Today's Lecture

1. Get to know each other
   - Instructor & TAs
   - Students
2. Module Introduction
3. Logistics
4. Linear Regression

Let’s get to know each other.

If you cannot come to the classroom

1. Join Zoom Meeting
   https://nus-sg.zoom.us/j/3997998157
2. Meeting ID: 399 799 8157

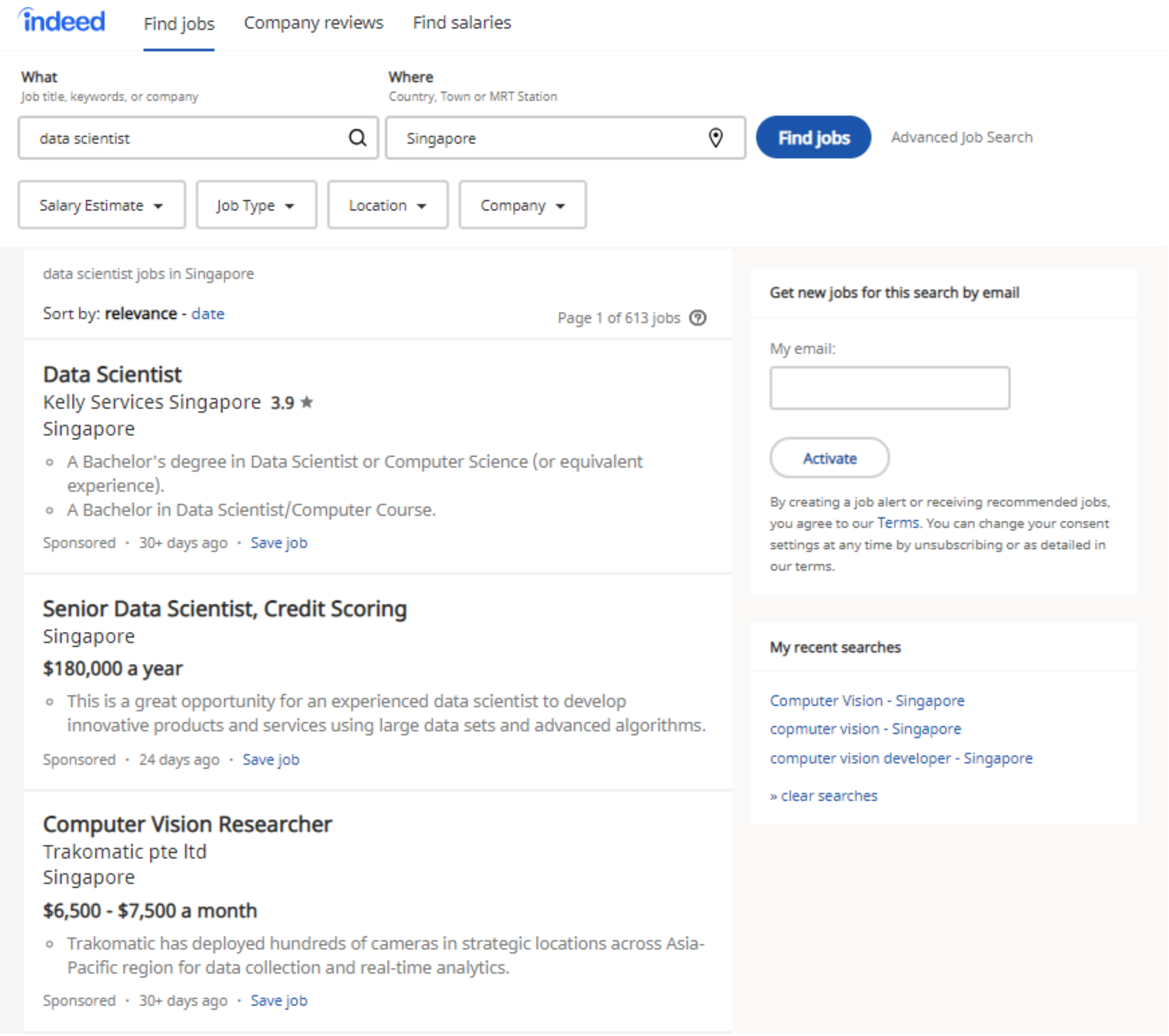
I heard that everyone got A's last term and am hoping for an easy boost to my GPA.

maybe, maybe not...

I want to earn the big bucks as a data scientist / ML engineer at Google / Amazon / Facebook.

check out

CS5260 – NN & Deep Learning II
CS4243 – Computer Vision & Pattern Recognition
CS4248 – Natural Language Processing



Page 1 of 613

$180,000 a year

I actually wanted CS5XXX but couldn't get that module.

too bad

Can't get into this module? Email for guest access.

My PhD advisor is making me take this course so that I can use deep learning in our research.

Neural Networks and Deep Learning:

- Andrew Ng
- https://www.coursera.org/learn/neural-networks-deep-learning

CSC 321: Intro to Neural Networks and Machine Learning

- Roger Grosse
- http://www.cs.toronto.edu/~rgrosse/courses/csc321_2017/

Neural Networks for Machine Learning

- Geoffrey Hinton
- https://www.coursera.org/learn/neural-networks

CS231n: Convolutional Neural Networks for Visual Recognition

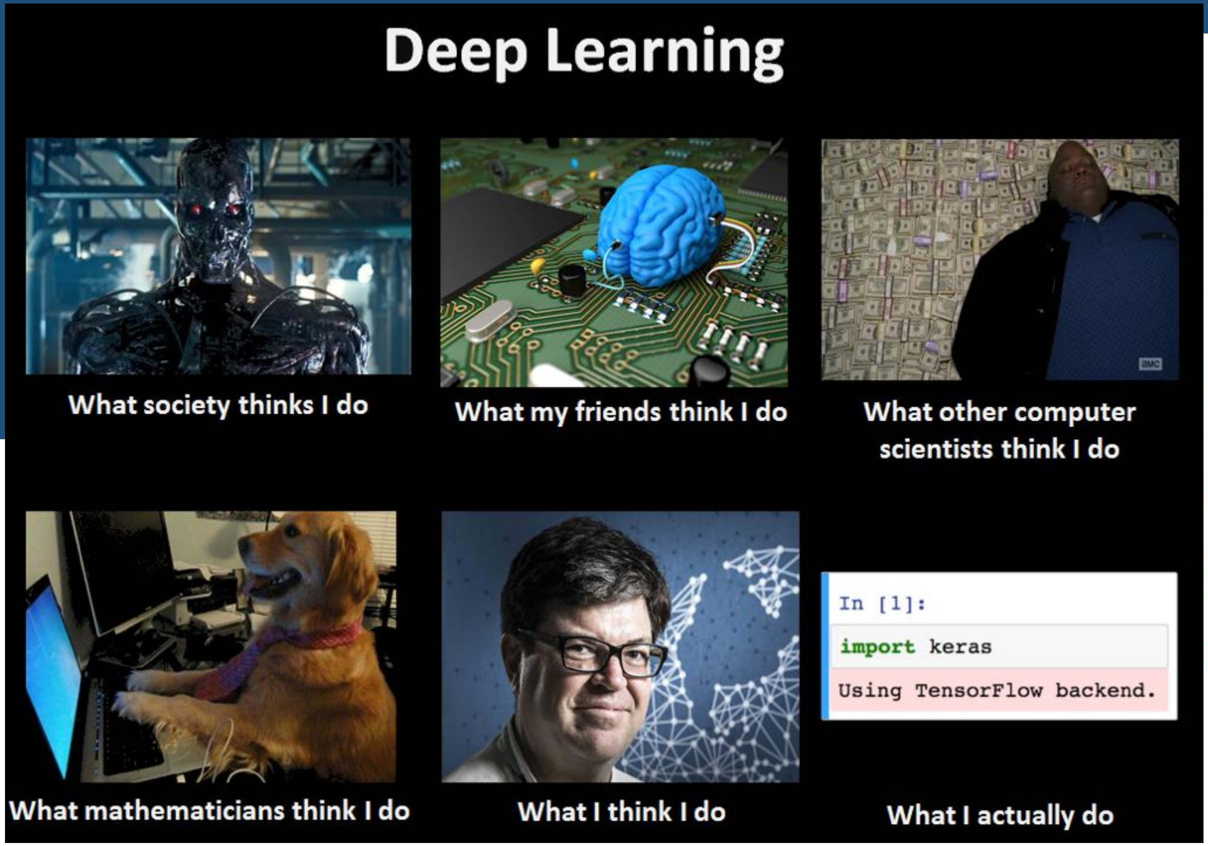
- Fei-Fei Li, Justin Johnson, Serena Yeung
- http://cs231n.stanford.edu/

