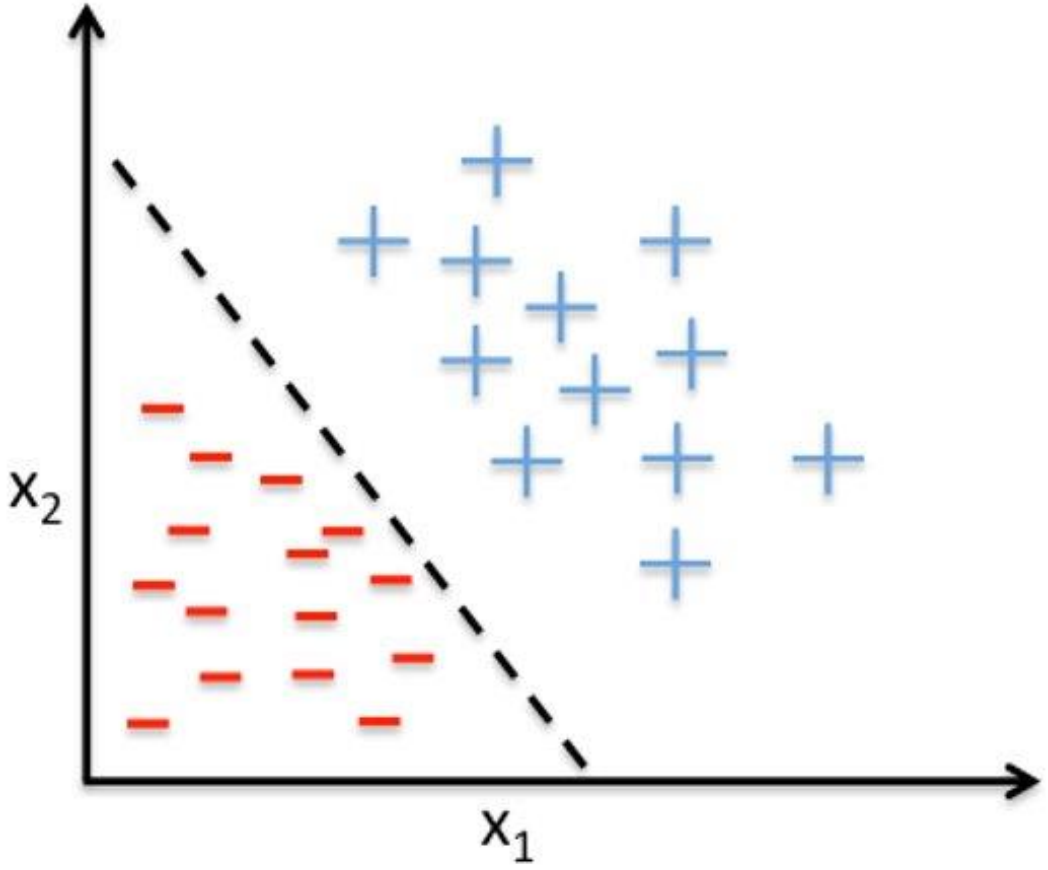
Neural networks

Class 1

- From kernels to Anns: perceptron, svm and kernels
- New hypothesis space (example with invariant functions)
- Shallow and deep networks, terminology;
- Invariant representations, sample complexity, intuition with ReLU.

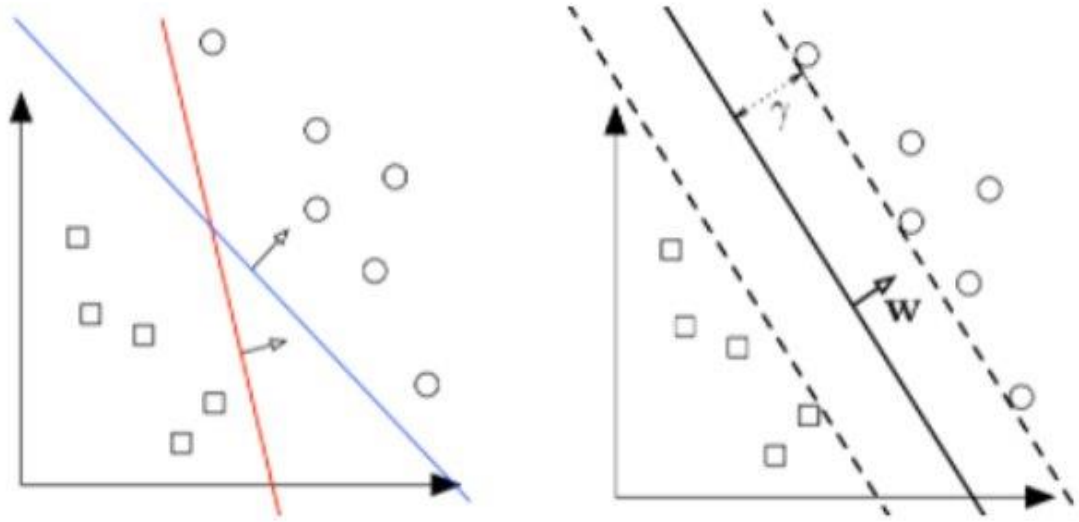
Perceptron: hyperplane separation



$$h(x) = sgn(w^T x + b)$$

$$L(w) = \frac{1}{N} \sum_i \max(h(x_i) y_i, 0)$$

SVM: maximal margin hyperplane separation

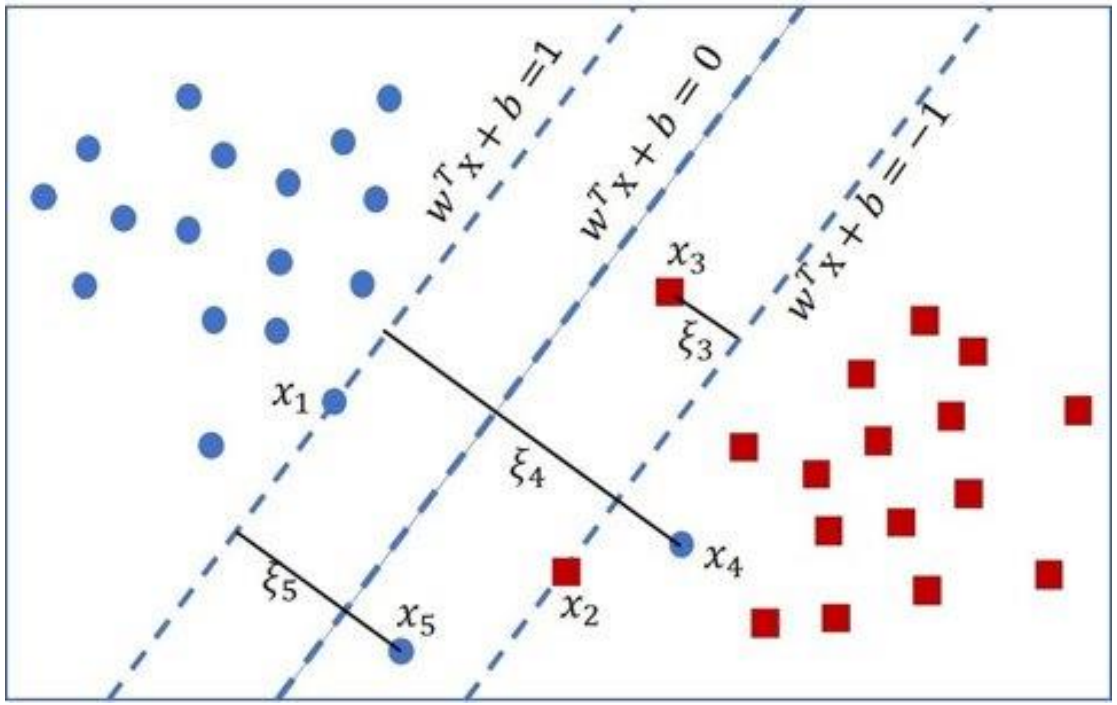


$$min_w \ \|w\|_2^2$$
$$s.t. \ \forall i \ \ y_i h(x_i) \geq 1$$

$$\mathcal{L}(w) = \frac{1}{N}\sum_{i=1}^{N} \max[y_i h(x_i), 0] + \lambda \|w\|_2^2$$

What's 'wrong' in the Loss?

Soft margin SVM and Kernels



$$\min_{\mathbf{w},b} \mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{n} \xi_i$$
$$s.t.\ \forall i\ y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i$$
$$\forall i\ \xi_i \geq 0$$

Soft margin SVM and Kernels

$$\min_{\mathbf{w},b} \mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{n}\xi_i$$
$$s.t.\ \forall i\ y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i$$
$$\forall i\ \xi_i \geq 0$$

Dual SVM

$$\min_{\alpha} \frac{1}{2}\sum_i\sum_j \alpha_i\alpha_j y_i y_j x_i^\top x_j - \sum_i \alpha_i$$

$$\text{s.t.} \quad \sum_i \alpha_i y_i = 0, \quad \alpha_i \in [0, C]$$

$$w = \sum_i \alpha_i y_i x_i \qquad b = y_i - \sum_j \alpha_j y_j x_j^\top x_i$$

for examples i where $0 < \alpha_i < C$

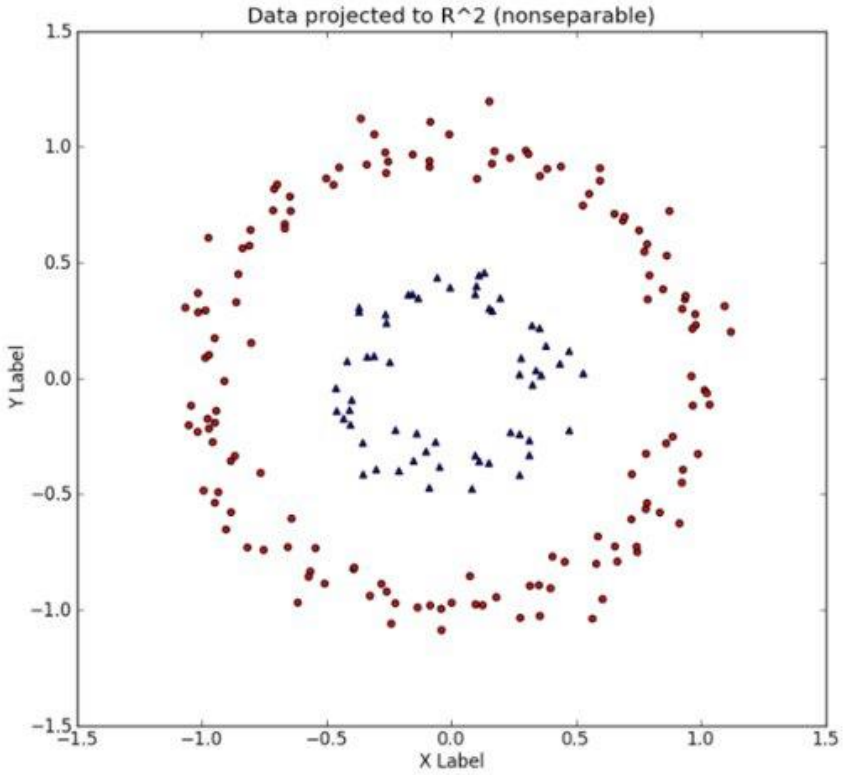
$$f(x) = w^\top x + b = \sum_i \alpha_i y_i x_i^\top x + b$$

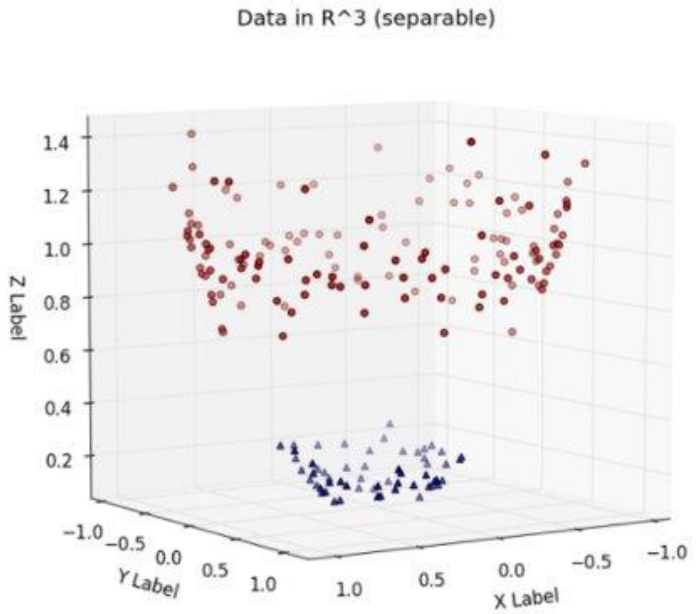
We can express the minimization using dot products of the input

$$K_{i,j} = K(x_i, x_j) = x_i^T x_j$$

Kernels

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





$$(x, y) \rightarrow (x, y, x^2 + y^2)$$

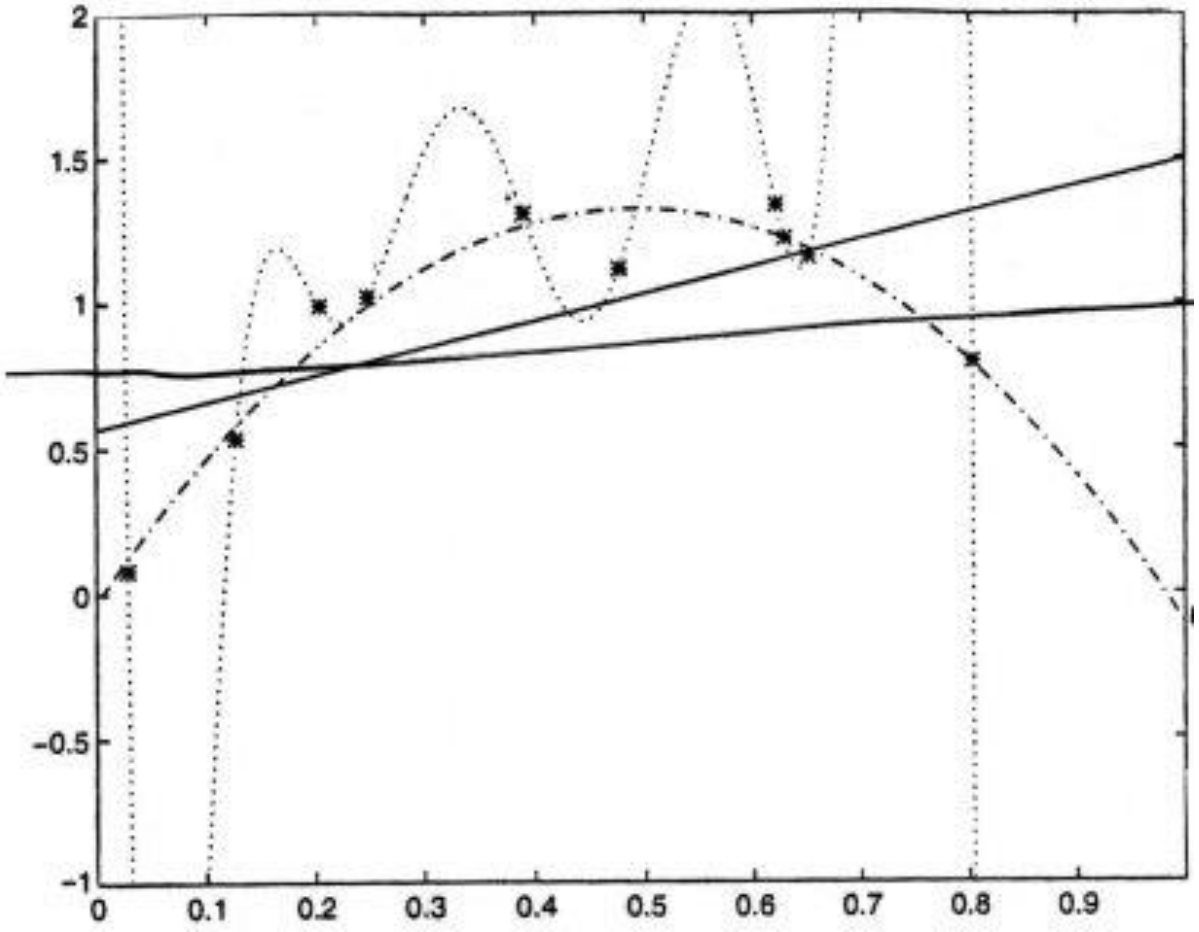
What do we change transforming the coordinates into?

$$(x, y) \rightarrow (x, y, x^2 + y^2)$$

- Through kernels we express our function using a very rich set of features (e.g. we include correlations among the original space input)

- We increase the set of possible learnable functions.

Hypothesis spaces



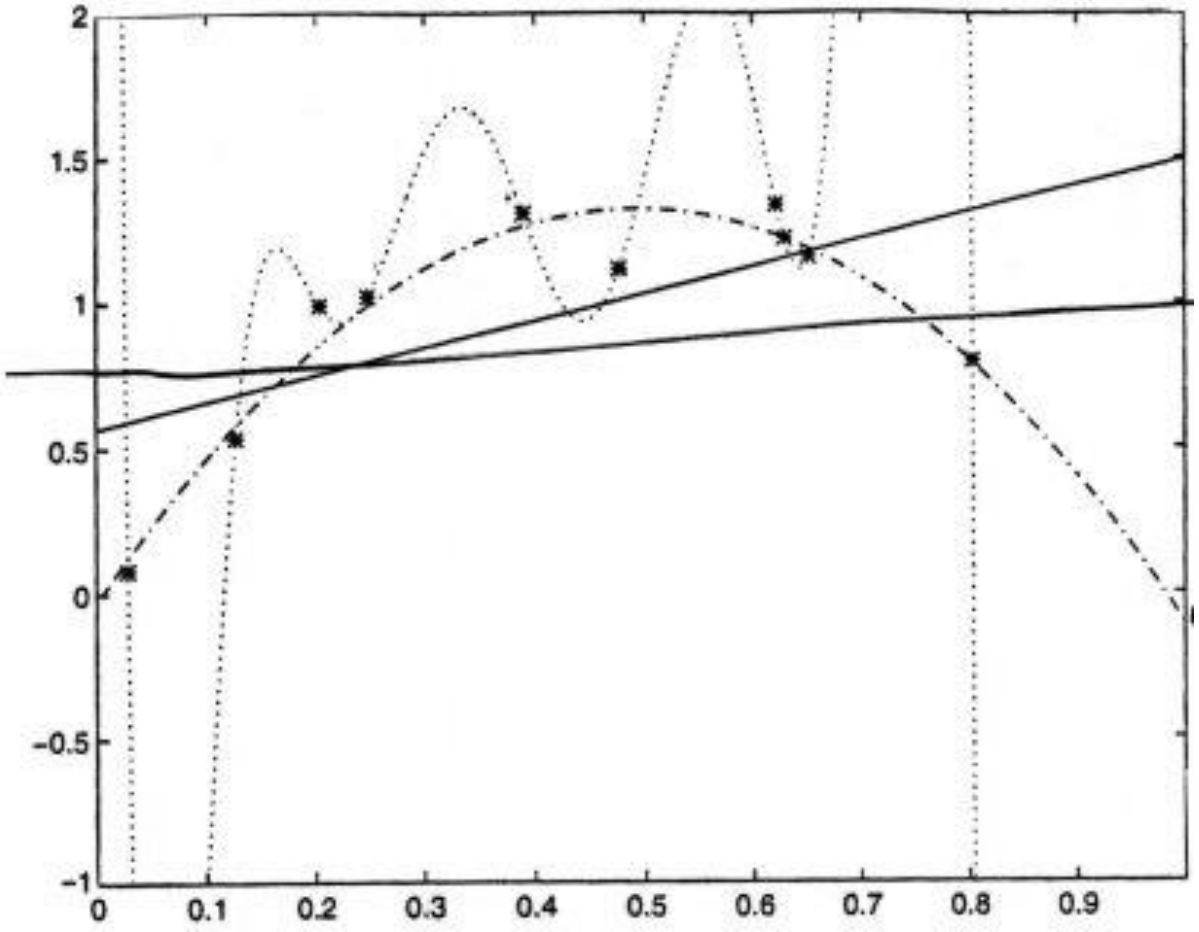
$$H_0 : f(x) = c$$

$$H_1 : f(x) = a + bx$$

$$H_2 : f(x) = a + bx + cx^2$$

What is the kernel Hypothesis space?

Hypothesis spaces



$$H_0 : f(x) = c$$

$$H_1 : f(x) = a + bx$$

$$H_2 : f(x) = a + bx + cx^2$$

$$H_K : f(x) = \sum_i \alpha_i K(x, x_i)$$

Hypothesis spaces

- Linear separable

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

We learn the separator on data

- Non-linear separable

$$f(x) = w^T \phi(x) = \sum_i \alpha_i K(x, x_i)$$

We learn the separator on a priori hand-crafted features

What is the limitation for kernels?

Hint: how do you choose K?

- Can we learn the features together with the separator?
- Which features?

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

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

How big is this Hypothesis Space?

Universal approximation theorem

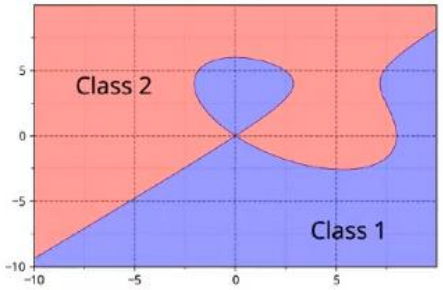
Any **continuous function** defined in the **n-dimensional unit hypercube** may be approximated by a **finite sum** of the type:

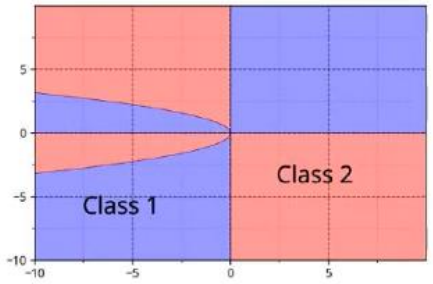
$$\sum_{j=1}^{N} v_j \, \varphi\left(\vec{\omega}^{(j)} \cdot \vec{x} + b_j\right),$$

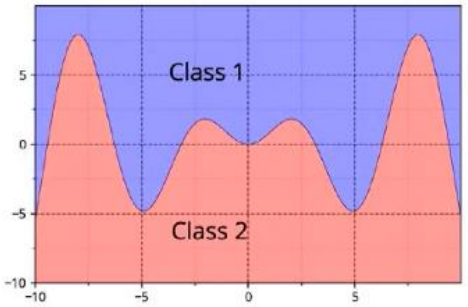
wherein $v_j, b_j \in \mathbb{R}$, $\vec{\omega}^{(j)} \in \mathbb{R}^n$, and φ is a **continuous discriminatory function**.

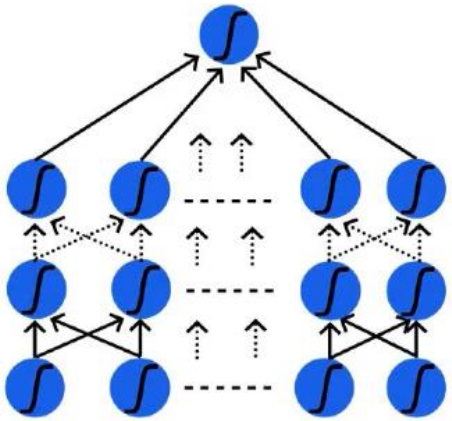
How big is N?

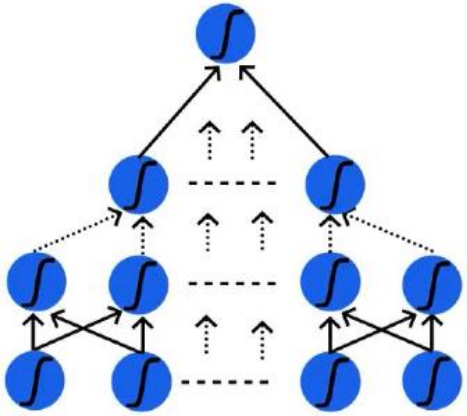
Universal approximation theorem

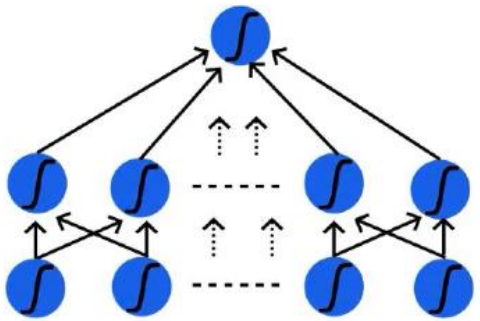






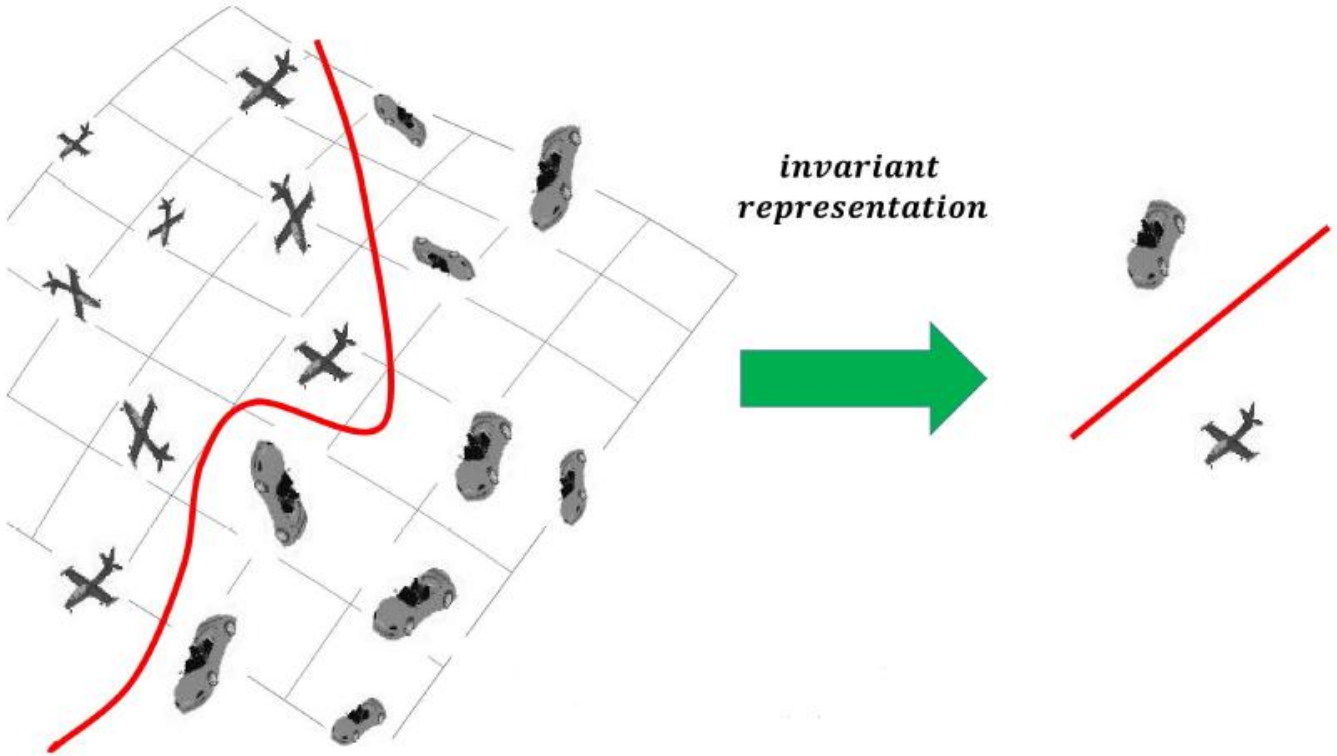




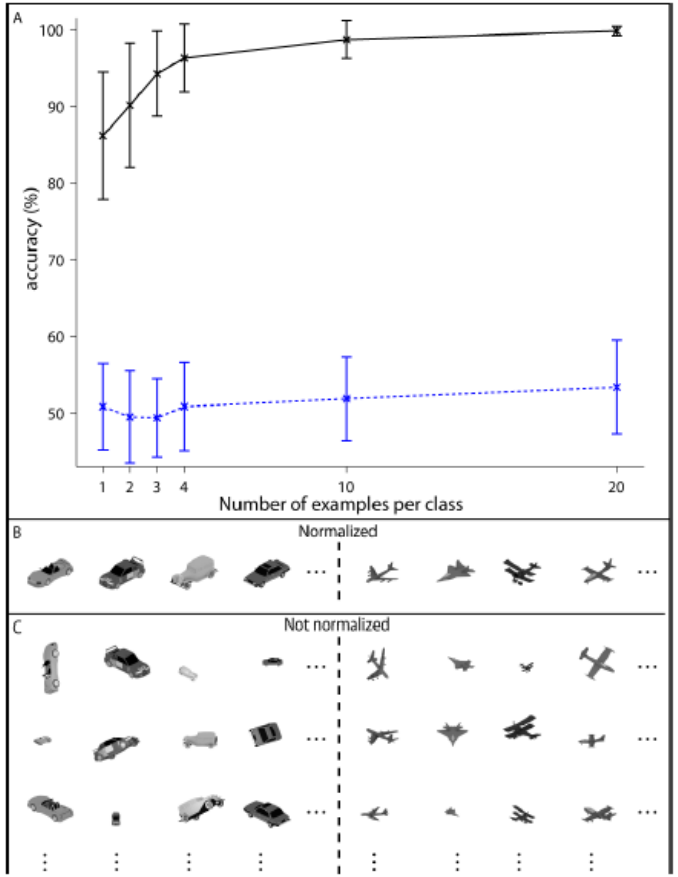


So: choose an ANN and that’s it?

Importance of choosing the right space of functions: example with invariance



Importance of choosing the right space of functions: example with invariance

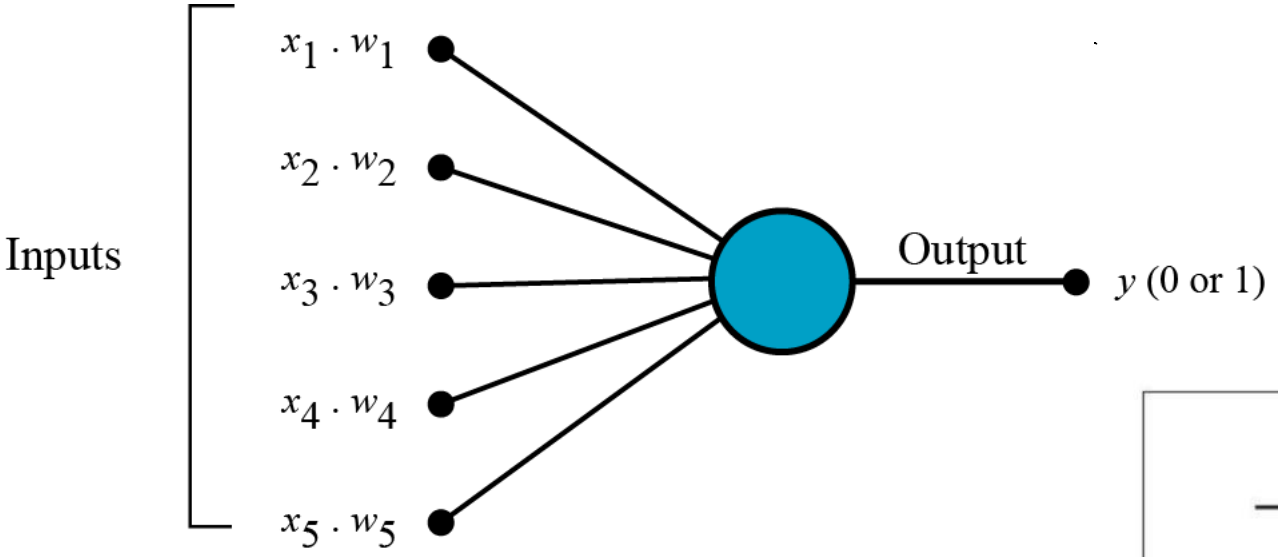


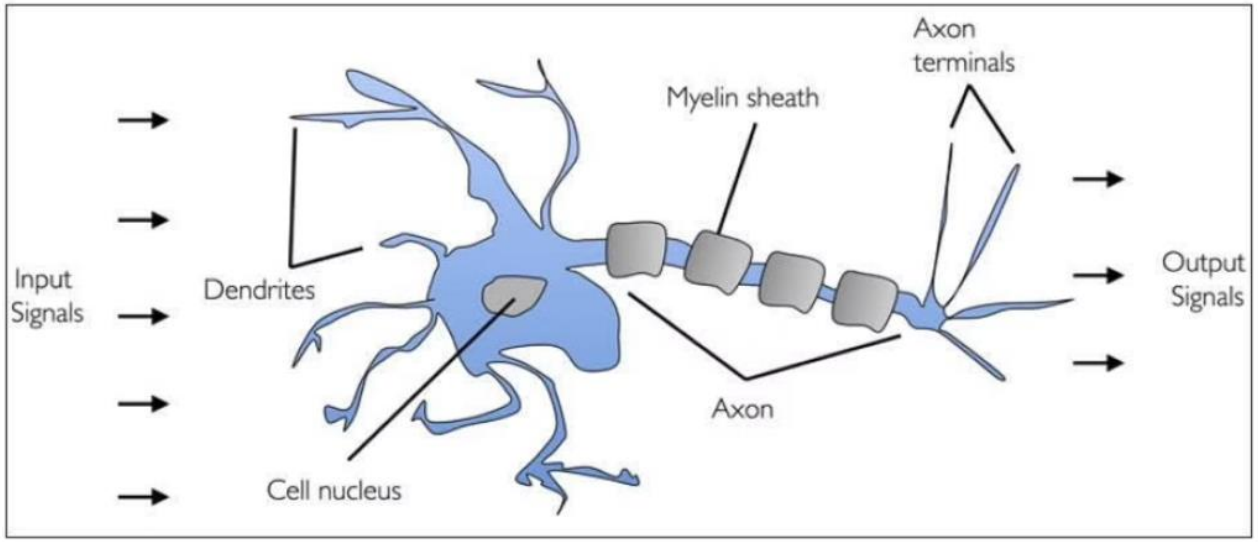
Sample complexity

- The **sample complexity** of a machine learning algorithm represents the number of training-samples that it needs in order to successfully learn a target function.

- Ex: invariant representations decrease the sample complexity

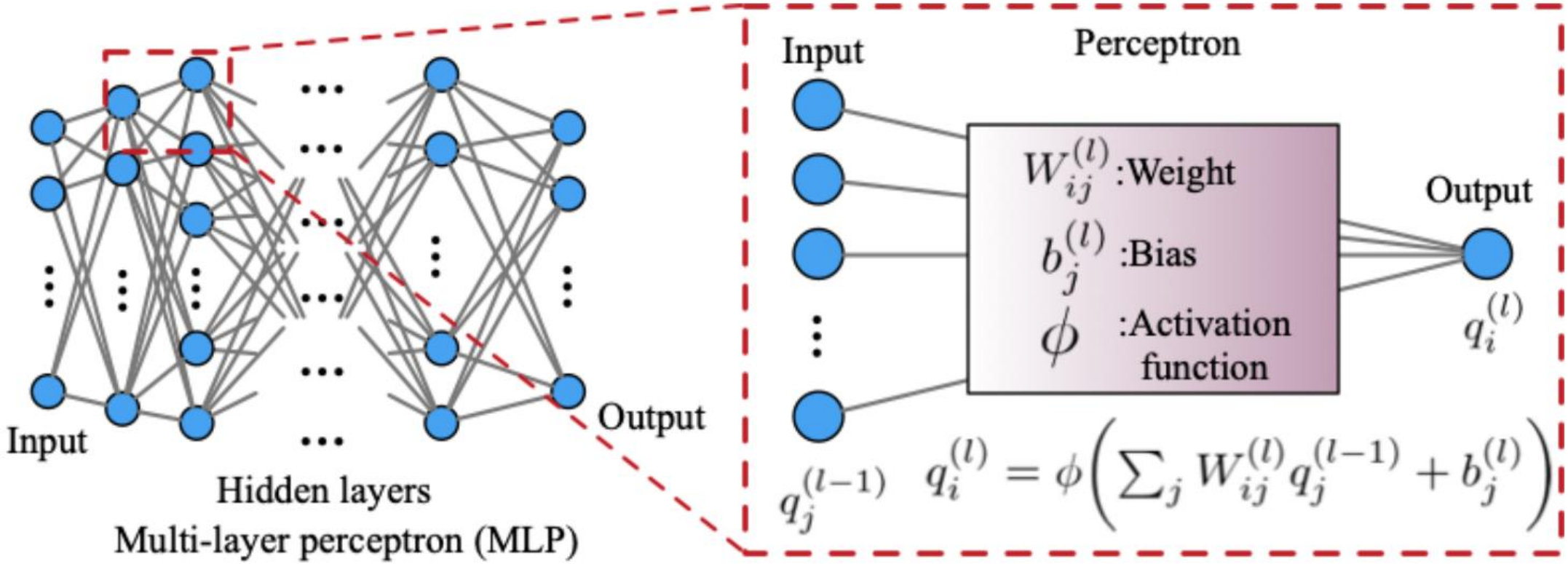
Simplest neural network: perceptron



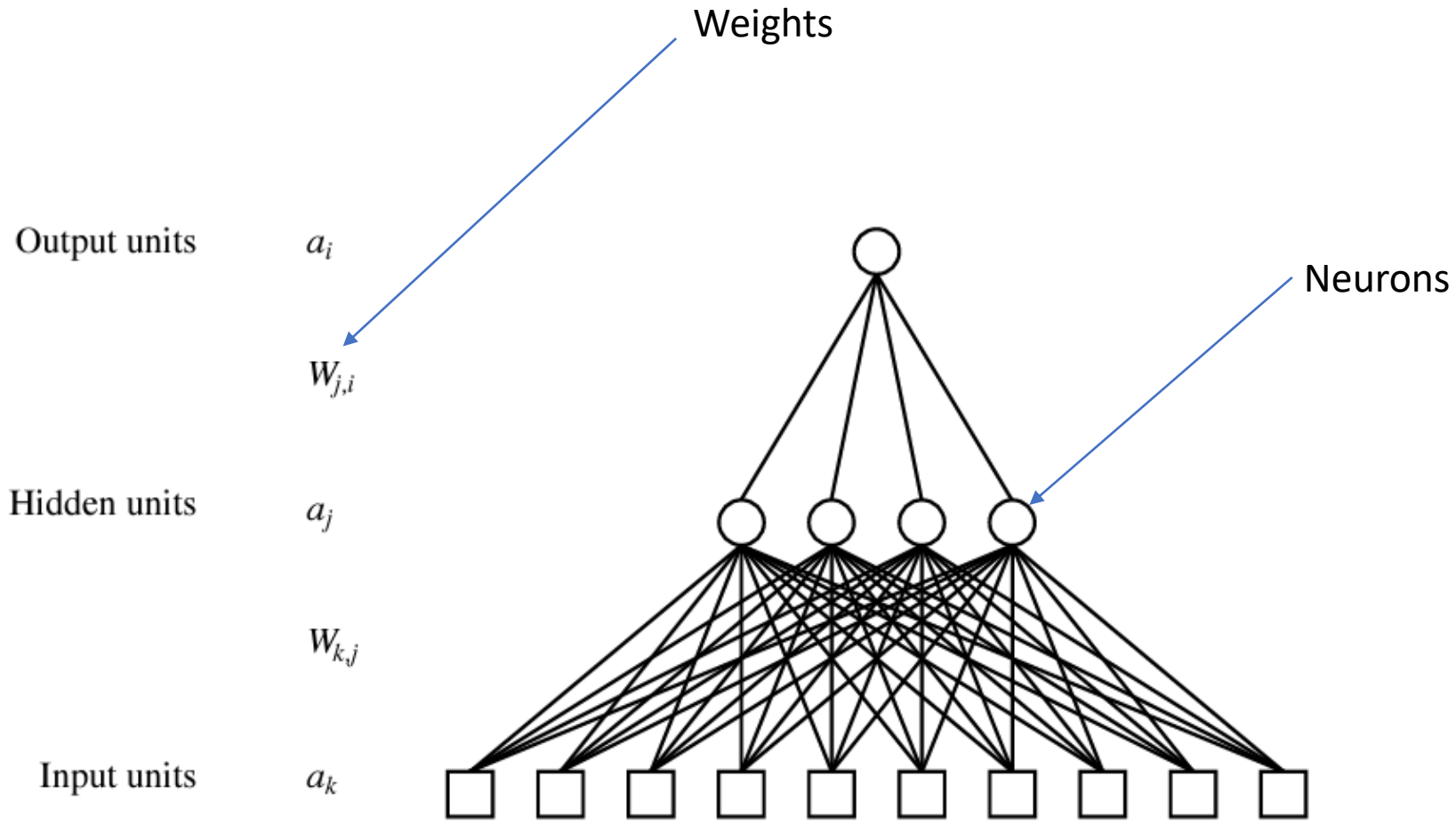


$$f(x) = sgn(w^T x + b)$$

Generalization: multi layer perceptron



Terminology



(1):Linear filtering

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

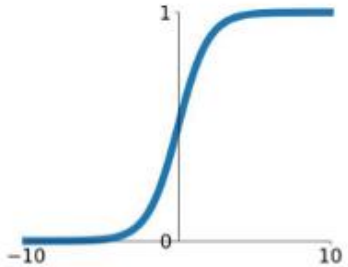
(2):Non-Linear pointwise operation

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

A zoo of “activation functions”

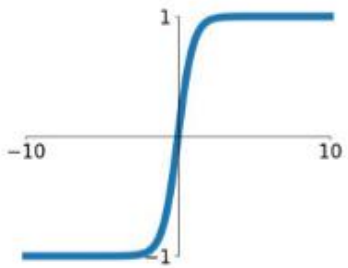
Sigmoid

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



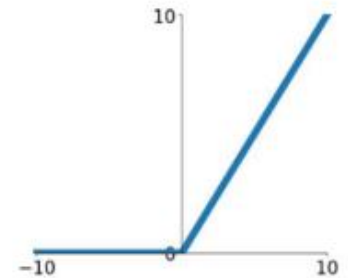
tanh

$$\tanh(x)$$



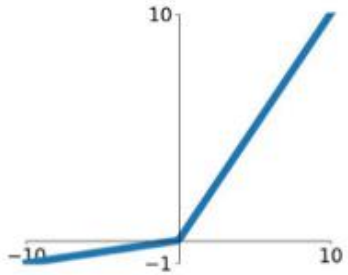
ReLU

$$\max(0, x)$$



Leaky ReLU

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

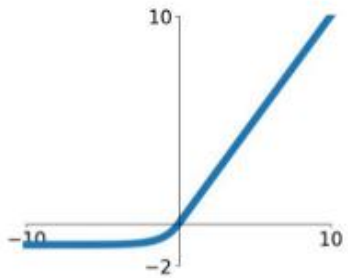


Maxout

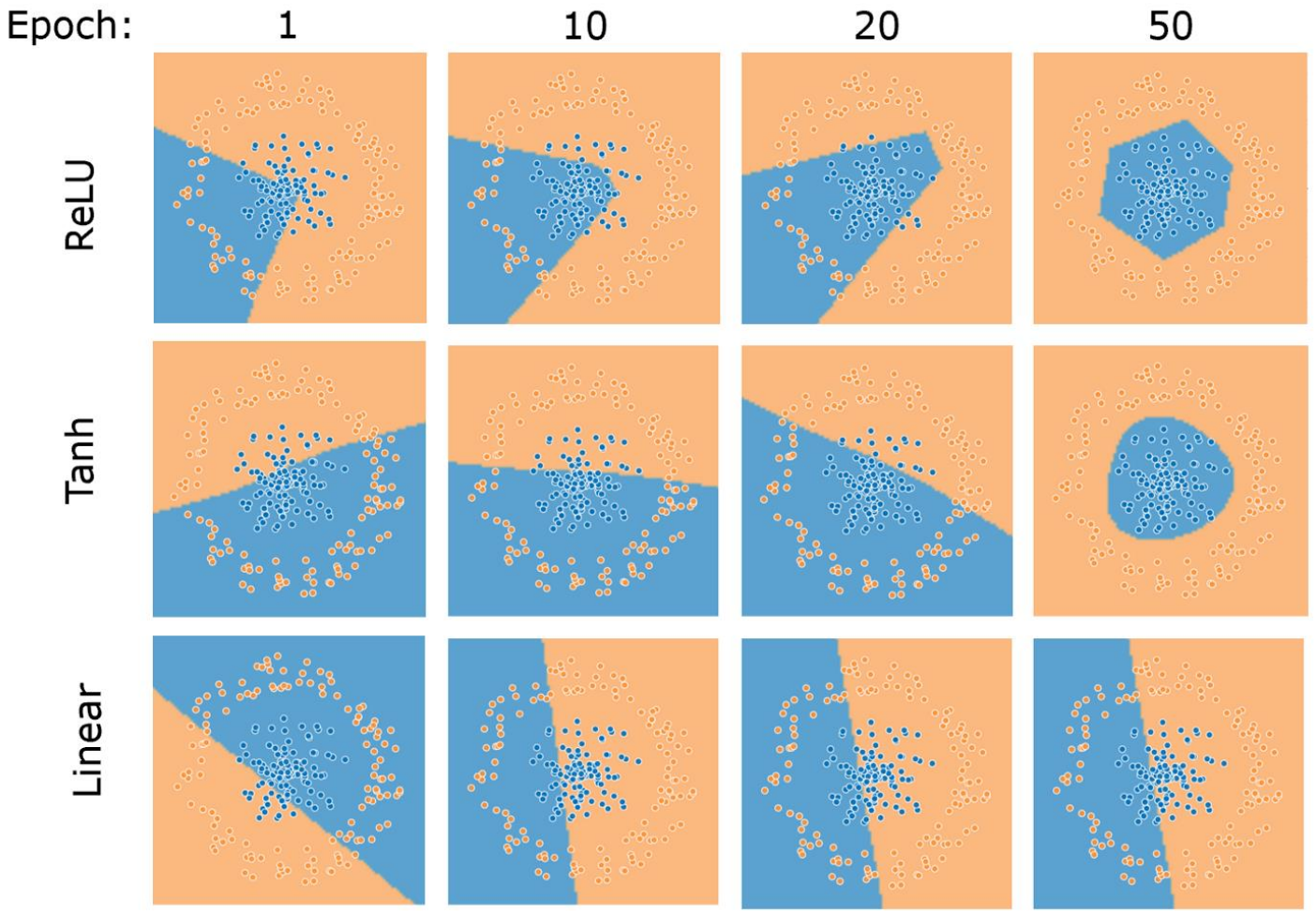
$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

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

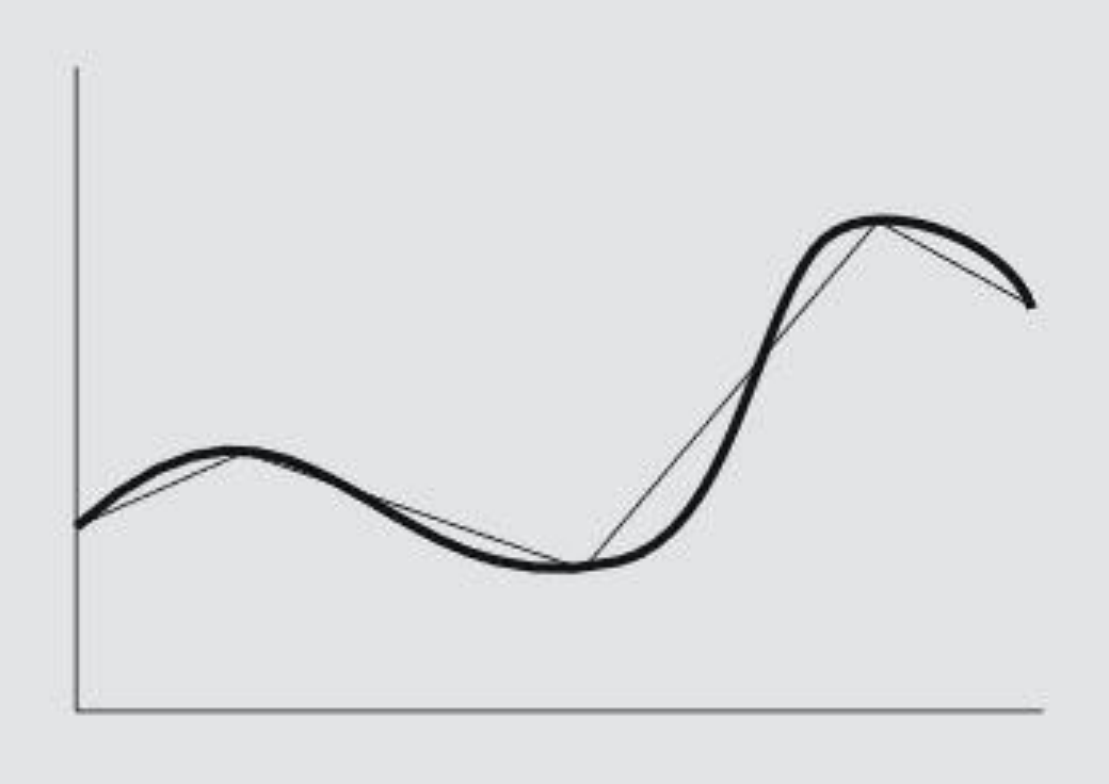


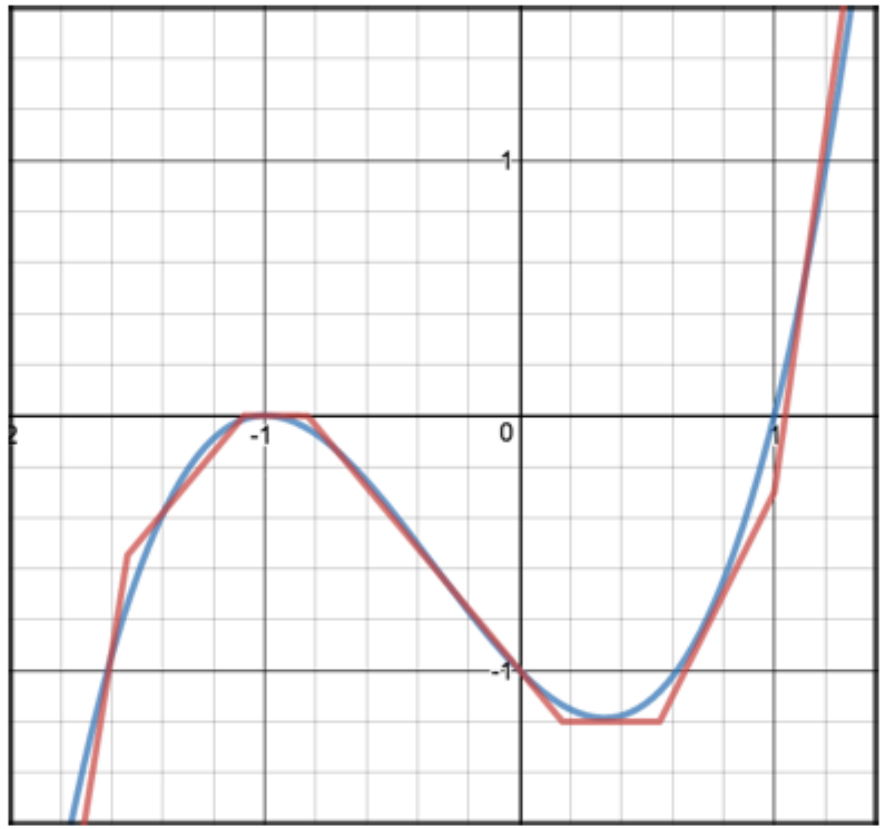
Examples



Intuition with a ReLu network

$$f(x) = w^T \sigma(W^T x) = \sum_i w_i \max(\widetilde{W}_i^T x + b_i, 0)$$



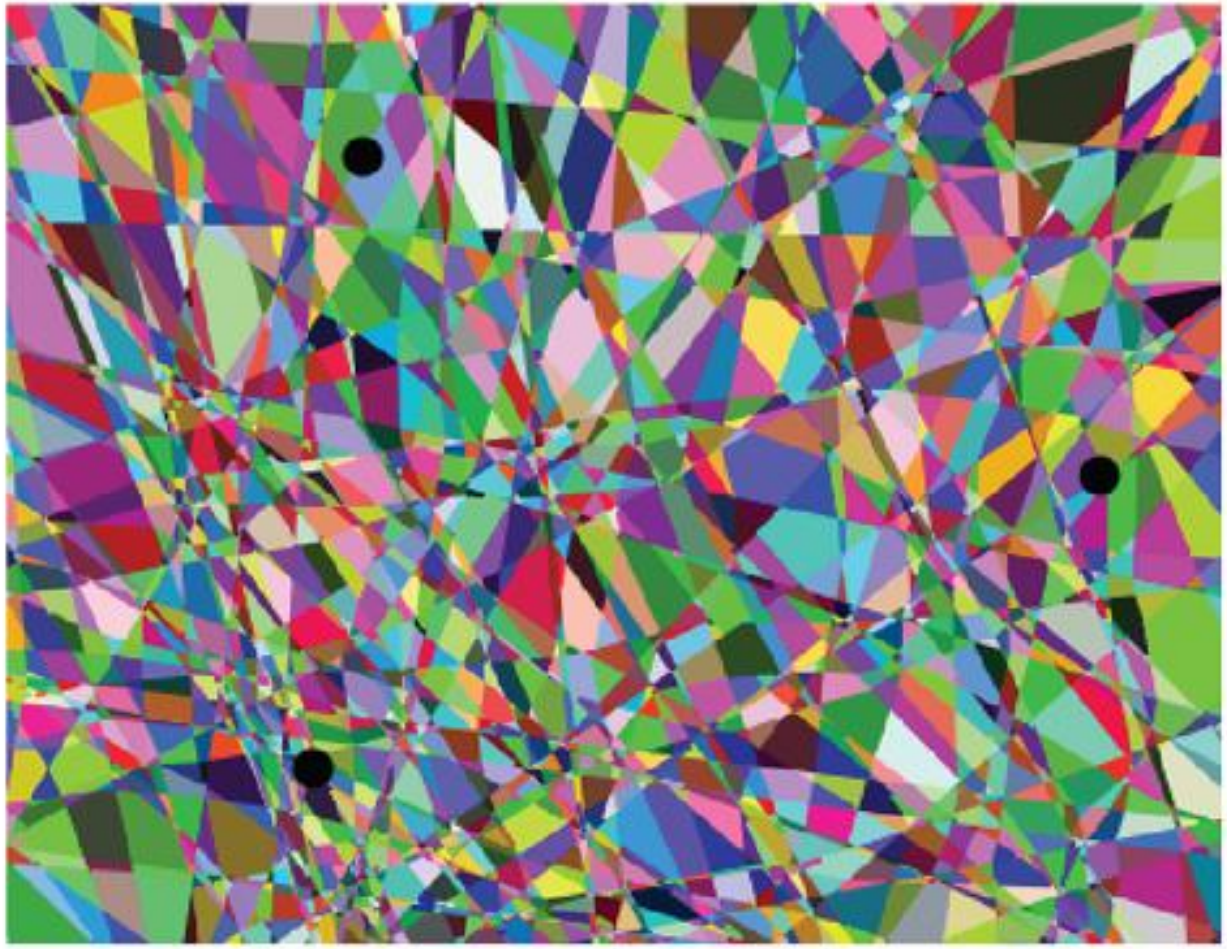


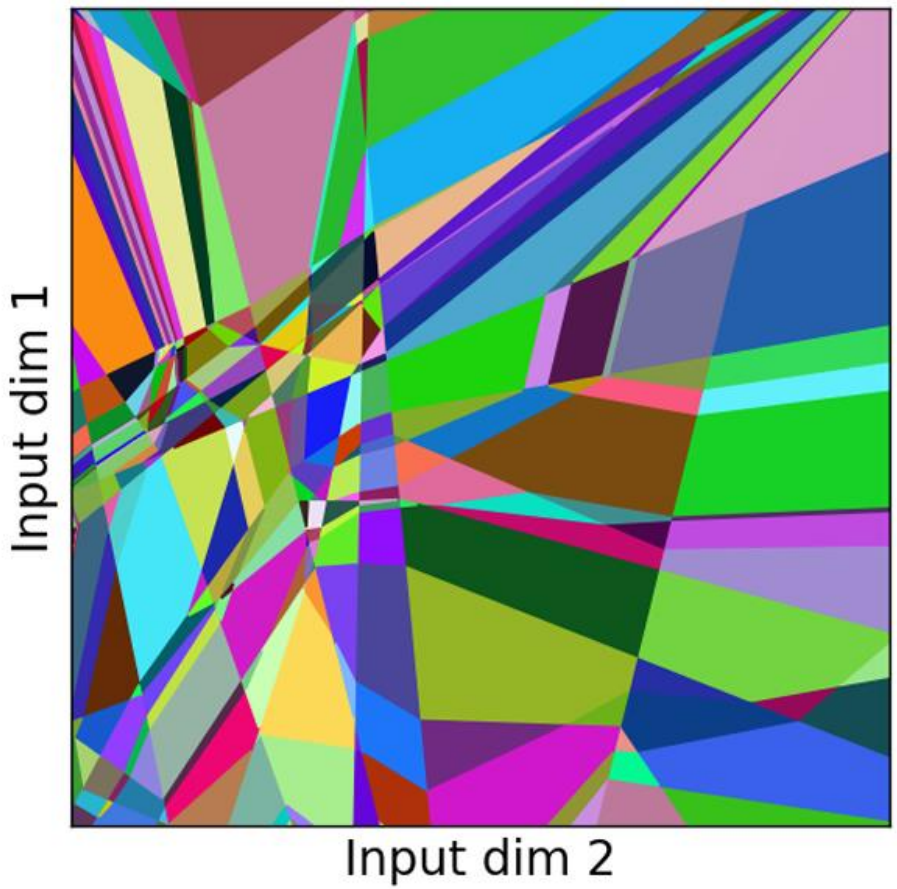
$$n_1(x) = Relu(-5x - 7.7)$$
$$n_2(x) = Relu(-1.2x - 1.3)$$
$$n_3(x) = Relu(1.2x + 1)$$
$$n_4(x) = Relu(1.2x - .2)$$
$$n_5(x) = Relu(2x - 1.1)$$
$$n_6(x) = Relu(5x - 5)$$

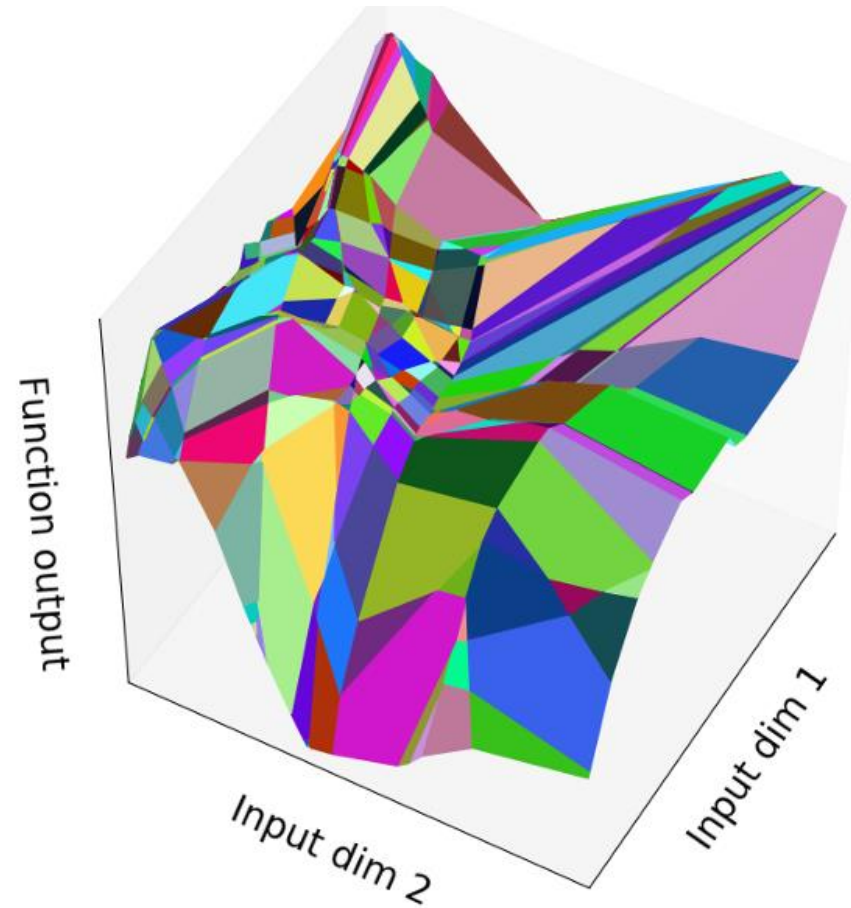
$$Z(x) = -n_1(x) - n_2(x) - n_3(x) + n_4(x) + n_5(x) + n_6(x)$$

2 Dimensional

$$f(x) = w^T \sigma(W^T x) = \sum_i w_i \max(\widetilde{W}_i^T x + b_i, 0)$$




Input dim 1
Input dim 2


Function output
Input dim 2
Input dim 1

Loss functions examples: penalizing errors

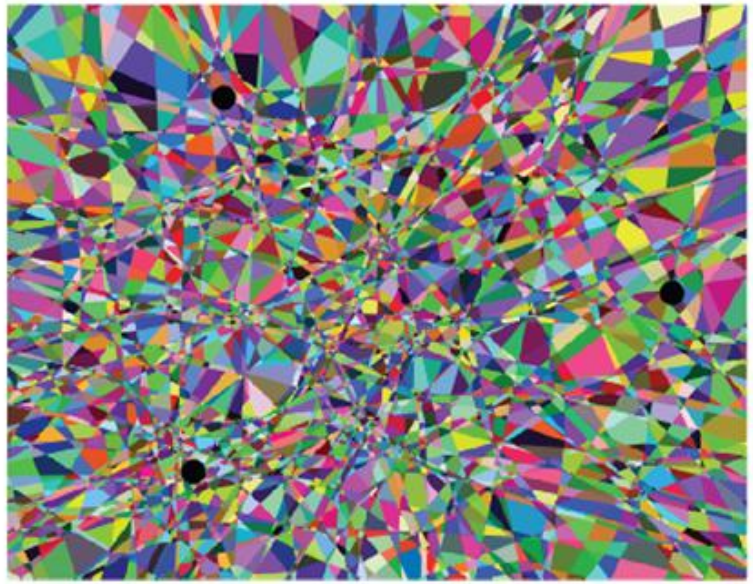
$$\mathcal{L}(w,b) = \frac{1}{N}\sum_i (y_i - (w^T x + b))^2$$

What is the main difference from an optimization point of view?

