Deep Learning

조희철

2017년 11월 22일

차 례

1 Language Model

이 내용은 Geoffrey Hinton 교수님의 Machine Learning 강의 중 Language Model(A Neural Probabilistic Language Model, Bengio et al)을 정리하는 것임을 밝힌다. ML의 초보자로서 처음 Language Model을 접했을 때, 이해하지 못 한 것들을 나름의 방식으로 이해하여 정리하고자 한다. 좀 더 자세히 말하면, Assignment2에 있는 코드를 분석하여 정리한 것이다. 다만, Back Propagation 계산에 필요한 Chain Rule은 어느정도 알고 있다고 가정하고 설명한다. (그림이나 내용은 추가적인 인용없이 사용하겠습니다. https://www.coursera.org/learn/neural-networks/exam/9l5Ts/programming-assignment-2-learning-word-representations)

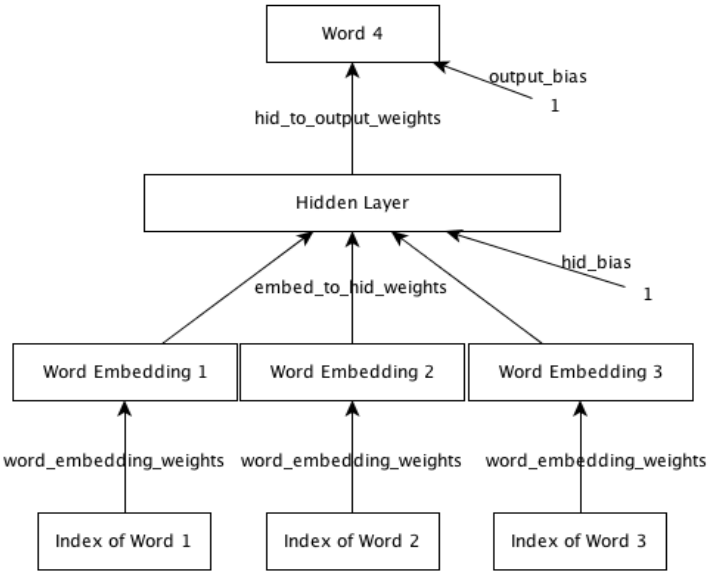


그림 1: 강의에 나오는 network 그림

• **들어가며**

1. 이 모델은 문장 속 4개의 연속된 단어를 학습한 후, 앞 3개의 단어가 주어졌을 때, 다음 네번째 단어를 예측하는 모델이다.
2. 앞으로 설명할 때, data의 개수, hidden layer의 dimension등은 N, M과 같이 일반화하여 나타내지 않고, 구체적인 숫자로 설명하고자 한다. 구체적인 숫자로 설명하는 것이 행렬의 사이즈를 따라가기 편하기 때문.
3. Assignment에 예로 나오는 숫자를 동일하게 기본으로 사용하겠습니다. 숫자를 달리하여 일반화 하는 것은 별 문제 없습니다.
    - 3: 학습하는 모델이 단어 세개를 받아들여서 학습하는 모델
    - 100: bacth size
    - 50: 각 단어의 feature를 만들기 위해 feature의 크기를 50으로 설정함(Dimensionality of embedding space).
    - 250: 학습에 등장하는 모든 단어가 250개.
    - 200: hidden layer의 dimension(또는, Number of units in hidden layer).

• **Forward Propagation**

0. learning에 사용할 단어는 모두 250개. 각 단어를 $1 \sim 250$의 숫자로 표시.
1. X_0: 입력 데이터로 3개의 단어와 batch size를 100으로 하기 때문에, 3×100 행렬.
2. W_0: 강의 에서는 이 변수를 'word embedding weights'라고 하는데, 250개의 각 단어들의 feature를 만들기 위한 것. 각 단어의 feature를 50(Dimensionality of embedding space)개로 설정하면, W_0의 크기는 250×50이 된다. 학습이 이루워지면, 이 값이 각 단어들의 feature가 되고, 이 feature가 유사하다는 것은 대체할 수 있다는 것으로 해석할 수도 있다. Assignment에서는 'display_nearest_words'라는 함수를 제공하고 있는데, 이

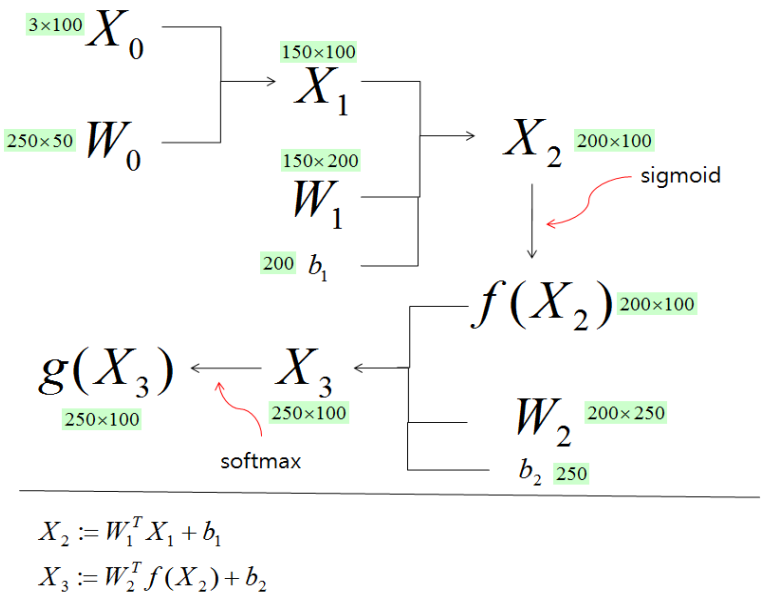


그림 2: 행렬로 표현한 network

함수는 유사한 feature를 가지 단어를 제시해 준다. 예를 들면 learning 후, 'could'를 'display_nearest_words'에 넣어보면, 'can', 'should', 'would', 'might', 'may', 'will' 순으로 유사성이 높은 단어로 제시해 준다.

3. X_0, W_0로 부터 X_1 생성 : 먼저 W_0를 행벡터 형식으로 표시하면,

$$W_0 = \begin{pmatrix} \boxed{w_{01}} \\ \boxed{w_{02}} \\ \vdots \\ \boxed{w_{0250}} \end{pmatrix}, w_{0i} : 1 \times 50.$$

예를 들어, X_0으로부터 X_1을 다음과 같이 생성한다.

$$X_0 = \begin{pmatrix} & 3 & \\ \cdots & 21 & \cdots \\ & 7 & \end{pmatrix} \quad \Longrightarrow \quad X_1 = \begin{pmatrix} & \boxed{w_{03}^T} & \\ & \boxed{w_{021}^T} & \\ \cdots & & \cdots \\ & \boxed{w_{07}^T} & \end{pmatrix}$$

이렇게 X_0, W_0로 부터, X_1을 생성하는 것은 간단할 수 있지만, 나중에 back propagation을 위한 미분 과정을 이해하기는 어려워진다.

4. 이번에는 X_1을 다른 방식으로 생성하기 위해, 먼저 X_0를 one hot encoding 형식으로 변환해 보자.

X_0의 첫행을 one hot encoding으로 변환한 행렬을 $K_1(250 \times 100)$,

X_0의 두번행을 one hot encoding으로 변환한 행렬을 K_2,

X_0의 세번행을 one hot encoding으로 변환한 행렬을 K_3.

이렇게 구해진 K_1, K_2, K_3는 각각 250×100 행렬이 된다. 이게 W_0와 K_1, K_2, K_3를 결합한 행렬과의 곱을 계산을 보자.

$$W_0^T \quad \Big(K_1 \mid K_2 \mid \ K_3\Big) = \Big(W_0^T K_1 \mid W_0^T K_2 \mid W_0^T K_3\Big)$$

이렇게 구한 $W_0^T K_1, W_0^T K_2, W_0^T K_3$를 세로로 재배치하면 X_1과 동일한 행렬이 된다. 즉,

$$X_1 = \begin{pmatrix} W_0^T K_1 \\ - - - \\ W_0^T K_2 \\ - - - \\ W_0^T K_3 \end{pmatrix} \tag{1}$$

여기서는 (행렬 연산이 아닌) 재배치 과정을 통해 X_1을 구했는데, 행렬 연산만으로 X_1을 구할 수 있는지는 모르겠음.

