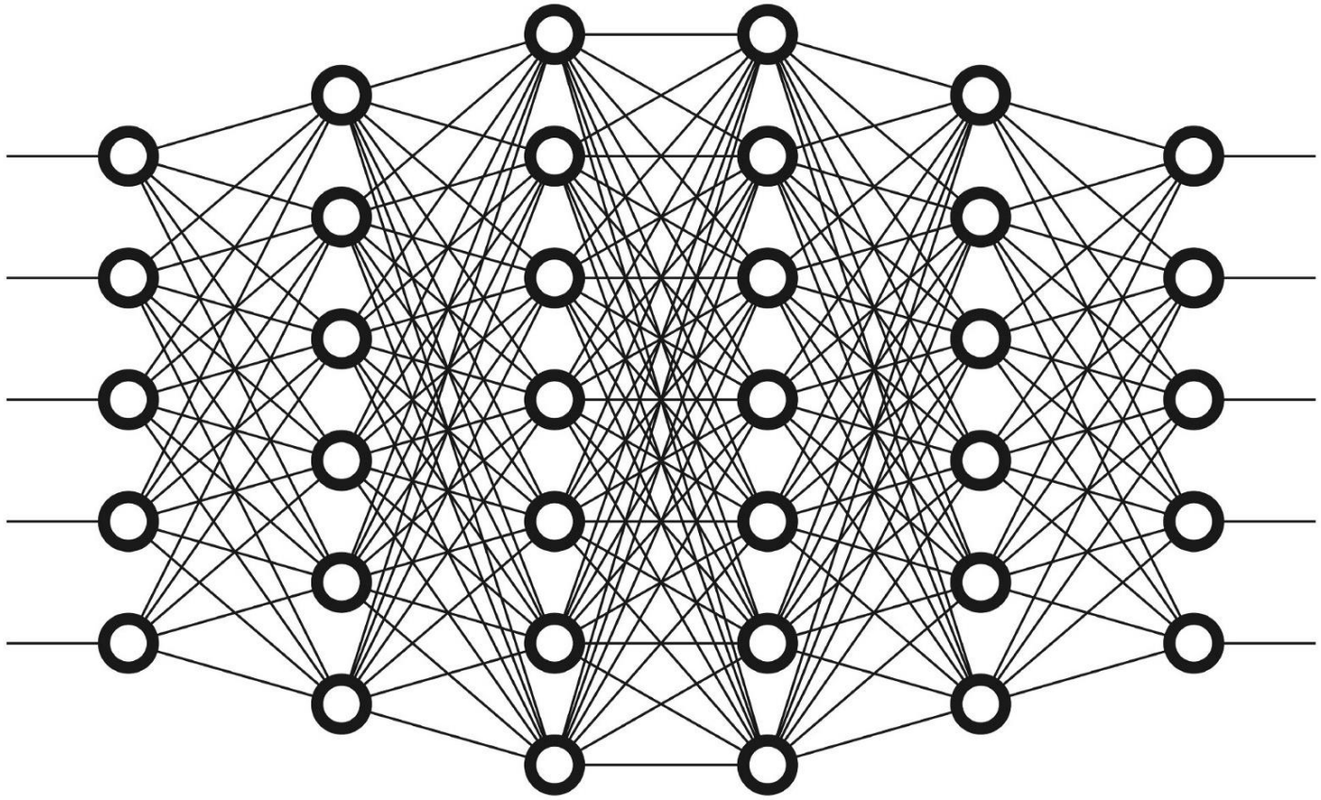
Strong prior assumptions are needed to learn the manifold.

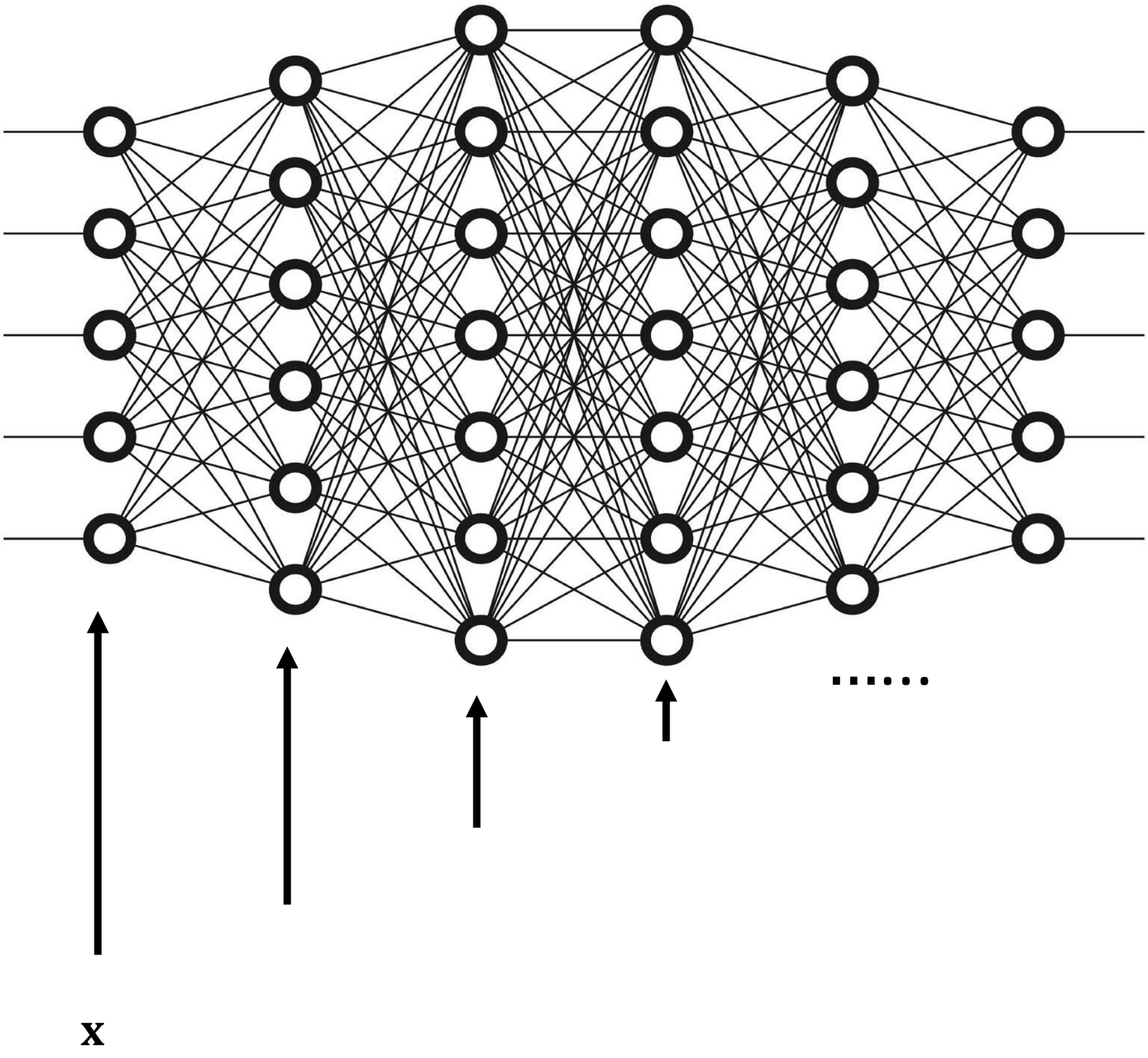
Composition of Features at Multiple Levels in a Hierarchy

- we assume that data was generated by composition of factors/features at multiple levels in a hierarchy;
- learning then involves discovering a set of underlying factors of variation that can be described in terms of other, simpler underlying factors, ... (arranged in a hierarchy).

