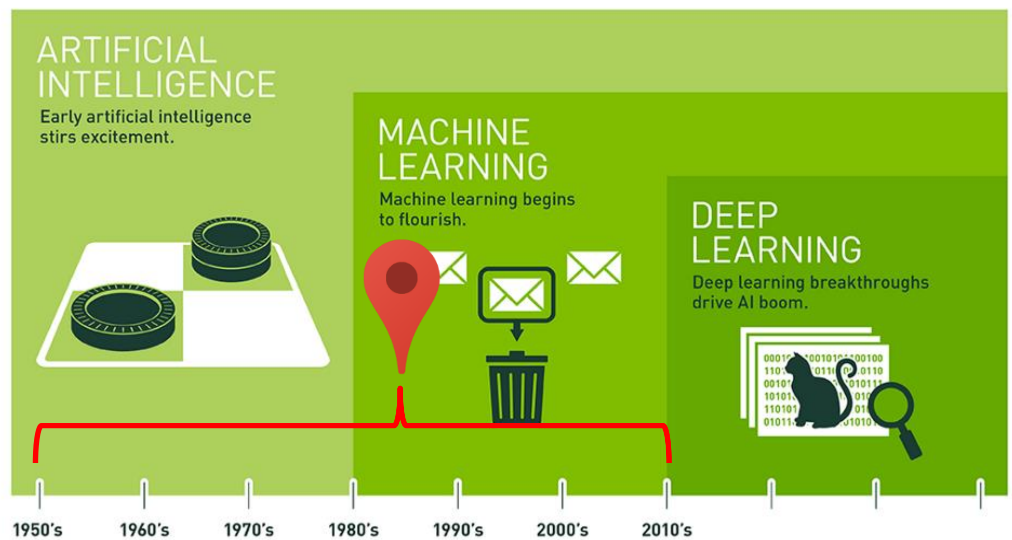
Artificial Intelligence: Introduction to Neural Networks

Perceptron, Backpropagation

Today

- Neural Networks

  - Perceptrons
  - Backpropagation



https://www.linkedin.com/pulse/goedels-incompleteness-theorem-emergence-ai-eberhard-schoeneburg/

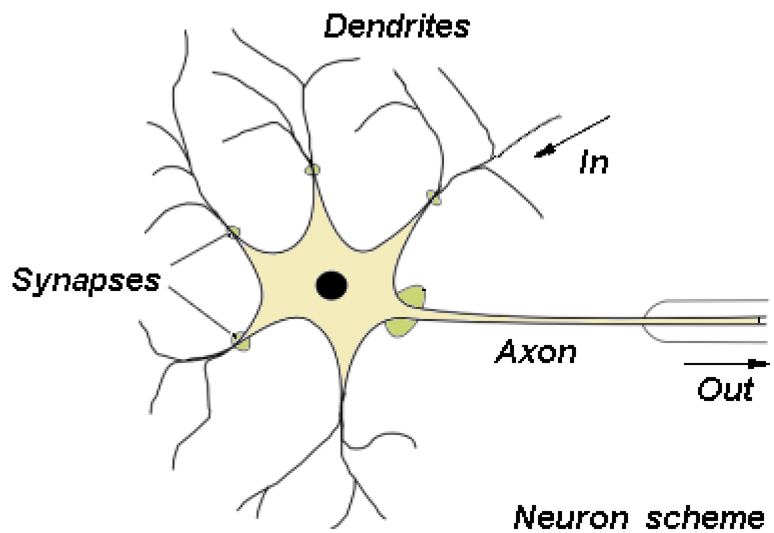
Neural Networks

- Radically different approach to reasoning and learning
- Inspired by biology
  - the neurons in the human brain
- Set of many simple processing units (neurons) connected together
- Behavior of each neuron is very simple
  - but a collection of neurons can have sophisticated behavior and can be used for complex tasks
- In a neural network, the behavior depends on weights on the connection between the neurons
- The weights will be learned given training data

Biological Neurons

- Human brain =
  - 100 billion neurons
  - each neuron may be connected to 10,000 other neurons
  - passing signals to each other via 1,000 trillion **synapses**

- A neuron is made of:
  - Dendrites: filaments that provide input to the neuron
  - Axon: sends an output signal
  - Synapses: connection with other neurons – releases neurotransmitters to other neurons



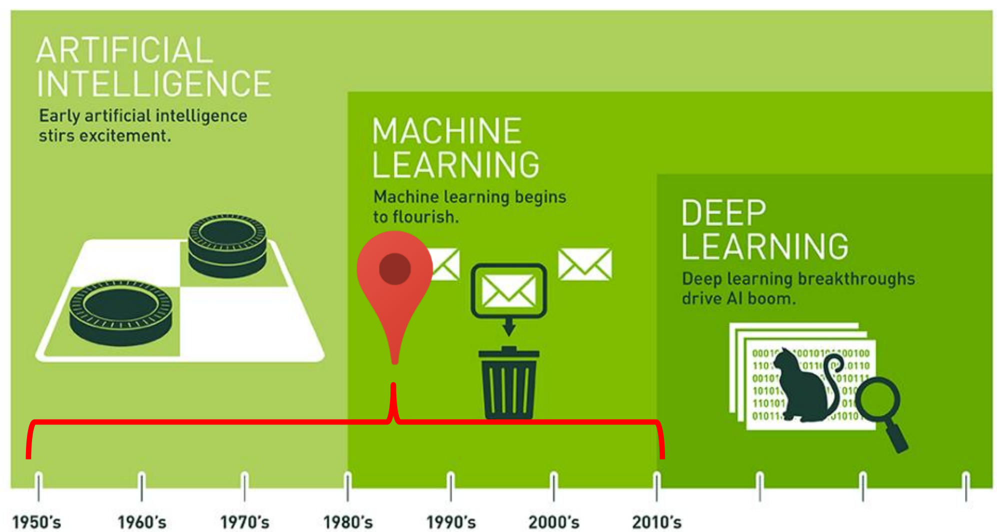
Source: http://www.human-memory.net/brain_neurons.html

Behavior of a Neuron

- A neuron receives inputs from its neighbors
- If enough inputs are received at the same time:
  - the neuron is activated
  - and fires an output to its neighbors
- Repeated firings across a synapse increases its sensitivity and the future likelihood of its firing
- If a particular stimulus repeatedly causes activity in a group of neurons, they become strongly associated

Today

- Neural Netwo
  - Perceptrons
  - Backpropagation



https://www.linkedin.com/pulse/goedels-incompleteness-theorem-emergence-ai-eberhard-schoeneburg/

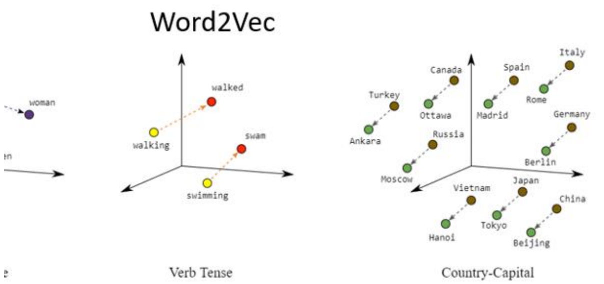
Feature Vector Representation

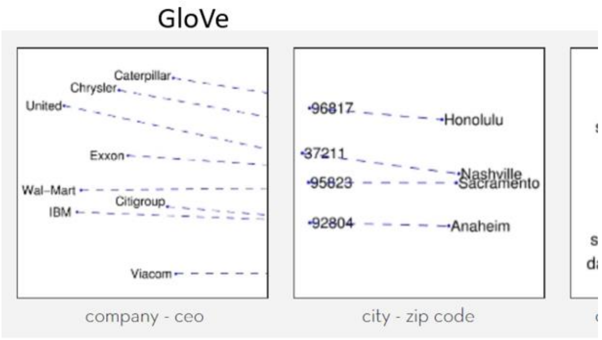
- Sources of Feature Vector x
  - Encoded image
  - Tabulated data
  - Embedded words
  - …

Tabular Data

columns = attributes for those observations

Player	Minutes	Points	Rebounds	Assists
A	41	20	6	5
B	30	29	7	6
C	22	7	7	2
D	26	3	3	9
E	20	19	8	0
F	9	6	14	14
G	14	22	8	3
I	22	36	0	9
J	34	8	1	3

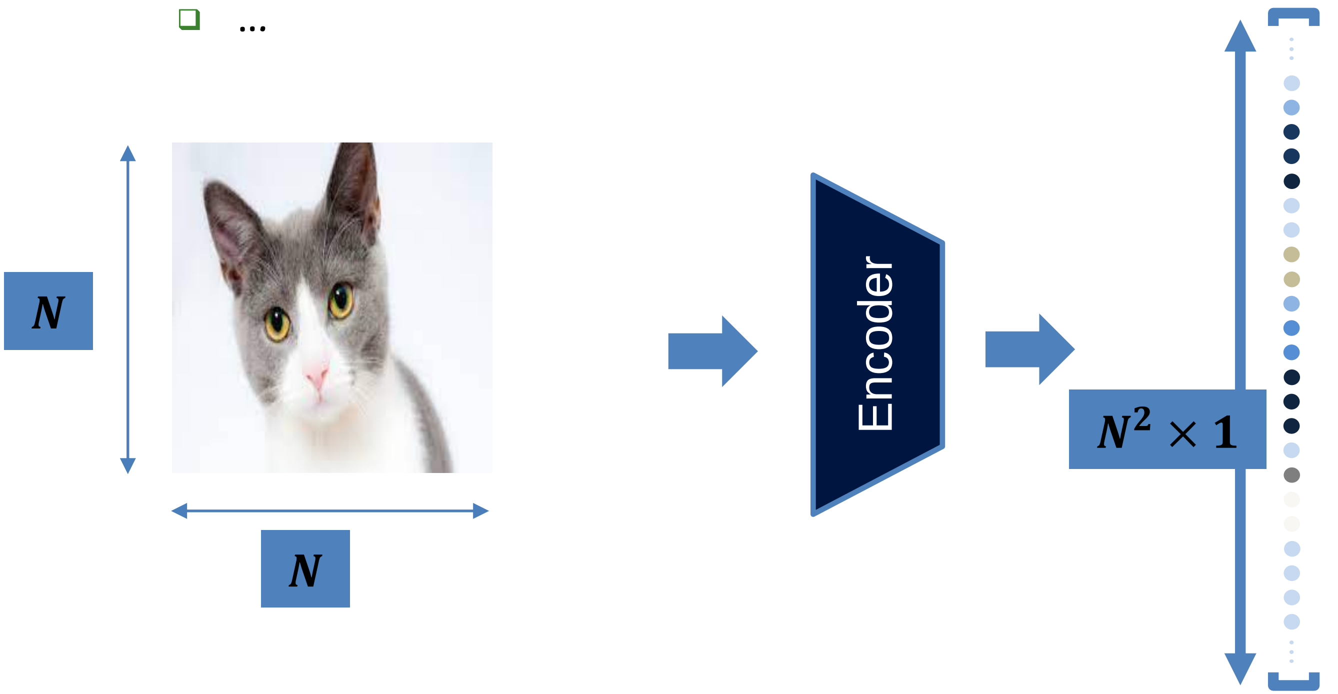




source: Luger (2005)

Feature Vector Representation

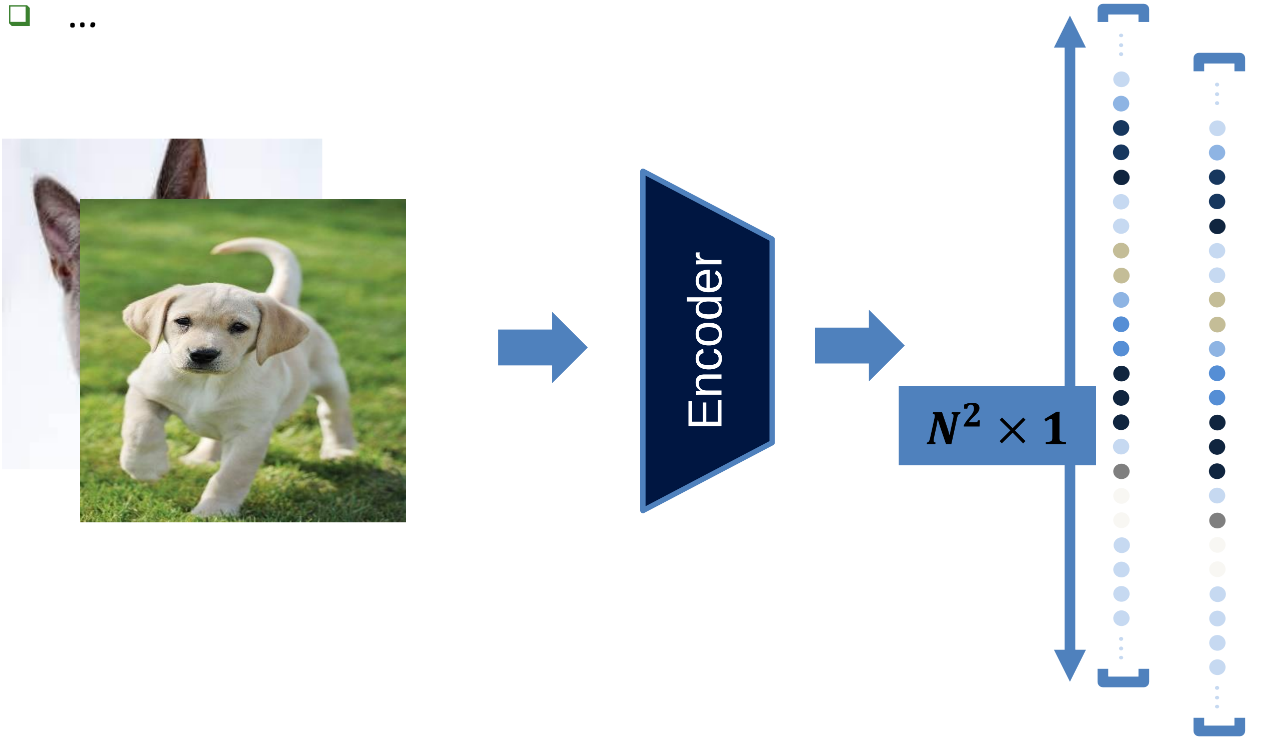
- Sources of Feature Vector x
  - Encoded image
  - Tabulated data
  - Embedded words
  - ...



