



Neural Networks

A. M. Sadeghzadeh, Ph.D.

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



October 4, 2024

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

Todays agenda

1 Recap

2 Softmax Classifier

3 Deep Neural Networks

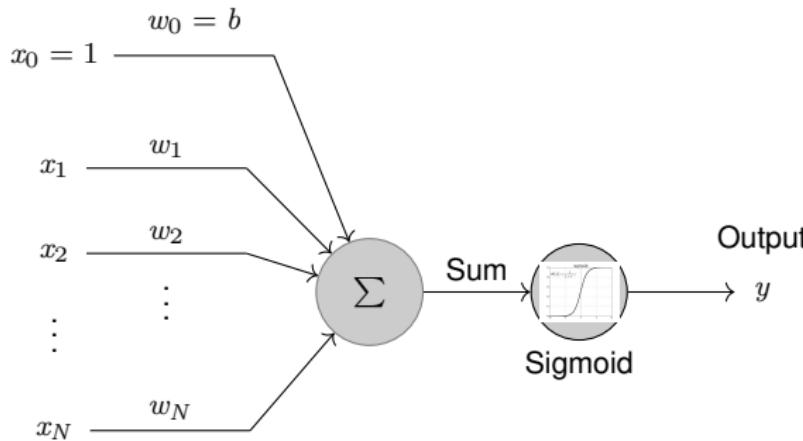
Recap

Logistic regression

It is the perceptron with a sigmoid activation

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

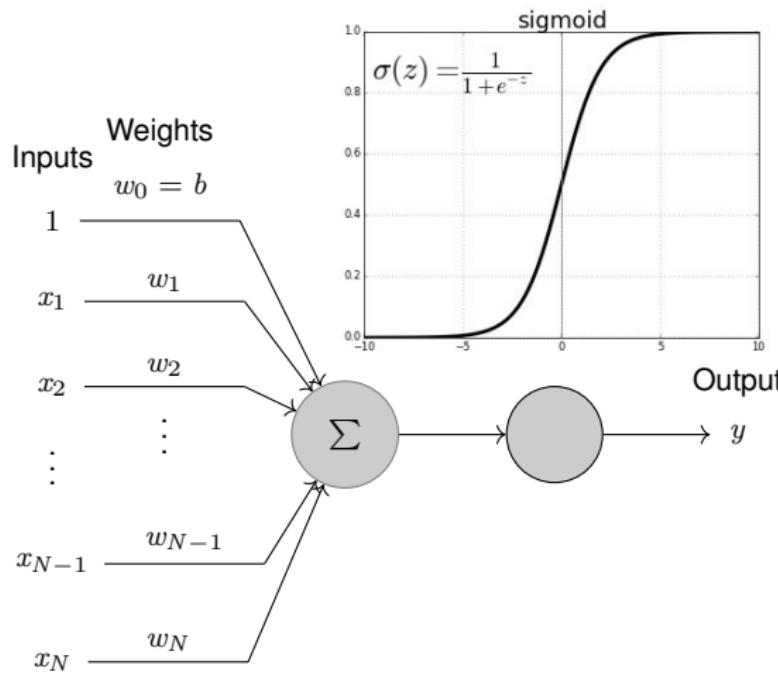
Inputs Weights



Logistic regression

It is the perceptron with a sigmoid activation

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



$$z = \sum_i w_i x_i$$

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

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

Vectorization

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

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

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:

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

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

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

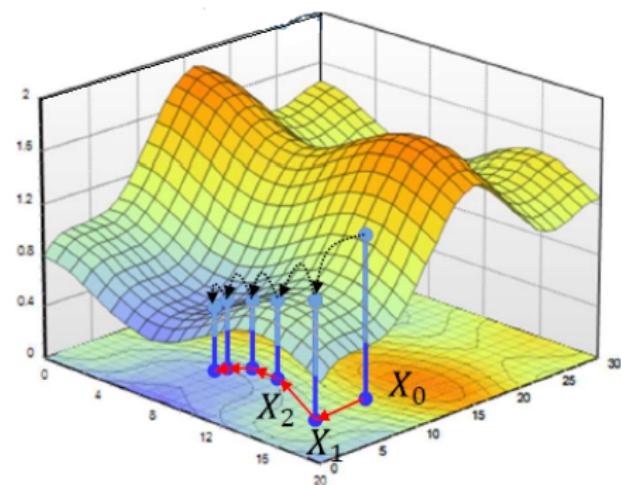
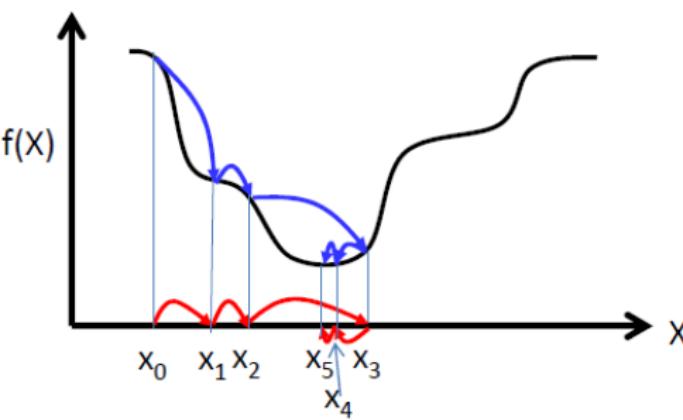
Iterative solutions (Gradient Descent)

- 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

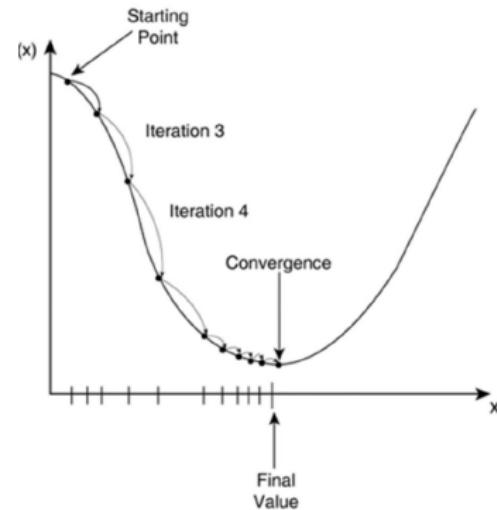
$$\mathbf{x}^{t+1} = \mathbf{x}^t - \eta \nabla_{\mathbf{x}} f(\mathbf{x}^t)$$

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

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

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

- k is a hyperparameter.



Regularization

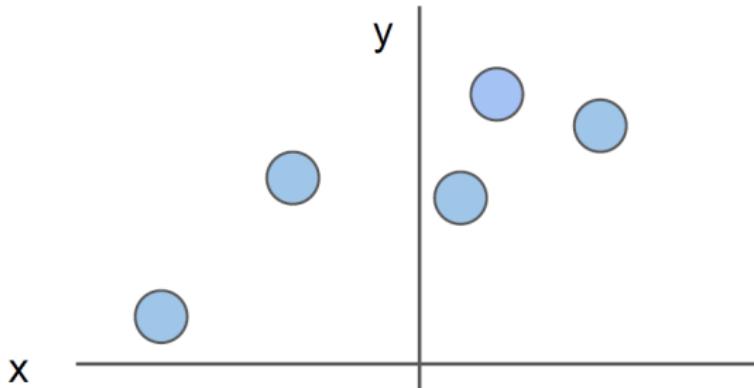
- **Regularization** is any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error.
 - We can give a learning algorithm a **preference** for one solution in its hypothesis space to another. This means that both functions are eligible,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

Regularization

- **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.

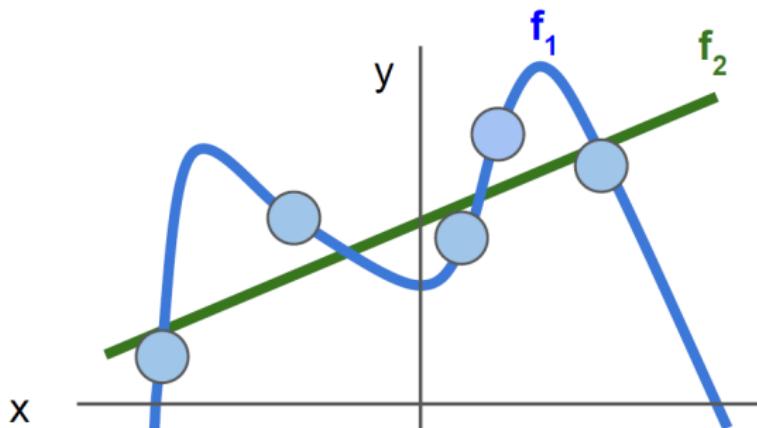
Regularization

- How do we choose between hypotheses?
 - Occams Razor: Among multiple competing hypotheses, the simplest is the best, (William of Ockham 1285-1347)
 - Avoid overfitting: prefer simple models that generalize better
- Regularization intuition: toy example training data



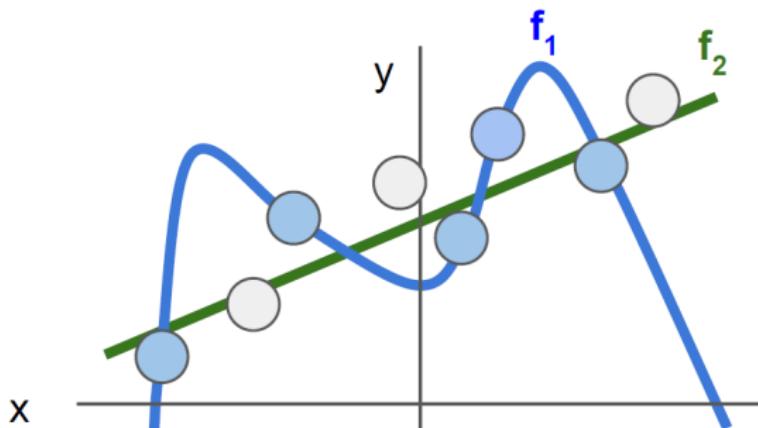
Regularization

- How do we choose between hypotheses?
 - Occams Razor: Among multiple competing hypotheses, the simplest is the best, (William of Ockham 1285-1347)
 - Avoid overfitting: prefer simple models that generalize better
- Regularization intuition: Prefer Simpler Models



Regularization

- How do we choose between hypotheses?
 - Occams Razor: Among multiple competing hypotheses, the simplest is the best, (William of Ockham 1285-1347)
 - Avoid overfitting: prefer simple models that generalize better
- Regularization: Prefer Simpler Models



- Regularization pushes against fitting the data too well so we don't fit noise in the data

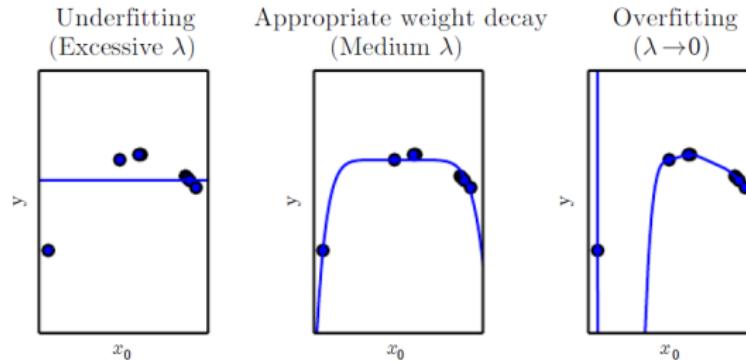
Regularization

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

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

- λ : regularization strength (hyperparameter)
- Simple examples

- L1 regularization: $R(\mathbf{w}) = \sum_i |w_i|$
- L2 regularization: $R(\mathbf{w}) = \sum_i w_i^2$



Regularization

- Which of w_1 or w_2 will the L_2 regularizer prefer? (Training loss is the same)
 - L_2 regularization likes to spread out the weights

$$\mathbf{x}^{(i)} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \quad \mathbf{w}_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad \mathbf{w}_2 = \begin{bmatrix} 0.25 \\ 0.25 \\ 0.25 \\ 0.25 \end{bmatrix}$$

$$\mathbf{w}_1^T \mathbf{x}^{(i)} = \mathbf{w}_2^T \mathbf{x}^{(i)} = 1$$

Regularization

- Which of \mathbf{w}_1 or \mathbf{w}_2 will the L_2 regularizer prefer? (Training loss is the same)
 - L_2 regularization likes to spread out the weights

$$\mathbf{x}^{(i)} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \quad \mathbf{w}_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad \mathbf{w}_2 = \begin{bmatrix} 0.25 \\ 0.25 \\ 0.25 \\ 0.25 \end{bmatrix}$$

$$\mathbf{w}_1^T \mathbf{x}^{(i)} = \mathbf{w}_2^T \mathbf{x}^{(i)} = 1$$

$$L2(\mathbf{w}_1) = 1, \quad L2(\mathbf{w}_2) = 0.25$$

Regularization

- Which of \mathbf{w}_1 or \mathbf{w}_2 will the L_2 regularizer prefer? (Training loss is the same)
 - L_2 regularization likes to spread out the weights

$$\mathbf{x}^{(i)} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \quad \mathbf{w}_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad \mathbf{w}_2 = \begin{bmatrix} 0.25 \\ 0.25 \\ 0.25 \\ 0.25 \end{bmatrix}$$

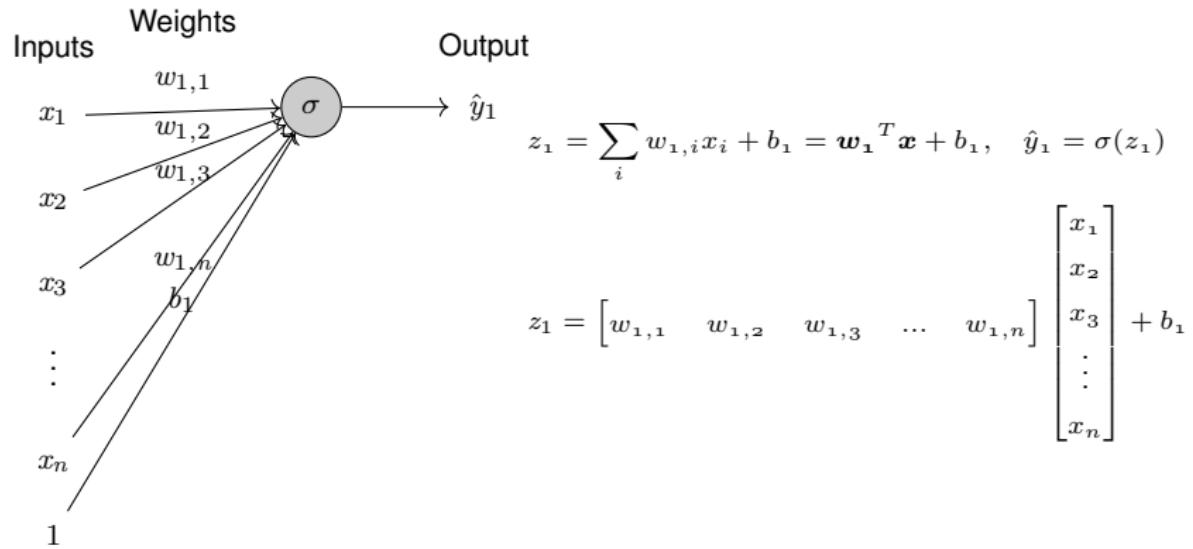
$$\mathbf{w}_1^T \mathbf{x}^{(i)} = \mathbf{w}_2^T \mathbf{x}^{(i)} = 1$$

$$L2(\mathbf{w}_1) = 1, \quad L2(\mathbf{w}_2) = 0.25$$

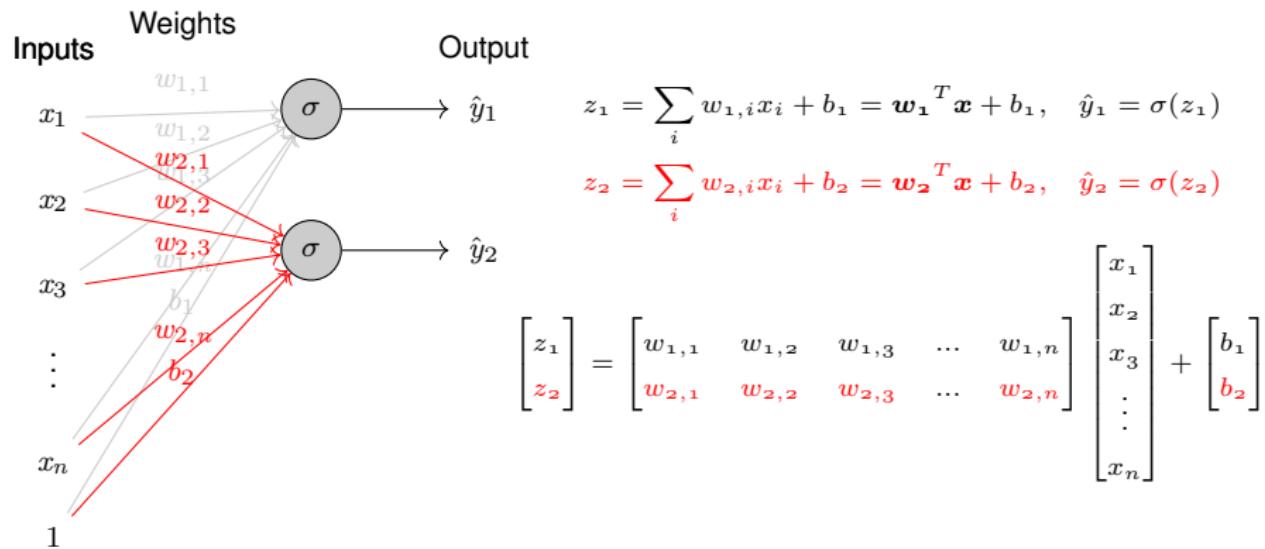
- More complex regularization methods
 - Dropout
 - MixUp
 - Stochastic depth