4. 이제 나머지 forward propagation은 [그림 2]를 보면, 쉽게 알 수 있다.

$$\begin{aligned} X_2 &= W_1^T X_1 + b_1 \longleftarrow 200 \times 100 \\ f(X_2) &: \text{sigmoid function} \\ X_3 &= W_2^T f(X_2) + b_2 \longleftarrow 250 \times 100 \\ g(X_3) &: \text{softmax function} \end{aligned}$$

• Backward Propagation

forwad propagation을 통해 구해진, output을 $y(250 \times 100)$, one hot encoding으로 변환된 target값을 $t(250 \times 100)$로 하자. cross-entropy-cost(regularization) 함수와 sortmax 함수를 미분하여 구해진, error 값 $y-t$로 부터 구해지는 gradient는 다음과 같다 ($\circ$: Hadamard product). λ를 weight decay coefficient라 하자.

$$\begin{aligned} dX_3 &= (y-t)/100 \\ dW_2 &= f(X_2)\ (y-t)^T + \lambda W_2 \longleftarrow 200 \times 250 \\ db_2 &= \text{sum over rows}(y-t) \longleftarrow 250 \\ dX_2 &= \Big(W_2(y-t)\Big) \circ \Big(f(X_2) \circ \big(1 - f(X_2)\big)\Big) \longleftarrow 200 \times 100 \\ dW_1 &= X_1\ dX_2^T + \lambda W_1 \longleftarrow 150 \times 200 \\ db_1 &= \text{sum over rows}(dX_2) \longleftarrow 200 \\ dX_1 &= W_1\ dX_2 \longleftarrow 150 \times 100 \end{aligned}$$

이제 남은 것은 dW_0를 구하는 것만 남았다. 식 (1)을 다시 보면 알 수 있듯이, W_0로부터 X_1을 구하는 과정을 보면 X_1은 3개의 block으로 분할되어 있다. W_0의 gradient는 각각의 block에 대한 W_0의 gradient를 구하여 합하면 된다. 따라서, $X_1(150 \times 100)$이 3개의 50×100 행렬로 나누어져 있으므로, $dX_1(150 \times 100)$도 동일하게 50×100 크기의 3개의 block$(dX_{11}, dX_{12}, dX_{13})$으로 분해하자.

$$X_1 = \begin{pmatrix} W_0^T K_1 \\ - - - \\ W_0^T K_2 \\ - - - \\ W_0^T K_3 \end{pmatrix}, \qquad dX_1 = \begin{pmatrix} dX_{11} \\ - - - \\ dX_{12} \\ - - - \\ dX_{13} \end{pmatrix}, \quad dX_{1i} : 50 \times 100$$

이제 dW_0은 다음과 같이 구해진다.

$$dW_0 = \sum_{i=1}^{3} K_i \; dX_{1i}^T$$

2 Chain Rule

• 행렬의 미분

$m \times n$ 행렬 A, $n \times k$ 행렬 X와 함수 $f : \mathbb{R}^{mk} \to \mathbb{R}$에 대하여, A, X의 각 원소에 대한 미분을 구해보자.

$$\begin{matrix} \boxed{A} \\ \\ \boxed{X} \end{matrix} \longmapsto AX \xmapsto{f} f(AX) \in \mathbb{R}$$

$$A = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{pmatrix}, \; X = \begin{pmatrix} x_{11} & \cdots & x_{1k} \\ \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{nk} \end{pmatrix}, \; AX = \begin{pmatrix} y_{11} & \cdots & y_{1k} \\ \vdots & \ddots & \vdots \\ y_{m1} & \cdots & y_{mk} \end{pmatrix}.$$

$\frac{\partial f}{\partial Y}$를 다음과 같이 정의하면,

$$\frac{\partial f}{\partial Y} := \begin{pmatrix} \frac{\partial f}{\partial y_{11}} & \cdots & \frac{\partial f}{\partial y_{1k}} \\ \vdots & \ddots & \vdots \\ \frac{\partial f}{\partial y_{m1}} & \cdots & \frac{\partial f}{\partial y_{mk}} \end{pmatrix}$$

A, X의 각 원소별 미분은 다음과 같다.

$$\begin{pmatrix} \frac{\partial f}{\partial a_{11}} & \cdots & \frac{\partial f}{\partial a_{1n}} \\ \vdots & \ddots & \vdots \\ \frac{\partial f}{\partial a_{m1}} & \cdots & \frac{\partial f}{\partial a_{mn}} \end{pmatrix} = \frac{\partial f}{\partial Y} X^T, \quad \begin{pmatrix} \frac{\partial f}{\partial x_{11}} & \cdots & \frac{\partial f}{\partial x_{1k}} \\ \vdots & \ddots & \vdots \\ \frac{\partial f}{\partial x_{n1}} & \cdots & \frac{\partial f}{\partial x_{nk}} \end{pmatrix} = A^T \frac{\partial f}{\partial Y}$$

• Chain Rule

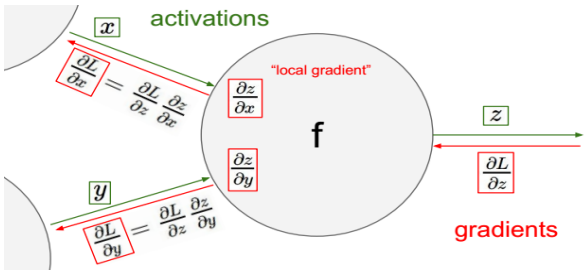


그림 3: Back Propagation & Chain Rule

$$\begin{array}{ccccccccccc} \boxed{x} & \underset{\times(-1)}{\overset{\times(-1)}{\rightleftarrows}} & \boxed{f} & \underset{g}{\overset{\exp}{\rightleftarrows}} & \boxed{g} & \underset{\times 1}{\overset{+1}{\rightleftarrows}} & \boxed{h} & \underset{-y^2}{\overset{inverse}{\rightleftarrows}} & \boxed{y} \\ \boxed{x} & \longrightarrow & \boxed{-x} & \longrightarrow & \boxed{e^{-x}} & \longrightarrow & \boxed{1+e^{-x}} & \longrightarrow & \boxed{\frac{1}{1+e^{-x}}} \\ \boxed{y^2 e^{-x}} & \longleftarrow & \boxed{-y^2 e^{-x}} & \longleftarrow & \boxed{-y^2} & \longleftarrow & \boxed{-y^2} & \longleftarrow & \boxed{1} \end{array}$$

• Example

[그림4]와 같이 모델이 주어져 있다고 하자.

$$
\begin{aligned}
z_1 &= w_1 x_1 + w_2 x_2 \\
z_2 &= w_3 x_2 + w_4 x_3 \\
h_1 &= \sigma(z_1) \\
h_2 &= \sigma(z_2) \\
y &= u_1 h_1 + u_2 h_2 \\
E &= \frac{1}{2}(t-y)^2
\end{aligned}
$$

그림 4: Example: back propagation

Forward Propagation:

$$
\begin{aligned}
& \text{Data: } \begin{pmatrix} x_1 & x_2 & x_3 \end{pmatrix}, \quad \text{weight: } \begin{pmatrix} w_1 & 0 \\ w_2 & w_3 \\ 0 & w_4 \end{pmatrix} \\
\longrightarrow \quad & \begin{pmatrix} x_1 & x_2 & x_3 \end{pmatrix} \begin{pmatrix} w_1 & 0 \\ w_2 & w_3 \\ 0 & w_4 \end{pmatrix} =: \begin{pmatrix} z_1 & z_2 \end{pmatrix} \\
\longrightarrow \quad & \begin{pmatrix} \sigma(z_1) & \sigma(z_2) \end{pmatrix} =: \begin{pmatrix} h_1 & h_2 \end{pmatrix}, \quad \sigma : \text{simgoid} \\
\longrightarrow \quad & \begin{pmatrix} h_1 & h_2 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} =: y, \quad u_i : \text{weight} \\
\longrightarrow \quad & \frac{1}{2}(t-y)^2 := E(\text{Error})
\end{aligned}
$$

