Neural Networks

A. M. Sadeghzadeh, Ph.D.

Sharif University of Technology
Computer Engineering Department (CE)
Data and Network Security Lab (DNSL)

February 25, 2023

Most slides have been adapted from Bhiksha Raj, 11-785, CMU 2020, Justin Johnson, EECS 498-007, University of Michigan 2020, and Fei Fei Li, cs231n, Stanford 2017

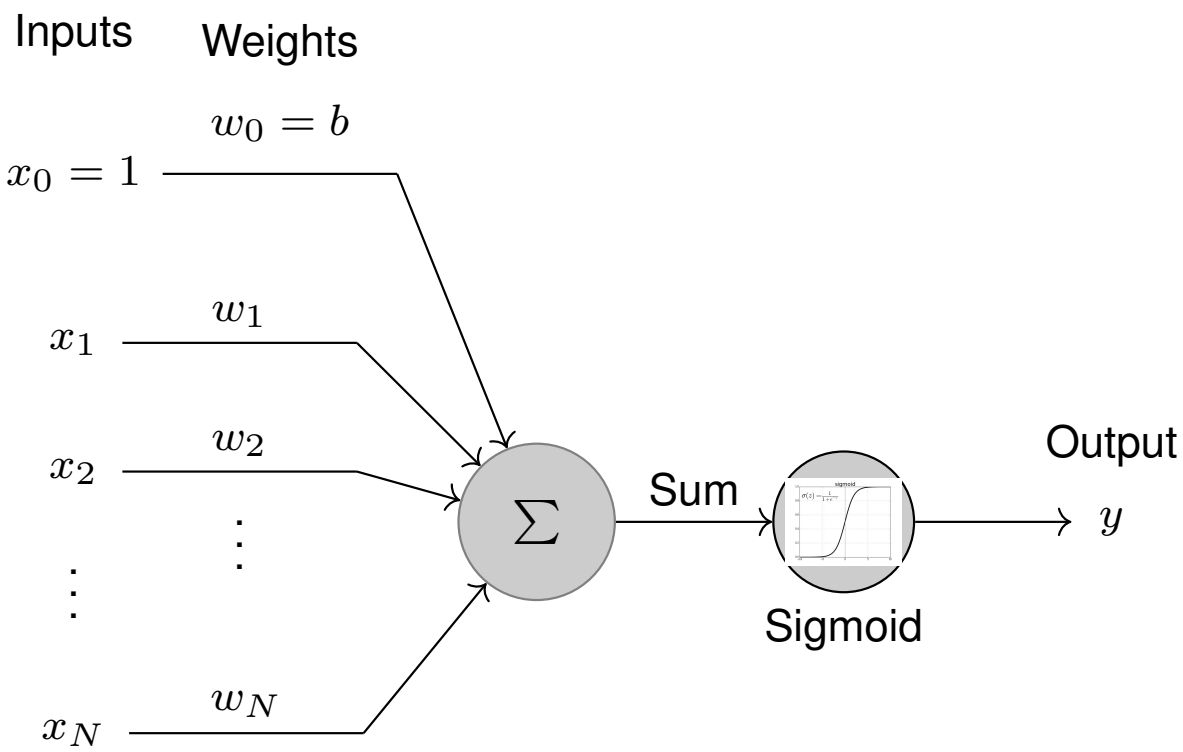
Today's agenda

Recap

Logistic regression

It is the perceptron with a sigmoid activation

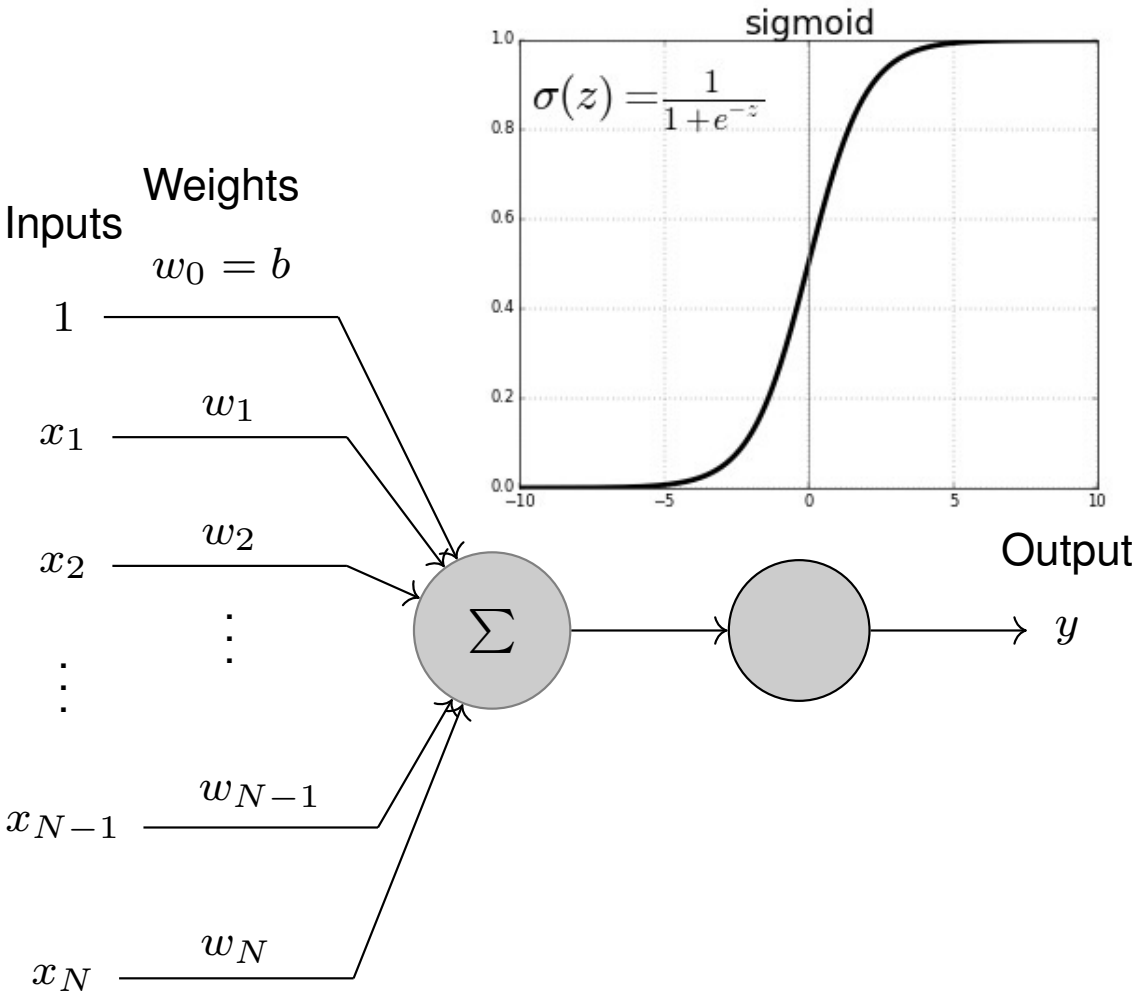
- It actually computes the probability that the input belongs to class 1



Logistic regression

It is the perceptron with a sigmoid activation

- It actually computes the probability that the input belongs to class 1



$$z = \sum_i w_i x_i$$

$$p(y = 1|x) = \sigma(z) = \frac{1}{1 + \exp(-z)}$$

$$y = \begin{cases} 1 & \text{if} \quad \sigma(z) \geq \frac{1}{2} \\ 0 & \text{if} \quad \sigma(z) < \frac{1}{2} \end{cases}$$

Vectorization

$$z = \sum_i w_i x_i = \boldsymbol{w}^T \boldsymbol{x} = \begin{bmatrix} w_0 = b & w_1 & w_2 & \dots & w_{N-1} & w_N \end{bmatrix} \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_{N-1} \\ x_N \end{bmatrix}$$

$$f(\boldsymbol{x}, \boldsymbol{w}) = \sigma(\boldsymbol{w}^T \boldsymbol{x}) = \frac{1}{1 + \exp(-\boldsymbol{w}^T \boldsymbol{x})}$$

Loss function

- Loss function quantifies our unhappiness with the scores across the training data.
- A loss function tells how good our current classifier is
  - Given a dataset of examples $\{(x^{(i)}, y^{(i)})\}_{i=1}^{N}$
  - $x^{(i)}$ is data and $y^{(i)}$ is binary label.
- Loss over the dataset is a average of loss over examples:

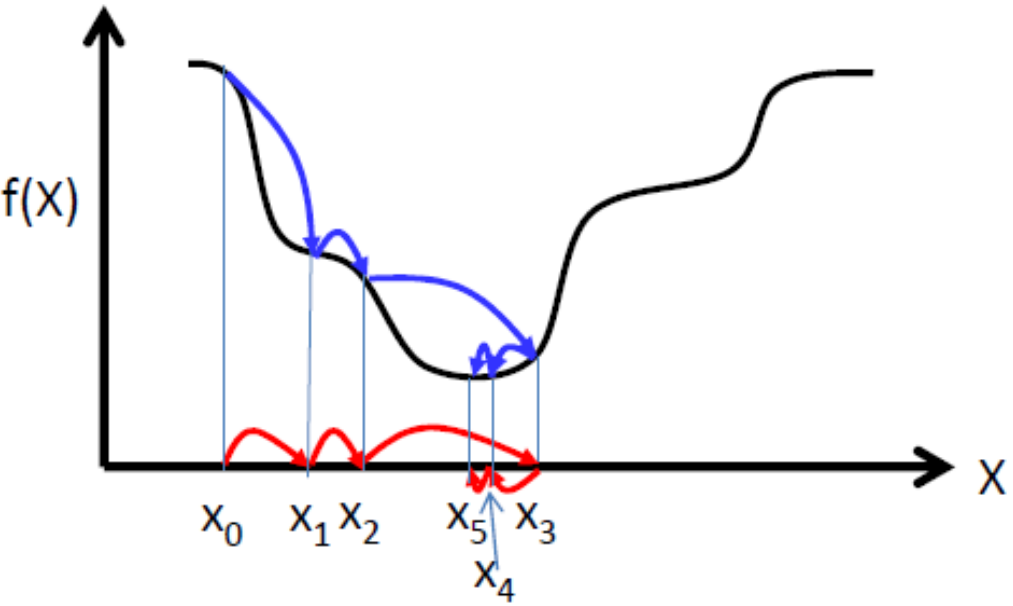
$$L(f(X, \boldsymbol{w}), \boldsymbol{y}) = \frac{1}{N} \sum_{i} L_i(f(\boldsymbol{x}^{(i)}, \boldsymbol{w}), y^{(i)})$$

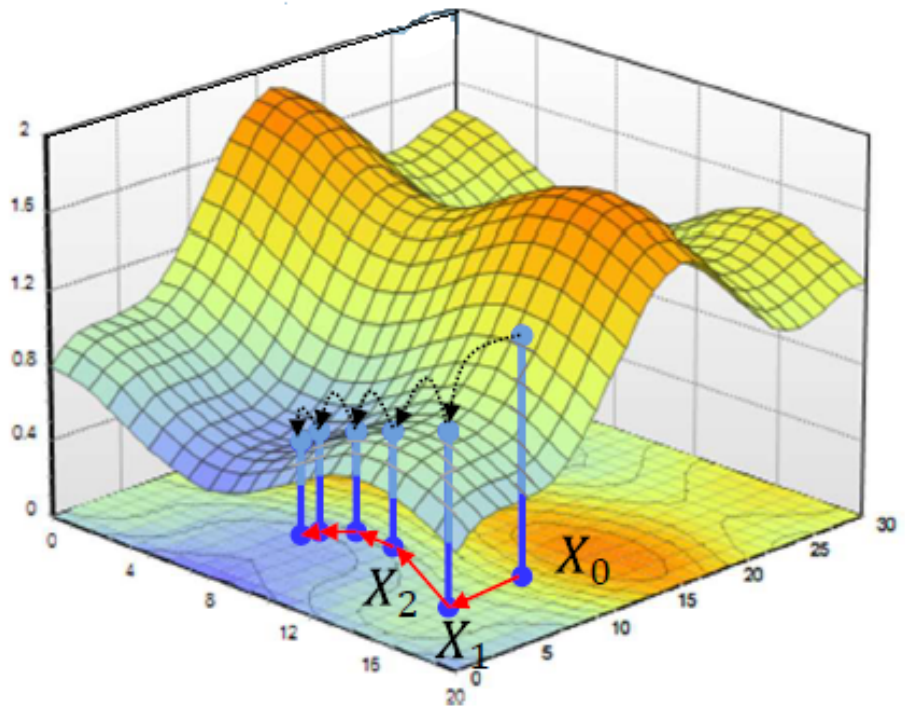
- Empirical loss L is only an empirical approximation of the true loss
- **Learning**
  - Estimate the parameters to minimize the empirical loss L

$$\boldsymbol{w}^* = \underset{\boldsymbol{w}}{\operatorname{argmin}}\ L(f(X, \boldsymbol{w}), \boldsymbol{y})$$

Iterative solutions (Gradient Descent)

- Iterative solutions
  - Start from an initial guess X_0 for the optimal X
  - Update the guess towards a (hopefully) "better" value of $f(X)$
  - Stop when $f(X)$ no longer decreases
- Problems
  - Which direction to step in
  - How big must the steps be





Gradient descent

- The gradient descent method to find the minimum of a function iteratively

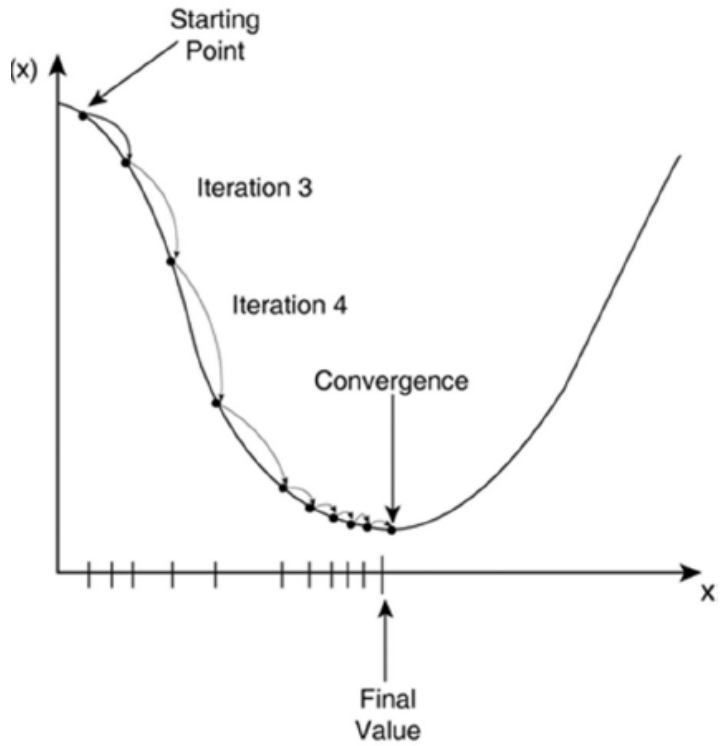
$$\boldsymbol{x}^{t+1} = \boldsymbol{x}^t - \eta \nabla_{\boldsymbol{x}} f(\boldsymbol{x}^t)$$

  - η is the "step size" (Also called "learning rate")

- The gradient descent algorithm converges when the following criterion is satisfied.

$$|f(\boldsymbol{x}^{t+k}) - f(\boldsymbol{x}^t)| < \epsilon$$

  - k is a hyperparameter.



Regularization

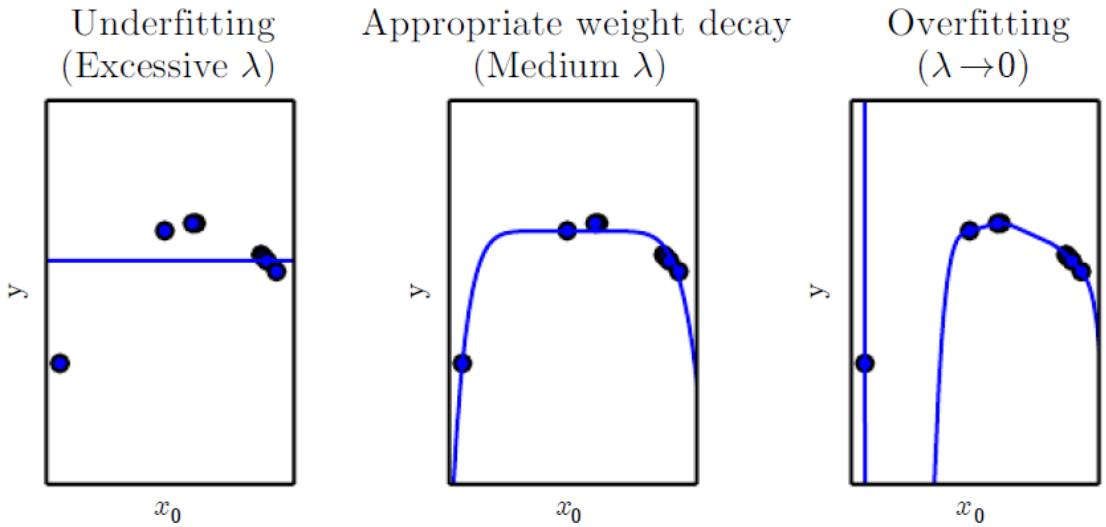
- **Regularization** is any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error.
  - We can give a learning algorithm a **preference** for one solution in its hypothesis space to another. This means that both functions are eligible, but one is preferred.

Regularization

- **Training loss:** Model predictions should match training data
- **Regularization**: Prevent the model from doing too well on training data $R(\boldsymbol{w})$

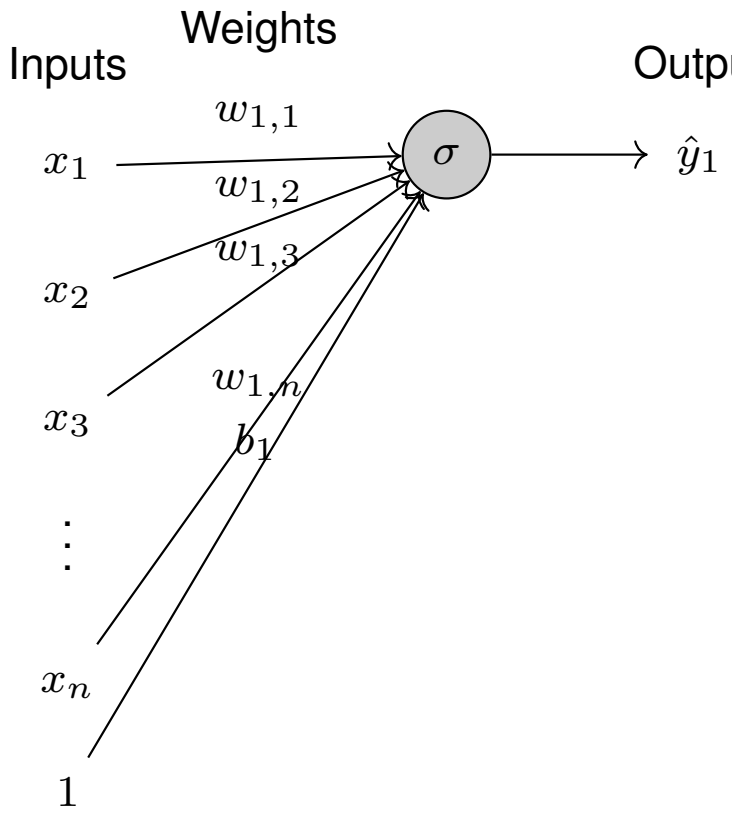
$$L(f(X, \boldsymbol{w}), \boldsymbol{y}) = \frac{1}{N} \sum_i L_i(f(\boldsymbol{x}^{(i)}, \boldsymbol{w}), y^{(i)}) + \lambda R(\boldsymbol{w})$$

- λ: regularization strength (hyperparameter)
- Simple examples
  - $L1$ regularization: $R(\boldsymbol{w}) = \sum_i |w_i|$
  - $L2$ regularization: $R(\boldsymbol{w}) = \sum_i w_i^2$



Softmax Classifier

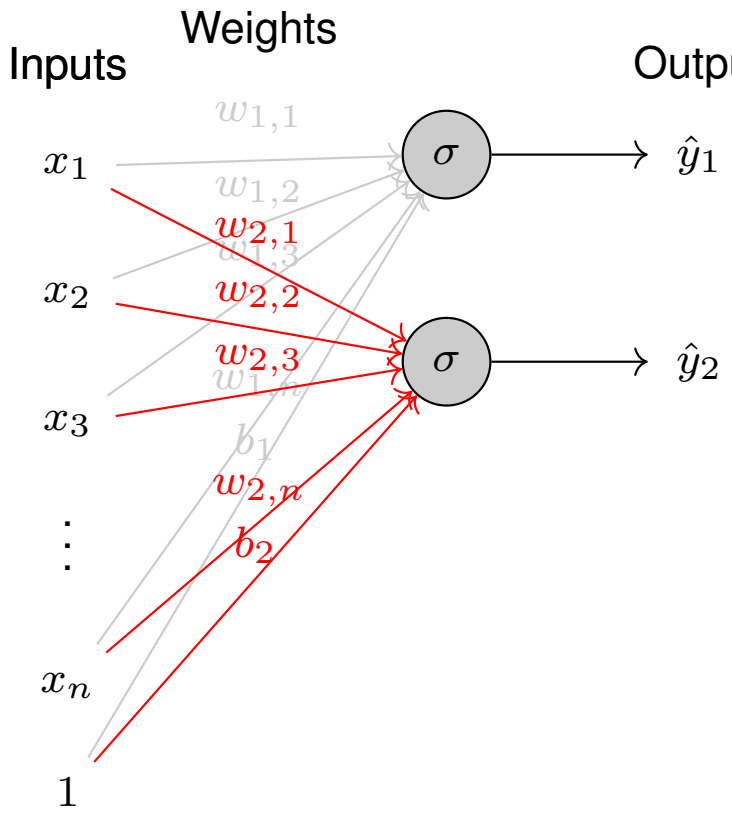
Multinomial logistic regression (softmax classifier)



$$z_1 = \sum_i w_{1,i} x_i + b_1 = \boldsymbol{w_1}^T \boldsymbol{x} + b_1, \quad \hat{y}_1 = \sigma(z_1)$$

$$z_1 = \begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} & \dots & w_{1,n} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} + b_1$$

Multinomial logistic regression (softmax classifier)

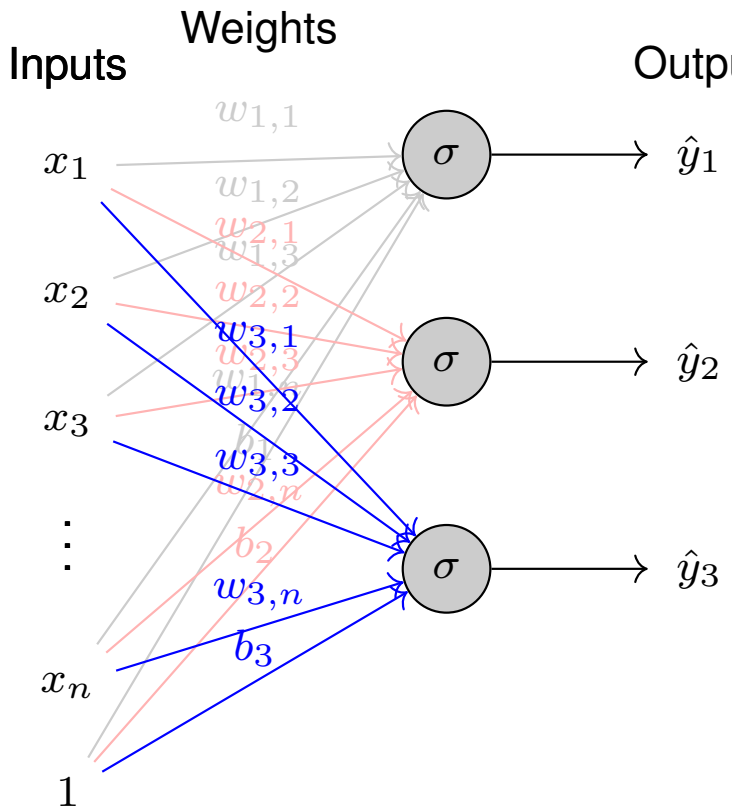


$$z_1 = \sum_i w_{1,i} x_i + b_1 = \boldsymbol{w_1}^T \boldsymbol{x} + b_1, \quad \hat{y}_1 = \sigma(z_1)$$

$$z_2 = \sum_i w_{2,i} x_i + b_2 = \boldsymbol{w_2}^T \boldsymbol{x} + b_2, \quad \hat{y}_2 = \sigma(z_2)$$

$$\begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} & \dots & w_{1,n} \\ w_{2,1} & w_{2,2} & w_{2,3} & \dots & w_{2,n} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

Multinomial logistic regression (softmax classifier)



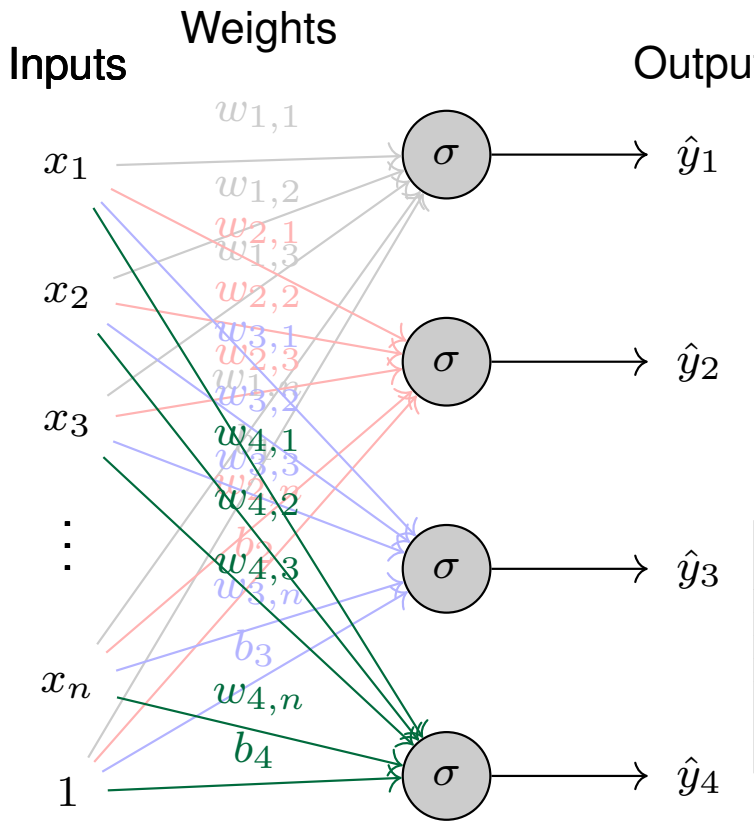
$$z_1 = \sum_i w_{1,i} x_i + b_1 = \boldsymbol{w_1}^T \boldsymbol{x} + b_1, \quad \hat{y}_1 = \sigma(z_1)$$

$$z_2 = \sum_i w_{2,i} x_i + b_2 = \boldsymbol{w_2}^T \boldsymbol{x} + b_2, \quad \hat{y}_2 = \sigma(z_2)$$

$$z_3 = \sum_i w_{3,i} x_i + b_3 = \boldsymbol{w_3}^T \boldsymbol{x} + b_3, \quad \hat{y}_3 = \sigma(z_3)$$

$$\begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} = \begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} & \dots & w_{1,n} \\ w_{2,1} & w_{2,2} & w_{2,3} & \dots & w_{2,n} \\ w_{3,1} & w_{3,2} & w_{3,3} & \dots & w_{3,n} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

Multinomial logistic regression (softmax classifier)



$$z_1 = \sum_i w_{1,i} x_i + b_1 = \boldsymbol{w_1}^T \boldsymbol{x} + b_1, \quad \hat{y}_1 = \sigma(z_1)$$

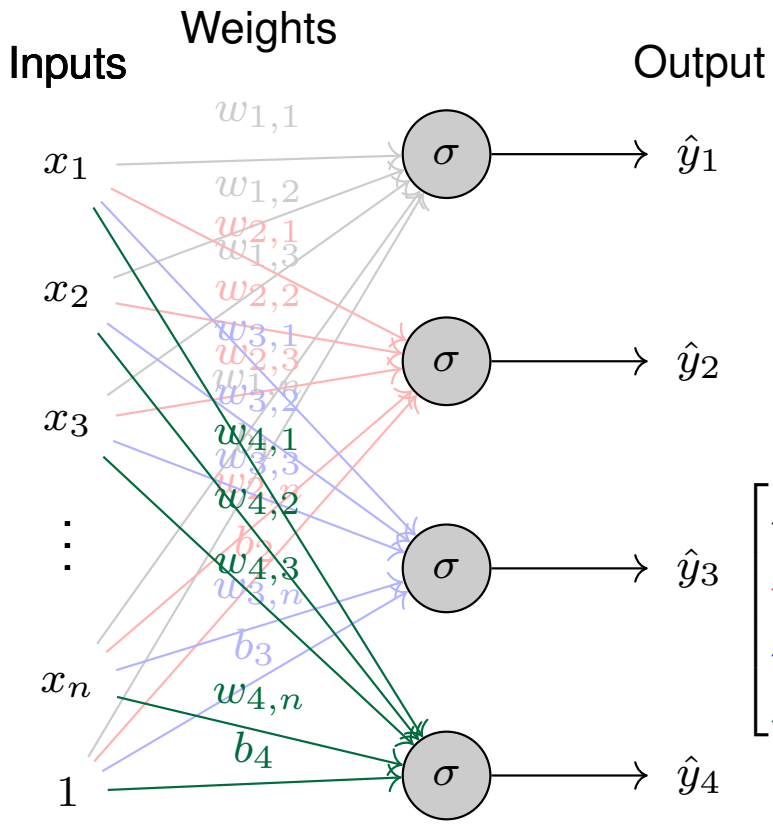
$$z_2 = \sum_i w_{2,i} x_i + b_2 = \boldsymbol{w_2}^T \boldsymbol{x} + b_2, \quad \hat{y}_2 = \sigma(z_2)$$

$$z_3 = \sum_i w_{3,i} x_i + b_3 = \boldsymbol{w_3}^T \boldsymbol{x} + b_3, \quad \hat{y}_3 = \sigma(z_3)$$

$$z_4 = \sum_i w_{4,i} x_i + b_4 = \boldsymbol{w_4}^T \boldsymbol{x} + b_4, \quad \hat{y}_4 = \sigma(z_4)$$

$$\begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \end{bmatrix} = \begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} & \cdots & w_{1,n} \\ w_{2,1} & w_{2,2} & w_{2,3} & \cdots & w_{2,n} \\ w_{3,1} & w_{3,2} & w_{3,3} & \cdots & w_{3,n} \\ w_{4,1} & w_{4,2} & w_{4,3} & \cdots & w_{4,n} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}$$

