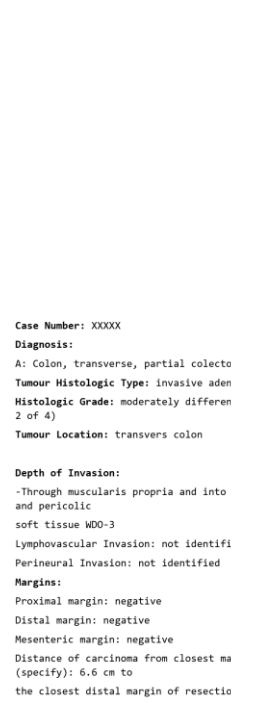
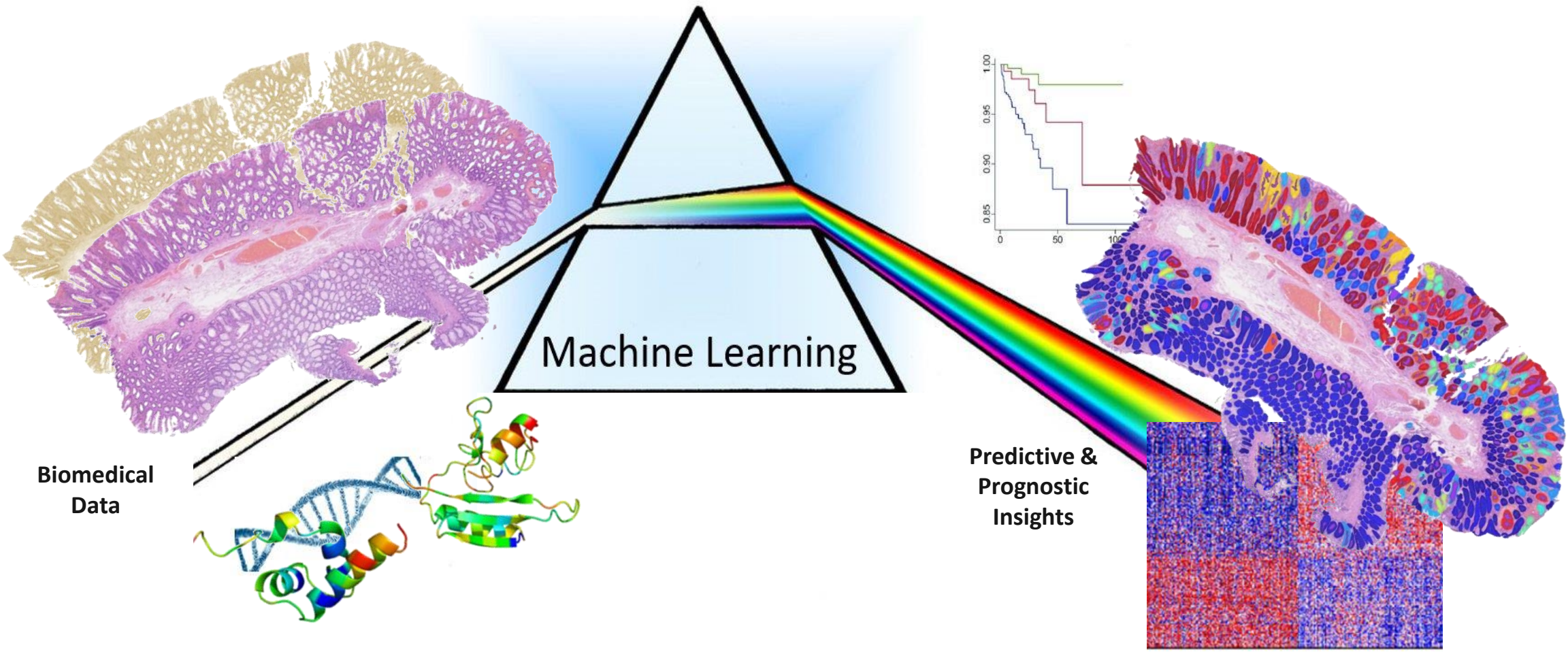
Deep Learning: From philosophy to AGI and applications in biomedicine

Dr. Fayyaz Minhas

Department of Computer Science
University of Warwick
https://warwick.ac.uk/fac/sci/dcs/teaching/material/cs909/

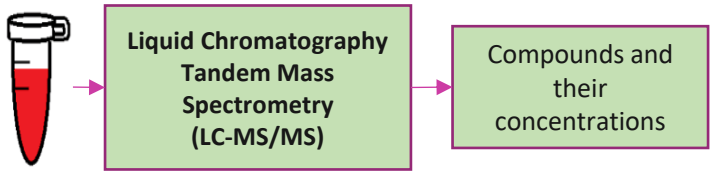




AI and ML to help discover and understand biology and pathology

@fayyazhere

• AI for Biology

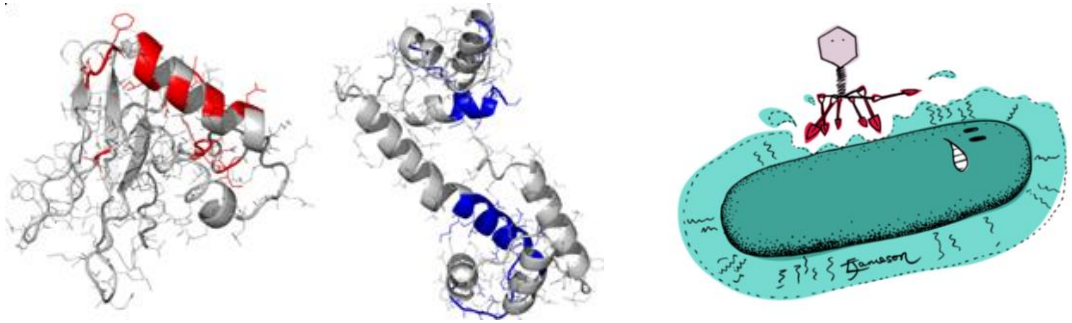


RAMClust: Spectral-matching-based annotation for metabolomics data

Nucleic Acids Research, 2020 **1**
doi: 10.1093/nar/gkaa219

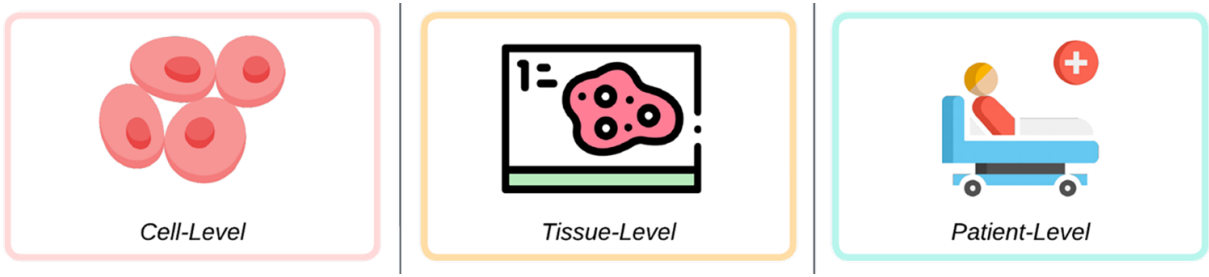
Machine learning predicts new anti-CRISPR proteins

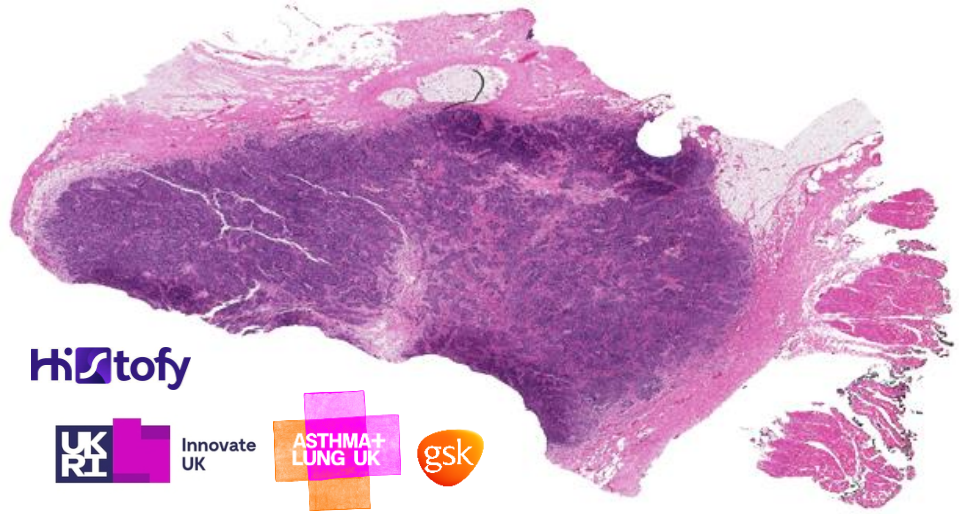
Simon Eitzinger[1,†], Amina Asif[2,3,†], Kyle E. Watters[1,†], Anthony T. Iavarone[4], Gavin J. Knott[1], Jennifer A. Doudna[1,5,6,7,8,9,*] and Fayyaz ul Amir Afsar Minhas[2,10,*]



Prediction of Protein Interfaces, Drug-Protein Interactions and Protein Function (Anti-CRISPR proteins, Anti-Microbial & Hemolytic peptides, Amyloid & Prion activity, Phage antibacterial proteins)

• AI for Pathology & Diagnostics





PathLAKE
Computational Pathology Excellence

CANCER RESEARCH UK

Histofy

UKRI Innovate UK

ASTHMA+ LUNG UK

gsk

EPSRC
Engineering and Physical Sciences Research Council

(Almost) Zero to LLMs and beyond

- What do we need?
  - Some philosophy
  - Some Calculus

Monday, August 12, 2024	
09:00 - 10:30	Day 6 Lecture 1 **Dr. Fayyaz ul Amir Afsar Minhas** The philosophy, principles and intuition of deep learning
1030 – 1100	**Refreshment**
11:00 - 12:30	Day 6 Lecture 2 **Dr. Fayyaz ul Amir Afsar Minhas** Review of foundational algorithms (Distance functions, Kernels, Perceptron's, CNNs, REO Framework and their applications)
12:30 - 13:45	Lunch and Prayer break
1345 - 15:15	Day 6 Lecture 3 **Dr. Fayyaz ul Amir Afsar Minhas** Coding practice of fundamental ML methods (on Colab)

Tuesday, August 13, 2024	
09:00 - 10:30	Day 7 Lecture 1 **Dr. Fayyaz ul Amir Afsar Minhas** From CNNs to Transformers
1030 – 1100	**Refreshment**
11:00 - 12:30	Day 7 Lecture 2 **Dr. Fayyaz ul Amir Afsar Minhas** Basics of Large Language Models
12:30 - 13:45	Lunch and Prayer break
1345 - 15:15	Day 7 Lecture 3 **Dr. Fayyaz ul Amir Afsar Minhas** Lab on CNNs and Transformers (on Colab)
Thursday, August 15, 2024	
09:00 - 10:30	Day 8 Lecture 1 **Dr. Fayyaz ul Amir Afsar Minhas** Building Large Language Models
1030 – 1100	**Refreshment**
11:00 - 12:30	Day 8 Lecture 2 **Dr. Fayyaz ul Amir Afsar Minhas** Generative Machine Learning: GANs and Diffusion
12:30 - 13:45	Lunch and Prayer break
1345 - 15:15	Day 8 Lecture 3 **Dr. Fayyaz ul Amir Afsar Minhas** Building the world's simplest language model
Friday, August 16, 2024	
09:00 - 10:30	Day 2 Lecture 1 **Dr. Fayyaz ul Amir Afsar Minhas** Graph Neural Networks
1030 – 1100	**Refreshment**
11:00 - 12:30	Day 2 Lecture 2 **Dr. Fayyaz ul Amir Afsar Minhas** Towards AGI: Research applications, limitations and future
12:30 - 13:45	Lunch and Prayer break
1345 - 15:15	Certificate Distribution and Closing Ceremony

Lecture 1
The philosophy, principles and intuition of deep learning

Dr. Fayyaz Minhas

Department of Computer Science
University of Warwick
https://warwick.ac.uk/fac/sci/dcs/teaching/material/cs909/

Philosophy

Train PhD students to be thinkers not just specialists

Many doctoral curricula aim to produce narrowly focused researchers rather than critical thinkers. That can and must change, says **Gundula Bosch**.

Under pressure to turn out productive lab members quickly, many PhD programmes in the biomedical sciences have shortened their courses, squeezing out opportunities for putting research into its wider context. Consequently, most PhD curricula are unlikely to nurture the big thinkers and creative problem-solvers that society needs.

That means students are taught every detail of a microbe's life cycle but little about the life scientific. They need to be taught to recognize how errors can occur. Trainees should evaluate case studies derived from flawed real research, or use interdisciplinary detective games to find logical fallacies in the literature. Above all, students must be shown the scientific process as it is — with its limitations and potential pitfalls as well as its fun side, such as serendipitous discoveries and hilarious blunders.

This is exactly the gap that I am trying to fill at Johns Hopkins University in Baltimore, Maryland, where a new graduate science programme is entering its second year. Microbiologist Arturo Casadevall and I began pushing for reform in early 2015, citing the need to put the philosophy back into the doctorate of philosophy: that is, the 'Ph' back into the PhD. We call our programme R3, which means that our students learn to apply rigour to their design and conduct of experiments; view their work through the lens of social responsibility; and to think critically, communicate better, and thus improve reproducibility. Although we are aware of many innovative individual courses developed along these lines, we are striving for more-comprehensive reform.

PUT THE **PHILOSOPHY** BACK INTO THE **DOCTORATE** OF PHILOSOPHY.

Our offerings are different from others at the graduate level. We have critical-thinking assignments in which students analyse errors in reasoning in a *New York Times* opinion piece about 'big sugar', and the ethical implications of the arguments made in a *New Yorker* piece by surgeon Atul Gawande entitled 'The Mistrust of Science'. Our courses on rigorous research, scientific integrity, logic, and mathematical and programming skills are integrated into students' laboratory and fieldwork. Those studying the influenza virus, for example, work with real-life patient data sets and wrestle with the challenges of applied statistics.

A new curriculum starts by winning allies. Both students and faculty members must see value in moving off the standard track. We used informal interviews and focus groups to identify areas in which students and faculty members saw gaps in their training. Recurring themes included the inability to apply theoretical knowledge in statistical tests in the laboratory, frequent mistakes in choosing an appropriate set of experimental controls, and significant difficulty in explaining work to non-experts.

Introducing our programme to colleagues in the Johns Hopkins life-sciences departments was even more sensitive. I was startled by the oft-expressed opinion that scientific productivity depended more on rote knowledge than on competence in critical thinking. Several principal investigators were uneasy about students committing more time to less conventional forms of education. The best way to gain their support was coffee: we repeatedly met lab heads to understand their concerns.

With the pilot so new, we could not provide data on students' performance, but we could address faculty members' scepticism. Some colleagues were apprehensive that students would take fewer courses in specialized content to make room for interdisciplinary courses on ethics, epistemology and quantitative skills. In particular, they worried that the R3 programme could lengthen the time required for students to complete their degree, leave them insufficiently knowledgeable in their subject areas and make them less productive in the lab.

We made the case that better critical thinking and fewer mandatory discipline-specific classes might actually position students to be more productive. We convinced several professors to try the new system and participate in structured evaluations on whether R3 courses contributed to students' performance.

So far, we have built 5 new courses from scratch and have enrolled 85 students from nearly a dozen departments and divisions. The courses cover the anatomy of errors and misconduct in scientific practice and teach students how to dissect the scientific literature. An interdisciplinary discussion series encourages broad and critical thinking about science. Our students learn to consider societal consequences of research advances, such as the ability to genetically alter sperm and eggs.

Discussions about the bigger-picture problems of the scientific enterprise get students to reflect on the limits of science, and where science's ability to do something competes with what scientists should do from a moral point of view. In addition, we have seminars and workshops on professional skills, particularly leadership skills through effective communication, teaching and mentoring.

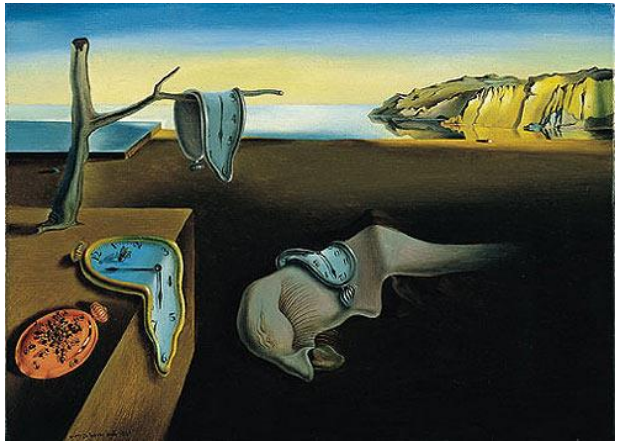
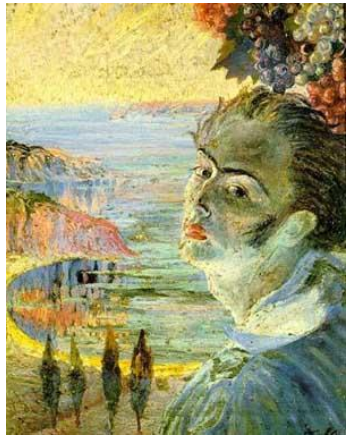
It is still early days for assessment. So far, however, trainees have repeatedly emphasized that gaining a broader perspective has been helpful. In future, we will collect information about the impact that the R3 approach has on graduates' career choices and achievements.

We believe that researchers who are educated more broadly will do science more thoughtfully, with the result that other scientists, and society at large, will be able to rely on this work for a better, more rational world. Science should strive to be self-improving, not just self-correcting. ■

Gundula Bosch *directs the R3 Graduate Science Initiative at Johns Hopkins Bloomberg School of Public Health in Baltimore, Maryland.*
e-mail: gbosch@jhu.edu

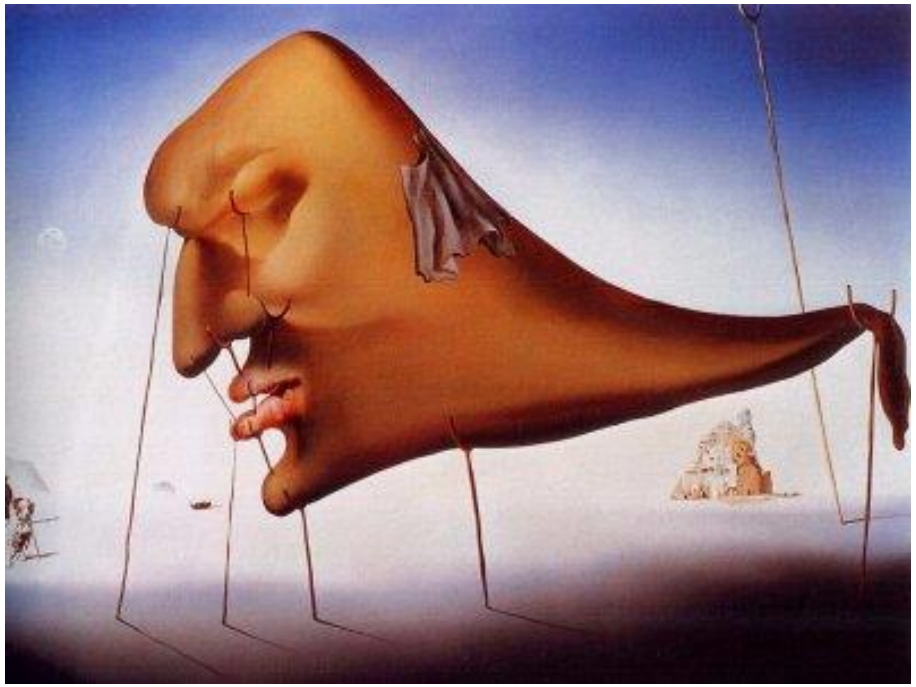
Paintings by two different painters



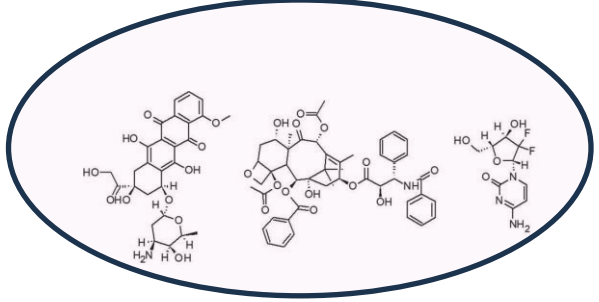





Who’s painting is this?

And this?

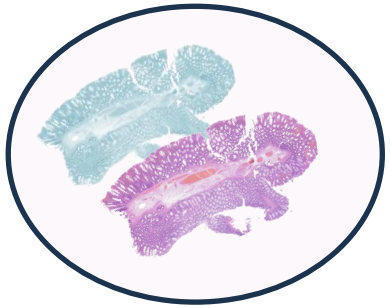


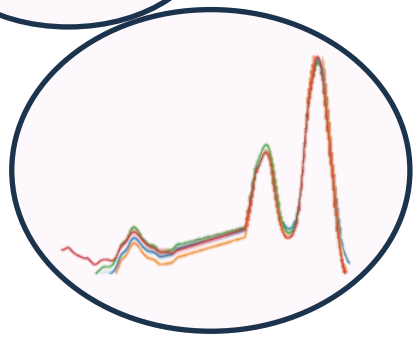
learning from data for generalization to unseen cases

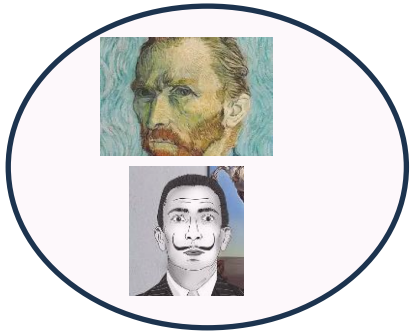


My self

A transformer that can transform into a yellow car







I. Entities have (explicit or implicit) representations

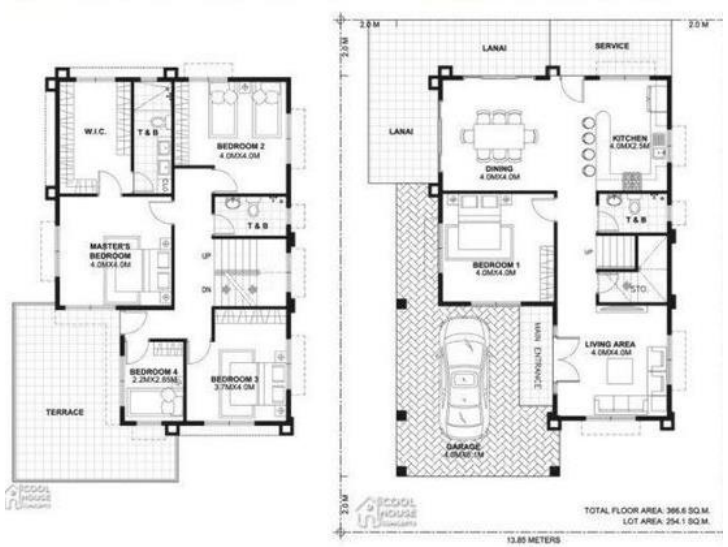
"Bank" in which statement is more semantically related to the picture?

- **A:** As he walked by the **bank**, he saw some boats
- **B:** As he walked by the **bank**, he saw some tellers

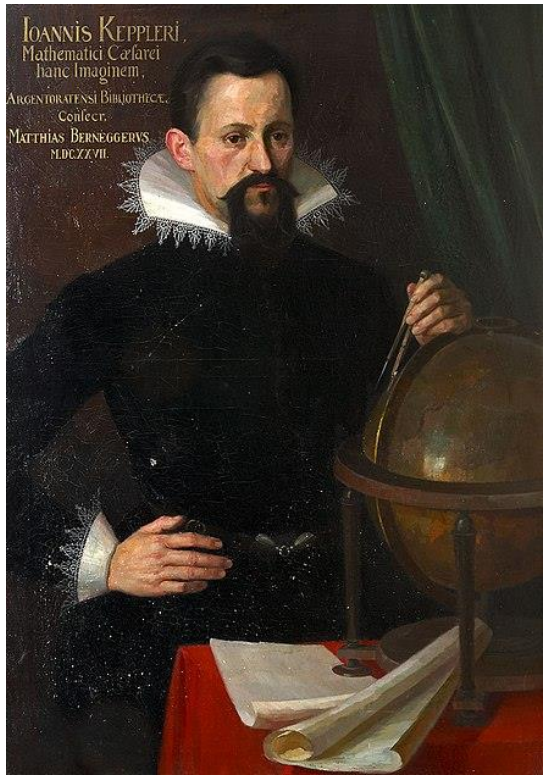
As he walked by the **bank**, he saw some boats

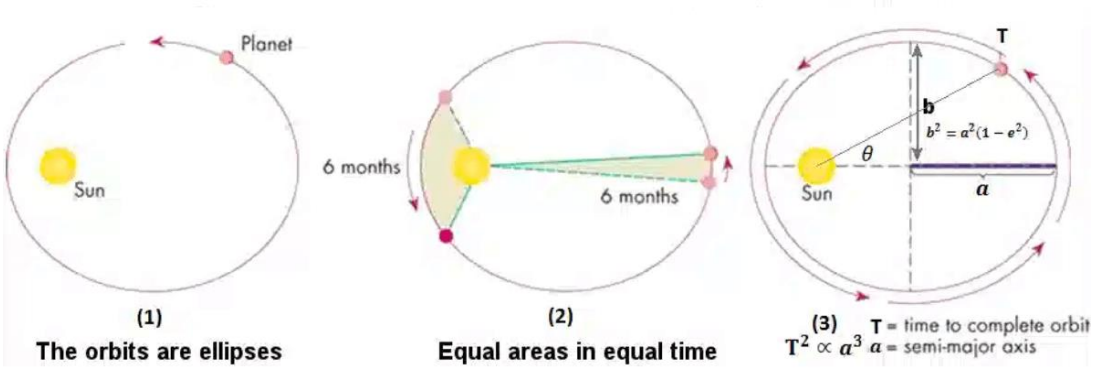
As he walked by the **bank**, he saw some tellers

II. Semantic relatedness of entities is context dependent and thus their representations are contextual



III. Representation of any entity can allow us to reconstruct or "generate" it





IV. It is possible to develop representations in an inductive manner (through empirical observations)

https://en.wikipedia.org/wiki/Johannes_Kepler
https://earthobservatory.nasa.gov/features/OrbitsHistory

US Airways Flight 1549

V. To act effectively and adaptively towards a goal, a being requires developing and using causal representations of entities at an appropriate level of complexity.

Only if we could have a mechanism that would enable developing such representations from empirical observations

Deep Learning

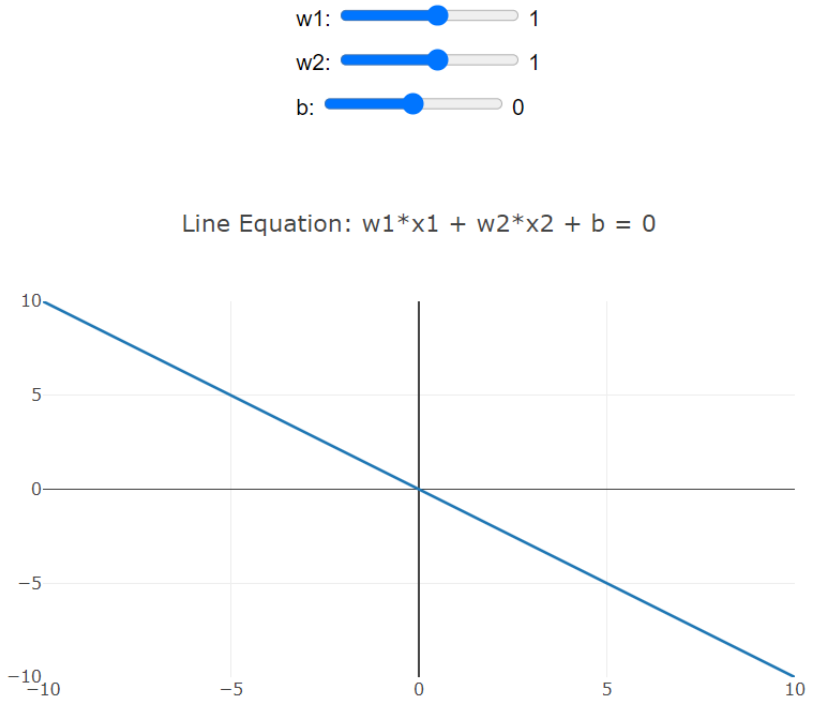
Learning Representations from training examples with “layers” of biologically inspired neurons

Philosophical basis	Algorithms
I. Entities have (explicit or implicit) representations	Feature analysis / Representation learning
II. Semantic relatedness of entities is context dependent and thus their representations are contextual	Using Convolutions, Transformers or Graph Layers
III. Representation of any entity can allow us to reconstruct or “generate” it to a “sufficient”	Generative Machine Learning: GANs, Latent Diffusion Models
IV. It is possible to develop representations in an inductive manner (through empirical observations)	Learning Algorithm: Optimization of model parameters through gradient descent based on existing data Learning mechanisms: Self Supervised Learning, Next word prediction
V. To act effectively and adaptively towards a goal, a being requires developing and using causal representations of entities at an appropriate level of complexity.	Deep Reinforcement Learning? Structural risk minimization (controls the model complexity and hence complexity of representations it learns)

Preliminaries and Intuition

Preliminaries

- Equations of lines and their properties



$$w_1 x_1 + w_2 x_2 + b = 0$$

$$x_2 = \frac{-w_1}{w_2} x_1 + \frac{-b}{w_2}$$

$$x_2 = m x_1 + c$$

$$f(\boldsymbol{x}; \boldsymbol{w}) = w_1 x_1 + w_2 x_2 + b = 0$$

$$f(\boldsymbol{x}; \boldsymbol{w}) = [w_0 \quad w_1 \quad w_2] \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix} = \boldsymbol{w}^T \boldsymbol{x} = 0$$

If a point $x = (x_1, x_2)$ is on the line then $f(x; \boldsymbol{w}) = w_1 x_1 + w_2 x_2 + b = 0$
If it is above the line then $f(x; \boldsymbol{w}) = w_1 x_1 + w_2 x_2 + b > 0$
If it is below the line then $f(x; \boldsymbol{w}) = w_1 x_1 + w_2 x_2 + b < 0$

https://foxtrotmike.github.io/CS909/lines.html

Preliminaries

- Distance function

$$d(\boldsymbol{a}, \boldsymbol{b}) = \|\boldsymbol{a} - \boldsymbol{b}\| = \sqrt{(\boldsymbol{a_1} - \boldsymbol{b_1})^2 + (\boldsymbol{a_2} - \boldsymbol{b_2})^2}$$

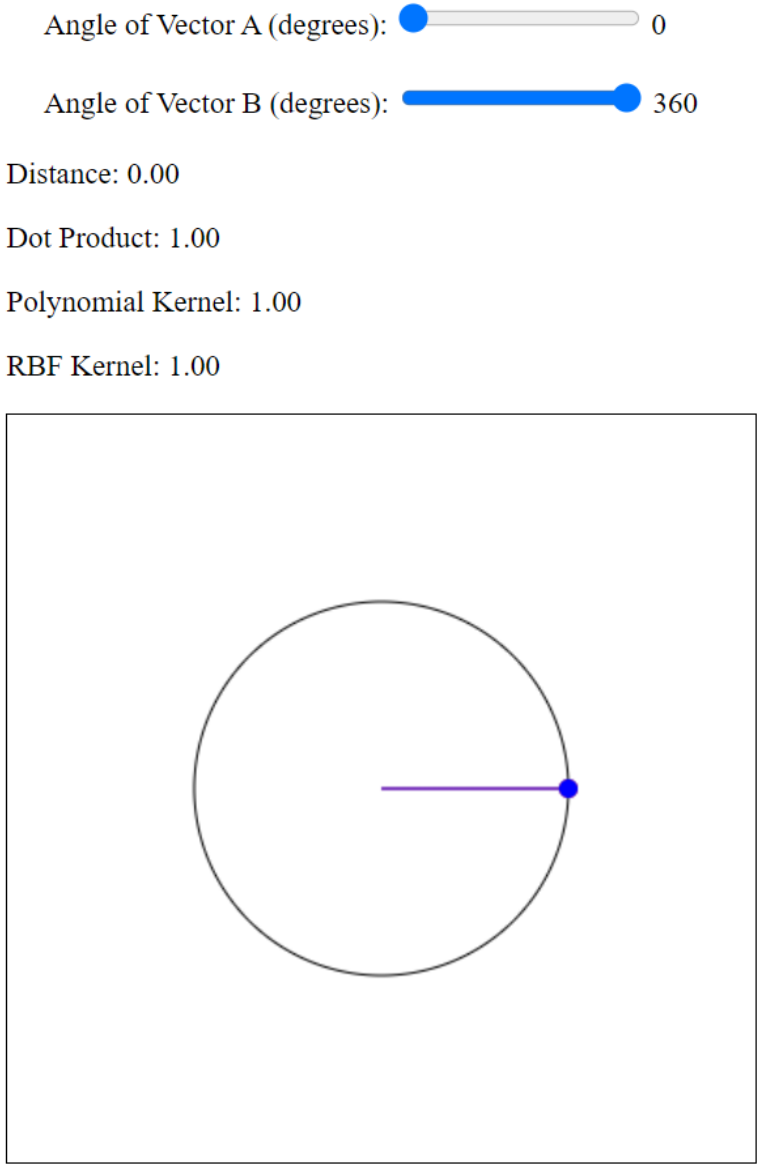
- Relation with Dot Product

$$d^2(\boldsymbol{a}, \boldsymbol{b}) = \|\boldsymbol{a} - \boldsymbol{b}\|^2 = (\boldsymbol{a_1} - \boldsymbol{b_1})^2 + (\boldsymbol{a_2} - \boldsymbol{b_2})^2 = \boldsymbol{a^T a} + \boldsymbol{b^T b} - \boldsymbol{2 a^T b}$$

- If $\boldsymbol{a}, \boldsymbol{b}$ are unit vectors then: $d^2(\boldsymbol{a}, \boldsymbol{b}) = 2 - 2(\boldsymbol{a^T b})$

- **Or the farther away or different two points are, the lower their dot product and vice-versa**

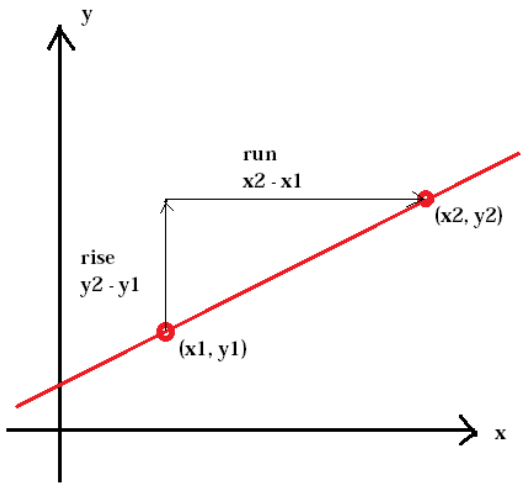
- **We can also have more generalized dot products called kernel functions that can measure similarity between two objects in a different way**
  - Linear kernel: $k(\boldsymbol{a}, \boldsymbol{b}) = \boldsymbol{a}^T \boldsymbol{b}$
  - Polynomial kernel: $k(\boldsymbol{a}, \boldsymbol{b}) = \left(\boldsymbol{a}^T \boldsymbol{b}\right)^2$
  - Gaussian or Radial Basis Function (RBF) Kernel: $k(\boldsymbol{a}, \boldsymbol{b}) = \exp\left(-\lambda \|\boldsymbol{a} - \boldsymbol{b}\|^2\right)$
  - Mahalanobis Kernel: $k(\boldsymbol{a}, \boldsymbol{b}; \boldsymbol{M}) = \exp\left(-(\boldsymbol{a} - \boldsymbol{b})^T \boldsymbol{M} (\boldsymbol{a} - \boldsymbol{b})\right)$
  - Exponential kernel: $\mathrm{k}(\boldsymbol{a}, \boldsymbol{b}; \boldsymbol{W_a}, \boldsymbol{W_b}) = exp\left(\frac{1}{\sqrt{d}} \langle \boldsymbol{a W_a}, \boldsymbol{b W_b} \rangle\right)$

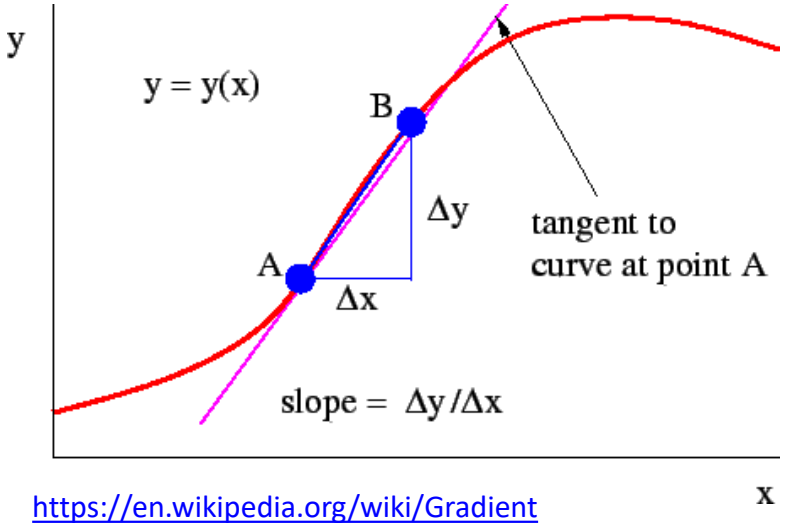


https://foxtrotmike.github.io/CS909/distance_dot.html

Preliminaries

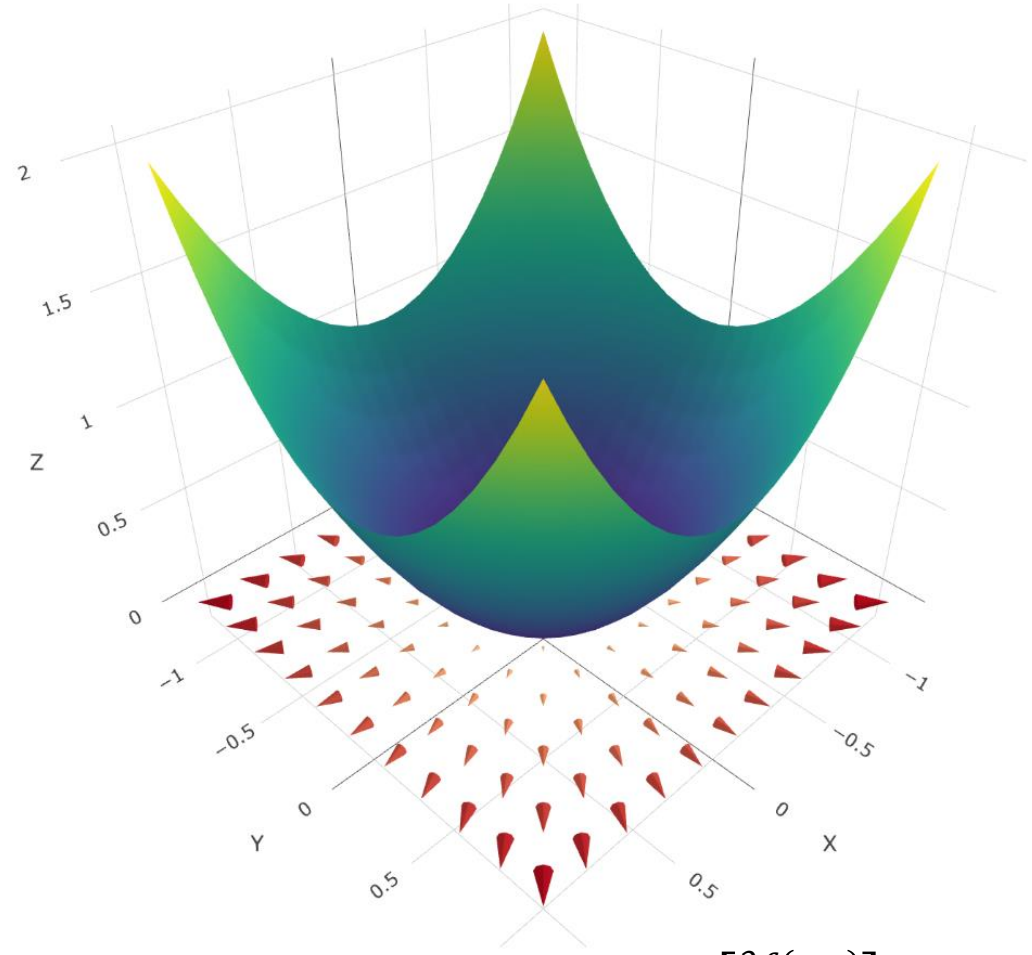
- Gradients





https://en.wikipedia.org/wiki/Gradient

$$\nabla f(\boldsymbol{x}) = \begin{bmatrix} \frac{\partial f(\boldsymbol{x})}{\partial x^{(1)}} \\ \frac{\partial f(\boldsymbol{x})}{\partial x^{(2)}} \\ \vdots \\ \frac{\partial f(\boldsymbol{x})}{\partial x^{(d)}} \end{bmatrix}$$



$$f(x,y) = x^2 + y^2 \text{ and } \nabla f(x,y) = \begin{bmatrix} \frac{\partial f(x,y)}{\partial x} \\ \frac{\partial f(x,y)}{\partial y} \end{bmatrix} = \begin{bmatrix} 2x \\ 2y \end{bmatrix}$$

https://foxtrotmike.github.io/CS909/gradients.html

Preliminaries: Finding minima and maxima of functions

- Given a function f(w)
- Take the derivative
- Substitute the derivative to zero
- Solve for x when $\frac{df}{dw} = 0$
- Works when we can solve for w

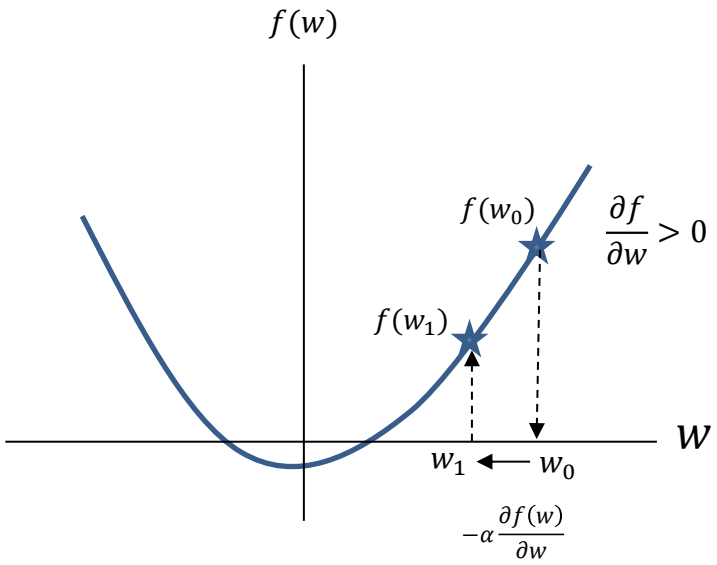
$$f(w) = (w - 0.5)^2$$
$$\frac{df}{dw} = 2(w - 0.5) = 0$$
$$w^* = 0.5$$

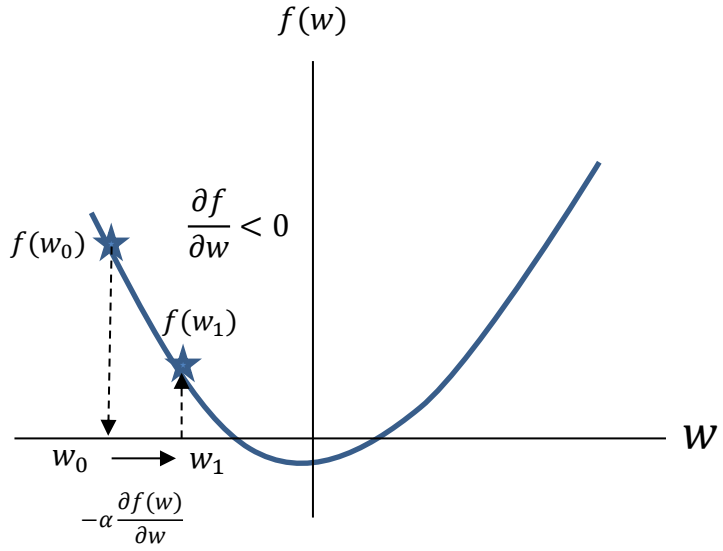
$$f(w) = (w - 0.5)^2 + sin(4w)$$
$$\frac{df}{dw} = 2(w - 0.5) + 4\cos(4w) = 0$$
$$w^* = ?$$

Preliminaries: Gradient Descent

- In order to find the minima of a function, keep taking steps along a direction opposite to the gradient of the function

$$\boldsymbol{w}^{(\boldsymbol{k+1})} = \boldsymbol{w}^{(\boldsymbol{k})} - \alpha \nabla f(\boldsymbol{w}^{(\boldsymbol{k})})$$





GD Implementation

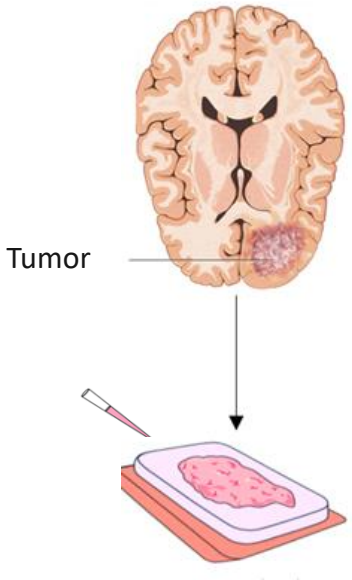
```
import numpy as np

def gd(fxn,dfxn,w0=0.0,lr = 0.01,eps=1e-4,nmax=1000, history = True):
    """
    Implementation of a gradient descent solver.
        fxn: function returns value of the target function for a given w
        dfxn: gradient function returns the gradient of fxn at w
        w0: initial position [Default 0.0]
        lr: learning rate [0.001]
        eps: min step size threshold [1e-4]
        nmax: maximum number of iters [1000]
        history: whether to store history of x or not [True]
    Returns:
        w: argmin_x f(w)
        converged: True if the final step size is less than eps else false
        H: history
    """
    H = []
    w = w0
    if history:
        H = [[w,fxn(w)]]
    for i in range(nmax):
        dw = -lr*dfxn(w) #gradient step
        if np.linalg.norm(dw)<eps: # we have converged
            break
        if history:
            H.append([w+dw,fxn(w+dw)])
        w = w+dw #gradient update
    converged = np.linalg.norm(dw)<eps
    return w,converged,np.array(H)
```

```
if __name__=='__main__':
    import matplotlib.pyplot as plt
    def myfunction(w):
        z = (w-0.5)**2#+np.sin(4*w)
        return z
    def mygradient(w):
        dz = 2*(w-0.5)#+4*np.cos(4*w)
        return dz


    wrange = np.linspace(-3,3,100)
    #select random initial point in the range
    w0 = np.min(wrange)+(np.max(wrange)-np.min(wrange))*np.random.rand()

    w,c,H = gd(myfunction,mygradient,w0=w0,lr = 0.01,eps=1e-4,nmax=1000, history = True)

    plt.plot(wrange,myfunction(wrange)); plt.plot(wrange,mygradient(wrange));
    plt.legend(['f(w)','df(w)'])
    plt.xlabel('w');plt.ylabel('value')
    s = 'Convergence in '+str(len(H))+' steps'
    if not c:
        s = 'No '+s
    plt.title(s)
    plt.plot(H[0,0],H[0,1],'ko',markersize=10)
    plt.plot(H[:,0],H[:,1],'r.-')
    plt.plot(H[-1,0],H[-1,1],'k*',markersize=10)
    plt.grid(); plt.show()
```

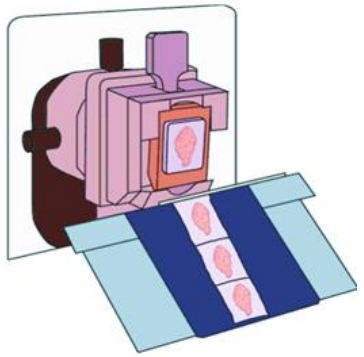
https://github.com/foxtrotmike/CS909/blob/master/gd.py
https://github.com/foxtrotmike/CS909/blob/master/dm_lab_2_fm.ipynb

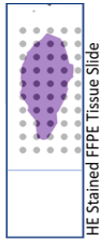
Application: Conventional Histopathology

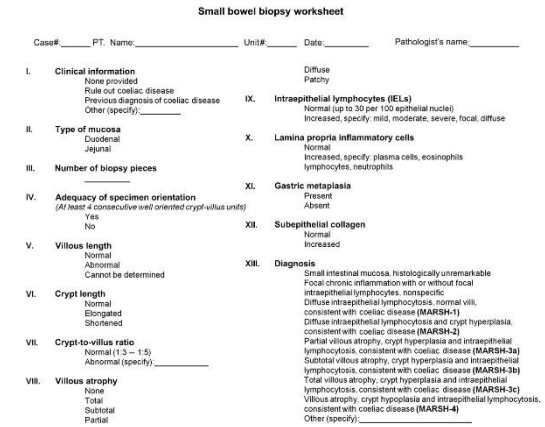
Tissue Acquisition → Slide Preparation → Conventional Microscopy → Clinical Decision Making



Block preparation
Addition of dyes







Small bowel biopsy worksheet

Case#:_____ PT. Name:_____ Unit#:_____ Date:_____ Pathologist's name:_____

I. **Clinical information**
None provided
Rule out coeliac disease
Previous diagnosis of coeliac disease
Other (specify):_____

II. **Type of mucosa**
Duodenal
Jejunal

III. **Number of biopsy pieces**
_____

IV. **Adequacy of specimen orientation**
(At least 4 consecutive well oriented crypt-villus units)
Yes
No

V. **Villous length**
Normal
Abnormal
Cannot be determined

VI. **Crypt length**
Normal
Elongated
Shortened

VII. **Crypt-to-villus ratio**
Normal (1:3 – 1:5)
Abnormal (specify):_____

VIII. **Villous atrophy**
None
Total
Subtotal
Partial
Diffuse
Patchy

IX. **Intraepithelial lymphocytes (IELs)**
Normal (up to 30 per 100 epithelial nuclei)
Increased, specify: mild, moderate, severe, focal, diffuse

X. **Lamina propria inflammatory cells**
Normal
Increased, specify: plasma cells, eosinophils
lymphocytes, neutrophils

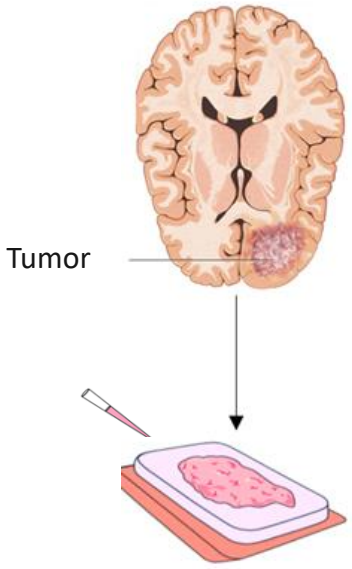
XI. **Gastric metaplasia**
Present
Absent

XII. **Subepithelial collagen**
Normal
Increased

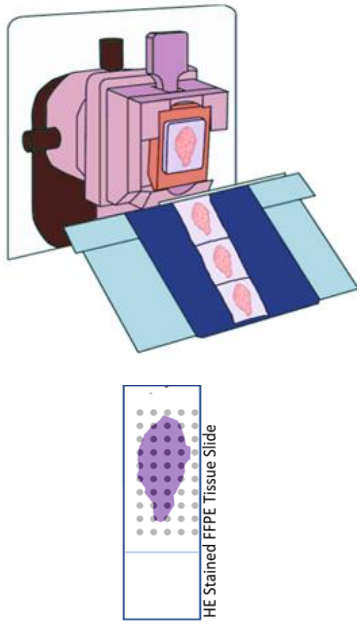
XIII. **Diagnosis**
Small intestinal mucosa, histologically unremarkable
Focal chronic inflammation with or without focal intraepithelial lymphocytes, nonspecific
Diffuse intraepithelial lymphocytosis, normal villi, consistent with coeliac disease **(MARSH-1)**
Diffuse intraepithelial lymphocytosis and crypt hyperplasia, consistent with coeliac disease **(MARSH-2)**
Partial villous atrophy, crypt hyperplasia and intraepithelial lymphocytosis, consistent with coeliac disease **(MARSH-3a)**
Subtotal villous atrophy, crypt hyperplasia and intraepithelial lymphocytosis, consistent with coeliac disease **(MARSH-3b)**
Total villous atrophy, crypt hyperplasia and intraepithelial lymphocytosis, consistent with coeliac disease **(MARSH-3c)**
Villous atrophy, crypt hypoplasia and intraepithelial lymphocytosis, consistent with coeliac disease **(MARSH-4)**
Other (specify):_____

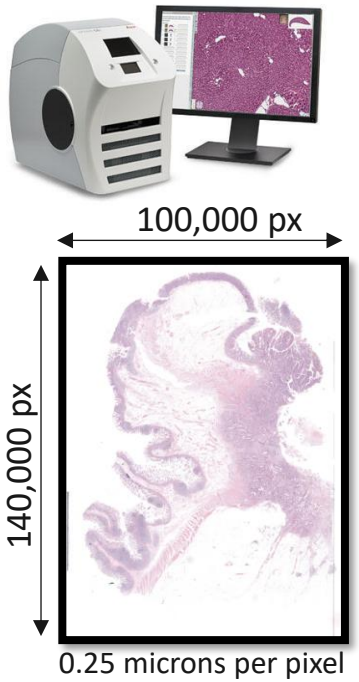
Application: Digital Pathology

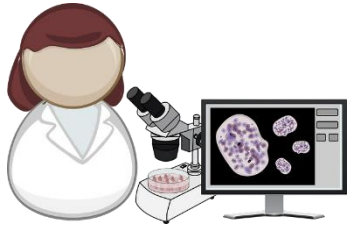
Tissue Acquisition → Slide Preparation → Whole Slide Scanning → Clinical Decision Making



Block preparation
Addition of dyes





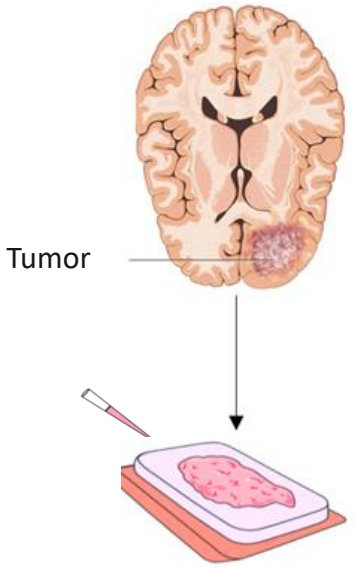


- Flexible working
- Good concordance with slide based clinical decision making based on equivalence studies
- However: *digitization of glass slides alone does not resolve the pressures of an increasing workload on a diminishing workforce of pathologists*

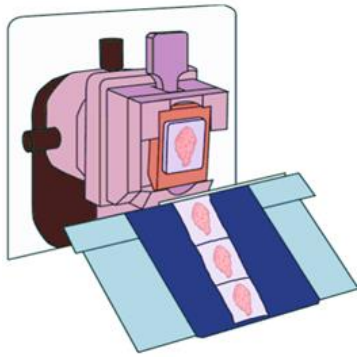
Example equivalence studies: Snead, David R. J., Yee-Wah Tsang, Aisha Meskiri, Peter K. Kimani, Richard Crossman, Nasir M. Rajpoot, Elaine Blessing, et al. "Validation of Digital Pathology Imaging for Primary Histopathological Diagnosis." *Histopathology* 68, no. 7 (June 2016): 1063–72. https://doi.org/10.1111/his.12879.
Hanna, Matthew G., Victor E. Reuter, Meera R. Hameed, Lee K. Tan, Sarah Chiang, Carlie Sigel, Travis Hollmann, et al. "Whole Slide Imaging Equivalency and Efficiency Study: Experience at a Large Academic Center." Modern Pathology 32, no. 7 (July 2019): 916–28. https://doi.org/10.1038/s41379-019-0205-0.

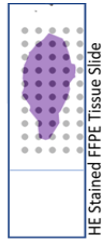
Application: Computational Pathology

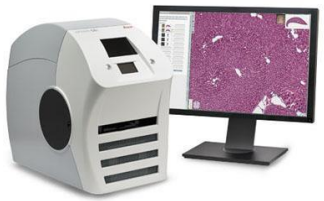
Tissue Acquisition → Slide Preparation → Whole Slide Scanning → **AI Assisted Clinical Decision Making**

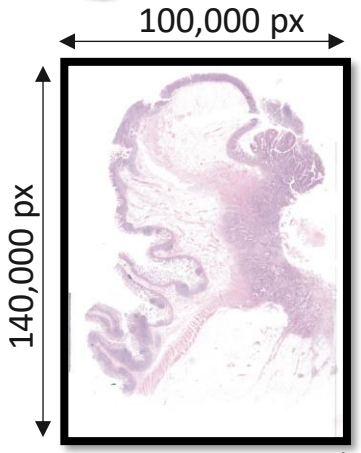


Block preparation
Addition of dyes

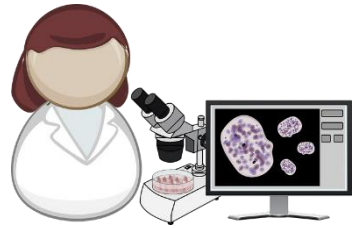




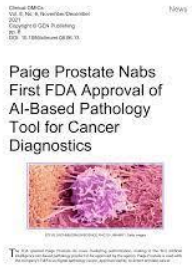




0.25 microns per pixel

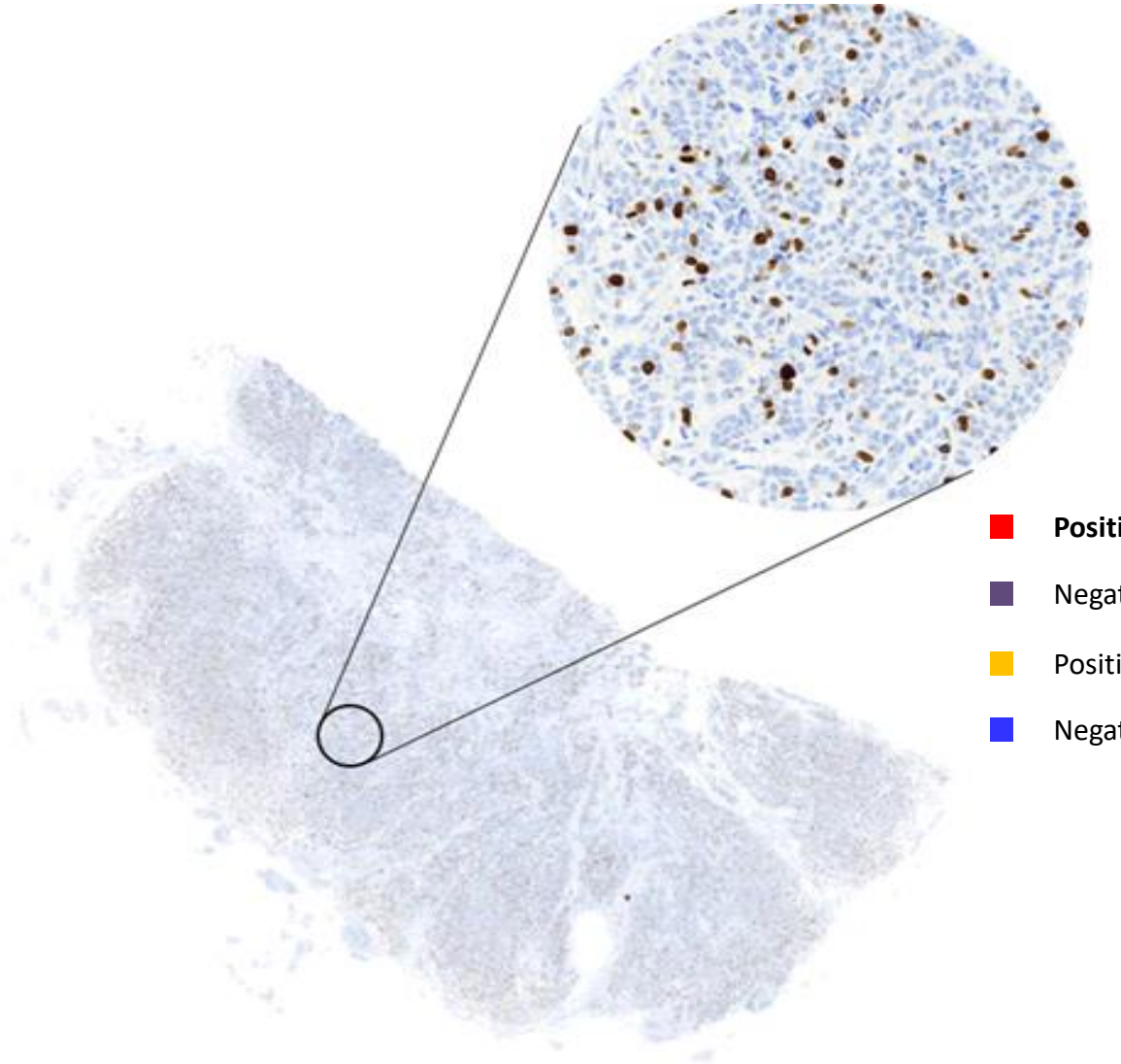


https://info.paige.ai/prostate

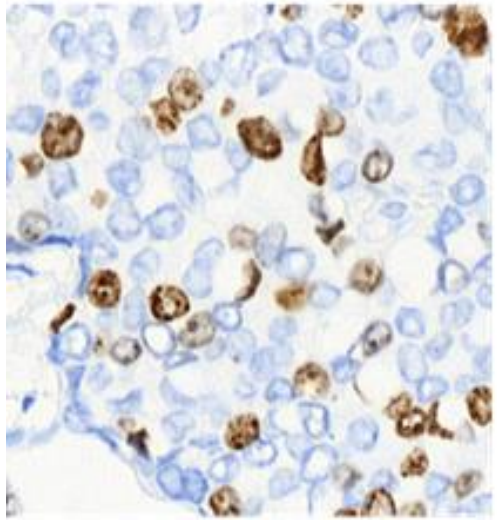


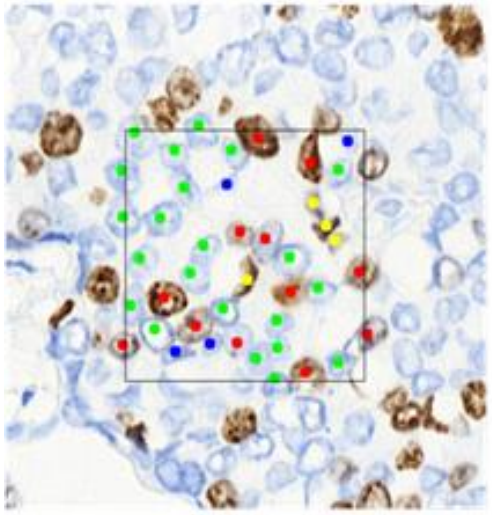
Example Independent validation study of PAIGE Prostate: Kanan, Christopher, Jillian Sue, Leo Grady, Thomas J. Fuchs, Sarat Chandarlapaty, Jorge S. Reis-Filho, Paulo G O Salles, Leonard Medeiros da Silva, Carlos Gil Ferreira, and Emilio Marcelo Pereira. "Independent Validation of Paige Prostate: Assessing Clinical Benefit of an Artificial Intelligence Tool within a Digital Diagnostic Pathology Laboratory Workflow." Journal of Clinical Oncology 38, no. 15_suppl (May 20, 2020): e14076–e14076. https://doi.org/10.1200/JCO.2020.38.15_suppl.e14076.