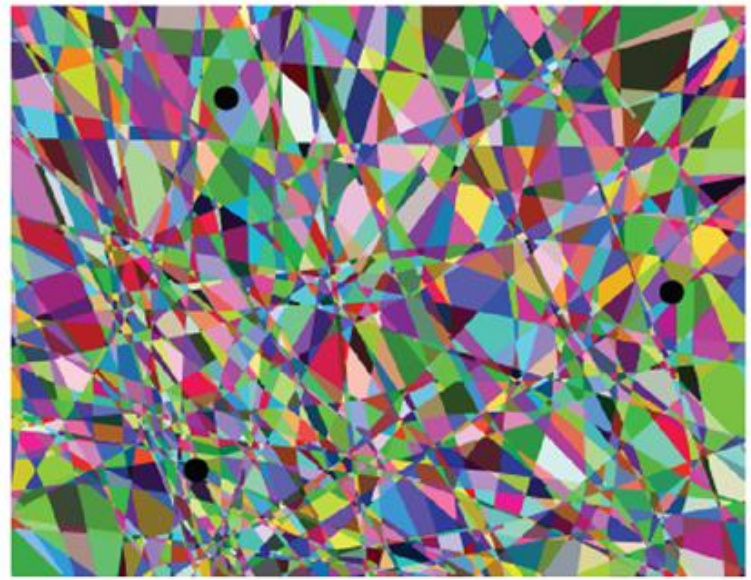
$$\mathcal{L}(w,W) = \frac{1}{N}\sum_{i=1}^{N} \left(y_i - w^T \sigma(W^T x)\right)^2$$

The Loss is minimized by gradient descent

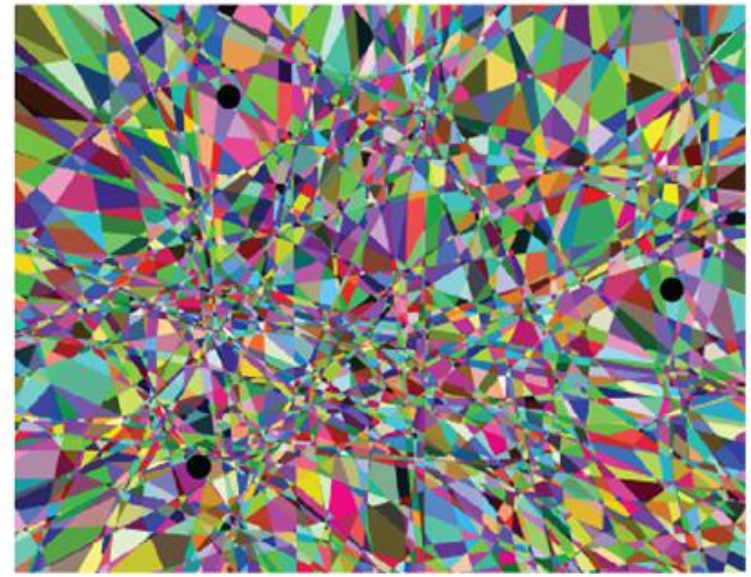
Epoch 0: 9744 regions



Epoch 1: 4196 regions



Epoch 20: 8541 regions

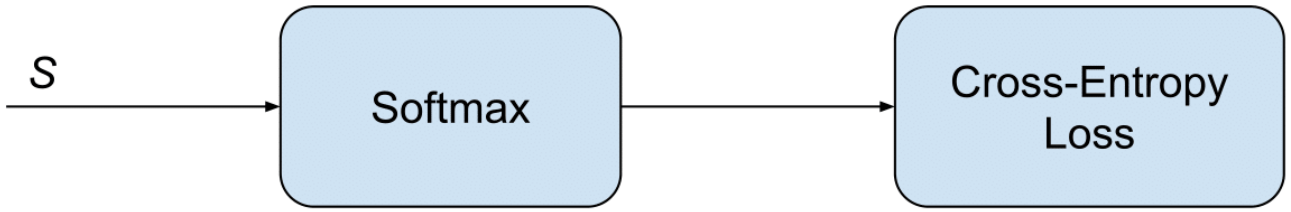


Other examples

$\mathcal{L}_1$	L_1 loss	$\|\mathbf{y}-\mathbf{o}\|_1$
$\mathcal{L}_2$	L_2 loss	$\|\mathbf{y}-\mathbf{o}\|_2^2$
$\mathcal{L}_1 \circ \sigma$	expectation loss	$\|\mathbf{y}-\sigma(\mathbf{o})\|_1$
$\mathcal{L}_2 \circ \sigma$	regularised expectation loss[1]	$\|\mathbf{y}-\sigma(\mathbf{o})\|_2^2$
$\mathcal{L}_\infty \circ \sigma$	Chebyshev loss	$\max_j \lvert\sigma(\mathbf{o})^{(j)}-\mathbf{y}^{(j)}\rvert$
hinge	hinge [13] (margin) loss	$\sum_j \max(0, \frac{1}{2}-\hat{\mathbf{y}}^{(j)}\mathbf{o}^{(j)})$
hinge^2	squared hinge (margin) loss	$\sum_j \max(0, \frac{1}{2}-\hat{\mathbf{y}}^{(j)}\mathbf{o}^{(j)})^2$
hinge^3	cubed hinge (margin) loss	$\sum_j \max(0, \frac{1}{2}-\hat{\mathbf{y}}^{(j)}\mathbf{o}^{(j)})^3$
log	log (cross entropy) loss	$-\sum_j \mathbf{y}^{(j)} \log \sigma(\mathbf{o})^{(j)}$
log^2	squared log loss	$-\sum_j [\mathbf{y}^{(j)} \log \sigma(\mathbf{o})^{(j)}]^2$
tan	Tanimoto loss	$\frac{-\sum_j \sigma(\mathbf{o})^{(j)}\mathbf{y}^{(j)}}{\|\sigma(\mathbf{o})\|_2^2+\|\mathbf{y}\|_2^2-\sum_j \sigma(\mathbf{o})^{(j)}\mathbf{y}^{(j)}}$
D_{CS}	Cauchy-Schwarz Divergence [3]	$-\log \frac{\sum_j \sigma(\mathbf{o})^{(j)}\mathbf{y}^{(j)}}{\|\sigma(\mathbf{o})\|_2\|\mathbf{y}\|_2}$

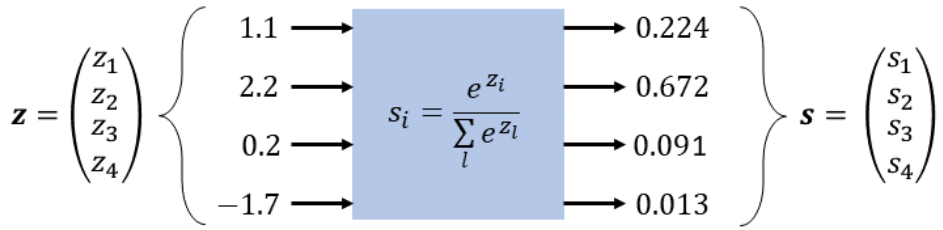
- Each loss is weighting errors in different ways and its choice depends on the problem
- Suppose e.g. we want to output a vector of probabilities for different classes…

Example of Classification Loss: cross entropy loss and softmax



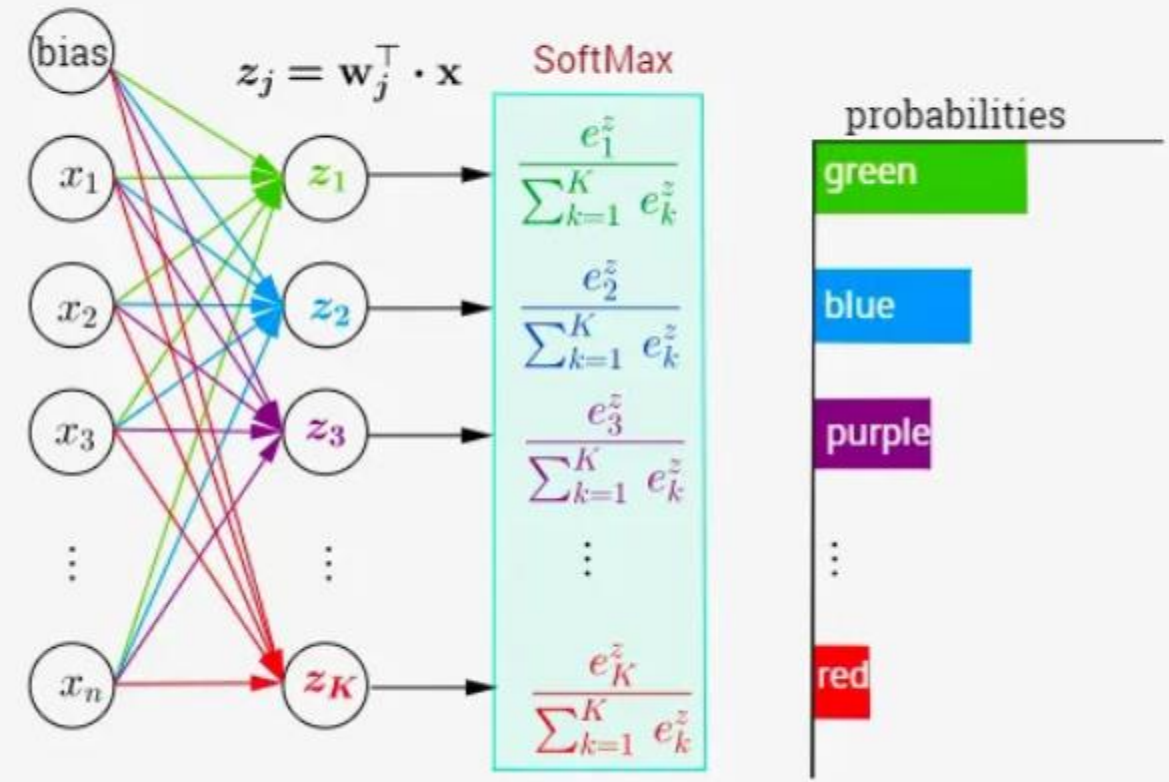
$$\hat{y}_i = \frac{e^{s_i}}{\sum_j e^{s_j}}$$

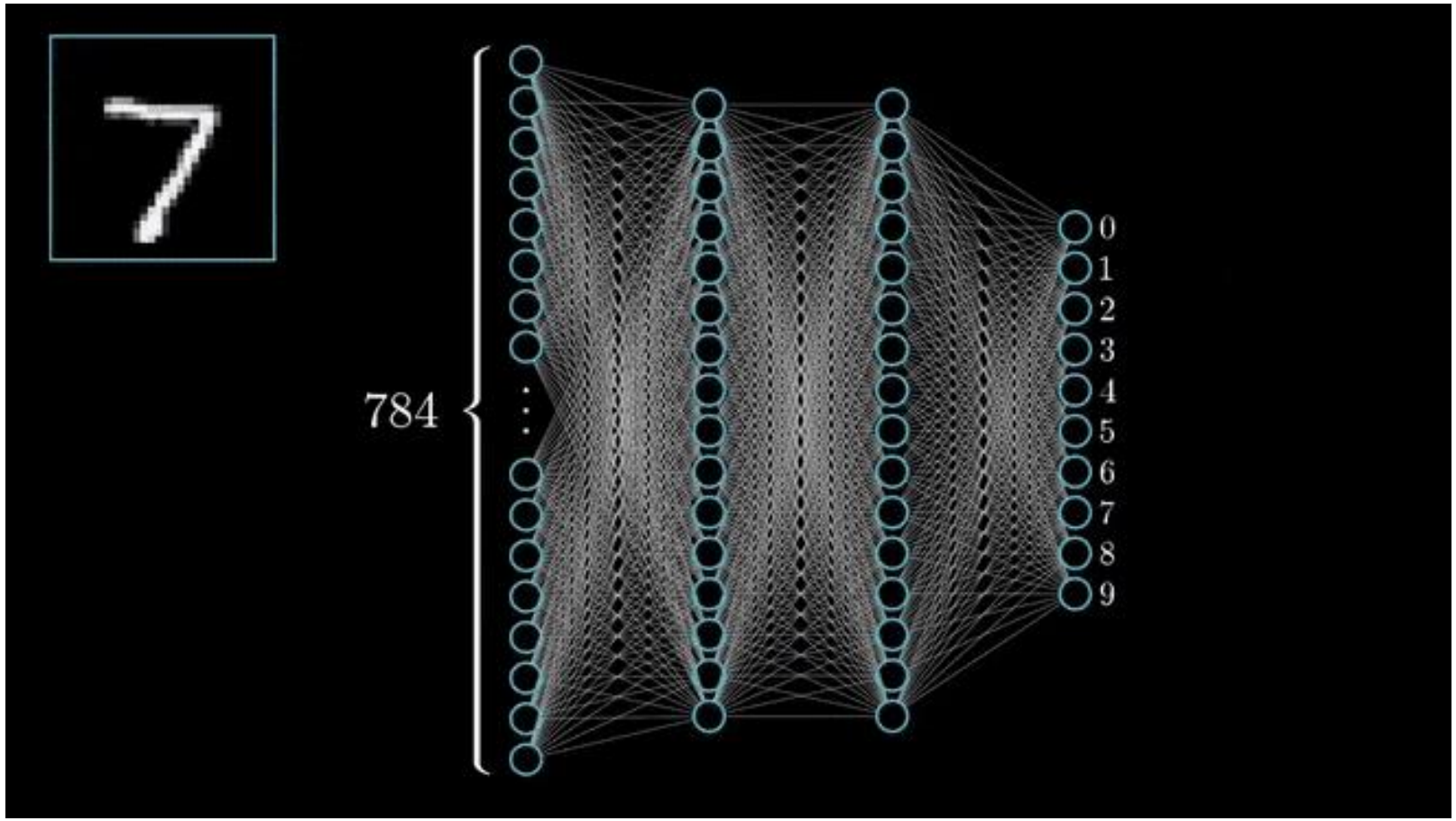
$$-\frac{1}{\text{output size}} \sum_{i=1}^{\text{output size}} y_i \cdot \log \hat{y}_i + (1 - y_i) \cdot \log (1 - \hat{y}_i)$$



Multi-Class Classification with NN and SoftMax Function

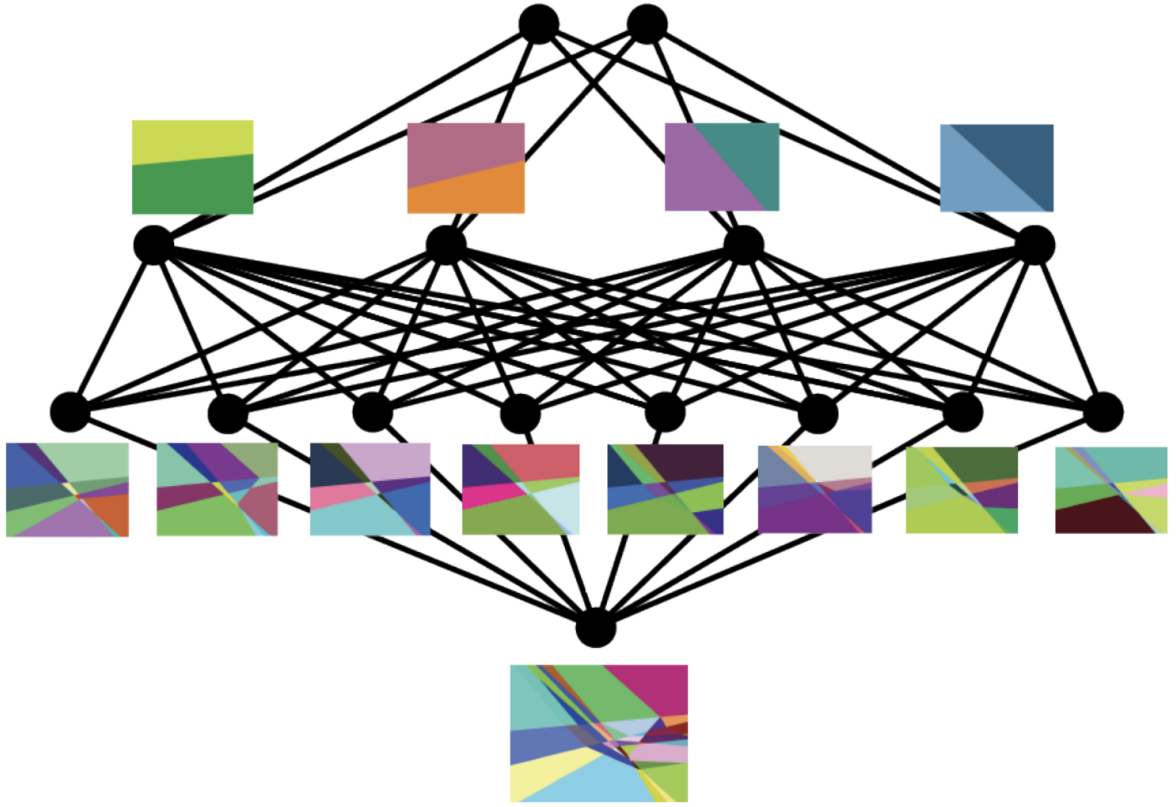
$$\mathbf{z} = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ \vdots \\ z_K \end{bmatrix} = \begin{bmatrix} \mathbf{w}_1^\top \\ \mathbf{w}_2^\top \\ \mathbf{w}_3^\top \\ \vdots \\ \mathbf{w}_K^\top \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix}$$




784
0
1
2
3
4
5
6
7
8
9

Deep networks

$$f(x) = w^T \sigma(W_2^T \sigma(W_1^T x))$$



Is the number of regions increasing linearly with depth?

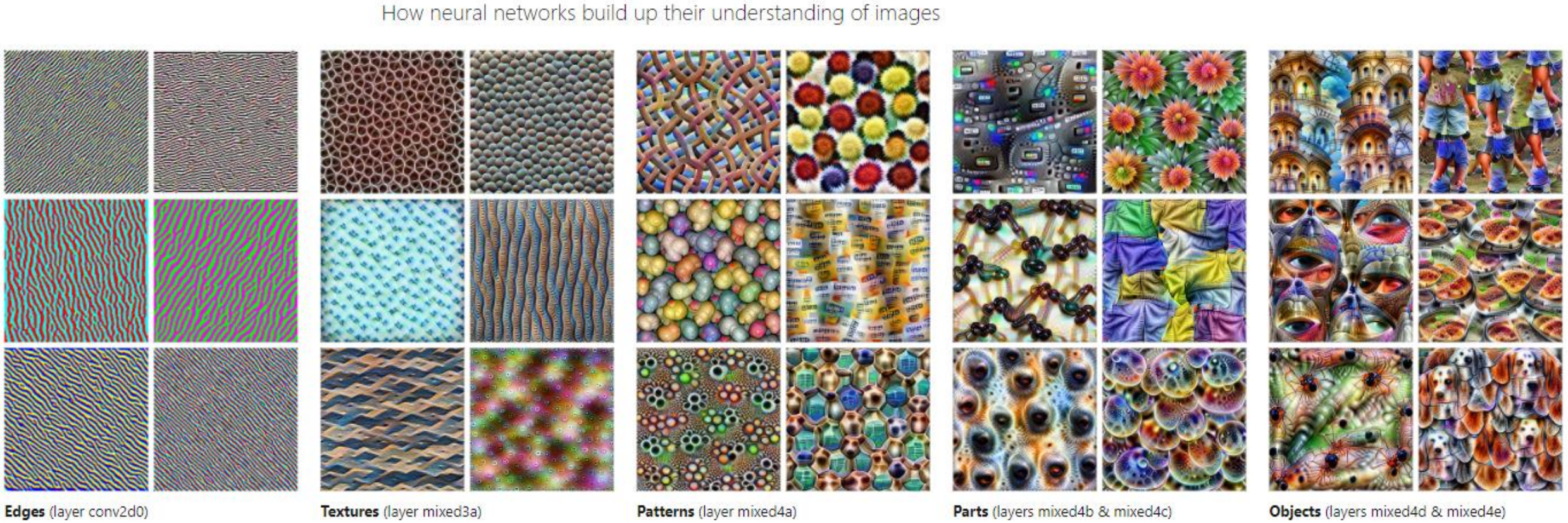
Intuition why depth: approximation reusability

From universal approximation theorem you need exponentially many units but you increase exponentially the repeated approximations with depth, reusing the previous layer approx.

Learned features

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

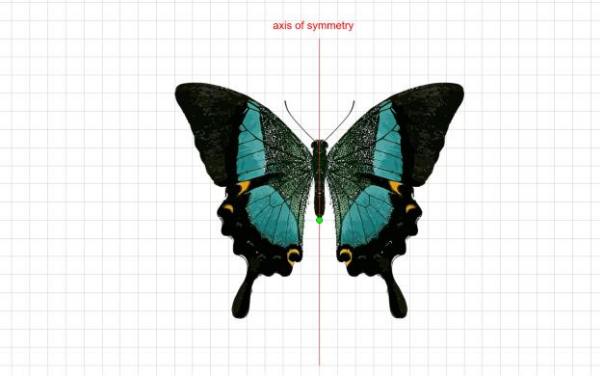
$$w \in \mathbb{R}^{k_L}, \quad W \in \mathbb{R}^{d \times k_\ell}, \quad \sigma : \mathbb{R} \to \mathbb{R}$$

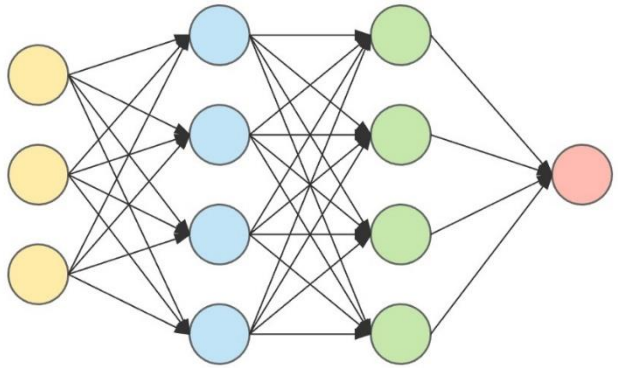


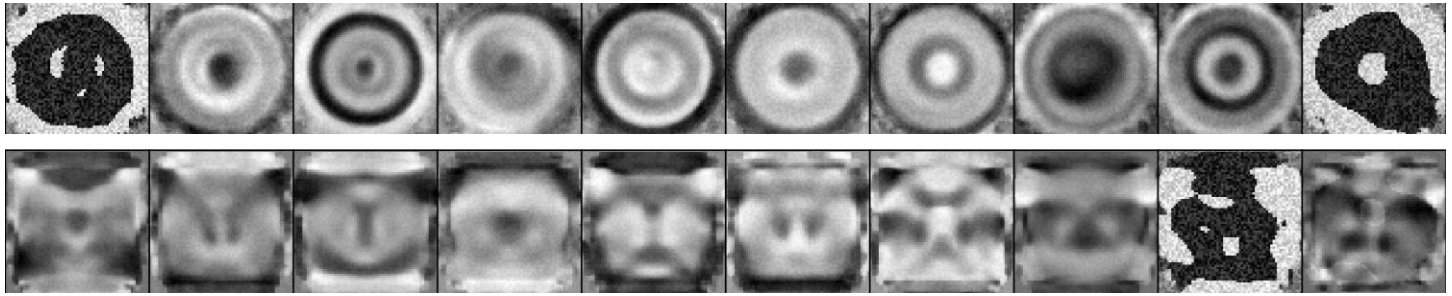
https://distill.pub/2017/feature-visualization/

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

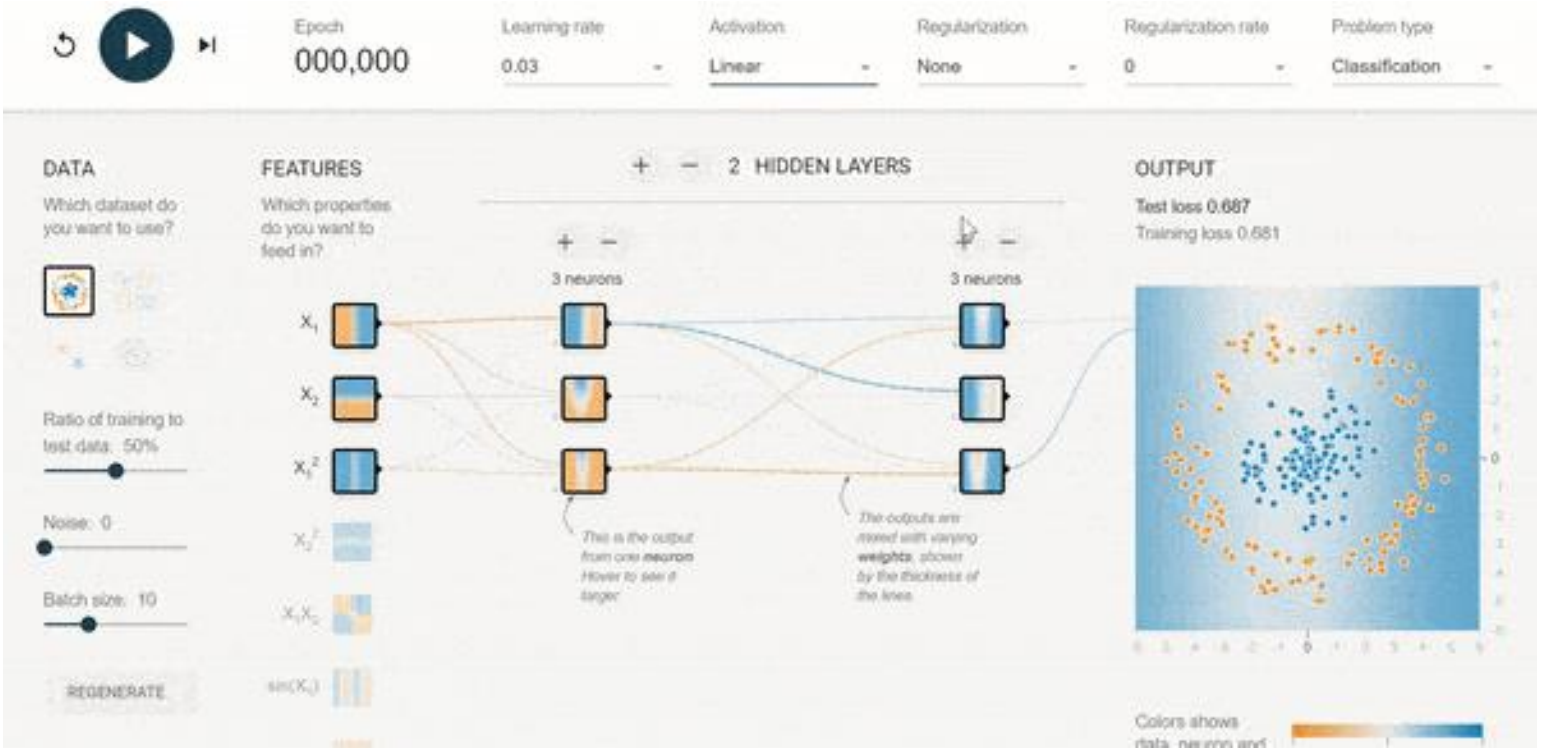
Symmetries/regularities in data/data statistics are reflected in the learned weights of a network.



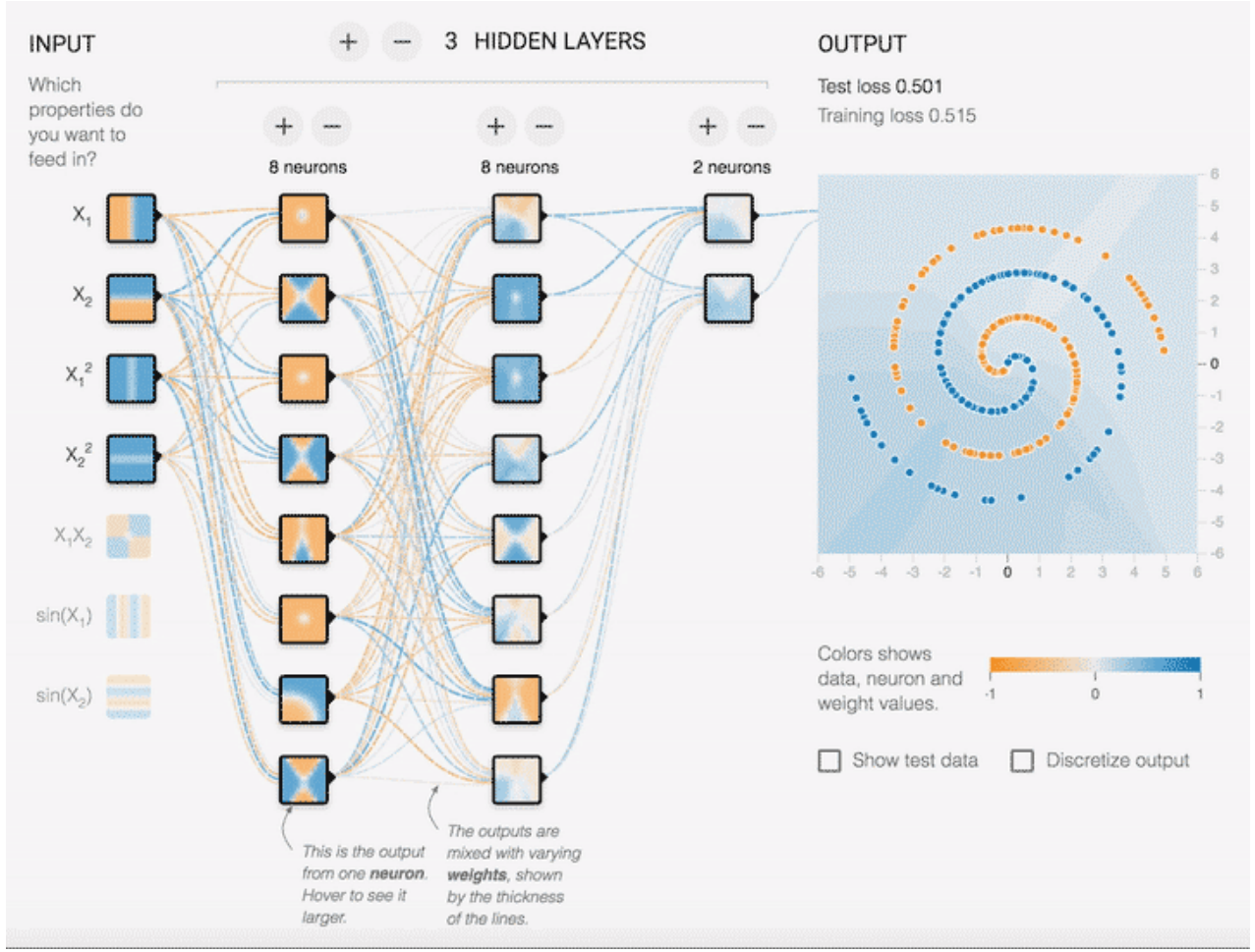




Compositional functions power



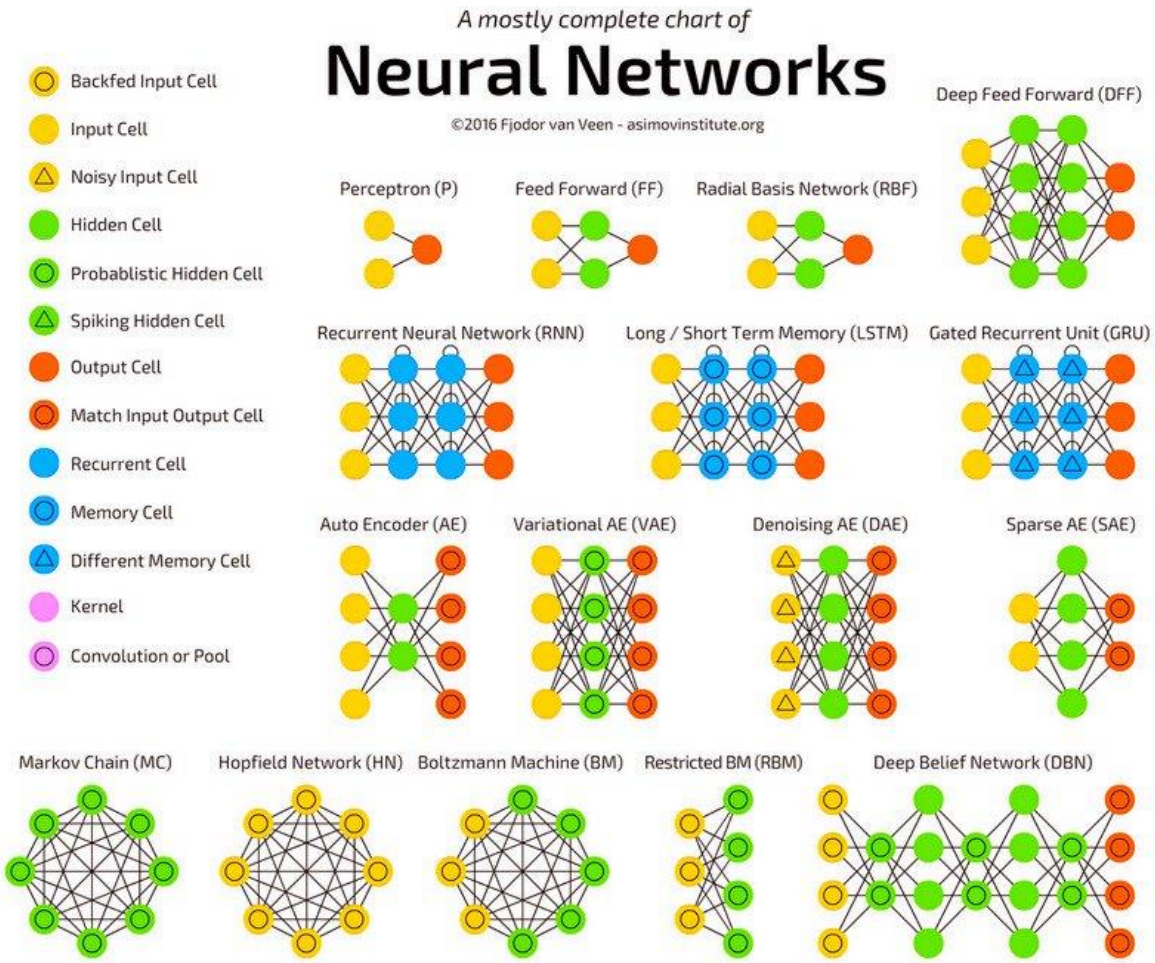
Compositional functions power

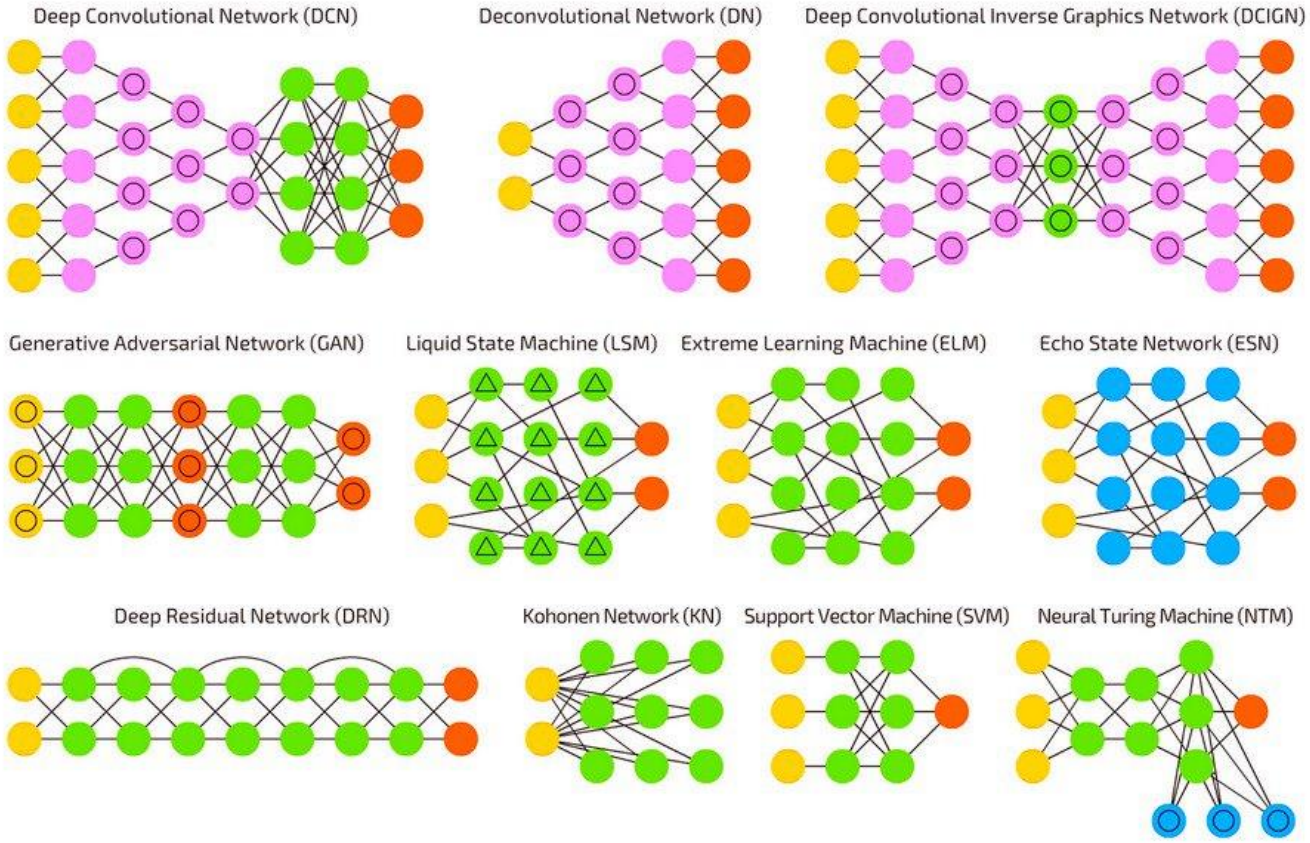


Tensor flow playground

- 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

A zoo of networks





https://modelzoo.co/

Class 2: training neural networks

- Learning: forward and backward pass

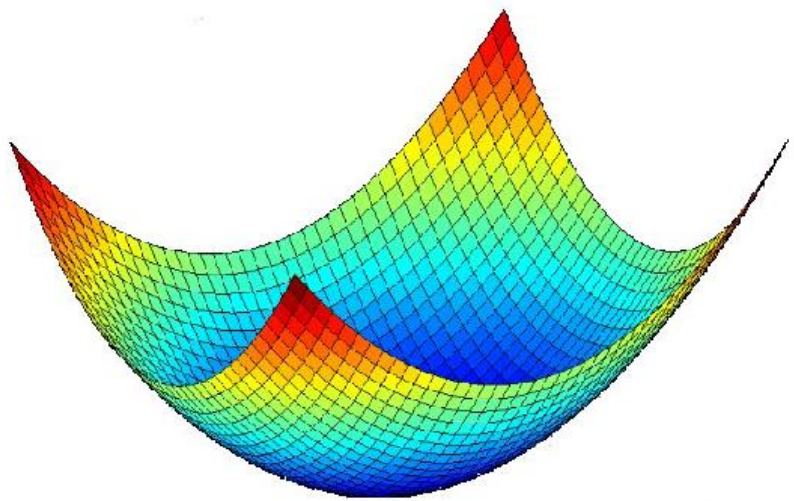
- Gradient descent and stochastic gradient descent

How do we learn w and W?

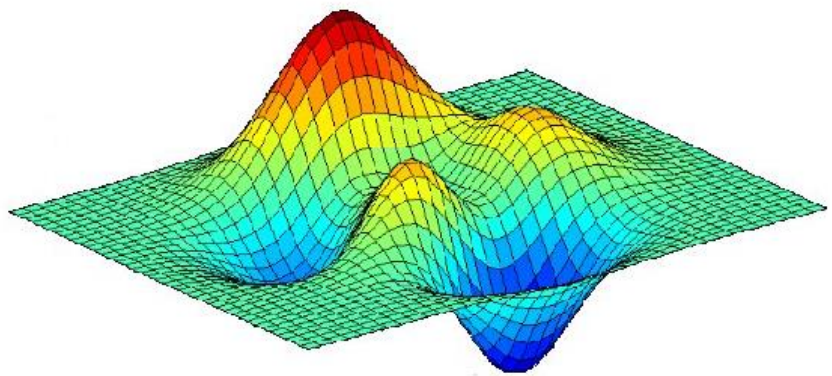
$$\mathcal{L}(w, W) = \frac{1}{N} \sum_{i=1}^{N} \left(y_i - w^T \sigma(W^T x) \right)^2$$

What's new w.r.t. the LSM?

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

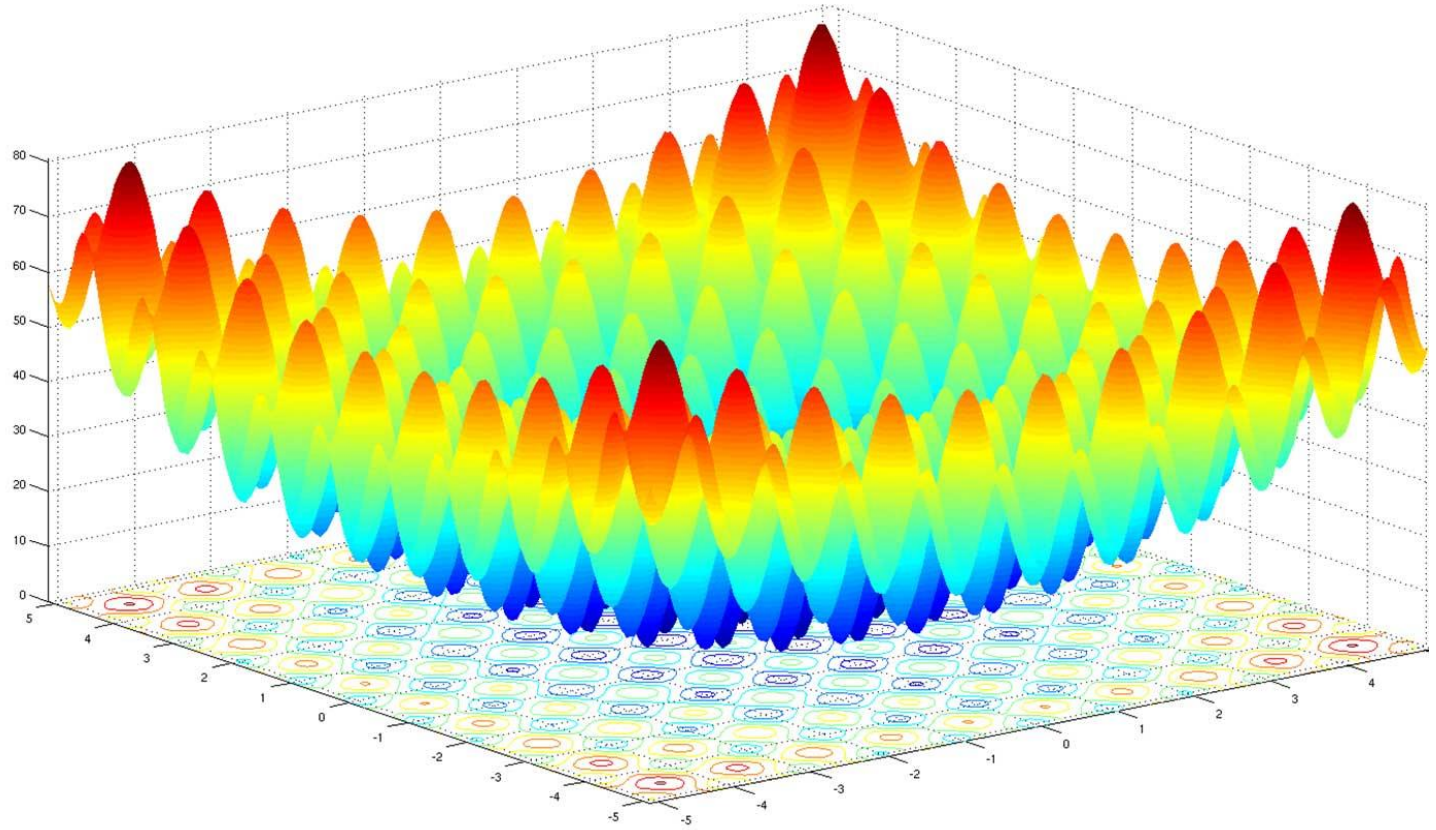


convex function



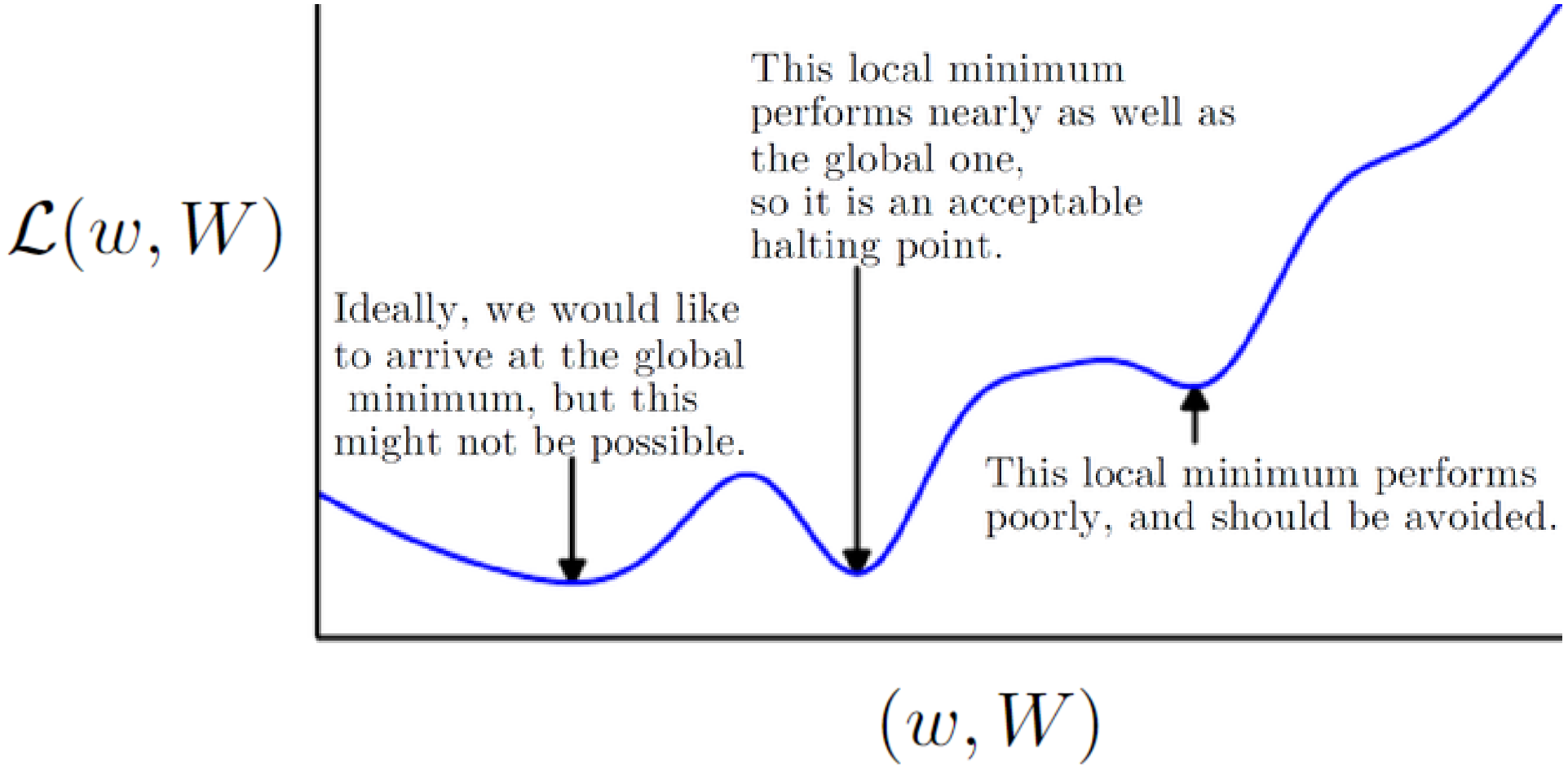
non-convex function

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

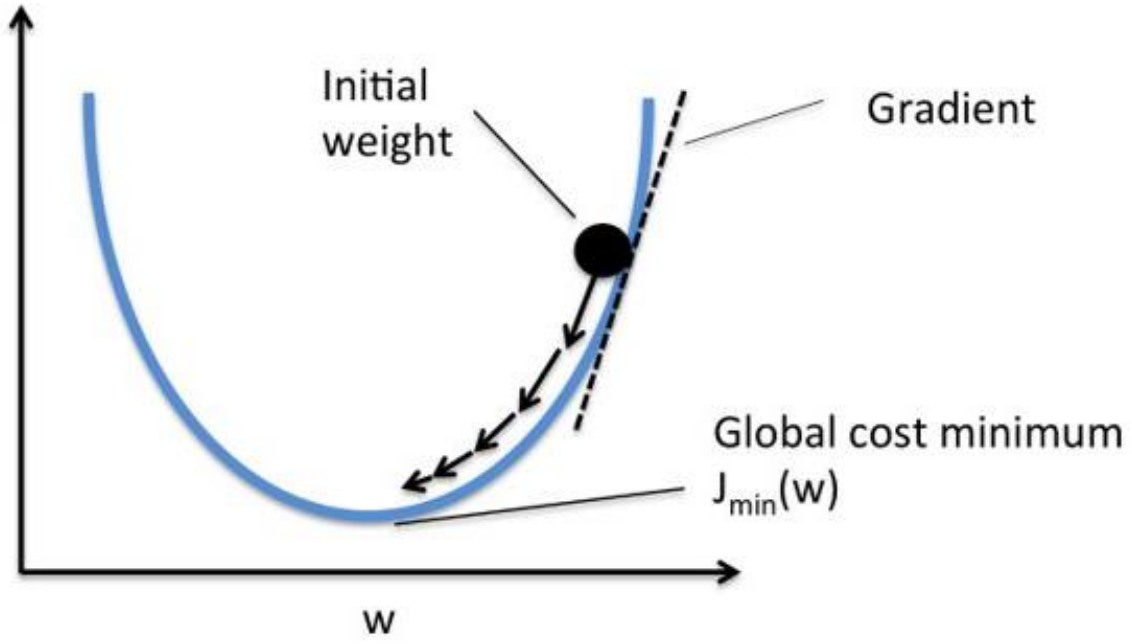


Local minima!!!!

