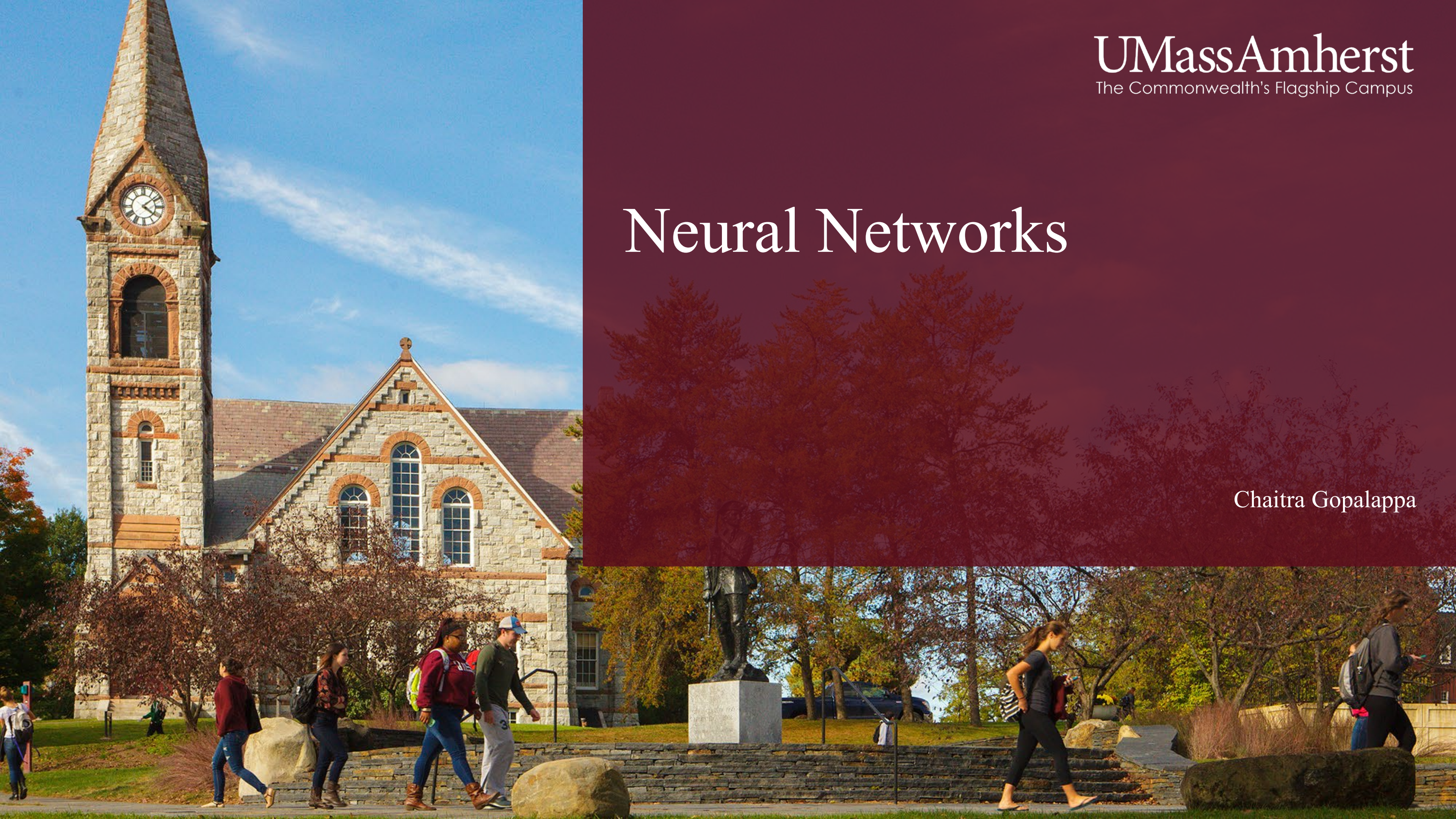


UMassAmherst
The Commonwealth's Flagship Campus

Neural Networks

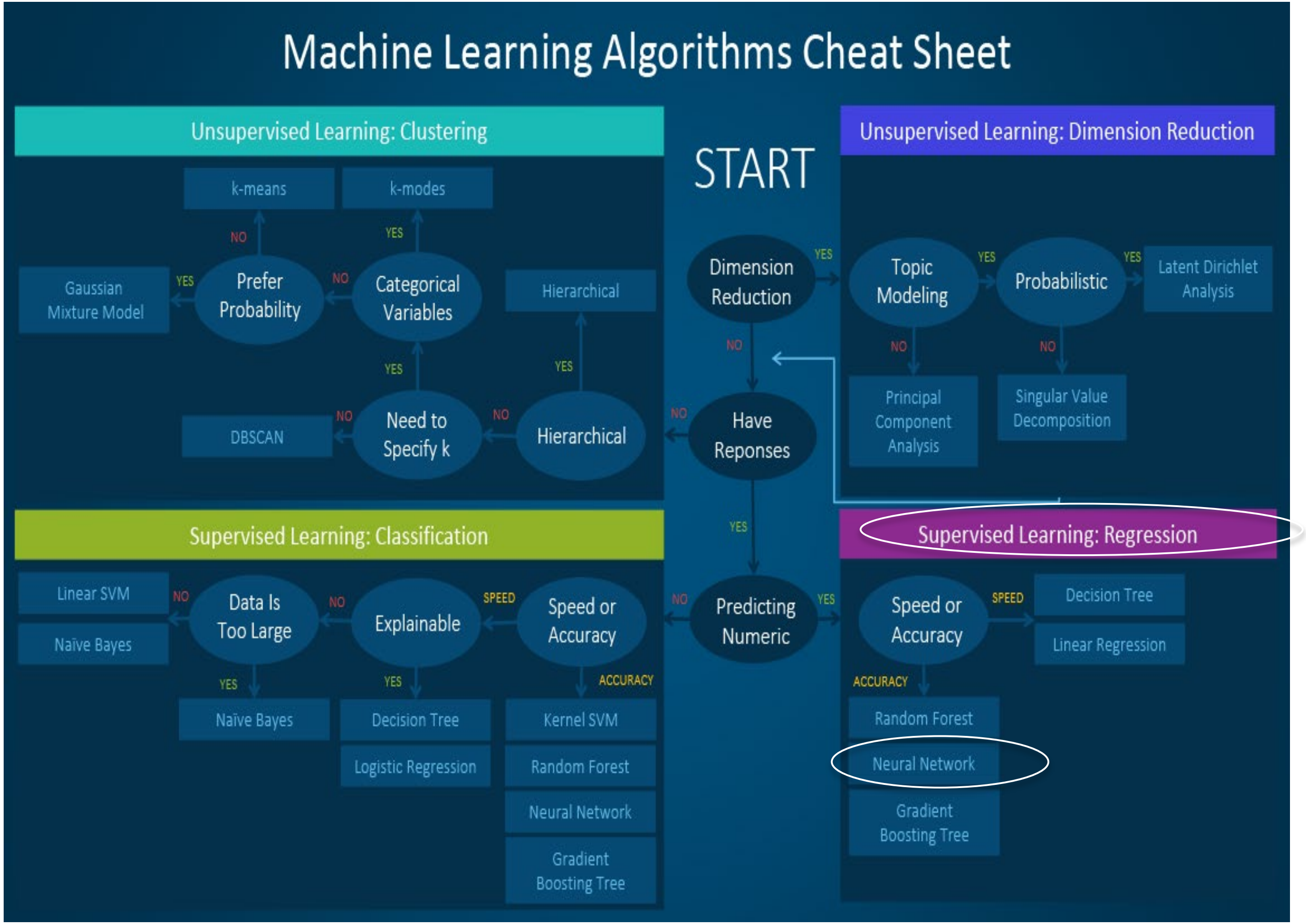
Chaitra Gopalappa

Function Approximation/Fitting

- Widrow-Hoff works if functional form is known (model-based)
- What if function form is not know?
  - Use Neural Networks (model-free)