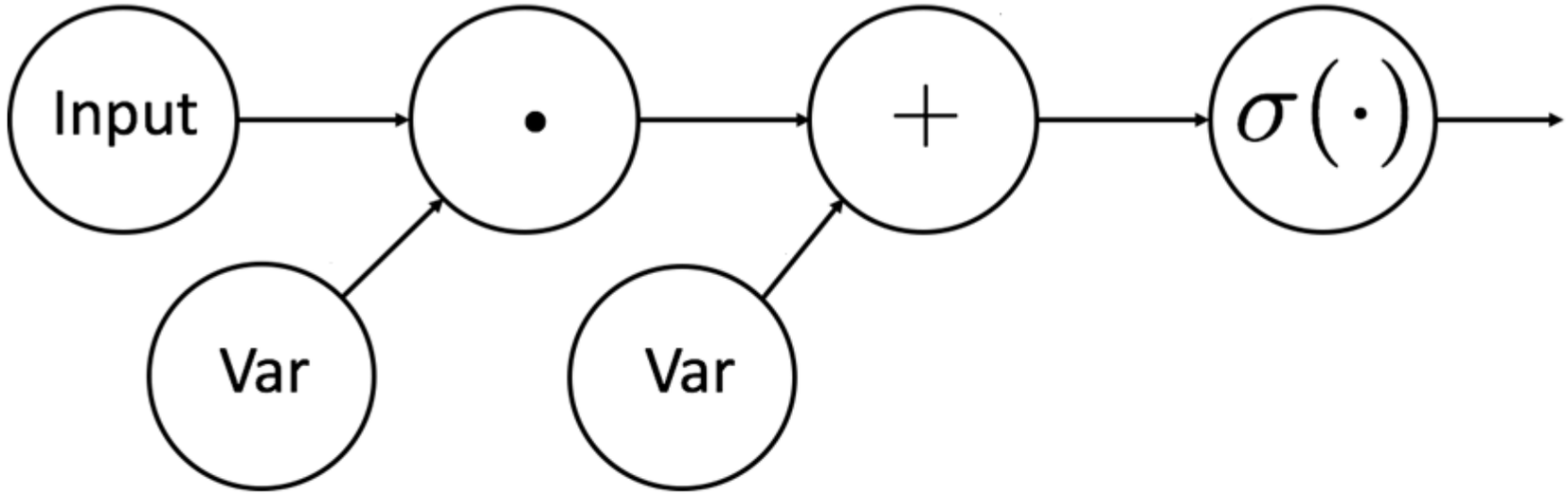


Computational Graphs




.....
x

Computational Graphs



Computational Graphs

A computational graph is a directed graph where

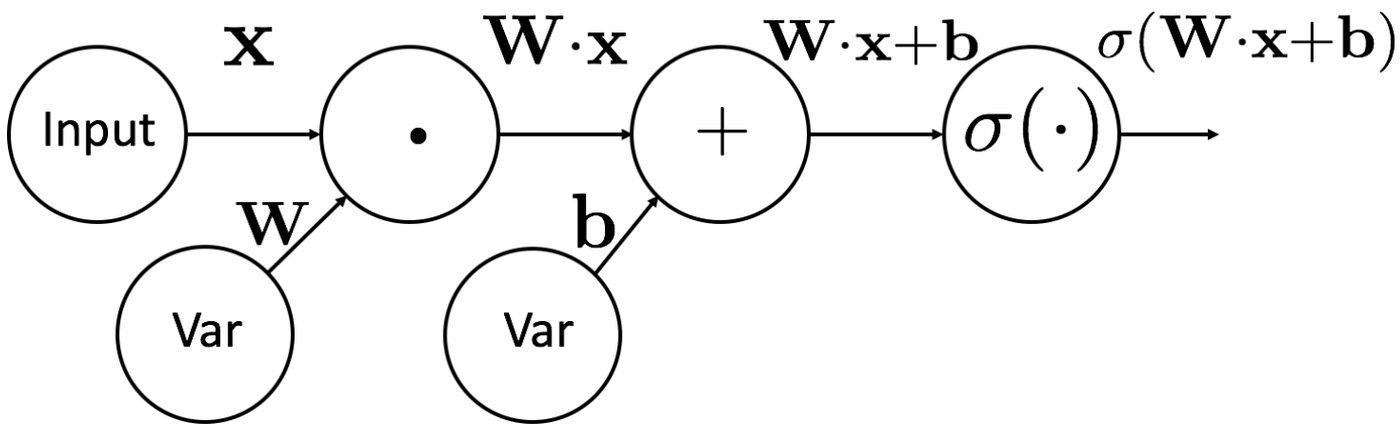
- *nodes* correspond to operations or input variables
- *edges* correspond to inputs of an operation which can originate from input variables or outputs of other operations.

Two types of input variables: Input data and model parameters.

Example

Single Layer of MLP $g(\mathbf{x};\mathbf{W};\mathbf{b}) = \sigma(\mathbf{W}\cdot\mathbf{x}+\mathbf{b})$

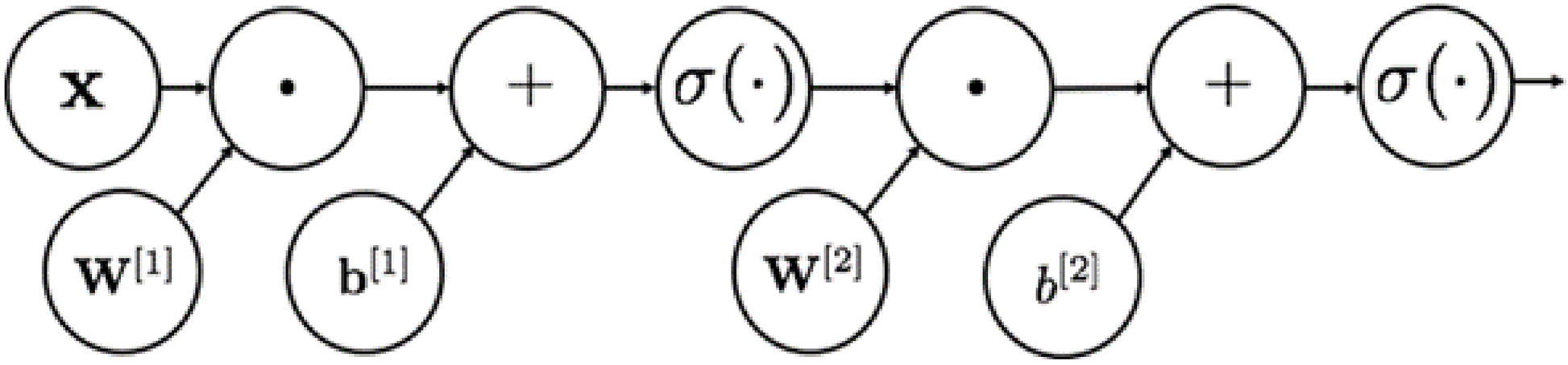
$\mathbf{x}$: n_x x 1
$\mathbf{W}$: n_1 x n_x
$\mathbf{b}$: n_1 x 1



Computational Graphs

Example Computational graph for a network with a hidden layer and an output layer

$$g\left(\mathbf{x};\mathbf{W}^{[1]};\mathbf{b}^{[1]};\mathbf{W}^{[2]};\mathbf{b}^{[2]}\right)=\sigma\left(\mathbf{W}^{[2]}\cdot\sigma\left(\mathbf{W}^{[1]}\cdot\mathbf{x}+\mathbf{b}^{[1]}\right)+\mathbf{b}^{[2]}\right)$$



input $\mathbf{x}$: n_x x 1

1st hidden layer $\mathbf{W}^{[1]}$: n_1 x n_x

$\mathbf{b}^{[1]}$: n_1 x 1

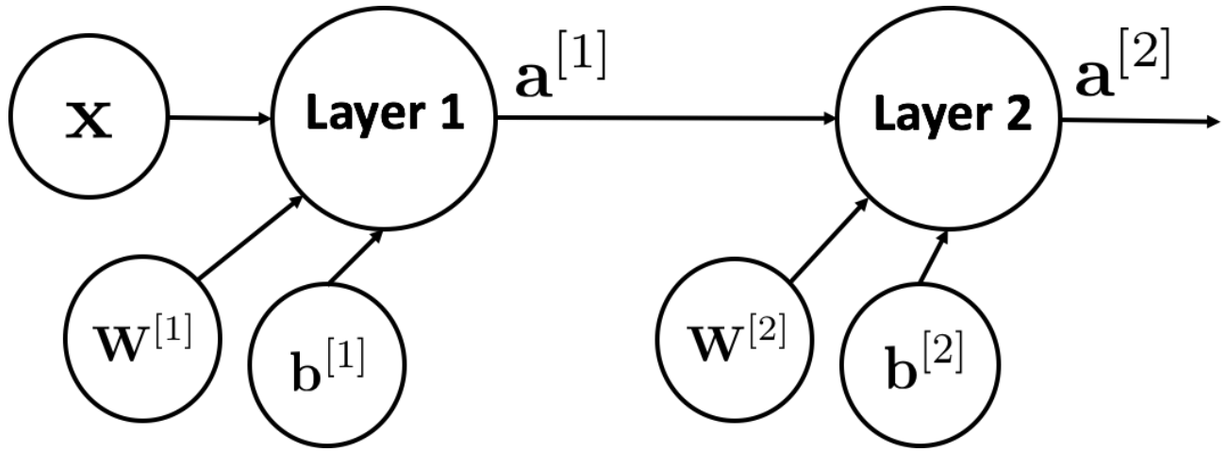
2nd hidden layer $\mathbf{W}^{[2]}$: n_2 x n_1

$\mathbf{b}^{[2]}$: n_2 x 1

Graphs Composed of Layers

The operations of a single layer (affine transformation and application of activation function) can be collapsed into a single node:

$$\mathbf{a}^{[l]} = \sigma\left(\mathbf{W}^{[l]} \cdot \mathbf{x} + \mathbf{b}^{[l]}\right)$$



Layers are the abstraction level most DL frameworks use to define the architecture of a DL model.

Example

`tensorflow 2.0/keras api`

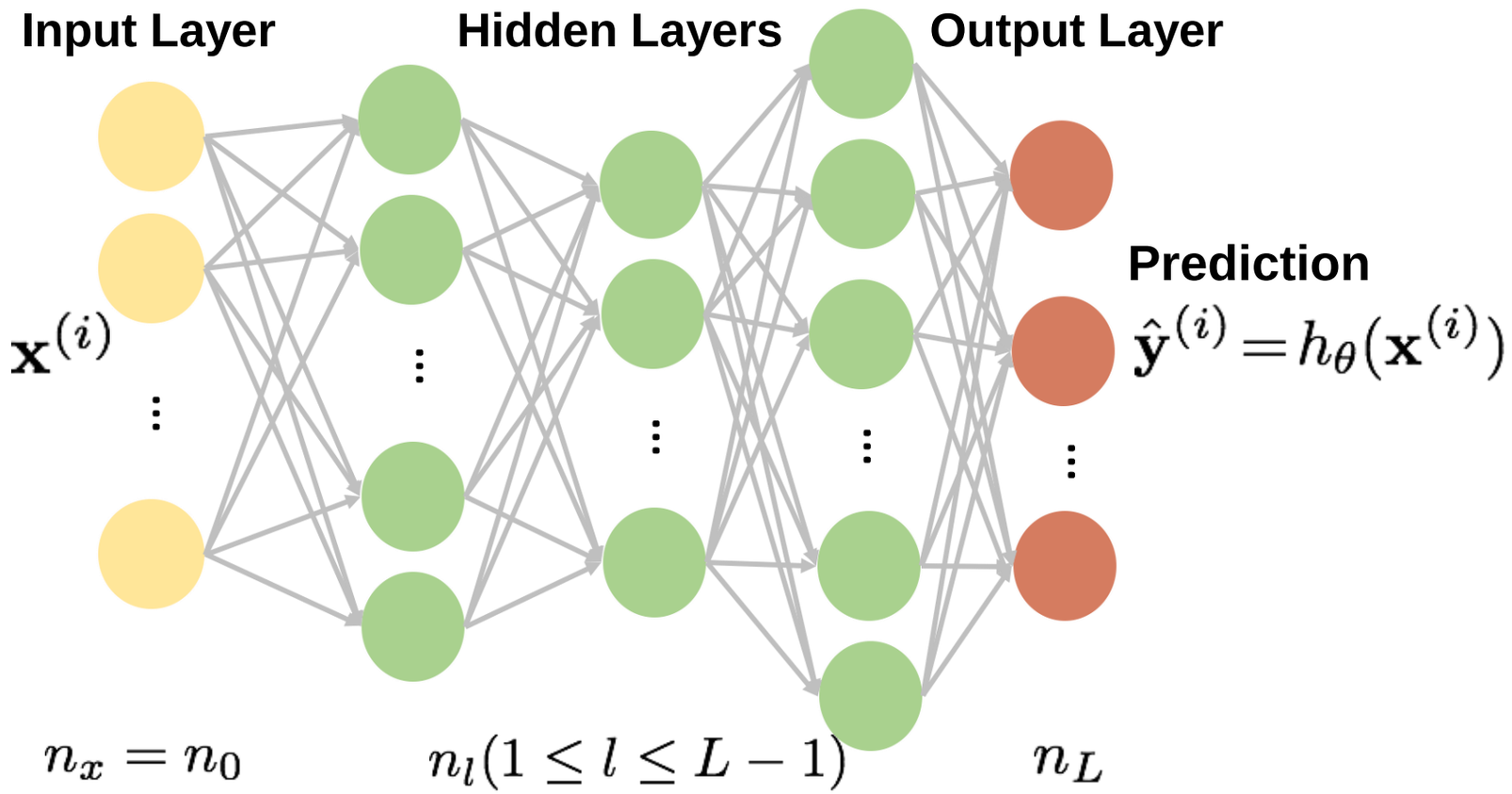
Fully connected perceptron layer instantiated as **`Dense`**

