Neural networks

Content

- From Kernels to Neural Networks, Neural Tangent Kernel (NTK) and implicit regularization

- Generalization and robustness:

1. Explicit regularization, noise, dropout
2. SGD (Implicit regularization)
3. Learning rate
4. Weights initialization
5. Convolutional networks and priors on the weight sharing
6. Early stopping
7. Depth and compositionality
8. Data augmentation and symmetries
9. Adversarial training

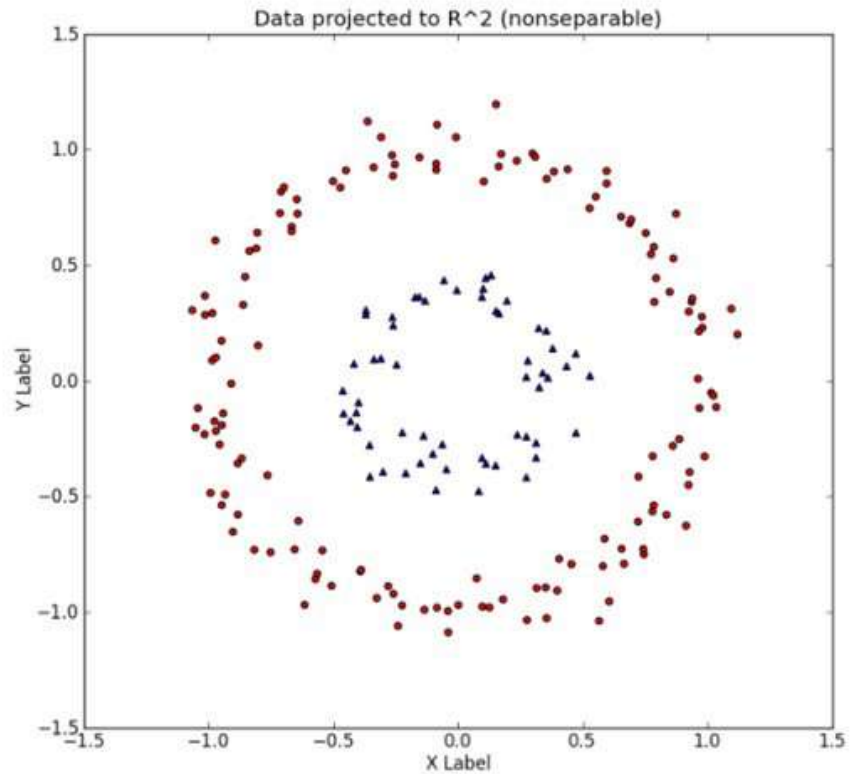
- Interpretability

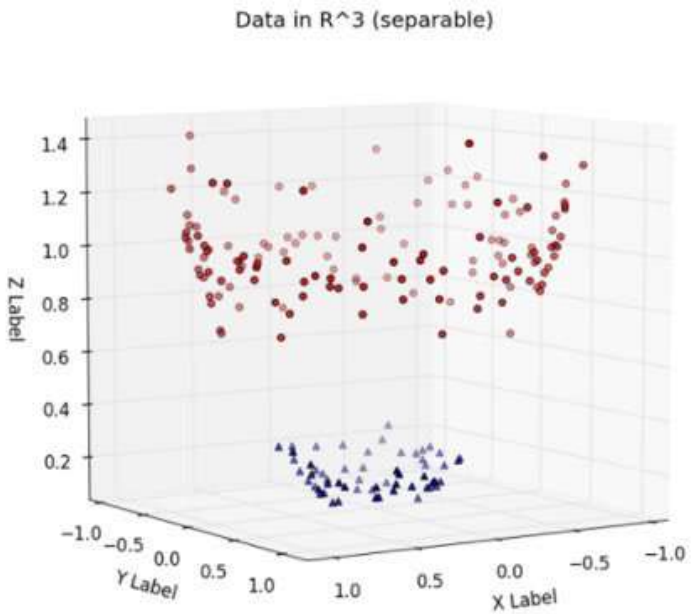
Class 1

- From kernels to ANNs
- Neural Tangent Kernel

Kernels: recap

$$x \quad \rightarrow \quad \phi(x) \qquad\qquad K_{i,j} = K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$$





Hypothesis space

- Linear separable

$$f(x) = w^T x$$

We learn the separator on data

- Non-linear separable

$$f(x) = w^T \phi(x)$$

We learn the separator on a priori hand-crafted features

Models

- Linear

$$f(x) = w^\top x.$$

- Features

$$f(x) = w^\top \Phi(x) = \sum_{j=1}^{p} w^j \varphi_j(x).$$

- Kernels

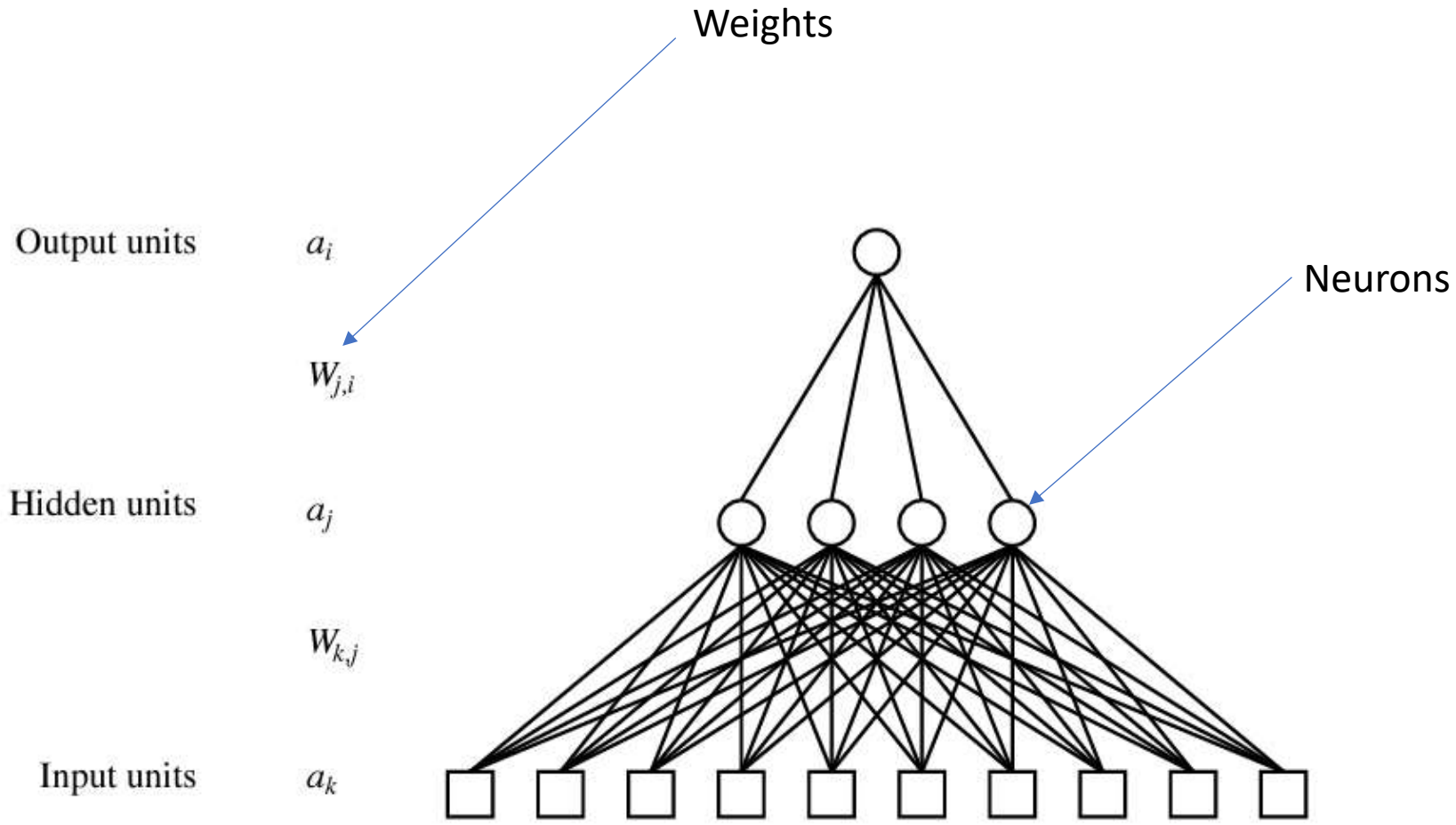
$$f(x) = \sum_{i=1}^{n} K(x, x_i)\, c_i.$$

- Can we learn the features together with the separator?
- Which features? We choose this parametrization:

$$f(x) = w^T \sigma(W^T x)$$

$$w \in \mathbb{R}^k, \quad W \in \mathbb{R}^{d \times k}, \quad \sigma : \mathbb{R} \to \mathbb{R}$$

Terminology



(1):Linear filtering

$$y = W^T x + b = \begin{bmatrix} w_1^T x + b_1 \\ w_2^T x + b_2 \\ \vdots \\ w_M^T x + b_M \end{bmatrix}$$

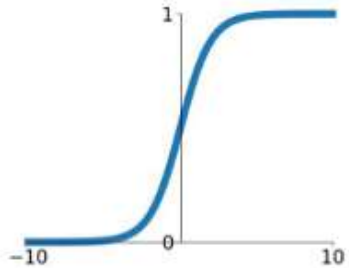
(2):Non-Linear pointwise filtering

$$y = \sigma(W^T x + b) = \begin{bmatrix} \sigma(w_1^T x + b_1) \\ \sigma(w_2^T x + b_2) \\ \vdots \\ \sigma(w_M^T x + b_M) \end{bmatrix}$$

Activation Functions

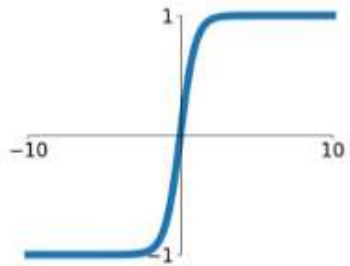
Sigmoid

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



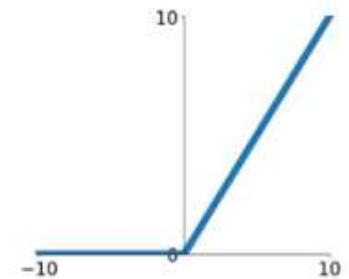
tanh

$$\tanh(x)$$



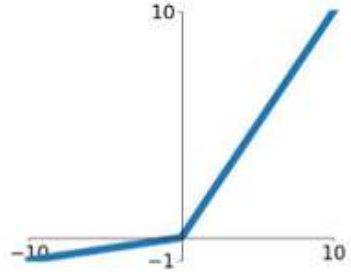
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

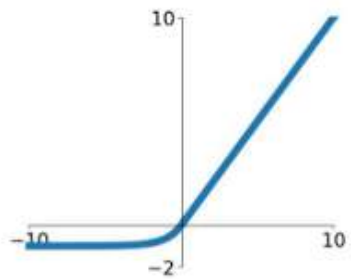


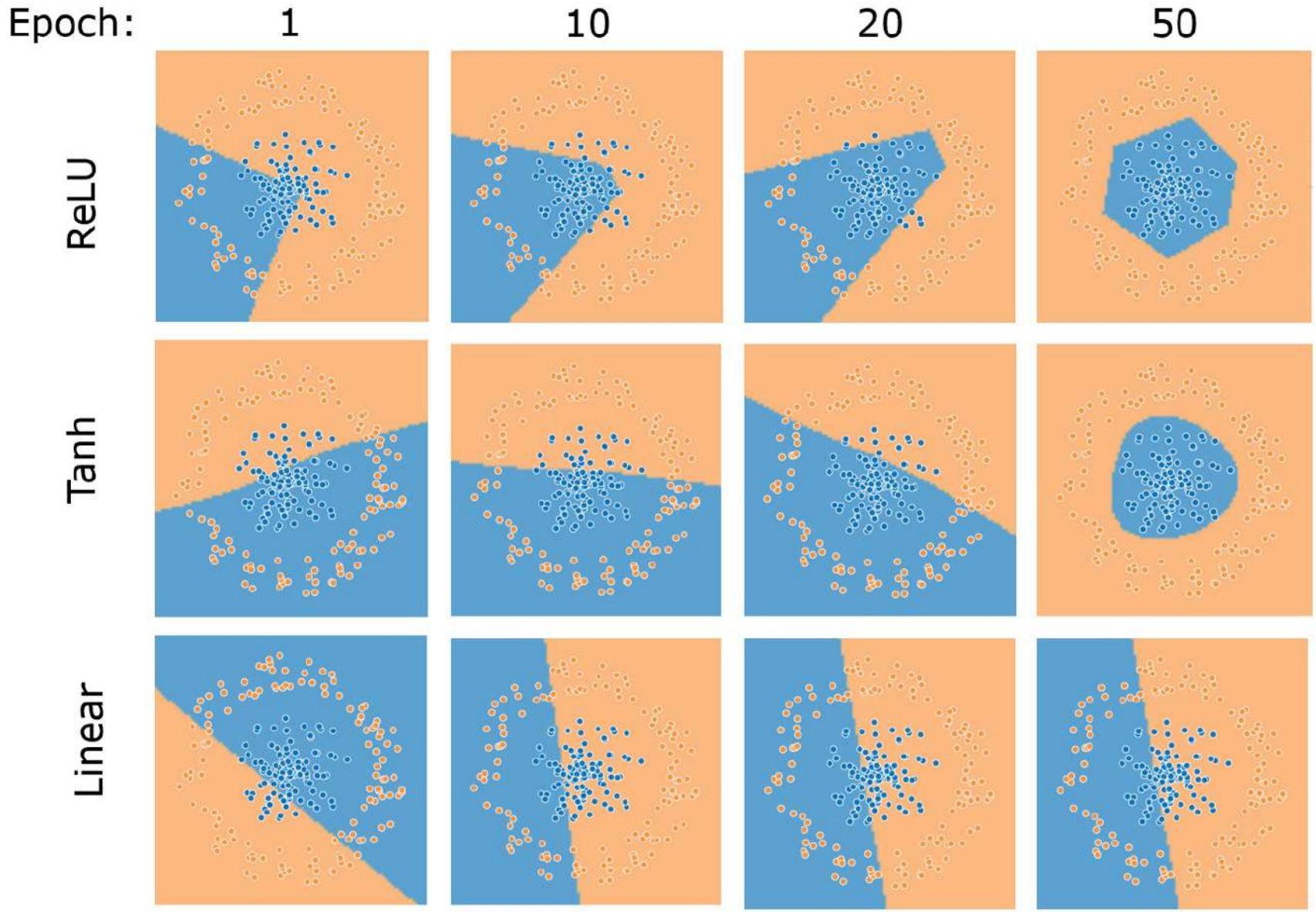
Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$




Epoch:
1
10
20
50
ReLU
Tanh
Linear

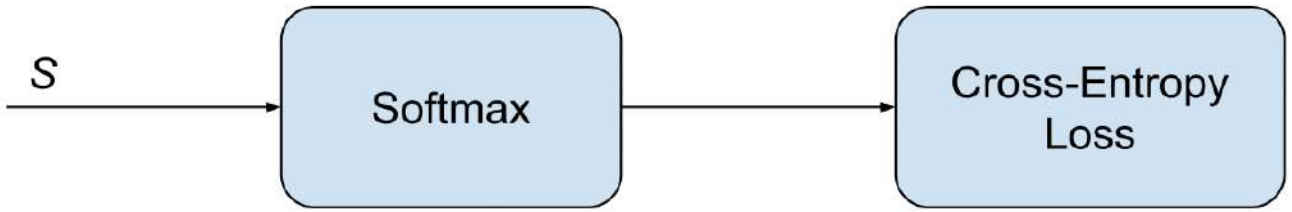
Loss functions examples: penalizing errors (regression)

$$\mathcal{L}(w,b) = \frac{1}{N}\sum_i (y_i - (w^T x_i + b))^2 = \frac{1}{N}\sum_i (y_i - \hat{y})^2$$

$$\mathcal{L}(w,W) = \frac{1}{N}\sum_{i=1}^{N} \left(y_i - w^T \sigma(W^T x_i)\right)^2$$

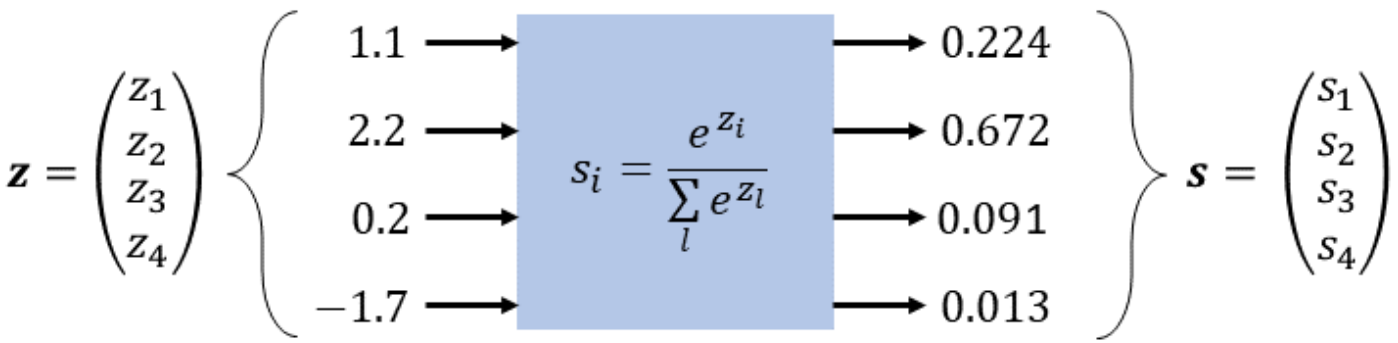
The Loss is minimized by gradient descent

Classification: cross entropy loss and softmax



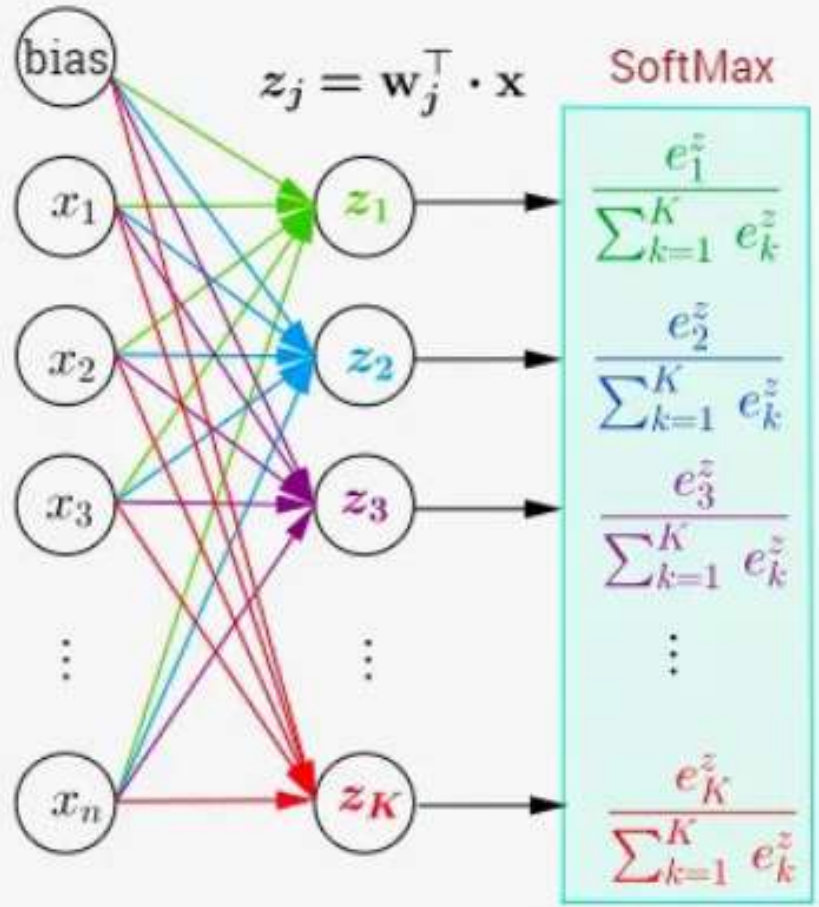
$$\hat{y}_i = \frac{e^{s_i}}{\sum_j e^{s_j}}$$

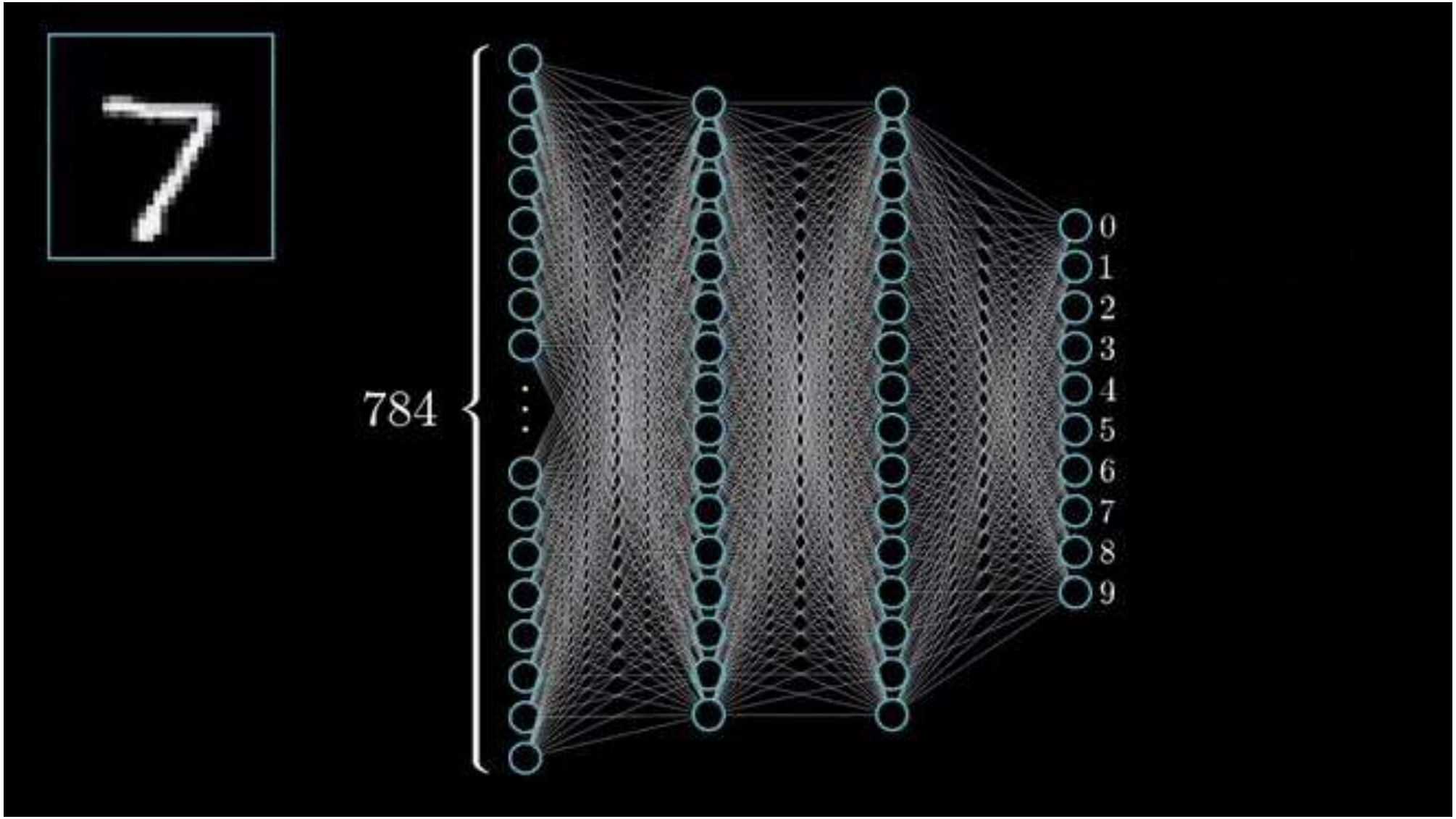
$$-\frac{1}{\text{output size}} \sum_{i=1}^{\text{output size}} y_i \cdot \log \hat{y}_i + (1 - y_i) \cdot \log (1 - \hat{y}_i)$$



Multi-Class Classification with NN and SoftMax Function

$$\mathbf{z} = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ \vdots \\ z_K \end{bmatrix} = \begin{bmatrix} \mathbf{w}_1^\top \\ \mathbf{w}_2^\top \\ \mathbf{w}_3^\top \\ \vdots \\ \mathbf{w}_K^\top \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix}$$

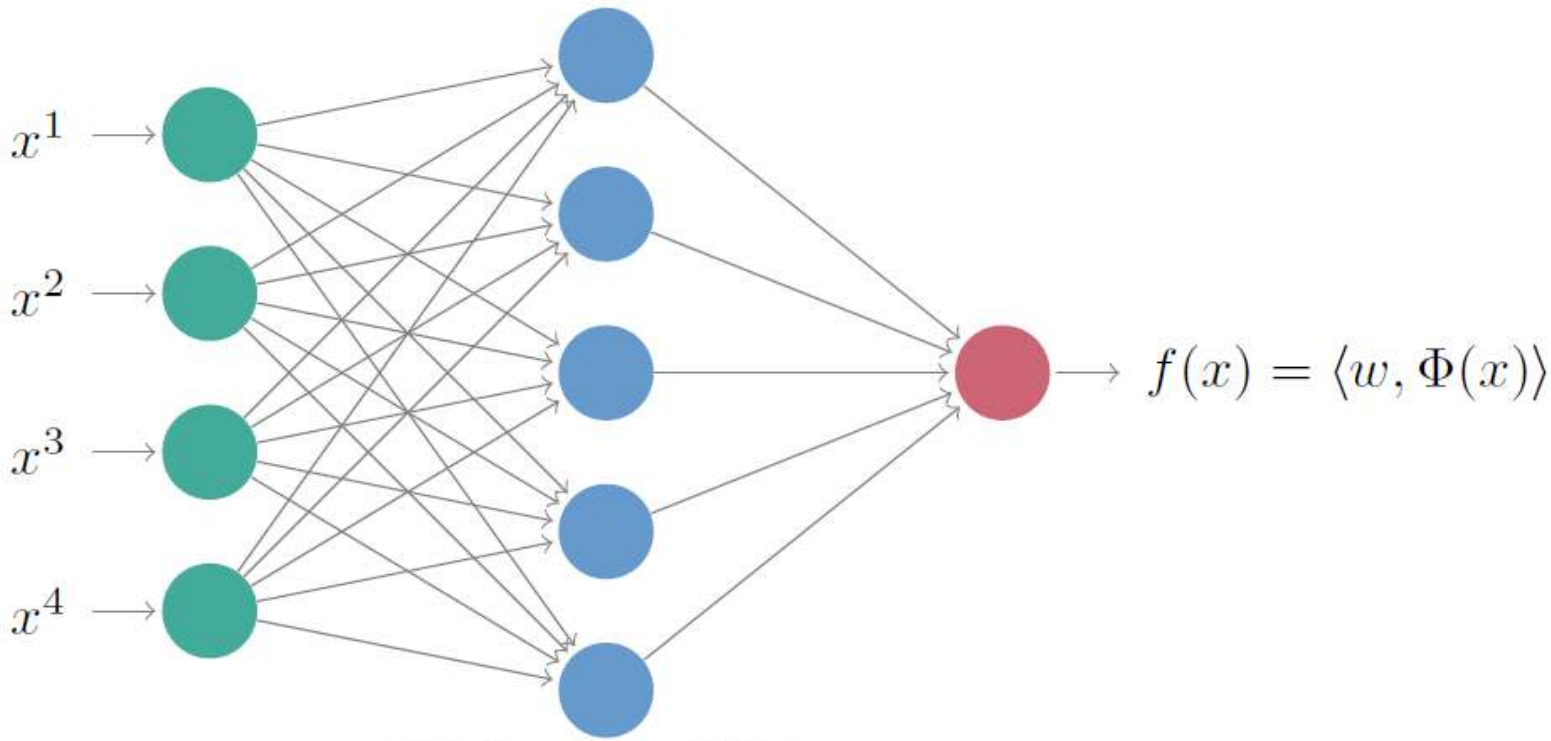




We can have multiple layers

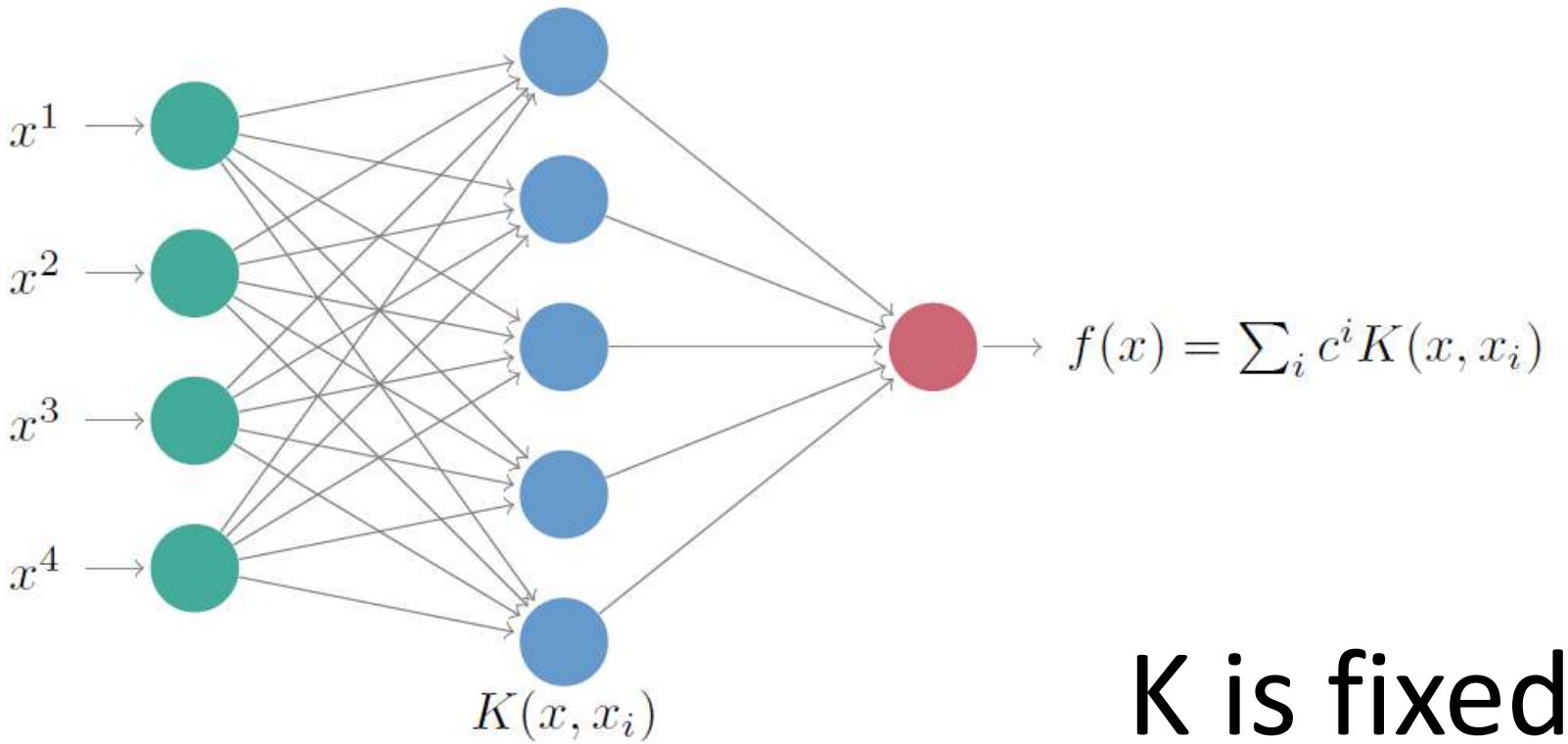
Shallow Neural Network Representations

l_0, input l_1, hidden layer l_L, output layer



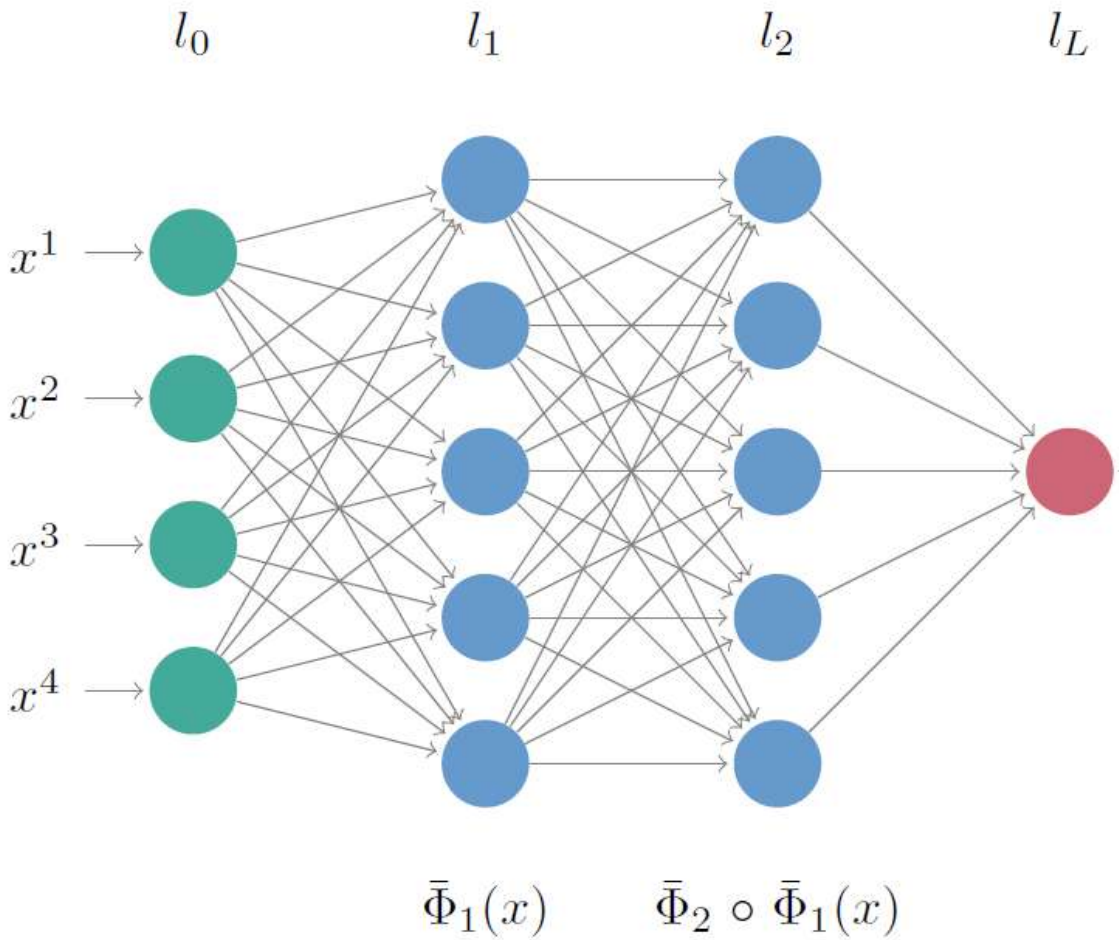
Kernel Machine Representations (e.g. RBF)

l_0, input l_1, hidden layer l_L, output layer



K is fixed!

Deep Neural Network Representations



$$f(x) = \langle w, \Phi_L(x) \rangle$$

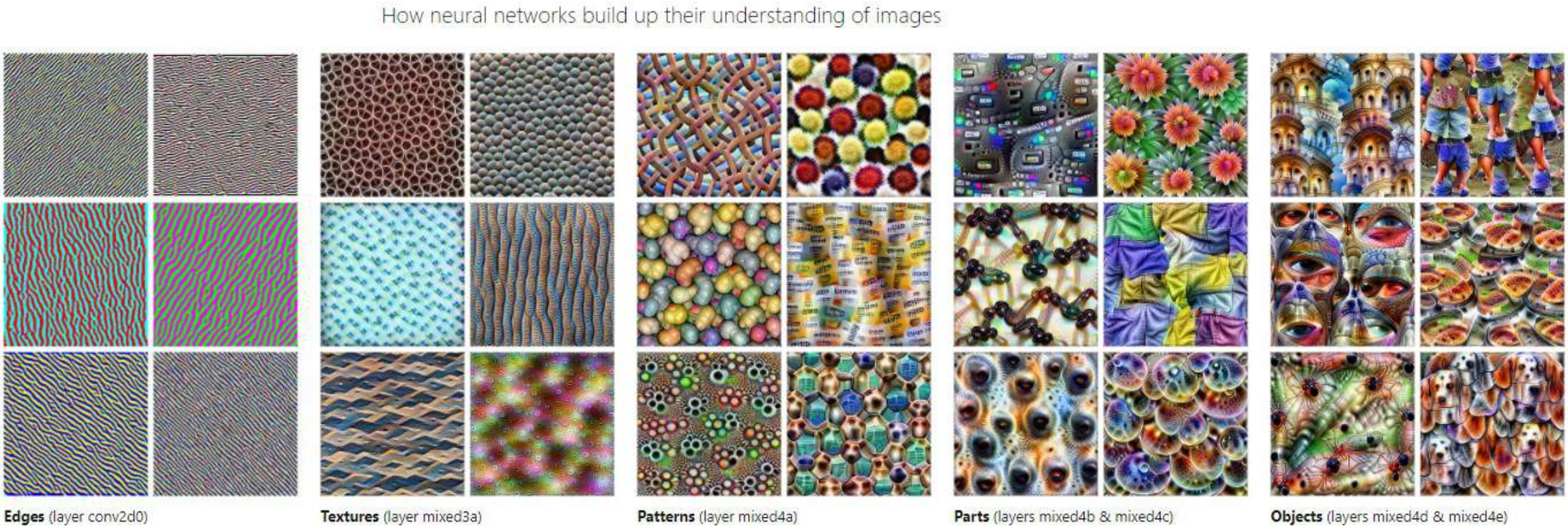
$$f(x) = w^T \sigma(W_L^T(\sigma(W_{L-1}^T \cdots \sigma(W_1^T x) \cdots)$$

$$w \in \mathbb{R}^{k_L}, \quad W \in \mathbb{R}^{d \times k_\ell}, \quad \sigma : \mathbb{R} \to \mathbb{R}$$

Learned features

$$f(x) = w^T \sigma(W_L^T(\sigma(W_{L-1}^T \cdots \sigma(W_1^T x) \cdots)$$

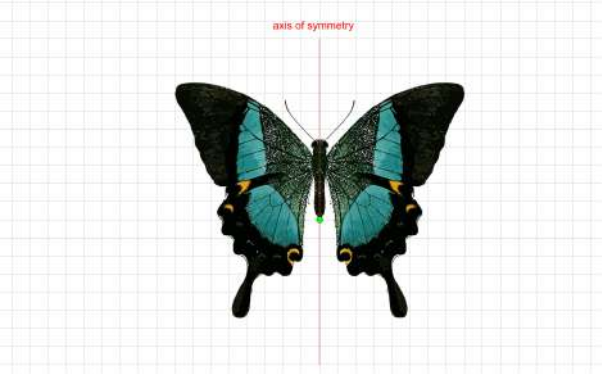
$$w \in \mathbb{R}^{k_L}, \quad W \in \mathbb{R}^{d \times k_\ell}, \quad \sigma : \mathbb{R} \to \mathbb{R}$$

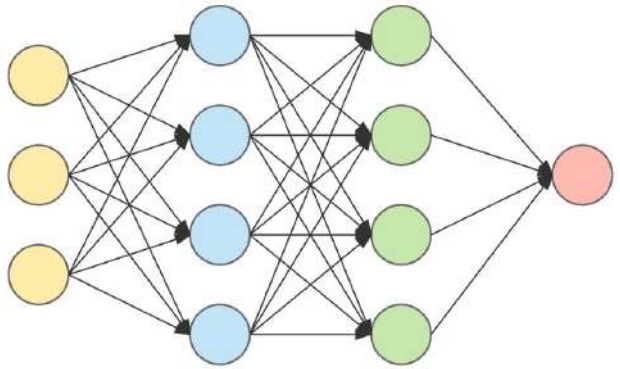


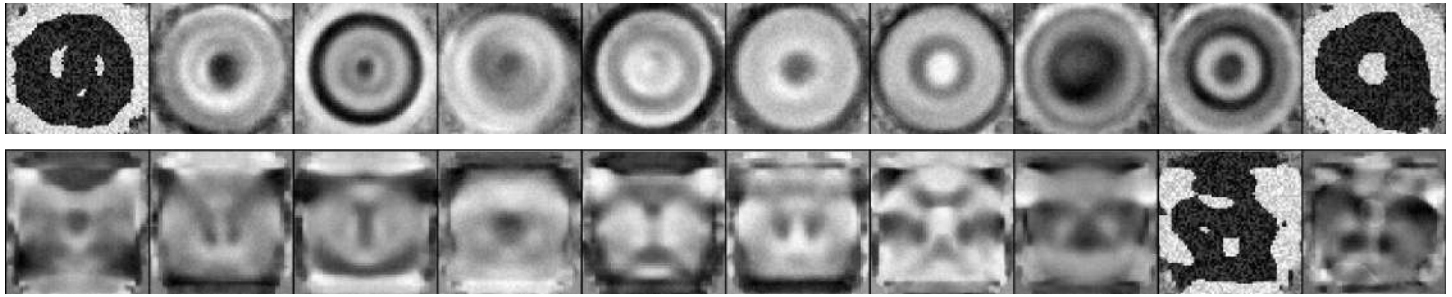
https://distill.pub/2017/feature-visualization/

https://towardsdatascience.com/visual-interpretability-for-convolutional-neural-networks-2453856210ce

Symmetries/regularities in data/data statistics are reflected in the learned weights of a network.







