

Introduction to Neural Networks, Selected Topics in Machine Learning, and Their Applications in Solving PDEs

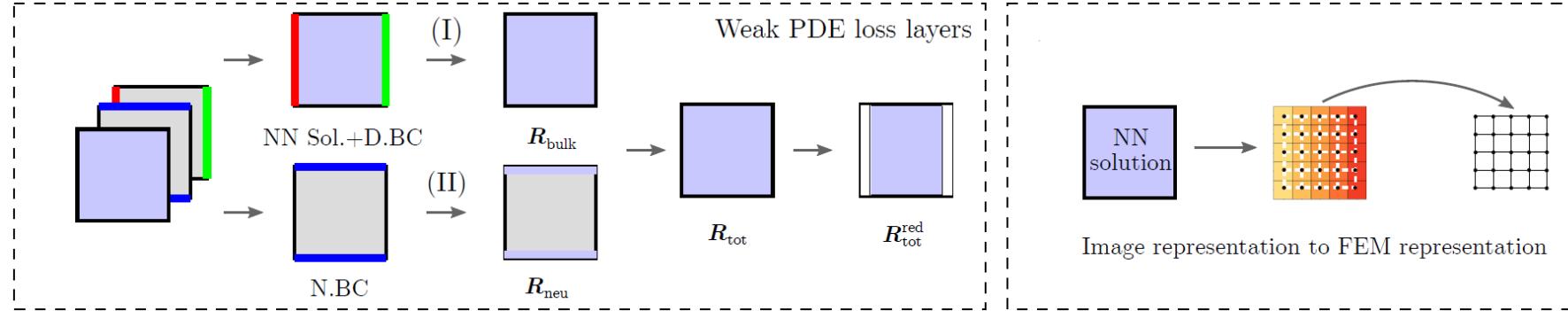
2023

Lectures Overview and timeline

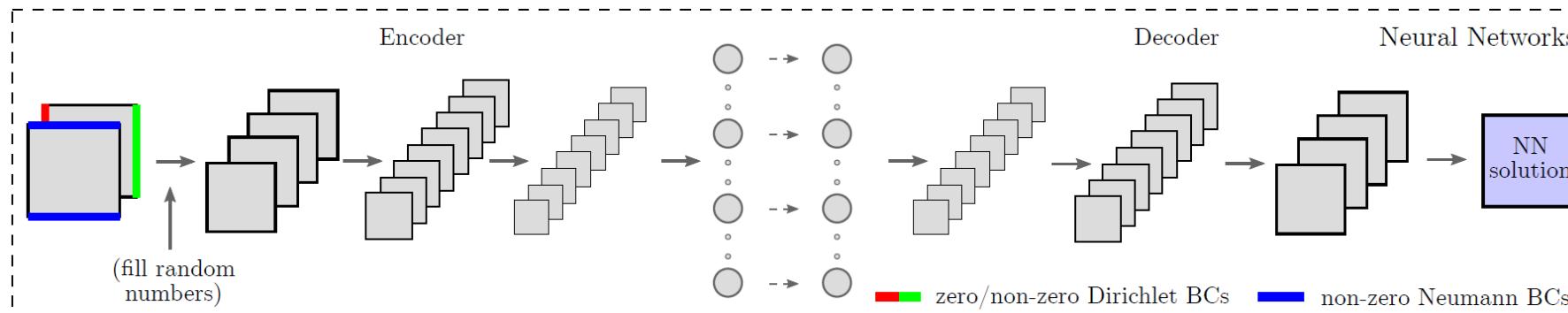
- [Lecture 1](#) – Essentials of Neural Networks and Machine Learning.
- [Lecture 2](#) – Google Colab Usage, Convolutional Neural Networks Introduction, ML-PDE Project Overview.
- [Lecture 3](#) – TensorFlow in Colab: Implementing NNs & CNNs for MNIST
- [Lecture 4](#) – Advanced Machine Learning & Deep Learning Topics; Project Discussion.

Final project overview

Part 1. Calculating the residual of a 2D linear Dirichlet boundary value problem (BVP) defined on a square image using quadrilateral finite elements



Part 2. Solving the problem by training a convolutional autoencoder neural network with the FEM residual (part 1) as a custom loss function.



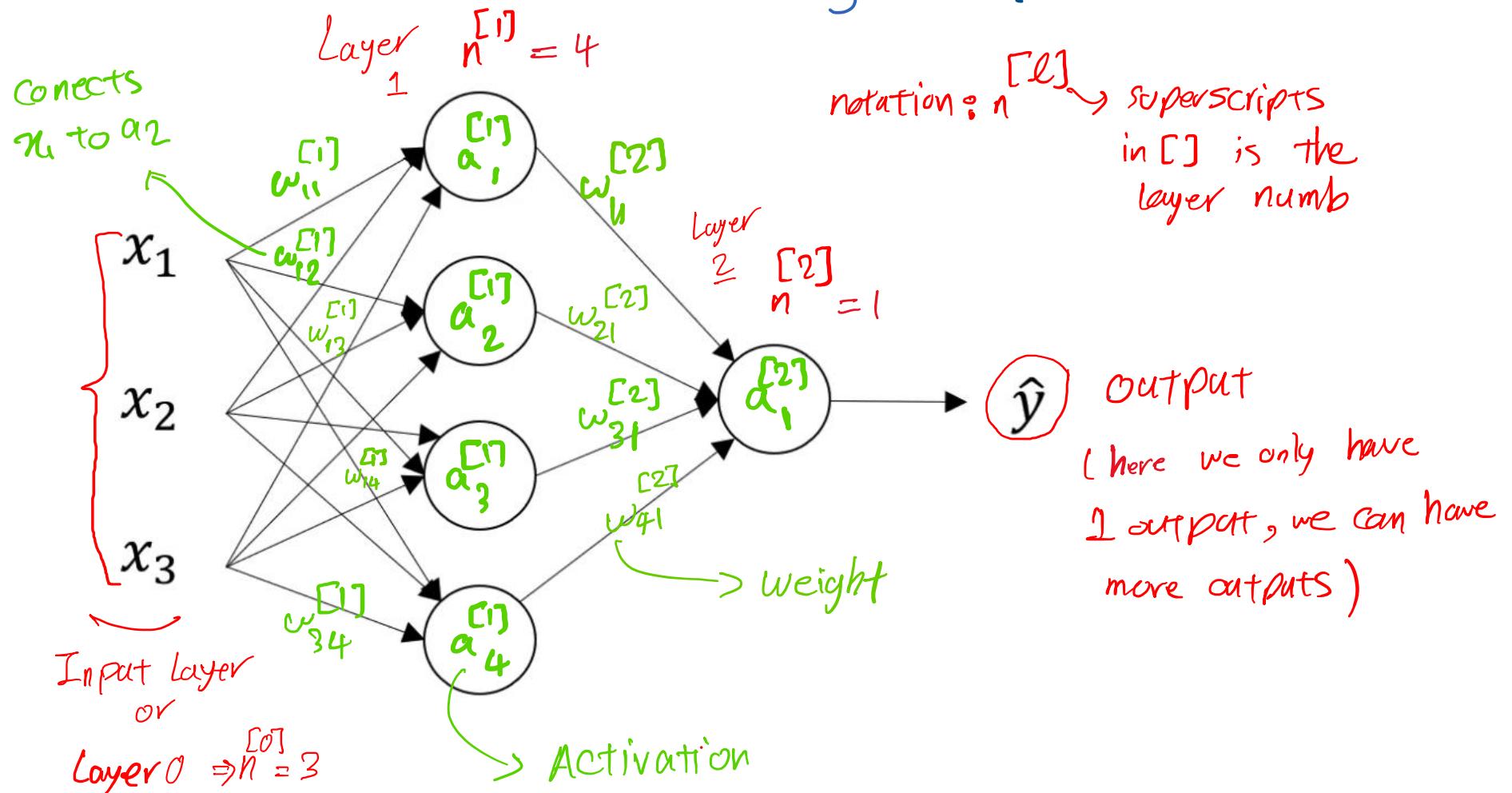
Tools: Python, NumPy, TensorFlow, Google Colab

Neural Network Representation

How to represent the architecture of NNs using labels?

2-Layer NN

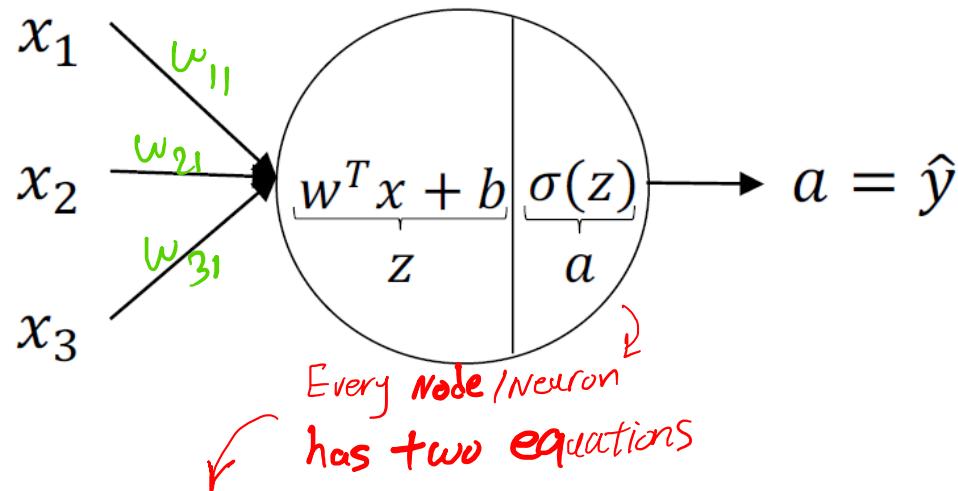
How we use these labels in equations?
(next slide)



Neural Network Representation

(Equations)

Simplest case: 1-Layer NN



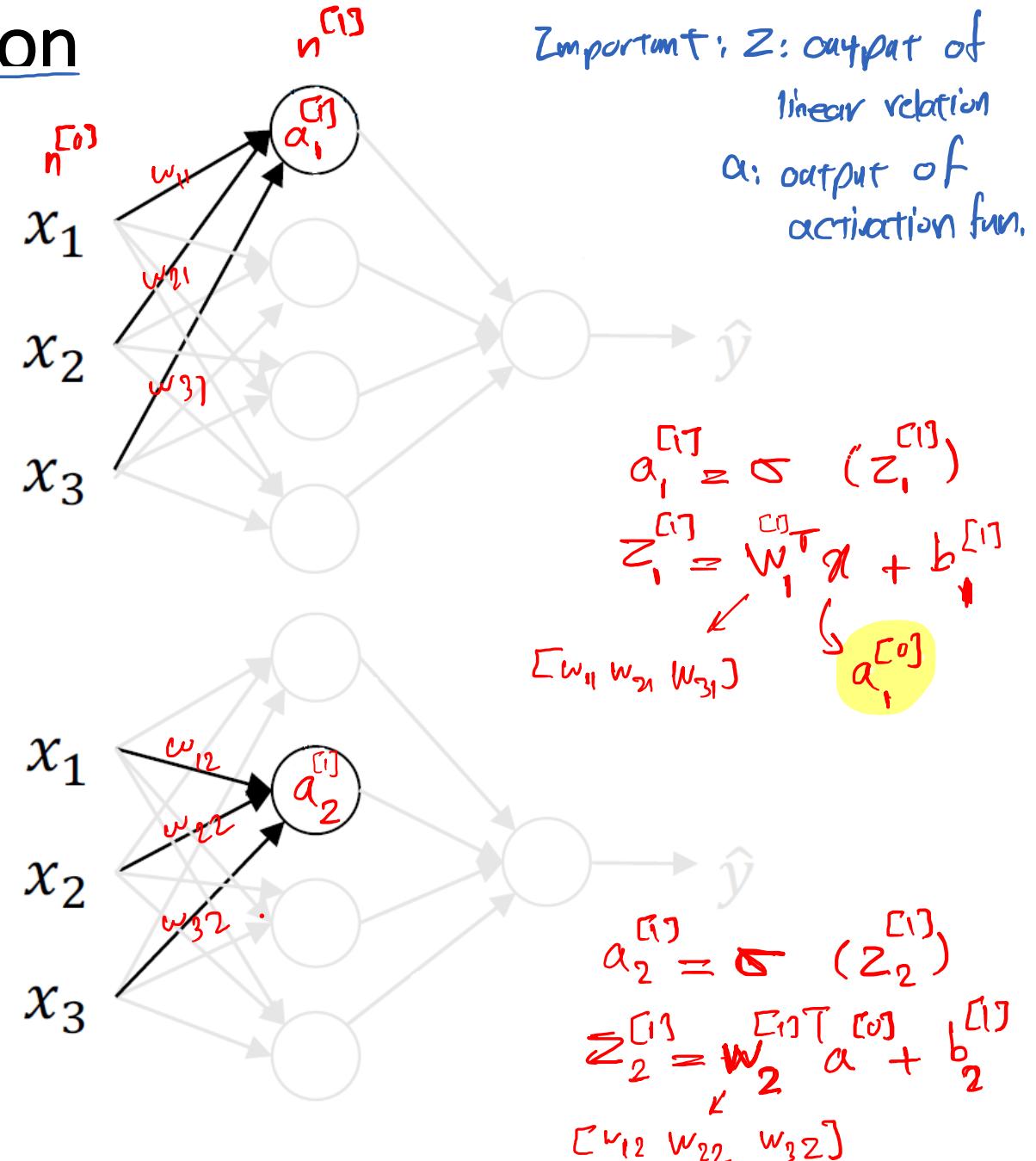
Linear $z = w^T x + b$
 weights w_{11}, w_{21}, w_{31} bias b

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

$$w^T = [w_{11} \ w_{21} \ w_{31}]^T$$

$$z_{1x1} = w^T x + b$$

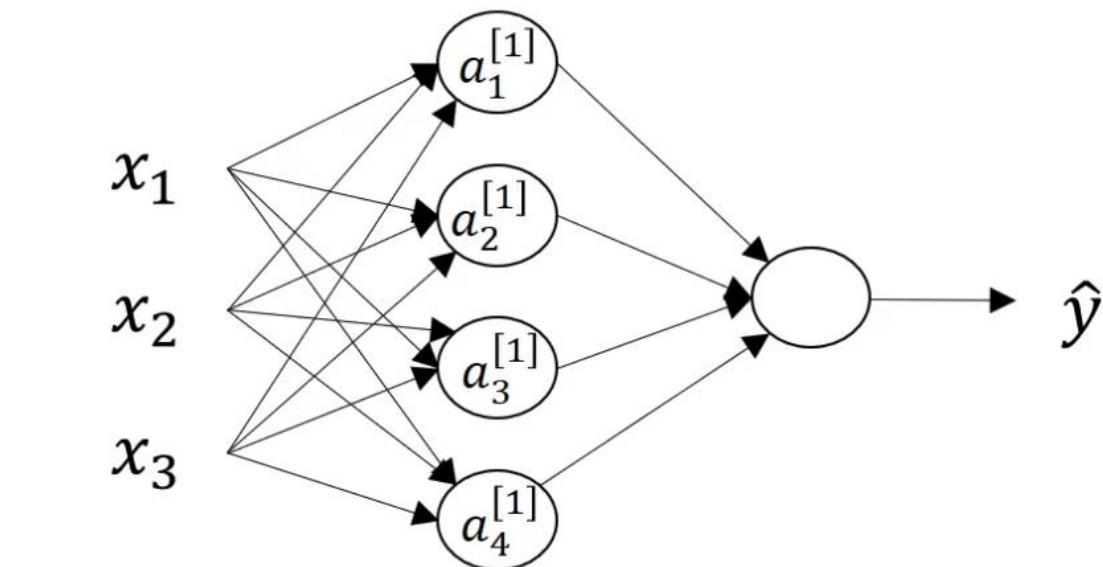
Nonlinear $a = \sigma(z)$ activation function (non linear)



Important: Z : output of linear relation
 a : output of activation fun.