Backward Propagation:

$$
\begin{aligned}
\frac{\partial E}{\partial y} &= -(t-y) \leftarrow 1 \times 1, \\
\begin{pmatrix} \frac{\partial E}{\partial h_1} & \frac{\partial E}{\partial h_2} \end{pmatrix} &= \begin{pmatrix} \frac{\partial E}{\partial y} \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \end{pmatrix}^T, \\
&= \begin{pmatrix} \frac{\partial E}{\partial y} u_1 & \frac{\partial E}{\partial y} u_2 \end{pmatrix} \leftarrow 1 \times 2, \\
\begin{pmatrix} \frac{\partial E}{\partial z_1} & \frac{\partial E}{\partial z_2} \end{pmatrix} &= \begin{pmatrix} h_1(1-h_1) & h_2(1-h_2) \end{pmatrix} \circ \begin{pmatrix} \frac{\partial E}{\partial y} u_1 & \frac{\partial E}{\partial y} u_2 \end{pmatrix} \\
&= \begin{pmatrix} h_1(1-h_1)\frac{\partial E}{\partial y} u_1 & h_2(1-h_2)\frac{\partial E}{\partial y} u_2 \end{pmatrix}, \\
\begin{pmatrix} \frac{\partial E}{\partial w_1} & 0 \\ \frac{\partial E}{\partial w_2} & \frac{\partial E}{\partial w_3} \\ 0 & \frac{\partial E}{\partial w_4} \end{pmatrix} &= \begin{pmatrix} x_1 & x_2 & x_3 \end{pmatrix}^T \begin{pmatrix} h_1(1-h_1)\frac{\partial E}{\partial y} u_1 & h_2(1-h_2)\frac{\partial E}{\partial y} u_2 \end{pmatrix} \\
&= \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \begin{pmatrix} h_1(1-h_1)\frac{\partial E}{\partial y} u_1 & h_2(1-h_2)\frac{\partial E}{\partial y} u_2 \end{pmatrix} \leftarrow 3 \times 2
\end{aligned}
$$

$W = \begin{pmatrix} w_1 & 0 \\ w_2 & w_3 \\ 0 & w_4 \end{pmatrix}$	$\searrow$				$\begin{pmatrix} u_1 \\ u_2 \end{pmatrix}$	$\searrow$	t	$\searrow$
$X = \begin{pmatrix} x_1 & x_2 & x_3 \end{pmatrix}$	$\rightarrow$	$\begin{pmatrix} z_1 & z_2 \end{pmatrix} = XW$	$\xrightarrow{\text{sigmoid}}$	$(h_1 h_2)$	$\rightarrow$	y	$\rightarrow$	E
$X^T \begin{pmatrix} \frac{\partial E}{\partial z_1} & \frac{\partial E}{\partial z_2} \end{pmatrix}$	$\nwarrow$			$(h_1 h_2)^T \frac{\partial E}{\partial y}$	$\nwarrow$	$(M \times 1)$		
		$\begin{pmatrix} \frac{\partial E}{\partial z_1} & \frac{\partial E}{\partial z_2} \end{pmatrix}$	$\longleftarrow$	$\frac{\partial E}{\partial y} \begin{pmatrix} u_1 \\ u_2 \end{pmatrix}^T$	$\longleftarrow$	$\frac{\partial E}{\partial y}$		

3 Deep Neural Net

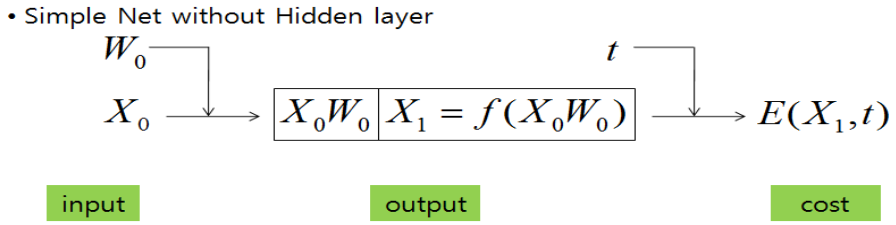
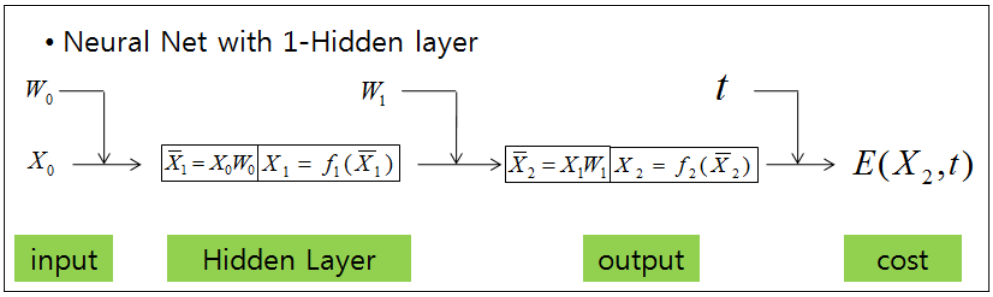
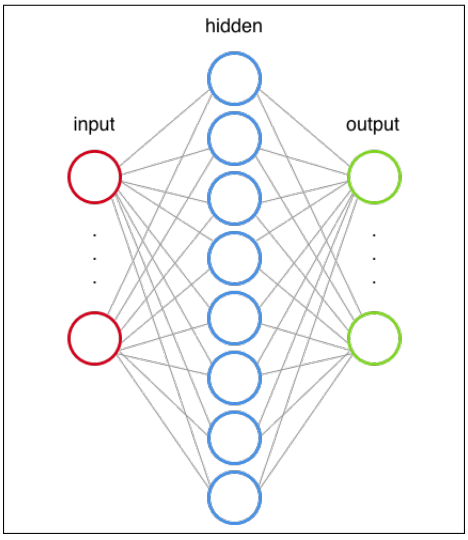


그림 5: Simple Net without Hidden Layer

- $f(x) = x$ 인 경우 : Linear Regression.
- $f(x) =$ sigmoid: Logistic Regression.
- $f(x) =$ softmax: Multinomial Classification.
- E: cost function, L_2-norm, cross entropy error.

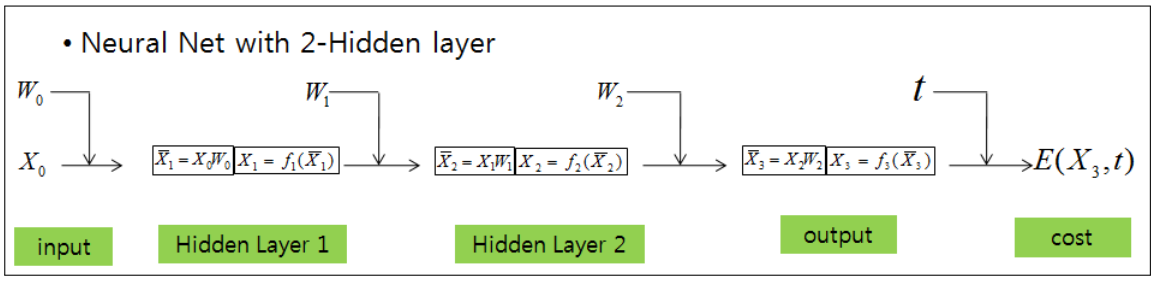


(a)

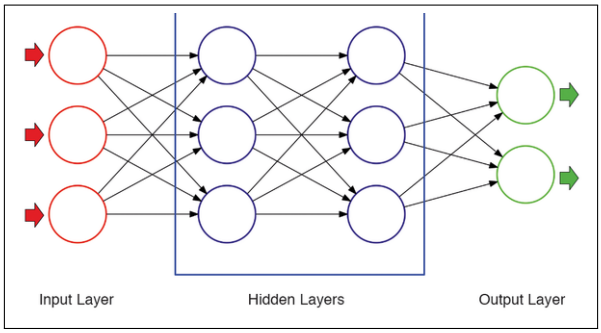


(b)

그림 6: Neural Network with 1-hidden layer



(a)



(b)

그림 7: Neural Network with 2-hidden layers

4 Back Propagation

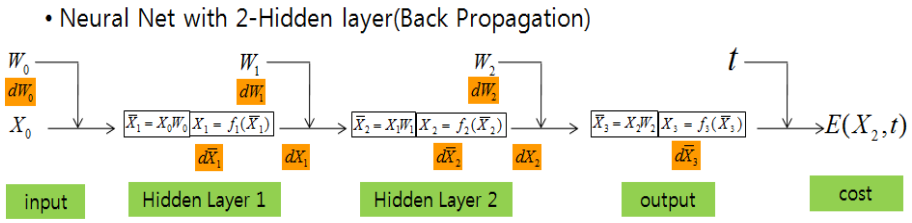


그림 8: Back Propagation

$X_i, \overline{X}_i$에 대한 미분값은 뒤로 계속 흘러가고, W_i는 각 layer마다 계산된다. 미분값이 뒤로 계속 흘러가는 것을 전파(back propagation)된다고 하는 것이다.

