- This session is recorded.
- Mute your mic.
- You can leave messages in the chat panel to ask questions.
- When I check your progress, please click 

if you are done.
- We will start at 6:35.

CS5242 Neural Networks and Deep Learning

Lecture 01: Introduction and Linear Regression

Wei WANG

TA: FU Di, LIANG Yuxuan, FuYujian

cs5242@comp.nus.edu.sg

change log:
V1: slide 41, the superscript should be n

NUS National University of Singapore | School of Computing

Agenda

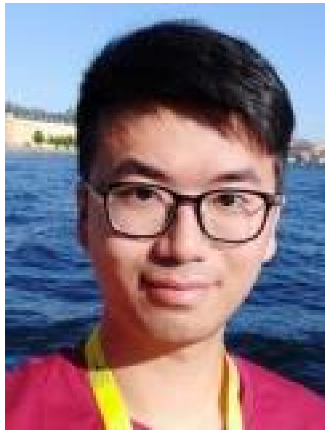
- Introduction of this module
  - Logistics
  - History
- Linear regression
  - Univariate
  - Multivariate

Instructor

- WANG Wei
  - COM2-04-09
  - wangwei@comp.nus.edu.sg

- Research
  - Machine learning system optimization
  - Multi-modal data analysis/applications

Teaching Assistants



LIANG Yuxuan

e0427783@u.nus.edu

Quiz

FU Di

e0409760@u.nus.edu

Assignment

FU Yujian

e0427770@u.nus.edu

Final project

Your background?

LumiNUS Poll

Pre-requisite

Course

- 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, Caffe, SINGA
- Jupyter notebook or Google colab

Grading policy

Weightage:

- Assignment 30 points
  - Two assignments
  - 15% off per day late (17:01 is the start of one day); 0 score if you submit it 7 days after the deadline
- Online quiz 30 points
  - 3 quizzes in total
  - You need a camera (from your laptop, mobile phone or external camera)
- Project 40 points
  - Kaggle competition
  - Group size <=4

Collaboration

- Every assignment is an individual task
- The project is a group-based task
- Avoid academic offence (cheating, plagiarism including copying code from the internet, e.g., github)

Contacts

- LumiNUS forum
  - For all issues
- Email: cs5242@comp.nus.edu.sg
  - For private/personal issues
- Consultation (make appointments on LumiNUS)
  - Wei WANG, wangwei@comp.nus.edu.sg
  - Wed, 11:00-12:00

GPU machines

- SoC GPU machines
  - https://dochub.comp.nus.edu.sg/cf/guides/compute-cluster/hardware
- Google Cloud Platform
  - Some free credit for new register
  - GPU is expensive
- Amazon EC2 (g2.8xlarge)
  - Some free credit for students
  - GPU is expensive
- NSCC (National Super-Computing Center)
  - Free for NUS students
  - The libraries are sometimes outdated
  - Jobs will be submitted into a queue for executing
- **Important notes if you use cloud platforms:**
  - STOP/TERMINATE the instance immediately after your program terminates
  - Check the usage status frequently to avoid high outstanding bills
  - Amazon/Google may charge you for additional storage volume

Syllabus & Schedule

- Learn various neural network models
  - From shallow to deep neural networks
  - Including the model structure, training algorithms, tricks and applications
  - Covering the principles, coding practices and a bit theory
- Check LumiNUS for the latest plan

Intended learning outcomes (my expectation)

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

Your expectation?

I want to get an easy boost to my GPA.