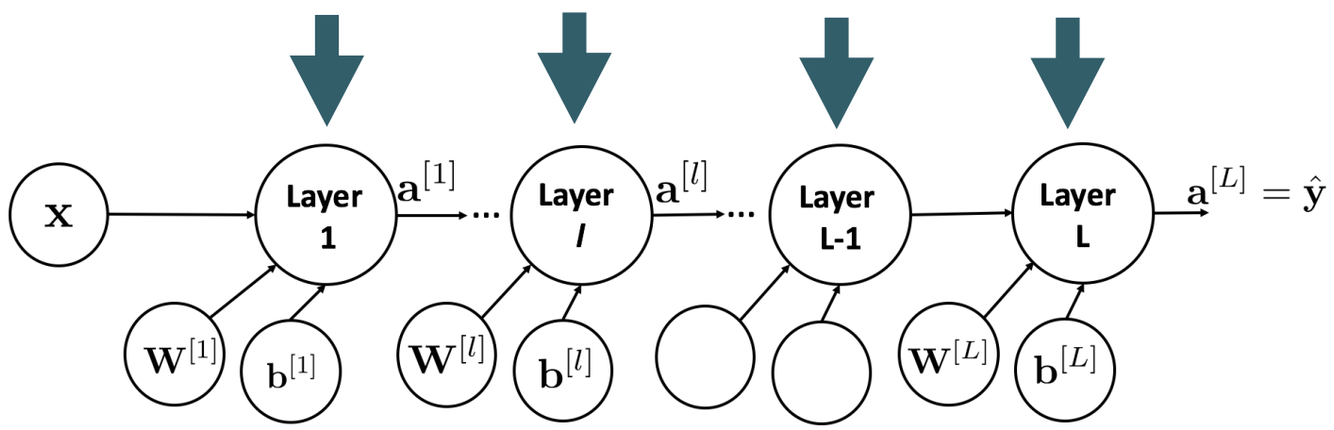
```
import tensorflow as tf
mnist = tf.keras.datasets.mnist

(x_train, y_train),(x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(512, activation=tf.nn.relu),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])
```

Representation of MLP as Computational Graph

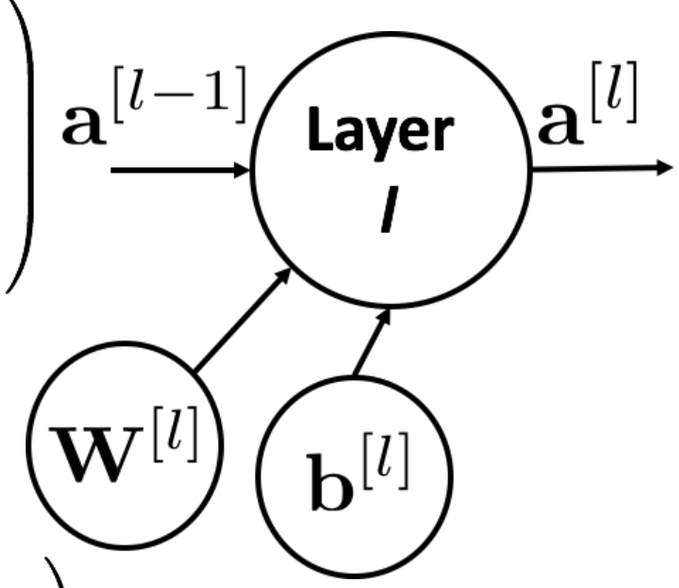




Notations for Layer in an MLP

activations of layer *'l -1'* (column vector with n_{l-1}elements) define the dynamic input for layer *'l'*

$$\mathbf{a}^{[l-1]} = \begin{pmatrix} a_1^{[l-1]} \\ a_2^{[l-1]} \\ \vdots \\ a_{n_{l-1}}^{[l-1]} \end{pmatrix}$$



$$\mathbf{a}^{[l]} = \begin{pmatrix} a_1^{[l]} \\ a_2^{[l]} \\ \vdots \\ a_{n_l}^{[l]} \end{pmatrix}$$

activations of layer *l* (column vector with n_l elements) define the dynamic output of layer 'l'

$$\mathbf{W}^{[l]} = \begin{pmatrix} w_{11} & \dots w_{1n_{l-1}} \\ w_{21} & \dots w_{2n_{l-1}} \\ \vdots & \\ w_{n_l 1} & \dots w_{n_l n_{l-1}} \end{pmatrix}$$

$n_l \times n_{l-1}$ matrix

$$\mathbf{b}^{[l]} = \begin{pmatrix} b_1^{[l]} \\ b_2^{[l]} \\ \vdots \\ b_{n_l} \end{pmatrix}$$

n_l-vector

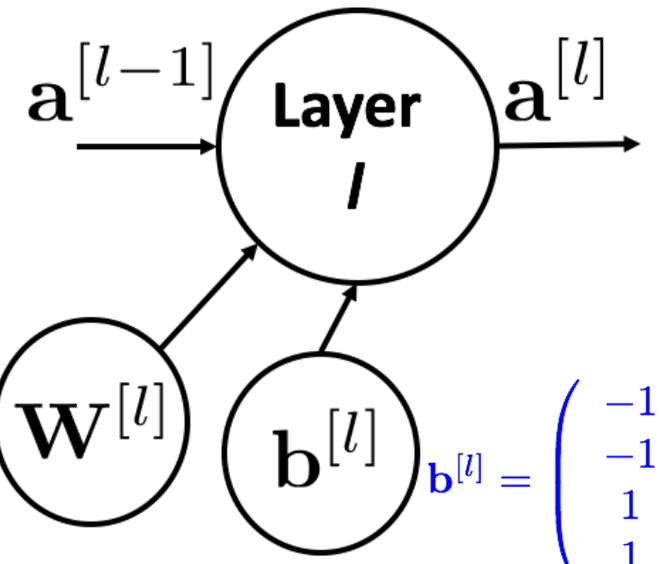
Operation in an (MLP) layer

$$\begin{aligned} \mathbf{z}^{[l]} &= \mathbf{W}^{[l]} \cdot \mathbf{a}^{[l-1]} + \mathbf{b}^{[l]} \\ \mathbf{a}^{[l]} &= g^{[l]}(\mathbf{z}^{[l]}) \end{aligned}$$

$g^{[l]}(\cdot)$: activation function for layer *'l'* (applied element-wise).

Example: Notations for Layer in an MLP

$n_{l-1} = 3$ $\quad \mathbf{a}^{[l-1]} = \begin{pmatrix} 1 \\ -1 \\ -0.5 \end{pmatrix}$ 3×1



$\mathbf{a}^{[l]} = \begin{pmatrix} 0.78 \\ 0.50 \\ 0.56 \\ 0.18 \end{pmatrix}$ 4×1 $\quad n_l = 4$

$\mathbf{W}^{[l]} = \begin{pmatrix} 1 & -1 & -0.5 \\ 1 & 0.5 & -1 \\ -0.2 & 0.3 & 0.5 \\ -1 & 1 & 1 \end{pmatrix}$ 4×3

$\mathbf{b}^{[l]} = \begin{pmatrix} -1 \\ -1 \\ 1 \\ 1 \end{pmatrix}$ 4×1

$g^{[l]}(\cdot) = \sigma(\cdot)$:
sigmoid activation function

$$\begin{aligned} \mathbf{z}^{[l]} &= \mathbf{W}^{[l]} \cdot \mathbf{a}^{[l-1]} + \mathbf{b}^{[l]} \\ \mathbf{a}^{[l]} &= g^{[l]}(\mathbf{z}^{[l]}) \end{aligned}$$

$$\mathbf{z}^{[l]} = \mathbf{W}^{[l]} \cdot \mathbf{a}^{[l-1]} + \mathbf{b}^{[l]} = \begin{pmatrix} 1.25 \\ 0. \\ 0.25 \\ -1.5 \end{pmatrix}$$

