Neural Networks

Backpropagation

Neural Networks

Classification

Softmax/ Cross-entropy loss

Neural Network Classifiers

Nonlinear decision boundaries

Window Classification

Named Entity Recognition

Matrix calculus

Backpropagation하기 이전의 복습단계!

Backpropagation

Classification

Classification in NLP

x_i (Input) : 단어, 문장, 문서

y_i(label) : 감정, named entities 등

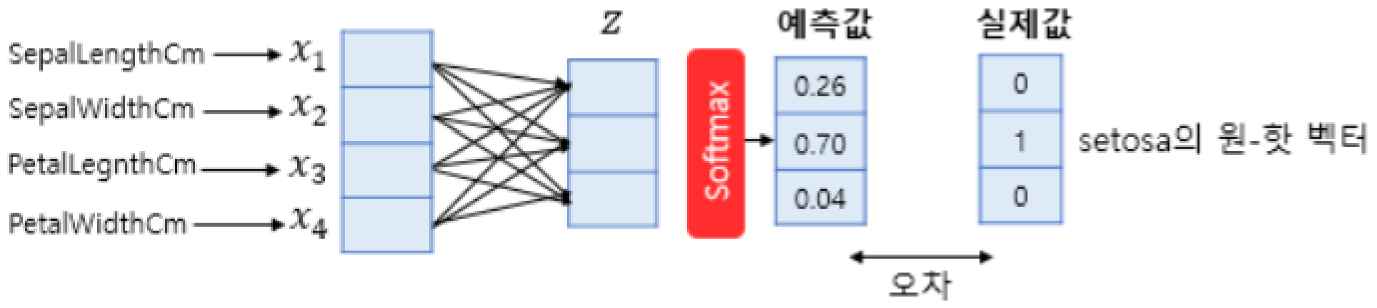
예측된 y값
(softmax 이용)

$$p(y|x) = \frac{\exp(W_y.x)}{\sum_{c=1}^{C} \exp(W_c.x)}$$

Softmax

- **이진 분류**를 하는 경우에는 시그모이드 함수를 사용
- **세 개 이상의 class**인 경우에는 소프트맥스 함수를 사용해서 0과 1 사이의 값으로 출력하며, 출력값의 총 합이 1이 되도록 한다

$$p_i = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}} \quad for\ i = 1, 2, \ldots k$$



예측값과 실제값의 오차를 계산하기 위해 비용 함수로
Cross Entropy 사용

Cross Entropy

$$H(p,q) = -\sum_{c=1}^{C} p(c)\log q(c)$$

(P는 실제 확률 분포, q는 계산된 확률)
이 함수가 비용함수로 적합한 이유?

