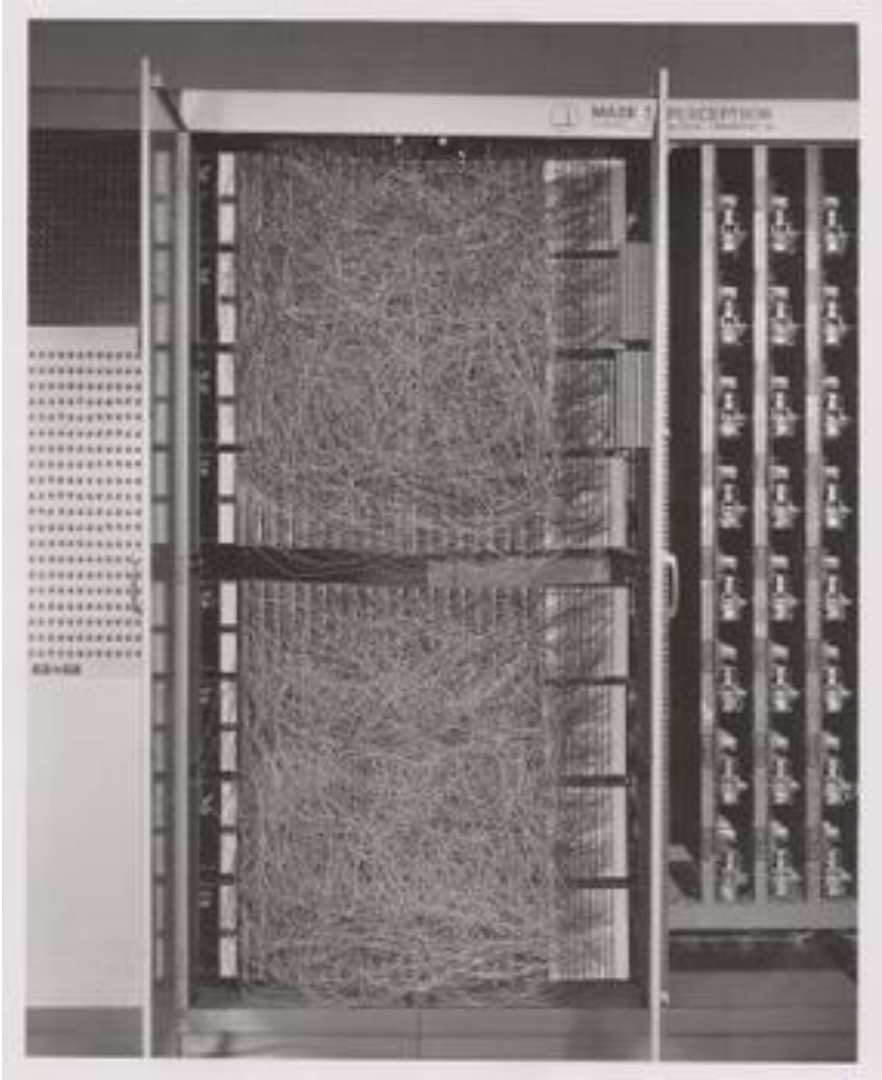
Outline

- **Single Layer Perceptron**
  - Decision Boundary
  - XOR is hard
- **Multilayer Perceptron**
  - Layers
  - Nonlinearities
  - Computational Cost

Perceptron

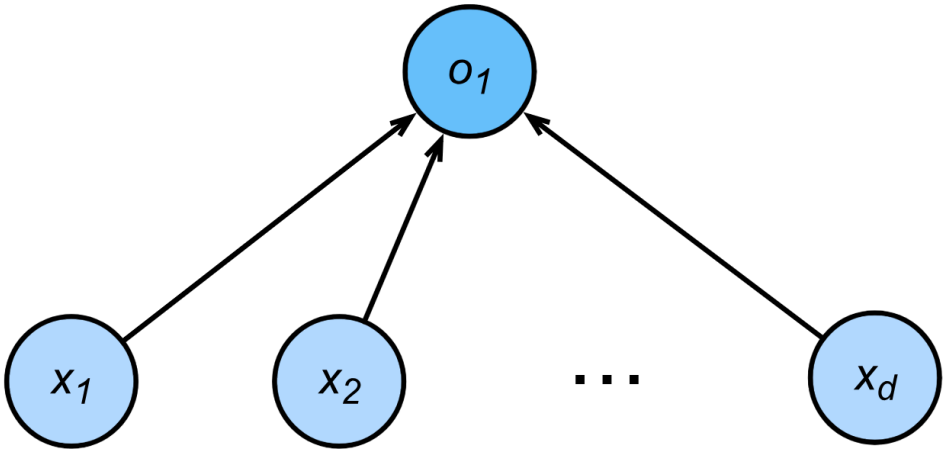
Mark I Perceptron, 1960
(wikipedia.org)



Perceptron

- Given input $\mathbf{x}$, weight $\mathbf{w}$ and bias b, perceptron outputs:

$$o = \sigma\left(\langle \mathbf{w}, \mathbf{x} \rangle + b\right) \qquad \sigma(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

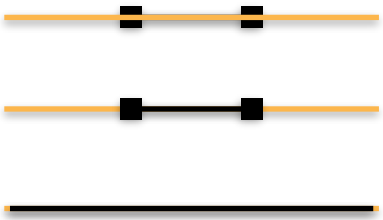


Perceptron

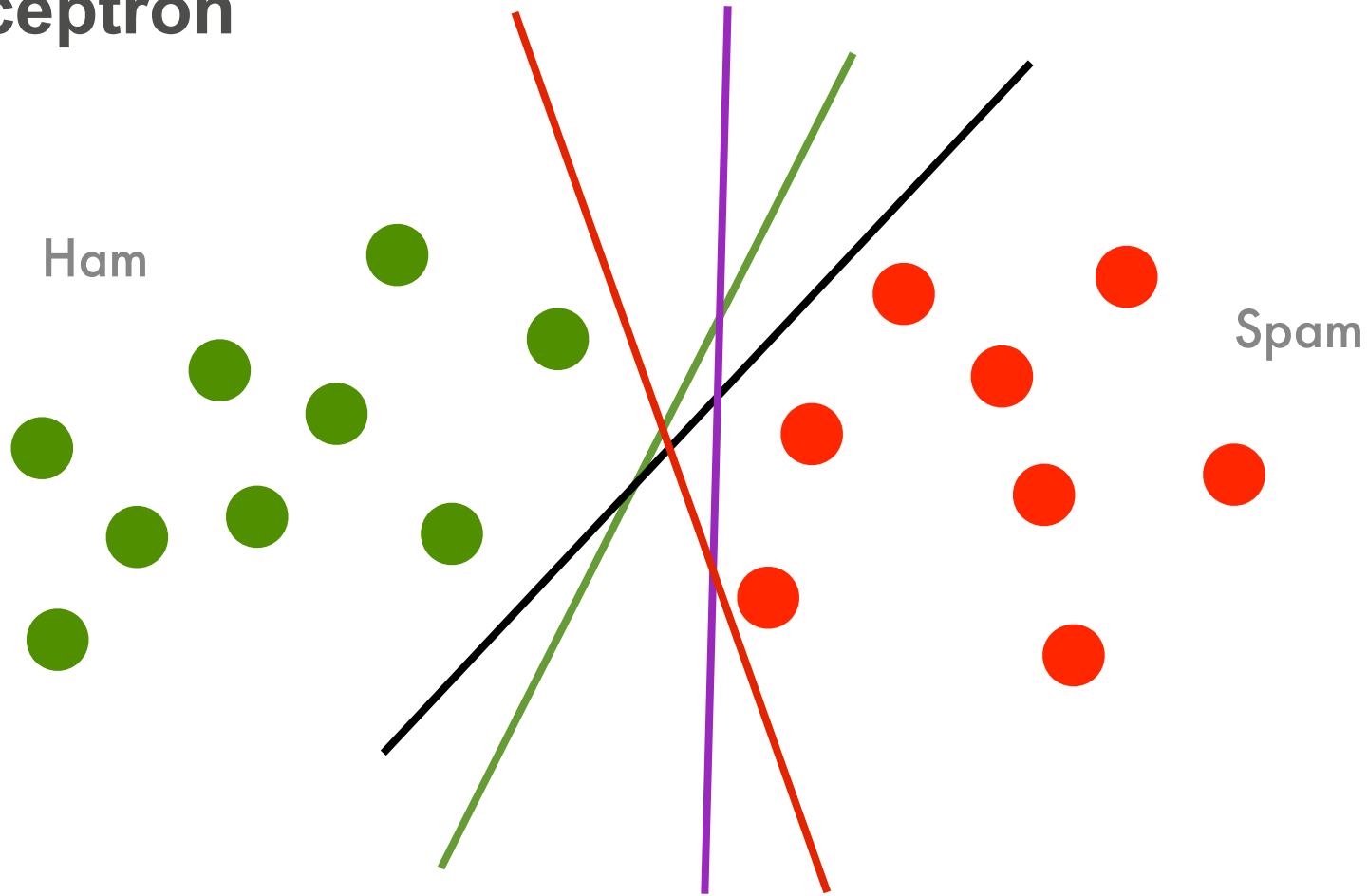
- Given input $\mathbf{x}$, weight $\mathbf{w}$ and bias b, perceptron outputs:

$$o = \sigma\left(\langle \mathbf{w}, \mathbf{x} \rangle + b\right) \qquad \sigma(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