Neural networks

- Type of machine learning method
- Dates back to 1940’s
- Nice read of the history 1940s to 2000: https://cs.stanford.edu/people/eroberts/courses/soco/projects/neural-networks/History/history1.html
- Foundational work leading to current deep learning



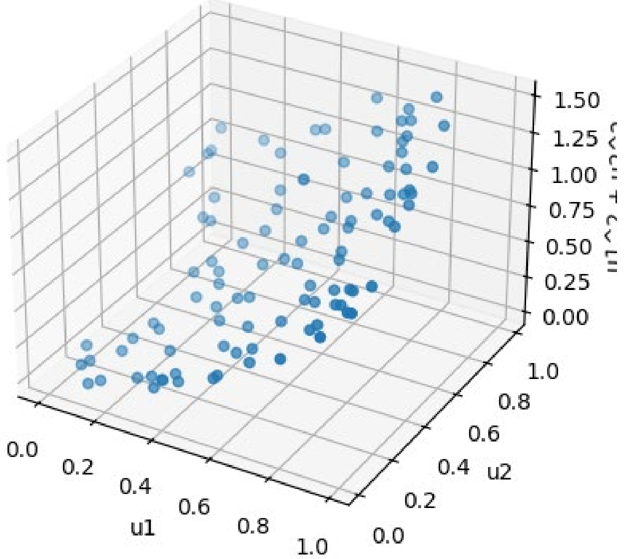
Source: https://blogs.sas.com/content/subconsciousmusings/2020/12/09/machine-learning-algorithm-use/

- In this class we will use Deep Learning/ Neural Network for Function Approximation (sometimes referred to as regression, though regression is one of the methods for function approximation)
- Deep Learning is a supervised Machine Learning method
  - Estimating a function that fits to the data

BASIC NN

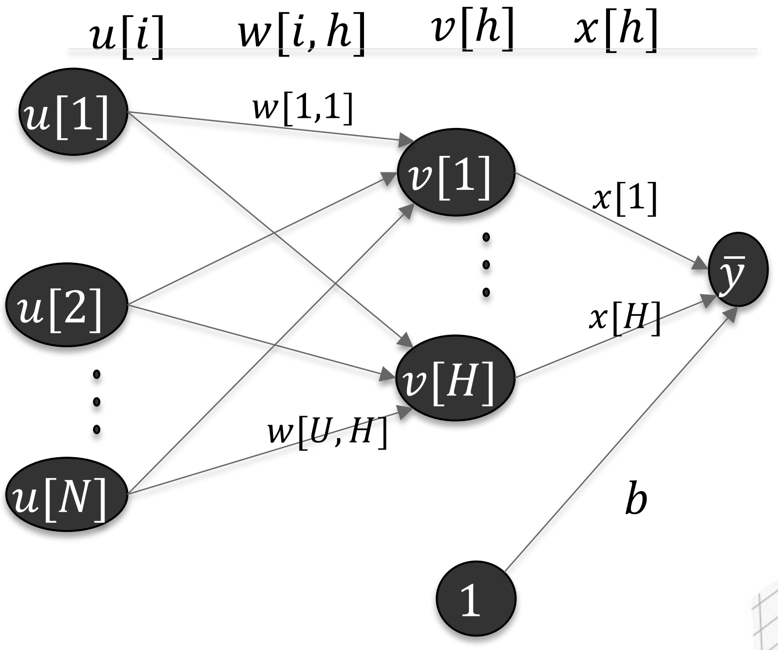
Nonlinear Neural Network (model- free):

- **Problem**: Fit a predictive model to the data. Suppose we do not know the analytical form of the function, but only know that $y = f(u_1, u_2, \ldots.)$

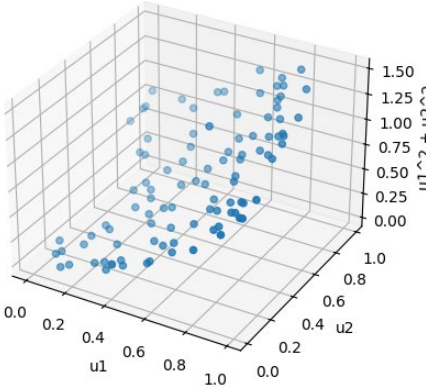


- Example $y = f(u_1, u_2)$

Neural Network (NN) architecture



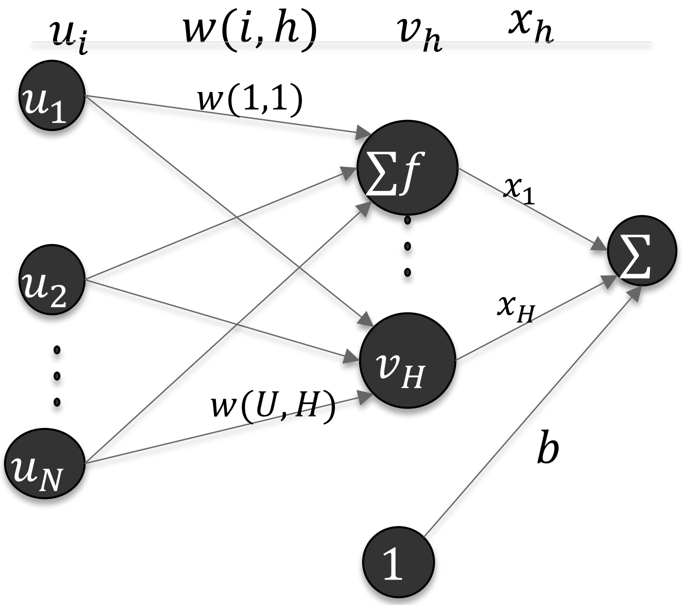
- $u[i]$ are the input nodes (independent variables of the function)
- $w[i,h]$and $x[h]$ are the weights (equivalent to coefficients in polynomial equations) of the NN
  - $w[i,h]$: are the weights of the links from node $u[i]$to node v[h]
  - x[h]: are the weights of the links from node v[h] to $\bar{y}$ (¯ (bar) to denote that it is an estimate of y)
- b is a bias node (equivalent to the intercept) term
- $v[h]$: are the nodes in the hidden layer and mathematically represented by **activation** functions (in below example we use a sigmoid function). **CORE of NN: adds non-linearity**



Neural Network terminologies

- Foundations are in linear algebra-All data are in arrays (vectors, matrices, 3D , ..)
- Most times notations not written in vector form (as on the right side figure)
- Will write in vector or matrix form on these slides to understand the foundations

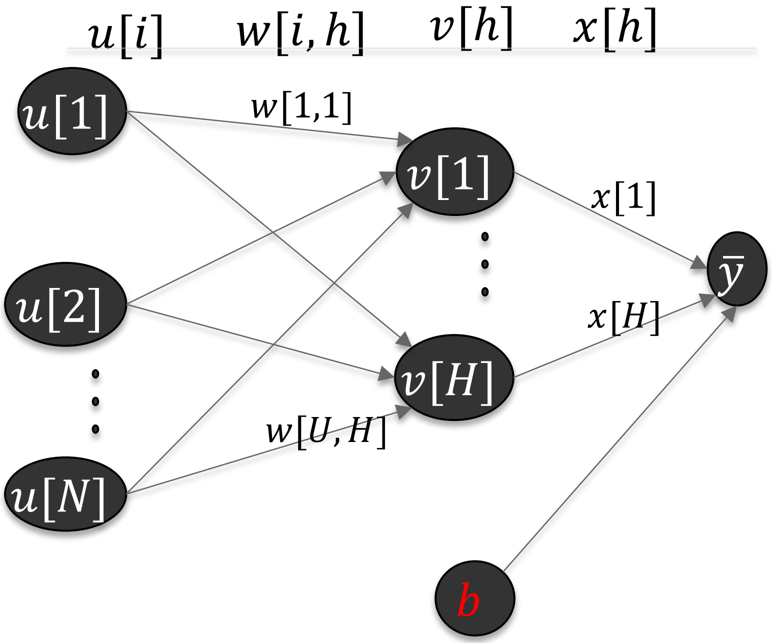
Suppose $y = f(\vec{u})$;
Corresponding NN architecture for feedforward NN



