SMU Singapore Management University
School of Computing and Information Systems

19 to 22 Aug 2024

AI Bootcamp

James KOH Boon Yong

jameskohby@smu.edu.sg

Materials available @ https://smu.sg/9bvk / https://smu.sg/iq4x
password: KOH2024

Schedule

Date	Time	Topic
19 Aug	1.5 hr	Different branches of AI; Principles of supervised learning Linear and Logistic regression Fully connected neural networks and activations functions
	1.5 hr	Prediction using generated datapoints (code together) Practical considerations when developing model
20 Aug	1.5 hr	Gradient descent; Loss functions; What it means to 'train' a model Difference between parameters/hyperparameters What happens in the training process Concept of convolutions and feature extraction
	1.5 hr	Building own CNN from scratch for multiclass classification on CIFAR10 (code together)
21 Aug	1.0 hr	Using a pre-trained model, change network architecture, useful command lines, save and load (code together)
	2.0 hr	Group work. Walk around to guide each group and advise.
22 Aug	0.5 hr	Final preparations and touch-up
	1.5 hr	Group presentation
	1.0 hr	Comments and sharing of industry practices in actual company Suggestions of future directions and domains for students to dive deeper

About Me – James Koh 许文勇

School of Computing and Information Systems

- Work Experience
  - Machine Learning Engineer
    - at a Chinese AI start-up
  - AI Scientist
    - at a Singapore statutory board

LinkedIn

- Teaching Experience
  - Undergraduate (NTU)
    - during my PhD days
    - ✓ Best Teaching Assistant
  - Masters (SMU)
    - MITB AI-courses
    - ✓ Outstanding Instructor Award AY2022/23

https://jklmgroup.com

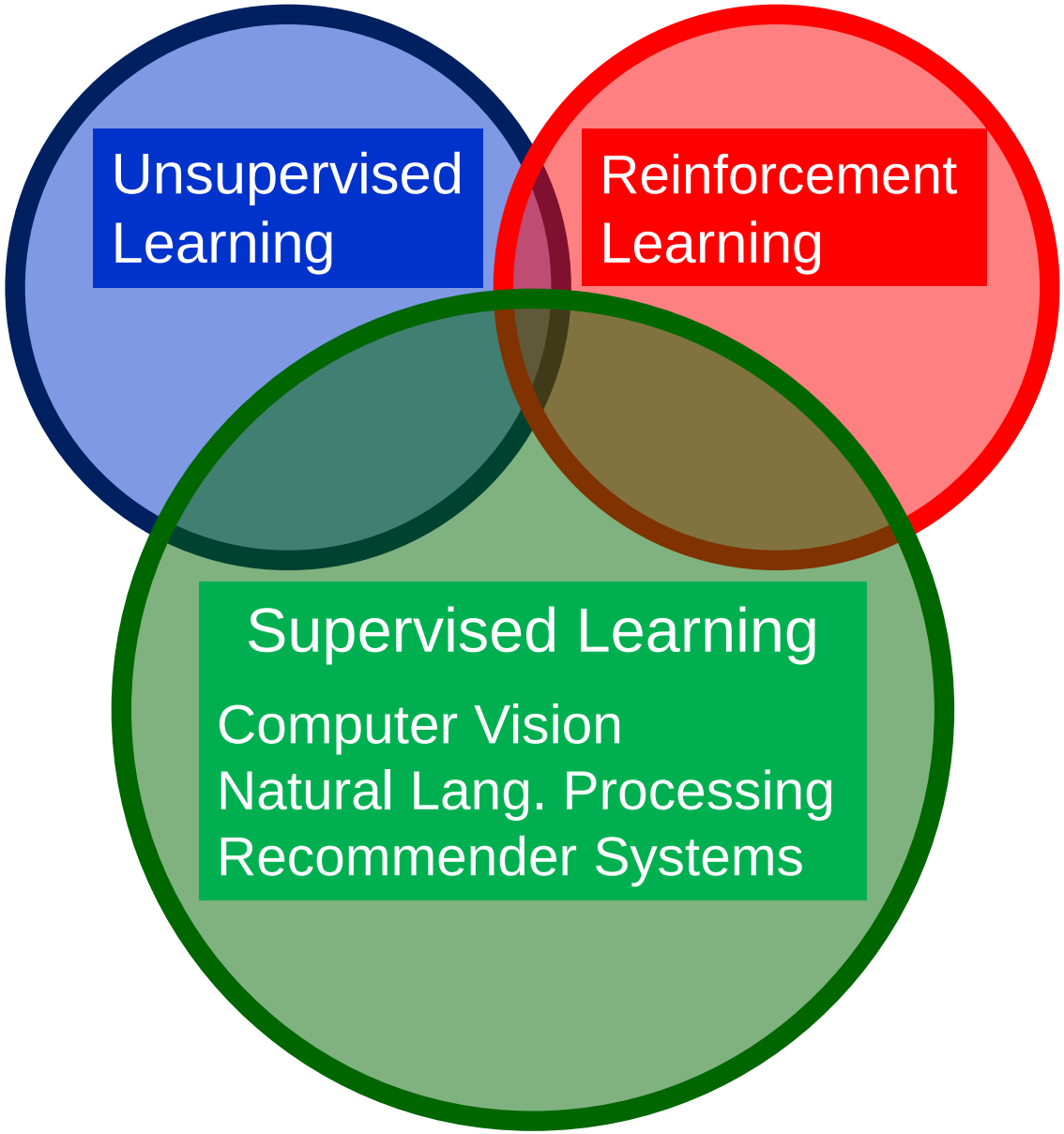
School of Computing and Information Systems

Day 1
(19 Aug 2024)

Outline for today

- Principles & examples of supervised learning
- Reality of modelling
- Linear regression (math & code)
- Logistic regression (math & code)
- Neural network
- Evaluation and considerations

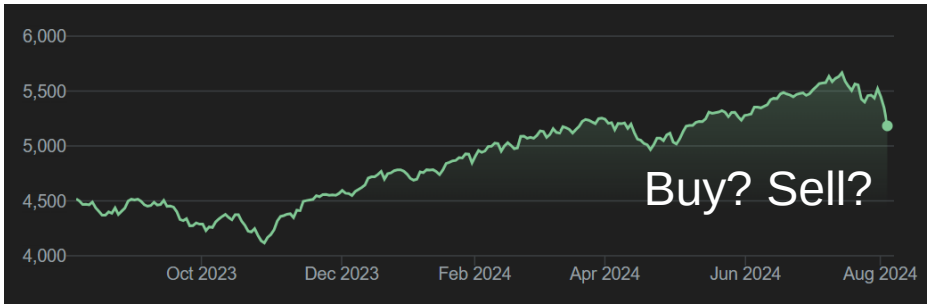
Overview


Unsupervised Learning
Reinforcement Learning
Supervised Learning
Computer Vision
Natural Lang. Processing
Recommender Systems

Principles of Supervised Learning

- Learning a mapping from observations to targets
  - If there are no patterns, prediction is not possible
- Labelled data
  - What if there are mistakes?
- Generalization to unseen data
  - Dealing with data drift

Examples



x = {Employed 3 years, $6000 income, outstanding car loan $50k, married, 2 children}
y = default?

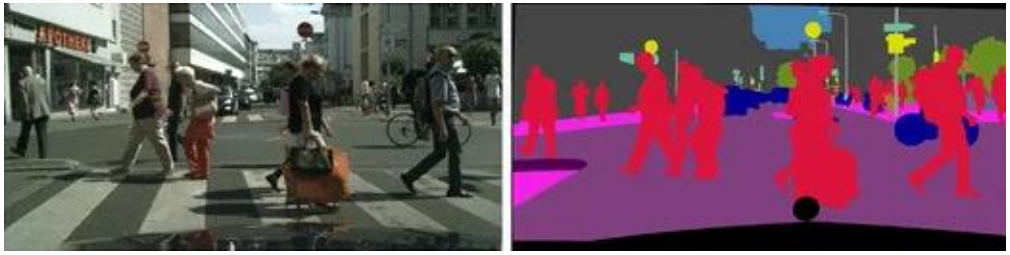
x = "*This chicken rice is not bad, although a little oily. Large serving size for the price.*"

y = rating?

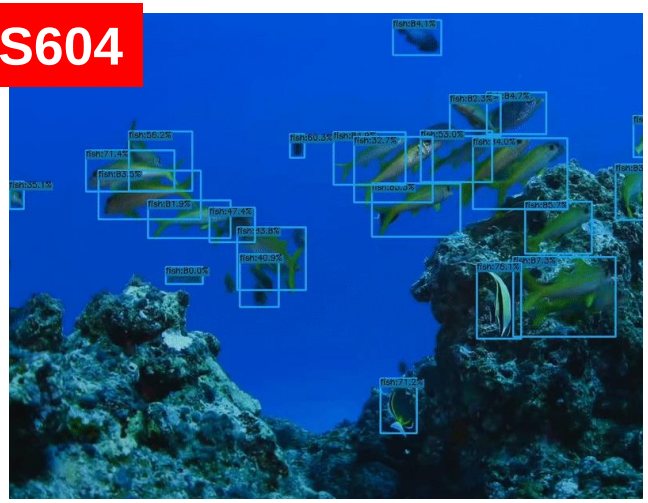
x = {Watched 'Crash Landing on You', 流星花园 & 'Squid Game'}

y = what to recommend?

Ivanovs *et al*. 2022, https://www.mdpi.com/1424-8220/22/6/2252



CS604



Reality

$$y = 537.2 + 26.5x_1 - 1.7x_2 + 4.1x_3 + 9.6x_4 + 3.0x_5 + \cdots + x_{9999}$$

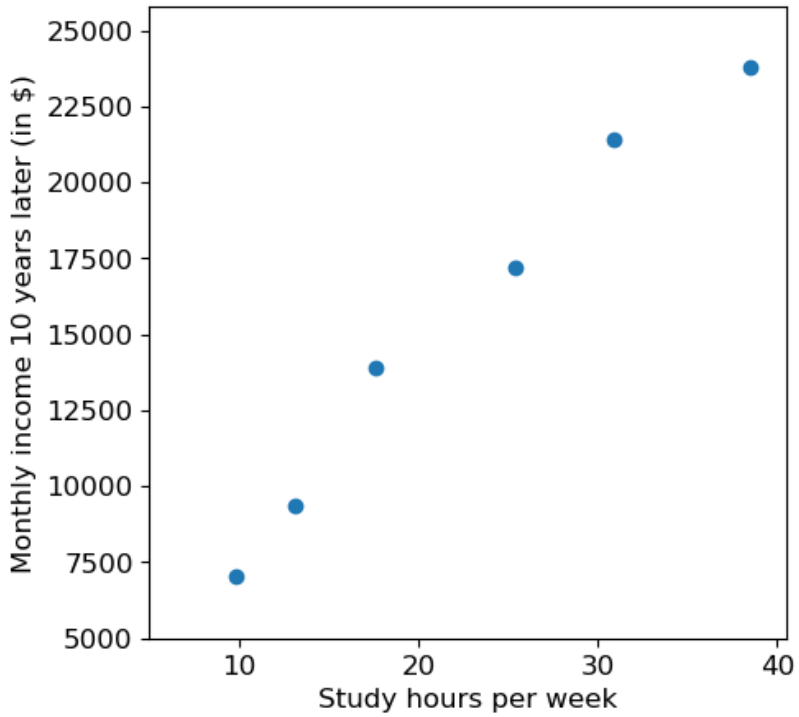
- Problem?
  - We do not have the full information of everything
  - Even IF we have everything, will be too complicated to solve

- Solution?
  - Consider only the important factors (that have access to)
  - Everything else is lumped into an 'error' term

$$y = w_0 + w_1 x_1 + \varepsilon$$

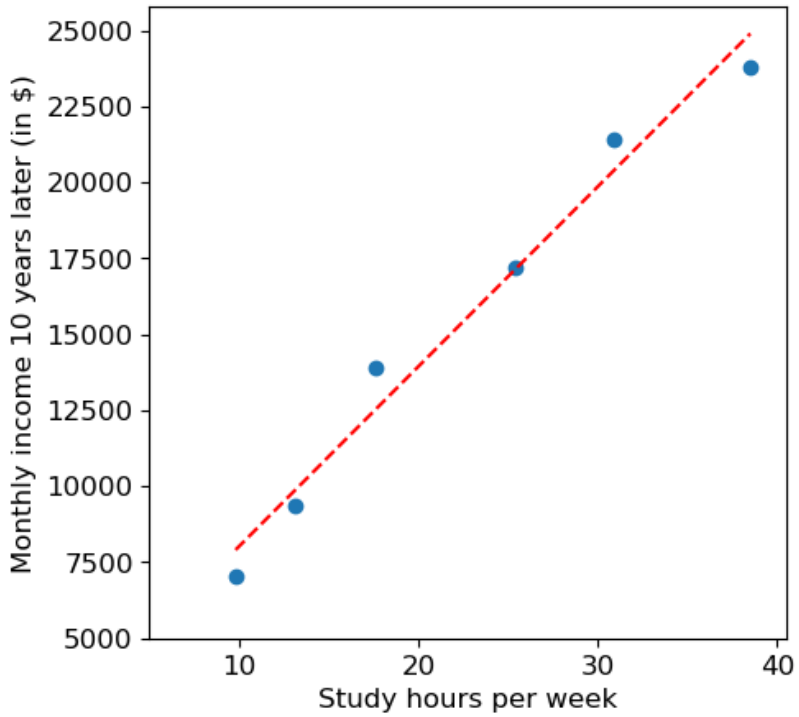
Linear Regression

Hours per week studying (x_1)	Monthly Salary in SGD (y)
9.8	7060
13.1	9350
17.6	13900
25.4	17200
30.9	21400
38.5	23800



Linear Regression

Hours per week studying (x_1)	Monthly Salary in SGD (y)
9.8	7060
13.1	9350
17.6	13900
25.4	17200
30.9	21400
38.5	23800



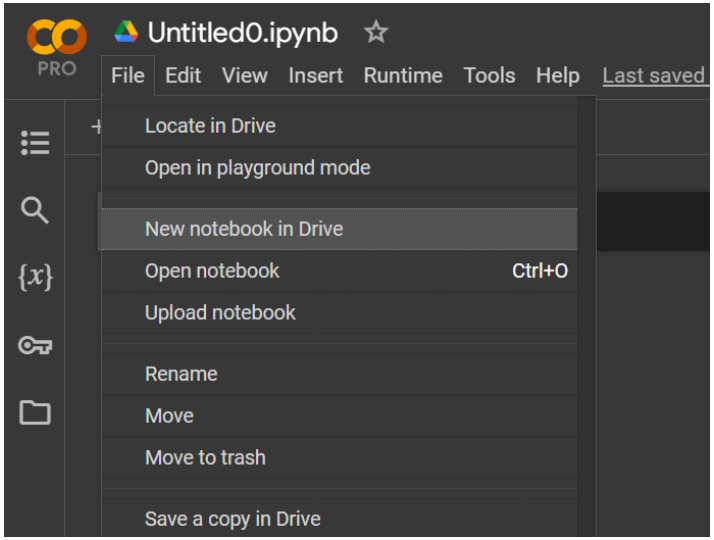
Google Colab

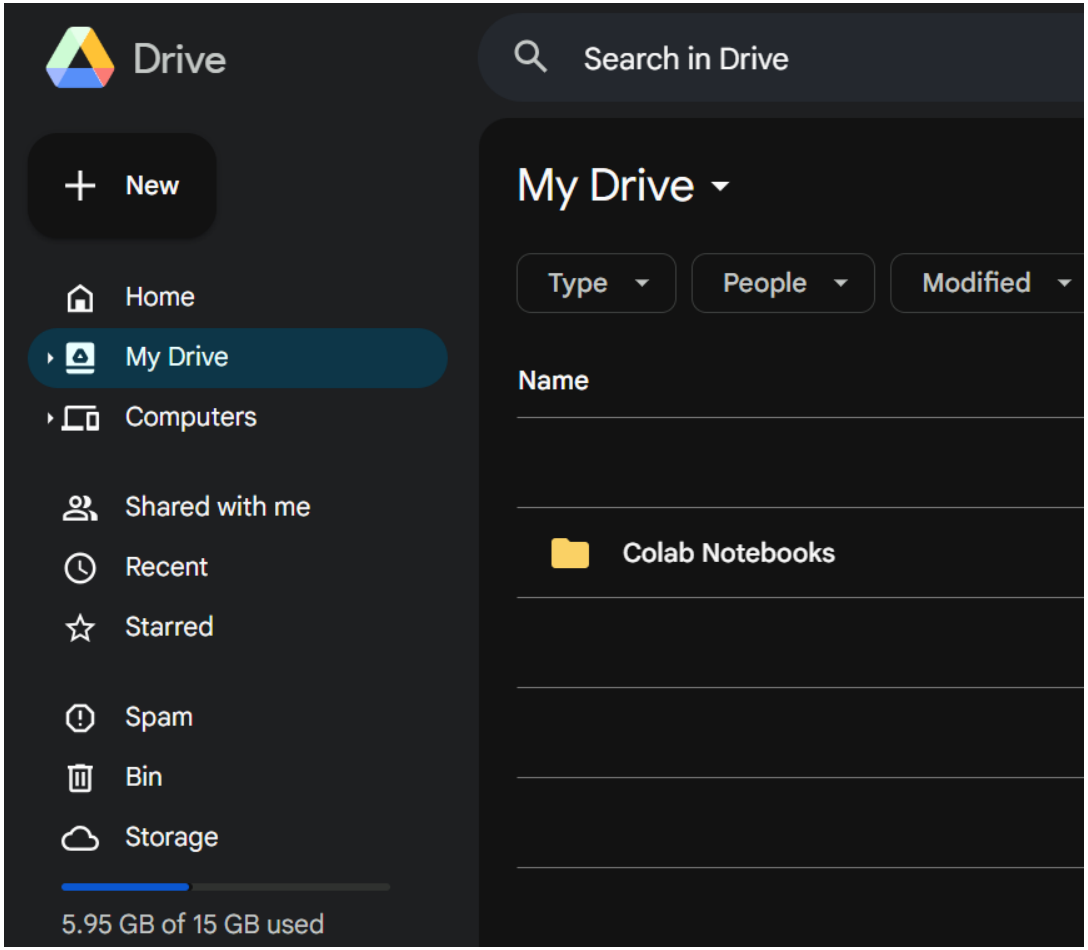
- Access to GPU even with free account
- Run computations on the cloud
  - Pre-installed libraries like pytorch
  - Integrate with Google Drive
  - Minimize strain/heat generation on your personal laptop

We will be using this together for this Summer Camp!

Google Colab

SMU Singapore Management University
School of Computing and Information Systems



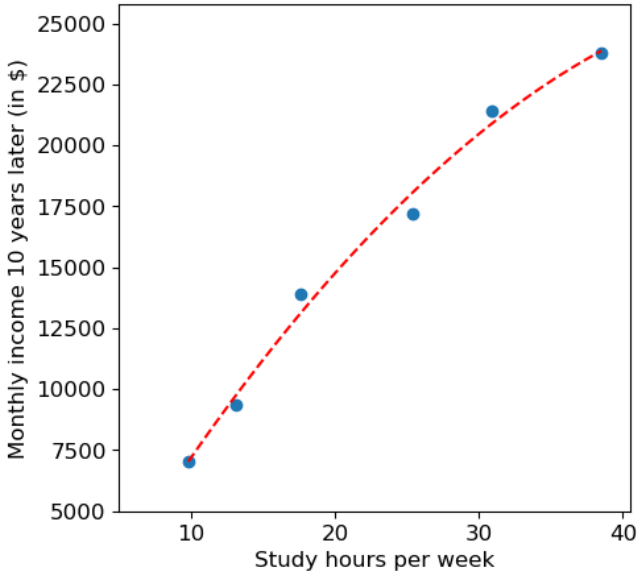


Hands-on coding


```python
import numpy as np
import matplotlib.pyplot as plt

x = np.array([9.8, 13.1, 17.6, 25.4, 30.9, 38.5])
y = np.array([7060, 9350, 13900, 17200, 21400, 23800])

coefficients = np.polyfit(x, y, 1)
slope, intercept = coefficients

x_fit = np.linspace(np.min(x), np.max(x), 50)
y_fit = slope * x_fit + intercept

plt.figure(figsize=(5,5))
plt.scatter(x, y)
plt.plot(x_fit, y_fit, 'r--')
plt.xlim([5, np.max(x)+2])
plt.ylim([5000, np.max(y)+2000])
plt.xlabel('Study hours per week')
plt.ylabel('Monthly income 10 years later (in $)')
plt.show()
```

Exploring np.polyfit & alternatives

- What needs to be changed for higher order fits?

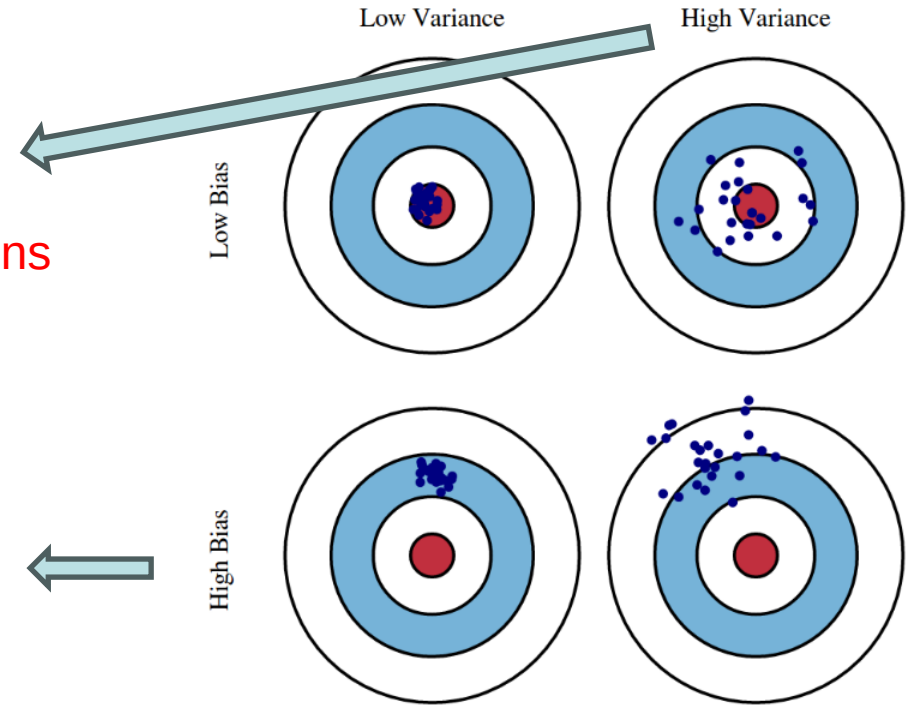


- Using sklearn
  - How do you check if the results are the same?

```
from sklearn.linear_model import LinearRegression

model = LinearRegression()
model.fit(x.reshape(-1, 1), y)

slope = model.coef_[0]
intercept = model.intercept_

x_fit = np.linspace(np.min(x), np.max(x), 50).reshape(-1, 1)
y_fit = model.predict(x_fit)
```

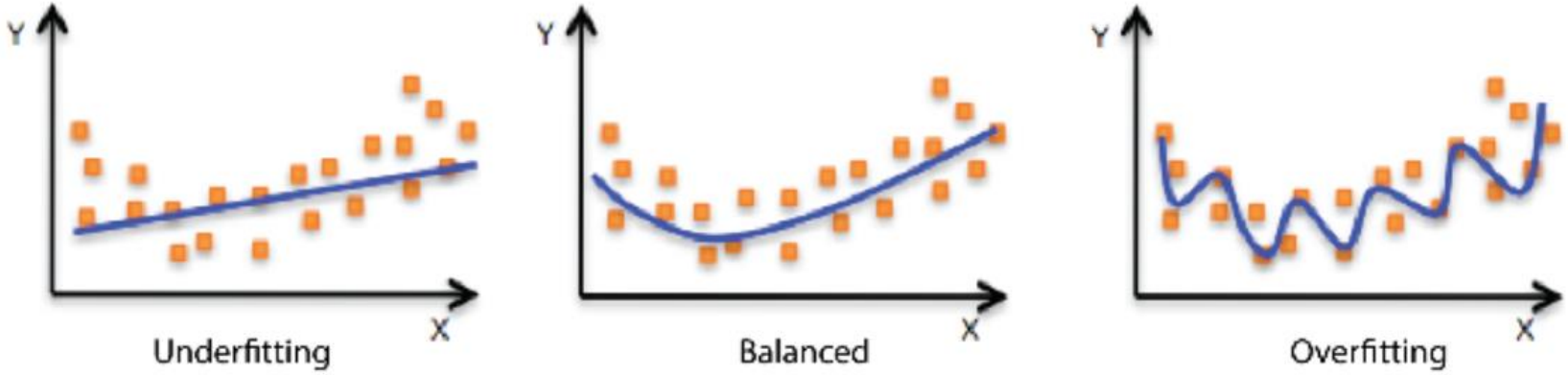
Bias-Variance Tradeoff

Model is overly complex and capturing noise in the training data; sensitive to small fluctuations



Model complexity is insufficient to capture underlying patters, leading to systematic errors

https://scott.fortmann-roe.com/docs/BiasVariance.html



https://docs.aws.amazon.com/machine-learning/latest/dg/model-fit-underfitting-vs-overfitting.html

Math in Linear Regression

- What is really happening?

