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

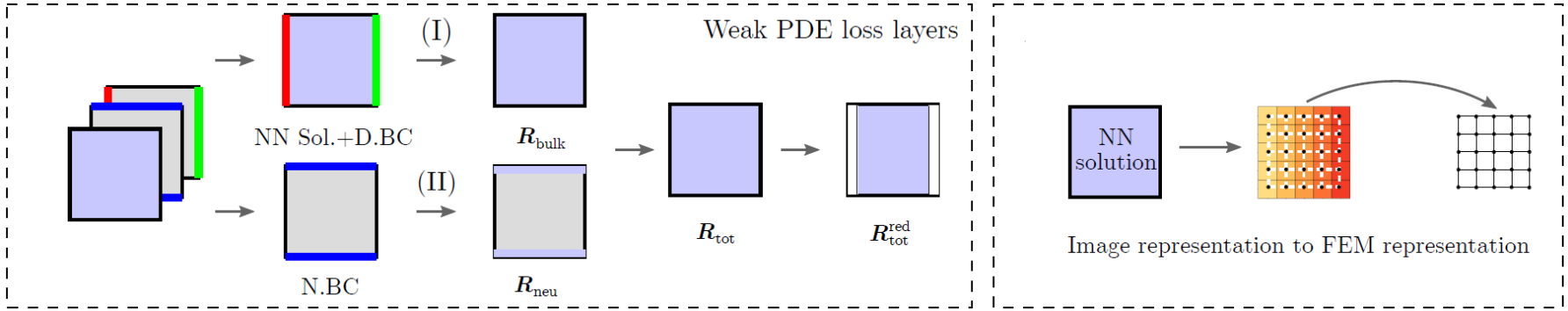
2023

Lectures Overview and timeline

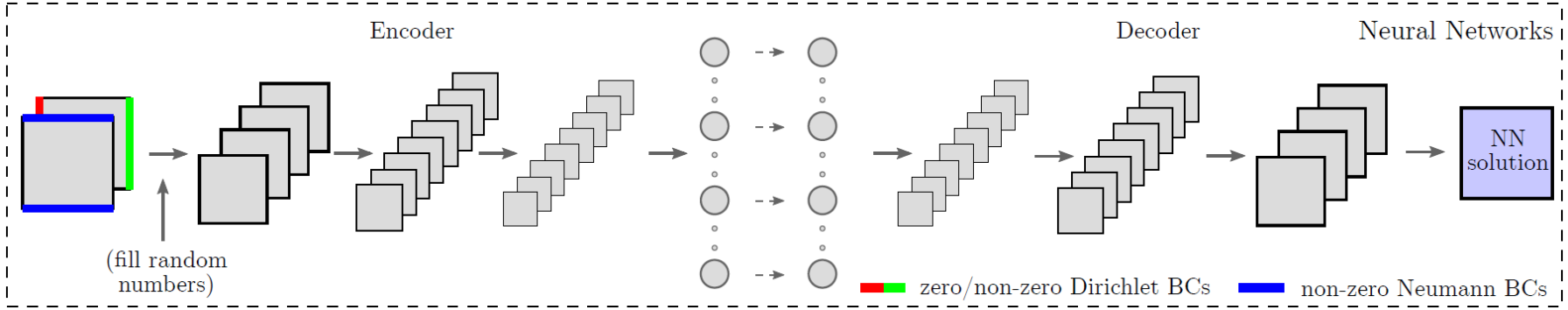
- Lecture 1 – Essentials of Neural Networks and Machine Learning.
- Lecture 2 – Google Colab Usage, Convolutional Neural Networks Introduction, ML-PDE Project Overview.
- Lecture 3 – TensorFlow in Colab: Implementing NNs & CNNs for MNIST
- Lecture 4 – Advanced Machine Learning & Deep Learning Topics; 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 neutral network with the FEM residual (part 1) as a custom loss function.



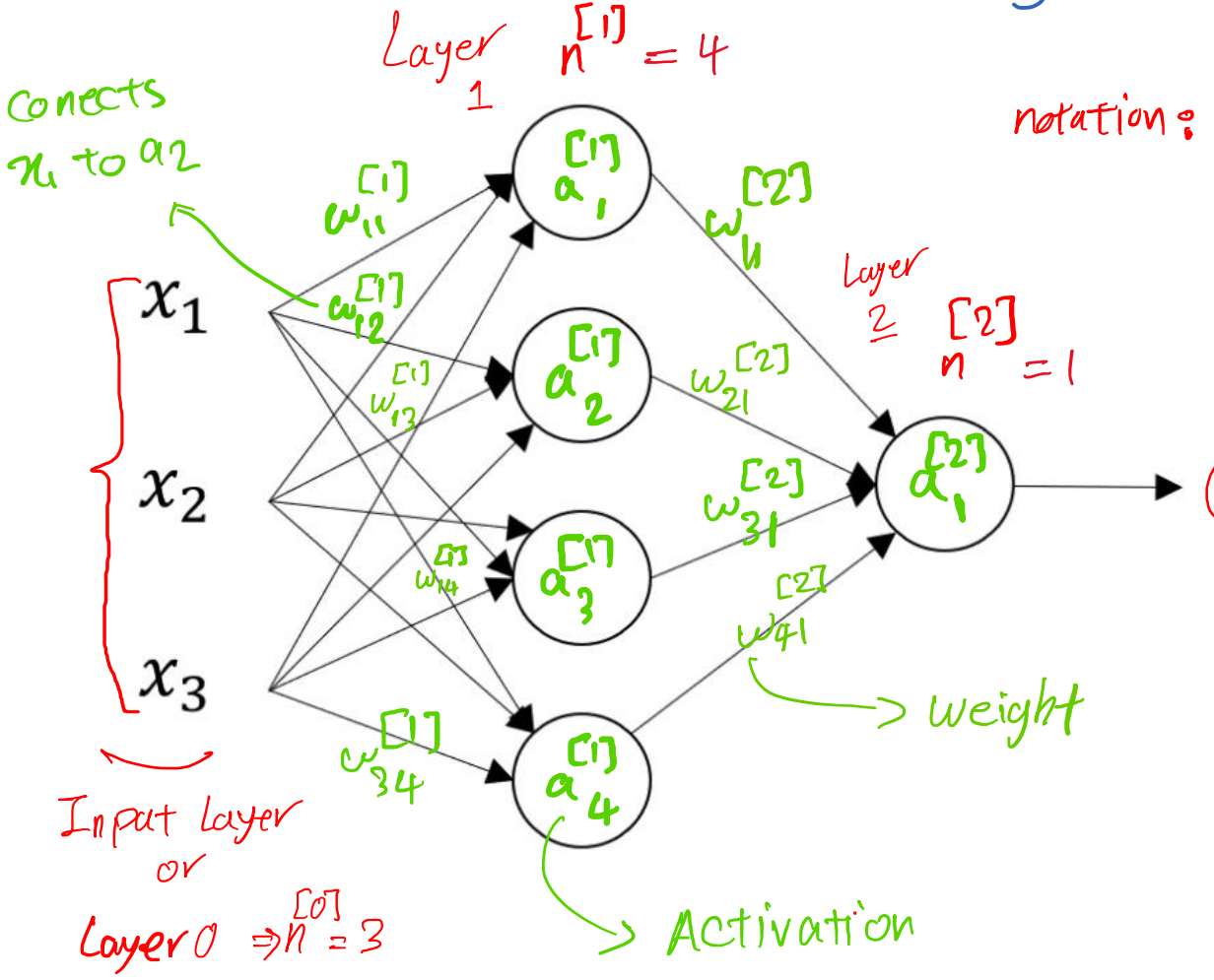
Tools: Python, NumPy, TensorFlow, Google Colab

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

Neural Network Representation

How to Represent the architecture of NNs using labels?

2-Layer NN



conects x_1 to a_2

notation: $n^{[l]}$ → superscripts in [] is the layer numb

output

(here we only have 1 output, we can have more outputs)

weight

Input Layer or Layer 0 $\Rightarrow n^{[0]} = 3$

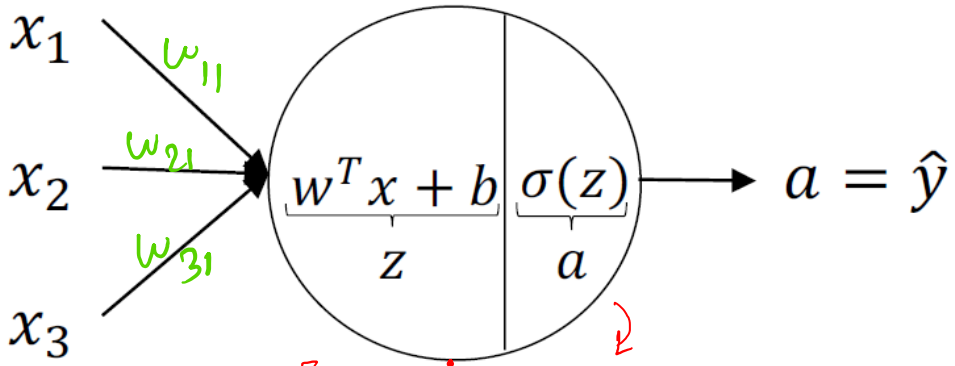
Activation

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

Neural Network Representation

(Equations)

Simplest case: 1-Layer NN



Every node/neuron has two equations

Linear $z = w^T x + b$ (weights, bias)

$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}_{3\times 1}$

$w^T = [w_{11}\ w_{21}\ w_{31}]_{1\times 3}$

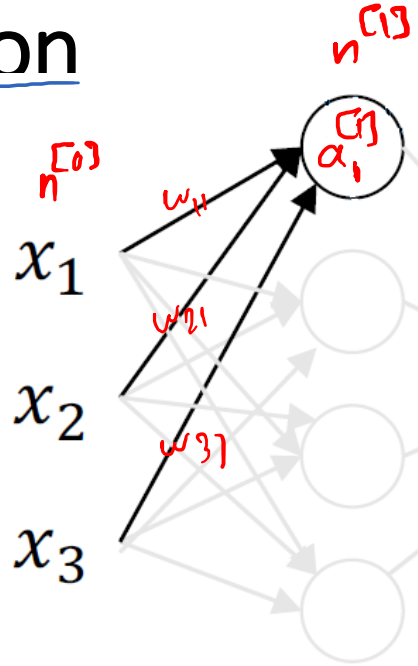
$b_{1\times 1}$

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

$(1\times 3)(3\times 1)$

non linear $a = \sigma(z)$

activation ← activation function (non linear)



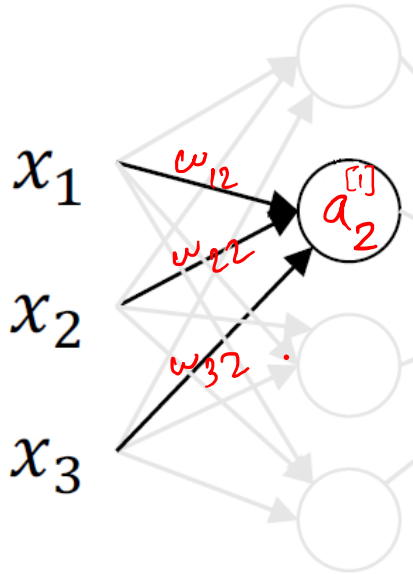
Important: z: output of linear relation

a: output of activation fun.

$a_1^{[1]} = \sigma(z_1^{[1]})$

$z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]}$

$[w_{11}\ w_{21}\ w_{31}]$ ← $a^{[0]}$



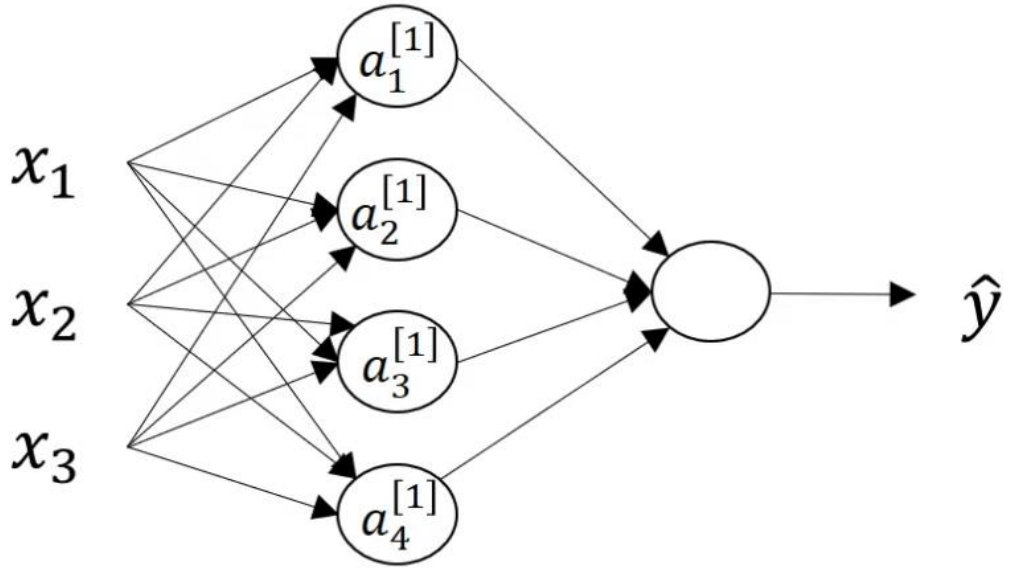
$a_2^{[1]} = \sigma(z_2^{[1]})$

$z_2^{[1]} = w_2^{[1]T} a^{[0]} + b_2^{[1]}$

$[w_{12}\ w_{22}\ w_{32}]$

Neural Network Representation: Vectorizing across W and b

efficiency in python and numpy, TF



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

all nodes in the first layer

linear — non linear

vectorizing

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

$$b^{[1]} = \begin{bmatrix} b_1^{[1]} \\ \vdots \\ b_4^{[1]} \end{bmatrix}_{4\times 1}, \quad z^{[1]} = \begin{bmatrix} z_1^{[1]} \\ \vdots \\ z_4^{[1]} \end{bmatrix}_{4\times 1}$$

$4 = n^{[1]}$:

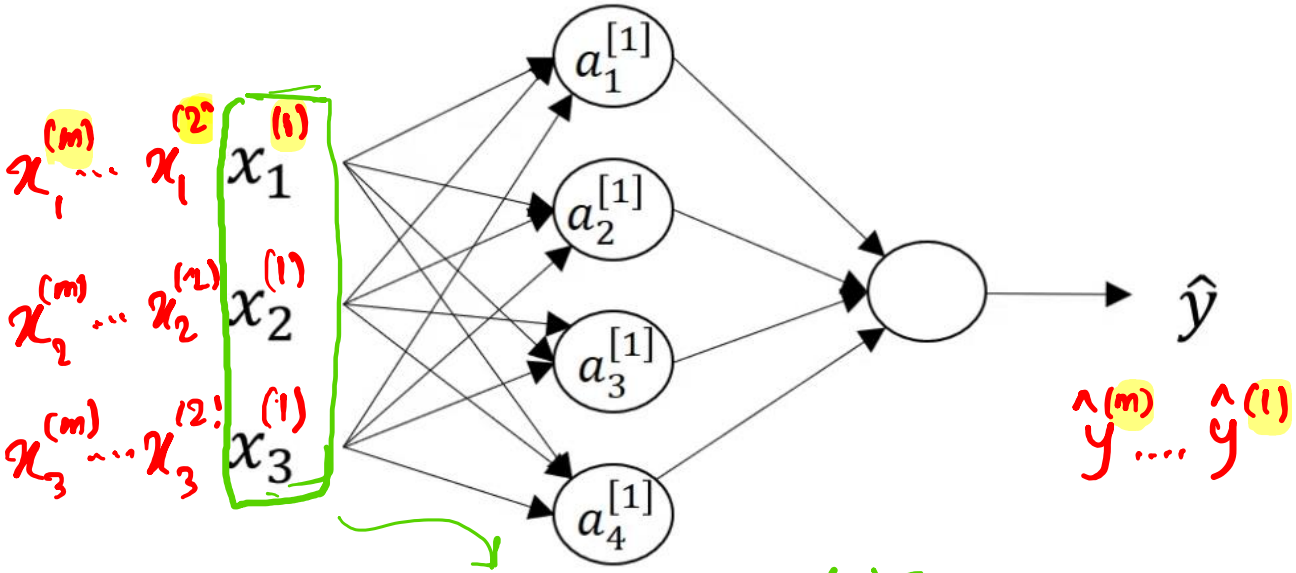
$$z^{[1]} = W^{[1]} x + b^{[1]}$$
$$a^{[1]} = \sigma(z^{[1]})$$

