# Discriminative Models II

Saumitra Yadav

# Last Lecture

Problem: Classify wolf and Brynden Rivers (as a raven in dream) (POS Tagging)

Generative Models (e.g: naive bayes):

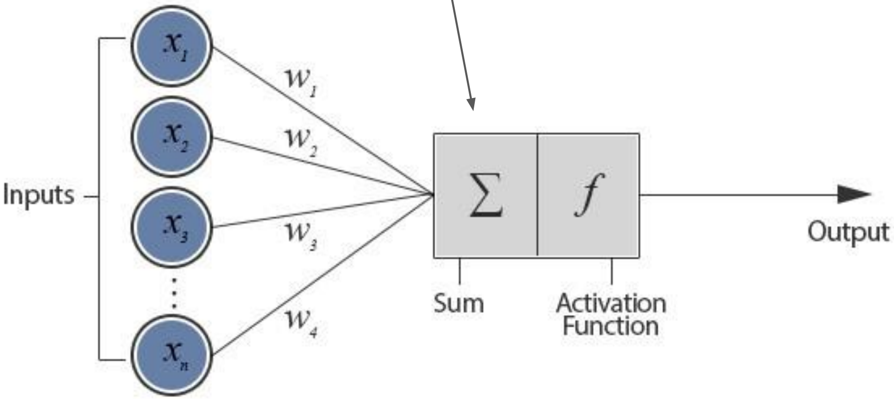
- What does wolf and Brynden Rivers (as a raven in dream) look like? ( Generating Words given POS Tags )

Discriminative Models (e.g: logistic regression, neural networks):

- Find features ( one lacks wings, a third eye and another one doesn’t stand on all limbs and is present in a dream) to discriminate between inputs. (what kind of distinctions do nouns have when compared with verb- distinctions like company they keep etc.)

# Artificial Neuron

$-x_0 w_0 \rightarrow$ bias



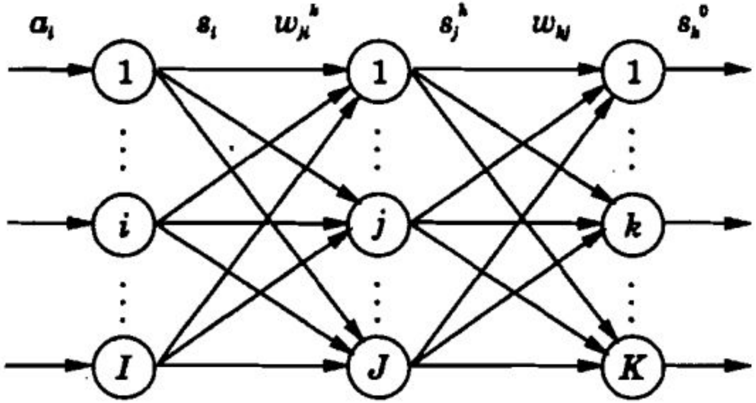

Taken From [1]

# Perceptron Learning Algorithm (Frank Rosenblatt)

- Working of neural network governed by neural dynamics.
  a. Short term memory - Activation Dynamics - (Sum : $\Sigma w_i x_i$)
  b. Long term memory - Synaptic Dynamics - ( W - containing all weights )
- Learning Laws are implementation models of Synaptic Dynamics
- Algorithm
  a. Initialise **W** to small random values
  b. Present a vector **x** as input and calculate the output **y**.
  c. Updates the weights according to
     - $w_j(t+1)=w_j(t)+\eta(y'-y)x$
     - Where
       - ***y'*** is desired output
       - ***t*** is the iteration number
       - ***η*** is the step size in range 0.0 and 1.0
  d. Repeat b and c until one of the following conditions is met:
     - Iteration error ( square of summation of all errors ) is less than a threshold value
     - ***t*** has reached a predetermined number for completion.