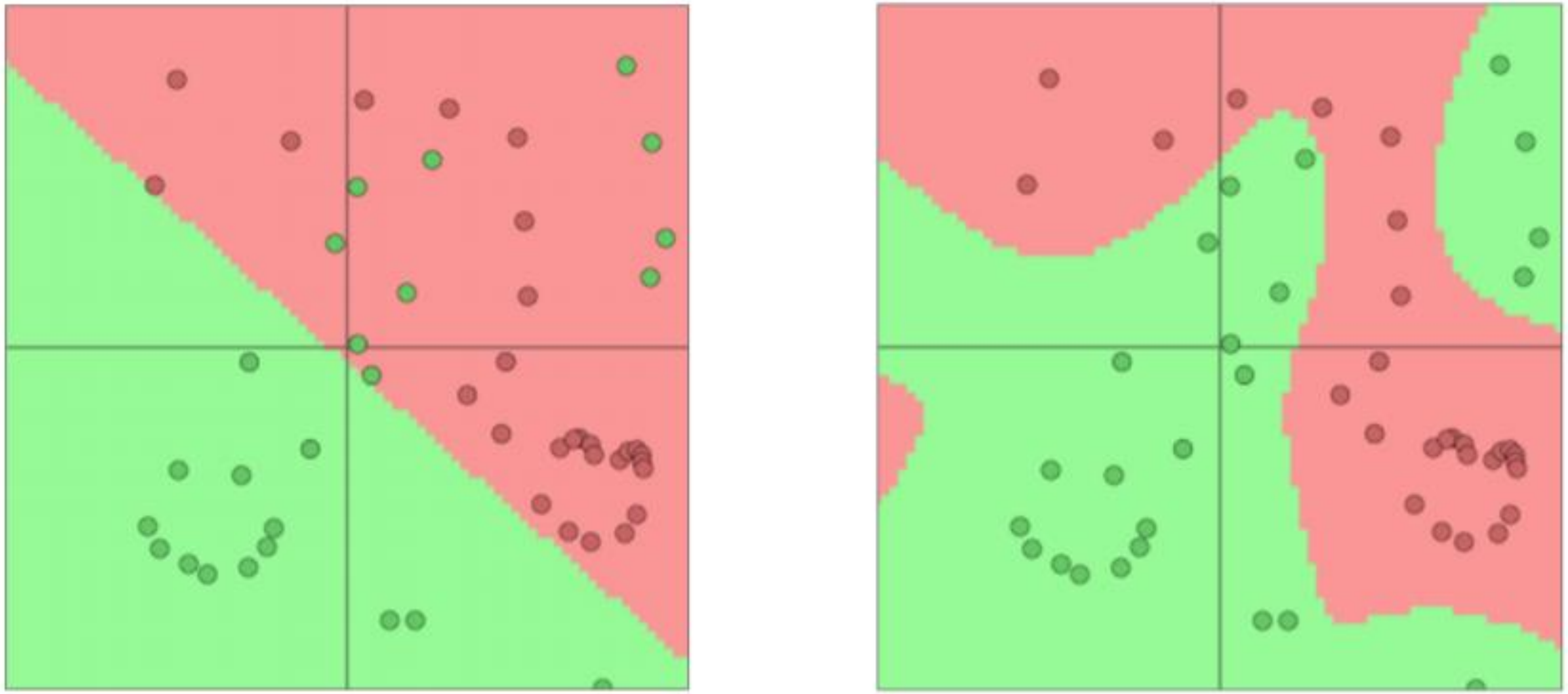
c가 one-hot vector에서 1을 가진 원소의 인덱스라고 하자
예측된 y값이 정확한 경우에 q(c)는 1이어야 한다.
이를 식에 대입해보면 -1log(1) = 0, 즉 loss 값이 0이 된다.

Cross entropy loss function over full dataset $\{x_i,y_i\}^N_{i=1}$

$$J(\theta) = \frac{1}{N}\sum_{i=1}^{N} -\log\left(\frac{e^{f_{y_i}}}{\sum_{c=1}^{C} e^{f_c}}\right)$$