source: Luger (2005)

Feature Vector Representation

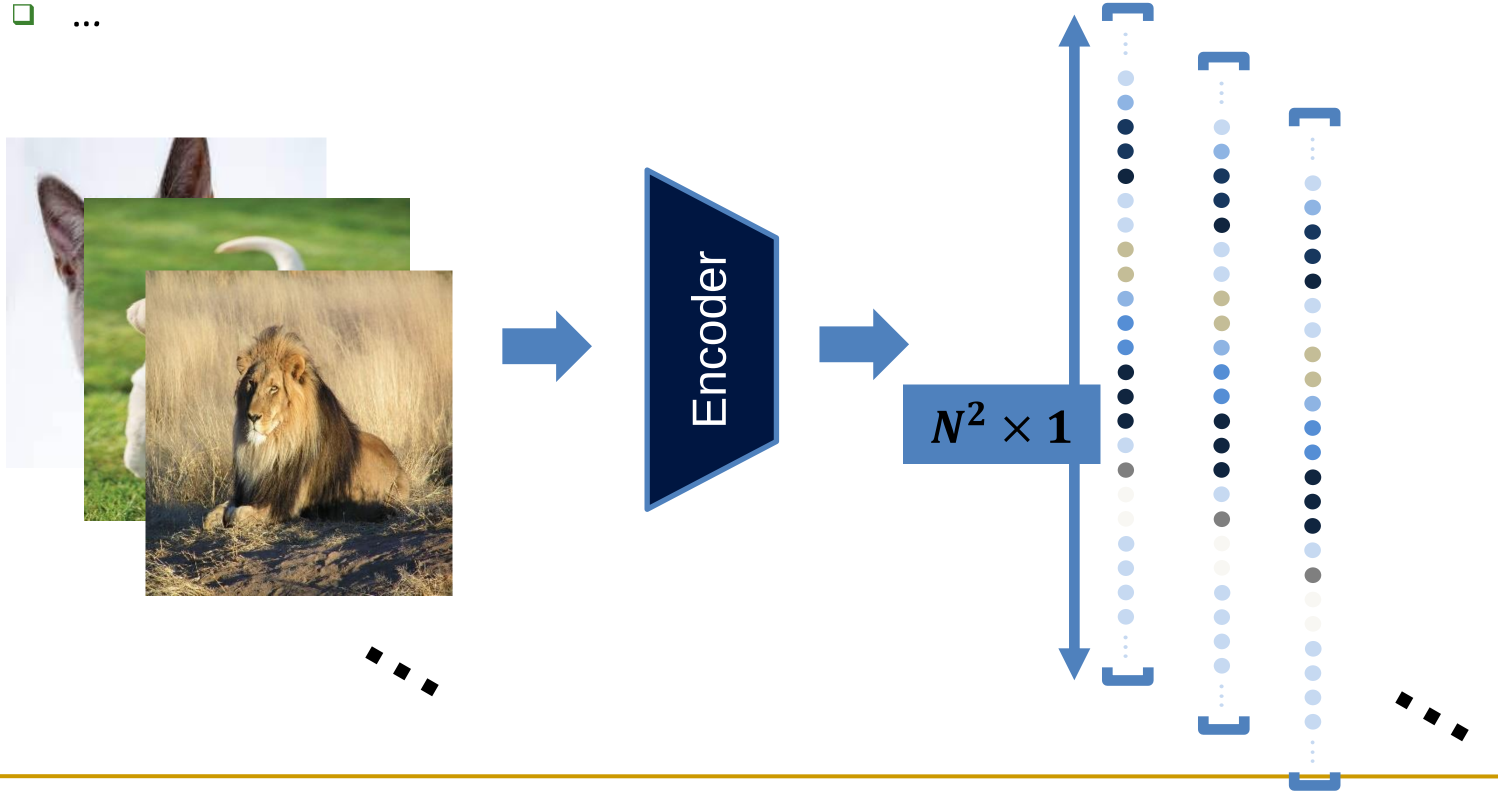
- Sources of Feature Vector x
  - Encoded image
  - Tabulated data
  - Embedded words
  - ...



source: Luger (2005)

Feature Vector Representation

- Sources of Feature Vector x
  - Encoded image
  - Tabulated data
  - Embedded words
  - …

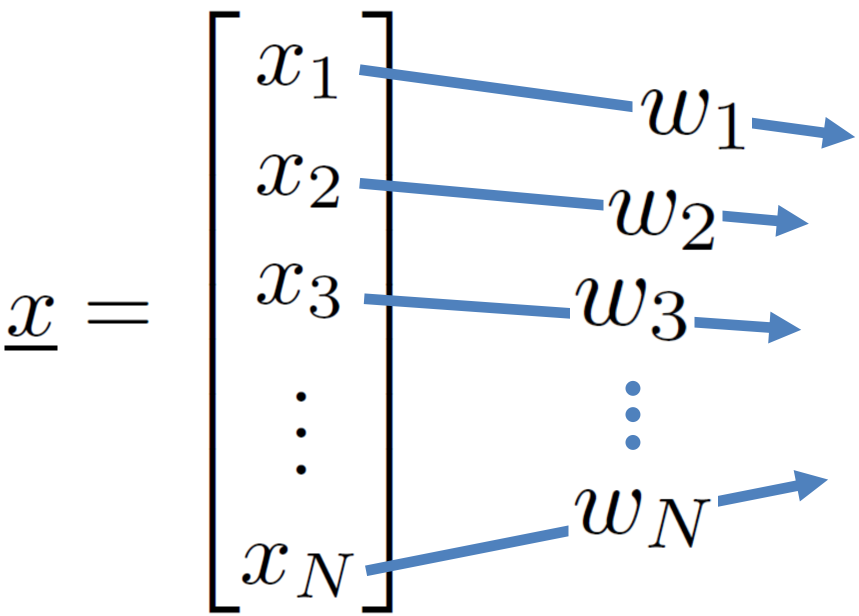


source: Luger (2005)

A Perceptron Network

- Goal: Map *Input* Feature Vector **x** into *Output* Feature Vector **y**

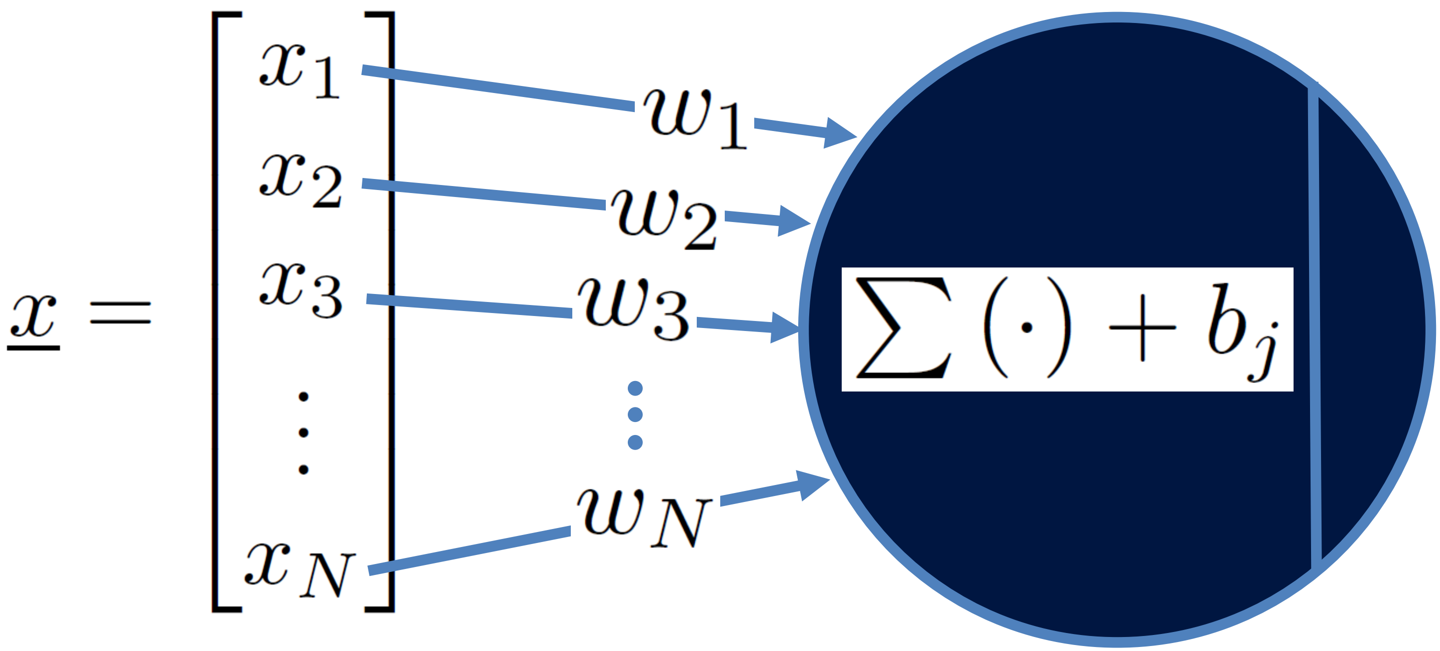
$$\boxed{\sum_{i=1}^{N} w_i x_i}$$



A Perceptron Network

- Goal: Map *Input* Feature Vector ***x*** into *Output* Feature Vector ***y***

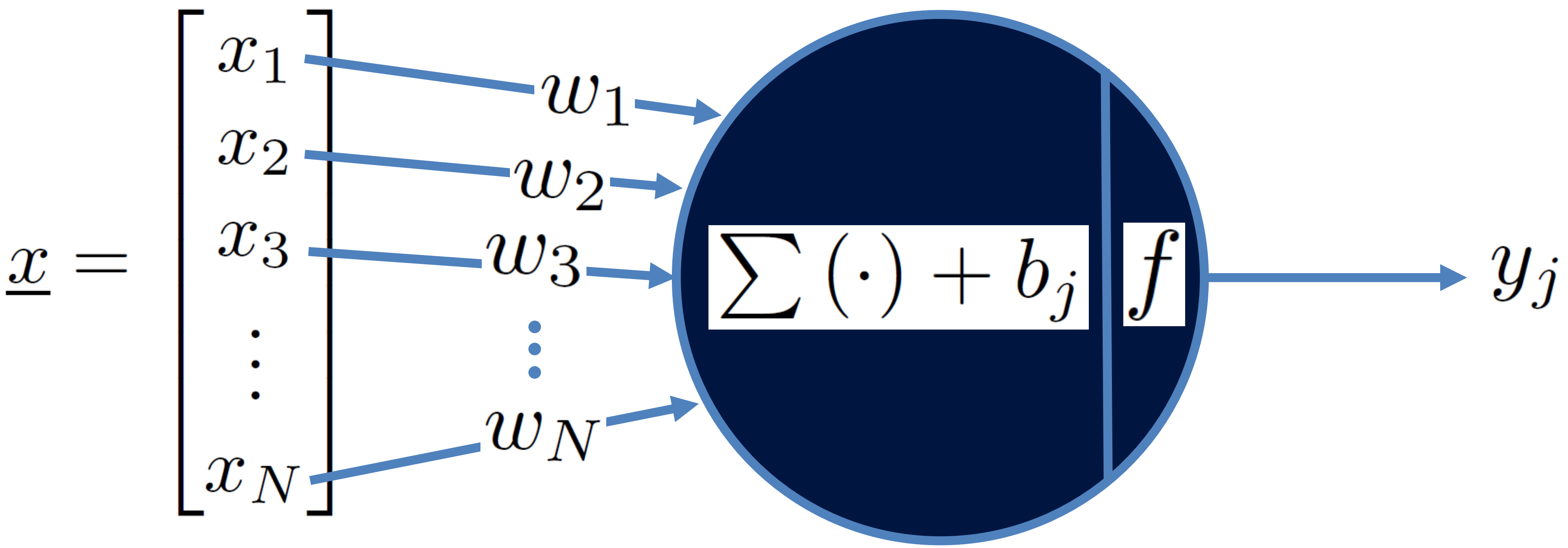
$$\boxed{\sum_{i=1}^{N} w_i x_i + b_j}$$



