

Deep Learning I

Itthi Chatnuntawech

Nanoinformatics and Artificial Intelligence (NAI)

National Nanotechnology Center (NANOTEC)

National Science and Technology Development Agency (NSTDA)

January 31, 2023

Outline

- ❖ Introduction to Supervised Machine Learning
 - ❖ AI vs ML vs DL
 - ❖ Traditional Machine Learning
 - ❖ Training, Validation, and Test Data
 - ❖ Overfitting and Underfitting
- ❖ Introduction to Supervised Deep Learning
 - ❖ Traditional ML vs Deep Learning
 - ❖ Artificial Neuron and Neural Network
 - ❖ Supervised Learning
- ❖ Deep Learning as a Function Approximator
- ❖ Deep Learning Components
 - ❖ Fully Connected Layers
 - ❖ Activation Functions
 - ❖ Optimization

Outline

- ❖ Introduction to Supervised Machine Learning
 - ❖ AI vs ML vs DL
 - ❖ Traditional Machine Learning
 - ❖ Training, Validation, and Test Data
 - ❖ Overfitting and Underfitting
- ❖ Introduction to Supervised Deep Learning
 - ❖ Traditional ML vs Deep Learning
 - ❖ Artificial Neuron and Neural Network
 - ❖ Supervised Learning
- ❖ Deep Learning as a Function Approximator
- ❖ Deep Learning Components
 - ❖ Fully Connected Layers
 - ❖ Activation Functions
 - ❖ Optimization

ARTIFICIAL INTELLIGENCE

Any technique that enables computers to mimic human behavior



1950's 1960's 1970's 1980's 1990's 2000's 2010's

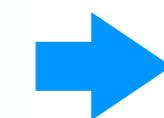
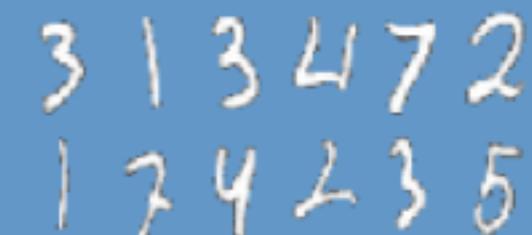
MACHINE LEARNING

Ability to learn without explicitly being programmed



DEEP LEARNING

Learn underlying features in data using neural networks



Fasting blood sugar test



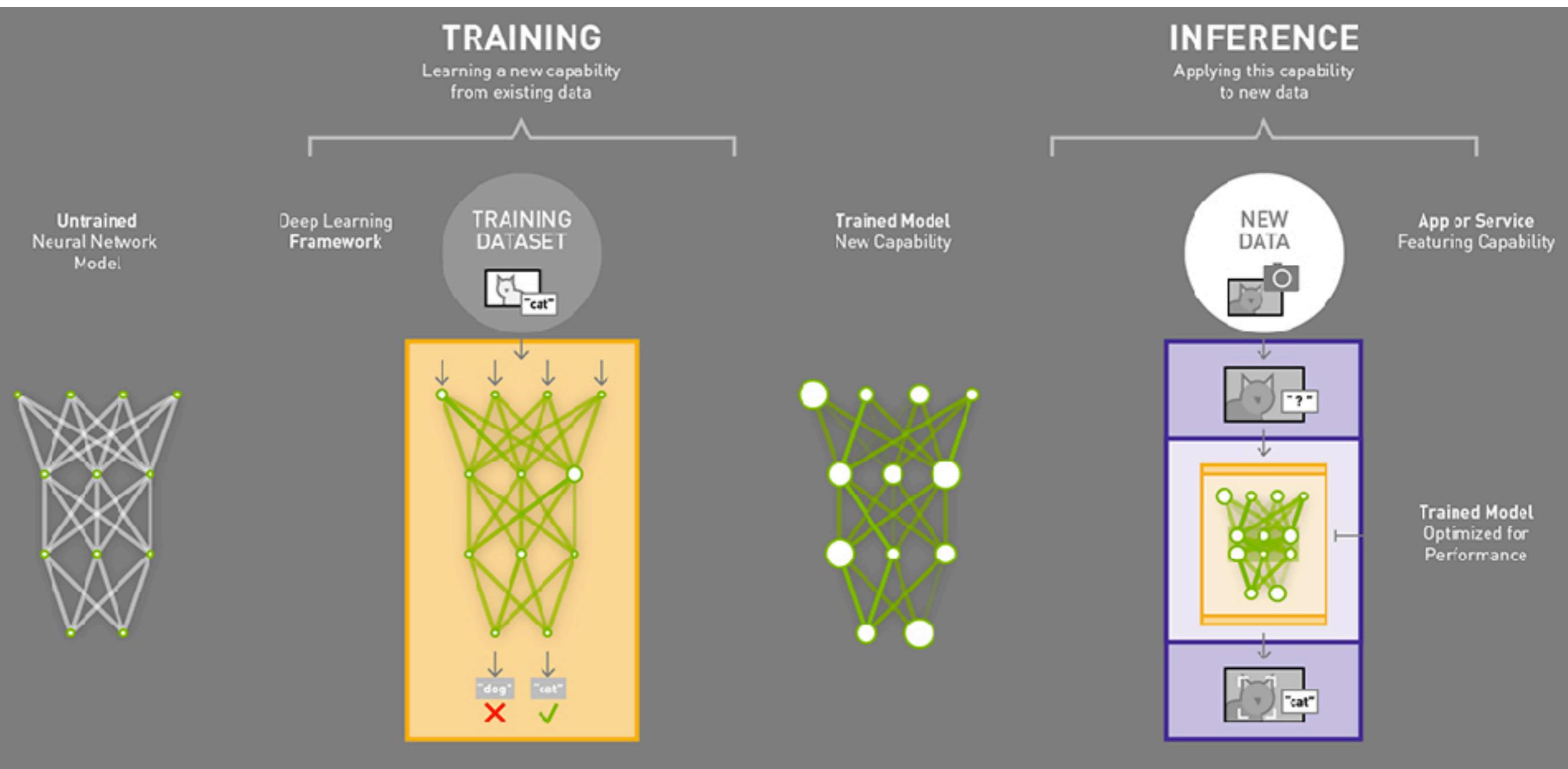
x mg/dL

normal if $x < 100$

prediabetes if $100 \leq x \leq 125$

diabetes if $125 < x$

Overview: Machine Learning



Create a model

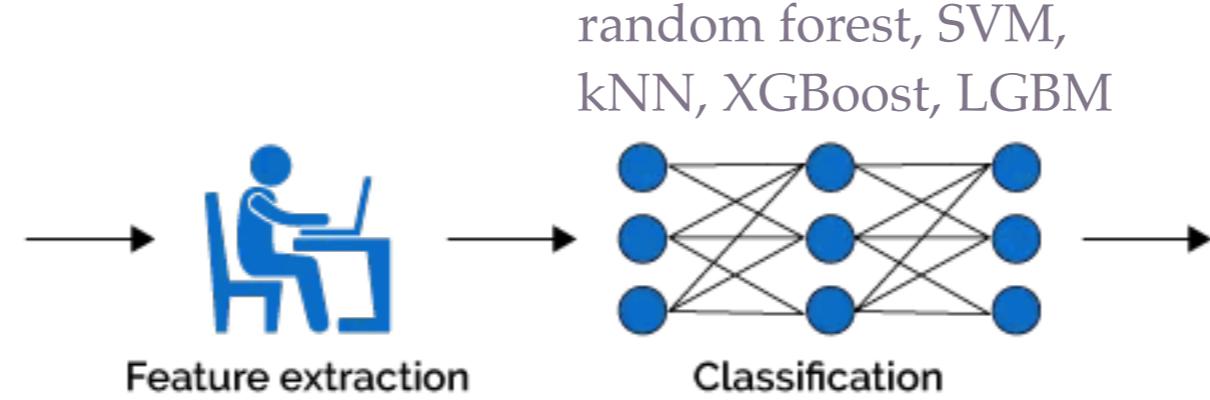
Train the model with **training data**

Test the trained model with **test data**

Example: Watermelon or Orange?

Traditional Machine Learning

Input image

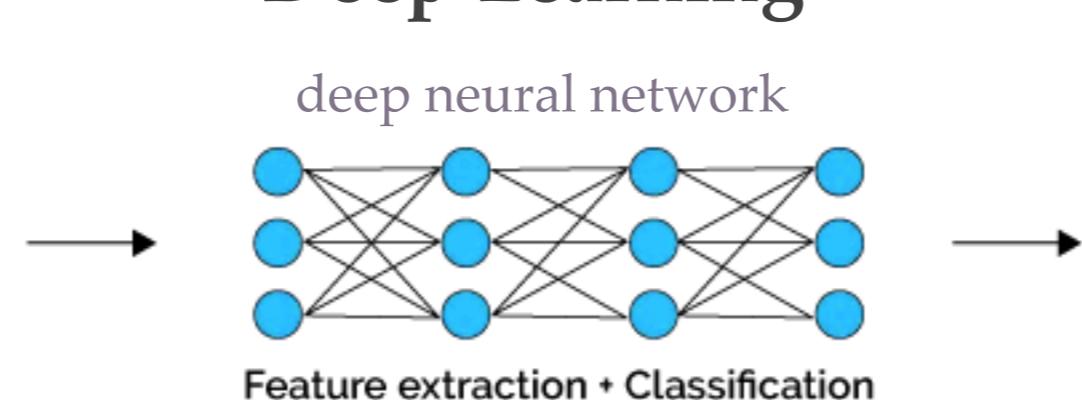


Output

watermelon

Deep Learning

Input image

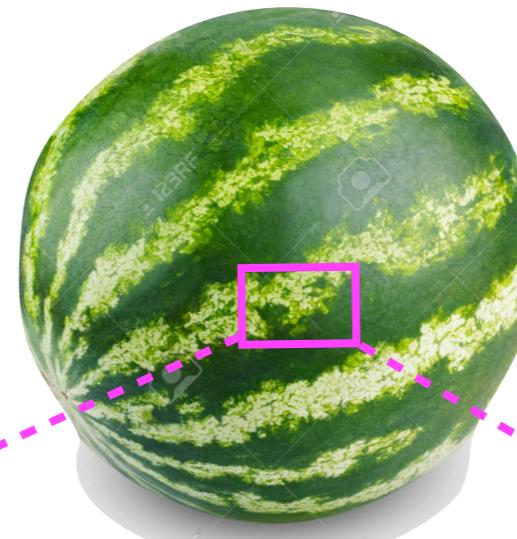


Output

watermelon

Some parts taken from: Azua Tech

What we see



orange: 0
watermelon: 1

What computers see

52	55	56	58	58	60	54	48	58	61	57	66	68	113	117	86	129	153	181	206	205	207	203	203	217	209		
55	75	97	78	94	64	55	52	57	55	71	87	72	97	133	148	198	204	204	209	204	200	204	194	153	204		
82	111	121	97	107	90	54	77	68	58	134	179	113	119	163	205	203	188	203	206	194	184	187	149	144	198		
108	98	134	143	119	84	69	72	77	103	169	199	187	185	178	190	172	188	201	164	188	157	95	98	140	171	107	
125	135	164	187	192	151	74	69	84	125	149	136	192	198	211	201	161	192	190	146	140	106	77	73	101	135	7	
159	197	194	192	177	118	98	132	132	140	173	138	160	161	195	192	191	176	125	139	123	99	93	79	113	123	3	
180	193	182	138	140	118	136	154	137	158	173	185	134	93	134	193	209	186	182	219	162	149	176	128	146	154	3	
190	172	175	158	174	179	147	153	181	195	189	182	124	184	158	190	193	185	168	137	98	99	132	152	166	164	9	
126	178	163	187	213	177	181	169	201	187	201	188	126	144	179	196	218	203	171	165	139	155	157	163	145	136	5	
159	171	138	177	179	140	191	166	138	158	208	195	201	194	165	155	157	164	201	186	178	187	160	109	75	99	9	
153	87	105	114	107	118	163	95	71	105	105	99	140	194	178	154	138	194	217	192	178	146	175	137	88	92	7	
106	138	146	107	87	57	76	54	56	64	54	64	58	76	167	163	156	188	183	171	173	121	123	155	137	90	189	
109	145	192	146	163	89	39	49	54	62	65	61	55	66	153	163	145	153	145	136	129	72	72	120	148	84	1	169
121	133	139	168	189	157	52	42	54	58	57	57	81	112	184	141	168	112	101	101	74	61	57	69	76	67	4	137
156	160	128	165	193	184	101	54	68	50	61	79	102	110	141	149	114	63	72	71	61	54	56	58	59	65	5	
173	162	160	172	171	138	151	110	55	54	55	59	121	115	96	139	75	56	70	61	69	62	57	60	62	64	7	
188	187	161	193	175	107	156	118	75	56	51	61	56	55	49	66	89	86	74	63	62	72	60	54	60	65	7	
180	166	161	210	172	127	126	119	66	54	44	53	49	48	69	95	99	96	80	54	56	61	64	57	57	59	1	
128	125	86	136	181	176	173	107	63	59	41	39	40	47	84	74	67	57	47	48	51	51	50	57	57	58	4	
56	84	69	49	74	84	109	85	51	49	40	41	40	43	46	55	47	47	41	41	46	53	49	51	50	54	5	
39	65	51	39	56	46	33	43	42	38	35	44	37	36	35	37	39	41	38	42	45	48	46	49	52	50	5	
288	186	178	219	188	151	153	148	94	88	68	78	73	73	98	129	132	138	112	82	83	88	92	88	90	93	1	
159	153	121	158	195	194	192	135	98	85	64	62	64	72	116	108	99	85	73	75	78	78	79	87	90	92	2	
90	119	100	82	104	189	133	116	76	74	63	66	66	69	74	83	73	73	57	67	73	81	78	82	82	87	1	
65	95	81	69	88	74	58	58	66	61	58	78	62	60	60	62	65	67	54	68	72	77	73	78	84	90	7	
106	102	104	104	108	63	49	43	25	20	29	33	30	30	33	40	39	40	37	34	40	40	44	51	40	44	45	
63	72	43	88	117	89	84	44	27	29	26	24	27	28	38	35	33	34	36	37	36	37	36	44	44	43		
29	42	36	25	33	41	58	35	27	28	25	27	27	35	30	36	31	35	31	31	33	39	37	37	39	37	40	
25	33	26	24	37	34	26	29	30	28	26	36	29	32	27	28	31	34	30	32	36	37	38	40	46	45		

Red

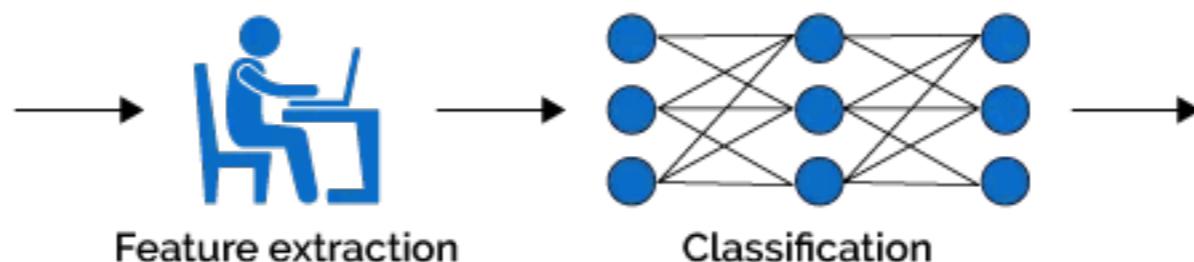
Green

Blue

Watermelon or Orange?

Traditional Machine Learning

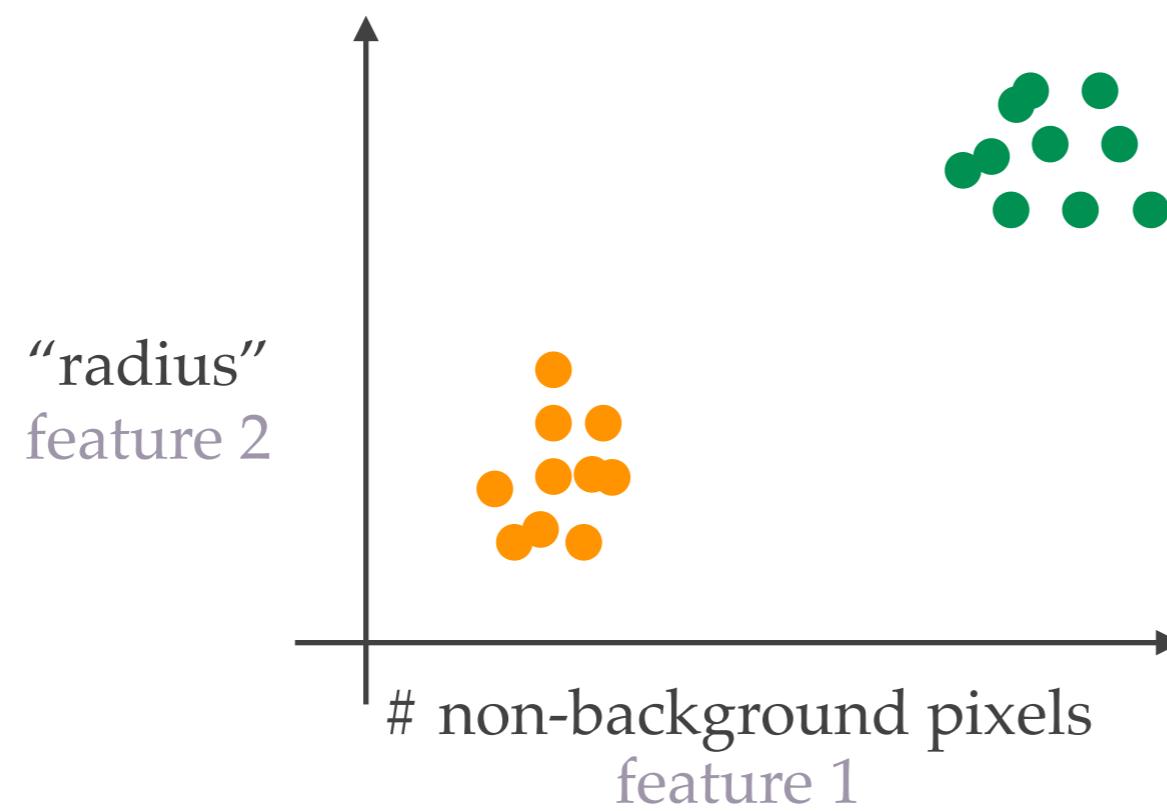
Input image



Output

watermelon

Phase: Training

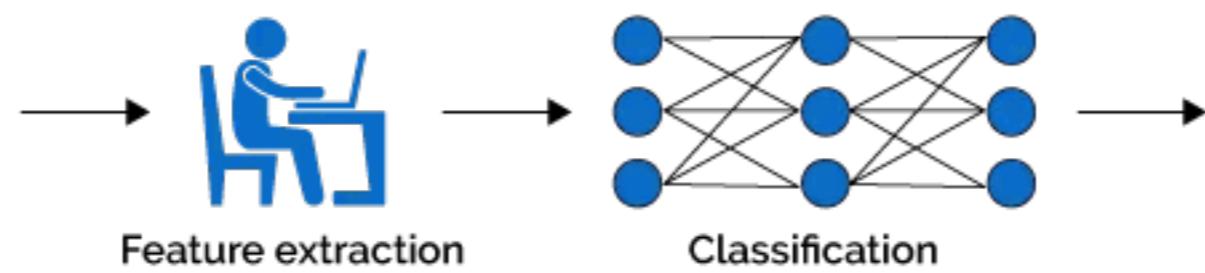


keywords: feature extraction, feature engineering, handcrafted / hand-designed features, feature space, descriptors

Watermelon or Orange?

