Lecture Notes for

Machine Learning in Python

⟦👨‍🏫, 👨‍💻, 🐍, 👨‍🔬⟧

Professor Eric Larson

MLP History + ~Town Hall

Class Logistics and Agenda

- Logistics:
  - Grading Update
  - Next time: Flipped Module on back propagation
- Multi Week Agenda:
  - Today: Neural Networks History, up to 1980
  - Today: Multi-layer Architectures
  - Town Hall, Lab 3 (probably next week)
  - **Flipped**: Programming Multi-layer training

Class Overview, by topic

Table Data Visualization

Numpy, Pandas, Seaborn
Overviews with some in-depth discussion

Dimension Reduction and Image Processing

Scikit-learn, Scikit Image,
Intuition only, Some mathematics

Linear and Logistic Regression

Numpy, Recreate API for Scikit-learn
Detailed mathematics for simple optimization
intuition for advanced optimization

Neural Networks and Back Prop.

Numpy
Detailed mathematics for NN operations

Wide and Deep Networks

Convolutional Networks

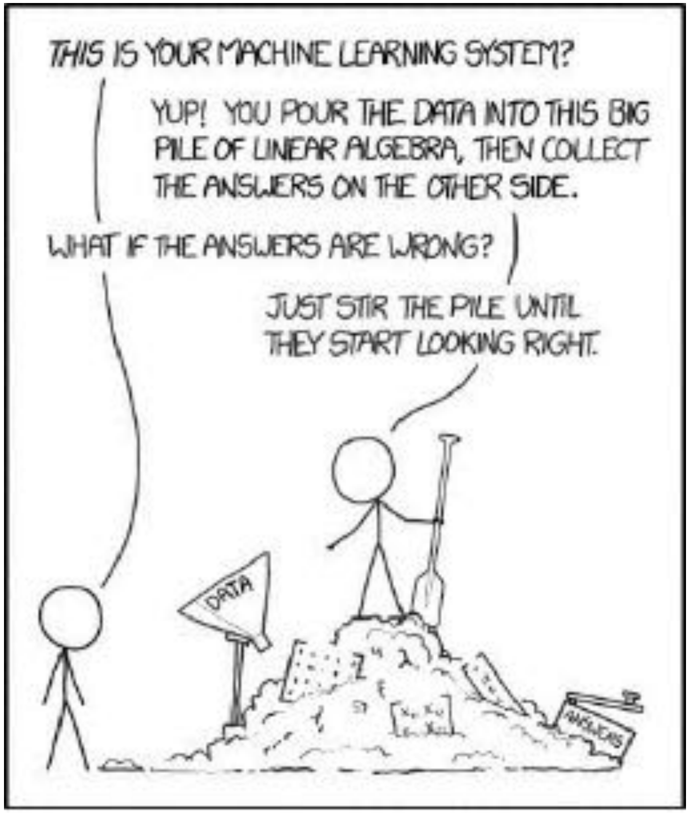
Recurrent Networks

Keras, Tensorflow
Intuition, Detailed implement.

Ethics in Language Models

ConceptNet
Case studies

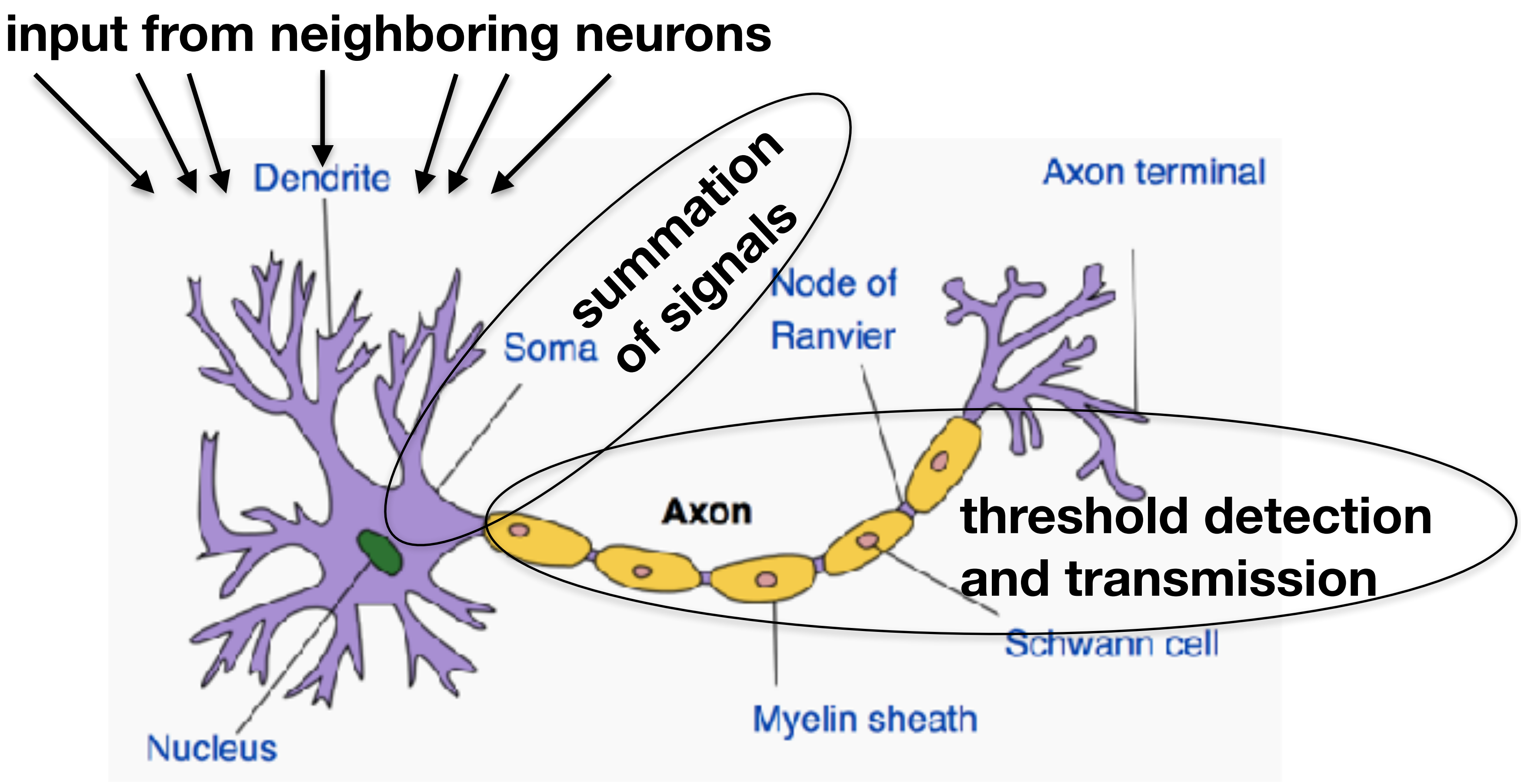
A History of Neural Networks



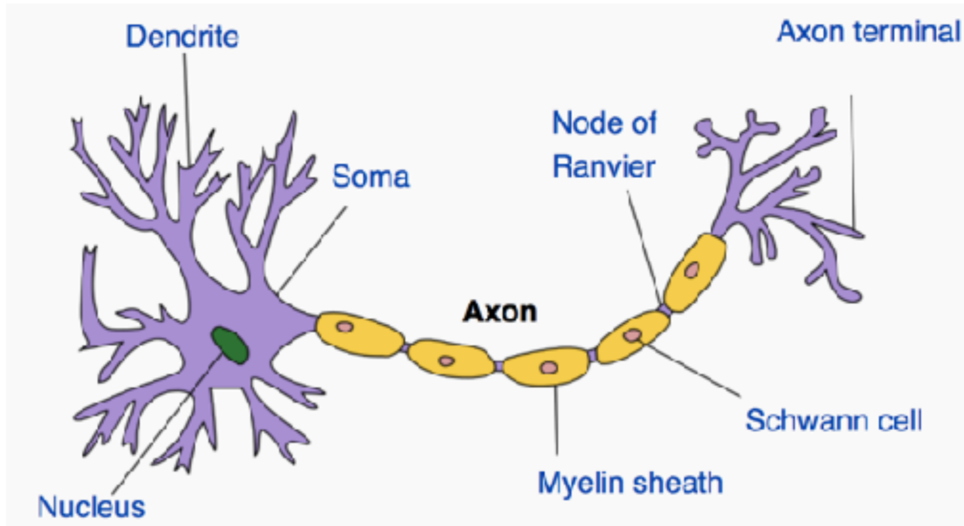
Machine Learning 101

Neurons

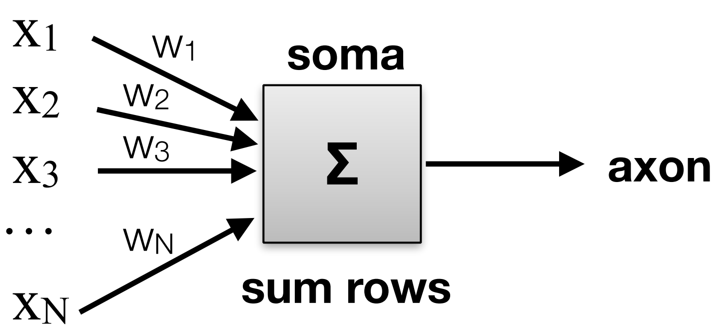
- From biology to modeling:



