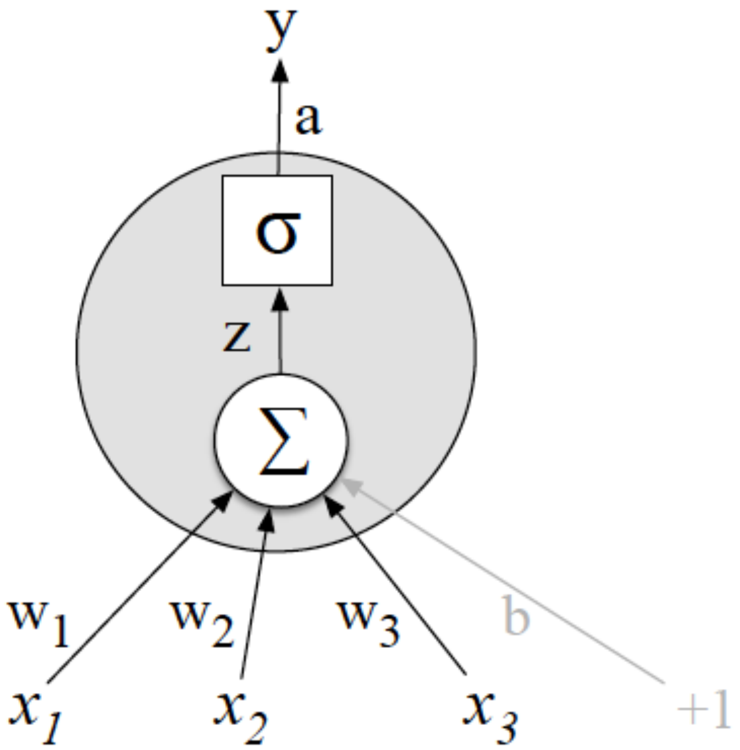
NEURAL LANGUAGE MODELS



Backgrounds

Computation ↔ Function

Computational Intelligence ↔ Learning the function

Function

$$f: M \rightarrow N$$

Transferring elements of source space (M) to target space (N)

$$f: R \rightarrow R : f(x) = y = x \times 2 + 1$$

f is known

f has two parameters (a=2, b=1)

Function

$f: M \to N$

Transferring elements of source space (M) to target space (N)

f: R $\to$ R : {(x, f(x))| (1,1), (2,4), (3,9), ...}

$f(x) = ?$

f is unknown!

Function

$$f\colon M \to N$$

Transferring elements of source space (M) to target space (N)

$$f\colon \mathrm{R} \to \mathrm{R} : \{(x, f(x)) \mid (1,1), (2,4), (3,9), \ldots\}$$

$$f(x) = ?$$

f is unknown! Might not exist!

Function

$$f\colon M \rightarrow N$$

Transferring elements of source space (M) to target space (N)

$f\colon R \rightarrow R : \{(x, f(x)) | (1,1), (2,4), (3,9), \ldots\}$

$f(x) = ?$

f is unknown! If exists, there may be more that one!

Function

$$f\colon M \rightarrow N$$

Transferring elements of source space (M) to target space (N)

f: R $\rightarrow$ R : {(x, f(x))| (1,1), (2,4), (3,9), …}

f(x) = ?

f is unknown! Even more than one exists, we may not find one!

Function

$$f\!: M \rightarrow N$$

Transferring elements of source space (M) to target space (N)

$$f\!: \mathrm{R} \rightarrow \mathrm{R} : \{(x, f(x)) \mid (1,1), (2,4), (3,9), \ldots\}$$

$$f(x) = ?$$

f is unknown! We assume at least one exists, we try to find it!

Transformation
$f: M \to N$

Transferring points (vectors) of source space (M) to target space (N)

$[T][X] = [Y]; X \in M, Y \in N$

T is a matrix that includes the parameters of *f*

Transformation (Linear Algebra)

$f: M \rightarrow N$

Transferring points (vectors) of source space (M) to target space (N)

$$\begin{bmatrix} 2 & 1 \end{bmatrix} \begin{bmatrix} x \\ 1 \end{bmatrix} = \begin{bmatrix} 2x + 1 \end{bmatrix}$$

$$f(x) = 2x + 1$$

Transformation (Linear Algebra)

$f: M \rightarrow N$

Transferring points (vectors) of source space (M) to target space (N)

$(AB)^T = B^T A^T \rightarrow$

$$[x \; 1]\begin{bmatrix} 2 \\ 1 \end{bmatrix} = [2x + 1]$$

$$f(x) = 2x + 1$$

Transformation (Linear Algebra)

$f\colon M \rightarrow N$

Transferring points (vectors) of source space (M) to target space (N)

$$[x\ 1]\begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = \{(x, f(x)) \mid (1,1), (2,4), (3,9), \ldots\}$$

$$f(x) = ?$$

Neural Network

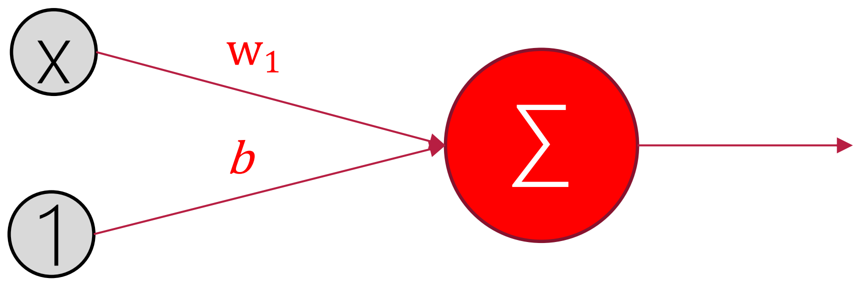
$f: M \rightarrow N$

Perceptron

Transferring points (vectors) of source space (M) to target space (N)

$$[x\ 1]\begin{bmatrix} w_1 \\ b \end{bmatrix} = y$$

$$f(x) = y$$



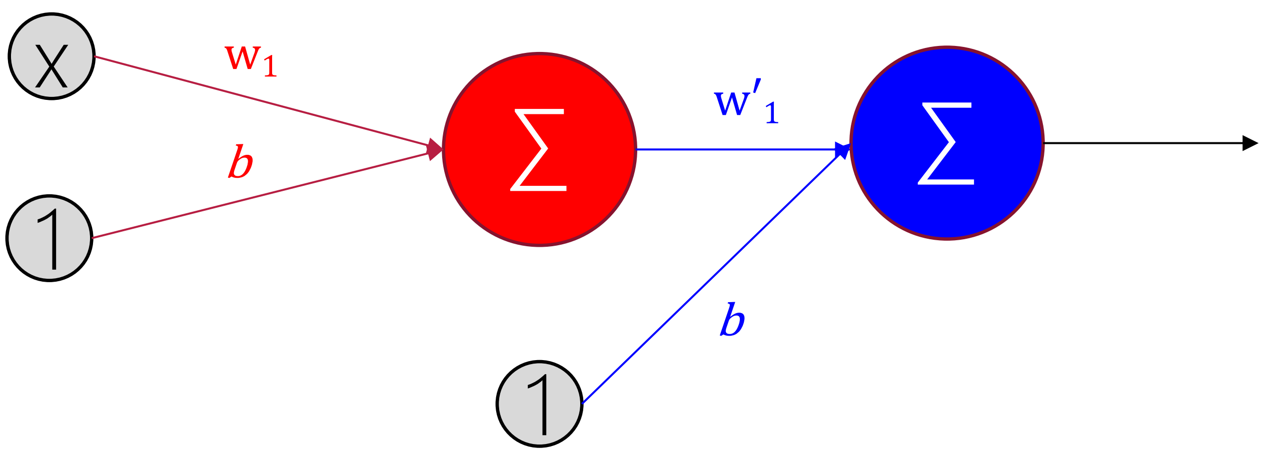
Neural Network

$$g\!: (f\!: M \rightarrow N) \rightarrow O$$

Transferring points (vectors) of source space (M) to target space (O)

$$[[x\ 1]\begin{bmatrix} w_1 \\ b \end{bmatrix}\ \ 1]\begin{bmatrix} w'_1 \\ b \end{bmatrix} = z$$

