

# CS7150 Deep Learning

Jiaji Huang

<https://jiaji-huang.github.io>

01/13/2024

# About the lecturer

- PhD in electrical engineering from Duke University
- Thesis: statistical signal processing and machine learning
- Worked at Baidu Research on NLP and Speech
- Now at AWS, on Large Language Models

# Agenda

- About this class
- Deep Learning nowadays
- Math
- Programming

# What to learn from the class

- Basic building blocks, concepts
  - e.g., conv layer, attention layer, optimizers, overfitting
- Important applications
  - e.g., language modeling
- Recent published models
  - e.g., Llama2, Gemini

# After the class, you should be

- able to implement and train typical deep nets
- aware of some emerging trends in the fields
- able to read recent deep learning papers

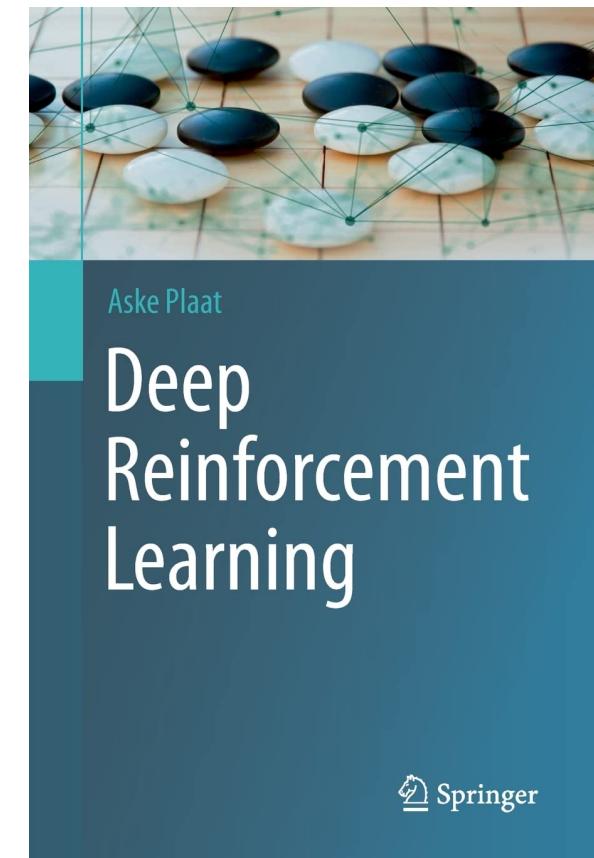
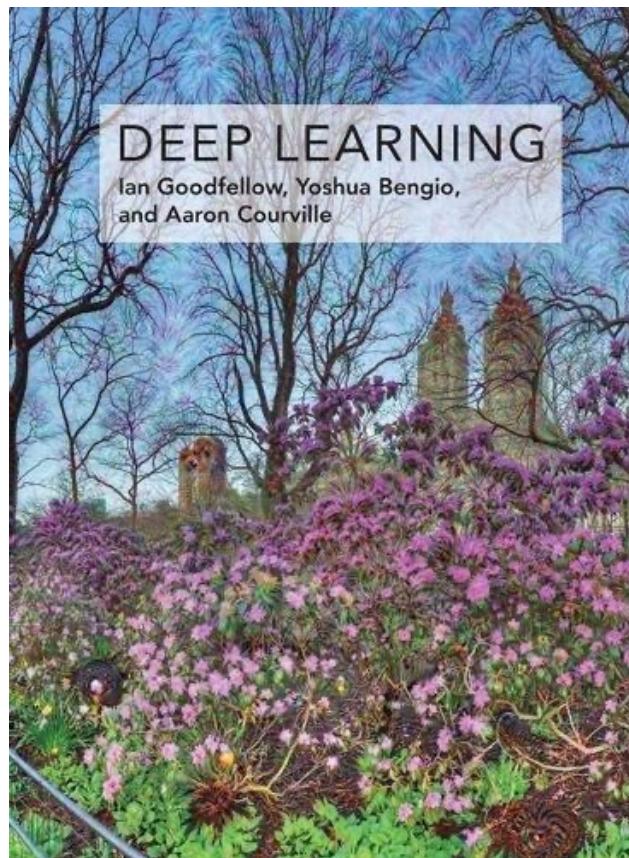
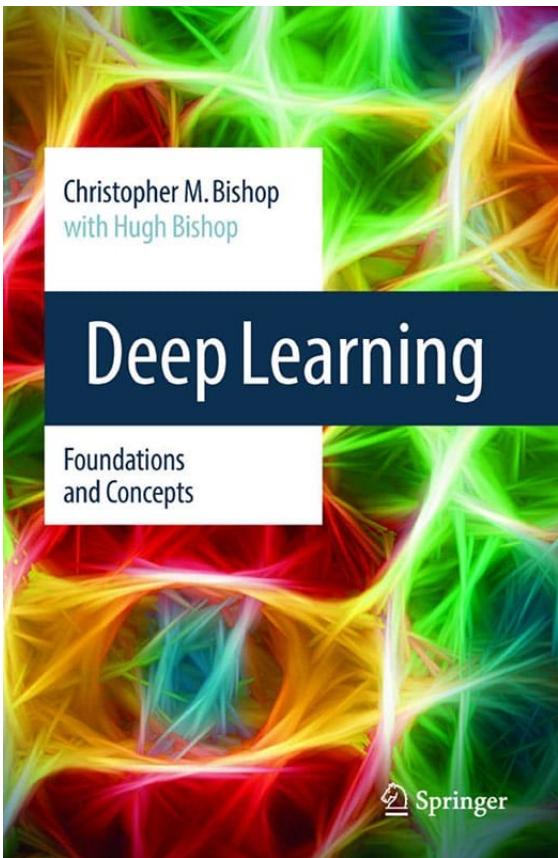
# This class is not expected to ...

- teach coding with pytorch
- be a introductory course for machine learning

# Logistics

- Classes
  - Each Saturday 9:00am-12:20pm
  - Before mid-term: focus on basics
  - After mid-term: advanced topics, 2-3 guest lectures (tentative)
- Office hour: TBD
- TA: Pratyaksh Bhalla ([bhalla.pr@northeastern.edu](mailto:bhalla.pr@northeastern.edu))

# Textbooks



# Logistics

- Grades
  - 20% homework + 20% paper presentation + 20% mid-term exam + 40% project
  - Policy for ☹
    - late submission of homework
    - Plagiarism
- Computational resources:
  - You'll need a laptop/desktop with python3
  - Khoury Cluster

# Paper Presentation

- Since class of 01/27, after lecture, 20-30min includes Q&A
- TA will generate a randomized list of presenters
- Paper selection:
  - Suggested reading materials (but only papers) in previous classes
  - paper identified by presenter
- Credits:
  - For presenter: clear presentation, good answers, driving discussion
  - For audience: raise question, involve in discussion

# How to identify interesting papers

- Subscribe to arxiv cs.AI: send an email like this

To: cs@arxiv.org

Subject: subscribe yourFirstName yourLastName

add Artificial Intelligence

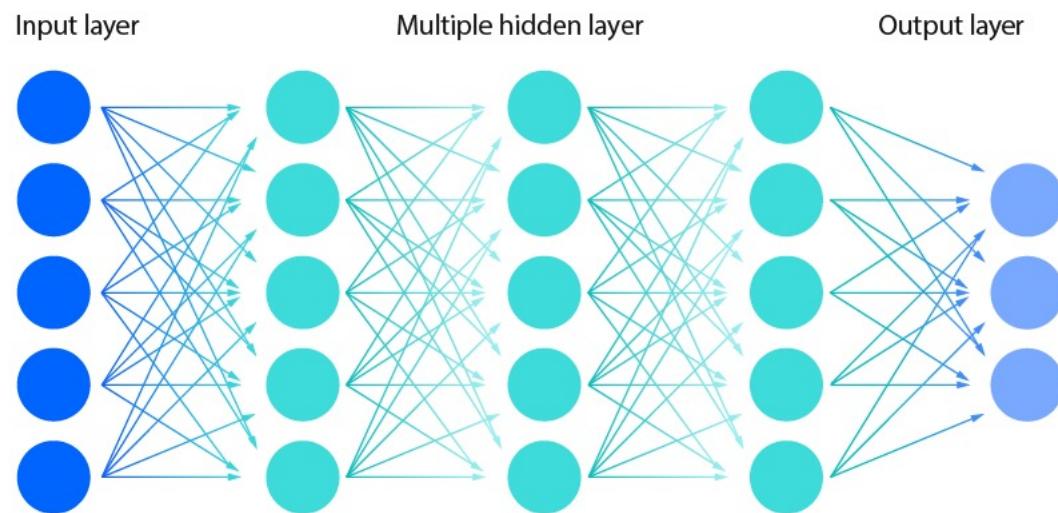
- Check recent accepted papers in conferences
  - Learning: Neurips, ICML, ICLR, AAAI, ...
  - NLP: ACL, EMLNP, NAACL, ...
  - Computer vision: ICCV, CVPR, ECCV, ...
- Media: twitter etc.

# Agenda

- About this class
- Deep Learning nowadays
- Math
- Programming

# What is Deep Learning

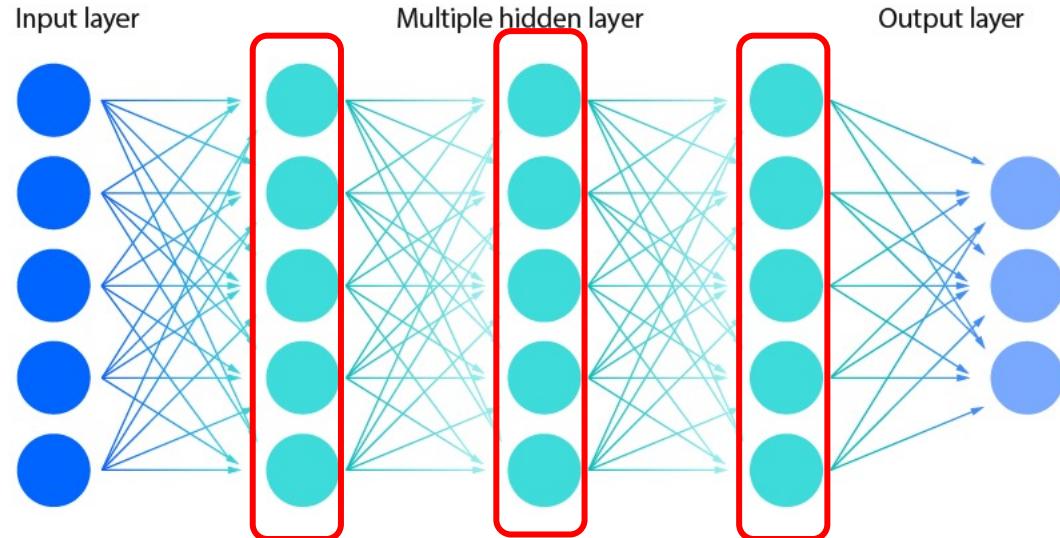
- Neural network: aka artificial neural networks (ANN)



- **Deep** neural network: many layers
- **Deep Learning**: A sub-area of machine learning that builds deep neural network to model the world