Neural Network Representation: Vectorizing across W and b

→ efficiency
in python and
numpy, TF



$$W^{[1]} = \begin{bmatrix} w_1^{[1]T} \\ \vdots \\ w_4^{[1]T} \end{bmatrix} \quad 4 \times 3$$

$$b^{[1]} = \begin{bmatrix} b_1^{[1]} \\ \vdots \\ b_4^{[1]} \end{bmatrix} \quad 4 \times 1$$

$$z^{[1]} = \begin{bmatrix} z_1^{[1]} \\ \vdots \\ z_4^{[1]} \end{bmatrix} \quad 4 \times 1$$

all nodes in the first layer

$$\begin{aligned} z_1^{[1]} &= w_1^{[1]T} x + b_1^{[1]}, & a_1^{[1]} &= \sigma(z_1^{[1]}) \\ z_2^{[1]} &= w_2^{[1]T} x + b_2^{[1]}, & a_2^{[1]} &= \sigma(z_2^{[1]}) \\ z_3^{[1]} &= w_3^{[1]T} x + b_3^{[1]}, & a_3^{[1]} &= \sigma(z_3^{[1]}) \\ z_4^{[1]} &= w_4^{[1]T} x + b_4^{[1]}, & a_4^{[1]} &= \sigma(z_4^{[1]}) \end{aligned}$$

linear non linear

vectorizing

$4 = n^{[1]}$: $\boxed{z^{[1]} = W^{[1]}x + b^{[1]}}$

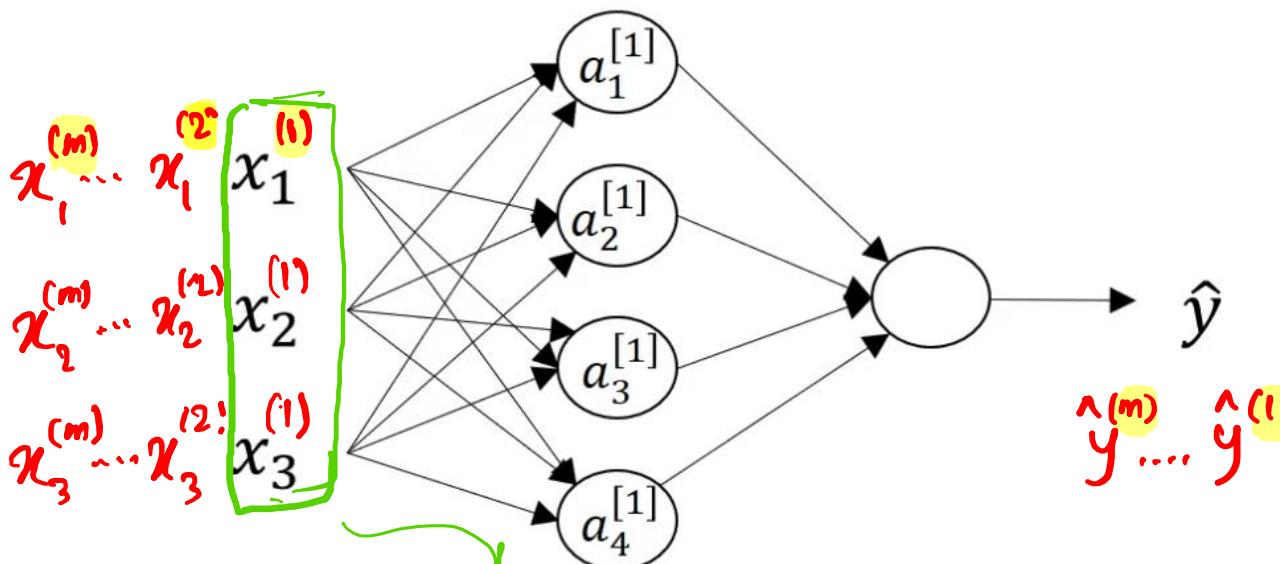
$4 = n^{[1]}$: $\boxed{a^{[1]} = \sigma(z^{[1]})}$

$1 = n^{[2]}$: $\boxed{z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}}$

$1 = n^{[2]}$: $\boxed{a^{[2]} = \sigma(z^{[2]})}$

No subscripts

Neural Network Representation: Vectorizing across examples x



$$X_{3 \times m} = \begin{bmatrix} x_1^{(1)} \\ \vdots \\ x_3^{(1)} \end{bmatrix} \dots \begin{bmatrix} x_1^{(m)} \\ \vdots \\ x_3^{(m)} \end{bmatrix} \quad (3 \times m)$$

$$Z_{4 \times m} = \begin{bmatrix} z_1^{(1)} \\ \vdots \\ z_4^{(1)} \end{bmatrix} \dots \begin{bmatrix} z_1^{(m)} \\ \vdots \\ z_4^{(m)} \end{bmatrix} \quad 4 \times m$$

\hookrightarrow evaluate for every entry

for $i = 1$ to m :

$$z^{[1](i)} = W^{[1]}x^{(i)} + b^{[1]}$$

$$a^{[1](i)} = \sigma(z^{[1](i)})$$

$$z^{[2](i)} = W^{[2]}a^{[1](i)} + b^{[2]}$$

$$a^{[2](i)} = \sigma(z^{[2](i)})$$

vectorizing

Uppercase
Capital

$\leftarrow X, Z, A$

$$Z^{[1]} = W^{[1]}X + b^{[1]}$$

$$A^{[1]} = \sigma(Z^{[1]})$$

$$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$$

$$A^{[2]} = \sigma(Z^{[2]})$$

broadcast

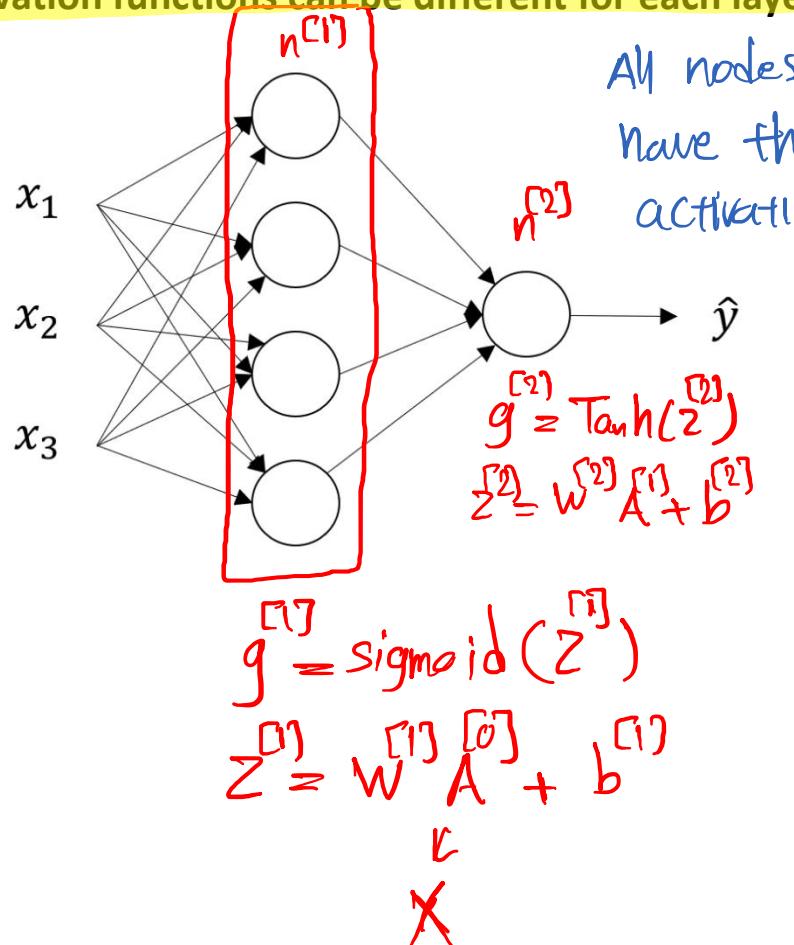
4×1 to $4 \times m$

by just repeating

$[:]_{4 \times 1} \rightarrow [:; :; \dots; :]_{4 \times m}$

Activation functions

Activation functions can be different for each layer



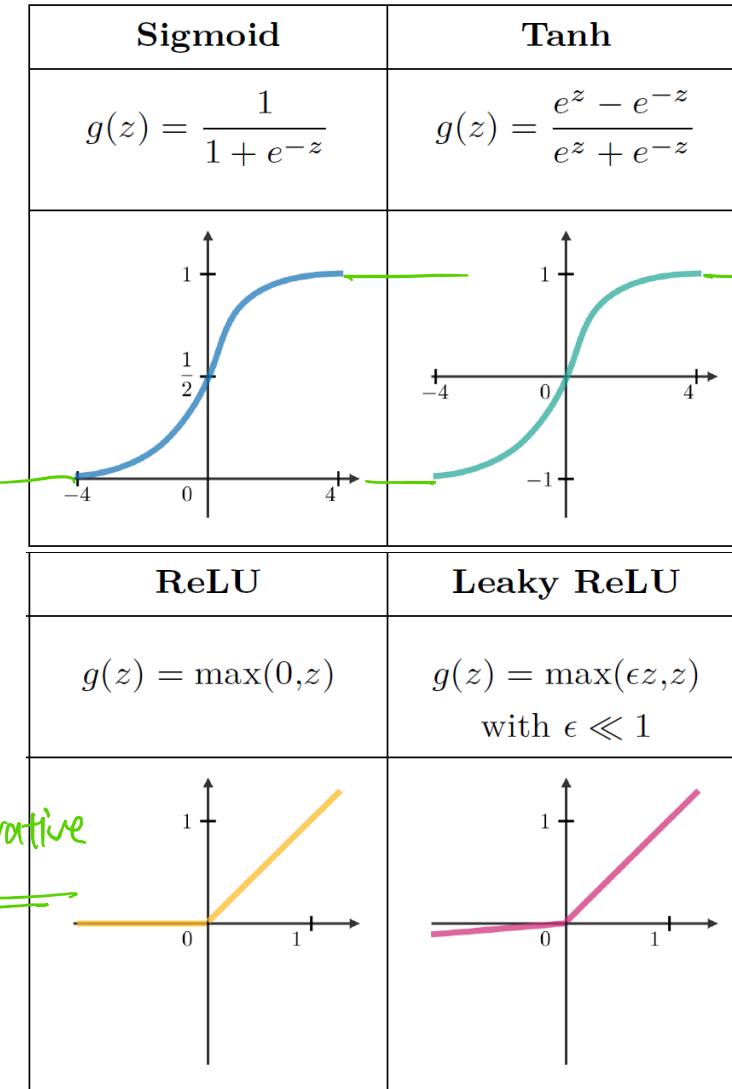
All nodes in a layer

have the same
activation function

but activation
function can be
different for
different layers

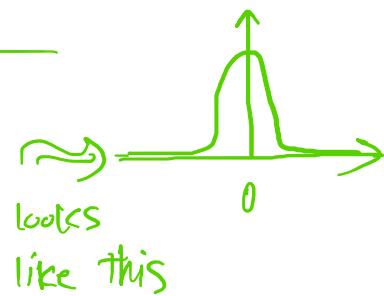
{
0 no derivative
1 $0 < x$

Activation functions introduce
non-linear complexities to the model.

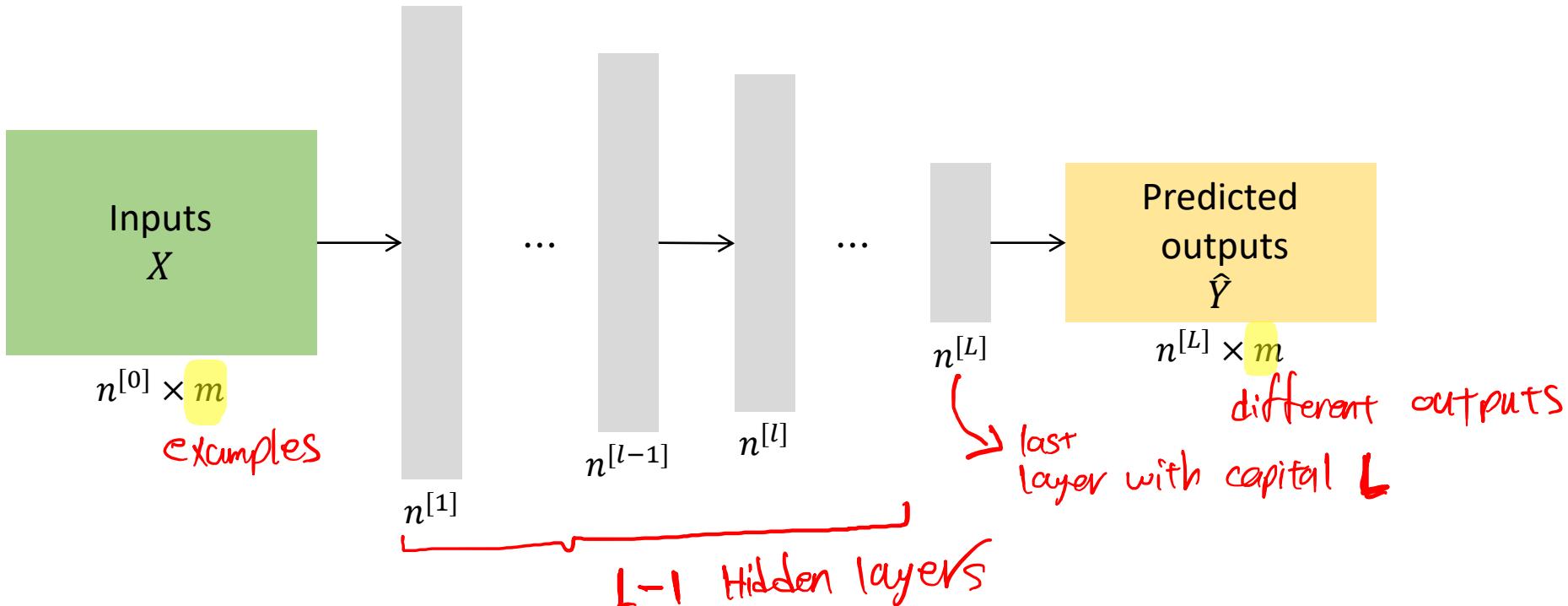


Take derivatives
in back prop.

↙
derivatives



Forward propagation



Forward prop using $X = A^{[0]}$

For $l = 1, \dots, L$:

$$Z^{[l]} = W^{[l]} A^{[l-1]} + b^{[l]}$$

$$A^{[l]} = g^{[l]}(Z^{[l]})$$

Sizes:

$$W^{[l]}: n^{[l]} \times n^{[l-1]}$$

$$b^{[l]}: n^{[l]} \times 1$$

$$A^{[l]}, Z^{[l]}: n^{[l]} \times m$$

$$A^{[l-1]}: n^{[l-1]} \times m$$

$$W^{[1]}: n^{[1]} \times n^{[0]}$$

$$b^{[1]}: n^{[1]} \times 1$$

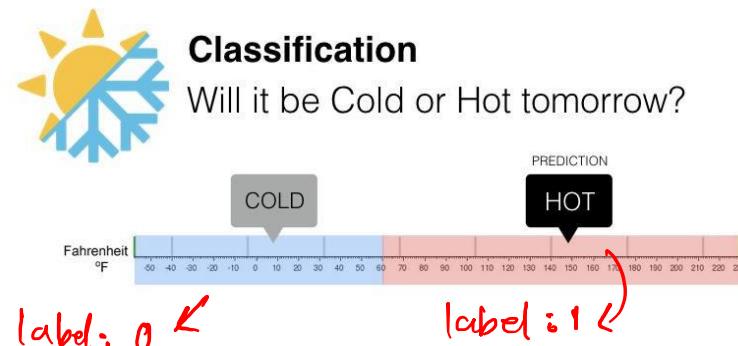
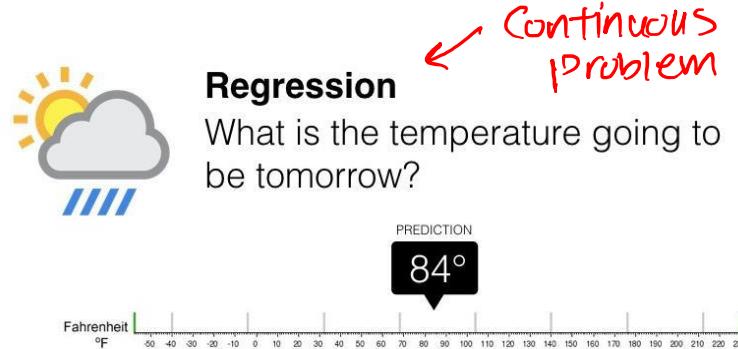
$$A^{[1]}, Z^{[1]}: n^{[1]} \times m$$

$$W^{[L]}: n^{[L]} \times n^{[L-1]}$$

$$b^{[L]}: n^{[L]} \times 1$$

$$A^{[L]}, Z^{[L]}: n^{[L]} \times m$$

Loss functions



Mean Squared Error (MSE)

$$MSE(\hat{y}_{n \times 1}) = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

Prediction
Ground truth

Regression:
temperature
 y



Continuous Problem

notation: use hat \hat{y} for NN's predictions \hat{y}

- **Loss function** quantifies the difference between prediction (NN output, \hat{y}) and the ground truth (Labels, y)
- For regression:
 - Mean Squared Error (MSE)
- For classification:
 - Cross Entropy Loss (CE)

we can define different loss functions based on the application

widely used loss functions

Cross Entropy Loss

classes →
 $n=c$

$$CE(\hat{y}_{n \times 1}) = - \sum_{i=1}^c y_i \log \hat{y}_i$$

Ground truth {0,1} ↪

classification
Hot or cold
1
0

Classification using softmax activation

last layer $n^{[L]} = 4 \rightarrow$  4 classes of animals



3

1

2

0

3

2

0

1

Ground truth y : one vs zero

koalas	cats	dogs	chicks
[1]	[0]	[0]	[0]
0	1	0	0
0	0	1	0
0	0	0	1

How
to label

4 classes

This transformation is
called one vs zero
one hot encoding

Prediction \hat{y} : Softmax activation

$$Z^{[L]} = W^{[l]} A^{[l-1]} + b^{[L]}$$

$$\hat{y}_{n^{[L]} \times 1}^{[L](j)} = \text{softmax}(z^{[L](j)}) = \frac{e^{z^{[L](j)}}}{\sum_{i=1}^{n^{[L]}} e^{z_i^{[L](j)}}}$$

- Softmax converts a vector of numbers into a vector of probabilities.
- Probability of a value is proportional to its relative scale in the vector.