$$(g \circ f)(x) = g(f(x)) = z$$



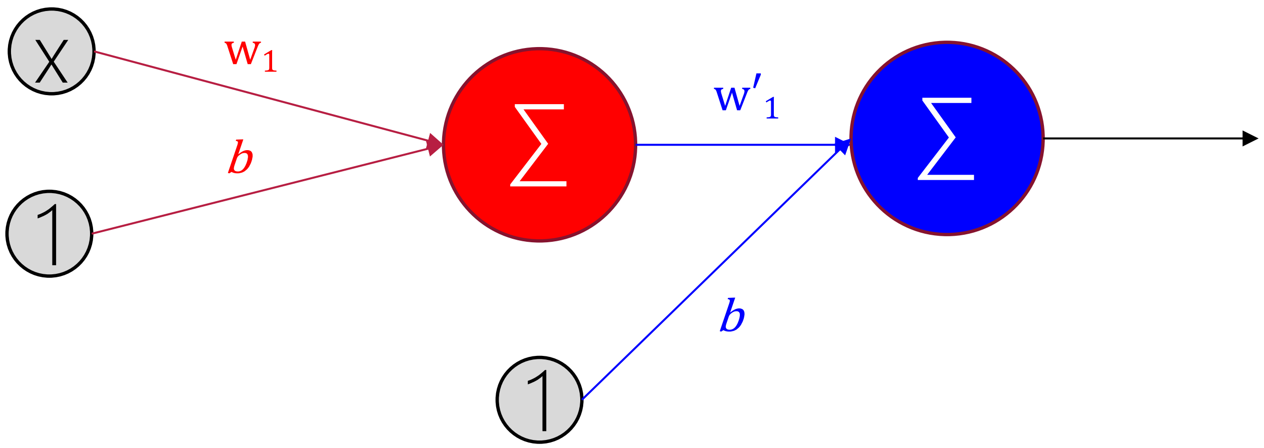
Neural Network

$$g\!: (f\!: M \to N) \to O$$

Transferring points (vectors) of source space (M) to target space (O)

$$[y\ 1]\begin{bmatrix} w'_1 \\ b \end{bmatrix} = z$$

$$g(f(x)) = g(y) = z$$



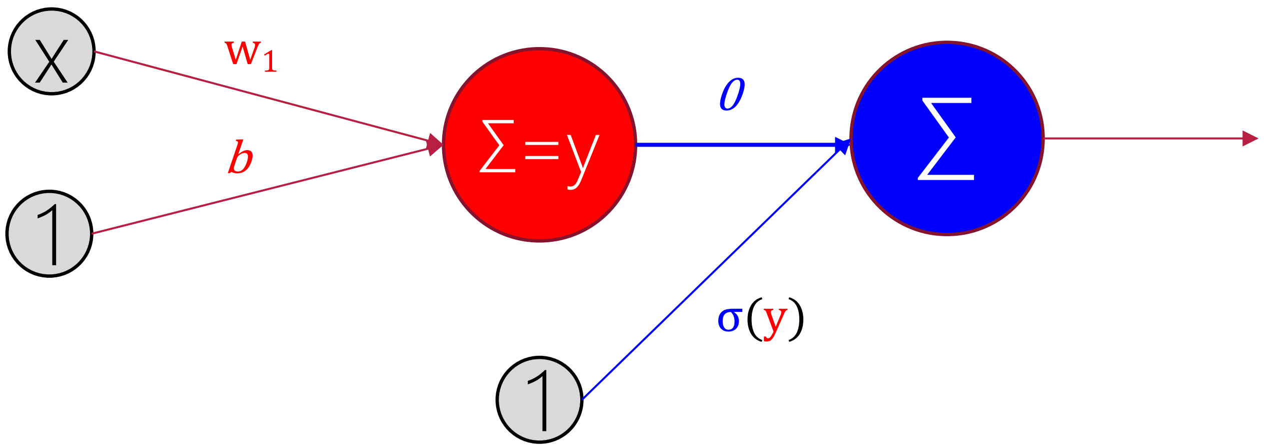
Neural Network

$$g\colon (f\colon M \to N) \to O$$

Transferring points (vectors) of source space (M) to target space (O)

$$[y\ \ 1]\begin{bmatrix} 0 \\ \sigma(y) \end{bmatrix} = y$$

$$g(f(x)) = g(y) = z$$



$$\left[[x\ \ 1]\begin{bmatrix} w_1 \\ b \end{bmatrix}\ \ 1\right]\begin{bmatrix} 0 \\ \sigma(x \times w_1 + b) \end{bmatrix} = z$$

Neural Network

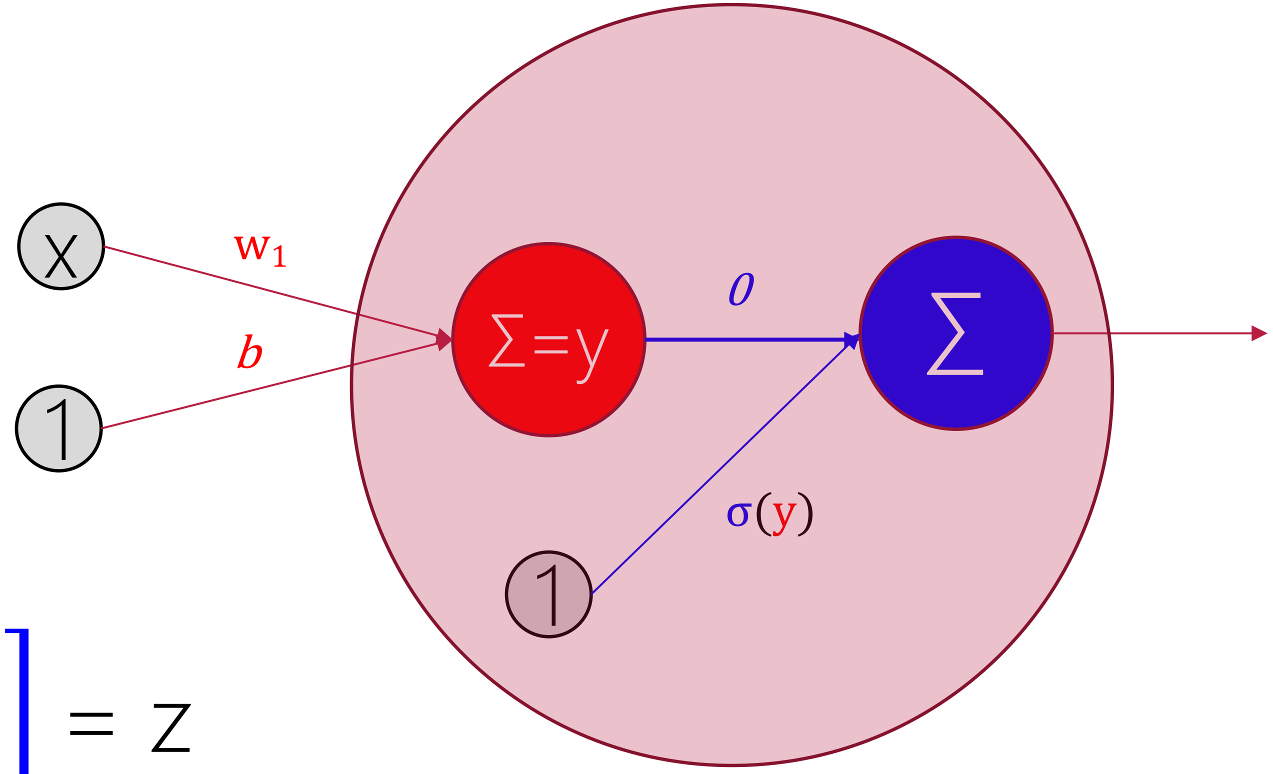
$$g\!:(f\!: M \rightarrow N) \rightarrow O$$

Transferring points (vectors) of source space (M) to target space (O)

$$[y\ 1]\begin{bmatrix} 0 \\ \sigma(y) \end{bmatrix} = y$$

$$g(f(x)) = g(y) = z$$

$$[[x\ 1]\begin{bmatrix} w_1 \\ b \end{bmatrix}\ \ 1]\begin{bmatrix} 0 \\ \sigma(x \times w_1 + b) \end{bmatrix} = z$$