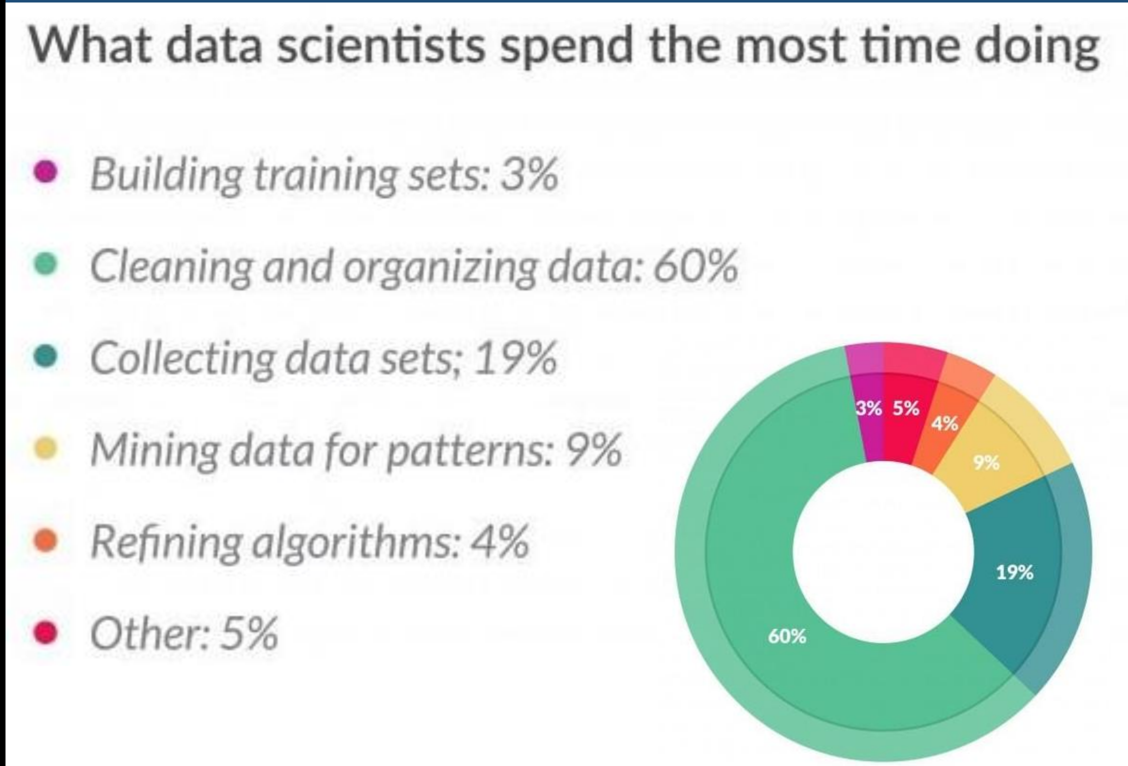
CS224d: Deep Learning for Natural Language Processing

- Richard Socher
- http://cs224d.stanford.edu/

Slide credit: Wang Wei

AI is cool and I want to build the next Skynet.





Intended Learning Outcomes

1. Explain the principles of the operations of different layers and training algorithms

2. Compare different neural network architectures

3. Implement popular neural networks

4. Solve problems using neural networks and deep learning techniques

Slide credit: Wang Wei

Course Logistics

Pre-requisite

Modules

- Machine Learning (CS3244)
  - Or https://www.coursera.org/learn/machine-learning
- Linear Algebra (MA1101R)
  - Or https://ocw.mit.edu/courses/mathematics/18-06-linear-algebra-spring-2010/
- Calculus (MA1521)
- Probability (ST2334)
- Brief summary

Coding

- Python (ONLY, version 3.x)
  - Numpy
  - Keras, TensorFlow, PyTorch, MxNet, Colossal-AI
- Jupyter notebook

Slide credit: Wang Wei

Syllabus & Schedule

- Learn various neural network models
  - From shallow to deep neural networks
  - model structures, training algorithms, tricks and applications
  - principles, theory & coding practices
- Check Canvas (not LumiNUS) for the latest plan
  - https://canvas.nus.edu.sg/