Traditional Machine Learning

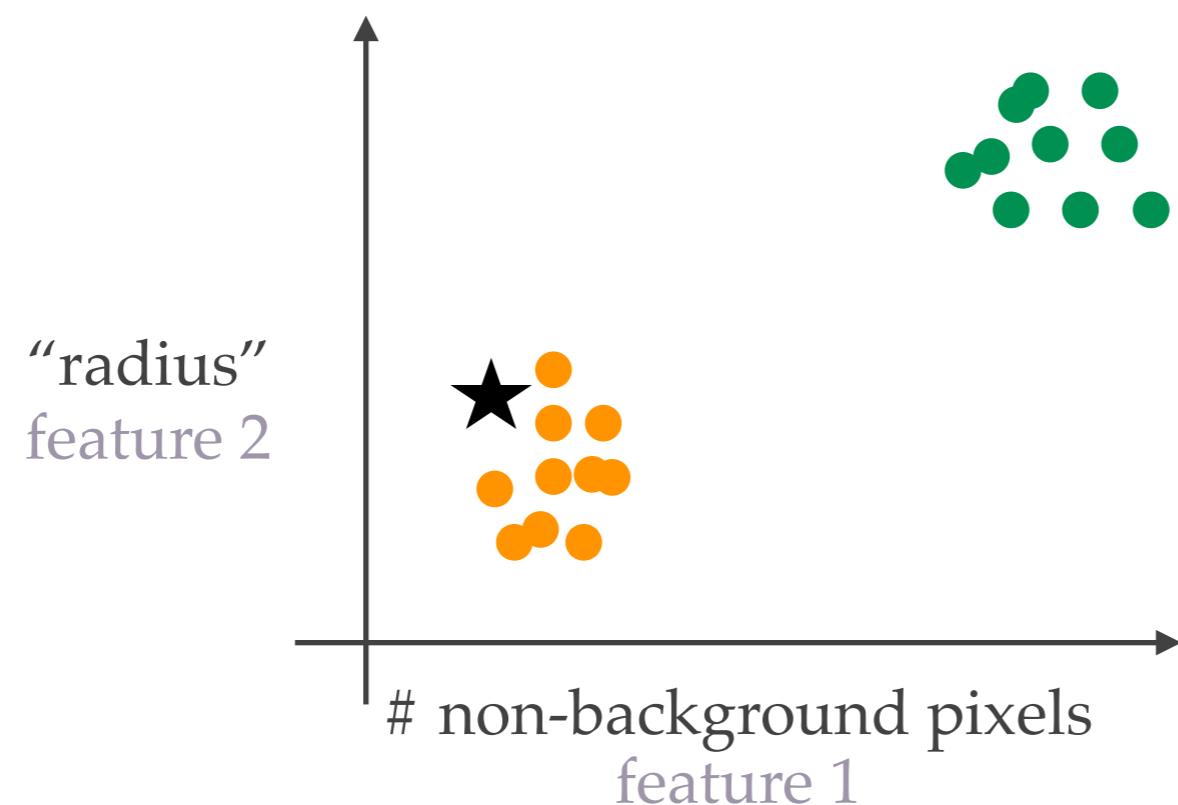
Input image



Output

watermelon

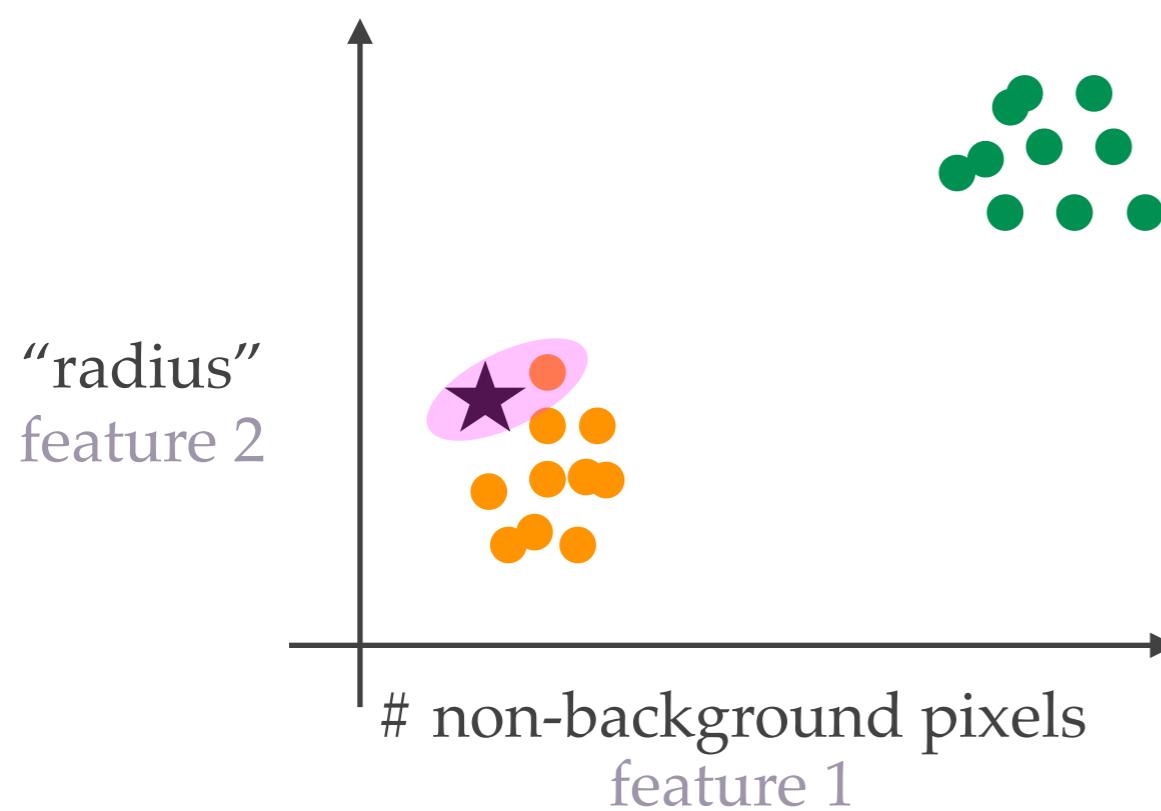
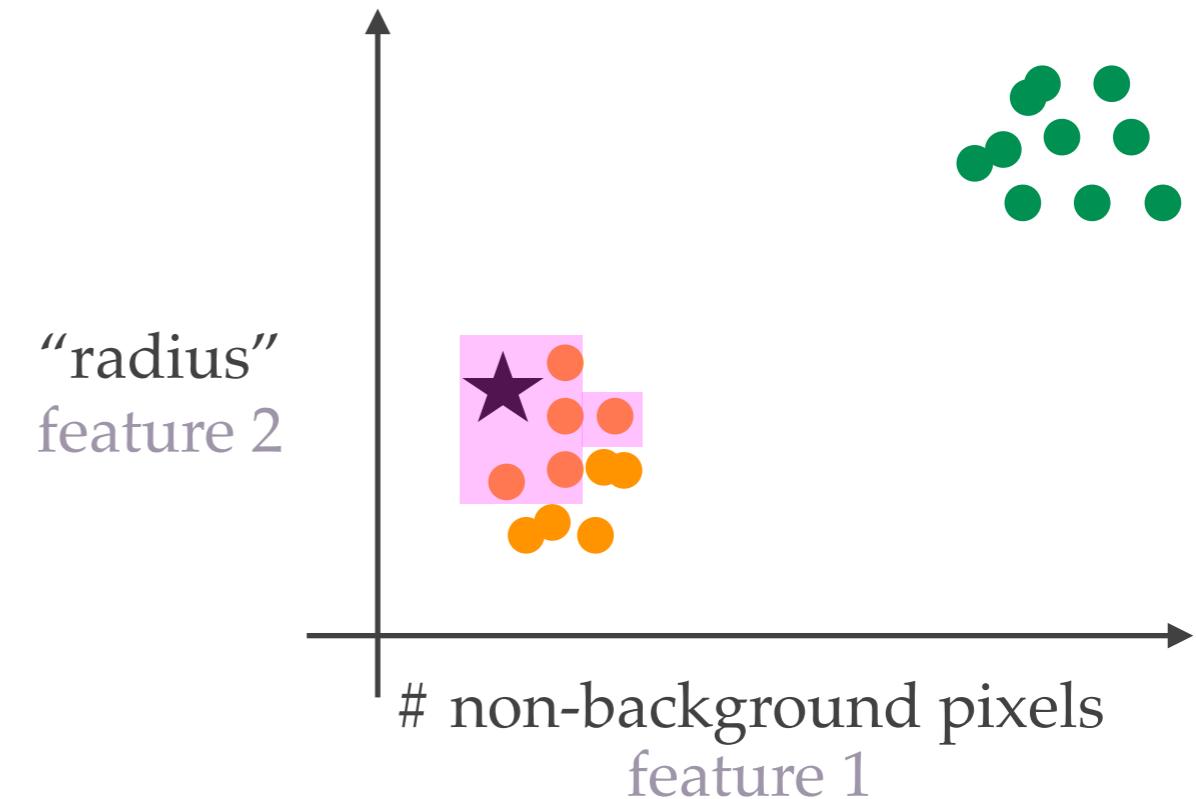
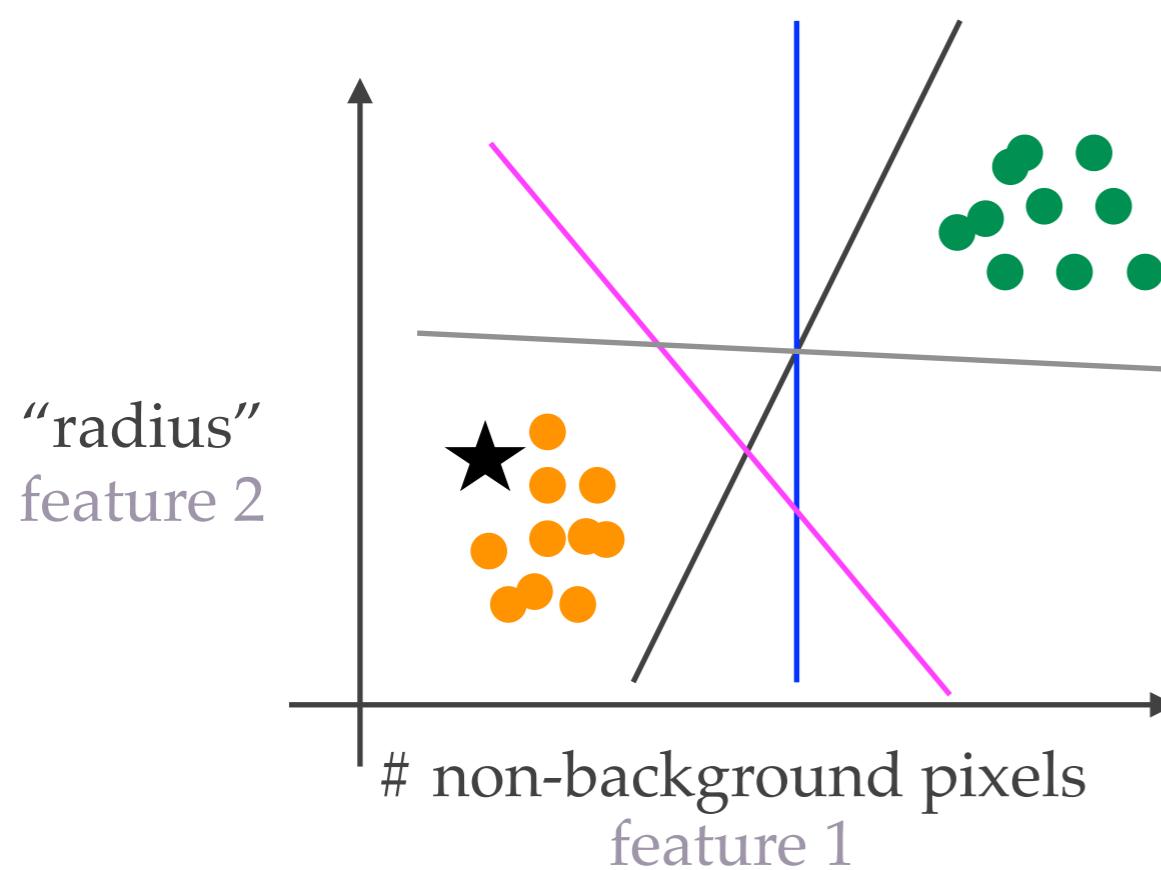
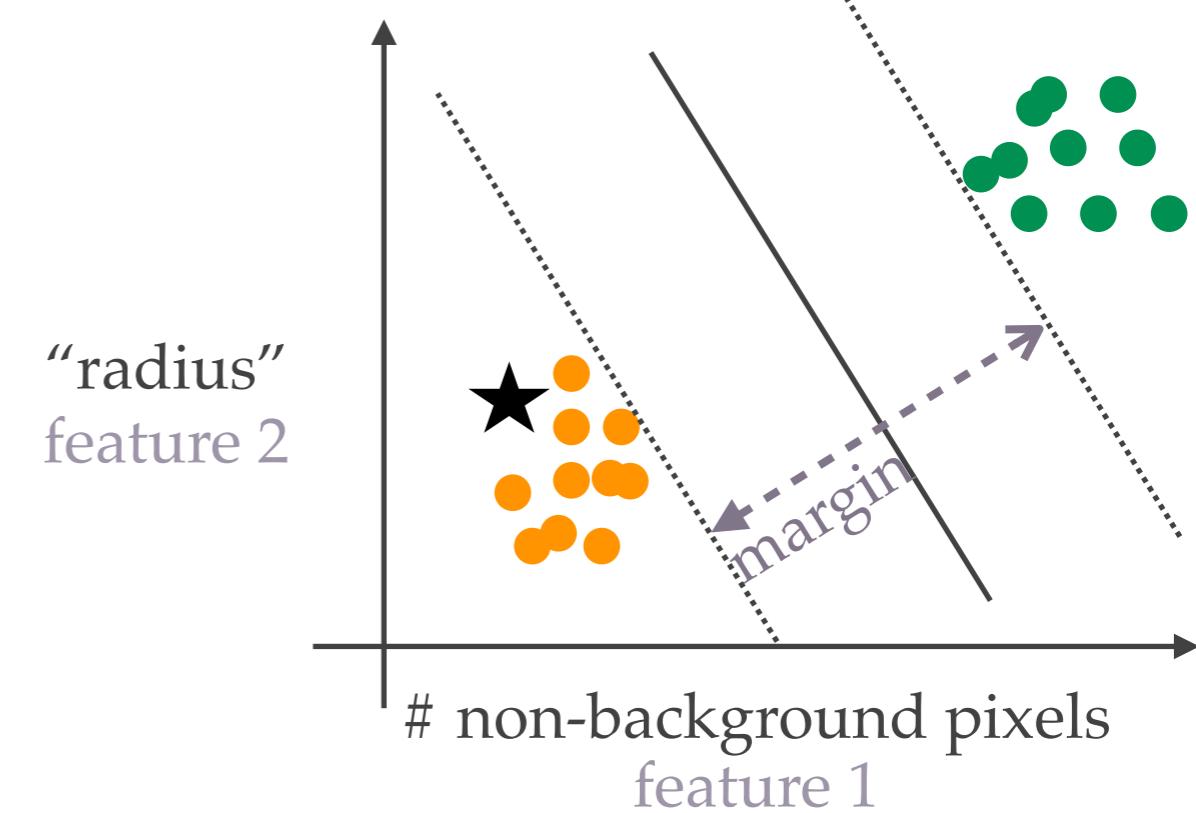
Phase: Testing



current sample



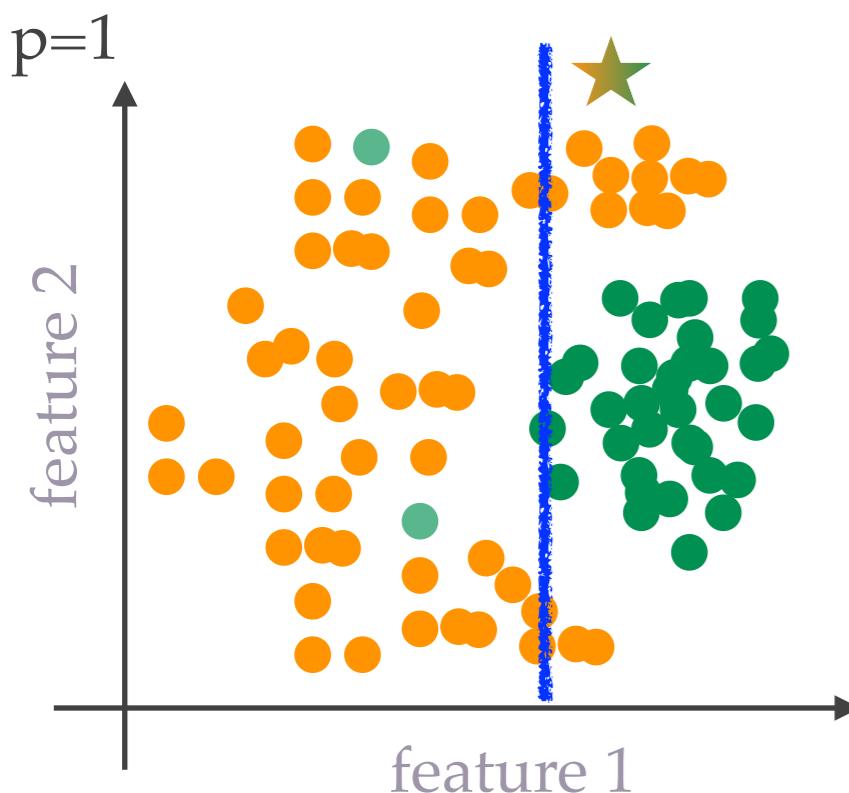
Orange!

Nearest Neighbor**k-Nearest Neighbors (kNN)****Linear Classifier****Support Vector Machine (SVM)
with a linear kernel**

What should we pick as our model?

If we use a p^{th} order polynomial to separate the classes, what happens when $p=1$? $p=2$? large p ?

incorrectly classified as green

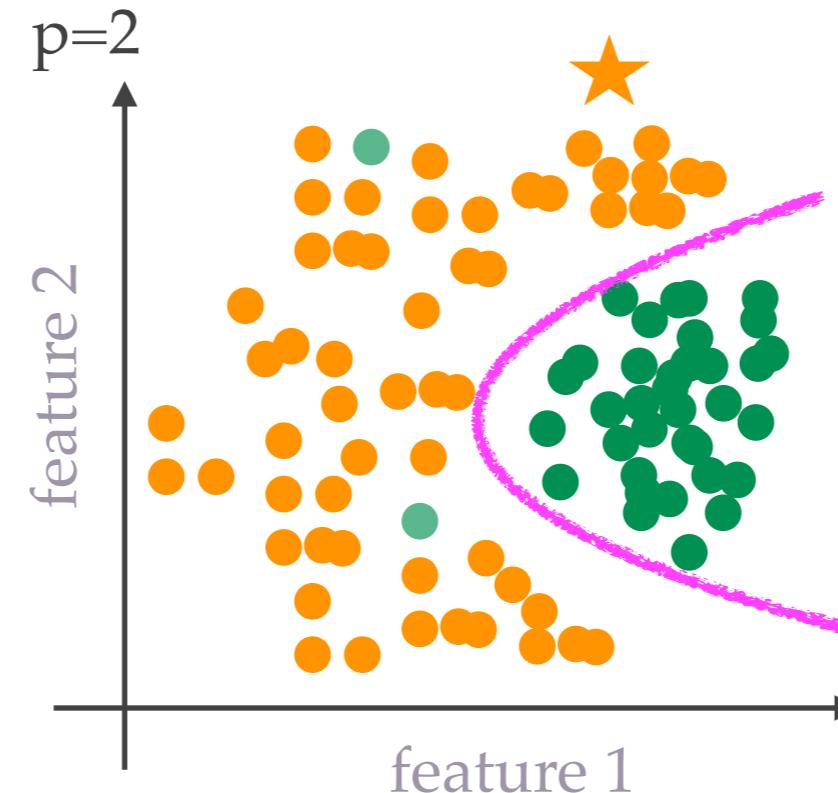


Underfit

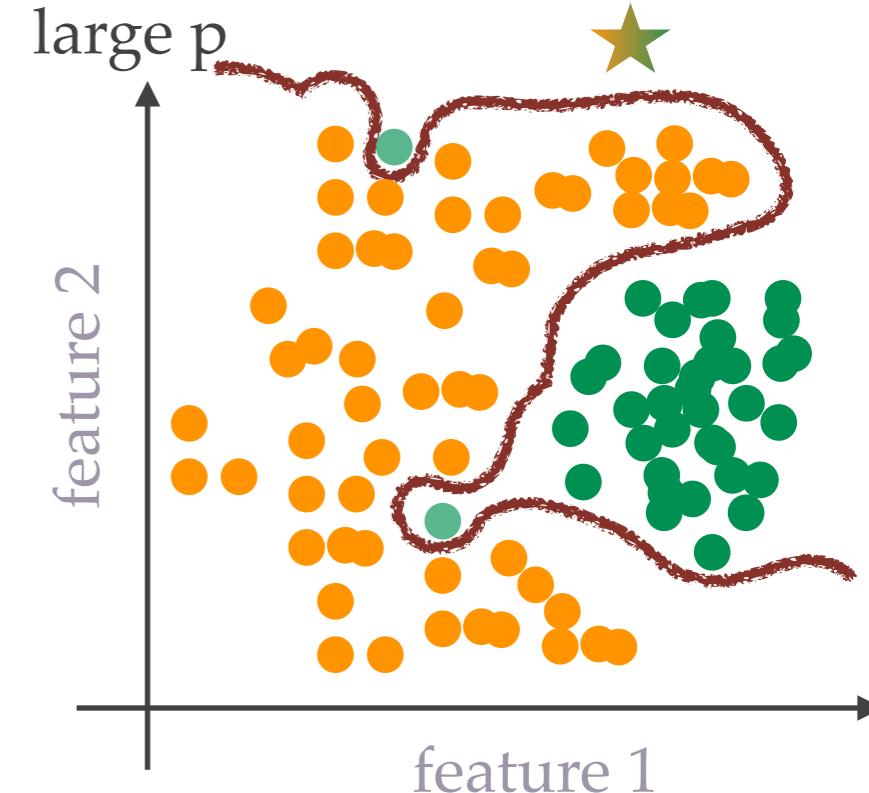
Doesn't perform well for both seen
(**training**) and unseen (**test**) data

- High training error
- High test error

correctly classified as orange



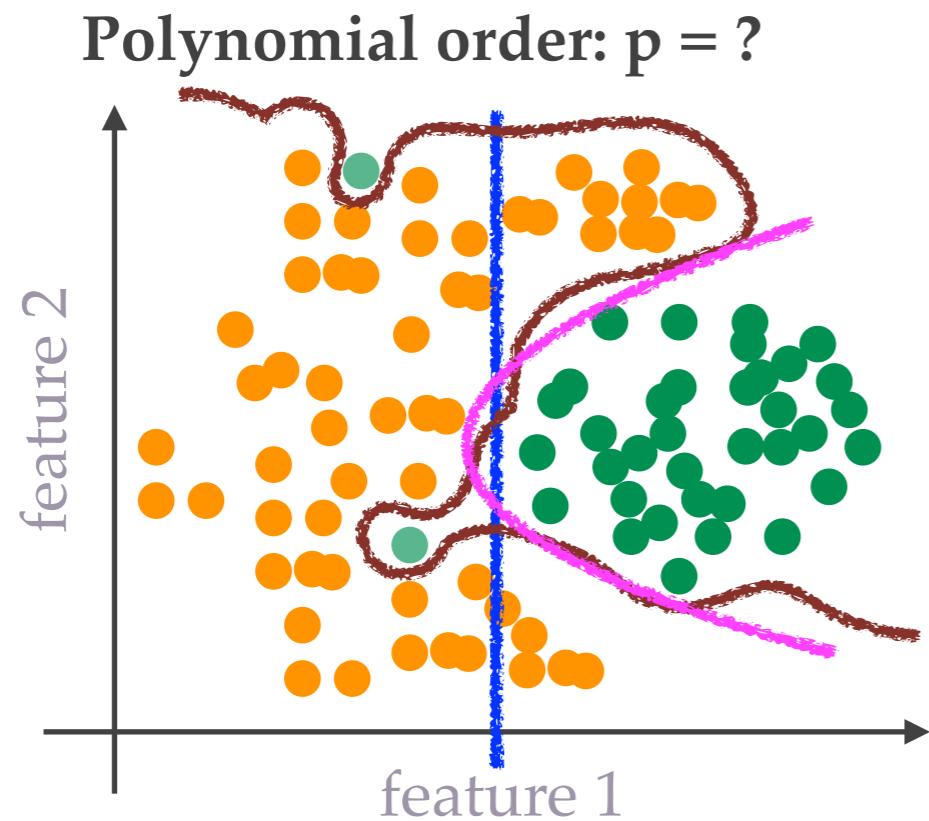
incorrectly classified as green



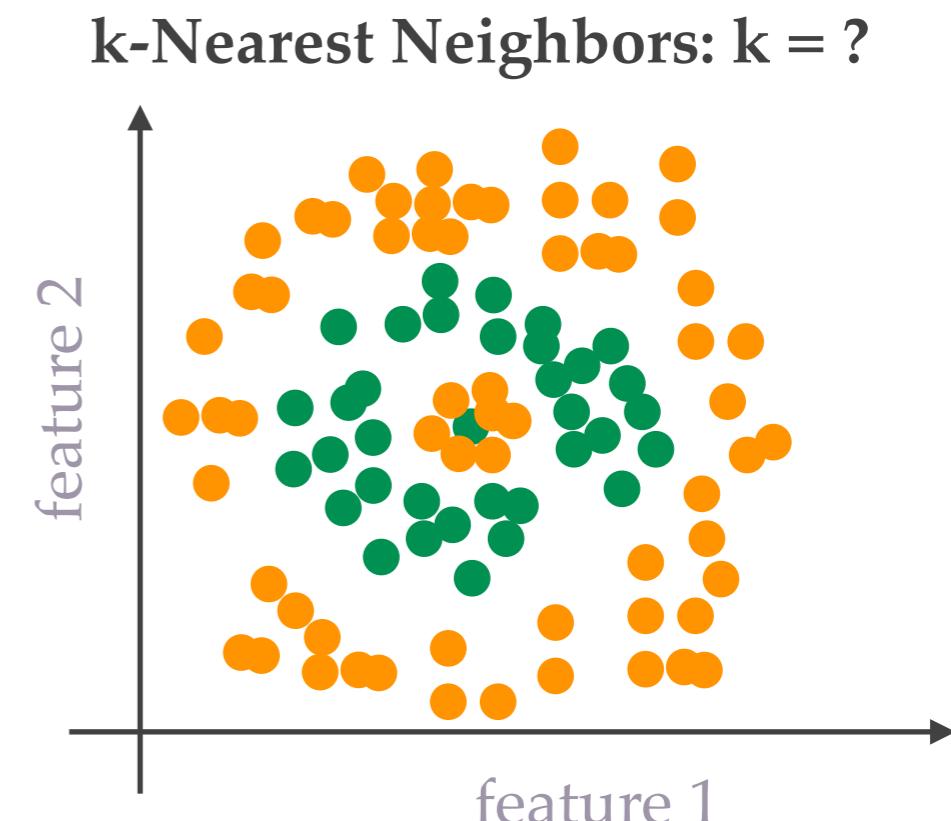
Overfit

- Perform well for training data, but not test data
- Low training error
 - High test error

What should we pick as our model?



If we use a p^{th} order polynomial to separate the classes, what happens when $p=1$? $p=2$? $p=1000$?



What happens when k is large (e.g., $k=1000$)?
What happens when $k=1$?

How do we pick a model (e.g., simple linear classifier, p^{th} order polynomial, k-nearest neighbors, logistic regressor)? How do we pick p and k ?

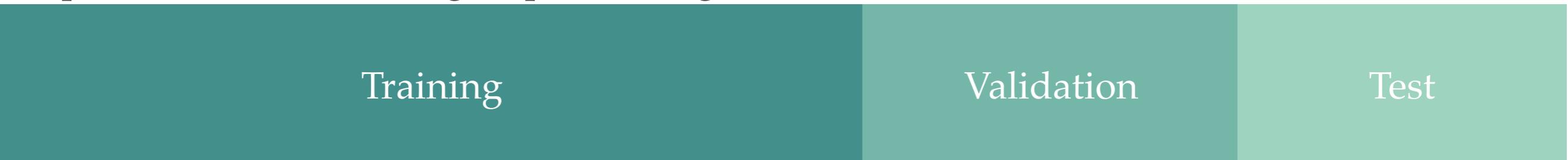
Validation data (in addition to training and test data, which we have seen earlier)!

Outline

- ❖ Introduction to Supervised Machine Learning
 - ❖ AI vs ML vs DL
 - ❖ Traditional Machine Learning
 - ❖ Training, Validation, and Test Data
 - ❖ Overfitting and Underfitting
- ❖ Introduction to Supervised Deep Learning
 - ❖ Traditional ML vs Deep Learning
 - ❖ Artificial Neuron and Neural Network
 - ❖ Supervised Learning
- ❖ Deep Learning as a Function Approximator
- ❖ Deep Learning Components
 - ❖ Fully Connected Layers
 - ❖ Activation Functions
 - ❖ Optimization

Simplified Pipeline & Training, Validation, and Test Data

1. Given a task, pick a model linear vs nonlinear? Polynomial of order $p = 3$ vs $p = 21$?
2. Split the data into three groups: training, validation, and test



3. Optimize the selected model using the training data
 4. Evaluate the trained model using the validation data
 5. Pick more models, optimize the models, and evaluate the trained models on the validation data
 6. Use the model with the lowest validation error as your final model
 7. Evaluate the final model on the test data
- Ex. Try a linear model, 8 kNNs with different k , and 20 polynomials with $p = 2, 3, \dots, 21$
- Ex. Pick the model with the smallest validation error (out of the 30 models)

Example: Training, Validation, and Test Data

Scenario: A time-limited, two-choice materials science exam will be held next month.

We have come up with four strategies (i.e., 4 models) for acing the upcoming exam

- AI #1: Always pick choice 1 choice 1 choice 2
- AI #2: Pick choice 1 for the first half of the exam and learn what to answer for the rest
- AI #3: Pick choice 2 for the first half of the exam and learn what to answer for the rest
- AI #4: Learn what to answer for all questions

Training	Validation	Test
----------	------------	------

Practice problems and previous exams from the last decade (except 2018-2021)

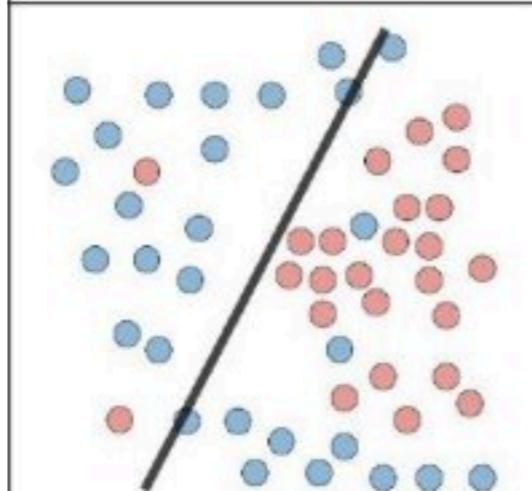
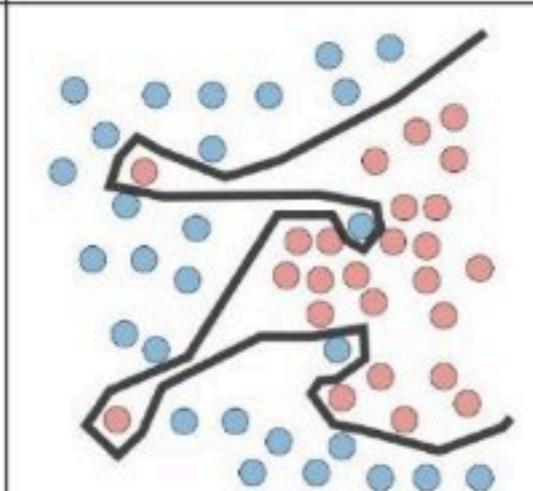
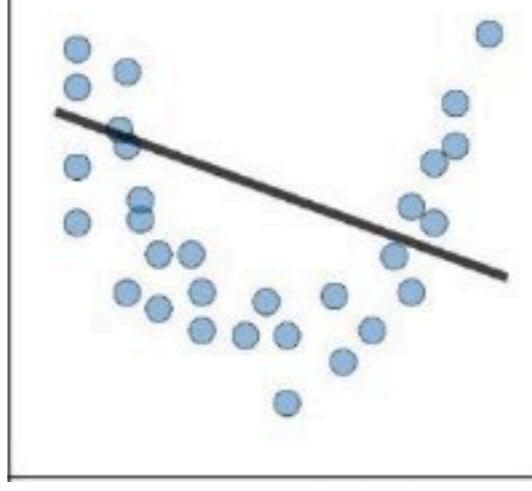
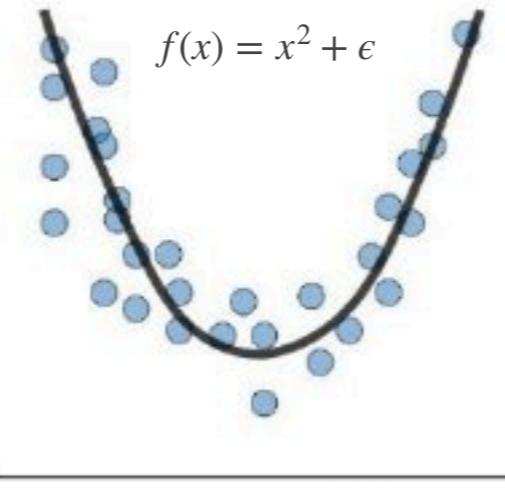
2018-2021 exams

The upcoming exam (unseen)

Steps

1. Use the **training problems and exams** to teach all the AI models.
2. Use the **validation exams** to see how the trained models perform.
3. Select the AI model that achieves the highest score on the validation exams to take the upcoming exam (**test exam**) for you

Overfitting and Underfitting

	Underfitting	Overfitting	
Classification			
Regression			
Likely symptoms	<ul style="list-style-type: none">• High training error• Training error close to test error	<ul style="list-style-type: none">• Training error slightly lower than test error	<ul style="list-style-type: none">• Low training error• Test error much higher than training error

Outline

- ❖ Introduction to Supervised Machine Learning
 - ❖ AI vs ML vs DL
 - ❖ Traditional Machine Learning
 - ❖ Training, Validation, and Test Data
 - ❖ Overfitting and Underfitting
- ❖ Introduction to Supervised Deep Learning
 - ❖ Traditional ML vs Deep Learning
 - ❖ Artificial Neuron and Neural Network
 - ❖ Supervised Learning
- ❖ Deep Learning as a Function Approximator
- ❖ Deep Learning Components
 - ❖ Fully Connected Layers
 - ❖ Activation Functions
 - ❖ Optimization

Watermelon or Orange?

Traditional Machine Learning

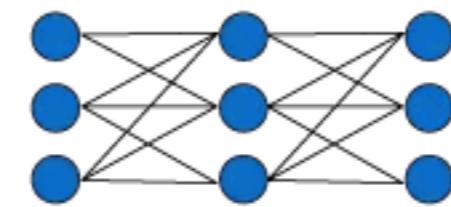
Input image



hand-engineered/
hand-crafted features



random forest, SVM,
kNN, XGBoost, LGBM



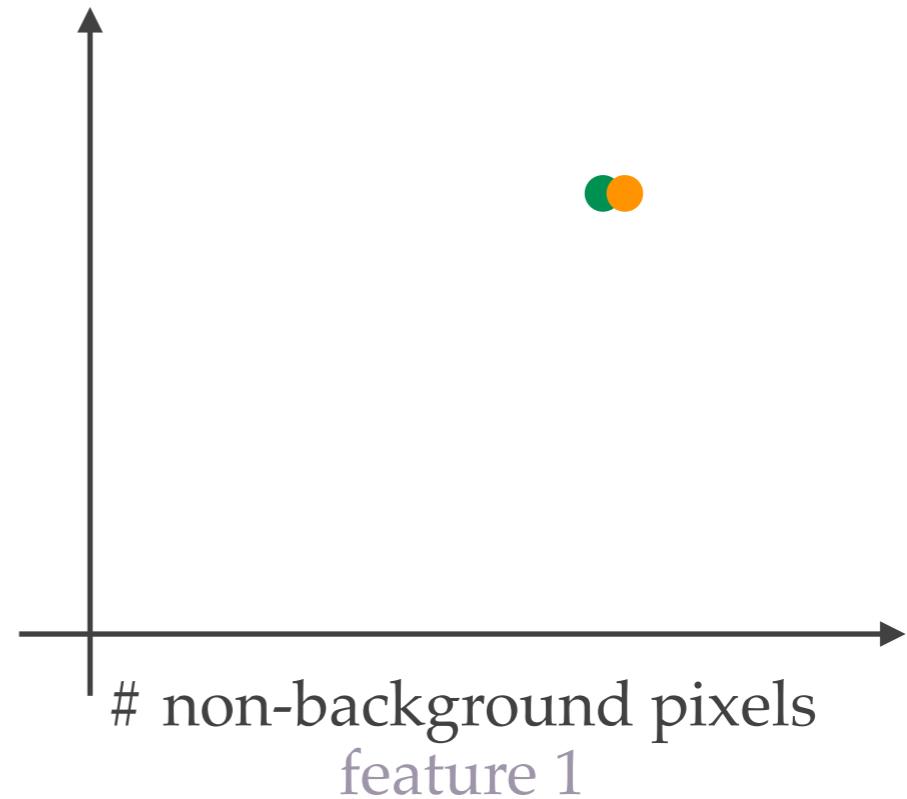
Output

watermelon

Problem with feature engineering?