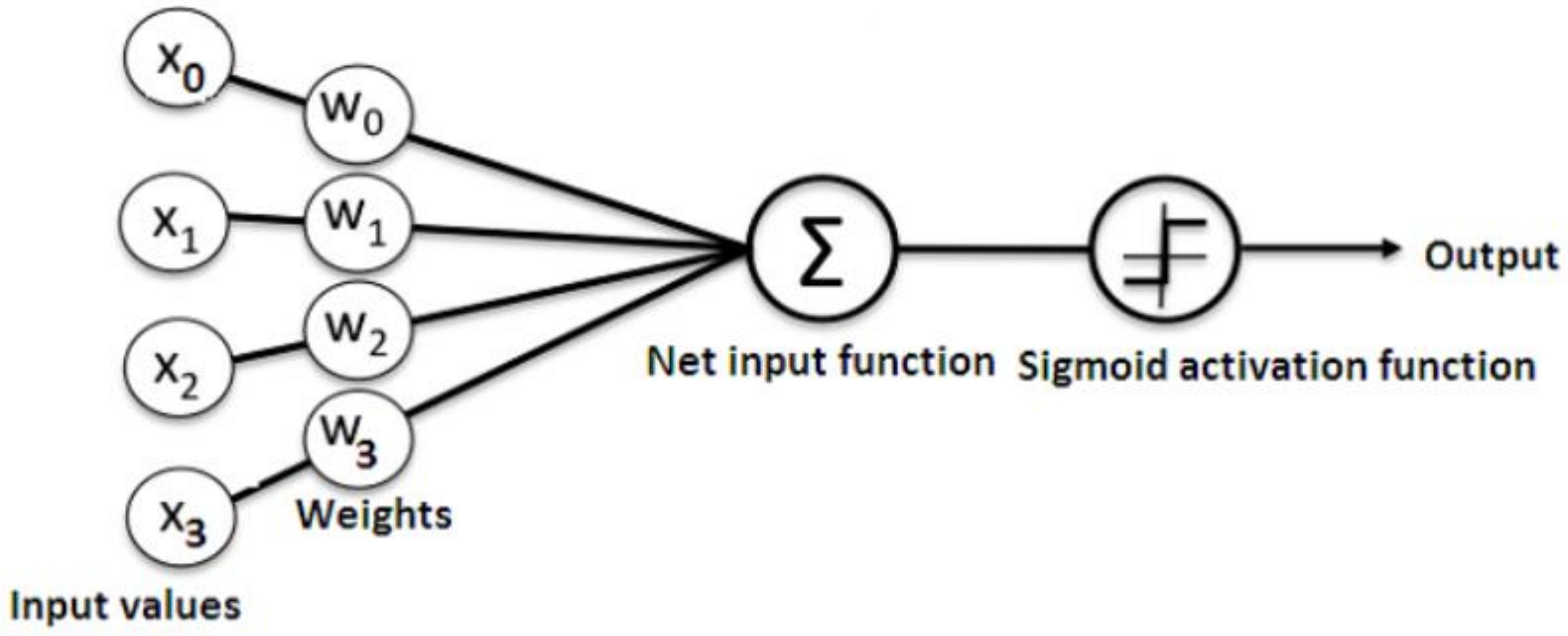
NN Classifiers

Neural Network Classifiers



Softmax은 linear한 decision boundary를 제공하기 때문에 제한적
Nonlinear decision boundary → Neural network

Neural Network Classifiers



활성화 함수는 선형함수가 아닌 **비선형 함수**여야 한다
- 선형함수를 사용하게 되면 hidden layer를 계속 쌓더라도 하나의 layer를 사용한 것과 차이 없음 $W_1 W_2 x = Wx$
- 신경망이 깊어져도 흥미로운 계산을 할 수 없게 된다!

NLP deep learning

$$\nabla_\theta J(\theta) = \begin{bmatrix} \nabla_{W_{\cdot 1}} \\ \vdots \\ \nabla_{W_{\cdot d}} \\ \nabla_{x_{aardvark}} \\ \vdots \\ \nabla_{x_{zebra}} \end{bmatrix}$$

NLP에서는 parameter(w)와 word vector(x)를 같이 학습시킨다

pre-trained word vector
훈련 데이터가 적은 상황에서 이미 훈련되어져 있는 pre-trained word vector를 임베딩 벡터로 사용하기도 한다

훈련 데이터가 적은 경우 해당 문제에 특화된 임베딩 벡터를 만드는게 쉽지 않음.
따라서 해당 문제에 특화된 것은 아니지만 일반적이고 많은 훈련데이터로 학습된 임베딩 벡터를 사용하면 성능을 개선할 수 있다.

Window Classification

Named Entity Recognition(NER)

NER은 문장 안에서 Location, Person, Organization 등 개체명(Named Entity)를 분류하는 방법론이다. '디카프리오가 나온 영화 틀어줘'라는 문장에서 '디카프리오'를 사람으로 인식하는 것을 목표로 한다.