So

Shallow learning (kernels): $f(x) = \langle w, \Phi(x) \rangle, \quad x \mapsto \Phi(x)$

- Φ: fixed feature map

$$\min_w \frac{1}{n} \sum_{i=1}^{n} (y_i - \langle w, \Phi(x_i) \rangle)^2$$

Deep learning: $f(x) = \langle w, \Phi_L(x) \rangle, \quad x \mapsto \Phi_L(x)$

- Φ_L: compositional $\Phi_L = \bar{\Phi}_L \circ \cdots \circ \bar{\Phi}_1, \quad \bar{\Phi}_j = \sigma \circ W_j, j = 1, \ldots, L$
- Φ_L: learned jointly with w

$$\min_{w, (W_j)_j} \frac{1}{n} \sum_{i=1}^{n} (y_i - \langle w, \Phi_L(x_i) \rangle)^2$$

What is the link?

Link with Kernels: Neural Tangent Kernel

Neural Tangent Kernel:
Convergence and Generalization in Neural Networks

Arthur Jacot
École Polytechnique Fédérale de Lausanne
arthur.jacot@netopera.net

Franck Gabriel
Imperial College London and École Polytechnique Fédérale de Lausanne
franckrgabriel@gmail.com

Clément Hongler
École Polytechnique Fédérale de Lausanne
clement.hongler@gmail.com

Slides by Nolan Dey

https://arxiv.org/abs/1806.07572

Setup

- We have some neural network: $f(x, w)$
  - Input data: $x \in \mathbb{R}^{n \times d}$
  - Network parameters: $w \in \mathbb{R}^{p}$
- Loss: $L(f(x, w), y) = \frac{1}{2}(f(x, w) - y)^2$
- Optimize using full-batch gradient descent

What is the Neural Tangent Kernel (NTK)?

- Infinitely wide neural networks can fit any function

Universal approximation theorem

- Infinite width networks trained to convergence can be described by the NTK
- NTK describes training dynamics: $\frac{df(x,w)}{dt} = -NTK(w_0)(f(x,w) - y)$

Is the kernel associated to the network features

Universal approximation theorem

Any **continuous function** defined in the **n-dimensional unit hypercube** may be approximated by a **finite sum** of the type:

$$\sum_{j=1}^{N} v_j \, \varphi \left(\vec{\omega}^{(j)} \cdot \vec{x} + b_j \right),$$

wherein $v_j, b_j \in \mathbb{R}$, $\vec{\omega}^{(j)} \in \mathbb{R}^n$, and φ is a **continuous discriminatory function**.

How big is N?

Recap on gradient

Matrix Calculus

Let $y, x \in \mathbb{R}$ be scalars, $\mathbf{y} \in \mathbb{R}^M$ and $\mathbf{x} \in \mathbb{R}^P$ be vectors, and $\mathbf{Y} \in \mathbb{R}^{M \times N}$ and $\mathbf{X} \in \mathbb{R}^{P \times Q}$ be matrices

Types of Derivatives (Denominator ↓ / Numerator →)	**scalar**	**vector**	**matrix**
scalar	$\frac{\partial y}{\partial x}$	$\frac{\partial \mathbf{y}}{\partial x}$	$\frac{\partial \mathbf{Y}}{\partial x}$
vector	$\frac{\partial y}{\partial \mathbf{x}}$	$\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$	$\frac{\partial \mathbf{Y}}{\partial \mathbf{x}}$
matrix	$\frac{\partial y}{\partial \mathbf{X}}$	$\frac{\partial \mathbf{y}}{\partial \mathbf{X}}$	$\frac{\partial \mathbf{Y}}{\partial \mathbf{X}}$

Types of Derivatives	**scalar**
scalar	$\frac{\partial y}{\partial x} = \left[\frac{\partial y}{\partial x}\right]$
vector	$\frac{\partial y}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y}{\partial x_1} \\ \frac{\partial y}{\partial x_2} \\ \vdots \\ \frac{\partial y}{\partial x_P} \end{bmatrix}$
matrix	$\frac{\partial y}{\partial \mathbf{X}} = \begin{bmatrix} \frac{\partial y}{\partial X_{11}} & \frac{\partial y}{\partial X_{12}} & \cdots & \frac{\partial y}{\partial X_{1Q}} \\ \frac{\partial y}{\partial X_{21}} & \frac{\partial y}{\partial X_{22}} & \cdots & \frac{\partial y}{\partial X_{2Q}} \\ \vdots & & & \vdots \\ \frac{\partial y}{\partial X_{P1}} & \frac{\partial y}{\partial X_{P2}} & \cdots & \frac{\partial y}{\partial X_{PQ}} \end{bmatrix}$

Types of Derivatives	**scalar**	**vector**
scalar	$\frac{\partial y}{\partial x} = \left[\frac{\partial y}{\partial x}\right]$	$\frac{\partial \mathbf{y}}{\partial x} = \left[\frac{\partial y_1}{\partial x} \quad \frac{\partial y_2}{\partial x} \quad \cdots \quad \frac{\partial y_N}{\partial x}\right]$
vector	$\frac{\partial y}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y}{\partial x_1} \\ \frac{\partial y}{\partial x_2} \\ \vdots \\ \frac{\partial y}{\partial x_P} \end{bmatrix}$	$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_2}{\partial x_1} & \cdots & \frac{\partial y_N}{\partial x_1} \\ \frac{\partial y_1}{\partial x_2} & \frac{\partial y_2}{\partial x_2} & \cdots & \frac{\partial y_N}{\partial x_2} \\ \vdots & & & \\ \frac{\partial y_1}{\partial x_P} & \frac{\partial y_2}{\partial x_P} & \cdots & \frac{\partial y_N}{\partial x_P} \end{bmatrix}$

Vector Gradient

$$\frac{\partial}{\partial w_j}(w^T x) = \frac{\partial}{\partial w_j}\sum_k w_k x_k = x_j$$

$$\nabla_w(w^T x) = x$$

The derivative of a vector w.r.t. a vector is a matrix.

$$A \in \mathbb{R}^{N \times D}, \omega \in \mathbb{R}^{D \times 1}, A\omega \in \mathbb{R}^{N \times 1} \Rightarrow \frac{\partial(A\omega)}{\partial \omega} \in \mathbb{R}^{N \times D}$$

We can denote this matrix by D, its general element is component j of $A\omega$ derived by component p of ω.

$$D_{jp} = \frac{\partial [A\omega]_j}{\partial \omega_p} = \frac{\partial}{\partial \omega_p} \sum_{i=1}^{D} A_{ji}\omega_i = \sum_{i=1}^{D} A_{ji} \frac{\partial}{\partial \omega_p} \omega_i = \sum_{i=1}^{D} A_{ji}\delta_{ip} = A_{jp}$$

Hence:

$$\frac{\partial(A\omega)}{\partial \omega} = A$$

Other examples

$$\frac{\partial \mathbf{x}^T\mathbf{a}}{\partial \mathbf{x}} = \frac{\partial \mathbf{a}^T\mathbf{x}}{\partial \mathbf{x}} = \mathbf{a}$$

$$\frac{\partial \mathbf{a}^T\mathbf{X}\mathbf{b}}{\partial \mathbf{X}} = \mathbf{a}\mathbf{b}^T$$

$$\frac{\partial \mathbf{a}^T\mathbf{X}^T\mathbf{b}}{\partial \mathbf{X}} = \mathbf{b}\mathbf{a}^T$$

$$\frac{\partial \mathbf{a}^T\mathbf{X}\mathbf{a}}{\partial \mathbf{X}} = \frac{\partial \mathbf{a}^T\mathbf{X}^T\mathbf{a}}{\partial \mathbf{X}} = \mathbf{a}\mathbf{a}^T$$

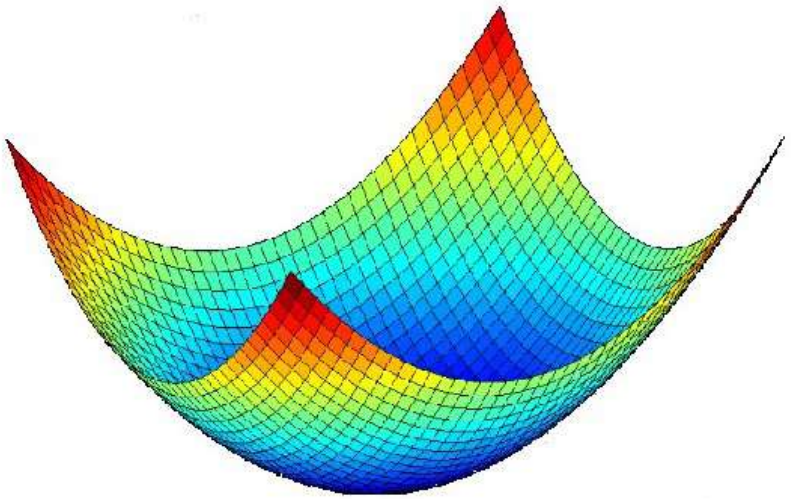
The Matrix Cookbook

[http://matrixcookbook.com]

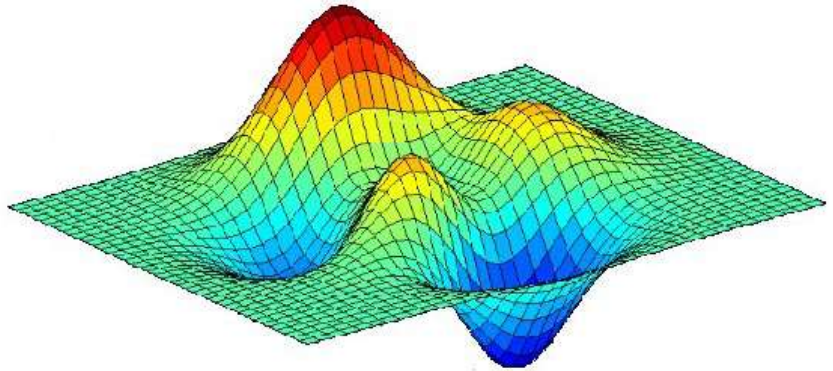
Kaare Brandt Petersen
Michael Syskind Pedersen

Recap on gradient descent

$$\mathcal{L}(w, W) = \frac{1}{N} \sum_{i=1}^{N} \left(y_i - w^T \sigma(W^T x) \right)^2$$

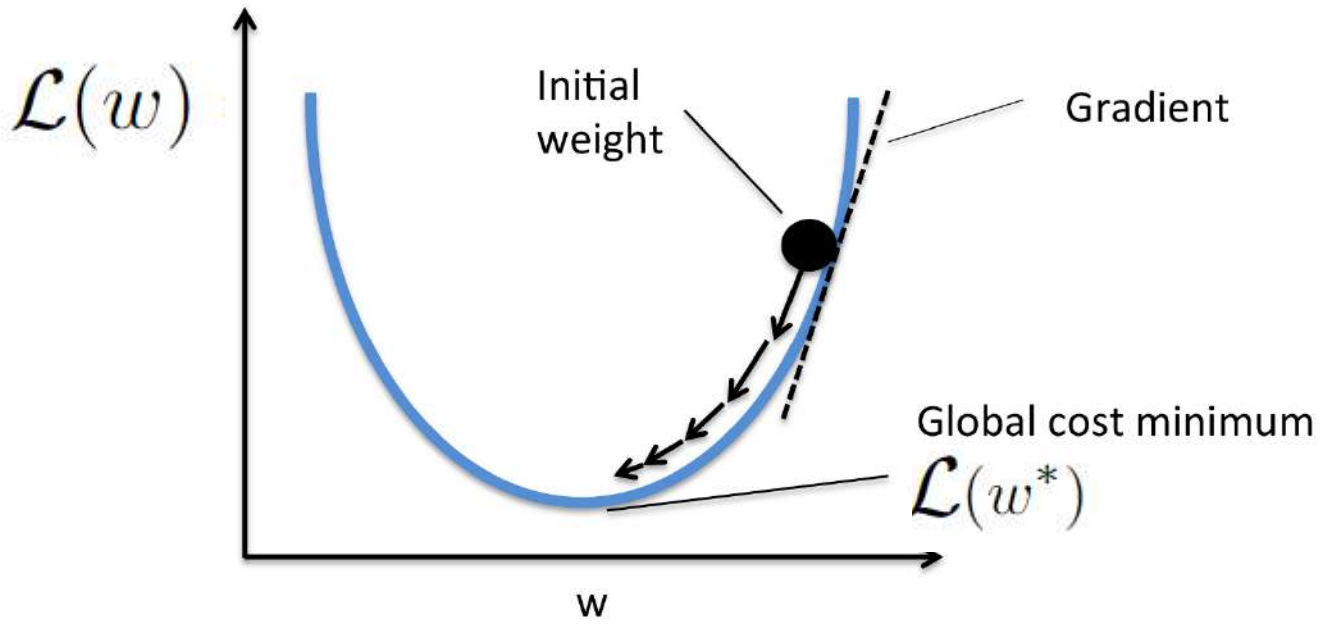


convex function



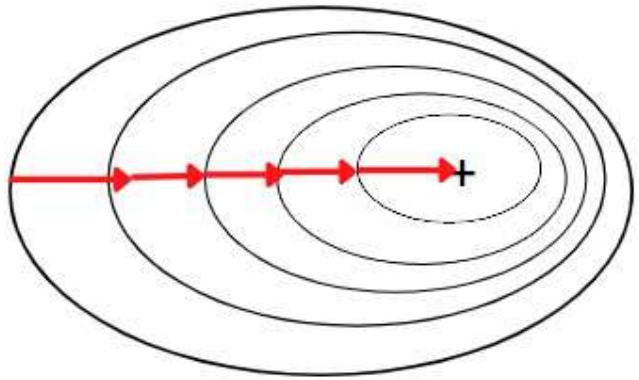
non-convex function

Finding minima with gradient descent

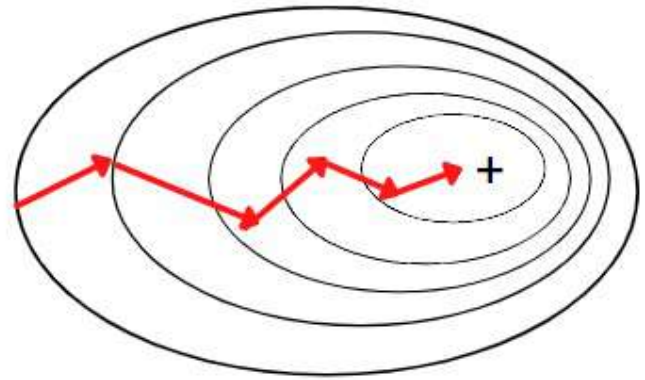


$$w_{t+1} = w_t - \gamma \frac{d\mathcal{L}(w)}{dw}$$

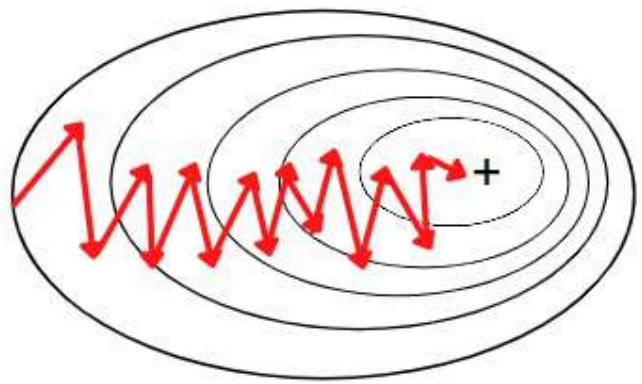
Batch Gradient Descent



Mini-Batch Gradient Descent



Stochastic Gradient Descent



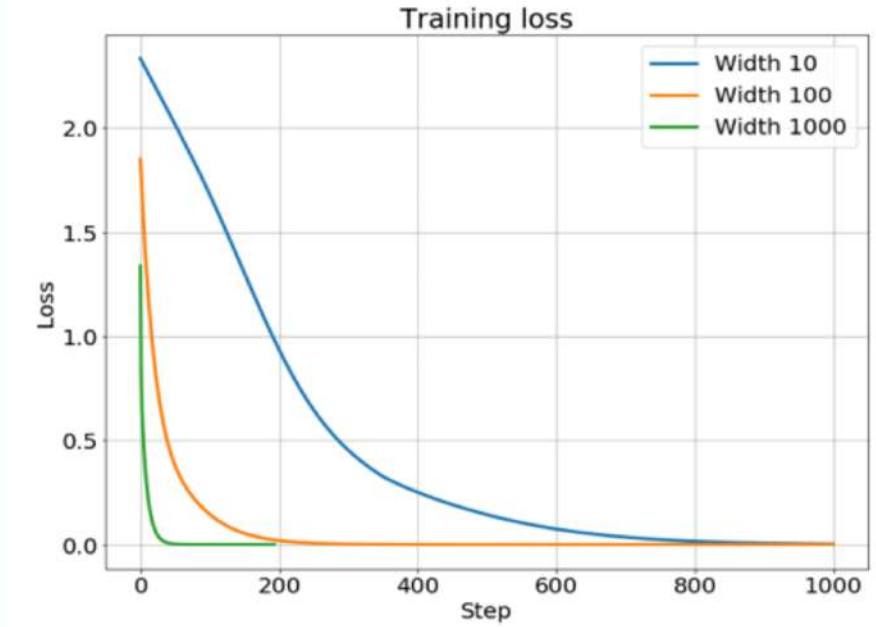
Taylor expansion of neural network

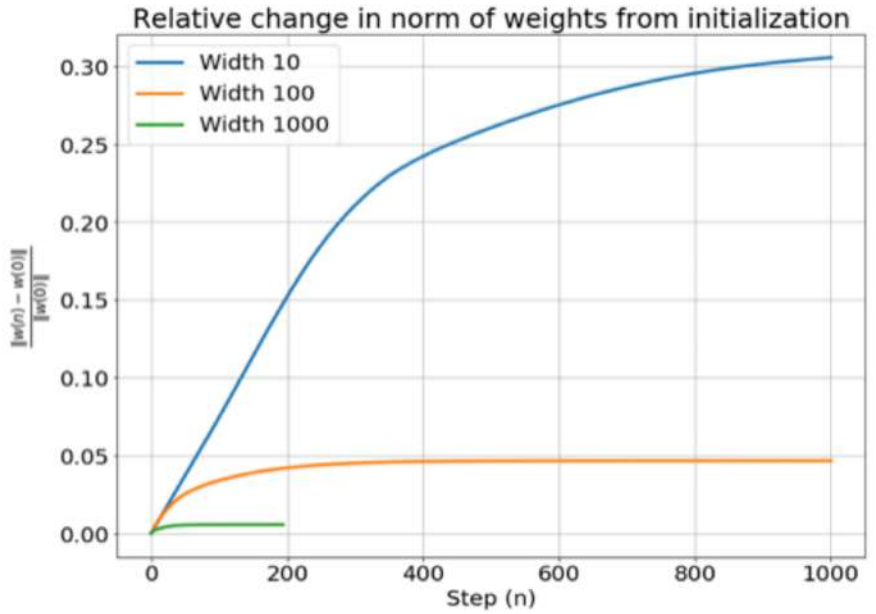
- Taylor expansion of neural network with respect to initialization weights w_0
  - $f(x, w) \approx f(x, w_0) + \nabla_w f(x, w_0)^T (w - w_0)$
- This Taylor expansion is only accurate when weights remain close to initialization

Small learning!

When is Taylor expansion accurate?

- Taylor expansion is only accurate when the weights don't change much during training
  - Weights don't change much when network is sufficiently wide
  - Lazy training = weights don't need to change





This is true when the width of the network is very large. Why?

Gradient Flow

- Gradient descent: $w_{t+1} = w_t - \eta \nabla_w L(w_t)$
- Rewrite as: $\frac{w_{t+1} - w_t}{\eta} = -\nabla_w L(w_t)$
  - Resembles finite difference^
- Take infinitesimally small learning rate η
- Gradient flow! —> $\frac{dw(t)}{dt} = -\nabla_w L(w(t))$

$$
\begin{aligned}
\frac{dw(t)}{dt} &= -\nabla_w \mathcal{L}(w(t), x) \\
&= \nabla_w \frac{1}{N} \sum_i (f(w(t), x_i) - y_i)^2 \\
&= -\frac{2}{N} \sum_i (f(w(t), x_i) - y_i) \nabla_w f(w(t), x_i)
\end{aligned}
$$

This is not static!!!!

$$
\begin{aligned}
\frac{df(w(t),x)}{dt} &= \nabla_w f(w(t),x)\frac{dw(t)}{dt} \\
&= \nabla_w^T f(w(t),x)\frac{-2}{N}\sum_i (f(w(t),x_i)-y_i)\nabla_w f(w(t),x_i) \\
&= -\frac{2}{N}\sum_i (f(w(t),x_i)-y_i)\nabla_w^T f(w(t),x)\nabla_w f(w(t),x_i) \\
&= -\frac{2}{N}\sum_i (f(w(t),x_i)-y_i)K(x,x_i)
\end{aligned}
$$

Feature $\Phi(x_i)$

Supposing quasi-static

If the width is infinite the change in w is very small and K is the kernel associated to the dynamic of the output

Calculate the kernel for

$$f(w,x) = \sum_i v_i \sigma(w_i^T x)$$

What does this mean?

- NTK is a useful tool for studying the dynamics of infinitely wide neural networks
- Successful nets in practice DO NOT OPERATE IN THE NTK REGIME
  - Finite nets still outperform their exactly computed infinite width counterparts
  - SGD vs full-batch gradient descent

This analysis is useful to show:

- Kernel models are not good enough to capture NNs
- Implicit Bias/Regularization (next)

Class 2

- NTK implicit bias, Double descent
- Hypothesis space, generalization
- L2 regularization, Noise and L2 regularization
- Dropout
- Data Normalization

Implicit Bias/regularization

- Explicit regularization imposes an explicit penalty on the parameters of the model (typically some measure of complexity or sensitivity)

- Implicit regularization occurs when the dynamics of training lead to certain minima rather than others

- Algorithm
- Initialization
- Architecture

Example of explicit regularization: ridge regression

$$\arg\min_{w} \|y - X^T w\|_2^2 + \lambda \|w\|_2^2$$

- Reconstruction term $\|y - X^T w\|_2^2$
- Complexity control $\lambda \|w\|_2^2$

Implicit regularization:NTK solution

$$
\begin{aligned}
\frac{df(\theta)}{dt} &= -\eta \nabla_\theta f(\theta)^\top \nabla_\theta f(\theta) \nabla_f \mathcal{L} \\
&= -\eta \nabla_\theta f(\theta)^\top \nabla_\theta f(\theta) \nabla_f \mathcal{L} \\
&= -\eta K(\theta) \nabla_f \mathcal{L} \\
&= -\eta K_\infty \nabla_f \mathcal{L}
\end{aligned}
$$

NTK solution

$$\frac{df(\theta)}{dt} = -\eta K_\infty (f(\theta) - \mathcal{Y})$$

$$\frac{dg(\theta)}{dt} = -\eta K_\infty g(\theta) \qquad ; \text{let } g(\theta) = f(\theta) - \mathcal{Y}$$

$$\int \frac{dg(\theta)}{g(\theta)} = -\eta \int K_\infty dt$$

$$g(\theta) = Ce^{-\eta K_\infty t}$$

When $t = 0$, we have $C = f(\theta(0)) - \mathcal{Y}$ and therefore,

$$f(\theta) = (f(\theta(0)) - \mathcal{Y})e^{-\eta K_\infty t} + \mathcal{Y} = f(\theta(0))e^{-K_\infty t} + (I - e^{-\eta K_\infty t})\mathcal{Y}$$

The expression dictates how the error changes in time!

Implicit regularization

Solving for f and using the fact that in the large-width limit (1-layer shallow network with many hidden units) the kernel is approximately stationary:

$$f(X, W(t)) \approx (I - e^{-Kt}) \cdot Y.$$

Now, since the kernel K is positive semi-definite, we can take its spectral decomposition $K = Q\Lambda Q^T$ where Q is an orthogonal matrix whose i-th column is the eigenvector q_i of K and Λ is a diagonal matrix whose diagonal entries λ_i are the corresponding eigenvalues. Since $e^{-Kt} = Qe^{-\Lambda t}Q^T$ we have

$$Q^T \left(f(X, W(t)) - Y\right) = -e^{-\Lambda t} Q^T Y,$$

i.e.,

$$\begin{bmatrix} q_1^T \\ q_2^T \\ \vdots \\ q_N^T \end{bmatrix} \left(f(X, W(t)) - Y\right) = \begin{bmatrix} e^{-\lambda_1 t} & & & \\ & e^{-\lambda_2 t} & \cdots & \\ & & \ddots & \\ & & & e^{-\lambda_N t} \end{bmatrix} \begin{bmatrix} q_1^T \\ q_2^T \\ \vdots \\ q_N^T \end{bmatrix} Y.$$

Implicit regularization

The above equation shows that the convergence rate of $q_i^T(f(X, W(t)) - Y)$, *the error in the* X *predictions*, is determined by the i-th eigenvalue λ_i. Moreover, we can decompose the training error into the eigenspace of the NTK as

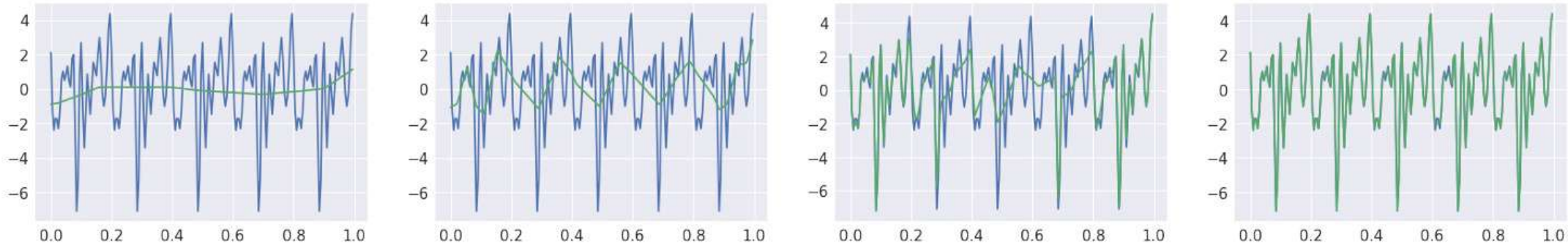
$$f(X, W(t)) - Y = \sum_{i=1}^{N} (f(X, W(t)) - Y, q_i) q_i$$

$$= \sum_{i=1}^{N} q_i^T (f(X, W(t)) - Y) q_i$$

$$= \sum_{i=1}^{N} \left(e^{-\lambda_i t} q_i^T Y\right) q_i.$$

Clearly, the network is biased to first learn the target function along the eigen-directions of the neural tangent kernel with larger eigenvalues, and then the rest components corresponding to smaller eigenvalues.

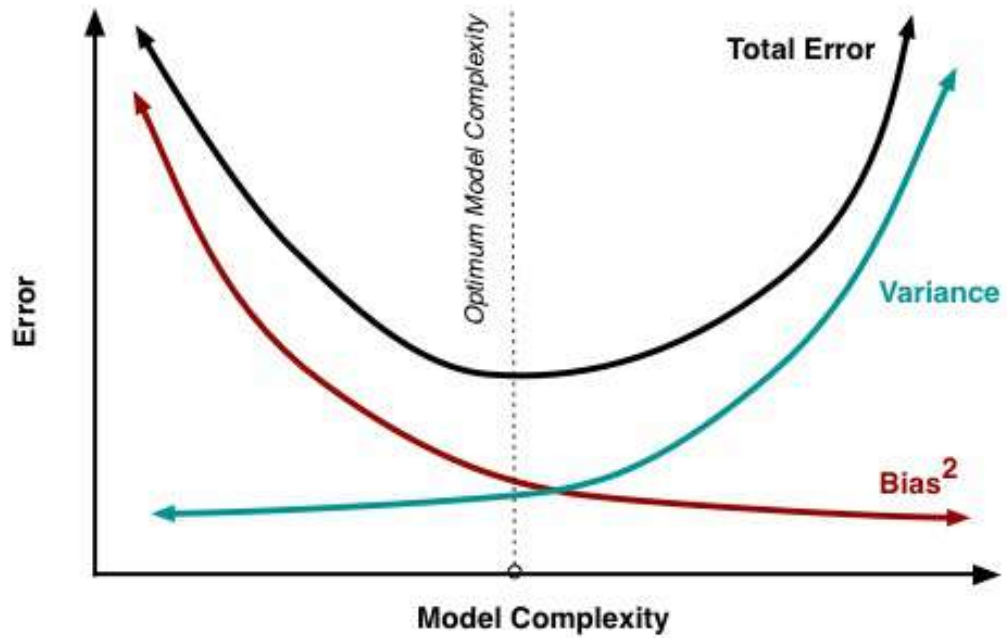
An example

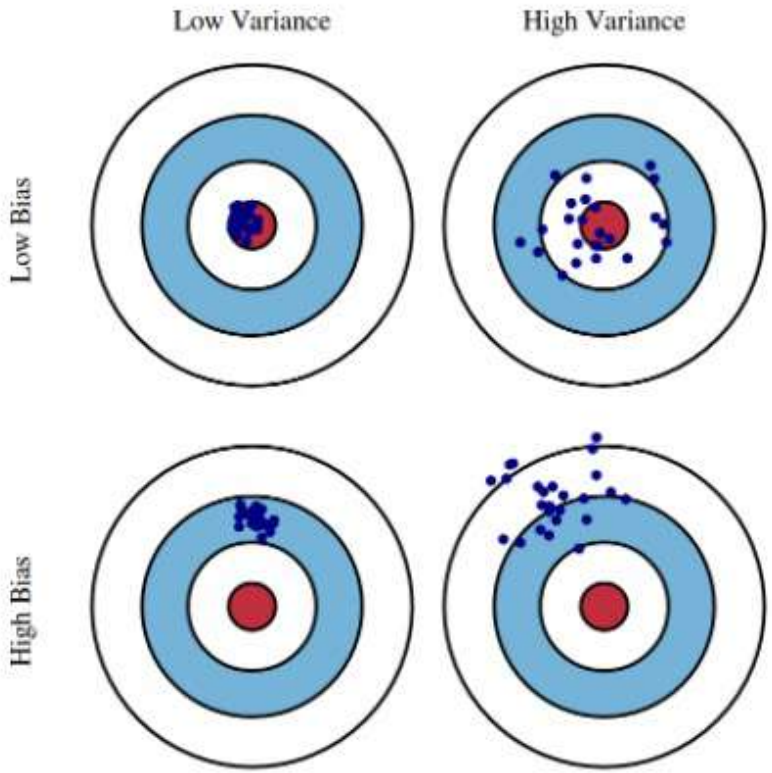


(a) Iteration 100 (b) Iteration 1000 (c) Iteration 10000 (d) Iteration 80000

On the Spectral Bias of Neural Networks Nasim Rahaman et al 2018

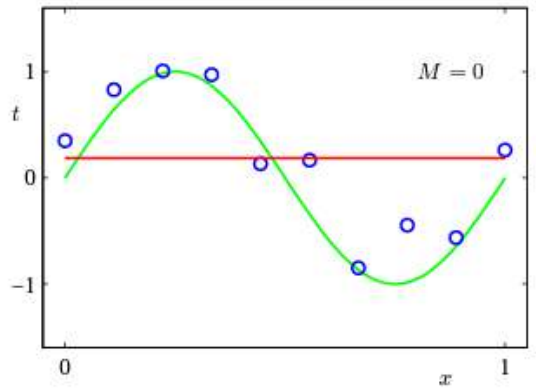
Classical bias variance



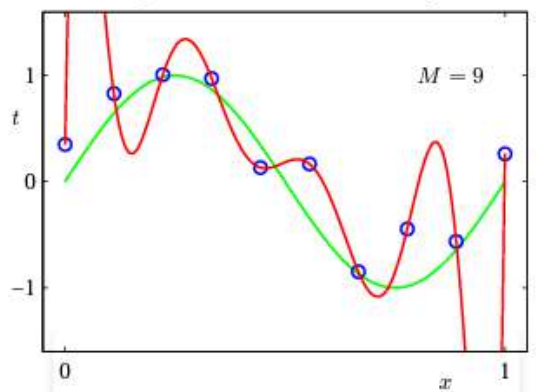


Tune Regularization for good generalization

Underfitting : model is too simple — does not fit the data.



Overfitting : model is too complex — fits perfectly, does not generalize.



An observation

Polynomial Coefficients

	$M = 0$	$M = 1$	$M = 3$	$M = 9$
$w_0^\star$	0.19	0.82	0.31	0.35
$w_1^\star$		-1.27	7.99	232.37
$w_2^\star$			-25.43	-5321.83
$w_3^\star$			17.37	48568.31
$w_4^\star$				-231639.30
$w_5^\star$				640042.26
$w_6^\star$				-1061800.52
$w_7^\star$				1042400.18
$w_8^\star$				-557682.99
$w_9^\star$				125201.43

How can we control the model complexity?

Example of explicit regularization: ridge regression

$$\arg\min_{w} \|y - X^T w\|_2^2 + \lambda \|w\|_2^2$$

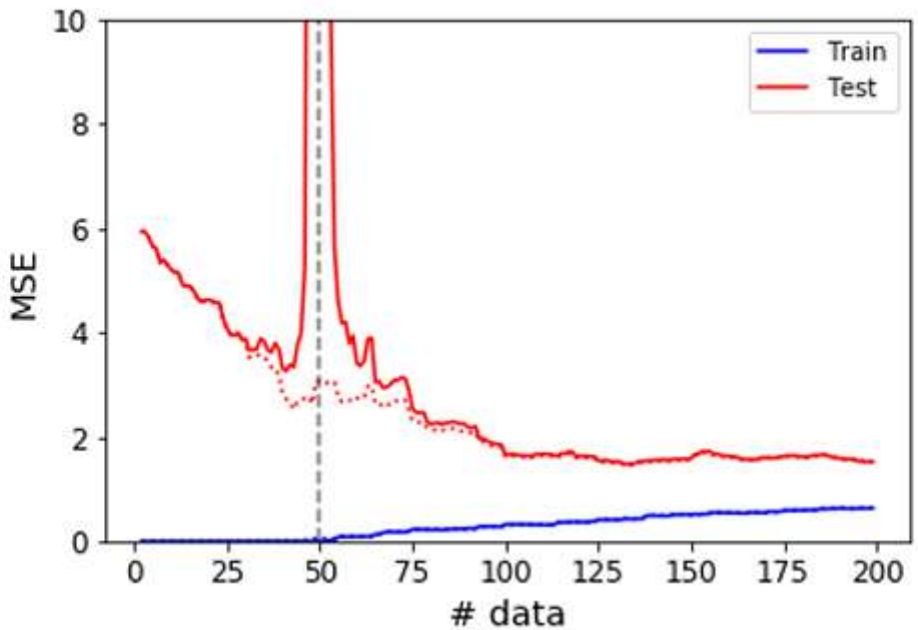
- Reconstruction term $\|y - X^T w\|_2^2$
- Complexity control $\lambda \|w\|_2^2$

Double Descent Phenomenon

https://openai.com/blog/deep-double-descent/

Intuition:

- **Case 1:** $N \gg D$. There's way more than enough data to pin down the optimal parameters, so it generalizes well.
- **Case 2:** $N \approx D$. It can memorize the training set, but just barely. It might need a large $\|\mathbf{w}\|$ to do so.
- **Case 3:** $N \ll D$. It can fit the training set easily. The implicit regularization of gradient descent makes it do so with a small $\|\mathbf{w}\|$.



Although the model a lot of free parameters that can potentially overfit the data its implicit bias decreases its effective complexity and produces good generalization

ANNs hypothesis space is in general complicate: the class of targeted functions is unknown: it strongly depends on the implicit bias!