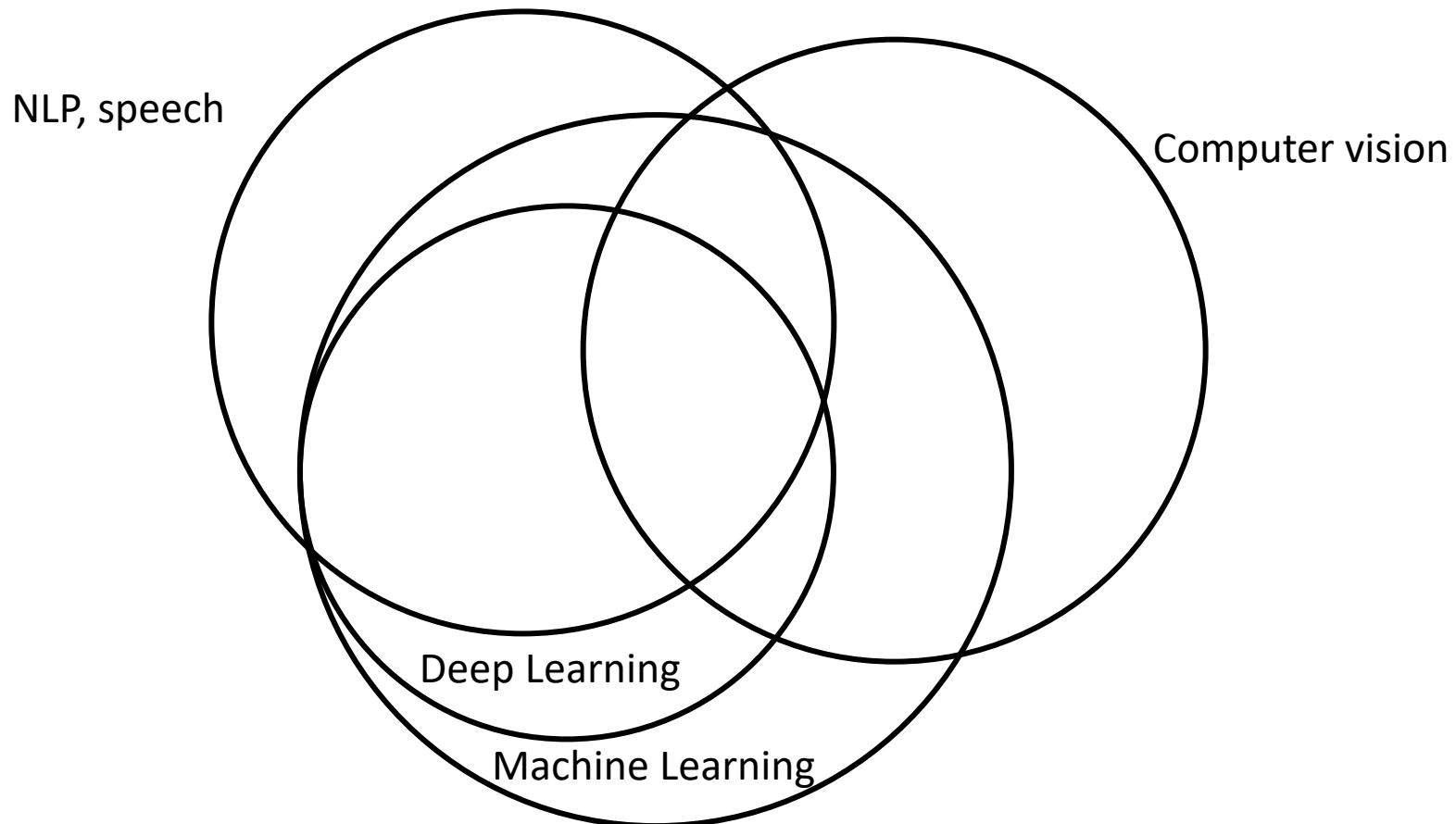
Illustration from <https://www.ibm.com/topics/neural-networks>

# A few more terminologies

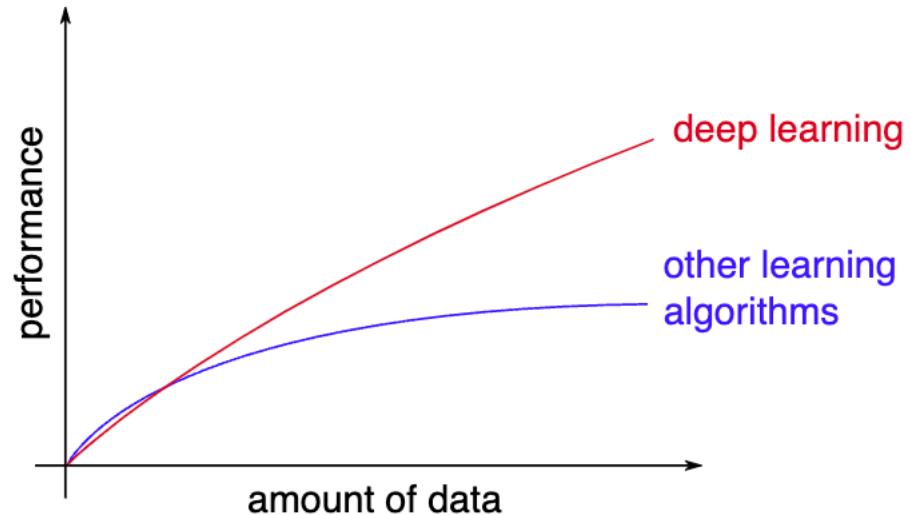


- Hidden representations/feature maps
- Learned Representation (as apposed to hand-crafted features)
- Distributed representations
- **End-to-end:** jointly learn the representation for the task

# Related areas



# Why is DL useful?



- Given enough computational power (though)

# DL and vision

- Image Classification (e.g., ImageNet, 1000 classes)
  - [Alexnet \(Krizhevsky et. al, 2012\)](#)

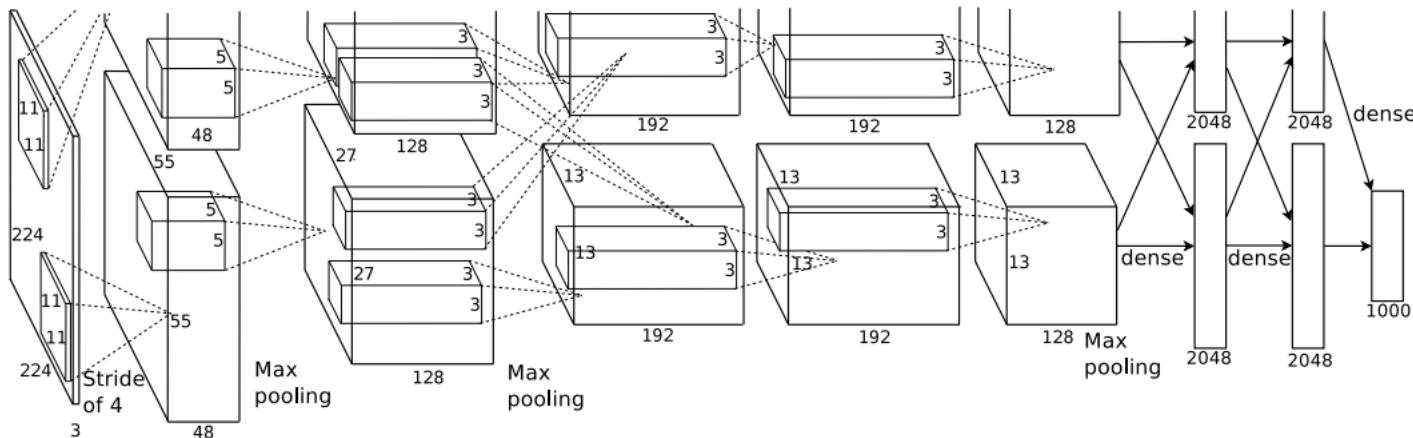


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

# DL and vision

- Beat non-DL methods by a large margin

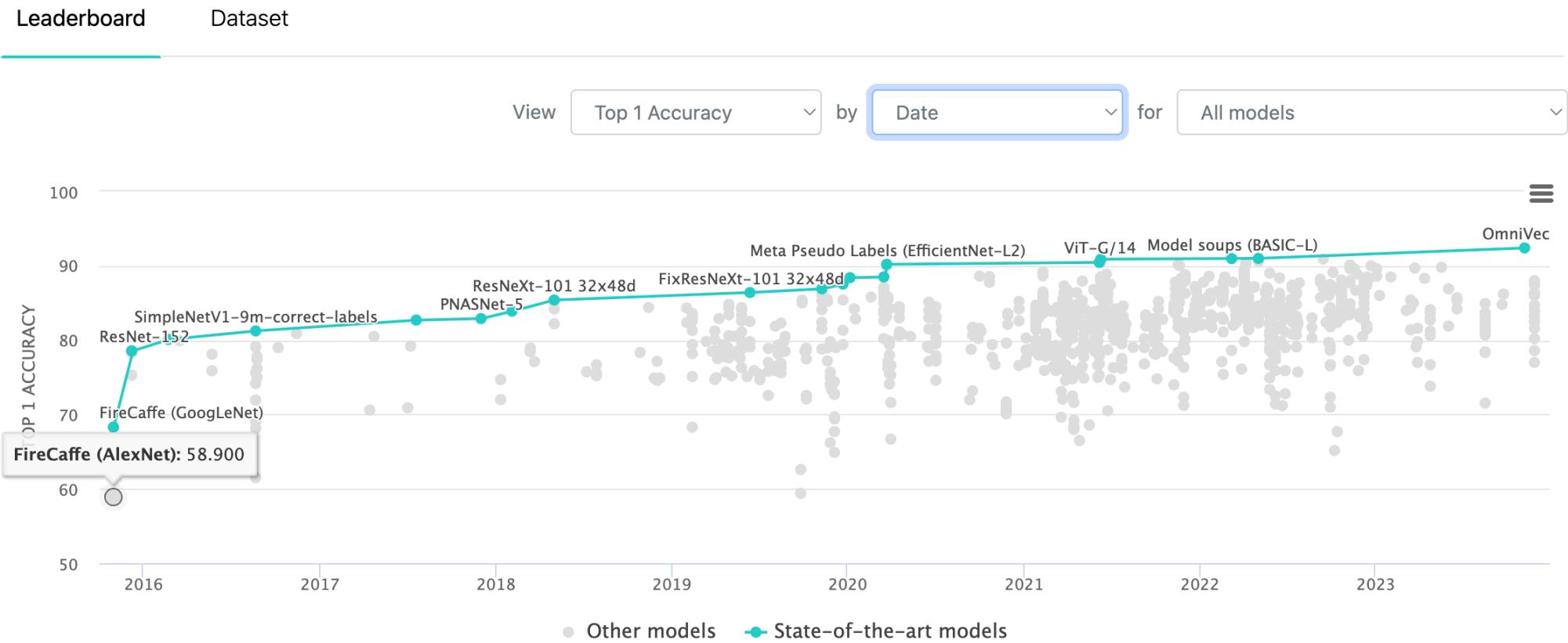
Model	Top-1	Top-5
<i>Sparse coding [2]</i>	47.1%	28.2%
<i>SIFT + FVs [24]</i>	45.7%	25.7%
CNN	<b>37.5%</b>	<b>17.0%</b>

Table 1: Comparison of results on ILSVRC-2010 test set. In *italics* are best results achieved by others.

- Limitations
  - The GPU memory was tiny (3GB), and slow
  - So they have to split the layer parameters (we may talk about this later)
  - and limit training time



# Since AlexNet ...



Trend according to [paperwithcode](#)

# DL and vision

- Object Detection
- [Fast R-CNN \(Girshick 2015\)](#)
- Track trend on [paperwithcode](#)

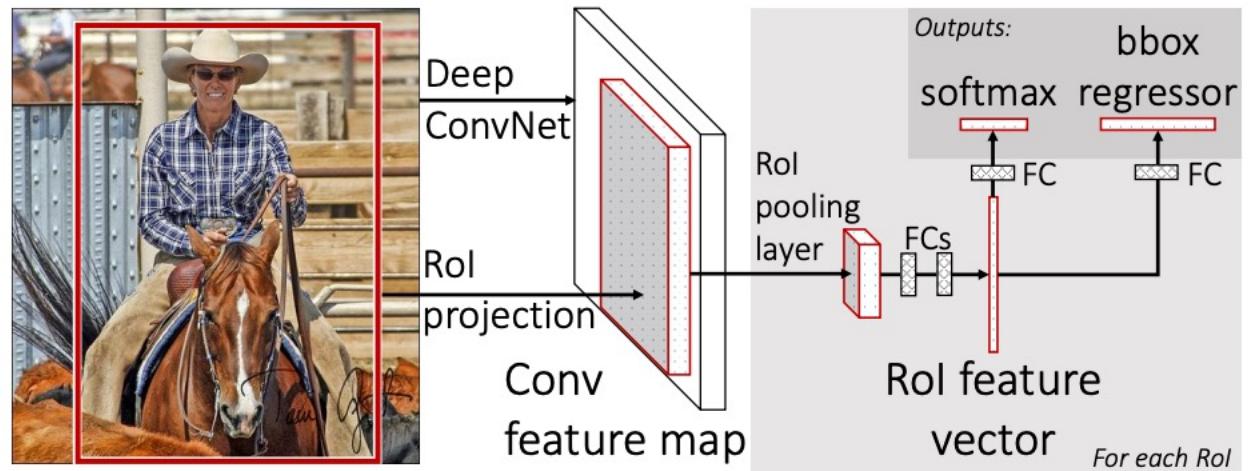
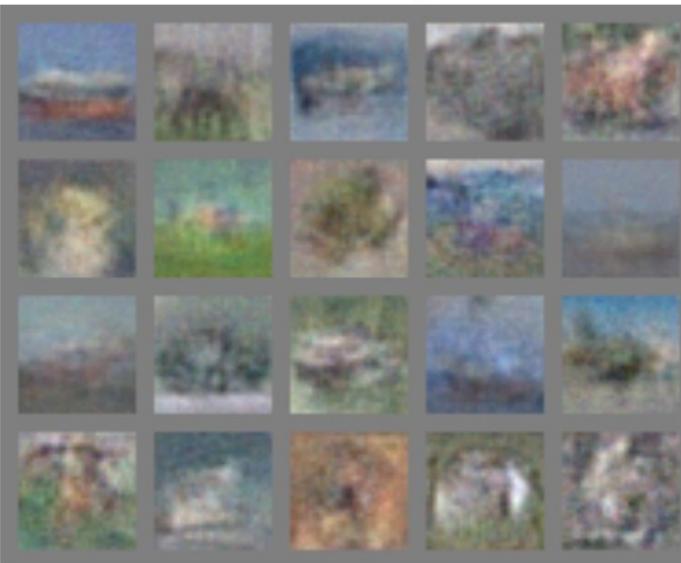


Figure 1. Fast R-CNN architecture. An input image and multiple regions of interest (ROIs) are input into a fully convolutional network. Each ROI is pooled into a fixed-size feature map and then mapped to a feature vector by fully connected layers (FCs). The network has two output vectors per ROI: softmax probabilities and per-class bounding-box regression offsets. The architecture is trained end-to-end with a multi-task loss.