Softmax Classifier

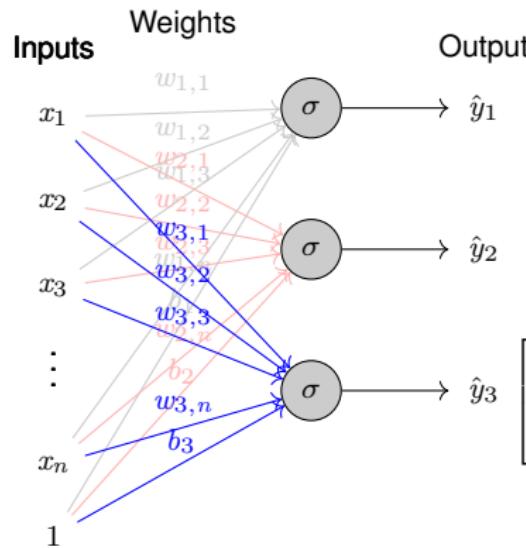
Multinomial logistic regression (softmax classifier)



Multinomial logistic regression (softmax classifier)



Multinomial logistic regression (softmax classifier)



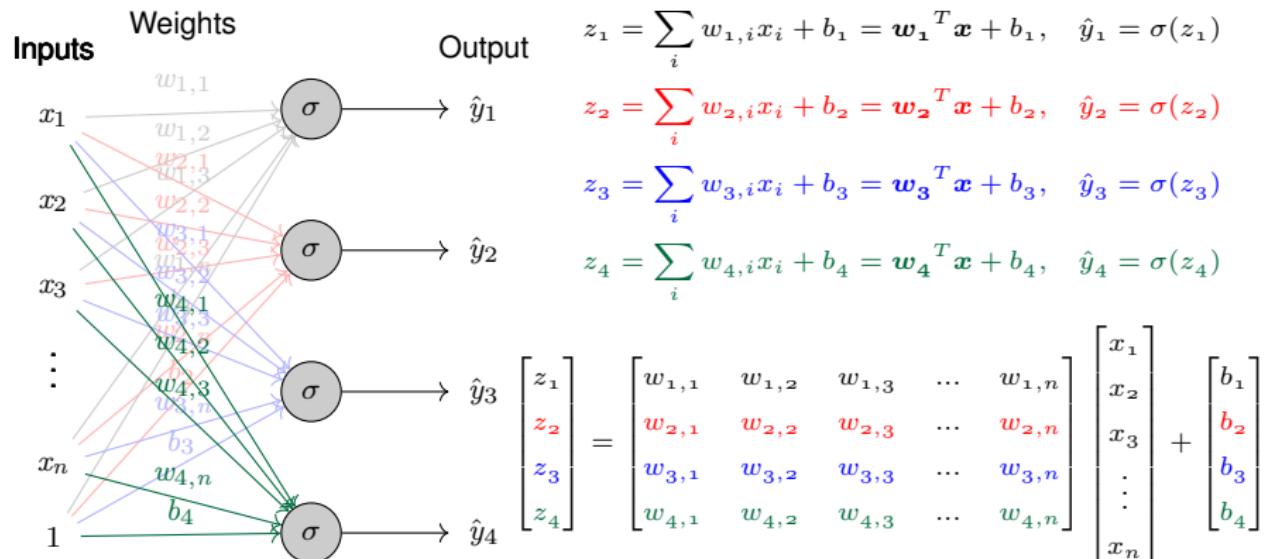
$$z_1 = \sum_i w_{1,i} x_i + b_1 = \mathbf{w}_1^T \mathbf{x} + b_1, \quad \hat{y}_1 = \sigma(z_1)$$

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

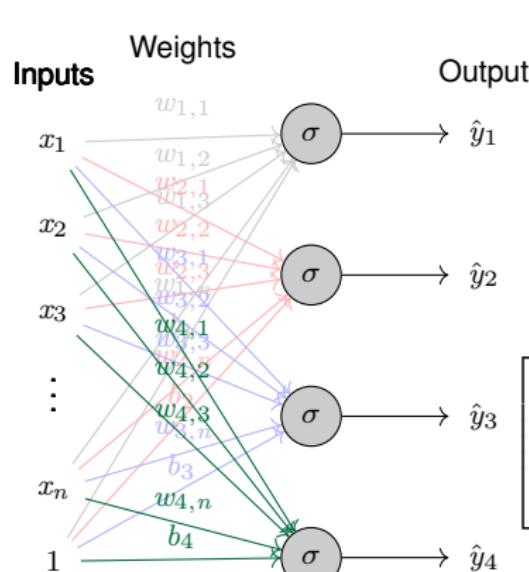
$$z_3 = \sum_i w_{3,i} x_i + b_3 = \mathbf{w}_3^T \mathbf{x} + b_3, \quad \hat{y}_3 = \sigma(z_3)$$

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

Multinomial logistic regression (softmax classifier)



Multinomial logistic regression (softmax classifier)



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

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

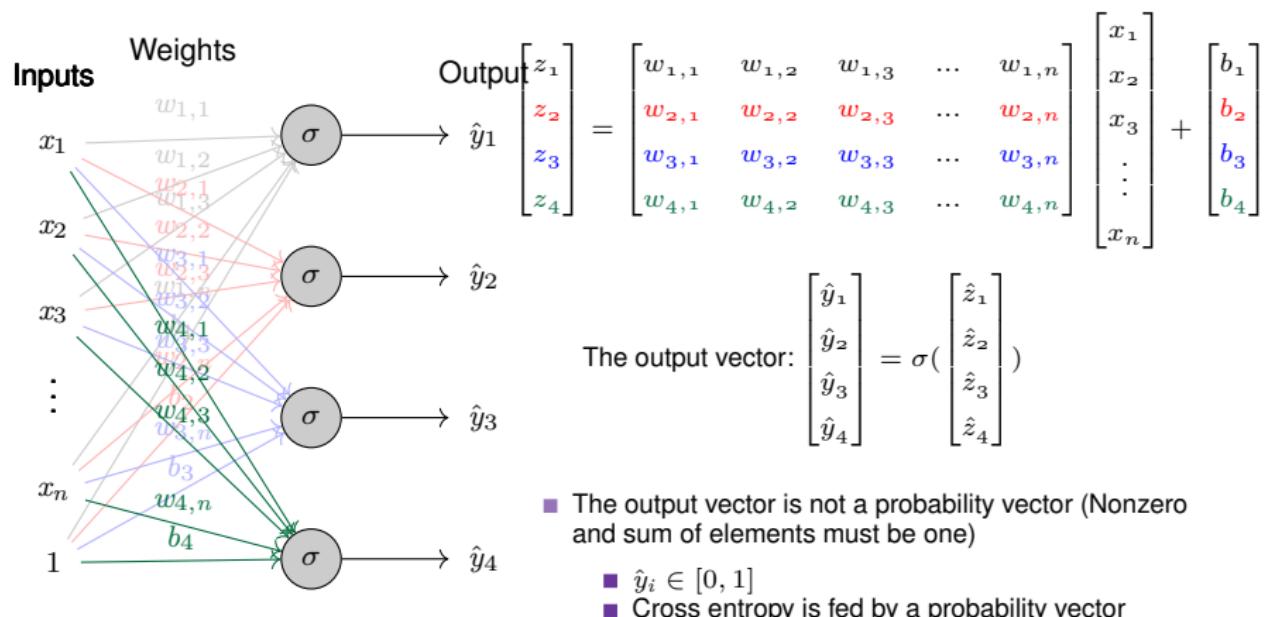
$$z_3 = \sum_i w_{3,i} x_i + b_3 = \mathbf{w}_3^T \mathbf{x} + b_3, \quad \hat{y}_3 = \sigma(z_3)$$

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

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

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

Multinomial logistic regression (softmax classifier)



- 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:

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

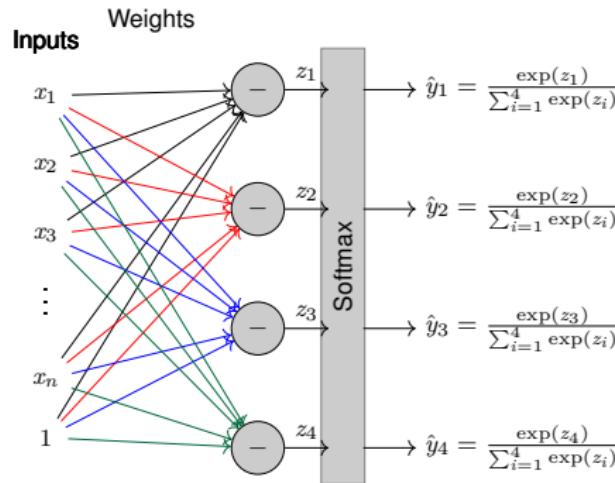
$$\text{Dog: } \begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix}^T$$

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

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

- 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:

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

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

The output of classifier: $\operatorname{argmax}_j \hat{y}_j$

Loss function

- Cross entropy:

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

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

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

Loss function

- Cross entropy:

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

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

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

example:



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

Loss function

- Cross entropy:

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

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

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

example:



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

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

Loss function

- Cross entropy:

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

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

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

example:



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

Loss function

- Cross entropy:

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

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

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

example:



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

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

Example

matrix multiply + bias offset

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	0.0
22	0.2
-44	-0.3
56	0.28

x_i

$$y_i \boxed{3}$$

-2.85	0.058
0.86	2.36
0.28	1.32

=

exp	→
0.058	0.016

cross-entropy loss (Softmax)

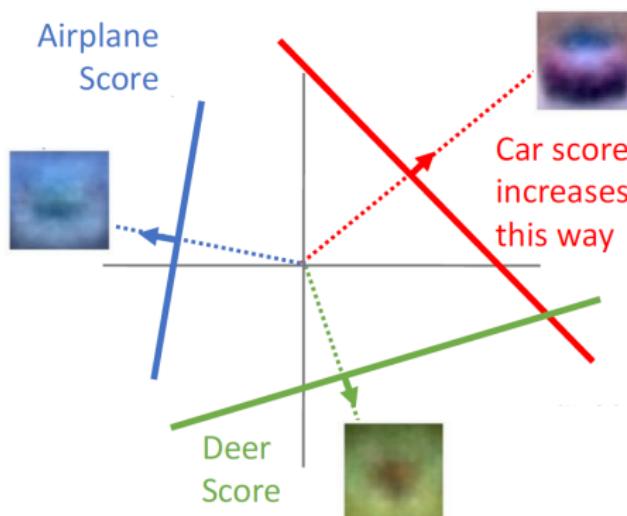
normalize	→
(to sum to one)	0.016
	0.631
	0.353

$$-\log(0.353)$$

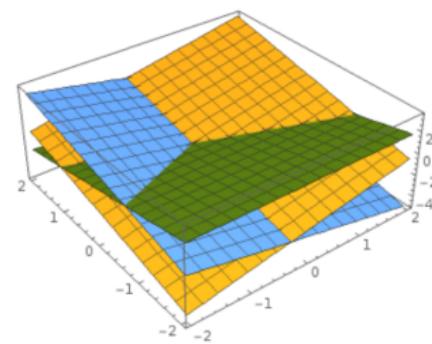
$$=$$

$$1.04$$

Softmax classifier



Hyperplanes carving up a high-dimensional space



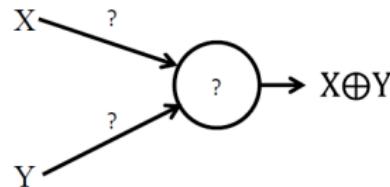
Plot created using Wolfram Cloud

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

Deep Neural Networks

Xor

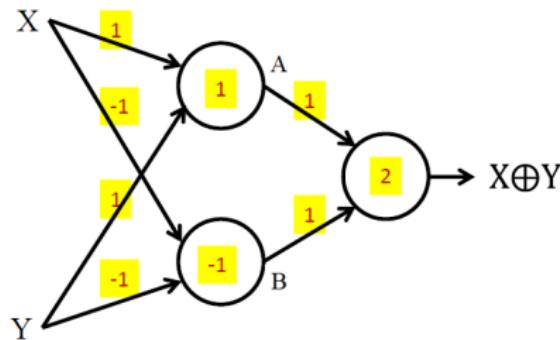
No solution for XOR!
Not universal!