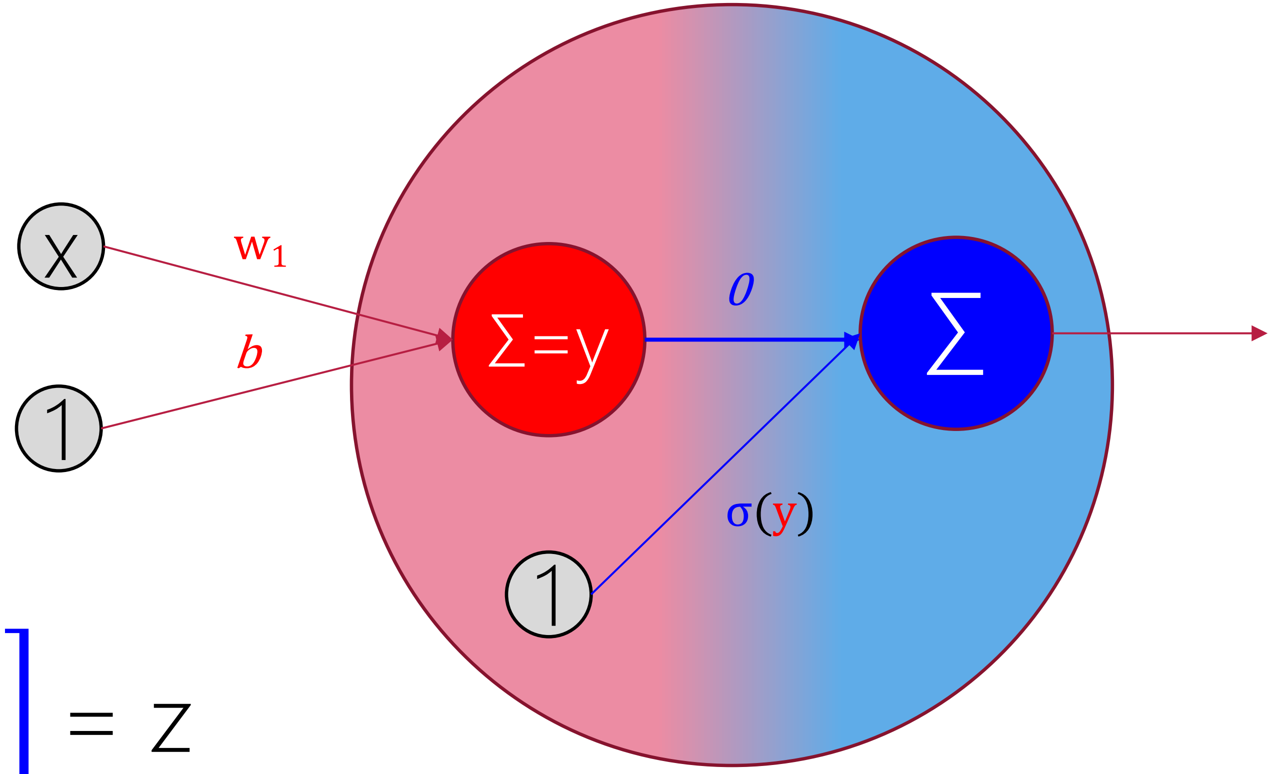


Neural Network

$$h: M \rightarrow O$$

Transferring points (vectors) of source space (M) to target space (O)

$$h(x) = \sigma(x \times w_1 + b)$$



$$[[x\ 1]\begin{bmatrix} w_1 \\ b \end{bmatrix}\ \ 1]\begin{bmatrix} 0 \\ \sigma(x \times w_1 + b) \end{bmatrix} = z$$

Neural Network

$$h: M \rightarrow O$$

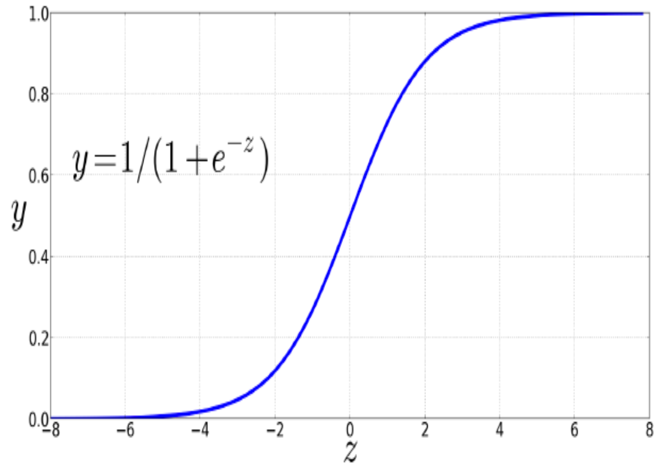
Transferring points (vectors) of source space (M) to target space (O)

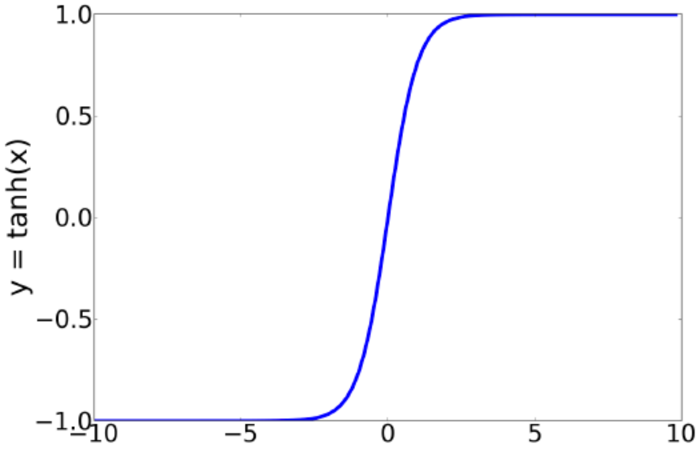
$$h(x) = \sigma(x \times w_1 + b)$$

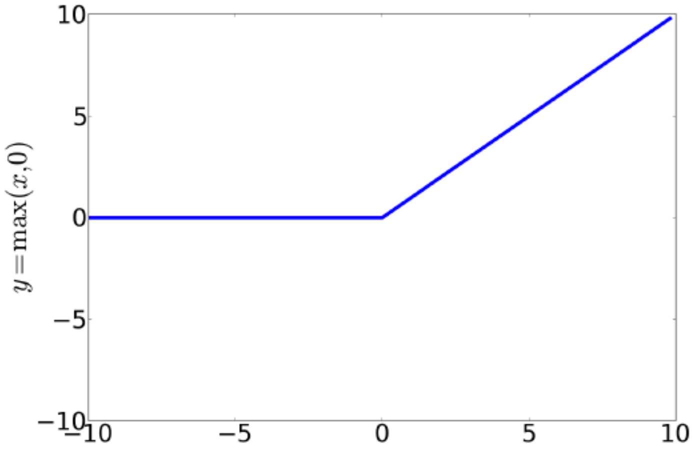
σ is a known non-linear function: Sigmoid, ReLU, Tanh

$$[[x\ 1]\begin{bmatrix} w_1 \\ b \end{bmatrix}\quad 1]\ \begin{bmatrix} 0 \\ \sigma(x \times w_1 + b) \end{bmatrix} = z$$

Neural Network: Activation Function σ







Neural Network

$$h: M=\{0,1\}^2 \rightarrow N=\{0,1\}$$

M = {(0,0), (0,1), (1,0), (1,1)}

N = {0, 1}

h = {((0,0), 0), ((0,1), 0), ((1,0), 0), ((1,1), 1)}

$$[[x_1 \quad x_2 \quad 1]\begin{bmatrix} w_1 \\ w_2 \\ b \end{bmatrix} 1] \begin{bmatrix} 0 \\ \sigma(x_1 \times w_1 + x_2 \times w_2 + b) \end{bmatrix} = z$$

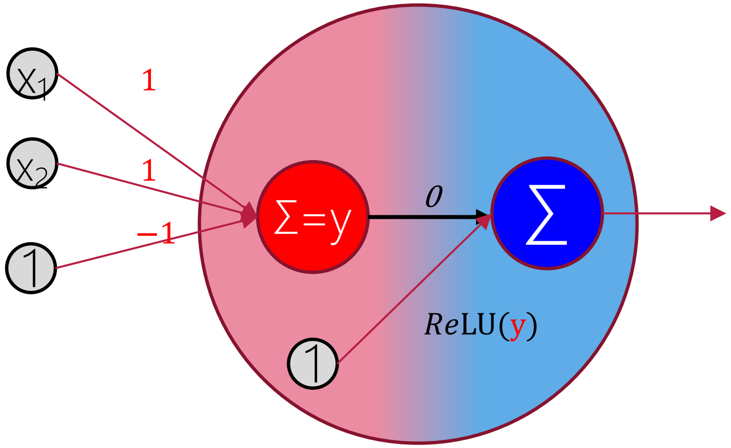
Neural Network

AND: $M=\{0,1\}^2 \rightarrow N=\{0,1\}$

M = {(0,0), (0,1), (1,0), (1,1)}

N = {0, 1}

h = {((0,0), 0), ((0,1), 0), ((1,0), 0), ((1,1), 1)}



$$[[x_1 \;\; x_2 \;\; 1]\begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix} 1]\left[\begin{matrix} 0 \\ max(0, x_1 \times 1 + x_2 \times 1 + -1) \end{matrix}\right] = Z$$

Neural Network

$OR: M=\{0,1\}^2 \rightarrow N=\{0,1\}$

M = {(0,0), (0,1), (1,0), (1,1)}

N = {0, 1}

h = {((0,0), 0), ((0,1), 1), ((1,0), 1), ((1,1), 1)}

$$[[x_1 \quad x_2 \quad 1]\begin{bmatrix} w_1 \\ w_2 \\ b \end{bmatrix} \; 1] \begin{bmatrix} 0 \\ \sigma(x_1 \times w_1 + x_2 \times w_2 + b) \end{bmatrix} = z$$

Neural Network

XOR: M={0,1}^{2} → N={0,1}

M = {(0,0), (0,1), (1,0), (1,1)}

N = {0, 1}

h = {((0,0), 0), ((0,1), 1), ((1,0), 1), ((1,1), 0)}

$$[[x_1 \quad x_2 \quad 1]\begin{bmatrix} w_1 \\ w_2 \\ b \end{bmatrix} \ 1]\begin{bmatrix} 0 \\ \sigma(x_1 \times w_1 + x_2 \times w_2 + b) \end{bmatrix} = z$$