# Multilayer Perceptron Network

- Classify non-linear separable data



$a_i$: **i**th unit of **I** dimensional input
$s_i$: output of activation of **i**th unit in input layer
$w_{ji}^h$: weight from **i**th unit to **j**th unit; where jth unit is in hidden layer
$s_j^h$: output of activation of **j**th unit in hidden layer
$w_{kj}$: weight from **j**th hidden unit to **k**th unit in output layer
$s_k^o$: output of activation of **k**th unit in output layer

Taken from [2]

# Backpropogation Learning Algorithm

- Network is a function mapping from input to output space $\mathbf{R}^{\mathbf{I}} : \mathbf{R}^{\mathbf{K}}$
- Using gradient search along error surface for optimum sets of weight
- Given input **a(m),** calculated output **b'(m)** and **b(m)** being actual output in $\mathbf{m}_{\mathbf{th}}$ iteration
- Let **E(m)** be error, then gradient descent along error surface so as to change weight connection units **j** and **i**.

$$\Delta w_{ij}(m) = -\eta \frac{\partial E(m)}{\partial w_{ij}}$$

Where **η** > 0

- Accordingly weight update will be as follows

$$w_{ij}(m+1) = w_{ij}(m) + \Delta w_{ij}(m)$$

# Backpropogation Learning Algorithm Contd.

- Error (mean squared error) for **m**th step is,

$$E(m) = \frac{1}{2}\sum_{k=1}^{K}[b_k(m) - b_k'(m)]^2 = \frac{1}{2}\sum_{k=1}^{K}[b_k(m) - s_k^o]^2$$

$$= \frac{1}{2}\sum_{k=1}^{K}[b_k(m) - f_k^o(x_k^o)]^2,$$

- Where

$$x_k^o = \sum_{j=1}^{J} w_{kj}\, s_j^h \qquad x_j^h = \sum_{i=1}^{I} w_{ji}^h\, s_i$$

$$s_j^h = f_j^h(x_j^h) \qquad s_i = x_i = a_i(m).$$

# Backpropogation Learning Algorithm Contd.

For weights leading to output unit

$$\Delta w_{kj}(m) = -\eta \frac{\partial E(m)}{\partial w_{kj}}$$

$$\frac{\partial E(m)}{\partial w_{kj}} = \frac{1}{2} \frac{\partial}{\partial w_{kj}} \left[ b_k - f_k^o \left( \sum_{j=1}^{J} w_{kj} s_j^h \right) \right]^2$$

$$= -(b_k - f_k^o) \dot{f}_k^o s_j^h$$

$$= -\delta_k^o s_j^h,$$

Where,

$$\delta_k^o = (b_k - f_k^o) \dot{f}_k^o.$$

# Backpropogation Learning Algorithm Contd.

Therefore $$\Delta w_{kj}(m) = \eta \delta_k^o s_j^h$$

And,

$$\begin{aligned} w_{kj}(m+1) &= w_{kj}(m) + \Delta w_{kj}(m) \\ &= w_{kj}(m) + \eta \delta_k^o s_j^h . \end{aligned}$$

Update is learning_rate **x** error propagated back by $k^{th}$ unit **x** activation output at **j**

Gradient for weights leading to units in hidden layer is,

$$\Delta w_{ji}^h(m) = -\eta \frac{\partial E(m)}{\partial w_{ji}^h}$$

# Backpropogation Learning Algorithm Contd.

Which gives us,

$$\frac{\partial E(m)}{\partial w_{ji}^h} = -\sum_{k=1}^{K} (b_k - f_k^o) \frac{\partial f_k^o \left( \sum_{j=1}^{J} w_{kj} s_j^h \right)}{\partial w_{ji}^h}$$

$$= -\sum_{k=l}^{K} (b_k - f_k^o) \dot{f}_k^o w_{kj} \frac{\partial s_j^h}{\partial w_{ji}^h},$$

And since $\boldsymbol{s_j^h = f_j^h(x_j^h)}$

$$\frac{\partial s_j^h}{\partial w_{ji}^h} = \dot{f}_j^h \frac{\partial x_j^h}{\partial w_{ji}^h}$$

# Backpropogation Learning Algorithm Contd.

And Since $x_j^h = \sum_{i=1}^{I} w_{ji}^h s_i$ so, $\dfrac{\partial x_j^h}{\partial w_{ji}^h} = s_i$

Hence,

$$\frac{\partial E(m)}{\partial w_{ji}^h} = -\sum_{k=1}^{K} (b_k - f_k^o)\, \dot{f}_k^o\, w_{kj}\, \dot{f}_j^h\, s_i$$

$$= -\delta_j^h s_i$$

Where, $\delta_j^h = \dot{f}_j^h \sum_{k=1}^{K} w_{kj}\, \delta_k^o$

# Backpropogation Learning Algorithm Contd.

Hence,

$$\Delta w_{ji}^{h}(m) = \eta\, \delta_j^h s_i = \eta\, \delta_j^h a_i(m)$$

**Update** is learning_rate **x** error propagated back from $j^{th}$ unit ( which consist of sum of all error propagated back to jth unit from all units in layer **K** ) **x** $i^{th}$ output of input layer

