AI504: Programming for Artificial Intelligence

Week 3: Neural Nets & Backpropagation

Edward Choi
Grad School of AI
edwardchoi@kaist.ac.kr

Today’s Topic

- Deep Learning Frameworks
- Logistic Regression
- Neural Networks
- Backpropagation
- Autograd (in PyTorch)

Deep Learning Frameworks

Deep Learning Libraries

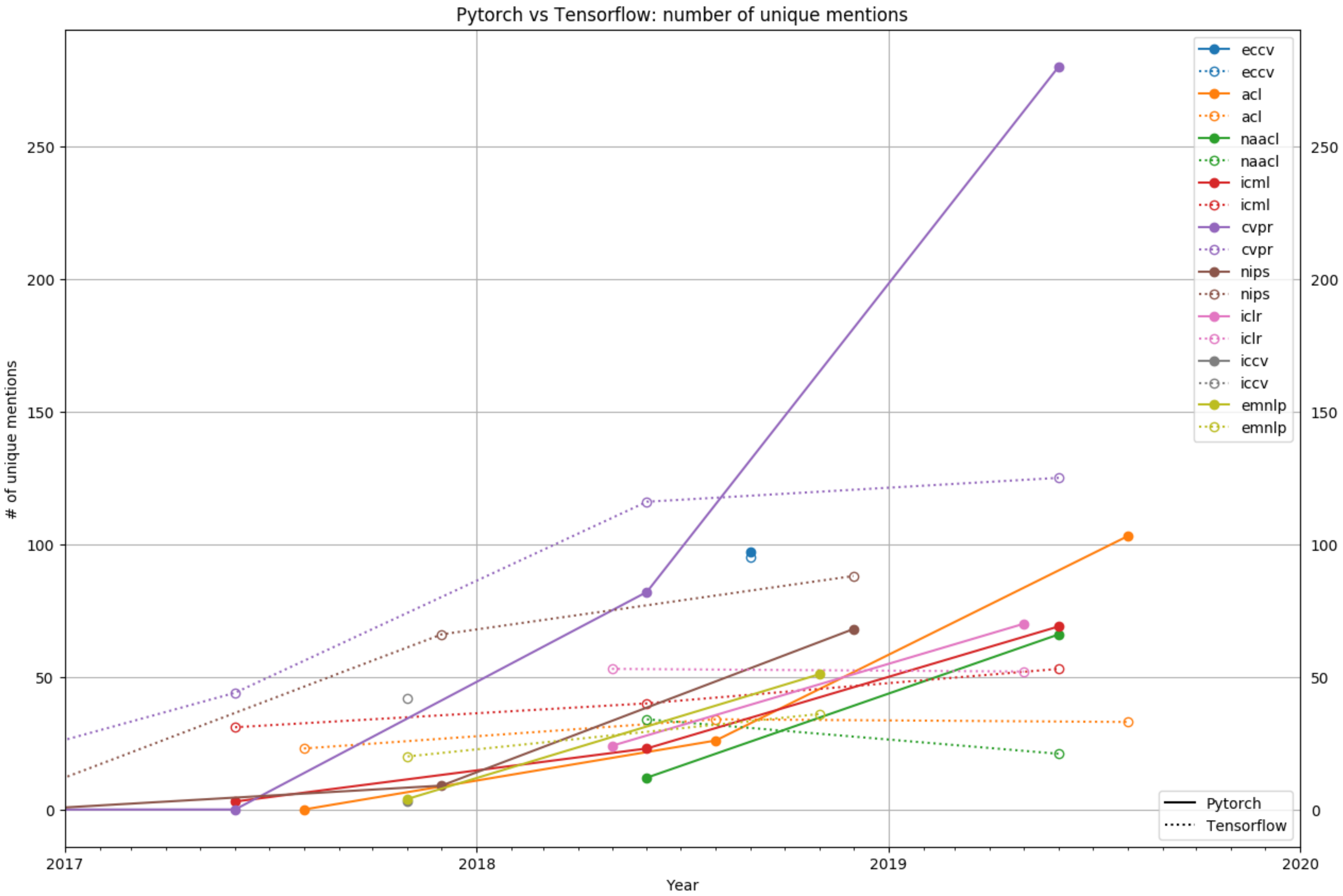
- Theano
  - Probably the first popular deep learning framework
  - Developed in MILA (Yoshua Bengio's group)
- Caffe
  - Deep learning framework in C++
  - Developed in UC Berkeley
- MXNet
  - Deep learning framework for speed and scalability
  - Supported by Amazon
- TensorFlow
- PyTorch

Deep Learning Libraries

- Theano
- Caffe
- MXNet
- TensorFlow 1.0
  - Compile the model then execute.
  - Big boilerplate.
  - Supposed to be fast.
  - Good for production.
- PyTorch (Came from Torch written in Lua)
  - Compile as you go.
  - Small boilerplate.
  - Supposed to be slow.
  - Good for research.

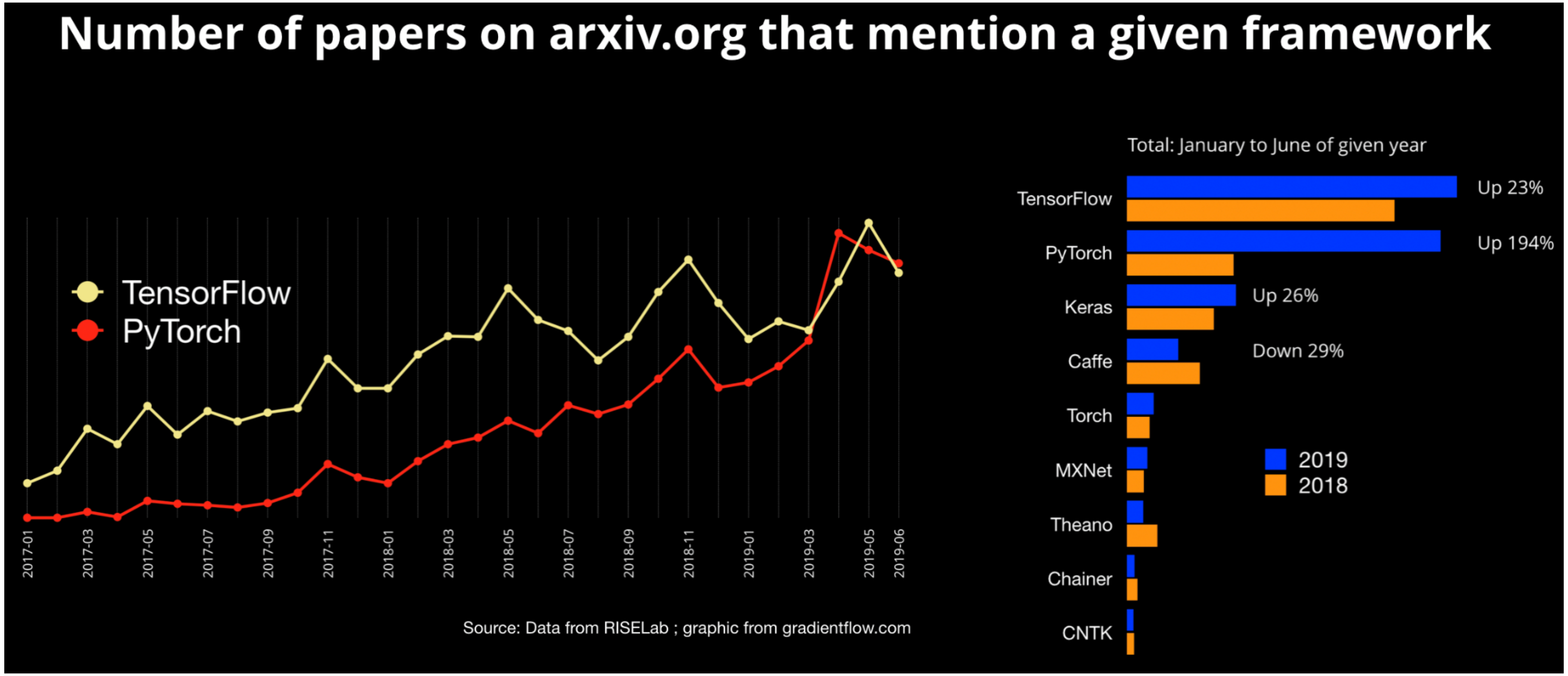
TensorFlow VS PyTorch

- PyTorch is catching up



TensorFlow VS PyTorch

- PyTorch is catching up



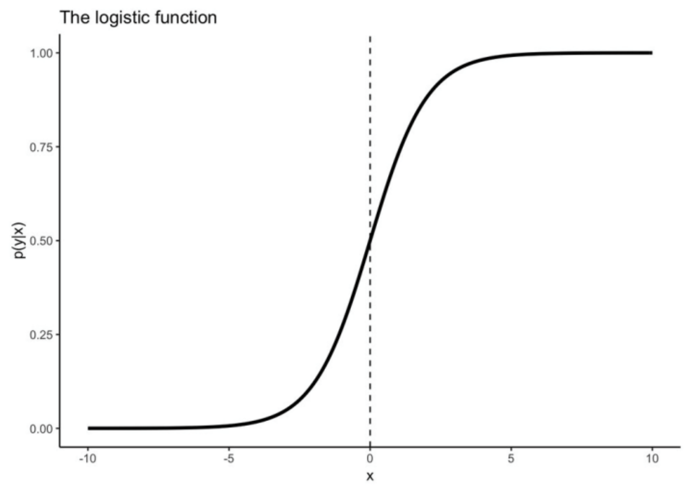
Alternatives

- TensorFlow 2.0
  - Trying to be similar to PyTorch.
  - Eager-mode (lazy compilation).
- JAX (& FLAX)
  - Support higher-order differentiation.
  - Good for calculating Hessian.

Logistic Regression

Logistic Regression

- Maybe the most popular model in statistical studies
  - A well-studied model. (Used since 19th century)
  - Analysis of coefficients of predictor variables (i.e. explanatory variable, independent variable, feature).



“Logistic Function”

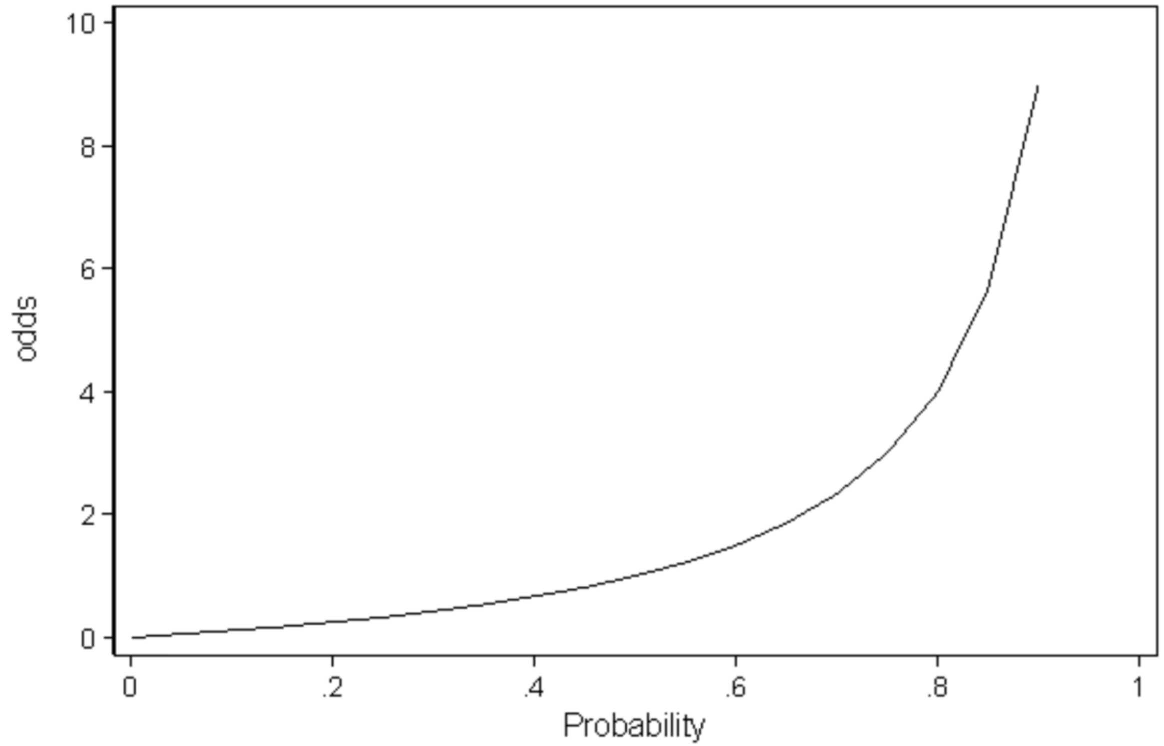
Multivariate Logistic Regression

$$p = \frac{1}{1 + b^{-(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_m x_m)}}$$

where usually $b = e$.

Odds

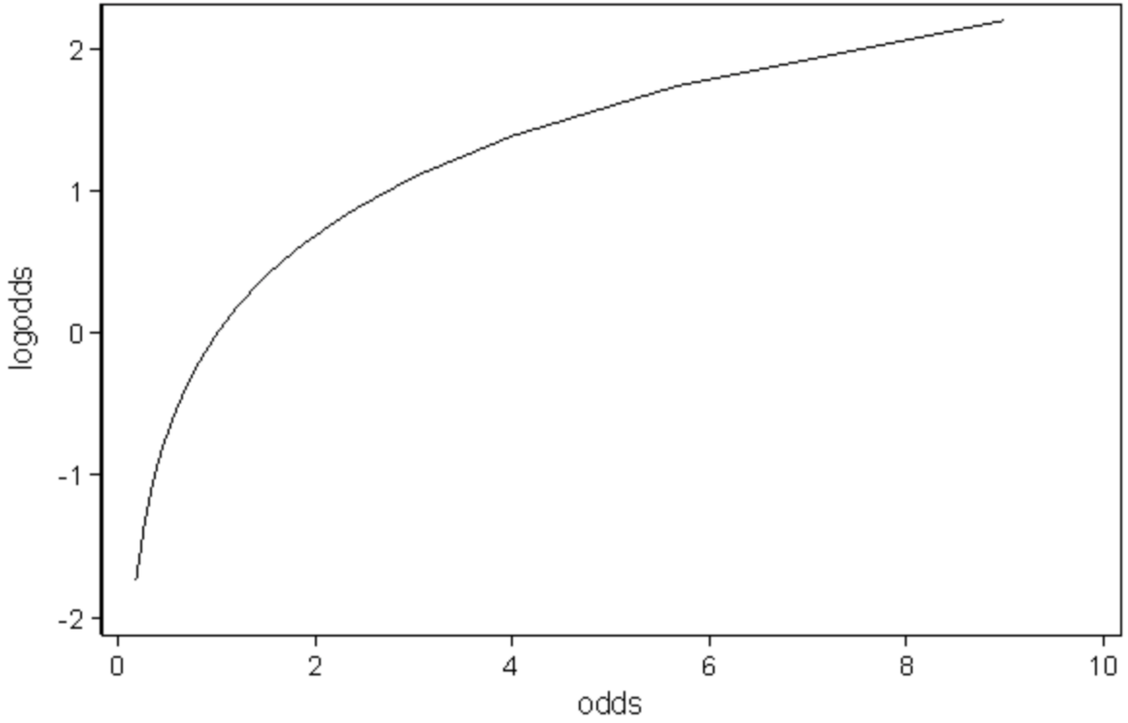
- Win Probability p
  - 0.2, 0.5, 0.8
  - [0, 1]
- Odds
  - $odds(p) = \frac{p}{1-p}$
  - $p = 0.8 \Rightarrow Odds = 4$
  - $p = 0.5 \Rightarrow Odds = 1$
  - $p = 0.1 \Rightarrow Odds = 0.1111\ldots$
  - [0, inf)



Reference: https://stats.idre.ucla.edu/other/mult-pkg/faq/general/faq-how-do-i-interpret-odds-ratios-in-logistic-regression/

Log-Odds (Logit)

- Logit
  - $logit(p) = log(\frac{p}{1-p})$
  - *p = 0.8* ➔ *logit = 1.3863*
  - *p = 0.5* ➔ *logit = 0*
  - *p = 0.1* ➔ *logit = -0.9542*
  - (-inf, inf)



Logistic Regression

- Linear Modeling of Logit

$$logit(p) = log(\frac{p}{1-p}) = \beta_0 + \beta_1 x_1 + \cdots + \beta_k x_k$$

$$\Downarrow$$

$$\frac{1-p}{p} = \frac{1}{exp(\beta_0 + \beta_1 x_1 + \cdots + \beta_k x_k)}$$

$$\Downarrow$$

$$p = \frac{exp(\beta_0 + \beta_1 x_1 + \cdots + \beta_k x_k)}{1 + exp(\beta_0 + \beta_1 x_1 + \cdots + \beta_k x_k)} \Rightarrow p = \frac{1}{1 + b^{-(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_m x_m)}}$$

where usually $b = e$.

Logistic Regression

- Vector form
  - Inner product

$$p = \frac{1}{1 + b^{-(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_m x_m)}} \Rightarrow h_\theta(X) = \frac{1}{1 + e^{-\theta^T X}} = \Pr(Y = 1 \mid X; \theta)$$

Logistic Regression

- Vector form
  - Inner product

$$p = \frac{1}{1 + b^{-(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_m x_m)}} \Rightarrow h_\theta(X) = \frac{1}{1 + e^{-\theta^T \boxed{X}}} = \Pr(Y = 1 \mid X; \theta)$$

Logistic Regression

- Vector form
  - Inner product

$$p = \frac{1}{1 + b^{-(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_m x_m)}} \quad \Rightarrow \quad h_\theta(X) = \frac{1}{1 + e^{-\theta^T \boxed{X}}} = \Pr(Y = 1 \mid X; \theta)$$

Image Representation Vector

0	1	0	0	0	1	0	0	0	0	1	1		…		…		…	

Logistic Regression

- Vector form
  - Inner product

$$p = \frac{1}{1 + b^{-(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_m x_m)}} \Rightarrow h_\theta(X) = \frac{1}{1 + e^{-\boxed{\theta}^T X}} = \Pr(Y = 1 \mid X; \theta)$$

Need to learn this to correctly predict for **X** (i.e. image representation vector)

Maximum Likelihood Estimate

- Need to estimate θ.
  - Learn from data ➔ Training pairs $(x_1, y_1), (x_2, y_2), (x_3, y_3), \ldots, (x_N, y_N)$
- Log Likelihood Function

Probability Function

$$h_\theta(X) = \frac{1}{1 + e^{-\theta^T X}} = \Pr(Y = 1 \mid X; \theta)$$

Likelihood Function

$$\begin{aligned} L(\theta \mid x) &= \Pr(Y \mid X; \theta) \\ &= \prod_i \Pr(y_i \mid x_i; \theta) \\ &= \prod_i \boxed{h_\theta(x_i)^{y_i} (1 - h_\theta(x_i))^{(1-y_i)}} \end{aligned}$$

Y is a binary variable following the Bernoulli Distribution

Log Likelihood Function

$$N^{-1} \log L(\theta \mid x) = N^{-1} \sum_{i=1}^{N} \log \Pr(y_i \mid x_i; \theta)$$

Negative Log Likelihood Function

$$-\frac{1}{N} \sum_{n=1}^{N} \left[y_n \log \hat{y}_n + (1 - y_n) \log(1 - \hat{y}_n) \right]$$

Negative Log Likelihood (NLL) is the same as the Cross Entropy loss

Maximum Likelihood Estimate

- Need to estimate θ.
  - Learn from data ➔ Training pairs $(x_1, y_1), (x_2, y_2), (x_3, y_3), \ldots, (x_N, y_N)$
- Log Likelihood Function

Probability Function

$$h_\theta(X) = \frac{1}{1 + e^{-\theta^T X}} = \Pr(Y = 1 \mid X; \theta)$$

Likelihood Function

$$\begin{aligned} L(\theta \mid x) &= \Pr(Y \mid X; \theta) \\ &= \prod_i \Pr(y_i \mid x_i; \theta) \\ &= \prod_i \boxed{h_\theta(x_i)^{y_i} (1 - h_\theta(x_i))^{(1-y_i)}} \end{aligned}$$

Y is a binary variable following the Bernoulli Distribution

Log Likelihood Function

$$N^{-1} \log L(\theta \mid x) = N^{-1} \sum_{i=1}^{N} \log \Pr(y_i \mid x_i; \theta)$$