Neural Network

XOR: M={0,1}² → N={0,1,2}² → O={0,1}

M = {(0,0), (0,1), (1,0), (1,1)} → N = {(0,0), (0,1), …, (2,2)} → O = {0, 1}

h = {((0,0), 0), ((0,1), 1), ((1,0), 1), ((1,1), 0)}

$$[[x_1 \quad x_2 \quad 1]\begin{bmatrix} w_1 \\ w_2 \\ b \end{bmatrix} 1] \begin{bmatrix} 0 \\ \sigma(x_1 \times w_1 + x_2 \times w_2 + b) \end{bmatrix} = y_1$$

$$[[x_1 \quad x_2 \quad 1]\begin{bmatrix} w'_1 \\ w'_2 \\ b' \end{bmatrix} 1] \begin{bmatrix} 0 \\ \sigma(x_1 \times w'_1 + x_2 \times w'_2 + b') \end{bmatrix} = y_2$$

$$[[y_1 \quad y_2 \quad 1]\begin{bmatrix} w''_1 \\ w''_2 \\ b'' \end{bmatrix} 1] \begin{bmatrix} 0 \\ \sigma(x_1 \times w''1 + x_2 \times w''2 + b'') \end{bmatrix} = z$$

Neural Network

$$XOR: M=\{0,1\}^2 \rightarrow N=\{0,1,2\}^2 \rightarrow O=\{0,1\}$$

M = {(0,0), (0,1), (1,0), (1,1)} → N = {(0,0), (0,1), …, (2,2)} → O = {0, 1}

h = {((0,0), 0), ((0,1), 1), ((1,0), 1), ((1,1), 0)}

$$[[x_1 \quad x_2 \quad 1]\begin{bmatrix} w_1 & w'1 \\ w_2 & w'2 \\ b & b' \end{bmatrix} \; 1] =$$

$$[x_1 \times w_1 + x_2 \times w_2 + b \quad x_1 \times w'1 + x_2 \times w'2 + b' \quad 1]\begin{bmatrix} 0 & 0 \\ 0 & 0 \\ \sigma(x_1 \times w_1 + x_2 \times w_2 + b) & \sigma(x_1 \times w'1 + x_2 \times w'2 + b') \end{bmatrix} =$$

$$[\underbrace{\sigma(x_1 \times w_1 + x_2 \times w_2 + b)}_{\alpha} \; \underbrace{\sigma(x_1 \times w'1 + x_2 \times w'2 + b')}_{\beta} \quad 1]\begin{bmatrix} w''_1 \\ w''_2 \\ b'' \end{bmatrix} \; 1]\begin{bmatrix} 0 \\ 0 \\ \sigma(\alpha \times w''1 + \beta \times w''2 + b'') \end{bmatrix} = z$$

Neural Network

$$XOR: M=\{0,1\}^2 \rightarrow N=\{0,1,2\}^2 \rightarrow O=\{0,1\}$$

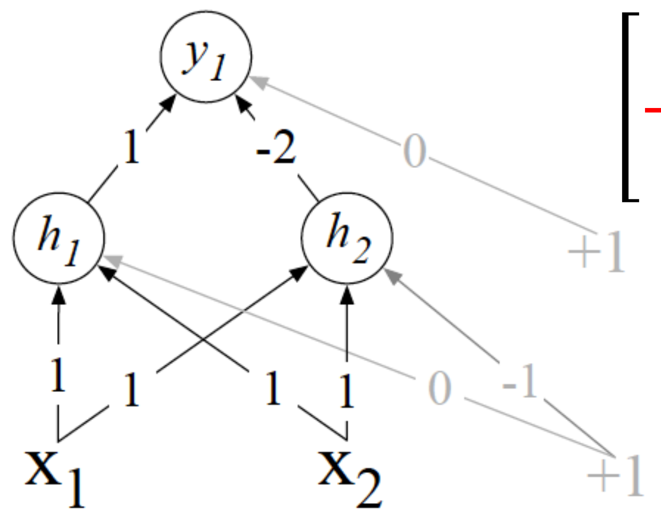
$$M = \{(0,0), (0,1), (1,0), (1,1)\} \rightarrow N = \{(0,0), (0,1), ..., (2,2)\} \rightarrow O = \{0, 1\}$$

$$h = \{((0,0), 0), ((0,1), 1), ((1,0), 1), ((1,1), 0)\}$$

$$[x_1\ \ x_2\ \ 1]\begin{bmatrix} w_1 & w'1 \\ w_2 & w'2 \\ w_3 & w'3 \end{bmatrix} =$$

$$[x_1\ \ x_2\ \ 1]\begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 0 & -1 \end{bmatrix} =$$

$$[h_1=\mathrm{ReLU}(x_1+x_2)\ \ \ h_2=\mathrm{ReLU}(x_1+x_2-1)]$$



$$\begin{bmatrix} 1 \\ -2 \\ 0 \end{bmatrix} = \begin{bmatrix} w''1 \\ w''2 \\ w''3 \end{bmatrix}$$

Figure 7.6 XOR solution after Goodfellow et al. (2016). There are three ReLU units, in two layers; we've called them h_1, h_2 (h for "hidden layer") and y_1. As before, the numbers on the arrows represent the weights w for each unit, and we represent the bias b as a weight on a unit clamped to +1, with the bias weights/units in gray.

Neural Network

$XOR: M=\{0,1\}^2 \rightarrow N=\{0,1,2\}^2 \rightarrow O=\{0,1\}$

M = {(0,0), (0,1), (1,0), (1,1)} → N = {(0,0), (0,1), ..., (2,2)} → O = {0, 1}

(0,0) → (0,0) → (0)
(0,1) → (1,0) → (1)
(1,0) → (1,0) → (1)
(1,1) → (2,1) → (0)

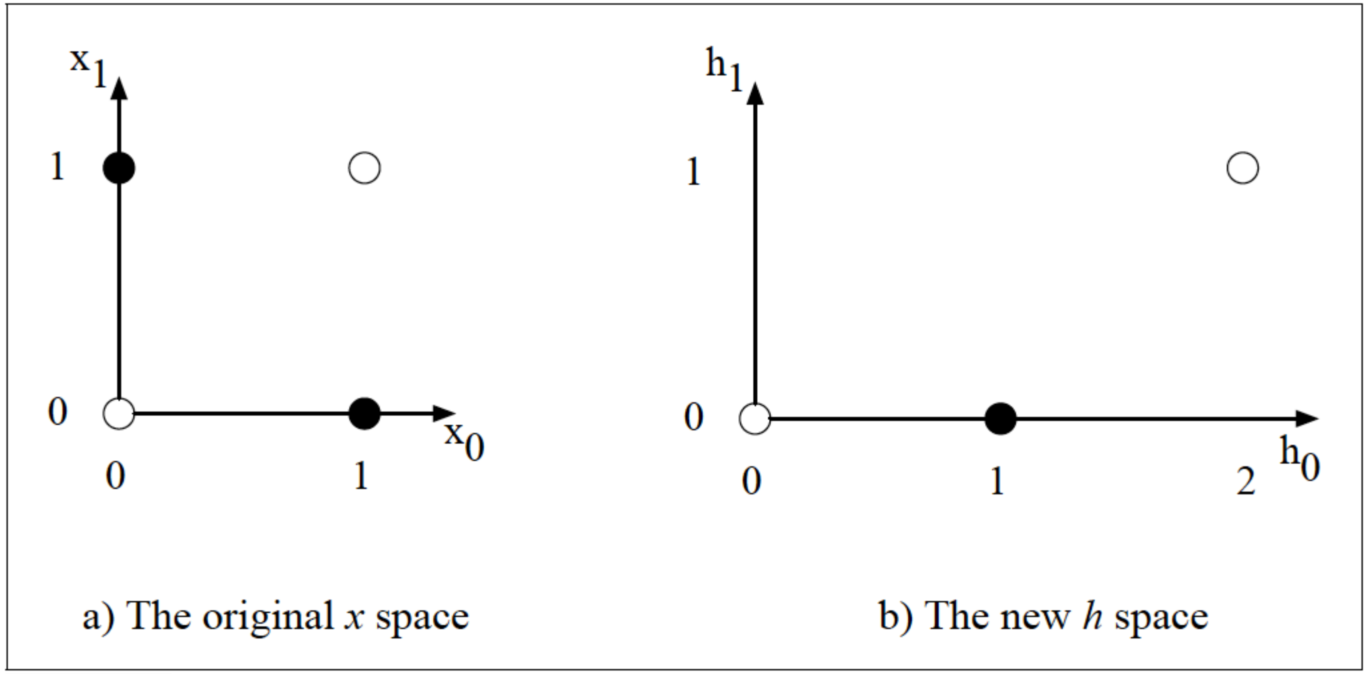


Figure 7.7 The hidden layer forming a new representation of the input. Here is the representation of the hidden layer, h, compared to the original input representation x. Notice that the input point [0 1] has been collapsed with the input point [1 0], making it possible to linearly separate the positive and negative cases of XOR. After Goodfellow et al. (2016).