Neural Network terminologies

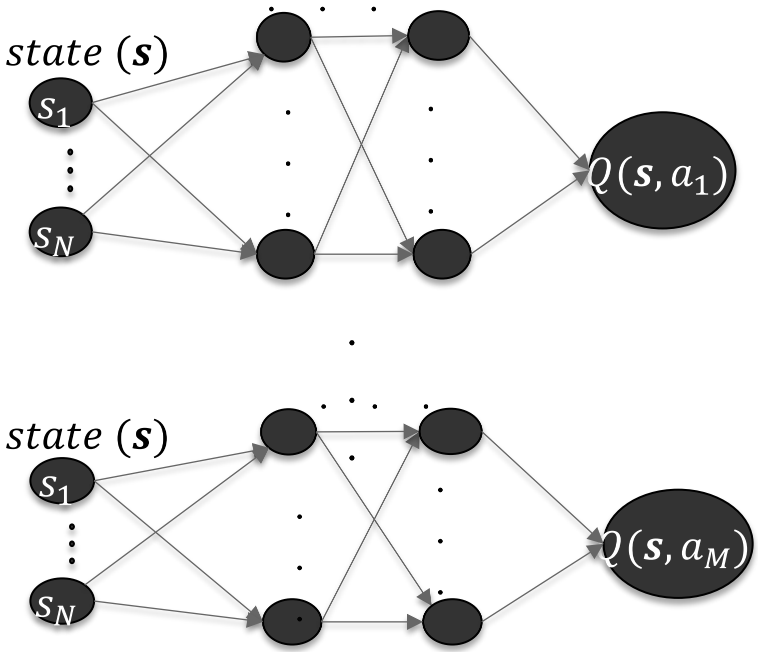
- NN architecture
  - Sometimes b written in the node, with nothing on arrow

Suppose $y = f(\vec{u})$;
Corresponding NN architecture for feedforward NN

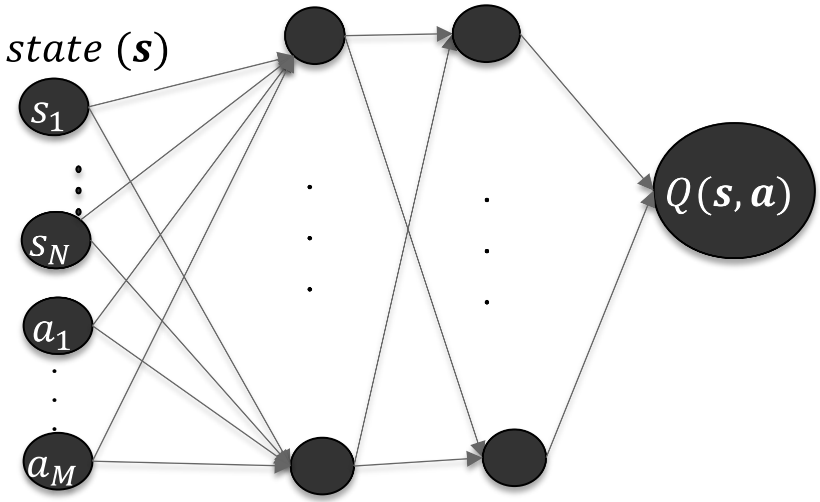


Example architectures - taking Q values

Example 1: One network for each action



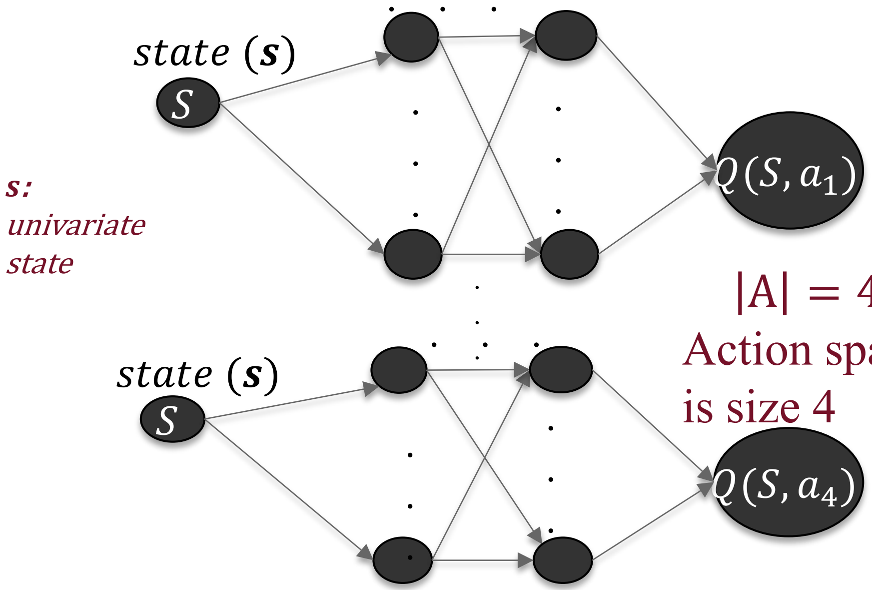
Example 2: Include action in input layer