“radius”
feature 2

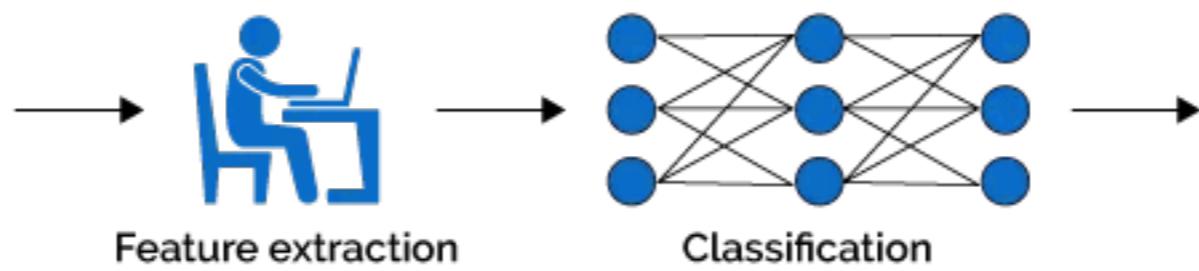


Some parts taken from: Azua Tech

Watermelon or Orange?

Traditional Machine Learning

Input image

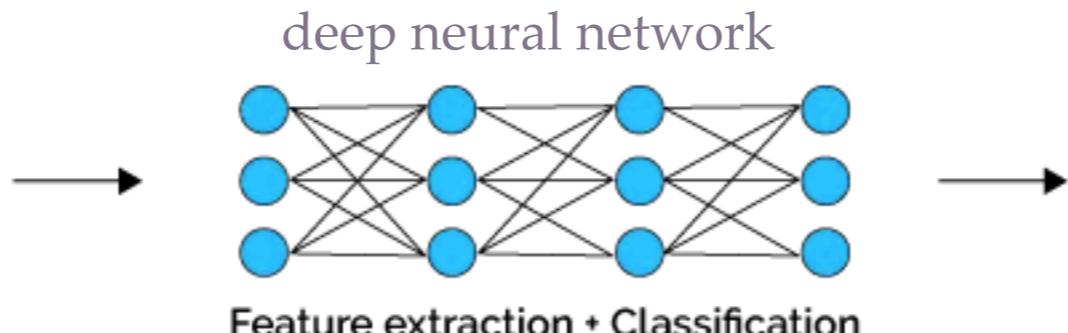


Output

watermelon

Deep Learning

Input image



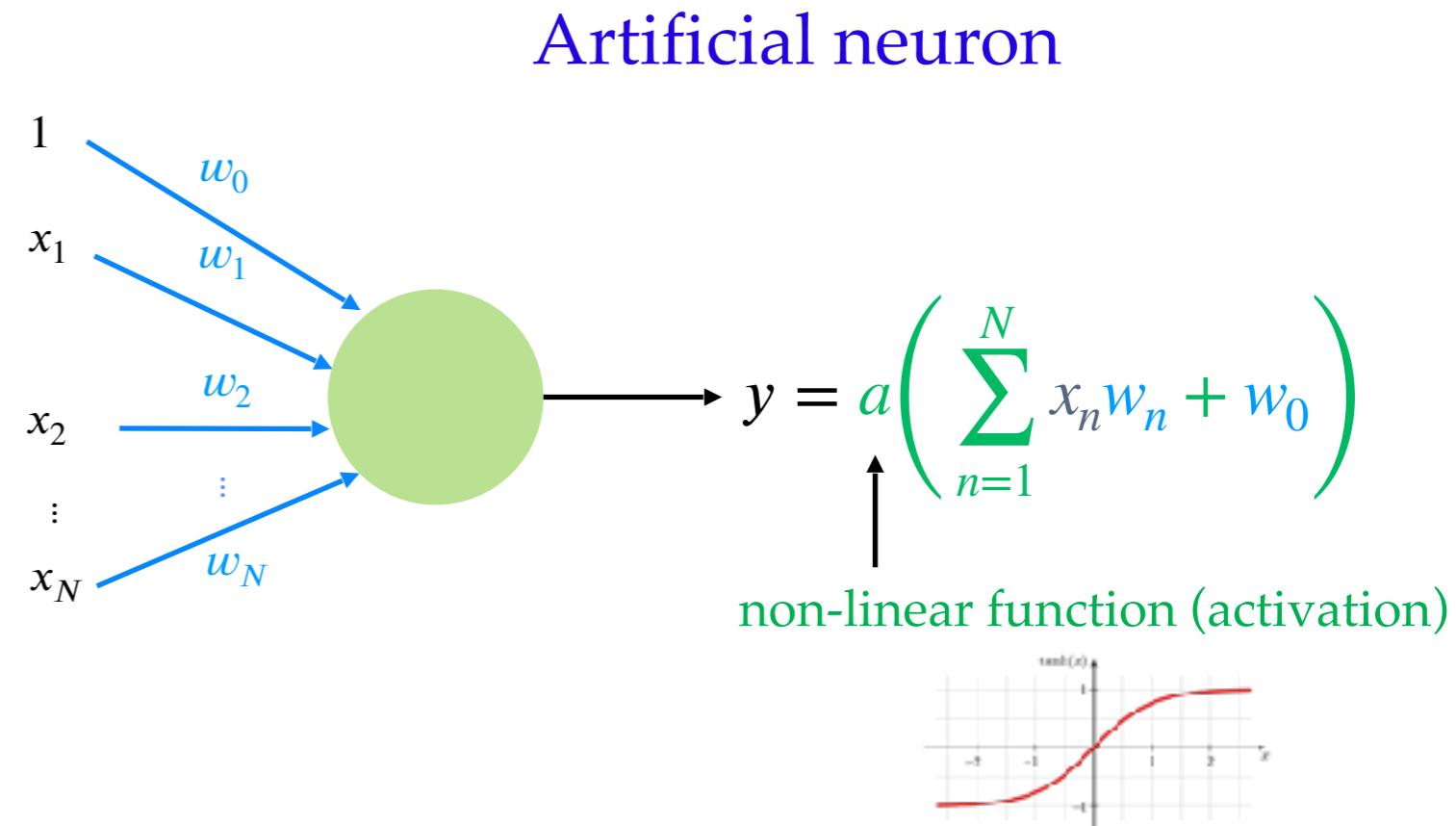
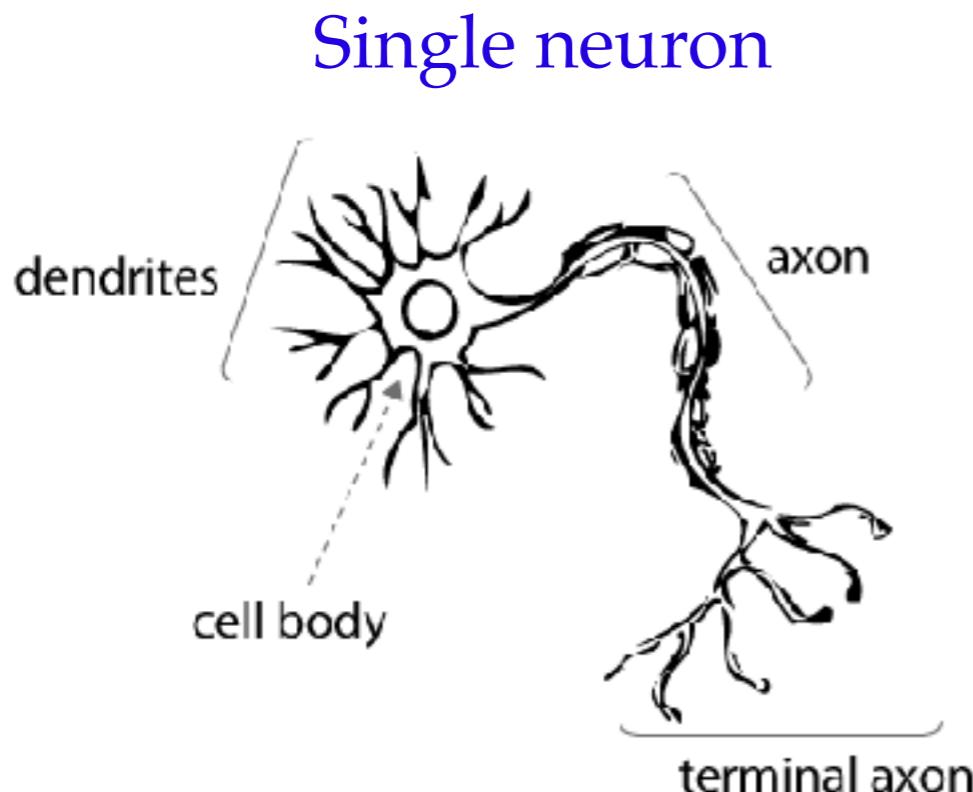
Output

watermelon

Some parts taken from: Azua Tech

Deep Learning

- ❖ Deep learning is a subfield of machine learning that stems from artificial neural networks (ANN)



If we set all the weights to 0, then $y = 0$.

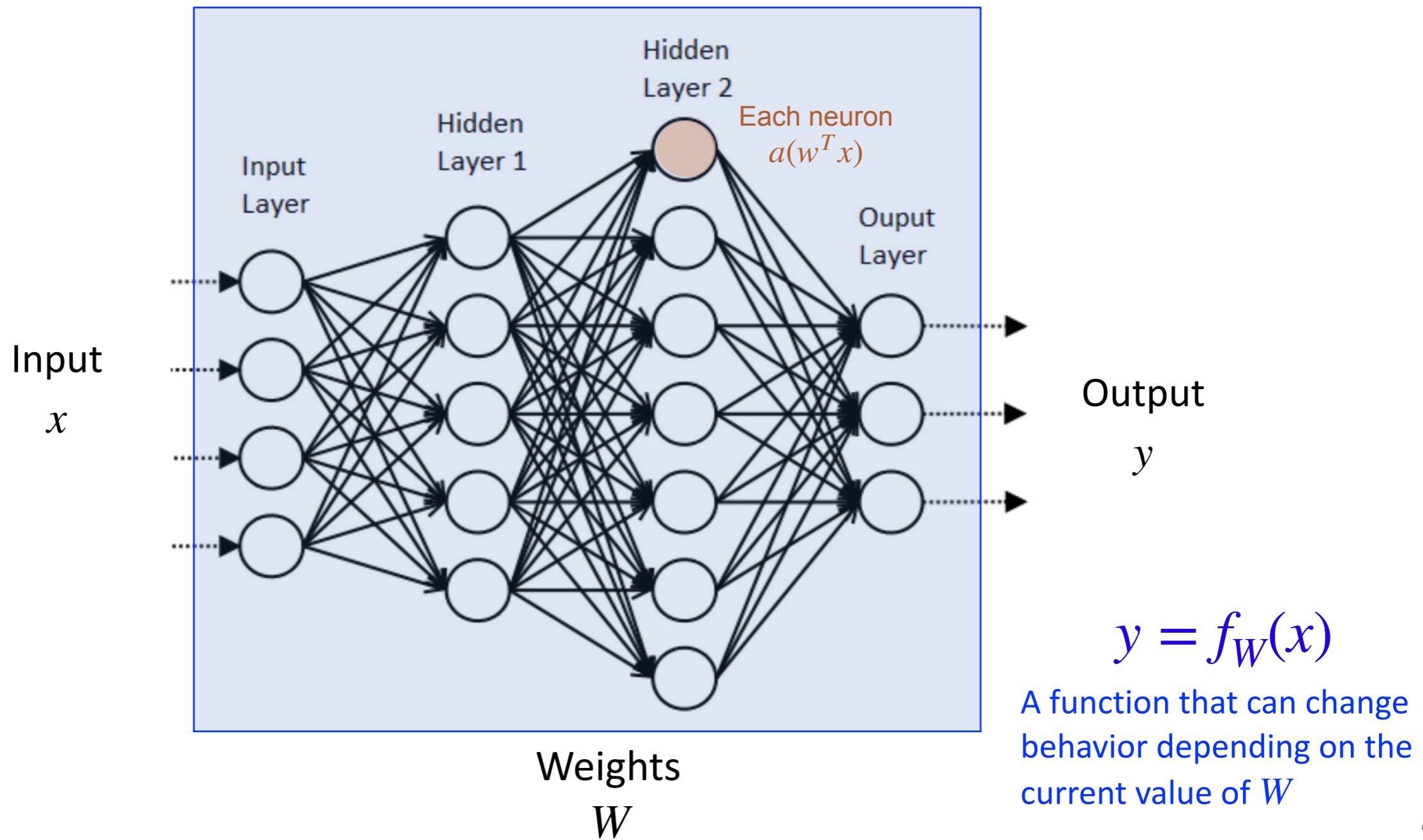
If we set $w_1 = 1$ and the rest to 0, then $y = a(x_1)$.

If we set $w_0 = 0$ and the rest to 1, then $y = a(x_1 + x_2 + \dots + x_N)$.

Different weights give rise
to different behavior

Deep Learning

- An artificial neural network with **many hidden layers** is called a **deep** artificial neural network (ANN)

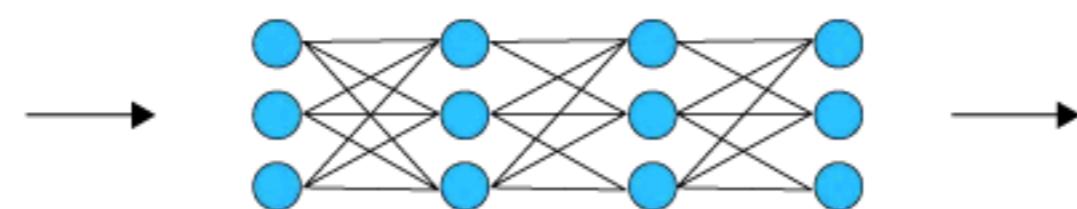


Deep Learning

- ❖ Think of a deep NN as a universal function approximator

Image classification

Input image

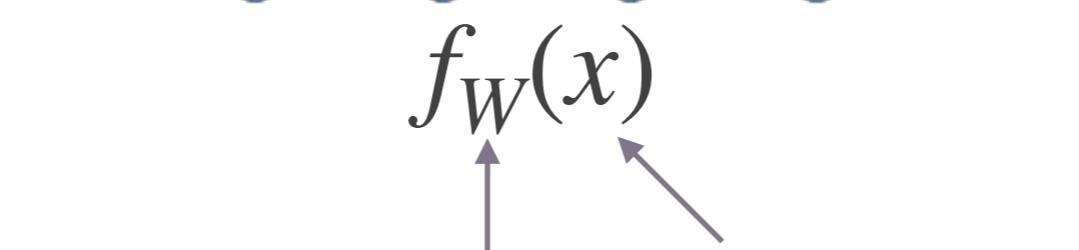


Output: Label

1 (watermelon)

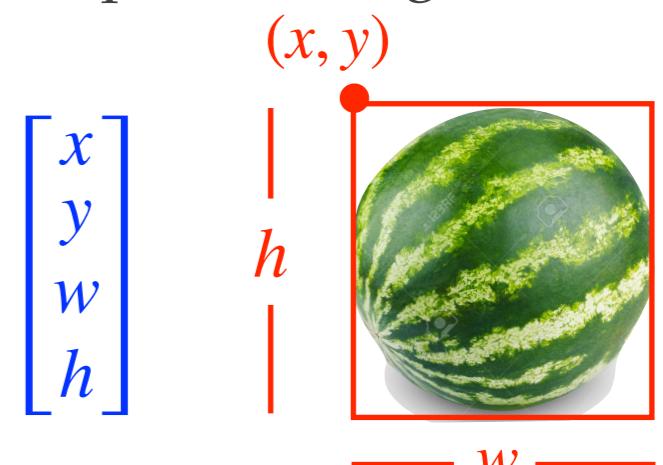
Object localization

Input image



Adjustable
Input
weight

Output: rectangular box





Fully connected layer
(Dense)

Convolutional layer
Conv1D, 2D, 3D, ...
separable Conv

Optimizer
SGD
Adam
RMSprop

Evaluation metric
accuracy
F1-score
AUC
confusion matrix

Loss function
categorical crossentropy
binary crossentropy
mean squared error
mean absolute error

Deep Learning

Regularization
Dropout
Data augmentation
 l_1, l_2 regularizations

Pooling layer
max-pooling
average-pooling

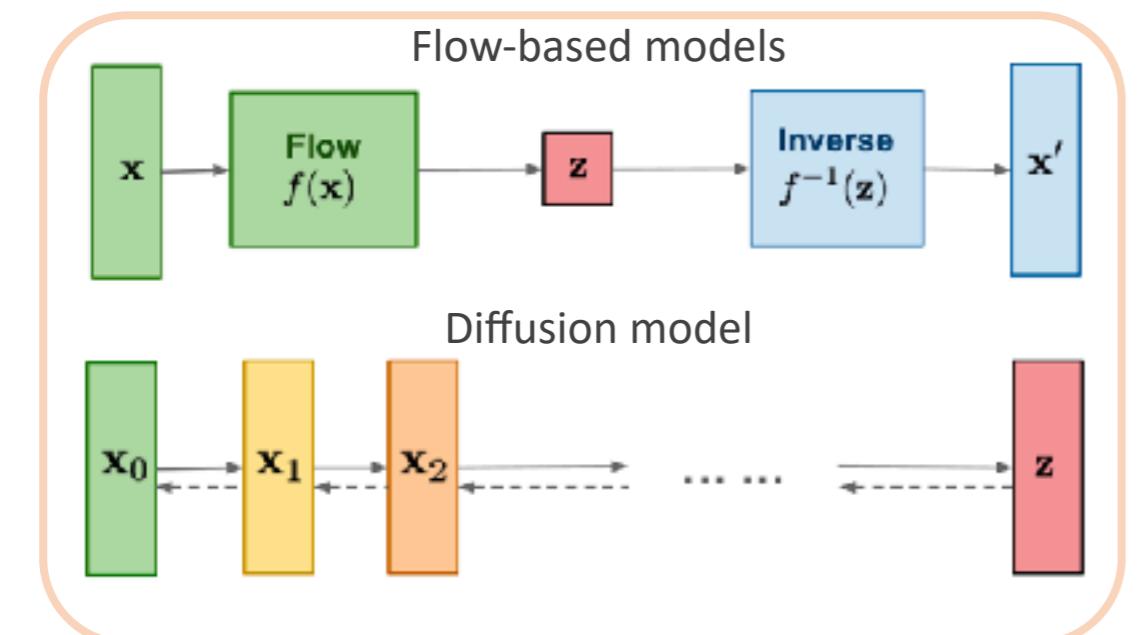
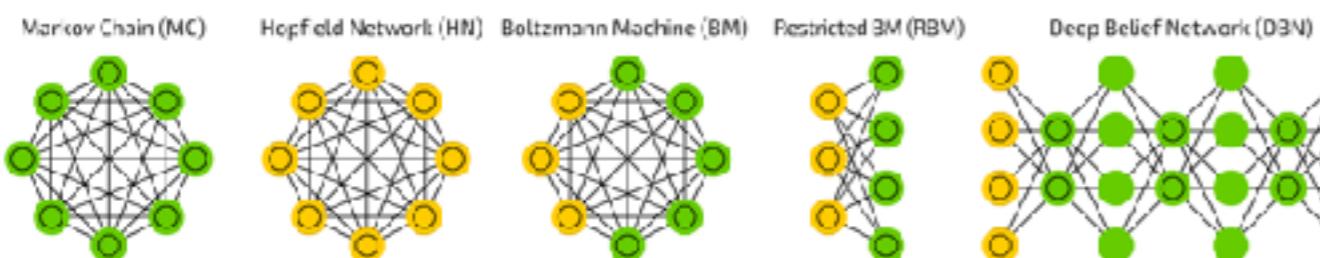
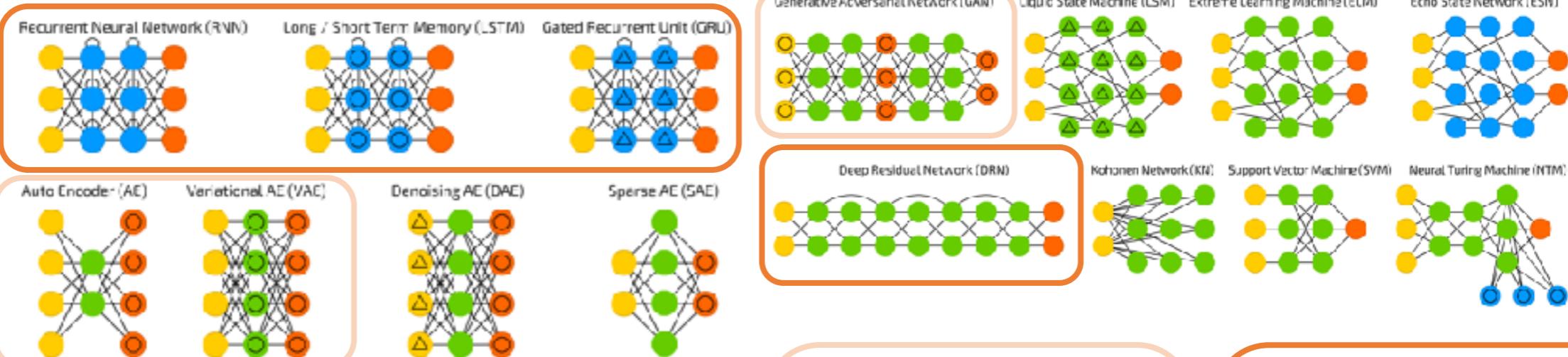
Activation function
sigmoid
softmax
ESP (swish)
ReLU

- ❖ **Combine basic components to build a neural network**
 - More components → “More” representative power

- BackFed Input Cell
- Input Cell
- △ Noisy Input Cell
- Hidden Cell
- Probabilistic Hidden Cell
- △ Spiking Hidden Cell
- Output Cell
- Match Input Output Cell
- Recurrent Cell
- Memory Cell
- △ Different Memory Cell
- Kernel
- Convolution or Pool.

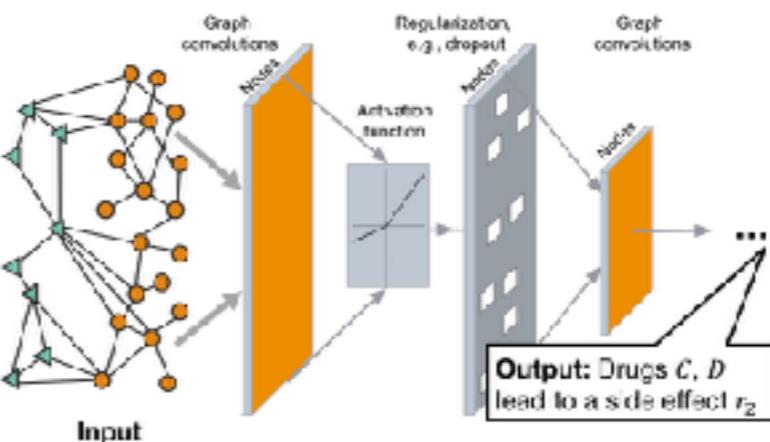
Neural Networks

©2016 Fodor van Veen - asimovinstitute.org

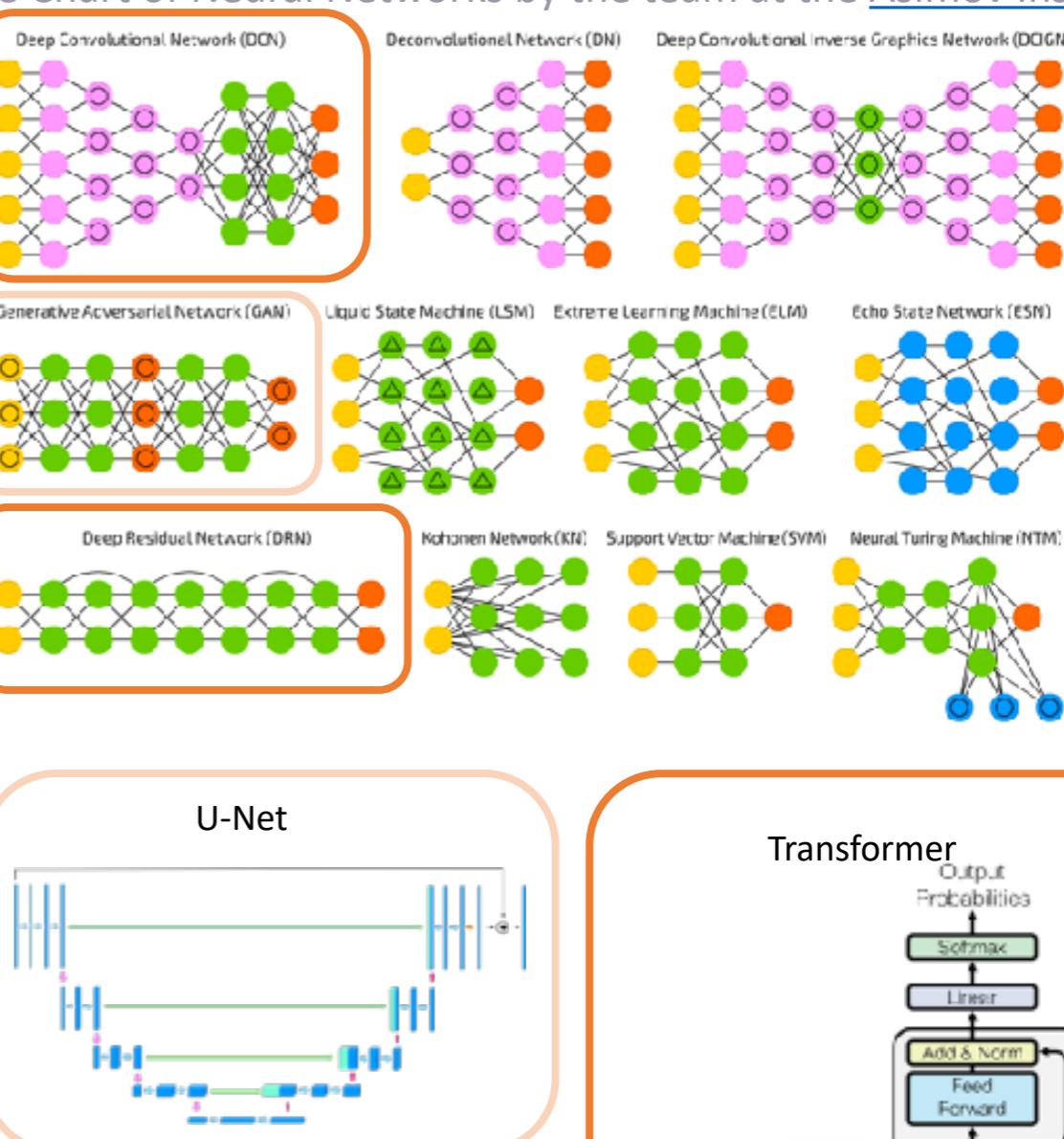


What are Diffusion Models?

Graph convolutional neural network



[Graph Neural Networks for Multirelational Link Prediction](#)



Vaswani, Ashish, et al. "Attention is all you need." *Advances in neural information processing systems* 30 (2017).