This course may not be easy

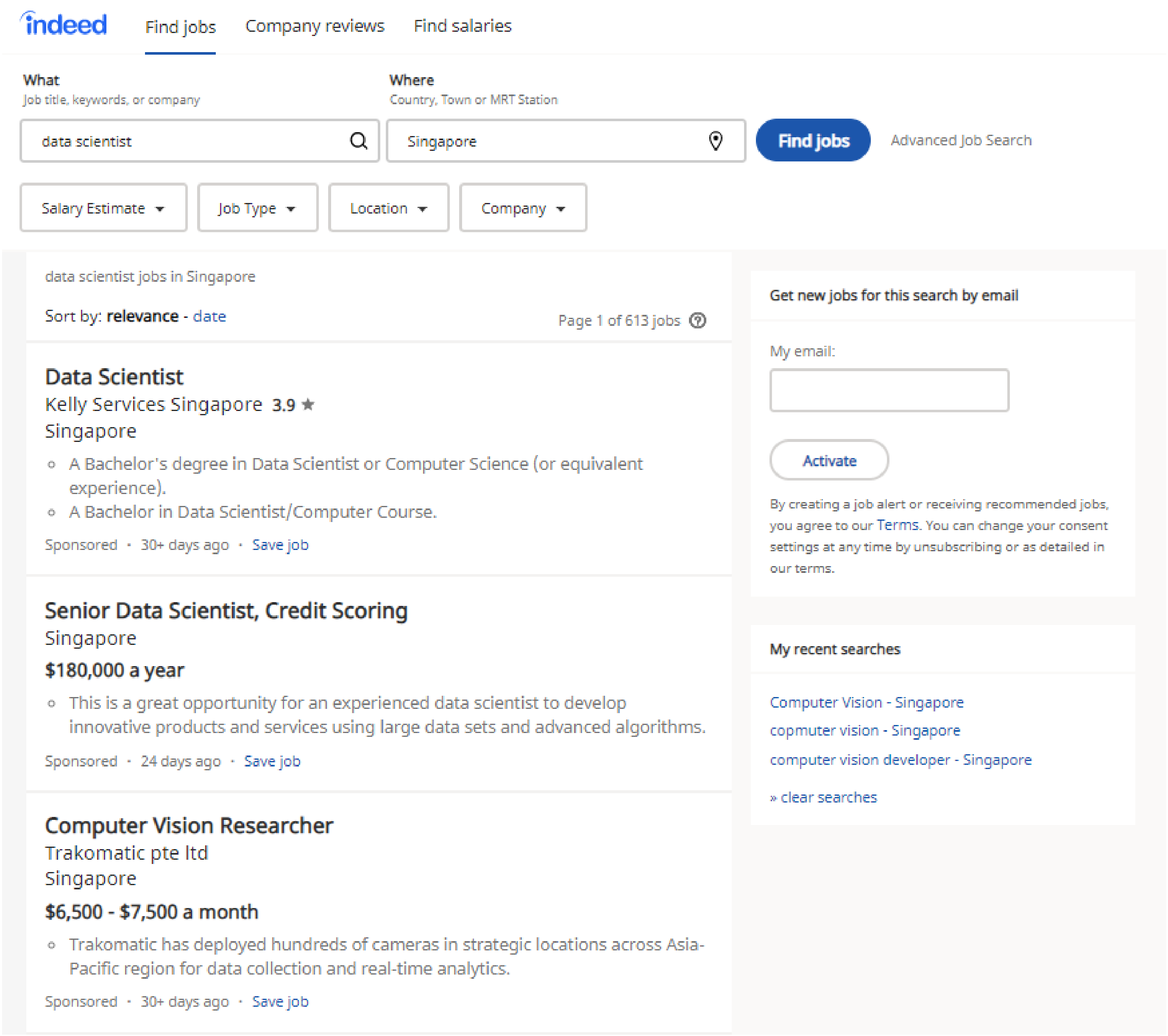
I want to earn the big bucks as a data scientist / ML engineer at Google / Amazon / Facebook.

This course provides the basics; but is not enough.

check out:

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

Slide credit: Angela Yao



Page 1 of 613

$180,000 a year

Slide credit: Angela Yao

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

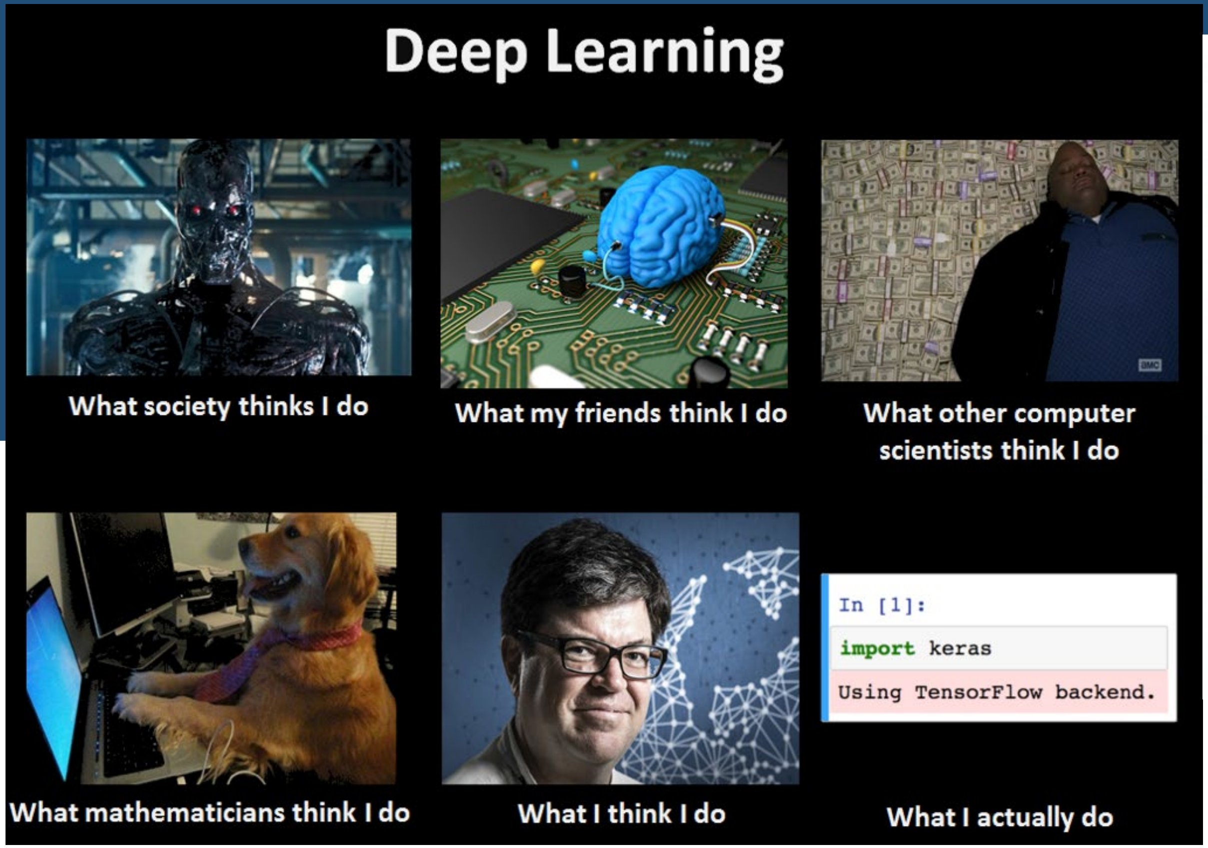
- Fei-Fei Li, Justin Johnson, Serena Yeung
- http://cs231n.stanford.edu/