$\boldsymbol{s}$: *state vector* $\boldsymbol{a}$: *action vector*

$\boldsymbol{s} = [s_1, \dots s_N]$ $\boldsymbol{a} = [a_1, \dots a_M]$

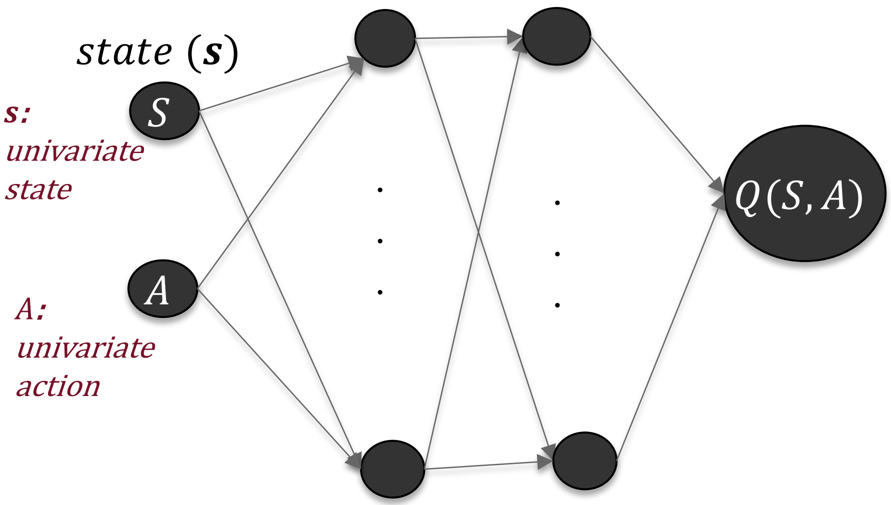
Example of uni-variate state and action



$|\mathrm{A}| = 4$

Action space is size 4

Include action in input layer



Suppose,

X_t =proportion of people with active infection

D_t =how often to test

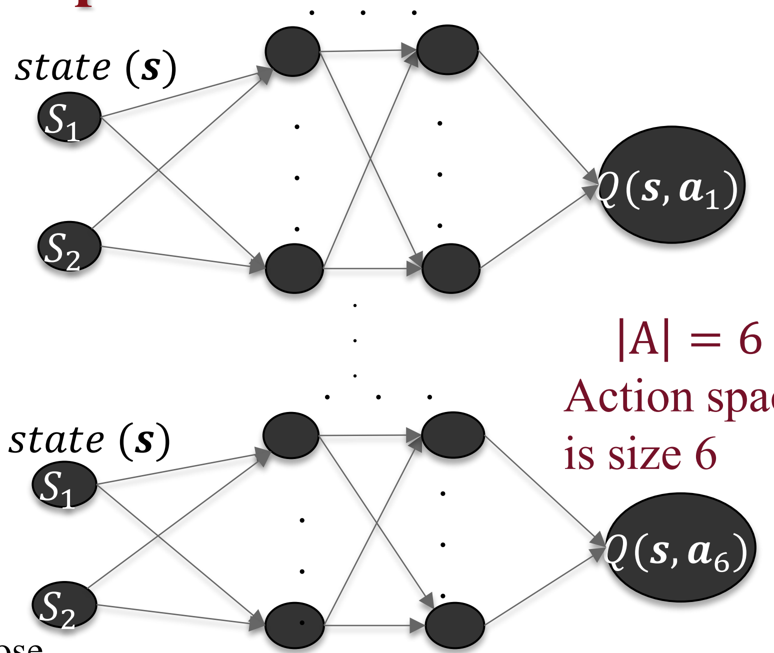
State space: $\mathrm{S} \in \mathbb{R}^1; \mathrm{S} \in [0,1]$

Action space:A = {test once a week, twice a week, three times a week, daily}

Action space:$\mathrm{A} \in \mathbb{R}^1; A \in [1,30]$

Example of multi-variate state and action

s: vector of 2 variates



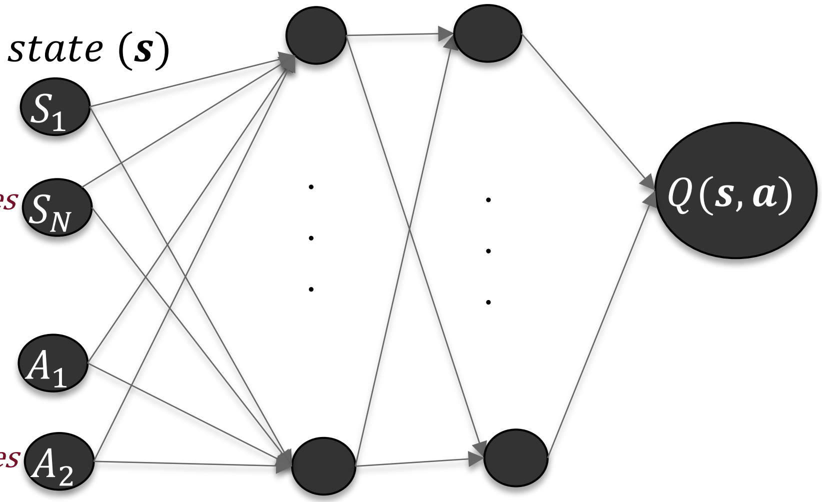
$|\mathrm{A}| = 6$
Action space is size 6

Suppose,
X_t =[proportion with active infection, proportion recovered]
D_t =[how often to test in a week, what %lockdown]
State space: $\mathrm{S} \in \mathbb{R}^2$; (vector of 2 real random variables)
Action space:A ={[once, 25%], [twice,25%], [thrice, 25%], [once, 50%], [twice,50%], [thrice, 50%] }

Include action in input layer

s: vector of 2 variates

a: vector of 2 variates



Action space: $\boldsymbol{a} \in \mathbb{R}^2$; (vector of 2 real random variables)

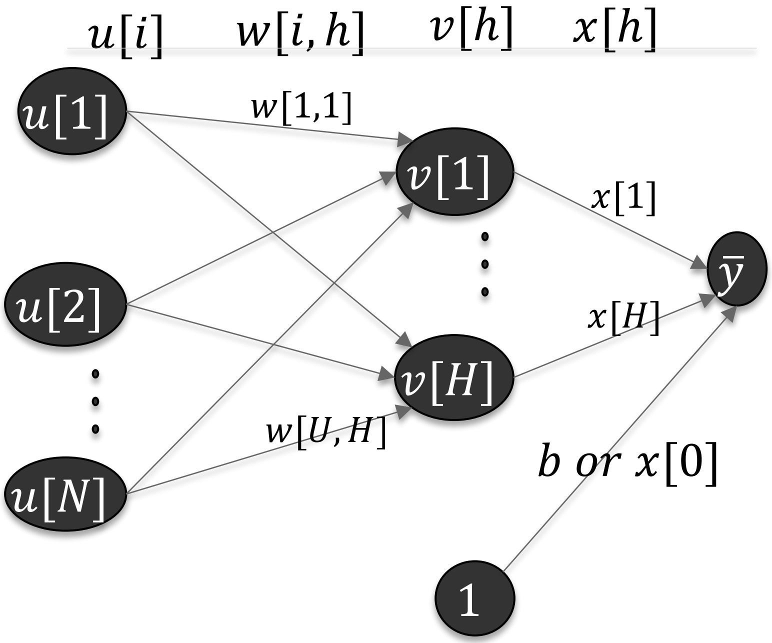
***s**: state vector*
$\boldsymbol{s} = [S_1, \dots S_N]$

***a**: action vector*
$\boldsymbol{a} = [A_1, \dots A_M]$

Neural Network terminologies

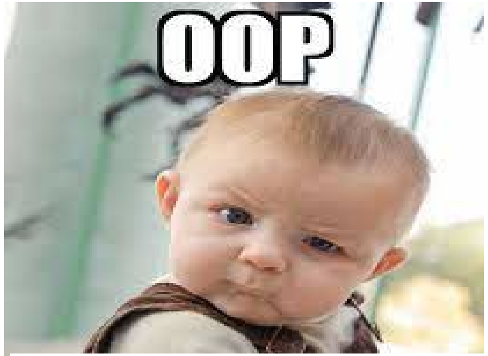
- NN architecture
  - Output layer ($\bar{\boldsymbol{y}}$ a vector) $\bar{\boldsymbol{y}} = f(\vec{u})$
    - Figure shows univariate output ($\bar{y}$)
  - Input layer ($\vec{u}$)
    - Input nodes ($u[i]$)
    - Number of nodes in input layer = $|\vec{u}|$
  - Hidden layer/s (figure shows one hidden layer $\vec{v}$)
  - Weights (arrows; $w[.,.], x[.]$)
  - Bias node (1 or b)
  - Activation function/s (v)
    - Non-linear transformation applied to linear combination of $w[.,.]$, $u[.]$)
- Loss-function
  - Objective function (minimize loss/error)
- NN optimizer
  - Algorithm to train the NN: solve for $w[.,.], x[.]$ to minimize loss

Suppose $y = f(\vec{u})$;
Corresponding NN architecture for feedforward NN



Guess what?