A Perceptron Network

- Goal: Map *Input* Feature Vector ***x*** into *Output* Feature Vector ***y***

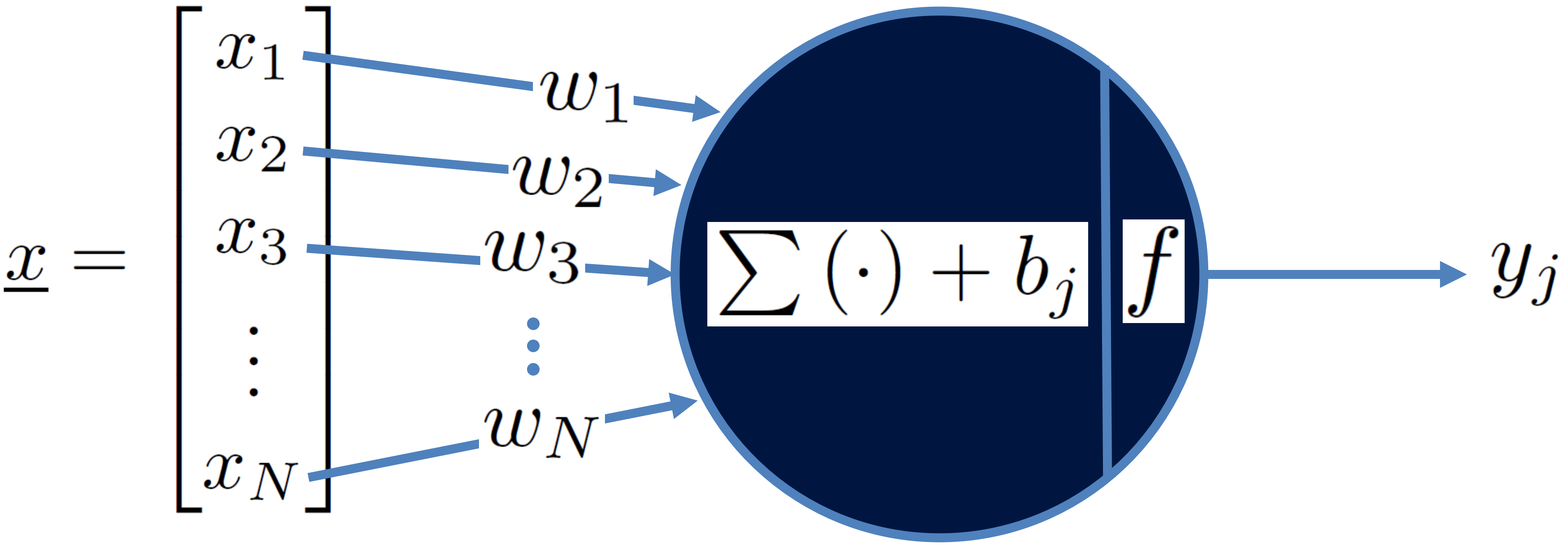
$$\boxed{y_j = f\left(\sum_{i=1}^{N} w_i x_i + b_j\right)}$$



A Perceptron Network

- Goal: Map *Input* Feature Vector **x** into *Output* Feature Vector **y**

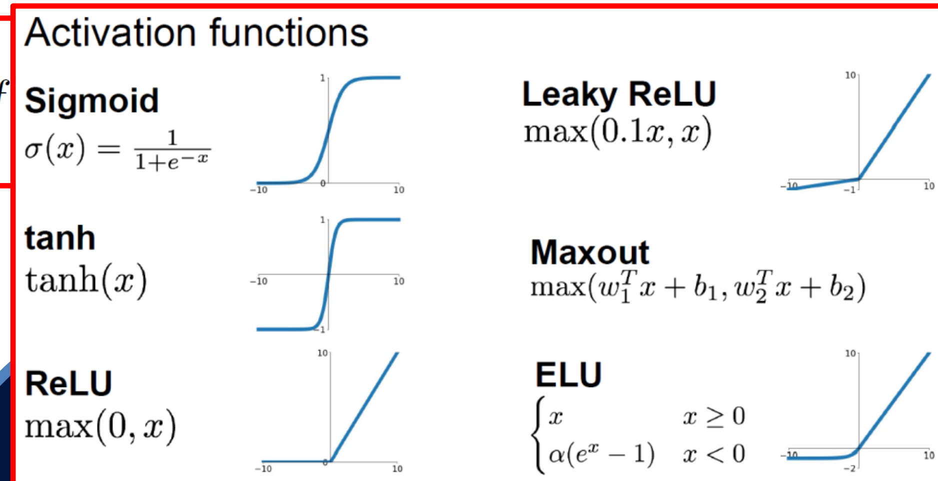
$$\boxed{y_j = f\left(\sum_{i=1}^{N} w_i x_i + b_j\right) = f\left(w^T x + b_j\right)}$$