example of output

$$\begin{bmatrix} 5.5 \\ 1.2 \\ 0.2 \\ 0.01 \end{bmatrix} \xrightarrow{\text{Softmax}} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}$$

a is the closest to 1 \Rightarrow Koala

$a+b+c+d=1$

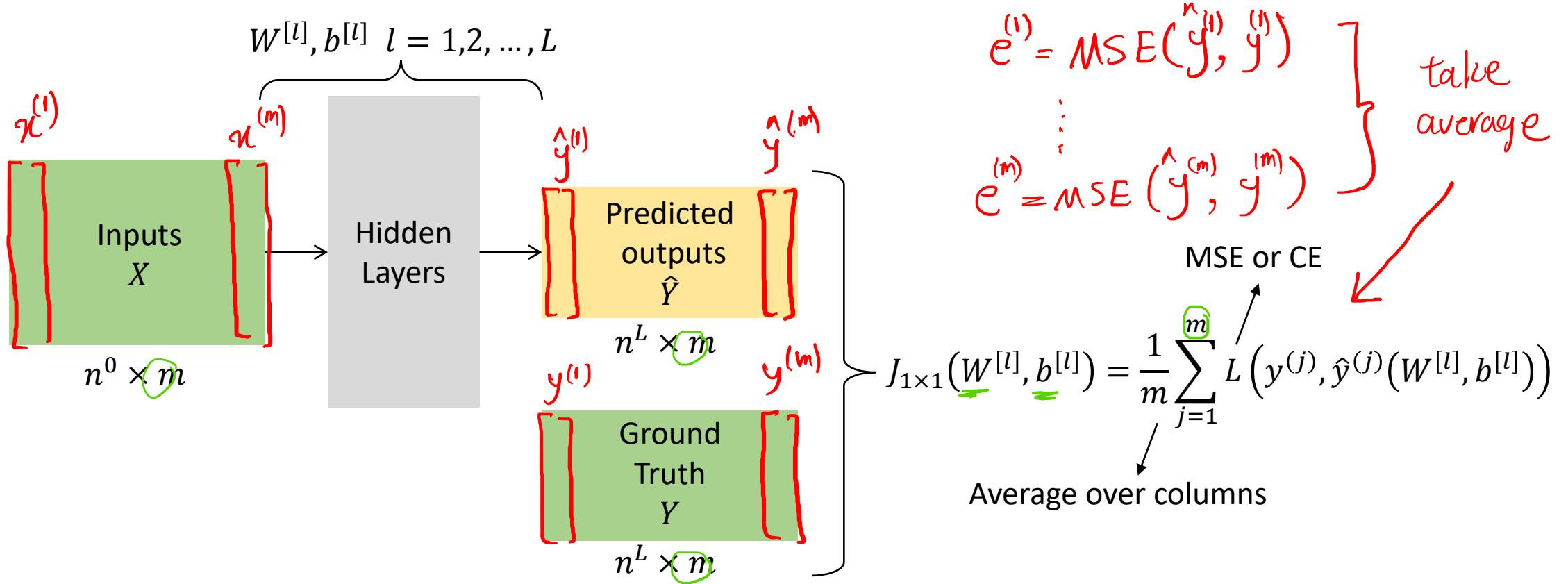
Cross Entropy Loss

$$CE(\hat{y}_{n^{[L]} \times 1}) = - \sum_{i=1}^{n^{[L]}} y_i \log \hat{y}_i$$

classes \downarrow
Prediction \downarrow
 $n^{[L]}$
Ground truth \uparrow

< 0
 $-\log a > 0$
minimized when $a \rightarrow 1$

Cost function of NNs as a function of parameters W, b



NN training problem:

Find $W^{[l]}, b^{[l]}$ for $l = 1, 2, \dots, L$ such that $J(W^{[1]}, b^{[1]}, \dots, W^{[L]}, b^{[L]})$ is minimized.

Gradient descent

Gradient descent (steepest descent) is an iterative optimization algorithm for finding a local minimum of a differentiable function

GD steps to find θ that approximately minimizes $J(\theta)$

Set

given by the user

- learning rate α
- Max. number of iterations I_{max}
- Stoppage criteria
- Initial guess/point θ_0

For $i = 0, 1, \dots, I_{max}$:

if stoppage criteria on θ_i met:

return θ_i

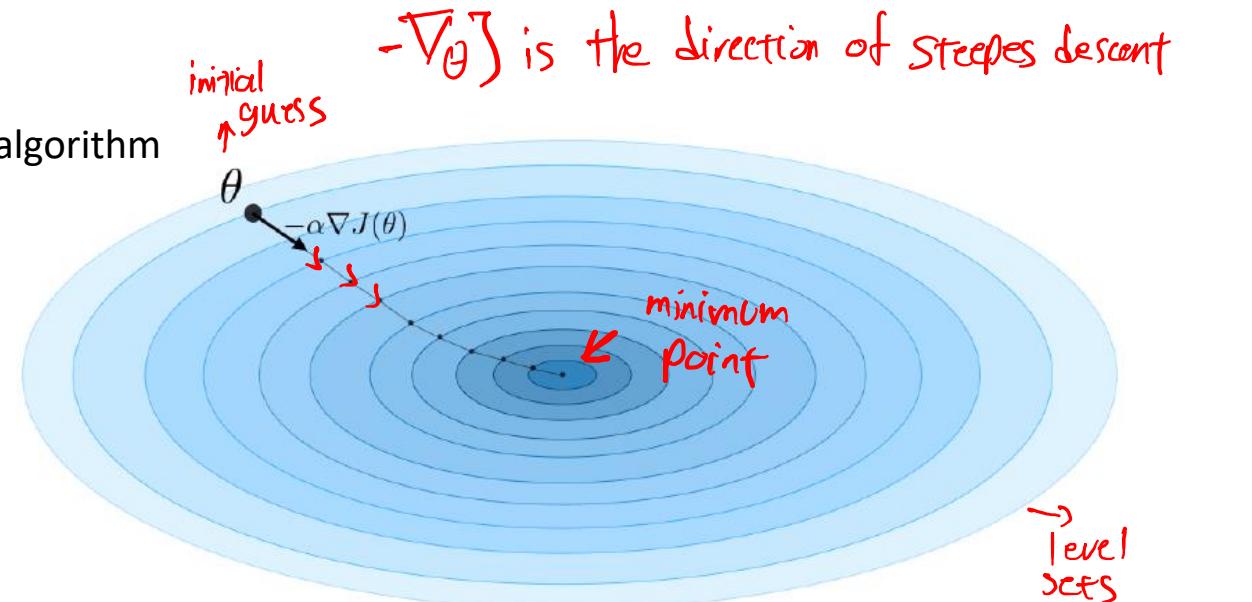
else:

$$\theta_{i+1} = \theta_i - \alpha \nabla_{\theta} J(\theta_i)$$

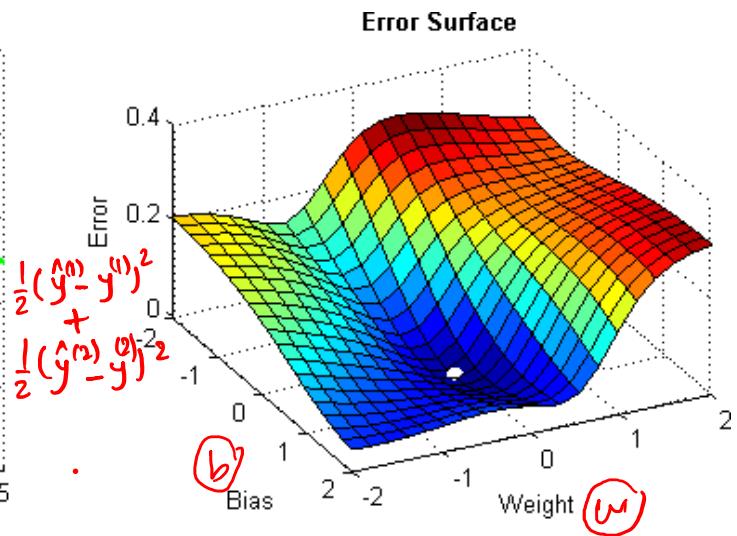
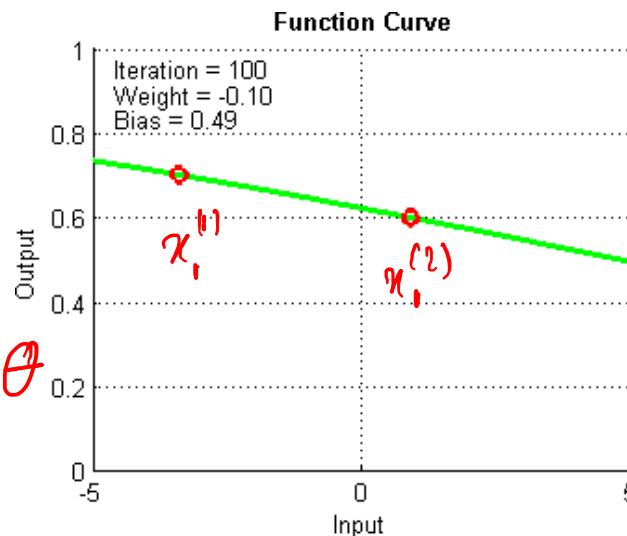
gradient of
cost function w.r.t. variables θ

important part is finding $\nabla_{\theta} J$

Design variables
(w, b)

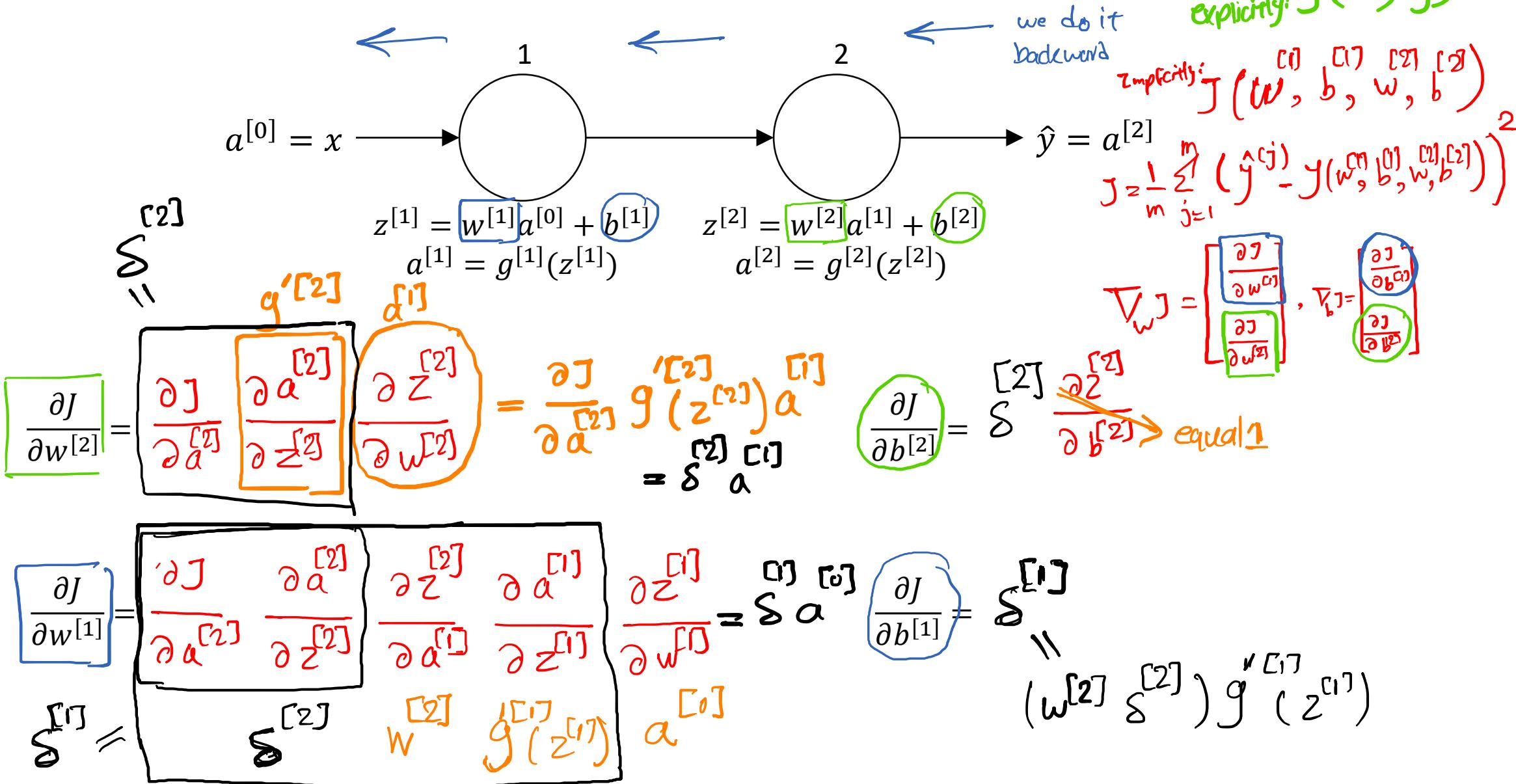


Level sets of J near a local minimum

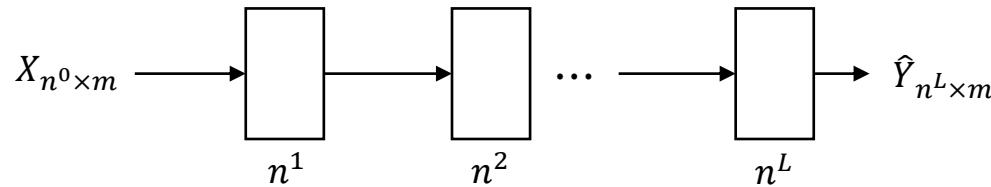


$$y = w\pi + b$$

Backpropagation – finding $\nabla_w J$, $\nabla_b J$ using chain-rule



Vectorized Forward and Backpropagation



Cost function:

$$J_{1 \times 1}(W^1, b^1, \dots, W^L, b^L) = \frac{1}{m} \sum_{j=1}^m L^{(j)}$$
$$L_{1 \times 1}^{(j)} = L\left(\hat{y}^{(j)}_{n^L \times 1}, y^{(j)}_{n^L \times 1}\right)$$
$$\hat{Y} = g^L(W^L A^{L-1} + b^L)$$

Forward propagation:

For $l = 1, 2, \dots, L$:

$$Z_{n^l \times m}^l = W_{n^l \times n^{l-1}}^l A_{n^{l-1} \times m}^{l-1} + b_{n^l \times 1}^l [1, \dots, 1]_{1 \times m}$$

$$A_{n^l \times m}^l = g^l(Z_{n^l \times m}^l)$$

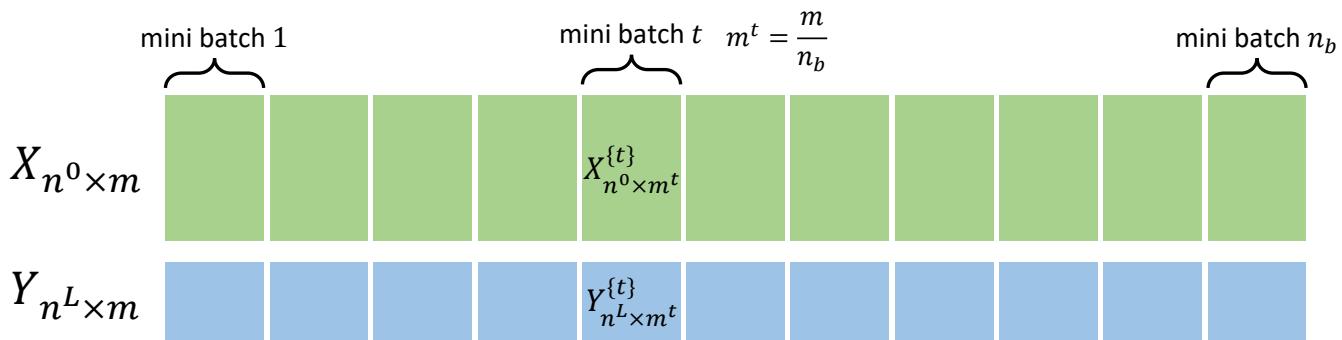
Backpropagation:

For $l = L, \dots, 2, 1$:

$$\Delta_{n^l \times m}^l = \begin{cases} \nabla_{a^l} J(A^l) \circ g'^l(Z^l), & l = L \\ \left(W^{l+1 \top} \Delta^{l+1}\right) \circ g'^l(Z^l), & 1 \leq l < L \end{cases}$$

$$\frac{\partial J}{\partial W^l} = \Delta^l A^{l-1 \top}, \quad \frac{\partial J}{\partial b^l} = \Delta^l \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}_{m \times 1}$$

Mini-batch gradient descent



While SG stoppage criteria is not met and $i < I_{max}$:

For $t = 1, \dots, n_h$:

Forward prop using $X^{\{t\}} = A^{\{t\}[0]}$

For $l = 1, \dots, L$:

$$Z^{[t][l]} = W^{[l]} A^{[t][l-1]} + b^{[l]}$$

$$A^{\{t\}[l]} = g^{[l]}(Z^{\{t\}[l]})$$

1 epoch ←
1 pass through
entire training set

Compute cost function $J^{\{t\}} = \frac{1}{m^t} \sum_{j=1}^{m^t} L(y^{\{t\}(j)}, \hat{y}^{\{t\}(j)})$