Negative Log Likelihood Function

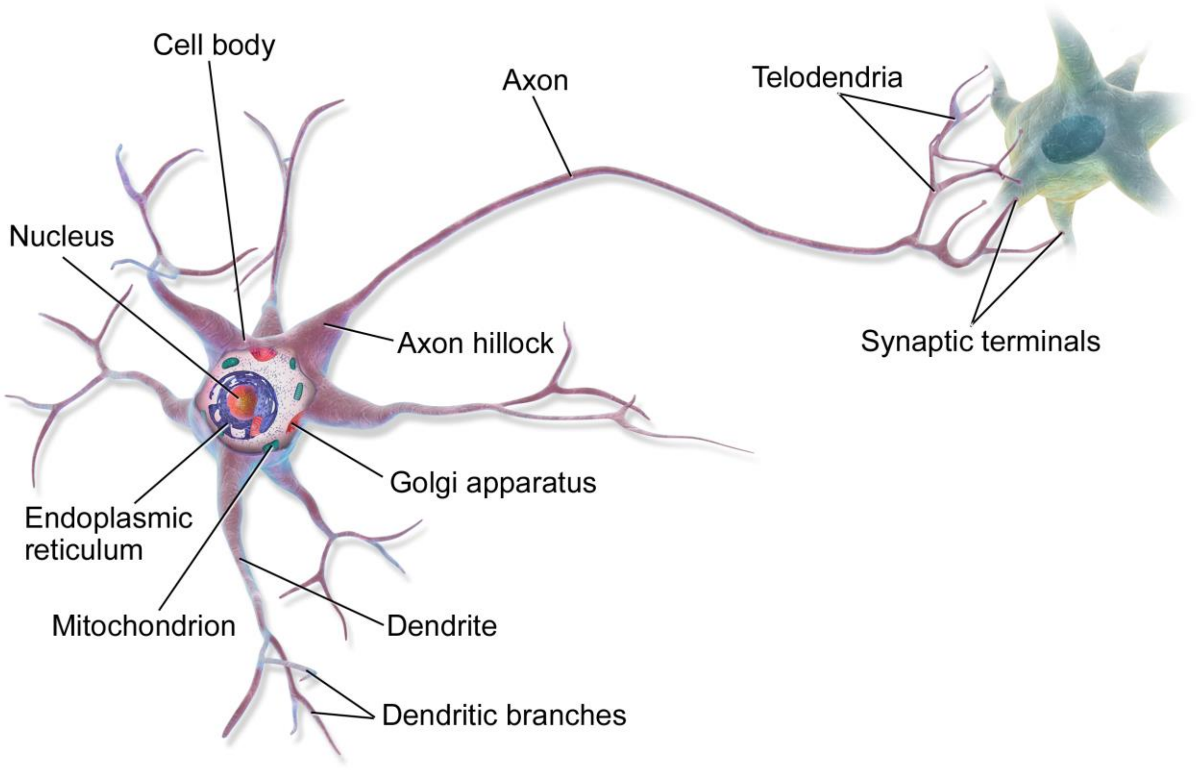
$$-\frac{1}{N} \sum_{n=1}^{N} \left[y_n \log \hat{y}_n + (1 - y_n) \log(1 - \hat{y}_n) \right]$$

Minimize the negative log likelihood (NLL) by Gradient Descent

Neural Networks

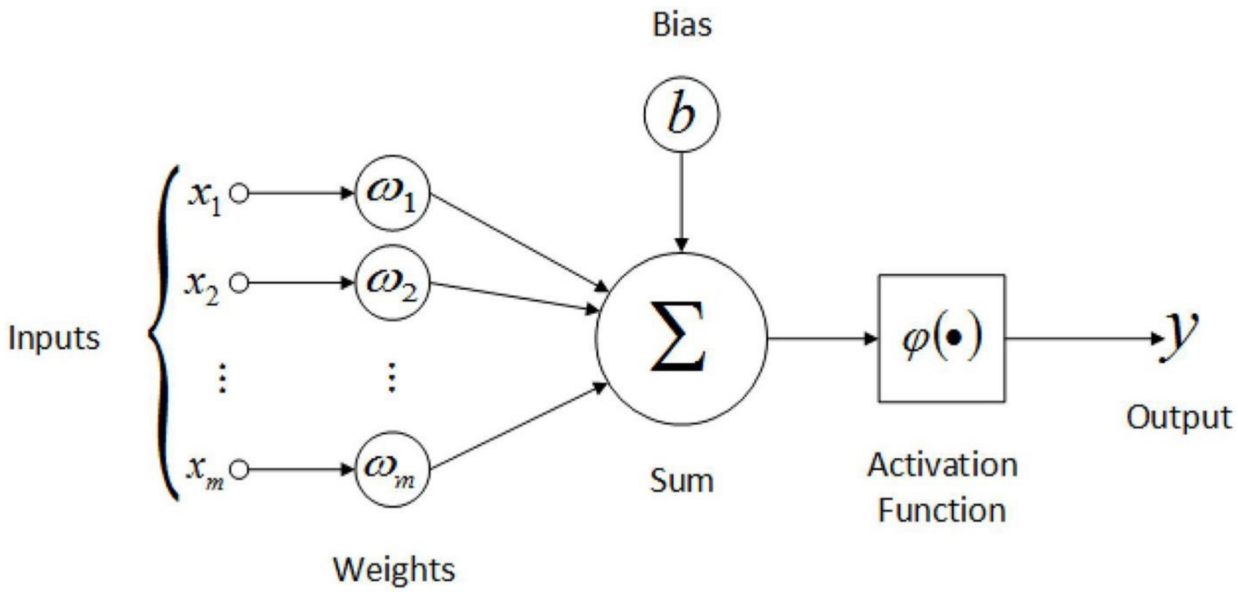
Neural Networks

- Neuron



Neural Networks

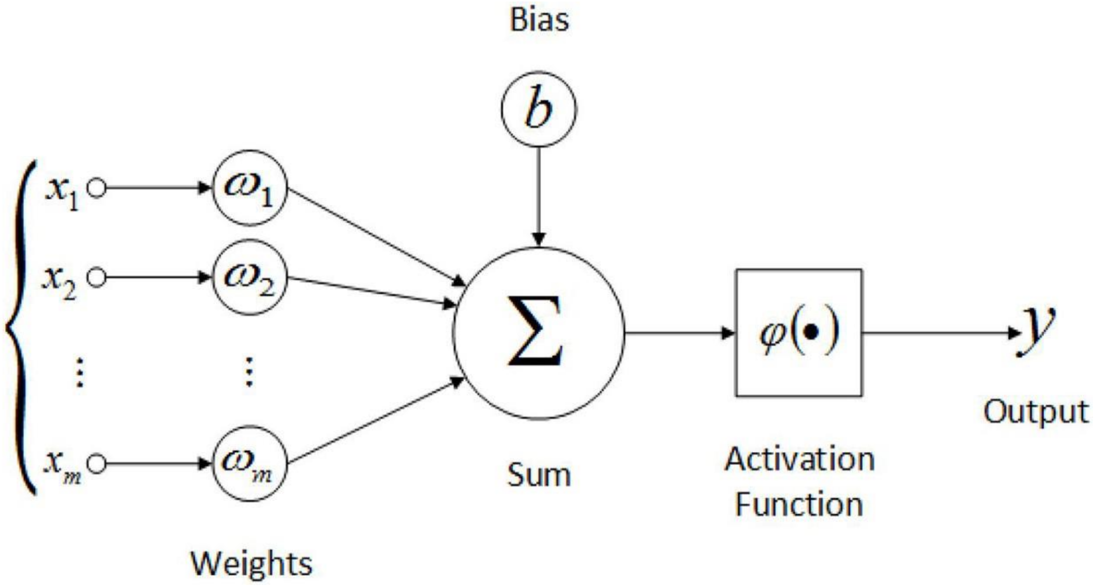
- Artificial neuron



Neural Networks

- Artificial neuron

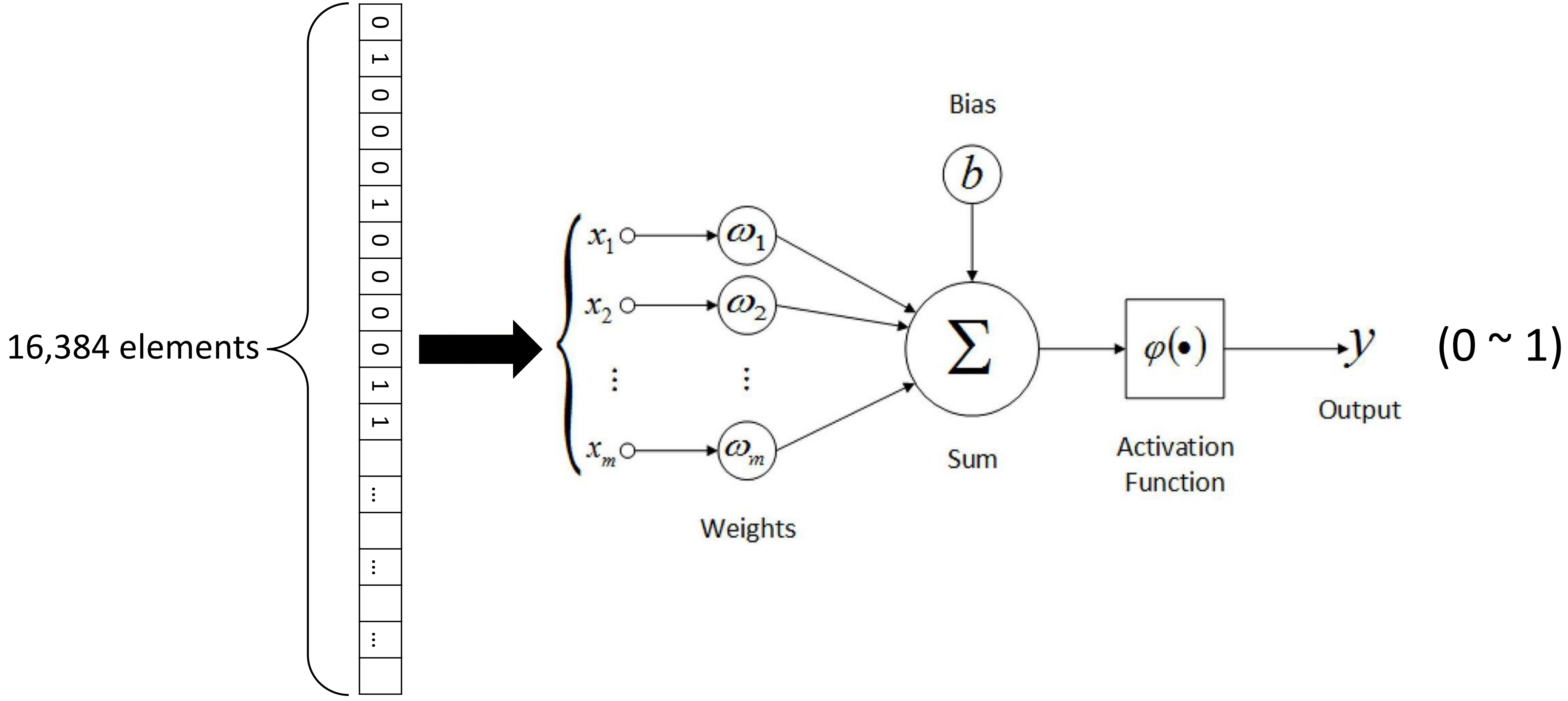
128 X 128 Image



(Man or woman?)

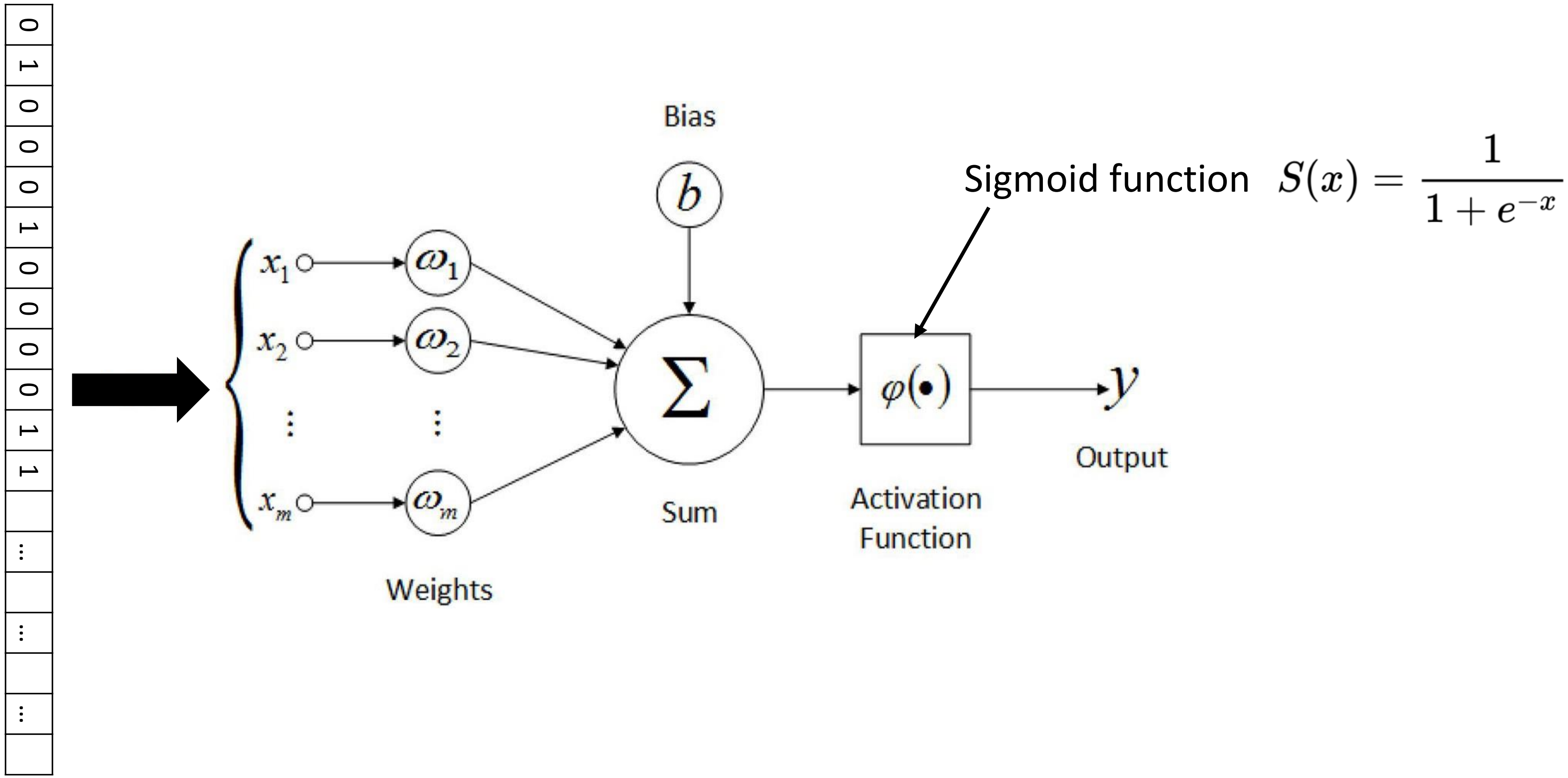
Neural Networks

- Artificial neuron



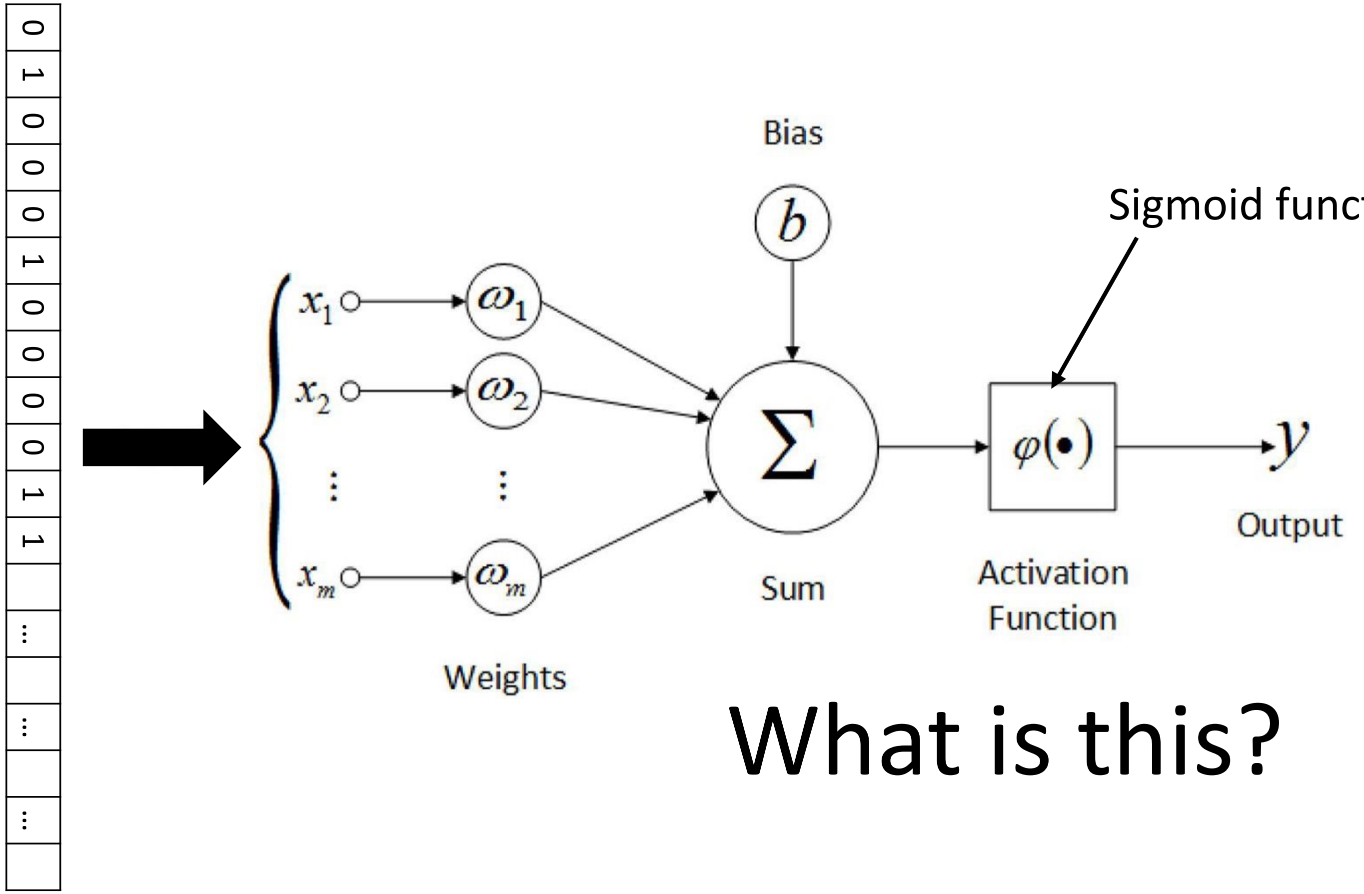
Neural Networks

- Artificial neuron



Neural Networks

- Artificial neuron

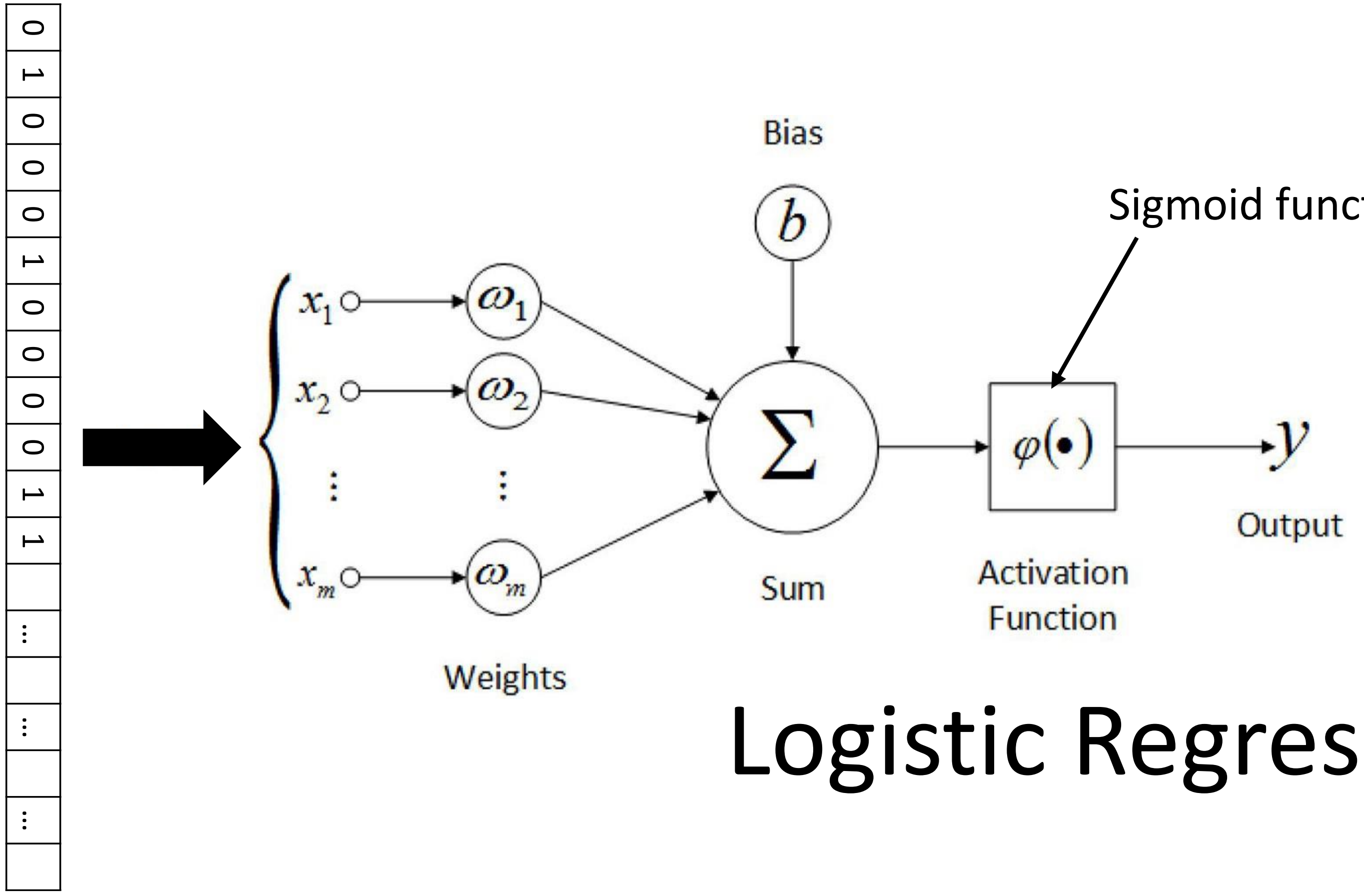


Sigmoid function $S(x) = \dfrac{1}{1+e^{-x}}$

What is this?

