

NLP + RNN

applekoong@naver.com 김정훈



- **Linear Regression**
- **Gradient Descent Algorithm**
- **Multi-Variable Linear Regression**
- **Logistic Classification**
- **Softmax, Cross-Entropy, FC**
- **NLTK**
- **KoNLPy**
- **Word Embedding**
- **NLP**
- **gensim**
- **RNN - LSTM, GRU**
- **RNN - TensorFlow API**



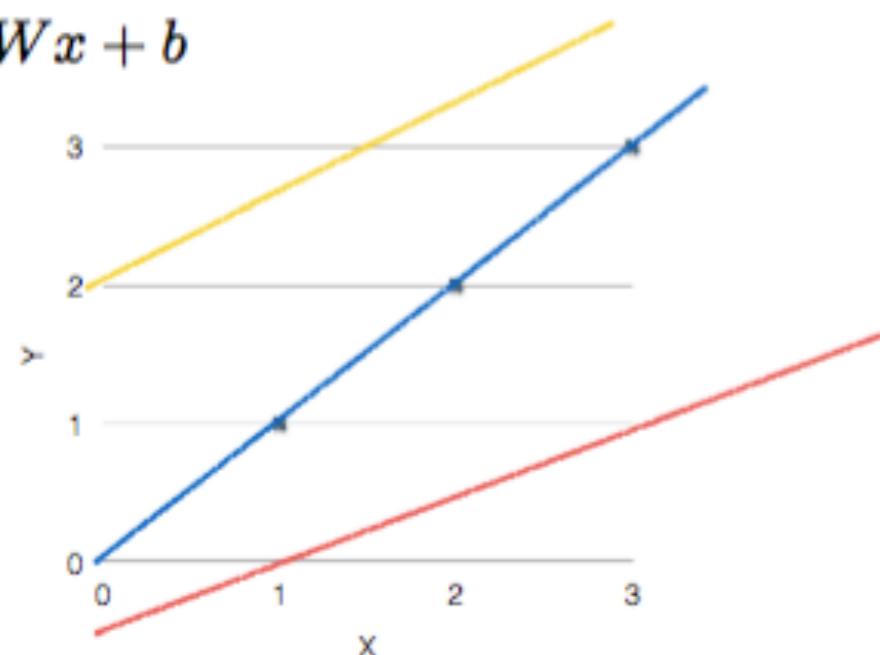
DEEP LEARNING

LINEAR REGRESSION

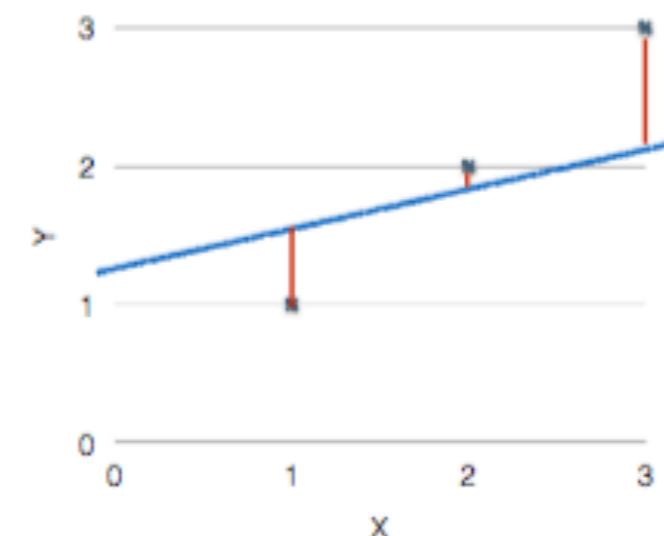
좋은 가설?

(Linear) Hypothesis

$$H(x) = Wx + b$$



Which hypothesis is better?



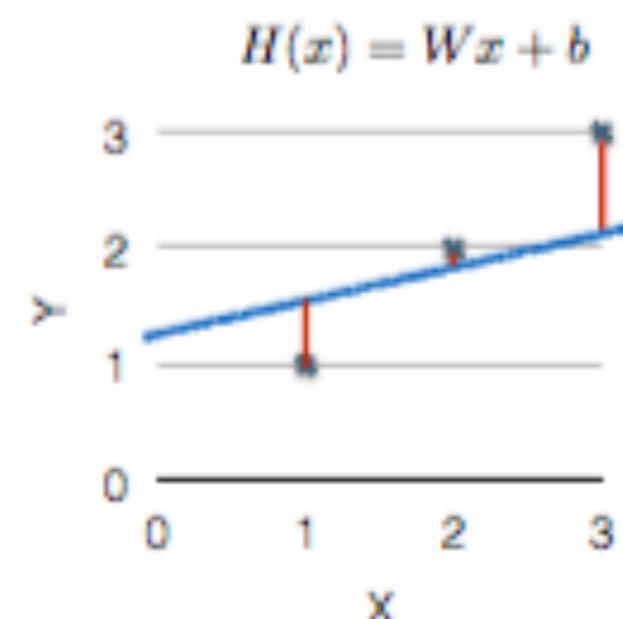
직선으로부터 점까지의 거리 계산

Cost function

- How fit the line to our (training) data

$$\frac{(H(x^{(1)}) - y^{(1)})^2 + (H(x^{(2)}) - y^{(2)})^2 + (H(x^{(3)}) - y^{(3)})^2}{3}$$

$$cost = \frac{1}{m} \sum_{i=1}^m (H(x^{(i)}) - y^{(i)})^2$$



COST 수식

$$cost = \frac{1}{m} \sum_{i=1}^m (H(x^{(i)}) - y^{(i)})^2$$

$$H(x) = Wx + b$$

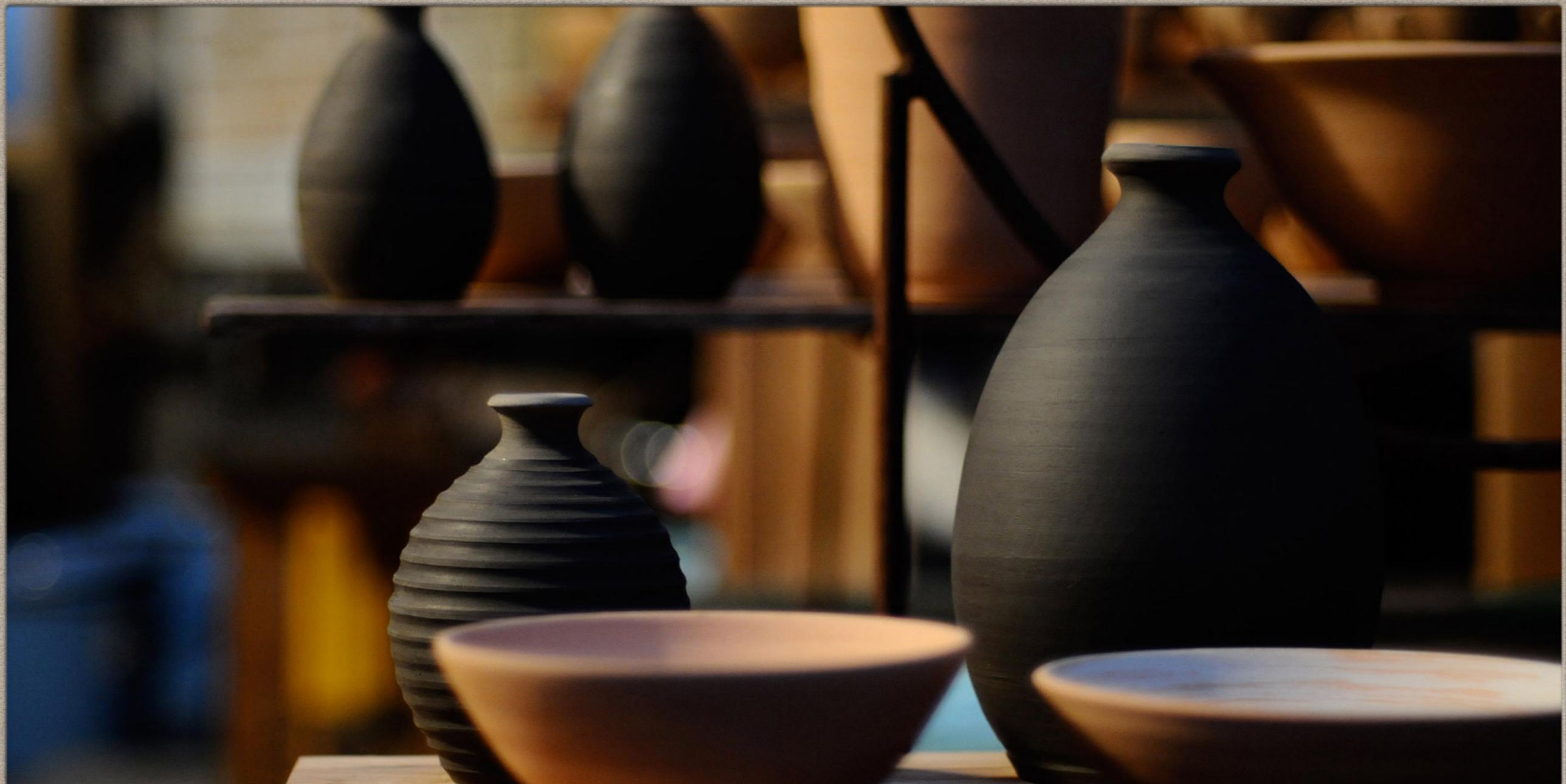
$$cost(W, b) = \frac{1}{m} \sum_{i=1}^m (H(x^{(i)}) - y^{(i)})^2$$

최저 COST를 만드는 W와 B는?

- ▶ $H(x) = wx + b$
- ▶ $x = [1, 2, 3]$
 $y = [1, 2, 3]$
- ▶ w와 b가 아래와 같다면?
 1. $w = 1/2, b = 0$
 2. $w = 0, b = 2$
 3. $w = 1, b = 1/2$
 4. $w = 1, b = 1$

THE GOAL OF LINEAR REGRESSION

- ▶ Cost를 최소로 만드는
W(Weight)와 b(bias)를 찾는 것.
- ▶ Linear Regression의 목표는 곧 머신러닝의 목표!



DEEP LEARNING

GRADIENT DESCENT ALGORITHM

HOW TO HIDE BIAS?

Hypothesis and Cost

$$H(x) = Wx + b$$

$$\text{cost}(W, b) = \frac{1}{m} \sum_{i=1}^m (H(x^{(i)}) - y^{(i)})^2$$

Simplified hypothesis

$$H(x) = Wx$$

$$\text{cost}(W) = \frac{1}{m} \sum_{i=1}^m (Wx^{(i)} - y^{(i)})^2$$

COST FUNCTION GRAPH

What $\text{cost}(W)$ looks like?

$$\text{cost}(W) = \frac{1}{m} \sum_{i=1}^m (Wx^{(i)} - y^{(i)})^2$$

X	Y
1	1
2	2
3	3

- $W=1, \text{cost}(W)=0$

$$\frac{1}{3}((1 * 1 - 1)^2 + (1 * 2 - 2)^2 + (1 * 3 - 3)^2)$$

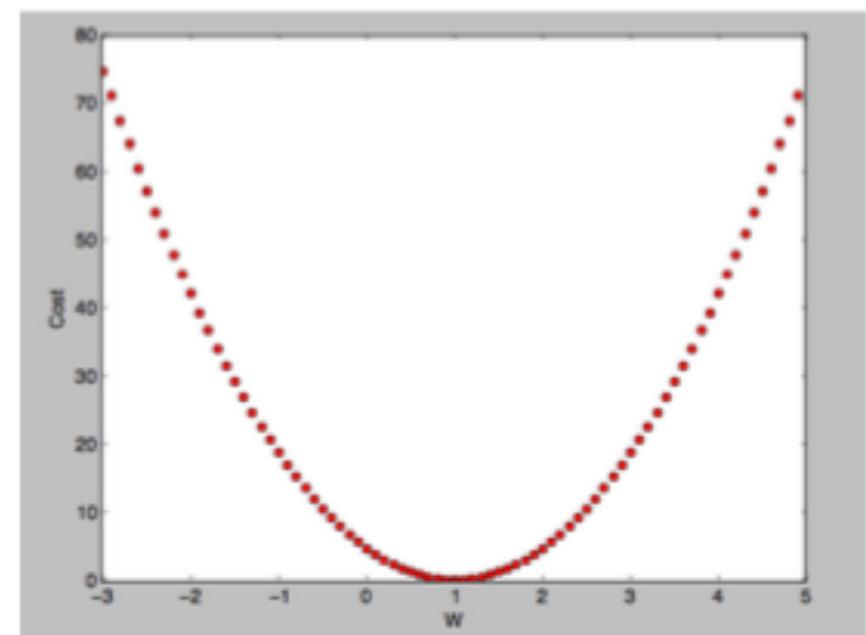
- $W=0, \text{cost}(W)=4.67$

$$\frac{1}{3}((0 * 1 - 1)^2 + (0 * 2 - 2)^2 + (0 * 3 - 3)^2)$$

- $W=2, \text{cost}(W)=?$

How to minimize cost?

$$\text{cost}(W) = \frac{1}{m} \sum_{i=1}^m (Wx^{(i)} - y^{(i)})^2$$

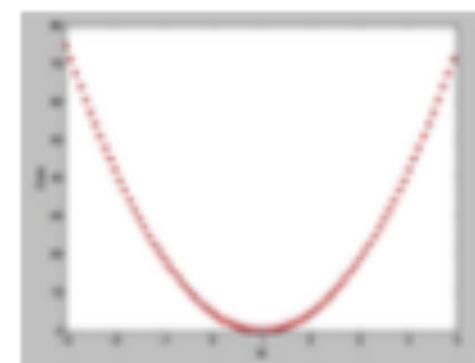


GRADIENT DESCENT ALGORITHM

- Minimize cost function
- Gradient descent is used many minimization problems
- For a given cost function, $\text{cost}(W, b)$, it will find W, b to minimize cost
- It can be applied to more general function: $\text{cost}(w_1, w_2, \dots)$

HOW IT WORKS?

- Start with initial guesses
 - Start at 0,0 (or any other value)
 - Keeping changing W and b a little bit to try and reduce $\text{cost}(W, b)$
- Each time you change the parameters, you select the gradient which reduces $\text{cost}(W, b)$ the most possible
- Repeat
- Do so until you converge to a local minimum
- Has an interesting property
 - Where you start can determine which minimum you end up



FORMAL DEFINITION

$$\text{cost}(W) = \frac{1}{m} \sum_{i=1}^m (Wx^{(i)} - y^{(i)})^2$$

$$W := W - \alpha \frac{\partial}{\partial W} \frac{1}{2m} \sum_{i=1}^m (Wx^{(i)} - y^{(i)})^2$$

$$\text{cost}(W) = \frac{1}{2m} \sum_{i=1}^m (Wx^{(i)} - y^{(i)})^2$$

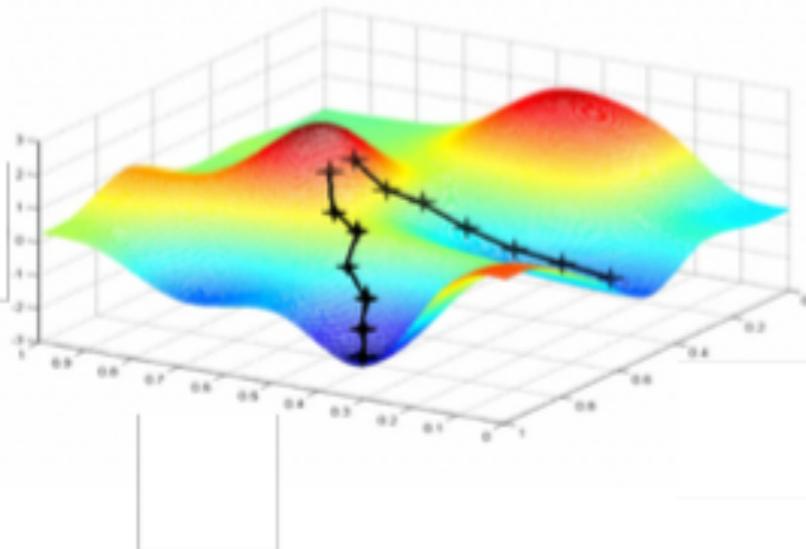
$$W := W - \alpha \frac{1}{2m} \sum_{i=1}^m 2(Wx^{(i)} - y^{(i)})x^{(i)}$$

$$W := W - \alpha \frac{\partial}{\partial W} \text{cost}(W)$$

$$W := W - \alpha \frac{1}{m} \sum_{i=1}^m (Wx^{(i)} - y^{(i)})x^{(i)}$$

CONVEX FUNCTION

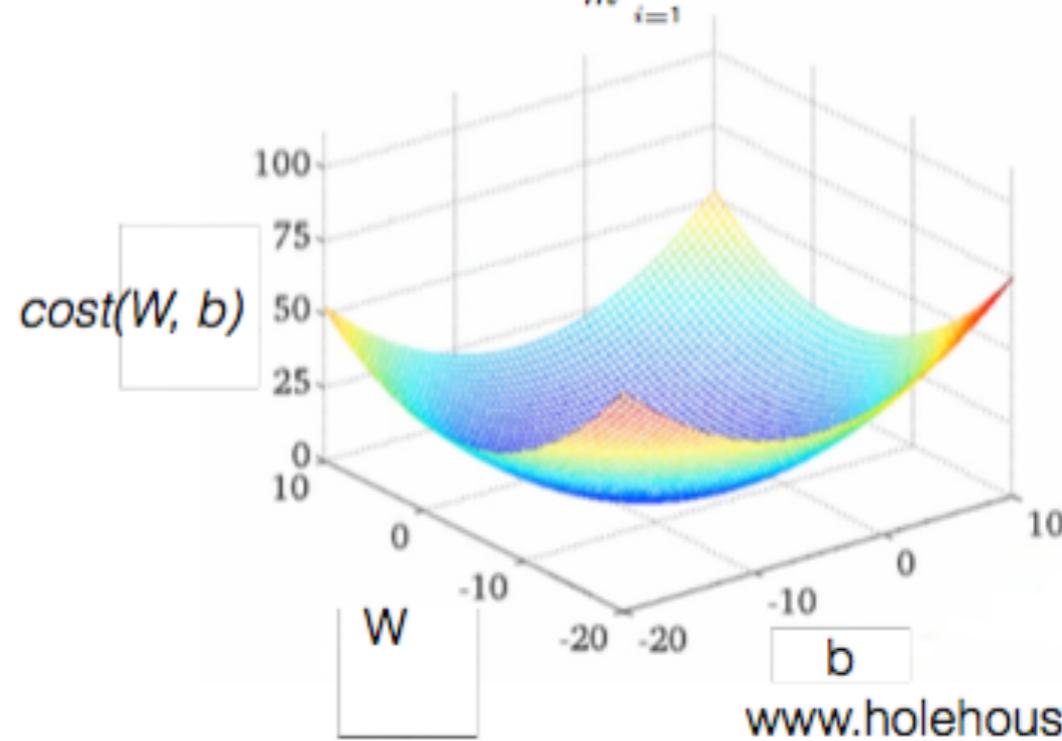
Convex function



www.holehouse.org/mlclass/

Convex function

$$cost(W, b) = \frac{1}{m} \sum_{i=1}^m (H(x^{(i)}) - y^{(i)})^2$$



www.holehouse.org/mlclass/



DEEP LEARNING

MULTI-VARIABLE LINEAR REGRESSION

MULTI-VARIABLE LINEAR REGRESSION

MULTI FEATURES

one-variable
one-feature

x (hours)	y (score)
10	90
9	80
3	50
2	60
11	40

multi-variable/feature

x1 (hours)	x2 (attendance)	y (score)
10	5	90
9	5	80
3	2	50
2	4	60
11	1	40

HYPOTHESIS AND COST FUNCTION

Hypothesis

$$H(x) = Wx + b$$

$$H(x_1, x_2) = w_1x_1 + w_2x_2 + b$$

Cost function

$$H(x_1, x_2) = w_1x_1 + w_2x_2 + b$$

$$cost(W, b) = \frac{1}{m} \sum_{i=1}^m (H(x^{(i)}) - y^{(i)})^2$$

MATRIX AND TRANSPOSE

Matrix multiplication

"Dot Product"

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \times \begin{bmatrix} 7 & 8 \\ 9 & 10 \\ 11 & 12 \end{bmatrix} = \begin{bmatrix} 58 \end{bmatrix}$$

Transpose

$$\begin{bmatrix} 6 & 4 & 24 \\ 1 & -9 & 8 \end{bmatrix}^T = \begin{bmatrix} 6 & 1 \\ 4 & -9 \\ 24 & 8 \end{bmatrix}$$

MULTI-FEATURES

**Predicting exam score:
regression using three inputs (x_1, x_2, x_3)**

multi-variable/feature

x_1 (quiz 1)	x_2 (quiz 2)	x_3 (midterm 1)	Y (final)
73	80	75	152
93	88	93	185
89	91	90	180
96	98	100	196
73	66	70	142

Test Scores for General Psychology

HYPOTHESIS USING MATRIX (1)

$$w_1x_1 + w_2x_2 + w_3x_3 + \dots + w_nx_n$$

$$(x_1 \quad x_2 \quad x_3) \cdot \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} = (x_1w_1 + x_2w_2 + x_3w_3)$$

$$H(X) = XW$$

HYPOTHESIS USING MATRIX (2)

$$H(x_1, x_2, x_3) = x_1w_1 + x_2w_2 + x_3w_3$$

x₁	x₂	x₃	Y
73	80	75	152
93	88	93	185
89	91	90	180
96	98	100	196
73	66	70	142

Test Scores for General Psychology

$$(x_1 \quad x_2 \quad x_3) \cdot \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} = (x_1w_1 + x_2w_2 + x_3w_3)$$

$$H(X) = XW$$

HYPOTHESIS USING MATRIX (3)

x_1	x_2	x_3	Y
73	80	75	152
93	88	93	185
89	91	90	180
96	98	100	196
73	66	70	142

Hypothesis using matrix

$$w_1x_1 + w_2x_2 + w_3x_3 + \dots + w_nx_n$$

$$\begin{pmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \\ x_{41} & x_{42} & x_{43} \\ x_{51} & x_{52} & x_{53} \end{pmatrix} \cdot \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} = \begin{pmatrix} x_{11}w_1 + x_{12}w_2 + x_{13}w_3 \\ x_{21}w_1 + x_{22}w_2 + x_{23}w_3 \\ x_{31}w_1 + x_{32}w_2 + x_{33}w_3 \\ x_{41}w_1 + x_{42}w_2 + x_{43}w_3 \\ x_{51}w_1 + x_{52}w_2 + x_{53}w_3 \end{pmatrix}$$

$$H(X) = XW$$

HYPOTHESIS USING MATRIX (4)

$$\begin{pmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \\ x_{41} & x_{42} & x_{43} \\ x_{51} & x_{52} & x_{53} \end{pmatrix} \cdot \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} = \begin{pmatrix} x_{11}w_1 + x_{12}w_2 + x_{13}w_3 \\ x_{21}w_1 + x_{22}w_2 + x_{23}w_3 \\ x_{31}w_1 + x_{32}w_2 + x_{33}w_3 \\ x_{41}w_1 + x_{42}w_2 + x_{43}w_3 \\ x_{51}w_1 + x_{52}w_2 + x_{53}w_3 \end{pmatrix}$$

[5, 3]

[3, 1]

[5, 1]

$$H(X) = XW$$

HYPOTHESIS USING MATRIX (5)

$$\begin{pmatrix} \mathbf{x} \end{pmatrix} * \begin{pmatrix} \mathbf{w} \end{pmatrix} = \begin{pmatrix} \mathbf{H(x)} \end{pmatrix}$$

[5, 3]

[?, ?]

[5, 1]

$$H(X) = XW$$

HYPOTHESIS USING MATRIX (6)

$$\begin{pmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \\ x_{41} & x_{42} & x_{43} \\ x_{51} & x_{52} & x_{53} \end{pmatrix} \cdot \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} = \begin{pmatrix} x_{11}w_1 + x_{12}w_2 + x_{13}w_3 \\ x_{21}w_1 + x_{22}w_2 + x_{23}w_3 \\ x_{31}w_1 + x_{32}w_2 + x_{33}w_3 \\ x_{41}w_1 + x_{42}w_2 + x_{43}w_3 \\ x_{51}w_1 + x_{52}w_2 + x_{53}w_3 \end{pmatrix}$$

[n, 3]

[3, 1]

[n, 1]

$$H(X) = XW$$

HYPOTHESIS USING MATRIX (7)

$$\begin{pmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \\ x_{41} & x_{42} & x_{43} \\ x_{51} & x_{52} & x_{53} \end{pmatrix} \cdot ? = \begin{pmatrix} x_{11}w_{11} + x_{12}w_{21} + x_{13}w_{31} & x_{11}w_{12} + x_{12}w_{22} + x_{13}w_{32} \\ x_{21}w_{11} + x_{22}w_{21} + x_{23}w_{31} & x_{21}w_{12} + x_{22}w_{22} + x_{23}w_{32} \\ x_{31}w_{11} + x_{32}w_{21} + x_{33}w_{31} & x_{31}w_{12} + x_{32}w_{22} + x_{33}w_{32} \\ x_{41}w_{11} + x_{42}w_{21} + x_{43}w_{31} & x_{41}w_{12} + x_{42}w_{22} + x_{43}w_{32} \\ x_{51}w_{11} + x_{52}w_{21} + x_{53}w_{31} & x_{51}w_{12} + x_{52}w_{22} + x_{53}w_{32} \end{pmatrix}$$

?

[n, 3]
[?, ?]
[n, 2]

$$H(X) = XW$$

HYPOTHESIS USING MATRIX (8)

$$\begin{pmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \\ x_{41} & x_{42} & x_{43} \\ x_{51} & x_{52} & x_{53} \end{pmatrix} \cdot \begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \end{pmatrix} = \begin{pmatrix} x_{11}w_{11} + x_{12}w_{21} + x_{13}w_{31} & x_{11}w_{12} + x_{12}w_{22} + x_{13}w_{32} \\ x_{21}w_{11} + x_{22}w_{21} + x_{23}w_{31} & x_{21}w_{12} + x_{22}w_{22} + x_{23}w_{32} \\ x_{31}w_{11} + x_{32}w_{21} + x_{33}w_{31} & x_{31}w_{12} + x_{32}w_{22} + x_{33}w_{32} \\ x_{41}w_{11} + x_{42}w_{21} + x_{43}w_{31} & x_{41}w_{12} + x_{42}w_{22} + x_{43}w_{32} \\ x_{51}w_{11} + x_{52}w_{21} + x_{53}w_{31} & x_{51}w_{12} + x_{52}w_{22} + x_{53}w_{32} \end{pmatrix}$$

[n, 3] [3, 2]
[n, 2]

$$H(X) = XW$$

HYPOTHESIS USING MATRIX (9)

- Lecture (theory):

$$H(x) = Wx + b$$

- Implementation (TensorFlow)

$$H(X) = XW$$



DEEP LEARNING

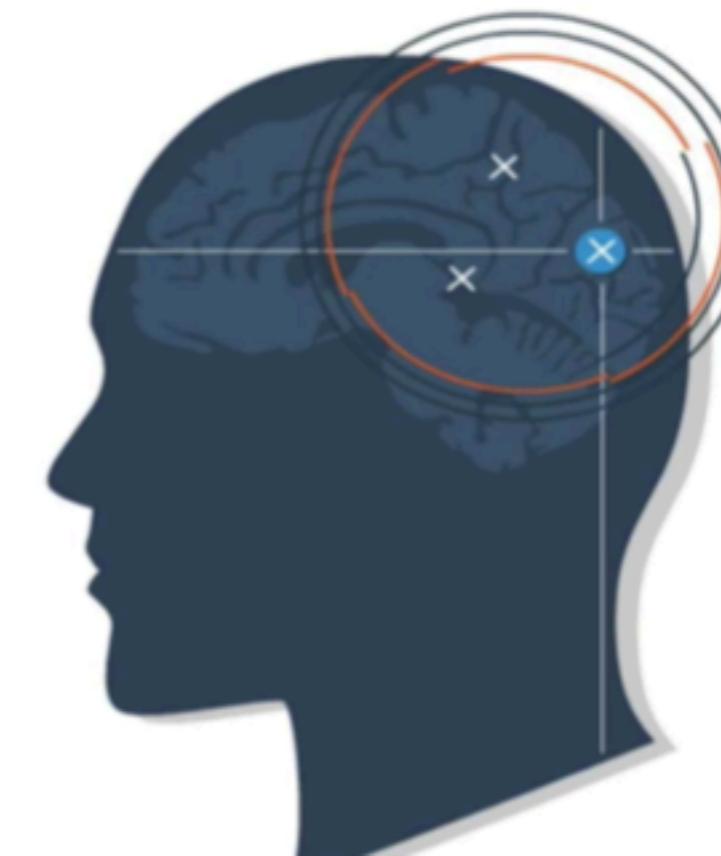
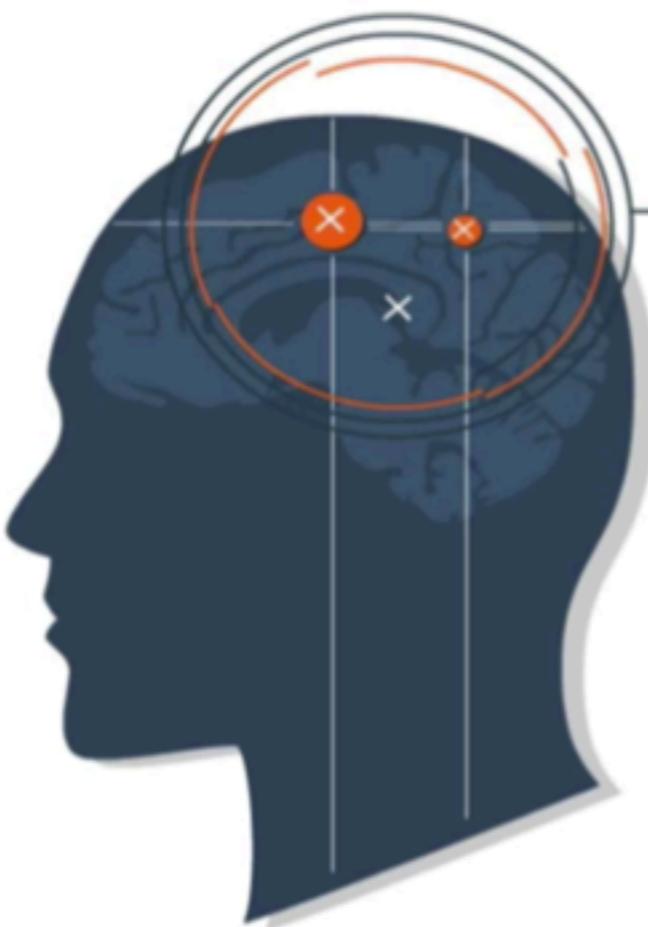
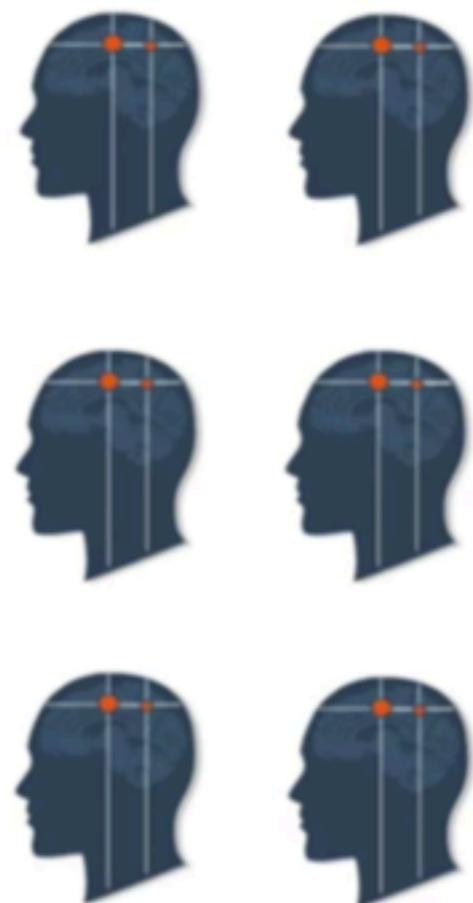
LOGISTIC CLASSIFICATION

CLASSIFICATION AND ENCODING

- Spam Detection: Spam or Ham
 - Facebook feed: show or hide
 - Credit Card Fraudulent Transaction detection: legitimate/fraud
-
- Spam Detection: Spam (1) or Ham (0)
 - Facebook feed: show(1) or hide(0)
 - Credit Card Fraudulent Transaction detection: legitimate(0) or fraud (1)