McCulloch and Pitts, 1943



dendrite



input

$$\mathbf{x} \cdot \mathbf{w} = a$$

for each neuron

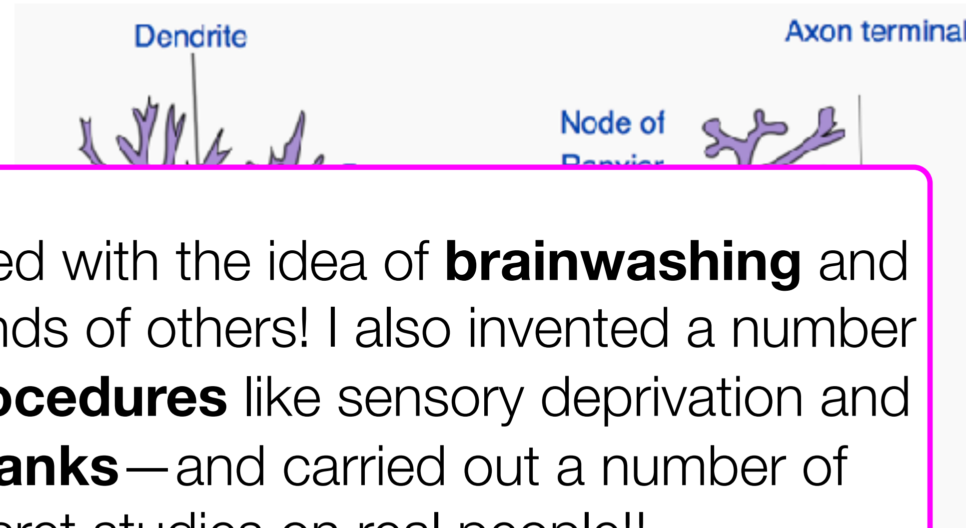
logic gates of the mind

Warren McCulloch

Walter Pitts

Neurons

- McCulloch and Pitts, 1943
- Donald Hebb, 1949
  - Hebb’s Law: close neurons fire together
  - neurons “learn
  - easier synaptic
  - basis of neura



I was infatuated with the idea of **brainwashing** and controlling minds of others! I also invented a number of **torture procedures** like sensory deprivation and **isolation tanks**—and carried out a number of secret studies on real people!!