- **Pay attention to the variables**; they typically change (get into the habit of deviating from fixed variables)
- Here we are using u for independent variables (in W-H they were x);
- **Always define the variables for every problem**, do not expect the reader to guess; they always guess wrong



YOU ASKED ME TO GUESS;
AND I GUESSED WRONG

Neural Network for Function Fitting

- Objective for function fitting:

$$\min_{b,\boldsymbol{x},\boldsymbol{w}} E = \min_{b,\boldsymbol{x},\boldsymbol{w}} \sum_{p=1}^{P} \left(y_p - \bar{y}_p\right)^2$$

We can then apply Gradient Descent

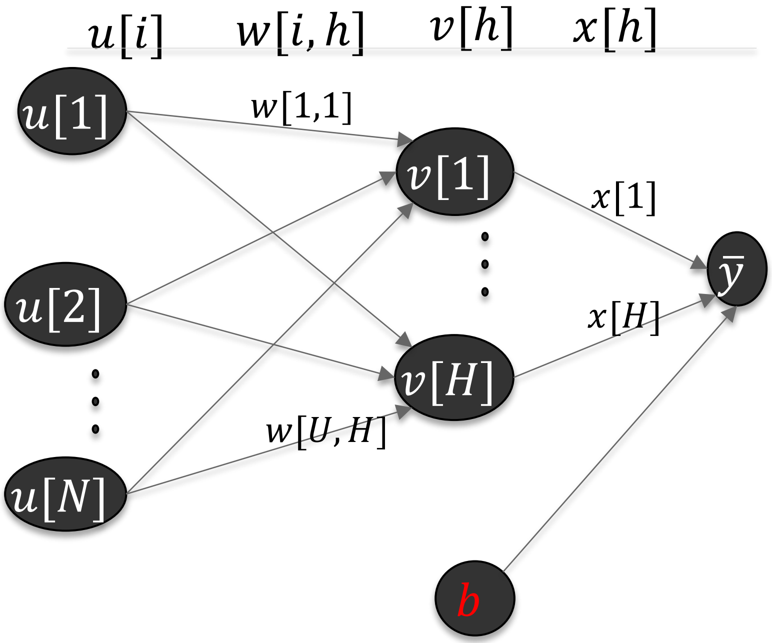
$$b \leftarrow b - \mu \frac{\partial(E)}{\partial b}$$

$$\omega[i,h] \leftarrow \omega[i,h] - \frac{\mu\partial(E)}{\partial\omega[i,h]}$$

$$x[h] \leftarrow x[h] - \mu \frac{\partial(E)}{\partial x[h]}$$

$\bar{y} = ?$

Suppose $y = f(\vec{u})$;

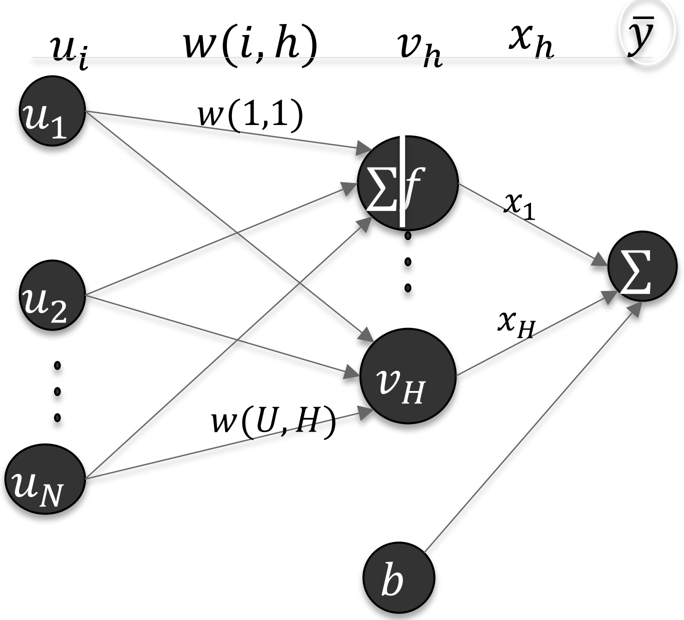


PROBLEM FORMULATION (mathematical representation of NN)

- We can mathematically represent the neural net with the following equations.
- $\bar{y} = b + \sum_{h=1,2,\ldots H} x[h]v[h]$
- $v[h]$ is a non-linear transformation applied to linear combination of $w[.,h], u[.]$
- $v[h] = f\left((\sum_{i=1:\mathrm{N}} w[i,h]u[i])\right)$
- $v[h] = f(w[.,h]u[.])$
- Suppose we use sigmoid activation function, then,

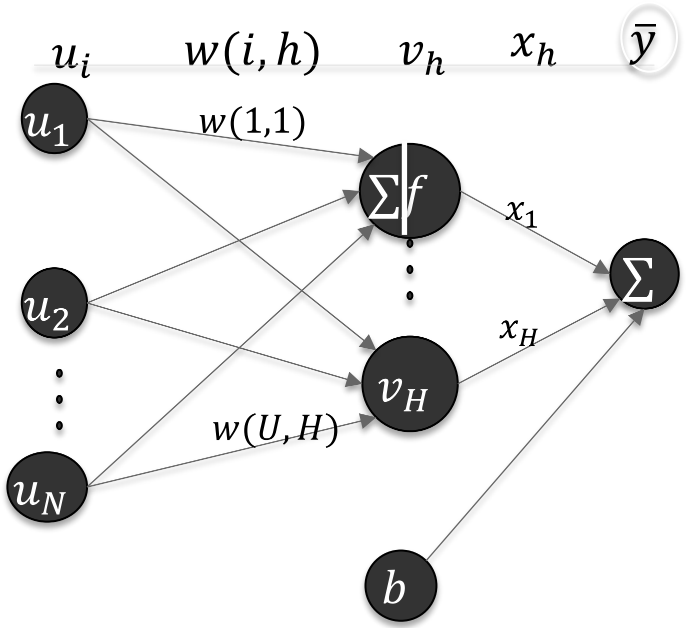
  $v[h] = \frac{1}{1+e^{-v^*[h]}}\}; \forall h$, where

  $v^*[h] = \sum_{i=1:\mathrm{N}} w[i,h]u[i]$



PROBLEM FORMULATION – with SIGMOID ACTIVATION

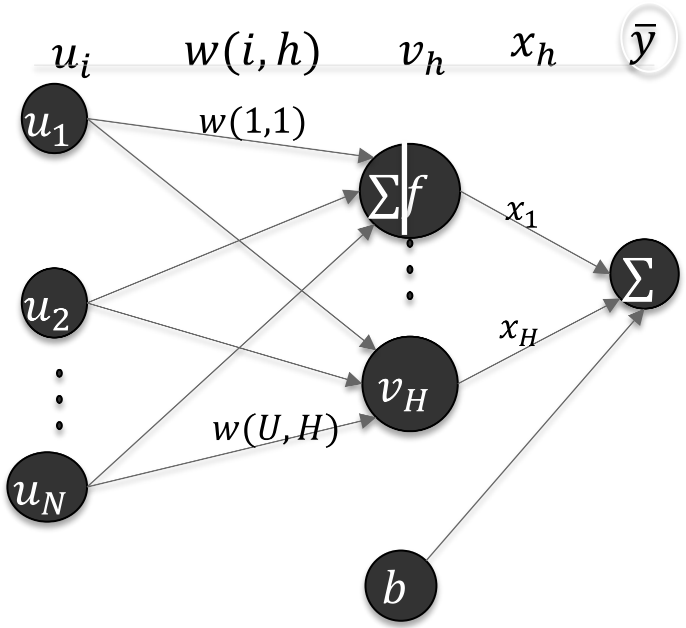
- $\bar{y} = b + \sum_{h=1,2,\ldots H} \; x[h]v[h]$
- $v[h] = \frac{1}{1+e^{-v^*[h]}} \Big\}; \forall h$, where
- $v^*[h] = \sum_{i=1:\mathrm{N}} \; w[i,h]u[i]$



PROBLEM FORMULATION (SIGMOID ACTIVATION)