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

Solution: add layers

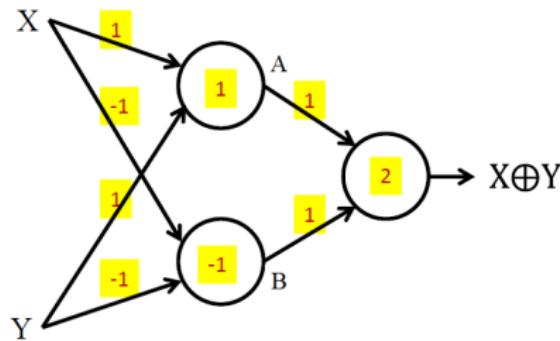
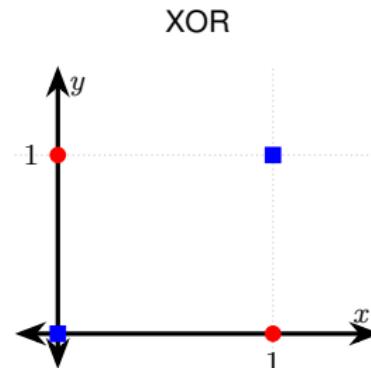
XOR



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

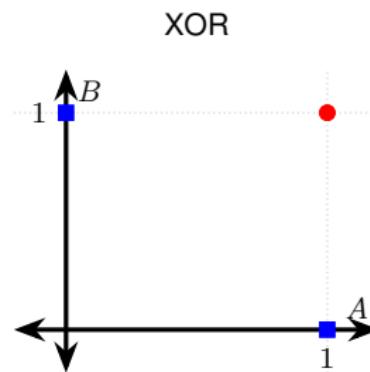
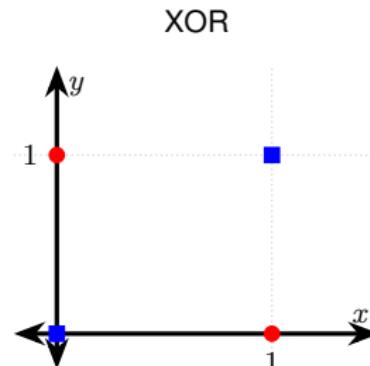
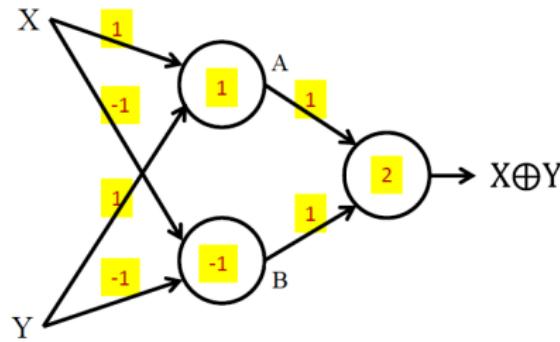
Transformation

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



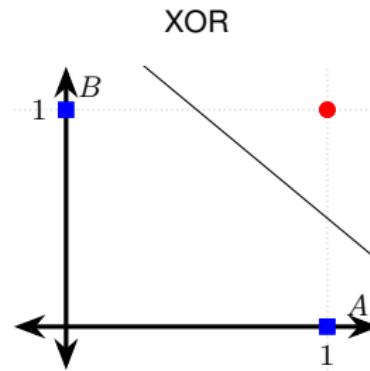
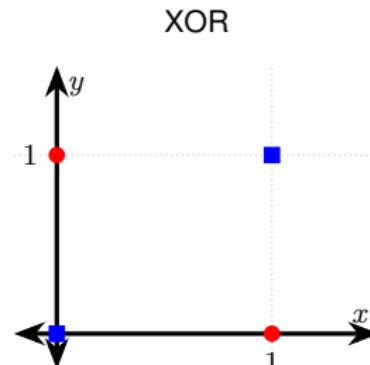
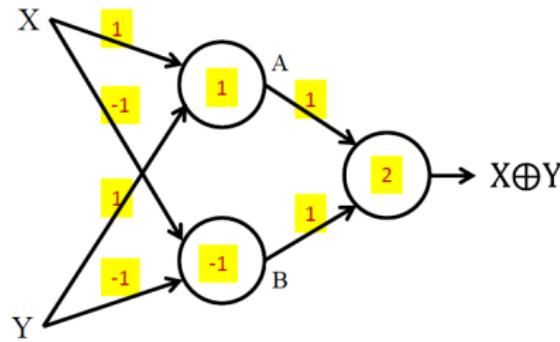
Transformation

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

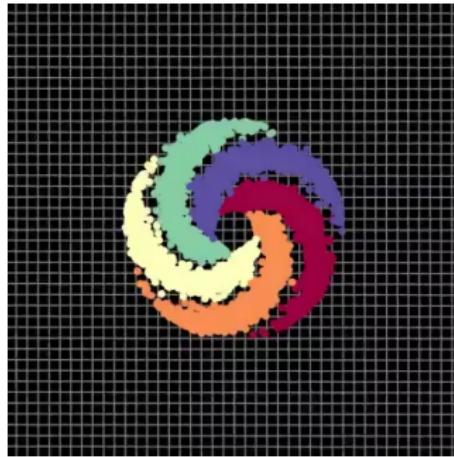


Transformation

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

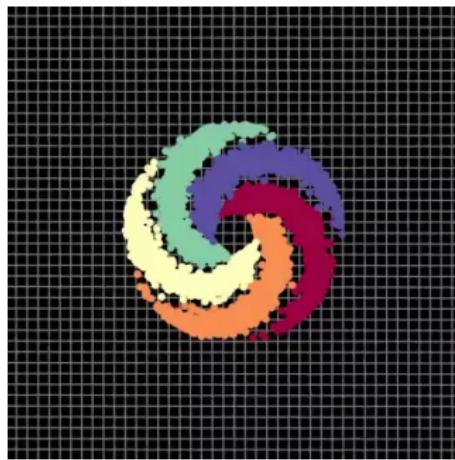


Needs more layers

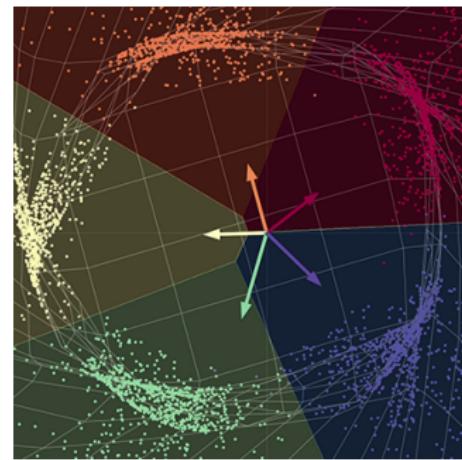


Input points, pre-network

Needs more layers



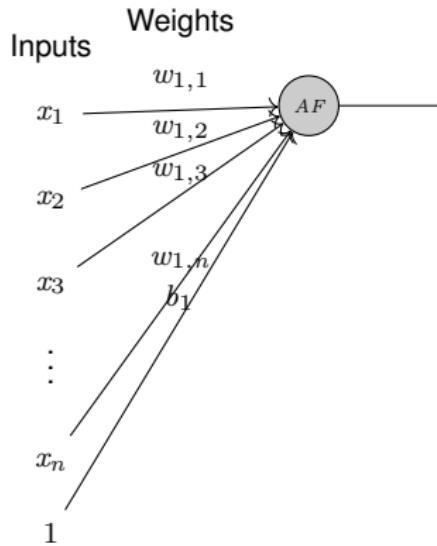
Input points, pre-network



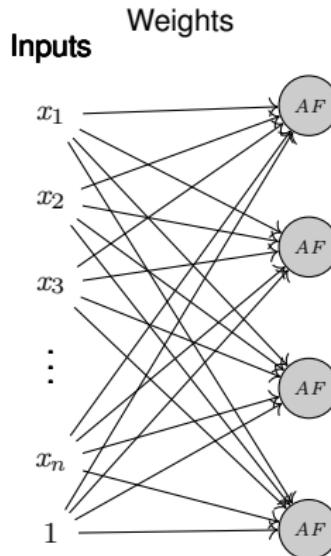
Output points, post-network

(Canziani, 2020)