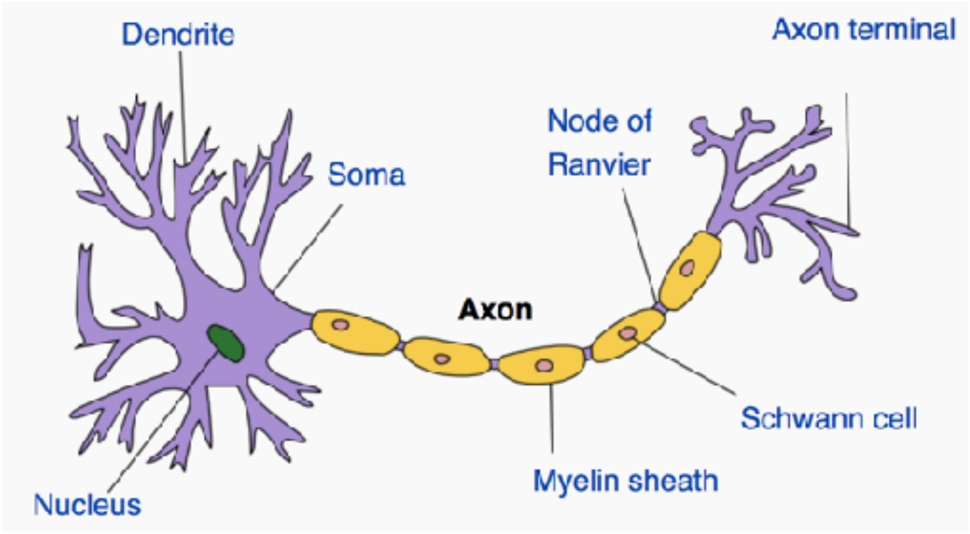
Donald O. Hebb

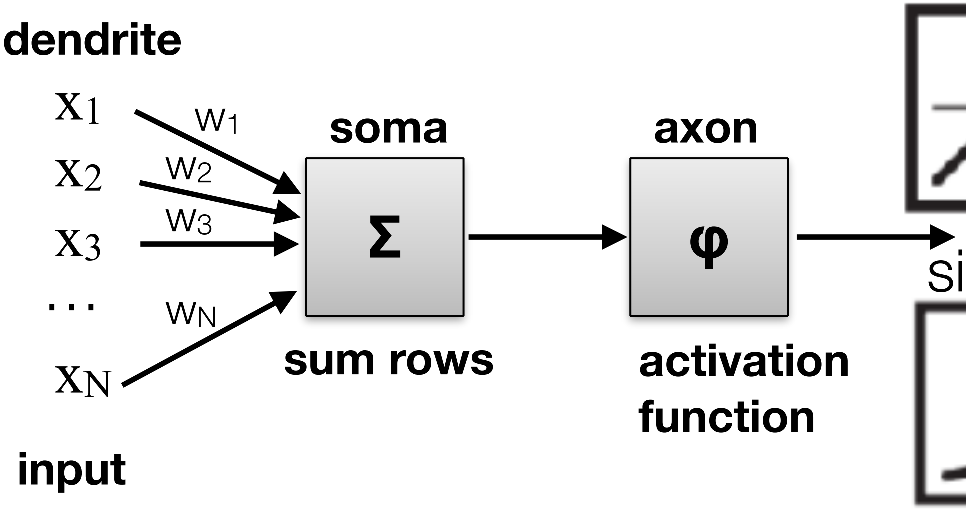
Warren McCulloch

Walter Pitts

Rosenblatt’s perceptron, 1957

Frank Rosenblatt





hard limit

$a=-1 \quad z<0$

$a=1 \quad z>=0$

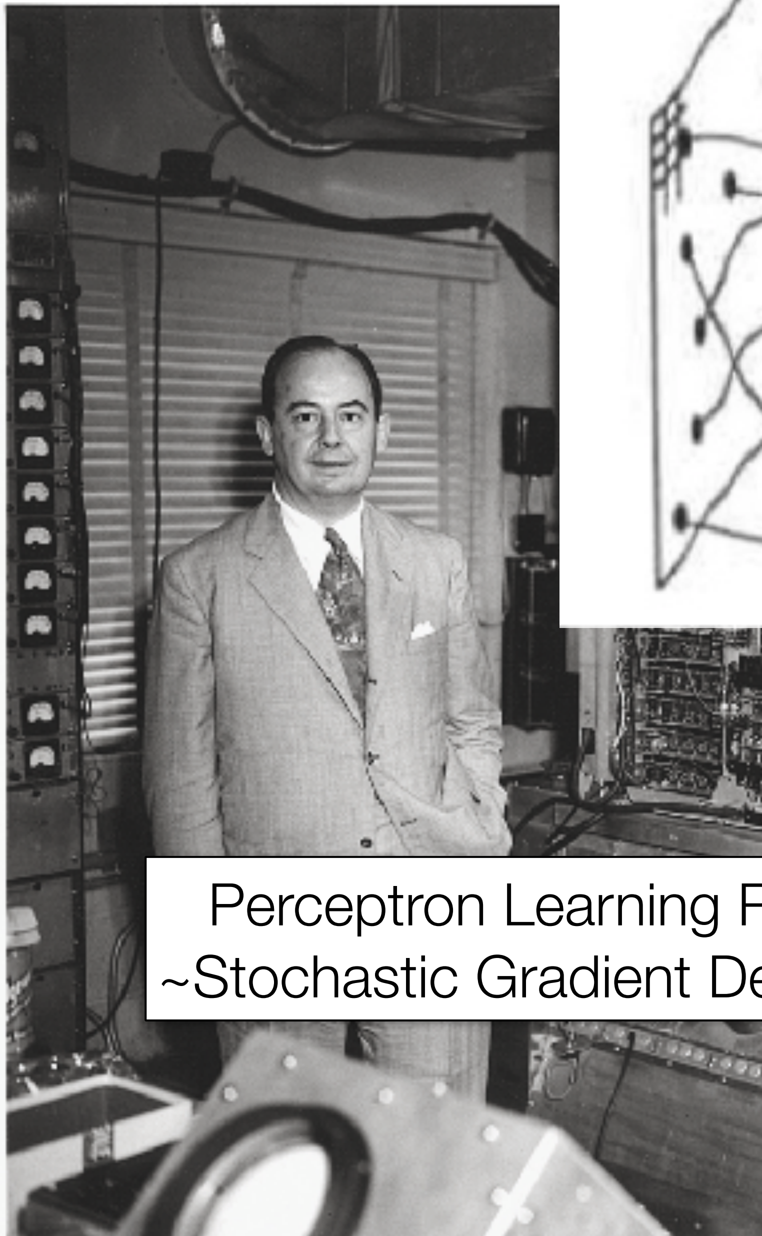
linear

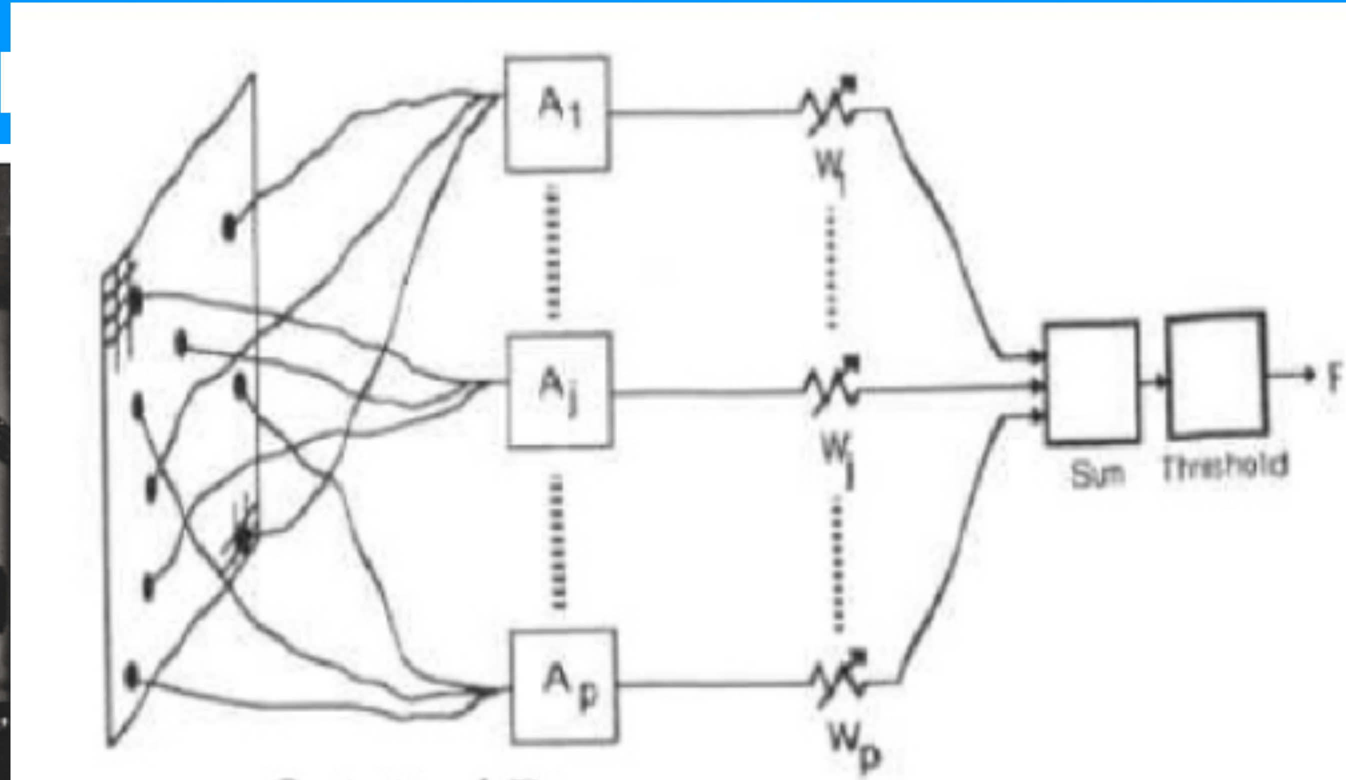
$a=z$

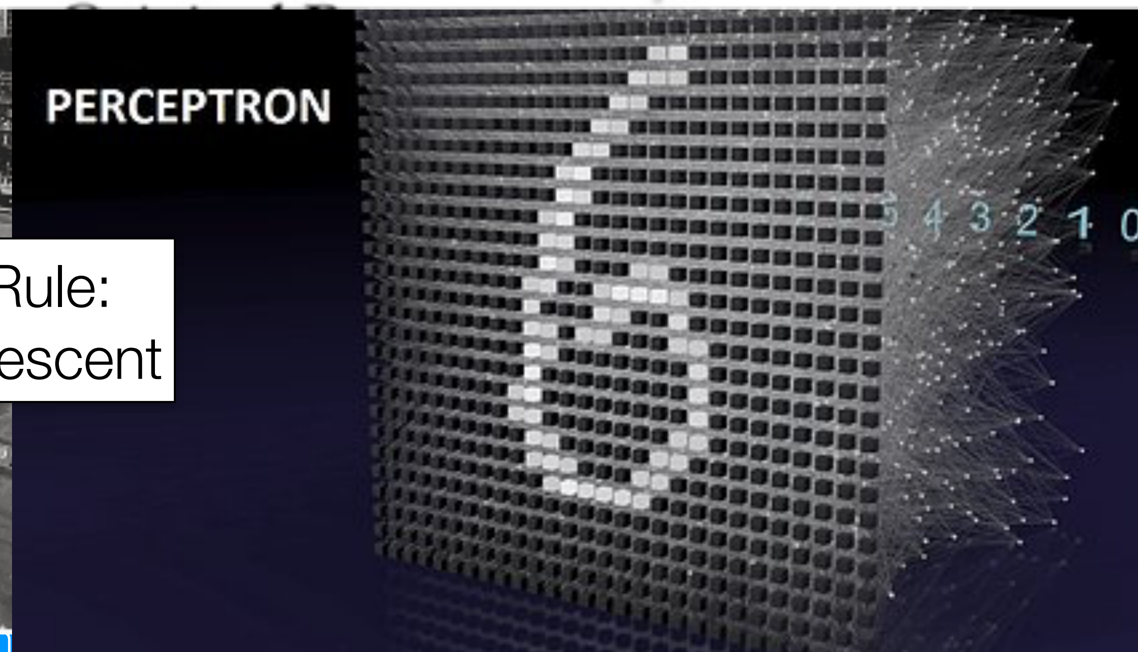
sigmoid

$$a=\frac{1}{1+\exp(-z)}$$

The Mark 1

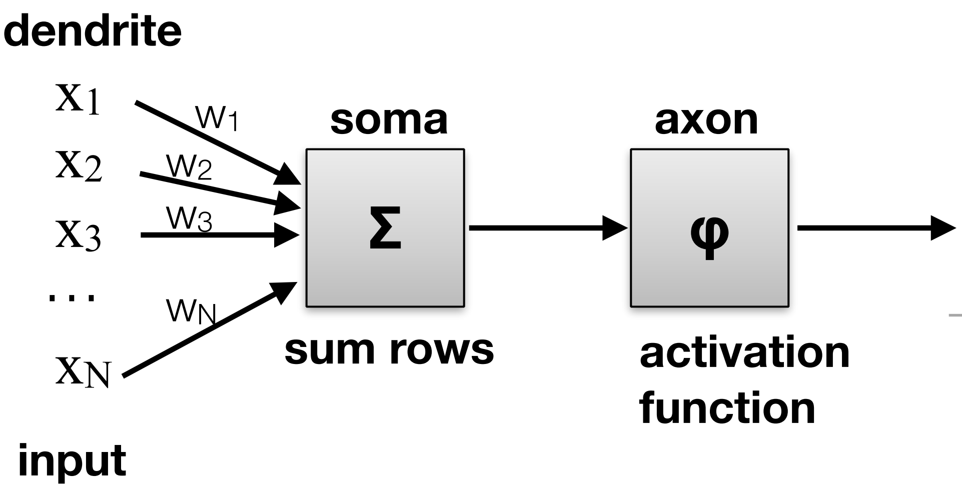






Perceptron Learning Rule:
~Stochastic Gradient Descent

Layers Notation

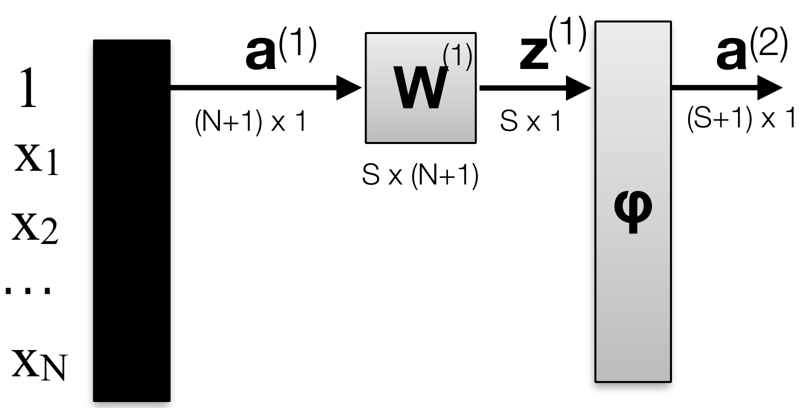


a=**x** with concat bias term

$$\mathbf{x}^{(i)} = \begin{bmatrix} x_1 \\ \vdots \\ x_j \\ \vdots \\ x_N \end{bmatrix}^{(i)} \qquad \mathbf{a} = \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_j \\ \vdots \\ x_N \end{bmatrix}$$

$$\mathbf{z} = \mathbf{W} \cdot \mathbf{a} = \mathbf{W}_{1:N} \cdot \mathbf{x}^{(i)} + \mathbf{b}$$

$$\mathbf{W} = \begin{bmatrix} \boxed{\begin{matrix} w_{1,0} \\ w_{2,0} \\ \vdots \\ w_{S,0} \end{matrix}} & \boxed{\begin{matrix} w_{1,1} & \dots & w_{1,N} \\ w_{2,1} & \dots & w_{2,N} \\ & & \\ w_{S,1} & \dots & w_{S,N} \end{matrix}} \end{bmatrix}$$



$$[\mathbf{z}^{(1)}]^{(i)} = \mathbf{W}^{(1)} \cdot [\mathbf{a}^{(1)}]^{(i)} = \mathbf{W}^{(1)}_{1:N} \cdot \mathbf{x}^{(i)} + \mathbf{b}^{(1)}$$