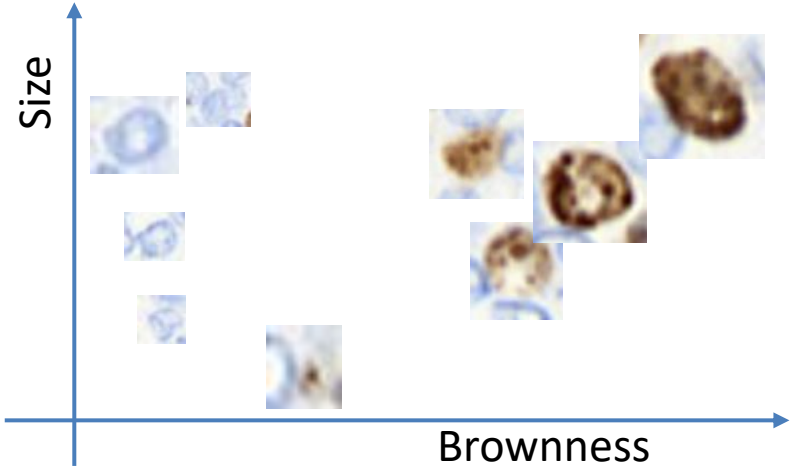
How does ML work?



- **Positive Tumor Cell**
- Negative Tumor Cell
- Positive Non-Tumor Cell
- Negative Non-Tumor Cell

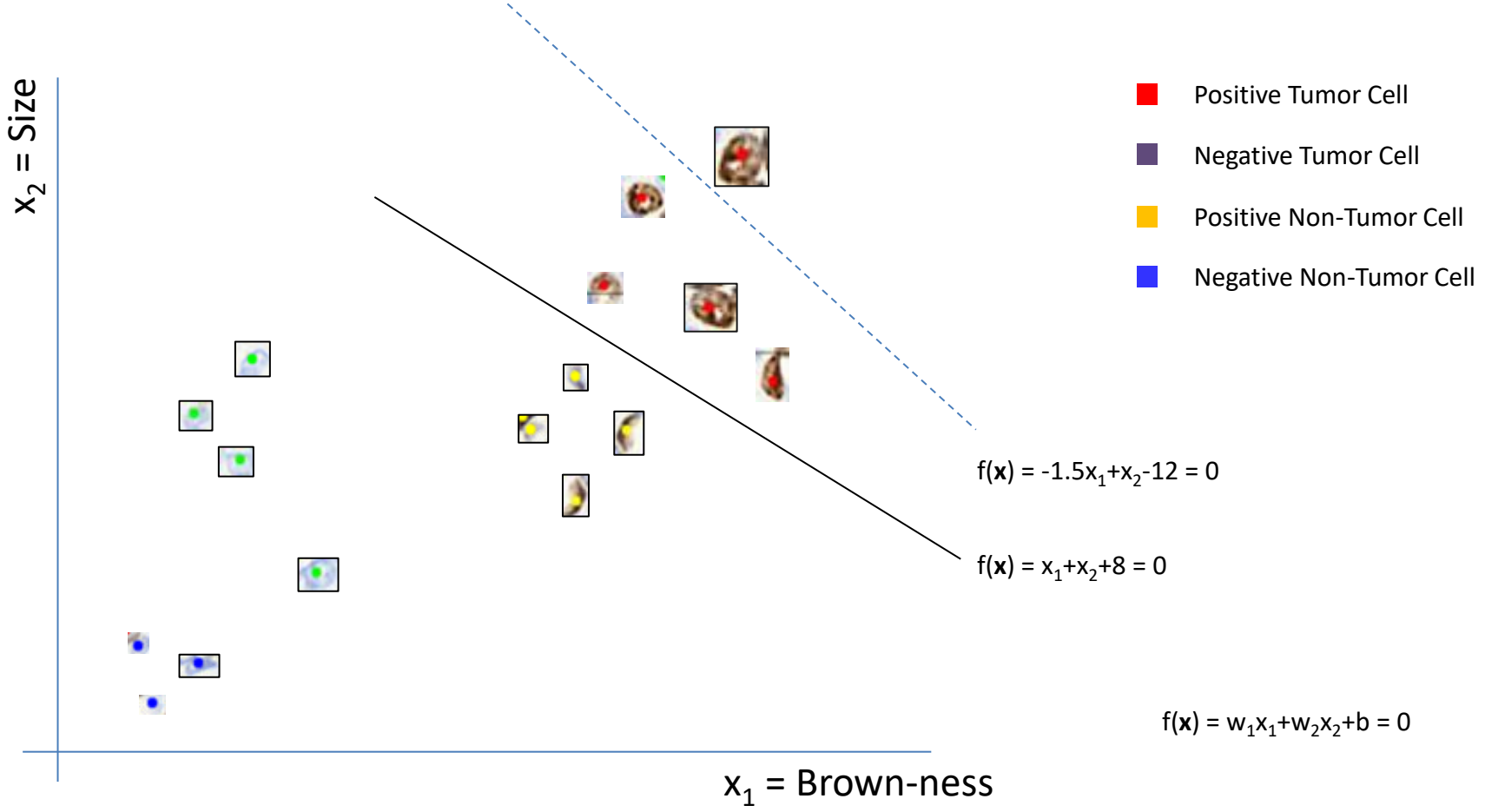






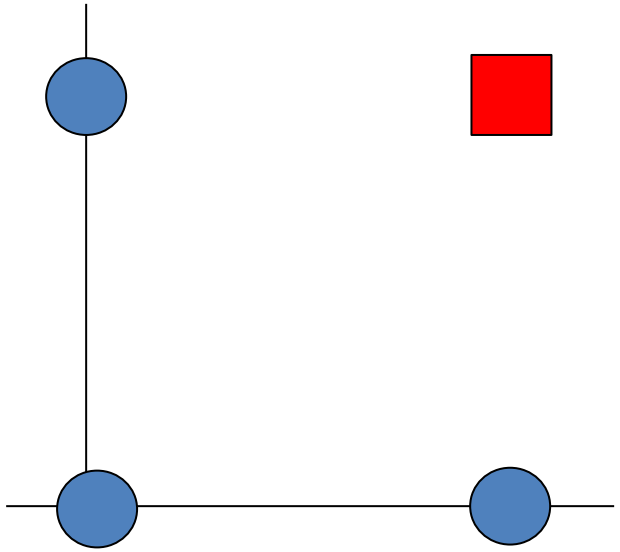
Wenqi Lu, Islam M Miligy, Fayyaz Minhas, Young Saeng Park, David R J Snead, Emad A Rakha, Clare Verrill, Nasir Rajpoot “Lessons from a Breast Cell Annotation Competition Series for School Pupils.” Scientific Reports, 2022. https://ora.ox.ac.uk/objects/uuid:9e34d4e6-c677-4380-9403-759808b349aa.

Machine Learning in Cpath with simple lines

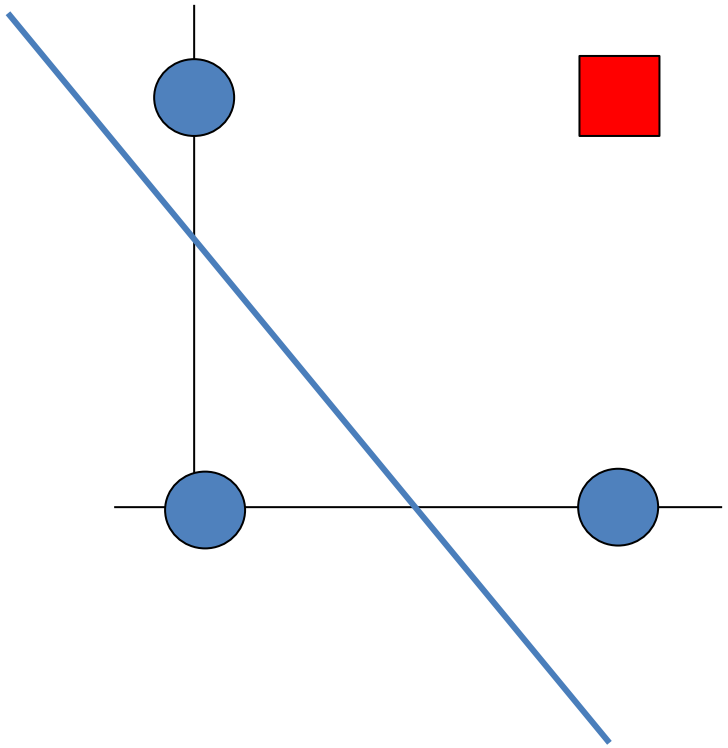


Goal is to be able to generalize to unseen data
With Deep Learning – we don't even need to define features!

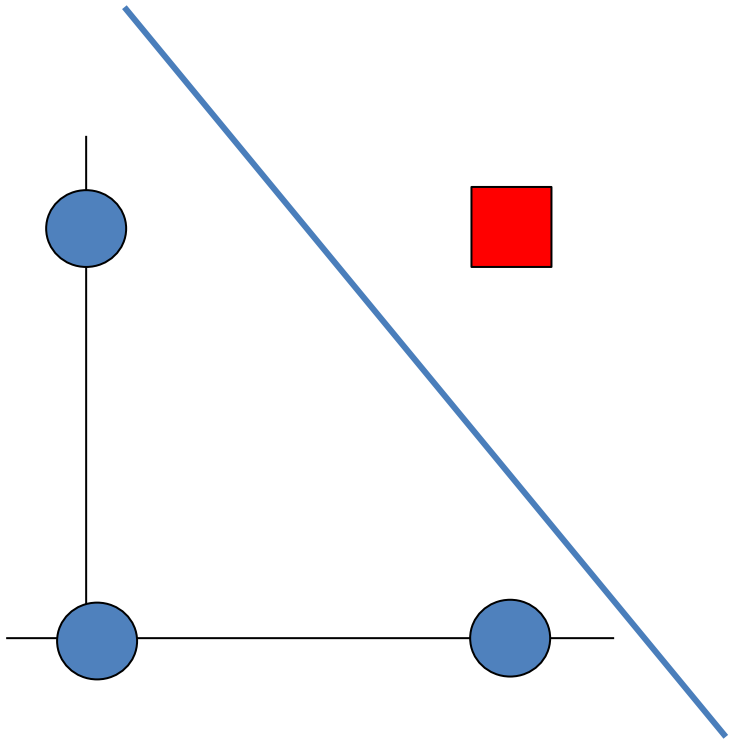
Exercise



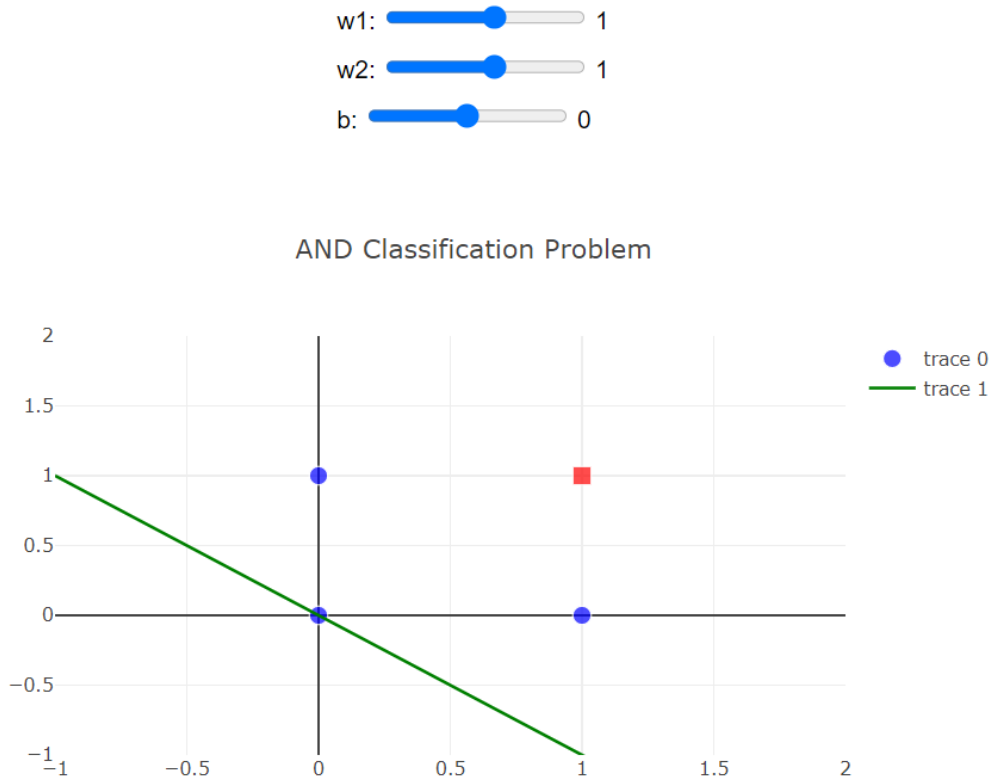
Exercise



Exercise

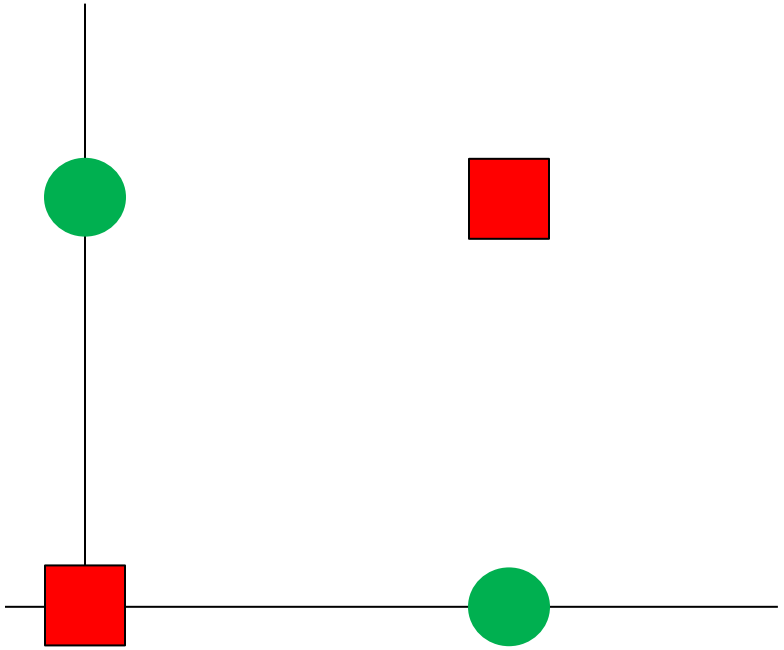


Doing it interactively



https://foxtrotmike.github.io/CS909/AND-NEURON.html

But what if we have a linearly inseparable problem?



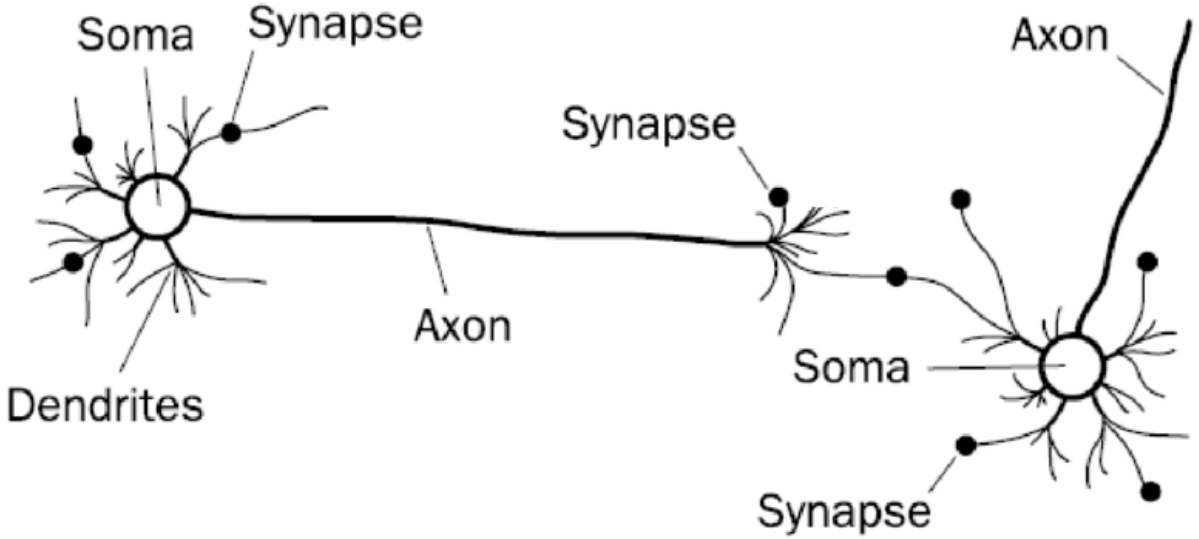
Machine learning and deep learning involve discovering meaningful representations of input data and then using these representations to partition data for various tasks

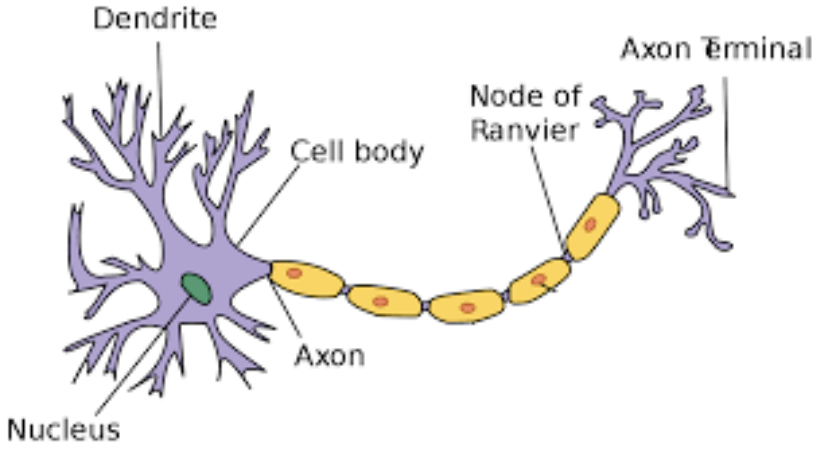
From single neurons to convolutional neural networks

A crash course

Understanding the foundational structure of all machine learning models

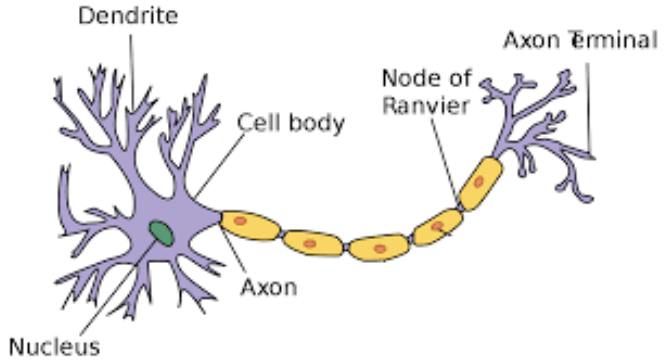
Biological Neurons and Networks

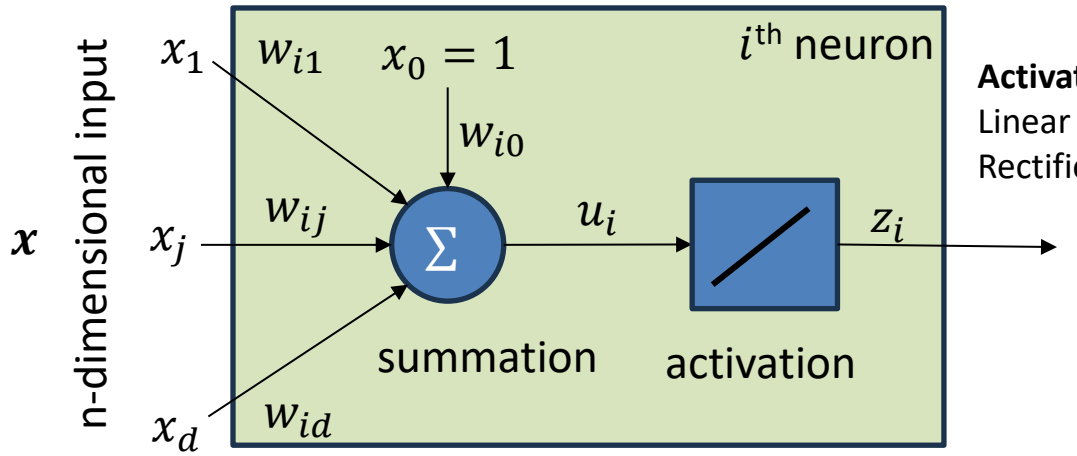


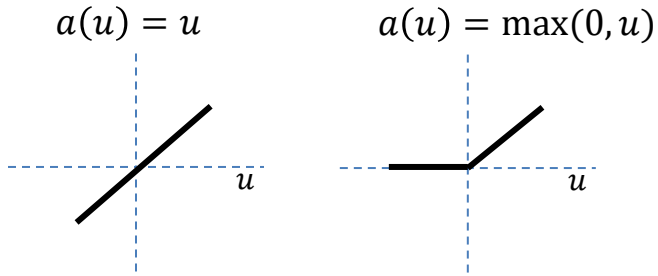


Single Neuron

- An abstraction of the biological neuron







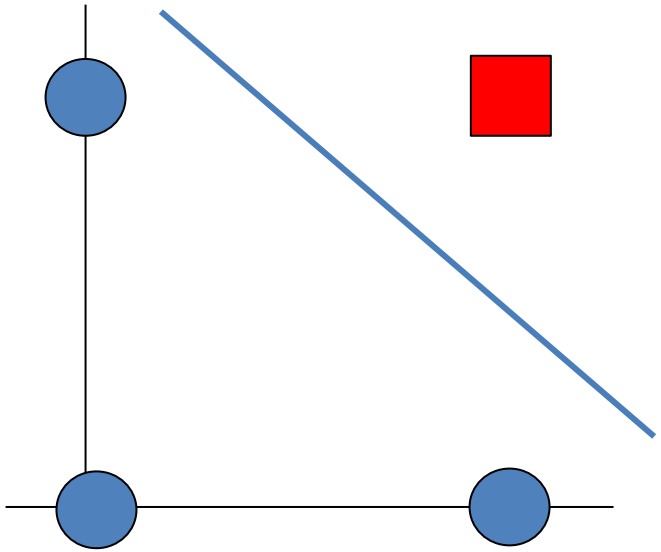
Activation Functions

Linear activation: $a(u) = u$

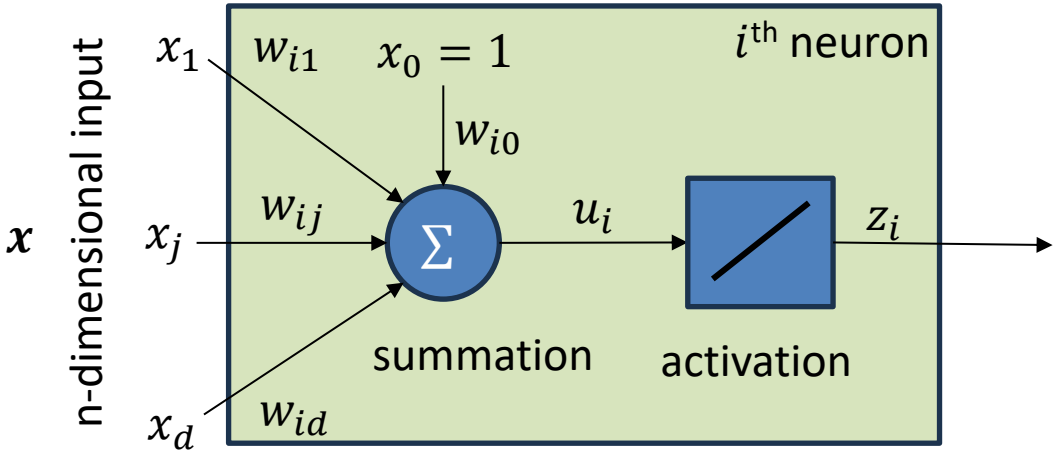
Rectified Linear Unit activation: $a(u) = ReLU(u) = \max(0, a)$

$$z_i = a(u_i) = a\left(\sum_j w_{ij} x_j\right)$$

$$u_i = w_{i1}x_1 + w_{i2}x_2 + \cdots + w_{ij}x_j + \cdots + w_{id}x_d + w_{i0}(1) = \sum_j w_{ij}x_j = \boldsymbol{w_i^T x}$$

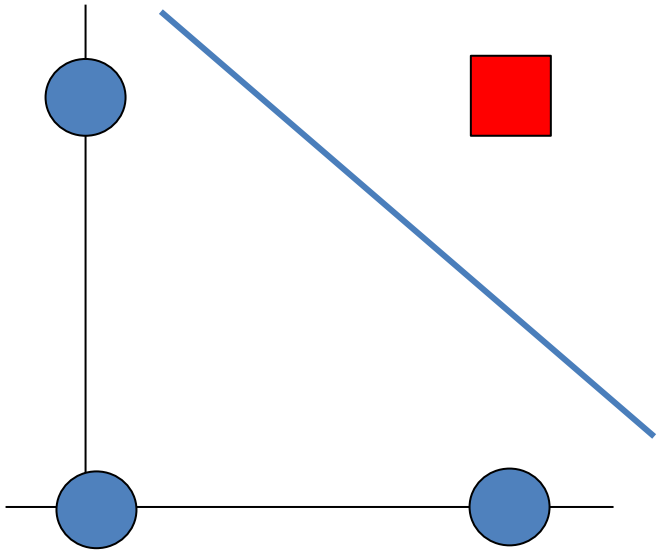


- Requirements for binary classification
  - If a training example $\boldsymbol{x}$ has a label of $y = +1, z = f(\boldsymbol{x})$ should be ____
  - If a training example $\boldsymbol{x}$ has a label of $y = -1, z = f(\boldsymbol{x})$ should be ____

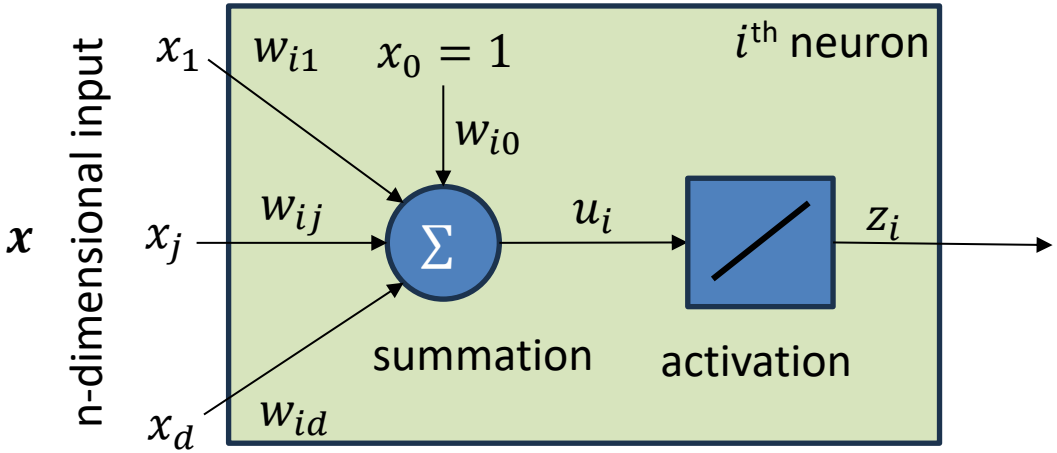


$$z_i = f(\boldsymbol{x_i}; \boldsymbol{w}) = a(u_i) = a\left(\sum_j w_{ij} x_j\right)$$

$$u_i = \sum_j w_{ij} x_j = \boldsymbol{w_i^T x}$$

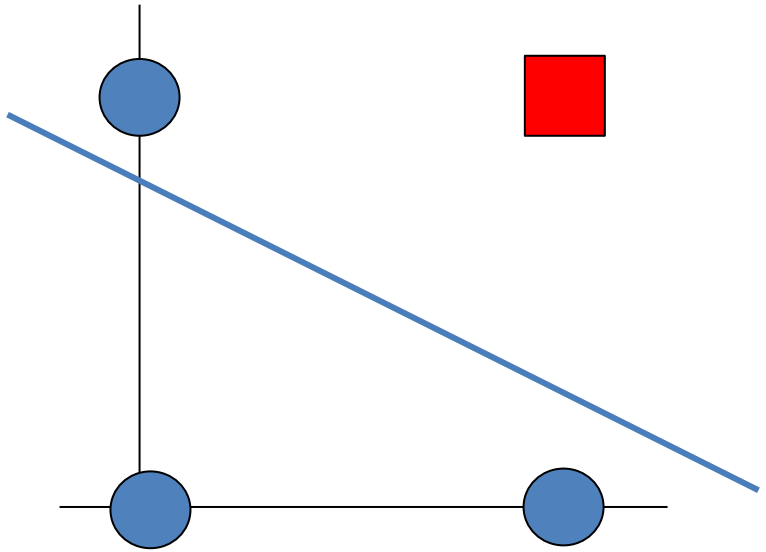


- Requirements for binary classification
  - If a training example $\boldsymbol{x}$ has a label of $y = +1$, $z = f(\boldsymbol{x})$ should be $> \mathbf{0}$
  - If a training example $\boldsymbol{x}$ has a label of $y = -1$, $z = f(\boldsymbol{x})$ should be $< \mathbf{0}$

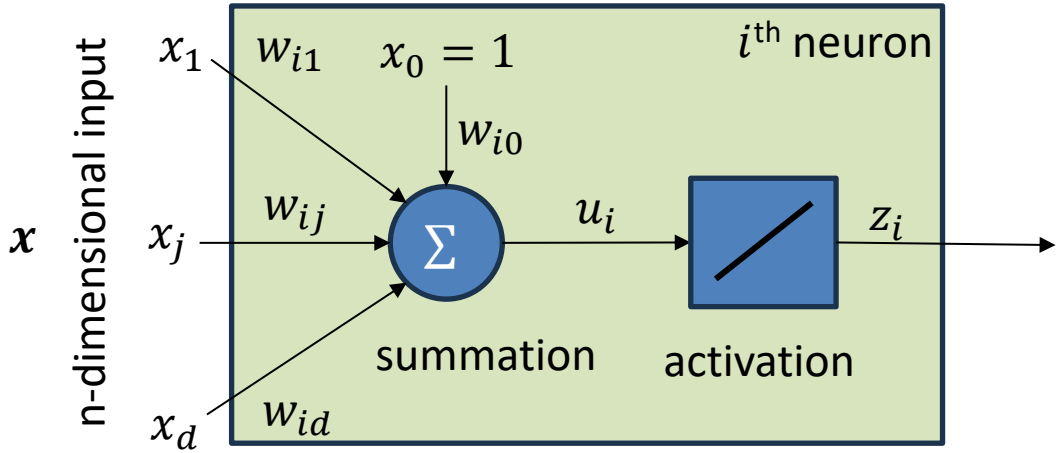


$$z_i = f(\boldsymbol{x_i}; \boldsymbol{w}) = a(u_i) = a\left(\sum_j w_{ij} x_j\right)$$

$$u_i = \sum_j w_{ij} x_j = \boldsymbol{w_i^T x}$$

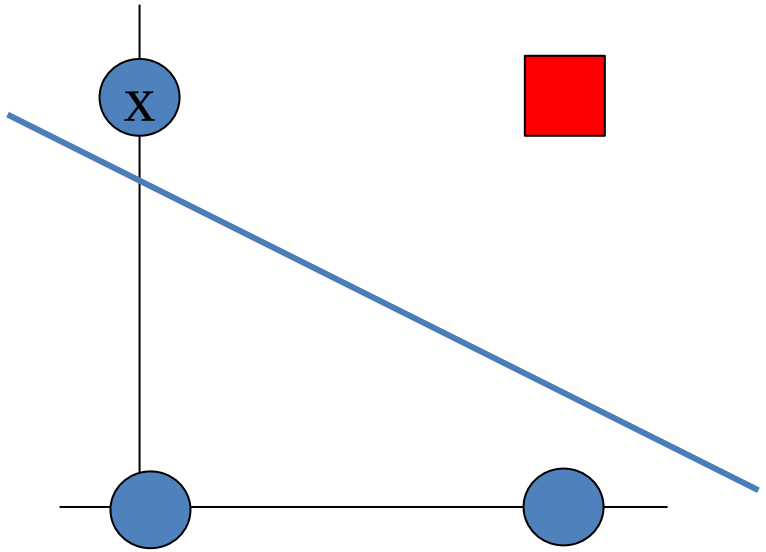


- Defining error:
  - If a training example $\boldsymbol{x}$ has a label of $y = +1, z = f(\boldsymbol{x}; \boldsymbol{w})$ should be $> \mathbf{0}$
  - If a training example $\boldsymbol{x}$ has a label of $y = -1, z = f(\boldsymbol{x}; \boldsymbol{w})$ should be $< \mathbf{0}$
  - **Thus, whenever $yf(\boldsymbol{x}; \boldsymbol{w}) < 0$, we have an error**



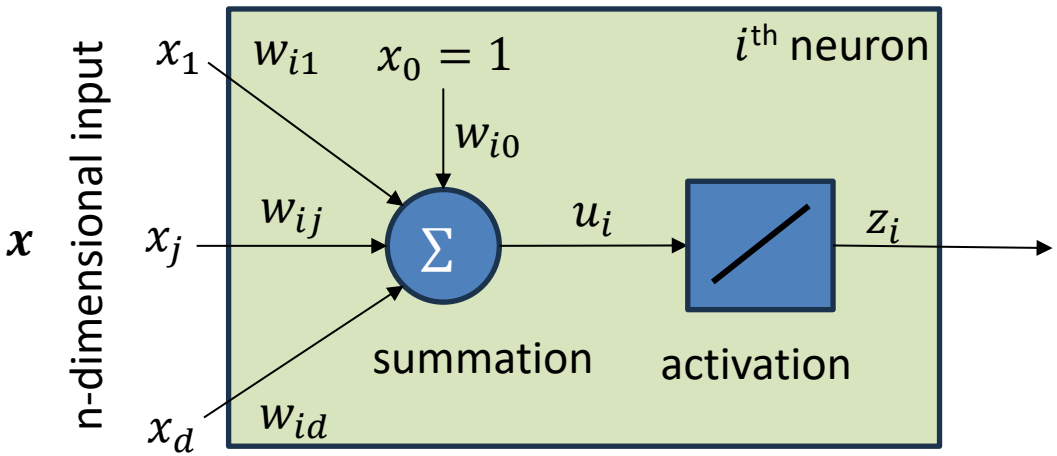
$$z_i = f(\boldsymbol{x_i}; \boldsymbol{w}) = a(u_i) = a\left(\sum_j w_{ij} x_j\right)$$

$$u_i = \sum_j w_{ij} x_j = \boldsymbol{w_i^T x}$$



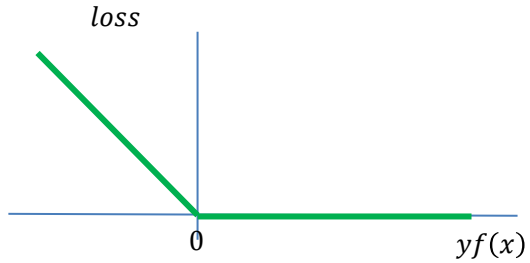
- Quantifying error:
  - If a training example $\boldsymbol{x}$ has a label of $y = +1, z = f(\boldsymbol{x}; \boldsymbol{w})$ should be $> \mathbf{0}$
  - If a training example $\boldsymbol{x}$ has a label of $y = -1, z = f(\boldsymbol{x}; \boldsymbol{w})$ should be $< \mathbf{0}$
  - **Thus, whenever $yf(\boldsymbol{x}; \boldsymbol{w}) < 0$, we have an error**
  - **The amount of error (loss) is thus**

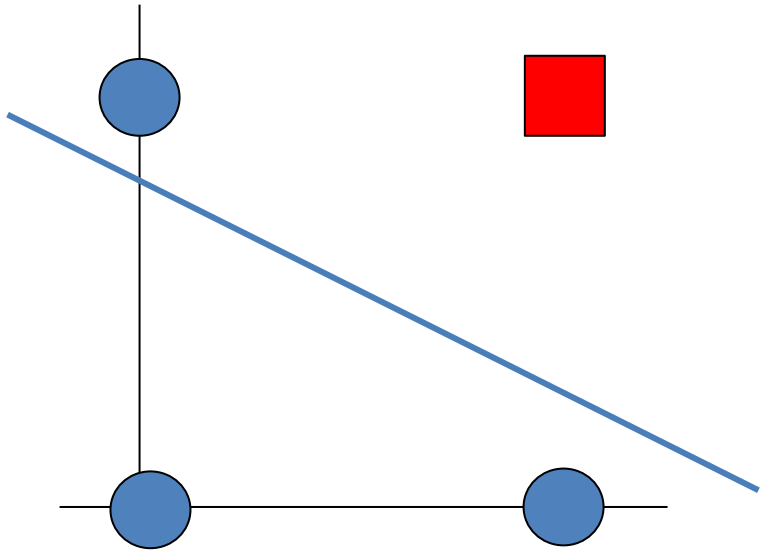
$$l(x; w) = \begin{cases} 0 & if\ yf(\boldsymbol{x}; \boldsymbol{w}) > 0 \\ -yf(\boldsymbol{x}; \boldsymbol{w}) & else \end{cases} = max(0, -yf(\boldsymbol{x}; \boldsymbol{w}))$$

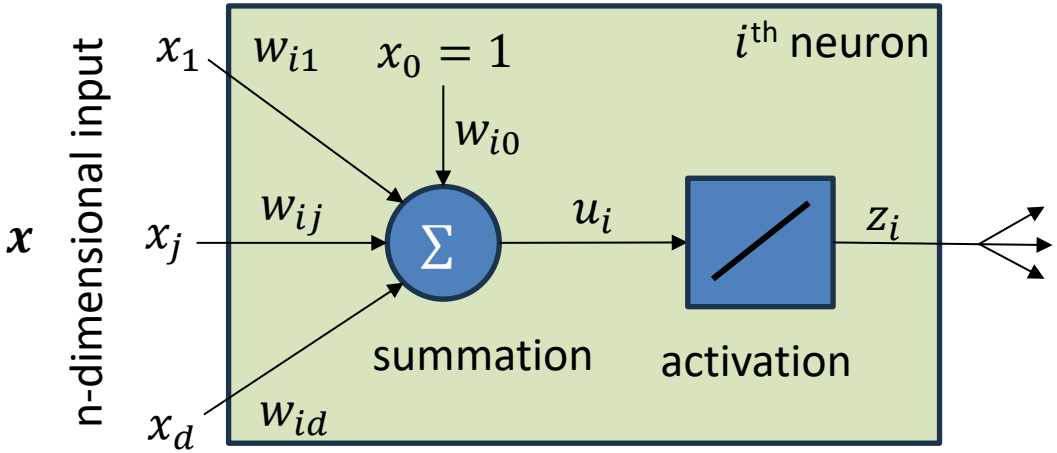


$$z_i = f(\boldsymbol{x_i}; \boldsymbol{w}) = a(u_i) = a\left(\sum_j w_{ij} x_j\right)$$

$$u_i = \sum_j w_{ij} x_j = \boldsymbol{w_i^T x}$$







$$u_i = \sum_j w_{ij} x_j = \boldsymbol{w}_i^T \boldsymbol{x}$$

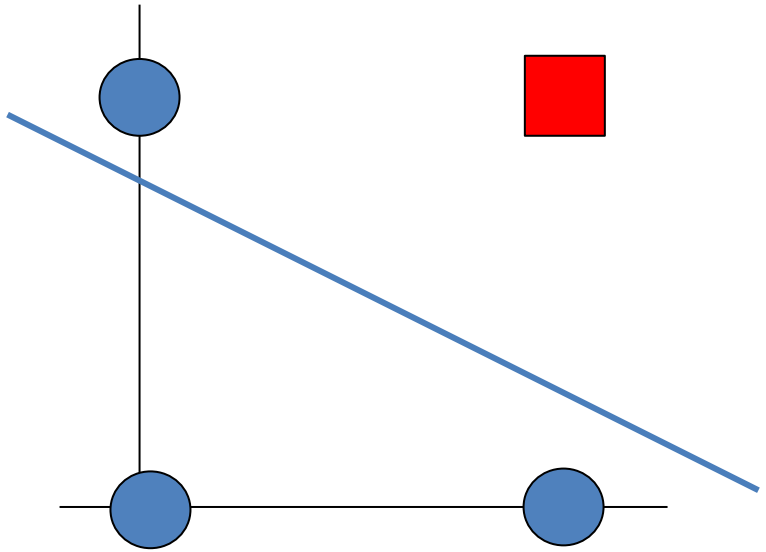
- Quantifying error:
  - **The amount of error (loss) is**

$$l(x; w) = max\big(0, -yf(\boldsymbol{x}; \boldsymbol{w})\big)$$

For all examples, the error is:

$$L(\boldsymbol{w}; \boldsymbol{X}) \overset{iid}{=} \sum_i l(x_i; \boldsymbol{w})$$

$$z_i = f(\boldsymbol{x_i}; \boldsymbol{w}) = a(u_i) = a\left(\sum_j w_{ij} x_j\right)$$

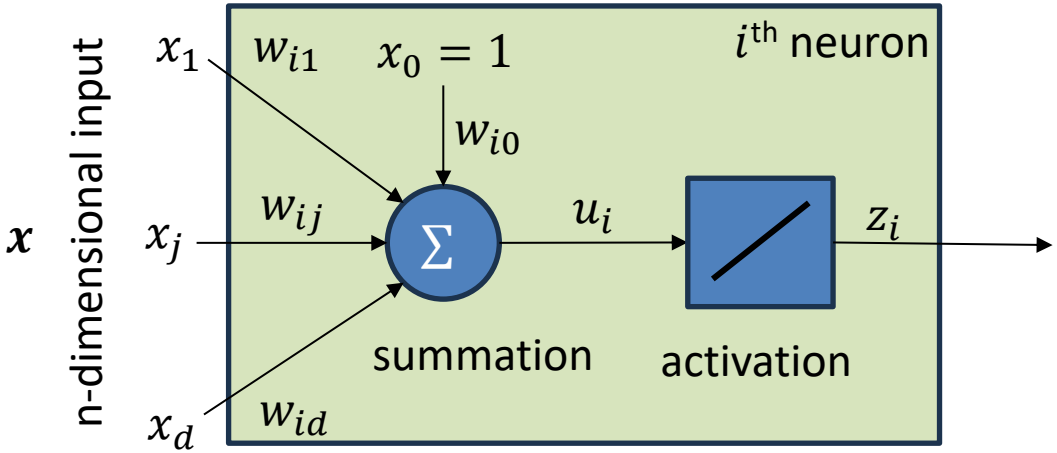


- Quantifying error:

  For all examples, the error is:

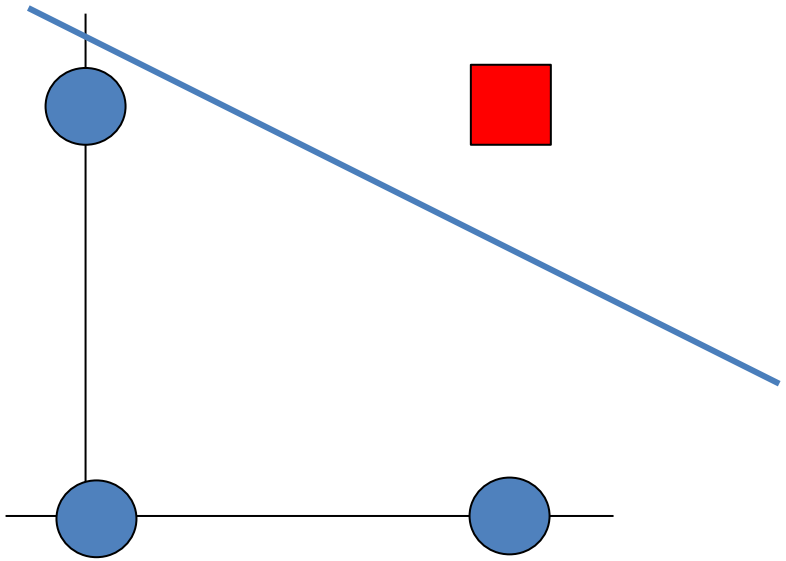
$$L(\boldsymbol{w}; \boldsymbol{X}) = \sum_i l(x_i; \boldsymbol{w})$$

We want to find such $\boldsymbol{w}$ so that $L(\boldsymbol{w}; \boldsymbol{X})$ is minimal



$$z_i = f(\boldsymbol{x_i}; \boldsymbol{w}) = a(u_i) = a\left(\sum_j w_{ij} x_j\right)$$

$$u_i = \sum_j w_{ij} x_j = \boldsymbol{w_i^T x}$$



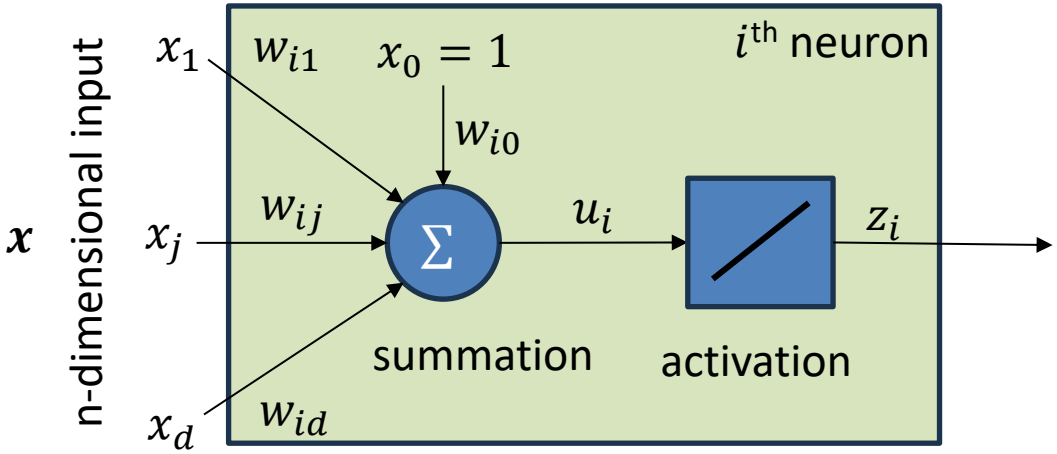
- Optimization:

  For all examples, the error is:

$$L(\boldsymbol{w}; \boldsymbol{X}) = \sum_i l(x_i; \boldsymbol{w})$$

  We want to find such $\boldsymbol{w}$ so that $L(\boldsymbol{w}; \boldsymbol{X})$ is minimal

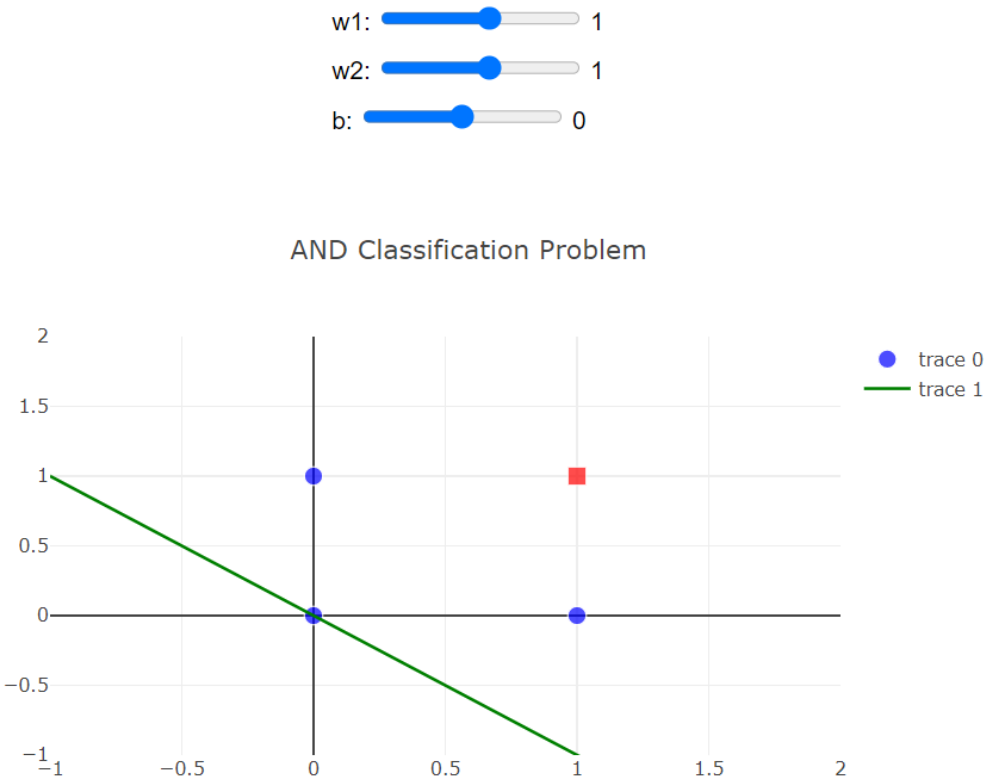
$$\min_{\boldsymbol{w}} L(\boldsymbol{w}; \boldsymbol{X})$$

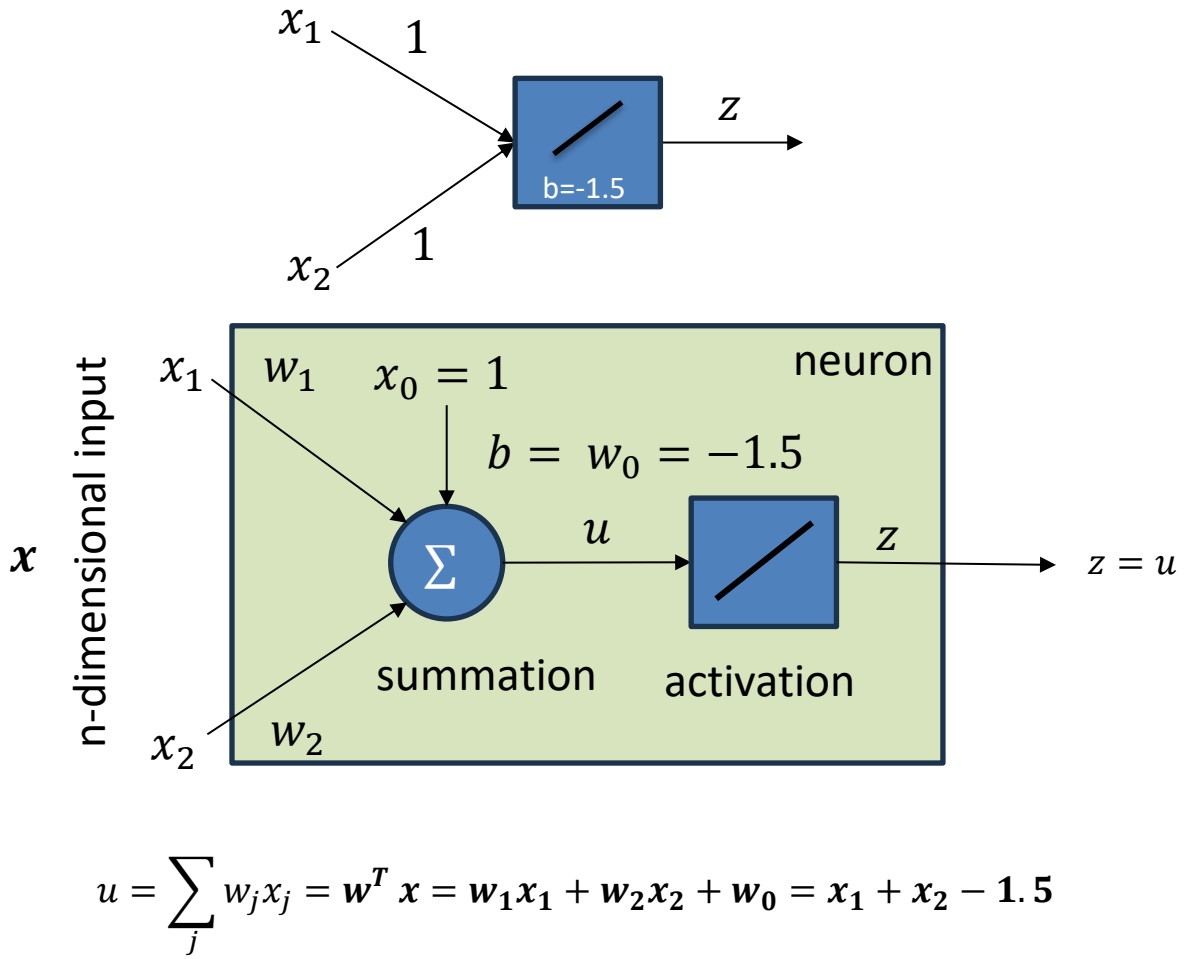


$$z_i = f(\boldsymbol{x_i}; \boldsymbol{w}) = a(u_i) = a\left(\sum_j w_{ij} x_j\right)$$

$$u_i = \sum_j w_{ij} x_j = \boldsymbol{w_i^T x}$$

Doing it interactively



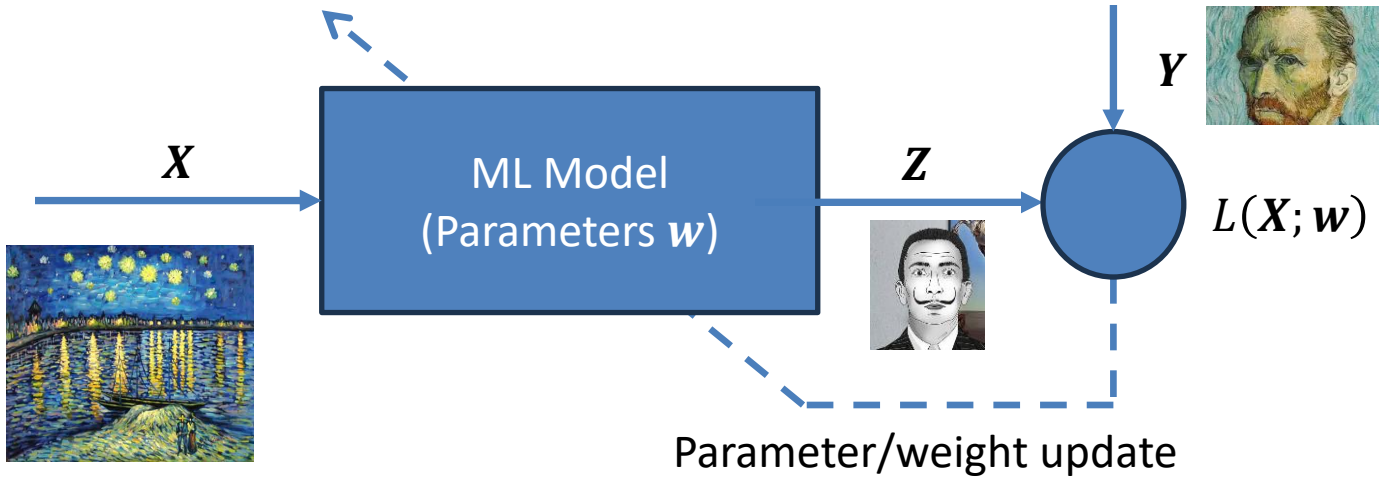


$$u = \sum_j w_j x_j = \boldsymbol{w}^T \boldsymbol{x} = \boldsymbol{w_1 x_1 + w_2 x_2 + w_0 = x_1 + x_2 - 1.5}$$

https://foxtrotmike.github.io/CS909/AND-NEURON.html

Optimization underpins Learning

$$\min_{\boldsymbol{w}} L(\boldsymbol{X}; \boldsymbol{w})$$



Optimization

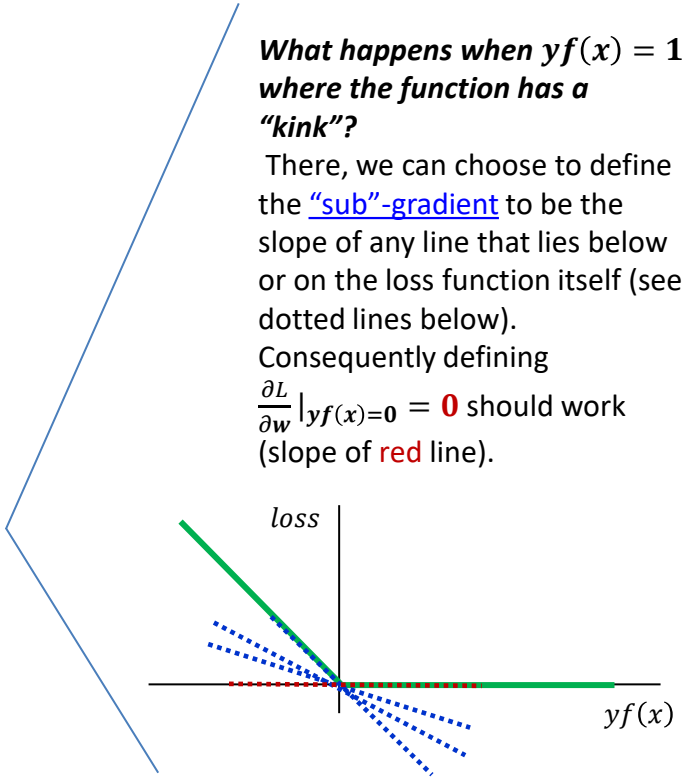
$$\min_w L(\boldsymbol{X}, \boldsymbol{Y}; \boldsymbol{w}) = \sum_{i=1}^{N} l(f(\boldsymbol{x}_i;\mathbf{w})), y_i) = \sum_{i=1}^{N} \max\{0, -y_i f(\boldsymbol{x}_i;\mathbf{w})\}$$

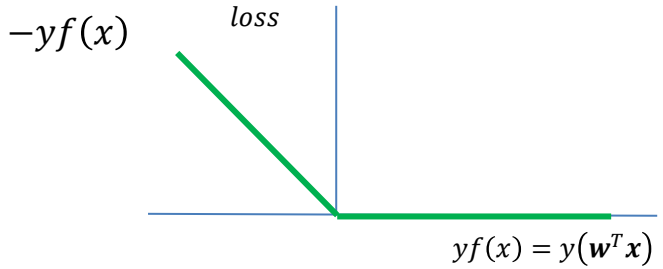
$$\frac{\partial L}{\partial \boldsymbol{w}} = \sum_{i=1}^{N} \frac{\partial l(f(\boldsymbol{x}_i;\mathbf{w})), y_i)}{\partial \boldsymbol{w}}$$

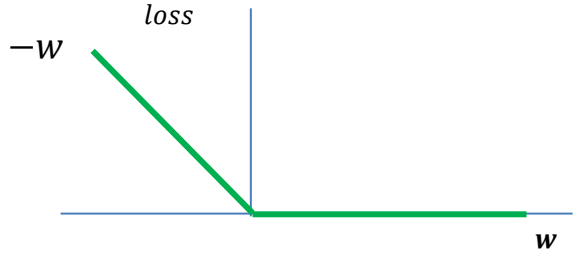
$$\frac{\partial}{\partial \boldsymbol{w}} \max\{0, -y(\boldsymbol{w}^T \boldsymbol{x})\} = \begin{cases} 0 & yf(\boldsymbol{x};\mathbf{w}) > 0 \\ -y\boldsymbol{x} & else \end{cases}$$

What happens when $yf(x) = 1$ where the function has a "kink"?

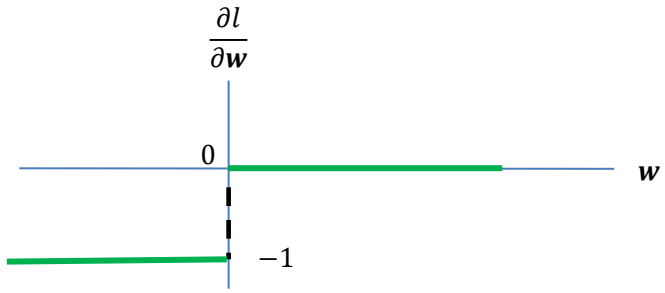
There, we can choose to define the "sub"-gradient to be the slope of any line that lies below or on the loss function itself (see dotted lines below). Consequently defining $\frac{\partial L}{\partial \boldsymbol{w}}|_{yf(x)=0} = \mathbf{0}$ should work (slope of red line).