Generalization: ability of a ML model to perform good on unseen data

Training Data

Testing Data

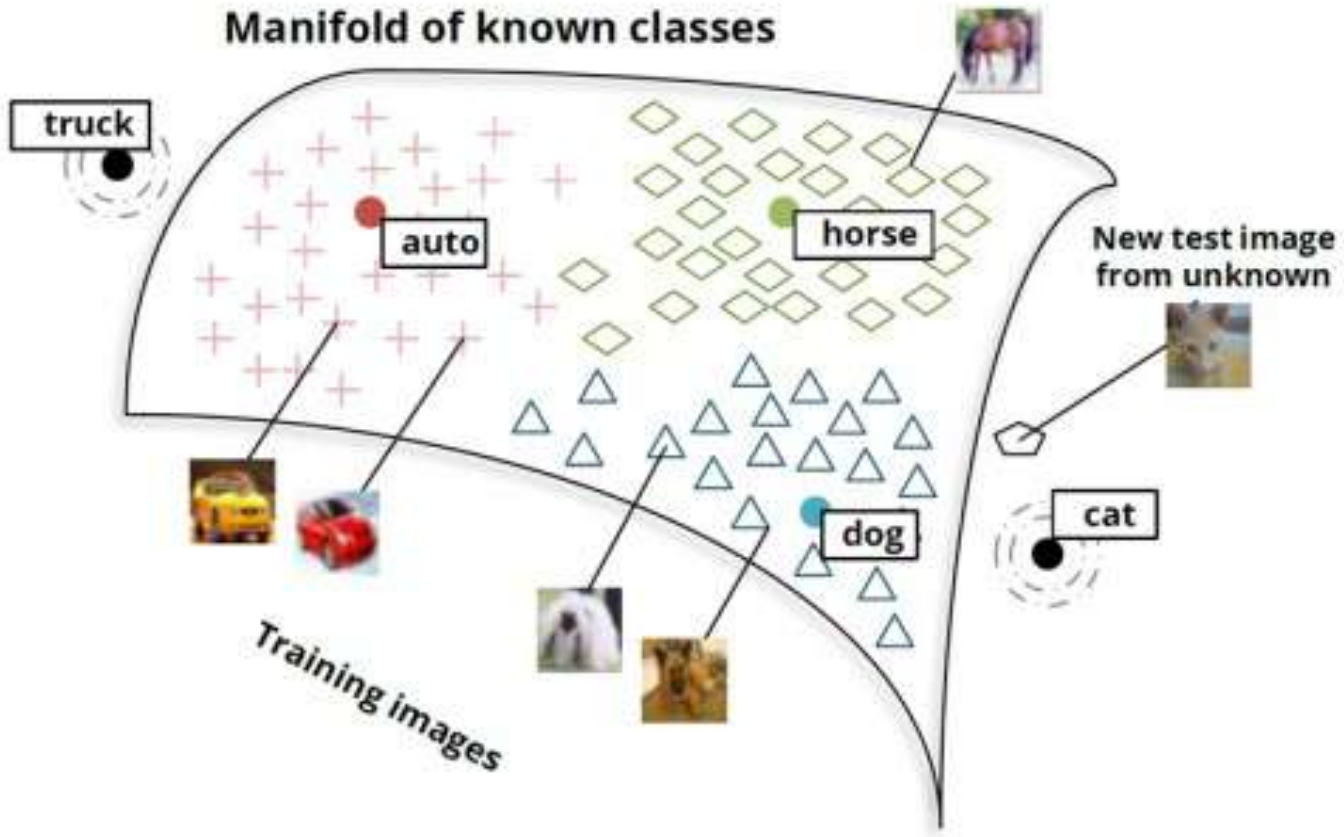
ANNs seem to target the right class of functions to have a good generalization

However…

Ood = out of distribution

Shortcut Learning in Deep Neural Networks, Robert Geirhos et al. 2022

Intuition: data manifold hypothesis



How can we understand if the network separator makes sense?

The network is capable to grasp the geometry of the manifold without overfitting.
How is this possible?

Ansuini et al.

The class of targeted functions are determined by many factors, e.g.

OPTIMIZATION

1. L2 regularization, Noise, Dropout, Data Normalization
2. Early stopping
3. Learning rate
4. Weights initialization
5. Stochasticity (SGD)

ARCHITECTURE

1. Convolutional networks and priors on the weight sharing
2. Depth and compositionality

DATA

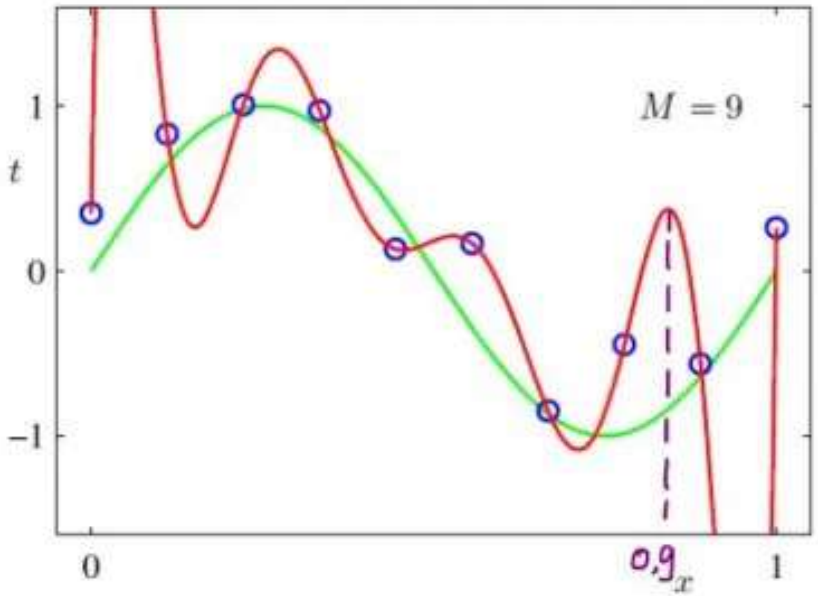
1.Data augmentation and symmetries

2.Adversarial training

Such choices decrease the number of effective degrees of freedom of the model producing good generalization

Let’s start from explicit “regularization”.

Explicit regularization: lp norm



$$h_a(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_m x^m$$

$$\mathcal{L}(w) = \sum_{i=1}^{N} (y_i - h(x_i))^2 + \lambda \|a\|_p$$

What is the effect of the regularization on the loss surface?

Weight decay and hessian interpretation

- We often refer to $\mathbf{H}$ as the curvature of a function.
- Suppose you move along a line defined by $\boldsymbol{\theta} + t\mathbf{v}$ for some vector $\mathbf{v}$.
- Second-order Taylor approximation:

$$\mathcal{J}(\boldsymbol{\theta} + t\mathbf{v}) \approx \mathcal{J}(\boldsymbol{\theta}) + t\nabla\mathcal{J}(\boldsymbol{\theta})^\top\mathbf{v} + \frac{t^2}{2}\mathbf{v}^\top\mathbf{H}(\boldsymbol{\theta})\mathbf{v}$$

- Hence, in a direction where $\mathbf{v}^\top\mathbf{H}\mathbf{v} > 0$, the cost function curves upwards, i.e. has positive curvature. Where $\mathbf{v}^\top\mathbf{H}\mathbf{v} < 0$, it has negative curvature.

Hessian matrix

- The Hessian matrix, denoted $\mathbf{H}$, or $\nabla^2 \mathcal{J}$ is the matrix of second derivatives:

$$\mathbf{H} = \nabla^2 \mathcal{J} = \begin{pmatrix} \frac{\partial^2 \mathcal{J}}{\partial \theta_1^2} & \frac{\partial^2 \mathcal{J}}{\partial \theta_1 \partial \theta_2} & \cdots & \frac{\partial^2 \mathcal{J}}{\partial \theta_1 \partial \theta_D} \\ \frac{\partial^2 \mathcal{J}}{\partial \theta_2 \partial \theta_1} & \frac{\partial^2 \mathcal{J}}{\partial \theta_2^2} & \cdots & \frac{\partial^2 \mathcal{J}}{\partial \theta_2 \partial \theta_D} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 \mathcal{J}}{\partial \theta_D \partial \theta_1} & \frac{\partial^2 \mathcal{J}}{\partial \theta_D \partial \theta_2} & \cdots & \frac{\partial^2 \mathcal{J}}{\partial \theta_D^2} \end{pmatrix}$$

- It's a symmetric matrix because $\frac{\partial^2 \mathcal{J}}{\partial \theta_i \partial \theta_j} = \frac{\partial^2 \mathcal{J}}{\partial \theta_j \partial \theta_i}$.

Assuming no bias parameter, the objective function is:

$$\tilde{J}(w; X, y) = \frac{\alpha}{2} w^\top w + J(w; X, y)$$

with the gradient:

$$\nabla_w \tilde{J}(w; X, y) = \alpha w + \nabla_w J(w; X, y).$$

A single gradient update step is:

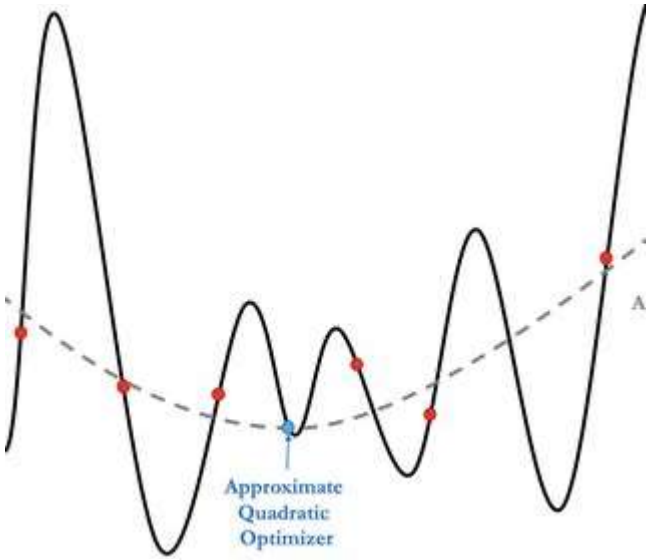
$$w \leftarrow w - \epsilon(\alpha w + \nabla_w J(w; X, y)),$$

or equivalently:

$$w \leftarrow (1 - \epsilon\alpha) w - \epsilon \nabla_w J(w; X, y).$$

This shows that weight decay modifies the learning rule by shrinking the weights by a factor of $(1 - \epsilon\alpha)$ before the gradient update. Further analysis considers a quadratic approximation around the unregularized minimum w^*.

A simplification



The approximation of the objective function $\tilde{J}$ is given by:

$$\tilde{J}(\theta) = J(w^*) + \frac{1}{2}(w - w^*)^\top H(w - w^*),$$

where H is the Hessian matrix of J with respect to w, evaluated at w^*. Since w^* is defined to be a minimum, the first-order term vanishes. The Hessian H is positive semidefinite because w^* is a minimum.

The minimum of $\tilde{J}$ occurs where its gradient:

$$\nabla_w \tilde{J}(w) = H(w - w^*) \qquad *$$

is equal to zero.

To study the effect of weight decay, we modify equation (*) by adding the weight decay gradient and solve for the minimum of the regularized objective function. The location of the minimum is represented by $\tilde{w}$. (the new minimum!)

Starting from:

$$\alpha\tilde{w} + H(\tilde{w} - w^*) = 0,$$

we can rewrite this as:

$$(H + \alpha I)\tilde{w} = Hw^*,$$

leading to:

$$\tilde{w} = (H + \alpha I)^{-1} Hw^*.$$

As $\alpha \to 0$, $\tilde{w}$ approaches w^*. To explore what happens as α increases, we decompose H into $Q\Lambda Q^\top$, where Q is an orthonormal matrix of eigenvectors and Λ is a diagonal matrix of eigenvalues. Applying this decomposition:

$$\tilde{w} = (Q\Lambda Q^\top + \alpha I)^{-1} Q\Lambda Q^\top w^*,$$

simplifying to:

$$\tilde{w} = Q(\Lambda + \alpha I)^{-1} \Lambda Q^\top w^*.$$

$$\tilde{w} = Q(\Lambda + \alpha I)^{-1}\Lambda Q^\top w^*$$

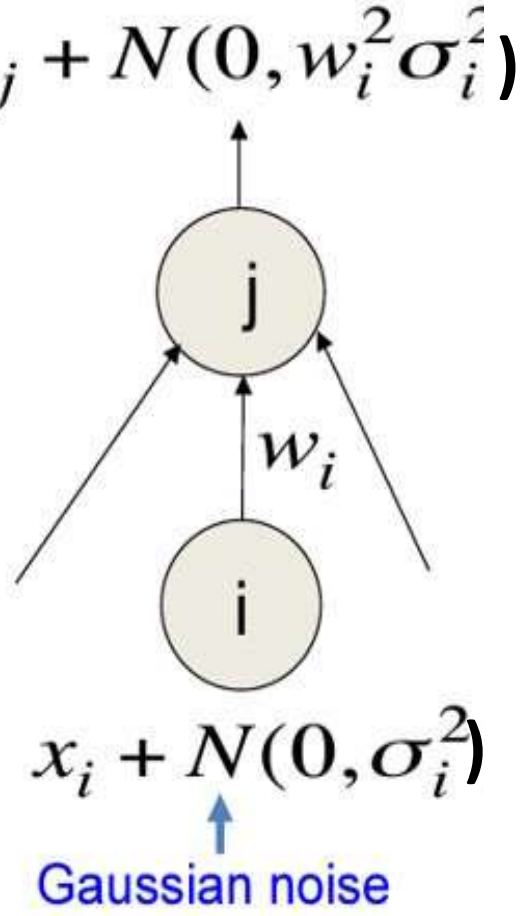
This shows that weight decay scales w^* along the eigenvector directions of H. Each component along the i-th eigenvector of H is rescaled by a factor of $\frac{\lambda_i}{\lambda_i+\alpha}$.

- If $\lambda_i \gg \alpha$, regularization has little effect.
- If $\lambda_i \ll \alpha$, components are shrunk significantly, approaching zero.

This explains how regularization influences directions with small eigenvalues, shrinking their magnitudes more aggressively.

L2 regularization via noisy inputs

- Suppose we add Gaussian noise to the inputs.
  - The variance of the noise is amplified by the squared weight before going into the next layer.
- In a simple net with a linear output unit directly connected to the inputs, the amplified noise gets added to the output.
- This makes an additive contribution to the squared error.
  - So minimizing the squared error tends to minimize the squared weights when the inputs are noisy.



output on one case → $y^{noisy} = \sum_i w_i x_i + \sum_i w_i \varepsilon_i$ *where* ε_i *is sampled from* $N(0, \sigma_i^2)$

$$E\left[(y^{noisy} - t)^2\right] = E\left[\left(y + \sum_i w_i \varepsilon_i - t\right)^2\right] = E\left[\left((y - t) + \sum_i w_i \varepsilon_i\right)^2\right]$$

$$= (y - t)^2 + E\left[2(y - t)\sum_i w_i \varepsilon_i\right] + E\left[\left(\sum_i w_i \varepsilon_i\right)^2\right]$$

$$= (y - t)^2 + E\left[\sum_i w_i^2 \varepsilon_i^2\right]$$

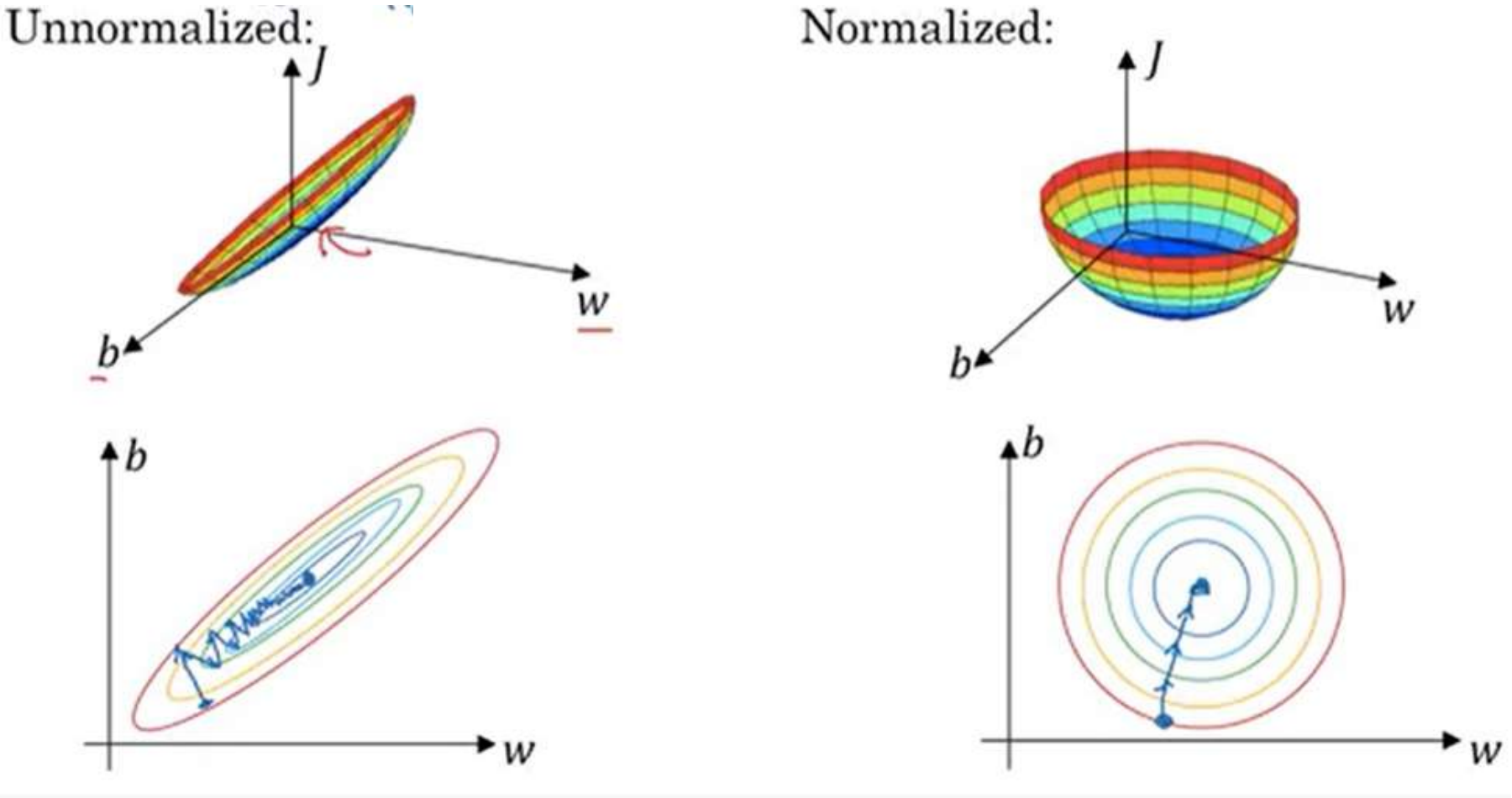
because ε_i *is independent of* ε_j
and ε_i *is independent of* $(y - t)$

$$= (y - t)^2 + \sum_i w_i^2 \sigma_i^2$$

So σ_i^2 is equivalent to an L2 penalty

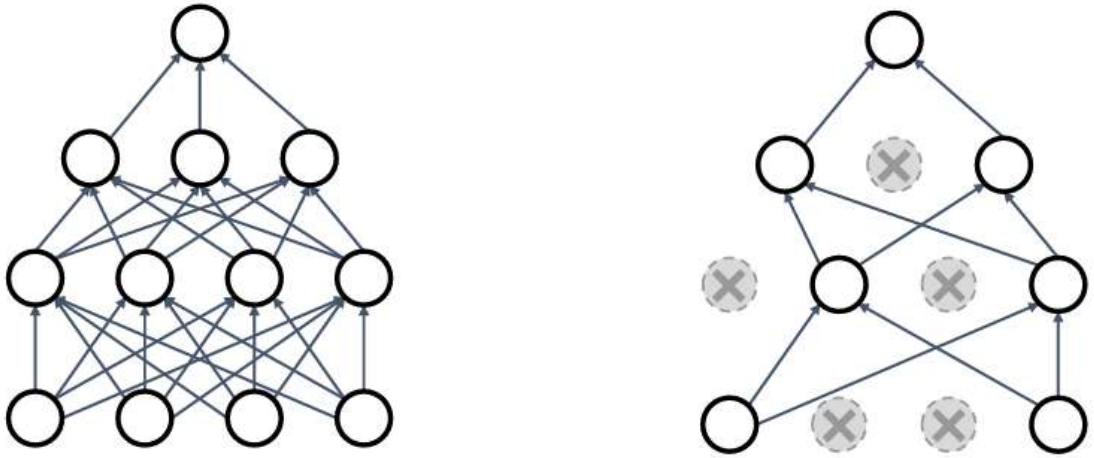
Is this true for more complex nets?

Data normalization changes the geometry of the loss

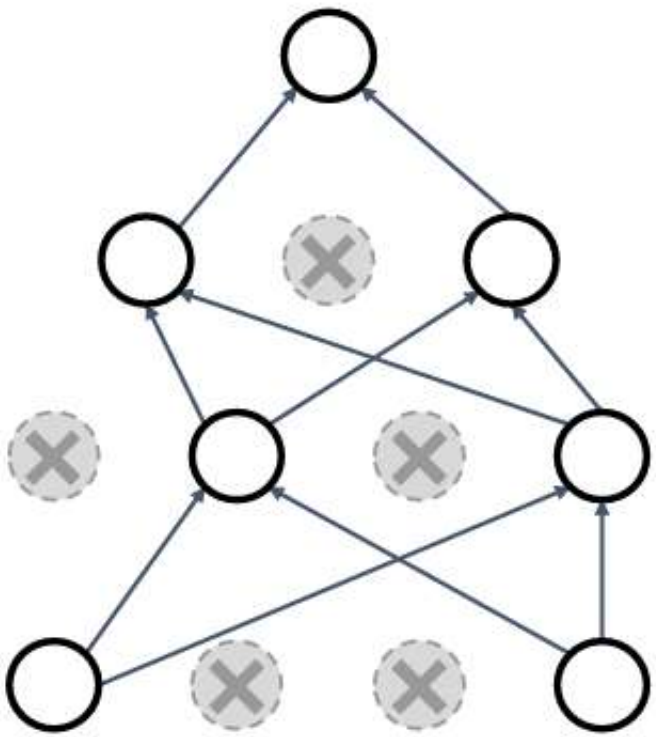


Another example of explicit NNs regularization: Dropout

In each forward pass, randomly set some neurons to zero
Probability of dropping is a hyperparameter; 0.5 is common

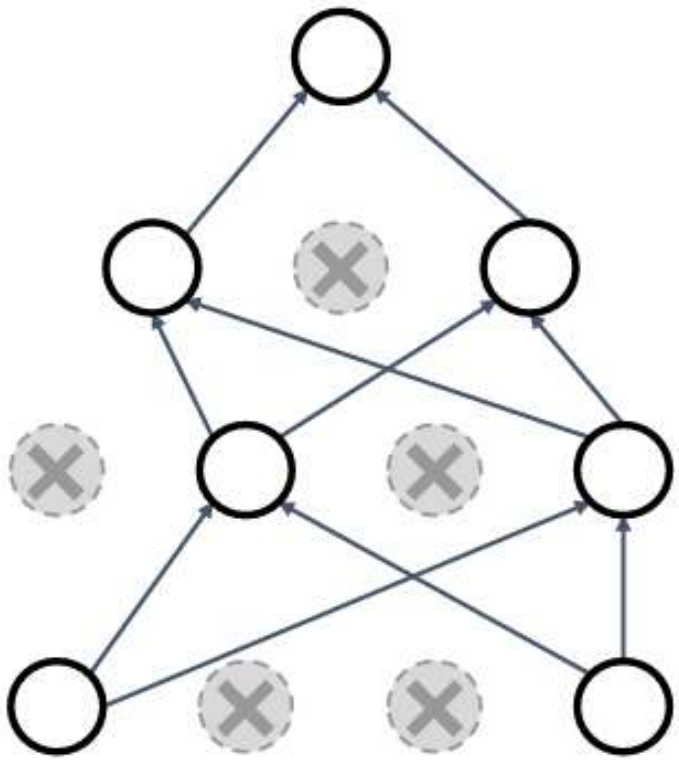


Regularization: Dropout



Forces the network to have a redundant representation; Prevents **co-adaptation** of features

Regularization: Dropout



Another interpretation:

Dropout is training a large **ensemble** of models (that share parameters).

Each binary mask is one model

An FC layer with 4096 units has
$2^{4096} \sim 10^{1233}$ possible masks!
Only $\sim 10^{82}$ atoms in the universe...

Class 3: Gradient Descent and implicit regularization

- Regression and minimum norm solution
- Matrix completion problem
- Early stopping and L2
- Weights initialization
- SGD regularization, discretization

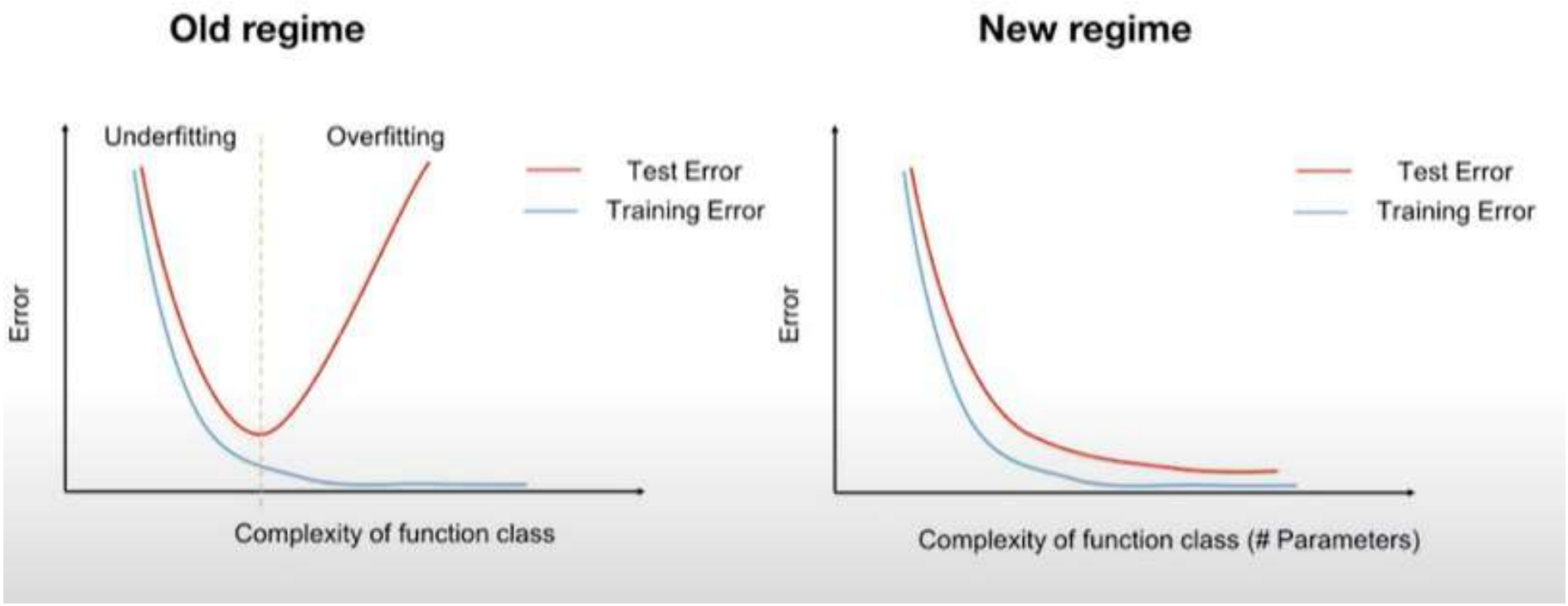
Importance of symmetries in data
Data augmented Loss example

$$\mathcal{L}(w) = \frac{1}{N}\sum_{i=1}^{N}\int \ell\left(\sigma\langle w, g_\theta x_i\rangle; y_i\right) d\theta$$

$$\begin{aligned}
\mathcal{L}(g_{\bar{\theta}} w) &= \frac{1}{N}\sum_{i=1}^{N}\int \ell\left(\sigma\langle w, g^* g_\theta x_i\rangle; y_i\right) d\theta \\
&= \sum_{i=1}^{N}\int \ell\left(\sigma\langle w, g_\theta x_i\rangle; y_i\right) d\theta \\
&= \mathcal{L}(w)
\end{aligned}$$

$$g_\theta g_{\theta'} = g_{\theta''}$$

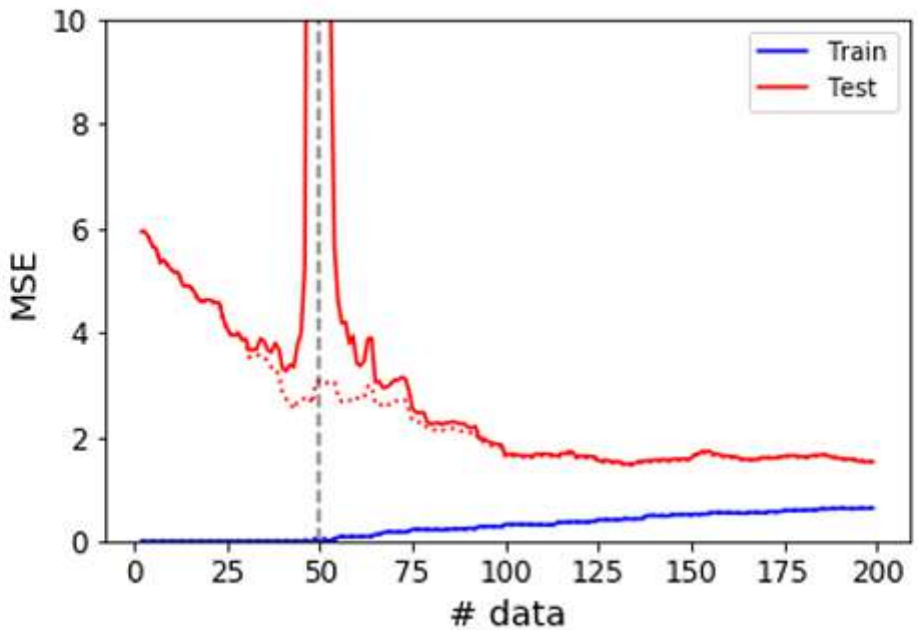
Implicit regularization



Number of parameters grows but the effective learnable parameters are bounded!!!!

Intuition:

- **Case 1:** $N \gg D$. There's way more than enough data to pin down the optimal parameters, so it generalizes well.
- **Case 2:** $N \approx D$. It can memorize the training set, but just barely. It might need a large $\|\mathbf{w}\|$ to do so.
- **Case 3:** $N \ll D$. It can fit the training set easily. The implicit regularization of gradient descent makes it do so with a small $\|\mathbf{w}\|$.



How do we measure the network complexity?

We are measuring the network complexity in the wrong way

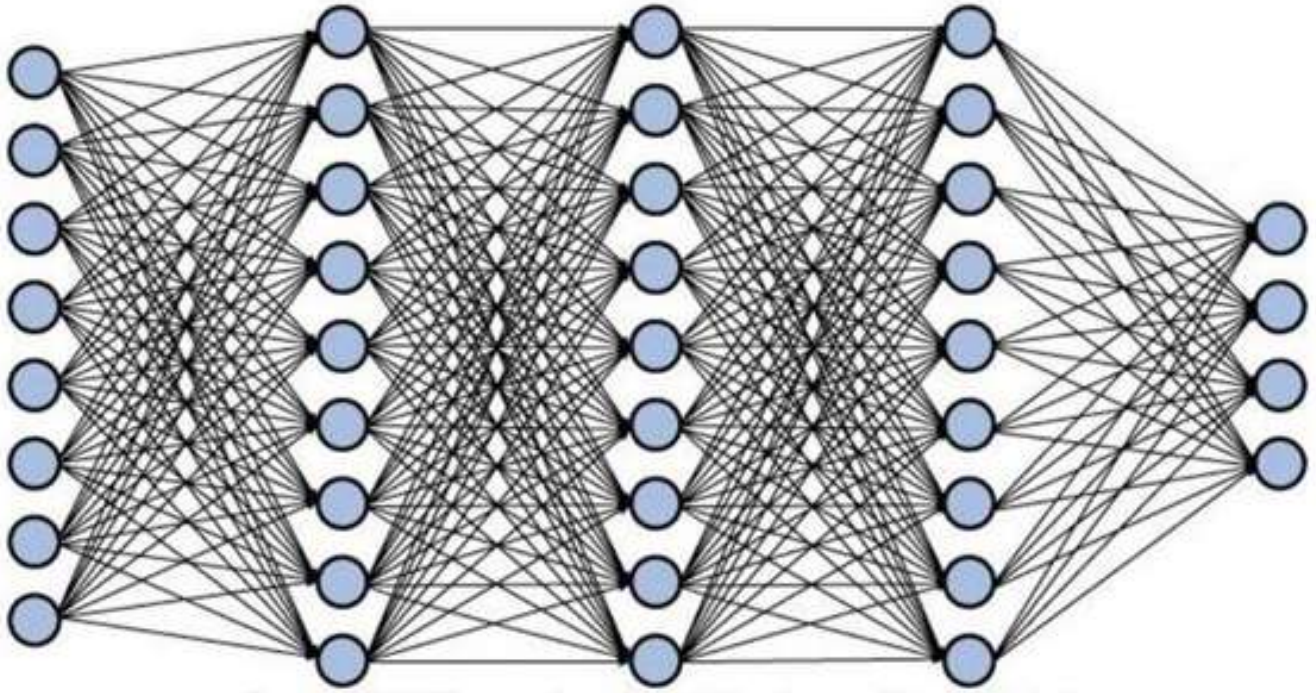
- Implicit regularization occurs when the dynamics of training lead to certain minima rather than others

- Algorithm
- Initialization
- Architecture

https://www.youtube.com/watch?v=gh9vrvLx7Mo (Srebro)

Geometric picture: many global minima

of learned weights $\gg$ # of training examples

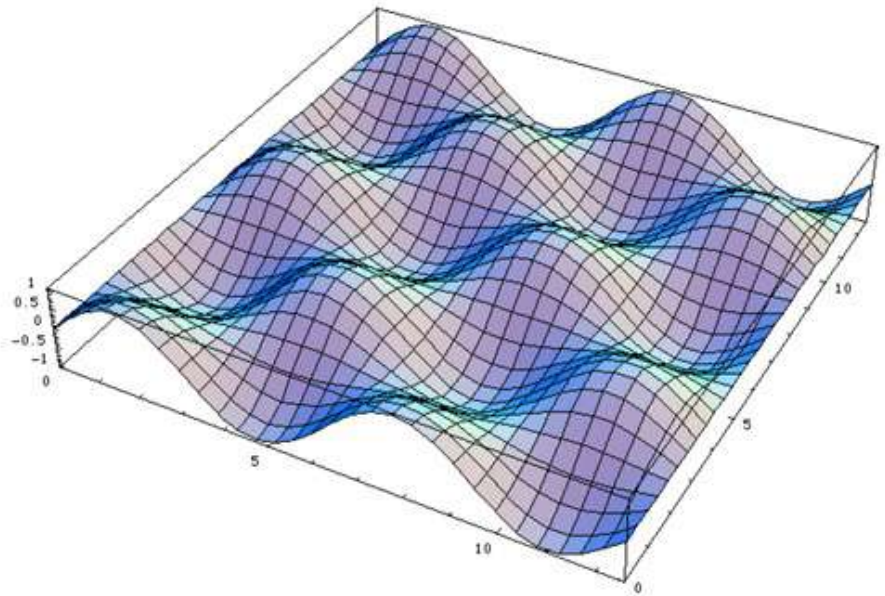


$\gg$

What do you expect in such regime?

Geometric picture: many global minima

Multiple global minima: some generalize well, others don't



Solution found by Gradient Descent (GD) often generalizes well

Conventional Wisdom

Gradient-based optimization induces an implicit regularization

Example with regression: minimum norm solution

$A \in \mathbb{R}^{m \times n}$, $x \in \mathbb{R}^n$, $m \gg n$

$Ax = y$

Parameters are more than data

one particular solution is

$$x_{\ln} = A^T(AA^T)^{-1}y$$

Moore-Penrose pseudo-inverse
(a possible algorithm)

(AA^T is invertible since A full rank)

in fact, $x_{\ln}$ is the solution of $y = Ax$ that minimizes $\|x\|$

i.e., $x_{\ln}$ is solution of optimization problem

$$\begin{array}{ll} \text{minimize} & \|x\| \\ \text{subject to} & Ax = y \end{array}$$

(with variable $x \in \mathbf{R}^n$)

Example with **regression**: close form is minimum norm solution

suppose $Ax = y$, so $A(x - x_{\ln}) = 0$ and

$$\begin{aligned}(x - x_{\ln})^T x_{\ln} &= (x - x_{\ln})^T A^T (AA^T)^{-1} y \\ &= (A(x - x_{\ln}))^T (AA^T)^{-1} y \\ &= 0\end{aligned}$$

i.e., $(x - x_{\ln}) \perp x_{\ln}$, so

$$\|x\|^2 = \|x_{\ln} + x - x_{\ln}\|^2 = \|x_{\ln}\|^2 + \|x - x_{\ln}\|^2 \geq \|x_{\ln}\|^2$$

i.e., $x_{\ln}$ has smallest norm of any solution

Regression: minimum norm solution with GD

Let cost function $f : \mathbb{R}^n \to \mathbb{R}$ be defined by

$$f(\mathrm{x}) := \frac{1}{2}\|\mathrm{Ax} - \mathrm{b}\|_2^2$$

$$A \in \mathbb{R}^{m \times n}, \; x \in \mathbb{R}^n, \quad m \gg n$$

whose gradient is

$$\nabla f(\mathrm{x}) = \mathrm{A}^\top (\mathrm{Ax} - \mathrm{b})$$

Using gradient descent with step $\mu > 0$,

$$\begin{aligned} \mathrm{x}_{k+1} &= \mathrm{x}_k - \mu \nabla f(\mathrm{x}_k) \\ &= \left(\mathrm{I} - \mu \mathrm{A}^\top \mathrm{A}\right) \mathrm{x}_k + \mu \mathrm{A}^\top \mathrm{b} \end{aligned}$$

Hence,

$$\mathrm{x}_k = \left(\mathrm{I} - \mu \mathrm{A}^\top \mathrm{A}\right)^k \mathrm{x}_0 + \mu \sum_{\ell=0}^{k-1} \left(\mathrm{I} - \mu \mathrm{A}^\top \mathrm{A}\right)^\ell \mathrm{A}^\top \mathrm{b}$$

$$\mathrm{A} = \mathrm{U}\Sigma\mathrm{V}^\top = \mathrm{U}\begin{bmatrix}\Sigma_1 & \mathrm{O}\end{bmatrix}\begin{bmatrix}\mathrm{V}_1^\top \\ \mathrm{V}_2^\top\end{bmatrix} = \mathrm{U}\Sigma_1\mathrm{V}_1^\top$$

Letting $\mathrm{y} := \mathrm{V}^\top \mathrm{x}$, we rewrite

$$\begin{aligned} \mathrm{y}_k &= \left(\mathrm{I} - \mu \Sigma^\top \Sigma\right)^k \mathrm{y}_0 + \mu \sum_{\ell=0}^{k-1} \left(\mathrm{I} - \mu \Sigma^\top \Sigma\right)^\ell \Sigma^\top \mathrm{U}^\top \mathrm{b} \\ &= \begin{bmatrix} \left(\mathrm{I} - \mu \Sigma_1^2\right)^k & \mathrm{O} \\ \mathrm{O} & \mathrm{I} \end{bmatrix} \mathrm{y}_0 + \mu \sum_{\ell=0}^{k-1} \begin{bmatrix} \left(\mathrm{I} - \mu \Sigma_1^2\right)^\ell & \mathrm{O} \\ \mathrm{O} & \mathrm{I} \end{bmatrix} \begin{bmatrix} \Sigma_1 \\ \mathrm{O} \end{bmatrix} \mathrm{U}^\top \mathrm{b} \\ &= \begin{bmatrix} \left(\mathrm{I} - \mu \Sigma_1^2\right)^k & \mathrm{O} \\ \mathrm{O} & \mathrm{I} \end{bmatrix} \mathrm{y}_0 + \mu \sum_{\ell=0}^{k-1} \begin{bmatrix} \left(\mathrm{I} - \mu \Sigma_1^2\right)^\ell \Sigma_1 \\ \mathrm{O} \end{bmatrix} \mathrm{U}^\top \mathrm{b} \end{aligned}$$

Example with **regression**: minimum norm solution with GD

Choosing $\mu > 0$ such that all eigenvalues of $\mathrm{I} - \mu\Sigma_1^2$ are strictly inside the unit circle, then $\mathrm{y}_k \to \mathrm{y}_\infty$, where

$$\mathrm{y}_\infty = \begin{bmatrix} \mathrm{O} & \mathrm{O} \\ \mathrm{O} & \mathrm{I} \end{bmatrix} \mathrm{y}_0 + \mu \sum_{\ell=0}^{\infty} \begin{bmatrix} \left(\mathrm{I} - \mu\Sigma_1^2\right)^\ell \Sigma_1 \\ \mathrm{O} \end{bmatrix} \mathrm{U}^\top \mathrm{b}$$

where

$$\mu \sum_{\ell=0}^{\infty} \left(\mathrm{I} - \mu\Sigma_1^2\right)^\ell \Sigma_1 = \mu\left(\mathrm{I} - \mathrm{I} + \mu\Sigma_1^2\right)^{-1} \Sigma_1 = \Sigma_1^{-1}$$

and, thus,

$$\mathrm{y}_\infty = \begin{bmatrix} \mathrm{O} & \mathrm{O} \\ \mathrm{O} & \mathrm{I} \end{bmatrix} \mathrm{y}_0 + \begin{bmatrix} \Sigma_1^{-1} \\ \mathrm{O} \end{bmatrix} \mathrm{U}^\top \mathrm{b}$$

Since $\mathrm{x} := \mathrm{V}\mathrm{y}$,

$$\boxed{\mathrm{x}_\infty = \mathrm{V}_2 \mathrm{V}_2^\top \mathrm{x}_0 + \underbrace{\mathrm{V}_1 \Sigma_1^{-1} \mathrm{U}^\top \mathrm{b}}_{=\mathrm{x}_{\mathrm{LN}}}}$$

This is the pseudo-inverse

Therefore, we conclude that if x_0 is orthogonal to the null space of A, then gradient descent will converge to the least-norm solution.