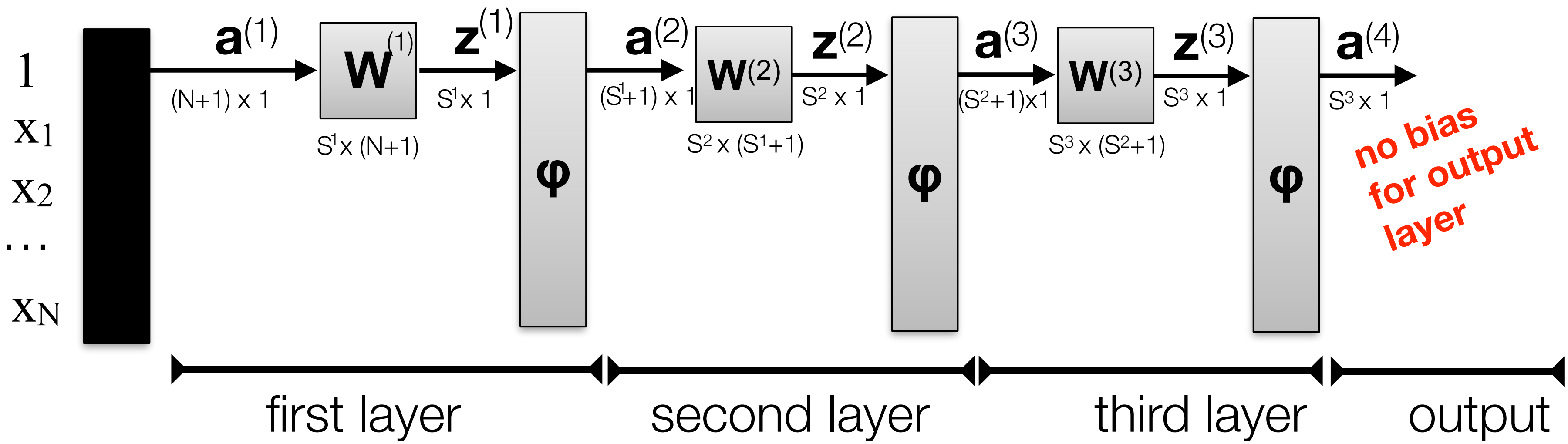
$$\mathbf{a}^{next} = \phi(\mathbf{z}^{\mathbf{current}}),$$ concat bias term

$$\mathbf{a}^{(next)} = \begin{bmatrix} 1 \\ \phi(z_1^{curr}) \\ \vdots \\ \phi(z_N^{curr}) \end{bmatrix} \rightarrow \mathbf{a}^{(L)} = \begin{bmatrix} 1 \\ \phi(z_1^{L-1}) \\ \vdots \\ \phi(z_N^{L-1}) \end{bmatrix}$$

$\mathbf{x}^{(i)}$ One row from Table data becomes input column to model

notation adapted from *Neural Network Design*, Hagan, Demuth, Beale, and De Jesus

Generic Multiple Layers Notation



$$\mathbf{a}^{(L+1)} = \phi(\mathbf{z}^{(L)}),$$ concat bias term

$\mathbf{a}^{(final)}$ size=unique classes

$$\mathbf{z}^{(L)} = \mathbf{W}^{(L)} \cdot \mathbf{a}^{(L)}$$

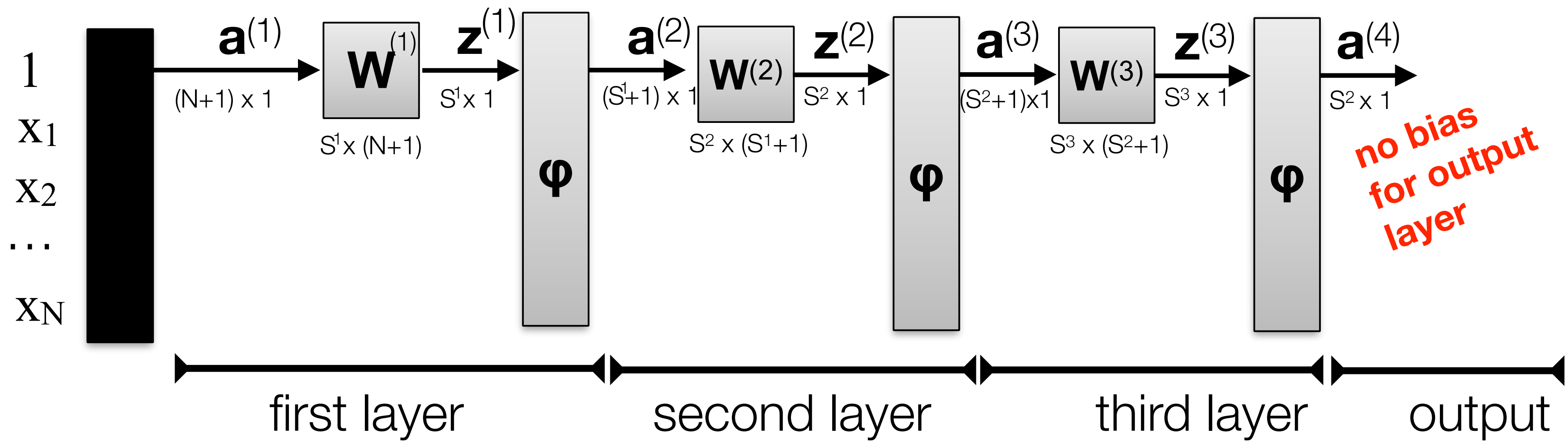
$$\mathbf{W} = \begin{bmatrix} w_{1,0} & w_{1,1} & \cdots & w_{1,N} \\ w_{2,0} & w_{2,1} & \cdots & w_{2,N} \\ \vdots & & & \\ w_{S,0} & w_{S,1} & \cdots & w_{S,N} \end{bmatrix}$$

$$\mathbf{z}^{(L)} = \mathbf{W}^{(L)} \cdot \phi(\mathbf{z}^{(L-1)})$$

$$\mathbf{z}^{(L)} = \mathbf{W}^{(L)} \cdot \phi(\mathbf{W}^{(L-1)} \cdot \phi(\mathbf{z}^{(L-2)}))$$

notation adapted from *Neural Network Design*, Hagan, Demuth, Beale, and De Jesus