ImageNet

IMAGENET

1000 classes, 14,197,122 images

Figure 4. (Left) Eight ILSVRC-2010 test images and the five labels considered most probable by our model. The correct label is written under each image, and the probability assigned to the correct label is also shown with a red bar (if it happens to be in the top 5). (Right) Five ILSVRC-2010 test images in the first column. The remaining columns show the six training images that produce feature vectors in the last hidden layer with the smallest Euclidean distance from the feature vector for the test image.

				
mite	container ship	motor scooter	leopard	
mite	container ship	motor scooter	leopard	
black widow	lifeboat	go-kart	jaguar	
cockroach	amphibian	moped	cheetah	
tick	fireboat	bumper car	snow leopard	
starfish	drilling platform	golfcart	Egyptian cat	
				
grille	mushroom	cherry	Madagascar cat	
convertible	agaric	dalmatian	squirrel monkey	
grille	mushroom	grape	spider monkey	
pickup	jelly fungus	elderberry	titi	
beach wagon	gill fungus	ffordshire bullterrier	Indri	
fire engine	dead-man's-fingers	currant	howler monkey	

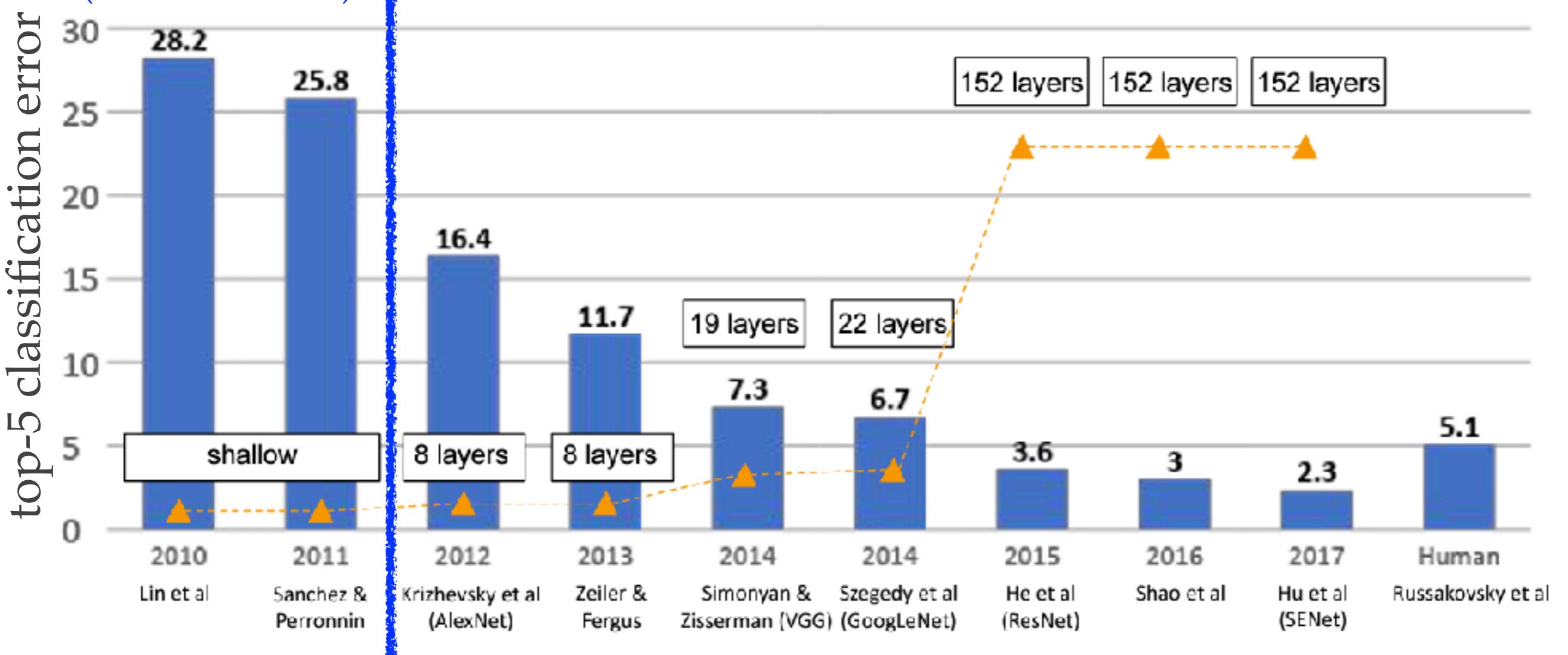


ImageNet

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

The SVM era
(traditional ML)

The deep learning era



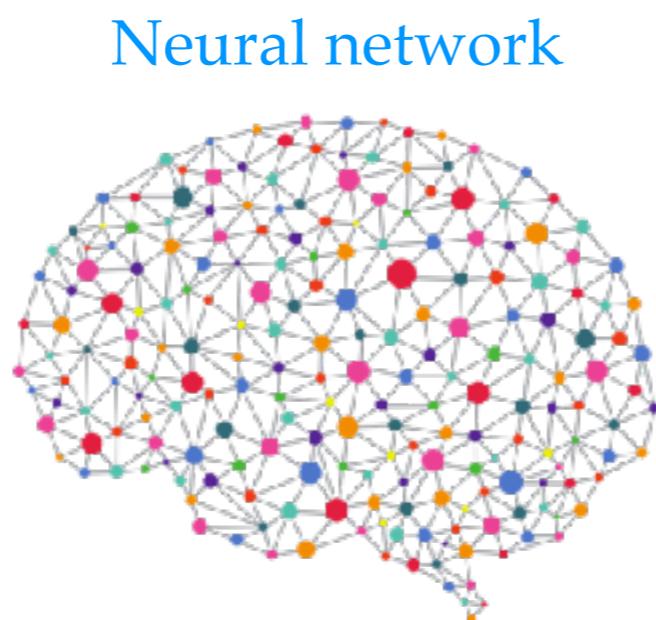
methods

Outline

- ❖ Introduction to Supervised Machine Learning
 - ❖ AI vs ML vs DL
 - ❖ Traditional Machine Learning
 - ❖ Training, Validation, and Test Data
 - ❖ Overfitting and Underfitting
- ❖ Introduction to Supervised Deep Learning
 - ❖ Traditional ML vs Deep Learning
 - ❖ Artificial Neuron and Neural Network
 - ❖ Supervised Learning
- ❖ Deep Learning as a Function Approximator
- ❖ Deep Learning Components
 - ❖ Fully Connected Layers
 - ❖ Activation Functions
 - ❖ Optimization

Supervised Model Training

Step 1: Create a neural network with some initial weight f_{W_0}



Training iteration 0
 f_{W_0}

Supervised Model Training

Step 2: Prepare a dataset which is a collection of input-output pairs

Prepared inputs

x



Prepared outputs
(true labels)

y

1

Neural network



0

Training iteration 0

f_{W_0}

0



1

orange: 0
watermelon: 1

Supervised Model Training

Step 3: Pass the prepared inputs to the network

Prepared inputs

x



Estimated outputs

$$\hat{y} = f_{W_0}(x)$$

0.6

Prepared outputs
(true labels)

y

1



Neural network



Training iteration 0
 f_{W_0}

0.6

0

0.5

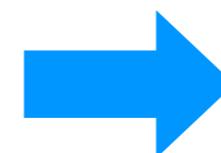
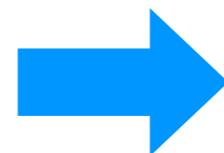
0

0.5

1

orange: 0

watermelon: 1



Supervised Model Training

Step 4: Compare \hat{y} to y and modify the weights of the neural network to make \hat{y} approach y using the backpropagation algorithm

Prepared inputs



x

A simple loss function: $L(y, \hat{y}) = \sum_{i=1}^N (\hat{y}_i - y_i)^2$

$$L(y, \hat{y}) = (0.6 - 1)^2 + (0.6 - 0)^2 + (0.5 - 0)^2 + (0.5 - 1)^2 = 1.02$$

Estimated outputs
 $\hat{y} = f_{W_0}(x)$

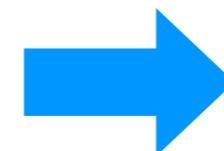
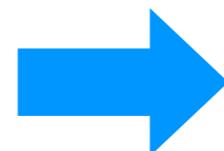
Prepared outputs
(true labels)

y

0.6

1

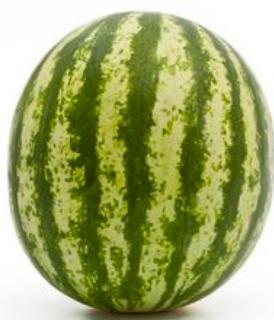
Neural network



Training iteration 0

f_{W_0}

Training loss



Iteration

0.6

0

0.5

0

0.5

orange: 0
watermelon: 1

Supervised Model Training

Repeat steps 3 and 4 to continuously improve the weights of the neural network

Prepared inputs

x



A simple loss function: $L(y, \hat{y}) = \sum_{i=1}^N (\hat{y}_i - y_i)^2$

$$L(y, \hat{y}) = (0.6 - 1)^2 + (0.4 - 0)^2 + (0.4 - 0)^2 + (0.6 - 1)^2 = 0.64$$

Estimated outputs
 $\hat{y} = f_{W_1}(x)$

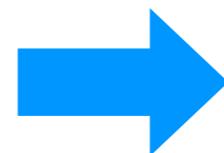
0.6

1

Prepared outputs
(true labels)

y

1



Neural network



0.4

0

0.4

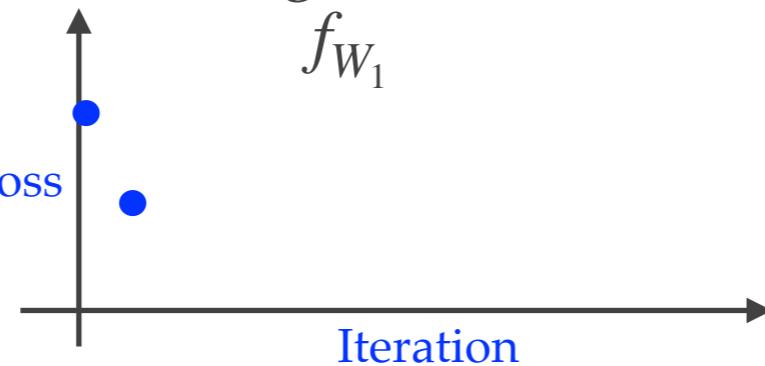
0



Training iteration 1

f_{W_1}

Training loss



0.6

1

orange: 0

watermelon: 1

Supervised Model Training

Repeat steps 3 and 4 to continuously improve the weights of the neural network

Prepared inputs

x



A simple loss function: $L(y, \hat{y}) = \sum_{i=1}^N (\hat{y}_i - y_i)^2$

$$L(y, \hat{y}) = (0.7 - 1)^2 + (0.35 - 0)^2 + (0.4 - 0)^2 + (0.6 - 1)^2 = 0.5325$$

Estimated outputs
 $\hat{y} = f_{W_2}(x)$

Prepared outputs
(true labels)

y

0.7

1



Neural network



0.35

0

0.4

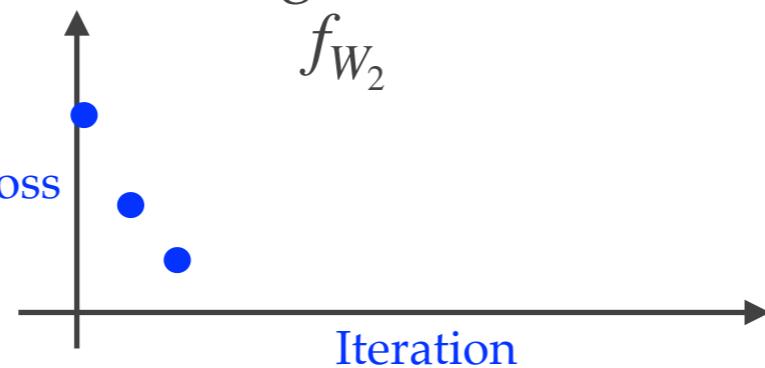
0



Training iteration 2

f_{W_2}

Training loss



0.6

orange: 0
watermelon: 1

Supervised Model Training

Repeat steps 3 and 4 to continuously improve the weights of the neural network

Prepared inputs
 x



A simple loss function: $L(y, \hat{y}) = \sum_{i=1}^N (\hat{y}_i - y_i)^2$

$$L(y, \hat{y}) = (0.8 - 1)^2 + (0.15 - 0)^2 + (0.1 - 0)^2 + (0.9 - 1)^2 = 0.0825$$

Estimated outputs
 $\hat{y} = f_{W_{50}}(x)$

0.8

1

Prepared outputs
(true labels)
 y

1



Neural network



0.15

0

0.1

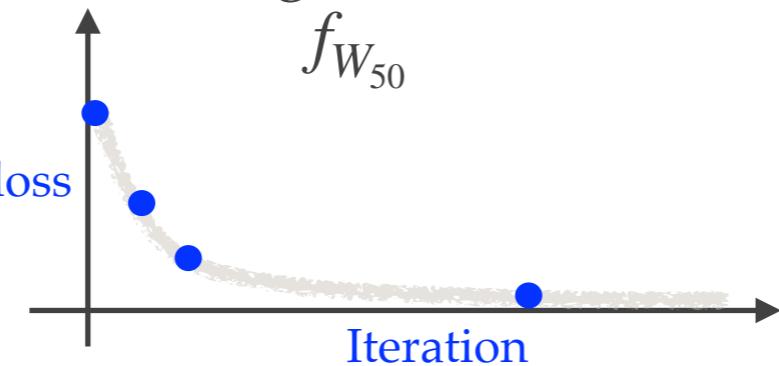
0



Training iteration 50

$f_{W_{50}}$

Training loss



orange: 0
watermelon: 1

Supervised Model Training

Repeat steps 3 and 4 to continuously improve the weights of the neural network

Prepared inputs
 x



A simple loss function: $L(y, \hat{y}) = \sum_{i=1}^N (\hat{y}_i - y_i)^2$

$$L(y, \hat{y}) = (0.99 - 1)^2 + (0.05 - 0)^2 + (0.01 - 0)^2 + (0.97 - 1)^2 = 0.0036$$

Estimated outputs
 $\hat{y} = f_{W_{1000}}(x)$

0.99

Prepared outputs
(true labels)
 y

1



Neural network



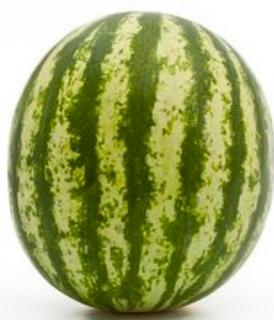
0.05



0

Training iteration 1000

$f_{W_{1000}}$



Training loss

Iteration

0.01

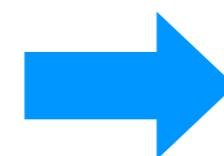
0.97

orange: 0

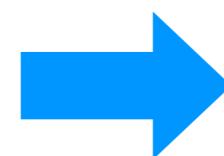
watermelon: 1

Test the Trained Model

Test image (unseen)



Trained neural network

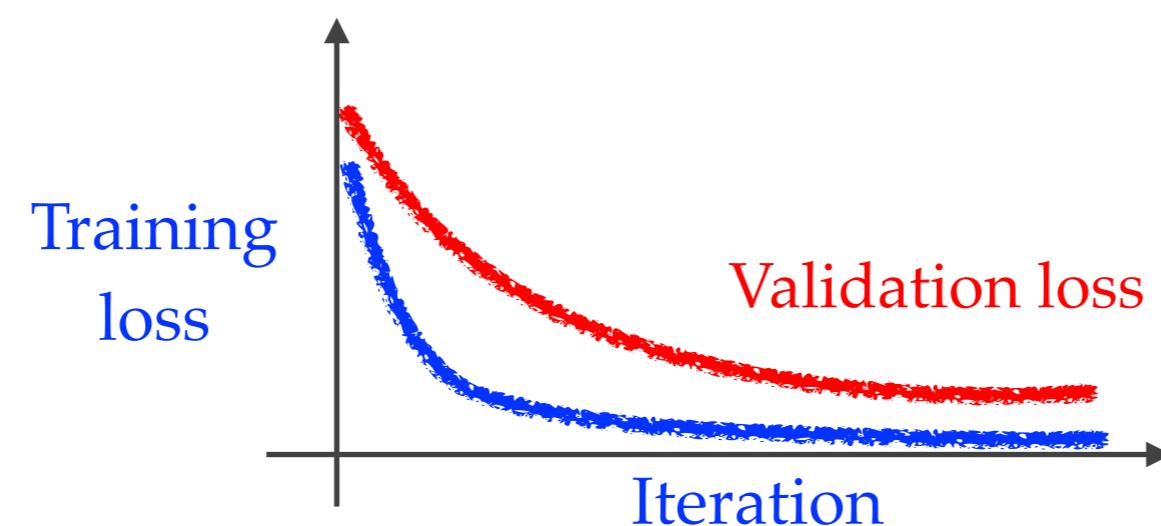


Predicted Label

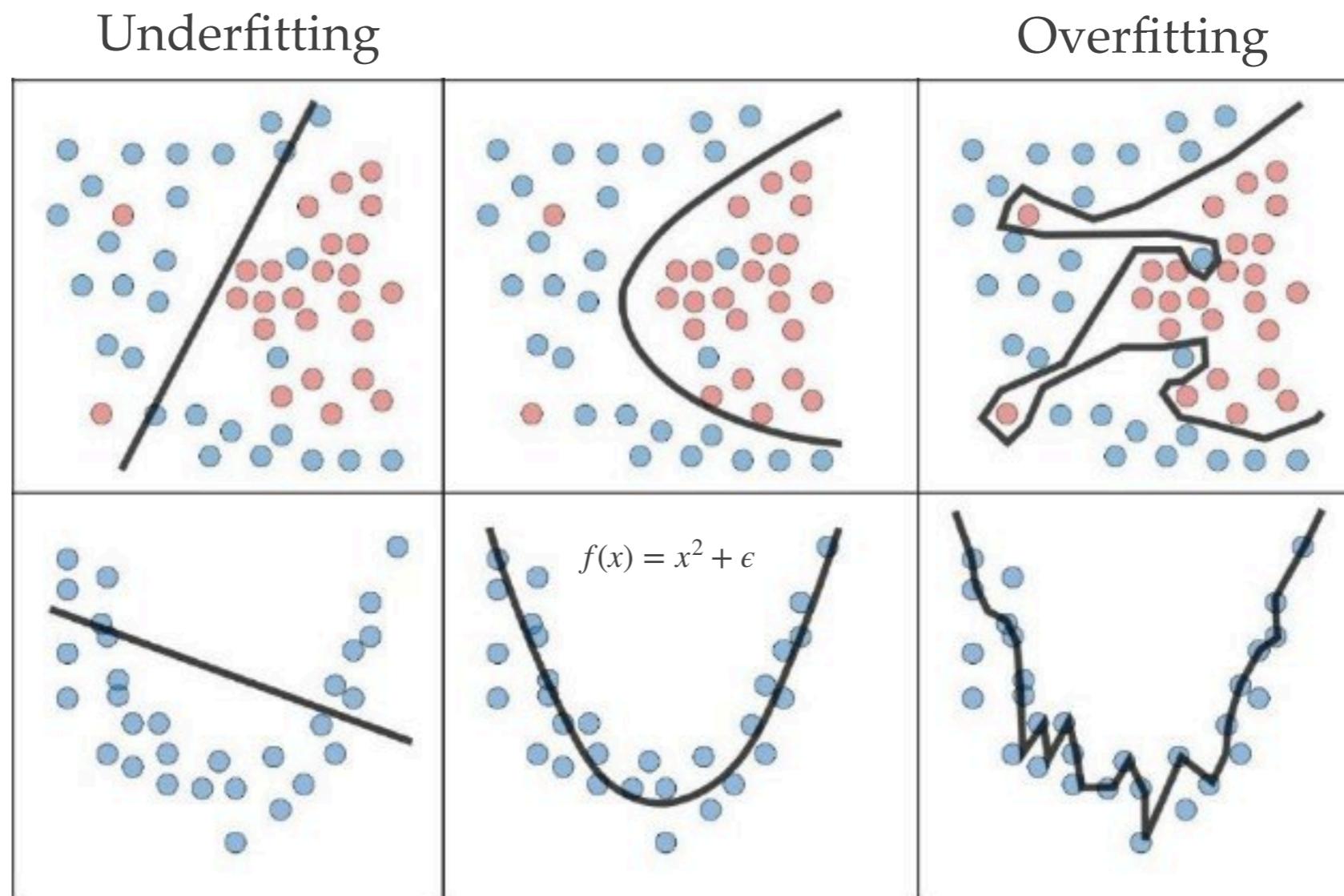
0.01

less than 0.5, so it is
likely orange

orange: 0
watermelon: 1



Classification



Likely symptoms

- High training error
- Training error close to test error
- Training error slightly lower than test error
- Low training error
- Test error much higher than training error

A way to check for the deep-learning-based methods (both classification and regression)

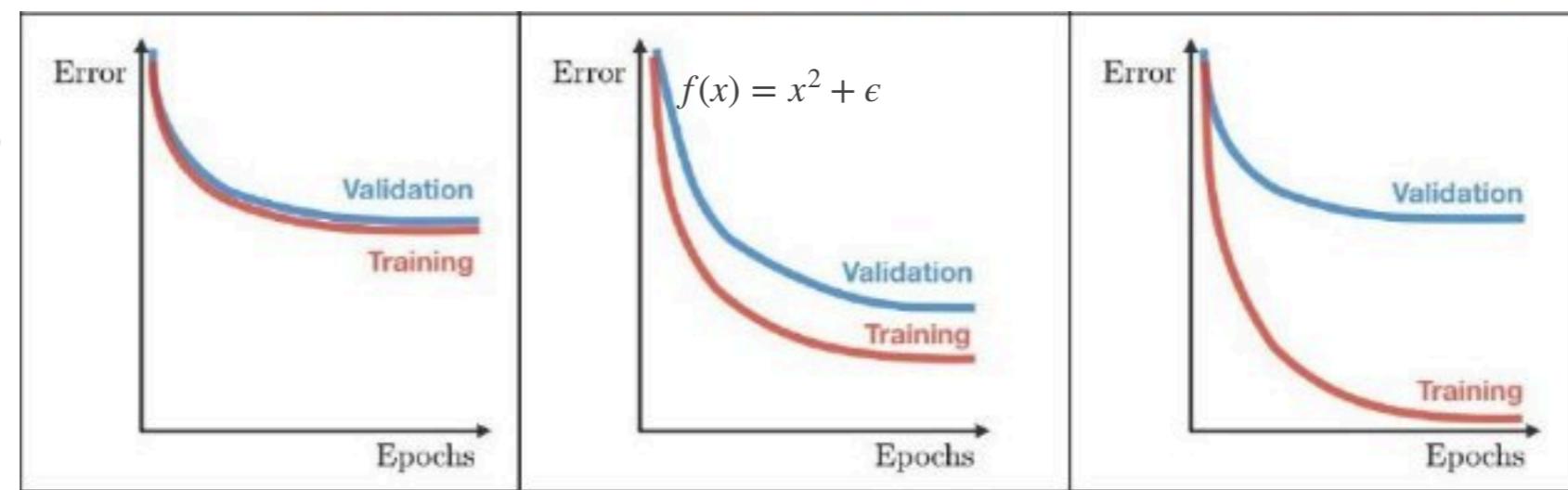
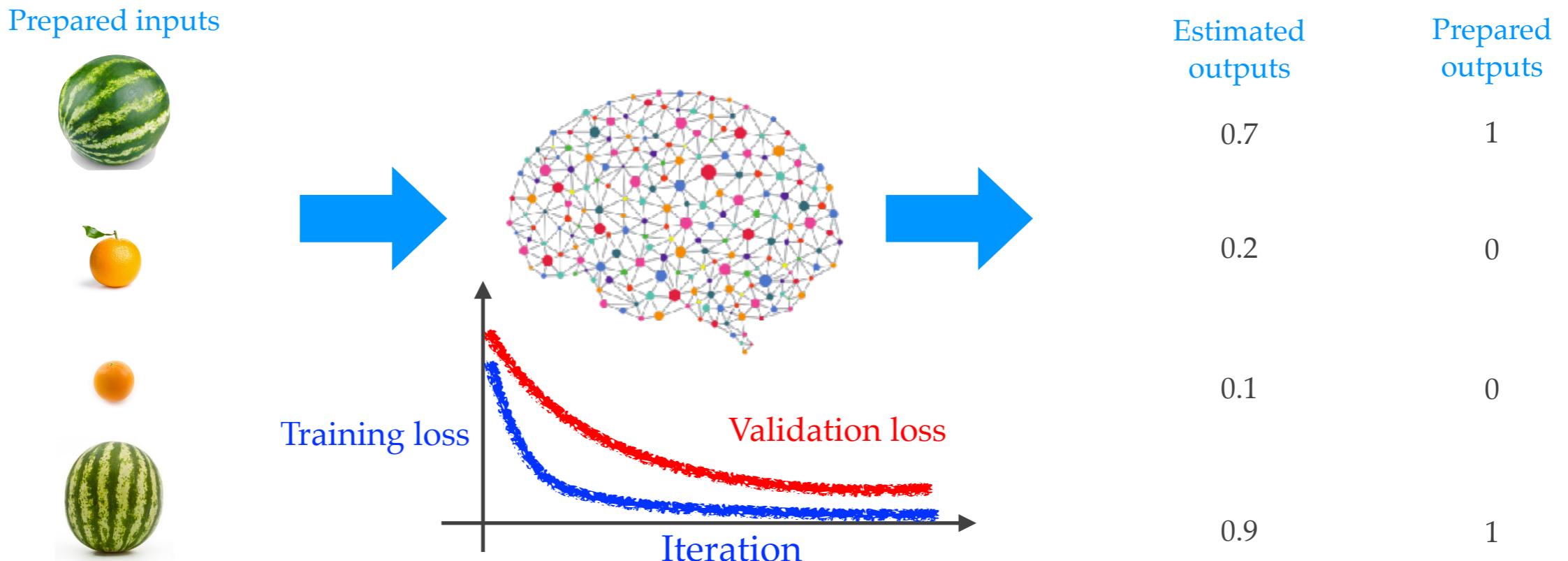


Image Classification Using Deep Learning

- ❖ Training phase: Optimize the weights of a deep neural network



- ❖ Test phase: Perform prediction using the trained neural network



Outline

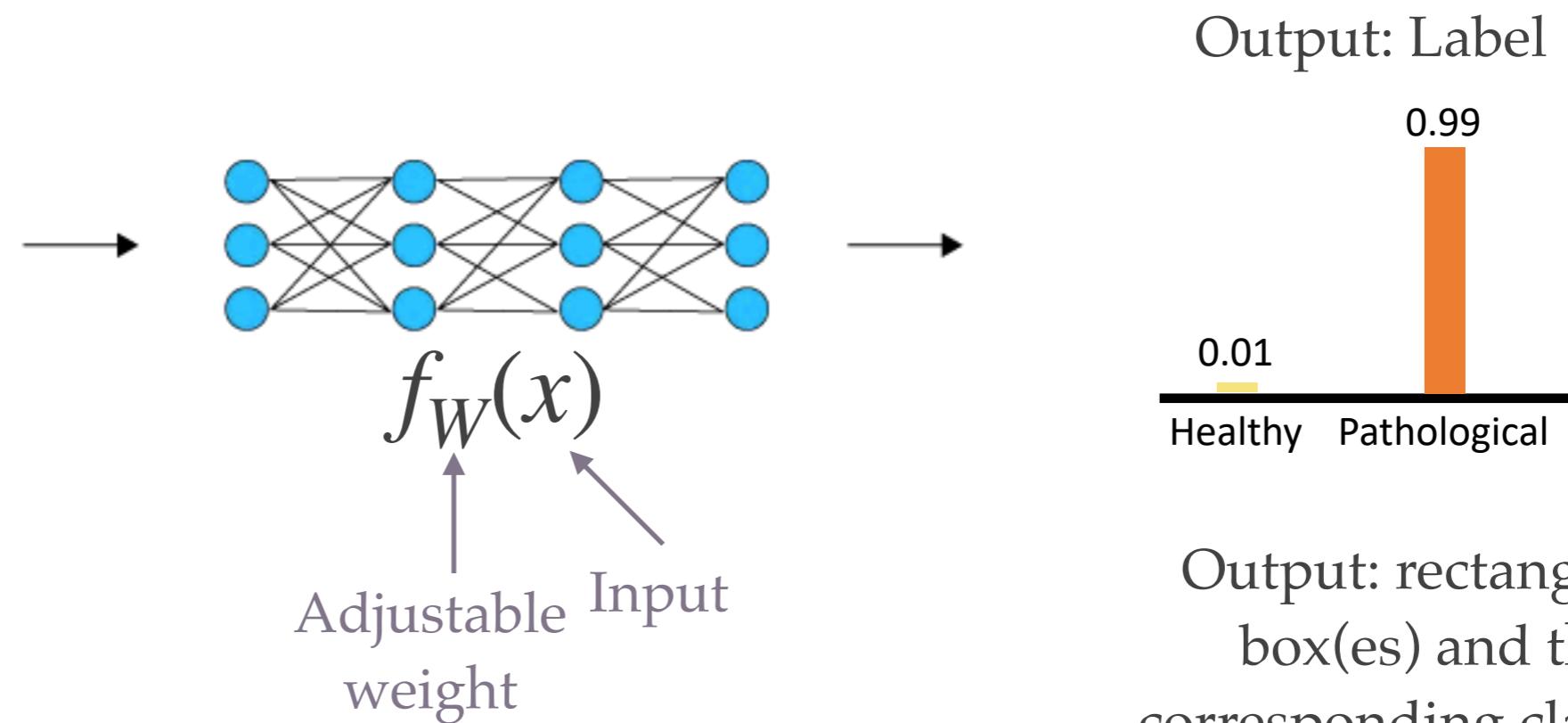
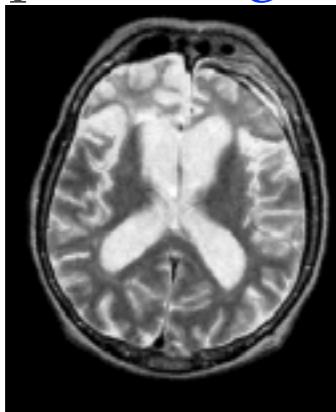
- ❖ Introduction to Supervised Machine Learning
 - ❖ AI vs ML vs DL
 - ❖ Traditional Machine Learning
 - ❖ Training, Validation, and Test Data
 - ❖ Overfitting and Underfitting
- ❖ Introduction to Supervised Deep Learning
 - ❖ Traditional ML vs Deep Learning
 - ❖ Artificial Neuron and Neural Network
 - ❖ Supervised Learning
- ❖ Deep Learning as a Function Approximator
- ❖ Deep Learning Components
 - ❖ Fully Connected Layers
 - ❖ Activation Functions
 - ❖ Optimization