RADIOLOGY

Radiology

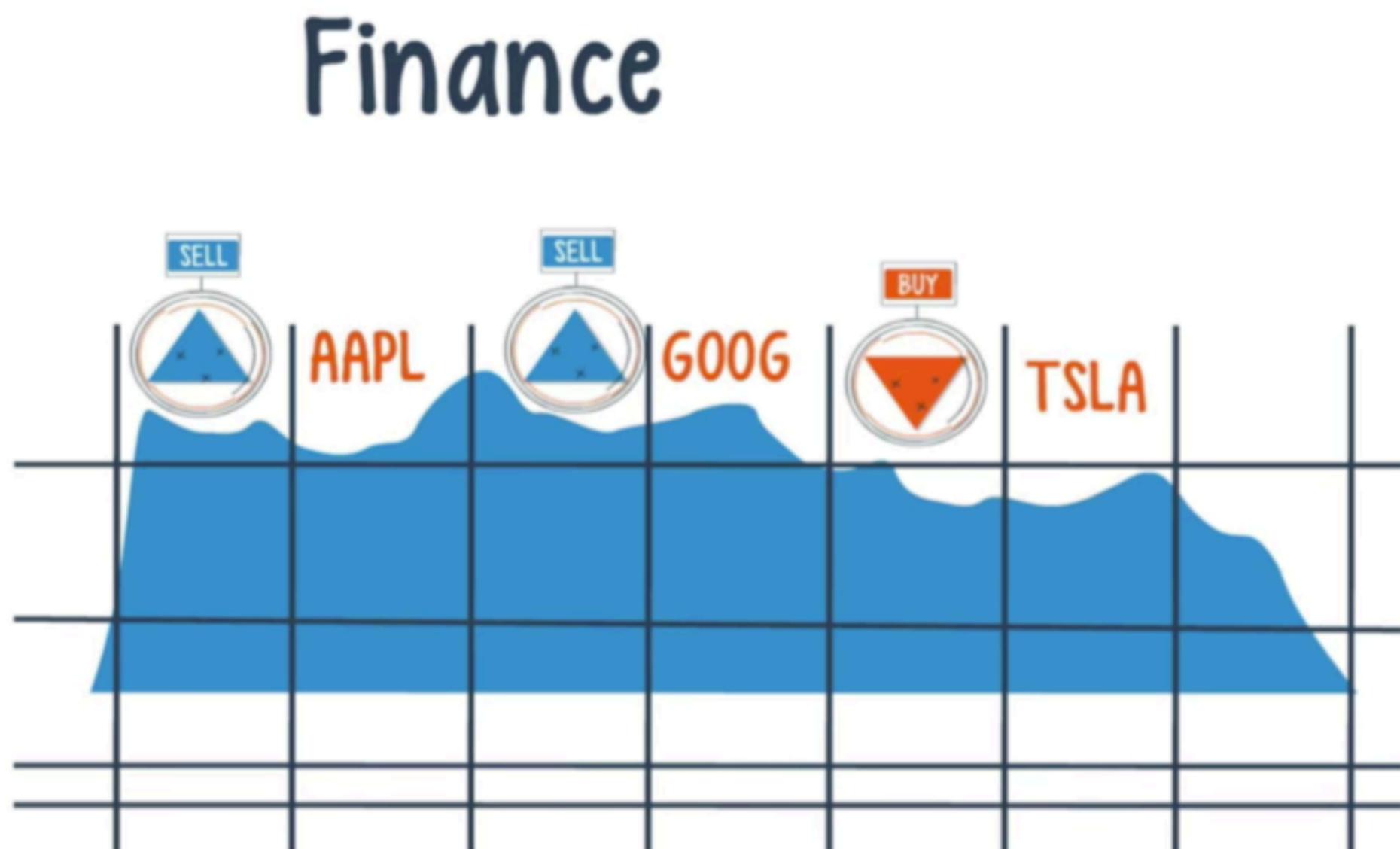


Malignant
tumor

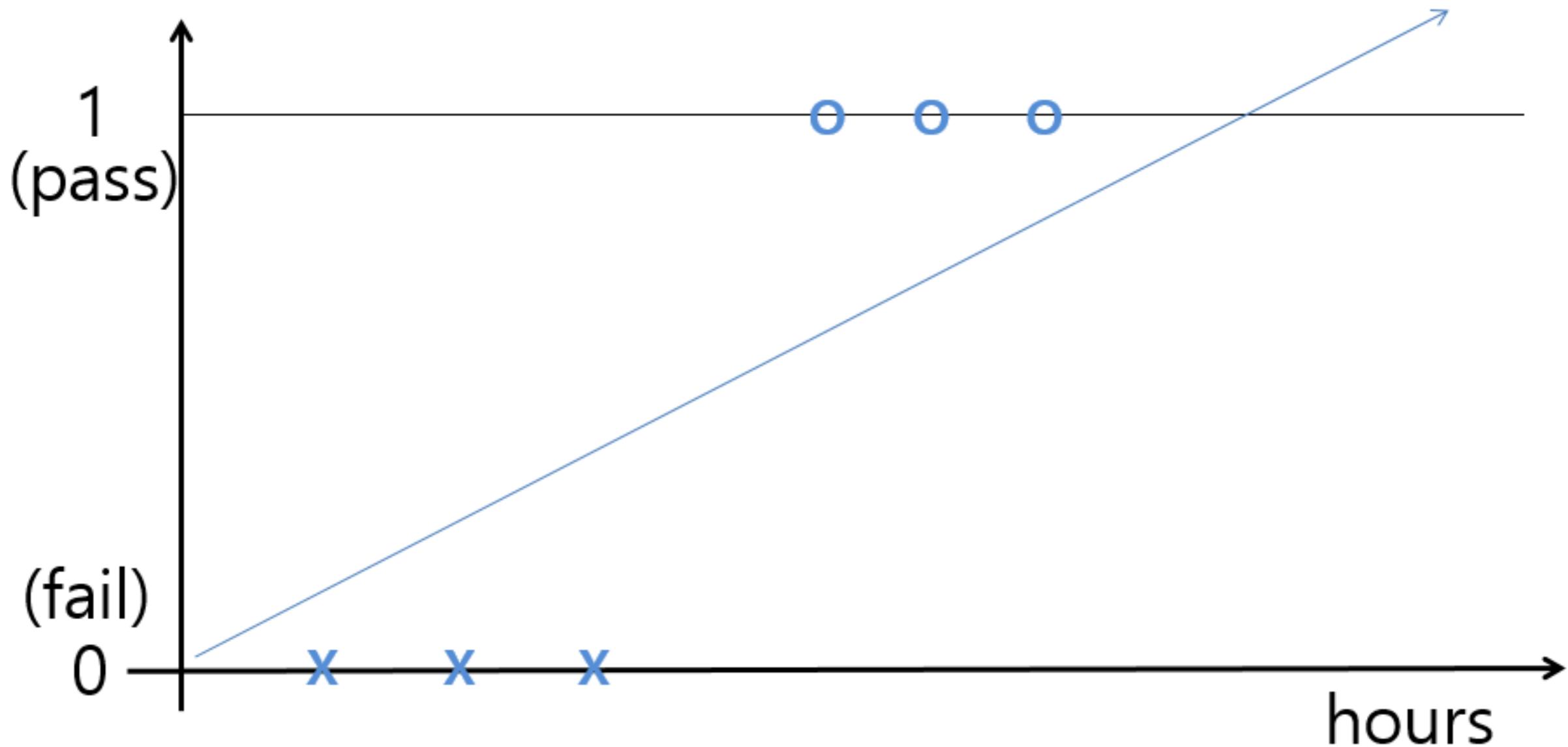
Benign
tumor

FINANCE

DWJI	17,499.10	▼
SP500	2,025.51	▼
NASDAQ	4,976.9	▲
AAPL	107.71	▲
GOOG	750.06	▲
TSLA	234.24	▼



LINEAR REGRESSION?

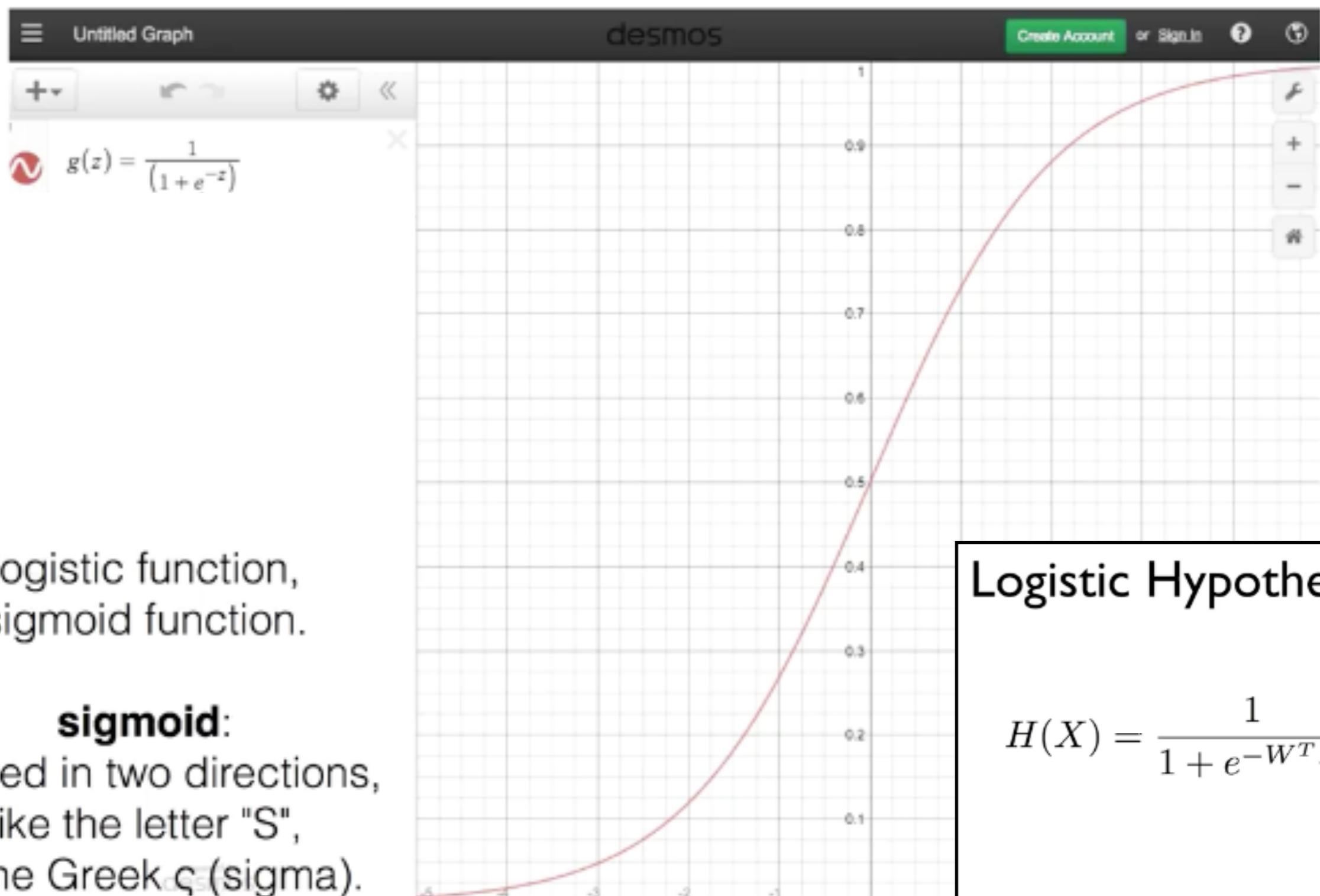


PROBLEMS

Linear regression

- We know Y is 0 or 1
$$H(x) = Wx + b$$
- Hypothesis can give values large than 1 or less than 0

SIGMOID



COST FUNCTION

$$cost(W, b) = \frac{1}{m} \sum_{i=1}^m (H(x^{(i)}) - y^{(i)})^2$$

$$H(x) = Wx + b$$

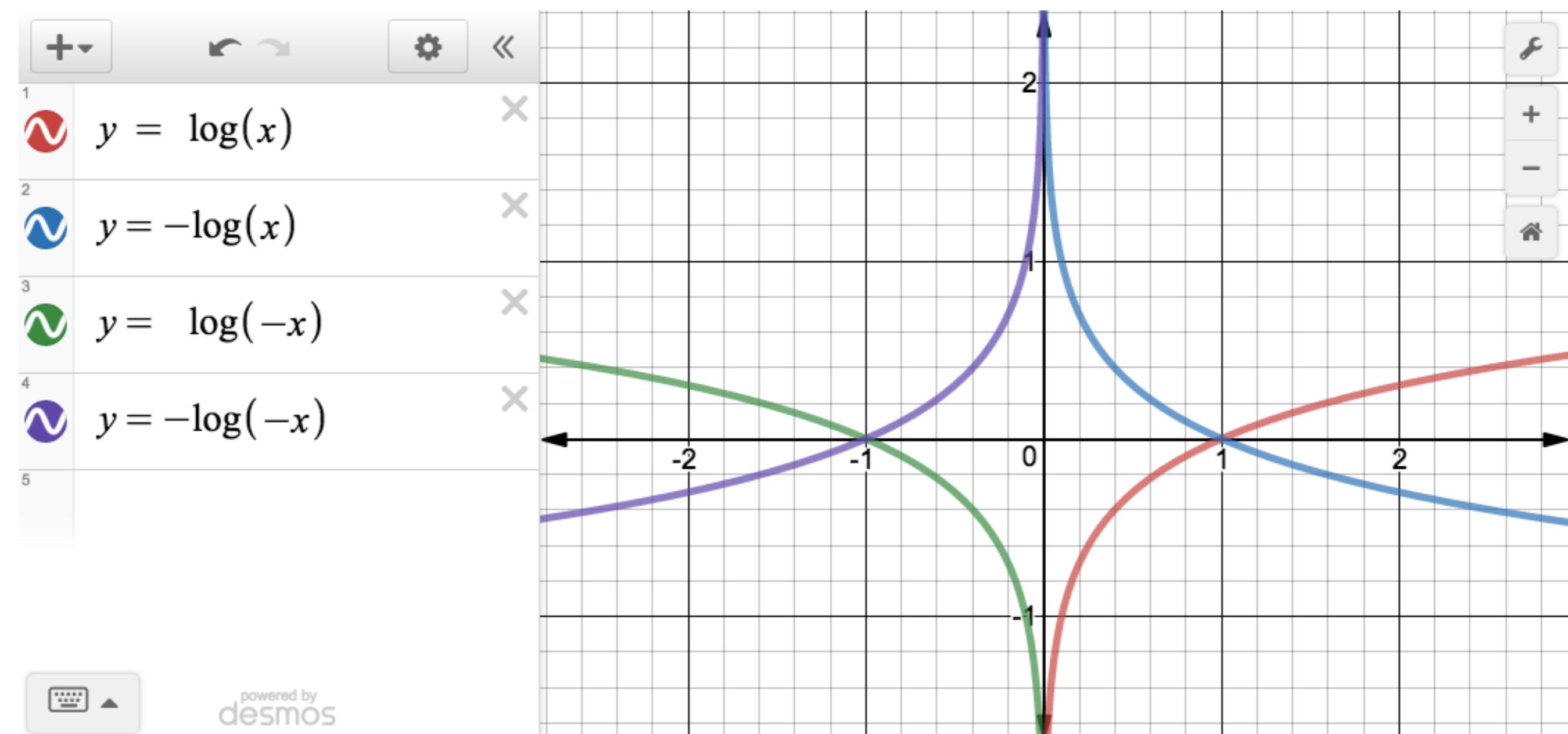
$$H(X) = \frac{1}{1 + e^{-W^T X}}$$

NEW COST FUNCTION FOR LOGISTIC

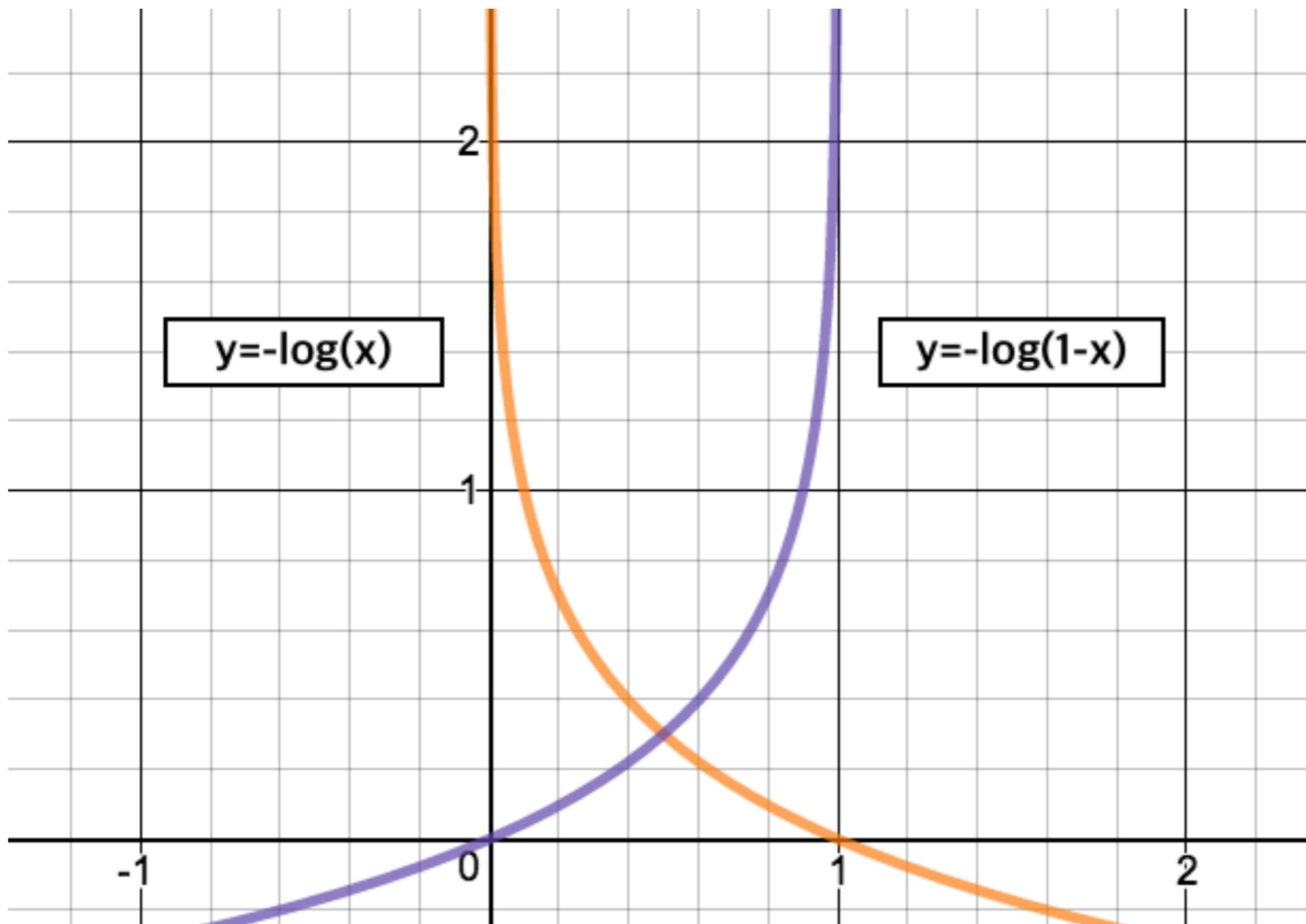
$$\text{cost}(W) = \frac{1}{m} \sum c(H(x), y)$$

$$c(H(x), y) = \begin{cases} -\log(H(x)) & : y = 1 \\ -\log(1 - H(x)) & : y = 0 \end{cases}$$

LOG FUNCTION



LOG FUNCTIONS WE NEED TO KNOW



UNDERSTANDING COST FUNCTION

$$C(H(x), y) = \begin{cases} -\log(H(x)) & : y = 1 \\ -\log(1-H(x)) & : y = 0 \end{cases}$$

$y = 1$

$H(x) = 1, \text{cost}(1) = 0$

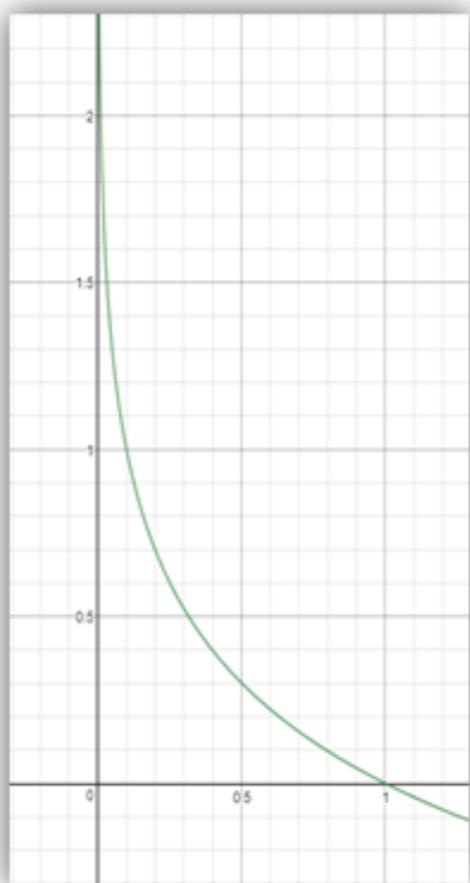
$H(x) = 0, \text{cost}(0) = \infty$

$y = 0$

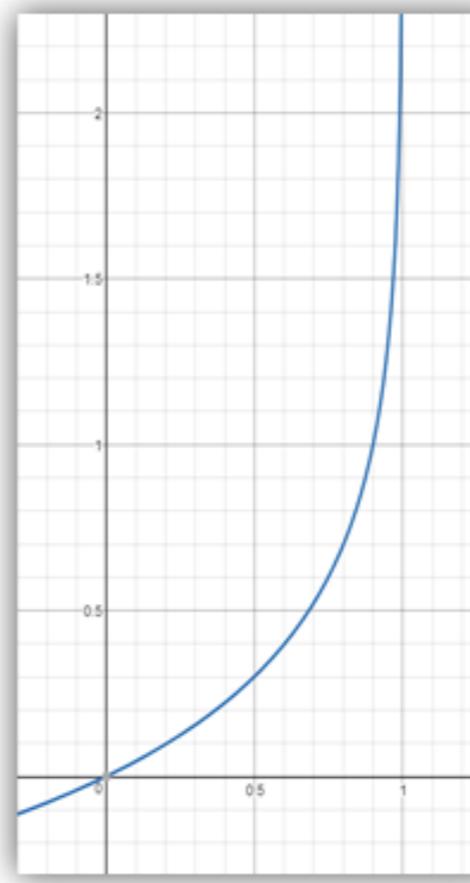
$H(x) = 0, \text{cost}(0) = 0$

$H(x) = 1, \text{cost}(1) = \infty$

$g(z) = -\log(z)$



$g(z) = -\log(1-z)$



COST FUNCTION

$$\text{cost}(W) = \frac{1}{m} \sum C(H(x), y)$$

$$C(H(x), y) = \begin{cases} -\log(H(x)) & : y = 1 \\ -\log(1 - H(x)) & : y = 0 \end{cases}$$

$$C(H(x), y) = -y \log(H(x)) - (1 - y) \log(1 - H(x))$$

GRADIENT DESCENT ALGORITHM

$$cost(W) = -\frac{1}{m} \sum y \log(H(x)) + (1 - y) \log(1 - H(x))$$

$$W := W - \alpha \frac{\partial}{\partial W} cost(W)$$

```
# cost function
cost = -tf.reduce_mean(Y*tf.log(hypothesis) + (1-Y)*tf.log(1-hypothesis))

# Minimize
a = tf.Variable(0.1) # Learning rate, alpha
optimizer = tf.train.GradientDescentOptimizer(a)
train = optimizer.minimize(cost)
```

CLASSIFYING DIABETES

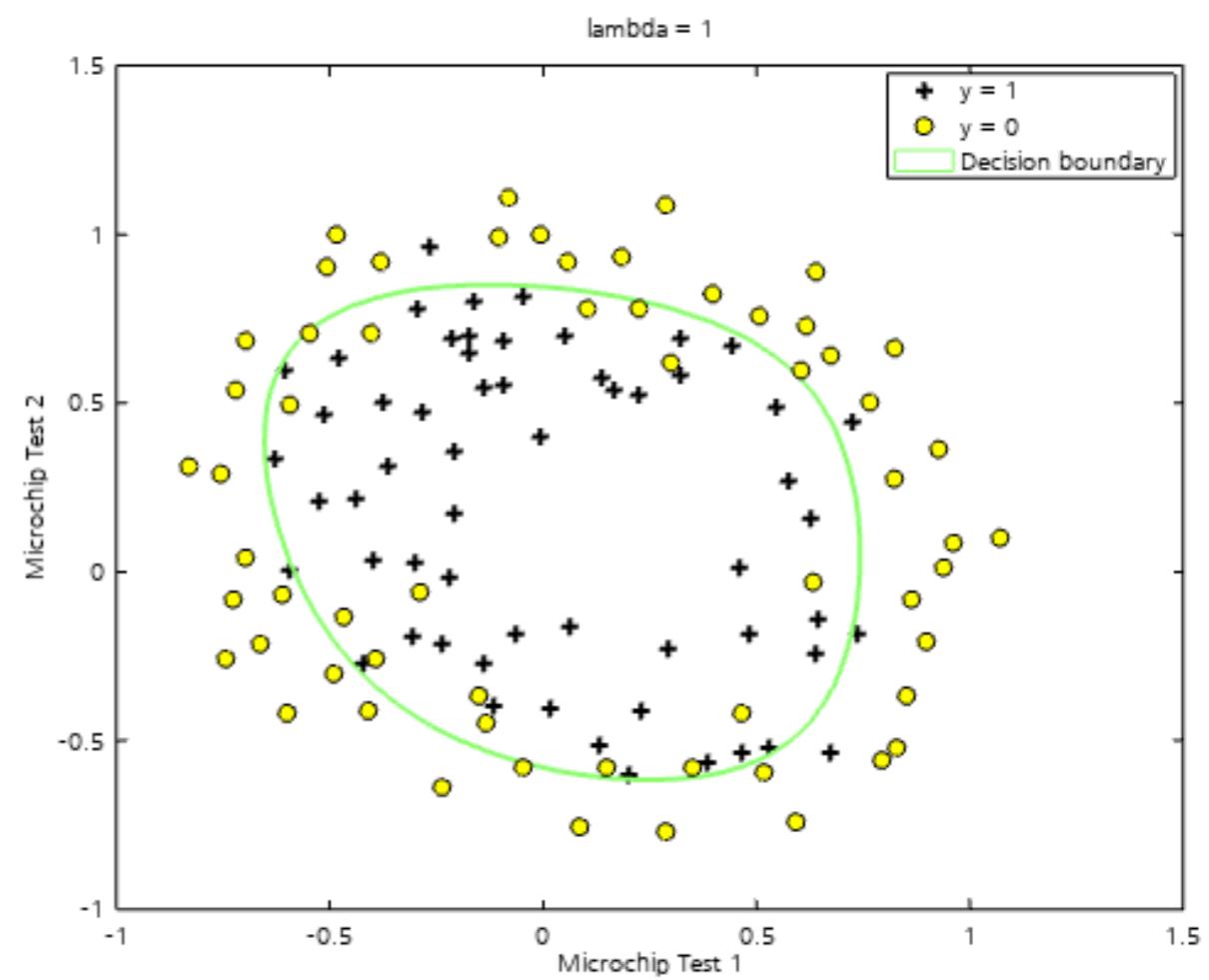
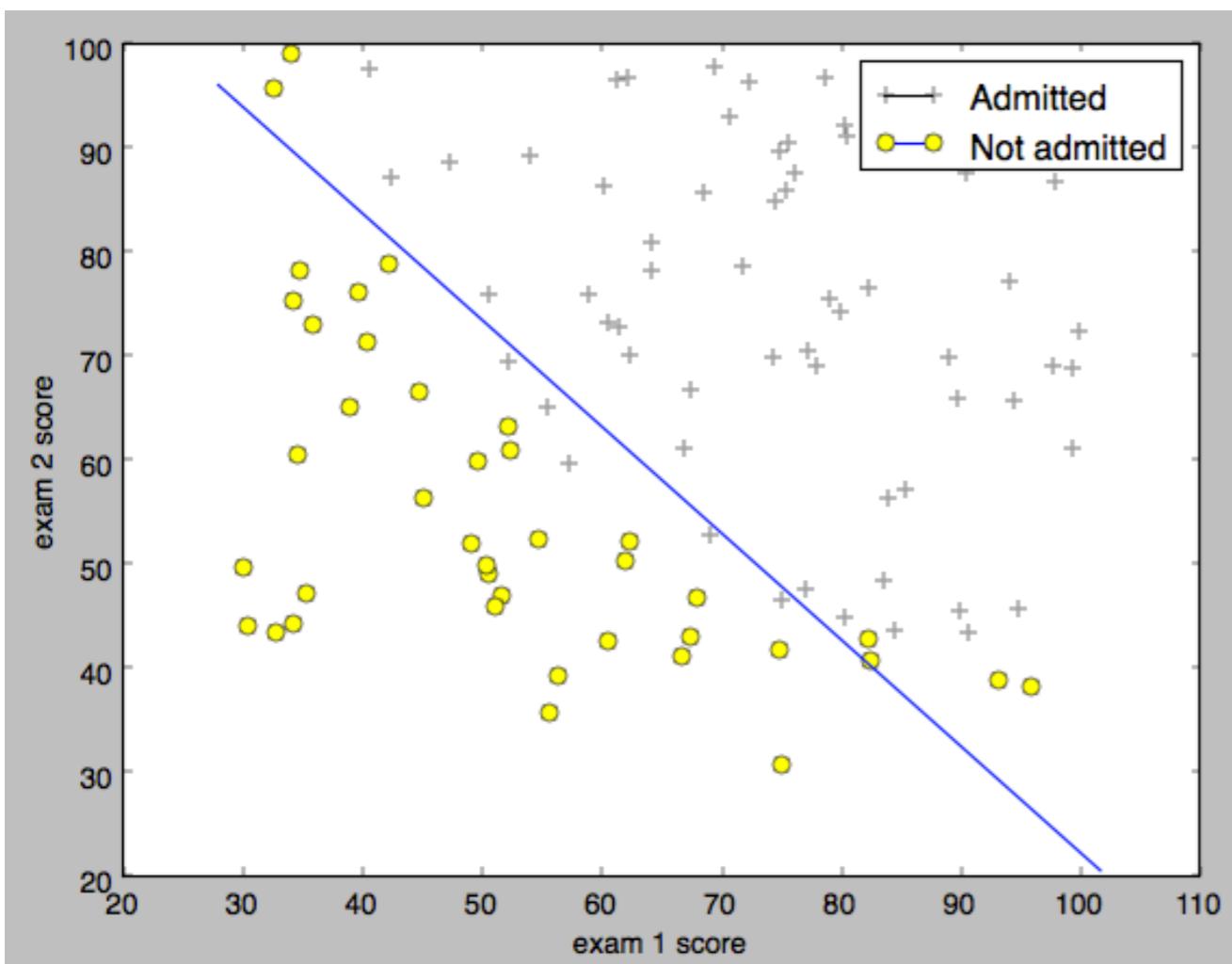
Classifying diabetes



-0.411765	0.165829	0.213115	0	0	-0.23696	-0.894962	-0.7	1
-0.647059	-0.21608	-0.180328	-0.353535	-0.791962	-0.0760059	-0.854825	-0.833333	0
0.176471	0.155779	0	0	0	0.052161	-0.952178	-0.733333	1
-0.764706	0.979899	0.147541	-0.0909091	0.283688	-0.0909091	-0.931682	0.0666667	0
-0.0588235	0.256281	0.57377	0	0	0	-0.868488	0.1	0
-0.529412	0.105528	0.508197	0	0	0.120715	-0.903501	-0.7	1
0.176471	0.688442	0.213115	0	0	0.132638	-0.608027	-0.566667	0
0.176471	0.396985	0.311475	0	0	-0.19225	0.163962	0.2	1