- Binary classification (0 or 1)
  - Vs. scalar real value for regression
  - Vs. probabilities for logistic regression



Perceptron



Training the Perceptron

initialize $w = 0$ and $b = 0$

repeat

  if $y_i \left[\langle w, x_i \rangle + b\right] \leq 0$ **then**

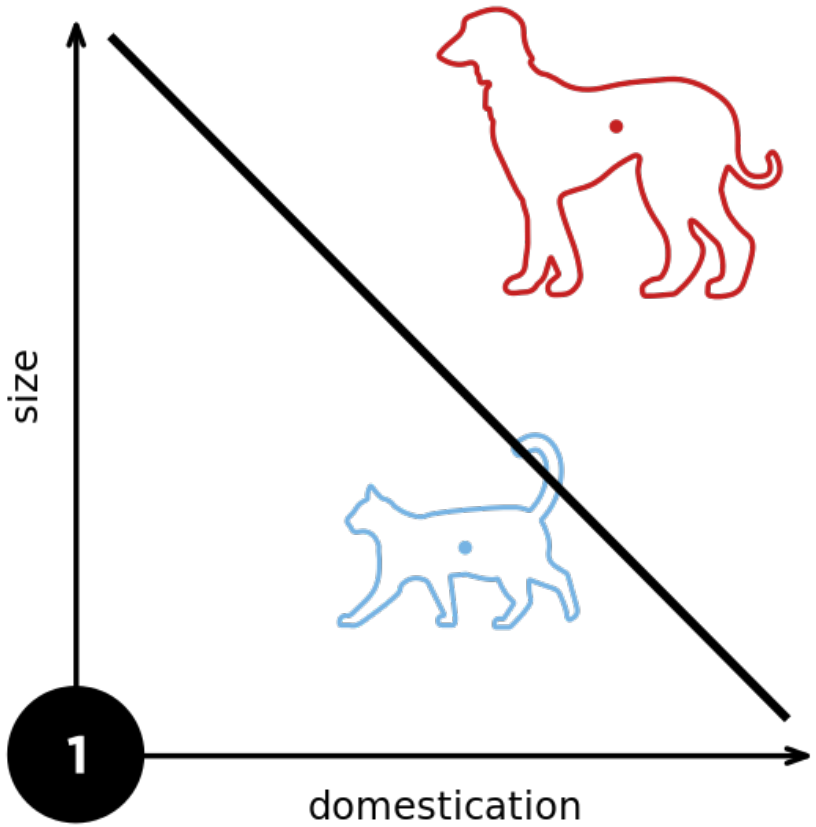
    $w \leftarrow w + y_i x_i$ and $b \leftarrow b + y_i$

  end if

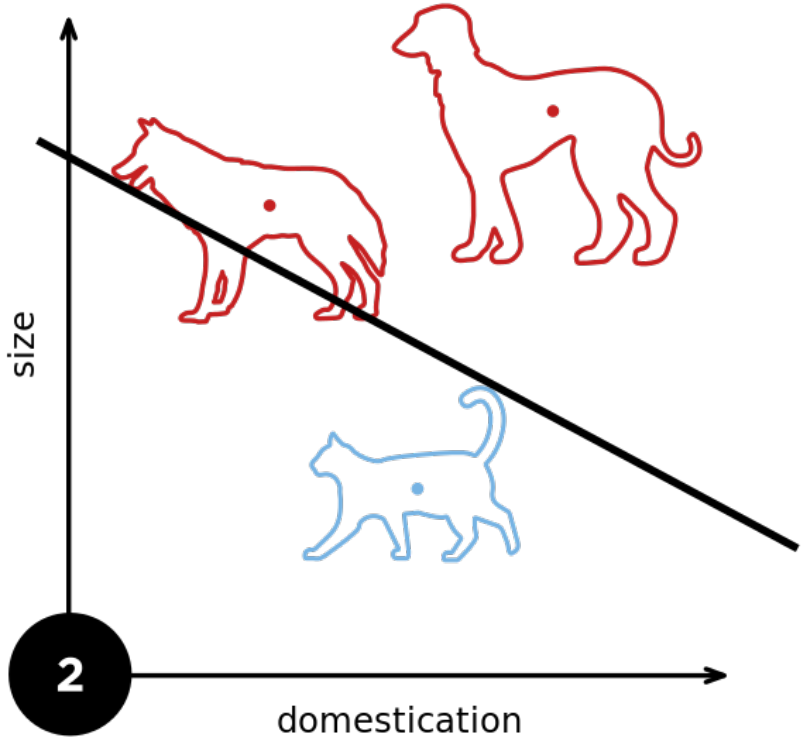
until all classified correctly

Equals to SGD (batch size is 1) with the following loss

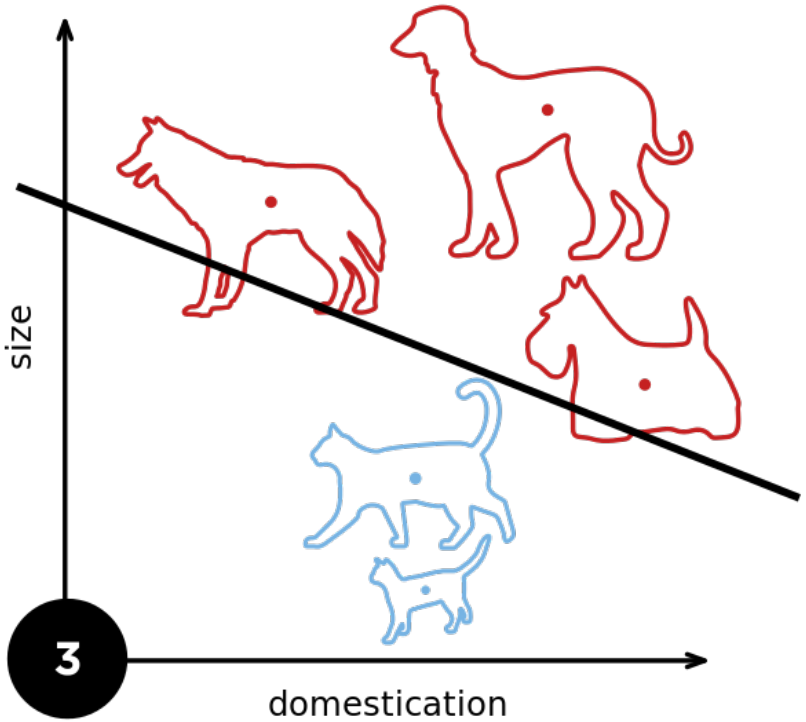
$$\ell(y, \mathbf{x}, \mathbf{w}) = \max(0, -y\langle \mathbf{w}, \mathbf{x} \rangle)$$