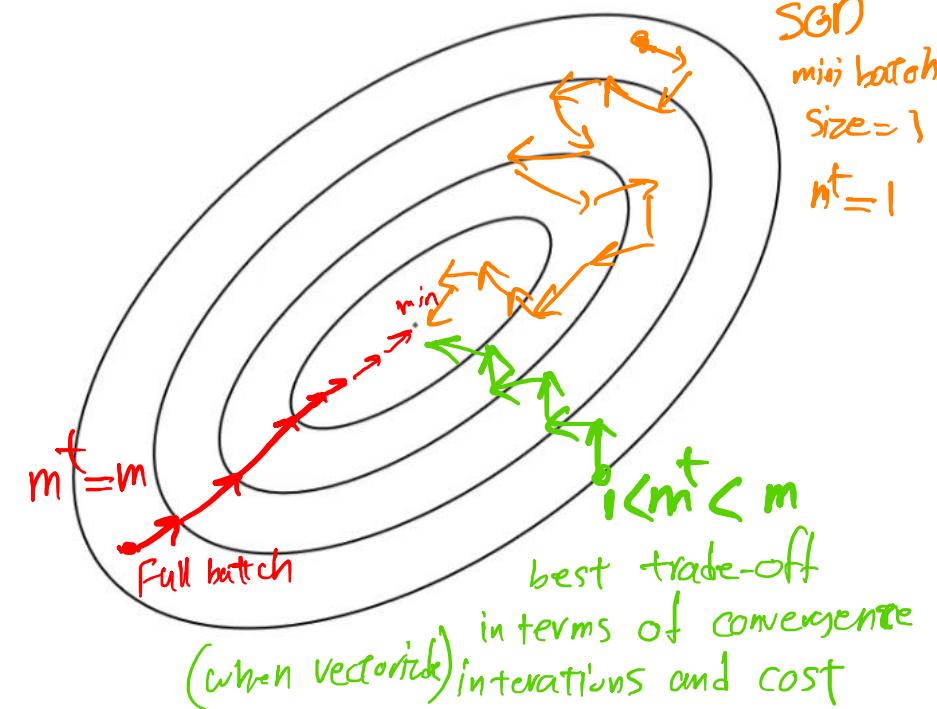
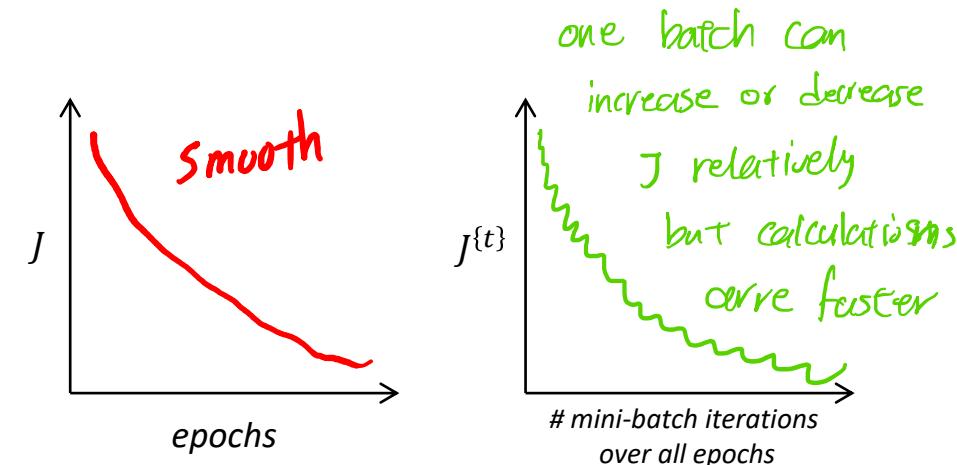
Backprop to compute gradients $\nabla_{w^{[l]}} J^{\{t\}}, \nabla_{h^{[l]}} J^{\{t\}}$

SG update:

$$W^{[l]} = W^{[l]} - \alpha \nabla_{W^{[l]}} J^{\{t\}}$$

$$b^{[l]} = b^{[l]} - \alpha \nabla_{b^{[l]}} J^{\{t\}}$$

Not the exact gradients
for entire data
just for batch $\{t\}$



What are hyperparameters?

Parameters: $W^{[l]}, b^{[l]}$ for $l = 1, 2, \dots, L$

Hyperparameters:

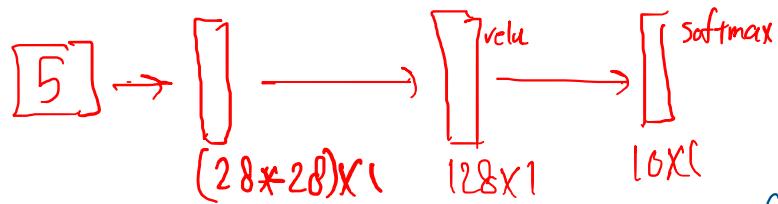
Parameters that control the final values of parameters (W, b) obtained in backpropagation

Some examples:

- Learning rate α
- Minimization method (GD, mini-batch GD, Adam , etc.)
- Max. number of minimization iterations
- Number of hidden layers
- Hidden unites $n^{[l]}$ for $l = 1, 2, \dots$
- Choice of activation functions (sigmoid, tanh, ReLU, etc.) $g^{[l]}$ for $l = 1, 2, \dots, L$
- Mini batch size
- Regularization parameters
- Momentum term
- Any other parameters that we set before training for our model

Hyperparameter search/optimization/tuning: Finding the optimal set of hyperparameters for the learning process and model

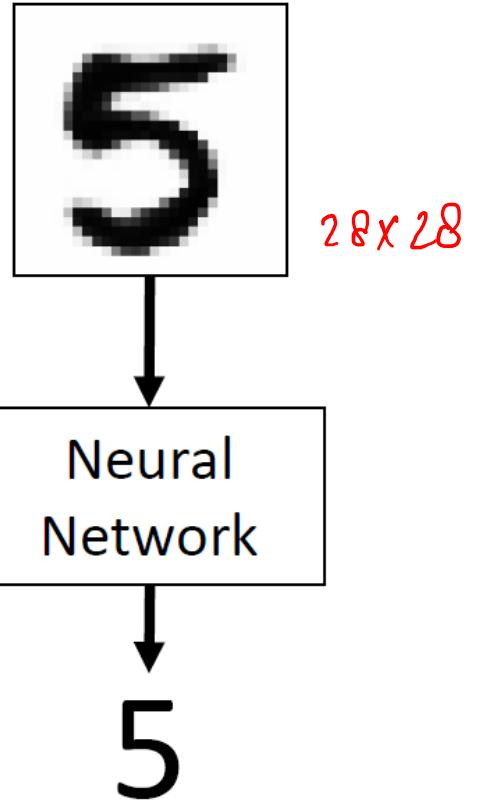
TensorFlow



all steps can be done
just by a few lines of code!

- 1 # import tensorflow and keras (tf.keras not "vanilla" Keras)
import tensorflow as tf
from tensorflow import keras
- 2 # get data
(train_images, train_labels), (test_images, test_labels) = \
keras.datasets.mnist.load_data()
- 3 # setup model
model = keras.Sequential([
 keras.layers.Flatten(input_shape=(28, 28)),
 keras.layers.Dense(128, activation=tf.nn.relu),
 keras.layers.Dense(10, activation=tf.nn.softmax)
])
- 4 model.compile(optimizer=tf.train.AdamOptimizer(),
 loss='sparse_categorical_crossentropy',
 metrics=['accuracy'])
- 5 # train model
model.fit(train_images, train_labels, epochs=5)
- 6 # evaluate
test_loss, test_acc = model.evaluate(test_images, test_labels)
print('test accuracy:', test_acc)
- 7 # make predictions
predictions = model.predict(test_images)

Input Image:



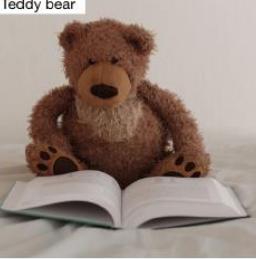
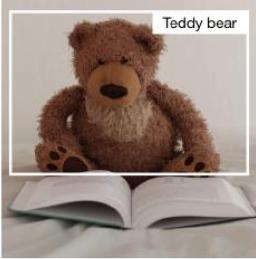
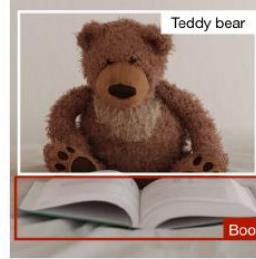
TensorFlow
Model:

Output:

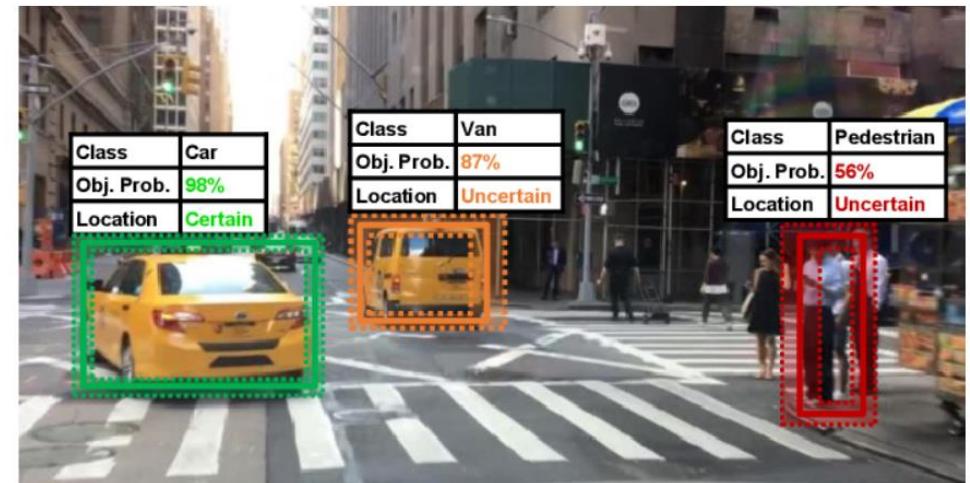
most of the time is correct

Convolutional Neural Networks (CNNs): Computer Vision Problems

Object detection: The goal of neural style transfer is to generate an image G based on a given content C and a given style S

Image classification	Classification w. localization	Detection
		
- Classifies a picture - Predicts probability of object	- Detects object in a picture - Predicts probability of object and where it is located	- Detects up to several objects in a picture - Predicts probabilities of objects and where they are located

Self-driving cars



Neural style transfer: The goal of neural style transfer is to generate an image G based on a given content C and a given style S



Face recognition: Is this one of the K persons in the database

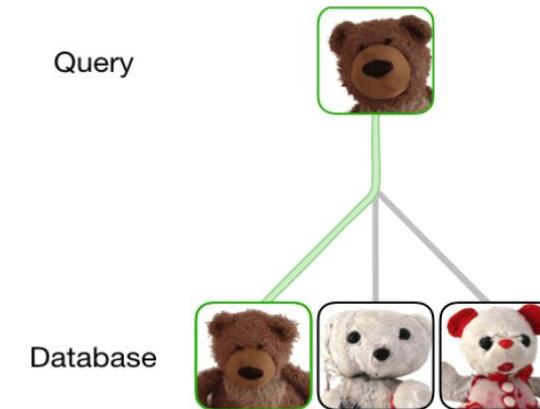
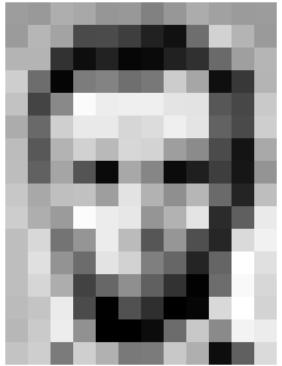


Image representation as matrices

Grayscale image

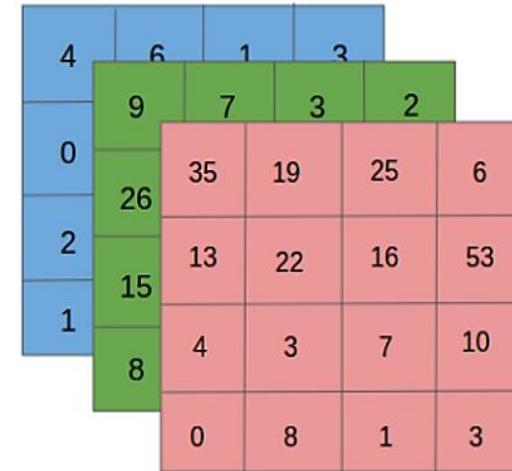
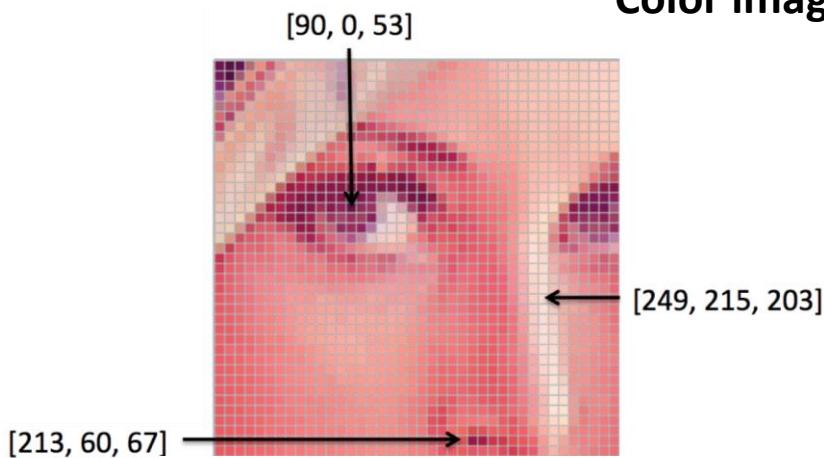


167	153	174	168	160	152	129	151	172	161	155	156
155	182	163	74	75	62	93	17	110	210	180	154
180	180	50	14	34	6	16	33	48	106	159	181
206	109	5	124	191	111	120	204	166	16	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	106	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	159	76	20	169
189	97	165	84	10	158	134	11	31	62	22	148
159	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
160	216	116	149	236	167	85	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
150	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	129	200	175	13	96	218

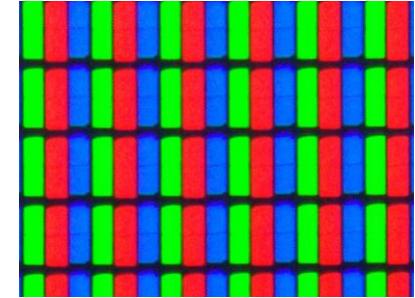
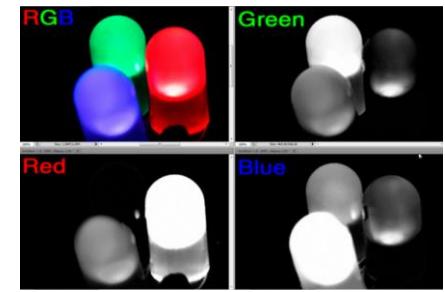
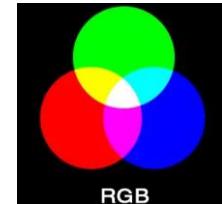
3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

$(n \times m \times 1)$

Color image



$(n \times m \times 3)$



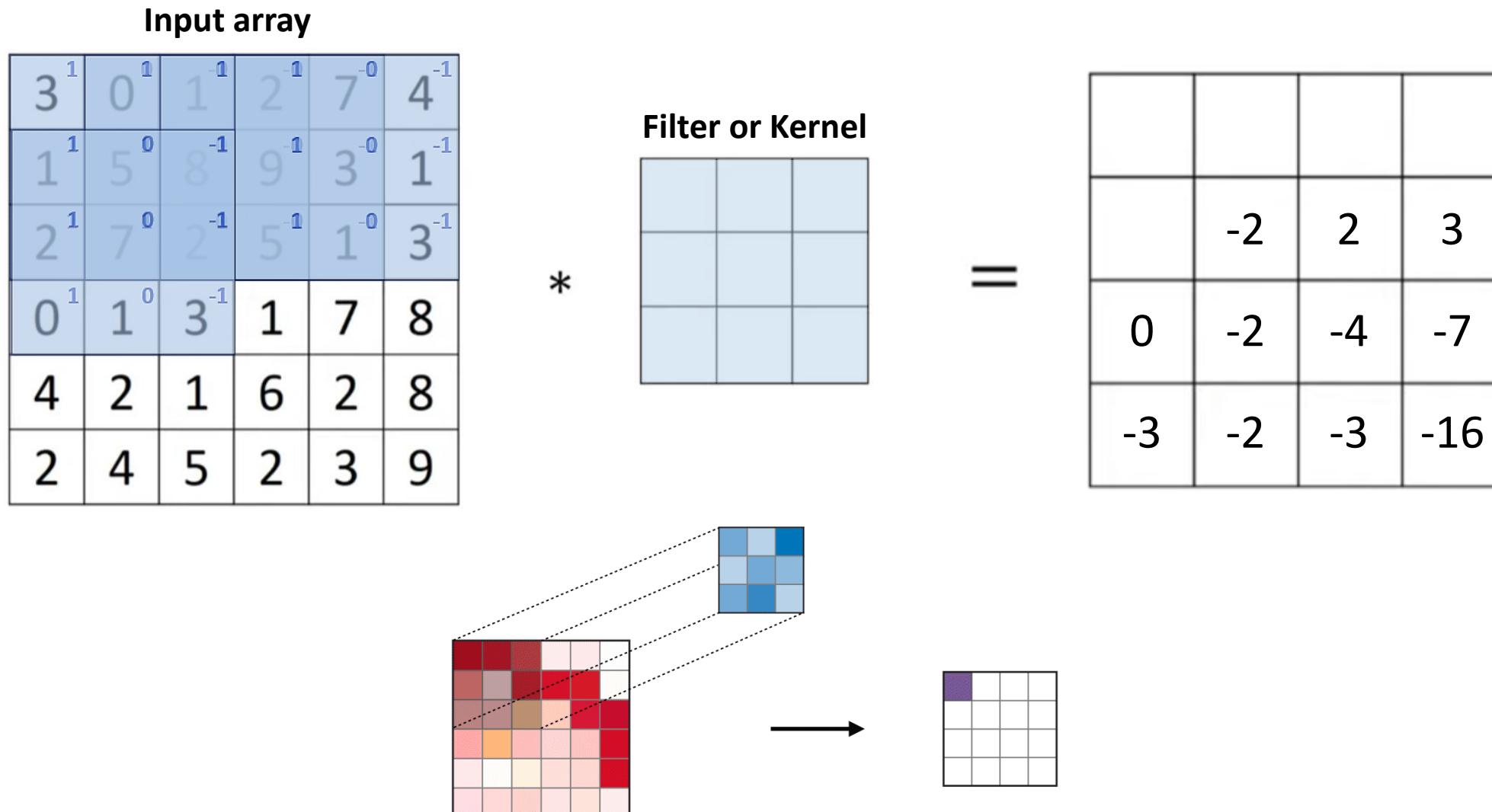
Monitor/TV screen

Images from

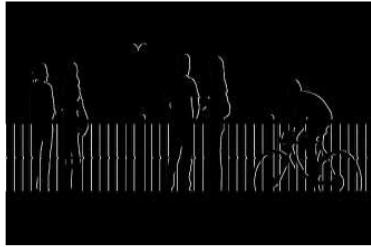
<https://ai.stanford.edu/~syyeung/cvweb/tutorial1.html>

<https://www.howtogeek.com/42393/rgb-cmyk-alpha-what-are-image-channels-and-what-do-they-mean/>

What is convolution in deep learning?