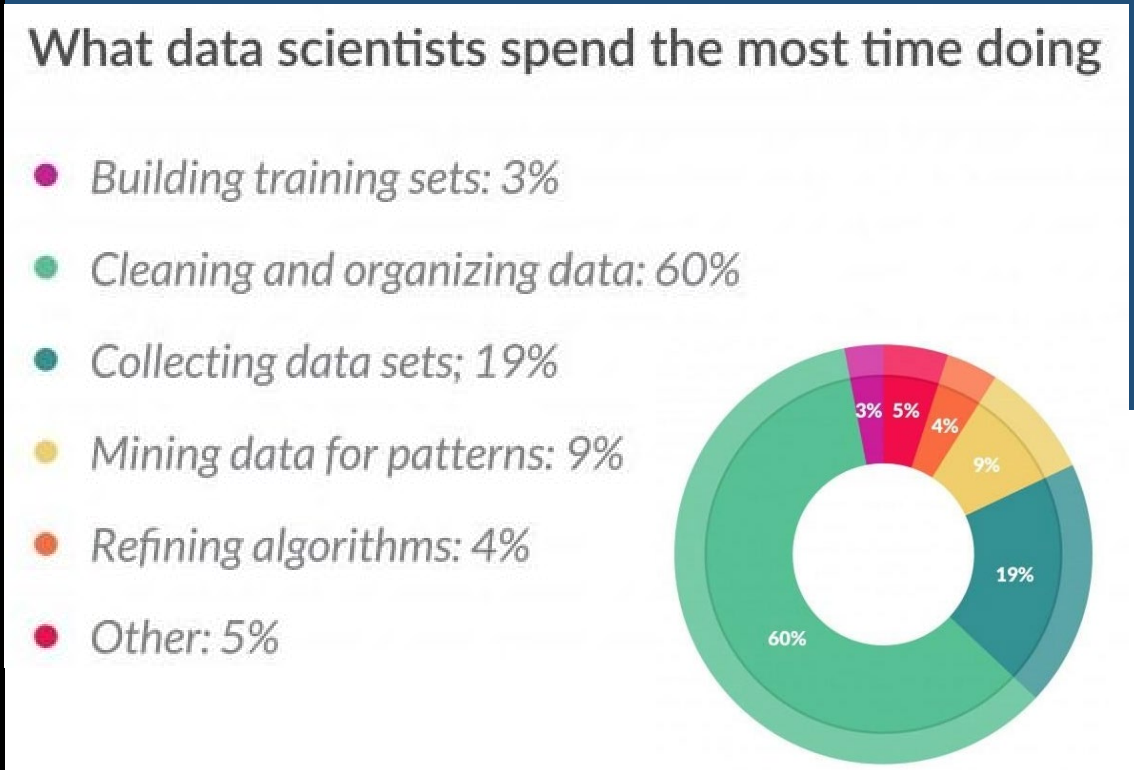
CS224d: Deep Learning for Natural Language Processing