(Low rank) Matrix completion and nuclear norm implicit regularization

Matrix completion: recover low-rank matrix given subset of entries

				
Bob	4	?	?	4
Alice	?	5	4	?
Joe	?	5	?	?

observed entries $\longleftrightarrow$ *training data*

unobserved entries $\longleftrightarrow$ *test data*

Denote observations by $\{b_{ij}\}_{(i,j)\in\Omega}$

Low rank and nuclear norm?

$$\|A\|_* = \sum_i \sigma_i(A)$$

Convex Programming Approach

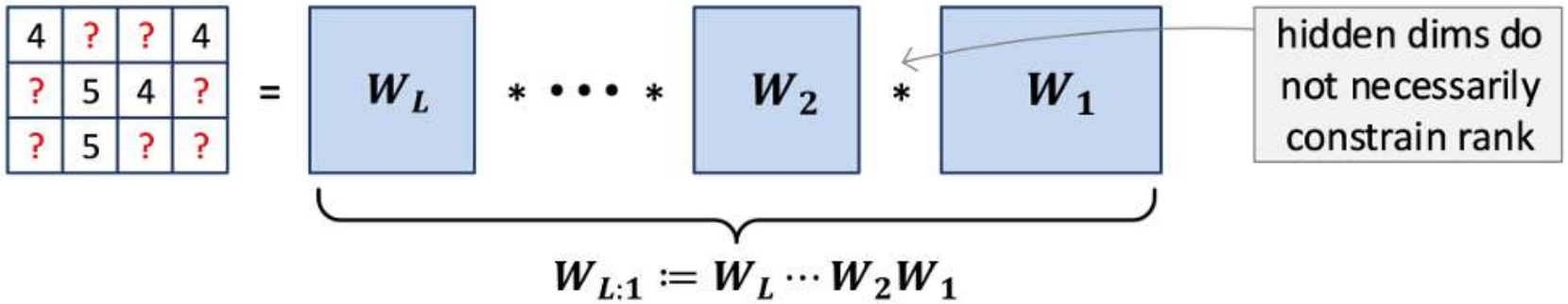
Find minimal nuclear norm solution:

$$\min \|W\|_{nuclear} \quad s.t. \quad W_{ij} = b_{ij} \quad \forall (i,j) \in \Omega$$

Perfectly recovers if observations are sufficiently many (Candes & Recht 2008)

Matrix completion and nuclear norm

Parameterize solution as LNN and fit observations using GD (over ℓ_2 loss)



GD is run w.r.t. $W_1, \ldots, W_L$ over:

$$\ell(W_{L:1}) = \frac{1}{2} \sum_{(i,j) \in \Omega} \left((W_{L:1})_{ij} - b_{ij} \right)^2$$

To which solutions does **product matrix** $W_{L:1}$ converge?

Empirical phenomenon: low-rank matrix often recovered accurately

Matrix completion and nuclear norm

Conjecture (Gunasekar et al. 2017)

With small learning rate and init close to the origin, GD over depth 2 matrix factorization converges to min nuclear norm solution.

GD implicitly solves convex programming approach?

Gunasekar et al. supported conjecture with:

- Experiments
- Proof for certain restricted case

Conjecture established under other restricted conditions:

- Li et al. 2018
- Belabbas 2020

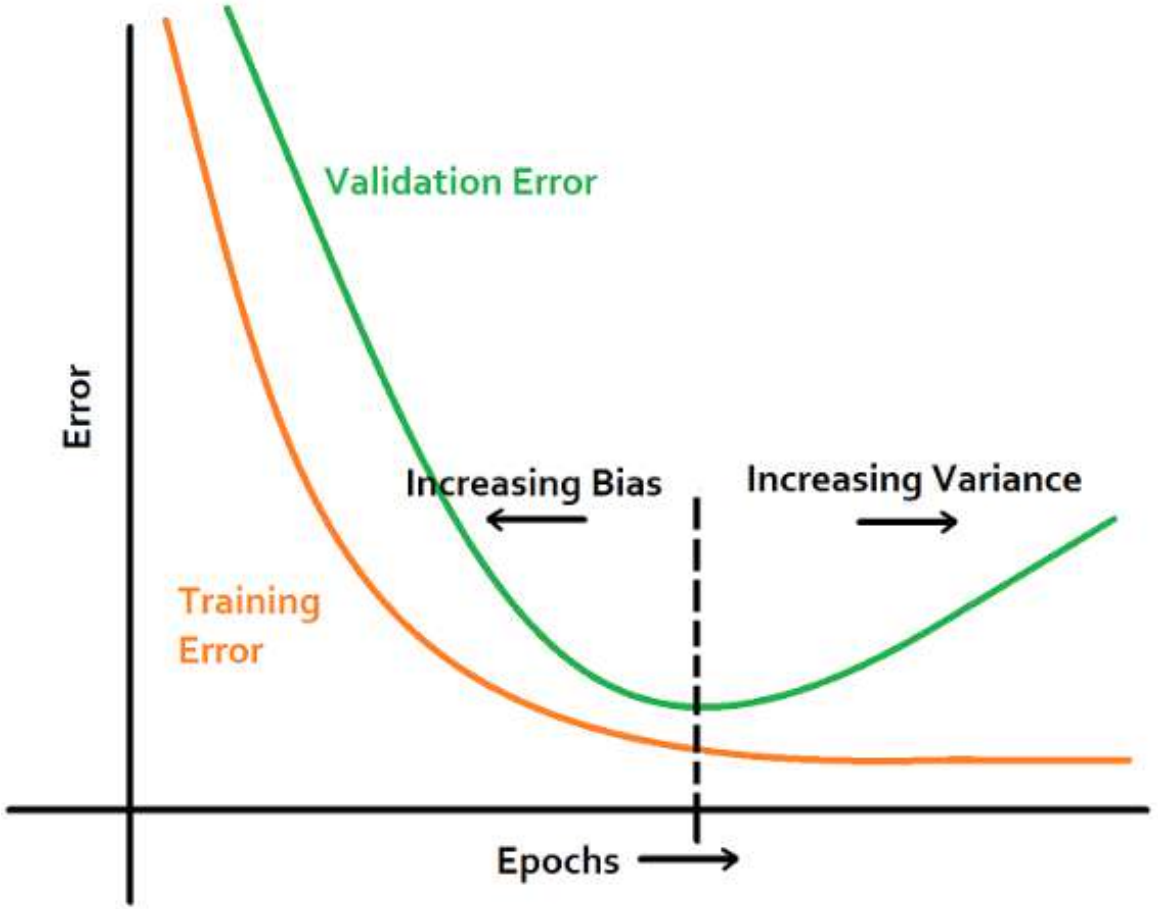
Convergence occurs when:

Data are low rank

The IR of the model is favoring low rank solutions

Is crucial that the data priors matches the model and optimization strategy (IR) !!!!

Early stopping, L2 norm and learning rate



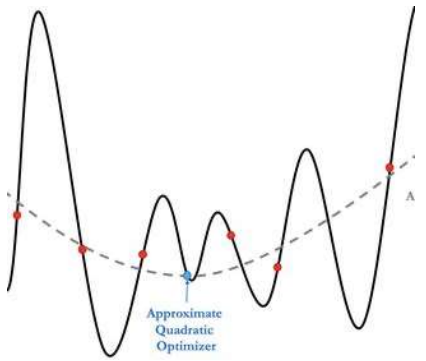
From Taylor series expansion, we will make a quadratic approximation around the empirically optimal value of the weights $\boldsymbol{w}^*$. The matrix $\mathbf{H}$ is the hessian matrix.

$$\hat{J}(\boldsymbol{\theta}) = J(\boldsymbol{w}^*) + \frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}^*)^\top \boldsymbol{H}(\boldsymbol{w} - \boldsymbol{w}^*),$$

$$\nabla_{\boldsymbol{w}} \hat{J}(\boldsymbol{w}) = \boldsymbol{H}(\boldsymbol{w} - \boldsymbol{w}^*).$$

Since $\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} J(\boldsymbol{w})$

$$\nabla_{\boldsymbol{w}} J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) \Big|_{\boldsymbol{w}^*} = 0$$



The gradient of this approximation comes out to be a linear function of H and w. Now, we can calculate the weights at each step of the gradient descent.

$$\begin{aligned} \boldsymbol{w}^{(\tau)} &= \boldsymbol{w}^{(\tau-1)} - \epsilon \nabla_{\boldsymbol{w}} \hat{J}(\boldsymbol{w}^{(\tau-1)}) \\ &= \boldsymbol{w}^{(\tau-1)} - \epsilon \boldsymbol{H}(\boldsymbol{w}^{(\tau-1)} - \boldsymbol{w}^*) \\ \boldsymbol{w}^{(\tau)} - \boldsymbol{w}^* &= (\boldsymbol{I} - \epsilon \boldsymbol{H})(\boldsymbol{w}^{(\tau-1)} - \boldsymbol{w}^*). \end{aligned}$$

Since the Hessian matrix is real, symmetric and positive semi-definite. By eigenvalue decomposition. matrix H can be written as,

$$\begin{aligned} \boldsymbol{H} &= \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^\top \\ \boldsymbol{w}^{(\tau)} - \boldsymbol{w}^* &= (\boldsymbol{I} - \epsilon \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^\top)(\boldsymbol{w}^{(\tau-1)} - \boldsymbol{w}^*) \\ \boldsymbol{Q}^\top(\boldsymbol{w}^{(\tau)} - \boldsymbol{w}^*) &= (\boldsymbol{I} - \epsilon \boldsymbol{\Lambda})\boldsymbol{Q}^\top (\boldsymbol{w}^{(\tau-1)} - \boldsymbol{w}^*) \end{aligned}$$

Assuming ε is chosen to be small and $w(0) = 0$. The weights after τ **iterations** are:

$$\boldsymbol{Q}^\top \boldsymbol{w}^{(\tau)} = [\boldsymbol{I} - (\boldsymbol{I} - \epsilon \boldsymbol{\Lambda})^\tau] \boldsymbol{Q}^\top \boldsymbol{w}^* \qquad (1)$$

The optimal weights obtained on imposing L2 norm penalty on the objective function is given by (refer):

$$\boldsymbol{Q}^\top \tilde{\boldsymbol{w}} = (\boldsymbol{\Lambda} + \alpha \boldsymbol{I})^{-1} \boldsymbol{\Lambda} \boldsymbol{Q}^\top \boldsymbol{w}^*$$
$$\boldsymbol{Q}^\top \tilde{\boldsymbol{w}} = [\boldsymbol{I} - (\boldsymbol{\Lambda} + \alpha \boldsymbol{I})^{-1} \alpha] \boldsymbol{Q}^\top \boldsymbol{w}^* \qquad (2)$$

Comparing Eq(1) and Eq(2), we derive the following relation between τ, **w, and** ε**.** The equivalence can be seen between the L2 regularization and early stopping.

$$(\boldsymbol{I} - \epsilon \boldsymbol{\Lambda})^\tau = (\boldsymbol{\Lambda} + \alpha \boldsymbol{I})^{-1} \alpha,$$

Taking log both sides and using the series approximation of log(1+x), we can conclude that if all λi are small (that is, ελi << 1 and λi/α << 1) then the following equation holds.

$$\tau \approx \frac{1}{\epsilon \alpha},$$
$$\alpha \approx \frac{1}{\tau \epsilon}.$$

Here, α is the regularization constant, τ is no. of iterations, and ε is the learning rate.

$$1 - \frac{\alpha}{\Lambda + \alpha I} = \frac{\Lambda + \alpha I - \alpha I}{\Lambda + \alpha I} = \frac{\Lambda}{\Lambda + \alpha I}$$

$$\log(1 - x) \approx x$$

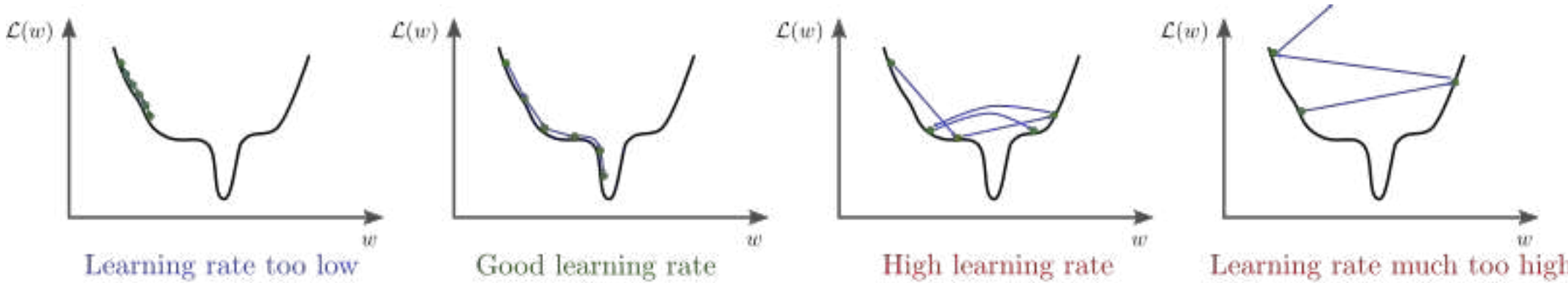
$$\tau \log(I - \epsilon \Lambda) = \log(\alpha) - \log(\Lambda + \alpha I)$$
$$\tau \log(I - \epsilon \Lambda) = \log(\alpha) - \log(\alpha(\frac{\Lambda}{\alpha} + I))$$
$$\tau \log(I - \epsilon \Lambda) = -\log(\frac{\Lambda}{\alpha} + I)$$
$$\tau \epsilon \Lambda = \frac{\lambda}{\alpha}$$

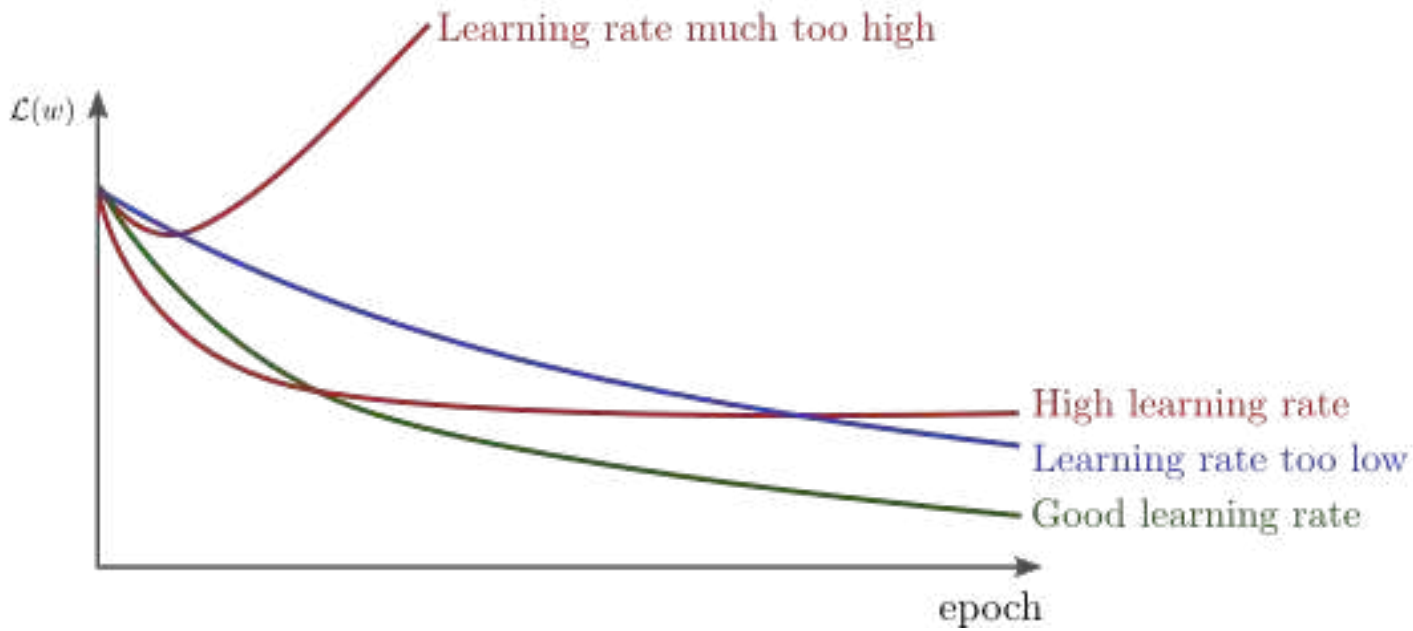
Increasing no. of epochs/iterations τ is equivalent to reducing the regularization constant. Similarly, early stopping the model, i.e. reducing the no. of iterations is similar to L2 regularization with large α. Thus, we can say that early stopping regularizes the model.

References:

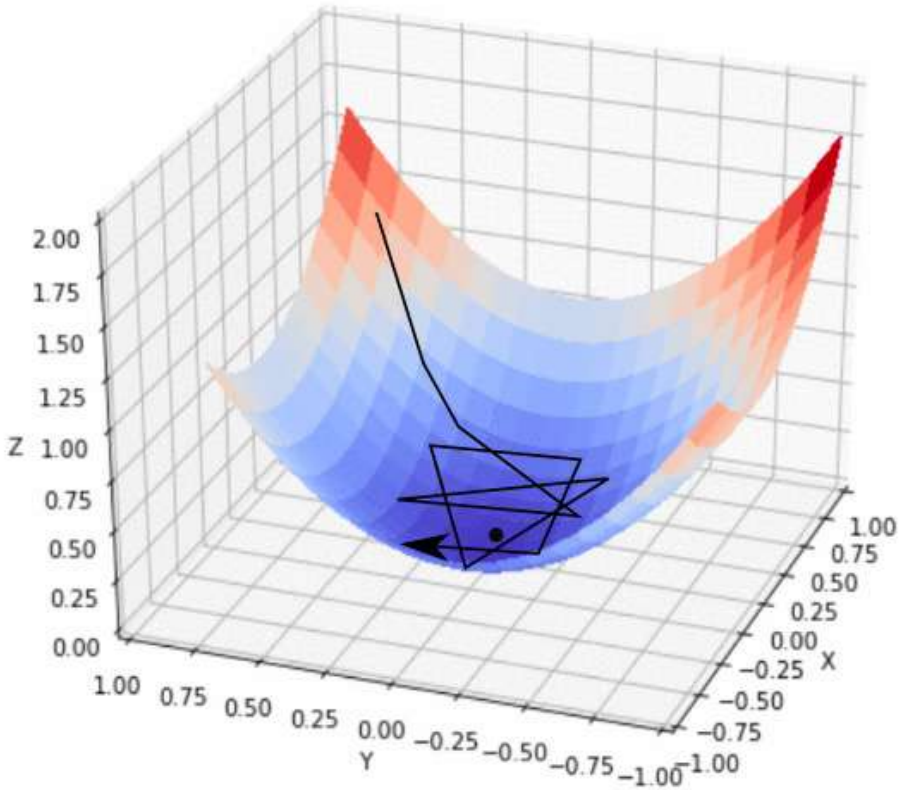
Deep Learning (Adaptive Computation and Machine Learning series) — By Ian Goodfellow, Yoshua Bengio, Aaron Courville

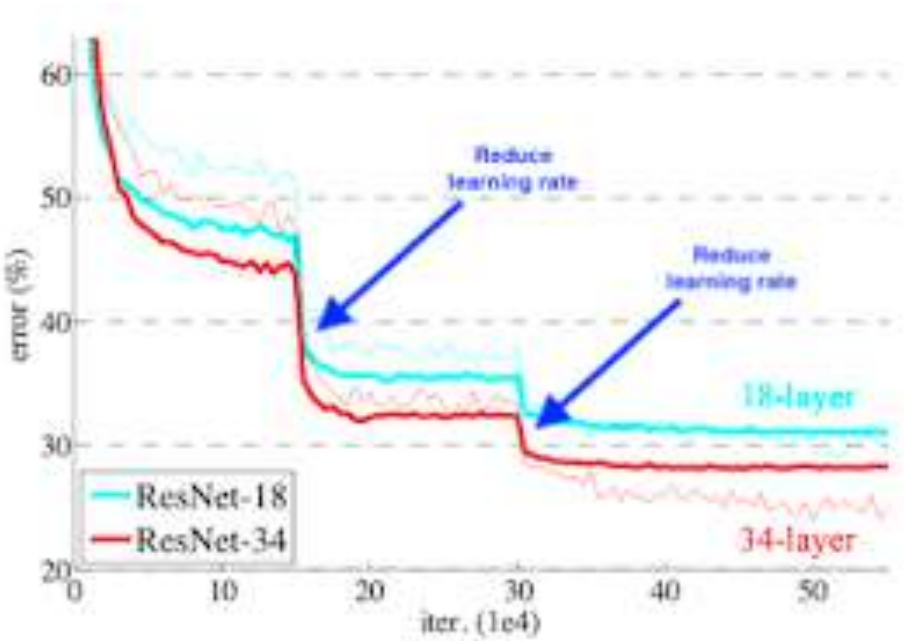
Learning rate is important





Learning rate for SGD: scheduling





Implicit L2!

Weights initialization

Weight Initialization: Xavier Initialization

"Xavier" initialization:
std = 1/sqrt(Din)

Derivation: Variance of output = Variance of input

$y = Wx$ $\qquad y_i = \sum_{j=1}^{Din} x_j w_j$

$\text{Var}(y_i) = Din * \text{Var}(x_i w_i)$ [Assume x, w are iid]

$= Din * (E[x_i^2]E[w_i^2] - E[x_i]^2 E[w_i]^2)$ [Assume x, w independent]

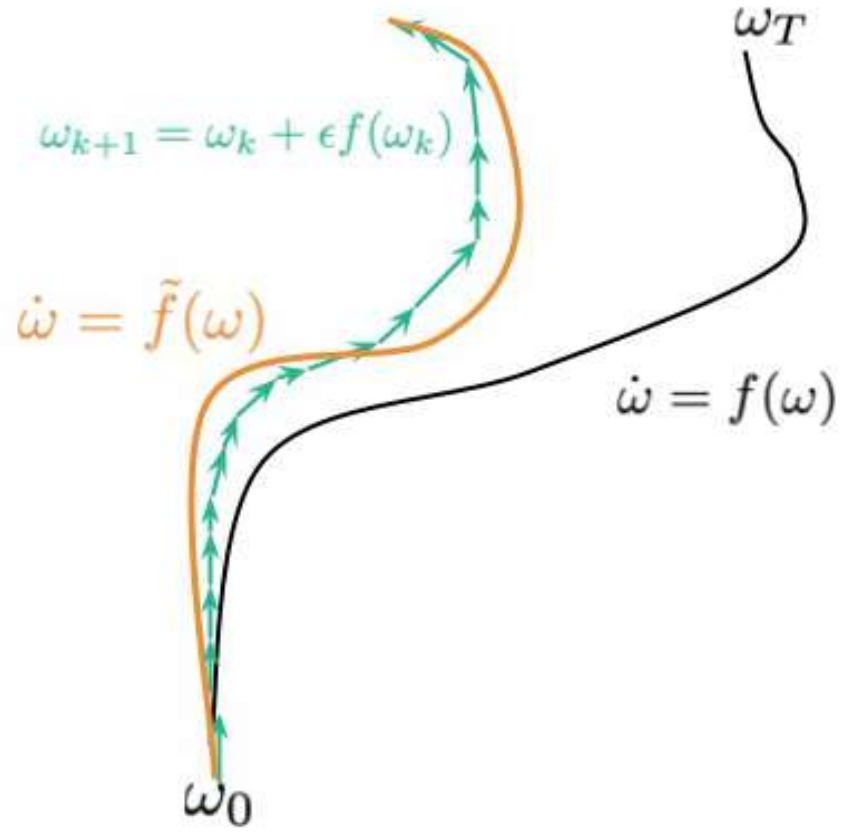
$= Din * \text{Var}(x_i) * \text{Var}(w_i)$ [Assume x, w are zero-mean]

If $\text{Var}(w_i) = 1/Din$ then $\text{Var}(y_i) = \text{Var}(x_i)$

Effectively reduce the complexity of the model!!

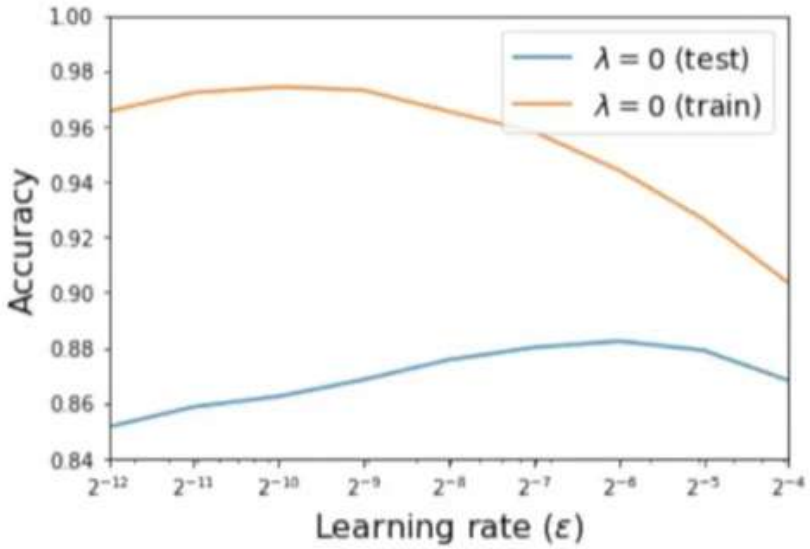
Discretization and implicit regularization

Discretization in GD and implicit regularization



Let's say we have a differential equation $\dot{\omega} = f(\omega)$. The solution to this ODE with initial condition ω_0 is a continuous trajectory ω_t, shown in the image in black. We usually can't compute this solution in closed form, and instead simulate the ODE using the Euler's method, $\omega_{k+1} = \omega_k + \epsilon f(\omega_k)$. This results in a discrete trajectory shown in teal. Due to discretization error, for finite stepsize ϵ, this discrete path may not lie exactly where the continuous black path lies. Errors accumulate over time, as shown in this illustration. The goal of backward error analysis is to find a different ODE, $\dot{\omega} = \tilde{f}(\omega)$ such that the approximate discrete path we got from Euler's method lieas near the the continuous path which solves this new ODE. Our goal is to reverse engineer a modified $\tilde{f}$ such that the discrete iteration can be well-modelled by an ODE.

Discretization in SGD and implicit regularization



Key claim (informal)

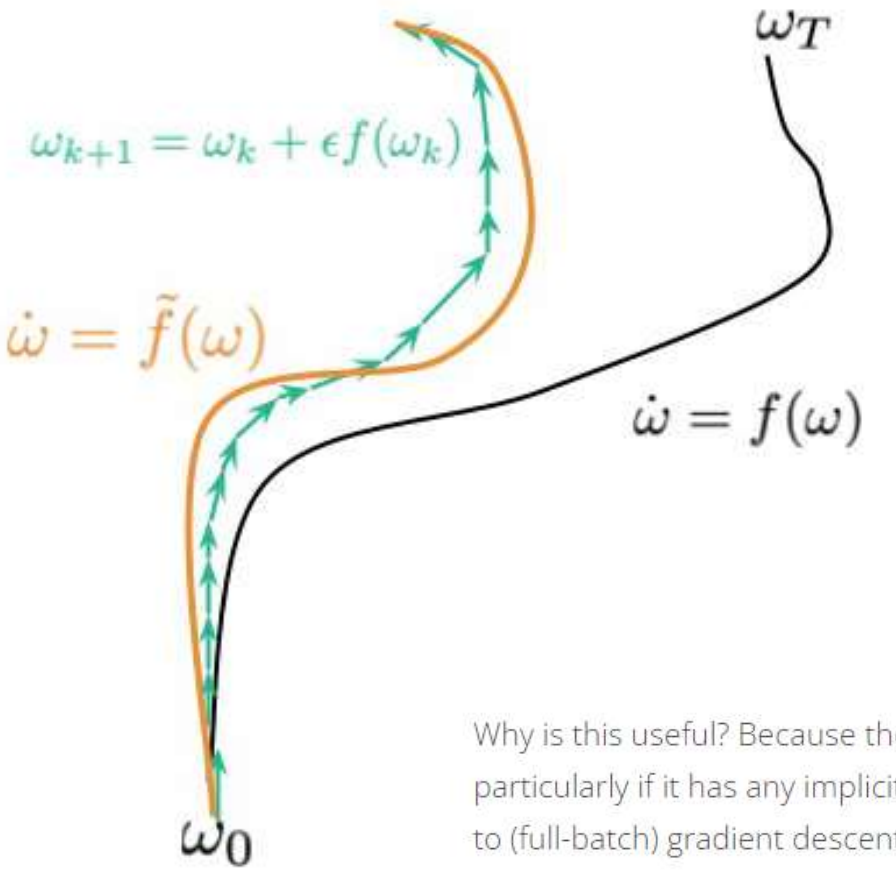
The SGD iterates stay close to the path of gradient flow on a modified loss:

$$\widetilde{C}_{SGD}(\omega) = C(\omega) + \epsilon R(\omega)$$

Implicit regularizer

For the solution to stay close to the gradient flow we need to modify the Loss!!!

See also: https://arxiv.org/pdf/2101.12176.pdf , https://www.inference.vc/notes-on-the-origin-of-implicit-regularization-in-stochastic-gradient-descent/



IMPLICIT GRADIE REGULARIZATION DUE TO DISCRETIZATION

Why is this useful? Because the form $\tilde{f}$ takes can reveal interesting aspects of the behaviour of the discrete algorithm, particularly if it has any implicit bias towards moving into different areas of the space. When the authors apply this technique to (full-batch) gradient descent, it already suggests the kind of implicit regularization bias gradient descent has.

In Gradient descent with a cost function C, the original ODE is $f(\omega) = -\nabla C(\omega)$. The modified ODE which corresponds to a finite stepsize ϵ takes the form $\dot{\omega} = -\nabla \tilde{C}_{GD}(\omega)$ where

$$\tilde{C}_{GD}(\omega) = C(\omega) + \frac{\epsilon}{4}\|\nabla C(\omega)\|^2$$

Problem Setup

Given a continuous flow represented by an ordinary differential equation (ODE) of the form:

$$\dot{\omega} = f(\omega)$$

we are interested in finding a modified loss function that accounts for the discretization error when integrating this ODE using a discrete method, like Euler's method.

Discretization Error

For simplicity, let's assume that the ODE is solved using a discrete update rule, such as the explicit Euler method:

$$\omega_{i+1} = \omega_i + \epsilon f(\omega_i)$$

where ϵ is the step size (e.g., the learning rate in gradient descent). This discrete step introduces an approximation error because the true solution of the continuous flow is not exactly reached by the discrete step.

Backward Error Analysis

The goal is to find a modified flow (and hence, a modified loss function) such that the continuous flow with this modified loss matches the discrete flow to a given order in ϵ. This means we want to express the continuous flow as:

$$\dot{\omega} = f_{\text{mod}}(\omega) = f(\omega) + \epsilon f_1(\omega) + \epsilon^2 f_2(\omega) + \dots$$

where the correction terms $f_1(\omega), f_2(\omega), \dots$ account for the discretization error at different orders of ϵ.

Derivation of the First Correction Term

We start by expanding the discrete step ω_{i+1} in a Taylor series about ω_i:

$$\omega_{i+1} = \omega_i + \epsilon\dot{\omega}_i + \frac{\epsilon^2}{2}\ddot{\omega}_i + \mathcal{O}(\epsilon^3)$$

where $\dot{\omega}_i = f(\omega_i)$ and $\ddot{\omega}_i$ is the time derivative of $\dot{\omega}_i$. Using the chain rule, $\ddot{\omega}_i$ is:

$$\ddot{\omega}_i = \frac{d}{dt}f(\omega) = \nabla f(\omega) \cdot \dot{\omega} = \nabla f(\omega) f(\omega)$$

Now we match this Taylor expansion to the discrete update $\omega_{i+1} = \omega_i + \epsilon f(\omega_i)$, which only has terms of order ϵ. To ensure consistency between the continuous and discrete flows, we need to add correction terms to the continuous flow at higher orders in ϵ.

At order ϵ^2, the correction term $f_1(\omega)$ is derived by requiring that the Taylor expansion of the continuous flow matches the discrete update. This leads to the condition:

$$f_1(\omega) + \frac{1}{2}\nabla f(\omega) f(\omega) = 0$$

Solving for $f_1(\omega)$, we obtain:

$$f_1(\omega) = -\frac{1}{2}\nabla f(\omega) f(\omega)$$

General Correction to the Loss

To interpret this result in terms of a correction to the **loss function**, we note that the flow $\dot{\omega} = f(\omega)$ is typically derived as the gradient of a loss function $C(\omega)$:

$$f(\omega) = -\nabla C(\omega)$$

Substituting this into the expression for $f_1(\omega)$, we find that the correction term becomes:

$$f_1(\omega) = -\frac{1}{2}\nabla\nabla C(\omega)\nabla C(\omega)$$

This is exactly the gradient of $\frac{1}{4}||\nabla C(\omega)||^2$, meaning the corrected flow is:

$$\dot{\omega} = -\nabla C(\omega) - \frac{\epsilon}{4}\nabla||\nabla C(\omega)||^2$$

Thus, the **corrected loss function** to match the discrete flow to the continuous flow up to order ϵ is:

$$C_{\text{mod}}(\omega) = C(\omega) + \frac{\epsilon}{4}||\nabla C(\omega)||^2$$

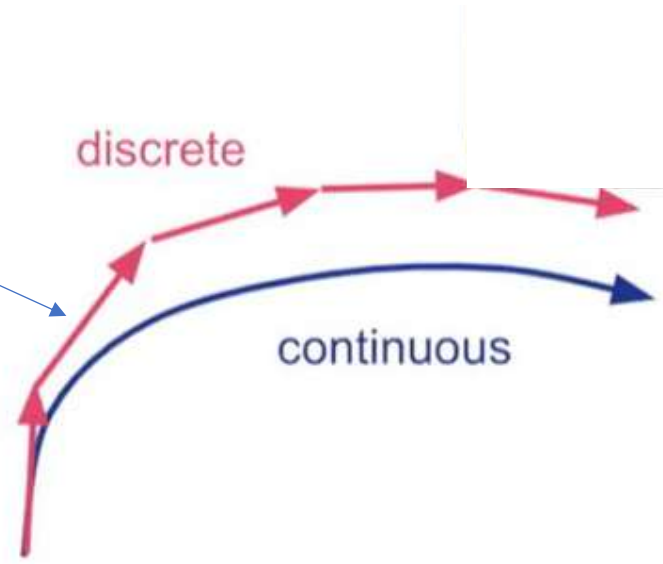
Discretization in GD and implicit regularization

Intuition:

On average, the gradient is half a step stale

Taylor expansion

$$\nabla C(\omega + (\epsilon/2)\nabla C) \approx \nabla C + (\epsilon/2)\nabla\nabla C \nabla C$$
$$= \nabla C + (\epsilon/4)\nabla\left(||\nabla C||^2\right)$$



For "small finite learning rates":

$$\dot{\omega} = -\nabla \widetilde{C}_{GD}(\omega) + O(\epsilon^2)$$
$$\widetilde{C}_{GD}(\omega) = C(\omega) + (\epsilon/4)||\nabla C(\omega)||^2$$

Original loss

Implicit regularizer

Stochasticity and implicit regularization

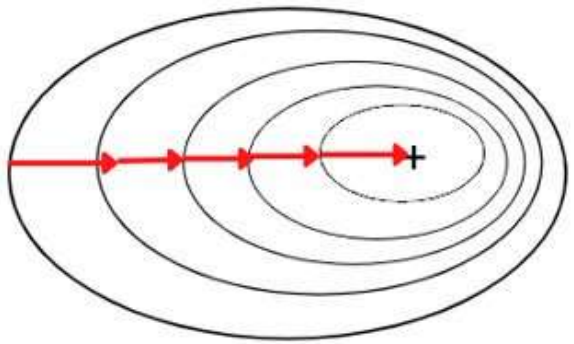
$$\mathcal{L}(w, X) = \sum_i l_i(w, x_i)$$

$$w_{t+1} = w_t - \eta \nabla_w \ell_i$$ Stochastic GD

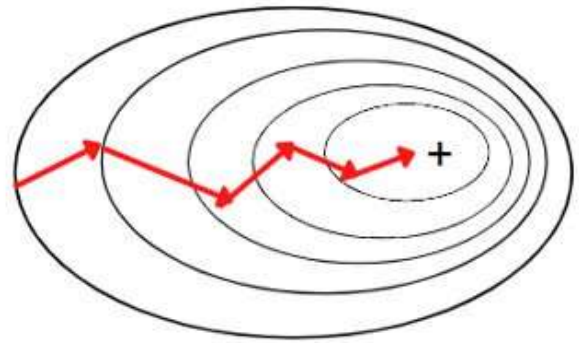
$$w_{t+1} = w_t - \eta \sum_{i \in I} \nabla_w \ell_i$$ Mini-batch GD

$$w_{t+1} = w_t - \eta \sum_i \nabla_w \ell_i$$ Full-batch GD

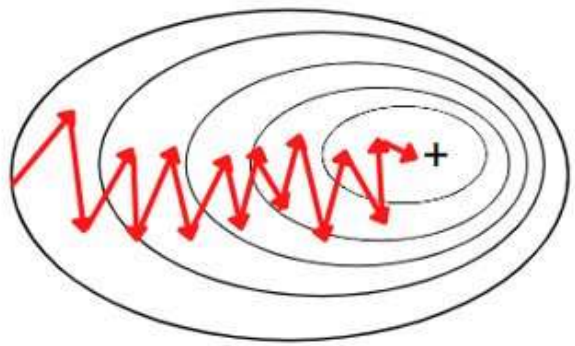
Batch Gradient Descent



Mini-Batch Gradient Descent

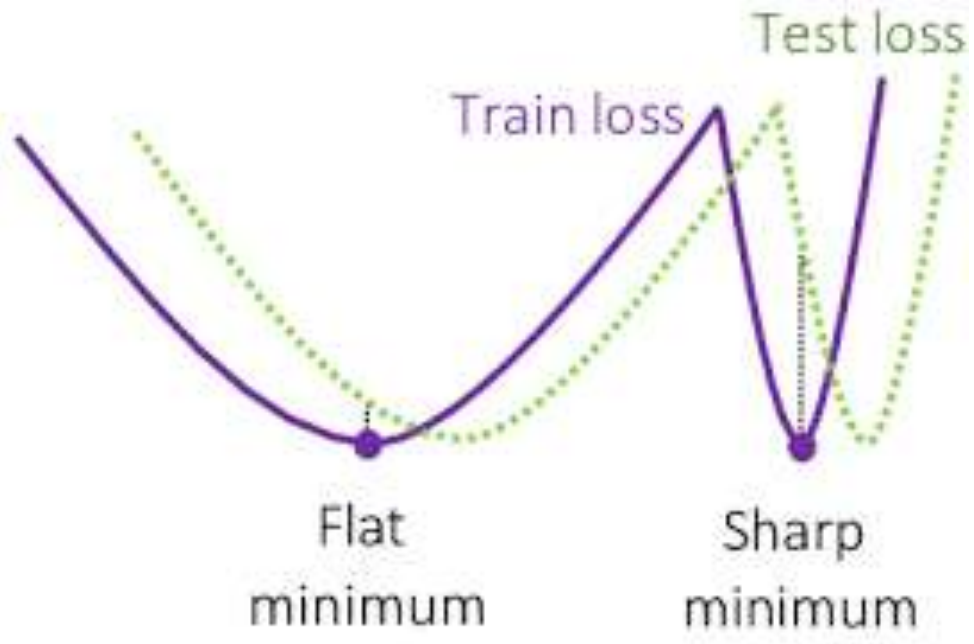


Stochastic Gradient Descent



SGD and generalization

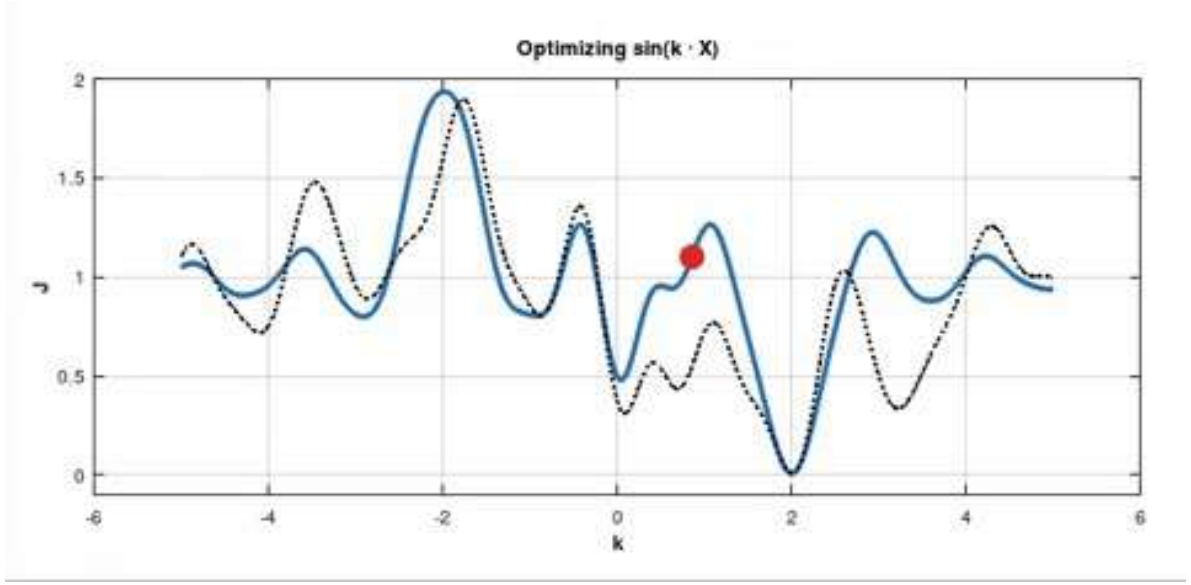
- Skip bad local minima, converge to flat minima (basins of attraction). Good for generalization, robustness.