Neural Networks

- Logistic Regression

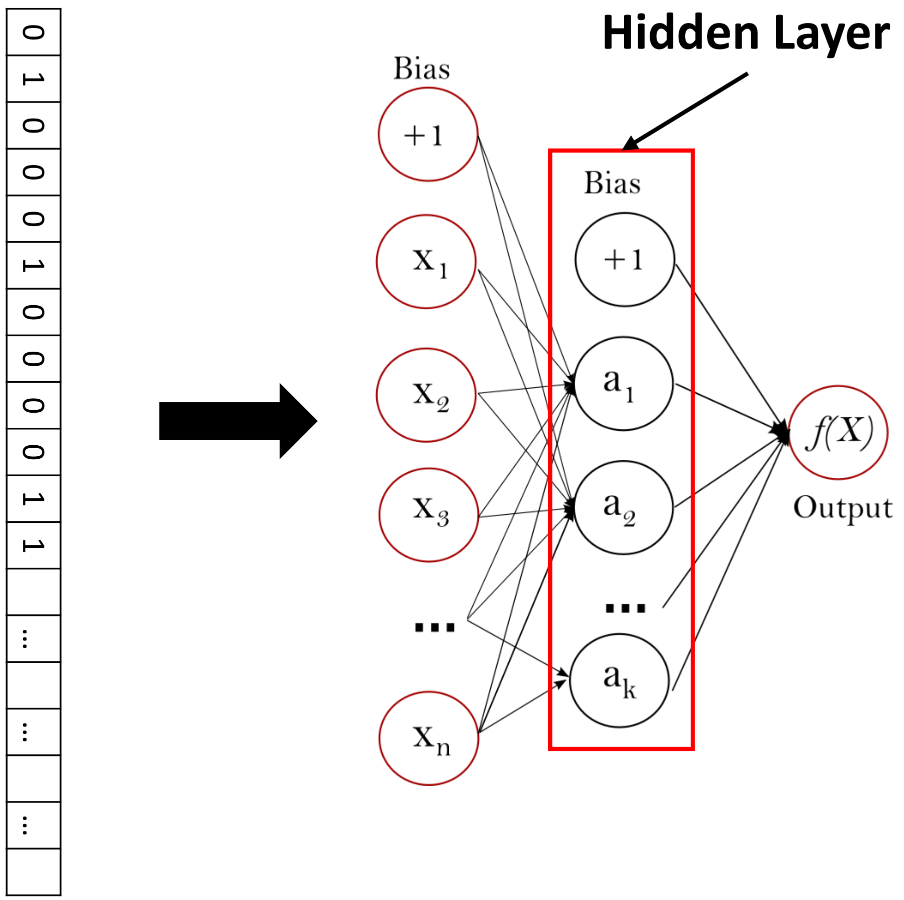


Sigmoid function $S(x) = \dfrac{1}{1+e^{-x}}$

Logistic Regression

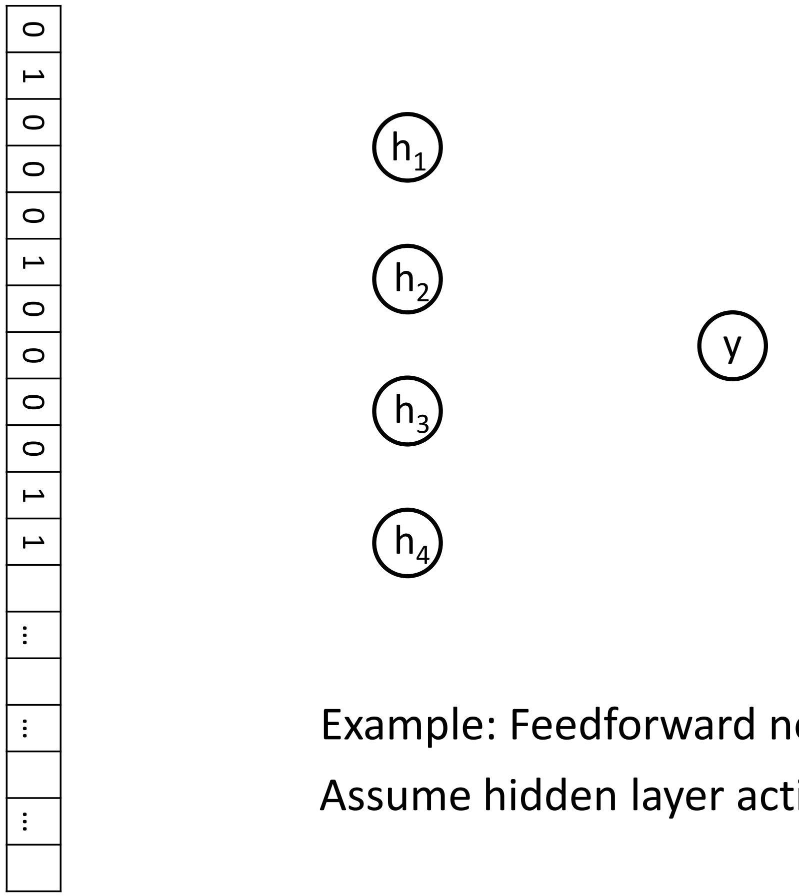
Neural Networks

- Another layer of neurons.



Neural Networks

- Another layer of neurons.

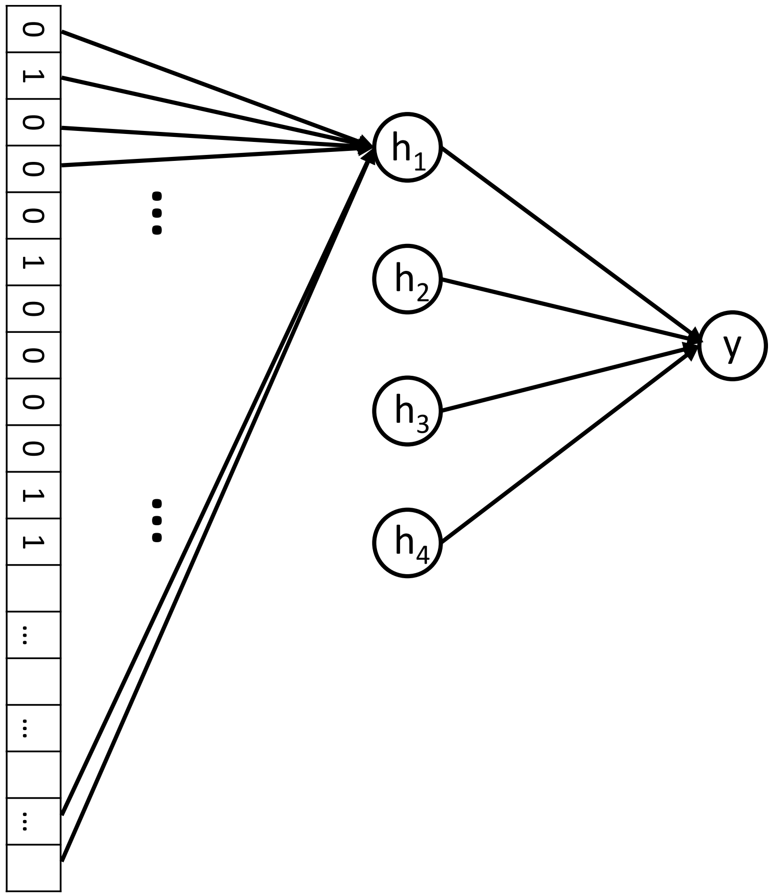


Example: Feedforward neural network with one hidden layer of 4 neurons.

Assume hidden layer activation function ➔ sigmoid function

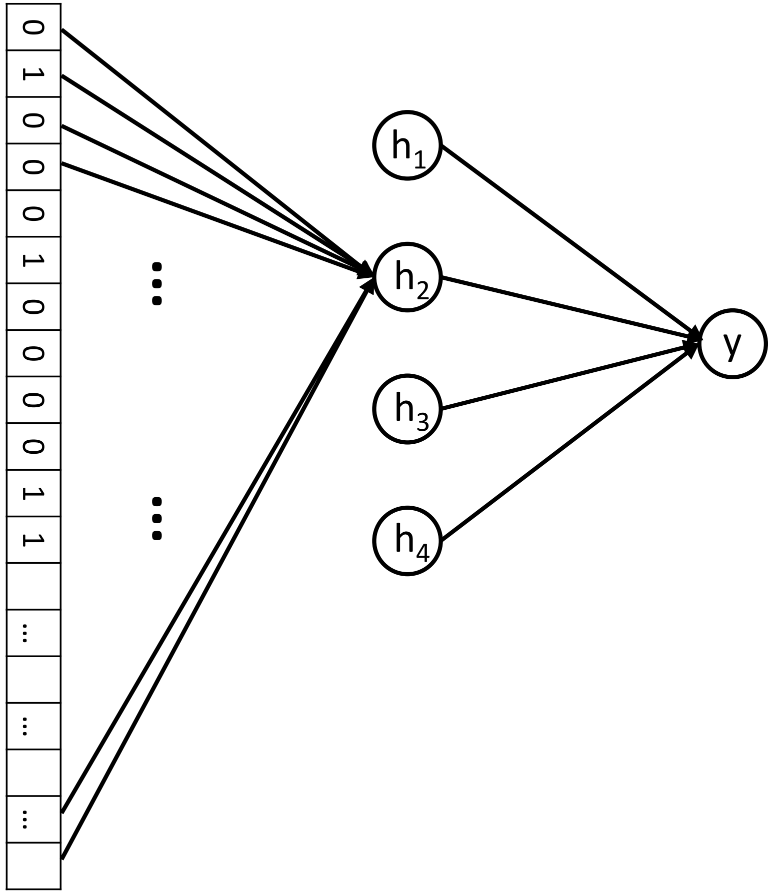
Neural Networks

- Another layer of neurons.



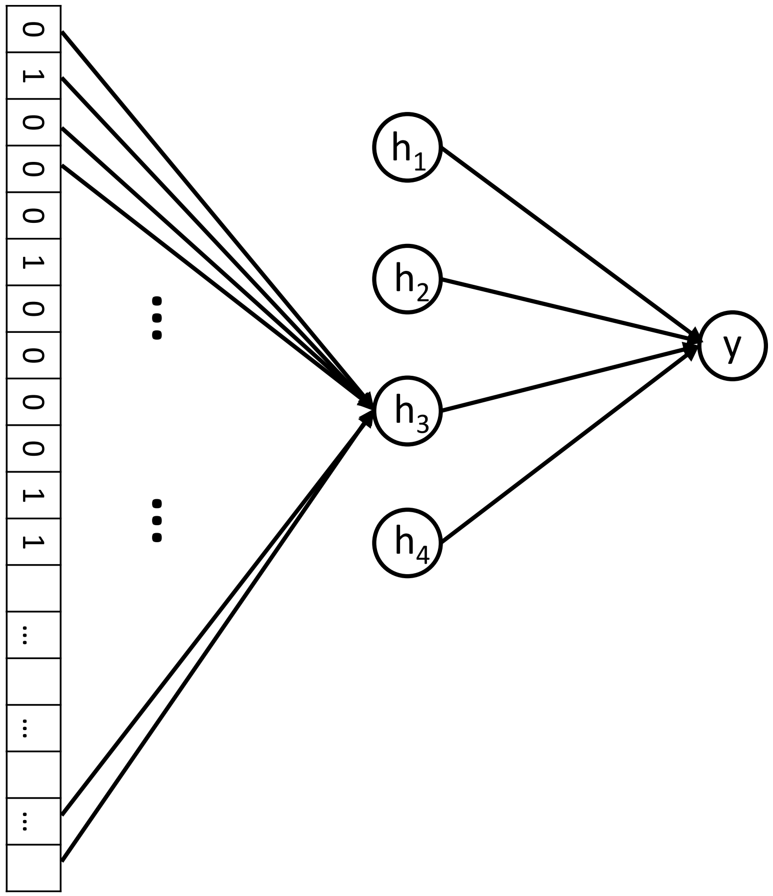
Neural Networks

- Another layer of neurons.



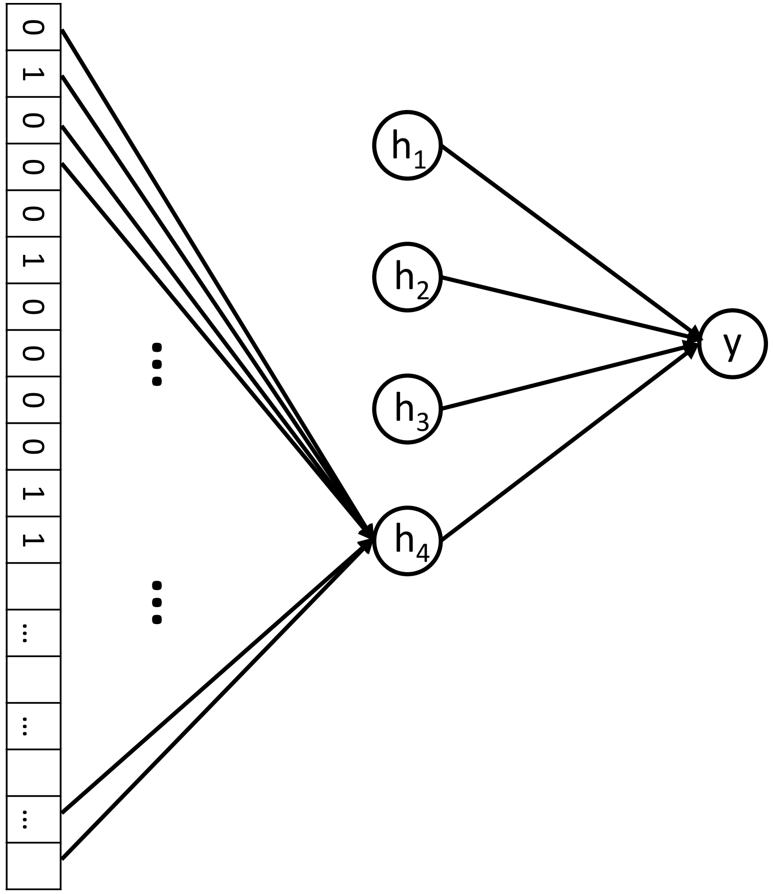
Neural Networks

- Another layer of neurons.



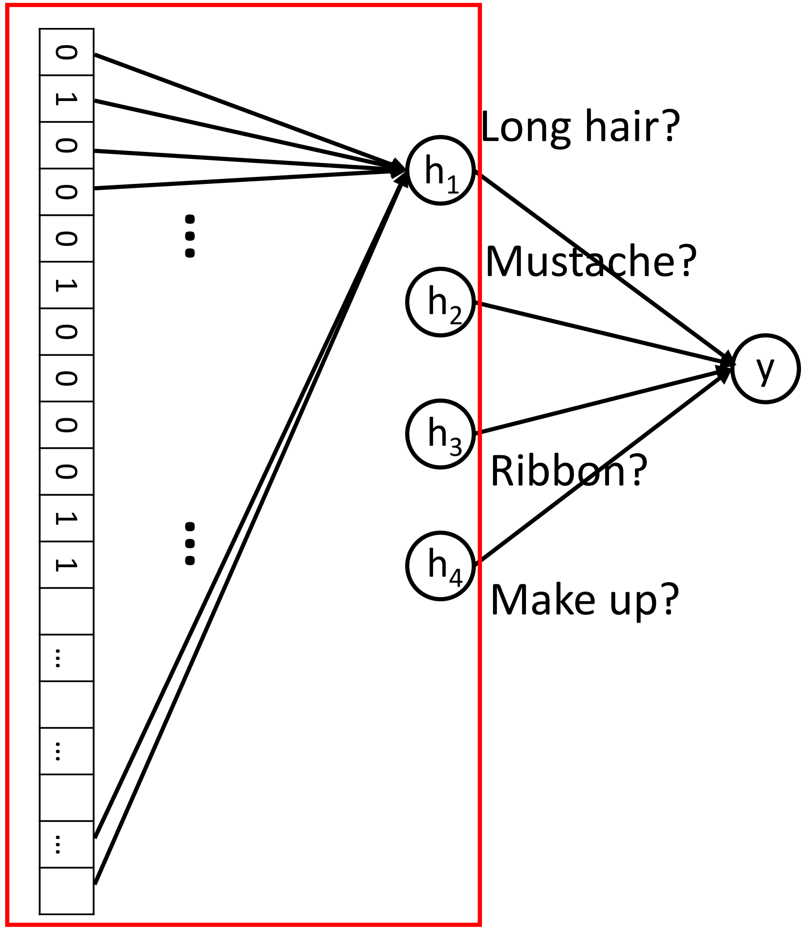
Neural Networks

- Another layer of neurons.



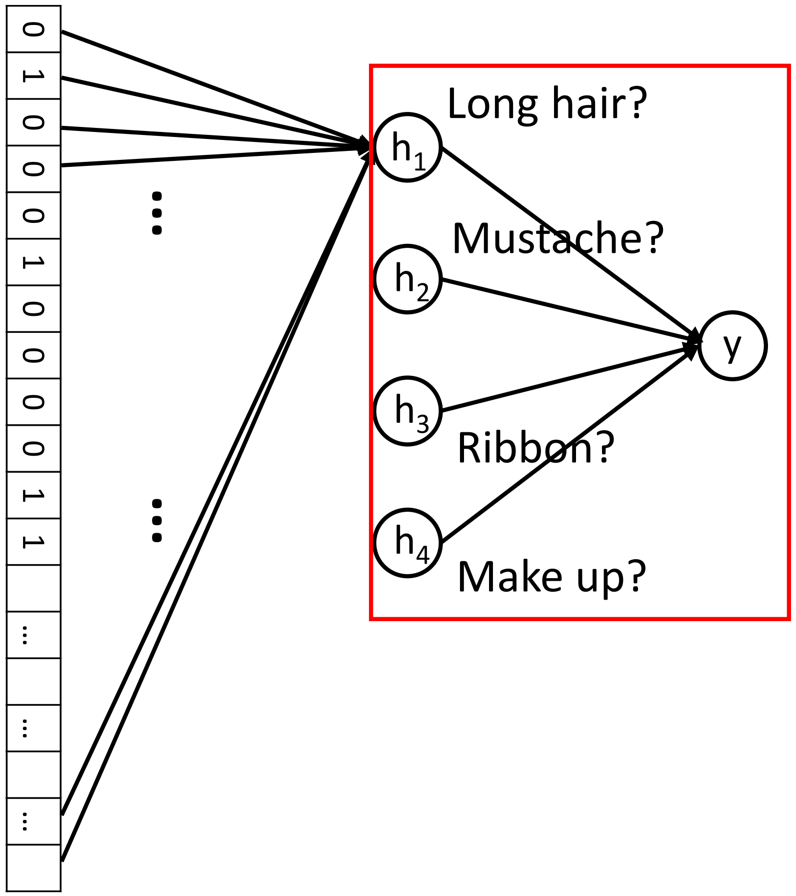
Neural Networks

- 4 logistic regression classifiers (when using sigmoid activation)