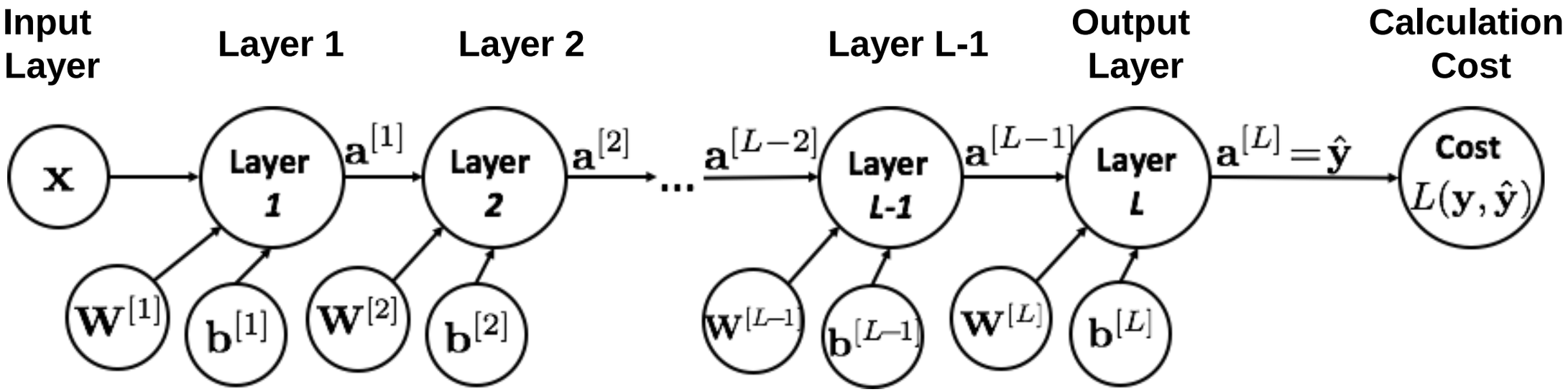
$$\mathbf{a}^{[l]} = \sigma(\mathbf{z}^{[l]}) = \begin{pmatrix} 0.78 \\ 0.50 \\ 0.56 \\ 0.18 \end{pmatrix}$$

4×1

code snippet in numpy

```
import numpy as np
W = np.array([[1, -1, -0.5],[1, 0.5, -1], [-0.2, 0.3, 0.5],[-1, 1, 1]]).reshape(4,3)
b = np.array([-1, -1, 1, 1]).reshape(4,1)
aprev = np.array([1, -1, -0.5]).reshape(3,1)
z = np.matmul(W,aprev)+b
a = 1./(1.0+np.exp(-z))
```

Forward Propagation — Chaining Calculation of Activations in MLP



$$\mathbf{z}^{[1]} = \mathbf{W}^{[1]} \cdot \mathbf{a}^{[0]} + \mathbf{b}^{[1]}$$
$$\mathbf{a}^{[1]} = g^{[1]}(\mathbf{z}^{[1]})$$

$$\mathbf{z}^{[2]} = \mathbf{W}^{[2]} \cdot \mathbf{a}^{[1]} + \mathbf{b}^{[2]}$$
$$\mathbf{a}^{[2]} = g^{[2]}(\mathbf{z}^{[2]})$$

$$\mathbf{z}^{[L-1]} = \mathbf{W}^{[L-1]} \cdot \mathbf{a}^{[L-2]} + \mathbf{b}^{[L-1]}$$
$$\mathbf{a}^{[L-1]} = g^{[L-1]}(\mathbf{z}^{[L-1]})$$

$$\mathbf{z}^{[L]} = \mathbf{W}^{[L]} \cdot \mathbf{a}^{[L-1]} + \mathbf{b}^{[L]}$$
$$\mathbf{a}^{[L]} = g^{[L]}(\mathbf{z}^{[L]})$$

$$L(\hat{\mathbf{y}}, \mathbf{y})$$

(single sample)

Computing the Output of MLP – for Batch

Vectorise over the samples in a (mini-)batch. This will allow for efficient parallelisation on GPU. Put the input samples $\mathbf{x}^{(i)}$ as columns in a $n_x \times m$ matrix:

$$\mathbf{X} = \mathbf{A}^{[0]} = \begin{pmatrix} \vdots & \vdots & & \vdots \\ \mathbf{x}^{(1)} & \mathbf{x}^{(2)} & \dots & \mathbf{x}^{(m)} \\ \vdots & \vdots & & \vdots \end{pmatrix}$$

Similarly, for the activations and the logits in layer $l = 1, \dots, L$ ($n_l \times m$-matrices):

$$\mathbf{A}^{[l]} = \begin{pmatrix} \vdots & \vdots & & \vdots \\ \mathbf{a}^{[l](1)} & \mathbf{a}^{[l](2)} & \dots & \mathbf{a}^{[l](m)} \\ \vdots & \vdots & & \vdots \end{pmatrix} \qquad \mathbf{Z}^{[l]} = \begin{pmatrix} \vdots & \vdots & & \vdots \\ \mathbf{z}^{[l](1)} & \mathbf{z}^{[l](2)} & \dots & \mathbf{z}^{[l](m)} \\ \vdots & \vdots & & \vdots \end{pmatrix}$$

We can now very compactly write the equations for layer $l = 1, \dots, L$:

matrix-multiplication

$$\begin{aligned} \mathbf{Z}^{[l]} &= \mathbf{W}^{[l]} \cdot \mathbf{A}^{[l-1]} + \mathbf{b}^{[l]} \\ \mathbf{A}^{[l]} &= g^{[l]}(\mathbf{Z}^{[l]}) \end{aligned}$$

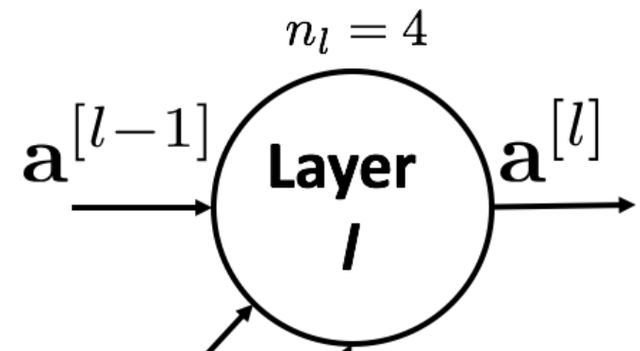
activation function applied element-wise

BROADCASTING (see numpy tutorial)

$$\begin{aligned} \mathbf{A}^{[l-1]} &: n_{l-1} \times m \\ \mathbf{W}^{[l]} &: n_l \times n_{l-1} \\ \mathbf{A}^{[l]} &: n_l \times m \end{aligned}$$

Layer in an MLP: Example

$$\mathbf{A}^{[l-1]} = \begin{pmatrix} 1 & -1 \\ -1 & -0.5 \\ -0.5 & 1 \end{pmatrix}$$



$$\mathbf{A}^{[l]} = \begin{pmatrix} 0.78 & 0.12 \\ 0.50 & 0.04 \\ 0.56 & 0.83 \\ 0.18 & 0.92 \end{pmatrix}$$

$$\mathbf{W}^{[l]} = \begin{pmatrix} 1 & -1 & -0.5 \\ 1 & 0.5 & -1 \\ -0.2 & 0.3 & 0.5 \\ -1 & 1 & 1 \end{pmatrix} \qquad \mathbf{b}^{[l]} = \begin{pmatrix} -1 \\ -1 \\ 1 \\ 1 \end{pmatrix}$$

$g^{[l]}(\cdot) = \sigma(\cdot)$:
sigmoid activation function

$$\begin{aligned} \mathbf{Z}^{[l]} &= \mathbf{W}^{[l]} \cdot \mathbf{A}^{[l-1]} + \mathbf{b}^{[l]} \\ \mathbf{A}^{[l]} &= g^{[l]}(\mathbf{Z}^{[l]}) \end{aligned}$$

$$\mathbf{Z}^{[l]} = \mathbf{W}^{[l]} \cdot \mathbf{A}^{[l-1]} + \mathbf{b}^{[l]} = \begin{pmatrix} 1.25 & -2.0 \\ 0.0 & -3.25 \\ 0.25 & 1.55 \\ -1.5 & 2.5 \end{pmatrix}$$

$$\mathbf{A}^{[l]} = \sigma(\mathbf{Z}^{[l]}) = \begin{pmatrix} 0.78 & 0.12 \\ 0.50 & 0.04 \\ 0.56 & 0.83 \\ 0.18 & 0.92 \end{pmatrix}$$

code snippet in `numpy`

```
import numpy as np
W = np.array([[1, -1, -0.5],[1, 0.5, -1], [-0.2, 0.3, 0.5],[-1, 1, 1]]).reshape(4,3)
b = np.array([-1, -1, 1, 1]).reshape(4,1)
aprev = np.array([[1, -1, -0.5],[-1, -0.5, 1]]).T.reshape(3,2)
z = np.matmul(W,aprev)+b
a = 1./(1.0+np.exp(-z))
```

Implementation in `numpy`

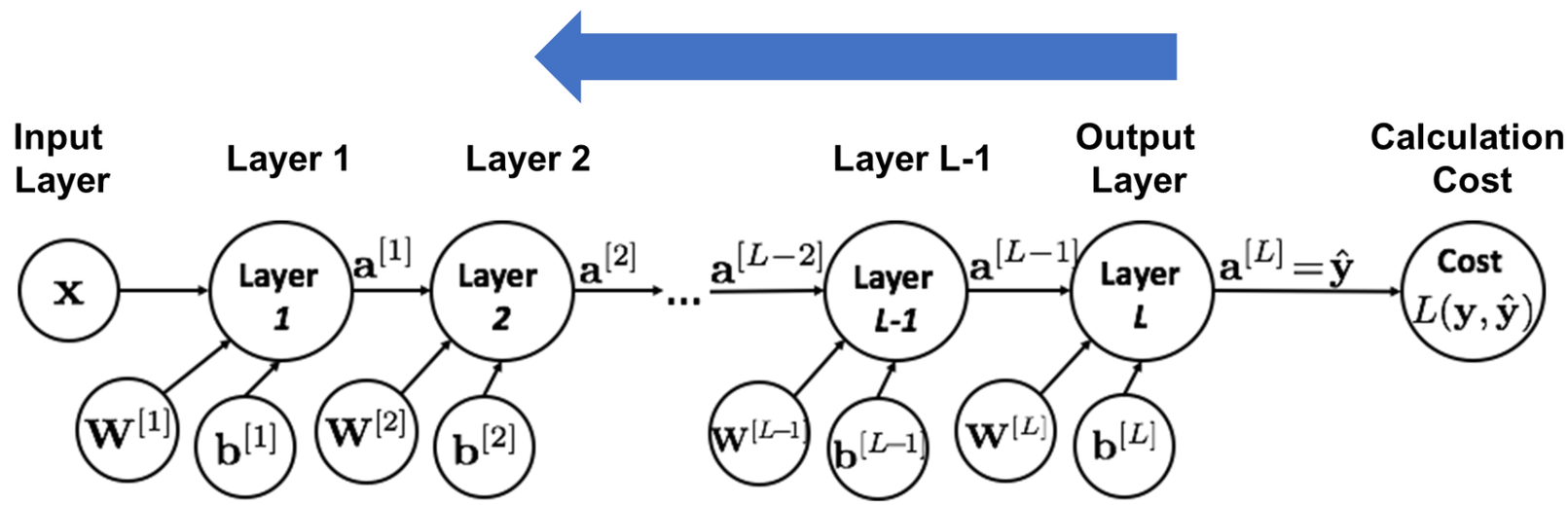
Some helpful commands:

Create Array with zeros, with shape (m,n)	`np.zeros(shape=(m,n), dtype='float')`
Create Array with standard normal, shape (m,n)	`np.random.randn(shape=(m,n))`
Reshape array A (e.g. (mn,1)) to (m,n)	`A.reshape(m,n)`
Transpose matrix A	`A.T`
Matrix-multiplication of A (m,n) and B (n,k)	`np.dot(A,B) or A@B`
Element-wise multiply of A, B	`np.multiply(A,B) or A*B`
Sum elements of A along axis 1	`np.sum(A, axis=1, keepdims=True)`

If set to True, the axes which are reduced are left in the result as dimensions with size one –> result will broadcast correctly against the input array.

Use `numpy` arrays and avoid explicit looping over its elements as far as you can!

How to Propagate Back Results?



$$\mathbf{z}^{[1]} = \mathbf{W}^{[1]} \cdot \mathbf{a}^{[0]} + \mathbf{b}^{[1]}$$
$$\mathbf{a}^{[1]} = g^{[1]}(\mathbf{z}^{[1]})$$

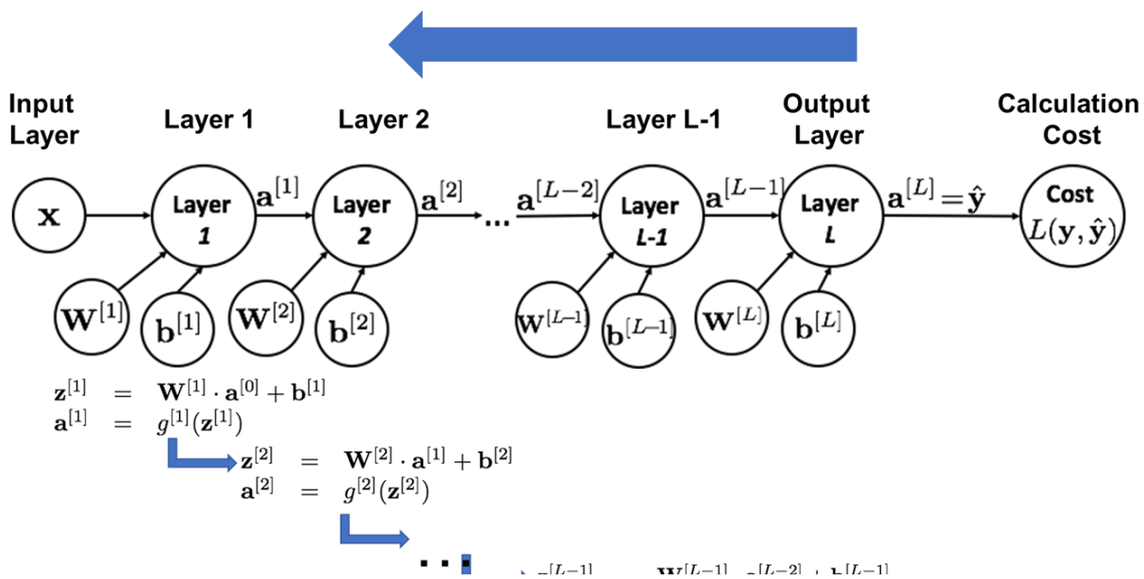
$$\mathbf{z}^{[2]} = \mathbf{W}^{[2]} \cdot \mathbf{a}^{[1]} + \mathbf{b}^{[2]}$$
$$\mathbf{a}^{[2]} = g^{[2]}(\mathbf{z}^{[2]})$$

$$\mathbf{z}^{[L-1]} = \mathbf{W}^{[L-1]} \cdot \mathbf{a}^{[L-2]} + \mathbf{b}^{[L-1]}$$
$$\mathbf{a}^{[L-1]} = g^{[L-1]}(\mathbf{z}^{[L-1]})$$

$$\mathbf{z}^{[L]} = \mathbf{W}^{[L]} \cdot \mathbf{a}^{[L-1]} + \mathbf{b}^{[L]}$$
$$\mathbf{a}^{[L]} = g^{[L]}(\mathbf{z}^{[L]})$$

$$L(\hat{\mathbf{y}}, \mathbf{y})$$

How to Propagate Back Results?



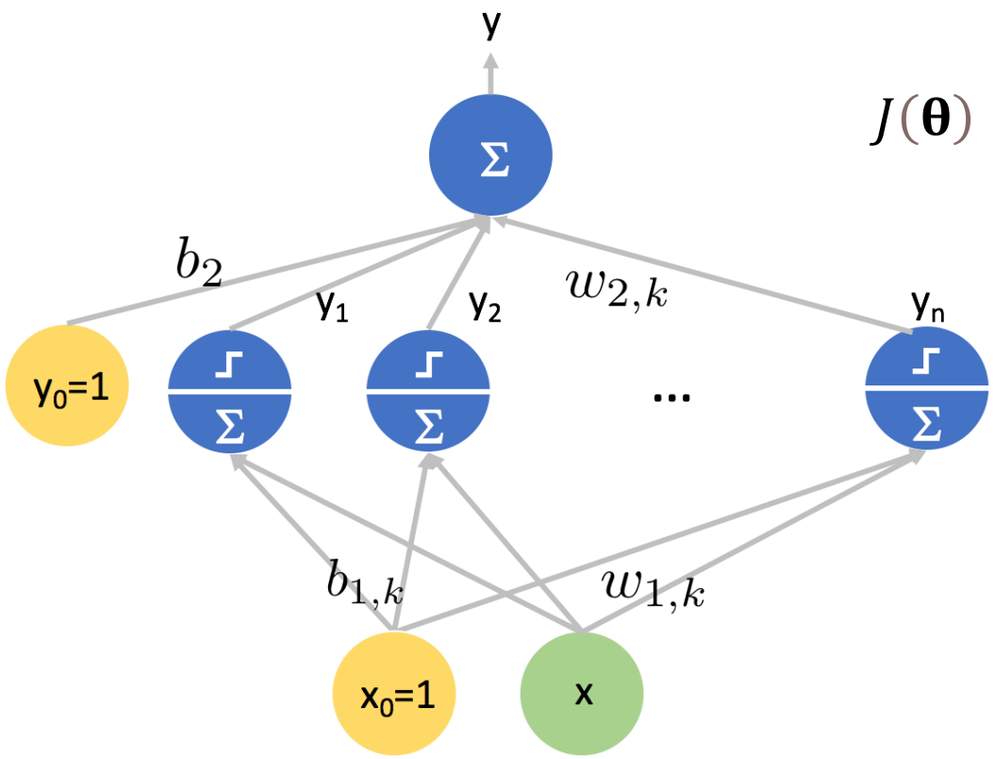
$$J(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m} L\left(\hat{\mathbf{y}}^{(i)}, \mathbf{y}^{(i)}\right) = \frac{1}{m}\sum_{i=1}^{m} L\left(h_{\theta,y^{(i)}}\left(\mathbf{x}^{(i)}\right), \mathbf{y}^{(i)}\right)$$

$$\boldsymbol{\theta} = \left(\mathbf{W}^{[1]}, \mathbf{b}^{[1]}, \ldots, \mathbf{W}^{[L]}, \mathbf{b}^{[L]}\right)$$

$$\mathbf{W}^{[l]} \leftarrow \mathbf{W}^{[l]} - \alpha \cdot \frac{\partial J(\boldsymbol{\theta})}{\partial \mathbf{W}^{[l]}}$$