For a simple example in which $x = 1, y = 1$



For a simple example in which $x = 1, y = 1$

Algorithm

- Given:
  - Training Examples: $\{(\boldsymbol{x}_i, y_i)|i = 1 \dots N\}, \mathrm{y_i} \in \{-1, +1\}$
  - Learning rate (step size): α
- Initialize $w^{(0)}$ at random
- Until Convergence ($k = 1 \dots K$ epochs)
  - For $i = 1 \dots N$
    - Pick example $\boldsymbol{x}_i$ with label y_i
    - Compute $f(\boldsymbol{x}_i) = \boldsymbol{w}^{(\boldsymbol{k-1})^{\boldsymbol{T}}} \boldsymbol{x}_{\boldsymbol{i}}$
    - If $y_i f(\boldsymbol{x}_i) < 0$ then update weight vector using gradient descent

$$\boldsymbol{w}^{(\boldsymbol{k})} = \boldsymbol{w}^{(\boldsymbol{k-1})} - \alpha \nabla \boldsymbol{l}\left(\boldsymbol{w}^{(\boldsymbol{k-1})}\right) = \boldsymbol{w}^{(\boldsymbol{k-1})} - \alpha(-y_i \boldsymbol{x}_i) = \boldsymbol{w}^{(\boldsymbol{k-1})} + \alpha y_i \boldsymbol{x}_i$$

    - Check for convergence to stop

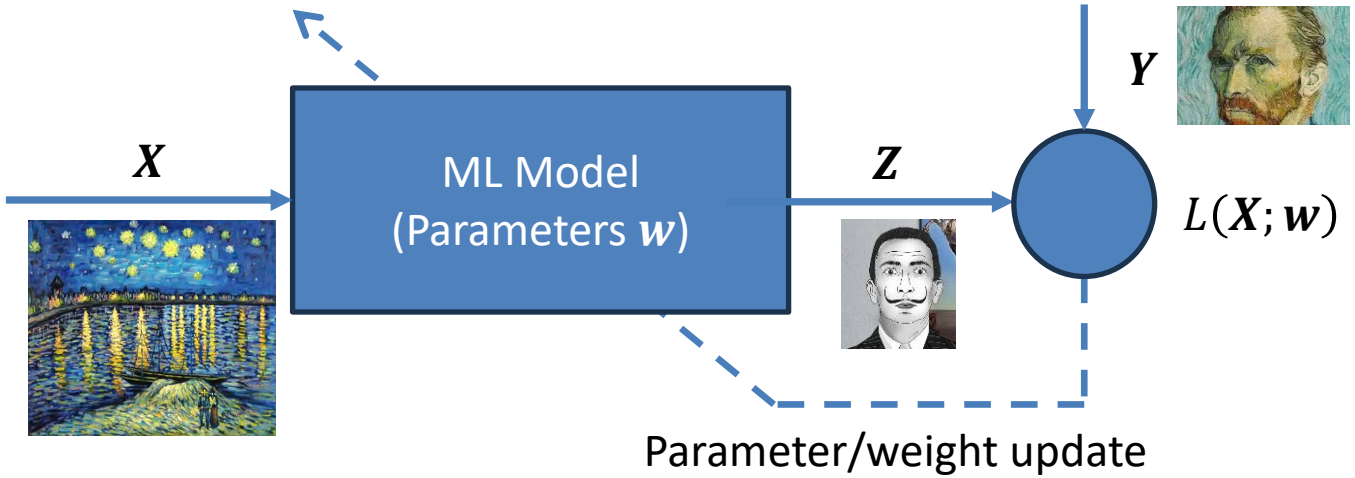
$$\nabla_{\boldsymbol{w}} \max\{0, -y(\boldsymbol{w}^T \boldsymbol{x})\} = \begin{cases} 0 & -yf(\boldsymbol{x};\mathbf{w}) < 0 \\ -y\boldsymbol{x} & else \end{cases}$$

Structural Components of Any ML Model

- **Representation**
  - How does the model produce its output given its input
    - Neuron: $f(\boldsymbol{x}_i; \boldsymbol{w}) = \boldsymbol{w}^T \boldsymbol{x}_i$
- **Evaluation**
  - How do we quantify error in training

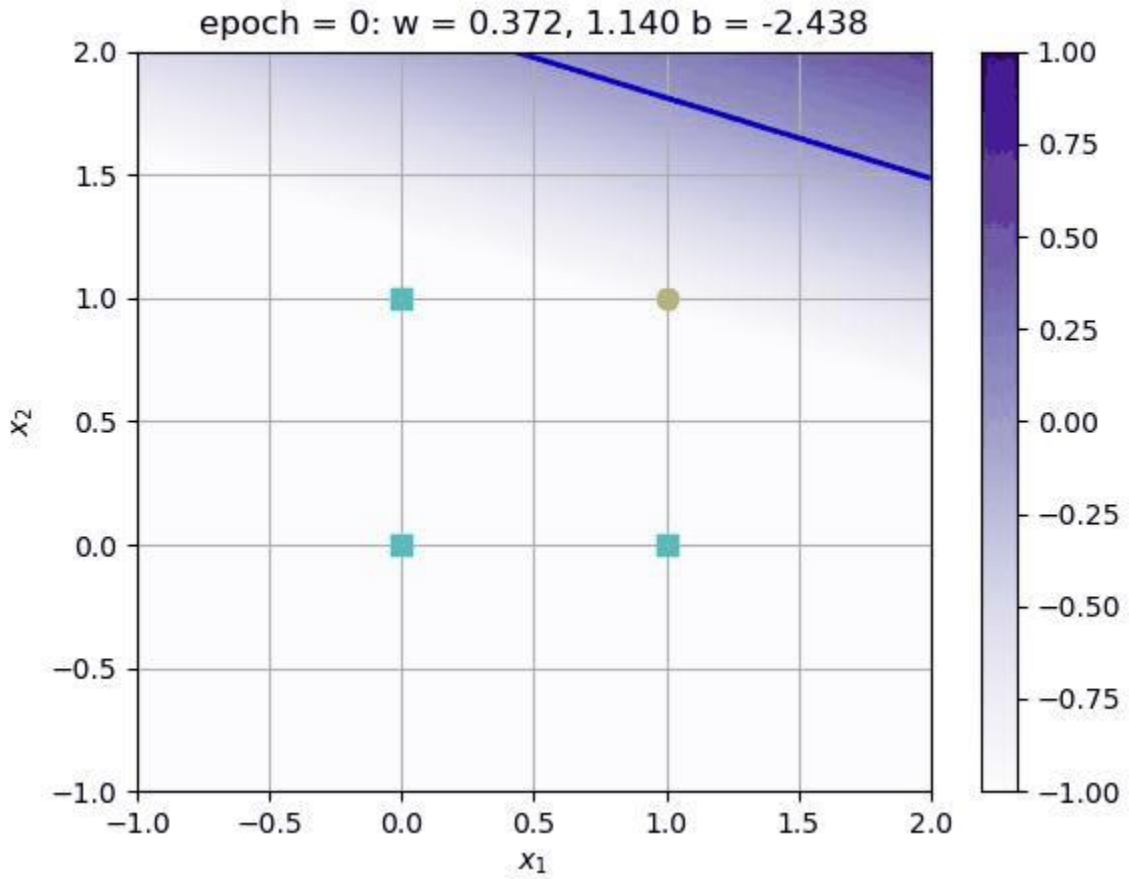
$$L(X, Y; w) = \sum_{i=1}^{N} \max\{0, -y_i f(\boldsymbol{x_i}; \mathbf{w})\}$$

- **Optimization:**
  - How do we update model parameters to minimize error
  - Gradient Descent

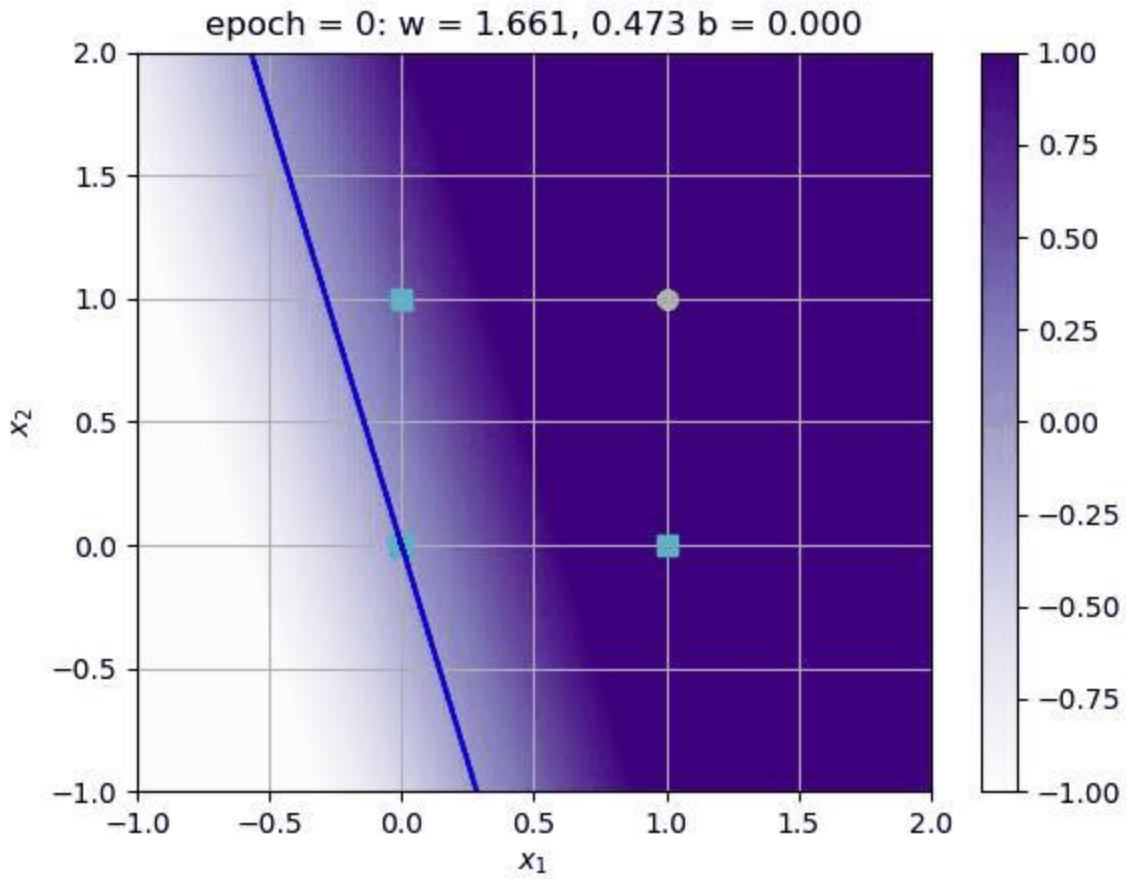


Lab Exercise

- https://github.com/foxtrotmike/CS909/blob/master/dm_lab_2_fm.ipynb
- Understand Gradient Descent
- Use Pytorch to solve a linear classification problem



https://github.com/foxtrotmike/CS909/blob/master/perceptron_video.py

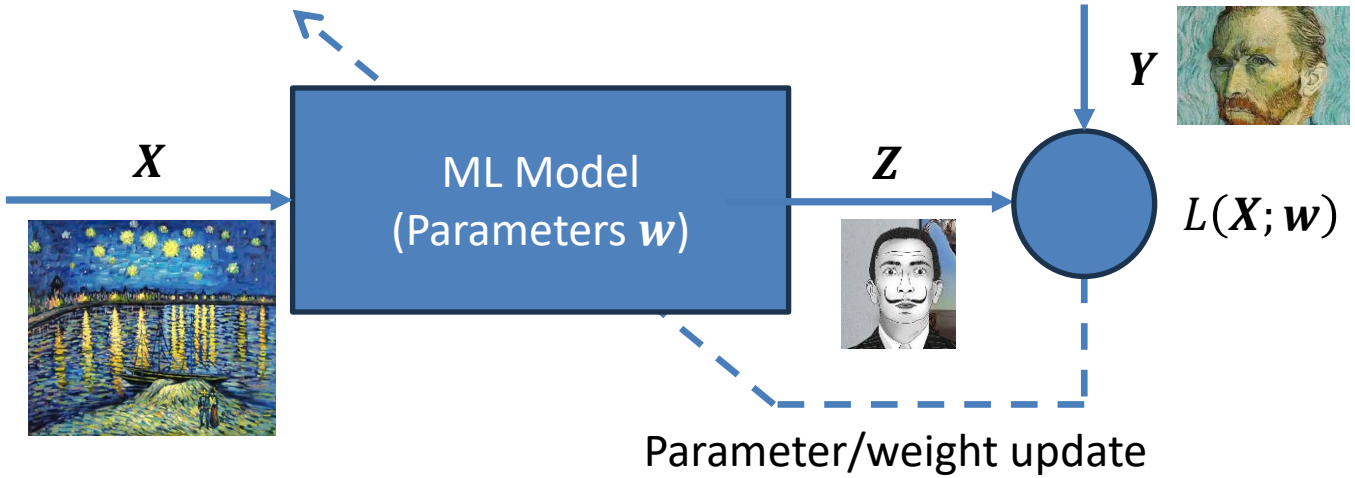

epoch = 0: w = 1.661, 0.473 b = 0.000
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
x_2
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
x_1
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00

Structural Components of Any ML Model

- **Representation**
  - How does the model produce its output given its input
    - Neuron: $f(\boldsymbol{x}_i; \boldsymbol{w}) = \boldsymbol{w}^T \boldsymbol{x}_i$
- **Evaluation**
  - How do we quantify error in training

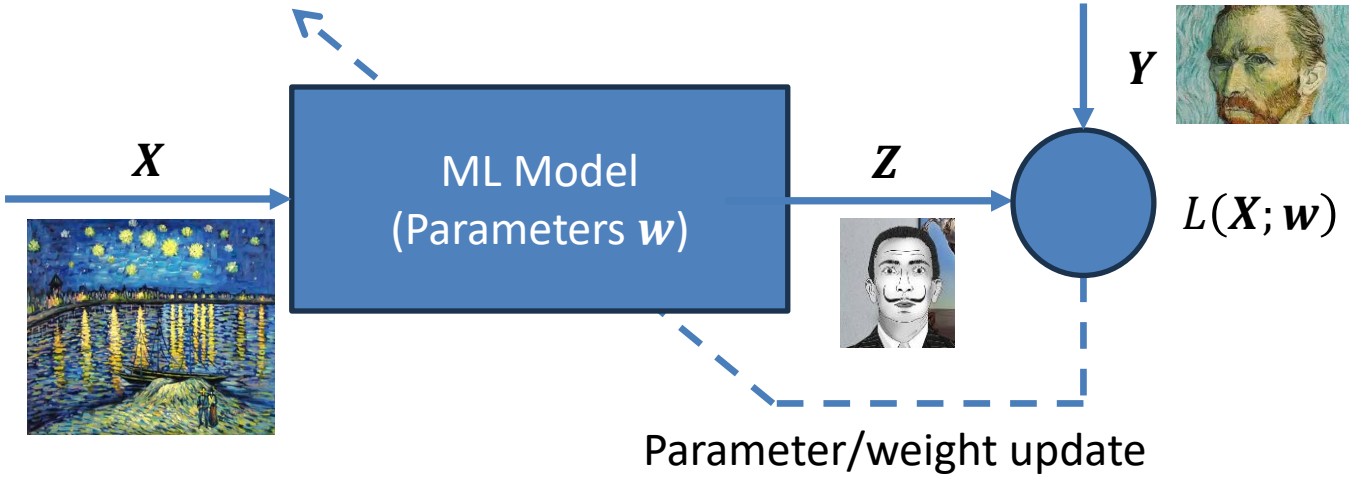
$$L(X, Y; w) = \sum_{i=1}^{N} \max\{0, -y_i f(\boldsymbol{x}_i; \mathbf{w})\}$$

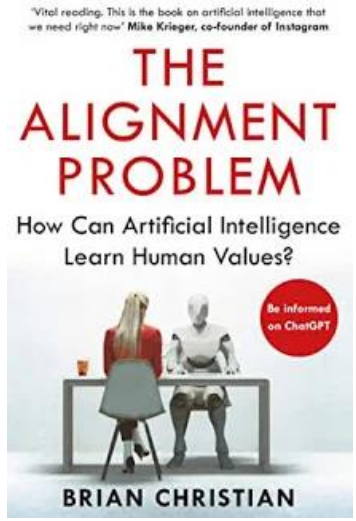
- **Optimization:**
  - How do we update model parameters to minimize error
  - Gradient Descent



Structural Components of Any ML Model

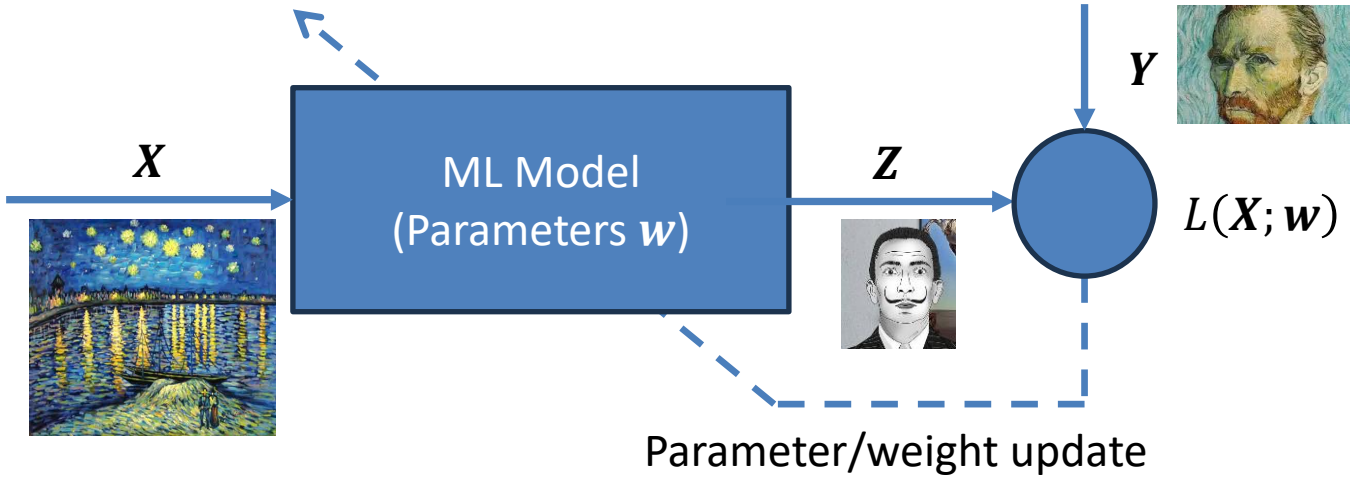
- **Representation**
  - How does the model produce its output given its input
    - Neuron: $f(\boldsymbol{x_i}; \boldsymbol{w}) = \boldsymbol{w^T x_i}$
- **Evaluation**
  - **Changing the loss function changes the type of problem the model is able to solve**
    - Classification
    - Regression
    - Ranking
    - Image Generation
    - …
  - Source of "**the Alignment Problem**"
- **Optimization:**
  - How do we update model parameters to minimize error
  - Gradient Descent





So far

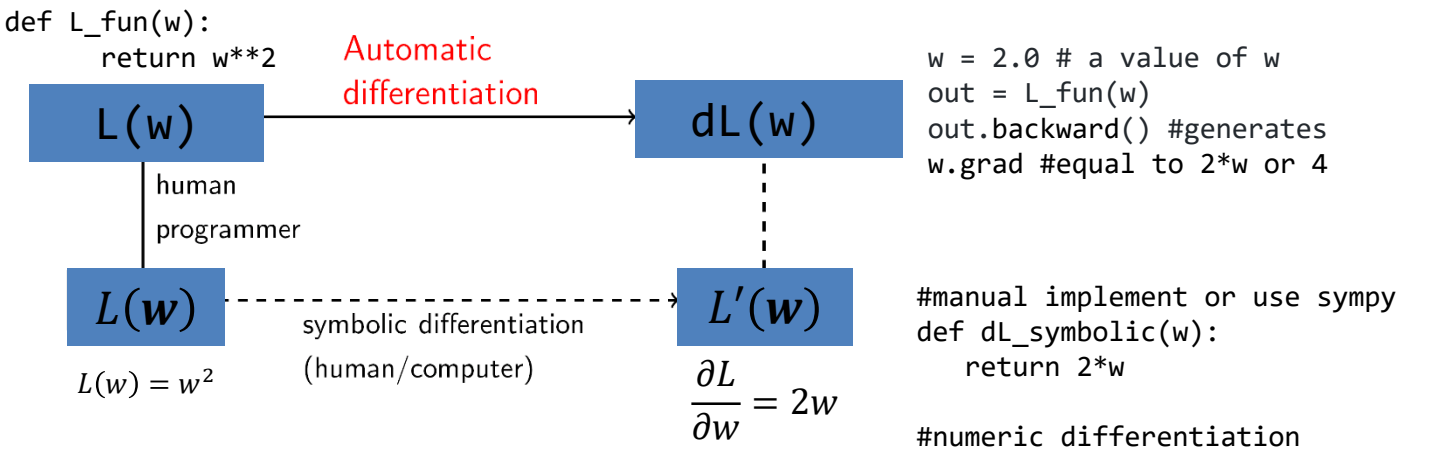
- Understand the REO Framework



- However, we still need to manually calculate derivatives for use in gradient descent…

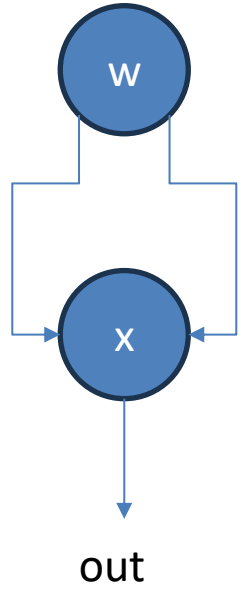
Automatic Differentiation

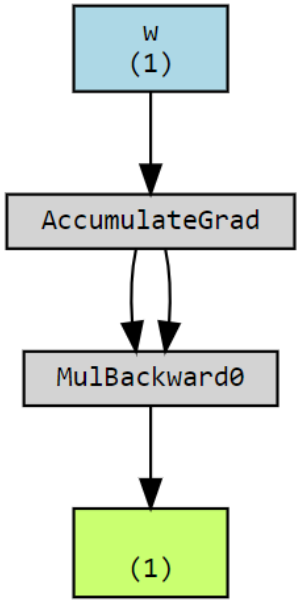
- Libraries like PyTorch, TensorFlow, and JAX compute derivatives automatically without manual derivative functions.
- Automatic differentiation computes derivatives of functions efficiently by breaking them down into elementary operations and applying the chain rule.




```
w = 2.0 # a value of w
out = L_fun(w)
out.backward() #generates
w.grad #equal to 2*w or 4
```

```
#manual implement or use sympy
def dL_symbolic(w):
   return 2*w

#numeric differentiation
def dL_numeric(w,d=0.01):
   return (L_fun(w+d)-L_fun(w))/d
```





https://en.wikipedia.org/wiki/Automatic_differentiation
https://github.com/foxtrotmike/CS909/blob/master/barebones.ipynb

Optimization Methods

- **Gradient Descent:** Minimizes the cost function by iteratively moving in the descent direction.

$$w \leftarrow w - \alpha \nabla_w L(X, Y; w)$$

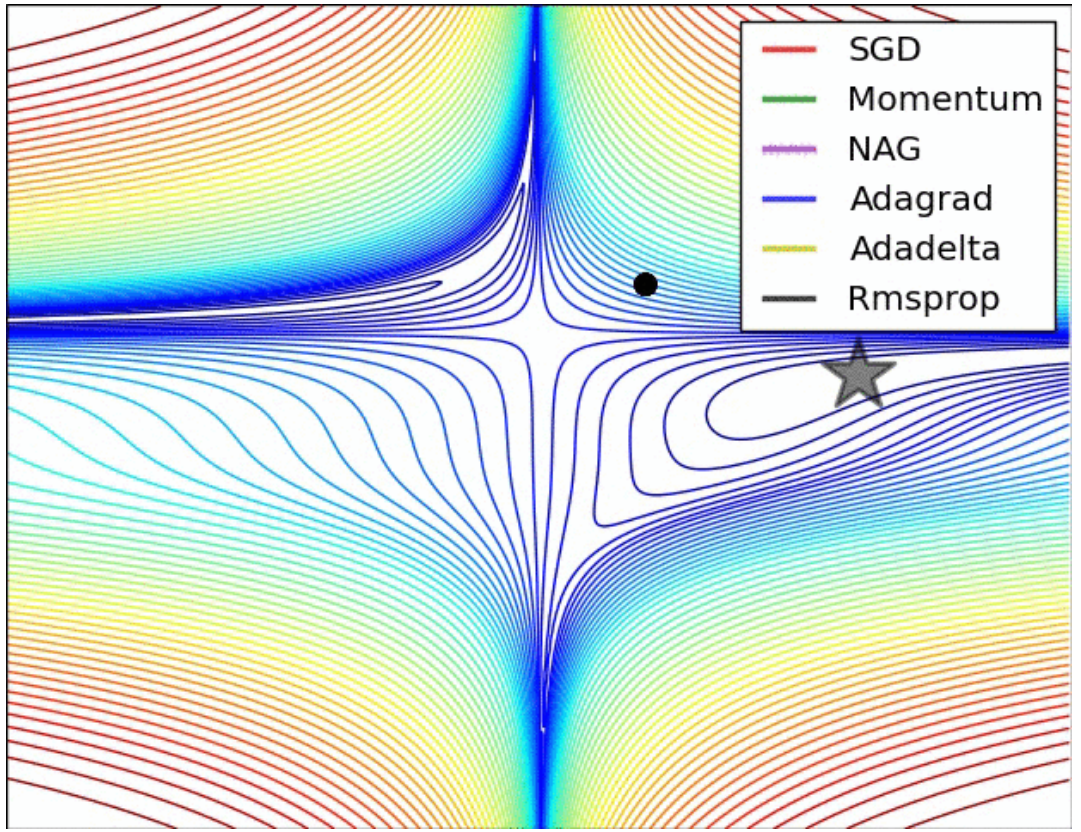
- **SGD:** Updates parameters for each training example; fast but noisy.

$$w \leftarrow w - \alpha \nabla_w L(x^{(i)}, y^{(i)}; w)$$

- **Mini-batch GD:** Uses small batches for updates; balances speed and noise.

$$w \leftarrow w - \alpha \nabla_w L(x^{(i:i+b)}, y^{(i+b)}; w)$$

- **SGD with Momentum:** Accelerates convergence by smoothing updates with past gradients.
- **Nesterov Momentum:** Looks ahead to adjust direction, preventing over-acceleration.
- **Adagrad:** Adapts learning rate per parameter based on accumulated gradients.
- **AdaDelta/RMSProp:** Limits past gradient accumulation to keep adaptive learning rates effective.
- **Adam:** Combines adaptive learning rates and momentum for robust optimization.
- **Learning Rate Scheduling:** Adjusts learning rates during training to improve convergence.
  - OneCycleLR (PyTorch): Schedules learning rate changes to optimize training efficiency https://pytorch.org/docs/stable/generated/torch.optim.lr_scheduler.OneCycleLR.html



Adam

$$w \leftarrow w - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

where

$$\hat{m}_t = \frac{\beta_1 m_{t-1} + (1-\beta_1) g_t}{1-\beta_1^t}$$

$$\hat{v}_t = \frac{\beta_2 v_{t-1} + (1-\beta_2) g_t^2}{1-\beta_2^t}$$

$$g_t = \nabla_w L(x^{(i:i+b)}, y^{(i:i+b)}; w)$$

```
>>> data_loader = torch.utils.data.DataLoader(...)
>>> optimizer = torch.optim.SGD(model.parameters(), lr=0.1,
momentum=0.9)
>>> scheduler = torch.optim.lr_scheduler.OneCycleLR(optimizer,
max_lr=0.01, steps_per_epoch=len(data_loader), epochs=10)
>>> for epoch in range(10):
>>>     for batch in data_loader:
>>>         train_batch(...)
>>>         optimizer.step()
>>>         scheduler.step()
```

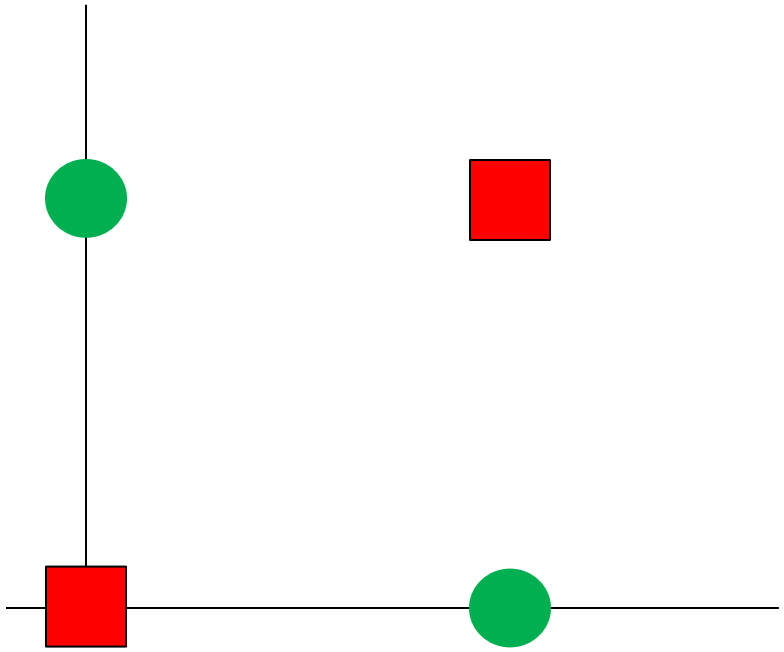
An overview of gradient descent optimization algorithms by Sebastian Ruder, 20-16 http://sebastianruder.com/optimizing-gradient-descent/ , https://arxiv.org/abs/1609.04747

What this means for us?

- If we can define
  - **Representation** of the machine learning model
  - **Evaluation** (or loss function)
- **The Optimization step can be pretty much automated**

- *Successful optimization is a necessary (but not sufficient) condition for learning and should always be ensured*
  - Plot the loss function or convergence curves (Tensorboard and W&B help)

But what if we have a linearly inseparable problem?

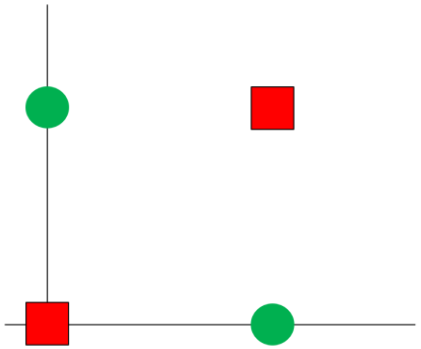


We need more effective model representations

Solution

- Non-linear transformations of the features may allow us to perform effective partitioning

$$\phi(\boldsymbol{x}): \boldsymbol{R}^{\boldsymbol{d}} \rightarrow \boldsymbol{R}^{\boldsymbol{d}'}$$



$x^{(1)}$	$x^{(2)}$	y
0	0	-1
0	1	+1
1	0	+1
1	1	-1

No Solution

Transformation

$$\boldsymbol{\phi}\left(\begin{bmatrix} \boldsymbol{x}^{(1)} \\ \boldsymbol{x}^{(2)} \end{bmatrix}\right) = \begin{bmatrix} \boldsymbol{x}^{(1)^2} \\ \boldsymbol{x}^{(2)^2} \\ \sqrt{2}\boldsymbol{x}^{(1)}\boldsymbol{x}^{(2)} \end{bmatrix}$$

$f(\mathbf{x};\mathbf{w}) = w_1x^{(1)}+w_2x^{(2)}+w_3x^{(3)} +b= 0$

$x'^{(1)}$	$x'^{(2)}$	$x'^{(3)}$	y
0	0	0	-1
0	1	0	+1
1	0	0	+1
1	1	$\sqrt{2}$	-1

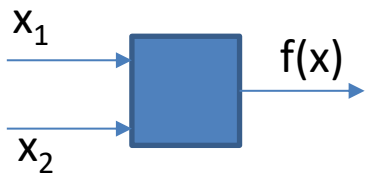
Solution: $w_1 = 2$, $w_2 = 2$, $w_3 = -3$, $b = -1$

https://foxtrotmike.github.io/CS909/transformation.html

How to fold the space?

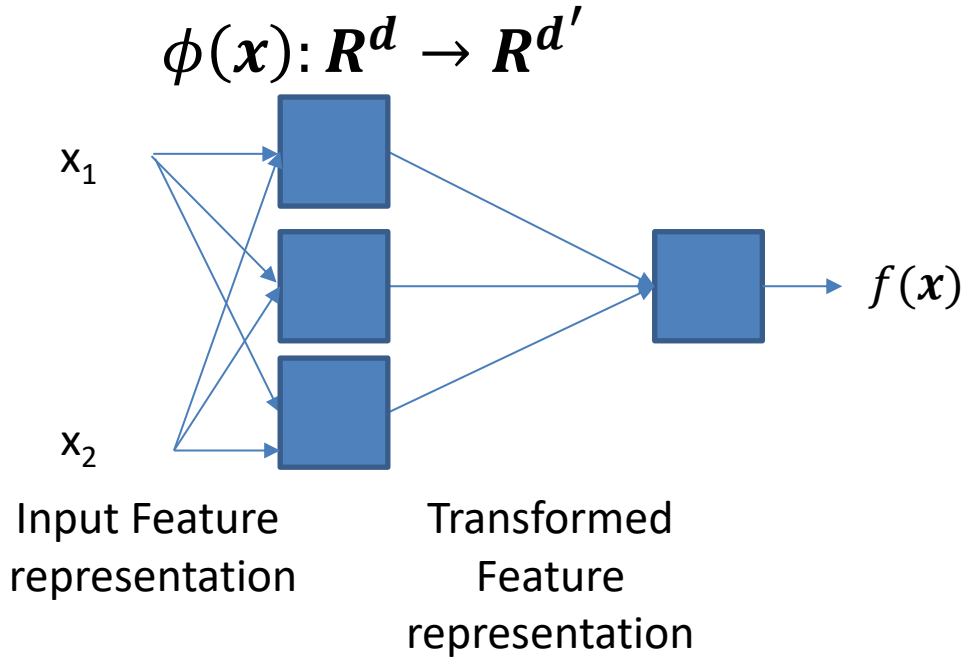
- Change the definition of distance between points

- How to achieve this?
  - By transforming the features of the examples to another space



How to fold the space?

- Change the definition of distance between points
- How to achieve this?
  - By transforming the features of the examples to another space

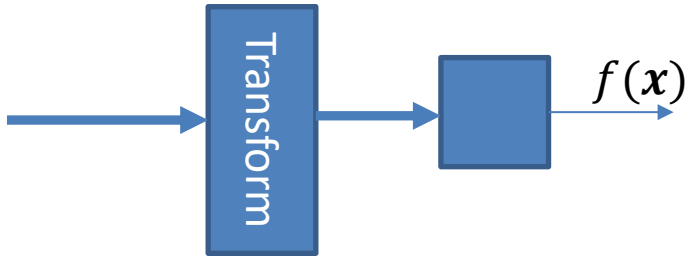


How to fold the space?

- Change the definition of distance between points

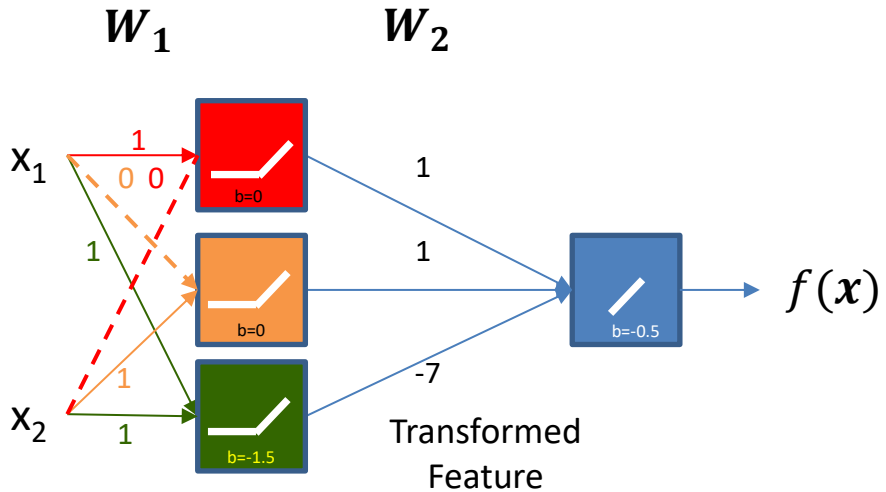
- How to achieve this?
  - By transforming the features of the examples to another space
  - At an abstract level

$$\phi(\boldsymbol{x}): \boldsymbol{R^d} \rightarrow \boldsymbol{R^{d'}}$$

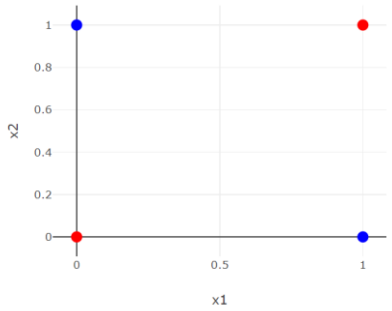


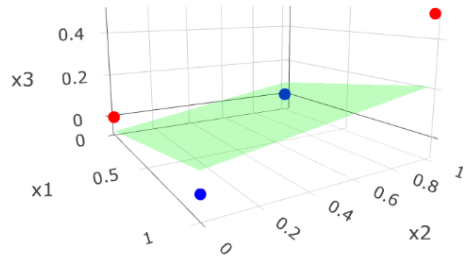
Non-linear transformations in multilayered perceptron

Input Feature representation $\boldsymbol{x} \in \boldsymbol{R^d}$



Transformed Feature representation





https://foxtrotmike.github.io/CS909/transformation.html

Mathematical Representation
(ignoring biases)

Input: $\boldsymbol{x} \in \boldsymbol{R}^{d=2}$

Output of layer 1: $\boldsymbol{R}^{d=2} \rightarrow \boldsymbol{R}^{d_1=3}$

$$\boldsymbol{f_1(x) = a_1(W_1 x)}$$

$\boldsymbol{W_1}$ **is of shape** $d_1 \times d$

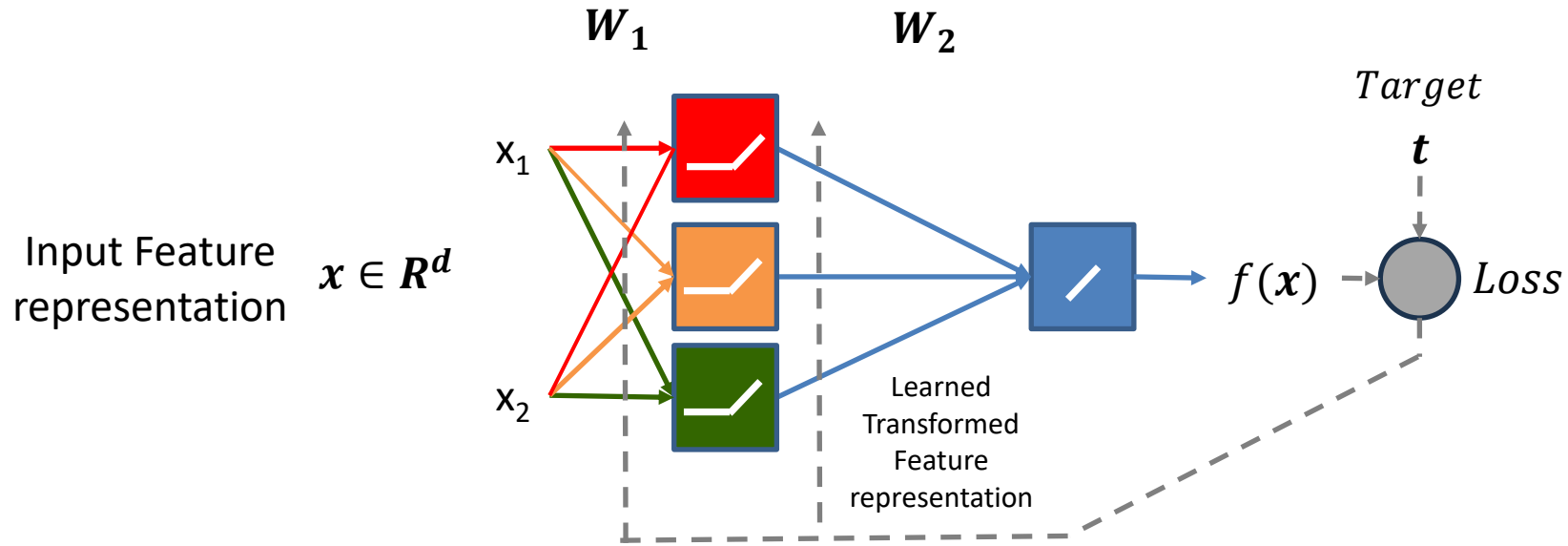
Output of layer 2 (Final): $\boldsymbol{R}^{d_1=3} \rightarrow \boldsymbol{R}^{d_2=1}$

$$\boldsymbol{f(x) = f_2(x) = a_2(W_2 f_1(x))}$$

$$\boldsymbol{f(x) = a_2(W_2 a_1(W_1 x))}$$

$\boldsymbol{W_2}$ **is of shape** $d_2 \times d_1$

Non-linear transformations in multilayered perceptron



Mathematical Representation
(ignoring biases)

Input: $x \in R^{d=2}$

Output

$$f(x) = a_2(W_2 a_1(W_1 x))$$

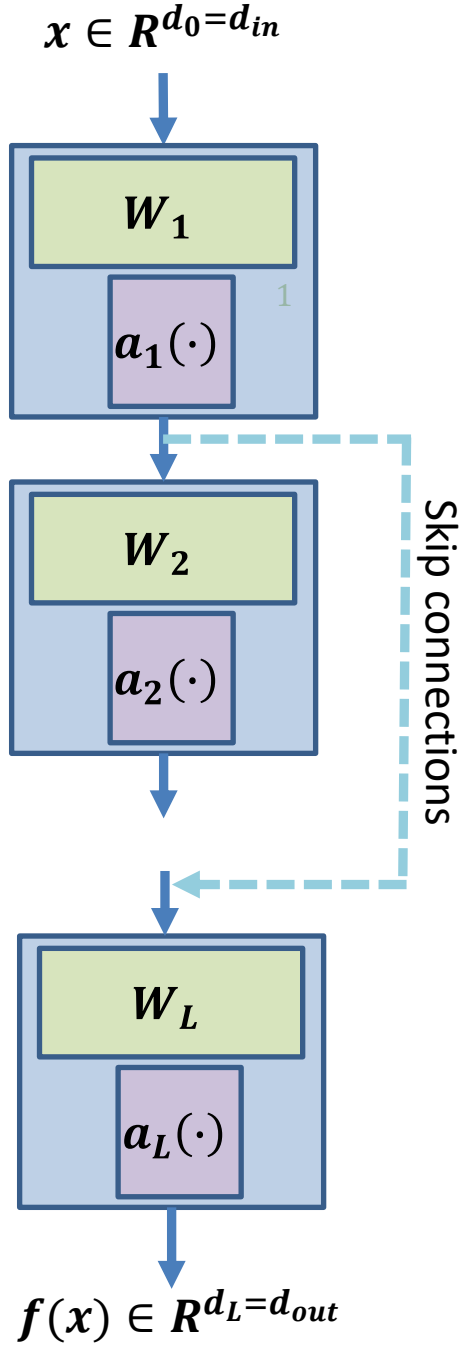
REO for MLPs

- **Representation**
  - Mathematically, a single "Fully Connected" or "Dense Layer" performs the operation
    - $\boldsymbol{f_L(x) = a_L(W_L x)}$
  - Multiple such Layers can be stacked to build MLP modules

$$f(\boldsymbol{x}; \boldsymbol{W}) = \boldsymbol{a}_L\left(\boldsymbol{W}_L\left(\ldots\left(\boldsymbol{a}_2\left(\boldsymbol{W}_2(\boldsymbol{a}_1(\boldsymbol{W}_1\boldsymbol{x}))\right)\right)\right)\right)$$

- **Evaluation**: Problem dependent

- **Optimization**:
  - Use gradient descent with gradients computed via automatic differentiation
  - However, optimization can be difficult due to a long path between input and output and the non-linear effects of a large number of parameters.
  - Needs careful weight initialization and architecture choices such as skip connections (Residual Networks)



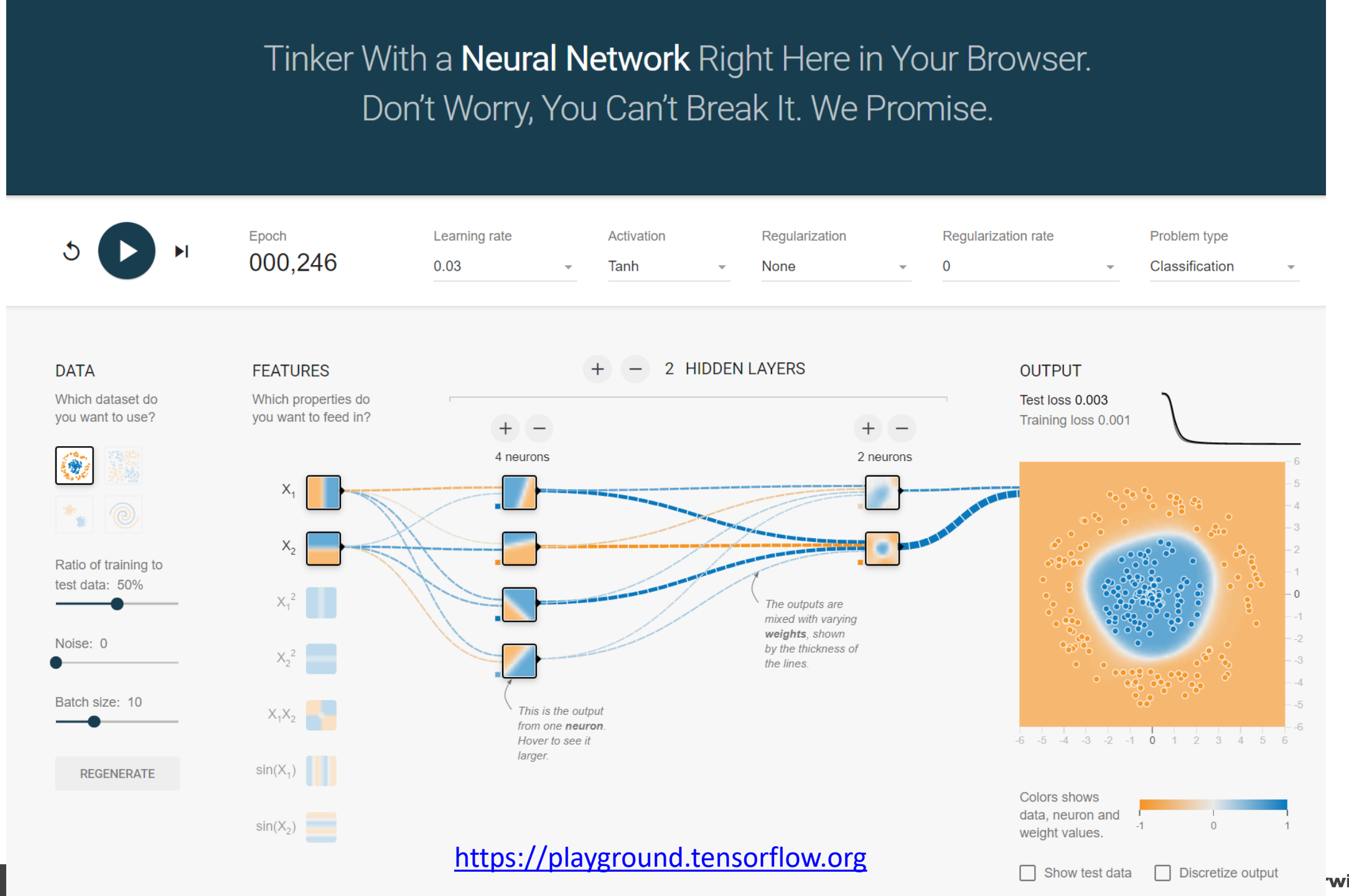
Pytorch fully connected layer: https://pytorch.org/docs/stable/generated/torch.nn.Linear.html
Pytorch module documentation: https://pytorch.org/docs/stable/generated/torch.nn.Module.html

Key Insights

1. Artificial Neurons can be interconnected to form networks leading to more complex machine learning model "Representations“
2. Nonlinear activation functions combined with multiple layers of neurons enable complex data transformations.
3. By optimizing the weights of neurons to reduce the loss function, the network can learn complex representations from the data
4. These transformations alter how the input data is represented within the network and thus redefine the distance or similarity between data points.

Universal approximation theorem: https://en.wikipedia.org/wiki/Universal_approximation_theorem

Delalleau, Olivier, and Yoshua Bengio. “Shallow vs. Deep Sum-Product Networks.” In Advances in Neural Information Processing Systems, Vol. 24. Curran Associates, Inc., 2011. https://papers.nips.cc/paper_files/paper/2011/hash/8e6b42f1644ecb1327dc03ab345e618b-Abstract.html.



https://playground.tensorflow.org

Lab Exercise: Solve XOR and Digit Classification

- Understand the MLP and adapt it to solve XOR Classification
- https://github.com/foxtrotmike/CS909/blob/master/pytorch_nn_barebones.ipynb

- Solve Digit Classification (demonstrates tensorboard use)
- https://github.com/foxtrotmike/CS909/blob/master/pytorch_mlp_mnist.ipynb

Computation Graph of a two-layer network

Representation

```
model = torch.nn.Sequential(
          torch.nn.Linear(2, 2),
          torch.nn.Sigmoid(),
          torch.nn.Linear(2, 1),
          torch.nn.Sigmoid()
        ).to(device)

z = model(x)
```

Evaluation

```
e = loss_fn(z, y)
```

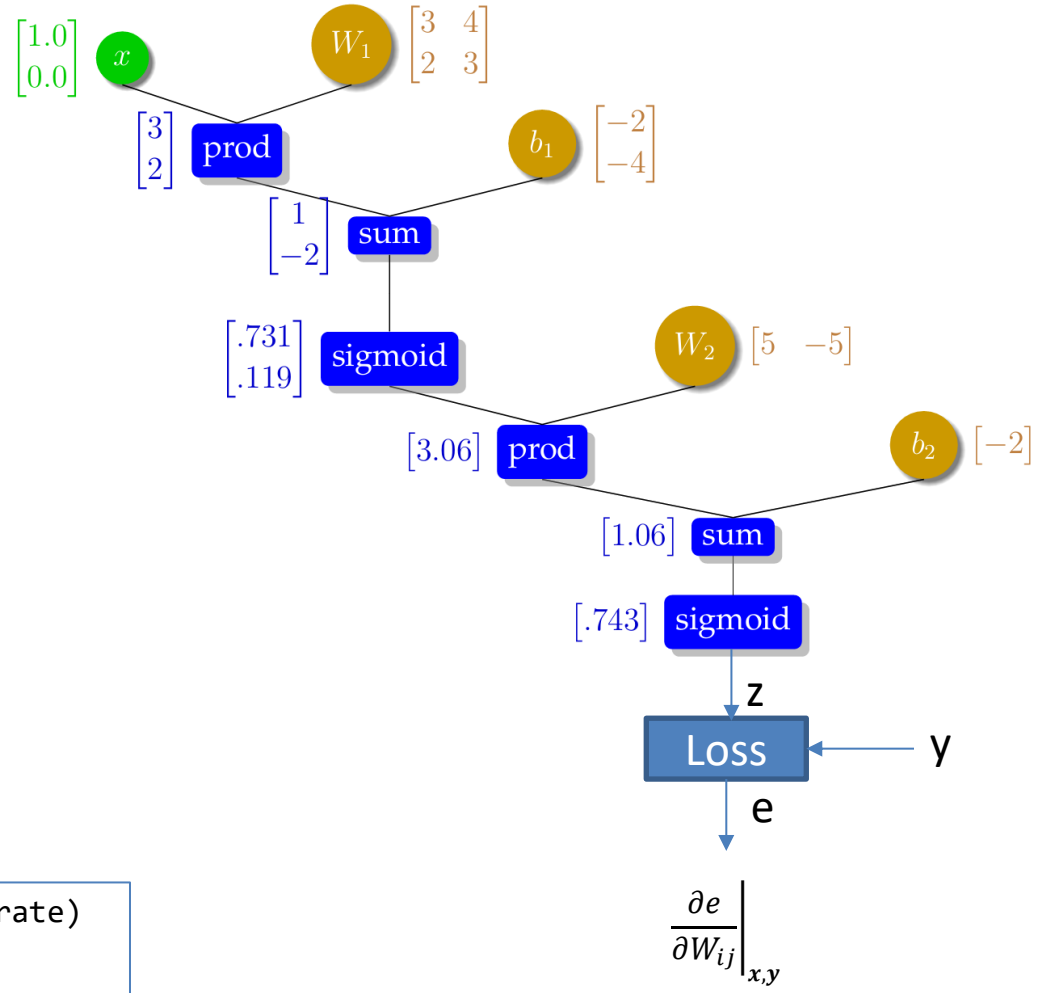
Optimization

Manual Gradient Descent

```
model.zero_grad()
e.backward()


with torch.no_grad():
for param in model.parameters():
  param.data -= learning_rate * param.grad
```

Using Built-in Optimizer

```
# optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
model.zero_grad()
e.backward()
optimizer.step()
```



Note on Deep Learning Libraries Libraries

- All Neural Network/Deep Learning Libraries Do three things
  1. Automatic Differentiation (Efficient Algorithms such as Reverse mode autodiff!)
  2. Implement Optimizers
  3. Use efficient hardware for multiprocessing (GPUs)
- Support model representation development

TensorFlow

- Static Computing Graphs
  - Build before you go (new version has dynamic graphs too!)
  - Compile then run/fit
- Good Documentation
- Distributed Computing / Delivery
- TensorFlow.js

pyTorch

- Dynamic Computing Graphs
  - Graph built at run time
  - Build as you go
- Good for research

```
using Zygote

# Define a simple function
f(x) = 3x^2 + 2x + 1
# derivative of f at x = 2
gradient(f, 2)
```

Key Concept: Optimization is not Learning

- What does optimization of the loss over training data achieve?
  - Minimizing error over **training data**

- What is the goal of machine learning?
  - Effective generalization to **unseen data**

- **Structural Risk Minimization**
  - **Empirical risk minimization (i.e., minimization of error over training data) is not enough**
  - Control the representations learned by the model by controlling its capacity
    - Validation and Early Stopping
    - Regularization strategies: Weight regularization, Drop out and batch normalization
    - Data Augmentation
    - Building invariances
    - …

Key Concept: Validation and Good experiment design

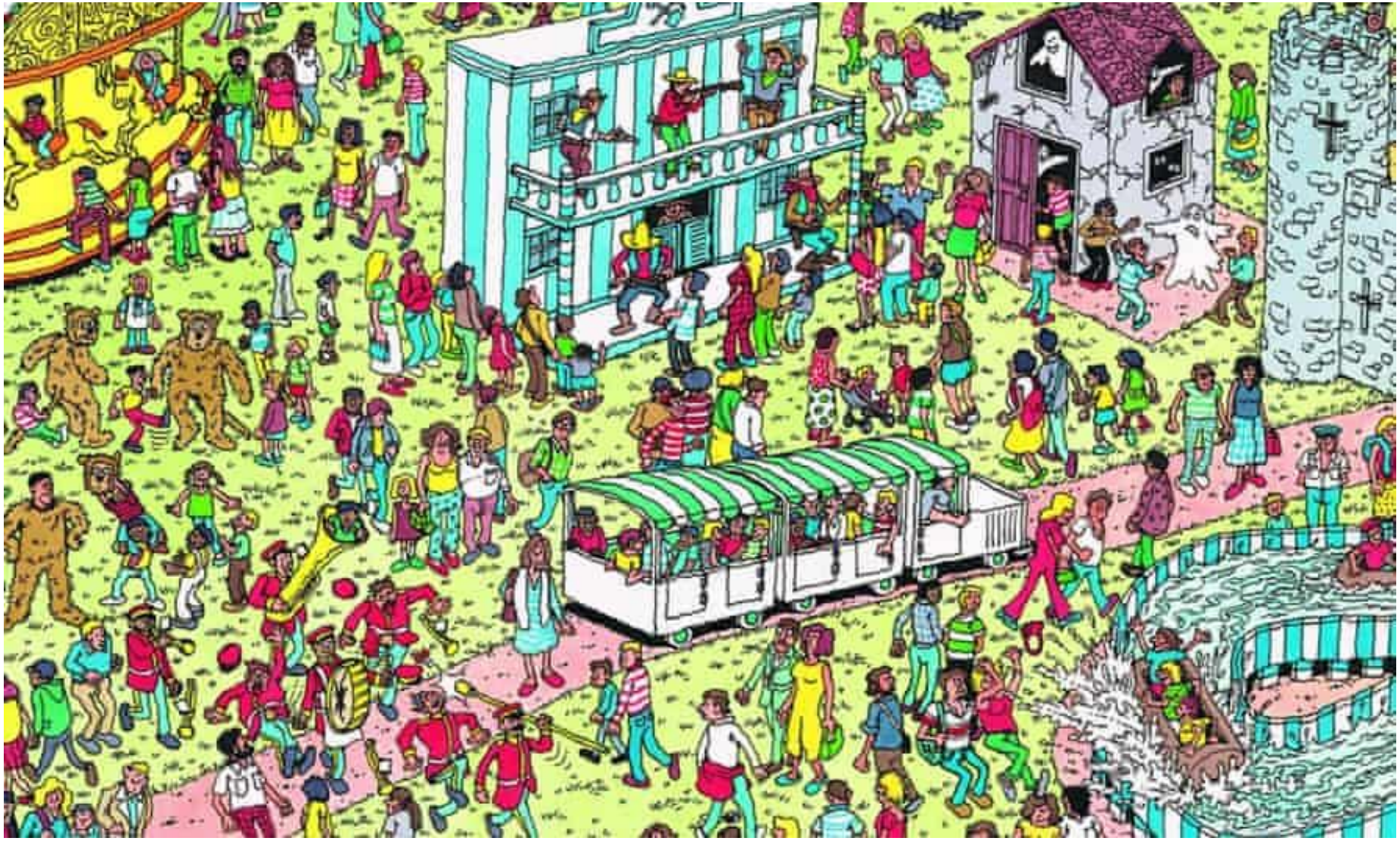
- Three Simple principles
  1. Ensure that the model is evaluated in an environment that closely mirrors its intended deployment scenario
  2. Do Not Use Test Data in Training or Validation
  3. Choose appropriate metrics

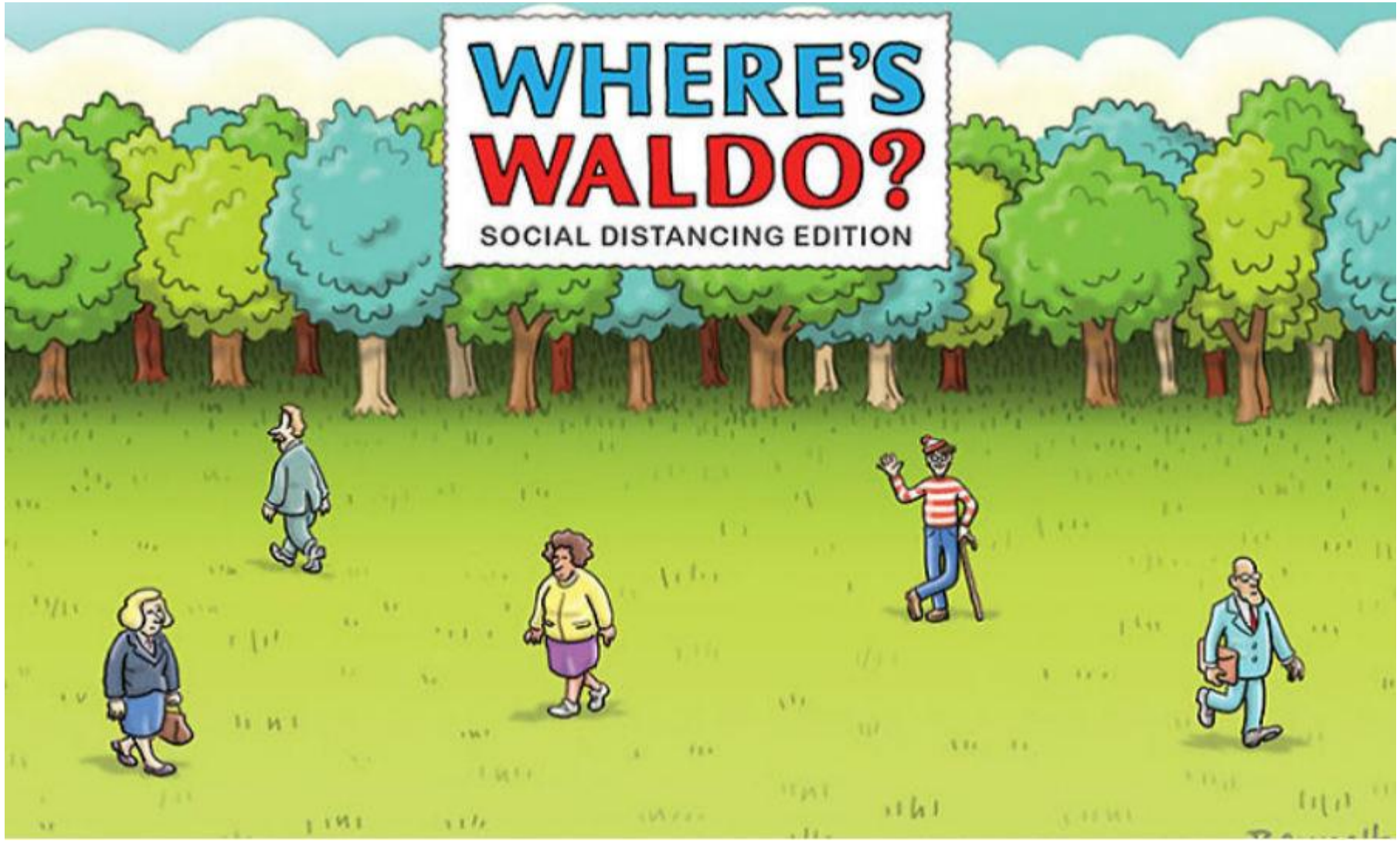
See Model Evaluation Lectures in Week 3: https://warwick.ac.uk/fac/sci/dcs/teaching/material/cs909/

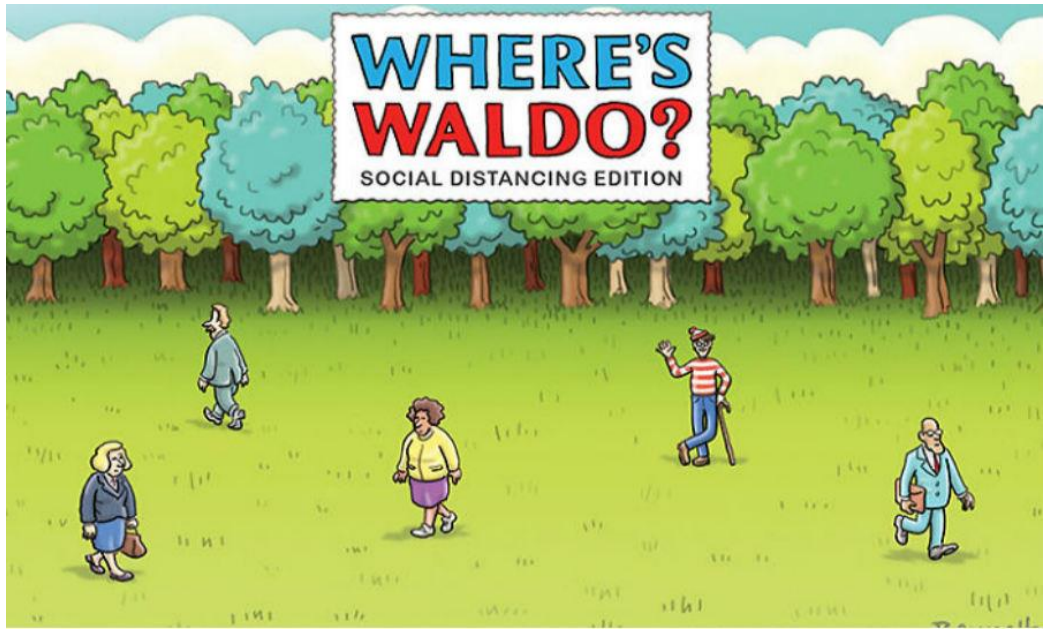
Minhas, Fayyaz, Amina Asif, and Asa Ben-Hur. “Ten Ways to Fool the Masses with Machine Learning.” *ArXiv:1901.01686 [Cs, Stat]*, January 7, 2019. http://arxiv.org/abs/1901.01686.