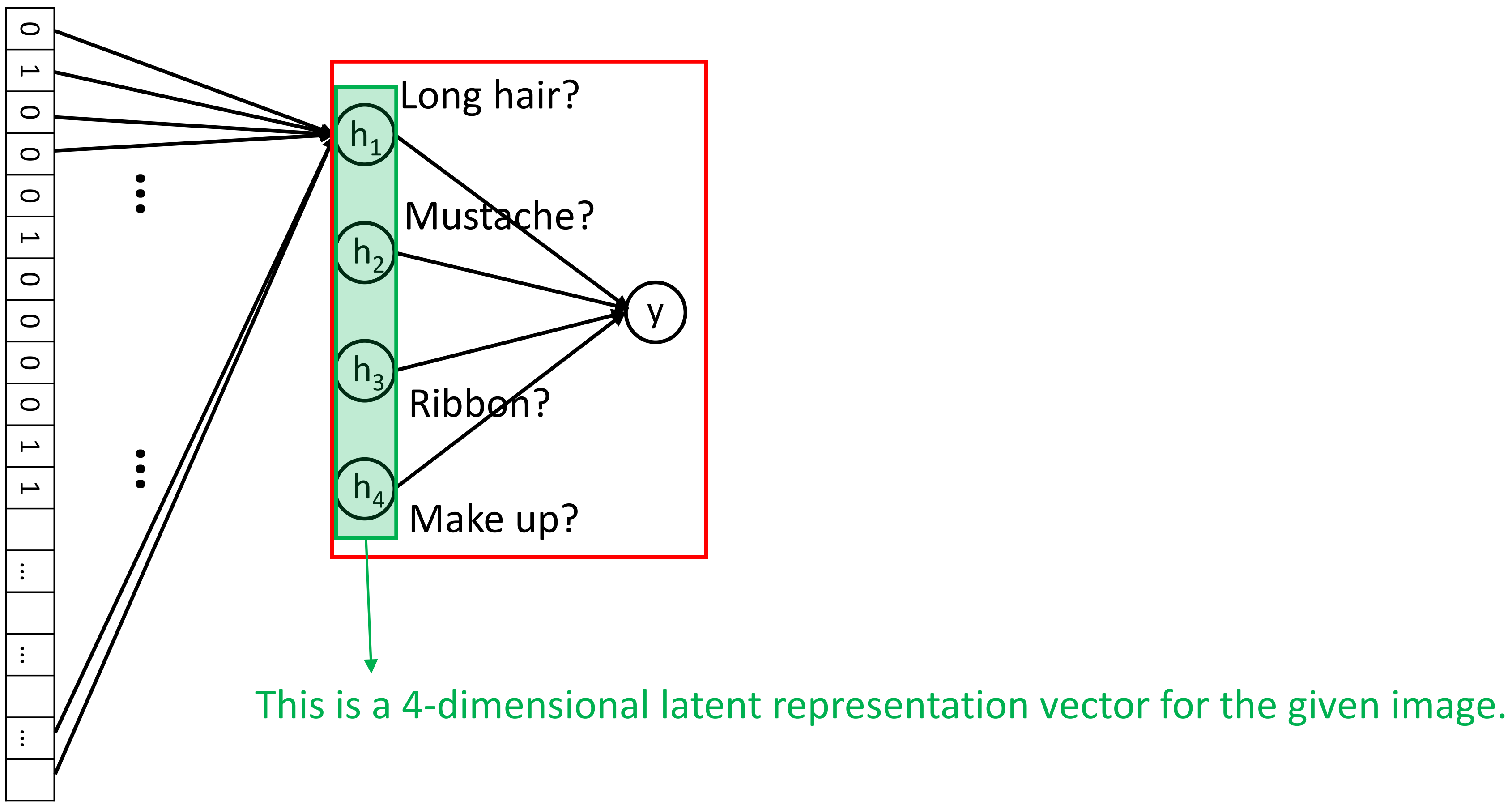
Neural Networks

- Higher-level logistic regression classifiers



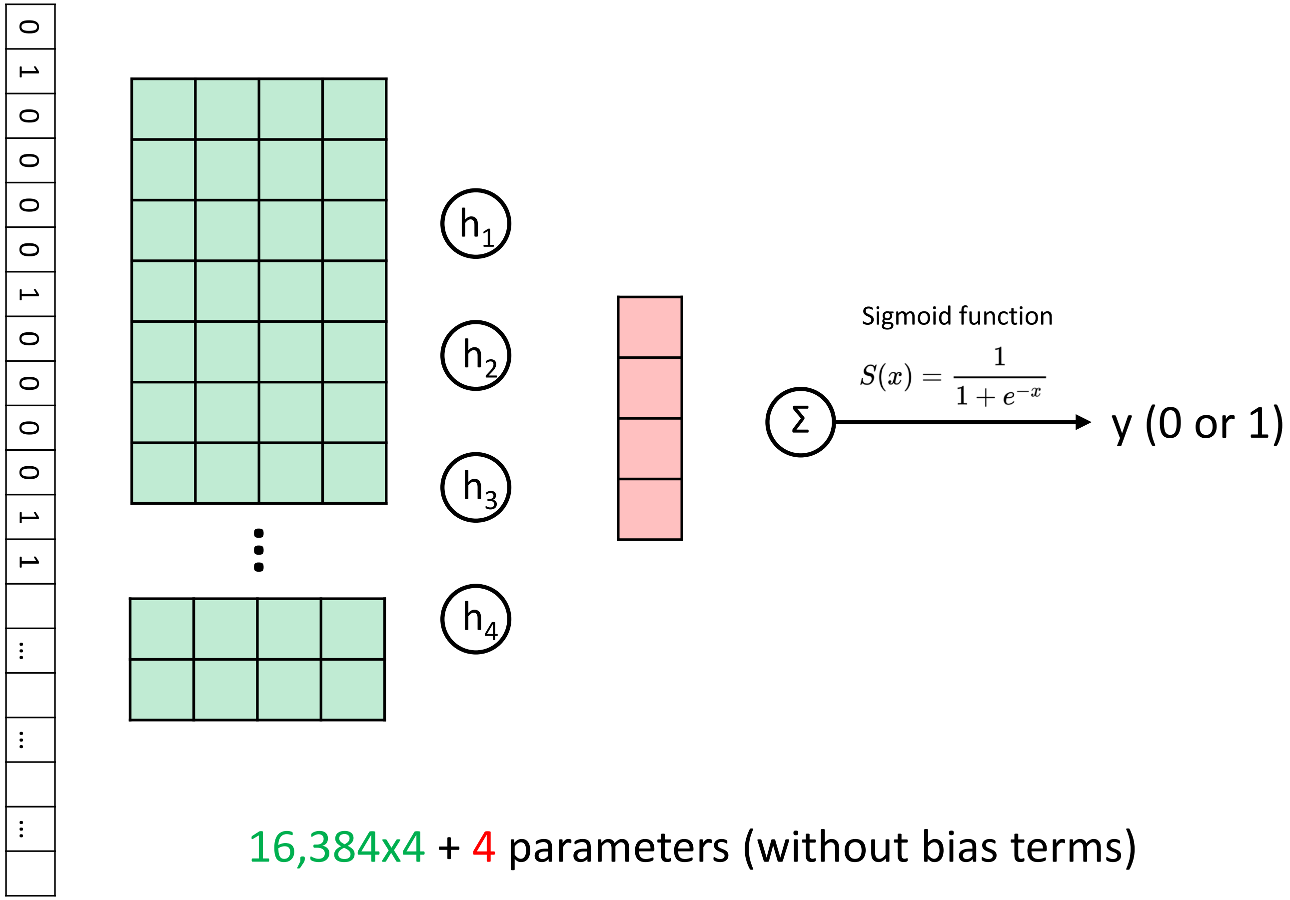
Neural Networks

- Higher-level logistic regression classifiers



Training

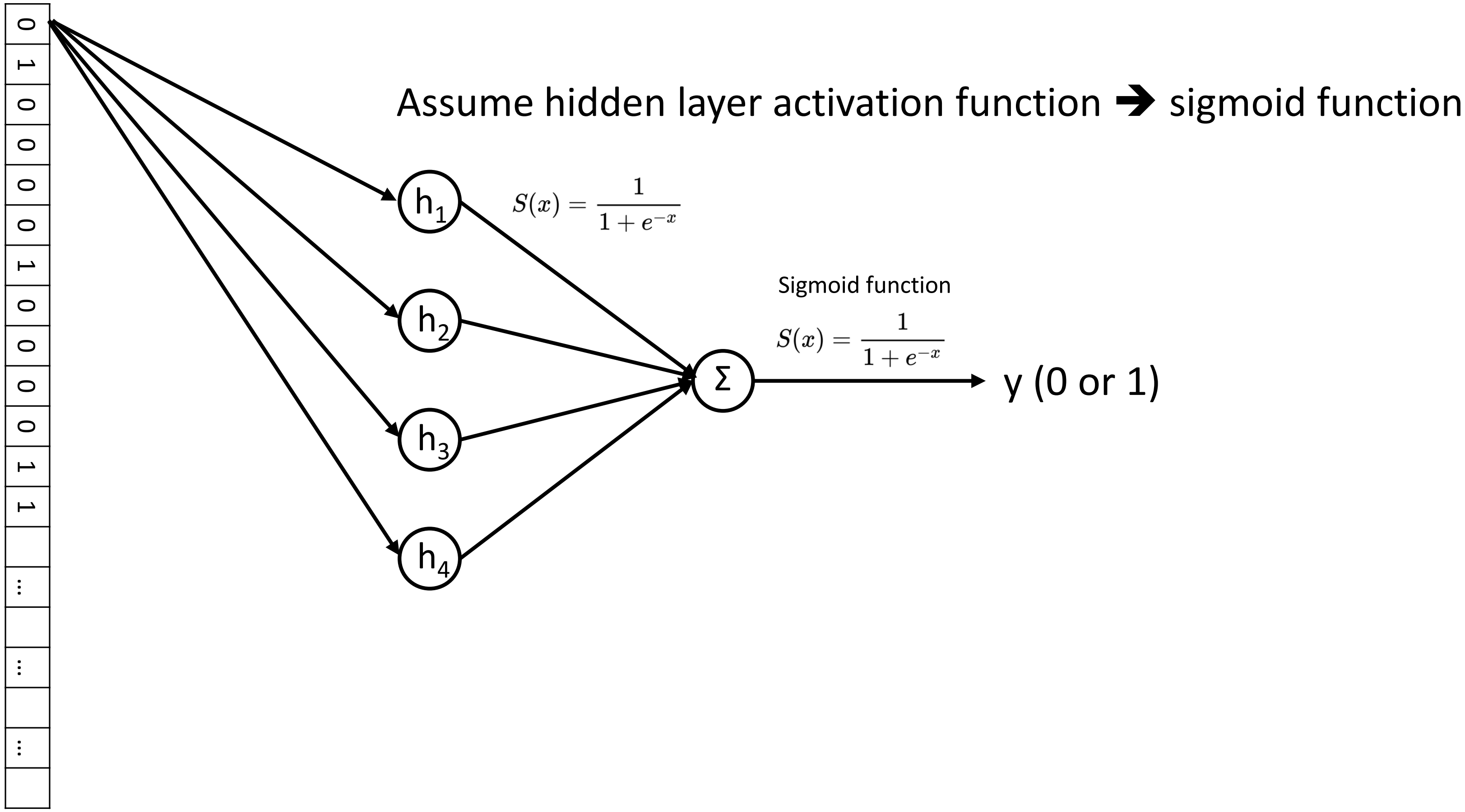
- Need to estimate all parameter values



16,384x4 + 4 parameters (without bias terms)

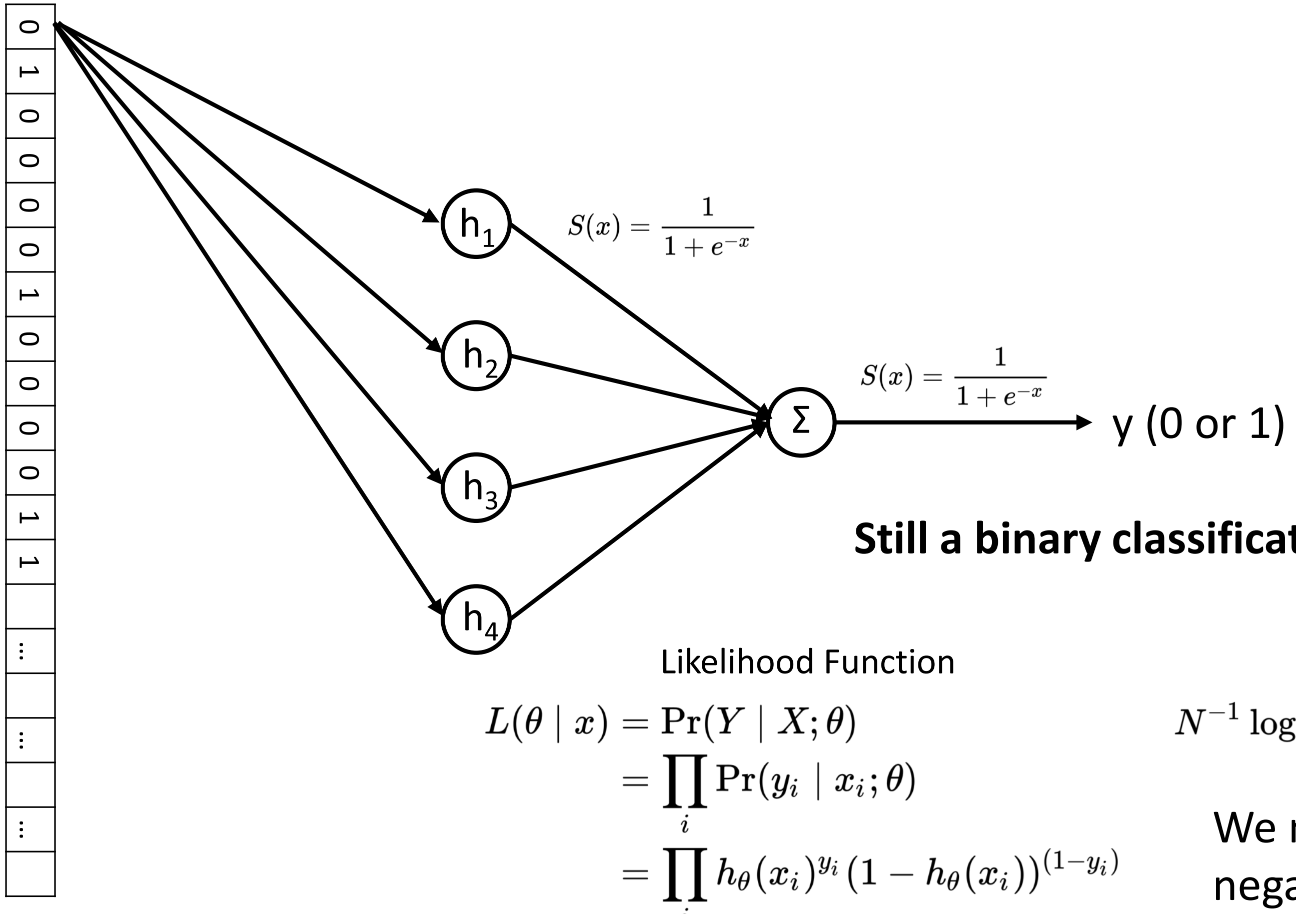
Training

- Need to estimate all parameter values



Training

- Maximum likelihood estimation



Still a binary classification problem

Likelihood Function

$$
\begin{aligned}
L(\theta \mid x) &= \Pr(Y \mid X; \theta) \\
&= \prod_i \Pr(y_i \mid x_i; \theta) \\
&= \prod_i h_\theta(x_i)^{y_i} (1 - h_\theta(x_i))^{(1-y_i)}
\end{aligned}
$$

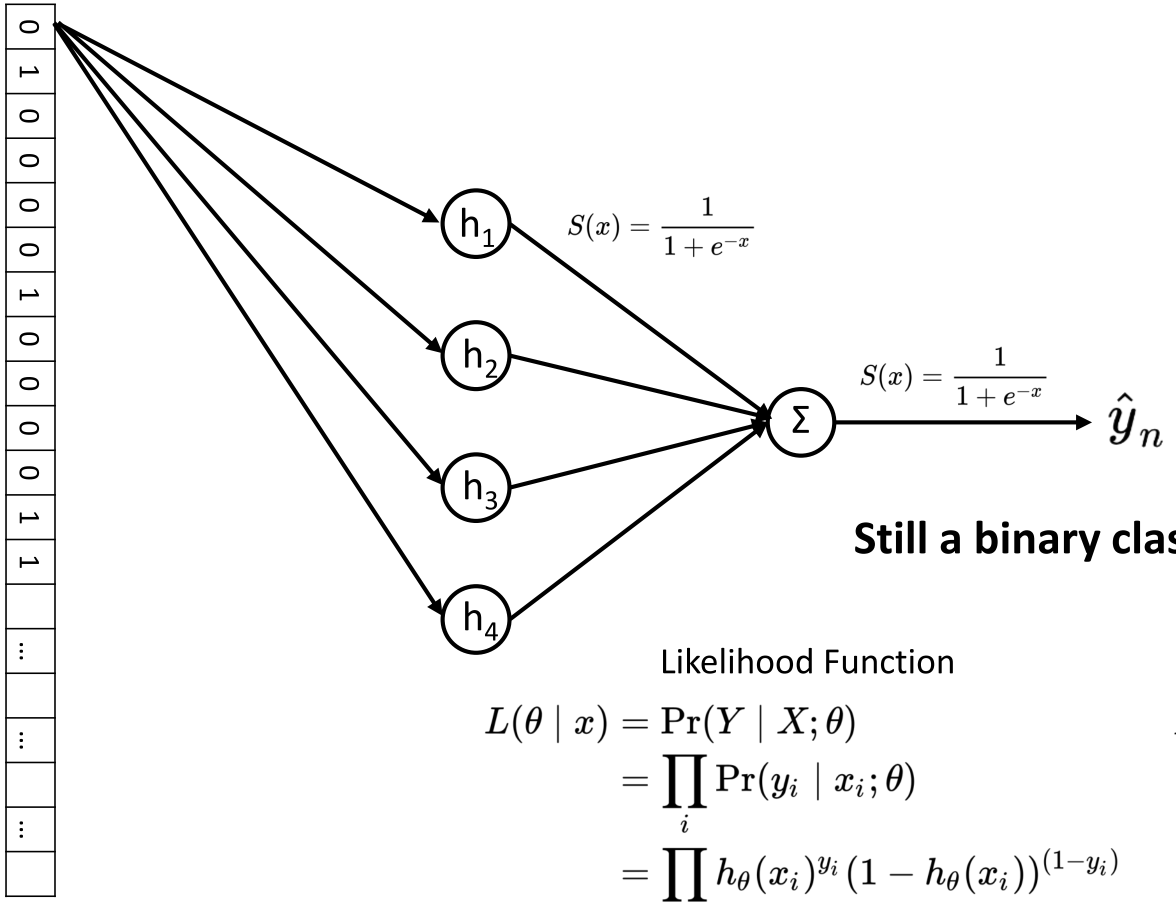
Log Likelihood Function

$$
N^{-1} \log L(\theta \mid x) = N^{-1} \sum_{i=1}^{N} \log \Pr(y_i \mid x_i; \theta)
$$

We need to minimize negative log-likelihood function

Training

- Minimize NLL with gradient descent.



Still a binary classification problem

Likelihood Function

$$L(\theta \mid x) = \Pr(Y \mid X; \theta) = \prod_i \Pr(y_i \mid x_i; \theta) = \prod_i h_\theta(x_i)^{y_i} (1 - h_\theta(x_i))^{(1-y_i)}$$

Log Likelihood Function

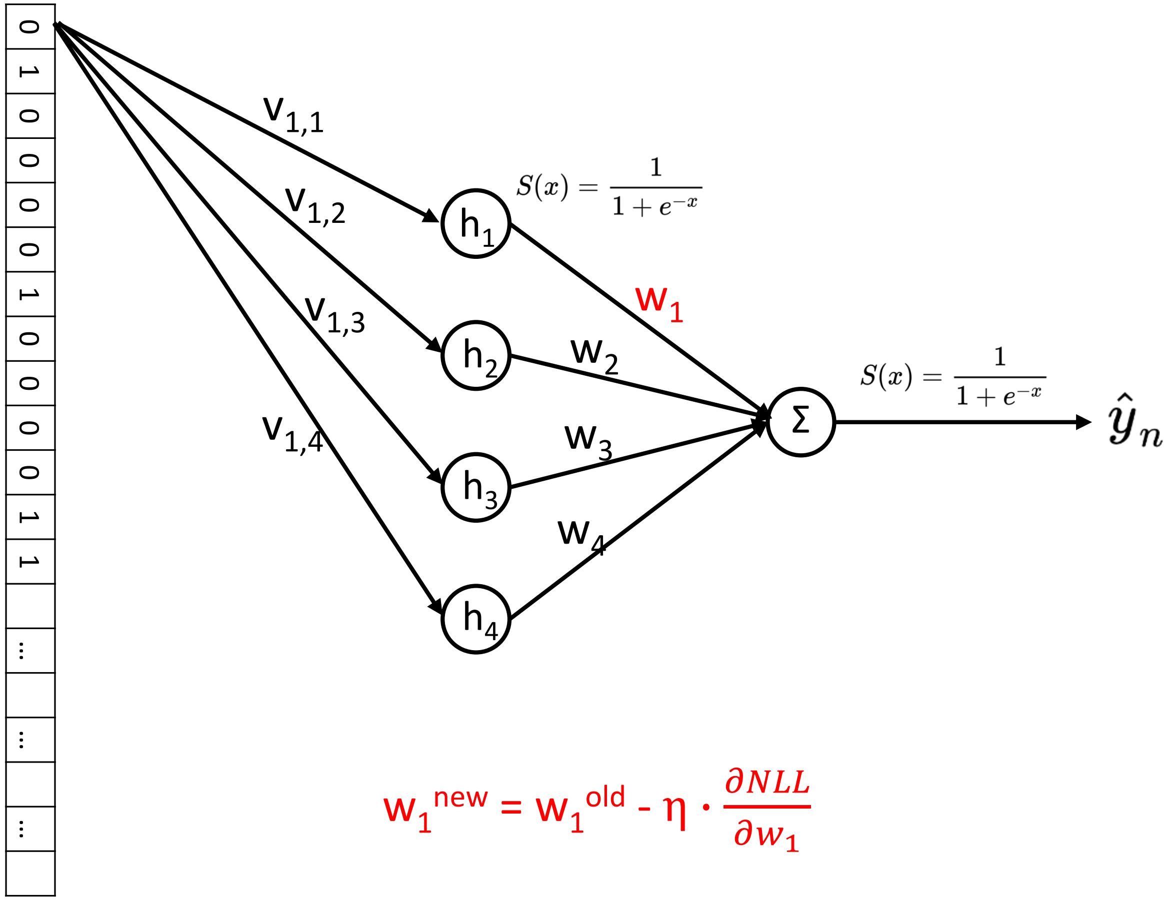
$$N^{-1} \log L(\theta \mid x) = N^{-1} \sum_{i=1}^{N} \log \Pr(y_i \mid x_i; \theta)$$