LOGISTIC CLASSIFICATION

DECISION BOUNDARY

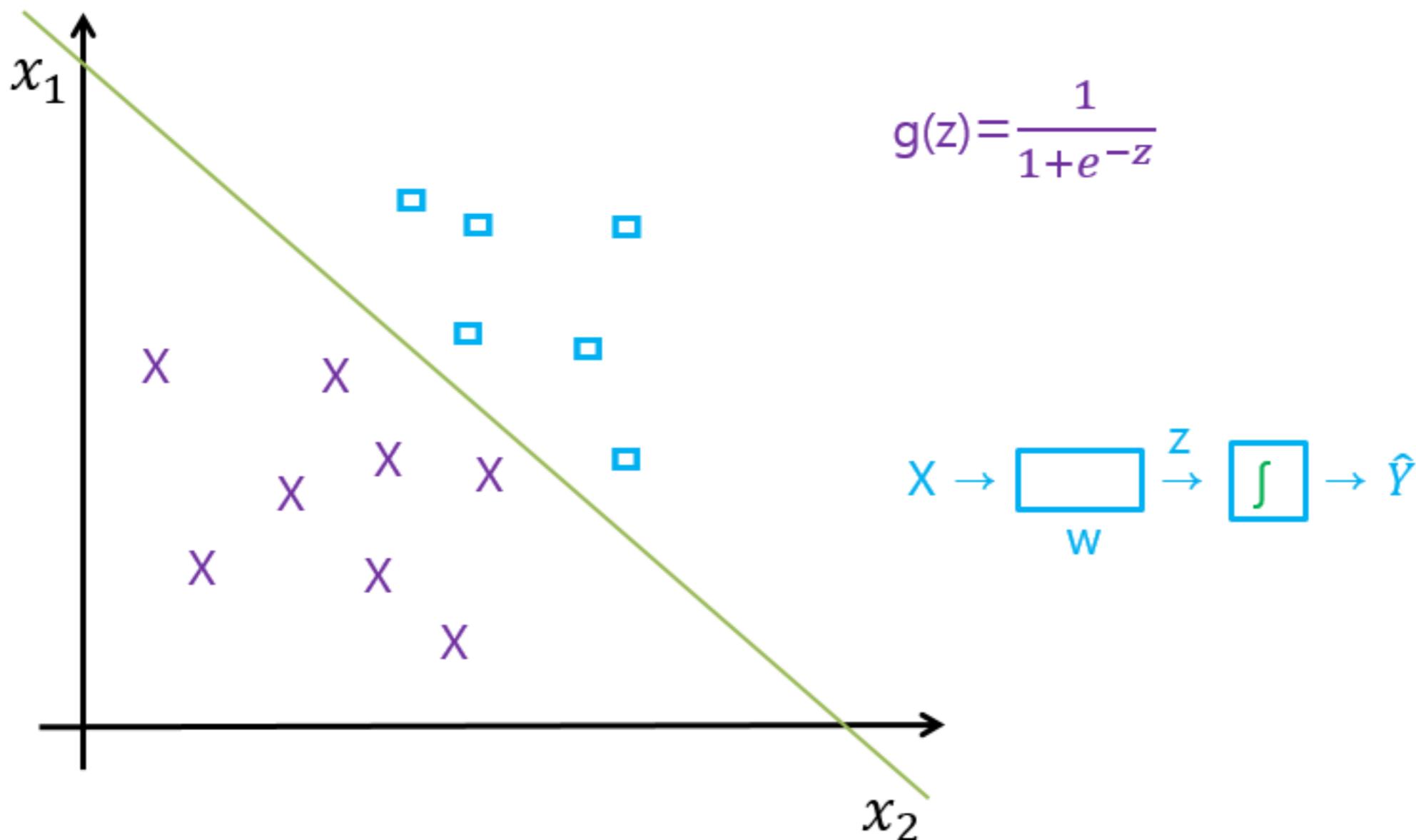




DEEP LEARNING

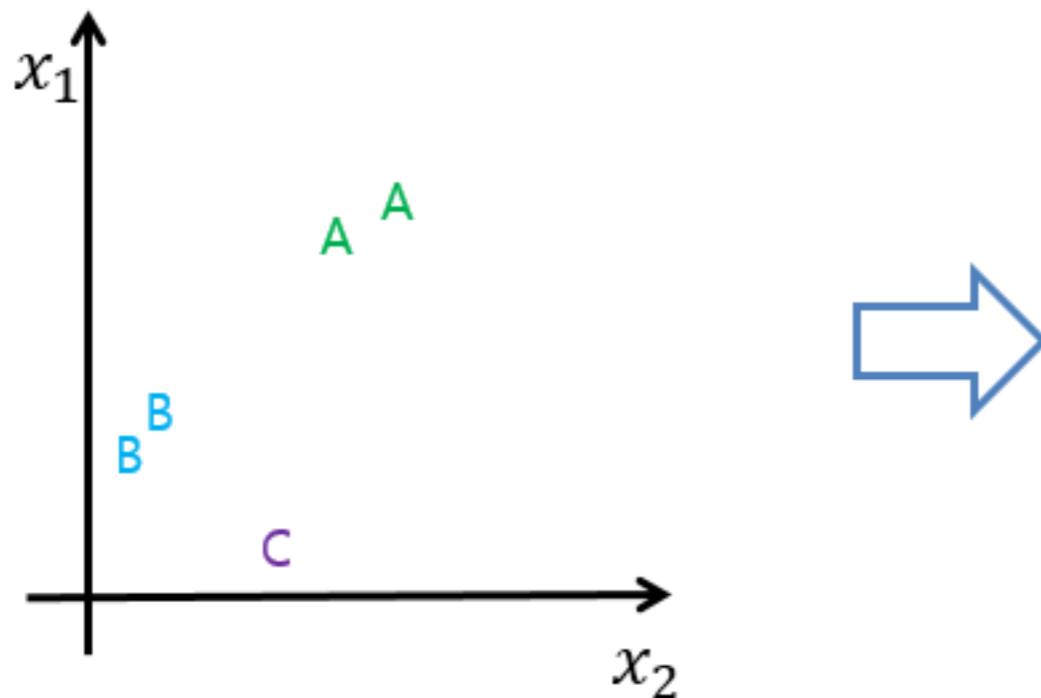
SOFTMAX, CROSS-ENTROPY, FC

LOGISTIC REGRESSION

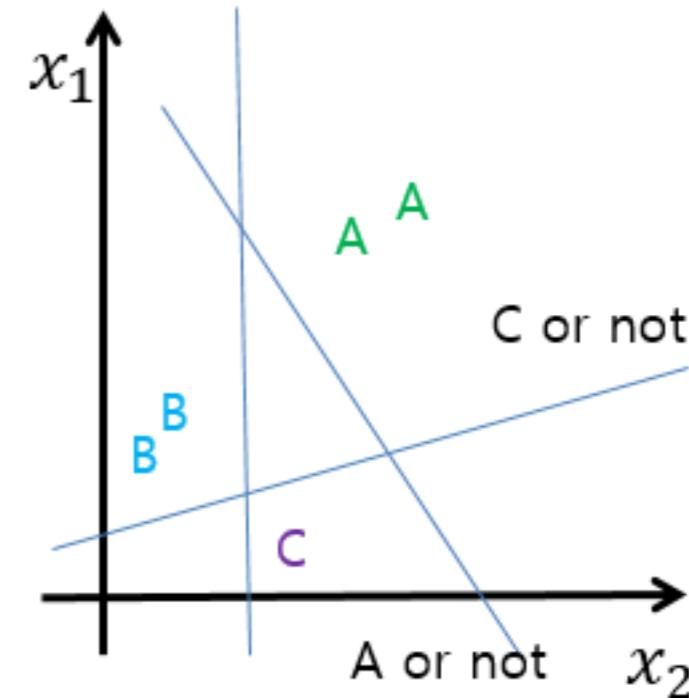


MULTINOMIAL CLASSIFICATION

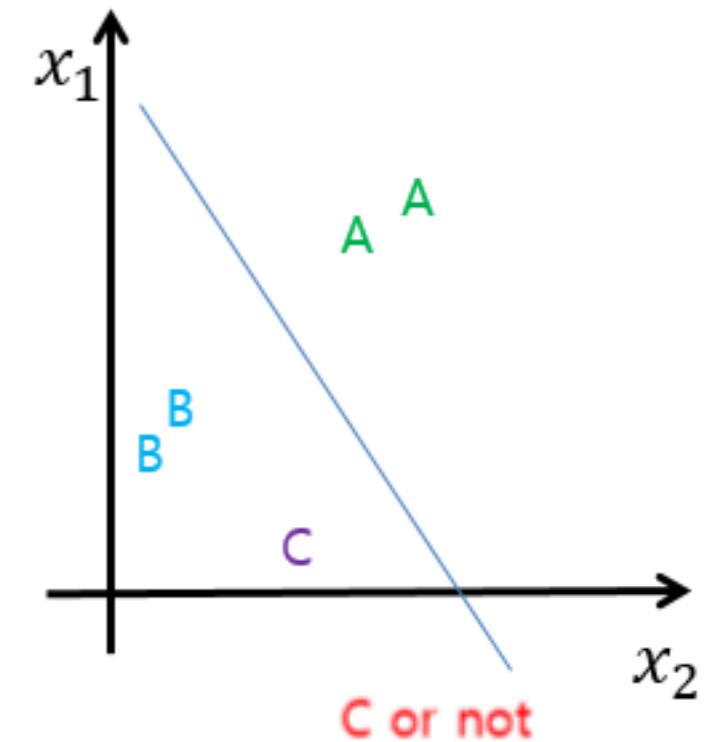
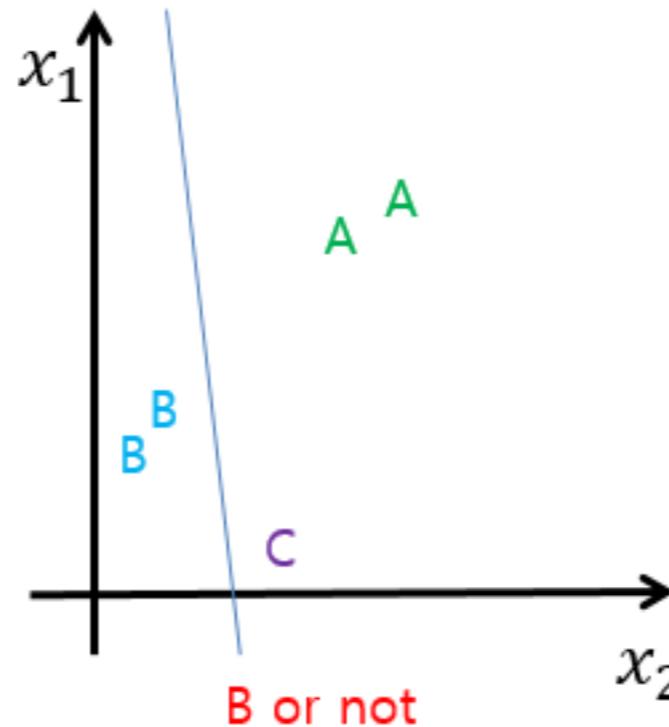
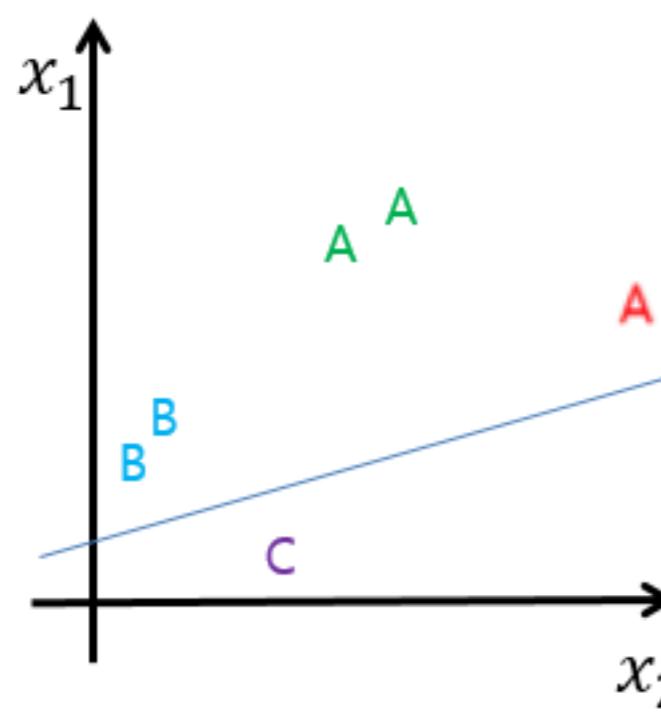
x_1 (hours)	x_2 (attendance)	y (grade)
10	5	A
9	5	A
3	2	B
2	4	B
11	1	C



B or not



MULTINOMIAL CLASSIFICATION



$$X \rightarrow \boxed{\quad} \rightarrow \hat{Y}$$

A

$$X \rightarrow \boxed{\quad} \rightarrow \hat{Y}$$

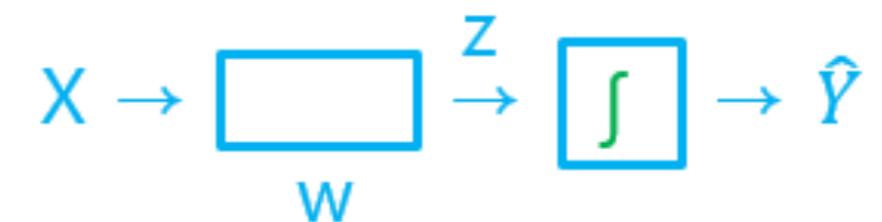
B

$$X \rightarrow \boxed{\quad} \rightarrow \hat{Y}$$

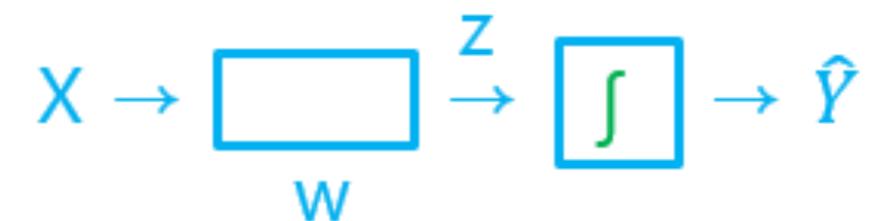
C

MULTINOMIAL CLASSIFICATION

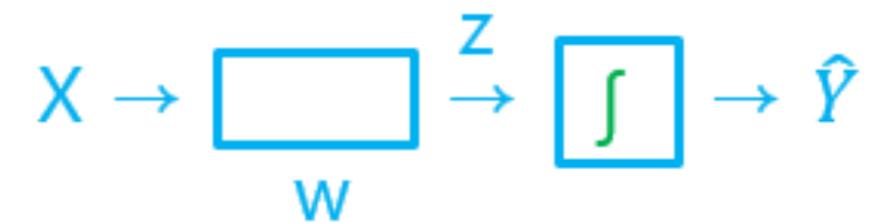
$$[w_1 \ w_2 \ w_3] \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = [w_1 x_1 + w_2 x_2 + w_3 x_3]$$



$$[w_1 \ w_2 \ w_3] \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = [w_1 x_1 + w_2 x_2 + w_3 x_3]$$

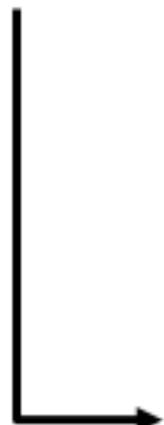


$$[w_1 \ w_2 \ w_3] \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = [w_1 x_1 + w_2 x_2 + w_3 x_3]$$

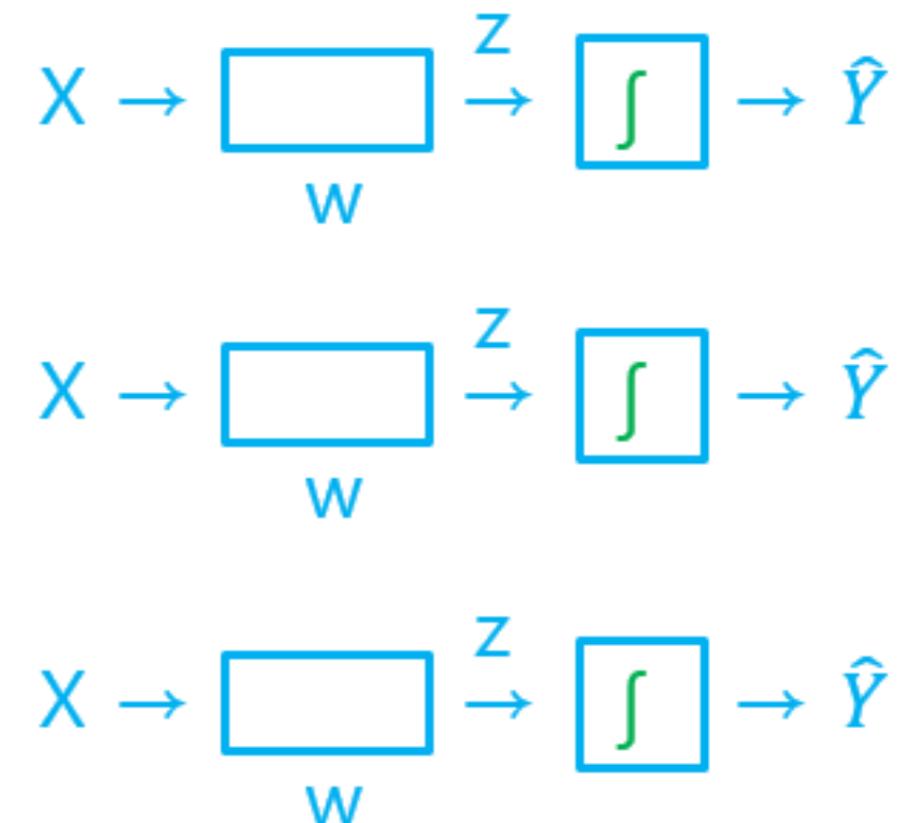


MULTINOMIAL CLASSIFICATION

$$[w_1 \ w_2 \ w_3] \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = [w_1 x_1 + w_2 x_2 + w_3 x_3]$$

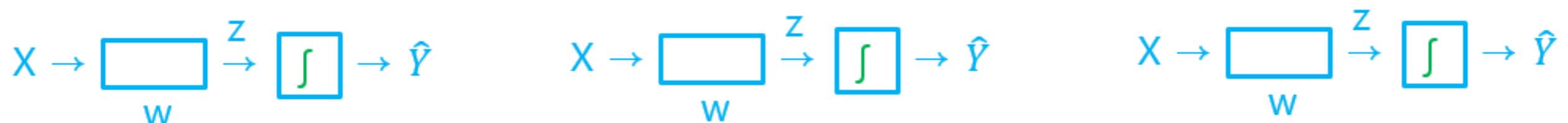


$$\begin{bmatrix} w_{A1} & w_{A2} & w_{A3} \\ w_{B1} & w_{B2} & w_{B3} \\ w_{C1} & w_{C2} & w_{C3} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = ?$$



MULTINOMIAL CLASSIFICATION

$$\begin{bmatrix} w_{A1} & w_{A2} & w_{A3} \\ w_{B1} & w_{B2} & w_{B3} \\ w_{C1} & w_{C2} & w_{C3} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} w_{A1}x_1 + w_{A2}x_2 + w_{A3}x_3 \\ w_{B1}x_1 + w_{B2}x_2 + w_{B3}x_3 \\ w_{C1}x_1 + w_{C2}x_2 + w_{C3}x_3 \end{bmatrix} = \begin{bmatrix} \hat{Y}_A \\ \hat{Y}_B \\ \hat{Y}_C \end{bmatrix}$$



WHERE IS SIGMOID?

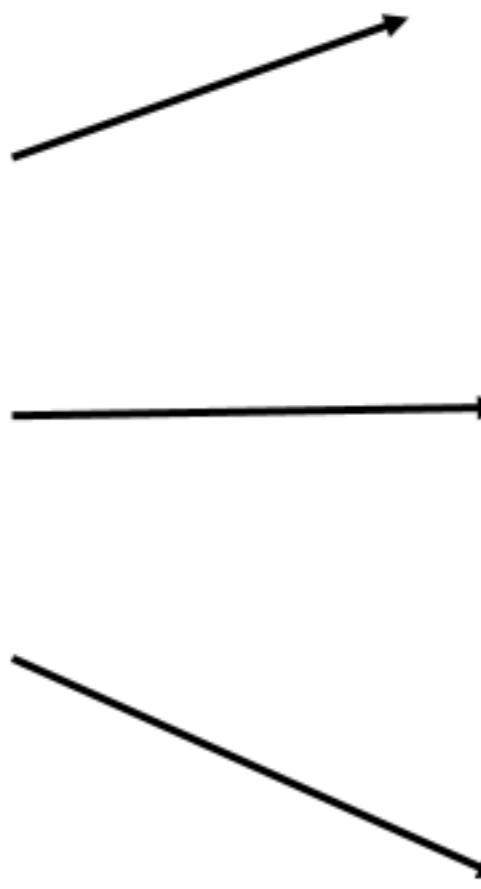
$$\begin{bmatrix} w_{A1} & w_{A2} & w_{A3} \\ w_{B1} & w_{B2} & w_{B3} \\ w_{C1} & w_{C2} & w_{C3} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} w_{A1}x_1 + w_{A2}x_2 + w_{A3}x_3 \\ w_{B1}x_1 + w_{B2}x_2 + w_{B3}x_3 \\ w_{C1}x_1 + w_{C2}x_2 + w_{C3}x_3 \end{bmatrix} = \begin{bmatrix} \hat{Y}_A \\ \hat{Y}_B \\ \hat{Y}_C \end{bmatrix} \begin{bmatrix} 2.0 \\ 1.0 \\ 0.1 \end{bmatrix}$$



SIGMOID?

LOGISTIC CLASSIFIER

$$WX = Y \begin{bmatrix} 2.0 \\ 1.0 \\ 0.1 \end{bmatrix}$$



$p = 0.7$



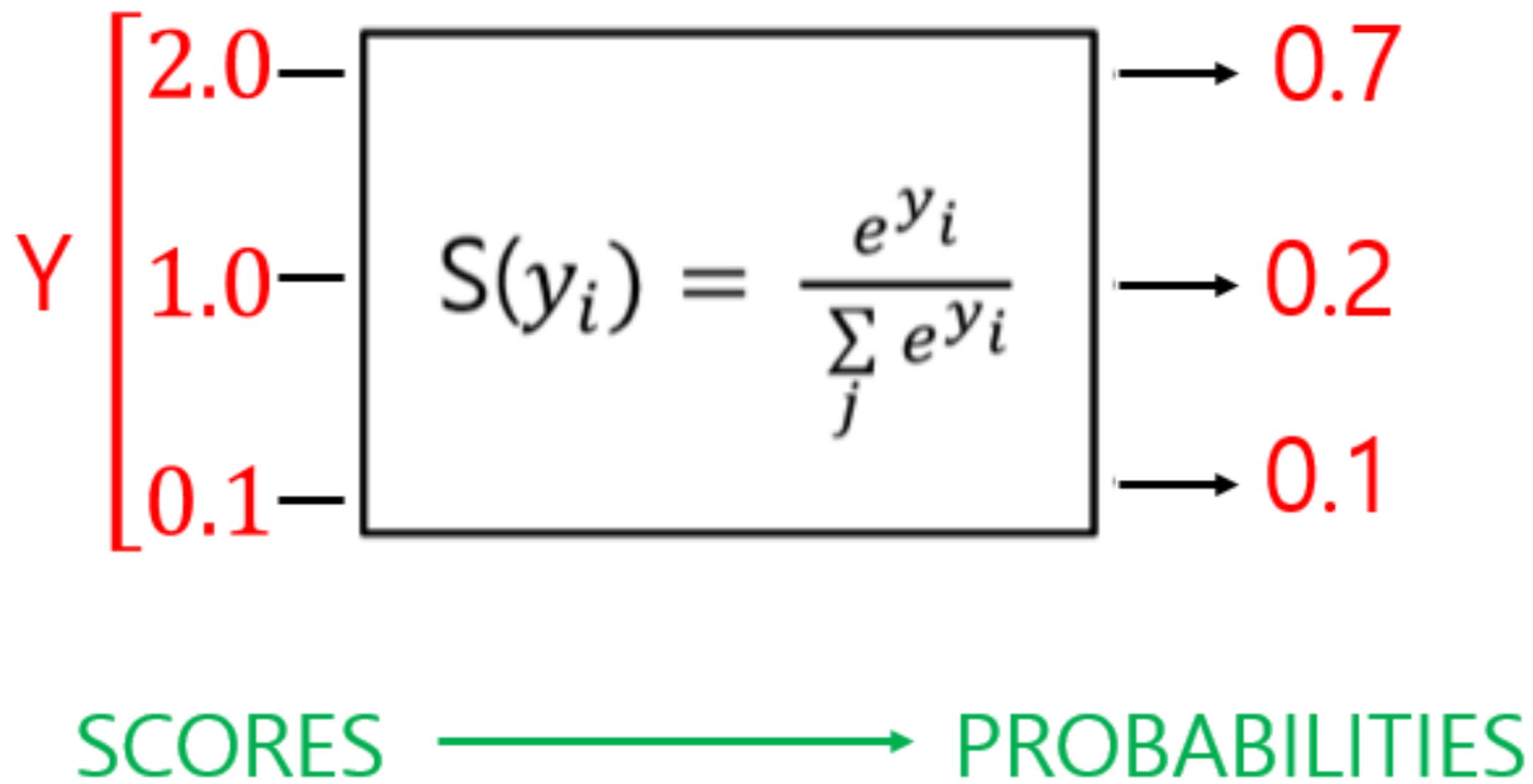
$p = 0.2$



$p = 0.1$

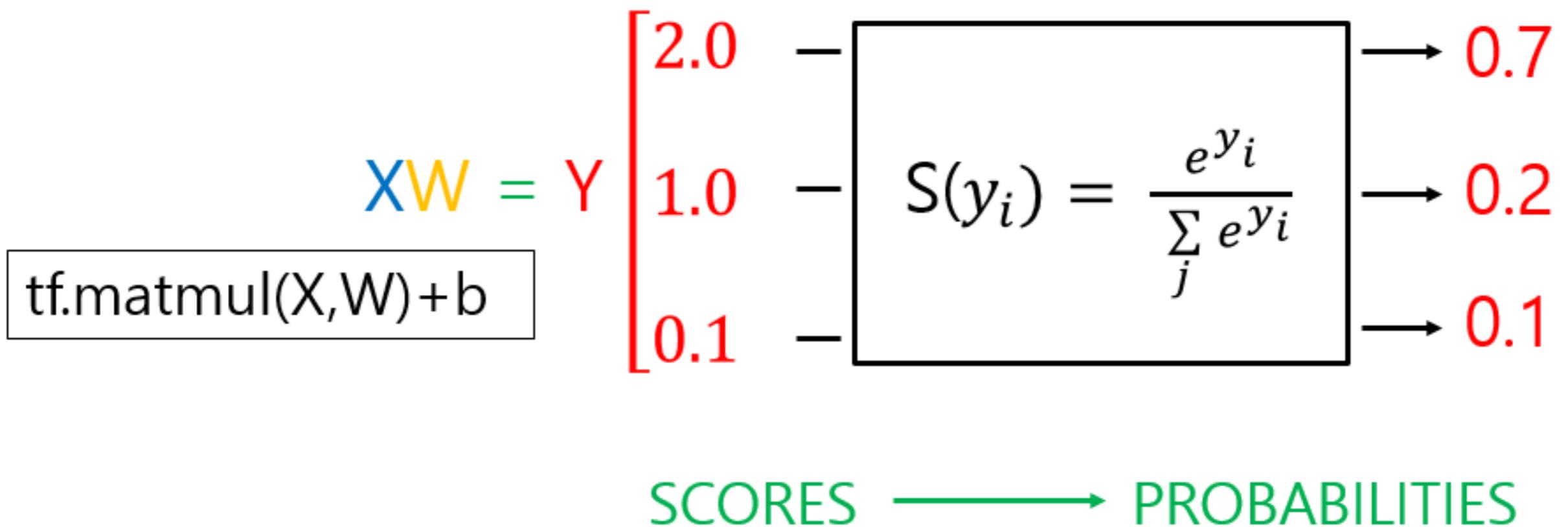


SOFTMAX



SOFTMAX

```
hypothesis = tf.nn.softmax(tf.matmul(X,W)+b)
```



SOFTMAX_CROSS_ENTROPY_WITH_LOGITS

```
logits = tf.matmul(X, W) + b  
hypothesis = tf.nn.softmax(logits)
```

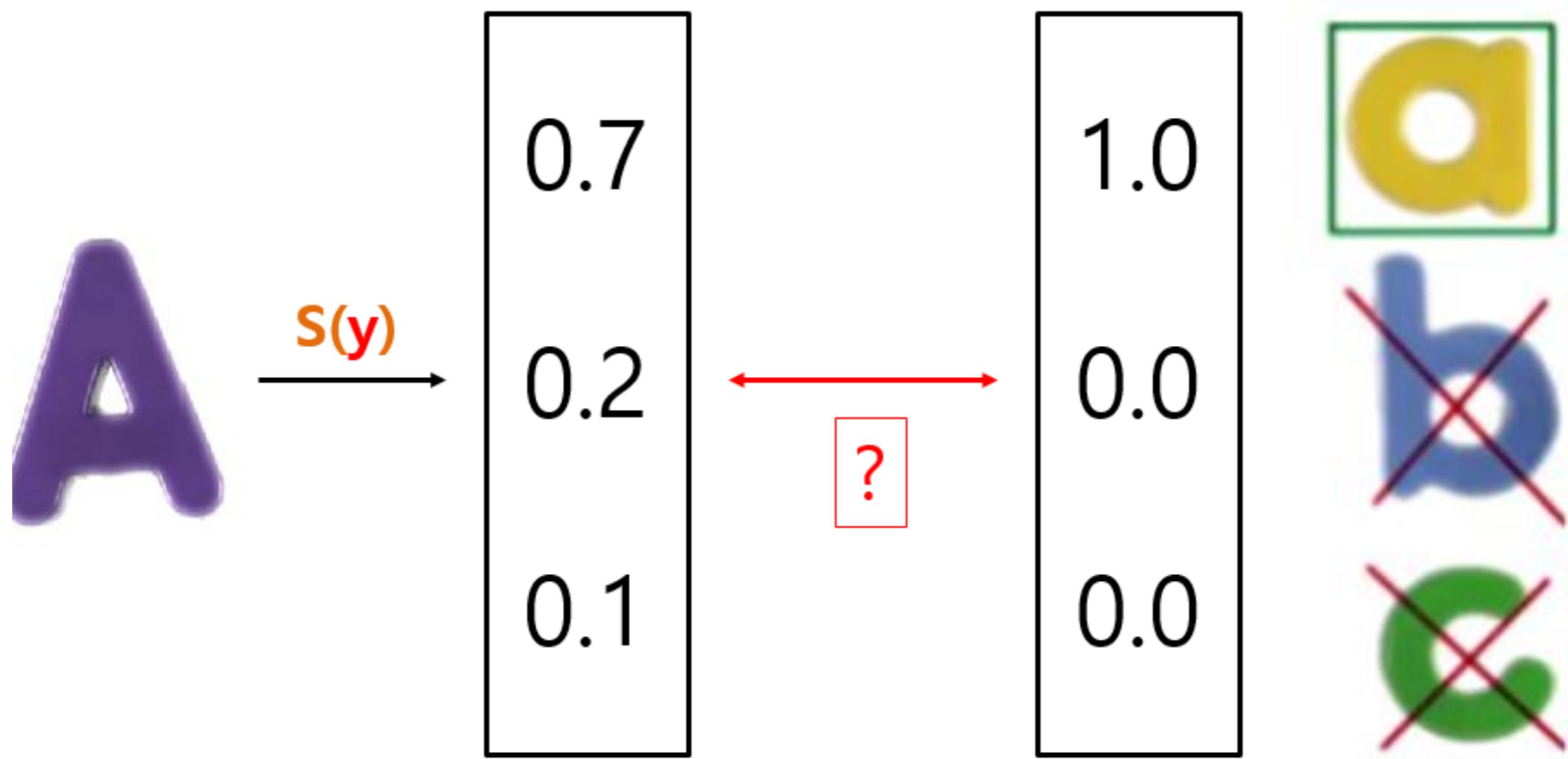
1

```
# Cross entropy cost/Loss  
cost = tf.reduce_mean(-tf.reduce_sum(Y * tf.log(hypothesis), axis=1))
```