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

Slide credit: Angela Yao

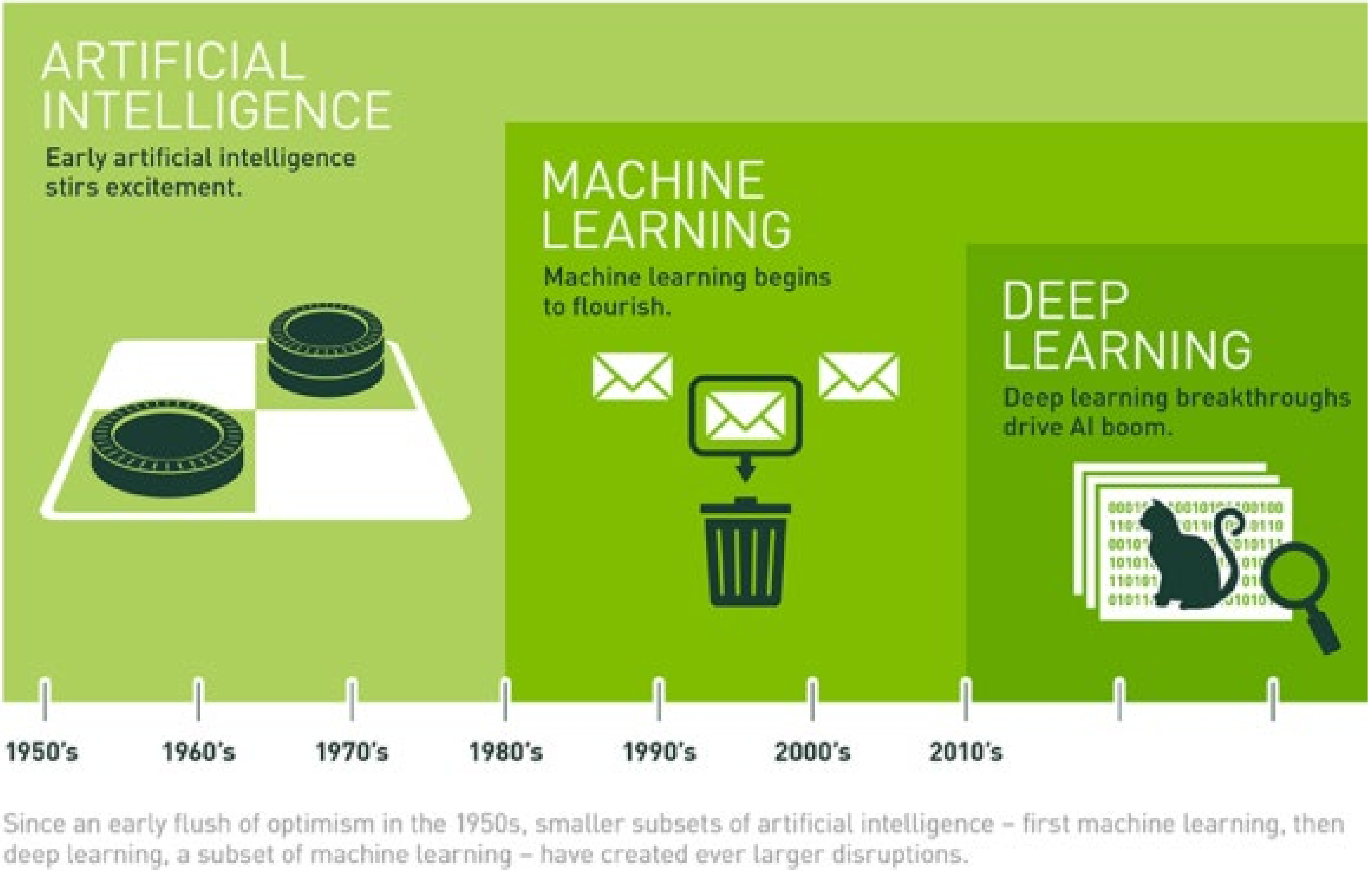
AI is cool and I want to build the next Skynet.





Slide credit: Angela Yao

Terminology Overload: AI vs. ML vs. DL



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

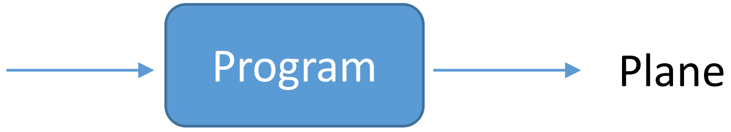
AI

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

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

Machine Learning

- 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 with rules (like bubble sorting) to do
    - Image recognition
    - Speech recognition