  $h(x) = w_0 + w_1 x_1$

  Mean-squared error $J(w_0, w_1) = \frac{1}{n}\sum_{i=1}^{n}[h(x_i) - y_i]^2$

  Find w_0 and w_1 such that MSE is minimized

- How?

$$\frac{\partial J(w_0, w_1)}{\partial w_0} = \frac{2}{n}\sum_{i=1}^{n}(w_0 + w_1 x_i - y_i) = 0$$

$$\frac{\partial J(w_0, w_1)}{\partial w_1} = \frac{2}{n}\sum_{i=1}^{n} x_i(w_0 + w_1 x_i - y_i) = 0$$

$$w_0 = \frac{\sum_{i=1}^{n} x_i^2 \sum_{i=1}^{n} y_i - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} x_i y_i}{n\sum_{i=1}^{m} x_i^2 - (\sum_{i=1}^{n} x_i)^2} \qquad w_1 = \frac{n\sum_{i=1}^{n} x_i y_i - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i}{n\sum_{i=1}^{m} x_i^2 - (\sum_{i=1}^{n} x_i)^2}$$

Multiple variables

- Mathematically, can extend the single variable case

$$h(x) = w_0 + w_1 x_1$$

to

$$h(x) = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_d x_d$$

- For compactness, we define $x_0 = 1$

$$h(\boldsymbol{x}) = \sum_{j=0}^{d} w_j x_j = \boldsymbol{w}^T \boldsymbol{x}$$

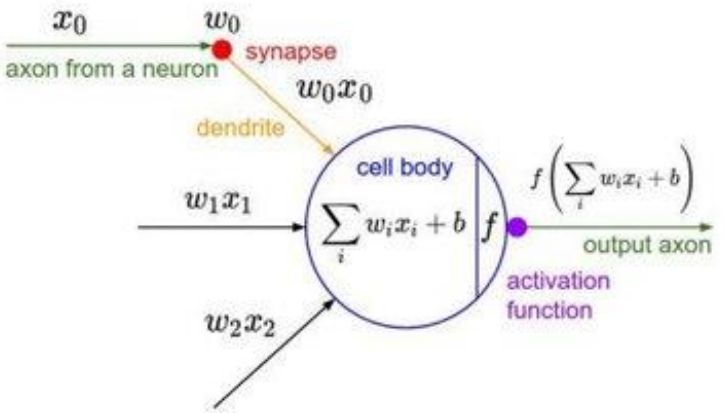
Quick high-school recap

- Why $h(x) = \boldsymbol{w}^T \boldsymbol{x}$

$$\boldsymbol{w}^T = [w_0 \quad \dots \quad w_d] \qquad \boldsymbol{x} = \begin{bmatrix} x_0 \\ \vdots \\ x_d \end{bmatrix}$$

$$\boldsymbol{w}^T \boldsymbol{x} = [w_0 \quad \dots \quad w_d] \begin{bmatrix} x_0 \\ \vdots \\ x_d \end{bmatrix} = w_0 + w_1 x_1 + \cdots + w_d x_d$$

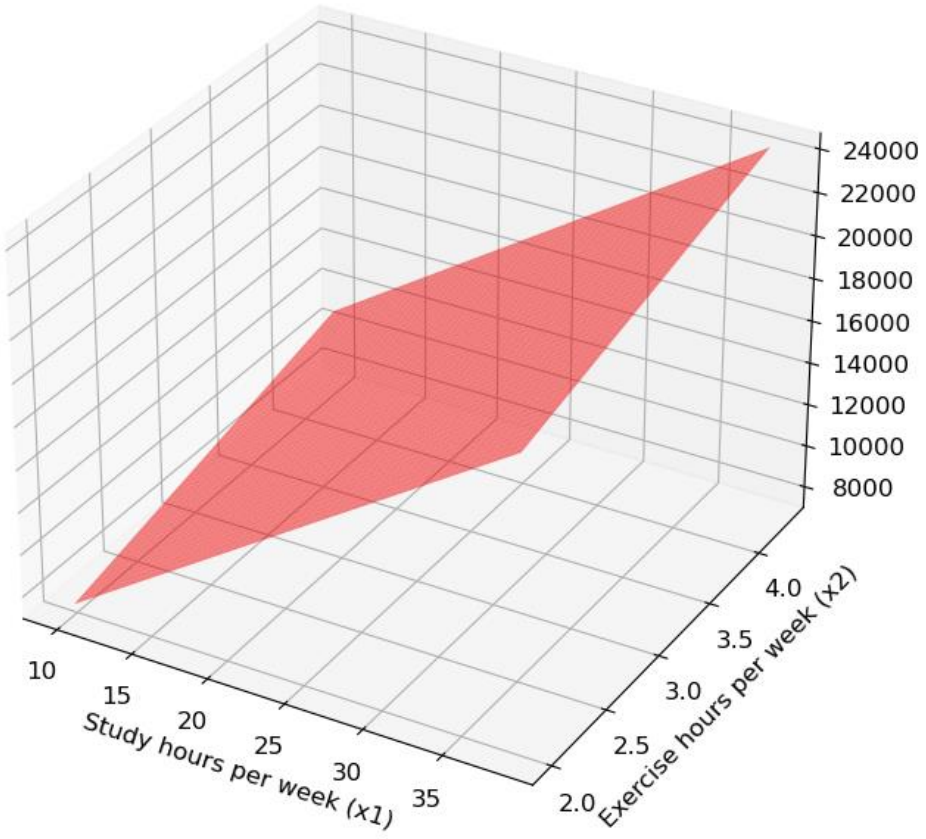
Teaser: It will be used later!

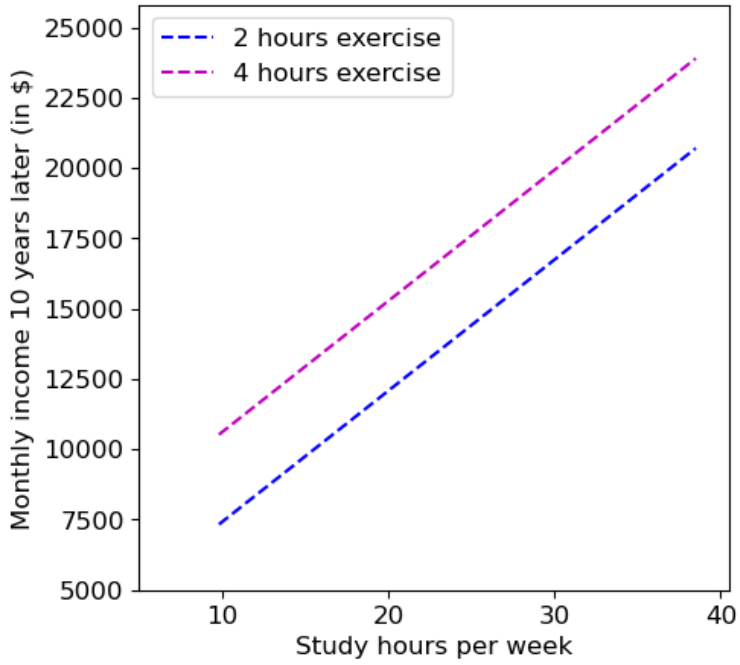


https://hal.science/hal-02075977v1/file/ANN_method.pdf

Multiple variables

- What happens if there are many independent variables?





Hands-on coding


```
model = LinearRegression()
model.fit(X, y)
coef = model.coef_
intercept = model.intercept_

x1_lin = np.linspace(np.min(x1), np.max(x1), 50)
x2_lin = np.linspace(np.min(x2), np.max(x2), 50)
x1_grid, x2_grid = np.meshgrid(x1_lin, x2_lin)
y_fit = coef[0] * x1_grid + coef[1] * x2_grid + intercept

fig = plt.figure(figsize=(8,8))
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(x1_fit, x2_fit, y_fit, color='red', alpha=0.5)
ax.set_xlabel('Study hours per week (x1)')
ax.set_ylabel('Exercise hours per week (x2)')
ax.tick_params(axis='z', pad=10)
plt.show()
```

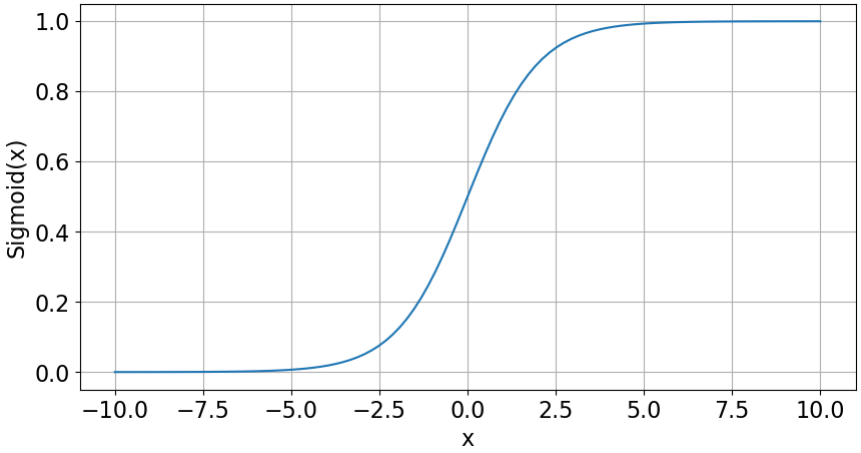
```
y_fit_x2_2 = coef[0] * x1_lin + coef[1] * 2 + intercept
y_fit_x2_4 = coef[0] * x1_lin + coef[1] * 4 + intercept

plt.figure(figsize=(5,5))
plt.plot(x_fit, y_fit_x2_2, 'b--', label='2 hours exercise')
plt.plot(x_fit, y_fit_x2_4, 'm--', label='4 hours exercise')
plt.xlim([5, np.max(x)+2])
plt.ylim([5000, np.max(y)+2000])
plt.xlabel('Study hours per week')
plt.ylabel('Monthly income 10 years later (in $)')
plt.legend()
plt.show()
```

Binary problems

- What if the outputs are either yes/success/present or no/failure/absent?
  - Can we still use the technique from before?
- Sigmoid function
  - value range from 0 to 1
  - slope varies

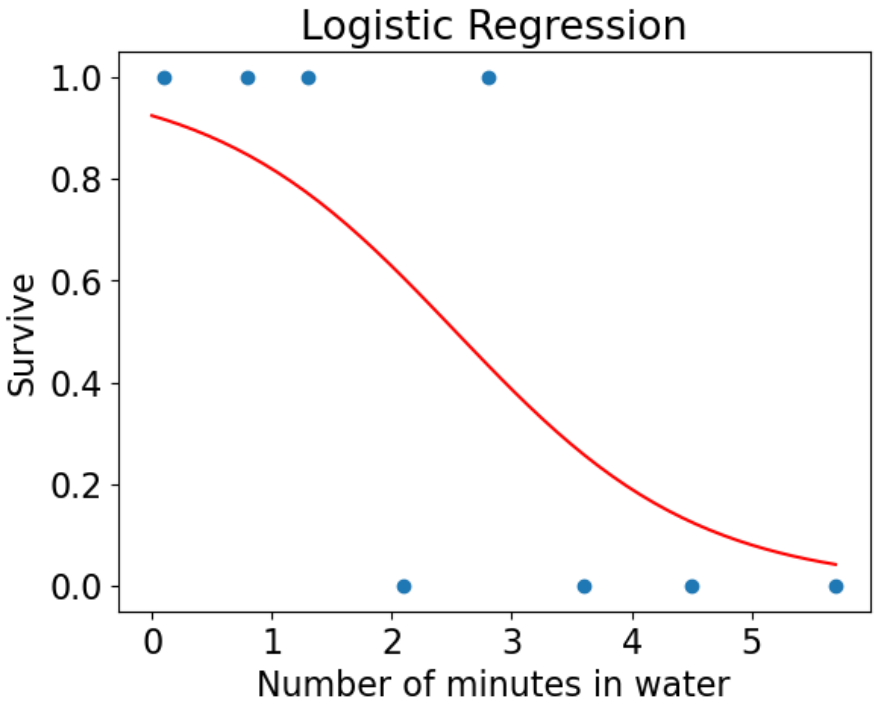
$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

```
def sigmoid(x):
    return 1 / (1 + np.exp(-x))
```



Logistic Regression

Number of minutes in water	Survive?
0.7	Yes
1.3	Yes
2.1	No
2.8	Yes
3.6	No
4.5	No



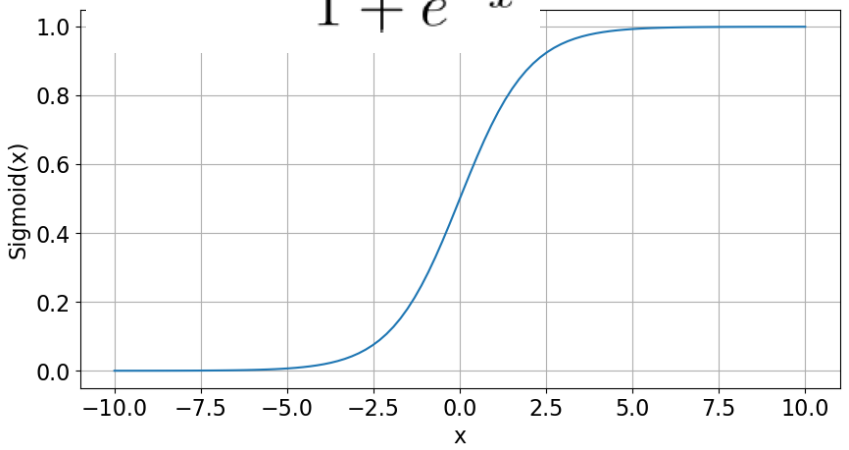
Hands-on coding


```
model = LogisticRegression()
model.fit(x, y)

x_fit = np.linspace(0, np.max(x), 50)
y_fit = model.predict_proba(x_fit.reshape(-1, 1))[:, 1]

plt.figure()
plt.scatter(x, y, label='Data points')
plt.xlim([-0.3,5.99])
plt.xlabel('Number of minutes in water')
plt.ylabel('Survive')
plt.title('Logistic Regression')
plt.show()

plt.figure()
plt.scatter(x, y, label='Data points')
plt.plot(x_fit, y_fit, color='red', label='Logistic regression fit')
plt.xlabel('Number of minutes in water')
plt.ylabel('Survive')
plt.title('Logistic Regression')
plt.show()
```

Mathematical model

- Earlier, we saw →

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



- In Linear Regression, used:

$$h(\boldsymbol{x}) = \sum_{j=0}^{d} w_j x_j = \boldsymbol{w}^T \boldsymbol{x}$$

- Now in Logistic Regression:

$$p(1|\boldsymbol{x}_i) = \frac{1}{1 + e^{-\boldsymbol{w}^\intercal \boldsymbol{x}_i}}$$

$$p(1|\boldsymbol{x}_i) + p(-1|\boldsymbol{x}_i) = 1$$

$$\frac{1}{1 + e^{-\boldsymbol{w}^\intercal \boldsymbol{x}_i}} + \frac{1}{1 + e^{\boldsymbol{w}^\intercal \boldsymbol{x}_i}}$$

$$p(y_i|\boldsymbol{x}_i) = \frac{1}{1 + e^{-y_i \boldsymbol{w}^\intercal \boldsymbol{x}_i}}$$

Likelihood

- Let's take a slight detour.
  - Given training data $D = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \cdots, (\boldsymbol{x}_n, y_n)\}$
  - Likelihood of observing D: $p(D|\boldsymbol{w}) = \prod_{i=1}^{n} p(y_i|\boldsymbol{x}_i; \boldsymbol{w})$
- Want to find that $\boldsymbol{w}^*$ maximizes $p(D|w)$
- In practice, maximize log likelihood rather than likelihood

$$\ln p(D|\boldsymbol{w}) = \sum_{i=1}^{n} \ln p(y_i|\boldsymbol{x}_i; \boldsymbol{w})$$

$$\frac{1}{1 + e^{-y_i \boldsymbol{w}^\top \boldsymbol{x}_i}}$$

Recall:

$p = a \cdot b \cdot c$

$\log(p) = \log(a \cdot b \cdot c) = \log(a) + \log(b) + \log(c)$

Intuition & Physical significance

- Let's suppose
  - universe has just three classes: dog, cat, and rabbit.
  - feature vector condensed to a single value ('large black nose')
  - have a dataset of 75 dogs, 59 cats and 66 rabbits.
  - feature described by either 0 or 1 (for simply; holds true even with floating values)

75 dogs	**No** large black nose	5 dogs (without)
	Has large black nose	70 dogs (with)
59 cats	**No** large black nose	39 cats (without)
	Has large black nose	20 cats (with)
66 rabbits	**No** large black nose	56 rabbits (without)
	Has large black nose	10 rabbits (with)

https://medium.com/mitb-for-all/intuition-behind-probabilities-from-supervised-learning-391a4eaf2ac6

Intuition & Physical significance

- A large black nose might correspond to dog/cat/rabbit → the model cannot be perfectly accurate in all predictions.

ID	Learnable features	Truth y	Prediction $\hat{y}$	Loss
1	**No** large black nose	Dog (1, 0, 0)	(a, b, c)	log(a) + 0 + 0
2		Cat (0, 1, 0)	(a, b, c)	0 + log(b) + 0
		Rabbit (0, 0, 1)	(a, b, c)	0 + 0 + log(c)
		Cat (0, 1, 0)	(a, b, c)	0 + log(b) + 0
100		Rabbit (0, 0, 1)	(a, b, c)	0 + 0 + log(c)
101	**Has** large black nose	Dog (1, 0, 0)	(d, e, f)	log(d) + 0 + 0
102		Dog (1, 0, 0)	(d, e, f)	log(d) + 0 + 0
		Rabbit (0, 0, 1)	(d, e, f)	0 + 0 + log(f)
		Cat (0, 1, 0)		
200		Dog (1, 0, 0)		

Aggregated	Learnable features	Truth	Loss	Optimal bet from the model
1 – 100	No large black nose	5% dog 39% cat 56% rabbit	5 log(a) + 39 log(b) + 56 log(c)	$\hat{y}$ = (0.05, 0.39, 0.56)
101 – 200	**Has** large black nose	70% dog 20% cat 10% rabbit	70 log(d) + 20 log(e) + 10 log(f)	$\hat{y}$ = (0.70, 0.20, 0.10)

https://medium.com/mitb-for-all/intuition-behind-probabilities-from-supervised-learning-391a4eaf2ac6

Logistic Regression

- Building a logistic regression model is actually finding $\boldsymbol{w}^*$ which maximizes log likelihood. $\sum_{i=1}^{n} \ln p(y_i|\boldsymbol{x}_i; \boldsymbol{w})$

- The words can be expressed in Math as:

$$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} \sum_{i=1}^{n} \ln(1 + e^{-y_i \boldsymbol{w}^\intercal \boldsymbol{x}_i})$$

Negative log-likelihood

- Solved by numerical methods, unlike linear regression which has a closed form solution

Neural Networks

- Many decisions
  - Architecture
  - Activation functions
  - Loss functions
  - Optimizer
  - Learning rate schedulers
  - Evaluation metrics
- For each combination, still have hyperparameter tuning of base learning rate, batch size.
- Apart from building the model, there's still what happens *before* and *after*.

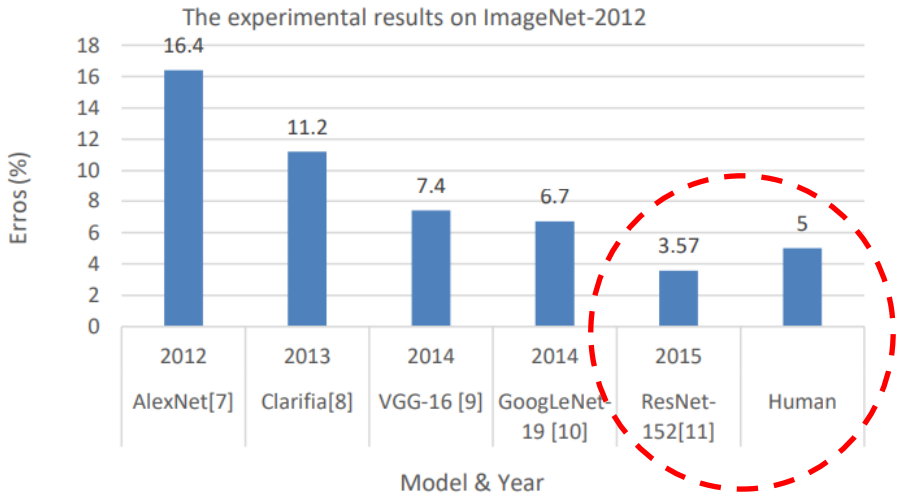
Background

- First proposed ~80 years ago (McCulloch & Pitts, 1943)
  - Lacks training mechanism, no layers
- Perceptron, by Cornell University psychologist (Rosenblatt, 1957)
- Exponential increase in compute power since 1990's
  - Moore's law
- Ability to train large models at reasonable costs
- Works well in practice

McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, *5*, 115-133.
Rosenblatt, F. (1957). The perceptron, a perceiving and recognizing automaton Project Para. Cornell Aeronautical Laboratory.