4.1 A Neural Network in 11 Lines

```
01.  X = np.array([ [0,0,1],[0,1,1],[1,0,1],[1,1,1] ])
02.  y = np.array([[0,1,1,0]]).T
03.  syn0 = 2*np.random.random((3,4)) - 1
04.  syn1 = 2*np.random.random((4,1)) - 1
05.  for j in xrange(60000):
06.      l1 = 1/(1+np.exp(-(np.dot(X,syn0))))
07.      l2 = 1/(1+np.exp(-(np.dot(l1,syn1))))
08.      l2_delta = (y - l2)*(l2*(1-l2))
09.      l1_delta = l2_delta.dot(syn1.T) * (l1 * (1-l1))
10.      syn1 += l1.T.dot(l2_delta)
11.      syn0 += X.T.dot(l1_delta)
```

그림 9: A Neural Network in 11 Lines

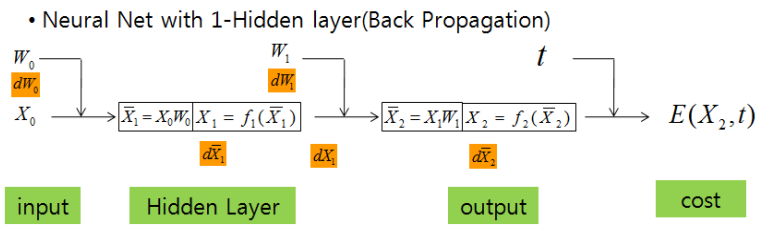


그림 10: Back Propagation with 1-Hidden Layer

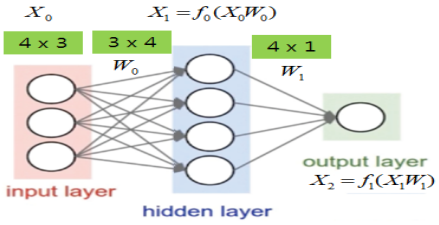


그림 11: Size of Neural Net

$$
\begin{aligned}
dX_2 &= -(t - X_2) \\
d\overline{X}_2 &= dX_2 \circ (1 - X_2) \circ X_2 = -(t - X_2) \circ (1 - X_2) \circ X_2 \\
dW_1 &= X_1^T d\overline{X}_2 \\
dX_1 &= d\overline{X}_2 W_1^T \\
d\overline{X}_1 &= dX_1 \circ (1 - X_1) \circ X_1 = d\overline{X}_2 W_1^T \circ (1 - X_1) \circ X_1 \\
dW_0 &= X_0^T d\overline{X}_1
\end{aligned}
$$

```
X0 = np.array([ [0,0,1],[0,1,1],[1,0,1],[1,1,1] ])
t = np.array([[0,1,1,0]]).T
W0 = 2*np.random.random((3,4)) - 1
W1 = 2*np.random.random((4,1)) - 1
for j in range(60000):
    X1 = 1/(1+np.exp(-(np.dot(X0,W0))))
    X2 = 1/(1+np.exp(-(np.dot(X1,W1))))
    dX2 = (t - X2)*(X2*(1-X2))
    dX1 = dX2.dot(W1.T) * (X1 * (1-X1))
    W1 += X1.T.dot(dX2)
    W0 += X0.T.dot(dX1)
```

그림 12: A Neural Network in 11 Lines

5 Convolution Neural Network

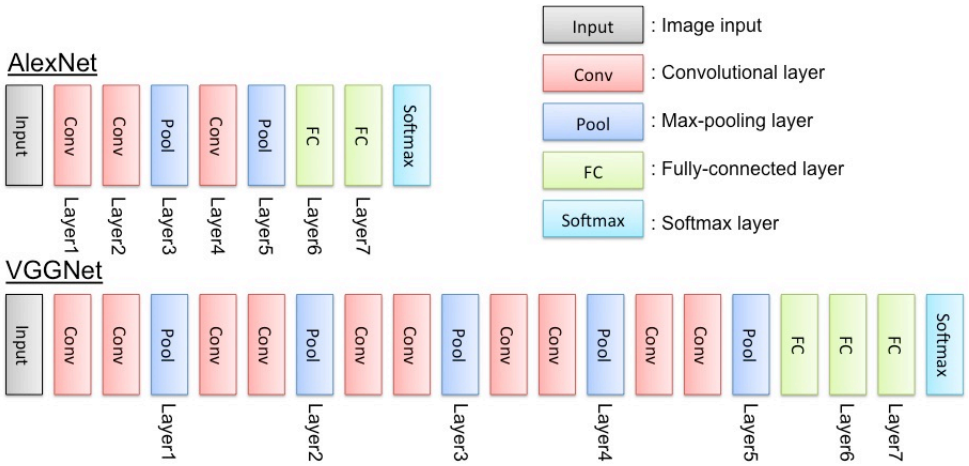


그림 13: CNN Architecture. Convolution Layer는 ReLU를 포함하고 있음.

• **im2col**

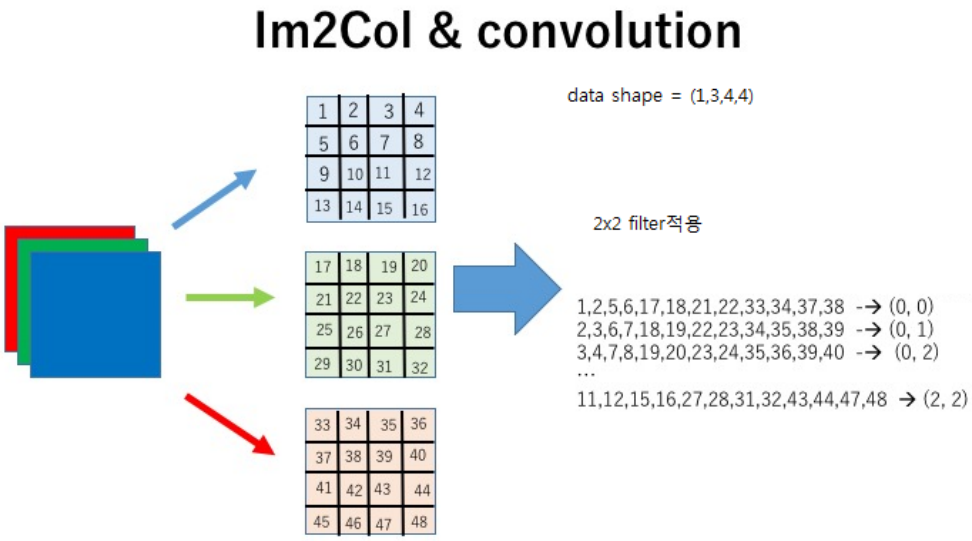


그림 14: im2col

[그림 15]에 대한 설명 :

1. $N = 1, C = 3, H \times W = 7 \times 7$ image, data shape $= (1, 3, 7, 7)$. 이 Data를 주어지 필터와 colvolution하기 위해서는 계산 목적으로 Data를 im2col 변환하는 것이 필요하다.