Neural networks and Deep Learning

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

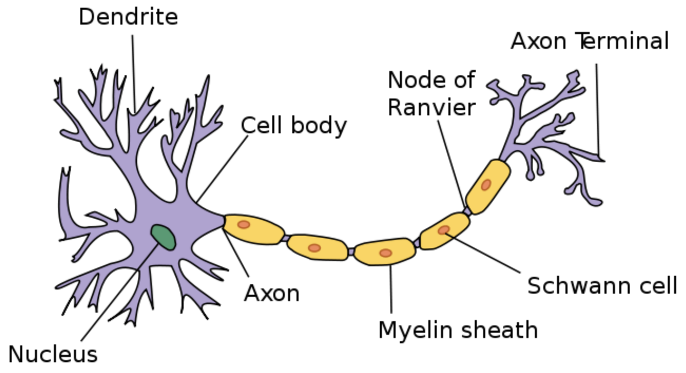
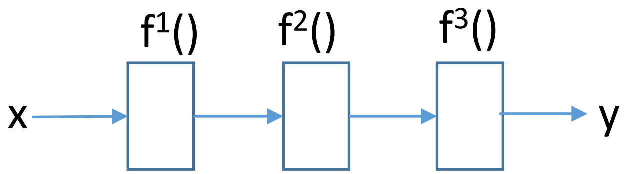


Image source: http://pediaa.com/difference-between-nerve-and-neuron/

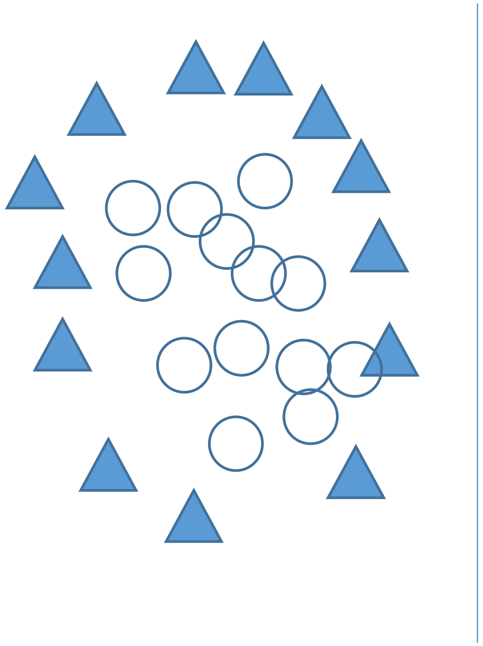


Neural nets/perceptrons are **loosely** inspired by biology. But they certainly are **not** a model of how the brain works, or even how neurons work.