Convolution for edge detection



Horizontal edge detection

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10

*

1	1	1
0	0	0
-1	-1	-1

=

0	0	0	0
30	10	-10	-30
30	10	-10	-30
0	0	0	0



10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

*

1	0	-1
1	0	-1
1	0	-1



0	30	30	0
0	30	30	0
0	30	30	0
0	30	30	0

=

0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10

*

1	0	-1
1	0	-1
1	0	-1



0	-30	-30	0
0	-30	-30	0
0	-30	-30	0
0	-30	-30	0

=



Convolution for edge detection

Other edge detection filters:

1	0	-1
2	0	-2
1	0	-1

Vertical Sobel filter

3	0	-3
10	0	-10
3	0	-3

Vertical Scharr filter



Developing convolution filters

Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

Gaussian blur



$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Deep learning: consider a filter as some parameters and let backpropagation learn the best possible filter based on data and the loss function

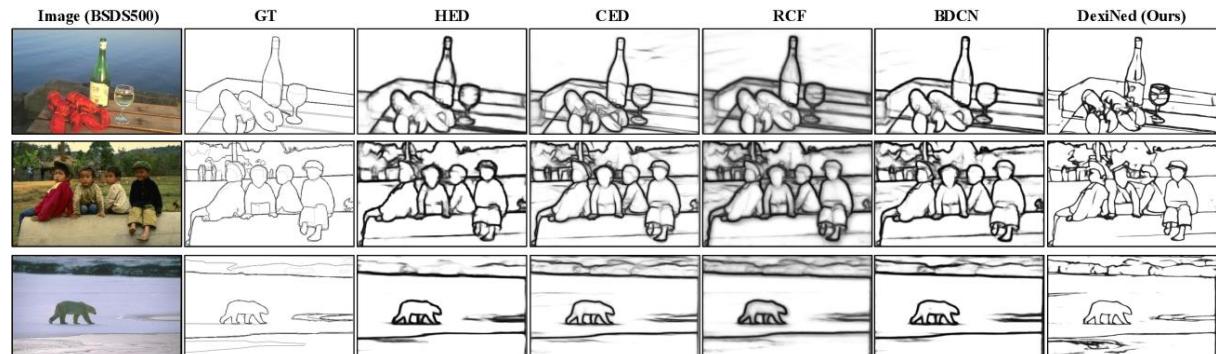
3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

*

w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

filter

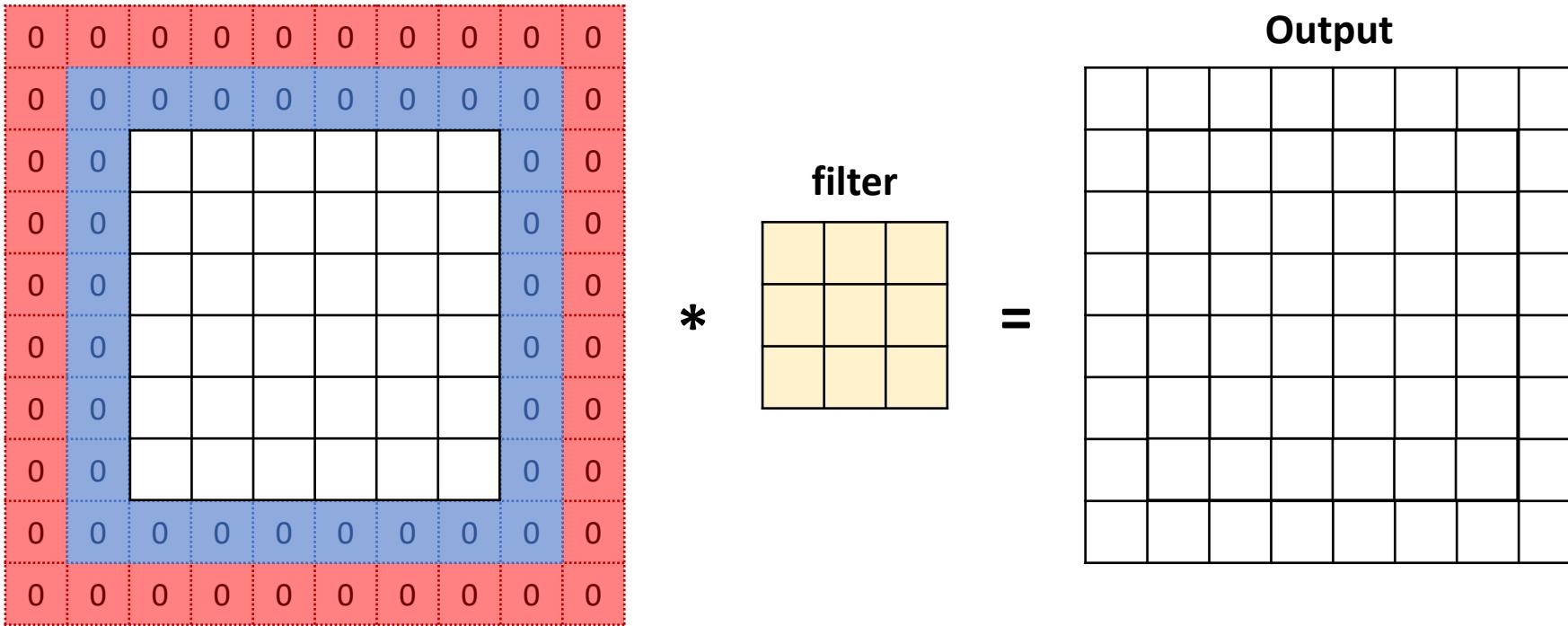
9 parameters



Edge detection using deep CNNs

<https://doi.org/10.1109/WACV45572.2020.9093290>

Padding in convolution



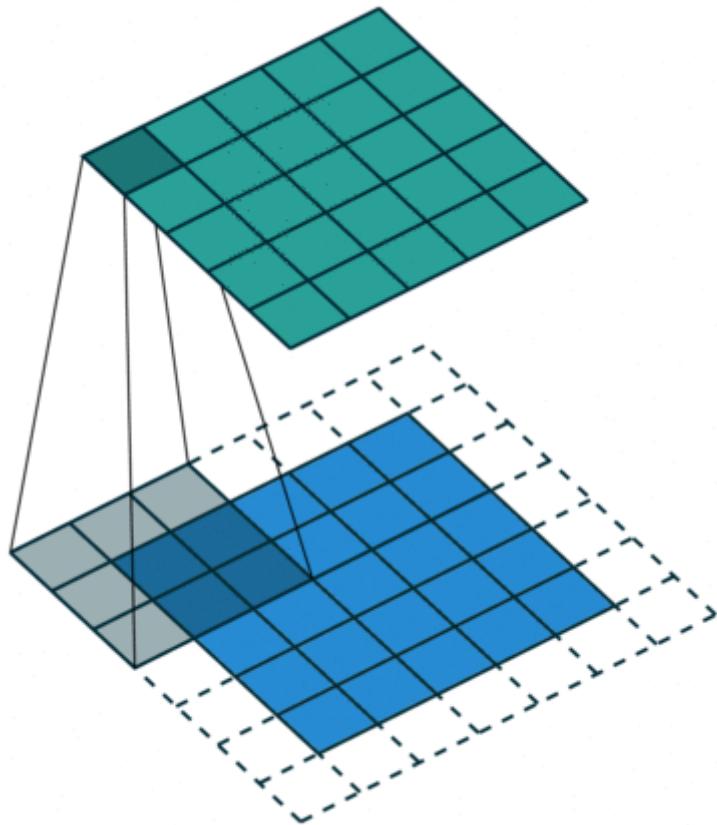
Output size given input size of $n \times n$ and filter size f :

Without padding

- Output size shrinks
- losing information at the edges – pixels at the edges used less

Output size given input size of $n \times n$, filter size f , and padding p :

Padding in convolution



Two common approaches:

“Valid” convolution – no padding $p = 0$

Input size = $m \times n$

Output size = $(m - f + 1) \times (n - f + 1)$

“Same” convolution – pad such that the output size is the same as the input size (f is usually odd)

Input size = Output size = $n \times n$

Strided convolution

2	3	7	4	6	2	9
6	6	9	8	7	4	3
3	4	8	3	8	9	7
7	8	3	6	6	3	4
4	2	1	8	3	4	6
3	2	4	1	9	8	3
0	1	3	9	2	1	4

*

3	4	5
1	0	2
-1	0	3

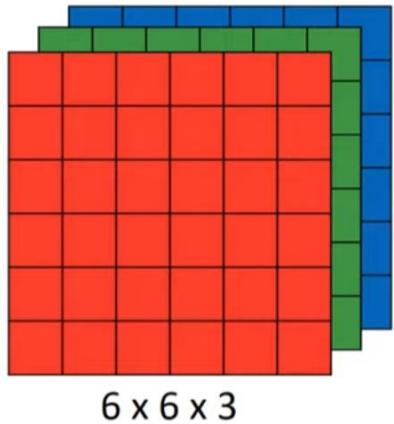
=

91	100	83
69	91	127
44	72	74

Output size given, input $n \times n$, filter size f , padding p , stride s :

2D Convolution over volume

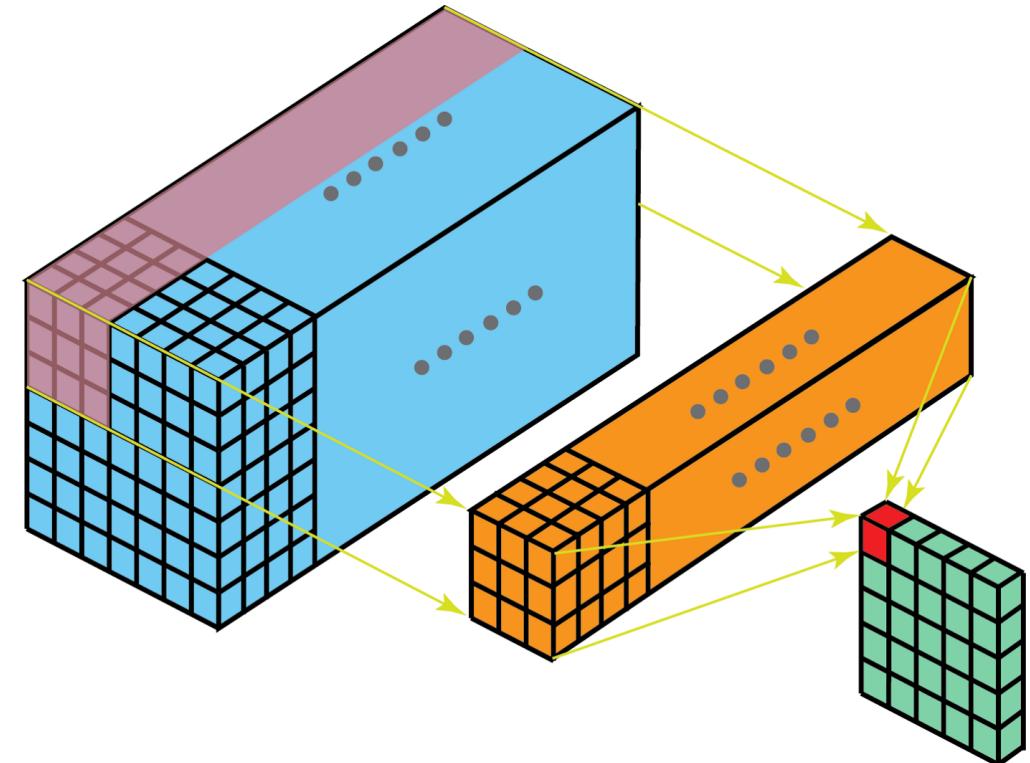
Color image (RGB) channels



$$\begin{matrix} * & \begin{matrix} 3 \times 3 \times 3 \end{matrix} \\ \begin{matrix} 3 \times 3 \times 3 \end{matrix} & = \begin{matrix} 4 \times 4 \end{matrix} \end{matrix}$$

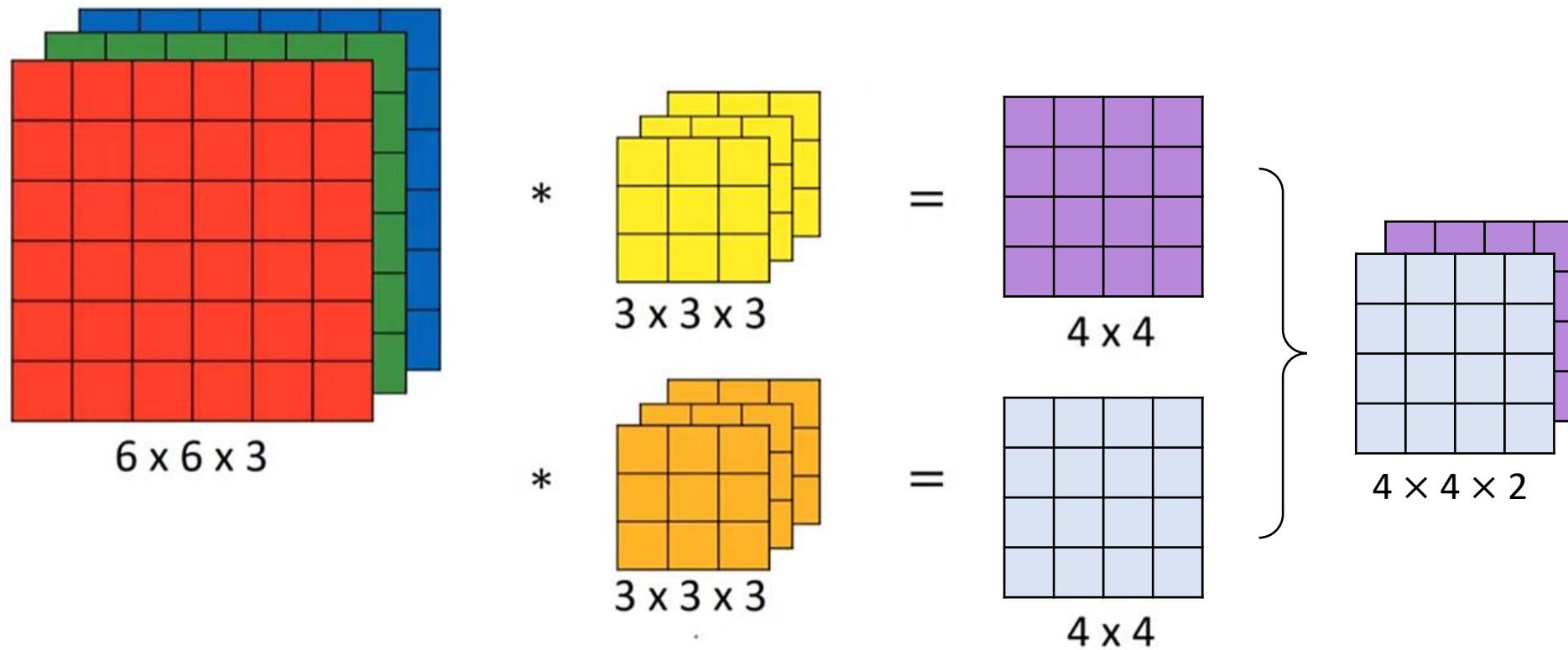
A diagram showing the convolution operation. On the left is a 3D input volume labeled $3 \times 3 \times 3$. In the center is a 3D filter kernel labeled $3 \times 3 \times 3$. To the right is a 2D output feature map labeled 4×4 . The asterisk (*) symbol indicates the convolution operation.