Slide credit: Wang Wei

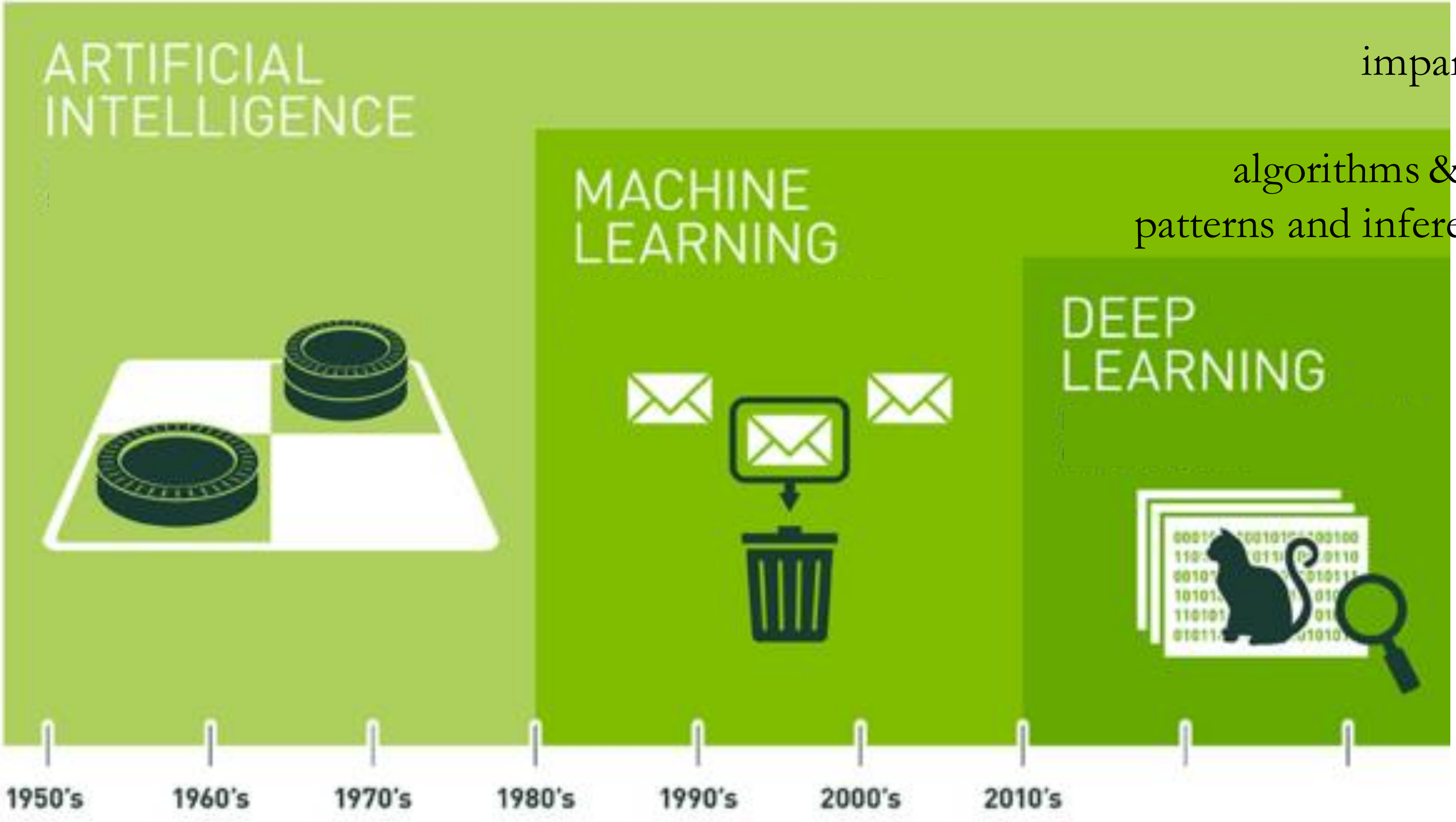
GPU machines

- SoC GPU machines:
  - https://dochub.comp.nus.edu.sg/cf/guides/compute-cluster/hardware
- NSCC (National Super-Computing Center)
  - Free for NUS students
  - The libraries are sometimes outdated
  - Jobs will be submitted into a queue for executing
- Google Cloud Platform
  - Some free credit for new register
  - GPU is expensive
- Amazon EC2 (g2.8xlarge)
  - Some free credit for students
  - GPU is expensive

Important notes if you use cloud platforms:
- STOP/TERMINATE the instance immediately after your program terminates
- Check usage status frequently to avoid high outstanding bills
- Amazon/Google may charge you for additional storage volume

Slide credit: Wang Wei

Terminology Overload: AI vs. ML vs. DL



Since an early flush of optimism in the 1950s, smaller subsets of artificial intelligence – first machine learning, then deep learning, a subset of machine learning – have created ever larger disruptions.

imparting cognitive abilities to machines

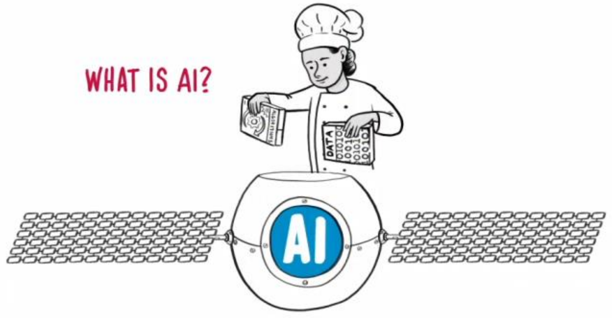
algorithms & models to perform tasks based on patterns and inference instead of specific instructions

branch of machine learning using deep neural networks

https://blogs.nvidia.com/blog/2016/07/29/whats-difference-artificial-intelligence-machine-learning-deep-learning-ai/

Artificial Intelligence (AI)

- 1950's
- "Human intelligence exhibited by machines"
  - Expert systems; Rules
  - Machine learning algorithms
- Narrow AI:
  - image recognition
  - machine translation
  - recommender system
  - …



https://www.youtube.com/watch?v=nASDYRkbQIY

Slide adapted from Wang Wei

Machine Learning (ML)

- 1980's
- "An approach to achieve AI through systems that can learn from experience to find patterns in that data"
    - Can we code a program to do
        - Image recognition

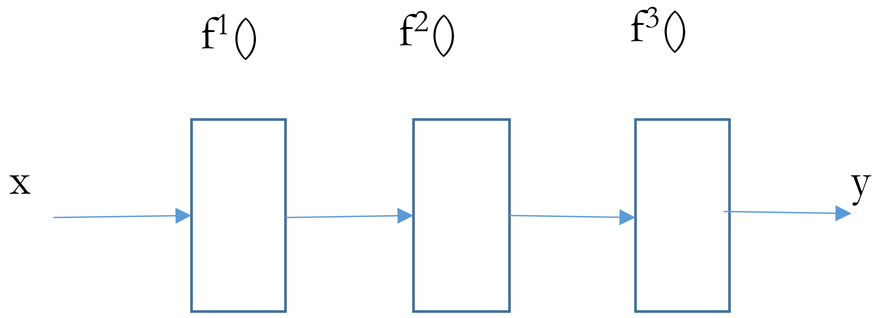
This Photo by Unknown Author is licensed under [CC BY-NC-ND](CC BY-NC-ND)

        - Speech recognition

Slide credit: Wang Wei

Neural networks and Deep Learning

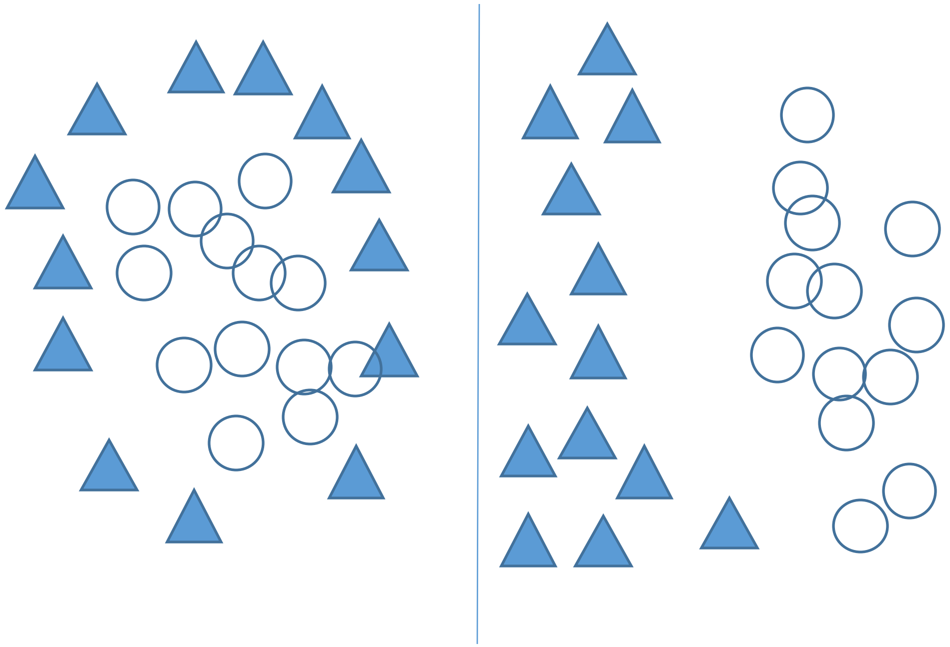
- 1950's / 2010's
- A class of machine learning algorithm that use a cascade of multiple layers of nonlinear processing units for feature extraction and transformation---Wikipedia