We need to minimize

$$-\frac{1}{N} \sum_{n=1}^{N} \left[y_n \log \hat{y}_n + (1 - y_n) \log(1 - \hat{y}_n) \right]$$

Training

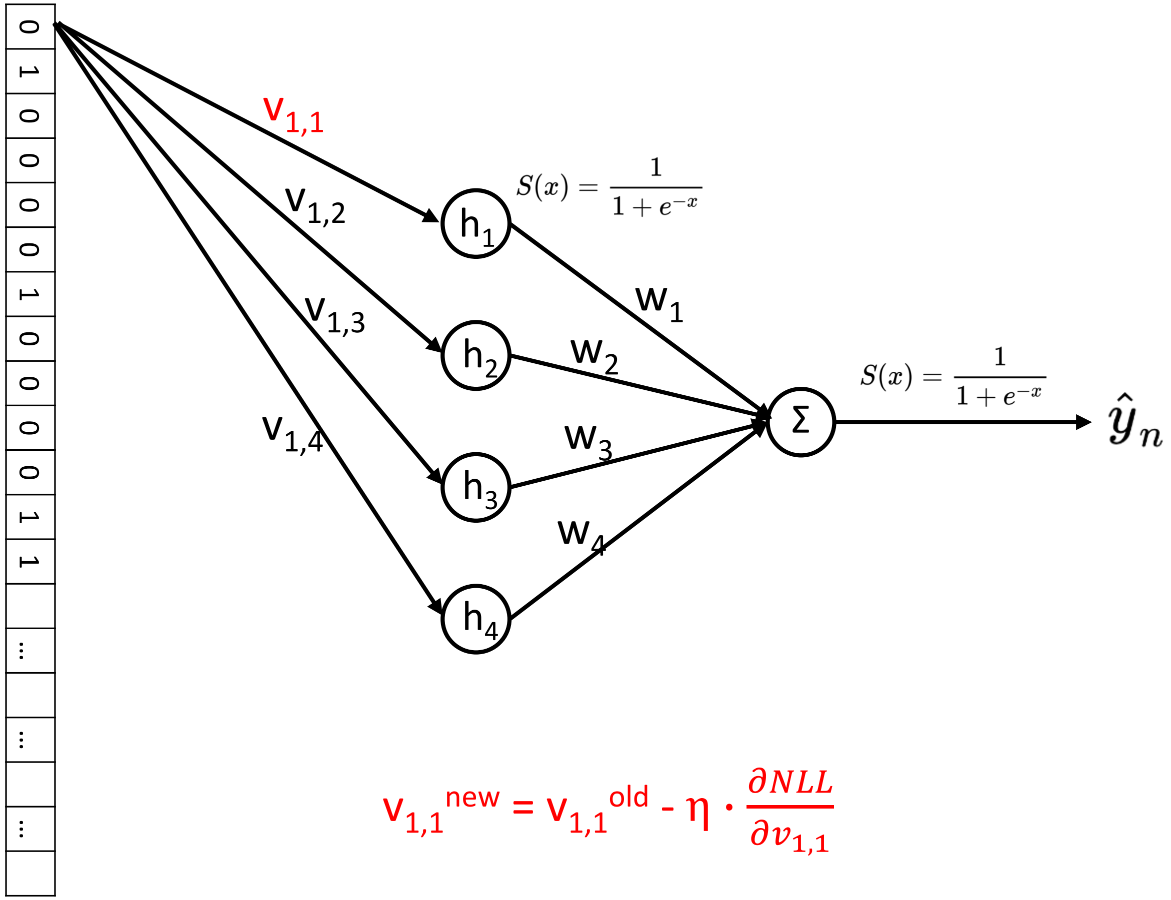
- Updating w_1



$$w_1^{new} = w_1^{old} - \eta \cdot \frac{\partial NLL}{\partial w_1}$$

Training

- Updating $v_{1,1}$



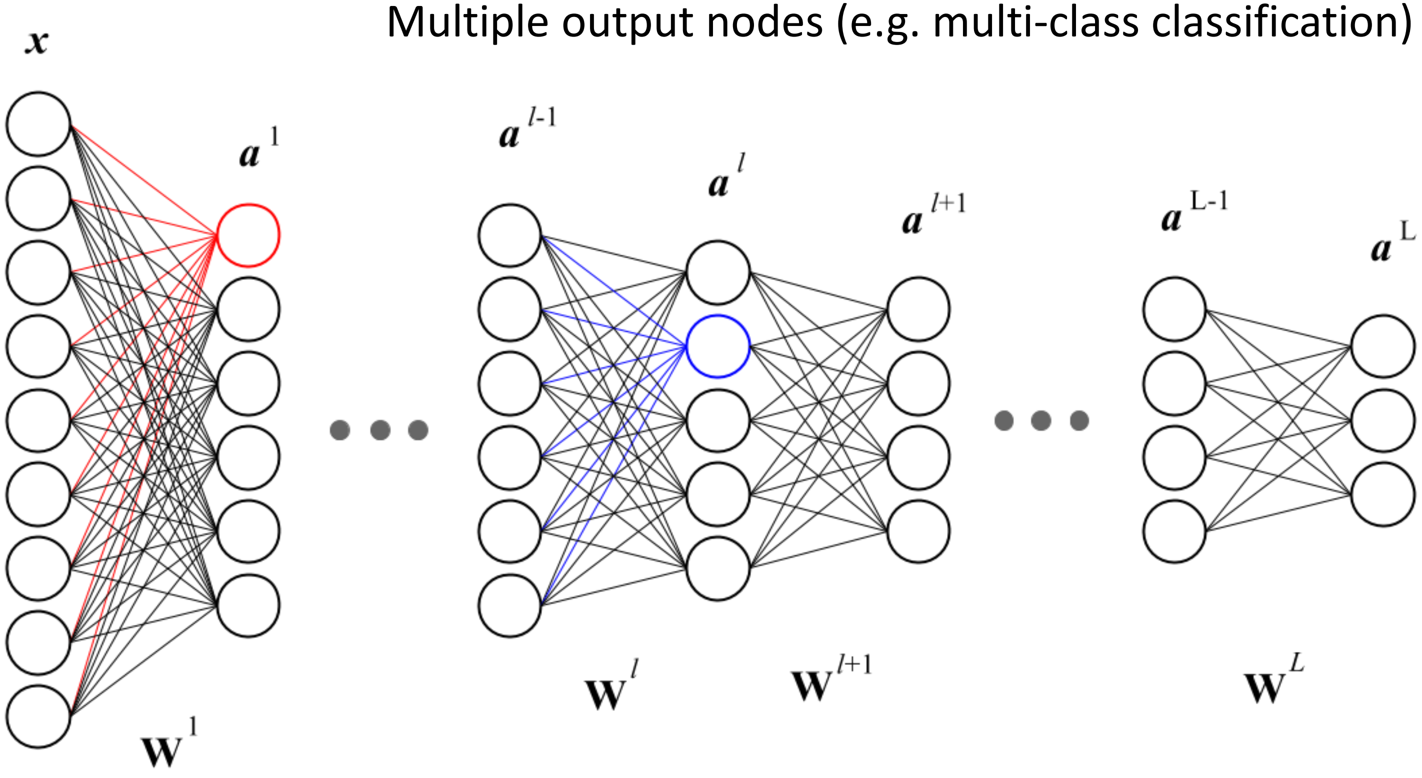
$$v_{1,1}^{new} = v_{1,1}^{old} - \eta \cdot \frac{\partial NLL}{\partial v_{1,1}}$$

Backpropagation

Training

- Backpropagation

Multiple hidden layers
Multiple output nodes (e.g. multi-class classification)



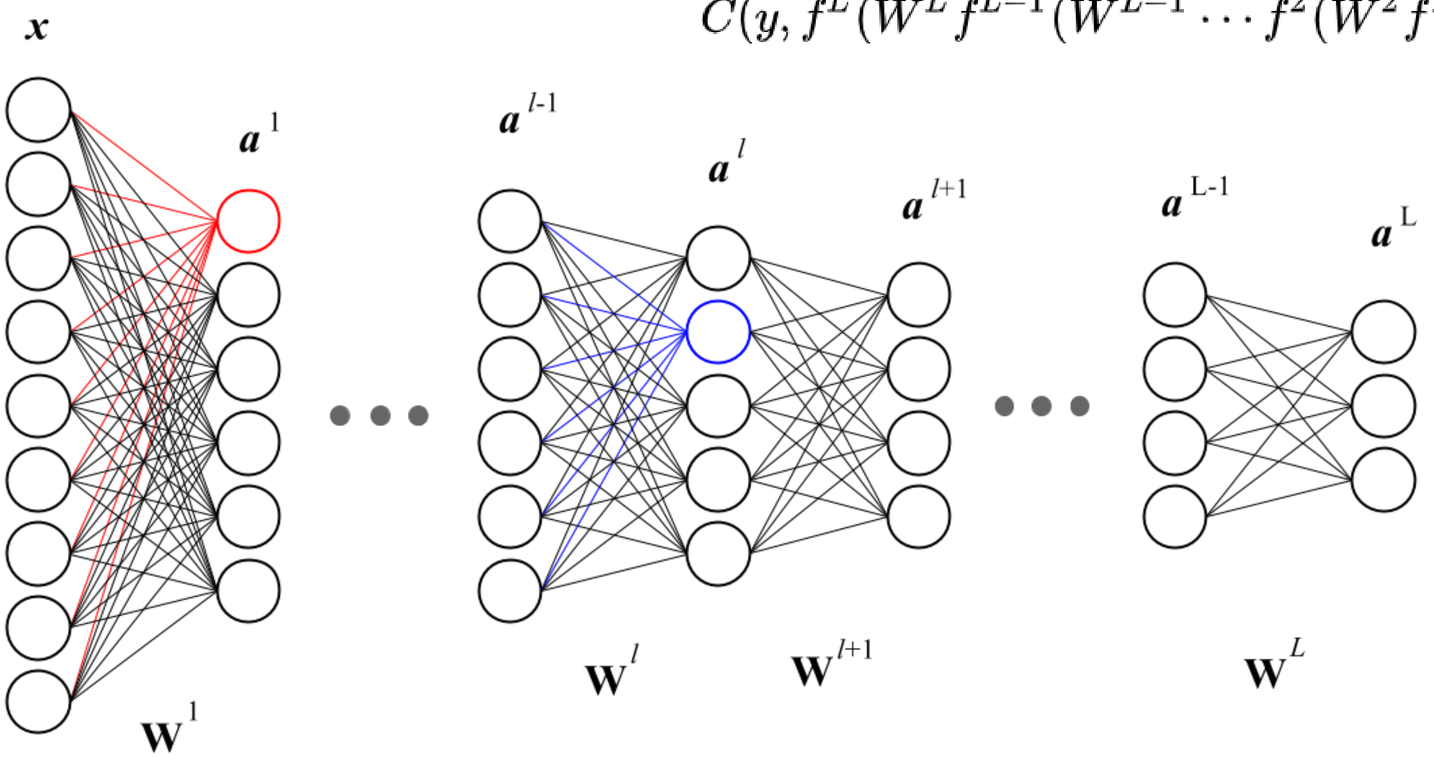
Reference: https://medium.com/@erikhallstrm/backpropagation-from-the-beginning-77356edf427d

Training

- Backpropagation

Given an input–output pair (x, y), the loss is:

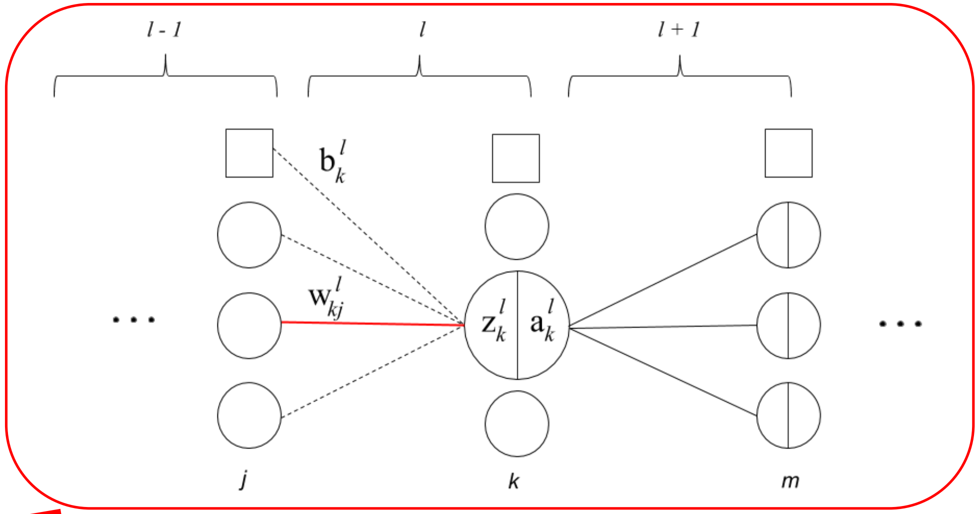
$$C(y, f^L(W^L f^{L-1}(W^{L-1} \cdots f^2(W^2 f^1(W^1 x)) \cdots)))$$

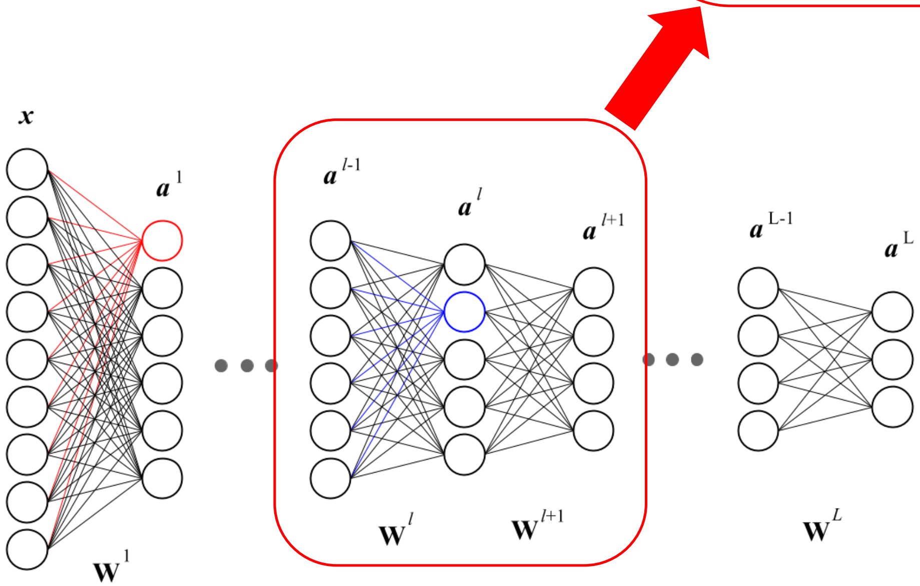


Reference: https://medium.com/@erikhallstrm/backpropagation-from-the-beginning-77356edf427d

Training

- Backpropagation





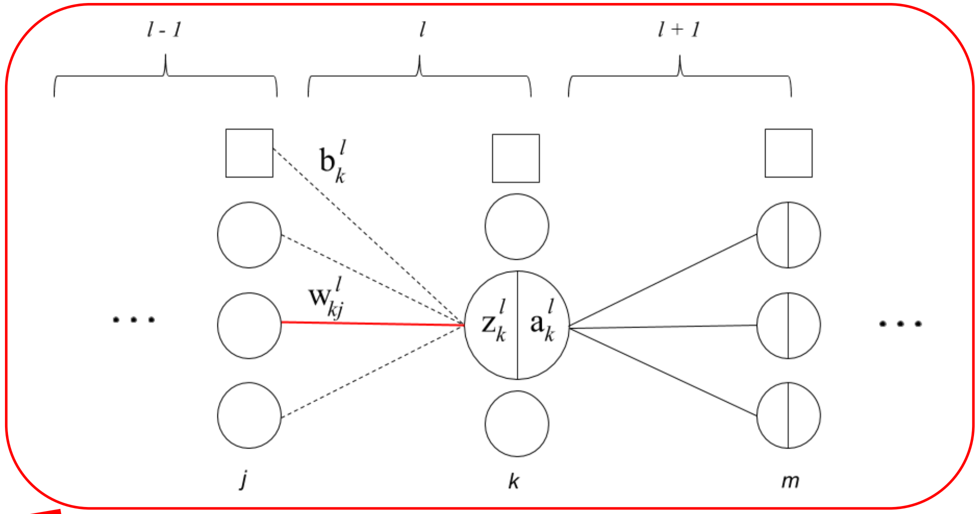
$$z_k^l = \sum_j w_{kj}^l a_j^{l-1} + b_k^l$$

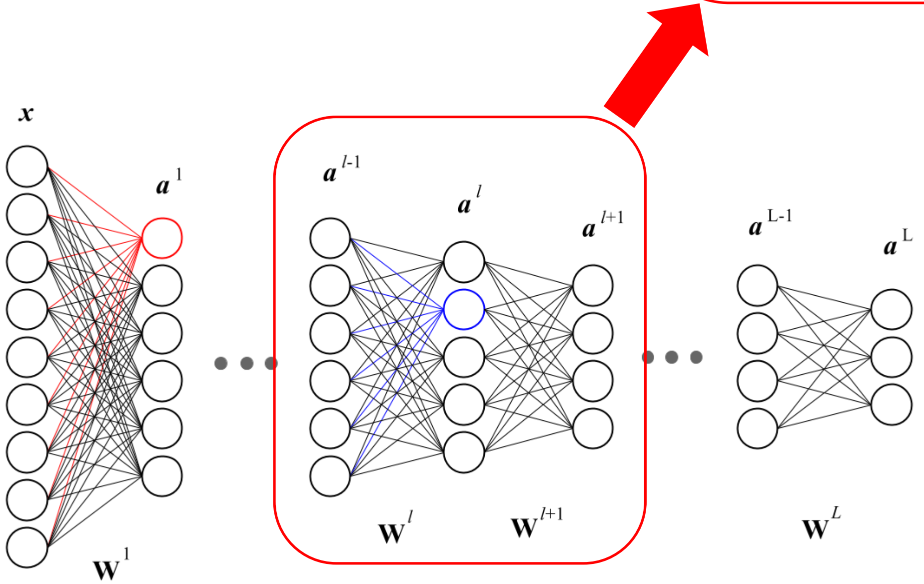
$$a_k^l = \sigma(z_k^l)$$

$$z_m^{l+1} = \sum_k w_{mk}^{l+1} a_k^l + b_m^l$$

Training

- Backpropagation



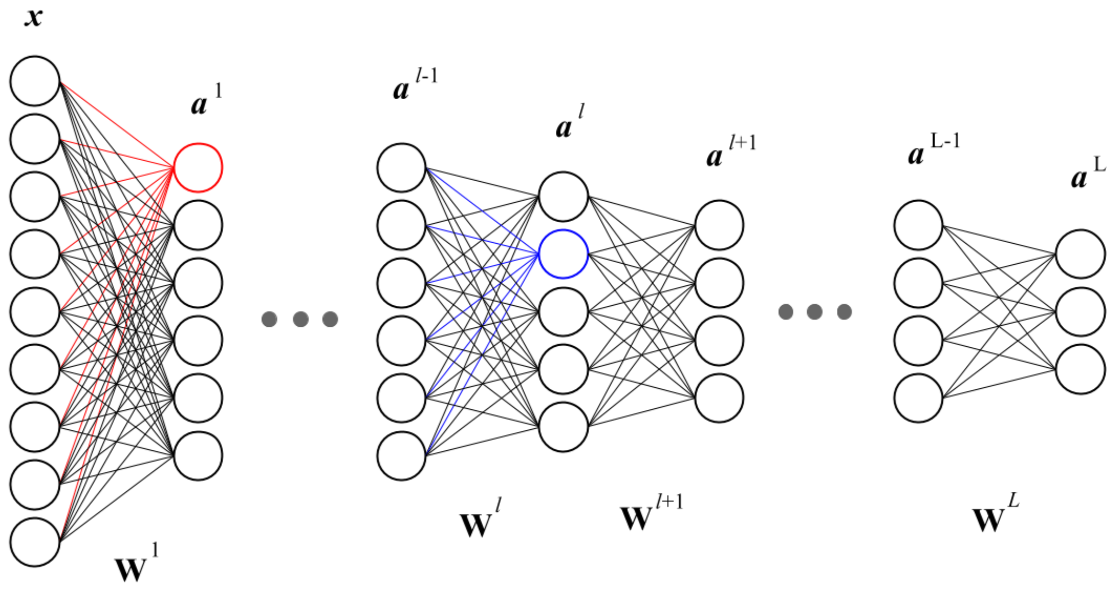


Derivative of the loss function C w.r.t. input **x**

$$\frac{dC}{da^L}\cdot\frac{da^L}{dz^L}\cdot\frac{dz^L}{da^{L-1}}\cdot\frac{da^{L-1}}{dz^{L-1}}\cdot\frac{dz^{L-1}}{da^{L-2}}\cdots\frac{da^1}{dz^1}\cdot\frac{\partial z^1}{\partial x}$$