Arbitrary number of input channels



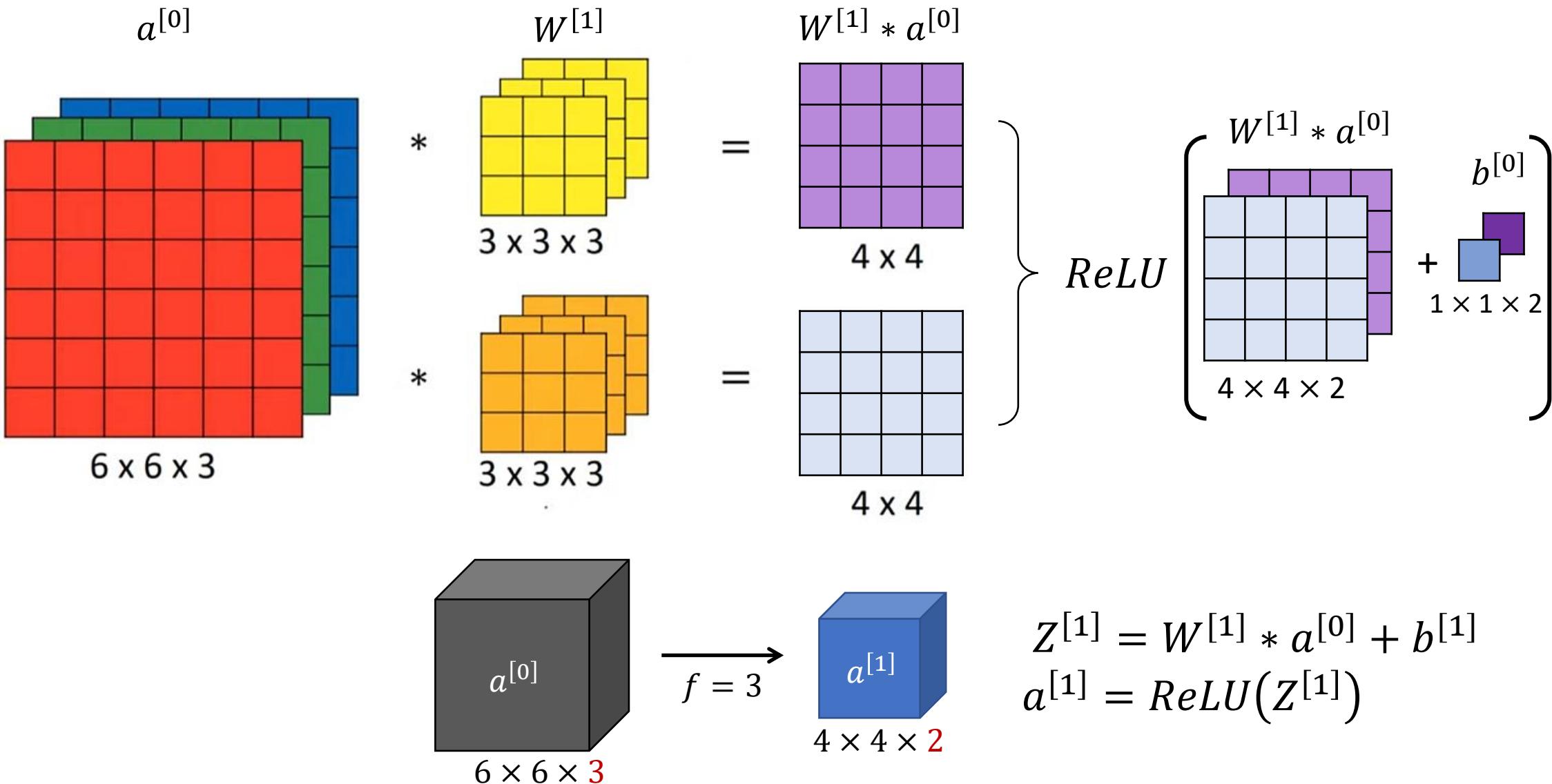
Input and filter have the same number of channels:

2D Convolution over volume – multiple filters



Output size given input size $n \times n \times n_c$ and filter size $f \times f \times n_c$:

One layer of a convolutional network



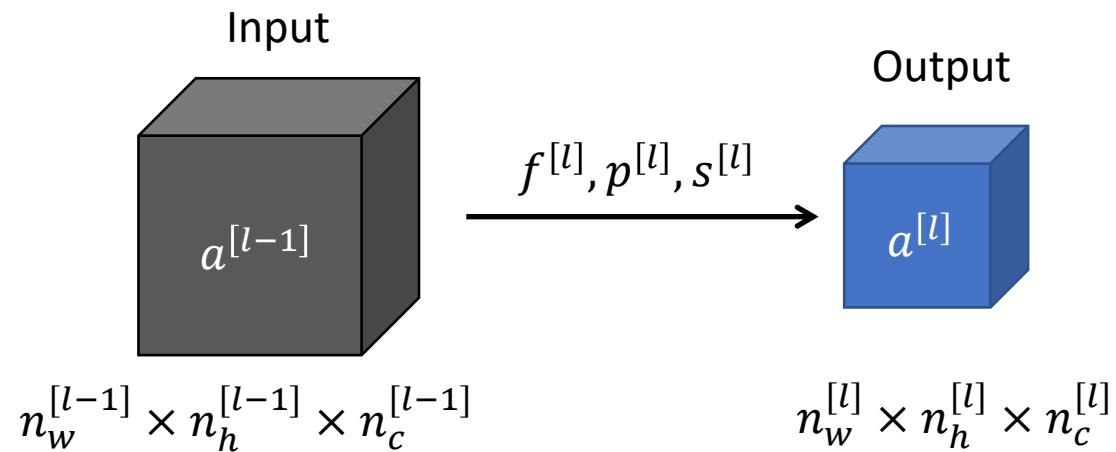
Size relations in convolutional networks

If layer l is a 2D convolution layer:

$f^{[l]}$ = filter size

$p^{[l]}$ = padding size

$s^{[l]}$ = stride



Find the following:

Number of filters:

Filter size:

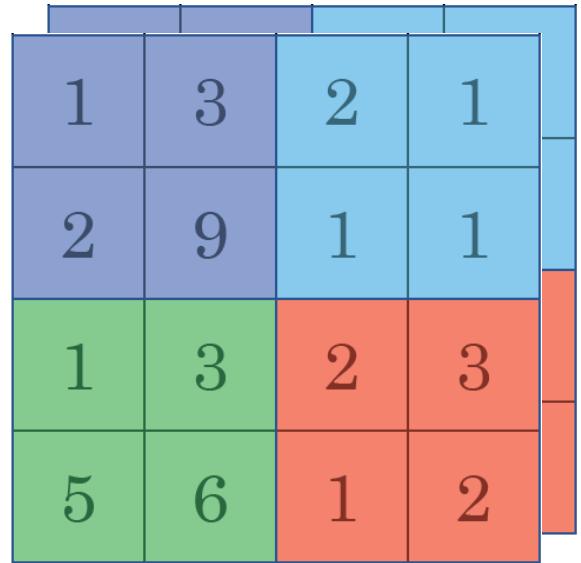
Output size:

Weight size:

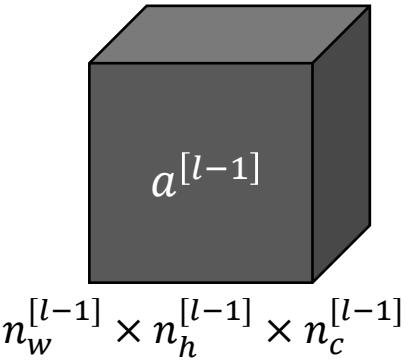
Bias size:

Activation size:

Pooling layers

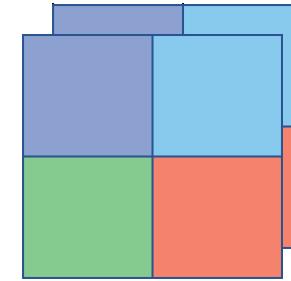


Input



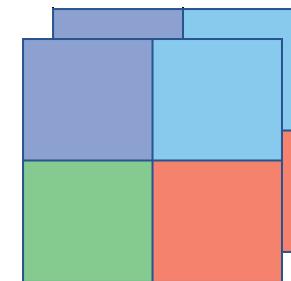
Max pooling

$$\xrightarrow{f = 2, S = 2}$$



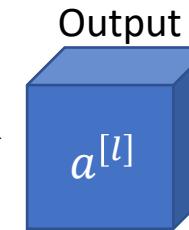
Average pooling

$$\xrightarrow{f = 2, S = 2}$$



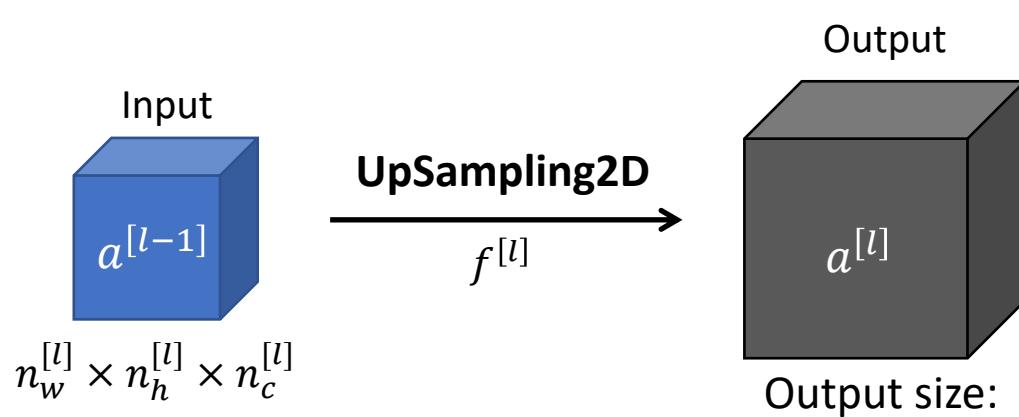
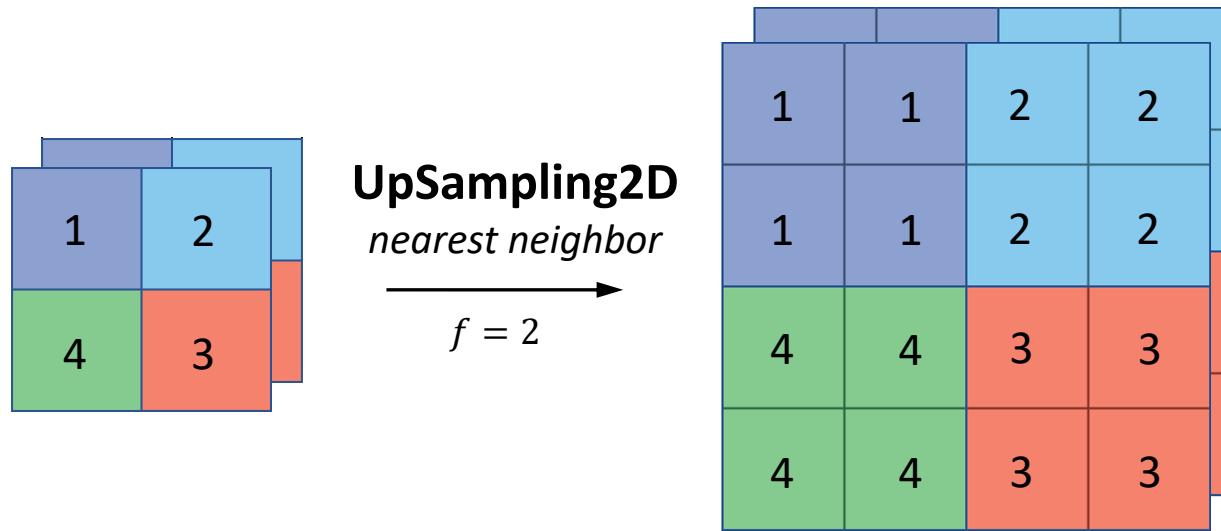
pooling

$$\xrightarrow{f^{[l]}, p^{[l]}, s^{[l]}}$$

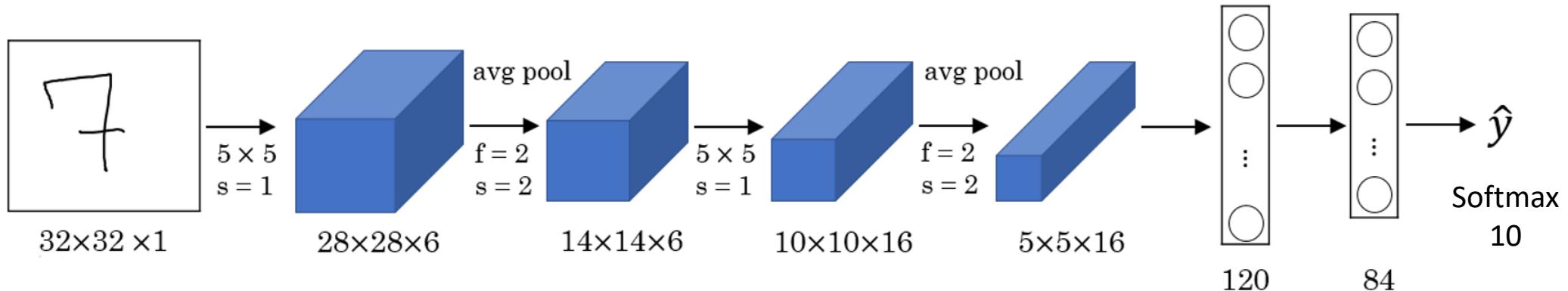


Output size:

UpSampling



Convolutional network example: LeNet - 5



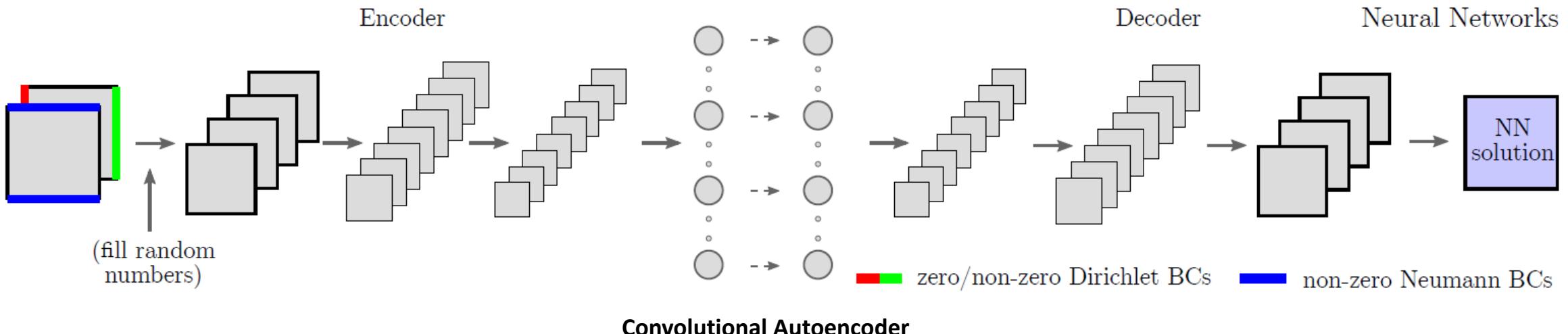
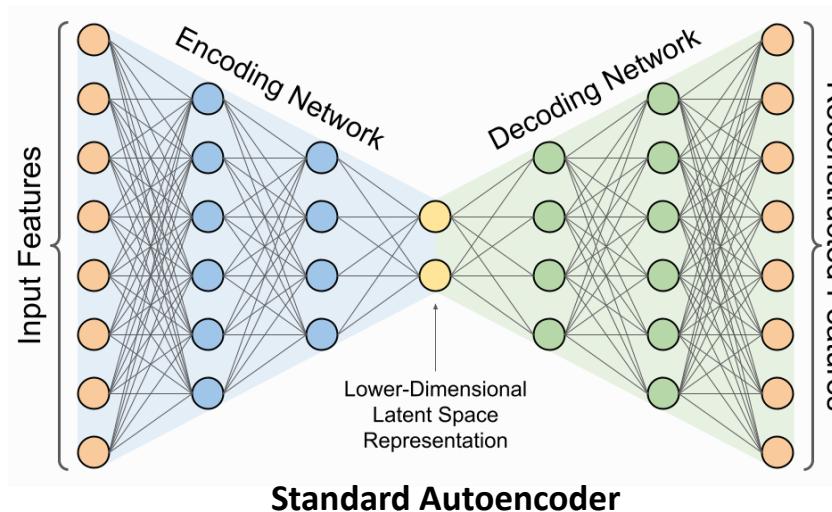
	Activation shape	Activation Size	# parameters
Input:	(32,32,3)	3,072	0
CONV1 ($f=5$, $s=1$)	(28,28,8)	6,272	208
POOL1	(14,14,8)	1,568	0
CONV2 ($f=5$, $s=1$)	(10,10,16)	1,600	416
POOL2	(5,5,16)	400	0
FC3	(120,1)	120	48,001
FC4	(84,1)	84	10,081
Softmax	(10,1)	10	841

Why convolution layers?

Parameter sharing: A feature detector (such as a vertical edge detector) that's useful in one part of the image is probably useful in another part of the image.

Sparsity of connections: In each layer, each output value depends only on a small number of inputs- much fewer weights to train

ML-PDE: NN architecture

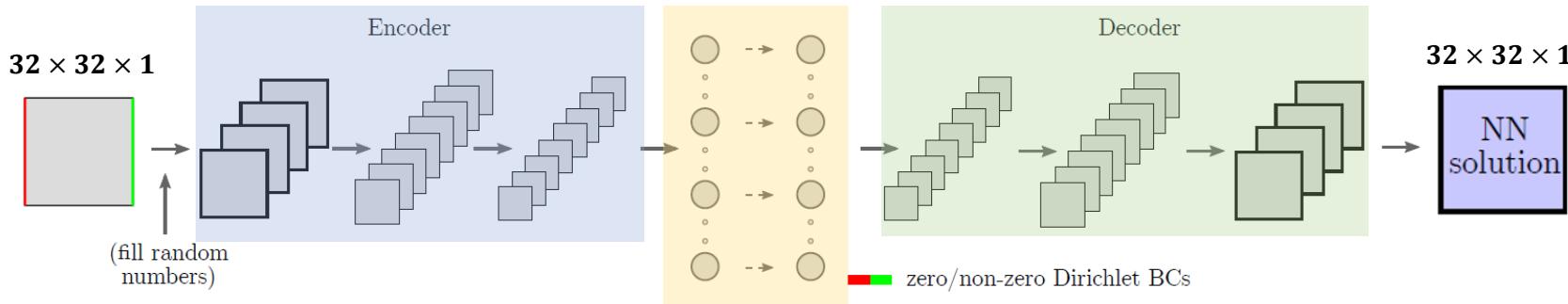


Zhang, X., & Garikipati, K. (2023). [Label-free learning of elliptic partial differential equation solvers with generalizability across boundary value problems](#).