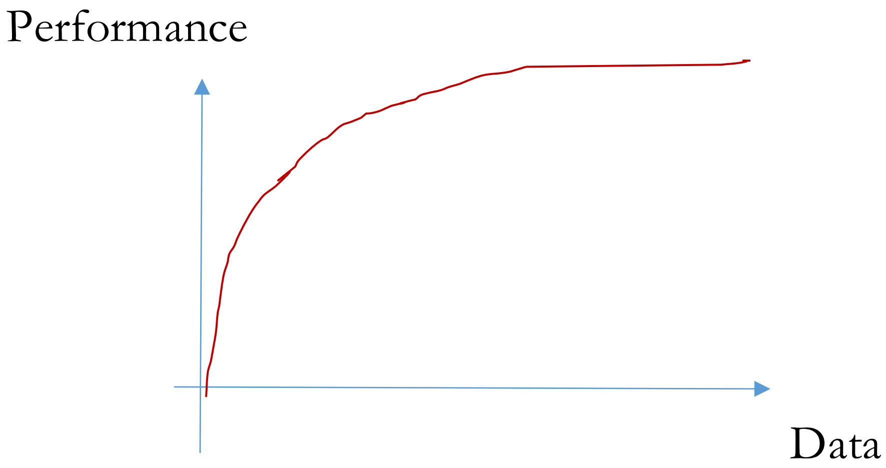


Slide credit: Wang Wei

Neural networks and Deep Learning

- Feature learning (transformation)

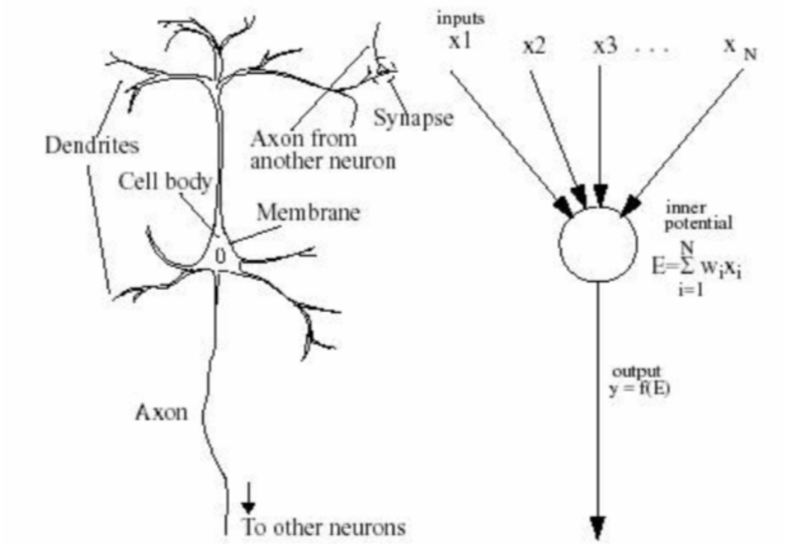




Slide credit: Wang Wei

“Neural” Networks?

The origins of neural networks come from efforts of mathematically representing information processing in biological systems (hence the term “neural”).



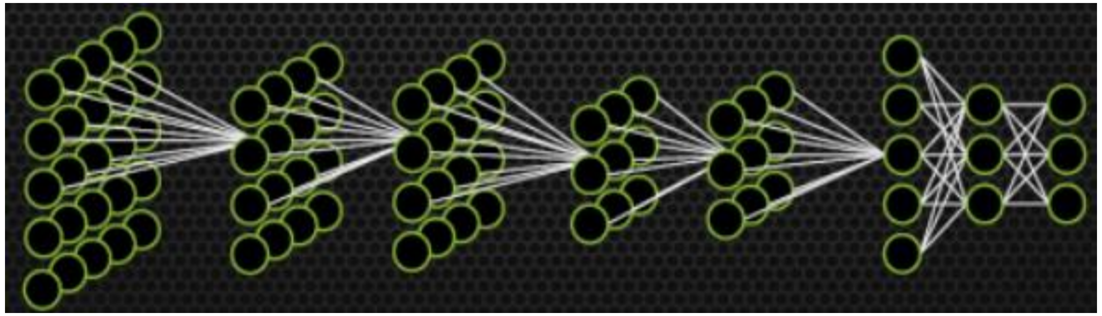
Neural nets/perceptrons are **loosely** inspired by biology. But they certainly are **not** a model of how the brain works, or even how neurons work.

http://commonsenseatheism.com/wp-content/uploads/2011/12/biological-and-artificial-neurons.jpg

Our interests in neural networks are less about biological systems and more about efficient models for statistical pattern recognition.

Neural networks (NN)

- Linear, polynomial, logistic regression
- Perceptron and multi-layer perceptron (MLP)
- Convolutional neural network (CNN)
- Recurrent neural networks (RNN)
- Transformers (Attention; Seq to Seq)
- Generative adversarial networks (GAN)
- (Restricted) Boltzmann machine
- Deep belief network
- Spike neural network
- Radial basis function neural network
- Hopfield networks



Source from [3]

Slide credit: Wang Wei

History [2]

1957 First perceptron, Frank Rosenblatt

1974 Backpropagation, Paul Werbos

1980 Neocogitron (inspired CNN), Kunihiko Fukushima

1986 Recurrent Neural Network (RNN), David Rumelhart

1990 LeNet

1998 Hochreiter & Schmidhube, LSTM

2006 Deep Belief Networks, Geoffrey Hinton

2012 AlexNet, Alex Krizhevsky & Geoffrey Hinton

2014 GoogleNet & VGG

2014 Generative Adversarial Networks

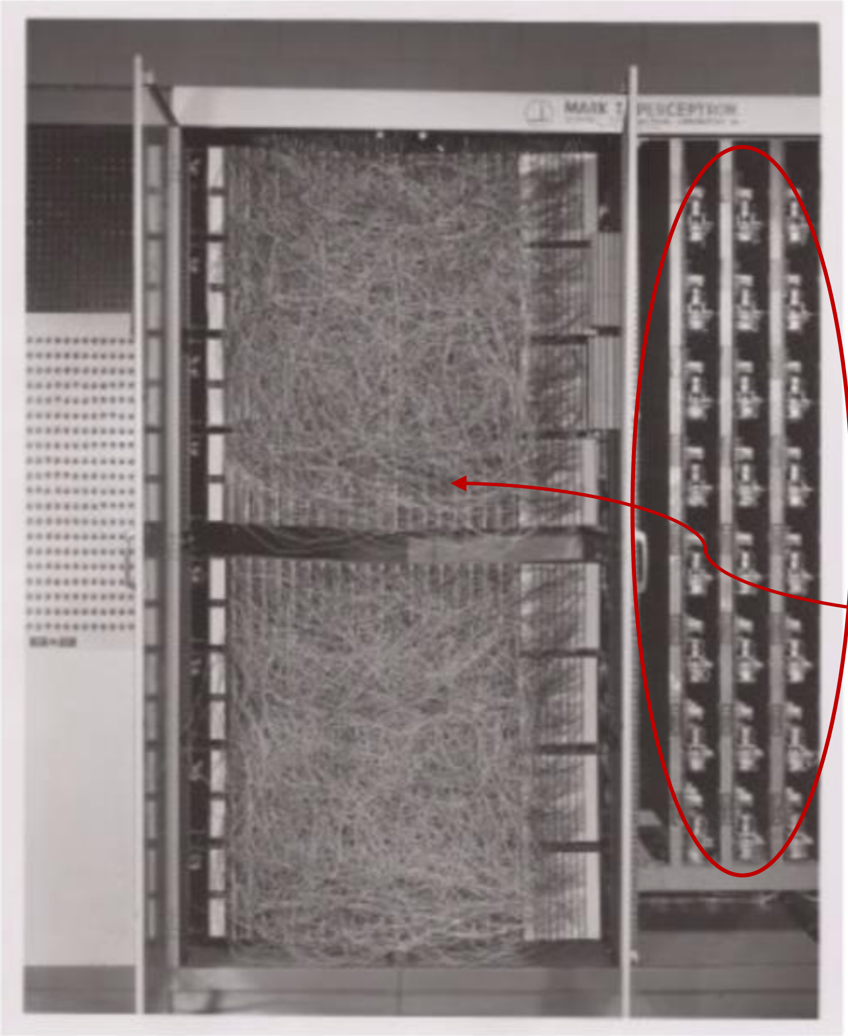
2015 ResNet

2015 AlphaGo
BERT
GPT-3

Refer to http://people.idsia.ch/~juergen/who-invented-backpropagation.html for more papers