No subscripts

$1 = n^{[2]}$:

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

Neural Network Representation: Vectorizing across examples x



$$X_{3\times m} = \begin{bmatrix} | & & | \\ x^{(1)} & \cdots & x^{(m)} \\ | & & | \end{bmatrix}_{(3\times m)}$$

$$Z_{4\times m} = \begin{bmatrix} | & & | \\ z^{(1)} & \cdots & z^{(m)} \\ | & & | \end{bmatrix}_{4\times m} \xrightarrow{\sigma} A_{4\times m}$$

evaluate for every entry

```
for i = 1 to m:
```

$$z^{[1](i)}_{4\times 1} = W^{[1]}_{4\times 3} x^{(i)} + b^{[1]}_{4\times 1}$$
$$a^{[1](i)}_{4\times 1} = \sigma(z^{[1](i)})$$
$$z^{[2](i)}_{1\times 1} = W^{[2]}_{1\times 4} a^{[1](i)}_{4\times 1} + b^{[2]}_{1\times 1}$$
$$a^{[2](i)}_{1\times 1} = \sigma(z^{[2](i)})$$

vectorizing

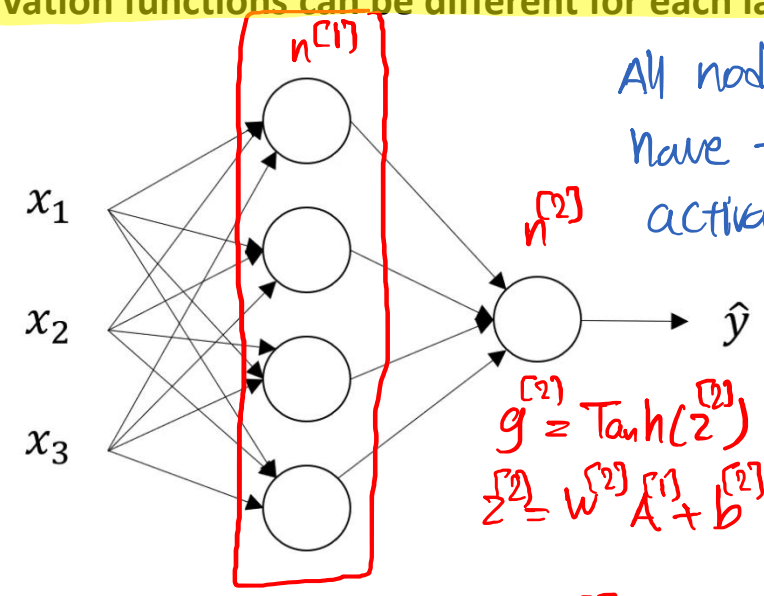
$$Z^{[1]}_{4\times m} = W^{[1]}_{4\times 3} X_{3\times m} + b^{[1]}_{4\times 1}$$
$$A^{[1]} = \sigma(Z^{[1]})$$
$$Z^{[2]} = W^{[2]} A^{[1]} + b^{[2]}$$
$$A^{[2]} = \sigma(Z^{[2]})$$

broadcast 4×1 to $4\times m$ by just repeating $\begin{bmatrix} \\ \\ \\ \\ \end{bmatrix}_{4\times 1} \rightarrow \begin{bmatrix} \cdots \\ \cdots \\ \cdots \\ \cdots \end{bmatrix}_{4\times m}$

Uppercase Capital ↙ X, Z, A

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

$g^{[2]} = \tanh(z^{[2]})$

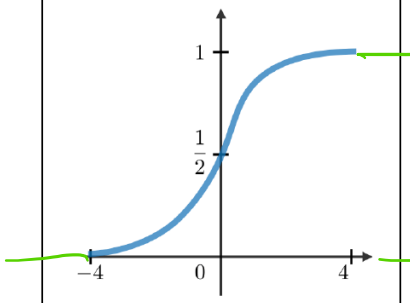
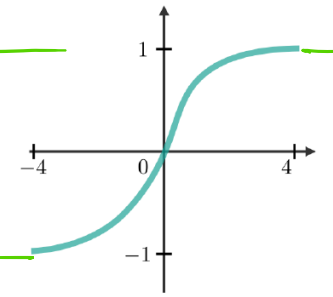
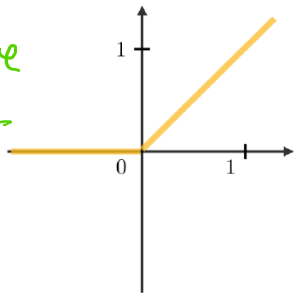
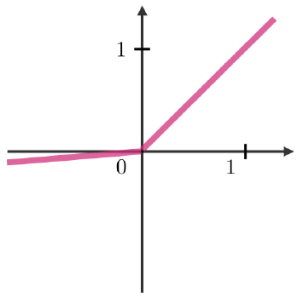
$z^{[2]} = w^{[2]} a^{[1]} + b^{[2]}$

$g^{[1]} = \text{sigmoid}(z^{[1]})$

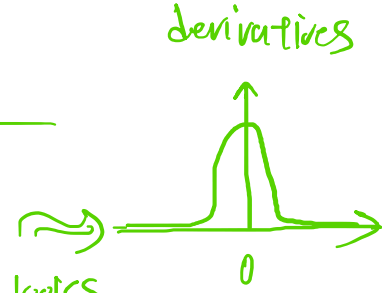
$z^{[1]} = w^{[1]} a^{[0]} + b^{[1]}$

$a^{[0]} = x$

Activation functions introduce non-linear complexities to the model.

Sigmoid	Tanh
$g(z) = \frac{1}{1+e^{-z}}$	$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$
	
ReLU	**Leaky ReLU**
$g(z) = \max(0,z)$	$g(z) = \max(\epsilon z, z)$ with $\epsilon \ll 1$
	

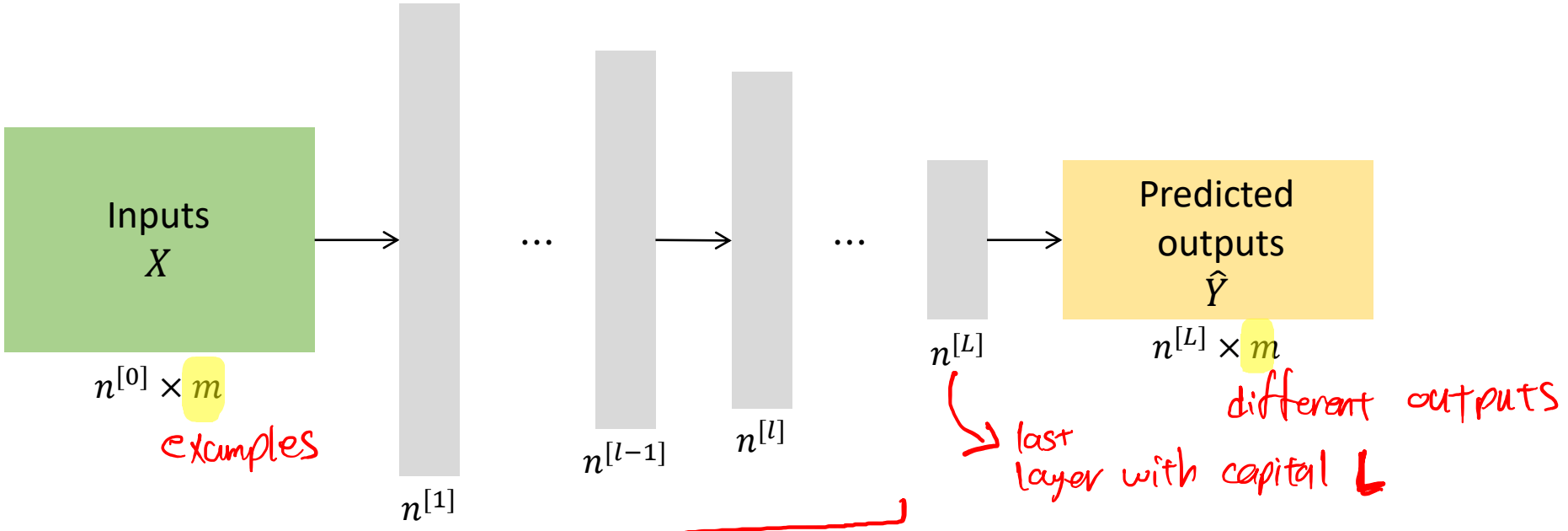
Take derivatives in back prop.



$\begin{cases} 0 & \text{no derivative} \\ 1 & 0 < x \end{cases}$

From Super VIP Cheatsheet: Deep Learning, Afshine Amidi and Shervine Amidi, 2018

Forward propagation



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

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

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

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

Sizes:

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

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

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

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

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

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

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

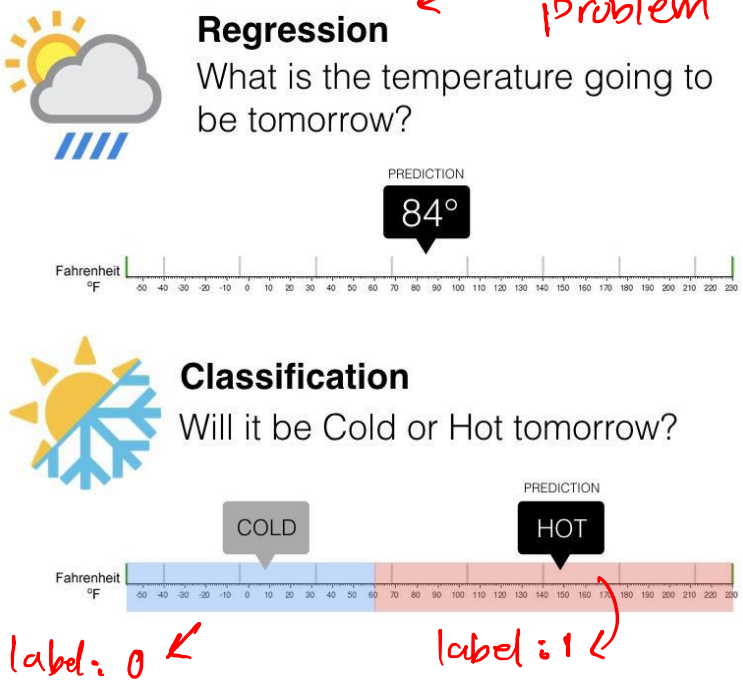
$W^{[L]}$: $n^{[L]} \times n^{[L-1]}$

$b^{[L]}$: $n^{[L]}$

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

Loss functions

notation: use hat ^ for NNs predictions $\hat{y}$



Continuous problem

- **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

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: temprature $\hat{y}$

Cross Entropy Loss

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

classes

Prediction

Ground truth {0,1}

Classification: Hot or Cold → 1, 0

Classification using softmax activation

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

3 1 2 0 3 2 0 1

Ground truth y: one vs zero

koalas	cats	dogs	chicks
$\begin{bmatrix}1\\0\\0\\0\end{bmatrix}$	$\begin{bmatrix}0\\1\\0\\0\end{bmatrix}$	$\begin{bmatrix}0\\0\\1\\0\end{bmatrix}$	$\begin{bmatrix}0\\0\\0\\1\end{bmatrix}$

0 1 2

How to label 4 classes

This transformation is called one vs zero one hot encoding

Prediction $\hat{y}$: Softmax activation

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

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

- 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 ⇒ Koala ✓

$a+b+c+d=1$

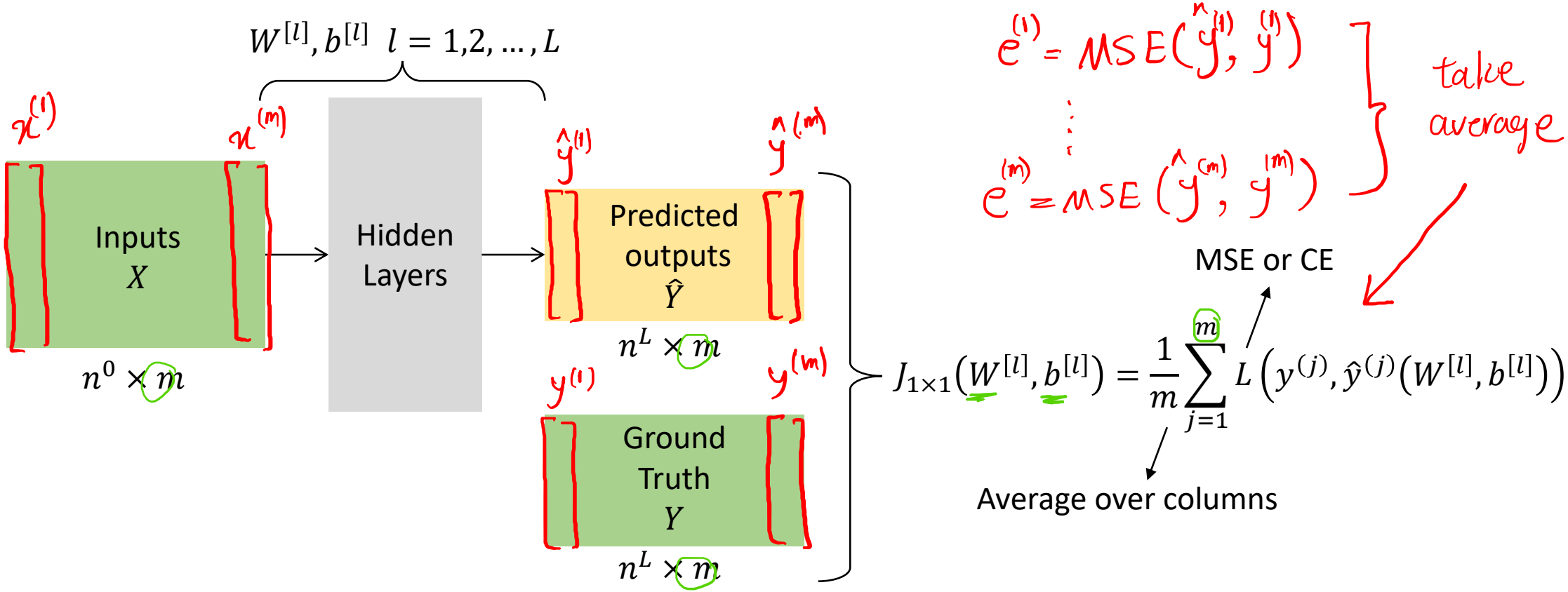
Cross Entropy Loss

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

classes, Prediction, Ground truth

$-\log a > 0$ (minimized when $a \to 1$)

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 $\boldsymbol{\theta}$ that approximately minimizes $J(\boldsymbol{\theta})$

Design variables (w, b)

Set

given by the user

- learning rate α
- Max. number of iterations I_{max}
- Stoppage criteria
- Initial guess/point $\boldsymbol{\theta}_0$

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

  if stoppage criteria on $\boldsymbol{\theta}_i$ met:

    return $\boldsymbol{\theta}_i$

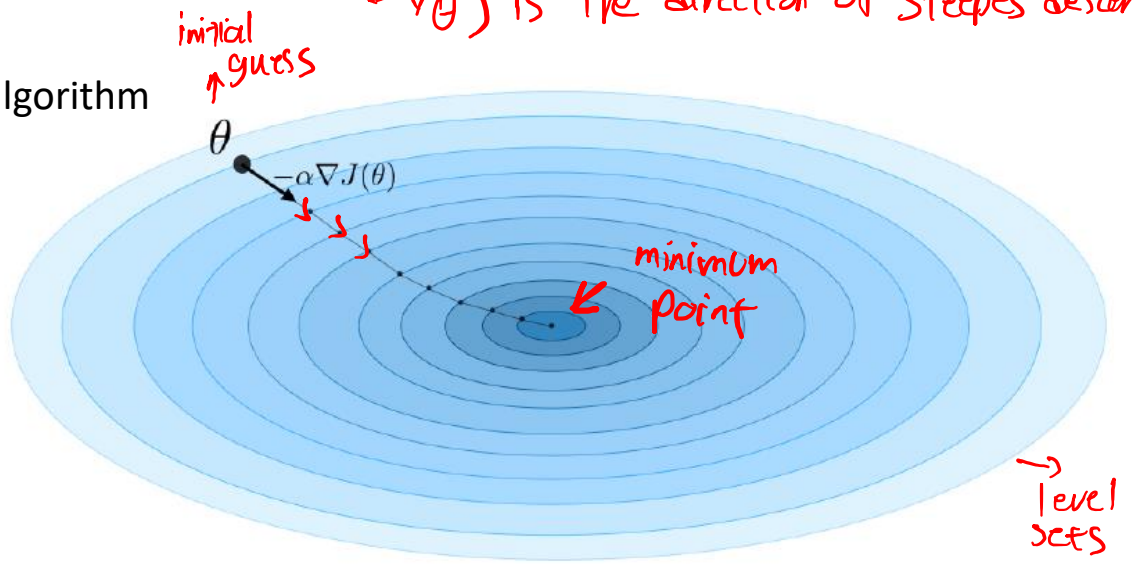
  else:

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \alpha \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_i)$$

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

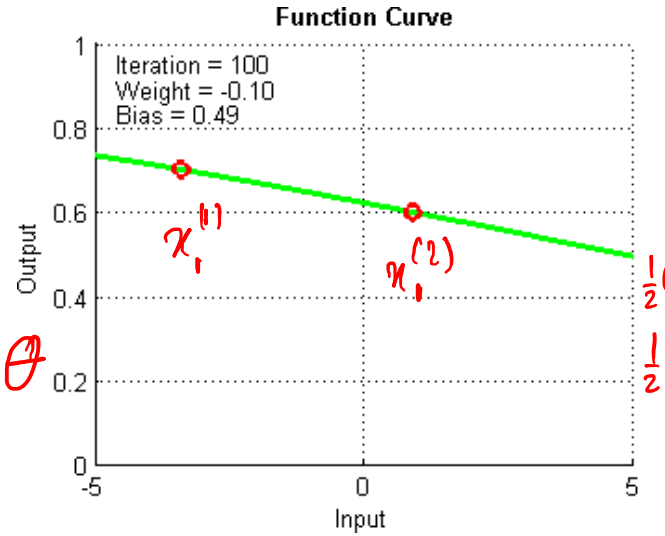
important part is finding $\nabla_\theta J$

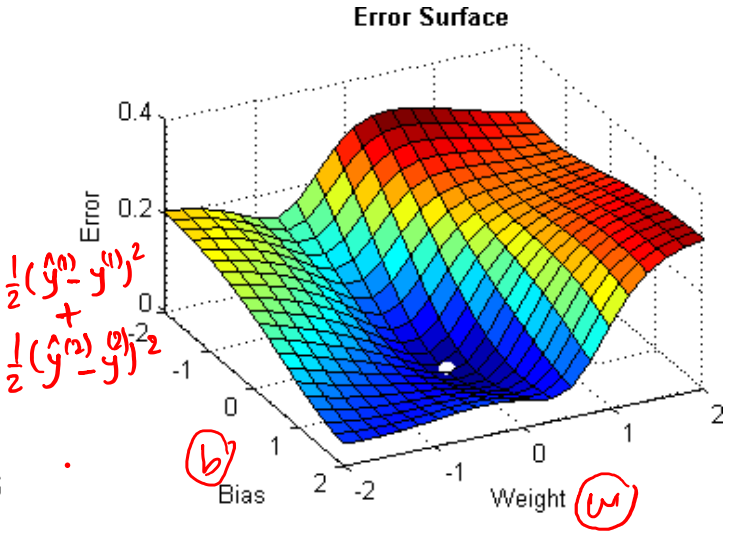
$-\nabla_\theta J$ is the direction of steepest descent



Level sets of J near a local minimum

$J(\theta)$ = Constant





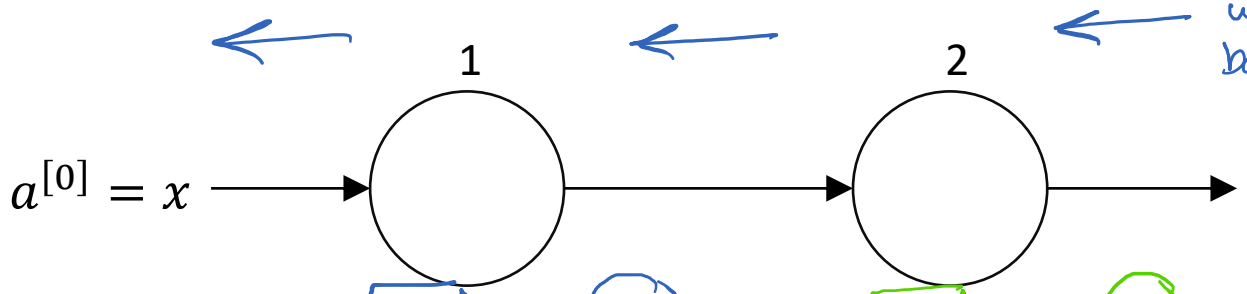
$\frac{1}{2}(\hat{y}^{(1)} - y^{(1)})^2 + \frac{1}{2}(\hat{y}^{(2)} - y^{(2)})^2$

$y = wx + b$

Image from From Super VIP Cheatsheet: Deep Learning, Afshine Amidi and Shervine Amidi, 2018

Gif from https://towardsdatascience.com/coding-deep-learning-for-beginners-linear-regression-gradient-descent-fcd5e0fc077d

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



we do it backward

explicitly: $J(a^{[2]}, y)$

implicitly: $J(w^{[1]}, b^{[1]}, w^{[2]}, b^{[2]})$

$$J = \frac{1}{m}\sum_{j=1}^{m}\left(\hat{y}^{(j)} - y(w^{[1]}, b^{[1]}, w^{[2]}, b^{[2]})\right)^2$$

$$\nabla_w J = \begin{bmatrix} \frac{\partial J}{\partial w^{[1]}} \\ \frac{\partial J}{\partial w^{[2]}} \end{bmatrix}, \quad \nabla_b J = \begin{bmatrix} \frac{\partial J}{\partial b^{[1]}} \\ \frac{\partial J}{\partial b^{[2]}} \end{bmatrix}$$

$$\frac{\partial J}{\partial w^{[2]}} = \underbrace{\frac{\partial J}{\partial a^{[2]}} \frac{\partial a^{[2]}}{\partial z^{[2]}}}_{\delta^{[2]}} \frac{\partial z^{[2]}}{\partial w^{[2]}} = \frac{\partial J}{\partial a^{[2]}} g'^{[2]}(z^{[2]}) a^{[1]} = \delta^{[2]} a^{[1]}$$

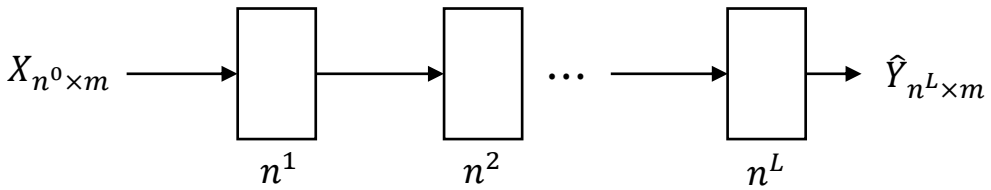
$$\frac{\partial J}{\partial b^{[2]}} = \delta^{[2]} \frac{\partial z^{[2]}}{\partial b^{[2]}} \rightarrow \text{equal } 1$$

$$\frac{\partial J}{\partial w^{[1]}} = \underbrace{\frac{\partial J}{\partial a^{[2]}} \frac{\partial a^{[2]}}{\partial z^{[2]}} \frac{\partial z^{[2]}}{\partial a^{[1]}} \frac{\partial a^{[1]}}{\partial z^{[1]}}}_{\delta^{[1]}} \frac{\partial z^{[1]}}{\partial w^{[1]}} = \delta^{[1]} a^{[0]}$$

$$\delta^{[1]} = \delta^{[2]} w^{[2]} g'^{[1]}(z^{[1]})$$

$$\frac{\partial J}{\partial b^{[1]}} = \delta^{[1]} = (w^{[2]} \delta^{[2]}) g'^{[1]}(z^{[1]})$$

Vectorized Forward propagation and Backpropagation



Cost function:

$$J_{1\times1}(W^1, b^1, \dots, W^L, b^L) = \frac{1}{m}\sum_{j=1}^{m} L^{(j)}$$

$$L^{(j)}_{1\times1} = L\left(\hat{y}^{(j)}_{n^L\times1}, y^{(j)}_{n^L\times1}\right)$$

$$\hat{Y} = g^L(W^L A^{L-1} + b^L)$$

Forward propagation:

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

$$Z^l_{n^l\times m} = W^l_{n^l\times n^{l-1}} A^{l-1}_{n^{l-1}\times m} + b^L_{n^l\times1}[1,\cdots,1]_{1\times m}$$

$$\mathrm{A}^l_{n^l\times m} = g^l(Z^l_{n^l\times m})$$

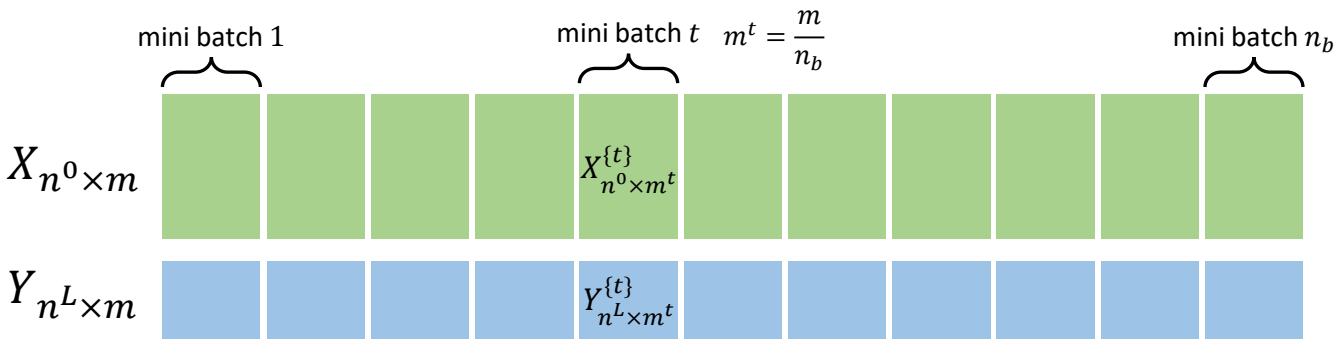
Backpropagation:

For $l = L,\dots,2,1$: (Hadamard product: $\circ$)

$$\Delta^l_{n^l\times m} = \begin{cases} \nabla_{a^l} J(A^l) \circ g'^l(Z^l), & l = L \\ \left(W^{l+1^{\mathrm{T}}}\Delta^{l+1}\right) \circ g'^l(Z^l), & 1 \le l < L \end{cases}$$

$$\frac{\partial J}{\partial W^l} = \Delta^l A^{l-1^{\mathrm{T}}}, \qquad \frac{\partial J}{\partial b^l} = \Delta^l \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}_{m\times1}$$

Mini-batch gradient descent



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

1 epoch
1 pass through entire training set

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

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

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

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

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

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

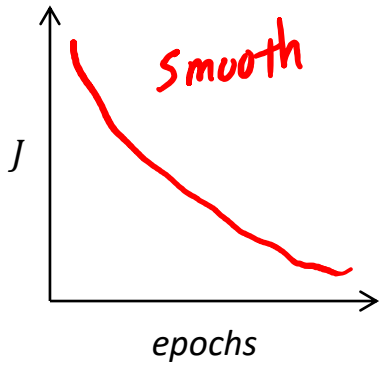
Backprop to compute gradients $\nabla_{W^{[l]}} J^{\{t\}}, \nabla_{b^{[l]}} J^{\{t\}}$

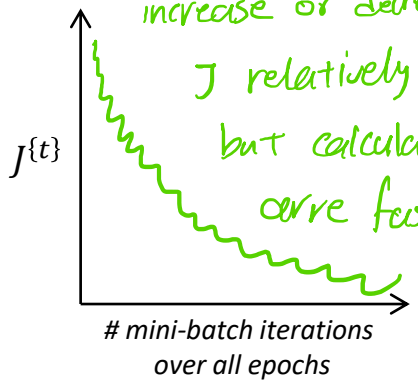
SG update:

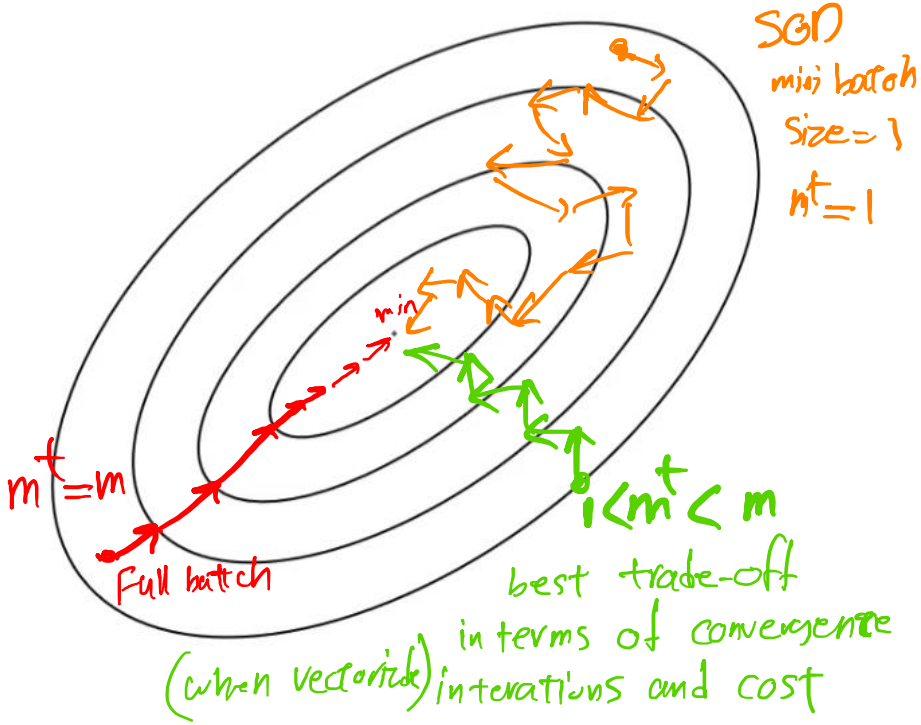
$$W^{[l]} = W^{[l]} - \alpha \nabla_{W^{[l]}} J^{\{t\}}$$

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

not the exact gradients for entire data just for batch {t}







What are hyperparameters?