Computer Methods in Applied Mechanics and Engineering, 116214

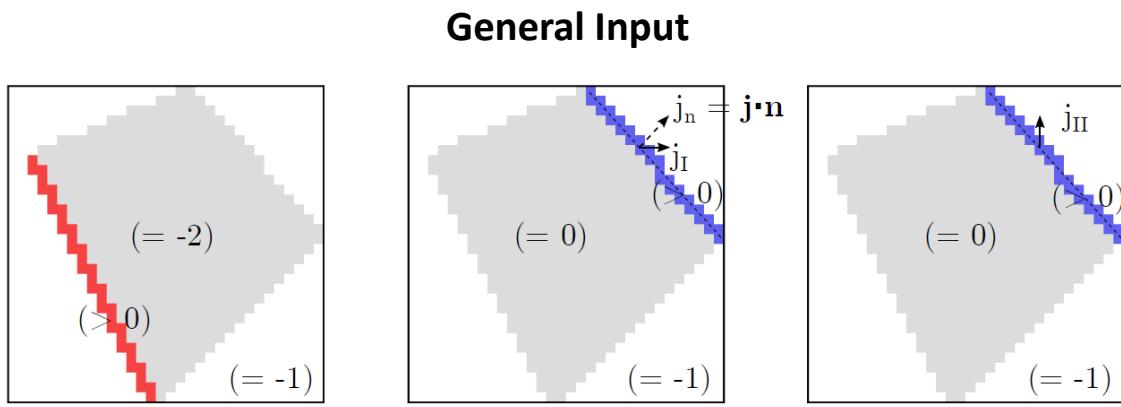
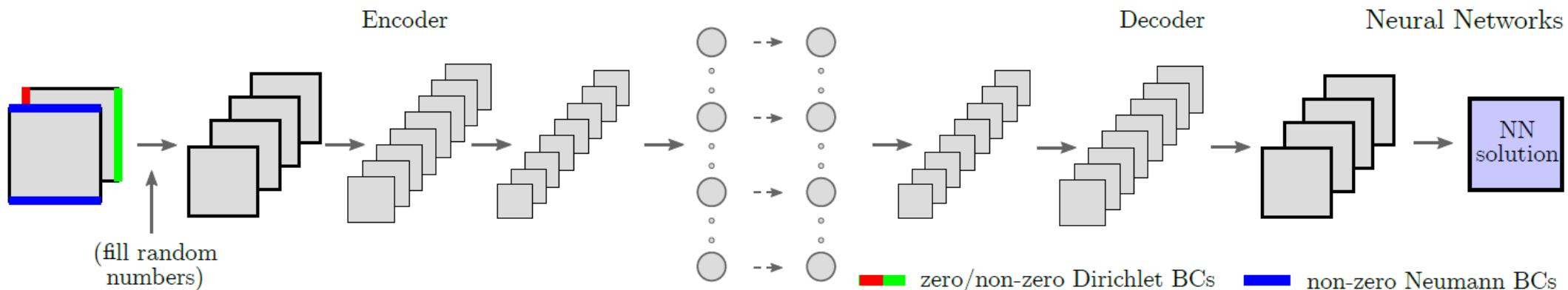
Image from <https://www.assemblyai.com/blog/introduction-to-variational-autoencoders-using-keras/>

ML-PDE: NN architecture

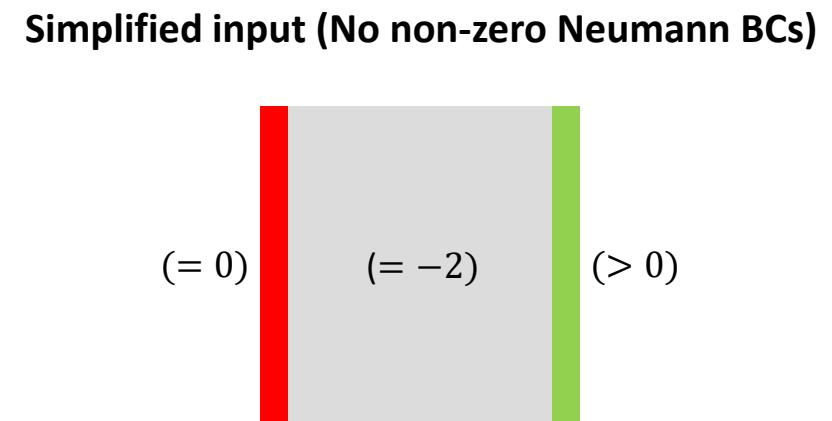


Deterministic	Size	Layer arguments
Input	-	-
LayerFillRandomNumber	-	-
Conv2D	filters = 8	kernel (5,5), padding: same, ReLU
MaxPooling2D	-	kernel (2,2), padding: same
Conv2D	filters = 8	kernel (5,5), padding: same, ReLU
MaxPooling2D	-	kernel (2,2), padding: same
Conv2D	filters = 8	kernel (5,5), padding: same, ReLU
MaxPooling2D	-	kernel (2,2), padding: same
Flatten	-	-
Dense	units = 32	ReLU
Dense	units = 32	ReLU
Reshape	-	$[4, 4, \frac{1}{2}]$
Conv2D	filters = 8	kernel (5,5), padding: same, ReLU
UpSampling2D	-	size (2,2)
Conv2D	filters = 8	kernel (5,5), padding: same, ReLU
UpSampling2D	-	size (2,2)
Conv2D	filters = 8	kernel (5,5), padding: same, ReLU
UpSampling2D	-	size (2,2)
Conv2D	filters = 8	kernel (5,5), padding: same, ReLU
Conv2D	filters = 1	kernel (5,5), padding: same, Linear

ML-PDE: NN Input

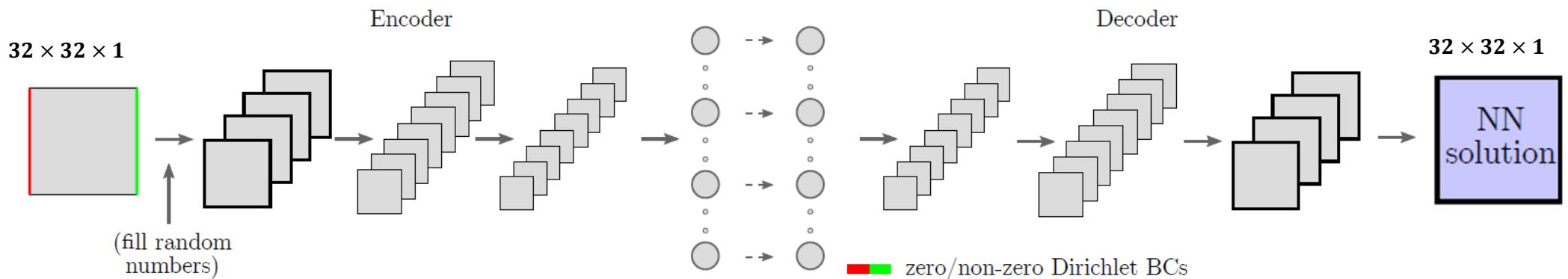


- $= -1$: margin between problem domain and background grid
- > 0 : boundary locations with boundary values
- $= -2$: domain interior filled with random numbers during training

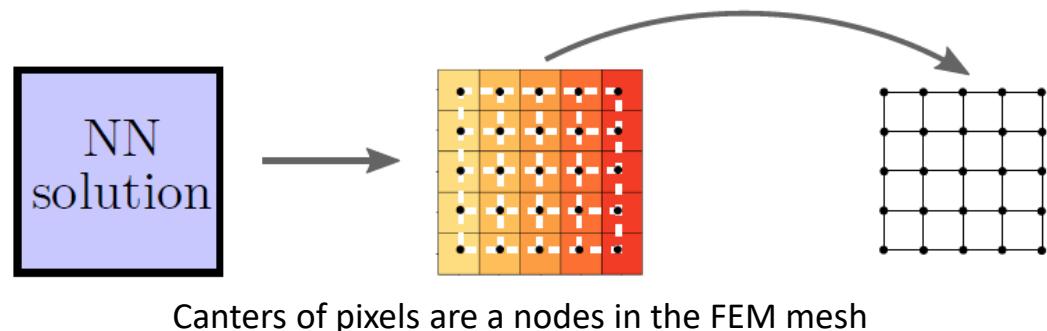


- $= 0$: zero Dirichlet boundary locations
- > 0 : non-zero Dirichlet boundary locations with boundary values
- $= -2$: domain interior filled with random numbers during training

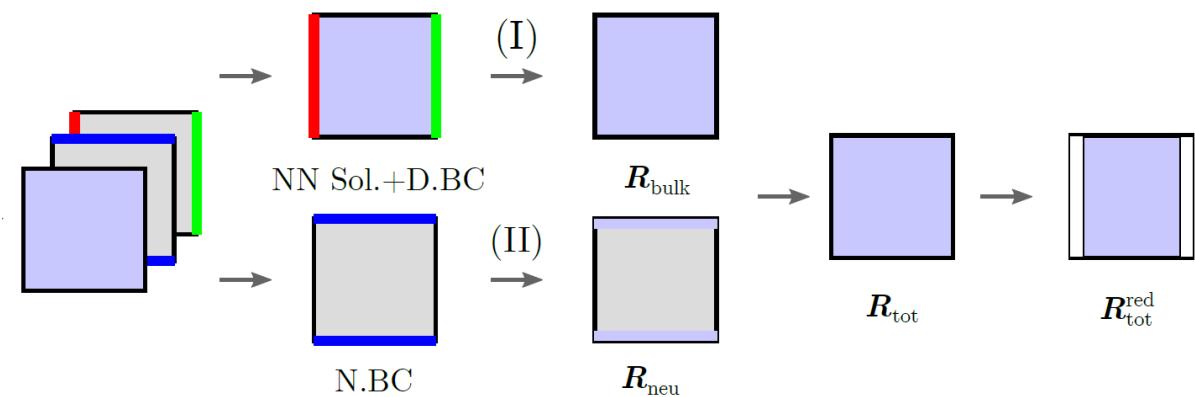
ML-PDE: NN output and loss function



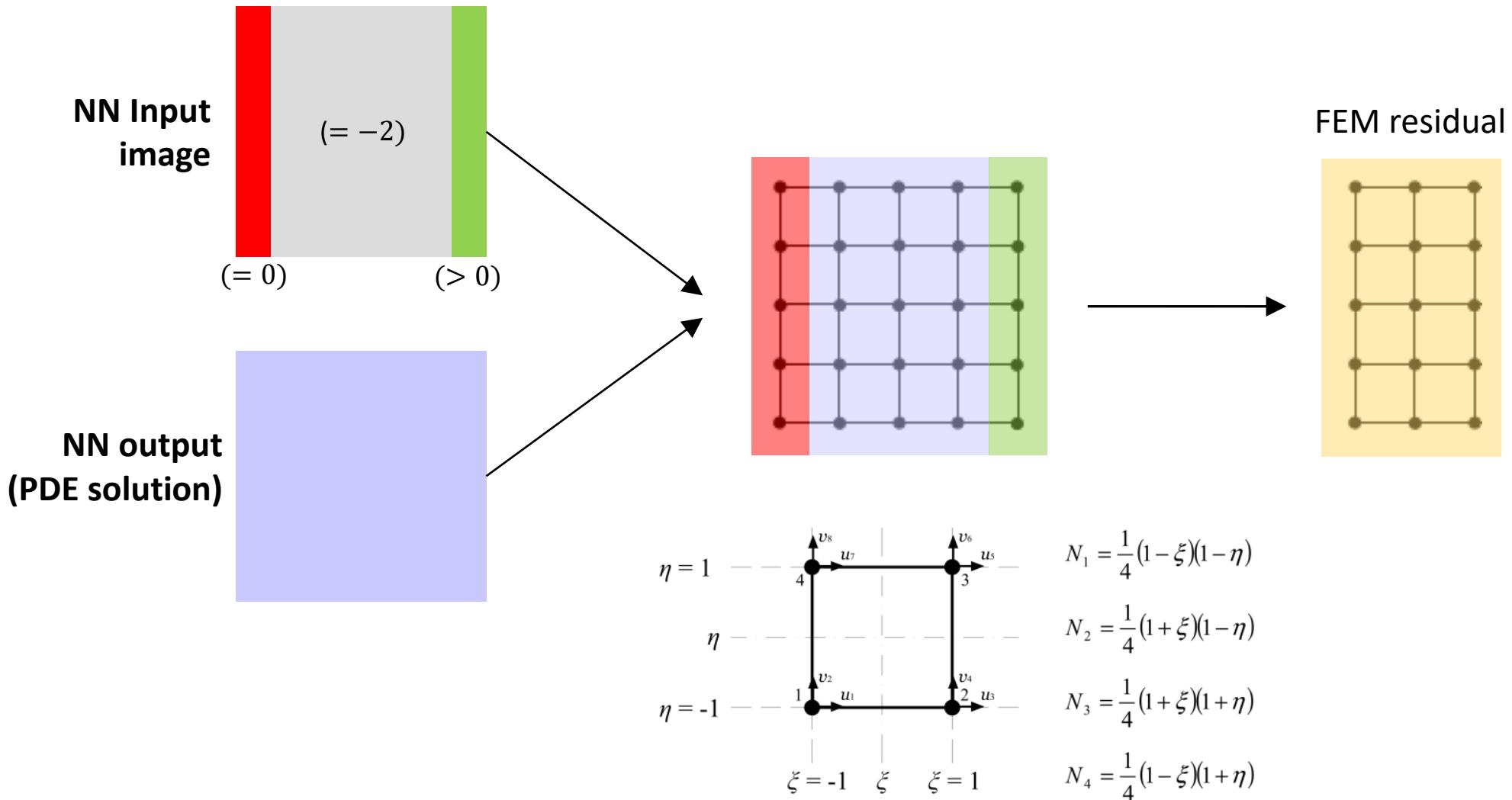
NN output is the solution of a PDE discretized by FEM



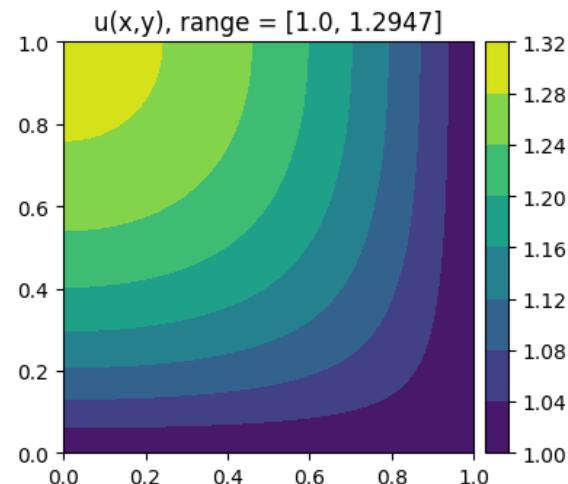
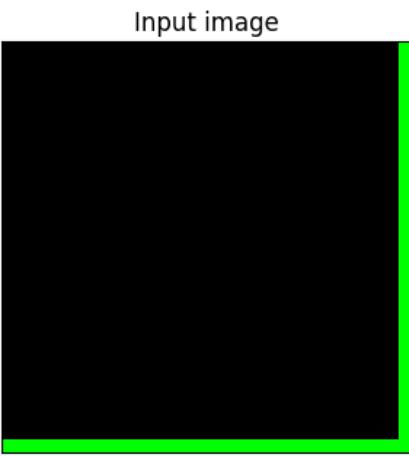
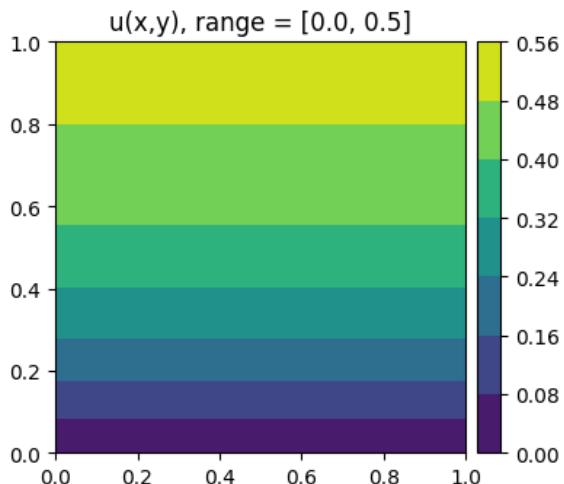
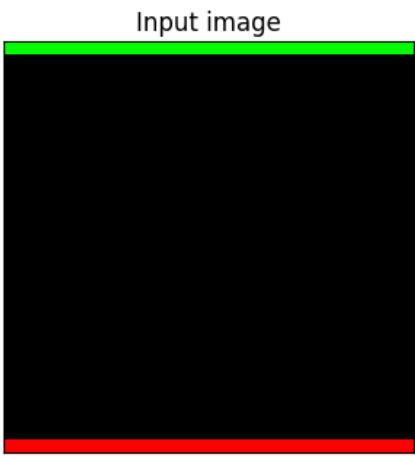
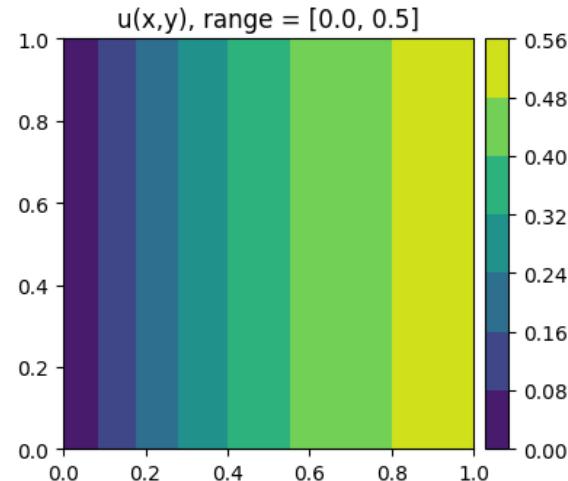
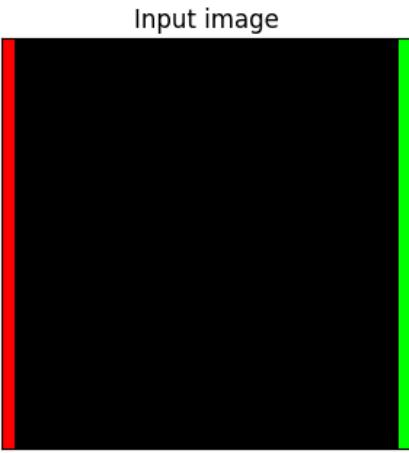
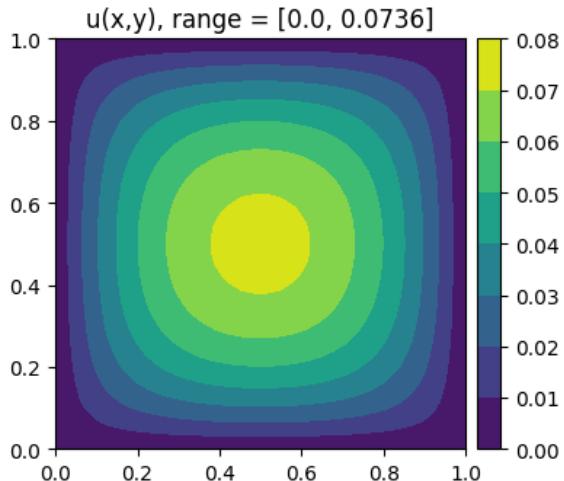
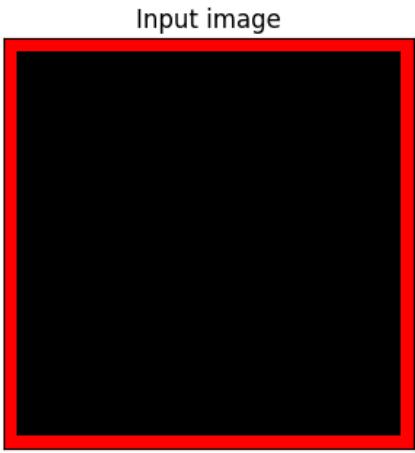
Label-free Loss function is based on residual of a PDE discretized by FEM