Fully connected neural networks

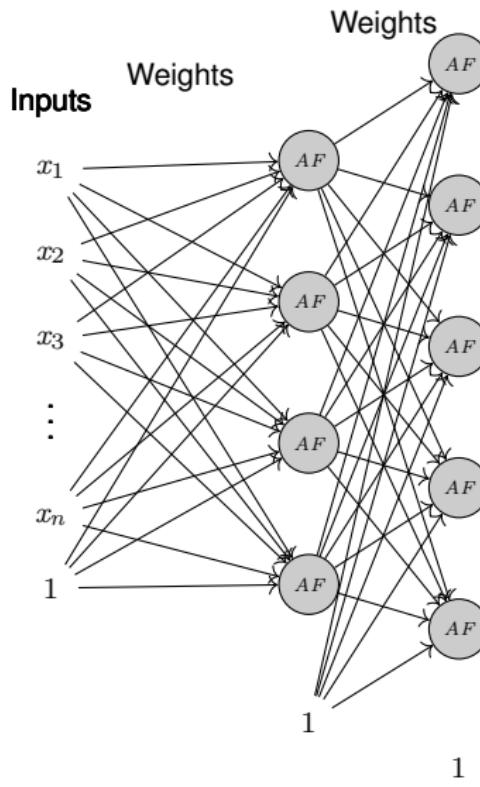


Fully connected neural networks

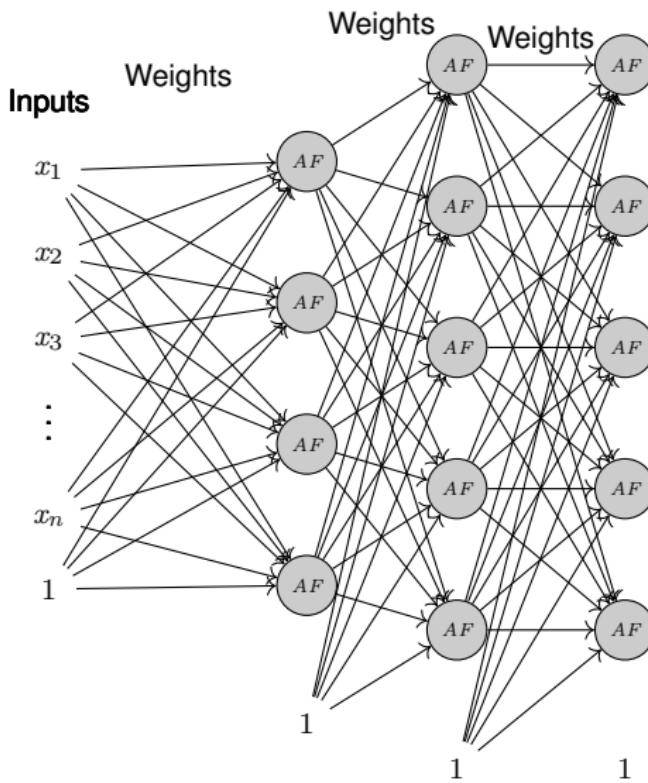


1

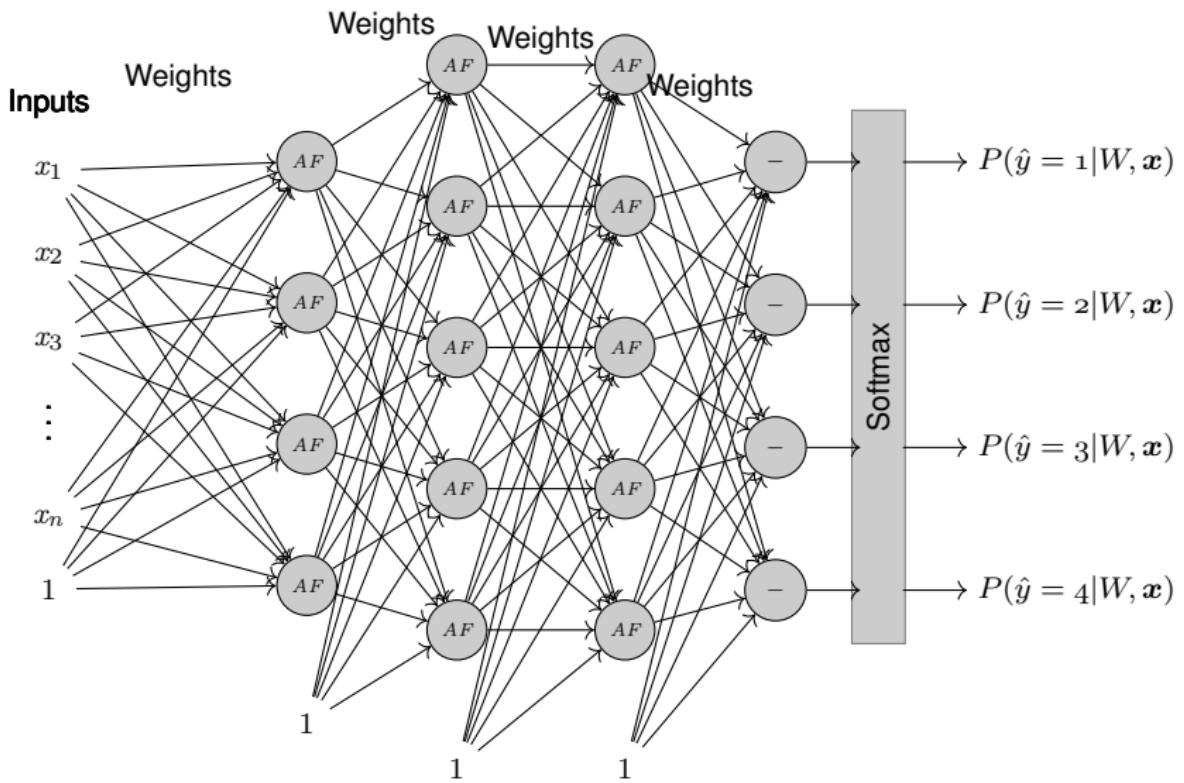
Fully connected neural networks



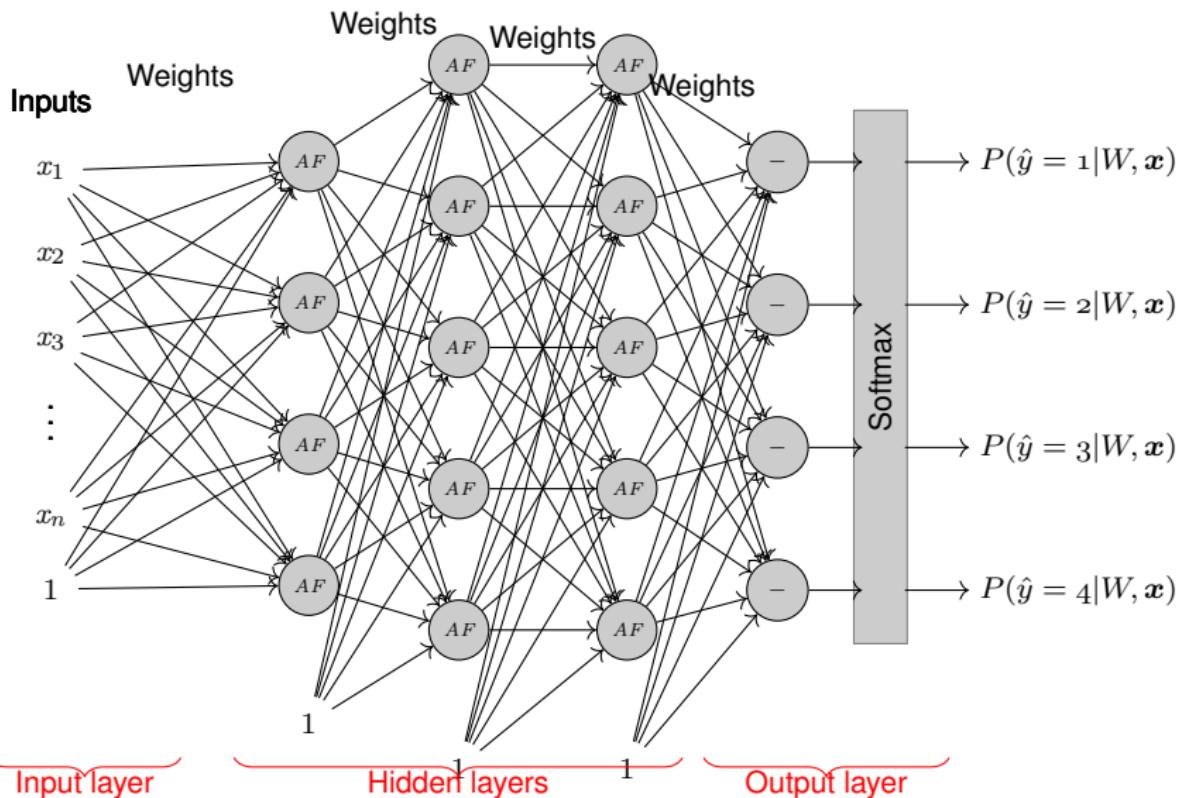
Fully connected neural networks



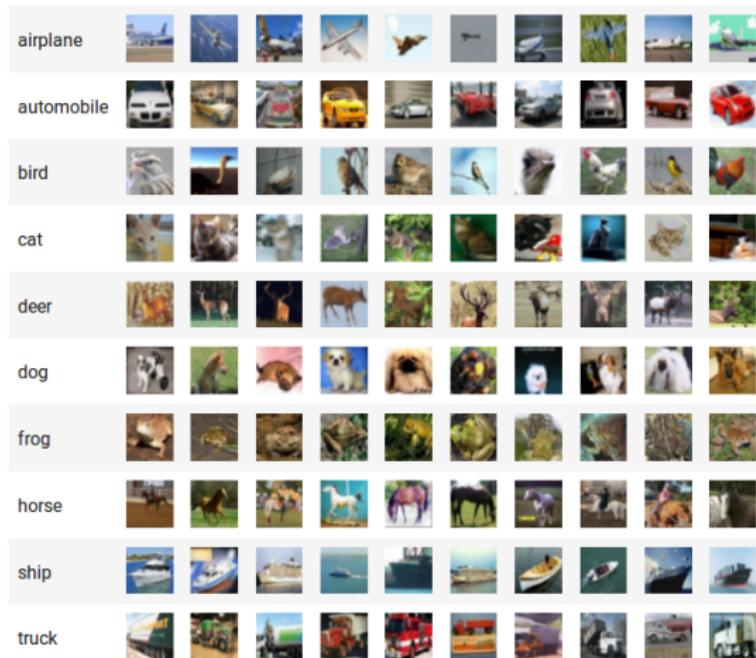
Fully connected neural networks



Fully connected neural networks

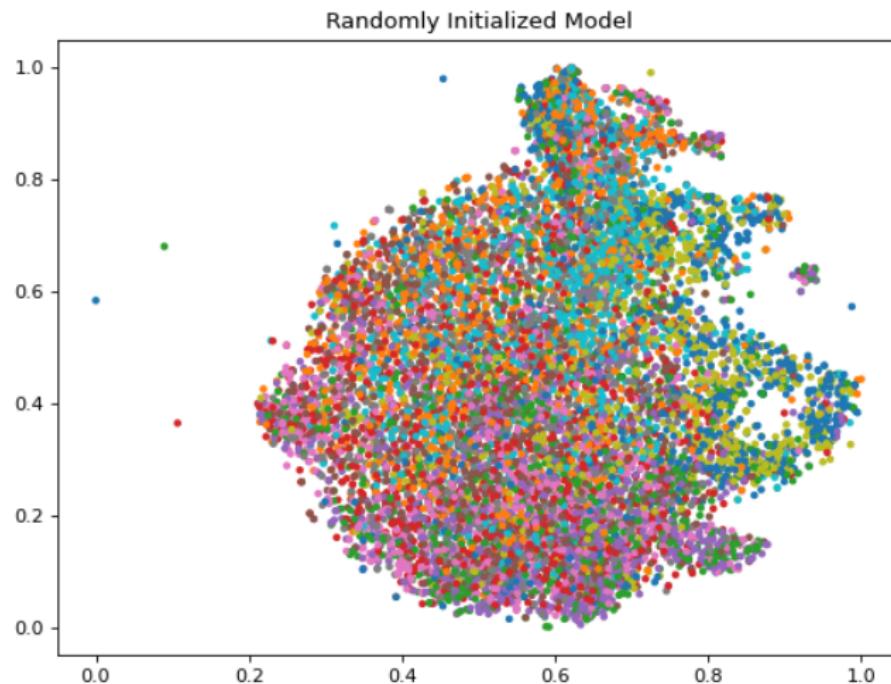


CIFAR10

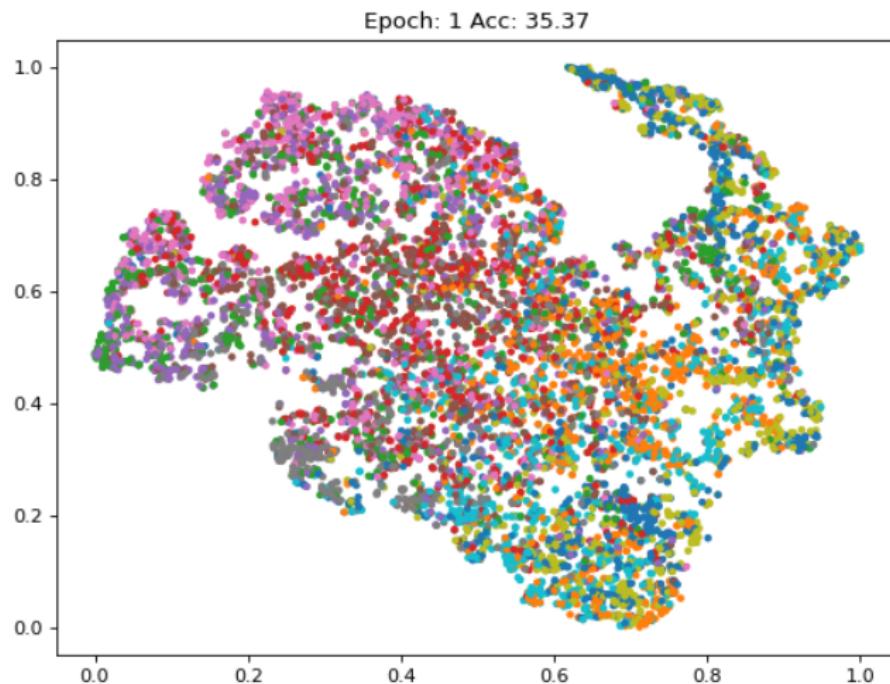


Example images from the **CIFAR-10** dataset.

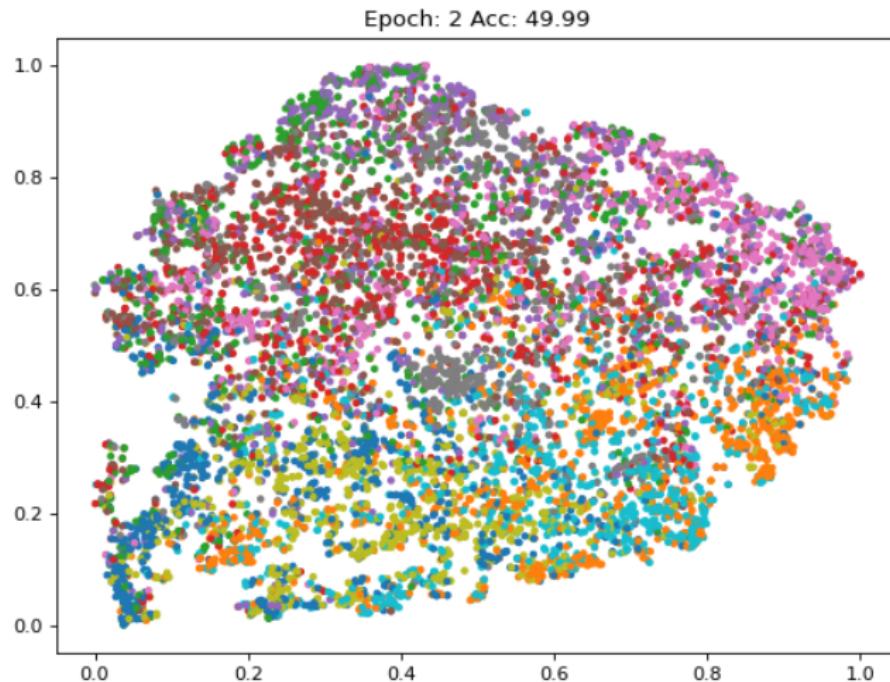
Visualization of the last hidden layer output for CIFAR10 samples



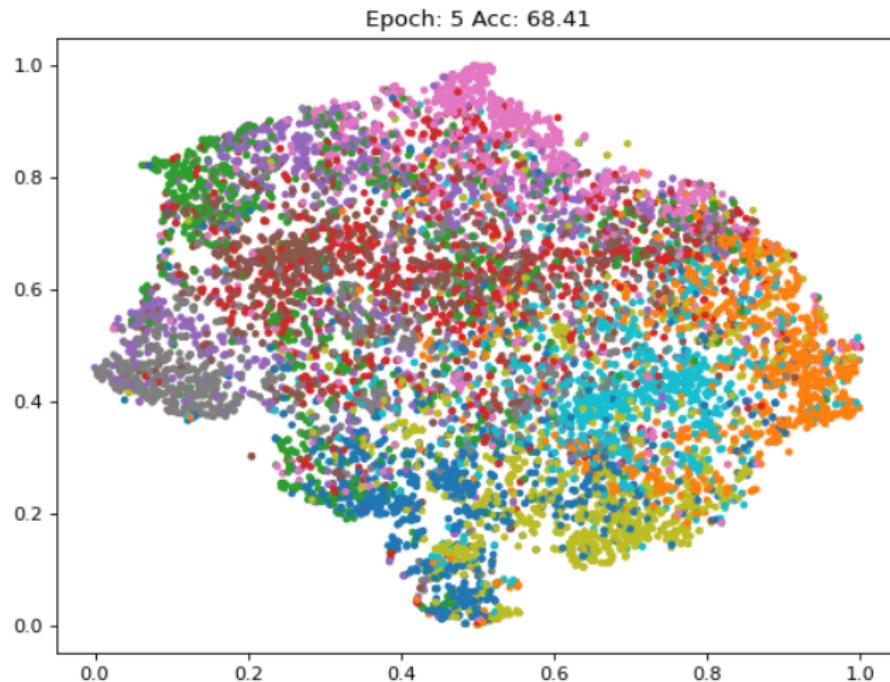
Visualization of the last hidden layer output for CIFAR10 samples



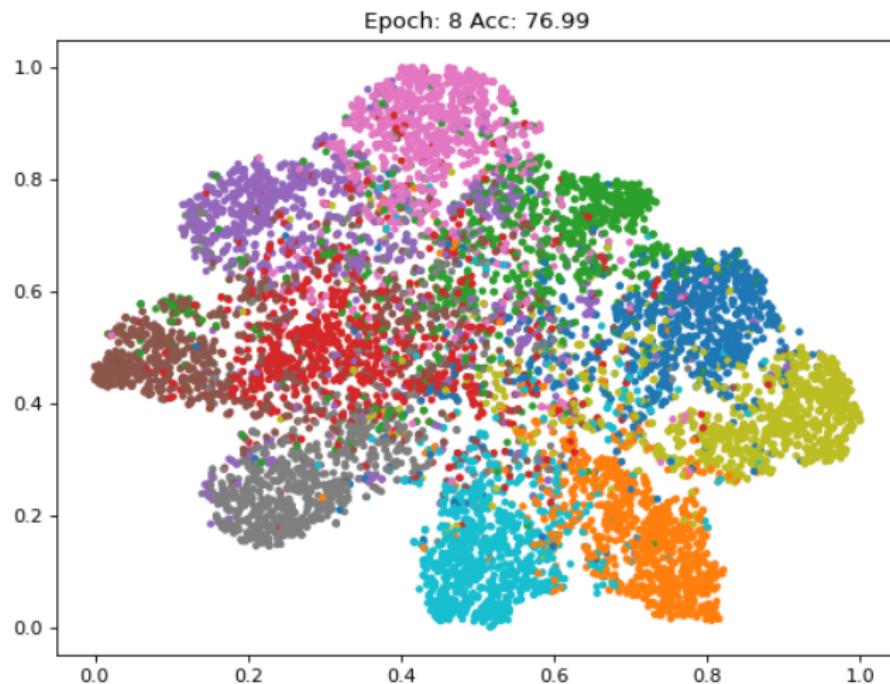
Visualization of the last hidden layer output for CIFAR10 samples



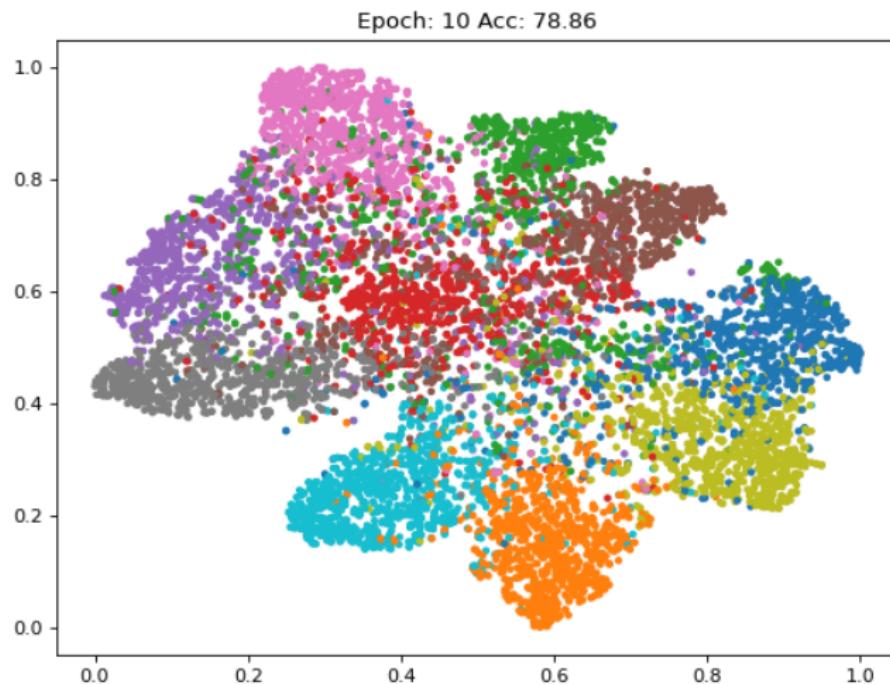
Visualization of the last hidden layer output for CIFAR10 samples



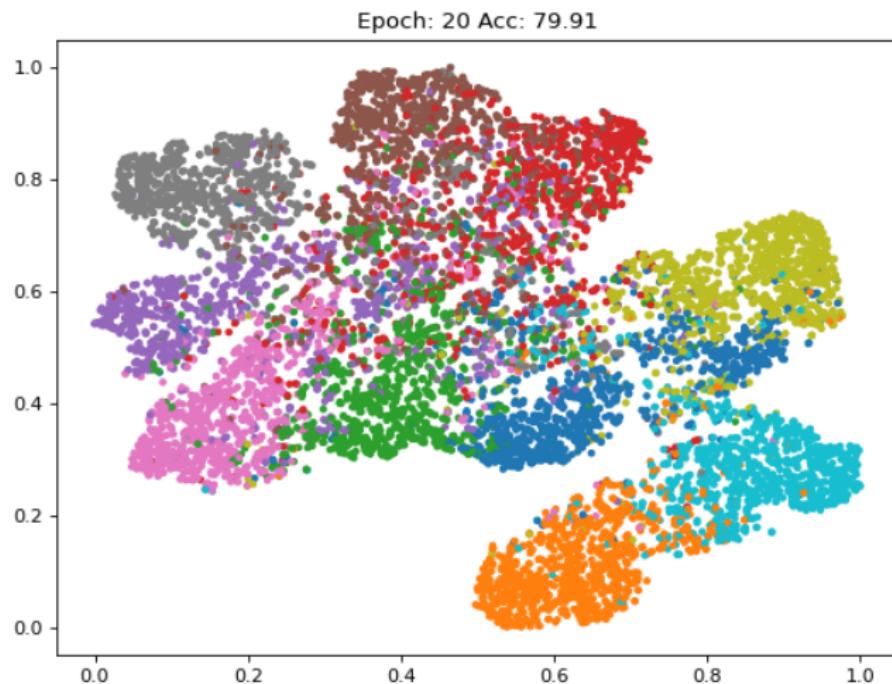
Visualization of the last hidden layer output for CIFAR10 samples