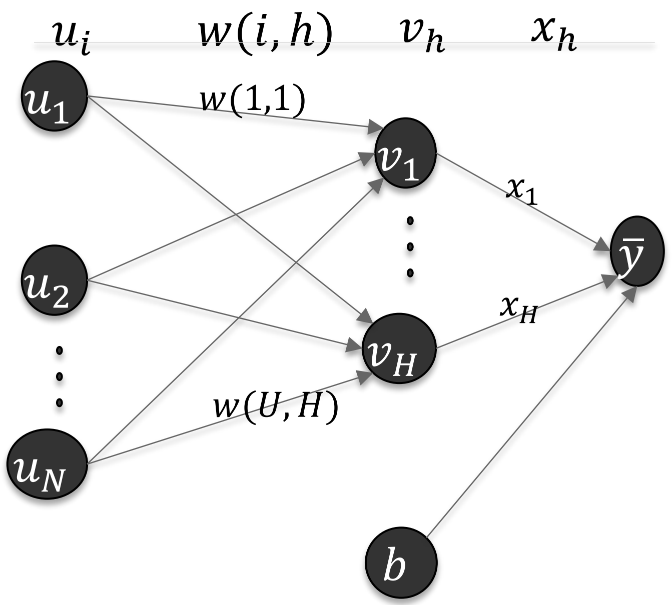
- $\bar{y} = b + \sum_{h=1,2,\ldots H} \; x[h]v[h]$
- $v[h] = \frac{1}{1+e^{-v^*[h]}}\Big\}; \forall h$, where
- $v^*[h] = \sum_{i=1:\mathrm{N}} \; w[i,h]u[i]$

- Objective is to $\min_{b,\boldsymbol{x},\boldsymbol{w}} SSE = \sum_{p=1}^{P} \left(y_p - \bar{y}_p\right)^2$



SOLUTION ALGORITHM: Backprop (back propagation)

- **Backprop concepts**
- Objective is to $\min_{b,\boldsymbol{x},\boldsymbol{w}} SSE \sim \min_{b,\boldsymbol{x},\boldsymbol{w}} \frac{SSE}{2}$
  - $SSE = f(b, \omega[.,.], x[.])$ (Notice this doesn't have u because they are data samples)
  - $b, \omega[.,.]$, and $x[.]$ are the decision variables from perspective of Backprop.
  - $\bar{y}_p$ = output from trained net for $\vec{u}_p$; p are the data samples
  - y_p = actual data
  - P = number of samples; N = number of input nodes



Backprop algorithm

1. Initialize
   1. Initialize $b, \omega(.,.)$, and $x(.)$ to random values
   2. Set SSE_{old} to very large value.
   3. Set $m = 0$ (iteration number)
2. Compute $v_p^*[h] = \sum_i w[i,h]u_p[\mathrm{i}]$ for each p and each h.
3. Compute $v_p[h] = \frac{1}{1+e^{-v_p^*[h]}}$ for each p and each h
4. Compute $\bar{y}_p = b + \sum_h x[h]v_p[h]$ for each p (data samples)

Feed forward

5. Apply SD transformations $(\mathrm{E} = \frac{1}{2}\sum_{p=1}^{P}(y_p - \bar{y}_p)^2)$

$$b_m \leftarrow b_{m-1} - \mu\frac{\partial(E)}{\partial b}$$

6.
$$\omega_m[i,h] \leftarrow \omega_{m-1}[i,h] - \mu\frac{\partial(E)}{\partial\omega[i,h]}$$

$$x_m[h] \leftarrow x_{m-1}[h] - \mu\frac{\partial(E)}{\partial x[h]}$$

7. Set $m = m + 1$
   1. Calculate $E_{new} = \sum_p (y_p - \bar{y}_p)^2$
   2. Update μ
   3. If $[E_{\text{new}} - E_{\text{old}}] <$ tolerance $\rightarrow$ *STOP* Otherwise set $E_{\text{old}} = E_{\text{new}}$ got step 2.

Recollect W-H for regression-uses Steepest Descent (SD)

- Initialize
  - Set $w[i]$ to values between 0 and 1;
  - Set E_{old} (the SSE) to a large number
  - Set $m = 0$; Set μ
- Compute $\bar{y}_p = \sum_{i=0}^{N}\omega[i]x_p[i]$ for each p (data samples)
- Apply SD transformations $(\mathrm{E} = \frac{1}{2}\sum_{p=1}^{P}(y_p - \bar{y}_p)^2)$

$$\omega_{m+1}[i] \leftarrow w_m[i] + \mu\frac{\partial(E)}{\partial \mathrm{w[i]}}$$

$$\frac{\partial(E)}{\partial \mathrm{w[i]}} = \sum_{p=1}^{m}(y_p - \bar{y}_p)x_p[i]$$

- Set $m = m + 1$.
  - Calculate $E_{\text{new}} = \sum_p (y_p - \bar{y}_p)^2$
  - Update μ.
  - If $|E_{\text{new}} - E_{\text{old}}| <$ tolerance STOP. Otherwise set $E_{\text{old}} = E_{\text{new}}$ and go back to step 2.

Backprop algorithm

1. Initialize
   1. Initialize $b, \omega(.,.)$, and $x(.)$ to random values
   2. Set SSE_{old} to very large value.
   3. Set $m = 0$ (iteration number)
2. Compute $v_p^*[h] = \sum_i w[i,h]u_p[i]$ for each p and each h.
3. Compute $v_p[h] = \frac{1}{1+e^{-v_p^*[h]}}$ for each p and each h
4. Compute $\bar{y}_p = b + \sum_h x[h]v_p[h]$ for each p.

Feed forward

5. Apply SD transformations (E $= \frac{1}{2}\sum_{p=1}^{P}\left(y_p - \bar{y}_p\right)^2$)

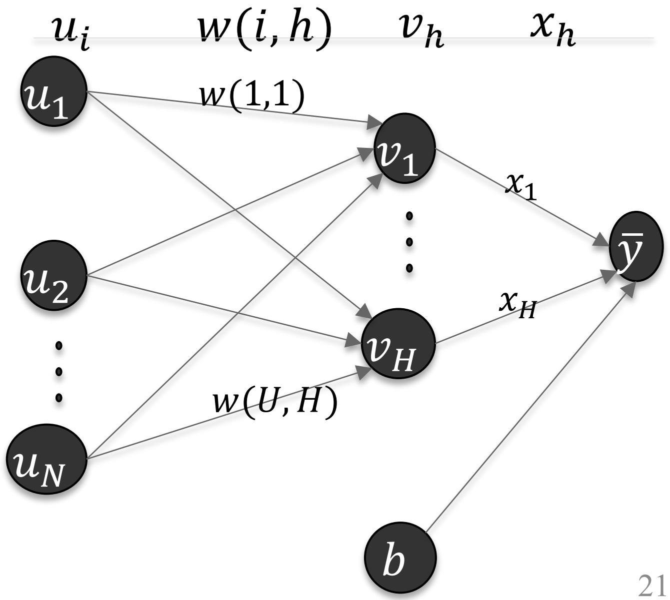
6.
$$b_m \leftarrow b_{m-1} - \mu\frac{\partial(E)}{\partial b}$$
$$\omega_m[i,h] \leftarrow \omega_{m-1}[i,h] - \frac{\mu\partial(E)}{\partial\omega[i,h]}$$
$$x_m[h] \leftarrow x_{m-1}[h] - \mu\frac{\partial(E)}{\partial x[h]}$$

7. Set $m = m + 1$
   1. Calculate $E_{new} = \sum_p\left(y_p - \bar{y}_p\right)^2$
   2. Update μ
   3. If $[E_{\text{new}} - E_{\text{old}}] <$ tolerance $\rightarrow STOP$ Otherwise set $E_{\text{old}} = E_{\text{new}}$ got step 2.