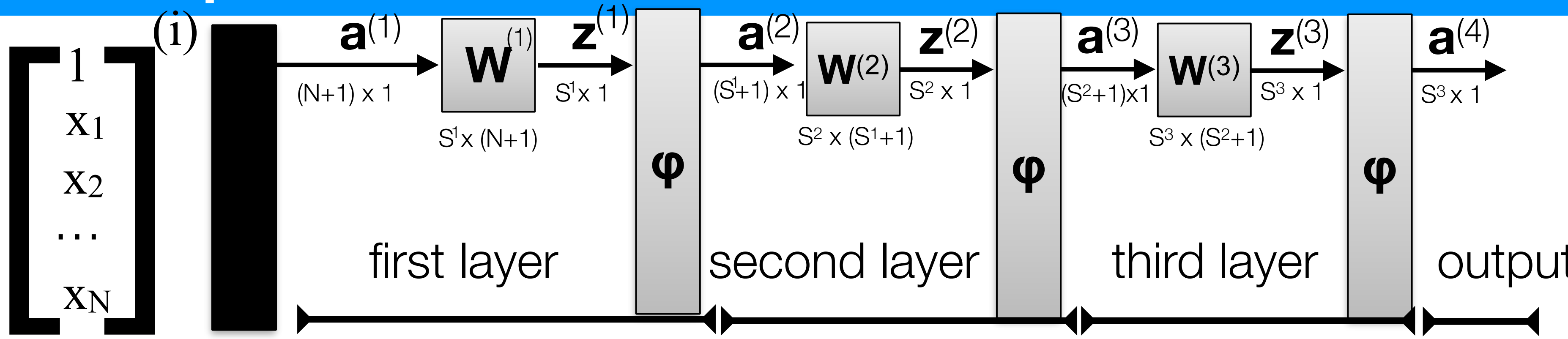
Multiple layers notation



- **Self test**: How many parameters need to be trained in the above network?
  - A. $[(N+1) \times S^1] + [(S^1+1) \times S^2] + [(S^2+1) \times S^3]$
  - B. $|\mathbf{W}^{(1)}| + |\mathbf{W}^{(2)}| + |\mathbf{W}^{(3)}|$
  - C. can't determine from diagram
  - D. it depends on the sizes of intermediate variables, $\mathbf{z}^{(i)}$

notation adapted from *Neural Network Design*, Hagan, Demuth, Beale, and De Jesus

Compact feedforward notation



$$\mathbf{z}^{(L)} = \mathbf{W}^{(L)} \cdot \mathbf{a}^{(L)}$$

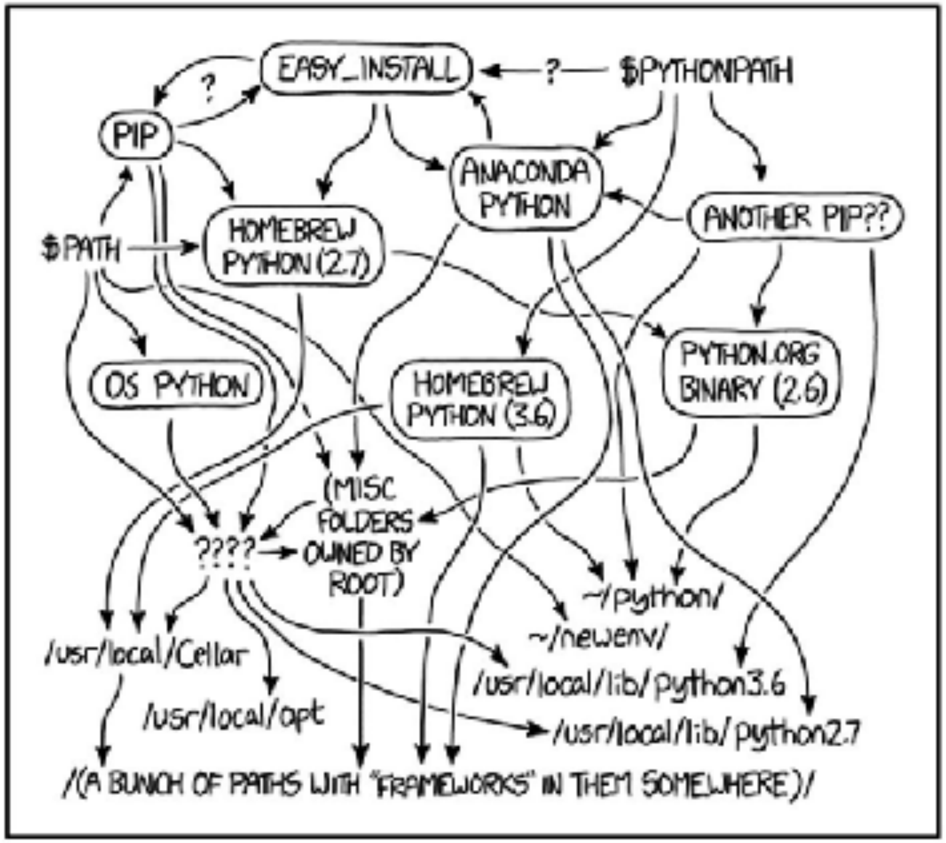
$$\left[\mathbf{z}^{(L)}\right]^{(i)} = \mathbf{W}^{(L)} \cdot \left[\mathbf{a}^{(L)}\right]^{(i)}$$

$$\begin{bmatrix} z_1^{(L)} \\ z_2^{(L)} \\ \vdots \\ z_{S^L}^{(L)} \end{bmatrix}^{(i)} = \mathbf{W}^{(L)} \cdot \begin{bmatrix} a_0^{(L)} \\ a_1^{(L)} \\ \vdots \\ a_{S^{L-1}}^{(L)} \end{bmatrix}^{(i)}$$

$$\left[\begin{bmatrix} z_1^{(L)} \\ z_2^{(L)} \\ \vdots \\ z_{S^L}^{(L)} \end{bmatrix}^{(1)}, \begin{bmatrix} z_1^{(L)} \\ z_2^{(L)} \\ \vdots \\ z_{S^L}^{(L)} \end{bmatrix}^{(2)} \cdots \begin{bmatrix} z_1^{(L)} \\ z_2^{(L)} \\ \vdots \\ z_{S^L}^{(L)} \end{bmatrix}^{(M)} \right] = \mathbf{W}^{(L)} \cdot \left[\begin{bmatrix} a_0^{(L)} \\ a_1^{(L)} \\ \vdots \\ a_{S^{L-1}}^{(L)} \end{bmatrix}^{(1)}, \begin{bmatrix} a_0^{(L)} \\ a_1^{(L)} \\ \vdots \\ a_{S^{L-1}}^{(L)} \end{bmatrix}^{(2)} \cdots \begin{bmatrix} a_0^{(L)} \\ a_1^{(L)} \\ \vdots \\ a_{S^{L-1}}^{(L)} \end{bmatrix}^{(M)} \right]$$

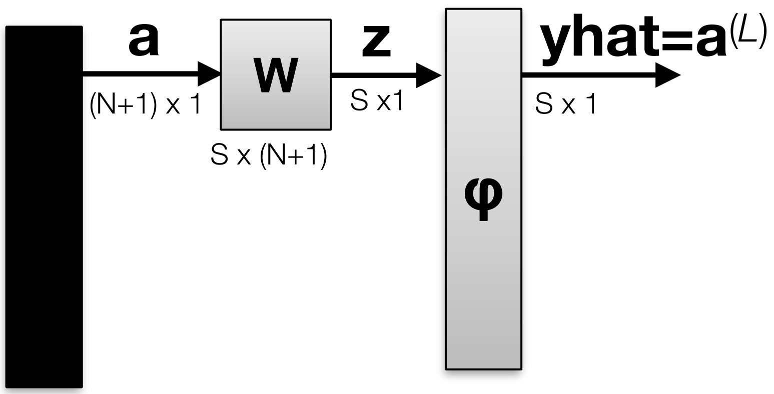
$$\mathbf{Z}^{(L)} = \mathbf{W}^{(L)} \cdot \mathbf{A}^{(L)} = \mathbf{W}^{(L)} \cdot \phi(\mathbf{Z}^{(L-1)})$$

Training Neural Network Architectures



MY PYTHON ENVIRONMENT HAS BECOME SO DEGRADED THAT MY LAPTOP HAS BEEN DECLARED A SUPERFUND SITE.