Multinomial logistic regression (softmax classifier)



$$z_1 = \sum_i w_{1,i} x_i + b_1 = \boldsymbol{w_1}^T \boldsymbol{x} + b_1, \quad \hat{y}_1 = \sigma(z_1)$$

$$z_2 = \sum_i w_{2,i} x_i + b_2 = \boldsymbol{w_2}^T \boldsymbol{x} + b_2, \quad \hat{y}_2 = \sigma(z_2)$$

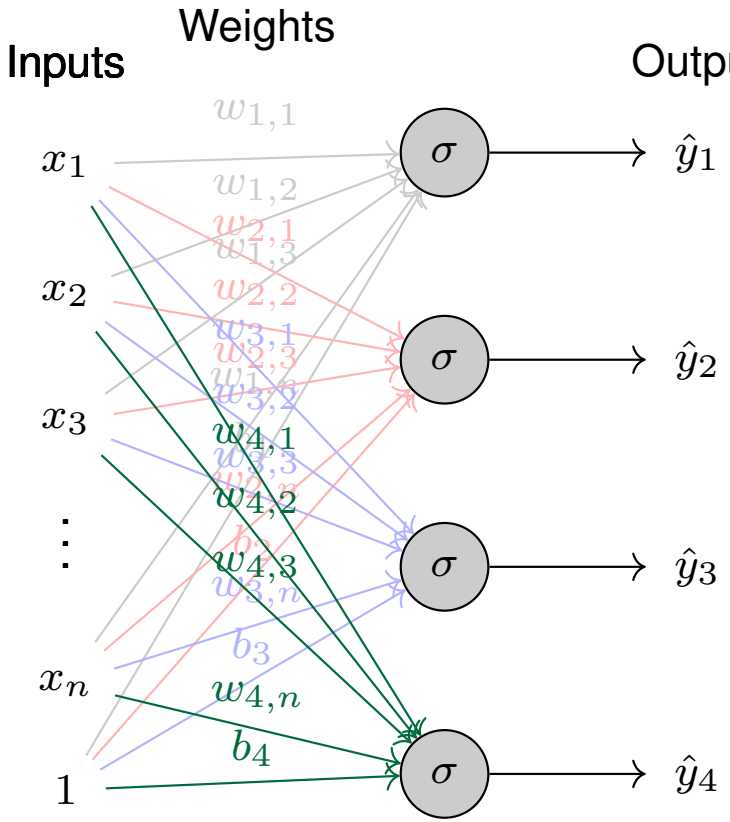
$$z_3 = \sum_i w_{3,i} x_i + b_3 = \boldsymbol{w_3}^T \boldsymbol{x} + b_3, \quad \hat{y}_3 = \sigma(z_3)$$

$$z_4 = \sum_i w_{4,i} x_i + b_4 = \boldsymbol{w_4}^T \boldsymbol{x} + b_4, \quad \hat{y}_4 = \sigma(z_4)$$

$$\begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \end{bmatrix} = \begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} & \cdots & w_{1,n} \\ w_{2,1} & w_{2,2} & w_{2,3} & \cdots & w_{2,n} \\ w_{3,1} & w_{3,2} & w_{3,3} & \cdots & w_{3,n} \\ w_{4,1} & w_{4,2} & w_{4,3} & \cdots & w_{4,n} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}$$

$$\boldsymbol{z} = W^T \boldsymbol{x} + \boldsymbol{b}$$

Multinomial logistic regression (softmax classifier)



$$\begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \end{bmatrix} = \begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} & \cdots & w_{1,n} \\ w_{2,1} & w_{2,2} & w_{2,3} & \cdots & w_{2,n} \\ w_{3,1} & w_{3,2} & w_{3,3} & \cdots & w_{3,n} \\ w_{4,1} & w_{4,2} & w_{4,3} & \cdots & w_{4,n} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}$$

The output vector: $\begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \hat{y}_3 \\ \hat{y}_4 \end{bmatrix} = \sigma\left(\begin{bmatrix} \hat{z}_1 \\ \hat{z}_2 \\ \hat{z}_3 \\ \hat{z}_4 \end{bmatrix} \right)$

- The output vector is not a probability vector (Nonzero and sum of elements must be one)
  - $\hat{y}_i \in [0, 1]$
  - Cross entropy is fed by a probability vector

Multi-class output: One-hot representations

- The labels are 0 and 1 in logistic regression
- Consider a network that must distinguish if an input is a cat, a dog, camel, a hat, or a flower
- For inputs of each of the four classes the desired output is a probability vector:

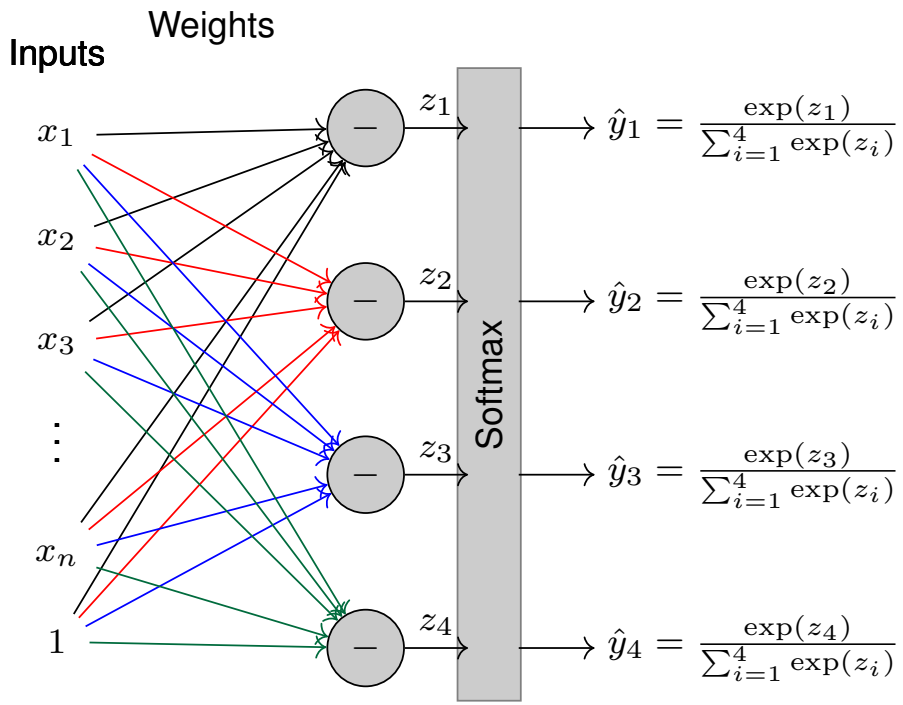
  Cat: $\begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}^T$

  Dog: $\begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix}^T$

  Fish: $\begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix}^T$

  Frog: $\begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}^T$
- For input of any class, we will have a four-dimensional vector output with four zeros and a single 1 at the position of the class
- This is a one hot vector

Softmax classifier



Softmax function:

$$P(\hat{y} = j|\boldsymbol{x}, W) = \frac{\exp(z_j)}{\sum_{i=1}^{K}\exp(z_i)} = \hat{y}_j$$

The output vector: $\hat{\boldsymbol{y}} = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \hat{y}_3 \\ \hat{y}_4 \end{bmatrix}$

The output of classifier: $\underset{j}{\text{argmax}}\ \hat{y}_j$

Loss function

- Cross entropy:

$$L = CE(\boldsymbol{y}, \hat{\boldsymbol{y}}) = -\sum_{i=1}^{K} y_i \log \hat{y}_i$$

- $\boldsymbol{y}$ is a one-hot vector
- If the true class of $\boldsymbol{x}$ is j, only the the j^{th} element of $\boldsymbol{y}$ is one, and the others are zero. Therefore:

$$L = CE(\boldsymbol{y}, \hat{\boldsymbol{y}}) = -\log \hat{y}_j$$

Loss function

- Cross entropy:

$$L = CE(\boldsymbol{y}, \hat{\boldsymbol{y}}) = -\sum_{i=1}^{K} y_i \log \hat{y}_i$$

- $\boldsymbol{y}$ is a one-hot vector
- If the true class of $\boldsymbol{x}$ is j, only the the j^{th} element of $\boldsymbol{y}$ is one, and the others are zero. Therefore:

$$L = CE(\boldsymbol{y}, \hat{\boldsymbol{y}}) = -\log \hat{y}_j$$

example:

$$\boldsymbol{y} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} \begin{matrix} dog \\ cat \\ fish \\ frog \end{matrix} \quad \boldsymbol{z} = \begin{pmatrix} 3.2 \\ 5.1 \\ -1.7 \\ -3.7 \end{pmatrix} \quad softmax(\boldsymbol{z}) = \hat{\boldsymbol{y}} = \begin{pmatrix} 0.13 \\ 0.87 \\ 0 \\ 0 \end{pmatrix}$$

Loss function

- Cross entropy:

$$L = CE(\boldsymbol{y}, \hat{\boldsymbol{y}}) = -\sum_{i=1}^{K} y_i \log \hat{y}_i$$

- $\boldsymbol{y}$ is a one-hot vector
- If the true class of $\boldsymbol{x}$ is j, only the the j^{th} element of $\boldsymbol{y}$ is one, and the others are zero. Therefore:

$$L = CE(\boldsymbol{y}, \hat{\boldsymbol{y}}) = -\log \hat{y}_j$$

example:

$$\boldsymbol{y} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} \begin{matrix} dog \\ cat \\ fish \\ frog \end{matrix} \quad \boldsymbol{z} = \begin{pmatrix} 3.2 \\ 5.1 \\ -1.7 \\ -3.7 \end{pmatrix} \quad softmax(\boldsymbol{z}) = \hat{\boldsymbol{y}} = \begin{pmatrix} 0.13 \\ 0.87 \\ 0 \\ 0 \end{pmatrix}$$

$$L = CE(\boldsymbol{y}, \hat{\boldsymbol{y}}) = -\log \hat{y}_2 = -\log 0.87 = 0.14$$

Loss function

- Cross entropy:

$$L = CE(\boldsymbol{y}, \hat{\boldsymbol{y}}) = -\sum_{i=1}^{K} y_i \log \hat{y}_i$$

- $\boldsymbol{y}$ is a one-hot vector
- If the true class of $\boldsymbol{x}$ is j, only the the j^{th} element of $\boldsymbol{y}$ is one, and the others are zero. Therefore:

$$L = CE(\boldsymbol{y}, \hat{\boldsymbol{y}}) = -\log \hat{y}_j$$

example:

$$\boldsymbol{y} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} \begin{matrix} dog \\ cat \\ fish \\ frog \end{matrix} \quad \boldsymbol{z} = \begin{pmatrix} 5.8 \\ 2.2 \\ -1.6 \\ 1.5 \end{pmatrix} \quad softmax(\boldsymbol{z}) = \hat{\boldsymbol{y}} = \begin{pmatrix} 0.96 \\ 0.03 \\ 0 \\ 0.01 \end{pmatrix}$$

Loss function

- Cross entropy:

$$L = CE(\boldsymbol{y}, \hat{\boldsymbol{y}}) = -\sum_{i=1}^{K} y_i \log \hat{y}_i$$

- $\boldsymbol{y}$ is a one-hot vector
- If the true class of $\boldsymbol{x}$ is j, only the the j^{th} element of $\boldsymbol{y}$ is one, and the others are zero. Therefore:

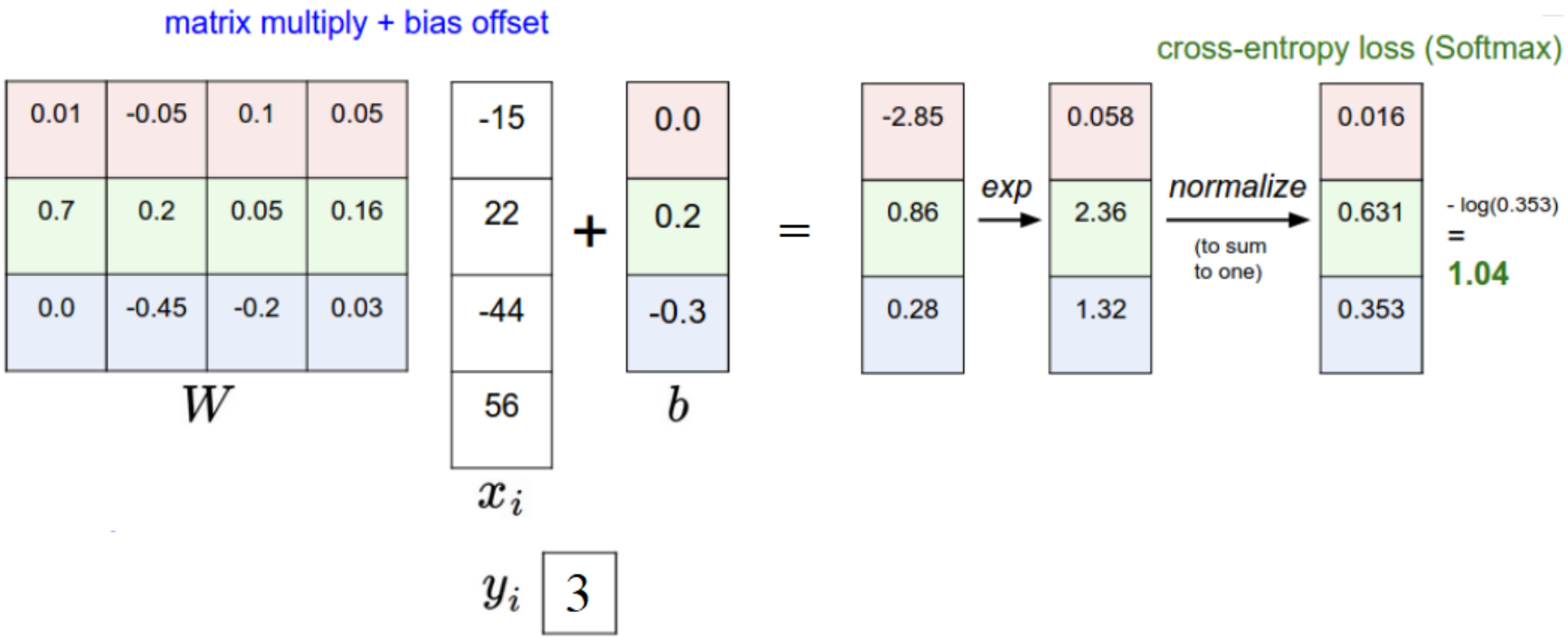
$$L = CE(\boldsymbol{y}, \hat{\boldsymbol{y}}) = -\log \hat{y}_j$$

example:

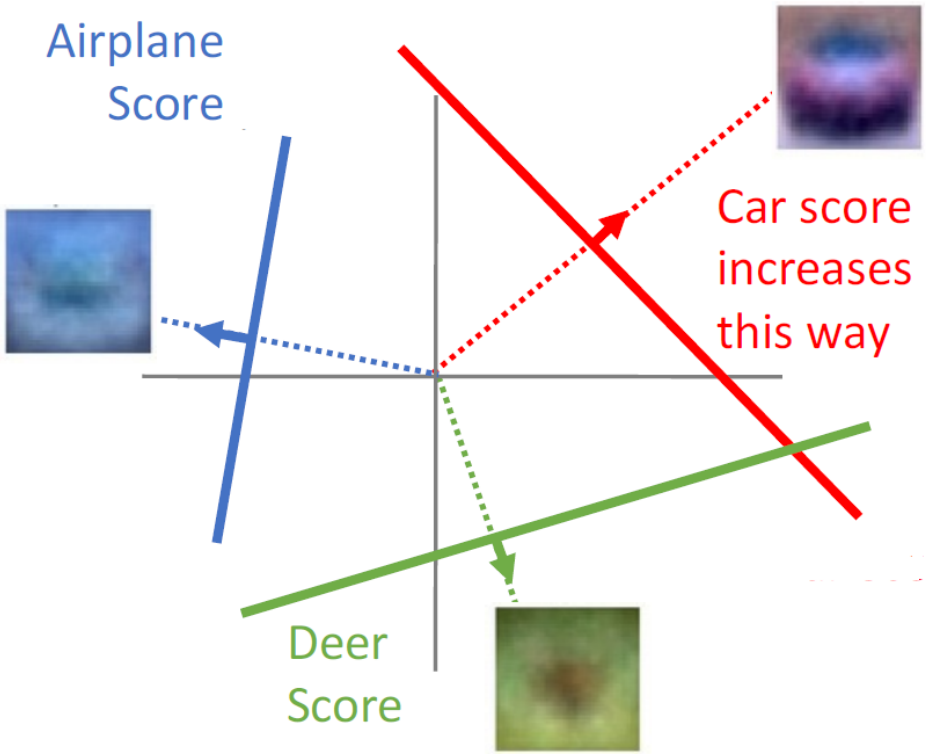
$$\boldsymbol{y} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} \begin{matrix} dog \\ cat \\ fish \\ frog \end{matrix} \quad \boldsymbol{z} = \begin{pmatrix} 5.8 \\ 2.2 \\ -1.6 \\ 1.5 \end{pmatrix} \quad softmax(\boldsymbol{z}) = \hat{\boldsymbol{y}} = \begin{pmatrix} 0.96 \\ 0.03 \\ 0 \\ 0.01 \end{pmatrix}$$

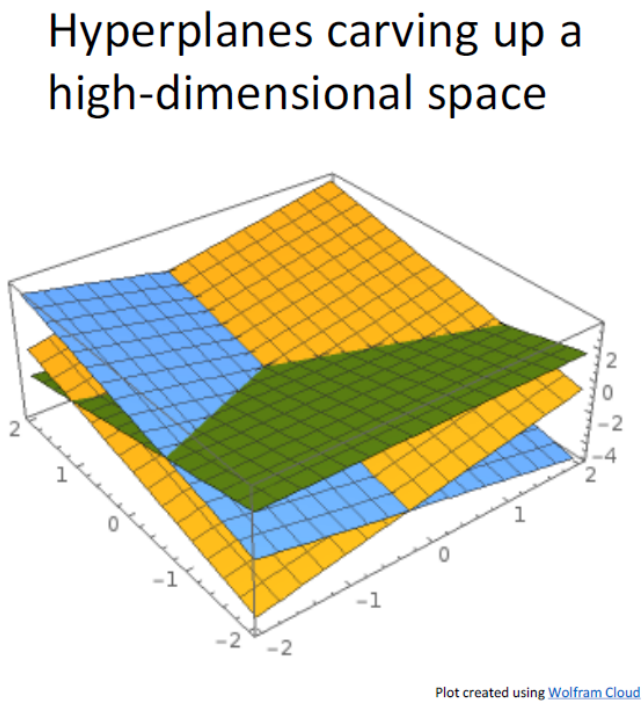
$$L = CE(\boldsymbol{y}, \hat{\boldsymbol{y}}) = -\log \hat{y}_2 = -\log 0.03 = 3.5$$

Example


matrix multiply + bias offset
cross-entropy loss (Softmax)
0.01 -0.05 0.1 0.05
0.7 0.2 0.05 0.16
0.0 -0.45 -0.2 0.03
W
-15
22
-44
56
x_i
+
0.0
0.2
-0.3
b
=
-2.85
0.86
0.28
exp
0.058
2.36
1.32
normalize
(to sum to one)
0.016
0.631
0.353
- log(0.353) = 1.04
y_i 3

Softmax classifier



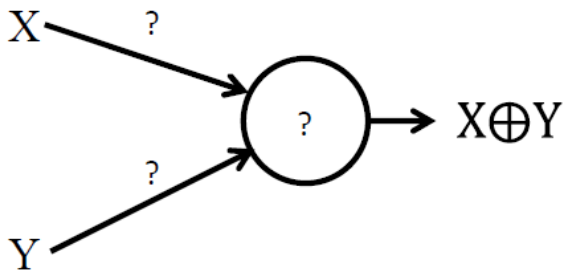


http://vision.stanford.edu/teaching/cs231n-demos/linear-classify/

Deep Neural Networks

Xor

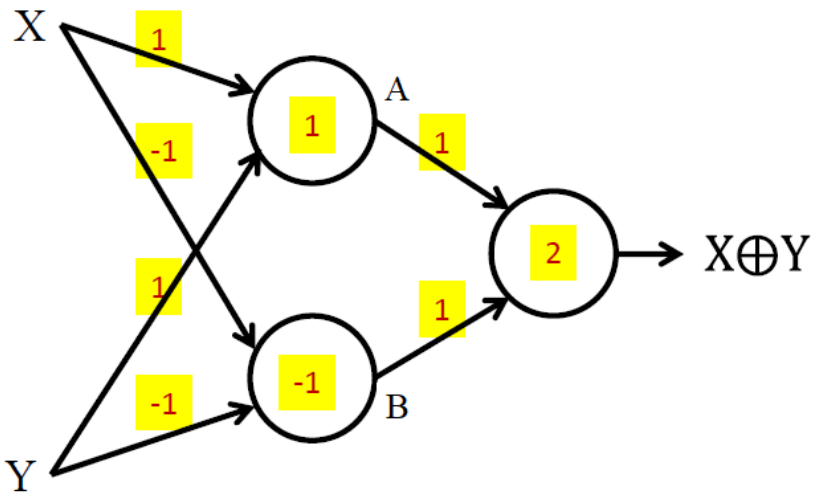
No solution for XOR!
Not universal!



Values shown on edges are weights, numbers in the circles are thresholds

Solution: add layers

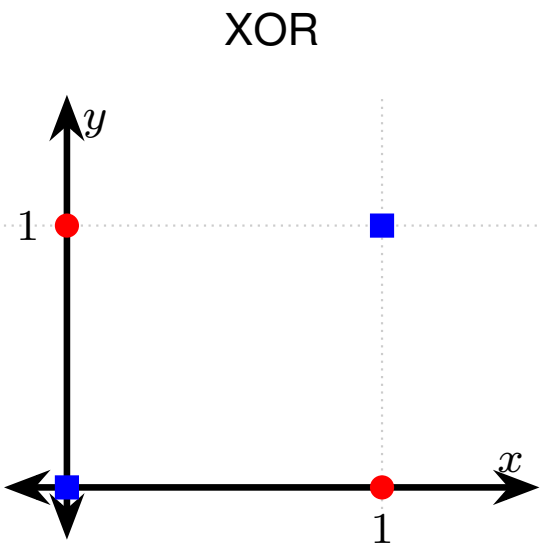
XOR

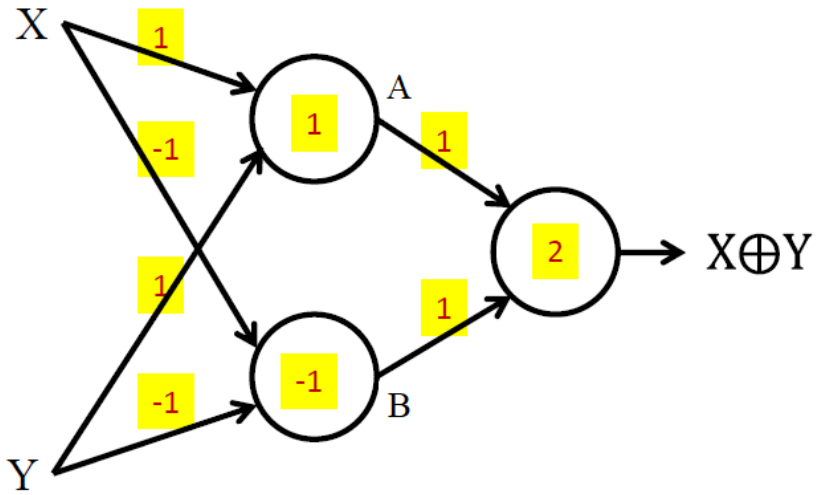


X	Y	A	B	Label
0	0	0	1	0
0	1	1	1	1
1	0	1	1	1
1	1	1	0	0

Transformation

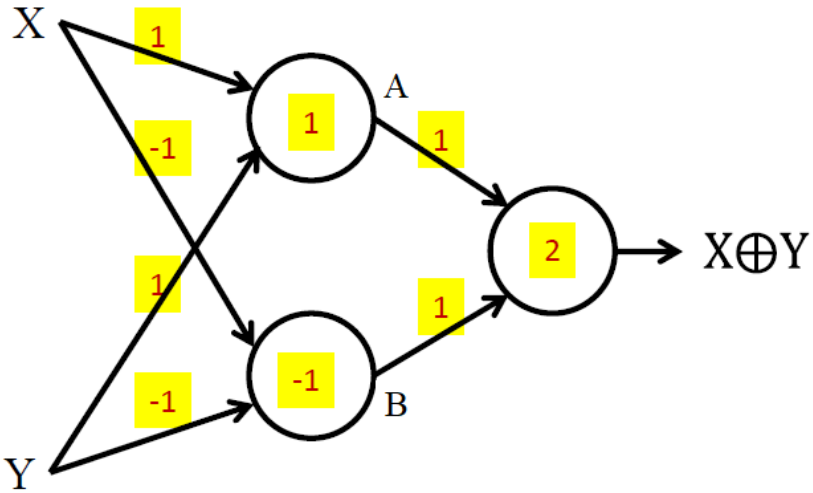
X	Y	A	B	Label
0	0	0	1	0
0	1	1	1	1
1	0	1	1	1
1	1	1	0	0