Training

- Backpropagation



Derivative of the loss function C w.r.t. input **x**

$$\frac{dC}{da^L} \cdot \frac{da^L}{dz^L} \cdot \frac{dz^L}{da^{L-1}} \cdot \frac{da^{L-1}}{dz^{L-1}} \cdot \frac{dz^{L-1}}{da^{L-2}} \cdots \frac{da^1}{dz^1} \cdot \frac{\partial z^1}{\partial x}$$

$$\Downarrow$$

$$\frac{dC}{da^L} \cdot (f^L)' \cdot W^L \cdot (f^{L-1})' \cdot W^{L-1} \cdots (f^1)' \cdot W^1$$

$$\Downarrow$$

$$\nabla_x C = (W^1)^T \cdot (f^1)' \cdots \cdot (W^{L-1})^T \cdot (f^{L-1})' \cdot (W^L)^T \cdot (f^L)' \cdot \nabla_{a^L} C$$

Define delta (error at layer l)

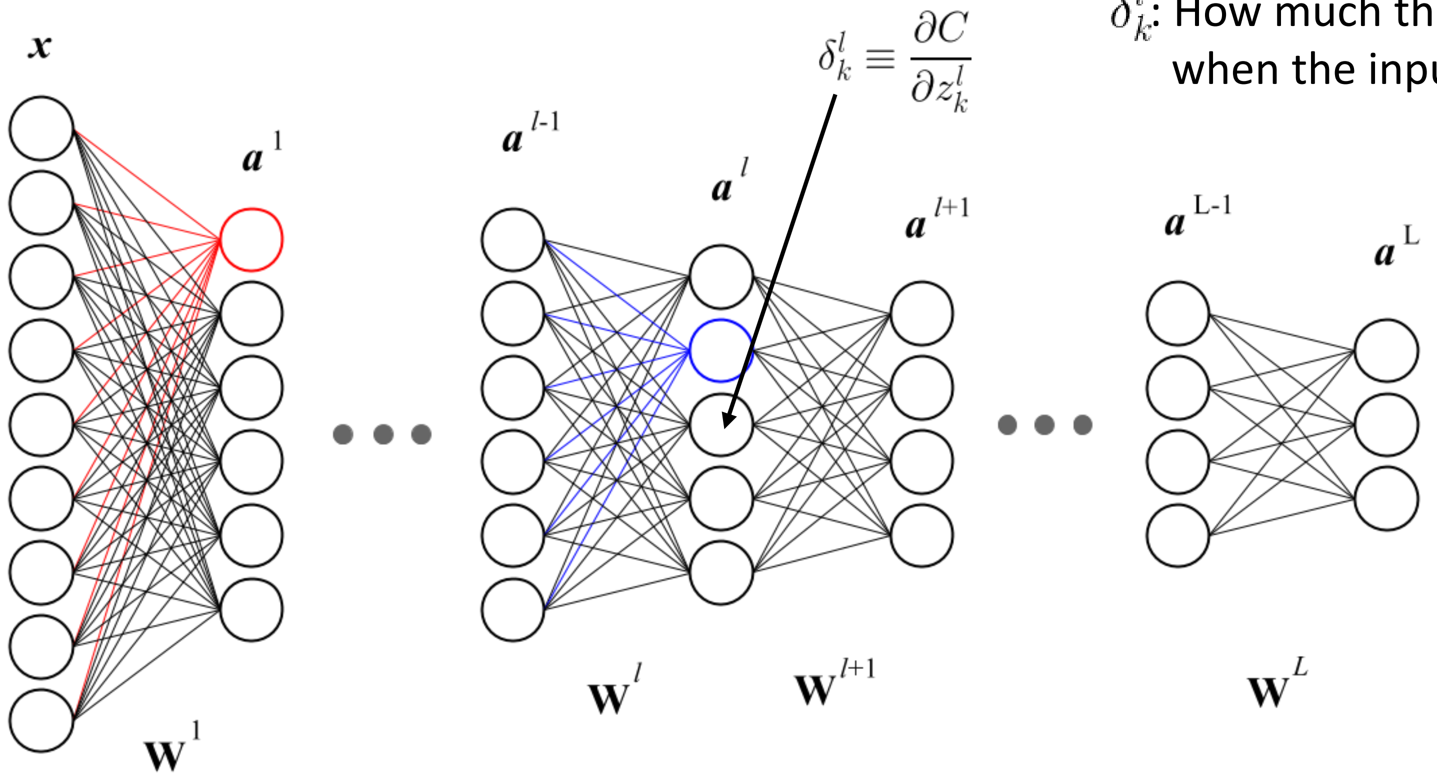
$$\delta^l := (f^l)' \cdot (W^{l+1})^T \cdots \cdot (W^{L-1})^T \cdot (f^{L-1})' \cdot (W^L)^T \cdot (f^L)' \cdot \nabla_{a^L} C$$

Training

- Backpropagation

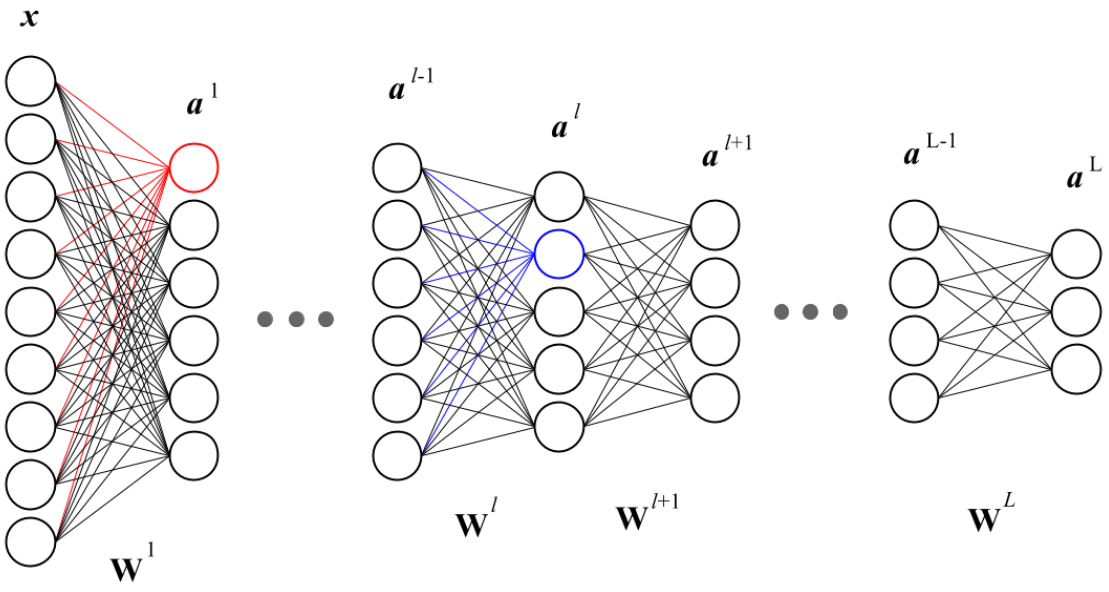
Define *delta* at each neuron

δ_k^l: How much the cost (loss) function changes when the input to the k-th neuron at layer l changes.



Training

- Backpropagation



Derivative of the loss function C w.r.t. input **x**

$$\nabla_x C = (W^1)^T \cdot (f^1)' \cdot \dots \cdot (W^{L-1})^T \cdot (f^{L-1})' \cdot (W^L)^T \cdot (f^L)' \cdot \nabla_{a^L} C$$

Define delta (error at layer *l*)

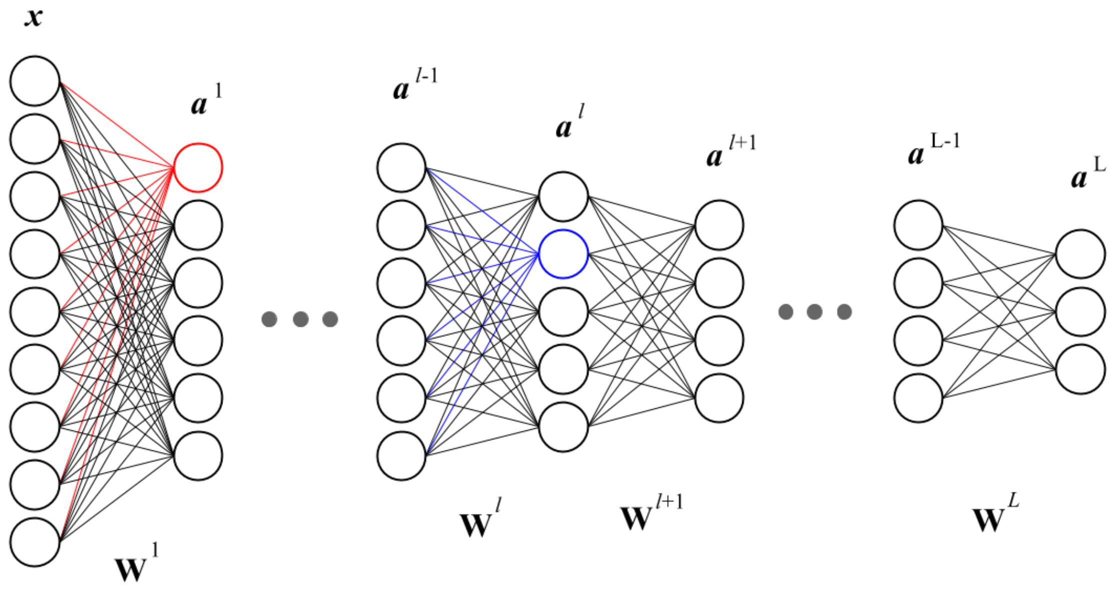
$$\delta^l := (f^l)' \cdot (W^{l+1})^T \cdot \dots \cdot (W^{L-1})^T \cdot (f^{L-1})' \cdot (W^L)^T \cdot (f^L)' \cdot \nabla_{a^L} C$$

Deltas at each layer

$$\begin{aligned}
\delta^1 &= (f^1)' \cdot (W^2)^T \cdot (f^2)' \cdot \dots \cdot (W^{L-1})^T \cdot (f^{L-1})' \cdot (W^L)^T \cdot (f^L)' \cdot \nabla_{a^L} C \\
\delta^2 &= (f^2)' \cdot \dots \cdot (W^{L-1})^T \cdot (f^{L-1})' \cdot (W^L)^T \cdot (f^L)' \cdot \nabla_{a^L} C \\
&\vdots \\
\delta^{L-1} &= (f^{L-1})' \cdot (W^L)^T \cdot (f^L)' \cdot \nabla_{a^L} C \\
\delta^L &= (f^L)' \cdot \nabla_{a^L} C,
\end{aligned}$$

Training

- Backpropagation



Derivative of the loss function C w.r.t. input **x**

$$\nabla_x C = (W^1)^T \cdot (f^1)' \cdot \cdots \cdot (W^{L-1})^T \cdot (f^{L-1})' \cdot (W^L)^T \cdot (f^L)' \cdot \nabla_{a^L} C$$

Define delta (error at layer l)

$$\delta^l := (f^l)' \cdot (W^{l+1})^T \cdot \cdots \cdot (W^{L-1})^T \cdot (f^{L-1})' \cdot (W^L)^T \cdot (f^L)' \cdot \nabla_{a^L} C$$

Deltas at each layer

$$\begin{aligned}
\delta^1 &= (f^1)' \cdot (W^2)^T \cdot (f^2)' \cdot \cdots \cdot (W^{L-1})^T \cdot (f^{L-1})' \cdot (W^L)^T \cdot (f^L)' \cdot \nabla_{a^L} C \\
\delta^2 &= (f^2)' \cdot \cdots \cdot (W^{L-1})^T \cdot (f^{L-1})' \cdot (W^L)^T \cdot (f^L)' \cdot \nabla_{a^L} C \\
&\vdots \\
\delta^{L-1} &= (f^{L-1})' \cdot (W^L)^T \cdot (f^L)' \cdot \nabla_{a^L} C \\
\delta^L &= (f^L)' \cdot \nabla_{a^L} C,
\end{aligned}$$

Delta can be recursively defined!

$$\delta^{l-1} := (f^{l-1})' \cdot (W^l)^T \cdot \delta^l$$

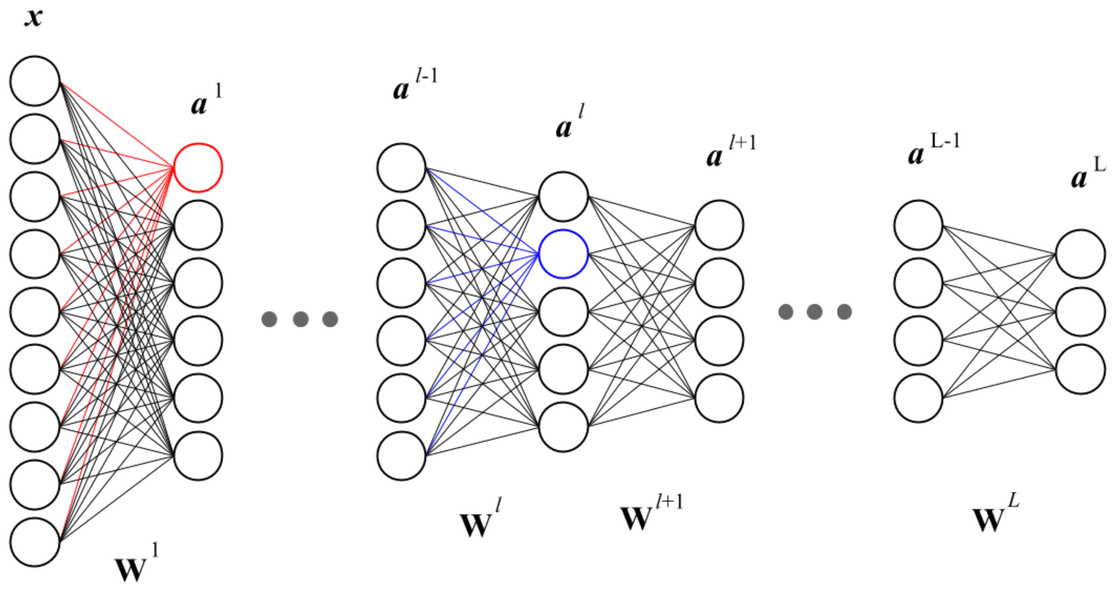
Derivative of C w.r.t weights can be defined via delta

$$\nabla_{W^l} C = \delta^l (a^{l-1})^T$$

Needed for ➔ $w_1^{new} = w_1^{old} - \eta \cdot \frac{\partial NLL}{\partial w_1}$

Training

- Backpropagation



Derivative of the loss function C w.r.t. input **x**

$$\nabla_x C = (W^1)^T \cdot (f^1)' \cdots (W^{L-1})^T \cdot (f^{L-1})' \cdot (W^L)^T \cdot (f^L)' \cdot \nabla_{a^L} C$$

Define delta (error at layer l)

$$\delta^l := (f^l)' \cdot (W^{l+1})^T \cdots (W^{L-1})^T \cdot (f^{L-1})' \cdot (W^L)^T \cdot (f^L)' \cdot \nabla_{a^L} C$$

Deltas at each layer

$$\begin{aligned}
\delta^1 &= (f^1)' \cdot (W^2)^T \cdot (f^2)' \cdots (W^{L-1})^T \cdot (f^{L-1})' \cdot (W^L)^T \cdot (f^L)' \cdot \nabla_{a^L} C \\
\delta^2 &= (f^2)' \cdots (W^{L-1})^T \cdot (f^{L-1})' \cdot (W^L)^T \cdot (f^L)' \cdot \nabla_{a^L} C \\
&\vdots \\
\delta^{L-1} &= (f^{L-1})' \cdot (W^L)^T \cdot (f^L)' \cdot \nabla_{a^L} C \\
\delta^L &= (f^L)' \cdot \nabla_{a^L} C,
\end{aligned}$$

Delta can be recursively defined!

$$\delta^{l-1} := (f^{l-1})' \cdot (W^l)^T \cdot \delta^l$$

Backprop is faster than forward-prop!
But need to store $(f^l)', a_k^l$ for every node in every layer.

Derivative of C w.r.t weights can be defined via delta

$$\nabla_{W^l} C = \delta^l (a^{l-1})^T$$

Needed for ➔ $w_1^{new} = w_1^{old} - \eta \cdot \frac{\partial NLL}{\partial w_1}$

Autograd (in PyTorch)

In Practice

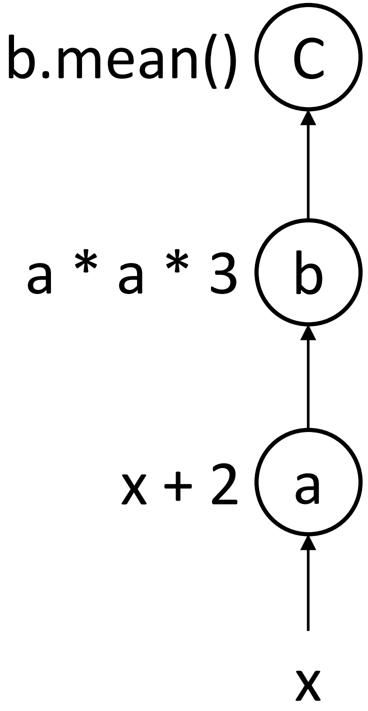
- Computer does backpropagation for you.
  - Autograd
  - Theano, TensorFlow, PyTorch
- A neural network model is represented as a Directed Acyclic Graph
  - Each node contains a mathematical operation.
  - Each node contains the derivative of the math operation.
  - The error signal is propagated from the output nodes to the input nodes.

Math Program as DAG

- x: [[1, 1], [1, 1]]
- a = x + 2
- b = a * a * 3
- c = b.mean()

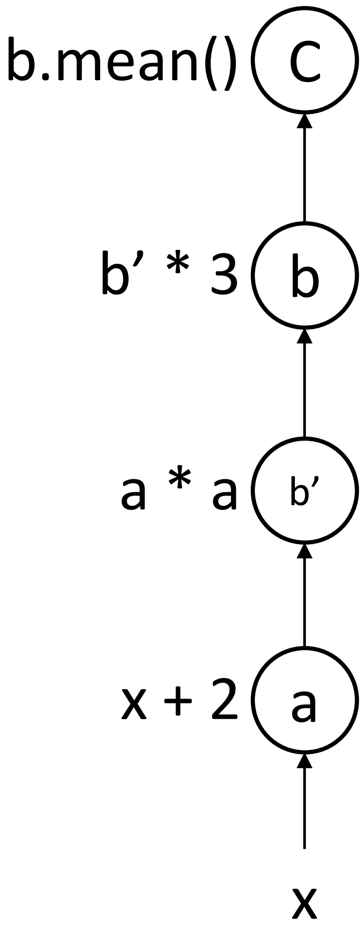
Math Program as DAG

- x: [[1, 1], [1, 1]]
- a = x + 2
- b = a * a * 3
- c = b.mean()



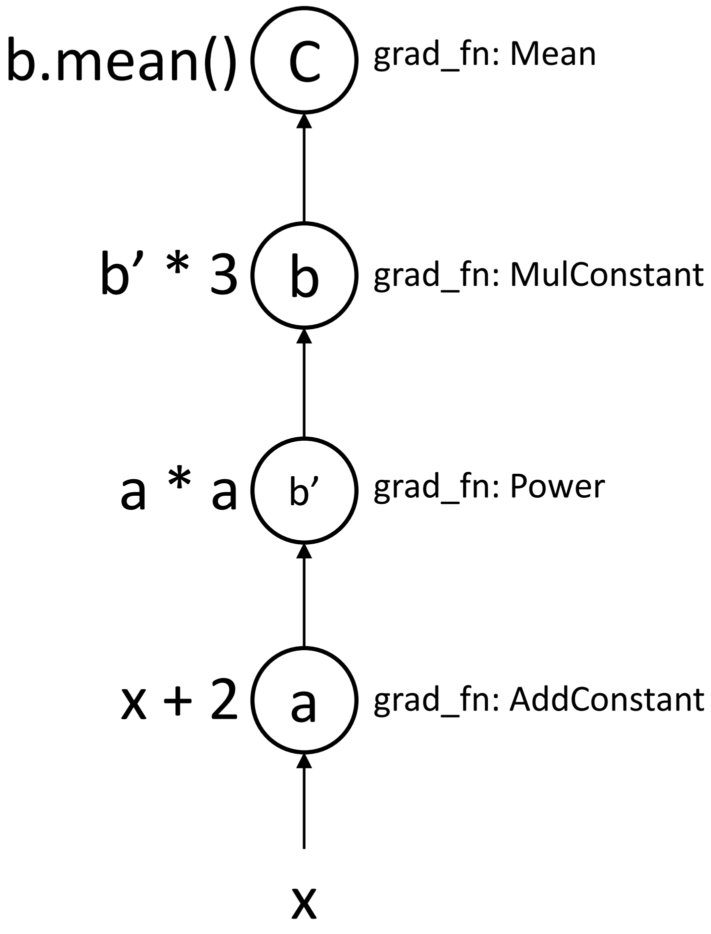
Math Program as DAG

- x: [[1, 1], [1, 1]]
- a = x + 2
- b’ = a * a
- b = b’ * 3
- c = b.mean()



Math Program as DAG

- x: [[1, 1], [1, 1]]
- a = x + 2
- b’ = a * a
- b = b’ * 3
- c = b.mean()