Lecture 2
Convolutional Neural Networks

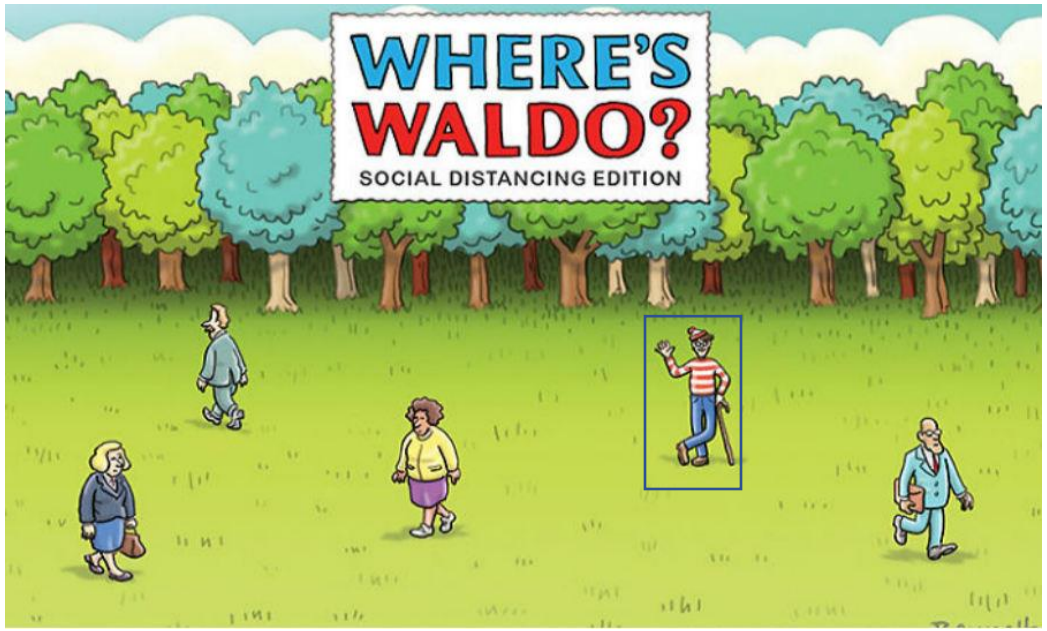
Where’s Waldo?

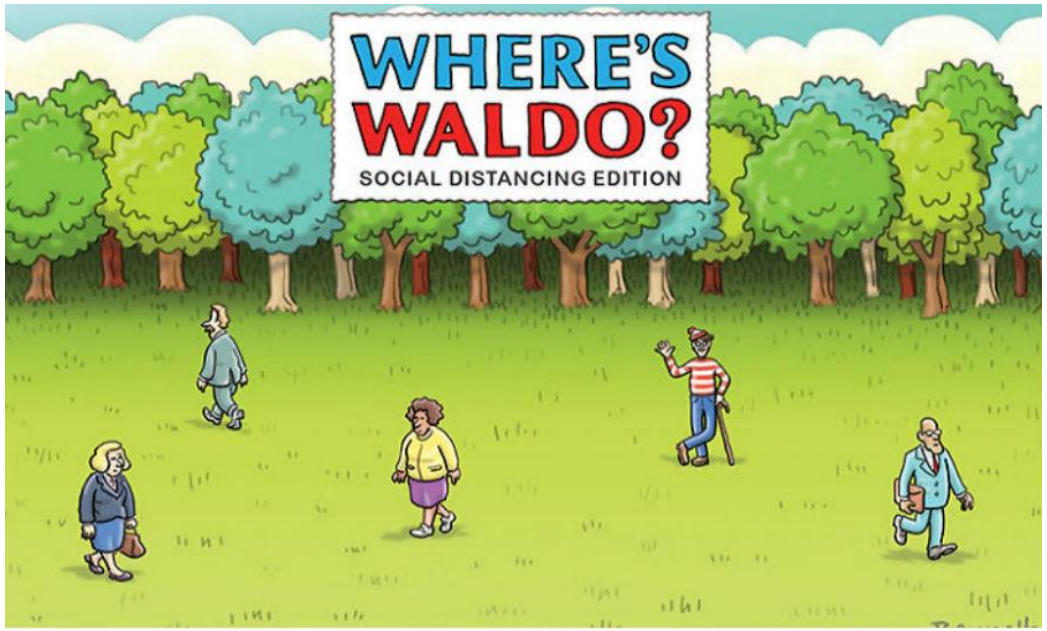



WHERE'S
WALDO?
SOCIAL DISTANCING EDITION


WHERE'S
WALDO?
SOCIAL DISTANCING EDITION

Model


WHERE'S
WALDO?
SOCIAL DISTANCING EDITION

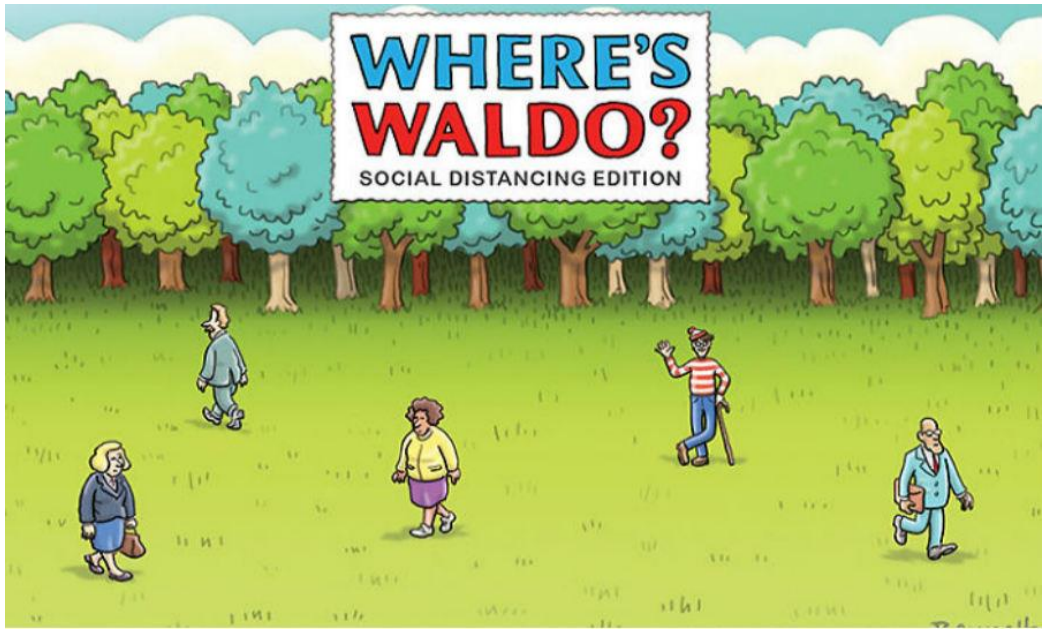


Model

- If we use a perceptron here then what would be the number of weights needed if the image size of the input and output is NxN?

Even single layer

$O(N \times N \times N \times N)$

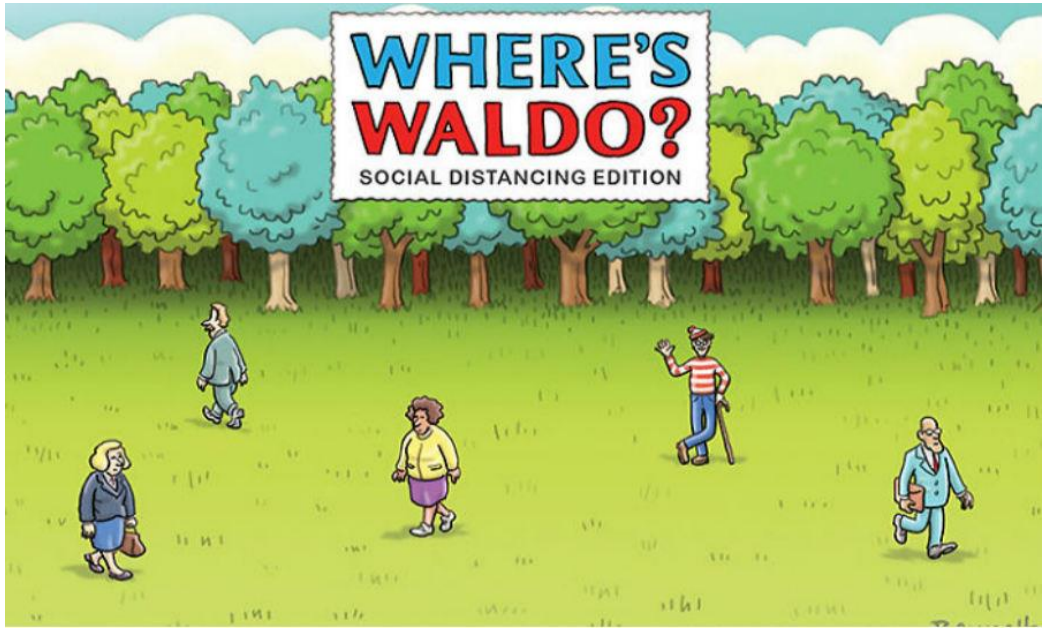


NxN

Model

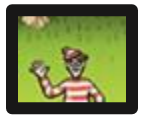
NxN

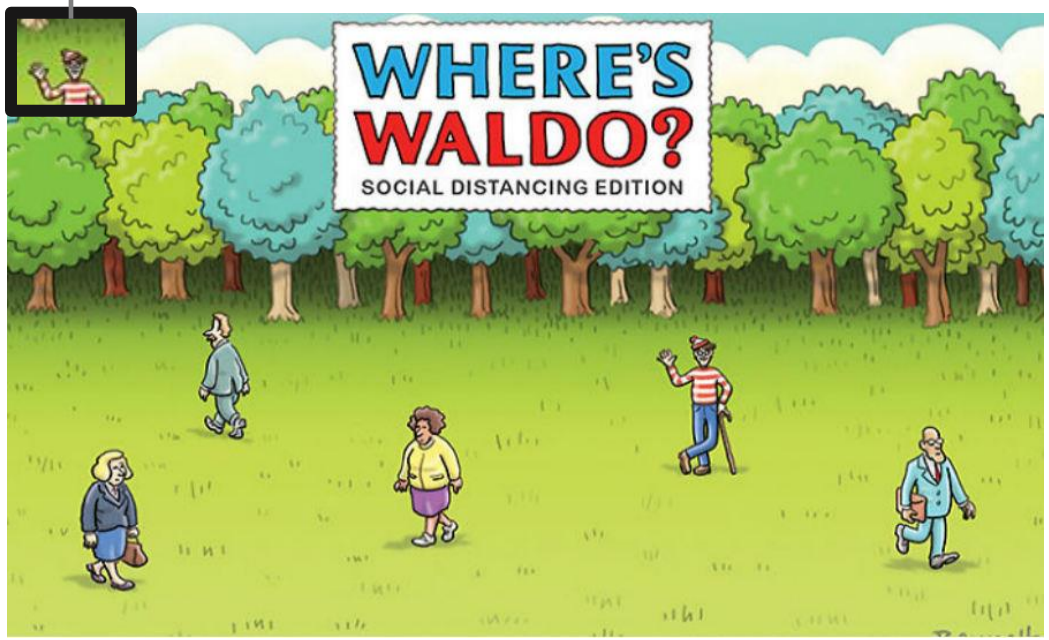
- Assume we have a cutout of what Waldo looks like
- And if we “scan” (formally called correlate or convolve) the cutout against the input image – we should see a peak at the location where Waldo occurs in the input image
- Correlelation is dot-product (a measure of similarity)



Model

$$\begin{bmatrix} 0.28 \\ 0.32 \\ \vdots \\ 0.37 \end{bmatrix} \cdot \begin{bmatrix} 0.79 \\ 0.32 \\ \vdots \\ 0.37 \end{bmatrix}$$



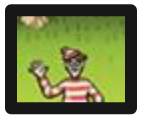


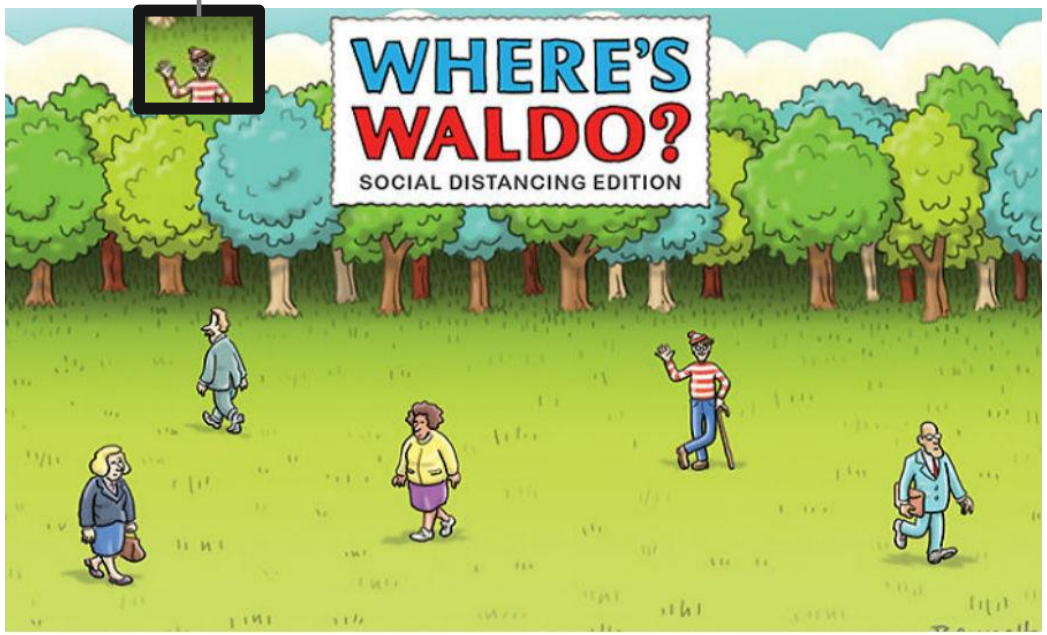
Model

0.45	0.68	0.87	0.39	0.19
0.2	0.12	0.9	0.63	0.38
0.99	0.86	0.4	0.82	0.28
0.36	0.98	0.8	0.065	0.58
0.28	0.21	0.067	0.25	0.36

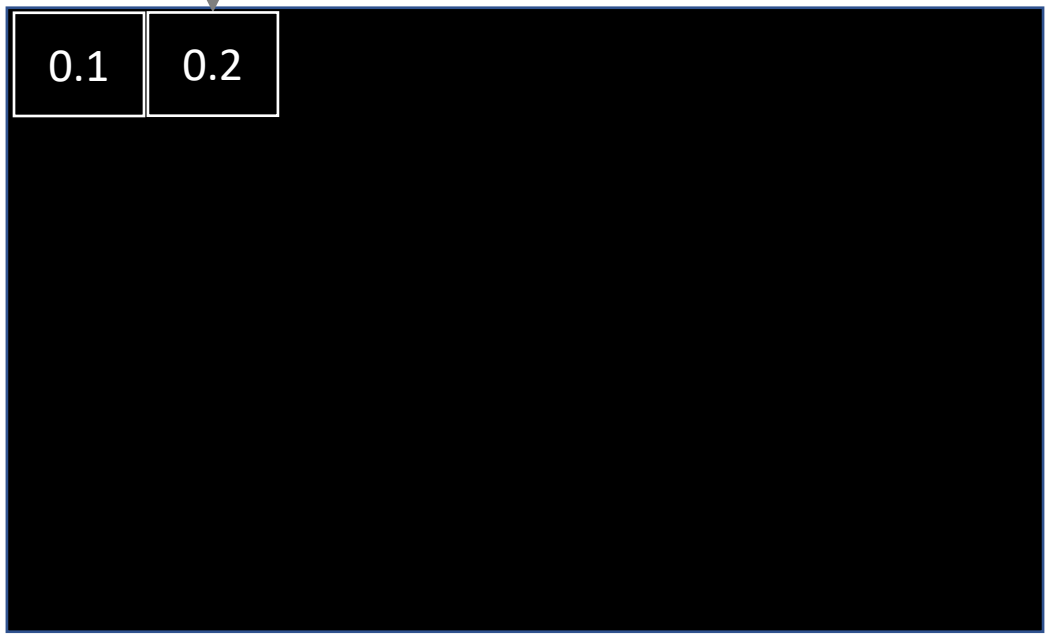
$$\begin{bmatrix} 0.45 \\ 0.68 \\ \vdots \\ 0.36 \end{bmatrix} \cdot \begin{bmatrix} 0.79 \\ 0.32 \\ \vdots \\ 0.37 \end{bmatrix}$$

0.79	0.32	0.9	0.47	0.88
0.86	0.57	0.16	0.39	0.013
0.93	0.53	0.74	0.67	0.82
0.41	0.7	0.011	0.26	0.35
0.74	0.68	0.93	0.19	0.37





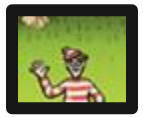
Model

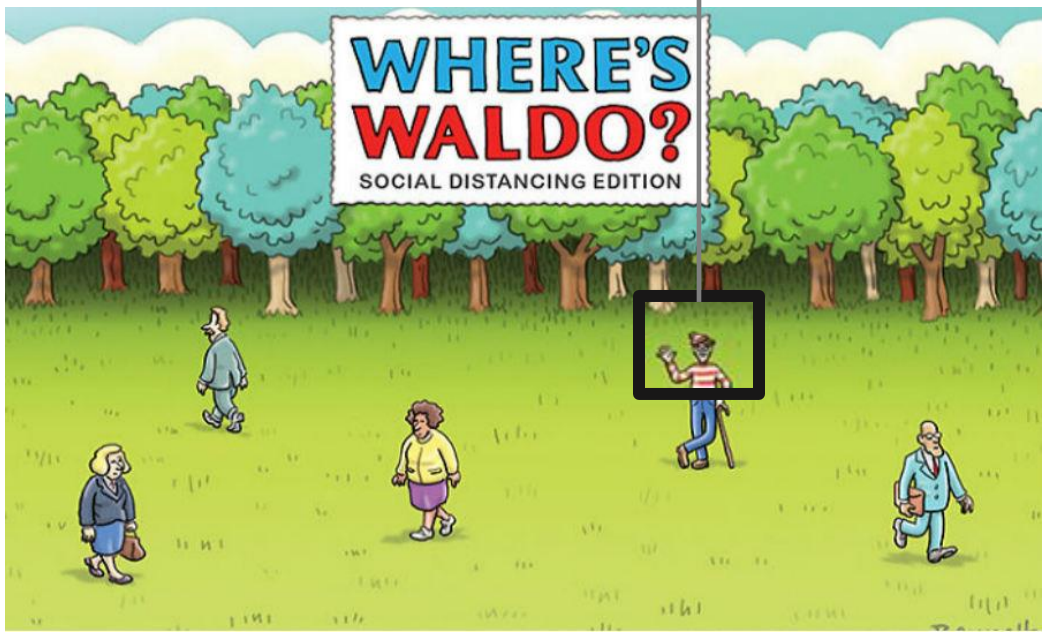


0.79	0.32	0.9	0.47	0.88
0.86	0.57	0.16	0.39	0.013
0.93	0.53	0.74	0.67	0.82
0.41	0.7	0.011	0.26	0.35
0.74	0.68	0.93	0.19	0.37

$$\begin{bmatrix} 0.79 \\ 0.32 \\ \vdots \\ 0.37 \end{bmatrix} \cdot \begin{bmatrix} 0.79 \\ 0.32 \\ \vdots \\ 0.37 \end{bmatrix}$$

0.79	0.32	0.9	0.47	0.88
0.86	0.57	0.16	0.39	0.013
0.93	0.53	0.74	0.67	0.82
0.41	0.7	0.011	0.26	0.35
0.74	0.68	0.93	0.19	0.37

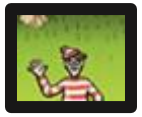


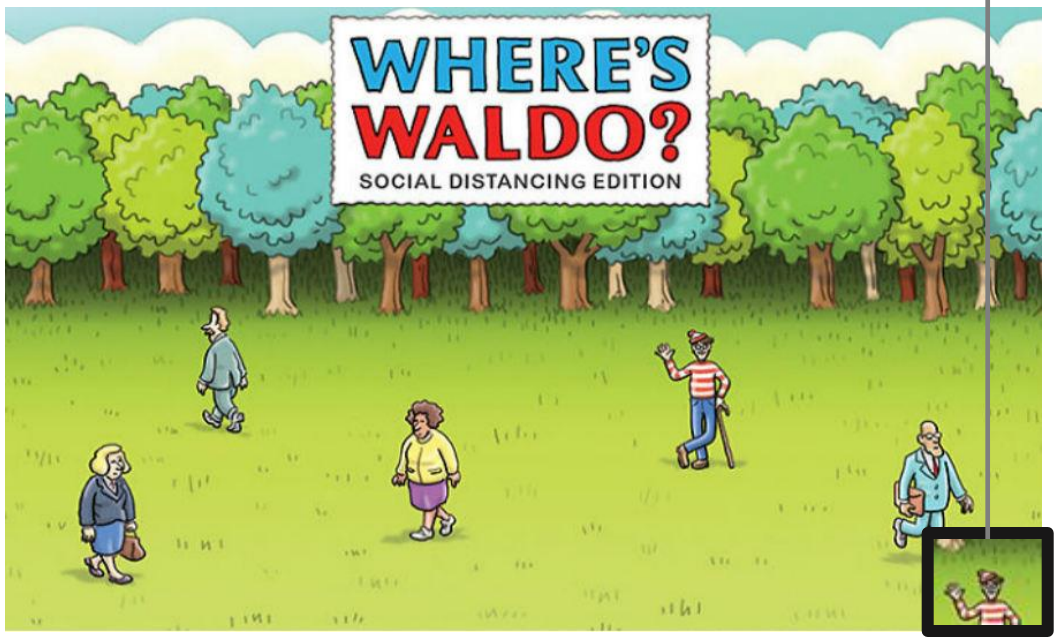


Model

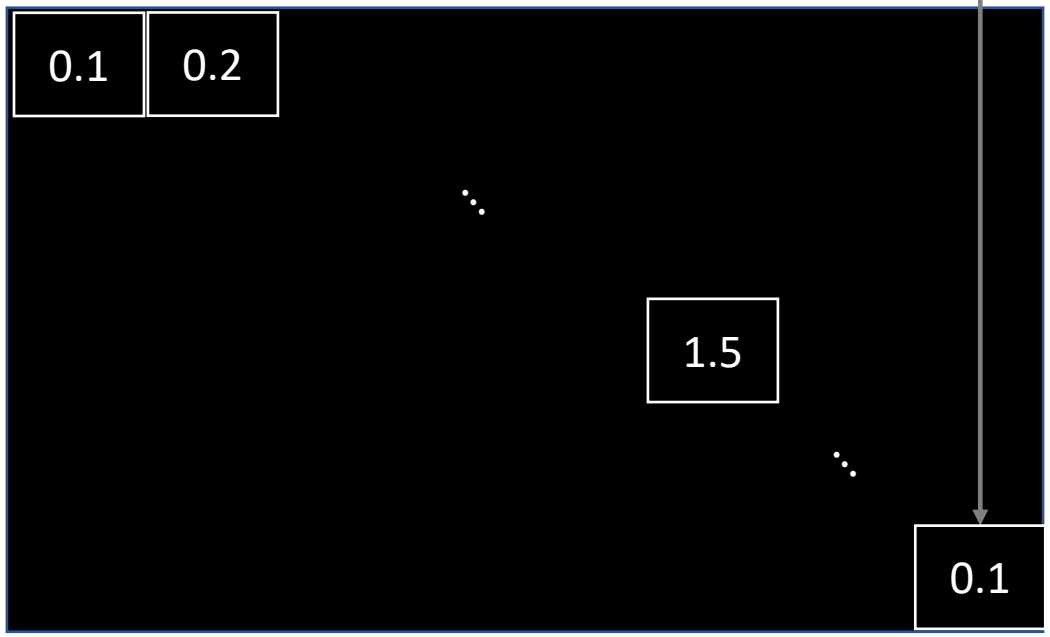
0.1 0.2 ⋱ 1.5

$$\begin{bmatrix} 0.10 \\ 0.70 \\ \vdots \\ 0.67 \end{bmatrix} \cdot \begin{bmatrix} 0.79 \\ 0.32 \\ \vdots \\ 0.37 \end{bmatrix}$$

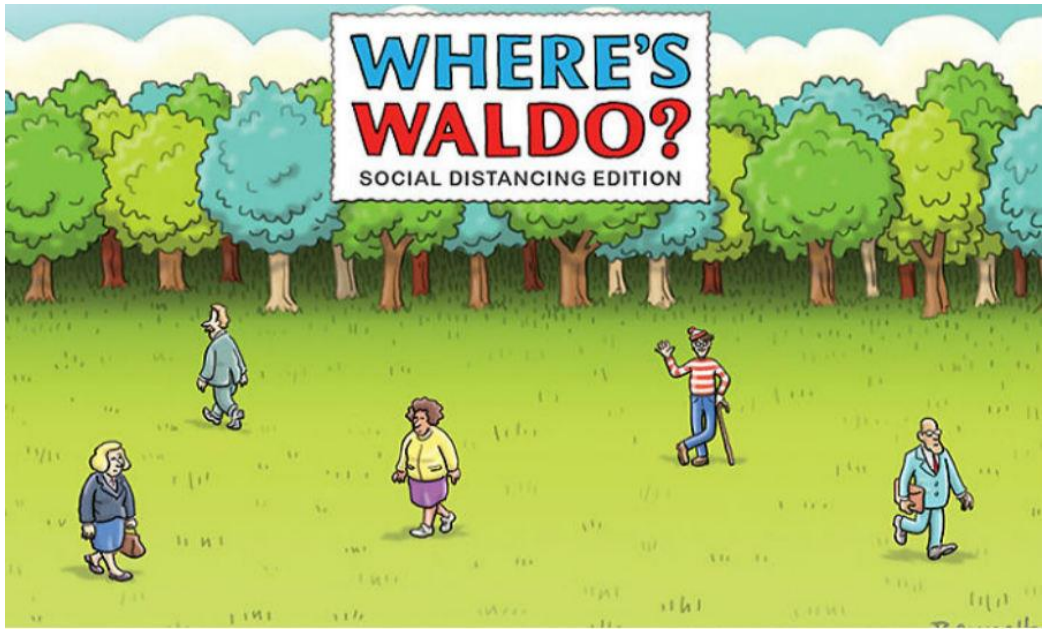




Model

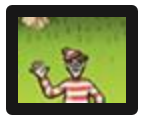


- Output of convolution (strictly speaking – correlation) with the template “filter” will have a peak at the location of the most similar object
- Mathematically, $Z = I * W$

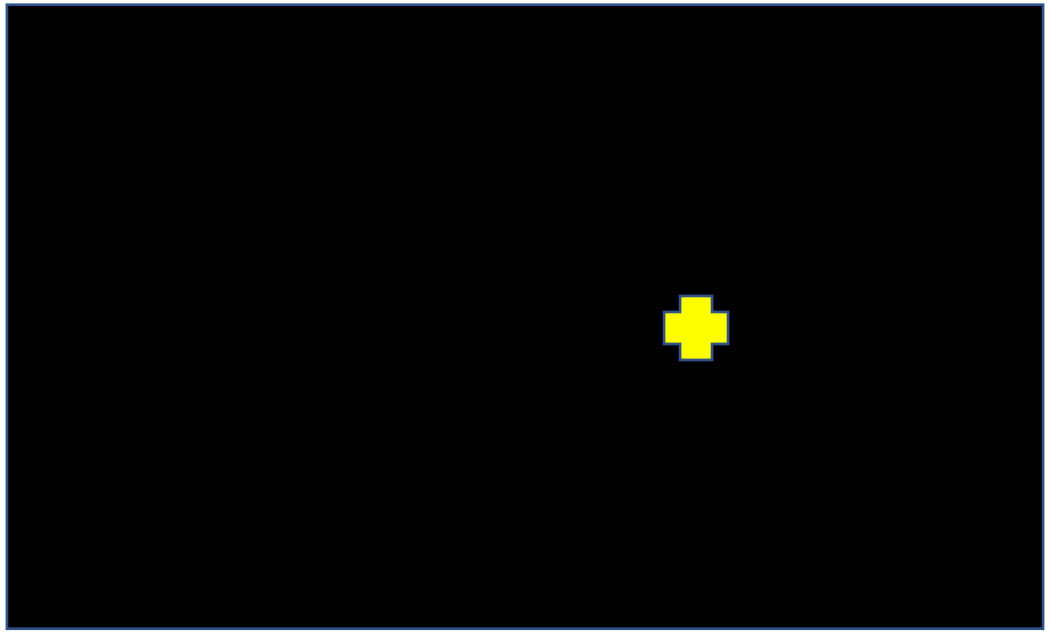


I

Model



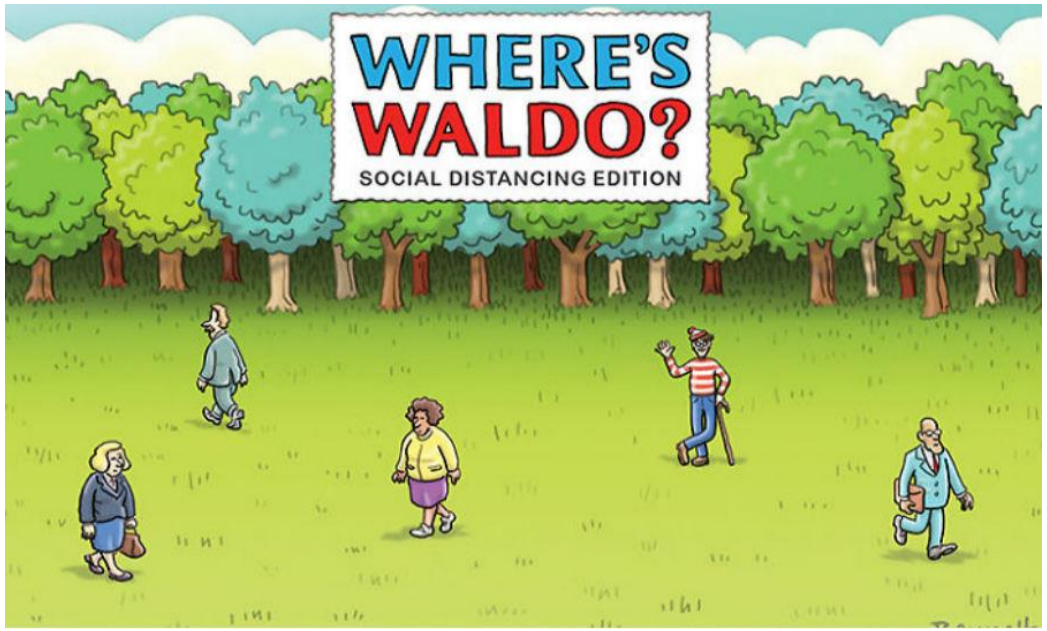
W



Z

- What if I don’t have a cut out of Waldo but I do have a target image specifying where the peak should be?

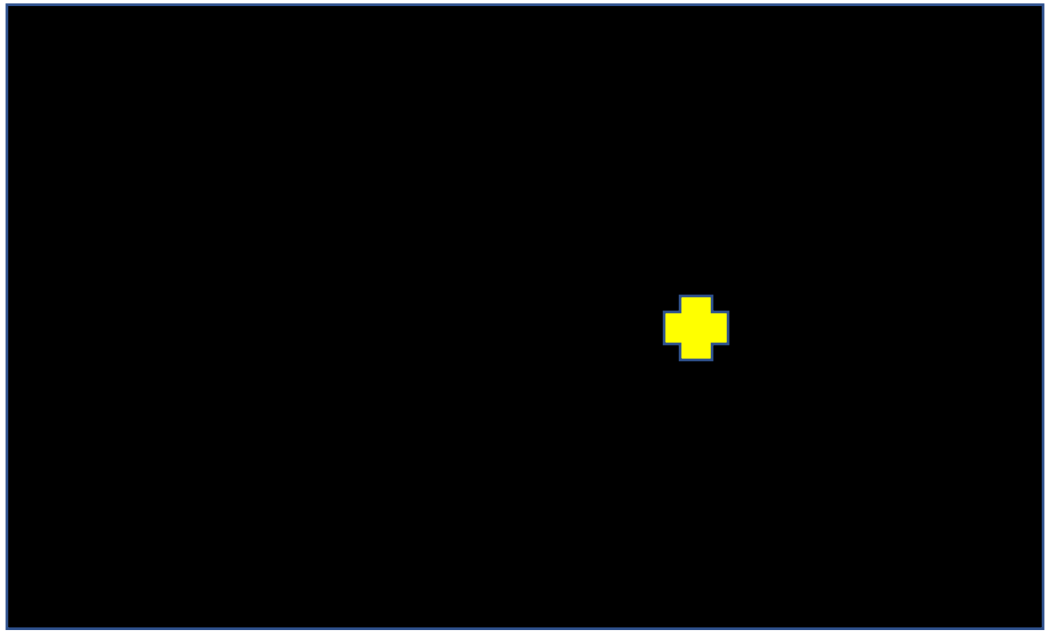
- Can we find him still?



I

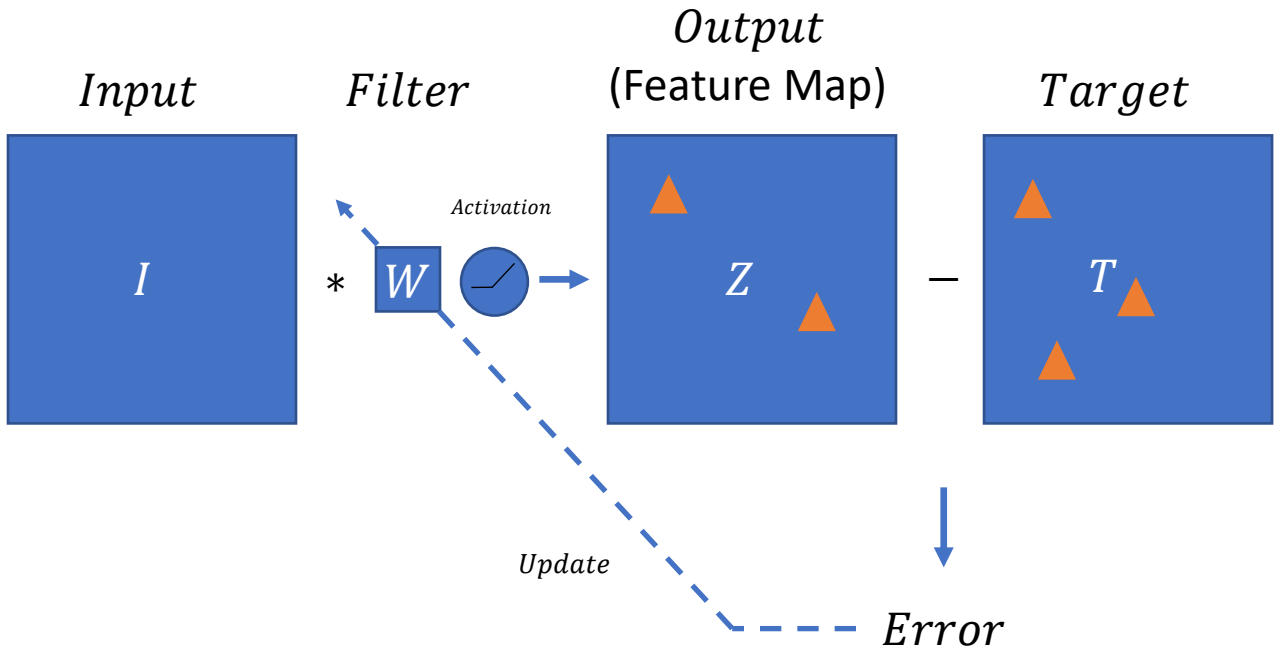
Model

$W = ?$



Z

(Very) basic "convolutional neural network"

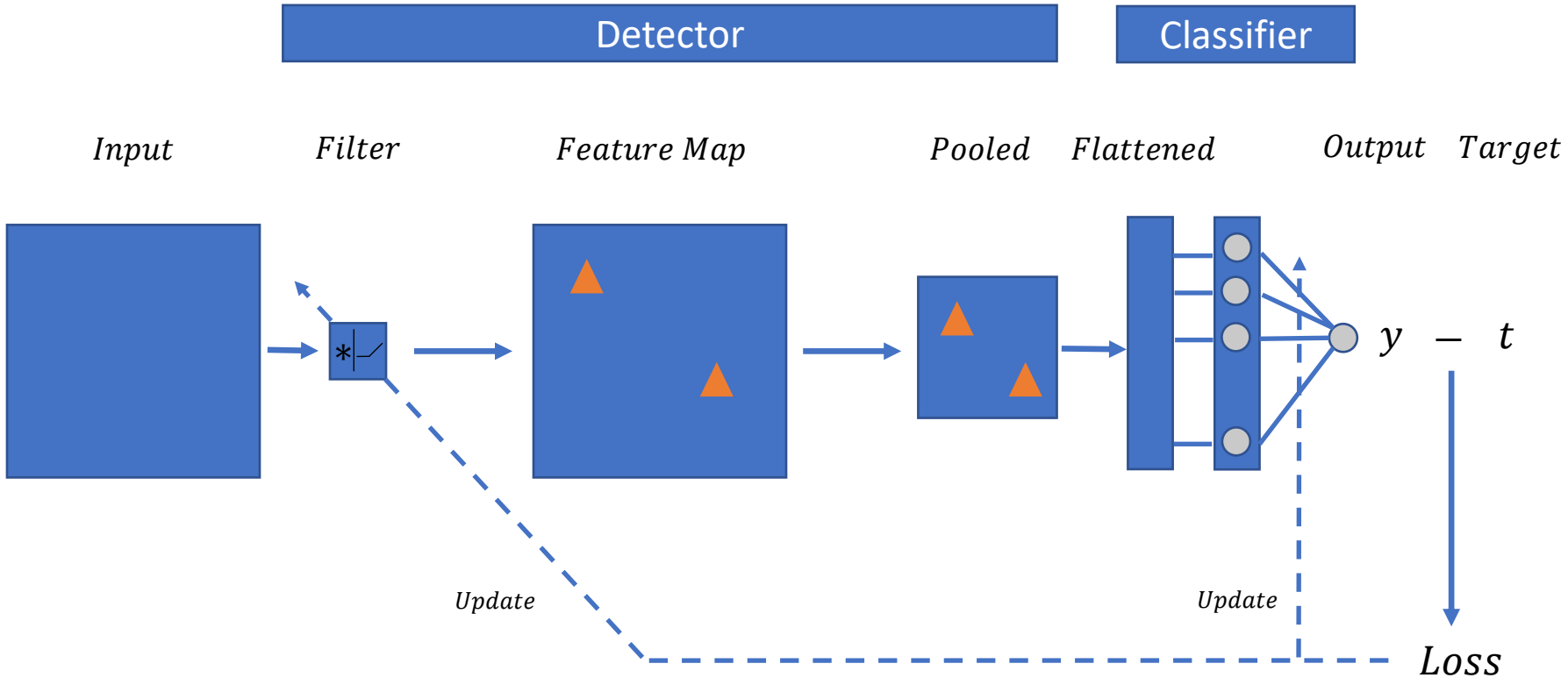


- Technically, this is a single layer in a Fully Convolutional Neural Network
  - **Intuition: Each "filter" learns to detect a specific "local" pattern**
- The **representation** of a convolutional layer is: $Z = a(I * W)$ different from MLPs
  - More efficient as W is much smaller
- The complete learning problem with **Evaluation** and **Optimization** is: $\min_W \|a(I * W) - T\|$

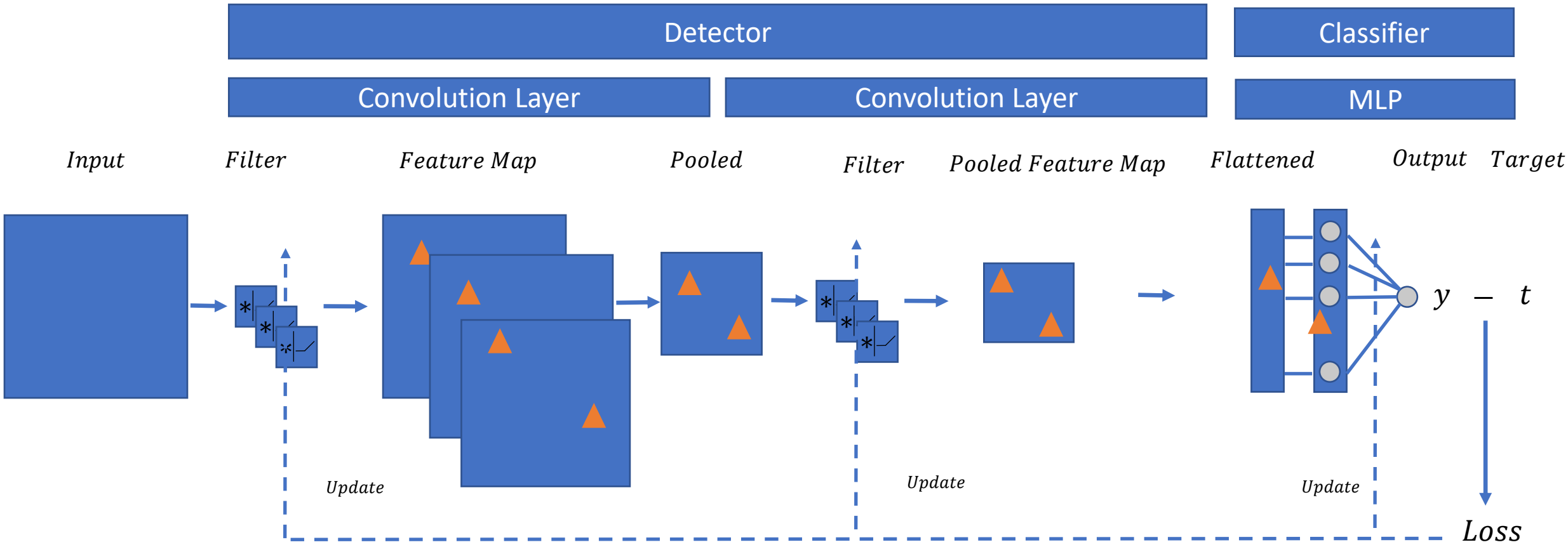
MLP Layer: $Z = a(WX)$
Convolutional Layer: $Z = a(I * W)$

https://github.com/foxtrotmike/CS909/blob/master/learn_filters.ipynb
https://github.com/foxtrotmike/CS909/blob/master/cnn_mnist_pytorch.ipynb

If we want to classify images…

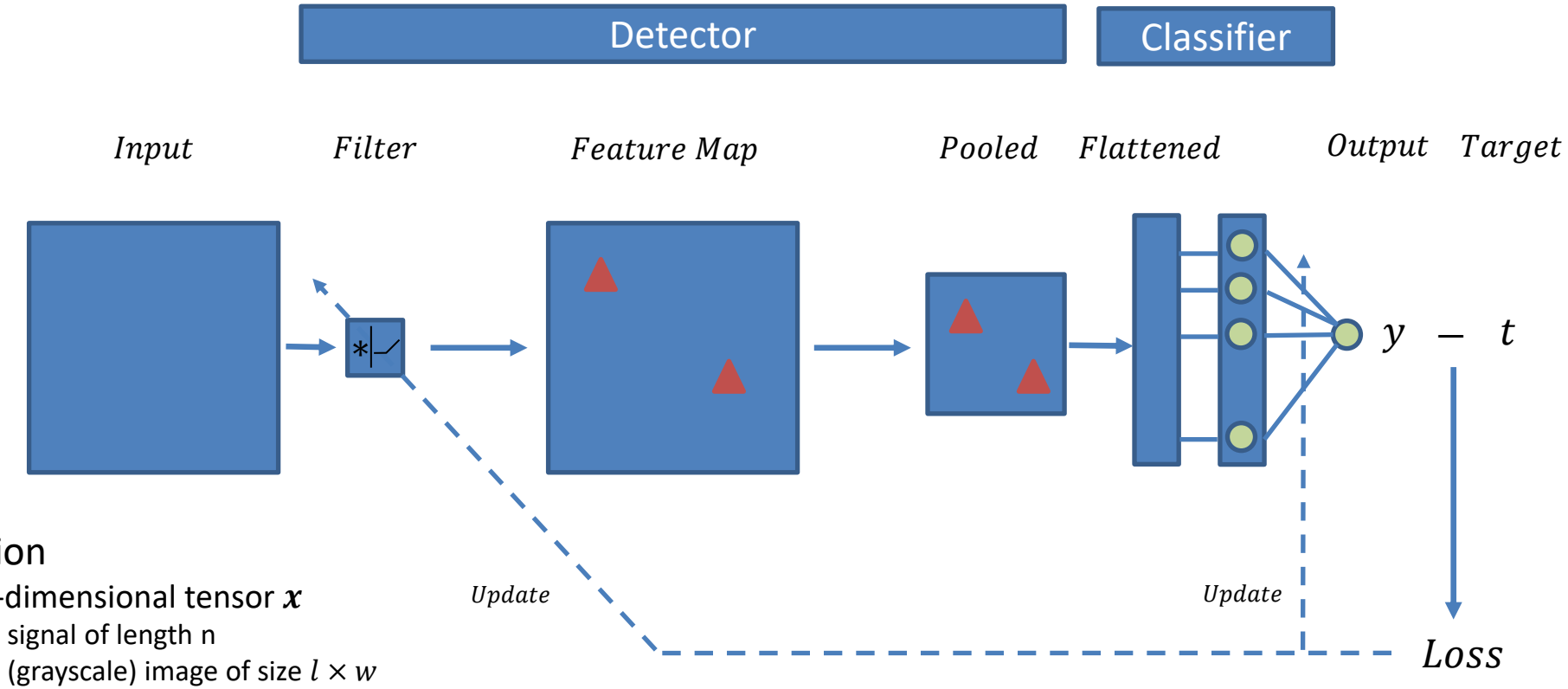


CNNs

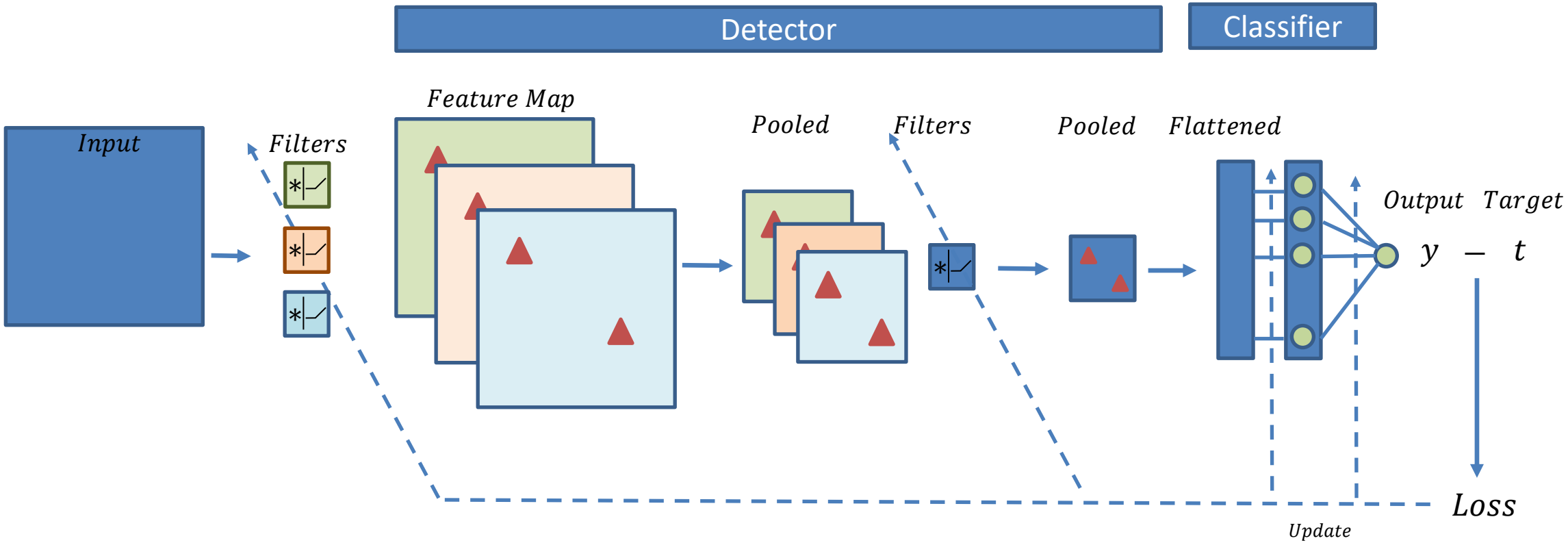


https://poloclub.github.io/cnn-explainer/

REO for a convolution neural network



- Representation
  - Input: a k-dimensional tensor $\boldsymbol{x}$
    - k = 1: signal of length n
    - k = 2: (grayscale) image of size $l \times w$
      - RGB channel image: $l \times w \times 3$
    - k = 3: $l \times w \times t$ video of frame size $l \times w$ with duration t
  - Output: A decision score y $= f(x; \boldsymbol{\theta})$ (can be multi-dimensional as well)
  - Structure
    - **Layers of Learnable filters each of which is correlated (or convolved) with the input tensor in parallel followed by convolution with other filters**
      - A single convolution is indicated by $\boldsymbol{z} = \boldsymbol{a}(\boldsymbol{x} \star \boldsymbol{\theta})$ where $\boldsymbol{\theta}$ is the representation of a single filter and $\boldsymbol{a}(\cdot)$ is an activation function. Filters are much smaller than $\boldsymbol{x}$.
      - Implemented as layers: Conv1d, Conv2d, Conv3d (in PyTorch)
    - The correlation output is then pooled (optional)
    - Nonlinear activation functions are applied
    - Aggregated to produce the final output (depending upon application)


Detector
Classifier
Input
Filters
Feature Map
Pooled
Filters
Pooled
Flattened
Output
Target
y
–
t
Loss
Update

Lab Exercise

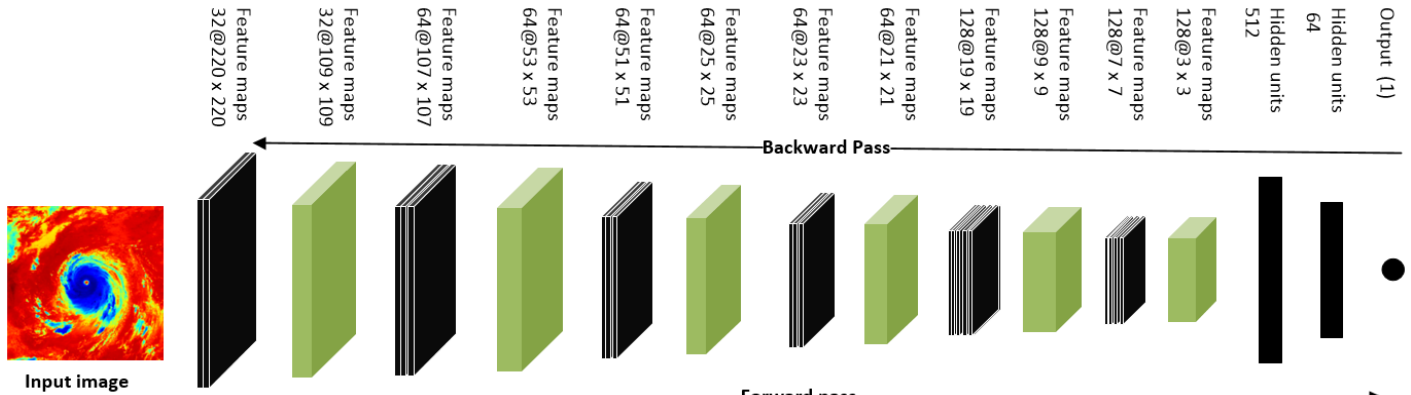
- The concept of filters, convolution and learning filters in PyTorch leading to the simplest possible convolutional neural network
- https://github.com/foxtrotmike/CS909/blob/master/learn_filters.ipynb

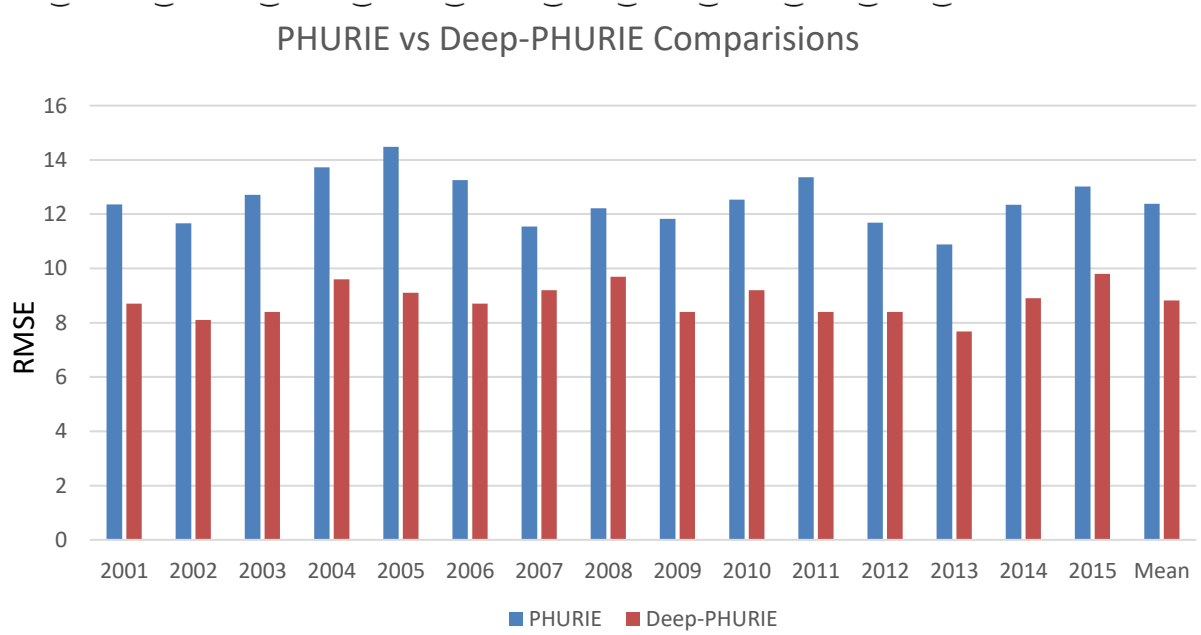
- Using Convolutional neural networks in Pytorch for digit classification

- https://github.com/foxtrotmike/CS909/blob/master/cnn_mnist_pytorch.ipynb

Predicting Hurricane Intensities

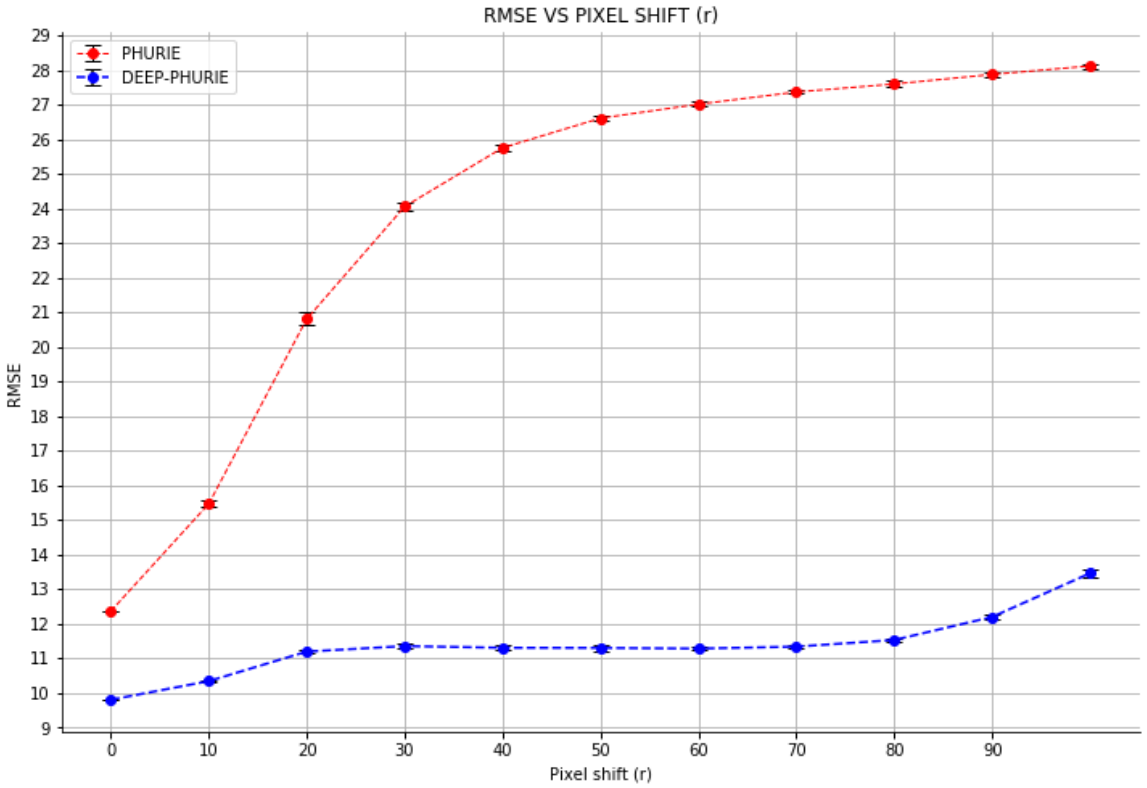
- Deep-PHURIE

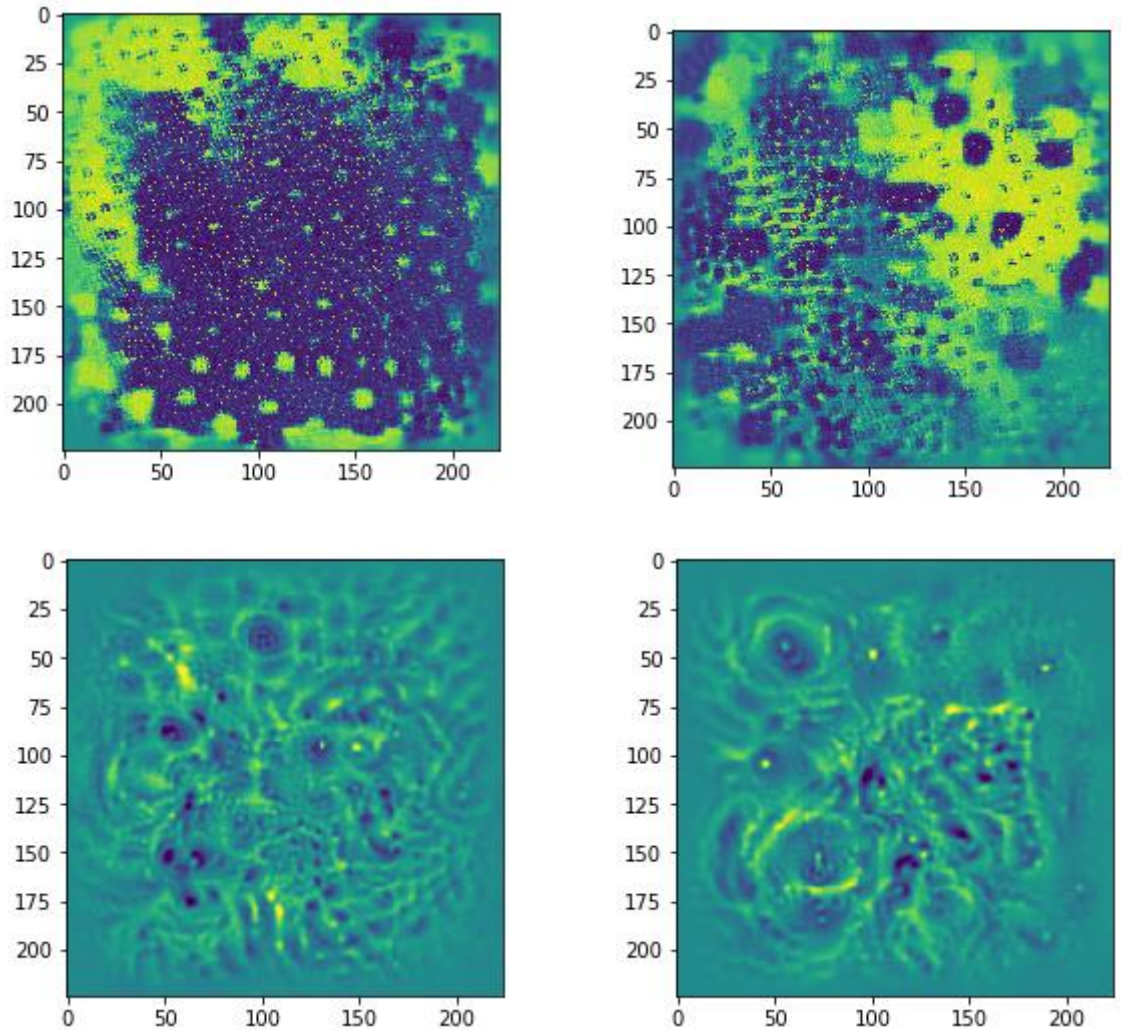




https://link.springer.com/article/10.1007/s00521-019-04410-7

Deep-PHURIE Robustness Analysis

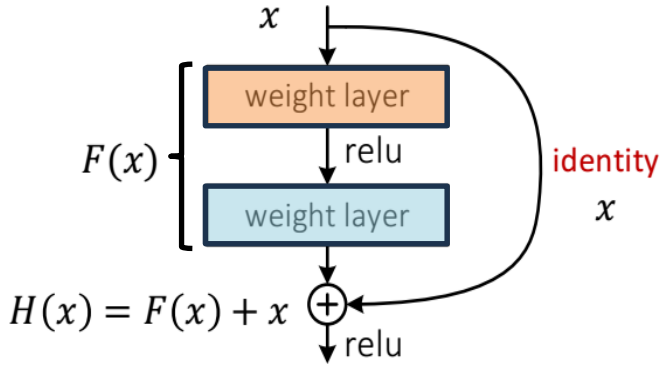


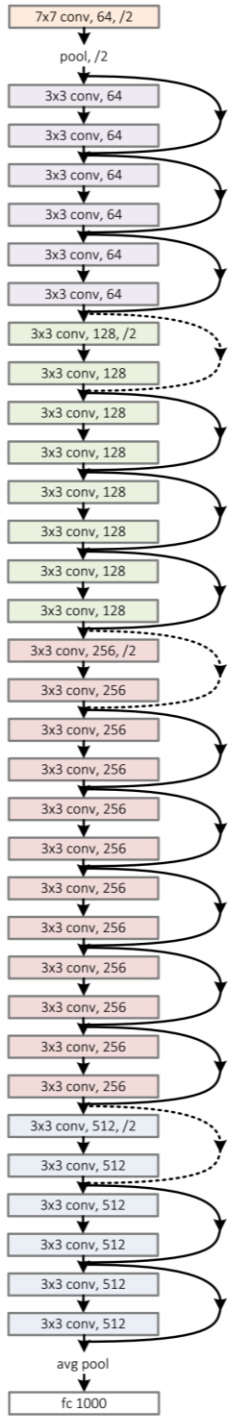


Activation Maps for Deep PHURIE

Deep Residual Neural Networks for Image Recognition

```
class ResidualBlock(nn.Module):
    def __init__(self, in_channels, out_channels, stride=1, downsample=None):
        super(ResidualBlock, self).__init__()
        self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size=3, stride=stride, padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(out_channels)
        self.relu = nn.ReLU(inplace=True)
        self.conv2 = nn.Conv2d(out_channels, out_channels, kernel_size=3, stride=1, padding=1, bias=False)
        self.bn2 = nn.BatchNorm2d(out_channels)
        self.downsample = downsample

    def forward(self, x):
        residual = x
        out = self.conv1(x)
        out = self.bn1(out)
        out = self.relu(out)
        out = self.conv2(out)
        out = self.bn2(out)
        # downsample only if dimensions of x and F(x) don't match
        if self.downsample:
            residual = self.downsample(x)
        out += residual
        out = self.relu(out)
        return out
```





ResNet

Practical convolutional networks use "skip" connections to improve performance

Strongly recommended: How to use a minimalistic residual network for MNIST Classification

https://github.com/foxtrotmike/CS909/blob/master/resnet_mnist.ipynb

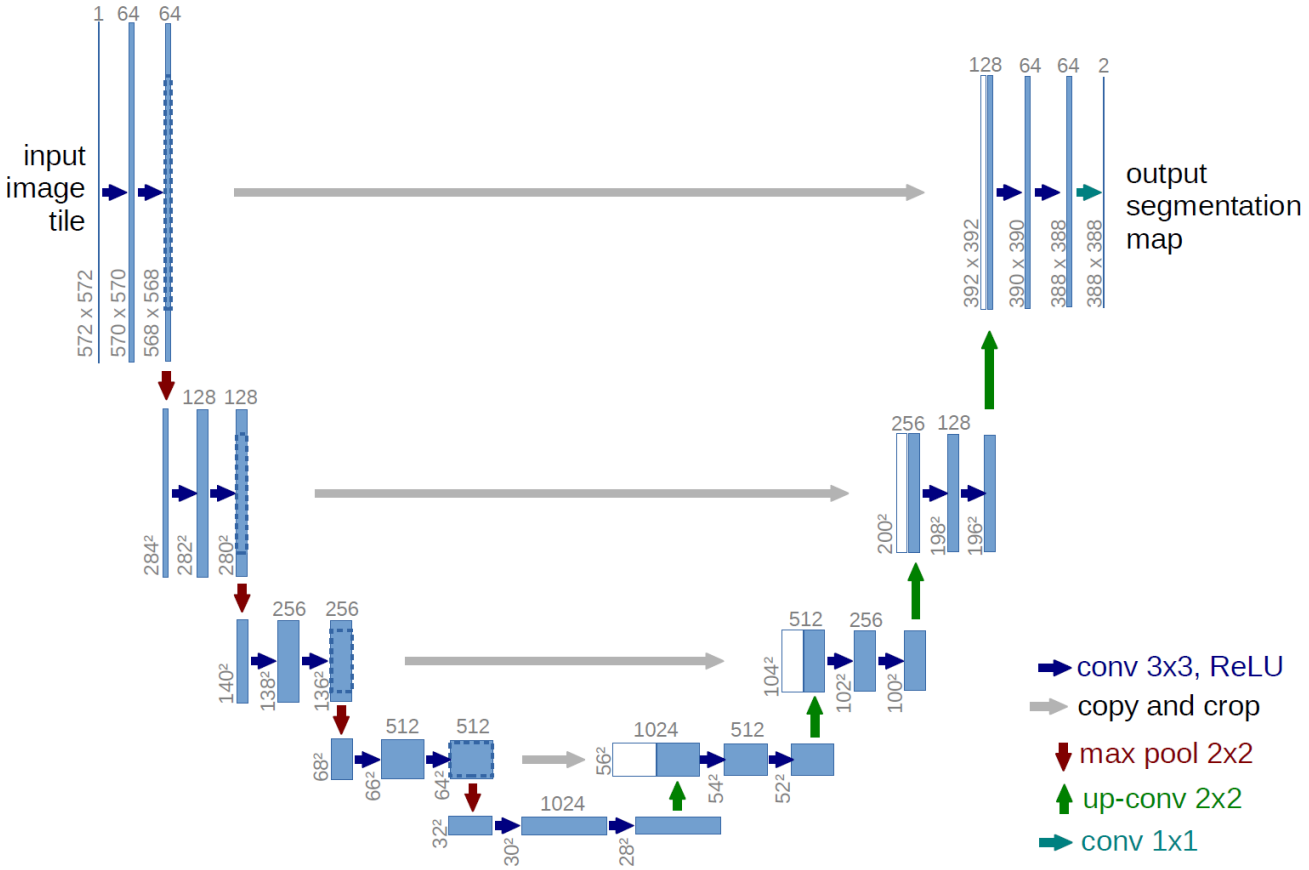
Required Reading: Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". CVPR 2016.