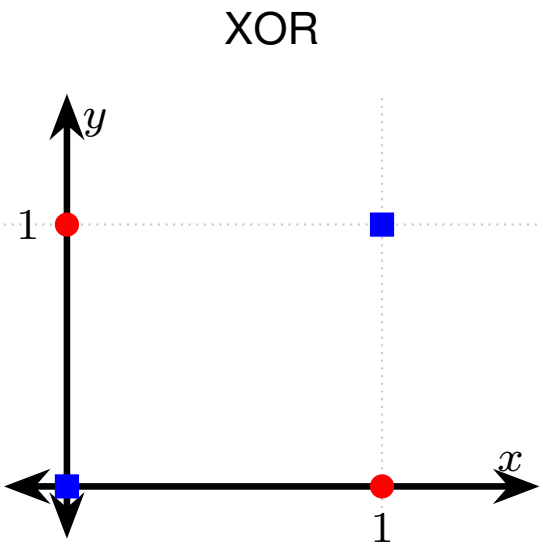


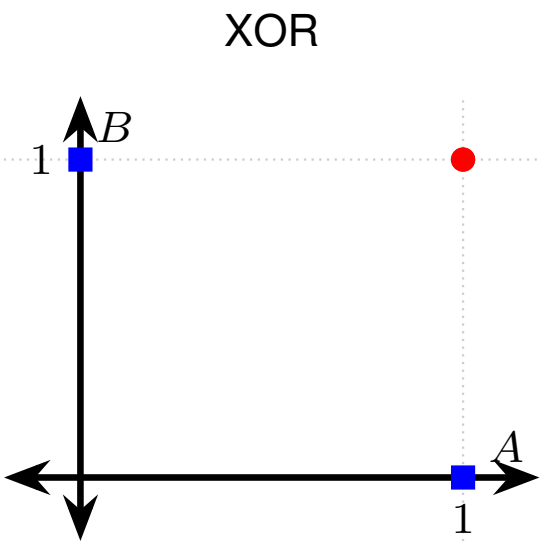


Transformation

X	Y	A	B	Label
0	0	0	1	0
0	1	1	1	1
1	0	1	1	1
1	1	1	0	0

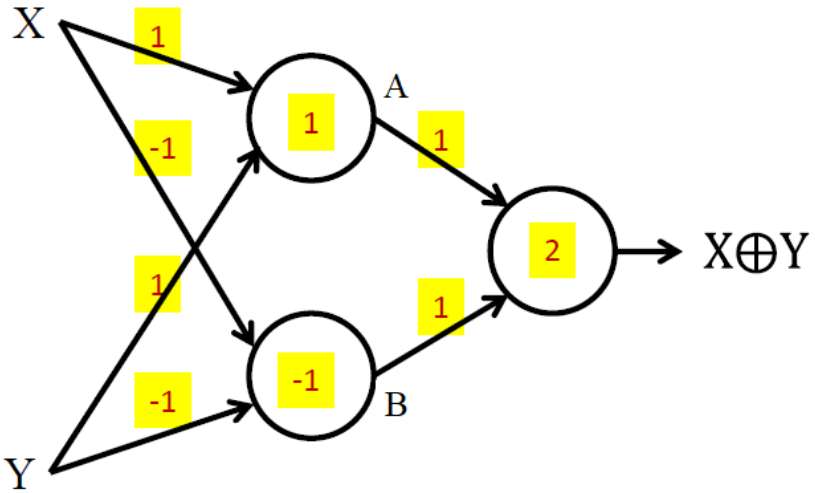


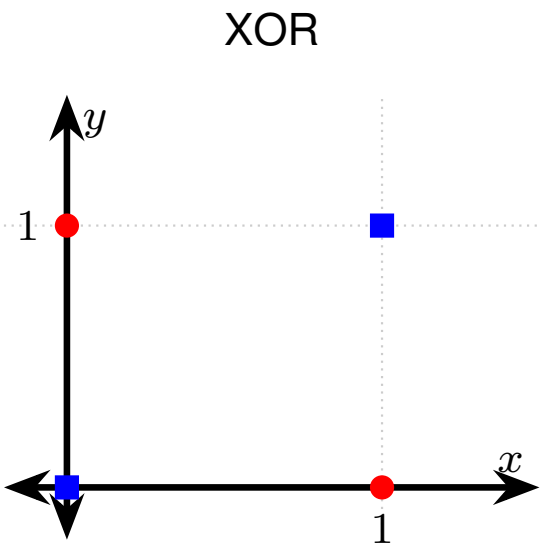


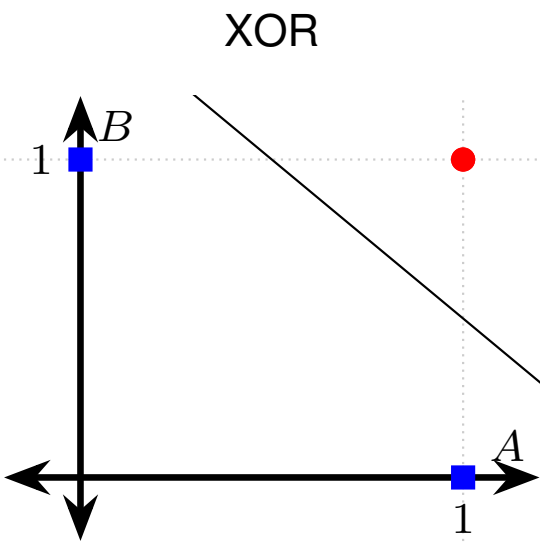


Transformation

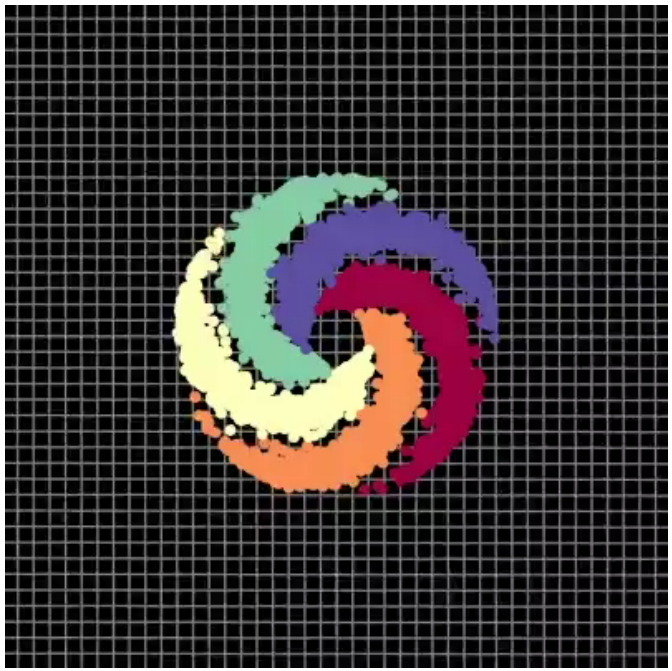
X	Y	A	B	Label
0	0	0	1	0
0	1	1	1	1
1	0	1	1	1
1	1	1	0	0





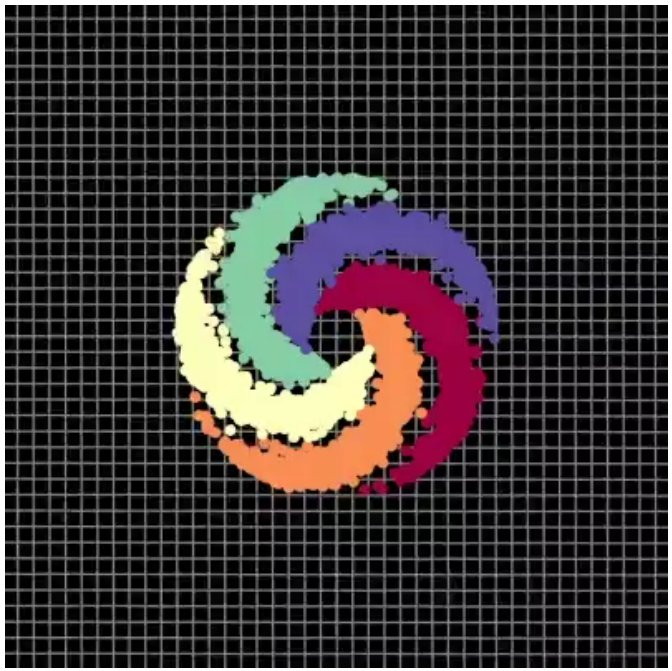


Needs more layers

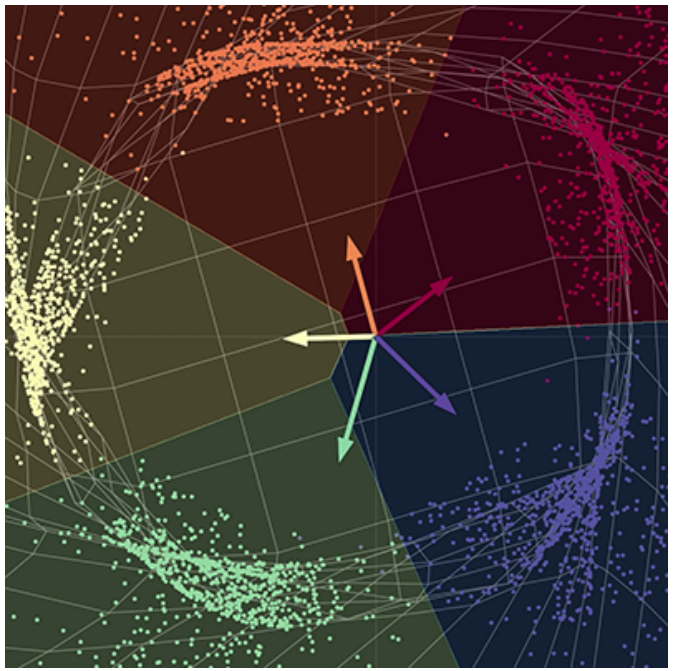


Input points, pre-network

Needs more layers



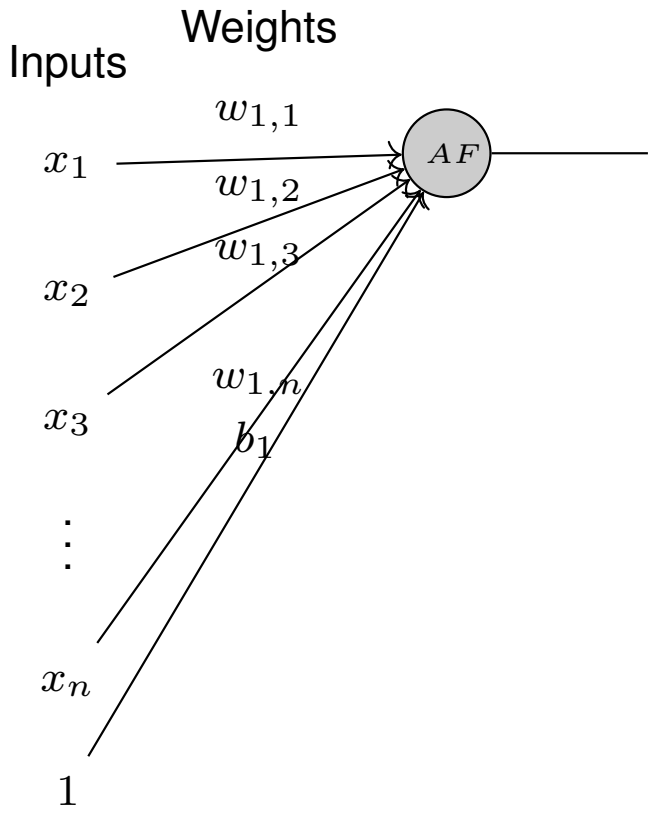
Input points, pre-network



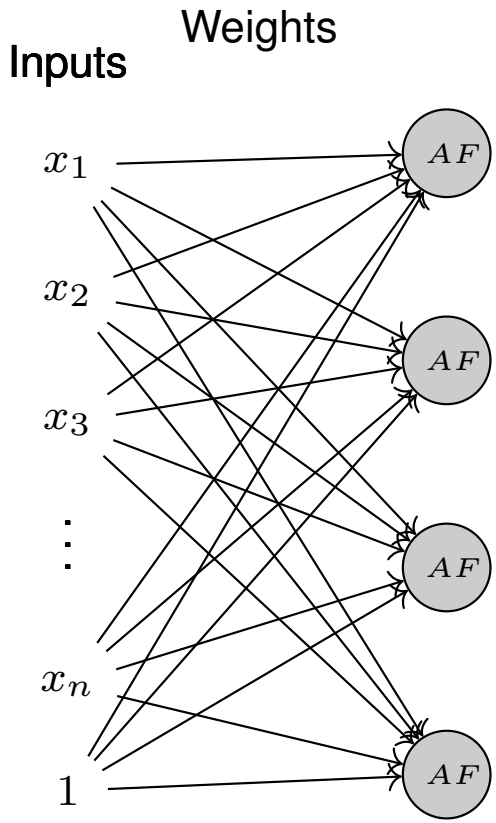
Output points, post-network

(Canziani, 2020)