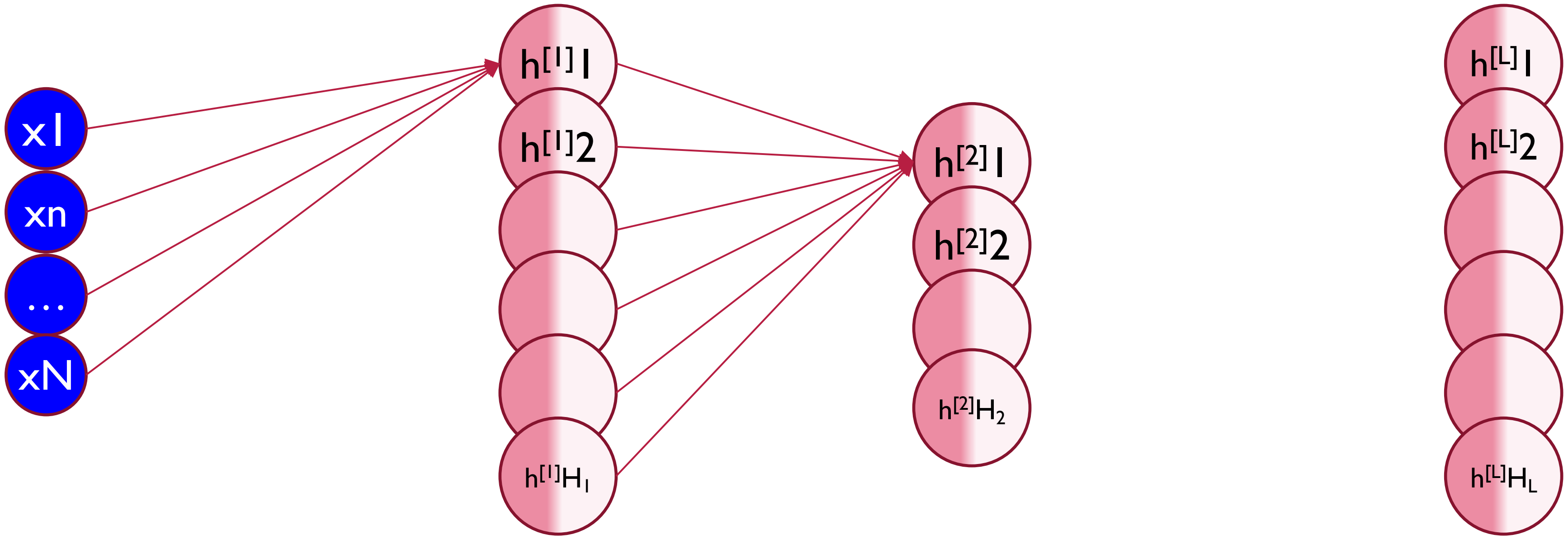
Universal Approximation Theorem

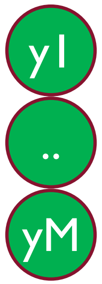
http://neuralnetworksanddeeplearning.com/chap4.html

Be carful, it shows the existence (power) of neural nets, but it does not show which architecture is the function!

How many transformation (#layers)?
To what intermediate space (#nodes)?

Neural Network: Feed-forward





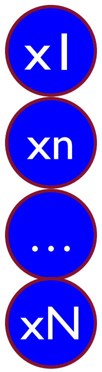
$$\mathbf{y} = \sigma(\mathbf{h}^{[L]}\mathbf{W}^{[L+1]} + \mathbf{b}^{[L+1]})$$

$\mathbf{W}^{[L]}: H_1 \times M$

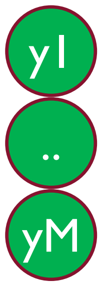
$$\mathbf{h}^{[1]} = \sigma(\mathbf{x}\,\mathbf{W}^{[1]} + \mathbf{b}^{[1]}) \qquad \mathbf{h}^{[2]} = \sigma(\mathbf{h}^{[1]}\mathbf{W}^{[2]} + \mathbf{b}^{[2]}) \qquad \mathbf{h}^{[L]} = \sigma(\mathbf{h}^{[L-1]}\mathbf{W}^{[L]} + \mathbf{b}^{[L]})$$

$\mathbf{x}: 1 \times N$ $\quad$ $\mathbf{W}^{[1]}: N \times H_1$ $\quad$ $\mathbf{W}^{[2]}: H_1 \times H_2$ $\quad$ $\mathbf{W}^{[L]}: H_{L-1} \times HL$

Neural Network: Feed-forward Train



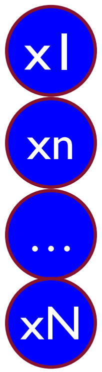
The only known is the data (input: X, output: Y)
Given the input point X, the net should land it to Y.



If the net should land it to f(X) = Y’ → error!
Correct the net to make Y’ close to Y.

How?

Neural Network: Feed-forward Train

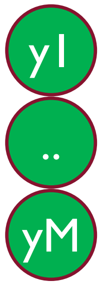


How?

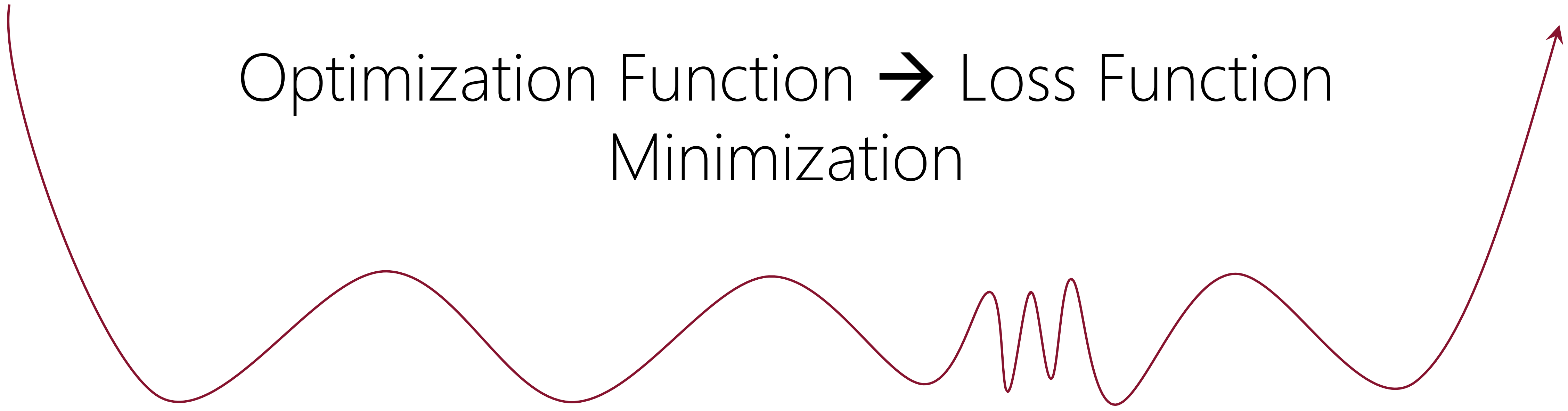
By minimizing the overall error.

What is error?

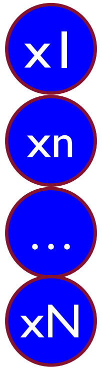
e = distance (Y, f(X)); for all (X, Y)



Optimization Function → Loss Function Minimization



Neural Network: Gradient

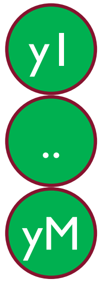


How?

By minimizing the overall error.

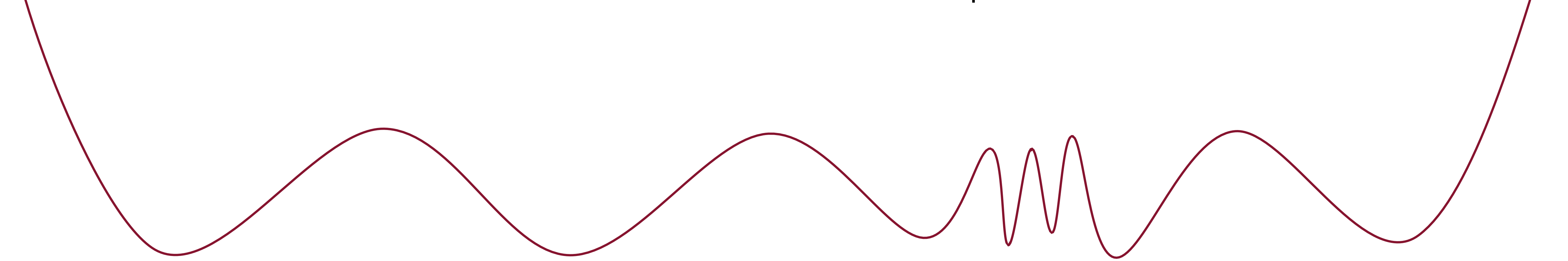
What is error?

e = distance (Y, f(X)); for all (X, Y)



We have to change f such that Y and f(X) get closer and closer!

Y and X is constant in this procedure.



Neural Network: Gradient

x1
xn
...
xN

How?

By minimizing the overall error.

What is error?