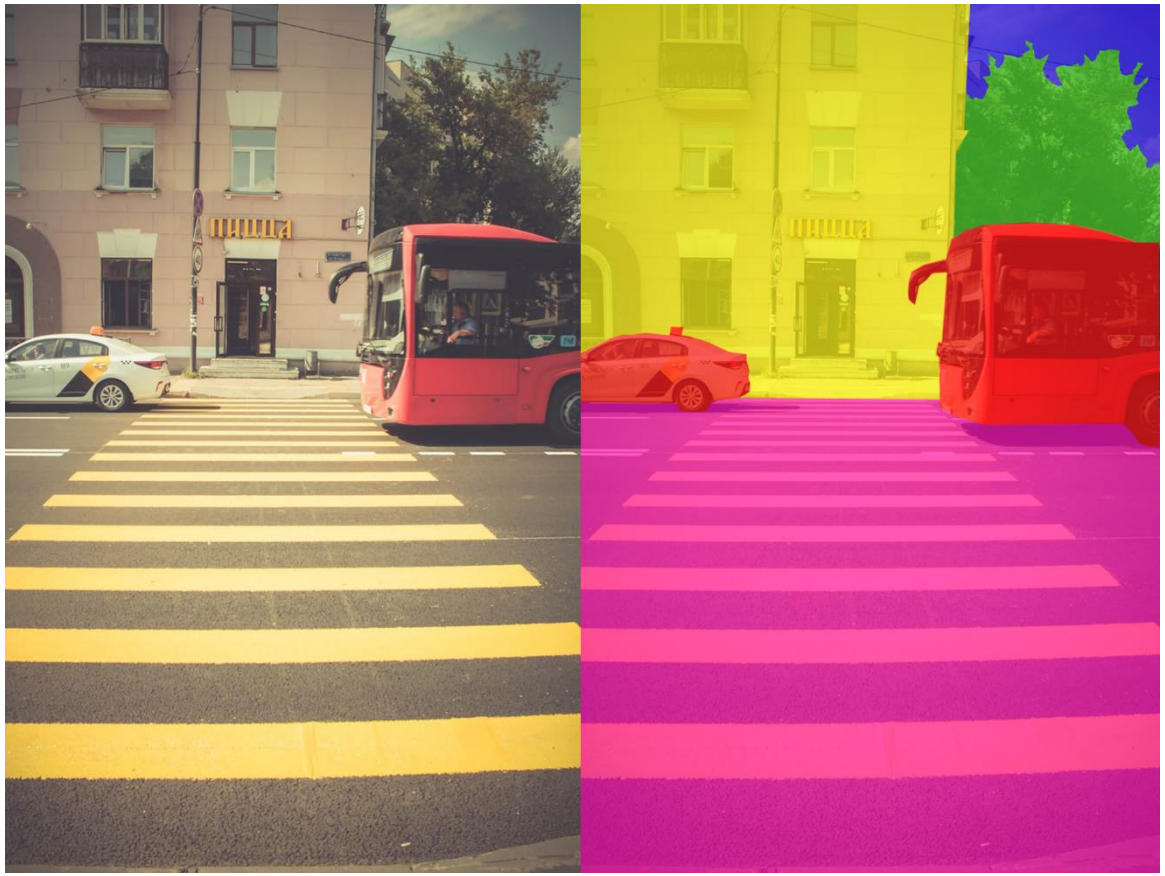
Many third-party implementations

list in https://github.com/KaimingHe/deep-residual-networks

Torch ResNet: https://github.com/pytorch/examples/tree/master/imagenet

U-Net for Segmentation





YOLO

- Convolution
- Residual Architecture
- Reversible function to allow preservation of relevant information
- Programmable gradient information

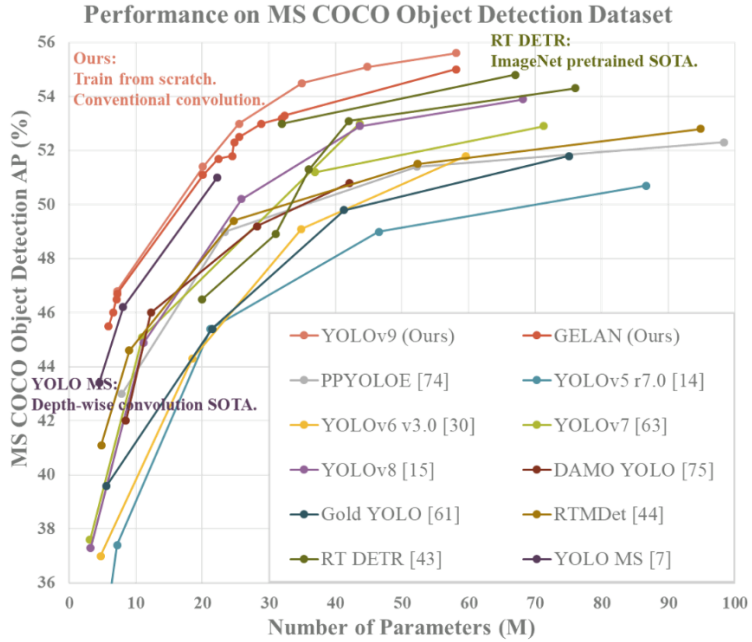


Figure 1. Comparisons of the real-time object detecors on MS COCO dataset. The GELAN and PGI-based object detection method surpassed all previous train-from-scratch methods in terms of object detection performance. In terms of accuracy, the new method outperforms RT DETR [43] pre-trained with a large dataset, and it also outperforms depth-wise convolution-based design YOLO MS [7] in terms of parameters utilization.

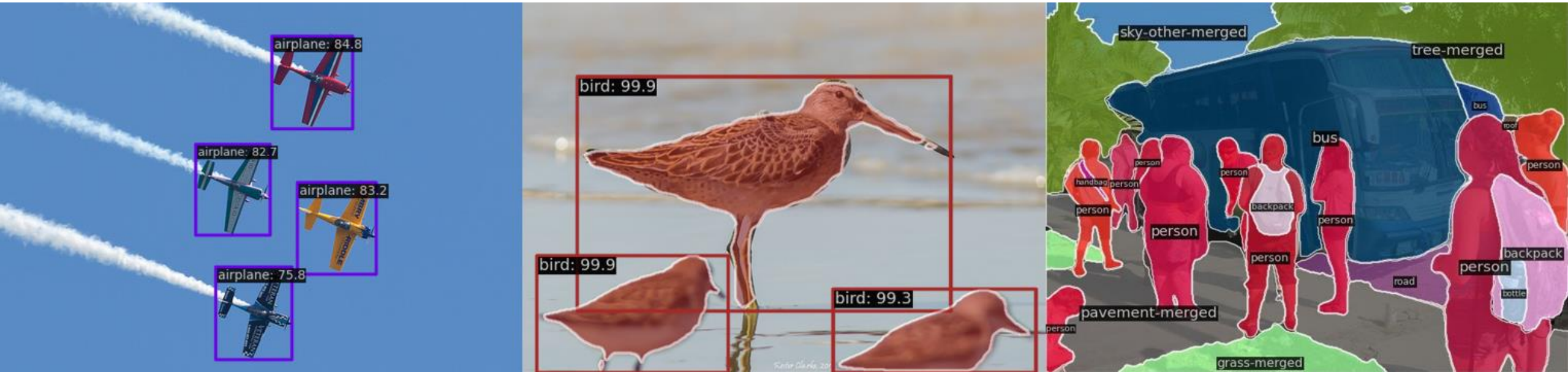
Wang, Chien-Yao, I.-Hau Yeh, and Hong-Yuan Mark Liao. “YOLOv9: Learning What You Want to Learn Using Programmable Gradient Information.” arXiv, February 21, 2024. https://doi.org/10.48550/arXiv.2402.13616.

Segmentation Algorithm Libraries

- https://github.com/open-mmlab/mmsegmentation

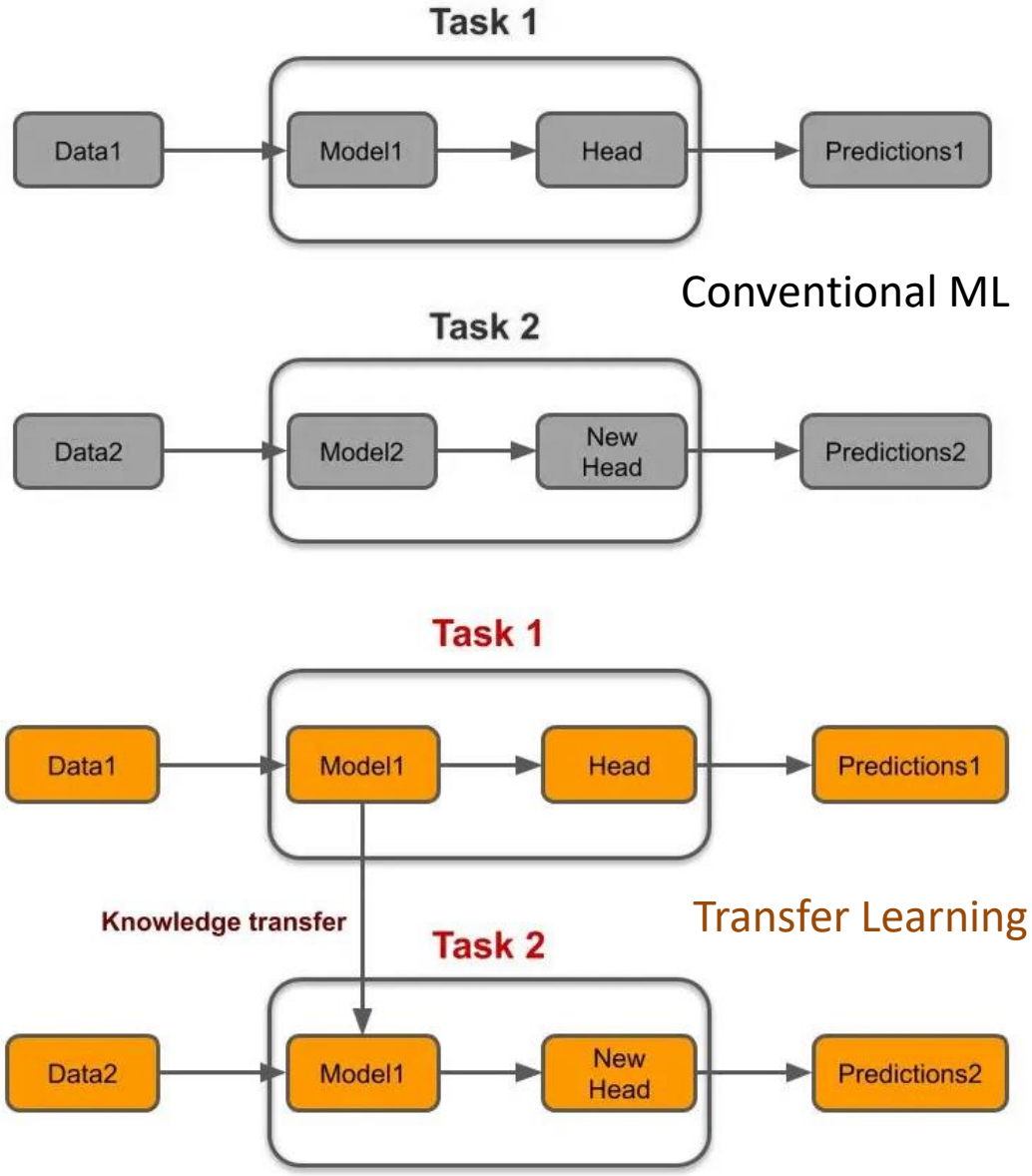
Detection algorithm libraries

- https://github.com/open-mmlab/mmdetection

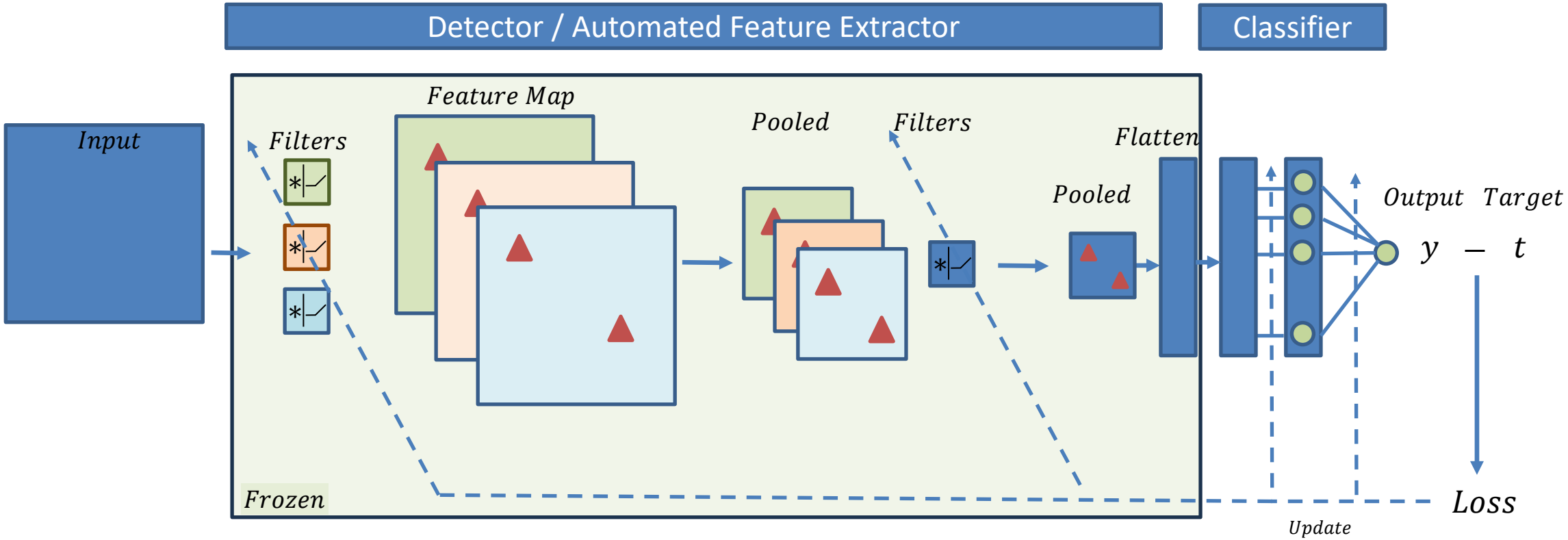


Transfer Learning and Fine Tuning

- Use a pretrained network for a task
- Keep the convolutional layers fixed (frozen) to solve a different problem
- Freezing layers
  - `for param in vgg.features.parameters(): param.requires_grad = False`
- **Transfer Learning:** Train the last layers (fully connected) for your task and/or add more layers as needed
- **Fine tuning:** Modify the weights of a few convolutional layers too



https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html
https://jimmy-shen.medium.com/pytorch-freeze-part-of-the-layers-4554105e03a6#
Transfer Learning with ResNet: https://www.pluralsight.com/guides/introduction-to-resnet



We can choose which layers to freeze depending upon the application and the level of similarity between tasks

Advanced: Adapters

- Generalize the concept of transfer learning

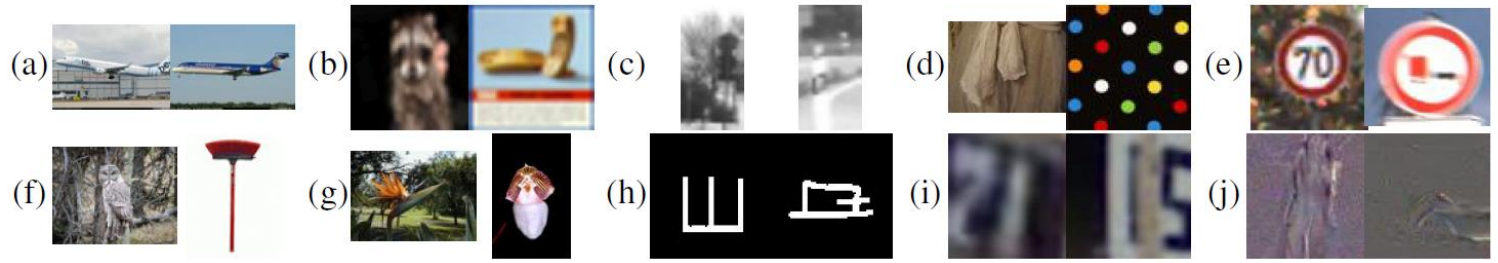


Figure 1: **Visual Decathlon.** We explore deep architectures that can learn simultaneously different tasks from very different visual domains. We experiment with ten representative ones: (a) Aircraft, (b) CIFAR-100, (c) Daimler Pedestrians, (d) Describable Textures, (e) German Traffic Signs, (f) ILSVRC (ImageNet) 2012, (g) VGG-Flowers, (h) OmniGlot, (i) SVHN, (j) UCF101 Dynamic Images.

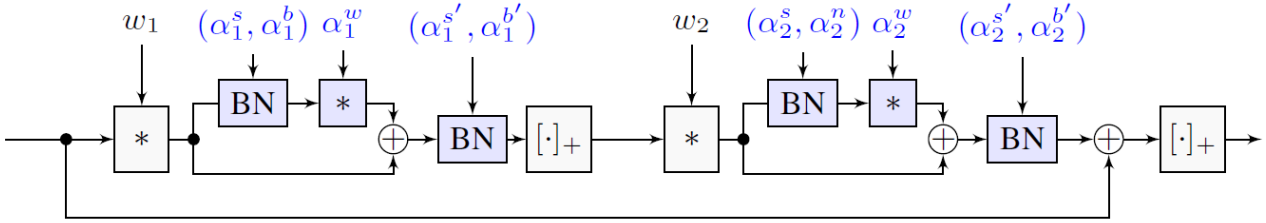


Figure 2: **Residual adapter modules.** The figure shows a standard residual module with the inclusion of adapter modules (in blue). The filter coefficients (w_1, w_2) are domain-agnostic and contains the vast majority of the model parameters; (α_1, α_2) contain instead a small number of domain-specific parameters.

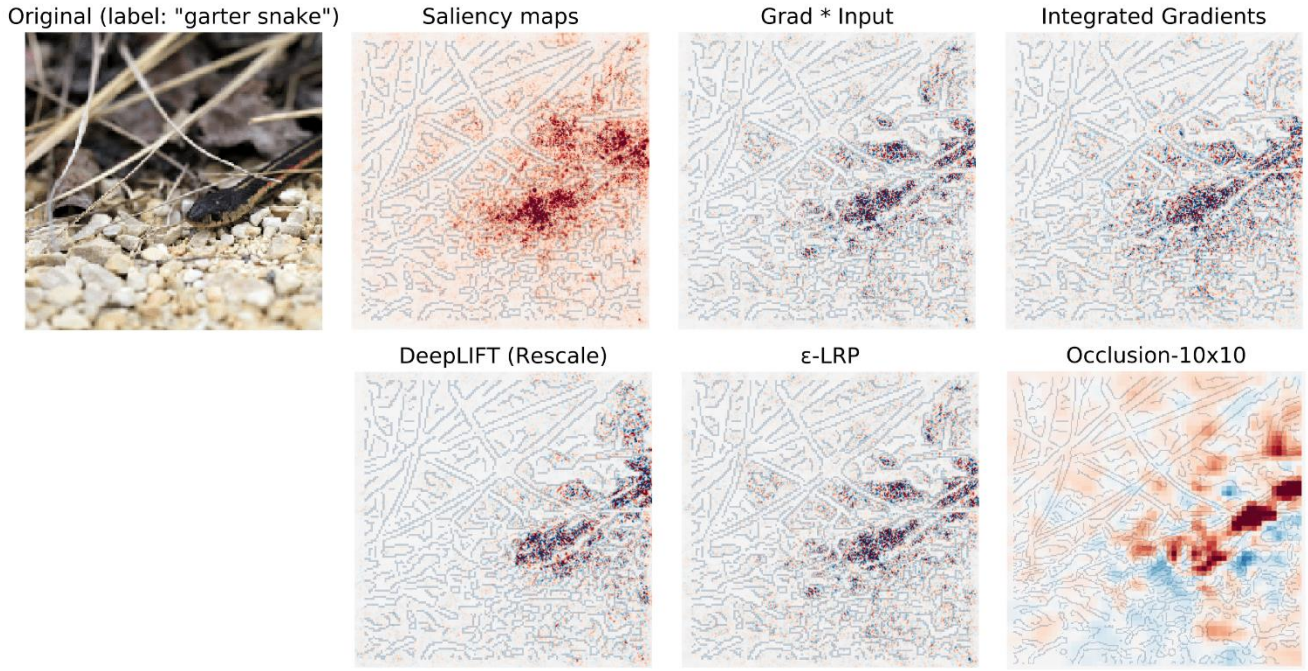
Rebuffi, Sylvestre-Alvise, Hakan Bilen, and Andrea Vedaldi. "Learning Multiple Visual Domains with Residual Adapters." arXiv, November 27, 2017. https://doi.org/10.48550/arXiv.1705.08045.

Why did my model produce a certain output? What is my model learning?

- **Interpretability**
  - Interpret why a certain model is producing a certain output for a given input
  - "What is the model doing?"

- **Explainable**
  - Explaining the "behavior" of the model across different inputs
  - "What is the model learning?"

- Model Agnostic Methods
  - Permutation Feature Invariance
  - LIME Analysis
  - SHAP Analysis
- **For CNNs**
  - Pixel Attribution (Saliency Maps)
    - Score-CAM
    - Grad-CAM
  - Testing with Concept Activation Vectors (TCAV)
  - DeepSHAP

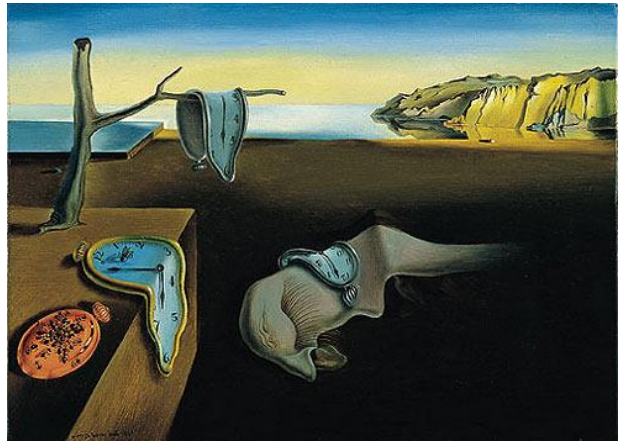


Great Resource on interpretable machine learning:
https://christophm.github.io/interpretable-ml-book/

https://github.com/marcoancona/DeepExplain

What are the limitations of CNNs

- Each filter learns a specific local feature
- Consecutive layers share or build up information across different parts
- May not be the most efficient way of ensuring effective “contextual” representations and long-range interactions



In addition to specific local features, global information or context is needed to develop effective representations for various task

TRANSFORMERS

Transformers

- Very useful and popular architecture for vision tasks though originally built for natural language processing
- Use “attention mechanism” to integrate information from different components of an input in a weighted manner to produce an output representation for the input that can be passed to predictor to generate predictions

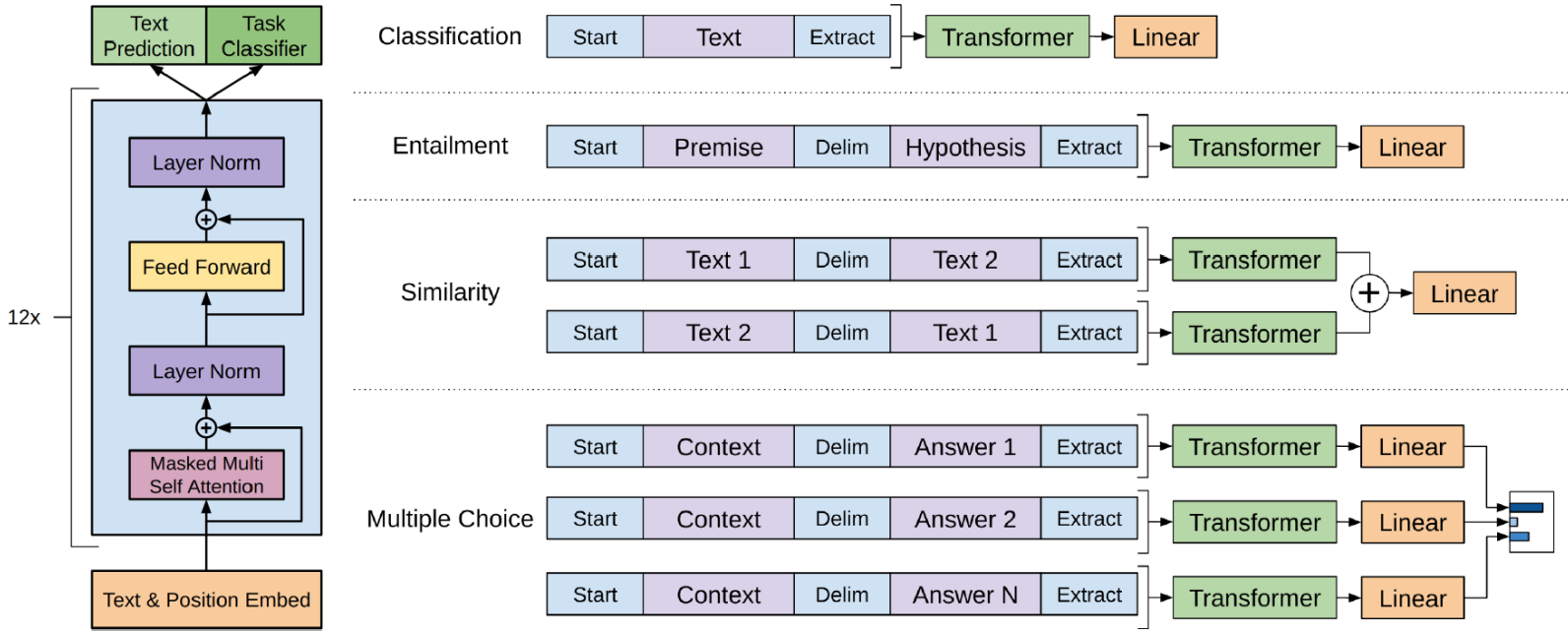
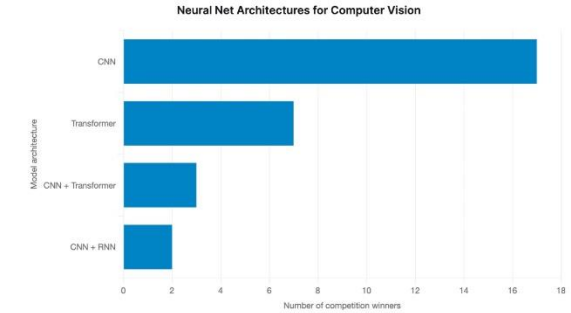


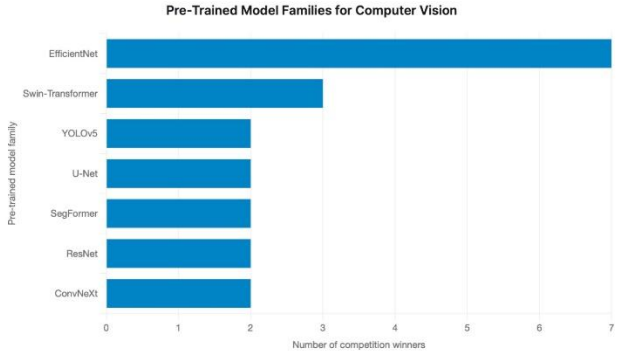
Figure from the **Generative Pre-trained Transformer (GPT)** paper

Radford, Alec, et al (OpenAI). “Improving Language Understanding by Generative Pre-Training,” 2018.

Convolutional neural networks still dominate computer vision



EfficientNet is the most popular pretrained architecture for computer vision

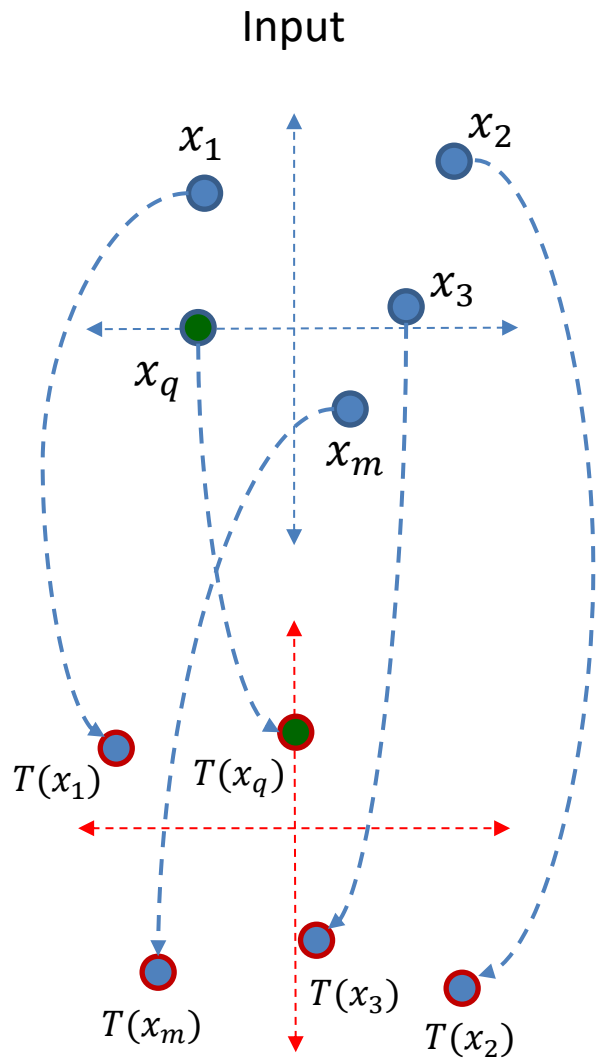


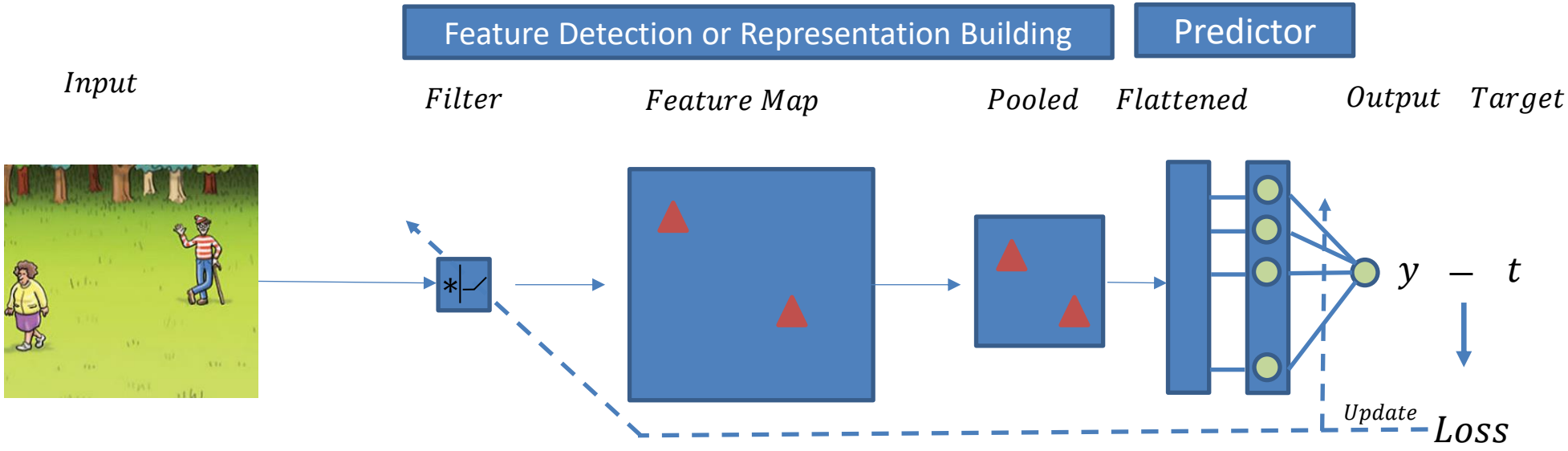
https://twitter.com/rasbt/status/1634564282535878661/photo/1

Background

- **Transformations**: $v(x;\theta): R^d \rightarrow R^{d'}$
  - Explicitly transform a point to a different feature space
  - Any layer in a neural network can be considered as performing a transformation
- A **kernel** $k(\boldsymbol{a}, \boldsymbol{b})$ is a generalized dot-product or a way of quantifying the degree of similarity between two examples or objects

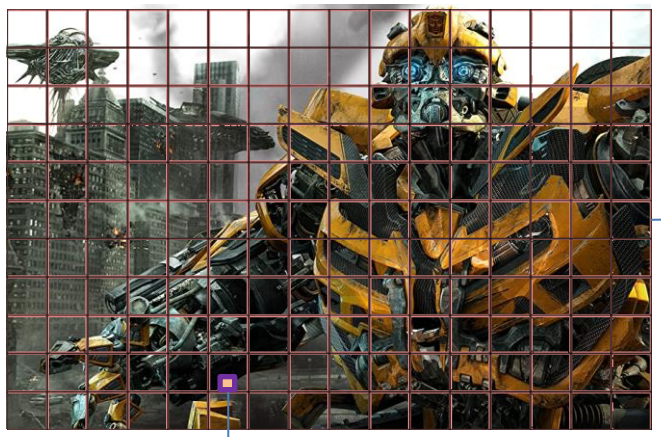
Kernel
Linear: $k(\boldsymbol{a}, \boldsymbol{b}) = \boldsymbol{a}^T\boldsymbol{b}$
Polynomial degree 2: $k(\boldsymbol{a}, \boldsymbol{b}) = (\boldsymbol{a}^T\boldsymbol{b})^2$ (Homogeneous)
Polynomial degree 2: $k(\boldsymbol{a}, \boldsymbol{b}) = (\boldsymbol{a}^T\boldsymbol{b} + 1)^2$
RBF Kernel: $k(\boldsymbol{a}, \boldsymbol{b}) = \exp(-\gamma\|\boldsymbol{a} - \boldsymbol{b}\|^2)$



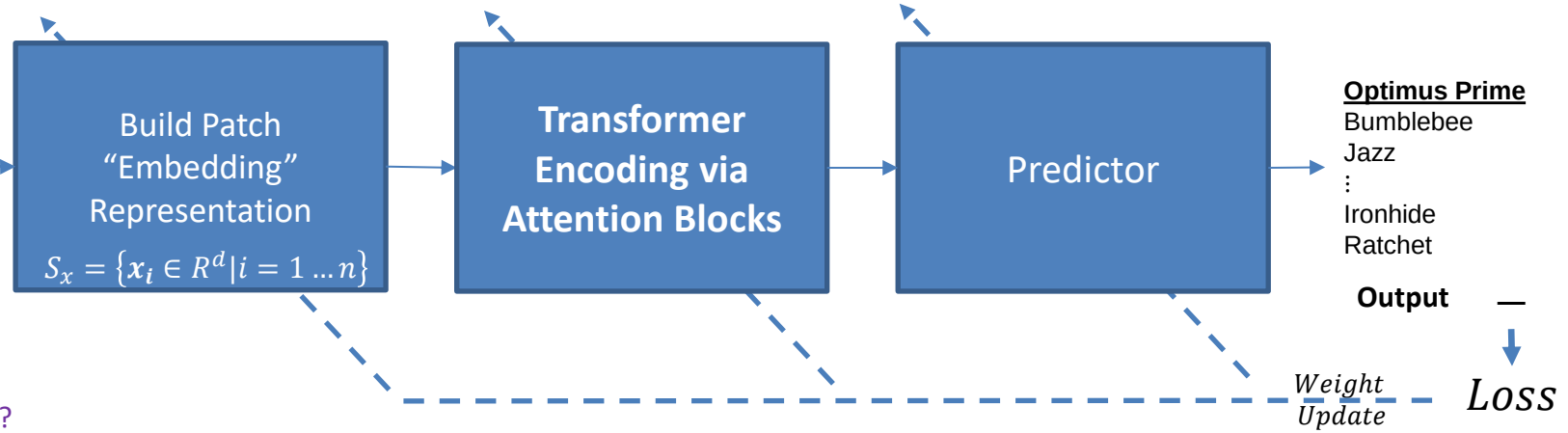


Convolutional Neural Network

(Vision) Transformers

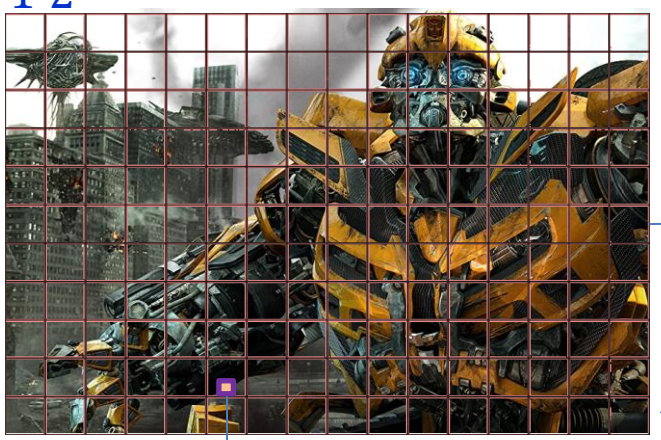


$x_i \equiv \phi(f_i, t_i)$ Feature Embedding: What is it?
Positional Embedding: Where is it?



Dosovitskiy, Alexey, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, et al. "An Image Is Worth 16x16 Words: Transformers for Image Recognition at Scale." arXiv, June 3, 2021. https://doi.org/10.48550/arXiv.2010.11929.

(Vision) Transformers (for classification)



n tokens/patches

$\boldsymbol{x_i} \equiv \phi(\boldsymbol{f_i}, \boldsymbol{t_i})$ Feature Embedding: What is it?
Positional Embedding: Where is it?

Build Patch "Embedding" Representation
$S_x = \{\boldsymbol{x_i} \in R^d | i = 1 \ldots n\}$

Transformer Encoding via Attention Blocks

Predictor

Optimus
Bumblebee
Jazz
⋮
Ironhide
Ratchet

Optimus
Bumblebee
Jazz
⋮
Ironhide
Ratchet

Output — **Target**

$Loss$

$Weight\ Update$

Building an integrated representation of how components form the overall object

A transformer that can transform into a yellow car is called _____.

$\boldsymbol{x_i} \equiv \phi(\boldsymbol{f_i}, \boldsymbol{t_i})$ Feature Embedding: What is it?
Positional Embedding: Where is it?

Build Token/Word "Embedding" Representation
$S_x = \{\boldsymbol{x_i} \in R^d | i = 1 \ldots n\}$

Transformer Encoding via Attention Blocks

Predictor

Optimus
Bumblebee
Jazz
⋮
Ironhide
Ratchet

Optimus
Bumblebee
Jazz
⋮
Ironhide
Ratchet

Output — **Target**

$Loss$

$Weight\ Update$

(NLP) Transformers (for next word prediction)

Simplest: $\phi(\boldsymbol{f_i}, \boldsymbol{t_i}) = \boldsymbol{f_i} + \boldsymbol{t_i}$

What is attention and why do you need it?

[Submitted on 12 Jun 2017 (v1), last revised 6 Dec 2017 (this version, v5)]

Attention Is All You Need

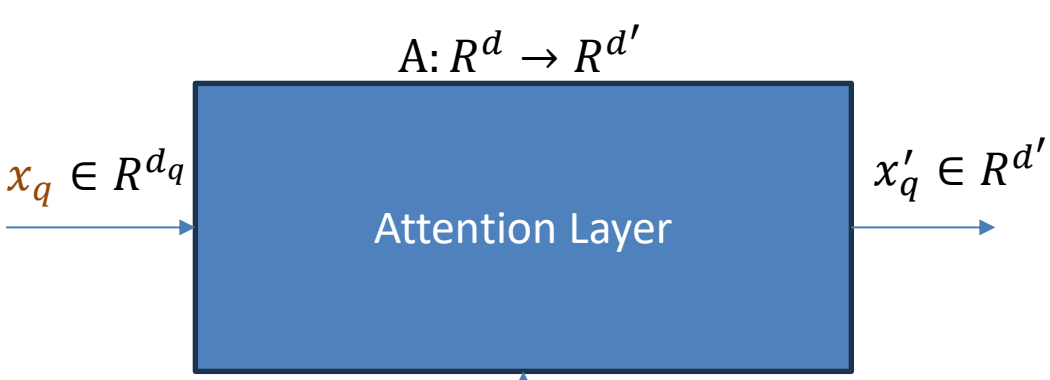
Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks in an encoder-decoder configuration. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.

We are going to have a no **gobbledygook** introduction to attention (using the paper below)!

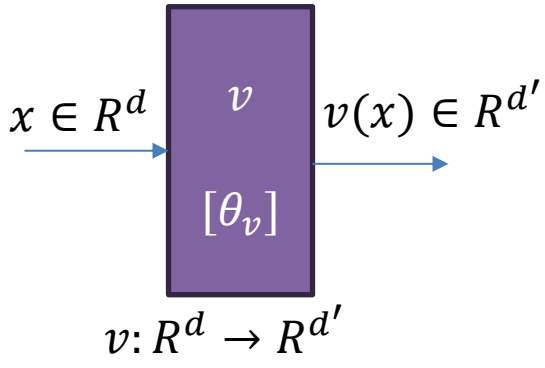
Tsai, Yao-Hung Hubert, Shaojie Bai, Makoto Yamada, Louis-Philippe Morency, and Ruslan Salakhutdinov. "**Transformer Dissection: A Unified Understanding of Transformer's Attention via the Lens of Kernel**." *ArXiv:1908.11775 [Cs, Stat]*, November 11, 2019. http://arxiv.org/abs/1908.11775.

General Attention Building Blocks

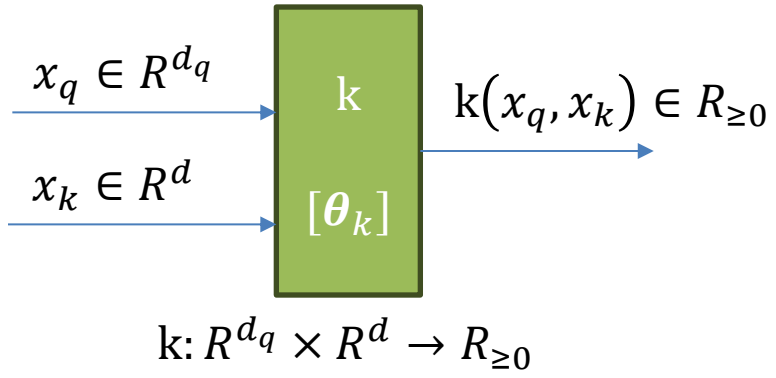


Let simplify and understand transformers (Masked out)

Transformation



Learnable "attention" kernel



How similar are x_q and x_k relative to similarity of x_q to all tokens

$$x'_q = A(x_q; M(x_q, S_{x_k}); \boldsymbol{\theta}) = \sum_{x_k \in M(x_q, S_{x_k})} \mathrm{a}(x_q, x_k ; M(x_q, S_{x_k}), \boldsymbol{\theta_k})\, v(x_k; \boldsymbol{\theta_v}) = \sum_{x_k \in M(x_q, S_{x_k})} \frac{\mathrm{k}(x_q, x_k; \boldsymbol{\theta_k})}{\sum_{x_{k'} \in M(x_q, S_{x_k})} \mathrm{k}(x_q, x_{k'}; \boldsymbol{\theta_k})}\, v(x_k; \boldsymbol{\theta_v})$$

Input:

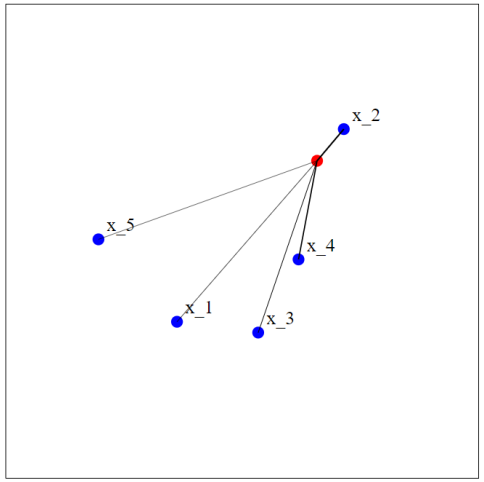
A "query" token $x_q \in R^{d_q}$ representation of a component (patch or token) which will be transformed. In turn, all tokens will take the role of the query token in classical attention.
A set of "key" tokens $S_{x_k} = \{x_1, x_2, \dots, x_m\}$. Can come from a different source (e.g., as in cross-attention).

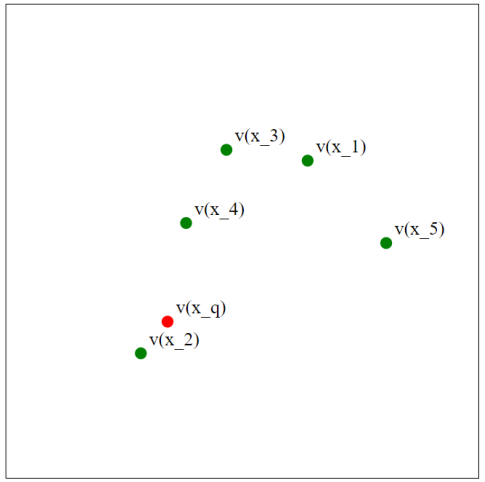
Output:

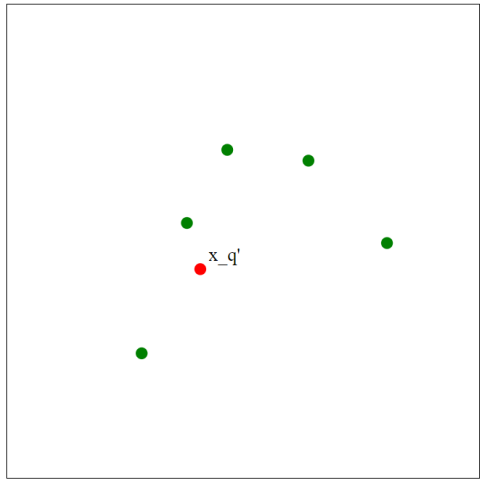
$x'_q \in R^{d'}$ Transformed representation of x_q which is based on the transformed representations of other tokens and the degree of association of x_q to those other tokens

Attention Parameters

- A "value" or transformation function $v(x): R^d \rightarrow R^{d'}$ that produces a vector for a given token (For simplicity, assume, $d = d'$)
- A "Masking" function $M(x_q, S_{x_k})$ which gives a subset of tokens from S_{x_k} to which a given query can be compared, e.g., text upto a certain point. For simplicity, assume, for all x_q, $M(x_q, S_{x_k}) = S_{x_k}$)
- A "kernel" function $\mathrm{k}(x_i, x_j)$ that gives us the association between two tokens. Used to determine the **attention scores** that tell us how associated are x_q and x_k relative to similarity of x_q to all tokens
- Different formulations for $\mathrm{k}(x_i, x_j)$, $M(x_q, S_{x_k})$ and $v(x)$ give you different flavours of attentions. Learnable parameters denoted by $\boldsymbol{\theta}$.







Legend:
All circles are Points in S_{x_k}
Filled circles are in $M(x_q, S_{x_k})$ and will be used in the layer
Note that points in $M(x_q, S_{x_k})$ will change depending upon x_q
Thickness of solid lines indicates attention scores $a_{qk} \in [0,1]$ which is a obtained by dividing $\mathrm{k}(x_q, x_k)$ by the sum of all kernel values involving x_q.

Note change in space

The new representation of the token (indicated by star) is based on the "pulls" (attention values a_{qk}) of different points on the query token or the weighted combination of all transformed points. This process can be applied for all tokens in the input one by one so if there are n tokens in the input, there would be n tokens in the output (with transformed representation). Note that the representation would change if input tokens change.

- **Input:**
  - A "query" token $x_q \in R^d$ representation of a component (patch or token)
  - A set of "key" tokens S_{x_k}
- Attention Parameters
  - A value function $v(x): R^d \rightarrow R^{d'}$ that produces a vector for a given token
  - A "Masking" function $M(x_q, S_{x_k})$ which gives a subset of tokens from S_{x_k} to which a given query can be compared (For simplicity, assume, for all x_q, $M(x_q, S_{x_k}) = S_{x_k}$)
  - A kernel function $\mathrm{k}(x_i, x_j)$ that can give us a degree of similarity between two tokens
  - Different formulations for $\mathrm{k}(x_i, x_j)$, $M(x_q, S_{x_k})$ and $v(x)$ give you different flavours of attentions but once chosen they remain the same for a given attention block

- Output:
  - A new representation for the query token (patch)

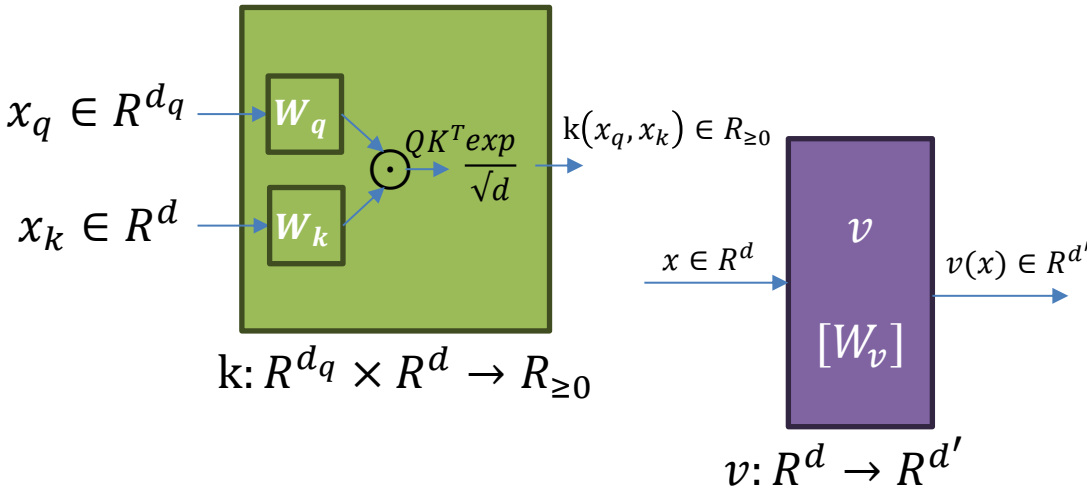
$$x'_q = A(x_q; M(x_q, S_{x_k}); \boldsymbol{\theta})$$
$$= \sum_{x_k \in M(x_q, S_{x_k})} \mathrm{a}(x_q, x_k; \boldsymbol{\theta_a})\, v(x_k; \boldsymbol{\theta_v})$$
$$= \sum_{x_k \in M(x_q, S_{x_k})} \frac{\mathrm{k}(x_q, x_k; \boldsymbol{\theta_k})}{\sum_{x_{k'} \in M(x_q, S_{x_k})} \mathrm{k}(x_q, x_{k'}; \boldsymbol{\theta_k})}\, v(x_k; \boldsymbol{\theta_v})$$

Interactive Demo: https://foxtrotmike.github.io/CS909/attention.html

Learnable Attention as (asymmetric, non-Mercer) kernel transformations

$$\boldsymbol{x}'_q = A\left(\boldsymbol{x}_q; M\left(\boldsymbol{x}_q, S_{\boldsymbol{x}_k}\right); \boldsymbol{\theta}\right) = \sum_{x_k \in M\left(x_q, S_{x_k}\right)} \mathrm{a}\left(\boldsymbol{x}_q, \boldsymbol{x}_k; \boldsymbol{\theta}_a\right) v\left(\boldsymbol{x}_k; \boldsymbol{\theta}_v\right) = \sum_{\boldsymbol{x}_k \in M\left(\boldsymbol{x}_q, S_{\boldsymbol{x}_k}\right)} \frac{\mathrm{k}\left(\boldsymbol{x}_q, \boldsymbol{x}_k; \boldsymbol{\theta}_k\right)}{\sum_{\boldsymbol{x}_{k'} \in M\left(x_q, S_{x_k}\right)} \mathrm{k}\left(\boldsymbol{x}_q, \boldsymbol{x}_{k'}; \boldsymbol{\theta}_k\right)} v\left(\boldsymbol{x}_k; \boldsymbol{\theta}_v\right)$$

- We can introduce learnable parameters
  - We can learn **which input tokens should associate more with other tokens** to produce a representation that when passed to the predictor should produce the target output
  - For example, **"Attention Is All You Need"** paper uses the following functions with three learnable weight matrices $\boldsymbol{W_q}$, $\boldsymbol{W_k}$ and $\boldsymbol{W_v}$



Dot Product

$$\mathrm{k}\left(\boldsymbol{x}_q, \boldsymbol{x}_k\right) = exp\left(\frac{1}{\sqrt{d}}\left\langle \boldsymbol{x}_q \boldsymbol{W_q}, \boldsymbol{x}_k \boldsymbol{W_k} \right\rangle\right)$$

$$v\left(\boldsymbol{x}_k\right) = \boldsymbol{x}_k \boldsymbol{W_v}$$

$$\boldsymbol{x}'_q = A\left(\boldsymbol{x}_q; M\left(\boldsymbol{x}_q, S_{\boldsymbol{x}_k}\right), \boldsymbol{\theta}\right)$$

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

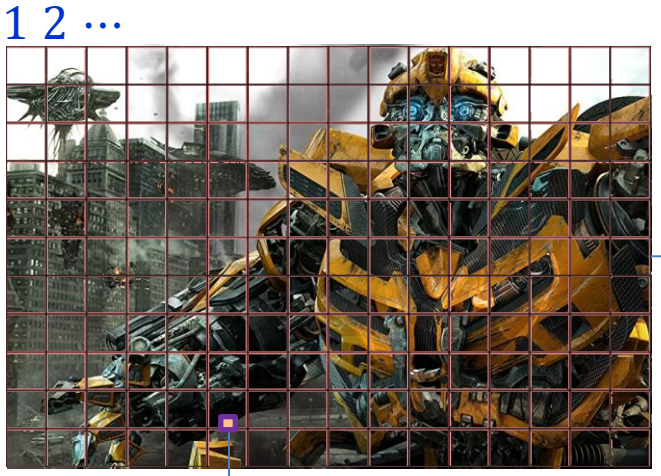
https://en.wikipedia.org/wiki/Softmax_function

$$\boldsymbol{x}'_q = A\left(\boldsymbol{x}_q; M\left(\boldsymbol{x}_q, S_{\boldsymbol{x}_k}\right)\right)$$

$$= \sum_{\boldsymbol{x}_k \in M\left(\boldsymbol{x}_q, S_{\boldsymbol{x}_k}\right)} \frac{exp\left(\frac{1}{\sqrt{d}}\left\langle \boldsymbol{x}_q \boldsymbol{W_q}, \boldsymbol{x}_k \boldsymbol{W_k} \right\rangle\right)}{\sum_{\boldsymbol{x}_{k'} \in M\left(\boldsymbol{x}_q, S_{\boldsymbol{x}_k}\right)} exp\left(\frac{1}{\sqrt{d}}\left\langle \boldsymbol{x}_q \boldsymbol{W_q}, \boldsymbol{x}_{k'} \boldsymbol{W_k} \right\rangle\right)} \boldsymbol{x}_k \boldsymbol{W_v}$$

$$= softmax\left(\frac{1}{\sqrt{d}} \boldsymbol{x}_q \boldsymbol{W_q} (\boldsymbol{x}_k \boldsymbol{W_k})^T\right) \boldsymbol{x}_k \boldsymbol{W_v} = softmax\left(\frac{1}{\sqrt{d}} q K^T\right) V$$

Output of a single attention layer

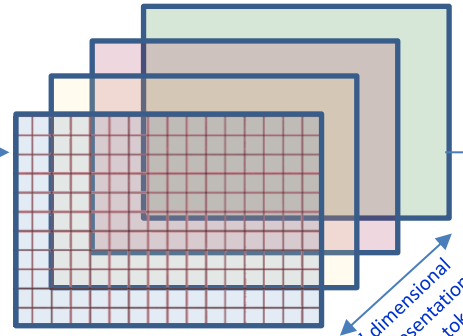


$\boldsymbol{x_i} \equiv \phi(\boldsymbol{f_i}, \boldsymbol{t_i})$ Feature Embedding: What is it?
Positional Embedding: Where is it?