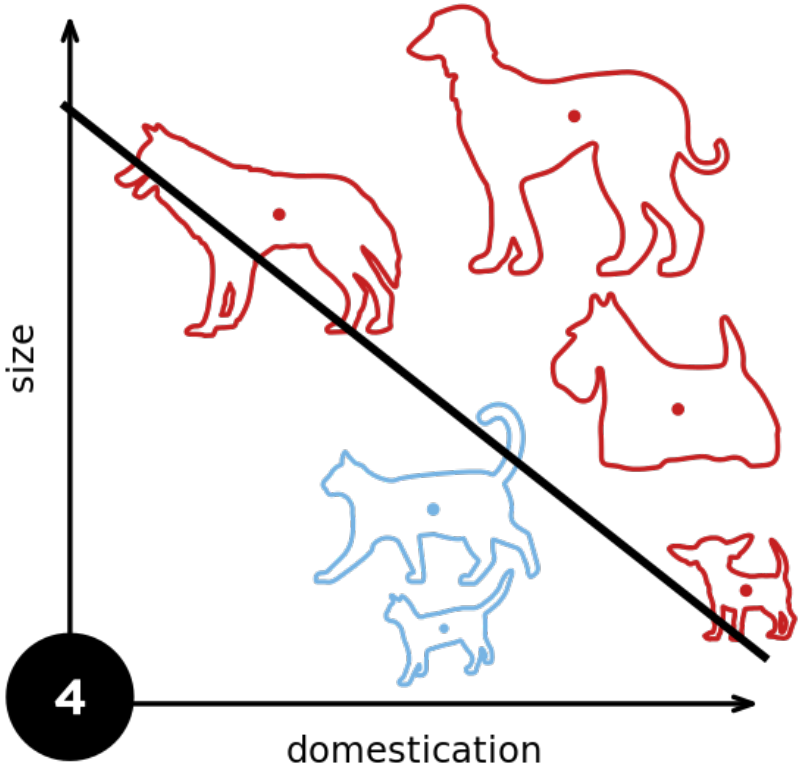
From wikipedia



From wikipedia



From wikipedia



From wikipedia

Convergence Theorem

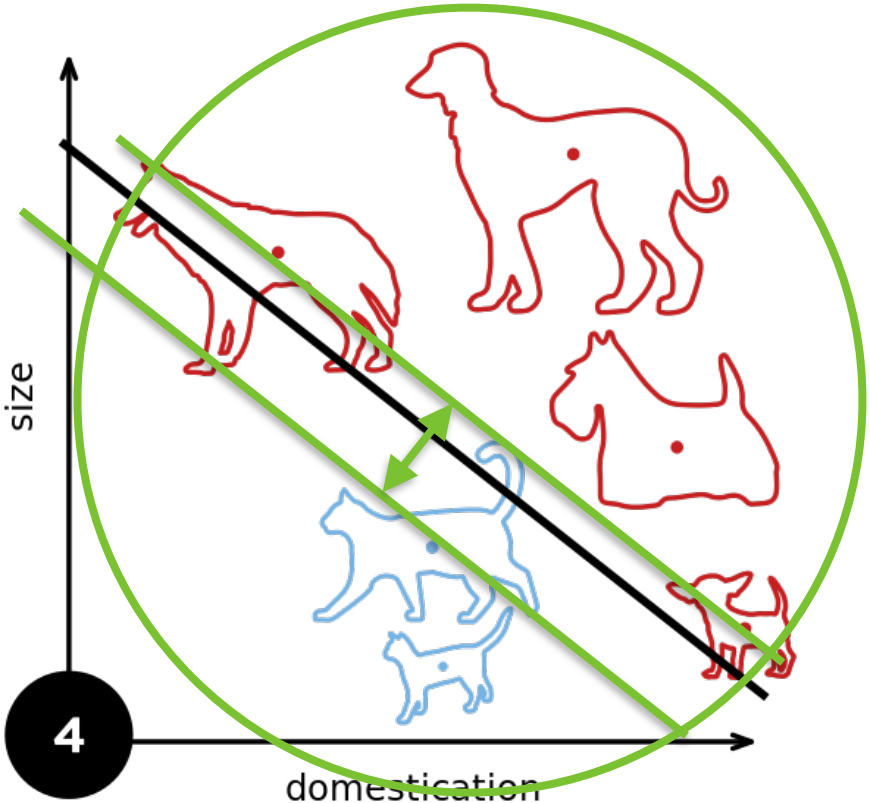
- Radius r enclosing the data
- Margin ρ separating the classes

$$y(\mathbf{x}^\top \mathbf{w} + b) \geq \rho$$

for $\|\mathbf{w}\|^2 + b^2 \leq 1$

- Guaranteed that perceptron will converge after

$$\frac{r^2 + 1}{\rho^2} \text{ steps}$$



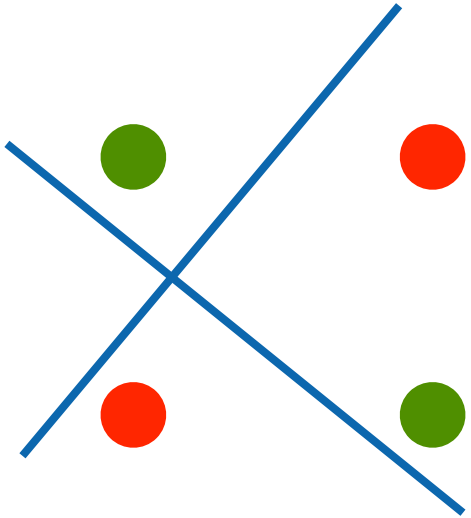
Consequences

- Only need to store errors. This gives a compression bound for perceptron.
- Fails with noisy data

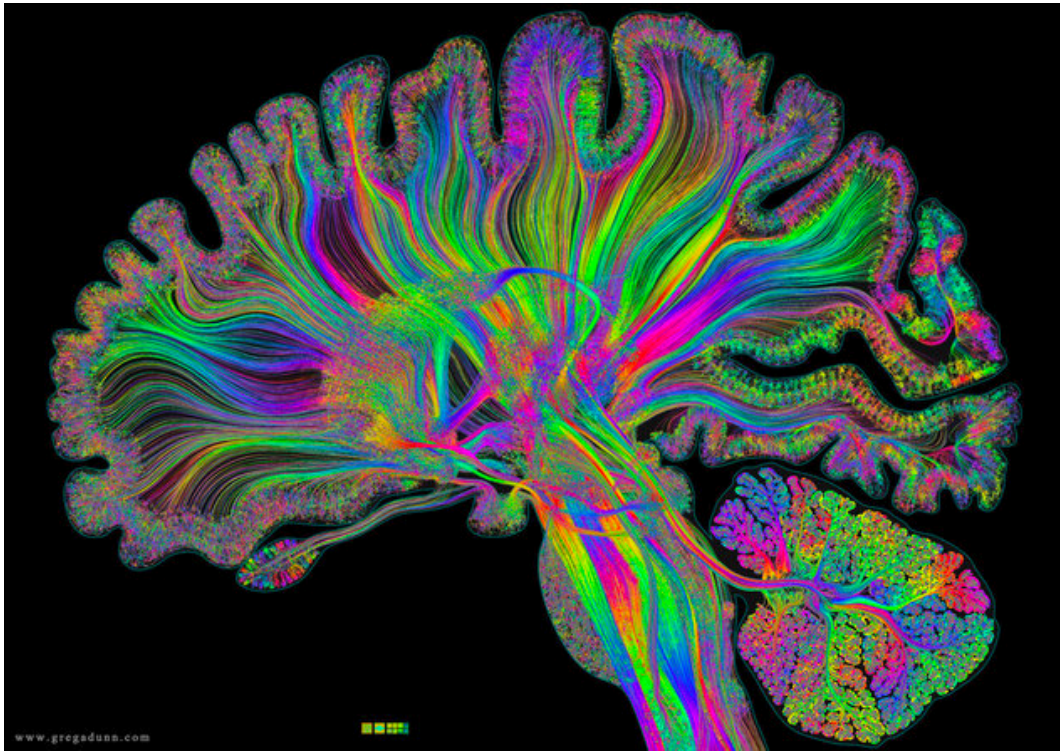
do NOT train your avatar with perceptrons

XOR Problem (Minsky & Papert, 1969)

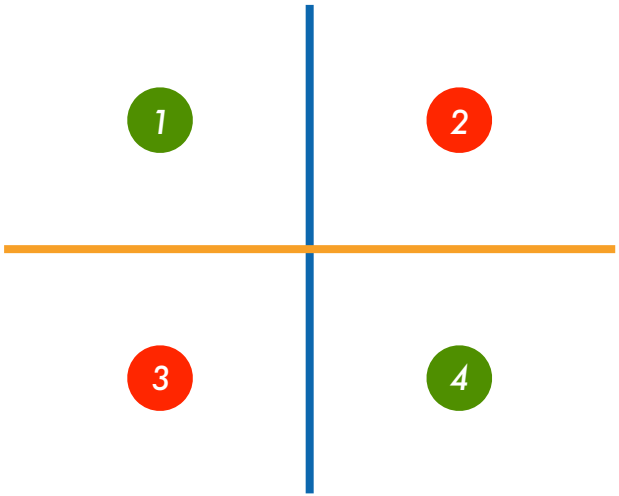
The perceptron cannot learn an XOR function
(neurons can only generate linear separators)