Deep Learning

- ❖ Train a deep neural network (NN) to perform some task from data
- ❖ Think of a deep NN as a universal function approximator

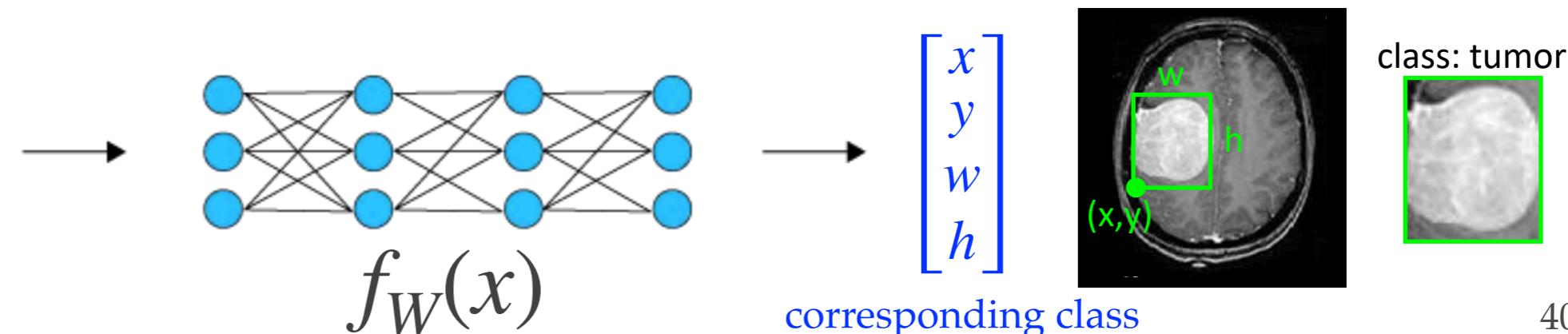
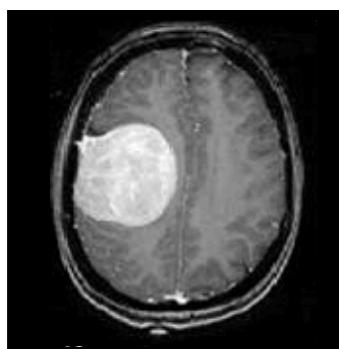
Image classification

Input: image



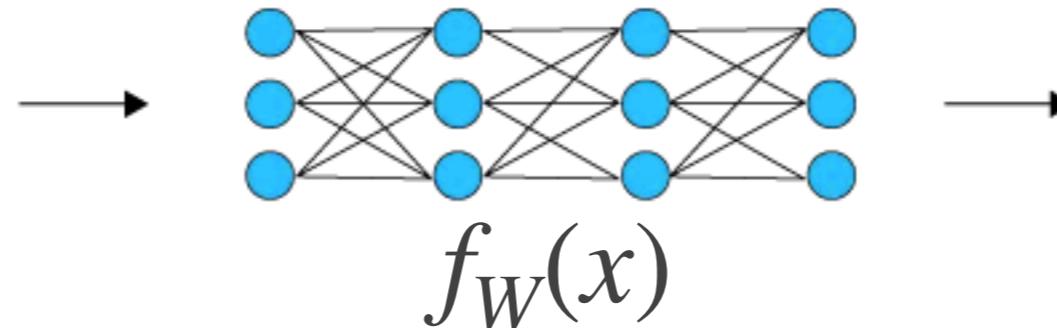
Object detection

Input: image

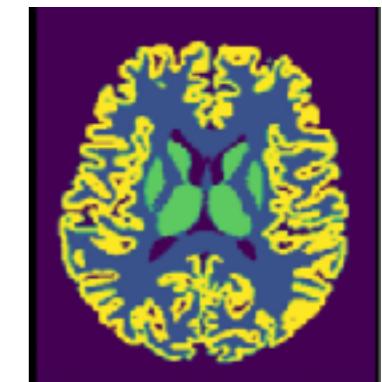


Segmentation

Input
image

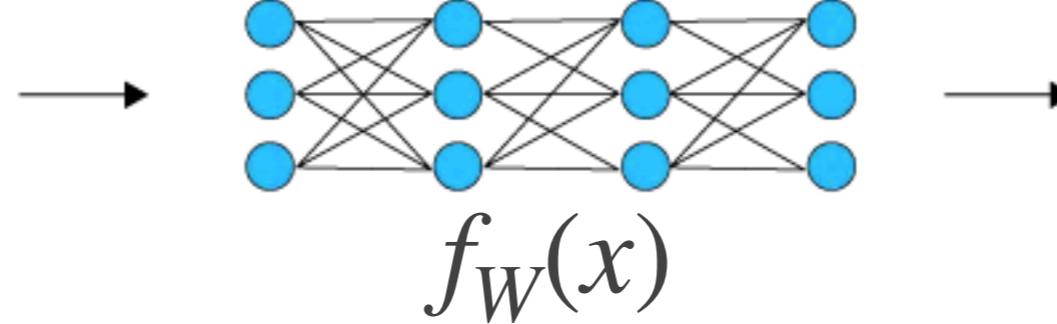
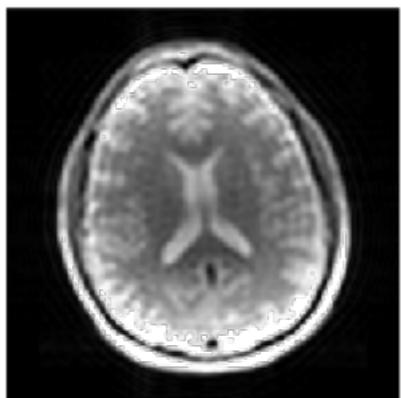


Output
segmentation map

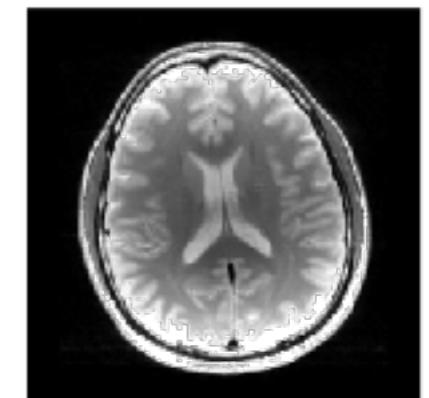


Super-resolution

Input
low-res image

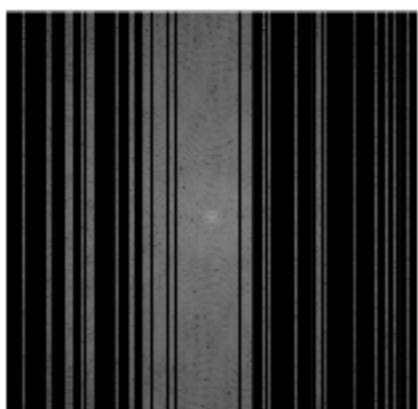


Output
high-res image



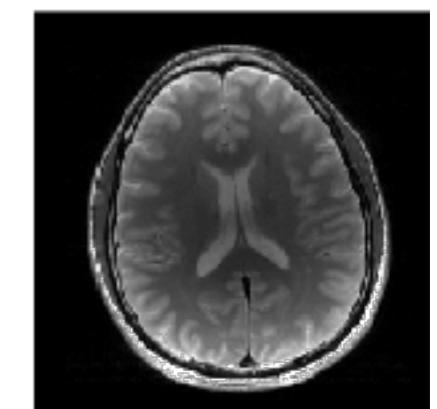
MRI reconstruction function

Input



$f_W(x)$

Output



Outline

- ❖ Introduction to Supervised Machine Learning
 - ❖ AI vs ML vs DL
 - ❖ Traditional Machine Learning
 - ❖ Training, Validation, and Test Data
 - ❖ Overfitting and Underfitting
- ❖ Introduction to Supervised Deep Learning
 - ❖ Traditional ML vs Deep Learning
 - ❖ Artificial Neuron and Neural Network
 - ❖ Supervised Learning
- ❖ Deep Learning as a Function Approximator
- ❖ Deep Learning Components
 - ❖ Fully Connected Layers
 - ❖ Activation Functions
 - ❖ Optimization



Fully connected layer (Dense)

Optimizer
SGD
Adam
RMSprop

Evaluation metric
accuracy
F1-score
AUC
confusion matrix

Loss function
categorical crossentropy
binary crossentropy
mean squared error
mean absolute error

Regularization
Dropout
Data augmentation
 l_1, l_2 regularizations

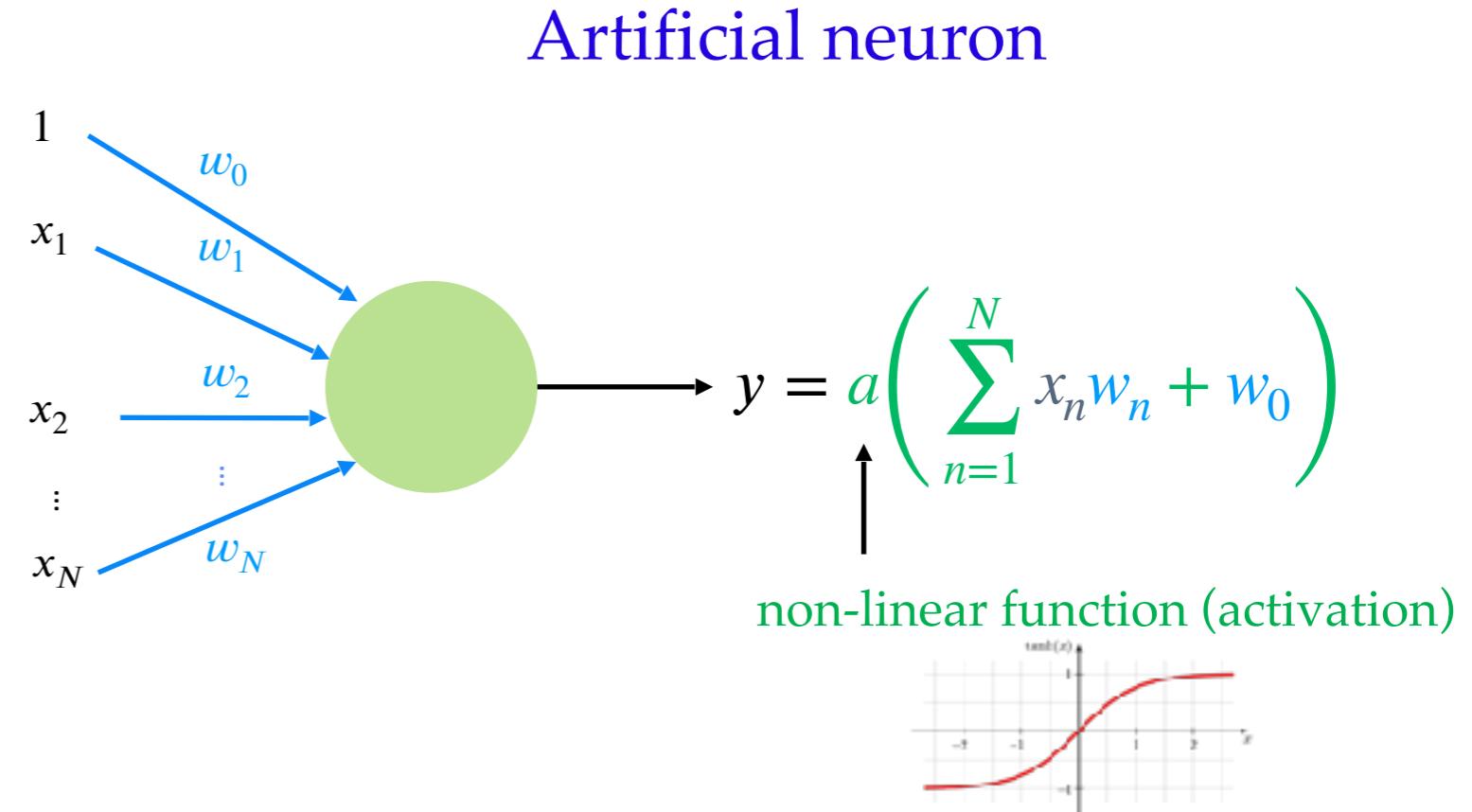
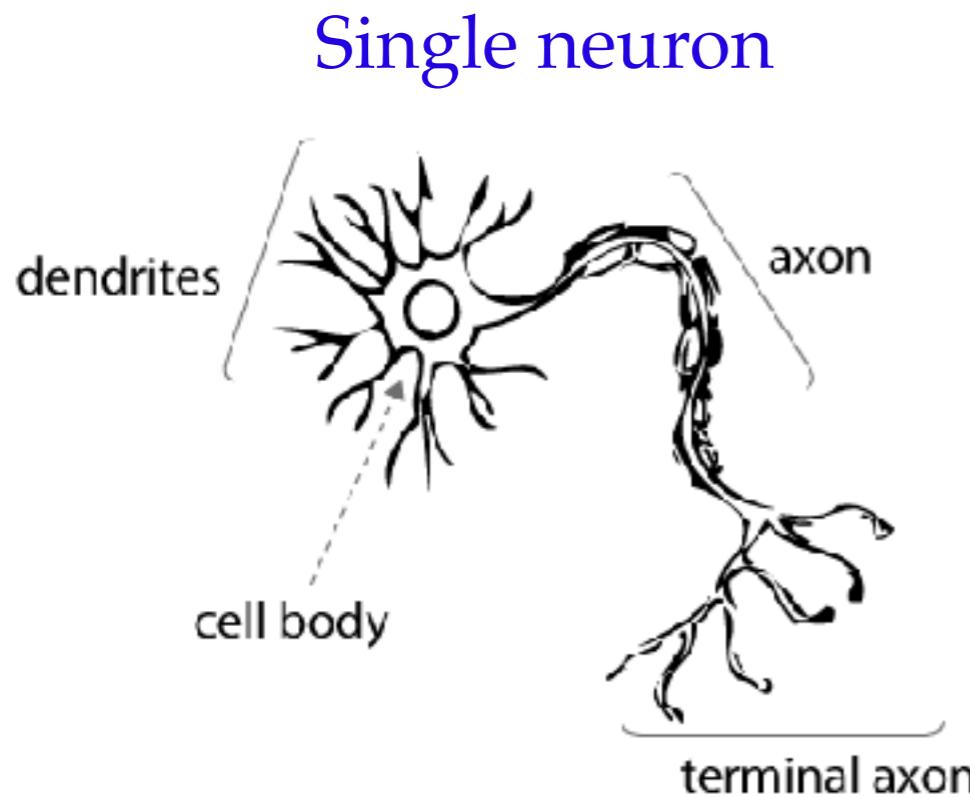
Convolutional layer
Conv1D, 2D, 3D, ...
separable Conv

Pooling layer
max-pooling
average-pooling

Activation function
sigmoid
softmax
ESP (swish)
ReLU

- ❖ **Combine basic components to build a neural network**
 - More components → “More” representative power

Artificial Neuron



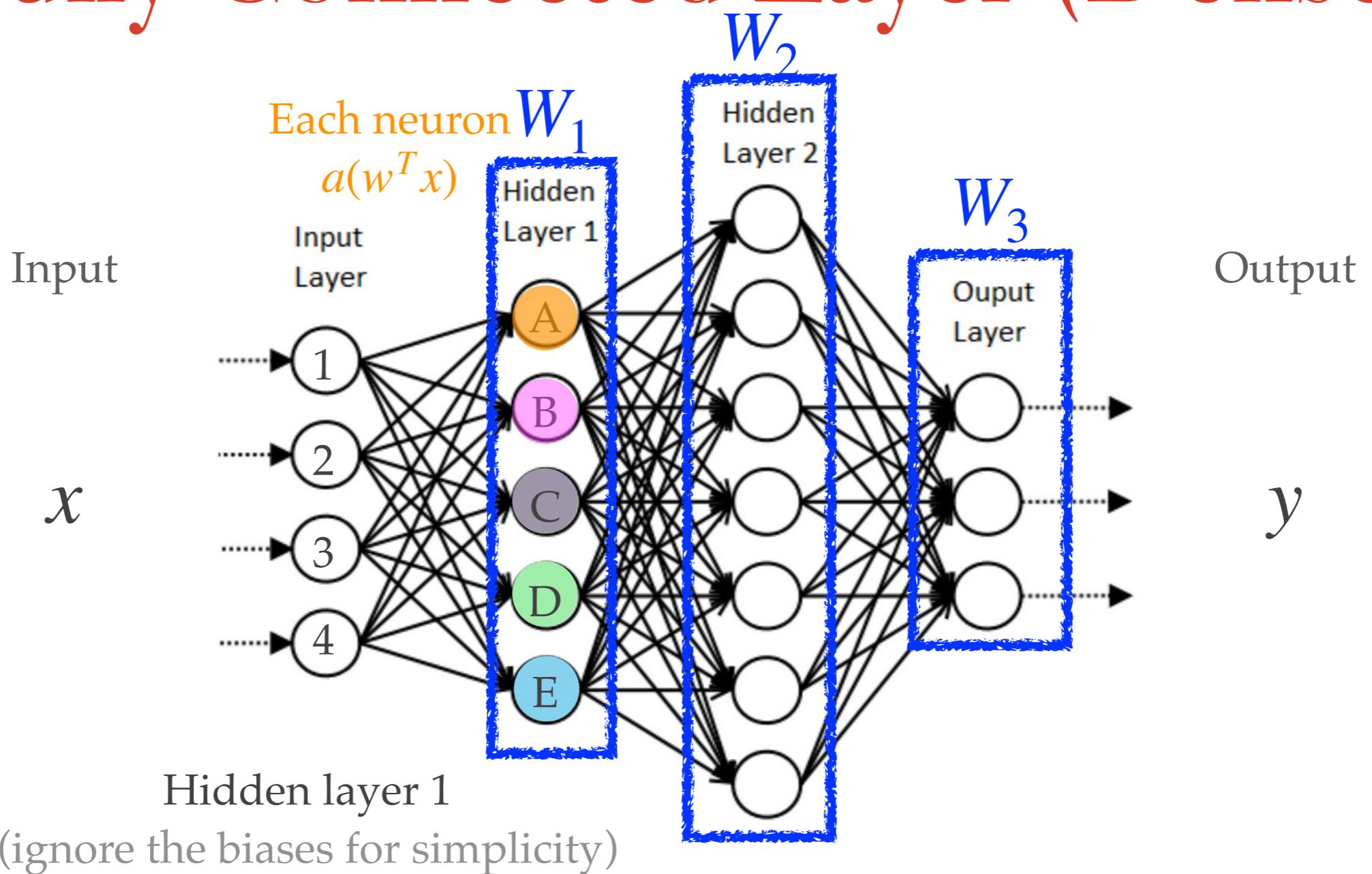
If we set all the weights to 0, then $y = 0$.

If we set $w_1 = 1$ and the rest to 0, then $y = a(x_1)$.

If we set $w_0 = 0$ and the rest to 1, then $y = a(x_1 + x_2 + \dots + x_N)$.

Different weights give rise
to different behavior

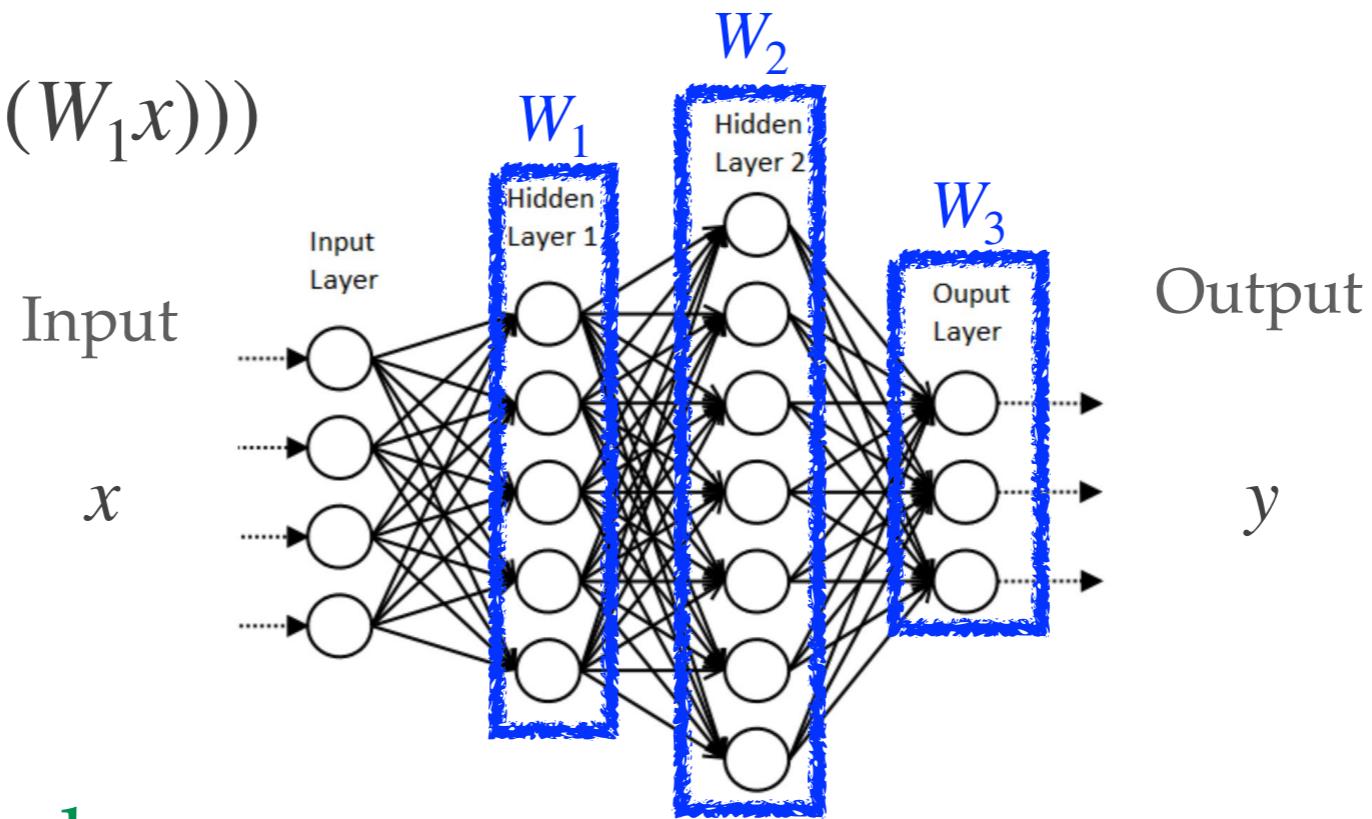
Fully Connected Layer (Dense)



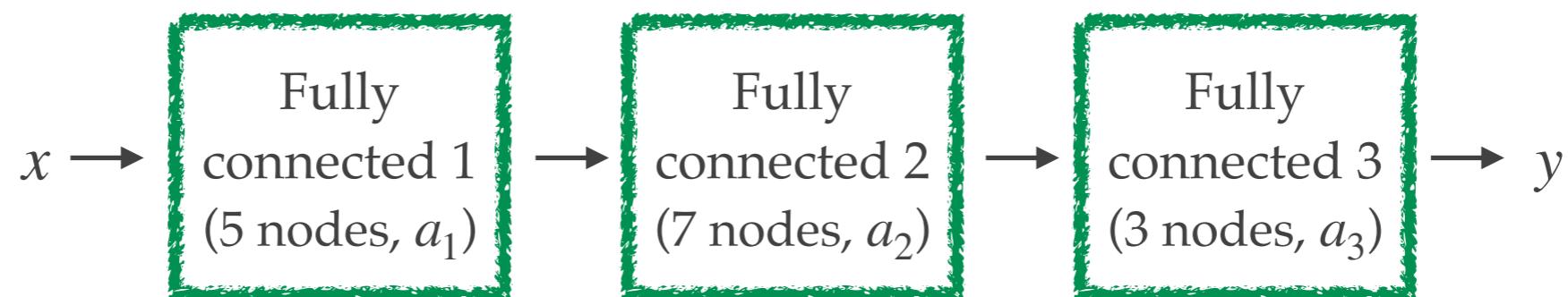
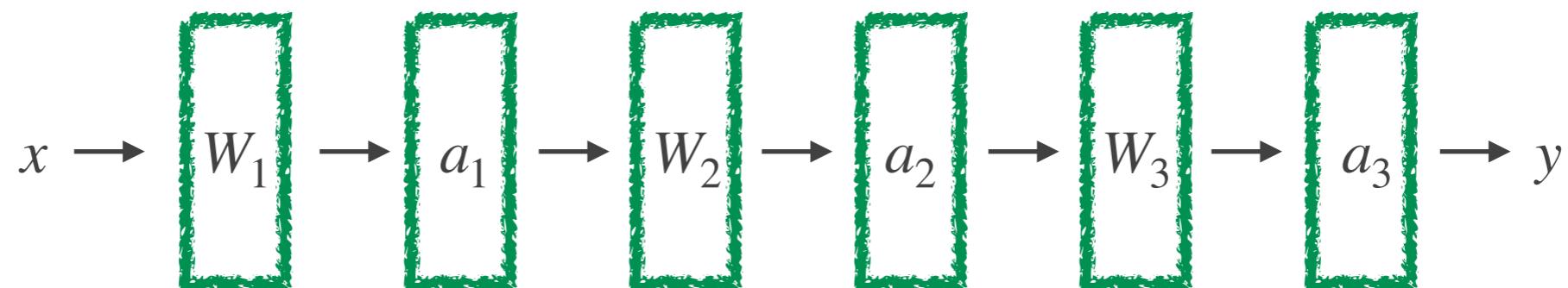
$$\begin{bmatrix} out_A \\ out_B \\ out_C \\ out_D \\ out_E \end{bmatrix} = a_1(W_1x) = a_1 \left(\begin{bmatrix} w_{A,1} & w_{A,2} & w_{A,3} & w_{A,4} \\ w_{B,1} & w_{B,2} & w_{B,3} & w_{B,4} \\ w_{C,1} & w_{C,2} & w_{C,3} & w_{C,4} \\ w_{D,1} & w_{D,2} & w_{D,3} & w_{D,4} \\ w_{E,1} & w_{E,2} & w_{E,3} & w_{E,4} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \right) \quad y = a_3(W_3a_2(W_2a_1(W_1x)))$$

Representing Neural Network

$$y = a_3(W_3 a_2(W_2 a_1(W_1 x)))$$



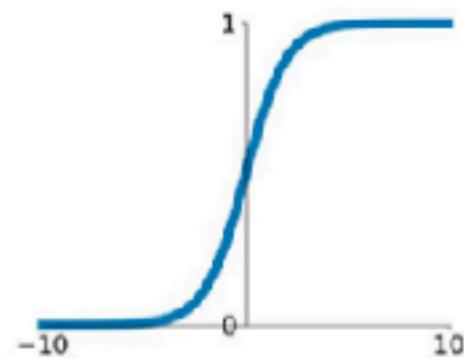
Computation graph



Activation Functions

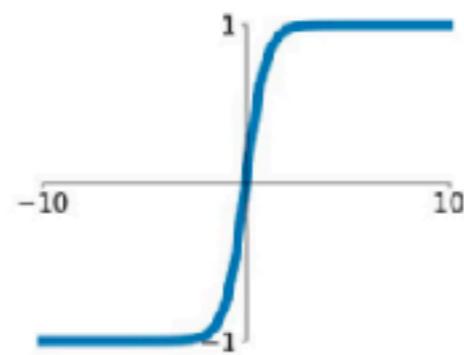
Sigmoid

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



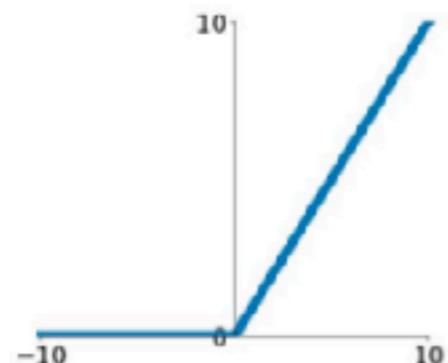
tanh

$$\tanh(x)$$



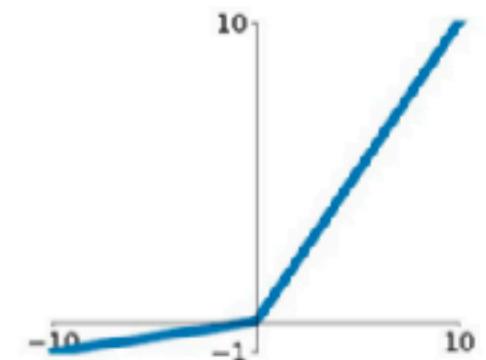
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