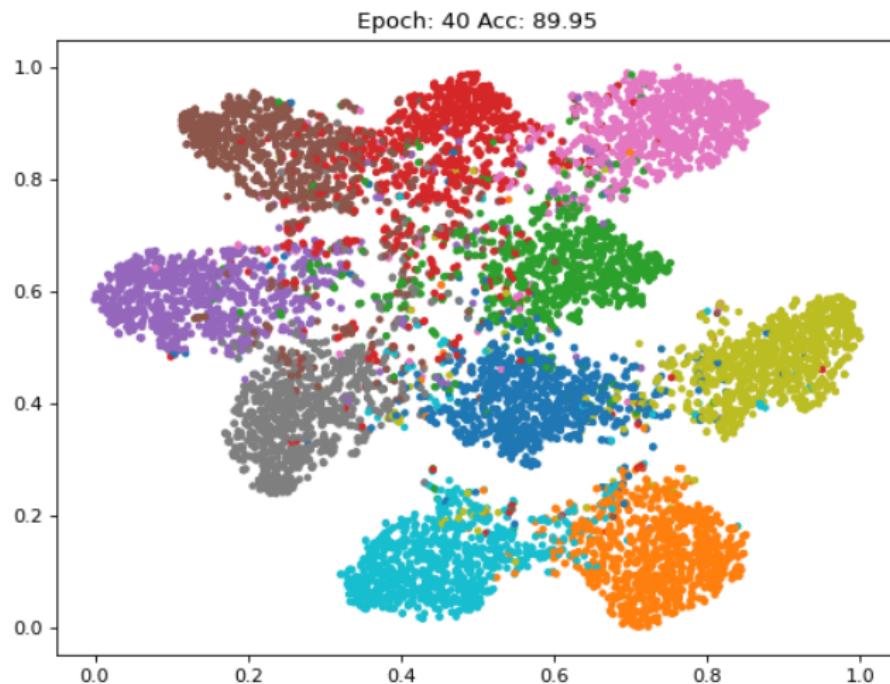
Visualization of the last hidden layer output for CIFAR10 samples



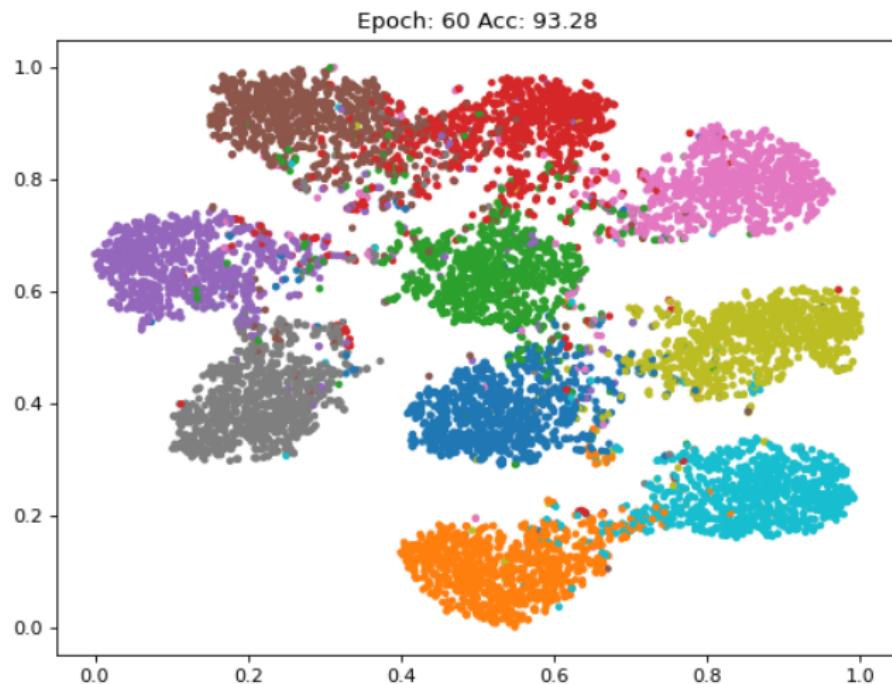
Visualization of the last hidden layer output for CIFAR10 samples



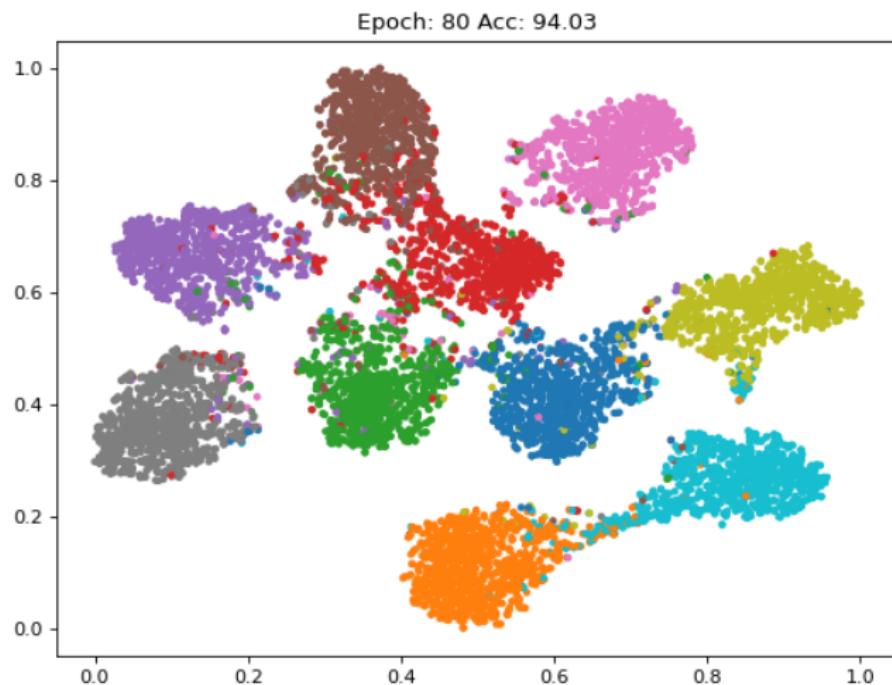
Visualization of the last hidden layer output for CIFAR10 samples



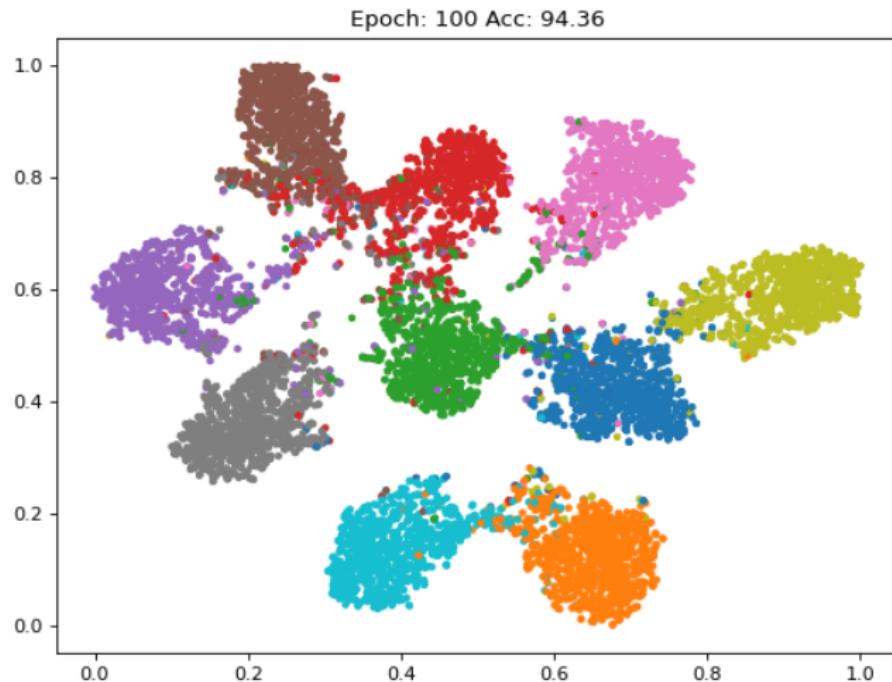
Visualization of the last hidden layer output for CIFAR10 samples



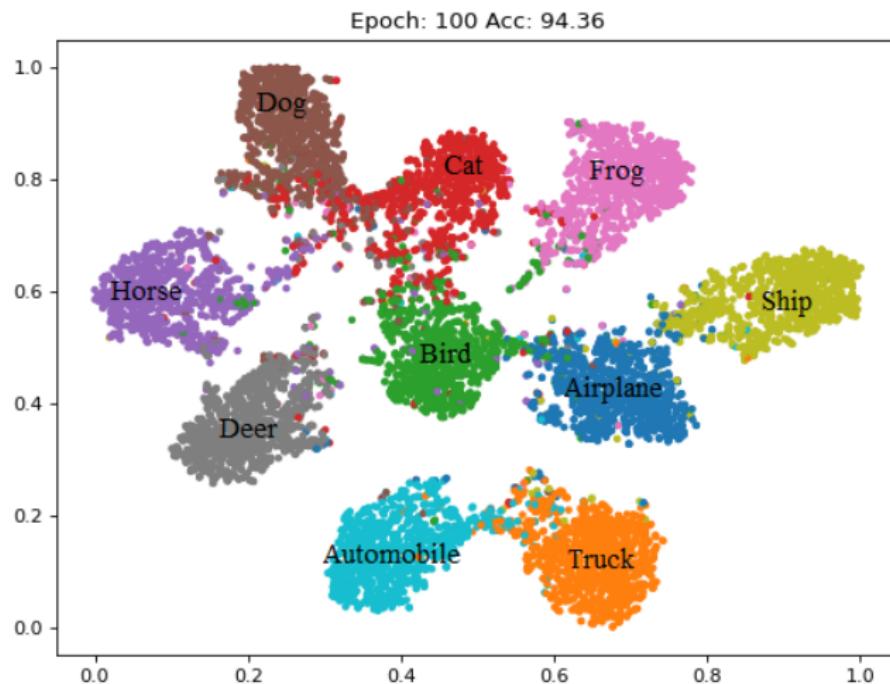
Visualization of the last hidden layer output for CIFAR10 samples



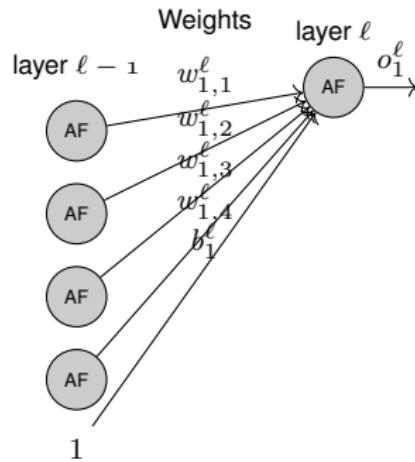
Visualization of the last hidden layer output for CIFAR10 samples



Visualization of the last hidden layer output for CIFAR10 samples



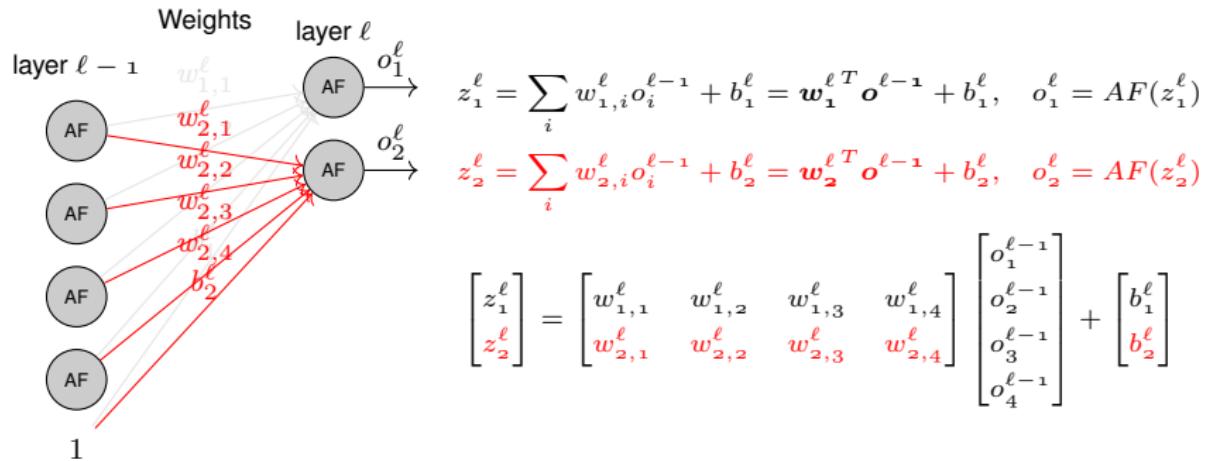
Fully connected neural networks



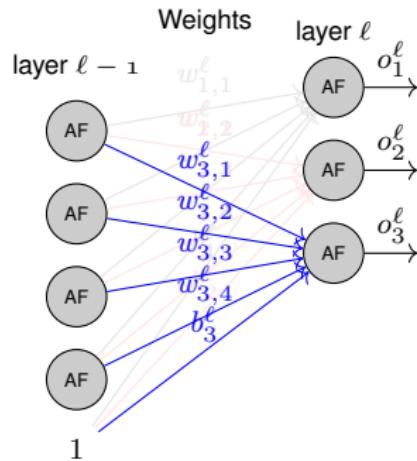
$$z_1^\ell = \sum_i w_{1,i}^\ell o_i^{\ell-1} + b_1^\ell = \mathbf{w}_1^\ell {}^T \mathbf{o}^{\ell-1} + b_1^\ell, \quad o_1^\ell = AF(z_1^\ell)$$

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

Fully connected neural networks



Fully connected neural networks



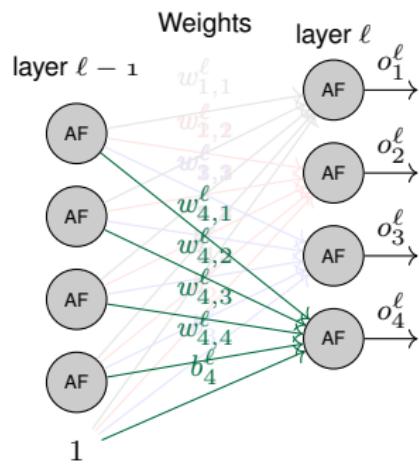
$$z_1^\ell = \sum_i w_{1,i}^\ell o_i^{\ell-1} + b_1^\ell = \mathbf{w}_1^\ell T \mathbf{o}^{\ell-1} + b_1^\ell, \quad o_1^\ell = AF(z_1^\ell)$$

$$z_2^\ell = \sum_i w_{2,i}^\ell o_i^{\ell-1} + b_2^\ell = \mathbf{w}_2^\ell T \mathbf{o}^{\ell-1} + b_2^\ell, \quad o_2^\ell = AF(z_2^\ell)$$

$$z_3^\ell = \sum_i w_{3,i}^\ell o_i^{\ell-1} + b_3^\ell = \mathbf{w}_3^\ell T \mathbf{o}^{\ell-1} + b_3^\ell, \quad o_3^\ell = AF(z_3^\ell)$$

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

Fully connected neural networks



$$z_1^\ell = \sum_i w_{1,i}^\ell o_i^{\ell-1} + b_1^\ell = \mathbf{w}_1^\ell T \mathbf{o}^{\ell-1} + b_1^\ell, \quad o_1^\ell = AF(z_1^\ell)$$