Start Simple: Simplifying to One Layer



where ground truth $\mathbf{Y}$ is one-hot encoded!

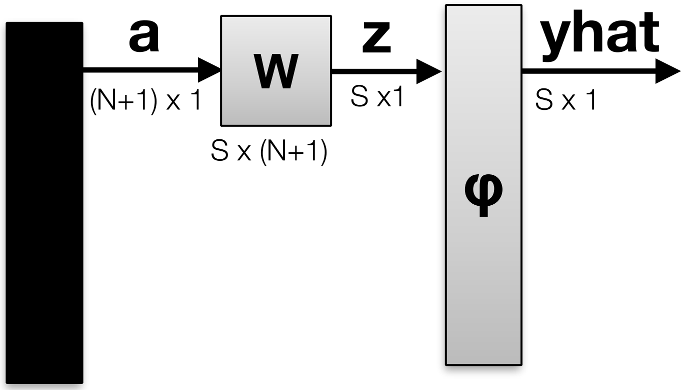
Need objective Function, minimize MSE

$$J(\mathbf{W}) = \left\| \mathbf{Y} - \hat{\mathbf{Y}} \right\|^2$$

$$J(\mathbf{W}) = \left| \underbrace{\begin{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_C \end{bmatrix}^{(1)} & \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_C \end{bmatrix}^{(2)} & \cdots & \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_C \end{bmatrix}^{(M)} \end{bmatrix}}_{\mathbf{Y}} - \underbrace{\begin{bmatrix} \begin{bmatrix} a_1^{(L)} \\ a_2^{(L)} \\ \vdots \\ a_C^{(L)} \end{bmatrix}^{(1)} & \begin{bmatrix} a_1^{(L)} \\ a_2^{(L)} \\ \vdots \\ a_C^{(L)} \end{bmatrix}^{(2)} & \cdots & \begin{bmatrix} a_1^{(L)} \\ a_2^{(L)} \\ \vdots \\ a_C^{(L)} \end{bmatrix}^{(M)} \end{bmatrix}}_{\hat{\mathbf{Y}}} \right|^2$$

Simple Architectures

- Rosenblatt's perceptron, 1957

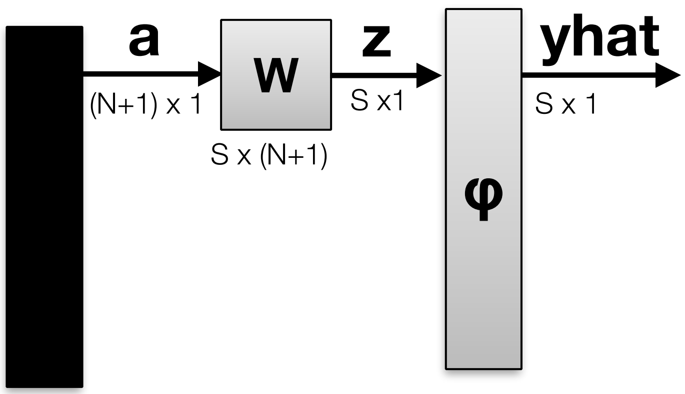


Self Test - If this is a binary classification problem, how large is *S*, the length of **yhat** and number of rows in **W**?

A. Can't determine
B. 2
C. 1
D. N

Simple Architectures

- Adaline network, Widrow and Hoff, 1960



linear

phi

Marcian “Ted” Hoff

Bernard Widrow

Simplify Objective Function:

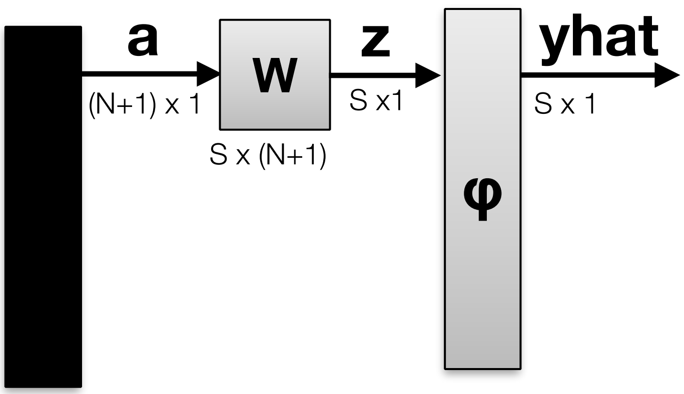
$$J(\mathbf{W}) = \left\| \mathbf{Y} - \hat{\mathbf{Y}} \right\|^2 \longrightarrow J(\mathbf{w}) = \left\| \mathbf{Y} - \mathbf{A} \cdot \mathbf{w} \right\|^2$$

Need gradient $\nabla J(\mathbf{w})$ for update equation $\mathbf{w} \leftarrow \mathbf{w} + \eta \nabla J(\mathbf{w})$

We have been using the **Widrow-Hoff Learning Rule**

Simple Architectures

- Adaline network, Widrow and Hoff, 1960



linear

phi

Marcian “Ted” Hoff

Bernard Widrow

Need gradient $\nabla J(\mathbf{w})$ for update equation $\mathbf{w} \leftarrow \mathbf{w} + \eta \nabla J(\mathbf{w})$

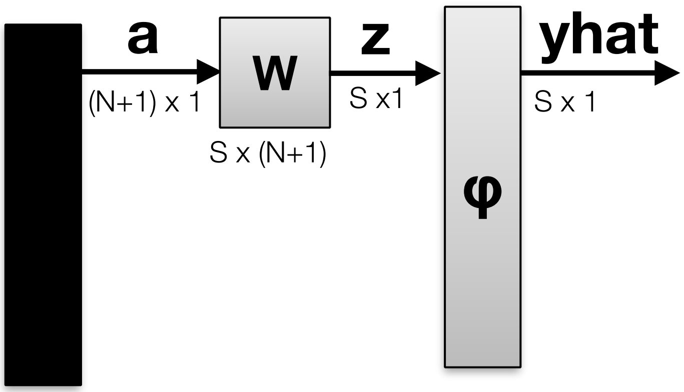
For case S=1, $\mathbf{W}$ has only one row, $\mathbf{w}$
this is just **linear regression…**

$$J(\mathbf{w}) = \sum_{i=1}^{M} (y^{(i)} - \mathbf{x}^{(i)} \cdot \mathbf{w})^2$$

$$\mathbf{w} = (\mathbf{X}^T \cdot \mathbf{X})^{-1} \cdot \mathbf{X}^T \cdot y$$

Simple Architectures

- Modern Perceptron network



sigmoid

$$g(z) = \phi(z) = \frac{1}{1 + \exp(-z)}$$

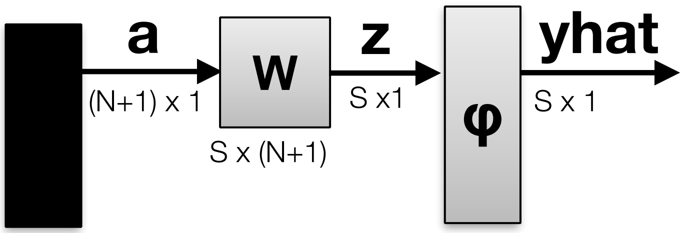
Need gradient $\nabla J(\mathbf{w})$ for update equation $\mathbf{w} \leftarrow \mathbf{w} + \eta \nabla J(\mathbf{w})$

For case S=1, this is just **logistic regression…**
and **we have already solved this**!

$$\mathbf{w} \leftarrow \mathbf{w} + \eta(\mathbf{y} - g(\mathbf{X} \cdot \mathbf{w})) \odot \mathbf{X}$$

What happens when S > 1 ?