And since ***$s_i = x_i = a_i(m)$*** therefore,

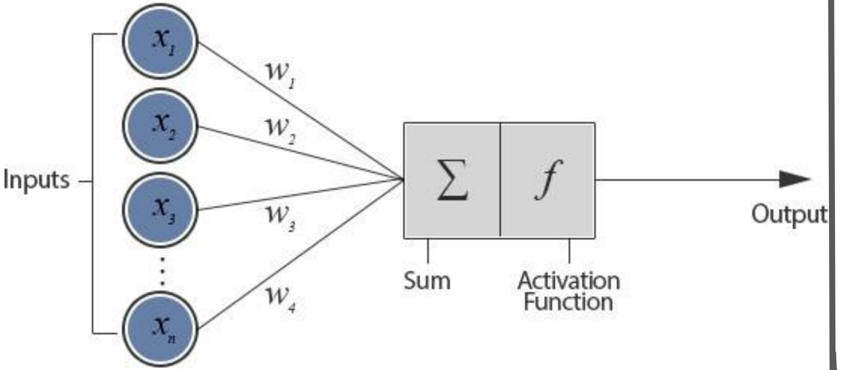
$$w_{ji}^{h}(m+1) = w_{ji}^{h}(m) + \Delta w_{ji}^{h}(m)$$

$$= w_{ji}^{h}(m) + \eta\, \delta_j^h a_i(m)$$

Where,

$$\delta_j^h = \dot{f}_j^h \sum_{k=1}^{K} w_{kj}\, \delta_k^o$$ , denotes the error propagated back.

# Artificial Neuron - Matrix Perspective




Taken From [1]

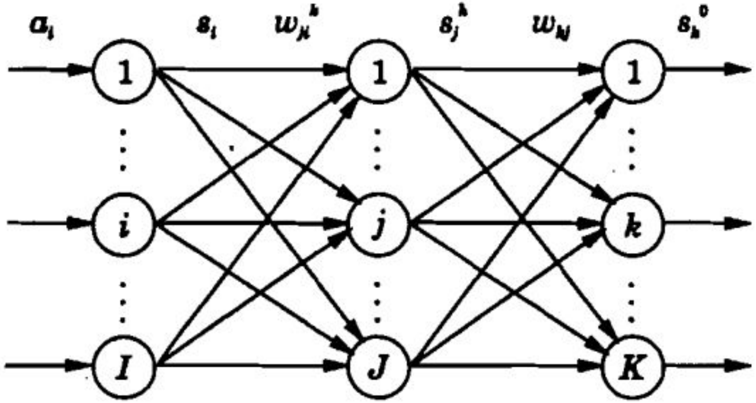
$$\begin{bmatrix} w_{11} & w_{21} & w_{31} & w_{41} & w_{51} & w_{61} & w_{71} & w_{81} & w_{91} \\ w_{12} & w_{22} & w_{32} & w_{42} & w_{52} & w_{62} & w_{72} & w_{82} & w_{92} \\ w_{13} & w_{23} & w_{33} & w_{43} & w_{53} & w_{63} & w_{73} & w_{83} & w_{93} \\ w_{14} & w_{24} & w_{34} & w_{44} & w_{54} & w_{64} & w_{74} & w_{84} & w_{94} \\ w_{15} & w_{25} & w_{35} & w_{45} & w_{55} & w_{65} & w_{75} & w_{85} & w_{95} \end{bmatrix} \begin{bmatrix} i_1 \\ i_2 \\ i_3 \\ i_4 \\ i_5 \\ i_6 \\ i_7 \\ i_8 \\ i_9 \end{bmatrix} = \begin{bmatrix} total\ input\ to\ o_1 \\ total\ input\ to\ o_2 \\ total\ input\ to\ o_3 \\ total\ input\ to\ o_4 \\ total\ input\ to\ o_5 \end{bmatrix}$$

Applying ***f*** on resulting vector

Taken from [4]