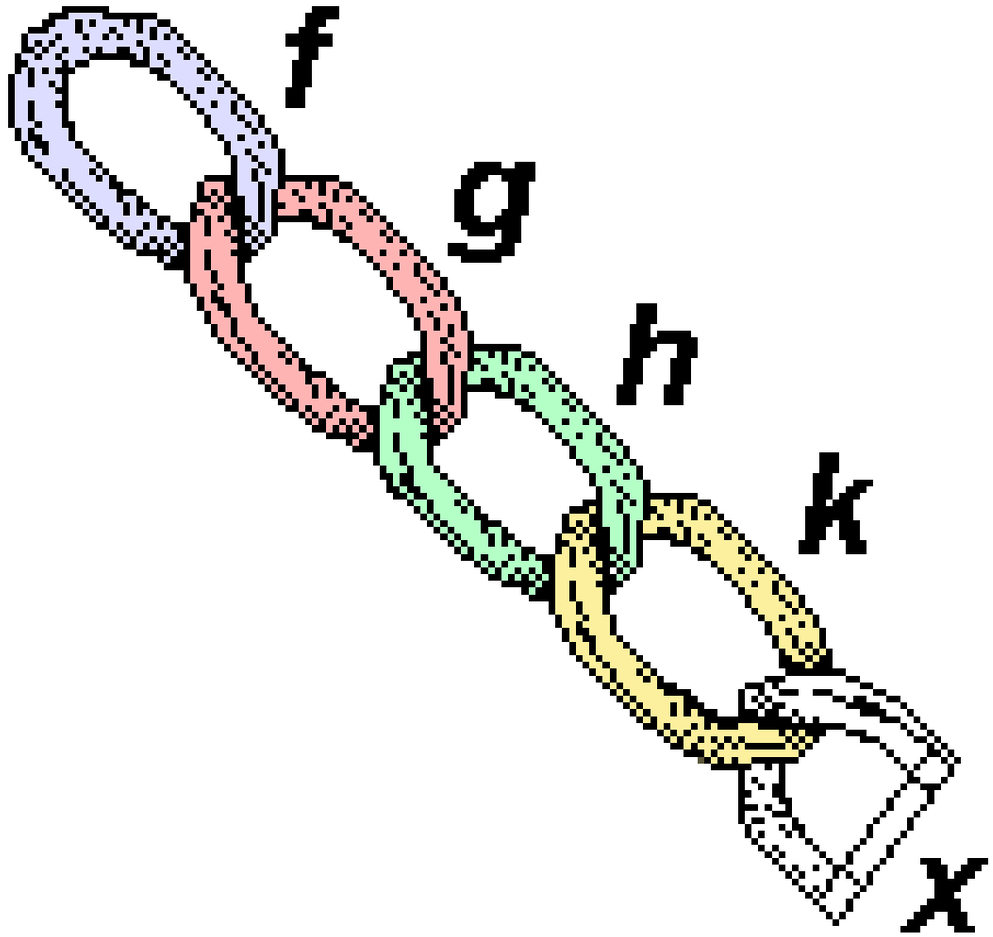
$$\mathbf{b}^{[l]} \leftarrow \mathbf{b}^{[l]} - \alpha \cdot \frac{\partial J(\boldsymbol{\theta})}{\partial \mathbf{b}^{[l]}}$$

Practical example: PW04



$$J(\boldsymbol{\theta}) = \frac{1}{2m}\sum_{i=1}^{m}\left(\mathrm{y}^{(i)} - \left(\sum_{k=1}^{n} w_{2,k}\cdot\sigma\left(w_{1,k}\cdot\mathrm{x}^{(i)} + b_{1,k}\right) + b_2\right)\right)^2$$

Chain Rule of Calculus



Example: Chain Rule

1d Example

$$f(x) = \sqrt{1+x^2}, \quad g(z) = \sqrt{z}, \quad h(x) = 1 + x^2$$

$$f(x) = g(h(x))$$

$$\frac{df(x)}{dx} = \frac{d}{dx} g\big(h(x)\big) = \left.\frac{dg(z)}{dz}\right|_{z=h(x)} \cdot \frac{dh(x)}{dx}$$

Variation of f w.r.t. x

Variation of g w.r.t. z

Variation of h w.r.t. x

Example: Chain Rule

1d Example

$$f(x) = \sqrt{1+x^2}, \quad g(z) = \sqrt{z}, \quad h(x) = 1 + x^2$$

$$f(x) = g(h(x))$$

$$\frac{df(x)}{dx} = \frac{d}{dx} g\big(h(x)\big) = \left.\frac{dg(z)}{dz}\right|_{z=h(x)} \cdot \frac{dh(x)}{dx}$$

Variation of f w.r.t. x

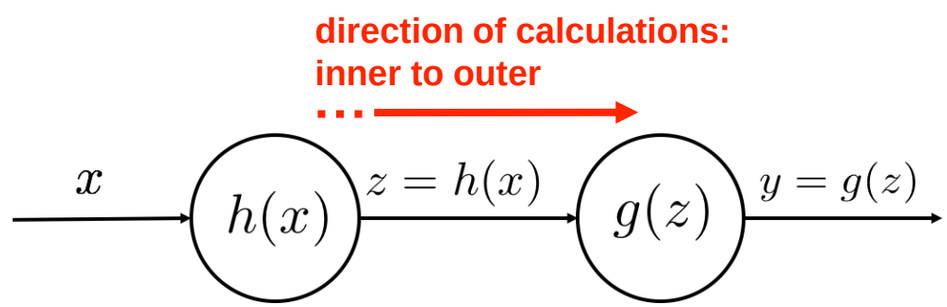
Variation of g w.r.t. z

Variation of h w.r.t. x

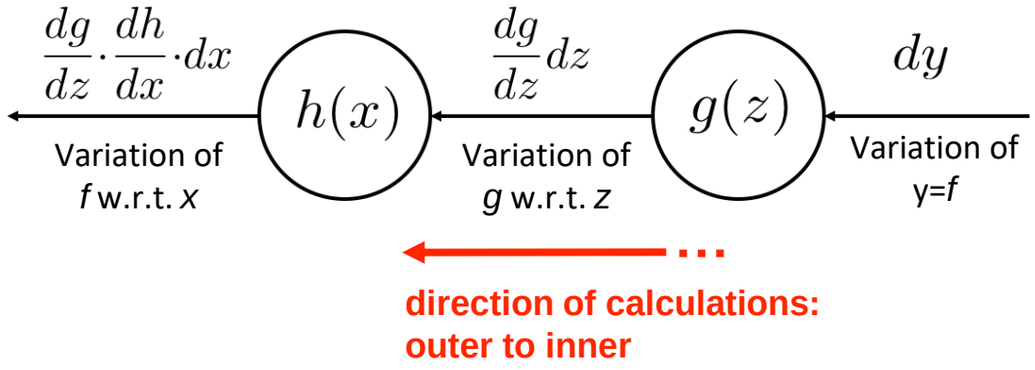
$$\frac{dg(z)}{dz} = \frac{d}{dz}\sqrt{z} = \frac{1}{2\sqrt{z}}$$