Slide credit: Angela Yao

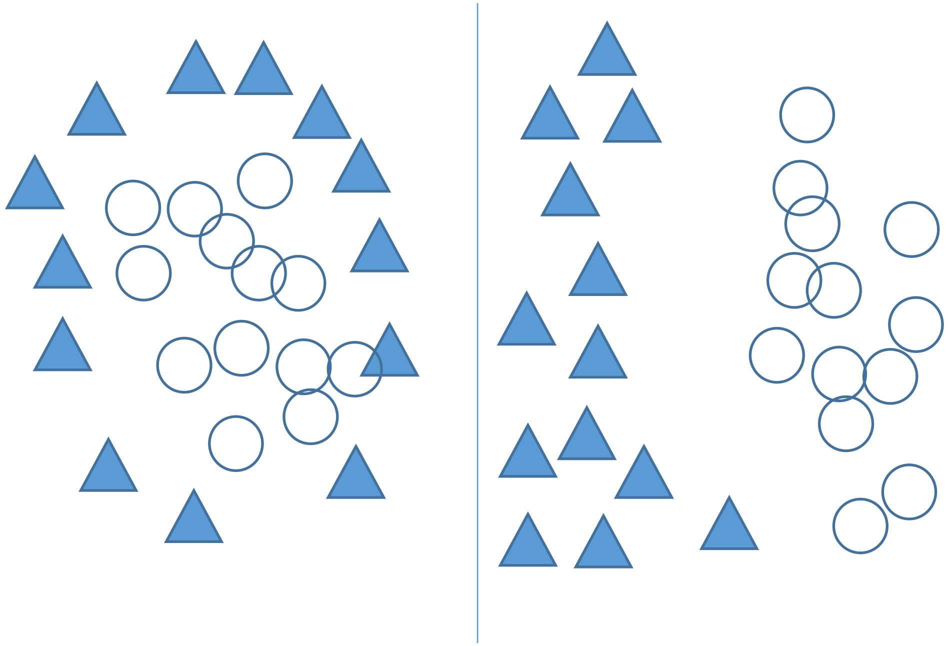
Neural networks and Deep Learning

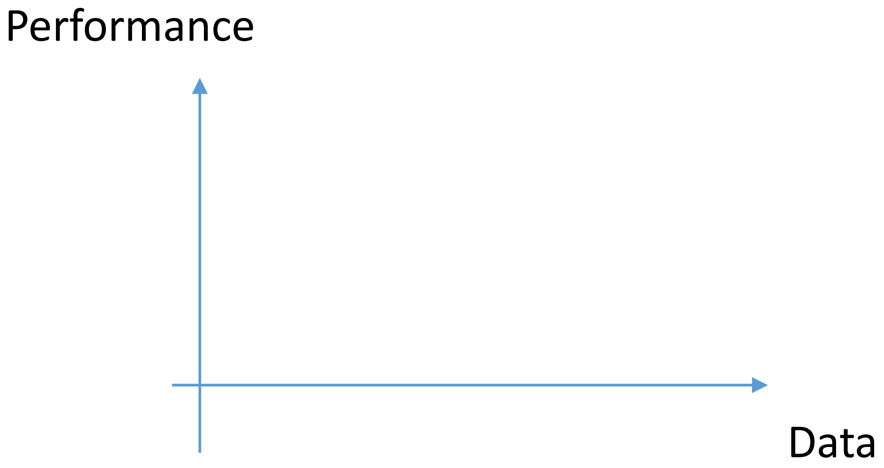
- Feature learning (transformation)



Neural networks and Deep Learning

- Feature learning (transformation)

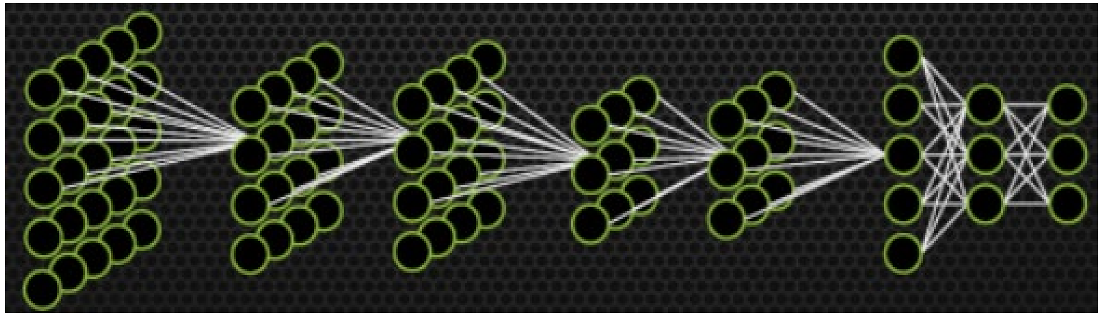




[Deep learning is not a killer for everything]()

Neural networks (NN)

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



Source from [3]

History [2]

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

1958
First perceptron, Frank Rosenblatt

1974
Backpropagation, Paul Werbos

1980
Neocogitron (inspired CNN), Kunihiko Fukushima

1986
Recurrent Neural Network (RNN), Michael I. Jordan

1990
LeNet, Yann LeCun

1998
Hochreiter & Schmidhube, LSTM

2006
Deep Belief Networks, Geoffrey Hinton

2012
AlexNet, Alex Krizhevsky & Geoffrey Hinton

2014
GoogleNet & VGG

2014
Generative adversarial network, Ian Goodfellow

2015
ResNet, Kaimin He

2015
AlphaGo, DeepMind

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

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

Univariate Linear Regression

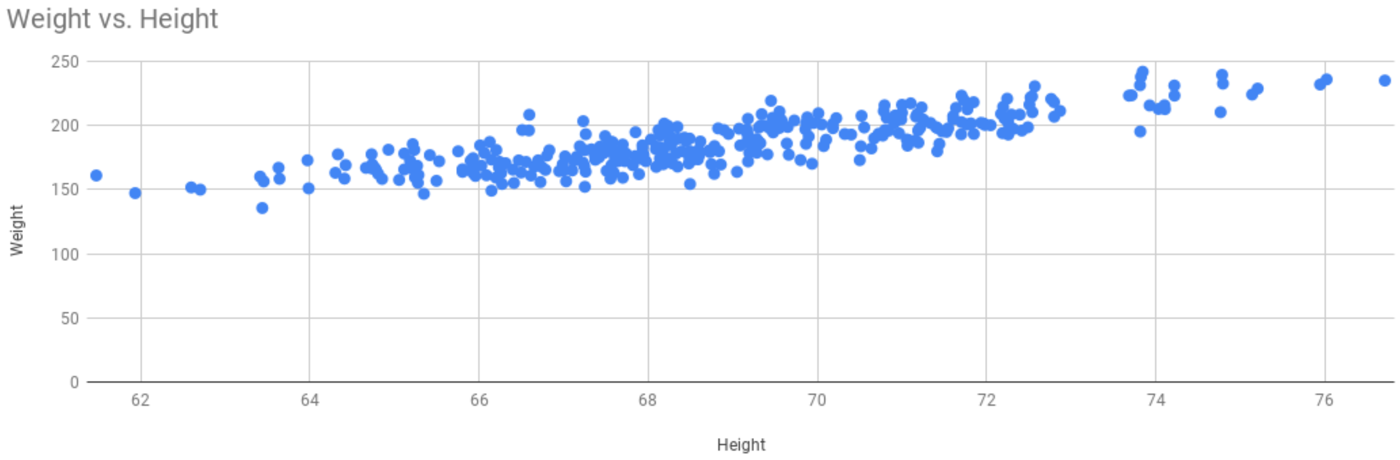
Weight vs Height

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

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**

Training

- Learn w and b to fit the training data $S_{train} = \left\{\left(x^{(i)}, y^{(i)}\right)\right\}, i = 1 \ \ldots m$
  - 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
  - $J(w, b) = \frac{1}{m}\sum_{i=1}^{m} L\left(x^{(i)}, y^{(i)} \middle| w, b\right)$