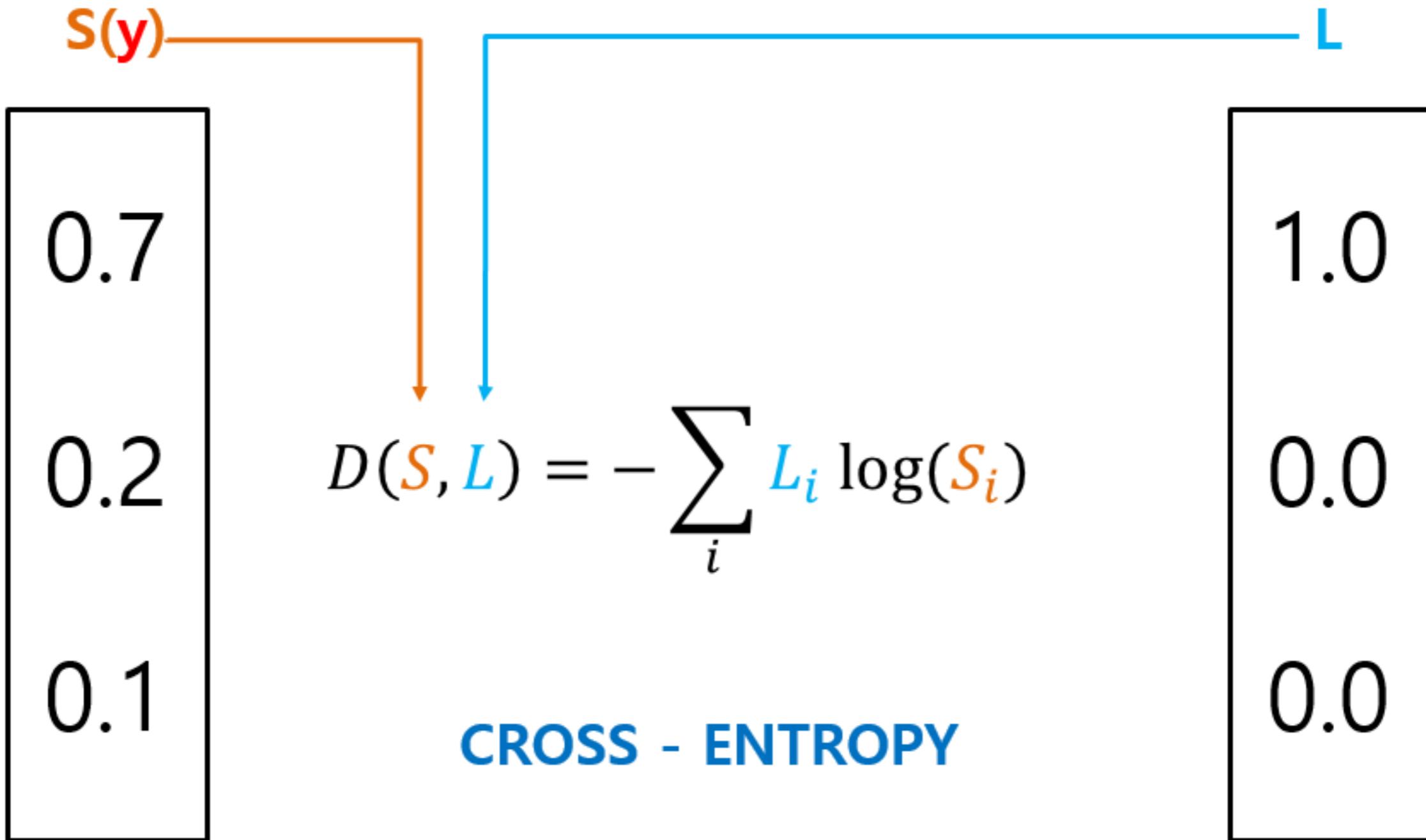
2

```
# Cross entropy cost/Loss  
cost_i = tf.nn.softmax_cross_entropy_with_logits(logits=logits,  
                                                labels=Y_one_hot)  
cost = tf.reduce_mean(cost_i)
```

ONE-HOT ENCODING

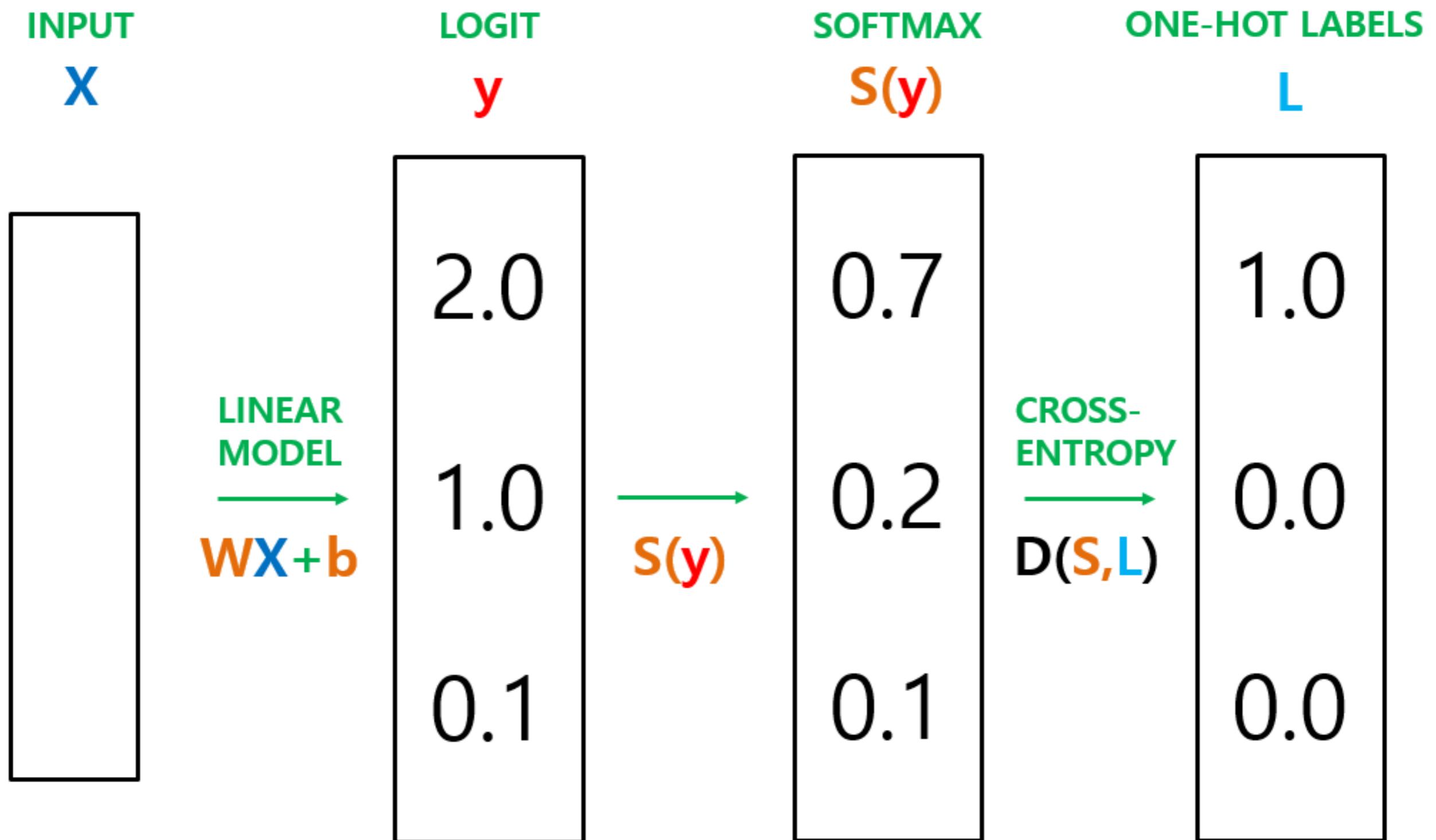


COST FUNCTION



SOFTMAX CROSS-ENTROPY

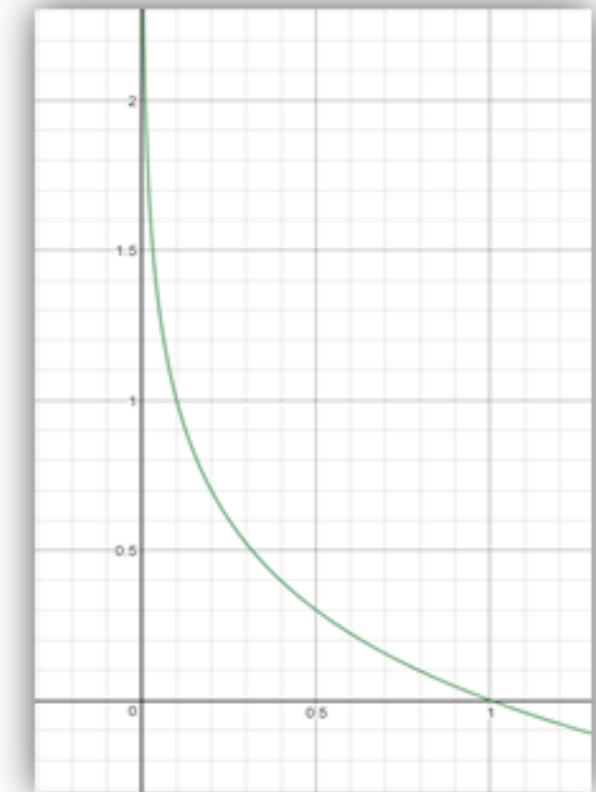
COST FUNCTION



CROSS-ENTROPY COST FUNCTION

$$-\sum_i L_i \log(s_i) \rightarrow -\sum_i L_i \log(\hat{y}_i) \rightarrow \sum_i L_i * -\log(\hat{y}_i)$$

$$L = \begin{matrix} A \\ B \end{matrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = B$$



$$\hat{Y} = \begin{matrix} A \\ B \end{matrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = B(0), \quad \begin{bmatrix} 0 \\ 1 \end{bmatrix} * -\log \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} * \begin{bmatrix} \infty \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} = 0$$

$$\hat{Y} = \begin{matrix} A \\ B \end{matrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = A(X), \quad \begin{bmatrix} 0 \\ 1 \end{bmatrix} * -\log \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} * \begin{bmatrix} 0 \\ \infty \end{bmatrix} = \begin{bmatrix} 0 \\ \infty \end{bmatrix} = \infty$$

LOGISTIC COST VS. CROSS ENTROPY

$$C(H(x), y) = -y \log(H(x)) - (1 - y) \log(1 - H(x))$$

$$D(\textcolor{brown}{S}, \textcolor{blue}{L}) = - \sum_i \textcolor{blue}{L}_i \log(\textcolor{brown}{S}_i)$$

COST FUNCTION : CROSS ENTROPY

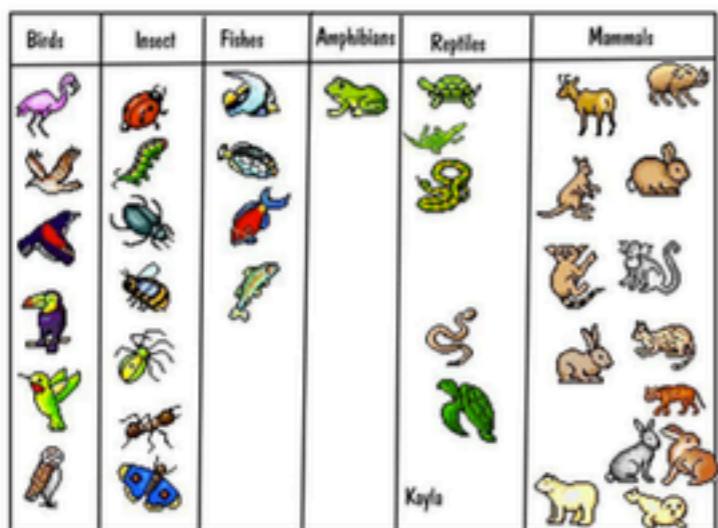
$$L = \frac{1}{N} \sum_i D(S(WX_i + b), L_i)$$

The diagram illustrates the calculation of the total loss L . It shows a large arrow pointing upwards from the text "TRAINING SET" to the summation symbol in the equation. From the tip of this large arrow, several smaller arrows branch off, each pointing to one of the terms $D(S(WX_i + b), L_i)$ in the summation, representing the individual losses for each training example i .

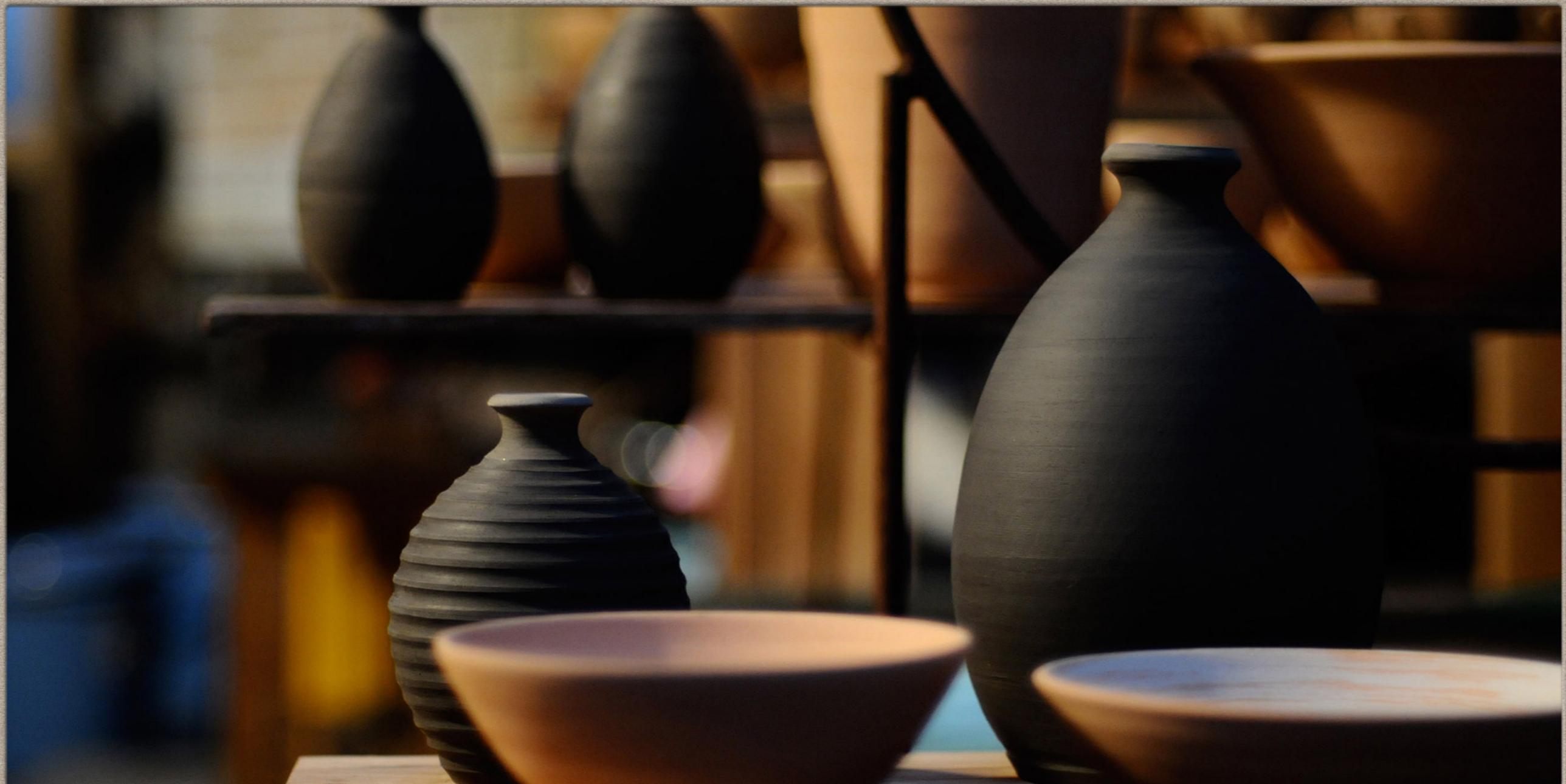
```
# Cross entropy cost/Loss  
cost = tf.reduce_mean(-tf.reduce_sum(Y * tf.log(hypothesis), axis=1))  
  
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.1).minimize(cost)
```

ANIMAL CLASSIFICATION

with *softmax_cross_entropy_with_logits*



1	0	0	1	0	0	0	1	1	0	0	4	3	0	1	0
0	0	1	0	0	1	1	1	1	0	0	1	0	1	0	3
1	0	0	0	1	0	0	1	1	1	0	0	4	0	0	1
1	0	0	0	1	0	0	1	1	0	0	4	1	0	1	0
1	0	0	0	1	0	0	1	1	0	0	4	1	0	1	0
1	0	0	0	1	0	0	1	1	0	0	4	1	0	1	0
1	0	0	0	1	0	0	1	1	0	0	4	1	0	1	0
0	0	1	0	0	1	0	1	1	0	0	1	0	1	0	3
0	0	1	0	0	1	0	1	1	0	0	1	0	1	0	3
0	0	1	0	0	1	0	1	1	0	0	1	0	1	0	3
0	0	1	0	0	1	0	1	1	0	0	1	0	1	0	3
0	0	1	0	0	1	0	1	1	0	0	1	0	1	0	3
0	0	1	0	0	1	0	1	1	0	0	1	0	1	0	3
0	0	1	0	0	1	0	1	1	0	0	1	0	1	0	3
0	0	1	0	0	1	0	1	1	0	0	1	0	1	0	3
0	0	1	0	0	1	0	1	1	0	0	1	0	1	0	3
0	0	1	0	0	1	0	1	1	0	0	1	0	1	0	3
1	0	0	1	0	0	0	1	1	1	0	0	4	1	0	1



NLTK

NATURAL LANGUAGE TOOLKIT

NLTK

- Natural Language Toolkit
- 교육용으로 개발된 자연어 처리 및 문서 분석용 파이썬 패키지
- 다양한 기능 및 예제 포함
- 연구를 비롯해서 현업에서도 많이 사용

주요 기능

- 말뭉치 corpus, corpora(corpus의 복수형)
- 토큰 생성 tokenizing
- 형태소 분석 morphological analysis
- 품사 태깅 POS tagging

CORPORA

- WordNet
`nltk.corpus.wordnet` - English Lexical Database
- Guttenberg
`nltk.corpus.gutenberg` - Books from the Gutenberg Project
- WebText
`nltk.corpus.webtext` - User generated content on the web
- Brown
`nltk.corpus.brown` - Text categorized by genre
- Reuters
`nltk.corpus.reuters` - News Corpus
- Words
`nltk.corpus.words` - English Vocabulary
- SentiWordNet
`nltk.corpus.sentiwordnet` - Sentiment polarities mapped over WordNet structure

GUTTENBERG CORPUS

- 'austen-emma.txt', 'austen-persuasion.txt', 'austen-sense.txt',
'bible-kjv.txt', 'blake-poems.txt', 'bryant-stories.txt',
'burgess-busterbrown.txt', 'carroll-alice.txt', 'chesterton-ball.txt',
'chesterton-brown.txt', 'chesterton-thursday.txt',
'edgeworth-parents.txt', 'melville-moby_dick.txt', 'milton-paradise.txt',
'shakespeare-caesar.txt', 'shakespeare-hamlet.txt',
'shakespeare-macbeth.txt', 'whitman-leaves.txt'

AUSTEN-EMMA.TXT

- 제인 오스틴(Jane Austen)
- 1816년에 집필한 소설
- 유명한 영국의 소설가
- 1775년 12월 16일 - 1817년 7월 18일
- 섬세한 시선과 재치있는 문체로 18세기 영국 중·상류층 여성들의 삶을 다루는 것이 특징
- 생전에는 그리 유명하지 않았으나, 20세기에 들어와서는 작품 중 《오만과 편견》, 《이성과 감성》등은 여러 번 영화화되는 등 인기를 끌고 있다.



Universal History Archive via Getty Images

토큰 생성

- 자연어 분석을 위해서는 긴 문자열을 작은 단위로 분할
- 분할된 작은 문자열을 토큰(token)이라고 부른다.
- 토큰으로 나누는 작업은 토큰 생성(tokenizing)이라고 부른다.

TOKENIZER

- `simple`
`split()` 함수 기반으로 동작하는 형태소 분리 모듈
- `word_tokenize()`
단어 단위로 형태소를 분리하는 함수
- `sent_tokenize()`
문장 단위로 형태소를 분리하는 함수
- `RegexpTokenizer()`
정규표현식을 사용해서 형태소를 분리하는 클래스

형태소 분석

- morphological analysis
- 형태소(morpheme)
언어학에서 일정한 의미가 있는 가장 작은 말의 단위
- 보통 자연어 처리에서는 토큰으로 형태소를 이용한다.
- 단어로부터 어근, 접두사, 접미사, 품사 등 다양한 언어적 속성을 파악하고 이를 이용하여 형태소를 찾아내거나 처리하는 작업
- 형태소 분석의 예
 - 어간 추출(stemming)
 - 원형 복원(lemmatizing)
 - 품사 부착(Part-Of-Speech tagging)

어간 추출

- 어간 추출(stemming)은 여러가지 이유로 변화된 단어의 접미사나 어미를 제거하여 같은 의미를 가지는 형태소의 실제 형태를 동일하게 만드는 방법이다. (Stemmers remove morphological affixes from words, leaving only the word stem.)
- NLTK는 PorterStemmer LancasterStemmer 등을 제공한다.
- 자세한 어간 추출 알고리즘은 다음 웹사이트를 참고한다.
<http://snowball.tartarus.org/algorithms/porter/stemmer.html>
- 어간 추출은 원형 복원(lemmatizing)의 일종이다.
- 원형 복원은 같은 의미를 가지는 여러 단어를
가장 근본적인 형태, 즉 사전형으로 통일하는 작업이다.

POS TAGGING

- 품사(POS, part-of-speech)는 낱말을 문법적인 기능이나 형태, 뜻에 따라 구분한 것이다.
- 품사의 구분은 언어마다 그리고 학자마다 다르다.
- NLTK에서는 펜 트리뱅크 태그세트(Penn Treebank Tagset)라는 것을 이용한다.
- 다음은 펜 트리뱅크 태그세트에서 사용하는 품사의 예이다.
 - NN 명사(단수형 혹은 집합형)
 - PRP 인칭대명사
 - CD 서수
 - DT 관형사
 - VBP 동사 현재형

품사 태그셋 (영어)

Tag	Description
CC	Coordinating conjunction
CD	Cardinal number
DT	Determiner
EX	Existential there
FW	Foreign word
IN	Preposition or subordinating
JJ	Adjective
JJR	Adjective, comparative
JJS	Adjective, superlative
LS	List item marker
MD	Modal
NN	Noun, singular or mass
NNS	Noun, plural
NNP	Proper noun, singular
NNPS	Proper noun, plural
PDT	Predeterminer
POS	Possessive ending
PRP	Personal pronoun

Tag	Description
PRP\$	Possessive pronoun
RB	Adverb
RBR	Adverb, comparative
RBS	Adverb, superlative
RP	Particle
SYM	Symbol
TO	to
UH	Interjection
VB	Verb, base form
VBD	Verb, past tense
VBG	Verb, gerund or present participle
VBN	Verb, past participle
VBP	Verb, non-3rd person singular
VBZ	Verb, 3rd person singular present
WDT	Wh-determiner
WP	Wh-pronoun
WP\$	Possessive wh-pronoun
WRB	Wh-adverb

국내 태그셋

- "21세기 세종계획 품사 태그세트"를 비롯하여 다양한 품사 태그세트가 있다.
- 세종계획 품사 태그세트의 자세한 내용은 "(21세기 세종계획)국어 기초자료 구축" 보고서의 "어절 분석 표지 표준안"을 참조한다.

언어 종류

- 고립어

대부분의 형태소가 그 자체로 낱말이 되는 언어인데, 중국어, 베트남어 등이 이에 속한다.
한자 하나가 나, 친구, 물, 말 등의 단어를 뜻하는 경우이다.

예) 중국어

- 굴절어

종합어의 한 분류로서, 한 문장 속의 기능에 따라 단어 형태 자체가 변한다.

예) 영어

- 교착어

굴절어와 고립어의 중간 단계이다. 단어의 중심이 되는 형태소에 접두사/접미사와 다른 형태소들이 덧붙어 단어가 구성된다.

예) 한국어

언어 사례

- 굴절어 : I, Me, My, 모양 자체가 변한다.
 - 교착어 : 나는, 내가, 나를, 나의, 어간+어미 = 명사+조사
 - 고립어 : 我, 我的, 모양이 변하는 일 없다.
-
- 굴절어 : Go, went, 모양 자체가 변한다.
 - 교착어 : 가다, 갔다, 어간+어미
 - 고립어 : 去, 去了, 모양이 변하는 일 없다.



KONLPY

KOREAN NLP IN PYTHON

KoNLPy

- KoNLPy is a Python package for Korean natural language processing.
한국어 정보처리를 위한 파이썬 패키지
- 지원 기능
 - 한국어 말뭉치
kolaw, kobill
 - 한국어 처리 유틸리티
한글이 리스트나 딕셔너리의 내부에 있을 때도 한글 글자 모양을 정상적으로 보여주는 pprint 유틸리티 함수 제공.
 - 형태소 분석 및 품사 태깅
tag 서브패키지에서 형태소 분석을 위한 5개의 클래스 제공
morphs(), nouns(), pos()

KoNLPy 말뭉치

- kolaw
 - 한국 법률 말뭉치
 - constitution.txt
- kobill
 - 대한민국 국회 의안 말뭉치. 파일 ID는 의안 번호를 의미
 - 1809890.txt - 1809899.txt
- kolaw 헌법 파일 경로
`/usr/local/Cellar/python3/3.6.3/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-packages/konlpy/data/corpus/kolaw/constitution.txt`

KoNLPy 지원 형태소 분석기

- 사전은 대부분 말뭉치를 이용해 구축되었으며 형태소 분석 및 품사 태깅에 사용됨.
- Kkma
<http://kkma.snu.ac.kr/>
- Hannanum
<http://semanticweb.kaist.ac.kr/hannanum/>
- Twitter
<https://github.com/twitter/twitter-korean-text/>
- Komoran
http://www.shineware.co.kr/?page_id=835
- Mecab
<https://bitbucket.org/eunjeon/mecab-ko-dic>

형태소 분석기 성능 비교

- Kkma: 5.6988 secs
- Komoran: 5.4866 secs
- Hannanum: 0.6591 secs
- Twitter: 1.4870 secs
- Mecab: 0.0007 secs
- 실행시간: 10만 문자의 문서를 대상으로 각 클래스의 pos 메소드를 실행하는데 소요되는 시간.

형태소 분석기 성능 비교

- Kkma: 35.7163 secs
- Komoran: 25.6008 secs
- Hannanum: 8.8251 secs
- Twitter: 2.4714 secs
- Mecab: 0.2838 secs
- 문자의 개수를 늘려감에 따라 모든 클래스의 실행 시간은 기하급수적으로 증가합니다.

Hannanum 형태소 분석기

- 한나눔 시스템 사전은 KAIST 말뭉치를 이용해 생성 (4.7MB)
- 위치
`./konlpy/java/data/kE/dic_system.txt`
- 예시

나라경제 ncn
나라기획 nqq
나라기획회장 ncn
나라꽃 ncn
나라님 ncn
나라도둑 ncn
나라따르 pvg
- 사용자 사전에 새로운 항목을 추가하려면 `./konlpy/java/data/kE/dic_user.txt` 수정

Kkma 형태소 분석기

- Kkma 시스템 사전은 세종 말뭉치를 이용해 생성 (32MB)
- 위치
꼬꼬마 형태소 분석기의 .jar 파일 안에 위치. 사전 파일을 직접 보기 위해서는 꼬꼬마 미러 사용
- 예시
아니/IC
후우/IC
그래서/MAC
그러나/MAC
그러니까/MAC
그러면/MAC
그러므로/MAC

MeCab-ko 형태소 분석기

- Mecab 시스템 사전은 세종 말뭉치로 만들어진 CSV 형태의 사전 (346MB)
- 위치
컴파일된 사전은 /usr/local/lib/mecab/dic/mecab-ko-dic (또는 MeCab 설치시 지정 경로)
- 예시

가오토,0,0,0,NNG,* ,F,가오토,* ,*,*,*,*
갑툭튀,0,0,0,NNG,* ,F,갑툭튀,* ,*,*,*,*
강퇴,0,0,0,NNG,* ,F,강퇴,* ,*,*,*,*
개드립,0,0,0,NNG,* ,T,개드립,* ,*,*,*,*
갠소,0,0,0,NNG,* ,F,갠소,* ,*,*,*,*
고퀄,0,0,0,NNG,* ,T,고퀄,* ,*,*,*,*
광삭,0,0,0,NNG,* ,T,광삭,* ,*,*,*
- KoNLPy의 Mecab() 클래스는 윈도우에서 지원되지 않습니다.

MeCab

- 오픈소스 형태소 분석 엔진 MeCab(메카브)
- MeCab, originally a Japanese morphological analyzer and POS tagger developed by the Graduate School of Informatics in Kyoto University, was modified to MeCab-ko by the Eunjeon Project to adapt to the Korean language.
- 학습용 데이터와 분석 사전을 준비하면 다른 언어 분석 가능
- MeCab를 이용한 한국어 형태소 분석
<http://porocise.sakura.ne.jp/wiki/korean/mecab.ko>
- 한국어 버전
<https://bitbucket.org/eunjeon/mecab-ko>

MeCab 도구

- **한국어 학습 보조 도구**

입력된 현대한국어 문장을 분석하여 단어의 학습 수준을 표시하거나 한자 표기로 변환하는 등 한국어 학습에 도움이 되는 정보를 제공합니다. 단어마다 웹 사전으로의 링크를 생성할 수도 있습니다.

<http://porocise.sakura.ne.jp/korean/mecab/main.html>

- **형태소 분석 / 한자 표기 변환**

현대한국어 문장을 분석하여 품사를 표시하거나 문장 중의 한자어를 한자 표기로 바꿔 줍니다.

<http://porocise.sakura.ne.jp/korean/mecab/analyzer.html>

- **히라가나의 한글 전사**

입력된 히라가나 표기를 한글로 전사합니다. 또한 한자가 섞인 일본어 문장도 MeCab로 분석한 다음에 한글로 전사할 수 있습니다.

<http://porocise.sakura.ne.jp/korean/hira2han/index.html>

MeCab 콘솔

1. 터미널에서 직접 사용할 수 있다.
2. mecab 명령만 입력하면 형태소 분석 모드로 들어간다.
3. 분석하고 싶은 문장 쓰고 엔터. 분석 결과를 보여주고 EOS 출력
4. 도움말 mecab -h
5. 분석 모드를 탈출하는 명령은 없다. 종료는 ctrl + c.

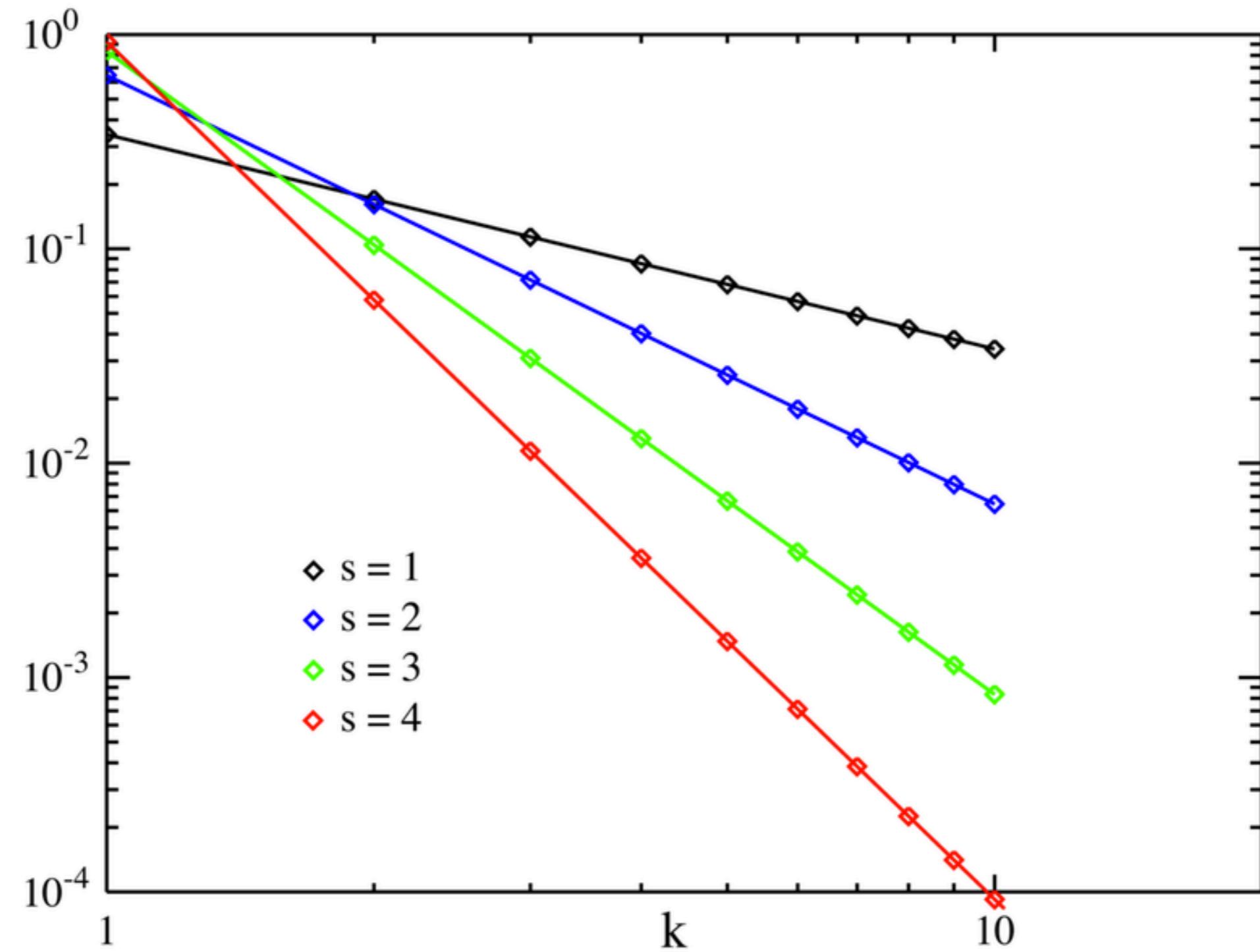
MeCab test

```
import konlpy  
  
mecab = konlpy.tag.Mecab()  
  
print(mecab.morphs('영등포구청역에 있는 맛집 좀 알려주세요.'))  
# ['영등포구', '청역', '에', '있', '는', '맛집', '좀', '알려', '주', '세요', '.']  
  
print(mecab.nouns('우리나라에는 무릎 치료를 잘하는 정형외과가 없는가!'))  
# ['우리', '나라', '무릎', '치료', '정형외과']  
  
print(mecab.pos('자연주의 쇼핑몰은 어떤 곳인가?'))  
# [('자연', 'NNG'), ('주', 'NNG'), ('의', 'JKG'), ('쇼핑몰', 'NNG'), ('은', 'JX'), ('어떤', 'MM'),  
('곳', 'NNG'), ('인가', 'VCP+EF'), ('?', 'SF')]
```

지프의 법칙(Zipf's law)

- 수학적 통계를 바탕으로 밝혀진 경험적 법칙
- 물리 및 사회과학 분야에서 연구된 많은 종류의 정보들이 지프 분포에 가까운 경향을 보이는 것을 뜻한다.
- 지프 분포는 이산 멱법칙 확률분포와 관계된 확률분포의 하나이다.
- 미국의 언어학자인 조지 킹슬리 지프가 최초로 이 법칙을 공식 제안(Zipf 1935, 1949).
- 지프의 법칙에 따르면 어떠한 자연어 말뭉치 표현에 나타나는 단어들을 그 사용 빈도가 높은 순서대로 나열하였을 때, 모든 단어의 사용 빈도는 해당 단어의 순위에 반비례한다. 따라서 가장 사용 빈도가 높은 단어는 두 번째 단어보다 빈도가 약 두 배 높으며, 세 번째 단어보다는 빈도가 세 배 높다. 예를 들어, 브라운 대학교 현대 미국 영어 표준 말뭉치의 경우, 가장 사용 빈도가 높은 단어는 영어 정관사 "the"이며 전체 문서에서 7%의 빈도(약 백만 개 남짓의 전체 사용 단어 중 69,971회)를 차지한다. 두 번째로 사용 빈도가 높은 단어는 "of"로 약 3.5% 남짓(36,411회)한 빈도를 차지하며, 세 번째로 사용 빈도가 높은 단어는 "and"(28,852회)로, 지프의 법칙에 정확히 들어 맞는다. 약 135개 항목의 어휘만으로 브라운 대학 말뭉치의 절반을 나타낼 수 있다.
- 지프의 법칙은 데이터의 순위와 빈도를 각 축에 로그 스케일로 나타낸 그래프를 통해 쉽게 확인할 수 있다.