Effectiveness of neural networks

SMU Singapore Management University
School of Computing and Information Systems

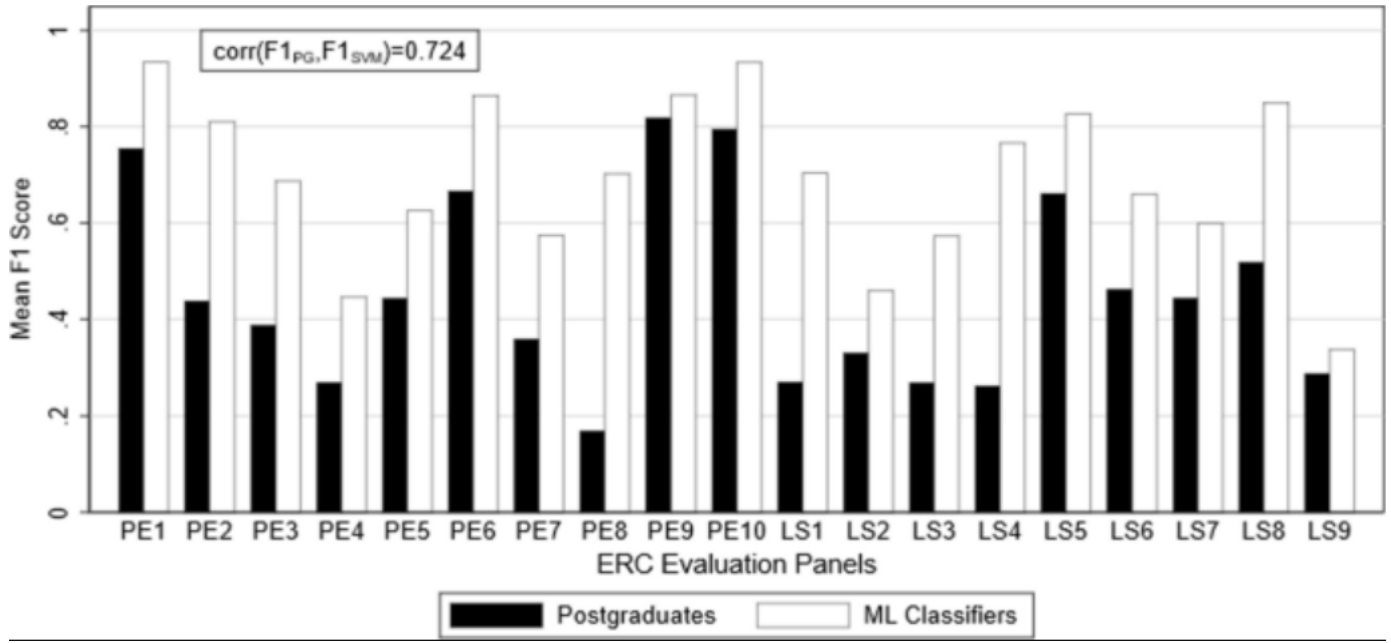


On image classification

Alom *et al*. 2018, https://arxiv.org/pdf/1803.01164

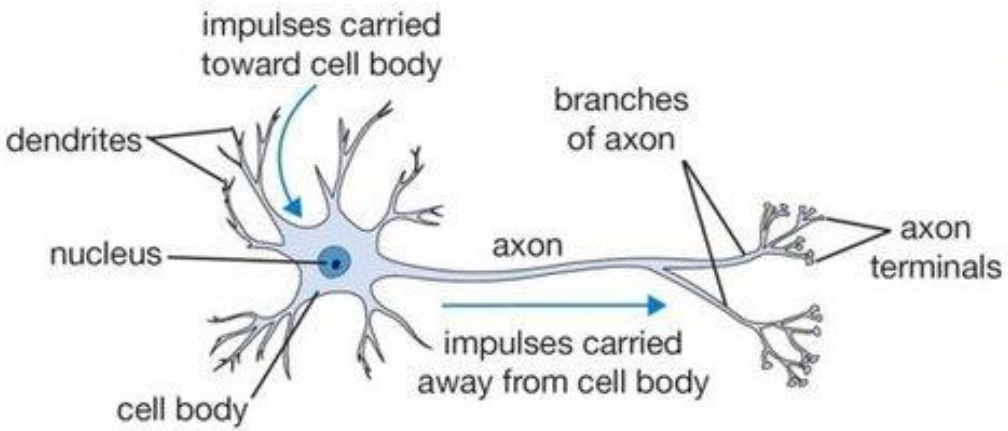
On scientific research abstracts

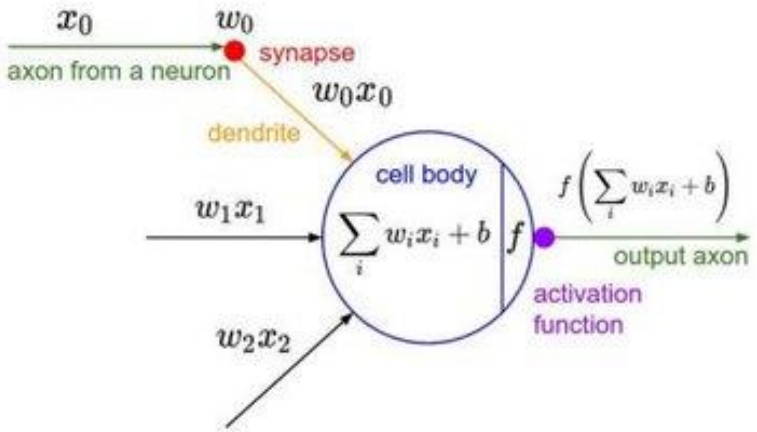
Yeow *et al*. 2020, https://link.springer.com/article/10.1007/s11192-020-03614-2



Motivation

- Nodes (artificial neurons) are connected to loosely model the way neurons in a biological brain are linked.
  - Sufficient input signals will trigger an output response
- Each node is simply a logistic unit
  - Weighted combinations, passed through an activation function

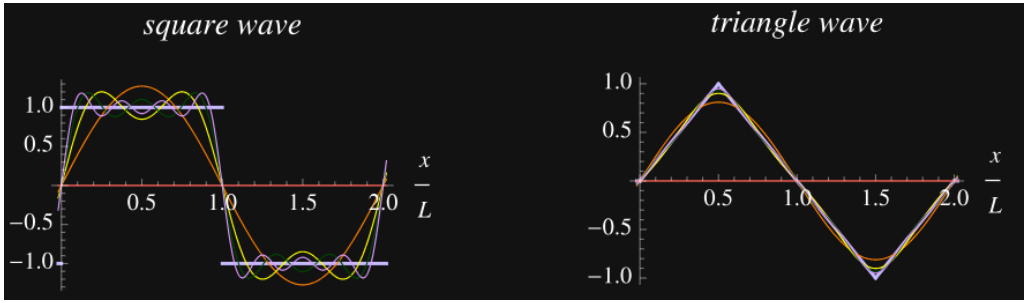




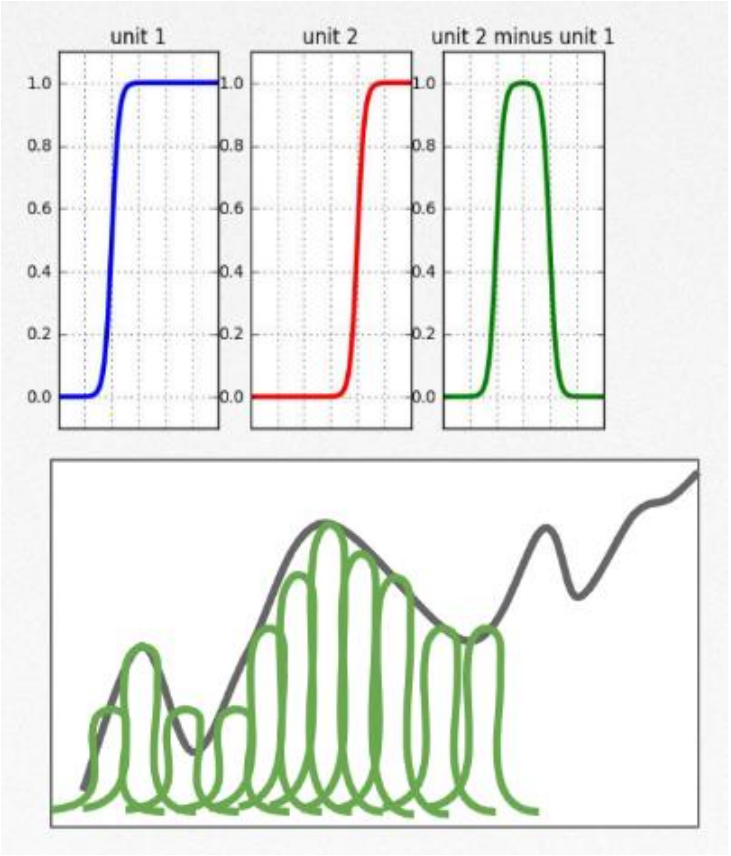
Konaté (2019). https://hal.science/hal-02075977v1/file/ANN_method.pdf

Universal Function Approximators

- Recall that from Fourier series, any periodic function can be represented using sines and cosines.
- By putting together many non-linear functions, can represent any arbitrary pattern.

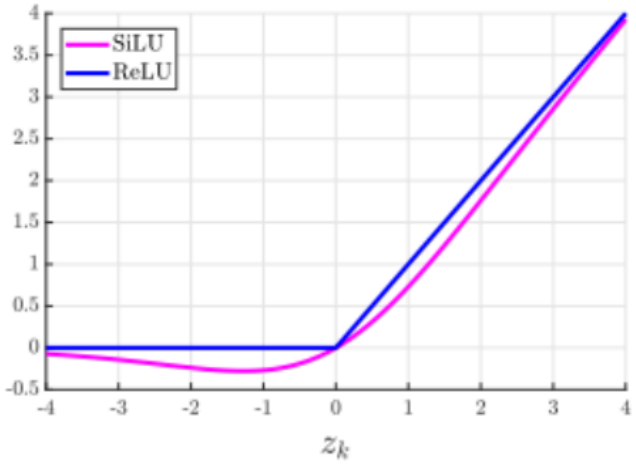


https://mathworld.wolfram.com/FourierSeries.html



Neural Networks Foundations by Simon Osindero (DeepMind), UCL

Types of Activation



VGGNet	ReLU	Simonyan & Zisserman, 2015
GoogleNet	ReLU	Szegedy et al., 2015
SqueezeNet	ReLU	Golkov et al., 2016
ResNet	ReLU	He et al., 2016
MobileNets	ReLU	Howard et al., 2017

ReLU $$f(x) = max(0, x) = \begin{cases} x_i, & if\ x_i \geq 0 \\ 0, & if\ x_i < 0 \end{cases}$$

LReLU $$f(x) = \alpha x + x = \begin{cases} x & if\ x > 0 \\ \alpha x & if\ x \leq 0 \end{cases}$$

PReLU $$f(x_i) = \begin{pmatrix} x_i\ , & if\ x_i > 0 \\ a_i x_i\ , & if\ x_i \leq 0 \end{pmatrix}$$

RReLU $$f(x_i) = \begin{pmatrix} x_{ji}\ , & if\ x_{ji} \geq 0 \\ a_{ji} x_{ji}\ , & if\ x_{ji} < 0 \end{pmatrix}$$

SReLU $$f(x) = \begin{pmatrix} t^r_i + a^r{}_i (x_i - t^r{}_i), & x_i \geq t^r{}_i \\ x_i\ , & t^r_i > x_i > t^l_i \\ t^l{}_i + a^l{}_i \left(x_i - t^l{}_i\right), & x_i \leq t^l{}_i \end{pmatrix}$$

ELU $$f(x) = \begin{pmatrix} x, & if\ x > 0 \\ \alpha\ exp(x) - 1, & if\ x \leq 0 \end{pmatrix}$$

PELU $$f(x) = \begin{pmatrix} cx, & if\ x > 0 \\ \alpha\ exp^{\frac{x}{b}} - 1, & if\ x \leq 0 \end{pmatrix}$$

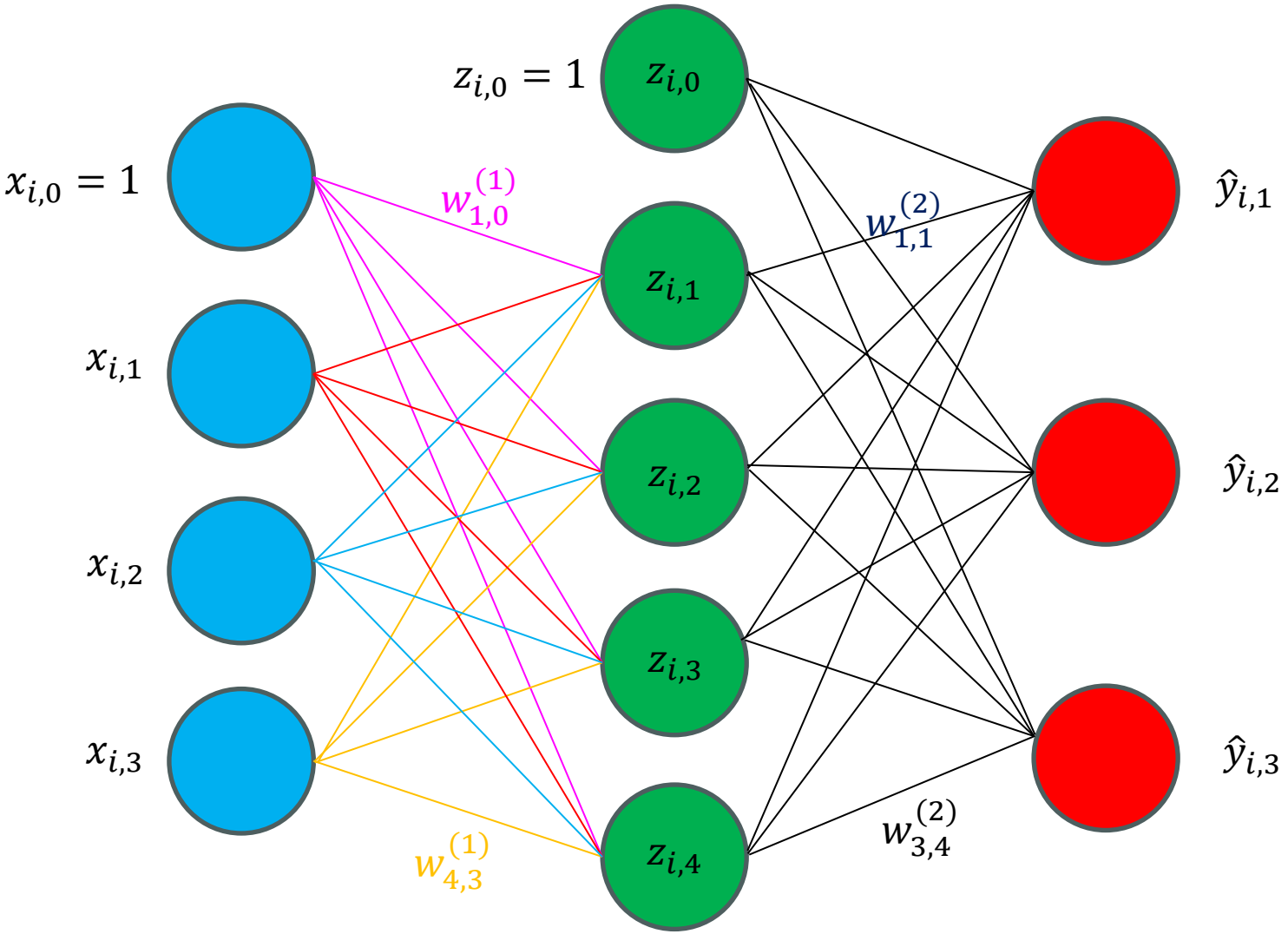
Swish $$f(x) = x \cdot sigmoid(x) = \frac{x}{1 + e^{-x}}$$

ELiSH $$f(x) = \begin{cases} \left(\frac{x}{1+e^{-x}}\right), & x \geq 0 \\ \left(\frac{e^x - 1}{1+e^{-x}}\right), & x < 0 \end{cases}$$

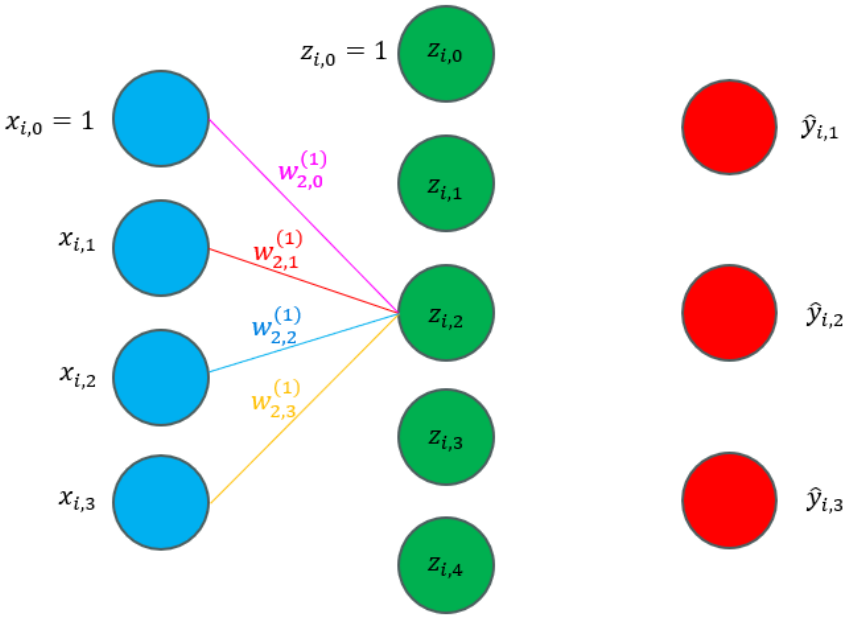
HardELiSH $$f(x) = \begin{cases} x \times max\left(0, min\left(1, \left(\frac{x+1}{2}\right)\right)\right), & x \geq 0 \\ (e^x - 1) \times max\left(0, min\left(1, \left(\frac{x+1}{2}\right)\right)\right), & x < 0 \end{cases}$$

Nwankpa et al. 2018 https://arxiv.org/pdf/1811.03378

Example of Neural Network



Example of Neural Network



$$\vdots$$

$$z_{i,2} = f\left(w_{2,0}^{(1)} x_{i,0} + \ \ldots \ + w_{2,4}^{(1)} x_{i,4}\right)$$

$$\vdots$$

$$\hat{y}_{i,1} = \sigma\left(w_{1,1}^{(2)} z_{i,1} + \ \ldots \ + w_{1,5}^{(2)} z_{i,5}\right)$$

$$\hat{y}_{i,2} = \sigma\left(w_{2,1}^{(2)} z_{i,1} + \ \ldots \ + w_{2,5}^{(2)} z_{i,5}\right)$$

$$\hat{y}_{i,3} = \sigma\left(w_{3,1}^{(2)} z_{i,1} + \ \ldots \ + w_{3,5}^{(2)} z_{i,5}\right)$$

More compactly, can write as:

$$z_{i,m} = f\left(\sum_{j=0}^{d} w_{m,j}^{(1)} x_{i,j}\right)$$

$$\hat{y}_{i,k} = \sigma\left(\sum_{j=0}^{m} w_{k,j}^{(2)} z_{i,j}\right)$$

Matrix Form

From the previous slide, we have:

$$z_{i,m} = f\left(\sum_{j=0}^{d} w_{m,j}^{(1)} x_{i,j}\right)$$

$$\hat{y}_{i,k} = \sigma\left(\sum_{j=0}^{m} w_{k,j}^{(2)} z_{i,j}\right)$$

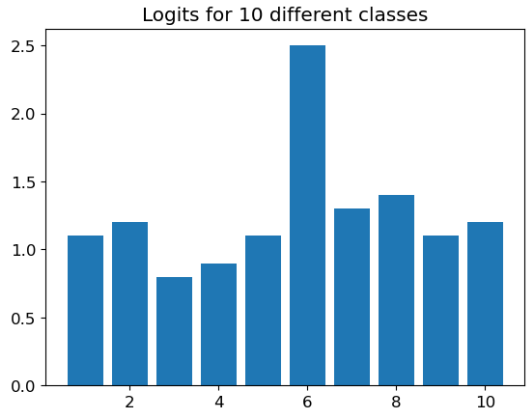
In Matrix form

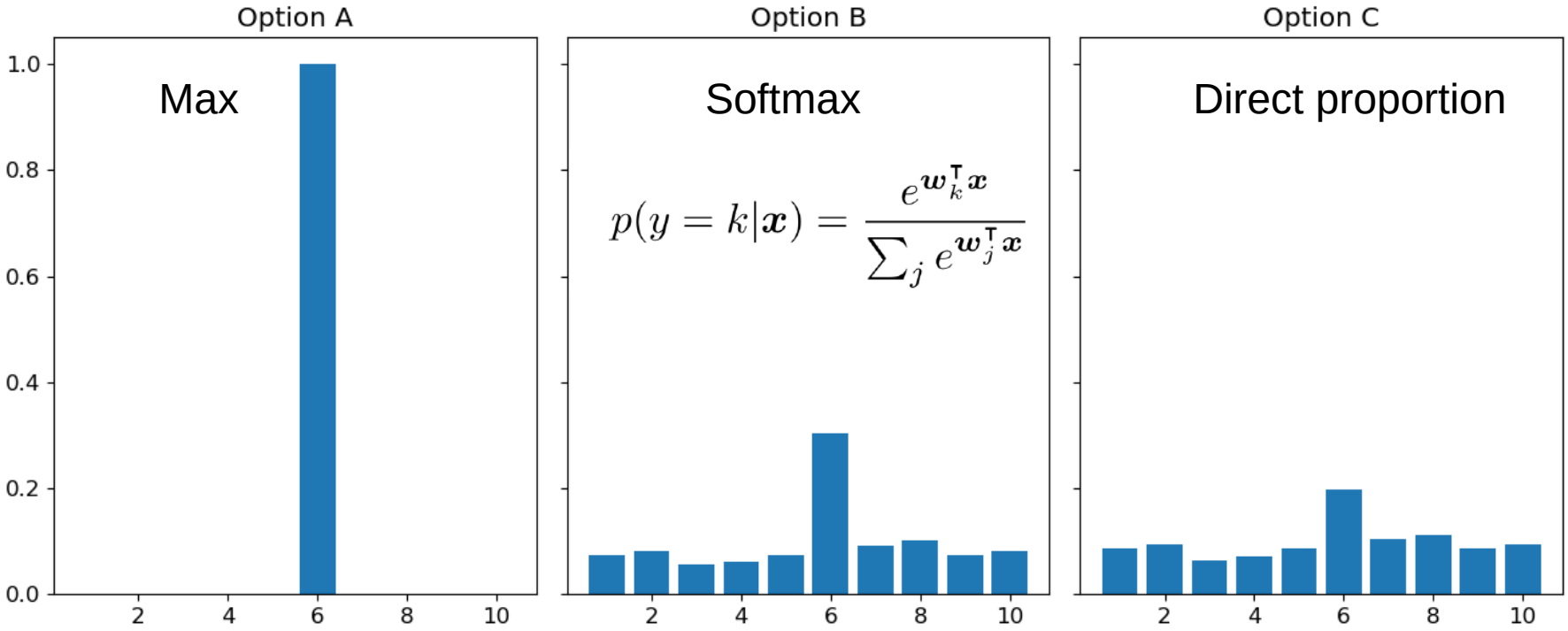
$$\mathbf{z}_i = f\left(\mathbf{w}^{(1)}\mathbf{x}_i\right) \qquad \mathbf{z}_i = \begin{bmatrix} z_{1,i} \\ \vdots \\ z_{m,i} \end{bmatrix} = f\left(\begin{bmatrix} w_{1,0}^{(1)} & w_{1,1}^{(1)} & \cdots & w_{1,d}^{(1)} \\ w_{2,0}^{(1)} & w_{2,1}^{(1)} & \cdots & w_{2,d}^{(1)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m,0}^{(1)} & w_{m,1}^{(1)} & \cdots & w_{m,d}^{(1)} \end{bmatrix} \begin{bmatrix} 1 \\ x_{1,i} \\ \vdots \\ x_{d,i} \end{bmatrix}\right)$$

$$\hat{\mathbf{y}}_i = \sigma\left(\mathbf{w}^{(2)}\mathbf{z}_i\right) \qquad \hat{\mathbf{y}}_i = \begin{bmatrix} \hat{y}_{1,i} \\ \vdots \\ \hat{y}_{k,i} \end{bmatrix} = \sigma\left(\begin{bmatrix} w_{1,0}^{(2)} & w_{1,1}^{(2)} & \cdots & w_{1,m}^{(2)} \\ w_{2,0}^{(2)} & w_{2,1}^{(2)} & \cdots & w_{2,m}^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{k,0}^{(2)} & w_{k,1}^{(2)} & \cdots & w_{k,m}^{(2)} \end{bmatrix} \begin{bmatrix} 1 \\ z_{1,i} \\ \vdots \\ z_{m,i} \end{bmatrix}\right)$$

Multiclass classification

- Probabilities add up to 1
- Use Softmax
  - In what way is it 'soft'?