$$\frac{dh(x)}{dx} = \frac{d}{dx}(1+x^2) = 2x$$

Computational Graph View

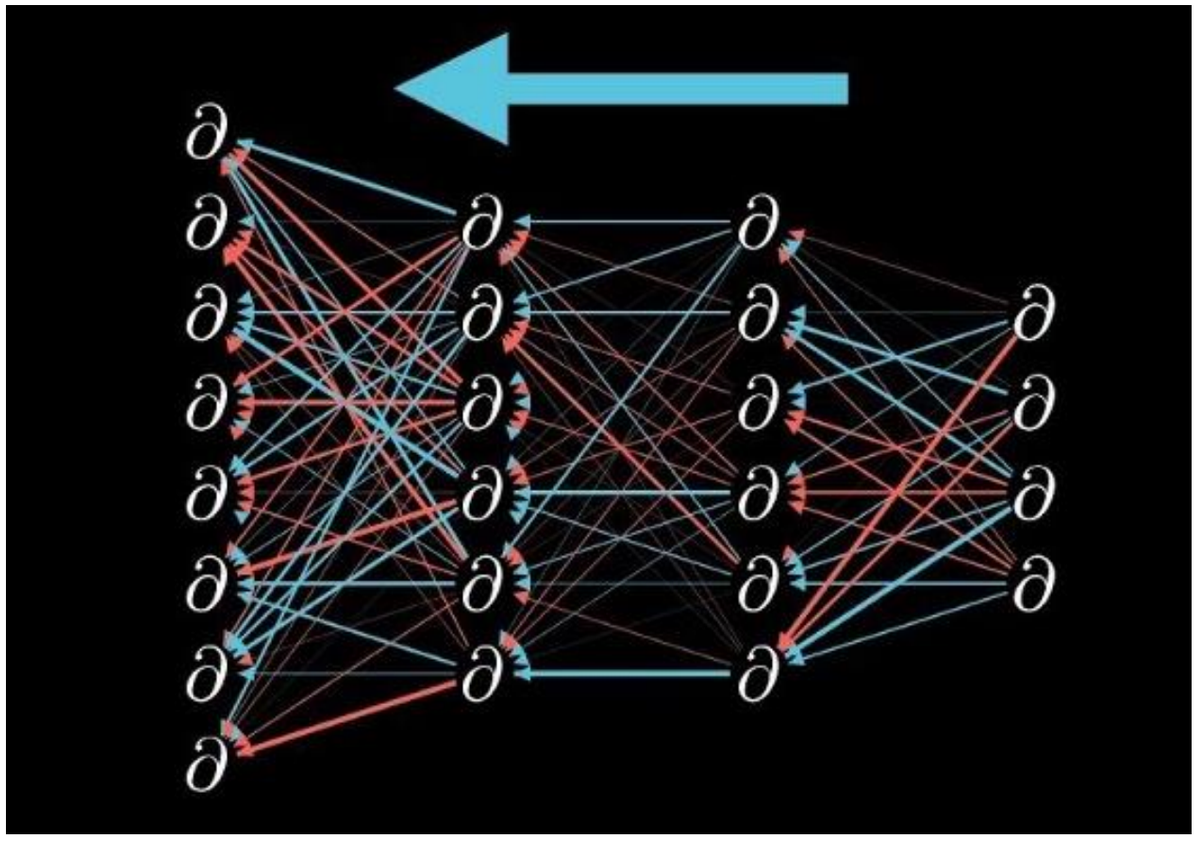


Forward Propagation



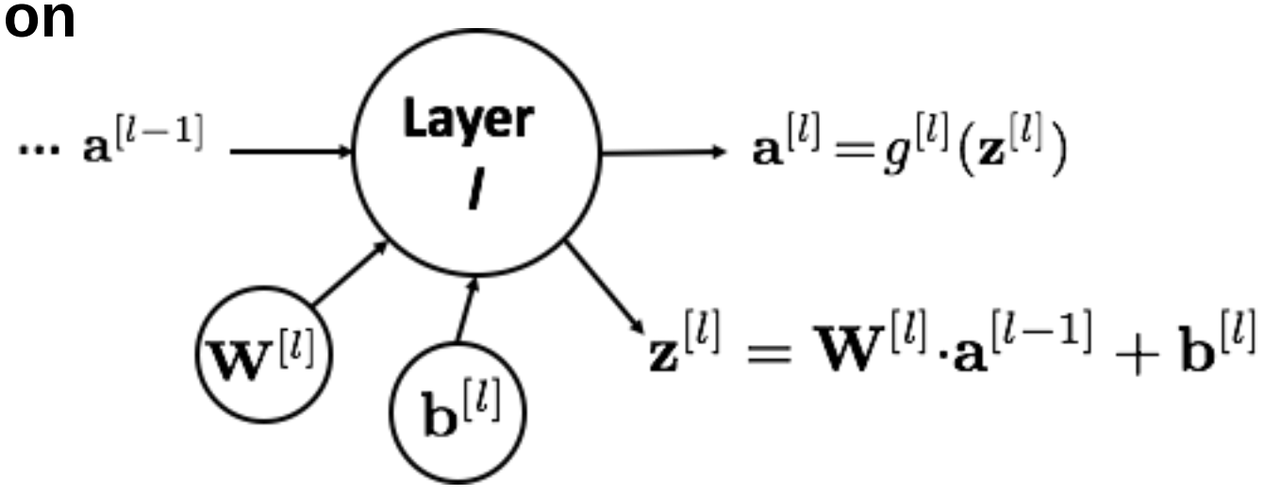
Backward Propagation

Back-Propagation

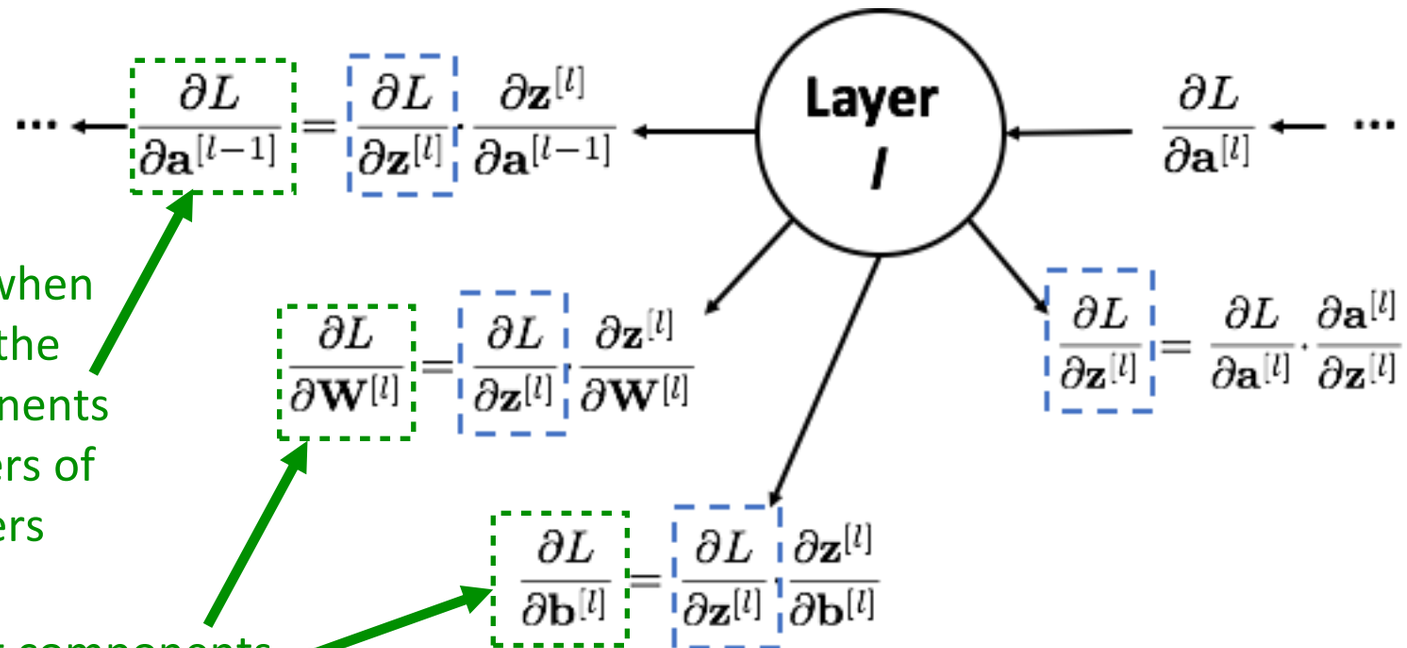


Backprop through a Single Layer

Forward Propagation
Layer represented
by a single node.



Backward Propagation
Proper handling of
tensor indices
largely ignored (see
next slide)



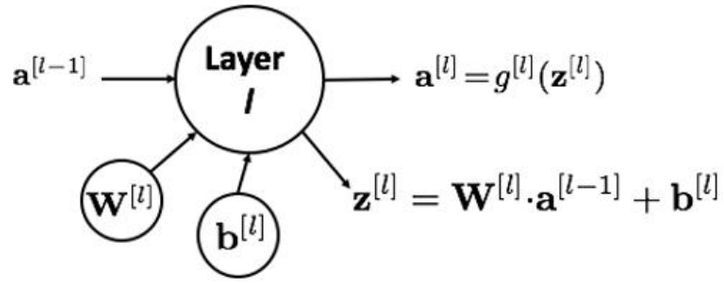
this is needed when interested in the gradient components w.r.t. parameters of previous layers

gradient components w.r.t. parameters of layer *l*

Backprop through a Single Layer

(indices explicit)

$$a_j^{[l]} = g^{[l]}\left(z_j^{[l]}\right) = g^{[l]}\left(\sum_i W_{ji}^{[l]} \cdot a_i^{[l-1]} + b_j^{[l]}\right)$$



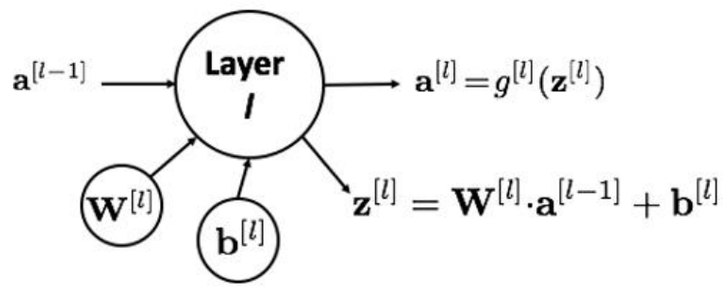
$$\frac{\partial L}{\partial z_j^{[l]}} =$$

$$\frac{\partial a_k^{[l]}}{\partial z_j^{[l]}} =$$

Backprop through a Single Layer

(indices explicit)

$$a_j^{[l]} = g^{[l]}\left(z_j^{[l]}\right) = g^{[l]}\left(\sum_i W_{ji}^{[l]} \cdot a_i^{[l-1]} + b_j^{[l]}\right)$$



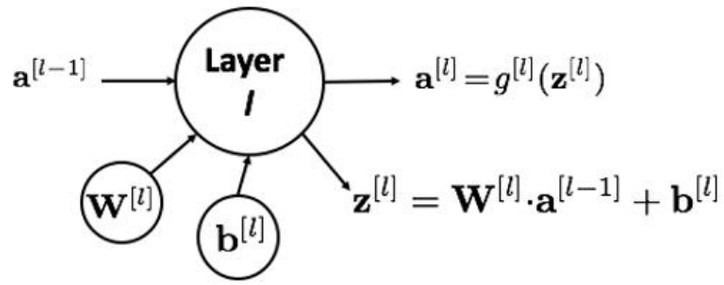
$$\frac{\partial L}{\partial z_j^{[l]}} = \sum_k \frac{\partial L}{\partial a_k^{[l]}} \cdot \frac{\partial a_k^{[l]}}{\partial z_j^{[l]}} =^{1)} \sum_k \frac{\partial L}{\partial a_k^{[l]}} \cdot \delta_{kj} \cdot \frac{dg^{[l]}\left(z_j^{[l]}\right)}{dz} = \frac{\partial L}{\partial a_j^{[l]}} \cdot \frac{dg^{[l]}\left(z_j^{[l]}\right)}{dz}$$

$$\boxed{\frac{\partial L}{\partial z_j^{[l]}} = \frac{\partial L}{\partial a_j^{[l]}} \cdot \frac{dg^{[l]}\left(z_j^{[l]}\right)}{dz}}$$

Backprop through a Single Layer

(indices explicit)

$$a_j^{[l]} = g^{[l]}\left(z_j^{[l]}\right) = g^{[l]}\left(\sum_i W_{ji}^{[l]} \cdot a_i^{[l-1]} + b_j^{[l]}\right)$$



$$\frac{\partial L}{\partial W_{ki}^{[l]}} =$$

$$\frac{\partial z_j^{[l]}}{\partial W_{ki}^{[l]}} =$$

Backprop through a Single Layer

(indices explicit)

$$\frac{\partial L}{\partial W_{ki}^{[l]}} = \sum_j \frac{\partial L}{\partial z_j^{[l]}} \cdot \frac{\partial z_j^{[l]}}{\partial W_{ki}^{[l]}} =^{1)} \sum_j \frac{\partial L}{\partial z_j^{[l]}} \cdot \delta_{kj} \cdot a_i^{[l-1]} = \frac{\partial L}{\partial z_k^{[l]}} \cdot a_i^{[l-1]}$$

$$\boxed{\frac{\partial L}{\partial W_{ki}^{[l]}} = \frac{\partial L}{\partial z_k^{[l]}} \cdot a_i^{[l-1]}}$$

$$\frac{\partial L}{\partial b_k^{[l]}} = \sum_j \frac{\partial L}{\partial z_j^{[l]}} \cdot \frac{\partial z_j^{[l]}}{\partial b_k^{[l]}} =^{1)} \sum_j \frac{\partial L}{\partial z_j^{[l]}} \cdot \delta_{kj} \cdot 1 = \frac{\partial L}{\partial z_k^{[l]}}$$

$$\boxed{\frac{\partial L}{\partial b_k^{[l]}} = \frac{\partial L}{\partial z_k^{[l]}}}$$