Fully connected neural networks

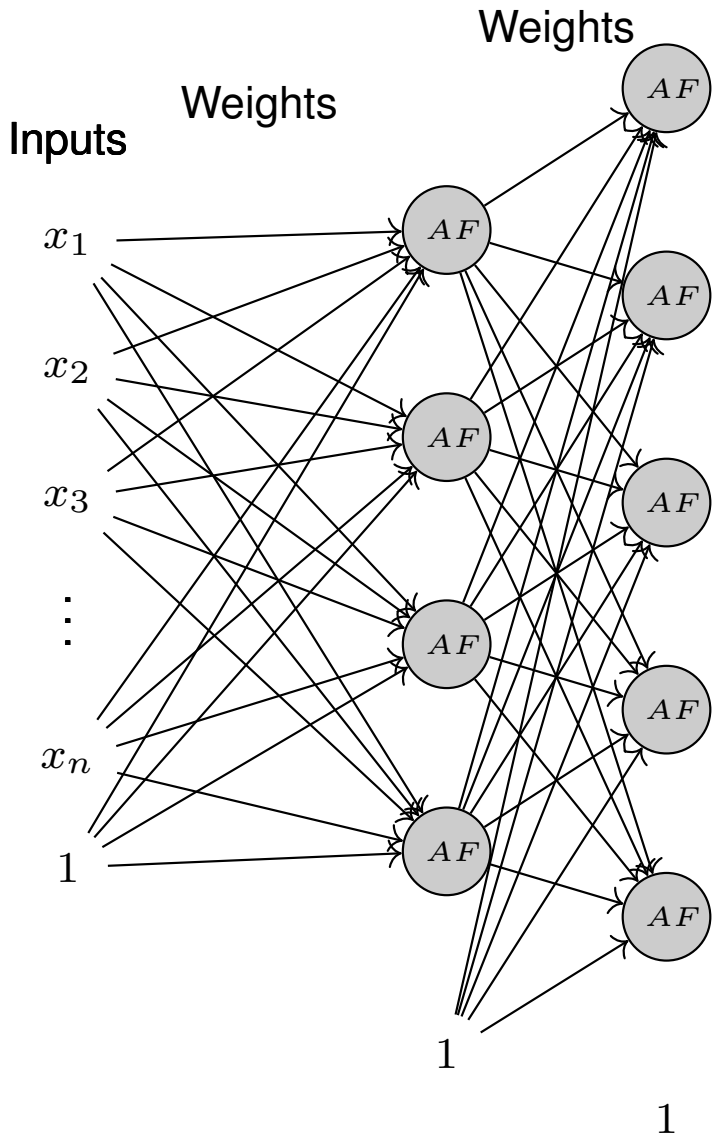


Fully connected neural networks

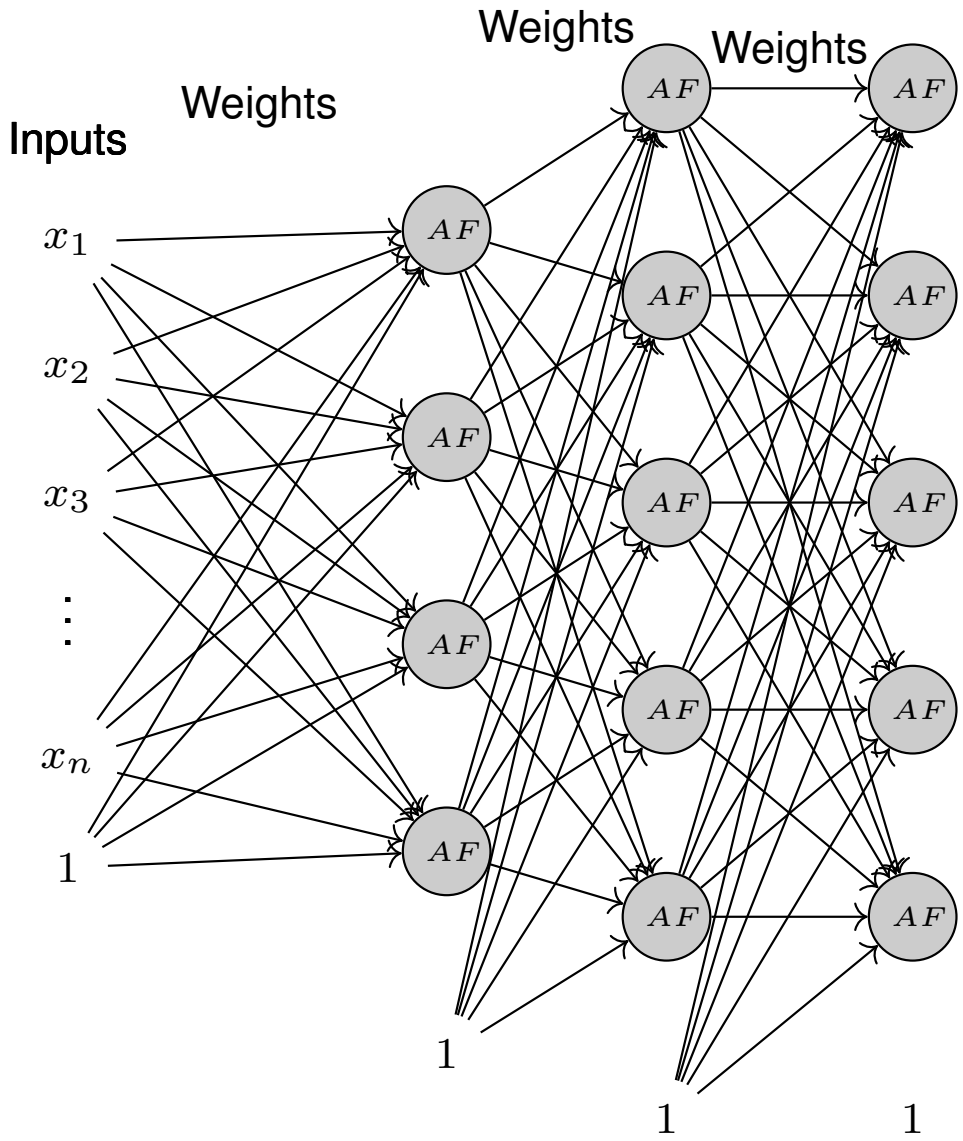


1

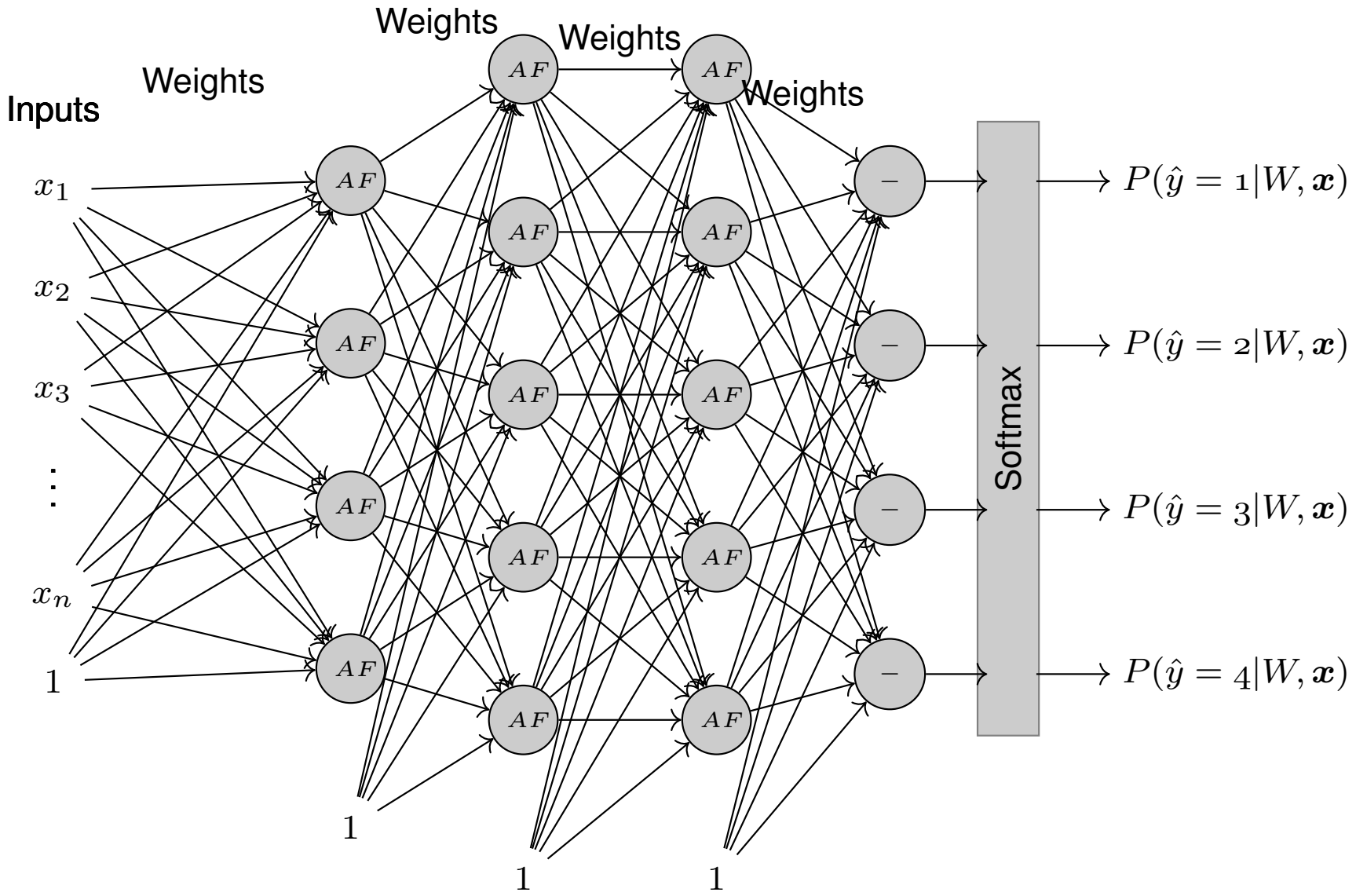
Fully connected neural networks



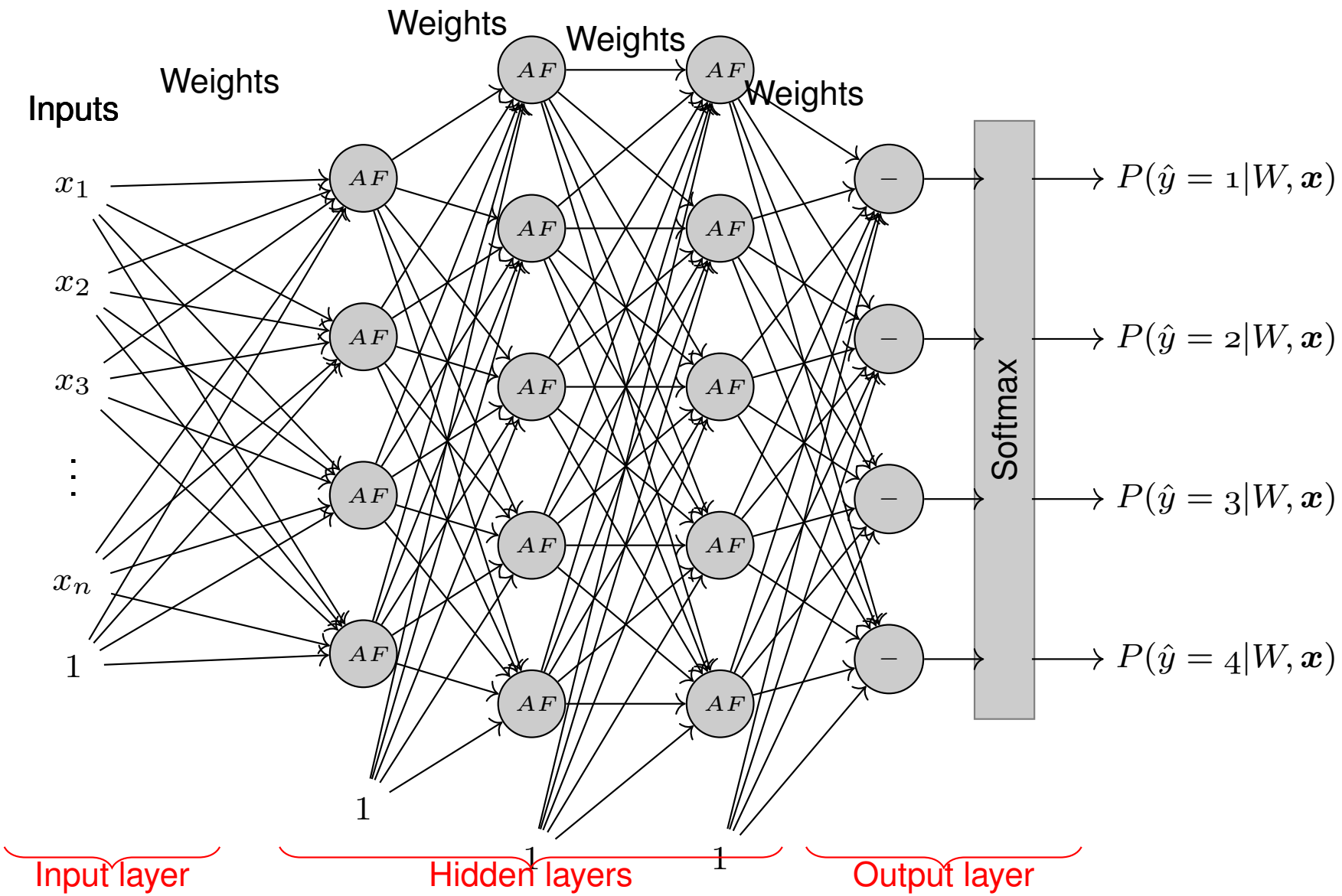
Fully connected neural networks



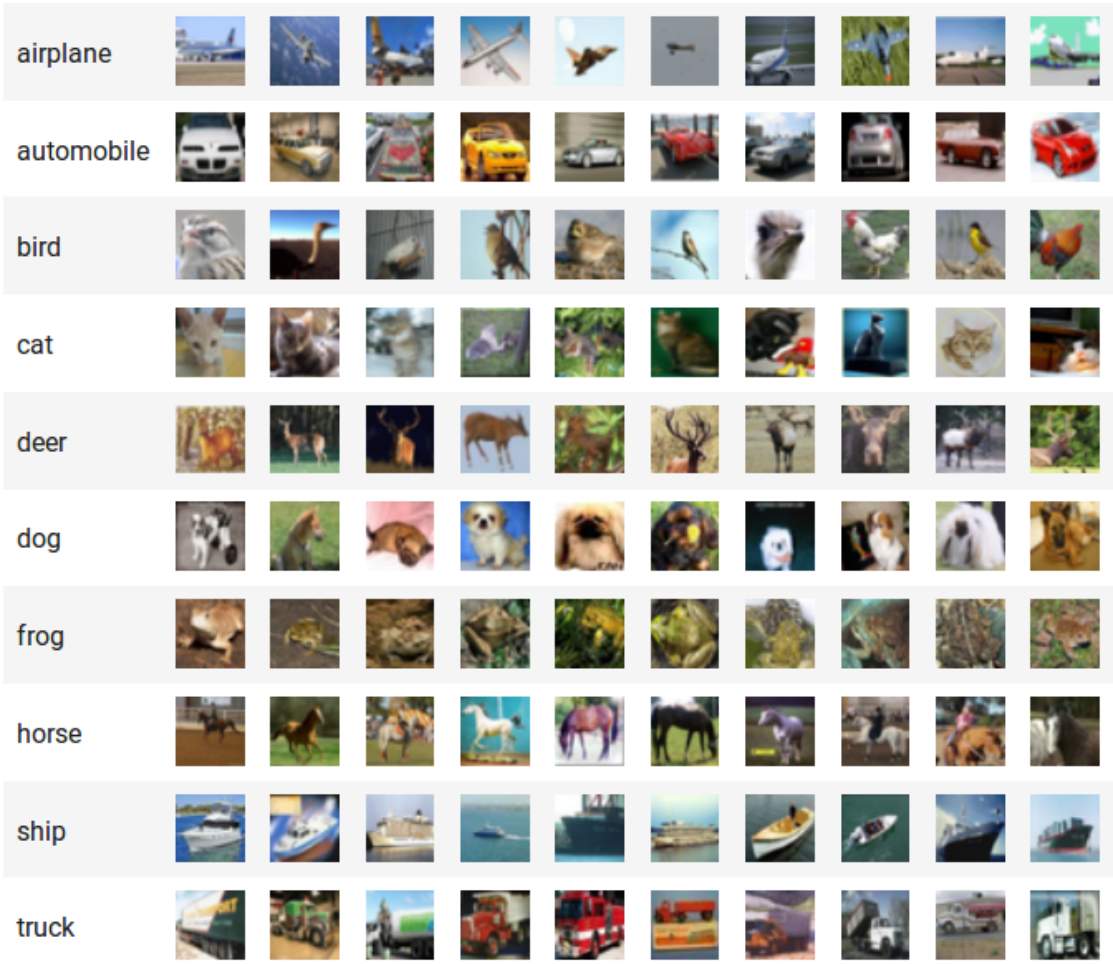
Fully connected neural networks



Fully connected neural networks

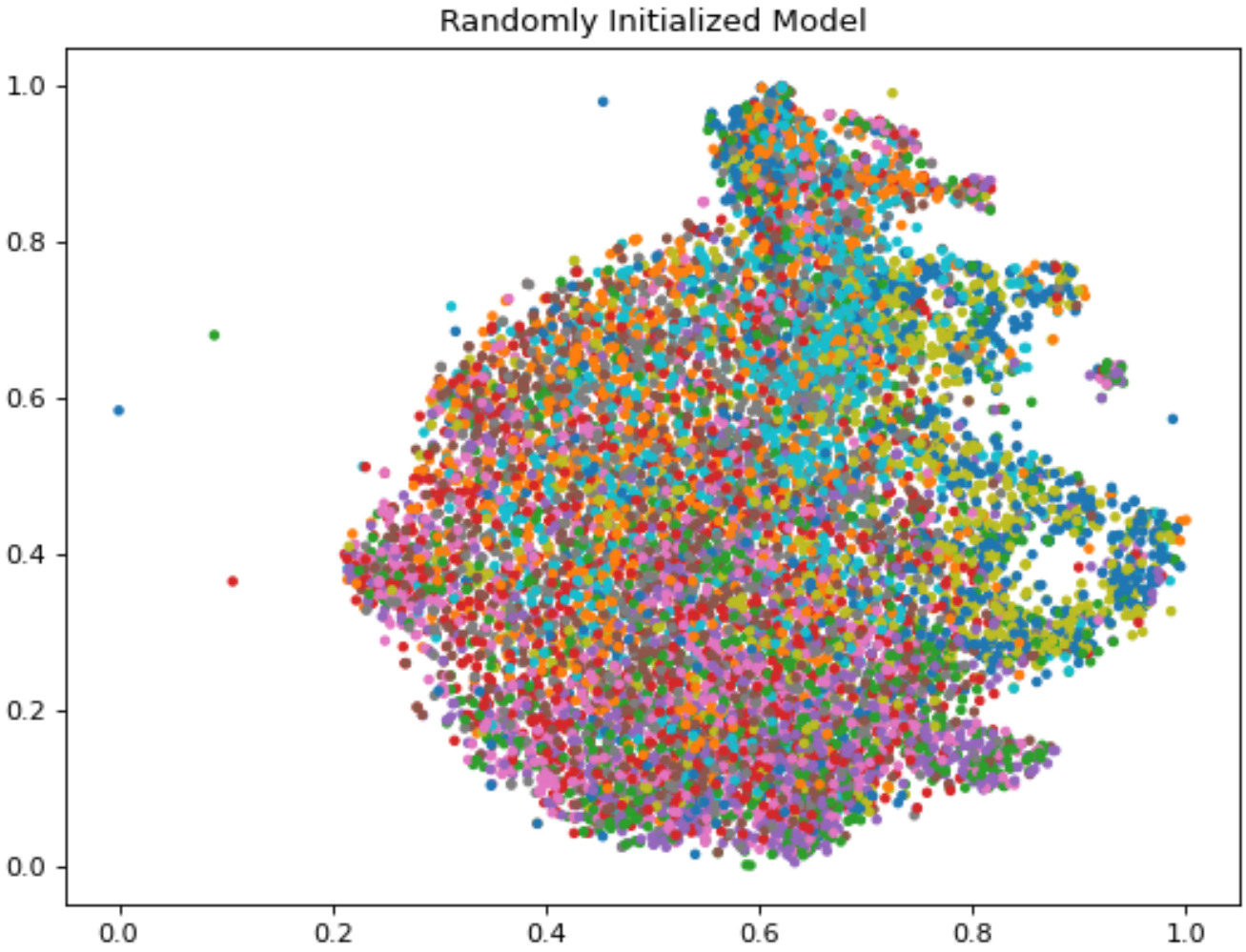


CIFAR10

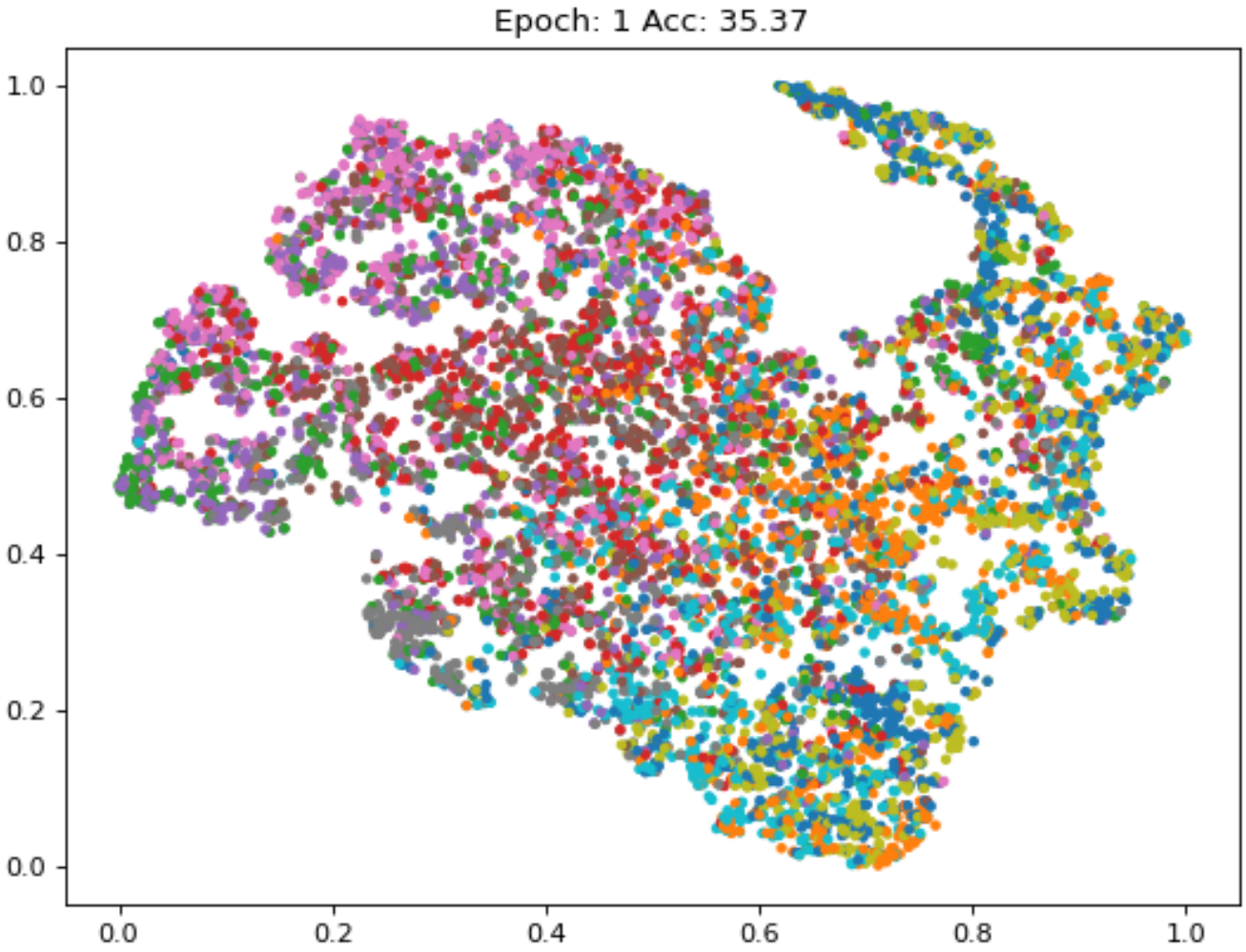


Example images from the CIFAR-10 dataset.

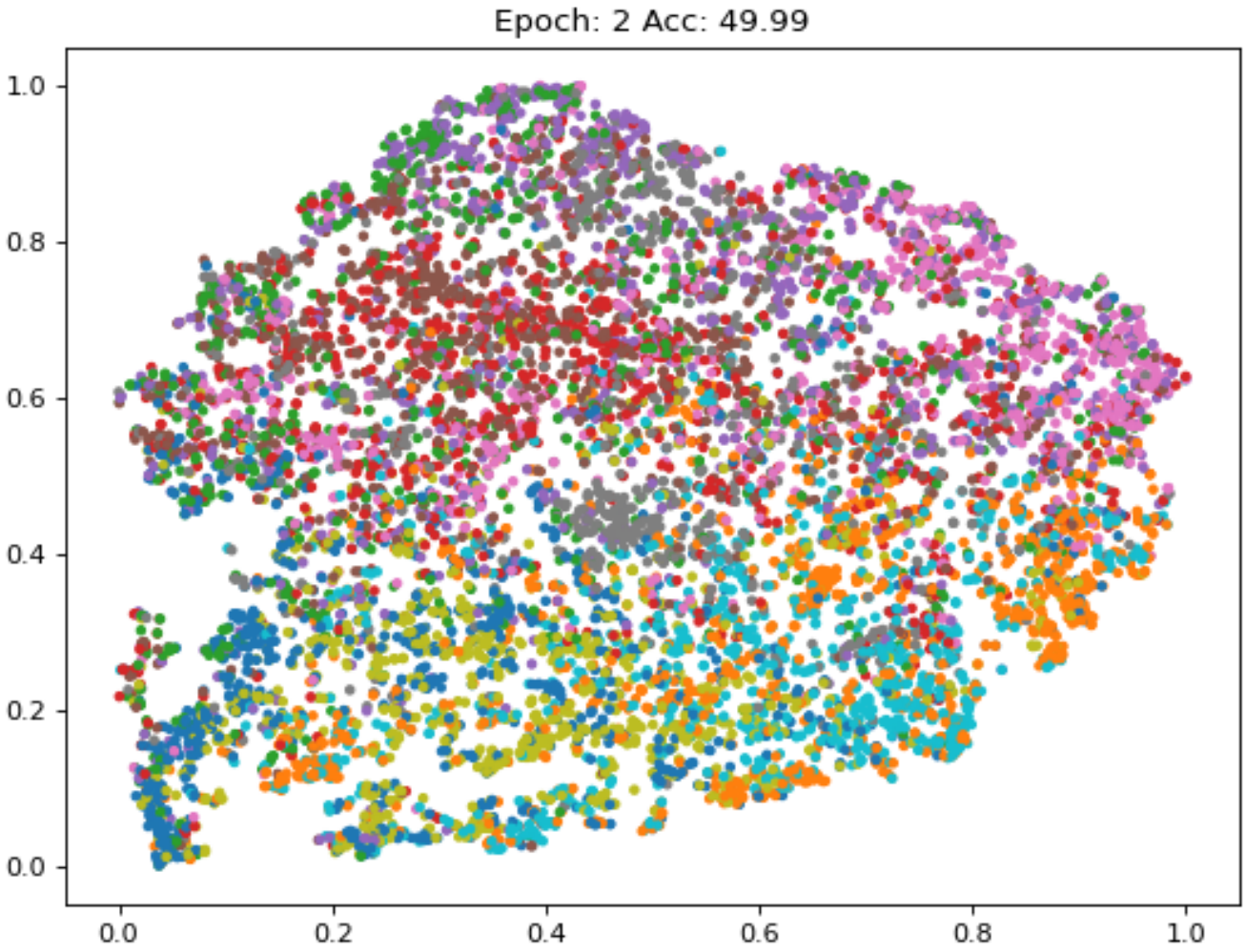
Visualization of the last hidden layer output for CIFAR10 samples



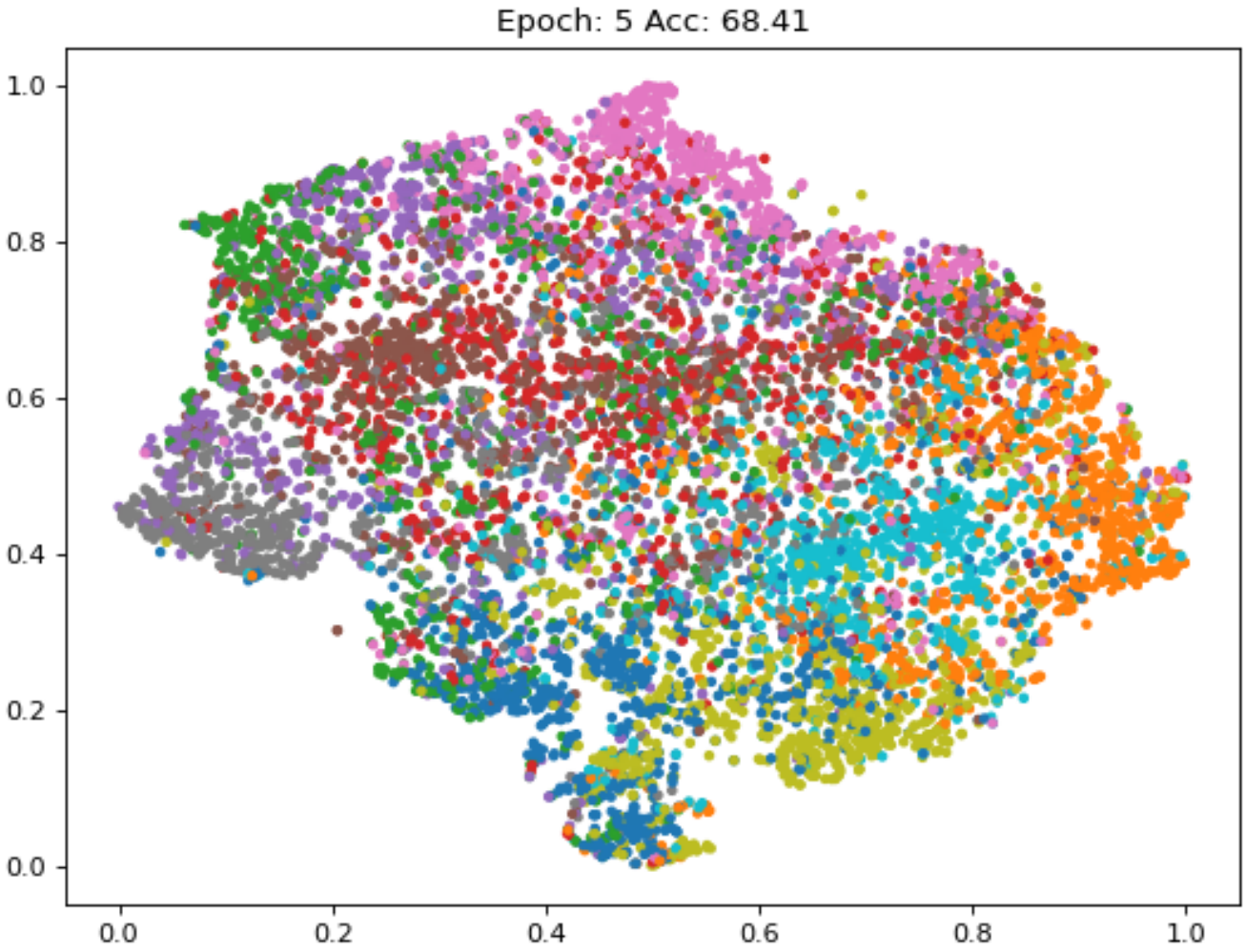
Visualization of the last hidden layer output for CIFAR10 samples



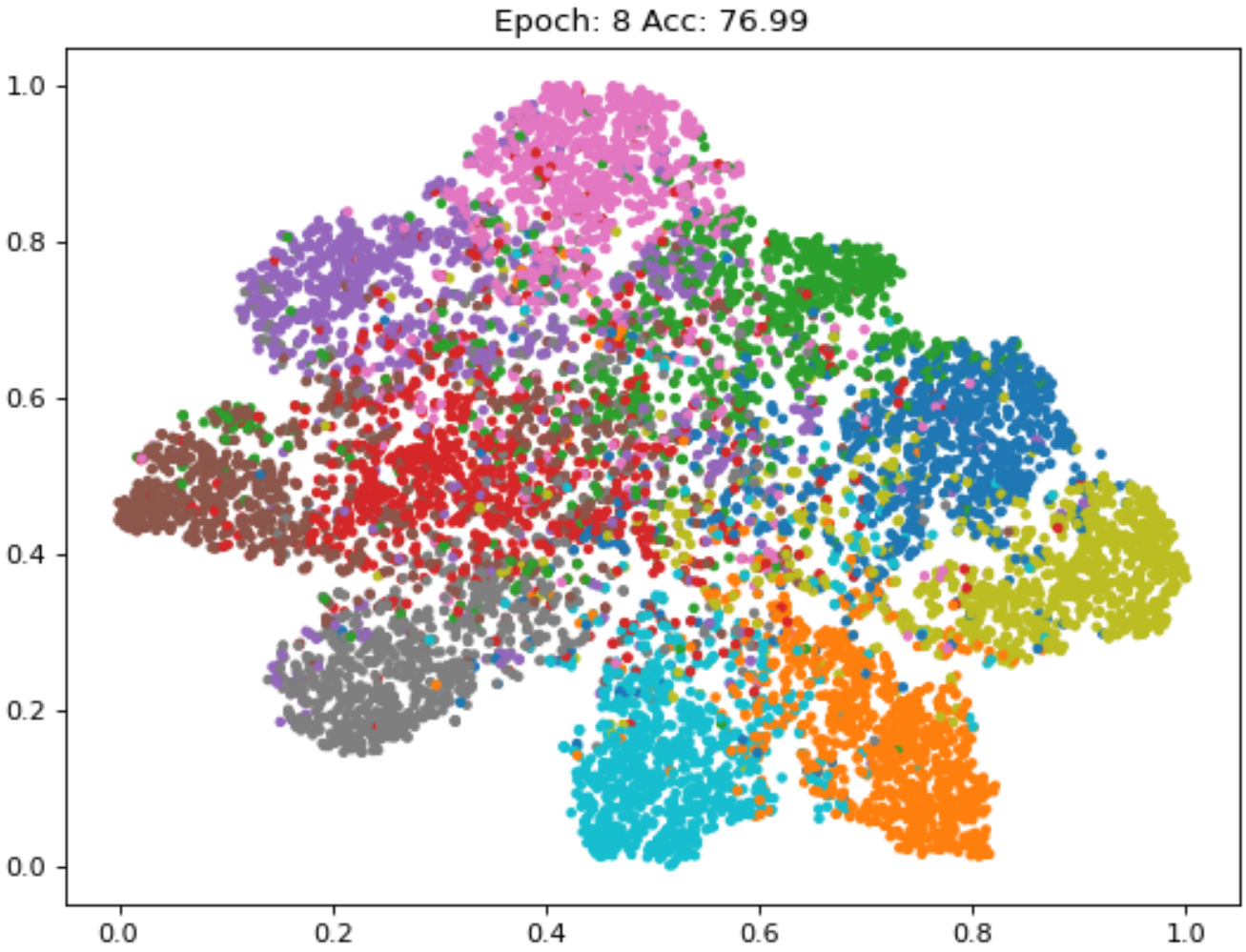
Visualization of the last hidden layer output for CIFAR10 samples



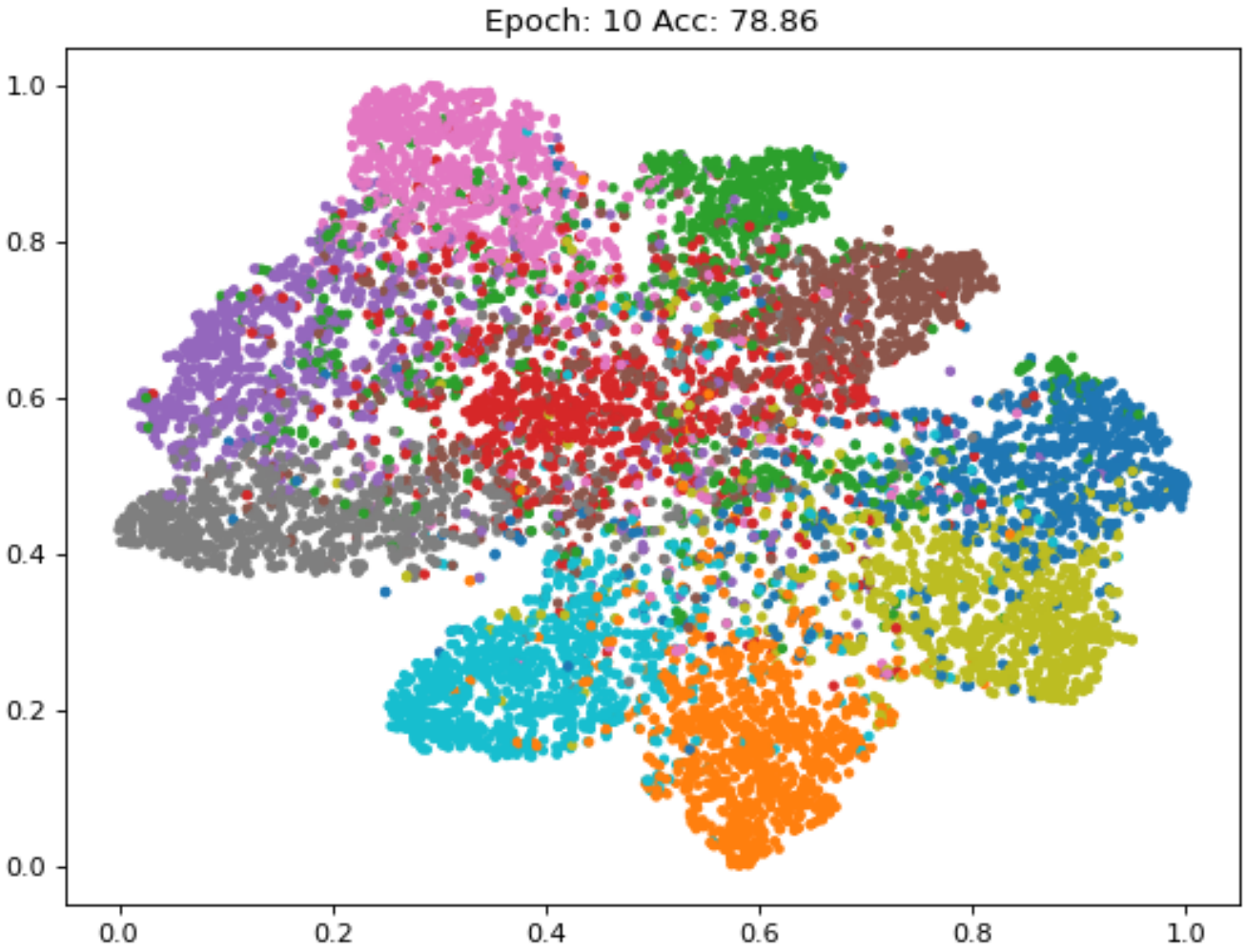
Visualization of the last hidden layer output for CIFAR10 samples



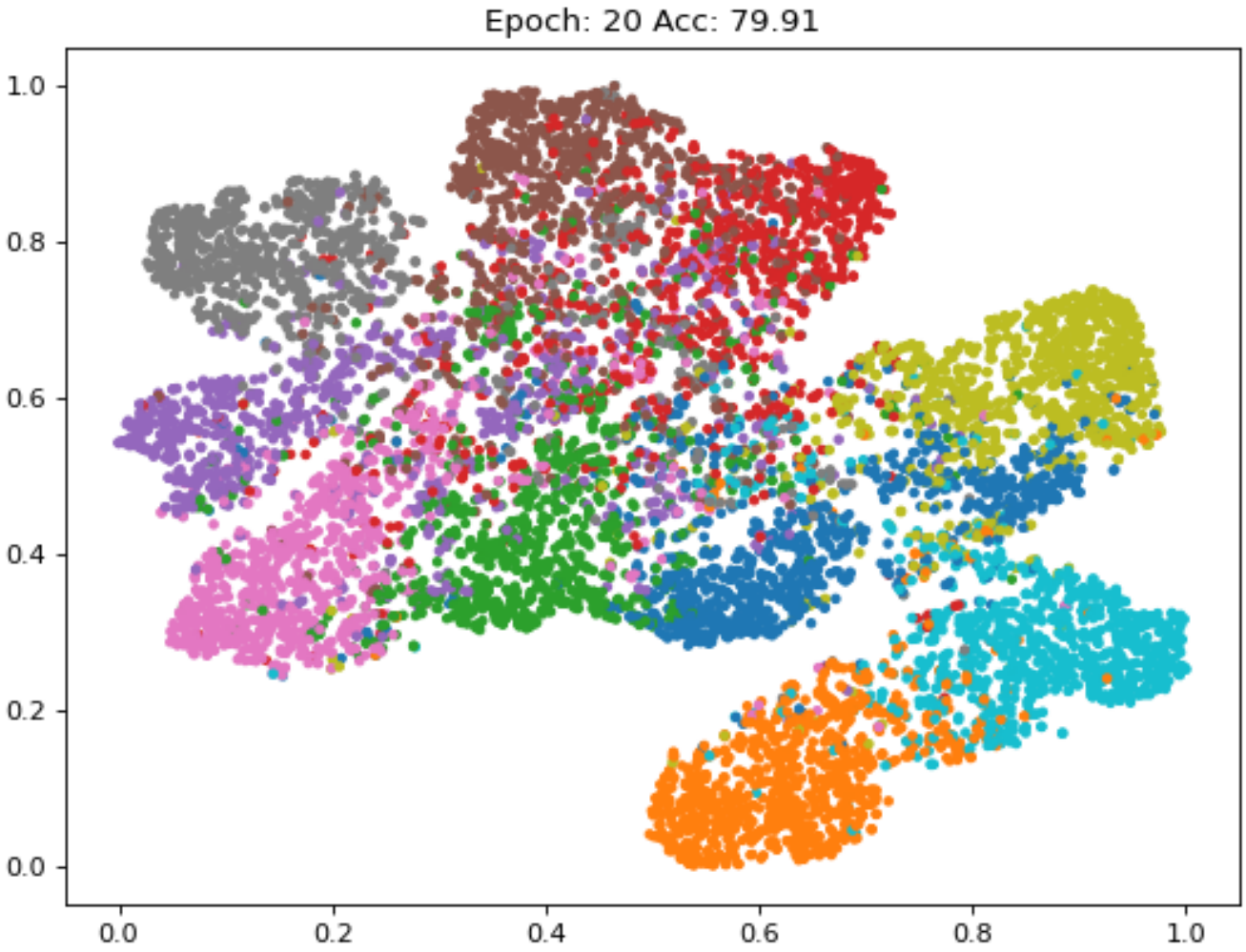
Visualization of the last hidden layer output for CIFAR10 samples



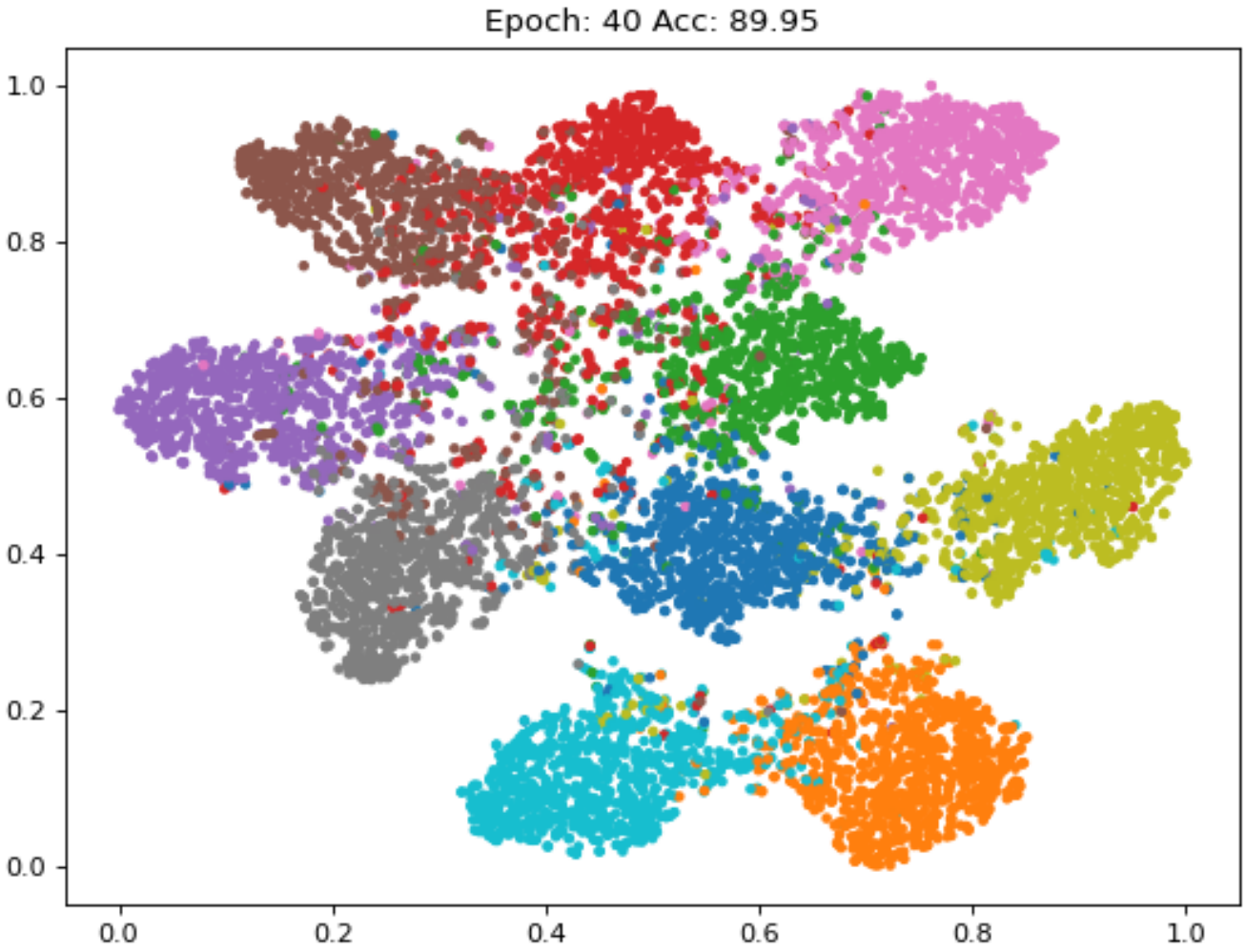
Visualization of the last hidden layer output for CIFAR10 samples