Slide adapted from Wang Wei

The Perceptron Machine (1957)



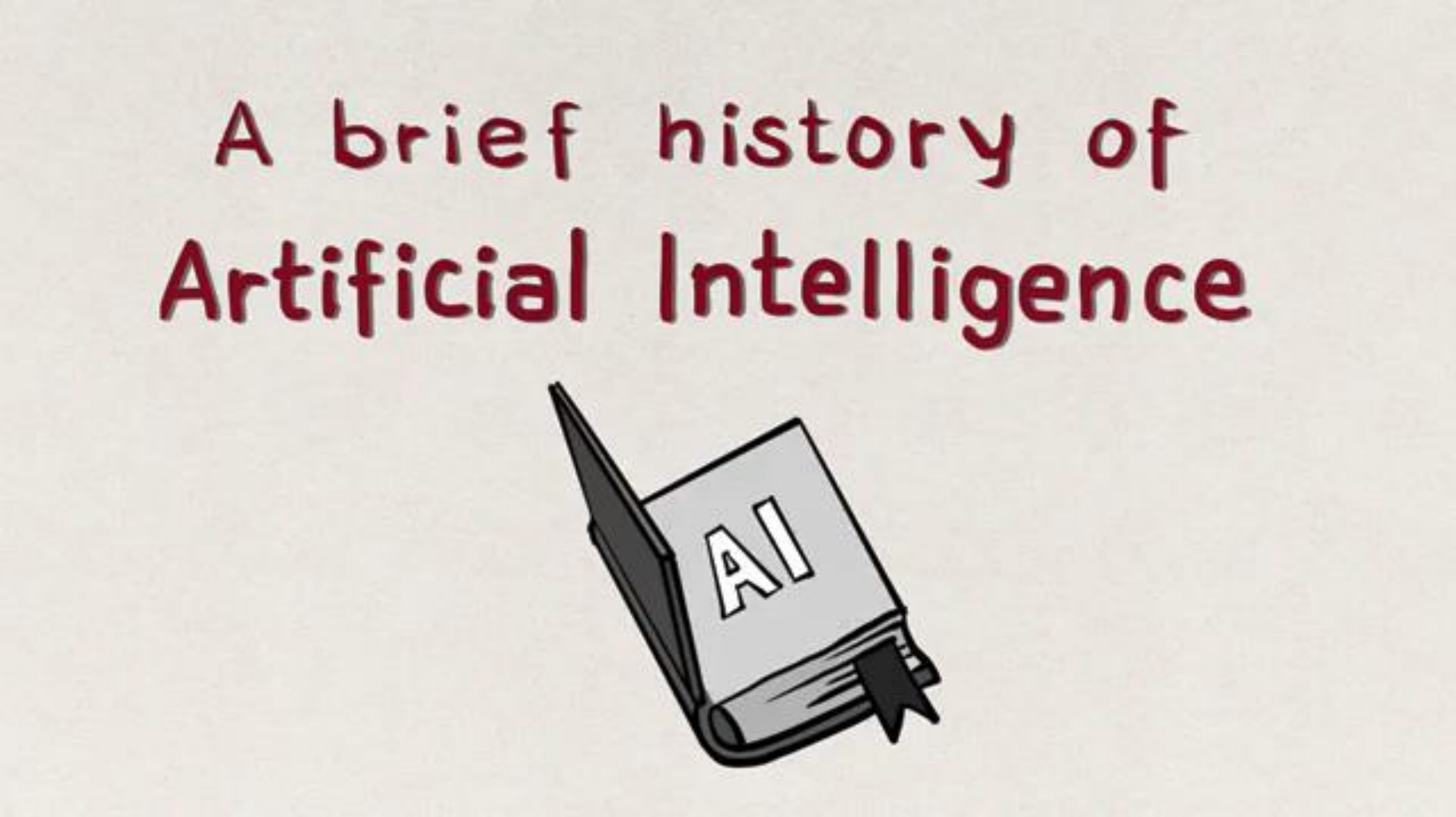
The Mark I Perceptron machine was a hardware implementation of the perceptron algorithm.

The Perceptron machine was connected to a 20x20 array of photocells producing a 400 pixel image (essentially a camera).

The nest of wires create different combinations of inputs.

Arrays of adjustable variable resistors, which could be used as adaptive ("hand-tuned") weights.

https://en.wikipedia.org/wiki/Perceptron#/media/File:Mark_I_perceptron.jpeg



https://www.youtube.com/watch?v=yaL5ZMvRRqE

Applications

Computer Vision

- Image classification
- Object detection, demo
- Scene text recognition
- Neural style transferring
- Image generation

Natural language processing

- Question answering
- Machine translation

Speech

- Speech recognition, e.g. Amazon Echo

Slide credit: Wang Wei

Univariate Linear Regression

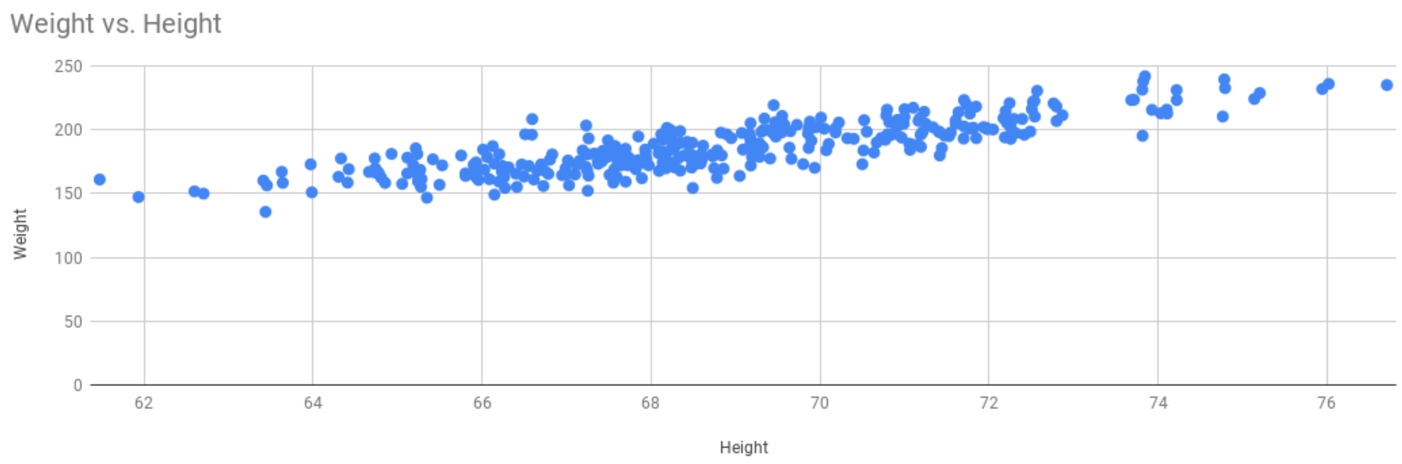
Weight vs Height

Problem definition:

- Predict the weight given the height of a person.