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

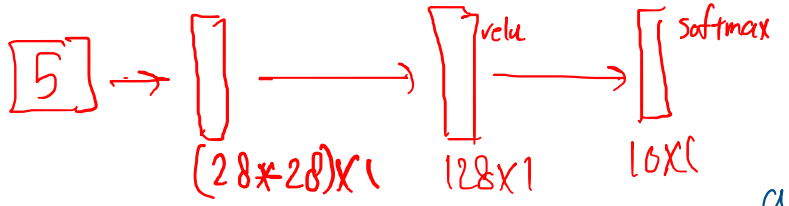
Hyperparameters:
Parameters that control the final values of parameters (W, b) obtained in backpropagation

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)
   ])

   model.compile(optimizer=tf.train.AdamOptimizer(),
                 loss='sparse_categorical_crossentropy',
                 metrics=['accuracy'])

4  # train model
   model.fit(train_images, train_labels, epochs=5)

5  # evaluate
   test_loss, test_acc = model.evaluate(test_images, test_labels)
   print('test accuracy:', test_acc)

6  # make predictions
   predictions = model.predict(test_images)
```

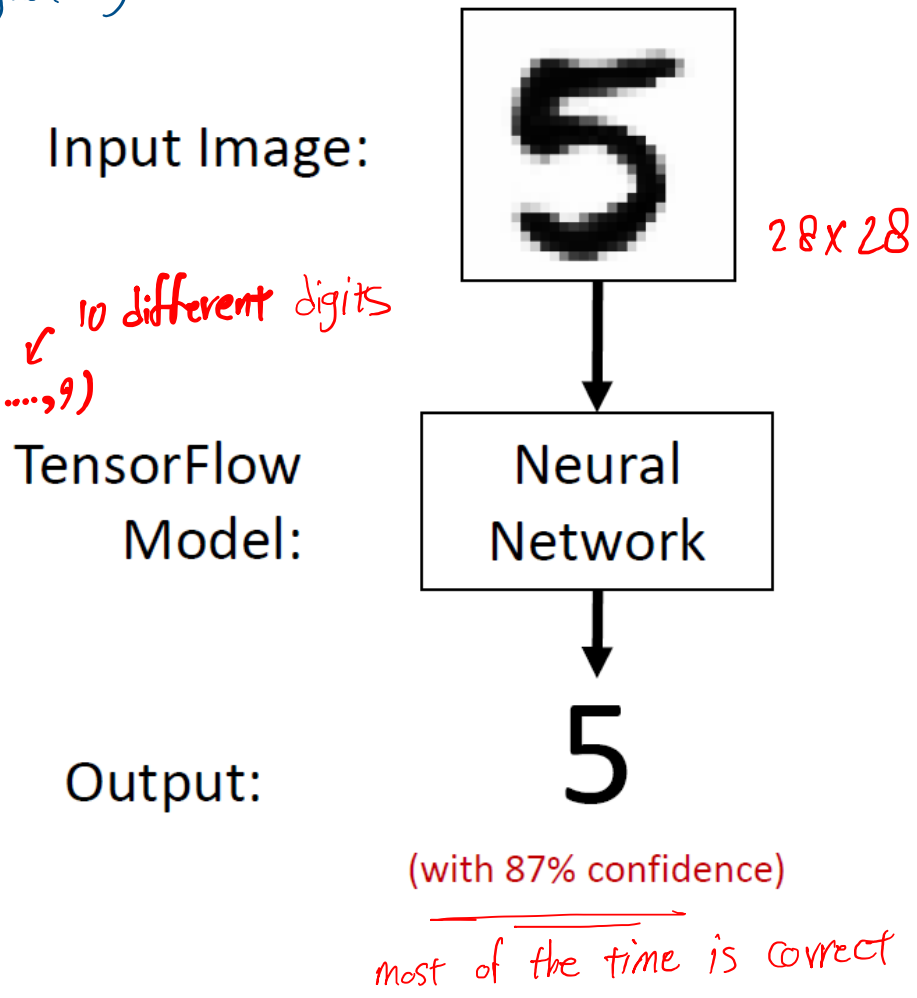
[1] n = 128

[2] n = 10 → (0, 1, ..., 9)

10 different digits

CE

using back prop.

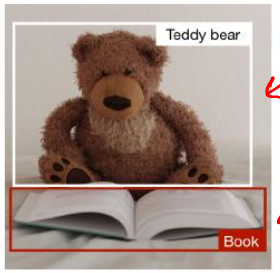


Based on 2019 slides by Lex Fridman for MIT Deep Learning and Artificial Intelligence Lectures

Convolutional Neural Networks (CNNs): Computer Vision Problems

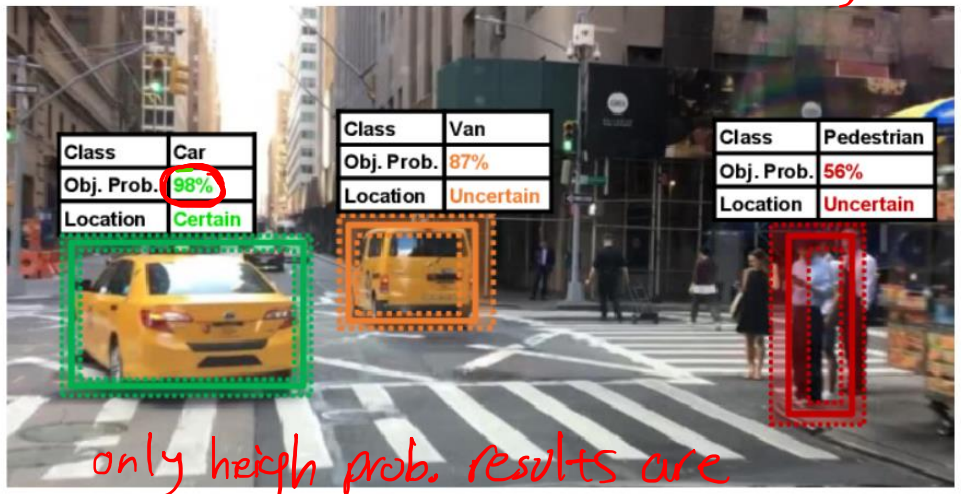
few applications of CNN-based models

Object detection

Image classification	Classification w. localization	Detection
	 finds its location	 ① ②
- 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

object detection for self-driving cars



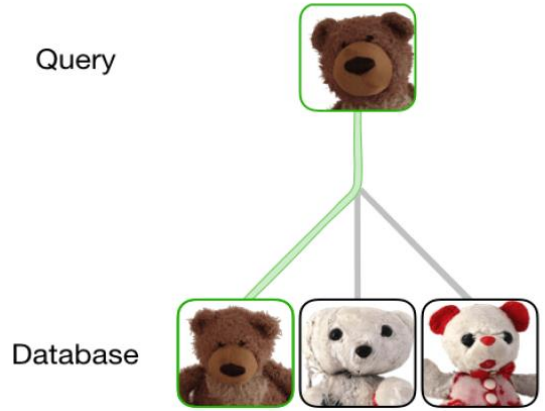
only heigh prob. results are reliable

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*

Content C Style S Generated image G

Another image as style

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



Content from Super VIP Cheatsheet: Deep Learning, Afshine Amidi and Shervine Amidi, 2018
Based on slides by DeepLearning.AI and Andrew Ng

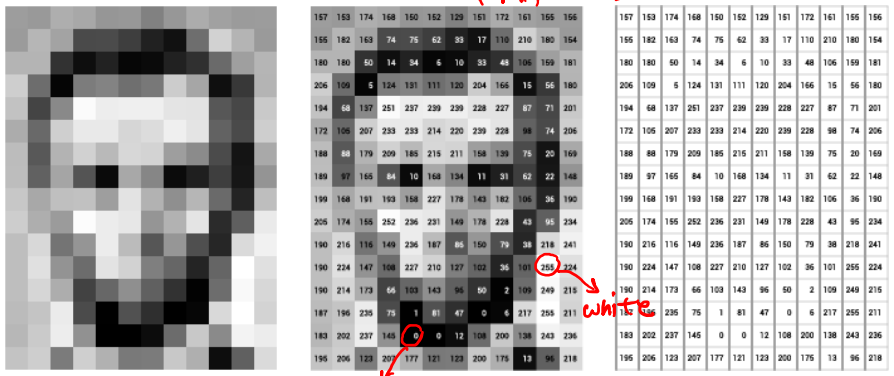
Image representation as matrices

use matrices as inputs for CNNs

associate a number to each pixel relative to their darkness/lightness

Grayscale image

(Darkest) black = 0 ... 255 = (Lightest) white



⇒ [ ] matrix

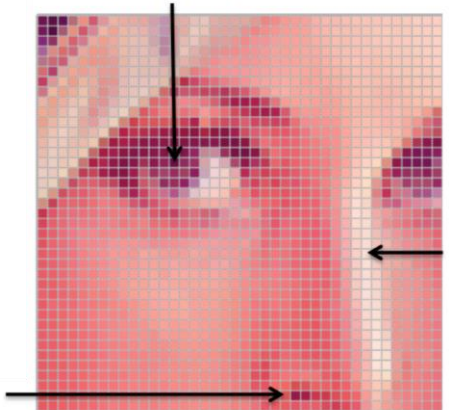
3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

→ can be illustrated as an image

$(n \times m \times 1)$

Color image

RGB (no green) [90, 0, 53]



R G B [213, 60, 67]

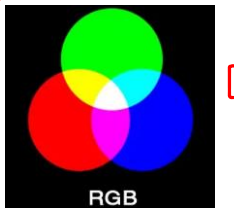
R G B [249, 215, 203]

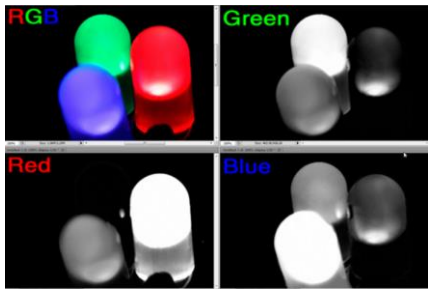
white: [255, 255, 255]

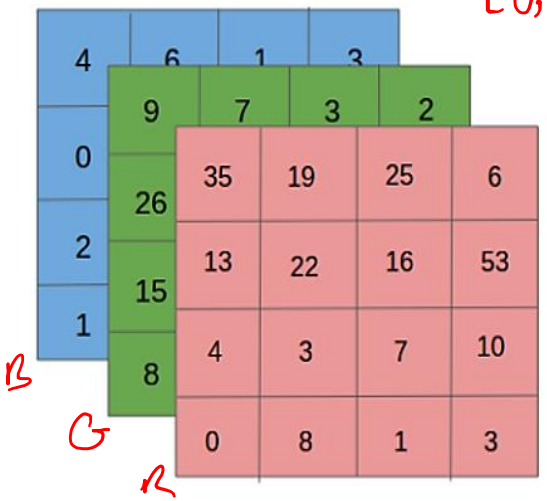
Black: [0, 0, 0]

3 channels

mixing gives different colors [R G B]

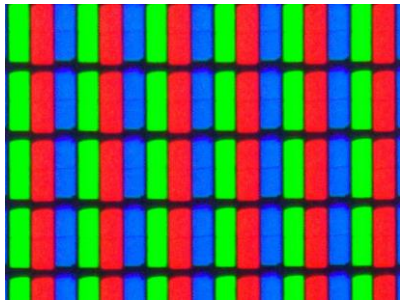






$(n \times m \times 3)$

each channel separately is a grayscale image



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?

is defined differently in math.

$1\times3+1\times1+2\times1 + 0\times0+0\times5+0\times7 + -1\times1+-1\times8+-1\times2 = -5$

Input array