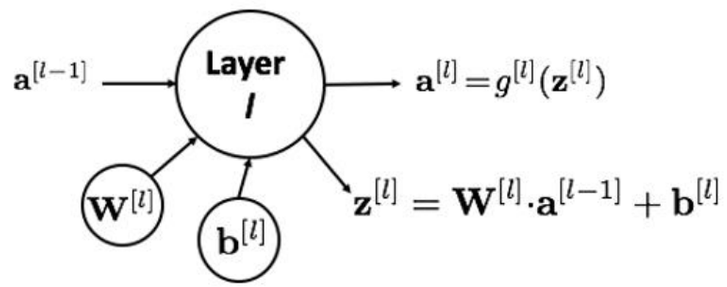
Backprop through a Single Layer

(indices explicit)

$$\frac{\partial L}{\partial a_k^{[l-1]}} = \sum_j \frac{\partial L}{\partial z_j^{[l]}} \cdot \frac{\partial z_j^{[l]}}{\partial a_k^{[l-1]}} =^{1)} \sum_j \frac{\partial L}{\partial z_j^{[l]}} \cdot W_{jk}$$

$$\frac{\partial L}{\partial a_k^{[l-1]}} = \sum_j \frac{\partial L}{\partial z_j^{[l]}} \cdot W_{jk}$$

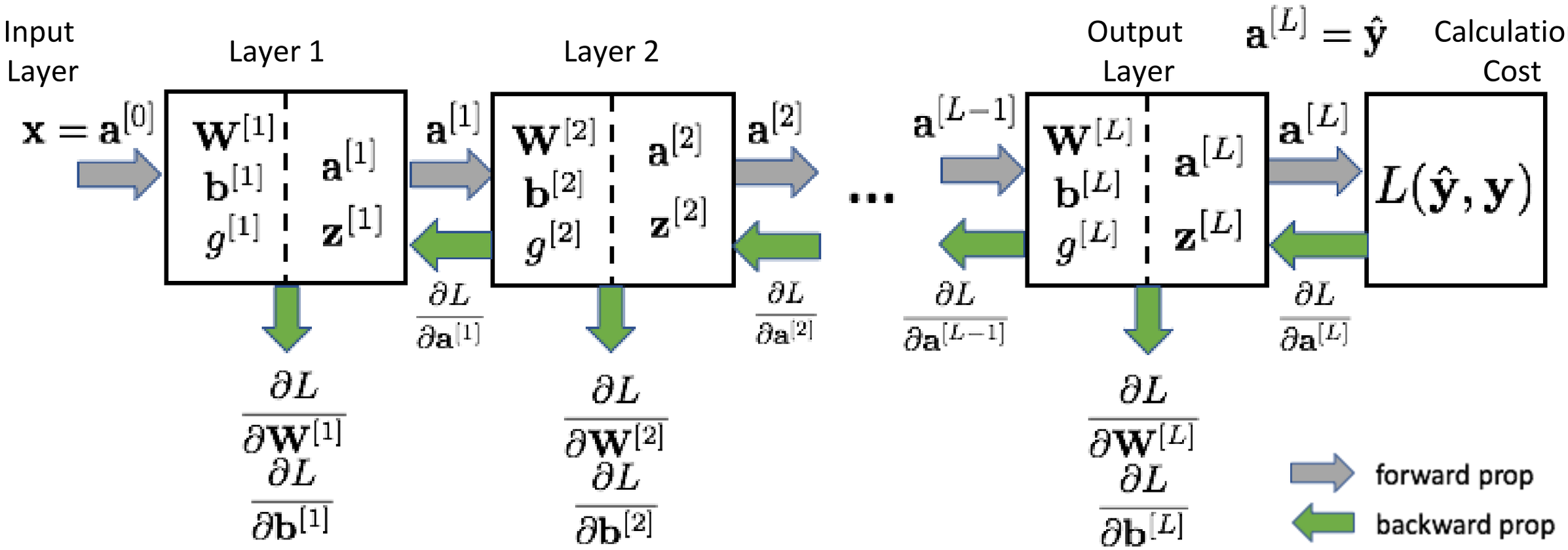
$$\frac{\partial L}{\partial z_j^{[l]}} = \frac{\partial L}{\partial a_j^{[l]}} \cdot \frac{dg^{[l]}\left(z_j^{[l]}\right)}{dz}$$



$$\frac{\partial L}{\partial W_{ki}^{[l]}} = \frac{\partial L}{\partial z_k^{[l]}} \cdot a_i^{[l-1]}$$

$$\frac{\partial L}{\partial b_k^{[l]}} = \frac{\partial L}{\partial z_k^{[l]}}$$

Overview Back-Propagation Scheme



Backprop through a Single Layer

(matrix notation)

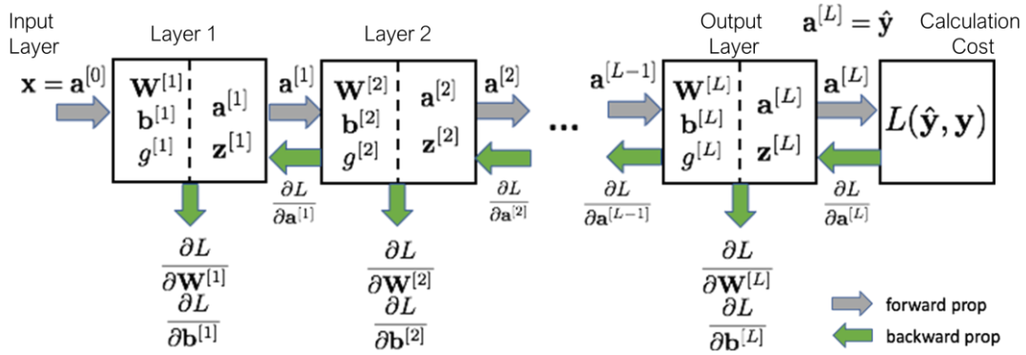
normal
MLP layer
(not Softmax)

$$\frac{\partial L}{\partial \mathbf{z}^{[l]}} = \frac{\partial L}{\partial \mathbf{a}^{[l]}} * \frac{dg^{[l]}\left(\mathbf{z}^{[l]}\right)}{dz}$$

$$\frac{\partial L}{\partial \mathbf{W}^{[l]}} = \frac{\partial L}{\partial \mathbf{z}^{[l]}} \cdot \left(\mathbf{a}^{[l-1]}\right)^T$$

$$\frac{\partial L}{\partial \mathbf{b}^{[l]}} = \frac{\partial L}{\partial \mathbf{z}^{[l]}}$$

$$\frac{\partial L}{\partial \mathbf{a}^{[l-1]}} = \left(\mathbf{W}^{[l]}\right)^T \cdot \frac{\partial L}{\partial \mathbf{z}^{[l]}}$$



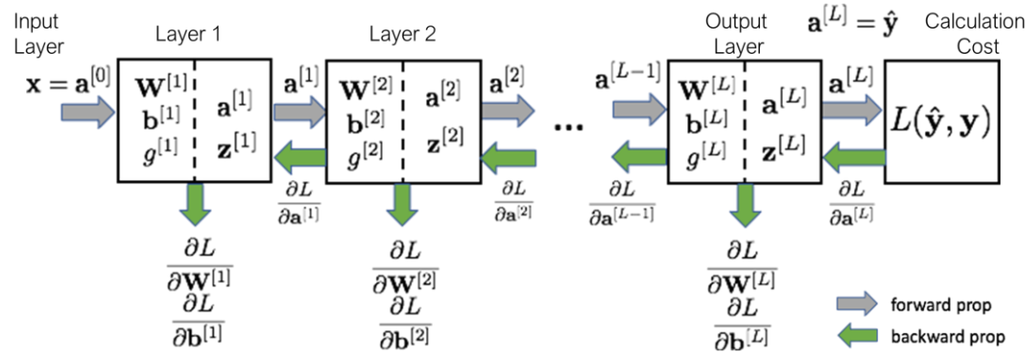
- $*$ is an *elementwise* multiplication of $\frac{\partial L}{\partial \mathbf{a}^{[l]}}$ and $\frac{\partial g^{[l]}(\mathbf{z}^{[l]})}{\partial z}$ both vectors of size n^l x 1

- Product between $\frac{\partial L}{\partial \mathbf{z}^{[l]}}$ and the transpose of $\mathbf{a}^{[l-1]}$ $((..)^T)$ is matrix product of $\frac{\partial L}{\partial \mathbf{z}^{[l]}}$ of size n^l x 1 and the transpose of $\mathbf{a}^{[l-1]}$ of size 1 x n^{l-1}

- The product of the transpose of $\mathbf{W}^{[l]}$ (dimension n^{l-1} x n^l) and $\frac{\partial L}{\partial \mathbf{z}^{[l]}}$ (dimension n^l x 1) is a matrix product.

Backprop through a Single Layer

(matrix notation)



Softmax with CE cost

$$\frac{\partial L}{\partial \mathbf{z}^{[l]}} = \hat{\mathbf{y}} - \mathbf{y}$$

$$\frac{\partial L}{\partial \mathbf{W}^{[l]}} = \frac{\partial L}{\partial \mathbf{z}^{[l]}} \cdot \left(\mathbf{a}^{[l-1]}\right)^T$$

$$\frac{\partial L}{\partial \mathbf{b}^{[l]}} = \frac{\partial L}{\partial \mathbf{z}^{[l]}}$$

$$\frac{\partial L}{\partial \mathbf{a}^{[l-1]}} = \left(\mathbf{W}^{[l]}\right)^T \cdot \frac{\partial L}{\partial \mathbf{z}^{[l]}}$$

- prediction $\hat{\mathbf{y}}$ and the true label $\mathbf{y}$ are of size n^L x 1. True label $\mathbf{y}$ is a one hot vector.
- Product between $\frac{\partial L}{\partial \mathbf{z}^{[l]}}$ and the transpose of $\mathbf{a}^{[l-1]}$ $((..)^T)$ is matrix product of $\frac{\partial L}{\partial \mathbf{z}^{[l]}}$ of size n^l x 1 and the transpose of $\mathbf{a}^{[l-1]}$ of size 1 x n^{l-1}
- The product of the transpose of $\mathbf{W}^{[l]}$ (dimension n^{l-1} x n^l) and $\frac{\partial L}{\partial \mathbf{z}^{[l]}}$ (dimension n^l x 1) is a matrix product.