Optimization/training

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

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

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

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

Optimization/training

1. Fix b and learn w

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

$$\rightarrow \min_w c_1 w^2 + c_2 w + c_3$$

2. Fix w and learn b

Gradient for univariate simple functions

Gradient: recall high school calculus and how to take derivatives

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

- $f(x) = 3, \frac{\partial f(x)}{\partial x} = 0$

- $f(x) = 3x, \frac{\partial f(x)}{\partial x} = 3$

- $f(x) = x^2, \frac{\partial f(x)}{\partial x} = 2x$

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

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

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

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

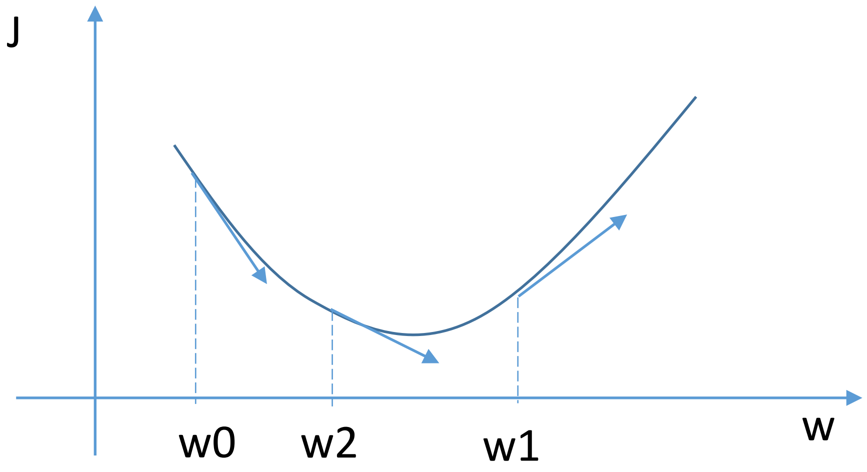
- $f(x) = g(u), u = h(x)$
  - $f(x) = \ln(x + x^2)$
- $f'(x) = g'(u)h'(x)$

HW1: solve for the derivatives / gradients of these functions.

Training by gradient descent

$J = c_1 w^2 + c_2 w + c_3$ 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
$w1 = w0 - \alpha \frac{\partial J}{\partial w}|_{w=w0}$

Compute $\frac{\partial J}{\partial w}|_{w=w1}$, positive;
Move w from w1 to the left by
$w2 = w1 - \alpha \frac{\partial J}{\partial w}|_{w=w1}$

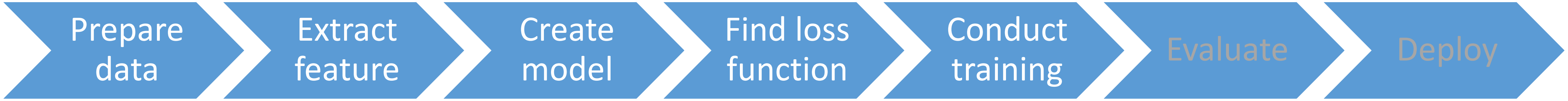
Compute $\frac{\partial J}{\partial w}|_{w=w2}$, negative
Move w from w2 to the right by
$w3 = w2 - \alpha \frac{\partial J}{\partial w}|_{w=w2}$

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 $\mathrm{w} = \mathrm{w} - \alpha \frac{\partial J}{\partial w}$

Update w, b repeatedly...
What if there are multiple parameters, e.g., multiple variables?

Machine learning pipeline



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

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

Model

- Map from input to output

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

  $= \sum_{i=1}^{n} w_i x_i + b$ The summation is over n elements, where n is the number of features per instance.

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

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

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

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

Optimization

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

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

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

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)

Gradient of vector and matrix (denominator layout)

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

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

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

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.

Gradient table (denominator layout)

Shape of $\frac{\partial y}{\partial x}$	Scalar y	Vector **y** (m ,1)	Matrix **Y** (m, n)
Scalar x	?	?	?
Vector **x** (n ,1)	(n, 1)	?	
Matrix **X** (p, q)	?		

HW2: fill in the table with the matrix dimensions of the resulting gradients.