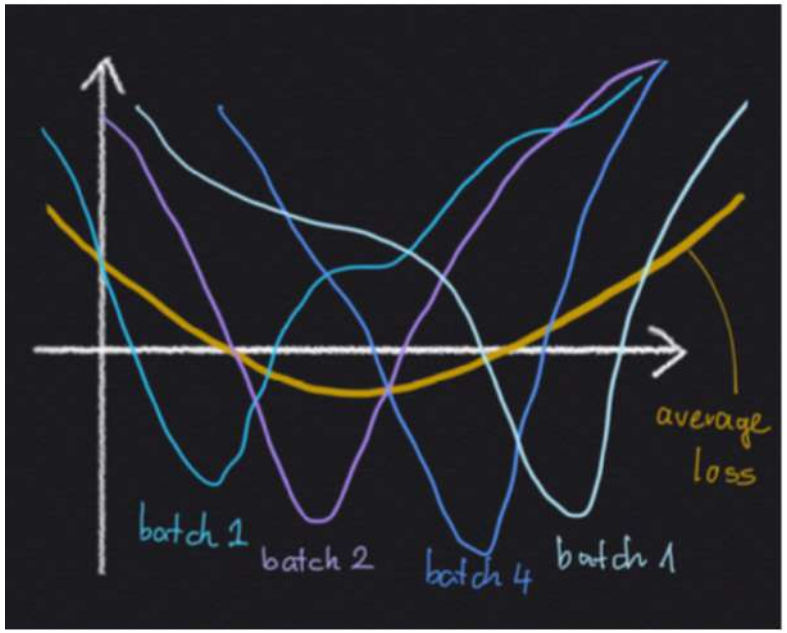


Another advantage?

Can we be more precise?

Intuition





At each step the SGD is looking to a “different, sampled” function

On average the main effect is dominated by big “basins of attraction”

Math project: adaptive learning rate, loss convexity and generalization

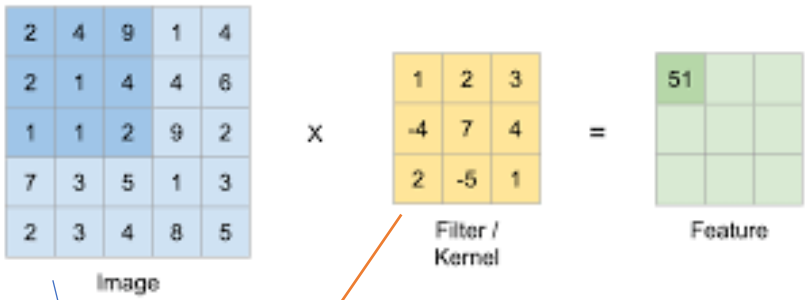
Advantages of SGD

- Good if the non-linearity is not differentiable (noisy)
- We need way less computations (proportional to 1 or Batch size)
- Converges to good minima (invariant to data perturbations)

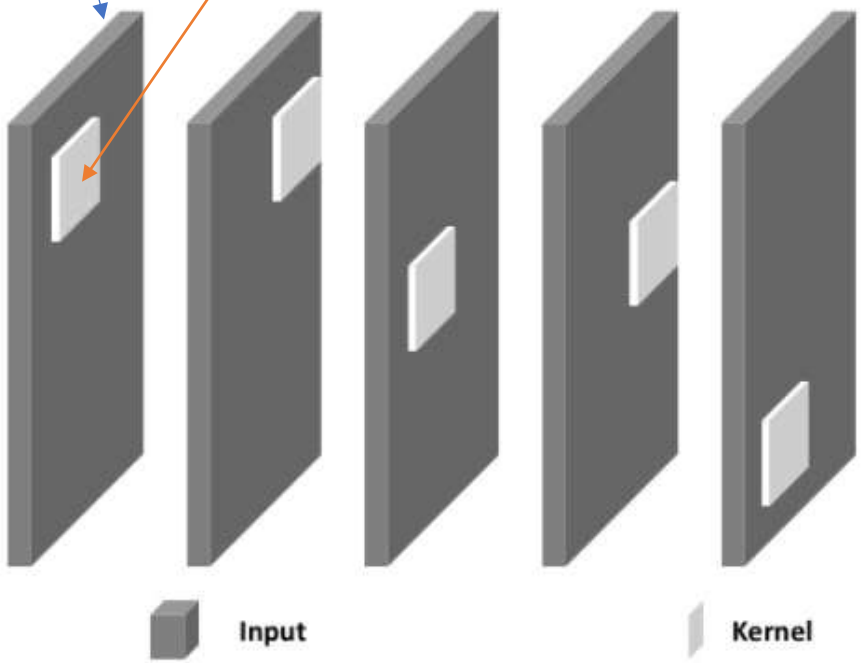
Class 4: architectural and data bias

- Convolutional networks and priors on the weight sharing
- Data augmentations and symmetries
- Depth and compositionality

Convolution operation



$$w * x$$



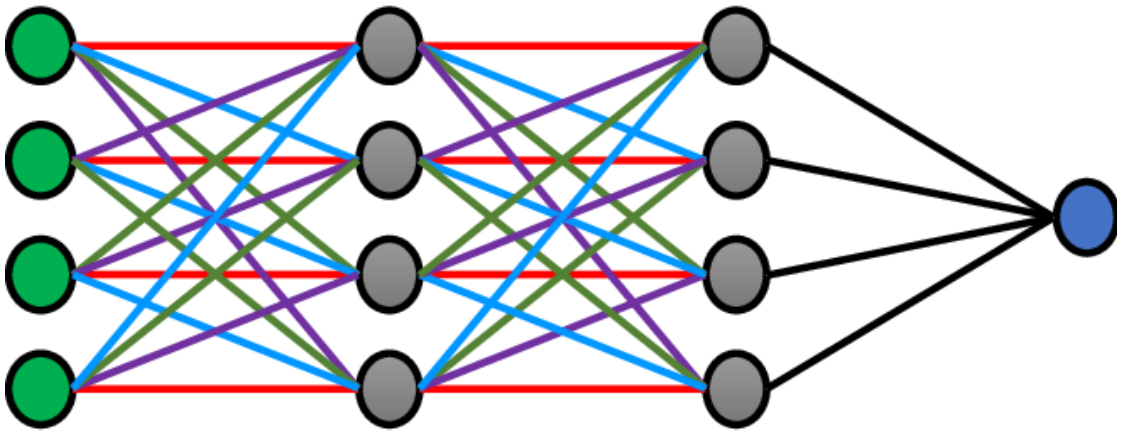
Input image

Convolution Kernel

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

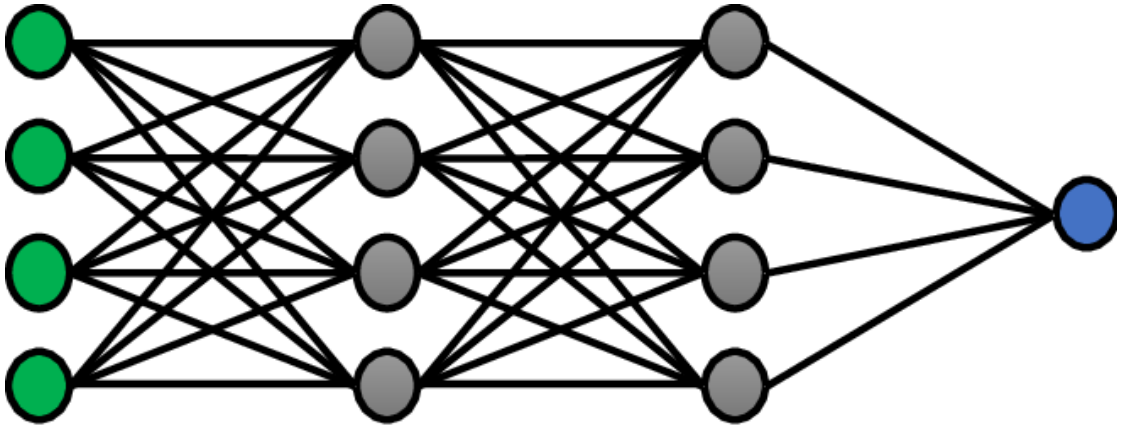
Feature map

Weights bias for fully connected and convolutions linear networks



(b) Convolutional network of depth L

$\overline{\boldsymbol{\beta}}^{\infty} \propto$ first order stationary point of $\underset{\forall n,\, y_n \langle \mathbf{x}_n, \boldsymbol{\beta} \rangle \geq 1}{\operatorname{argmin}} \|\widehat{\boldsymbol{\beta}}\|_{2/L}$

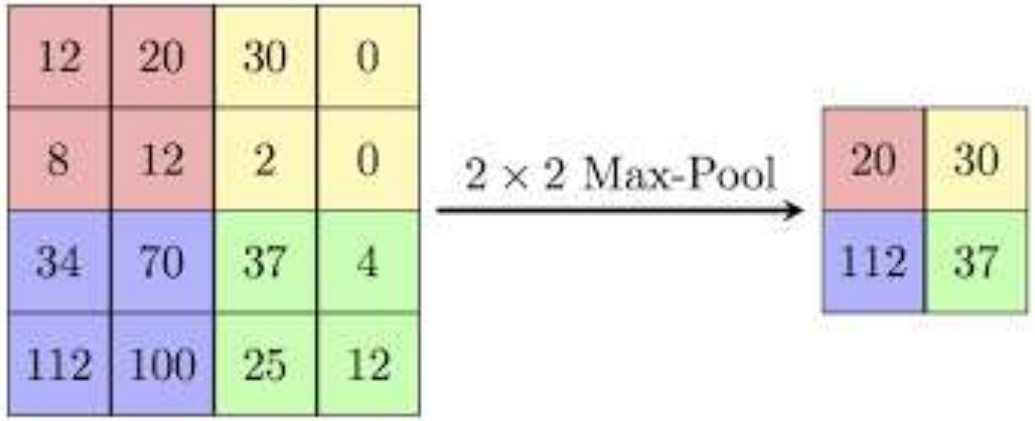


(a) Fully connected network of depth L

$\overline{\boldsymbol{\beta}}^{\infty} \propto \underset{\forall n,\, y_n \langle \mathbf{x}_n, \boldsymbol{\beta} \rangle \geq 1}{\operatorname{argmin}} \|\boldsymbol{\beta}\|_2$ (independent of L)

Gunasekar 2018

Pooling operation and invariance



Convolution operation, pooling and invariance: bias on the target function

$$\langle g_i w, x \rangle = (x * w)_i$$

$$f(x) = \sum_i \langle g_i w, x \rangle$$

$$f(gx) = f(x)$$

$$f(gx) = \sum_i \langle g_i w, gx \rangle = \sum_i \langle g^T g_i w, x \rangle = \sum_i \left\langle g_i^{'} w, x \right\rangle$$

$$g^T g_i = g_i^{'}$$

Convolution operation and equivariance

$$f(gx) = \begin{bmatrix} \sigma\left\langle g_1 w, gx \right\rangle \\ \vdots \\ \sigma\left\langle g_N w, gx \right\rangle \end{bmatrix} = \begin{bmatrix} \sigma\left\langle w, g_1^T gx \right\rangle \\ \vdots \\ \sigma\left\langle w, g_N^T gx \right\rangle \end{bmatrix} = P_g \begin{bmatrix} \sigma\left\langle w, gx \right\rangle \\ \vdots \\ \sigma\left\langle w, gx \right\rangle \end{bmatrix} = P_g f(x)$$

$$f(gx) = \begin{bmatrix} \sigma\left\langle g_1 w, gx \right\rangle \\ \vdots \\ \sigma\left\langle g_N w, gx \right\rangle \end{bmatrix} = \begin{bmatrix} \sigma\left\langle w, g_1^T gx \right\rangle \\ \vdots \\ \sigma\left\langle w, g_N^T gx \right\rangle \end{bmatrix} = P_g \begin{bmatrix} \sigma\left\langle w, gx \right\rangle \\ \vdots \\ \sigma\left\langle w, gx \right\rangle \end{bmatrix} = P_g f(x)$$

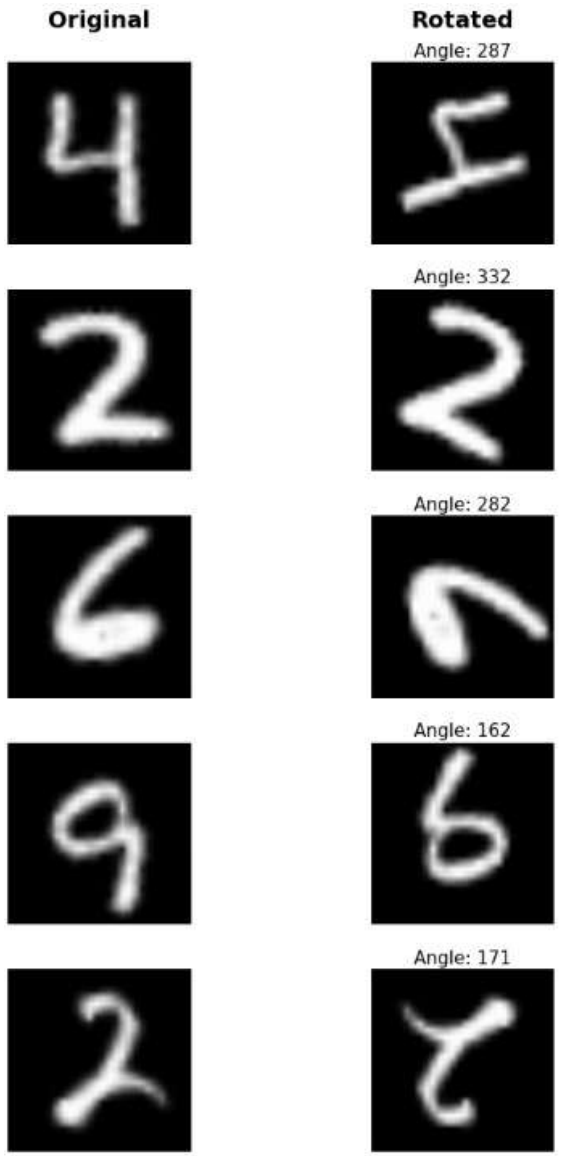
$$f(gx) = \Pi_g f(x)$$

Cnns are translation equivariant

$$\phi(T_\alpha x)(\beta) = \sigma(\int k(\beta - \gamma) T_\alpha x(\gamma) d\gamma) = T_\alpha \phi(x)(\beta)$$

Let’s prove it

What about?


Original
Rotated
Angle: 287
Angle: 332
Angle: 282
Angle: 162
Angle: 171

Equivariant and invariant networks

https://dmol.pub/dl/Equivariant.html#equivariant-neural-networks-with-constraints

Equivariance and convolutionality are one to one:

On the Generalization of Equivariance and Convolution in Neural Networks to the Action of Compact Groups

Risi Kondor [1] **Shubhendu Trivedi** [2]

Abstract

Convolutional neural networks have been extremely successful in the image recognition domain because they ensure equivariance to translations. There have been many recent attempts to generalize this framework to other domains, including graphs and data lying on manifolds. In this paper we give a rigorous, theoretical treatment of convolution and equivariance in neural networks with respect to not just translations, but the action of any compact group. Our main result is to prove that (given some natural constraints) convolutional structure is not just a sufficient, but also a necessary condition for equivariance to the action of a compact group. Our exposition makes use of concepts from representation theory and noncommutative harmonic analysis and derives new generalized convolution formulae.

Recently, there has been considerable interest in extending neural networks to more exotic types of data, such as graphs or functions on manifolds (Niepert et al., 2016; Defferrard et al., 2016; Duvenaud et al., 2015; Li et al., 2016; Cohen et al., 2018; Monti et al., 2017; Masci et al., 2015). In these domains, equivariance and multiscale structure are just as important as for images, but finding the right notion of convolution is not obvious.

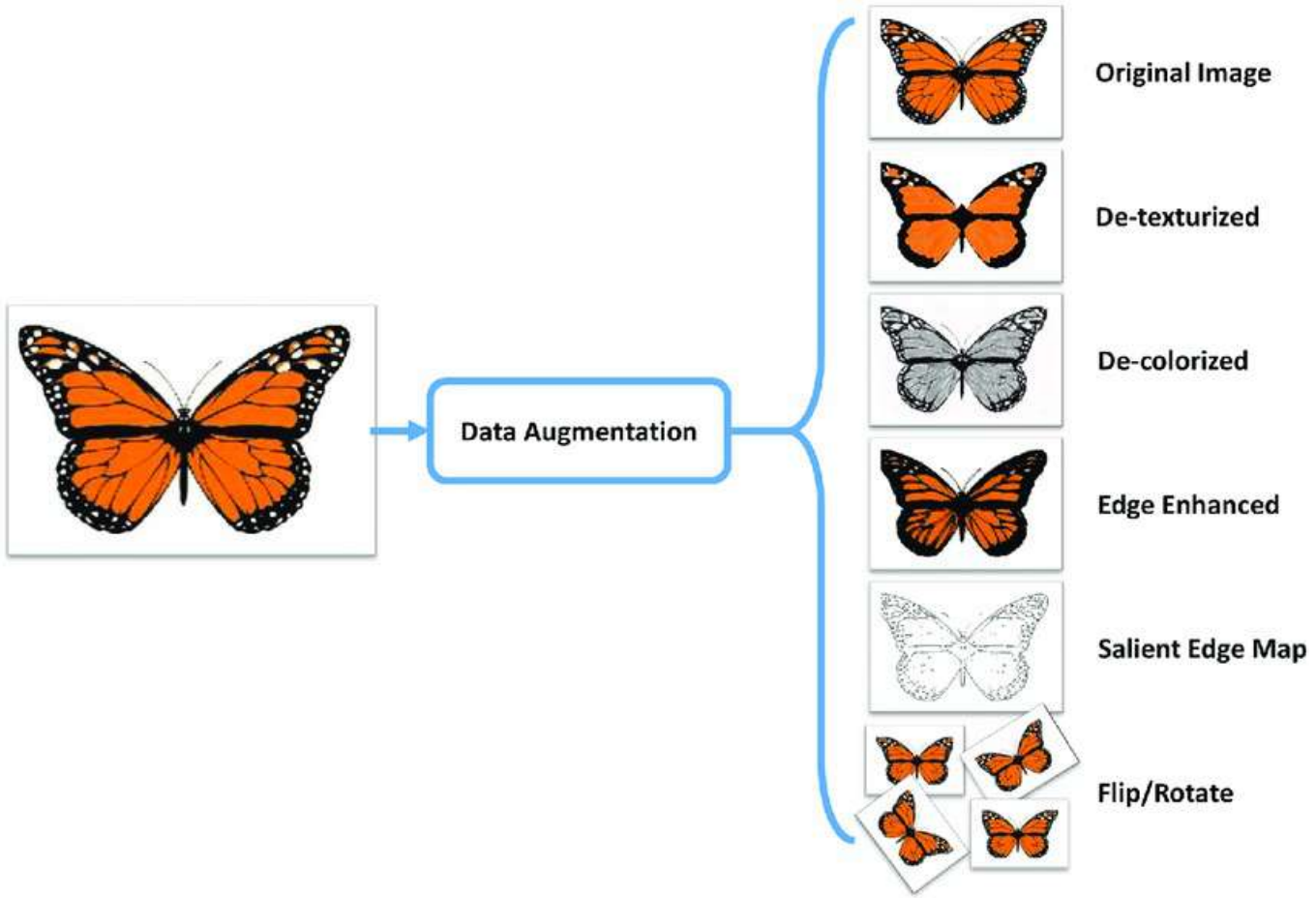
On the other hand, mathematics does offer a sweeping generalization of convolution tied in deeply with some fundamental ideas of abstract algebra: if G is a compact group and f and g are two functions $G \to \mathbb{C}$, then the convolution of f with g is defined

$$(f * g)(u) = \int_G f(uv^{-1})\, g(v)\, d\mu(v). \qquad (1)$$

Note the striking similarity of this formula to the ordinary notion of convolution, except that in the argument of f, $u - v$ has been replaced by the group operation uv^{-1}, and

How do we learn invariance or equivariance w.r.t. generic label-preserving transformations?

Data Augmentation



Effect of Data Augmentation: invariant loss for one layer network

$$\mathcal{L}(w) = \frac{1}{N}\sum_{i=1}^{N}\int \ell\left(\sigma\langle w, g_\theta x_i\rangle; y_i\right) d\theta$$

$$\begin{aligned}\mathcal{L}(g_{\bar{\theta}}w) &= \frac{1}{N}\sum_{i=1}^{N}\int \ell\left(\sigma\langle w, g^* g_\theta x_i\rangle; y_i\right) d\theta \\ &= \sum_{i=1}^{N}\int \ell\left(\sigma\langle w, g_\theta x_i\rangle; y_i\right) d\theta \\ &= \mathcal{L}(w)\end{aligned}$$

The Loss is an invariant function!

$$g_\theta g_{\theta'} = g_{\theta''}$$

Intuition with translation invariant/robust functions

- **Invariance:** Let $\underline{\mathrm{f}} : R \to R$ s.t. $\underline{\mathrm{f}}(w+q) = \underline{\mathrm{f}}(w), \ \ \forall q \in R.$ $\underline{\mathrm{f}}$ is constant and $\hat{\underline{\mathrm{f}}}(0)$ is the only non zero Fourier transform.

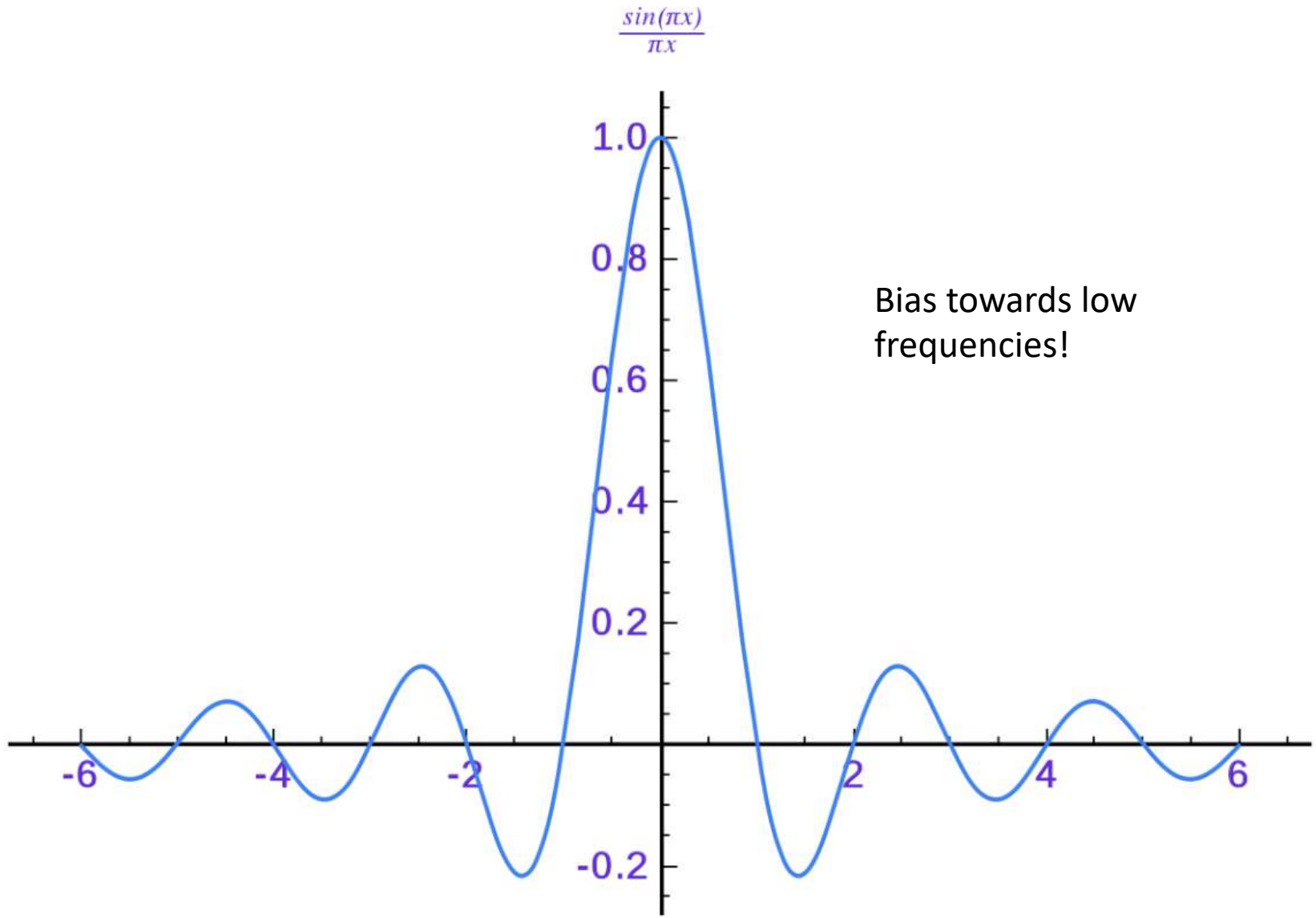
$$\underline{\mathrm{f}}(w) = \int_{-\infty}^{\infty} dt\ f(w-t), \qquad f : R \to R.$$

- **Robustness:** Suppose we relax this invariance properties asking for robustness/approximate invariance. One way is to integrate in the interval $[-a, a]$.

$$\underline{\mathrm{f}}(w) = \int_{-a}^{a} dt\ f(w-t) = \int_{-\infty}^{+\infty} dt\ Ind_{[-a,a]}(t) f(w-t).$$

Taking the Fourier and using $\widehat{f(\cdot - t)} = e^{ikt}\hat{f}(k)$

$$\hat{\underline{\mathrm{f}}}(k) = \Big(\int_{-\infty}^{\infty} dt\ e^{ikt} Ind_{[-a,a]}(t) \Big)\ \hat{f}(k) = 2a\ sinc(2ka)\hat{f}(k).$$



Bias towards low frequencies!

Effect on the gradient descent

From the group structure we have:

$$\mathcal{L}(U_\alpha w; X, y) = \mathcal{L}(w; X, y), \quad \alpha \in R.$$

Then

$$\frac{d}{d\alpha}\mathcal{L}(U_\alpha w) = 0 \;\; \rightarrow \;\; \langle \nabla_w \mathcal{L}(w), \partial_\alpha U_\alpha w \rangle = 0,$$

i.e. **some directions in the gradient are not allowed by the invariances of the Loss**.

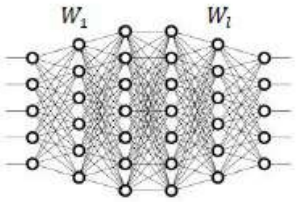
This is important since:

$$\frac{dw}{dt} = \nabla \mathcal{L}(w).$$

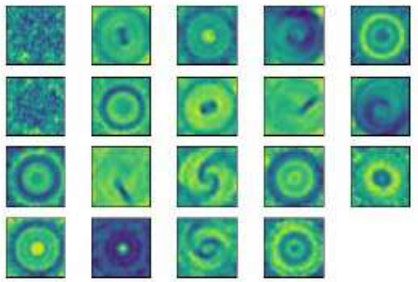
Learned weights reflects the symmetries of the data augmentation

Goal: $y_i = \varphi_W(x_i), \quad \varphi_{\mathbf{W}}(\mathbf{g}\mathbf{x_i}) = \varphi_{\mathbf{W}}(\mathbf{x_i})$

Data augmentation:



Rotation augmented



$$\mathcal{L}(w) = \frac{1}{N}\sum_{i=1}^{N}\int \ell\left(\sigma\langle w, g_\theta x_i\rangle; y_i\right) d\theta$$

Equivariance and learned weights

Theorem 1 (see also, [43]): Consider the Loss in eq. (5). The gradient of the Loss is an equivariant function in the $\mathcal{G}$ transformations, i.e.:

$$\nabla_W \mathcal{L}_a(gW; X, y) = g\nabla_W \mathcal{L}_a(W; X, y), \quad \forall\, g \in \mathcal{G}.$$

Proof 1: Taking the gradient of eq. (5) we have:

$$\begin{aligned}
&\nabla_W \mathcal{L}_a(W; X, y) = \\
&= \frac{1}{Q}\sum_{i=1}^{R}\sum_{j=1}^{|\mathcal{G}|} \ell'(v^T\sigma\left\langle W, g_j x_i\right\rangle, y_i) v^T\sigma'\left\langle W, g_j x_i\right\rangle g_j x_i
\end{aligned}$$

Calculating $\nabla_W \mathcal{L}_a(gW; X, y)$ we have:

$$\begin{aligned}
&\nabla_W \mathcal{L}_a(gW; X, y) = \\
&= \frac{1}{Q}\sum_{i=1}^{R}\sum_{j=1}^{|\mathcal{G}|} \ell'(v^T\sigma\left\langle gW, g_j x_i\right\rangle, y_i) v^T\sigma'\left\langle gW, g_j x_i\right\rangle g_j x_i \\
&= \frac{1}{Q}\sum_{i=1}^{R}\sum_{j=1}^{|\mathcal{G}|} \ell'(v^T\sigma\left\langle W, g^T g_j x_i\right\rangle, y_i) v^T\sigma'\left\langle W, g^T g_j x_i\right\rangle g_j x_i \\
&= \frac{1}{Q}\sum_{i=1}^{R}\sum_{j=1}^{|\mathcal{G}|} \ell'(v^T\sigma\left\langle W, \hat{g}_j x_i\right\rangle, y_i) v^T\sigma'\left\langle W, \hat{g}_j x_i\right\rangle g\hat{g}_j x_i = \\
&= g\nabla_W \mathcal{L}_a(W; X, y)
\end{aligned}$$

where in the fourth line we redefined $g^T g_j = \hat{g}_j$ to get $g_j = g\hat{g}_j$. The key property here is the closure of the group transformations. ■

Equivariance and learned weights

Corollary 1: Consider the Loss in eq. 5 i.e. the case of data augmentation with full orbits, i.e. $gX = X$ for all $g \in \mathcal{G}$, and suppose we initialize the network with symmetric weights i.e. $gW_0 = W_0$ for all $g \in \mathcal{G}$. Then the learned weights are symmetric $W_T = gW_T$.

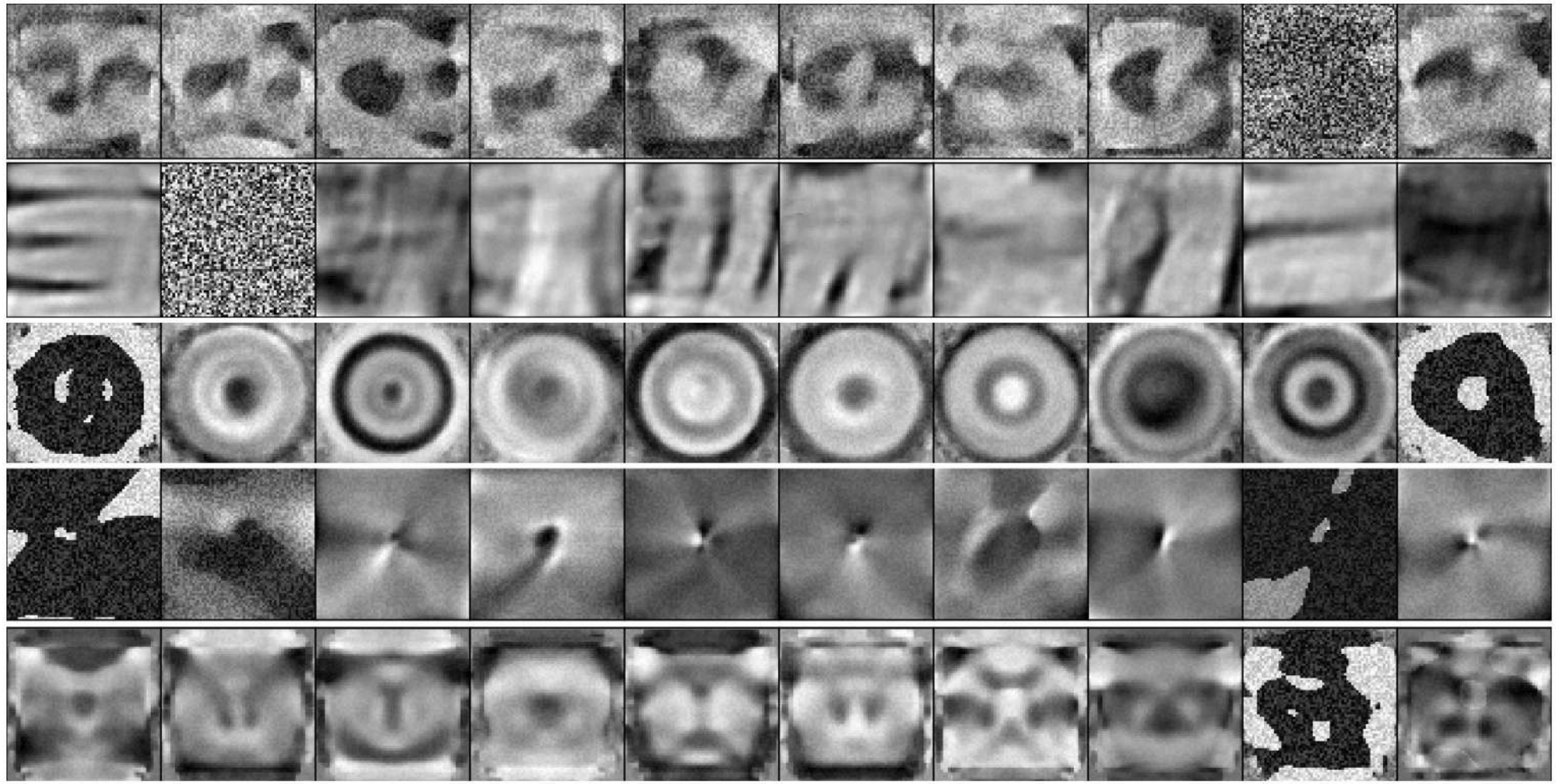
Proof 2: We first notice that due to the property $gX = X$, $\nabla_W \mathcal{L}_a(W; X, y) = \nabla_W \mathcal{L}_a(W; gX, y)$. Then by the dot product structure of the Loss we can see the transformation as applied to the weights i.e. $\langle W, gx \rangle = \langle g^T W, x \rangle$. Thus, using the equivariance property in theorem 1, we have that for all $g \in \mathcal{G}$:

$$\begin{aligned} \nabla_W \mathcal{L}_a(W; X, y) &= \nabla_W \mathcal{L}_a(W; gX, y) \\ &= \nabla_W \mathcal{L}_a(g^T W; X, y) \\ &= g^T \nabla_W \mathcal{L}_a(W; gX, y). \end{aligned}$$

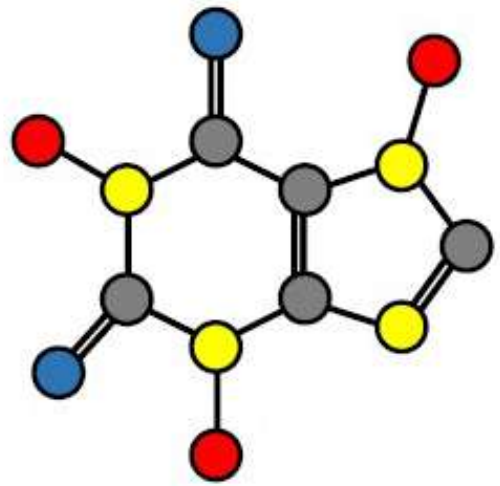
Therefore each weights update in the gradient descent is group invariant and as a consequence the learned weight at time T is invariant, being:

$$W_T = W_0 - \gamma \sum_{k=1}^{T} \nabla_W \mathcal{L}_a(W_k; X, y).$$

■



Permutation equivariant layer



Molecules

Permutation equivariant layer

Our goal is to design neural network layers that are equivariant to permutations of elements in the input $\mathbf{x}$. The function $\mathbf{f} : \mathcal{X}^M \to \mathcal{Y}^M$ is **equivariant** to the permutation of its inputs iff

$$\mathbf{f}(\pi\mathbf{x}) = \pi\mathbf{f}(\mathbf{x}) \quad \forall\pi \in \mathcal{S}_M$$

where the symmetric group $\mathcal{S}_M$ is the set of all permutation of indices $1, \ldots, M$.