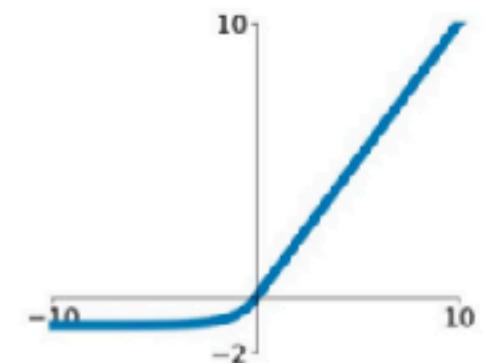


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

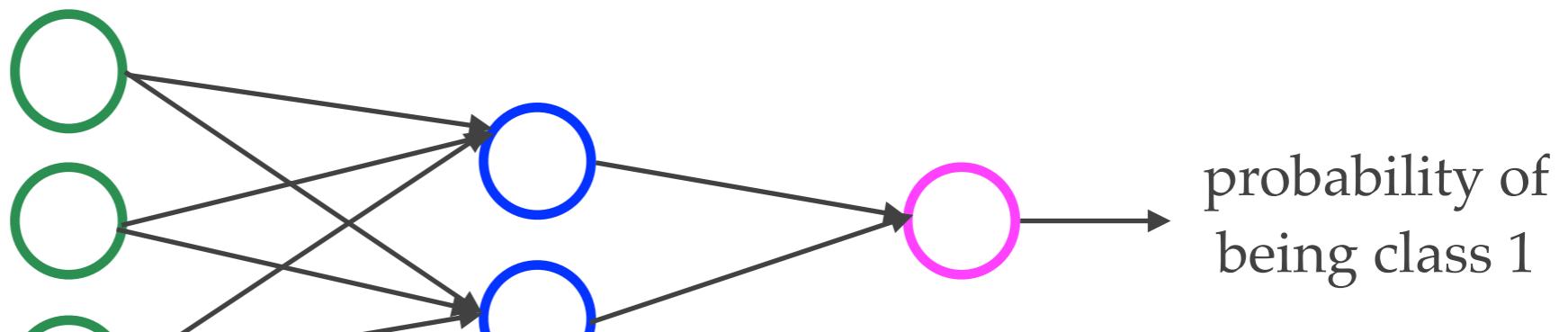
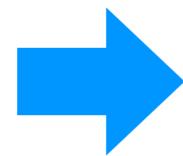
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



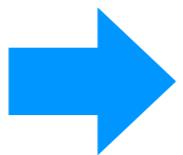
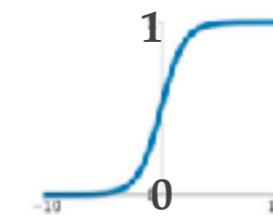
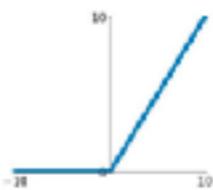
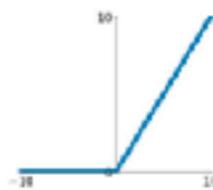
class 1 class 2



Two-class Classification



activation functions



Fully connected 1
(3 nodes,
ReLU)

Fully connected 2
(2 nodes,
ReLU)

Fully connected 3
(1 node,
sigmoid)

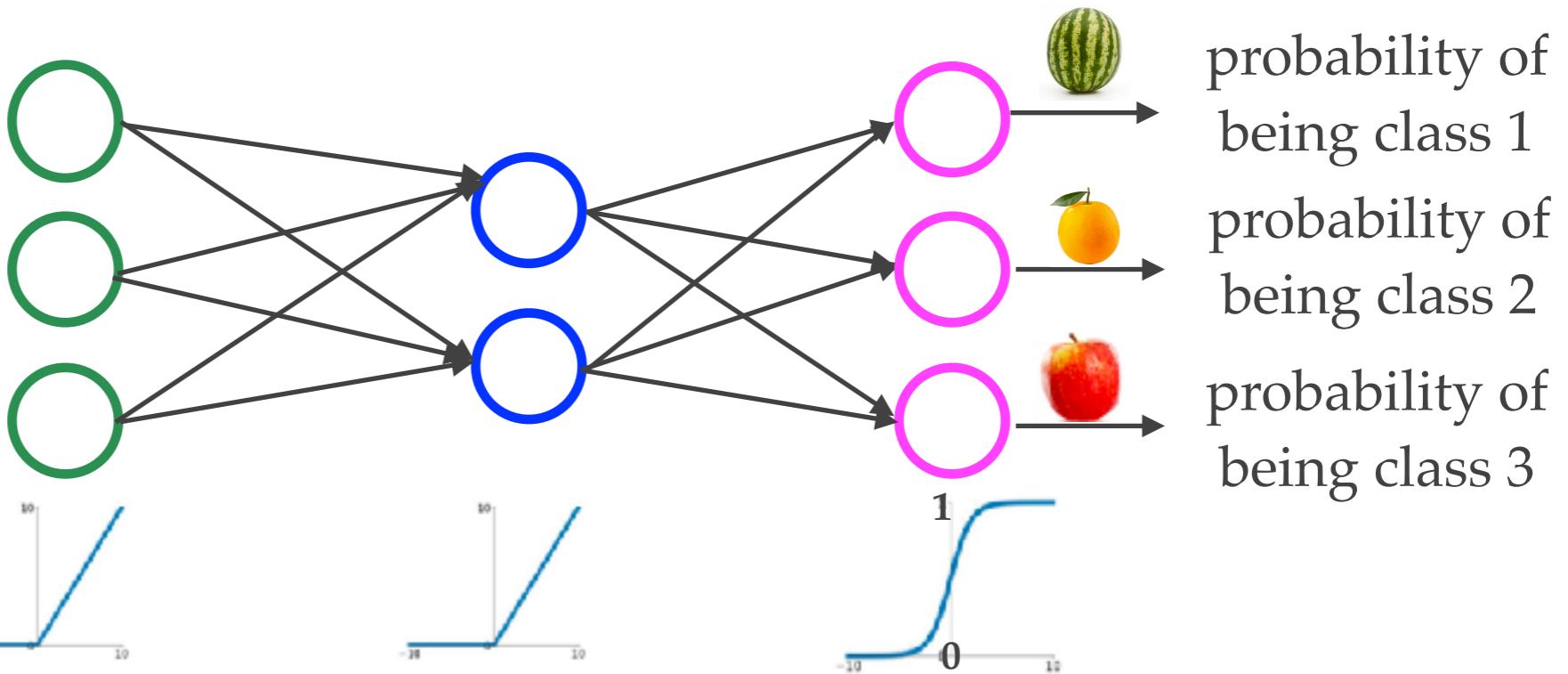
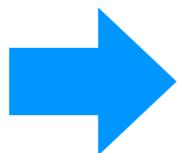
→ 0.99

```
model.add(tf.keras.layers.Dense(3, activation='relu'))  
model.add(tf.keras.layers.Dense(2, activation='relu'))  
model.add(tf.keras.layers.Dense(1, activation='sigmoid'))
```

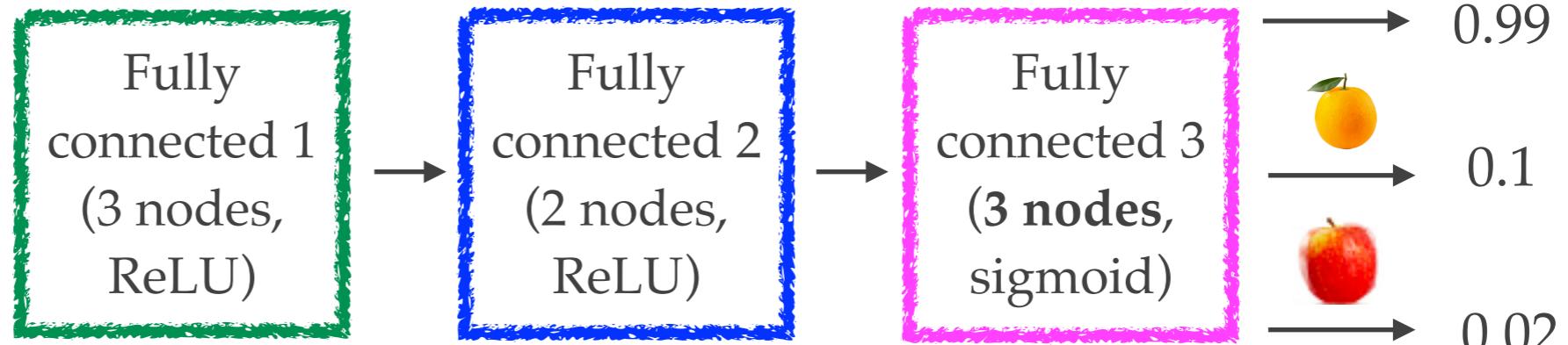
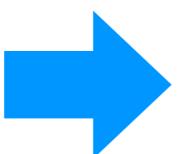
class 1 class 2 class 3



Three-class Classification



activation functions

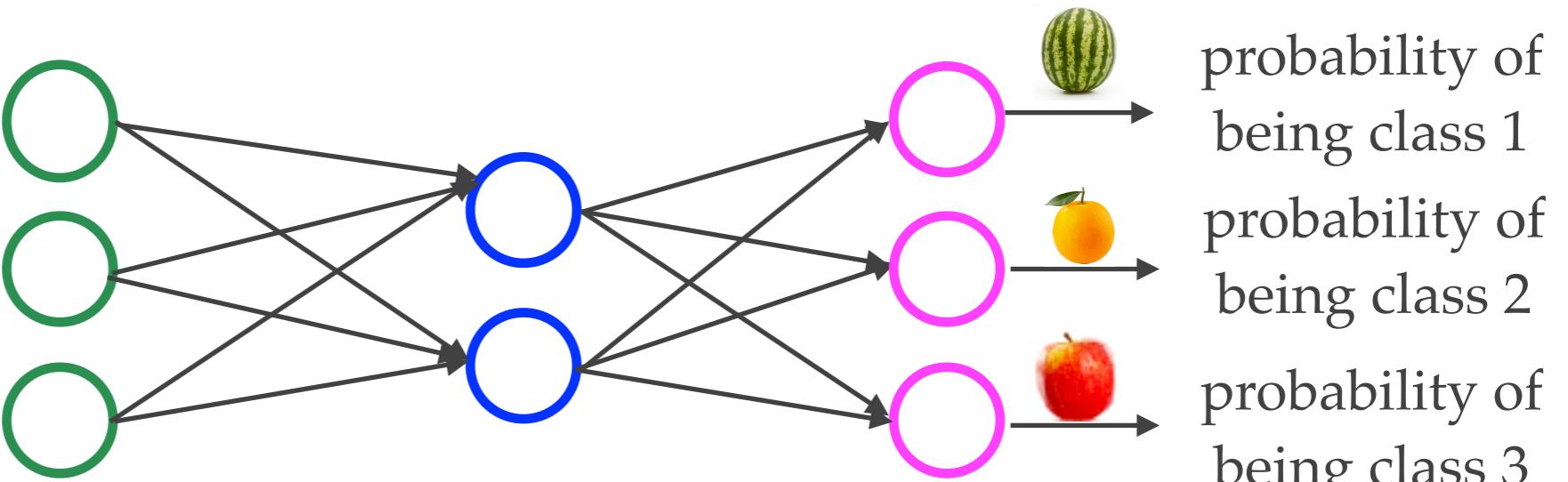
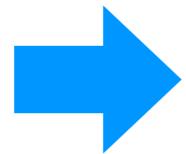


What if our model output something like 0.99 ? Not good
0.99
0.99

class 1 class 2 class 3



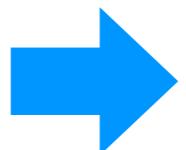
Three-class Classification



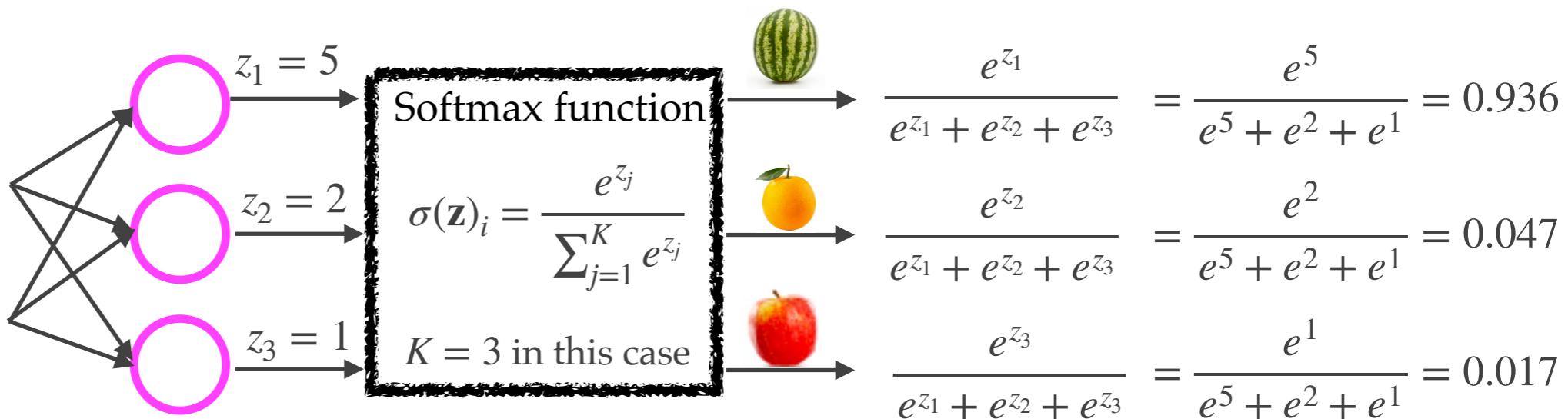
activation functions



Softmax function

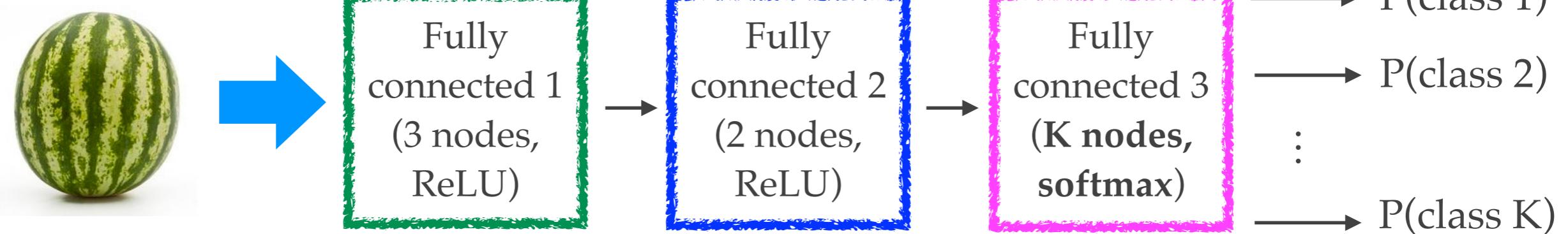


...



$$0.936 + 0.047 + 0.017 = 1$$

Extend the model to K-class Classification



```
model.add(tf.keras.layers.Dense(3, activation='relu'))  
model.add(tf.keras.layers.Dense(2, activation='relu'))  
model.add(tf.keras.layers.Dense(K, activation='softmax'))
```

Fully connected layer (Dense)

Convolutional layer
Conv1D, 2D, 3D, ...
separable Conv

Optimizer
SGD
Adam
RMSprop

Evaluation metric
accuracy
F1-score
AUC
confusion matrix

Loss function
categorical crossentropy
binary crossentropy
mean squared error
mean absolute error

Regularization
Dropout
Data augmentation
 l_1, l_2 regularizations

Pooling layer
max-pooling
average-pooling

Activation function
sigmoid
softmax
ESP (swish)
ReLU

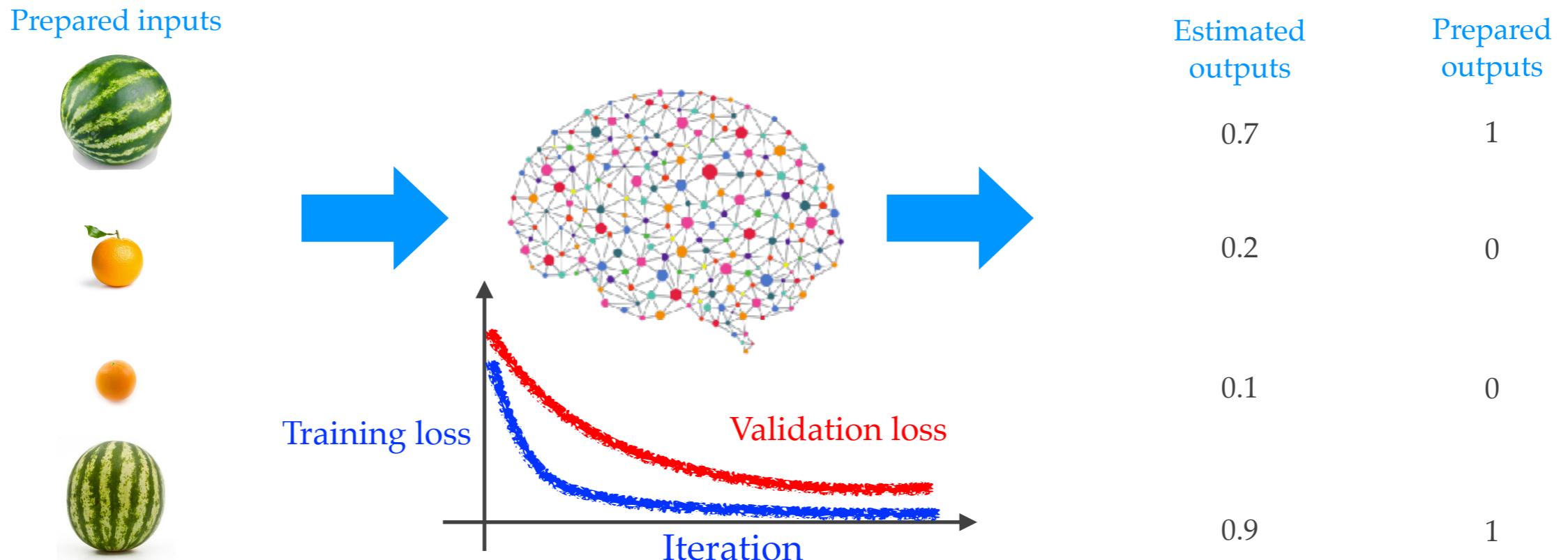
Deep Learning



- ❖ **Combine basic components to build a neural network**
 - More components → “More” representative power

Image Classification Using Deep Learning

- ❖ Training phase: Optimize the weights of a deep neural network



- ❖ Test phase: Perform prediction using the trained neural network



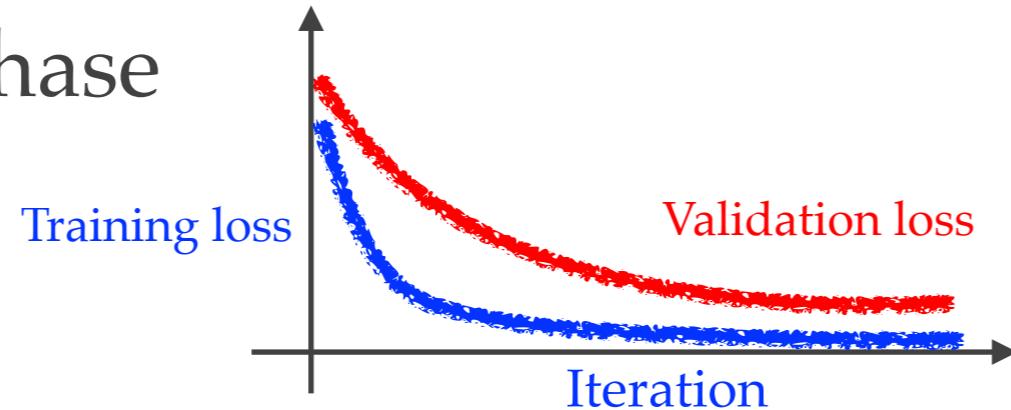
Loss function

(cost function and error function)

- ❖ A function L that maps values of variable(s) onto a real number

For example, $L(y, \hat{y})$ gives us a number that tell us how “different” y and \hat{y} are

- ❖ We have used it to monitor how our model performs during the training phase



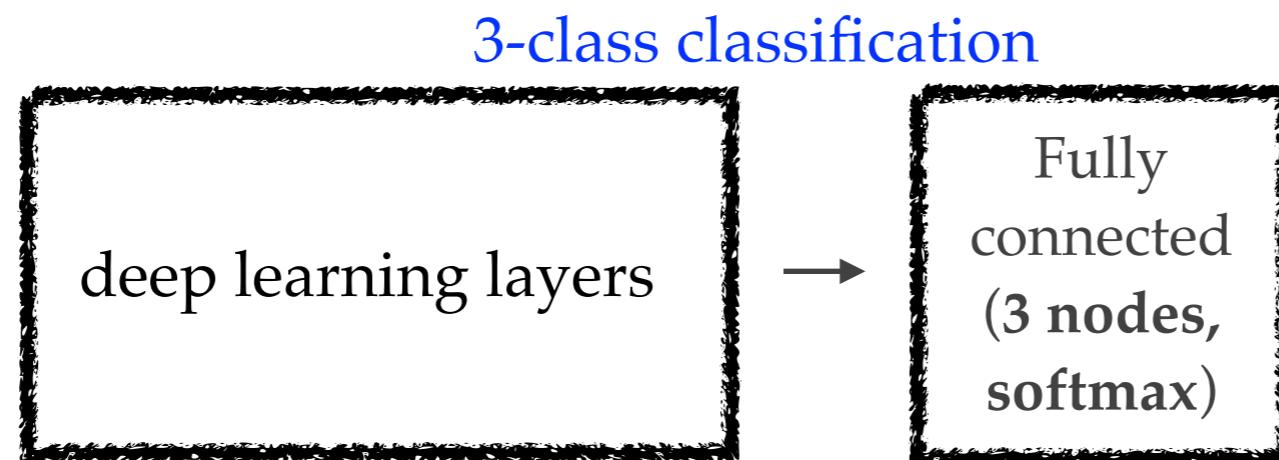
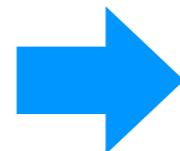
$$L(y, \hat{y}) = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2$$

$$L(y, \hat{y}) = \frac{1}{N} \sum_{i=1}^N |y_i - \hat{y}_i|$$

- ❖ Mean squared error (MSE) and mean absolute error (MAE) are two popular choices for regression
- ❖ Cross-entropy is the most common choice for a classification task

Cross-entropy

- Well suited to comparing probability distributions



predicted prob. dist.	true prob. dist.
\hat{y}	y
0.7	1
0.1	0
0.2	0

$$L(y, \hat{y}) = - \sum_{k=1}^K y_k \log \hat{y}_k \quad \text{by definition } (K = 3)$$
$$= -(1 \times \log(0.7) + 0 \times \log(0.1) + 0 \times \log(0.2)) = -1 \times \log(0.7) = 0.15$$

Assume that we have updated the weights of our model and got $\hat{y} = [0.9, 0.1, 0]$, then $L(y, \hat{y}) = -1 \times \log(0.9) + 0 + 0 = 0.046$

Lower loss. \hat{y} has become more similar to y than the previous case.
Our model performs better!

Optimizing Our Model

- ❖ To train our model f_W , we adjust its weights W in a way that decreases your loss function
- ❖ Particularly, we are minimizing our loss function with respect to W

$$L(y, \hat{y}) = L(y, f_W(x))$$

prepared output /
labels / targets prepared input

predicted Weights
output

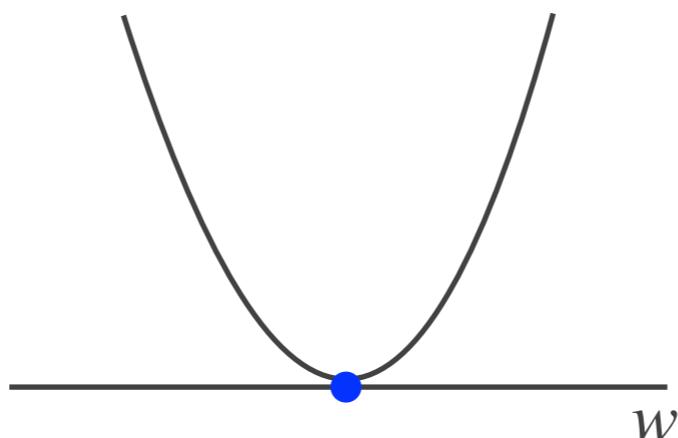
Calculus!

Compute the derivative of $L(y, f_W(x))$ with respect to W and set it to 0.

Example: Optimization

- ❖ Goal: Find w that minimizes the following loss function

$$L(w) = w^2$$



Method 1: Compute the derivative and set it to 0

Gradient $\frac{dL(w)}{dw} = \frac{dw^2}{dw} = 2w = 0 \rightarrow w = 0$

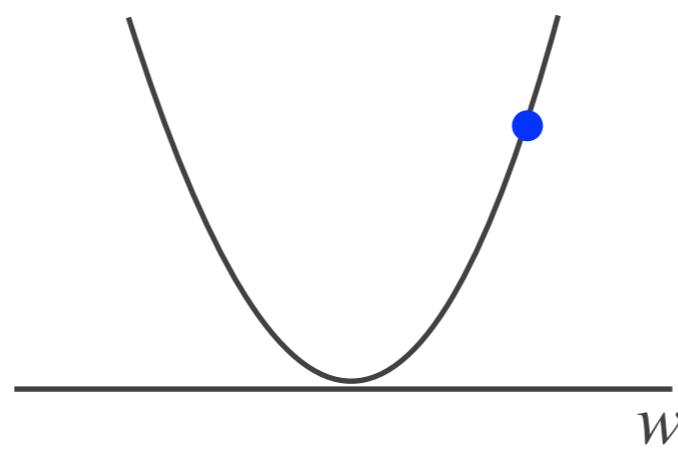
In our case, we have $L(y, f_W(x))$. Not simple to compute the gradient with respect to W .

Even when we have a way to compute the gradient and manage to set it to zero, we might not be able to get a closed-form solution for it.