지프의 법칙(Zipf's law)



세종 태그셋 (1)

- 한글 형태소 품사(POS) 태그표

<http://kkma.snu.ac.kr/documents/?doc=postag>

대분류	세종 품사 태그		심광섭 품사 태그		KKMA 단일 태그 V1.0						확률태 그	저장사전
	태그	설명	Class	설명	묶음 1	묶음 2	태그	설명				
체언	NNG	일반 명사	NN	명사	N	NN	NNG	보통 명사		NNA	noun.dic	
	NNP	고유 명사					NNP	고유 명사				
	NNB	의존 명사	NX	의존 명사			NNB	일반 의존 명사		NNB	simple.dic	
				단위 명사			NNM	단위 의존 명사				
	NR	수사	NU	수사		NR	NR	수사		NR		
용언	NP	대명사	NP	대명사		NP	NP	대명사		NP		
	VV	동사	VV	동사	V	VV	VV	동사		VV	verb.dic	
	VA	형용사	AJ	형용사		VA	VA	형용사		VA		
	VX	보조 용언	VX	보조 동사		VX	VXV	보조 동사		VX		
			AX	보조 형용사		VXA	VXA	보조 형용사		VXA		
관형사	VCP	긍정 지정사	CP	서술격 조사 '이다'	M	VC	VCP	긍정 지정사, 서술격 조사 '이다'		VCP	raw.dic	
	VCN	부정 지정사				VC	VCN	부정 지정사, 형용사 '아니다'		VCN		
	MM	관형사	DT	일반 관형사		MD	MDT	일반 관형사		MD		
			DN	수 관형사		MD	MDN	수 관형사		MD		
부사	MAG	일반 부사	AD	부사		MA	MAG	일반 부사		MAG	simple.dic	
	MAJ	접속 부사				MA	MAC	접속 부사		MAC		

세종 태그셋 (2)

감탄사	IC	감탄사	EX	감탄사	I	IC	IC	감탄사	IC					
조사	JKS	주격 조사	JO	조사	J	JK	JKS	주격 조사	JKS	raw.dic				
	JKC	보격 조사					JKC	보격 조사	JKC					
	JKG	관형격 조사					JKG	관형격 조사	JKG					
	JKO	목적격 조사					JKO	목적격 조사	JKO					
	JKB	부사격 조사					JKM	부사격 조사	JKM					
	JKV	호격 조사					JKI	호격 조사	JKI					
	JKQ	인용격 조사					JKQ	인용격 조사	JKQ					
	JX	보조사					JX	보조사	JX					
	JC	접속 조사					JC	접속 조사	JC					
선어말 어미	EP	선어말 어미	EP	선어말 어미	E	EP	EPH	존칭 선어말 어미	EP	raw.dic				
							EPT	시제 선어말 어미						
							EPP	공손 선어말 어미						
어말 어미	EF	종결 어미	EM	어말 어미	E	EF	EFN	평서형 종결 어미	EF	raw.dic				
							EFQ	의문형 종결 어미						
	EC	연결 어미					EFO	명령형 종결 어미						
							EFA	청유형 종결 어미						
							EFI	감탄형 종결 어미						
	ET	전성 어미					EFR	존칭형 종결 어미						
		EC				ECE	대등 연결 어미	raw.dic						
						ECD	의존적 연결 어미							
						ECS	보조적 연결 어미							
	ETN	명사형 전성 어미				ET	ETN	명사형 전성 어미	ETN	raw.dic				
							ETD	관형형 전성 어미	ETD					

세종 태그셋 (3)

접두사	XPN	체언 접두사	PF	접두사	X	XP	XPN	체언 접두사	XPN	simple.dic	
							XPV	용언 접두사	XPV		
접미사	XSN	명사 파생 접미사	SN	명사화 접미사		XS	XSN	명사 파생 접미사	XSN		
	XSV	동사 파생 접미사	SV	동사화 접미사			XSV	동사 파생 접미사	XSV		
	XSA	형용사 파생 접미사	SJ	형용사화 접미사			XSA	형용사 파생 접미사	XSA		
			SA	부사화 접미사			XSM	부사 파생 접미사	XSM		
			SF	기타 접미사			XSO	기타 접미사	XSO		
어근	XR	어근	XR			XR	XR	어근	XR		
부호	SF	마침표물음표,느낌표	SY	부호 의래어		SF	SF	마침표물음표,느낌표	SF	Symbol class	
	SP	쉼표,가운뎃점,콜론,빗금				SP	SP	쉼표,가운뎃점,콜론,빗금	SP		
	SS	따옴표,괄호표,줄표				SS	SS	따옴표,괄호표,줄표	SS		
	SE	줄임표				SE	SE	줄임표	SE		
	SO	불임표(물결,숨김,빠짐)				SO	SO	불임표(물결,숨김,빠짐)	SO		
	SW	기타기호 (논리수학기호,화폐기호)				SW	SW	기타기호 (논리수학기호,화폐기호)	SW		
분석 불능	NF	명사추정범주	NR	미등록어	U	UN	UN	명사추정범주	NNA	N/A	
	NV	용언추정범주				UV	UV	용언추정범주	N/A		
	NA	분석불능범주				UE	UE	분석불능범주	N/A		
한글 이외	SL	외국어			O	OL	OL	외국어	NNA		
	SH	한자				OH	OH	한자	NNA		
	SN	숫자				ON	ON	숫자	NR		

GRAMS

- Unigram
단어 1개
- Bigram
단어 2개가 연속된 쌍
- Trigram
단어 3개가 연속된 쌍

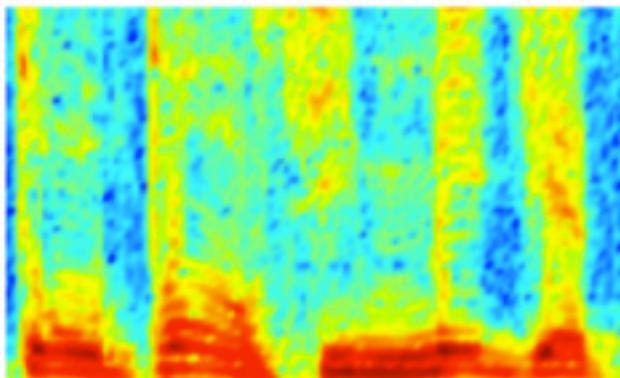


WORD EMBEDDING

WORD2VEC, COUNT VECTOR

DENSE & SPARSE

AUDIO



Audio Spectrogram

DENSE

IMAGES

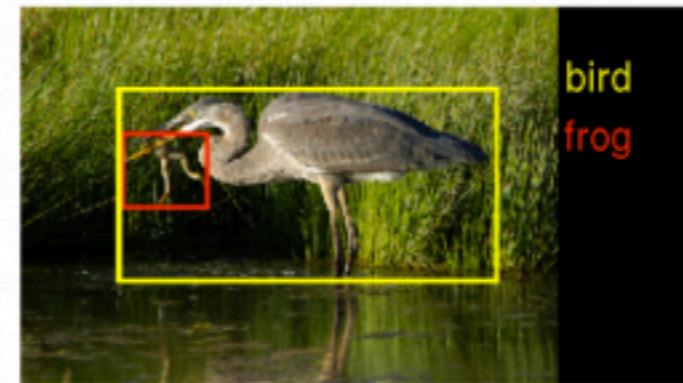


Image pixels

DENSE

TEXT



Word, context, or
document vectors

SPARSE

VECTOR SPACE MODEL

- VSM은 자연어 처리(NLP)에서 오랜기간 사용되어 왔다. 이 방법은 같은 context에 있는 단어는 같은 semantic meaning을 공유한다고 가정한다. 이런 가정을 Distributional Hypothesis라고 한다.
- Distributional Hypothesis
 - 1.비슷한 분포를 가진 단어들은 비슷한 의미를 가진다.
 - 2.비슷한 분포를 가졌다는 것은 기본적으로 단어들이 같은 문맥에서 등장한다는 것을 의미
- 종류
 - 1.count-based methods (e.g. Latent Semantic Analysis)
어떤 단어가 이웃 단어들과 같이 등장한 횟수를 계산하고 이 통계를 small, dense vector로 맵핑.
 - 2.predictive methods (e.g. neural probabilistic language models)
small, dense embedding vectors로 표현된 이웃 단어들을 이용해서 직접적으로 단어를 예측.

WORD EMBEDDINGS

- 오디오와 이미지와 같은 고차원의 데이터는 raw data로부터 인코딩(encoding)된 벡터로 특징을 뽑아내는 것이 쉽다.
- 텍스트의 경우, 이런식의 처리가 어렵기 때문에 전통적인 방법에서는 하나의 단어를 discrete atomic symbol로 표현
- 예시
cat = Id537, dog = Id143
- vector representation을 통해 위의 아래와 같은 두 가지 문제점을 해결할 수 있다.
 - 1.인코딩이 무작위적이고 데이터간의 관계를 보여주지 않는다. 따라서 모델이 “cats”라는 단어로부터 배운 특징을 “dogs”라는 단어를 처리할 때 적절하게 이용할 수 없다.
 - 2.단어들을 discrete ID로 맵핑함으로써 데이터를 sparse하게 만든다. 따라서 통계적 모델을 성공적으로 트레이닝 하려면 많은 데이터가 필요하다.

WORD EMBEDDINGS

- 주파수 기반 임베딩
 1. 카운트 벡터(Count Vector)
 2. TF-IDF(Term Frequency - Inverse Document Frequency)
 3. 동시 발생 행렬(Co-Occurrence Matrix)
- 예측 기반 임베딩
 1. CBOW(Continuous Bag Of Words)
 2. 스kip 그램(Skip-gram)

COUNT VECTOR

- 벡터의 갯수를 세는 방법이다. 말뭉치(Corpus)에 여러개의 문서들(Documents)이 있다고 했을 때 각 문서마다 고유의 토큰(Token)을 세어 이를 행렬(Matrix)로 나타내는 방법이다.
- 행(Row)은 각각의 문서를 나타내고 열은 각각의 고유 토큰(Token)을 나타낸다.

COUNT VECTOR

- 벡터의 갯수를 세는 방법이다. 말뭉치(Corpus)에 여러 개의 문서들(Documents)이 있다고 했을 때 각 문서마다 고유의 토큰(Token)을 세어 이를 행렬(Matrix)로 나타내는 방법이다. 행(Row)은 문서를 나타내고 열은 고유의 토큰(Token)을 나타낸다.
- D1 : He is a lazy boy. She is also lazy.
- D2 : Neeraj is a lazy person.
- 토큰 목록 : He, She, Lazy, Boy, Neeraj, person
- D = 2, N = 6

	He	She	lazy	boy	Neeraj	person
D1	1	1	2	1	0	0
D2	0	0	1	0	1	1

동시 발생 행렬

Quick Brown Fox Jump Over The Lazy Dog

Quick Brown Fox Jump Over The Lazy Dog

He is not lazy He is intelligent He is smart

He is not lazy He is intelligent He is smart

He is not lazy He is intelligent He is smart

He is not lazy He is intelligent He is smart

동시 발생 행렬

- He is not lazy. He is intelligent. He is smart

	He	is	not	lazy	intelligent	smart
He	0	4	2	1	2	1
is	4	0	1	2	2	1
not	2	1	0	1	0	0
lazy	1	2	1	0	0	0
intelligent	2	2	0	0	0	0
smart	1	1	0	0	0	0

WORD2VEC

- raw text로부터 word embedding을 학습하는 계산 효율성이 좋은 predictive model.

1. CBOW(Continuous Bag-Of-Words) model

- 소스 컨텍스트에서 타겟 단어 예측
- 예를 들어, 'the cat sits on the'라는 소스 컨텍스트로부터 'mat'이라는 타겟 단어 예측
- CBOW는 smaller 데이터셋에 적합

2. Skip-Gram model

- 타겟 단어로부터 소스 컨텍스트 예측
- 예를 들어, 'mat'이라는 타겟 단어로부터 'the cat sits on the'라는 소스 컨텍스트 예측
- Skip-Gram model은 larger 데이터셋에 적합



NLP

NATURAL LANGUAGE PROCESSING

자연어를 이해하는 방법

- 단어가 어떤 의미를 가지고 있는지를 알기 위해서는 문장의 컨텍스트(Context)를 이해해야 한다.
 - 사람은 Apple이 사과인지 회사인지 문맥을 통해 바로 구분할 수 있지만 기계는 할 수 없다.
 - '메시'로부터 '축구'와 '호나우도'처럼 단어간의 의미적 관계도 파악할 수 있지만 기계는 어렵다.
- 그렇기 때문에 단어를 수치적으로 표현하여 기계가 이해할 수 있도록 해야 한다.
- 이를 Word Embedding이라고 표현하는데, 간단히 말해서 텍스트를 숫자로 표현하는 것을 말하고, 같은 문자라고 하더라도 다른 수치로 표현할 수 있다.
- 단어를 사전을 사용하여 매핑하고 이를 벡터로 만드는 것이다.
ex) "the cat sat on the mat"
- [the, cat, sat, on, mat]의 단어별로 분류하고 이를 One hot encoding을 통해 0 또는 1로 나타낸다.
- 예를 들어 'cat'은 [0, 1, 0, 0, 0, 0]이며, 'the'의 경우 [1, 0, 0, 0, 0]으로 나타낸다.
- 벡터화를 시켜 숫자로 표현하면 분류나 회귀 등의 다양한 분석이 가능해진다.

문맥(CONTEXT)

- 유유상종

친구를 보면 그 사람을 안다.

- 단어의 주변을 보면 그 단어를 안다.

You shall know a word by the company it keeps.

-- J.R. Firth (1957)

예시 1

- 빈 칸에 맞는 말을 찾으세요.
 - 새로운 디자인의 선이 _____하다.
 - 야, 부엌 좀 _____히 하자.
 - 깜박하고 책상 위를 _____하게 치우지 않았다.

예시 2

- '매나니'라는 단어를 아시나요?
 - 그 녀석 무슨 배짱인지 '매나니'로 와서 일을 하겠다고 한다.
 - 삽이라도 있어야 땅을 파지 '매나니'로야 어떻게 하겠나?
 - SO만 있으면 코딩은 '매나니'로도 할 수 있다고 한다.

매나니

- 순 우리말(다음 사전)

일하는 데 아무런 도구나 연장이 없는 맨손

뜻/문법

고려대 ✓ 우리말샘

명사

(1) 일하는 데 아무런 도구나 연장이 없는 맨손.

| 그 녀석 무슨 배짱인지 매나니로 와서 일을 하겠다고 한다.

(2) 아무런 반찬이 없는 맨밥.

| 그는 입맛을 잃었다며 매나니로 요기를 하였다.

문맥

다시 말해, 단어의 의미는 해당 단어의 **문맥(context)**이 담고 있다.

영희가 철수에게 미안하다고 사과하면서

나무에서 갓 딴 맛있는 사과를 주었습니다

주변부 단어들이 “사과”的 문맥적 특성을 나타냄

- 문맥(context) := 정해진 구간(window) 또는 문장/문서 내의 단어들
 - 보통 영어의 구간은 좌우 1-8개 단어, 총 3-17개 단어의 문맥을 본다*
- 두 단어의 문맥이 비슷하면 "의미적"으로 유사한 단어
 - ex: "PyCon"은 "R"보다 "Python"에 가깝다?

동시 발생

"Co-occurrence"

두 단어가 정해진 구간 내에서 동시에 등장함

CO-OCCURRENCE 정의

```
documents = [["나는", "파이썬", "이", "좋다"],  
             ["나는", "R", "이", "좋다"]]
```

1. **Term-document matrix** ($X_{td} \in \mathbb{R}^{|V| \times M}$): 한 문서에 같이 등장하면 비슷한 단어
 - Computation이 문서의 개수 M 에 비례(!)

```
x_td = [[1, 1], # 나는  
         [1, 0], # 파이썬  
         [0, 1], # R  
         [1, 1], # 이  
         [1, 1]] # 좋다
```

1. **Term-term matrix** ($X_{tt} \in \mathbb{R}^{|V| \times |V|}$): 단어가 문맥 내에 같이 존재하면 비슷한 단어
 - 문맥의 길이가 짧을수록 syntactic, 길수록 semantic 정보를 포함
 - 앞뒤로 단어를 두 개씩 보는 경우 (문맥의 길이==5):

```
x_tt = [[0, 1, 1, 2, 0], # 나는  
         [1, 0, 0, 1, 1], # 파이썬  
         [1, 0, 0, 1, 1], # R  
         [2, 1, 1, 0, 2], # 이  
         [0, 1, 1, 2, 0]] # 좋다
```

단어간 유사도

Co-occurrence matrix를 있는 그대로 이용해도 단어 간 유사도를 구할 수는 있다.

```
import math

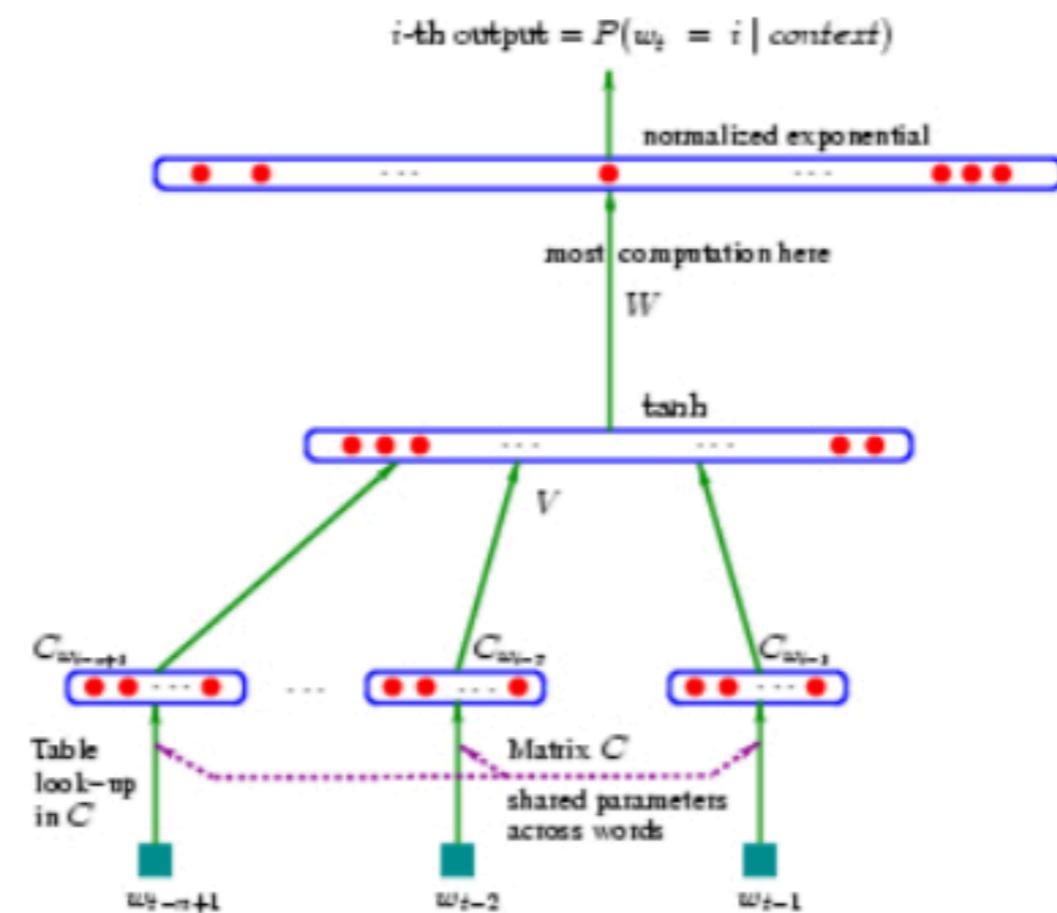
def dot_product(v1, v2):
    return sum(map(lambda x: x[0] * x[1], zip(v1, v2)))

def cosine_measure(v1, v2):
    prod = dot_product(v1, v2)
    len1 = math.sqrt(dot_product(v1, v1))
    len2 = math.sqrt(dot_product(v2, v2))
    return prod / (len1 * len2)
```

- 하지만:
 - 값들이 너무 skewed되어 있고 (즉, 빈도 높은 단어와 낮은 단어의 격차가 큼)
 - 정보성 낮은 단어 때문에 discriminative하지 않음 (ex: list('은는이가'))
- 유사도 정보가 담긴 단어 벡터를 구하는 더 나은 방법은?

NEURAL EMBEDDINGS

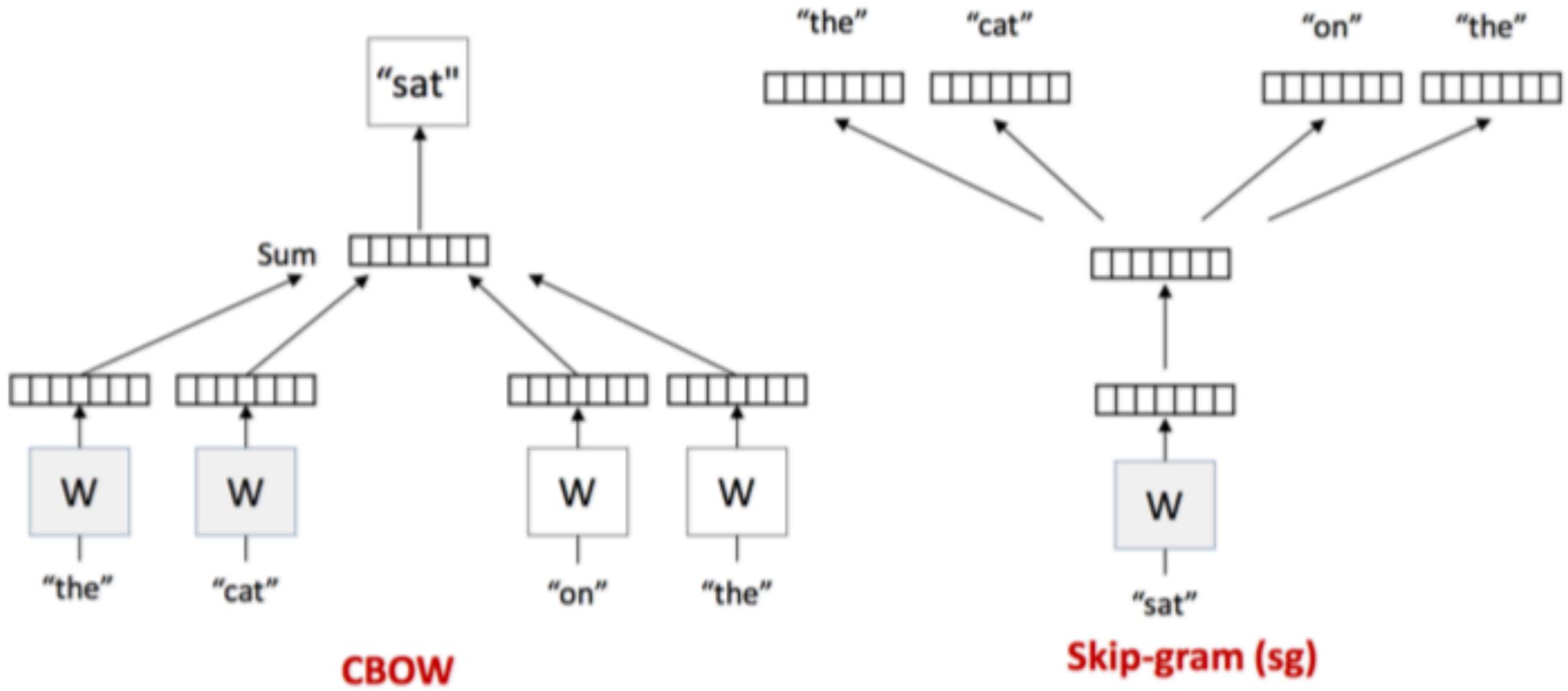
- 아이디어: 문맥에 있는 단어를 예측하라!
 - 언어 모델(language model) 활용
 - ex: ["나는", "파이썬", "이", "좋다"] 다음에 뭐가 나올까? (ex: "!", ".", "?")
- 그리고, 학습할 때 뉴럴넷을 쓰자.



Neural network language model (NNLM)*

WORD2VEC

- 요즘 핫한 T. Mikolov의 word2vec도 neural embedding
- 각종 계산 트릭을 적용해서 컴퓨테이션이 빨라짐! (n일 -> m시 간)



북한 신년사

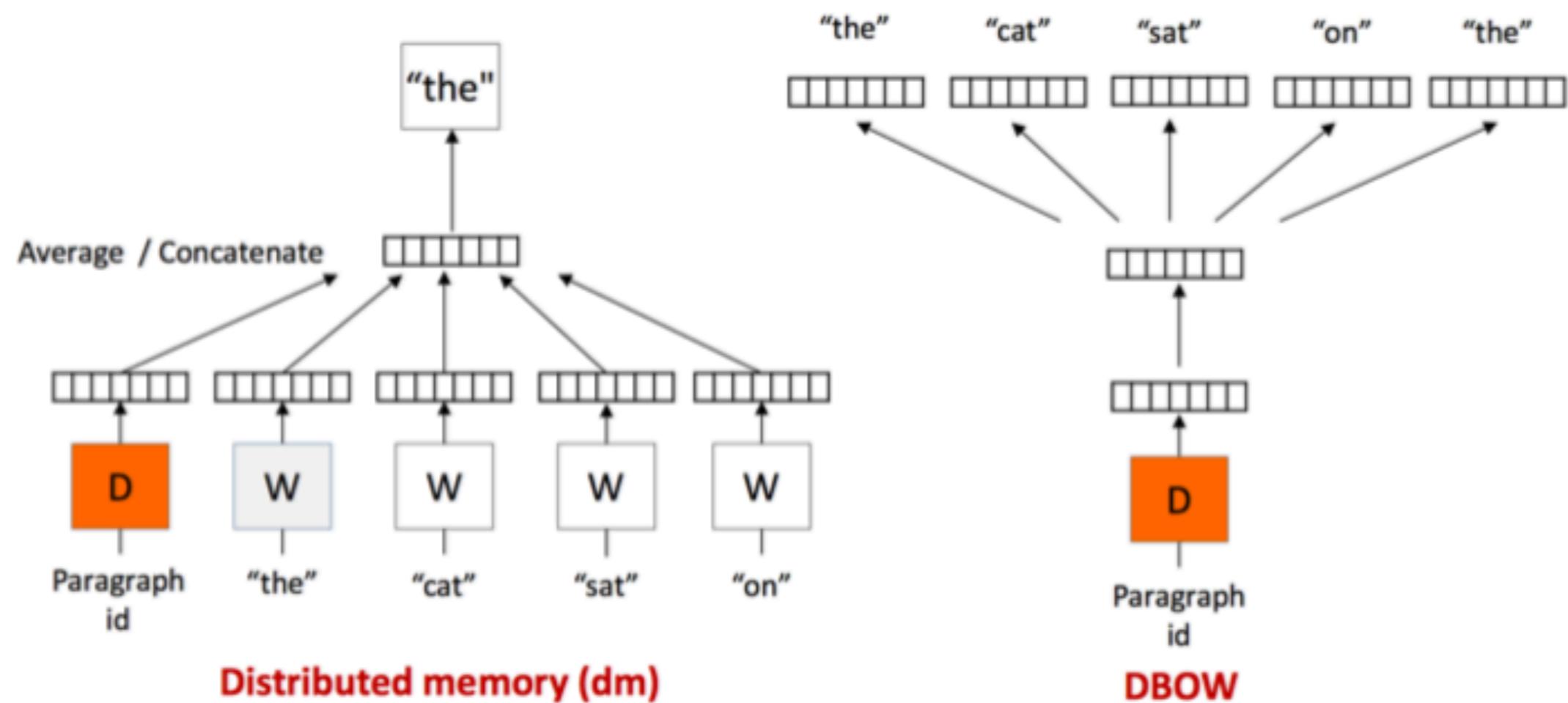


질문!

그럼 문서에 대한 *neural embedding*은?

DOC2VEC

- 아이디어: 문서(또는 문단) 벡터를 마치 단어인 양 학습시키자!
- 장점
 - 차원이 $|V|$ 에 비해 훨씬 적어진다.
 - 단어 벡터와 같은 공간에 문서를 전사할 수 있다.