Dependence on initialization



Gradient descent



$$w_{t+1} = w_t - \gamma \nabla_w \mathcal{L}(w, W)$$

$$W_{t+1} = w_t - \gamma \nabla_W \mathcal{L}(w, W)$$

For neural networks the procedure is called backpropagation

GD variations

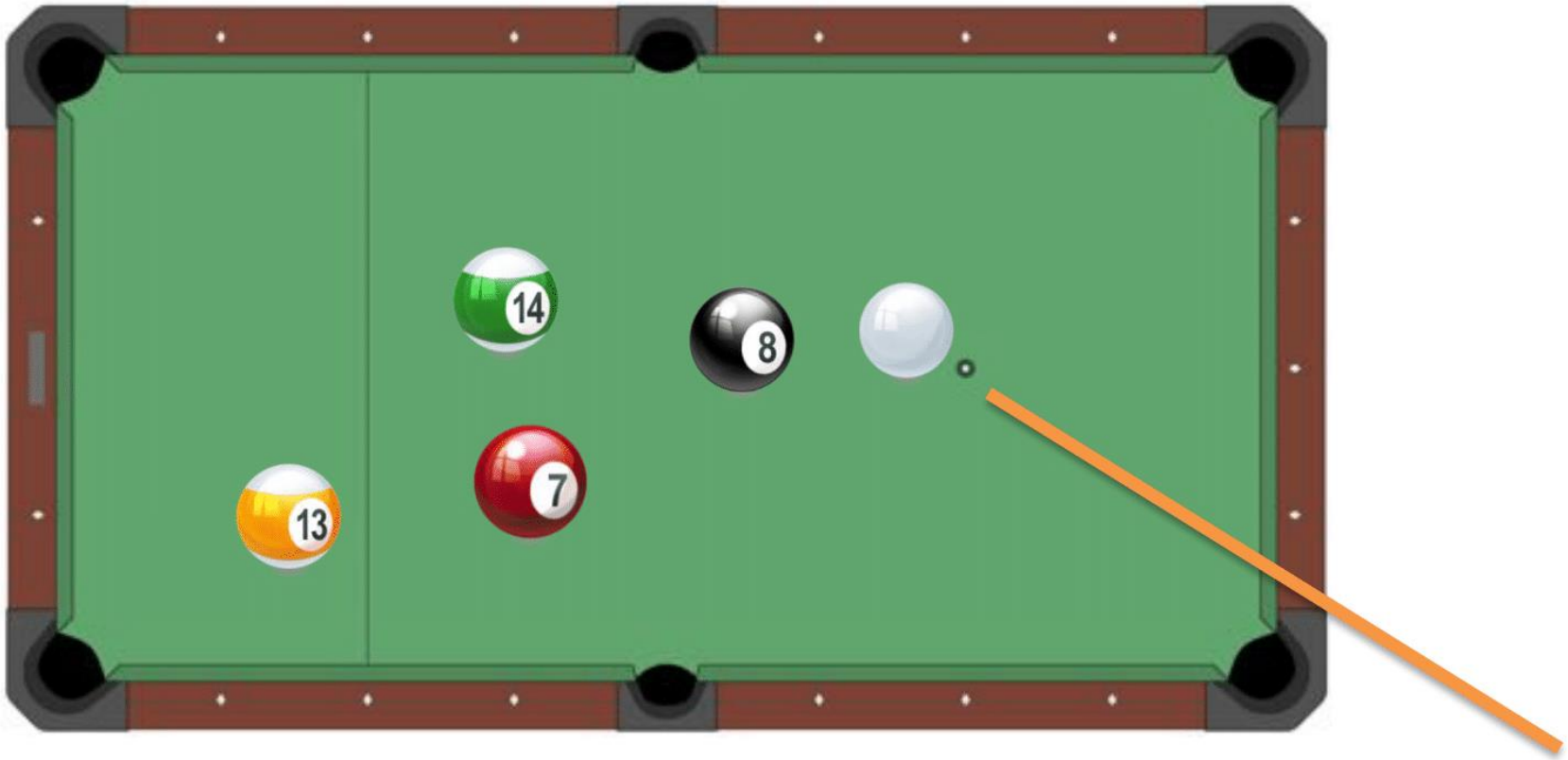
$$\mathcal{L}(w, X) = \sum_i l_i(w, x_i)$$

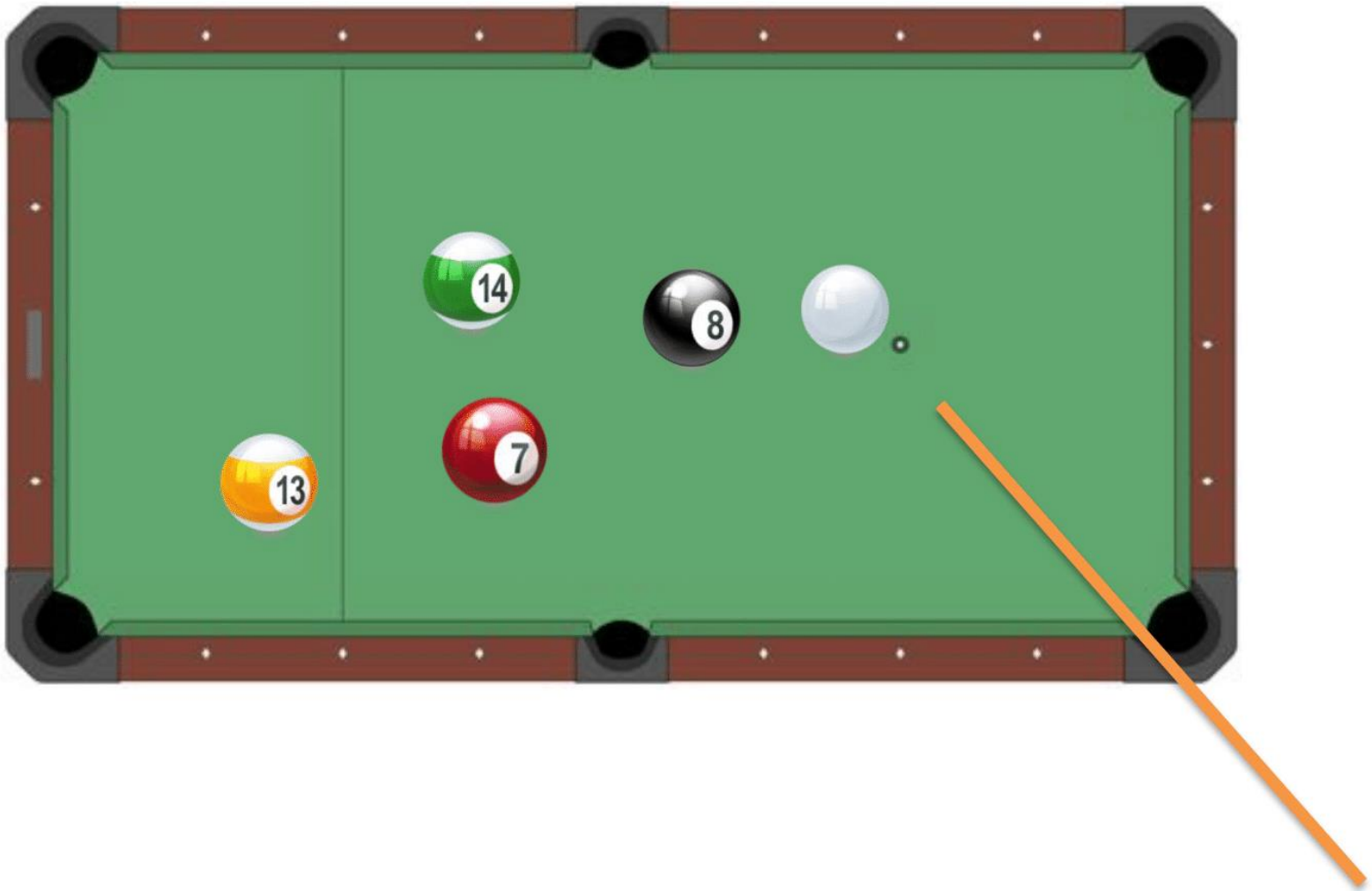
$$w_{t+1} = w_t - \eta \nabla_w \ell_i$$ Stochastic GD

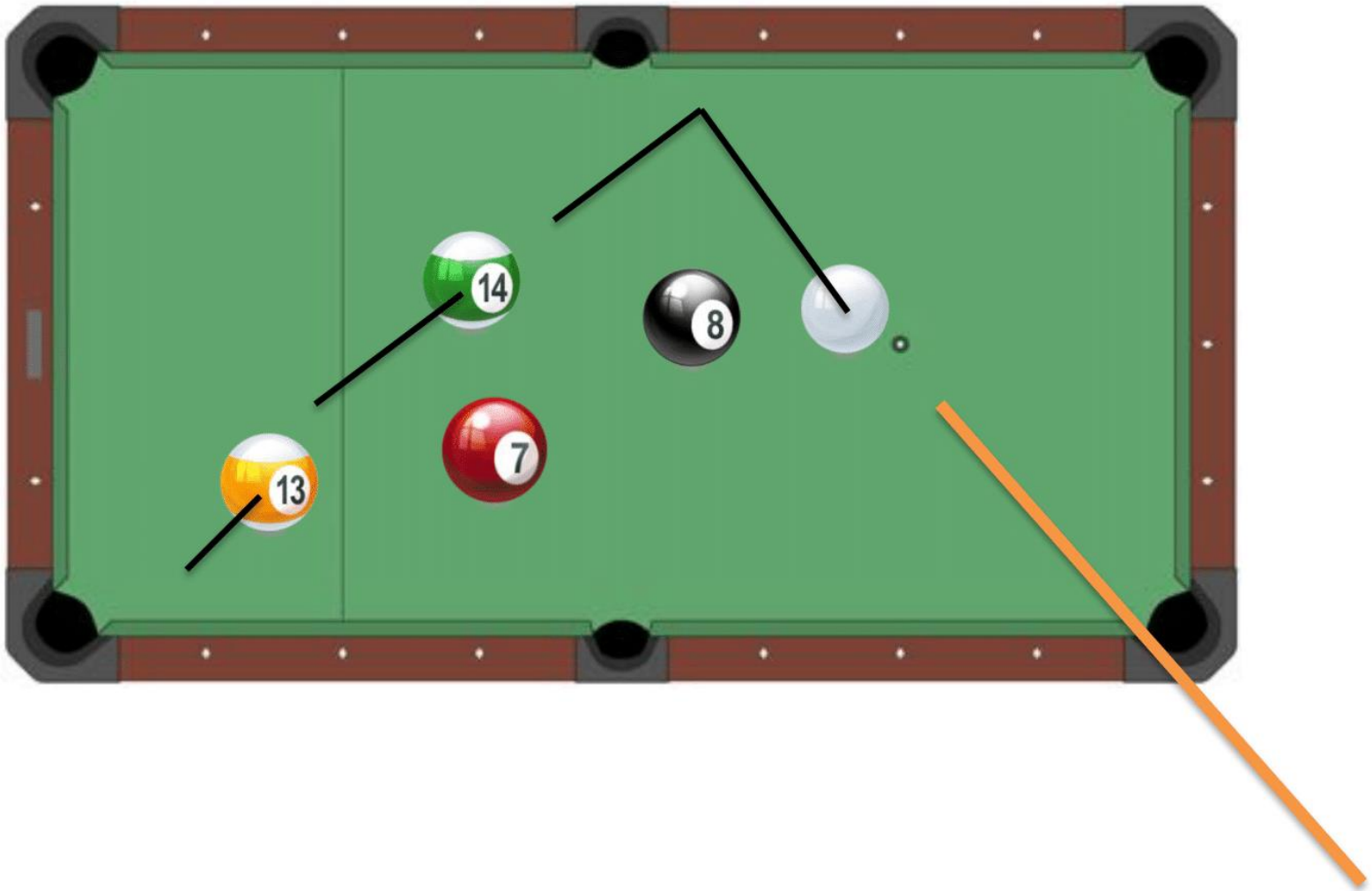
$$w_{t+1} = w_t - \eta \sum_{i \in I} \nabla_w \ell_i$$ Mini-batch GD

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

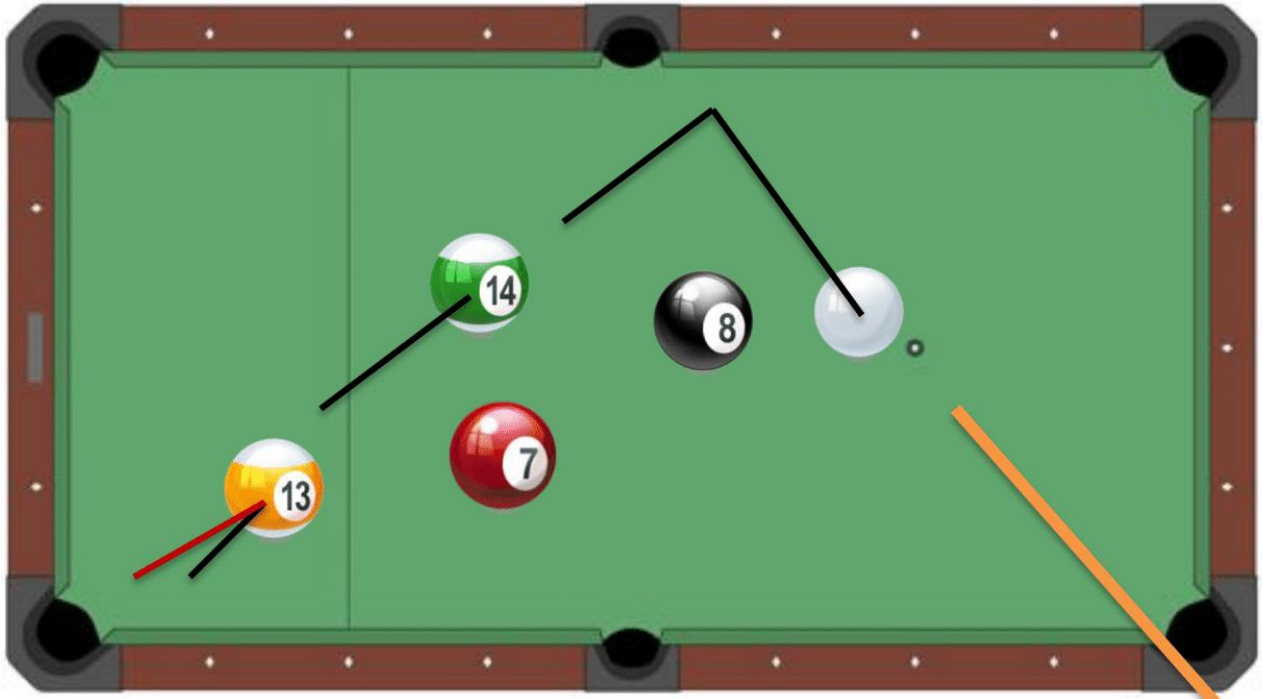
Intuition



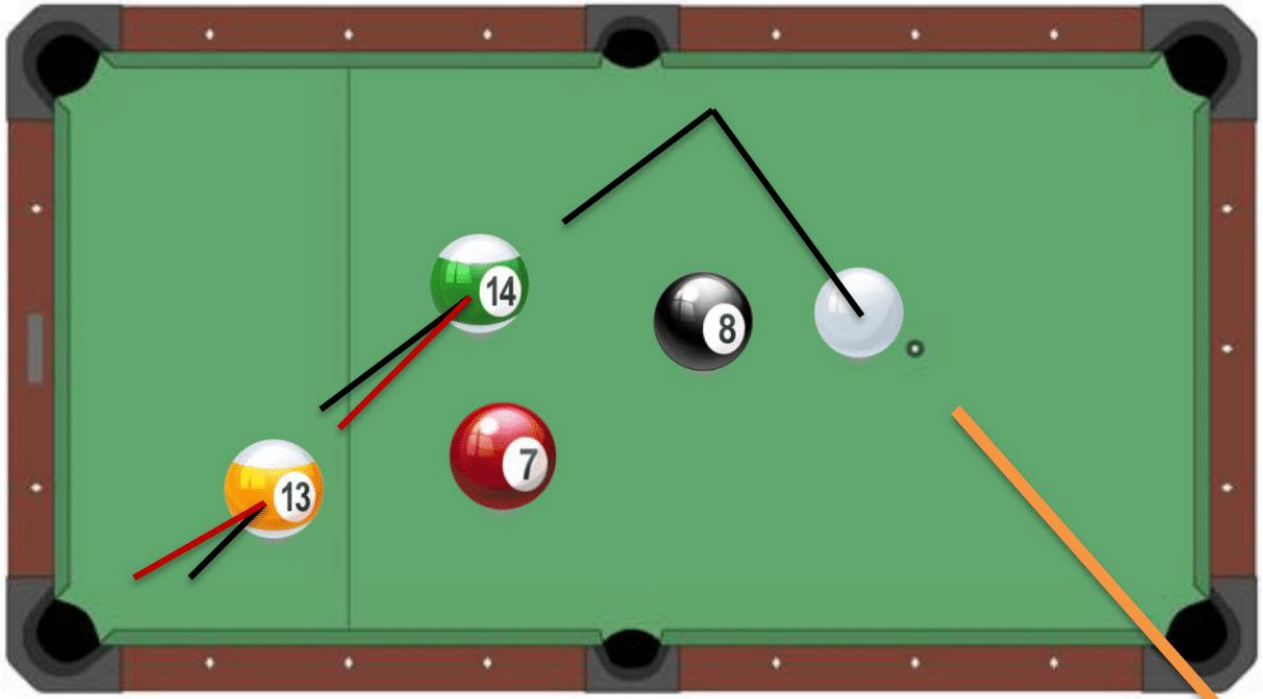

14
8
7
13



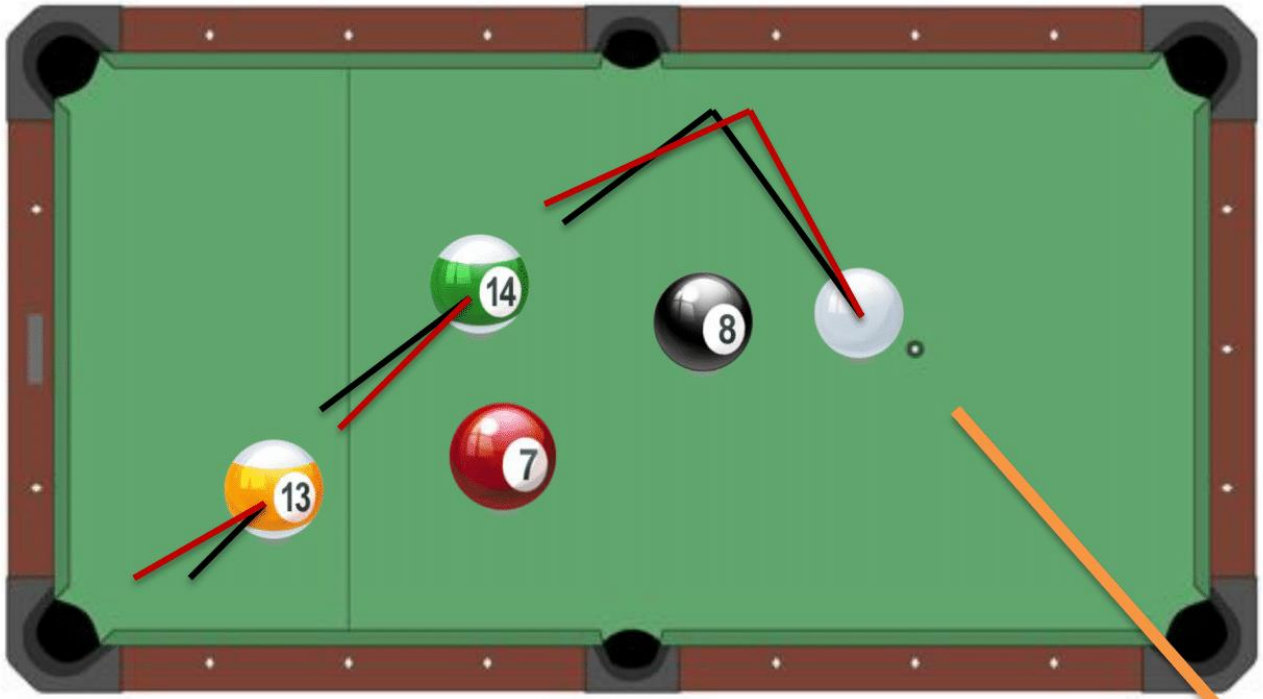
Forward pass



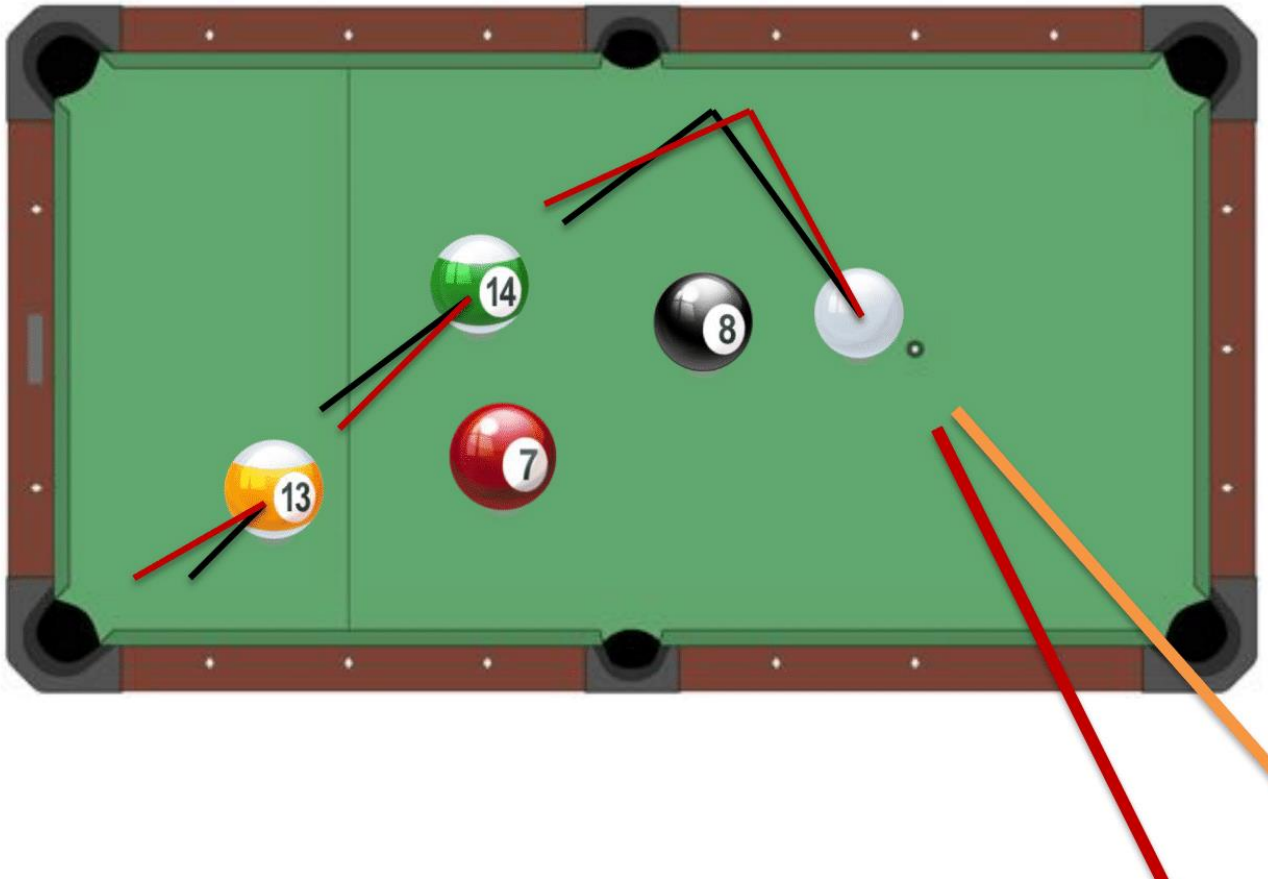
Backward pass



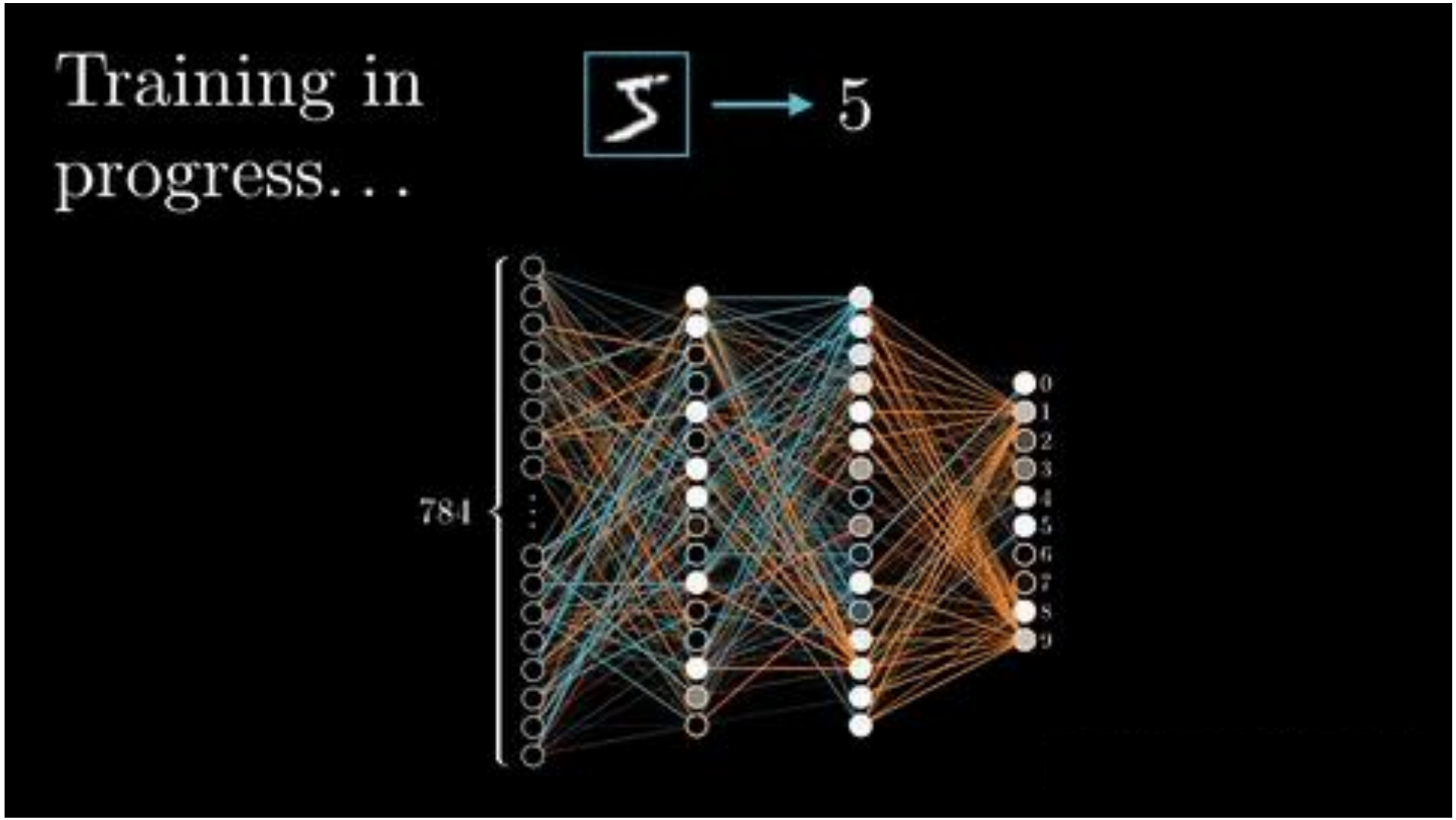
Backward pass

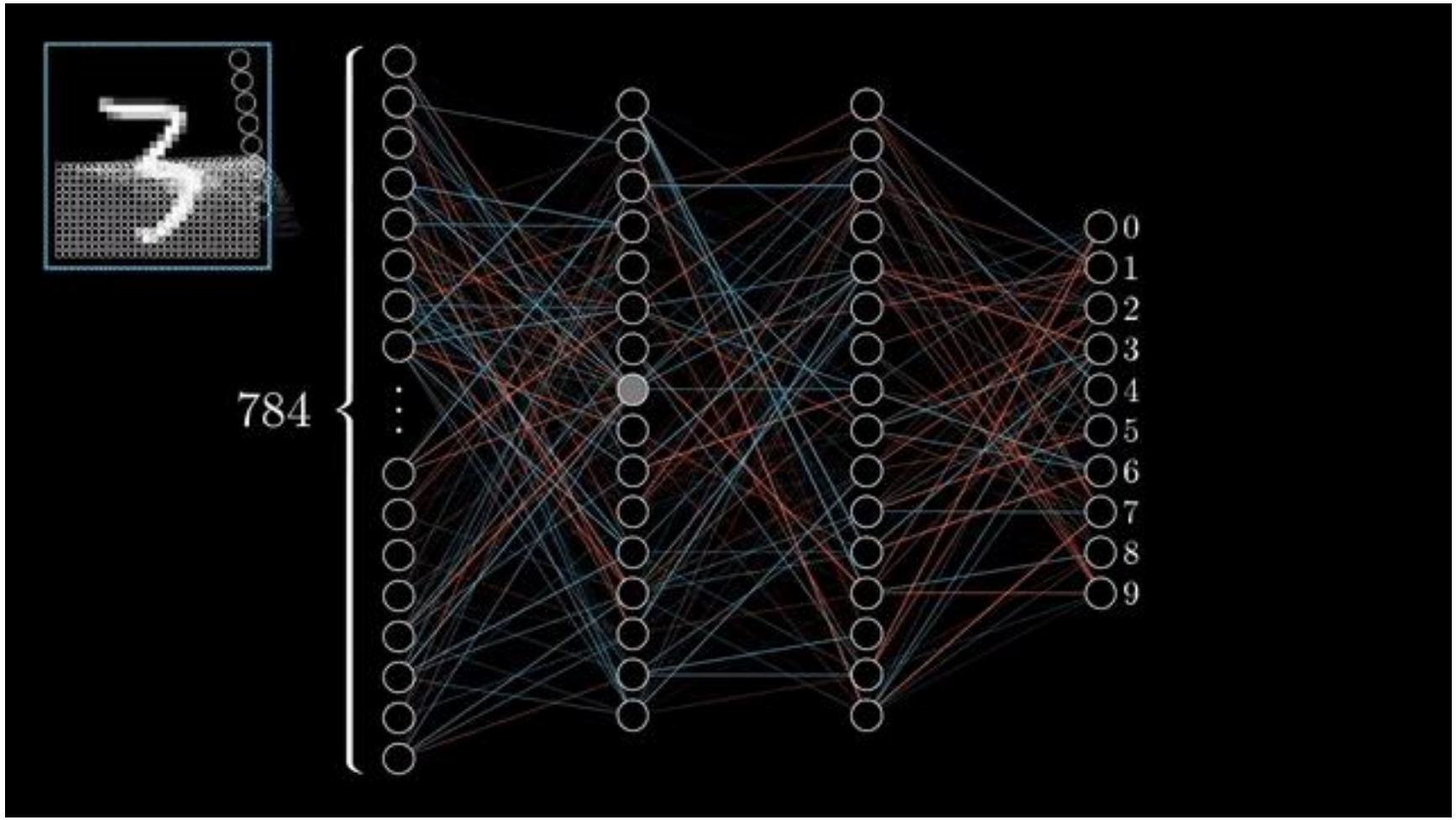


Backward pass



Backward pass


Training in progress...
5
784
0
1
2
3
4
5
6
7
8
9


784
0
1
2
3
4
5
6
7
8
9

How do we change the weights of the network to improve classification?

Chain rule

$$\frac{d[f \circ g]}{dx} = \frac{df(g)}{dg}\frac{dg}{dx}$$

$$\frac{d}{dx}cos(sin(x)) = \frac{d}{da}cos(a)\frac{da}{dx} = cos'(sin(x))sin'(x), \quad a = sin(x)$$

Training a 1 hidden layer neural network for classification task

Consider a one-layer NN:

$$\hat{y} = \text{softmax}(U\text{ReLU}(Wx + b_1) + b_2)$$

- Input: x
- Linear transformation: $z = Wx + b_1$
- Activation function: $h = \text{ReLU}(z)$
- Second linear transformation: $\theta = Uh + b_2$
- Output prediction: $\hat{y} = \text{softmax}(\theta)$
- Loss function: $J = CE(y, \hat{y})$

Dimensions: $x \in \mathbb{R}^{D_x \times 1}$, $b_1 \in \mathbb{R}^{D_h \times 1}$, $W \in \mathbb{R}^{D_h \times D_x}$, $b_2 \in \mathbb{R}^{N_c \times 1}$, $U \in \mathbb{R}^{N_c \times D_h}$

We want to calculate

- $\frac{\partial J}{\partial U}$
- $\frac{\partial J}{\partial b_2}$
- $\frac{\partial J}{\partial W}$
- $\frac{\partial J}{\partial b_1}$
- $\frac{\partial J}{\partial x}$

To start with, recall that $\mathrm{ReLU}(x) = \max(x, 0)$. This means:

$$\mathrm{ReLU}'(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases} = \mathrm{sgn}(\mathrm{ReLU}(x))$$

where sgn is the signum function.

Now, let's write out the chain rule for $\frac{\partial J}{\partial U}$ and $\frac{\partial J}{\partial b_2}$:

$$\frac{\partial J}{\partial U} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \theta} \frac{\partial \theta}{\partial U}$$

$$\frac{\partial J}{\partial b_2} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \theta} \frac{\partial \theta}{\partial b_2}$$

Notice that $\frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \theta} = \frac{\partial J}{\partial \theta}$ is present in both gradients.

To make computations more efficient, we define intermediate variables to represent the derivatives:

$$\delta_1 = \frac{\partial J}{\partial \theta} = (\hat{y} - y)^T$$

$$\delta_2 = \frac{\partial J}{\partial z} = \frac{\partial J}{\partial \theta} \frac{\partial \theta}{\partial h} \frac{\partial h}{\partial z}$$

Using the chain rule and substituting δ_1, we find:

$$\delta_2 = \delta_1 U \circ \mathsf{ReLU}'(z) = \delta_1 U \circ \mathsf{sgn}(h)$$

A good way of checking our work is by looking at the dimensions of the Jacobians:

$$\frac{\partial J}{\partial z} = \delta_1 U \circ \mathsf{sgn}(h)$$

Here, δ_1, U, and $\mathsf{sgn}(h)$ should align in dimensions to ensure the correct computation of gradients.

Gradient with respect to U:

$$\frac{\partial J}{\partial U} = \frac{\partial J}{\partial \theta}\frac{\partial \theta}{\partial U} = \delta_1 \frac{\partial \theta}{\partial U} = \delta_1^T h^T$$

Gradient with respect to b_2:

$$\frac{\partial J}{\partial b_2} = \frac{\partial J}{\partial \theta}\frac{\partial \theta}{\partial b_2} = \delta_1 \frac{\partial \theta}{\partial b_2} = \delta_1^T$$

Gradient with respect to W:

$$\frac{\partial J}{\partial W} = \frac{\partial J}{\partial \theta}\frac{\partial z}{\partial W} = \delta_2 \frac{\partial z}{\partial W} = \delta_2^T x^T$$

Gradient with respect to b_1:

$$\frac{\partial J}{\partial b_1} = \frac{\partial J}{\partial \theta}\frac{\partial z}{\partial b_1} = \delta_2 \frac{\partial z}{\partial b_1} = \delta_2^T$$

More in general:

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

$$act_l = W_l^T \sigma(act_{l-1})$$

$$\frac{\partial \sigma(act_L)}{\partial W_i} = \frac{\partial \sigma(act_L)}{\partial act_L} \frac{\partial act_L}{\partial \sigma(act_{L-1})} \cdots \frac{\partial \sigma(act_i)}{\partial act_i} \frac{\partial act_i}{\partial W_i}$$

$$\frac{\partial \sigma(act_l)}{\partial act_l} = \sigma'(act_l)$$

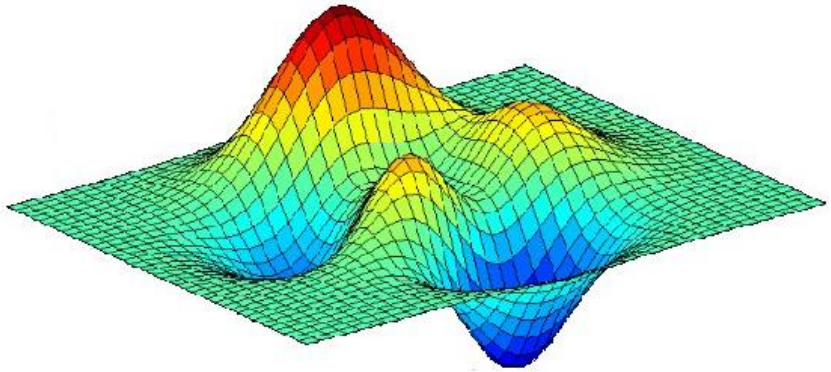
$$\frac{\partial act_L}{\partial \sigma(act_{l-1})} = W_l^T$$

Training neural networks: backward pass, W

$$
\begin{aligned}
\frac{\partial \sigma(act_L)}{\partial W_i} &= \sigma'(act_l) W_l^T \cdots \frac{\partial \sigma(act_i)}{\partial act_i} \frac{\partial act_i}{\partial W_i} \\
&= \sigma'(act_l) W_l^T \cdots \frac{\partial \sigma(W_i^T \sigma(act_{i-1}))}{\partial W_i^T \sigma(act_{i-1})} \frac{\partial act_i}{\partial W_i} \\
&= \sigma'(act_l) W_l^T \cdots \sigma'(W_i^T \sigma(act_{i-1})) \sigma(act_{i-1})
\end{aligned}
$$

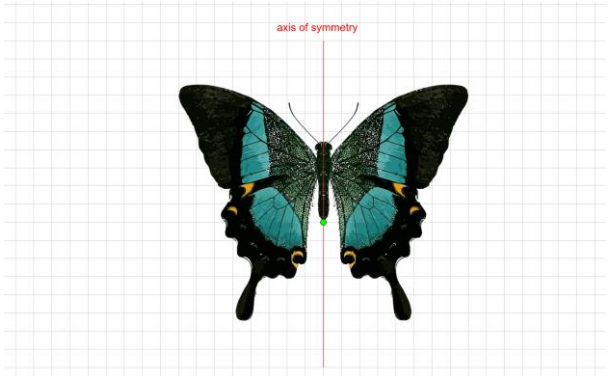
- We calculate the effect of the network on the Loss J with the forward pass
- We update the variables using the derivative at each stage of the calculation
- The update is done using the chain rule and the updates at the previous derivative

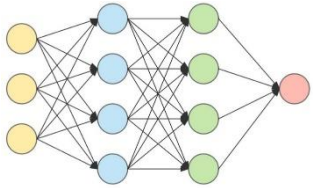
Why do we find good minima with backprop?

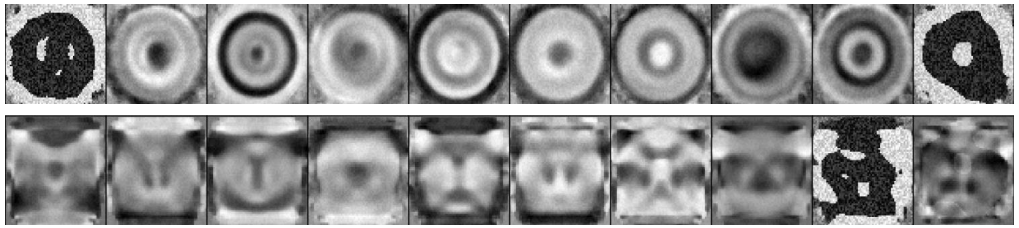


non-convex function

GD and data Symmetries





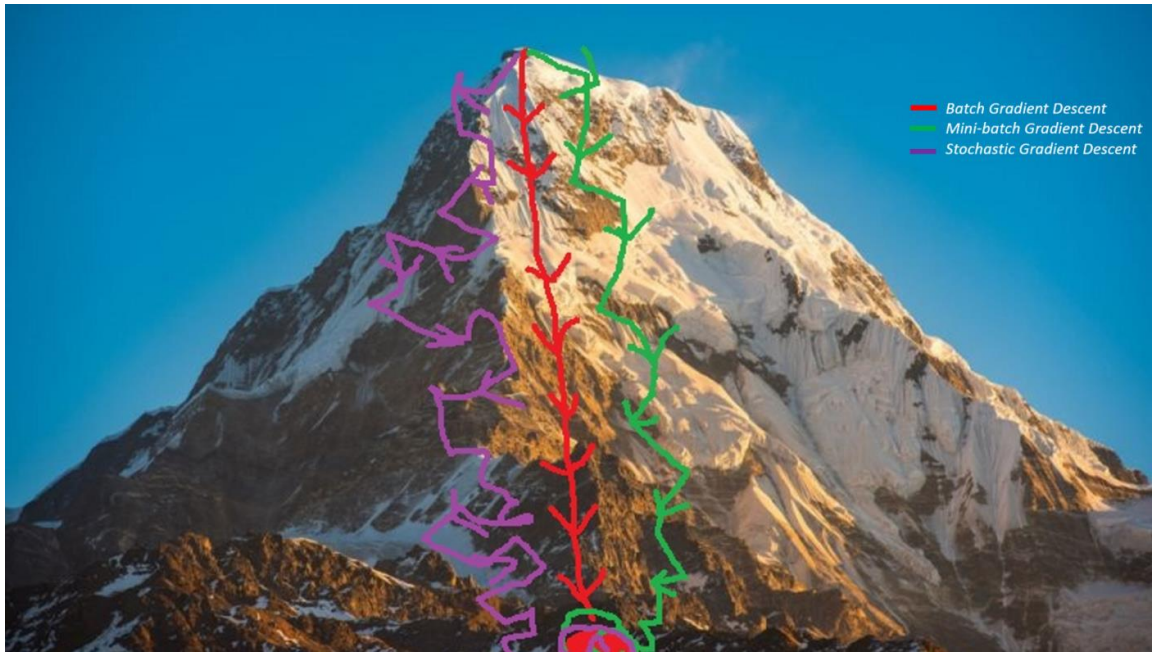


$$W_{t+1} - W_t \propto \nabla_W \mathcal{L}(W_t; X, y) =$$

$$= \frac{1}{R} \sum_{i=1}^{R} \ell'(v^T \sigma \langle W_t, x_i \rangle), y_i) v^T \sigma' \langle W_t, x_i \rangle x_i.$$

Class 3: Stochastic gradient descent, implicit bias and good solutions

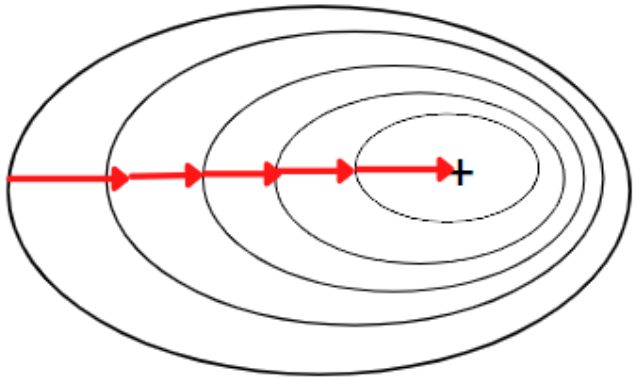
Mini-batch and Stochastic gradient decent



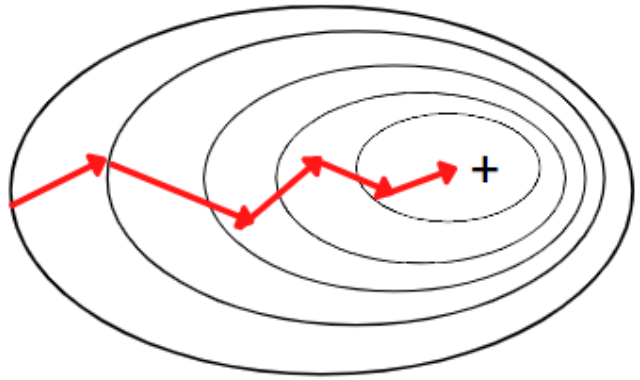
$$W_{t+1} = w_t - \gamma \nabla_W \frac{1}{B} \sum_{i=1}^{B} \left(y_i - w^T \sigma(W^T x_i) \right)^2$$