Consider the standard neural network layer

$$\mathbf{f}_\Theta(\mathbf{x}) \doteq \boldsymbol{\sigma}(\Theta\mathbf{x}) \quad \Theta \in \mathbb{R}^{M\times M} \tag{20}$$

where Θ is the weight vector and $\sigma : \mathbb{R} \to \mathbb{R}$ is a nonlinearity such as sigmoid function. The following lemma states the necessary and sufficient conditions for permutation-equivariance in this type of function.

Lemma 3 *The function $\mathbf{f}_\Theta : \mathbb{R}^M \to \mathbb{R}^M$ as defined in (20) is permutation equivariant if and only if all the off-diagonal elements of Θ are tied together and all the diagonal elements are equal as well. That is,*

$$\Theta = \lambda\mathbf{I} + \gamma\,(\mathbf{1}\mathbf{1}^\mathsf{T}) \qquad \lambda, \gamma \in \mathbb{R} \quad \mathbf{1} = [1, \ldots, 1]^\mathsf{T} \in \mathbb{R}^M$$

where $\mathbf{I} \in \mathbb{R}^{M\times M}$ is the identity matrix.

Deep Sets , Zaheer et al.
NIPS 2017

1. **Goal**: To show that the matrix $\Theta = \lambda I + \gamma(11^T)$ commutes with all permutation matrices $\pi \in S_M$, ensuring permutation equivariance of the function $f_\Theta(x) = \sigma(\Theta x)$.
2. **Linearity of Commute**:
    - The linear combination $\Theta = \lambda I + \gamma(11^T)$ commutes with any permutation matrix π because both the identity matrix I and the constant matrix 11^T commute with any permutation matrix.
3. **Diagonal Elements are Identical**:
    - For any transposition $\pi_{k,l}$, it is shown that the diagonal elements $\Theta_{i,i}$ must be identical for all i, implying $\Theta_{i,i} = \lambda$.
4. **Off-Diagonal Elements are Identical**:
    - By considering pairs of off-diagonal elements, it is shown that the off-diagonal elements $\Theta_{i,j}$ must be identical, concluding that $\Theta_{i,j} = \gamma$ for all off-diagonal $i \neq j$.
5. **Conclusion**: The necessary and sufficient conditions for permutation equivariance are met, as $\Theta = \lambda I + \gamma(11^T)$ commutes with all permutation matrices $\pi \in S_M$.

From definition of permutation equivariance $\mathbf{f}_\Theta(\pi\mathbf{x}) = \pi\mathbf{f}_\Theta(\mathbf{x})$ and definition of $\mathbf{f}$ in (20), the condition becomes $\boldsymbol{\sigma}(\Theta\pi\mathbf{x}) = \pi\boldsymbol{\sigma}(\Theta\mathbf{x})$, which (assuming sigmoid is a bijection) is equivalent to $\Theta\pi = \pi\Theta$. Therefore we need to show that the necessary and sufficient conditions for the matrix $\Theta \in \mathbb{R}^{M\times M}$ to commute with all permutation matrices $\pi \in \mathcal{S}_M$ is given by this proposition. We prove this in both directions:

- To see why $\Theta = \lambda\mathbf{I} + \gamma\ (\mathbf{1}\mathbf{1}^\mathsf{T})$ commutes with any permutation matrix, first note that commutativity is linear – that is

  $$\Theta_1\pi = \pi\Theta_1 \wedge \Theta_2\pi = \pi\Theta_2 \quad \Rightarrow \quad (a\Theta_1 + b\Theta_2)\pi = \pi(a\Theta_1 + b\Theta_2).$$

  Since both Identity matrix $\mathbf{I}$, and constant matrix $\mathbf{1}\mathbf{1}^\mathsf{T}$, commute with any permutation matrix, so does their linear combination $\Theta = \lambda\mathbf{I} + \gamma\ (\mathbf{1}\mathbf{1}^\mathsf{T})$.

- We need to show that in a matrix Θ that commutes with "all" permutation matrices
  - *All diagonal elements are identical*: Let $\pi_{k,l}$ for $1 \leq k, l \leq M, k \neq l$, be a transposition (*i.e.* a permutation that only swaps two elements). The inverse permutation matrix of $\pi_{k,l}$ is the permutation matrix of $\pi_{l,k} = \pi_{k,l}^\mathsf{T}$. We see that commutativity of Θ with the transposition $\pi_{k,l}$ implies that $\Theta_{k,k} = \Theta_{l,l}$:

    $$\pi_{k,l}\Theta = \Theta\pi_{k,l} \ \Rightarrow\ \pi_{k,l}\Theta\pi_{l,k} = \Theta \ \Rightarrow\ (\pi_{k,l}\Theta\pi_{l,k})_{l,l} = \Theta_{l,l} \ \Rightarrow\ \Theta_{k,k} = \Theta_{l,l}$$

    Therefore, π and Θ commute for any permutation π, they also commute for any transposition $\pi_{k,l}$ and therefore $\Theta_{i,i} = \lambda\, \forall i$.
  - *All off-diagonal elements are identical*: We show that since Θ commutes with any product of transpositions, any choice two off-diagonal elements should be identical. Let (i, j) and (i', j') be the index of two off-diagonal elements (*i.e.* $i \neq j$ and $i' \neq j'$). Moreover for now assume $i \neq i'$ and $j \neq j'$. Application of the transposition $\pi_{i,i'}\Theta$, swaps the rows i, i' in Θ. Similarly, $\Theta\pi_{j,j'}$ switches the j^{th} column with j'^{th} column. From commutativity property of Θ and $\pi \in \mathcal{S}_n$ we have

    $$\begin{aligned}
    &\pi_{j',j}\pi_{i,i'}\Theta = \Theta\pi_{j',j}\pi_{i,i'} \ \Rightarrow \pi_{j',j}\pi_{i,i'}\Theta(\pi_{j',j}\pi_{i,i'})^{-1} = \Theta \qquad\qquad \Rightarrow \\
    &\pi_{j',j}\pi_{i,i'}\Theta\pi_{i',i}\pi_{j,j'} = \Theta \ \Rightarrow (\pi_{j',j}\pi_{i,i'}\Theta\pi_{i',i}\pi_{j,j'})_{i,j} = \Theta_{i,j} \quad \Rightarrow\ \Theta_{i',j'} = \Theta_{i,j}
    \end{aligned}$$

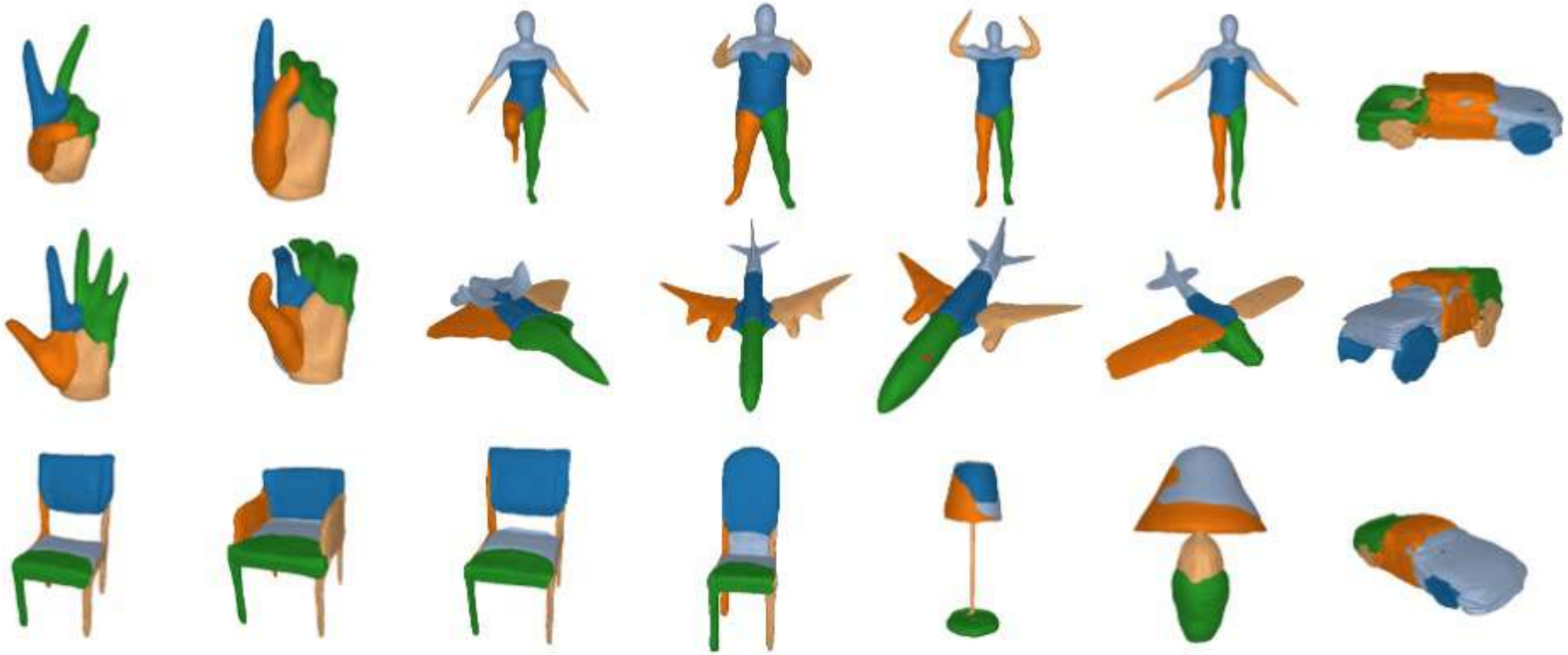
    where in the last step we used our assumptions that $i \neq i'$, $j \neq j'$, $i \neq j$ and $i' \neq j'$. In the cases where either $i = i'$ or $j = j'$, we can use the above to show that $\Theta_{i,j} = \Theta_{i'',j''}$ and $\Theta_{i',j'} = \Theta_{i'',j''}$, for some $i'' \neq i, i'$ and $j'' \neq j, j'$, and conclude $\Theta_{i,j} = \Theta_{i',j'}$.

How would you design a new layer that is equivariant wrt a group of stransformations?

Compositionality prior

- What is the computational advantage?

Possible project: Anns with notion of parts and interpretable parts



Capsules, Hinton

Possible learning strategy: whole is a (“simple”) sum of the parts. Equivariance.

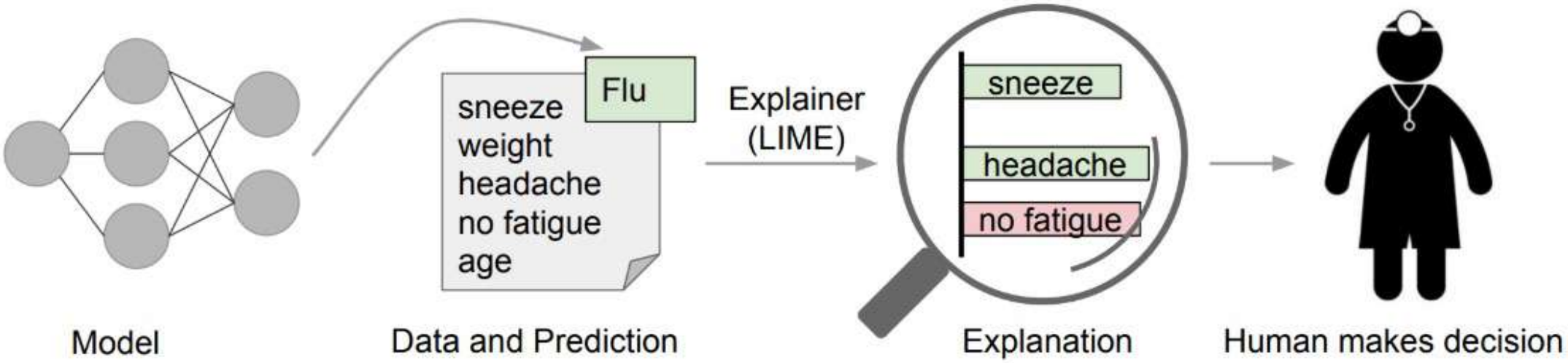
Tensor flow playground

- https://playground.tensorflow.org/#activation=tanh&batchSize=10&dataset=circle&regDataset=reg-plane&learningRate=0.03&regularizationRate=0&noise=0&networkShape=4,2&seed=0.31138&showTestData=false&discretize=false&percTrainData=50&x=true&y=true&xTimesY=false&xSquared=false&ySquared=false&cosX=false&sinX=false&cosY=false&sinY=false&collectStats=false&problem=classification&initZero=false&hideText=false

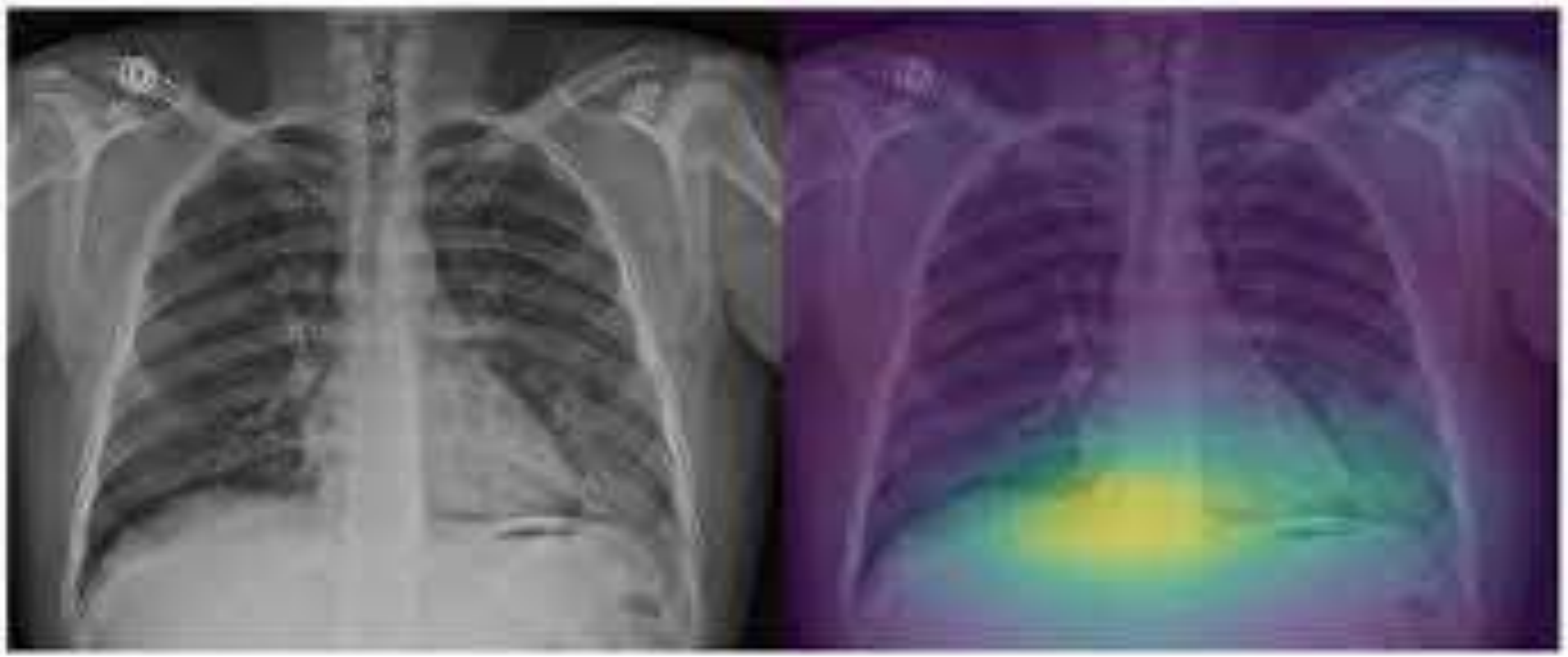
Class 5:

- NNs interpretability
- Short intro to generative models

Why interpretability is important?

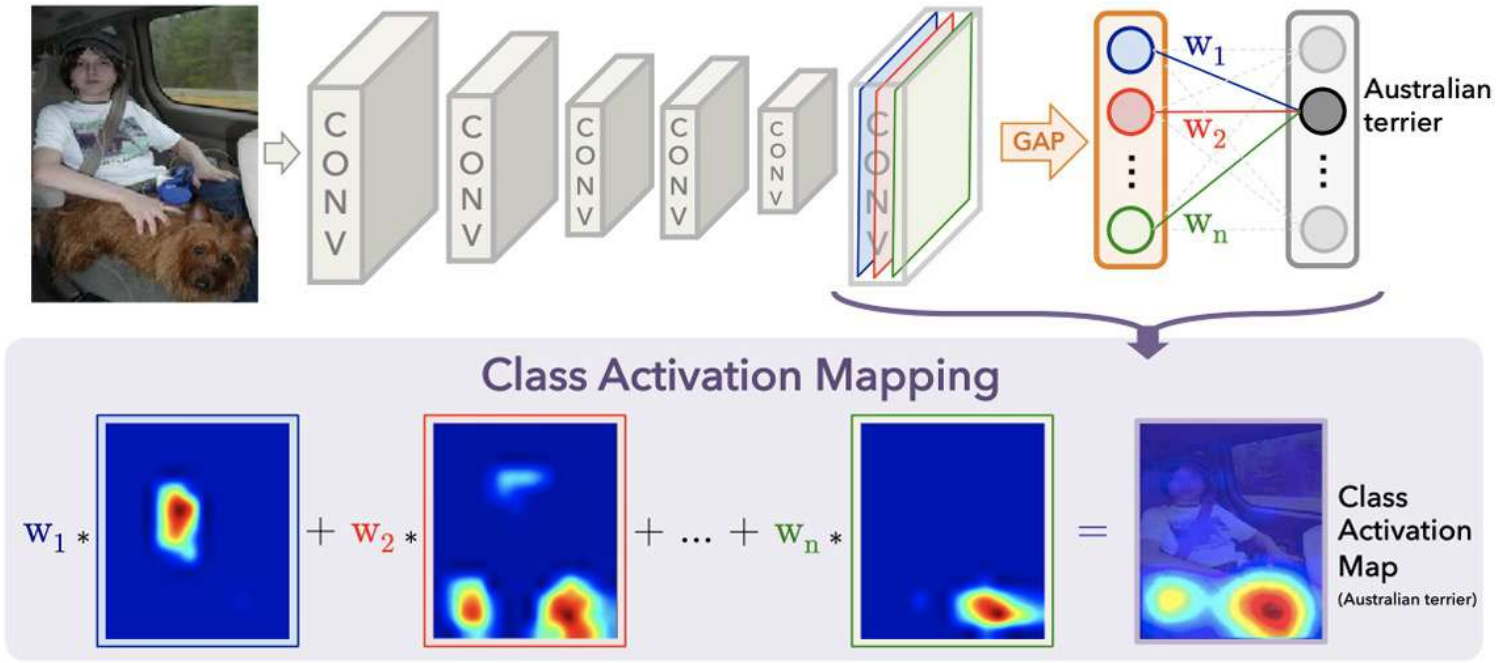


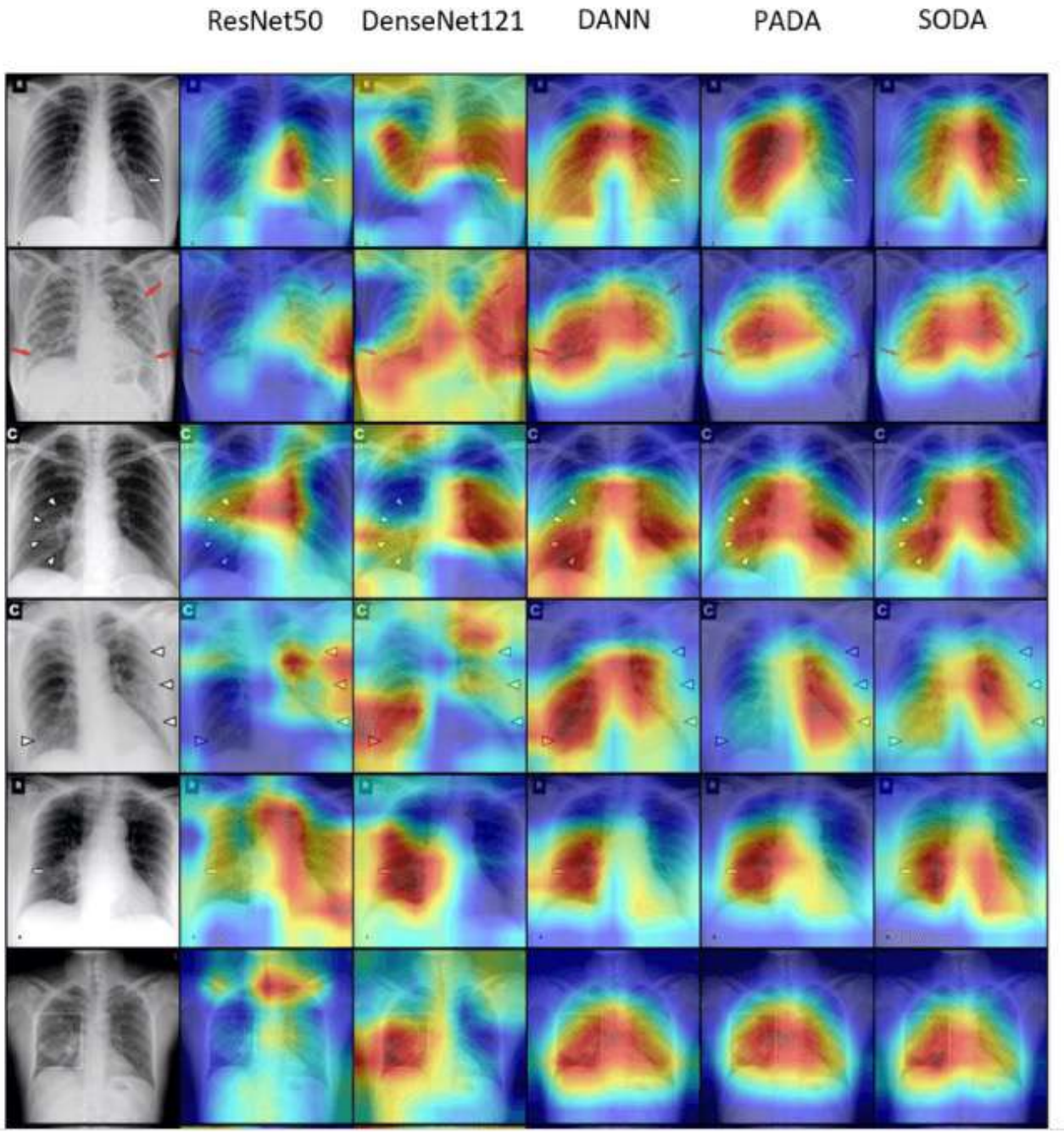
Why interpretability is important?



How are these maps calculated?

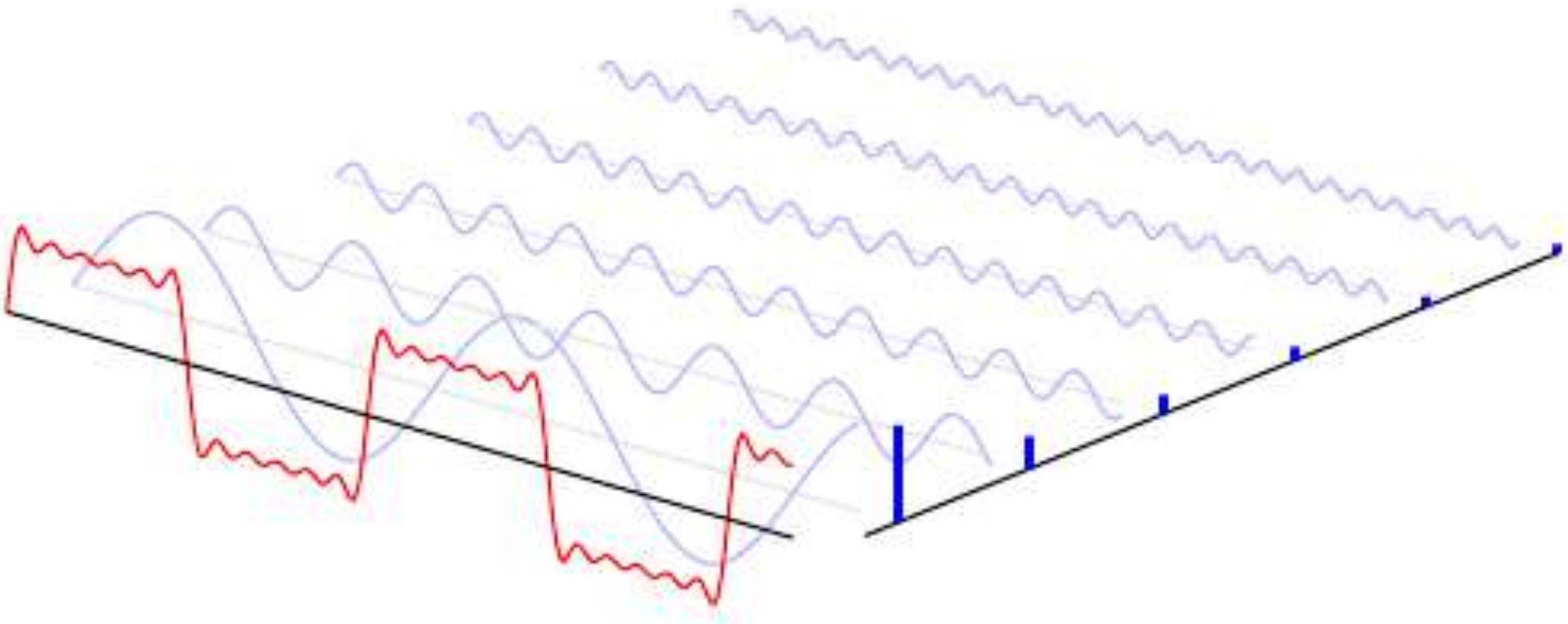
Where is a ANN looking at? Class activation mapping



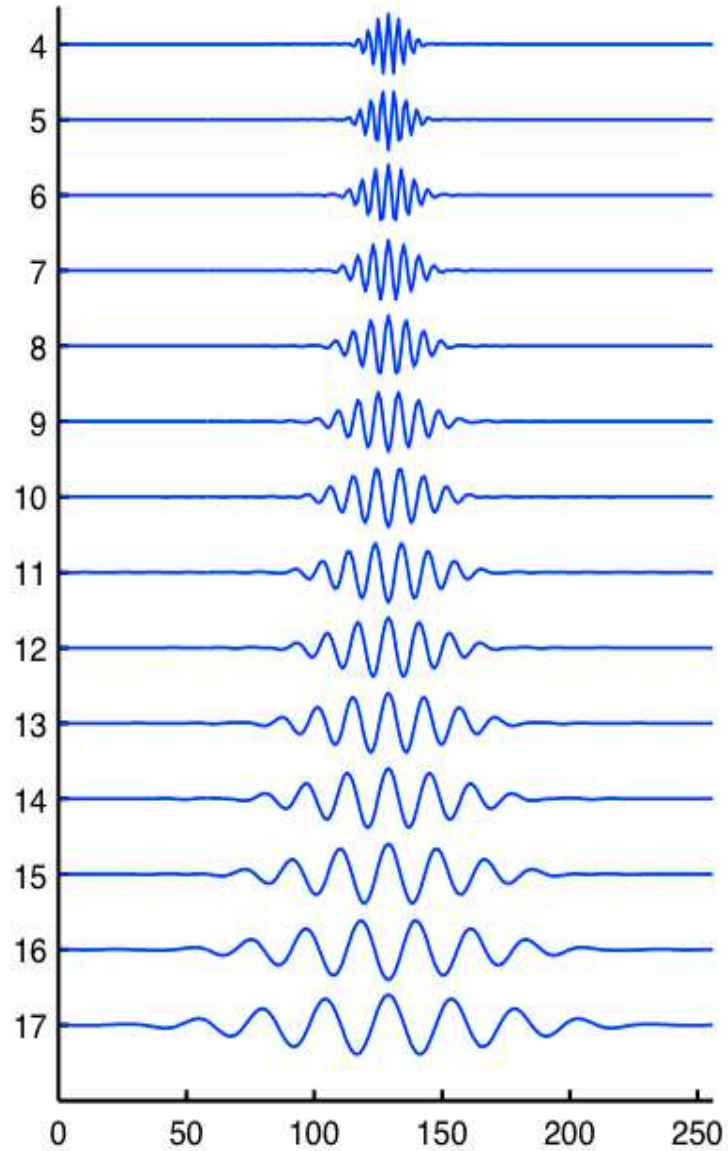


What is your conclusion?

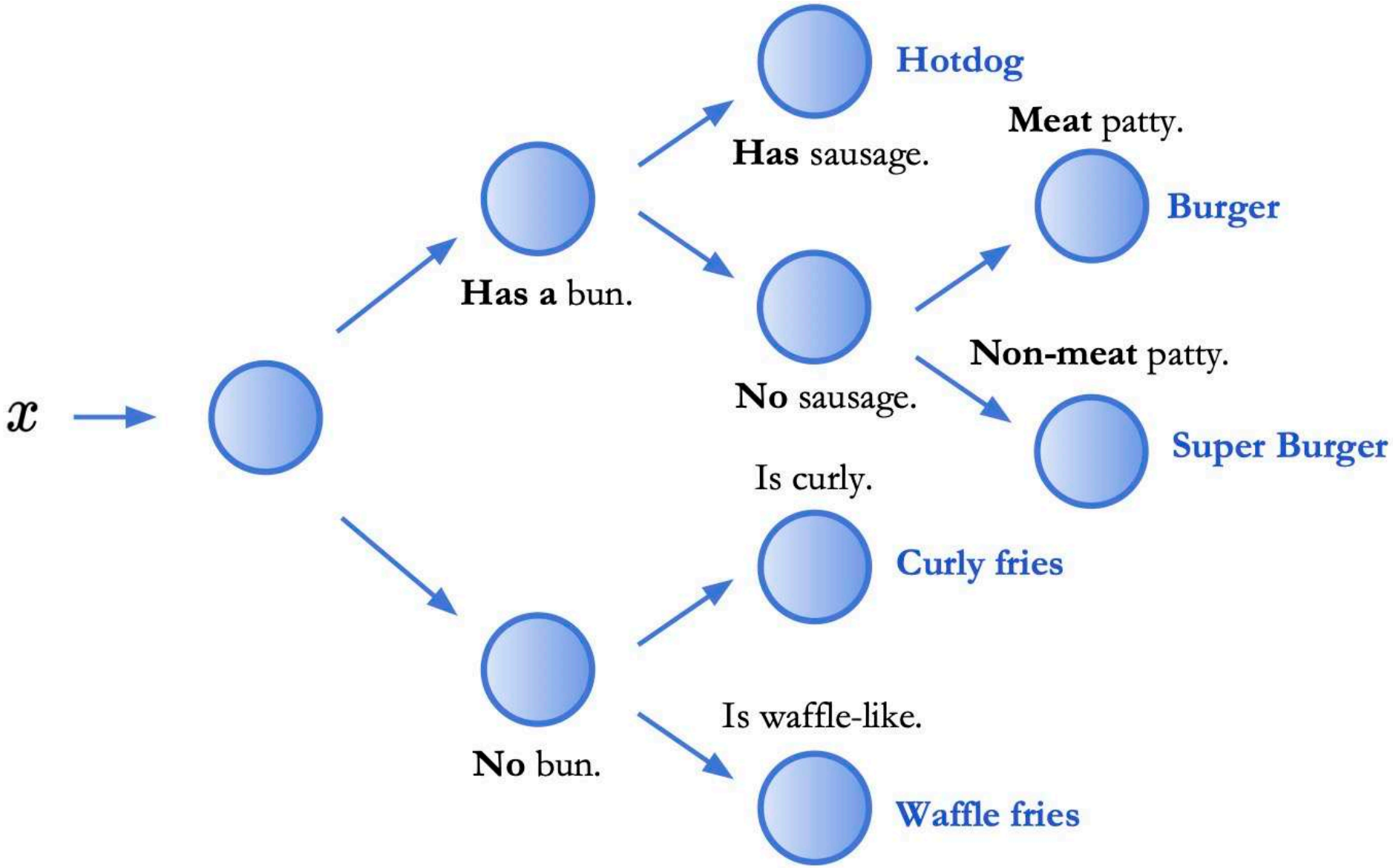
Designed features: Fourier transform



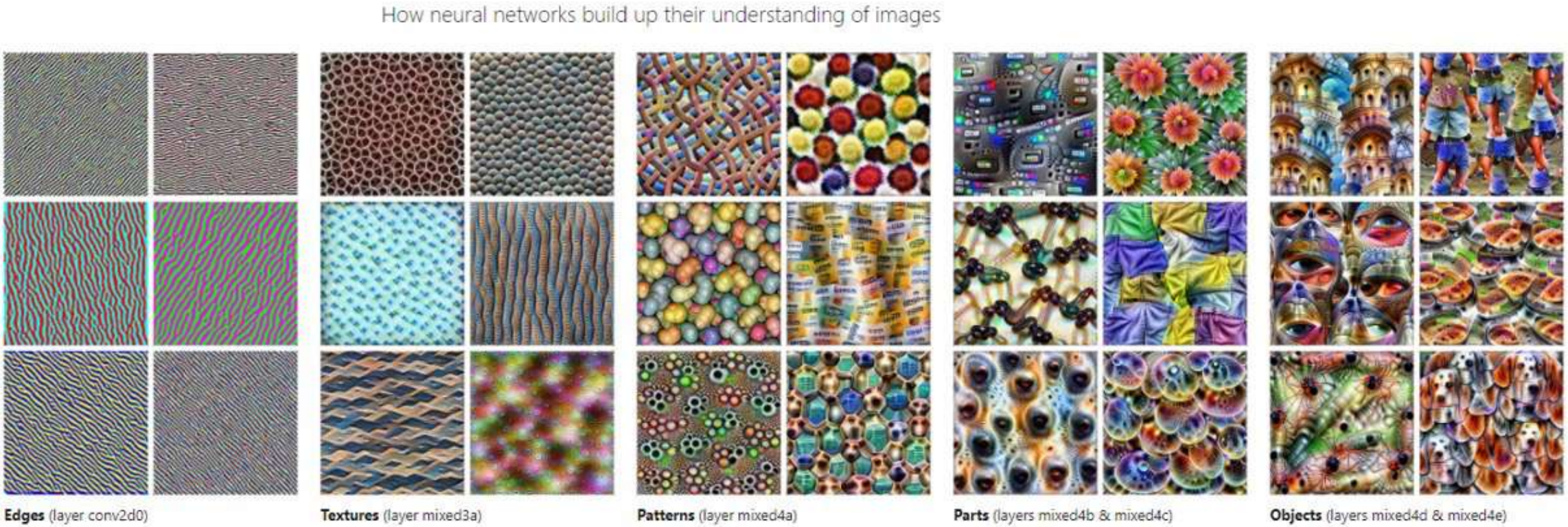
Wavelets



Trees are highly interpretable

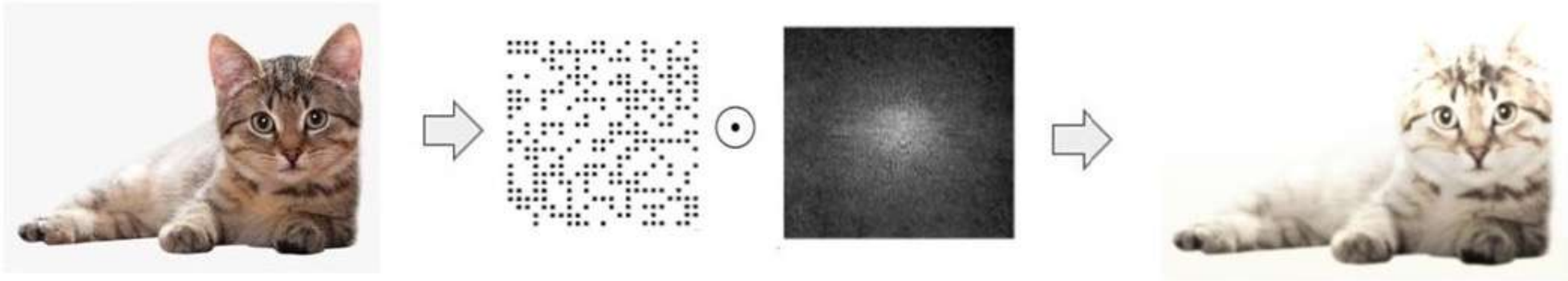


Are filters interpretable?

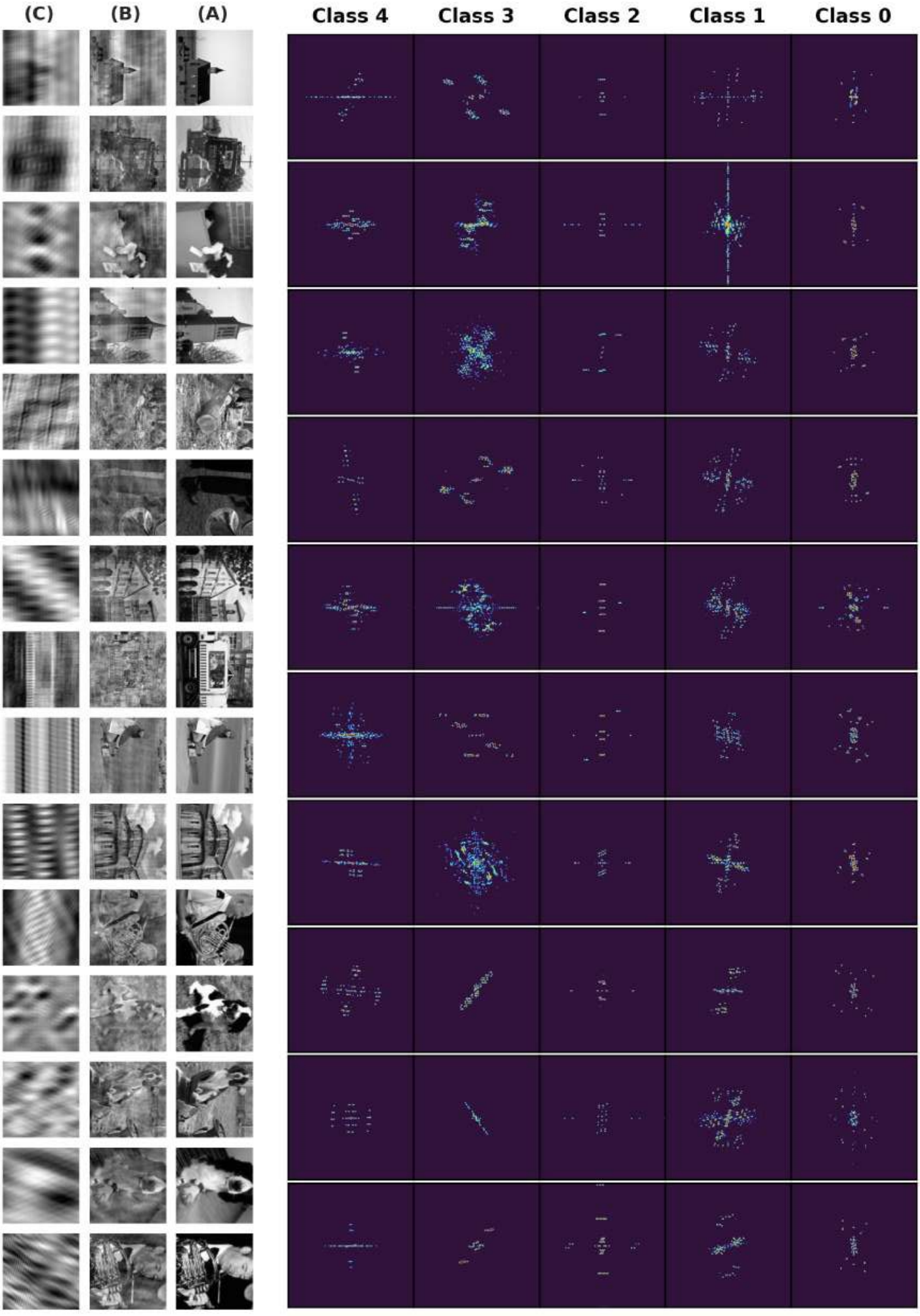


How does a network make a decision? Fourier important features.

x $\quad M_\Phi \odot \mathcal{F}x \quad \mathcal{F}^{-1}(M_\Phi \odot \mathcal{F}x)$



$$M_\Phi(\lambda, p) = \text{argmin}_{M_\Phi} \sum_{x \in \mathcal{X}_V} e^{[\mathcal{L}(\Phi(\bar{x}),y) - \mathcal{L}(\Phi(x),y)]^2} + \lambda \| M_\Phi \|_p, \quad \lambda \in \mathbb{R}_+$$


(C)
(B)
(A)
Class 4
Class 3
Class 2
Class 1
Class 0

Or texture?