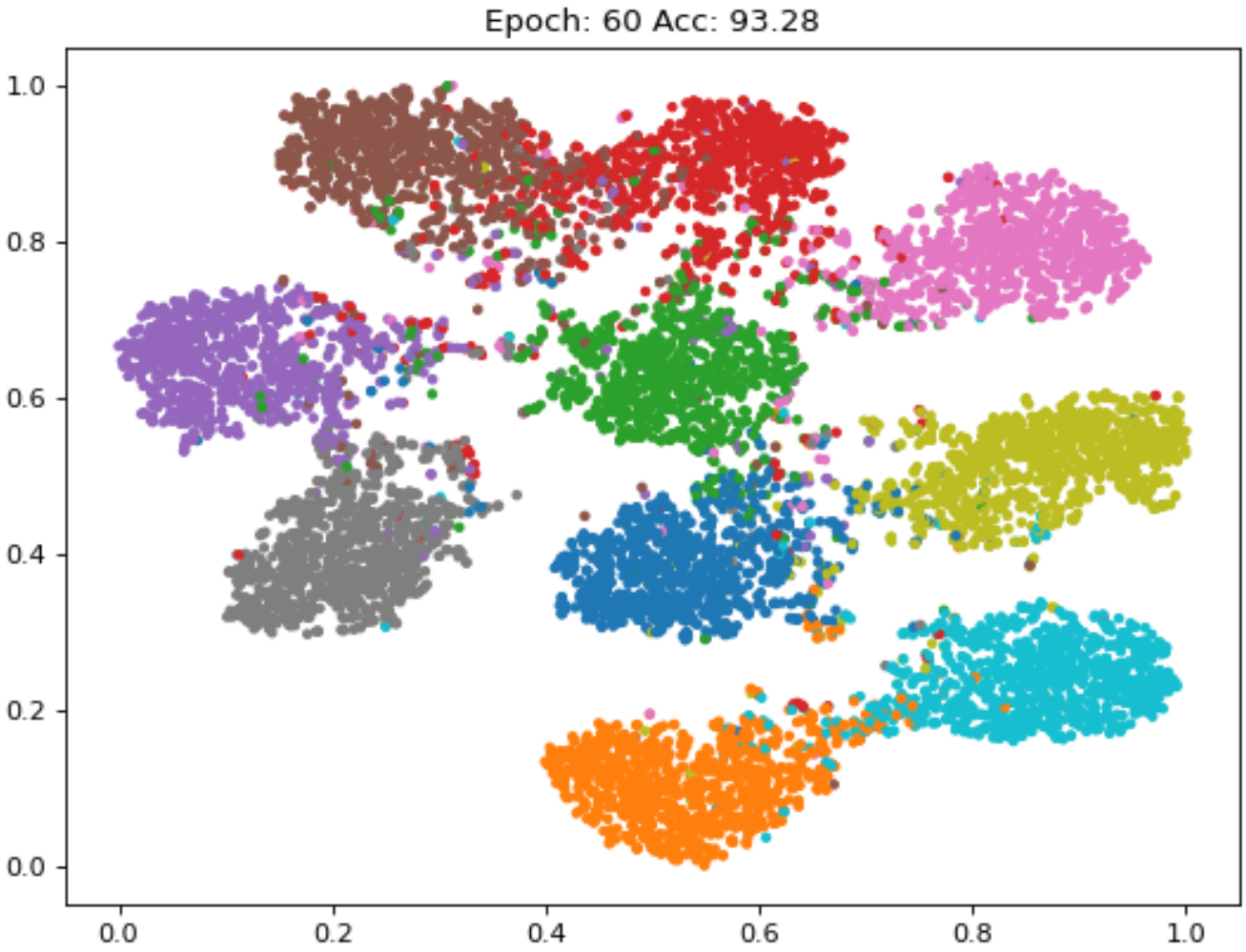
Visualization of the last hidden layer output for CIFAR10 samples



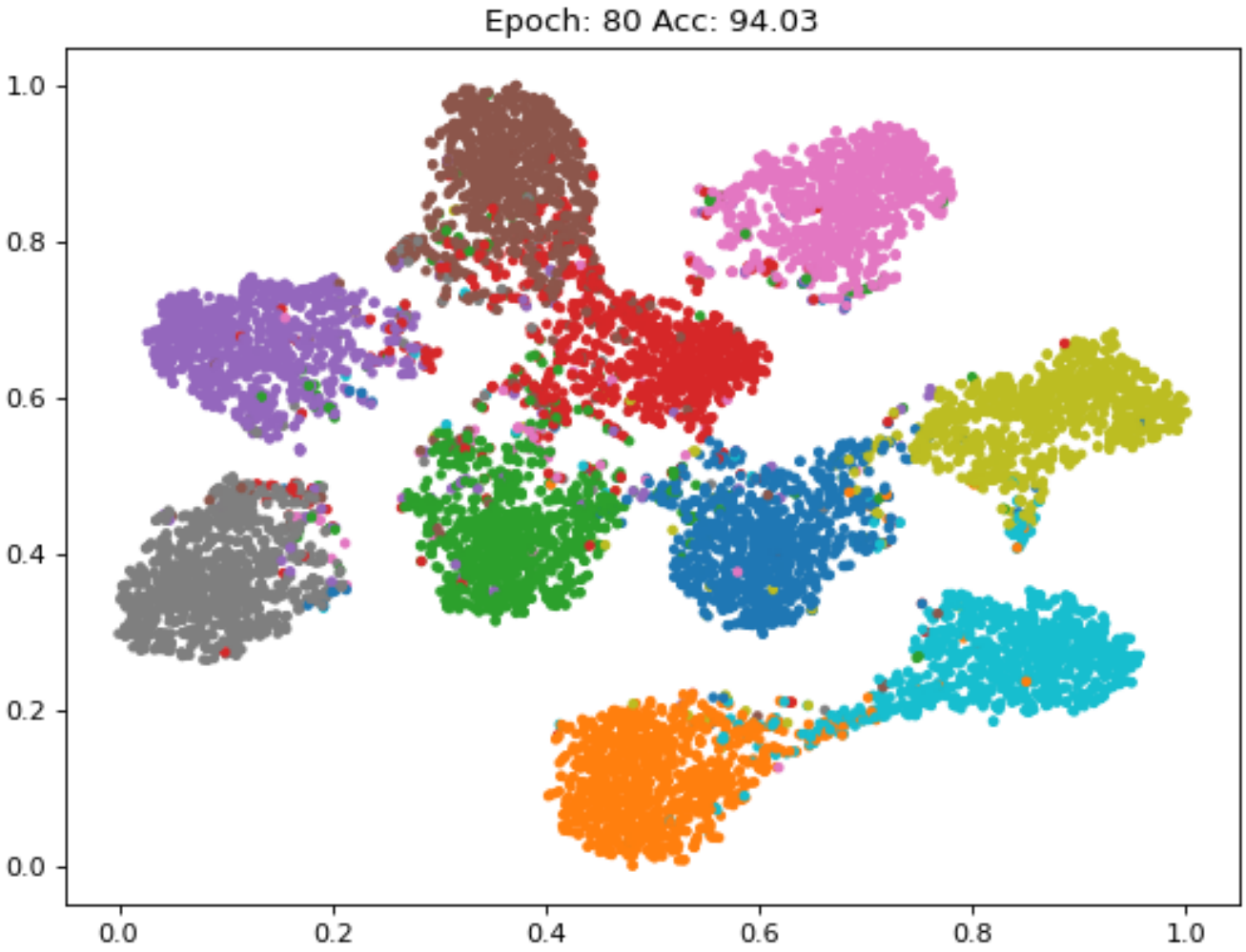
Visualization of the last hidden layer output for CIFAR10 samples



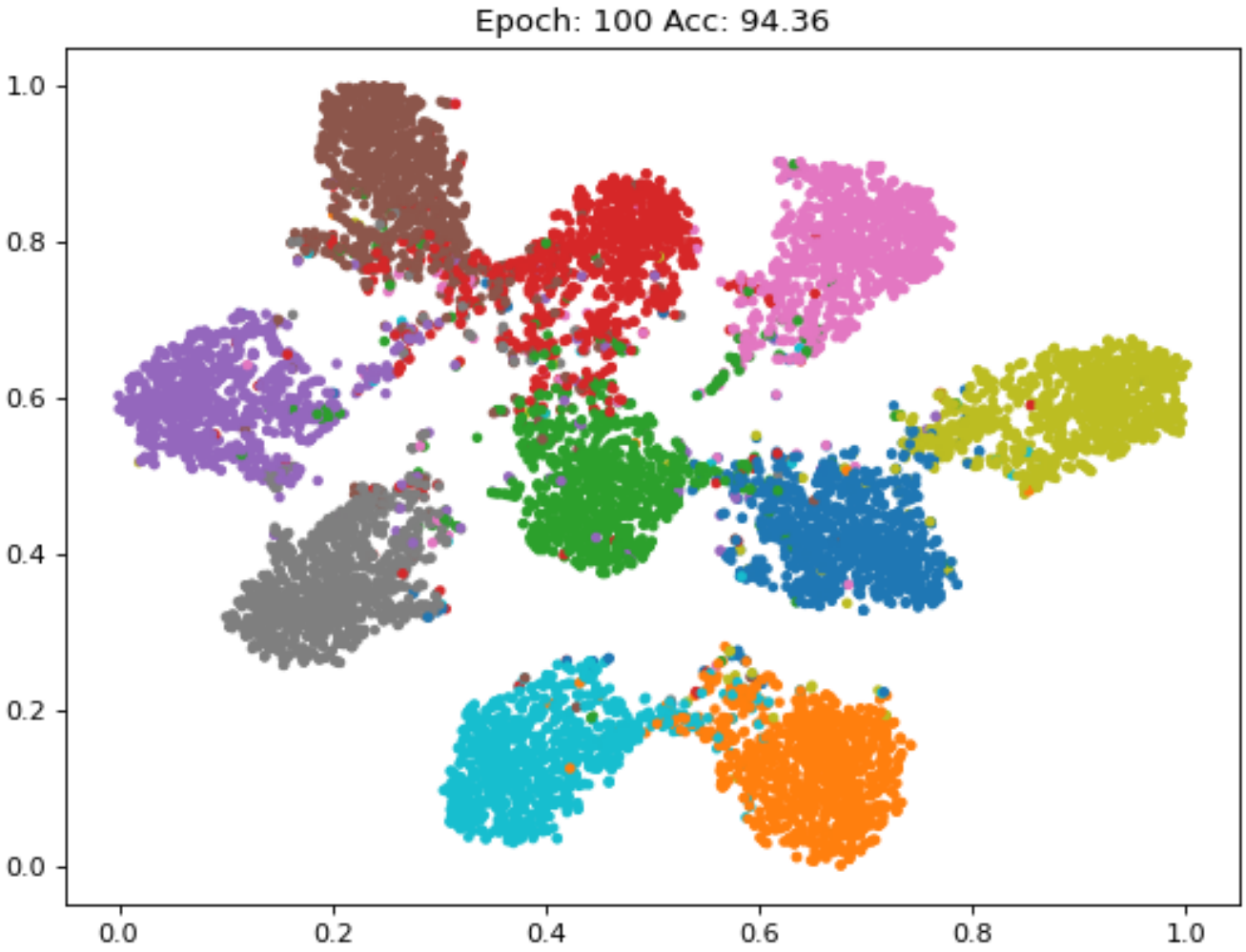
Visualization of the last hidden layer output for CIFAR10 samples



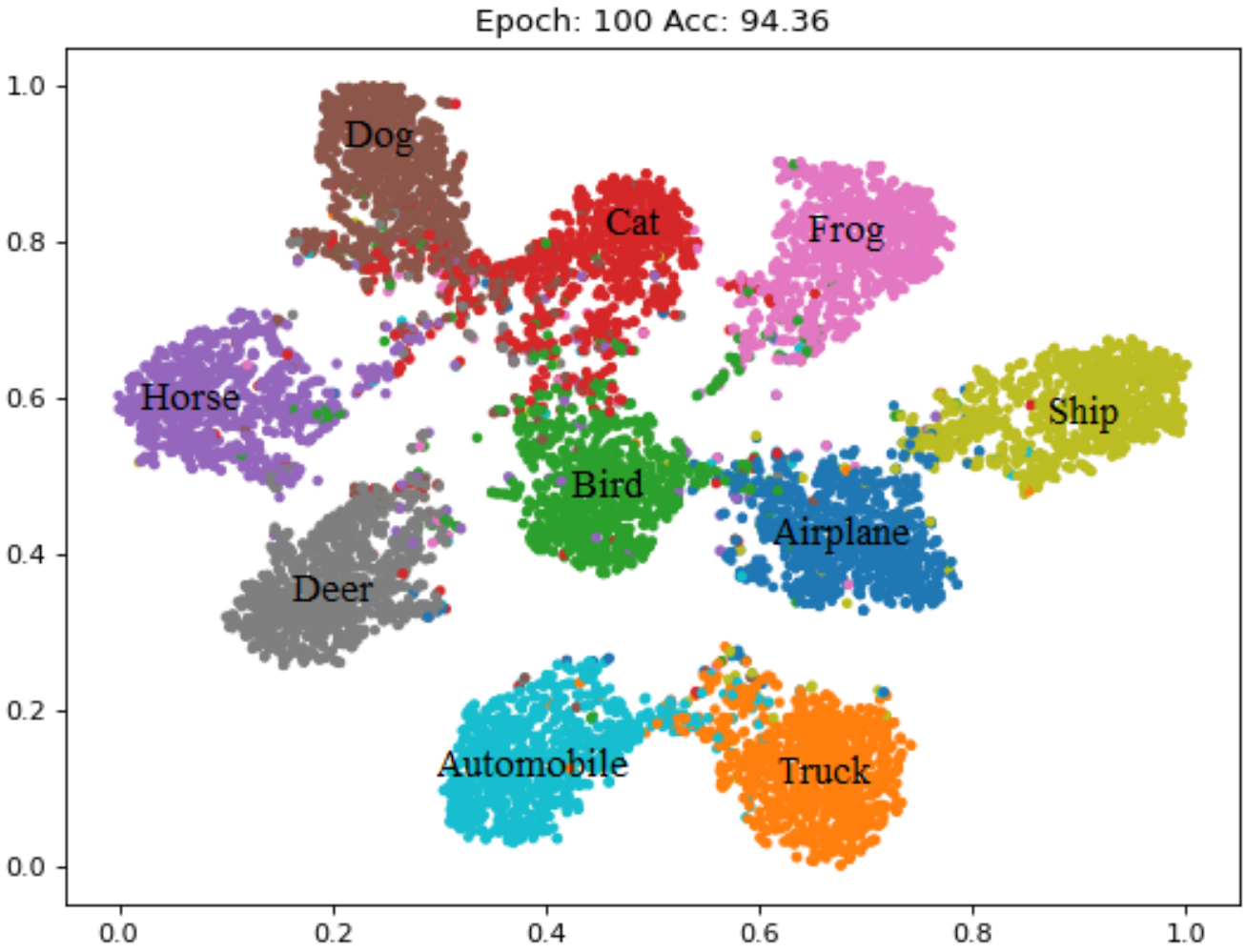
Visualization of the last hidden layer output for CIFAR10 samples



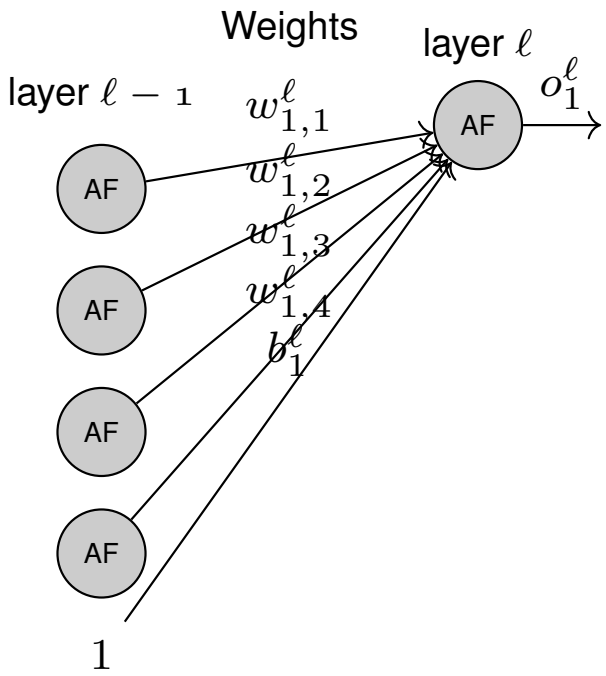
Visualization of the last hidden layer output for CIFAR10 samples



Visualization of the last hidden layer output for CIFAR10 samples



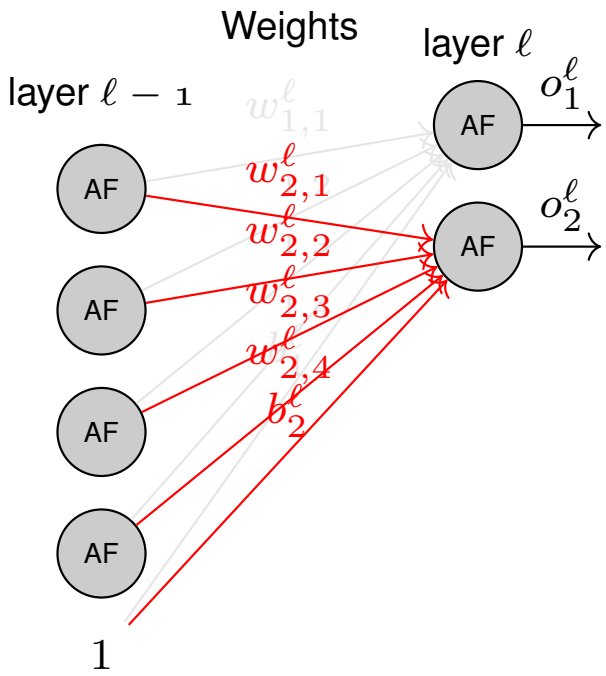
Fully connected neural networks



$$z_1^\ell = \sum_i w_{1,i}^\ell o_i^{\ell-1} + b_1^\ell = \boldsymbol{w_1^\ell}^T \boldsymbol{o}^{\ell-1} + b_1^\ell, \quad o_1^\ell = AF(z_1^\ell)$$

$$z_1^\ell = \begin{bmatrix} w_{1,1}^\ell & w_{1,2}^\ell & w_{1,3}^\ell & w_{1,4}^\ell \end{bmatrix} \begin{bmatrix} o_1^{\ell-1} \\ o_2^{\ell-1} \\ o_3^{\ell-1} \\ o_4^{\ell-1} \end{bmatrix} + b_1^\ell$$

Fully connected neural networks

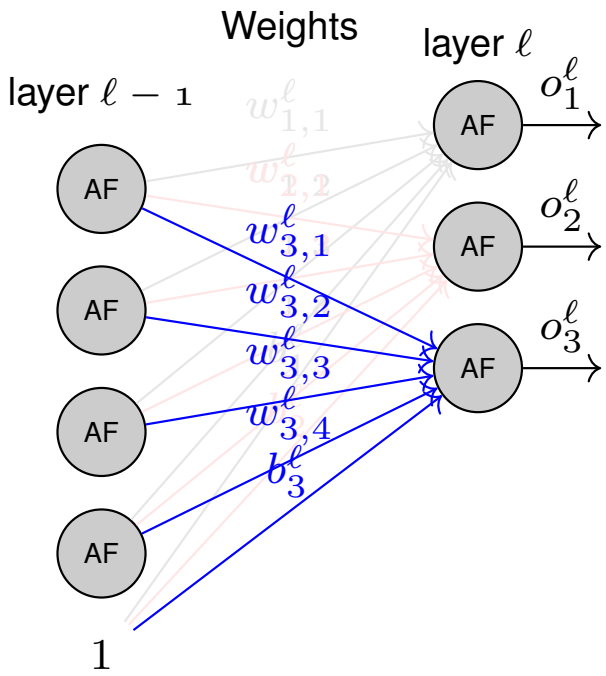


$$z_1^\ell = \sum_i w_{1,i}^\ell o_i^{\ell-1} + b_1^\ell = \boldsymbol{w_1^\ell}^T \boldsymbol{o^{\ell-1}} + b_1^\ell, \quad o_1^\ell = AF(z_1^\ell)$$

$$z_2^\ell = \sum_i w_{2,i}^\ell o_i^{\ell-1} + b_2^\ell = \boldsymbol{w_2^\ell}^T \boldsymbol{o^{\ell-1}} + b_2^\ell, \quad o_2^\ell = AF(z_2^\ell)$$

$$\begin{bmatrix} z_1^\ell \\ z_2^\ell \end{bmatrix} = \begin{bmatrix} w_{1,1}^\ell & w_{1,2}^\ell & w_{1,3}^\ell & w_{1,4}^\ell \\ w_{2,1}^\ell & w_{2,2}^\ell & w_{2,3}^\ell & w_{2,4}^\ell \end{bmatrix} \begin{bmatrix} o_1^{\ell-1} \\ o_2^{\ell-1} \\ o_3^{\ell-1} \\ o_4^{\ell-1} \end{bmatrix} + \begin{bmatrix} b_1^\ell \\ b_2^\ell \end{bmatrix}$$

Fully connected neural networks



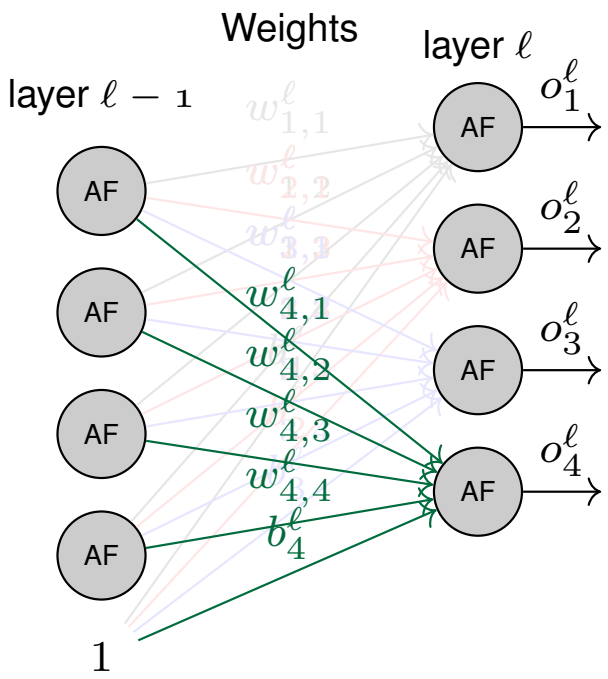
$$z_1^\ell = \sum_i w_{1,i}^\ell o_i^{\ell-1} + b_1^\ell = \boldsymbol{w_1^\ell}^T \boldsymbol{o^{\ell-1}} + b_1^\ell, \quad o_1^\ell = AF(z_1^\ell)$$

$$z_2^\ell = \sum_i w_{2,i}^\ell o_i^{\ell-1} + b_2^\ell = \boldsymbol{w_2^\ell}^T \boldsymbol{o^{\ell-1}} + b_2^\ell, \quad o_2^\ell = AF(z_2^\ell)$$

$$z_3^\ell = \sum_i w_{3,i}^\ell o_i^{\ell-1} + b_3^\ell = \boldsymbol{w_3^\ell}^T \boldsymbol{o^{\ell-1}} + b_3^\ell, \quad o_3^\ell = AF(z_3^\ell)$$

$$\begin{bmatrix} z_1^\ell \\ z_2^\ell \\ z_3^\ell \end{bmatrix} = \begin{bmatrix} w_{1,1}^\ell & w_{1,2}^\ell & w_{1,3}^\ell & w_{1,4}^\ell \\ w_{2,1}^\ell & w_{2,2}^\ell & w_{2,3}^\ell & w_{2,4}^\ell \\ w_{3,1}^\ell & w_{3,2}^\ell & w_{3,3}^\ell & w_{3,4}^\ell \end{bmatrix} \begin{bmatrix} o_1^{\ell-1} \\ o_2^{\ell-1} \\ o_3^{\ell-1} \\ o_4^{\ell-1} \end{bmatrix} + \begin{bmatrix} b_1^\ell \\ b_2^\ell \\ b_3^\ell \end{bmatrix}$$

Fully connected neural networks



$$z_1^\ell = \sum_i w_{1,i}^\ell o_i^{\ell-1} + b_1^\ell = \boldsymbol{w_1^\ell}^T \boldsymbol{o^{\ell-1}} + b_1^\ell, \quad o_1^\ell = AF(z_1^\ell)$$

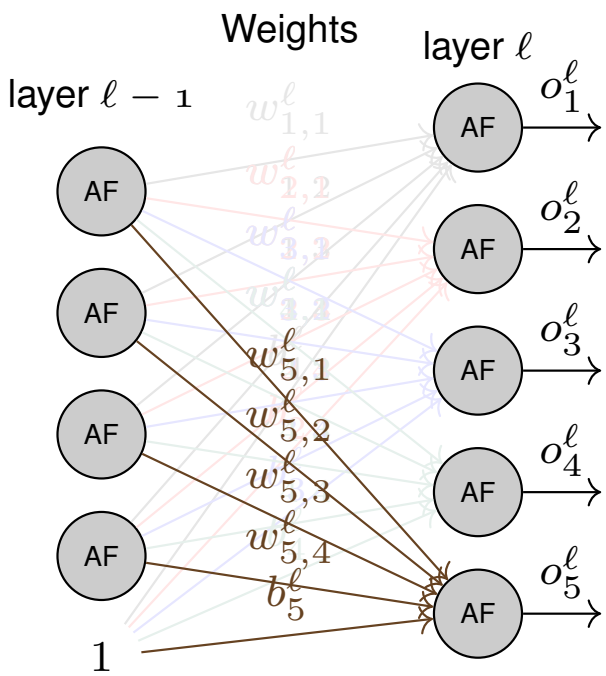
$$z_2^\ell = \sum_i w_{2,i}^\ell o_i^{\ell-1} + b_2^\ell = \boldsymbol{w_2^\ell}^T \boldsymbol{o^{\ell-1}} + b_2^\ell, \quad o_2^\ell = AF(z_2^\ell)$$

$$z_3^\ell = \sum_i w_{3,i}^\ell o_i^{\ell-1} + b_3^\ell = \boldsymbol{w_3^\ell}^T \boldsymbol{o^{\ell-1}} + b_3^\ell, \quad o_3^\ell = AF(z_3^\ell)$$

$$z_4^\ell = \sum_i w_{4,i}^\ell o_i^{\ell-1} + b_4^\ell = \boldsymbol{w_4^\ell}^T \boldsymbol{o^{\ell-1}} + b_4^\ell, \quad o_4^\ell = AF(z_4^\ell)$$

$$\begin{bmatrix} z_1^\ell \\ z_2^\ell \\ z_3^\ell \\ z_4^\ell \end{bmatrix} = \begin{bmatrix} w_{1,1}^\ell & w_{1,2}^\ell & w_{1,3}^\ell & w_{1,4}^\ell \\ w_{2,1}^\ell & w_{2,2}^\ell & w_{2,3}^\ell & w_{2,4}^\ell \\ w_{3,1}^\ell & w_{3,2}^\ell & w_{3,3}^\ell & w_{3,4}^\ell \\ w_{4,1}^\ell & w_{4,2}^\ell & w_{4,3}^\ell & w_{4,4}^\ell \end{bmatrix} \begin{bmatrix} o_1^{\ell-1} \\ o_2^{\ell-1} \\ o_3^{\ell-1} \\ o_4^{\ell-1} \end{bmatrix} + \begin{bmatrix} b_1^\ell \\ b_2^\ell \\ b_3^\ell \\ b_4^\ell \end{bmatrix}$$

Fully connected neural networks



$$z_1^\ell = \sum_i w_{1,i}^\ell o_i^{\ell-1} + b_1^\ell = \boldsymbol{w_1^\ell}^T \boldsymbol{o^{\ell-1}} + b_1^\ell, \quad o_1^\ell = AF(z_1^\ell)$$

$$z_2^\ell = \sum_i w_{2,i}^\ell o_i^{\ell-1} + b_2^\ell = \boldsymbol{w_2^\ell}^T \boldsymbol{o^{\ell-1}} + b_2^\ell, \quad o_2^\ell = AF(z_2^\ell)$$

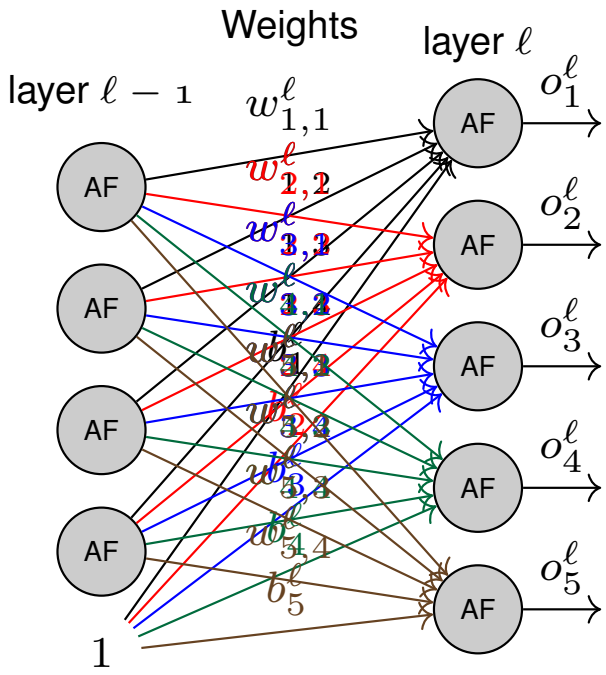
$$z_3^\ell = \sum_i w_{3,i}^\ell o_i^{\ell-1} + b_3^\ell = \boldsymbol{w_3^\ell}^T \boldsymbol{o^{\ell-1}} + b_3^\ell, \quad o_3^\ell = AF(z_3^\ell)$$

$$z_4^\ell = \sum_i w_{4,i}^\ell o_i^{\ell-1} + b_4^\ell = \boldsymbol{w_4^\ell}^T \boldsymbol{o^{\ell-1}} + b_4^\ell, \quad o_4^\ell = AF(z_4^\ell)$$

$$z_5^\ell = \sum_i w_{5,i}^\ell o_i^{\ell-1} + b_5^\ell = \boldsymbol{w_5^\ell}^T \boldsymbol{o^{\ell-1}} + b_5^\ell, \quad o_5^\ell = AF(z_5^\ell)$$

$$\begin{bmatrix} z_1^\ell \\ z_2^\ell \\ z_3^\ell \\ z_4^\ell \\ z_5^\ell \end{bmatrix} = \begin{bmatrix} w_{1,1}^\ell & w_{1,2}^\ell & w_{1,3}^\ell & w_{1,4}^\ell \\ w_{2,1}^\ell & w_{2,2}^\ell & w_{2,3}^\ell & w_{2,4}^\ell \\ w_{3,1}^\ell & w_{3,2}^\ell & w_{3,3}^\ell & w_{3,4}^\ell \\ w_{4,1}^\ell & w_{4,2}^\ell & w_{4,3}^\ell & w_{4,4}^\ell \\ w_{5,1}^\ell & w_{5,2}^\ell & w_{5,3}^\ell & w_{5,4}^\ell \end{bmatrix} \begin{bmatrix} o_1^{\ell-1} \\ o_2^{\ell-1} \\ o_3^{\ell-1} \\ o_4^{\ell-1} \end{bmatrix} + \begin{bmatrix} b_1^\ell \\ b_2^\ell \\ b_3^\ell \\ b_4^\ell \\ b_5^\ell \end{bmatrix}$$