Data

- Input: Height (inches)
- Output: Weight (pounds)
- Split the data into
  - 80% for training
  - 20% for evaluation



Data source: https://www.kaggle.com/mustafaali96/weight-height

Slide credit: Wang Wei

Modeling

- Notation
  - A person is called an example/instance
  - Height denoted as $x \in R$: input feature
  - Weight denoted as $y \in R$: the target or ground truth
- Map from input to output by **linear regression**
  - $\tilde{y} = xw + b, w \in R, b \in R$
  - w is the slope and b is the intercept
  - $\tilde{y}$ is called the **prediction**

Slide credit: Wang Wei

Training

- Learn w and b to fit the training data $S_{train} = \left\{(x^{(i)}, y^{(i)})\right\}, i = 1 \dots m$
  - That fit the data well
  - How to measure the quality of w and b?
- Loss function
  - The **smaller** the loss value, the better the prediction (closer to the target)
  - $L(x, y|w, b) = |\tilde{y} - y|$
  - $L(x, y|w, b) = \frac{1}{2}||\tilde{y} - y||^2 = \frac{1}{2}(\tilde{y} - y)^2$
    - Easier for optimization/training because it is differentiable
    - The coefficient $\frac{1}{2}$ is to make the gradient simple
- Define the training objective
  - $J(w, b) = \frac{1}{m}\sum_{i=1}^{m} L\left(x^{(i)}, y^{(i)} \middle| w, b\right)$

Slide credit: Wang Wei

Optimization/training

Tune the model parameters over the training instances to minimize the average loss, i.e., the training objective

$$\min_{w,b} J(w,b)$$

$$\to \min_{w,b} \frac{1}{m} \sum_{i=1}^{m} L\left(x^{(i)}, y^{(i)} \middle| w, b\right)$$

$$\to \min_{w,b} \frac{1}{2m} \sum_{i=1}^{m} \left(wx^{(i)} + b - y^{(i)}\right)^2$$

Slide credit: Wang Wei

Optimization/training

Fix b and learn w

$$\min_{w} \frac{1}{2m}\sum_{i=1}^{m}\left(x^{(i)}\right)^2 w^2 + \frac{1}{m}\sum_{i=1}^{m} x^{(i)}\left(b - y^{(i)}\right) w + \frac{1}{2m}\sum_{i=1}^{m}\left(b - y^{(i)}\right)^2$$

$$\rightarrow \min_{w} c_1 w^2 + c_2 w + c_3$$

Fix w and learn b

Slide credit: Wang Wei

Gradient for univariate simple functions

Gradient: recall high school calculus and how to take derivatives

- $\frac{\partial f(x)}{\partial x} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$
- $f(x) = 3, \frac{\partial f(x)}{\partial x} = 0$
- $f(x) = 3x, \frac{\partial f(x)}{\partial x} = 3$
- $f(x) = x^2, \frac{\partial f(x)}{\partial x} = 2x$

Slide credit: Wang Wei

Gradient for univariate simple functions

- Subgradient for non-differentiable points
  - $f(x) = |x|, \frac{\partial f(x)}{\partial x} = \begin{cases} 1, for\ x > 0 \\ -1, for\ x < 0 \\ a \in [-1, 1], for\ x = 0 \end{cases}$

    Sign of the variable if it's non-zero, anything in [−1, 1] if it's zero.

  - $f(x) = \max(x, 0), \frac{\partial f(x)}{\partial x} = \begin{cases} 1, for\ x > 0 \\ 0, for\ x < 0 \\ a \in [0, 1], for\ x = 0 \end{cases}$

Slide credit: Wang Wei

Gradient for univariate composite functions

We use $f'(x)$ and $\frac{\partial f(x)}{\partial x}$ interchangeably for the gradient of f(x) with respect to x, where x and f(x) are scalars

- $f(x) = g(x) + h(x)$
  - $f(x) = 3x + x^2$
- $f'(x) = g'(x) + h'(x)$

- $f(x) = g(x)h(x)$
  - $f(x) = xx^2$
- $f'(x) = g'(x)h(x) + g(x)h'(x)$

- $f(x) = \frac{g(x)}{h(x)}$
  - $f(x) = \frac{e^x}{x}$
- $f'(x) = \frac{g'(x)h(x) - g(x)h'(x)}{h(x)^2}$

- $f(x) = g(u), u = h(x)$
  - $f(x) = \ln(x + x^2)$
- $f'(x) = g'(u)h'(x)$

Slide credit: Wang Wei

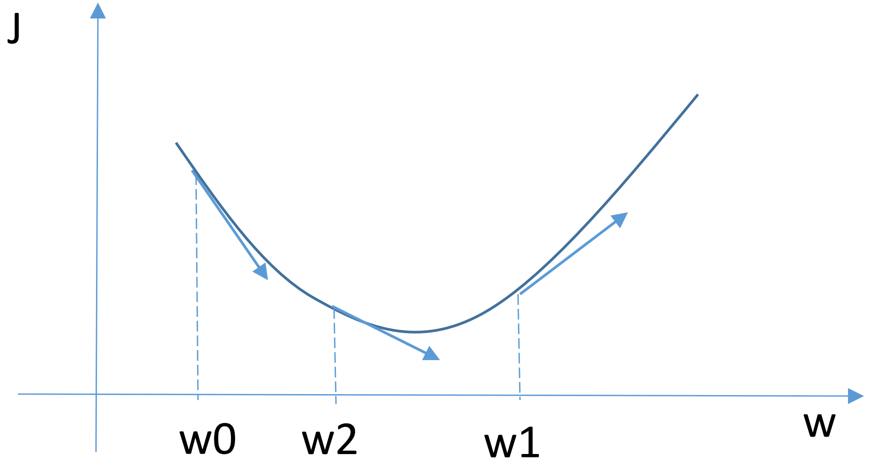
Training by gradient descent

$$J(w) \approx w^2 c_1 + w c_2 + c_3$$

Find the **w** for which we have the lowest* J.

For example:

$c_1 = 1, c_2 = -2, c_3 = 1$



α is the **learning rate**, which controls the moving step length. It's important for convergence. If it is large, w oscillates around the optimal position. If it is small, it takes many iterations to reach the optimum.

Initialize w as w0

Compute $\frac{\partial J}{\partial w}|_{w=w0}$, negative;

Move w from w0 to the right by

$$\mathrm{w1} = \mathrm{w0} - \alpha \frac{\partial J}{\partial w}|_{w=w0}$$

Compute $\frac{\partial J}{\partial w}|_{w=w1}$, positive;

Move w from w1 to the left by

$$\mathrm{w2} = \mathrm{w1} - \alpha \frac{\partial J}{\partial w}|_{w=w1}$$

Compute $\frac{\partial J}{\partial w}|_{w=w2}$, negative

Move w from w2 to the right by

$$\mathrm{w3} = \mathrm{w2} - \alpha \frac{\partial J}{\partial w}|_{w=w2}$$

Slide credit: Wang Wei

Training by gradient descent

- Gradient descent algorithm for optimization
- Set w = 0.1 or a random number
- Repeat
  - For each data sample, compute $\tilde{y} = xw + b$
  - Compute the average loss, $\sum_{<x,y>\in S_{train}} L(x, y|w, b) \,/ |S_{train}|$
  - Compute $\frac{\partial J}{\partial w}$
  - Update $w = w - \alpha \frac{\partial J}{\partial w}$

Update w, b repeatedly…

Slide credit: Wang Wei

Machine learning pipeline

Prepare data → Extract feature → Create model → Find loss function → Conduct training → Evaluate → Deploy

Slide credit: Wang Wei

Multivariate Linear Regression

Multivariate linear regression

- Consider the problem of house price prediction
- Each instance in the training dataset
  - Denote the features using a column vector
    - $\boldsymbol{x} \in R^{n\times 1}$: x_i is the i-th feature
    - size, floor, location, age, lease, etc.
  - Target $y \in R$: price

$$\boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{pmatrix}$$

Slide credit: Wang Wei

Model

- Map from input to output

  $y = \boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x} + b, \boldsymbol{w} \in R^{n\times 1}, b \in R$, w_i is the i-th element of $\boldsymbol{w}$ (importance of i-th feature)

$$= \sum_{i=1}^{m} w_i x_i + b$$

$$= (w_1, w_2, \dots, w_n) \begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{pmatrix} + b$$

$$= (w_1, w_2, \dots, w_n, b) \begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_n \\ 1 \end{pmatrix}$$

$$\rightarrow \quad \tilde{y} = \bar{\boldsymbol{w}}^T \bar{\boldsymbol{x}}$$

For the rest of this module, we use $\boldsymbol{w}$ and $\boldsymbol{x}$ for $\bar{\boldsymbol{w}}$ and $\bar{\boldsymbol{x}}$ respectively unless there is a special definition of $\boldsymbol{w}$ and $\boldsymbol{x}$.