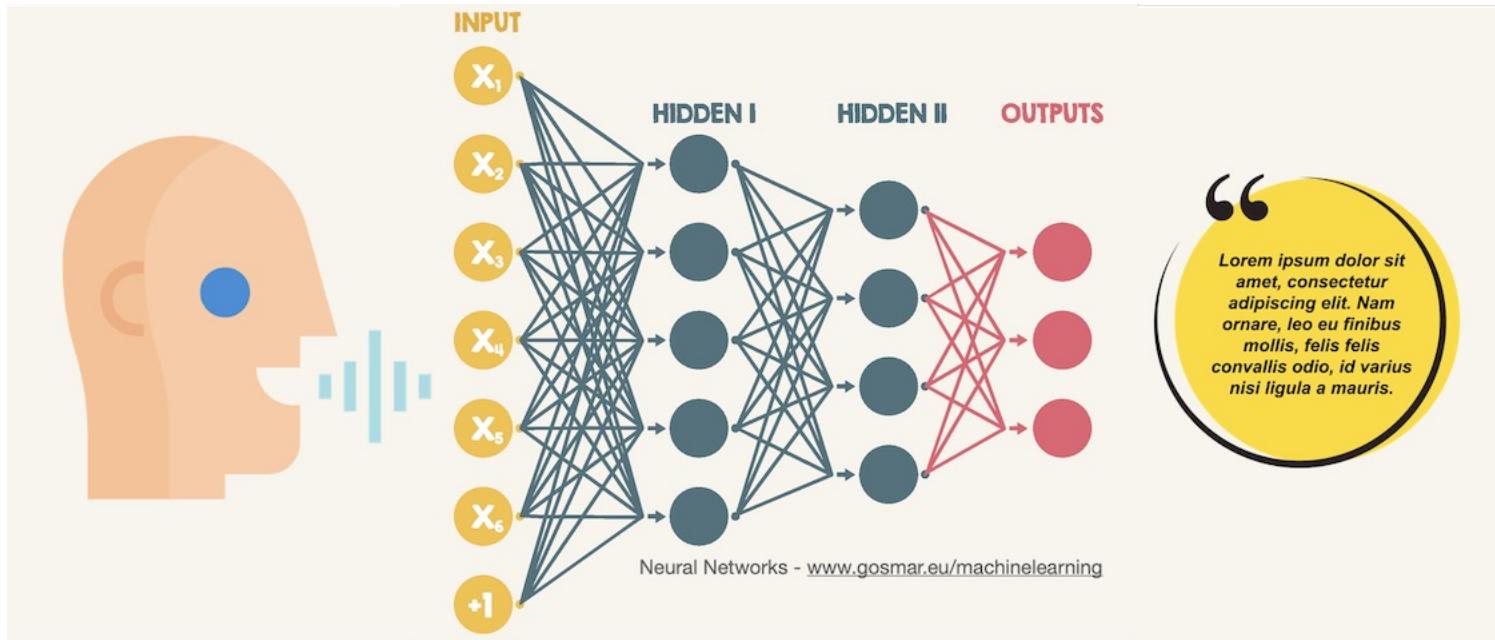
# DL and vision

- Image generation
- [GAN \(Goodfellow et. al, 2014\)](#)
- [Stable Diffusion](#)



# DL and Speech

- Speech Recognition



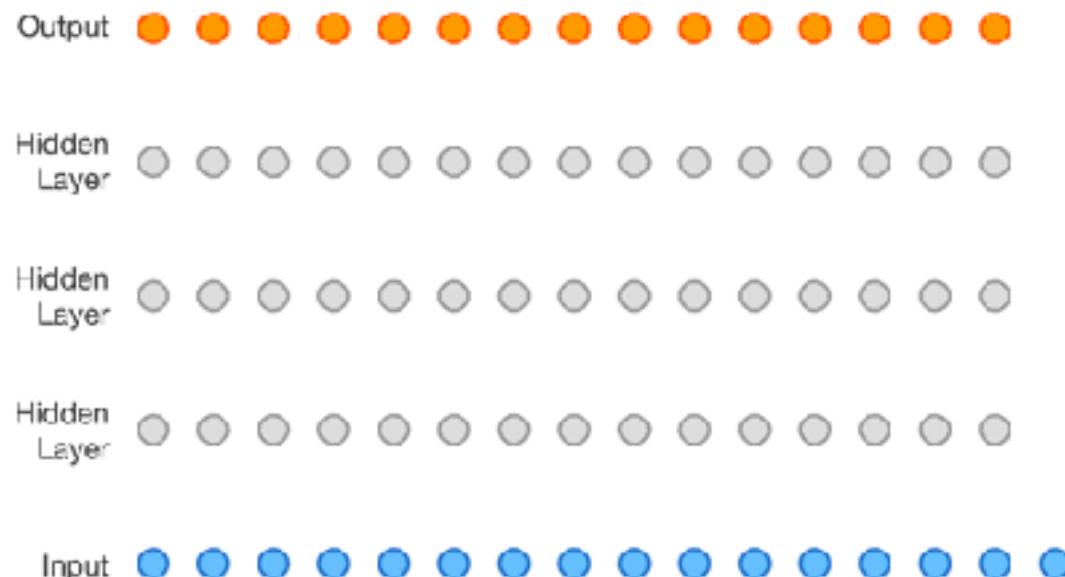
- Check trend of Word Error Rate (WER) on [paperwithcode](#)

Illustration from this [blogpost](#)

# DL and Speech

- Speech synthesis

- Wavenet



# DL and NLP

- Language Understanding

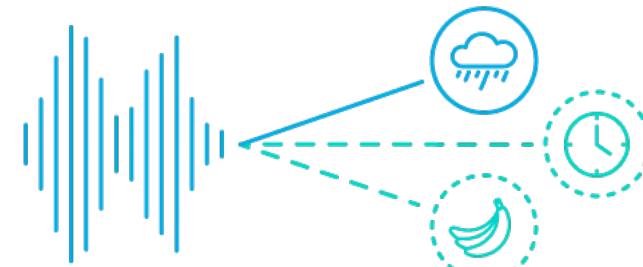


Photo from [here](#) and [here](#)

# DL and NLP

- Language generation

- conversation

You suggest 5 warm places to visit during Christmas, describe each within 20 words

ChatGPT

1. **Maui, Hawaii:** Sunny beaches, festive luaus, and lush landscapes create a tropical Christmas paradise with a unique island charm.
2. **Sydney, Australia:** Celebrate down under with beach picnics, vibrant fireworks at the iconic Opera House, and summer festivities.
3. **Phuket, Thailand:** Escape to stunning beaches, enjoy Thai hospitality, and witness vibrant celebrations with a tropical twist.
4. **Cancun, Mexico:** White sandy beaches, lively fiestas, and cultural warmth make Christmas in Cancun a blissful escape.
5. **Cape Town, South Africa:** Experience a summer Christmas with breathtaking landscapes, outdoor adventures, and diverse cultural celebrations.

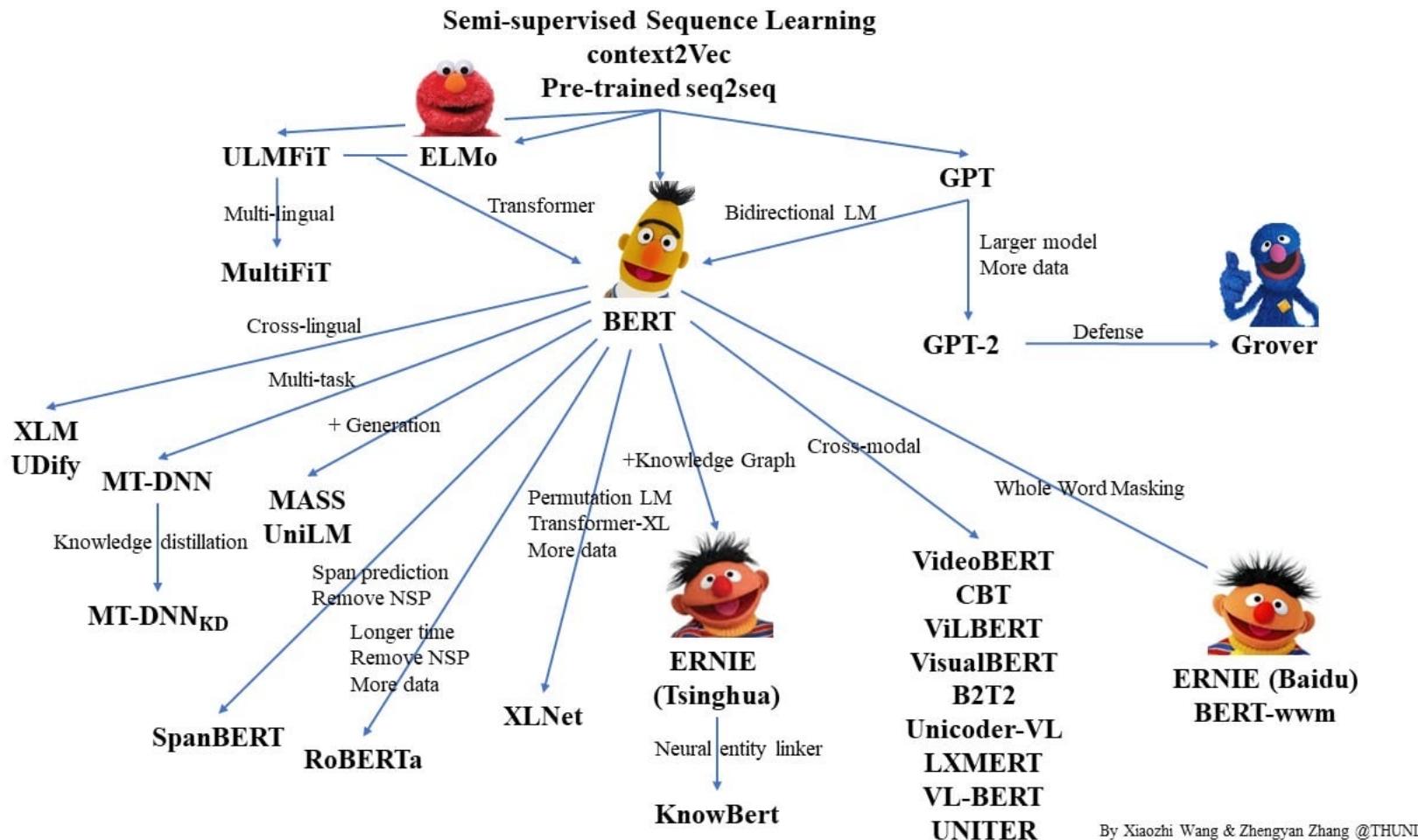
You pick the cheapest one for me

ChatGPT

**Phuket, Thailand:** Offers affordability with budget-friendly accommodations, delicious street food, and a variety of affordable activities amid stunning tropical scenery.

- Translation
- Summarization
- Essay writing
- ....