What if we have more than S=1?



$$J(\mathbf{W}) = \left\| \mathbf{Y} - \hat{\mathbf{Y}} \right\|^2$$

$$J(\mathbf{W}) = \left\| \mathbf{Y} - \phi(\mathbf{W} \cdot \mathbf{X}) \right\|^2$$

$$\boxed{J(\mathbf{w}_{row=1}) = \sum_i (y_1^{(i)} - \phi(\mathbf{x}^{(i)} \cdot \mathbf{w}_{row=1}))^2}$$

$$\cdots$$

$$\boxed{J(\mathbf{w}_{row=C}) = \sum_i (y_C^{(i)} - \phi(\mathbf{x}^{(i)} \cdot \mathbf{w}_{row=C}))^2}$$

$$\mathbf{Y} = \left[\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_C \end{bmatrix}^{(1)} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_C \end{bmatrix}^{(2)} \cdots \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_C \end{bmatrix}^{(M)} \right] \rightarrow \left[\begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}^{(1)} \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}^{(2)} \cdots \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}^{(M)} \right]$$

Each target class in **Y** can be independently optimized

$$\hat{\mathbf{Y}} = \left[\begin{bmatrix} \phi(\mathbf{x}^{(1)} \cdot \mathbf{w}_{row=1}) \\ \phi(\mathbf{x}^{(1)} \cdot \mathbf{w}_{row=2}) \\ \vdots \\ \phi(\mathbf{x}^{(1)} \cdot \mathbf{w}_{row=C}) \end{bmatrix} \begin{bmatrix} \phi(\mathbf{x}^{(2)} \cdot \mathbf{w}_{row=1}) \\ \phi(\mathbf{x}^{(2)} \cdot \mathbf{w}_{row=2}) \\ \vdots \\ \phi(\mathbf{x}^{(2)} \cdot \mathbf{w}_{row=C}) \end{bmatrix} \cdots \begin{bmatrix} \phi(\mathbf{x}^{(M)} \cdot \mathbf{w}_{row=1}) \\ \phi(\mathbf{x}^{(M)} \cdot \mathbf{w}_{row=2}) \\ \vdots \\ \phi(\mathbf{x}^{(M)} \cdot \mathbf{w}_{row=C}) \end{bmatrix} \right]$$

which is one versus-all!

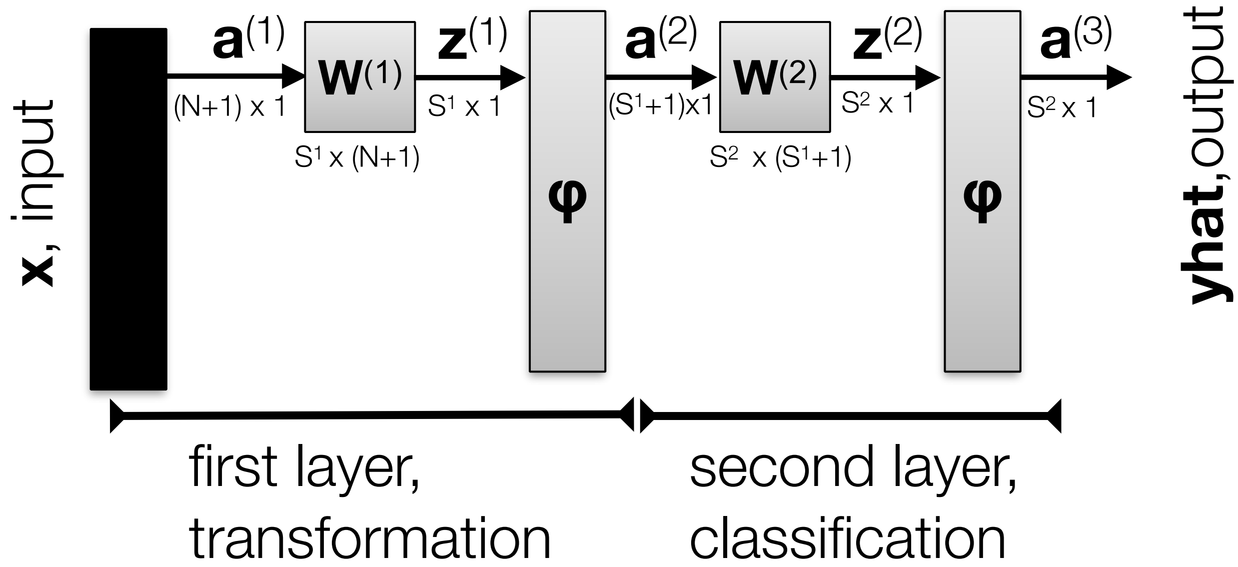
Early Architectures: Summary

- Adaline network, Widrow and Hoff, 1960
  - linear regression, iterative updates
- Perceptron
  - *with sigmoid*: logistic regression
- One-versus-all implementation is the same as having $\mathbf{w}_{class}$ be rows of weight matrix, $\mathbf{W}$

- **But what about when we have more than one layer?**

Moving to multiple layers…

- The multi-layer perceptron (MLP):
  - two layers shown, but could be arbitrarily many layers



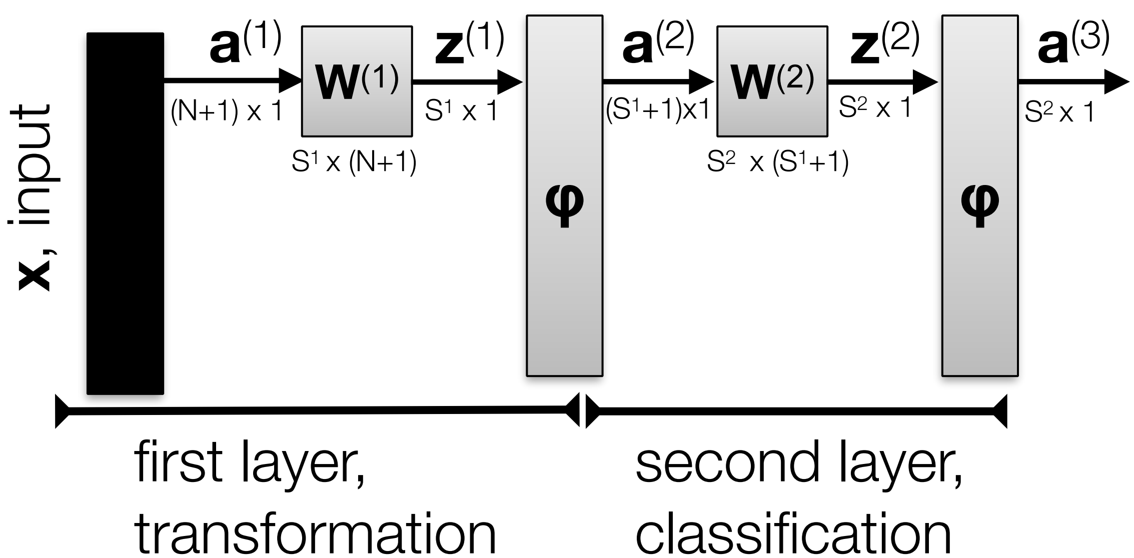
each row of **yhat** is no longer independent of the rows in early **W** so we cannot optimize using one versus all 😢

$$\begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_C \end{bmatrix} = \begin{bmatrix} \phi(\phi(\mathbf{z}^{(1)}) \cdot \mathbf{w}^{(2)}_{row=1}) \\ \phi(\phi(\mathbf{z}^{(1)}) \cdot \mathbf{w}^{(2)}_{row=2}) \\ \vdots \\ \phi(\phi(\mathbf{z}^{(1)}) \cdot \mathbf{w}^{(2)}_{row=C}) \end{bmatrix}$$

$$\boxed{\mathbf{z}^{(1)} = \mathbf{W}^{(1)} \cdot \mathbf{a}^{(1)}}$$

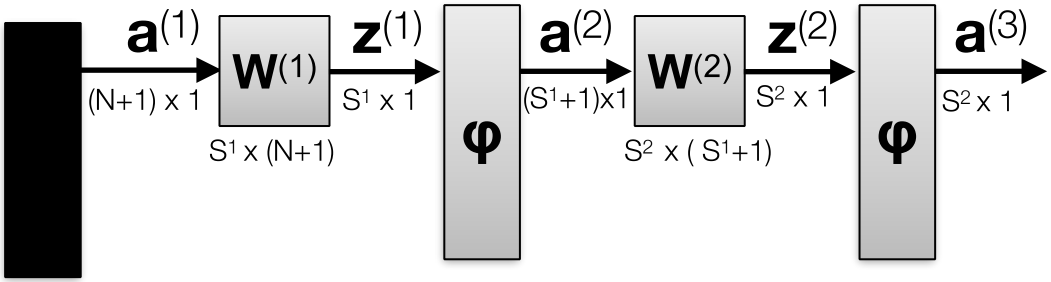
Back propagation

- Optimize all weights of network at once
- Steps:
  - propagate weights forward
  - calculate gradient at final layer
  - back propagate gradient for each layer
    - via recurrence relation



Back-propagation is solved in flipped assignment!!