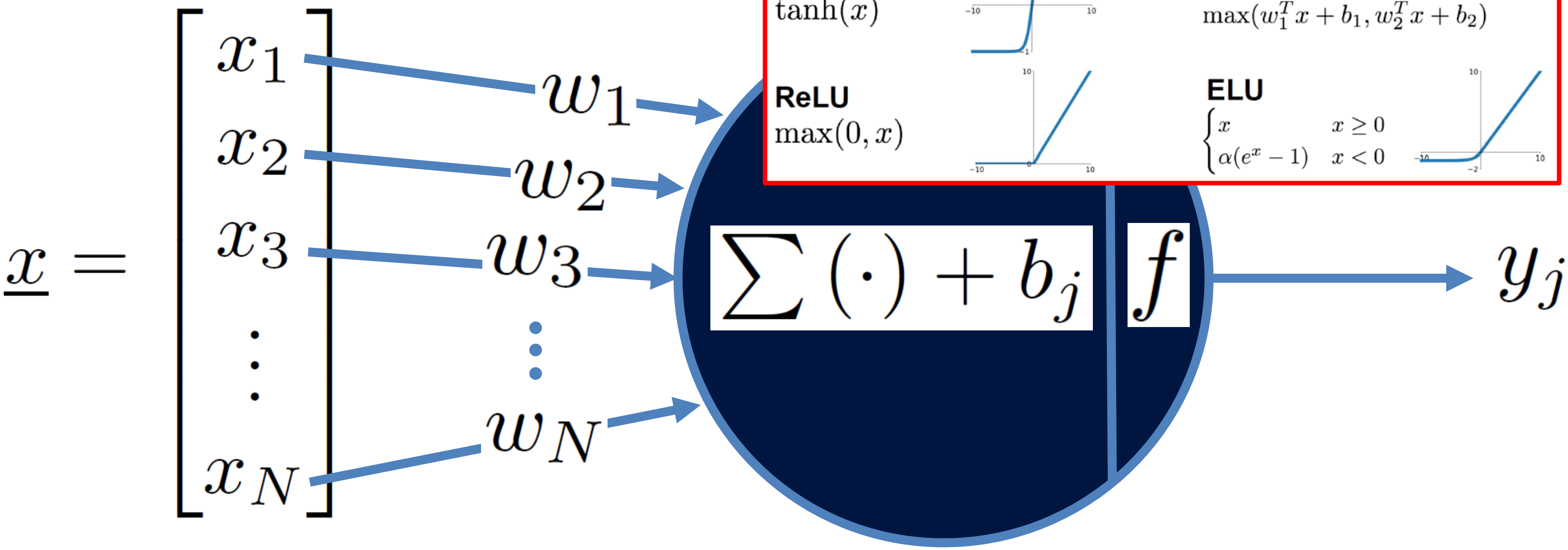


A Perceptron Network

- Goal: Map *Input* Feature Vector **x** into *Output* Feature Vector **y**

$$y_j = f\left(\sum_{i=1}^{N} w_i x_i + b_j\right) = f$$

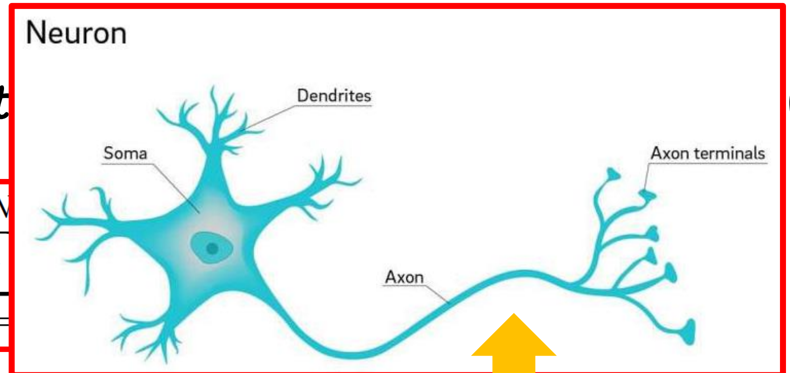


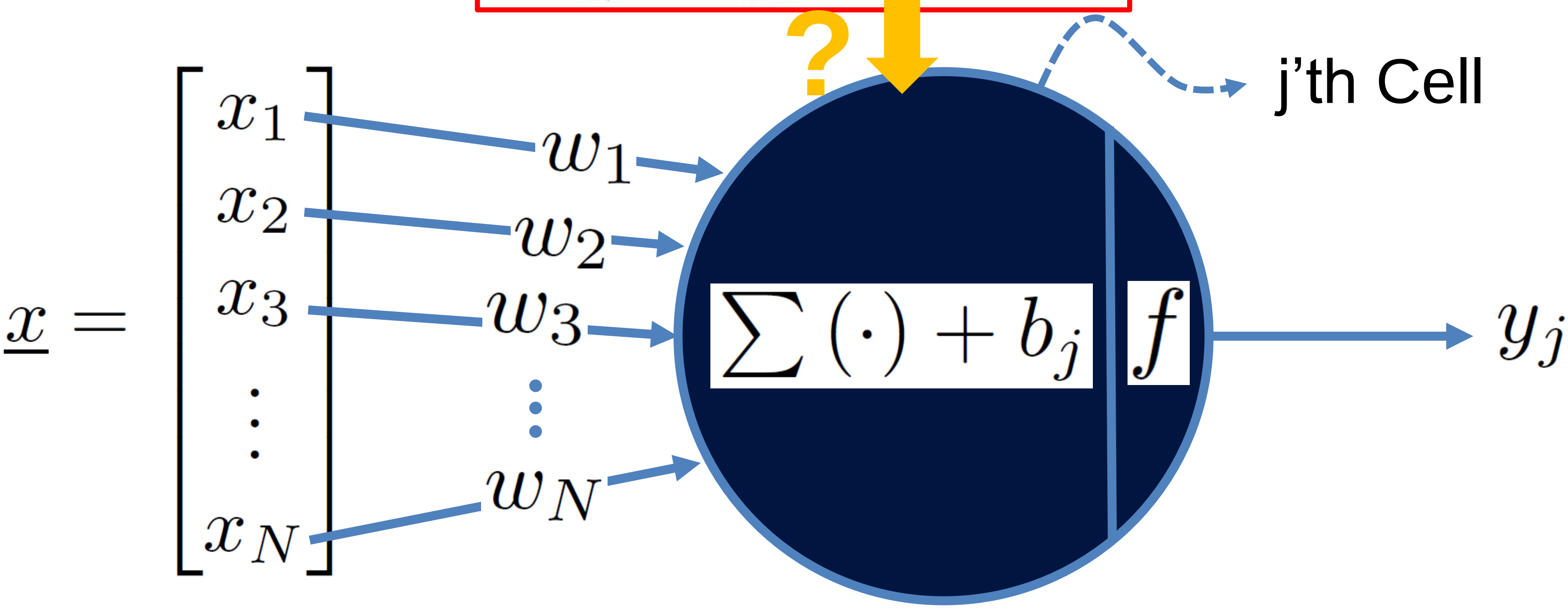


A Perceptron Network

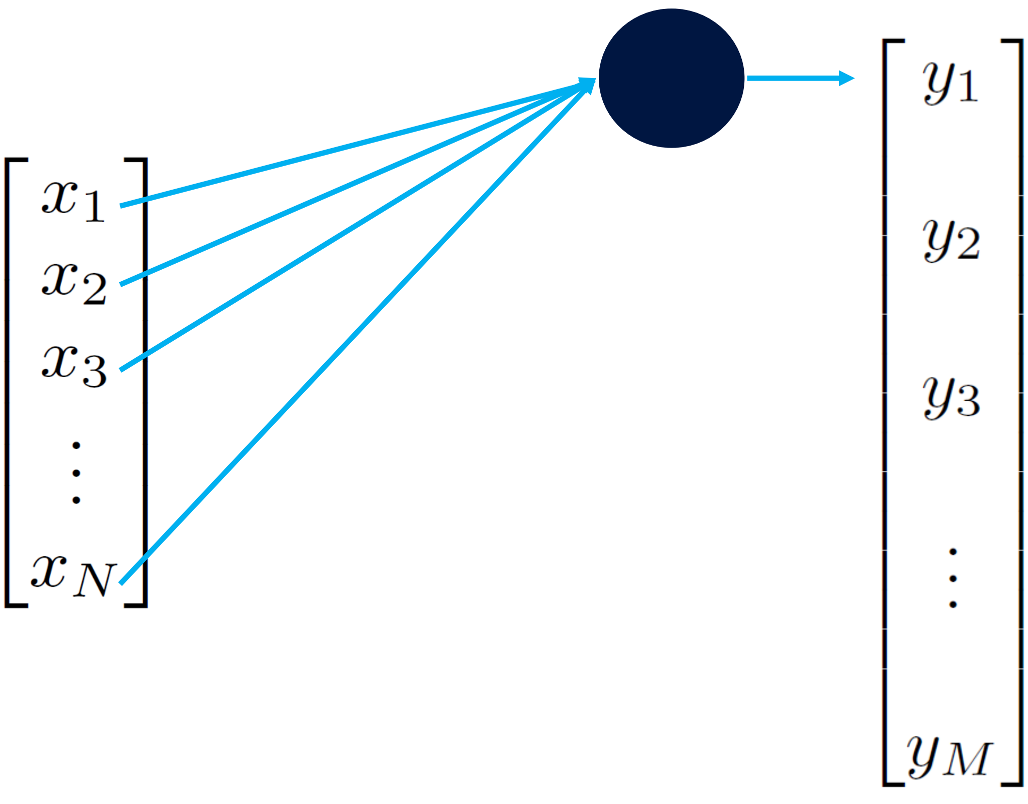
- Goal: Map Input ure Vector *y*

$y_j = f\left(\sum_{i=}^{N}\right.$

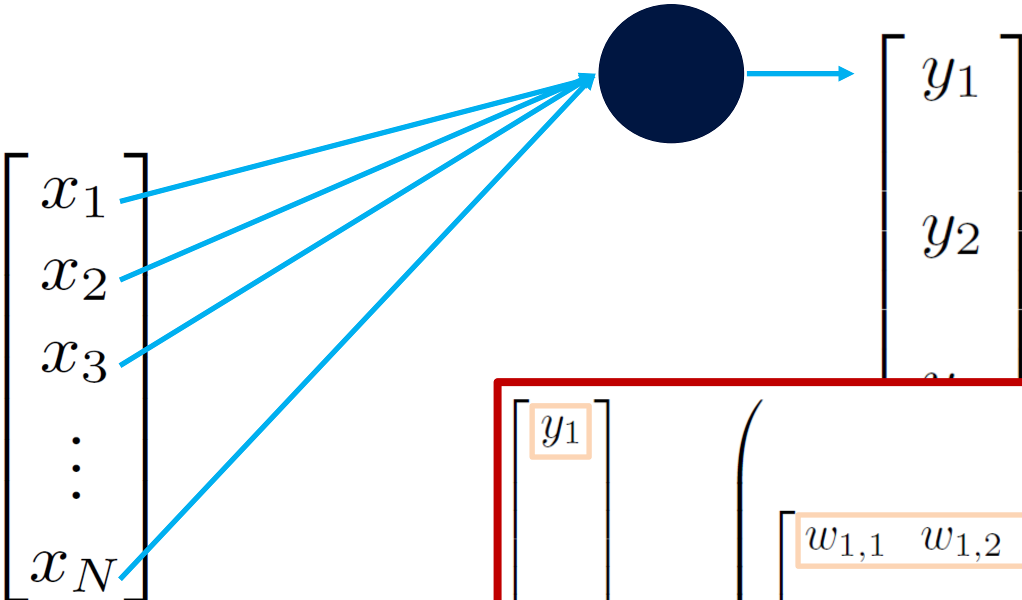




A Perceptron Network

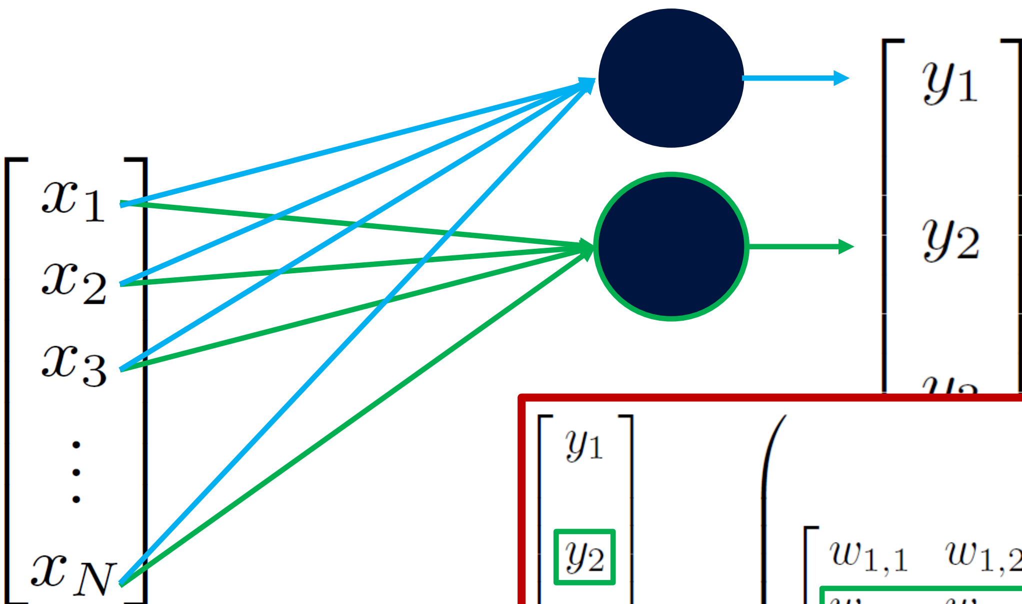


A Perceptron Network



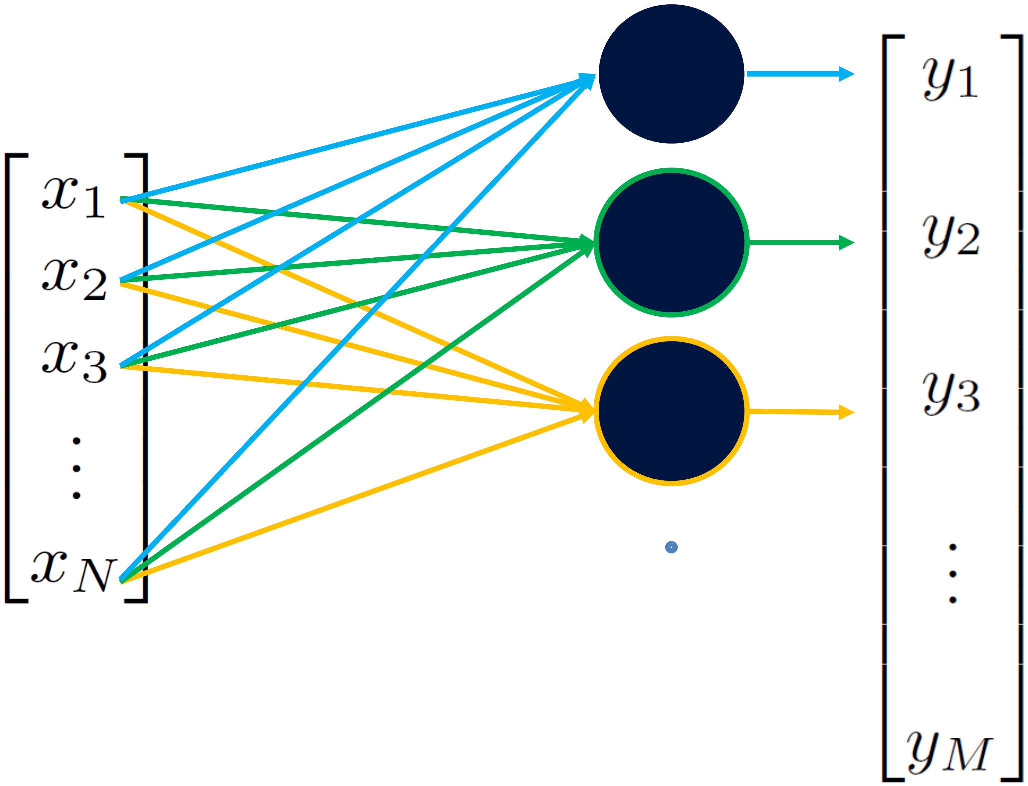
$$\begin{bmatrix} \boxed{y_1} \\ \\ \\ \\ \end{bmatrix} = f\left(\begin{bmatrix} \boxed{w_{1,1} \quad w_{1,2} \quad w_{1,3} \quad \cdots \quad w_{1,N}} \\ \\ \\ \\ \end{bmatrix} \boxed{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_N \end{bmatrix}} + \begin{bmatrix} \boxed{b_1} \\ \\ \\ \\ \end{bmatrix} \right)$$

A Perceptron Network

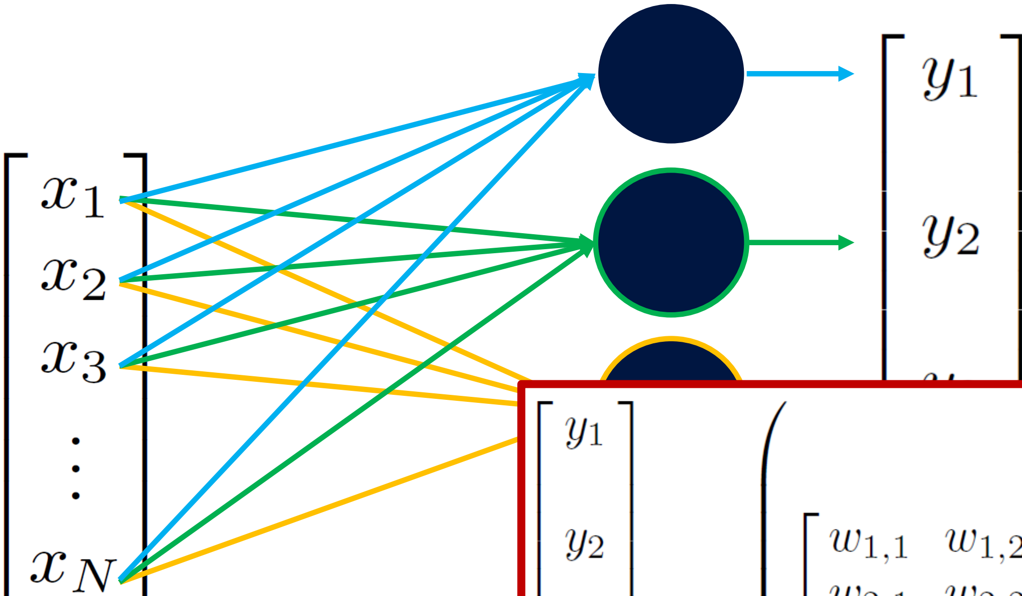


$$
\begin{bmatrix} y_1 \\ \boxed{y_2} \\ \\ \\ \end{bmatrix} = f\left(\begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} & \cdots & w_{1,N} \\ w_{2,1} & w_{2,2} & w_{2,3} & \cdots & w_{2,N} \\ & & & & \\ & & & & \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_N \end{bmatrix} + \begin{bmatrix} b_1 \\ \boxed{b_2} \\ \\ \\ \end{bmatrix} \right)
$$