# DL and NLP

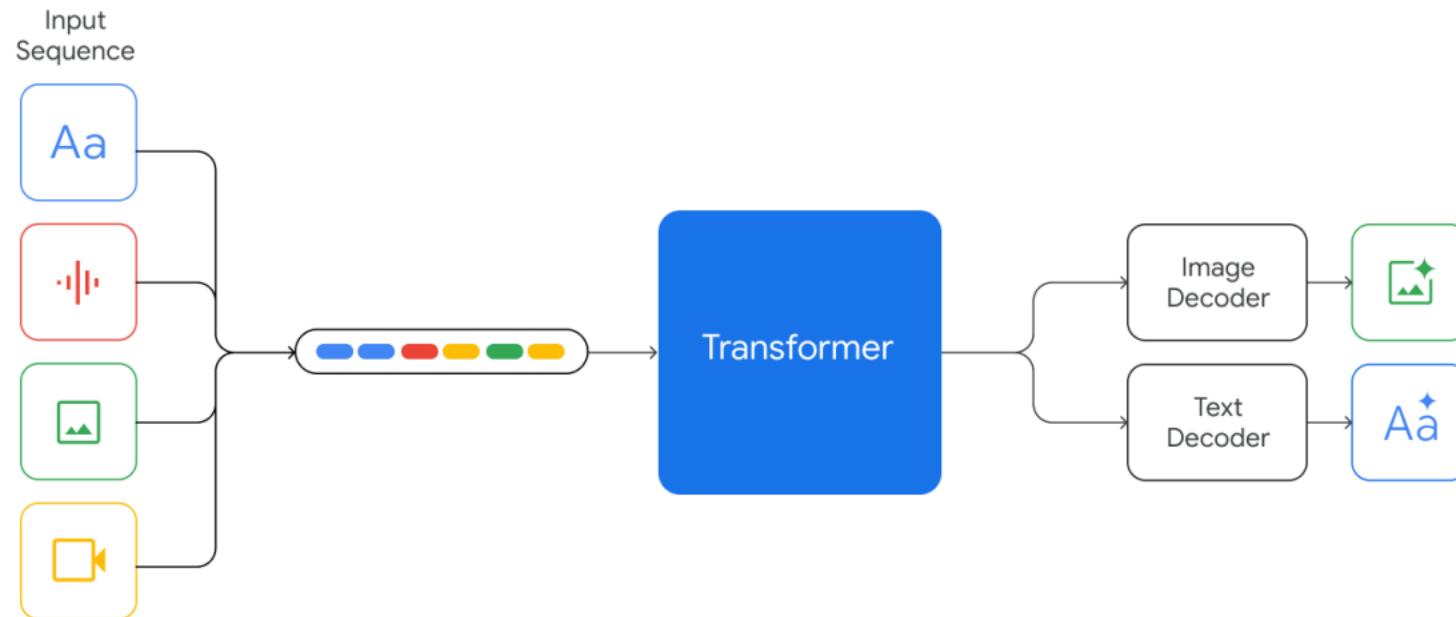


By Xiaozhi Wang & Zhengyan Zhang @THUNLP

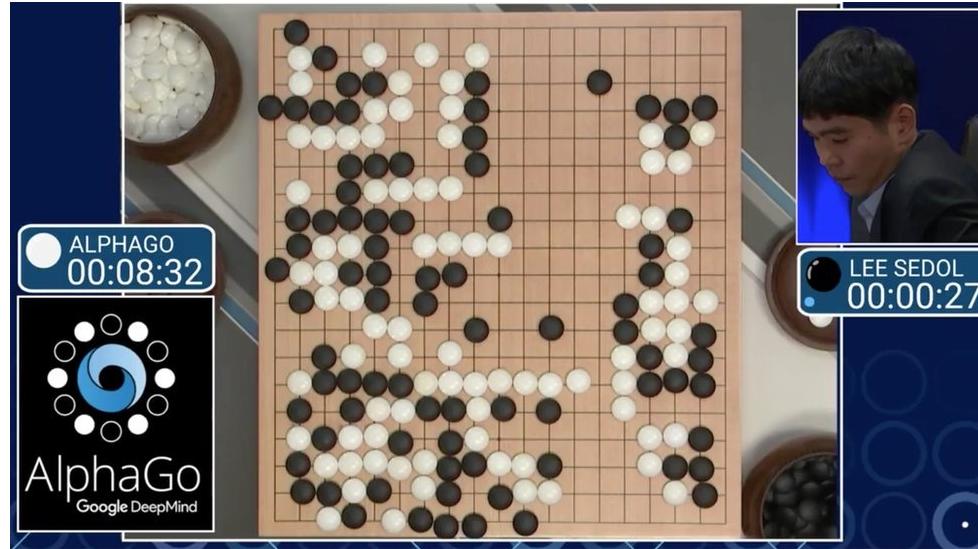
Taxonomy of language models from this [page](#) (outdated of course)

# DL and multi-modality

- Gemini



# DL and Decision Making



Images from [here](#) and [here](#)

# DL and Science

- Predicting protein's 3D structure: [AlphaFold](#)
- Drug discovery

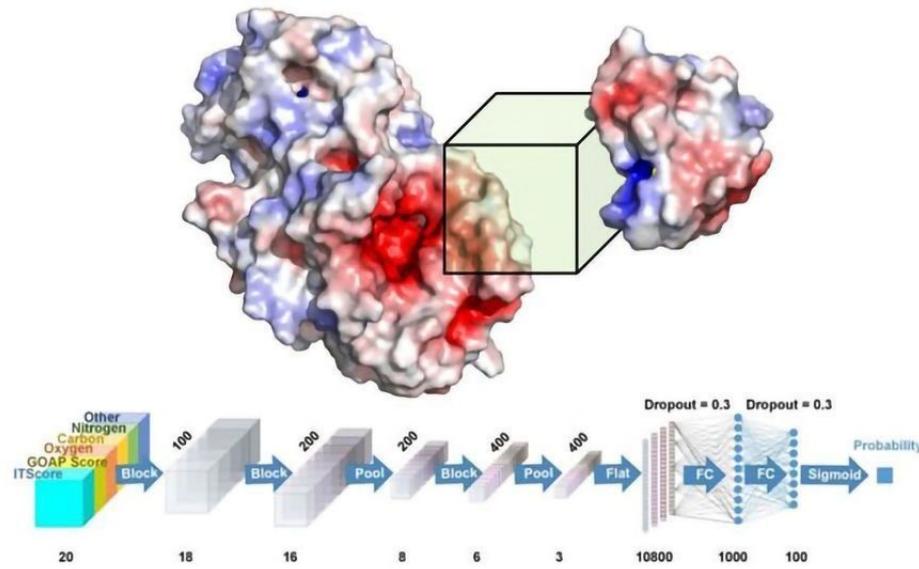
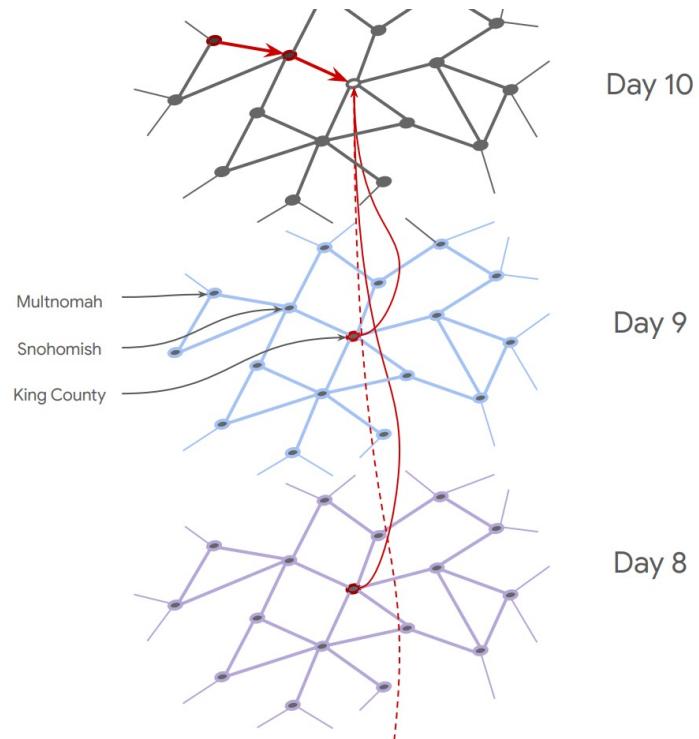


Illustration from this [page](#)

# DL and Social Science

- Modeling spread of Covid



- Recommendation system

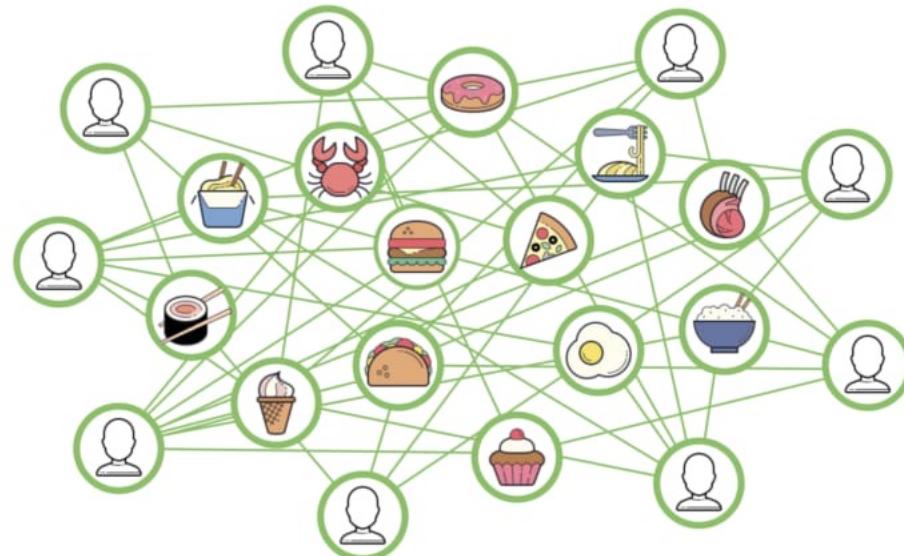
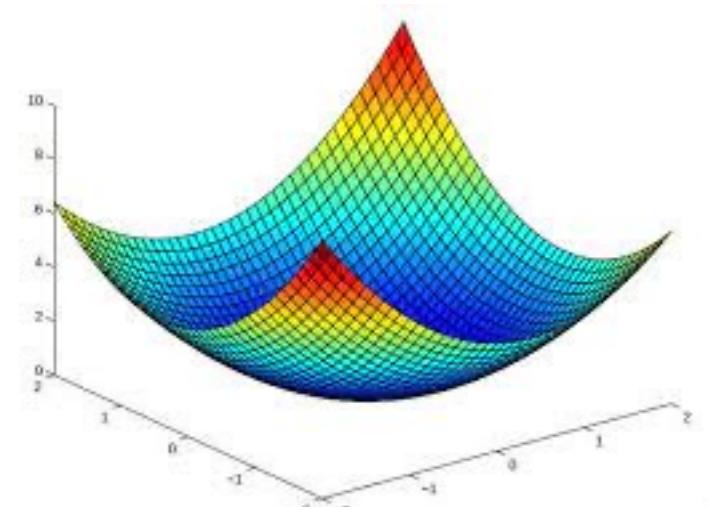
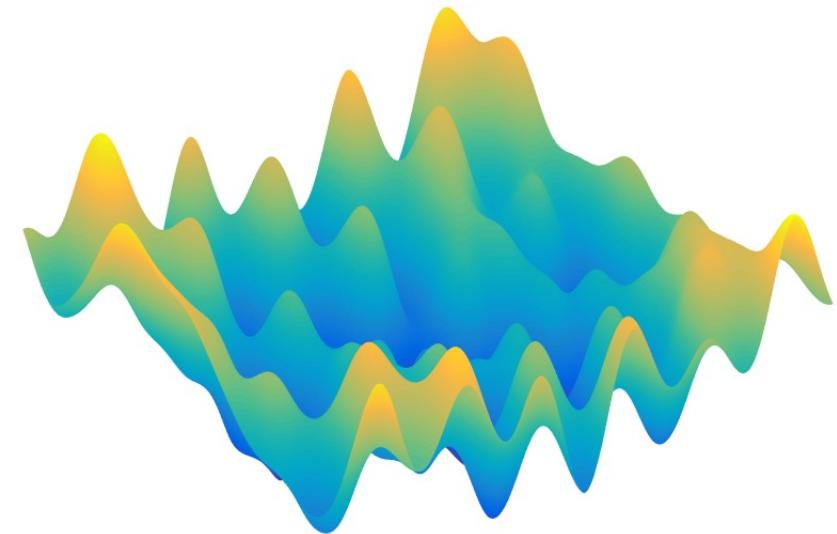


Illustration from [here](#) and [here](#)

# Open problems

- Con-convex optimization
  - Training could land at somewhere sub-optimal
  - where we land at is non-deterministic due to
    - Randomness in Initialization
    - Randomness in feeding training data
    - Hyper parameters like learning rate
- As apposed to convex optimization
  - Unique or at least equally good minimums
  - Convergence Guarantees



# Open Problems

- Generalization
  - Traditional wisdom: model with many parameters may overfit
  - Challenges traditional wisdom ([Zhang et.al 2017](#))
- Robustness

$$\begin{array}{ccc} \text{panda} & + .007 \times & \text{nematode} \\ \text{x} & & \text{sign}(\nabla_x J(\theta, x, y)) \\ & & \text{"nematode"} \\ & & 8.2\% \text{ confidence} \\ & & = \\ & & \text{gibbon} \\ & & \epsilon \text{sign}(\nabla_x J(\theta, x, y)) \\ & & \text{"gibbon"} \\ & & 99.3 \% \text{ confidence} \end{array}$$

The diagram illustrates a adversarial attack on a panda image. It shows the original image  $x$  labeled "panda" with 57.7% confidence. This is combined with a scaled version of the gradient of the loss function with respect to the input,  $.007 \times \text{sign}(\nabla_x J(\theta, x, y))$ , which is labeled "nematode" with 8.2% confidence. The result is a new image labeled "gibbon" with 99.3% confidence.