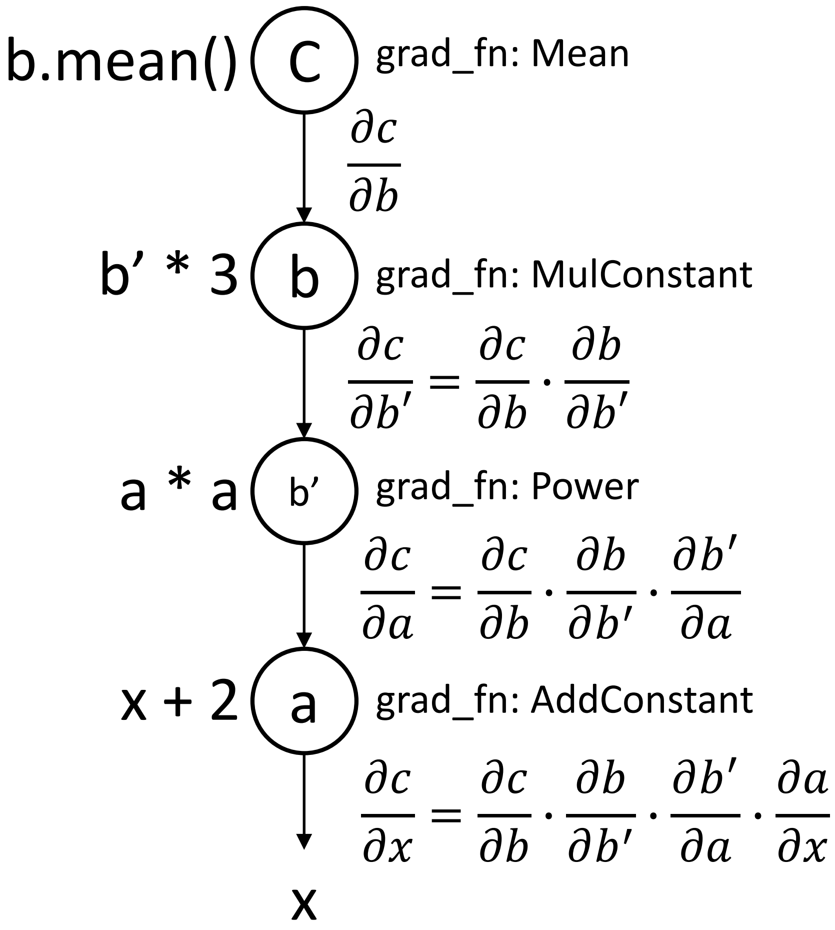


Each grad_fn has

- forward
  - Compute forward propagation
  - Save input & misc. for backward
- backward
  - Given δ^l, calculate δ^{l-1}
  - Calculate $\nabla_{W^l} C = \delta^l (a^{l-1})^T$, if necessary

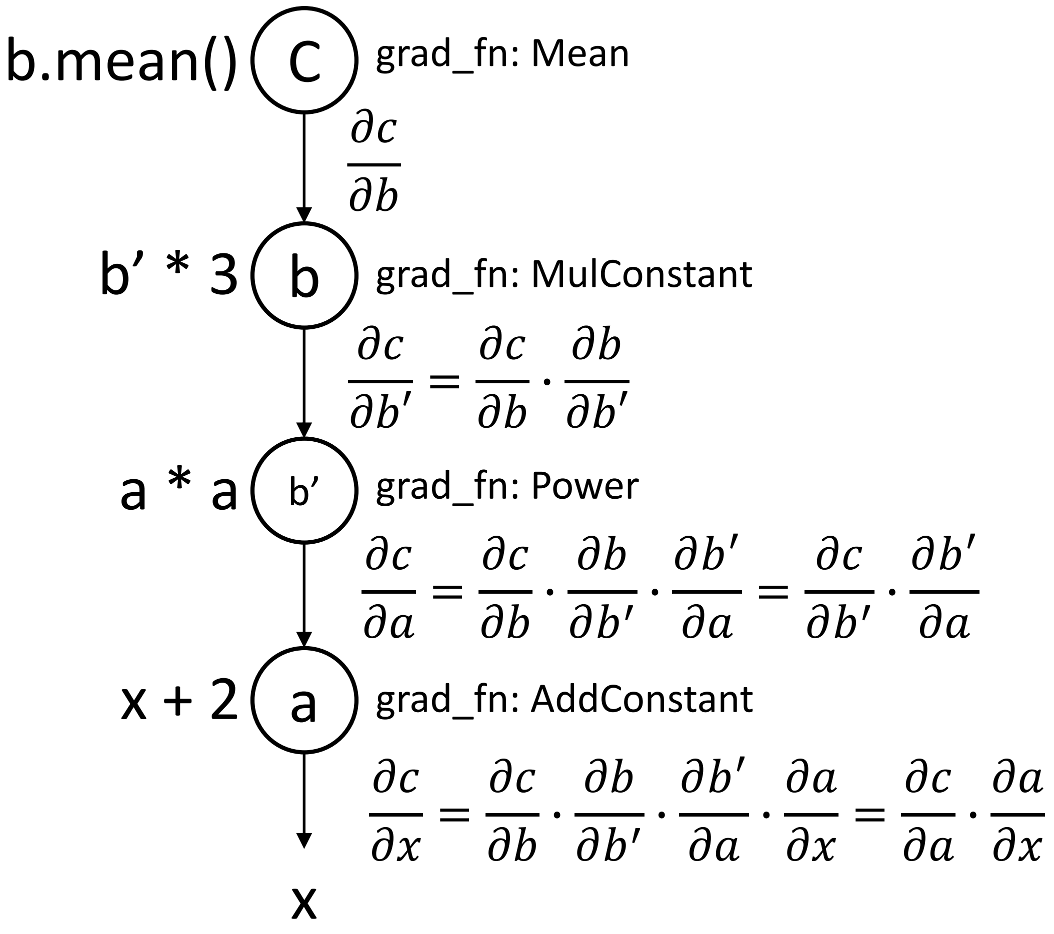
Math Program as DAG

- x: [[1, 1], [1, 1]]
- a = x + 2
- b’ = a * a
- b = b’ * 3
- c = b.mean()



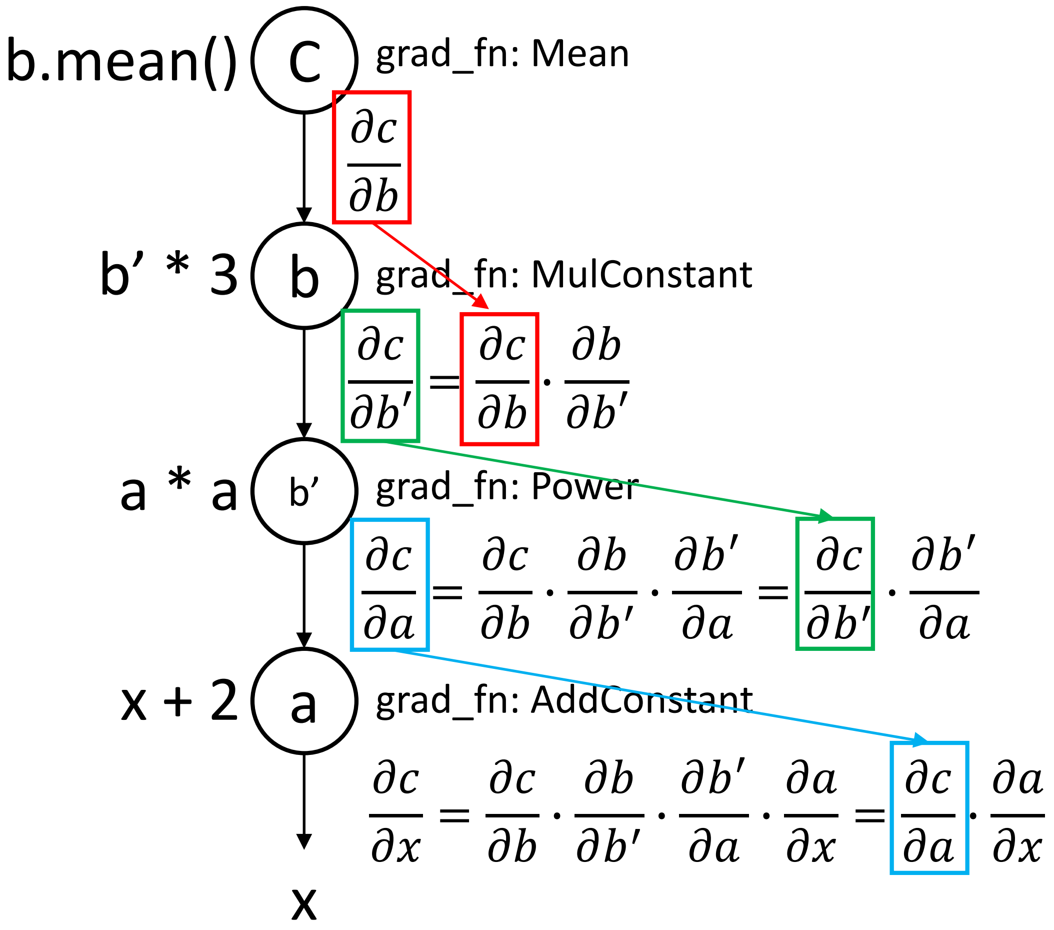
Math Program as DAG

- x: [[1, 1], [1, 1]]
- a = x + 2
- b’ = a * a
- b = b’ * 3
- c = b.mean()



Math Program as DAG

- x: [[1, 1], [1, 1]]
- a = x + 2
- b’ = a * a
- b = b’ * 3
- c = b.mean()