IMAGENET-TRAINED CNNS ARE BIASED TOWARDS TEXTURE; INCREASING SHAPE BIAS IMPROVES ACCURACY AND ROBUSTNESS

Robert Geirhos
University of Tübingen & IMPRS-IS
robert.geirhos@bethgelab.org

Patricia Rubisch
University of Tübingen & U. of Edinburgh
p.rubisch@sms.ed.ac.uk

Claudio Michaelis
University of Tübingen & IMPRS-IS
claudio.michaelis@bethgelab.org

Matthias Bethge*
University of Tübingen
matthias.bethge@bethgelab.org

Felix A. Wichmann*
University of Tübingen
felix.wichmann@uni-tuebingen.de

Wieland Brendel*
University of Tübingen
wieland.brendel@bethgelab.org

(a) Texture image

81.4%	**Indian elephant**
10.3%	indri
8.2%	black swan

(b) Content image

71.1%	**tabby cat**
17.3%	grey fox
3.3%	Siamese cat

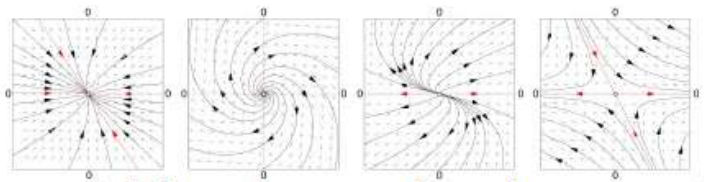
(c) Texture-shape cue conflict

63.9%	**Indian elephant**
26.4%	indri
9.6%	black swan

How humans solve problems?

Example: **separable differential equation**

$$\frac{dy}{dx} = \frac{y}{x} + \frac{y^2}{x^3}$$



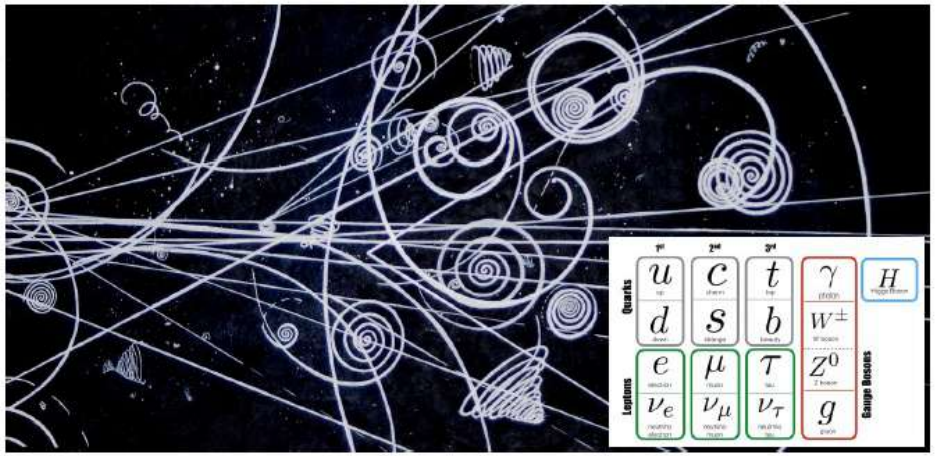
Find a **good transformation** of the variables that make the equation **simple**:

$$r = \frac{y}{x}, \quad s = -\frac{1}{x} \quad \Rightarrow \quad \frac{ds}{dr} = \frac{1}{r^2} \quad \Rightarrow \quad \int ds = \int \frac{dr}{r^2}$$

- Advantage: we have control and understanding of the steps.
- Disadvantage: is limited to our understanding of the problem.

Physical laws

A **physical law** is as powerful as it describes the biggest amount of experimental evidence with the greatest simplicity. In other words physical laws are **very efficient algorithms** to describe nature.



Key in the formulation of a laws is:

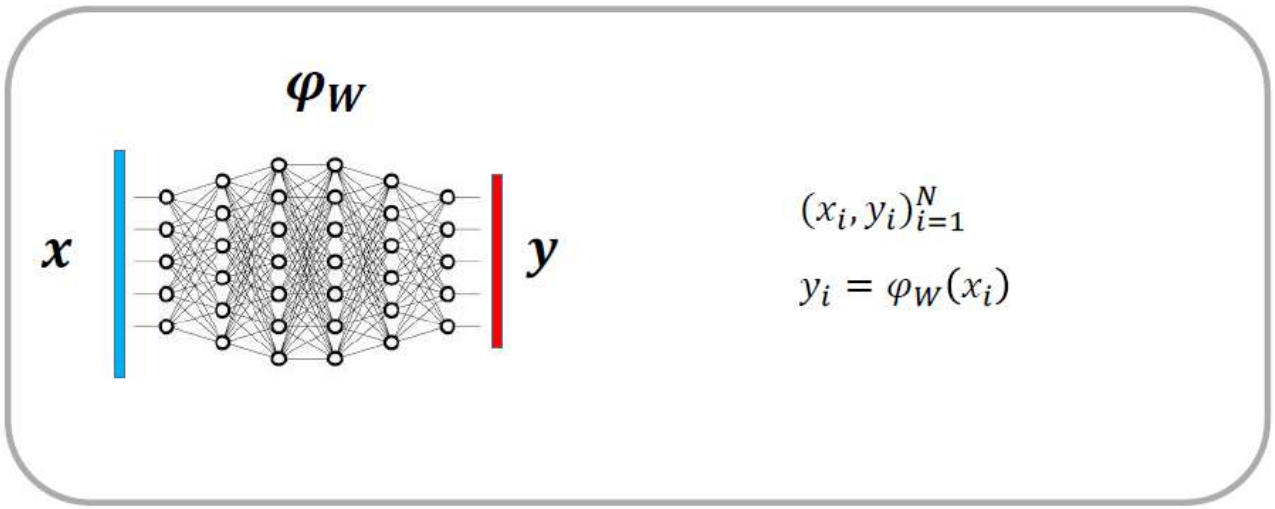
- Select relevant variables.
- Find the simplest function of the variables that explain experiments.

How deep learning solve a problem: data speak for themselves

Frame the problem as $f : \mathbb{R}^d \to \mathbb{R}^k$:

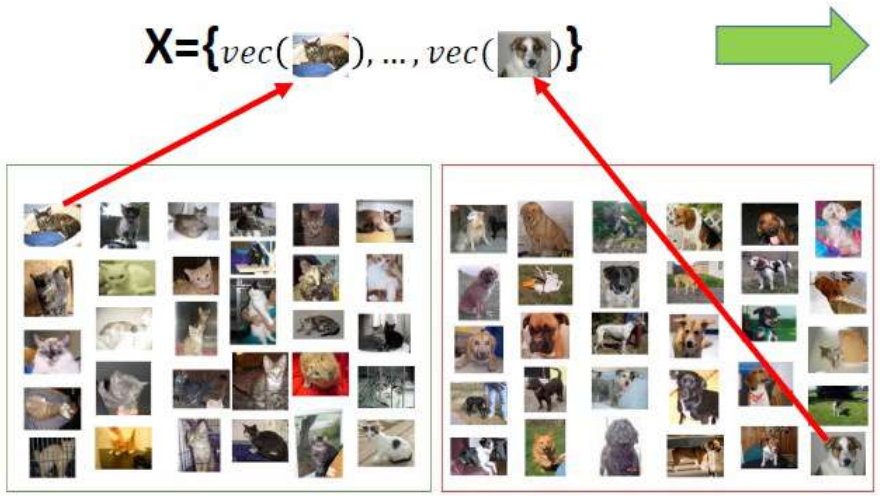
$$y = f(x).$$

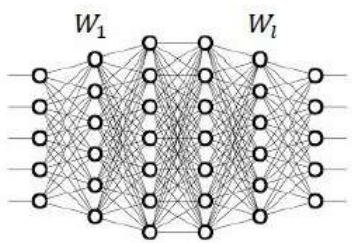
Replace f with a neural network φ_W and learn it from input/output examples:



- Advantage: no biases, data "speak" for themselves.
- Disadvantage: results are difficult/impossible to interpret.

An example of success: object recognition



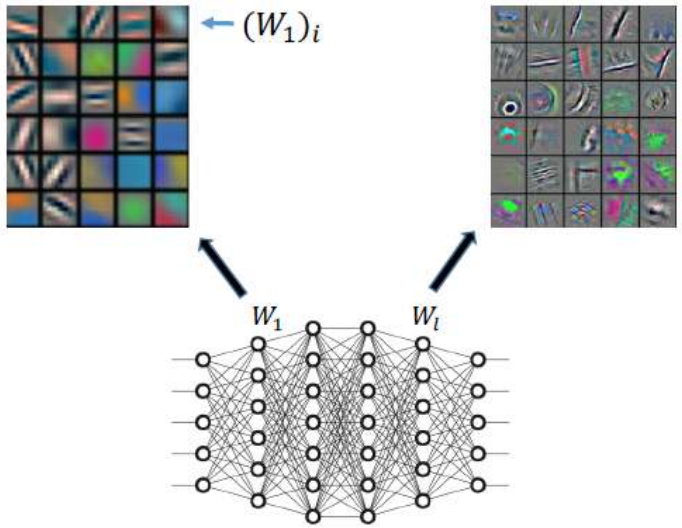


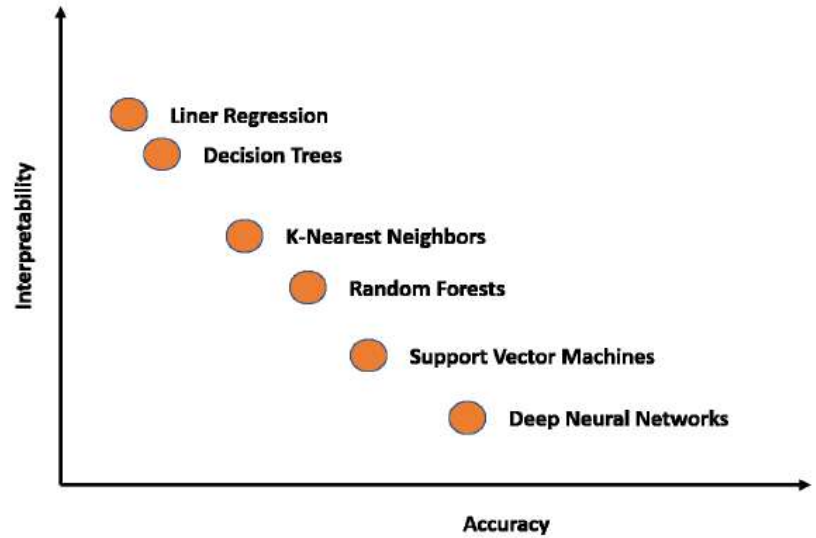
$$y = \varphi_W(x) = \sigma(W_l\sigma(\dots\sigma(W_1 x)\dots)$$

$$x \in R^d, W_i \in R^{k_i} \qquad \sigma(u)_i = \max(0, u_i)$$

$$W^* = \underset{W}{argmin}|Y - \varphi_W(X)|$$

Interpretability problem



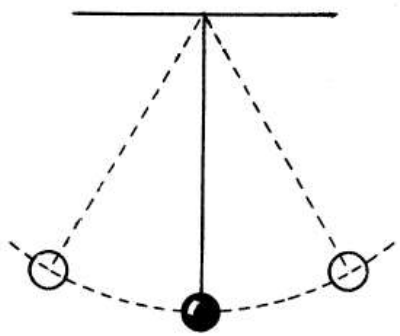


- Even in image recognition **we do not make sense of the filters**!
- In physics we need interpretable objects:

$$\partial_{\mathbf{x}}, \partial_{\mathbf{t}}, \nabla, \mathbf{P}, \mathbf{V}, ...$$

Data-driven discovery of new interpretable dynamical models

Suppose you want to learn the harmonic oscillator equation from data:



$$\{u((x,t)_i) \equiv u_i\}_{i=1}^N$$
$$\downarrow$$
$$(\partial_t - \omega^2 \partial_{xx})u \equiv Lu = 0$$

A way to proceed is to make the hypothesis that the differential equation is of the form:

$$\partial_t u = \{1, u, u^2, u_x, u_x^2, uu_x, u_{xx}, u_{xx}^2, uu_{xx}, u_x, u_{xx}\}\alpha = D\alpha$$

and minimize the number of active terms in α. Minimization problem:

Dictionary Approach

$$\alpha^* = \arg\min_{\alpha} \|\partial_{\mathbf{t}} \mathbf{u_i} - \mathbf{D_i}\alpha\|_{\mathbf{2}} + \|\alpha\|_{\mathbf{0}}$$

Data-driven discovery of new interpretable dynamical models

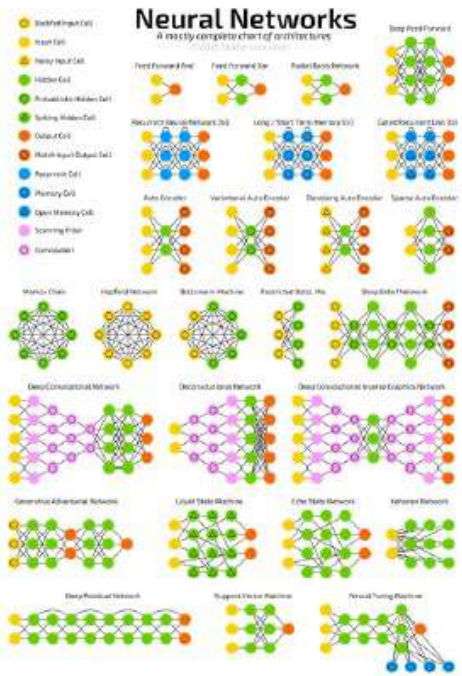
SCIENCE ADVANCES | RESEARCH ARTICLE

APPLIED MATHEMATICS

Data-driven discovery of partial differential equations

Samuel H. Rudy,[1]* Steven L. Brunton,[2] Joshua L. Proctor,[3] J. Nathan Kutz[1]

PDE	Form	Error (no noise, noise)	Discretization
 KdV	$u_t + 6uu_x + u_{xxx} = 0$	1%±0.2%, 7%±5%	x∈[−30, 30], n=512, t∈[0, 20], m=201
 Burgers	$u_t + uu_x - \epsilon u_{xx} = 0$	0.15%±0.06%, 0.8%±0.6%	x∈[−8, 8], n=256, t∈[0, 10], m=101
 Schrödinger	$iu_t + \frac{1}{2}u_{xx} - \frac{x^2}{2}u = 0$	0.25%±0.01%, 10%±7%	x∈[−7.5, 7.5], n=512, t∈[0, 10], m=401
 NLS	$iu_t + \frac{1}{2}u_{xx} + \lvert u\rvert^2 u = 0$	0.05%±0.01%, 3%±1%	x∈[−5, 5], n=512, t∈[0, π], m=501
 KS	$u_t + uu_x + u_{xx} + u_{xxxx} = 0$	1.3%±1.3%, 70%±27%	x∈[0, 100], n=1024, t∈[0, 100], m=251
u  v Reaction Diffusion	$u_t = 0.1\nabla^2 u + \lambda(A)u - \omega(A)v$ $v_t = 0.1\nabla^2 v + \omega(A)u + \lambda(A)v$ $A^2 = u^2 + v^2, \omega = -\beta A^2, \lambda = 1 - A^2$	0.02% ± 0.01%, 3.8% ± 2.4%	x, y∈[−10, 10], n=256, t∈[0, 10], m=201 subsample 1.14%
 Navier Stokes	$\omega_t + (\mathbf{u}\cdot\nabla)\omega = \frac{1}{Re}\nabla^2\omega$	1% ± 0.2% , 7% ± 6%	x∈[0, 9], n_x=449, y∈[0, 4], n_y=199, t∈[0, 30], m=151, subsample 2.22%



Navier-Stokes Equations

Continuity Equation

$$\nabla \cdot \vec{V} = 0$$

Momentum Equations

$$\rho \frac{D\vec{V}}{Dt} = -\nabla p + \rho \vec{g} + \mu \nabla^2 \vec{V}$$

Total derivative: $\rho[\frac{\partial V}{\partial t} + (V.\nabla)V]$

- Change of velocity with time
- Convective term

Pressure gradient: Fluid flows in the direction of largest change in pressure.

Body force term: External forces, that act on the fluid (gravitational force or electromagnetic).

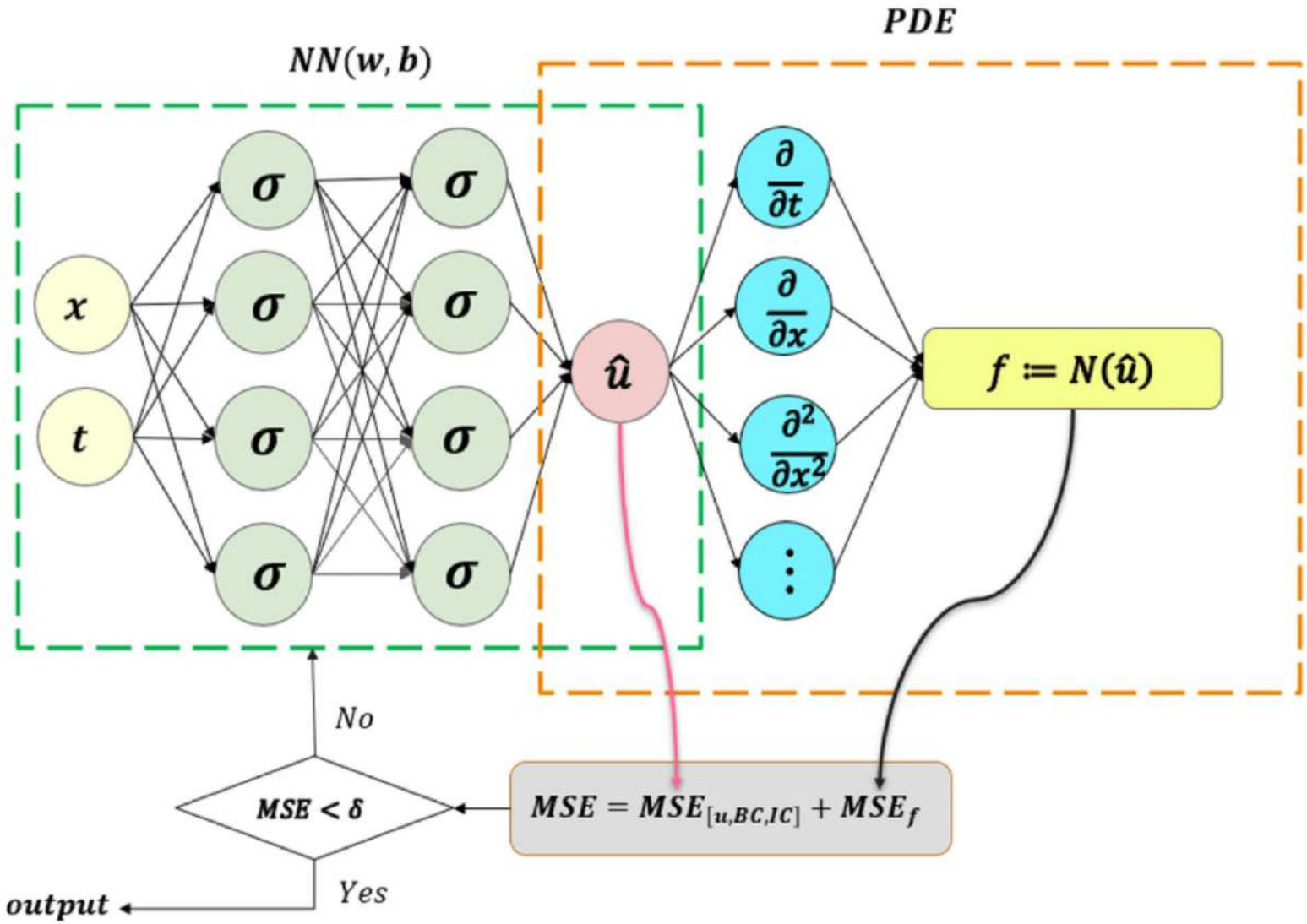
Diffusion term: For a Newtonian fluid, viscosity operates as a diffusion of momentum.

Suppose we learnt a φ_W from input/output to a physical system: $y_i = \varphi_W(x_i)$

Approximate φ_W with a dictionary of known operators:

$$\arg\min_{\alpha} \sum_{\mathbf{i}} \|\varphi_{\mathbf{W}}(\mathbf{x_i}) - (\{\partial_{\mathbf{t}}, \partial_{\mathbf{x}}, ...\}\alpha)(\mathbf{x_i})\|_{\mathbf{2}} + \|\alpha\|_{\mathbf{0}}$$

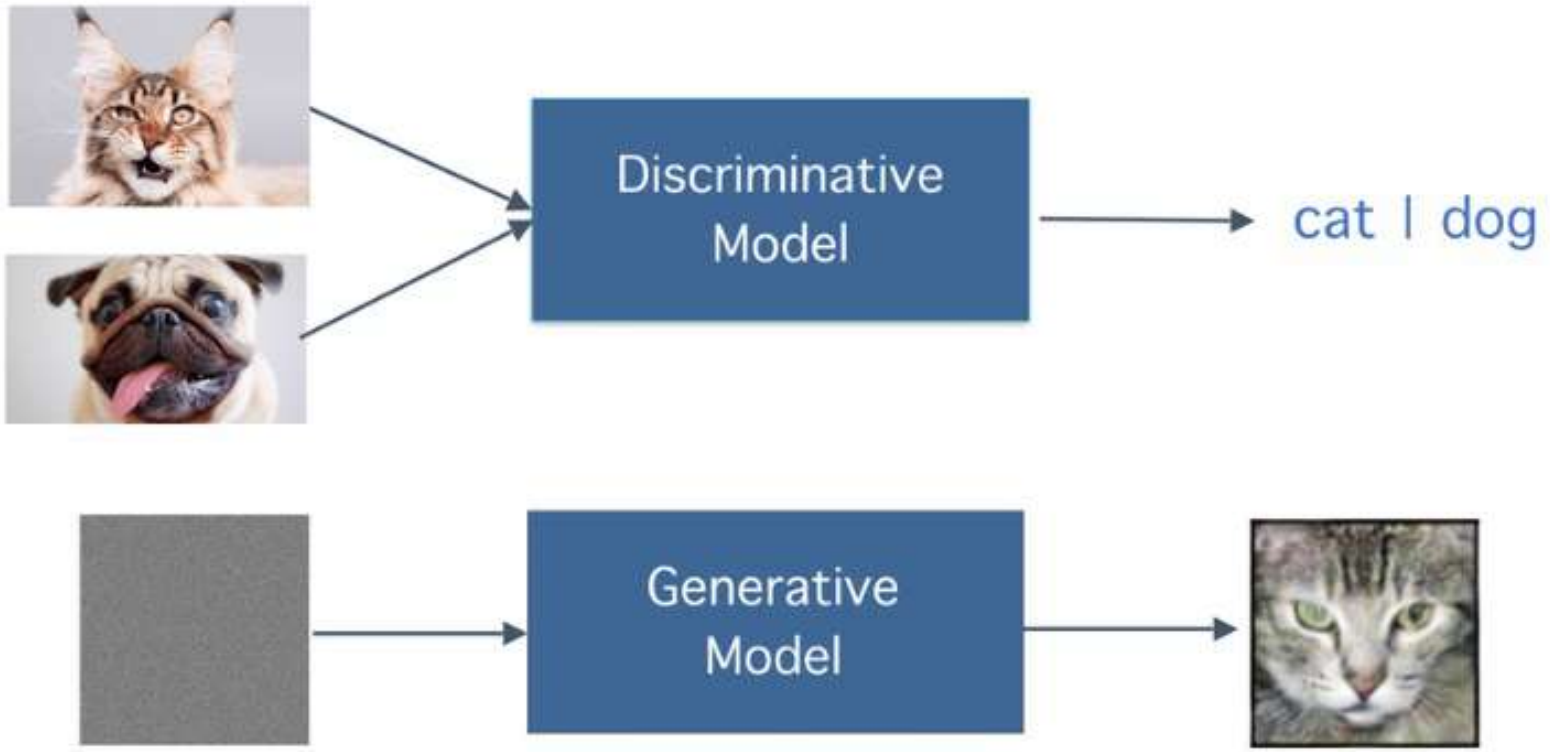
Learning differential equations



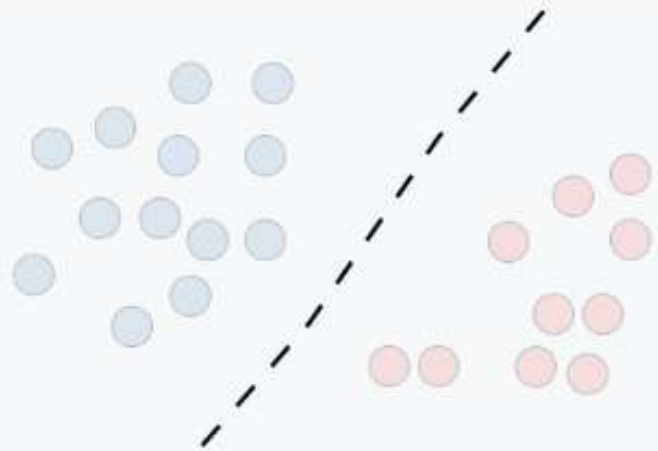
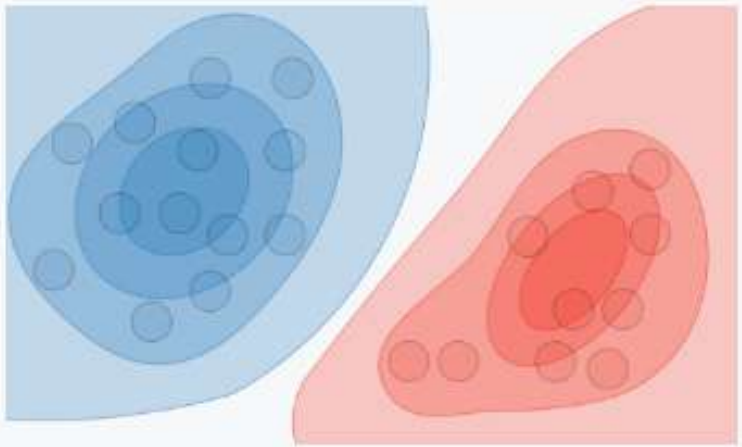
Project: learning differential equations using hierarchical dictionary of learnable nls

Short intro to generative models

Generative vs discriminative model



Generative vs discriminative model

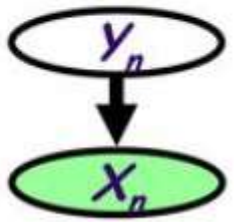
	Discriminative model	Generative model
Goal	Directly estimate $P(y\|x)$	Estimate $P(x\|y)$ to then deduce $P(y\|x)$
What's learned	Decision boundary	Probability distributions of the data
Illustration		

Can you give some examples of discriminative models?

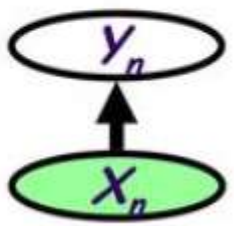
Classifiers

- Goal: Wish to learn f: $X \rightarrow Y$, e.g., $P(Y|X)$

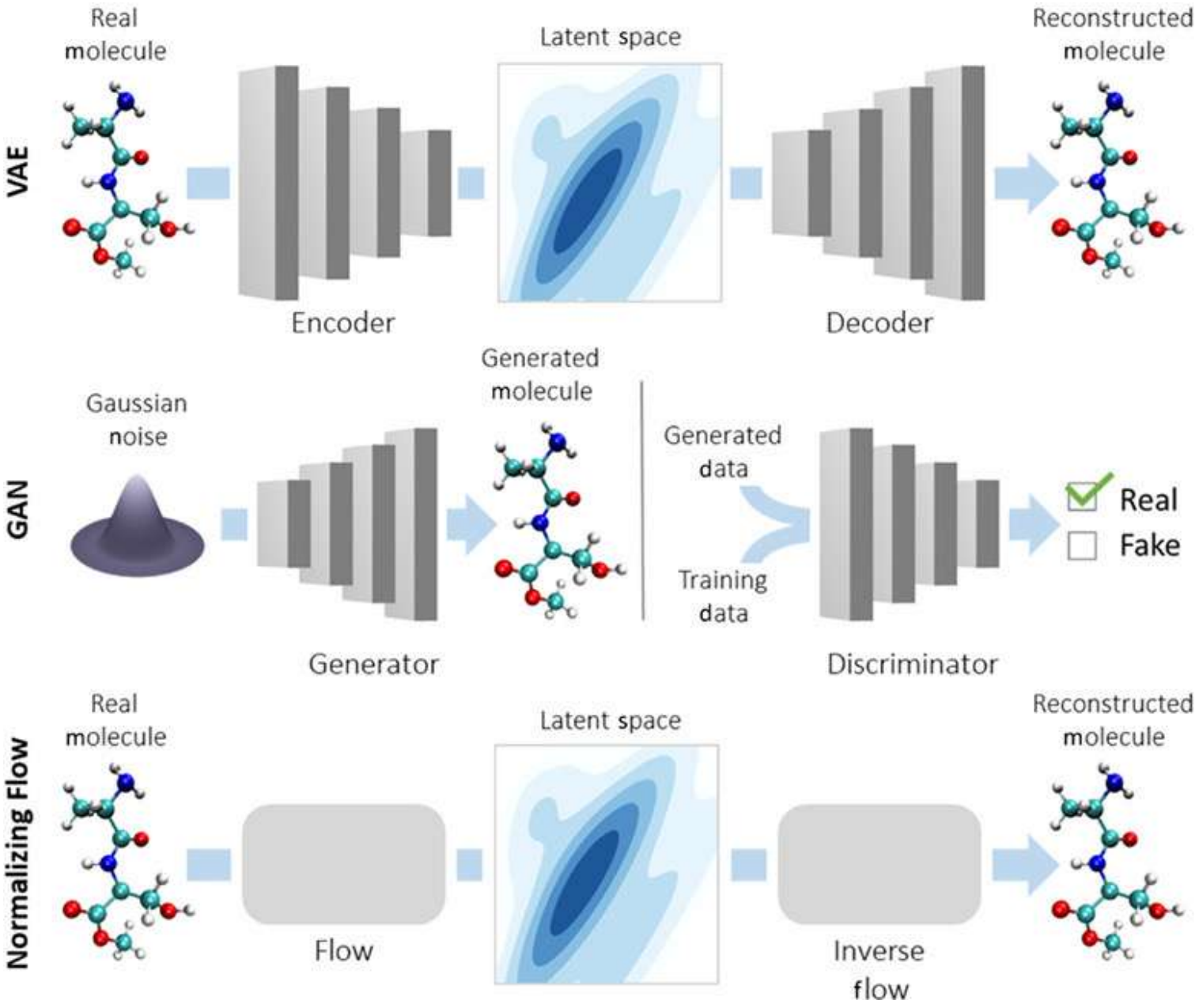
- Generative classifiers (e.g., Naïve Bayes):
  - Assume some functional form for $P(X|Y)$, $P(Y)$
    This is a '***generative***' model of the data!
  - Estimate parameters of $P(X|Y)$, $P(Y)$ directly from training data
  - Use Bayes rule to calculate $P(Y|X= x)$



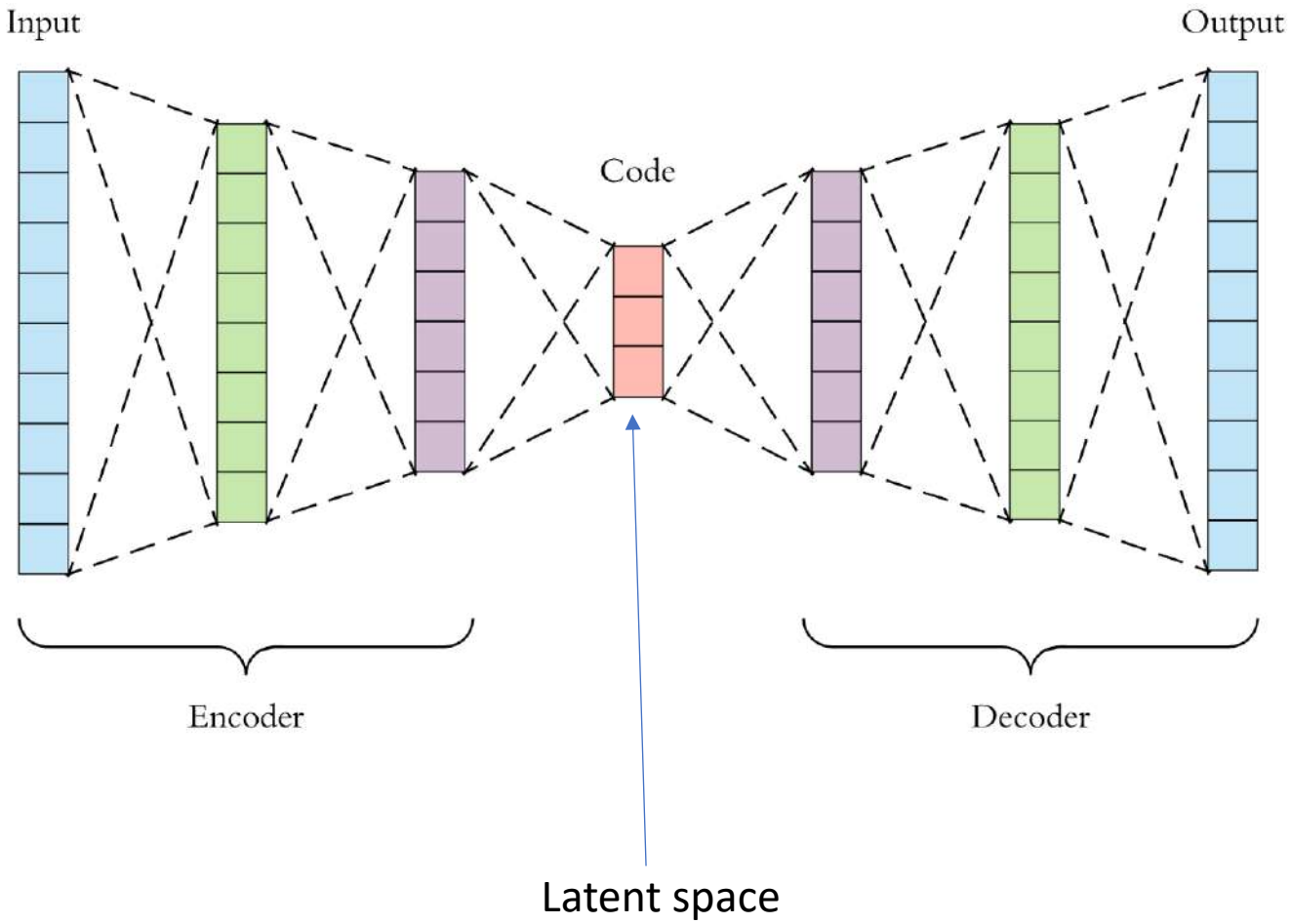
- Discriminative classifiers (e.g., logistic regression)
  - Directly assume some functional form for $P(Y|X)$
    This is a '***discriminative***' model of the data!
  - Estimate parameters of $P(Y|X)$ directly from training data



Types of generative models

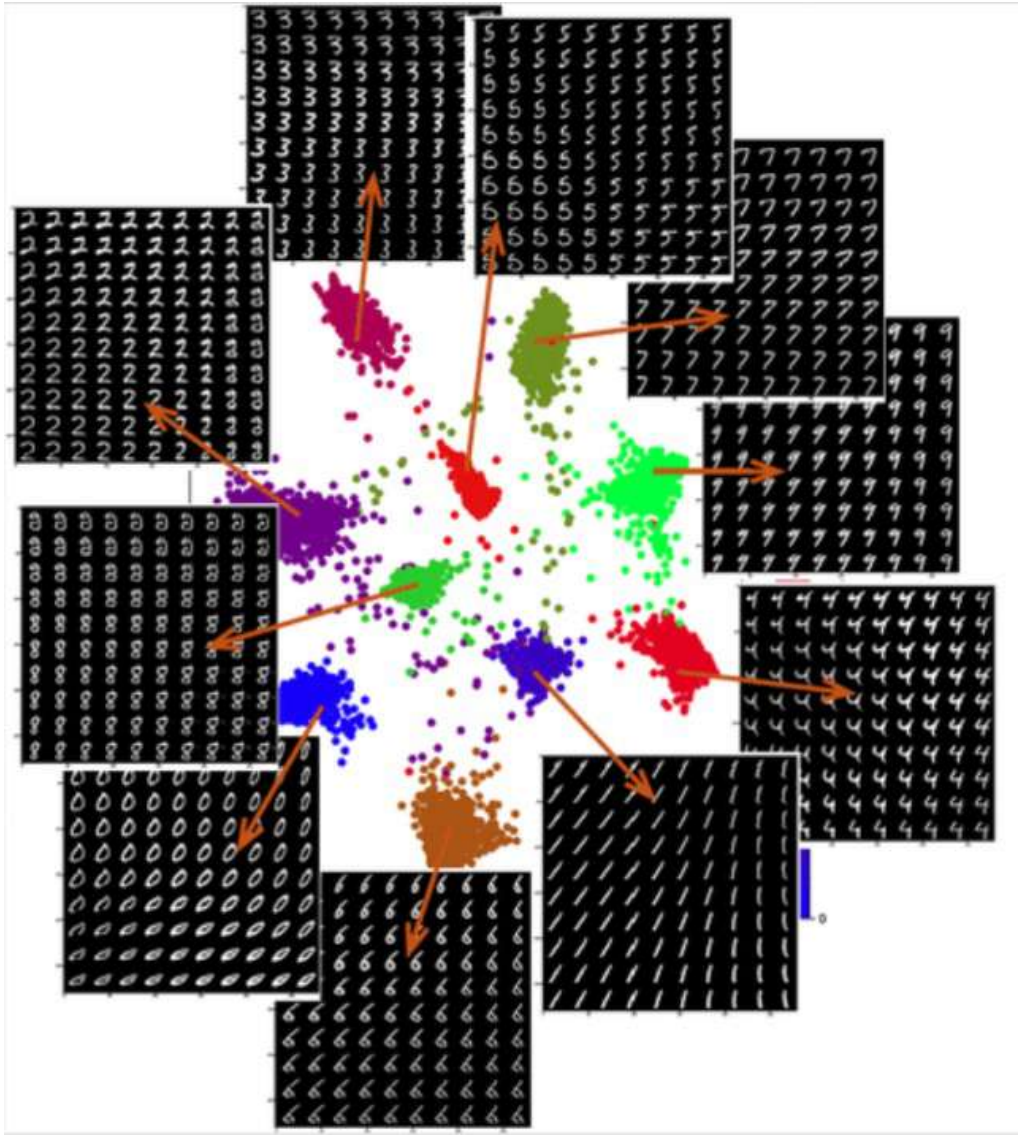


Autoencoders

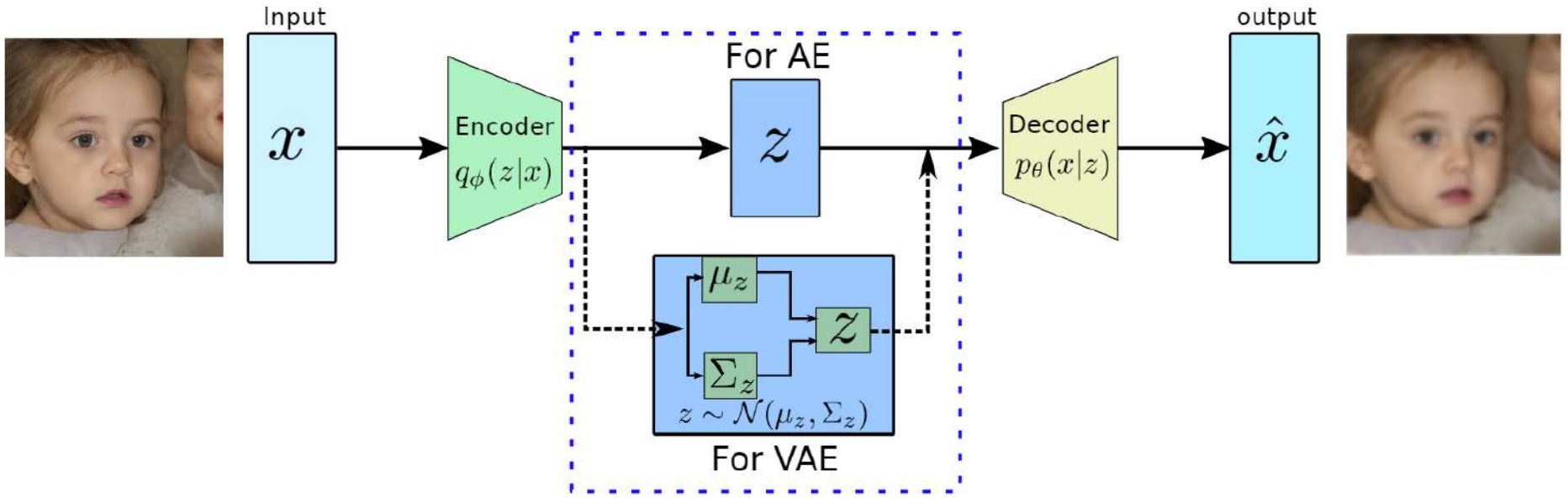


How would you write the loss of an autoencoder?
How would you use an autoencoder as a generative model?