Gradient table: vector by vector (denominator layout)

HW3: fill in the gradient table with the resulting gradients

$\boldsymbol{y} =$	$\boldsymbol{a}$	$\boldsymbol{x}$	$\boldsymbol{A}\boldsymbol{x}$	$\boldsymbol{x}^{\boldsymbol{T}}\boldsymbol{A}$
$\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}} =$	?	?	$\boldsymbol{A}^T$	?

$\boldsymbol{y} =$	$a\boldsymbol{u}$ $\boldsymbol{u} = \boldsymbol{u}(\boldsymbol{x})$	$v\boldsymbol{u}$ $v = v(\boldsymbol{x}), \boldsymbol{u} = \boldsymbol{u}(\boldsymbol{x})$	$\boldsymbol{v} + \boldsymbol{u}$ $\boldsymbol{v} = \boldsymbol{v}(\boldsymbol{x}), \boldsymbol{u} = \boldsymbol{u}(\boldsymbol{x})$	$\boldsymbol{A}\boldsymbol{u}$ $\boldsymbol{u} = \boldsymbol{u}(\boldsymbol{x})$	$g(\boldsymbol{u})$ $\boldsymbol{u} = \boldsymbol{u}(\boldsymbol{x})$
$\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}} =$	?	$v\frac{\partial \boldsymbol{u}}{\partial \boldsymbol{x}} + \frac{\partial v}{\partial \boldsymbol{x}}\boldsymbol{u}^T$	?	?	?

$y =$	a	$\boldsymbol{u}^{\boldsymbol{T}}\boldsymbol{v}$ $\boldsymbol{v} = \boldsymbol{v}(\boldsymbol{x}), \boldsymbol{u} = \boldsymbol{u}(\boldsymbol{x})$	$g(u)$ $u = u(\boldsymbol{x})$	$\boldsymbol{x}^{\boldsymbol{T}}A\boldsymbol{x}$
$\frac{\partial y}{\partial \boldsymbol{x}} =$	?	?	?	?

Optimization

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

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

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

$$z = \boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x} - y$$

$$J(\boldsymbol{w}) = \frac{1}{2}z^2$$

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

Shape Check!

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

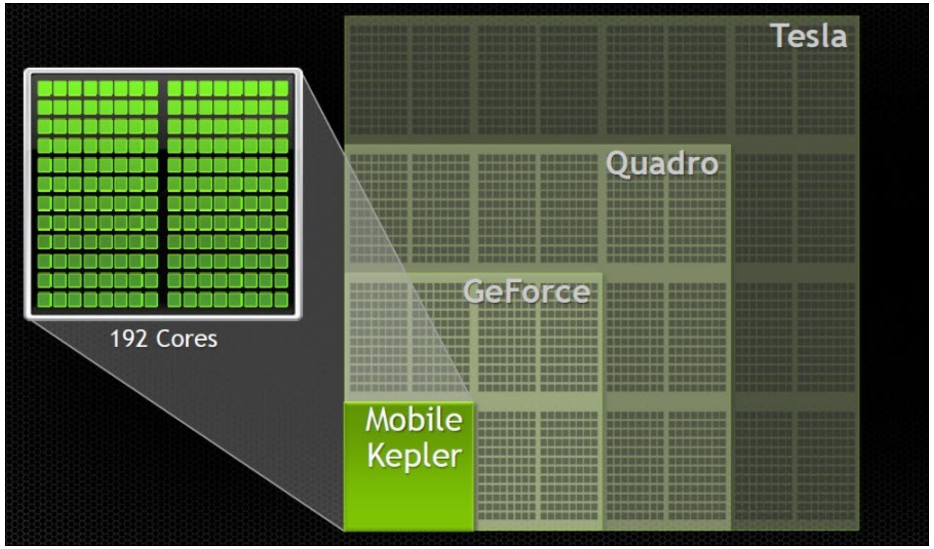


Image source: https://www.ubergizmo.com/what-is/gpu-graphics-processing-unit/

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

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

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

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

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

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

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

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

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

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

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

Shape Check!

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 $\mathbf{w} = \mathbf{w} - \alpha \frac{\partial J}{\partial \boldsymbol{w}}$

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 $\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}$ we should lay out the gradient $\frac{\partial y}{\partial \boldsymbol{x}}$ as a column vector, and $\frac{\partial \boldsymbol{y}}{\partial x}$ as a row vector.

Summary

- AI vs. ML vs. Deep Learning
- Terminologies & Notations
  - Feature, label, loss
- Univariate & multivariate linear regression
  - Solving parameters via gradient descent
  - Taking gradients of vectors & matrices

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

Homework

Theory

1. Solve for the gradients of the functions. (slide 34)
2. Solve for the dimensionality of the gradient vectors (slide 44)
3. Fill out the gradient tables (slide 45).

Practical

1. https://tinyurl.com/y26qj2ts

Next Lecture

(Shallow) Neural Networks