Build Patch "Embedding" Representation

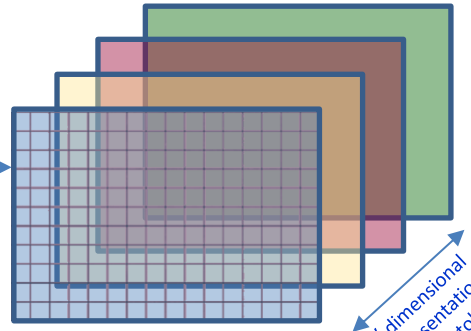
$S_x = \{\boldsymbol{x_i} \in R^d | i = 1 \ldots n\}$

n tokens

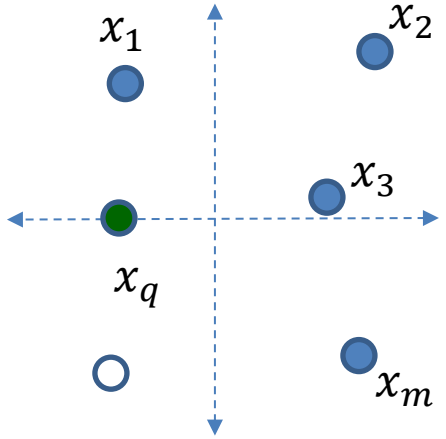


Representation of n tokens (small grid squares)

Attention Layer

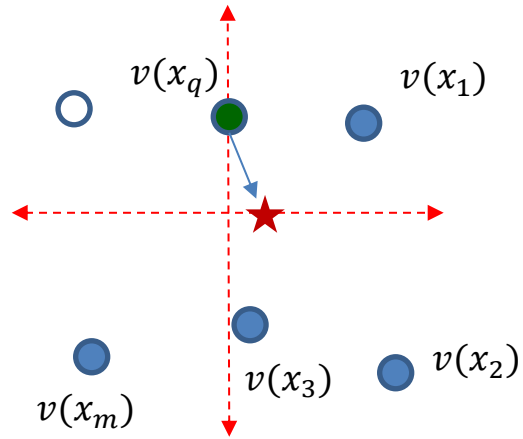


A total of n tokens (small grid squares) in output



"Learnable" Feature Transformation via attention

(shown for a single token as a scatter plot but this is applied to each point in the input)



Note that the representation of each patch (or token) at the output of attention is dependent upon the representation of all other patches in a end-to-end learnable manner so that when this representation is used for a prediction task, the loss is minimized

Attention gives transformations

- Another way of looking at an attention operation

Learnable "attention" values

$$A\left(x_q; M\left(x_q, S_{x_k}\right)\right) = \sum_{x_k \in M\left(x_q, S_{x_k}\right)} \frac{\mathrm{k}\left(x_q, x_k\right)}{\sum_{x_{k'} \in M\left(x_q, S_{x_k}\right)} \mathrm{k}\left(x_q, x_k\right)} v(x_k) = \sum_{x_k \in S_{x_k}} a(x_q, x_k; W)\, v(x_k; W')$$

Learnable data transformation

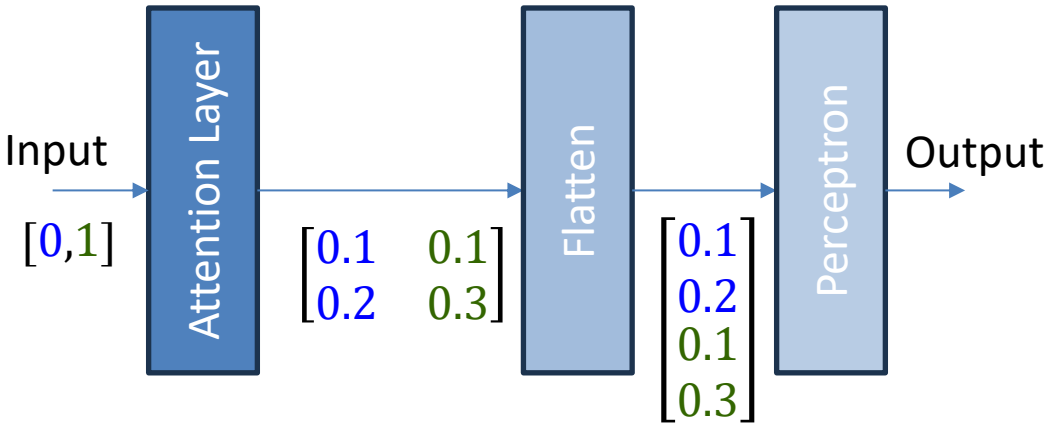
- But a classic neural network layer also "learns" to "transform"

$$F(x; \theta) = activation(\theta_{d' \times d} x_{d \times 1})$$

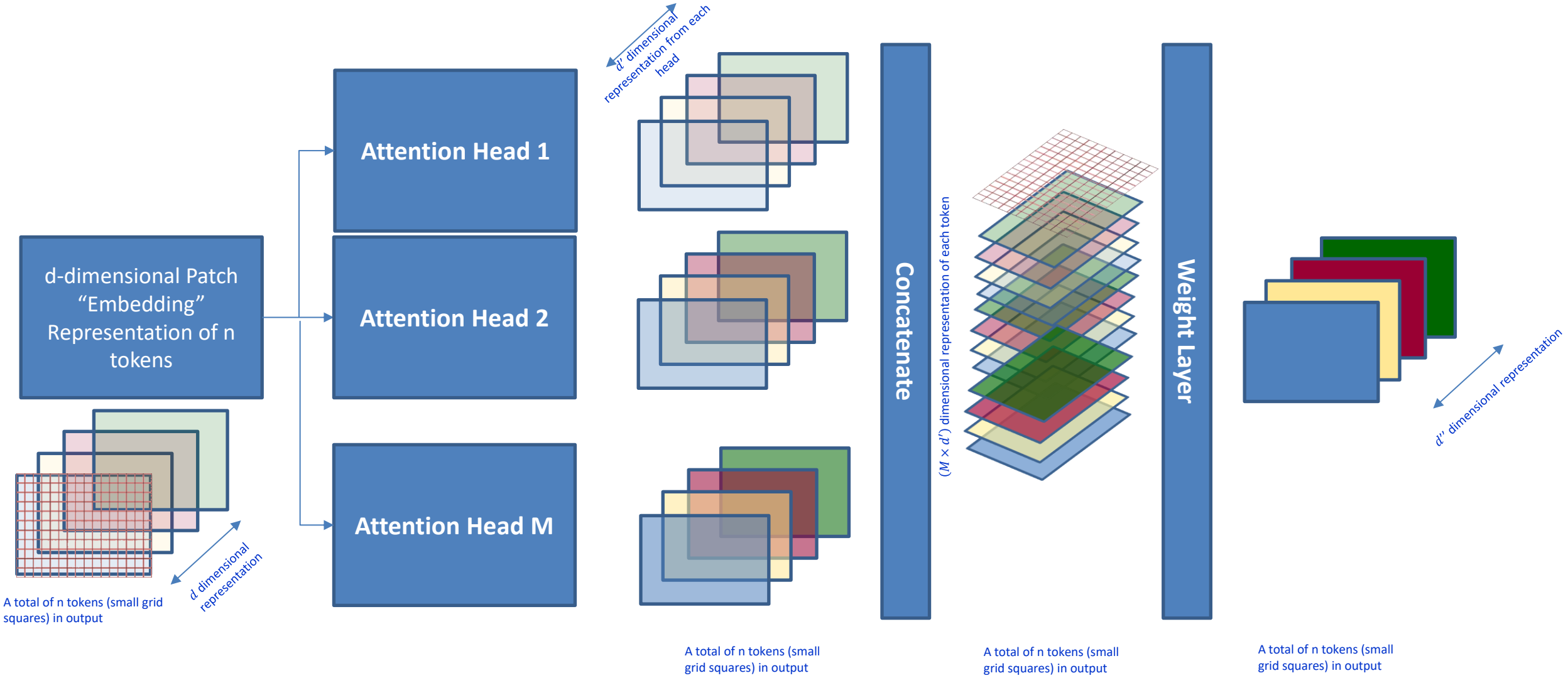
- Where is the extra information coming from?
  - From comparing against all tokens and using a supervisory signal to learn the transform
  - Weight sharing across all patches is still there like in a convolutional neural network

Lab Session

- Attention based solution of XOR
  - https://github.com/foxtrotmike/CS909/blob/master/xor_transformer.ipynb
  - Models each input bit as a token and then develops contextualized representations of each bit
  - Single Attention Layer
    - Represents each token as a 2D feature thus an input like [0,1] would be represented by a 4D vector
  - Passed to a perceptron model for producing the target label
  - Notice the different representation of each token which is context dependent
    - These attention weights demonstrate how the attention mechanism dynamically adjusts the importance of each input bit (token) based on the overall input context, enabling the model to capture relationships between bits that are essential for solving the XOR problem.



Multi-headed Attention



Cross Attention

- We had the query and keys coming from the same “modality”
- Multimodality data can be handled by attention as well
- Consider an image captioning tasks
  - Source: Image tokens
  - Target: text tokens
- Modelling through attention layer
  - $\boldsymbol{x}_q$: Comes from the target modality
  - $\boldsymbol{x}_k$: Come from the Source Modality
  - $\mathrm{a}\left(\boldsymbol{x}_q, \boldsymbol{x}_k; \boldsymbol{\theta}_a\right)$ determines the cross-attention
  - $v\left(\boldsymbol{x}_k; \boldsymbol{\theta}_v\right)$ maps the source features

$$\boldsymbol{x}'_q = A\left(\boldsymbol{x}_q; M\left(\boldsymbol{x}_q, S_{x_k}\right); \boldsymbol{\theta}\right) = \sum_{x_k \in M\left(x_q, S_{x_k}\right)} \mathrm{a}\left(\boldsymbol{x}_q, \boldsymbol{x}_k; \boldsymbol{\theta}_a\right) v\left(\boldsymbol{x}_k; \boldsymbol{\theta}_v\right)$$

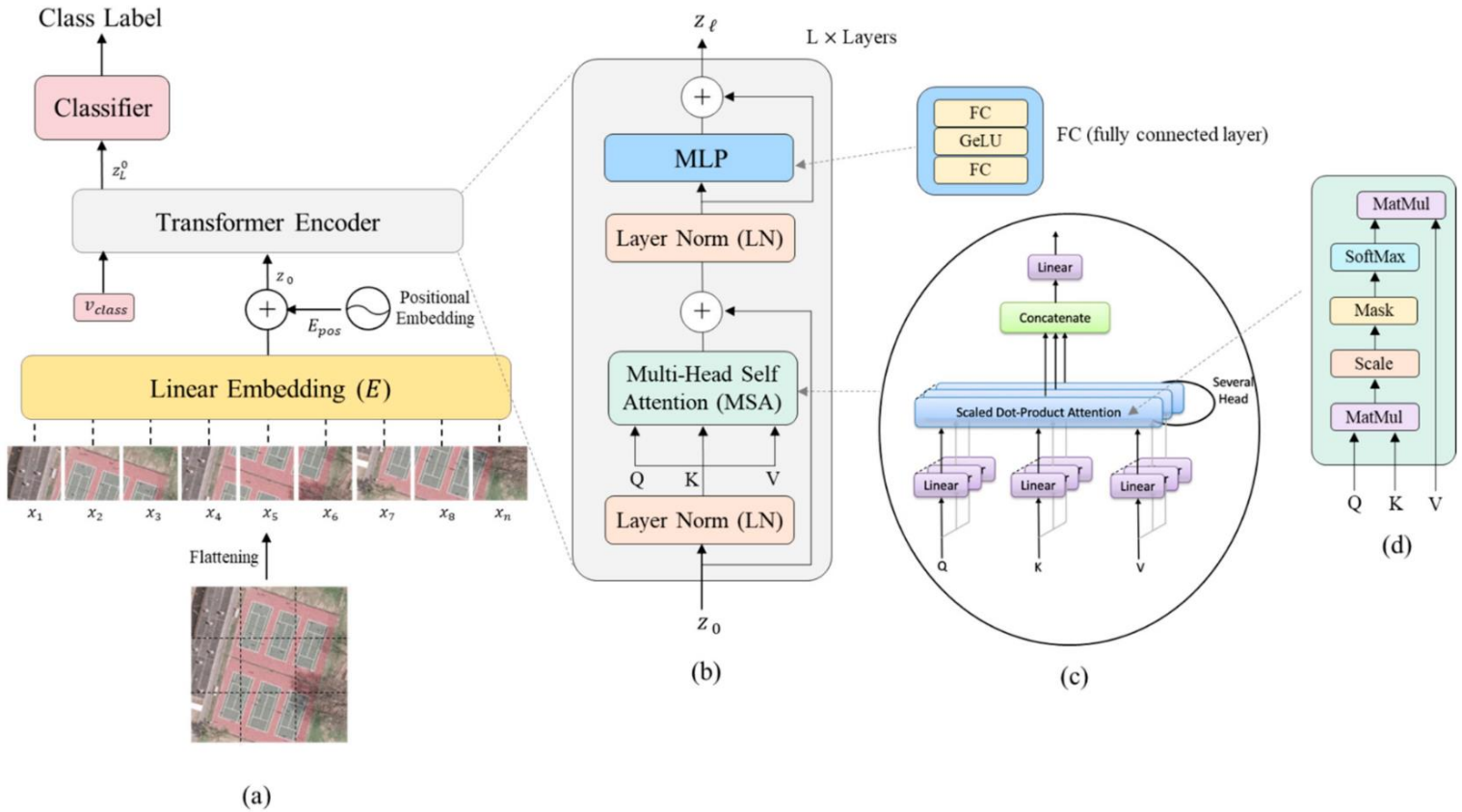
As he walked by the **bank**, he saw some boats

As he walked by the **bank**, he saw some tellers

Transformers

Key Components

- Multiple Multi-headed Attention Blocks
- Layer Normalization
  - standardization across features of the same input
- Skip Connections
- Various types of positional encodings
- "Class" tokens
  - Add global features to each example to enable global sharing of information across examples
- Masking strategies
  - Needed for training in sentence completion or related problems where the next work cannot be used for generating the output
- Computational Complexity
  - As we compare each token against every other, transformers can be quite complex
  - **Performer** architectures
    - Uses kernel approximation to reduce complexity



My PyTorch tutorial: https://github.com/foxtrotmike/CS909/blob/master/mnist_transformer.ipynb
Another Tutorial: https://medium.com/mlearning-ai/vision-transformers-from-scratch-pytorch-a-step-by-step-guide-96c3313c2e0c
Figure from : Bazi, Yakoub, Laila Bashmal, Mohamad M. Al Rahhal, Reham Al Dayil, and Naif Al Ajlan. "Vision Transformers for Remote Sensing Image Classification." *Remote Sensing* 13, no. 3 (January 2021): 516. https://doi.org/10.3390/rs13030516.
Krzysztof, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, et al. "Rethinking Attention with Performers." arXiv, November 19, 2022. https://doi.org/10.48550/arXiv.2009.14794
Dosovitskiy, Alexey, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, et al. "**An Image Is Worth 16x16 Words: Transformers for Image Recognition at Scale**." arXiv, June 3, 2021. https://doi.org/10.48550/arXiv.2010.11929.

What do transformers see?

Do Vision Transformers See Like Convolutional Neural Networks?

Maithra Raghu
Google Research, Brain Team
maithrar@gmail.com

Thomas Unterthiner
Google Research, Brain Team
unterthiner@google.com

Simon Kornblith
Google Research, Brain Team
kornblith@google.com

Chiyuan Zhang
Google Research, Brain Team
chiyuan@google.com

Alexey Dosovitskiy
Google Research, Brain Team
adosovitskiy@google.com

Abstract

Convolutional neural networks (CNNs) have so far been the de-facto model for visual data. Recent work has shown that (Vision) Transformer models (ViT) can achieve comparable or even superior performance on image classification tasks. This raises a central question: *how are Vision Transformers solving these tasks*? Are they acting like convolutional networks, or learning entirely different visual representations? Analyzing the internal representation structure of ViTs and CNNs on image classification benchmarks, we find striking differences between the two architectures, such as ViT having more uniform representations across all layers. We explore how these differences arise, finding crucial roles played by self-attention, which enables early aggregation of global information, and ViT residual connections, which strongly propagate features from lower to higher layers. We study the ramifications for spatial localization, demonstrating ViTs successfully preserve input spatial information, with noticeable effects from different classification methods. Finally, we study the effect of (pretraining) dataset scale on intermediate features and transfer learning, and conclude with a discussion on connections to new architectures such as the MLP-Mixer.

Understanding Robustness of Transformers for Image Classification

Srinadh Bhojanapalli*, Ayan Chakrabarti*, Daniel Glasner*, Daliang Li*, Thomas Unterthiner*, Andreas Veit*
Google Research
{bsrinadh, ayanchakrab, dglasner, daliangli, unterthiner, aveit}@google.com

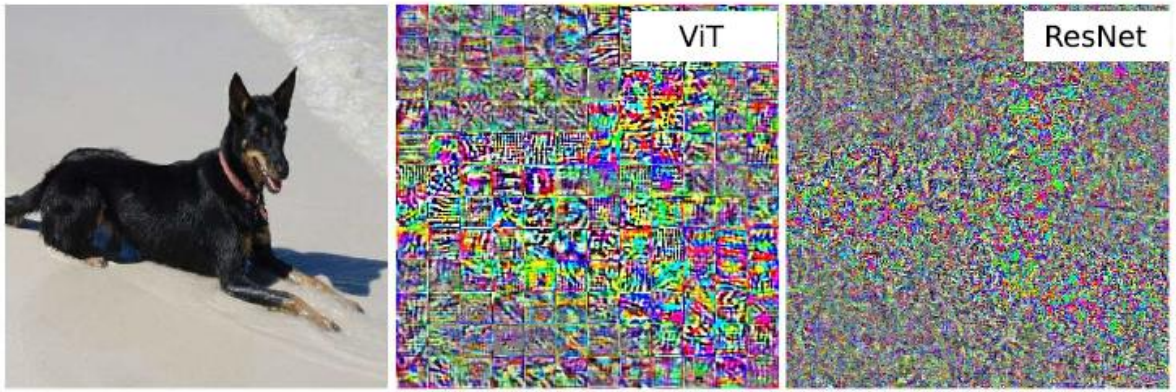


Figure 1. **Transformers vs. ResNets**. While they achieve similar performance for image classification, Transformer and ResNet architectures process their inputs very differently. Shown here are adversarial perturbations computed for a Transformer and a ResNet model, which are qualitatively quite different.

Are convolutions and attention really necessary?

- MLP Mixer Paper: “In this paper we show that while convolutions and attention are both sufficient for good performance, neither of them are necessary. ”
- gMLP: “self-attention is not critical for Vision Transformers”
- Attention with Convolution may be more useful ☺
- Attention Free Transformer

[Submitted on 12 Jun 2017 (v1), last revised 6 Dec 2017 (this version, v5)]

Attention Is All You Need

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks in an encoder-decoder configuration. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.

“Similar to fully-connected networks, the ViT architecture (and transformer architecture in general) lacks the inductive bias for spatial invariance/equivariance that convolutional networks have. Consequently, ViTs require more data for pretraining to acquire useful "priors" from the training data.” (S. Raschka)

https://twitter.com/rasbt/status/1363671712467177472

Tolstikhin, Ilya, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, et al. “MLP-Mixer: An All-MLP Architecture for Vision.” arXiv, June 11, 2021. https://doi.org/10.48550/arXiv.2105.01601.
Liu, Hanxiao, Zihang Dai, David R. So, and Quoc V. Le. “Pay Attention to MLPs.” arXiv, June 1, 2021. https://doi.org/10.48550/arXiv.2105.08050.
https://paperswithcode.com/method/attention-free-transformer

Lab Session

- Transformer based model for digit classification
  - https://github.com/foxtrotmike/CS909/blob/master/mnist_transformer.ipynb
- Hugging Face Transformers Library
  - Examples: https://huggingface.co/docs/transformers/model_doc/vit
  - Tutorial notebook on finetuning: https://github.com/NielsRogge/Transformers-Tutorials/blob/master/VisionTransformer/Fine_tuning_the_Vision_Transformer_on_CIFAR_10_with_the_%F0%9F%A4%97_Trainer.ipynb
- Flash Attention (Faster)

From Transformers to Large Language Models

How is an attention layer used in Chat-GPT?

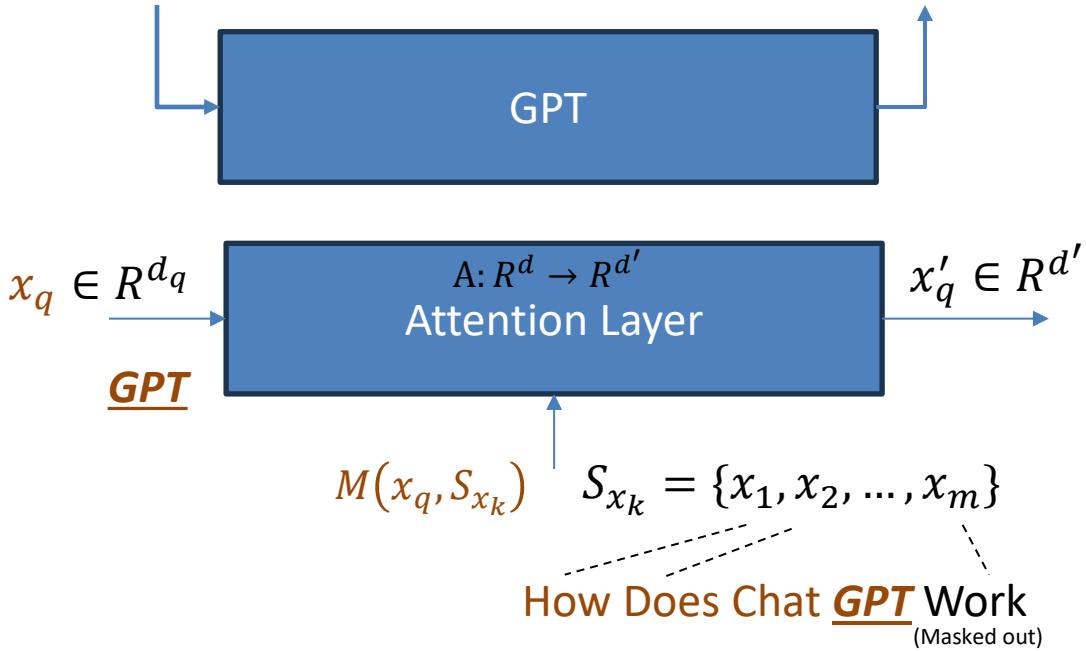
- GPTs are essentially sophisticated auto-complete mechanisms
  - Self-Supervised Learning: Predict next word
- **Training Principle**
- Taken each "document" as a set of tokens:

$S_{x_k} = \{x_1, x_2, \dots, x_m\}$

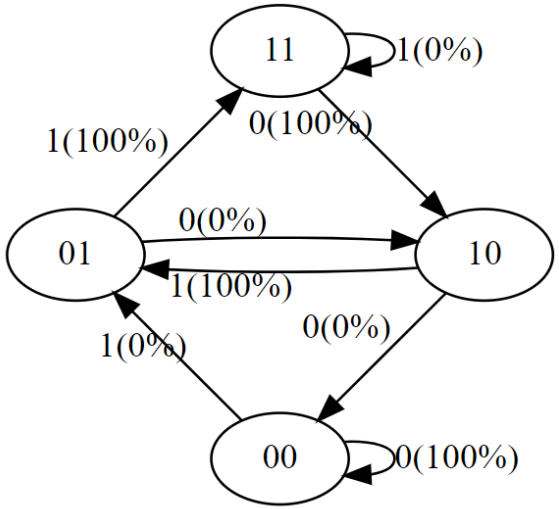
1. Take a single next-word prediction task from the document (see **bold** text on the right)
   a. For each token in the input, apply the attention layers to a single token
      i. Take a single "query" token x_q in the input for which we want to generate a representation
      ii. For the given "example" input, mask the next token, i.e., set $M(x_q, S_{x_k})$ to be a subset of only those tokens that are available as inputs
      iii. Pass x_q and $M(x_q, S_{x_k})$ to the attention layer to generate x'_q
2. Pass the updated representation through other downstream layers until you generate the output probability of the target token
3. Maximize the probability of the target token while minimizing the probability of all other (non-target) tokens

https://github.com/karpathy/nanoGPT

Input (Context length 4)	Next Token
How [Empty] [Empty] [Empty]	Does
How does [Empty] [Empty]	Chat
How does Chat [Empty]	GPT
How does Chat GPT	**Work**
… all sentences in the internet corpus …	



Modelling XOR as a “next token” problem



GPT is becoming a Turing machine: Here are some ways to program it

Ana Jojic, Zhen Wang, Nebojsa Jojic

We demonstrate that, through appropriate prompting, GPT-3 family of models can be triggered to perform iterative behaviours necessary to execute (rather than just write or recall) programs that involve loops, including several popular algorithms found in computer science curricula or software developer interviews. We trigger execution and description of Iterations by Regimenting Self-Attention (IRSA) in one (or a combination) of three ways: 1) Using strong repetitive structure in an example of an execution path of a target program for one particular input, 2) Prompting with fragments of execution paths, and 3) Explicitly forbidding (skipping) self-attention to parts of the generated text. On a dynamic program execution, IRSA leads to larger accuracy gains than replacing the model with the much more powerful GPT-4. IRSA has promising applications in education, as the prompts and responses resemble student assignments in data structures and algorithms classes. Our findings hold implications for evaluating LLMs, which typically target the in-context learning: We show that prompts that may not even cover one full task example can trigger algorithmic behaviour, allowing solving problems previously thought of as hard for LLMs, such as logical puzzles. Consequently, prompt design plays an even more critical role in LLM performance than previously recognized.

Input	Next “Target” Token	Target Probability	
		P(0)	P(1)
0,0	0	1	0
0,1	1	0	1
1,0	1	0	1
1,1	0	1	0

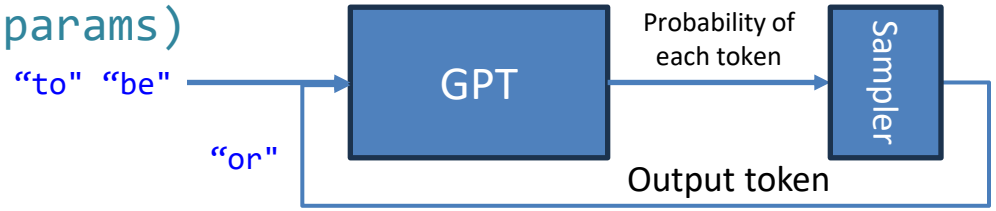
https://github.com/foxtrotmike/CS909/blob/master/xor_gpt_finite_state.ipynb
https://github.com/foxtrotmike/CS909/blob/master/gpt_finite_state.ipynb


GOOD FREND FOR IESVS SAKE FORBEARE,
TO DIGG THE DVST ENCLOASED HEARE.
BLESTE BE Y MAN Y SPARES THES STONES,
AND CVRST BE HE Y MOVES MY BONES.
THE GRAVE OF THE POET
WILLIAM
SHAKESPEARE

GPT: To Be or Not to Be…

```
def train(texts: list[list[str]], params) -> float:
    for text in texts:
        inputs = tokenizer.encode(text)
        x, y = inputs[:-1], inputs[1:]
        output = gpt(x, params)
        loss = np.mean(-np.log(output[y]))
        gradients = compute_gradients (loss, params)
        params = gradient_descent_update (gradients, params)
    return params
```

```
Tokenization converts a text to tokens to
represent them as input to the model

vocab = [“be", “not", “or", “to", ".“, …]
inputs = [3, 0] # “to" “be" #Tokenization
# In reality, it is more complex (see tiktoken)
```

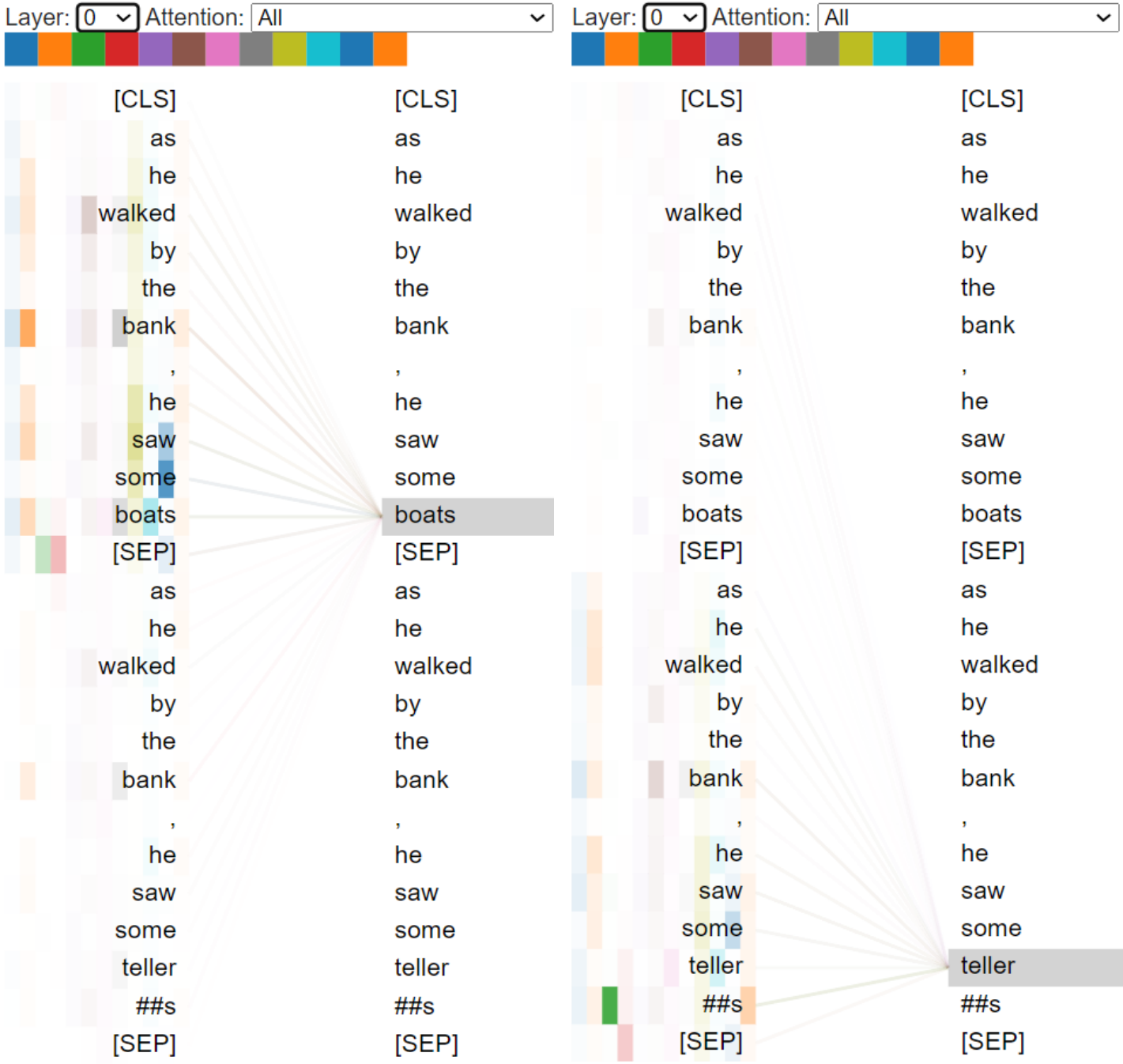



```
def generate(inputs, n_tokens_to_generate):
    for _ in range(n_tokens_to_generate): # auto-regressive decode loop
        output = gpt(inputs) # model forward pass
        next_id = np.argmax(output[-1]) # greedy sampling
        inputs.append(int(next_id)) # append prediction to input
        return inputs[len(inputs) - n_tokens_to_generate :] # return generated ids
```

https://jaykmody.com/blog/gpt-from-scratch/
https://github.com/jaymody/picoGPT
https://github.com/eniompw/nanoGPTshakespeare

Training

- What happens at the end of the Training phase?
  - [For NLP] The representation or embedding of different tokens only in reference to representations of other tokens



https://github.com/jessevig/bertviz

Prompting & In-Context Learning

- The original model was meant to be used as an embedding model
- However, it was found that one can prompt it as well
  - Example
    - What did Shakespeare say?
    - Answer: “To be or not to be”

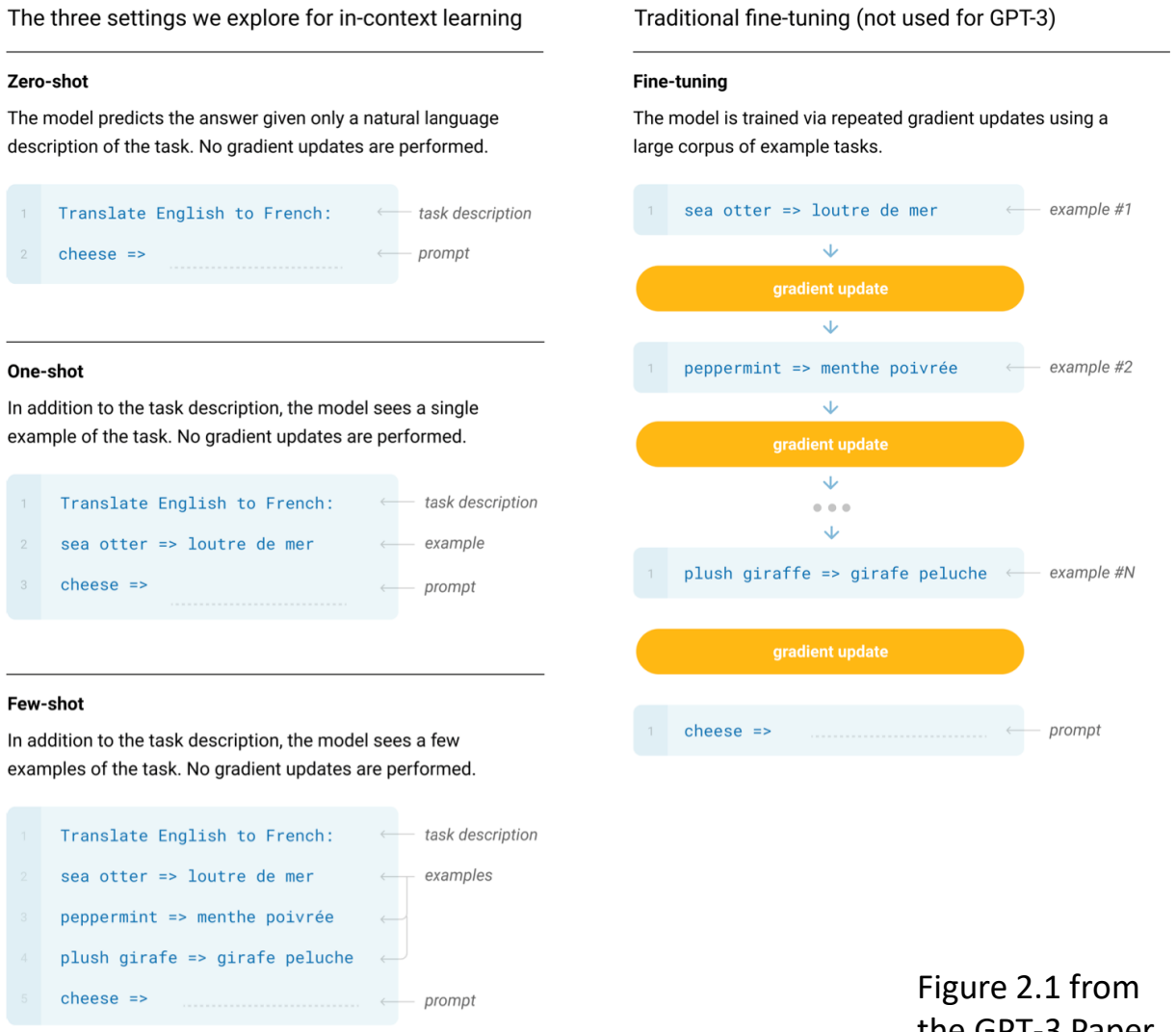


Figure 2.1 from the GPT-3 Paper

Fine tuning with Reinforcement Learning from Human Feedback (RLHF)

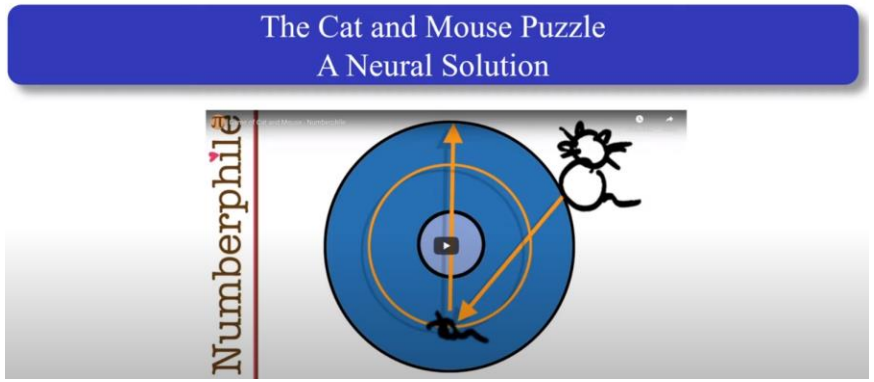
- **Prompt**: "Tell me, what is love?"
- **Generated Responses**
  1. **Response 1**: "Love is a gentle breeze that doth whisper sweet nothings in the ears of lovers."
  2. **Response 2**: "Love is but a fleeting shadow, a figment of the mind, a sweet torment that binds the heart and soul."
  3. **Response 3**: "I know not what love is. Perhaps thou should ask another."

- **Human Evaluation**
  - Human evaluators rank these responses
    - **Response 2**
    - **Response 1**
    - **Response 3**

- **Reward Model Training**
  - The reward model is trained using these rankings to predict human preferences for responses
- **Fine-Tuning with Reinforcement Learning**
  - The original language model is fine-tuned using the reward model's feedback. This helps the model learn to generate responses that are more likely to be preferred by humans

Understanding Reinforcement Learning

- Reinforcement Learning
  - Learning from experience
  - Example: Learning to levitate or helping a mouse escape from a cat
  - https://github.com/foxtrotmike/RL-MagLev/blob/master/RL.ipynb
  - https://github.com/foxtrotmike/RL-MagLev/blob/master/cat_mouse.ipynb



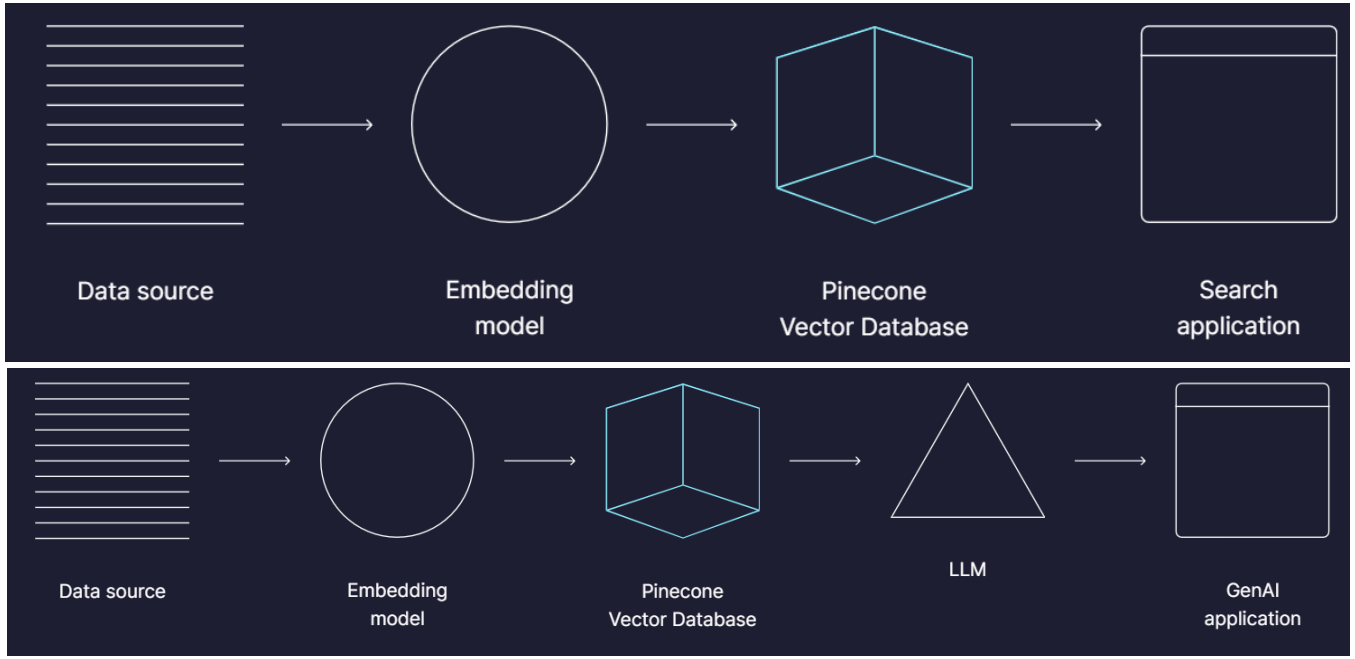
My RL Tutorial Video: https://youtu.be/N20h6vpR13Y

Retrieval Augmented Generation

- Prompt: "What are the key contributions of Alan Turing to computer science?"
- **Retrieval Component:**
  - The retrieval model searches a large database of documents and retrieves passages related to Alan Turing's work, such as his paper on computable numbers, his role in breaking the Enigma code during World War II, and his contributions to the development of early computers.
- **Generative Component:**
  - The generative model takes the input query and the retrieved passages as context and generates a response that synthesizes this information.
  - Generated Response:
    - "Alan Turing made several key contributions to computer science. He introduced the concept of a Turing machine, which is a fundamental model of computation. Turing also played a crucial role in breaking the Enigma code during World War II, which significantly contributed to the Allied victory. Additionally, he worked on the development of early computers and proposed the idea of artificial intelligence, famously known as the Turing Test."

The Rise of Vector Databases

- Flowise, langchain



https://www.pinecone.io/

https://github.com/pinecone-io/examples/blob/master/docs/gpt-4-langchain-docs.ipynb

Emergent Properties of LLMs

- capabilities or behaviors that arise from the scale and complexity of these models, which were not explicitly programmed into them but instead emerged as a result of their training on vast amounts of data
  - Zero-shot and Few-shot Learning
  - Language understanding, generation, summarization
  - Code Generation
  - Translational and multilingual abilities
  - Reasoning (with Chain/Tree/Graph of Thought)
  - Agent Use
  - Sentience(?)
  - Hallucinations and overcoming hallucinations

Lab Exercise

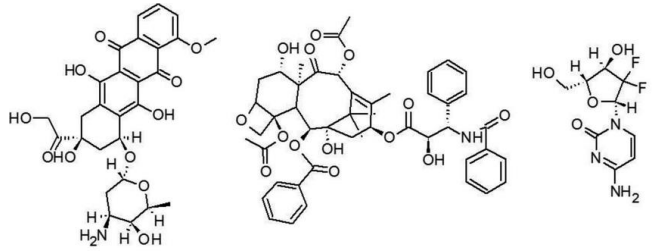
- World's simplest GPT: https://github.com/foxtrotmike/CS909/blob/master/xor_gpt_finite_state.ipynb
- PicoGPT: https://jaykmody.com/blog/gpt-from-scratch/
- Shakespeare Training of Nano-GPT: https://github.com/eniompw/nanoGPTshakespeare
- Using Embeddings
- Application: Getting structured pathology reports

GPT4All: Copy from Ethar

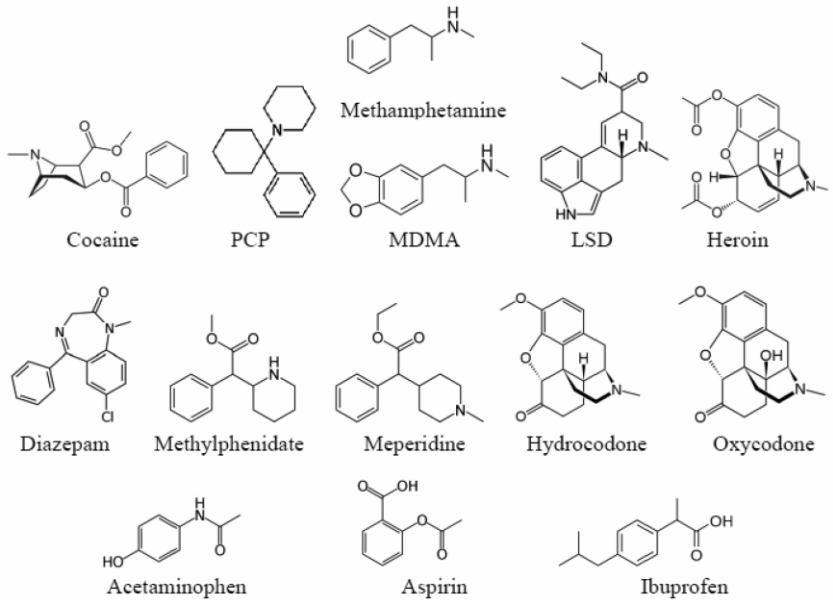
GRAPH NEURAL NETWORKS

Graph Neural Networks

- The Need
  - Example
    - Classifying chemical compounds
  - It is difficult to model arbitrary input data structures with SVMs, MLPs, CNNs and Transformers
    - Images and text have “Linear Structure”
      - Text is 1-dimensional
      - Image is 2-dimensional
      - But each can be mapped onto a grid



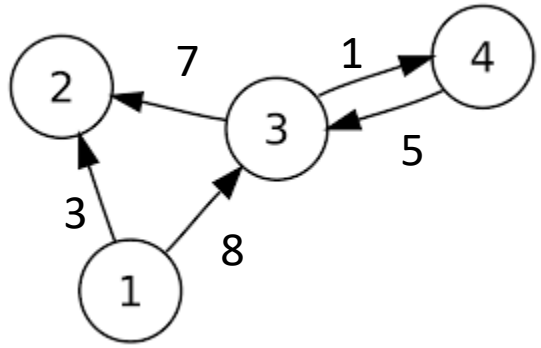
Cancer Drugs



Other Drugs

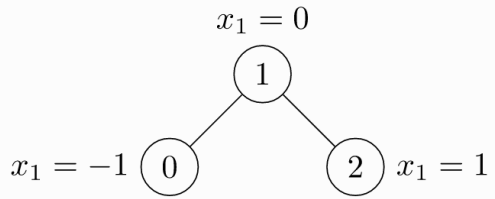
Graphs

- Graph Modelling
  - Very flexible data structure



- Components of a graph
  - Vertices/Node Set: $V = \{x_1, x_2, x_3, x_4\}$
    - Each element of the set can have a vector descriptor of its properties
  - Edge Set: $E = \{e_{1,2}, e_{1,3}, e_{3,2}, e_{3,4}, e_{4,3}\} \subseteq V \times V$
    - Each element of the set can have a vector descriptor of its properties

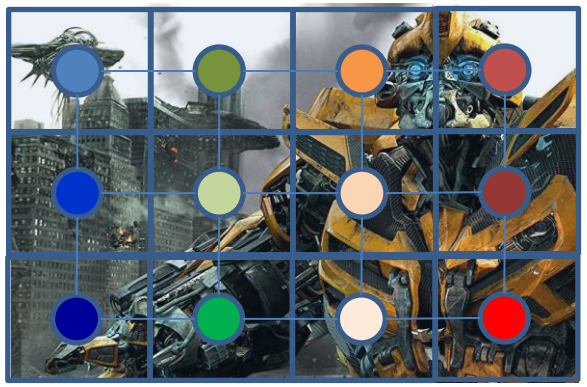
```
import torch
from torch_geometric.data import Data

edge_index = torch.tensor([[0, 1, 1, 2],
                           [1, 0, 2, 1]], dtype=torch.long)
x = torch.tensor([[-1], [0], [1]], dtype=torch.float)

data = Data(x=x, edge_index=edge_index)
>>> Data(edge_index=[2, 4], x=[3, 1])
```

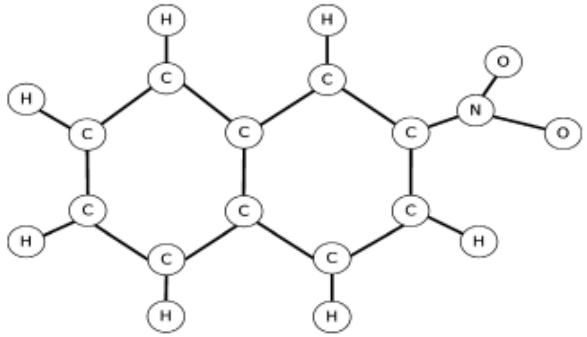


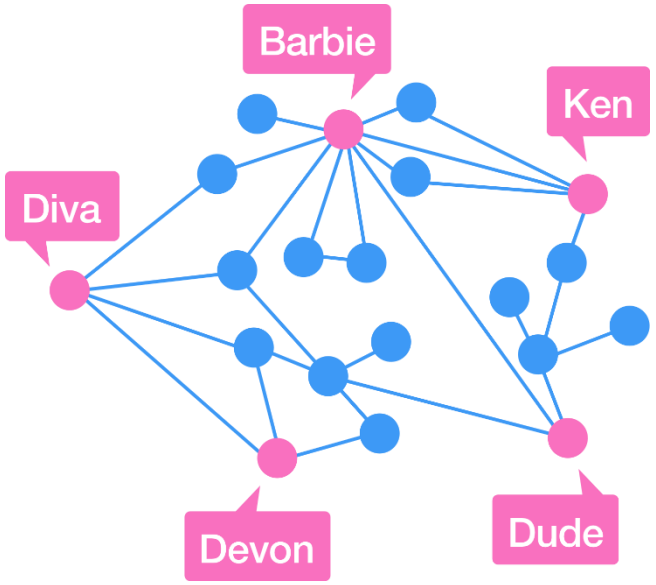
https://en.wikipedia.org/wiki/Graph_(discrete_mathematics)
https://en.wikipedia.org/wiki/Adjacency_matrix

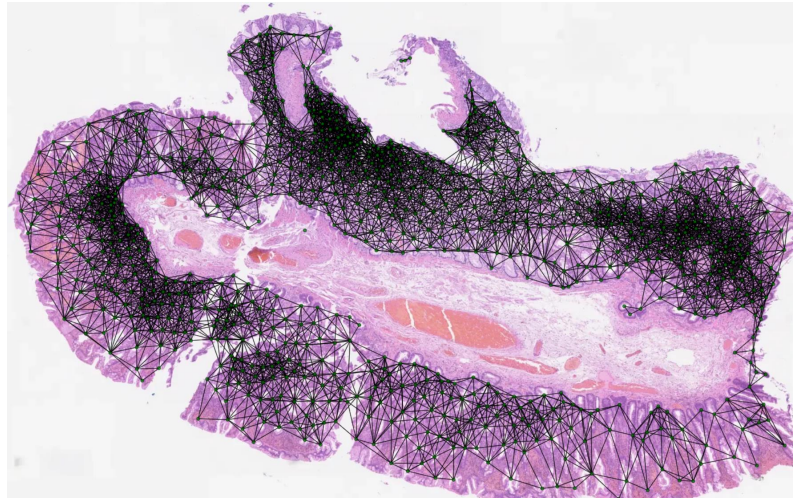
Examples of graphs

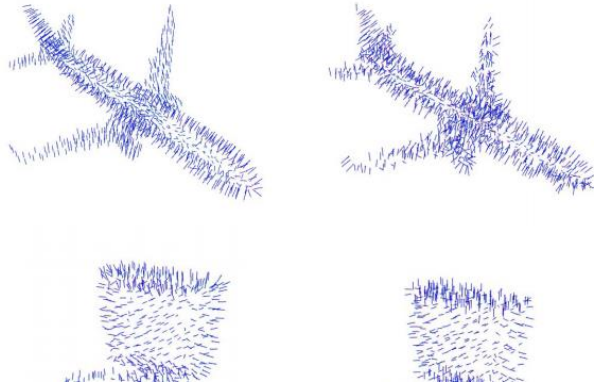
This is a graph





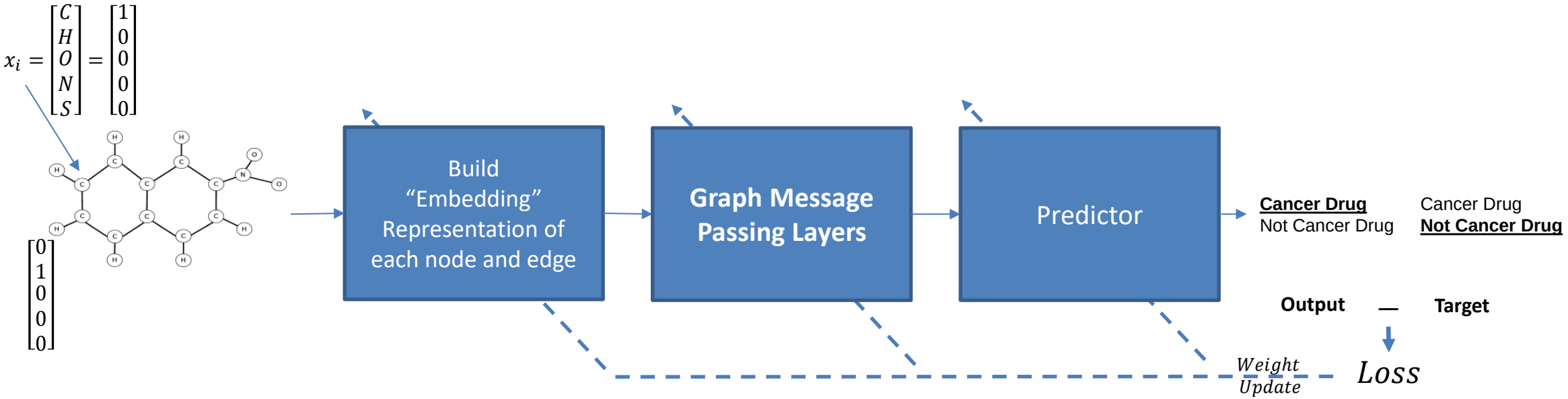






Graph Neural Networks

- Simple Graph Classification Example
  - Node and edge level prediction problems also possible



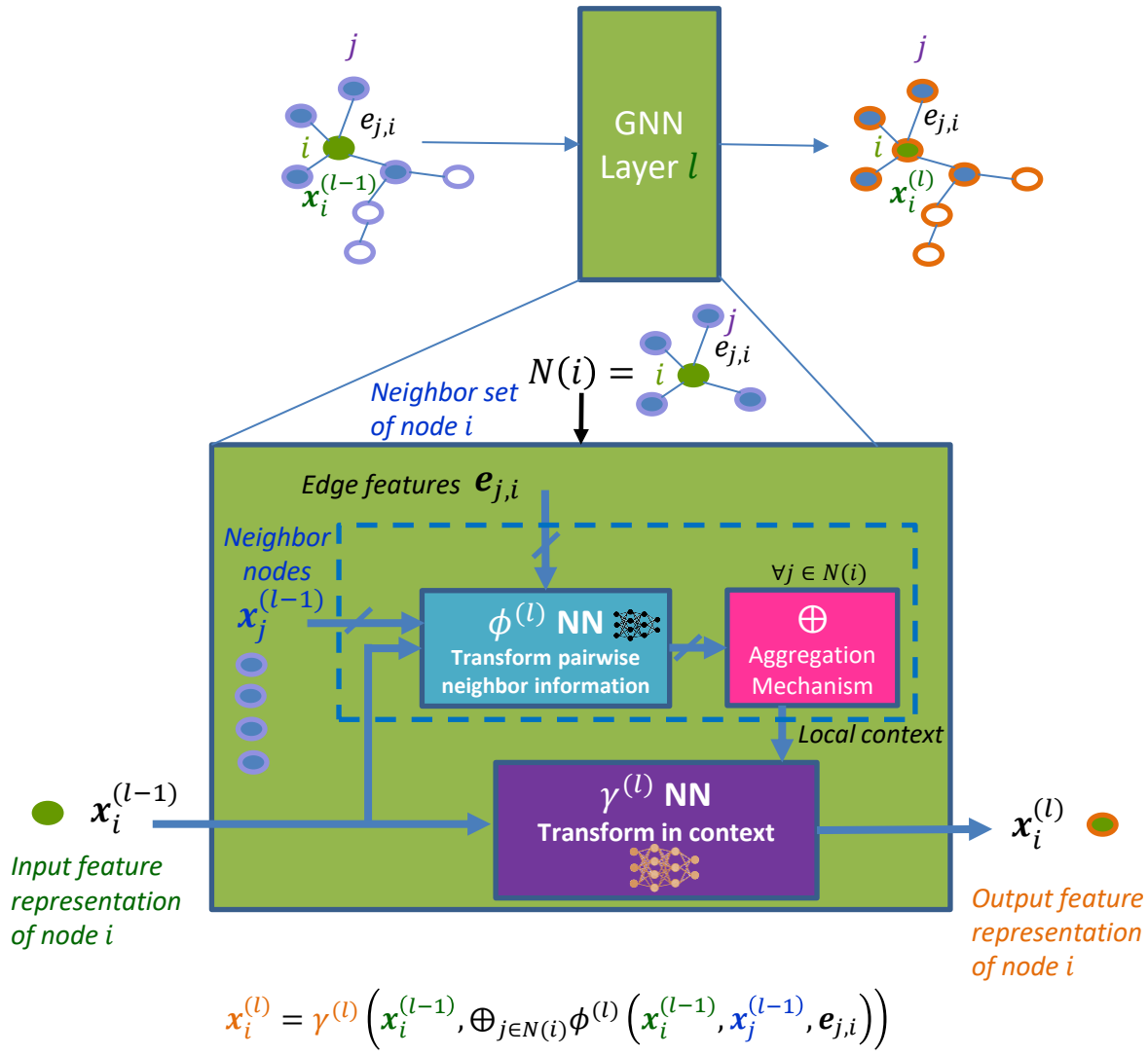
Input: Graph consisting of

Node set: what are things (each node has feature representation)

Edge set: how are they connected (each edge can have a feature representation but, in the very least, it tells us what nodes are connected by an edge)

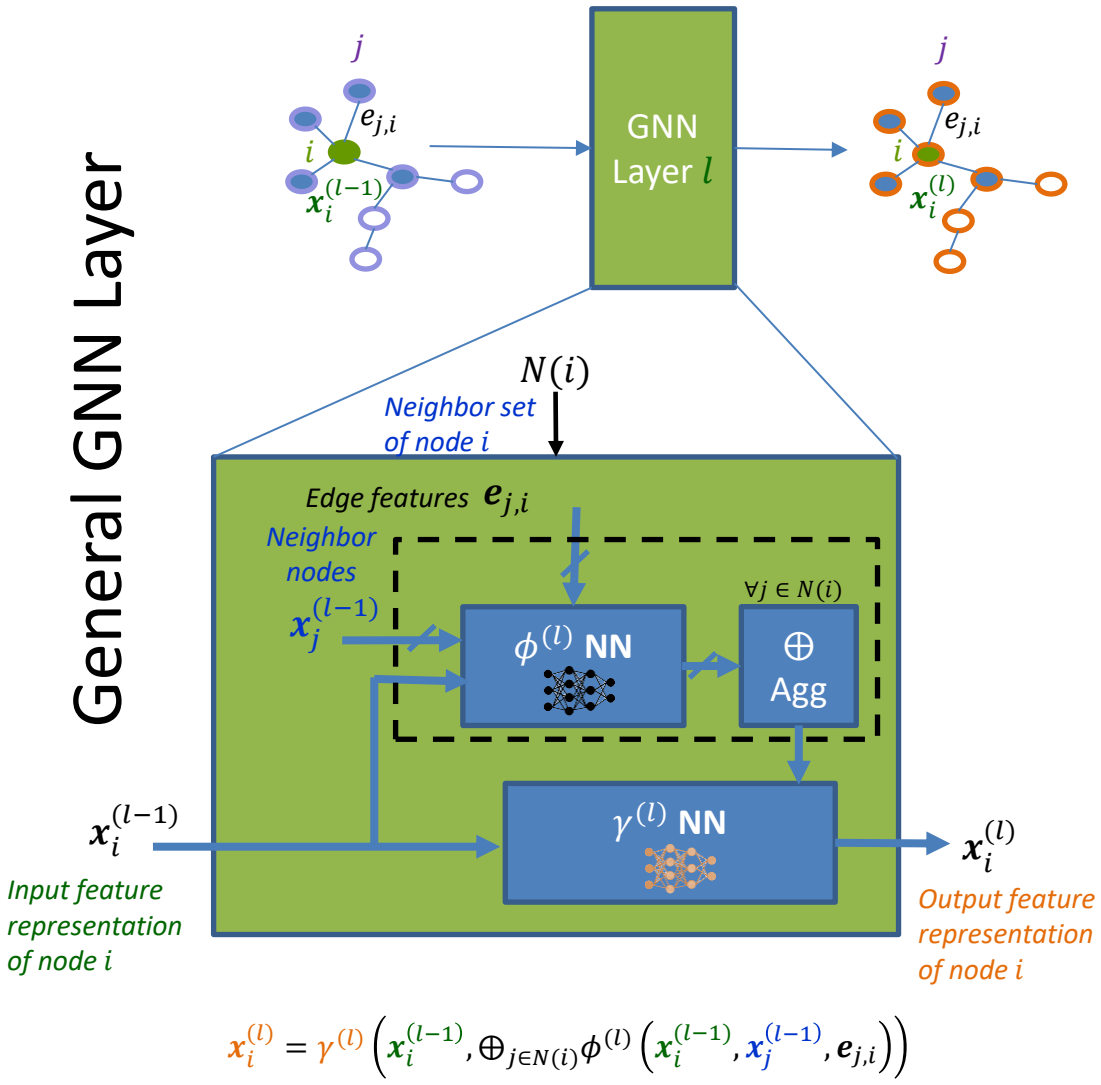
How does a graph neural network layer work?