AI504: Programming for Artificial Intelligence

Week 3: Neural Nets & Backpropagation

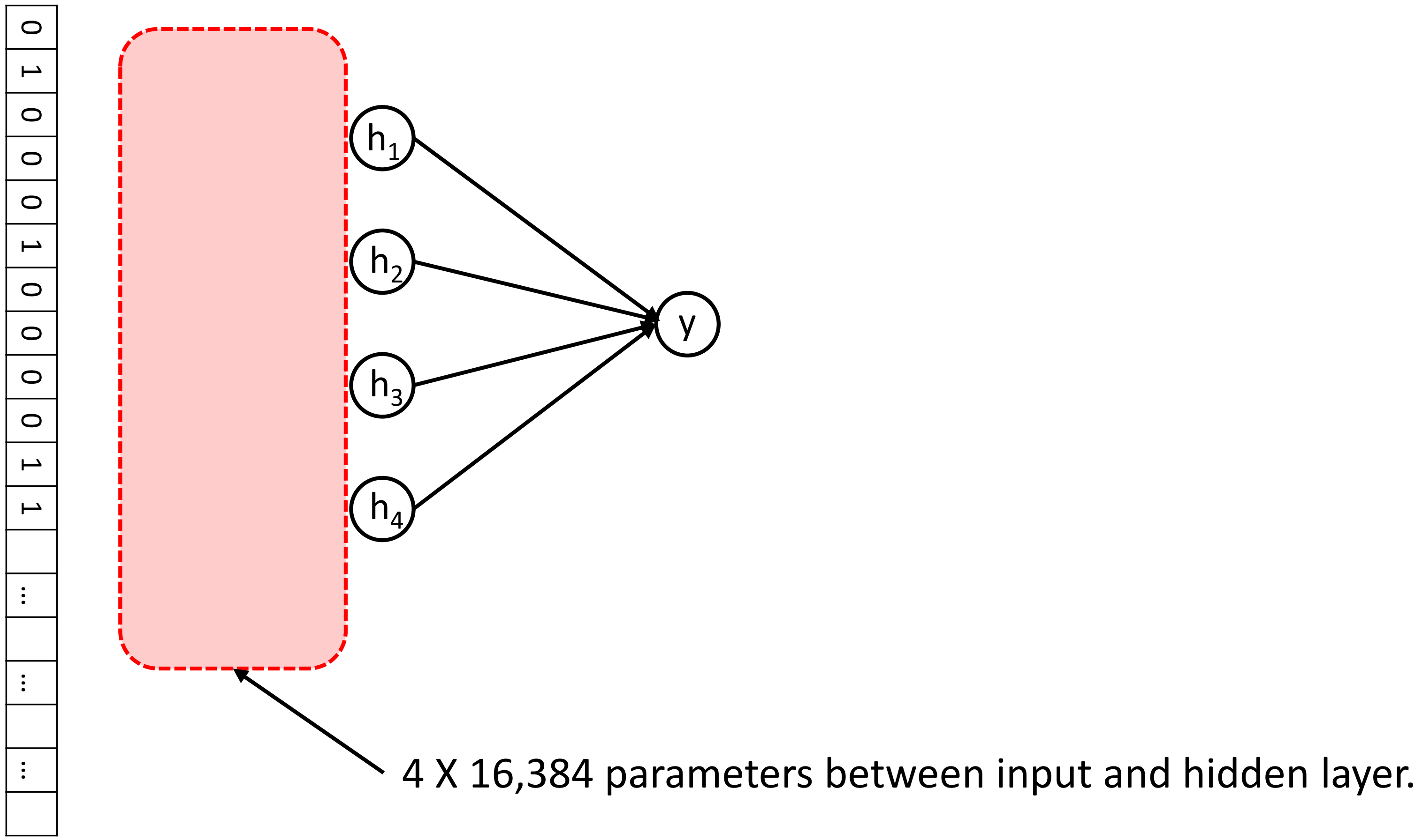
Edward Choi

Grad School of AI

edwardchoi@kaist.ac.kr

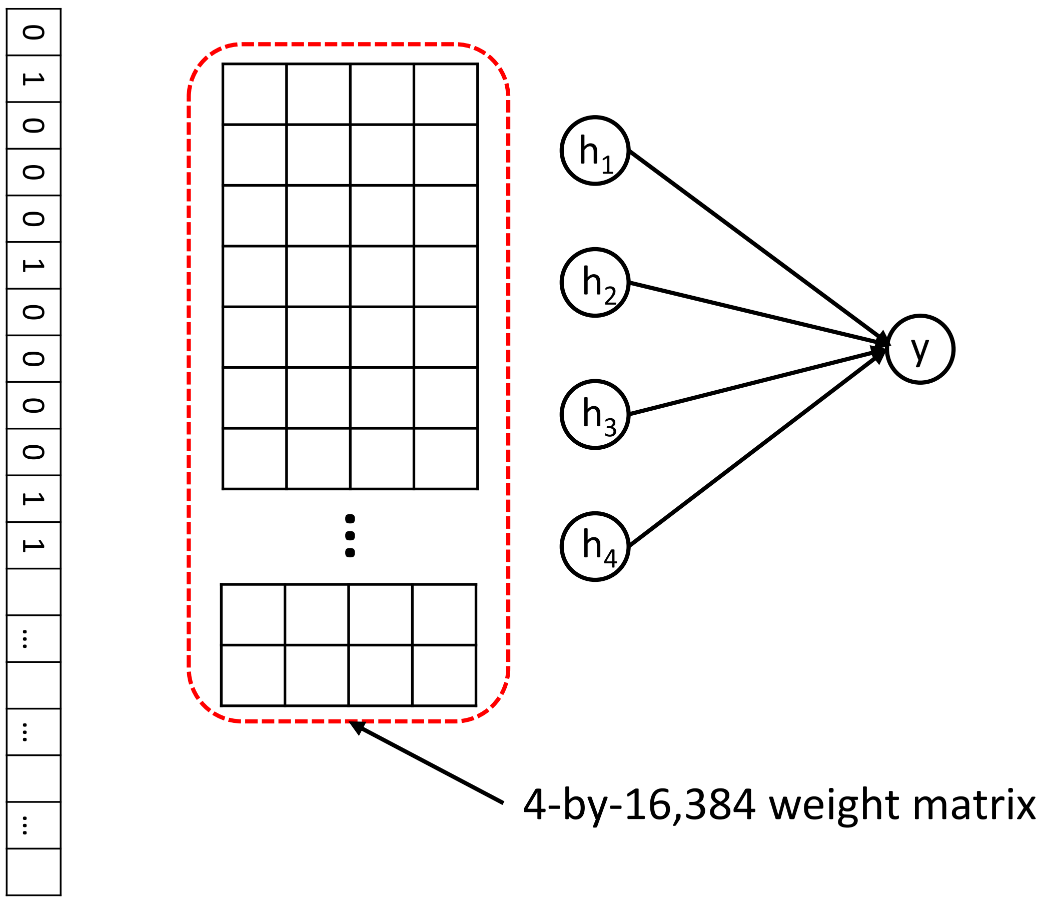
Neural Networks

- There is a matrix between layers



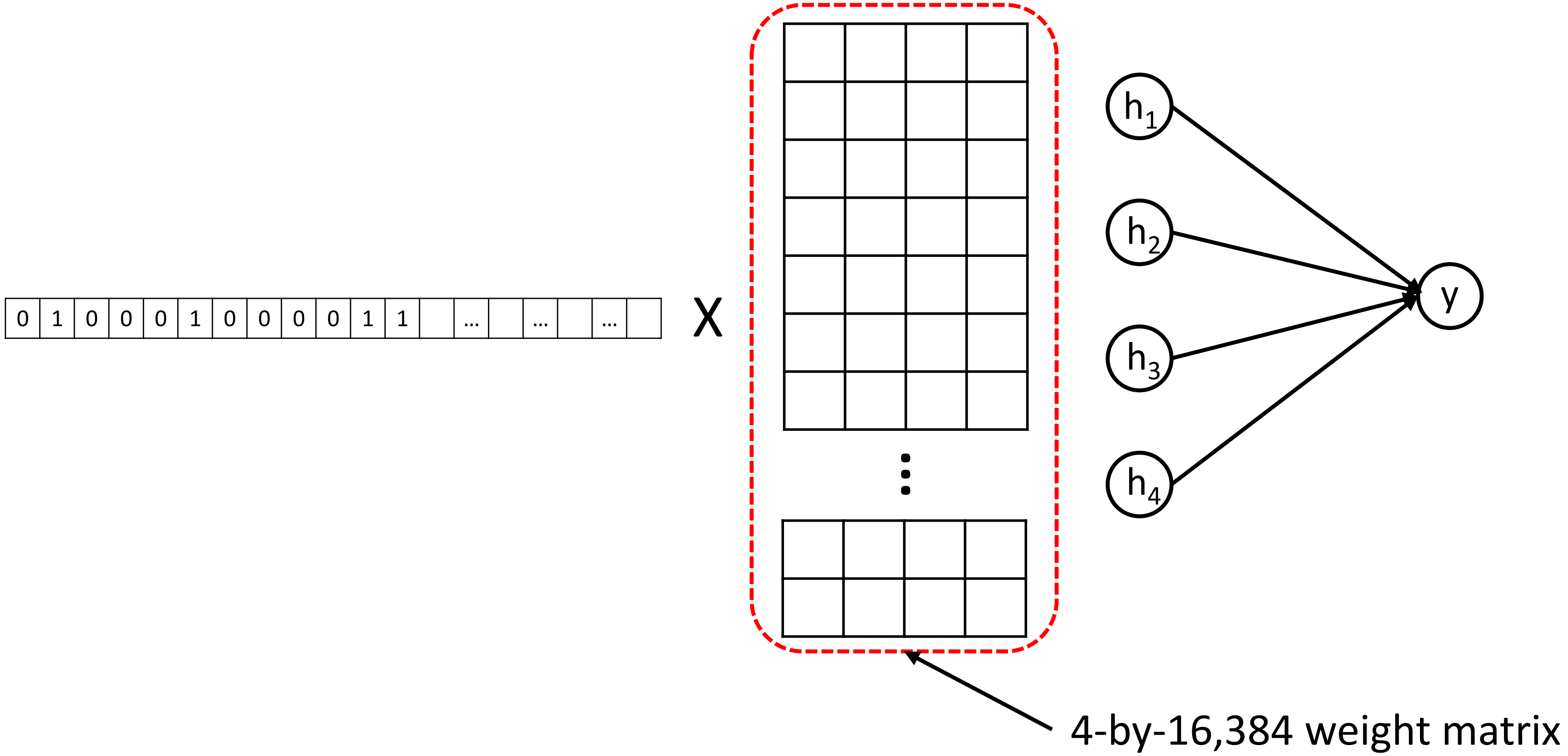
Neural Networks

- Weight matrix



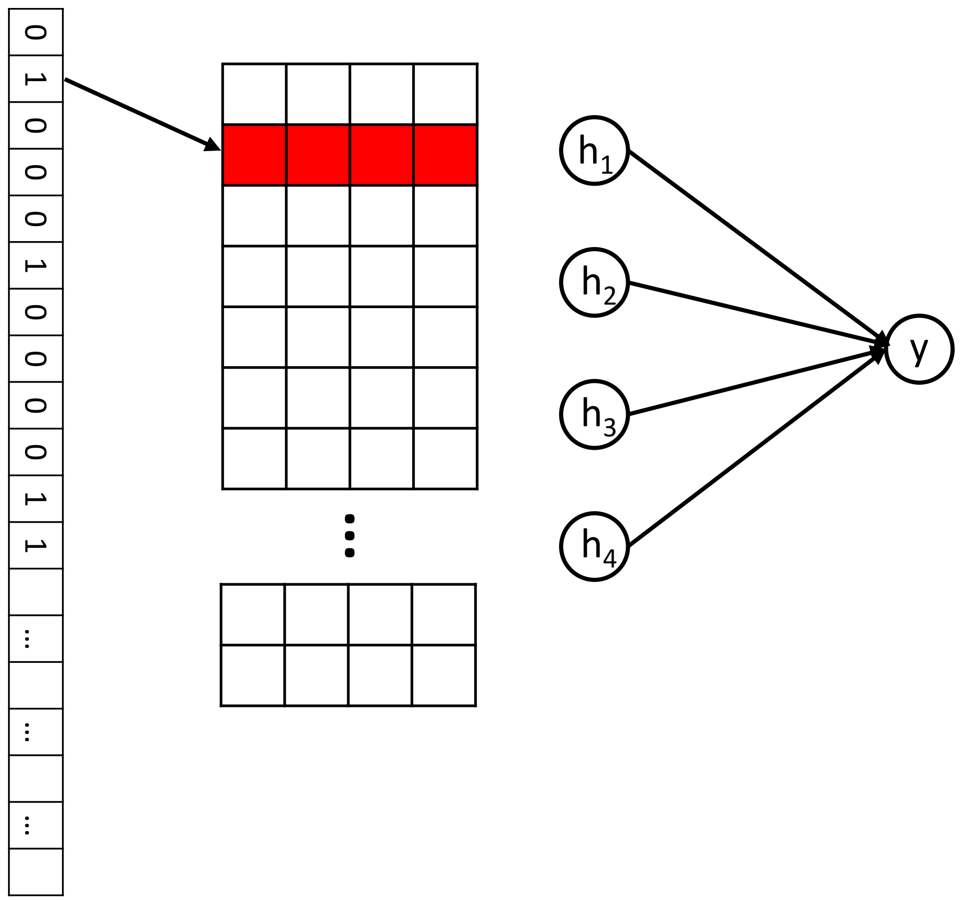
Neural Networks

- Weight matrix



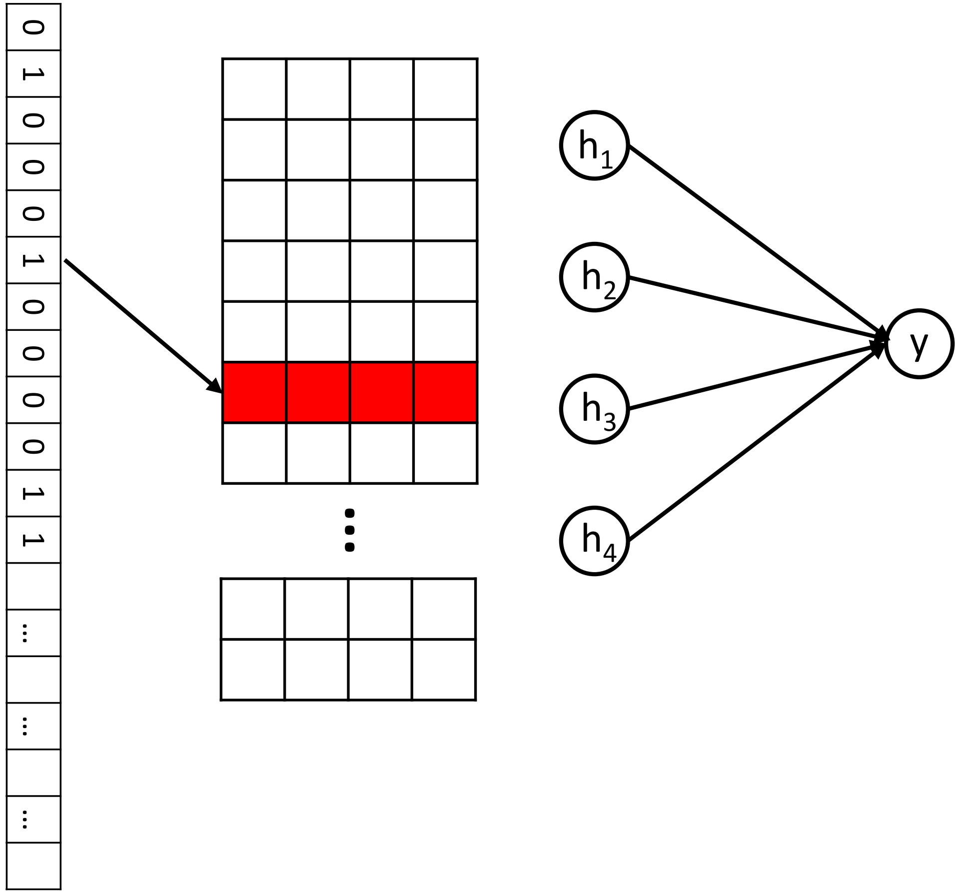
Neural Networks

- A bit of linear algebra



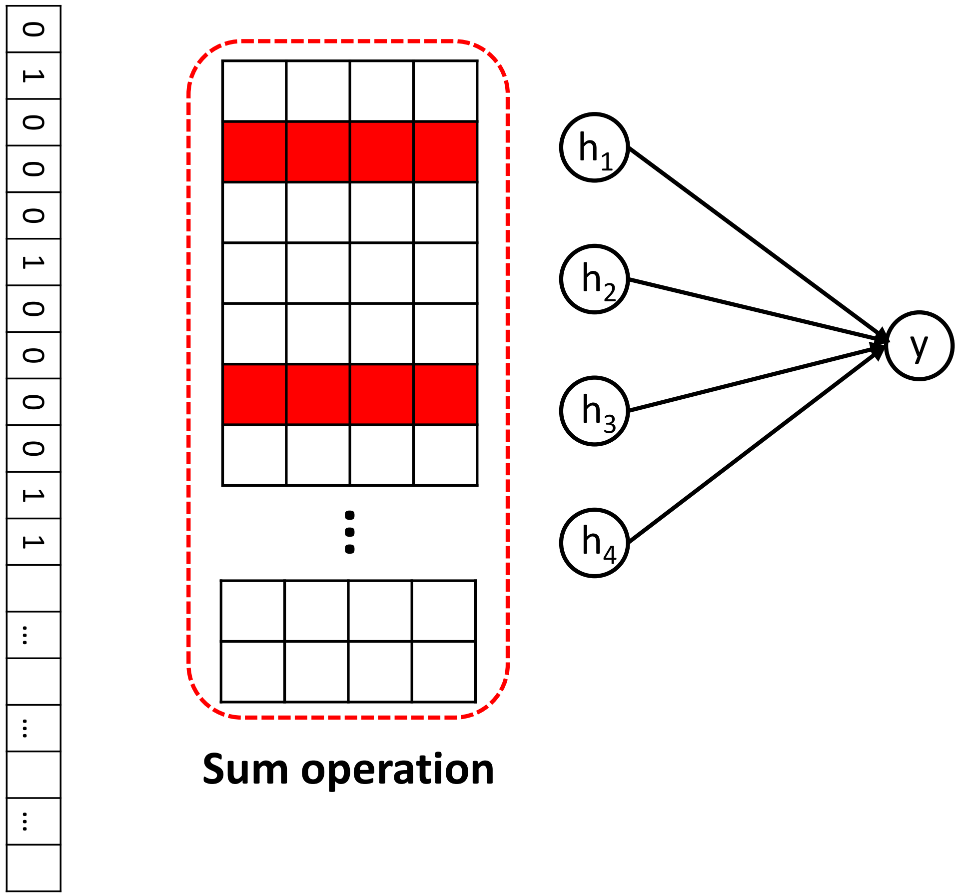
Neural Networks

- A bit of linear algebra



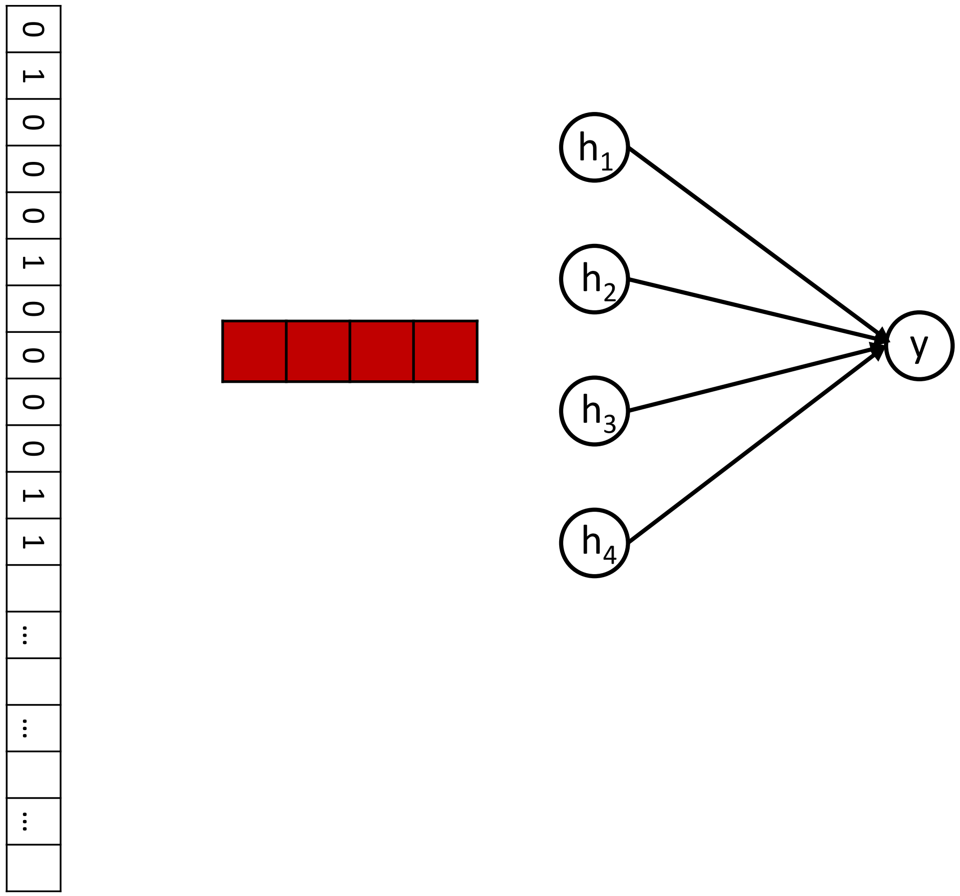
Neural Networks

- A bit of linear algebra



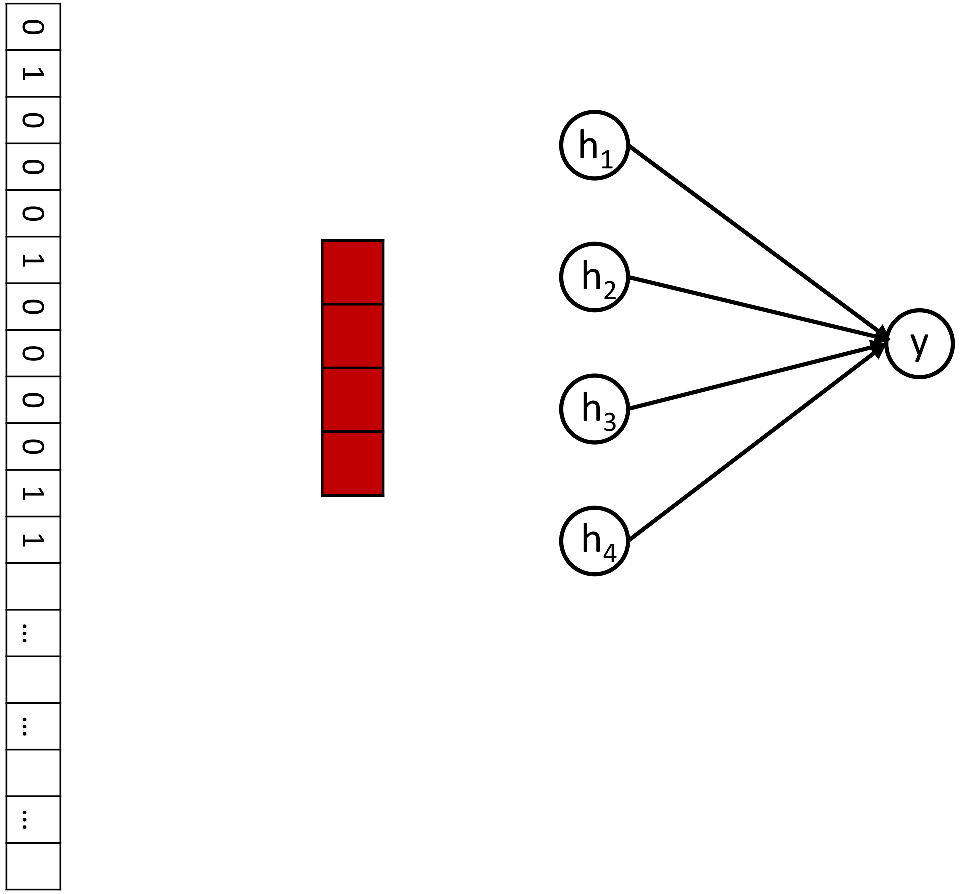
Neural Networks

- A bit of linear algebra



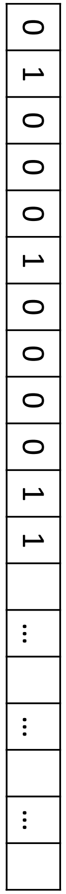
Representation

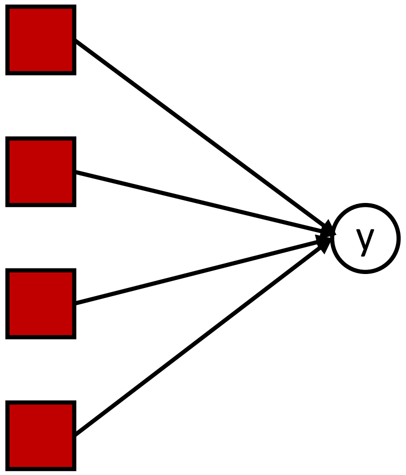
- Hidden representation (i.e. latent representation)



Representation

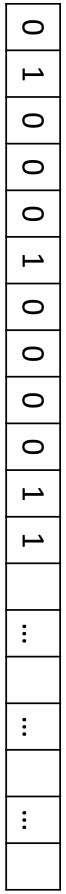
- Hidden representation (i.e. latent representation)

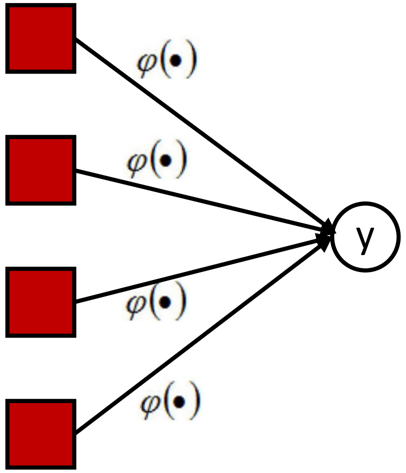




Representation

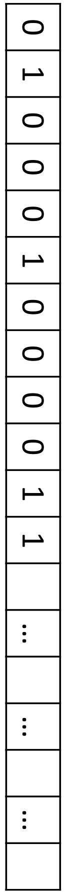
- Hidden representation (i.e. latent representation)

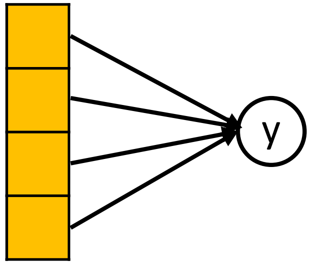




Representation

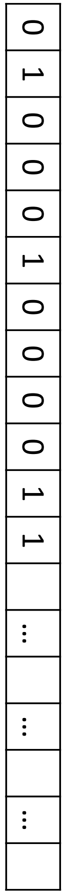
- Hidden representation (i.e. latent representation)

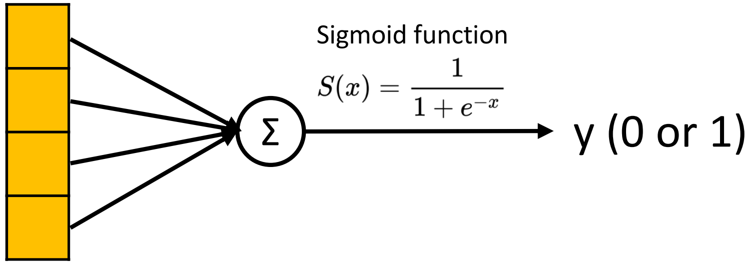




Representation

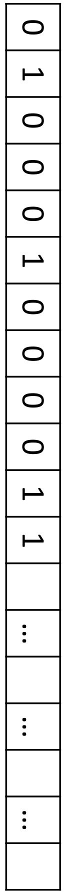
- Hidden representation (i.e. latent representation)

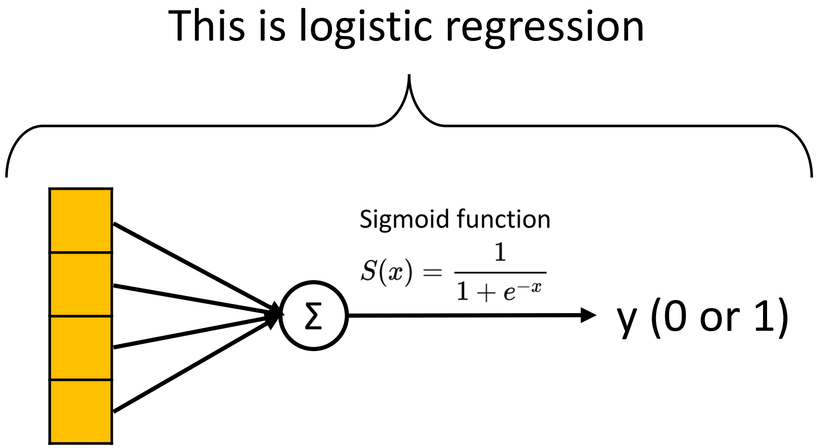




Representation

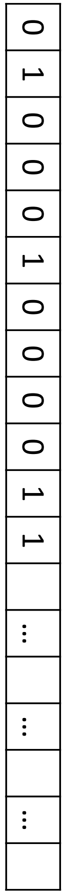
- Hidden representation (i.e. latent representation)



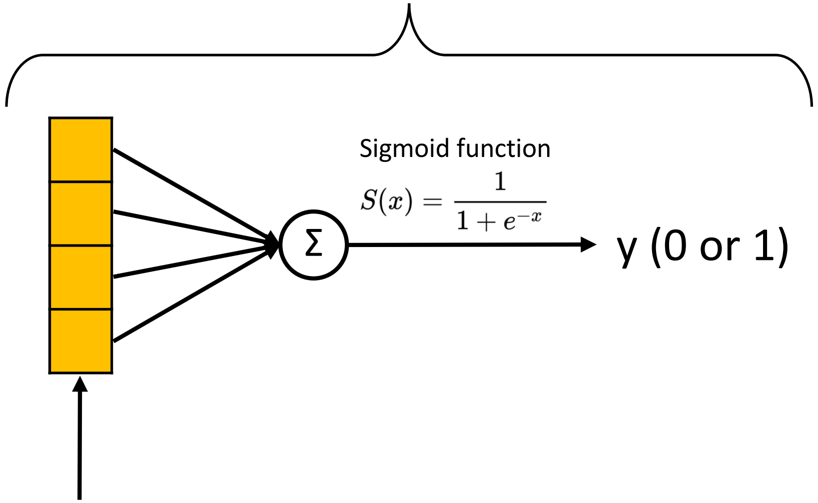


Representation

- Hidden representation (i.e. latent representation)



This is logistic regression



HIdden (i.e. Latent) image representation

Representation

- Logistic regression VS feedforward neural network

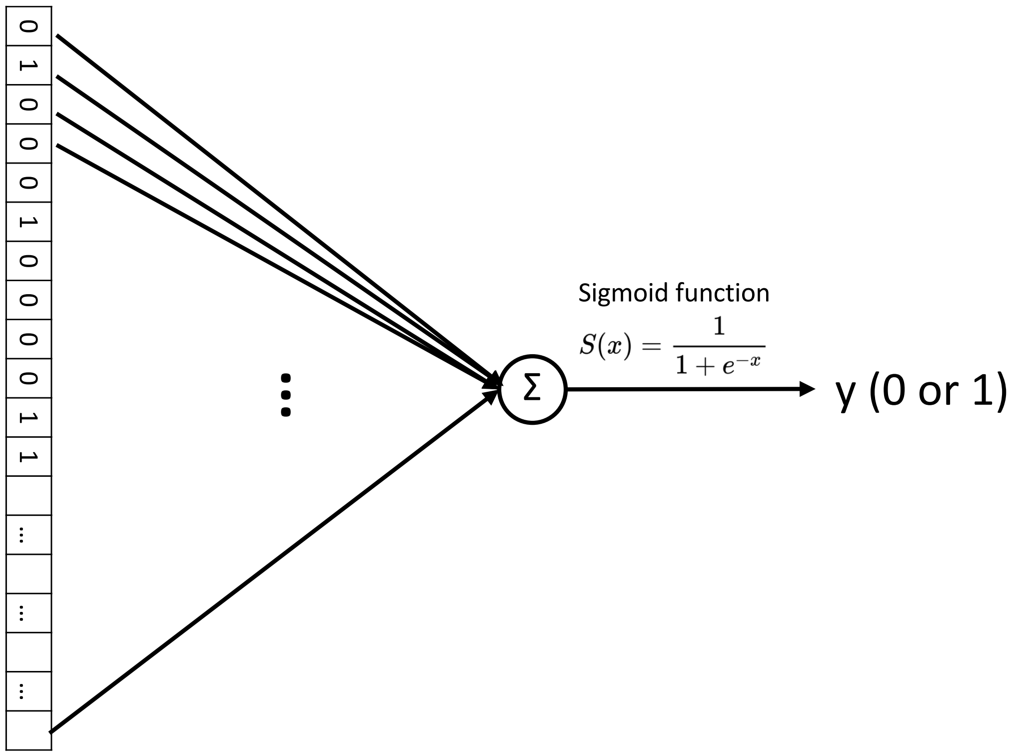


Image representation

Neural Networks

- Logistic regression VS feedforward neural network