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

symbol for convolution

*

Filter or Kernel

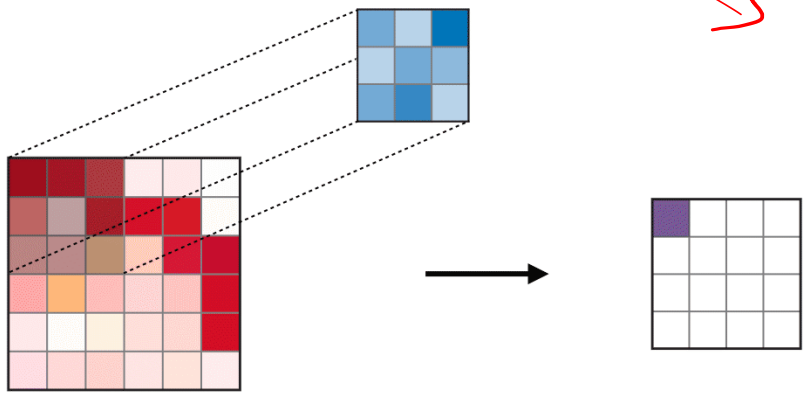
1	0	-1
1	0	-1
1	0	-1

=

output

-5			
	-2	2	3
0	-2	-4	-7
-3	-2	-3	-16

vertical **edge** detector



Based on slides by DeepLearning.AI and Andrew Ng
Content from Super VIP Cheatsheet: Deep Learning, Afshine Amidi and Shervine Amidi, 2018

Convolution for edge detection

vertical edges

Horizontal edges

Vertical edge detection

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

*

filter

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

from light to dark

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

=

edge

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

*

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

negative values indicates going from dark to light

Convolution for edge detection

90° rotation → $\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$ ⇒ Horizontal edge filter

Other edge detection filters:

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

Vertical Sobel filter

larger weight for the center pixels

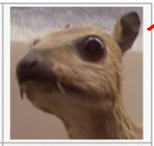
3	0	-3
10	0	-10
3	0	-3

Vertical Scharr filter

Vertical and Horizontal edges.

Based on slides by DeepLearning.AI and Andrew Ng
Sobel operator Wikipedia

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)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

It is NOT identity matrix

original image

take average of 9 pixels

Gaussian blur

3x3 filter

10 x10 filter

Gaussian function

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

↳ discretize over a grid to get filters

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

$w^{[l]}$

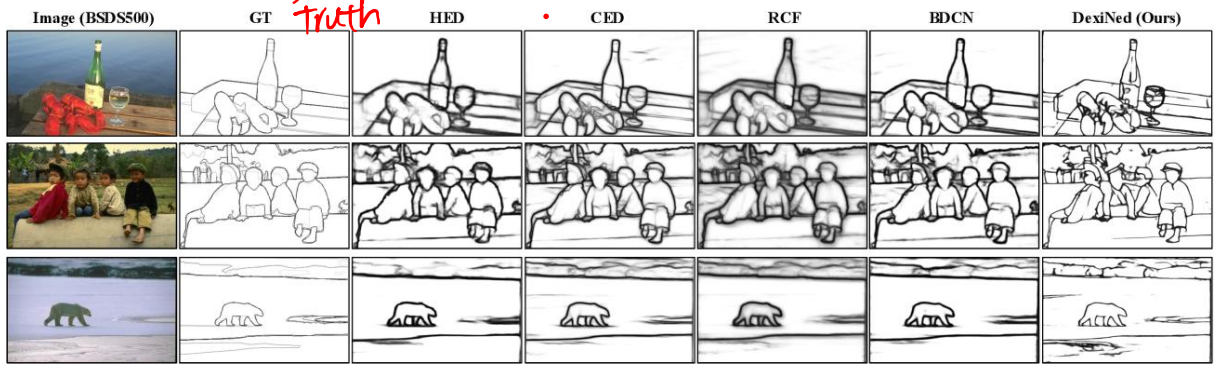
$Z^{[l]} = W^{[l]} * A^{[l-1]} + b^{[l]}$

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

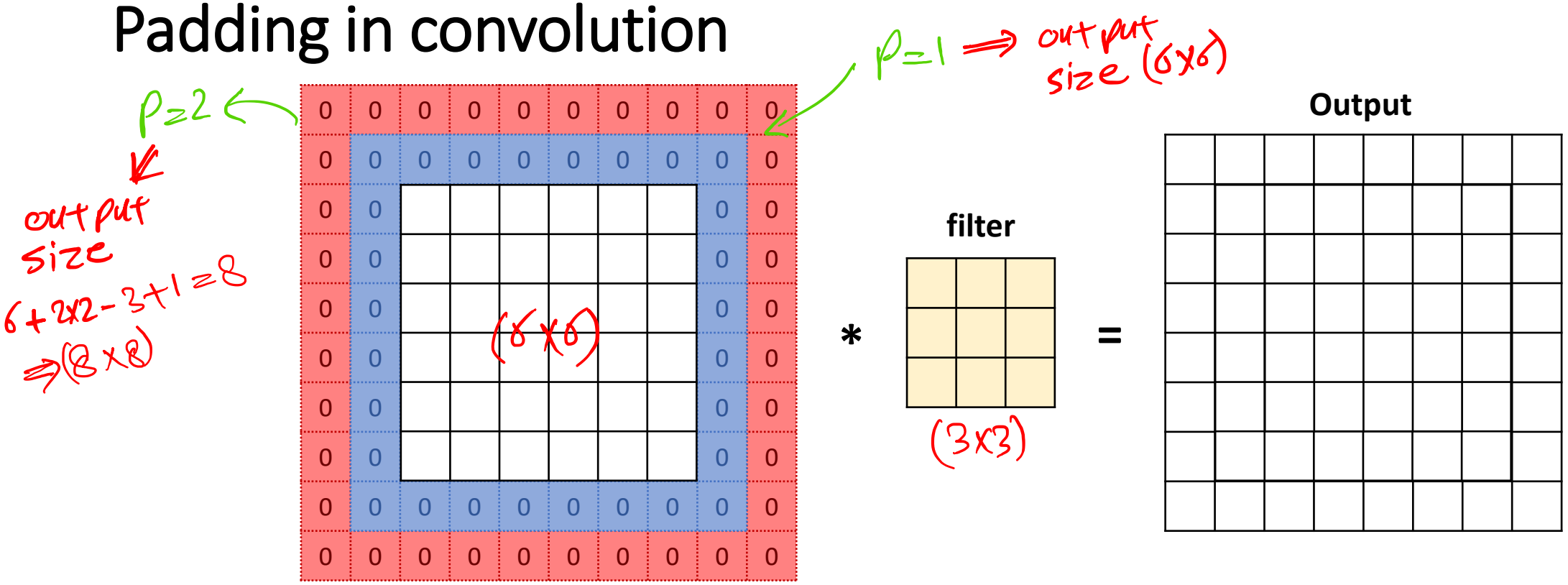
filter
9 parameters



Edge detection using deep CNNS
https://doi.org/10.1109/WACV45572.2020.9093290

Based on slides by DeepLearning.AI and Andrew Ng
Wikipedia Gaussian blur

Padding in convolution



Output size given input size of $n \times n$ and filter size f (f x f) filter

output size:

$(n-f+1) \times (n-f+1)$

$6-3+1=4 \leftarrow$ output

Without padding

- Output size shrinks
- loosing 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:

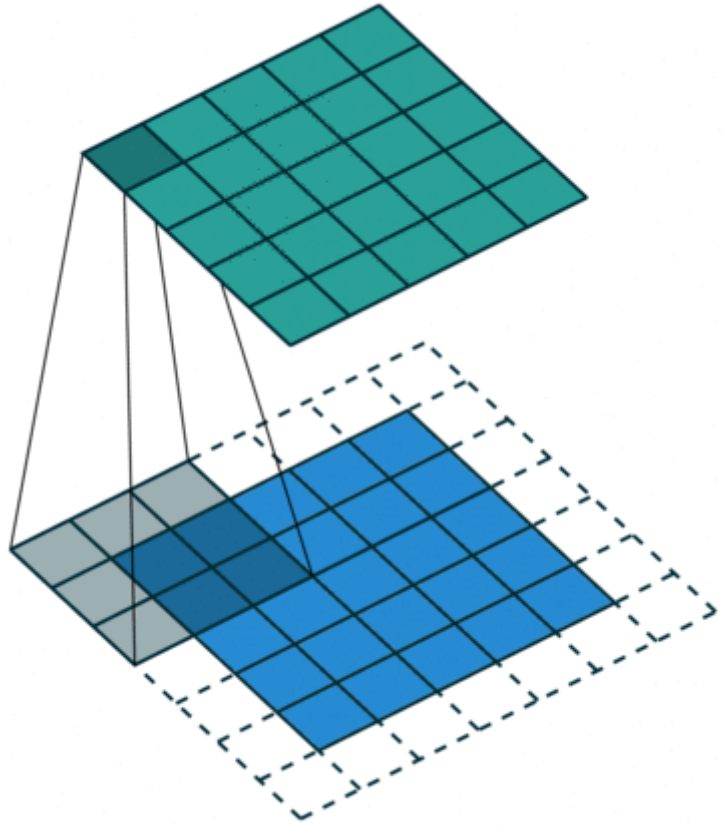
$(n+2p-f+1) \times (n+2p-f+1)$

p=1

$6+2x1-3+1=6$

⇒ 6x6 output

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 $= n \times n$

size: n

output size: $n + 2p - f + 1 = n \Rightarrow \boxed{p = \frac{f-1}{2}}$

f is an odd number $\Downarrow$ p is integer

Strided convolution

S=2

don't calculate
⇒ using ⌊ ⌋