A Perceptron Network

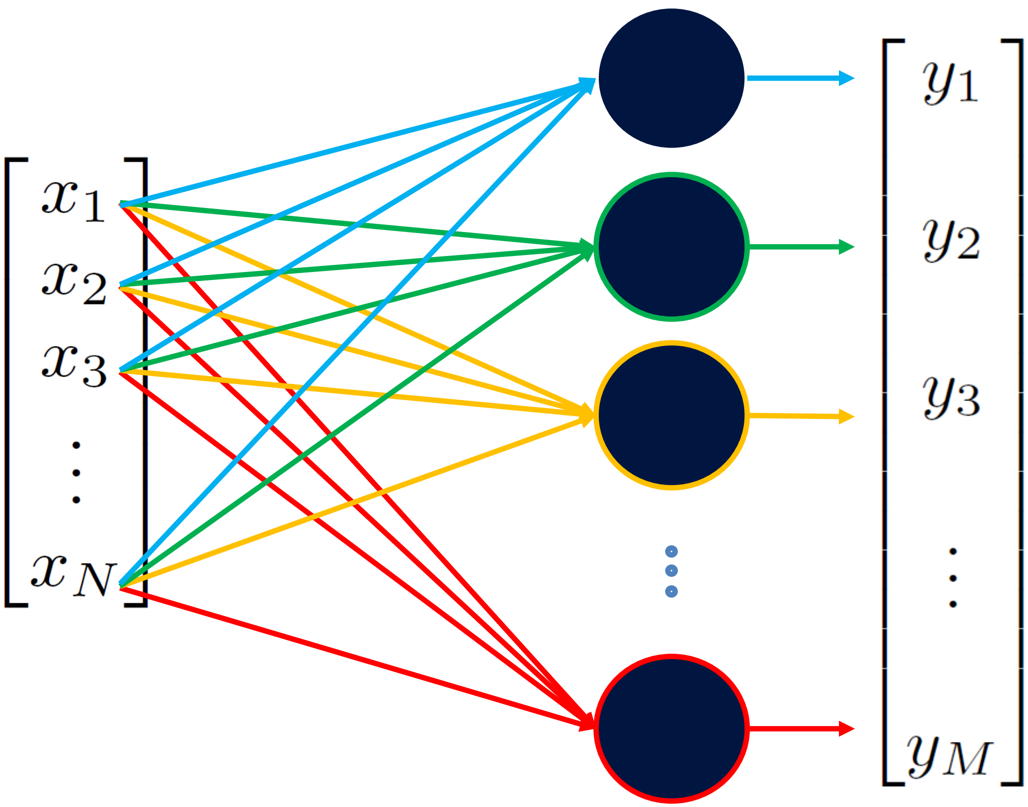


A Perceptron Network

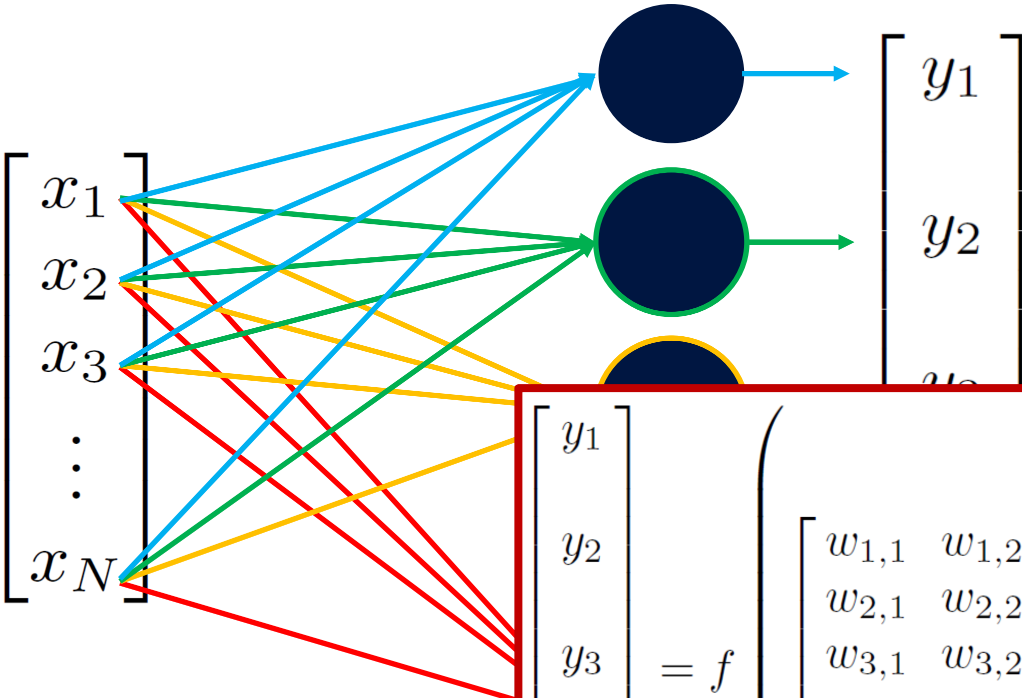


$$\begin{bmatrix} y_1 \\ y_2 \\ \boxed{y_3} \\ \\ \\ \end{bmatrix} = f\left(\begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} & \cdots & w_{1,N} \\ w_{2,1} & w_{2,2} & w_{2,3} & \cdots & w_{2,N} \\ \boxed{w_{3,1} \quad w_{3,2} \quad w_{3,3} \quad \cdots \quad w_{3,N}} \\ \\ \end{bmatrix} \boxed{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_N \end{bmatrix}} + \begin{bmatrix} b_1 \\ b_2 \\ \boxed{b_3} \\ \\ \\ \end{bmatrix} \right)$$

A Perceptron Network

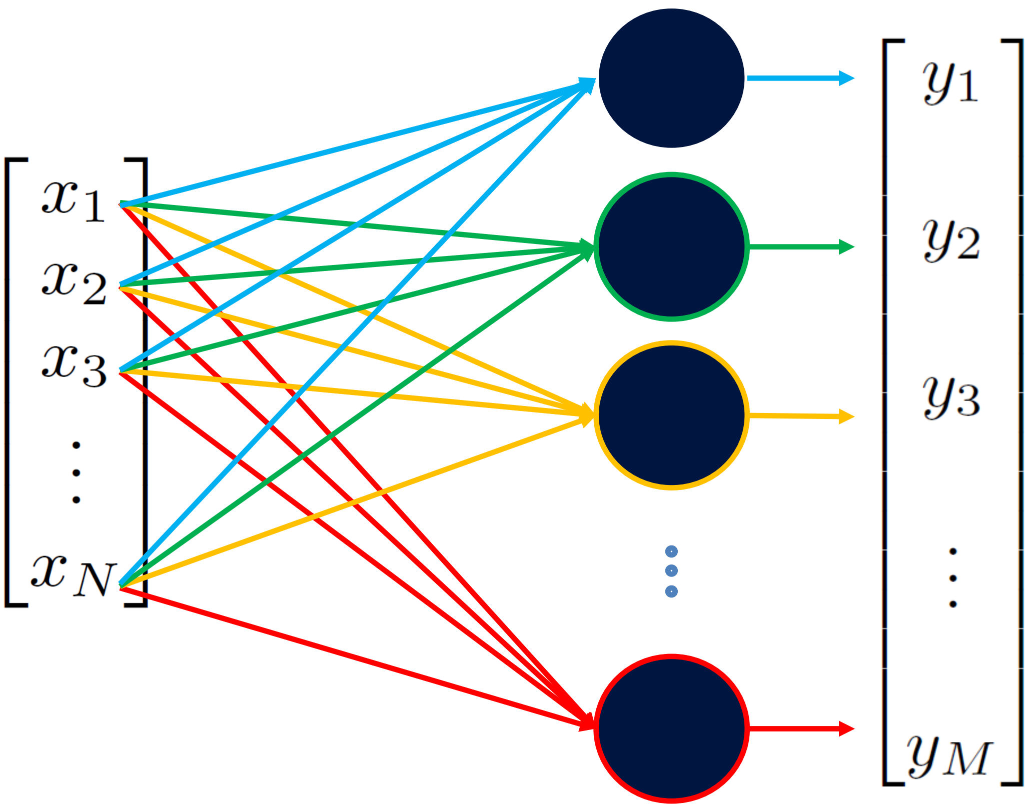


A Perceptron Network



$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ \boxed{y_M} \end{bmatrix} = f\left(\begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} & \cdots & w_{1,N} \\ w_{2,1} & w_{2,2} & w_{2,3} & \cdots & w_{2,N} \\ w_{3,1} & w_{3,2} & w_{3,3} & \cdots & w_{3,N} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{M,1} & w_{M,2} & w_{M,3} & \cdots & w_{M,N} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_N \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ \boxed{b_M} \end{bmatrix} \right)$$

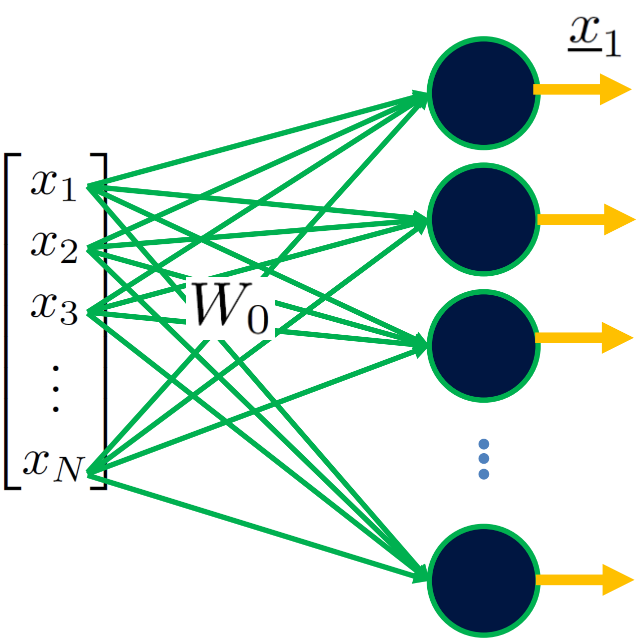
A Perceptron Network



$$\underline{y} = f\left(W^T \underline{x} + \underline{b}\right)$$

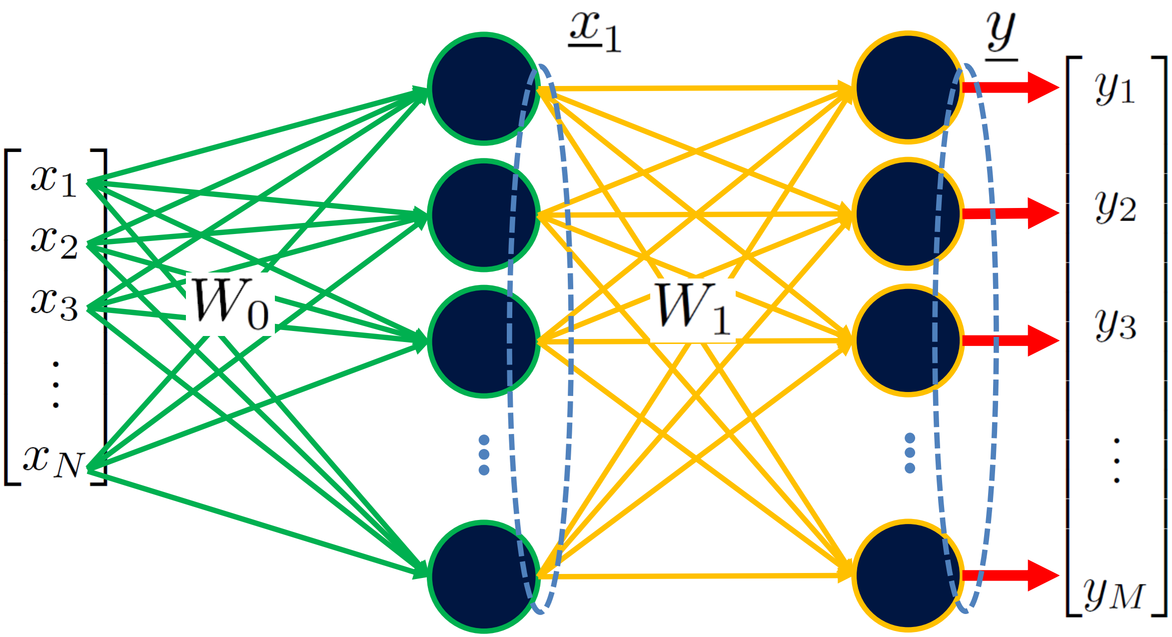
$\underline{x}$: input feature vector $N \times 1$
$\underline{y}$: input feature vector $M \times 1$
W: weight matrix $M \times N$
$\underline{\text{b}}$: bias vector $M \times 1$
$f(\cdot)$: activation function