2. $FH = 5, FW = 5, \text{stride} = 1, \text{pad} = 0$ 인 필터를 적용하기 위한 im2col 변환을 생각해 보자.

$$OH = \frac{H - 2 \times \text{pad} - FH}{\text{stride}} + 1 = 3, OW = \frac{W - 2 \times \text{pad} - FW}{\text{stride}} + 1 = 3$$

3. im2col 변환은 필터 5×5 만큼의 Data를 행으로 펼쳐 첫번째 행를 생성한다. 2번째 행은 window를 이동시켜 5×5 Data를 행으로 펼친다. 이렇게 새로운 Data를 생성하면, $9 (= 3 \times 3 = OH \times OW)$ 만큼의 행이 만들어진다.

im2col결과

0	1	2	3	4	7	8	9	10	11	14	15	16	17	18	21	**22**	23	24	25	28	29	30	31	32	49	50	
1	2	3	4	5	8	9	10	11	12	15	16	17	18	19	**22**	23	24	25	26	29	30	31	32	33	50	51	
2	3	4	5	6	9	10	11	12	13	16	17	18	19	20	23	24	25	26	27	30	31	32	33	34	51	52	
7	8	9	10	11	14	15	16	17	18	21	**22**	23	24	25	28	29	30	31	32	35	36	37	38	39	56	57	
8	9	10	11	12	15	16	17	18	19	**22**	23	24	25	26	29	30	31	32	33	36	37	38	39	40	57	58	...
9	10	11	12	13	16	17	18	19	20	23	24	25	26	27	30	31	32	33	34	37	38	39	40	41	58	59	
14	15	16	17	18	21	**22**	23	24	25	28	29	30	31	32	35	36	37	38	39	42	43	44	45	46	63	64	
15	16	17	18	19	**22**	23	24	25	26	29	30	31	32	33	36	37	38	39	40	43	44	45	46	47	64	65	
16	17	18	19	20	23	24	25	26	27	30	31	32	33	34	37	38	39	40	41	44	45	46	47	48	65	66	

given dcol(ex: 임의로 생성)

-0.15	0.12	1.29	-1.64	0.02	0.62	-0.81	0.21	0.70	-0.66	1.26	-1.99	0.31	0.64	2.38	0.25	**-0.53**	0.04	-1.16	0.55	2.45	-0.65	2.08	1.84	-0.11	-0.03	-0.97	
1.67	1.50	0.11	0.64	0.24	-1.42	0.66	-0.72	-0.09	0.23	-1.27	-0.65	0.88	-1.57	1.63	**0.77**	-0.31	0.92	-0.28	1.41	0.58	-1.21	-2.26	0.63	1.36	0.82	-1.88	
0.35	0.62	1.07	-0.35	-1.08	-0.72	-0.57	1.17	0.35	0.02	-0.74	0.20	1.57	1.20	-0.41	1.33	-0.68	0.83	0.39	-1.90	0.61	-0.51	0.10	0.12	0.45	0.15	-0.61	
1.52	1.76	1.38	0.33	0.73	-0.58	0.45	-0.57	-0.15	0.37	1.14	**0.05**	0.94	0.39	1.22	-0.91	1.02	-1.52	-1.03	0.64	-1.23	0.53	0.79	-1.84	-0.50	-0.09	0.65	
0.16	2.69	-1.02	-0.93	0.09	-0.70	1.45	-0.97	1.02	1.09	**0.03**	1.11	1.17	0.17	0.11	-1.21	0.61	-0.88	0.94	0.14	-1.02	0.89	-0.83	-0.52	-0.43	0.62	0.22	...
-0.27	-1.04	-0.10	-0.32	0.65	-2.08	-0.99	-0.80	-0.11	0.08	-1.06	0.58	-0.66	0.11	-1.42	-1.28	0.71	-0.67	-0.10	-1.63	0.50	-0.69	-1.65	-0.62	-0.35	2.05	0.22	
-0.99	-0.30	-2.08	0.31	0.30	-	**-0.29**	0.21	1.18	-1.24	-0.48	-1.62	0.54	-1.34	1.44	-0.81	0.96	3.76	1.01	-1.55	0.13	0.68	-1.74	0.19	0.05	-1.28	-0.03	
-0.04	0.47	-0.57	-0.46	0.45	**-0.29**	0.55	0.08	-1.07	0.45	-0.94	0.62	0.16	-1.17	0.36	0.45	1.31	0.40	-0.45	-0.83	1.85	0.90	0.08	0.34	-1.97	-0.99	0.62	
0.78	1.06	0.20	-0.39	-1.77	-1.34	-1.19	-0.60	-0.33	-0.23	-0.13	0.73	-1.63	-0.45	0.07	1.10	0.02	-0.35	-0.71	-0.16	-0.68	0.59	0.19	-1.07	0.47	-0.14	0.26	

input image

0	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31	32	33	34
35	36	37	38	39	40	41
42	43	44	45	46	47	48
49	50	51	52	53	54	55
56	57	58	59	60	61	62
63	64	65	66	67	68	69
70	71	72	73	74	75	76
77	78	79	80	81	82	83
84	85	86	87	88	89	90
91	92	93	94	95	96	97
98	99	100	101	102	103	104
105	106	107	108	109	110	111
112	113	114	115	116	117	118
119	120	121	122	123	124	125
126	127	128	129	130	131	132
133	134	135	136	137	138	139
140	141	142	143	144	145	146

gradient

-0.15	1.55	3.14	-0.91	1.69	-0.11	-1.08
2.14	-0.31	3.53	-2.32	0.12	-0.11	0.67
-2.83	-3.85	-3.11	0.41	-4.53	0.61	-2.10
1.39	**0.32**	-2.01	1.29	-2.18	2.14	-3.55
1.06	-4.86	-3.84	-2.58	1.23	1.43	-1.11
-2.04	-2.06	8.35	-1.93	-5.02	-2.59	-0.51
0.13	2.53	-1.52	0.86	0.58	-3.04	0.47
-0.03	-0.15	-2.74	0.73	0.44	-0.69	-0.37
-0.18	1.25	2.70	3.43	-1.29	-1.35	1.61
-1.44	-2.12	1.65	4.17	-2.87	0.45	1.01
1.90	-2.84	-2.15	-0.05	-0.08	1.40	-0.81
1.26	3.58	-0.93	-2.50	-0.21	-0.32	1.53
0.40	3.22	1.28	1.67	4.31	-4.17	-0.90
0.67	0.78	-0.50	-3.06	-0.47	-2.73	-0.06
0.47	-0.47	1.98	2.77	-0.06	-0.55	1.49
0.03	-2.50	0.47	-1.18	0.90	0.89	-1.85
-0.84	2.18	6.30	-4.36	2.55	1.78	-3.66
-1.34	-1.05	-2.55	0.74	-0.90	-0.69	1.03
-3.77	-2.83	-0.72	-2.22	5.61	1.60	0.70
-0.51	4.40	-2.71	1.68	1.43	2.90	0.45
-1.37	0.37	-0.91	1.50	-0.17	-0.43	-1.61

그림 15: im2col 예 : $N = 1, C = 3, H \times W = 7 \times 7$ image. data shape = (1,3,7,7), $FH = 5, FW = 5$, stride $= 1$, pad $= 0$. 입력 Data를 필터 size에 따라 im2col변환 후, back propagation을 위해 넘어온 dcol로 부터 입력 Data에 대한 gradient 계산.

4. 채널 갯수 C만큼 열 size가 C배 늘어난다. im2col의 결과는 (N×OH×OW) × (FH×FW×C) 크기의 행렬이 만들어 진다.

5. 생성된 im2col 결과를 다른 관점에서 볼 수도 있다. im2col 열(숫자 9개)을 순서대로 보면, 입력 Data를 3×3 window를 이동시키면서 생성한 것으로 볼 수도 있다.

6. 이제, Back Propagation에서 dcol로 부터 입력 Data에 대한 gradient를 계산해 보자.

7. [그림 15]를 보면 im2col에서 22가 나타나는 위치의 dcol의 값을 합한 것이 22에 대한 gradient가 된다.

$$(-0.53) + 0.77 + 0.05 + 0.03 + 0.29 + (-0.29) = 0.32$$

6 Recurrent Neural Network

일반적인 Neural Network는 RNN과 대비하기 위해 Feed Forward Neural Network라고 하기도 한다. 기본적인 RNN 모델을 LSTM과 같이 복잡한 모델과 대비하기 위해 Vanilla RNN이라 부르기도 한다. RNN은 speech recognition, language modeling, translation, image captioning 등에 활용된다.

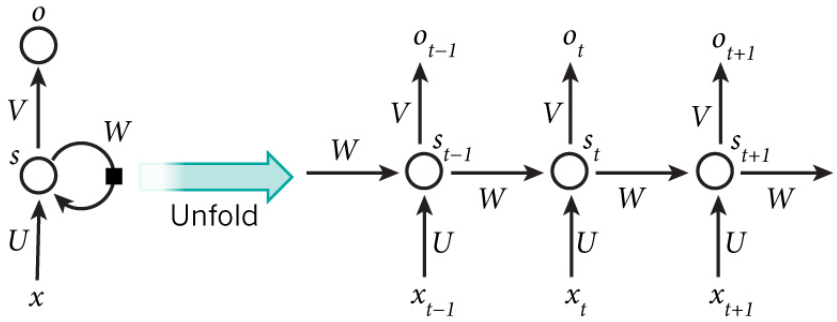


그림 16: RNN

$$
\begin{aligned}
s_{-1} &= 0, \\
s_t &= f(Ux_t + Ws_{t-1} + b_h), \text{ where } f = \tanh \text{ or ReLU}, \\
o_t &= \text{SoftMax}(Vs_t + b_y)
\end{aligned}
$$

모든 t에 동일한 U, W, V, b_h, b_y를 적용하는데, shared weights라고 한다.