(7x7)

*

(3x3)

3	4	5
1	0	2
-1	0	3

=

(3x3)

91	100	83
69	91	127
44	72	74

$$\left\lfloor \frac{7+2\times0-3}{2}+1 \right\rfloor = 3$$

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

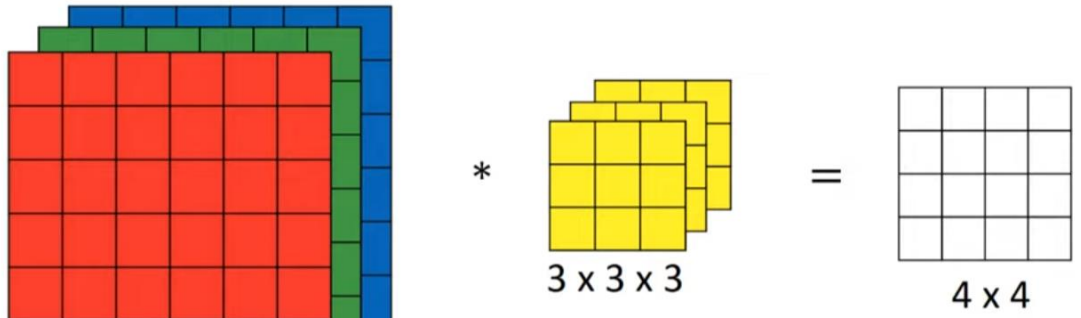
$$\left\lfloor \frac{f+2p-f}{s}+1 \right\rfloor \times \left\lfloor \frac{f+2p-f}{s}+1 \right\rfloor$$

floor function

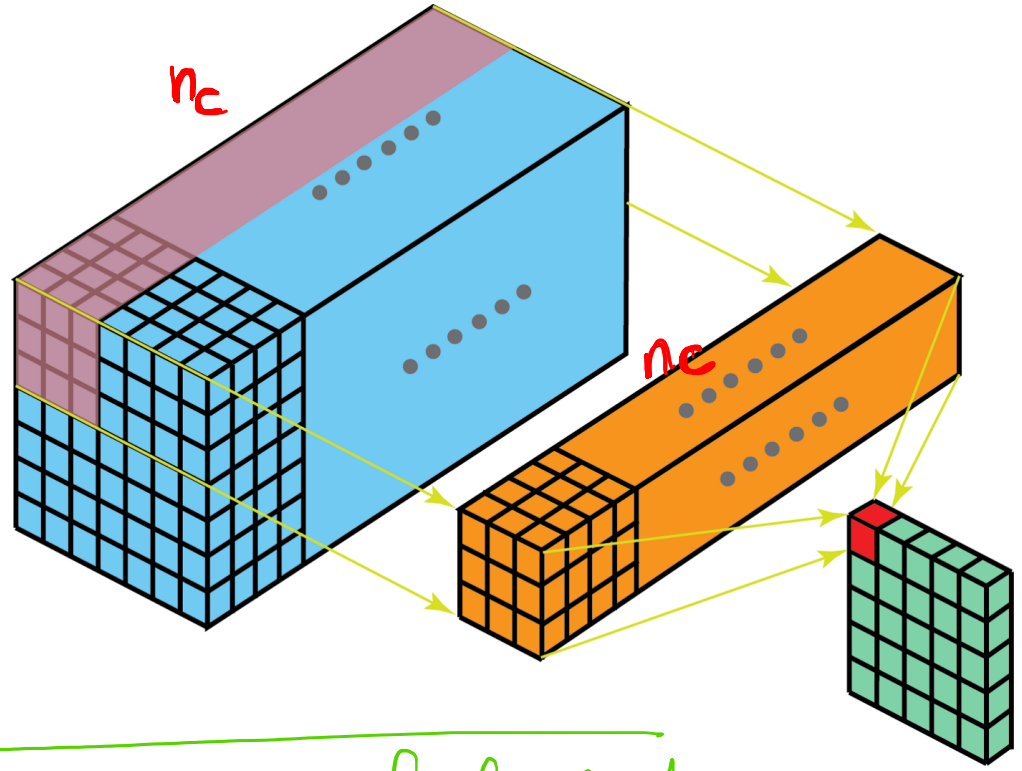
$\lfloor 1.9 \rfloor = \lfloor 1.2 \rfloor = \lfloor 1.5 \rfloor = 1$

2D Convolution over volume

Color image (RGB) channels



Arbitrary number of input channels



Convolution over volume

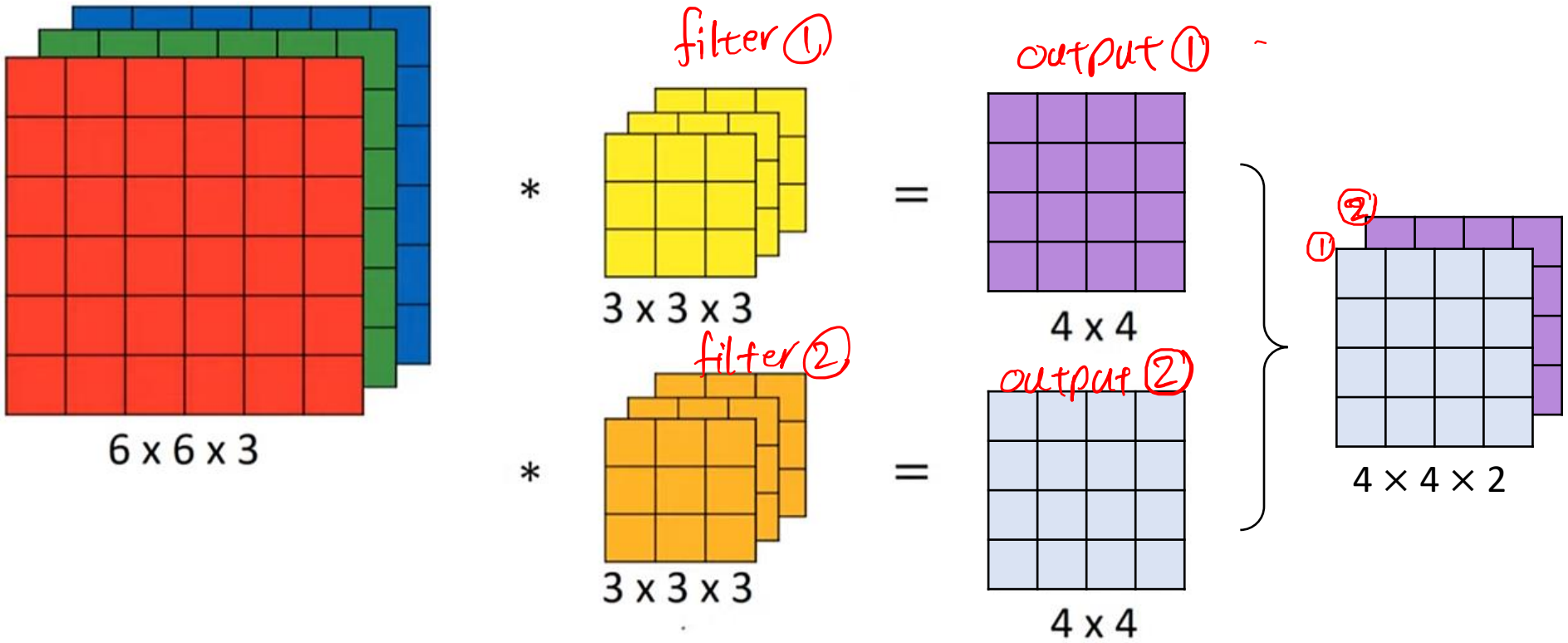
Input and filter have the same number of channels:

6x6x3 3x3x3

$n \times n \times n_c$ and $f \times f \times n_c$

Based on slides by DeepLearning.AI and Andrew Ng
Images from https://blog.csdn.net/shiner_chen/article/details/124319246

2D Convolution over volume – multiple filters



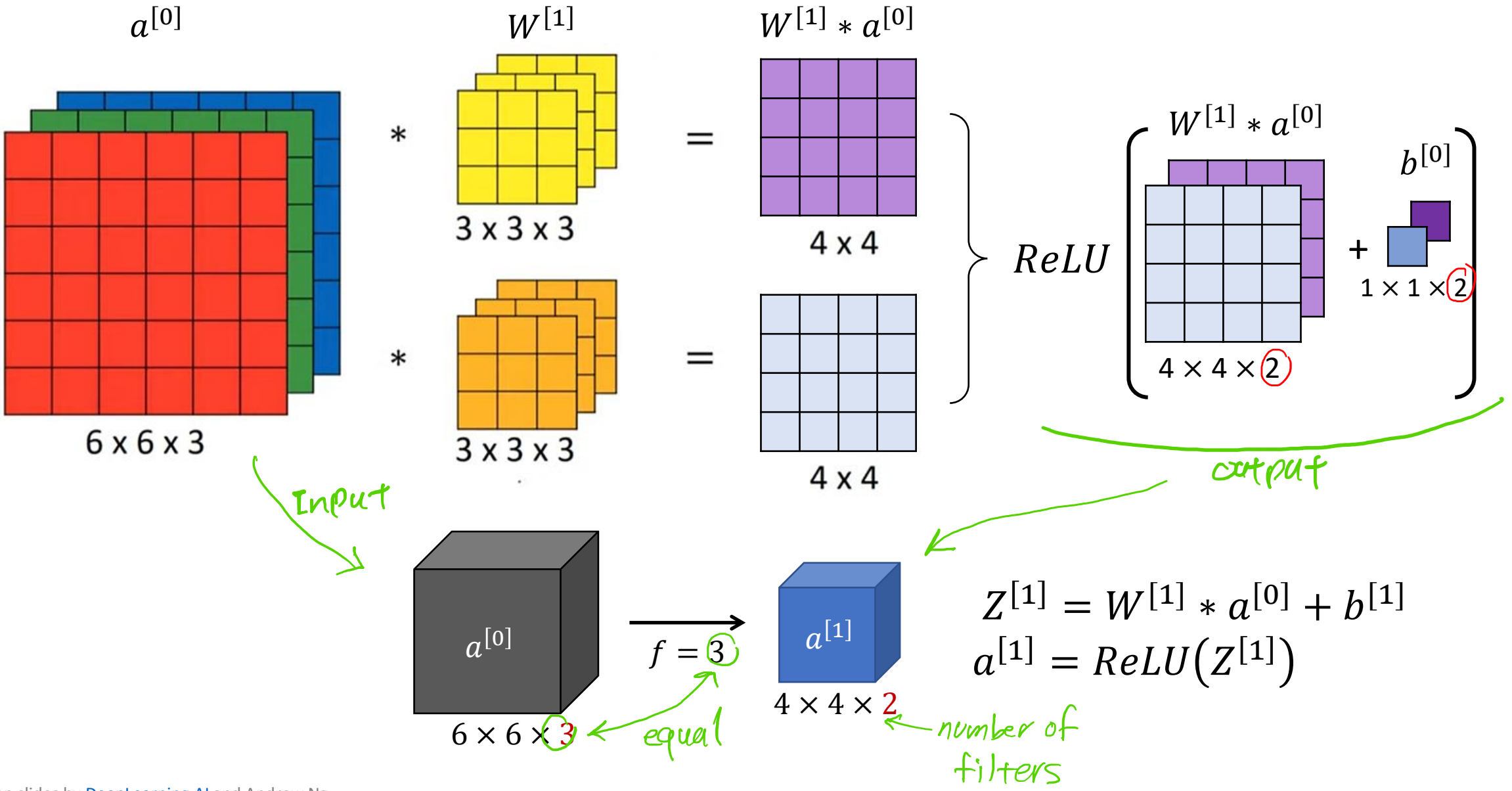
Output size given input size $n \times n \times n_c$ and filter size $f \times f \times n_c$:

$$(n-f+1) \times (n-f+1) \times n_f$$

$6-3+1 = 4$

n_f: 2 filters

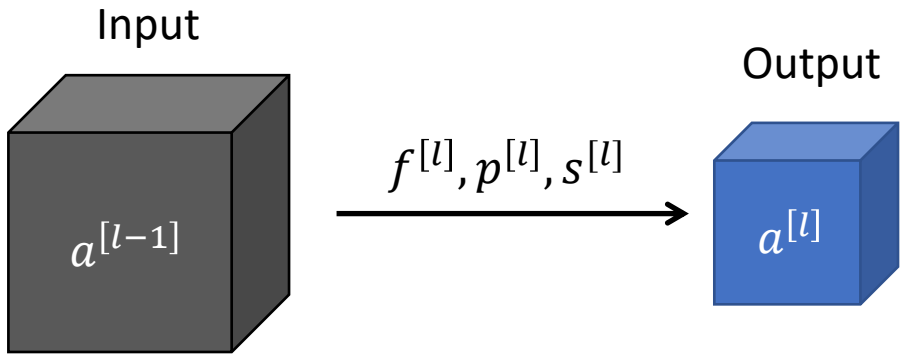
One layer of a convolutional network



Size relations in convolutional networks

If layer l is a 2D convolution layer:

$f^{[l]}$ = filter size
$p^{[l]}$ = padding size
$s^{[l]}$ = stride



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

How many weights we should train?

Find the following:

Number of filters: $n_c^{[l]}$

Filter size: $f^{[l]} \times f^{[l]} \times n_c^{[l-1]}$

Weight size: $\underbrace{f^{[l]} \times f^{[l]} \times n_c^{[l-1]}}_{\text{filter size}} * n_c^{[l]}$ → number of filters

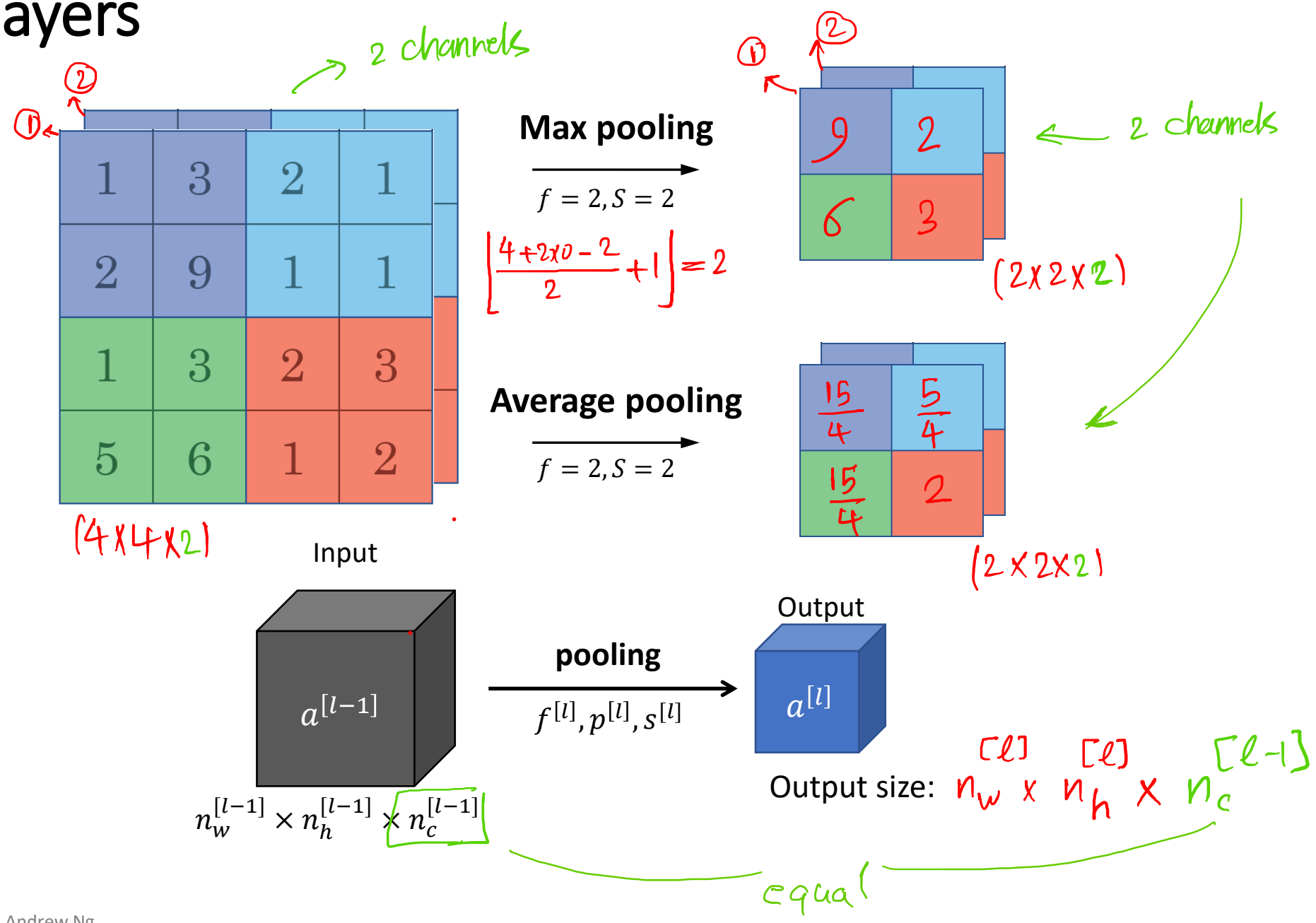
Bias size: $n_c^{[l]}$

Output size: $n_w^{[l]} \times n_h^{[l]} \times n_c^{[l]}$

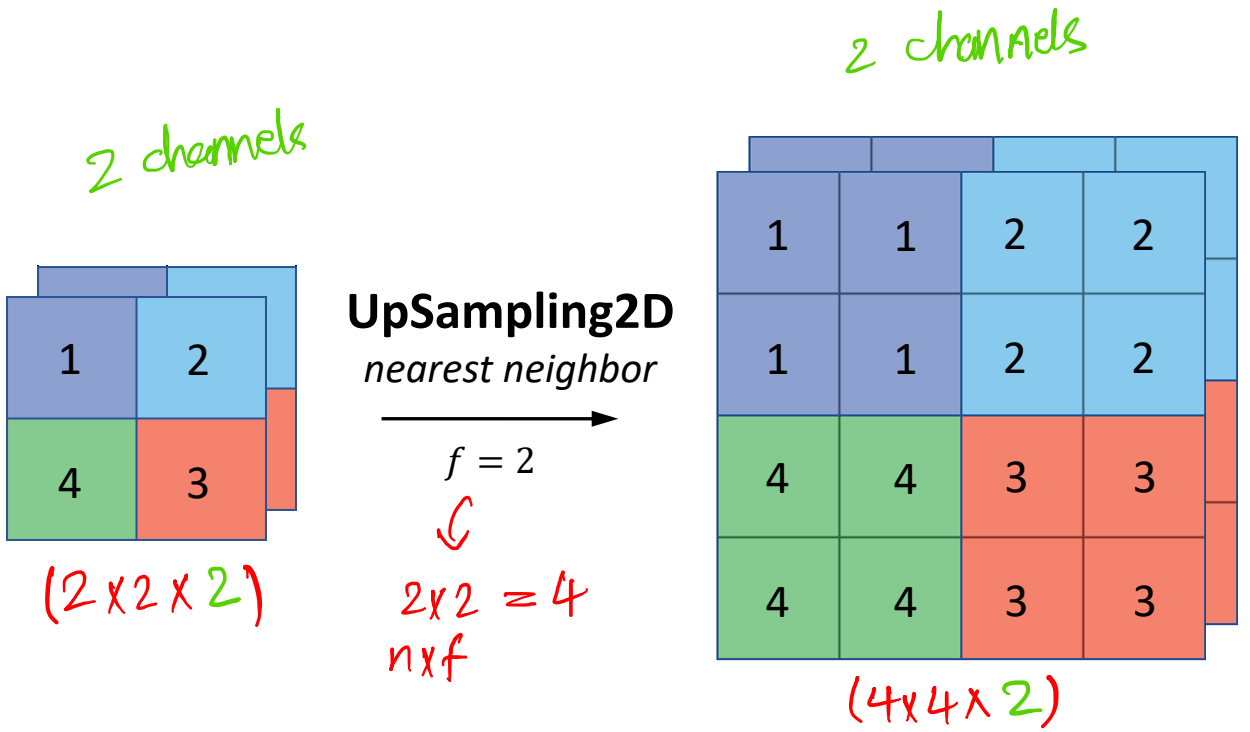
$n_w^{[l]} = \left\lfloor \frac{n_w^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \right\rfloor$

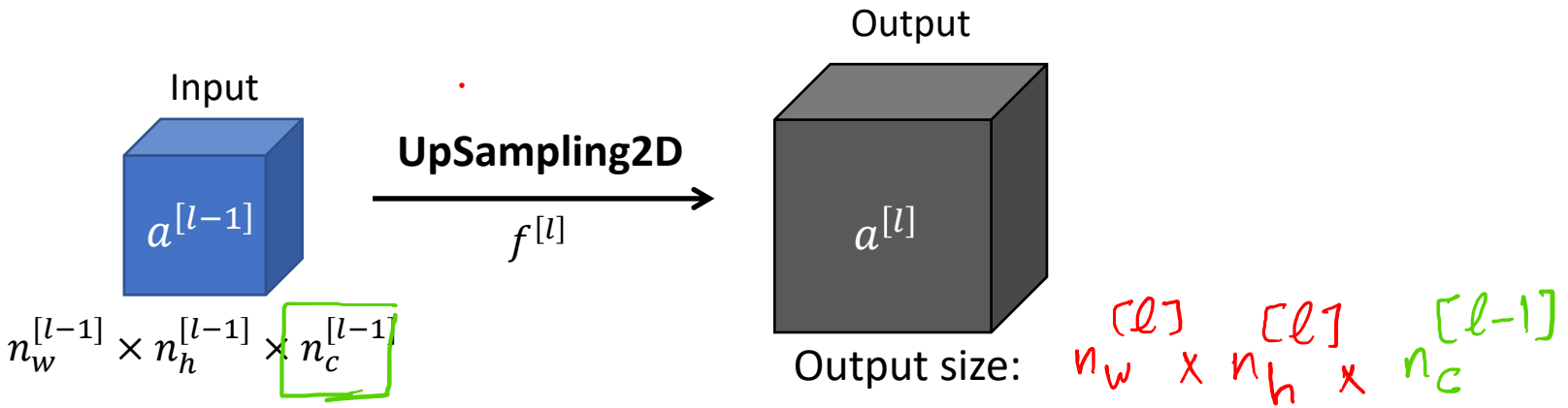
Activation size: $A^{[l]} = g^{[l]}(Z^{[l]}) \Rightarrow n_w^{[l]} \times n_h^{[l]} \times n_c^{[l]}$

Pooling layers

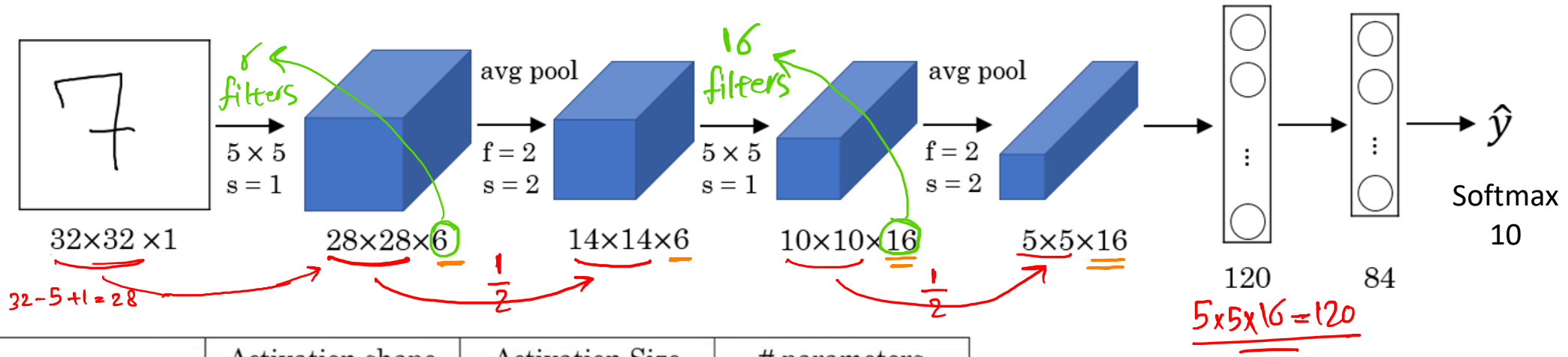


UpSampling


2 channels
1
2
4
3
(2x2x2)
UpSampling2D
nearest neighbor
$f = 2$
2x2 = 4
nxf
2 channels
1 1 2 2
1 1 2 2
4 4 3 3
4 4 3 3
(4x4x2)


Input
$a^{[l-1]}$
$n_w^{[l-1]} \times n_h^{[l-1]} \times n_c^{[l-1]}$
UpSampling2D
$f^{[l]}$
Output
$a^{[l]}$
Output size: $n_w^{[l]} \times n_h^{[l]} \times n_c^{[l-1]}$

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?

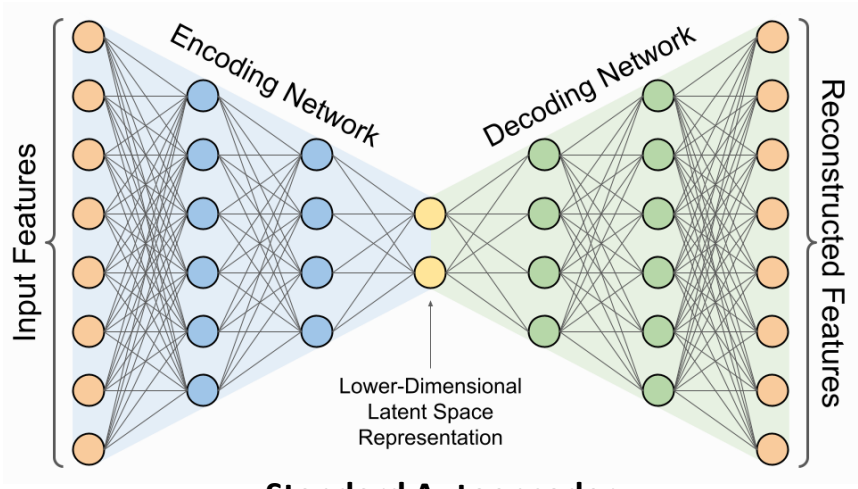
Parameter sharing: A feature detector (such as a vertical edge detector) that's useful in one part of the image is probably useful in another part of the image.