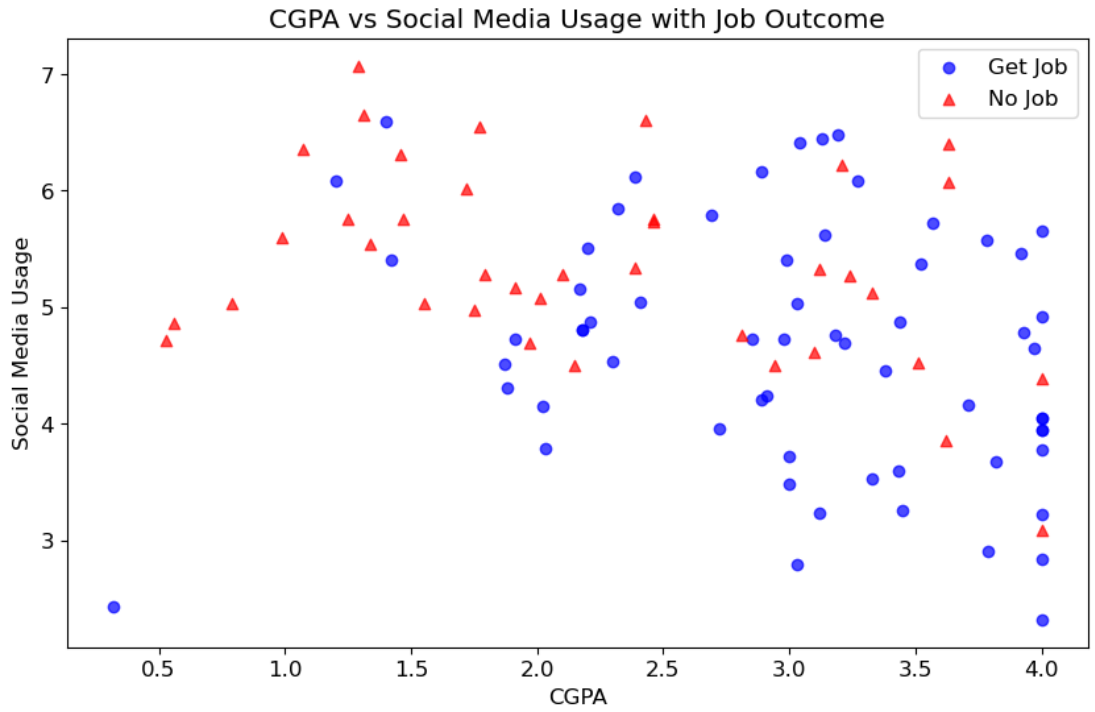




Classify job search outcome

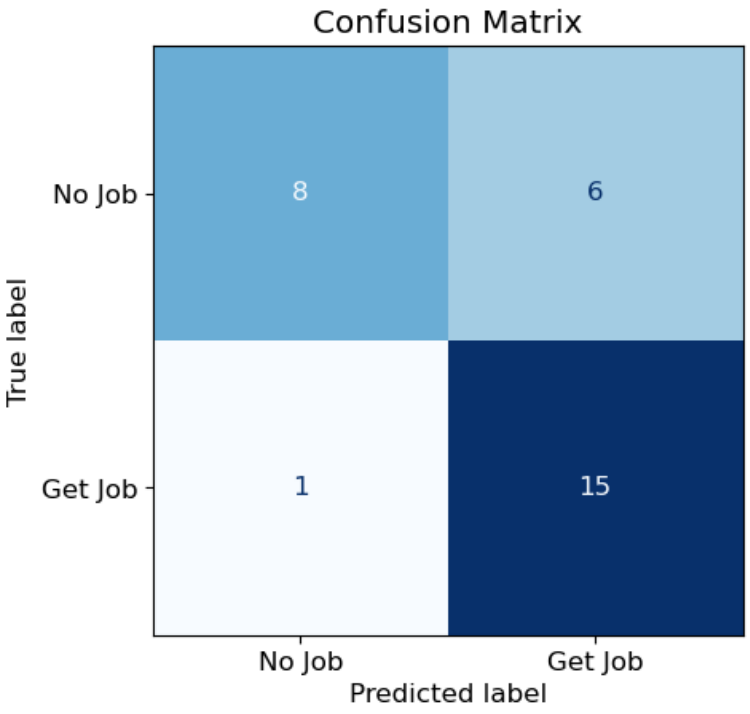
School of Computing and Information Systems

	Study Hours per Week	CGPA	Social Media Usage	Get Job Before Graduation
0	8.13	1.31	6.65	0
1	9.07	3.57	5.72	1
2	10.24	4.00	2.32	1
3	8.74	3.63	6.07	0
4	9.55	4.00	3.95	1

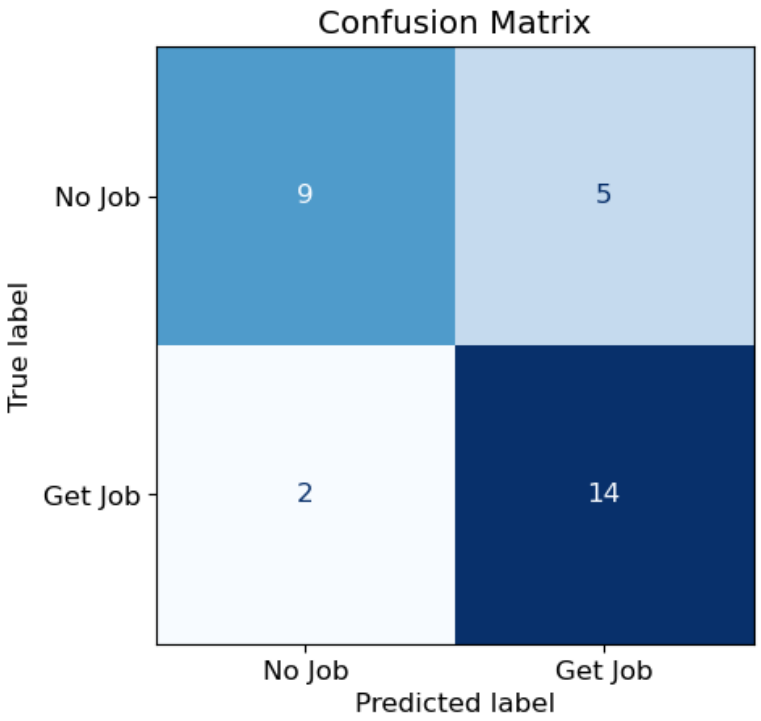


Confusion Matrix

Logistic Regression



Neural Networks



Confusion Matrix

School of Computing and Information Systems

- Definitions

	Predicted Negative	Predicted Positive
True Label Negative	TN	FP
True Label Positive	FN	TP

$$\text{Precision} = \frac{TP}{TP+FP}$$

$$\text{Recall} = \frac{TP}{TP+FN}$$

$$\text{F1\_score} = \frac{2 * TP}{2 * TP+FP+FN}$$

Hands-on coding


```
X = df.iloc[:,:3]
y = df.iloc[:,-1]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=2024)

X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

model = LogisticRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

cm = confusion_matrix(y_test, y_pred)

disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['No Job', 'Get Job'])
disp.plot(cmap=plt.cm.Blues)
plt.title('Confusion Matrix')
plt.show()
```

Hands-on coding


```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

X_train = torch.tensor(X_train, dtype=torch.float32)
X_test = torch.tensor(X_test, dtype=torch.float32)
y_train = torch.tensor(list(y_train), dtype=torch.float32).view(-1, 1)
y_test = torch.tensor(list(y_test), dtype=torch.float32).view(-1, 1)
```

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(3, 10)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(10, 1)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        x = self.fc1(x)
        x = self.relu(x)
        x = self.fc2(x)
        x = self.sigmoid(x)
        return x

model = Net()
criterion = nn.BCELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

```
for epoch in range(100):
    model.train()

    outputs = model(X_train)
    loss = criterion(outputs, y_train)

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
    print('loss: %.2f' % loss)

model.eval()
with torch.no_grad():
    y_pred = model(X_test)
    y_pred_class = (y_pred >= 0.5).float()

cm = confusion_matrix(y_test.numpy(), y_pred_class.numpy())
```

Real life considerations

- 100% is almost never possible
  - Two people may have the exact study patterns, CGPA and social media usage. Yet, only one gets a job before graduation.
  - Why?

Remember this?

Reality

$$y = 537.2 + 26.5x_1 - 1.7x_2 + 4.1x_3 + 9.6x_4 + 3.0x_5 + \cdots + x_{9999}$$

- Problem?
  - We do not have the full information of everything
  - Even IF we have everything, will be too complicated to solve
- Solution?
  - Consider only the important factors (that have access to)
  - Everything else is lumped into an 'error' term

$$y = w_0 + w_1 x_1 + \varepsilon$$

Real life considerations

- Mistakes are not equal

Here, positive is defined as getting a job before graduation

	Predicted Negative	Predicted Positive
True Label Negative	Correctly provide additional support	Student failed to secure a job; no support received
True Label Positive	Student received unnecessary support; able to secure job without help	Everyone happy, no support required

See You Tomorrow!

Day 2
(20 Aug 2024)

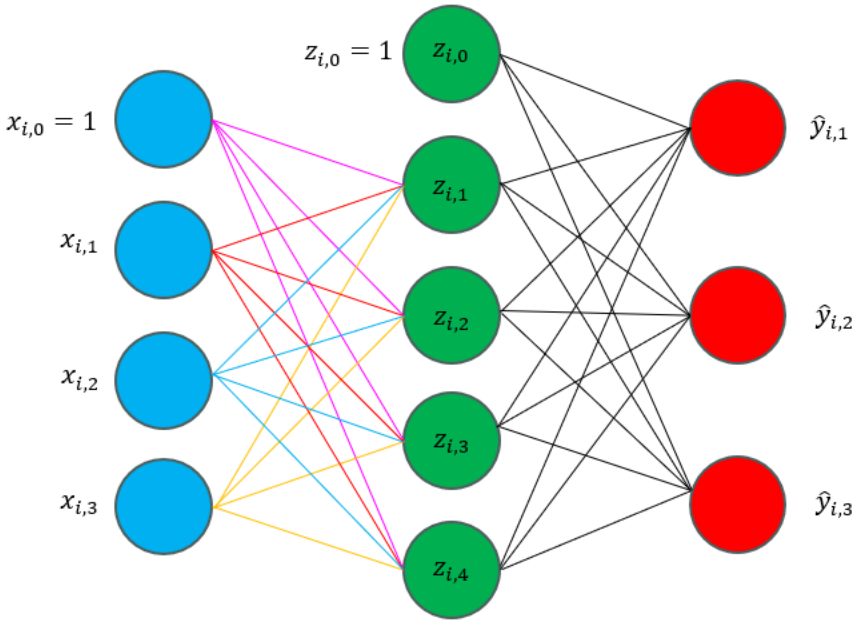
Quick Recap of Day 1

- Principles & examples of supervised learning
- Reality of modelling
- Linear regression (math & code)
- Logistic regression (math & code)
- Neural network
- Evaluation and considerations

Outline for today

- Gradient descent
- What it means to 'train' a model
- Understanding images
- Convolutions and CNN
- Multi-class classification
- Practical considerations

Recall Day 1



$$\vdots$$

$$z_{i,2} = f\left(w_{2,0}^{(1)} x_{i,0} + \ \ldots \ + w_{2,4}^{(1)} x_{i,4}\right)$$

$$\vdots$$

$$\hat{y}_{i,1} = \sigma\left(w_{1,1}^{(2)} z_{i,1} + \ \ldots \ + w_{1,5}^{(2)} z_{i,5}\right)$$

$$\hat{y}_{i,2} = \sigma\left(w_{2,1}^{(2)} z_{i,1} + \ \ldots \ + w_{2,5}^{(2)} z_{i,5}\right)$$

$$\hat{y}_{i,3} = \sigma\left(w_{3,1}^{(2)} z_{i,1} + \ \ldots \ + w_{3,5}^{(2)} z_{i,5}\right)$$

More compactly, can write as:

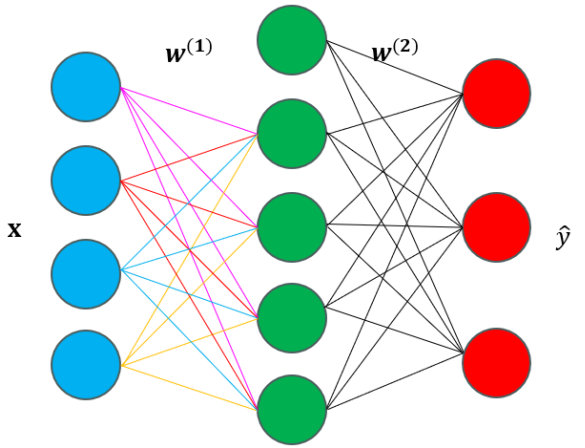
$$z_{i,m} = f\left(\sum_{j=0}^{d} w_{m,j}^{(1)} x_{i,j}\right)$$

$$\hat{y}_{i,k} = \sigma\left(\sum_{j=0}^{m} w_{k,j}^{(2)} z_{i,j}\right)$$

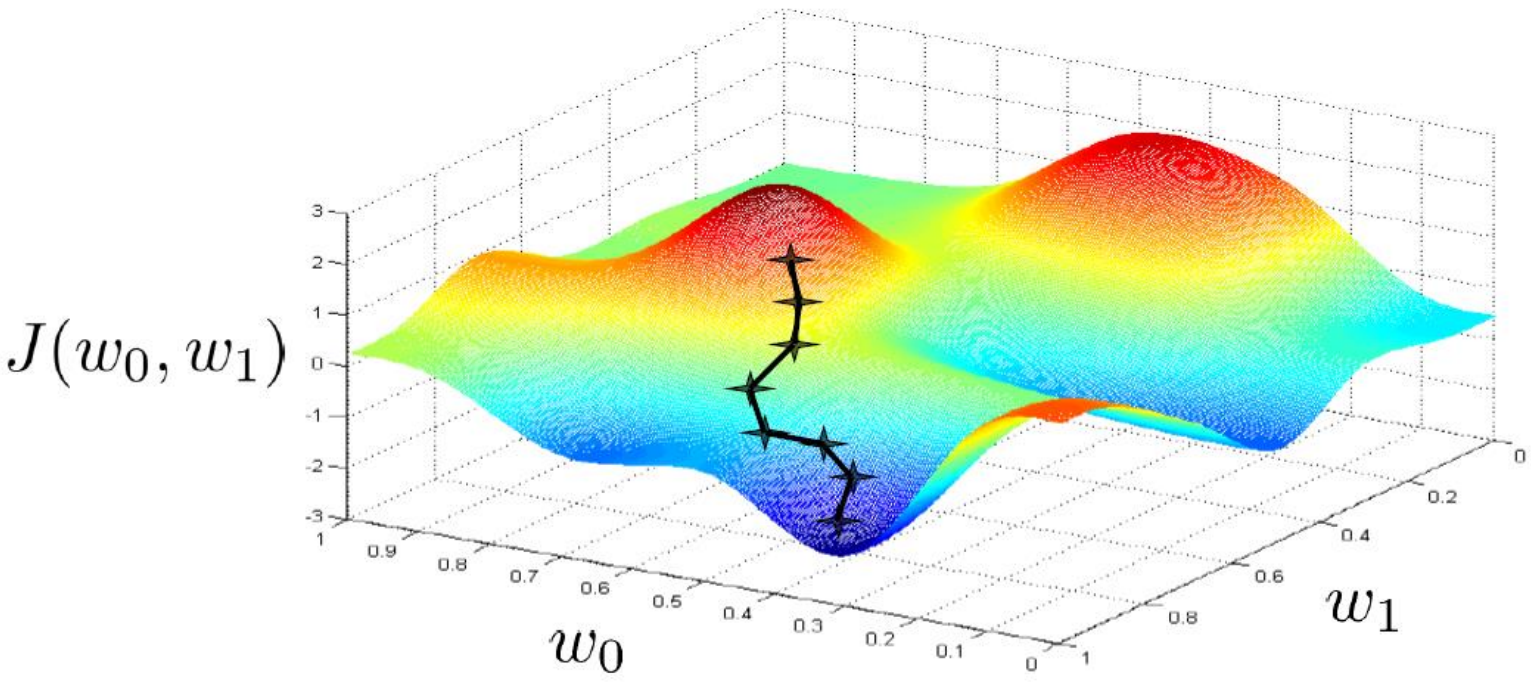
What values to use? How to train?

Training the model

- Start with observation $\mathbf{x}$
- Through multiplications and additions, end up with $\hat{\boldsymbol{y}}$.
- Depends on weights $\mathbf{w}^{(1)}$ and $\mathbf{w}^{(2)}$
- Want predictions $\hat{\boldsymbol{y}}$ to be close to actual targets $\mathbf{y}$.
  - Definition of 'close' is based on loss function
- Update $\mathbf{w}^{(1)}$ and $\mathbf{w}^{(2)}$ through gradient descent



Gradient Descent



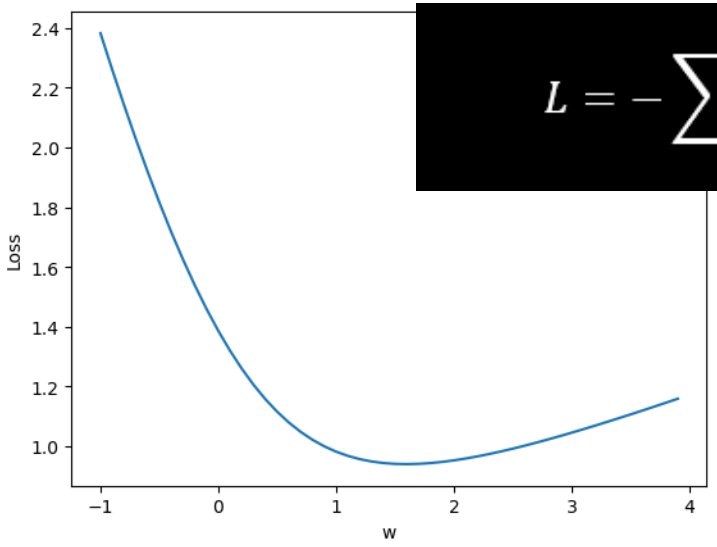
Simple worked example

- Consider a training dataset with only two samples, and a simple model which makes a prediction on a single feature x.

Sample	x	y
#0001	1.6	1
#0002	0.2	−1

- The loss function, which we want to minimize, is

$$L = -\sum_{i=1}^{n} \log\left(\frac{1}{1 + e^{-y_i \mathbf{w}^T \mathbf{x}_i}}\right)$$



refer to slides 26, 29

Simple worked example

- Consider what happens at a particular *w*, say, *w* = 0.5.

$$p(1|\mathbf{x}_{\#0001}) = \frac{1}{1+e^{-(0.5)(1.6)}} = 0.69$$

$$p(1|\mathbf{x}_{\#0002}) = \frac{1}{1+e^{-(0.5)(0.2)}} = 0.52$$

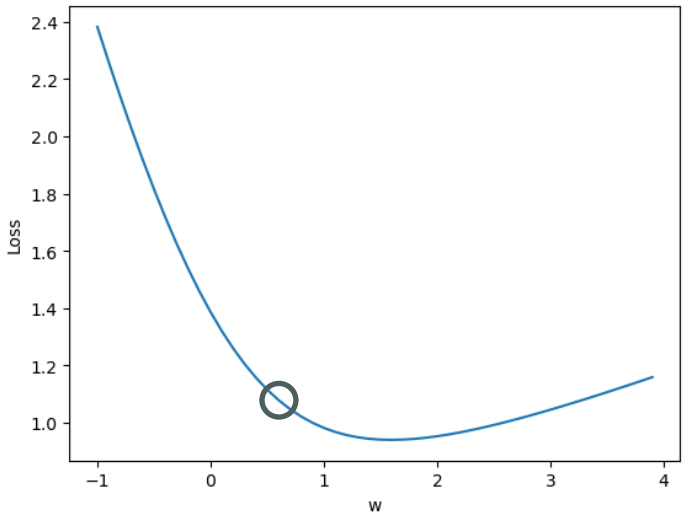
Ideally, probability of first sample should be close to 1, probability of second sample be close to 0.

- Apply gradient update. *w* will change, giving a smaller loss in the next iteration

$$\nabla L(\mathbf{w}) = \sum_{i=1}^{n} \frac{-y_i \mathbf{x}_i e^{-y_i \mathbf{w}^T \mathbf{x}_i}}{1+e^{-y_i \mathbf{w}^T \mathbf{x}_i}}$$

$$= -\sum_{i=1}^{n} y_i \mathbf{x}_i [1 - p(y_i|\mathbf{x}_i)]$$

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \nabla L(\mathbf{w}_t)$$



Derivations (FYI)

$$L(\mathbf{w}) = -\sum_{i=1}^{n} \log\left(\frac{1}{1+e^{-y_i \mathbf{w}^T \mathbf{x}_i}}\right)$$

$$= \sum_{i=1}^{n} \log\left(1 + e^{-y_i \mathbf{w}^T \mathbf{x}_i}\right)$$

$$\nabla_{\mathbf{w}} L(\mathbf{w}) = \nabla_{\mathbf{w}} \sum_{i=1}^{n} \left(\log\left(1 + e^{-y_i \mathbf{w}^T \mathbf{x}_i}\right)\right)$$

$$= \sum_{i=1}^{n} \nabla_{\mathbf{w}} \left(\log\left(1 + e^{-y_i \mathbf{w}^T \mathbf{x}_i}\right)\right)$$

$$= \sum_{i=1}^{n} \frac{\nabla_{\mathbf{w}} (1+e^{-y_i \mathbf{w}^T \mathbf{x}_i})}{1+e^{-y_i \mathbf{w}^T \mathbf{x}_i}}$$

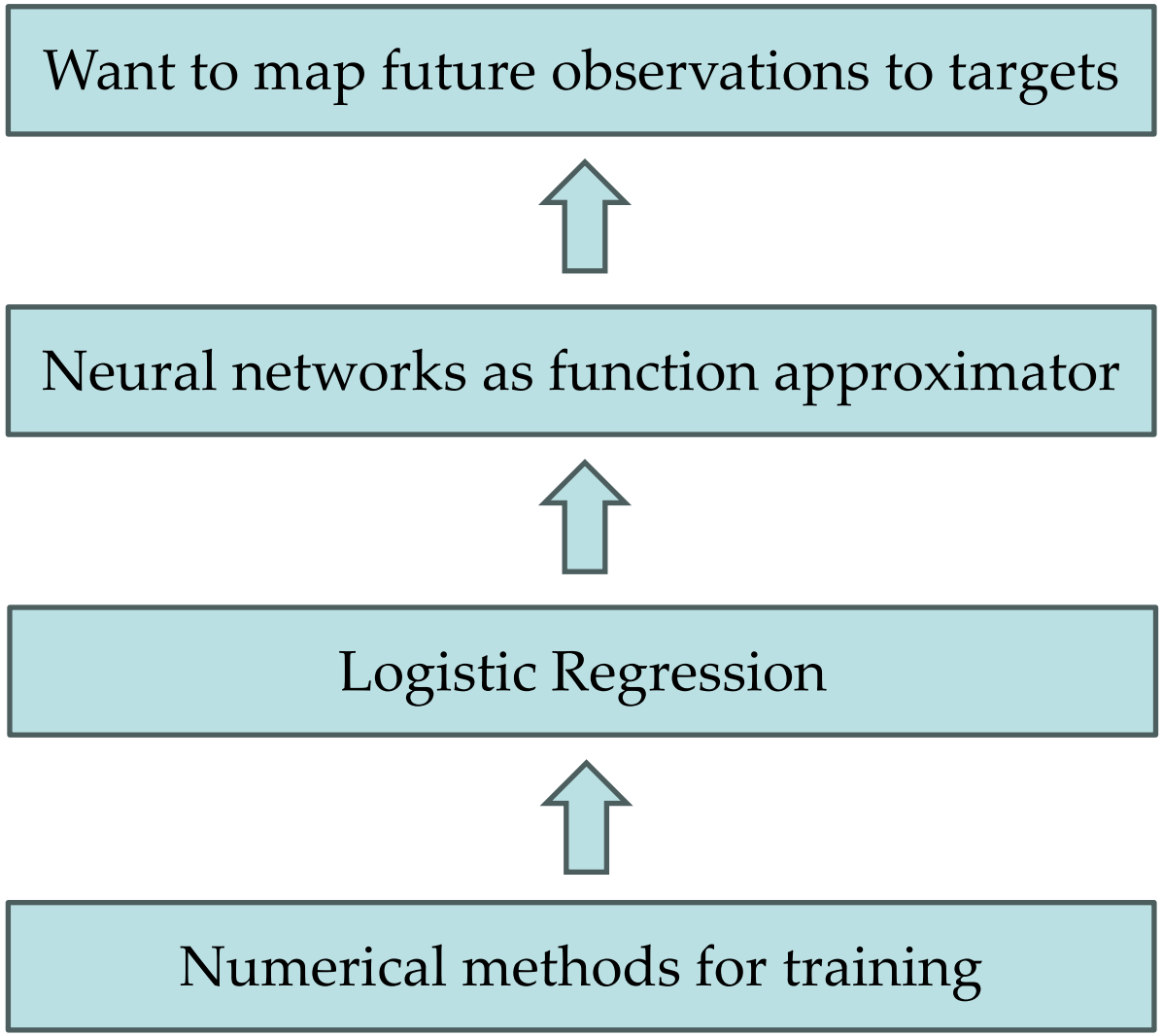
$$= \sum_{i=1}^{n} \frac{-y_i \mathbf{x}_i \, e^{-y_i \mathbf{w}^T \mathbf{x}_i}}{1+e^{-y_i \mathbf{w}^T \mathbf{x}_i}}$$

$$= -\sum_{i=1}^{n} y_i \mathbf{x}_i \frac{e^{-y_i \mathbf{w}^T \mathbf{x}_i}}{1+e^{-y_i \mathbf{w}^T \mathbf{x}_i}}$$

$$= -\sum_{i=1}^{n} y_i \mathbf{x}_i \left[1 - p(y_i|\mathbf{x}_i)\right]$$

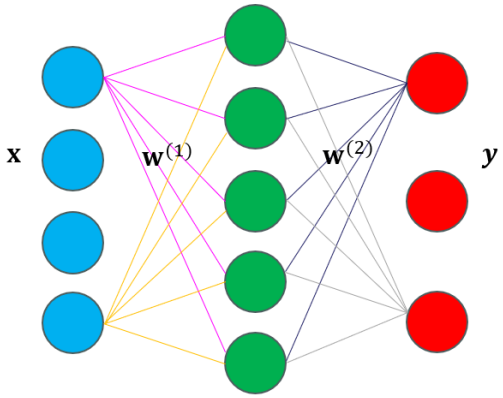
https://medium.com/mitb-for-all/a-worked-example-on-updating-weights-by-gradient-descent-9a12181ed2b9

What's the point of all these?