"First National Bank Donates 2 Vans To Future School of Fort Smith"

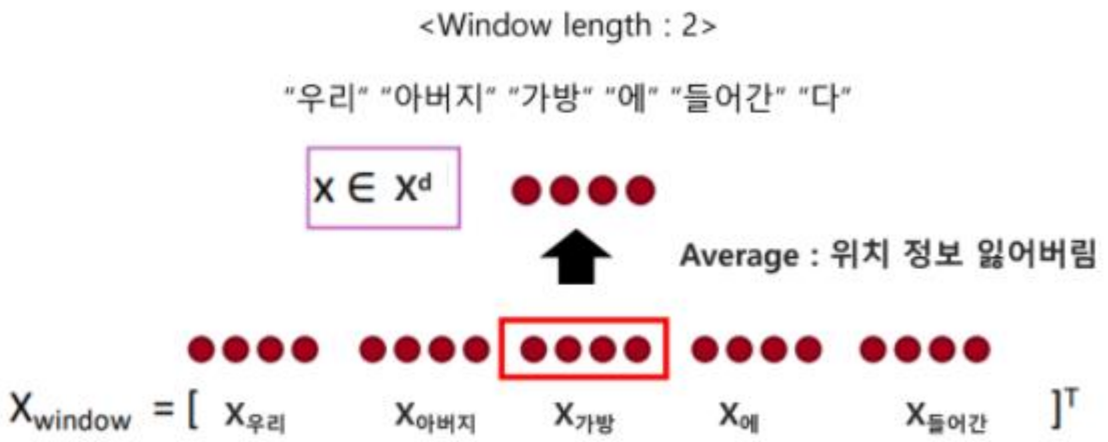
NER 힘든 이유

1. Entity 경계를 정하기 힘듦. (First National Bank vs. National Bank
2. Entity 구분하기 힘듦. (Future school?)
3. Unknown entit에 대한 class를 얻기 힘듦.

Input data: 토큰화와 품사 태깅 전처리를 끝내고 난 상태를 입력으로 한다

Named Entity Recognition(NER)