# Multilayer Perceptron Network - Matrix Perspective

- Classify non-linear separable data



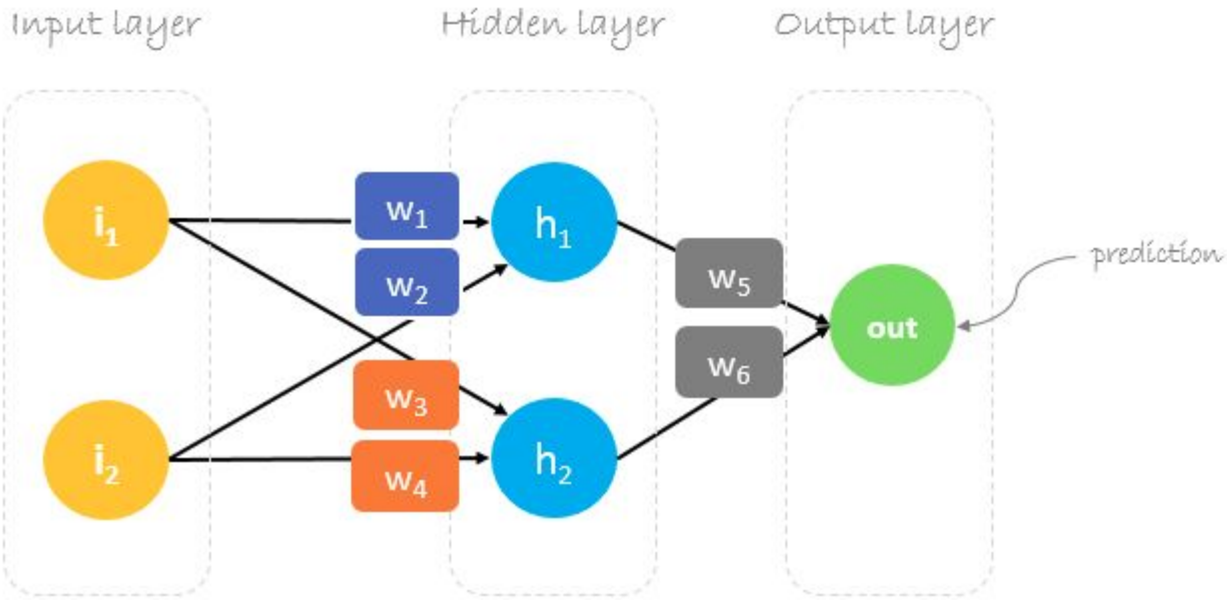

$a_i$: **i**th unit of **I** dimensional input
$s_i$: output of activation of **i**th unit in input layer
$w_{ji}^h$: weight from **i**th unit to **j**th unit; where jth unit is in hidden layer
$s_j^h$: output of activation of **j**th unit in hidden layer
$w_{kj}$: weight from **j**th hidden unit to **k**th unit in output layer
$s_k^o$: output of activation of **k**th unit in output layer

Taken from [2]

$$\text{output} = f^o(W^o \boldsymbol{x} f^h(W^h \boldsymbol{x} A))$$

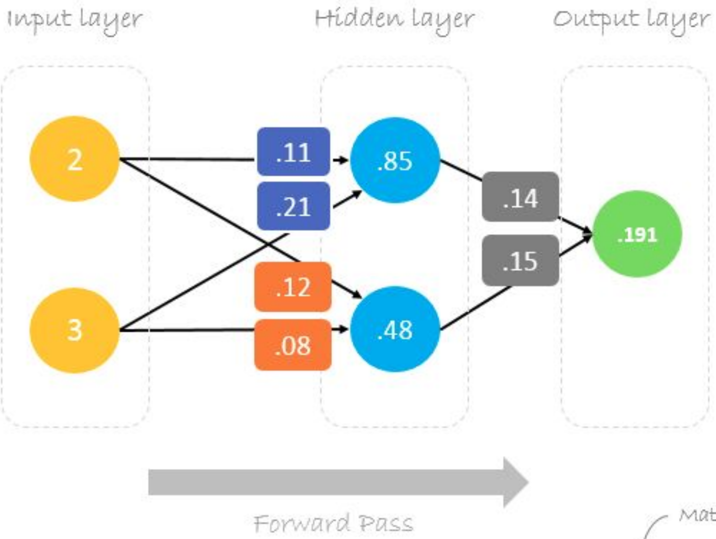
Where,
A: input vector/matrix of size n**x**v, n is number of dimension and v is 1/number of samples.
$W^h$: Weight Matrix between hidden and input layer
$f^h$: activation function of hidden layer
$W^o$: Weight Matrix between hidden and output layer
$f^o$: activation function of output layer at each cell