- Just like any other neural network layer, the goal of a graph layer is to transform the representation of the input to a new representation in a learnable/trainable fashion so that we can optimize the parameters in the layer to reduce our loss or error function
- **Input:** A Graph with node and edge level features
- **Output:** A Graph with (transformed) node and edge features
- The GNN layer transforms the feature representation of each node as follows:
  - **Where am I?** Generate context for each input node
    - **Node pair transform:** Transform features of each node connected to an input node while taking pairwise edge information into account (using a neural network)
    - **Aggregation:** Aggregate information of neighbors of the node to provide the local context in the form of a fixed dimensional feature vector (max, sum, average, etc.)
  - **What should I become?** Transform each node in the context of its neighbors (using a neural network)
- Each GNN layer thus incorporates information from one hop away of each node thus multiple GNN layers in series can be used to incorporate information from multiple layers

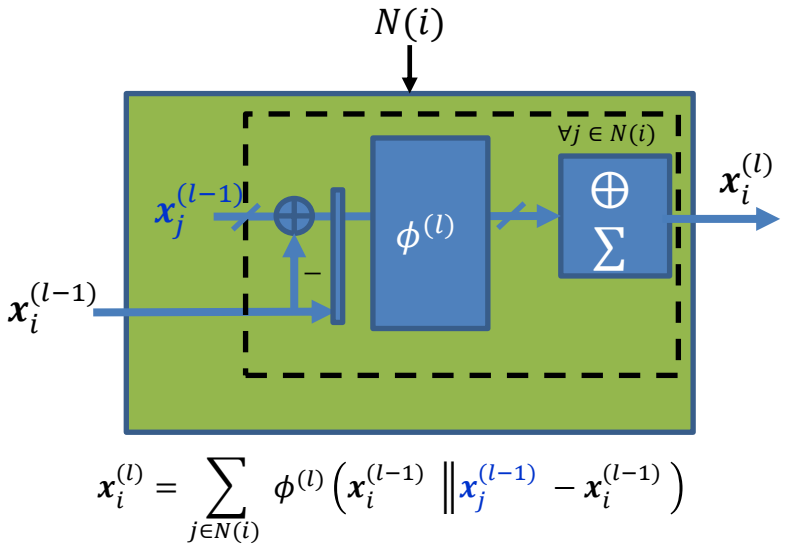


https://pytorch-geometric.readthedocs.io/en/latest/tutorial/create_gnn.html

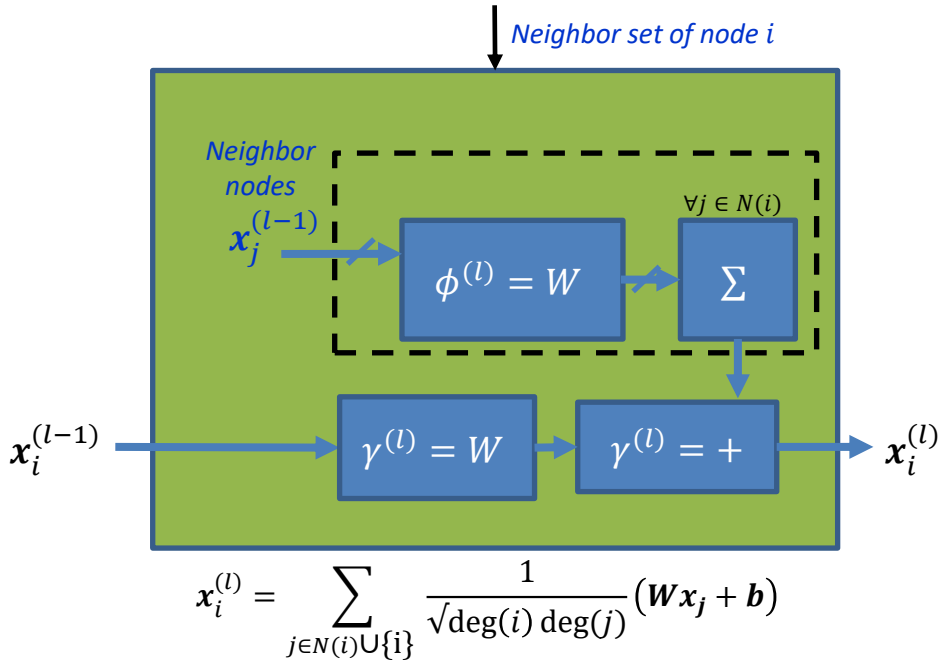
Implementing Different Graph Neural Network Layers



$$x_i^{(l)} = \gamma^{(l)}\left(x_i^{(l-1)}, \oplus_{j \in N(i)} \phi^{(l)}\left(x_i^{(l-1)}, x_j^{(l-1)}, e_{j,i}\right)\right)$$



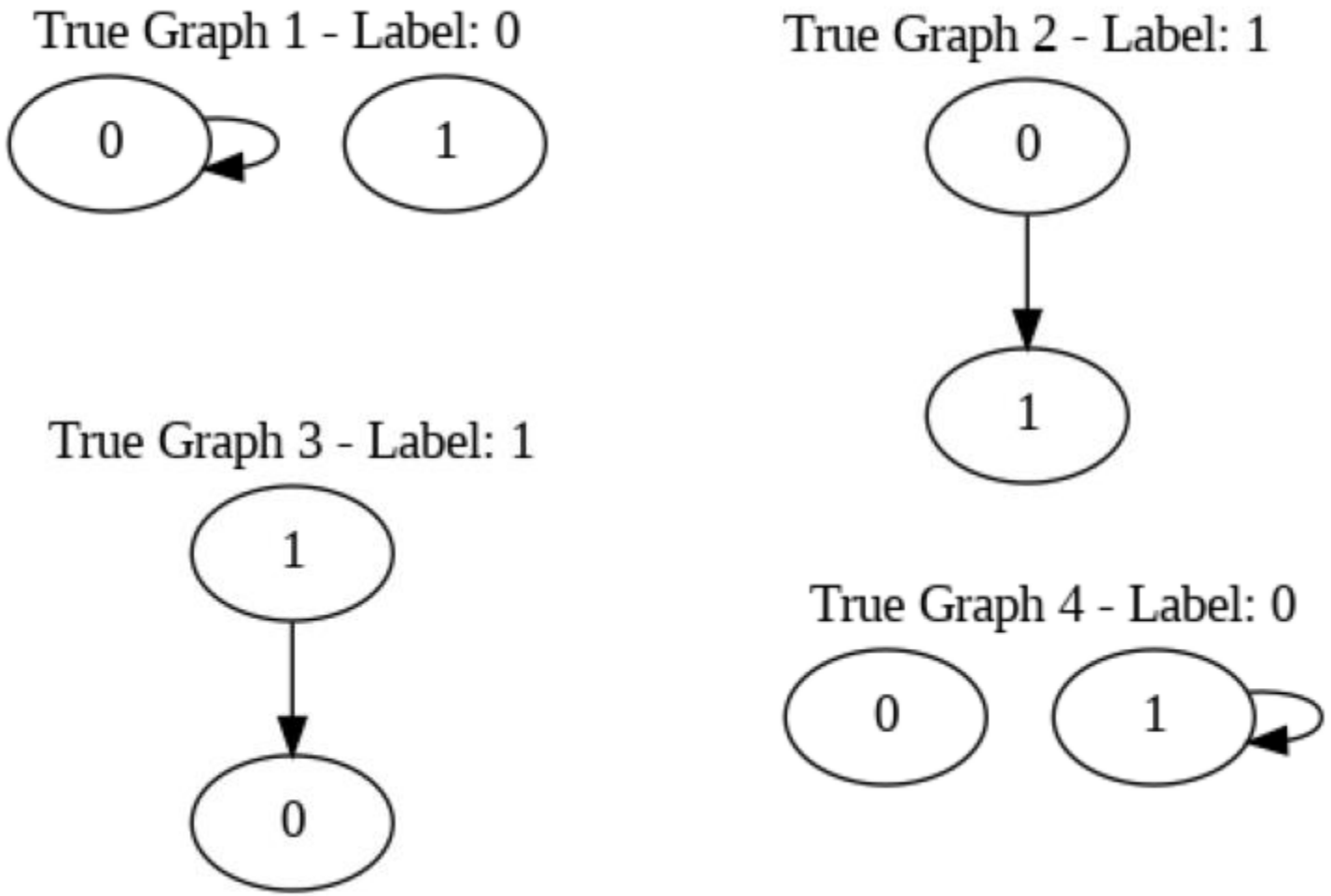
$$x_i^{(l)} = \sum_{j \in N(i)} \phi^{(l)}\left(x_i^{(l-1)} \left\| x_j^{(l-1)} - x_i^{(l-1)}\right.\right)$$



$$x_i^{(l)} = \sum_{j \in N(i) \cup \{i\}} \frac{1}{\sqrt{\deg(i)\deg(j)}}\left(W x_j + b\right)$$

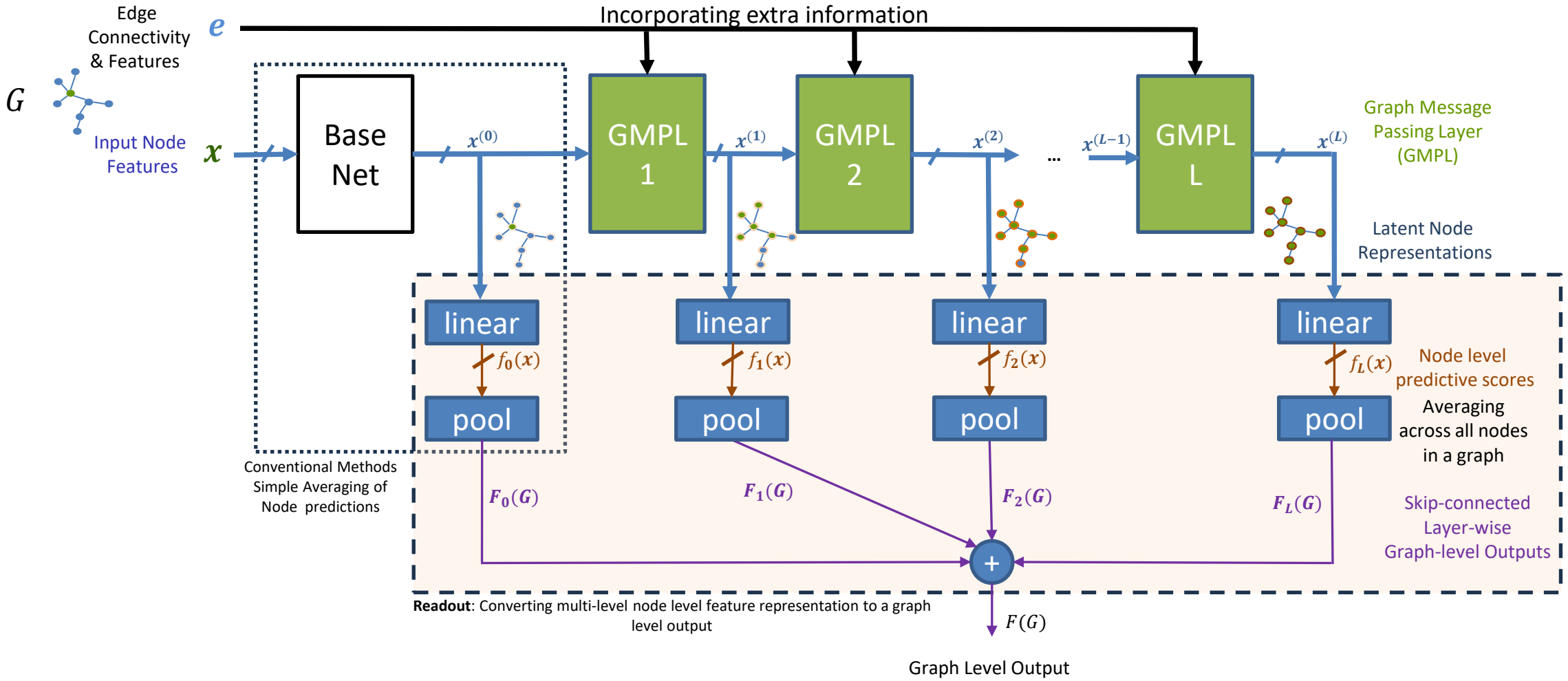
https://pytorch-geometric.readthedocs.io/en/latest/tutorial/create_gnn.html

XOR as a Graph Classification Problem



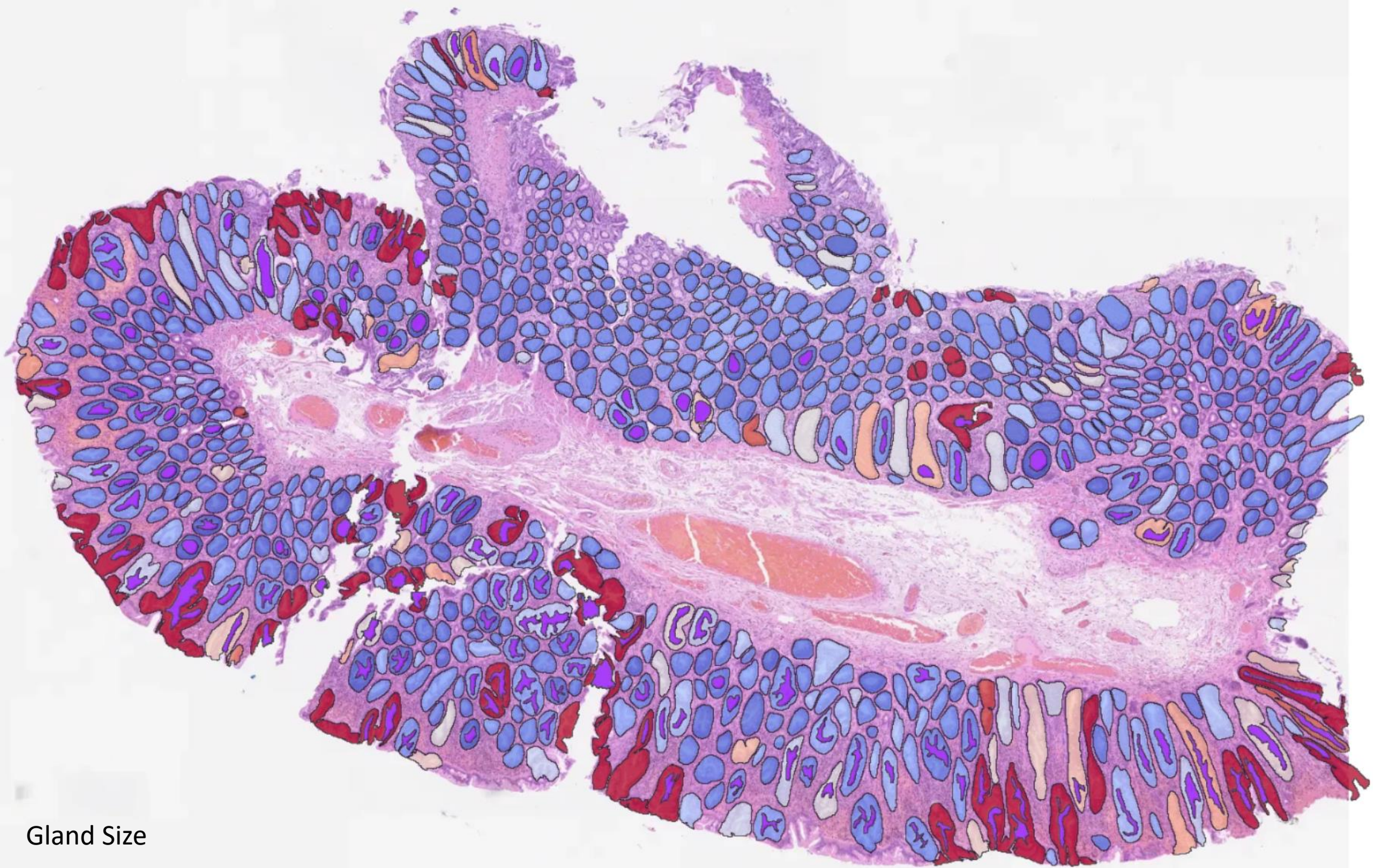
https://github.com/foxtrotmike/CS909/blob/master/xor_gnn.ipynb

Message Passing Based Graph Neural Networks



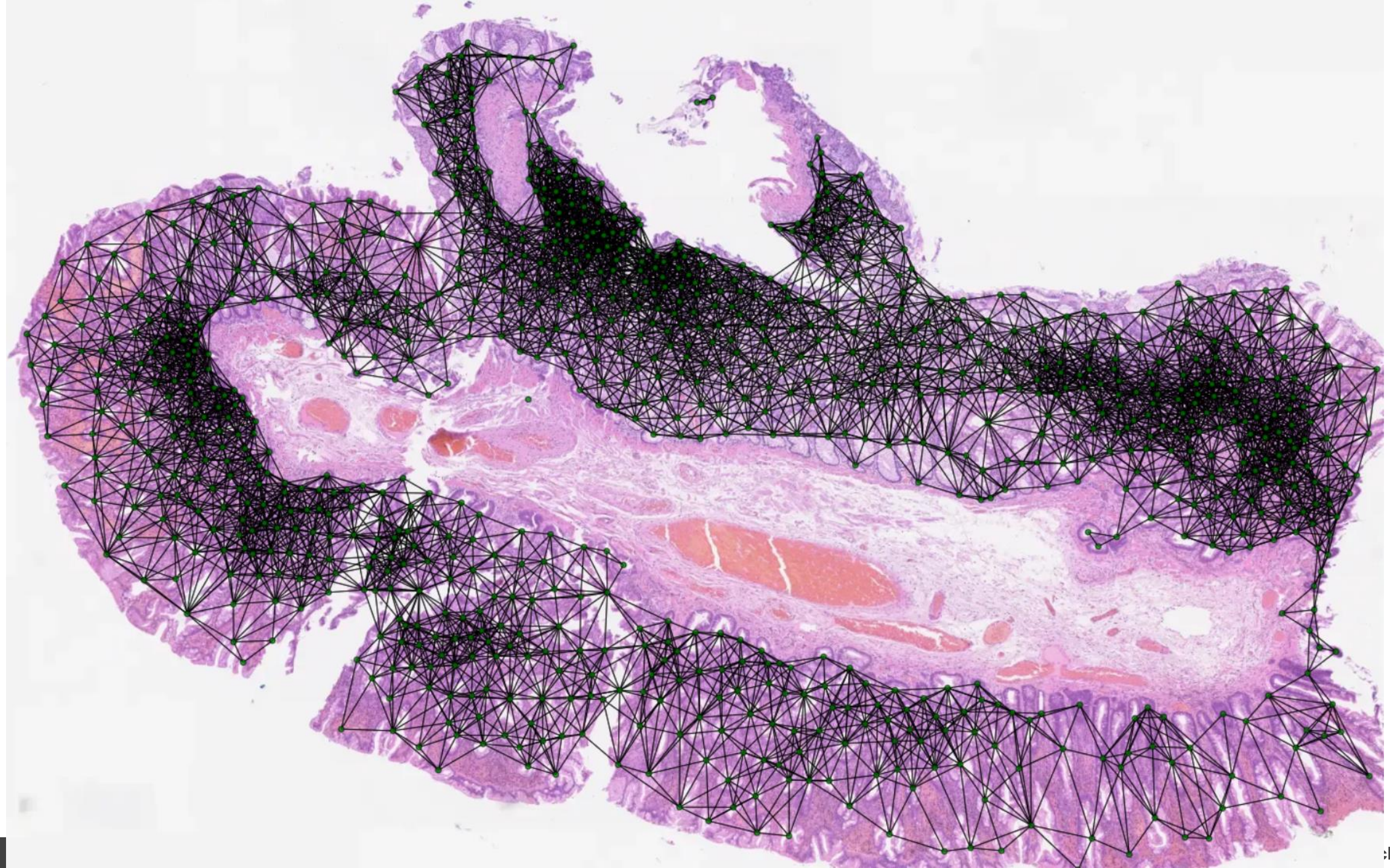
Code: https://tia-toolbox.readthedocs.io/en/latest/_notebooks/jnb/inference-pipelines/slide-graph.html

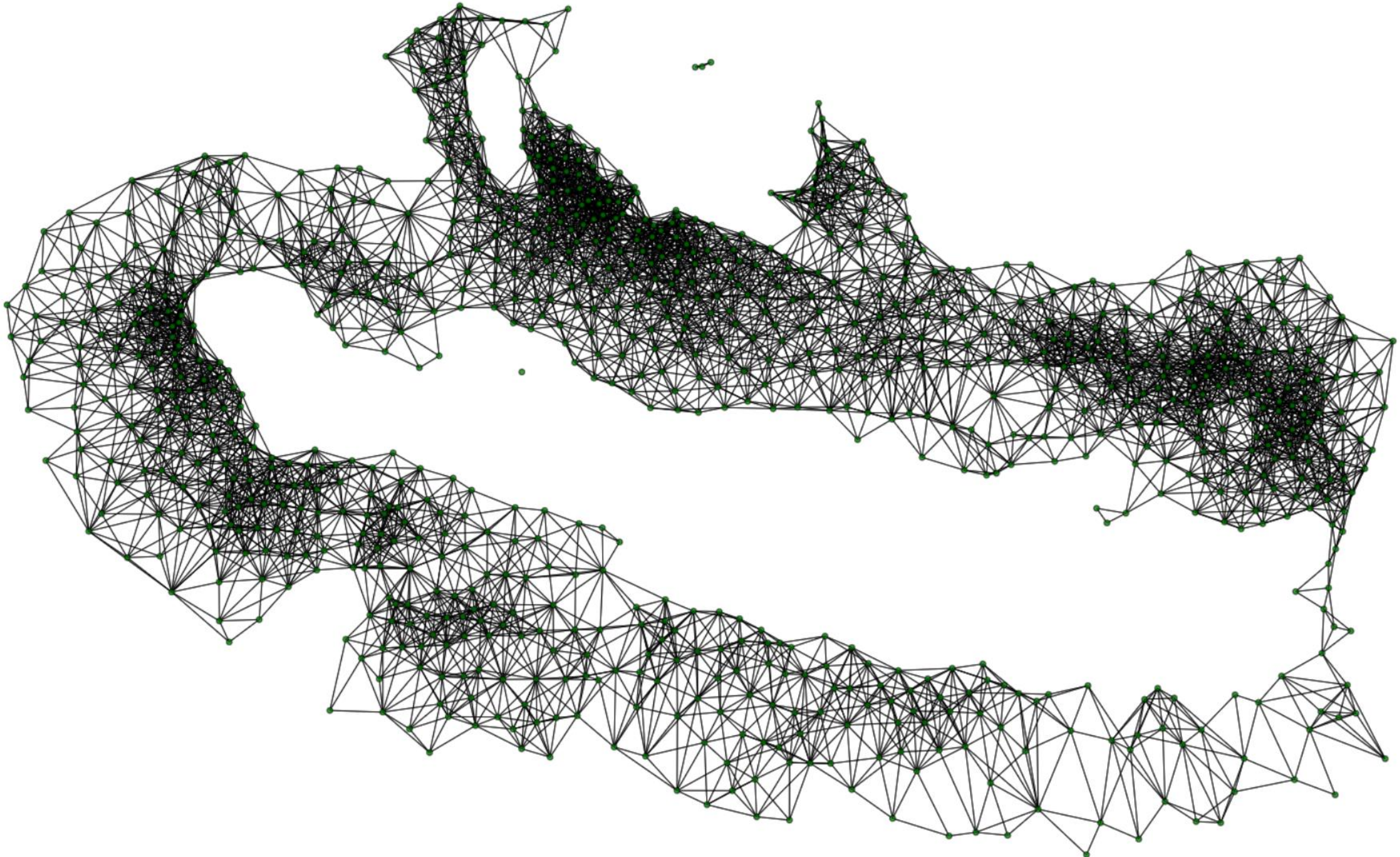
Fayyaz Minhas, Whole Slide Images Are Graphs, 2020. https://www.youtube.com/watch?v=Of1u0i7roS0.



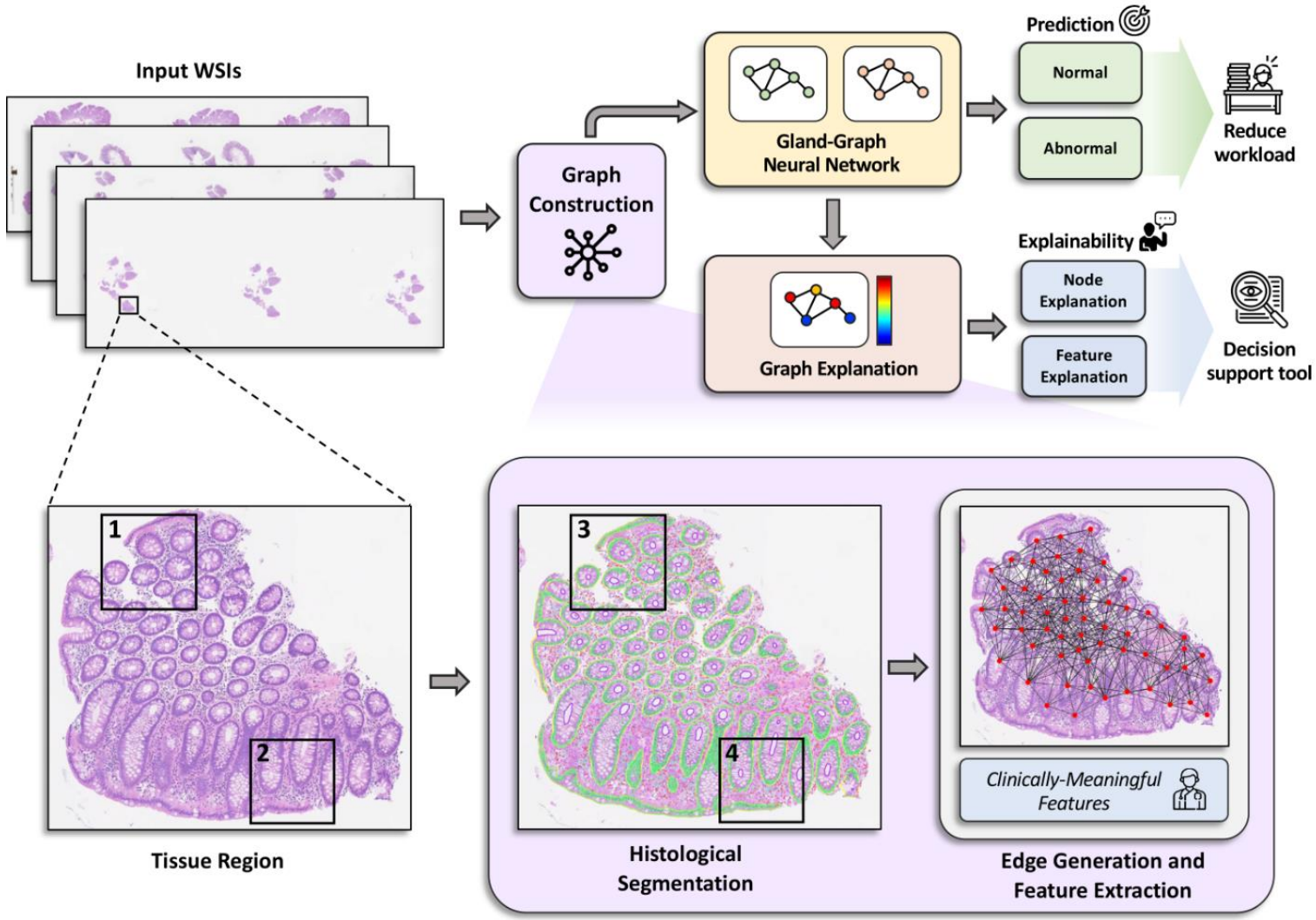
Gland Size

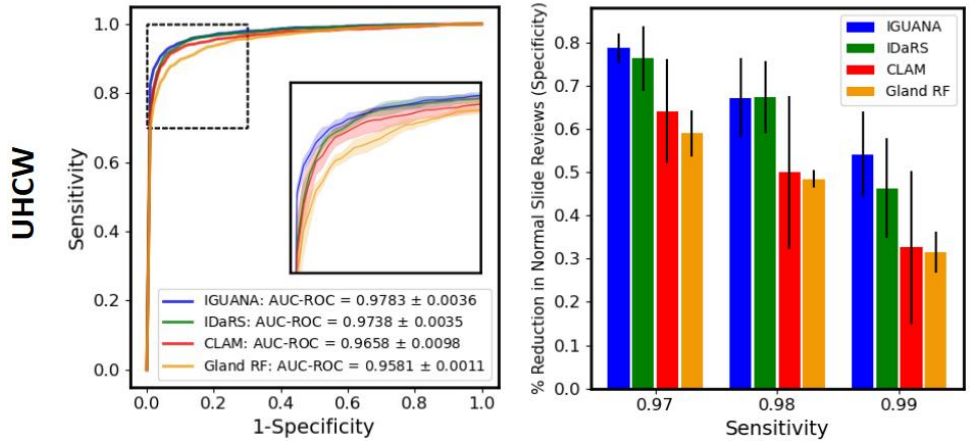
Lumen Shape Irregularity



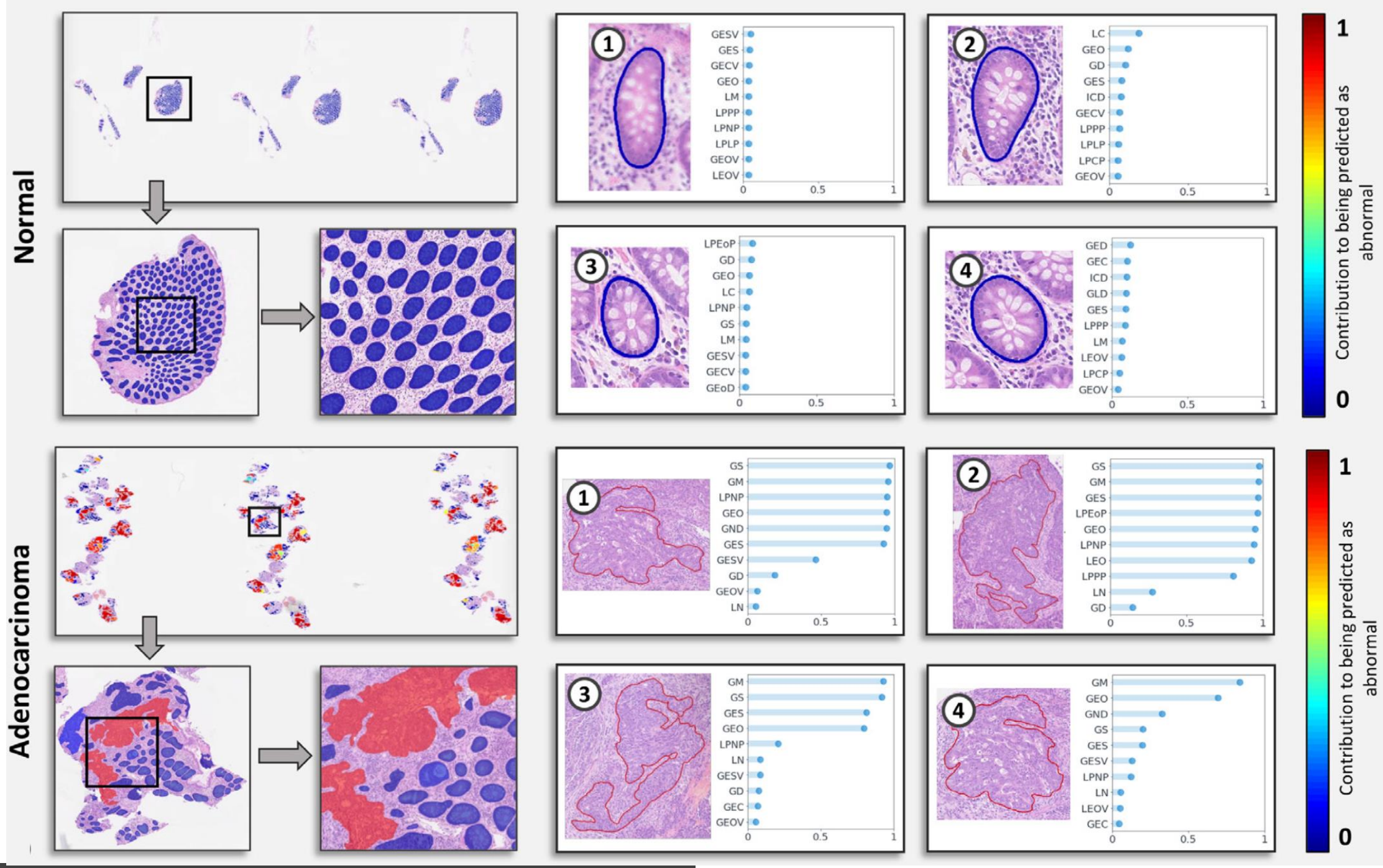


IGUANA





Graham, Simon, Fayyaz Minhas, Mohsin Bilal, Mahmoud Ali, Yee Wah Tsang, Mark Eastwood, Noorul Wahab, et al. "Screening of Normal Endoscopic Large Bowel Biopsies with Interpretable Graph Learning: A Retrospective Study." *Gut*, May 12, 2023. https://doi.org/10.1136/gutjnl-2023-329512.
Demo: https://tiademos.dcs.warwick.ac.uk/bokeh_app?demo=iguana


Normal
Adenocarcinoma
1
GESV
GES
GECV
GEO
LM
LPPP
LPNP
LPLP
GEOV
LEOV
2
LC
GEO
GD
GES
ICD
GECV
LPPP
LPLP
LPCP
GEOV
3
LPEoP
GD
GEO
LC
LPNP
GS
LM
GESV
GECV
GEoD
4
GED
GEC
ICD
GLD
GES
LPPP
LM
LEOV
LPCP
GEOV
0
0.5
1
Contribution to being predicted as abnormal
GS
GM
LPNP
GEO
GND
GES
GESV
GD
GEOV
LN
LPEoP
LEO
GEC

Are Transformers (secretly) GNNs?

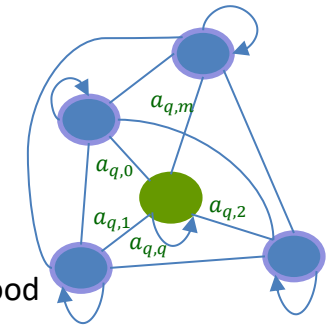
Assume

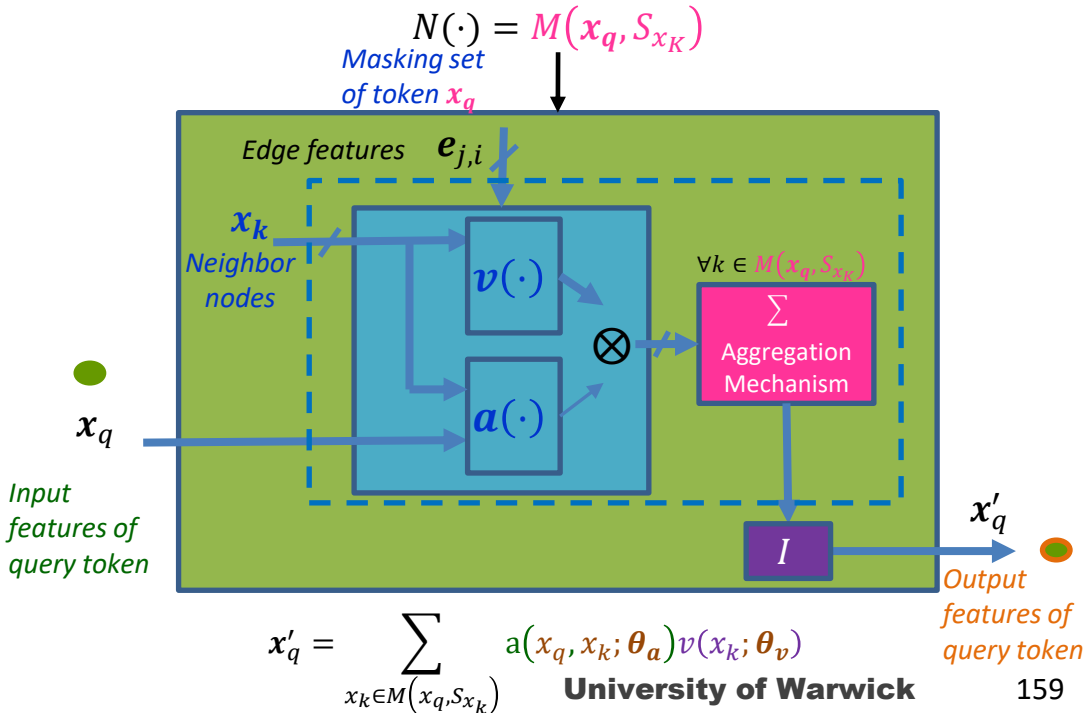
- We have a set of "nodes" S_{x_K} and for a given "query" node x_q, we have a masking set set $M(x_q, S_{x_K})$ (for simplicity assume $M(x_q, S_{x_K}) = S_{x_K}$)
- Each node is connected to all other nodes including itself (i.e., neighborhood $N(x_q) = M(x_q, S_{x_K})$) (Fully Connected Graph)

Now consider a specific graph neural network layer in which

- $\gamma^{(k)}(\mathrm{a}, \mathrm{b}) = \mathrm{b}$
- $\phi^{(l)}\left(\boldsymbol{x}_i^{(l-1)}, \boldsymbol{x}_j^{(l-1)}, \boldsymbol{e}_{j,i}\right) = \mathrm{a}\left(\boldsymbol{x}_i^{(l-1)}, \boldsymbol{x}_j^{(l-1)}; \boldsymbol{\theta}_a\right) v\left(\boldsymbol{x}_j^{(l-1)}; \boldsymbol{\theta}_v\right)$
- $\oplus_{j \in N(q)}(\cdot) = \sum_{\boldsymbol{x_k} \in M\left(\boldsymbol{x_q}, S_{x_k}\right)}(\cdot)$

- Then the output of the GNN layer:
  - $\boldsymbol{x}_i^{(l)} = \gamma^{(l)}\left(\boldsymbol{x}_i^{(l-1)}, \oplus_{j \in N(i)} \phi^{(l)}\left(\boldsymbol{x}_i^{(l-1)}, \boldsymbol{x}_j^{(l-1)}, \boldsymbol{e}_{j,i}\right)\right)$
- Becomes (with notation $\boldsymbol{x}_i^{(l-1)} = x_q$, $\boldsymbol{x}_j^{(l-1)} = x_k$ and $\boldsymbol{x}_i^{(l)} = \boldsymbol{x}'_q$)
  - $\boldsymbol{x}'_q = \sum_{x_k \in M\left(x_q, S_{x_k}\right)} \mathrm{a}\left(x_q, x_k; \boldsymbol{\theta}_a\right) v(x_k; \boldsymbol{\theta}_v)$
- Which is an attention layer (assuming position encoding is built into node features)
- **An attention layer is a special case of a GNN layer!**
  - **Attention scores can be viewed as pairwise weights of edges between nodes**





Reading on Graph Neural Networks

- Xu*, Keyulu, Weihua Hu*, Jure Leskovec, and Stefanie Jegelka. “How Powerful Are Graph Neural Networks?,” 2023. https://openreview.net/forum?id=ryGs6iA5Km.
- Kanatsoulis, Charilaos I., and Alejandro Ribeiro. “Graph Neural Networks Are More Powerful Than We Think.” arXiv, October 2, 2022. https://doi.org/10.48550/arXiv.2205.09801.
- Bronstein, Michael M., Joan Bruna, Taco Cohen, and Petar Veličković. “Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges.” arXiv, May 2, 2021. https://doi.org/10.48550/arXiv.2104.13478.
- http://web.stanford.edu/class/cs224w/
- Libraries
  - PyTorch Geometric
  - DGL
  - Topological Neural Networks

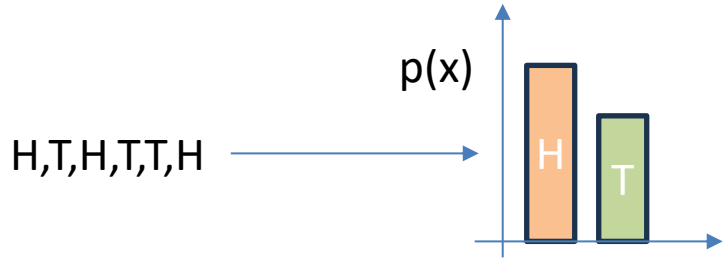
GENERATIVE MACHINE LEARNING

Creating noise from data is easy; creating data from noise is generative modeling.*

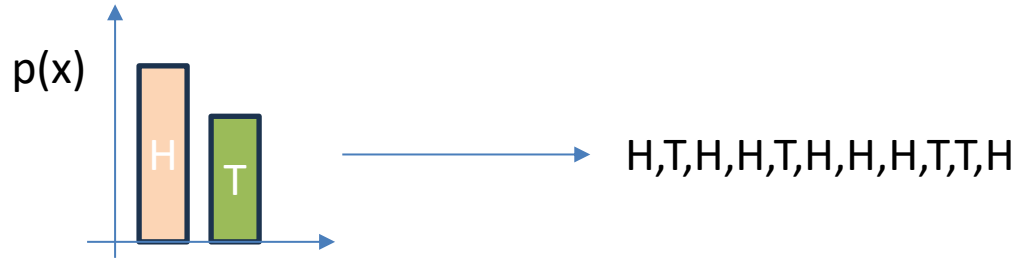
[*] Song, Yang, Jascha Sohl-Dickstein, Diederik P. Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. “Score-Based Generative Modeling through Stochastic Differential Equations.” arXiv, February 10, 2021. https://doi.org/10.48550/arXiv.2011.13456.

Background: Introduction to Sampling

- **Empirical distribution Modelling**: Making a distribution from observations (Density Estimation)
  - Example:
    - Observations: {H,T,H,T,H}
    - P(H) = 3/5 = 0.6, P(T) = 2/5 = 0.4
    - Shown as probability distribution (normalized histogram)

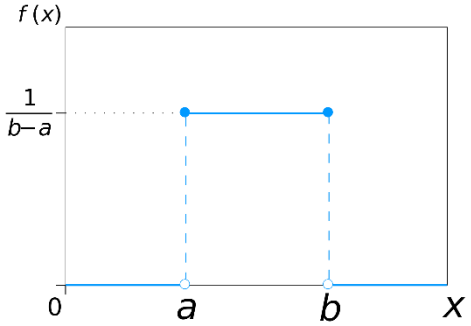


- **Sampling** from a distribution
  - Assume you are given a probability distribution p(x), then if you "sample" from it, you will be generating samples x which when observed will give you p(x)
  - Example
    - Given: P(H) = 0.6, P(T) = 0.4
    - Generated Samples: {H,T,H,T,H,T,H,H,T,H}



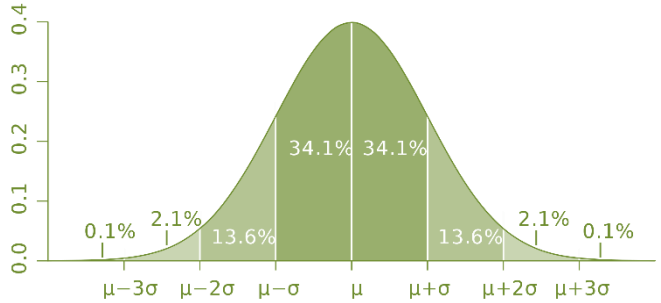
Background: Generating samples

- Can we **generate** samples of a target distribution using samples from a source distribution as input?



$x \sim U(a = 0, b = 1)$

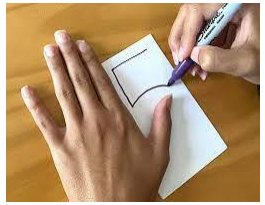
X = np.random.rand(N)

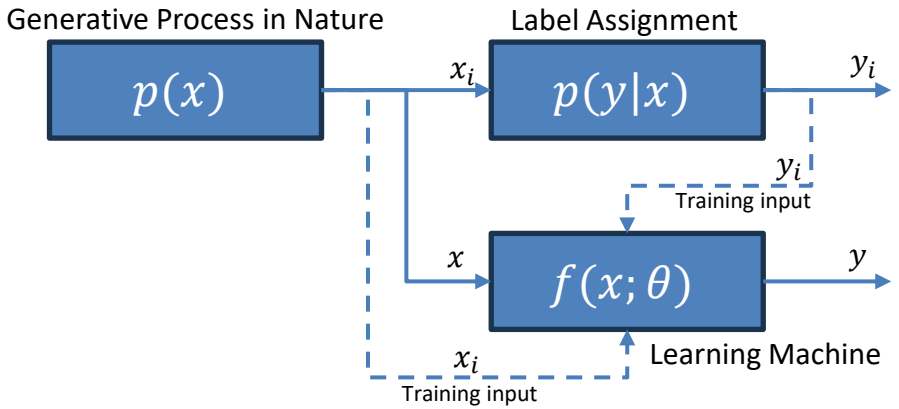


$z \sim N(\mu = 0.5, \sigma = 1)$

Z = np.random.randn(N)+0.5

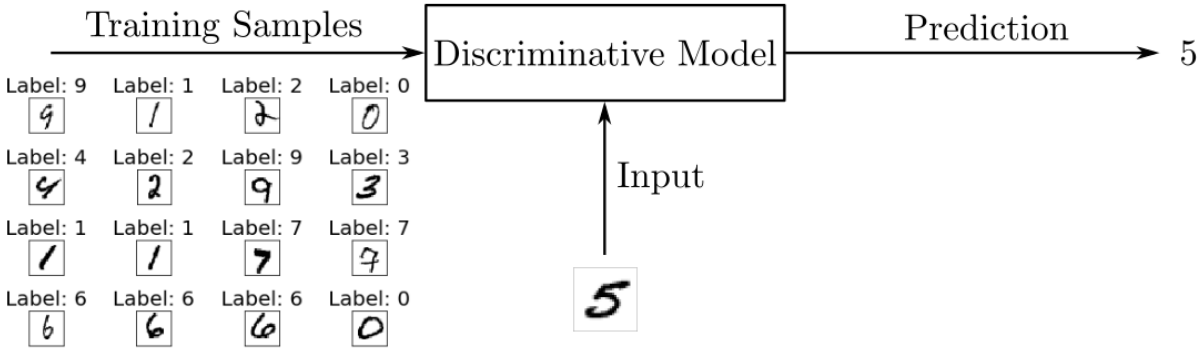
A generative look at Machine Learning





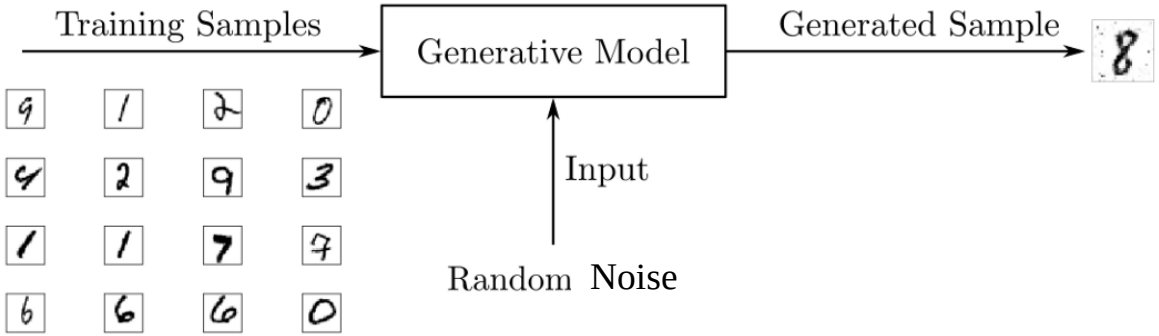
Fundamental aim of a discriminative model

Learn a model of p(y|x) from observations



Fundamental aim of a Generative Model

Learn a model of $p(x)$ or $p(x|y)$ from observations to generate samples from random noise input

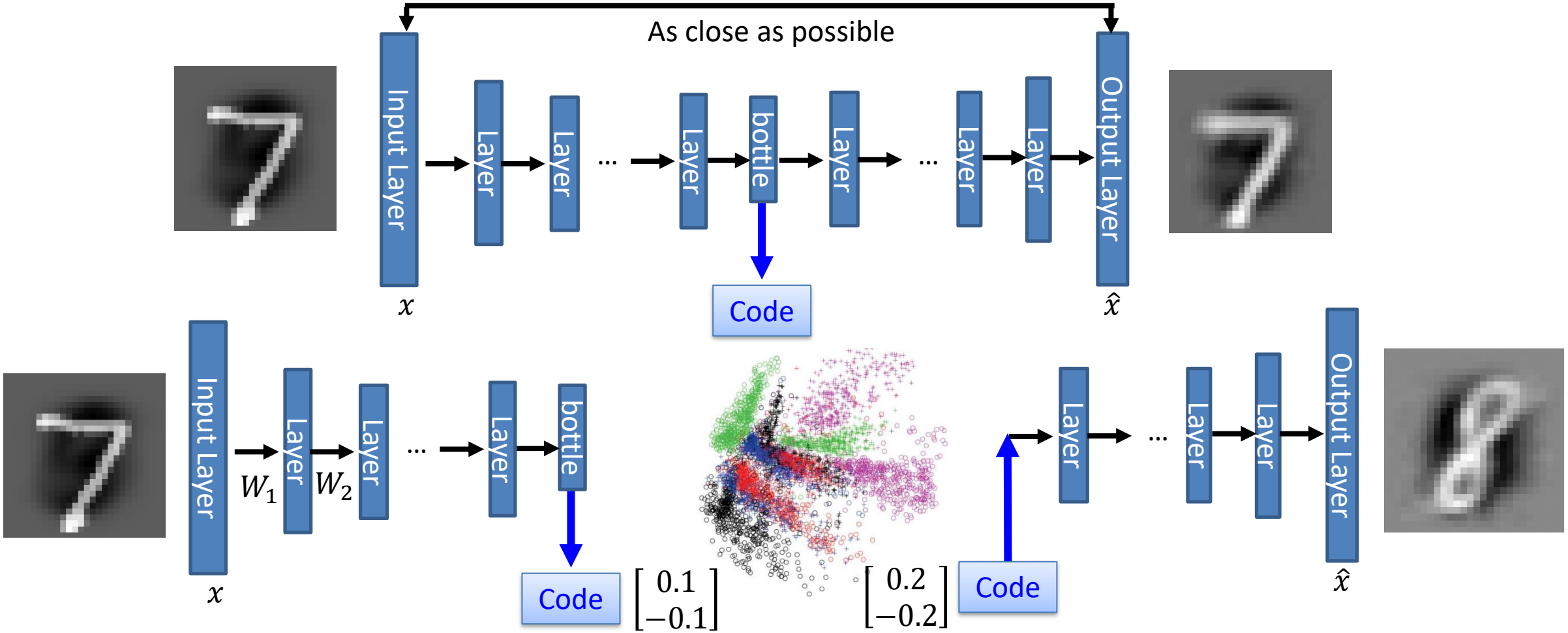


Lex Fridman. *Complete Statistical Theory of Learning (Vladimir Vapnik) | MIT Deep Learning Series*, 2020.
https://www.youtube.com/watch?v=Ow25mjFjSmg.

Generating data with machine learning

- Can we generate examples that follow the same distribution as a given set of examples using noise as input?
- Sampling from the multi-dimensional distribution of data
- How?
  - Density Modelling
    - Modelling the Probability of observing a given point $p(x)$
    - Once I have an explicit or implicit $p(x)$, I can sample from that distribution to generate an example

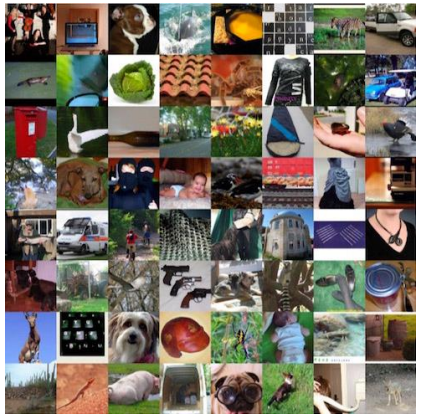
Generating Data with Autoencoders



Tutorial Implementation: https://github.com/foxtrotmike/CS909/blob/master/autoenncoders.ipynb

Generative Models

- Can we build a model to approximate a data distribution from given examples?



Real image (training data) ~ $p_{data}(x)$

Generated samples ~ $p_{model}(x)$

Want to learn $p_{model}(x)$ similar to $p_{data}(x)$

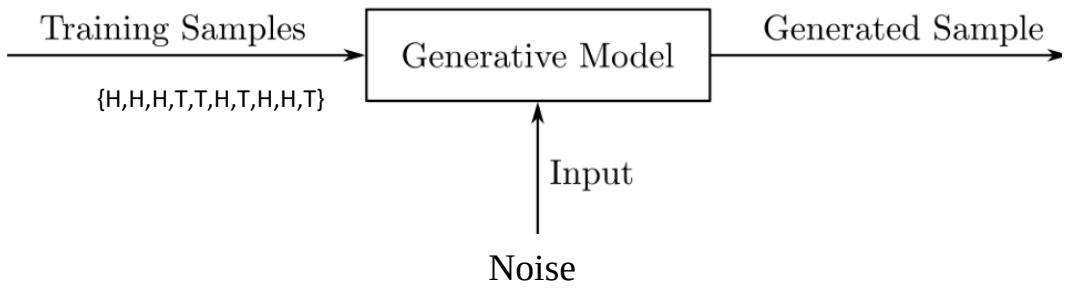
Density estimation: a core problem in unsupervised learning

Several flavors:

- **Explicit density estimation**: explicitly define and solve for $p_{model}(x)$
  - Algorithms: Gaussian Mixture Models, Kernel Density Estimation, Variational Autoencoders
- **Implicit density estimation**: learn model that can sample from $p_{model}(x)$ w/o explicitly defining it
  - Algorithms: Vanilla autoencoder, Generative adversarial networks (GANs), Diffusion Models, Normalizing Flows

A Simple Generative Machine Learning Example

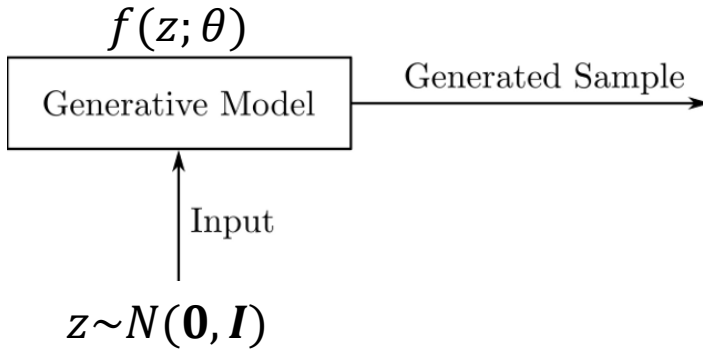
- Nature
  - A coin with p(x=H)=0.7 and p(x=T)=0.3
  - Generates data
- Given Data
  - {H,H,H,T,T,H,T,H,H,T}



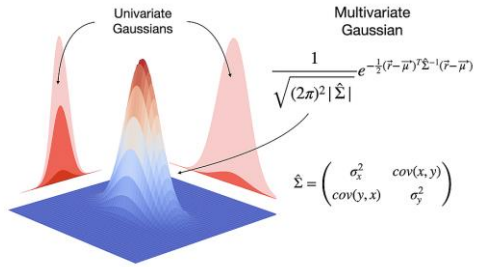
- Goal of Generative Learning
  - Make a machine learning model that can generate data (heads or tails) that follows the same distribution as data from the real world or natural process.
  - The difference between the probability distributions of real and generated samples should be small

REO for Generative Models

- Goal
  - Given a set of real-world examples: $x \sim p(x)$. $p(x)$ is not explicitly known.
  - Learn parameters θ of the model $f(z;\theta)$ so that the examples generated by the model follow the same distribution as the real-world examples $x \sim p(x)$

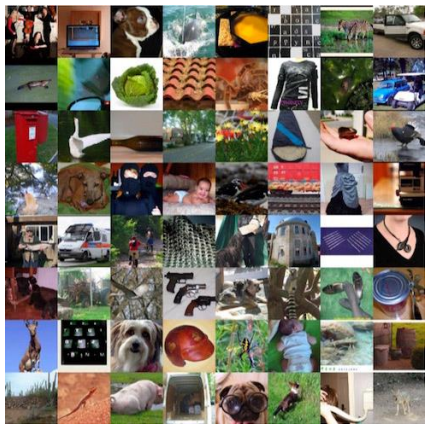


- Representation: $x = f(z;\theta)$ with $z \sim Noise$
  - Let's denote the distribution of examples generated by this model as $p_\theta(x)$.
  - Note that the model may not have an explicit internal formula for this distribution.



- Evaluation:
  - Differences between the probability distribution of x in nature $p(x)$ and of the generated samples $p_\theta(x)$ from $f(z;\theta)$
    - That is, if I sample from $p(x)$ or if I sample from$p_\theta(x)$, the real and generated samples are similar

- Optimization
  - Use gradient descent to optimize for θ

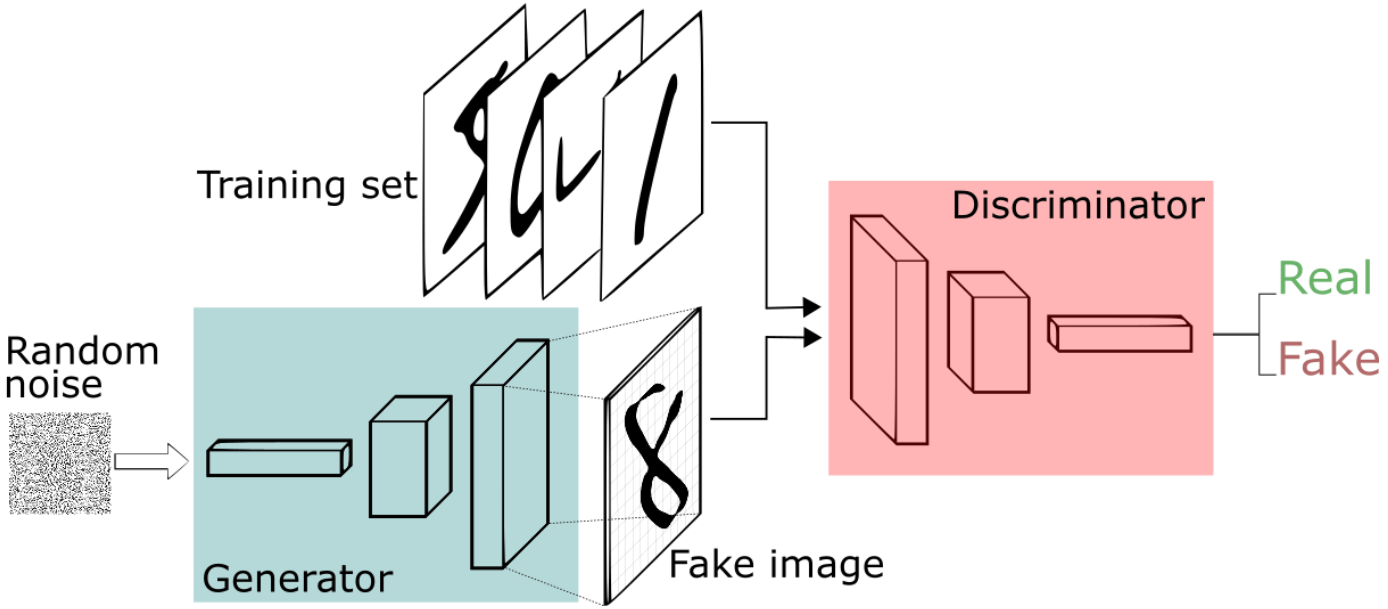


Real image (training data) ~ $p(x)$
$p(x)$ is not given.

Generated samples ~ $p_\theta(x)$
$p_\theta(x)$ may be implicit.

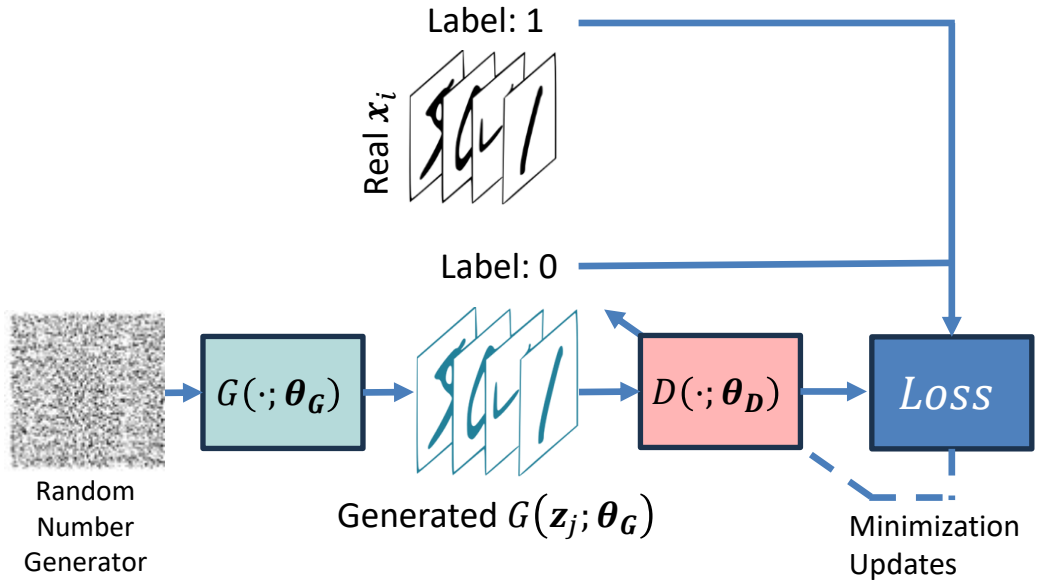
Generative Adversarial Networks

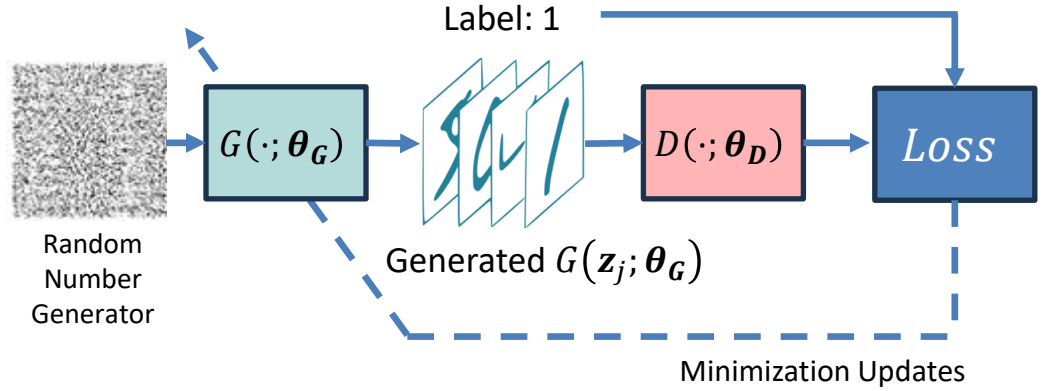
- Use “Adversarial Training” to train a generator and discriminator simultaneously
- Generator: Generate samples from noise
- Discriminator: Detect “fake” or generated samples



Adversarial Training in a GAN

- GAN Training the goal is to:
  - **Train the discriminator** to be good at detecting fakes
    - Simple classification: Discriminator should produce 1 for real and 0 for generated
    - $\min_{\boldsymbol{\theta_D}} \sum_{x_i \in R} l(D(\boldsymbol{x}_i; \boldsymbol{\theta_D}), 1) + \sum_{z_j \sim N} l(D(G(\boldsymbol{z}_j; \boldsymbol{\theta_G}); \boldsymbol{\theta_D}), 0)$
  - **Train the generator** to be so good that the discriminator labels generated samples as "Real"
    - The generator exploits the discriminator's ability or knowledge to distinguish between real and generated samples to its advantage
    - The generator is optimized such that the discriminator produces 1 for generated examples
    - $\min_{\boldsymbol{\theta_G}} \sum_{\boldsymbol{z}_j \sim N} l(D(G(\boldsymbol{z_j}; \boldsymbol{\theta_G}); \boldsymbol{\theta_D}), 1)$
    - OR equivalently, the generator is optimized such that the discriminator generates errors in classifying generated examples (note the max below)
    - $\max_{\boldsymbol{\theta_G}} \sum_{\boldsymbol{z_j} \sim N} l(D(G(z_j; \boldsymbol{\theta_G}); \boldsymbol{\theta_D}), 0)$
  - Can also add additional loss terms for quality/realism etc.