- Do multiple iterations of the algorithm to start at different initial points
- Initialize $b, \omega[.,.]$, and $x[.]$ to select from a random range; Try a few different ranges
- Try a few different options for learning rate ($\mu = \frac{A}{B+m}; \mu = \frac{1}{m}; \mu = 0.001(small\ constant)$)

Derivation of Backprop derivatives

- Main transformation is the steepest descent $\vec{x} \leftarrow \vec{x} - \mu \nabla \mathrm{f}(\vec{x})$; but here we are solving for $\omega[.,.]$, and $x[.]$
- $b_m \leftarrow b_{m-1} - \mu \frac{\partial(E)}{\partial b}$
- $\omega_m[i,h] \leftarrow \omega_{m-1}[i,h] - \frac{\mu\partial(E)}{\partial\omega[i,h]}$
- $x_m[h] \leftarrow x_{m-1}[h] - \mu \frac{\partial(E)}{\partial x[h]}$



$\frac{\partial E}{\partial x[h]}$=? $\frac{\partial E}{\partial b}$=?

- $\frac{\partial E}{\partial x[h]} = \frac{1}{2}\sum_{p=1}^{P} \frac{\partial}{\partial x[h]}\left(y_p - \bar{y}_p\right)^2$

$$= \frac{2}{2}\sum_{p=1}^{P} \left(y_p - \bar{y}_p\right)\frac{\partial}{\partial x[h]}\left(y_p - \bar{y}_p\right)$$

$$= \sum_p \left(y_p - \bar{y}_p\right)\left(-\frac{\partial \bar{y}_p}{\partial x[h]}\right)$$

$$= \sum_p \left(y_p - \bar{y}_p\right)\left(-\frac{\partial\left(b + \sum_h x[h]v_p[h]\right)}{\partial x[h]}\right)$$

$$\frac{\partial E}{\partial x[h]} = -\sum_p \left(y_p - \bar{y}_p\right)v_p[h]$$

- $\frac{\partial E}{\partial b} = -\sum_p \left(y_p - \bar{y}_p\right)$

$$\frac{\partial E}{\omega(i,h)} = ?$$

$$\frac{\partial E}{\omega[i,h]} = \frac{1}{2}\sum_p \frac{\partial}{\partial \omega[i,h]}\left(y_p - \bar{y}_p\right)^2$$

$$= \frac{1}{2}\sum_p 2\left(y_p - \bar{y}_p\right)\frac{\partial}{\partial \omega[i,h]}\left(y_p - \bar{y}_p\right)$$

$$= \sum_p \left(y_p - \bar{y}_p\right)\left(-\frac{\partial}{\partial w[i,h]}\bar{y}_p\right)$$

$$= -\sum_p \left(y_p - \bar{y}_p\right) \cdot \left(\frac{\partial \bar{y}_p}{\partial v_p[h]} \frac{\partial v_p[h]}{\partial v_p^*[h]} \frac{\partial v_p^*[h]}{\partial \omega[i,h]}\right)$$

$$\frac{\partial E}{\partial w[i,h]} = -\sum_p \left(\left(y_p - \bar{y}_p\right)x[h]v_p[h]\left(1 - v_p[h]\right)u_p[i]\right)$$

This changes for different activation functions

- Feed forward equations
  - $v_p^*[h] = \sum_i \omega[i,h]u_p[\mathrm{i}];$
  - $v_p[h] = \frac{1}{1+e^{-v^*[h]}};$
  - $y_p = b + \sum_h v_p[h]x[h]$
- $\frac{\partial v_p^*[h]}{\partial w(i,h)} = u_p[\mathrm{i}]$
- $\frac{\partial v_p[h]}{\partial v_p^*[h]} = \frac{\partial\left(1+e^{-v_p^*[h]}\right)^{-1}}{\partial v_p^*[h]}$

$$= \frac{-1}{\left[1+e^{-v_p^*[h]}\right]^2} e^{-v_p^*[h]}(-1)$$

$$= \frac{1+e^{-v_p^*[h]}-1}{\left[1+e^{-v_p^*[h]}\right]^2} \textit{(add + 1 , - 1 in numerator)}$$

$$= \frac{1+e^{-v_p^*[h]}}{\left[1+e^{-v_p^*[h]}\right]^2} - \frac{1}{\left[1+e^{-v_p^*[h]}\right]^2}$$

$$= \frac{1}{1+e^{-v_p^*[h]}} - \frac{1}{\left[1+e^{-v_p^*[h]}\right]^2}$$

$$= v_p[h]\left(1 - v_p[h]\right)$$

Backprop algorithm

1. Initialize
    1. Initialize $b, \omega(.,.)$, and $x(.)$ to random values
    2. Set SSE_{old} to very large value.
    3. Set $m = 0$ (iteration number)
2. Compute $v_p^*[h] = \sum_i w[i,h]u_p[\mathrm{i}]$ for each p and each h.
3. Compute $v_p[h] = \frac{1}{1+e^{-v_p^*[h]}}$ for each p and each h
4. Compute $\bar{y}_p = b + \sum_h x[h]v_p[h]$for each p (data samples)
5. Apply SD transformations ($\mathrm{E} = \frac{1}{2}\sum_{p=1}^{P}\left(y_p - \bar{y}_p\right)^2$)

$$b_m \leftarrow b_{m-1} - \mu\frac{\partial(E)}{\partial b}; \qquad \frac{\partial E}{\partial b} = -\sum_p \left(y_p - \bar{y}_p\right)$$

6. $$\omega_m[i,h] \leftarrow \omega_{m-1}[i,h] - \mu\frac{\partial(E)}{\partial\omega[i,h]}; \qquad \frac{\partial E}{\partial \mathrm{w}[i,h]} = -\sum_p \left(\left(y_p - \bar{y}_p\right)x[h]v_p[h](1 - v_p[h])u_p[i]\right)$$

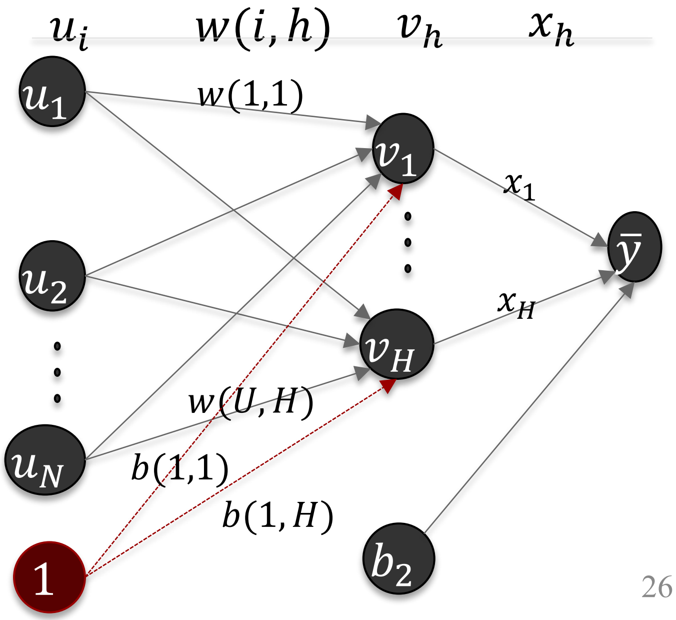
$$x_m[h] \leftarrow x_{m-1}[h] - \mu\frac{\partial(E)}{\partial x[h]}; \qquad \frac{\partial E}{\partial x[h]} = -\sum_p \left(y_p - \bar{y}_p\right)v_p[h]$$

7. Set $m = m + 1$
    1. Calculate $E_{new} = \sum_p\left(y_p - \bar{y}_p\right)^2$
    2. Update μ
    3. If $[E_{\mathrm{new}} - E_{\mathrm{old}}] <$ tolerance $\rightarrow$ *STOP* Otherwise set $E_{\mathrm{old}} = E_{\mathrm{new}}$ got step 2.

