Named Entity Recognition

- Idea : classify each word in its context window of neighboring words
- dictionary 사용 (x) NN에 넣어서 특정 label에 대한 확률 계산 (o)
- Ex) "Paris"가 location인지 window length 2로 살펴보자.

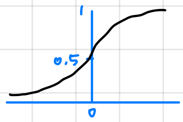
the museums in Paris are amazing to see

$x_{window} = [\, x_{museums} \; x_{in} \; x_{Paris} \; x_{are} \; x_{amazing} \,]^T$

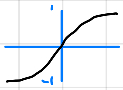
$= x \in \mathbb{R}^{5d}$

Non-linearities

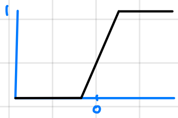
- logistic (sigmoid) : $f(z) = \dfrac{1}{1+\exp(-z)}$



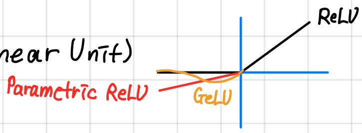
- $\tanh(z) = \dfrac{e^z - e^{-z}}{e^z + e^{-z}} = 2\,\mathrm{logistic}(2z) - 1$



- $\mathrm{HardTanh} = \begin{cases} -1 & \text{if } x < -1 \\ x & \text{if } -1 \le x \le 1 \\ 1 & \text{if } x > 1 \end{cases}$



- ReLU (Rectified Linear Unit)



* Logistic과 tanh도 쓰이지만 deep networks에서는 보통 ReLU 사용. GeLU는 Transformer에 자주 사용

* non-linearity를 써야 하는 이유 : linear function을 사용하면 layer를 여러개로 하는 의미가 없어짐

$\nabla_\theta J(\theta)$를 계산하는 방법

1) By hand (matrix calculus)

Gradient Computing

n개의 input이 주어지면 1개의 output이 나오는 함수 $f(x) = f(x_1, \cdots, x_n)$에 대하여

$$\frac{\partial f}{\partial x} = \left[\frac{\partial f}{\partial x_1}, \cdots, \frac{\partial f}{\partial x_n} \right]$$

m개의 output과 n개의 input이 주어지는 함수 $f(x) = [\, f_1(x_1, \cdots, x_n), \cdots, f_m(x_1, \cdots, x_n) \,]$에 대하여
(sub function)

$$\frac{\partial f}{\partial x} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}, \qquad \left(\frac{\partial f}{\partial x} \right)_{ij} = \frac{\partial f_i}{\partial x_j}$$

$h = f(z)$

$z = W\vec{x} + \vec{b}$ ($h, z \in \mathbb{R}^n$) 처럼 output과 input의 개수가 동일할 때는 $n \times n$ Jacobian 사용

Jacobian

$h_i = f(z_i)$

$$\left(\frac{\partial h}{\partial z} \right)_{ij} = \frac{\partial h_i}{\partial z_j} \overset{\text{정의}}{=} \frac{\partial}{\partial z_j} f(z_i) = \begin{cases} f'(z_i) & \text{if } i = j \\ 0 & \text{if otherwise} \end{cases} \qquad \therefore \frac{\partial h}{\partial z} = \begin{pmatrix} f'(z_1) & & 0 \\ & \ddots & \\ 0 & & f'(z_n) \end{pmatrix} = \mathrm{diag}(f'(z))$$

- other Jacobian

$$\frac{\partial}{\partial x}(W\vec{x} + \vec{b}) = W$$

$$\frac{\partial}{\partial b}(W\vec{x} + \vec{b}) = I$$

$$\frac{\partial}{\partial u}(u^T h) = h^T$$

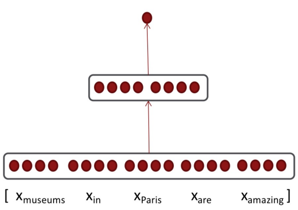
Neural Network

$s = \boldsymbol{u}^T \boldsymbol{h}$

$\boldsymbol{h} = f(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})$ ← non-linearity

$z = Wx + b$

$\boldsymbol{x}$ (input)



- loss J_t의 gradient를 바로 알기 어렵기 때문에 score의 gradient를 계산

$$\frac{\partial s}{\partial b} = \boxed{\frac{\partial s}{\partial h} \times \frac{\partial h}{\partial z}} \times \frac{\partial z}{\partial b}$$ } by using Jacobians

$$= u^T \operatorname{diag}(f'(z)) I$$

$$= u^T \odot f'(z)$$

↳ Hamard Product : element-wise multiplication of 2 vectors to give vector

ex) $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$, $A \odot A = \begin{bmatrix} 1^2 & 2^2 \\ 3^2 & 4^2 \end{bmatrix}$

$$\frac{\partial s}{\partial W} = \boxed{\frac{\partial s}{\partial h} \times \frac{\partial h}{\partial z}} \times \frac{\partial z}{\partial W}$$

$\frac{\partial s}{\partial b}$ 에서와 겹치기 때문에 계산을 줄이기 위해 delta로 바꿈

$$= \delta \frac{\partial z}{\partial b} = \delta$$ ($\delta = \frac{\partial s}{\partial h} \times \frac{\partial h}{\partial z} = u^T \odot f'(z)$ 로 upstream gradient (error signal) 이다.)

- 문제점 long row vector로 나타서 계산하기 힘듦

↓ 해결

Shape Conversion

transpose로 gradient와 parameter들의 shape을 같게 만든다

$$\frac{\partial s}{\partial W} = \delta \frac{\partial z}{\partial W} = \delta \frac{\partial}{\partial W}(Wx + b) = \delta^T x^T$$

↱ upstream gradient

↳ local input signal

2) Algorithmically (Backpropagation)

re-use derivatives computed for higher layers in computing derivatives for lower layers to minimize computation

downstream gradient = upstream gradient × local gradient