Back propagation Preview



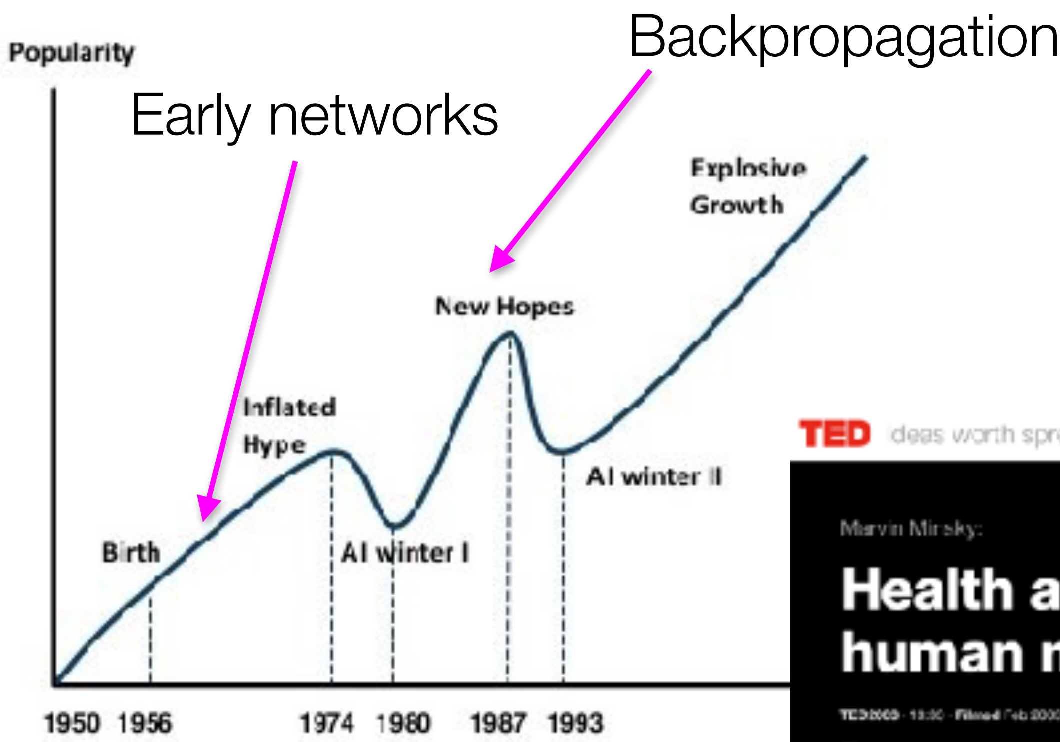
$$w_{i,j}^{(l)} \leftarrow w_{i,j}^{(l)} - \eta \boxed{\frac{\partial J(\mathbf{W})}{\partial w_{i,j}^{(l)}}}$$

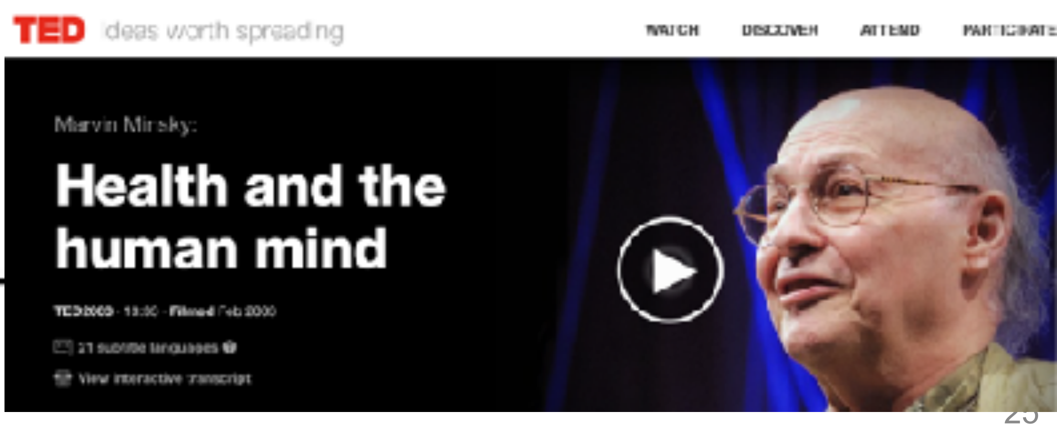
use chain rule: $$\frac{\partial J(\mathbf{W})}{\partial w_{i,j}^{(l)}} = \boxed{\frac{\partial J(\mathbf{W})}{\partial \mathbf{z}^{(l)}}} \frac{\partial \mathbf{z}^{(l)}}{\partial w_{i,j}^{(l)}}$$

use chain rule again: $$\frac{\partial J(\mathbf{W})}{\partial \mathbf{z}^{(l)}} = \frac{\partial J(\mathbf{W})}{\partial \mathbf{z}^{(l+1)}} \frac{\partial \mathbf{z}^{(l+1)}}{\partial \mathbf{z}^{(l)}}$$

This is solved in explainer video for next flipped assignment!

The First AI Winter (if time)





The Rosenblatt-Widrow-Hoff Dilemma

- 1960's: Rosenblatt got into a public academic argument with Marvin Minsky and Seymour Papert

> "Given an elementary α-perceptron, a stimulus world W, and any classification C(W) for which a solution exists; let all stimuli in W occur in any sequence, provided that each stimulus must reoccur in finite time; then beginning from an arbitrary initial state, an error correction procedure will always yield a solution to C(W) in finite time…"

- Minsky and Papert publish limitations paper, 1969:

> "the style of research being done on the perceptron is doomed to failure because of these limitations."

- Widrow and Rosenblatt try to build bigger networks without limitations and fail
  - Neural Networks research **basically stops** for **17 years**
- **Until**: researchers revisit training bigger networks
  - Neural Networks with multiple layers

Stable Training of Multi-layer Architectures: history

- 1986: *Rumelhart*, *Hinton*, and *Williams* popularize gradient calculation for multi-layer network
  - *technically* introduced by Werbos in 1982
- **difference**: Rumelhart *et al.* validated ideas with a computer
- until this point no one could train a multiple layer network consistently
- algorithm is popularly called **Back-Propagation**
- wins pattern recognition prize in 1993, becomes de-facto machine learning algorithm until: SVMs and Random Forests in ~2004
- would eventually see a resurgence for its ability to train algorithms for Deep Learning applications: **Hinton is widely considered the founder of deep learning**

David Rumelhart

Geoffrey Hinton

End of Session

- thanks! **Next time is Flipped Assignment!!!**

More help on neural networks to prepare for next time:

Sebastian Raschka

https://github.com/rasbt/python-machine-learning-book/blob/master/code/ch12/ch12.ipynb

Martin Hagan

https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&cad=rja&uact=8&ved=0ahUKEwioprvn27fPAhWMx4MKHYbwDIwQFggeMAA&url=http%3A%2F%2Fhagan.okstate.edu%2FNNDesign.pdf&usg=AFQjCNG5YbM4xSMm6K5HNsG-4Q8TvOu_Lw&sig2=bgT3k-5ZDDTPZ07Qu8Oreg

Michael Nielsen

http://neuralnetworksanddeeplearning.com