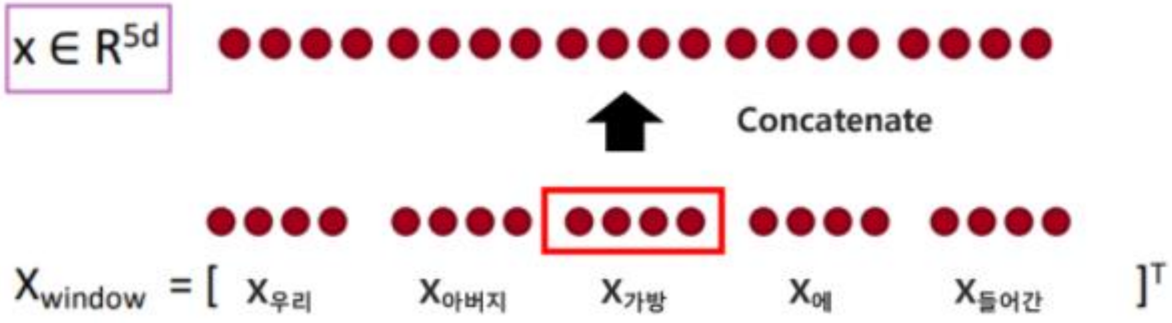
방법1 : window에서 word-vector의 평균 구하기



문제점:각 단어의 위치 정보를 잃어버린다

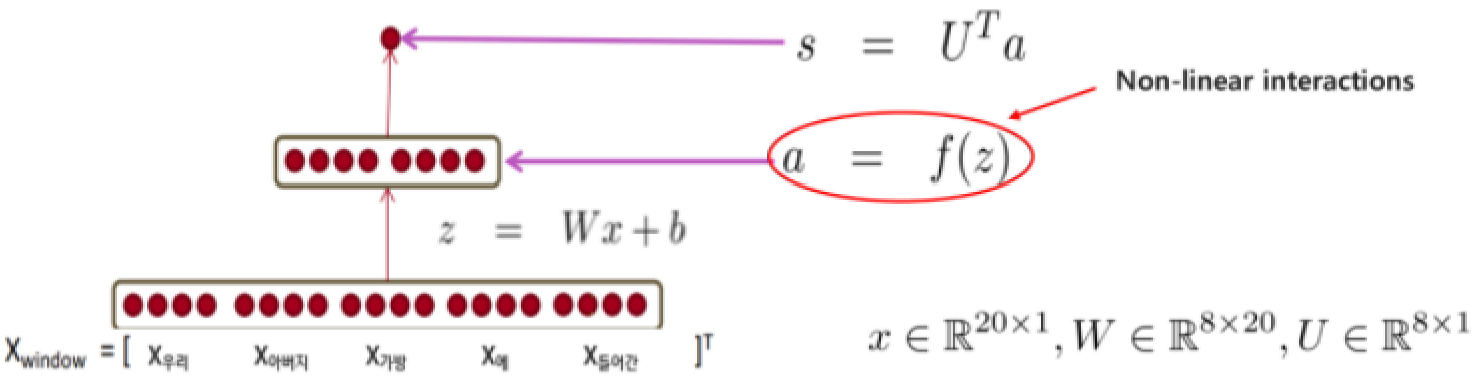
Named Entity Recognition(NER)

방법2: 모든 word-vector concatenate 하기



Named Entity Recognition(NER)

방법2: 모든 word-vector concatenate 하기



이 때 비선형 활성화 함수를 사용해야 단어 간의 비 선형적인 관계를 나타낼 수 있다(interaction term)

예를 들어 첫번째 단어가 museum이고 두번째 단어가 in 이나 or 같은 전치사인 경우에 그 다음 center 단어가 location일 수 있다는 좋은 신호가 된다.

Output s의 경우 확률 값으로 변환을 하거나, 그대로 사용하기도 함

Named Entity Recognition(NER)

Case 1 : 확률 값으로 변환

Output s를 softmax를 사용해서 확률 값으로 변환, cross entropy loss를 사용

$$p(y|\mathrm{s}) = \frac{\exp(\,\mathbf{s}\,)}{\sum_{c=1}^{C} \exp(\,\mathbf{s}\,)} = \mathrm{softmax}(\,\mathbf{s}\,)$$

$$J(\theta) = \frac{1}{N} \sum_{i=1}^{N} -\log p(y|\mathrm{s})$$

Named Entity Recognition(NER)

Case 2: 그대로 사용(unnormalized scores)

Output s를 그대로, Max-margin loss 사용

$X_{window} = [\ x_{museums} \quad x_{in} \quad x_{Paris} \quad x_{are} \quad x_{amazing}\ ]$

가운데 단어가 Location인지 아닌지 분류해보자

Corpus 내 모든 위치에서 학습시키며,
negative-sampling 적용함

- True window: museums in Paris are amazing
- Corrupt window: Not all museums in Paris

Center 단어가 NER location으로 label되지 않은 경우 모두 corrupt window에 해당한다

Min-margin loss

아이디어: true window의 score는 크게
corrupt window의 score는 낮게 나오도록 하자

s = score(museums in Paris are amazing)

s_c = score(Not all museums in Paris)

$$\text{minimize } J = \max(1 + s_c - s, 0)$$

$s_c - s > 0$ 인 경우 학습이 더 진행되어야 하지만,
$s - s_c > 0$인 경우 true window의 score가 더 크기 때문에 학습 중단

여기서 1은 margin으로 true window와 corrupt window의 구분을 명확히 하기 위함

Matrix Calculus

Jacobian Matrix

$f(x) = [f_1(x_1+x_2+\ \dots + x_n), \dots, f_m(x_1+x_2+\ \dots + x_n)]$

Input n개 output m개인 함수에 대한 Jacobian Matrix
편미분으로 구성된 mxn 크기의 matrix

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

연산에서 사용할 유용한 Jacobians

1. $\frac{\partial}{\partial x}(Wx + b) = \mathrm{W}$
2. $\frac{\partial}{\partial b}(Wx + b) = \mathrm{I}$
3. $\frac{\partial}{\partial u}(u^T h) = h^T$

계산은 알아서 해보자!