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

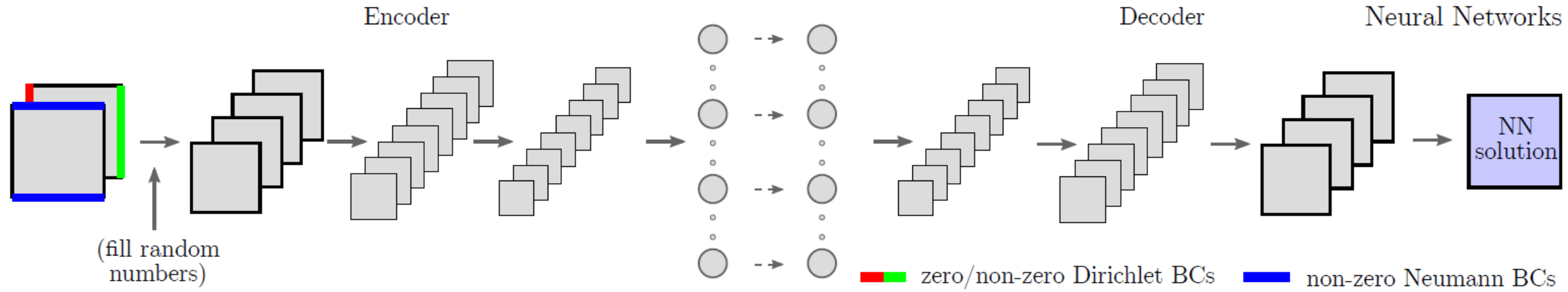
Based on slides by DeepLearning.AI and Andrew Ng

[LeCun et al., 1998. Gradient-based learning applied to document recognition]

ML-PDE: NN architecture



Standard Autoencoder

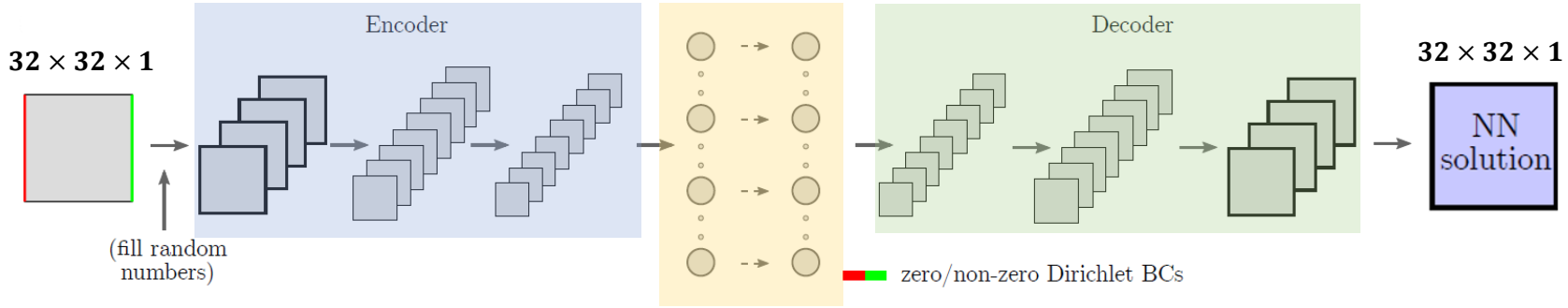


Convolutional Autoencoder

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/

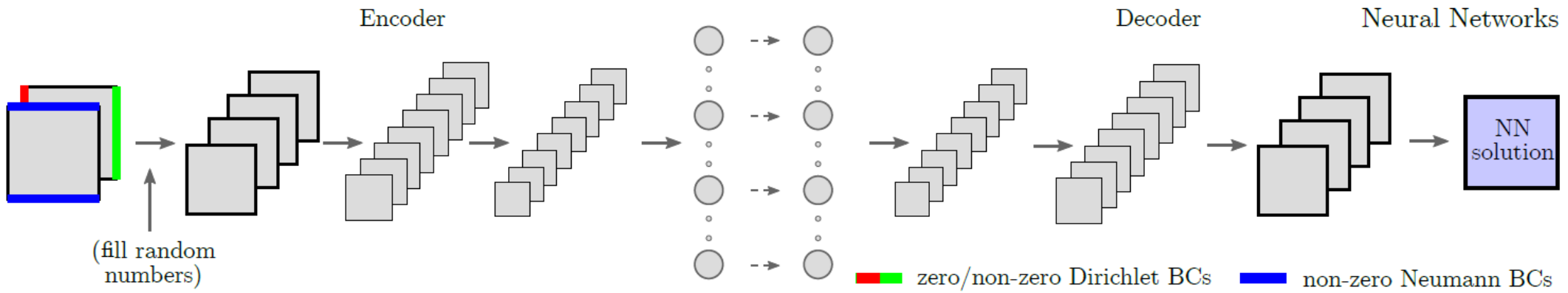
ML-PDE: NN architecture



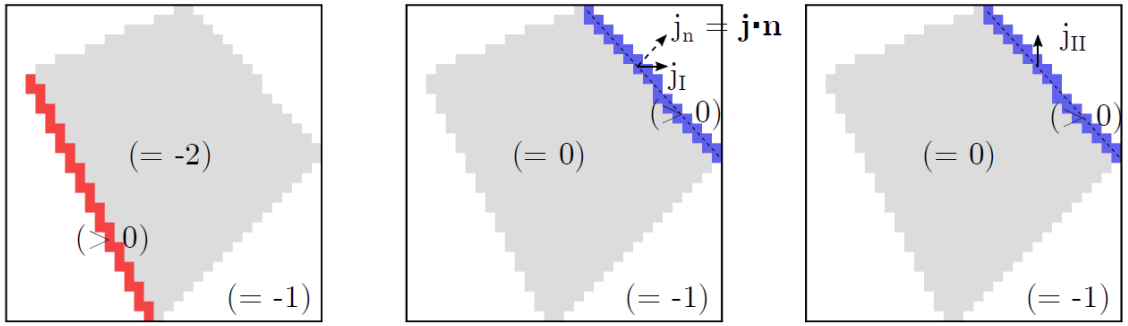
Deterministic	Size	Layer arguments
Input	-	-
LayerFillRandomNumber	-	-
Conv2D	filters = 8	kernel (5,5), padding: same, ReLU
MaxPooling2D	-	kernel (2,2), padding: same
Conv2D	filters = 8	kernel (5,5), padding: same, ReLU
MaxPooling2D	-	kernel (2,2), padding: same
Conv2D	filters = 8	kernel (5,5), padding: same, ReLU
MaxPooling2D	-	kernel (2,2), padding: same
Flatten	-	-
Dense	units = 32	ReLU
Dense	units = 32	ReLU
Reshape	-	$[4, 4, 2]$
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

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

ML-PDE: NN Input

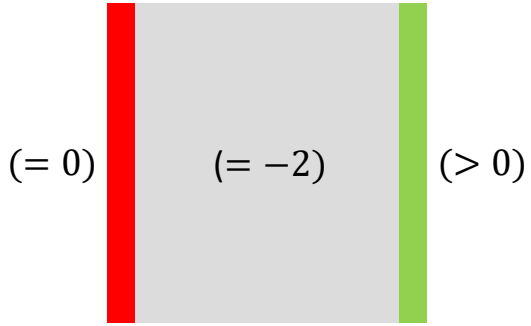


General Input



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

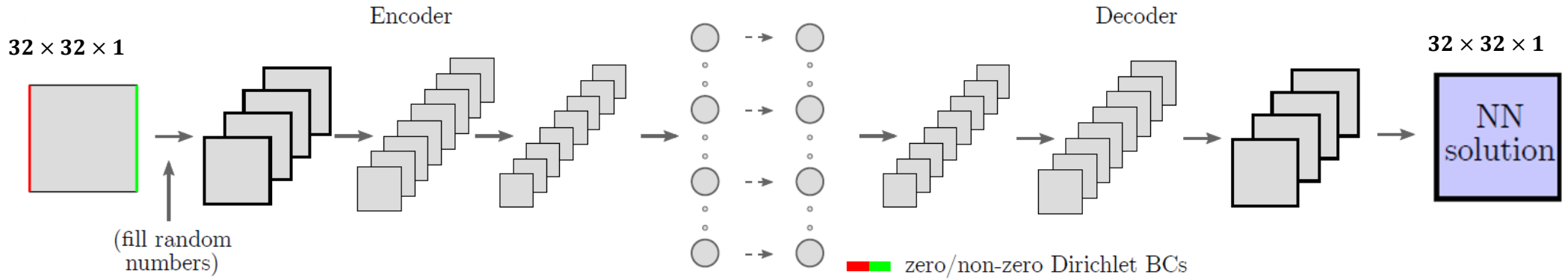
Simplified input (No non-zero Neumann BCs)



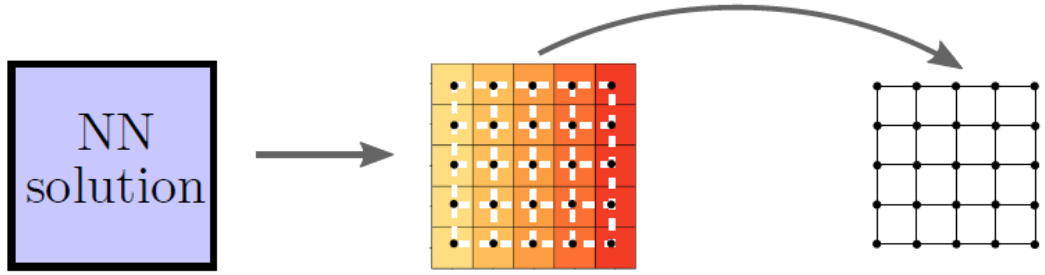
- $= 0$: zero Dirichlet boundary locations
- > 0 : non-zero Dirichlet boundary locations with boundary values
- $= -2$: domain interior filled with random numbers during training

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

ML-PDE: NN output and loss function

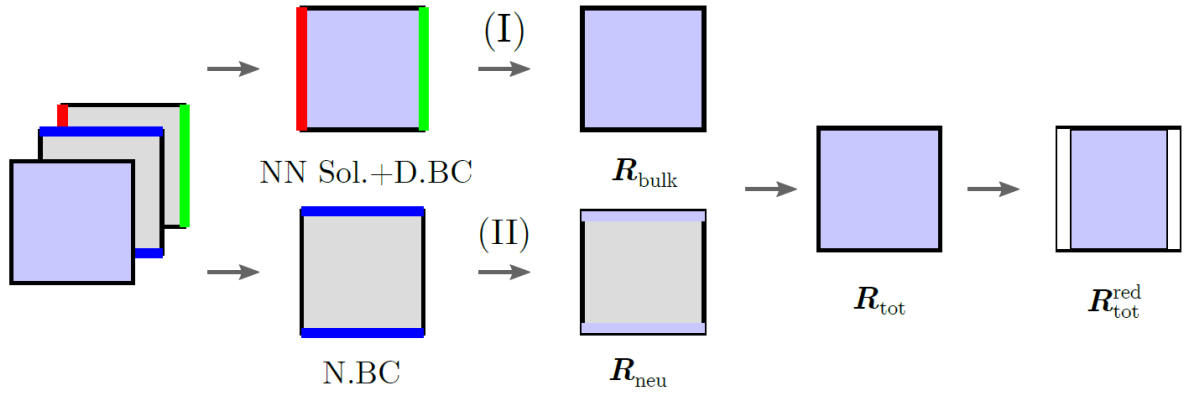


NN output is the solution of a PDE discretized by FEM



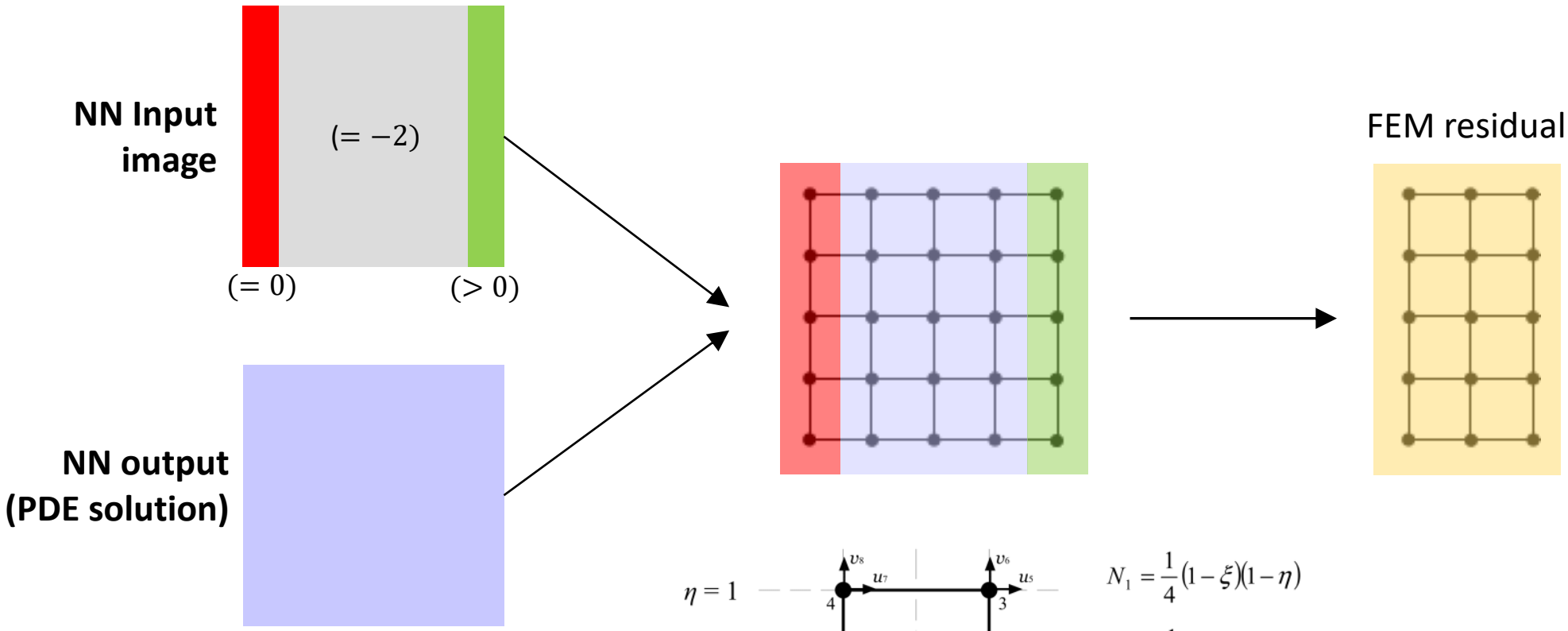
Canters of pixels are a nodes in the FEM mesh

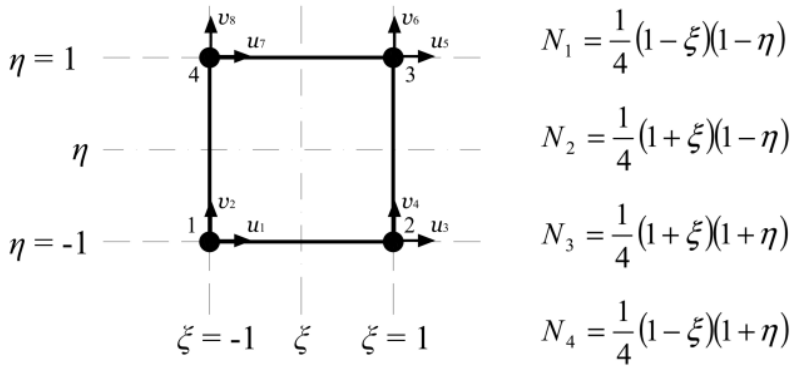
Label-free Loss function is based on residual of a PDE discretized by FEM



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

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





$$N_1 = \frac{1}{4}(1-\xi)(1-\eta)$$

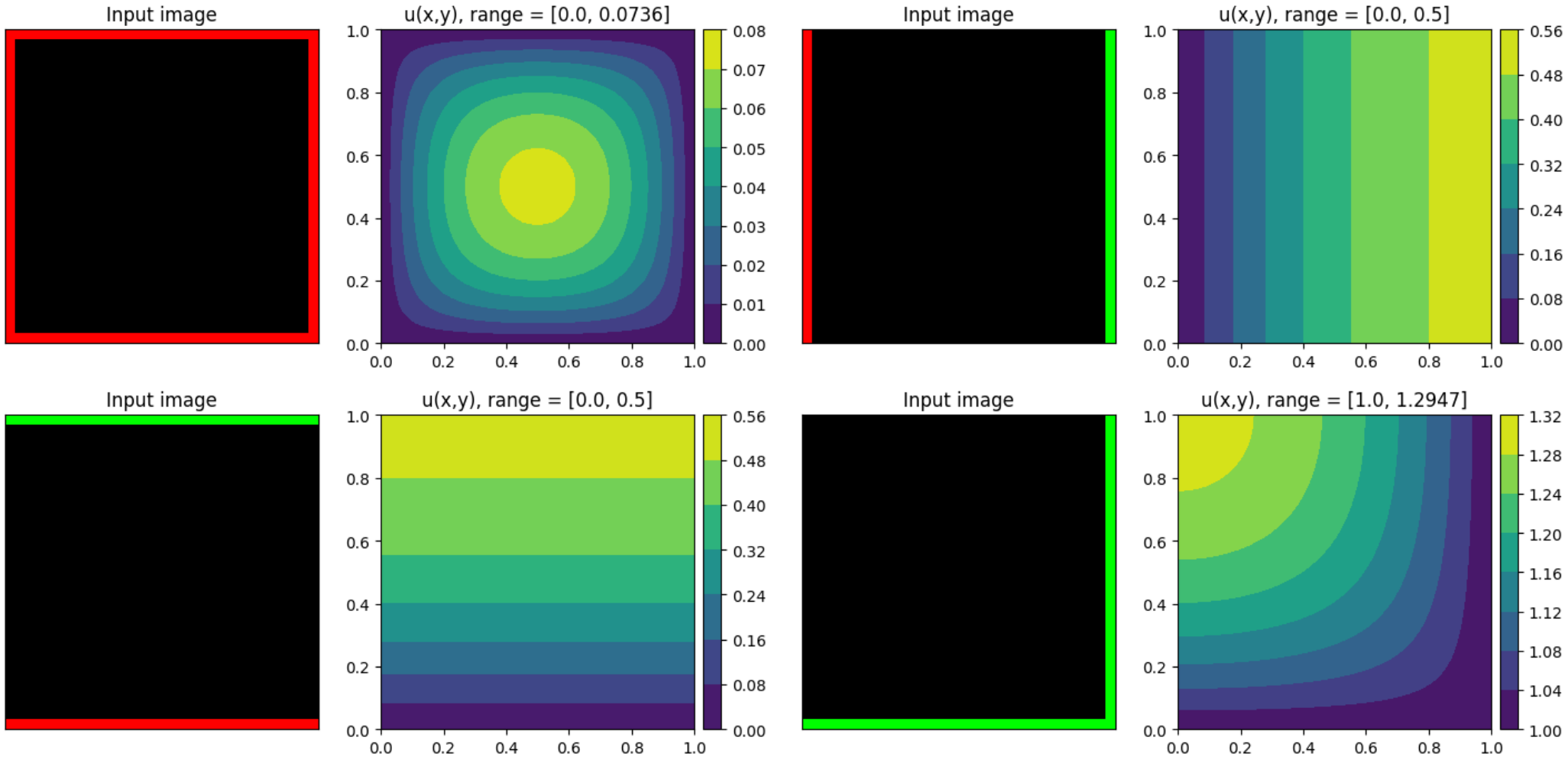
$$N_2 = \frac{1}{4}(1+\xi)(1-\eta)$$

$$N_3 = \frac{1}{4}(1+\xi)(1+\eta)$$

$$N_4 = \frac{1}{4}(1-\xi)(1+\eta)$$

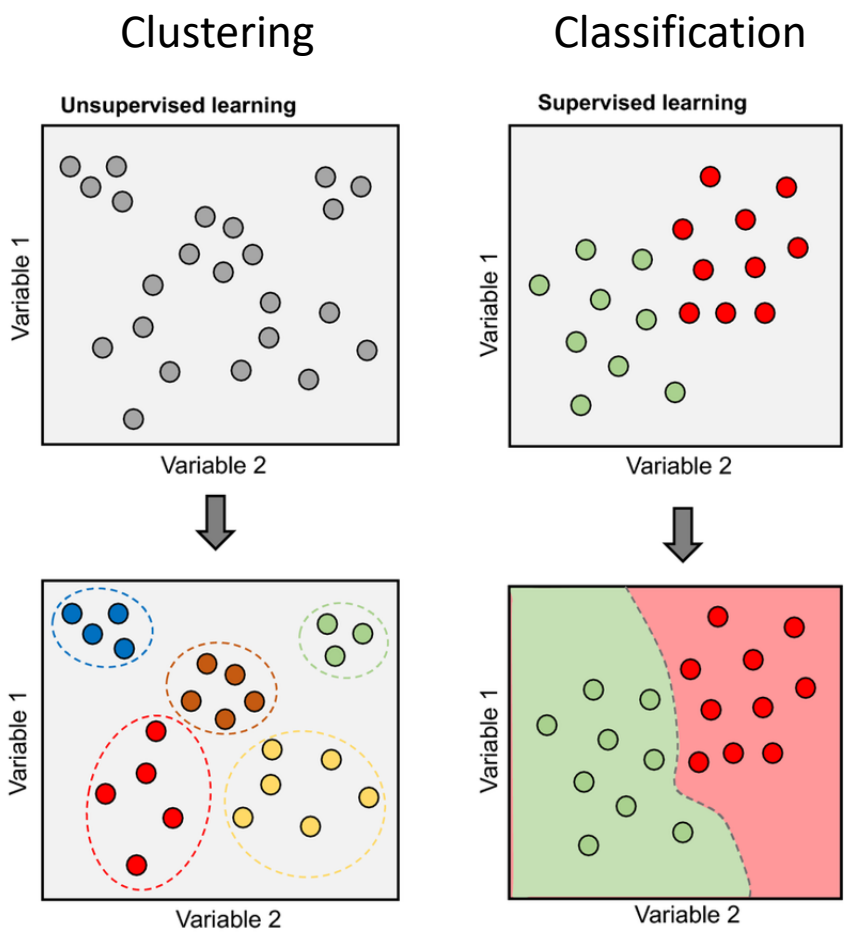
Element image from
https://www.researchgate.net/publication/322932020_Numerical_results_quality_in_dependence_on_abaqus_plane_stress_elements_type_in_big_displacements_compression_test

Sample results



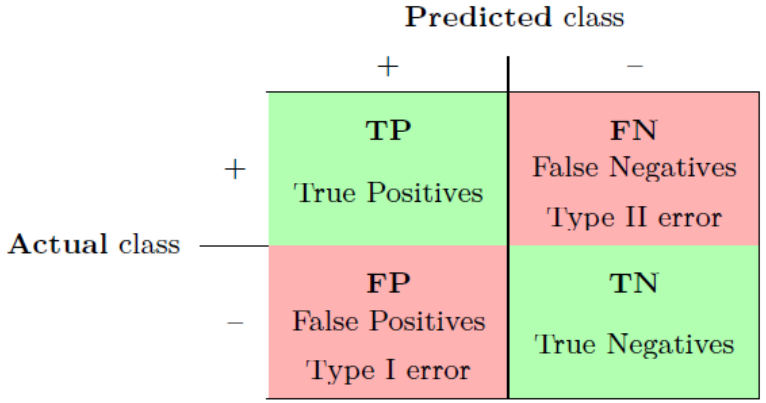
Supervised vs unsupervised learning

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
Classificationhttps://www.altexsoft.com/blog/unsupervised-machine-learning/

Classification metrics

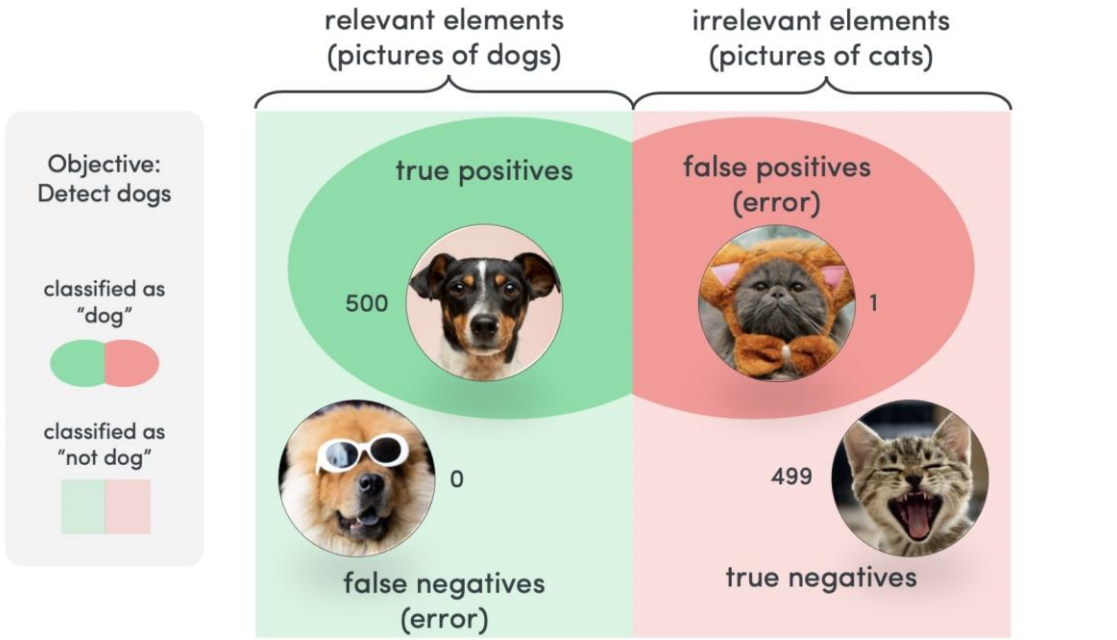