TERM FREQUENCY

- 문서 벡터의 크기가 단어의 수 $|V|$ 와 같음. 원소들은 양의 정수

```
documents = [  
    ['왕자', '가', '공주', '를', '좋아한다'],  
    ['공주', '가', '왕자', '를', '좋아한다'],  
    ['시녀', '가', '왕자', '를', '싫어한다'],  
    ['공주', '가', '시녀', '를', '싫어한다'],  
    ['시녀', '가', '왕자', '를', '독살한다'],  
    ['시녀', '가', '공주', '를', '독살한다']  
]  
  
words = set(sum(documents, []))  
print(words)  
#=> {'가', '공주', '독살한다', '를', '시녀', '싫어한다', '왕자', '좋아한다'}
```

TERM FREQUENCY

```
def term_frequency(document):
    # do something
    return vector

vectors = [term_frequency(doc) for doc in documents]
print(vectors)
# => [
#     'S1': [1, 1, 0, 1, 0, 0, 1, 1],
#     'S2': [1, 1, 0, 1, 0, 0, 1, 1],
#     'S3': [1, 0, 0, 1, 1, 1, 1, 0],
#     'S4': [1, 1, 0, 1, 1, 1, 0, 0],
#     'S5': [1, 0, 1, 1, 1, 0, 1, 0],
#     'S6': [1, 1, 1, 1, 1, 0, 0, 0]
# ]
```

DOC2VEC

- 문서 벡터의 크기가 단어의 수 $|V|$ 보다 작음. 원소들은 실수.

```
documents = [
```

```
    ['왕자', '가', '공주', '를', '좋아한다'],
    ['공주', '가', '왕자', '를', '좋아한다'],
    ['시녀', '가', '왕자', '를', '싫어한다'],
    ['공주', '가', '시녀', '를', '싫어한다'],
    ['시녀', '가', '왕자', '를', '독살한다'],
    ['시녀', '가', '공주', '를', '독살한다']
```

```
]
```

```
words = set(sum(documents, []))
```

```
print(words)
```

```
#=> {'가', '공주', '독살한다', '를', '시녀', '싫어한다', '왕자', '좋아한다'}
```

DOC2VEC

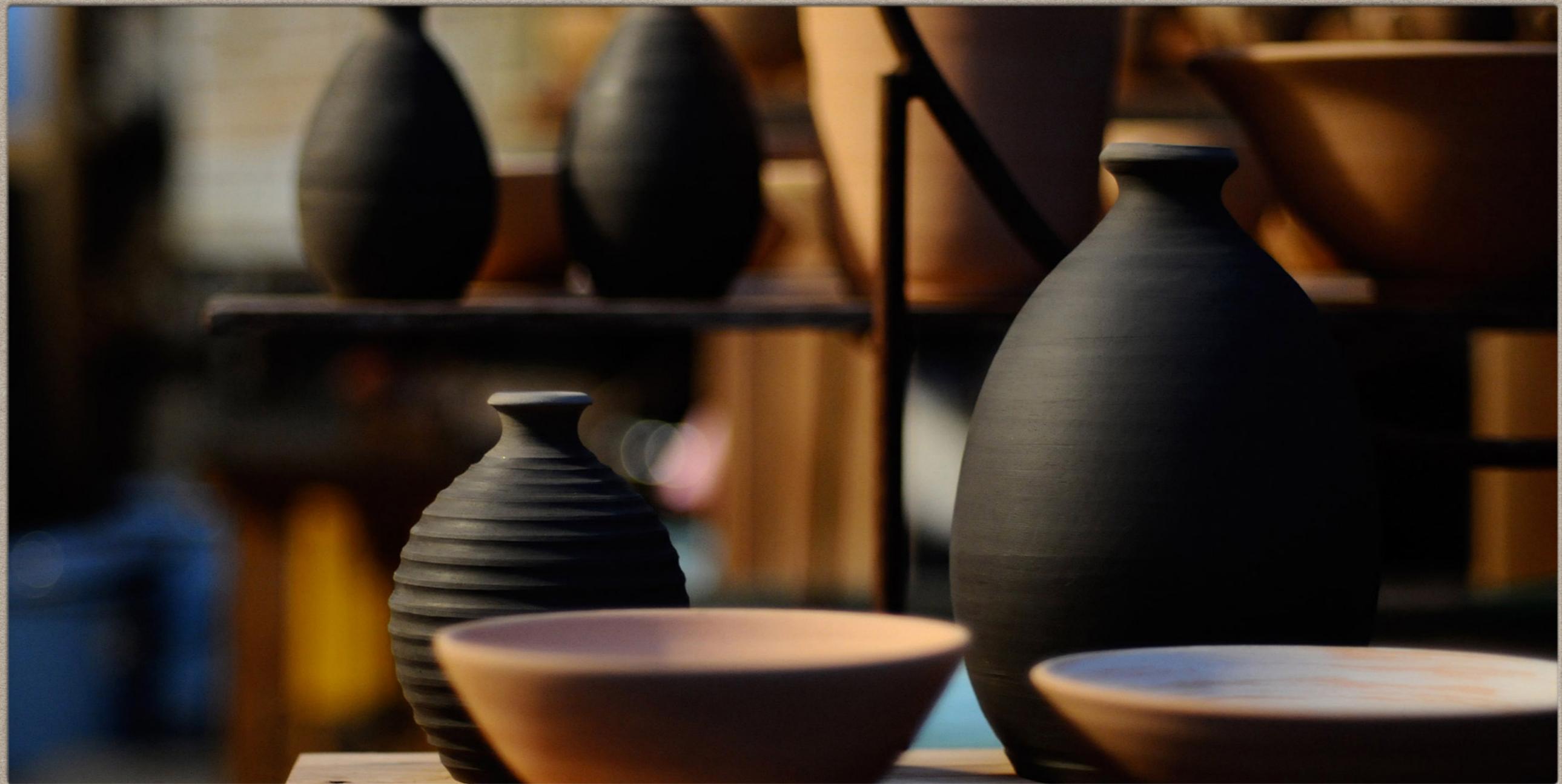
```
def doc2vec(document):
    # do something else (뒤에서 Gensim이 해줄 것!)
    return vector

vectors = [doc2vec(doc) for doc in documents]
print(vectors)
#=> [
#      'S1': [-0.01599941, 0.02135301, 0.0927715],
#      'S2': [0.02333261, 0.00211833, 0.00254255],
#      'S3': [-0.00117272, -0.02043504, 0.00593186],
#      'S4': [0.0089237, -0.00397806, -0.13199195],
#      'S5': [0.02555059, 0.01561624, 0.03603476],
#      'S6': [-0.02114407, -0.01552016, 0.01289922]
# ]
```

요약

- 단어의 문맥을 이용해서 단어를 표현할 수 있다.
- 텍스트의 의미를 벡터로 표현하는 몇 가지 방법:

	Sparse, long vectors	Dense, short vectors
단어	1-hot-vectors	word2vec
문서	bag-of-words (term existance, TF, TF-IDF 등)	doc2vec



gensim
NLP Library

개요

- gensim는 자연어 처리 라이브러리
- 자연어 처리에서 일반적인 기능을 갖춘 nltk에 비해 토픽 모델링에 특화
- word embedding
 - 딥러닝 기반의 자연어 처리의 기본
 - word2vec, glove 등 다양한 알고리즘 존재
 - tensorflow를 사용해서 직접 구현 가능
 - 실제 사용은 word2vec에서만 큼은 케라스보다 쉽고 유용
 - clustering, cosine similarity 등에서 압도적인 우위

개요

- Gensim
 - Open-source vector space modeling and topic modeling toolkit implemented in Python
 - It uses NumPy, SciPy and optionally Cython for performance.
 - Specifically designed to handle large text collections, using data streaming and efficient incremental algorithms
 - It differentiates from most other scientific software packages that only target batch and in-memory processing
- Implementations
 - tf-idf
 - random projections
 - word2vec
 - document2vec
 - hierarchical Dirichlet processes (HDP)
 - latent semantic analysis (LSA, LSI, SVD)
 - latent Dirichlet allocation (LDA)



RNN

OVERVIEW, LSTM, GRU

OVERVIEW

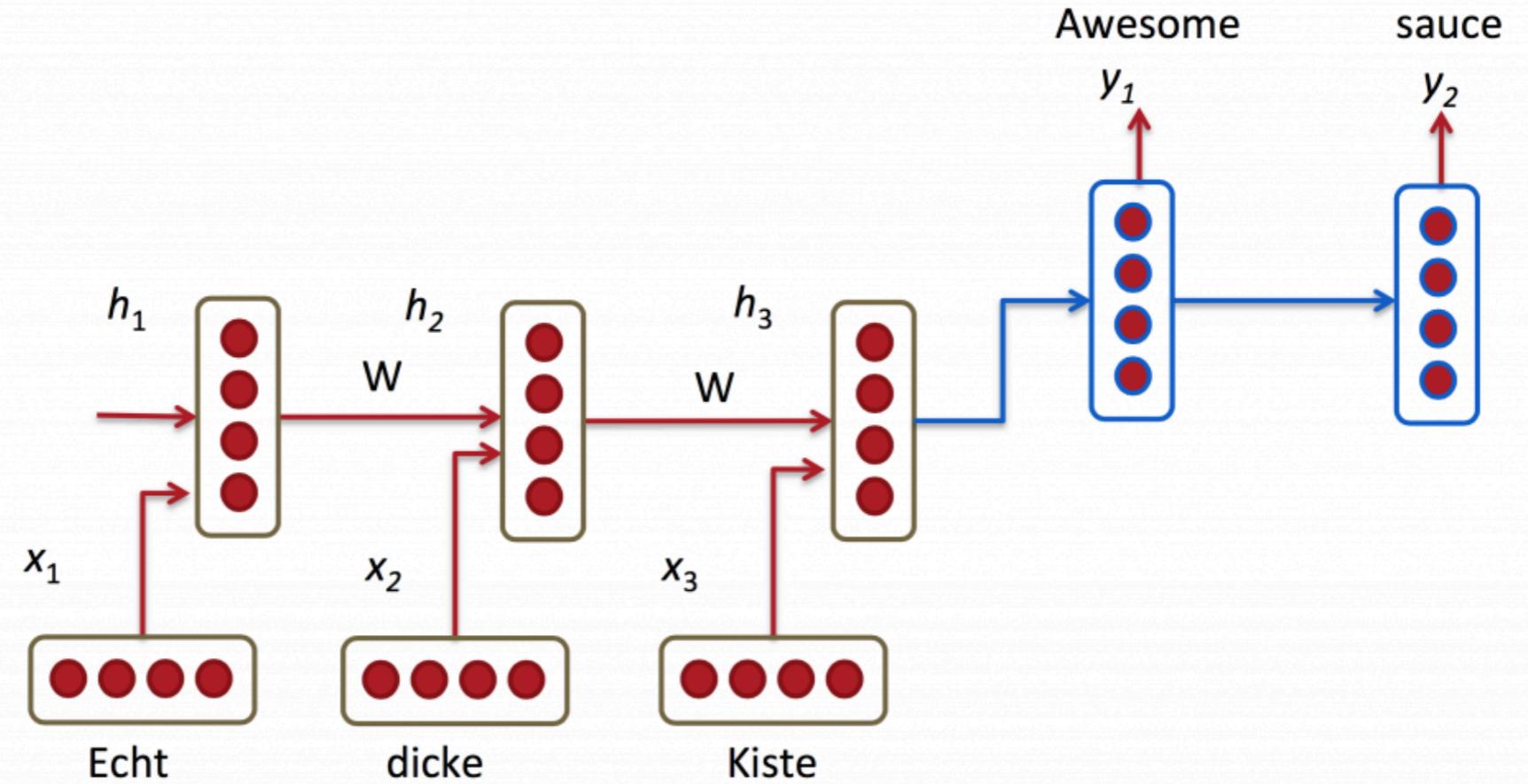
- Recurrent Neural Network
- 히든 노드가 방향을 가진 엣지로 연결돼 순환구조를 이루는(directed cycle) 인공신경망의 한 종류
- 음성, 문자 등 순차적으로 등장하는 데이터 처리에 적합한 모델
- Convolutional Neural Networks(CNN)과 더불어 최근 들어 각광 받고 있는 알고리즘
- 시퀀스 길이에 관계없이 입력과 출력을 받아들일 수 있는 네트워크 구조
- 필요에 따라 다양하고 유연하게 구조를 만들 수 있다는 점이 RNN의 가장 큰 장점

활용 1

- 언어 모델링과 텍스트 생성

언어 모델은 주어진 문장에서 이전 단어들을 보고 다음 단어가 나올 확률을 계산해주는 모델이다. 언어 모델은 어떤 문장이 실제로 존재할 확률이 얼마나 되는지 계산해주기 때문에, 자동 번역의 출력값으로 어떤 문장을 내보내는 것이 더 좋은지 (실생활에서 높은 확률로 존재하는 문장들은 보통 문법적/의미적으로 올바르기 때문) 알려줄 수 있다. 문장에서 다음 단어가 나타날 확률을 계산해주는 주 목적 외의 부수적인 효과로 생성(generative) 모델을 얻을 수 있는데, 출력 확률 분포에서 샘플링을 통해 문장의 다음 단어가 무엇이 되면 좋을지 정한다면 기존에 없던 새로운 문장을 생성할 수 있다. 또한, 학습 데이터에 따라 다양하고 재밌는 여러 가지를 만들어 낼 수도 있다. 언어 모델에서의 입력값은 단어들의 시퀀스 (e.g. one-hot encoded 벡터 시퀀스)이고, 출력은 추측된 단어들의 시퀀스이다. 네트워크를 학습할 때에는 시간 스텝 t 에서의 출력값이 실제로 다음 입력 단어가 되도록 $o_t = x_{\{t+1\}}$ 로 정해준다.

활용 2



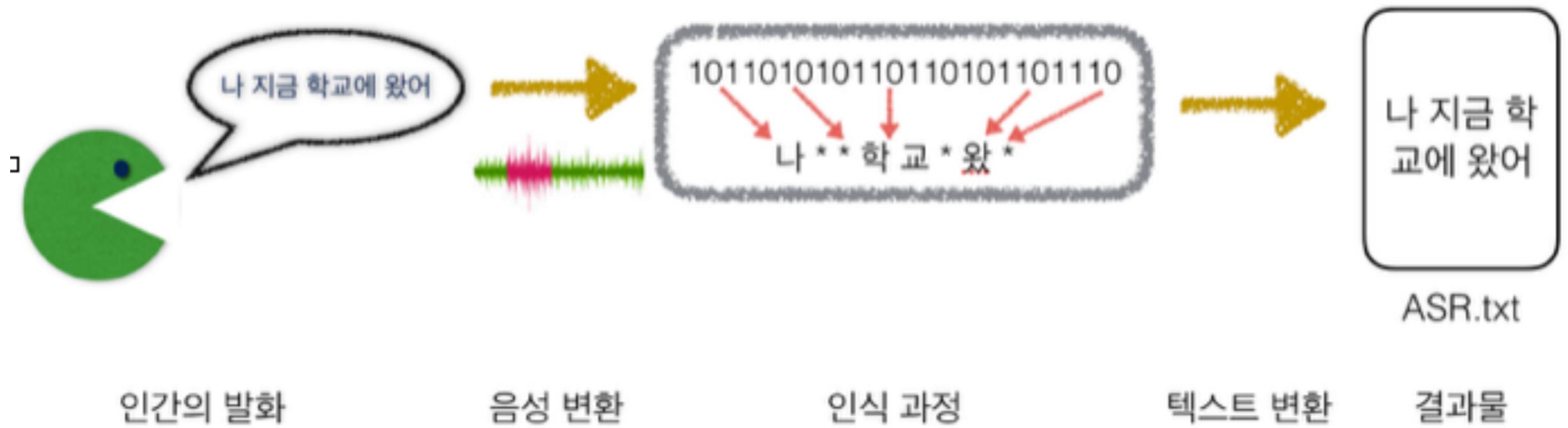
- 자동 번역 (기계 번역)

기계 번역 문제는 입력이 단어들의 시퀀스라는 점에서 언어 모델링과 비슷하지만, 출력값이 다른 언어로 되어있는 단어들의 시퀀스라는 점에서 차이가 있다. 네트워크 상에서의 중요한 차이점은, 입력값을 전부 다 받아들인 다음에서야 네트워크가 출력값을 내보낸다는 점에 있는데, 번역 문제에서는 어순이 다른 문제 등이 있기 때문에 대상 언어의 문장의 첫 단어를 알기 위해선 번역할 문장 전체를 봐야 할 수도 있기 때문이다.

활용 3

- 음성 인식

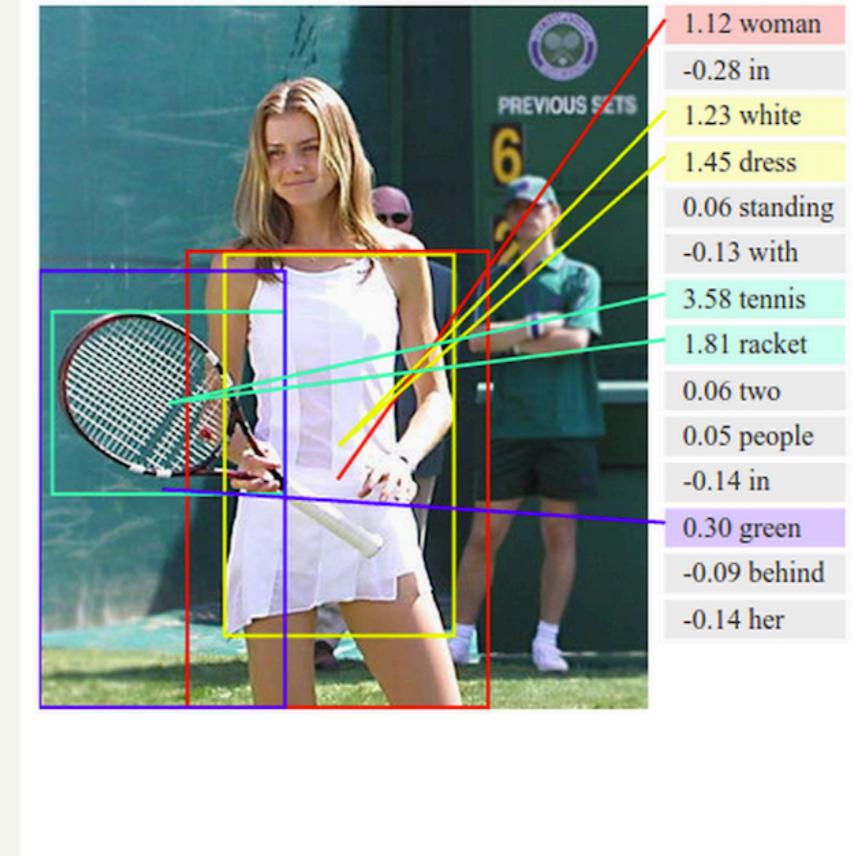
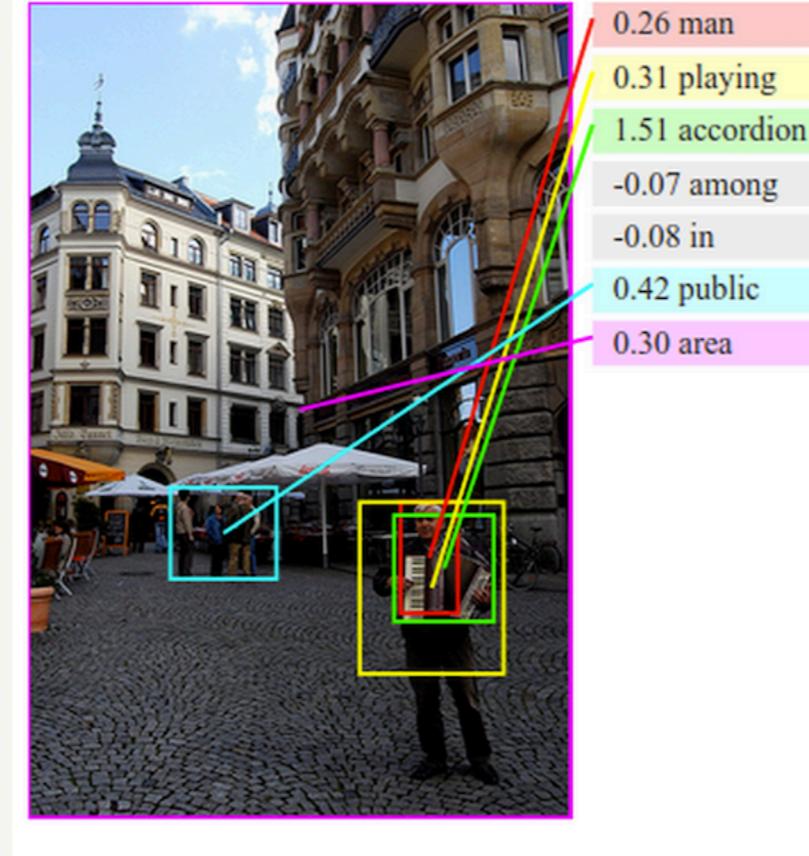
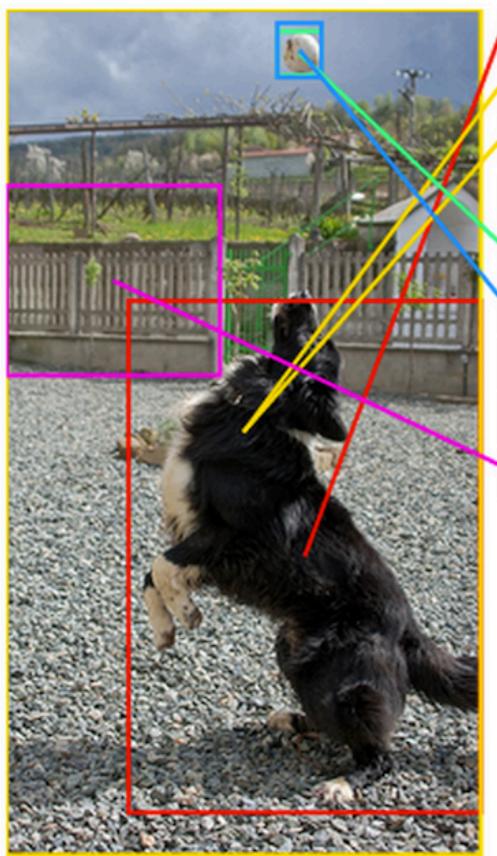
사운드 웨이브의 음향 신호(acoustic signal)를 입력으로 받아들이고, 출력으로는 음소(phonetic segment)들의 시퀀스와 각각의 음소별 확률 분포를 추측할 수 있다.



활용 4

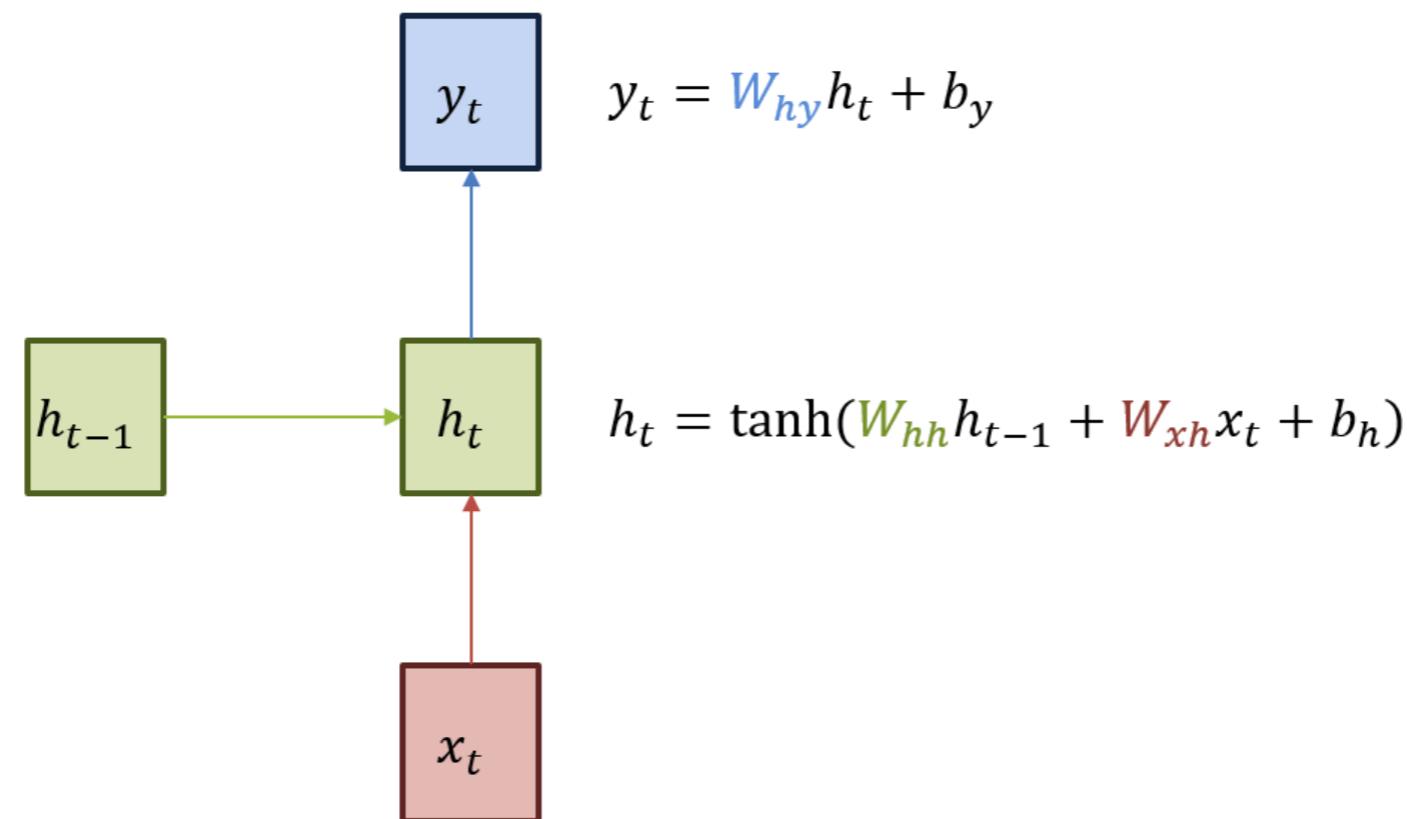
- 이미지 캡션 생성

컴퓨터 비전에서 활발하게 사용된 convolutional neural network(CNN)과 RNN을 함께 사용한다면, 임의의 이미지를 텍스트로 설명해주는 시스템을 만드는 것도 가능하다. 실제로 어떻게 왜 동작하는지는 상당히 신기하다. CNN과 RNN을 합친 모델은 이미지로부터 얻어낸 주요 단어들과 이미지의 각 부분을 매칭해줄 수도 있다.

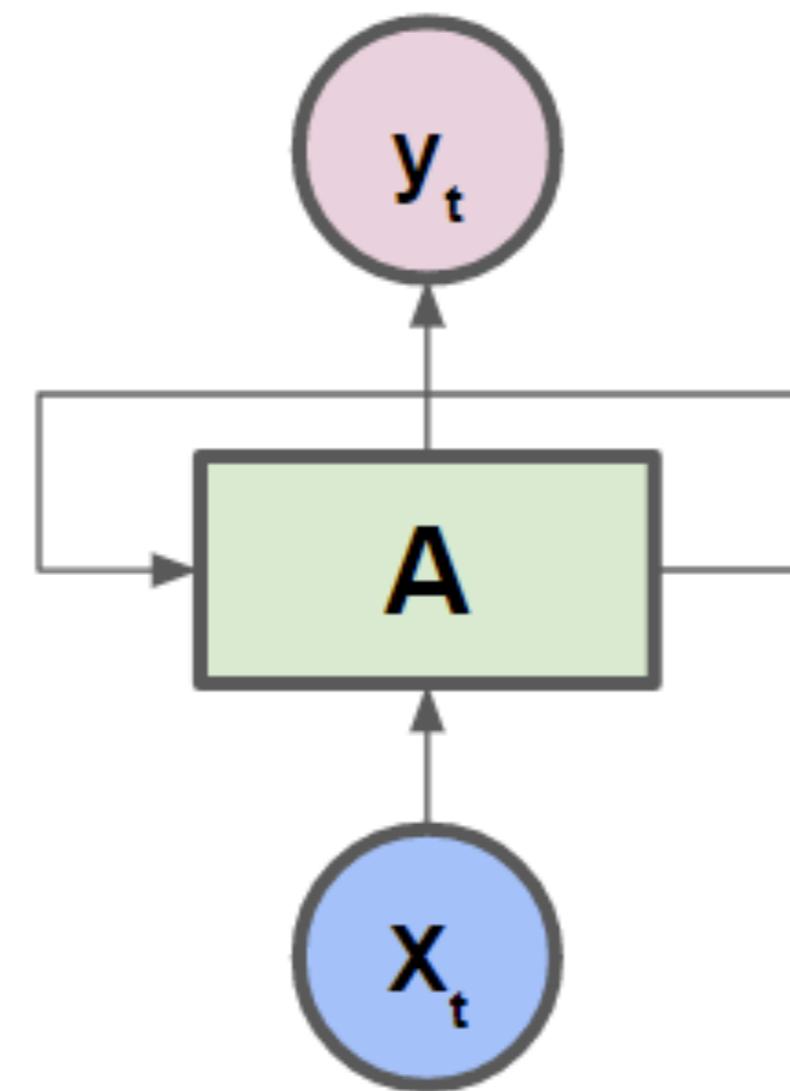


RNN 구조

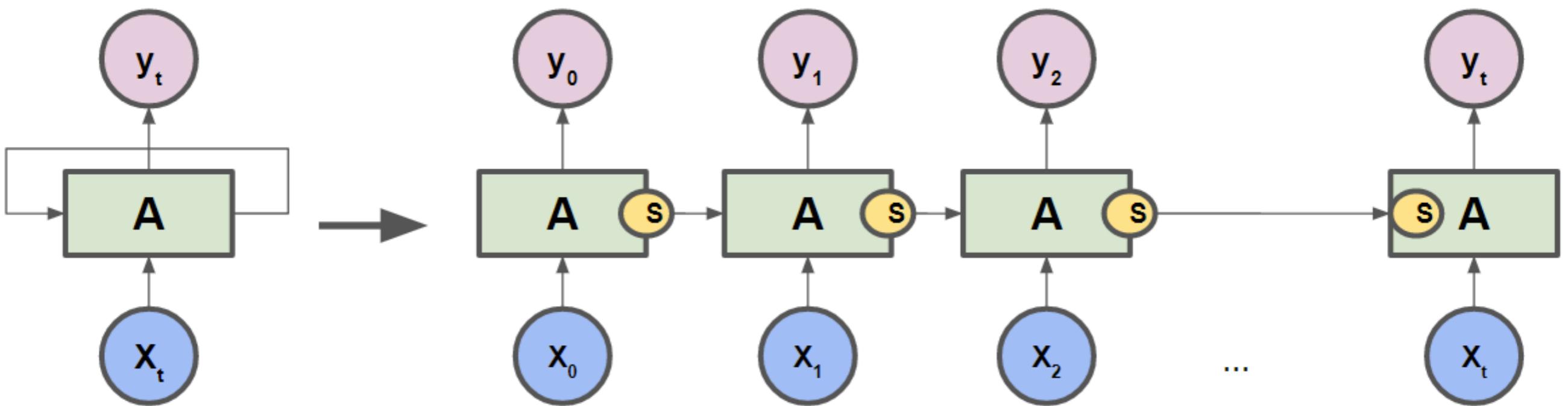
- 녹색 박스는 히든 state를 의미. 빨간 박스는 인풋 x, 파란 박스는 아웃풋 y.
현재 상태의 히든 state h_t 는 직전 시점의 히든 state h_{t-1} 를 받아 갱신됩니다.
- 현재 상태의 아웃풋 y_t 는 h_t 를 전달받아 갱신되는 구조입니다. 히든 state의 활성함수 (activation function)는 비선형 함수인 하이퍼볼릭탄젠트(tanh)입니다.