일반적으로 Deep하게 쌓으면, Neural Network은 성능이 좋아지므로, 독립적인 RNN(또는 LSTM)을 여러개 쌓아서 Deep Networks을 구성할 수 있다.

$$
\begin{aligned}
y_1 &= RNN_1(x) \\
y_2 &= RNN_2(y_1) \\
&\vdots \\
y &= RNN_n(y_{n-1})
\end{aligned}
$$

• **RNN Toy Model: i am trask blog**

$n = 7$자리 2진수$(0 \sim 2^7 - 1)$ 2개를 더하는 training을 수행한다. 예를 들어, $9(00001001), 60(00111100)$를 입력하면 결과로 $69(01000101)$ 가 나오기를 기대하는 Model을 만들고자 한다. 7자리 2진수를 더하기 때문에 그 합은 8자리까지 된다. (RNN에서는 batch size = 1이 기본이다.)

output(y_i)	1	0	1	0	0	0	1	0
input	1	0	0	1	0	0	0	0
(x_i)	0	0	1	1	1	1	0	0
time i	0	1	2	3	4	5	6	7

- Cost Function으로 L^2-norm을 사용하자. L^2-norm 대신에 Logistic Cost를 사용해도 됨.
- 2개의 숫자를 더하는 Model이므로 $x_i : 2 \times 1$, $h_i : N_h \times 1, y_i : 1 \times 1$.
- $W_{xh} : N_h \times 2, W_{hh} : N_h \times N_h, W_{hy} : 1 \times N_h$

- Forward Propagation

$$\begin{aligned} z_i &= W_{xh}x_i + W_{hh}h_{i-1} \\ h_i &= \sigma(z_i) = \sigma(W_{xh}x_i + W_{hh}h_{i-1}) \longleftarrow (N_h \times 1) \\ y_i &= \sigma(W_{hy}h_i) \longleftarrow (1 \times 1) \end{aligned}$$

- Backward Propagation:

$$\begin{aligned} dW_{hy} &= \sum_{i=0}^{n} \big((t_i - y_i) \circ (y_i(1-y_i))\big)h_i^T \longleftarrow (1 \times 1)(1 \times N_{hy}) \\ dW_{hh} &= \sum_{i=0}^{n} dz_i h_{i-1}^T, \text{where } dz_i = \Big(W_{hy}^T\big((t_i - y_i) \circ (y_i(1-y_i))\big) + W_{hh}^T dz_{i+1}\Big) \circ \Big(h_i(1-h_i)\Big) \\ dW_{xh} &= \sum_{i=0}^{n} dz_i x_i^T \end{aligned}$$

- L^2-norm 대신에 Logistic Cost Function을 사용한다면,

$$\begin{aligned} dW_{hy} &= \sum_{i=0}^{n} \big(t_i - y_i\big)h_i^T \longleftarrow (1 \times 1)(1 \times N_{hy}) \\ dW_{hh} &= \sum_{i=0}^{n} dz_i h_{i-1}^T, \text{where } dz_i = W_{hy}^T\big(t_i - y_i\big) + W_{hh}^T dz_{i+1} \end{aligned}$$

- $z_{n+1} = 0$이고, $h_{-1} = 0$이므로 $i = 0$에서의 dW_{hh} 계산은 생략해도 됨. dz_i의 값은 dz_{i+1}에서 구해지므로, dW_{hh}는 $n, n-1, ..., 1$ 순으로 계산.

- 이 RNN은 0 또는 1로 된 두 수와 넘어온 carry h_{i-1} 모두 3개의 숫자를 결합(연산)하여 y_i와 carry로 넘겨줄 새로운 h_i를 구하는 과정을 구현한 것임. 따라서 7자리 2진수로 training한 RNN에 10자리 이진수를 적용해도 좋은 결과를 얻을 수 있다.

- **DENNY BRITZ Language Model**

```
I joined a new league this year and they have different scoring rules than I'm used to.
It's a slight PPR league- .2 PPR.
Standard besides 1 points for 15 yards receiving, .2 points per completion, 6 points per TD thrown,
My question is, is it wildly clear that QB has the highest potential for points?
...
x_train[0]: SENTENCE_START i joined a new league this year and they have different
scoring rules than i 'm used to .
  [0, 6, 3492, 7, 155, 792, 25, 223, 8, 33, 20, 202, 4952, 349, 91, 6, 66, 207, 5, 2]
y_train[0]: i joined a new league this year and they have different scoring rules
than i 'm used to . SENTENCE_END
  [6, 3492, 7, 155, 792, 25, 223, 8, 33, 20, 202, 4952, 349, 91, 6, 66, 207, 5, 2, 1]

x_train[1]: SENTENCE_START it 's a slight ppr UNKNOWN_TOKEN UNKNOWN_TOKEN ppr .
  [0, 11, 17, 7, 3095, 5974, 7999, 7999, 5974, 2]
y_train[1]: it 's a slight ppr UNKNOWN_TOKEN UNKNOWN_TOKEN ppr . SENTENCE_END
  [11, 17, 7, 3095, 5974, 7999, 7999, 5974, 2, 1]
```

- data 생성 : 입력 파일을 parsing하여 단어를 구하고, 빈도수 기준으로 상위 N_x 개만을 대상으로 한다. 그 외의 단어는 'UNKNOWN_TOKEN'으로 대체한다. 문장 단위로 training data를 만든다. 각 문장의 앞에는 'SENTENCE_START'를 넣는다. target data의 끝에는 'SENTENCE_END'른 넣는다. 각각의 단어는 숫자에 mapping한다.
- 하나의 data(문장 1개)가 length T일때, $t = 0, \cdots, T-1$. 각각의 training data마다 T가 달라 진다.
- $U : N_h \times N_x$. N_h는 hidden layer의 dimension. N_x는 input data의 차원. 예를 들어, Language Model에서 사용하는 모든 단의의 갯수가 8,000개 이면, $N_x = 8,000$. U의 각 column은 각 단어의 embedding weight라고 볼 수 있는데, Bengio Language Model에서의 embedding weight와 유사하다고 볼 수 있다.
- $W : N_h \times N_h$, $V : N_x \times N_h$.
- training을 시키기 위해서는 training data를 하나씩 입력하여, SGD를 적용한다. loop를 통해 모든 data를 훈련시킨 후, 원하는 epoch만큼 반복한다.
- 문장의 생성 : training 결과를 이용하여 문장을 생성하기 위해서는 먼저, 'SENTENCE_START' 1개로 된 문장을 입력하여 output으로 다음 단어의 확률을 구한다. 확률을 이용하여 random하게 다음 단어를 정한다. 이제 2개의 단어로 된 문장을 이용하여 다음 단어를 구한다. 이 작업을 마지막 단어가 'SENTENCE_END'가 나올 때까지 반복한다.
- RNN에서 gradient vanishing 현상은 Feed Forward Neural Network보다 잘 나타난다.
- gradient vanishing을 해결하는 방법은 W의 초기값을 잘 주는 방법, regularization을 사용하는 방법, sigmoid나 tanh 대신 RELU를 사용하는 방법이 있다.
- 이보다 나은 방법으로 Long Short Term Memory(LSTM, 1997년), Gated Recurrent Unit(GRU, 2004년)이 있다. GRU는 LSTM의 간소화 버전이다.