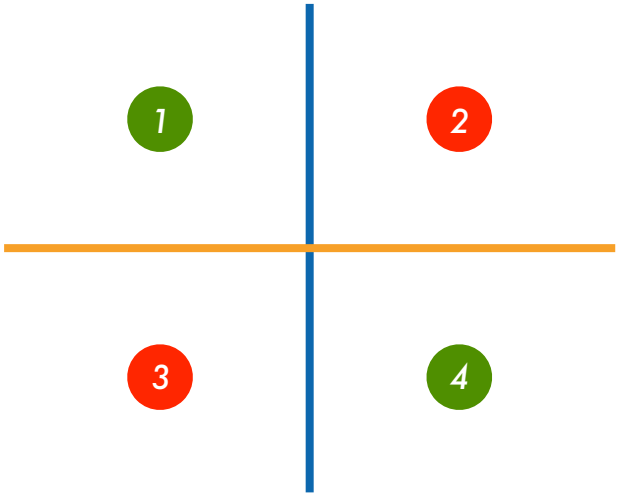
Multilayer Perceptron



Learning XOR

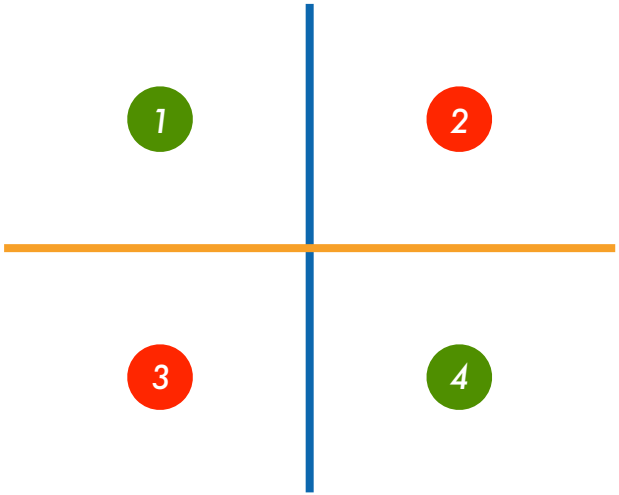


Learning XOR

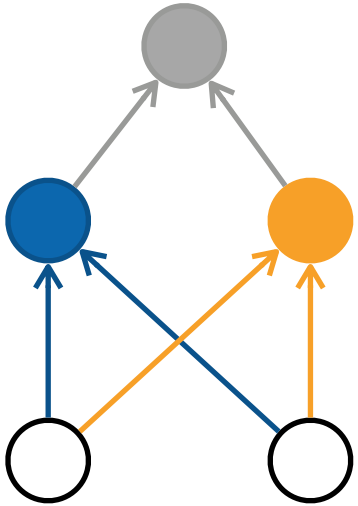


	1	2	3	4
	+	-	+	-
	+	+	-	-
product	+	-	-	+

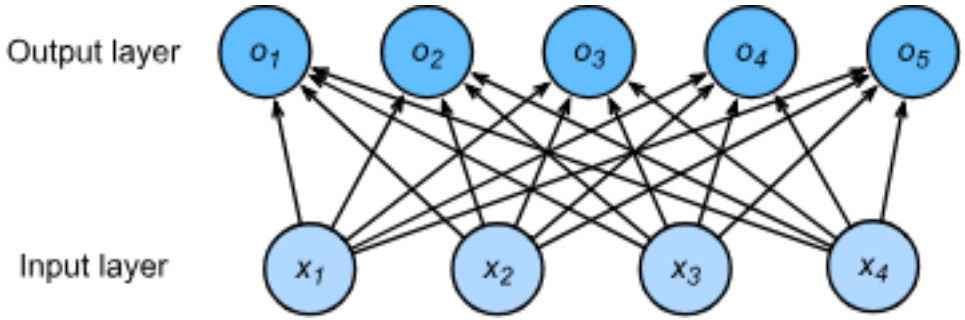
Learning XOR



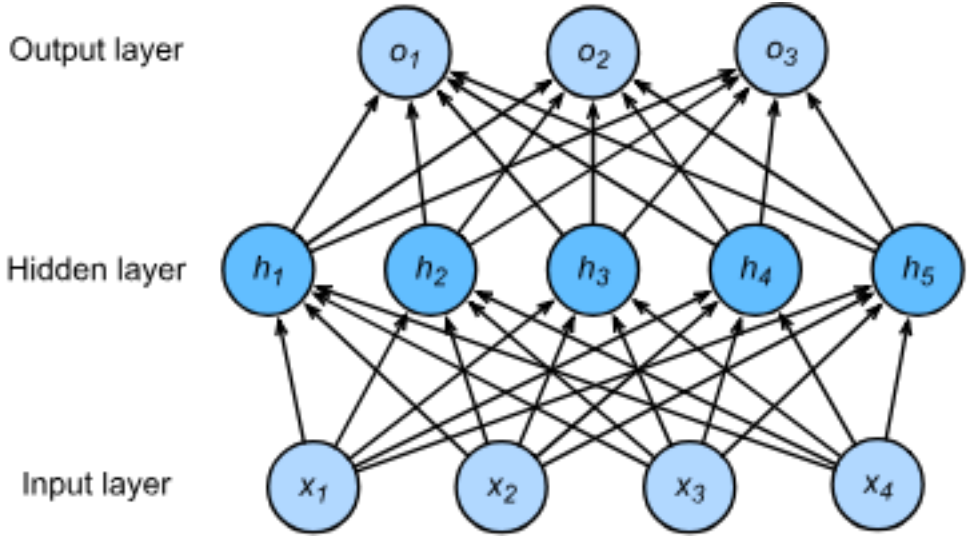
	1	2	3	4
	+	-	+	-
	+	+	-	-
product	+	-	-	+



Single Hidden Layer



Single Hidden Layer



Hyperparameter - size m of hidden layer

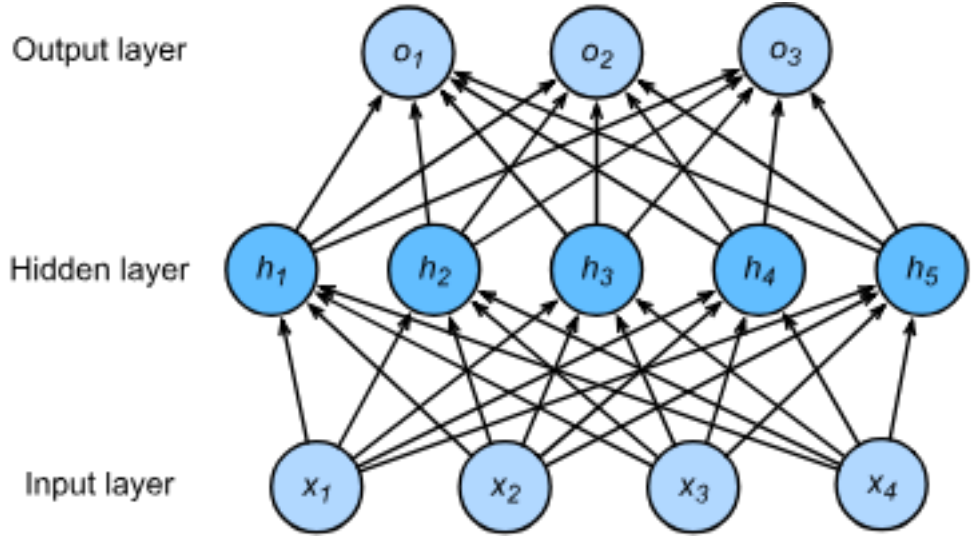
Single Hidden Layer

- Input $\mathbf{x} \in \mathbb{R}^n$
- Hidden $\mathbf{W}_1 \in \mathbb{R}^{m \times n}, \mathbf{b}_1 \in \mathbb{R}^m$
- Output $\mathbf{w}_2 \in \mathbb{R}^m, b_2 \in \mathbb{R}$

$$\mathbf{h} = \sigma(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1)$$

$$\mathbf{o} = \mathbf{w}_2^T \mathbf{h} + b_2$$

σ is an element-wise activation function



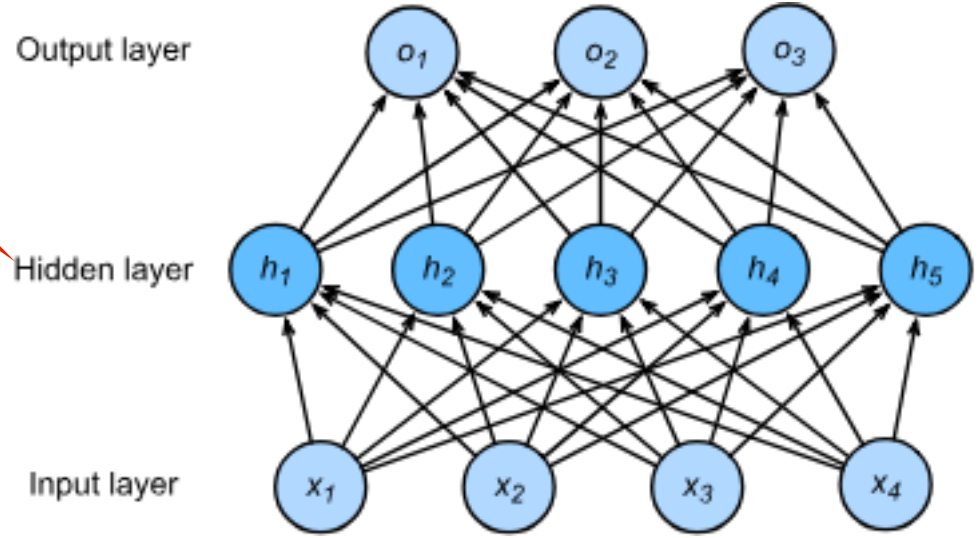
Single Hidden Layer

Why do we need an a nonlinear activation?

$$\mathbf{h} = \sigma(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)$$

$$\mathbf{o} = \mathbf{w}_2^T\mathbf{h} + b_2$$

σ is an element-wise activation function



Single Hidden Layer

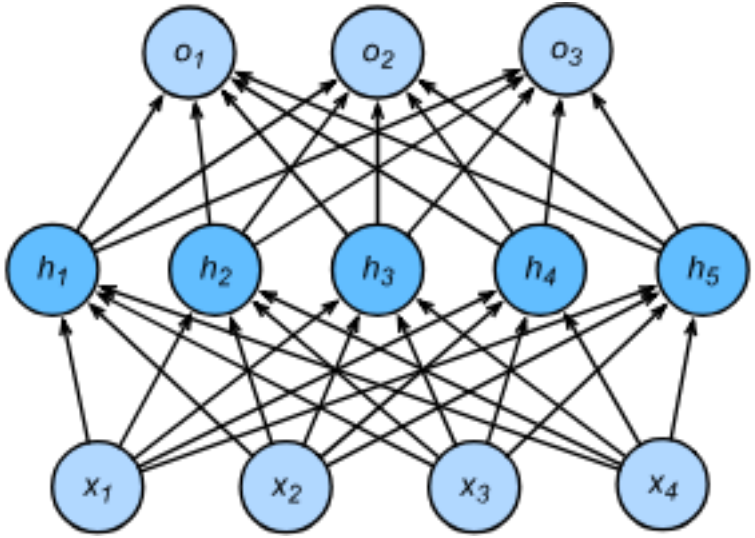
Why do we need an a nonlinear activation?