$$W_{t+1} = w_t - \gamma \nabla_W \left(y_i - w^T \sigma(W^T x_i) \right)^2$$

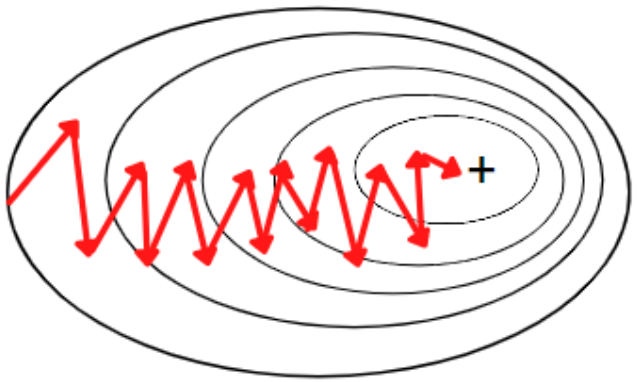
Batch Gradient Descent



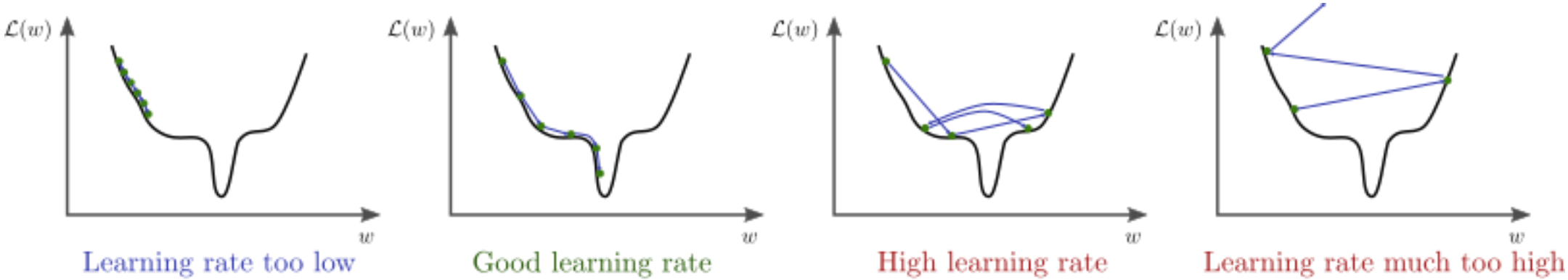
Mini-Batch Gradient Descent

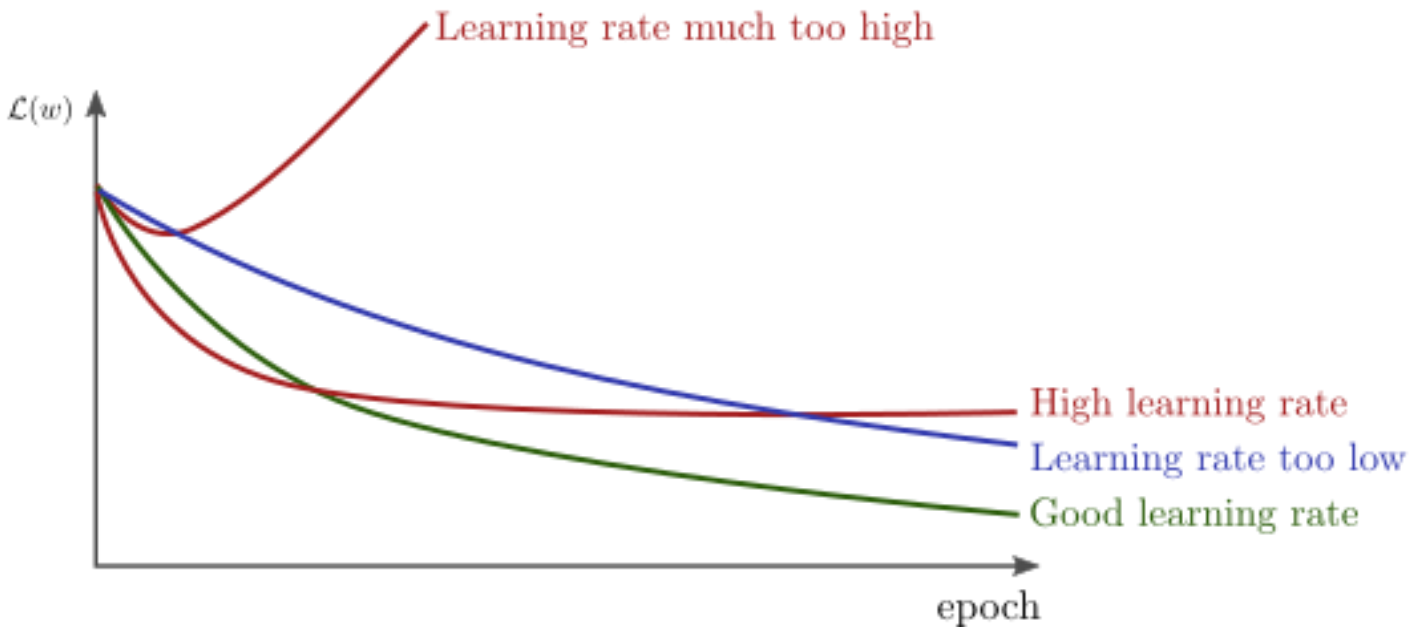


Stochastic Gradient Descent

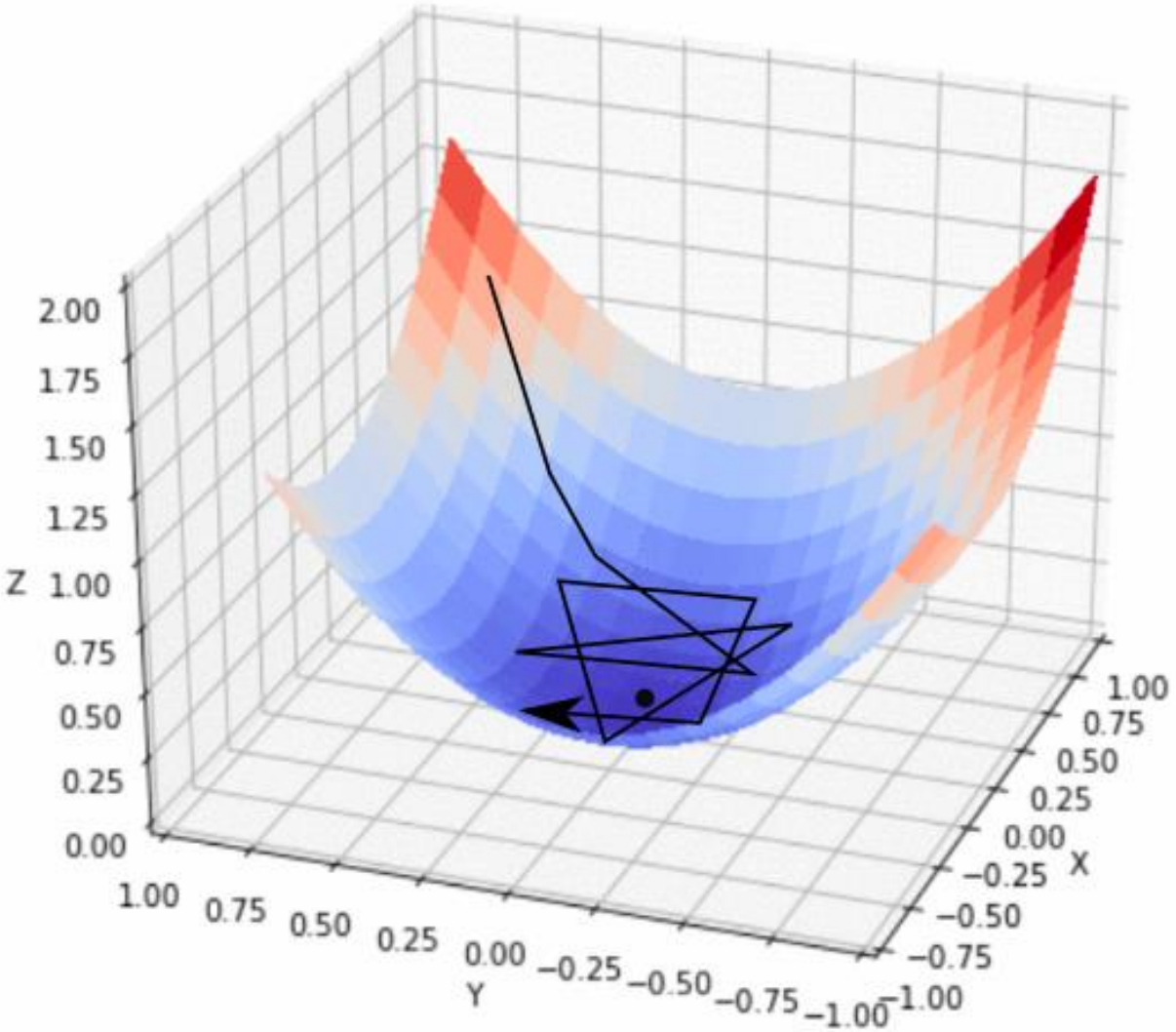


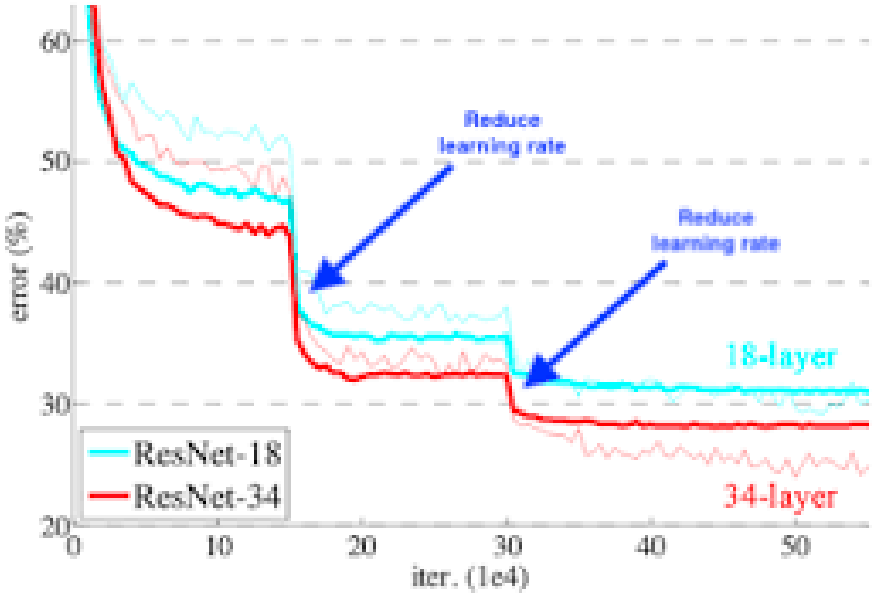
Learning rate





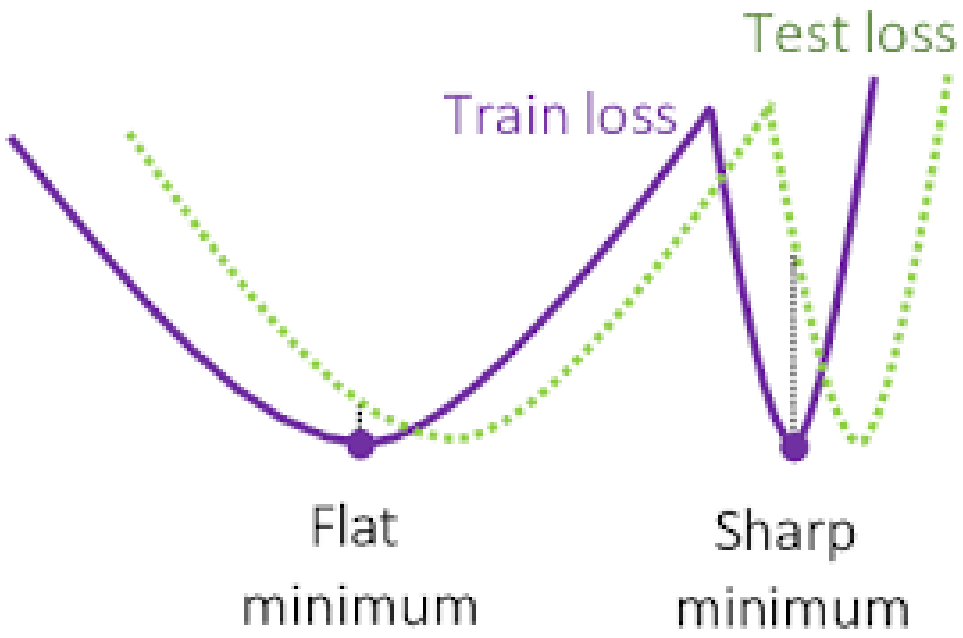
Learning rate for SGD: scheduling



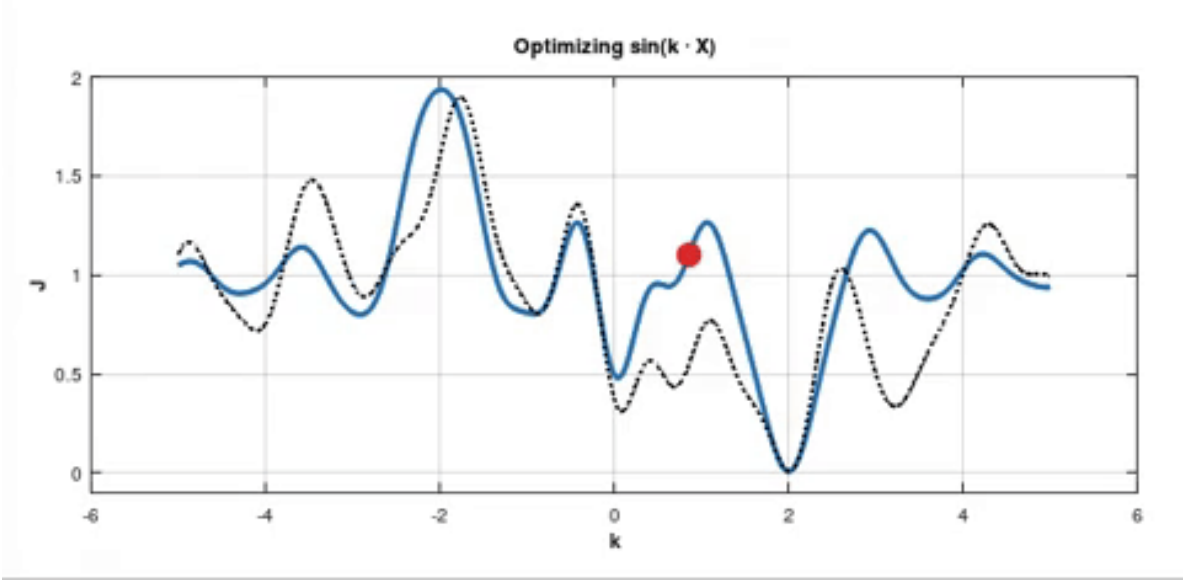


Advantages of SGD

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



Intuition



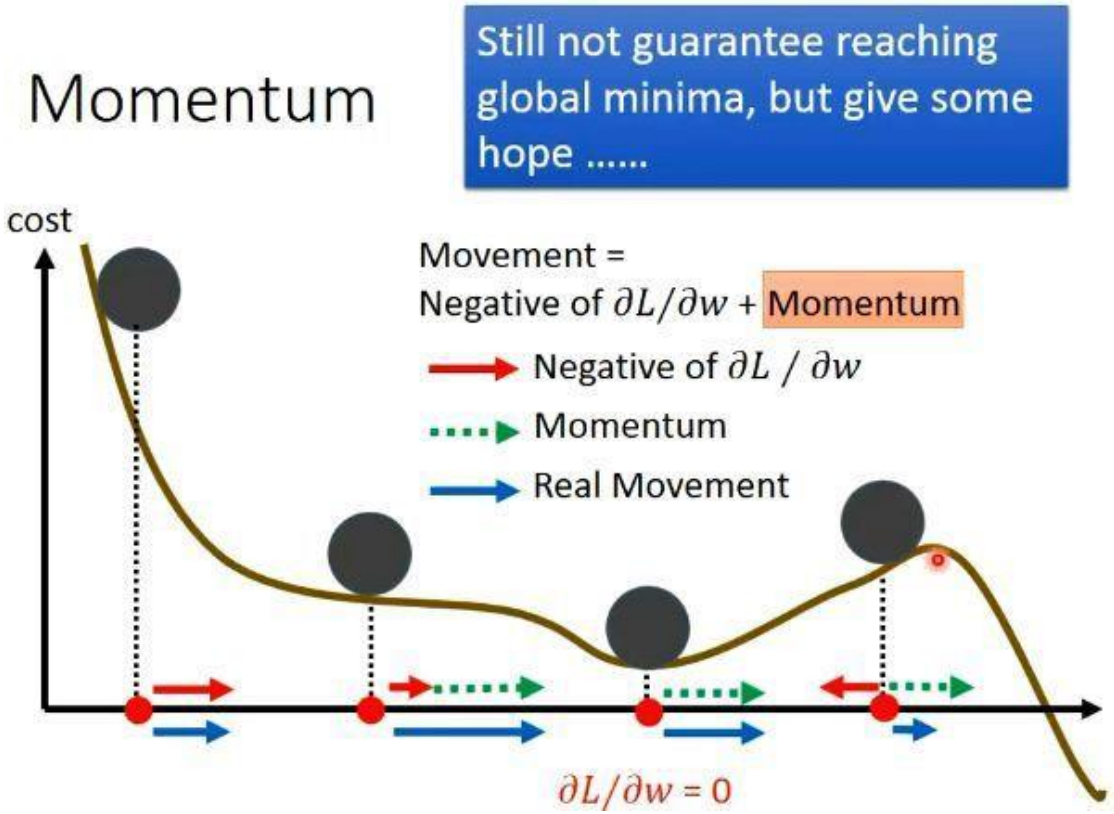
At each step the SGD is looking to a “different, sampled” function

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

Advantages of SGD

- We need way less computations (proportional to 1 or Batch size)
- Converges to good minima (invariant to data perturbations)

Other strategies to reach good minima: add momentum

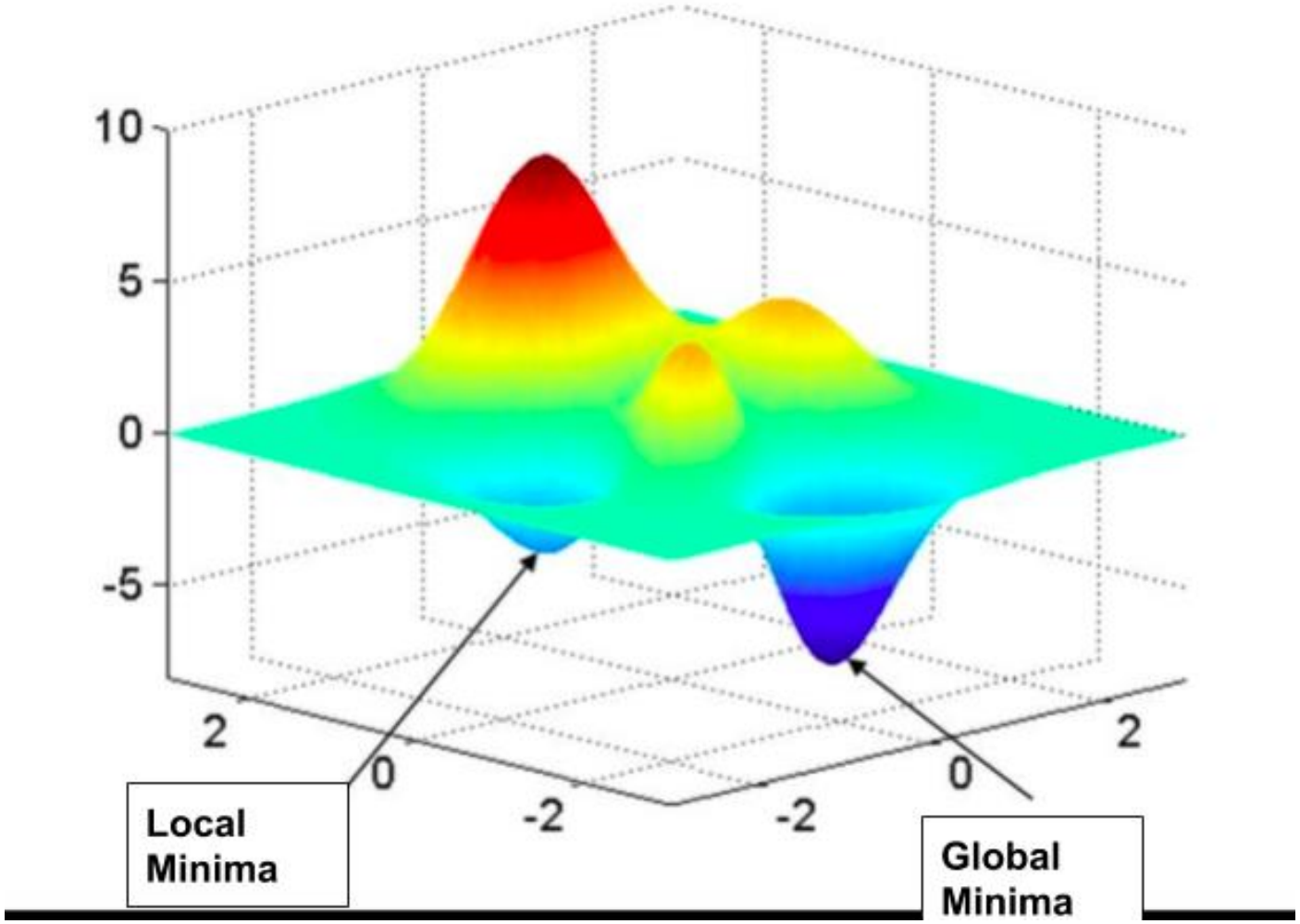


$$v_{t+1} = \rho v_t + \nabla_w \mathcal{L}(w_t)$$

$$w_{t+1} = w_t - \gamma v_{t+1}$$

Going out from flat bad minima

Importance of initialization



Which other factors influence the convergence to good minima?

Implicit 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 with regression: minimum norm solution

one particular solution is

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

(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: minimum norm solution

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

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

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

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

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

Example with regression: minimum norm solution with GD

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

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

whose gradient is

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

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

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

where

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

and, thus,

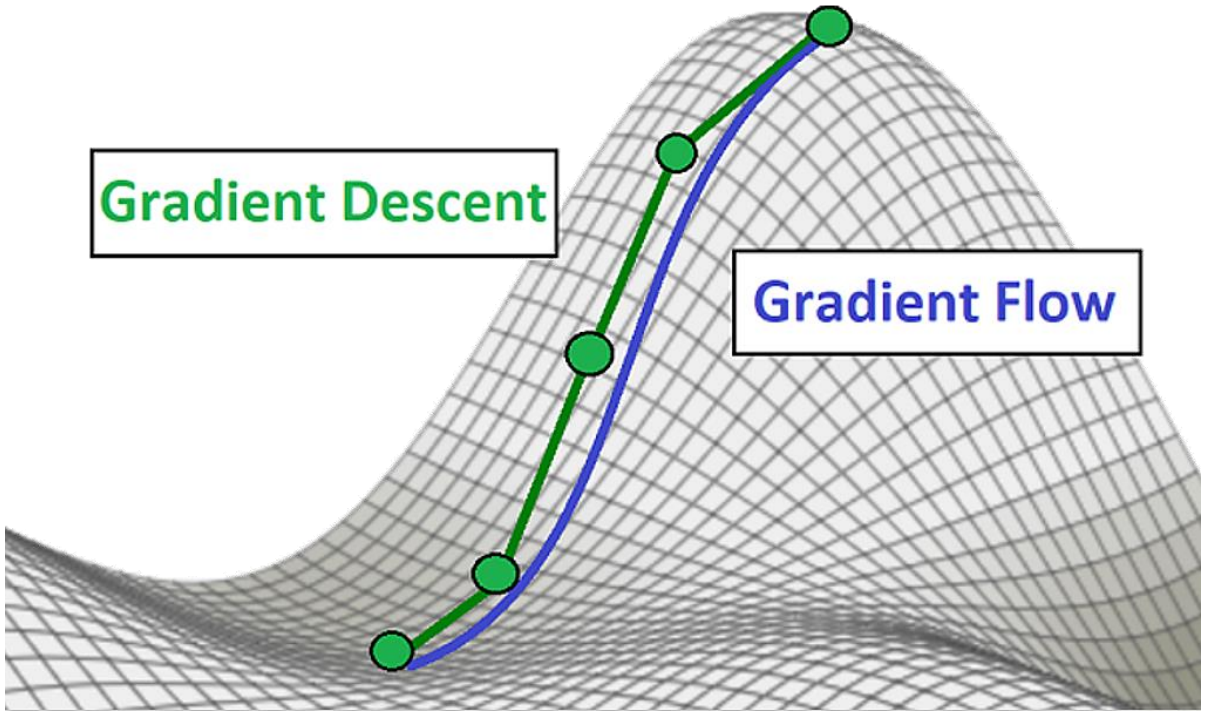
$$\mathrm{y}_\infty = \begin{bmatrix} \mathrm{O} & \mathrm{O} \\ \mathrm{O} & \mathrm{I} \end{bmatrix} \mathrm{y}_0 + \begin{bmatrix} \Sigma_1^{-1} \\ \mathrm{O} \end{bmatrix} \mathrm{U}^\top \mathrm{b}$$

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

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

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.

Implicit regularization for neural networks with discrete GD

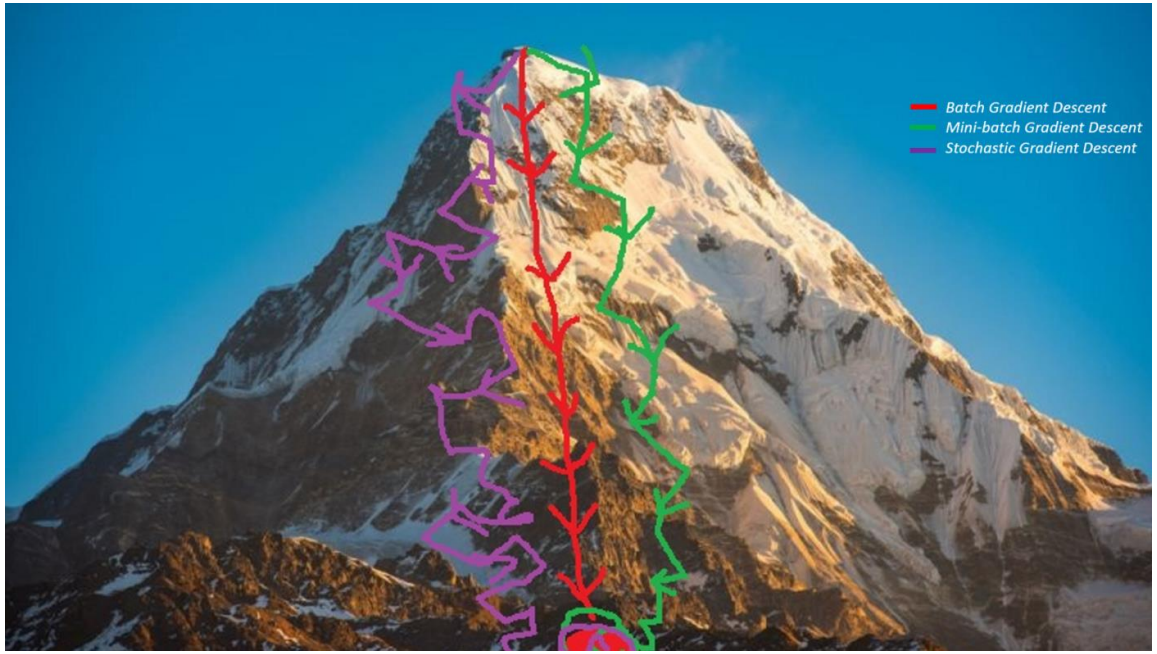


Effective Loss due to GD discretization

$$\tilde{\mathcal{L}}_{GD}(\boldsymbol{w}) = \mathcal{L}(\boldsymbol{w}) + \frac{\epsilon}{4}||\nabla \mathcal{L}(\boldsymbol{w})||^2$$

$$\dot{\boldsymbol{w}} = -\nabla_{\boldsymbol{w}} \tilde{\mathcal{L}}_{GD}(\boldsymbol{w})$$

Implicit regularization for neural networks due to SGD



Effective Loss for SGD + discretization

$$\tilde{\mathcal{L}}_{SGD}(\boldsymbol{w}) = \tilde{\mathcal{L}}_{GD}(\boldsymbol{w}) + \frac{\epsilon}{4}\sum_{k=1}^{m} ||\nabla \mathcal{L}_k(\boldsymbol{w}) - \mathcal{L}(\boldsymbol{w})||^2$$

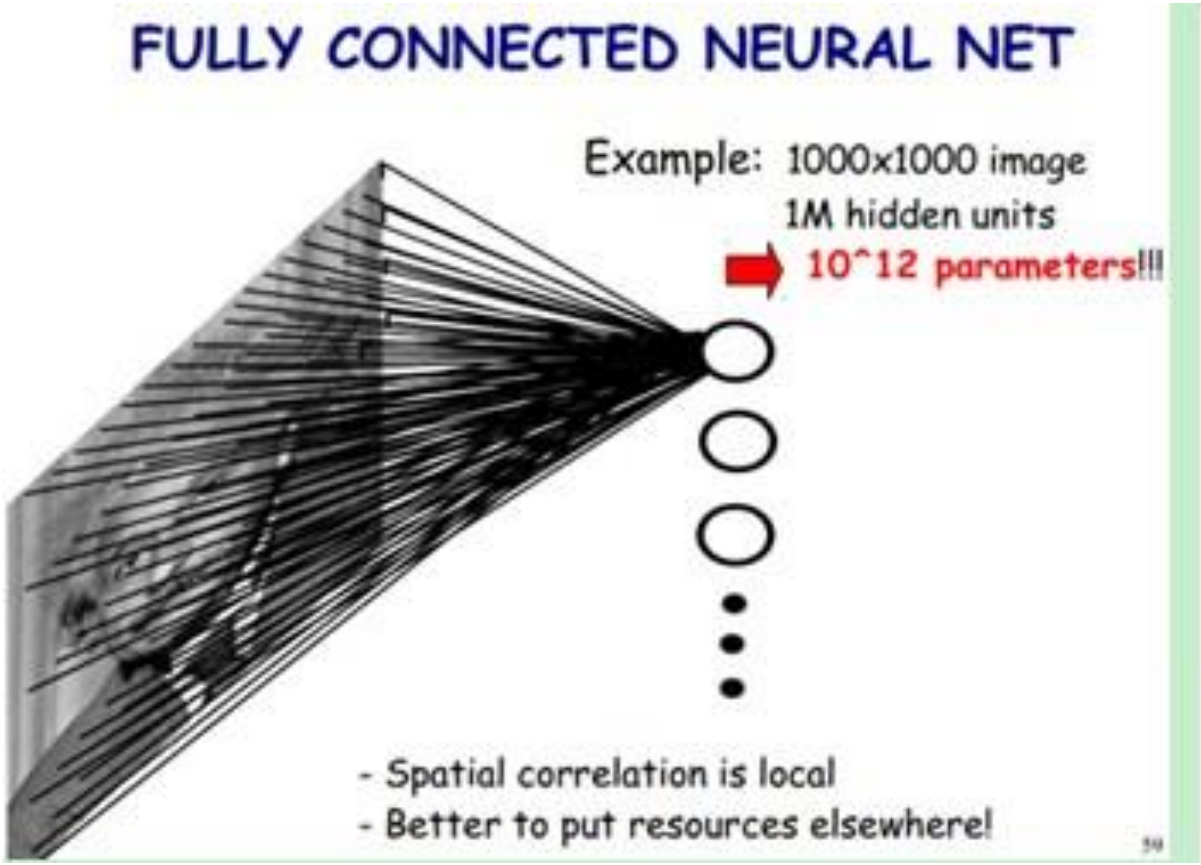
The second term estimates the variance of the minibatch gradients, which is a measure of stability, and hence of generalization ability.

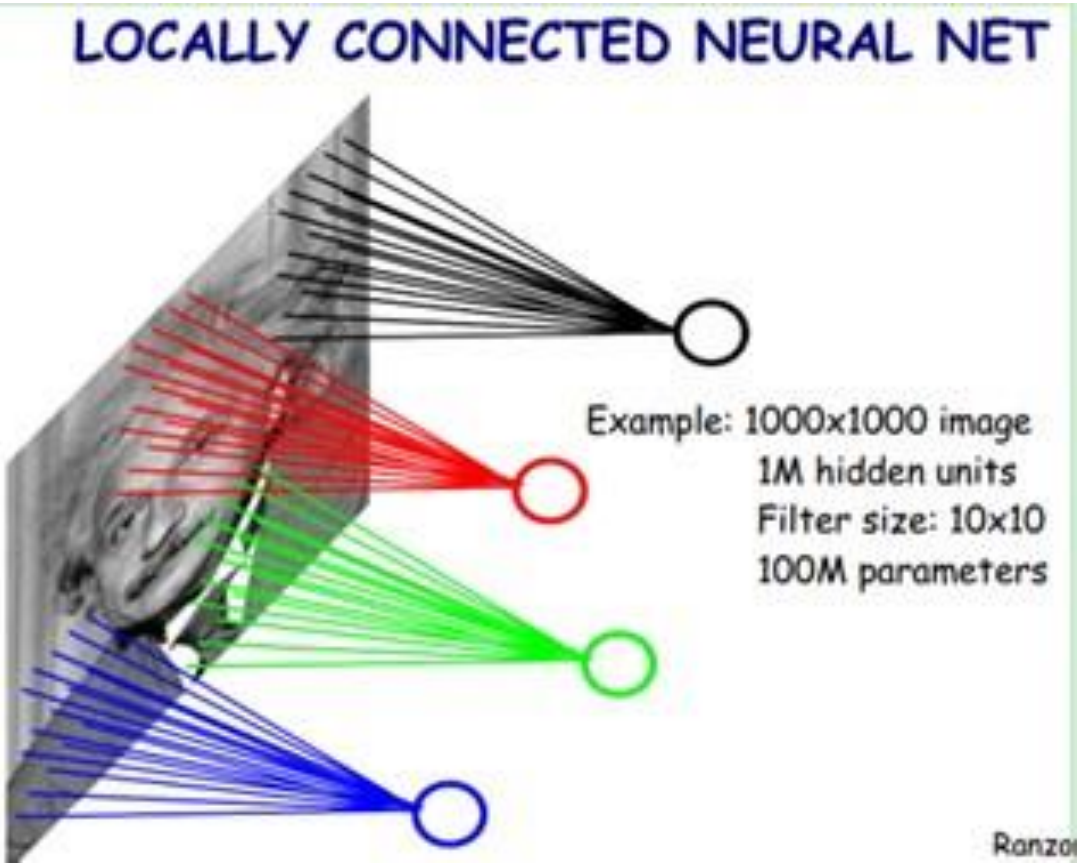
Class 4

- Convolution neural networks: invariance, equivariance, locality.

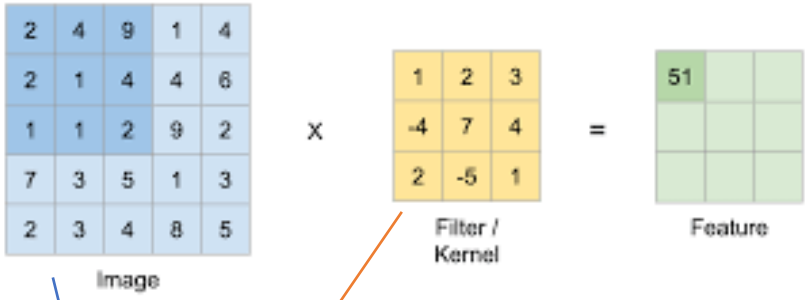
- Implementing other types of invariances: data augmentations

Convolutional vs fully connected networks

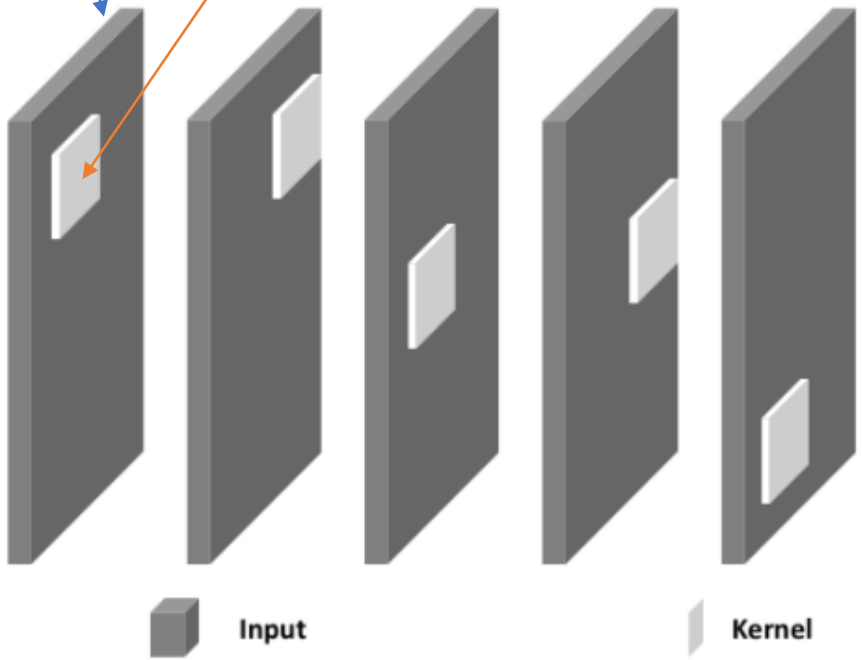




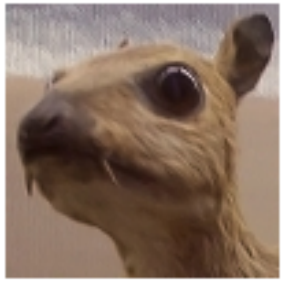
Convolution operation



$$w * x$$



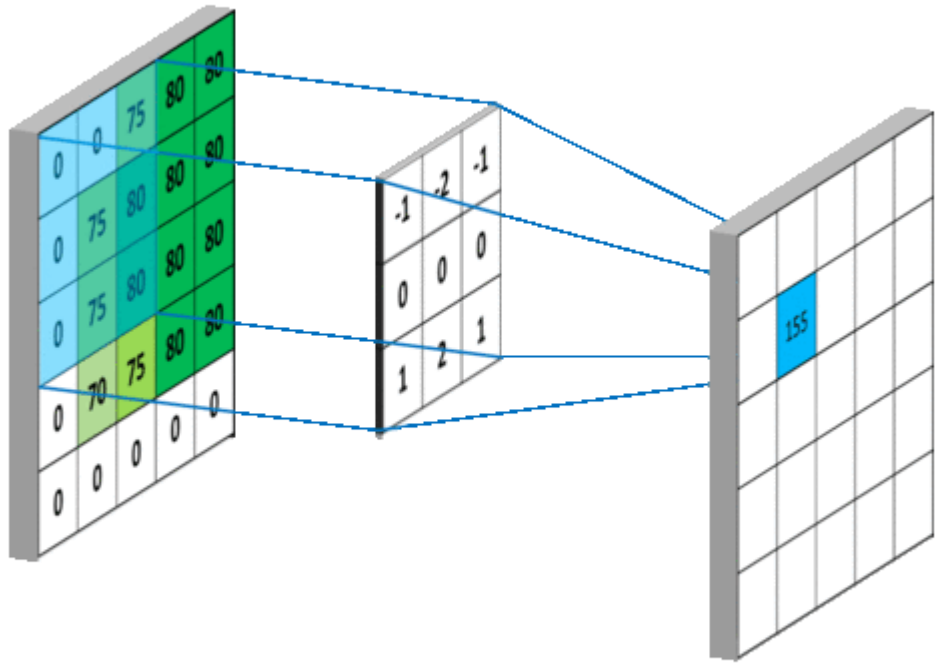
Input image



Convolution Kernel

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

Feature map

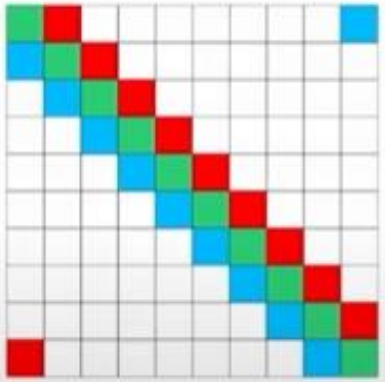


$$(f * g)(t) \quad \stackrel{\text{def}}{=} \quad \int_{-\infty}^{\infty} f(\tau) g(t - \tau) \, d\tau \quad = \int_{\infty}^{-\infty} f(\tau) T_t g(-\tau) d\tau = \langle f, T_t g \rangle$$

Discrete version of the convolution

$$C(w) = \begin{bmatrix} c_1 & c_3 & c_2 \\ c_2 & c_1 & c_3 \\ c_3 & c_2 & c_1 \end{bmatrix}$$

w



Why convolutions? Invariance and equivariance

$$(w * x)_i = (W^T x)_i = \langle w_i, x \rangle = \langle g_i w, x \rangle$$

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

Why convolutions? Invariance and equivariance

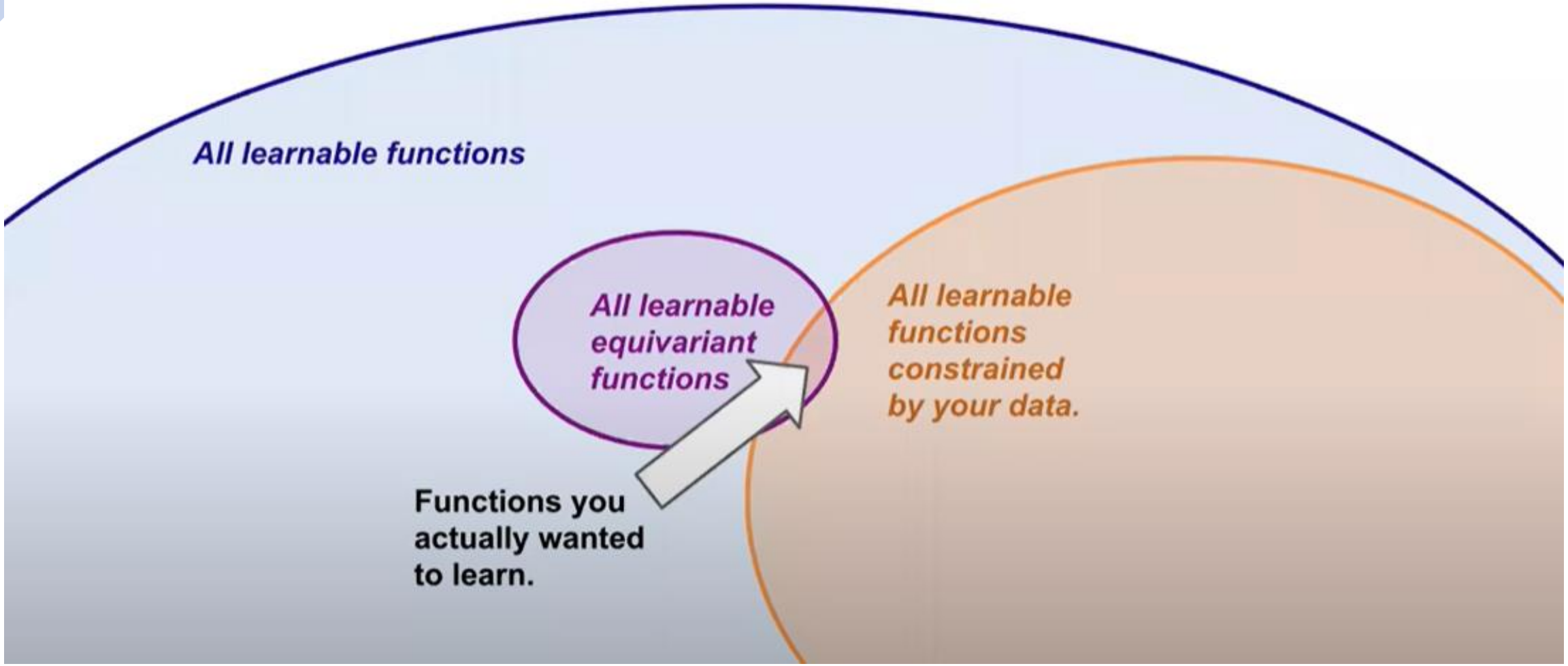
Group: set $\mathcal{G}$ with composition/multiplication operation satisfying:

1. closure: $gg^{'} \in \mathcal{G}, \ \forall \ g, g^{'} \in \mathcal{G}$
2. associativity: $(gg^{'})g^{''} = g(g^{'}g^{''}) \in \mathcal{G}, \ \forall \ g, g^{'}, g^{''} \in \mathcal{G}$
3. identity: there exists $Id \in \mathcal{G}$ such that $Idg = gId = g \ \forall \ g \in \mathcal{G}$
4. invertibility: $\forall \ g \in \mathcal{G}$ there exists $g^{-1} \in G \ : \ gg^{-1} = Id.$

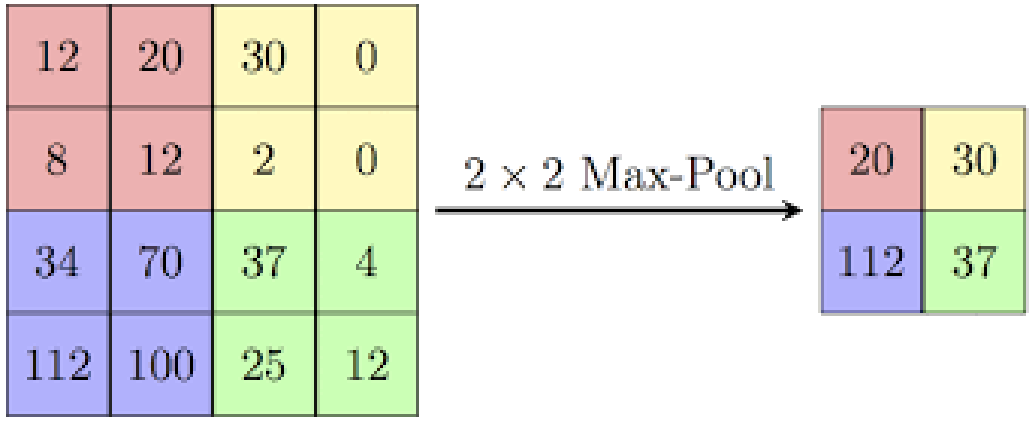
Rotation

Why convolutions? Invariance and equivariance

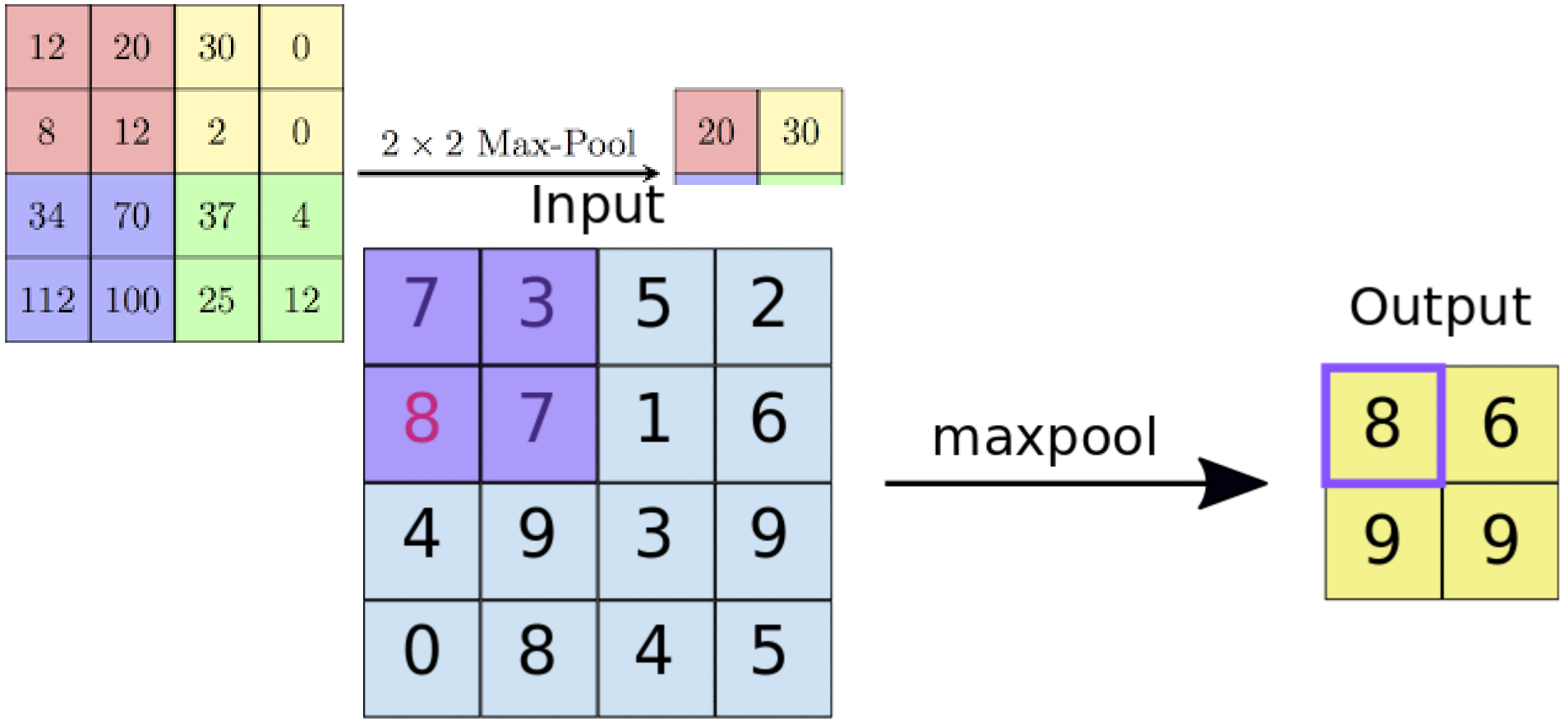
$$f(gx) = \begin{bmatrix} \sigma\left\langle g_1 w, gx \right\rangle \\ \vdots \\ \sigma\left\langle g_N w, gx \right\rangle \end{bmatrix} = \begin{bmatrix} \sigma\left\langle w, g_1^T gx \right\rangle \\ \vdots \\ \sigma\left\langle w, g_N^T gx \right\rangle \end{bmatrix} = P_g \begin{bmatrix} \sigma\left\langle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


All learnable functions
All learnable equivariant functions
All learnable functions constrained by your data.
Functions you actually wanted to learn.

Pooling operation



Pooling operation



Can you think of other pooling operations?

From equivariance to invariance

$$\sum_i (f(gx))_i = \sum_i (P_g f(x))_i$$

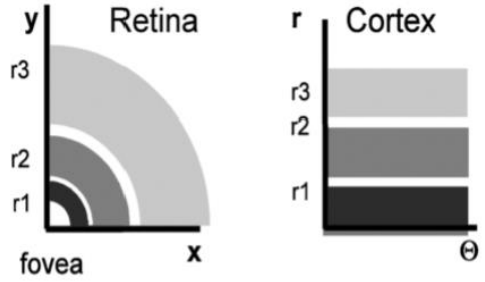
Why is this expression invariant to g transformations?

Why do we care? Mimicking the brain

Equivariance and invariance are important properties not only in deep nets but also in visual cortex signal processing

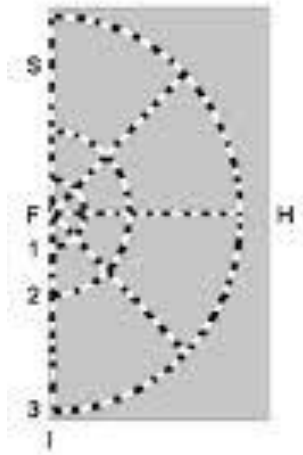
Invariant object recognition

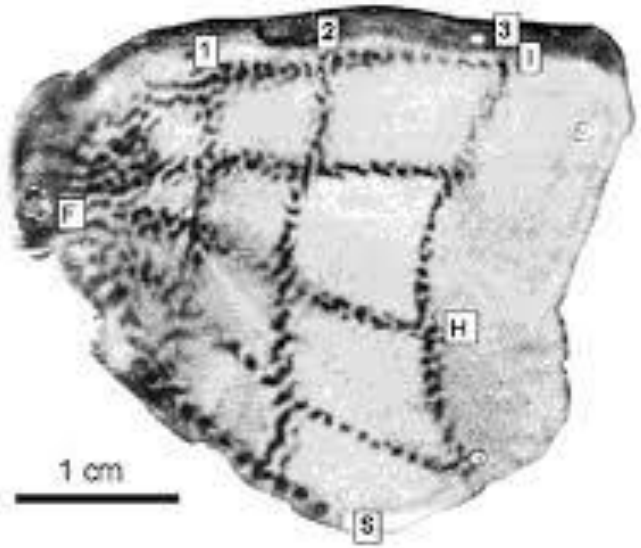
Retinotopic mapping



Invariance: $f(gx) = f(x)$.

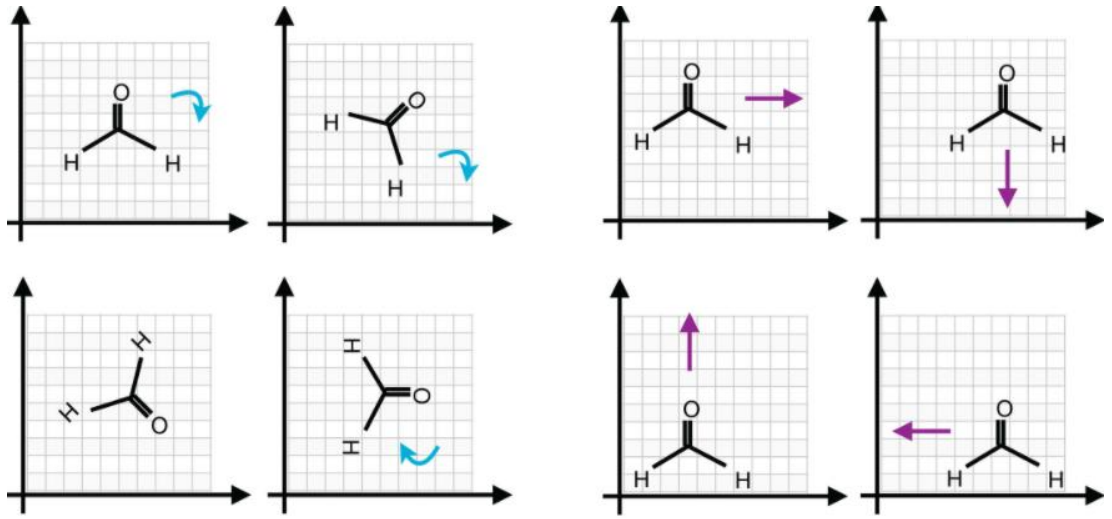
Equivariance: $f(gx) = gf(x)$



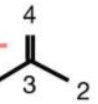


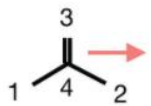
While in CNNs invariance/equivariance are natural (at least for translations) how these properties could possibly be implemented in visual cortex is not clear.

Why do we care? Molecule symmetries



$$\begin{bmatrix} H & 0 & 1 & 0 \\ 0 & H & 1 & 0 \\ 1 & 1 & C & 2 \\ 0 & 0 & 2 & O \end{bmatrix}$$





$$\begin{bmatrix} H & 0 & 0 & 1 \\ 0 & H & 0 & 1 \\ 0 & 0 & O & 2 \\ 1 & 1 & 2 & C \end{bmatrix}$$

Permutations

Convolutional networks and convolution kernels

Convolution layer representation (with average pooling),

$$\Phi(x) = \left(\sum_g s(\langle x, gt_1 \rangle), \ldots, \sum_g s(\langle x, gt_T \rangle) \right), \quad x \in \mathcal{X}.$$

The **associated kernel** can be written as

$$K(x, x) = \langle \Phi(x), \Phi(x') \rangle = \sum_t \left(\sum_g s(\langle x, gt_1 \rangle) \cdot \sum_g s(\langle x', gt_1 \rangle) \right)$$

Convolutional networks and convolution kernels (cont.)

Using linearity

$$K(x,x') = \sum_g \sum_{g'} \underbrace{\left(\sum_t s(\langle x, gt_1 \rangle) \cdot s(\langle x', g't_1 \rangle) \right)}_{K'(x,x')}.$$

Convolution kernels

$$K(x,x') = \sum_g \sum_{g'} K'(gx, g'x')$$

- Kernel of the above form are a special case of **convolution kernels**.
- If $g_1, \ldots, g_G$ form a **group** then the kernel is invariant $K(x,x') = K(x,gx')$, i.e. $\Phi(x) = \Phi(gx)$.

What if we do not have a group transformation?

Group: set $\mathcal{G}$ with composition/multiplication operation satisfying:

1. closure: $gg^{'} \in \mathcal{G}, \ \forall \ g, g^{'} \in \mathcal{G}$
2. associativity: $(gg^{'})g^{''} = g(g^{'}g^{''}) \in \mathcal{G}, \ \forall \ g, g^{'}, g^{''} \in \mathcal{G}$
3. identity: there exists $Id \in \mathcal{G}$ such that $Idg = gId = g \ \forall \ g \in \mathcal{G}$
4. invertibility: $\forall \ g \in \mathcal{G}$ there exists $g^{-1} \in G \ : \ gg^{-1} = Id.$

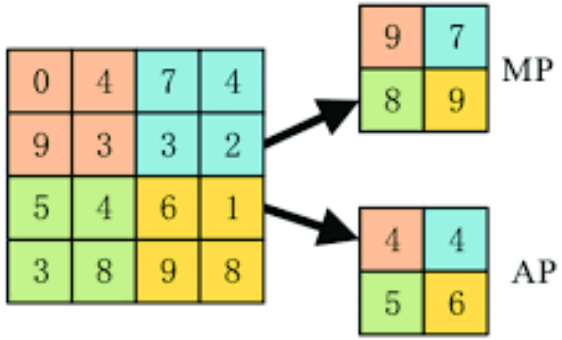
Rotation

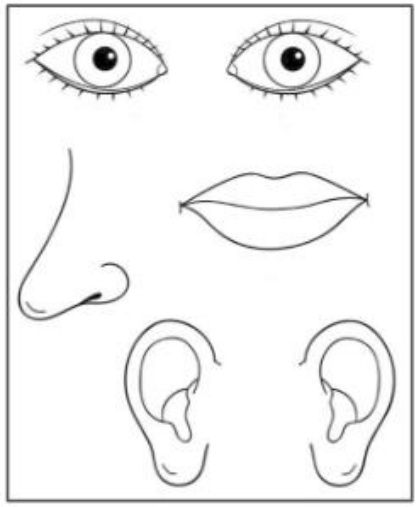
Why local filters?