# Open Problems

- Interpretability
  - What feature is responsible
  - What training sample is responsible
- Factuality, Hallucination



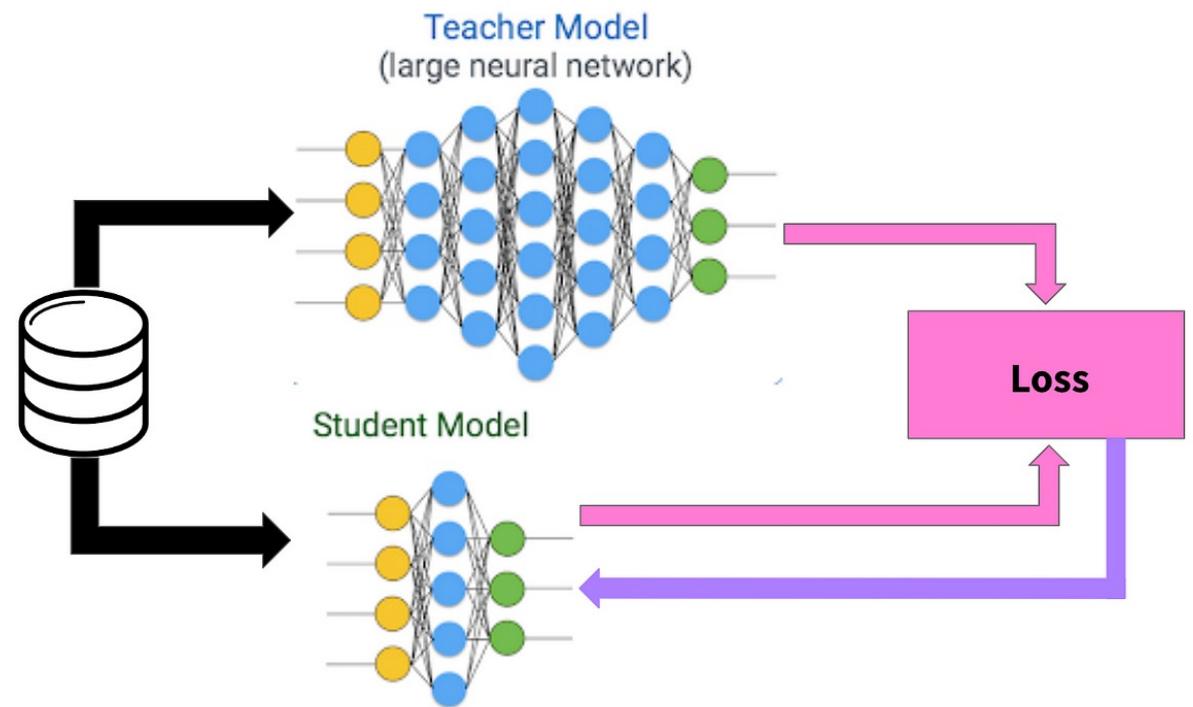
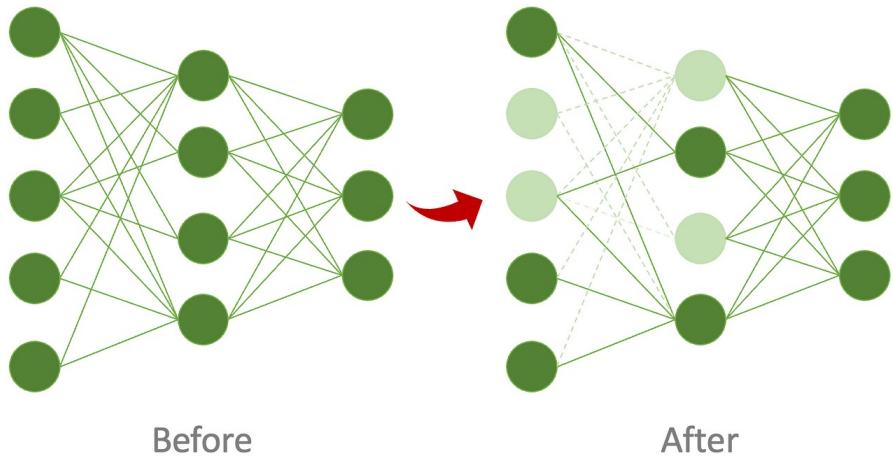
**On the Dangers of Stochastic Parrots: Can Language Models Be Too Big?** 

---

**Authors:**  [Emily M. Bender](#),  [Timnit Gebru](#),  [Angelina McMillan-Major](#),  [Shmargaret Shmitchell](#) [Authors Info & Claims](#)

# Open Problems

- Model efficiency

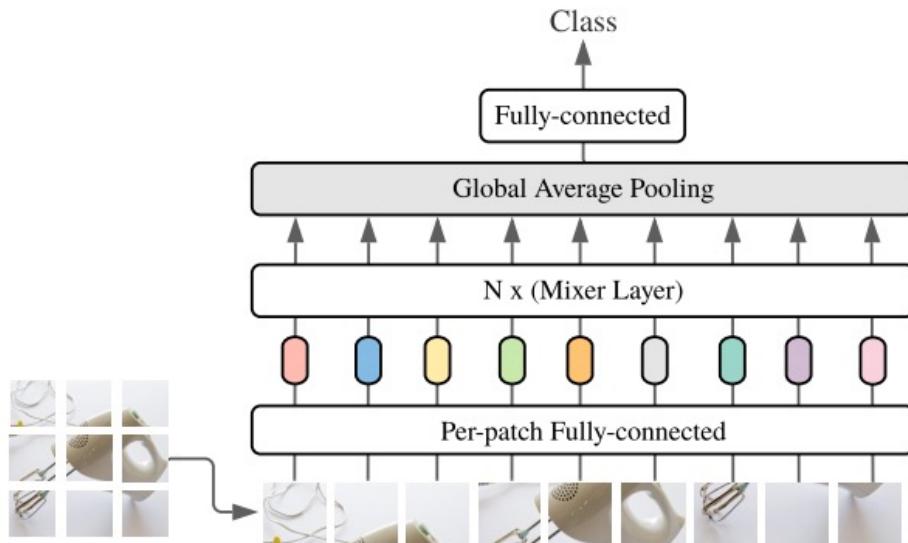


- Neural Architecture Search

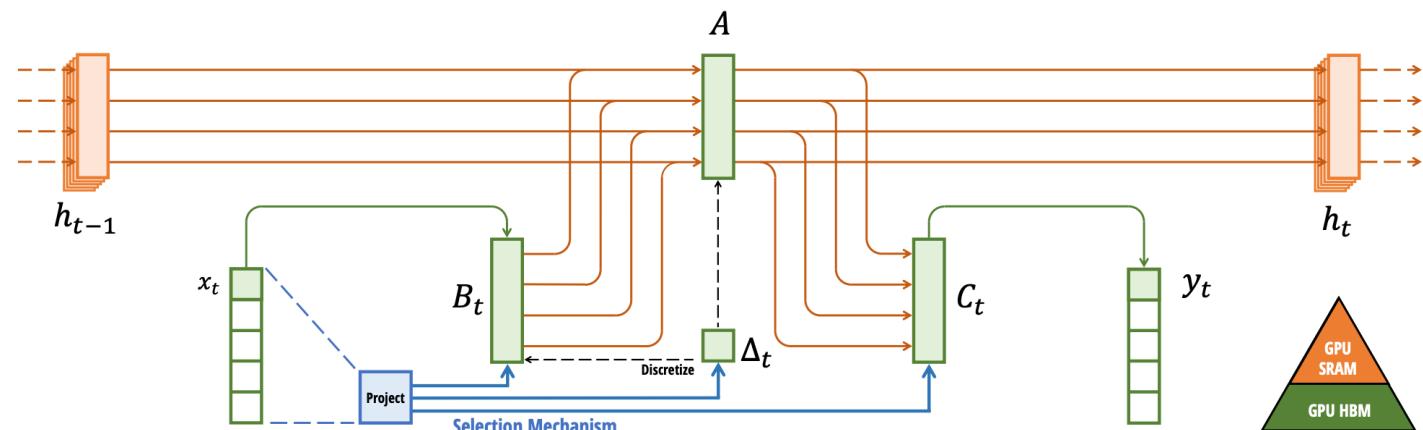
Illustration from [here](#) and [here](#)

# Open Problems

- After transformers?



MLP mixer



Linear-time State Space Models

# Criticism: rush for scaling, expensive



---

## Money Is All You Need

---

Nick Debu  
Tokyo Institute of Bamboo Steamer

### Abstract

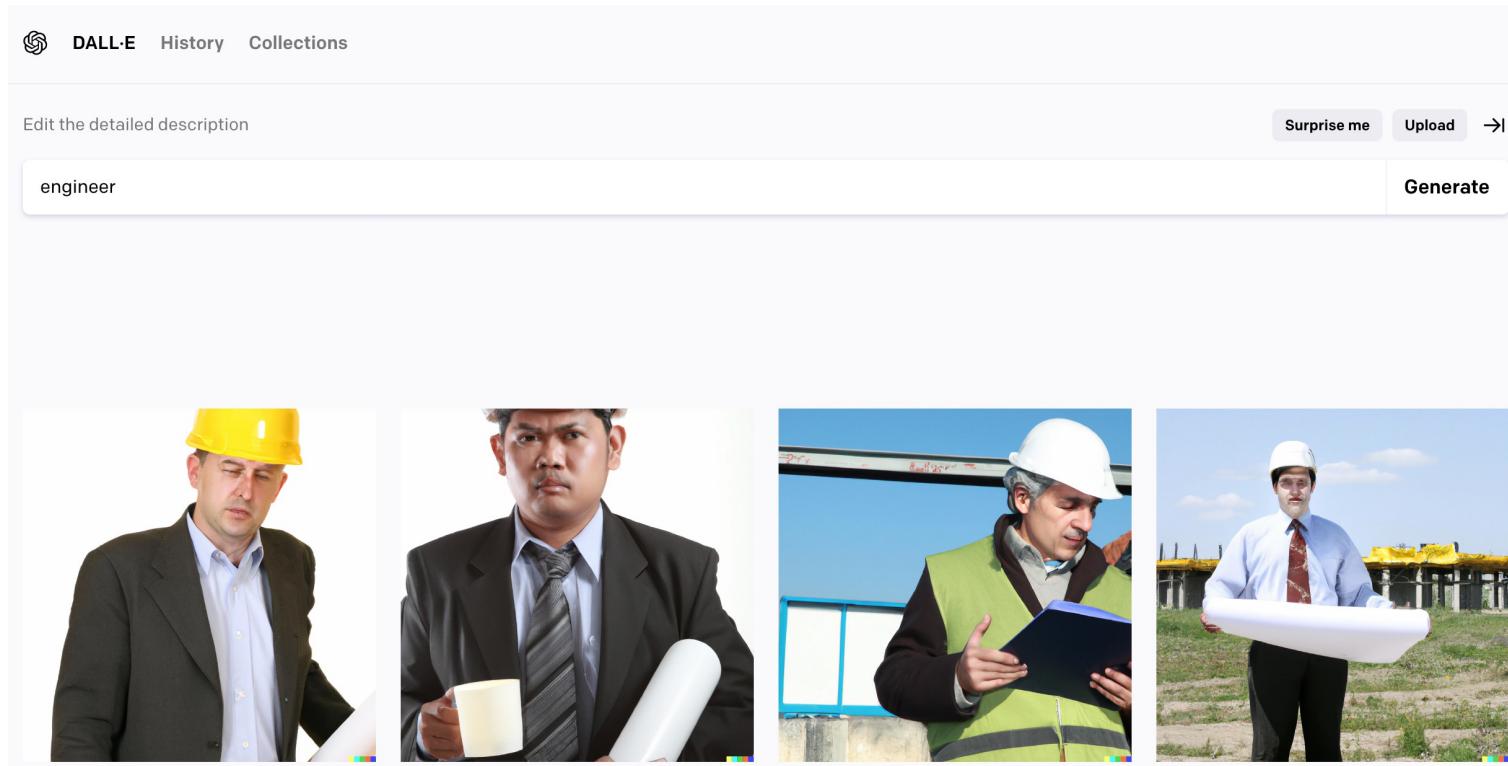
Transformer-based models routinely achieve state-of-the-art results on a number of tasks but training these models can be prohibitively costly, especially on long sequences. We introduce one technique to improve the performance of Transformers. We replace NVIDIA P100s by TPUs, changing its memory from hoge GB to piyo GB. The resulting model performs on par with Transformer-based models while being much more ""TSUYO TSUYO"".