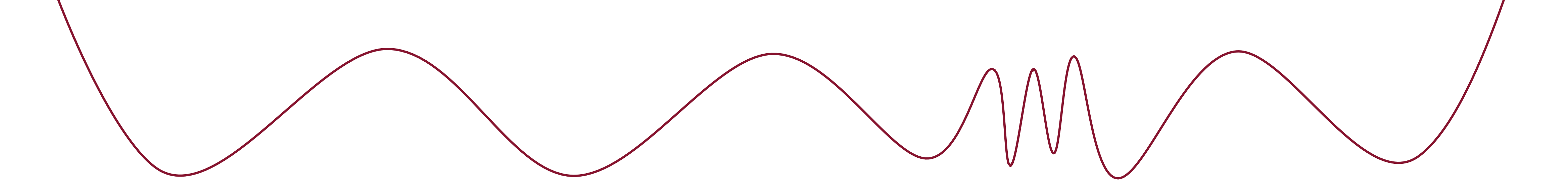
e = distance (Y, f(X)); for all (X, Y)

y1
..
yM

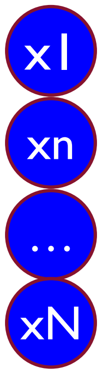
We have to change parameters such that Y and f(X) get closer and closer!

Y and X is constant in this procedure.

$e(\boldsymbol{\Theta}) = \text{distance}(Y, f(X; \boldsymbol{\Theta}))$



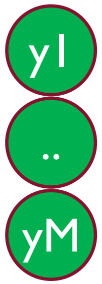
Neural Network: Gradient



$$f(x) = 3x+1 \rightarrow f(x; \boldsymbol{\Theta}) = f(x; [3,1]) = 3x+1$$

$$f(x; [w_1,w_2]) = w_1 x + w_2$$

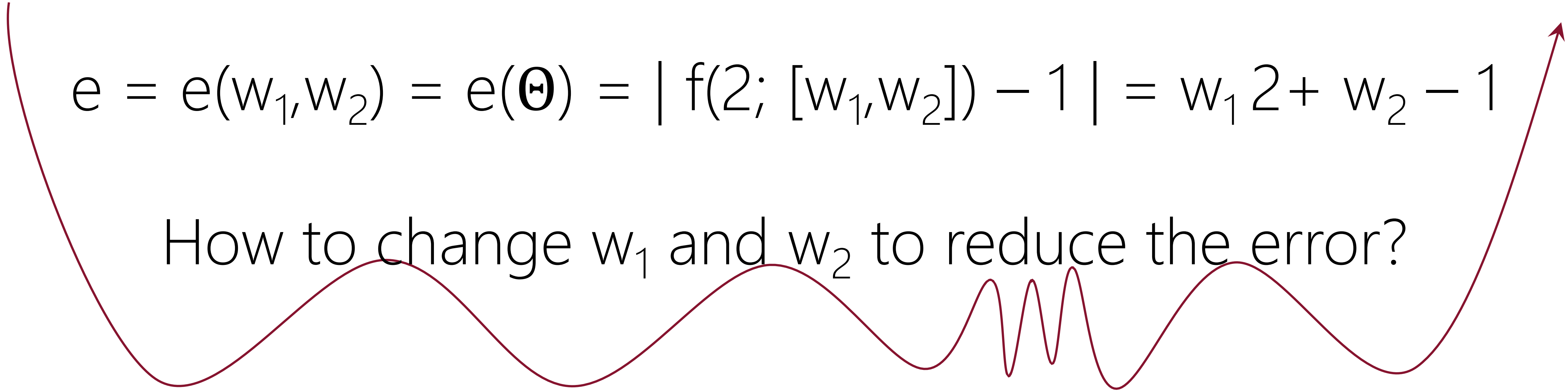
$$(x, y) = (2, 1)$$



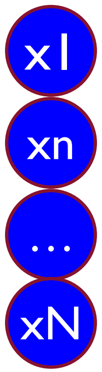
$$f(2; [w_1,w_2]) = w_1 2 + w_2$$

$$e = e(w_1,w_2) = e(\boldsymbol{\Theta}) = | f(2; [w_1,w_2]) - 1 | = w_1 2 + w_2 - 1$$

How to change w_1 and w_2 to reduce the error?



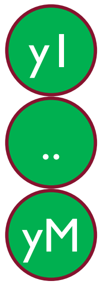
Neural Network: Gradient Descent



$$f(x) = 3x+1 \rightarrow f(x;\, \boldsymbol{\Theta}) = f(x;\, [3,1]) = 3x+1$$

$$f(x;\, [w_1,w_2]) = w_1 x + w_2$$

$$(x,\, y) = (2,\, 1)$$



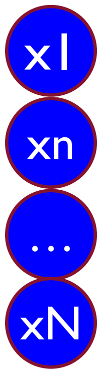
$$f(2;\, [w_1,w_2]) = w_1 2 + w_2$$

$$e = e(w_1,w_2) = e(\boldsymbol{\Theta}) = |\, f(2;\, [w_1,w_2]) - 1 \,| = w_1 2 + w_2 - 1$$

$$w_1 = w_1 - \eta \frac{\partial e}{\partial w_1} = w_1 - 0.1 \times 2 = w_1 - 0.2$$

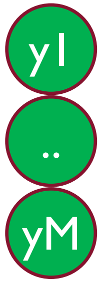
$$w_2 = w_2 - \eta \frac{\partial e}{\partial w_2} = w_2 - 0.1 \times 1 = w_1 - 0.1$$

Neural Network: Gradient Descent

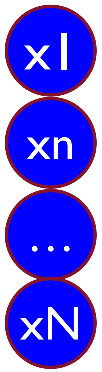


$$f(x;\ \boldsymbol{\Theta}) = f(x;\ [2.8, 0.9]) = 2.8x+0.9 =$$
$$f(x;\ \boldsymbol{\Theta}) = f(x;\ [2.6, 0.8]) = 2.6x+0.8 =$$
$$f(x;\ \boldsymbol{\Theta}) = f(x;\ [2.4, 0.7]) = 2.4x+0.7 =$$

...

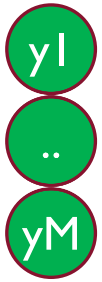


Neural Network: Gradient Descent



Convex Error Function

Many Parameters (all matrices' elements)



Backpropagation!

Computation Graph

Forward Pass

Backward Differentiation.

Neural Language Model

Neural Language Model

Bigram Language Model

$$P(w_i | w_{i-1}) = \frac{\#(wi,\ wi_{-1})}{\#(wi)}$$

$$f: (\{0,1\}^{|V|})^2 \longrightarrow R^{[0,1]};\ f(w_i,\ w_{i-1}) = P(w_i | w_{i-1})$$

Input: one-hot vector

w_{i-1} = [0 0 0 … 1 0… 0 0]

w_i = [0 0 0 … 0 1 … 0 0]

Output:

[y]

Neural Language Model

Bigram Language Model

$$P(w_i | w_{i-1}) = \frac{\#(wi, wi_{-1})}{\#(wi)}$$

$$f: (R^{|V|})^2 \longrightarrow R^{[0,1]};\ f(w_i, w_{i-1}) = P(w_i | w_{i-1})$$

Input: sparse semantic vector

w_{i-1} = [1 2 3 ... 1 2... 1 0]

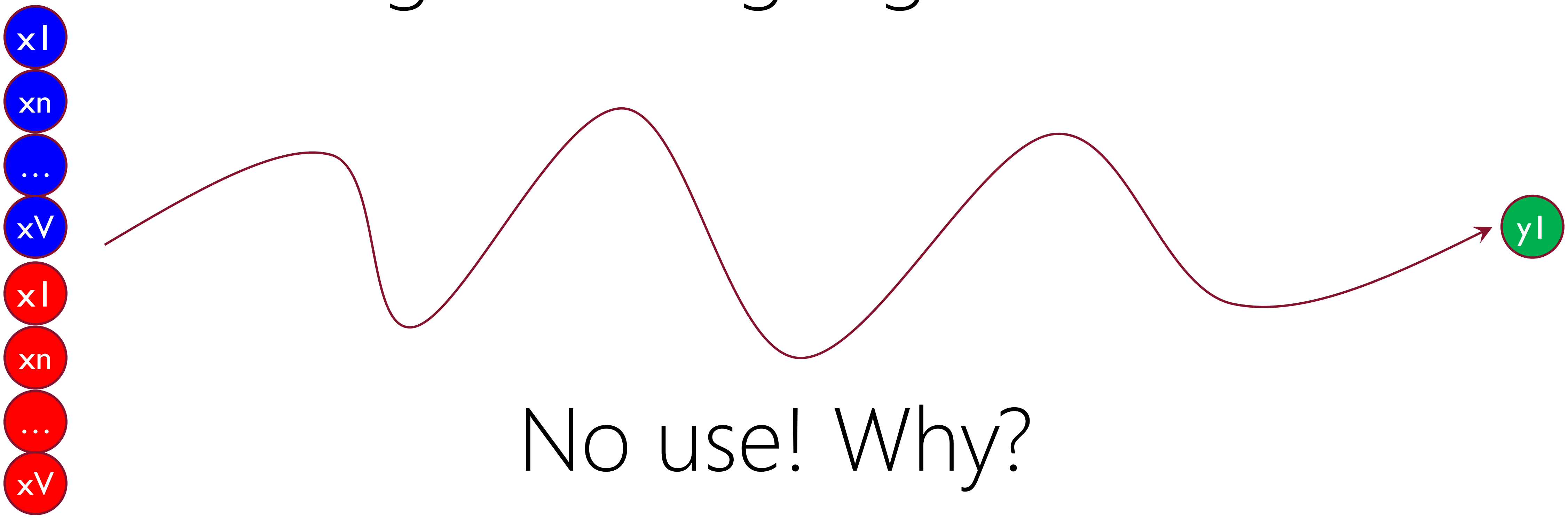
w_i = [4 0 2 ... 0 5 ... 3 0]