- Robustness to (small) deformations

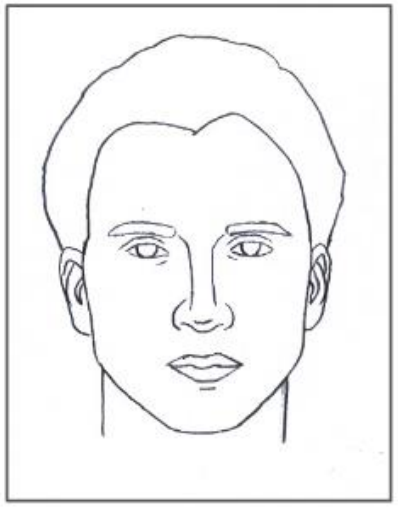
- Parts reusability in a hierarchical model

Robustness to deformations: pooling+ locality

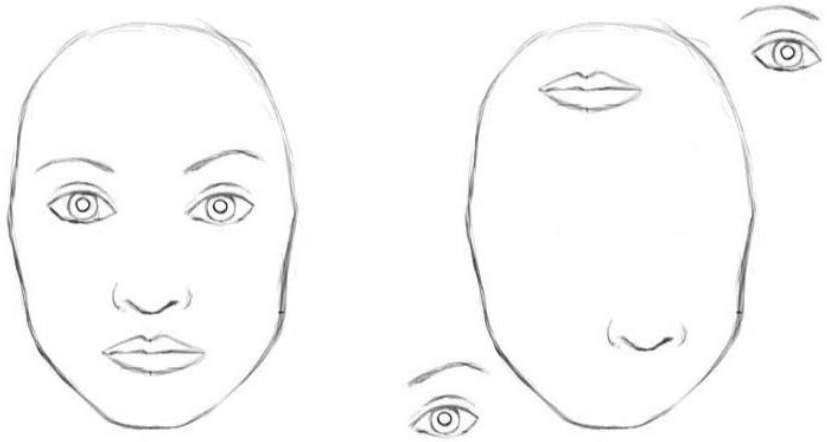


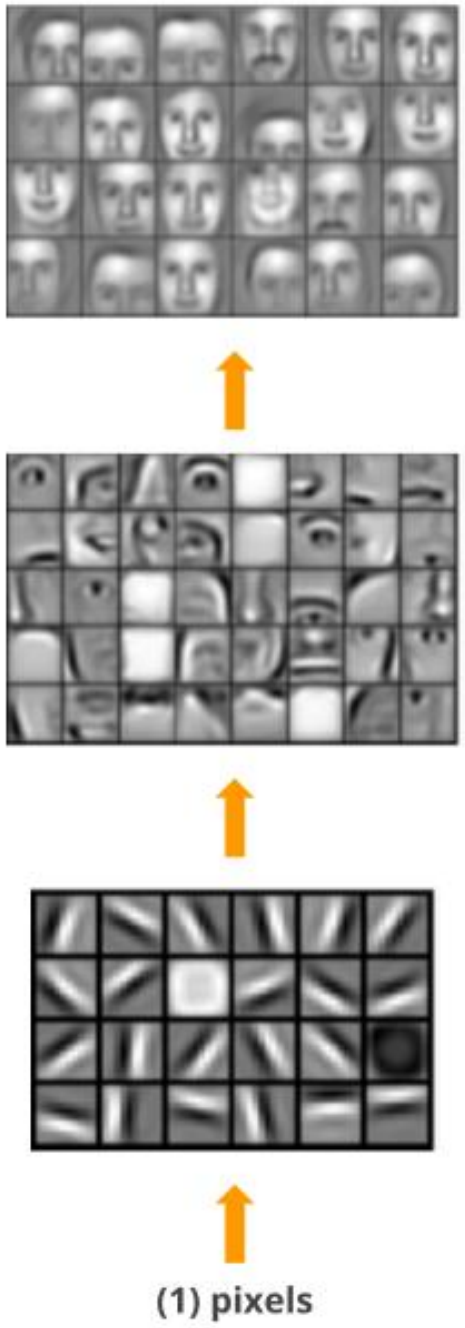


Face

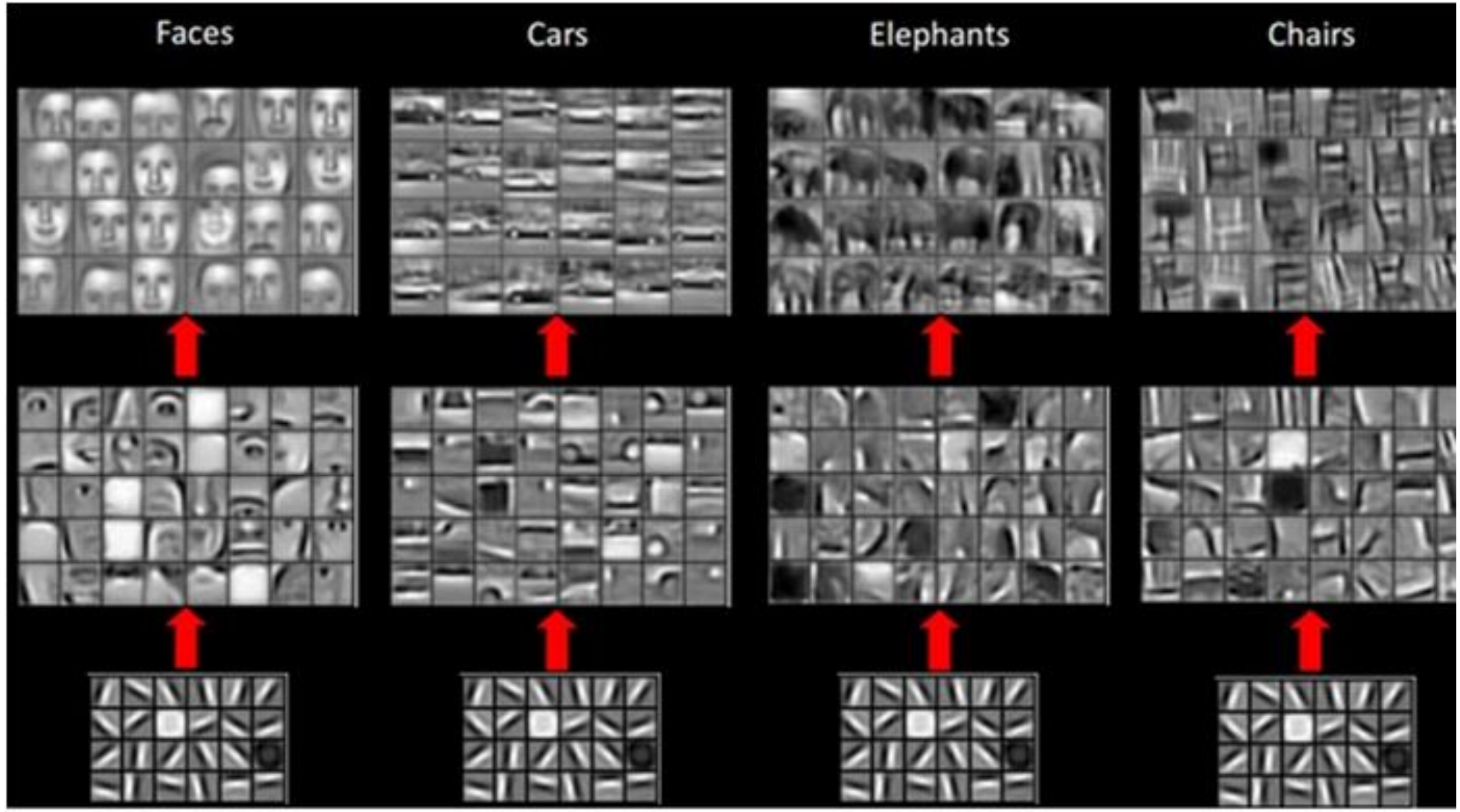


Face





Locality, parts, compositionality: reusability

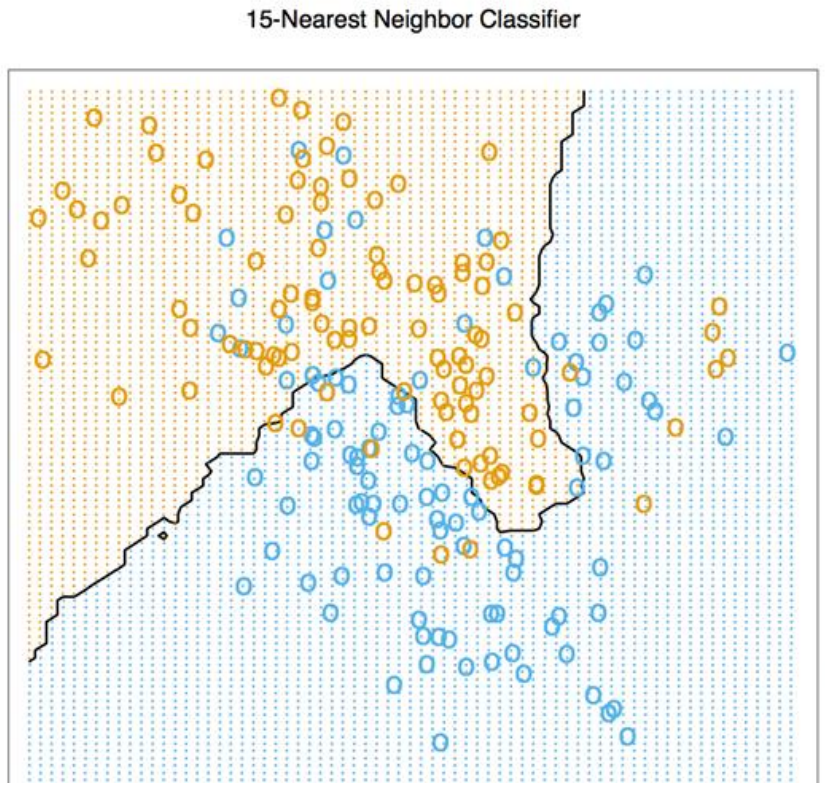

Faces
Cars
Elephants
Chairs

Why convolutionality, locality and compositionality help to have a "better" learning?

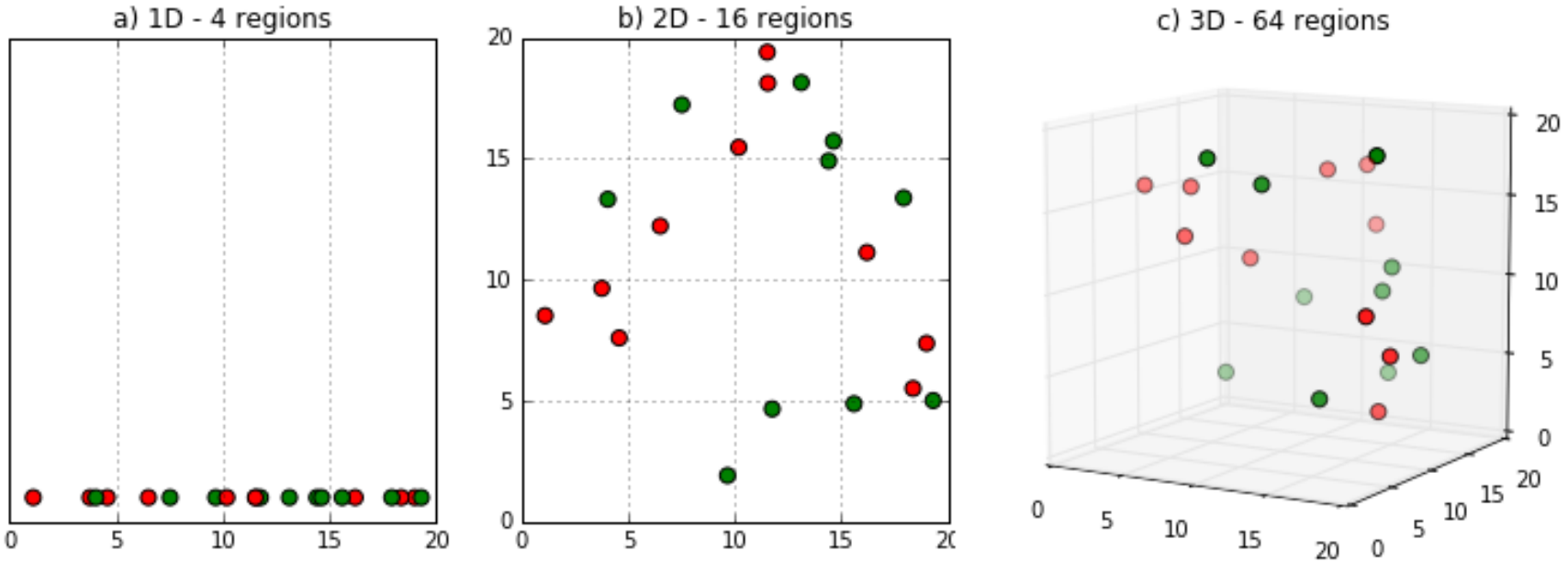
How do you estimate a function? Example with NNC

Supervised classification or regression often rely on local averages:

- Classification : you know the classes of n points from your learning database, you can classify a new point x by computing the most represented class in the neighborhood of x.



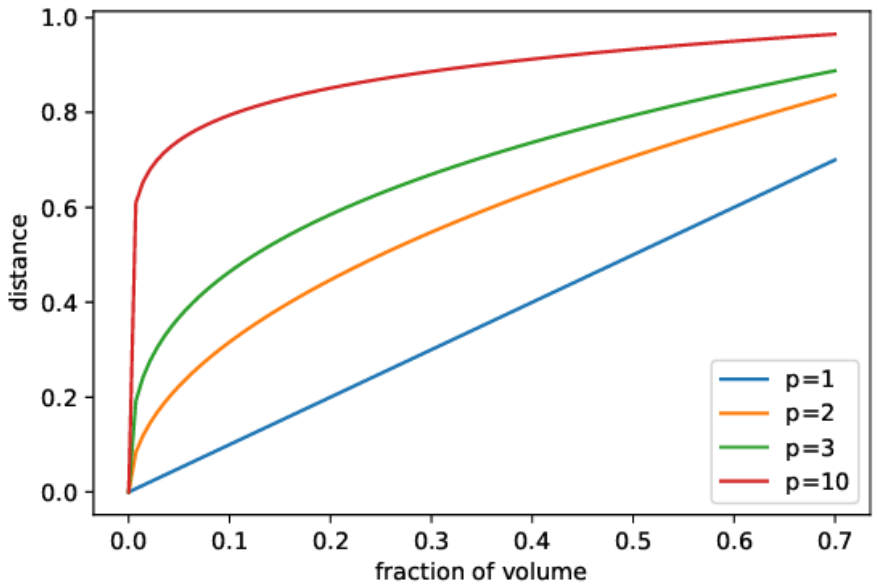
However…



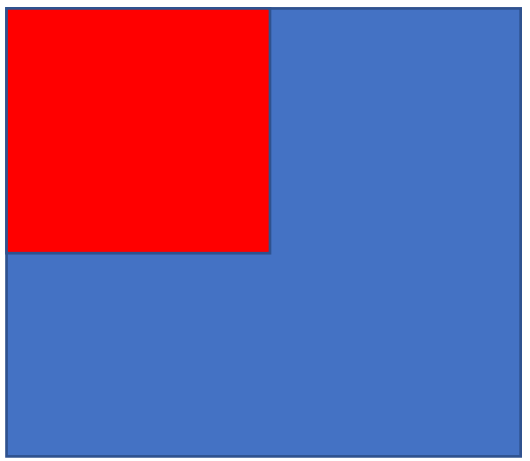
High dimensional spaces are empty

Assume your data lives in $[0,1]^p$. To capture a neighborhood which represents a fraction s of the hypercube volume, you need the edge length to be $s^{1/p}$

- $s = 0.1$, $p = 10$, $s^{1/p} = 0.63$
- $s = 0.01$, $p = 10$, $s^{1/p} = 0.8$

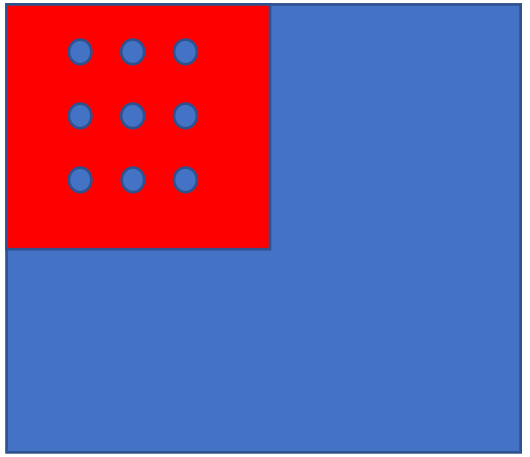


Neighborhoods are no longer local



Curse of dimensionality

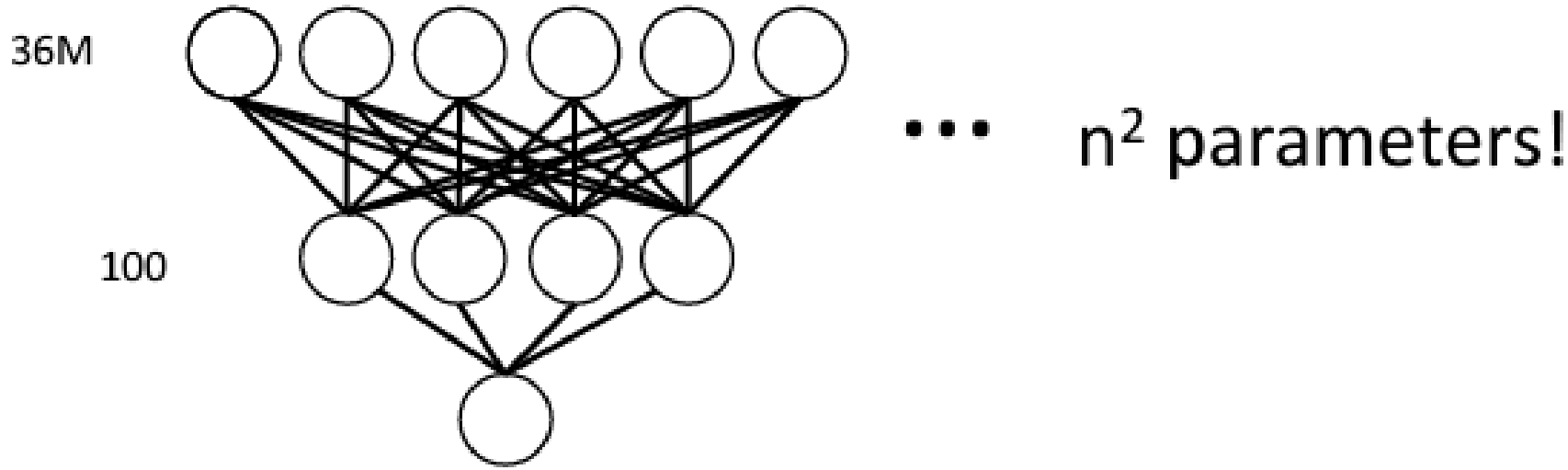
To approximate a (Lipschitz) continuous function $f\colon \mathbb{R}^d \to \mathbb{R}$ with ϵ accuracy one needs $O(\epsilon^{-d})$ samples



Input image resolution = 12 Mpixel * 3 channels = 36M elements
With $\epsilon \sim 0.1$, we need $10^{36000000}$ samples (10^{78} to 10^{82} atoms in the known, observable universe)

Same sample density (same precision): we need P^d samples

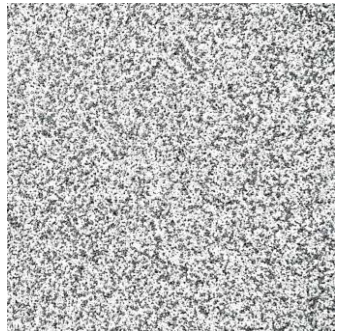
Curse of dimensionality: networks



- Input image resolution = 12 Mpixel * 3 channels = 36M elements
- MLP with one hidden layer of width=100 has **3.6B parameters**

Do we need all parameters? This is true if the image space has all possible images

but also



But the statistics of the images is translation and scale invariant

So why weight sharing helps is helping in reducing the curse of dimensionality?

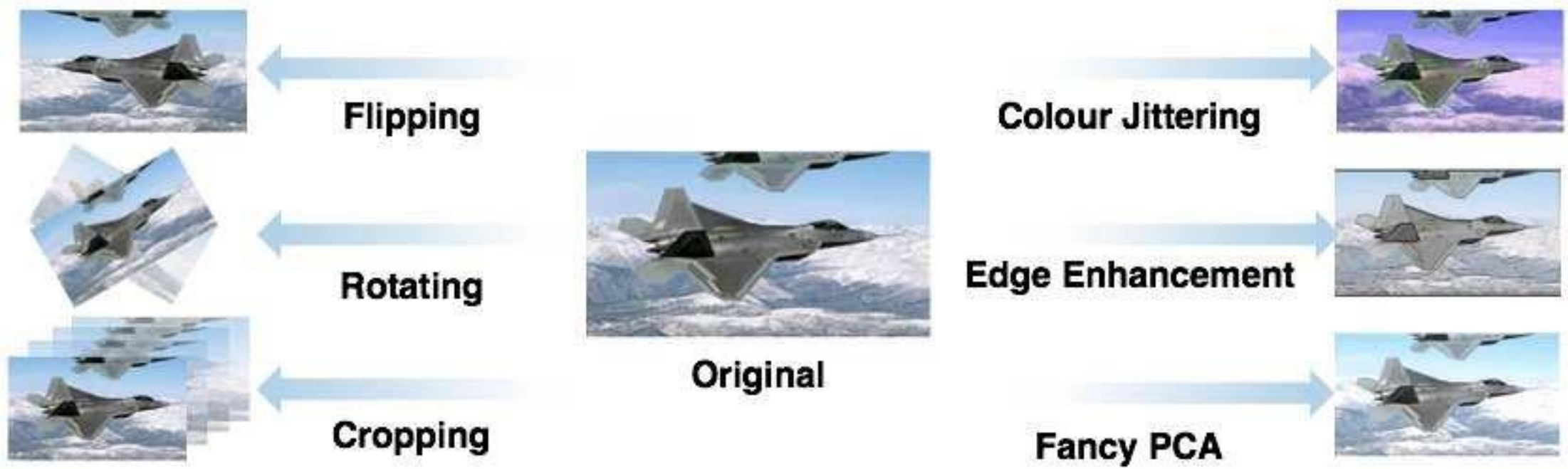
Data augmentation and other types of invariances: building an invariant representation

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

$$f(x) = \sum \sigma \langle w, g_i x \rangle = f(gx)$$

$$\{x_i\}_i \rightarrow \{g_j x_i\}_{ij}$$

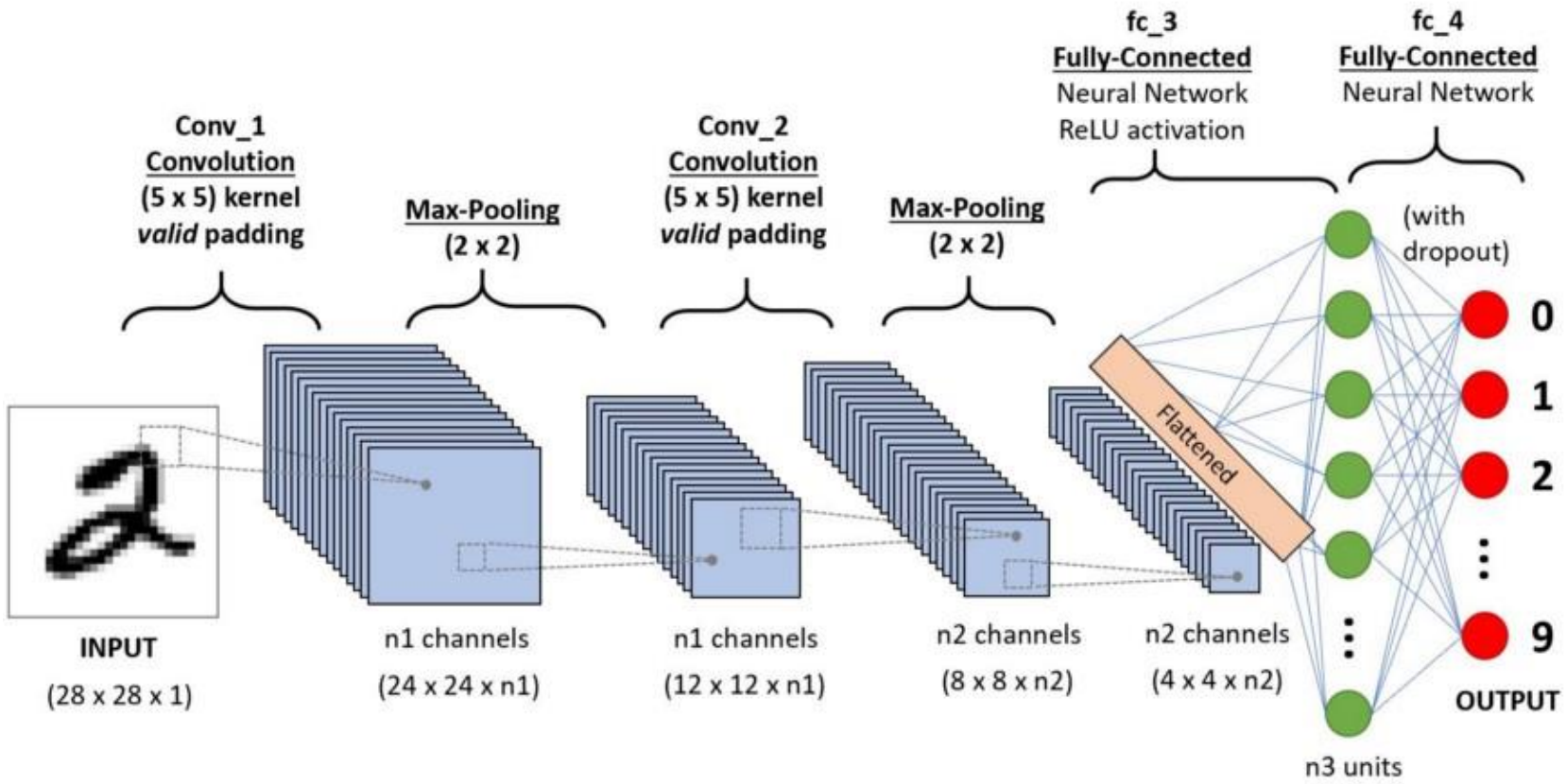
Data augmentation and other types of invariances



An example of a convolutional architecture: Lenet

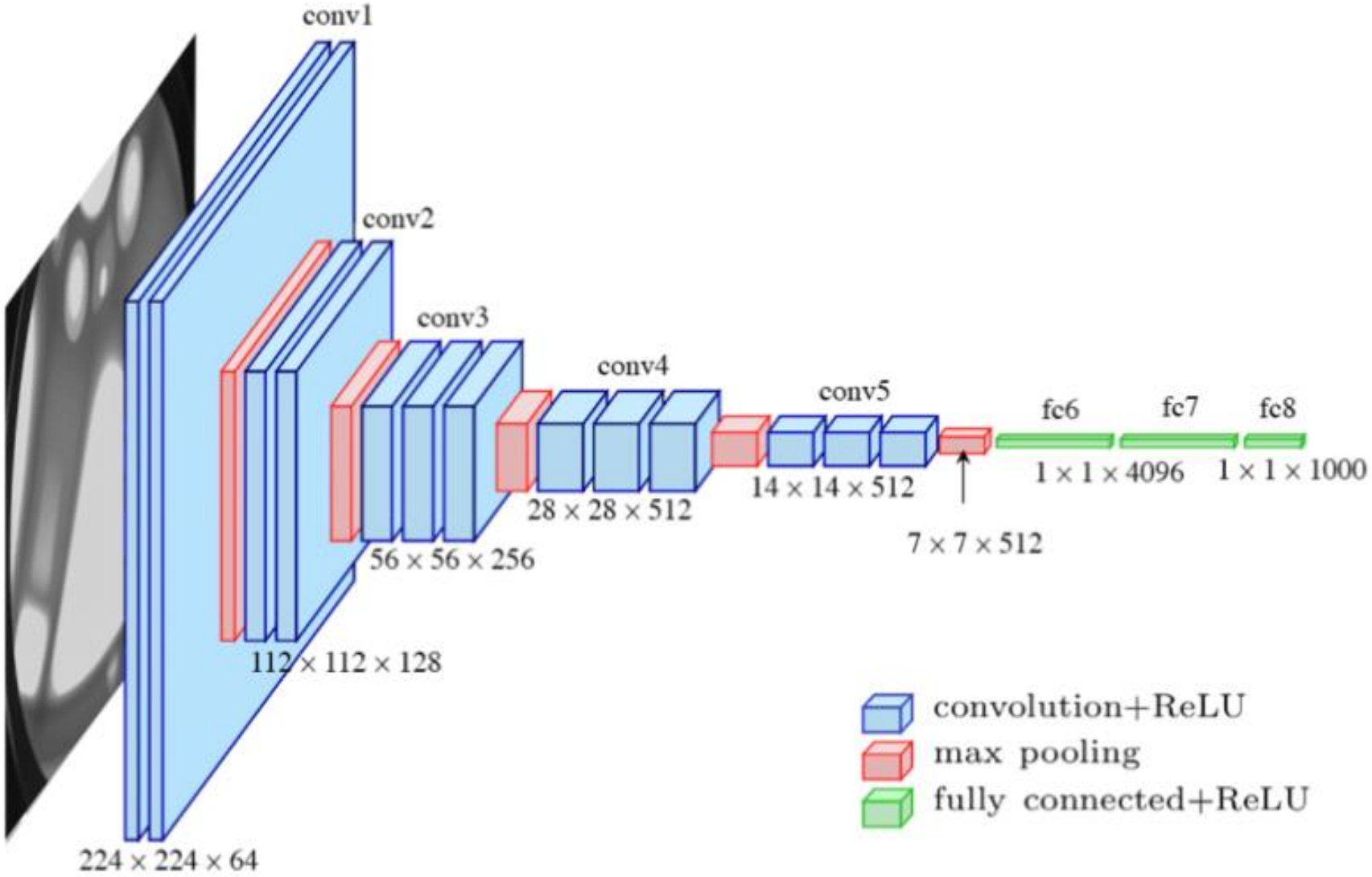
MNIST dataset

CNNs started the ML revolution



Many CNNs and image datasets

VGG architecture



Beyond convolution: Resnet

[He et al., 2015]

Hypothesis: the problem is an *optimization* problem, deeper models are harder to optimize

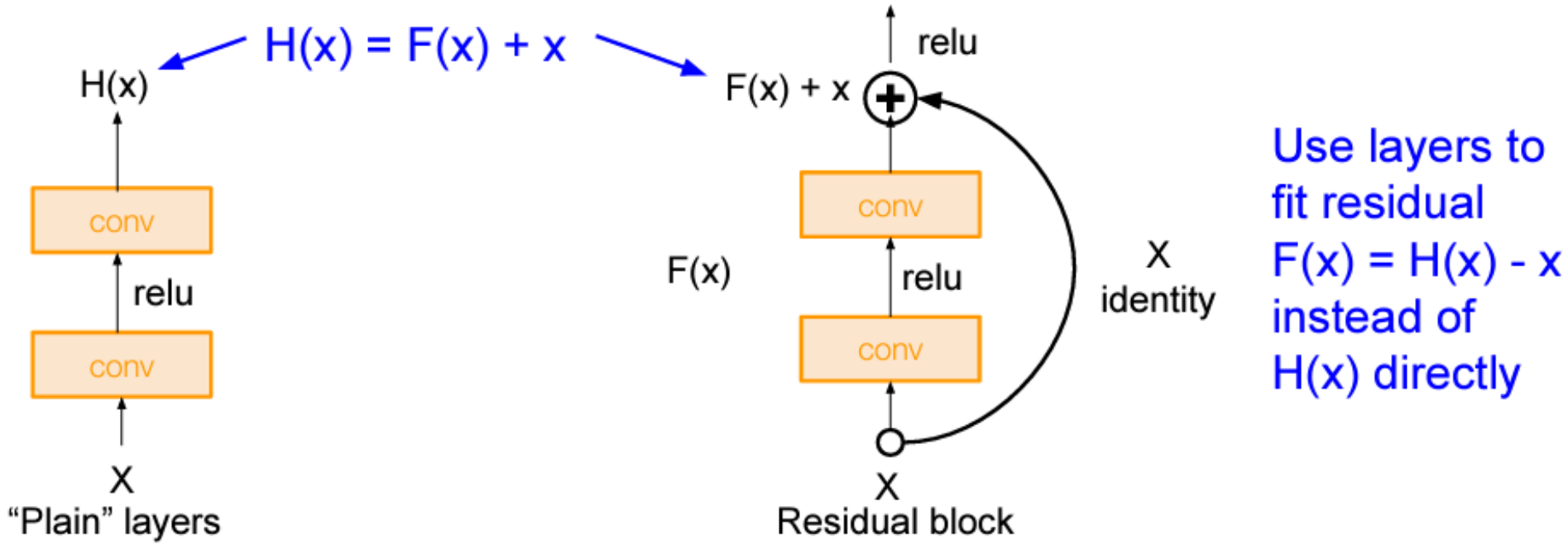
The deeper model should be able to perform at least as well as the shallower model.

A solution by construction is copying the learned layers from the shallower model and setting additional layers to identity mapping.

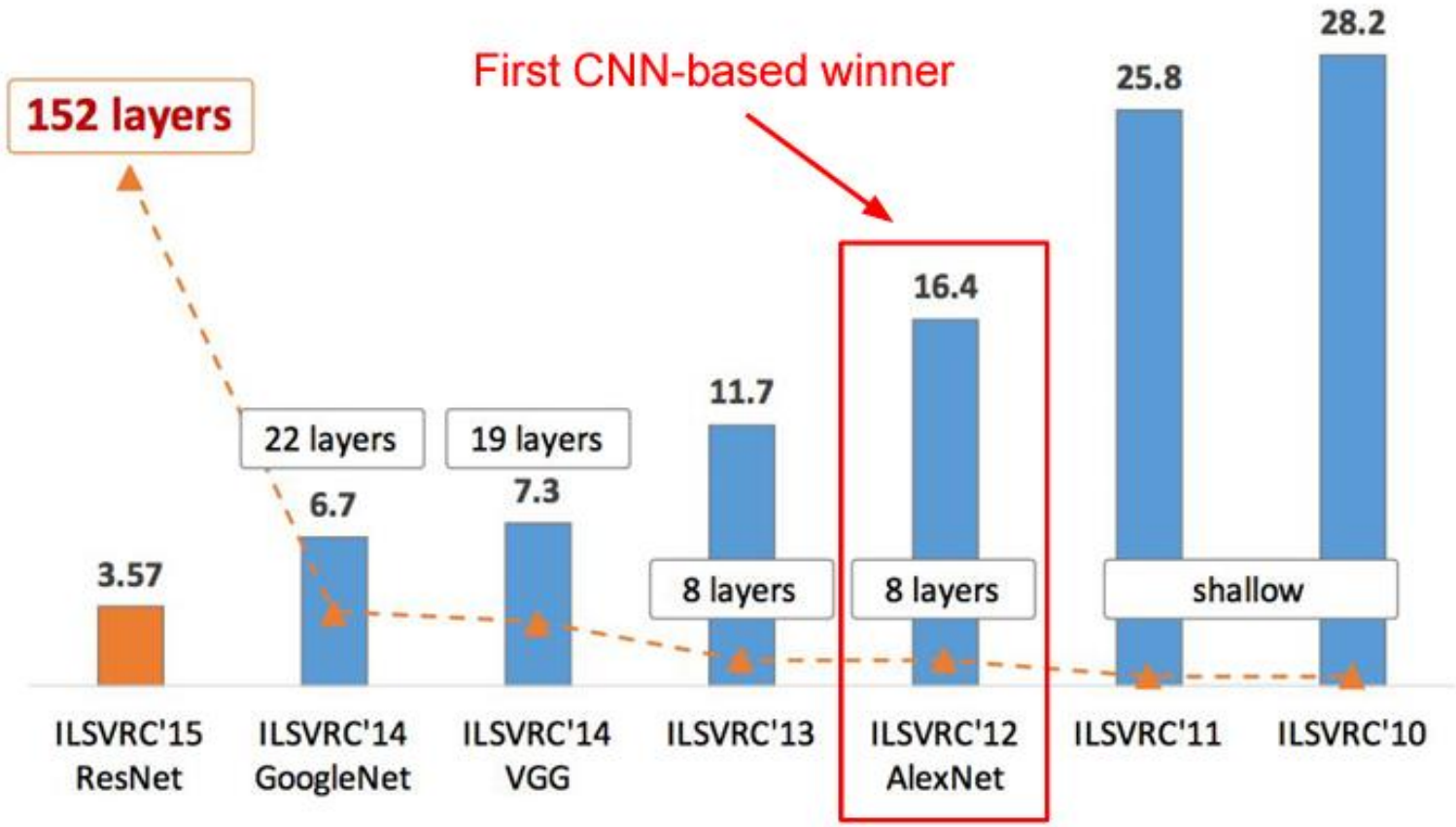
Resnet: intuition

[He et al., 2015]

Solution: Use network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping



ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



Class 5

Adversarial attacks

What is an adversarial example?

Sheepdog or Mop?

What is an adversarial example?

Sloth or Pain au chocolat?

What is an adversarial example?

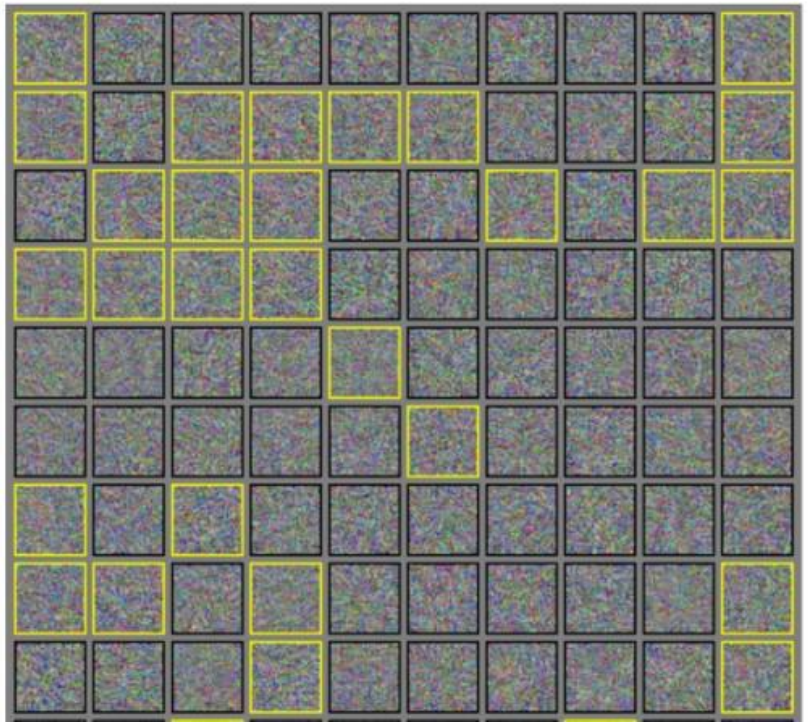
Puppy or Bagel?

What is an adversarial example?

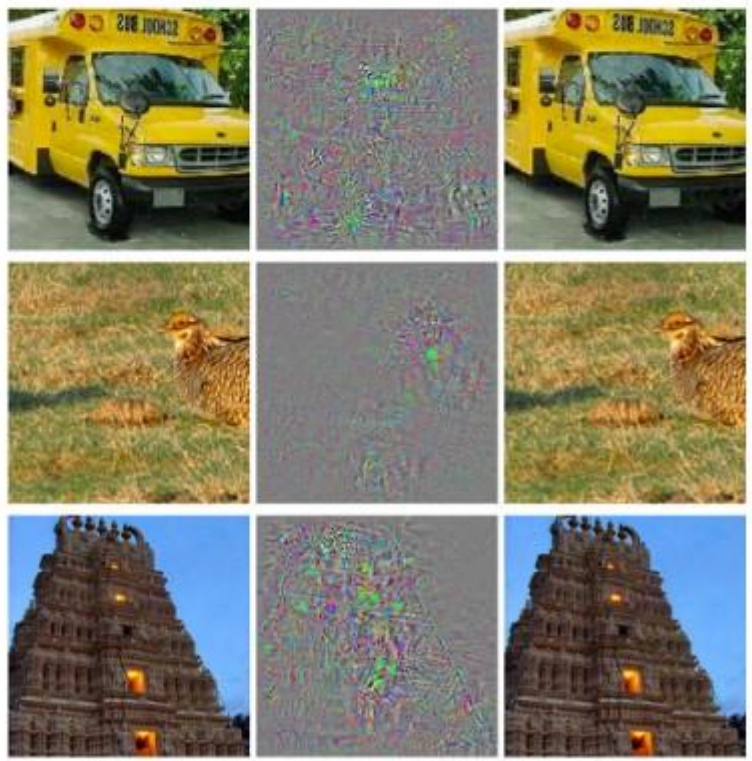
Chihuahua or Muffin?

Suppose you train a net to classify airplanes vs cars. Then…

- If you start with random noise and take one gradient step, you can often produce a confident classification as some category.
- The images highlighted in yellow are classified as "airplaine" with $> 50\%$ probability.



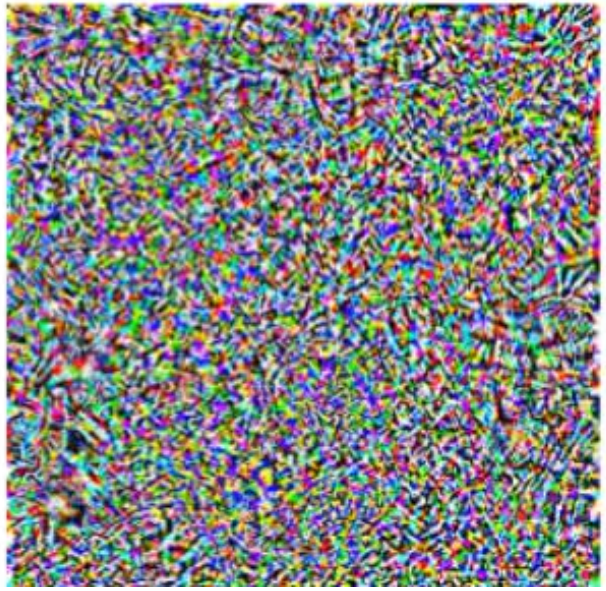
- A variant: search for the image closest to the original one which is misclassifed as a particular category (e.g. ostrich).
- This is called a targeted adversarial example, since it targets a particular category.
- The following adversarial examples are misclassified as ostriches. (Middle = perturbation $\times 10$.)



Adversarial attacks

“pig”

+ 0.005 x

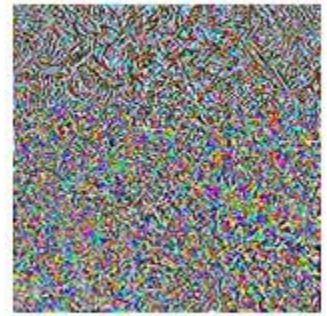


=

“airliner”

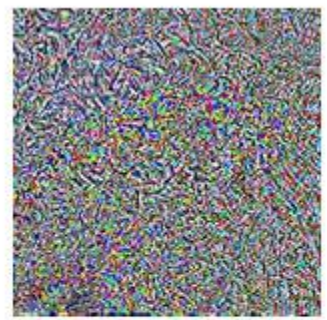
Adversarial examples for CNNs

Alps: 94%



Dog: 100%

Puffer: 98%



Crab: 100%

Dong et al. (2018). Boosting Adversarial Attacks with Momentum. *CVPR*

Adversarial Examples for different norms

Shafahi et al. (2019). Are adversarial examples inevitable? *ICLR (to appear)*

- The paper which introduced adversarial examples (in 2013) was titled "Intriguing Properties of Neural Networks."
- Now they're regarded as a serious security threat.
  - Nobody has found a reliable method yet to defend against them.
  - Adversarial examples transfer to different networks trained on a disjoint subset of the training set!
  - You don't need access to the original network; you can train up a new network to match its predictions, and then construct adversarial examples for that.
    - Attack carried out against proprietary classification networks accessed using prediction APIs (MetaMind, Amazon, Google)

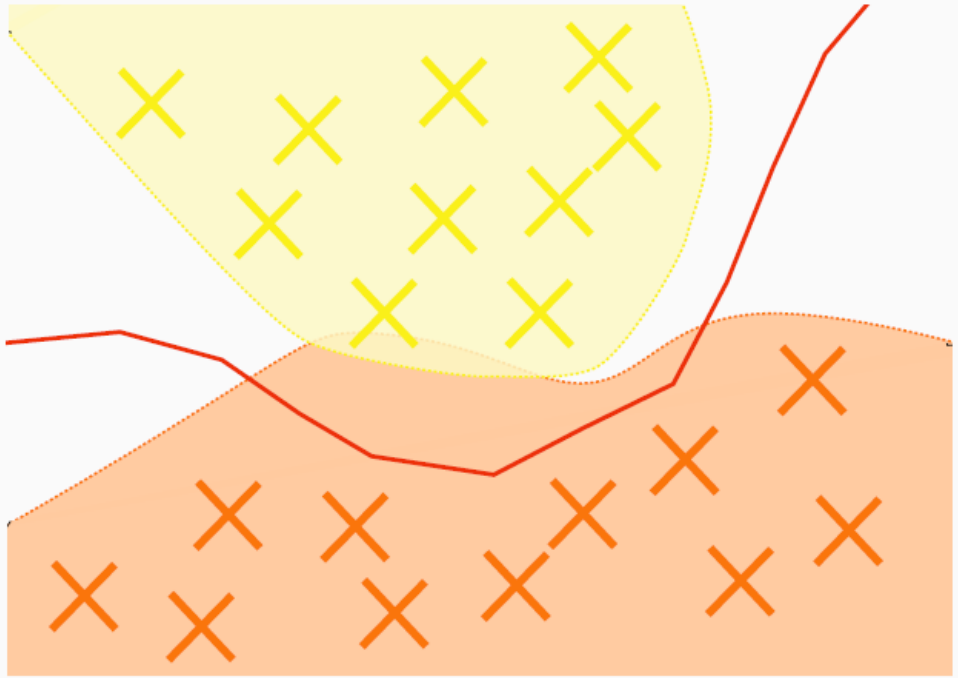
Adversarial examples are transferable!!!!

- adversarial examples are highly transferable
- it is very likely that an adversarial example of one network can fool another network
- transferability depends on the type of attack

OOD generalization problems

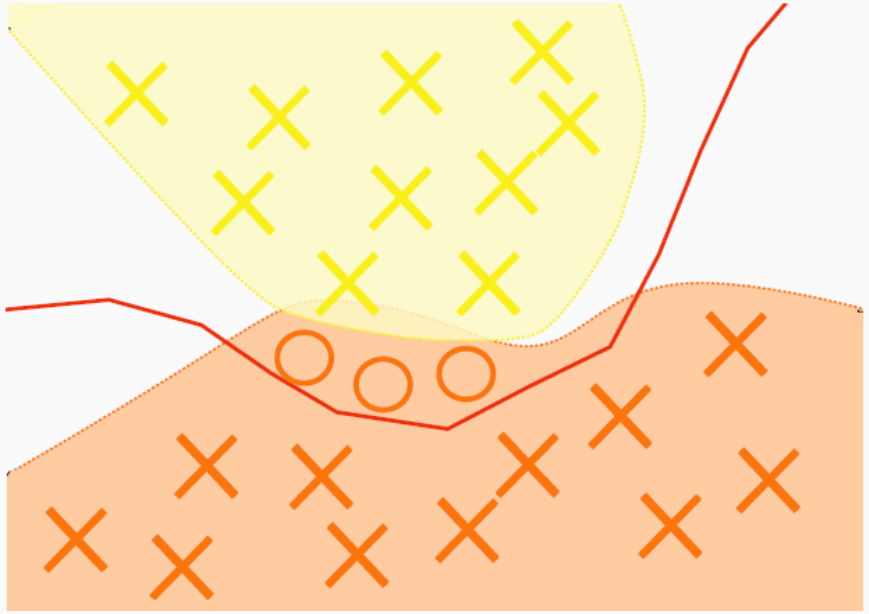
What is the intuition for adversarial attacks?

Decision Boundary of the Model



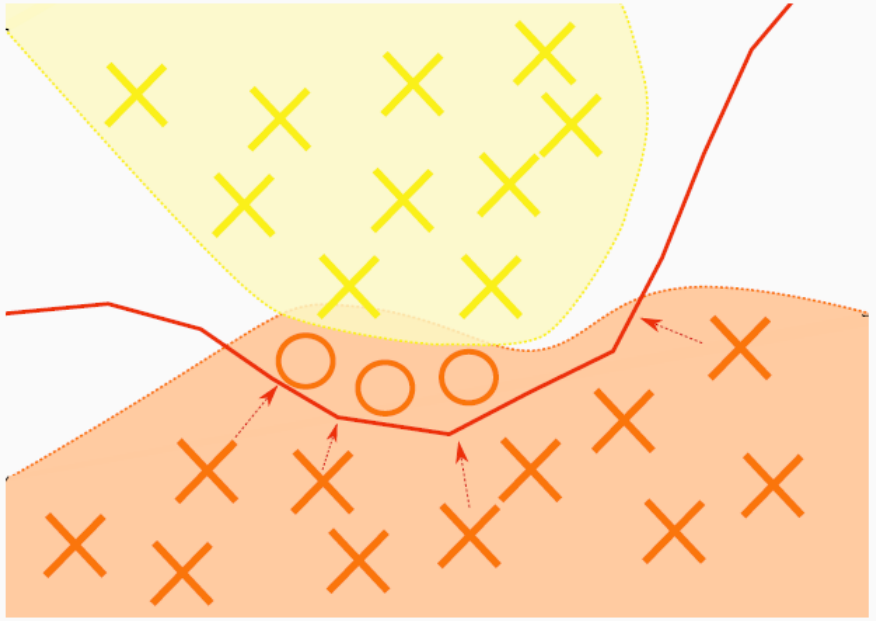
The **learned model** slightly differs from the **true** data distribution...

The Space of Adversarial Examples



... which makes room for **adversarial examples**.

Attack: Use the Adversarial Directions



- Most attacks try to move inputs across the boundary.
- Attacking with a random distortion doesn't work well in practice.

Definition 1 (Adversarial Attack)**.** *Let* $\boldsymbol{x}_0 \in \mathbb{R}^d$ *be a data point belong to class* $\mathcal{C}_i$*. Define a target class* $\mathcal{C}_t$*. An* ***adversarial attack*** *is a mapping* $\mathcal{A} : \mathbb{R}^d \to \mathbb{R}^d$ *such that the perturbed data*

$$\boldsymbol{x} = \mathcal{A}(\boldsymbol{x}_0)$$

is misclassified as $\mathcal{C}_t$*.*

Definition 2 (Additive Adversarial Attack)**.** *Let* $\boldsymbol{x}_0 \in \mathbb{R}^d$ *be a data point belong to class* $\mathcal{C}_i$*. Define a target class* $\mathcal{C}_t$*. An* ***additive*** *adversarial attack is an addition of a perturbation* $\boldsymbol{r} \in \mathbb{R}^d$ *such that the perturbed data*

$$\boldsymbol{x} = \boldsymbol{x}_0 + \boldsymbol{r}$$

is misclassified as $\mathcal{C}_t$*.*

Targeted vs untargeted attacks

Non-targeted adversarial examples: mislead the model to provide **any wrong** prediction

$$\max_{x^*} \ell(f_\theta(x^*), y)$$
$$\text{s.t. } d(x, x^*) \leq B$$

Max distance from the true prediction

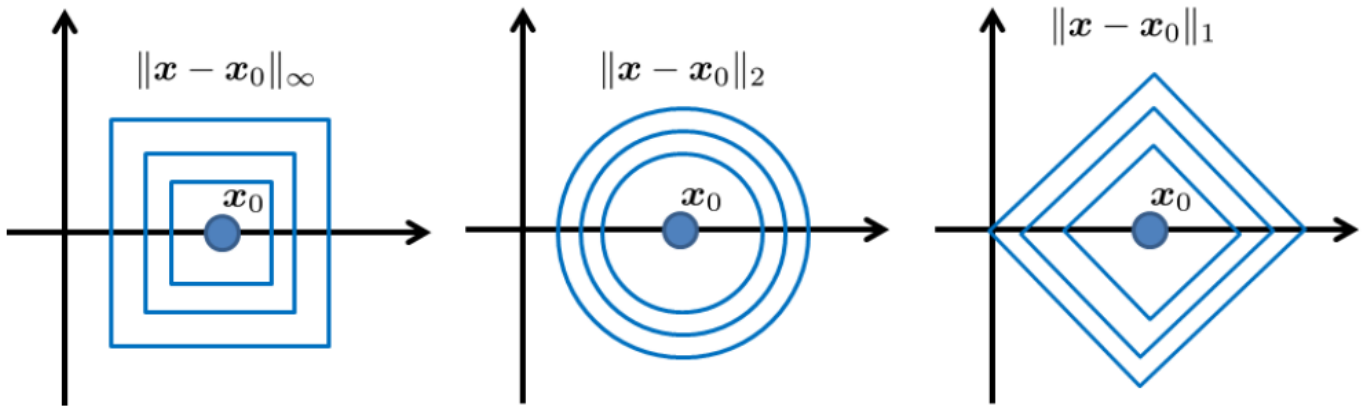
Targeted adversarial examples: mislead the model to provide the **target prediction $y^* \neq y$** specified by the adversary

$$\min_{x^*} \ell(f_\theta(x^*), y^*)$$
$$\text{s.t. } d(x, x^*) \leq B$$

Min distance from the wrong prediction

$d(x, x^*)$ is an ℓ_p norm in most existing work

Minimum norm attack



Theorem 2 (Minimum ℓ_2 Norm Attack for Two-Class Linear Classifier). *The adversarial attack to a two-class linear classifier is the solution of*

$$\underset{\boldsymbol{x}}{\text{minimize}} \quad \|\boldsymbol{x}-\boldsymbol{x}_0\|^2 \quad \text{subject to} \quad \boldsymbol{w}^T\boldsymbol{x}+w_0=0, \tag{3.25}$$

which is given by

$$\boldsymbol{x}^* = \boldsymbol{x}_0 - \left(\frac{\boldsymbol{w}^T\boldsymbol{x}_0+w_0}{\|\boldsymbol{w}\|_2}\right)\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|_2}. \tag{3.26}$$

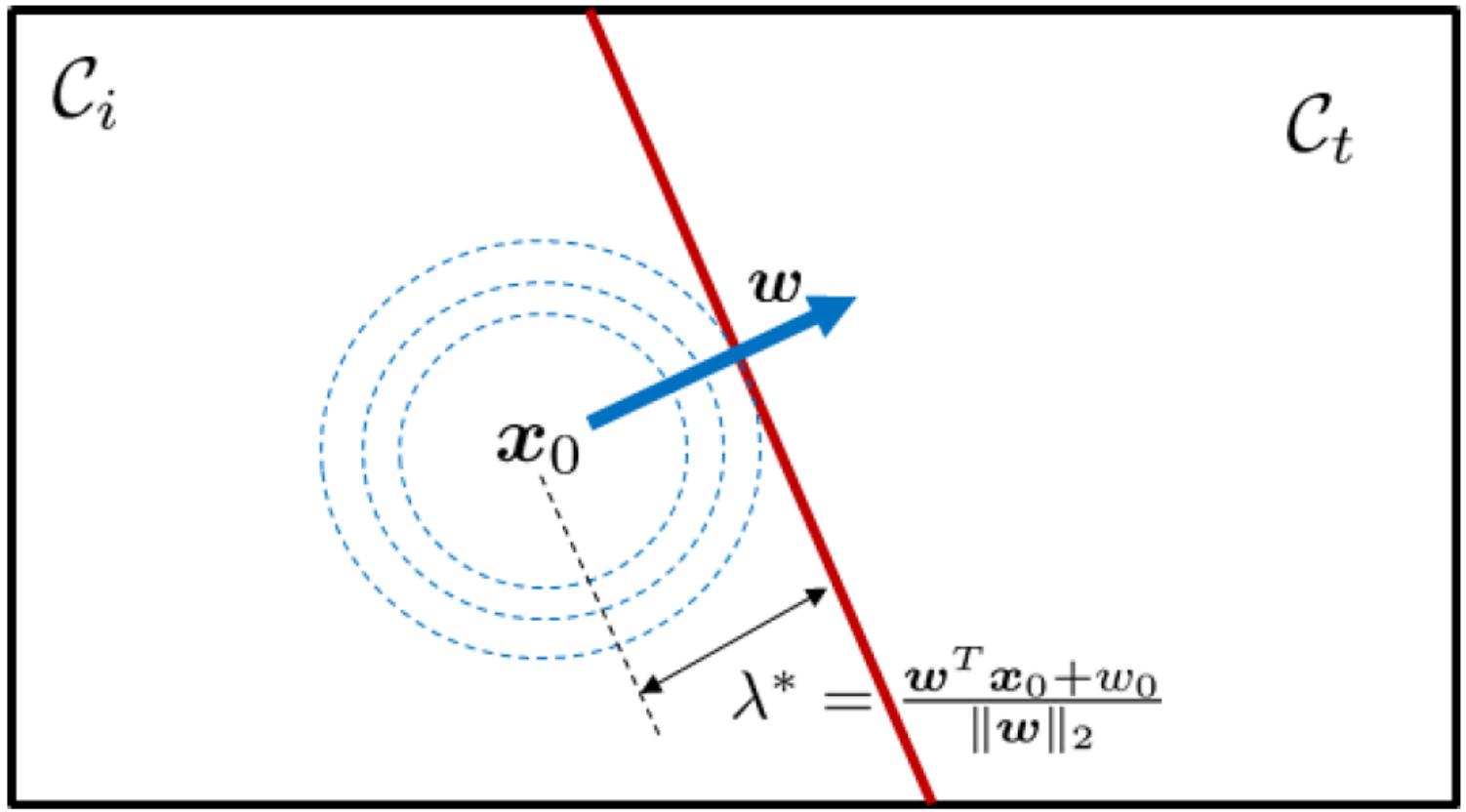
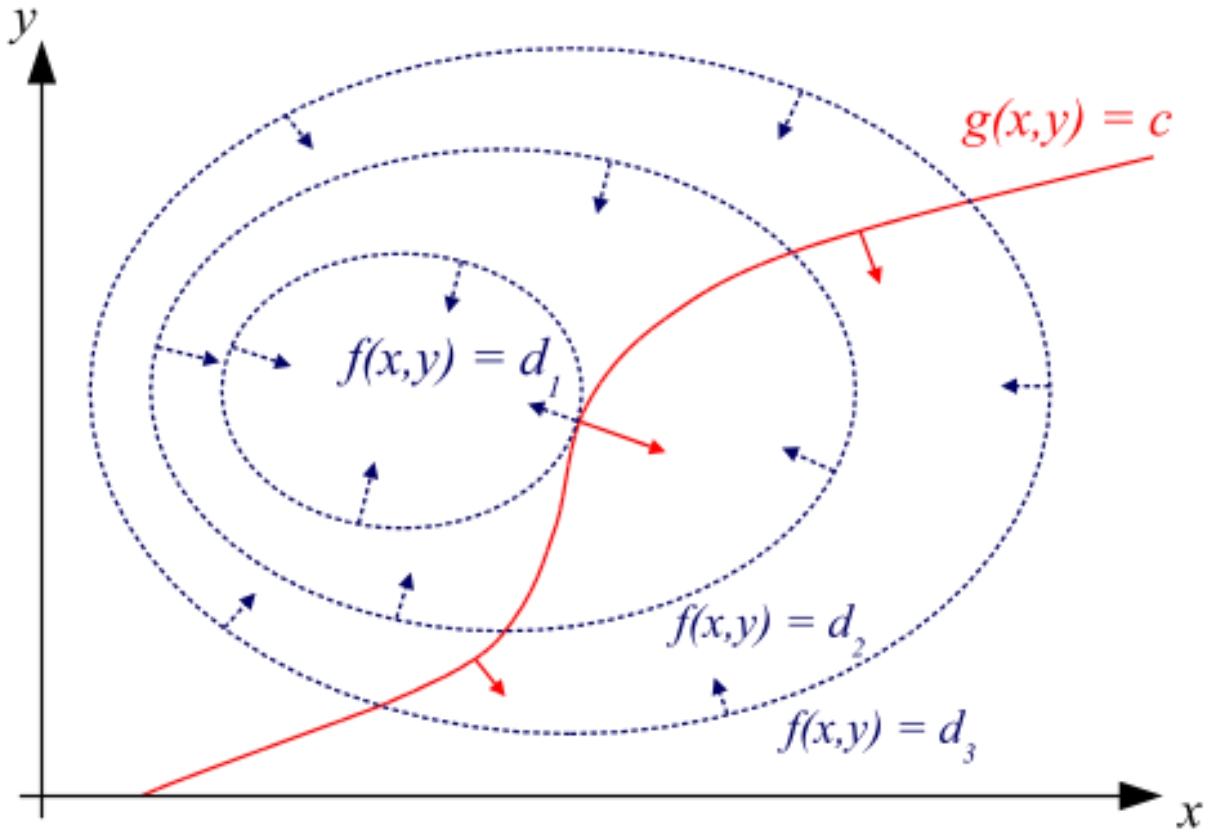


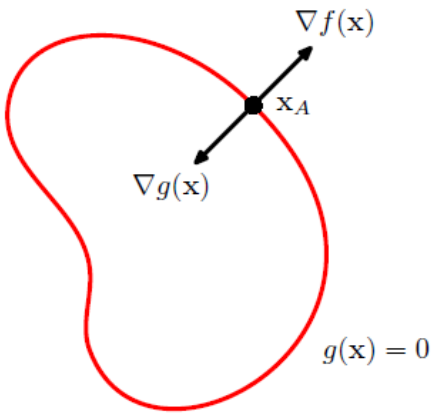
Figure 3.8: Geometry of minimum-norm attack for a two-class linear classifier with objective function $\|\boldsymbol{x} - \boldsymbol{x}_0\|^2$. The solution is a projection of the input $\boldsymbol{x}_0$ onto the separating hyperplane of the classifier.