$$\mathbf{h} = \mathbf{W}_1\mathbf{x} + \mathbf{b}_1$$

$$\mathbf{o} = \mathbf{w}_2^T\mathbf{h} + b_2$$

hence $o = \mathbf{w}_2^\top\mathbf{W}_1\mathbf{x} + b'$

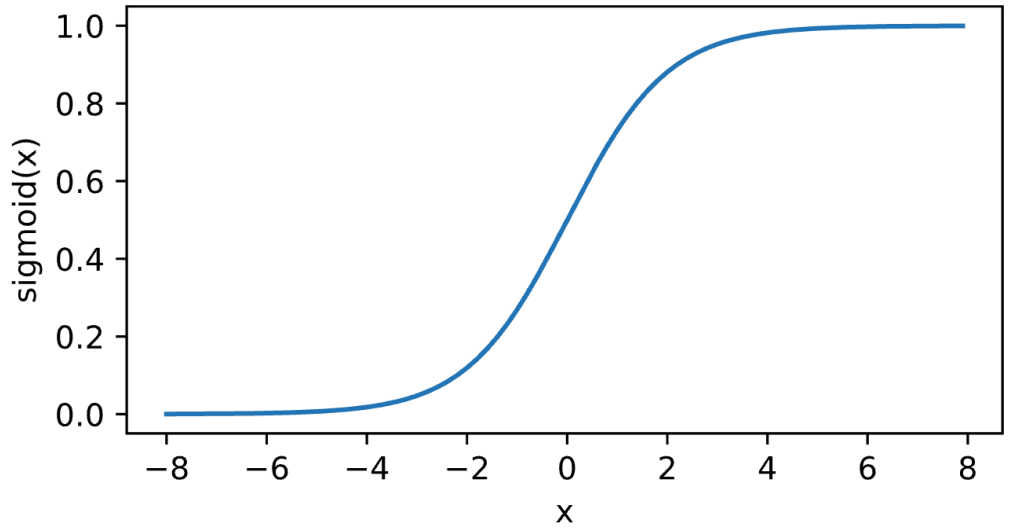
Linear …



Sigmoid Activation

Map input into (0, 1), a soft version of $\sigma(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$

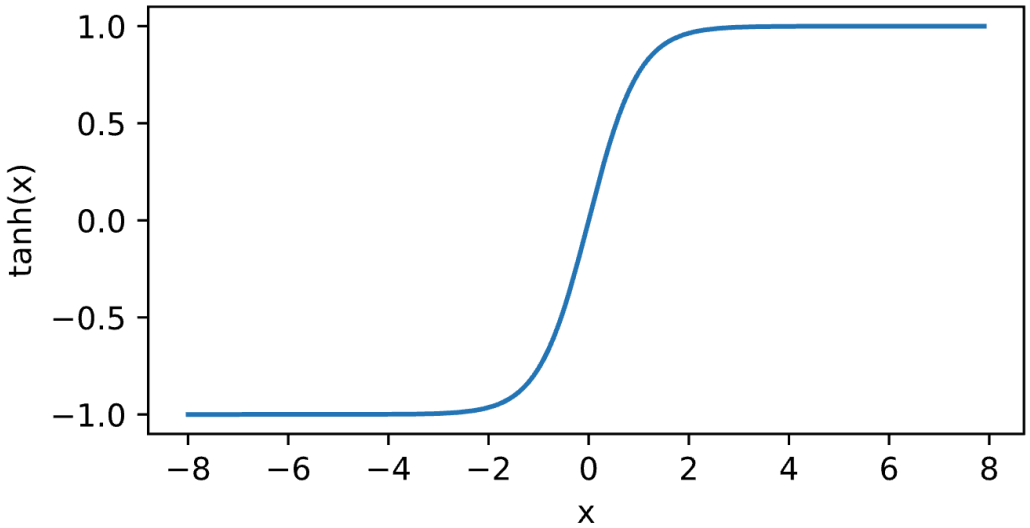
$$\text{sigmoid}(x) = \frac{1}{1 + \exp(-x)}$$



Tanh Activation

Map inputs into (-1, 1)

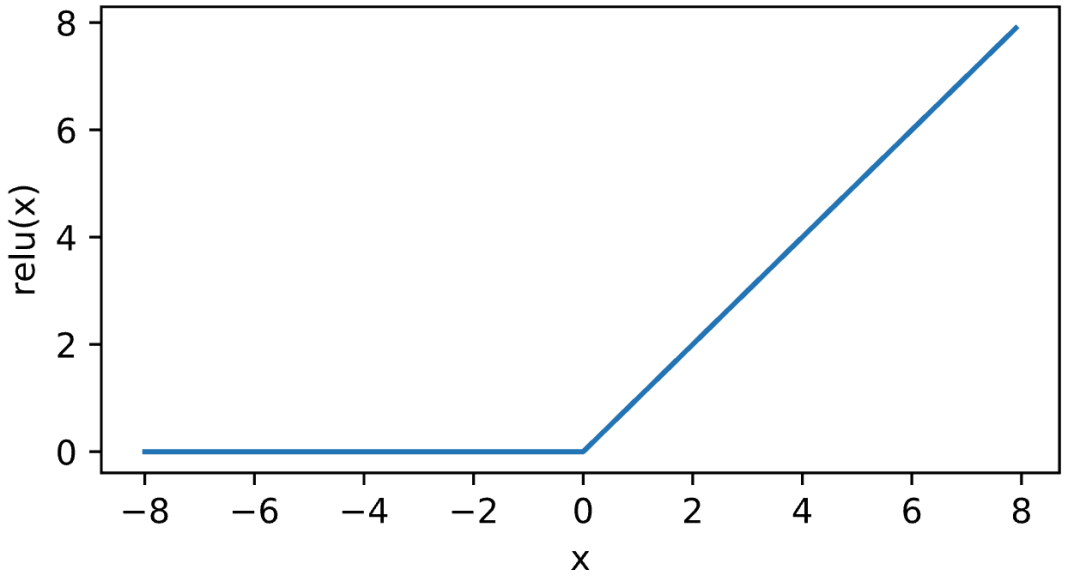
$$\tanh(x) = \frac{1 - \exp(-2x)}{1 + \exp(-2x)}$$