Output:

[y]

Neural Language Model

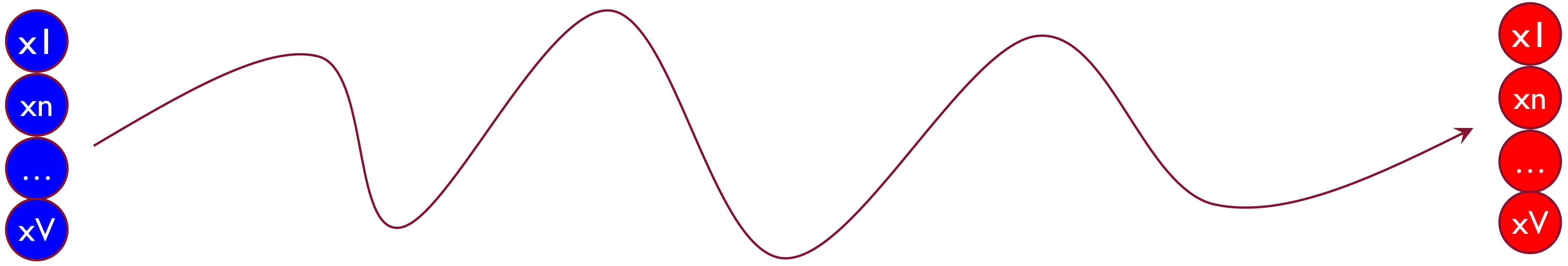
Bigram Language Model



No use! Why?

Neural Language Model

Bigram Language Model



$P(w_i | w_{i-1})$ = ? Unknown Initially

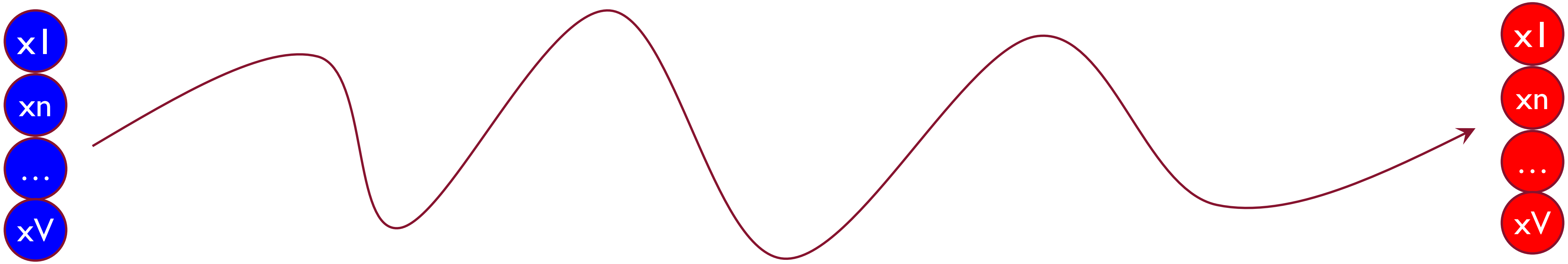
While enumerating the text stream, when I see $[w_{i-1}\ w_i]$ then

$f: \{0,1\}^{|V|} \rightarrow \{0,1\}^{|V|};\ f(w_{i-1}) \rightarrow w_i$

$[0\ 0\ 0 \ldots 1\ 0 \ldots 0\ 0\ 0] \rightarrow [0\ 0\ 0 \ldots 0\ 1 \ldots 0\ 0\ 0]$

Neural Language Model

Bigram Language Model



$P(w_i | w_{i-1})$ = ? Unknown Initially

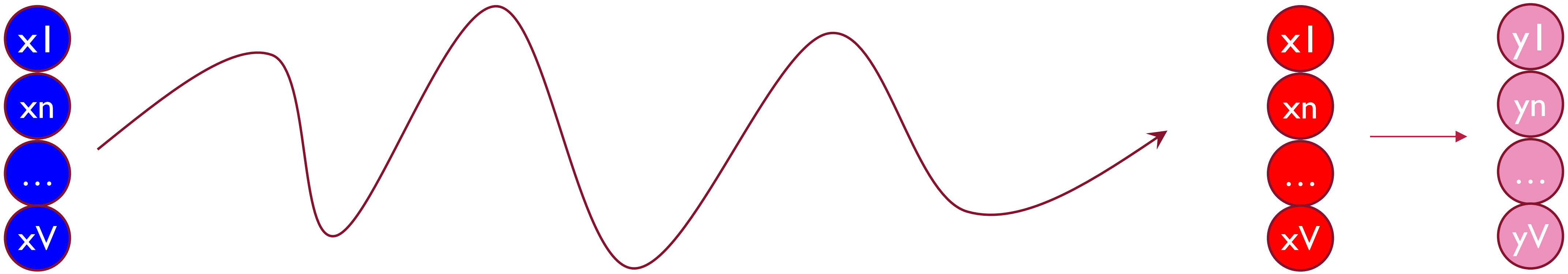
While enumerating the text stream, when I see $[w_{i-1}\ w_i]$ then

$$f: \{0,1\}^{|V|} \rightarrow R^{|V|};\ f(w_{i-1}) \rightarrow \#(w_{i-1}w_i)$$

$$[0\ 0\ 0\ \ldots\ 1\ 0\ \ldots\ 0\ 0\ 0] \rightarrow [0\ 1\ 0\ \ldots\ 0\ 2\ \ldots\ 1\ 0\ 0]$$

Neural Language Model

Bigram Language Model



$P(w_i | w_{i-1})$ = ? Unknown Initially

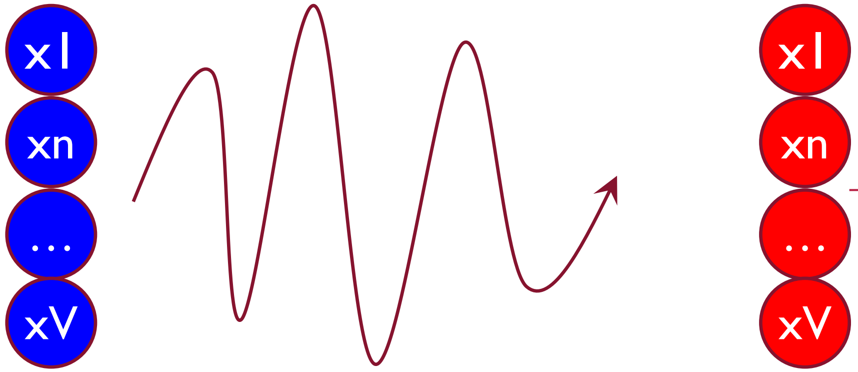
While enumerating the text stream, when I see $[w_{i-1}\ w_i]$ then

$f: \{0,1\}^{|V|} \rightarrow R^{|V|}$; $f(w_{i-1}) \rightarrow \#(w_{i-1}w_i) \rightarrow$ Normalized

$[0\ 0\ 0 \ldots 1\ 0 \ldots 0\ 0\ 0] \rightarrow [0\ 1\ 0 \ldots 0\ 2 \ldots 1\ 0\ 0] \rightarrow$ Softmax

Neural Language Model

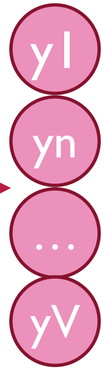
Bigram Language Model



$$y_i = Softmax(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{V} e^{x_j}}$$