- Do multiple iterations of the algorithm to start at different initial points
- Initialize $b, \omega[.,.]$, and $x[.]$ to select from a random range; Try a few different ranges
- Try a few different options for learning rate ($\mu = small\ constant;\ \mu = \frac{A}{B+m};\ \mu = \frac{1}{m};$)

This applies to sigmoid activation; change if using a different activation function

Backprop for below (addition of a bias node in input layer)?



Visualization of NN

- https://cs.stanford.edu/people/karpathy/convnetjs/demo/regression.html
- https://cs.stanford.edu/people/karpathy/convnetjs/
- https://playground.tensorflow.org

Further study

- Activation functions:
  - Sigmoid, Tanh, (for function approximation); vanishing gradient/exploding gradient challenges when layers increase
  - ReLU (overcomes vanishing gradient/exploding gradient problem)
  - Numerous other AF: https://arxiv.org/abs/1811.03378
- Neural network architectures:
  - https://towardsdatascience.com/the-mostly-complete-chart-of-neural-networks-explained-3fb6f2367464
- Optimizers (Backprop is typically the core, but estimations of gradient and types of learning rates may vary)
  - Adam, RMSProp, SGDm AdamW, Adadelta, Adamax, Adafactor, Nadam;
  - https://keras.io/api/optimizers/
  - https://pytorch.org/docs/stable/optim.html
  - Schmidt, et.al., ICML 2021; https://github.com/SirRob1997/Crowded-Valley---Results
- Deep learning: Sarker, I.H. Deep Learning: A Comprehensive Overview on Techniques, Taxonomy, Applications and Research Directions. *SN COMPUT. SCI.* **2**, 420 (2021). https://doi.org/10.1007/s42979-021-00

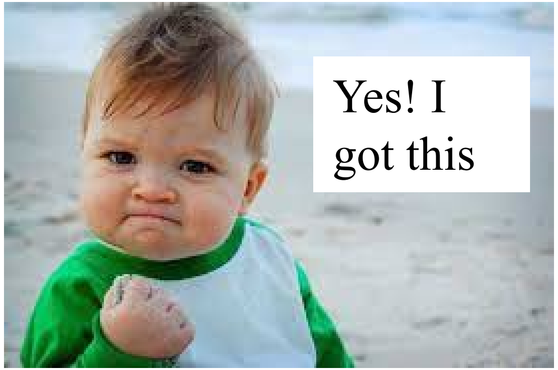
Output layer

- Multiple nodes in output layer→ when predicting multinomial functions
  - AF may then be used in output layer as well,
  - In RL, for policy gradient, we will use multiple nodes in output layer with softmax AF.
  - Lapan Ch3-Module1 code uses softmax
- Droput in output layer :
  - It is a form of regularization in supervised learning, especially when data is sparse; https://arxiv.org/pdf/1207.0580.pdf
  - It is not used that much in reinforcement learning, but there is some research in its use in policy gradient https://doi.org/10.48550/arXiv.2202.11818
  - Lapan Ch3-Module1 code uses dropout

Do you know?

- What we looked at is the simplest neural net architecture
- NN is a **single layer perceptron** developed in 1950’s
- Deep learning- generally multi-layer NN are called DL
- **We looked at specific DL called feed forward**,
- Other types of deep learning (potentially a course in Fall 2025)
  - CNN (convolution neural network)
  - RNN (recurrent neural network)
  - Autoencoders
  - Attention networks
  - Self-attention mechanisms (Transformers)
- Single layer feedforward is very powerful for representation of most non-linear functions of any form – Universal approximators
- Other architectures were developed for image and text processing, processing of sequence data
  - Sequence based methods can be used for dynamic decision making (when we get to RL we will discuss deep learning)

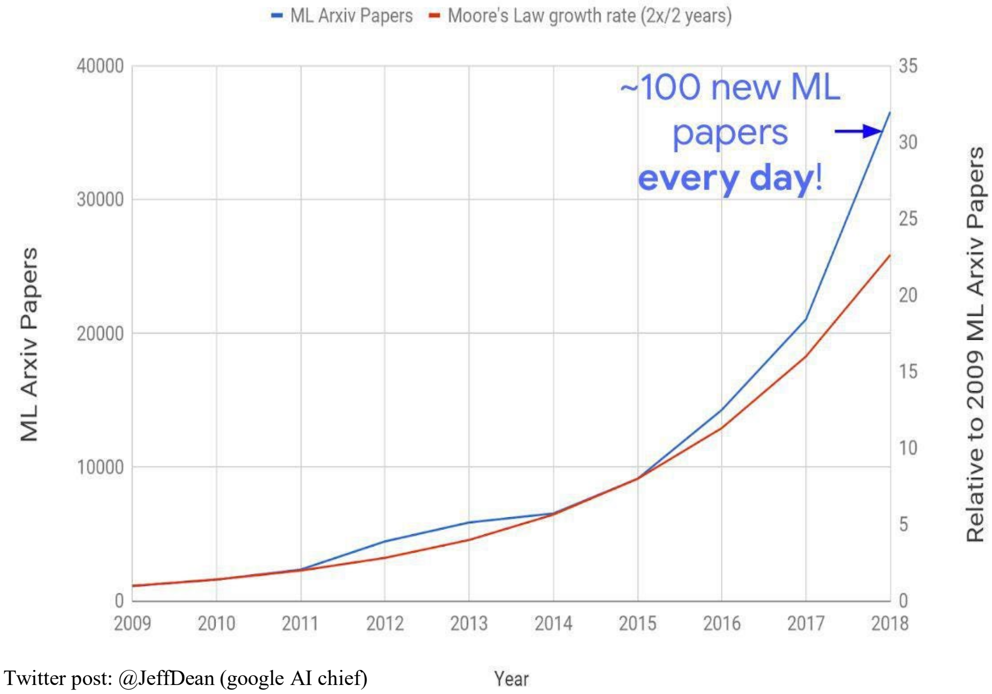
Work colleague: I saw deep learning on your resume, I have a question for you..
New graduate:



Work colleague: Great, we are debating on whether CNN or RNN for this ………………….. what do you think?

Let’s try to figure it out

Fast growing; hard to keep up with everything



Twitter post: @JeffDean (google AI chief)

Older version: Synced, 2018, https://syncedreview.com/2018/07/26/google-ai-chief-jeff-deans-ml-system-architecture-blueprint/

Student asks question

Professor’s response: Just restates the question in different words

If your mathematical foundation is strong, you can figure it out

Questions

- Have one hidden layer but increase number of nodes
- Vs.
- Have multiple hidden layers with fewer nodes in each
- The first does piecewise non-linear fits, while the second does non-linear transformations to a higher lever;
  - So for a complex (highly non-linear function), number nodes needed to get same fit will be much higher than adding multiple layers.
  - The first would also be overfitting

Application areas of NN

- As a machine learning tool (static applications)– prediction, classification, function approximation
  - This application will follow typical machine learning rules for model training and accuracy; spilt by test and train data; evaluate predictive power using test scores etc.
  - How neural networks learn (insight into application of NN)
  - https://www.youtube.com/watch?v=TkwXa7Cvfr8
- As function approximation of environment (dynamic system representation)
  - e.g., recurrent neural network (RNN) can be trained to replace large-scale high-computation simulation models
  - Computational burden of training NN is high, but once trained computation burden is minimal
- As function approximation for learning the policy function or value function in control optimization /intelligent systems (dynamic decision making – we will see in deep RL)
  - This is the focus of this class
  - We will skip typical steps of ML training and testing here and look at overall model performance with RL

UMassAmherst

The Commonwealth's Flagship Campus