RNN CELL



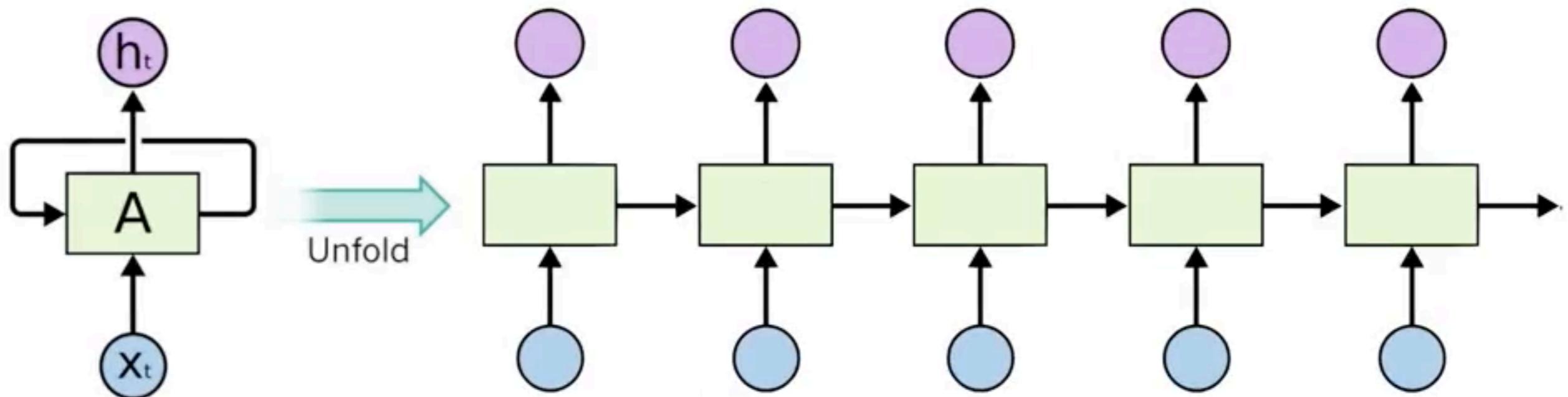
전개 : STATE



전개 : batch_size

hidden_size, sequence_length = 2, 5

shape=(1,5,2): [[[x,x], [x,x], [x,x], [x,x], [x,x]]]

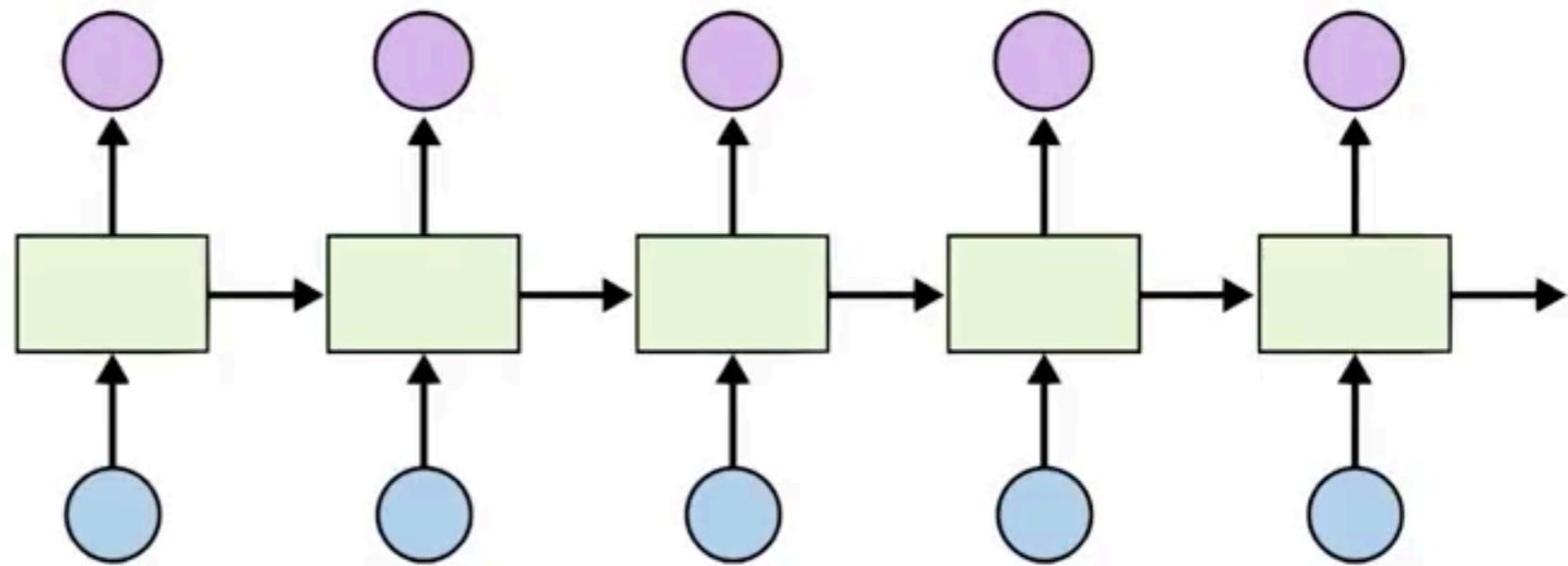


shape=(1,5,4): [[[1,0,0,0], [0,1,0,0], [0,0,1,0], [0,0,1,0], [0,0,0,1]]]
h e l l o

전개 : batch_size

batch_size, hidden_size, sequence_length = 3, 2, 5

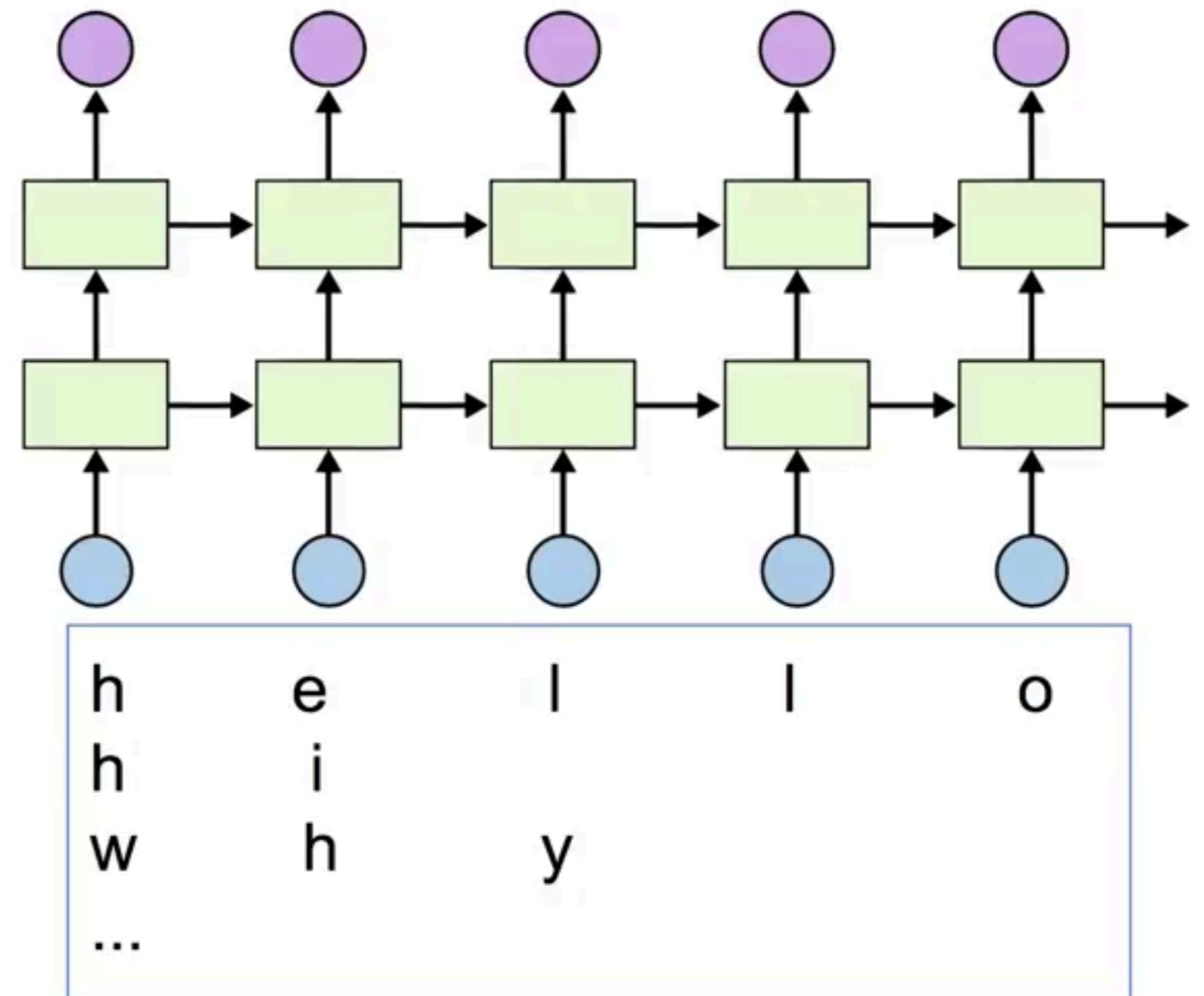
```
shape=(3,5,2): [[[x,x], [x,x], [x,x], [x,x], [x,x]],  
[[x,x], [x,x], [x,x], [x,x], [x,x]],  
[[x,x], [x,x], [x,x], [x,x], [x,x]]]
```



```
shape=(3,5,4): [[[1,0,0,0], [0,1,0,0], [0,0,1,0], [0,0,1,0], [0,0,0,1]], # hello  
[[0,1,0,0], [0,0,0,1], [0,0,1,0], [0,0,1,0], [0,0,1,0]] # eol11  
[[0,0,1,0], [0,0,1,0], [0,1,0,0], [0,1,0,0], [0,0,1,0]]] # 1leel
```

dynamic_rnn()

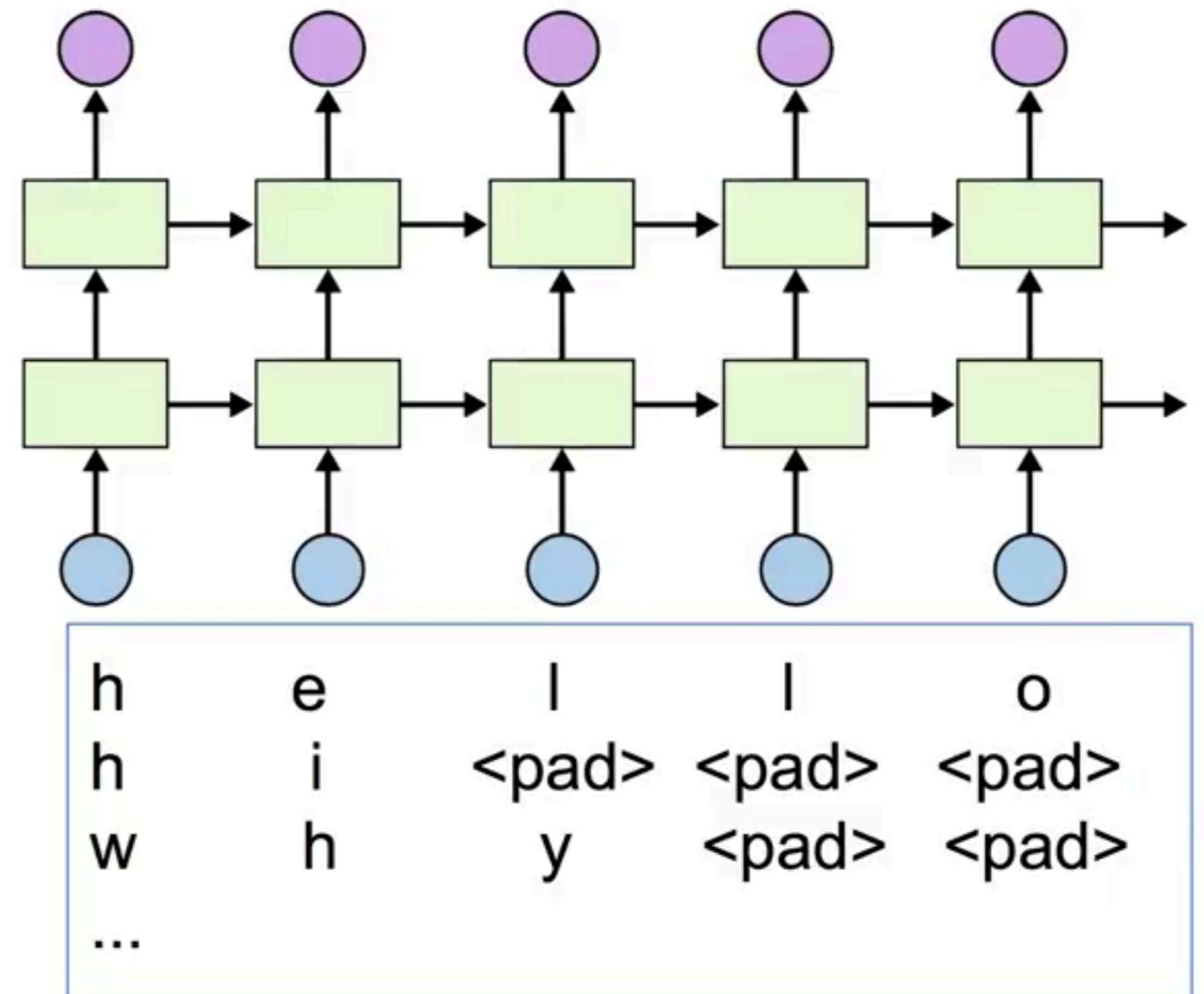
길이가 다른 문자열이
들어온다면?



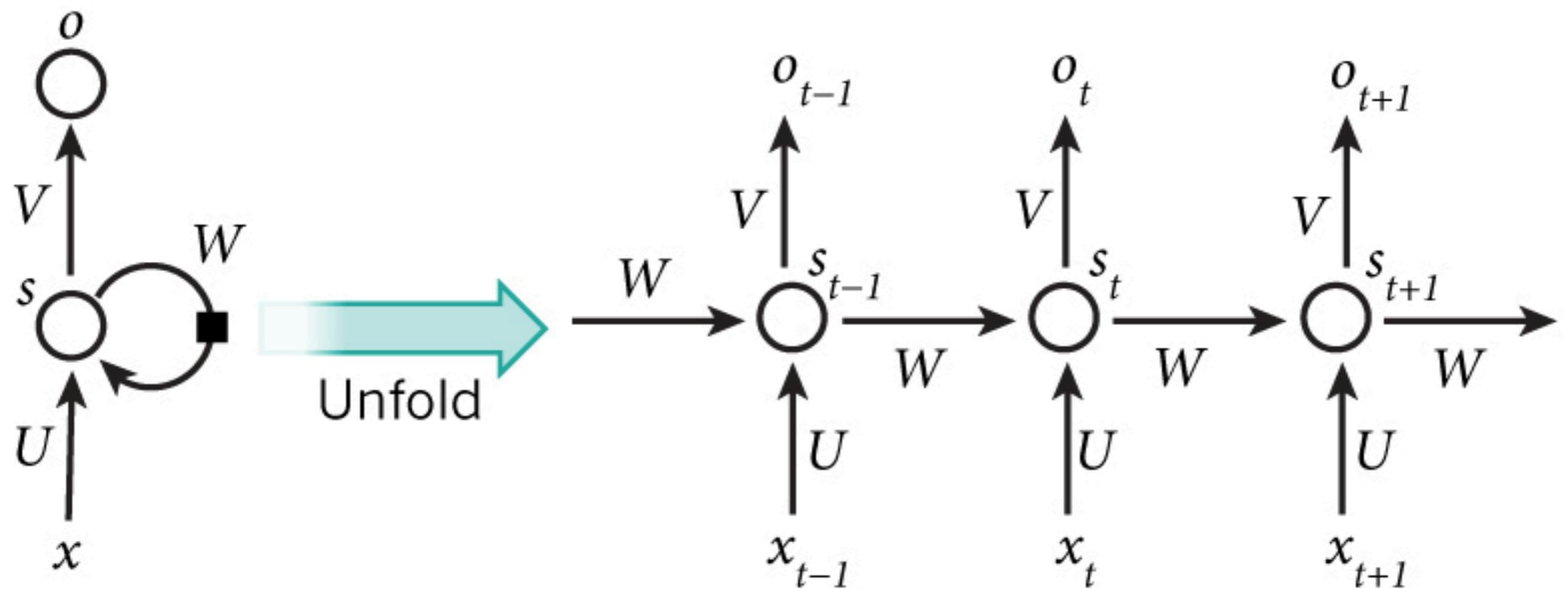
dynamic_rnn()

<pad>와 같은
식별자 추가

sequence_length
옵션으로 해결



UNFOLD



<http://aikorea.org/blog/rnn-tutorial-1>

UNFOLD

- x_t 는 시간 스텝(time step) t에서의 입력값이다.
- s_t 는 시간 스텝 t에서의 hidden state이다. 네트워크의 "메모리" 부분으로서, 이전 시간 스텝의 hiddent state 값과 현재 시간 스텝의 입력값에 의해 계산된다:
$$s_t = f(Ux_t + Ws_{t-1})$$
.
비선형 함수 f 는 보통 tanh나 ReLU가 사용되고,
첫 hidden state를 계산하기 위한 s_{-1} 은 보통 0으로 초기화시킨다.
- o_t 는 시간 스텝 t에서의 출력값이다. 예를 들어,
문장에서 다음 단어를 추측하고 싶다면 단어 수만큼의 차원의 확률 벡터가 될 것이다.
$$o_t = \text{softmax}(Vs_t)$$

UNFOLD

- Hidden state s_t 는 네트워크의 메모리라고 생각할 수 있다. s_t 는 과거의 시간 스텝들에서 일어난 일들에 대한 정보를 전부 담고 있고, 출력값 o_t 는 오로지 현재 시간 스텝 t 의 메모리에만 의존한다. 하지만 위에서 잠깐 언급했듯이, 실제 구현에서는 너무 먼 과거에 일어난 일들은 잘 기억하지 못한다.
- 각 layer마다의 파라미터 값들이 전부 다 다른 기존의 deep한 신경망 구조와 달리, RNN은 모든 시간 스텝에 대해 파라미터 값을 전부 공유하고 있다 (위 그림의 U, V, W). 이는 RNN이 각 스텝마다 입력값만 다를 뿐 거의 똑같은 계산을 하고 있다는 것을 보여준다. 이는 학습해야 하는 파라미터 수를 많이 줄여준다.
- 위 다이어그램에서는 매 시간 스텝마다 출력값을 내지만, 문제에 따라 달라질 수도 있다. 예를 들어, 문장에서 긍정/부정적인 감정을 추측하고 싶다면 굳이 모든 단어 위치에 대해 추측값을 내지 않고 최종 추측값 하나만 내서 판단하는 것이 더 유용할 수도 있다. 마찬가지로, 입력값 역시 매 시간 스텝마다 꼭 다 필요한 것은 아니다. RNN에서의 핵심은 시퀀스 정보에 대해 어떠한 정보를 추출해 주는 hidden state이기 때문이다.

학습

- RNN 네트워크를 학습하는 것은 기존의 신경망 모델 학습과 매우 유사하다.
- 네트워크의 각 시간 스텝마다 파라미터들이 공유되기 때문에 펼쳐진 네트워크에서 기존의 backpropagation 알고리즘을 그대로 사용하진 못한다.
- Backpropagation Through Time (BPTT)라는 약간 변형된 알고리즘을 사용한다. 그 이유는, 각 출력 부분에서의 gradient가 현재 시간 스텝에만 의존하지 않고 이전 시간 스텝들에도 의존하기 때문이다. 즉, $t=4$ 에서의 gradient를 계산하기 위해서는 시간 스텝 3 개 이전부터 gradient를 전부 더해주어야 한다.
- vanishing/exploding gradient라는 문제 등에 의해서 단순한 RNN을 BPTT로 학습시키는 것은 긴 시퀀스를 다루기 어렵다. 이를 해결하기 위한 여러 트릭들이 존재하고, LSTM 등이 문제를 해결하기 위한 다양한 변종(확장된) RNN 모델들도 존재한다.

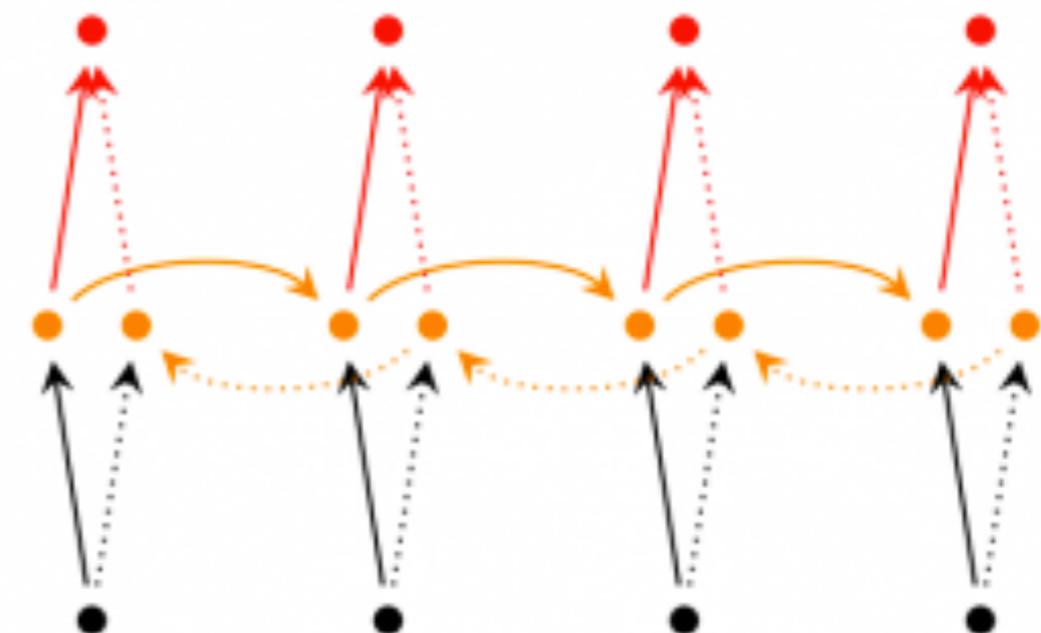
확장 모델

- Bidirectional RNN

시간 스텝 t 에서의 출력값이 이전 시간 스텝 외에, 이후의 시간 스텝에서 들어오는 입력값에도 영향을 받을 수 있다는 아이디어에 기반한다.

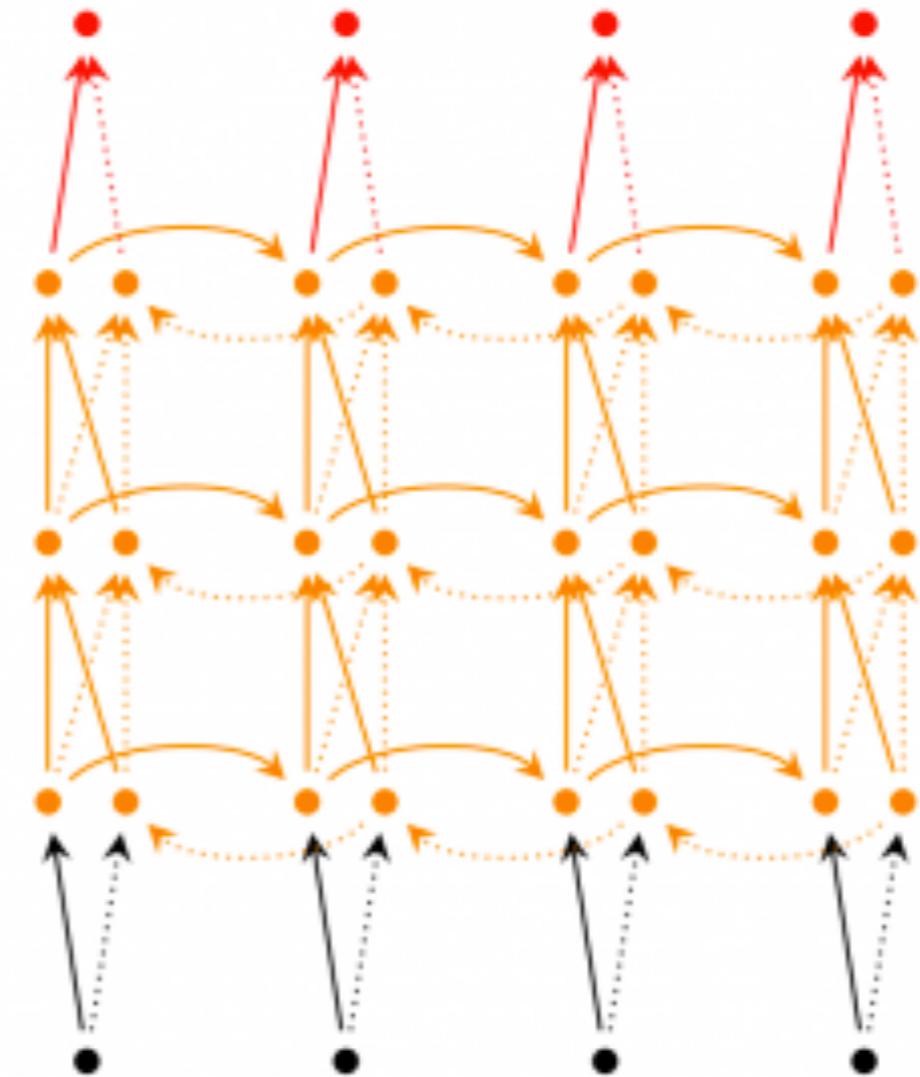
예를 들어, 영어 문제에서 빈칸에 가장 알맞는 단어를 채우기 위해서는 빈칸보다 앞쪽 문장들도 봐야겠지만, 빈칸 이후의 단어들도 문맥을 파악하는데 도움이 될 것이기 때문이다.

네트워크 구조는 RNN에서 단순히 확장되어서, 그림처럼 두 개의 RNN이 동시에 존재하고, 출력값은 두 RNN의 hidden state에 모두 의존하도록 계산된다.



확장 모델

- Deep (Bidirectional) RNN
앞의 구조와 비슷하지만, 매 시간 스텝
마다 여러 layer가 있다.
실제 구현에서는 이러한 구조가 학습
할 수 있는 capacity가 크다.
(당연히, 학습 데이터는 훨씬 더 많이
필요하다).



확장 모델

- LSTM

RNN에 비해 본질적으로 다른 구조를 갖고 있다고 하긴 힘들지만, hidden state를 계산하는 데 다른 식을 사용한다.

LSTM에서는 RNN의 뉴런 대신에 메모리 셀이라고 불리는 구조를 사용하는데, 입력값으로 이전 state h_{t-1} 와 현재 입력값 x_t 를 입력으로 받는 블랙박스 형태로 생각하면 된다.

메모리 셀 내부에서는 이전 메모리 값을 그대로 남길지 지울지 정하고, 현재 state와 메모리 셀의 입력값을 토대로 현재 메모리에 저장할 값을 계산한다.

이러한 구조는 긴 시퀀스를 기억하는데 매우 효과적이다.

언어 모델링

- m 개의 단어로 이루어져 있는 문장이 있다고 하면,
언어 모델에서 이 문장이 특정 데이터셋에서 나타날 확률은 다음과 같다.
- 수식
$$P(w_1, \dots, w_m) = \prod_{i=1}^m P(w_i | w_1, \dots, w_{i-1})$$
- 문장이 나타날 확률은 이전 단어를 알고 있을 때 각 단어가 나타날 확률의 곱이 된다. 따라서 "He went to buy some chocolate"라는 문장의 확률은 "He went to buy some"이 주어졌을 때 "chocolate"의 확률 곱하기 "He went to buy"가 주어졌을 때 "some"의 확률 곱하기 ... 문장의 시작에서 아무것도 안 주어졌을 때 "He"의 확률까지의 곱이 된다.
- 앞의 수식에서 각 단어들의 확률은 이전에 나왔던 모든 단어들에 의존하고 있다. 하지만 실제 구현에서는 많은 모델들이 계산량, 메모리 문제 등으로 인해 long-term dependency 를 효과적으로 다루지 못해서 긴 시퀀스는 처리하는 것이 힘들다. 이론적으로 RNN 모델은 임의의 길이의 시퀀스를 전부 기억할 수 있지만 실제로는 조금 더 복잡하다.

언어 모델 필요성

- 점수를 매기는 메커니즘으로 활용될 수 있다. 예를 들어, 자동 기계 번역 시스템은 보통 하나의 입력 문장에 대해 여러 개의 후보 답안 문장을 생성한다. 여기서 언어 모델로 가장 확률이 높은 문장을 고를 수 있을 것이다. 직관적으로 보면, 가장 확률이 높은 문장은 문법적으로도 더 맞을 확률이 높다. 음성 인식 시스템에서도 비슷한 방식으로 점수를 매기는데 활용된다.
- 언어 모델 문제를 풀다보면 상당히 재미있는 부산물이 나타난다. 문장에서 이전 위치에 나타나는 단어들을 알 때 다음 단어가 나타날 확률을 얻을 수 있기 때문에, 이를 기반으로 새로운 텍스트를 생성해낼 수도 있는 것이다.
즉, 생성 모델 (generative model)이 나타난다. 현재 갖고 있는 단어들의 시퀀스를 주고 결과로 얻은 단어들의 확률 분포에서 다음 단어를 샘플링하고, 문장이 완성될 때까지 계속 이 과정을 반복할 수 있다.
- Andrej Karpathy가 블로그 포스트에 언어 모델이 어떤 일들을 할 수 있는지에 대해 훌륭하게 정리해 주었다. Karpathy의 모델은 단어 기준이 아니라 글자(character) 단위로 학습되었고, 셰익스피어부터 리눅스 소스 코드까지 전부 다 생성해낼 수 있다.

데이터 전처리

1. 텍스트의 토큰화 (Tokenize Text)

텍스트 데이터에서 단어 단위로 예측을 하기 위해서는 댓글을 문장으로 토큰화 하고, 문장을 단어 단위로 쪼개야 한다. 단순히 공백(스페이스바)을 기준으로 자를 수도 있겠지만, 이는 문장 부호들을 제대로 처리하지 못하게 된다. 예시로, "He left!"라는 문장은 3개의 토큰 - "He", "left", "!" - 으로 이루어져야 한다. 여기서는 NLTK의 `word_tokenize`와 `sent_tokenize` 방식을 사용하였다.

데이터 전처리

2. 빈도수가 낮은 단어들 없애기

데이터셋에 있는 대부분의 단어들은 한 번 내지 두 번 정도 등장한다. 이렇게 드문드문 나타나는 단어들은 없애는 것이 더 도움이 된다. 기억해야 할 단어의 종류가 너무 커지면 모델을 학습하는데 시간이 더 오래 걸리고, 빈도수가 낮은 단어들의 경우에는 어떤 상황에서 이런 단어들이 나타나는지에 대한 예시가 별로 없어서 학습하기도 힘들다. 사람이 배우는 것과도 비슷한데, 어떤 단어의 의미를 제대로 파악하려면 여러 상황에서 활용된 예시문을 봐야 할 것이다. 텍스트에 등장하는 빈도순으로 `vocabulary_size` 변수만큼으로 단어 수를 제한한다. 단어장에 없는 단어들은 전부 `UNKNOWN_TOKEN`으로 바꿔준다. 예시로, 단어장에 "`nonlinearities`"라는 단어가 없다면, "`nonlinearities are important in neural networks`"라는 문장은 "`UNKNOWN_TOKEN are important in neural networks`"로 바뀔 것이다. `UNKNOWN_TOKEN`이라는 단어는 단어장에 추가되고, 다른 단어처럼 나타날 확률 예측도 하게 된다. 새로운 텍스트를 생성할 때는 `UNKNOWN_TOKEN`을 단어장에 없는 단어 중에서 아무거나 랜덤으로 뽑아서 대체할 수도 있고, 아니면 `UNKNOWN_TOKEN`이 나오기 전까지만 문장을 생성하는 방법도 있다.

데이터 전처리

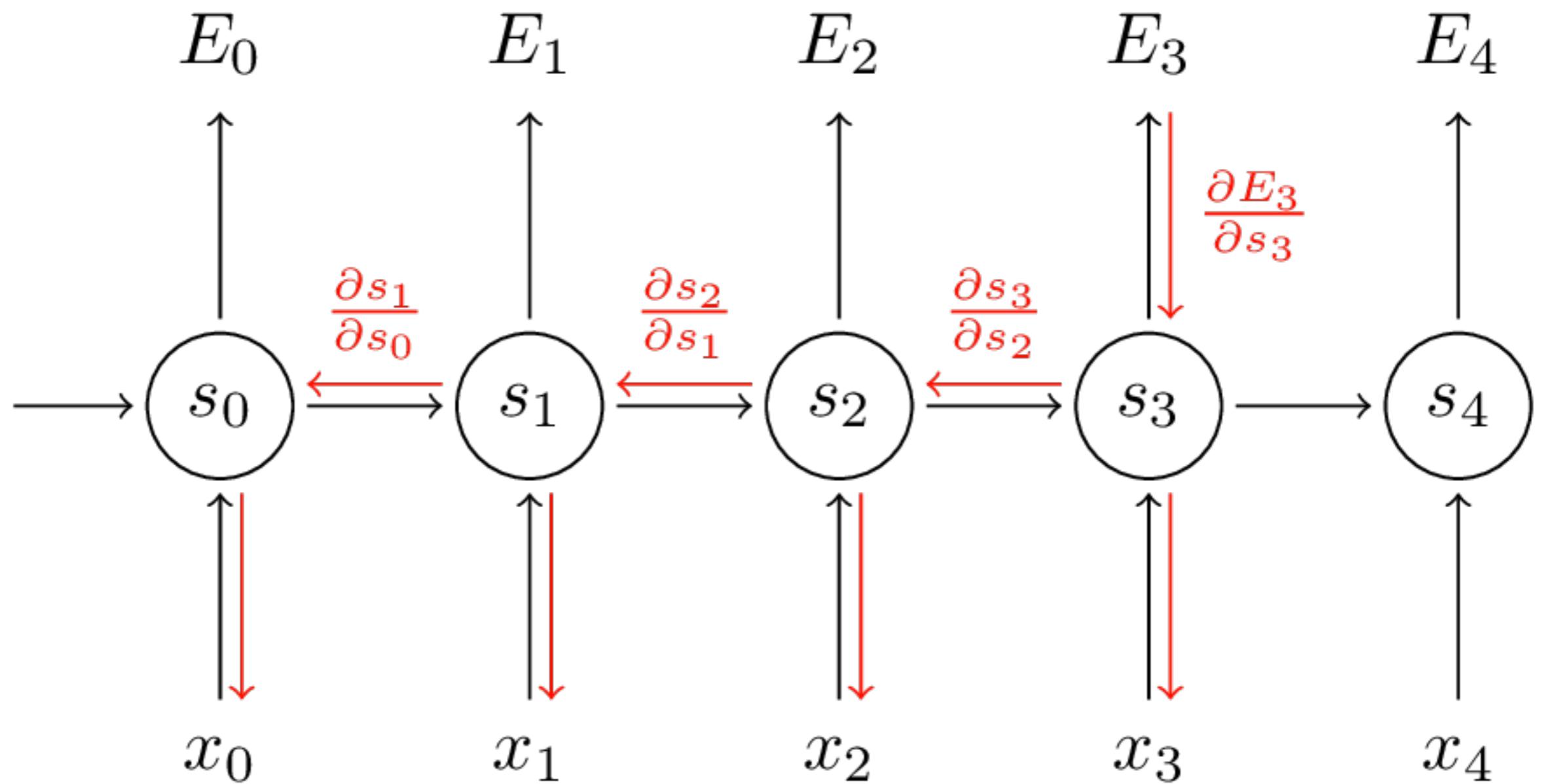
3. 시작 토큰과 끝 토큰 연결

언어 모델은 어떤 단어들이 문장의 맨 처음 나타나고, 어떤 것들이 맨 뒤에 나타나는지도 학습하고자 한다. 이를 위해서 특별히 SENTENCE_START 토큰을 문장의 맨 앞에 이어붙이고, SENTENCE_END 토큰을 문장의 맨 뒤에 붙일 것이다. 모든 문장에 대해 이 과정을 처리해 주고나면, 문제는 다음과 같이 바뀐다: 첫 번째 토큰이 SENTENCE_START일 때, 다음 단어는 무엇일까? (실제 문장의 첫 단어)

4. 학습 데이터 행렬 구성

RNN 모델의 입력은 문자열이 아니라 벡터이기 때문에 단어들과 인덱스들 사이의 매핑 - index_to_word와 word_to_index를 먼저 만든다. 예를 들어, "friendly"라는 단어는 2001번 위치에 있을 수 있다. 학습 데이터 x 는 [0, 179, 341, 416]과 같이 생겼을 것이고, 여기서 0은 SENTENCE_START를 뜻한다. 해당하는 정답 y 는 [179, 341, 416, 1] 정도로 나타내질 것이다. 우리 목적은 다음 단어를 예측하는 것이기 때문에 y 는 단순히 x 벡터를 오른쪽으로 한 칸 옮기고 마지막 위치에 SENTENCE_END 토큰을 넣어준 것이어야 한다. 즉, 179 번 단어의 올바른 예측값은 실제 다음 단어인 341이 되어야 한다.