Example: Optimization

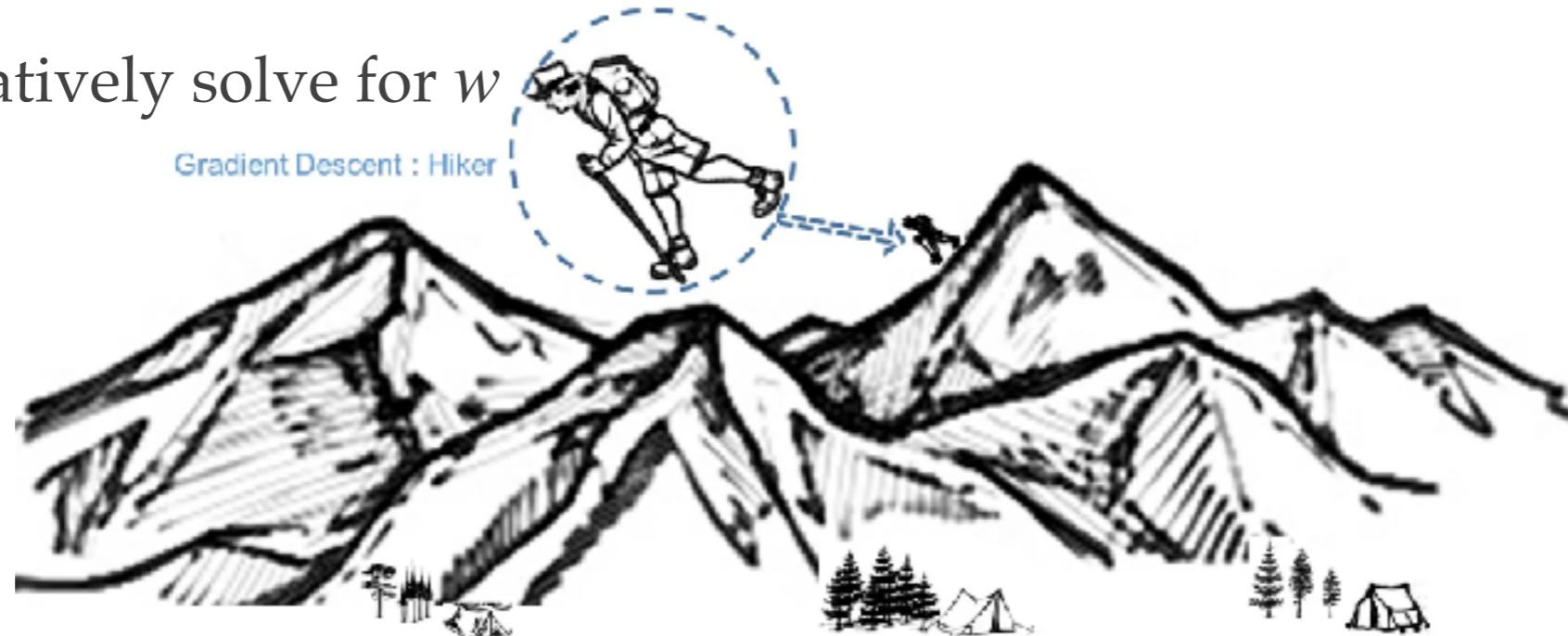
- ❖ Goal: Find w that minimizes the following loss function

$$L(w) = w^2$$



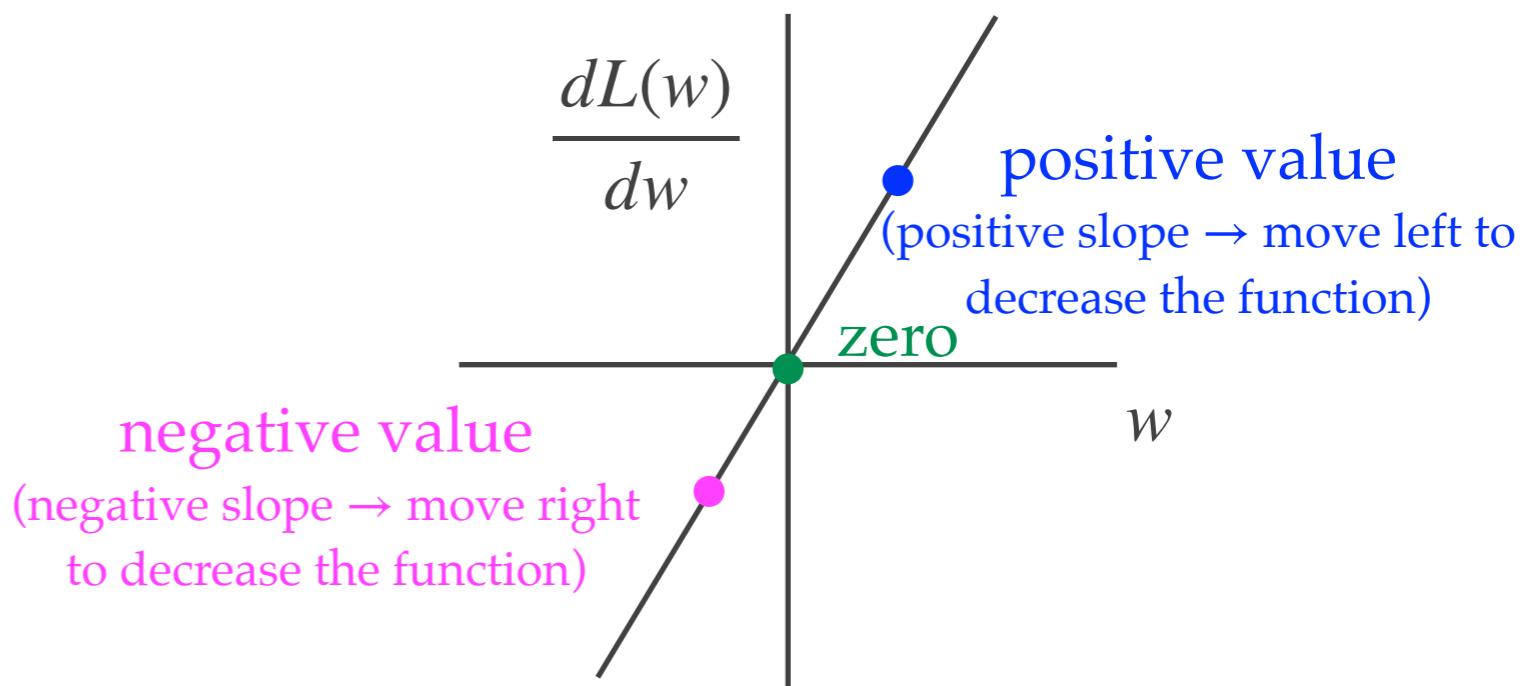
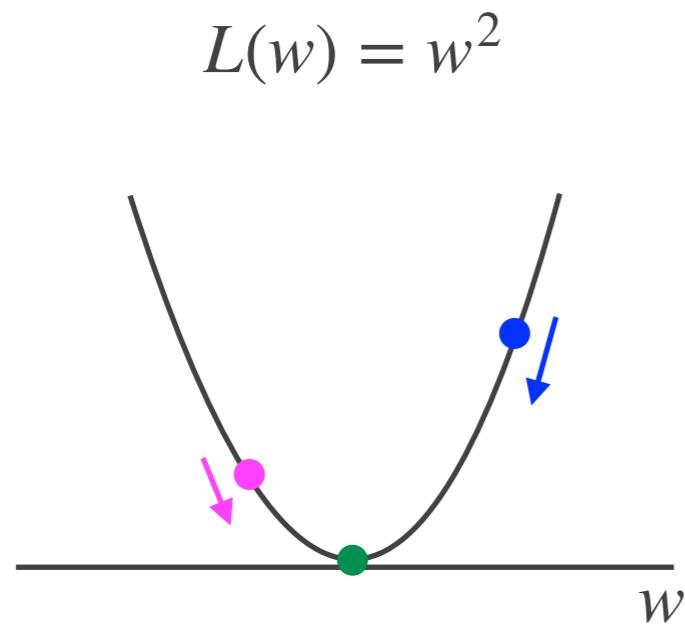
- Randomly start somewhere
- Gradually move to the minimum of the function

Method 2: Iteratively solve for w



Example: Optimization

- Goal: Find w that minimizes the following loss function



Method 2: Iteratively solve for w

1. Guess an initial solution w_0 and pick α
- For $k = 0, 1, 2, \dots$

2. Compute $\frac{dL(w)}{dw} \Big|_{w=w^{(k)}} = 2w \Big|_{w=w^{(k)}} = 2w^{(k)}$

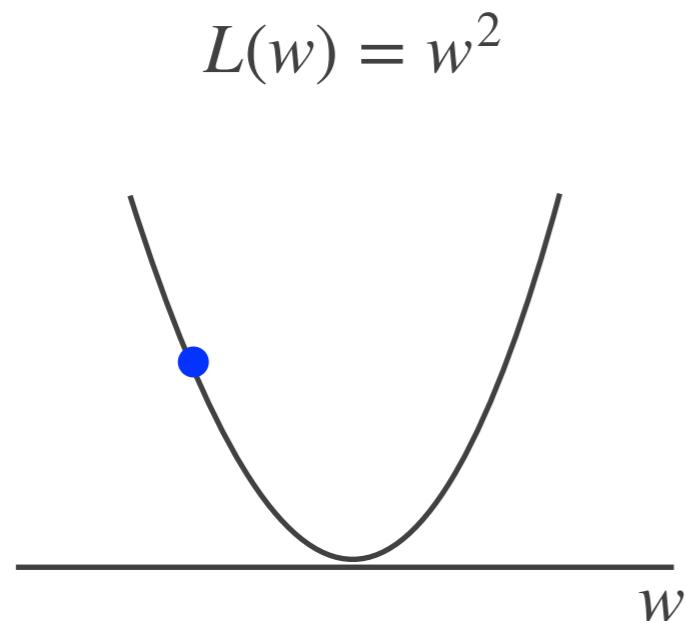
3. Compute a new solution $w^{(k+1)} := w^{(k)} - \alpha \frac{dL(w)}{dw} \Big|_{w=w^{(k)}} = w^{(k)} - \alpha * 2w^{(k)}$
4. Repeat step 2 and 3 until converge

The negative of the gradient $-\frac{dL(w)}{dw}$ tells us the direction we should move to decrease the function

\uparrow
step size/learning rate α indicates how far we want to move

Example: Optimization

- ❖ Goal: Find w that minimizes the following loss function



$$w_{(0)} = -2, \alpha = 0.5$$

$$w_{(1)} = w_{(0)} - \alpha * 2w_{(0)} = -2 - 0.5 * 2 * -2 = 0$$

$$w_{(2)} = w_{(1)} - \alpha * 2w_{(1)} = 0$$

$$w_{(3)} = w_{(2)} - \alpha * 2w_{(2)} = 0$$

Method 2: Iteratively solve for w

1. Guess an initial solution w_0 and pick α

For $k = 0, 1, 2, \dots$

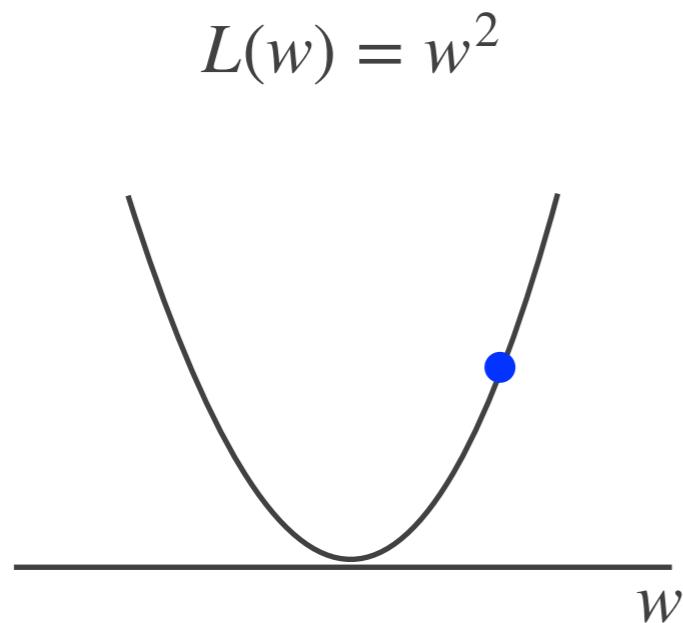
2. Compute $\frac{dL(w)}{dw} \Big|_{w=w(k)} = 2w \Big|_{w=w(k)} = 2w_{(k)}$

3. Compute a new solution $w_{(k+1)} := w_{(k)} - \alpha \frac{dL(w)}{dw} \Big|_{w=w(k)} = w_{(k)} - \alpha * 2w_{(k)}$

4. Repeat step 2 and 3 until converge

Example: Optimization

- ❖ Goal: Find w that minimizes the following loss function



$$w_{(0)} = 2, \alpha = 0.5$$

$$w_{(1)} = w_{(0)} - \alpha * 2w_{(0)} = 2 - 0.5 * 2 * 2 = 0$$

$$w_{(2)} = w_{(1)} - \alpha * 2w_{(1)} = 0$$

$$w_{(3)} = w_{(2)} - \alpha * 2w_{(2)} = 0$$

Method 2: Iteratively solve for w

1. Guess an initial solution w_0 and pick α

For $k = 0, 1, 2, \dots$

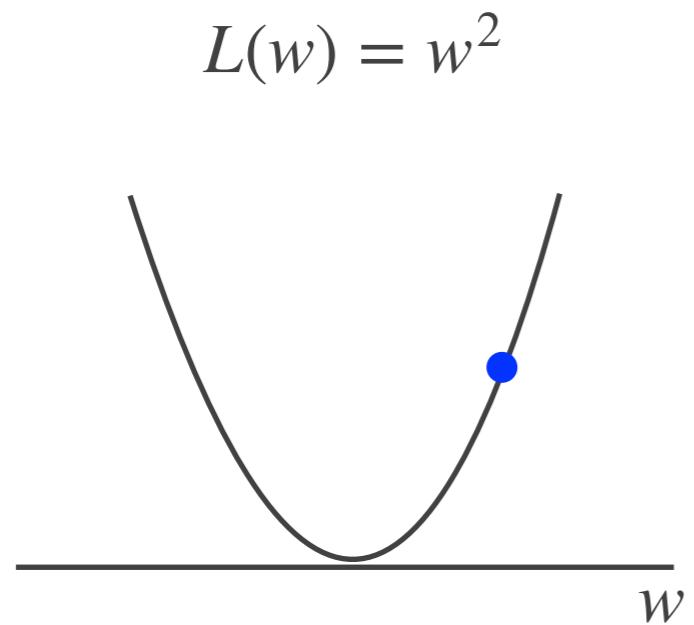
2. Compute $\frac{dL(w)}{dw} \Big|_{w=w_{(k)}} = 2w \Big|_{w=w_{(k)}} = 2w_{(k)}$

3. Compute a new solution $w_{(k+1)} := w_{(k)} - \alpha \frac{dL(w)}{dw} \Big|_{w=w_{(k)}} = w_{(k)} - \alpha * 2w_{(k)}$

4. Repeat step 2 and 3 until converge

Example: Optimization

- Goal: Find w that minimizes the following loss function



$$L(w) = w^2$$

$$w_{(0)} = 2, \alpha = 0.25$$

$$w_{(1)} = w_{(0)} - \alpha * 2w_{(0)} = 2 - 0.25 * 2 * 2 = 1$$

$$w_{(2)} = w_{(1)} - \alpha * 2w_{(1)} = 1 - 0.25 * 2 * 1 = 0.5$$

$$w_{(3)} = w_{(2)} - \alpha * 2w_{(2)} = 0.5 - 0.25 * 2 * 0.5 = 0.25$$

$$w_{(4)} = w_{(3)} - \alpha * 2w_{(3)} = 0.125$$

$$w_{(5)} = w_{(4)} - \alpha * 2w_{(4)} = 0.0625$$

Method 2: Iteratively solve for w

1. Guess an initial solution w_0 and pick α

For $k = 0, 1, 2, \dots$

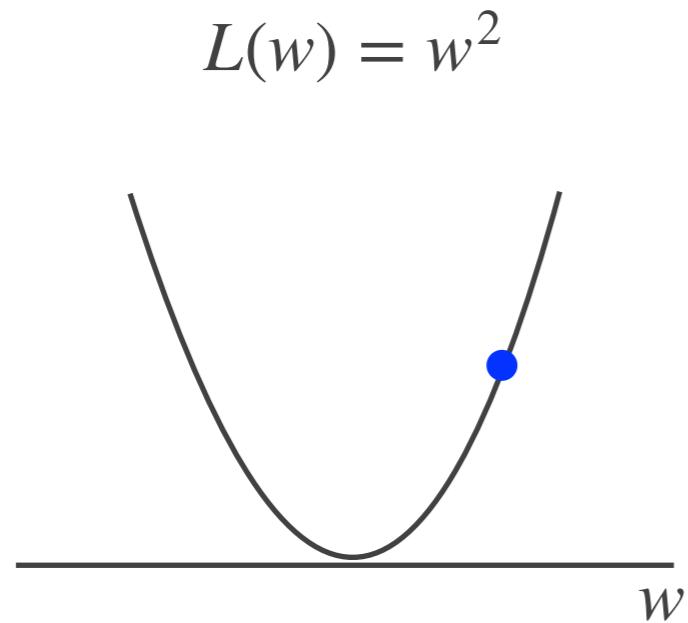
2. Compute $\frac{dL(w)}{dw} \Big|_{w=w_{(k)}} = 2w \Big|_{w=w_{(k)}} = 2w_{(k)}$

3. Compute a new solution $w_{(k+1)} := w_{(k)} - \alpha \frac{dL(w)}{dw} \Big|_{w=w_{(k)}} = w_{(k)} - \alpha * 2w_{(k)}$

4. Repeat step 2 and 3 until converge

Example: Optimization

- Goal: Find w that minimizes the following loss function



$$L(w) = w^2$$

$$w_{(0)} = 2, \alpha = 1$$

$$w_{(1)} = w_{(0)} - \alpha * 2w_{(0)} = 2 - 1 * 2 * 2 = -2$$

$$w_{(2)} = w_{(1)} - \alpha * 2w_{(1)} = -2 - 1 * 2 * -2 = 2$$

$$w_{(3)} = w_{(2)} - \alpha * 2w_{(2)} = -2$$

$$w_{(4)} = w_{(3)} - \alpha * 2w_{(3)} = 2$$

$$w_{(5)} = w_{(4)} - \alpha * 2w_{(4)} = -2$$

Method 2: Iteratively solve for w

1. Guess an initial solution w_0 and pick α

For $k = 0, 1, 2, \dots$

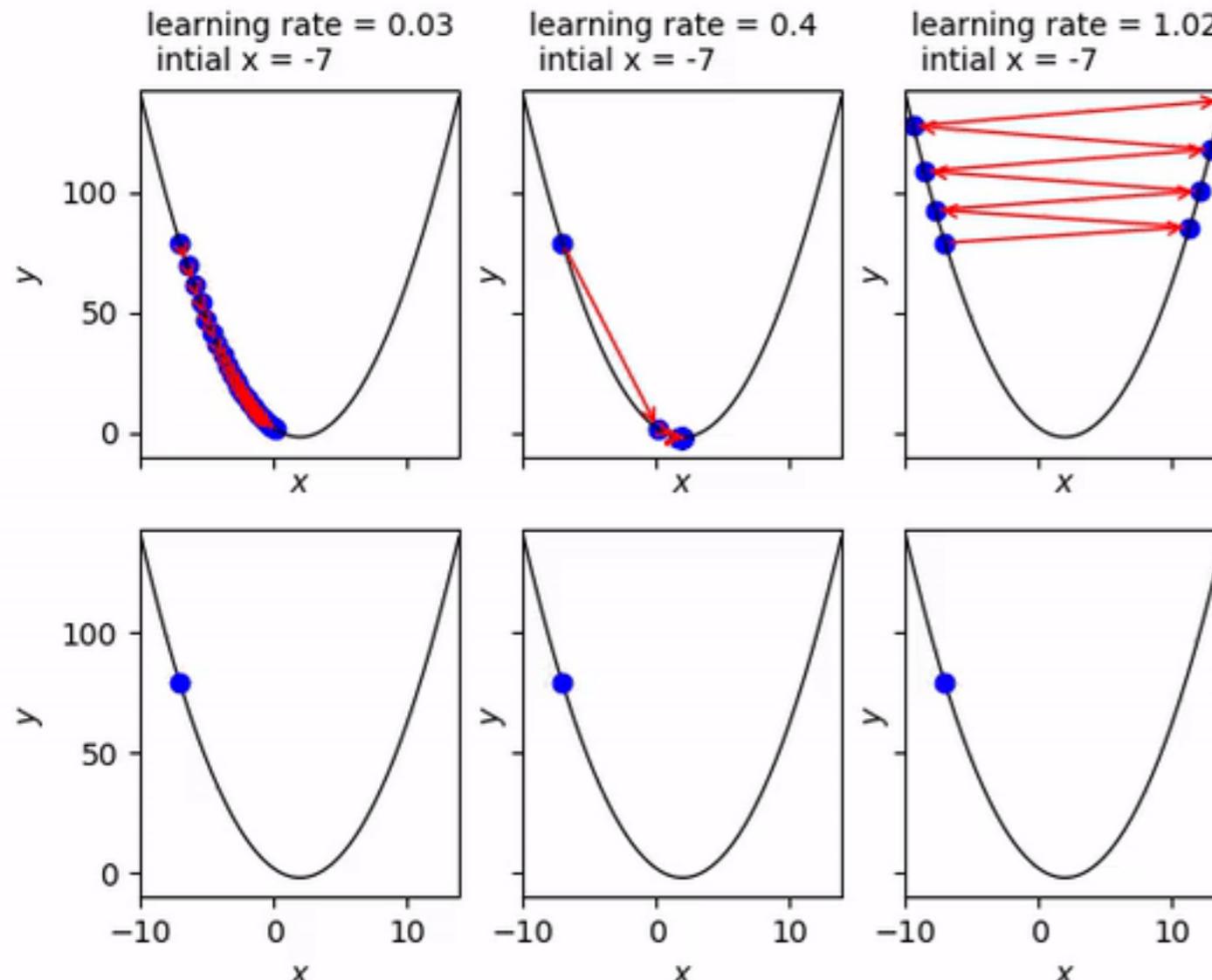
α : learning rate

2. Compute $\frac{dL(w)}{dw} \Big|_{w=w_{(k)}} = 2w \Big|_{w=w_{(k)}} = 2w_{(k)}$

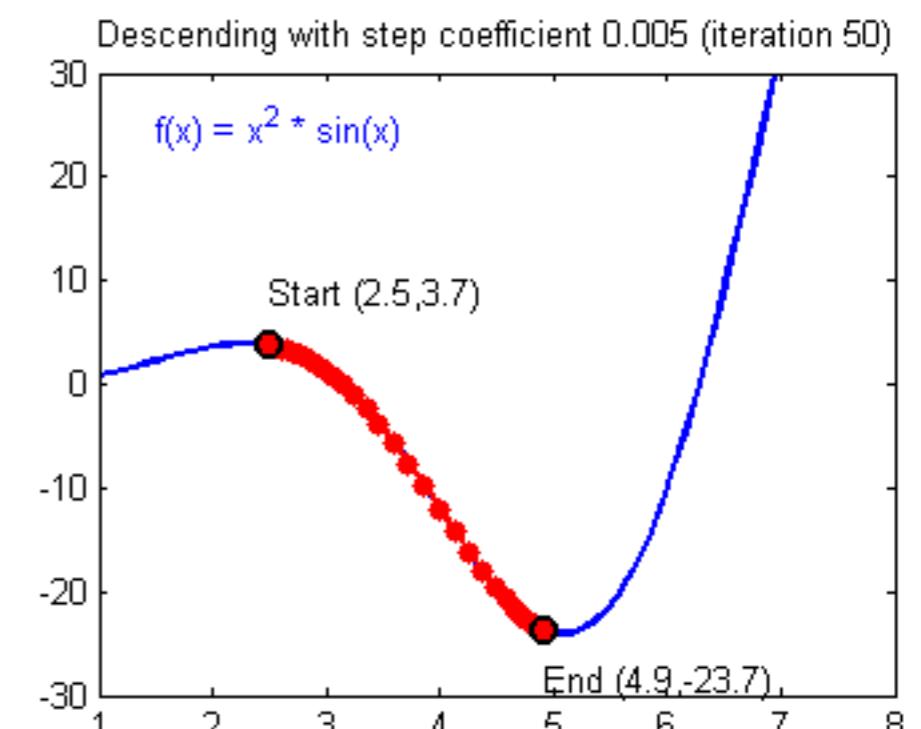
3. Compute a new solution $w_{(k+1)} := w_{(k)} - \alpha \frac{dL(w)}{dw} \Big|_{w=w_{(k)}} = w_{(k)} - \alpha * 2w_{(k)}$

4. Repeat step 2 and 3 until converge

Optimization: Learning Rate



Fixed learning rate

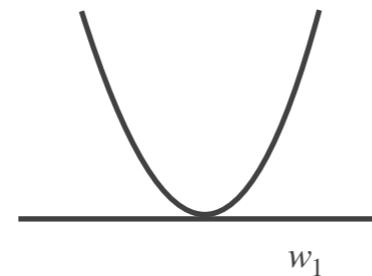


Adaptive learning rate

Gradient

1-dimensional case

$$L(w_1) = w_1^2$$



Derivative of $L(w_1)$

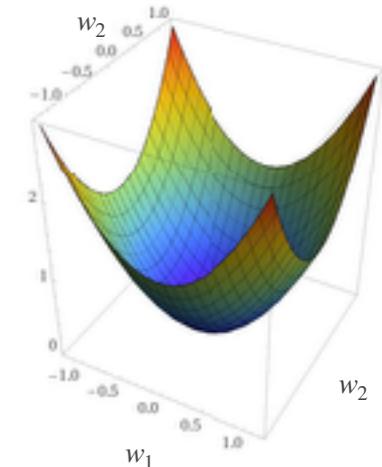
$$\frac{dL(w_1)}{dw_1} = \frac{dw_1^2}{dw_1} = 2w_1$$

Weight update

$$\begin{aligned} w_{1,(k+1)} &:= w_{1,(k)} - \alpha \frac{dL(w_1)}{dw_1} \Big|_{w_1=w_{1,(k)}} \\ &= w_{1,(k)} - 2w_{1,(k)} \end{aligned}$$

2-dimensional case

$$L(w_1, w_2) = w_1^2 + w_2^2$$



Partial derivatives of $L(w_1, w_2)$

$$\frac{\partial L(w_1, w_2)}{\partial w_1} = \frac{\partial}{\partial w_1}(w_1^2 + w_2^2) = 2w_1$$

$$\frac{\partial L(w_1, w_2)}{\partial w_2} = \frac{\partial}{\partial w_2}(w_1^2 + w_2^2) = 2w_2$$

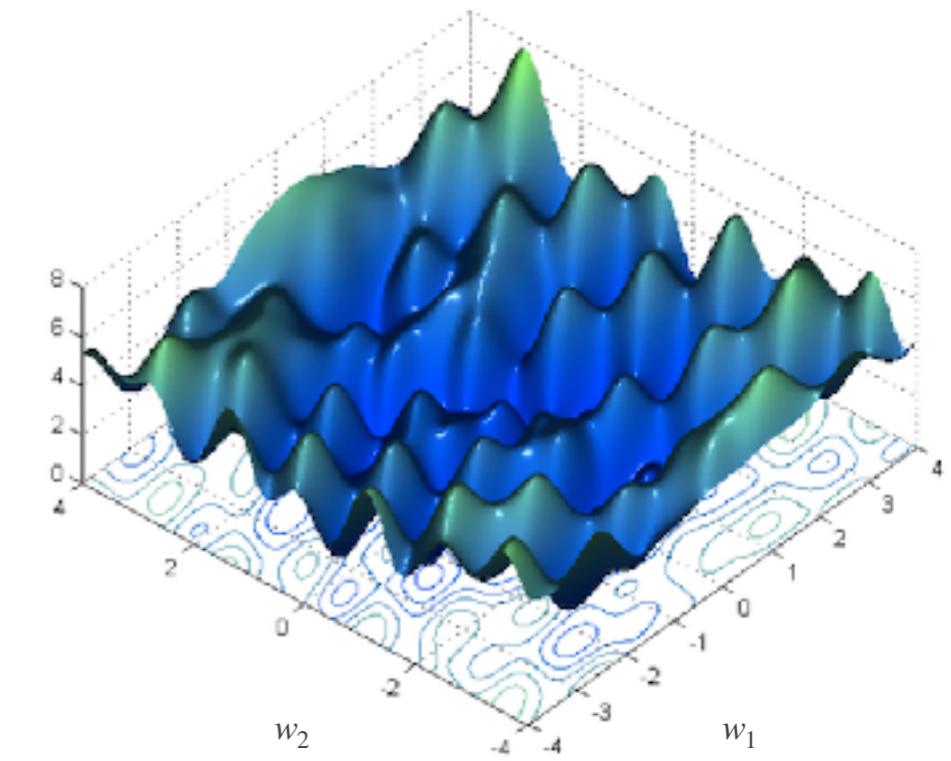
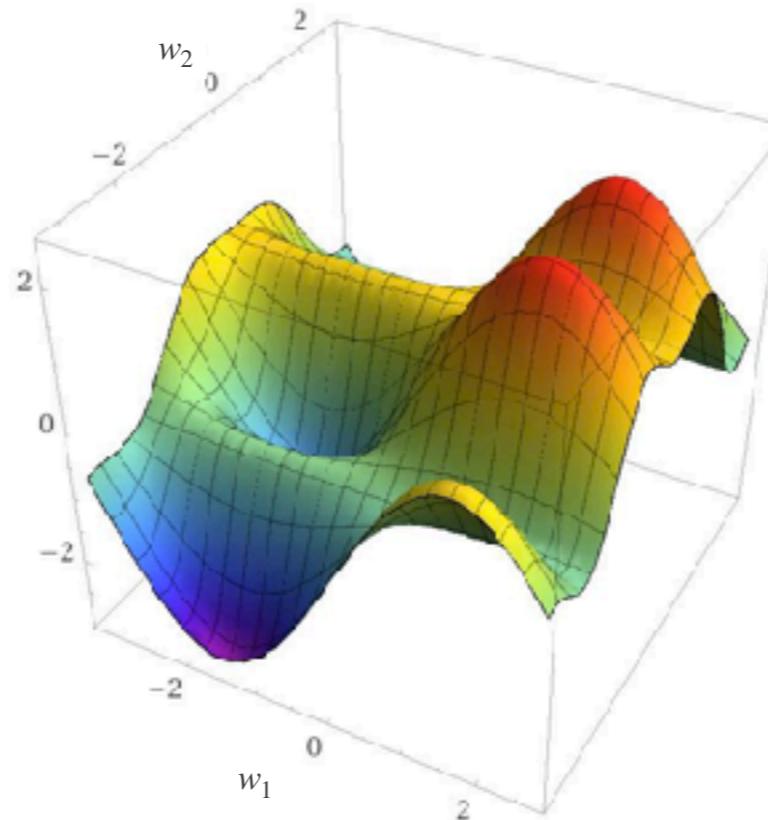
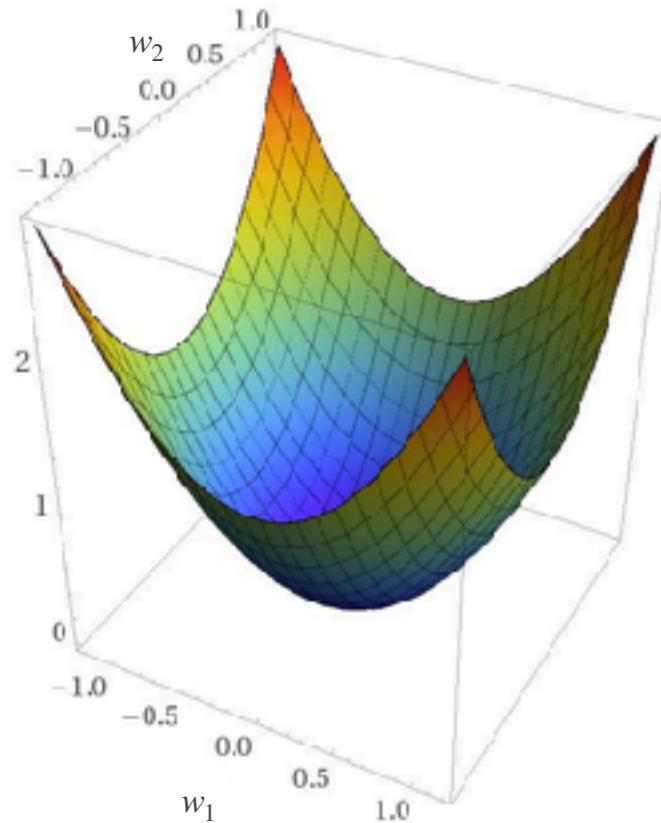
Weight update

$$\begin{aligned} \begin{bmatrix} w_{1,(k+1)} \\ w_{2,(k+1)} \end{bmatrix} &= \begin{bmatrix} w_{1,(k)} \\ w_{2,(k)} \end{bmatrix} - \alpha \begin{bmatrix} \frac{\partial L(w_1, w_2)}{\partial w_1} \Big|_{w_1=w_{1,(k)}} \\ \frac{\partial L(w_1, w_2)}{\partial w_2} \Big|_{w_2=w_{2,(k)}} \end{bmatrix} \\ &= \begin{bmatrix} w_{1,(k)} \\ w_{2,(k)} \end{bmatrix} - \alpha \begin{bmatrix} 2w_{1,(k)} \\ 2w_{2,(k)} \end{bmatrix} \end{aligned}$$