Latent space for MNIST

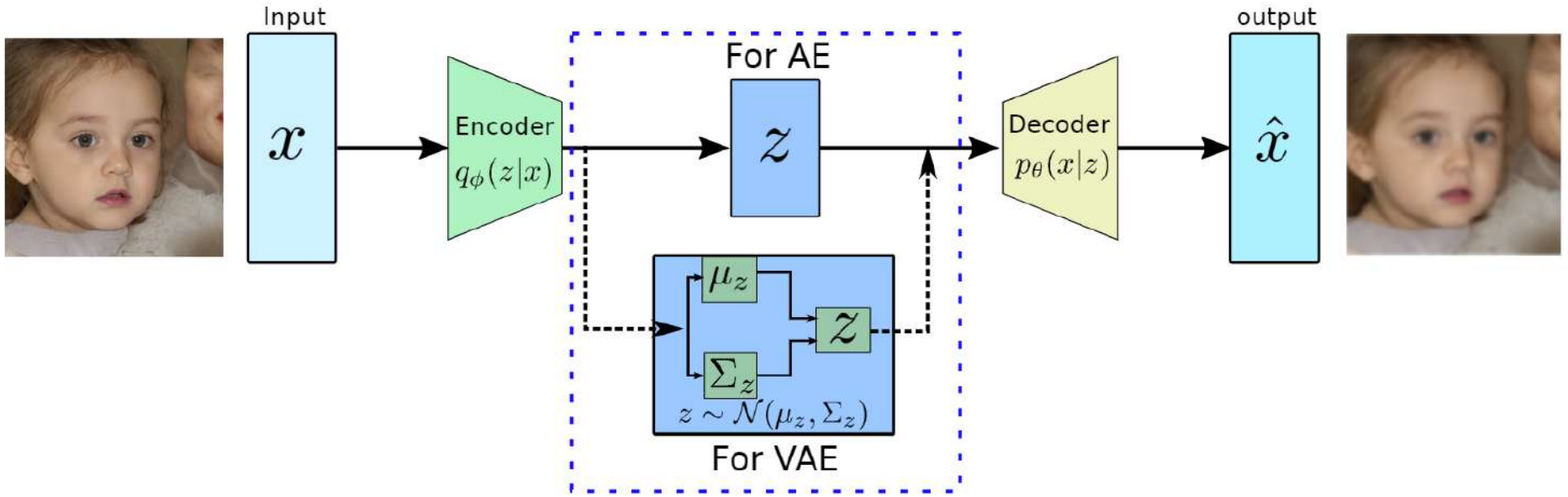


Variational autoencoder



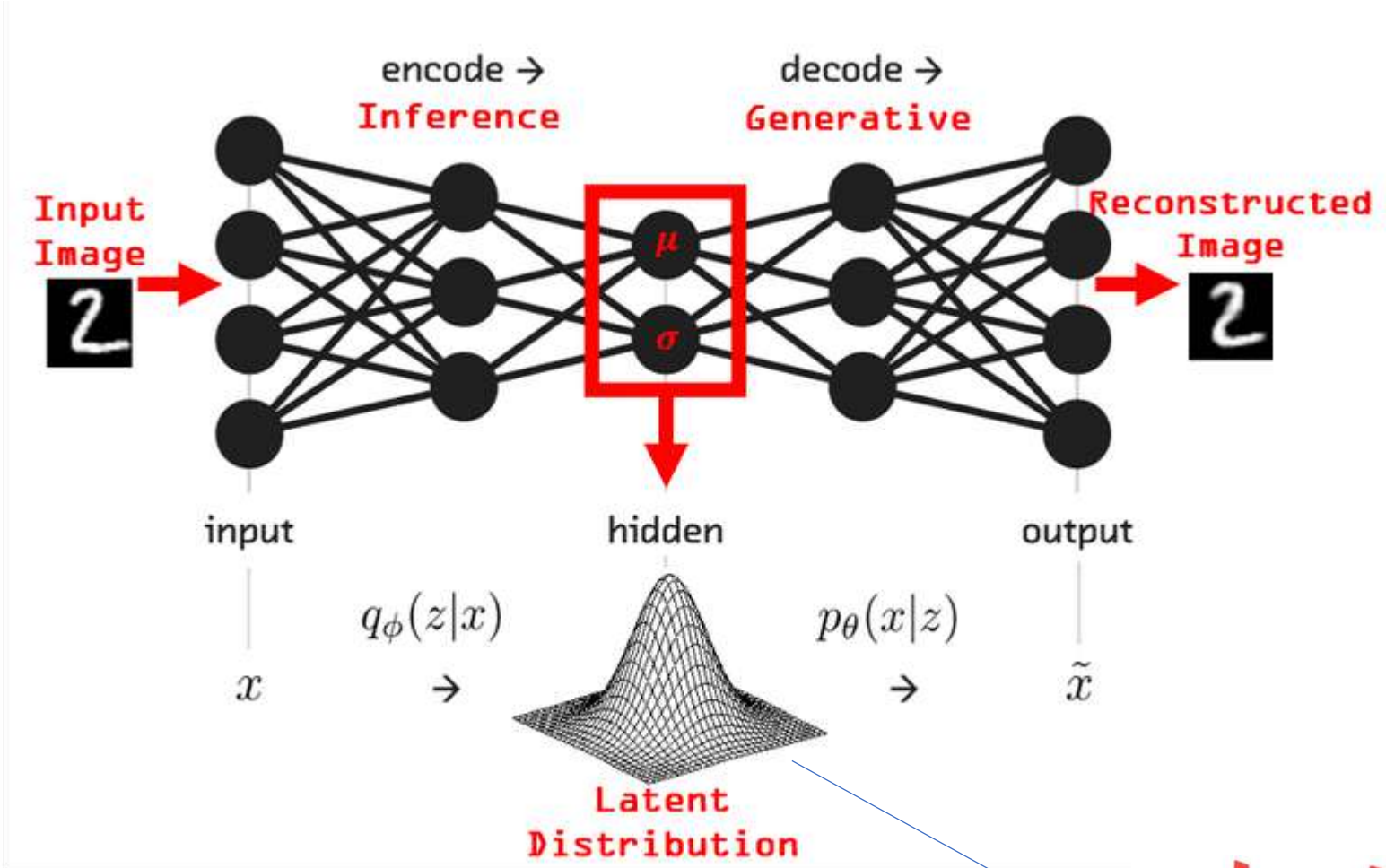
Instead of outputting the vectors in the latent space, the encoder of VAE outputs parameters of a pre-defined distribution in the latent space for every input. The VAE then imposes a constraint on this latent distribution forcing it to be a normal distribution.

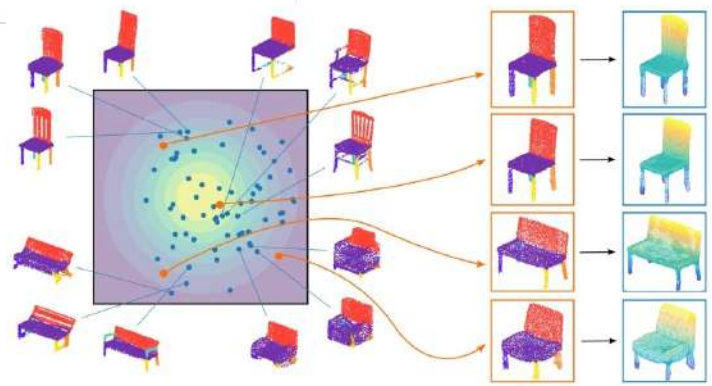
Variational autoencoder



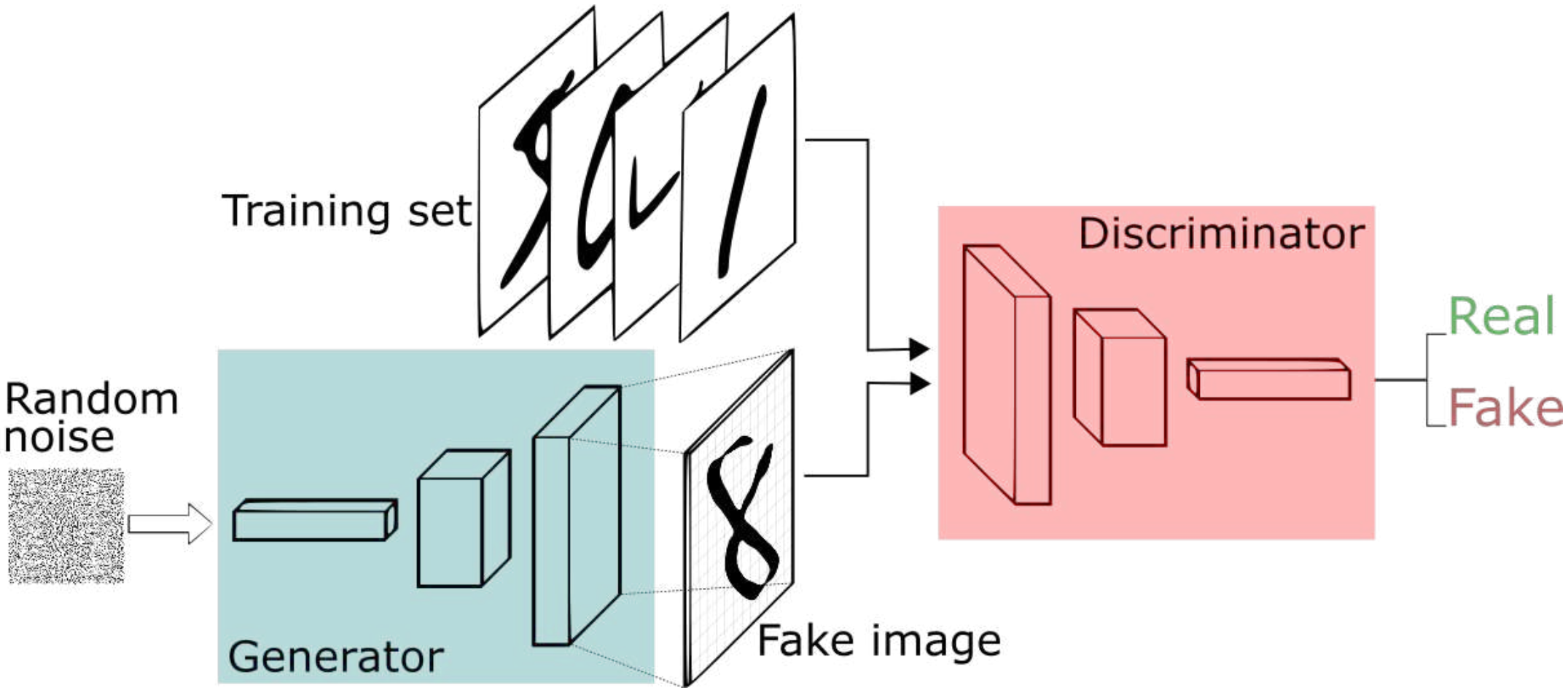
The latent vector is sampled from the encoder-generated distribution before feeding it to the decoder. This random sampling makes it difficult for backpropagation to happen for the encoder since we can't trace back errors due to this random sampling. Hence we use a reparameterization trick to model the sampling process which makes it possible for the errors to propagate through the network. The latent vector z is represented as a function of the encoder's output.

$$z = \mu_x + \sigma_x \epsilon, \quad \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

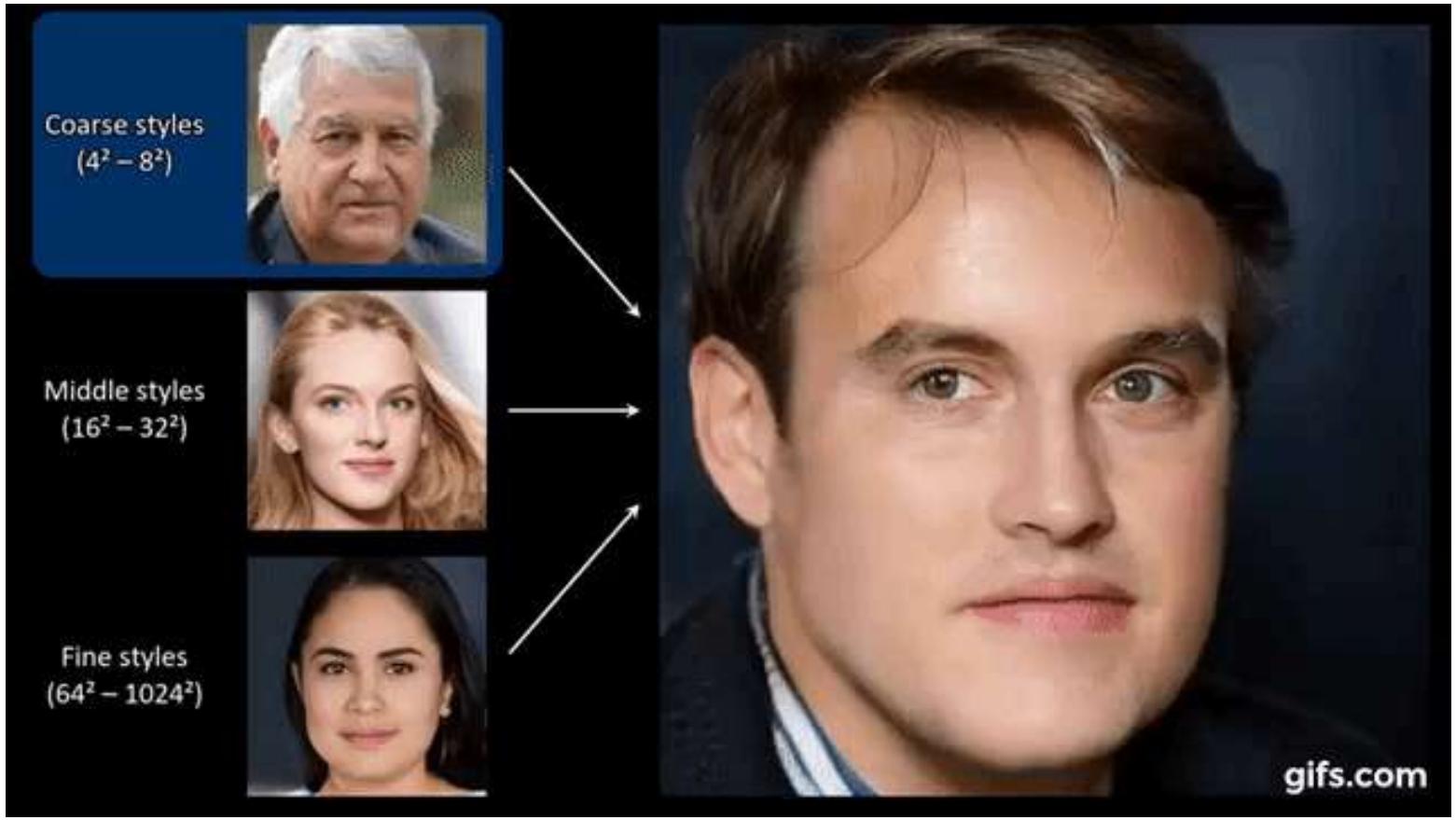

encode →
Inference
decode →
Generative
Input Image
Reconstructed Image
μ
σ
input
hidden
output
$q_\phi(z|x)$
$p_\theta(x|z)$
x
→
→
$\tilde{x}$
Latent Distribution



Generative adversarial networks



Living portraits


Coarse styles
(4² – 8²)
Middle styles
(16² – 32²)
Fine styles
(64² – 1024²)
gifs.com

Class 6: Loose hands

Gan Loss

gradient ascent

predict well on real images
=> want probability close to 1

predict well on fake images
=> want probability close to 0

$$\nabla_{\mathbf{w}_D} \frac{1}{n} \sum_{i=1}^{n} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right]$$

Discriminator objective in the neg. log-likelihood (binary cross entropy) perspective:

Real images, $y = 1$

$$\mathcal{L}(\mathbf{w}) = -y^{(i)} \log\left(\hat{y}^{(i)}\right) - \left(1 - y^{(i)}\right) \log\left(1 - \hat{y}^{(i)}\right)$$

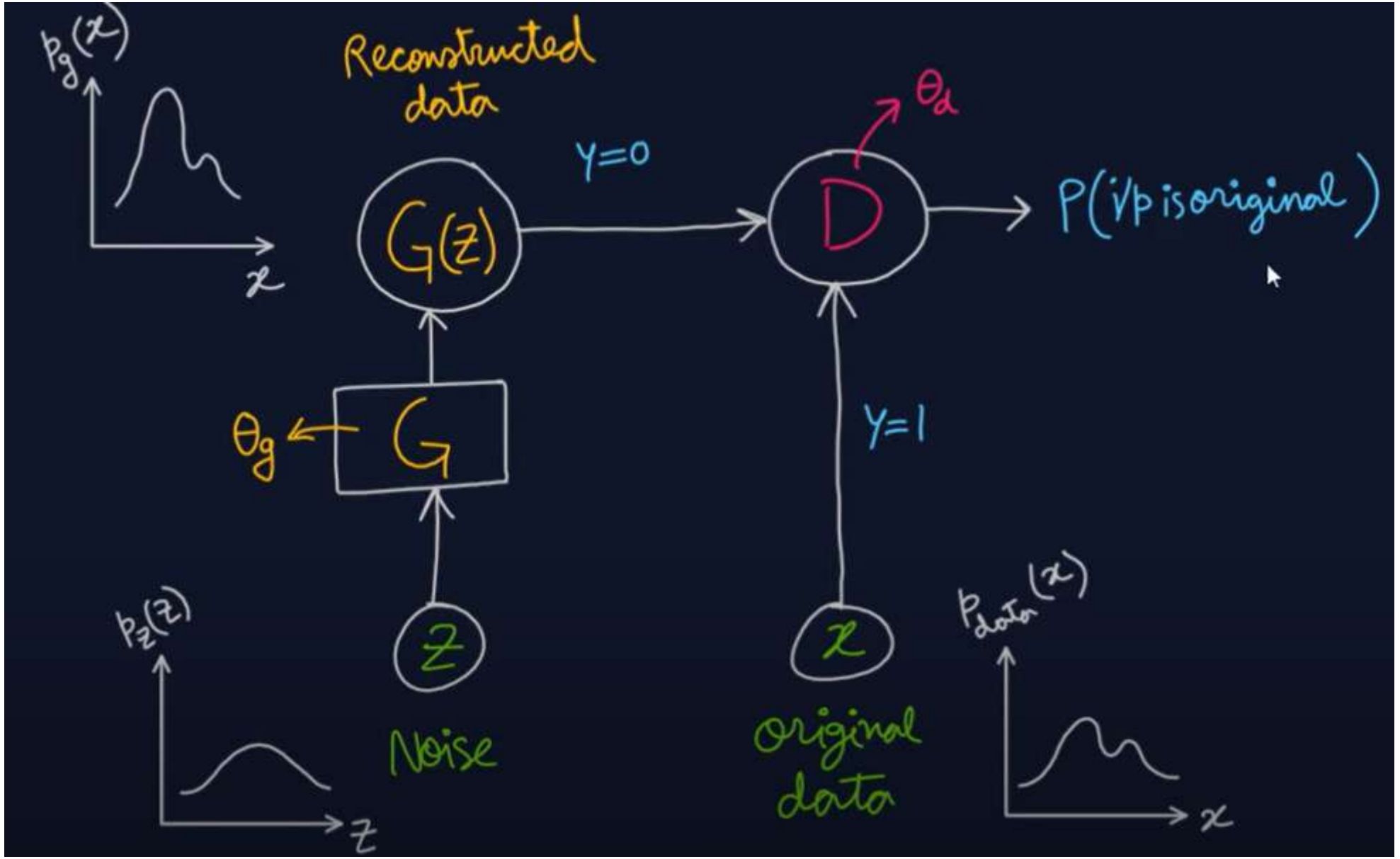
Want, $\hat{y} = 1$

Fake images, $y = 0$

$$\mathcal{L}(\mathbf{w}) = -y^{(i)} \log\left(\hat{y}^{(i)}\right) - \left(1 - y^{(i)}\right) \log\left(1 - \hat{y}^{(i)}\right)$$

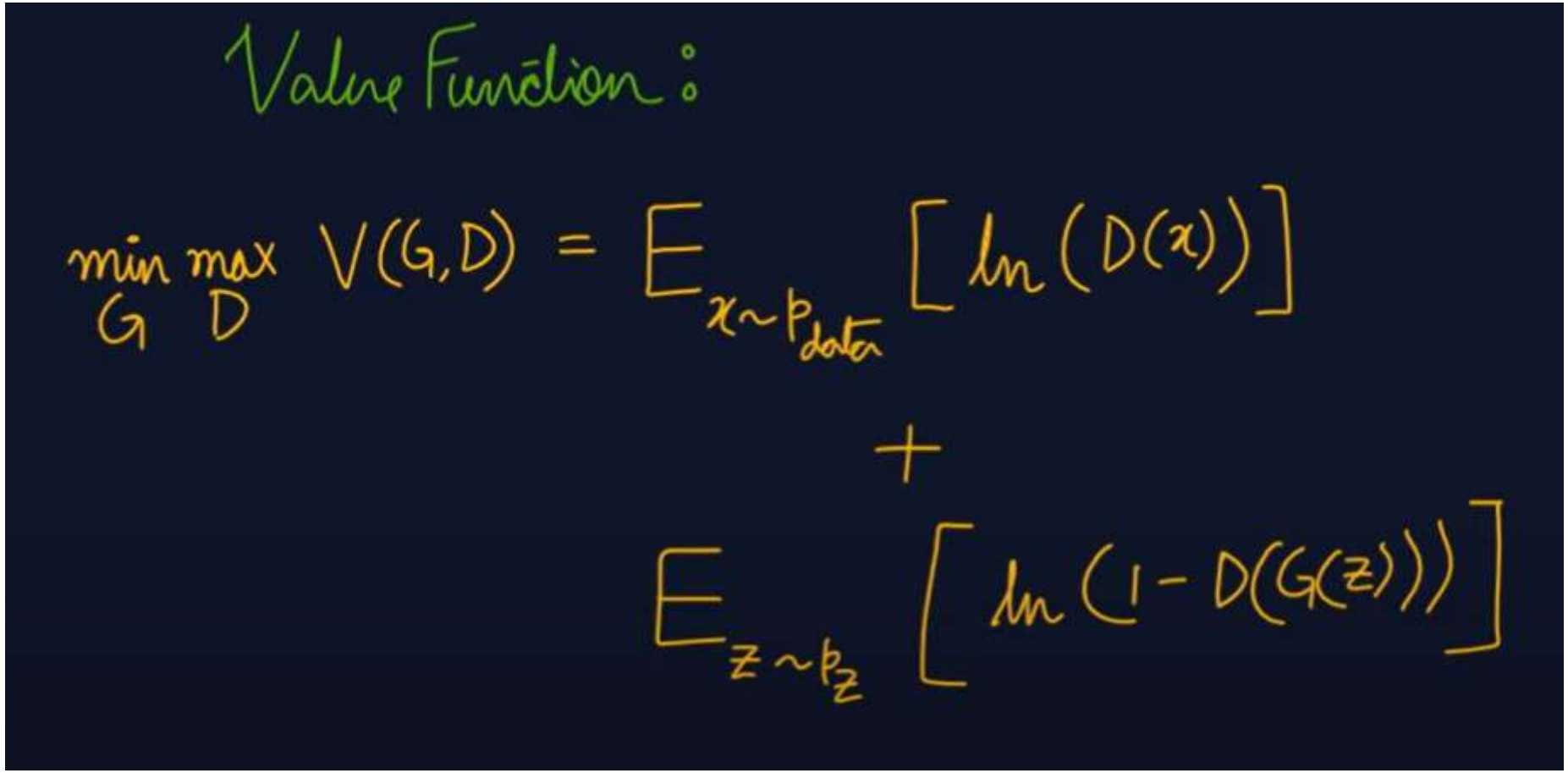
Want, $\hat{y} = 0$

General setting

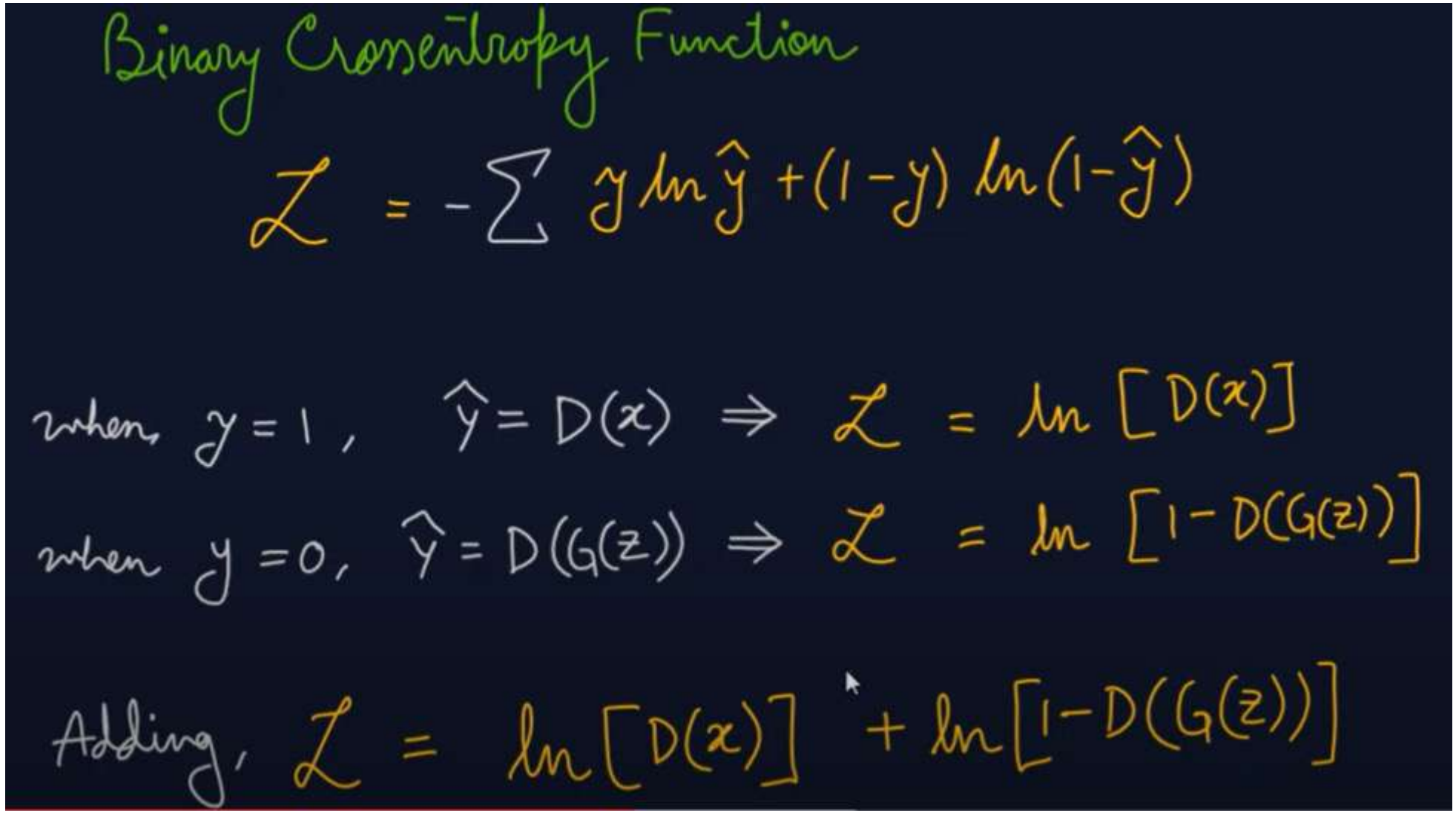


https://www.youtube.com/watch?v=Gib_kiXgnvA

The problem: the discriminator try to mimic data distribution; the generator tries to fool the discriminator



Analogy with Binary Cross-Entropy



Expectation

$$E(\mathcal{L}) = E\left(\ln[D(x)]\right) + E\left(\ln[1 - D(G(z))]\right)$$

$$\sum p_{data}(x) \ln[D(x)] + \sum p_z(z) \ln[1 - D(G(z))]$$

$$\int p_{data}(x) \ln[D(x)]\, dx + \int p_z(z) \ln[1 - D(G(z))]\, dz$$

$$V(G,D) = E_{x \sim p_{data}}\left[\ln(D(x))\right] + E_{z \sim p_z}\left[\ln(1 - D(G(z)))\right]$$

Traning loop:

* fix the learning of G *

Inner loop for D:

- take m data samples & m fake data samples
- update θ_d by grad. ascent

$$\frac{\partial}{\partial \theta_d} \frac{1}{m} \left[\ln [D(x)] + \ln [1 - D(G(z))] \right]$$

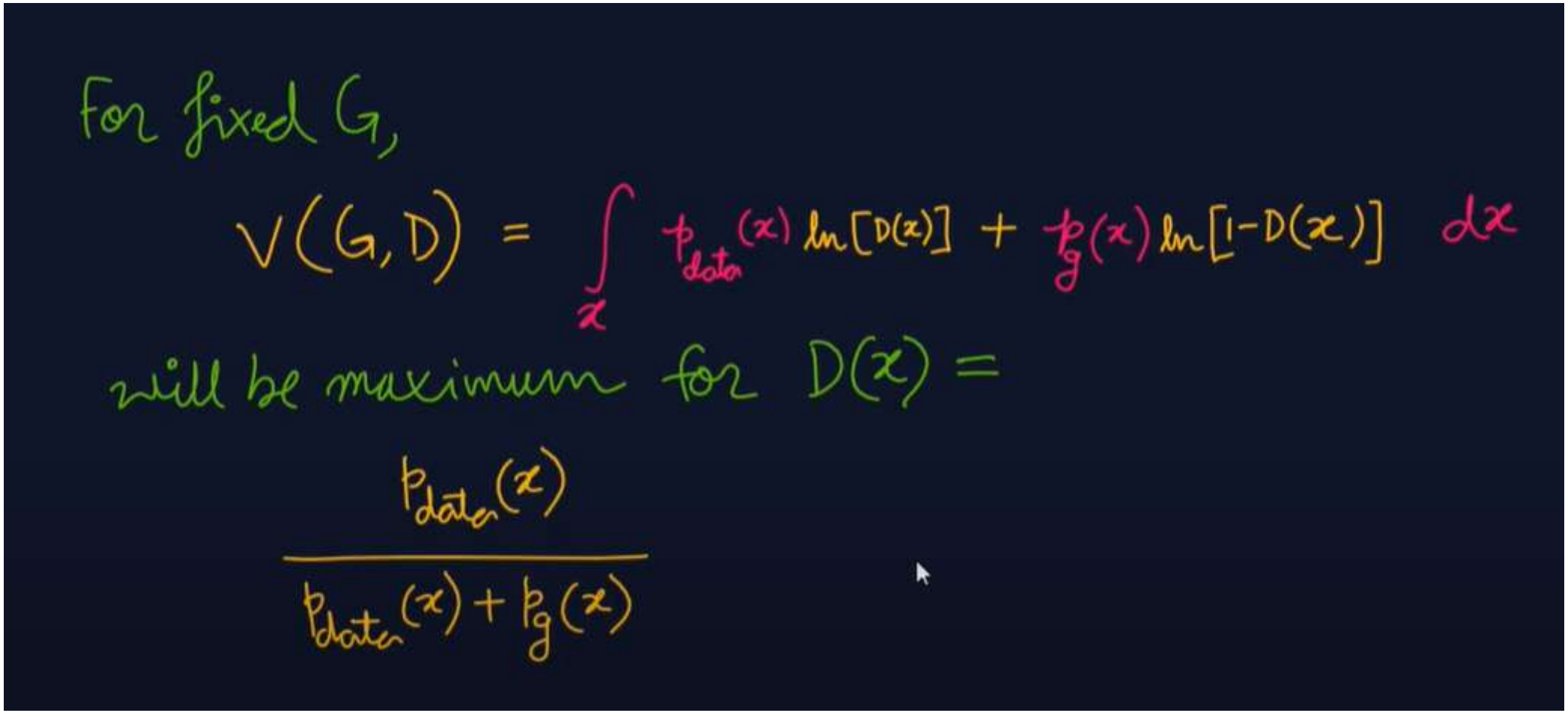
* fix the learning of D *

take m fake data samples

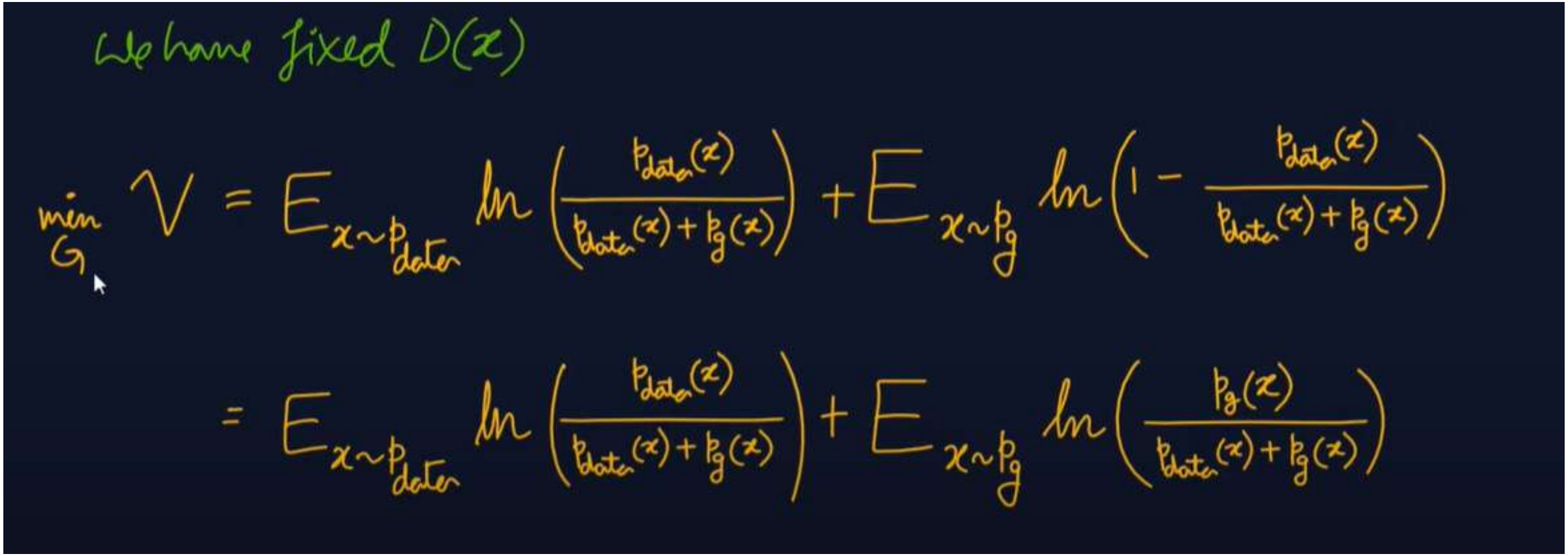
update θ_g by grad. descent

$$\frac{\partial}{\partial \theta_g} \frac{1}{m} \left[\ln [1 - D(G(z))] \right]$$

What is the Optimization doing?



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



$$= E_{x \sim p_{data}} \ln \left(\frac{p_{data}(x)}{p_{data}(x) + p_g(x)} \right) + E_{x \sim p_g} \ln \left(\frac{p_g(x)}{p_{data}(x) + p_g(x)} \right)$$

JS divergence

$$JS(p_1 \| p_2) = \frac{1}{2} E_{x \sim p_1} \ln \left(\frac{p_1}{\frac{p_1 + p_2}{2}} \right) + \frac{1}{2} E_{x \sim p_2} \left(\frac{p_2}{\frac{p_1 + p_2}{2}} \right)$$

$$\min_G V = E_{x \sim p_{data}} \ln\left(\frac{p_{data}(x)}{\frac{p_{data}(x) + p_g(x)}{2}}\right) + E_{x \sim p_g} \ln\left(\frac{p_g(x)}{\frac{p_{data}(x) + p_g(x)}{2}}\right) - 2\ln 2$$

$$\min_G V = 2\, JS(p_{data} \| p_g) - 2\ln 2$$

The optimization tries to match the training data and the generator data probability distributions