*Photo Source: Internet*

Images from this [page](#)

# Social Impact: Bias and Fairness

Stereotypes: generate more images of one gender/ethnic group



Example from this [article](#)

# Social Impact: Legal Issues

- “deepfaked” celebrities

**A viral video that appeared to show Obama calling Trump a 'dips---' shows a disturbing new trend called 'deepfakes'**

Kaylee Fagan Apr 17, 2018, 1:48 PM PDT

Share | Save



<https://www.youtube.com/watch?v=cQ54GDm1eL0>

See article [here](#) and [here](#)

- Use of data

BUSINESS

Israel-Hamas war Can Trump still run for president? NYE store hours Missouri beat Ohio Kathy

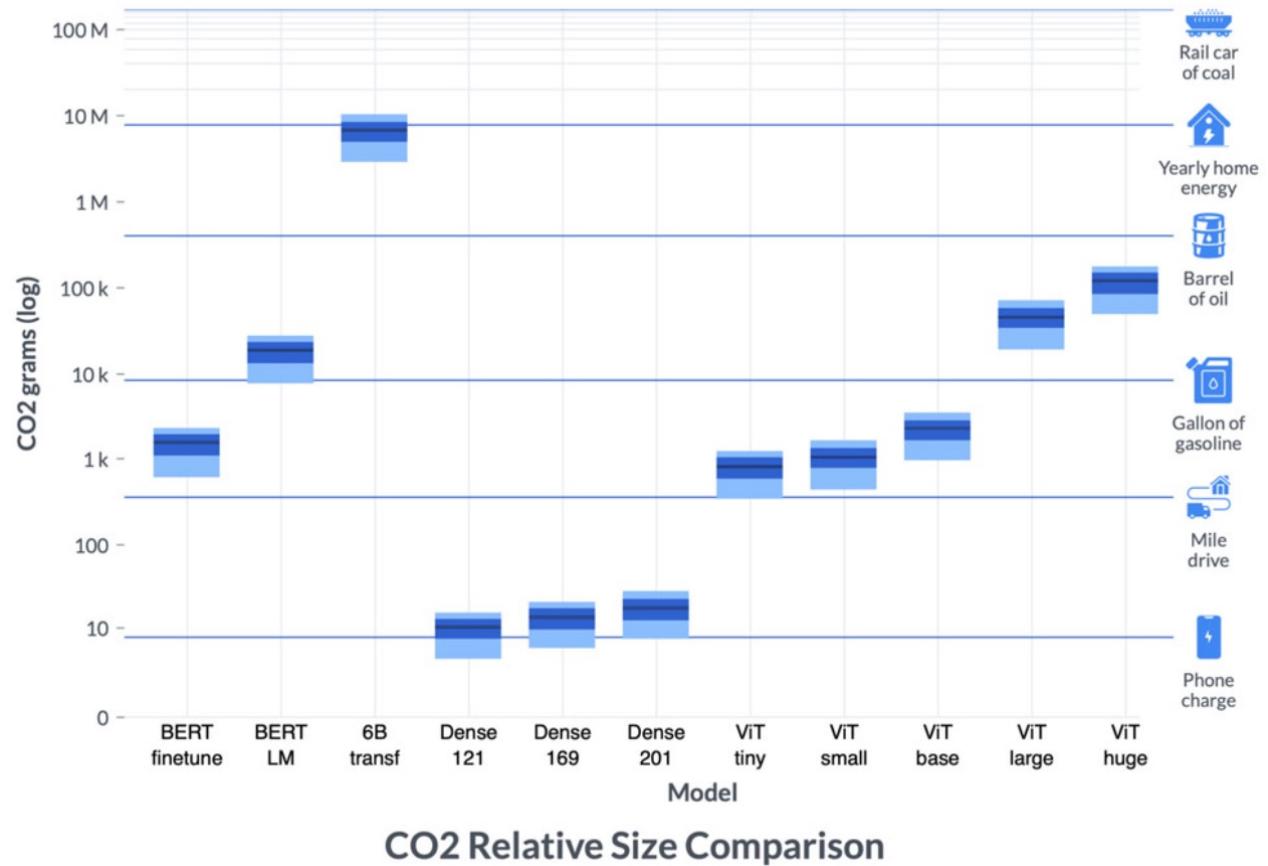
**The New York Times sues OpenAI and Microsoft for using its stories to train chatbots**



# Social impact: Carbon footprint

Consumption	CO <sub>2</sub> e (lbs)
Air travel, 1 passenger, NY↔SF	1984
Human life, avg, 1 year	11,023
American life, avg, 1 year	36,156
Car, avg incl. fuel, 1 lifetime	126,000
Training one model (GPU)	
NLP pipeline (parsing, SRL)	39
w/ tuning & experimentation	78,468
Transformer (big)	192
w/ neural architecture search	626,155

Table 1: Estimated CO<sub>2</sub> emissions from training common NLP models, compared to familiar consumption.<sup>1</sup>



See article [here](#) and [here](#)

# Agenda

- About this class
- Deep Learning nowadays
- Math
- Programming

# Linear algebra: vectors

- $n$  dimensional vector  $\mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$
- If  $\mathbf{x}$  is real (common in deep learning), we can write  $\mathbf{x} \in \mathbb{R}^n$
- Inner (dot) product between n-dimensional vectors  $\mathbf{x}$  and  $\mathbf{y}$

$$\mathbf{x} \cdot \mathbf{y} = \sum_{i=1}^n x_i y_i$$

- We call  $\mathbf{x}$  and  $\mathbf{y}$  orthogonal if  $\mathbf{x} \cdot \mathbf{y} = 0$

# Linear algebra: vectors

- Norm of a vector  $x \in \mathbb{R}^n$

- $L^p$  norm for  $p \in \mathbb{R}, p \geq 1$

$$\|x\|_p = \left( \sum_{i=1}^n x_i^p \right)^{\frac{1}{p}}$$

- For  $0 < p < 1$ , the above is not a norm
  - $L^0$  norm: number of non-zero elements in  $x$
  - Unit norm ball for  $p$  in 0.1 to 2



([wiki](#))

# Linear algebra

- Matrix multiplication

$A \in \mathbb{R}^{m \times n}$ ,  $B \in \mathbb{R}^{n \times p}$ , then  $AB$  is  $m \times p$ :

$$(AB)_{i,j} = \sum_{k=1}^n A_{ik} B_{kj}$$

- diagonal matrix: off-diagonal elements are all zero
- Identity matrix  $I$ : diagonal matrix with all 1's on its diagonal
- Inverse of a square matrix  $A$ :  $A^{-1}$  such that  $A^{-1}A = AA^{-1} = I$
- Transpose of a matrix:  $A^T$

# Special Matrices

- Orthogonal matrix: a square matrix  $A$  such that

$$A^T = A^{-1}$$

- Symmetric matrix: a square matrix  $A$  such that

$$A = A^T$$

- Toeplitz matrix: a square matrix where each diagonal is constant

$$\begin{bmatrix} a_0 & a_{-1} & a_{-2} & \cdots & \cdots & a_{-(n-1)} \\ a_1 & a_0 & a_{-1} & \ddots & & \vdots \\ a_2 & a_1 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & a_{-1} & a_{-2} \\ \vdots & & \ddots & a_1 & a_0 & a_{-1} \\ a_{n-1} & \cdots & \cdots & a_2 & a_1 & a_0 \end{bmatrix}$$

# Special Matrices

- Circulant matrix: a Toeplitz matrix, each row being a circulant shift of the preceding

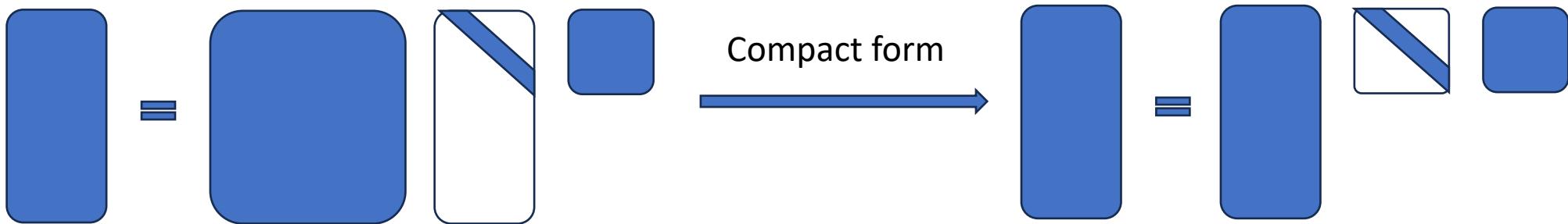
$$\begin{bmatrix} c_0 & c_{n-1} & \cdots & c_2 & c_1 \\ c_1 & c_0 & c_{n-1} & & c_2 \\ \vdots & c_1 & c_0 & \ddots & \vdots \\ c_{n-2} & & \ddots & \ddots & c_{n-1} \\ c_{n-1} & c_{n-2} & \cdots & c_1 & c_0 \end{bmatrix}$$

- Givens rotation matrix (we restrict to 2D): the operator that rotates a 2D vector counter clockwise by  $\theta$

$$\begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix}$$

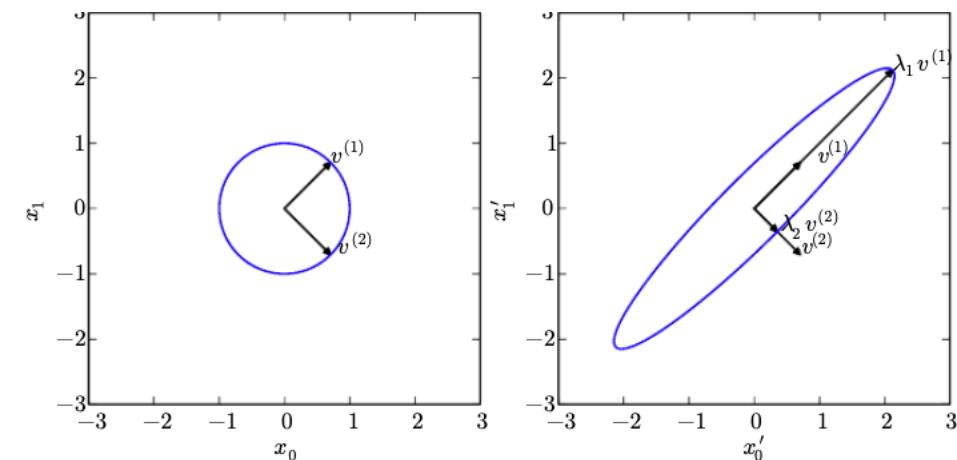
# Singular Value decomposition

- Any matrix  $A \in \mathbb{R}^{m \times n}$  can be factorized into  $A = U\Sigma V^T$
- $U \in \mathbb{R}^{m \times m}$ ,  $V \in \mathbb{R}^{n \times n}$  orthogonal
- $\Sigma \in \mathbb{R}^{m \times n} = \text{diag}(\sigma_1, \dots, \sigma_{\min(m,n)}), \sigma_i \geq 0$
- Illustration



# Eigen Value Decomposition

- Square matrix  $A \in \mathbb{R}^{m \times m}$ , decompose into  $A = U\Lambda U^{-1}$
- Symmetric  $A$  (common for this class), then
- $U$  is orthogonal, and  $A = U\Lambda U^T$
- $\Lambda = \text{diag}\{\lambda_1, \dots, \lambda_m\}$  in descending order
- Viewing  $A$  as operator, columns of  $U$ 
  - $U_1$ : the direction amplified the most
  - $U_2$ : the **orthogonal** direction amplified the most
  - ...
- When all  $\lambda_i \geq 0$ , we call  $A$  positive semi definite (PSD), denoted as  $A \succeq 0$



# Operators

- Trace of a square matrix  $A$ :  $tr(A) = \sum_i A_{i,i}$
- $tr(AB) = tr(BA)$
- Norm of a matrix  $A$ 
  - 1-norm:  $\|A\|_1 = \max_j \sum_i |A_{i,j}|$
  - $\infty$ -norm:  $\|A\|_\infty = \max_i \sum_j |A_{i,j}|$
  - Frobenius norm

$$\|A\|_F = \sqrt{\sum_{i,j} |A_{i,j}|^2} = \sqrt{\sum_i \sigma_i^2} = \sqrt{tr(A^T A)} = \sqrt{tr(A A^T)}$$

- $tr(AB) \leq \|A\|_F \cdot \|B\|_F$

# Operators

- Spectral norm  $\|A\|_2 = \sigma_1$
- Nuclear-norm  $\|A\|_* = \sum_i \sigma_i$
- Rank( $A$ )= $\sum_{i=1}^{\min(m,n)} \mathbb{I}(\sigma_i > 0)$
- Determinant of a square matrix  $A$ :  $\det(A) = \prod_i \sigma_i^{\max \sigma_i}$
- Condition number of square matrix  $A$ :  $\kappa(A) = \frac{\max_i \sigma_i}{\min_i \sigma_i}$ 
  - Solving  $Ax = b \Rightarrow x = A^{-1}b$ , perturb  $b$  by  $\epsilon$ , relative error
$$\frac{\|A^{-1}\epsilon\|}{\|A^{-1}b\|} / \frac{\|b\|}{\|e\|} \leq \kappa(A)$$
  - Convergence of optimization problems