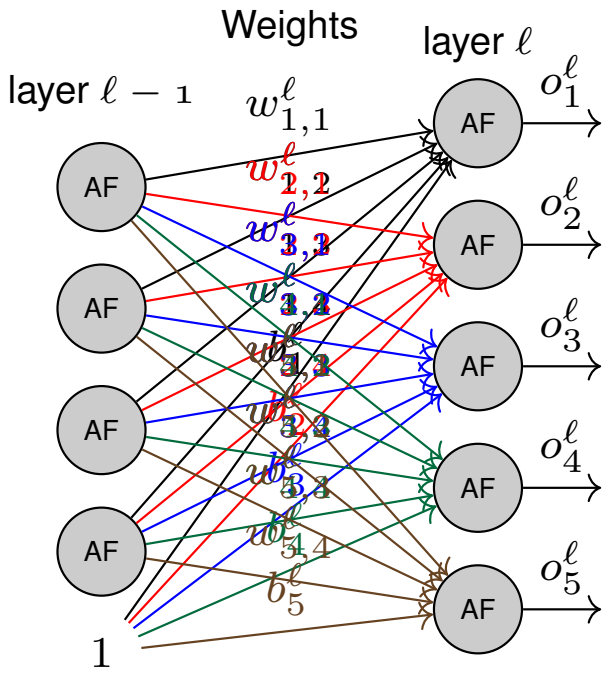
Vector activation



$$\begin{bmatrix} z_1^\ell \\ z_2^\ell \\ z_3^\ell \\ z_4^\ell \\ z_5^\ell \end{bmatrix} = \begin{bmatrix} w_{1,1}^\ell & w_{1,2}^\ell & w_{1,3}^\ell & w_{1,4}^\ell \\ w_{2,1}^\ell & w_{2,2}^\ell & w_{2,3}^\ell & w_{2,4}^\ell \\ w_{3,1}^\ell & w_{3,2}^\ell & w_{3,3}^\ell & w_{3,4}^\ell \\ w_{4,1}^\ell & w_{4,2}^\ell & w_{4,3}^\ell & w_{4,4}^\ell \\ w_{5,1}^\ell & w_{5,2}^\ell & w_{5,3}^\ell & w_{5,4}^\ell \end{bmatrix} \begin{bmatrix} o_1^{\ell-1} \\ o_2^{\ell-1} \\ o_3^{\ell-1} \\ o_4^{\ell-1} \end{bmatrix} + \begin{bmatrix} b_1^\ell \\ b_2^\ell \\ b_3^\ell \\ b_4^\ell \\ b_5^\ell \end{bmatrix}$$

$$\begin{bmatrix} o_1^\ell \\ o_2^\ell \\ o_3^\ell \\ o_4^\ell \\ o_5^\ell \end{bmatrix} = \begin{bmatrix} AF(z_1^\ell) \\ AF(z_2^\ell) \\ AF(z_3^\ell) \\ AF(z_4^\ell) \\ AF(z_5^\ell) \end{bmatrix}$$

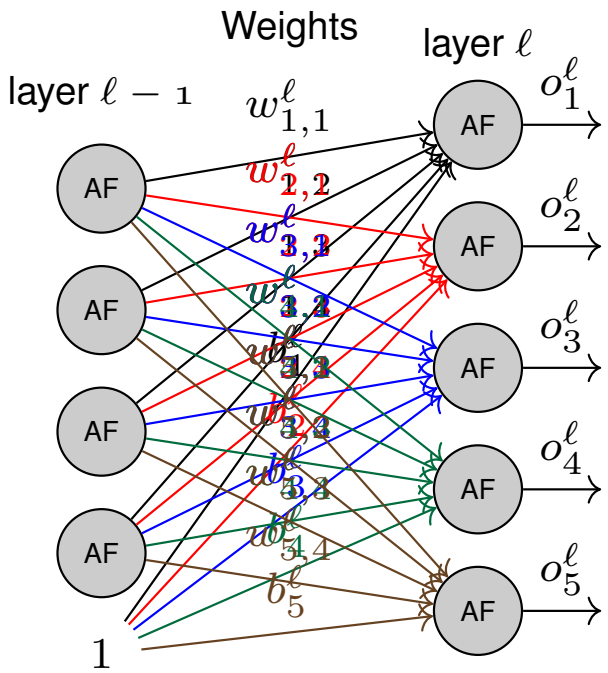
Vector activation



$$\begin{bmatrix} z_1^\ell \\ z_2^\ell \\ z_3^\ell \\ z_4^\ell \\ z_5^\ell \end{bmatrix} = \begin{bmatrix} w_{1,1}^\ell & w_{1,2}^\ell & w_{1,3}^\ell & w_{1,4}^\ell \\ w_{2,1}^\ell & w_{2,2}^\ell & w_{2,3}^\ell & w_{2,4}^\ell \\ w_{3,1}^\ell & w_{3,2}^\ell & w_{3,3}^\ell & w_{3,4}^\ell \\ w_{4,1}^\ell & w_{4,2}^\ell & w_{4,3}^\ell & w_{4,4}^\ell \\ w_{5,1}^\ell & w_{5,2}^\ell & w_{5,3}^\ell & w_{5,4}^\ell \end{bmatrix} \begin{bmatrix} o_1^{\ell-1} \\ o_2^{\ell-1} \\ o_3^{\ell-1} \\ o_4^{\ell-1} \end{bmatrix} + \begin{bmatrix} b_1^\ell \\ b_2^\ell \\ b_3^\ell \\ b_4^\ell \\ b_5^\ell \end{bmatrix}$$

$$\begin{bmatrix} o_1^\ell \\ o_2^\ell \\ o_3^\ell \\ o_4^\ell \\ o_5^\ell \end{bmatrix} = \begin{bmatrix} AF(z_1^\ell) \\ AF(z_2^\ell) \\ AF(z_3^\ell) \\ AF(z_4^\ell) \\ AF(z_5^\ell) \end{bmatrix} = AF\left(\begin{bmatrix} z_1^\ell \\ z_2^\ell \\ z_3^\ell \\ z_4^\ell \\ z_5^\ell \end{bmatrix} \right)$$

Vector activation

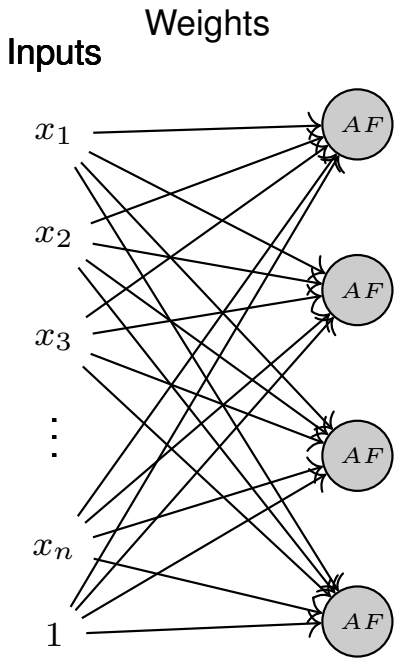


$$\begin{bmatrix} z^\ell_1 \\ z^\ell_2 \\ z^\ell_3 \\ z^\ell_4 \\ z^\ell_5 \end{bmatrix} = \begin{bmatrix} w^\ell_{1,1} & w^\ell_{1,2} & w^\ell_{1,3} & w^\ell_{1,4} \\ w^\ell_{2,1} & w^\ell_{2,2} & w^\ell_{2,3} & w^\ell_{2,4} \\ w^\ell_{3,1} & w^\ell_{3,2} & w^\ell_{3,3} & w^\ell_{3,4} \\ w^\ell_{4,1} & w^\ell_{4,2} & w^\ell_{4,3} & w^\ell_{4,4} \\ w^\ell_{5,1} & w^\ell_{5,2} & w^\ell_{5,3} & w^\ell_{5,4} \end{bmatrix} \begin{bmatrix} o^{\ell-1}_1 \\ o^{\ell-1}_2 \\ o^{\ell-1}_3 \\ o^{\ell-1}_4 \end{bmatrix} + \begin{bmatrix} b^\ell_1 \\ b^\ell_2 \\ b^\ell_3 \\ b^\ell_4 \\ b^\ell_5 \end{bmatrix}$$

$$\begin{bmatrix} o^\ell_1 \\ o^\ell_2 \\ o^\ell_3 \\ o^\ell_4 \\ o^\ell_5 \end{bmatrix} = \begin{bmatrix} AF(z^\ell_1) \\ AF(z^\ell_2) \\ AF(z^\ell_3) \\ AF(z^\ell_4) \\ AF(z^\ell_5) \end{bmatrix} = AF\left(\begin{bmatrix} z^\ell_1 \\ z^\ell_2 \\ z^\ell_3 \\ z^\ell_4 \\ z^\ell_5 \end{bmatrix} \right)$$

$$\boldsymbol{o}^\ell = AF(W^\ell \boldsymbol{o}^{\ell-1} + \boldsymbol{b}^\ell)$$

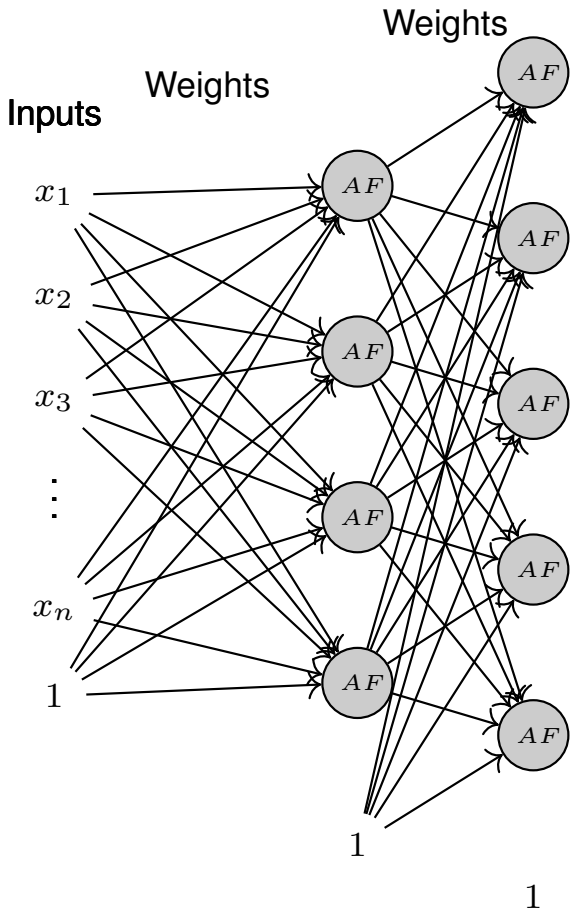
Fully connected neural networks



1

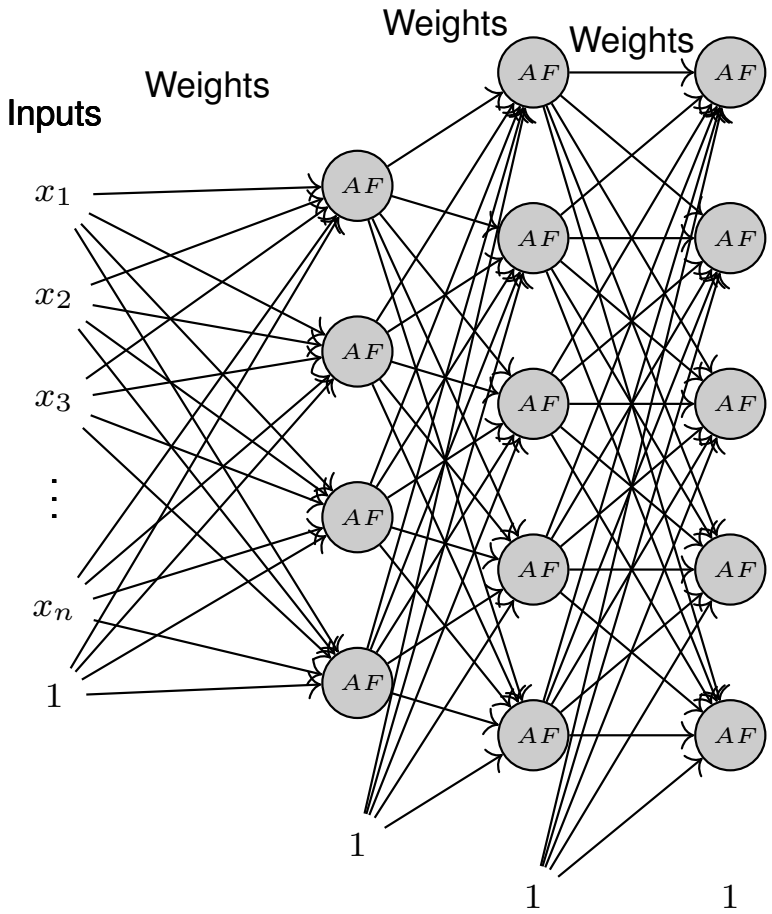
$$\boldsymbol{o}^{\mathbf{1}} = AF(W^{\mathbf{1}}\boldsymbol{x} + \boldsymbol{b}^{\mathbf{1}}), \quad W^{\mathbf{1}} \in \mathbb{R}^{4\times n},\ \boldsymbol{x} \in \mathbb{R}^{n},\ \boldsymbol{o}^{\mathbf{1}} \in \mathbb{R}^{4},\ \boldsymbol{b} \in \mathbb{R}^{4}.$$

Fully connected neural networks



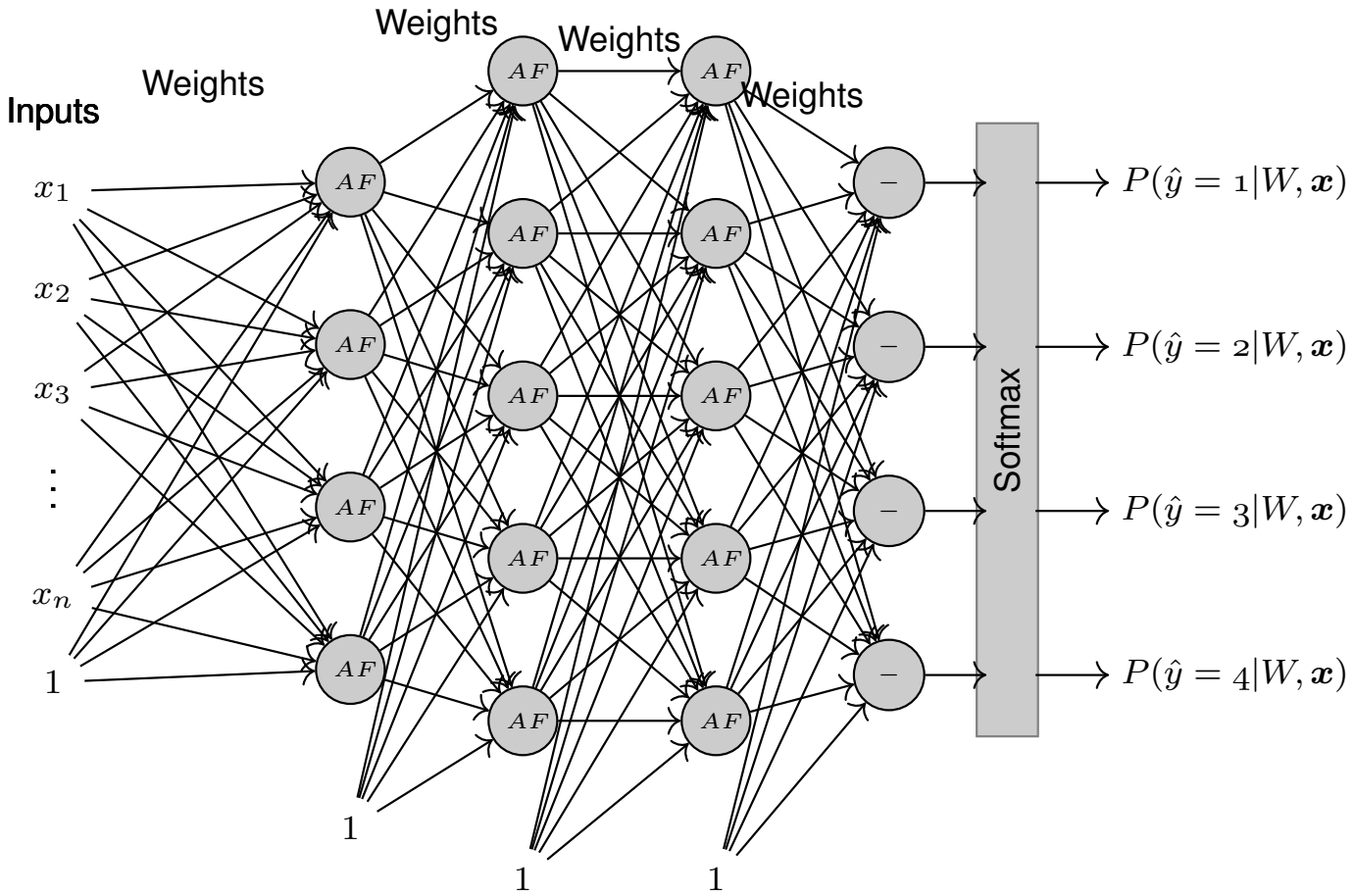
$$\boldsymbol{o^2} = AF(W^2\boldsymbol{o^1} + \boldsymbol{b^2}), \quad W^2 \in \mathbb{R}^{5\times 4},\ \boldsymbol{o^1} \in \mathbb{R}^4,\ \boldsymbol{o^2} \in \mathbb{R}^5,\ \boldsymbol{b^2} \in \mathbb{R}^5.$$

Fully connected neural networks



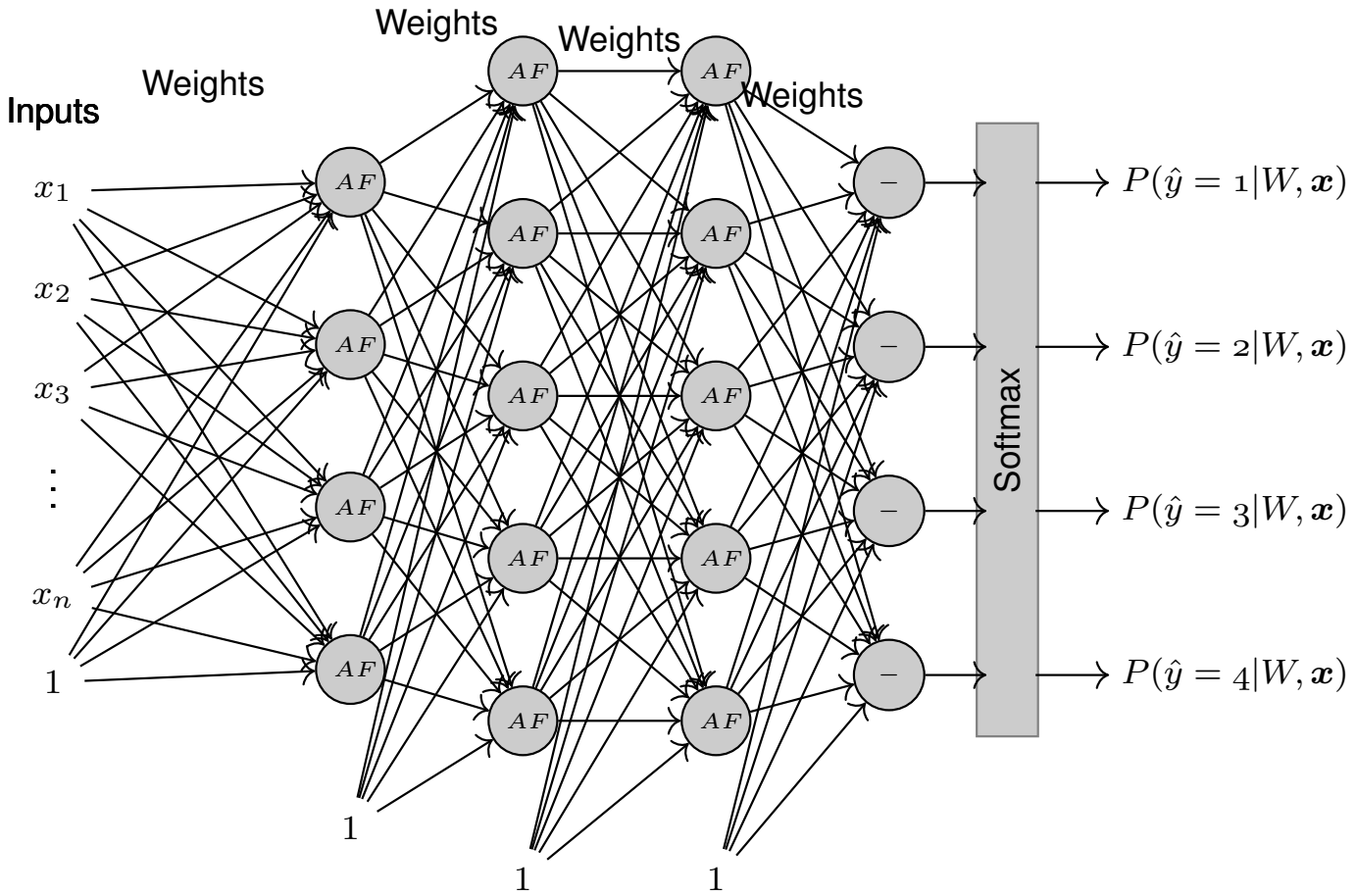
$$\boldsymbol{o}^3 = AF(W^3\boldsymbol{o}^2 + \boldsymbol{b}^3), \quad W^3 \in \mathbb{R}^{5\times 5},\ \boldsymbol{o}^2 \in \mathbb{R}^5,\ \boldsymbol{o}^3 \in \mathbb{R}^5,\ \boldsymbol{b}^3 \in \mathbb{R}^5.$$

Fully connected neural networks



$$\hat{\boldsymbol{y}} = Softmax(W^4\boldsymbol{o}^3 + \boldsymbol{b}^4), \quad W^4 \in \mathbb{R}^{4\times 5},\ \boldsymbol{o}^3 \in \mathbb{R}^5,\ \hat{\boldsymbol{y}} \in \mathbb{R}^4,\ \boldsymbol{b}^4 \in \mathbb{R}^4.$$

Fully connected neural networks



$$\hat{\boldsymbol{y}} = Softmax(W^4(AF(W^3(AF(W^2(AF(W^1\boldsymbol{x} + \boldsymbol{b}^1)) + \boldsymbol{b}^2)) + \boldsymbol{b}^3)) + \boldsymbol{b}^4)$$

Batch of inputs

- The output of the first hidden layer for a batch of inputs

$$\begin{bmatrix} z_1^{1,(1)} & z_1^{1,(2)} & z_1^{1,(3)} \\ z_2^{1,(1)} & z_2^{1,(2)} & z_2^{1,(3)} \\ z_3^{1,(1)} & z_3^{1,(2)} & z_3^{1,(3)} \\ z_4^{1,(1)} & z_4^{1,(2)} & z_4^{1,(3)} \end{bmatrix} = \begin{bmatrix} w_{1,1}^1 & w_{1,2}^1 & w_{1,3}^1 & \dots & w_{1,n}^1 \\ w_{2,1}^1 & w_{2,2}^1 & w_{2,3}^1 & \dots & w_{2,n}^1 \\ w_{3,1}^1 & w_{3,2}^1 & w_{3,3}^1 & \dots & w_{3,n}^1 \\ w_{4,1}^1 & w_{4,2}^1 & w_{4,3}^1 & \dots & w_{4,n}^1 \end{bmatrix} \begin{bmatrix} x_1^{(1)} & x_1^{(2)} & x_1^{(3)} \\ x_2^{(1)} & x_2^{(2)} & x_2^{(3)} \\ x_3^{(1)} & x_3^{(2)} & x_3^{(3)} \\ \vdots & \vdots & \vdots \\ x_n^{(1)} & x_n^{(2)} & x_n^{(3)} \end{bmatrix}$$

$$+ \begin{bmatrix} b_1^1 & b_1^1 & b_1^1 \\ b_2^1 & b_2^1 & b_2^1 \\ b_3^1 & b_3^1 & b_3^1 \\ b_4^1 & b_4^1 & b_4^1 \end{bmatrix}$$

$$\begin{bmatrix} o_1^{1,(1)} & o_1^{1,(2)} & o_1^{1,(3)} \\ o_2^{1,(1)} & o_2^{1,(2)} & o_2^{1,(3)} \\ o_3^{1,(1)} & o_3^{1,(2)} & o_3^{1,(3)} \\ o_4^{1,(1)} & o_4^{1,(2)} & o_4^{1,(3)} \end{bmatrix} = AF\left(\begin{bmatrix} z_1^{1,(1)} & z_1^{1,(2)} & z_1^{1,(3)} \\ z_2^{1,(1)} & z_2^{1,(2)} & z_2^{1,(3)} \\ z_3^{1,(1)} & z_3^{1,(2)} & z_3^{1,(3)} \\ z_4^{1,(1)} & z_4^{1,(2)} & z_4^{1,(3)} \end{bmatrix} \right)$$

Batch of inputs

- The output of the first hidden layer for a batch of inputs

$$\begin{bmatrix} z_1^{1,(1)} & z_1^{1,(2)} & z_1^{1,(3)} \\ z_2^{1,(1)} & z_2^{1,(2)} & z_2^{1,(3)} \\ z_3^{1,(1)} & z_3^{1,(2)} & z_3^{1,(3)} \\ z_4^{1,(1)} & z_4^{1,(2)} & z_4^{1,(3)} \end{bmatrix} = \begin{bmatrix} w_{1,1}^1 & w_{1,2}^1 & w_{1,3}^1 & \dots & w_{1,n}^1 \\ w_{2,1}^1 & w_{2,2}^1 & w_{2,3}^1 & \dots & w_{2,n}^1 \\ w_{3,1}^1 & w_{3,2}^1 & w_{3,3}^1 & \dots & w_{3,n}^1 \\ w_{4,1}^1 & w_{4,2}^1 & w_{4,3}^1 & \dots & w_{4,n}^1 \end{bmatrix} \begin{bmatrix} x_1^{(1)} & x_1^{(2)} & x_1^{(3)} \\ x_2^{(1)} & x_2^{(2)} & x_2^{(3)} \\ x_3^{(1)} & x_3^{(2)} & x_3^{(3)} \\ \vdots & \vdots & \vdots \\ x_n^{(1)} & x_n^{(2)} & x_n^{(3)} \end{bmatrix}$$

$$+ \begin{bmatrix} b_1^1 & b_1^1 & b_1^1 \\ b_2^1 & b_2^1 & b_2^1 \\ b_3^1 & b_3^1 & b_3^1 \\ b_4^1 & b_4^1 & b_4^1 \end{bmatrix}$$

$$\begin{bmatrix} \begin{bmatrix} o_1^{1,(1)} \\ o_2^{1,(1)} \\ o_3^{1,(1)} \\ o_4^{1,(1)} \end{bmatrix} & \begin{bmatrix} o_1^{1,(2)} \\ o_2^{1,(2)} \\ o_3^{1,(2)} \\ o_4^{1,(2)} \end{bmatrix} & \begin{bmatrix} o_1^{1,(3)} \\ o_2^{1,(3)} \\ o_3^{1,(3)} \\ o_4^{1,(3)} \end{bmatrix} \end{bmatrix} = AF\left(\begin{bmatrix} z_1^{1,(1)} & z_1^{1,(2)} & z_1^{1,(3)} \\ z_2^{1,(1)} & z_2^{1,(2)} & z_2^{1,(3)} \\ z_3^{1,(1)} & z_3^{1,(2)} & z_3^{1,(3)} \\ z_4^{1,(1)} & z_4^{1,(2)} & z_4^{1,(3)} \end{bmatrix} \right)$$

$$O^1 = AF(W^1 X + B^1), \quad W^1 \in \mathbb{R}^{4\times n},\ X \in \mathbb{R}^{n\times 3},\ O^1 \in \mathbb{R}^{4\times 3},\ B \in \mathbb{R}^{4\times 3}.$$

Activation functions

What happens if we build a neural network with no activation function?

$$\hat{\boldsymbol{y}} = Softmax(W^4(AF(W^3(AF(W^2(AF(W^1\boldsymbol{x} + \boldsymbol{b}^1)) + \boldsymbol{b}^2)) + \boldsymbol{b}^3)) + \boldsymbol{b}^4)$$

Activation functions

What happens if we build a neural network with no activation function?

$$\hat{\boldsymbol{y}} = Softmax(W^4(AF(W^3(AF(W^2(AF(W^1\boldsymbol{x} + \boldsymbol{b}^1)) + \boldsymbol{b}^2)) + \boldsymbol{b}^3)) + \boldsymbol{b}^4)$$

$$\hat{\boldsymbol{y}} = Softmax(W^4(W^3(W^2(W^1\boldsymbol{x} + \boldsymbol{b}^1) + \boldsymbol{b}^2) + \boldsymbol{b}^3) + \boldsymbol{b}^4)$$

Activation functions

What happens if we build a neural network with no activation function?

$$\hat{\boldsymbol{y}} = Softmax(W^4(AF(W^3(AF(W^2(AF(W^1\boldsymbol{x} + \boldsymbol{b}^1)) + \boldsymbol{b}^2)) + \boldsymbol{b}^3)) + \boldsymbol{b}^4)$$

$$\hat{\boldsymbol{y}} = Softmax(W^4(W^3(W^2(W^1\boldsymbol{x} + \boldsymbol{b}^1) + \boldsymbol{b}^2) + \boldsymbol{b}^3) + \boldsymbol{b}^4)$$

$$\hat{\boldsymbol{y}} = Softmax(W^4W^3W^2W^1\boldsymbol{x} + b')$$

Activation functions

What happens if we build a neural network with no activation function?

$$\hat{\boldsymbol{y}} = Softmax(W^4(AF(W^3(AF(W^2(AF(W^1\boldsymbol{x} + \boldsymbol{b}^1)) + \boldsymbol{b}^2)) + \boldsymbol{b}^3)) + \boldsymbol{b}^4)$$

$$\hat{\boldsymbol{y}} = Softmax(W^4(W^3(W^2(W^1\boldsymbol{x} + \boldsymbol{b}^1) + \boldsymbol{b}^2) + \boldsymbol{b}^3) + \boldsymbol{b}^4)$$

$$\hat{\boldsymbol{y}} = Softmax(W^4W^3W^2W^1\boldsymbol{x} + b')$$

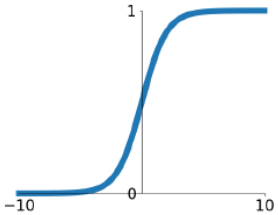
$$\hat{\boldsymbol{y}} = Softmax(W'\boldsymbol{x} + b')$$

We end up with a softmax classifier!