GAN Tutorial

Open in Colab

A Barebones GAN in PyTorch for generating coin flips

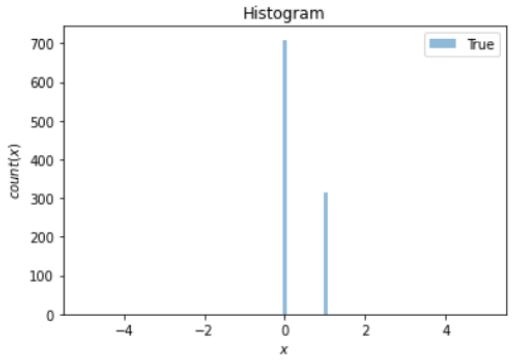
By Fayyaz Minhas

Let's consider a very simple coing toss as a process that generates coin flips with a probability of 0.3 of producing heads. We can describe the underlying probability distribution for this generative process (coin toss) as $p(x)$ where $x \in \{H = 1, T = 0\}$ is sampled from $p(x)$, i.e., $x \backsim p(x)$. We would like to use a Generative Adversarial Network (GAN) to model this process using a number of data samples or observations from the original process for training. Specifically, we would like to have a GAN with such a generator that you (and its discriminator) wouldn't be able to tell if a series of coin tosses has been generated using the GAN or the underlying true process! In more mathematical terms, we would like to train a generative model $x = G(z; \theta_G)$ that can generate samples x using Normally distributed random input ($z \backsim N(0, 1)$) such that the probability distribution of these generated samples $p_G(x)$ is close to $p(x)$ without knowing $p(x)$ in advance or explicitly modelling $p_G(x)$.

Using a GAN is an overkill for this simple task and there are much simpler and more effective ways of modelling this simple problem. However, this GAN based solution is intended to help you understand how GANs can model complex densities implicitly and can be used to generate samples that mimic the true or natural generative process.

We first simulate the coin toss and generate 1024 training samples below. The histogram shows the (sample estimate of) the true density.

```
]: """
   A toy GAN to generate coin tosses
   """

   # Let's model the natural density and generate some data using that

   import torch
   from torch import nn

   import math
   import matplotlib.pyplot as plt
   import numpy as np
   train_data_length = 1024
   def cointoss(t):
     phead = 0.3
     return 1.0*(t
```

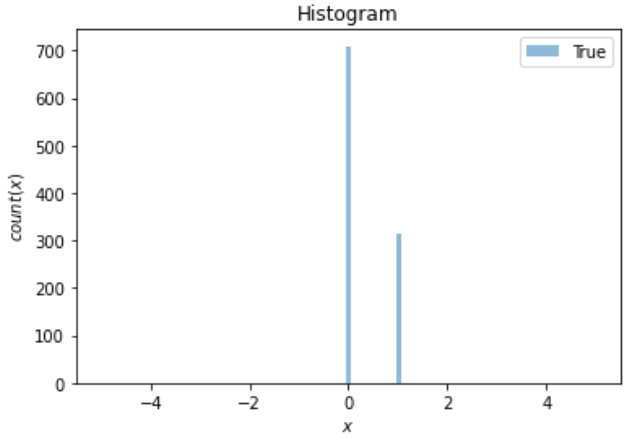



```
Example Data tensor([[0., 0., 0.,  ..., 0.,
1., 1.]])
```

https://github.com/foxtrotmike/CS909/blob/master/simpleGAN.ipynb

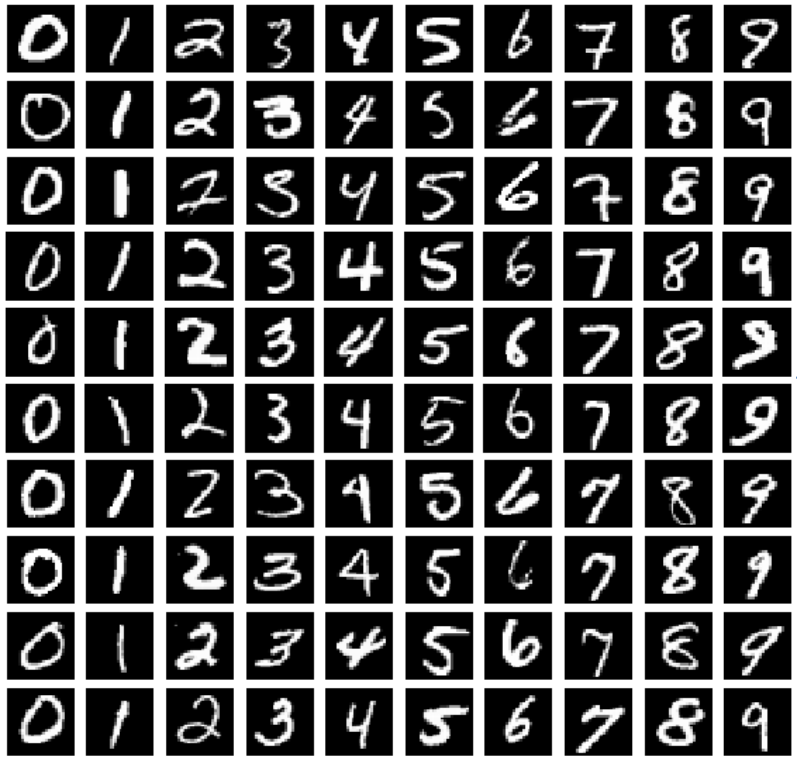
How to go from generating coin flips to images?

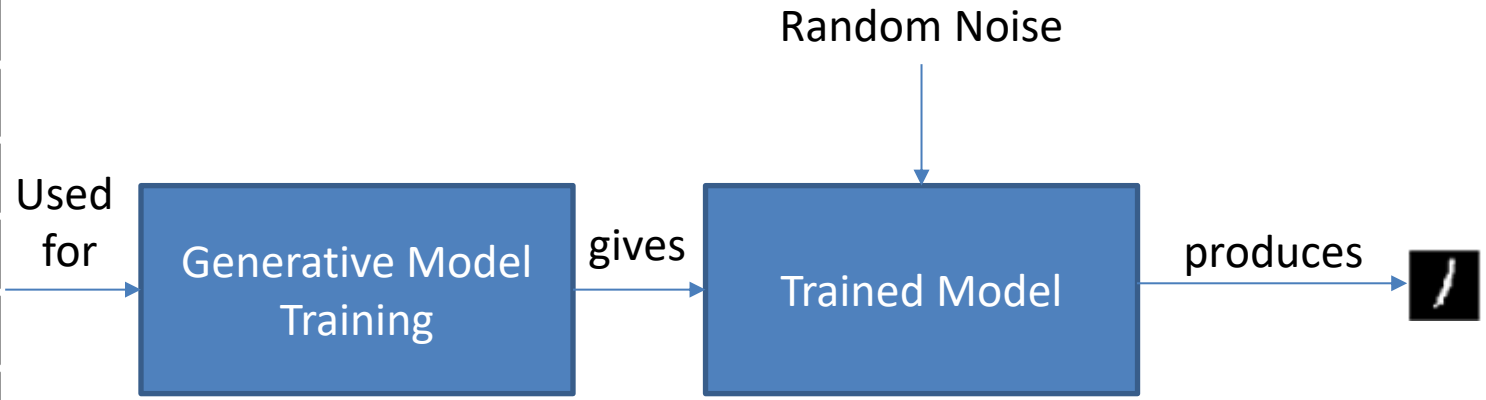
- Assume you are given B&W images for training a GAN to generate more images like that.
- Let's look at a single pixel location in each image
  - We have a distribution of pixel values across all images at that location
    - We would like our GAN to generate data according to that distribution at that pixel location
    - Naïve idea: Have multiple GANs – one for each pixel location
      - Assumes each pixel is independent of the other
      - Computationally intensive
    - We can train a single GAN to generate a multi-dimensional probability distribution by using a multi-output generator.



Unconditional vs Conditional Generation

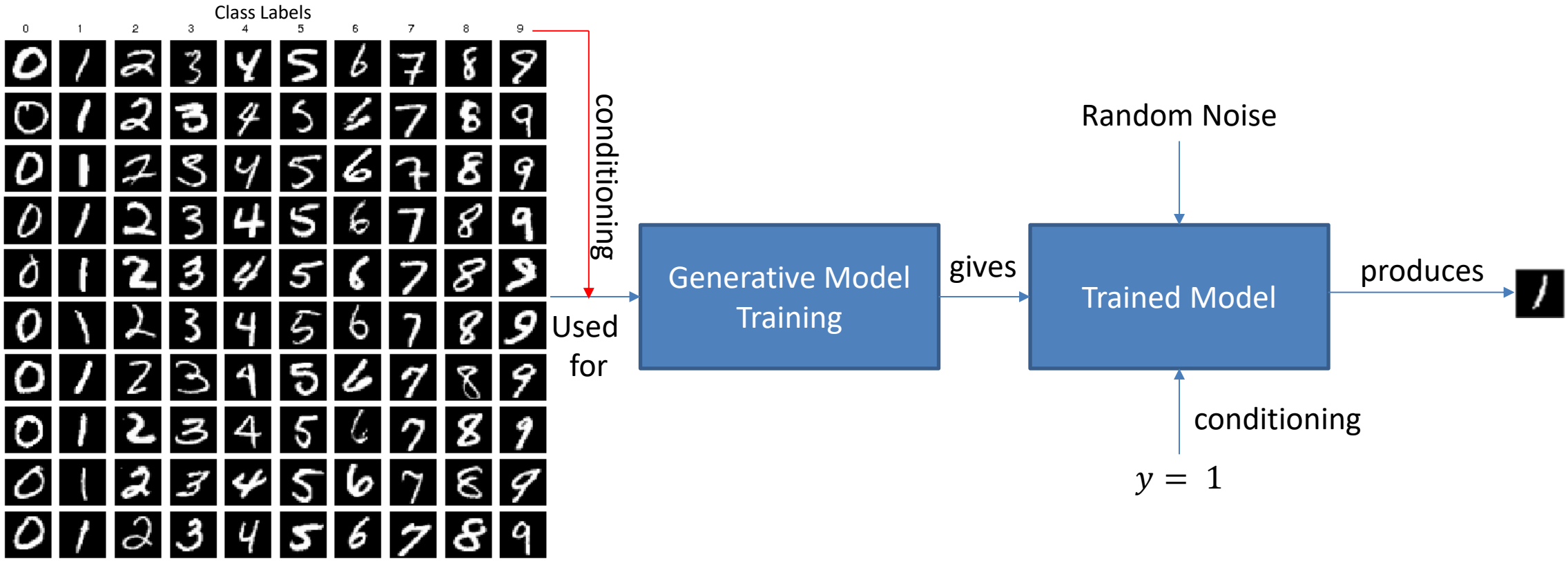
- Unconditional Generative Modelling
  - Simple model the probability distribution of the data p(x)
    - Example: Generating images without paying any regard to the digit





Unconditional vs Conditional Generation

- Conditional Generative Modelling
  - Model the distribution p(x|y) of data x conditioned on a variable y
    - Example: Generating images for a given digit

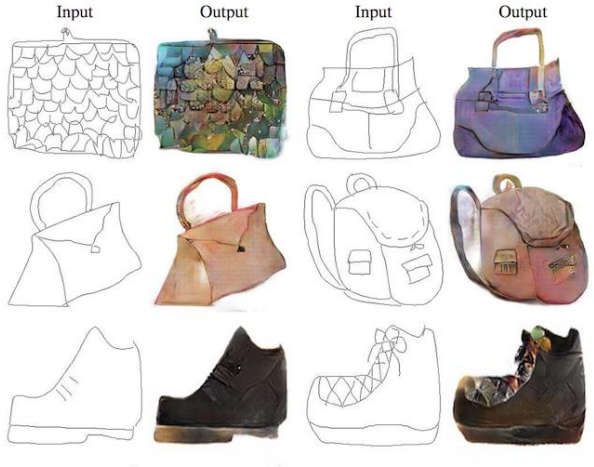


GANs Applications

- GANs have some impressive applications
  - Synthetic Image Generation
  - Speech Generation
  - Image to Image Translation
  - Style Transfer
  - Deep Fakes

Barebones GAN
https://github.com/foxtrotmike/CS909/blob/master/simpleGAN.ipynb

Raevskiy, Mikhail. "Write Your First Generative Adversarial Network Model on PyTorch." Medium, August 31, 2020. https://medium.com/dev-genius/write-your-first-generative-adversarial-network-model-on-pytorch-7dc0c7c892c7.



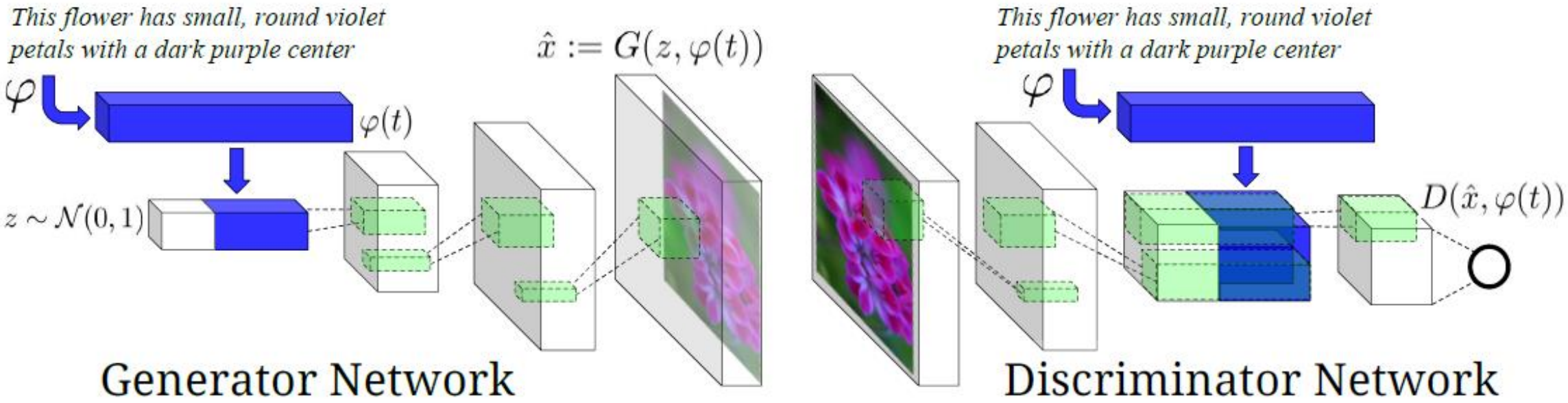
Ian Goodfellow. NIPS 2016 Tutorial: Generative Adversarial Networks. https://arxiv.org/abs/1701.00160
https://github.com/eriklindernoren/PyTorch-GAN
https://machinelearningmastery.com/impressive-applications-of-generative-adversarial-networks/
https://affinelayer.com/pixsrv/

The GAN Zoo

- GAN - Generative Adversarial Networks
- 3D-GAN - Learning a Probabilistic Latent Space of Object Shapes via 3D Generative-Adversarial Modeling
- acGAN - Face Aging With Conditional Generative Adversarial Networks
- AC-GAN - Conditional Image Synthesis With Auxiliary Classifier GANs
- AdaGAN - AdaGAN: Boosting Generative Models
- AEGAN - Learning Inverse Mapping by Autoencoder based Generative Adversarial Nets
- AffGAN - Amortised MAP Inference for Image Super-resolution
- AL-CGAN - Learning to Generate Images of Outdoor Scenes from Attributes and Semantic Layouts
- ALI - Adversarially Learned Inference
- AM-GAN - Generative Adversarial Nets with Labeled Data by Activation Maximization
- AnoGAN - Unsupervised Anomaly Detection with Generative Adversarial Networks to Guide Marker Discovery
- ArtGAN - ArtGAN: Artwork Synthesis with Conditional Categorial GANs
- b-GAN - b-GAN: Unified Framework of Generative Adversarial Networks
- Bayesian GAN - Deep and Hierarchical Implicit Models
- BEGAN - BEGAN: Boundary Equilibrium Generative Adversarial Networks
- BiGAN - Adversarial Feature Learning
- BS-GAN - Boundary-Seeking Generative Adversarial Networks
- CGAN - Conditional Generative Adversarial Nets
- CaloGAN - CaloGAN: Simulating 3D High Energy Particle Showers in Multi-Layer Electromagnetic Calorimeters with Generative Adversarial Networks
- CCGAN - Semi-Supervised Learning with Context-Conditional Generative Adversarial Networks
- CatGAN - Unsupervised and Semi-supervised Learning with Categorical Generative Adversarial Networks
- CoGAN - Coupled Generative Adversarial Networks
- Context-RNN-GAN - Contextual RNN-GANs for Abstract Reasoning Diagram Generation
- C-RNN-GAN - C-RNN-GAN: Continuous recurrent neural networks with adversarial training
- CS-GAN - Improving Neural Machine Translation with Conditional Sequence Generative Adversarial Nets
- CVAE-GAN - CVAE-GAN: Fine-Grained Image Generation through Asymmetric Training
- CycleGAN - Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks
- DTN - Unsupervised Cross-Domain Image Generation
- DCGAN - Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks
- DiscoGAN - Learning to Discover Cross-Domain Relations with Generative Adversarial Networks
- DR-GAN - Disentangled Representation Learning GAN for Pose-Invariant Face Recognition
- DualGAN - DualGAN: Unsupervised Dual Learning for Image-to-Image Translation
- EBGAN - Energy-based Generative Adversarial Network
- f-GAN - f-GAN: Training Generative Neural Samplers using Variational Divergence Minimization
- FF-GAN - Towards Large-Pose Face Frontalization in the Wild
- GAWWN - Learning What and Where to Draw
- GeneGAN - GeneGAN: Learning Object Transfiguration and Attribute Subspace from Unpaired Data
- Geometric GAN - Geometric GAN
- GoGAN - Gang of GANs: Generative Adversarial Networks with Maximum Margin Ranking
- GP-GAN - GP-GAN: Towards Realistic High-Resolution Image Blending
- IAN - Neural Photo Editing with Introspective Adversarial Networks
- iGAN - Generative Visual Manipulation on the Natural Image Manifold
- IcGAN - Invertible Conditional GANs for image editing
- ID-CGAN - Image De-raining Using a Conditional Generative Adversarial Network
- Improved GAN - Improved Techniques for Training GANs
- InfoGAN - InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets
- LAGAN - Learning Particle Physics by Example: Location-Aware Generative Adversarial Networks for Physics Synthesis
- LAPGAN - Deep Generative Image Models using a Laplacian Pyramid of Adversarial Networks

https://github.com/hindupuravinash/the-gan-zoo

Text-to-Image Synthesis



S. Reed, Z. Akata, X. Yan, L. Logeswaran, B. Schiele, H. Lee, "Generative Adversarial Text-to-Image Synthesis", ICML 2016

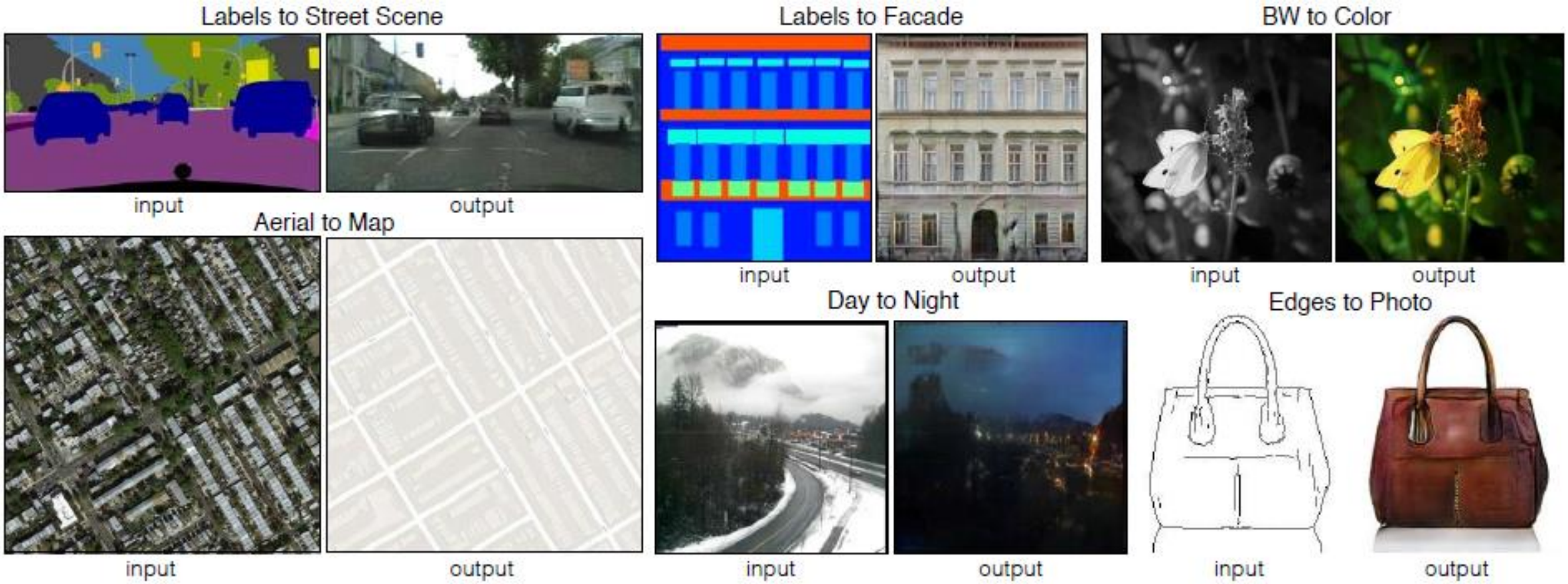
H. Zhang, T. Xu, H. Li, S. Zhang, X. Huang, X. Wang, D. Metaxas, "StackGAN: Text to Photo-realistic Image Synthesis with Stacked Generative Adversarial Networks", arXiv prepring, 2016

S. Reed, Z. Akata, S. Mohan, S. Tenka, B. Schiele, H. Lee, "Learning What and Where to Draw", NIPS 2016

Text to Image – Results

Caption	Image
a pitcher is about to throw the ball to the batter	
a group of people on skis stand in the snow	
a man in a wet suit riding a surfboard on a wave	

Image-to-image Translation

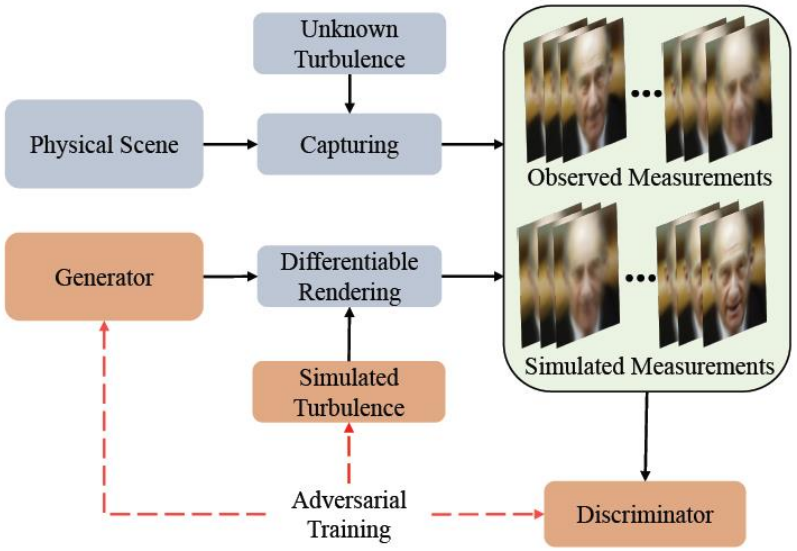


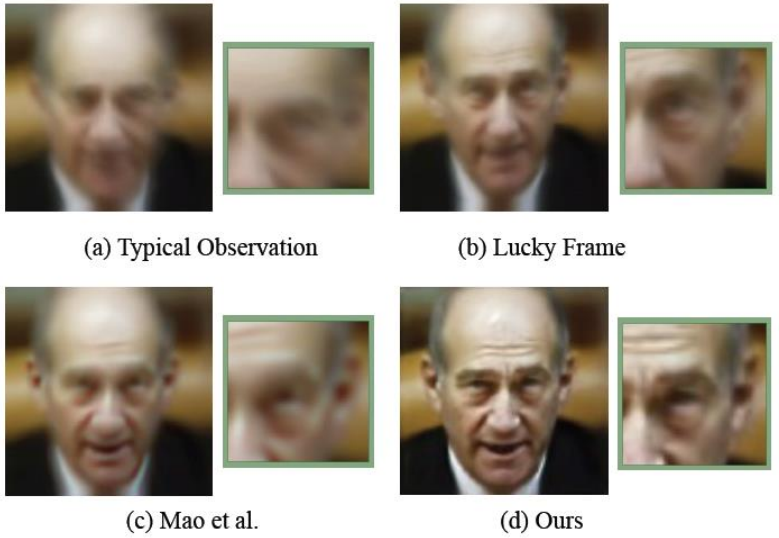
P. Isola, J.-Y. Zhu, T. Zhou, A.A. Efros, "Image-to-Image Translation with Conditional Adversarial Networks", arXiv preprint, 2016

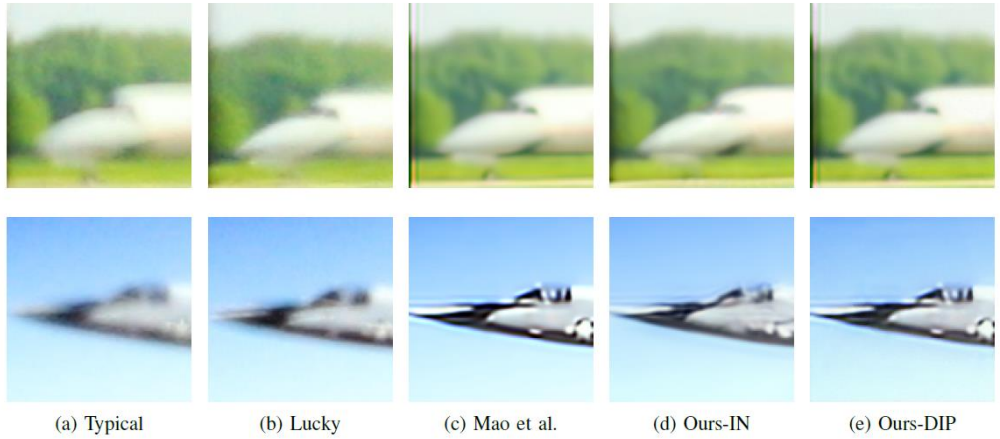
Unpaired Transformation – Cycle GAN, Disco GAN

Transform an object from one domain to another ***without paired data***

TurbuGAN



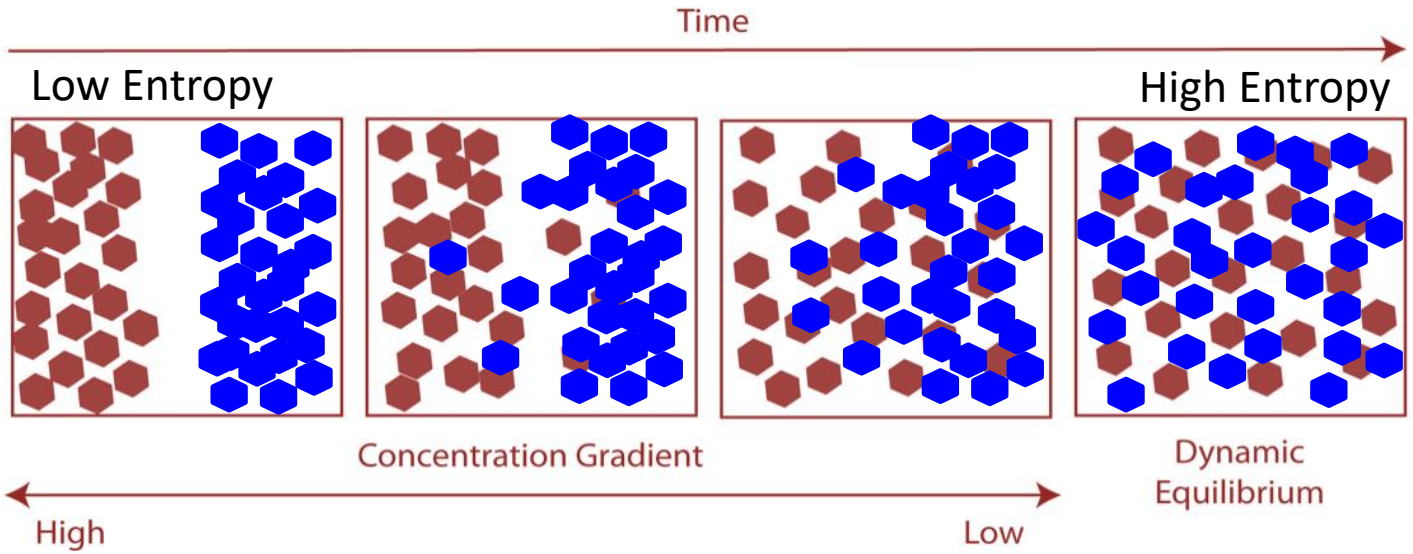




Feng, Brandon Yushan, Mingyang Xie, and Christopher A. Metzler. "TurbuGAN: An Adversarial Learning Approach to Spatially-Varying Multiframe Blind Deconvolution with Applications to Imaging Through Turbulence." arXiv, August 22, 2022. https://doi.org/10.48550/arXiv.2203.06764.

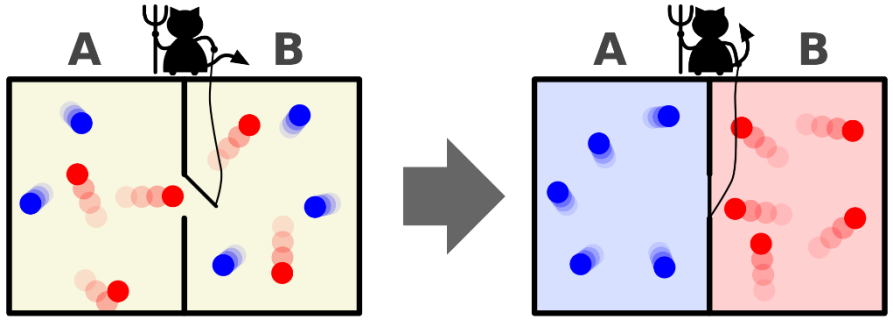
Diffusion Models

- What is diffusion?

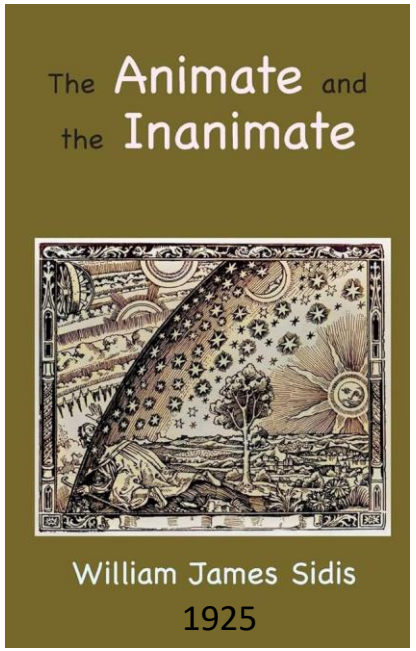


- Can we learn to reverse it?

Maxwell's Demon



https://en.wikipedia.org/wiki/Maxwell%27s_demon

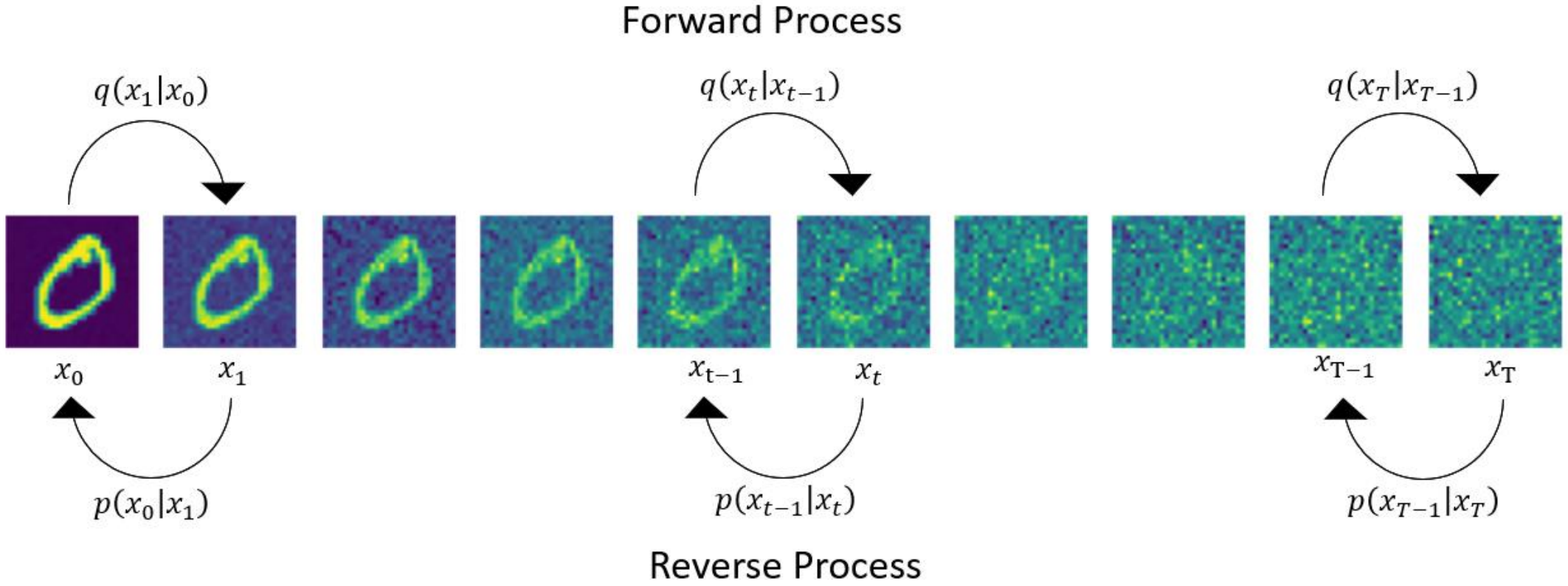


PREFACE

This work sets forth a theory which is speculative in nature, there being no verifying experiments. It is based on the idea of the reversibility of everything in time; that is, that every type of process has its time-image, a corresponding process which is its exact reverse with respect to time. This accounts for all physical laws but one, namely, the second law of thermodynamics. This law has been found during the nineteenth century to be a source of a great deal of difficulty. The eminent physicist, Clerk-Maxwell, in the middle of the nineteenth century, while giving a proof of that law, admitted that reversals are possible by imagining a "sorting demon" who could sort out the smaller particles, and separate the slower ones from the faster ones. This second law of thermodynamics brought in the idea of energy-level, of unavailable energy (or "entropy" as it was called by Clausius) which was constantly increasing.

Diffusion Models

- Main idea: Learn to reverse a “diffusion” process



Tutorial: https://github.com/wgrgwrght/Simple-Diffusion/blob/main/SimpleDiffusion.ipynb

Dhariwal, Prafulla, and Alex Nichol. “Diffusion Models Beat GANs on Image Synthesis.” arXiv, June 1, 2021. https://doi.org/10.48550/arXiv.2105.05233.

Nichol, Alex, and Prafulla Dhariwal. “Improved Denoising Diffusion Probabilistic Models.” arXiv, February 18, 2021. https://doi.org/10.48550/arXiv.2102.09672.

Diffusion Models

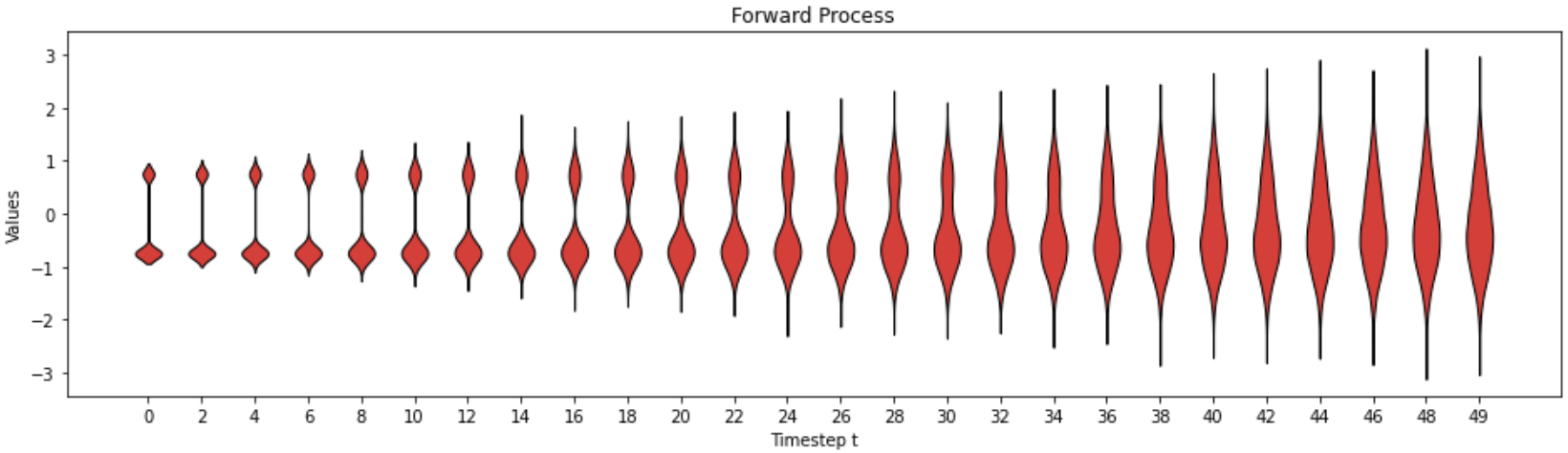
- **Generation by learning to reverse entropy**
- **Forward Process:** Generate noisy signals from data
  - Data distribution gets gradually converted to noise
- **Reverse Process:** Learn to denoise
  - Using a neural network $\epsilon_\theta(x_t, t)$ with weights θ which takes the noisy data x_t as input along with the time step t (and possibly other "conditioning" variables) to output an estimate of the noise ϵ_t that has been added to x_0 to generate x_t. This is achieved by solving the following optimization problem:

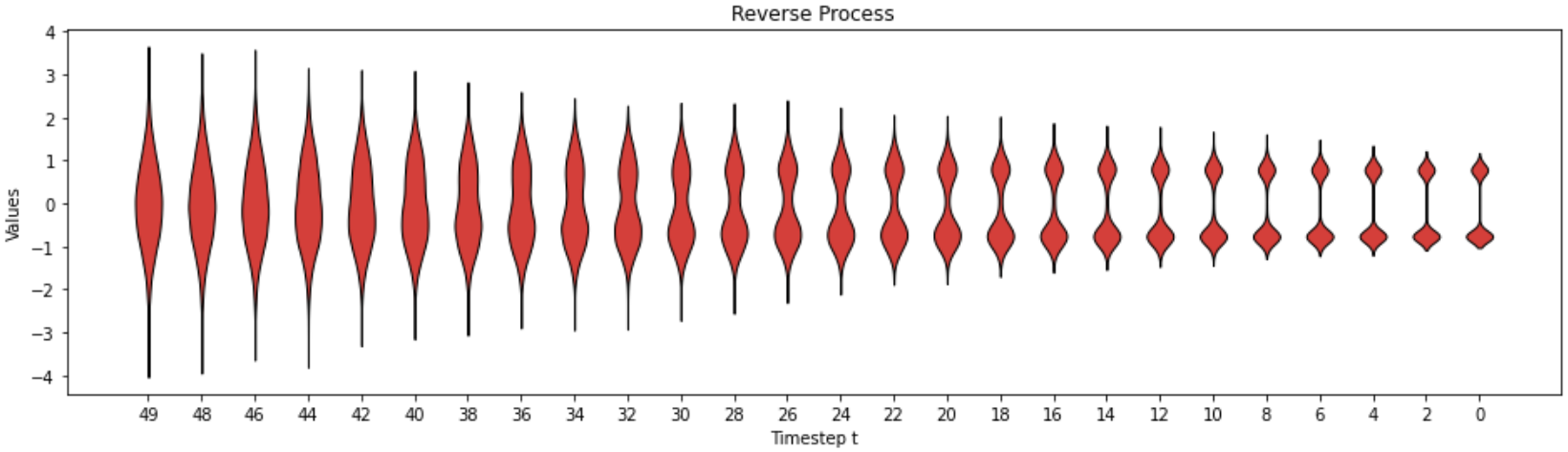
$$\min_\theta E_{t,x_0\epsilon}|\epsilon_t - \epsilon_\theta(x_t, t)|^2$$

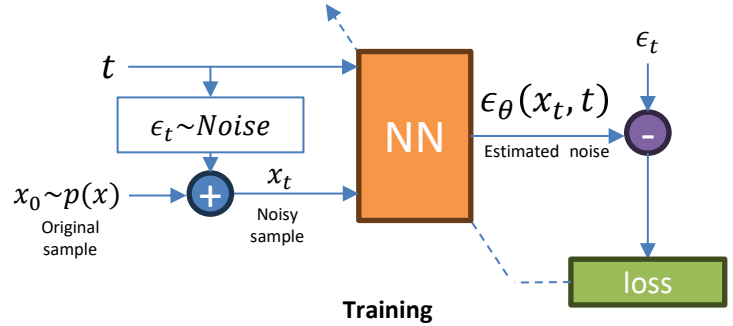
- **Generation:** Once the neural network is trained, we can generate data using:

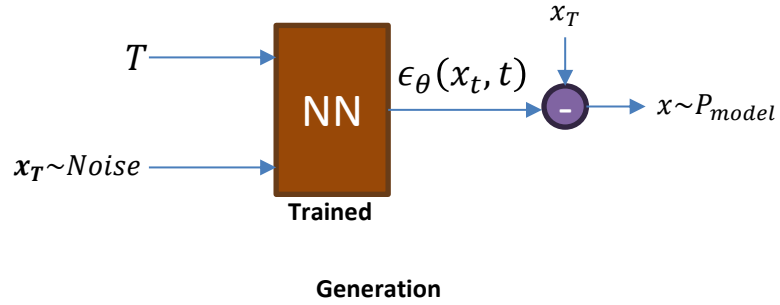
$$x = x_T - \epsilon_\theta(x_T, T) \text{ with } x_T \sim N(0,1)$$

- Can be improved by operating in a compressed or latent space: **Latent diffusion**



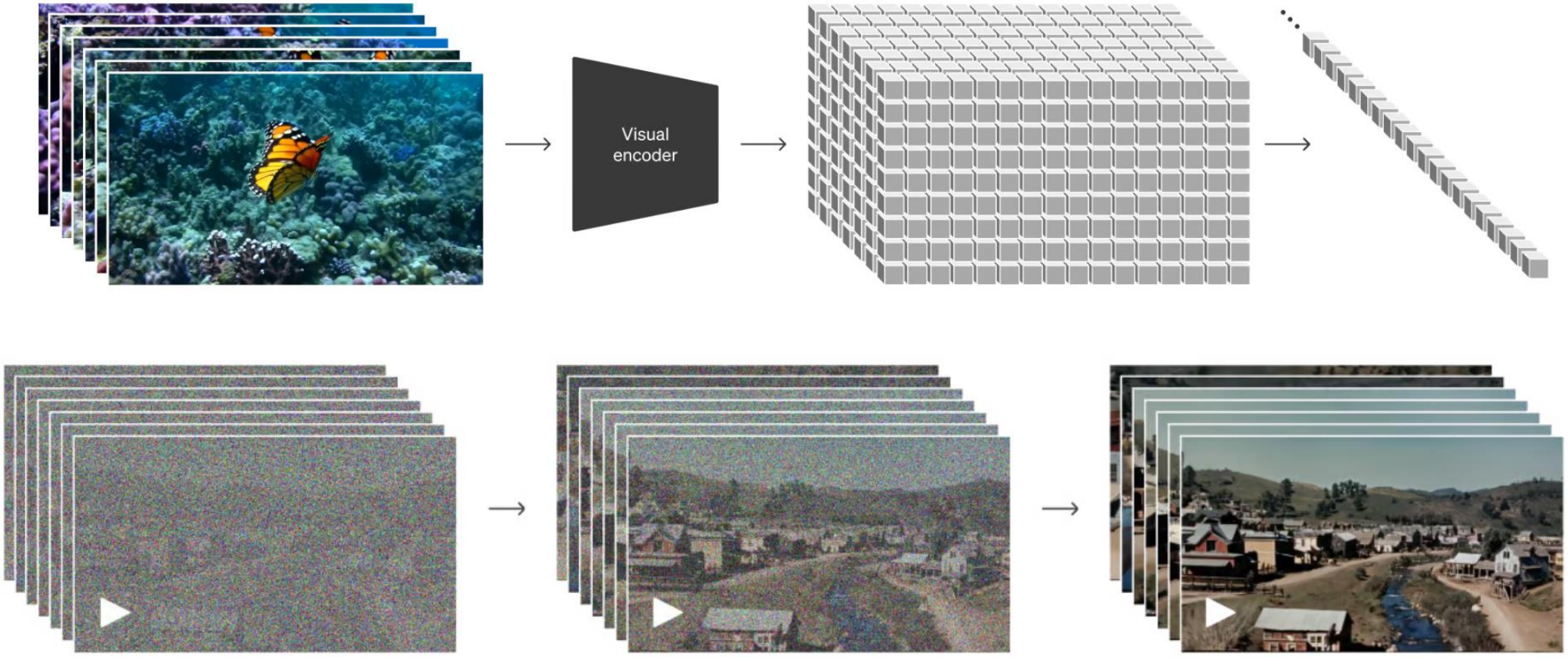






[Simplest Diffusion Tutorial: https://github.com/wgrgwrght/Simple-Diffusion/blob/main/SimpleDiffusion.ipynb](https://github.com/wgrgwrght/Simple-Diffusion/blob/main/SimpleDiffusion.ipynb)

SORA: Diffusion Transformer



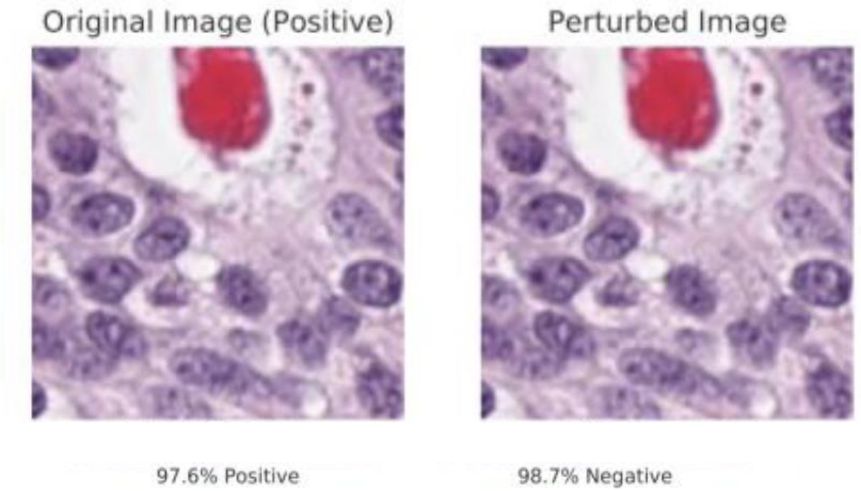
https://openai.com/research/video-generation-models-as-world-simulators

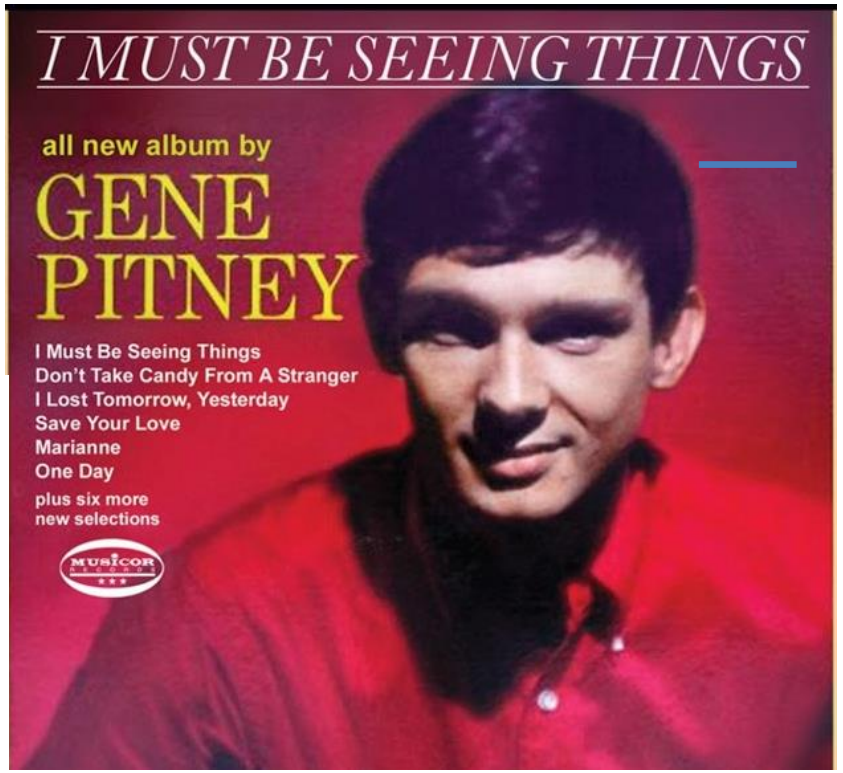
Application of Generative Modelling

CONCLUSIONS

Issues

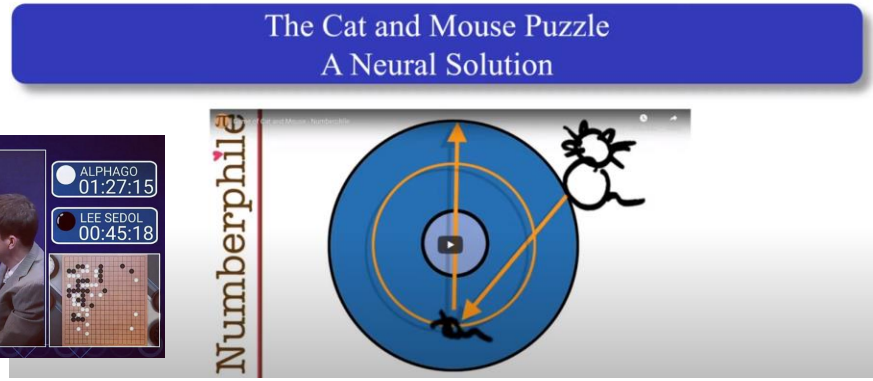
- Deep Neural Networks are Easily Fooled
  - https://arxiv.org/abs/1412.1897v4
- Failures of deep learning
  - https://arxiv.org/abs/1703.07950
- To understand deep learning we need to understand kernel learning
  - https://arxiv.org/abs/1802.01396
- Understanding deep learning requires rethinking generalization
- Steps toward deep kernel methods from infinite neural networks
  - https://arxiv.org/abs/1508.05133
- Do Deep Neural Networks Really Need to be Deep?
- One pixel attack for fooling deep neural networks
  - https://www.youtube.com/watch?v=SA4YEAWVpbk
  - https://github.com/Hyperparticle/one-pixel-attack-keras
- Adversarial Examples that Fool both Computer Vision and Time-Limited Humans
- Alchemy? https://www.youtube.com/watch?v=ORHFOnaEzPc
  - Ali Rahimi



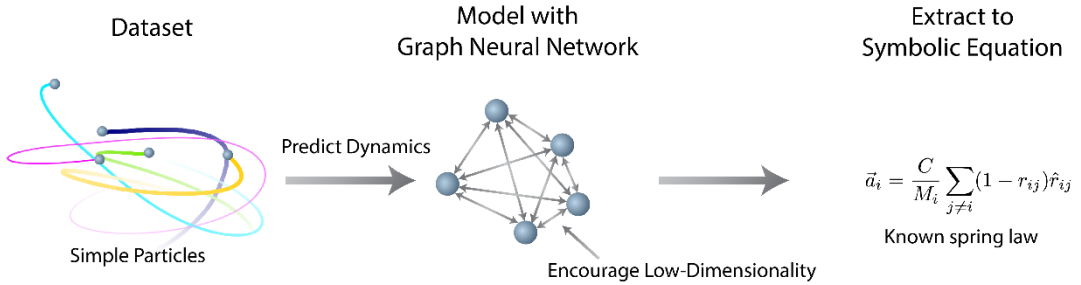


Other Topics

- Recurrent Neural Networks
- Reinforcement Learning
  - Learning from experience
  - Example: Learning to levitate or helping a mouse escape from a cat
  - https://github.com/foxtrotmike/RL-MagLev/blob/master/RL.ipynb
  - https://github.com/foxtrotmike/RL-MagLev/blob/master/cat_mouse.ipynb
- Learning Paradigms
  - Multi-task Learning
  - Multi-Label Learning
  - Self-Supervised Learning
    - Learn a task to learn a feature representation and adapt it to other tasks
    - Contrastive Learning
  - Zero Shot and Few Shot Learning
- Bayesian Neural Networks and Uncertainty Quantification (Conformal Prediction)
- Neural Ordinary Differential Equations (NODE)
  - https://github.com/foxtrotmike/NODE-Tutorial/blob/main/node_tutorial%20(2).ipynb
- Data Efficient Learning
- Symbolic Regression
- Learning to Learn
- Quantum ML
- Domain Generalization
- Robustness
- Building invariances into machine learning models
- Link between Causality, Symmetry, Invariance and Generalization
- Prompt Engineering, Retrieval Augmented Generation



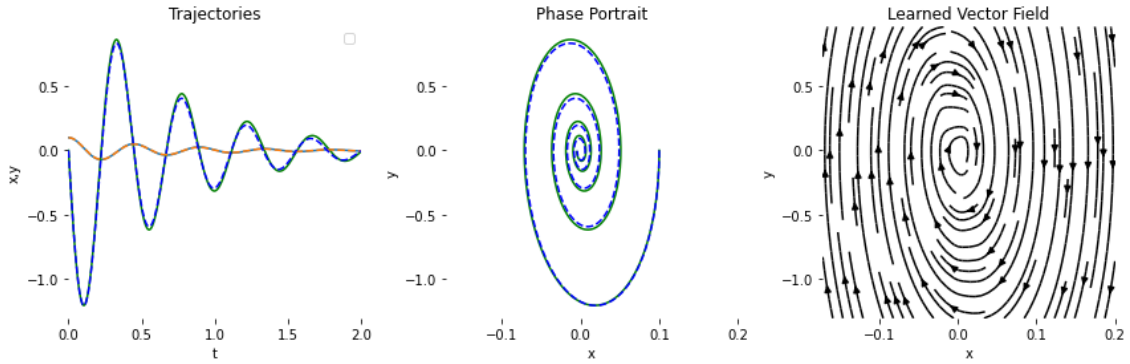
My RL Tutorial Video: https://youtu.be/N20h6vpR13Y



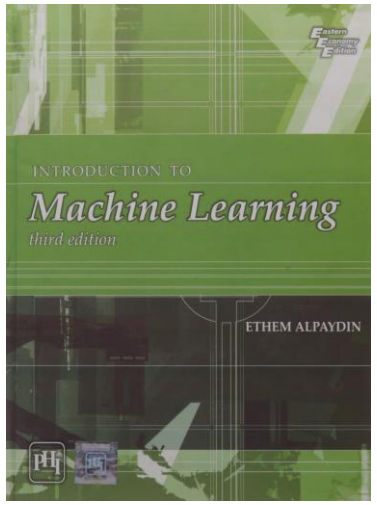
https://astroautomata.com/paper/symbolic-neural-nets/

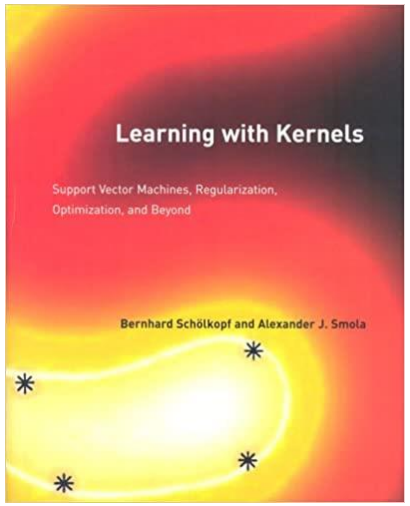
$$\dot{x_1}(t) = v(t) = x_2(t)$$

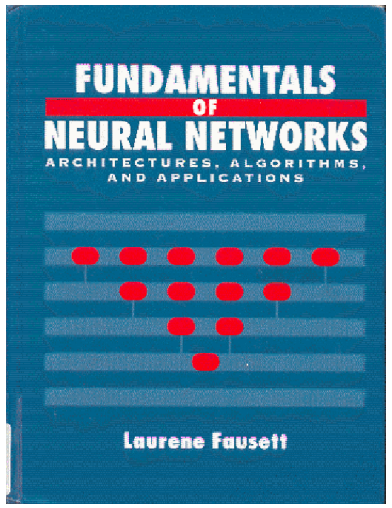
$$\dot{x_2}(t) = \dot{v}(t) = -\frac{g}{l}sin(x(t)) - \frac{k}{ml}v(t) = -\frac{g}{l}sin(x_1(t)) - \frac{k}{ml}x_2(t)$$

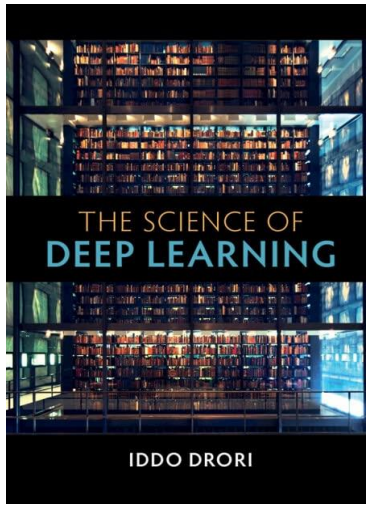


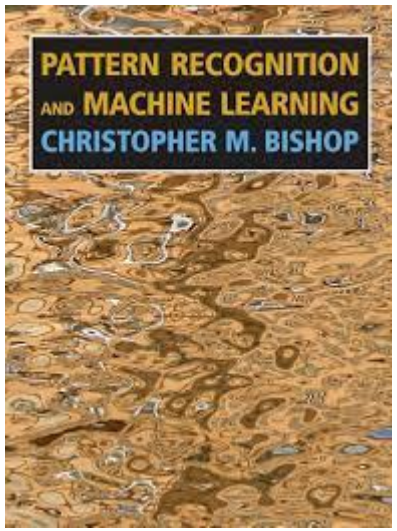
Books

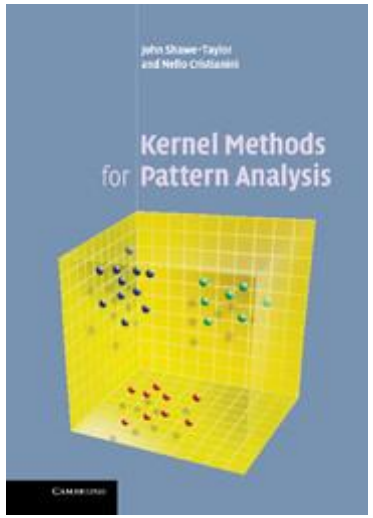


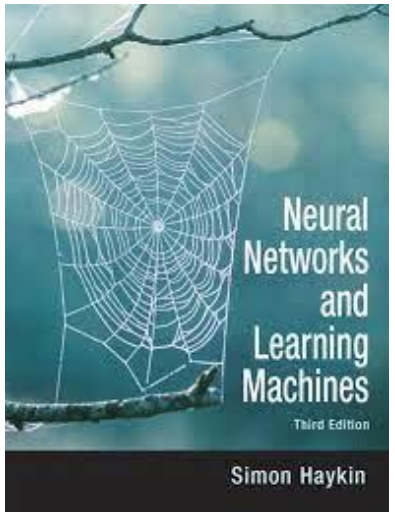


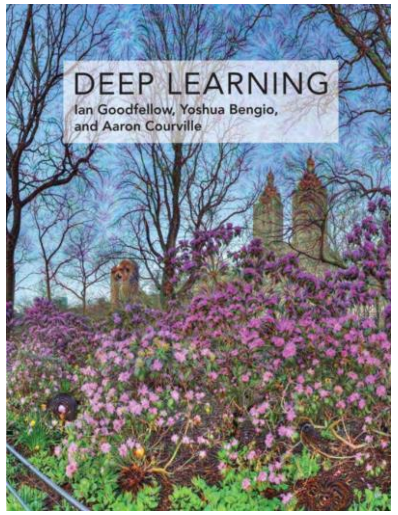


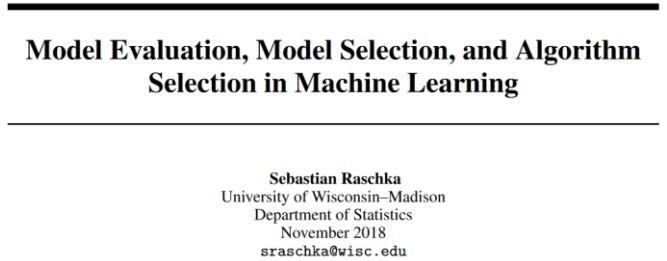











Model Evaluation, Model Selection, and Algorithm Selection in Machine Learning

Sebastian Raschka
University of Wisconsin–Madison
Department of Statistics
November 2018
sraschka@wisc.edu

Understanding Deep Learning by Simon J.D. Prince
https://udlbook.github.io/udlbook/

Foundations | SVMs and Kernels | Backpropagation and MLPs | Deep Learning