Tiny example

- Suppose we want to train a FNN with 3 samples
  - Ignore overfitting/train-test split for now; focus on one concept at a time



Sample	$\mathbf{x} = (x_1, x_2, x_3, x_4)$				y
#01	0.1	-0.2	0.3	-0.4	category A
#02	0.5	0.6	-0.7	-0.8	category B
#03	-0.3	0.5	-0.4	0.2	category C

Hands-on coding


```python
class SimpleNN(nn.Module):
    def __init__(self):
        super(SimpleNN, self).__init__()
        self.fc1 = nn.Linear(4, 5)
        self.fc2 = nn.Linear(5, 3)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.fc1(x)
        x = self.relu(x)
        x = self.fc2(x)
        return x
```

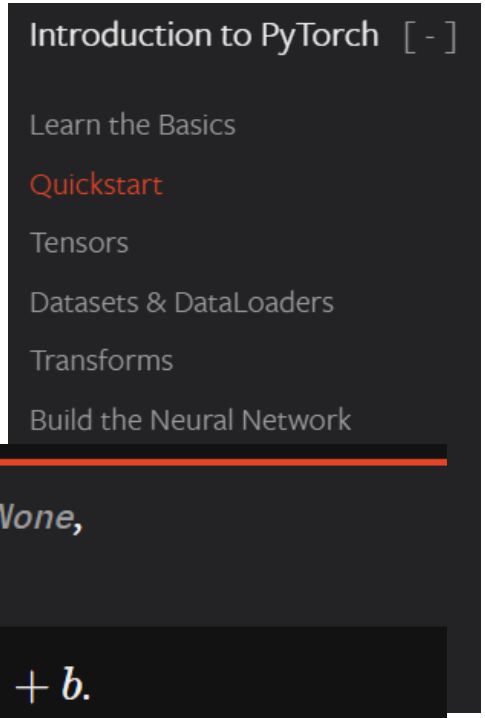
```python
X = torch.tensor([
    [0.1, -0.2, 0.3, -0.4],
    [0.5, 0.6, -0.7, -0.8],
    [-0.3, 0.5, -0.4, 0.2],
], dtype=torch.float32)

y = torch.tensor([
    [1, 0, 0],
    [0, 1, 0],
    [0, 0, 1],
], dtype=torch.float32)
```

```python
model = SimpleNN()
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.01)

for epoch in range(1000):
    y_pred = model(X)
    loss = criterion(y_pred, torch.max(y, 1)[1])
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
    if (epoch+1) % 50 == 0:
        print('Epoch %d: Loss: %.3f' % (epoch+1, loss))

y_hat = torch.softmax(model(X), dim=1)
print(y_hat)
```

The details

- Getting started: https://pytorch.org/tutorials/beginner/basics/quickstart_tutorial.html

- nn.Linear()



torch.nn.Linear(*in_features*, *out_features*, *bias=True*, *device=None*, *dtype=None*) [SOURCE]

Applies an affine linear transformation to the incoming data: $y = xA^T + b$.

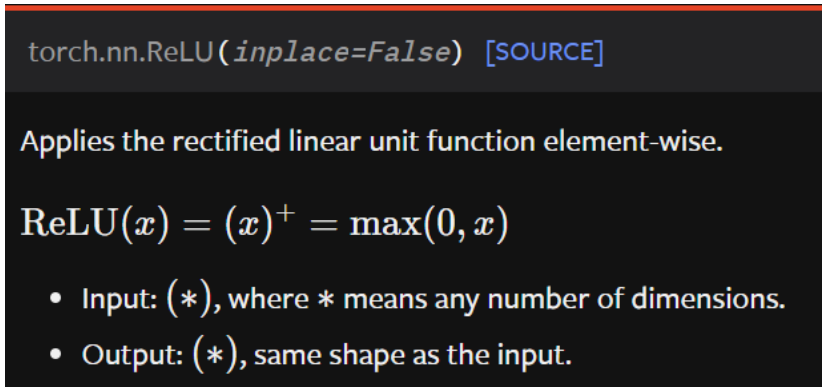
Parameters

- **in_features** (*int*) – size of each input sample
- **out_features** (*int*) – size of each output sample
- **bias** (*bool*) – If set to `False`, the layer will not learn an additive bias. Default: `True`

https://pytorch.org/docs/stable/generated/torch.nn.Linear.html#torch.nn.Linear

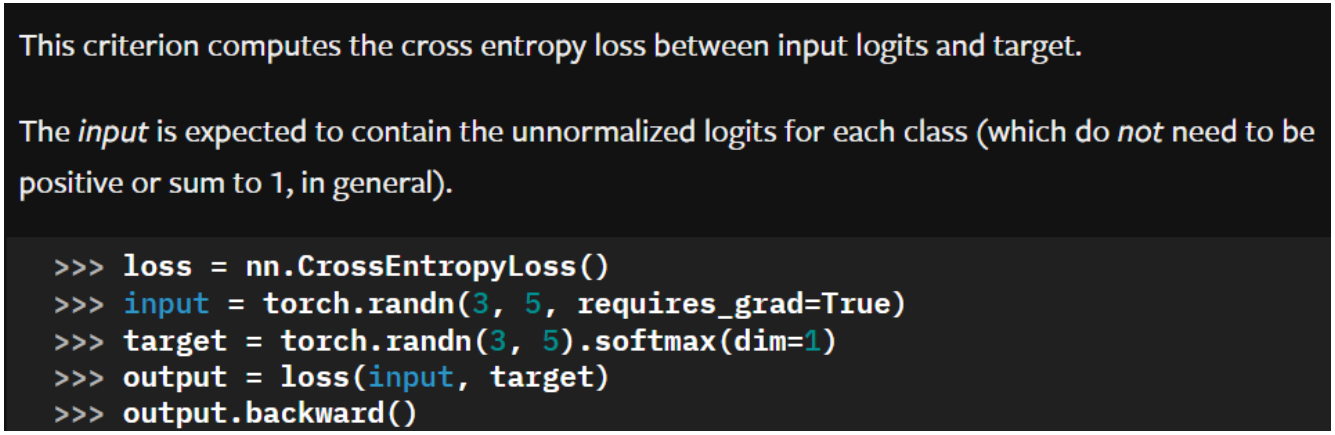
The details

- nn.ReLU()



https://pytorch.org/docs/stable/generated/torch.nn.ReLU.html

- nn.CrossEntropyLoss()



https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html
(Screenshot is filtered. You need to scroll)

Sanity Check

- Is the predicted probability of the training data moving towards the direction of the ground truth?
- Is nn.CrossEntropyLoss() working like what you think?

```
logits = torch.tensor([
    [1.2, -0.3, 0.5],
    [0.1, 0.7, -0.2],
    [0.3, 0.4, 1.3],
], dtype=torch.float32)
criterion = nn.CrossEntropyLoss()
criterion(logits, torch.max(y_onehot, 1)[1])
```

```
tensor(0.5954)
```

How do you see the weights and biases within?

```
for name, param in model.named_parameters():
    print(name, param)
```

```
softmax = torch.softmax(logits, dim=1)
log_probs = torch.log(softmax)
y_true = torch.argmax(y, dim=1)
selected_log_probs = log_probs[range(len(y_true)), y_true]
-selected_log_probs.mean()
```

```
tensor(0.5954)
```

Training the model

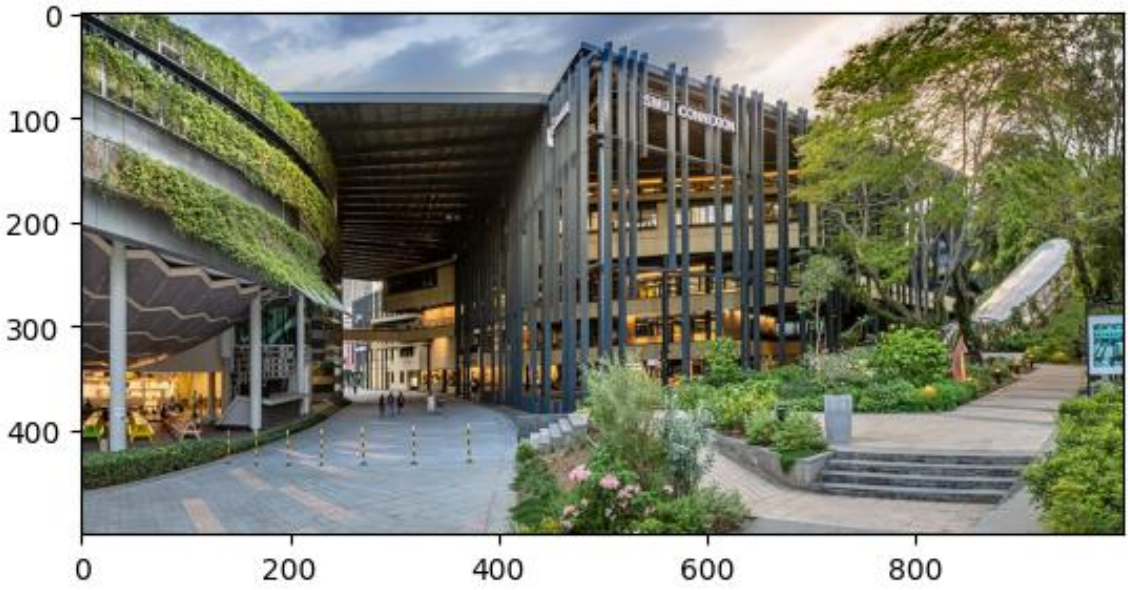

```
Epoch 50: Loss: 1.023
Epoch 100: Loss: 0.968
Epoch 150: Loss: 0.914
Epoch 200: Loss: 0.857
Epoch 250: Loss: 0.796
Epoch 300: Loss: 0.733
Epoch 350: Loss: 0.669
Epoch 400: Loss: 0.605
Epoch 450: Loss: 0.540
Epoch 500: Loss: 0.477
Epoch 550: Loss: 0.417
Epoch 600: Loss: 0.362
Epoch 650: Loss: 0.312
Epoch 700: Loss: 0.269
Epoch 750: Loss: 0.232
Epoch 800: Loss: 0.201
Epoch 850: Loss: 0.175
Epoch 900: Loss: 0.152
Epoch 950: Loss: 0.134
Epoch 1000: Loss: 0.119
tensor([[0.8605, 0.0524, 0.0871],
        [0.0440, 0.9187, 0.0373],
        [0.0811, 0.0319, 0.8870]],
```

- Practical tips
  - Many choices; know what matters.
  - How do you decide whether there's a problem
    - with your code, or
    - the choice of model, or
    - the hyperparameters, or
    - simply the data?

What is an image?

SMU Singapore Management University
School of Computing and Information Systems

```
from PIL import Image
image = Image.open('/content/smu_photo.jpg')

plt.imshow(image)
plt.show()
```



https://www.smu.edu.sg/campus-life/health-and-safety

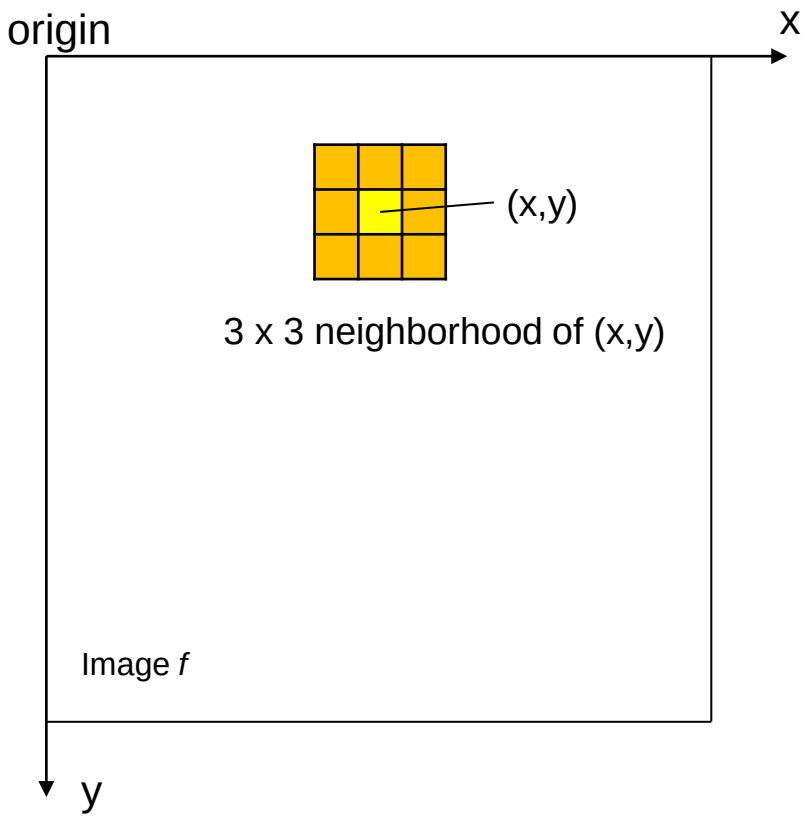
```
image_array

ndarray (500, 1000, 3)  hide data
array([[[ 10,  17,  23],
        [ 28,  35,  41],
        [ 75,  82,  88],
        ...,
        [ 40,  48,   0],
        [ 42,  51,   0],
        [ 55,  64,   7]],

       [[  5,  12,  18],
        [ 13,  20,  26],
        [ 14,  21,  27],
        ...,
        [ 84,  91,  50],
        [ 48,  56,   9],
        [ 82,  90,  39]],
```

Simple grayscale example


```
heart_shape = [
    [ 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0],
    [ 0,  0,  0, 10, 30, 30, 10,  0,  0, 10, 30, 30, 10,  0,  0,  0],
    [ 0,  0, 10, 50, 99, 99, 50, 10, 10, 50, 99, 99, 50, 10,  0,  0],
    [ 0, 10, 50, 99, 99, 99, 99, 50, 50, 99, 99, 99, 99, 50, 10,  0],
    [10, 50, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 50, 10],
    [10, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 50],
    [10, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 50],
    [ 0, 50, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 50, 10],
    [ 0, 10, 50, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 50, 10,  0],
    [ 0,  0, 10, 50, 99, 99, 99, 99, 99, 99, 99, 99, 50, 10,  0,  0],
    [ 0,  0,  0, 10, 50, 99, 99, 99, 99, 99, 99, 50, 10,  0,  0,  0],
    [ 0,  0,  0,  0, 10, 50, 99, 99, 99, 99, 50, 10,  0,  0,  0,  0],
    [ 0,  0,  0,  0,  0, 10, 50, 99, 99, 50, 10,  0,  0,  0,  0,  0],
    [ 0,  0,  0,  0,  0,  0, 10, 50, 50, 10,  0,  0,  0,  0,  0,  0],
    [ 0,  0,  0,  0,  0,  0,  0, 10, 10,  0,  0,  0,  0,  0,  0,  0],
    [ 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0],
]
plt.imshow(heart_shape, cmap='gray')
```

Spatial filtering

- 'Filtering' concept from signal processing.
- Mathematical operation (multiplication, addition, activation) on pixels.
- Can identify edges, and higher-level features with multiple layers



Edge detection

(a)

(b)

(a) Original Image (b) Sobel
https://masters.donntu.ru/2019/fknt/strokin/library/article6.pdf

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

Kernel for Gx

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

Kernel for Gy

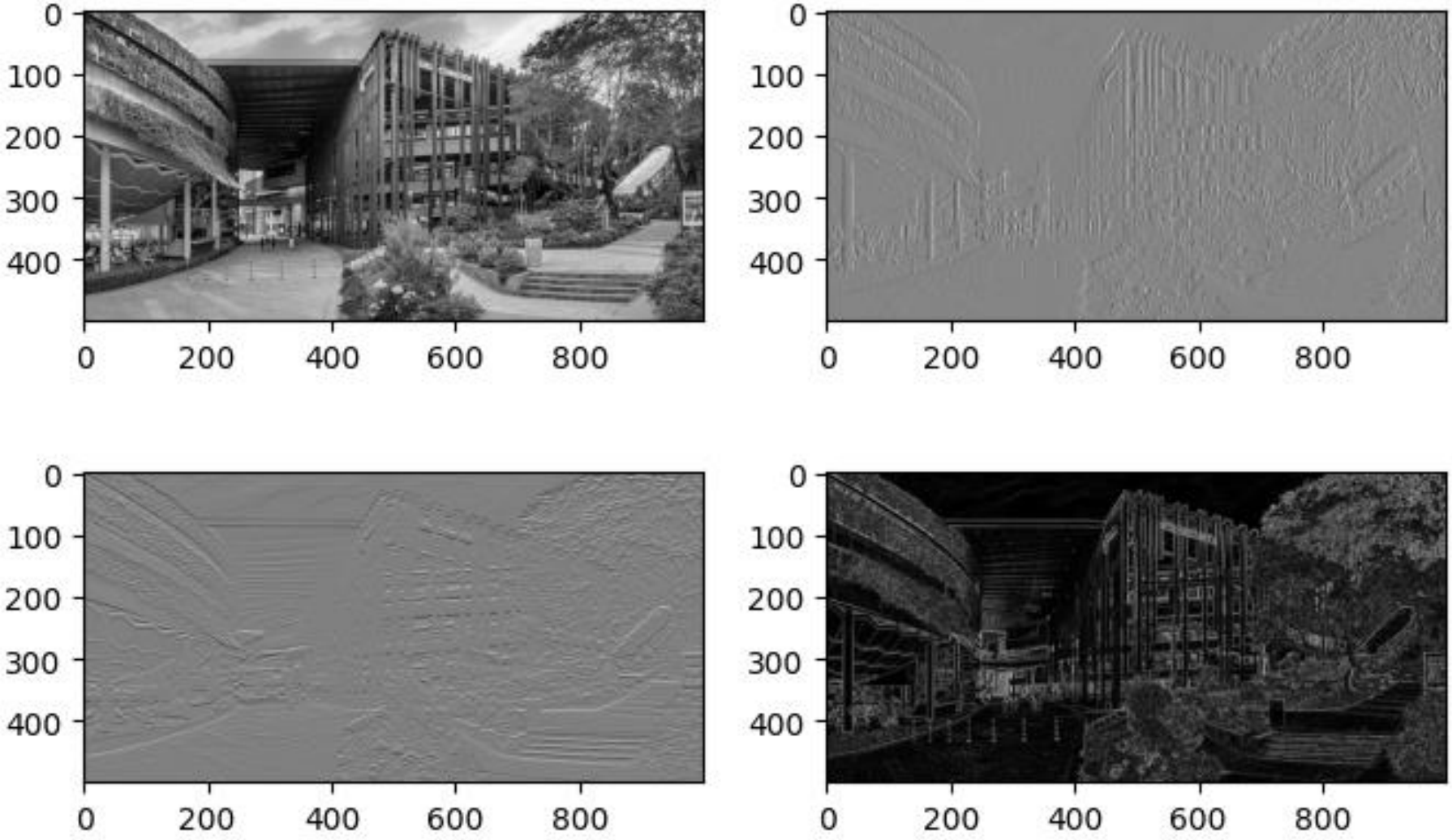
$$|G| = \sqrt{Gx^2 + Gy^2}$$

Your own edge detection


```
from scipy.ndimage import convolve

sobel_x = np.array(
    [[-1, 0, 1],
     [-2, 0, 2],
     [-1, 0, 1]]
)
sobel_y = np.array(
    [[-1, -2, -1],
     [0,  0,  0],
     [1,  2,  1]]
)
```

```
grayscale_image = np.mean(image_array, axis=2)
Gx = convolve(grayscale_image, sobel_x)
Gy = convolve(grayscale_image, sobel_y)
gradient_magnitude = np.sqrt(Gx**2 + Gy**2)

fig, axes = plt.subplots(2,2,figsize=(8,5))
axes[0,0].imshow(grayscale_image, cmap='gray')
axes[0,1].imshow(Gx, cmap='gray')
axes[1,0].imshow(Gy, cmap='gray')
axes[1,1].imshow(gradient_magnitude, cmap='gray')
plt.show()
```

Your own edge detection



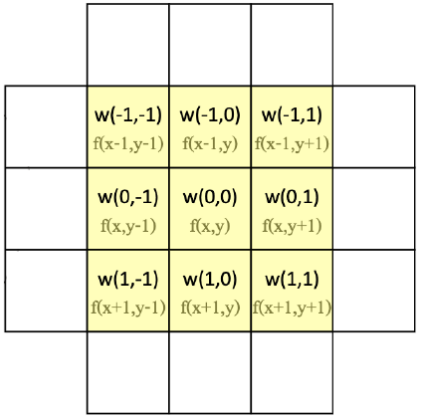
Mathematical operation

origin

x

```
[ 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0]
[ 0,  0,  0, 10, 30, 30, 10,  0,  0, 10, 30, 30, 10,  0,  0,  0]
[ 0,  0, 10, 50, 99, 99, 50, 10, 10, 50, 99, 99, 50, 10,  0,  0]
[ 0, 10, 50, 99, 99, 99, 99, 50, 50, 99, 99, 99, 99, 50, 10,  0]
[10, 50, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 50, 10]
[10, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 50]
[10, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 50]
[ 0, 50, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 50, 10]
[ 0, 10, 50, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 50, 10,  0]
[ 0,  0, 10, 50, 99, 99, 99, 99, 99, 99, 99, 99, 50, 10,  0,  0]
[ 0,  0,  0, 10, 50, 99, 99, 99, 99, 99, 99, 50, 10,  0,  0,  0]
[ 0,  0,  0,  0, 10, 50, 99, 99, 99, 99, 50, 10,  0,  0,  0,  0]
[ 0,  0,  0,  0,  0, 10, 50, 99, 99, 50, 10,  0,  0,  0,  0,  0]
[ 0,  0,  0,  0,  0,  0, 10, 50, 50, 10,  0,  0,  0,  0,  0,  0]
[ 0,  0,  0,  0,  0,  0,  0, 10, 10,  0,  0,  0,  0,  0,  0,  0]
[ 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0]
```