• **Minimal Character-Level Vanilla RNN Model by Andrej Karpathy**

$$
\begin{aligned}
s_t &= \tanh(Ux_t + Ws_{t-1} + b_h) \\
y_t &= \mathrm{SoftMax}(Vs_t + b_y)
\end{aligned}
$$

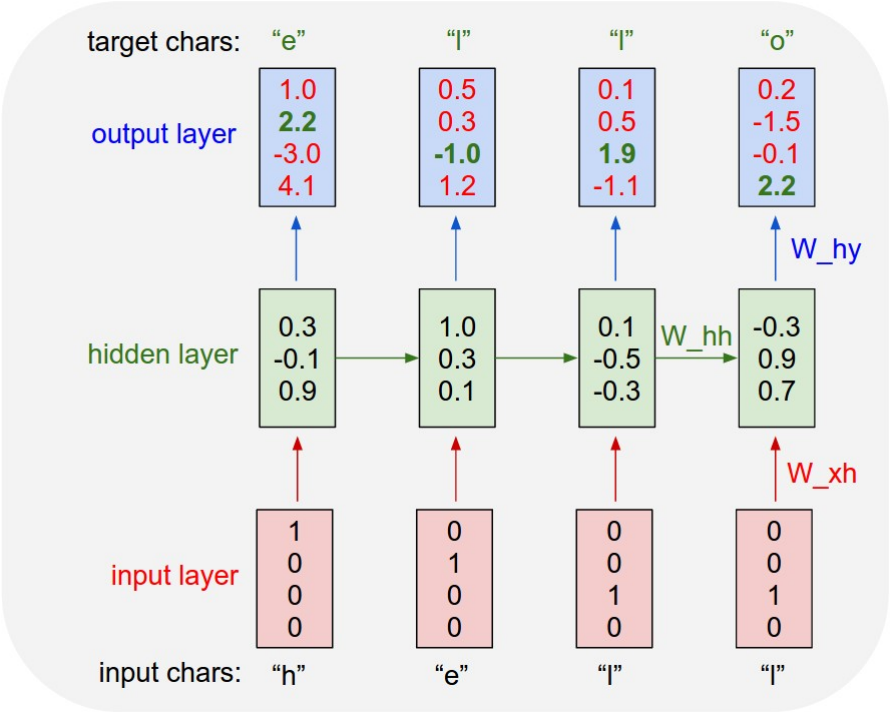


그림 17: An example RNN with 4-dimensional input and output layers, and a hidden layer of 3 units(source: Karpathy's Blog).

```
First Citizen:
Before we proceed any further, hear me speak.

All:
Speak, speak.
...

x_train[0]: 'First Citizen: Before we  '
y_train[0]: 'irst Citizen: Before we  p'

x_train[1]: 'proceed any further, hear'
y_train[1]: 'roceed any further, hear '

x_train[2]: ' me speak.  All: Speak, s'
y_train[2]: 'me speak.  All: Speak, sp'
```

- 예를 들어, Shakespeare의 작품을 training data로 활용한다. 입력 data는 단어가 아닌 character 단위로 쪼개어 입력한다. 이경우, character 개수 $N_x = 65$.

- training 후, 1개의 character를 입력하여 다음 character를 구한다(T=1인 RNN모델에 적용). 이 작업을 원하는 만큼(예를 들어, 200번) 반복한다. 각각의 작업이 완전히 독립적인 것은 아니다. 왜냐하면, Hidden State 값이 입력 값으로 전달되고, 다시 update되기 때문이다(Recurrent Connection).
- hidden state의 dimension을 $N_h = 100$으로 잡으면, $W : N_h \times N_h, U : N_h \times N_x, V : N_x \times N_h$가 된다.
- bias $b_h : N_h \times 1, b_y : N_x \times 1$.
- 이 Model에서는 sequence 길이 $T(= 24)$를 고정 한다. 즉, 전체 문장 data를 T개씩 잘라 training data를 만든다.
- Andrej Karpathy는 Paul Graham의 에세이, Shakespeare의 작품, LaTeX코드, Linux 소스 C 코드, 아기 이름을 Multi-Layer RNN이나 LSTM 모델을 적용하여 그 결과를 보여주고 있다.

• **LSTM(Long Short Term Memory) Networks**

Colah's Blog에 있는 내용을 정리.

- Long Short Term Memory by Hochreiter, Schmidhuber(1997)

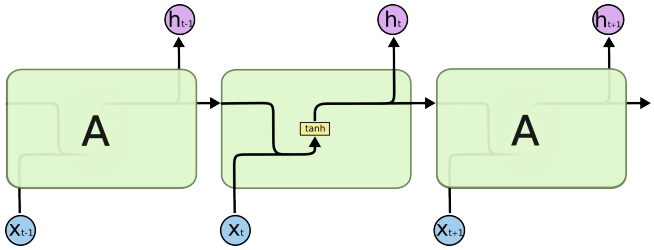


그림 18: The repeating module in a standard RNN contains a single layer(source: Colah's Blog).

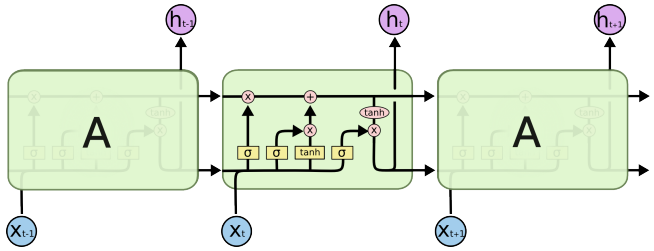
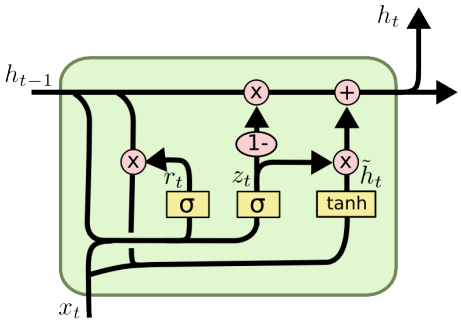


그림 19: The repeating module in an LSTM contains four interacting layers(source: Colah's Blog).

$$
\begin{aligned}
f_t &= \sigma(W^f_{xh} x_t + W^f_{hh} h_{t-1} + b^f_h) \longleftarrow \text{forget gate} \\
i_t &= \sigma(W^i_{xh} x_t + W^i_{hh} h_{t-1} + b^i_h) \longleftarrow \text{input gate} \\
o_t &= \sigma(W^o_{xh} x_t + W^o_{hh} h_{t-1} + b^o_h) \longleftarrow \text{output gate} \\
g_t &= \tanh(W^g_{xh} x_t + W^g_{xh} h_{t-1} + b^g_h) \\
c_t &= f_t \circ c_{t-1} + i_t \circ g_t \longleftarrow \text{cell state} \\
h_t &= o_t \circ \tanh(c_t) \longleftarrow \text{hidden state} \\
y_t &= \text{Softmax}(W_{hy} h_t + b_y) \longleftarrow \text{output}
\end{aligned}
$$

• **GRU(Gated Recurrent Unit) Networks**



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

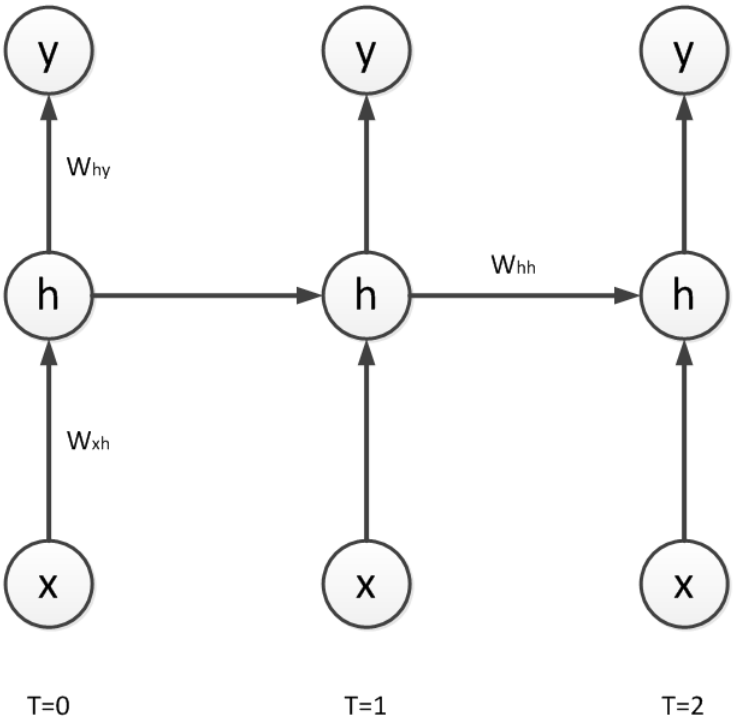
그림 20: GRU(source: Colah's Blog).