A Perceptron Network



$$\underline{x}_1 = f\left(W_0^T \underline{x} + \underline{b}_0\right)$$

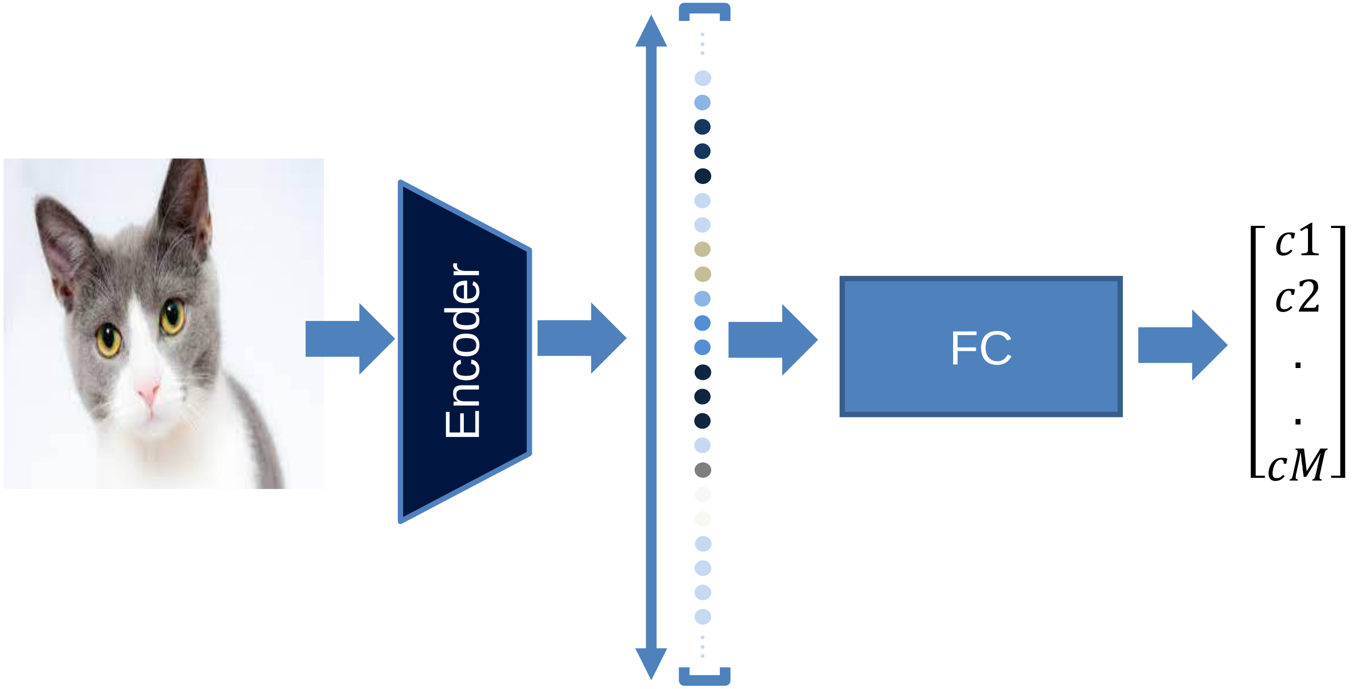
A Perceptron Network



$$\underline{x}_1 = f\left(W_0^T \underline{x} + \underline{b}_0\right)$$

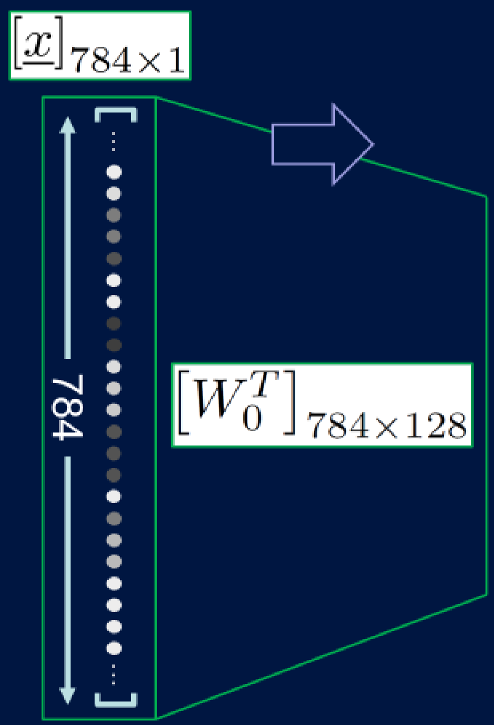
$$\underline{y} = f\left(W_1^T \underline{x}_1 + \underline{b}_1\right)$$

Fully Connected (FC) Network



Neural Network: FC Layer Design

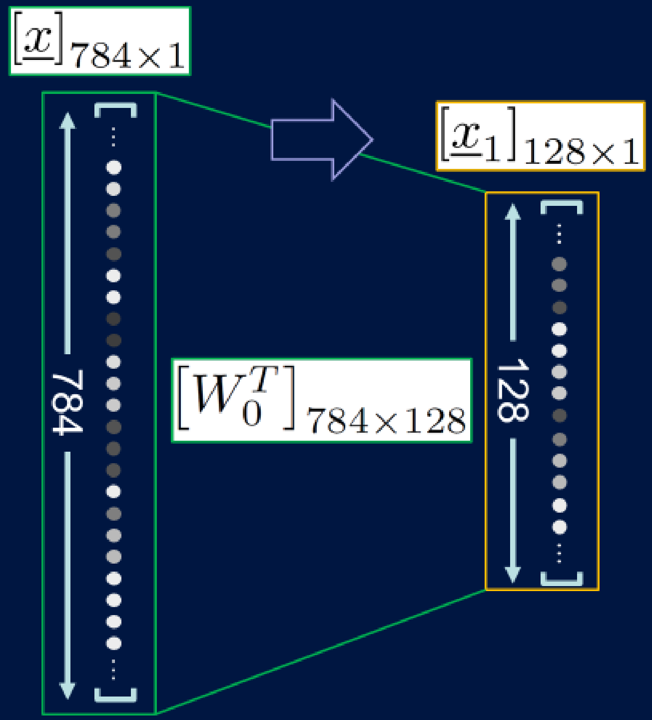
- First layer design: map input feature vector into smaller vector



Neural Network: FC Layer Design

- First layer design: activate features using non-linear activation

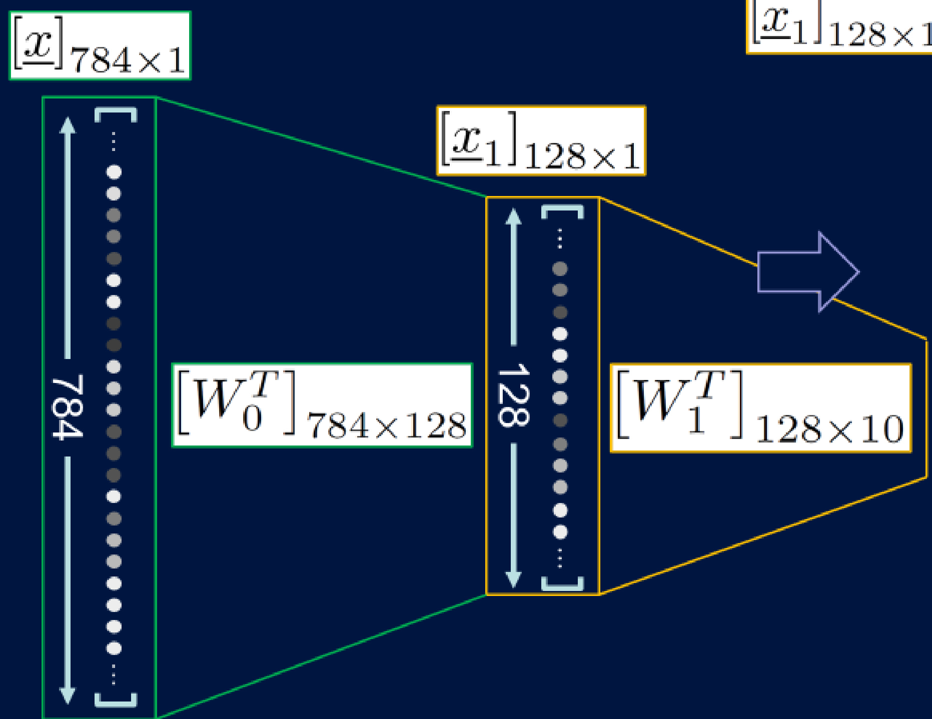
$$\boxed{[\underline{x}_1]_{128\times 1} = f\left(W_0^T \underline{x} + \underline{b}_0\right)}$$



Neural Network: FC Layer Design

- Second layer design: map input feature vector into #Classes

$$[\underline{x}_1]_{128\times1} = f\left(W_0^T \underline{x} + \underline{b}_0\right)$$

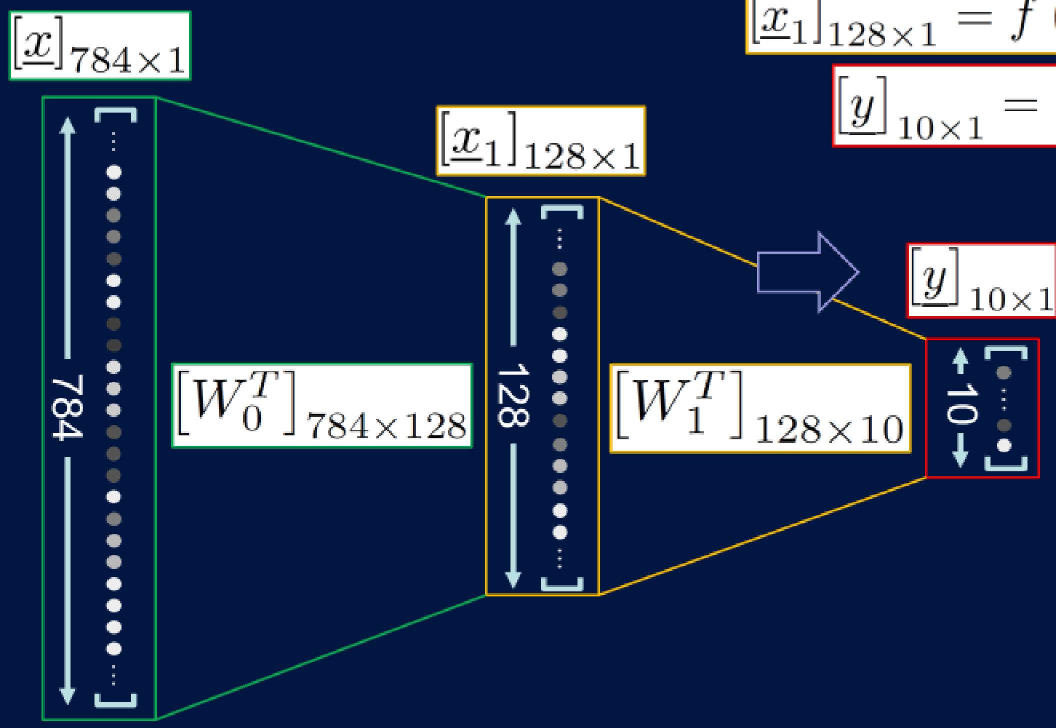


Neural Network: FC Layer Design

- Second layer design: no activation is required

$$[\underline{x}_1]_{128\times 1} = f\left(W_0^T \underline{x} + \underline{b}_0\right)$$

$$[\underline{y}]_{10\times 1} = W_1^T \underline{x}_1 + \underline{b}_1$$

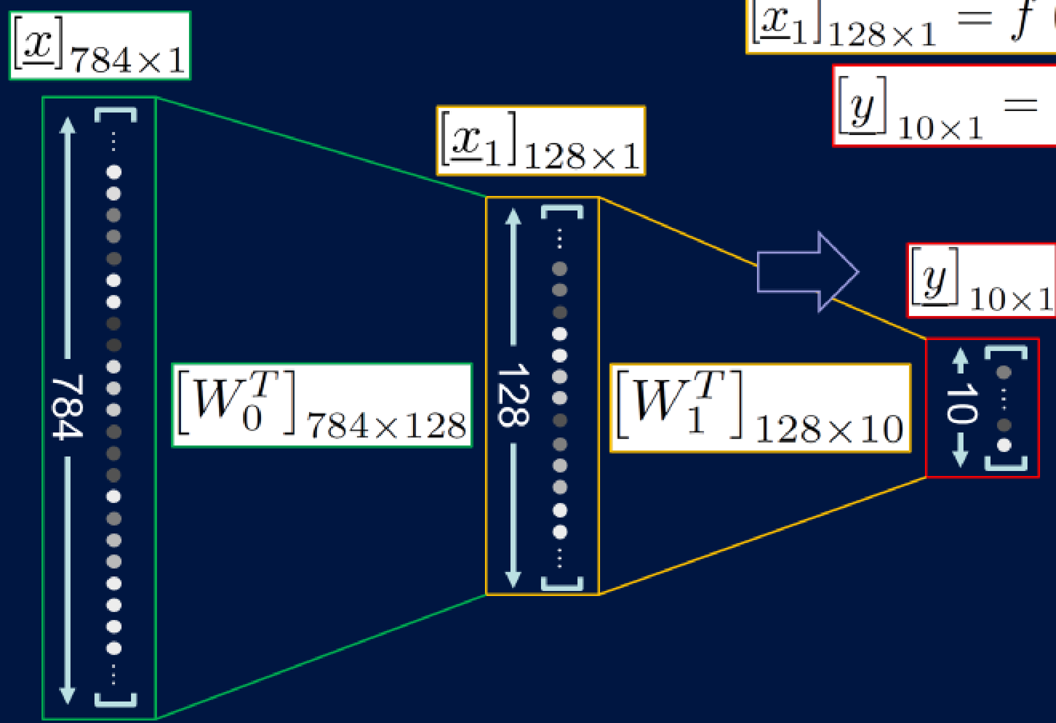


Neural Network: FC Layer Design

Quiz: What is the #Learnable Parameters?