y



$$g(x,y) = \sum_{i=-1}^{1} \sum_{j=-1}^{1} w(i,j) f(x+i, y+j)$$

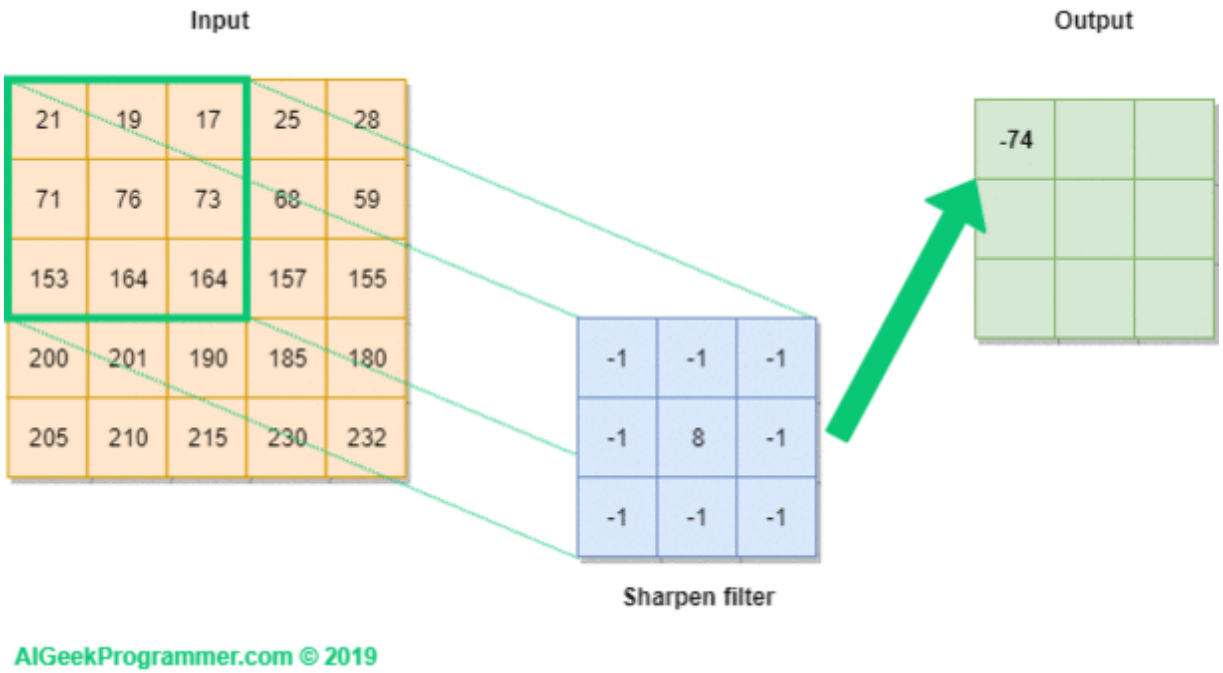
-1	0	1
-2	0	2
-1	0	1

kernel

-1*30	0*10	1*0
-2*99	0*50	2*10
-1*99	0*99	1*50

-1(30) + 0 + 1(0) -2(99) + 0 + 2(10) -1(99) + 0 + 1(50) = **-277**

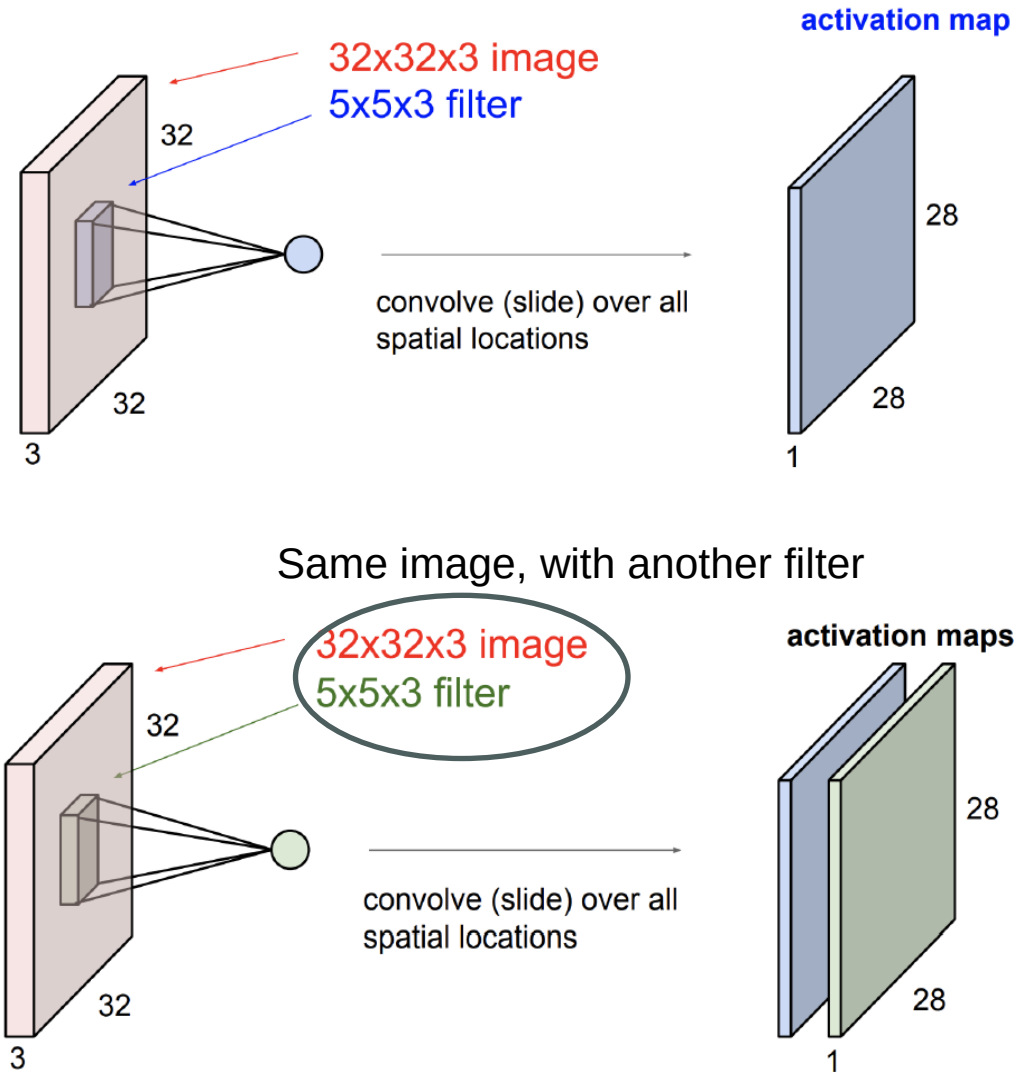
Convolution step-by-step



https://commons.wikimedia.org/wiki/File:CNN-filter-animation-1.gif

Pixel values are typically represented in [0,255] or normalized to [0,1]

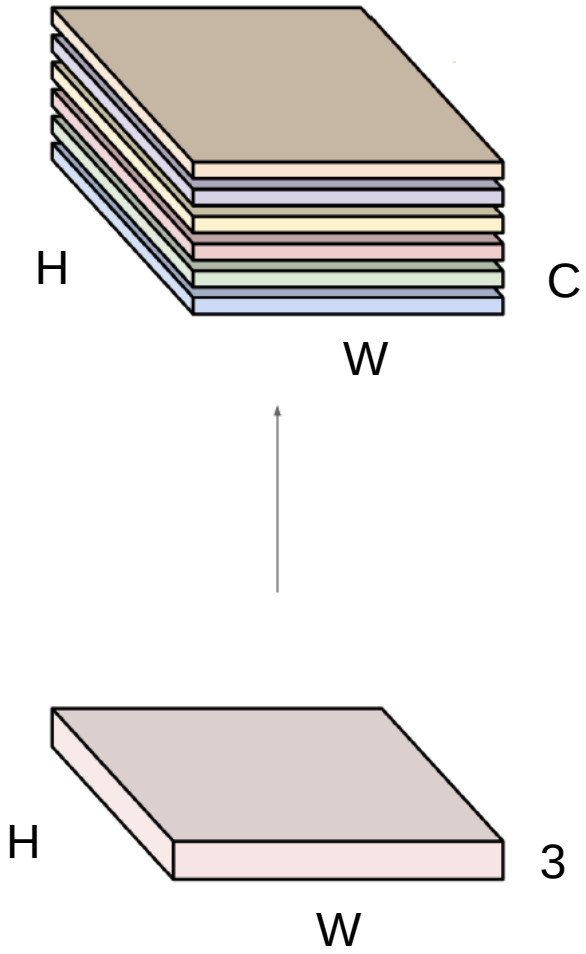
Feature maps

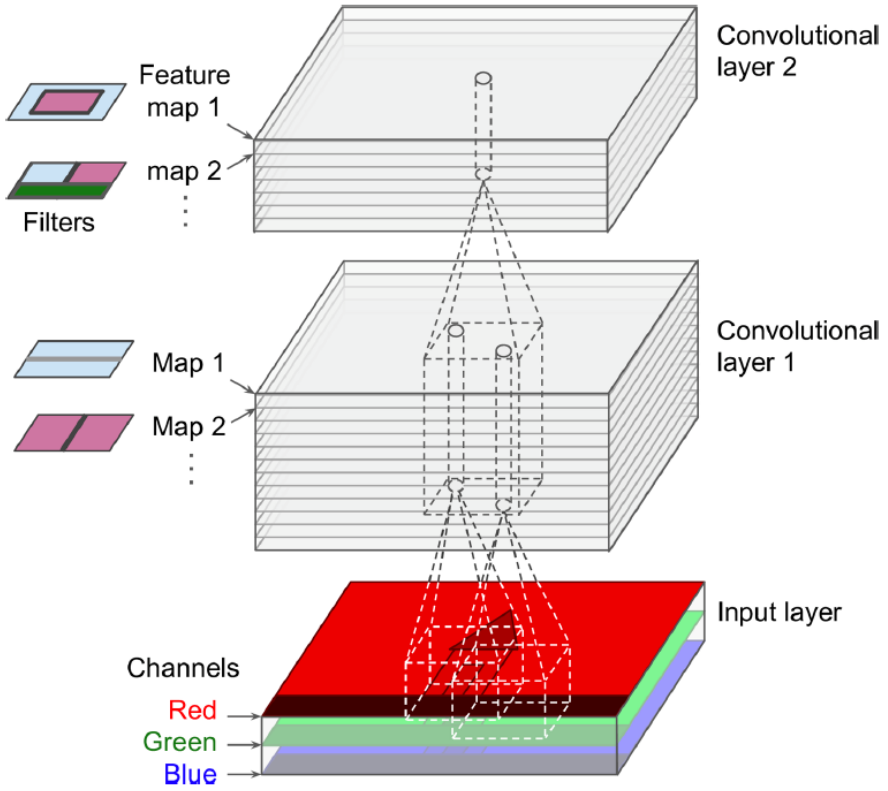


http://cs231n.stanford.edu/

- Multiple filters can be applied at a single convolution layer
- Output involves multiple feature maps
  - Response of the respective filters
  - Indicates presence or absence of the learnt features

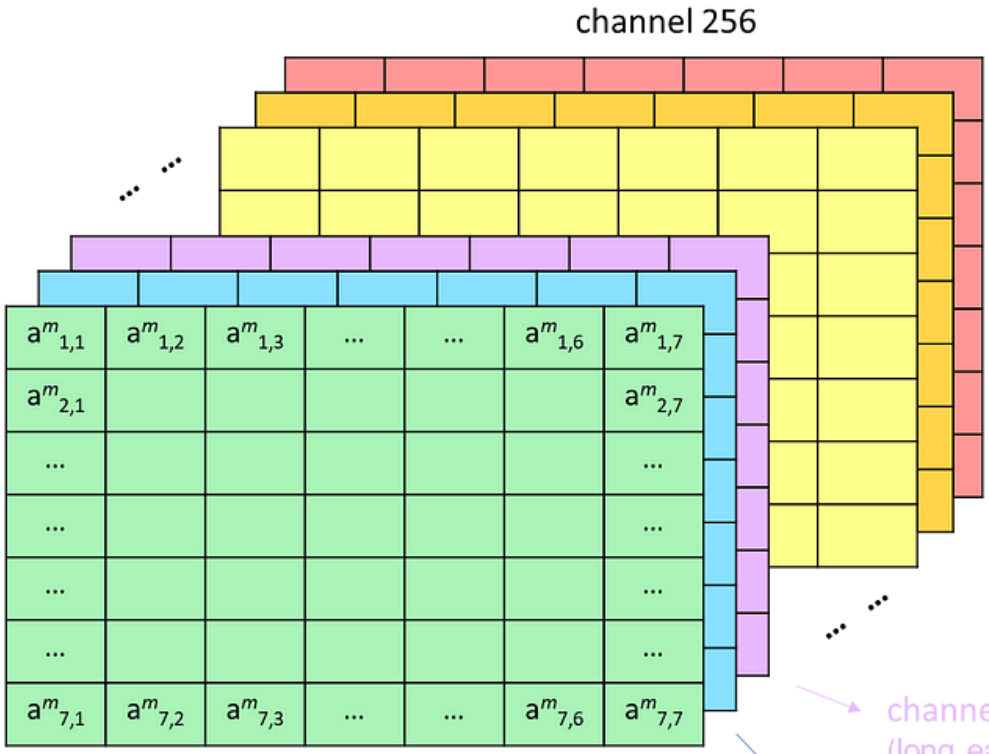
Multiple Convolutions



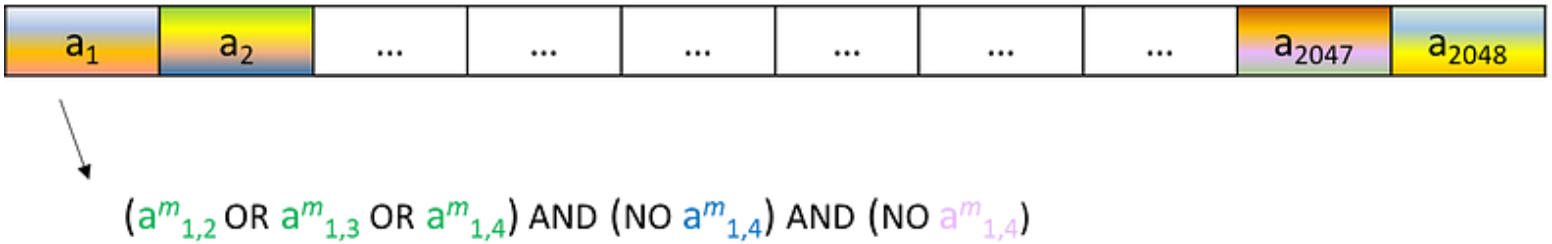


Aurelien Geron. Hands-on Machine Learning with Scikit-Learn & Tensorflow

Feature Maps



https://medium.com/mitb-for-all/intuition-behind-probabilities-from-supervised-learning-391a4eaf2ac6

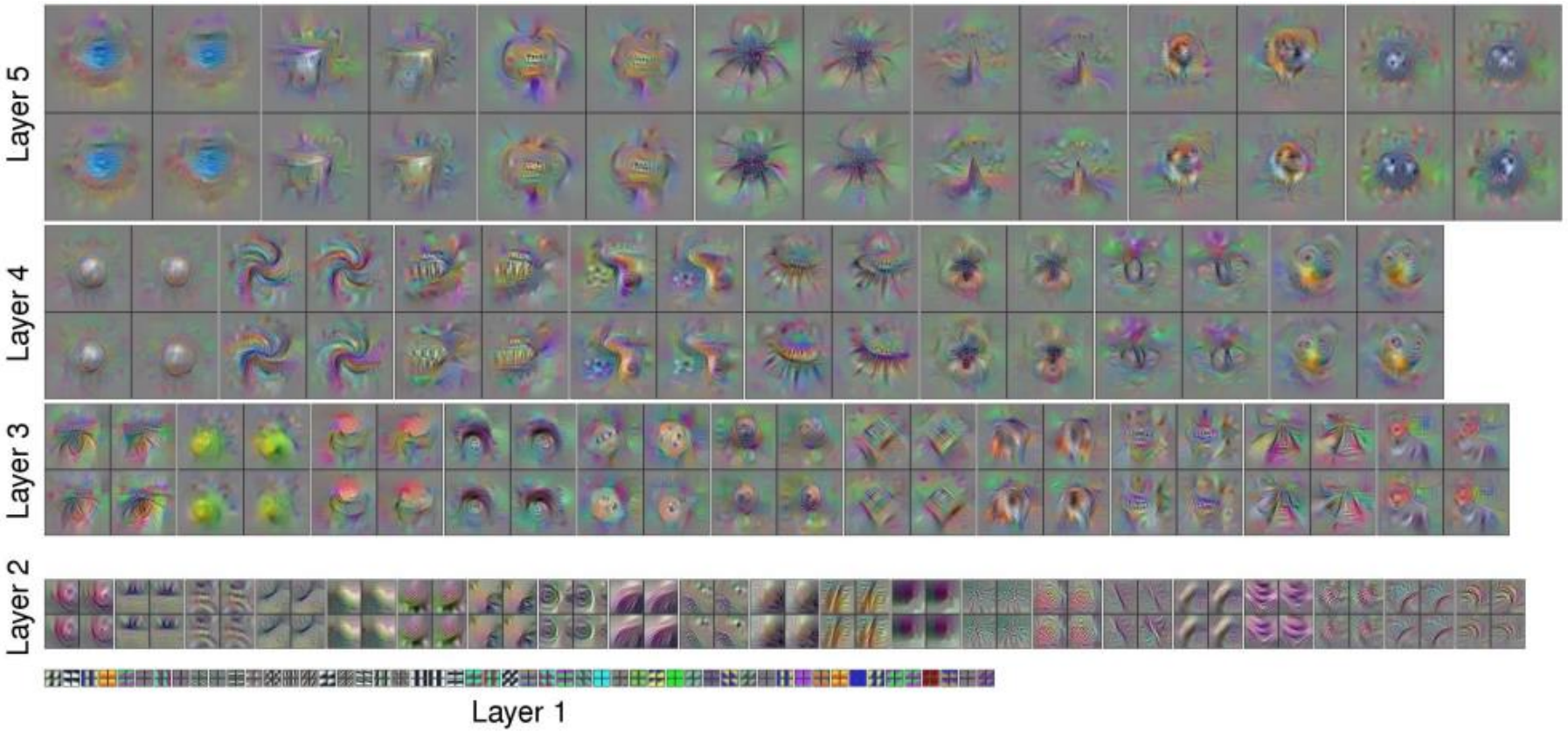


CNN – What & Why?

- Local Receptive Fields
  - Focus on small, local regions of an input image
  - Logical for images, since neighboring pixels are usually related
- Parameter Sharing
  - Same filters (weights) applied to multiple spatial locations
  - Reduces the number of trainable parameters (for computational efficiency and generalization)
  - Efficiently learn local patterns
- Translation Invariance
  - Learn features regardless of their location in an image

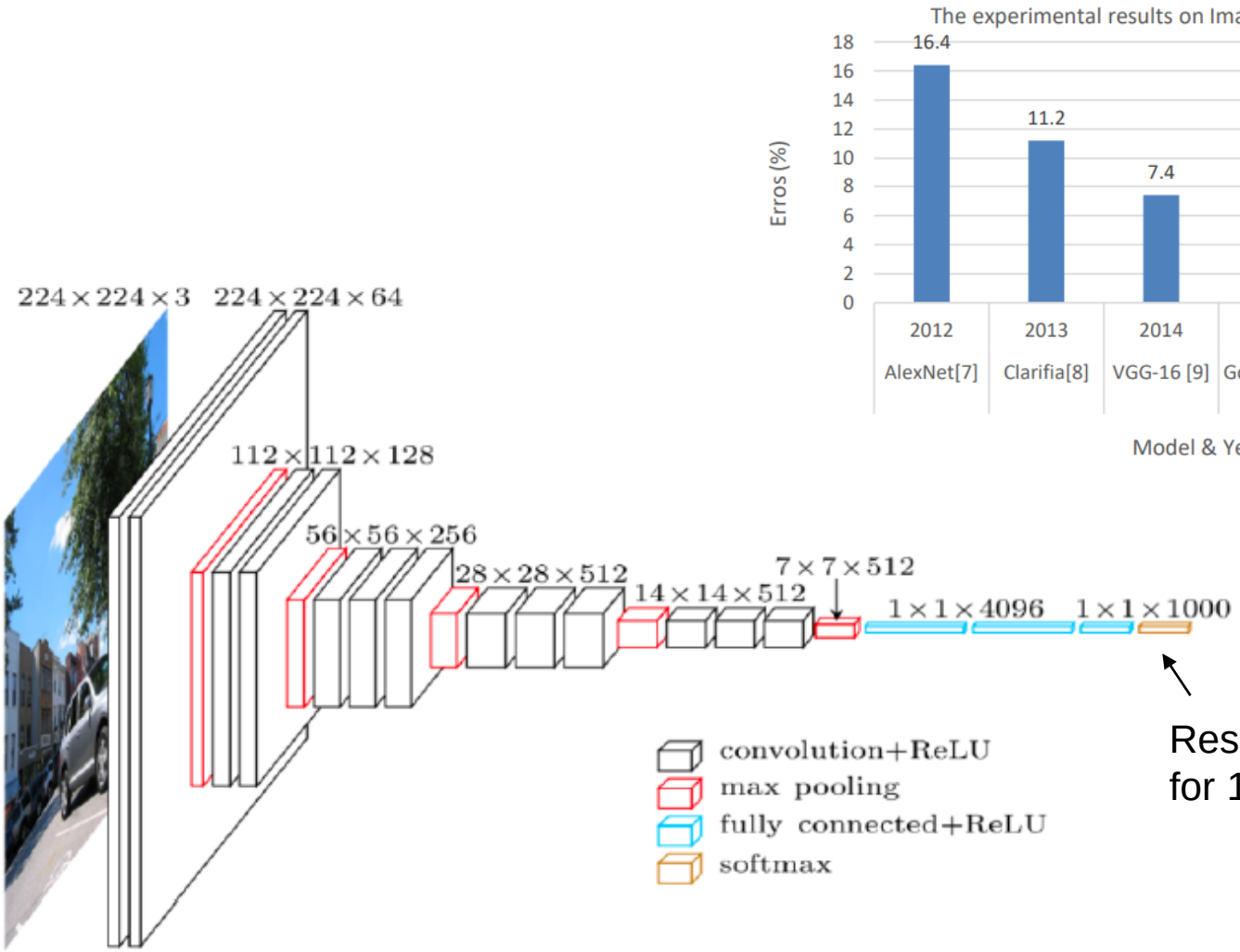
Higher level features

- Through a series of convolution layers with pooling (typically max-pooling)



Yosinski et al. 2015 https://arxiv.org/abs/1506.06579

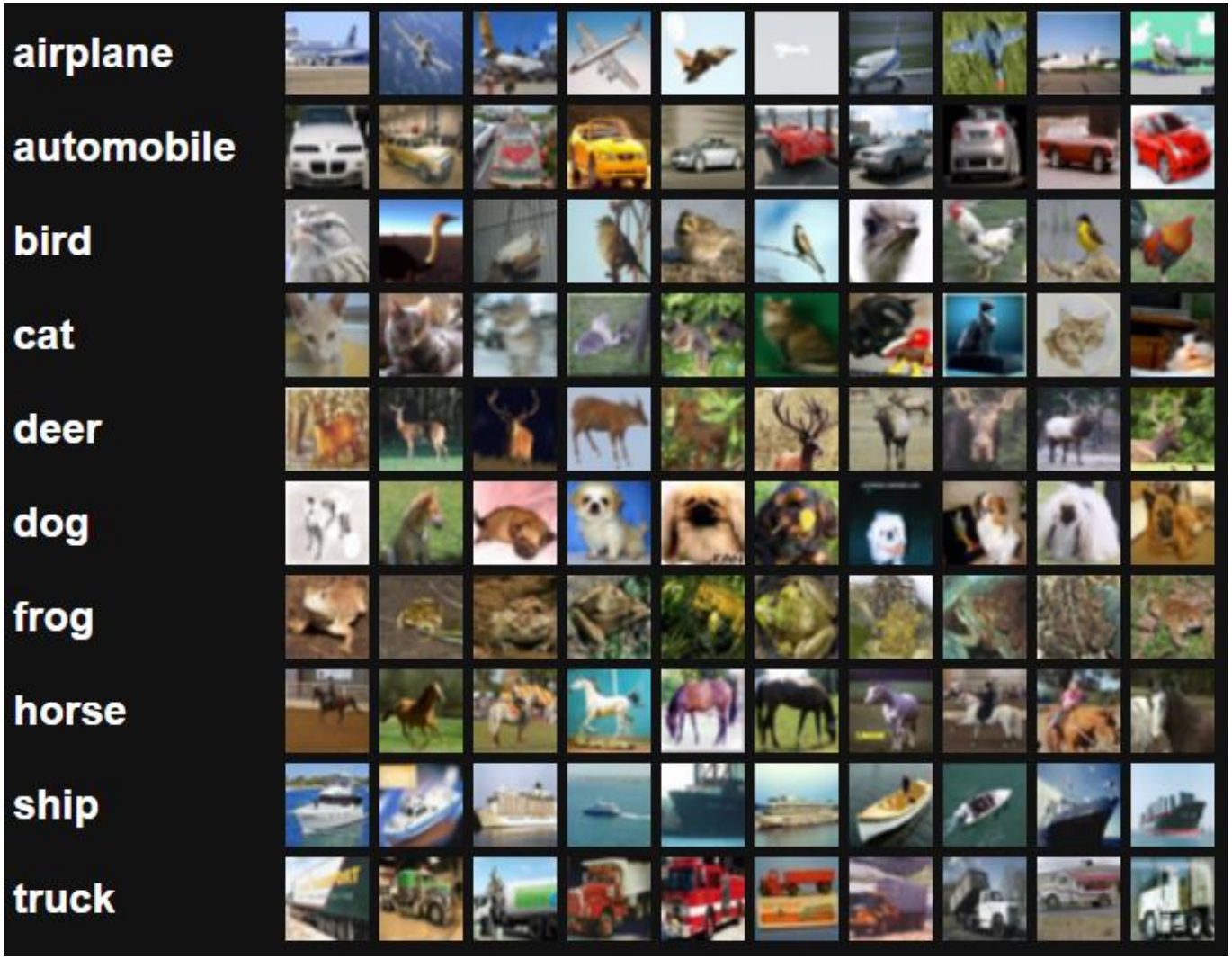
VGG



Respective probabilities for 1000 possible classes

Simonyan & Zisserman (2014). Very Deep Convolutional Networks for Large Scale Image Recognition. Computer Vision and Pattern Recognition

CIFAR-10



32x32 color images
10 classes
50k train images
10k test images

https://www.cs.toronto.edu/~kriz/cifar.html

Hands-on coding


```python
class DataManager:
    def __init__(self):
        self.C, self.H, self.W = 3, 32, 32
        self.batch_size = 64
        self.transform = transforms.Compose([
            transforms.ToTensor(),
            transforms.Normalize(mean, std)
        ])
        self.trainset = torchvision.datasets.CIFAR10(
            root=".", train=True,
            download=True, transform=self.transform
        )
        self.testset = torchvision.datasets.CIFAR10(
            root=".", train=False,
            download=True, transform=self.transform
        )
        self.class_to_idx = self.trainset.class_to_idx
        self.classes = list(self.class_to_idx.keys())

        self.trainloader = torch.utils.data.DataLoader(
            self.trainset, batch_size=self.batch_size,
            shuffle=True, num_workers=0
        )
        self.testloader = torch.utils.data.DataLoader(
            self.testset, batch_size=self.batch_size,
            shuffle=False, num_workers=0
        )
```

```python
class BasicNet(nn.Module):
    def __init__(self, input_shape, num_classes):
        super().__init__()
        C, H, W = input_shape
        self.conv1 = nn.Conv2d(C, 16, 3, padding=1)
        self.conv2 = nn.Conv2d(16, 32, 3, padding=1)
        self.relu = nn.ReLU()
        self.pool = nn.MaxPool2d(2, 2)
        self.fc1 = nn.Linear(32*8*8, 128)
        self.fc2 = nn.Linear(128, 64)
        self.fc3 = nn.Linear(64, num_classes)


    def forward(self, x):
        x = self.relu(self.conv1(x))
        x = self.pool(x)
        x = self.relu(self.conv2(x))
        x = self.pool(x)
        x = torch.flatten(x, 1)
        x = self.relu(self.fc1(x))
        x = self.relu(self.fc2(x))
        x = self.fc3(x)   # logits
        return x
```

Take note of memory constraints!