Activation functions

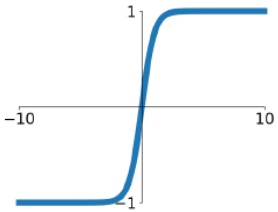
Sigmoid

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



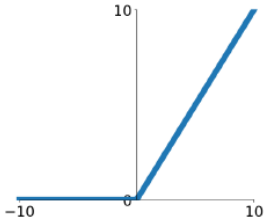
tanh

$$\tanh(x)$$



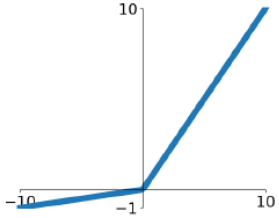
ReLU

$$\max(0, x)$$



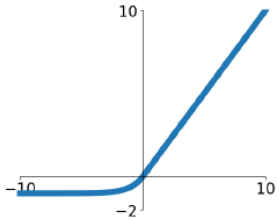
Leaky ReLU

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



ELU

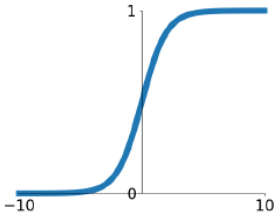
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Activation functions

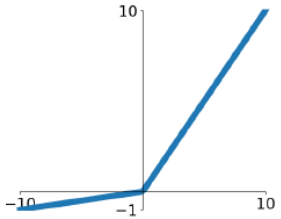
Sigmoid

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



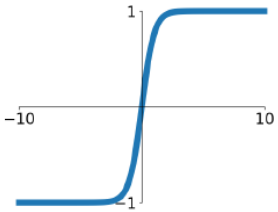
Leaky ReLU

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



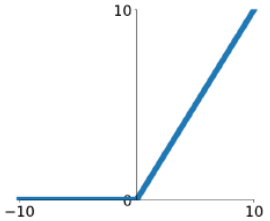
tanh

$$\tanh(x)$$



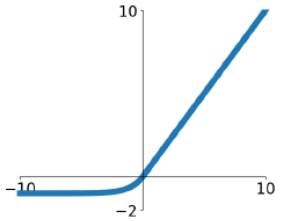
ReLU

$$\max(0, x)$$



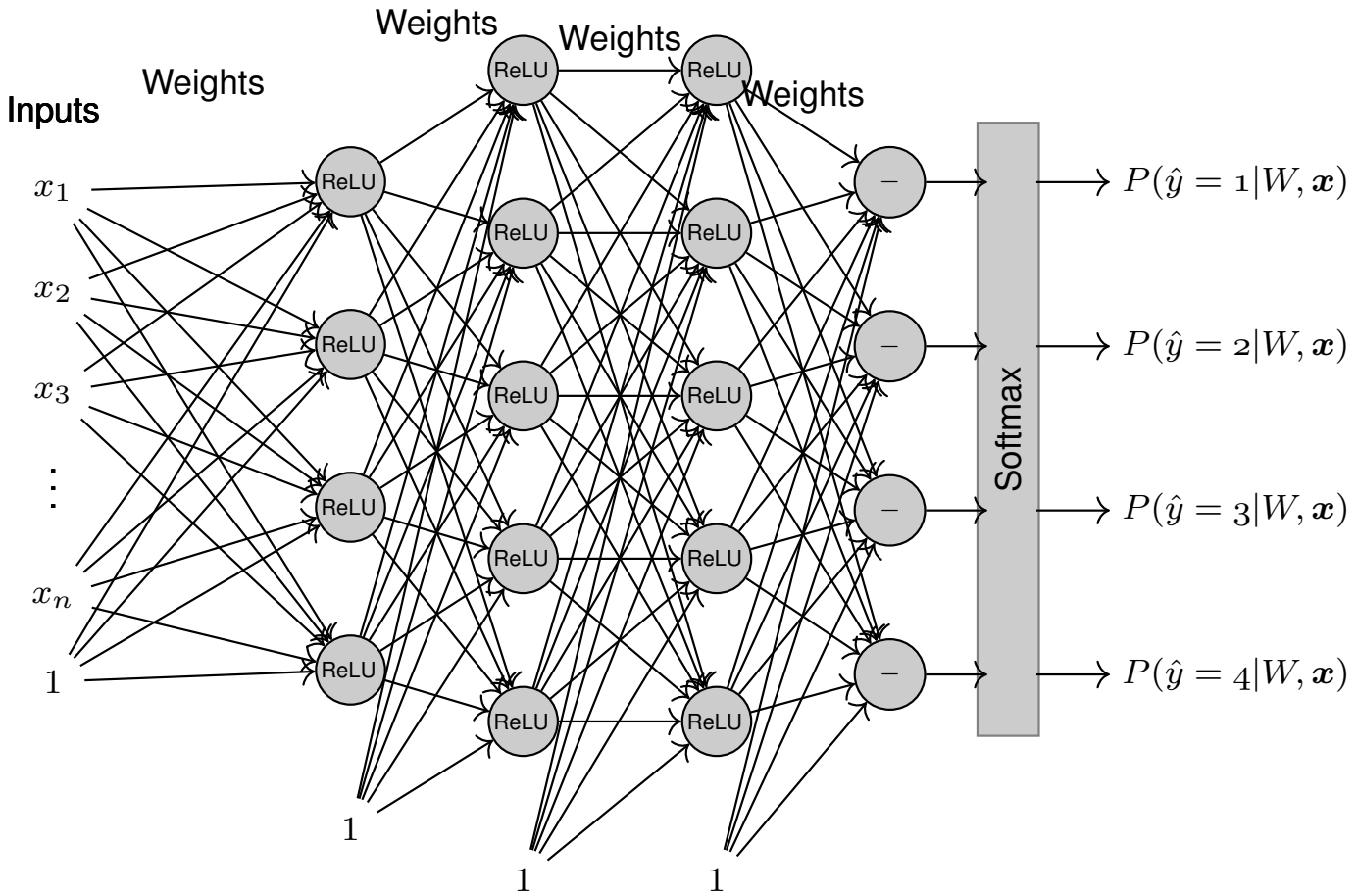
ELU

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



ReLU is a good default choice for most problems

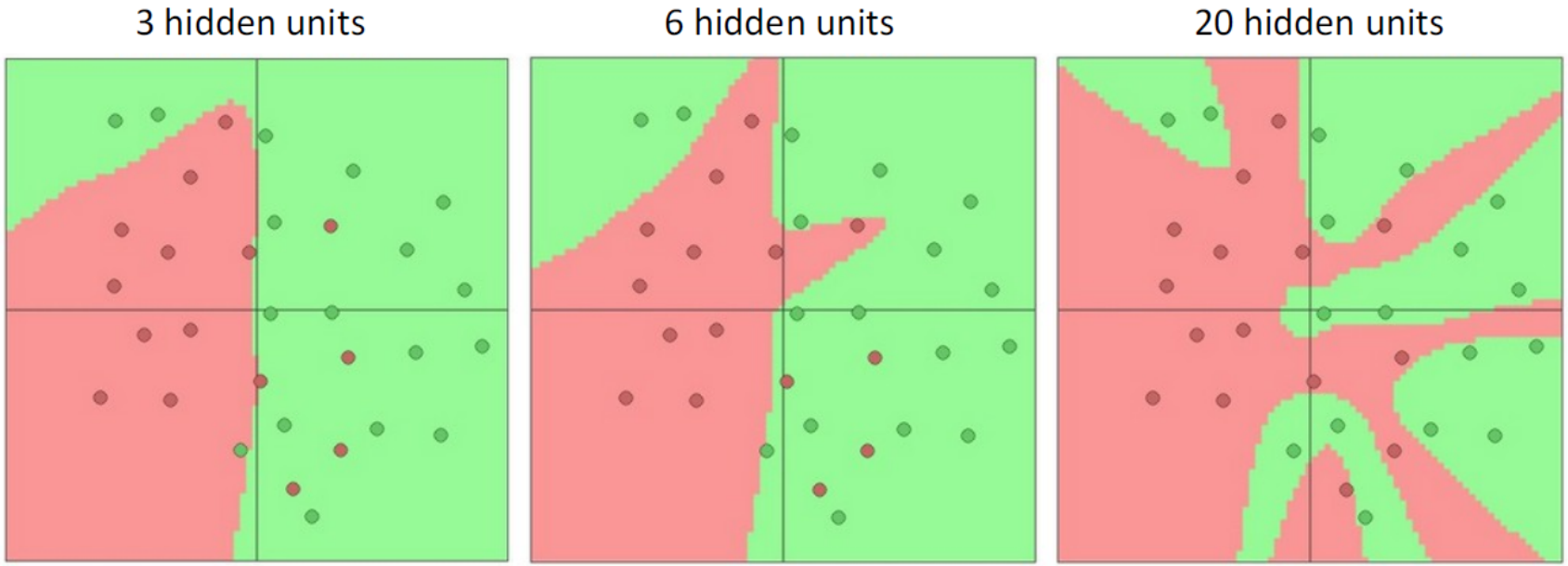
ReLU activation function



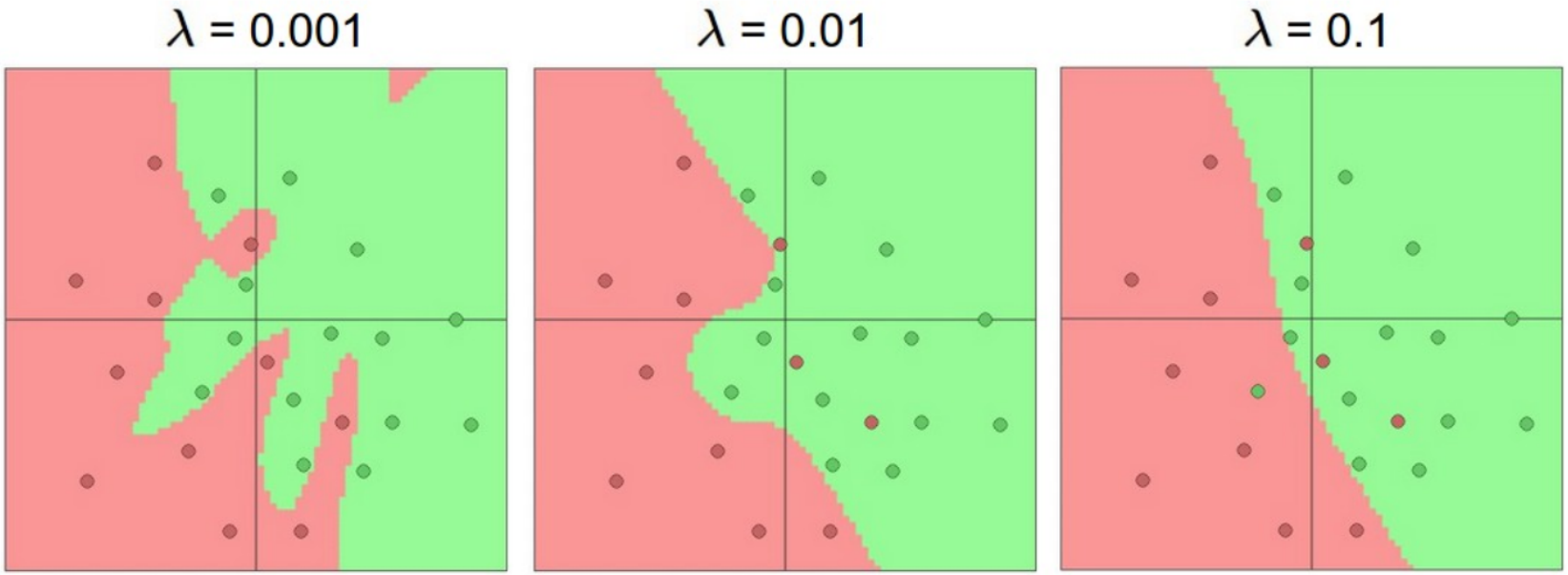
$$\hat{\boldsymbol{y}} = Softmax(W^4(max(0, W^3(max(0, W^2(max(0, W^1\boldsymbol{x} + \boldsymbol{b}^1)) + \boldsymbol{b}^2)) + \boldsymbol{b}^3)) + \boldsymbol{b}^4)$$

Setting the number of layers and their sizes

More hidden units = more capacity



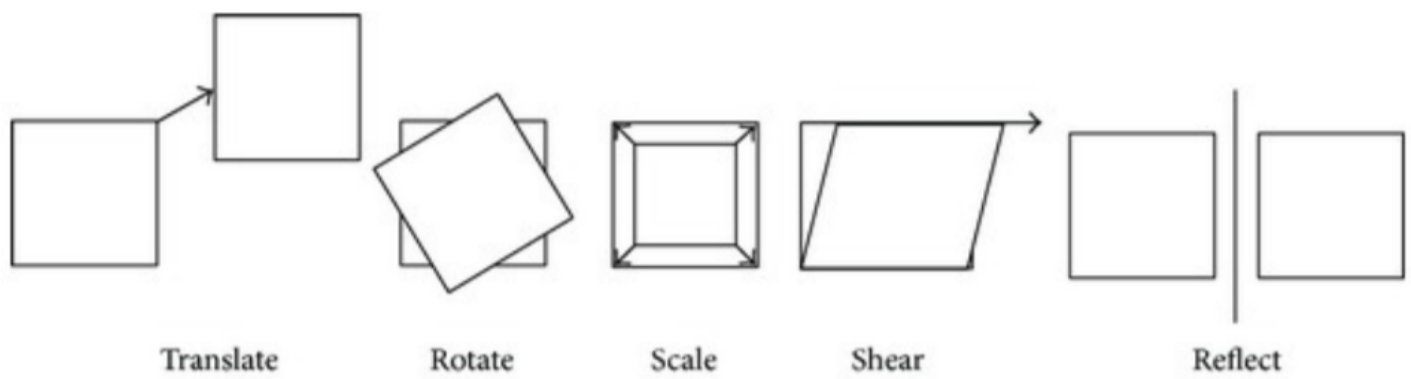
Don’t regularize with size; instead use stronger L2



http://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html

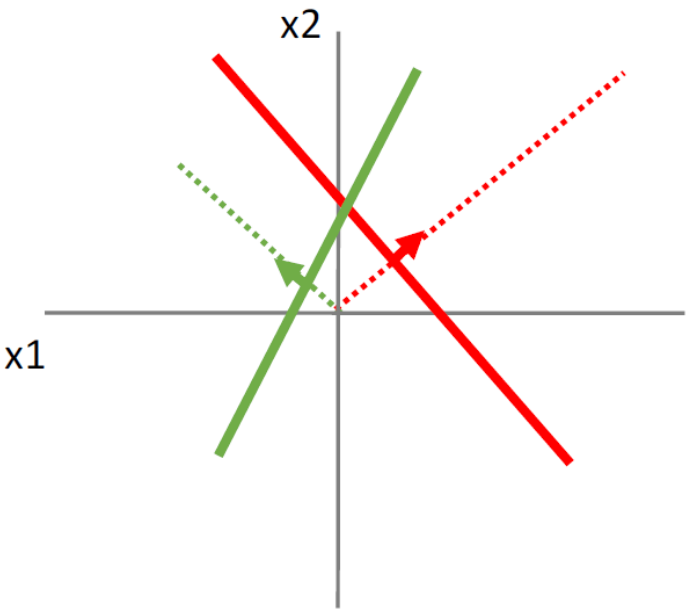
Affine transformation

- Affine transformation
  - Translate
  - Rotate
  - Scale
  - Shear
  - Reflect
- Preserve parallel lines



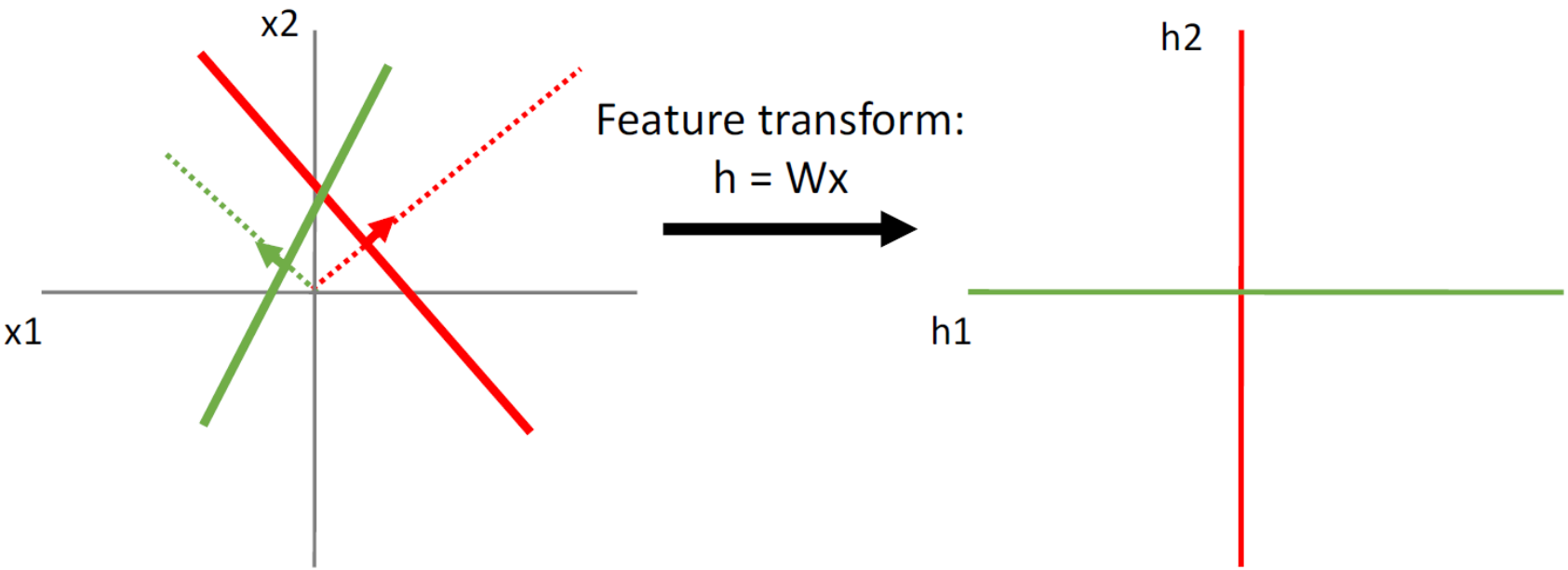
Space warping

- Consider a linear transform: $\boldsymbol{h} = W\boldsymbol{x}$ Where $\boldsymbol{x}, \boldsymbol{h}$ are both 2-dimensional



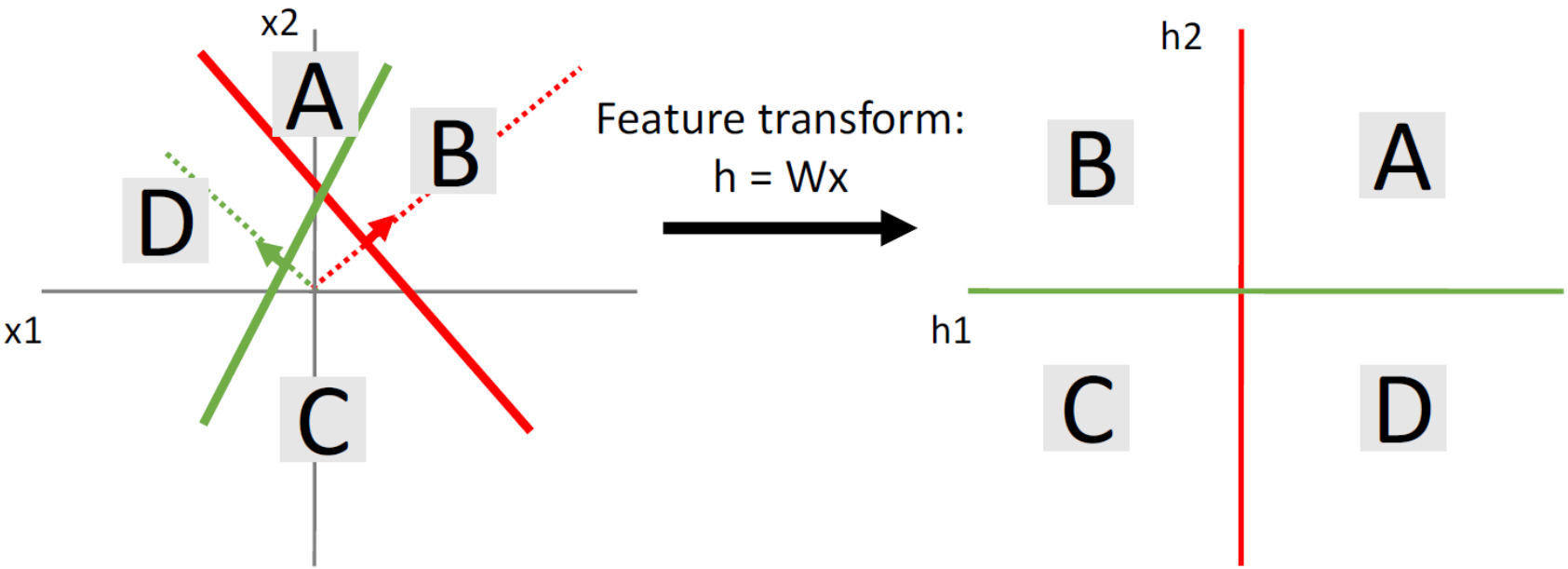
Space warping

- Consider a linear transform: $\boldsymbol{h} = W\boldsymbol{x}$ Where $\boldsymbol{x}, \boldsymbol{h}$ are both 2-dimensional



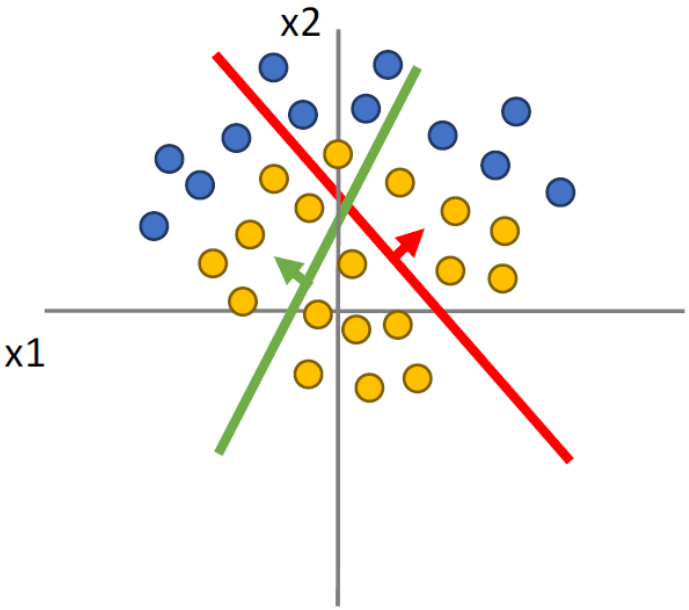
Space warping

- Consider a linear transform: $\boldsymbol{h} = W\boldsymbol{x}$ Where $\boldsymbol{x}, \boldsymbol{h}$ are both 2-dimensional



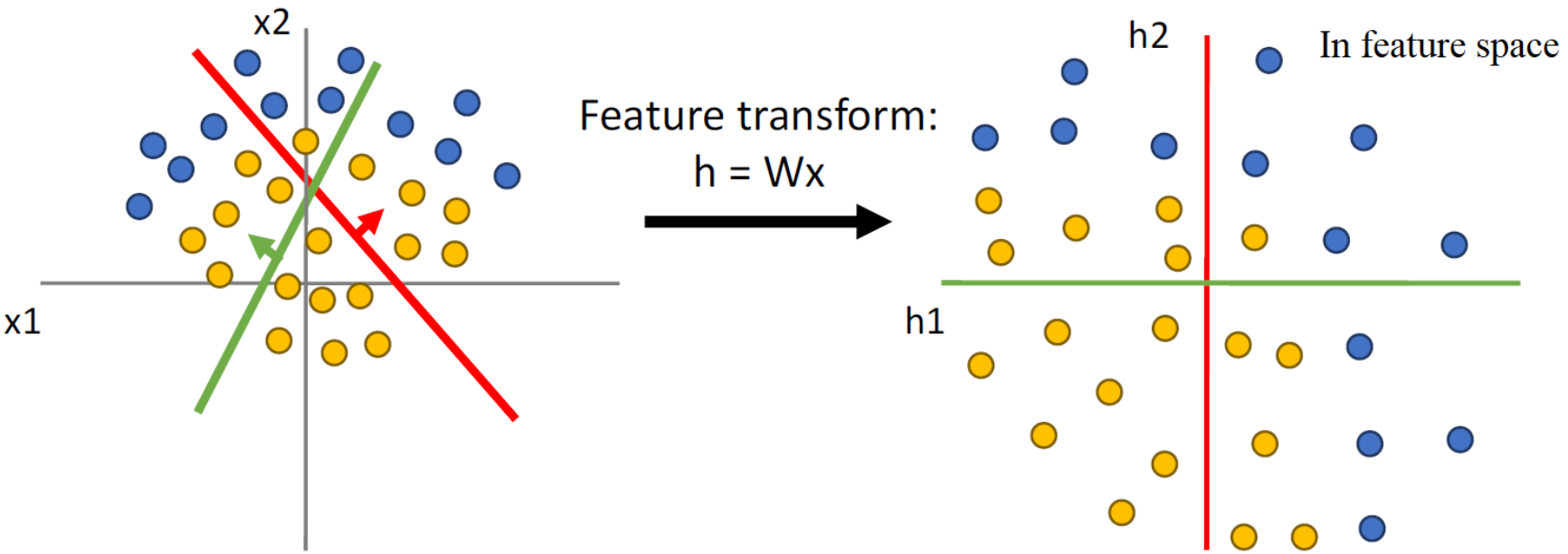
Space warping

- Consider a linear transform: $\boldsymbol{h} = W\boldsymbol{x}$ Where $\boldsymbol{x}, \boldsymbol{h}$ are both 2-dimensional
- Points not linearly separable in original space



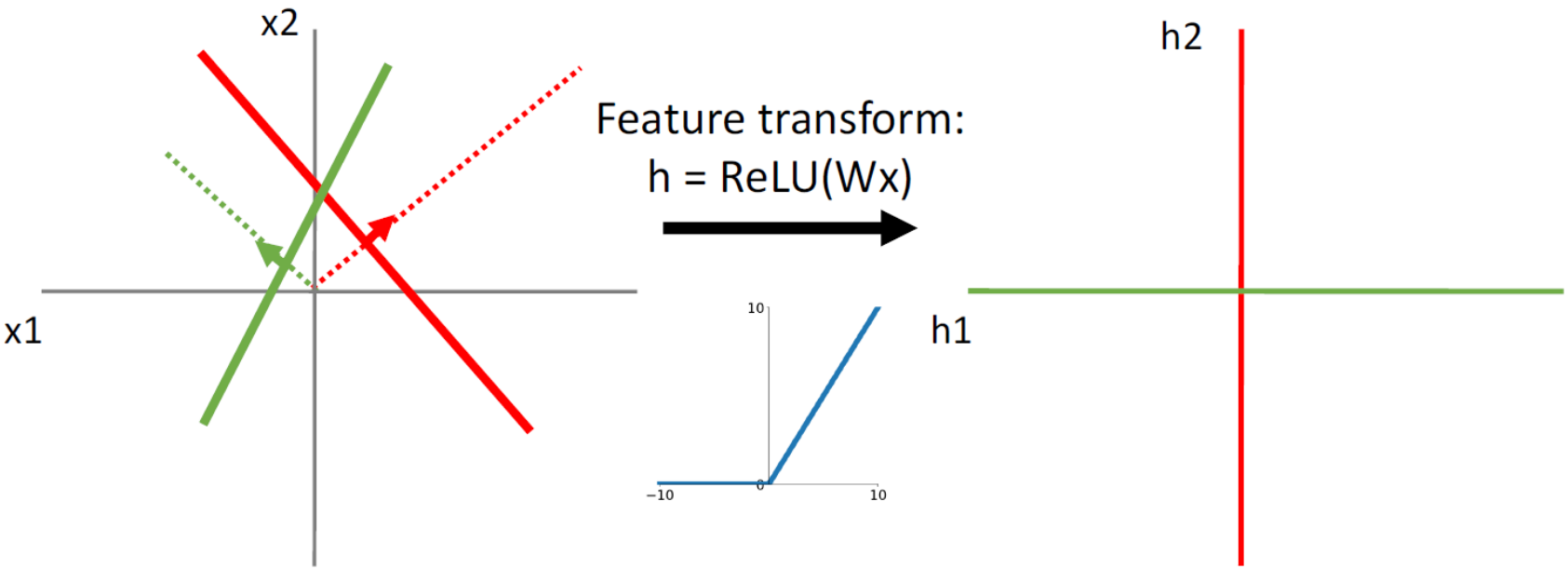
Space warping

- Consider a linear transform: $\boldsymbol{h} = W\boldsymbol{x}$ Where $\boldsymbol{x}, \boldsymbol{h}$ are both 2-dimensional
- Points not linearly separable in original space
  - Not linearly separable in feature space



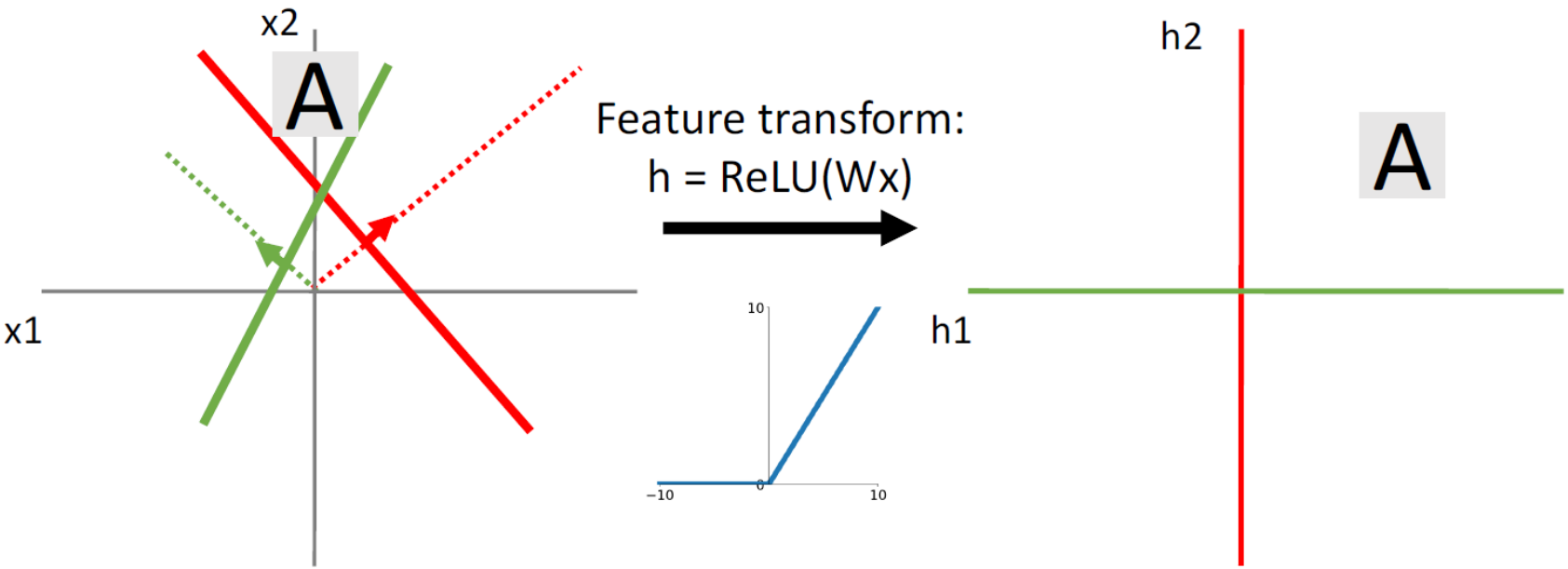
Space warping

- Consider a neural net hidden layer: $\boldsymbol{h} = ReLU(W\boldsymbol{x}) = max(0, W\boldsymbol{x})$.
  - Where x, h are both 2-dimensional.