# Linear dependency

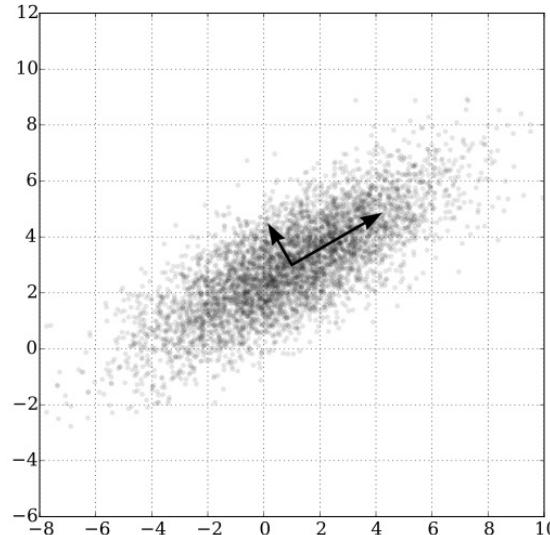
- Consider a set of vectors  $[x_1, \dots x_n]$ , where  $x_i \in \mathbb{R}^m$
- **Linearly independent:** none of them can be written as the linear combination of the rest
  - Question: what if  $n > m$
- $\text{span}(x_1, \dots x_n)$  is the set of all linear combinations of  $x_i$ 's,

$$\sum_{i=1}^n \alpha_i x_i, \forall \alpha_i \in \mathbb{R}$$

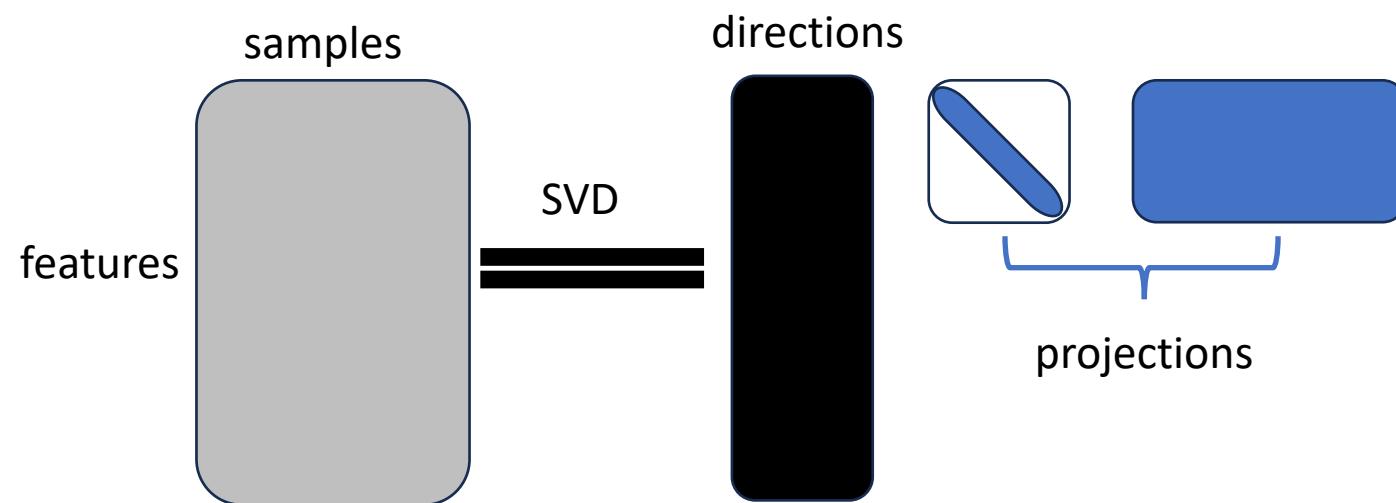
# Subspace

- Consider  $n < m$ , and Linearly independent  $\mathbf{x}_1, \dots, \mathbf{x}_n$
- $\text{span}(\mathbf{x}_1, \dots, \mathbf{x}_n)$  defines a subspace of  $\mathbb{R}^m$
- $\mathbf{x}_1, \dots, \mathbf{x}_n$  is a basis for the subspace
- **Intrinsic** dimension is  $n$ , e.g.,  $m=3$ ,  $n=2$ , 2D plane in  $\mathbb{R}^3$
- How to get the basis from linearly dependent vectors?
  - left singular vectors corresponding to non-zero singular values

# Principal Component Analysis (PCA)



- For  $i=1, \dots, \text{reduced\_dimension}$ :
  - Find next orthogonal direction with biggest variance
  - Project feature to this direction
- Easily solved by singular value decomposition (SVD)



# PCA: More Math

- Data matrix  $X = [x_1, \dots, x_n] \in \mathbb{R}^{m \times n}$ , assuming “centered”
- Find first vector  $g \in \mathbb{R}^m$  to project  $x_i$ ’s upon, and capture biggest variance
- Solve  $\max_{g: \|g\|=1} \{\|g^T X\|_2^2 = g^T X X^T g\}$
- Denote the SVD of  $X := U \Sigma V^T$ , so  $X X^T = U \Sigma^2 U^T$
- Maximized when  $g = U_1$
- The 2nd vector should be  $U_2, \dots$

# Linear Least Squares

$$\min_x \|Ax - b\|^2$$

- $A \in \mathbb{R}^{m \times n}$ ,  $x \in \mathbb{R}^n$ ,  $m \geq n$
- $x^* = A^\dagger b$ , where  $A^\dagger = (A^T A)^{-1} A^T$  is pseudo-inverse
- What if  $m < n$ ?
  - More than one solution
  - Usually impose some **regularizer** on  $x$ , e.g.,  $L^1$  or  $L^2$  norm
  - $\min_x \|Ax - b\|^2 + \lambda \|x\|^2 \Rightarrow x^* = (A^T A + \lambda I)^{-1} A^T b$  (ridge regression)
  - $\min_x \|Ax - b\|^2 + \lambda \|x\|_1$  (Lasso)

# Gradient and Hessian

- Many learning problems aim at

$$\min_{\mathbf{w}} \sum_i \ell(\mathbf{w}, \mathbf{x}_i)$$

- Many solvers will involve gradient

$$\nabla \ell(\mathbf{w}, \mathbf{x}_i) = \frac{\partial \ell}{\partial \mathbf{w}}$$

- Sometimes we will also talk about 2<sup>nd</sup> order gradient (Hessian)

$$\mathbf{H} = \frac{\partial^2 \ell}{\partial \mathbf{w} \partial \mathbf{w}^T}$$

# Matrix Calculus

- Calculate derivative of function defined on matrices/vectors
- $f: \mathbb{R}^m \mapsto \mathbb{R}$ : gradient is  $\mathbb{R}^m$
- $f: \mathbb{R}^{m \times n} \mapsto \mathbb{R}$ : gradient is  $\mathbb{R}^{m \times n}$
- $f: \mathbb{R}^m \mapsto \mathbb{R}^n$ : gradient is  $\mathbb{R}^{m \times n}$  (Jacobian)
- Hessian:
  - we typically deal with  $f: \mathbb{R}^m \mapsto \mathbb{R}$  only
  - $m \times m$  matrix
- I often check this [wikipage](#)

# Chain rule

- For an  $L$ -layer network,  $\ell(\mathbf{w}, \mathbf{x})$  is a composite function, i.e.,

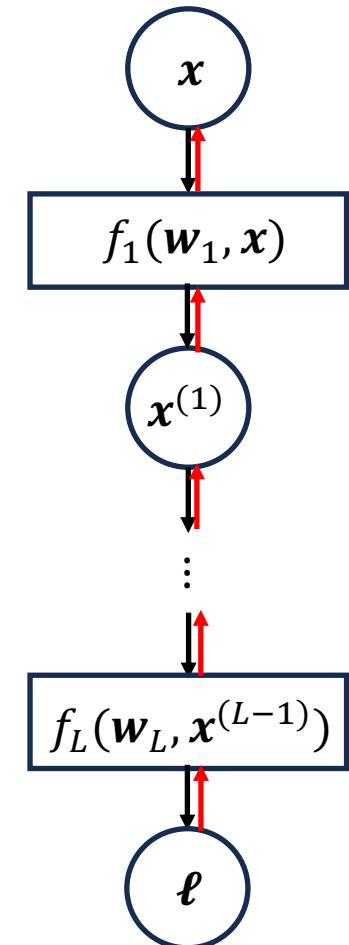
$$\begin{aligned}\mathbf{x}^{(1)} &= f_1(\mathbf{w}_1, \mathbf{x}) \\ \mathbf{x}^{(2)} &= f_2(\mathbf{w}_2, \mathbf{x}^{(1)}) \\ &\vdots \\ \ell &= f_L(\mathbf{w}_L, \mathbf{x}^{(L-1)})\end{aligned}$$

- Chain rule and back-propagation:

$$\frac{\partial \ell}{\partial \mathbf{x}^{(l)}} = \frac{\partial \ell}{\partial \mathbf{x}^{(l+1)}} \cdot \boxed{\frac{\partial \mathbf{x}^{(l+1)}}{\partial \mathbf{x}^{(l)}}}$$

$$\frac{\partial \ell}{\partial \mathbf{w}_l} = \frac{\partial \ell}{\partial \mathbf{x}^{(l)}} \cdot \boxed{\frac{\partial \mathbf{x}^{(l)}}{\partial \mathbf{w}_l}}$$

Jacobian

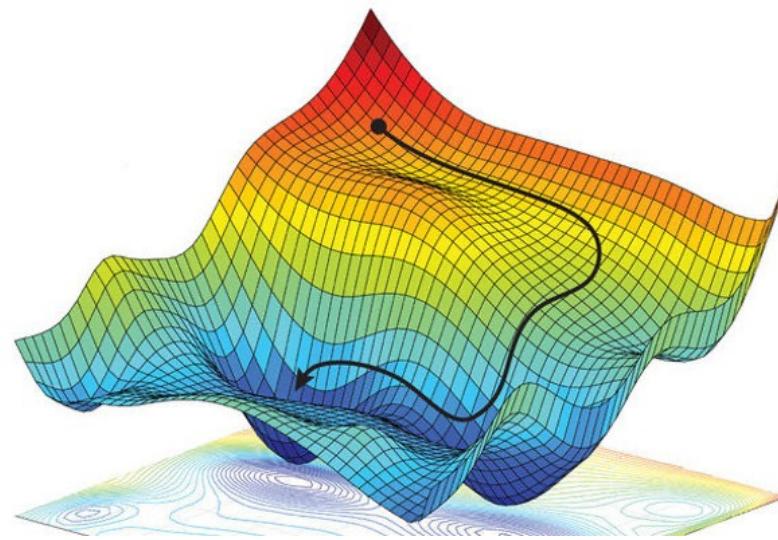


# Gradient Descent (GD)

For step  $s = 0, 1, \dots$

$$\mathbf{w}(s + 1) = \mathbf{w}(s) - \eta \nabla \ell(\mathbf{w}(s)),$$

till  $\nabla \ell(\mathbf{w}(s)) \approx 0$  (or max steps reached, or loss reduction is tiny)



# Stochastic Gradient Descent (SGD)

- Remember we want to  $\min_{\mathbf{w}} \sum_{i=1}^N \ell(\mathbf{w}, \mathbf{x}_i)$
- If  $N$  is big, Compute  $\nabla \ell(\mathbf{w}, \mathbf{x}_i)$  for all  $i$ 's can be costly
- So we sample a mini-batch of  $i$ 's
- For step  $s = 0, 1, \dots$

$$\mathbf{w}(s+1) = \mathbf{w}(s) - \eta \sum_{i \in \mathcal{B}} \nabla \ell(\mathbf{w}(s), \mathbf{x}_i),$$

till stopping criteria met

# Probability Distribution

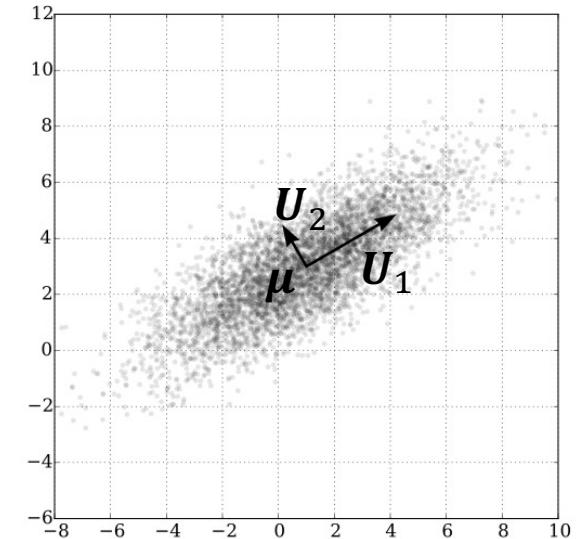
- Discrete random variable
  - Probability Mass Function (PMF):  $P(X = x_i)$
  - $P(X = x_i) \geq 0$  and  $\sum_i P(X = x_i) = 1$
- Continuous random variable
  - Probability Density Function (PDF):  $p(x)$
  - $p(x) \geq 0$  and  $\int_x p(x) = 1$
- The above can be generalized to random vectors
- Cumulative Distribution Function (CDF) for real valued random variable
  - $F(x) = \Pr(X \leq x)$
  - Continuous case:  $p(x) = \frac{dF(x)}{dx}$