# Matrix Perspective naive example




Taken from [3]

# Matrix Perspective naive example contd.




Matrix multiplication

$$[2 \quad 3] \cdot \begin{bmatrix}0.11 & 0.12\\0.21 & 0.08\end{bmatrix} = [0.85 \quad 0.48] \cdot \begin{bmatrix}0.14\\0.15\end{bmatrix} = [0.191]$$

Details

2 x .11 + 3 x .21 = .85

2 x .12 + 3 x .08 = .48

.85 x .14 + .48 x .15 = .191

Actual Output : 1

Error = $(0.191 - 1)^2/2$

= 0.327

Derivative of Error with respect to weight

Old weight

$$^*W_x = W_x - a\left(\frac{\partial Error}{\partial W_x}\right)$$

New weight

Learning rate

# Matrix Perspective naive example contd.

$$\frac{\partial Error}{\partial W_6} = \frac{\partial Error}{\partial prediction} * \frac{\partial prediction}{\partial W_6}$$ chain rule

$$\boxed{Error = \frac{1}{2}(prediction - actual)^2}$$

$$prediction = (i_1 w_1 + i_2 w_2) w_5 + (i_1 w_3 + i_2 w_4) w_6$$

$$\frac{\partial Error}{\partial W_6} = \frac{\frac{1}{2}(predictoin - actula)^2}{\partial prediciton} * \frac{\partial\,(i_1 w_1 + i_2 w_2) w_5 + (i_1 w_3 + i_2 w_4) w_6}{\partial W_6}$$

$$\frac{\partial Error}{\partial W_6} = 2 * \frac{1}{2}(\mathbf{predictoin} - \mathbf{actula}) \frac{\partial(predictoin - actula)}{\partial prediciton} * (i_1 w_3 + i_2 w_4)$$

$$h_2 = i_1 w_3 + i_2 w_4$$

$$\frac{\partial Error}{\partial W_6} = (\mathbf{predictoin} - \mathbf{actula}) * (h_2)$$

$$\Delta = prediction - actual$$ delta

$$\frac{\partial Error}{\partial W_6} = \Delta h_2$$

# Matrix Perspective naive example contd.

updated weights

$$
\begin{aligned}
{}^*w_6 &= w_6 - a\,(h_2\ .\ \Delta)\\
{}^*w_5 &= w_5 - a\,(h_1\ .\ \Delta)\\
{}^*w_4 &= w_4 - a\,(i_2\ .\ \Delta w_6)\\
{}^*w_3 &= w_3 - a\,(i_1\ .\ \Delta w_6)\\
{}^*w_2 &= w_2 - a\,(i_2\ .\ \Delta w_5)\\
{}^*w_1 &= w_1 - a\,(i_1\ .\ \Delta w_5)
\end{aligned}
$$

# Matrix Perspective naive example contd.

$$\begin{bmatrix} w_5 \\ w_6 \end{bmatrix} = \begin{bmatrix} w_5 \\ w_6 \end{bmatrix} - a\,\Delta \begin{bmatrix} h_1 \\ h_2 \end{bmatrix} = \begin{bmatrix} w_5 \\ w_6 \end{bmatrix} - \begin{bmatrix} ah_1\Delta \\ ah_2\Delta \end{bmatrix}$$

$$\begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} = \begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} - a\,\Delta \begin{bmatrix} i_1 \\ i_2 \end{bmatrix} . \begin{bmatrix} w_5 & w_6 \end{bmatrix} = \begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} - \begin{bmatrix} a\, i_1 \Delta w_5 & a\, i_1 \Delta w_6 \\ a\, i_2 \Delta w_5 & a\, i_2 \Delta w_6 \end{bmatrix}$$