$$[\underline{x}_1]_{128\times 1} = f\left(W_0^T \underline{x} + \underline{b}_0\right)$$

$$[\underline{y}]_{10\times 1} = W_1^T \underline{x}_1 + \underline{b}_1$$

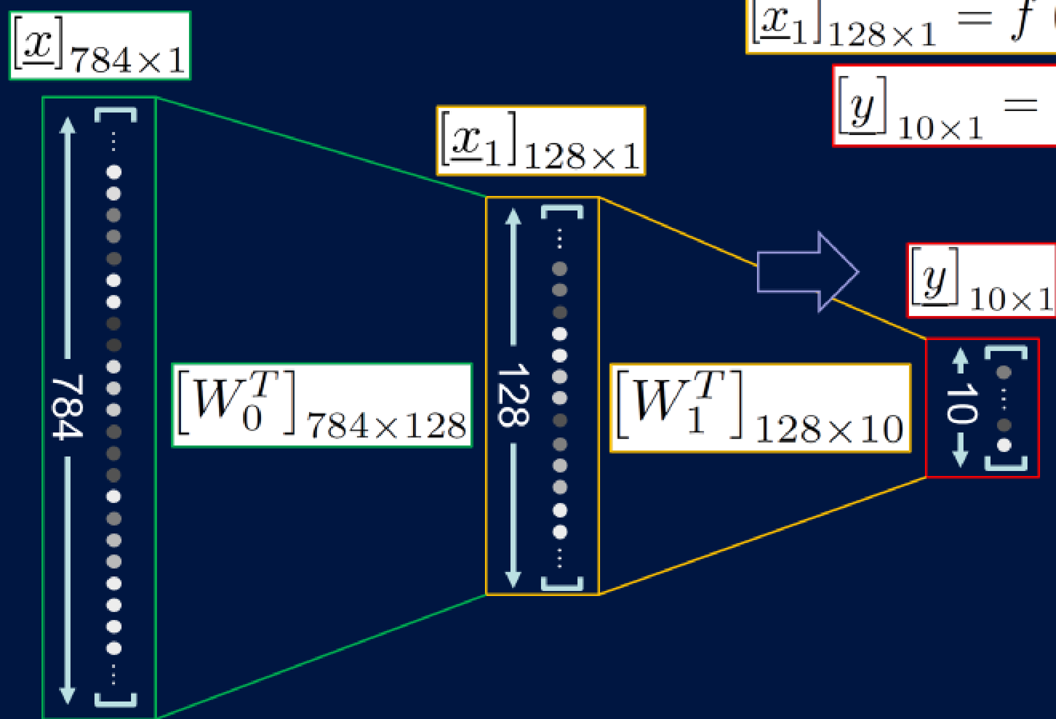


Neural Network: FC Layer Design

#Learnable Parameters = 784x128 + 128 + 128x10 + 10 = 101,770

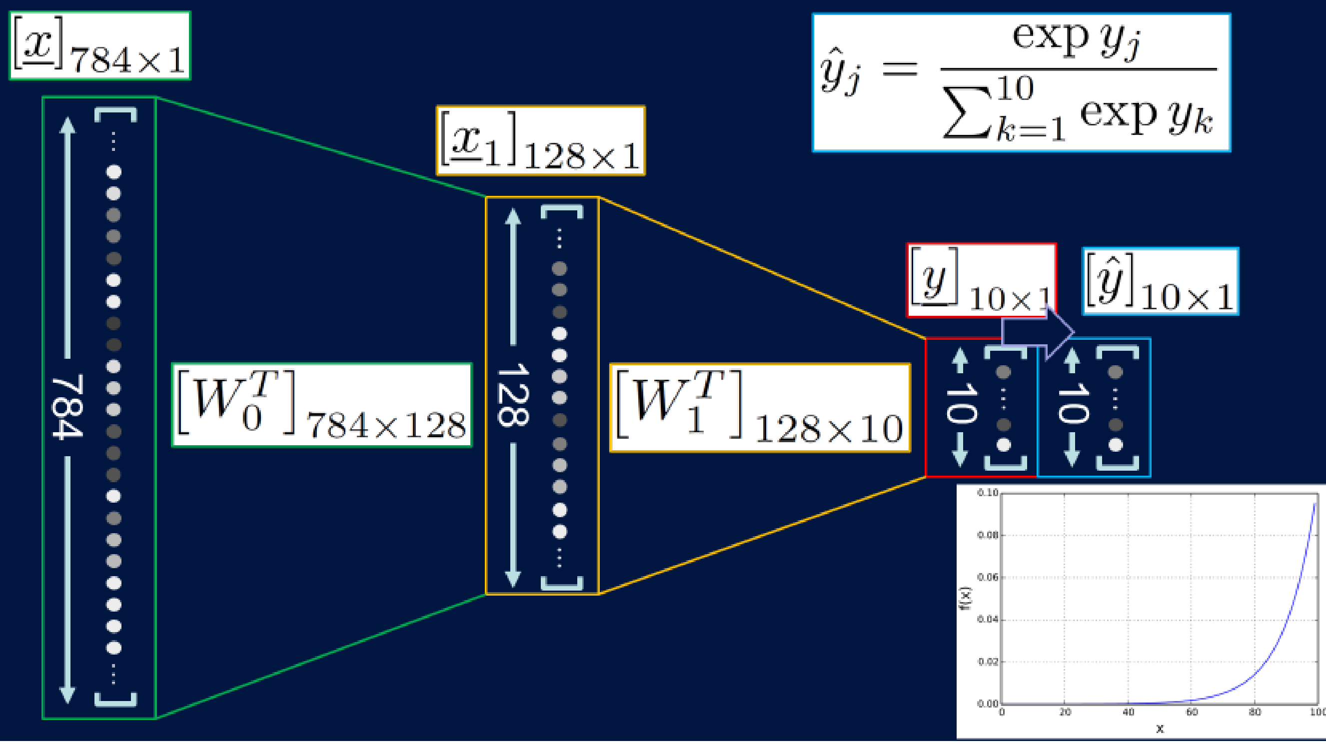
$$[\underline{x}_1]_{128\times 1} = f\left(W_0^T \underline{x} + \underline{b}_0\right)$$

$$[\underline{y}]_{10\times 1} = W_1^T \underline{x}_1 + \underline{b}_1$$



Neural Network: FC Layer Design

Apply Softmax regression model to map output classes in range between [0,1]



Fully Connected NN: Error-Gradient Backpropagation

- To train NN, the gradient of error-loss is calculated with respect to each learnable parameter in the network

$$\frac{\partial \epsilon}{\partial w_{0,i,j}}, \frac{\partial \epsilon}{\partial b_{0,j}}, \frac{\partial \epsilon}{\partial w_{1,i,j}}, \frac{\partial \epsilon}{\partial b_{1,j}}$$

The Edward S. Rogers Sr. Department of Electrical & Computer Engineering UNIVERSITY OF TORONTO

FACULTY OF APPLIED SCIENCE & ENGINEERING

Fully Connected NN: Error-Gradient Backpropagation

- To train NN, the gradient of error-loss is calculated with respect to each learnable parameter in the network

$$\frac{\partial \epsilon}{\partial w_{0,i,j}}, \frac{\partial \epsilon}{\partial b_{0,j}}, \frac{\partial \epsilon}{\partial w_{1,i,j}}, \frac{\partial \epsilon}{\partial b_{1,j}}$$

Rumelhart et al., 1986 introduced Backpropagation for training NN

- Error gradients are used to update the network parameters in iterative minimization using gradient descent

$$\text{e.g. } w_{0,i,j}^{k+1} \leftarrow w_{0,i,j}^{k} - \eta \frac{\partial \epsilon}{\partial w_{0,i,j}^{k+1}}$$

The Edward S. Rogers Sr. Department of Electrical & Computer Engineering UNIVERSITY OF TORONTO

FACULTY OF APPLIED SCIENCE & ENGINEERING

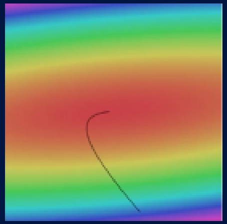
Fully Connected NN: Error-Gradient Backpropagation

- To train NN, the gradient of error-loss is calculated with respect to each learnable parameter in the network

$$\frac{\partial \epsilon}{\partial w_{0,i,j}}, \frac{\partial \epsilon}{\partial b_{0,j}}, \frac{\partial \epsilon}{\partial w_{1,i,j}}, \frac{\partial \epsilon}{\partial b_{1,j}}$$

- Error gradients are used to update the network parameters in iterative minimization using gradient descent

$$\text{e.g. } w_{0,i,j}^{k+1} \leftarrow w_{0,i,j}^{k} - \eta \frac{\partial \epsilon}{\partial w_{0,i,j}^{k+1}}$$



The Edward S. Rogers Sr. Department of Electrical & Computer Engineering
UNIVERSITY OF TORONTO
FACULTY OF APPLIED SCIENCE & ENGINEERING

Fully Connected NN: Error-Gradient Backpropagation

- To train NN, the gradient of error-loss is calculated with respect to each learnable parameter in the network

$\frac{\partial \epsilon}{\partial w_{0,i,j}}, \frac{\partial}{\partial b_0}$

- Error gradien
iterative mini

e.g. $w_{0,i,j}^{k+1} \leftarrow$

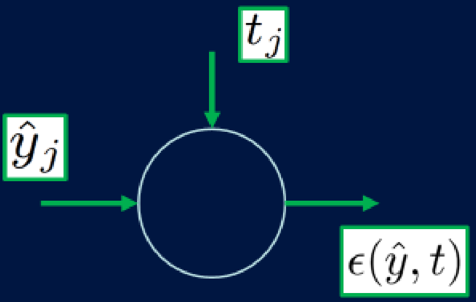
```python
iteration_p = 0
for p in group['params']:
    if p.grad is None:
        iteration_p += 1
        continue
    d_p = p.grad.data
    if weight_decay != 0:
        d_p.add_(weight_decay, p.data)
    if momentum != 0:
        param_state = self.state[p]
        if 'momentum_buffer' not in param_state:
            buf = param_state['momentum_buffer'] = torch.clone(d_p).detach()
        else:
            buf = param_state['momentum_buffer']
            buf.mul_(momentum).add_(1 - dampening, d_p)
        if nesterov:
            d_p = d_p.add(momentum, buf)
        else:
            d_p = buf

    p.data.add_(-group['lr'], d_p)
    iteration_p += 1

return loss
```

Fully Connected NN: Calculating Error-Gradients using Chain-Rule

- Classification sub-module

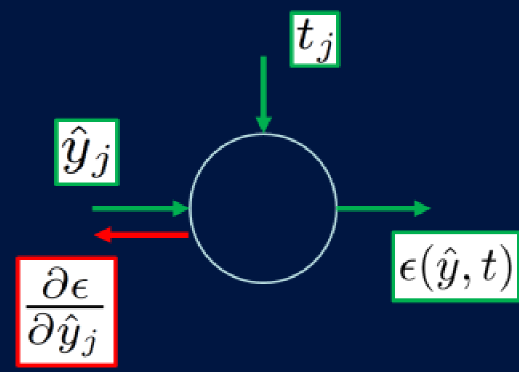


$$\boxed{\epsilon(\hat{y}, t) = -\sum_{k=1}^{10} t_k \ln \hat{y}_k = -\underline{t}^T \ln \underline{\hat{y}}}$$

The Edward S. Rogers Sr. Department of Electrical & Computer Engineering
UNIVERSITY OF TORONTO
FACULTY OF APPLIED SCIENCE & ENGINEERING

Fully Connected NN: Calculating Error-Gradients using Chain-Rule

- Classification sub-module



$$\boxed{\epsilon(\hat{y}, t) = -\sum_{k=1}^{10} t_k \ln \hat{y}_k = -\underline{t}^T \ln \underline{\hat{y}}}$$

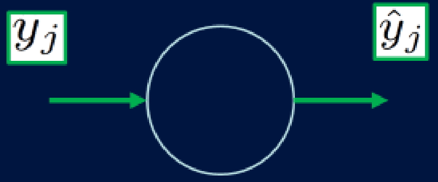
$$\boxed{\frac{\partial \epsilon}{\partial \hat{y}_j} = -\frac{t_j}{\hat{y}_j}}$$

The Edward S. Rogers Sr. Department of Electrical & Computer Engineering UNIVERSITY OF TORONTO