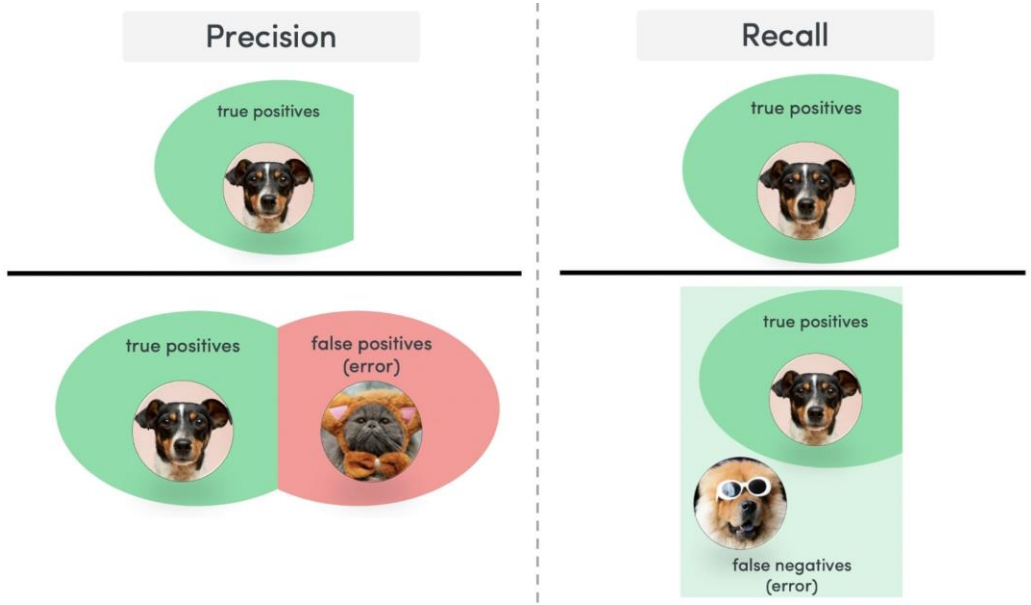
Confusion matrix

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

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

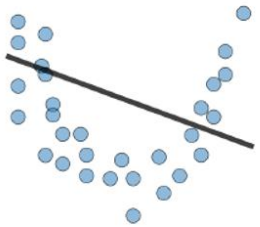
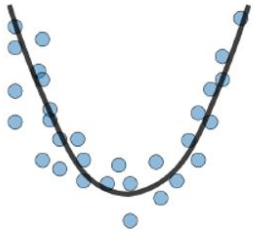
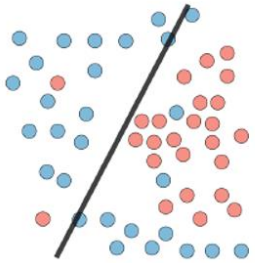
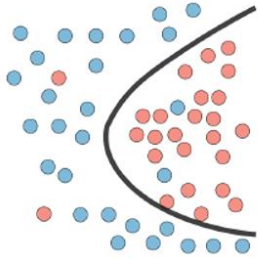
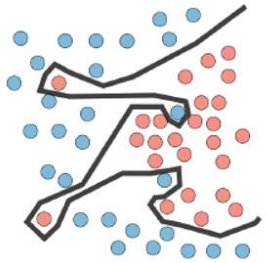
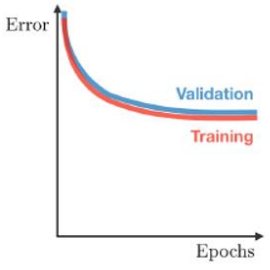
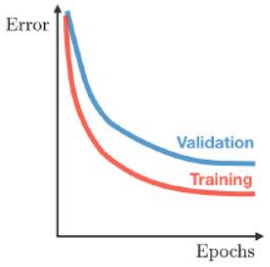
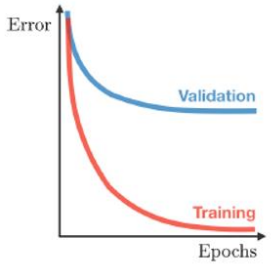
$$F_\beta \text{ Measure} = \frac{(1+\beta)\times Precision\times Recall}{(\beta\times Precision)+Recall}$$



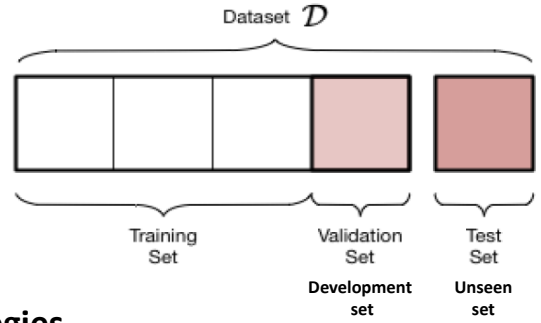


Content from Super VIP Cheatsheet: Deep Learning, Afshine Amidi and Shervine Amidi, 2018
Images form https://levity.ai/blog/precision-vs-recall

Model diagnostics

	Underfitting	Just right	Overfitting
Symptoms	- High training error - Training error close to test error - High bias	- Training error slightly lower than test error	- Low training error - Training error much lower than test error - High variance
Regression			
Classification			
Deep learning			
Remedies	- Complexify model - Add more features - Train longer		- 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.



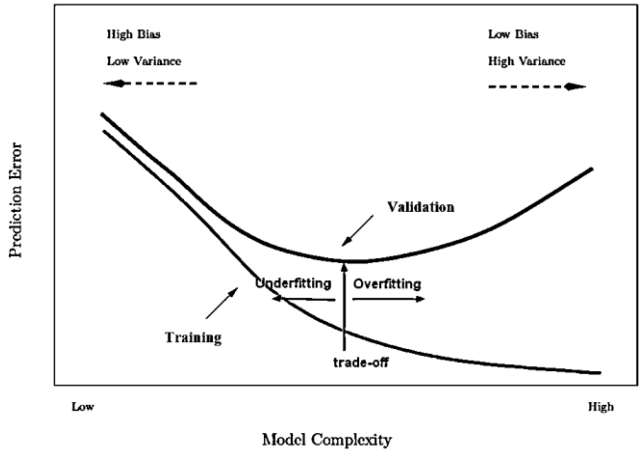
Model diagnostics terminologies

"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)

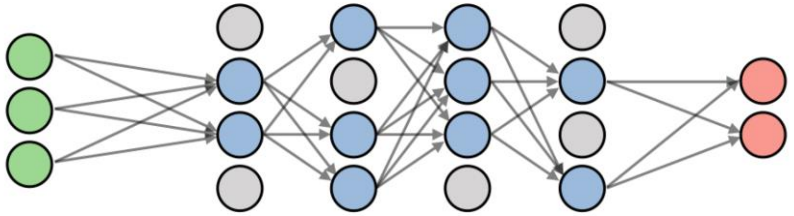
Content from Super VIP Cheatsheet: Deep Learning, Afshine Amidi and Shervine Amidi, 2018 and 2019 slides by Lex Fridman for MIT Deep Learning and Artificial Intelligence Lectures

Regularization

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

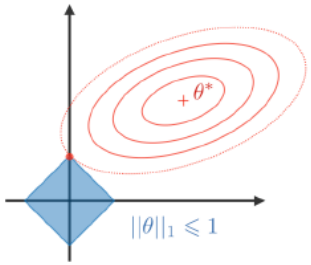
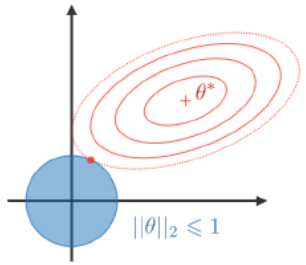
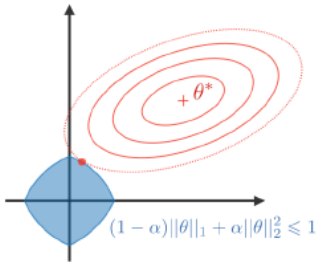
Examples:

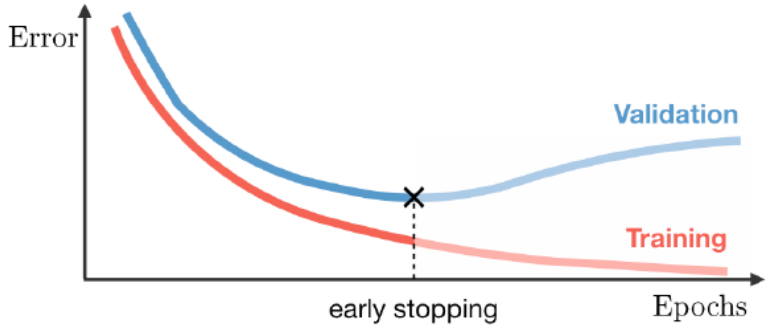
- Early stoppage in model training
- Dropout in NNs
- Parameter/weight regularization
- Data augmentation



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

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
		
$\ldots + \lambda\|\|\theta\|\|_1$ $\lambda \in \mathbb{R}$	$\ldots + \lambda\|\|\theta\|\|_2^2$ $\lambda \in \mathbb{R}$	$\ldots + \lambda\left[(1-\alpha)\|\|\theta\|\|_1 + \alpha\|\|\theta\|\|_2^2\right]$ $\lambda \in \mathbb{R}, \alpha \in [0,1]$

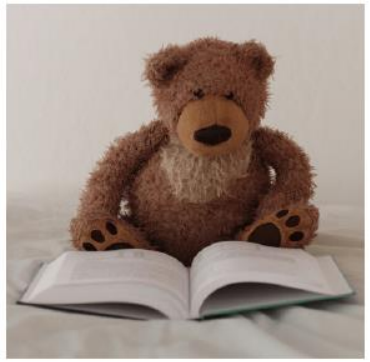


Early stopping – Stop the training process as soon as the validation loss reaches a plateau or starts to increase.

Content from Super VIP Cheatsheet: Deep Learning, Afshine Amidi and Shervine Amidi, 2018

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
			

Content from Super VIP Cheatsheet: Deep Learning, Afshine Amidi and Shervine Amidi, 2018