# Expectation, Variance, Covariance

- Expectation  $\mathbb{E}_{X \sim p}[f(X)] = \int f(x)p(x)dx$
- Variance  $Var_{X \sim p}[f(X)] = \mathbb{E}_{X \sim p}[(f(X) - \mathbb{E}_{X \sim p}[f(X)])^2]$
- Covariance  
$$Cov(f(X), g(Y)) = \mathbb{E}[(f(X) - \mathbb{E}[f(X)])(g(Y) - \mathbb{E}[g(Y)])]$$
- Covariance matrix of random vector  $\mathbf{x} \in \mathbb{R}^n$   
$$Cov(\mathbf{x})_{i,j} = Cov(x_i, x_j)$$
$$Cov(\mathbf{x})_{i,i} = Var(x_i)$$

# (multivariate) Gaussian Distribution

- $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) \in \mathbb{R}^n$ , where  $\boldsymbol{\Sigma} \succcurlyeq \mathbf{0}$
- $p(\boldsymbol{x}) = \frac{1}{\sqrt{(2\pi)^n \det \boldsymbol{\Sigma}}} \exp\left(-\frac{1}{2} (\boldsymbol{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\boldsymbol{x} - \boldsymbol{\mu})\right)$
- Picturized as ellipsoid (centered at  $\boldsymbol{\mu}$ ) in  $\mathbb{R}^n$ 
  - Let eigen decomposition  $\boldsymbol{\Sigma} = \boldsymbol{U} \boldsymbol{\Lambda} \boldsymbol{U}^T$
  - Axes along columns  $\boldsymbol{U}_i$ 's. Axis length related to  $\lambda_i$ 's
- Maximum Likelihood Estimate (MLE) for  $\boldsymbol{\mu}$  and  $\boldsymbol{\Sigma}$ :
$$(\hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\Sigma}}) = \arg \max_{\boldsymbol{\mu}, \boldsymbol{\Sigma}} \sum_{i=1}^N \log p(\boldsymbol{x}_i | \boldsymbol{\mu}, \boldsymbol{\Sigma})$$
$$\Rightarrow \hat{\boldsymbol{\mu}} = \frac{1}{N} \sum_{i=1}^N \boldsymbol{x}_i, \hat{\boldsymbol{\Sigma}} = \frac{1}{N} \sum_{i=1}^N (\boldsymbol{x}_i - \hat{\boldsymbol{\mu}})(\boldsymbol{x}_i - \hat{\boldsymbol{\mu}})^T$$



# Conditional Probability

- $P(Y = y|X = x) = \frac{P(X=x, Y=y)}{P(X=x)}$
- “Lazy” expression:  $p(y|x) = \frac{p(x,y)}{p(x)}$
- Bayes Rule

$$p(x|y) = \frac{p(x)p(y|x)}{p(y)}$$

- **Marginalize**  $p(x) = \int p(x, y)dy$
- Independence:  $p(x, y) = p(x)p(y)$

# Mixture of Gaussian

- latent variable  $c \sim p(c)$
- Component distribution  $p(x|c) = \mathcal{N}(x; \boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c)$
- $p(x) = \sum_{c=1}^C p(c)p(x|c)$
- MLE estimator for  $\{\boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c\}$ 's (via Expectation-Maximization):
- Initialize  $\{\boldsymbol{\mu}_c(0), \boldsymbol{\Sigma}_c(0)\}$  for  $c = 1, \dots, C$ . Denoted as  $\boldsymbol{\theta}(0)$
- for step  $s = 0, 1, \dots$ 
  - E-step: estimate  $p(c|x_i, \boldsymbol{\theta}(0))$  for all  $i$ 's, then get expected log-likelihood
$$\sum_i \mathbb{E}_{c \sim p(c|x_i, \boldsymbol{\theta}(0))} [\log p(x_i, c|\boldsymbol{\theta})]$$
  - M-step:  $\boldsymbol{\theta}(s+1) \leftarrow \text{maximizer of the above}$

# Simple application of EM: k-means

- Assume  $\Sigma = \sigma^2 \mathbf{I}$  where  $\sigma^2$  is some constant that we don't care
- We only need to estimate cluster centers  $\mu_c$ , for  $c = 1, \dots, C$
- Random initialize guessed  $\{\mu_c(0)\}_{c=1}^C$ . For step  $s = 0, 1, \dots$
- E-step: calculate cluster assignment for each sample  $x_i$ :

$$p(c|x_i, \{\mu_c(s)\}_{c=1}^C)$$

We may simplify by assuming 1-hot assignment

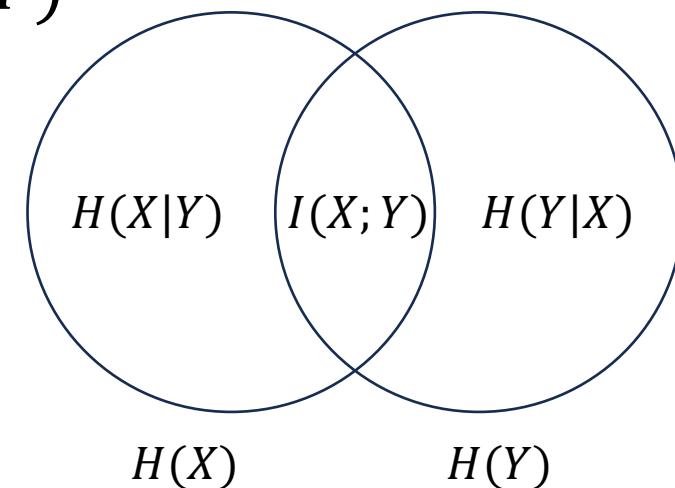
- M-step: update centers  $\{\mu_c(s + 1)\}_{c=1}^C$  based on the assignment

# Entropy

- Shannon Entropy:  $H(X) = \mathbb{E}_{x \sim p}[-\log p(x)]$ 
  - sometimes denoted as  $H(p)$
  - bits to encode a random variable, given its pdf
- Cross Entropy  $H(p, q) = \mathbb{E}_{x \sim p}[-\log q(x)]$ 
  - Bits to encode a random variable, according to a different pdf,  $q(\cdot)$
  - Need more bits than  $H(p)$
- KL divergence  $KL(p||q) = \mathbb{E}_{x \sim p} \left[ \log \frac{p(x)}{q(x)} \right] = H(p, q) - H(p)$ 
  - Asymmetric  $KL(p||q) \neq KL(q||p)$
  - In supervised learning, we often have  $p$  as ground-truth label, and  $q$  as model prediction

# Mutual Information

- Joint entropy  $H(X, Y) = \mathbb{E}_{x,y \sim p(x,y)}[-\log p(x, y)]$
- Conditional Entropy  $H(Y|X) = \mathbb{E}_{x,y \sim p(x,y)}[-\log p(y|x)]$ 
  - How much uncertainty left in  $Y$  given  $X$
- $I(X; Y) = H(Y) - H(Y|X) = H(X) - H(X|Y)$   
 $= H(X) + H(Y) - H(X, Y)$
- $I(X; Y) = KL(p(X, Y)||p(X)p(Y))$
- $I(X; Y) = 0$ :  $X$  and  $Y$  are independent



# Transform of a random variable/vector

- Let  $f: \mathbb{R} \mapsto \mathbb{R}$  be a monotonically increasing function. Consider Random variable  $X$  with pdf  $p(x)$ . Then  $Y = f(X)$  has pdf

$$p(f^{-1}(y)) \left| \frac{\partial f^{-1}(y)}{\partial y} \right|$$

- Generalize to random vectors:

Let  $f: \mathbb{R}^n \rightarrow \mathbb{R}^n$  be a injective function. Consider random vector with pdf  $p(\mathbf{x})$ . Then  $Y = f(X)$  has pdf

$$p(f^{-1}(\mathbf{y})) \left| \det \frac{\partial f^{-1}(\mathbf{y})}{\partial \mathbf{y}} \right|$$

# Agenda

- About this class
- Deep Learning nowadays
- Math
- Programming

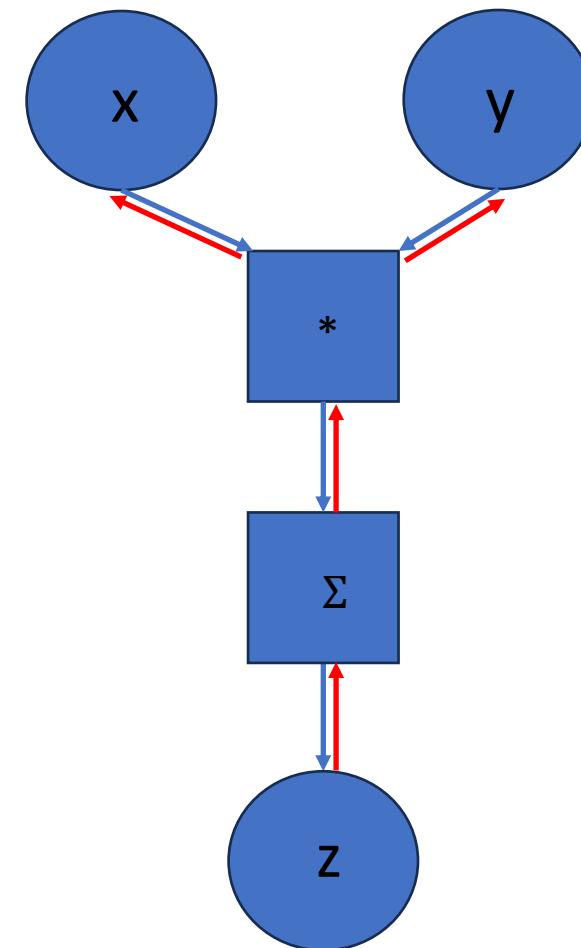
# PyTorch: Tensors

- Installation: pip3 install torch
- The most basic object: torch.Tensor
  - Very much like NumPy Arrays, but also supports:
  - auto-grad (torch.Tensor.backward)
  - GPU acceleration (torch.Tensor.to(GPU\_id))

```
[>>> import torch
[>>> x=torch.tensor([1.0, 2.0], requires_grad=True)
[>>> y=torch.tensor([3.0, 4.0], requires_grad=True)
[>>> z=torch.sum(x*y)
[>>> x
tensor([1., 2.], requires_grad=True)
[>>> y
tensor([3., 4.], requires_grad=True)
[>>> z
tensor(11., grad_fn=<SumBackward0>)
[>>> z.backward()
[>>> x.grad
tensor([3., 4.])
[>>> y.grad
tensor([1., 2.])
```

# PyTorch: Computation Graph

- $z = \text{torch.sum}(x * y)$
- When we call `z.backward()`
  - Populates a field `.grad`
  - Avoids deriving gradient by hand



# PyTorch: Model

- inherits `torch.nn.Module`
- implement at least two functions
  - `__init__(self, **kwargs)`: defines all necessary model parameters, variables
  - `forward(self, inputs: torch.Tensor)`: creates computation graph

```
class Net(nn.Module):  
    def __init__(self):  
        super(Net, self).__init__()  
        self.conv1 = nn.Conv2d(3, 6, 5)  
        self.pool = nn.MaxPool2d(2, 2)  
        self.conv2 = nn.Conv2d(6, 16, 5)  
        self.fc1 = nn.Linear(16 * 5 * 5, 120)  
        self.fc2 = nn.Linear(120, 84)  
        self.fc3 = nn.Linear(84, 10)  
  
    def forward(self, x):  
        x = self.pool(F.relu(self.conv1(x)))  
        x = self.pool(F.relu(self.conv2(x)))  
        x = x.view(-1, 16 * 5 * 5)  
        x = F.relu(self.fc1(x))  
        x = F.relu(self.fc2(x))  
        x = self.fc3(x)  
        return x
```

Example from [official tutorial page](#)

```
net = Net()
```

# PyTorch: Loss and Optimizer

- Cross-entropy loss:  $H(\text{data label}, \text{predicted label})$ 
  - Off-the-shelf, inherits torch.nn.Module

```
criterion = nn.CrossEntropyLoss()
```

- Gradient based optimizer (we will talk about it later)
  - Off-the-shelf

```
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
```

# PyTorch: Training

```
for epoch in range(2):  # loop over the dataset multiple times

    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        # get the inputs
        inputs, labels = data

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.item()
        if i % 2000 == 1999:    # print every 2000 mini-batches
            print('[%d, %5d] loss: %.3f' %
                  (epoch + 1, i + 1, running_loss / 2000))
            running_loss = 0.0

print('Finished Training')
```