Recap: lagrangian multipliers



$$\max \; f(x,y), \quad s.t. \quad g(x,y) = 0$$

Lagrange Multipliers



Consider the problem:

$$\max_{\boldsymbol{x}} f(\boldsymbol{x})$$
$$s.t. \quad g(\boldsymbol{x}) = 0$$

- Points on $g(\boldsymbol{x}) = 0$ must have $\nabla g(\boldsymbol{x})$ normal to surface
- A stationary point must have no change in f in the direction of the constraint surface, so $\nabla f(\boldsymbol{x})$ must also be normal to the surface.
  - So there must be some $\lambda \neq 0$ such that $\nabla f(\boldsymbol{x}) + \lambda \nabla g(\boldsymbol{x}) = 0$
- Define Lagrangian:

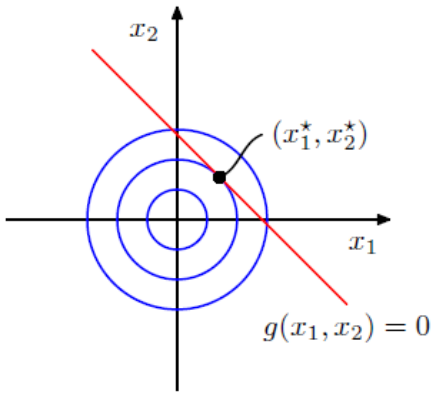
$$L(\boldsymbol{x}, \lambda) = f(\boldsymbol{x}) + \lambda g(\boldsymbol{x})$$

- Stationary points of $L(\boldsymbol{x}, \lambda)$ have $\nabla_x L(\boldsymbol{x}, \lambda) = \nabla f(\boldsymbol{x}) + \lambda \nabla g(\boldsymbol{x}) = 0$ and $\nabla_\lambda L(\boldsymbol{x}, \lambda) = g(\boldsymbol{x}) = 0$

This is because they are level curves

This is because they are the same vector up to a multiplicative constant

Lagrange Multipliers Example



- Consider the problem

$$\max_{\boldsymbol{x}} f(x_1, x_2) = 1 - x_1^2 - x_2^2$$
$$s.t. \quad g(x_1, x_2) = x_1 + x_2 - 1 = 0$$

- Lagrangian:

$$L(\boldsymbol{x}, \lambda) = 1 - x_1^2 - x_2^2 + \lambda(x_1 + x_2 - 1)$$

- Stationary points require:

$$\begin{aligned} \partial L/\partial x_1 &= -2x_1 + \lambda = 0 \\ \partial L/\partial x_2 &= -2x_2 + \lambda = 0 \\ \partial L/\partial \lambda &= x_1 + x_2 - 1 = 0 \end{aligned}$$

- So stationary point is $(x_1^*, x_2^*) = (\frac{1}{2}, \frac{1}{2})$, $\lambda = 1$

How to calculate the adversarial perturbation?

Proof. The Lagrange multiplier of the constrained optimization is given by

$$\mathcal{L}(\boldsymbol{x}, \lambda) = \frac{1}{2}\|\boldsymbol{x} - \boldsymbol{x}_0\|^2 + \lambda(\boldsymbol{w}^T\boldsymbol{x} + w_0).$$

The solution of the optimization is the saddle point $(\boldsymbol{x}^*, \lambda^*)$ such that $\nabla_{\boldsymbol{x}}\mathcal{L} = 0$ and $\nabla_\lambda\mathcal{L} = 0$.

Taking derivative with respect to $\boldsymbol{x}$ and λ yields

$$\nabla_{\boldsymbol{x}}\mathcal{L} = \boldsymbol{x} - \boldsymbol{x}_0 + \lambda\boldsymbol{w} = 0,$$
$$\nabla_\lambda\mathcal{L} = \boldsymbol{w}^T\boldsymbol{x} + w_0 = 0.$$

How to calculate the adversarial perturbation?

Multiplying the first equation by $\boldsymbol{w}^T$ yields

$$0 = \boldsymbol{w}^T\boldsymbol{x} - \boldsymbol{w}^T\boldsymbol{x}_0 + \lambda\|\boldsymbol{w}\|^2 = -w_0 - \boldsymbol{w}^T\boldsymbol{x}_0 + \lambda\|\boldsymbol{w}\|^2. \tag{3.27}$$

Thus, the optimal λ is

$$\lambda^* = (\boldsymbol{w}^T\boldsymbol{x}_0 + w_0)/\|\boldsymbol{w}\|^2. \tag{3.28}$$

Correspondingly, the optimal $\boldsymbol{x}$ is

$$\boldsymbol{x}^* = \boldsymbol{x}_0 - \lambda^*\boldsymbol{w} = \boldsymbol{x}_0 - \left(\frac{\boldsymbol{w}^T\boldsymbol{x}_0 + w_0}{\|\boldsymbol{w}\|}\right)\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|_2}.$$

□

The result of this theorem provides many useful interpretation. First, the search direction is $-\boldsymbol{w}$ (or $-\boldsymbol{w}/\|\boldsymbol{w}\|_2$ for unit norm.) The negative sign is there because $\boldsymbol{w} = \boldsymbol{w}_i - \boldsymbol{w}_t$ is pointing from class $\mathcal{C}_t$ to $\mathcal{C}_i$, which is opposite to the desired search direction. The search step $\frac{\boldsymbol{w}^T\boldsymbol{x}_0 + w_0}{\|\boldsymbol{w}\|_2}$ measures the distance from the input $\boldsymbol{x}_0$ to the target class $\mathcal{C}_t$. This is coherent to the projection perspective we presented earlier.

Deep fool: non-linear decision boundary

Definition 6 (DeepFool Attack by Moosavi-Dezfooli et al. 2016). *The DeepFool attack for a two-class classification generates the attack by solving the optimization*

$$\underset{\boldsymbol{x}}{minimize} \;\; \|\boldsymbol{x} - \boldsymbol{x}_0\|^2 \;\; subject \;\; to \;\; g(\boldsymbol{x}) = 0, \tag{3.29}$$

where $g(\boldsymbol{x}) = 0$ is the nonlinear decision boundary separating the two classes.

Corollary 1 (DeepFool Algorithm for Two-Class Problem). *An iterative procedure to obtain the DeepFool attack solution is*

$$\begin{aligned} \boldsymbol{x}^{(k+1)} &= \underset{\boldsymbol{x}}{argmin} \;\; \|\boldsymbol{x} - \boldsymbol{x}^{(k)}\|^2 \;\; subject \;\; to \;\; g(\boldsymbol{x}^{(k)}) + \nabla_{\boldsymbol{x}} g(\boldsymbol{x}^{(k)})^T (\boldsymbol{x} - \boldsymbol{x}^{(k)}) = 0 \\ &= \boldsymbol{x}^{(k)} - \left(\frac{g(\boldsymbol{x}^{(k)})}{\|\nabla_{\boldsymbol{x}} g(\boldsymbol{x}^{(k)})\|^2} \right) \nabla_{\boldsymbol{x}} g(\boldsymbol{x}^{(k)}). \end{aligned} \tag{3.31}$$

Due to the nonlinearity of $g(\boldsymbol{x})$, it is generally very difficult to derive a closed-form solution. The numerical procedure to compute the solution can be derived by iteratively updating $\boldsymbol{x}$, where each iteration minimizes $\boldsymbol{x}$ over the first order approximation of $g(\boldsymbol{x})$:

$$g(\boldsymbol{x}) \approx g(\boldsymbol{x}^{(k)}) + \nabla_x g(\boldsymbol{x}^{(k)})^T (\boldsymbol{x} - \boldsymbol{x}^{(k)}),$$

where $\boldsymbol{x}^{(k)}$ is the k-th iterate of the solution. In other words, we are defining a procedure

$$\boldsymbol{x}^{(k+1)} = \underset{\boldsymbol{x}}{\operatorname{argmin}} \ \|\boldsymbol{x} - \boldsymbol{x}^{(k)}\|^2 \ \text{ subject to } \ g(\boldsymbol{x}^{(k)}) + \nabla_x g(\boldsymbol{x}^{(k)})^T (\boldsymbol{x} - \boldsymbol{x}^{(k)}) = 0. \tag{3.30}$$

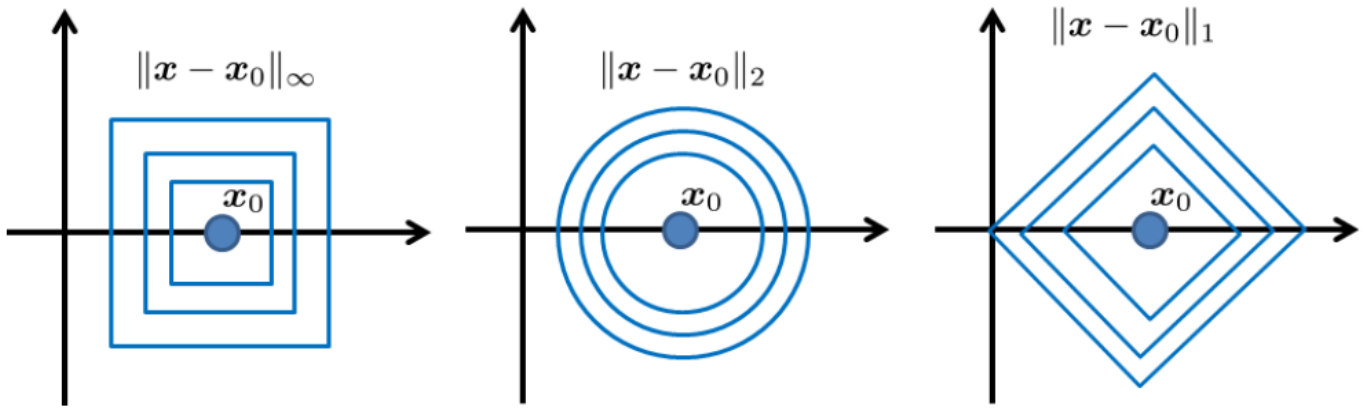
By identifying $\boldsymbol{w}^{(k)} = \nabla_x g(\boldsymbol{x}^{(k)})$ and $w_0^{(k)} = g(\boldsymbol{x}^{(k)}) - \nabla_x g(\boldsymbol{x}^{(k)})^T \boldsymbol{x}^{(k)}$, the linearized problem Equation (3.30) becomes

$$\boldsymbol{x}^{(k+1)} = \underset{\boldsymbol{x}}{\operatorname{argmin}} \ \|\boldsymbol{x} - \boldsymbol{x}^{(k)}\|^2 \ \text{ subject to } \ (\boldsymbol{w}^{(k)})^T \boldsymbol{x} + w_0^{(k)} = 0$$

The solution can thus be found by using Equation (3.26), yielding

$$\begin{aligned} \boldsymbol{x}^{(k+1)} &= \boldsymbol{x}^{(k)} - \left(\frac{(\boldsymbol{w}^{(k)})^T \boldsymbol{x}^{(k)} + w_0^{(k)}}{\|\boldsymbol{w}^{(k)}\|^2} \right) \boldsymbol{w}^{(k)} \\ &= \boldsymbol{x}^{(k)} - \left(\frac{g(\boldsymbol{x}^{(k)})}{\|\nabla_x g(\boldsymbol{x}^{(k)})\|^2} \right) \nabla_x g(\boldsymbol{x}^{(k)}). \end{aligned}$$

Exercise!

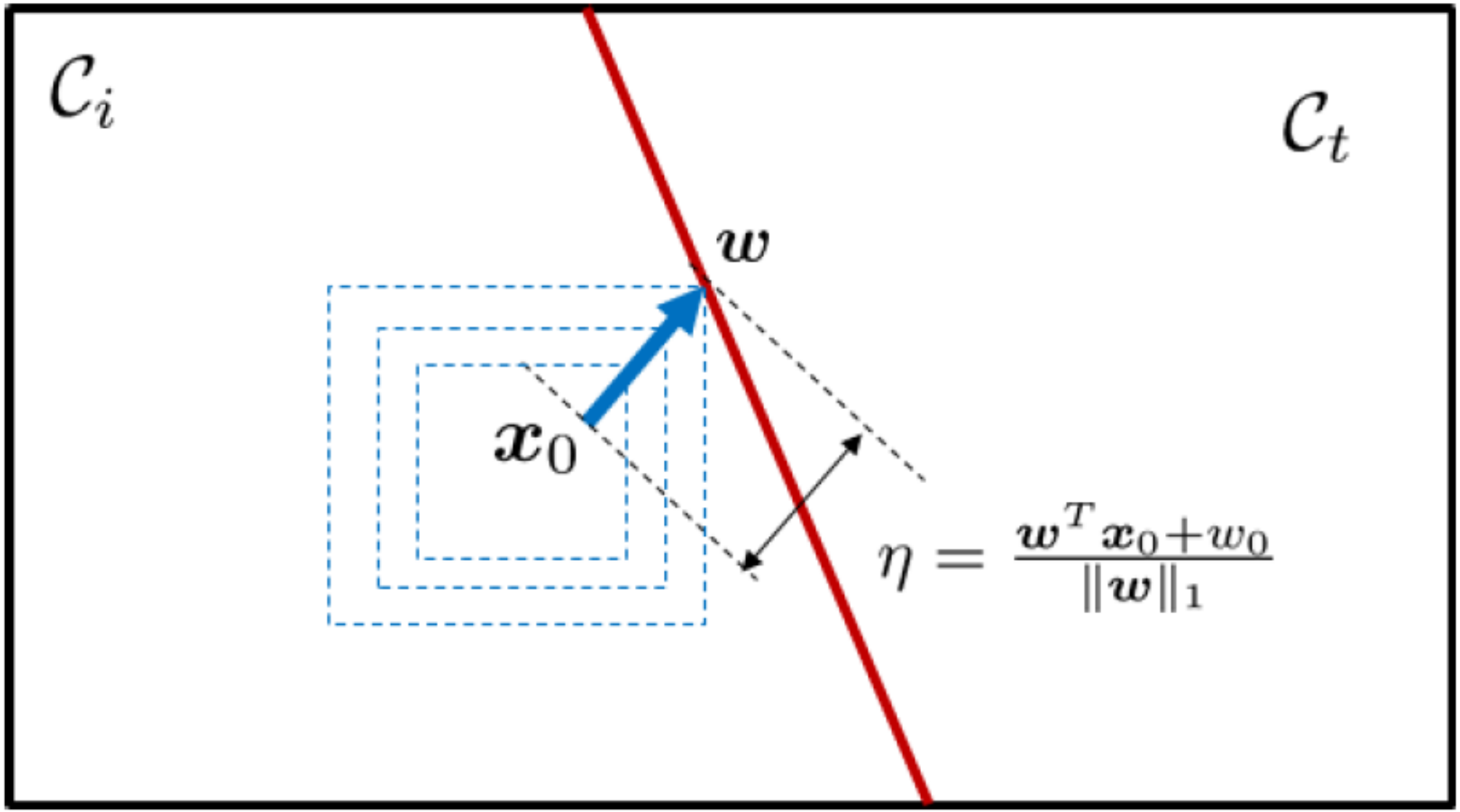


Theorem 4 (Minimum ℓ_∞ Norm Attack for Two-Class Linear Classifier)**.** *The minimum ℓ_∞ norm attack for a two-class linear classifier, i.e.,*

$$\underset{\boldsymbol{x}}{\text{minimize}} \ \|\boldsymbol{x} - \boldsymbol{x}_0\|_\infty \ \text{ subject to } \ \boldsymbol{w}^T\boldsymbol{x} + w_0 = 0 \tag{3.33}$$

is given by

$$\boldsymbol{x} = \boldsymbol{x}_0 - \left(\frac{\boldsymbol{w}^T\boldsymbol{x}_0 + w_0}{\|\boldsymbol{w}\|_1}\right) \cdot sign(\boldsymbol{w}). \tag{3.34}$$


C_i
C_t
w
x_0
$\eta = \frac{w^T x_0 + w_0}{\|w\|_1}$

How is minimum ℓ_∞ norm attack related to the maximum allowable ℓ_∞ norm attack? Recall that the maximum allowable attack is given by

$$\underset{\boldsymbol{x}}{\text{minimize}} \;\; \boldsymbol{w}^T\boldsymbol{x} + w_0 \;\; \text{subject to} \;\; \|\boldsymbol{x} - \boldsymbol{x}_0\|_\infty \leq \eta. \tag{3.35}$$

By defining $\boldsymbol{x} = \boldsymbol{x}_0 + \boldsymbol{r}$, we have $\boldsymbol{w}^T\boldsymbol{x} + w_0 = \boldsymbol{w}^T\boldsymbol{x}_0 + \boldsymbol{w}^T\boldsymbol{r} + w_0$. By defining $b_0 = -(\boldsymbol{w}^T\boldsymbol{x}_0 + w_0)$, the optimization can be rewritten as

$$\underset{\boldsymbol{r}}{\text{minimize}} \;\; \boldsymbol{w}^T\boldsymbol{r} - b_0 \;\; \text{subject to} \;\; \|\boldsymbol{r}\|_\infty \leq \eta. \tag{3.36}$$

Using the Holder's inequality (the negative side), we can show that

$$\boldsymbol{w}^T\boldsymbol{r} \geq -\|\boldsymbol{r}\|_\infty\|\boldsymbol{w}\|_1 \geq -\eta\|\boldsymbol{w}\|_1.$$

The lower bound of $\boldsymbol{w}^T\boldsymbol{r}$ is attained when $\boldsymbol{r} = -\eta \cdot \text{sign}(\boldsymbol{w})$, because

$$\boldsymbol{w}^T\boldsymbol{r} = -\eta\boldsymbol{w}^T\text{sign}(\boldsymbol{w}) = -\eta\sum_{i=1}^{d} w_i\text{sign}(w_i) = -\eta\sum_{i=1}^{d}|w_i| = -\eta\|\boldsymbol{w}\|_1.$$

Therefore, the attacked data point $\boldsymbol{x}$ is $\boldsymbol{x} = \boldsymbol{x}_0 - \eta \cdot \text{sign}(\boldsymbol{w})$. This leads to the following theorem.

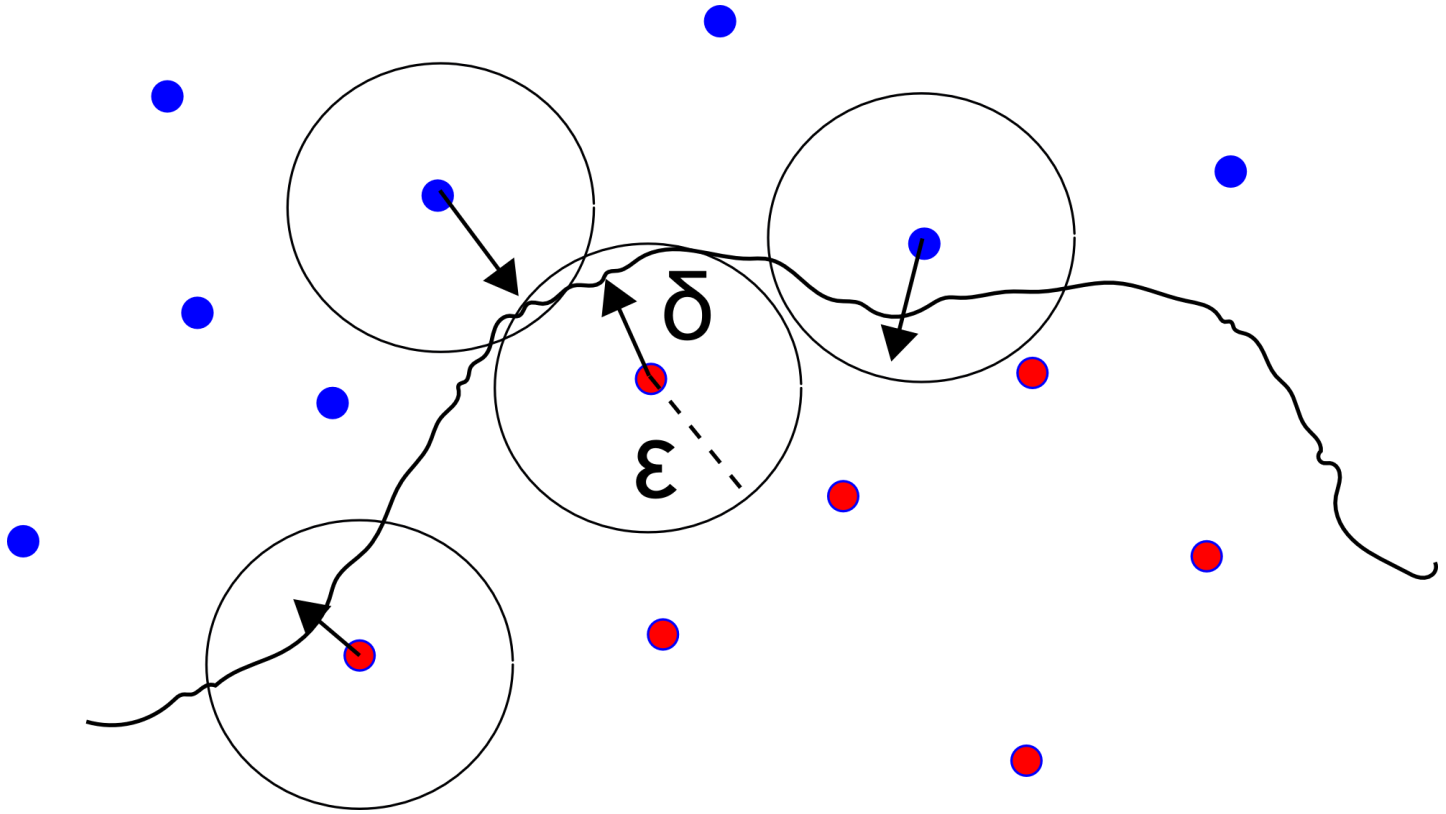
Theorem 5 (Maximum Allowable ℓ_∞ Norm Attack of Two-Class Linear Classifier)**.** *The maximum allowable ℓ_∞ norm attack for a two-class linear classifier, i.e.,*

$$\underset{\boldsymbol{x}}{\textit{minimize}} \;\; \boldsymbol{w}^T\boldsymbol{x} + w_0 \;\; \textit{subject to} \;\; \|\boldsymbol{x} - \boldsymbol{x}_0\|_\infty \leq \eta. \tag{3.37}$$

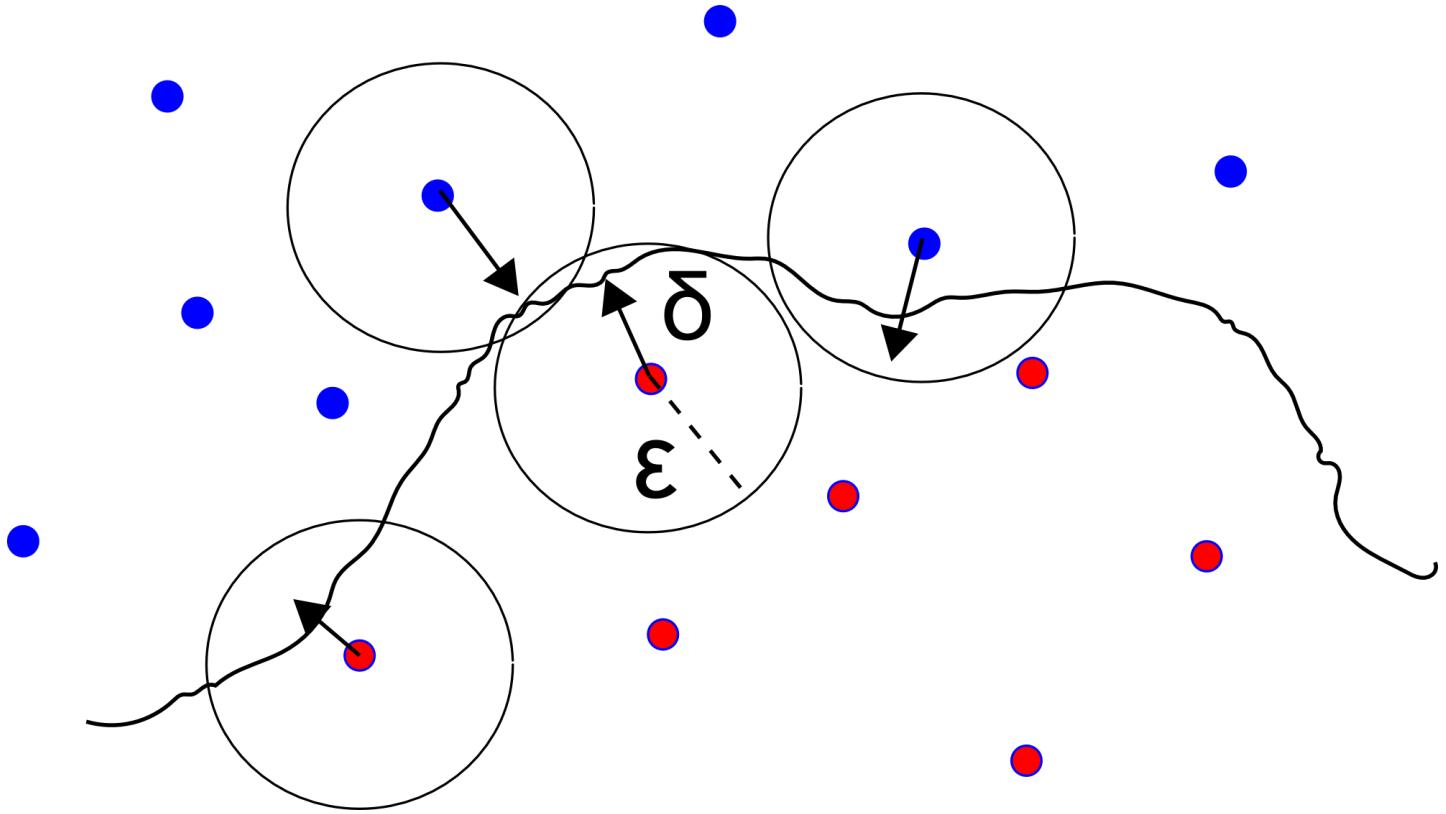
is given by

$$\boldsymbol{x} = \boldsymbol{x}_0 - \eta \cdot \textit{sign}(\boldsymbol{w}). \tag{3.38}$$

How can we "fix" the adversarial weakness? Adversarial training



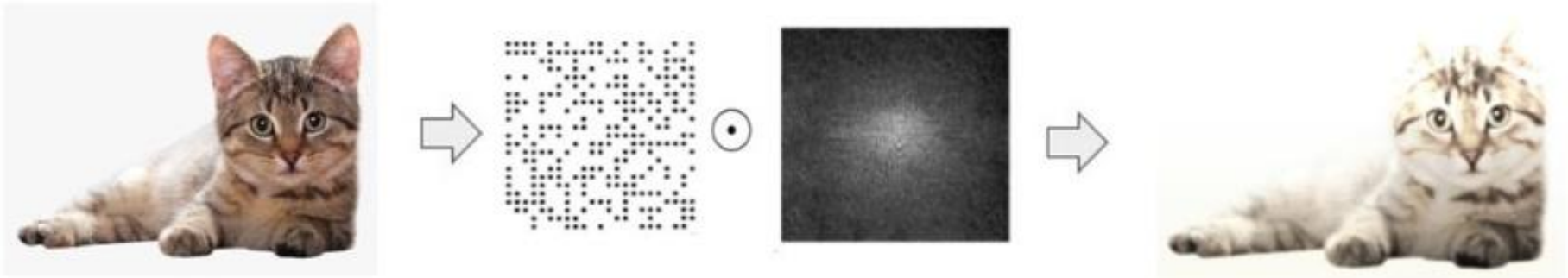
We need to modify the boundaries such that the adv examples are correctly classified



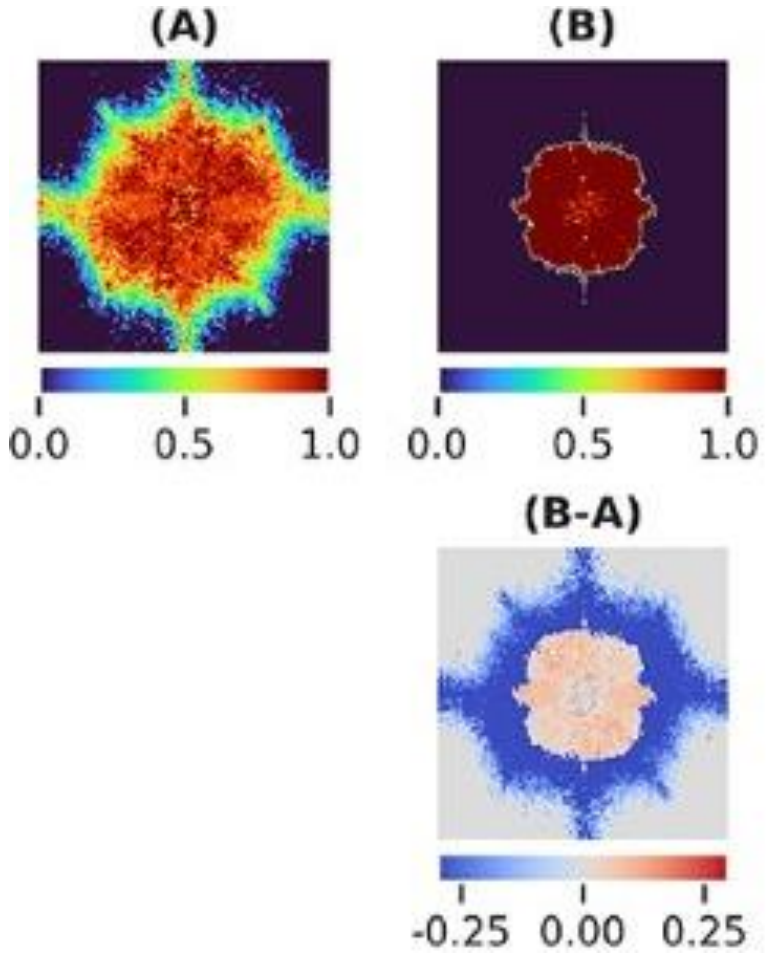
Origin of adversarial weakness and adversarial robustness in Fourier space

Adversarial attacks in Fourier space

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



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



Before and after adversarial training: what is your conclusion in terms of frequency bias of the Adversarial attacks?

Class 6

Interpretability

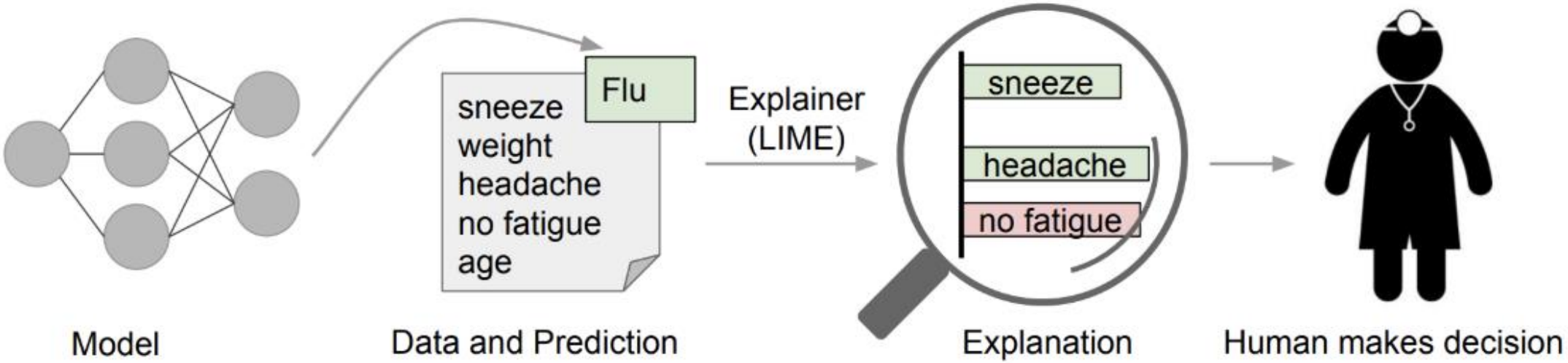
- Problems of annotation artifacts, biases and adversarial attacks

What does it mean for a model to be interpretable?

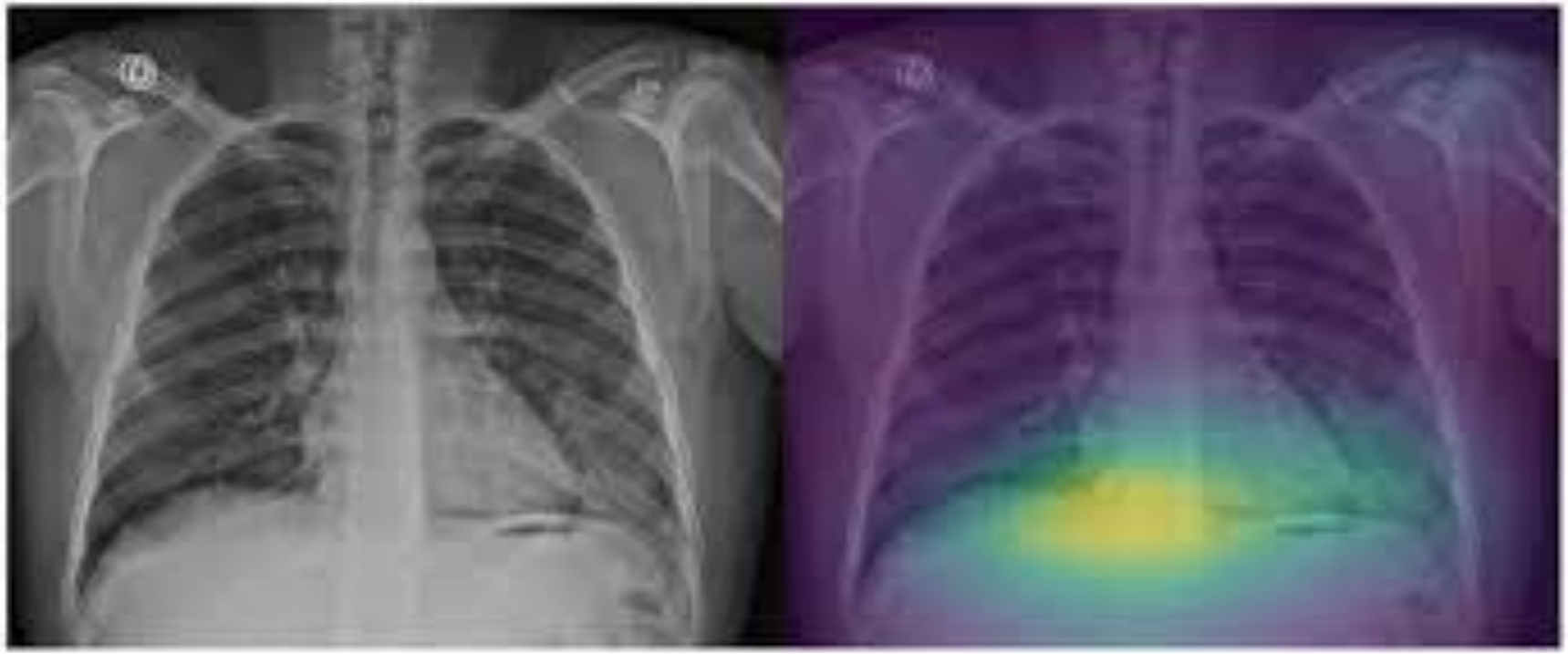
- No mathematical definition of interpretability.
  - *"Interpretability is the degree to which a human can understand the cause of a decision." (*)*
  - *"Interpretability is the degree to which a human can consistently predict the model's result" (**)*
- Interpret on the level of **predictions**
  - Why did the model make certain predictions?
- Interpret on the level of **components**
  - How does the model make predictions?

*Miller, Tim. "Explanation in artificial intelligence: Insights from the social sciences." arXiv Preprint arXiv:1706.07269. (2017)
** Kim, Been, Rajiv Khanna, and Oluwasanmi O. Koyejo. "Examples are not enough, learn to criticize! Criticism for interpretability." Advances in Neural Information Processing Systems (2016)

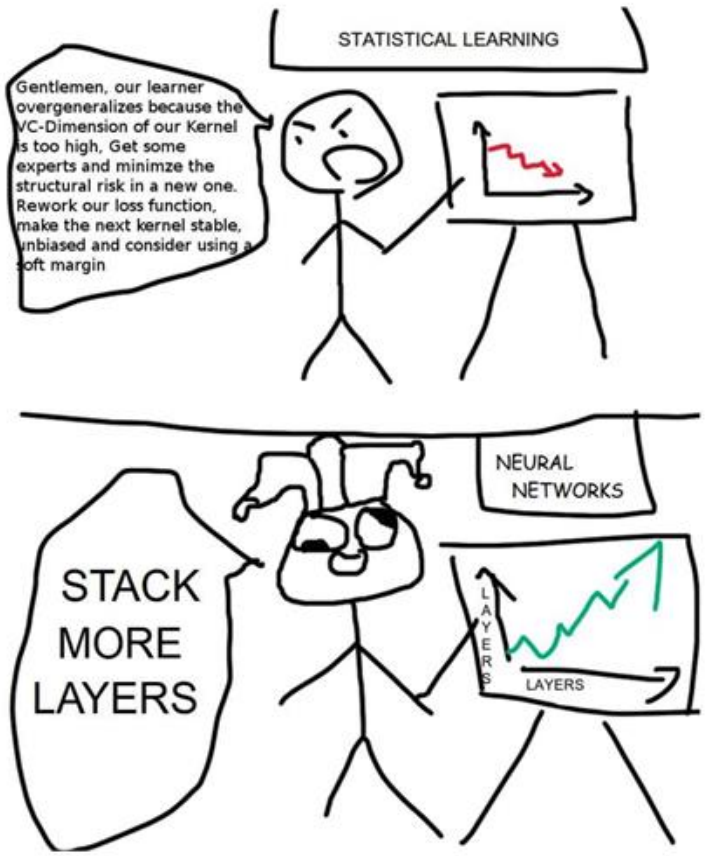
Why interpretability is important?



Why interpretability is important?



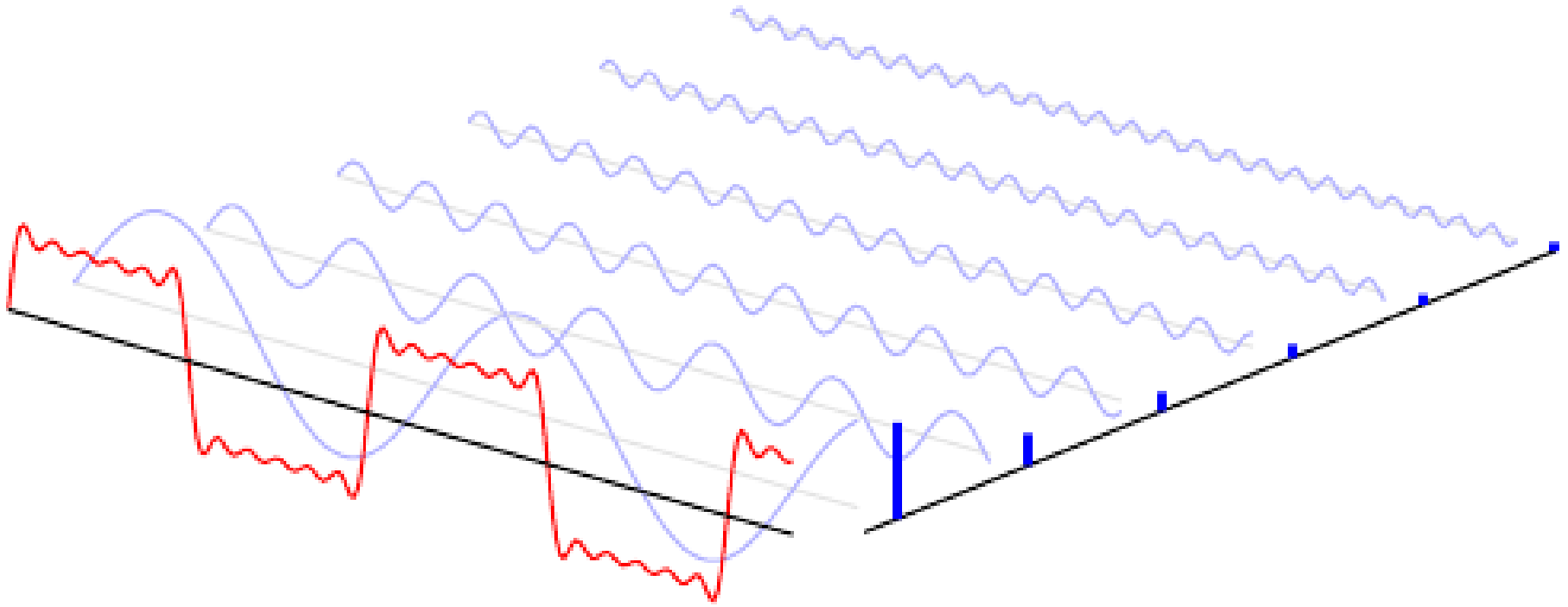
Why interpretability is important?



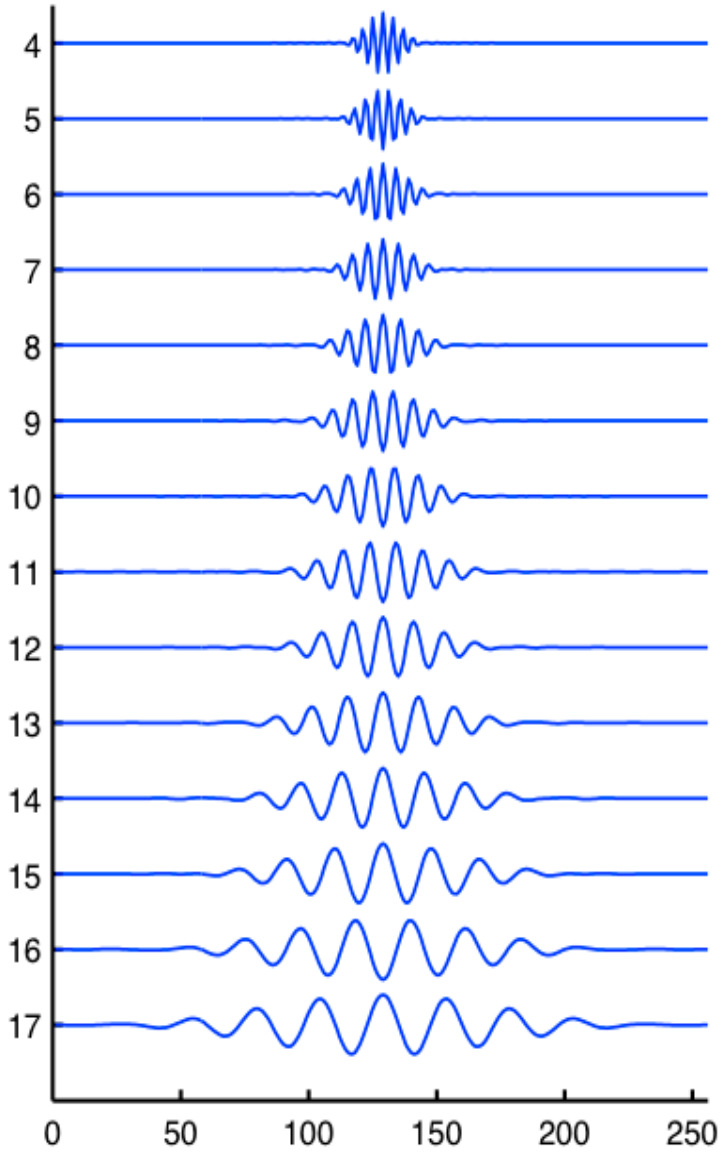
[Reddit; source unknown]

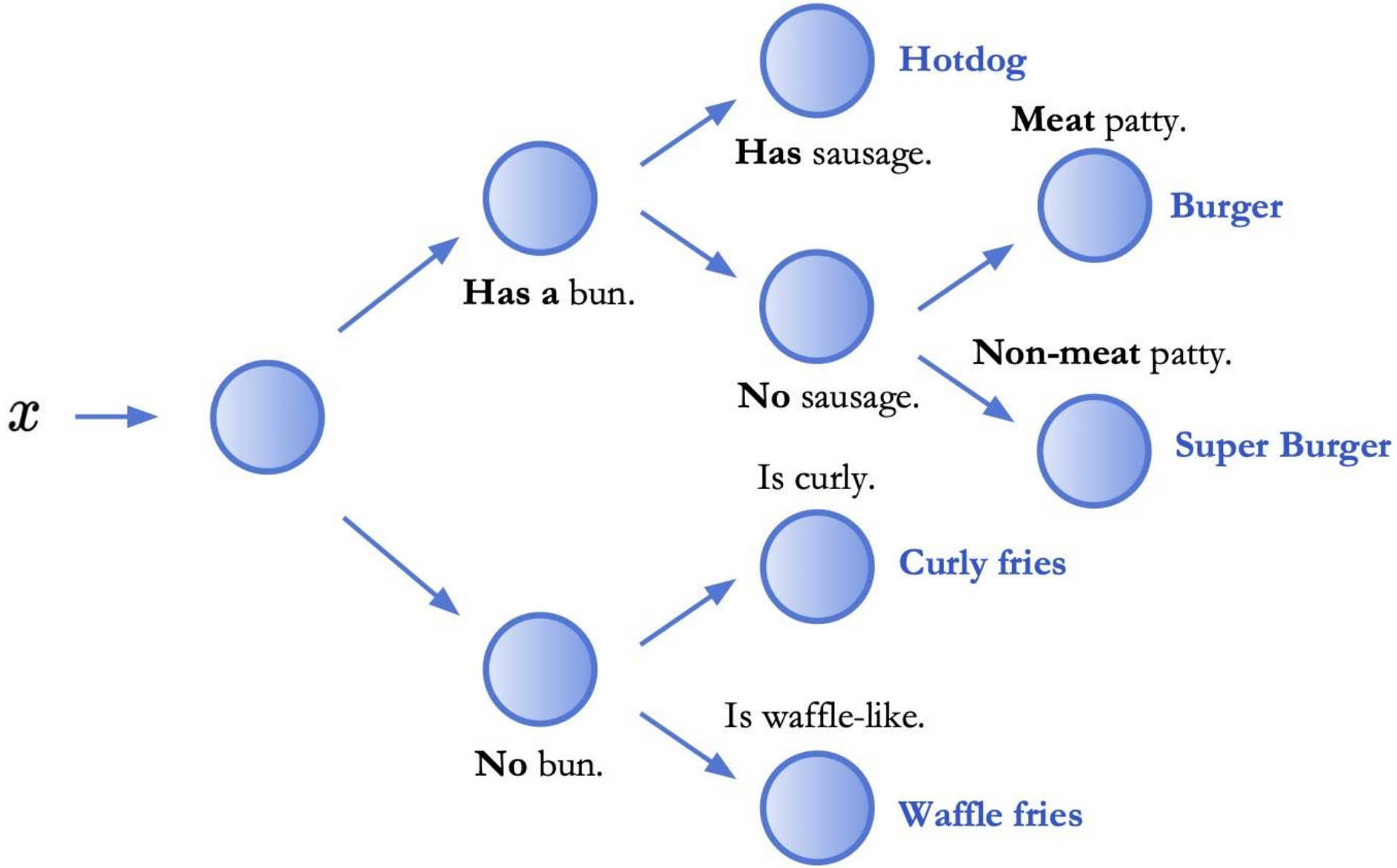
[xkcd.com]

Designed features: Fourier transform



Designed features: Wavelet transform





Can we extract a similar interpretability looking at learned convolutional filters?

Models and Hypothesis space

- Linear separable

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

We learn the separator on data

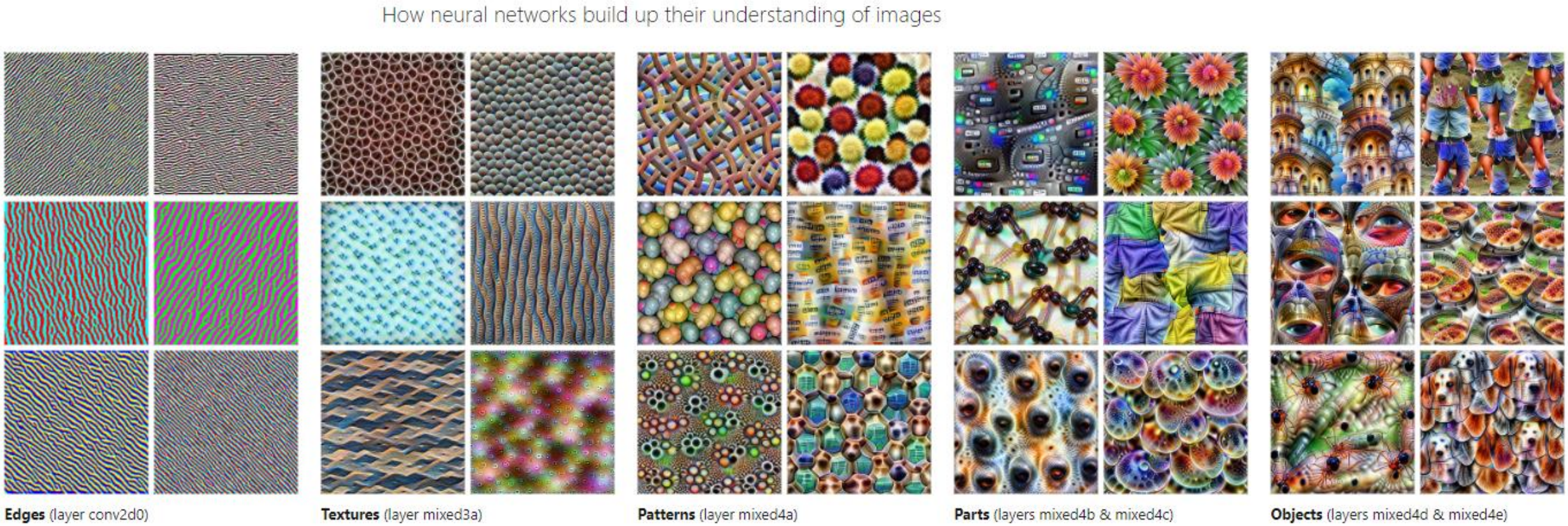
- Non-linear separable

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

We learn the separator on a priori hand-crafted features

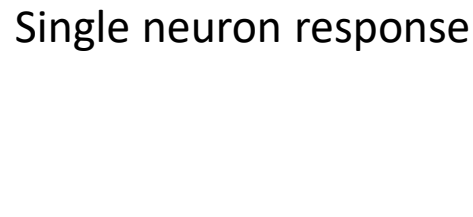
What about Anns?

What images/patterns maximally activate a neuron?



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

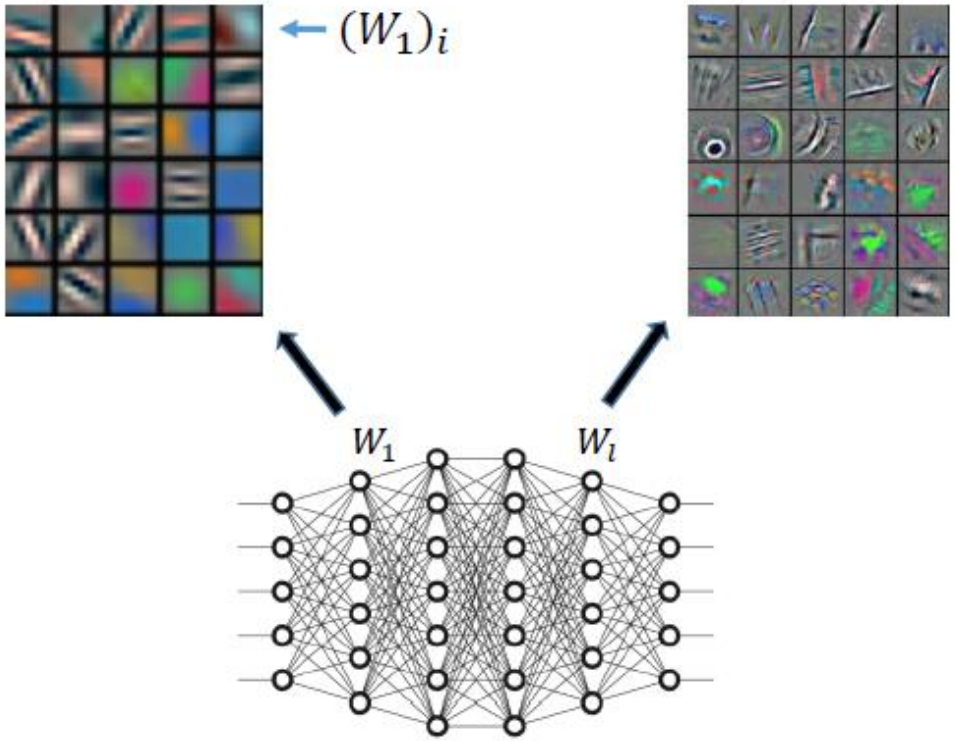
How are those images calculated?