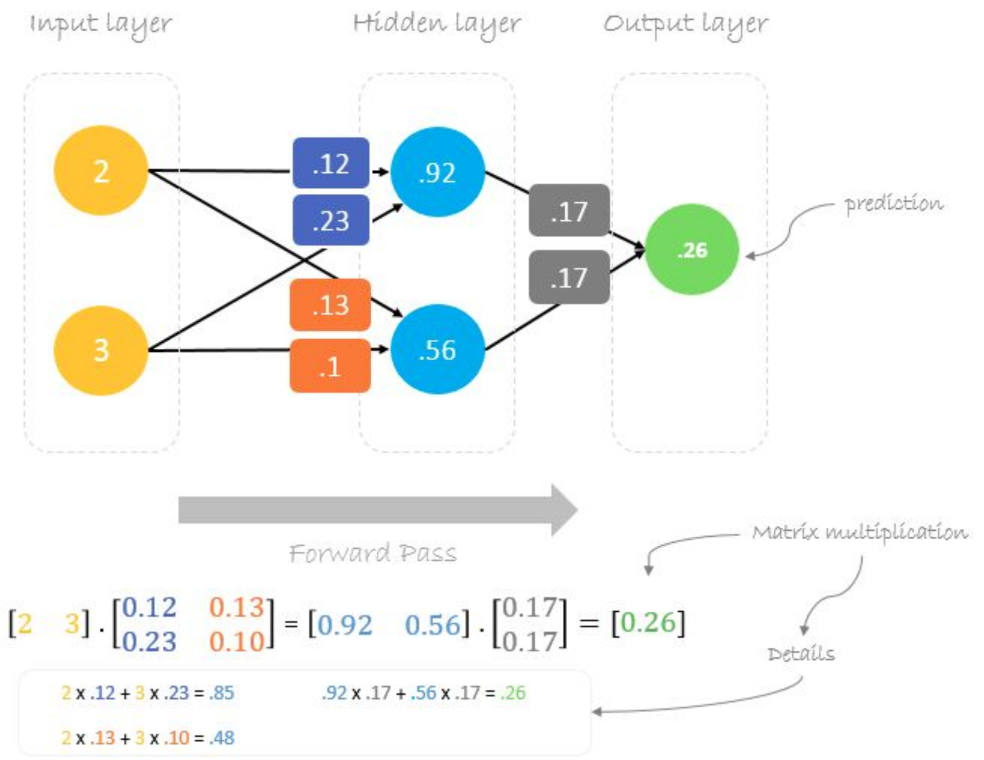
$\Delta = 0.191 - 1 = -0.809$ ← **Delta** = prediction - actual

$a = 0.05$ ← **Learning rate,** we smartly guess this number

$$\begin{bmatrix} w_5 \\ w_6 \end{bmatrix} = \begin{bmatrix} 0.14 \\ 0.15 \end{bmatrix} - 0.05(-0.809) \begin{bmatrix} 0.85 \\ 0.48 \end{bmatrix} = \begin{bmatrix} 0.14 \\ 0.15 \end{bmatrix} - \begin{bmatrix} -0.034 \\ -0.019 \end{bmatrix} = \begin{bmatrix} 0.17 \\ 0.17 \end{bmatrix}$$