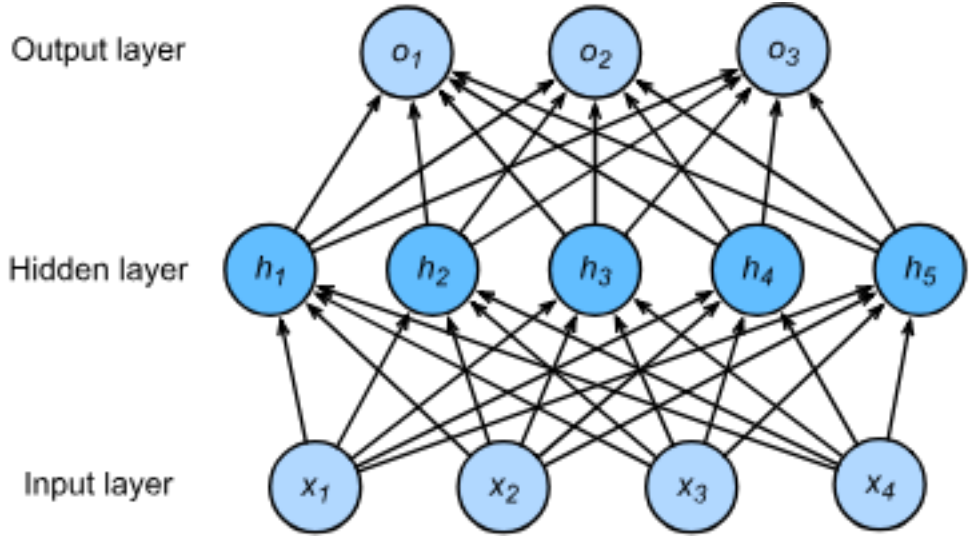
ReLU Activation

ReLU: rectified linear unit

$$\text{ReLU}(x) = \max(x, 0)$$

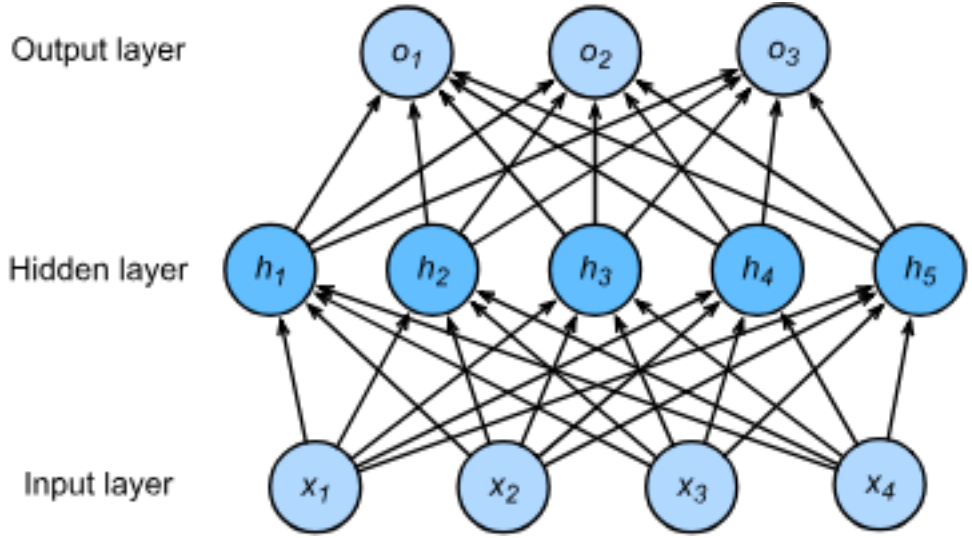


Multiclass Classification



Multiclass Classification

$$y_1, y_2, \ldots, y_k = \text{softmax}(o_1, o_2, \ldots, o_k)$$



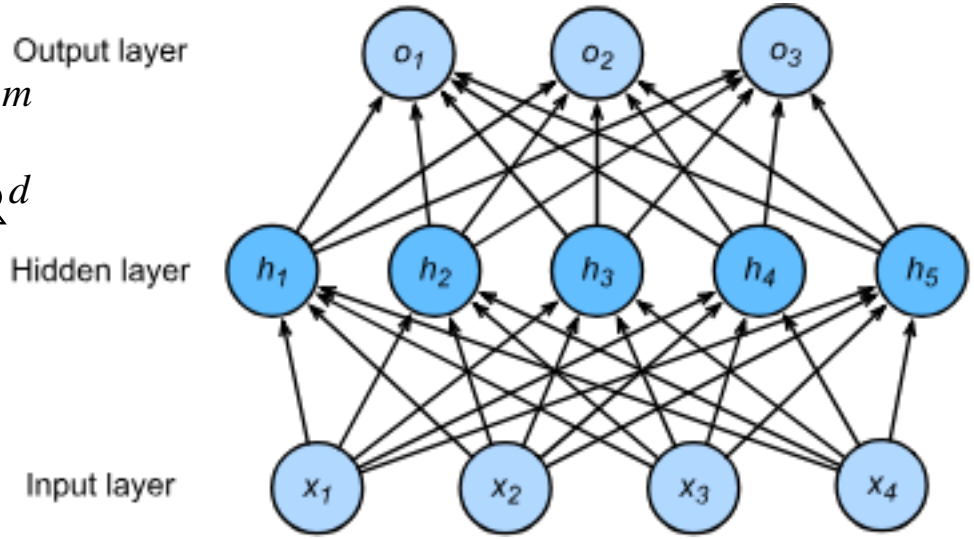
Multiclass Classification

- Input $\mathbf{x} \in \mathbb{R}^n$
- Hidden $\mathbf{W}_1 \in \mathbb{R}^{m \times n}$ and $\mathbf{b}_1 \in \mathbb{R}^m$
- Output $\mathbf{W}_2 \in \mathbb{R}^{m \times d}$ and $\mathbf{b}_2 \in \mathbb{R}^d$

$$\mathbf{h} = \sigma(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1)$$

$$\mathbf{o} = \mathbf{w}_2^T \mathbf{h} + \mathbf{b}_2$$

$$\mathbf{y} = \text{softmax}(o)$$



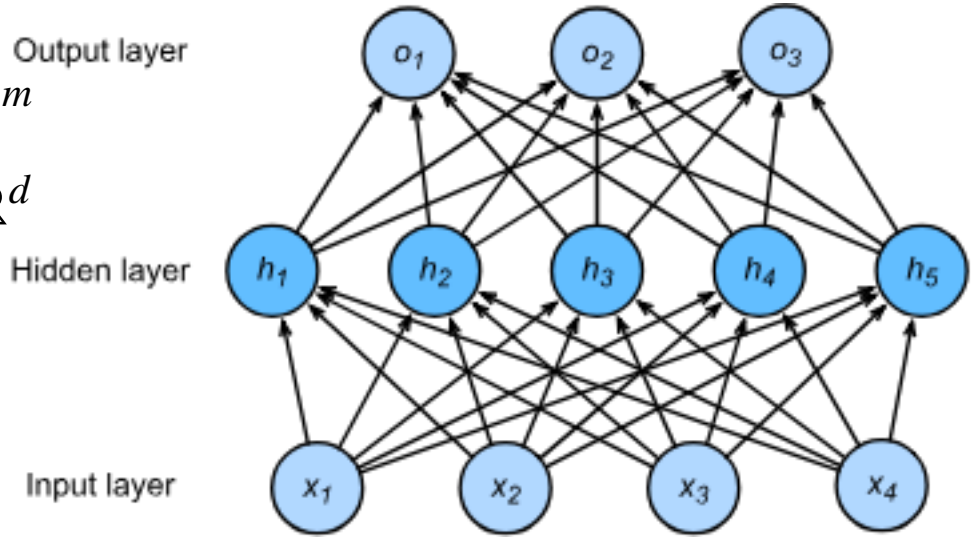
Multiclass Classification

- Input $\mathbf{x} \in \mathbb{R}^n$
- Hidden $\mathbf{W}_1 \in \mathbb{R}^{m \times n}$ and $\mathbf{b}_1 \in \mathbb{R}^m$
- Output $\mathbf{W}_2 \in \mathbb{R}^{m \times d}$ and $\mathbf{b}_2 \in \mathbb{R}^d$

$$\mathbf{h} = \sigma(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1)$$

$$\mathbf{o} = \mathbf{w}_2^T \mathbf{h} + \mathbf{b}_2$$

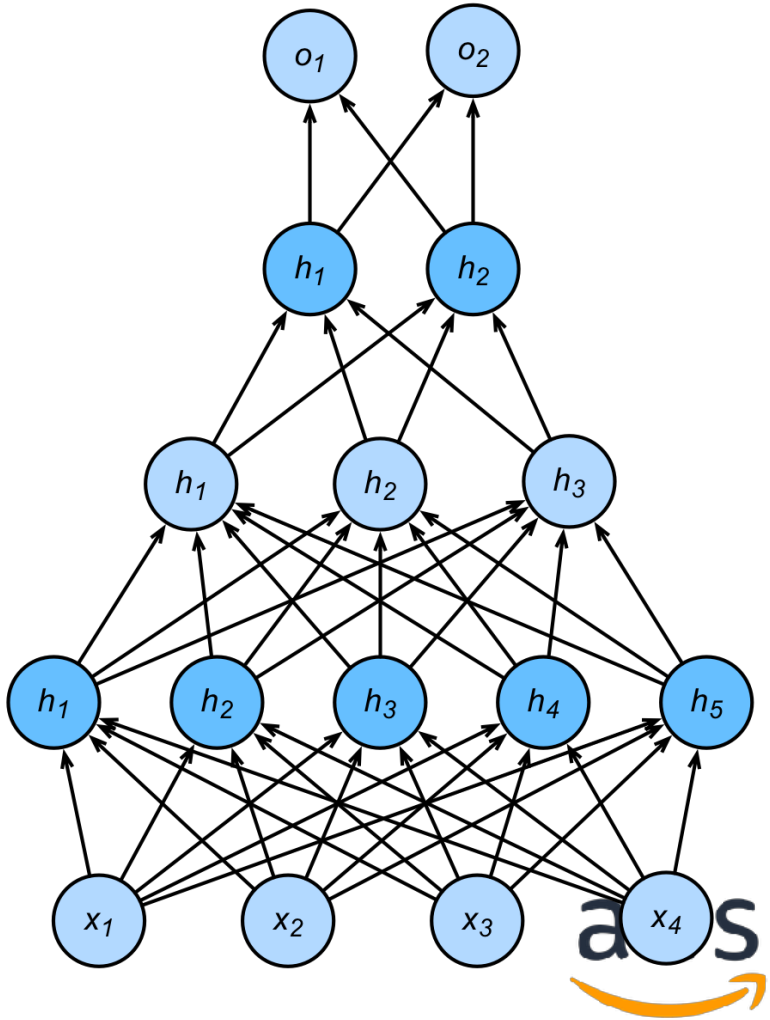
$$\mathbf{y} = \text{softmax}(o)$$



Multiple Hidden Layers

$$\mathbf{h}_1 = \sigma(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)$$
$$\mathbf{h}_2 = \sigma(\mathbf{W}_2\mathbf{h}_1 + \mathbf{b}_2)$$
$$\mathbf{h}_3 = \sigma(\mathbf{W}_3\mathbf{h}_2 + \mathbf{b}_3)$$
$$\mathbf{o} = \mathbf{W}_4\mathbf{h}_3 + \mathbf{b}_4$$

Hyper-parameters
- # of hidden layers
- Hidden size for each layer



Summary

- Perceptron
  - Simple updates
  - Limited function complexity
- Multilayer Perceptron
  - Multiple layers add more complexity
  - Nonlinearity is needed
  - Simple composition (but architecture search needed)

Model Evaluation

Training Error and Generalization Error

- Training error: model error on the training data
- Generalization error: model error on new data
- Example: practice a future exam with past exams
  - Doing well on past exams (training error) doesn't guarantee a good score on the future exam (generalization error)
  - Student A gets 0 error on past exams by rote learning
  - Student B understands the reasons for given answers

Validation Dataset and Test Dataset

- Validation dataset: a dataset used to evaluate the model
  - E.g. Take out 50% of the training data
  - Should not be mixed with the training data (#1 mistake)
- Test dataset: a dataset can be used once, e.g.
  - A future exam
  - The house sale price I bided
  - Dataset used in private leaderboard in Kaggle