BPTT



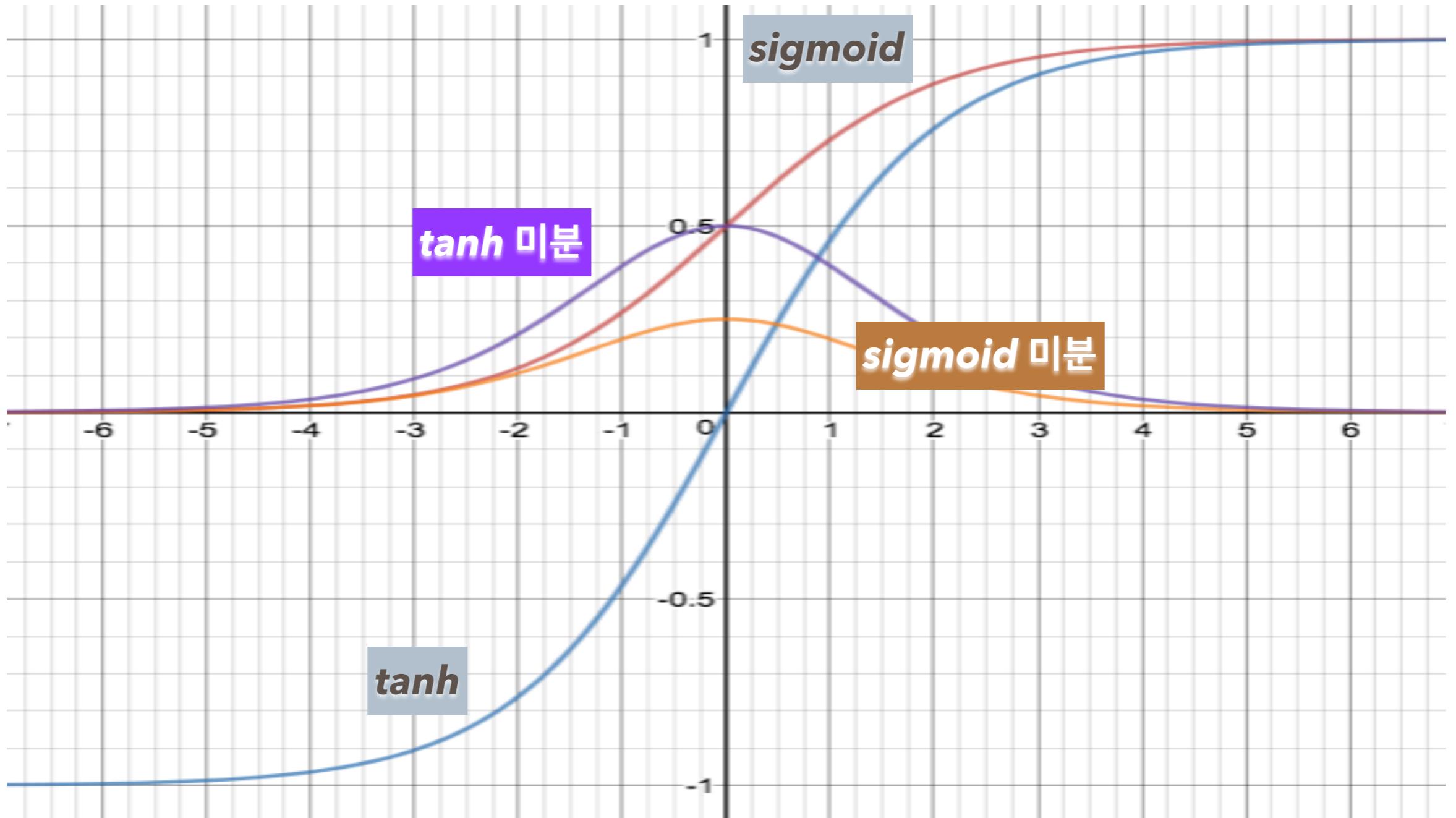
BPTT

- Backpropagation Through Time
- deep Feedforward Neural Network(피드백 연결이 없는 네트워크)에서 사용하는 원래의 backpropagation 알고리즘과 똑같다.
- 중요한 차이점은 매 시간 스텝마다 W 에 대한 gradient를 더해준다는 점이다. 기존의 신경망 구조에서는 layer별로 파라미터를 공유하지 않기 때문에 계산 결과들을 서로 더해줄 필요가 없다.

VANISHING GRADIENT

- RNN은 긴 시퀀스를 처리하는데 (long-range dependency를 처리하는데) 한계가 있다. 즉, 주요 단어들 사이에 여러 시간 스텝이 지났다면 잘 기억하지 못한다.
- 이것은 문장의 의미를 파악하는데 있어서 가까이 있지 않은 단어들이 밀접한 관련이 있을 수도 있기 때문에 문제가 된다.
- 예로, "The man who wore a wig on his head went inside."라는 문장을 보면, 이 문장은 "man"이 "inside"로 가는 것에 대한 문장이지 "wig(가발)"에 대한 것이 아니다. 그러나, 기본 RNN 모델은 남자보다 가발에 대한 정보를 더 잘 기억하게 된다.

Vanishing Gradient



VANISHING GRADIENT

- tanh 함수와 sigmoid 함수는 양쪽 끝에서 미분값이 0으로 수렴하는 것을 볼 수 있다.
- 이 현상이 발생할 때, 그 뉴런이 포화되었다고 말하는데, 이런 뉴런들은 gradient가 거의 0이기 때문에 곱해지는 이전 layer의 gradient들도 0으로 수렴하게 만든다.
- 따라서, 행렬에 작은 값들이 들어있고 여러 ($t-k$ 번) 행렬곱이 이루어지면 gradient는 지수 함수로 감소하고, 시간 스텝 몇 번만 지나도 사라져 버린다 (vanish!).
- 시퀀스에서 여러 시간 스텝이 떨어진 곳에서는 gradient가 전달되지 못하고, 먼 과거의 상태(state)는 현재 스텝의 학습에 아무 도움이 되지 못하게 된다. 즉, long-range dependency를 제대로 배우지 못한다.
- Vanishing gradient 문제는 RNN에서만 나타나는 것이 아니다. Deep Feedforward Neural Network에서도 마찬가지로 발생하지만, RNN은 보통 시간 스텝 횟수만큼 매우 깊은 구조이기 때문에 이 문제가 훨씬 잘 나타난다.

VANISHING GRADIENT

- 자코비안 행렬을 통한 Gradient 계산을 보면, 행렬 안의 값들이 크다면 activation 함수와 네트워크 파라미터 값에 따라 gradient가 사라지는게 아니라 지수 함수로 증가하는 경우도 충분히 상상해볼 수 있다
- 이 문제는 exploding gradient 문제로 알려져 있다.
- Vanishing gradient 문제가 더 많은 관심을 받는 이유
 - exploding gradient 문제는 쉽게 알아차릴 수 있다는 점이다. Gradient 값들이 NaN (not a number)이 될 것이고 프로그램이 죽을 것이기 때문이다.
 - gradient 값이 너무 크다면 미리 정해준 적당한 값으로 잘라버리는 방법으로 쉬우면서도 효율적으로 해결할 수 있기 때문이다.
 - Vanishing gradient 문제는 언제 발생하는지 바로 확인하기가 힘들고 간단한 해결법이 없기 때문에 문제였다.

VANISHING GRADIENT 해결책

- W 행렬을 적당히 좋은 값으로 잘 초기화 해준다면 vanishing gradient의 영향을 줄일 수 있고, regularization을 잘 정해줘도 비슷한 효과를 볼 수 있다.
- 더 보편적으로 사용되는 방법은 tanh나 sigmoid activation 함수 말고 ReLU를 사용하는 것이다. ReLU는 미분값의 최대치가 1로 정해져있지 않기 때문에 gradient 값이 없어져버리는 일이 크게 줄어든다.
- 이보다 더 인기있는 해결책은 Long Short-Term Memory (LSTM)이나 Gated Recurrent Unit (GRU) 구조를 사용하는 방법이다.
 - LSTM은 1997년에 처음 제안되었고, 현재 자연어처리 분야에서 가장 널리 사용되는 모델 중 하나이다. GRU는 2014년에 처음 나왔고, LSTM을 간략화한 버전이다.
 - 두 가지 RNN의 변형 구조 모두 vanishing gradient 문제 해결을 위해 디자인되었고, 효과적으로 긴 시퀀스를 처리할 수 있다는 것이 보여졌다.

LSTM

- Long Short Term Memory
- 1997년에 스위스의 Sepp Hochreiter와 Jürgen Schmidhuber에 의해 처음 제안
- 자연어처리 분야에 활용되는 딥러닝 기법 중 가장 널리 사용되고 있는 모델 중의 하나

$$i = \sigma(x_t U^i + s_{t-1} W^i)$$

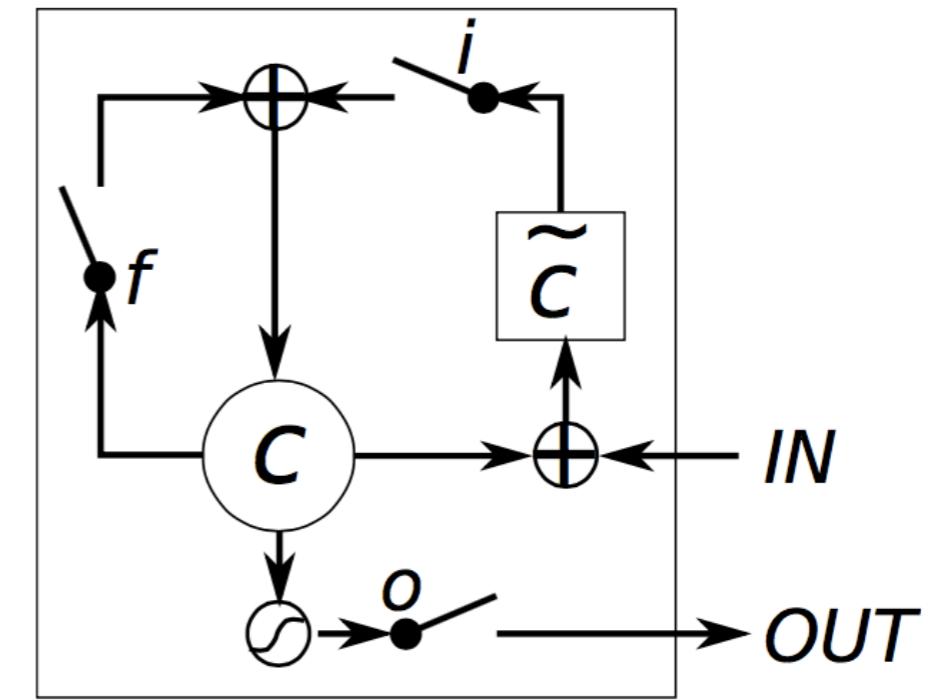
$$f = \sigma(x_t U^f + s_{t-1} W^f)$$

$$o = \sigma(x_t U^o + s_{t-1} W^o)$$

$$g = \tanh(x_t U^g + s_{t-1} W^g)$$

$$c_t = c_{t-1} \circ f + g \circ i$$

$$s_t = \tanh(c_t) \circ o$$



LSTM

- i, f, o 는 각각 input, forget, output 게이트라고 부릅니다.
- 각 게이트의 수식은 동일한 형태를 띠고 있고, 파라미터 행렬만 다릅니다.
- 이들이 게이트라고 부르는 이유는, sigmoid 함수가 이 벡터들의 값을 0에서 1 사이로 제한시키고 이를 다른 벡터와 element-wise 곱을 취한다면 그 다른 벡터값의 얼마만큼을 통과시킬지를 정해 주는 것과 같기 때문입니다.
- input 게이트는 새 hidden state 값을 계산하는데 있어서 입력 벡터값을 얼마큼 사용할지를 정해줍니다.
- forget 게이트는 이전 state 값을 얼마큼 기억하고 있을지를 정해줍니다.
- output 게이트는 현재의 내부 state 값의 얼마큼을 LSTM 모듈의 바깥쪽에서 (더 깊은 레이어나 이후 시간 스텝에서) 볼 수 있을지를 정해줍니다.
- 모든 게이트들은 hidden state와 같은 차원을 갖게 됩니다.

LSTM

- g 는 현재 입력과 이전 hidden state의 값을 기반으로 계산된 현재 hidden state 값의 "후보"라고 할 수 있습니다.
- RNN 기본형 모델에서 본 것과 동일한 수식으로 계산되는데, 파라미터 행렬의 이름만 U 와 W 가 U_g 와 W_g 로 바뀌었습니다. 그러나 이전처럼 g 를 바로 새 hidden state로 정하는 대신, 여기서는 입력 게이트를 사용하여 일부만 사용합니다.
- c_t 는 LSTM 유닛(모듈)의 내부 메모리입니다. 이것은 이전에 저장된 메모리인 c_{t-1} 과 forget 게이트의 곱, 그리고 새로 계산된 hidden state g 와 input 게이트의 곱을 합친 형태로 계산됩니다. 따라서, 간단히 말하면 이전 메모리와 현재의 새 입력을 어떻게 합칠까에 대한 부분입니다. 이전의 메모리를 전부 무시하도록 (forget 게이트 값이 전부 0) 정하거나 새로운 입력값을 통째로 무시하도록 (input 게이트 값이 전부 0) 할 수도 있지만, 보통은 양극단보다는 중간의 좋은 지점을 찾도록 합니다.
- 메모리값 c_t 가 주어지면, 메모리와 output 게이트의 곱으로 최종적으로 출력 hidden state s_t 가 계산됩니다. 모든 내부 메모리 값이 네트워크의 다른 유닛들에서 활용할 필요는 없을 수도 있기 때문에 output 게이트를 통과시키는 것입니다.

LSTM

- 메모리값 c_t 가 주어지면, 메모리와 출력 게이트의 곱으로 최종적으로 출력 hidden state s_t 가 계산됩니다. 모든 내부 메모리 값이 네트워크의 다른 유닛들에서 활용할 필요는 없을 수도 있기 때문에 출력 게이트를 통과시키는 것입니다.

GRU

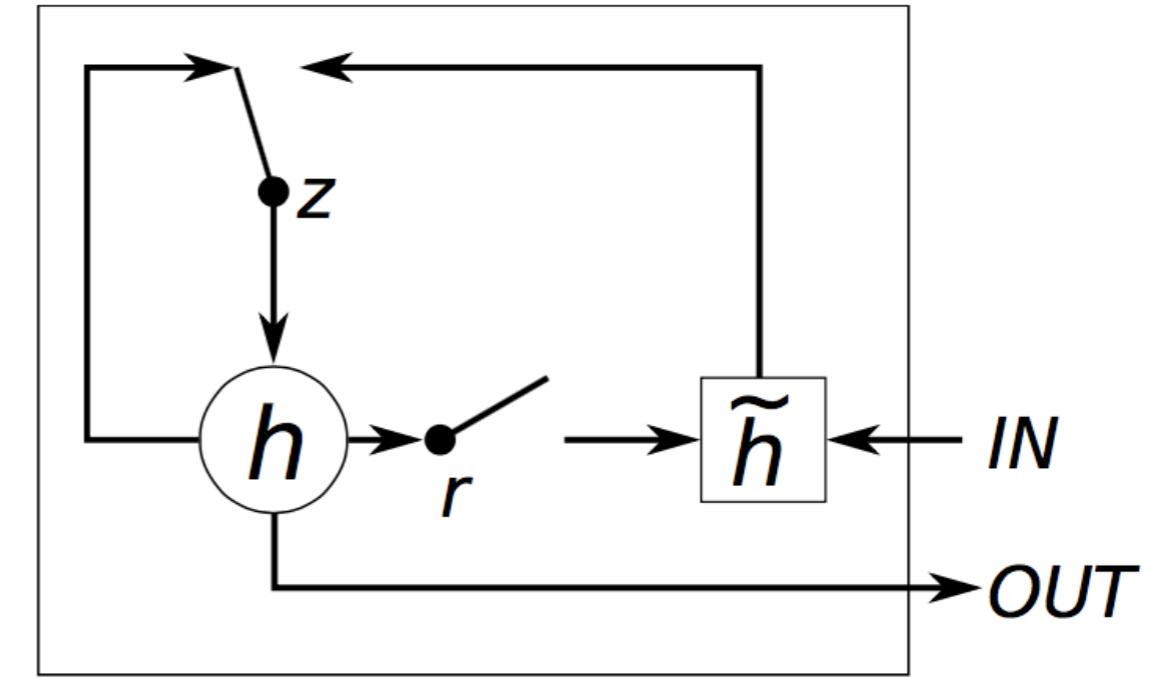
- Gated Recurrent Unit
- GRU는 2014년에 처음 사용되었는데 (뉴욕대의 조경현 교수님께서 처음 제안)
- 대부분 LSTM과 비슷한 성질을 갖지만 더 간단한 구조

$$z = \sigma(x_t U^z + s_{t-1} W^z)$$

$$r = \sigma(x_t U^r + s_{t-1} W^r)$$

$$h = \tanh(x_t U^h + (s_{t-1} \circ r) W^h)$$

$$s_t = (1 - z) \circ h + z \circ s_{t-1}$$



GRU

- GRU는 리셋 게이트 r 과 업데이트 게이트 z 로, 총 두 가지 게이트가 있습니다.
- reset 게이트는 새로운 입력을 이전 메모리와 어떻게 합칠지를 정해줍니다.
- update 게이트는 이전 메모리를 얼마나 기억할지를 정해줍니다.
- reset 게이트 값을 전부 1로 정해주고 update 게이트를 전부 0으로 정한다면, 기본 RNN 구조가 됩니다.

LSTM VS. GRU

- 게이팅 메커니즘을 통해 긴 시퀀스를 잘 기억하도록 해준다는 점에서 비슷하다.
- GRU는 게이트가 2개이고, LSTM은 3개입니다.
- GRU는 내부 메모리 값 (ct)이 외부에서 보게되는 hidden state 값과 다르지 않습니다. LSTM에 있는 output 게이트가 없기 때문입니다.
- input 게이트와 forget 게이트가 update 게이트 z 로 합쳐졌고, reset 게이트 r 은 이전 hidden state 값에 바로 적용됩니다. 따라서, LSTM의 forget 게이트의 역할이 r 과 z 양쪽으로 나눠졌다고 생각할 수 있습니다.
- 출력값을 계산할 때 추가적인 비선형 함수를 적용하지 않습니다.

LSTM VS. GRU

- 여러 논문에서의 실험들에 따르면,
두 모델 모두 좋은 성능을 보여주고 있어서 어떤 것이 좋다라고 얘기할 수는 없습니다.
- 레이어 크기같은 파라미터 튜닝을 잘 하는 것이 모델을 고르는 것보다 더 중요합니다.
- GRU는 파라미터 수가 적어서 (U 와 W 가 더 작다)
학습 시간이 짧고 적은 데이터로도 학습이 가능할 수 있습니다.
- 데이터가 충분한 경우에는
LSTM의 우수한 모델링 파워가 더 좋은 결과를 보여줄 수도 있을 것입니다.

성능 향상

- Rmsprop나 Adam 등의 최적화 알고리즘 사용

- 임베딩 레이어 추가

Word2vec이나 GloVe와 같은 단어 임베딩을 추가하는 것은 모델의 성능을 향상시키기 위해 자주 사용되는 방법입니다. 각 단어의 one-hot 벡터 표현법과 달리 word2vec이나 GloVe에서 학습된 낮은 차원의 벡터 표현은 그 단어의 의미 정보를 담게 됩니다. 즉, 비슷한 단어는 비슷한 벡터 값을 갖게 됩니다. 이것들을 사용하는 것은 pre-training (딥러닝에서 한번에 학습이 어렵기 때문에 단계별로 네트워크의 일부분을 미리 학습해 두는 작업)처럼 생각할 수 있습니다. 이 과정을 통해 네트워크가 언어에 대한 정보를 (미리 어느정도는 학습되어 있어서) 학습해야 될 부분이 줄어들게 되는 것입니다. Pre-train된 벡터들은 학습할 데이터가 많지 않을 때 특히 유용한데, 네트워크가 사전에 보지 못한 단어들에 대해서도 일반화가 가능해지기 때문입니다.

- 두 번째 GRU 레이어 추가하기

두 번째 레이어를 추가하는 것은 고차원적인 정보를 담을 수 있게 해줍니다. 그러나, 2-3 레이어 이후부터는 성능이 더 안 좋아지는 것을 확인할 가능성이 큰데, 데이터가 많지 않은 이상 레이어가 많아진다고 해서 성능에 큰 차이가 없을 것이고, 오히려 과적합될 수 있습니다.

성능 향상

- 파라미터를 업데이트할 때 배치(batch)로 gradient를 합치는 것입니다.
- 한 번에 한 문장씩 학습을 하기보다는, 같은 길이의 문장들을 (또는 같은 길이가 되도록 zero-padding을 해서) 그룹으로 묶고, 큰 행렬 곱연산을 수행하여 그 배치에 대한 gradient들을 전부 더해줍니다. 이것이 효율적인 이유는, 큰 행렬들의 곱연산은 GPU를 통해 효율적으로 처리될 수 있기 때문인데, 이 방식을 취하지 않는다면 GPU를 사용하는 이점이 별로 없어져서 학습이 매우 느려질 것입니다.
- 큰 모델을 학습시키려면 성능에 대해 최적화가 잘 되어있는 현존하는 딥러닝 라이브러리 중 하나를 사용하는 것을 추천합니다.



RNN

TENSORFLOW API

BasicRNNCell

- `__init__(num_units, activation=None, reuse=None, name=None)`
- The most basic RNN cell.
- Args:
 - `num_units`: int, The number of units in the RNN cell.
 - `activation`: Nonlinearity to use. Default: `tanh`.
 - `reuse`: (optional) Python boolean describing whether to reuse variables in an existing scope. If not `True`, and the existing scope already has the given variables, an error is raised.
 - `name`: String, the name of the layer. Layers with the same name will share weights, but to avoid mistakes we require `reuse=True` in such cases.

BasicLSTMCell

- `__init__(num_units, forget_bias=1.0, state_is_tuple=True, activation=None, reuse=None, name=None)`
- Basic LSTM recurrent network cell.
The implementation is based on: <http://arxiv.org/abs/1409.2329>.
We add `forget_bias` (default: 1) to the biases of the forget gate in
order to reduce the scale of forgetting in the beginning of the
training.
It does not allow cell clipping, a projection layer, and does not use
peep-hole connections: it is the basic baseline.
- When restoring from CudnnLSTM-trained checkpoints, must use
`CudnnCompatibleLSTMCell` instead.

BasicLSTMCell

- Args:
 - num_units: int, The number of units in the LSTM cell.
 - forget_bias: float, The bias added to forget gates (see above). Must set to 0.0 manually when restoring from CudnnLSTM-trained checkpoints.
 - state_is_tuple: If True, accepted and returned states are 2-tuples of the c_state and m_state. If False, they are concatenated along the column axis. The latter behavior will soon be deprecated.
 - activation: Activation function of the inner states. Default: tanh.
 - reuse: (optional) Python boolean describing whether to reuse variables in an existing scope. If not True, and the existing scope already has the given variables, an error is raised.
 - name: String, the name of the layer. Layers with the same name will share weights, but to avoid mistakes we require reuse=True in such cases.

GRUCell

- `__init__(num_units, activation=None, reuse=None, kernel_initializer=None, bias_initializer=None, name=None)`
- Gated Recurrent Unit cell (cf. <http://arxiv.org/abs/1406.1078>).
- Args:
 - `num_units`: int, The number of units in the GRU cell.
 - `activation`: Nonlinearity to use. Default: `tanh`.
 - `reuse`: (optional) Python boolean describing whether to reuse variables in an existing scope. If not `True`, and the existing scope already has the given variables, an error is raised.
 - `kernel_initializer`: (optional) The initializer to use for the weight and projection matrices.
 - `bias_initializer`: (optional) The initializer to use for the bias.
 - `name`: String, the name of the layer. Layers with the same name will share weights, but to avoid mistakes we require `reuse=True` in such cases.

LSTMCell

- `__init__(num_units, use_peepholes=False, cell_clip=None, initializer=None, num_proj=None, proj_clip=None, num_unit_shards=None, num_proj_shards=None, forget_bias=1.0, state_is_tuple=True, activation=None, reuse=None, name=None)`
- Long short-term memory unit (LSTM) recurrent network cell.
- The default non-peephole implementation is based on:
<http://www.bioinf.jku.at/publications/older/2604.pdf>
- S. Hochreiter and J. Schmidhuber.
"Long Short-Term Memory". Neural Computation, 9(8):1735-1780, 1997.
- The peephole implementation is based on:
<https://research.google.com/pubs/archive/43905.pdf>

LSTMCell

- Hasim Sak, Andrew Senior, and Francoise Beaufays.
"Long short-term memory recurrent neural network architectures for large scale acoustic modeling." INTERSPEECH, 2014.
- The class uses optional peep-hole connections, optional cell clipping, and an optional projection layer.
- When restoring from CudnnLSTM-trained checkpoints, use CudnnCompatibleLSTMCell instead.
- Args:
 - num_units: int, The number of units in the LSTM cell.
 - use_peepholes: bool, set True to enable diagonal/peephole connections.
 - cell_clip: (optional) A float value, if provided the cell state is clipped by this value prior to the cell output activation.

LSTMCell

- Args:
 - initializer: (optional) The initializer to use for the weight and projection matrices.
 - num_proj: (optional) int, The output dimensionality for the projection matrices. If None, no projection is performed.
 - proj_clip: (optional) A float value. If num_proj > 0 and proj_clip is provided, then the projected values are clipped elementwise to within [-proj_clip, proj_clip].
 - num_unit_shards: Deprecated, will be removed by Jan. 2017. Use a variable_scope partitioner instead.
 - num_proj_shards: Deprecated, will be removed by Jan. 2017. Use a variable_scope partitioner instead.
 - forget_bias: Biases of the forget gate are initialized by default to 1 in order to reduce the scale of forgetting at the beginning of the training. Must set it manually to 0.0 when restoring from CudnnLSTM trained checkpoints.

LSTMCell

- Args:
 - `state_is_tuple`: If True, accepted and returned states are 2-tuples of the `c_state` and `m_state`. If False, they are concatenated along the column axis. This latter behavior will soon be deprecated.
 - `activation`: Activation function of the inner states. Default: `tanh`.
 - `reuse`: (optional) Python boolean describing whether to reuse variables in an existing scope. If not True, and the existing scope already has the given variables, an error is raised.
 - `name`: String, the name of the layer. Layers with the same name will share weights, but to avoid mistakes we require `reuse=True` in such cases.

MultiRNNCell

- `__init__(cells, state_is_tuple=True)`
- RNN cell composed sequentially of multiple simple cells.
- Args:
 - `cells`: list of RNNCells that will be composed in this order.
 - `state_is_tuple`: If True, accepted and returned states are n-tuples, where n = `len(cells)`. If False, the states are all concatenated along the column axis. This latter behavior will soon be deprecated.
- Example:

```
num_units = [128, 64]
cells = [BasicLSTMCell(num_units=n) for n in num_units]
stacked_rnn_cell = MultiRNNCell(cells)
```

dynamic_rnn

- `tf.nn.dynamic_rnn(cell, inputs, sequence_length=None, initial_state=None, dtype=None, parallel_iterations=None, swap_memory=False, time_major=False, scope=None)`
- Creates a recurrent neural network specified by RNNCell cell.
Performs fully dynamic unrolling of inputs.

dynamic_rnn

```
# create a BasicRNNCell
```

```
rnn_cell = tf.nn.rnn_cell.BasicRNNCell(hidden_size)
```

```
# 'outputs' is a tensor of shape [batch_size, max_time, cell_state_size]
```

```
# defining initial state
```

```
initial_state = rnn_cell.zero_state(batch_size, dtype=tf.float32)
```

```
# 'state' is a tensor of shape [batch_size, cell_state_size]
```

```
outputs, state = tf.nn.dynamic_rnn(rnn_cell, input_data,
```

```
initial_state=initial_state, dtype=tf.float32)
```

dynamic_rnn

```
# create 2 LSTMCells
```

```
rnn_layers = [tf.nn.rnn_cell.LSTMCell(size) for size in [128, 256]]
```

```
# create a RNN cell composed sequentially of a number of RNNCells
```

```
multi_rnn_cell = tf.nn.rnn_cell.MultiRNNCell(rnn_layers)
```

```
# 'outputs' is a tensor of shape [batch_size, max_time, 256]
```

'state' is a N-tuple where N is the number of LSTMCells containing a

```
# tf.contrib.rnn.LSTMStateTuple for each cell
```

```
outputs, state = tf.nn.dynamic_rnn(cell=multi_rnn_cell,
```

`inputs=data, dtype=tf.float32)`

dynamic_rnn

- Args:
 - cell: An instance of RNNCell.
 - inputs: The RNN inputs. If `time_major == False` (default), this must be a Tensor of shape: `[batch_size, max_time, ...]`, or a nested tuple of such elements. If `time_major == True`, this must be a Tensor of shape: `[max_time, batch_size, ...]`, or a nested tuple of such elements. This may also be a (possibly nested) tuple of Tensors satisfying this property. The first two dimensions must match across all the inputs, but otherwise the ranks and other shape components may differ. In this case, input to cell at each time-step will replicate the structure of these tuples, except for the time dimension (from which the time is taken). The input to cell at each time step will be a Tensor or (possibly nested) tuple of Tensors each with dimensions `[batch_size, ...]`.
 - `sequence_length`: (optional) An `int32/int64` vector sized `[batch_size]`. Used to copy-through state and zero-out outputs when past a batch element's sequence length. So it's more for correctness than performance.

dynamic_rnn

- Args:
 - `initial_state`: (optional) An initial state for the RNN. If `cell.state_size` is an integer, this must be a Tensor of appropriate type and shape `[batch_size, cell.state_size]`. If `cell.state_size` is a tuple, this should be a tuple of tensors having shapes `[batch_size, s]` for `s` in `cell.state_size`.
 - `dtype`: (optional) The data type for the initial state and expected output. Required if `initial_state` is not provided or RNN state has a heterogeneous `dtype`.
 - `parallel_iterations`: (Default: 32). The number of iterations to run in parallel. Those operations which do not have any temporal dependency and can be run in parallel, will be. This parameter trades off time for space. Values $\gg 1$ use more memory but take less time, while smaller values use less memory but computations take longer.

dynamic_rnn

- Args:
 - swap_memory: Transparently swap the tensors produced in forward inference but needed for back prop from GPU to CPU. This allows training RNNs which would typically not fit on a single GPU, with very minimal (or no) performance penalty.
 - time_major: The shape format of the inputs and outputs Tensors. If true, these Tensors must be shaped [max_time, batch_size, depth]. If false, these Tensors must be shaped [batch_size, max_time, depth]. Using time_major = True is a bit more efficient because it avoids transposes at the beginning and end of the RNN calculation. However, most TensorFlow data is batch-major, so by default this function accepts input and emits output in batch-major form.
 - scope: VariableScope for the created subgraph; defaults to "rnn".

THE END.