Hands-on coding


```
class Classifier:
    def __init__(self, dataMananger):
        self.dataMananger = dataMananger
        self.trainloader = self.dataMananger.trainloader
        self.testloader = self.dataMananger.testloader
        self.classes = self.dataMananger.classes
        self.artifacts_dir = "./artifact/"
        if not os.path.exists(self.artifacts_dir):
            os.makedirs(self.artifacts_dir)

        self.device = torch.device(
            torch.cuda.current_device() if torch.cuda.is_available() else 'cpu'
        )
        print("Using device %s" % self.device)

        input_shape = \
            (self.dataMananger.C, self.dataMananger.H, self.dataMananger.W)
        self.model = BasicNet(
            input_shape, num_classes=len(self.classes)
        )
        self.model.to(self.device)
        self.loss_function = nn.CrossEntropyLoss()
```

Hands-on coding


```
def train(self, epochs=1, lr=1e-3, save=True, overfit=False):
    self.lr = lr
    self.optimizer = optim.Adam(self.model.parameters(), lr=self.lr)

    print("Beginning training for %d epochs" % epochs)
    self.model.train()
    for epoch in range(epochs):
        for i, data in enumerate(self.trainloader):
            images, y_true = data
            images, y_true = images.to(self.device), y_true.to(self.device)

            self.optimizer.zero_grad()
            outputs = self.model(images)
            loss = self.loss_function(outputs, y_true)
            loss.backward()
            torch.nn.utils.clip_grad_norm_(self.model.parameters(), max_norm=1)
            self.optimizer.step()

            if (i+1) % 100 == 0:
                print("Epoch %d Batch %d -- loss: %.3f" % (epoch+1, i+1, loss))
```

Initialization of weights matters, but the default by Pytorch works fine

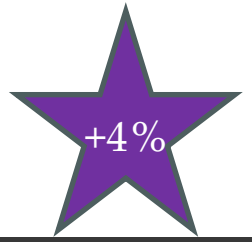
Hands-on coding


```python
def test(self, on_train_set=False):
    holder = {}
    holder['y_true'], holder['y_hat'] = [], []

    if on_train_set is True:
        dataloader = self.trainloader
    else:
        dataloader = self.testloader

    self.model.eval()
    with torch.no_grad():
        for data in dataloader:
            images, y_true = data
            images, y_true = images.to(self.device), y_true.to(self.device)

            outputs = self.model(images)
            _, y_hat = torch.max(outputs, 1)
            holder['y_true'].extend(
                list(y_true.cpu().detach().numpy())
            )
            holder['y_hat'].extend(
                list(y_hat.cpu().detach().numpy())
            )

    y_true_all = holder['y_true']
    y_pred_all = holder['y_hat']
    M = confusion_matrix(y_true_all, y_pred_all)
    print("Confusion matrix: \n", M)
    print(classification_report(y_true_all, y_pred_all))
```

See what happens

- Let's try changing just the activation function, and keep EVERYTHING else constant.

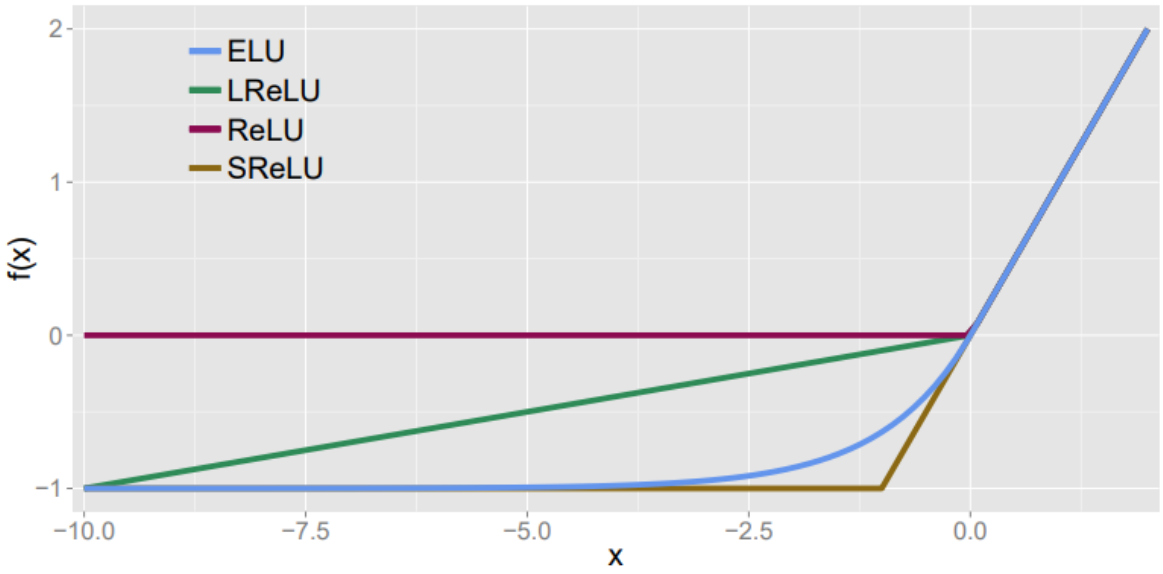
	On Train set	On Test set
ReLU	accuracy 0.57 50000 macro avg 0.57 0.57 0.56 50000 weighted avg 0.57 0.57 0.56 50000	
	Epoch 5 Batch 600 -- loss: 1.359 Epoch 5 Batch 700 -- loss: 1.234	accuracy 0.56 10000 macro avg 0.57 0.56 0.55 10000 weighted avg 0.57 0.56 0.55 10000
ELU	accuracy 0.61 50000 macro avg 0.61 0.61 0.61 50000 weighted avg 0.61 0.61 0.61 50000	
	Epoch 5 Batch 600 -- loss: 1.093 Epoch 5 Batch 700 -- loss: 1.176	accuracy 0.60 10000 macro avg 0.60 0.60 0.59 10000 weighted avg 0.60 0.60 0.59 10000



Difference in Activation

- ELU (Exponential Linear Unit)

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha\,(\exp(x) - 1) & \text{if } x \leq 0 \end{cases}, \quad f'(x) = \begin{cases} 1 & \text{if } x > 0 \\ f(x) + \alpha & \text{if } x \leq 0 \end{cases}$$



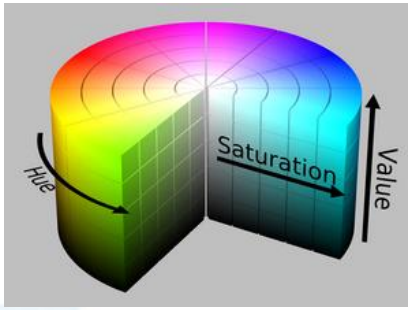
Clevert et al. (2015) https://arxiv.org/pdf/1511.07289

Other things that can be explored

- Model
  - Change architecture, eg. add more conv layers
  - Change loss function
  - Hyperparameters tuning (learning rate, optimizer, scheduler, number of epochs, batch size)
  - Use pre-trained model (recommended; will see tomorrow)
- Regularization
  - Add batch normalization
  - Add dropout
- Data
  - Augmentation
  - Collect more data & of better quality (not quite applicable here, but very helpful in real life)

Data Augmentation

SMU School of Computing and Information Systems

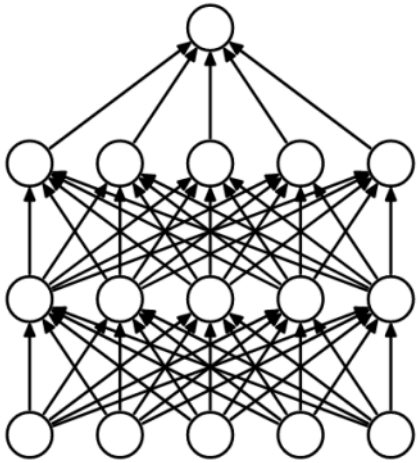


Data Augmentation

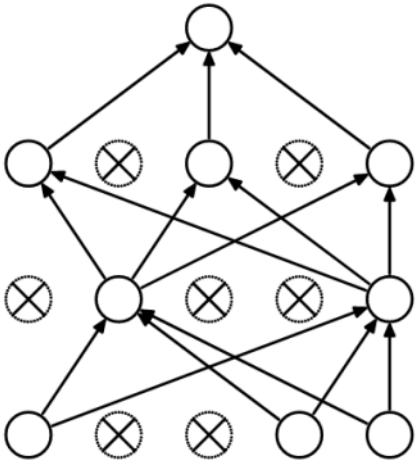
https://developers.google.com/machine-learning/practica/image-classification/preventing-overfitting

Dropout

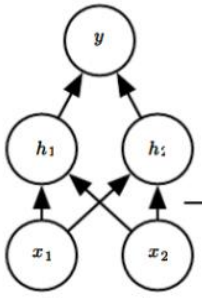
- A form of regularization
  - Binary mask (randomly drop nodes) in a gradient step
  - Reduces over reliance on any particular node
  - Higher the drop probability, the stronger the regularization



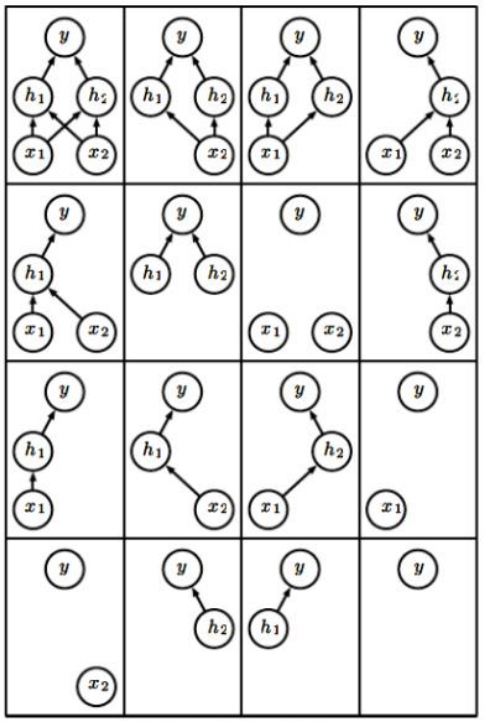
(a) Standard Neural Net



(b) After applying dropout.



Base network



Ensemble of subnetworks

https://cedar.buffalo.edu/~srihari/CSE676/7.12%20Dropout.pdf

See You Tomorrow!

Day 3
(21 Aug 2024)

Quick Recap of Day 2

- Gradient descent
- What it means to 'train' a model
- Understanding images
- Convolutions and CNN
- Multi-class classification
- Practical considerations

Outline for today

School of Computing and Information Systems

- Using pre-trained models
- Changing model architecture
- Working with Colab
- Saving and loading model
- Your own datasets
- Group work

Default ResNet18 architecture


```
model = resnet18(weights=ResNet18_Weights.DEFAULT)

for name, param in model.named_parameters():
    print(name, param.shape)
```

Pre-training is in here

```
layer4.0.conv1.weight torch.Size([512, 256, 3, 3])
layer4.0.bn1.weight torch.Size([512])
layer4.0.bn1.bias torch.Size([512])
layer4.0.conv2.weight torch.Size([512, 512, 3, 3])
layer4.0.bn2.weight torch.Size([512])
layer4.0.bn2.bias torch.Size([512])
layer4.0.downsample.0.weight torch.Size([512, 256, 1, 1])
layer4.0.downsample.1.weight torch.Size([512])
layer4.0.downsample.1.bias torch.Size([512])
layer4.1.conv1.weight torch.Size([512, 512, 3, 3])
layer4.1.bn1.weight torch.Size([512])
layer4.1.bn1.bias torch.Size([512])
layer4.1.conv2.weight torch.Size([512, 512, 3, 3])
layer4.1.bn2.weight torch.Size([512])
layer4.1.bn2.bias torch.Size([512])
fc.weight torch.Size([1000, 512])
fc.bias torch.Size([1000])
```

1000 nodes at output layer

Changing architecture


```
last_layer_name, last_layer = list(model.named_modules())[-1]
featveclen = last_layer.weight.shape[1]
n_class = 10
exec("model.%s = nn.Linear(%s,%s)" %
    (last_layer_name, featveclen, n_class)
)
```

```
layer4.0.conv1.weight torch.Size([512, 256, 3, 3])
layer4.0.bn1.weight torch.Size([512])
layer4.0.bn1.bias torch.Size([512])
layer4.0.conv2.weight torch.Size([512, 512, 3, 3])
layer4.0.bn2.weight torch.Size([512])
layer4.0.bn2.bias torch.Size([512])
layer4.0.downsample.0.weight torch.Size([512, 256, 1, 1])
layer4.0.downsample.1.weight torch.Size([512])
layer4.0.downsample.1.bias torch.Size([512])
layer4.1.conv1.weight torch.Size([512, 512, 3, 3])
layer4.1.bn1.weight torch.Size([512])
layer4.1.bn1.bias torch.Size([512])
layer4.1.conv2.weight torch.Size([512, 512, 3, 3])
layer4.1.bn2.weight torch.Size([512])
layer4.1.bn2.bias torch.Size([512])
fc.weight torch.Size([10, 512])
fc.bias torch.Size([10])
```

What happened?

10 nodes at output layer

Pre-trained models


```
class Classifier:
    def __init__(self, dataMananger):
        self.dataMananger = dataMananger
        self.trainloader = self.dataMananger.trainloader
        self.testloader = self.dataMananger.testloader
        self.classes = self.dataMananger.classes
        self.device = torch.device(
            torch.cuda.current_device() if torch.cuda.is_available() else 'cpu'
        )
        print("Using device %s" % self.device)

        input_shape = \
          (self.dataMananger.C, self.dataMananger.H, self.dataMananger.W)
        # self.model = BasicNet(
        #     input_shape, num_classes=len(self.classes)
        # )
        ### Replace with this ###
        self.model = resnet18(weights=ResNet18_Weights.DEFAULT)
        last_layer_name, last_layer = list(self.model.named_modules())[-1]

        featveclen = last_layer.weight.shape[1]
        n_class = len(self.classes)
        exec("self.model.%s = nn.Linear(%s,%s)" % \
            (last_layer_name, featveclen, n_class)
        )
        ########################
        self.model.to(self.device)
        self.loss_function = nn.CrossEntropyLoss()
```

Performance improvement

School of Computing and Information Systems

	On Train set				On Test set			
Own CNN								
	accuracy			0.61	50000			
	macro avg	0.61	0.61	0.61	50000			
	weighted avg	0.61	0.61	0.61	50000			
	Epoch 5 Batch 600 -- loss: 1.093							
	Epoch 5 Batch 700 -- loss: 1.176							
						accuracy		0.60	10000
						macro avg	0.60	0.60	0.59	10000
						weighted avg	0.60	0.60	0.59	10000
ResNet 18								
	accuracy			0.83	50000			
	macro avg	0.83	0.83	0.83	50000			
	weighted avg	0.83	0.83	0.83	50000			
	Epoch 1 Batch 600 -- loss: 0.867							
	Epoch 1 Batch 700 -- loss: 0.597							
						accuracy		0.77	10000
						macro avg	0.78	0.77	0.77	10000
						weighted avg	0.78	0.77	0.77	10000

More on Colab/bash

- Check current directory

```
!pwd
```

```
/content
```

- Check contents in directory

```
!ls
```

```
sample_data
```

- Connect to Google Drive

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

- Copy from source to destination

```
!cp -r /content/drive/MyDrive/Data/Food5 /content/
```

- Check contents

```
!ls /content/Food5/
```

```
test  train
```

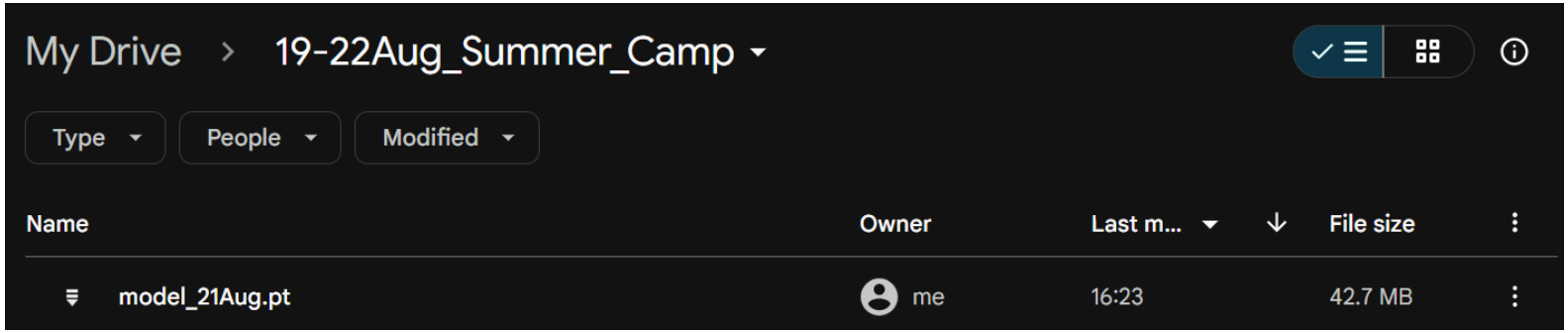
```
!find /content/Food5/ -type f | wc -l
```

```
2750
```

Save & Load

- [Method 1]
  - Saving state_dict of torch model

```
file_name = "/content/drive/MyDrive/19-22Aug_Summer_Camp/model_21Aug.pt"
torch.save(classifier.model.state_dict(), file_name)
```



  - To use, define model with matching architecture and call .load_state_dict

```
new_model.load_state_dict(torch.load(file_name))
```

```
<All keys matched successfully>
```

To show it works

- Re-initialize model, and change last layer

```
new_model = resnet18(weights=ResNet18_Weights.DEFAULT)

last_layer_name, last_layer = list(new_model.named_modules())[-1]
featveclen = last_layer.weight.shape[1]
n_class = 10
exec("new_model.%s = nn.Linear(%s,%s)" % \
    (last_layer_name, featveclen, n_class)
)
new_model.load_state_dict(torch.load(file_name))
```

What happens if this is not done?

- Perform inference to get predictions

…

```
new_model.eval()
with torch.no_grad():
    for data in dataset.testloader:
        images, y_true = data
        images, y_true = images.to(device), y_true.to(device)

        outputs = new_model(images)
        _, y_hat = torch.max(outputs, 1)
```

…

Save & Load

- [Method 2]
  - Universal approach to save and load any python object
  - Save

```
import pickle

file_name_2 = "/content/drive/MyDrive/19-22Aug_Summer_Camp/model_2_21Aug.pkl"

with open(file_name_2, 'wb') as f:
    pickle.dump(classifier.model, f)
```

  - Load

```
with open(file_name_2, 'rb') as f:
    new_model = pickle.load(f)
```

Your own datasets


```
class DataManager:
    def __init__(self, data_dir):
        self.C, self.H, self.W = 3, 224, 224
        self.batch_size = 8
        self.transform =   ...
        self.transform_ag =  ...
        self.trainset = torchvision.datasets.ImageFolder(
            os.path.join(data_dir,'train'),
            transform = self.transform_ag
        )
        self.testset = torchvision.datasets.ImageFolder(
            os.path.join(data_dir,'test'),
            transform=self.transform
        )
        self.class_to_idx = self.trainset.class_to_idx
        self.classes = list(self.class_to_idx.keys())

        self.trainloader = torch.utils.data.DataLoader(
            self.trainset, batch_size=self.batch_size,
            shuffle=True, num_workers=0
        )
        self.testloader = torch.utils.data.DataLoader(
            self.testset, batch_size=self.batch_size,
            shuffle=False, num_workers=0
        )
```

Putting it as a Class


```python
class Classifier:
    def __init__(self, dataMananger, load_model=True):
        self.dataMananger = dataMananger
        self.trainloader = self.dataMananger.trainloader
        self.testloader = self.dataMananger.testloader
        self.classes = self.dataMananger.classes
        self.artifacts_dir = "./artifact/"
        if not os.path.exists(self.artifacts_dir):
            os.makedirs(self.artifacts_dir)

        self.device = torch.device(
            torch.cuda.current_device() if torch.cuda.is_available()
        )
        print("Using device %s" % self.device)

        self.model = resnet18(weights=ResNet18_Weights.DEFAULT)
        self.replace_model_last_layer(len(self.classes))

        if load_model is True:
            self.load_model()

        self.model.to(self.device)
        self.create_loss_function()
```

```python
    def create_loss_function(self):
        def custom_loss(y_pred_logits, y_true):
            """ Do what you want here, then return the loss """
            loss = None
            return loss

        # self.loss_function = custom_loss
        self.loss_function = nn.CrossEntropyLoss()


    def replace_model_last_layer(self, n_class):
        last_layer_name, last_layer = list(self.model.named_modules())[-1]

        featveclen = last_layer.weight.shape[1]
        exec("self.model.%s = nn.Linear(%s,%s)" % \
            (last_layer_name, featveclen, n_class)
        )
```

```python
def save_model(self):
    if not os.path.exists(self.artifacts_dir+"checkpoint/"):
        os.makedirs(self.artifacts_dir+"checkpoint/")

    timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
    torch.save(
        self.model.state_dict(),
        self.artifacts_dir+"checkpoint/model_%s.pt" % timestamp
    )
    print("model_%s.pt successfully saved!" % timestamp)
    return timestamp


def load_model(self, filename=None):
    try:
        if filename is None:
            # get latest file, since files are named by date_time
            filename = sorted(
                os.listdir(self.artifacts_dir+"checkpoint/")
            )[-1]
        self.model.load_state_dict(
            torch.load("%s/checkpoint/%s" % (self.artifacts_dir, filename))
        )
        print("%s successfully loaded..." % filename)
    except:
        print("Unable to load %s..." % filename)
```

Putting it as a Class


```
def train(self, epochs=1, lr=1e-3, save=True):
    self.lr = lr
    self.optimizer = optim.Adam(self.model.parameters(), lr=se
    self.scheduler = \
        lr_scheduler.CosineAnnealingWarmRestarts(self.optimize

    self.history = []
    print("Beginning training for %d epochs" % epochs)
    print("lr: ", self.optimizer.param_groups[0]['lr'])

    self.model.train()
    for epoch in range(epochs):
        print("Epoch %d" % (epoch+1))
        for i, data in tqdm(enumerate(self.trainloader)):
            images, y_true = data
            images, y_true = images.to(self.device), y_true.to

            self.optimizer.zero_grad()
            outputs = self.model(images)
            loss = self.loss_function(outputs, y_true)
            loss.backward()
            torch.nn.utils.clip_grad_norm_(self.model.paramete
            self.optimizer.step()
            self.scheduler.step()

            if (i+1) % 10 == 0:
                self.history.append(loss.item())
                print("Epoch %d Batch %d -- loss: %.3f" % (epo

        if save is True:
            self.save_model()
```

```
def test(self, on_train_set=False):
    holder = {}
    holder['y_true'] = []
    holder['y_hat'] = []

    if on_train_set is True:
        print("Predicting on train set to get metrics")
        dataloader = self.trainloader
    else:
        print("Predicting on eval set to get metrics")
        dataloader = self.testloader

    self.model.eval()
    with torch.no_grad():
        for data in dataloader:
            images, y_true = data
            images, y_true = images.to(self.device), y_true.to(

            outputs = self.model(images)
            _, y_hat = torch.max(outputs, 1)
            holder['y_true'].extend(
                list(y_true.cpu().detach().numpy())
            )
            holder['y_hat'].extend(
                list(y_hat.cpu().detach().numpy())
            )

    y_true_all = holder['y_true']
    y_pred_all = holder['y_hat']
    M = confusion_matrix(y_true_all, y_pred_all)
    print("Confusion matrix: \n", M)
    print(classification_report(y_true_all, y_pred_all))
```

Truncated to fit

Group Task

- A restaurant owner approached you to to create a classifier that can identify the following food in a buffet:
  - Apple pie
  - Cheesecake
  - Chocolate cake
  - French toast
  - Garlic bread

Dataset


```
|-- train
    |-- cheesecake
        |-- 402953.jpg
        |-- 619535.jpg and 298 more files
    |-- chocolate_cake
        |-- 457220.jpg
        |-- 106203.jpg and 298 more files
    |-- garlic_bread
        |-- 760007.jpg
        |-- 2479731.jpg and 298 more files
    |-- french_toast
        |-- 1416043.jpg
        |-- 2273627.jpg and 298 more files
    |-- apple_pie
        |-- 299931.jpg
        |-- 89035.jpg and 298 more files
```

```
|-- test
    |-- cheesecake
        |-- 250 files
    |-- chocolate_cake
        |-- 250 files
    |-- garlic_bread
        |-- 250 files
    |-- french_toast
        |-- 250 more files
    |-- apple_pie
        |-- 250 more files
```

Evaluation metric

- Not all mistakes are equal

Misclassification cost matrix		Prediction				
		Apple pie	Cheese-cake	Chocolate cake	French toast	Garlic bread
True label	Apple pie	0	2	5	3	4
	Cheesecake	2	0	6	4	3
	Chocolate cake	5	6	0	1	2
	French toast	3	4	1	0	5
	Garlic bread	4	3	2	5	0

Group score = 60% of unseen dataset and 40% of test set

Group Work

- Please get into groups of 5
- I will walk around to offer guidance
- Each group must submit a model (compatible with the given jupyter notebook) by **9am on 21 Aug**
- Presentation of 6 to 8 minutes per group, any number of speakers
  - Max 5 slides
  - Share your model, loss function, as well as other modifications
  - Describe the process
  - Suggest feasible improvements
  - Bonus (optional): create a simple app

See You Tomorrow!