Example 6.1. *(그림 21)*과 같이 주어진 *RNN*에서 $\frac{\partial E}{\partial z_1}$을 계산해 보자.

$$\begin{aligned}
\frac{\partial E}{\partial z_1} &= \frac{\partial E_1}{\partial z_1} + \frac{\partial E_2}{\partial z_1} \\
\frac{\partial E_1}{\partial z_1} &= \frac{\partial E_1}{\partial y_1}\frac{\partial y_1}{\partial h_1}\frac{\partial h_1}{\partial z_1} = (y_1 - t_1) \times W_{hy} \times h_1(1 - h_1) \\
\frac{\partial E_2}{\partial z_1} &= \frac{\partial E_2}{\partial y_2}\frac{\partial y_2}{\partial h_2}\frac{\partial h_2}{\partial z_2}\frac{\partial z_2}{\partial h_1}\frac{\partial h_1}{\partial z_1} = (y_2 - t_2) \times W_{hy} \times h_2(1 - h_2) \times W_{hh} \times h_1(1 - h_1)
\end{aligned}$$

4. The figure below shows a Recurrent Neural Network (RNN) with one input unit x, one logistic hidden unit h, and one linear output unit

y. The RNN is unrolled in time for T=0,1, and 2.



The network parameters are: $W_{xh} = -0.1$, $W_{hh} = 0.5$, $W_{hy} = 0.25$, $h_{\text{bias}} = 0.4$, and $y_{\text{bias}} = 0.0$.

If the input x takes the values $18, 9, -8$ at time steps $0, 1, 2$ respectively, the hidden unit values will be $0.2, 0.4, 0.8$ and the output unit values will be $0.05, 0.1, 0.2$ (you can check these values as an exercise). A variable z is defined as the total input to the hidden unit before the logistic nonlinearity.

If we are using the squared loss, with targets t_0, t_1, t_2, then the sequence of calculations required to compute the total error E is as follows:

$$
\begin{array}{lll}
z_0 = W_{xh}x_0 + h_{\text{bias}} & z_1 = W_{xh}x_1 + W_{hh}h_0 + h_{\text{bias}} & z_2 = W_{xh}x_2 + W_{hh}h_1 + h_{\text{bias}} \\
h_0 = \sigma(z_0) & h_1 = \sigma(z_1) & h_2 = \sigma(z_2) \\
y_0 = W_{hy}h_0 + y_{\text{bias}} & y_1 = W_{hy}h_1 + y_{\text{bias}} & y_2 = W_{hy}h_2 + y_{\text{bias}} \\
E_0 = \frac{1}{2}(t_0 - y_0)^2 & E_1 = \frac{1}{2}(t_1 - y_1)^2 & E_2 = \frac{1}{2}(t_2 - y_2)^2 \\
E = E_0 + E_1 + E_2 & &
\end{array}
$$

If the target output values are $t_0 = 0.1, t_1 = -0.1, t_2 = -0.2$ and the squared error loss is used, what is the value of the error derivative just before the hidden unit nonlinearity at $T = 1$ (i.e. $\frac{\partial E}{\partial z_1}$)? Write your answer up to at least the fourth decimal place.

그림 21: Example

7 Redistricted Boltzmann Machine

• **Partition Function**

visible state V with 3 units, hidden state H with 2 units, weight matrix W.

$$\begin{aligned} V &= \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix}, \\ W &= \begin{pmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{pmatrix} \end{aligned}$$

- $H = (0\ 0)^T$ 인 경우,

$$\begin{aligned} H^TWV &= \begin{pmatrix} 0 & 0 \end{pmatrix} \begin{pmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} \\ &= 0. \end{aligned}$$

  V 의 3원소 v_1, v_2, v_3에 대한, $0, 1$ 조합은 모두 $2^3 = 8$가지. 따라서, $\sum_{h=(0,0)} e^{-E(V,H)} = 8$.

- $H = (0\ 1)^T$ 인 경우,

$$\begin{aligned} H^TWV &= \begin{pmatrix} 0 & 1 \end{pmatrix} \begin{pmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} \\ &= \begin{pmatrix} w_{21} & w_{22} & w_{23} \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} \\ &= w_{21}v_1 + w_{22}v_2 + w_{23}v_3. \\ \sum_{H=(0,1),V} e^{-E(V,H)} &= \sum e^{w_{21}v_1 + w_{22}v_2 + w_{23}v_3} \\ &= (1 + e^{w_{21}})(1 + e^{w_{22}})(1 + e^{w_{23}}). \end{aligned}$$

- $H = (1\ 0)^T$ 인 경우,

$$\begin{aligned} H^TWV &= \begin{pmatrix} 1 & 0 \end{pmatrix} \begin{pmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} \\ &= \begin{pmatrix} w_{11} & w_{12} & w_{13} \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} \\ &= w_{11}v_1 + w_{12}v_2 + w_{13}v_3. \\ \sum_{H=(1,0),V} e^{-E(V,H)} &= \sum e^{w_{11}v_1 + w_{12}v_2 + w_{13}v_3} \\ &= (1 + e^{w_{11}})(1 + e^{w_{12}})(1 + e^{w_{13}}). \end{aligned}$$

- $H = (1\ 1)^T$ 인 경우,

$$\begin{aligned} H^T WV &= \begin{pmatrix} 1 & 1 \end{pmatrix} \begin{pmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} \\ &= \begin{pmatrix} w_{11}+w_{21} & w_{12}+w_{22} & w_{13}+w_{23} \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} \\ &= (w_{11}+w_{21})v_1 + (w_{12}+w_{22})v_2 + (w_{13}+w_{23})v_3. \\ \sum_{H=(1,1),V} e^{-E(V,H)} &= \sum e^{(w_{11}+w_{21})v_1+(w_{12}+w_{22})v_2+(w_{13}+w_{23})v_3} \\ &= (1+e^{w_{11}+w_{21}})(1+e^{w_{12}+w_{22}})(1+e^{w_{13}+w_{23}}). \end{aligned}$$

- 4가지 경우를 모두 합치면 다음과 같다.

$$\begin{aligned} H^T &= \begin{pmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{pmatrix}, \\ H^T WV &= \begin{pmatrix} 0 & 0 & 0 \\ w_{21} & w_{22} & w_{23} \\ w_{11} & w_{12} & w_{13} \\ w_{11}+w_{21} & w_{12}+w_{22} & w_{13}+w_{23} \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix}, \\ &\longrightarrow \begin{pmatrix} 1+e^0 & 1+e^0 & 1+e^0 \\ 1+e^{w_{21}} & 1+e^{w_{22}} & 1+e^{w_{23}} \\ 1+e^{w_{11}} & 1+e^{w_{12}} & 1+e^{w_{13}} \\ 1+e^{w_{11}+w_{21}} & 1+e^{w_{12}+w_{22}} & 1+e^{w_{13}+w_{23}} \end{pmatrix}. \end{aligned}$$

  row-wise product and then sum.

8 Optimization Methods

- **Mini Batch and SGD**

  - In Stochastic Gradient Descent, you use only 1 training example before updating the gradients. When the training set is large, SGD can be faster. But the parameters will "oscillate" toward the minimum rather than converge smoothly.
  - The difference between gradient descent, mini-batch gradient descent and stochastic gradient descent is the number of examples you use to perform one update step.
  - You have to tune a learning rate hyperparameter α.
  - With a well-turned mini-batch size, usually it outperforms either gradient descent or stochastic gradient descent (particularly when the training set is large).

- **GD and Momentum**

  - Momentum takes past gradients into account to smooth out the steps of gradient descent. It can be applied with batch gradient descent, mini-batch gradient descent or stochastic gradient descent.
  - You have to tune a momentum hyperparameter β and a learning rate α.
  - Common values for β range from 0.8 to 0.999. If you don't feel inclined to tune this, $\beta = 0.9$ is often a reasonable default.

$$\begin{aligned} v &\leftarrow \beta v + (1-\beta) dW \\ W &\leftarrow W - \alpha v \end{aligned}$$

- **Adam**

$$\begin{aligned} v &\leftarrow \beta_1 v + (1-\beta_1) dW = v + (1-\beta_1)(dW - v) \\ v_{bc} &= \frac{v}{1-\beta_1^t} \\ s &\leftarrow \beta_2 s + (1-\beta_2) dW^2 = s + (1-\beta_2)(dW^2 - s) \\ s_{bc} &= \frac{s}{1-\beta_2^t} \\ W &\leftarrow W - \alpha \frac{v_{bc}}{\sqrt{s_{bc} + \epsilon}} \end{aligned}$$