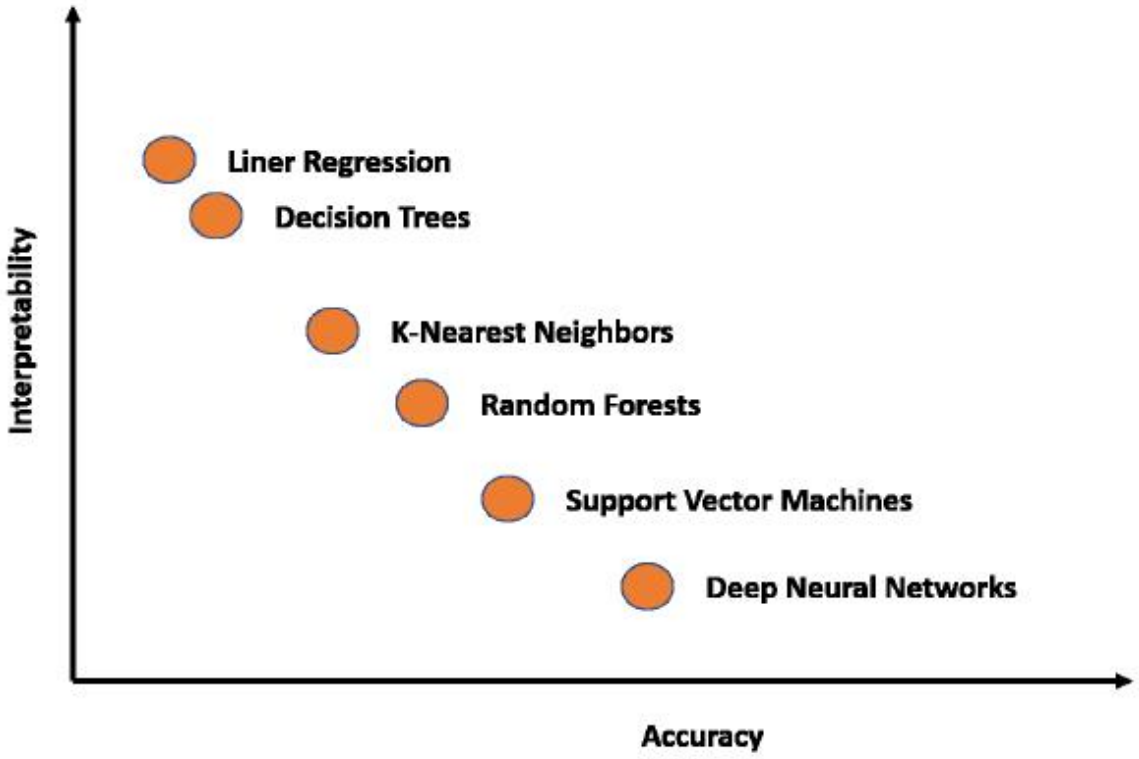


$$\arg\max_{x} f_i(x)$$

$$x_{t+1} = x_t + \eta \nabla_x f_i(x)$$

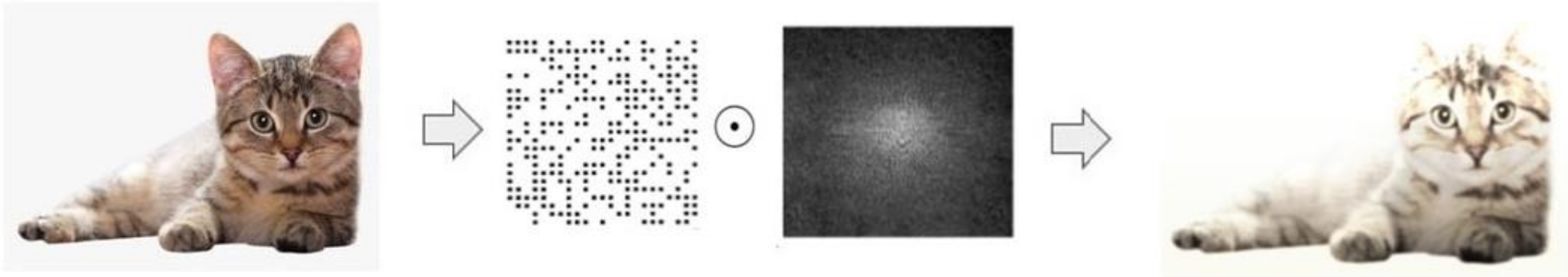
Interpretable features: images





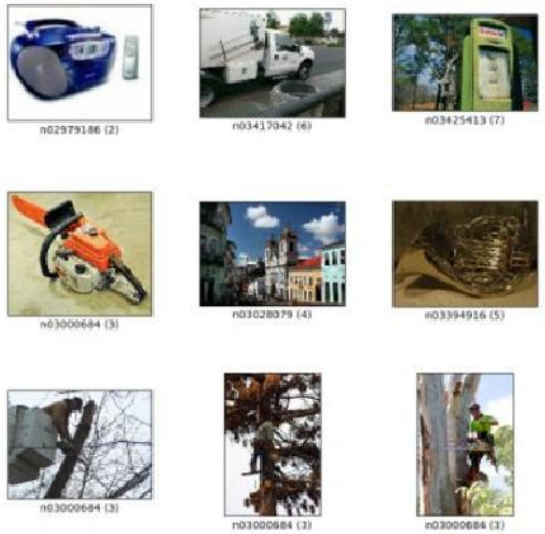
How does a network make a decision? Fourier important features.

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

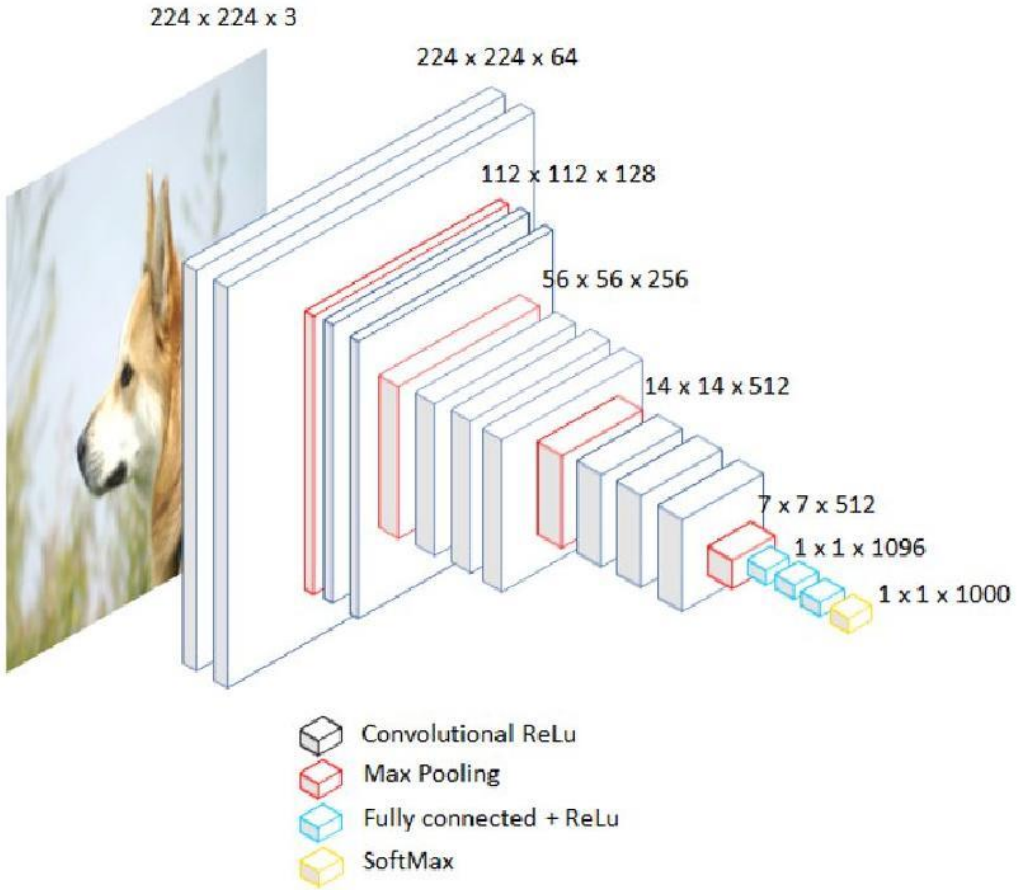


$$M_\Phi(\lambda, p) = \text{argmin}_{M_\Phi} \sum_{x \in \mathcal{X}_V} e^{[\mathcal{L}(\Phi(\bar{x}), y) - \mathcal{L}(\Phi(x), y)]^2} + \lambda \| M_\Phi \|_p, \quad \lambda \in \mathbb{R}_+$$

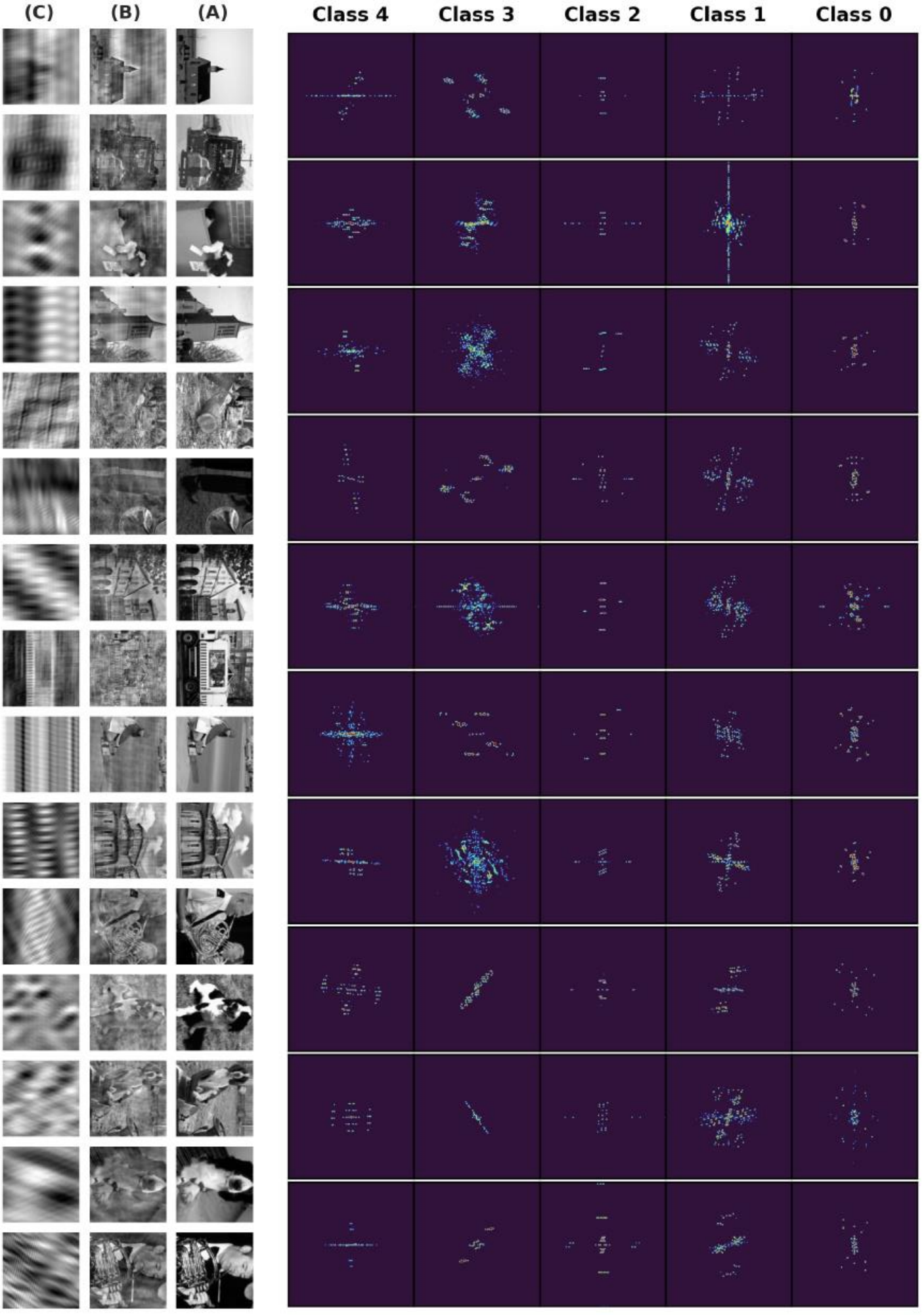
Data and Model: Imagenette and VGG11



```
lbl_dict = dict(
    n01440764='tench',
    n02102040='English springer',
    n02979186='cassette player',
    n03000684='chain saw',
    n03028079='church',
    n03394916='French horn',
    n03417042='garbage truck',
    n03425413='gas pump',
    n03445777='golf ball',
    n03888257='parachute'
)
```



6, 644 image/label pairs: training set. 1, 934 pairs: validation set. For simplicity, we used grayscale versions.


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

What is your conclusion?

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

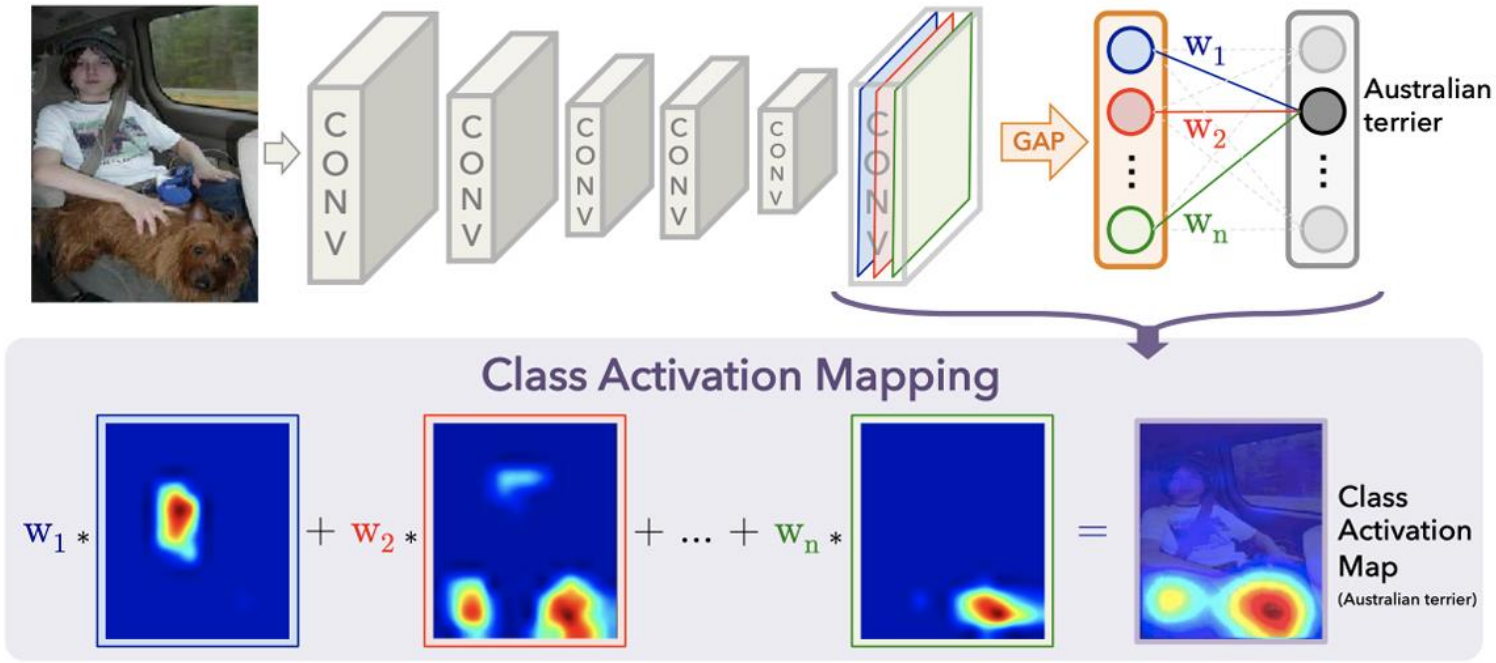
(b) Content image

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

(c) Texture-shape cue conflict

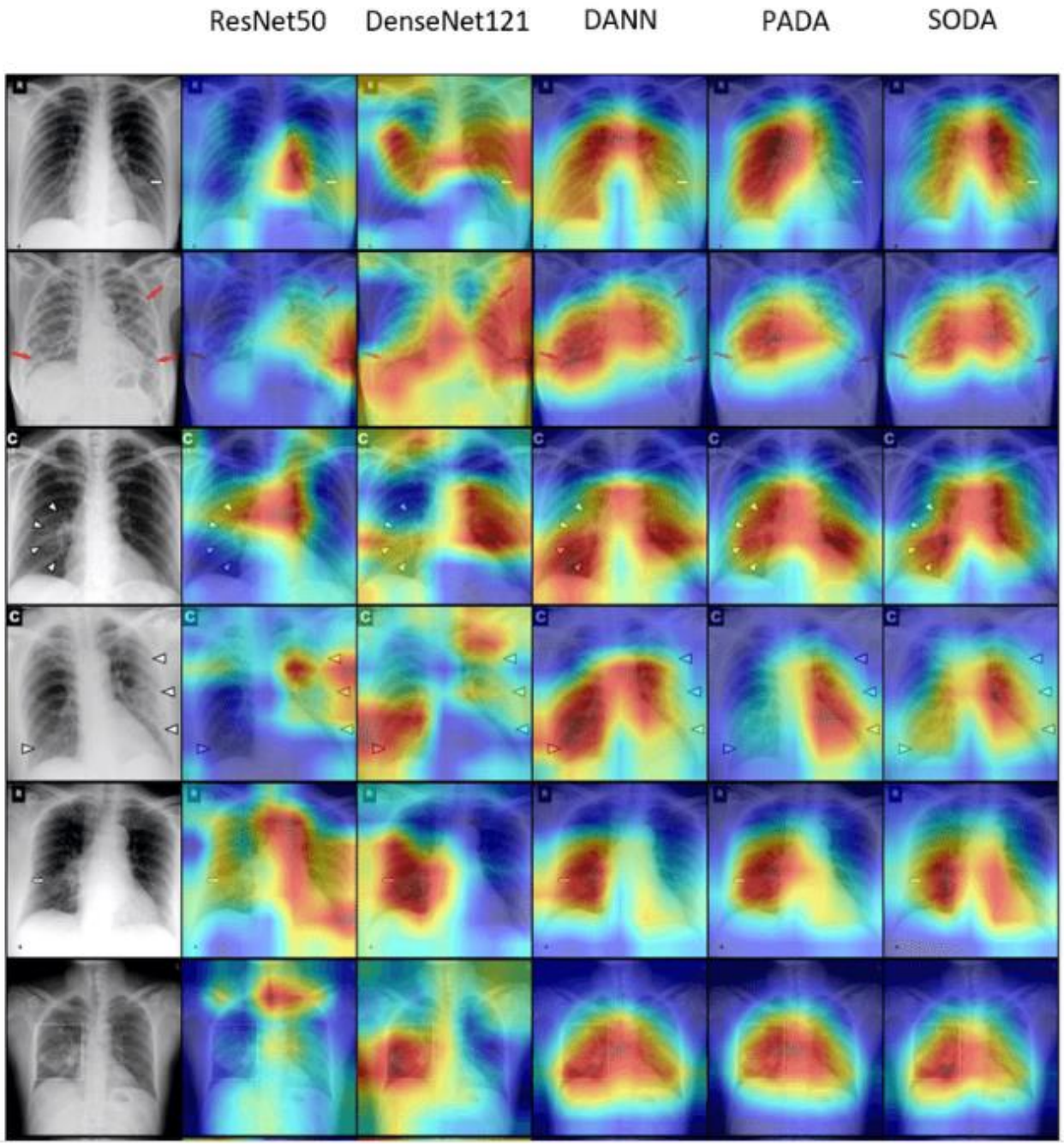
63.9%	**Indian elephant**
26.4%	indri
9.6%	black swan

Where is a ANN looking at? Class activation mapping



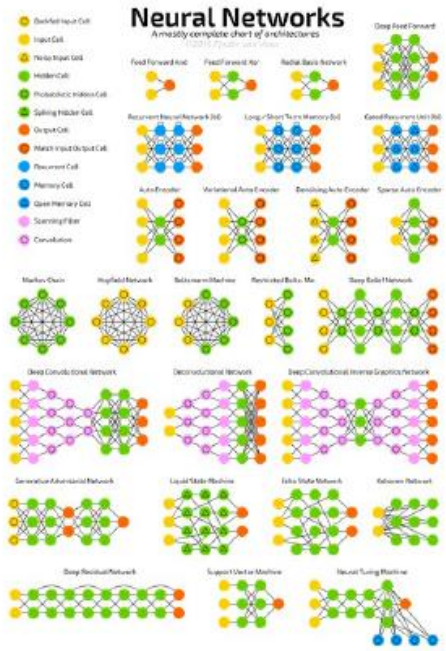
Class activation mapping

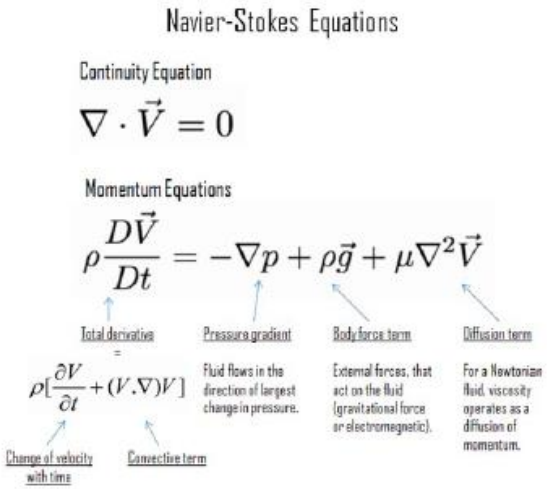
- $f_k(x, y)$: Activation of unit k in spatial location (x, y)
- $F^k = \sum_{x,y} f_k(x, y)$: Result of global average pooling
- $S_c = \sum_k w_k^c F_k$: input to Softmax layer for class c
- $M_c(x, y) = \sum_k w_k^c f_k(x, y)$: CAM for class c



What is your conclusion?

Interpretable operations



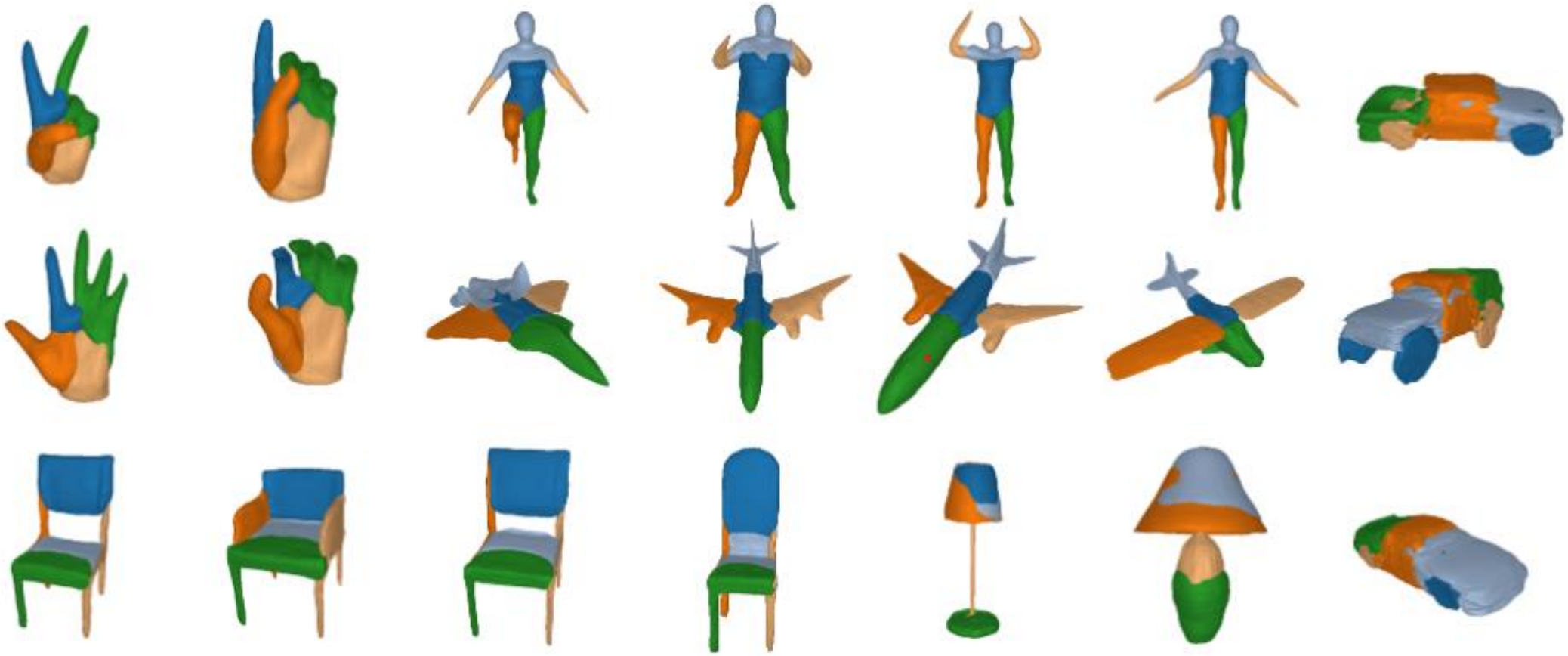


Suppose we learnt a φ_W from input/output to a physical system: $y_i = \varphi_W(x_i)$

Approximate φ_W with a dictionary of known operators:

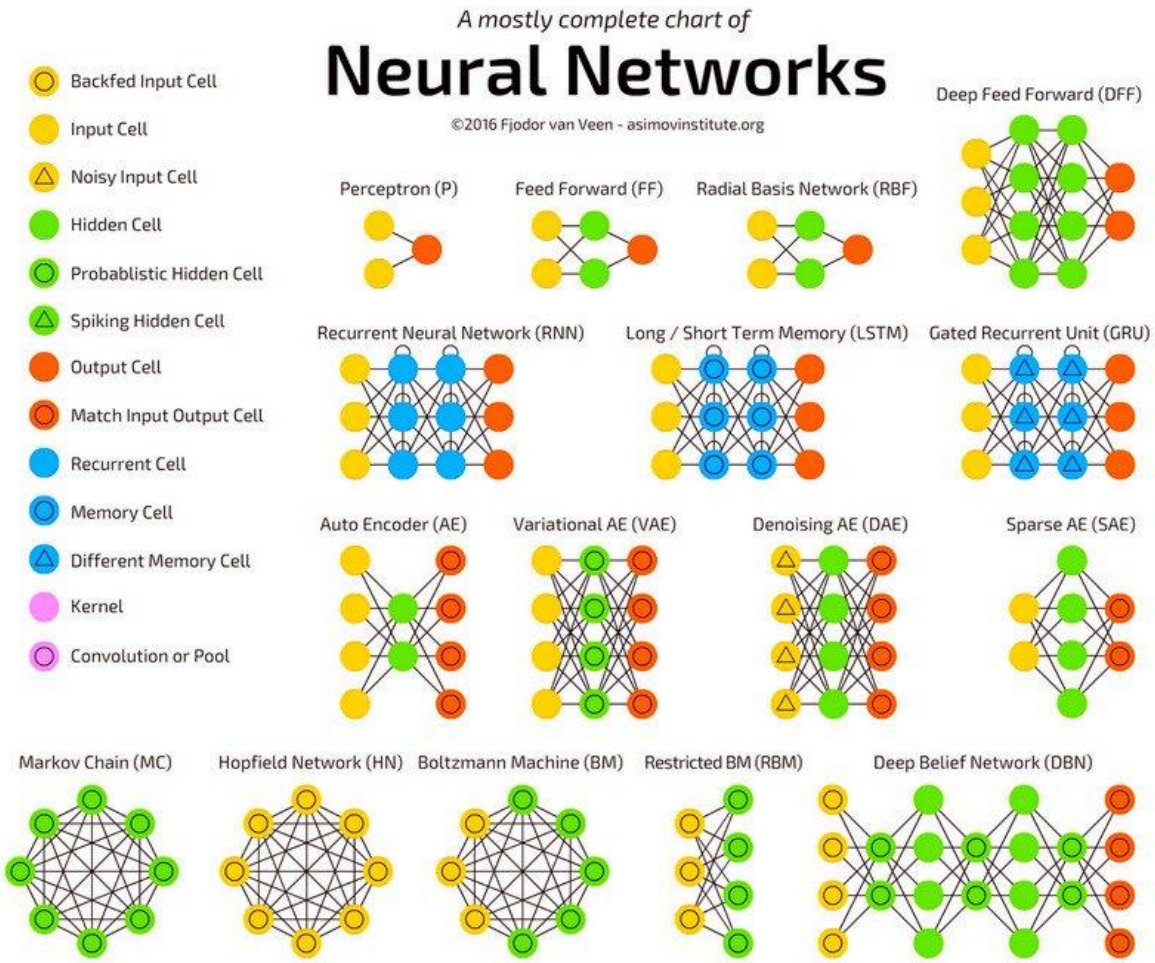
$$\arg\min_{\alpha} \sum_{\mathbf{i}} \|\varphi_{\mathbf{W}}(\mathbf{x_i}) - (\{\partial_{\mathbf{t}}, \partial_{\mathbf{x}}, ...\}\alpha)(\mathbf{x_i})\|_{\mathbf{2}} + \|\alpha\|_{\mathbf{0}}$$

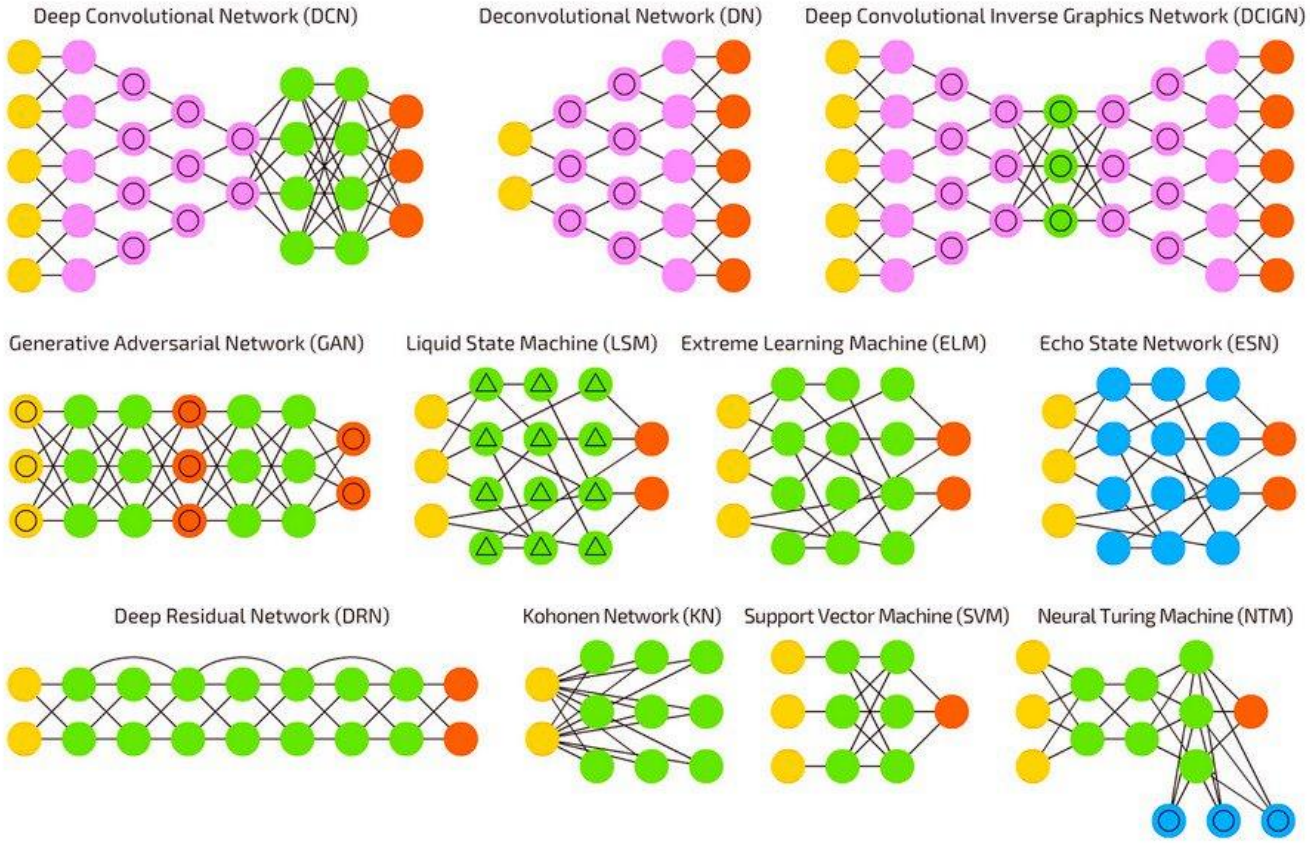
How do we understand a visual scene? Interpretable parts



Other architectures and beyond images

A zoo of networks

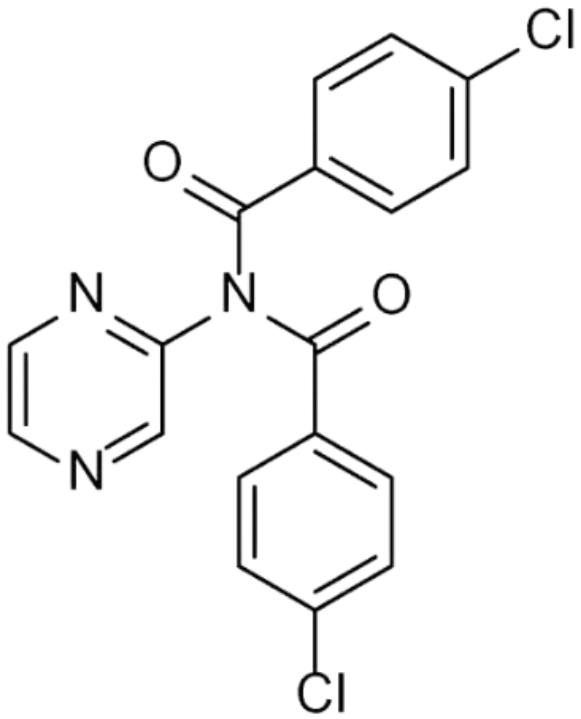


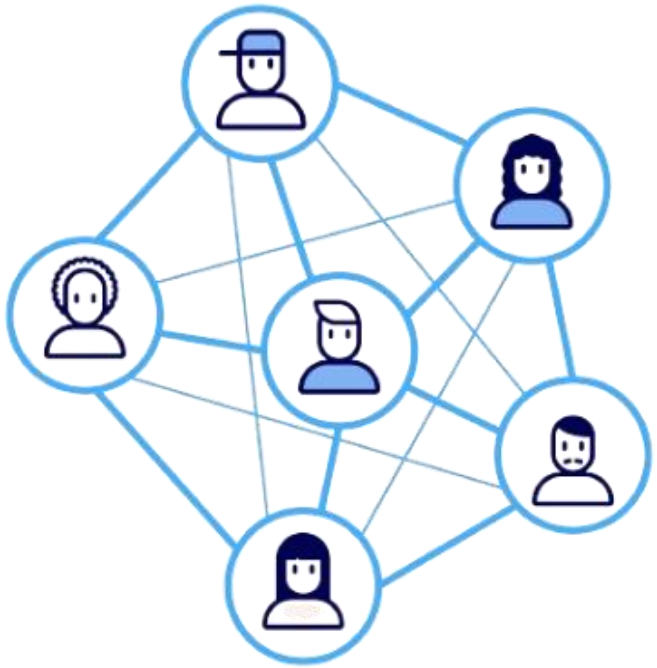


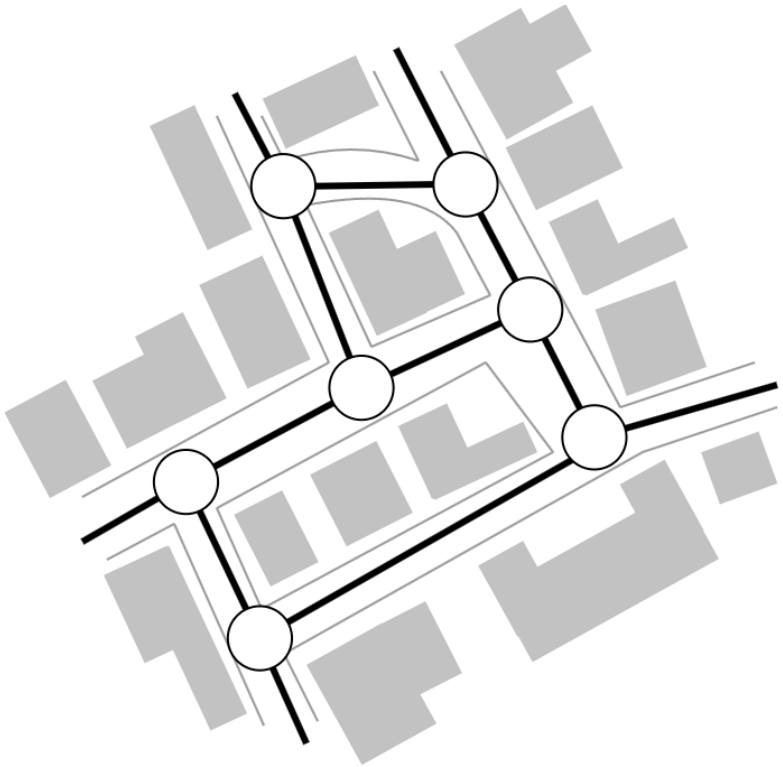
https://modelzoo.co/

Short intro to graph neural networks

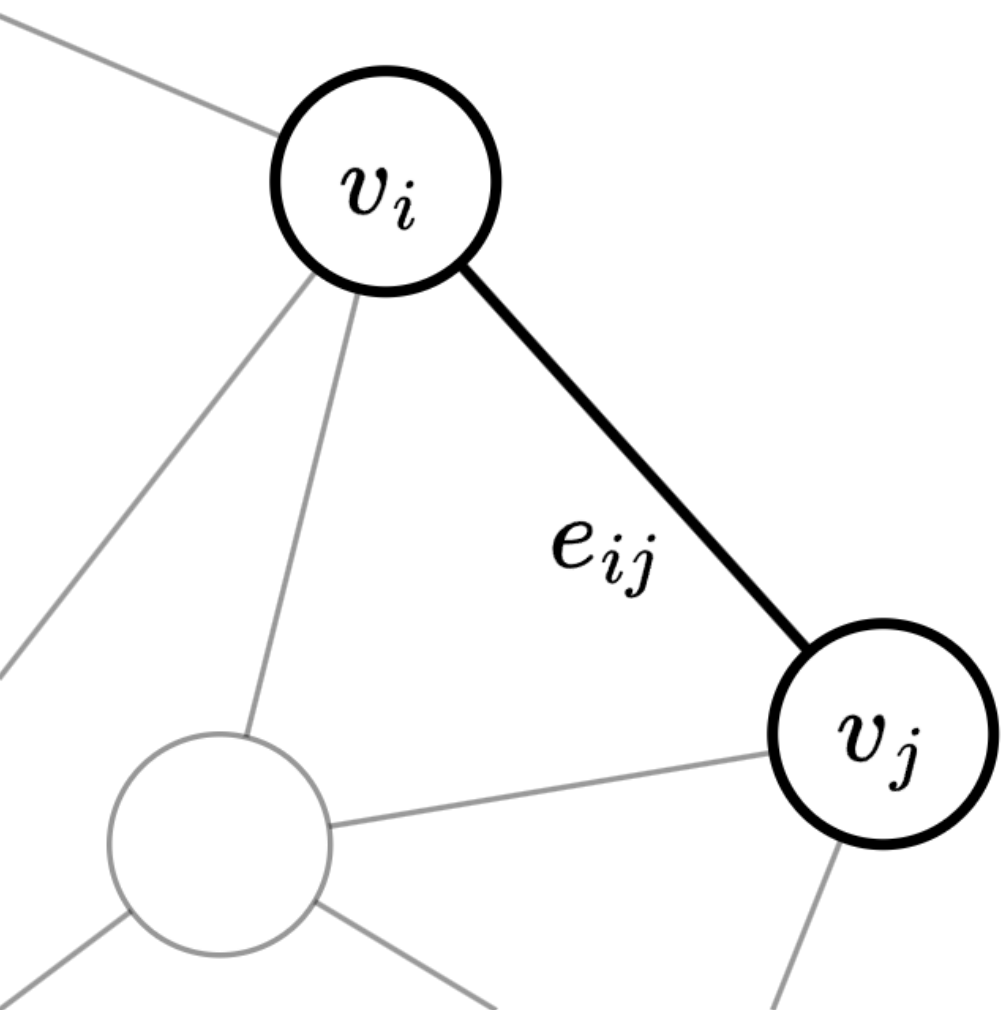
Graph structures in the real world





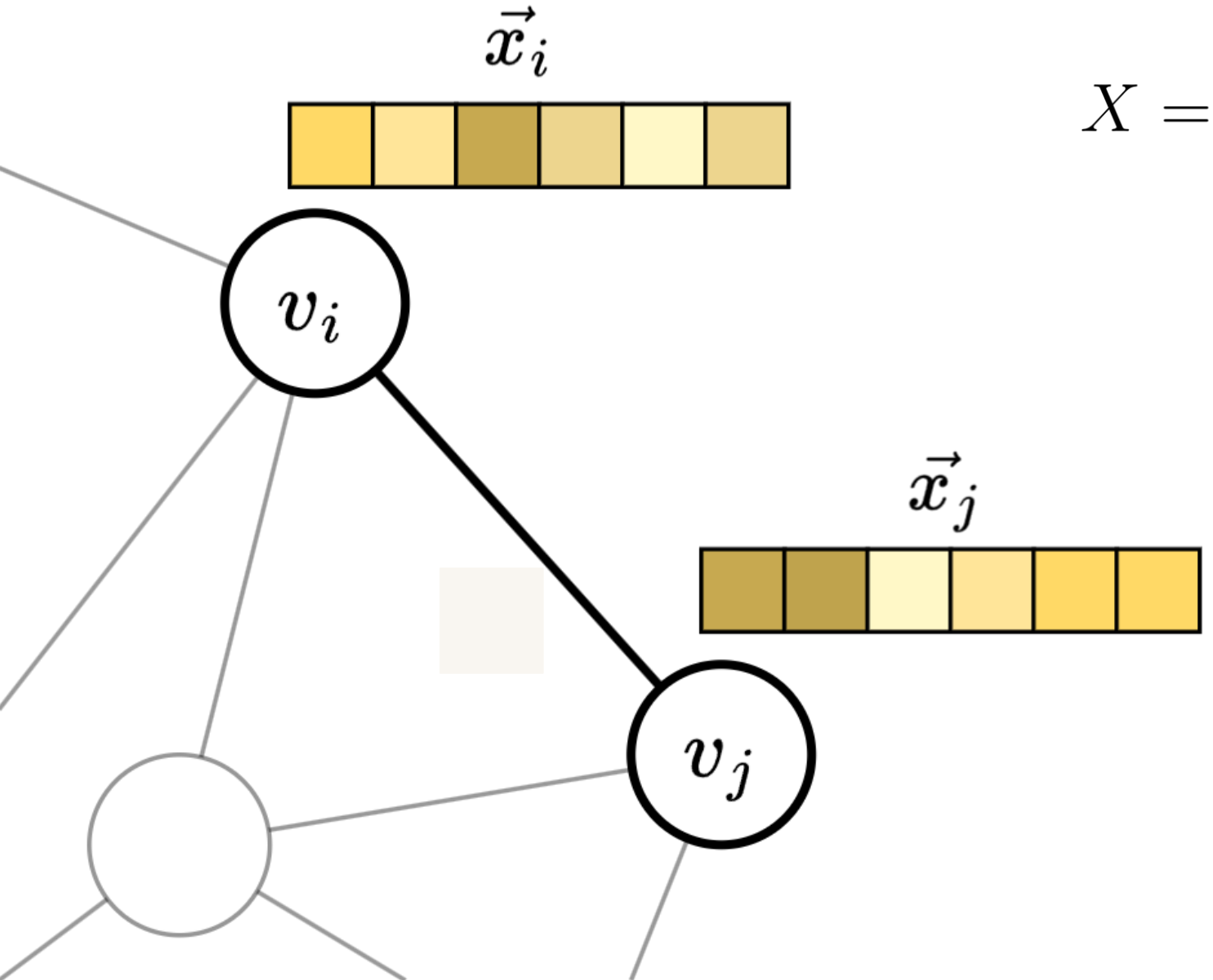


$$\mathcal{G} = (V, E)$$



Adjacency matrix of $\mathcal{G}$

$$A_{ij} = \begin{cases} 0 & \text{if } (i,j) \notin E \\ 1 & \text{if } (i,j) \in E \end{cases}$$



$$X = \begin{pmatrix} \vec{x_1} \\ \vdots \\ \vec{x_i} \\ \vdots \\ \vec{x_n} \end{pmatrix}$$

node features of $\mathcal{G}$

for example:

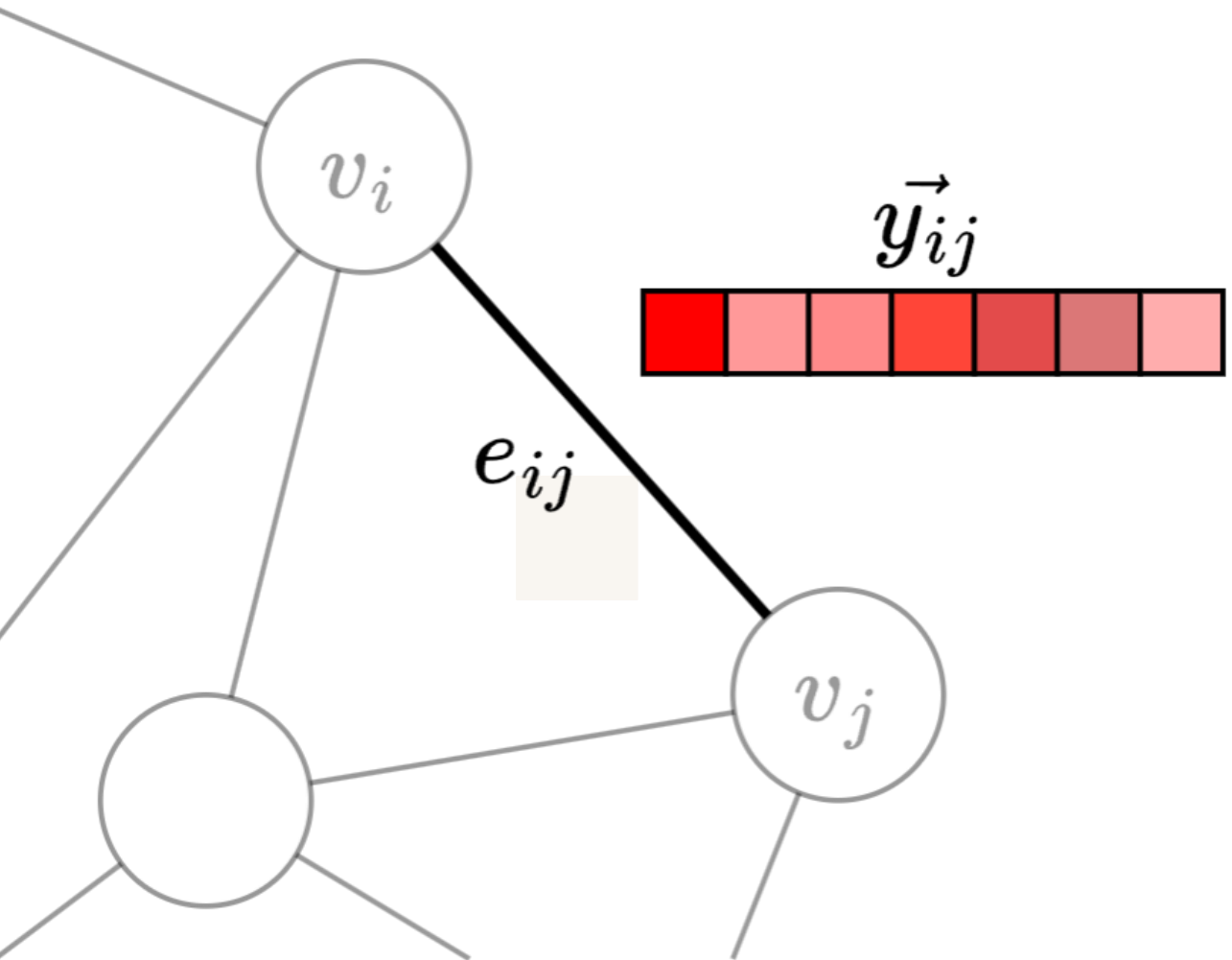
- properties of the atom
- data of the account
- number/type of crossroads

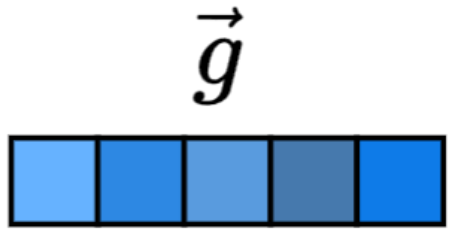
$$Y_{ij} = \vec{y_{ij}}$$

edge features of $\mathcal{G}$

for example:

- type of bond
- date of friend request, relationship
- average traffic, number of car parks





global features of $\mathcal{G}$

for example:

- known properties
- year
- day of the week

Tasks on graphs

Global tasks

tasks on the **whole graph.**
Both classification and regression.

Examples:

- predicting properties of a molecule
- predicting the hobby shared by all the people
- predicting the likelihood of a car accident

Tasks on graphs

Node tasks

tasks at the level of the **single node.**
Both classification and regression.

Examples:

- predicting if an atom is part of an aromatic ring
- predicting political stance of each users on a topic
- classifying intersections based on predicted traffic

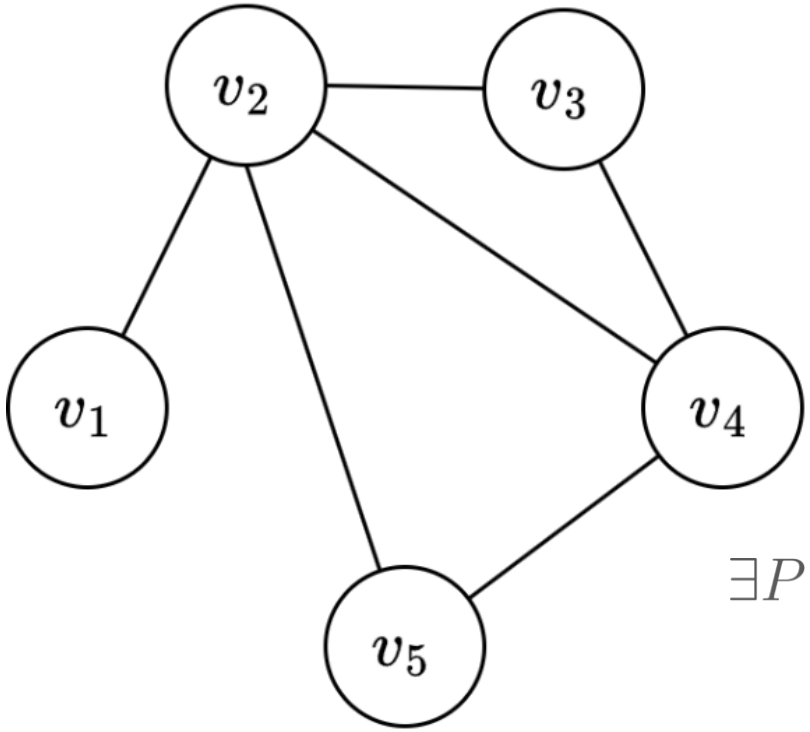
Tasks on graphs

Edge tasks

tasks at the level of the **single edge.**
Both classification and regression.