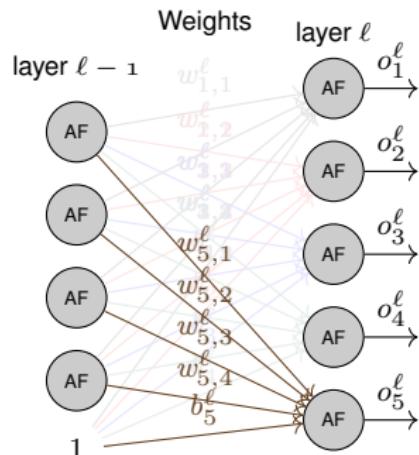
$$z_2^\ell = \sum_i w_{2,i}^\ell o_i^{\ell-1} + b_2^\ell = \mathbf{w}_2^\ell T \mathbf{o}^{\ell-1} + b_2^\ell, \quad o_2^\ell = AF(z_2^\ell)$$

$$z_3^\ell = \sum_i w_{3,i}^\ell o_i^{\ell-1} + b_3^\ell = \mathbf{w}_3^\ell T \mathbf{o}^{\ell-1} + b_3^\ell, \quad o_3^\ell = AF(z_3^\ell)$$

$$z_4^\ell = \sum_i w_{4,i}^\ell o_i^{\ell-1} + b_4^\ell = \mathbf{w}_4^\ell T \mathbf{o}^{\ell-1} + b_4^\ell, \quad o_4^\ell = AF(z_4^\ell)$$

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

Fully connected neural networks



$$z_1^\ell = \sum_i w_{1,i}^\ell o_i^{\ell-1} + b_1^\ell = \mathbf{w}_1^\ell T \mathbf{o}^{\ell-1} + b_1^\ell, \quad o_1^\ell = AF(z_1^\ell)$$

$$z_2^\ell = \sum_i w_{2,i}^\ell o_i^{\ell-1} + b_2^\ell = \mathbf{w}_2^\ell T \mathbf{o}^{\ell-1} + b_2^\ell, \quad o_2^\ell = AF(z_2^\ell)$$

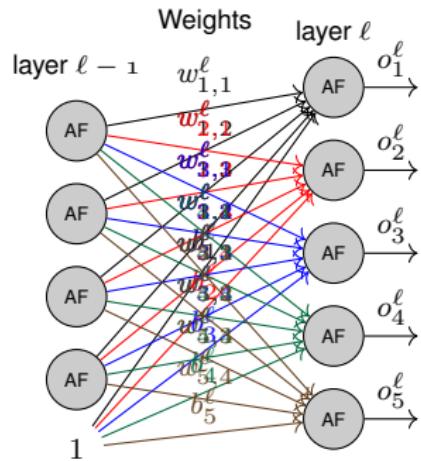
$$z_3^\ell = \sum_i w_{3,i}^\ell o_i^{\ell-1} + b_3^\ell = \mathbf{w}_3^\ell T \mathbf{o}^{\ell-1} + b_3^\ell, \quad o_3^\ell = AF(z_3^\ell)$$

$$z_4^\ell = \sum_i w_{4,i}^\ell o_i^{\ell-1} + b_4^\ell = \mathbf{w}_4^\ell T \mathbf{o}^{\ell-1} + b_4^\ell, \quad o_4^\ell = AF(z_4^\ell)$$

$$z_5^\ell = \sum_i w_{5,i}^\ell o_i^{\ell-1} + b_5^\ell = \mathbf{w}_5^\ell T \mathbf{o}^{\ell-1} + b_5^\ell, \quad o_5^\ell = AF(z_5^\ell)$$

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

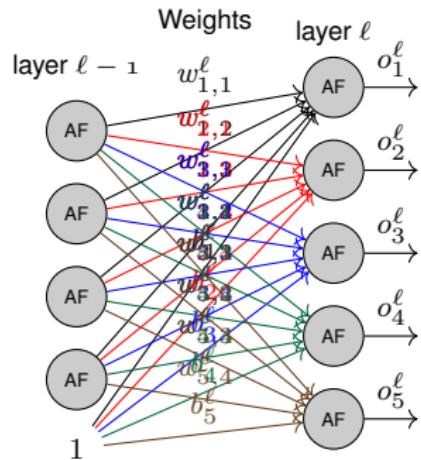
Vector activation



$$\begin{bmatrix} z_1^{\ell} \\ z_2^{\ell} \\ z_3^{\ell} \\ z_4^{\ell} \\ z_5^{\ell} \end{bmatrix} = \begin{bmatrix} w_{1,1}^{\ell} & w_{1,2}^{\ell} & w_{1,3}^{\ell} & w_{1,4}^{\ell} \\ w_{2,1}^{\ell} & w_{2,2}^{\ell} & w_{2,3}^{\ell} & w_{2,4}^{\ell} \\ w_{3,1}^{\ell} & w_{3,2}^{\ell} & w_{3,3}^{\ell} & w_{3,4}^{\ell} \\ w_{4,1}^{\ell} & w_{4,2}^{\ell} & w_{4,3}^{\ell} & w_{4,4}^{\ell} \\ w_{5,1}^{\ell} & w_{5,2}^{\ell} & w_{5,3}^{\ell} & w_{5,4}^{\ell} \end{bmatrix} \begin{bmatrix} o_1^{\ell-1} \\ o_2^{\ell-1} \\ o_3^{\ell-1} \\ o_4^{\ell-1} \\ o_5^{\ell-1} \end{bmatrix} + \begin{bmatrix} b_1^{\ell} \\ b_2^{\ell} \\ b_3^{\ell} \\ b_4^{\ell} \\ b_5^{\ell} \end{bmatrix}$$

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

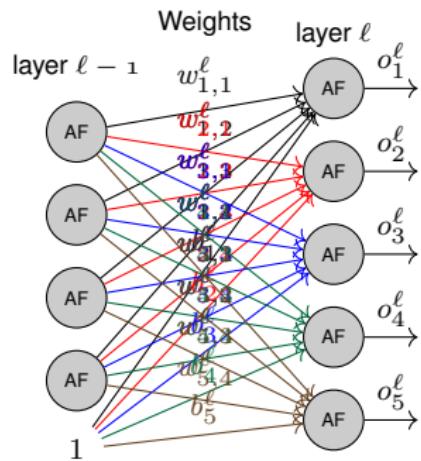
Vector activation



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

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

Vector activation

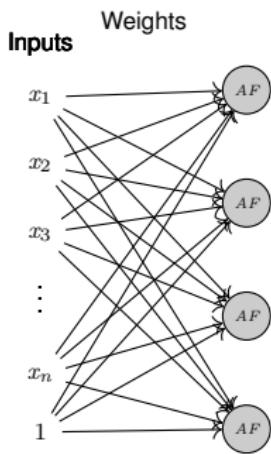


$$\begin{bmatrix} z_1^\ell \\ z_2^\ell \\ z_3^\ell \\ z_4^\ell \\ z_5^\ell \end{bmatrix} = \begin{bmatrix} w_{1,1}^\ell & w_{1,2}^\ell & w_{1,3}^\ell & w_{1,4}^\ell \\ w_{2,1}^\ell & w_{2,2}^\ell & w_{2,3}^\ell & w_{2,4}^\ell \\ w_{3,1}^\ell & w_{3,2}^\ell & w_{3,3}^\ell & w_{3,4}^\ell \\ w_{4,1}^\ell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

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

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

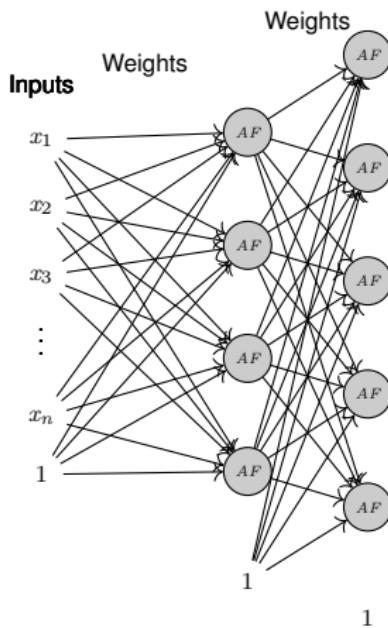
Fully connected neural networks



1

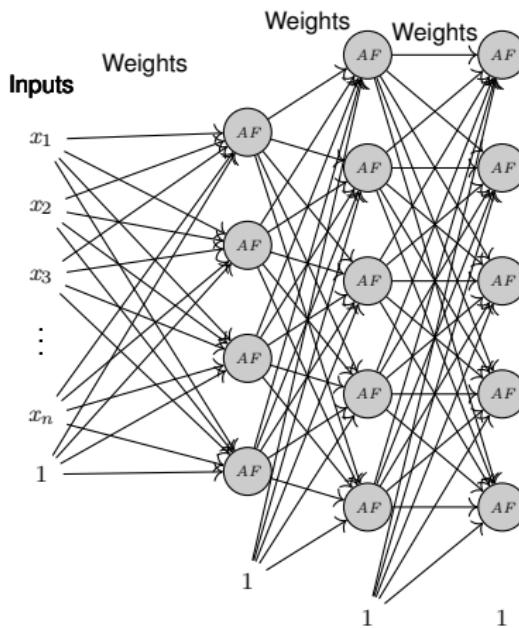
$$\mathbf{o}^1 = AF(W^1 \mathbf{x} + \mathbf{b}^1), \quad W^1 \in \mathbb{R}^{4 \times n}, \quad \mathbf{x} \in \mathbb{R}^n, \quad \mathbf{o}^1 \in \mathbb{R}^4, \quad \mathbf{b} \in \mathbb{R}^4.$$

Fully connected neural networks



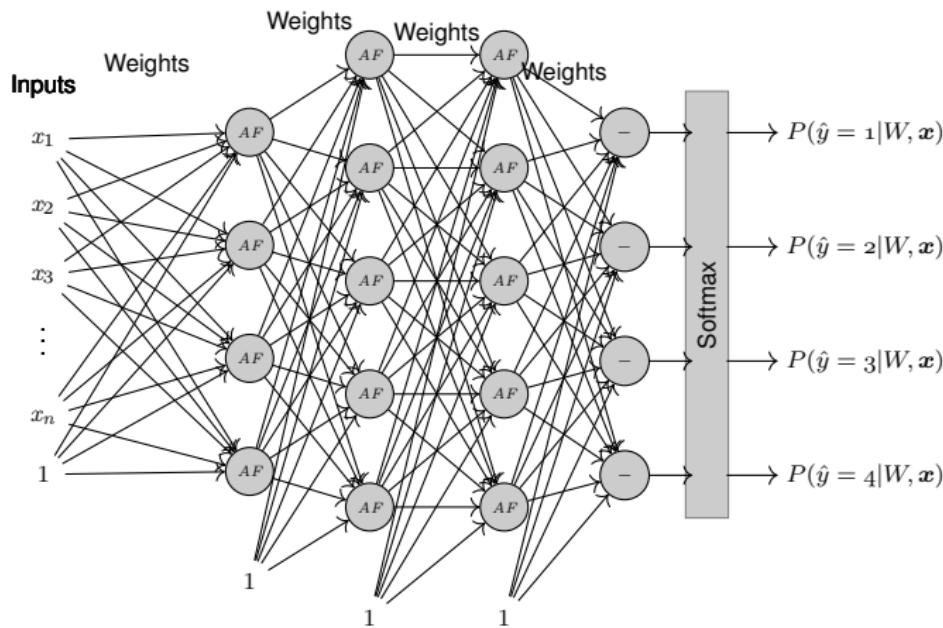
$$\mathbf{o}^2 = AF(W^2\mathbf{o}^1 + \mathbf{b}^2), \quad W^2 \in \mathbb{R}^{5 \times 4}, \quad \mathbf{o}^1 \in \mathbb{R}^4, \quad \mathbf{o}^2 \in \mathbb{R}^5, \quad \mathbf{b}^2 \in \mathbb{R}^5.$$

Fully connected neural networks



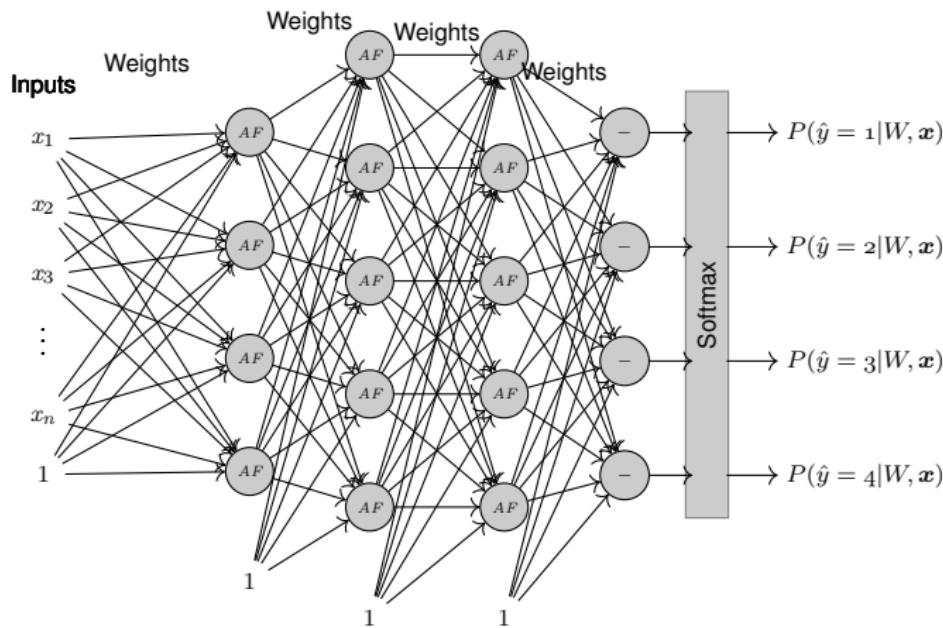
$$\mathbf{o}^3 = AF(W^3\mathbf{o}^2 + \mathbf{b}^3), \quad W^3 \in \mathbb{R}^{5 \times 5}, \quad \mathbf{o}^2 \in \mathbb{R}^5, \quad \mathbf{o}^3 \in \mathbb{R}^5, \quad \mathbf{b}^3 \in \mathbb{R}^5.$$

Fully connected neural networks



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

Fully connected neural networks



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

Batch of inputs

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

$$\begin{aligned}
 & \begin{bmatrix} z_1^{1,(1)} & z_1^{1,(2)} & z_1^{1,(3)} \\ z_2^{1,(1)} & z_2^{1,(2)} & z_2^{1,(3)} \\ z_3^{1,(1)} & z_3^{1,(2)} & z_3^{1,(3)} \\ z_4^{1,(1)} & z_4^{1,(2)} & z_4^{1,(3)} \end{bmatrix} = \begin{bmatrix} w_{1,1}^1 & w_{1,2}^1 & w_{1,3}^1 & \dots & w_{1,n}^1 \\ w_{2,1}^1 & w_{2,2}^1 & w_{2,3}^1 & \dots & w_{2,n}^1 \\ w_{3,1}^1 & w_{3,2}^1 & w_{3,3}^1 & \dots & w_{3,n}^1 \\ w_{4,1}^1 & w_{4,2}^1 & w_{4,3}^1 & \dots & w_{4,n}^1 \end{bmatrix} \begin{bmatrix} x_1^{(1)} & x_1^{(2)} & x_1^{(3)} \\ x_2^{(1)} & x_2^{(2)} & x_2^{(3)} \\ x_3^{(1)} & x_3^{(2)} & x_3^{(3)} \\ \vdots & \vdots & \vdots \\ x_n^{(1)} & x_n^{(2)} & x_n^{(3)} \end{bmatrix} \\
 & + \begin{bmatrix} b_1^1 & b_2^1 & b_3^1 \\ b_2^1 & b_3^1 & b_4^1 \\ b_3^1 & b_4^1 & b_1^1 \\ b_4^1 & b_1^1 & b_2^1 \end{bmatrix} \\
 & \begin{bmatrix} o_1^{1,(1)} & o_1^{1,(2)} & o_1^{1,(3)} \\ o_2^{1,(1)} & o_2^{1,(2)} & o_2^{1,(3)} \\ o_3^{1,(1)} & o_3^{1,(2)} & o_3^{1,(3)} \\ o_4^{1,(1)} & o_4^{1,(2)} & o_4^{1,(3)} \end{bmatrix} = AF(\begin{bmatrix} z_1^{1,(1)} & z_1^{1,(2)} & z_1^{1,(3)} \\ z_2^{1,(1)} & z_2^{1,(2)} & z_2^{1,(3)} \\ z_3^{1,(1)} & z_3^{1,(2)} & z_3^{1,(3)} \\ z_4^{1,(1)} & z_4^{1,(2)} & z_4^{1,(3)} \end{bmatrix})
 \end{aligned}$$