Slide credit: Wang Wei

Optimization

- Compute the gradient of the loss with respect to (**w.r.t**) **w**
  - Consider a single instance

$$J(\boldsymbol{w}) = L \quad (\boldsymbol{x}, y|\boldsymbol{w}) = \frac{1}{2}\left|\left|\, \tilde{y} - y\right|\right|^2 = \frac{1}{2}\left(\boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x} - y\right)^2$$

$$\frac{\partial J(\boldsymbol{w})}{\partial \boldsymbol{w}} = ?$$

Slide credit: Wang Wei

Gradient of vector and matrix (denominator layout)

- Vectors
  - By default, is a column vector
  - Denoted as $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}$

$$\frac{\partial y}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y}{\partial x_1} \\ \frac{\partial y}{\partial x_2} \\ \vdots \\ \frac{\partial y}{\partial x_n} \end{bmatrix}.$$

$$\frac{\partial \mathbf{y}}{\partial x} = \begin{bmatrix} \frac{\partial y_1}{\partial x} & \frac{\partial y_2}{\partial x} & \cdots & \frac{\partial y_m}{\partial x} \end{bmatrix}$$

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_2}{\partial x_1} & \cdots & \frac{\partial y_m}{\partial x_1} \\ \frac{\partial y_1}{\partial x_2} & \frac{\partial y_2}{\partial x_2} & \cdots & \frac{\partial y_m}{\partial x_2} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y_1}{\partial x_n} & \frac{\partial y_2}{\partial x_n} & \cdots & \frac{\partial y_m}{\partial x_n} \end{bmatrix}.$$

Denominator layout: the result shape is (n, m)

Slide credit: Wang Wei

Gradient of vector and matrix

(denominator layout)

- Matrix
  - Denoted as **X, Y, Z**

$$\frac{\partial \boldsymbol{Y}}{\partial x} = \begin{bmatrix} \frac{\partial y_{11}}{\partial x} & \frac{\partial y_{21}}{\partial x} & \cdots & \frac{\partial y_{m1}}{\partial x} \\ \frac{\partial y_{12}}{\partial x} & \frac{\partial y_{22}}{\partial x} & \cdots & \frac{\partial y_{m2}}{\partial x} \\ \vdots & & \ddots & \vdots \\ \frac{\partial y_{1n}}{\partial x} & \frac{\partial y_{2n}}{\partial x} & \cdots & \frac{\partial y_{mn}}{\partial x} \end{bmatrix}$$

$$\frac{\partial y}{\partial \mathbf{X}} = \begin{bmatrix} \frac{\partial y}{\partial x_{11}} & \frac{\partial y}{\partial x_{12}} & \cdots & \frac{\partial y}{\partial x_{1q}} \\ \frac{\partial y}{\partial x_{21}} & \frac{\partial y}{\partial x_{22}} & \cdots & \frac{\partial y}{\partial x_{2q}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y}{\partial x_{p1}} & \frac{\partial y}{\partial x_{p2}} & \cdots & \frac{\partial y}{\partial x_{pq}} \end{bmatrix}$$

Gradient of matrix with respect to (w.r.t) matrix, matrix w.r.t vector, vector w.r.t matrix?
Too complex. Not commonly used.

Slide credit: Wang Wei

Optimization

- Compute the gradient of the loss w.r.t **w**
  - Consider a single instance

$$J(\boldsymbol{w}) = L\ \ (\boldsymbol{x}, y|\boldsymbol{w}) = \frac{1}{2}||\tilde{y} - y||^2 = \frac{1}{2}(\boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x} - y)^2$$

$$\frac{\partial J(\boldsymbol{w})}{\partial \boldsymbol{w}} = ?$$

$$z = \boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x} - y$$
$$J(\boldsymbol{w}) = \frac{1}{2}z^2$$
$$\frac{\partial J(\boldsymbol{w})}{\partial \boldsymbol{w}} = \frac{\partial J}{\partial z}\frac{\partial z}{\partial \boldsymbol{w}} = z\boldsymbol{x} = (\boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x} - y)\boldsymbol{x}$$

Shape Check!

Slide credit: Wang Wei

Vectorization

- Consider multiple examples $\{(\boldsymbol{x}^{(i)}, y^{(i)})\}$, i=1, 2, …, m
- Naïve approach of computing the gradient
  - for each instance $(\boldsymbol{x}^{(i)}, y^{(i)})$
    - Accumulate the loss $L(\boldsymbol{x}^{(i)}, y^{(i)})$ into J
  - Average J over m
  - for each instance $(\boldsymbol{x}^{(i)}, y^{(i)})$
    - Accumulate the gradient of $\frac{\partial L(\boldsymbol{x}^{(i)}, y^{(i)})}{\partial \boldsymbol{w}}$
- Not as fast as matrix operations

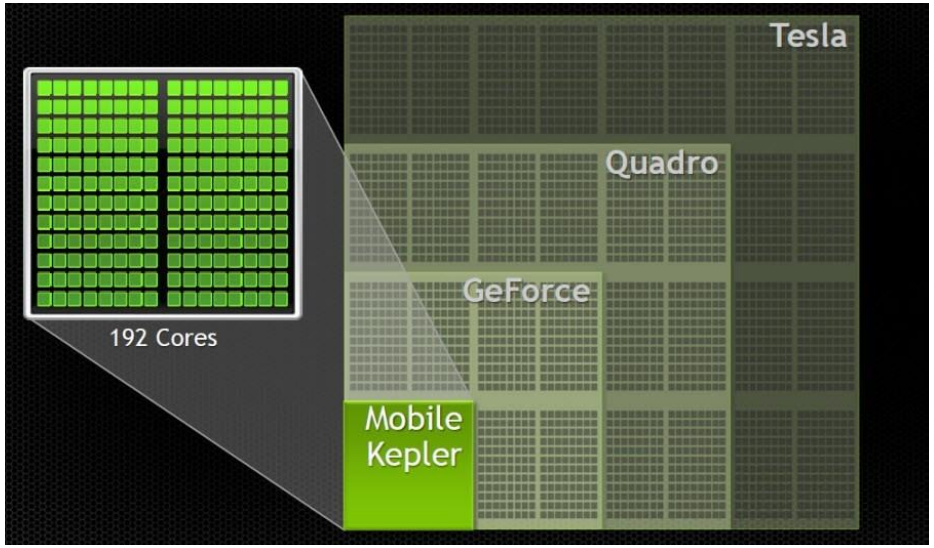


Image source: https://www.ubergizmo.com/what-is/gpu-graphics-processing-unit/

Slide credit: Wang Wei

Vectorization

- Consider multiple examples $\{(\boldsymbol{x}^{(i)}, y^{(i)})\}$, i=1, 2, …, m
- Put each instance (the feature vector) into one row of a matrix
  - For fast memory access as Numpy stores data in row-major format
- Put each target into one element of a column vector