Day 4
(22 Aug 2024)

Quick Recap of Day 3

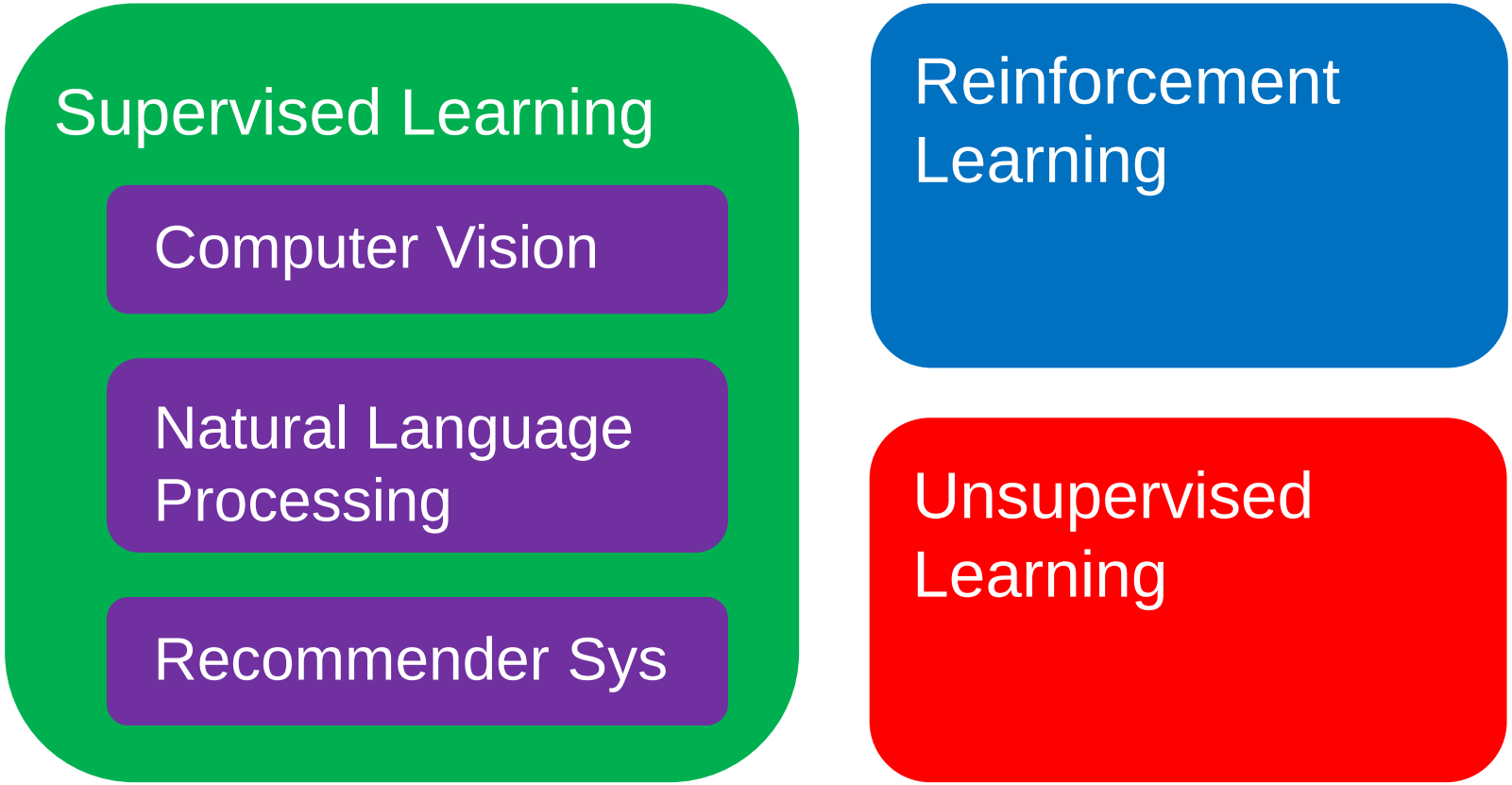
- Using pre-trained models
- Changing model architecture
- Working with Colab
- Saving and loading model
- Your own datasets
- Group work

Outline for today

- Final touch-ups
- Group presentations
- Unsupervised Learning in brief
- Preview of Reinforcement Learning
- Career in the Data Science field

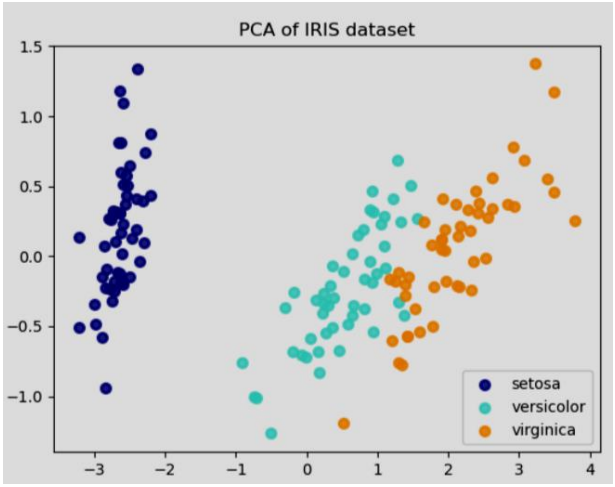
Group Presentations

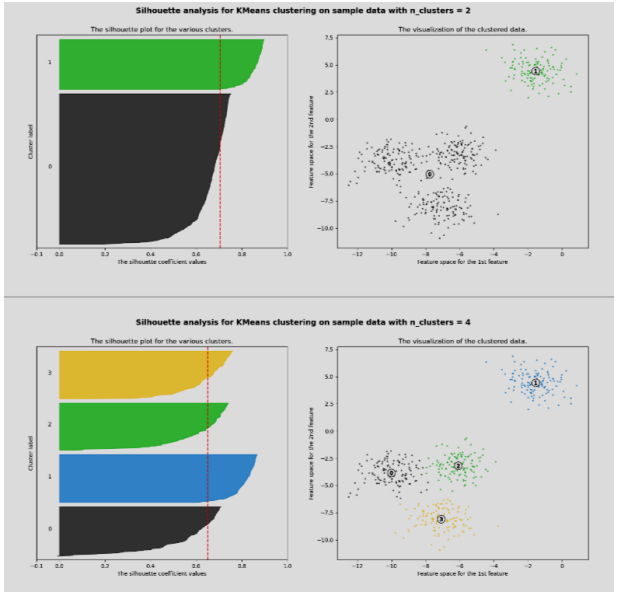
Possible Future Directions



Essence of Unsupervised Learning

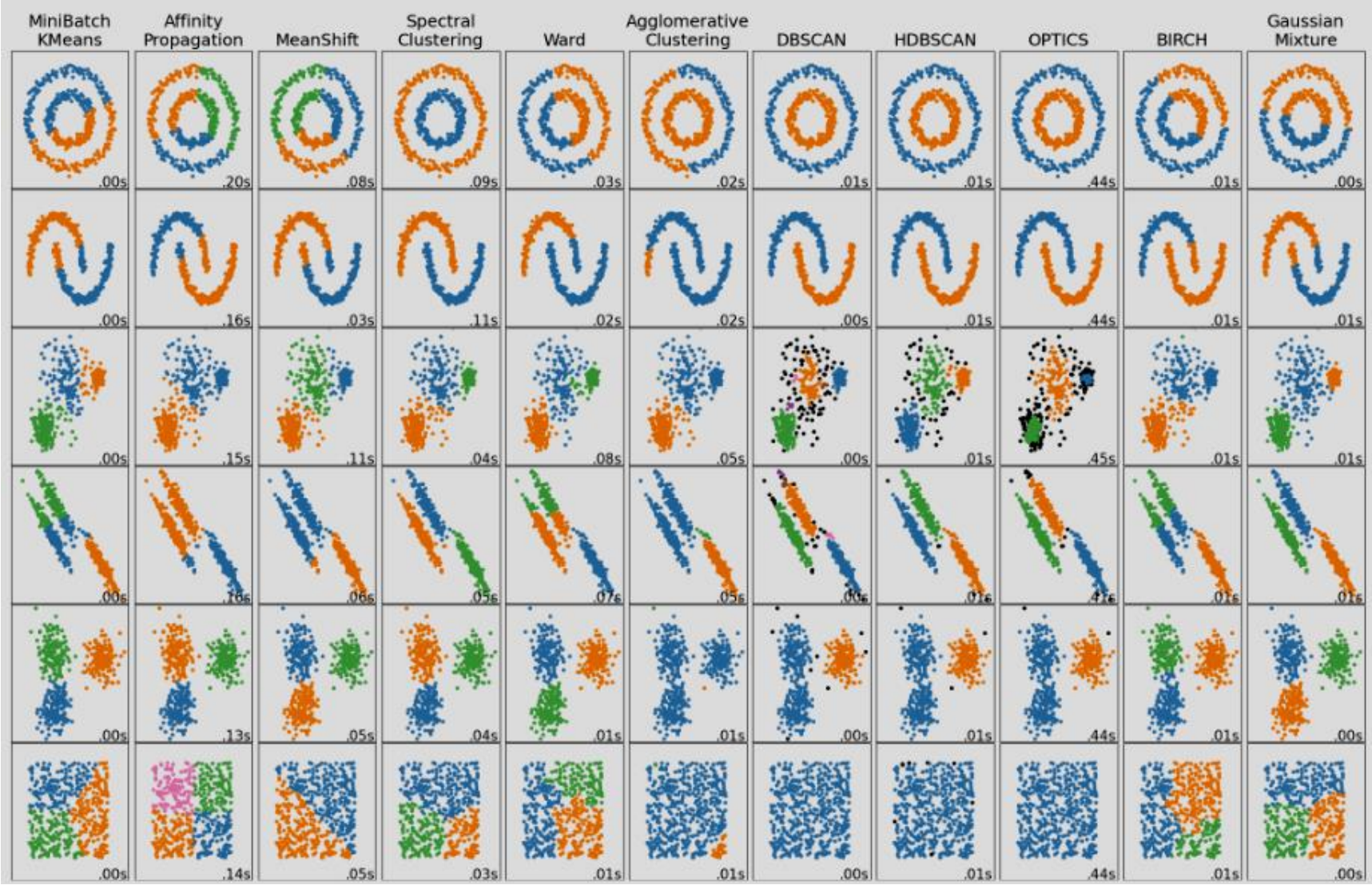
- Discover patterns within data without labels
  - Clustering (useful for customer segmentation)
  - Anomaly Detection
  - Dimensionality Reduction
- Can be a component in supervised learning or RL
- Evaluation metrics are different since there is no 'truth'





https://scikit-learn.org/stable/auto_examples/decomposition/plot_pca_vs_lda.html
https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_silhouette_analysis.html

No single best method



https://scikit-learn.org/stable/auto_examples/cluster/plot_cluster_comparison.html

Essence of RL

- To choose actions that would lead to higher expected cumulative rewards in the long term
  - Rewards should be consistent with the goal in mind, without asserting how you want the goal to be reached
  - Actions taken now will influence the state-action space later; ie. availability of options
- How do we determine what is 'expected'?
  - Through knowledge of the environment transition probabilities (unlikely in many real cases)
  - or, Through experience
- When to use RL?

RL against humans

- AlphaGo & AlphaGo Zero by DeepMind
  - Defeated Lee Sedol in 2016.
  - 10^{170} possible configurations; more than atoms in the universe
  - AlphaGo Zero learns by playing against itself, starting from completely random play.
- DOTA2 by OpenAI
  - consumed 800 petaflop/s-days and experienced about 45,000 years of Dota self-play over 10 real-time months
- GPT3 by OpenAI
  - There is no 'perfect' label to perform supervised learning on
  - Trainers rate outputs; want output with large rewards

Reinforcement Learning

Action Space

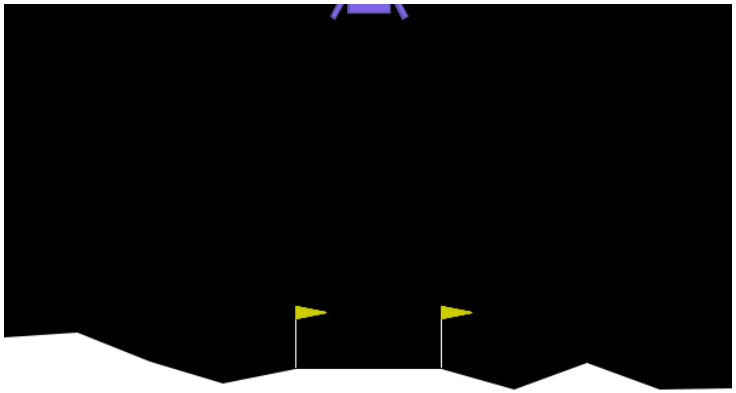
- (1) do nothing
- (2) fire left engine
- (3) fire main engine
- (4) fire right engine

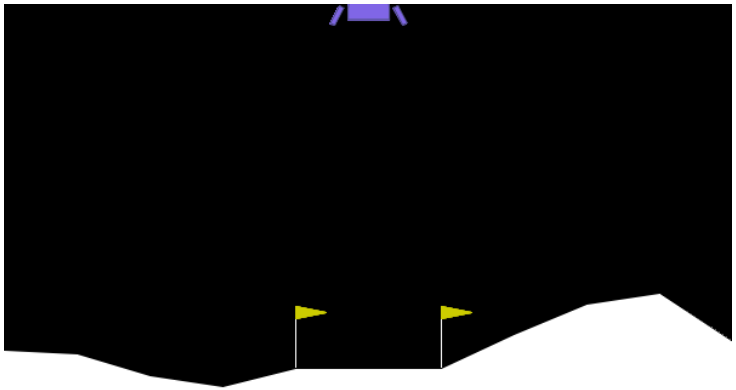
Observation Space

- 8D vector

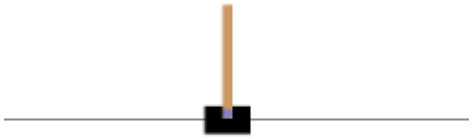
Rewards

- 100 pts for safe landing
- -100 pts for crash
- 10 pts for leg on ground
- -0.3 pts firing main engine
- -0.03 pts firing side engine





Classic Environments

Discrete Action	Continuous Action
 Acrobot  Cart Pole	 Pendulum
 Mountain Car	 Mountain Car Continuous

https://gymnasium.farama.org/environments/classic_control/

Real Projects

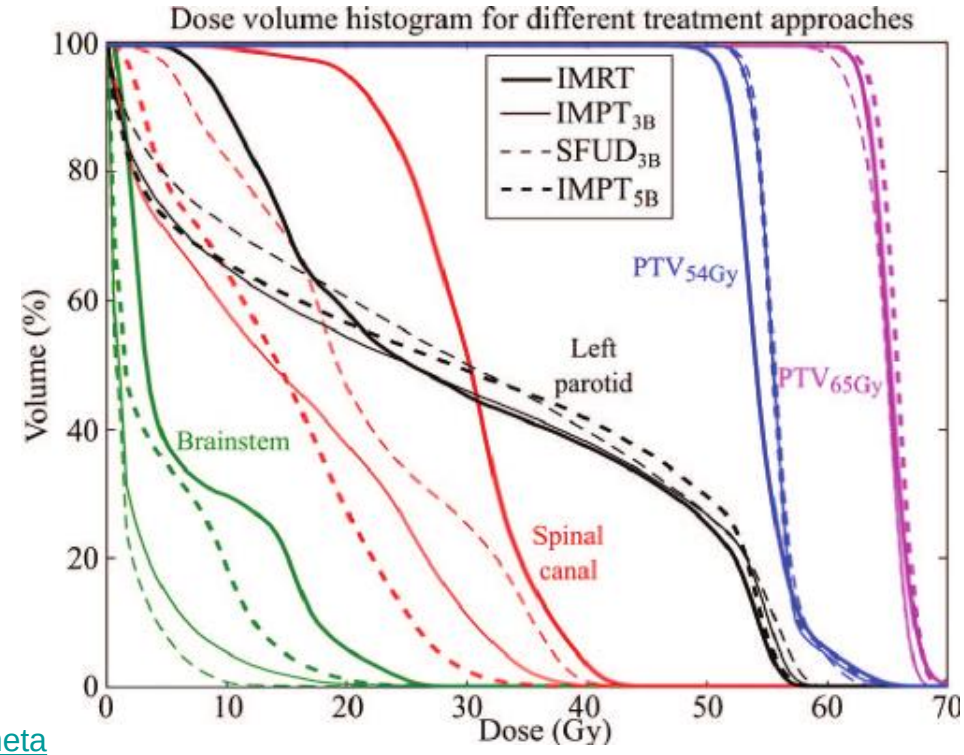
National Institutes of Health (NIH) (.gov)
https://www.ncbi.nlm.nih.gov › articles › PMC9297721

The Development of a Deep Reinforcement Learning ...

by D Sprouts · 2022 · Cited by 11 — In this work, we show an end-to-end study to train a **deep reinforcement learning** (DRL) based virtual **treatment** planner (VTP) that can behave like a…

Abstract · INTRODUCTION · METHODS AND MATERIALS · DISCUSSION

The development of a deep reinforcement learning network for dose-volume-constrained treatment planning in prostate cancer intensity modulated radiotherapy

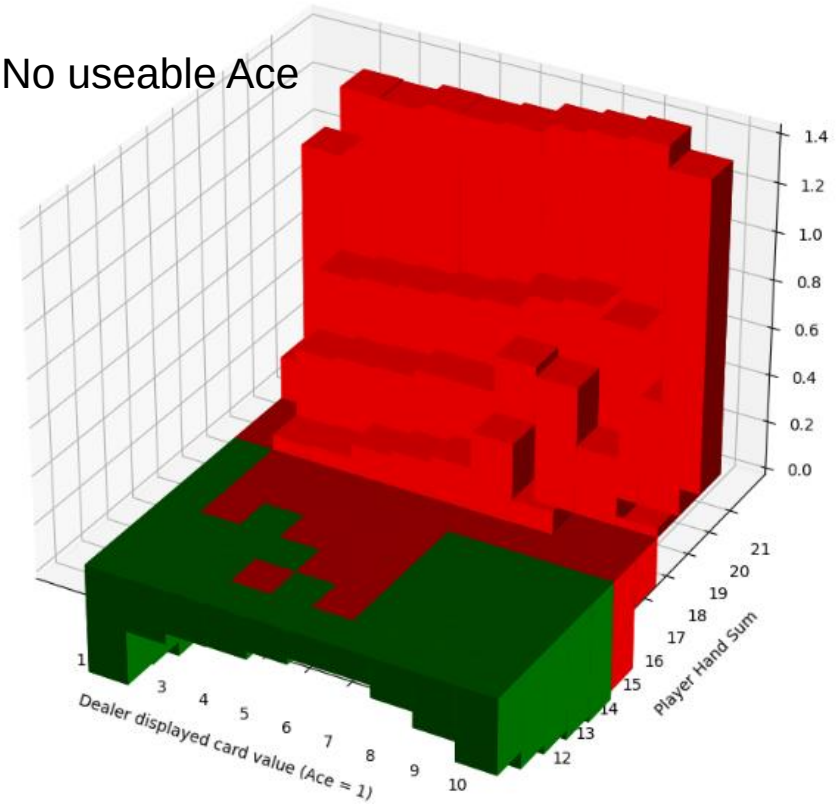


https://iopscience.iop.org/article/10.1088/2057-1976/ac6d82/meta
Cone-beam computed tomography and deformable registration-based “dose of the day” calculations for adaptive proton therapy

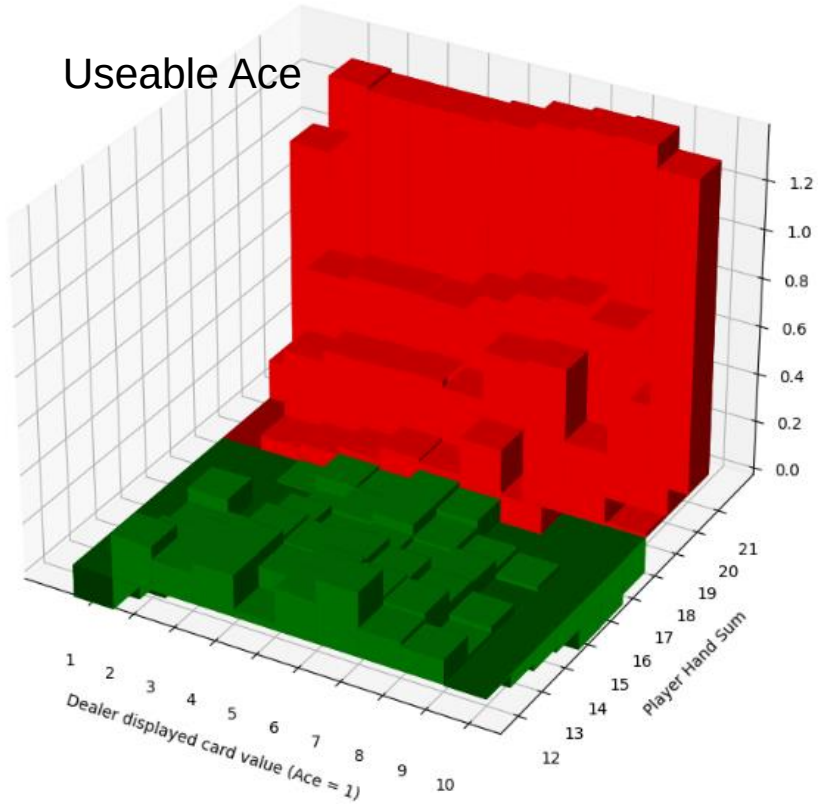
Sample assignment

SMU Singapore Management University
School of Computing and Information Systems

No useable Ace



Useable Ace



Greedy Policy. Green regions indicate states where best action is to Hit.
Red regions indicate states where best action is to Stand

Going further

- Would take at least hundreds of hours
  - No such thing as 'hero-to-zero' in a few hours or days.
  - New knowledge is constantly being added.
- Consider a professional Masters
  - E.g. Master of IT in Business (MITB) by
    SMU School of Computing and Information Systems

MITB brochure

MITB administrative info

Working in this field

- My experience in the private sector and public section
  - What you need to know
  - Getting the job
  - Things you would be doing
  - Don't be a hammer
  - The nice aspects and the challenges
  - What lies ahead

Disclaimer: Take what I share with you as a "datapoint". Do not "overfit". Speak to others, see things in the larger context, and make sense of things yourself.

Thank You
for having me in your journey