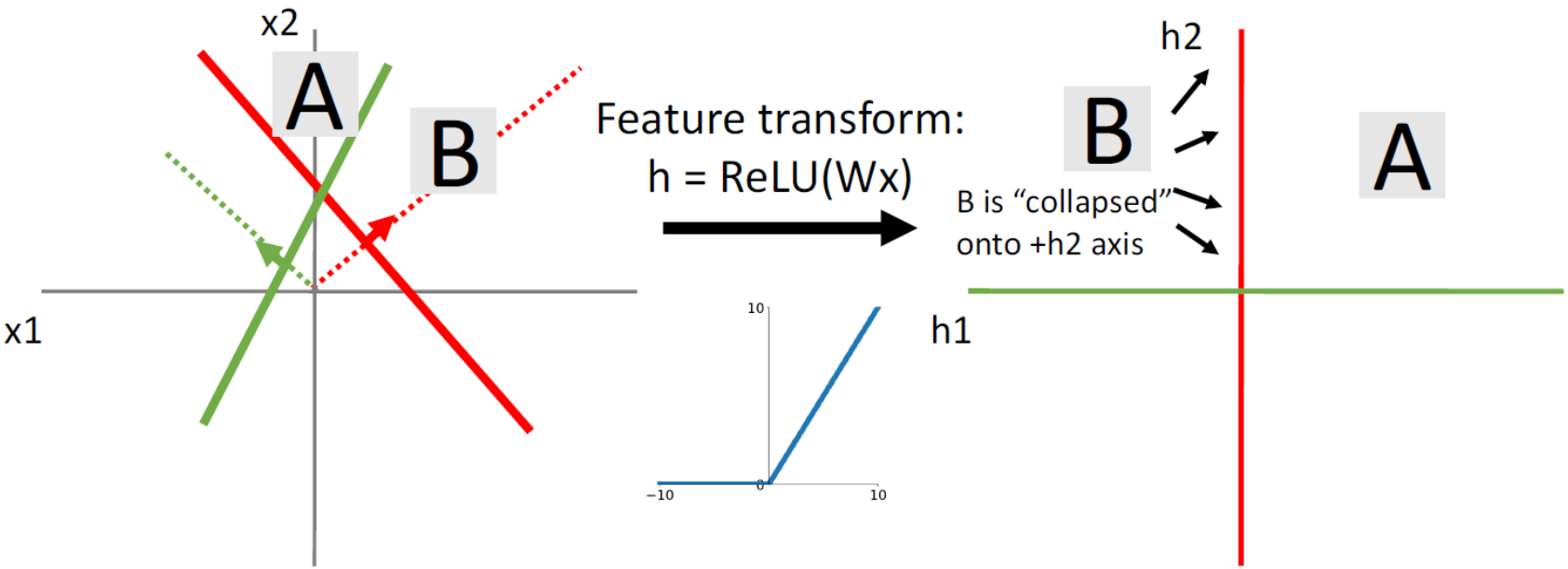
Space warping

- Consider a neural net hidden layer: $\boldsymbol{h} = ReLU(W\boldsymbol{x}) = max(0, W\boldsymbol{x})$.
  - Where x, h are both 2-dimensional.



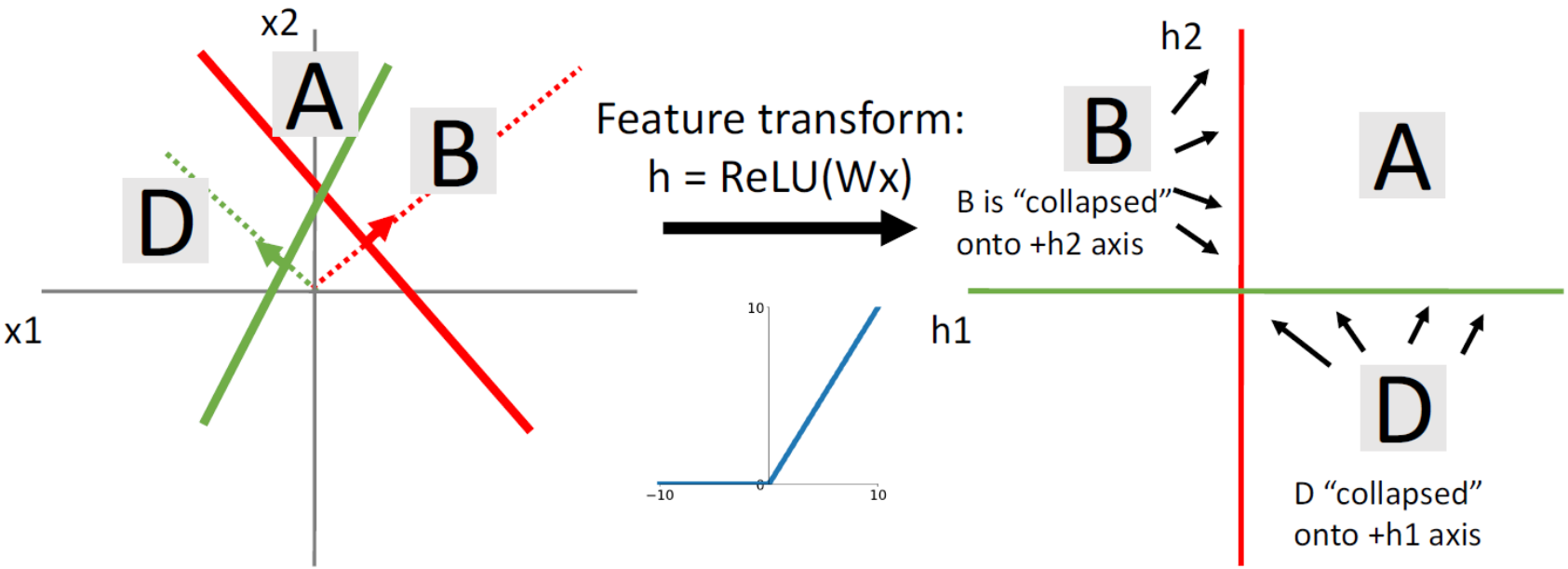
Space warping

- Consider a neural net hidden layer: $\boldsymbol{h} = ReLU(W\boldsymbol{x}) = max(0, W\boldsymbol{x})$.
  - Where x, h are both 2-dimensional.



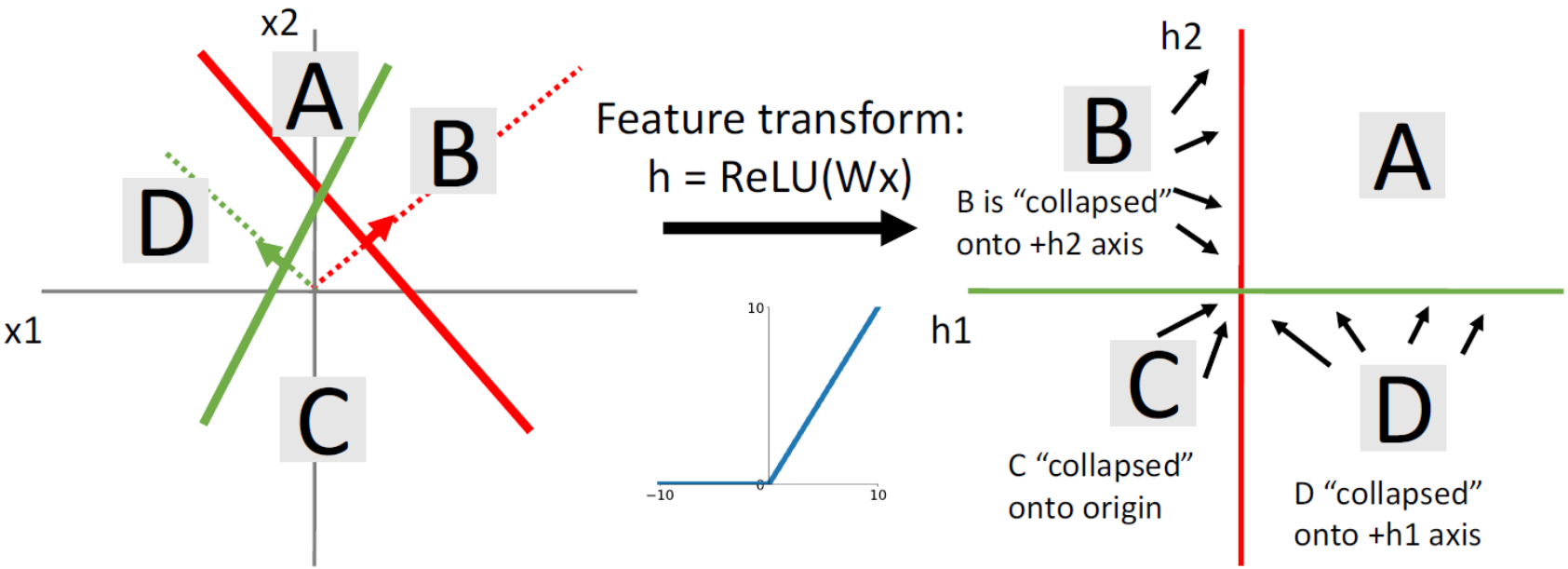
Space warping

- Consider a neural net hidden layer: $\boldsymbol{h} = ReLU(W\boldsymbol{x}) = max(0, W\boldsymbol{x})$.
  - Where x, h are both 2-dimensional.



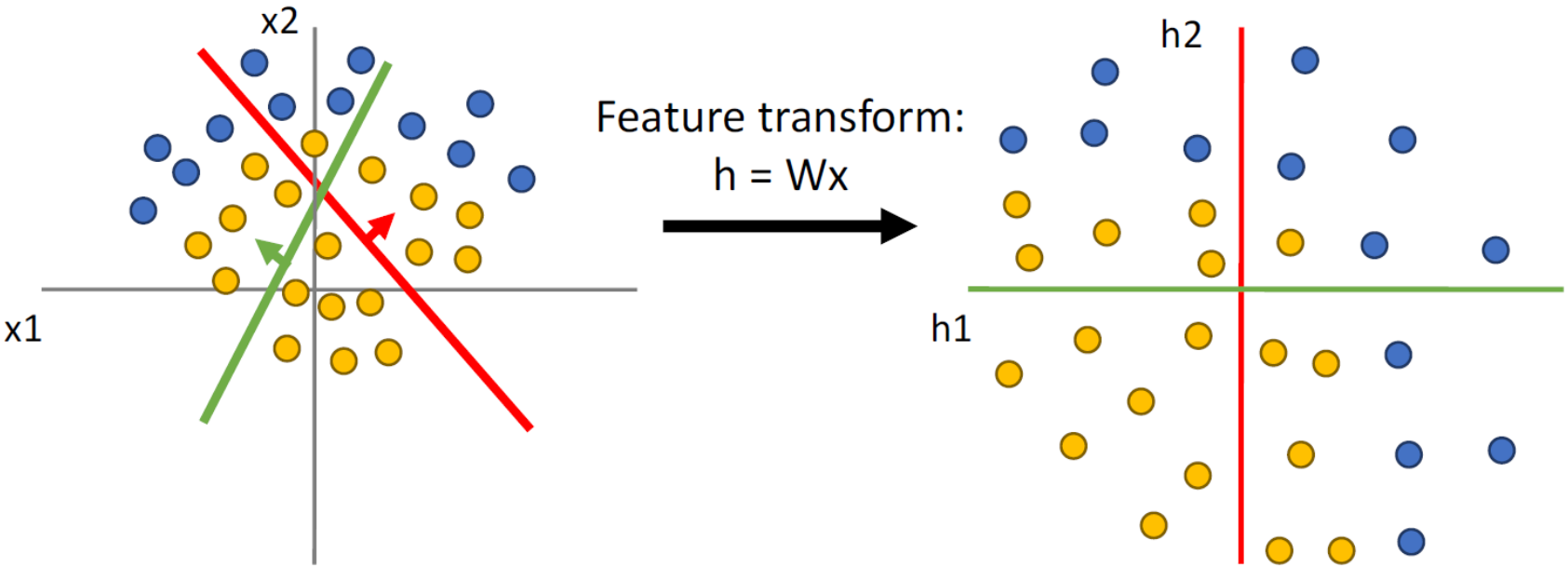
Space warping

- Consider a neural net hidden layer: $\boldsymbol{h} = ReLU(W\boldsymbol{x}) = max(0, W\boldsymbol{x})$.
  - Where x, h are both 2-dimensional.



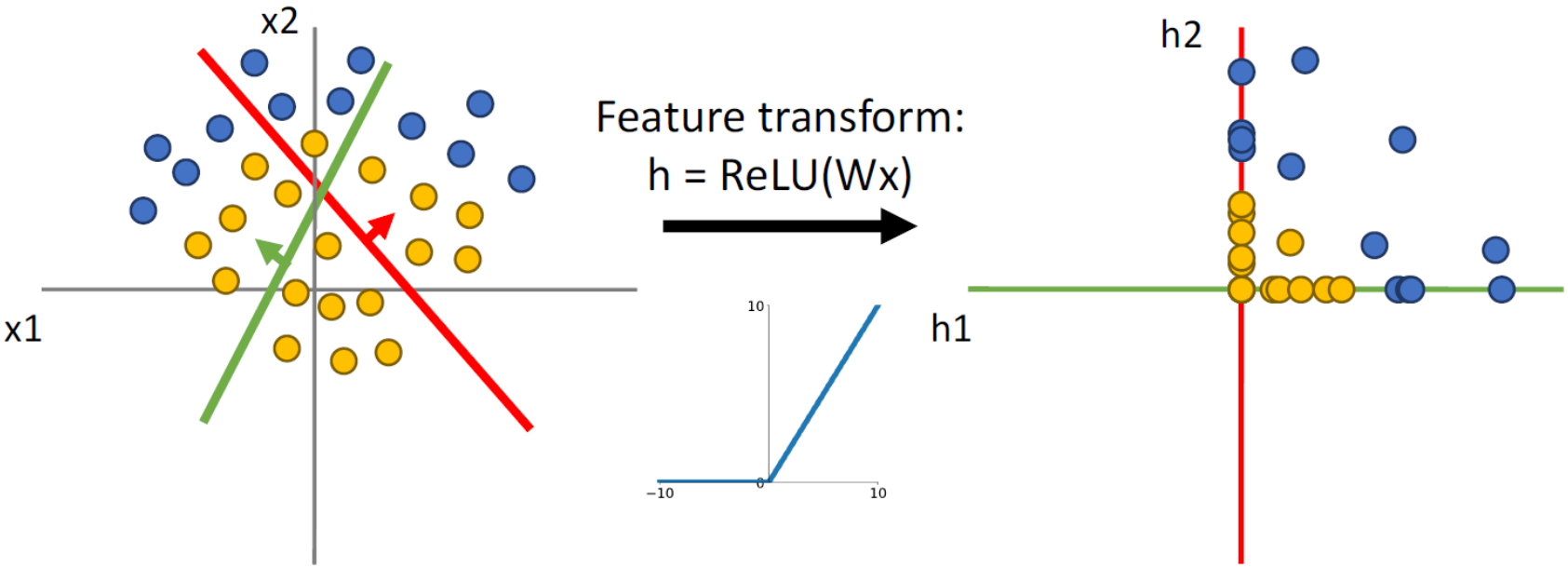
Space warping

- Consider a neural net hidden layer: $\boldsymbol{h} = ReLU(W\boldsymbol{x}) = max(0, W\boldsymbol{x})$.
  - Where x, h are both 2-dimensional.
- Points not linearly separable in original space



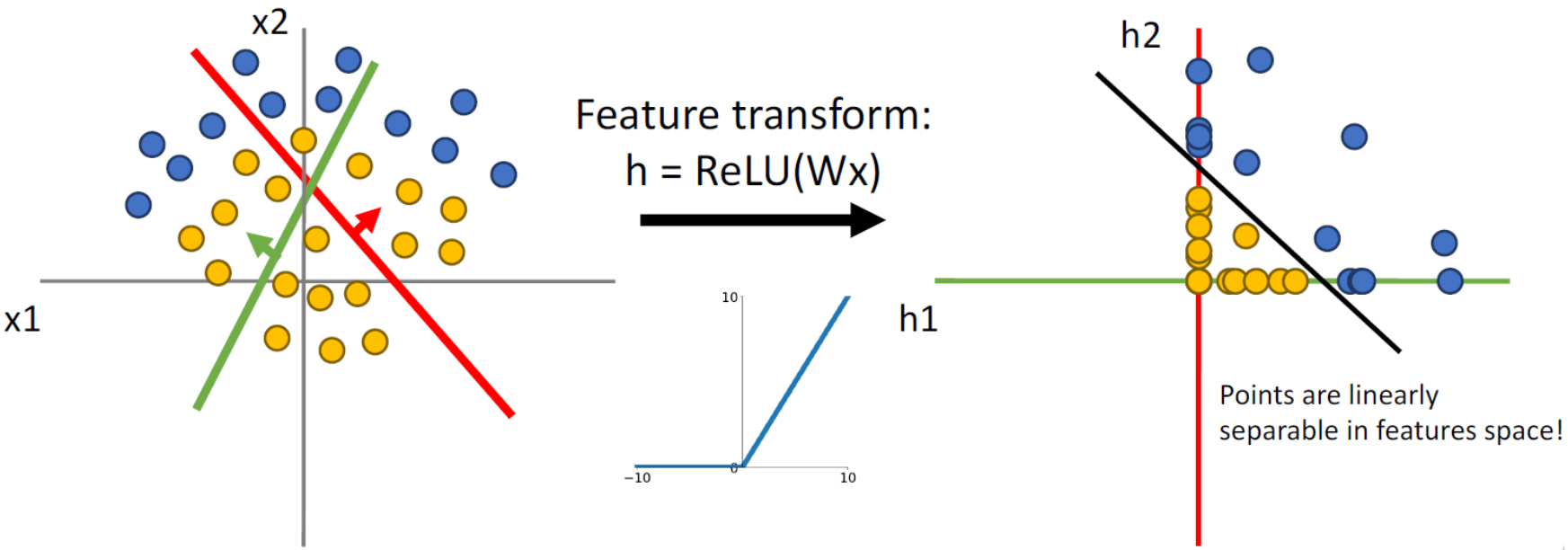
Space warping

- Consider a neural net hidden layer: $\boldsymbol{h} = ReLU(W\boldsymbol{x}) = max(0, W\boldsymbol{x})$.
  - Where x, h are both 2-dimensional.
- Points not linearly separable in original space



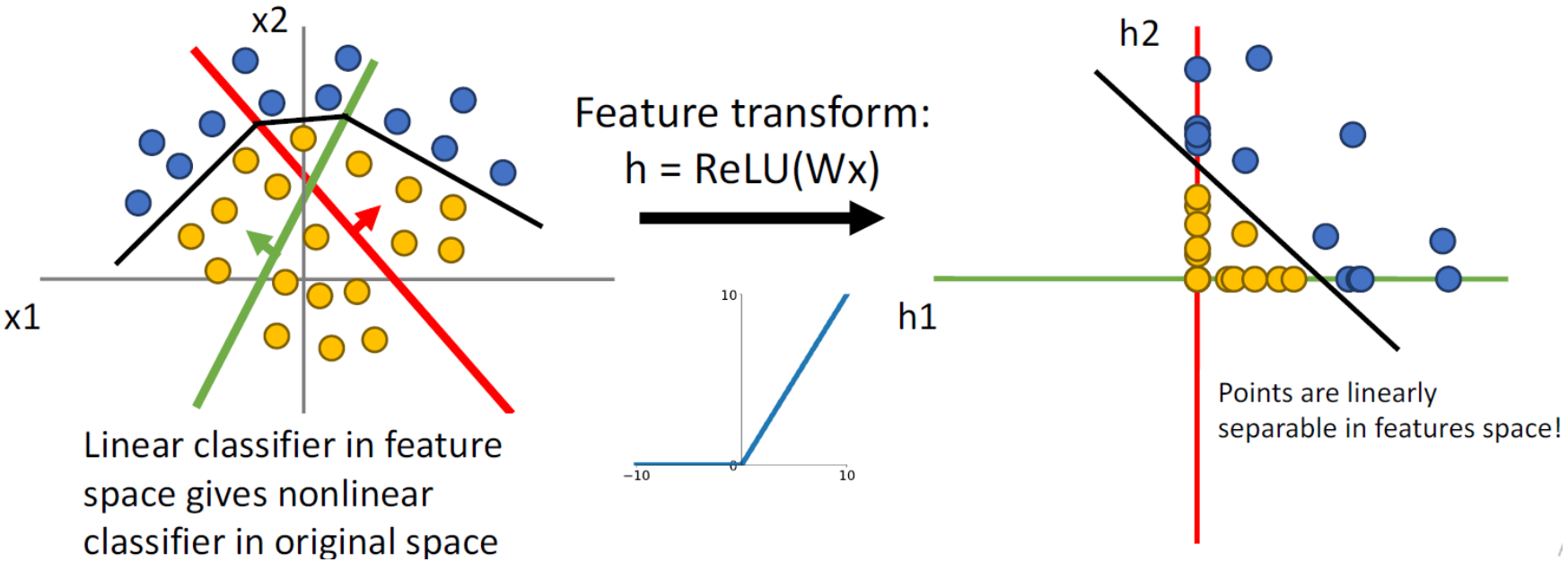
Space warping

- Consider a neural net hidden layer: $\boldsymbol{h} = ReLU(W\boldsymbol{x}) = max(0, W\boldsymbol{x})$.
  - Where x, h are both 2-dimensional.
- Points not linearly separable in original space

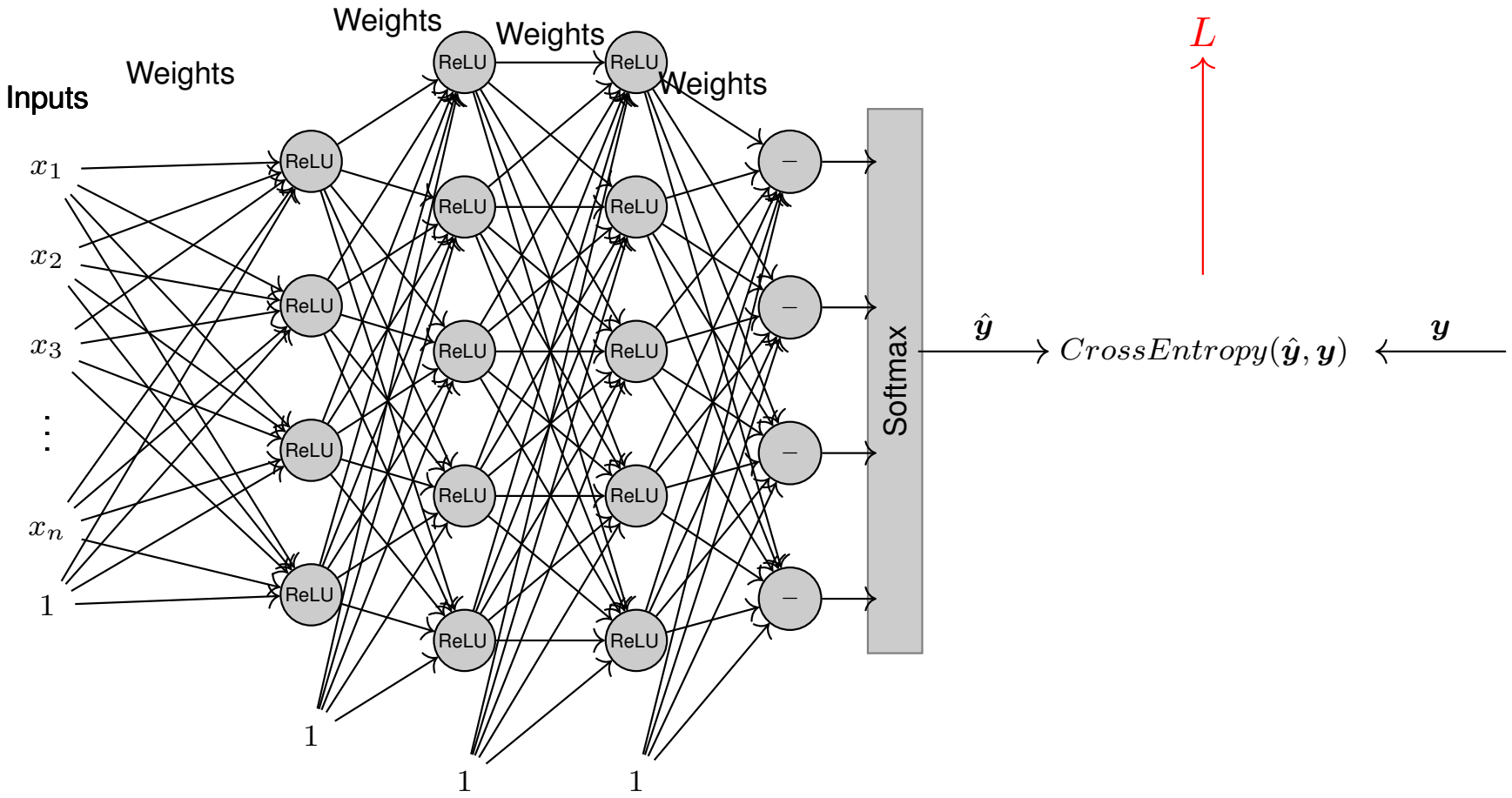


Space warping

- Consider a neural net hidden layer: $\boldsymbol{h} = ReLU(W\boldsymbol{x}) = max(0, W\boldsymbol{x})$.
  - Where x, h are both 2-dimensional.
- Points not linearly separable in original space



Fully connected neural network loss



$$\hat{y} = Softmax(W^4(max(0, W^3(max(0, W^2(max(0, W^1 x + b^1)) + b^2)) + b^3)) + b^4)$$

References

- Deep Learning, Ian Goodfellow, MIT Press, Ch.6, 7 ,and 8