$$0 \leq y_i \leq 1$$

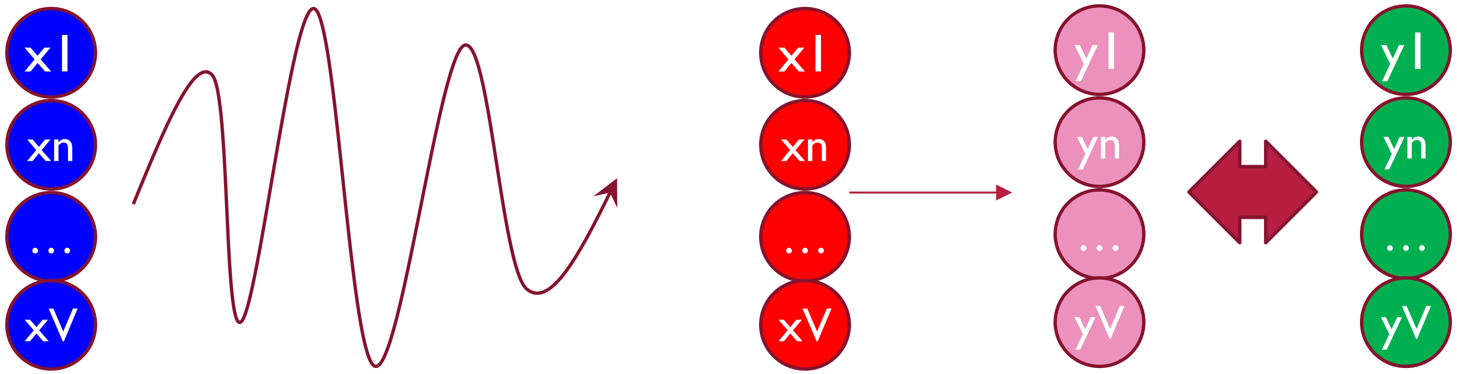
$$\sum_{j=1}^{V} y_i = 1$$



f: f: $\{0,1\}^{|V|} \rightarrow [0,1]^{|V|}$; $f(w_{i-1}) = P(w_i | w_{i-1})$

Neural Language Model

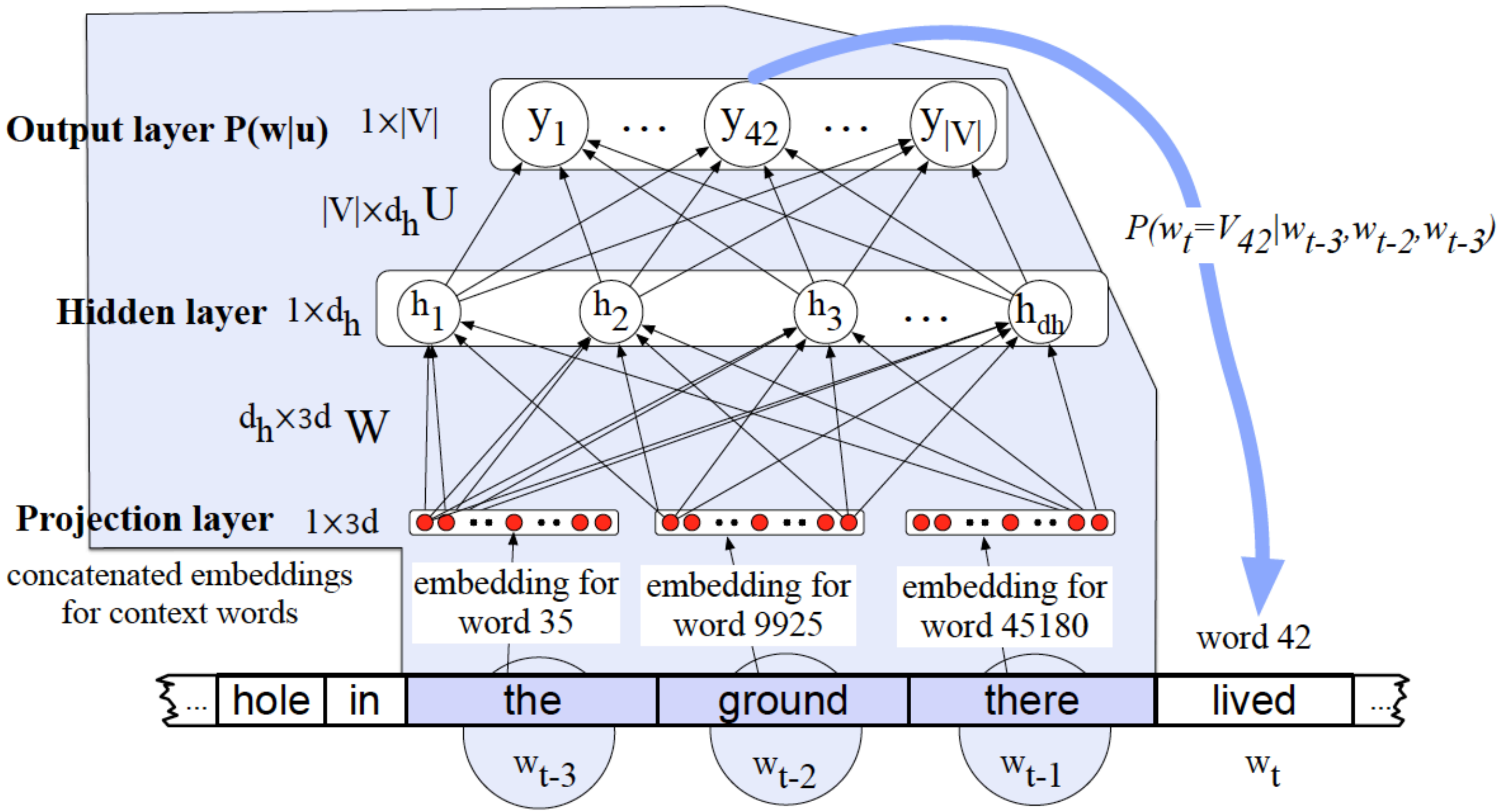
Bigram Language Model



$$\text{Error} = \text{Loss} = -\,\text{Log}\, P(w_i |\, w_{i-1})$$

Neural Language Model

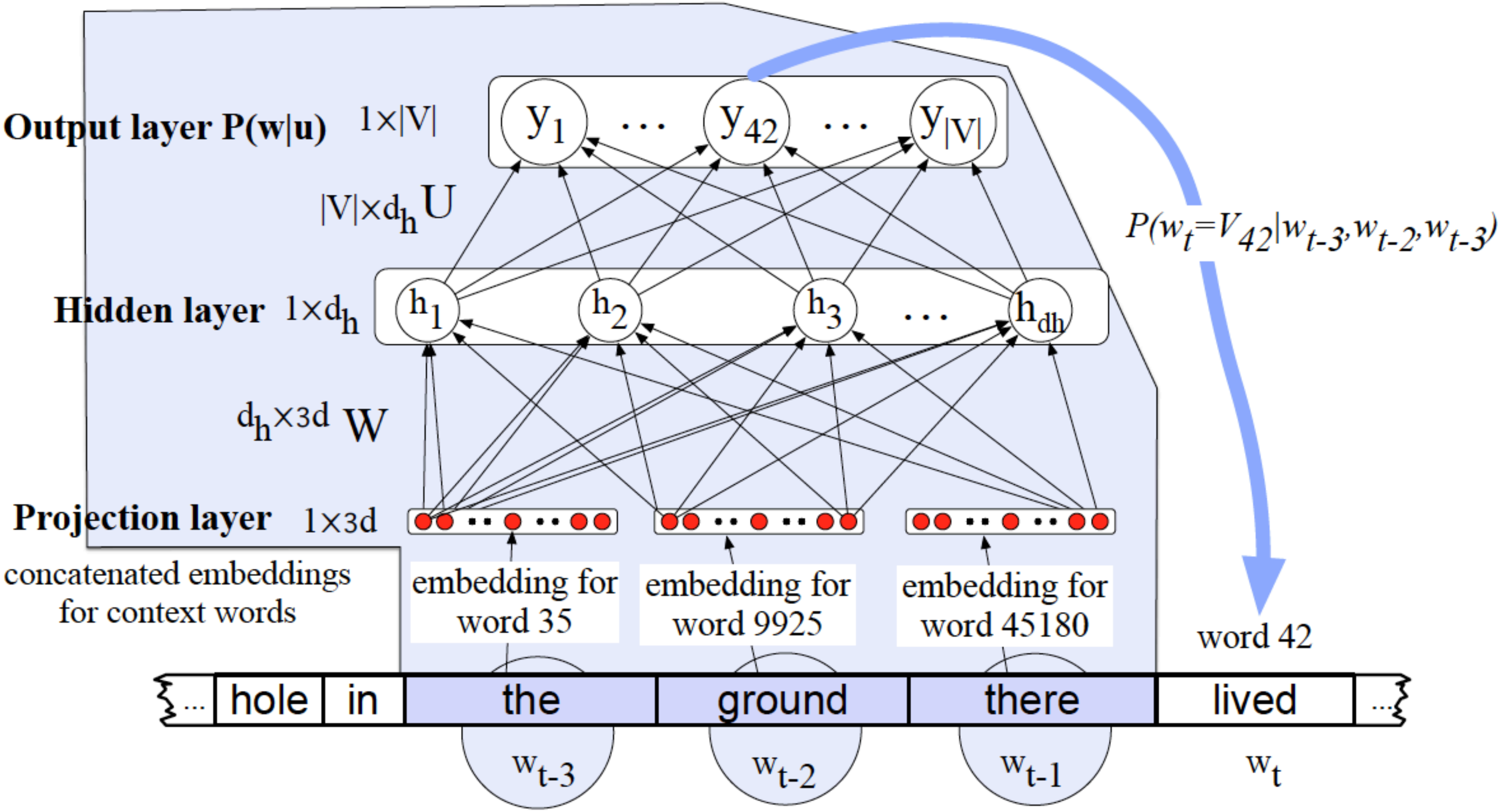
Bengio, Yoshua, et al. "A neural probabilistic language model." The journal of machine learning research 3 (2003): 1137-1155.



N-gram or Bag-of-Word?

Neural Language Model

Bengio, Yoshua, et al. "A neural probabilistic language model." The journal of machine learning research 3 (2003): 1137-1155.



Bottleneck → Softmax → Hierarchical Softmax

Neural Language Model