Batch of inputs

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

$$\begin{aligned}
 & \begin{bmatrix} z_1^{1,(1)} & z_1^{1,(2)} & z_1^{1,(3)} \\ z_2^{1,(1)} & z_2^{1,(2)} & z_2^{1,(3)} \\ z_3^{1,(1)} & z_3^{1,(2)} & z_3^{1,(3)} \\ z_4^{1,(1)} & z_4^{1,(2)} & z_4^{1,(3)} \end{bmatrix} = \begin{bmatrix} w_{1,1}^1 & w_{1,2}^1 & w_{1,3}^1 & \dots & w_{1,n}^1 \\ w_{2,1}^1 & w_{2,2}^1 & w_{2,3}^1 & \dots & w_{2,n}^1 \\ w_{3,1}^1 & w_{3,2}^1 & w_{3,3}^1 & \dots & w_{3,n}^1 \\ w_{4,1}^1 & w_{4,2}^1 & w_{4,3}^1 & \dots & w_{4,n}^1 \end{bmatrix} \begin{bmatrix} x_1^{(1)} & x_1^{(2)} & x_1^{(3)} \\ x_2^{(1)} & x_2^{(2)} & x_2^{(3)} \\ x_3^{(1)} & x_3^{(2)} & x_3^{(3)} \\ \vdots & \vdots & \vdots \\ x_n^{(1)} & x_n^{(2)} & x_n^{(3)} \end{bmatrix} \\
 & + \begin{bmatrix} b_1^1 & b_2^1 & b_3^1 \\ b_2^1 & b_2^1 & b_2^1 \\ b_3^1 & b_3^1 & b_3^1 \\ b_4^1 & b_4^1 & b_4^1 \end{bmatrix} \\
 & \begin{bmatrix} \begin{bmatrix} o_1^{1,(1)} \\ o_2^{1,(1)} \\ o_3^{1,(1)} \\ o_4^{1,(1)} \end{bmatrix} & \begin{bmatrix} o_1^{1,(2)} \\ o_2^{1,(2)} \\ o_3^{1,(2)} \\ o_4^{1,(2)} \end{bmatrix} & \begin{bmatrix} o_1^{1,(3)} \\ o_2^{1,(3)} \\ o_3^{1,(3)} \\ o_4^{1,(3)} \end{bmatrix} \end{bmatrix} = AF(\begin{bmatrix} z_1^{1,(1)} & z_1^{1,(2)} & z_1^{1,(3)} \\ z_2^{1,(1)} & z_2^{1,(2)} & z_2^{1,(3)} \\ z_3^{1,(1)} & z_3^{1,(2)} & z_3^{1,(3)} \\ z_4^{1,(1)} & z_4^{1,(2)} & z_4^{1,(3)} \end{bmatrix})
 \end{aligned}$$

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

Activation functions

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

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

Activation functions

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

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

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

Activation functions

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

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

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

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

Activation functions

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

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

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

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

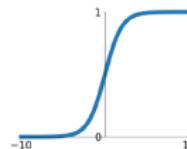
$$\hat{y} = \text{Softmax}(W' \mathbf{x} + b')$$

We end up with a softmax classifier!

Activation functions

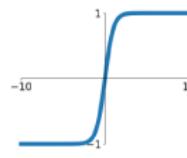
Sigmoid

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



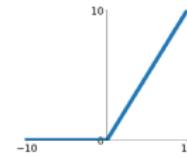
tanh

$$\tanh(x)$$



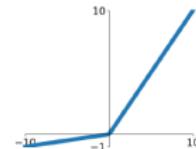
ReLU

$$\max(0, x)$$



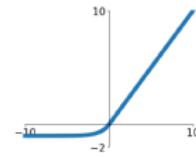
Leaky ReLU

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



ELU

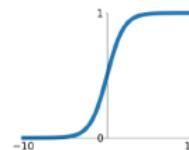
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Activation functions