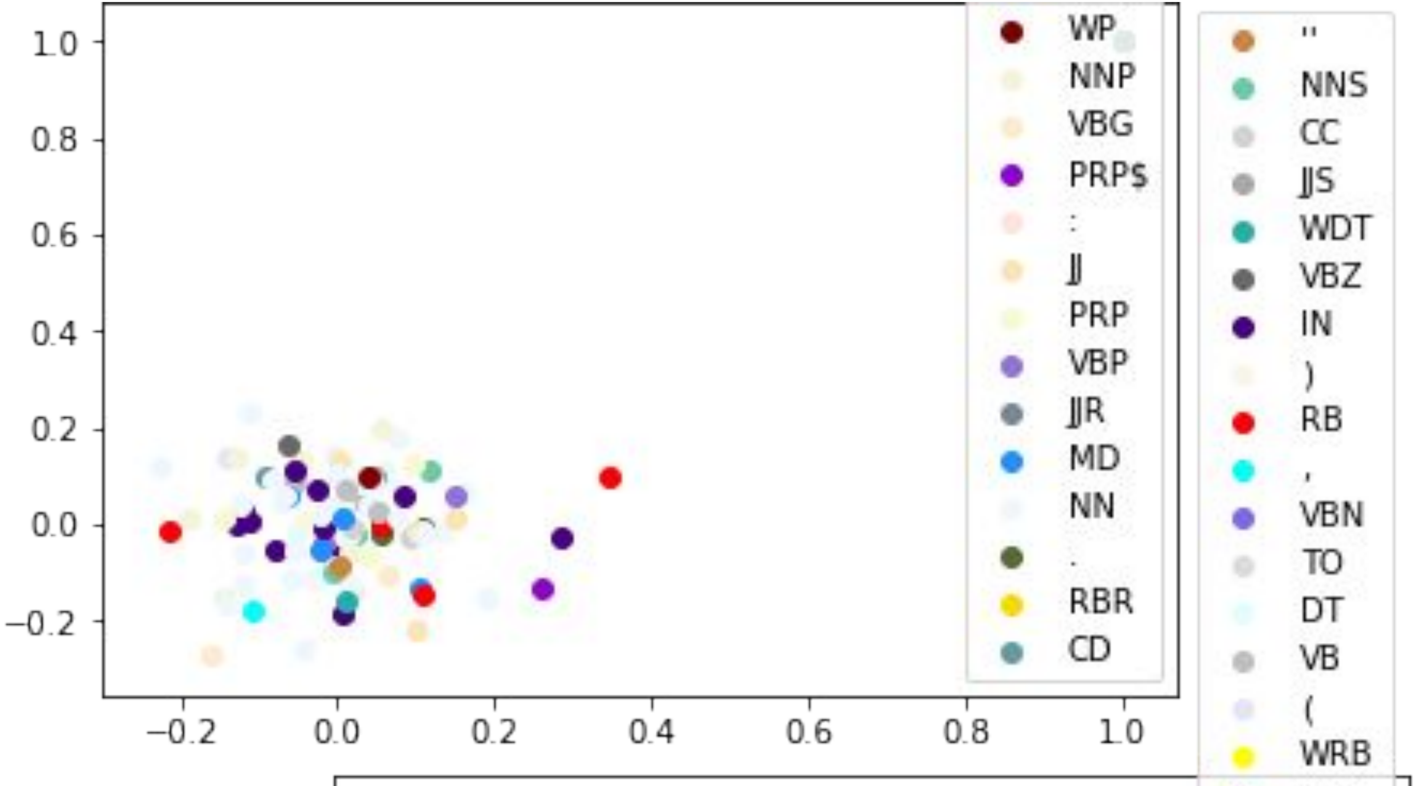
$$\begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} = \begin{bmatrix} .11 & .12 \\ .21 & .08 \end{bmatrix} - 0.05(-0.809) \begin{bmatrix} 2 \\ 3 \end{bmatrix} . \begin{bmatrix} 0.14 & 0.15 \end{bmatrix} = \begin{bmatrix} .11 & .12 \\ .21 & .08 \end{bmatrix} - \begin{bmatrix} -0.011 & -0.012 \\ -0.017 & -0.018 \end{bmatrix} = \begin{bmatrix} .12 & .13 \\ .23 & .10 \end{bmatrix}$$

# Matrix Perspective naive example contd.




$$\begin{bmatrix} 2 & 3 \end{bmatrix} \cdot \begin{bmatrix} 0.12 & 0.13 \\ 0.23 & 0.10 \end{bmatrix} = \begin{bmatrix} 0.92 & 0.56 \end{bmatrix} \cdot \begin{bmatrix} 0.17 \\ 0.17 \end{bmatrix} = \begin{bmatrix} 0.26 \end{bmatrix}$$

# One Application



# References for Text and Images

1. https://medium.com/technologymadeeasy/for-dummies-the-introduction-to-neural-networks-we-all-need-c50f6012d5eb
2. Artificial Neural Network by B.Yegnanarayana
3. http://hmkcode.github.io/ai/backpropagation-step-by-step/
4. http://www.sharetechnote.com/html/EngMath_Matrix_NeuralNetwork.html

# If you want to further read about MLP (specifically backprop)

1. https://www.analyticsvidhya.com/blog/2017/05/neural-network-from-scratch-in-python-and-r/
2. https://medium.com/datathings/neural-networks-and-backpropagation-explained-in-a-simple-way-f540a3611f5e
3. https://medium.com/@14prakash/back-propagation-is-very-simple-who-made-it-complicated-97b794c97e5c