K-fold Cross Validation

- Useful when not sufficient data
- Algorithm:
  - Partition the training data into *K* parts
  - For *i = 1, …, K*
    - Use the *i*-th part as the validation set, the rest for training
  - Report the averaged the *K* validation errors
- Popular choices: *K = 5 or 10*

Underfitting
Overfitting

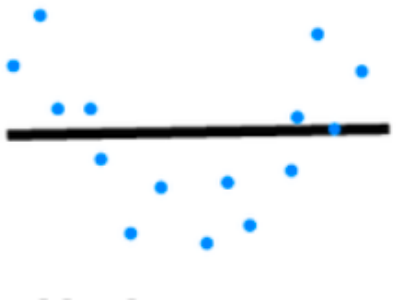
Image credit: hackernoon.com

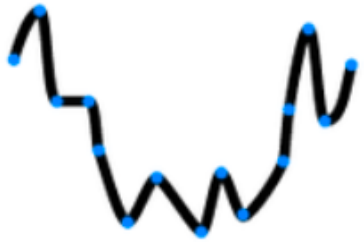
Underfitting and Overfitting

		Data complexity	
		Simple	Complex
Model capacity	Low	Normal	Underfitting
	High	Overfitting	Normal

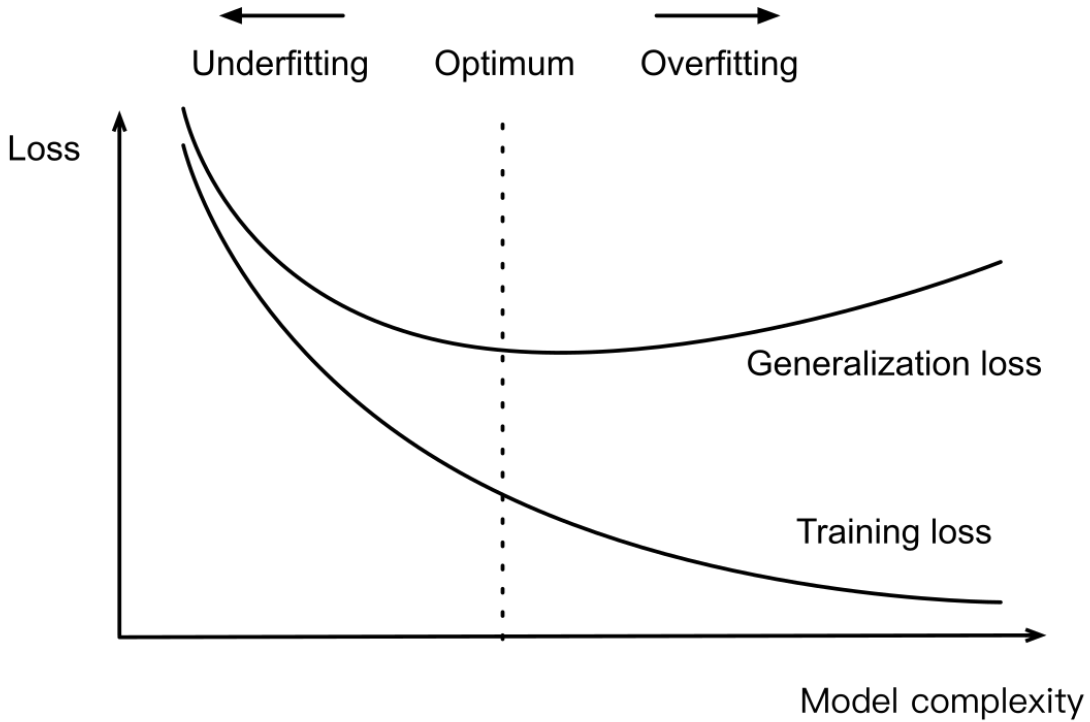
Model Capacity

- The ability to fit variety of functions
- Low capacity models struggles to fit training set
  - Underfitting
- High capacity models can memorize the training set
  - Overfitting





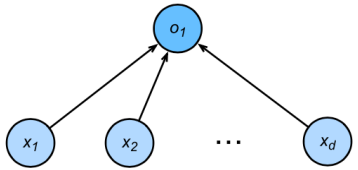
Influence of Model Complexity



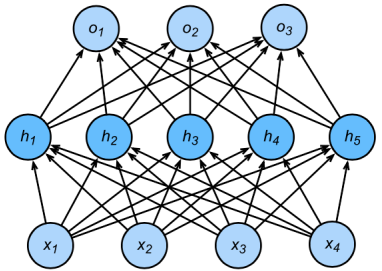
Estimate Model Capacity

- It's hard to compare complexity between different algorithms
  - e.g. tree vs neural network
- Given an algorithm family, two main factors matter:
  - The number of parameters
  - The values taken by each parameter

$d + 1$



$(d + 1)m + (m + 1)k$



VC Dimension

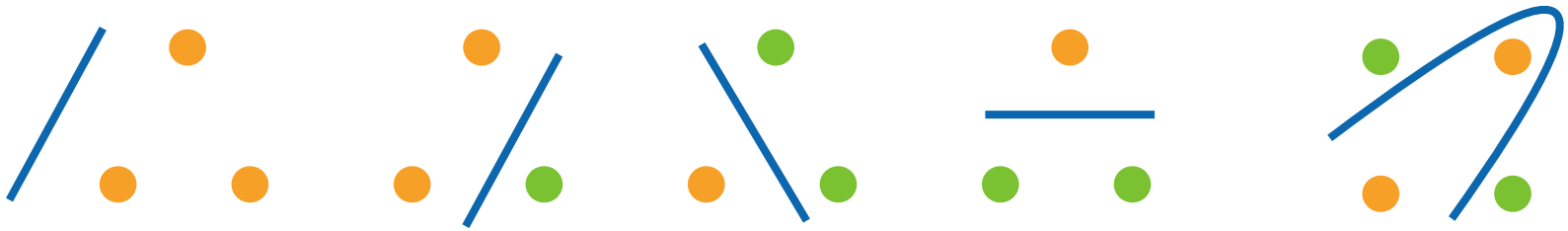
- A center topic in Statistic Learning Theory
- For a classification model, it's the size of the largest dataset, no matter how we assign labels, there exist a model to classify them perfectly

Vladimir **V**apnik

Alexey **C**hervonenkis

VC-Dimension for Linear Classifier

- 2-D perceptron: VCdim = 3
  - Can classify any 3 points, but not 4 points (xor)



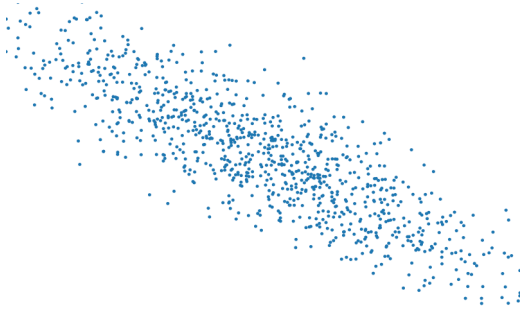
- Perceptron with *N* parameters: VCdim = N
- Some Multilayer Perceptrons: VCdim = $O(N \log_2(N))$

Usefulness of VC-Dimension

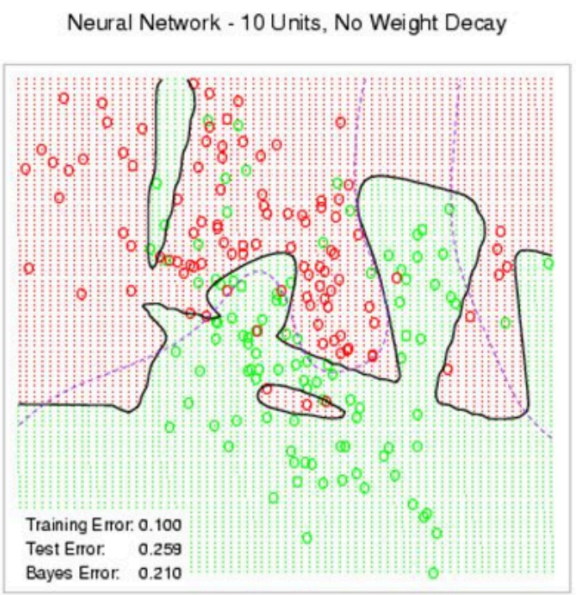
- Provides theory insights why a model works
  - Bound the gap between training error and generalization error
- Rarely used in practice with deep learning
  - The bounds are too loose
  - Difficulty to compute VC-dimension for deep neural networks
- Same for other statistic learning theory tools

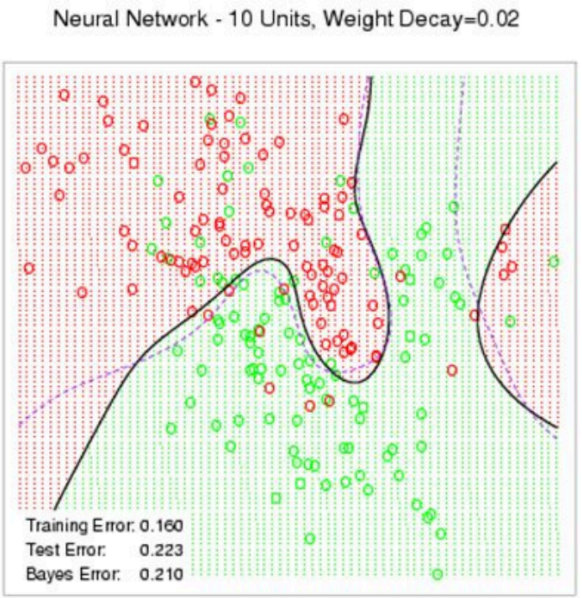
Data Complexity

- Multiple factors matters
  - # of examples
  - # of elements in each example
  - time/space structure
  - diversity