Examples:

- predicting which bond is easier to break
- predicting how likely two users will become friends
- predicting how busy a road will be in an hour

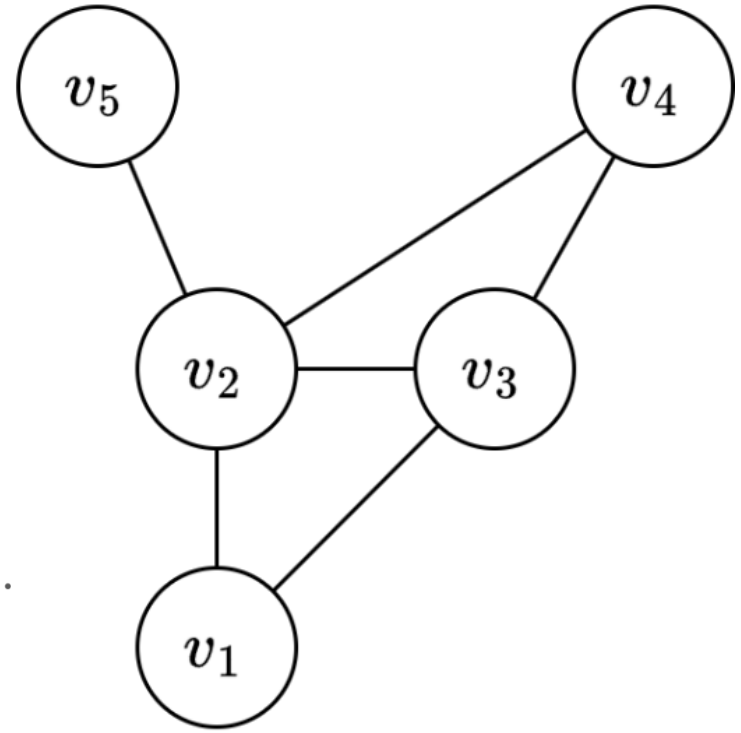


$\cong$

$\iff$

$\exists P$ permutation matrix s.t.

$$PA_1P^{-1} = A_2$$



$$A_1 = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \end{bmatrix}$$

$$A_2 = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

$\forall P$ permutation matrix

invariance in the context of global tasks

$$F_G(A, X, Y, \vec{g}) = F_G(PAP^{-1}, PX, PYP^{-1}, \vec{g}) \in \mathbb{R} \text{ or } \in \mathbb{R}^d$$

equivariance in the context of node or edge tasks

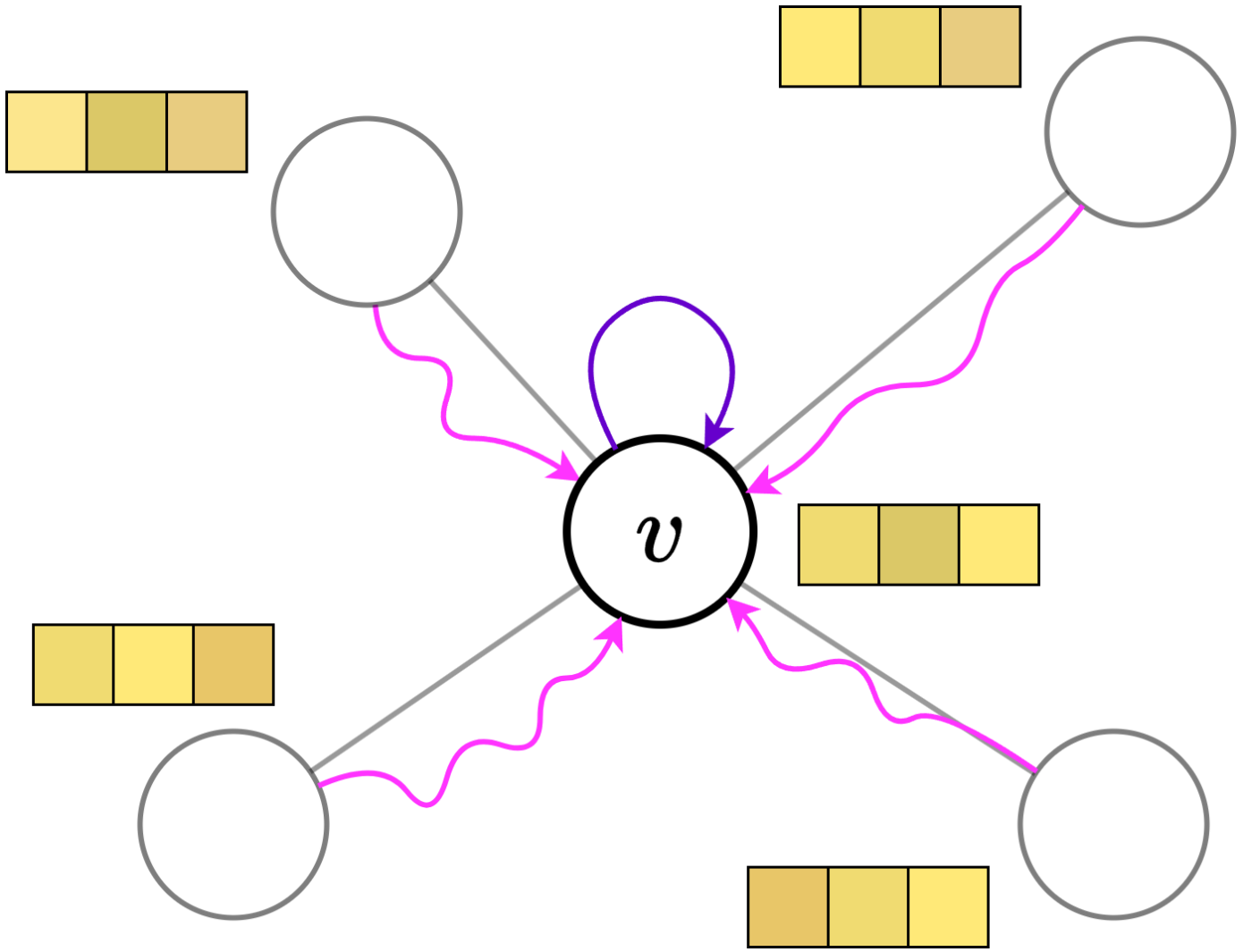
$$F_V(A, X, Y, \vec{g}) = P^{-1} F_V(PAP^{-1}, PX, PYP^{-1}, \vec{g})$$
$$\in \mathbb{R}^n \text{ or } \in \mathbb{R}^{n \cdot d}$$

$$F_E(A, X, Y, \vec{g}) = P^{-1} F_E(PAP^{-1}, PX, PYP^{-1}, \vec{g}) P$$
$$\in \mathcal{M}_{\mathbb{R}}(n, n) \text{ or } \in \mathcal{M}_{\mathbb{R}^d}(n, n)$$

in the context of node/global tasks

input: A , X

- **Aggregating** over $\mathcal{N}_v$
- **Updating** $\vec{x_v}$



Convolutional approach

$$\vec{x_v} = \phi\left(\vec{x_v}, \bigoplus_{u \in \mathcal{N}_v} c_{vu}\psi(\vec{x_u})\right)$$

Aggregation

must be a **permutation invariant** operator

- mean
- max
- sum

$$\vec{x_v} = \phi\left(\vec{x_v}, \bigoplus_{u \in \mathcal{N}_v} c_{vu}\psi(\vec{x_u})\right)$$

Update

$$\phi(\vec{x}, \vec{z}) = \sigma(W\vec{x} + U\vec{z} + \vec{b})$$

learnable

coefficient

in the case of summation can be

$$c_{uv} = \frac{1}{\sqrt{deg(u) \cdot deg(v)}}$$

transformation of the features

$$\psi(\vec{x}) = W\vec{x} + \vec{b}$$

learnable

Attentional approach

$$\vec{x_v} = \phi\left(\vec{x_v}, \bigoplus_{u \in \mathcal{N}_v} a(\vec{x_v}, \vec{x_u})\psi(\vec{x_u})\right)$$

coefficients are **learned**
via NN and softmax
normalized

$$a(\vec{x_v}, \vec{x_u}) \in \mathbb{R}$$

Message-passing approach

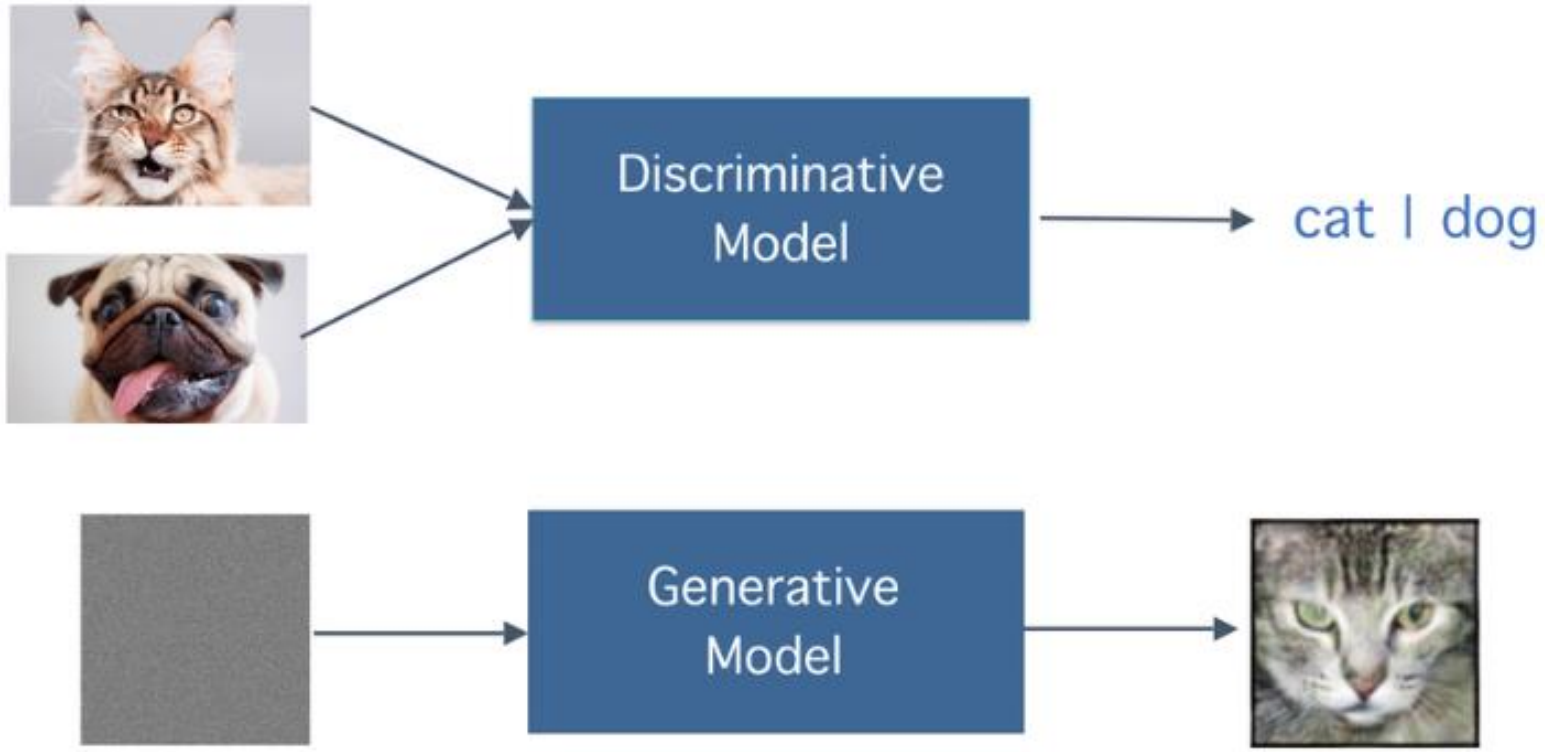
$$\vec{x_v} = \phi\left(\vec{x_v}, \bigoplus_{u \in \mathcal{N}_v} \psi(\vec{x_v}, \vec{x_u})\right)$$

ψhere is a **learnable message function**

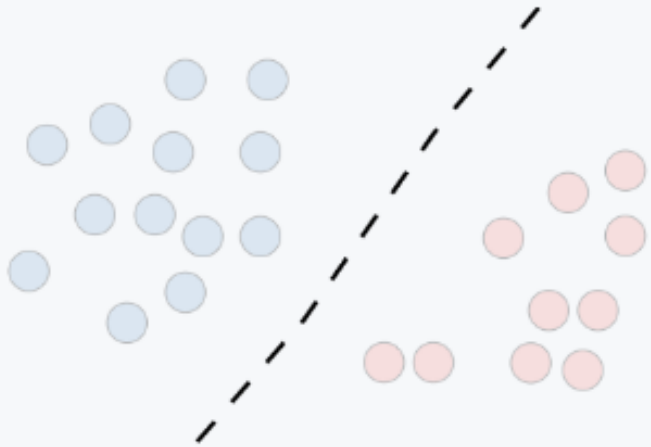
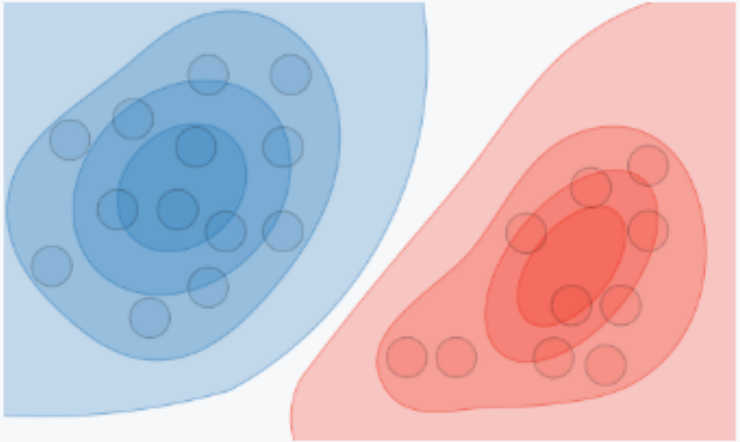
$\psi(\vec{x_v}, \vec{x_u}) \in \mathbb{R}^m$

Short intro to generative models

Generative vs discriminative model



Generative vs discriminative model

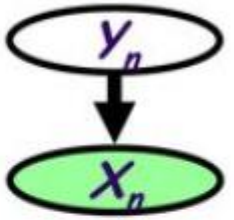
	Discriminative model	Generative model
Goal	Directly estimate $P(y\|x)$	Estimate $P(x\|y)$ to then deduce $P(y\|x)$
What's learned	Decision boundary	Probability distributions of the data
Illustration		

Some examples of discriminative models?

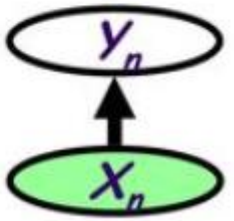
Classifiers

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

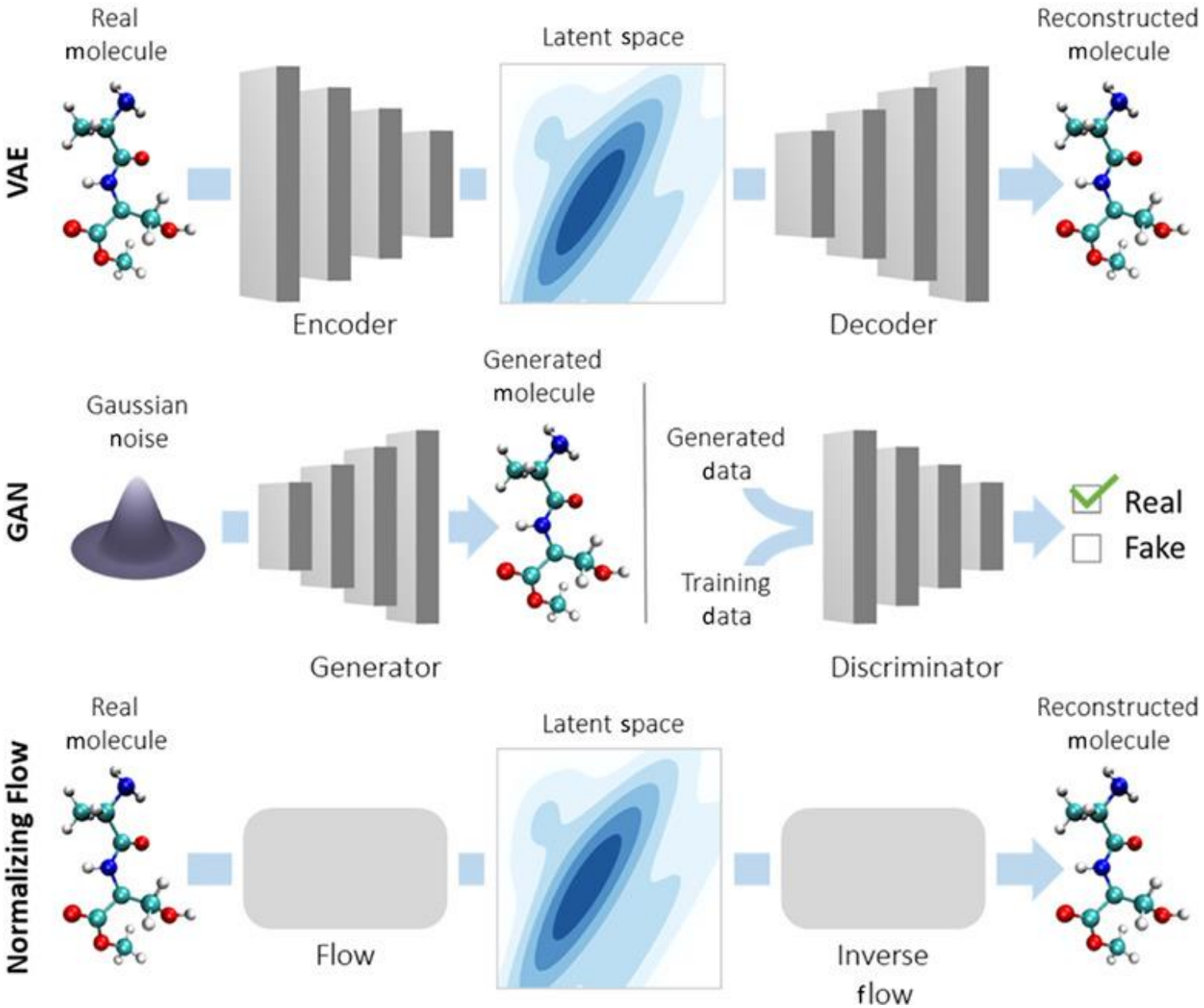
- Generative classifiers (e.g., Naïve Bayes):
  - Assume some functional form for P(X|Y), P(Y)
    This is a '***generative***' model of the data!
  - Estimate parameters of P(X|Y), P(Y) directly from training data
  - Use Bayes rule to calculate P(Y|X= x)



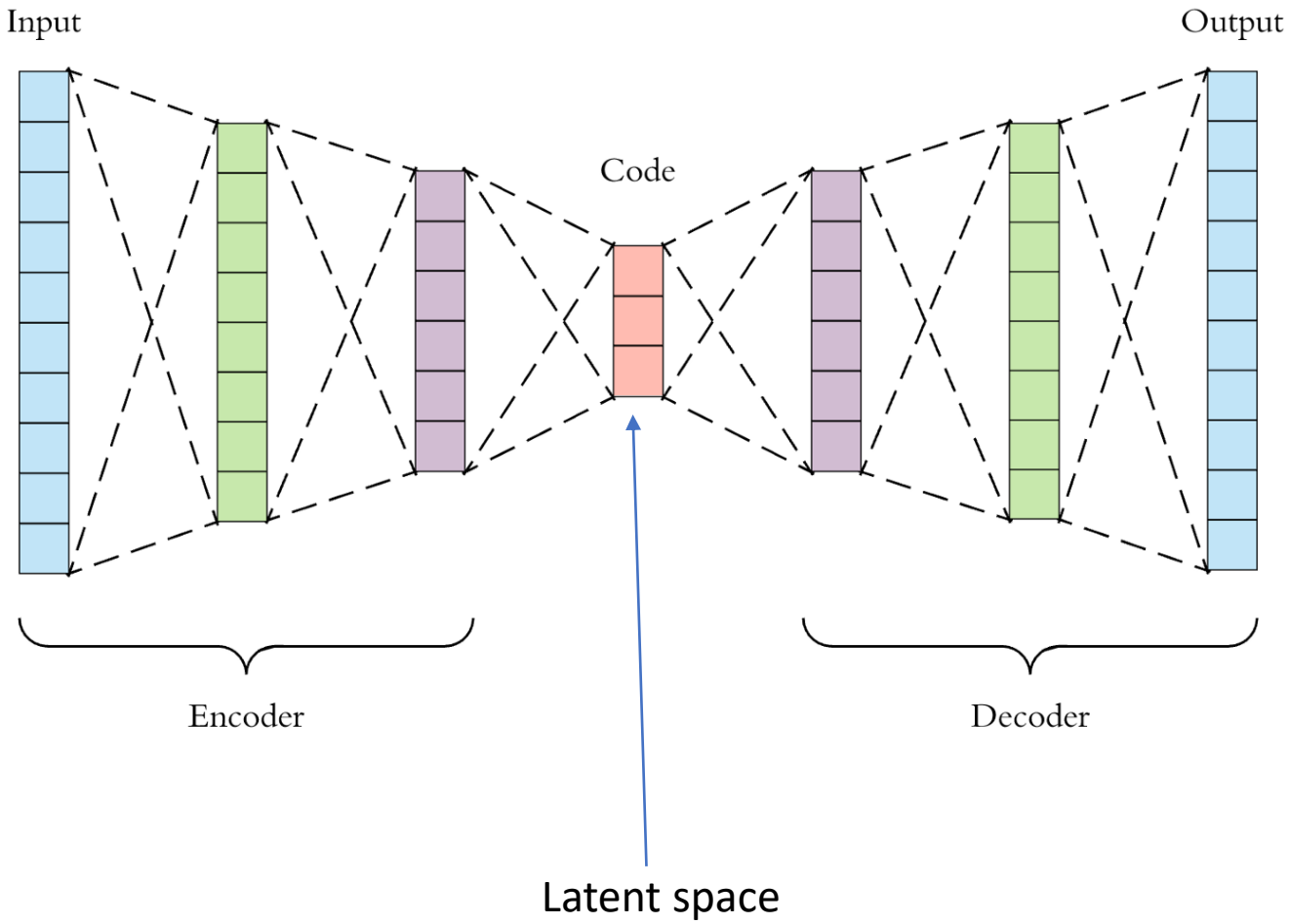
- Discriminative classifiers (e.g., logistic regression)
  - Directly assume some functional form for P(Y|X)
    This is a '***discriminative***' model of the data!
  - Estimate parameters of P(Y|X) directly from training data



Types of generative models

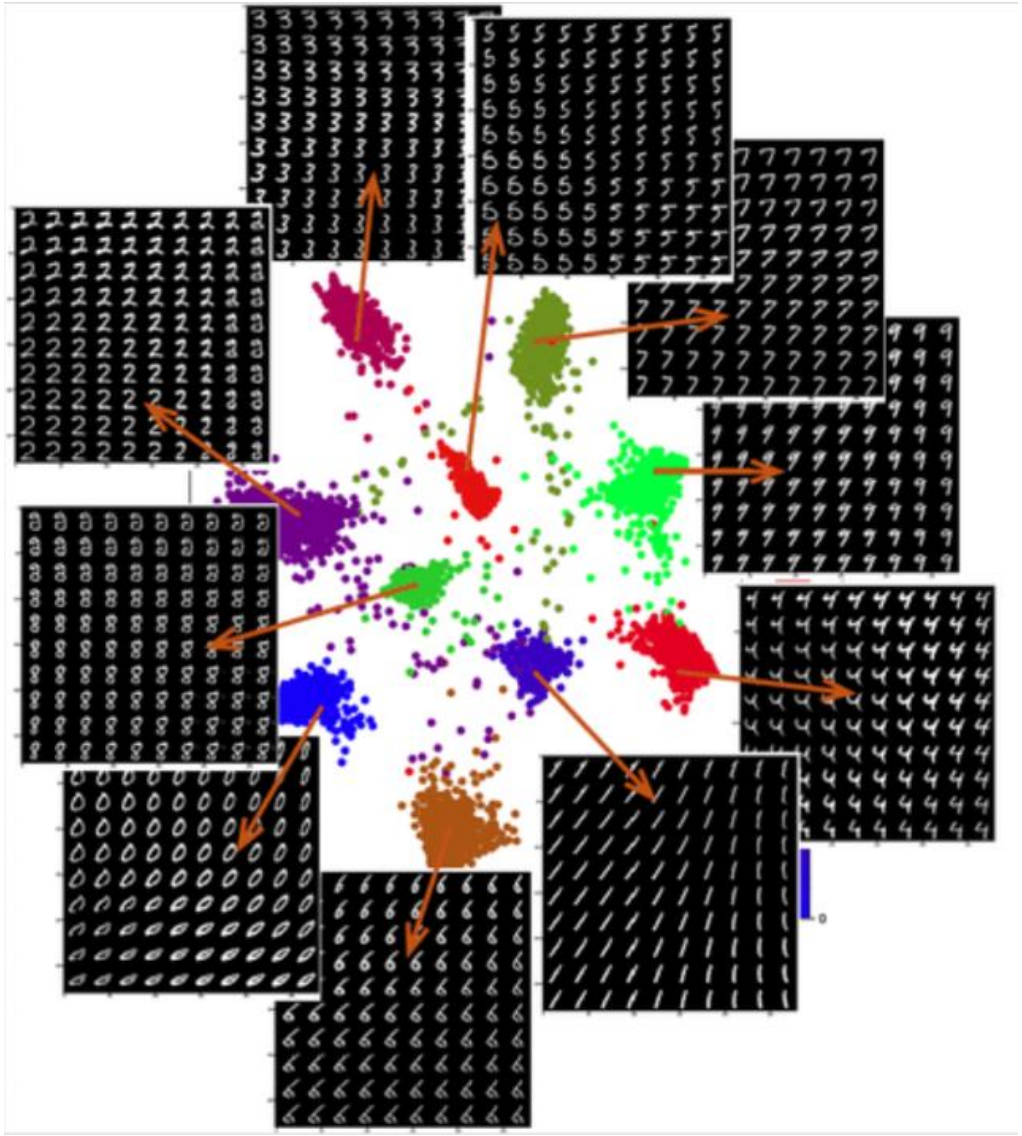


Autoencoders

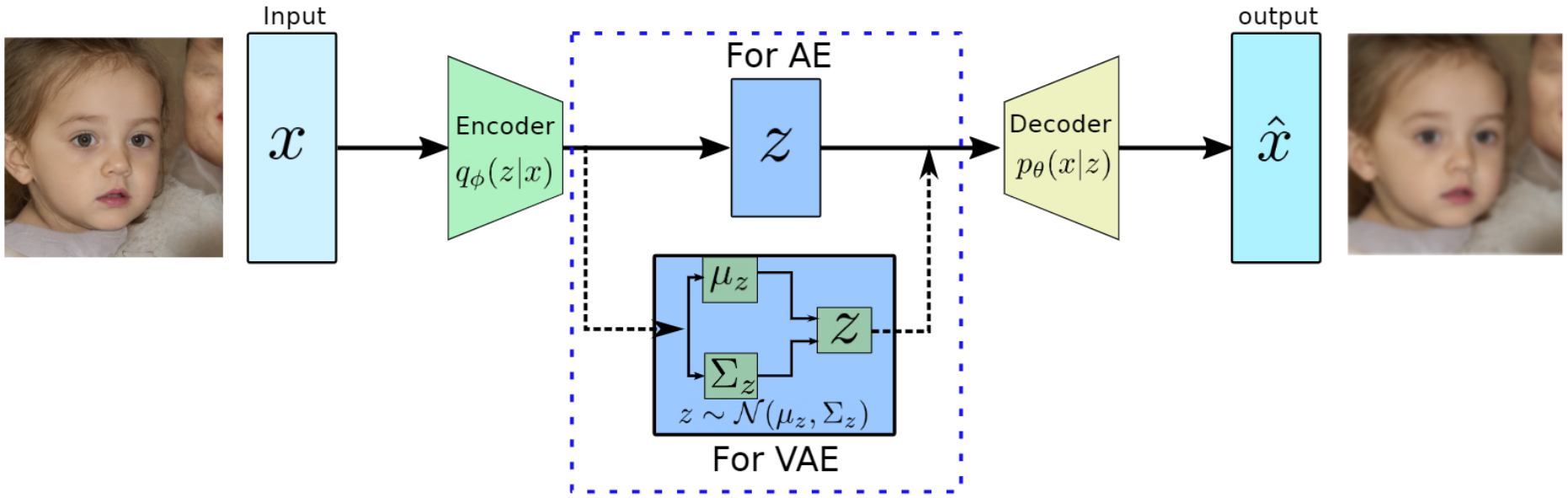


How would you write the loss of an autoencoder?
How would you use an autoencoder as a generative model?

Latent space for MNIST

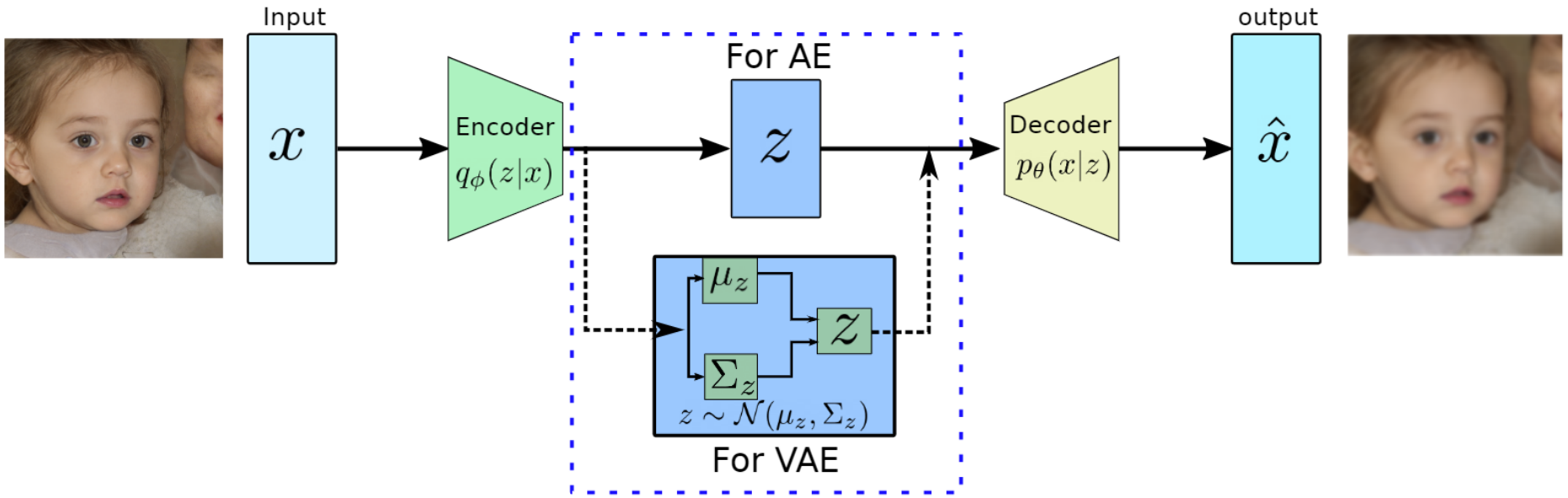


Variational autoencoder



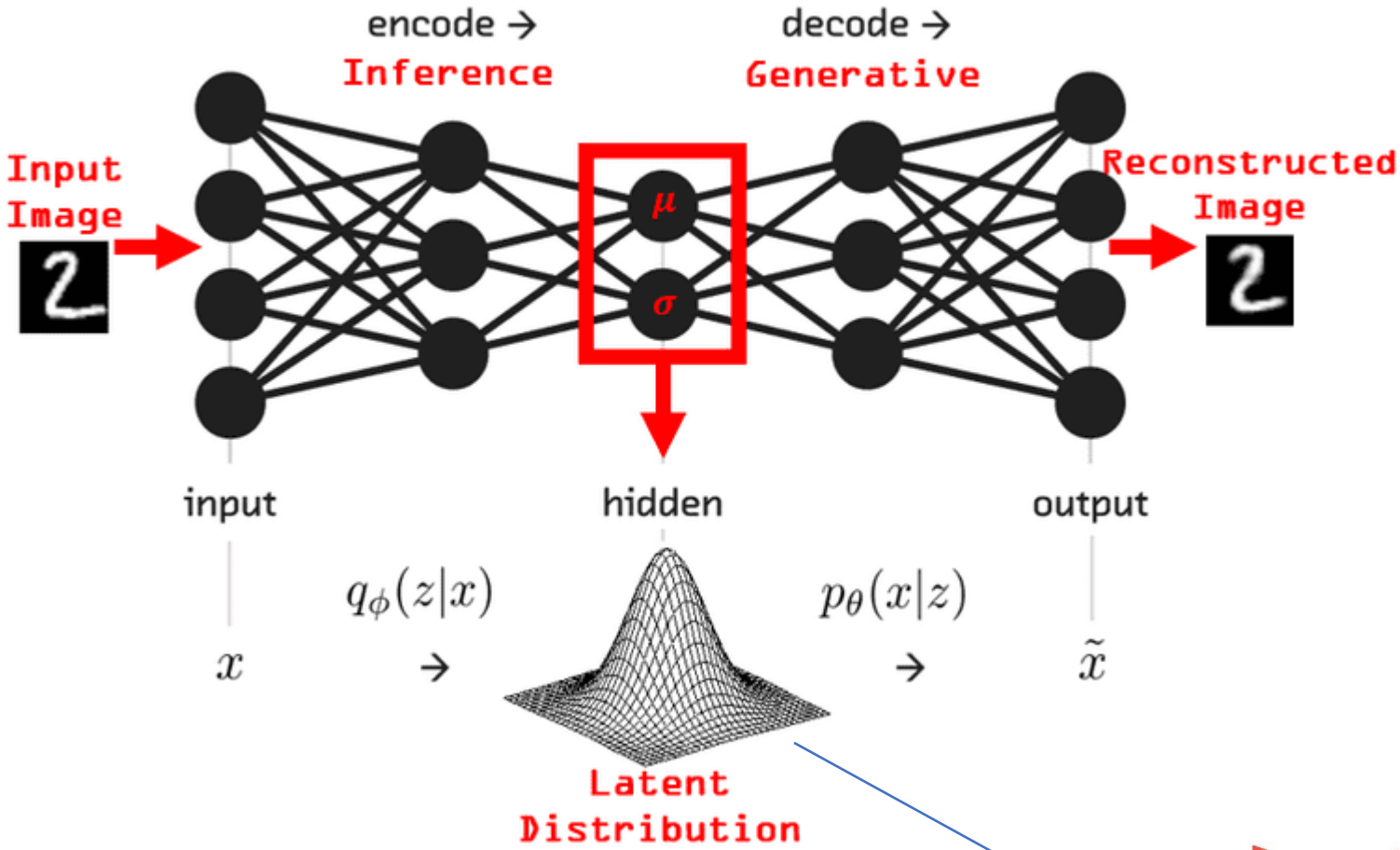
Instead of outputting the vectors in the latent space, the encoder of VAE outputs parameters of a pre-defined distribution in the latent space for every input. The VAE then imposes a constraint on this latent distribution forcing it to be a normal distribution.

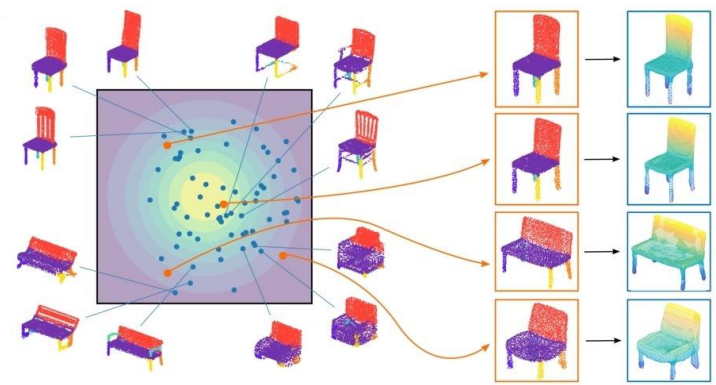
Variational autoencoder



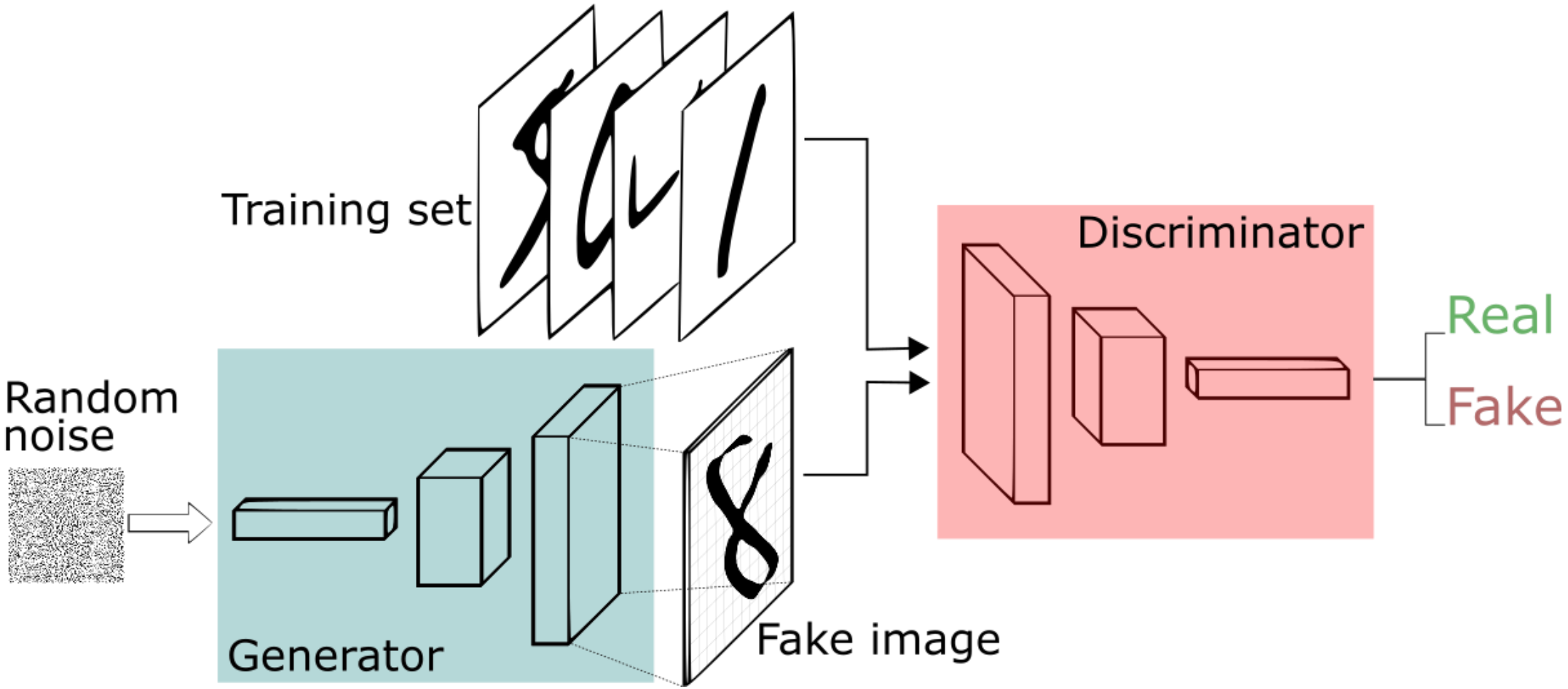
The latent vector is sampled from the encoder-generated distribution before feeding it to the decoder. This random sampling makes it difficult for backpropagation to happen for the encoder since we can't trace back errors due to this random sampling. Hence we use a reparameterization trick to model the 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



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, $\hat{y} = 0$

https://poloclub.github.io/ganlab/

https://pianalytix.com/6-eye-catching-applications-of-gans/

Can we use NNs to process text?

Representing words as discrete symbols

In traditional NLP, we regard words as discrete symbols:
hotel, conference, motel — a localist representation

one 1, the rest 0's

Words can be represented by one-hot vectors:

hotel = [0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]
motel = [0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0]

Vector dimension = number of words in vocabulary (e.g., 500,000)

Challenge: How to compute similarity of two words?

Representing words by their context

Distributional hypothesis: words that occur in similar contexts tend to have similar meanings

J.R.Firth 1957

- "You shall know a word by the company it keeps"
- One of the most successful ideas of modern statistical NLP!

…government debt problems turning into	***banking***	*crises as happened in 2009…*
…saying that Europe needs unified	***banking***	*regulation to replace the hodgepodge…*
…India has just given its	***banking***	*system a shot in the arm…*

These context words will represent *banking*.

Distributional hypothesis

"tejuino"

C1: A bottle of ____ is on the table.

C2: Everybody likes ____.

C3: Don't have ____ before you drive.

C4: We make ____ out of corn.

Distributional hypothesis

C1: A bottle of ____ is on the table.

C2: Everybody likes ____.

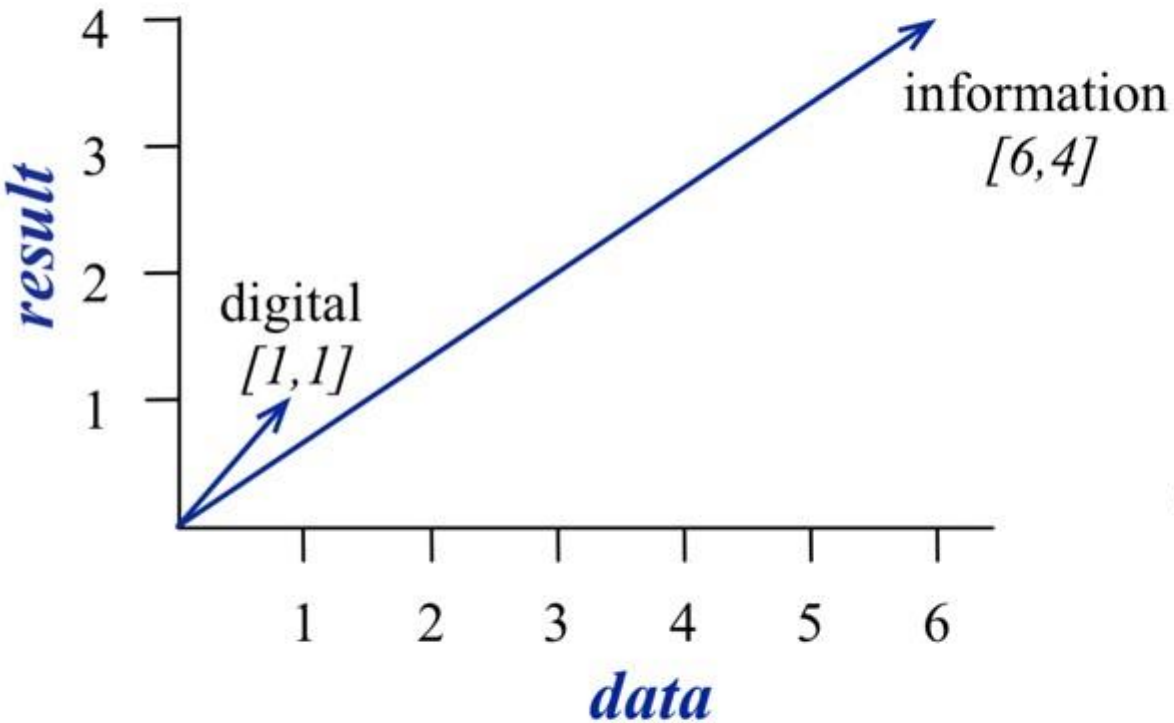
C3: Don't have ____ before you drive.

C4: We make ____ out of corn.

	C1	C2	C3	C4
tejuino	1	1	1	1
loud	0	0	0	0
motor-oil	1	0	0	0
tortillas	0	1	0	1
choices	0	1	0	0
wine	1	1	1	0

"words that occur in similar contexts tend to have similar meanings"

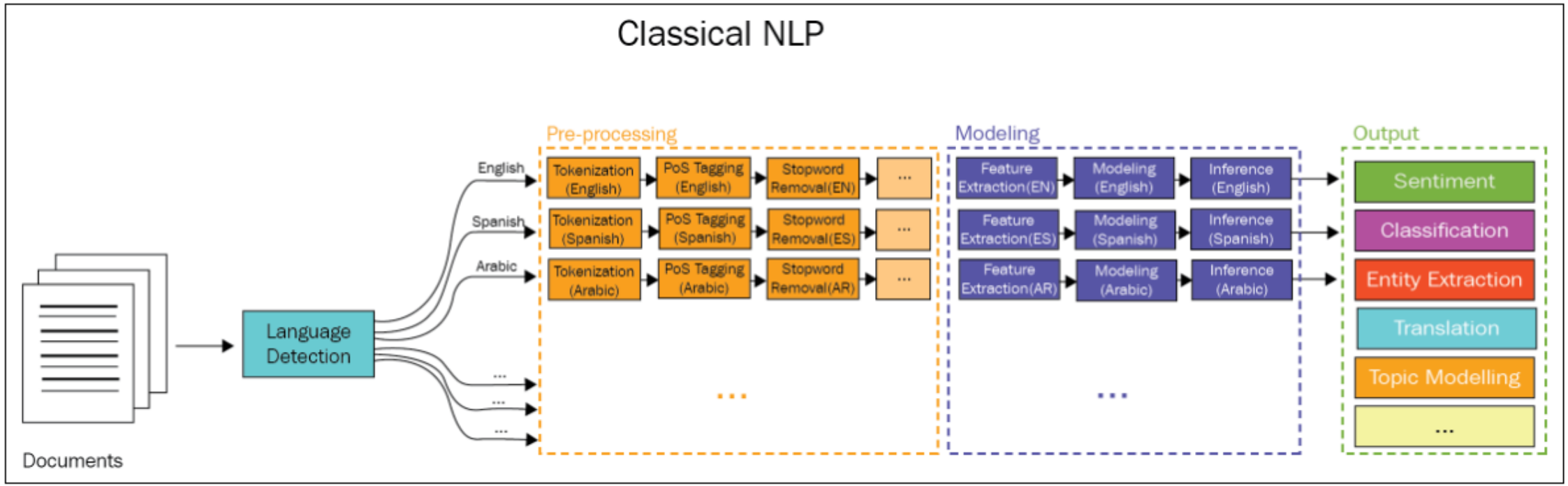
Words as vectors



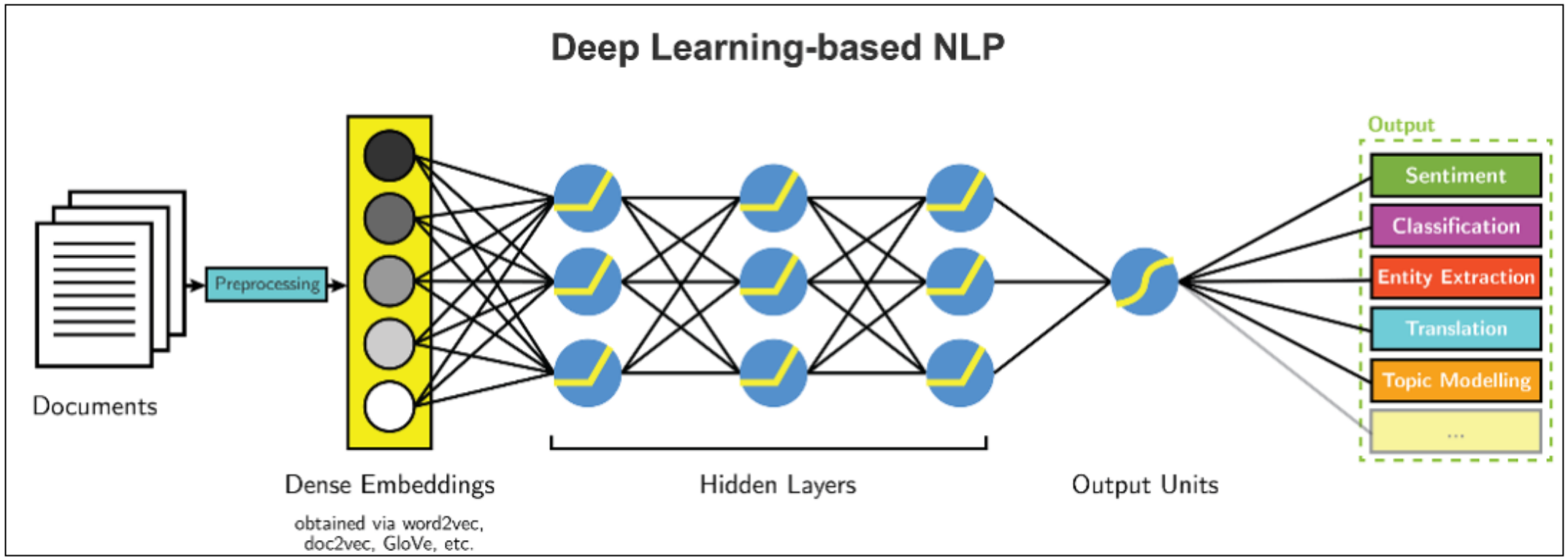
$$cos(\mathbf{u}, \mathbf{v}) = \frac{\mathbf{u} \cdot \mathbf{v}}{\|\mathbf{u}\|\|\mathbf{v}\|}$$

$$cos(\mathbf{u}, \mathbf{v}) = \frac{\sum_{i=1}^{V} u_i v_i}{\sqrt{\sum_{i=1}^{V} u_i^2}\sqrt{\sum_{i=1}^{V} v_i^2}}$$

What is the range of cos(·)?


Classical NLP
Pre-processing
Modeling
Output
English
Spanish
Arabic
Documents
Language Detection
Tokenization (English)
PoS Tagging (English)
Stopword Removal(EN)
Tokenization (Spanish)
PoS Tagging (Spanish)
Stopword Removal(ES)
Tokenization (Arabic)
PoS Tagging (Arabic)
Stopword Removal(AR)
Feature Extraction(EN)
Modeling (English)
Inference (English)
Feature Extraction(ES)
Modeling (Spanish)
Inference (Spanish)
Feature Extraction(AR)
Modeling (Arabic)
Inference (Arabic)
Sentiment
Classification
Entity Extraction
Translation
Topic Modelling

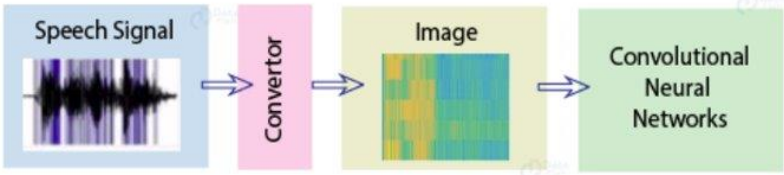
Feed-forward neural LM is a standard feedforward network that takes as input at time *t a representation of some number of previous words* (w_{t-1},w_{t-2}…) and outputs *probability distribution* over possible next words

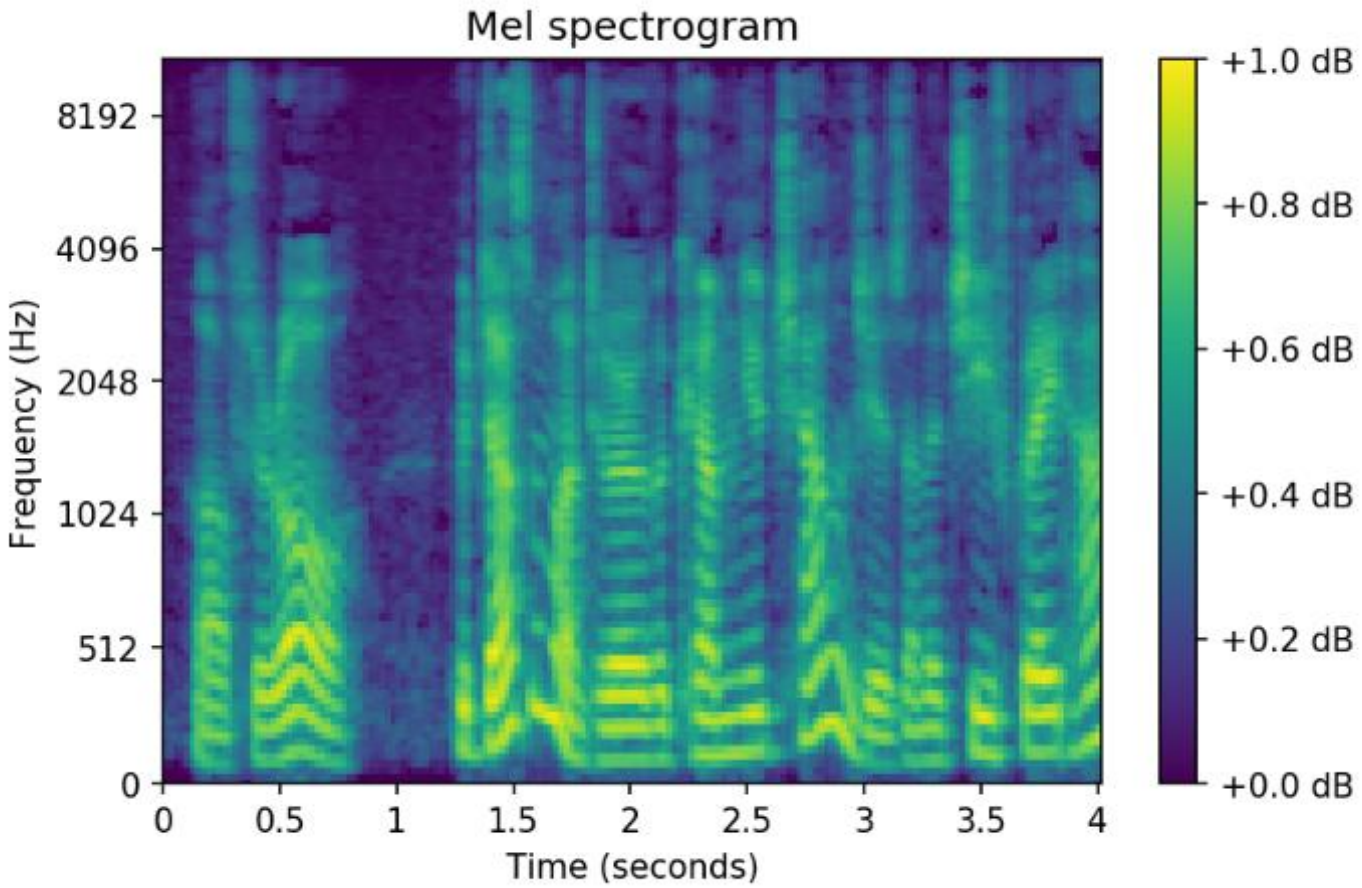


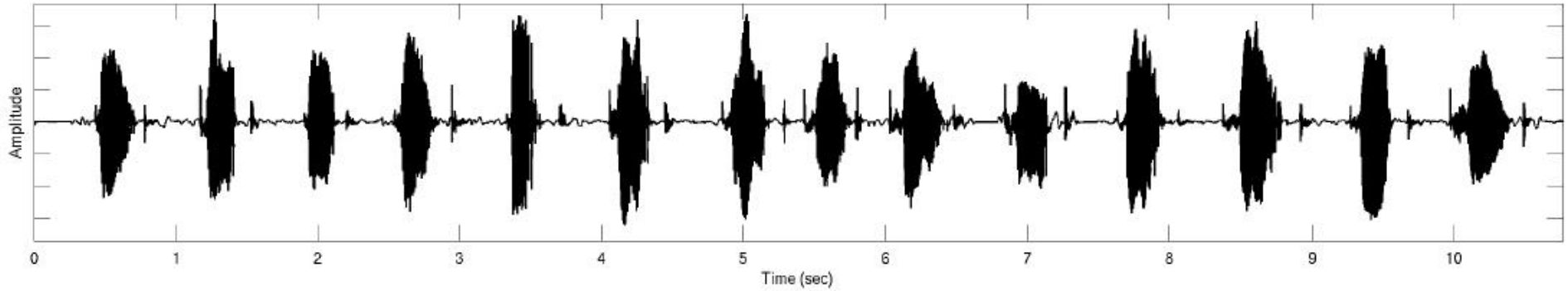
Transformers and large language models

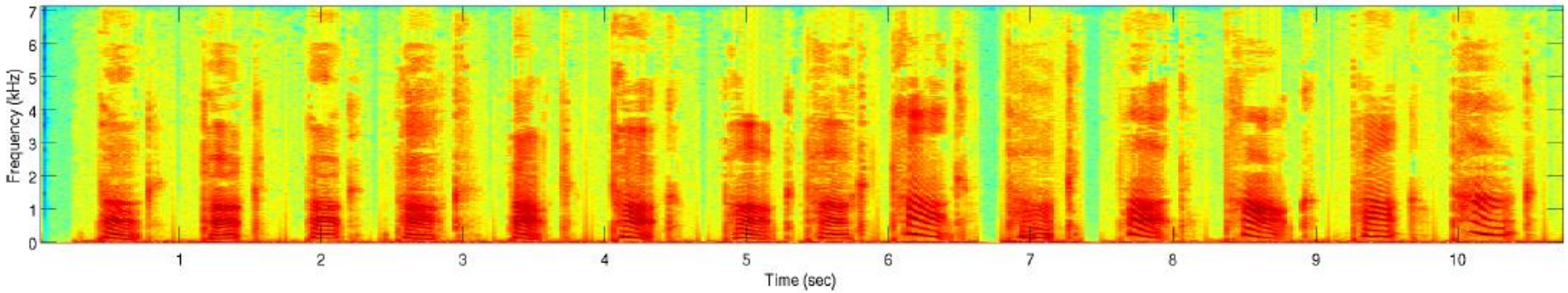
What about sound?

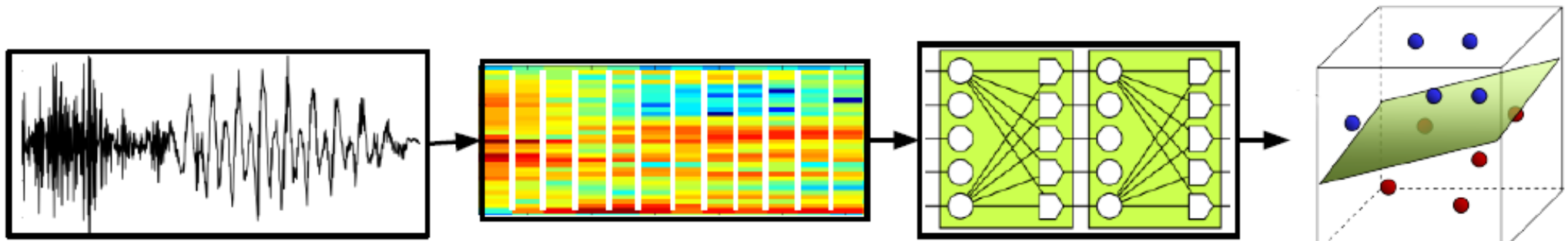
Mel Spectrogram











input waveform

layer 1
("base" encoding)

layers 2 - n
(representation)

linear learner (RLS)

Class 7

Loose hands, questions