$$\boldsymbol{X} = \begin{pmatrix} \boldsymbol{x}^{(\boldsymbol{1})^T} \\ \boldsymbol{x}^{(\boldsymbol{2})^T} \\ \dots \\ \boldsymbol{x}^{(\boldsymbol{m})^T} \end{pmatrix} \qquad \boldsymbol{y} = \begin{pmatrix} y^{(1)} \\ y^{(2)} \\ \dots \\ y^{(m)} \end{pmatrix} \qquad \tilde{\boldsymbol{y}} = \boldsymbol{X}\boldsymbol{w}$$

$$J(\boldsymbol{w}) = ?$$

Slide credit: Wang Wei

Vectorization

- Consider multiple examples $\{(\boldsymbol{x}^{(i)}, y^{(i)})\}$, i=1, 2, …, m
- Put each instance (the feature vector) into one row of a matrix
  - For fast memory access as Numpy stores data in row-major format
- Put each target into one element of a column vector

$$\boldsymbol{X} = \begin{pmatrix} \boldsymbol{x}^{(\mathbf{1})^{T}} \\ \boldsymbol{x}^{(\mathbf{2})^{T}} \\ \dots \\ \boldsymbol{x}^{(\boldsymbol{m})^{T}} \end{pmatrix} \qquad \boldsymbol{y} = \begin{pmatrix} y^{(1)} \\ y^{(2)} \\ \dots \\ y^{(m)} \end{pmatrix} \qquad \tilde{\boldsymbol{y}} = \boldsymbol{X}\boldsymbol{w}$$

$$J(\boldsymbol{w}) = \frac{1}{2m}\left|\left|\tilde{\boldsymbol{y}} - y\right|\right|^{2} = \frac{1}{2m}\left(\tilde{\boldsymbol{y}} - \boldsymbol{y}\right)^{\mathrm{T}}\left(\tilde{\boldsymbol{y}} - \boldsymbol{y}\right) \qquad \boldsymbol{u} = \tilde{\boldsymbol{y}} - \boldsymbol{y}$$

$$\frac{\partial J(\boldsymbol{w})}{\partial \boldsymbol{w}} = ?$$

Slide credit: Wang Wei

Vectorization

- Consider multiple examples $\{(\boldsymbol{x}^{(i)}, y^{(i)})\}$, i=1, 2, …, m
- Put each instance (the feature vector) into one row of a matrix
  - For fast memory access as numpy stores data in row-major format
- Put each target into one element of a column vector

$$X = \begin{pmatrix} \boldsymbol{x}^{(1)^T} \\ \boldsymbol{x}^{(2)^T} \\ \dots \\ \boldsymbol{x}^{(m)^T} \end{pmatrix} \qquad \boldsymbol{y} = \begin{pmatrix} y^{(1)} \\ y^{(2)} \\ \dots \\ y^{(m)} \end{pmatrix} \qquad \tilde{\boldsymbol{y}} = X\boldsymbol{w}$$

$$J(\boldsymbol{w}) = \frac{1}{2m}\left|\left|\tilde{\boldsymbol{y}} - y\right|\right|^2 = \frac{1}{2m}(\tilde{\boldsymbol{y}} - \boldsymbol{y})^{\mathrm{T}}(\tilde{\boldsymbol{y}} - \boldsymbol{y}) \qquad \boldsymbol{u} = \tilde{\boldsymbol{y}} - \boldsymbol{y}$$

$$\frac{\partial J(\boldsymbol{w})}{\partial \boldsymbol{w}} = \frac{1}{2m}\left(\frac{\partial \boldsymbol{u}}{\partial \boldsymbol{w}}\boldsymbol{u} + \frac{\partial \boldsymbol{u}}{\partial \boldsymbol{w}}\boldsymbol{u}\right) = \frac{1}{m}\boldsymbol{X}^{\boldsymbol{T}}\boldsymbol{u} = \frac{1}{m}\boldsymbol{X}^{\boldsymbol{T}}(\tilde{\boldsymbol{y}} - \boldsymbol{y})$$

Shape Check!

Slide credit: Wang Wei

Training by gradient descent

- Gradient descent algorithm for optimization
- initialize $\boldsymbol{w}$ randomly
- Repeat
  - Compute the objective $J(\boldsymbol{w})$ over all training instances
  - Compute $\frac{\partial J}{\partial \boldsymbol{w}}$
  - Update $\mathbf{w} = \mathbf{w} - \alpha \frac{\partial J}{\partial \boldsymbol{w}}$

Slide credit: Wang Wei

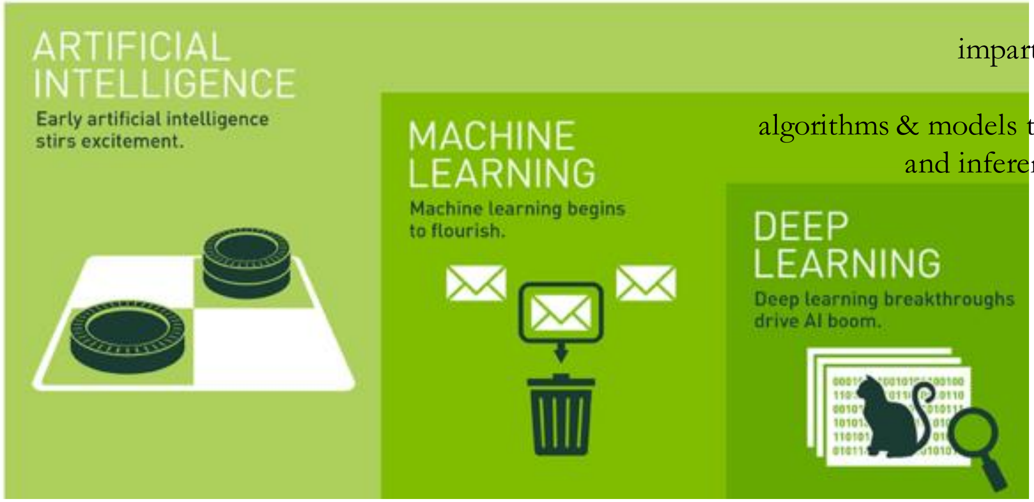
Notation summary

- Scalar x
- Vector $\boldsymbol{x} \in R^{n\times 1}$
  - i-th element of a vector, x_i
  - i-th example $\boldsymbol{x}^{(i)}$
- Matrix $\boldsymbol{X}$
  - i-th row and j-th column X_{ij}
- If we choose denominator layout for $\frac{\boldsymbol{\partial y}}{\boldsymbol{\partial x}}$ we should lay out the gradient $\frac{\boldsymbol{\partial y}}{\boldsymbol{\partial x}}$ as a column vector, and $\frac{\boldsymbol{\partial y}}{\boldsymbol{\partial} x}$ as a row vector.

Slide credit: Wang Wei

Summary

- AI vs. ML vs. Deep Learning



imparting cognitive abilities to machines

algorithms & models to perform tasks based on patterns and inference instead of specific instructions

branch of machine learning using deep neural networks

- Taking gradients of vectors & matrices
- Univariate & multivariate linear regression
  - Updating parameters via gradient descent

References

- [1] Goodfellow Ian, Bengio Yoshua, Courville Aaron. Deep learning. MIT Press. http://www.deeplearningbook.org
- [2] Haohan Wang, Bhiksha Raj. On the Origin of Deep Learning. 2017 https://arxiv.org/abs/1702.07800
- [3] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng. "Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations." In ICML 2009
- http://neuralnetworksanddeeplearning.com/chap1.html
- https://www.analyticsvidhya.com/blog/2017/06/a-comprehensive-guide-for-linear-ridge-and-lasso-regression/
- https://medium.com/meta-design-ideas/math-stats-and-nlp-for-machine-learning-as-fast-as-possible-915ef47ced5f

Slide credit: Wang Wei

Homework

TAs will let you know soon

Next Lecture

(Shallow) Neural Networks