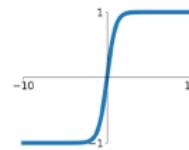
Sigmoid

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



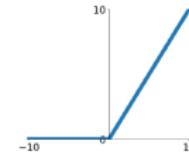
tanh

$$\tanh(x)$$



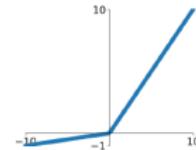
ReLU

$$\max(0, x)$$



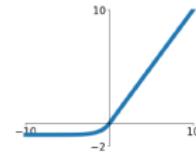
Leaky ReLU

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



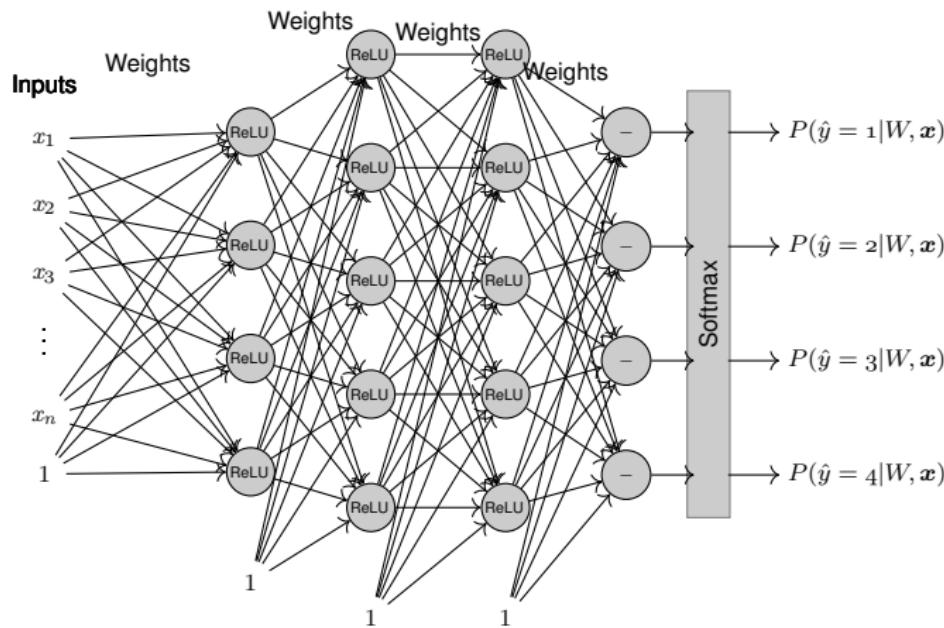
ELU

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



ReLU is a good default choice for most problems

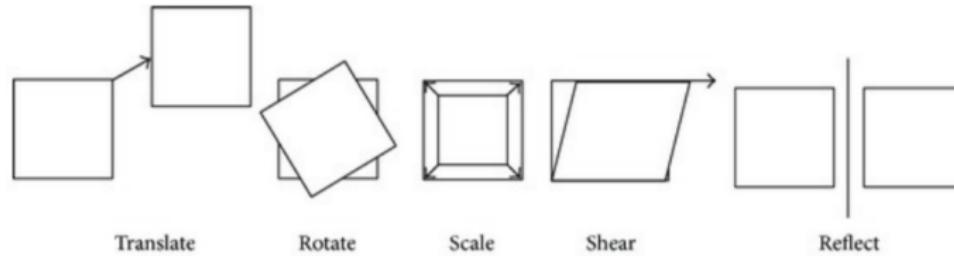
ReLU activation function



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

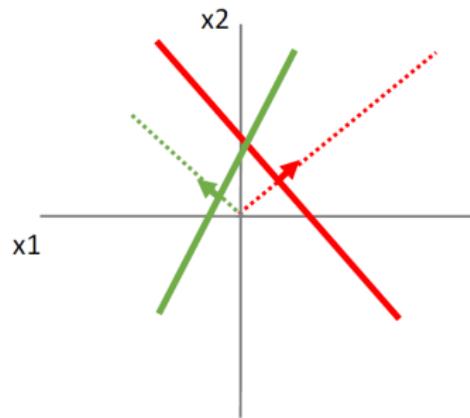
Affine transformation

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



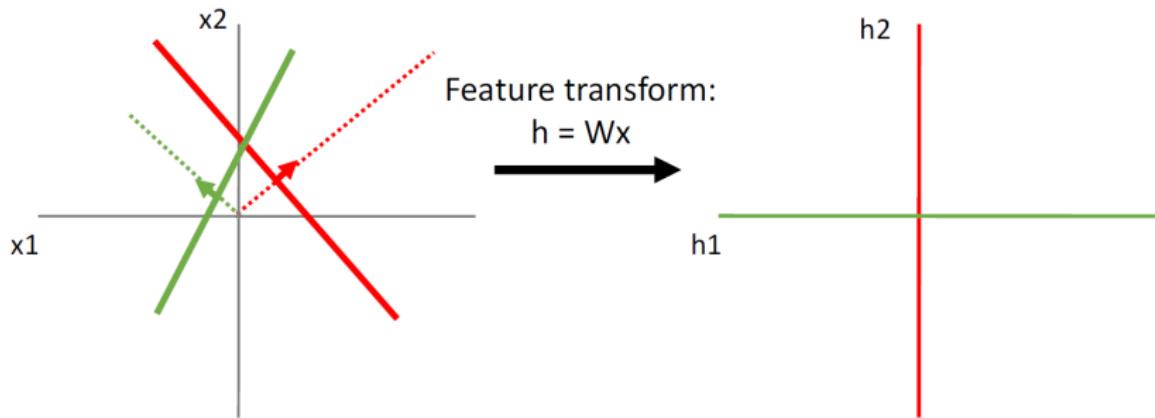
Space warping

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



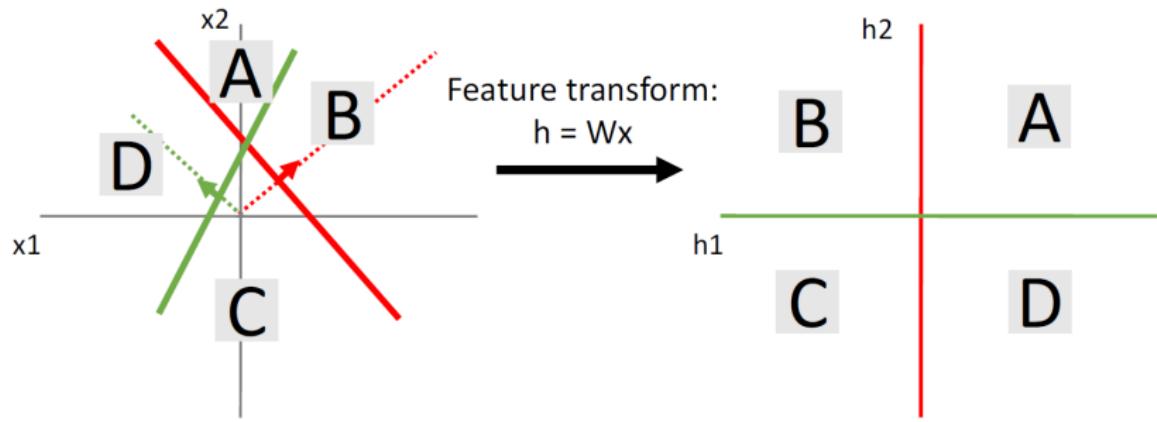
Space warping

- Consider a linear transform: $h = Wx$ Where x, h are both 2-dimensional



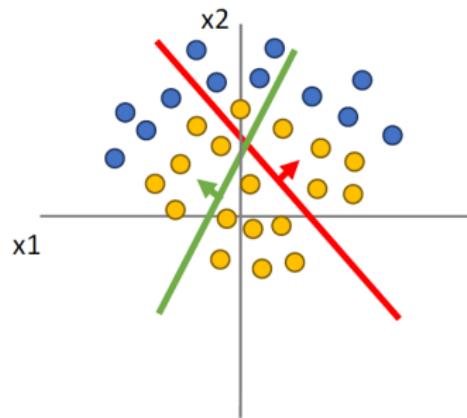
Space warping

- Consider a linear transform: $h = Wx$ Where x, h are both 2-dimensional



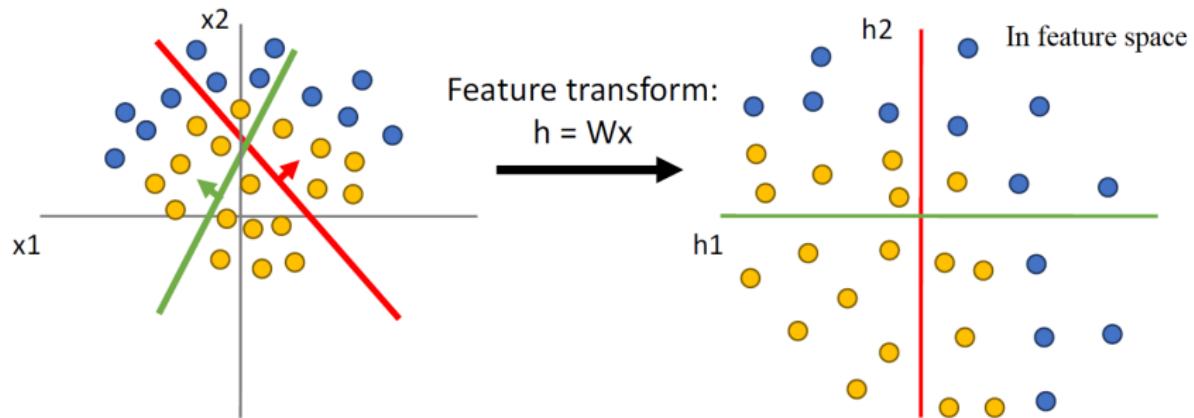
Space warping

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



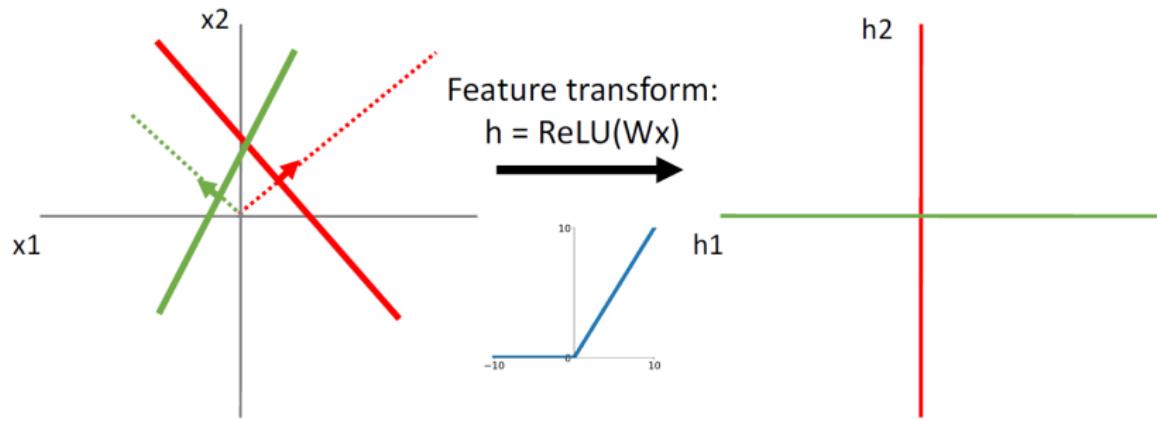
Space warping

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



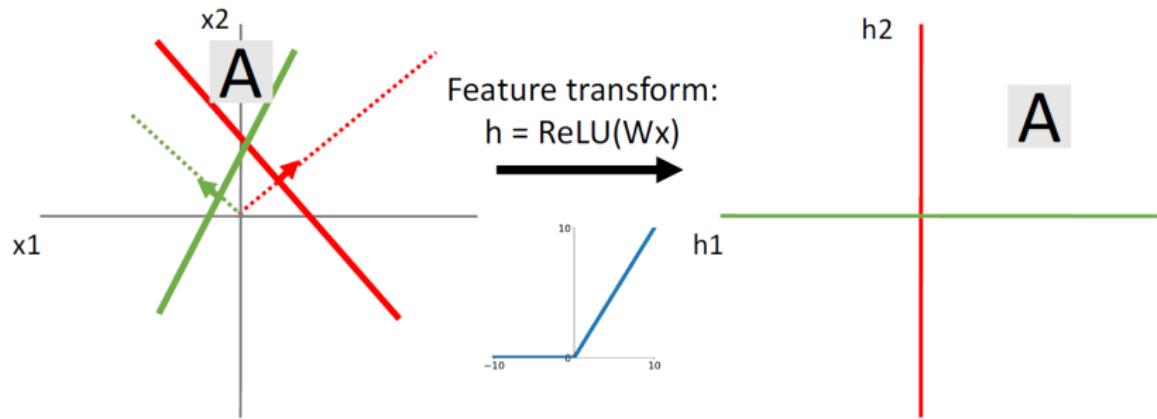
Space warping

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



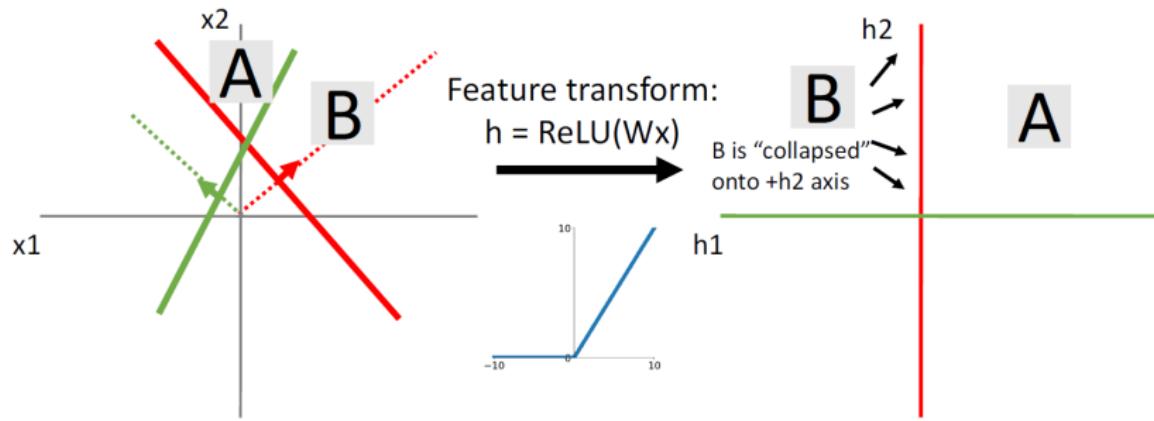
Space warping

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



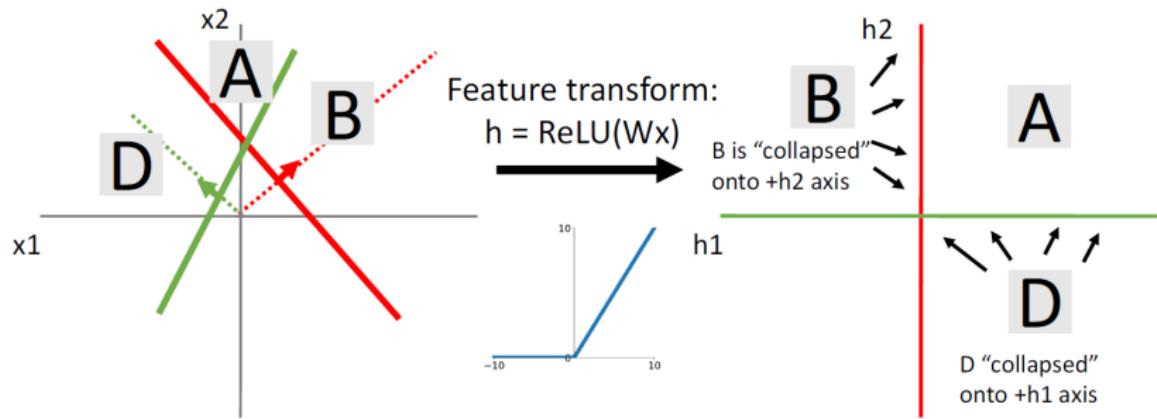
Space warping

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



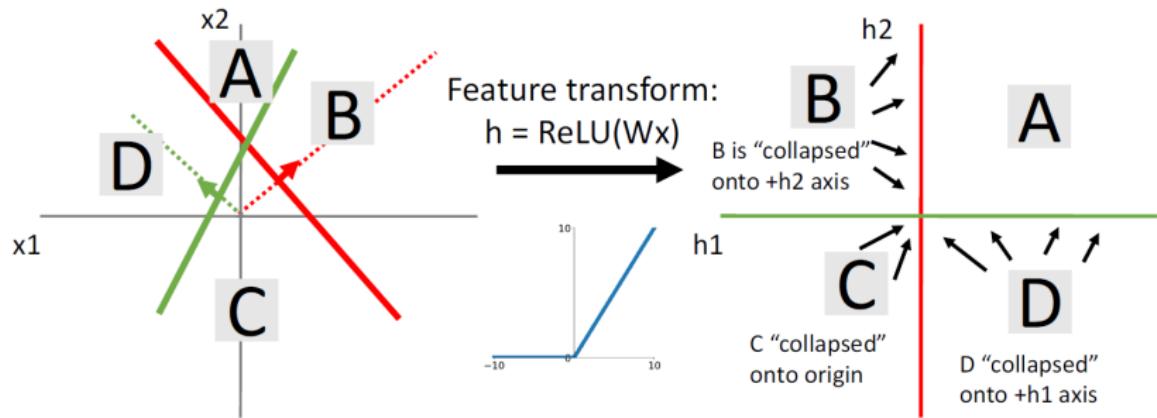
Space warping

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



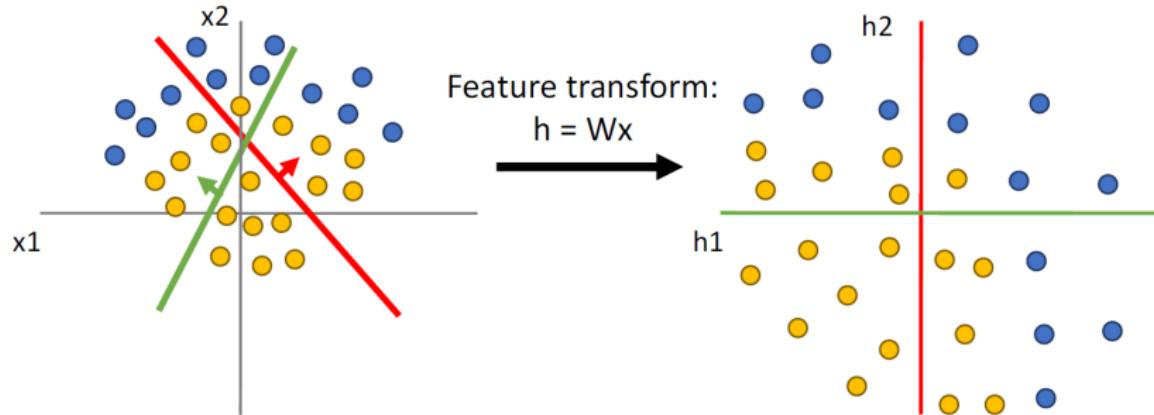
Space warping

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



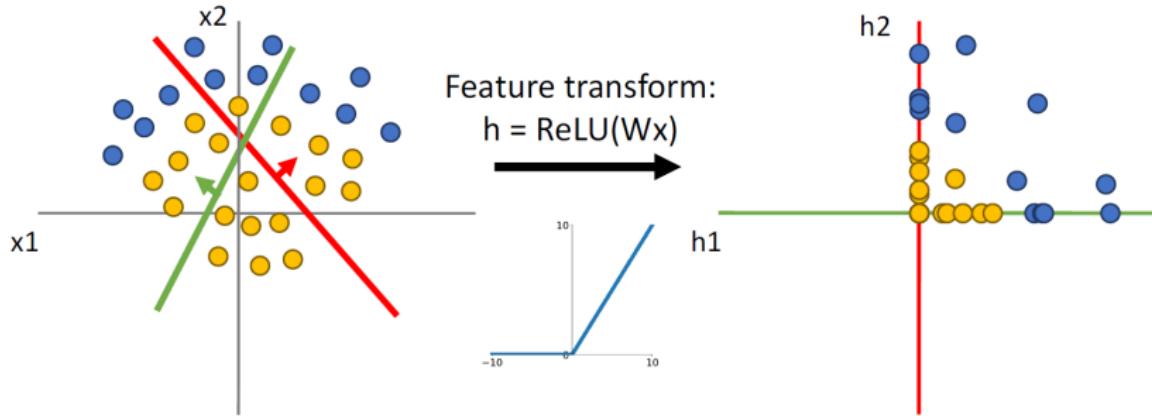
Space warping

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



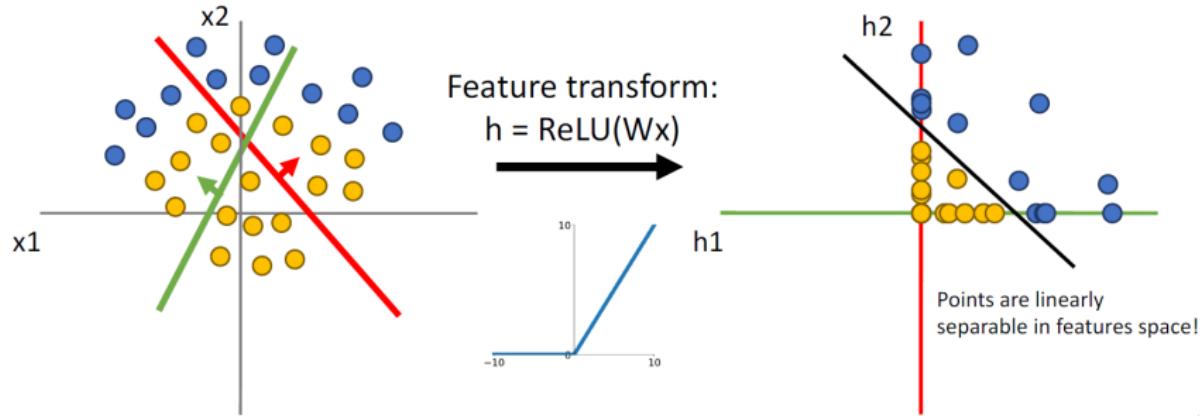
Space warping

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



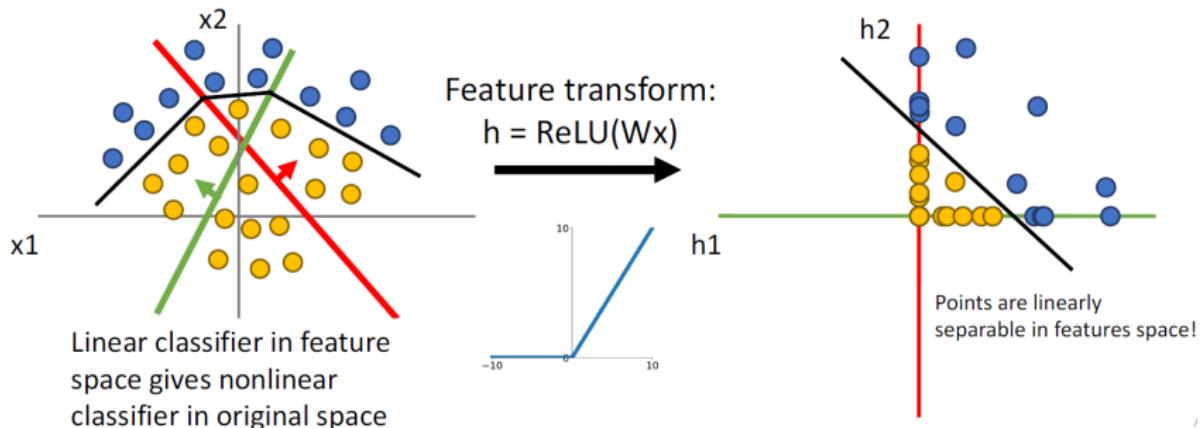
Space warping

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

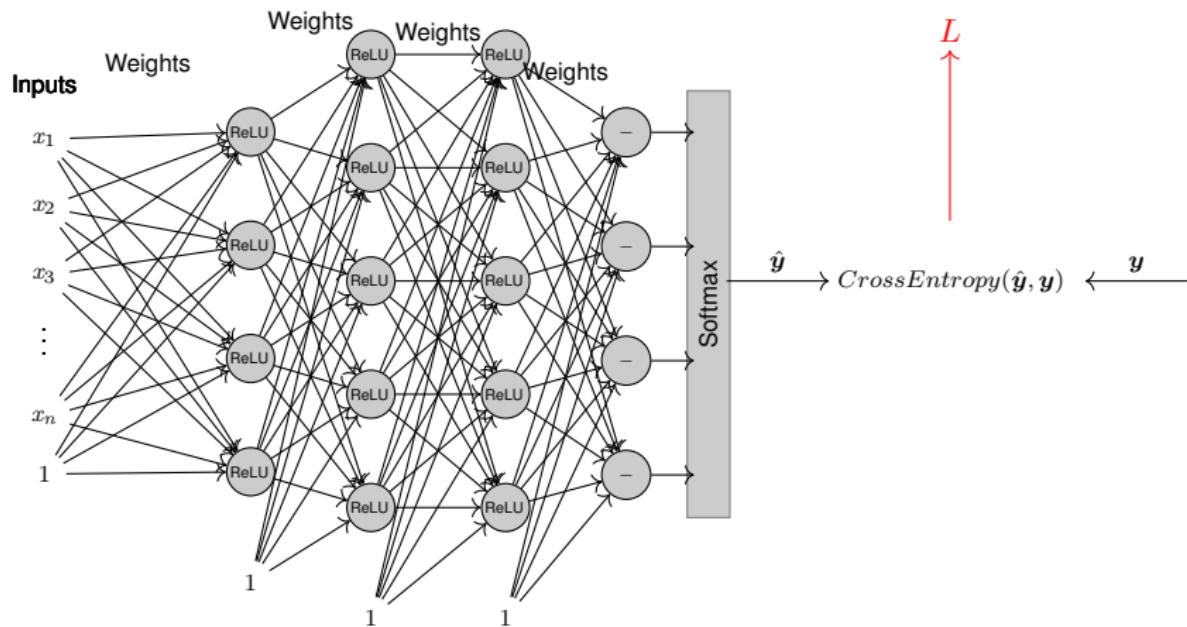


Space warping

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



Fully connected neural network loss



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

References

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