Weight Decay



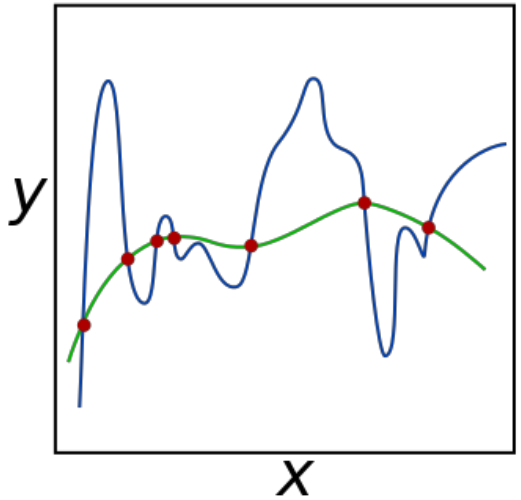


Squared Norm Regularization as Hard Constraint

- Reduce model complexity by limiting value range

$$\min \ \ell(\mathbf{w}, b) \quad \text{subject to} \quad \|\mathbf{w}\|^2 \leq \theta$$

  - Often do not regularize bias b
    - Doing or not doing has little difference in practice
  - A small θ means more regularization



Squared Norm Regularization as Soft Constraint

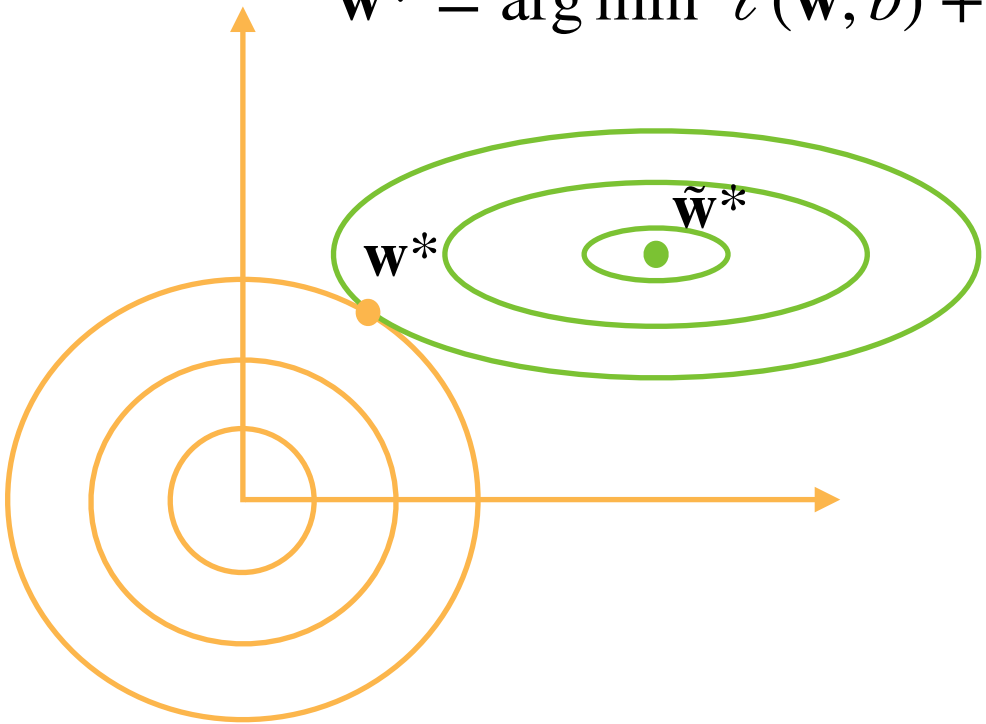
- For each θ, we can find λ to rewrite the hard constraint version as

$$\min \ \ell(\mathbf{w}, b) + \frac{\lambda}{2}\|\mathbf{w}\|^2$$

  - Can prove by Lagrangian multiplier method
  - Hyper-parameter λ controls regularization importance
  - $\lambda = 0$: no effect
  - $\lambda \to \infty, \mathbf{w}^* \to \mathbf{0}$

Illustrate the Effect on Optimal Solutions

$$\mathbf{w}^* = \arg\min \ \ell(\mathbf{w}, b) + \frac{\lambda}{2}\|\mathbf{w}\|^2$$



$$\tilde{\mathbf{w}}^* = \arg\min \ \ell(\mathbf{w}, b)$$

Update Rule

- Compute the gradient

$$\frac{\partial}{\partial \mathbf{w}} \left(\ell(\mathbf{w}, b) + \frac{\lambda}{2} \|\mathbf{w}\|^2 \right) = \frac{\partial \ell(\mathbf{w}, b)}{\partial \mathbf{w}} + \lambda \mathbf{w}$$

- Update weight at time t

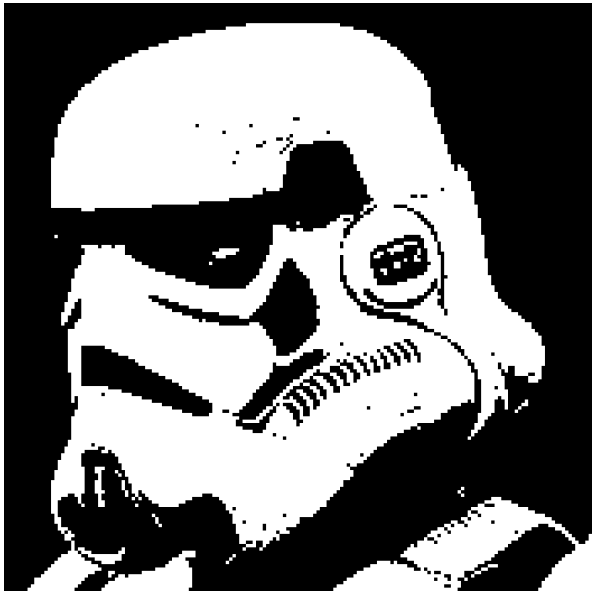
$$\mathbf{w}_{t+1} = (1 - \eta\lambda)\mathbf{w}_t - \eta \frac{\partial \ell(\mathbf{w}_t, b_t)}{\partial \mathbf{w}_t}$$

  - Often $\eta\lambda < 1$, so also called weight decay in deep learning

Dropout

Motivation

- A good model should be robust under modest changes in the input
  - Training with input noise equals to Tikhonov Regularization
  - Dropout: inject noises into internal layers



Add Noise without Bias

- Add noise into ***x*** to get ***x'***, we hope

$$\mathbf{E}[\mathbf{x}'] = \mathbf{x}$$

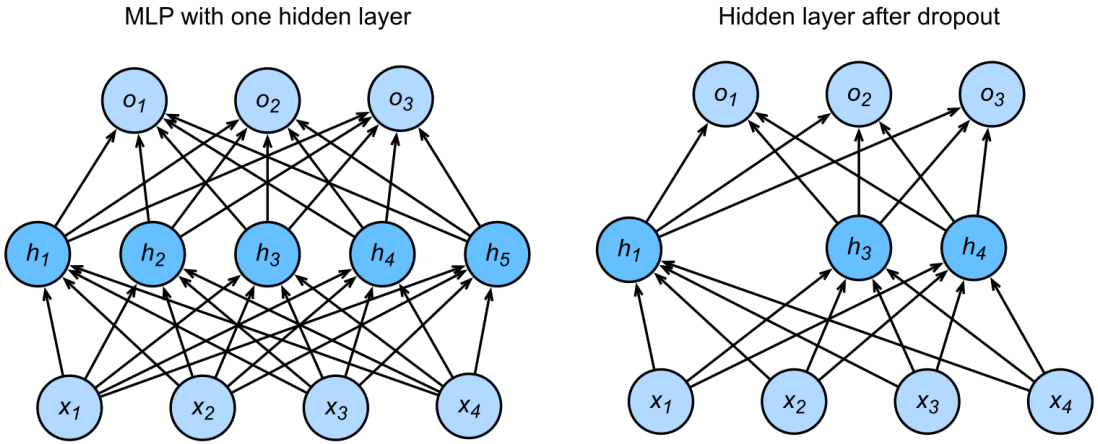
- Dropout perturbs each element by

$$x_i' = \begin{cases} 0 & \text{with probablity } p \\ \frac{x_i}{1-p} & \text{otherise} \end{cases}$$

Apply Dropout

- Often apply dropout on the output of hidden fully-connected layers

$$\mathbf{h} = \sigma(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)$$
$$\mathbf{h}' = \text{dropout}(\mathbf{h})$$
$$\mathbf{o} = \mathbf{W}_2\mathbf{h}' + \mathbf{b}_2$$
$$\mathbf{y} = \text{softmax}(o)$$



Dropout in Inference

- Regularization is only used in training
- The dropout layer for inference is

$$\mathbf{h}' = \text{dropout}(\mathbf{h})$$

  - Guarantee deterministic results