ML-PDE: Simplified loss function using FEM (project part 1)

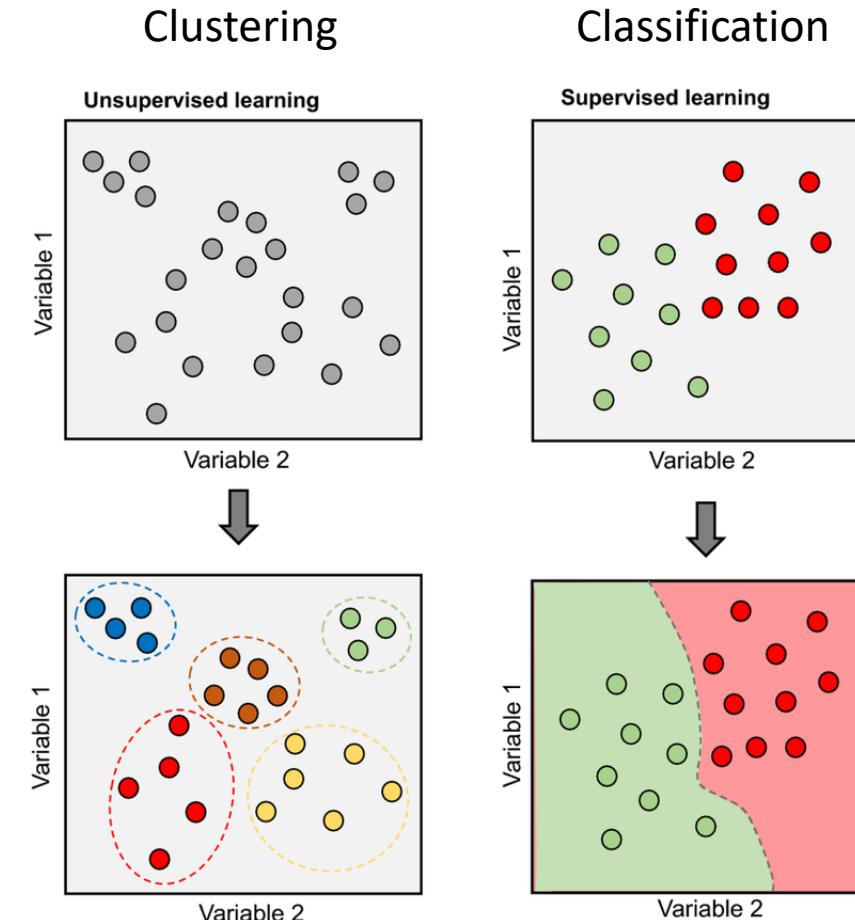


Sample results



Supervised vs unsupervised learning

Unsupervised learning	Supervised learning
Unsupervised learning is the type of machine learning that happens without human supervision. A machine tries to find any patterns in data by itself.	Supervised learning is the type of machine learning that happens under human supervision, meaning people label input data with answer keys showing a machine the desired outputs.
Unlabeled	Labeled
A model is given only input variables (X) and no corresponding output data.	A model is given input variables (X), output variables (Y), and an algorithm to learn the function from input to output.
You don't know what you're looking for in data.	You know what you're looking for in data.
Clustering and association problems	Classification and regression problems



Images from

<https://doi.org/10.1186/s12052-021-00147-x>

Classification <https://www.altexsoft.com/blog/unsupervised-machine-learning/>

Classification metrics

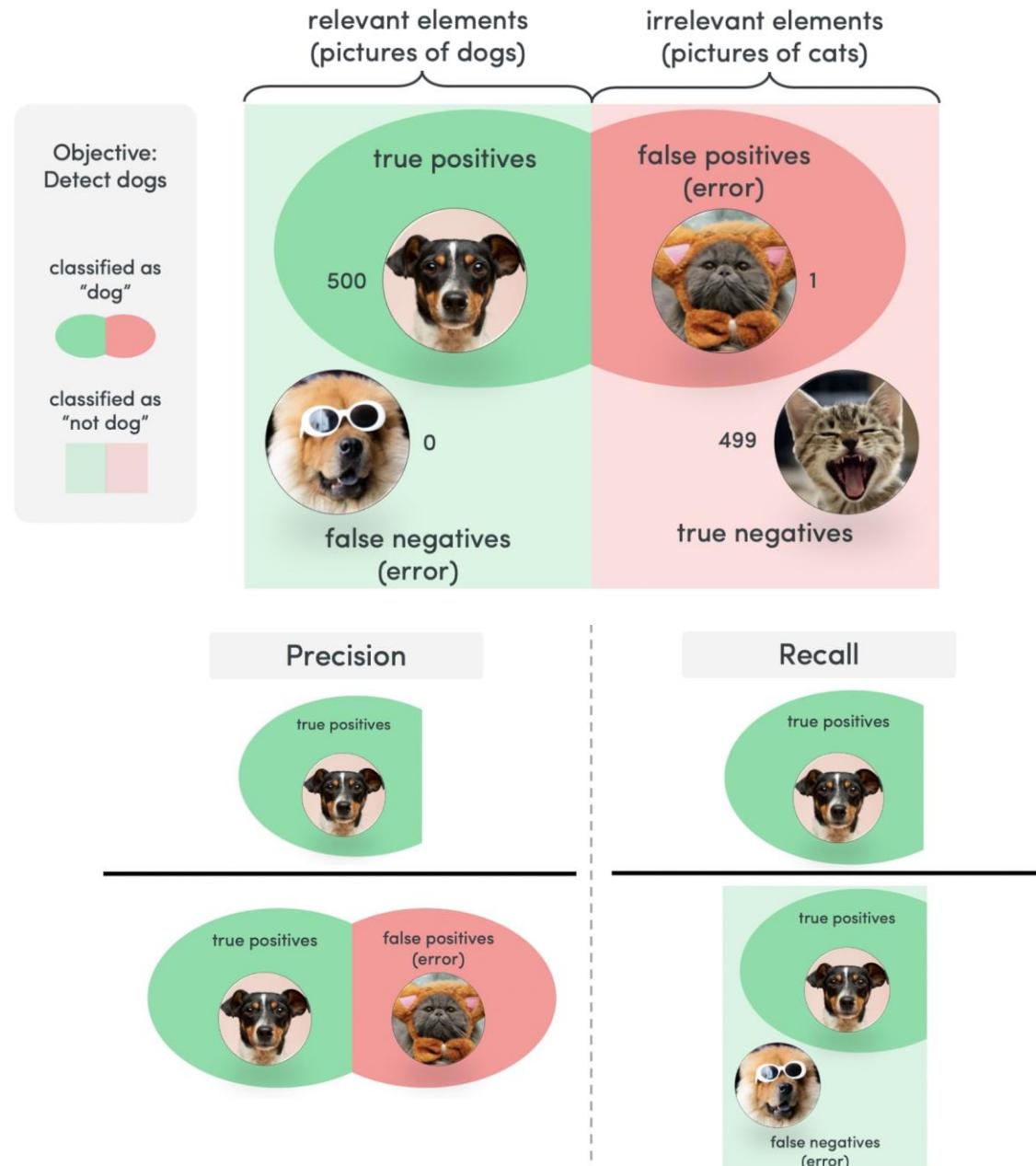
		Predicted class	
		+	-
Actual class	+	TP True Positives	FN False Negatives Type II error
	-	FP False Positives Type I error	TN True Negatives

Confusion matrix

Metric	Formula	Interpretation
Accuracy	$\frac{TP + TN}{TP + TN + FP + FN}$	Overall performance of model
Precision	$\frac{TP}{TP + FP}$	How accurate the positive predictions are
Recall Sensitivity	$\frac{TP}{TP + FN}$	Coverage of actual positive sample
Specificity	$\frac{TN}{TN + FP}$	Coverage of actual negative sample
F1 score	$\frac{2TP}{2TP + FP + FN}$	Hybrid metric useful for unbalanced classes

$$F_1 \text{ Measure} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

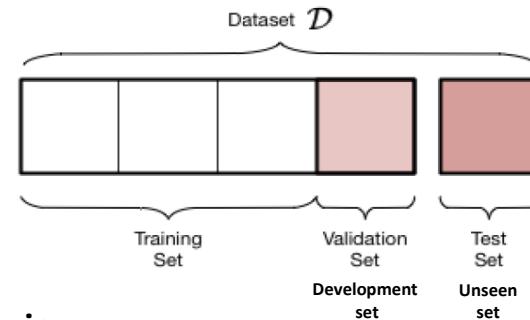
$$F_\beta \text{ Measure} = \frac{(1+\beta) \times \text{Precision} \times \text{Recall}}{(\beta \times \text{Precision}) + \text{Recall}}$$



Model diagnostics

	Underfitting	Just right	Overfitting
Symptoms	<ul style="list-style-type: none"> - High training error - Training error close to test error - High bias 	<ul style="list-style-type: none"> - Training error slightly lower than test error 	<ul style="list-style-type: none"> - Low training error - Training error much lower than test error - High variance
Regression			
Classification			
Deep learning			
Remedies	<ul style="list-style-type: none"> - Complexify model - Add more features - Train longer 	<ul style="list-style-type: none"> - Regularize - Get more data 	

- Validation set is assumed to be a representative of the testing set.
- Final model is tested on the **unseen** test set.



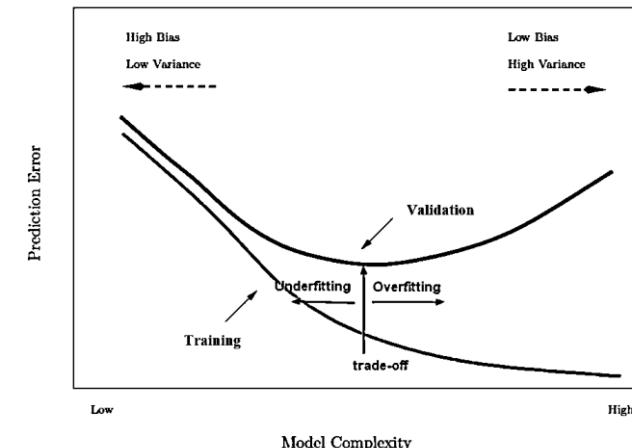
Model diagnostics terminologies

“From a training standpoint”

- **Underfitting**
- **Overfitting**

“From a validation/testing standpoint”

- **High Bias:** large change in inputs results in a small change in model outputs
- **High Variance:** small change in inputs results in a large change in model outputs



Bias/variance tradeoff – The simpler the model, the higher the bias, and the more complex the model, the higher the variance (**not necessary a tradeoff** for DNNs)

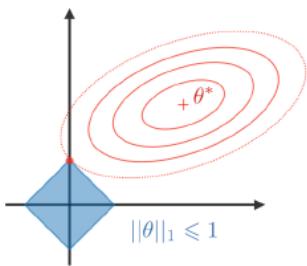
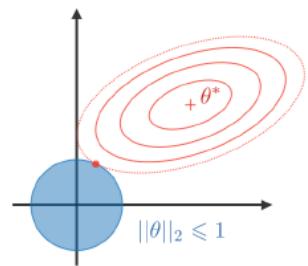
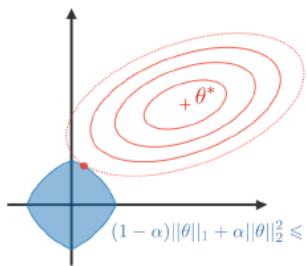
Regularization

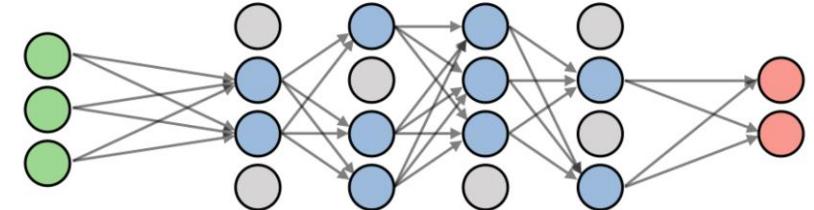
Techniques for avoiding the model to overfit data and thus dealing with high variance issues.

Examples:

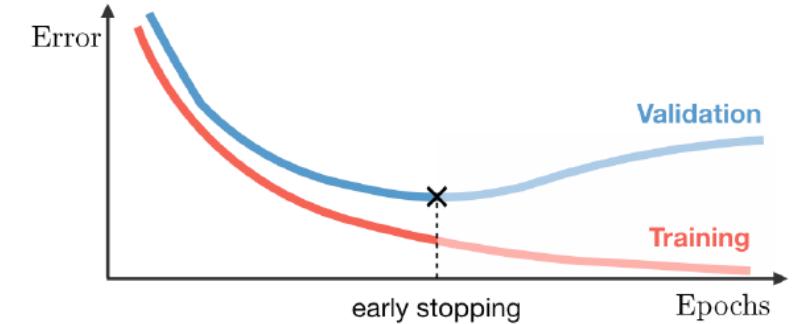
- Early stoppage in model training
- Dropout in NNs
- Parameter/weight regularization
- Data augmentation

Main weight regularization techniques

LASSO	Ridge	Elastic Net
- Shrinks coefficients to 0 - Good for variable selection	Makes coefficients smaller	Tradeoff between variable selection and small coefficients
 $ \theta _1 \leq 1$	 $ \theta _2 \leq 1$	 $(1 - \alpha) \theta _1 + \alpha \theta _2^2 \leq 1$
$\dots + \lambda \theta _1$ $\lambda \in \mathbb{R}$	$\dots + \lambda \theta _2^2$ $\lambda \in \mathbb{R}$	$\dots + \lambda \left[(1 - \alpha) \theta _1 + \alpha \theta _2^2 \right]$ $\lambda \in \mathbb{R}, \alpha \in [0,1]$



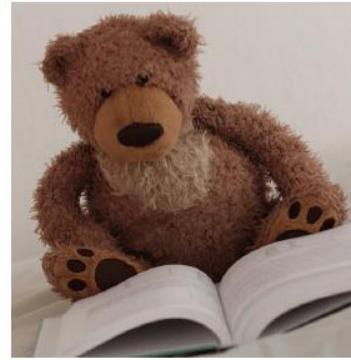
Dropout: dropping out neurons with probability $p > 0$



Early stopping – Stop the training process as soon as the validation loss reaches a plateau or starts to increase.

Data augmentation

- Deep learning models need a lot of data to be trained.
- Get more data from the existing ones using data augmentation techniques.

Original	Flip	Rotation	Random crop
			
Color shift	Noise addition	Information loss	Contrast change
			