FACULTY OF APPLIED SCIENCE & ENGINEERING

Fully Connected NN: Calculating Error-Gradients using Chain-Rule

- Probability sub-module

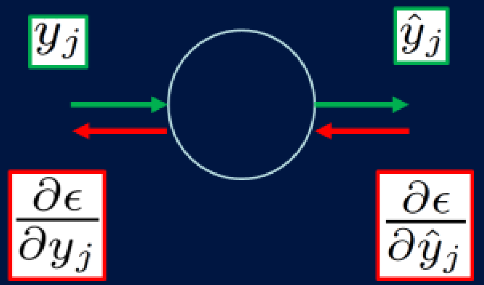


$$\hat{y}_j = \frac{\exp y_j}{\sum_{k=1}^{10} \exp y_k}$$

The Edward S. Rogers Sr. Department
of Electrical & Computer Engineering
UNIVERSITY OF TORONTO
FACULTY OF APPLIED SCIENCE & ENGINEERING

Fully Connected NN: Calculating Error-Gradients using Chain-Rule

- Probability sub-module



$$\hat{y}_j = \frac{\exp y_j}{\sum_{k=1}^{10} \exp y_k}$$

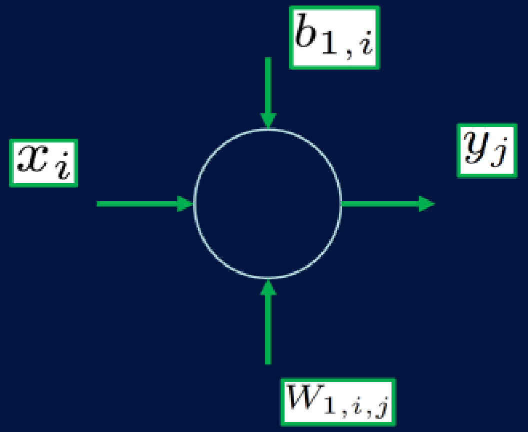
$$\frac{\partial \epsilon}{\partial y_j} = \frac{\partial \epsilon}{\partial \hat{y}_j} \cdot \frac{\partial \hat{y}_j}{\partial y_j} = \frac{\partial \epsilon}{\partial \hat{y}_j} \cdot \hat{y}_j \cdot (1 - \hat{y}_j)$$

The Edward S. Rogers Sr. Department of Electrical & Computer Engineering UNIVERSITY OF TORONTO

FACULTY OF APPLIED SCIENCE & ENGINEERING

Fully Connected NN: Calculating Error-Gradients using Chain-Rule

- FC Layer sub-module



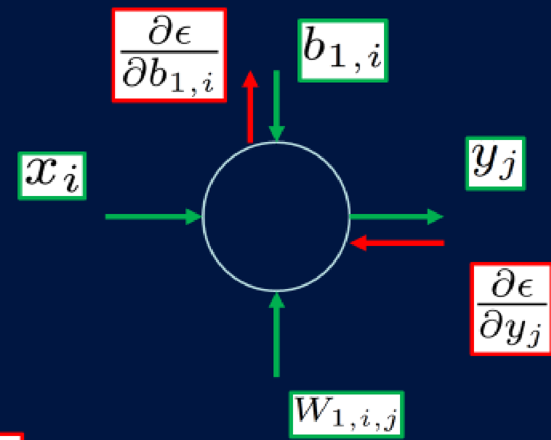
$$\boxed{y_j = \sum_{k=1}^{128} W_{1,k,j} x_k + b_{1,j}}$$

The Edward S. Rogers Sr. Department of Electrical & Computer Engineering
UNIVERSITY OF TORONTO

FACULTY OF APPLIED SCIENCE & ENGINEERING

Fully Connected NN: Calculating Error-Gradients using Chain-Rule

- FC Layer sub-module



$$\boxed{y_j = \sum_{k=1}^{128} W_{1,k,j} x_k + b_{1,j}}$$

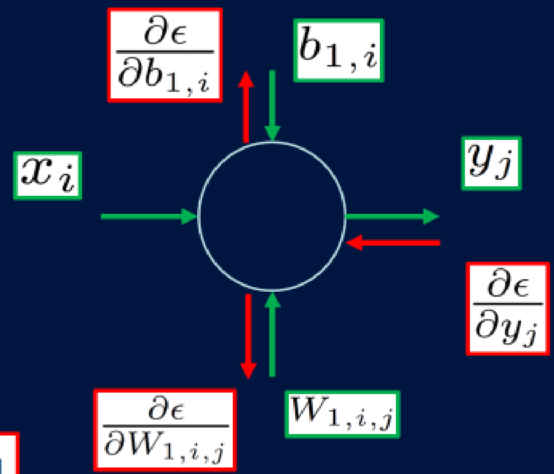
$$\boxed{\frac{\partial \epsilon}{\partial b_{1,i}} = \frac{\partial \epsilon}{\partial y_j} \cdot \frac{\partial y_j}{\partial b_{1,i}} = \frac{\partial \epsilon}{\partial y_j} \cdot 1}$$

The Edward S. Rogers Sr. Department of Electrical & Computer Engineering UNIVERSITY OF TORONTO

FACULTY OF APPLIED SCIENCE & ENGINEERING

Fully Connected NN: Calculating Error-Gradients using Chain-Rule

- FC Layer sub-module



$$y_j = \sum_{k=1}^{128} W_{1,k,j} x_k + b_{1,j}$$

$$\frac{\partial \epsilon}{\partial b_{1,i}} = \frac{\partial \epsilon}{\partial y_j} \cdot \frac{\partial y_j}{\partial b_{1,i}} = \frac{\partial \epsilon}{\partial y_j} \cdot 1$$

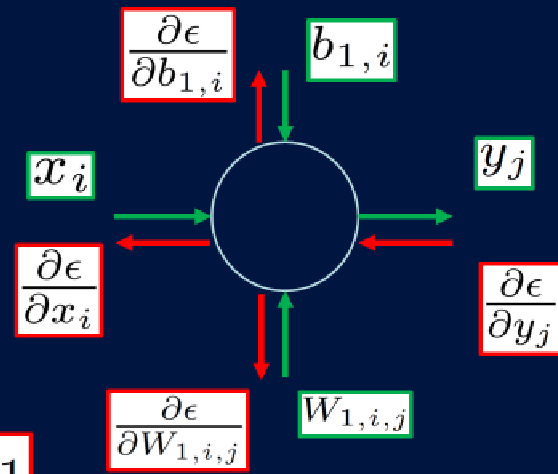
$$\frac{\partial \epsilon}{\partial W_{1,i,j}} = \frac{\partial \epsilon}{\partial y_j} \cdot \frac{\partial y_j}{\partial W_{1,i,j}} = \frac{\partial \epsilon}{\partial y_j} \cdot x_i$$

The Edward S. Rogers Sr. Department of Electrical & Computer Engineering
UNIVERSITY OF TORONTO

FACULTY OF APPLIED SCIENCE & ENGINEERING

Fully Connected NN: Calculating Error-Gradients using Chain-Rule

- FC Layer sub-module



$$y_j = \sum_{k=1}^{128} W_{1,k,j} x_k + b_{1,j}$$

$$\frac{\partial\epsilon}{\partial b_{1,i}} = \frac{\partial\epsilon}{\partial y_j}\cdot\frac{\partial y_j}{\partial b_{1,i}} = \frac{\partial\epsilon}{\partial y_j}\cdot 1$$

$$\frac{\partial\epsilon}{\partial W_{1,i,j}} = \frac{\partial\epsilon}{\partial y_j}\cdot\frac{\partial y_j}{\partial W_{1,i,j}} = \frac{\partial\epsilon}{\partial y_j}\cdot x_i$$

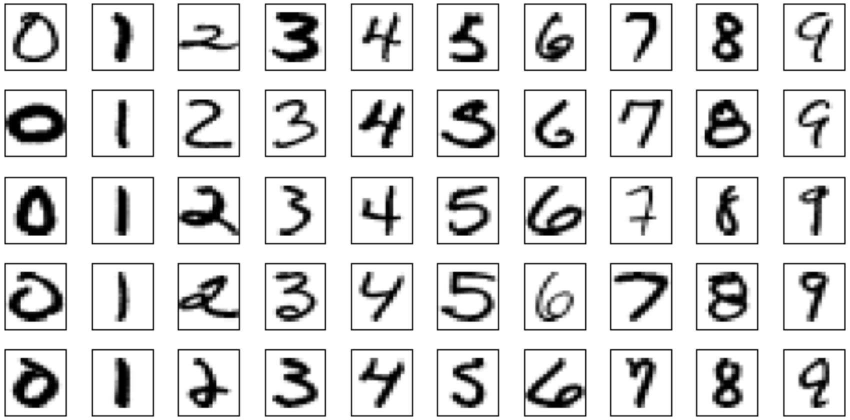
$$\frac{\partial\epsilon}{\partial x_i} = \frac{\partial\epsilon}{\partial y_j}\cdot\frac{\partial y_j}{\partial x_i} = \frac{\partial\epsilon}{\partial y_j}\cdot W_{1,i,j}$$

The Edward S. Rogers Sr. Department of Electrical & Computer Engineering
UNIVERSITY OF TORONTO

FACULTY OF APPLIED SCIENCE & ENGINEERING

Applications of Neural Networks

- Handwritten digit recognition
  - Training set = set of handwritten digits (0…9)
  - Task: given a bitmap, determine what digit it represents
  - Input: 1 feature for each pixel of the bitmap
  - Output: 1 output unit for each possible character (only 1 should be activated)
  - After training, network should work for fonts (handwriting) never encountered

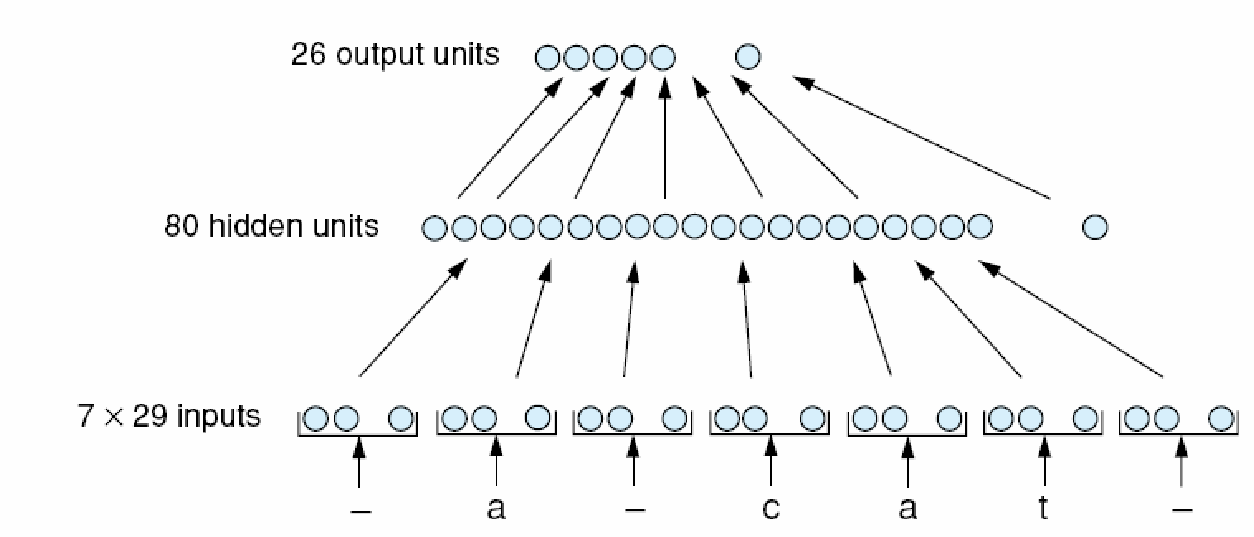


- Related pattern recognition applications:
  - recognize postal codes
  - recognize signatures
  - …

Applications of Neural Networks

- Speech synthesis
  - Learning to pronounce English words
  - Difficult task for a rule-based system because English pronunciation is highly irregular
  - Examples:
    - letter "c" can be pronounced [k] (*cat*) or [s] (*cents*)
    - *Woman* vs *Women*
  - NETtalk:
    - uses the context and the letters around a letter to learn how to pronounce a letter
    - Input: letter and its surrounding letters
    - Output: phoneme

NETtalk Architecture



Ex: *a cat* → *c is pronounced K*

- Network is made of 3 layers of units
- input unit corresponds to a 7 character window in the text
- each position in the window is represented by 29 input units (26 letters + 3 for punctuation and spaces)
- 26 output units – one for each possible phoneme

Listen to the output through iterations: https://www.youtube.com/watch?v=gakJlr3GecE

source: Luger (2005)

Neural Networks

- Disadvantage:
  - result is not easy to understand by humans (set of weights compared to decision tree)… it is a black box
- Advantage:
  - robust to noise in the input (small changes in input do not normally cause a change in output) and graceful degradation

Today

- Introduction to Neural Networks
  - Perceptrons
  - Backpropagation