Gradient $\nabla_w L(w_1, w_2)$

Optimization

Real Life



No local minima
problem
(global minimum
= local minimum)

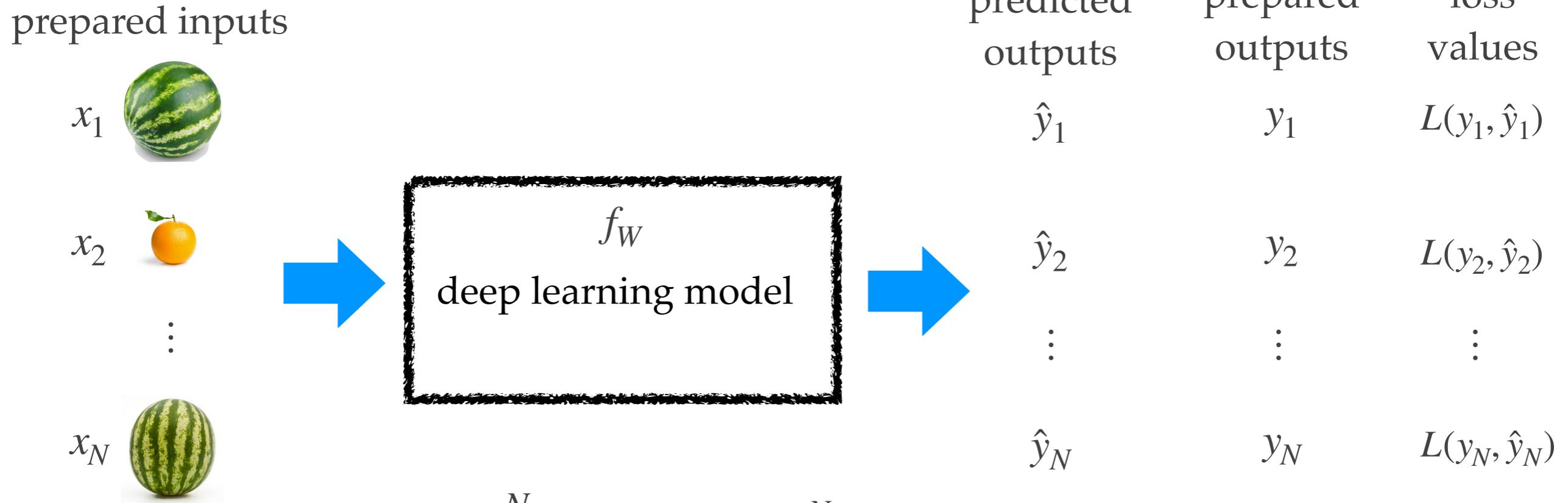
Furthermore, how do we compute a complicated loss
function such as the one in our example $L(y, f_W(x))$?

Could get stuck in
local minima

@#Q#@%!!!!!!

Backpropagation!

Training Our Model



$$\text{Total loss } L(y, \hat{y}) = \sum_{i=1}^N L(y_i, \hat{y}_i) = \sum_{i=1}^N L(y_i, f_W(x_i))$$

Update the weights

Compute the gradient of the total loss w.r.t. to W using backpropagation

$$\nabla_W L = \sum_{i=1}^N \nabla_W L(y_i, f_W(x_i))$$

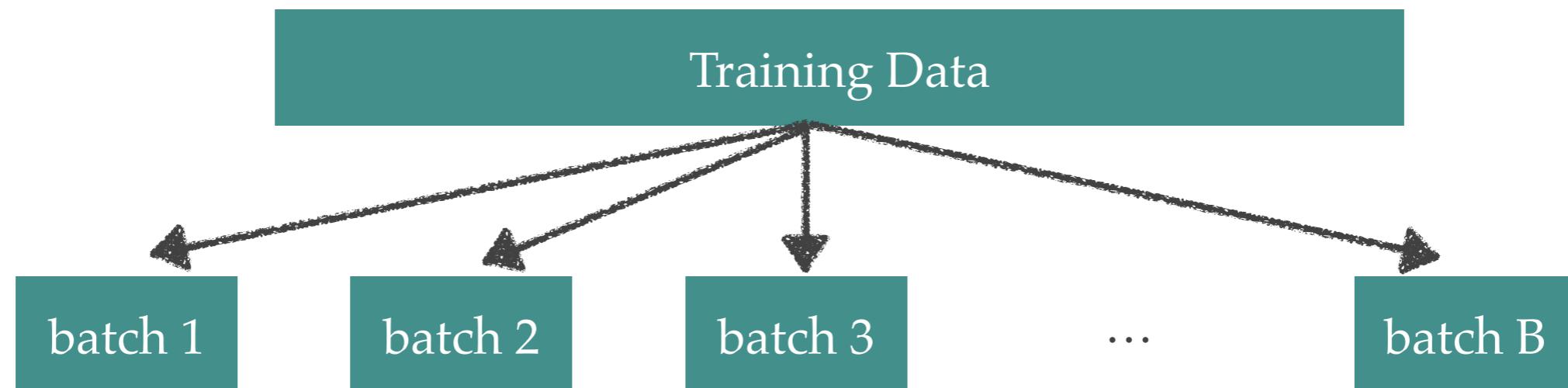
Assign the new weights $W \leftarrow W - \alpha \nabla_W L$

Stochastic Gradient Descent

- Every time we want to update W , we need to compute the loss functions and their gradients of all samples

$$\nabla_W L = \sum_{i=1}^N \nabla_W L(y_i, f_W(x_i))$$

- For very large N , we might not be able to do it due to both the time and memory constraints 14,000,000 images?
- For stochastic gradient descent (SGD), we update W based on subsets of samples called a **mini-batch**

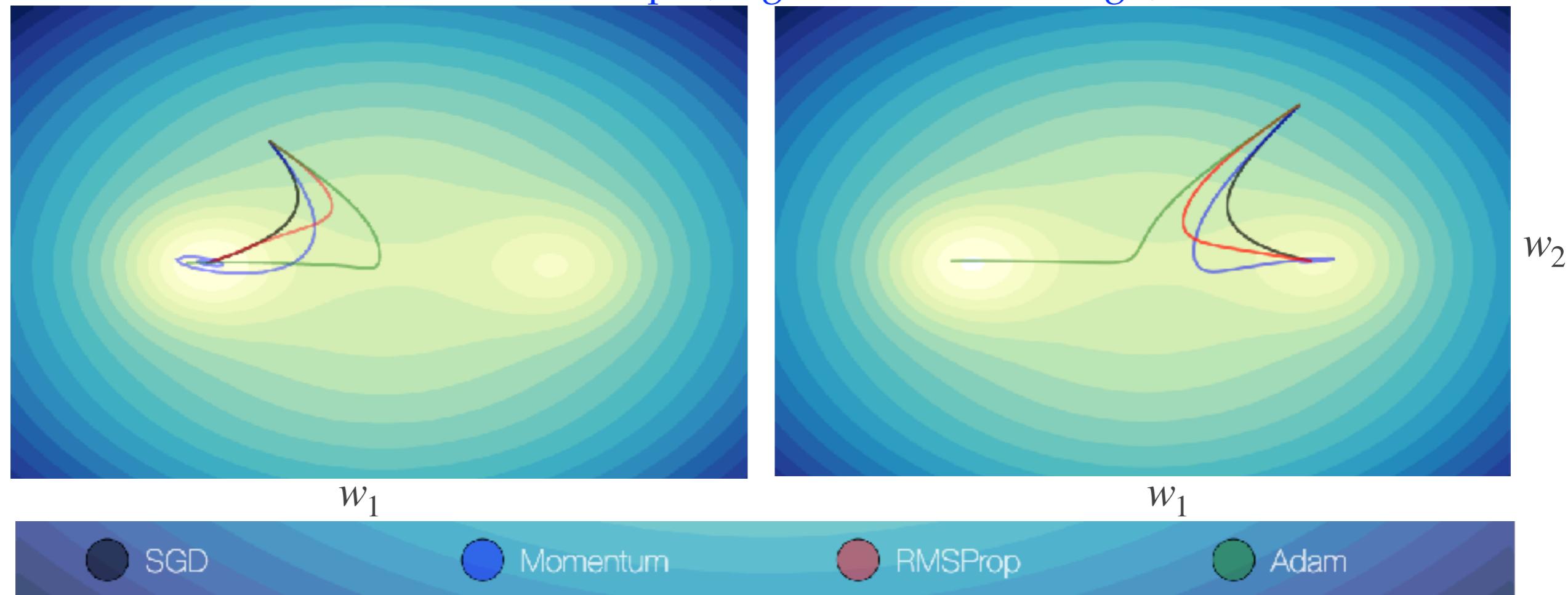


1 epoch = the model has seen the whole training data once (i.e., B batches in this case)
In this example, the weight matrix W is updated B times per epoch.

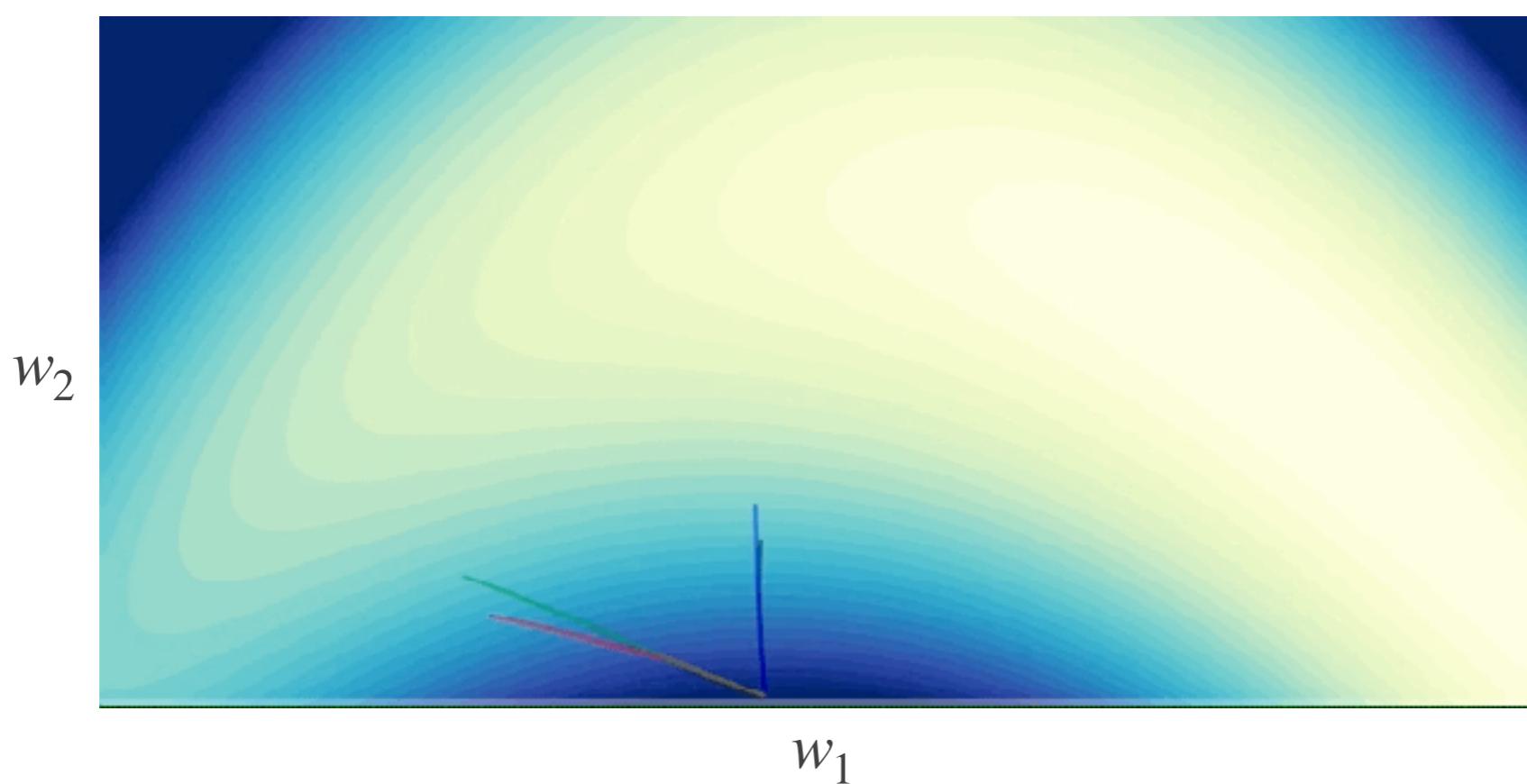
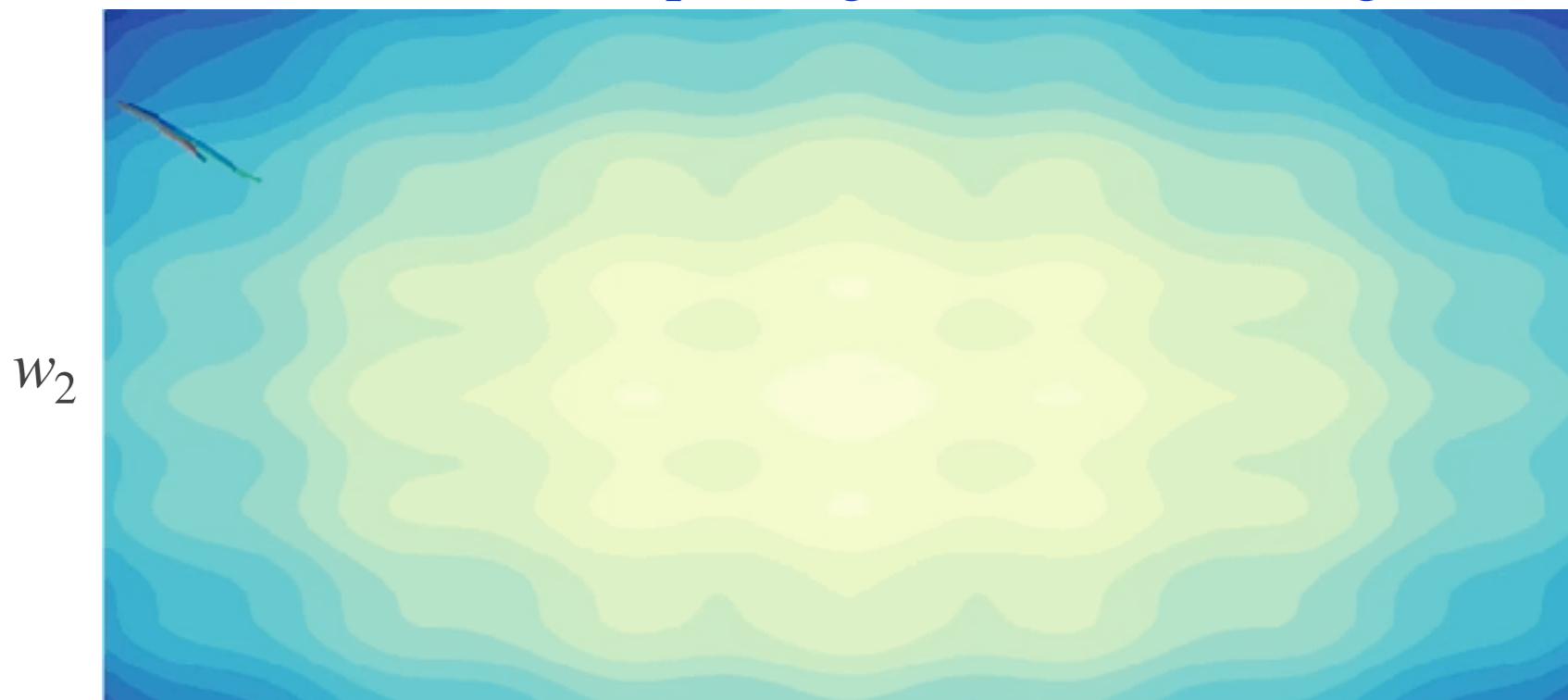
Optimizers

- ❖ Popular choices are
 - ❖ Stochastic gradient descent (SGD)
 - ❖ SGD with momentum
 - ❖ RMSprop
 - ❖ Adam
 - ❖ ADADELTA
 - ❖ AdaGrad

Loss landscape (bright = low, dark = high)



Loss landscape (bright = low, dark = high)





Fully connected layer (Dense)

Optimizer

SGD

Adam

RMSprop

Evaluation metric

accuracy

F1-score

AUC

confusion matrix

Loss function

categorical crossentropy

binary crossentropy

mean squared error

mean absolute error

Deep Learning

Convolutional layer

Conv1D, 2D, 3D, ...

separable Conv

Pooling layer

max-pooling

average-pooling

Activation function

sigmoid

softmax

ESP (swish)

ReLU

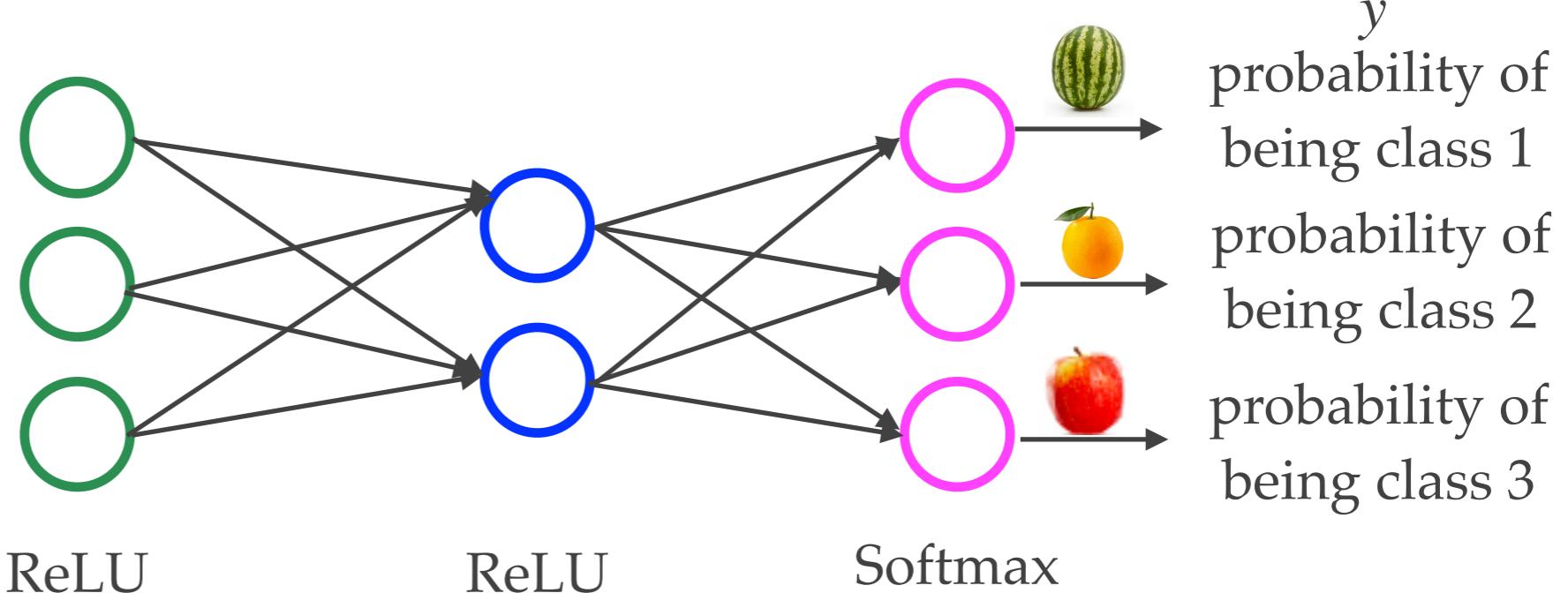
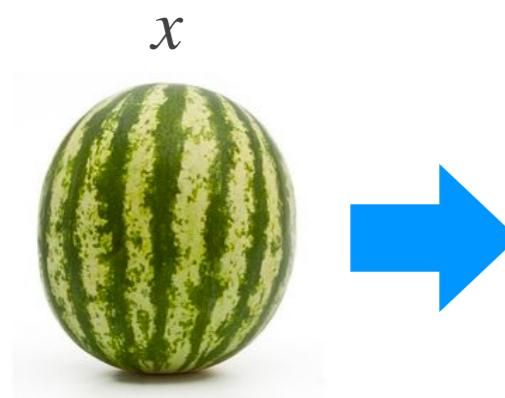
Regularization

Dropout

Data augmentation

l_1, l_2 regularizations

- ❖ **Combine basic components to build a neural network**
 - More components → “More” representative power

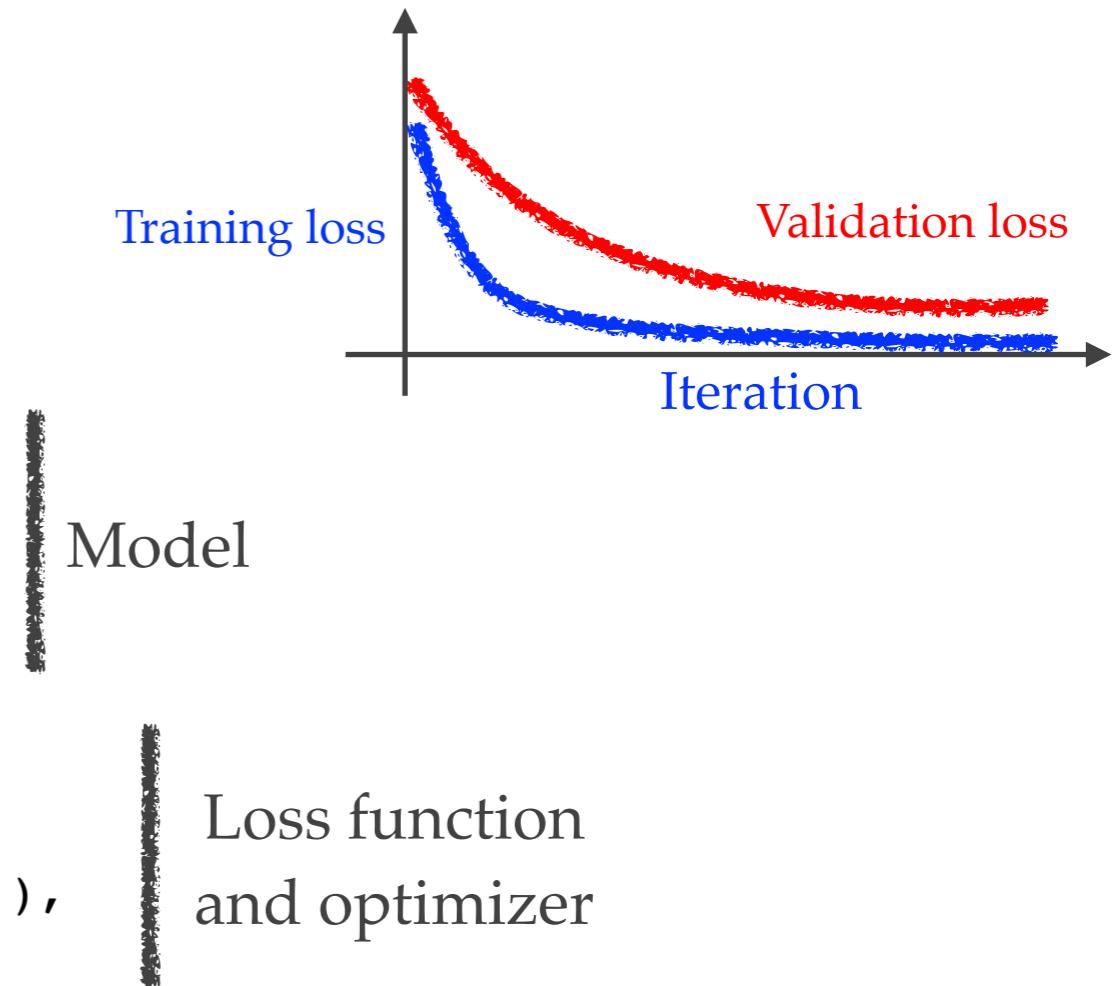


```
# Import necessary modules
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

# Create the model
model = keras.Sequential()
model.add(layers.Dense(3, activation="relu"))
model.add(layers.Dense(2, activation="relu"))
model.add(layers.Dense(3, activation="softmax"))

# Compile the model
model.compile(
    optimizer='adam',
    loss=tf.keras.losses.CategoricalCrossentropy(),
)

# Train the model for 100 epochs with a batch size of 32
model.fit(x_train, y_train, batch_size=32, epochs=100, validation_data=(x_val,y_val))
```



Outline

- ❖ Introduction to Supervised Machine Learning
 - ❖ AI vs ML vs DL
 - ❖ Traditional Machine Learning
 - ❖ Training, Validation, and Test Data
 - ❖ Overfitting and Underfitting
- ❖ Introduction to Supervised Deep Learning
 - ❖ Traditional ML vs Deep Learning
 - ❖ Artificial Neuron and Neural Network
 - ❖ Supervised Learning
- ❖ Deep Learning as a Function Approximator
- ❖ Deep Learning Components
 - ❖ Fully Connected Layers
 - ❖ Activation Functions
 - ❖ Optimization

Next Lecture

- ❖ Deep Learning Components
 - ❖ Regularization
 - ❖ ℓ_2 and ℓ_1 regularization
 - ❖ Dropout
 - ❖ Batch normalization
 - ❖ Convolutional layer
 - ❖ Pooling layer
- ❖ Convolutional Neural Network (CNN)
- ❖ Hands-on: MRI Tumor Classification Using CNN
 - ❖ <https://github.com/ichatnun/CMU-medical-imaging-deep-learning>