Back to Neural Net!

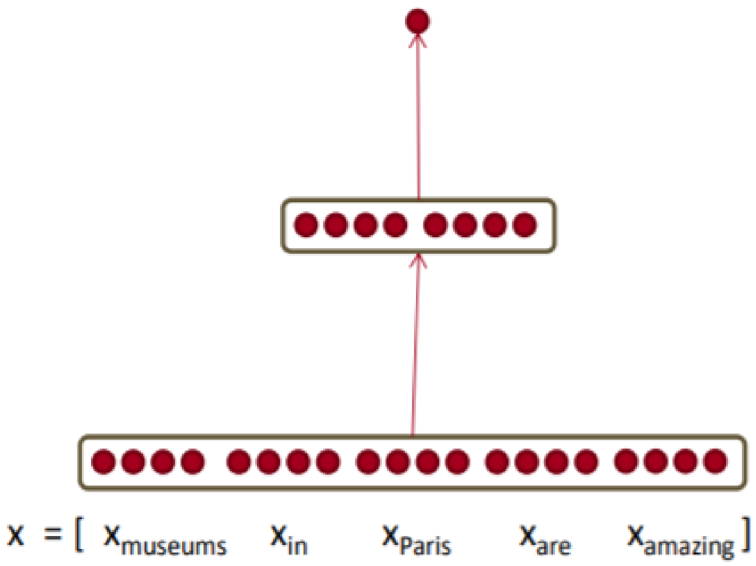
$$s = \boldsymbol{u}^T \boldsymbol{h}$$

$$\boldsymbol{h} = f(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})$$

$$\boldsymbol{h} = f(\boldsymbol{z})$$

$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$$

$$\boldsymbol{x} \quad \text{(input)}$$



Chain rule을 써서
S를 b에 관해 미분해보
자

$$\frac{\partial s}{\partial \boldsymbol{b}} = \frac{\partial s}{\partial \boldsymbol{h}} \quad \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}} \quad \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{b}}$$

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

$$= \boldsymbol{u}^T \circ f'(\boldsymbol{z})$$

Back to Neural Net!

이번에는 W에 관해 미분해보자

$$\frac{\partial s}{\partial \boldsymbol{W}} = \frac{\partial s}{\partial \boldsymbol{h}} \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{W}}$$

$$\frac{\partial s}{\partial \boldsymbol{b}} = \frac{\partial s}{\partial \boldsymbol{h}} \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{b}}$$

앞부분 연산과정은 동일하다!
중복된 연산을 피할 수 있다

Back to Neural Net!

Output shape

$\frac{\partial s}{\partial W}$의 Input nxm개 output 1개 ➔ 1xnm matrix??

NO

$$\frac{\partial s}{\partial \boldsymbol{W}} = \frac{\partial s}{\partial \boldsymbol{h}} \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{W}} = \boldsymbol{\delta} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{W}}$$

$\boldsymbol{\delta}$: 1xn $\quad \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{W}} = \boldsymbol{x}$: mx1

nxm 사이즈가 나올 수 없다

파라미터 업데이트를 위해
우리가 원하는 output 사이즈는
nxm

$$\begin{bmatrix} \frac{\partial s}{\partial W_{11}} & \cdots & \frac{\partial s}{\partial W_{1m}} \\ \vdots & \ddots & \vdots \\ \frac{\partial s}{\partial W_{n1}} & \cdots & \frac{\partial s}{\partial W_{nm}} \end{bmatrix}$$

Back to Neural Net!

$$\frac{\partial s}{\partial \boldsymbol{W}} = \boldsymbol{\delta}^T \quad \boldsymbol{x}^T$$

$$[n \times m] \quad [n \times 1][1 \times m]$$

우리가 원하는 형태 nxm matrix를 얻기 위해 행렬 변환을 하자!

Backpropagation

Matrix Gradients for Neural Nets

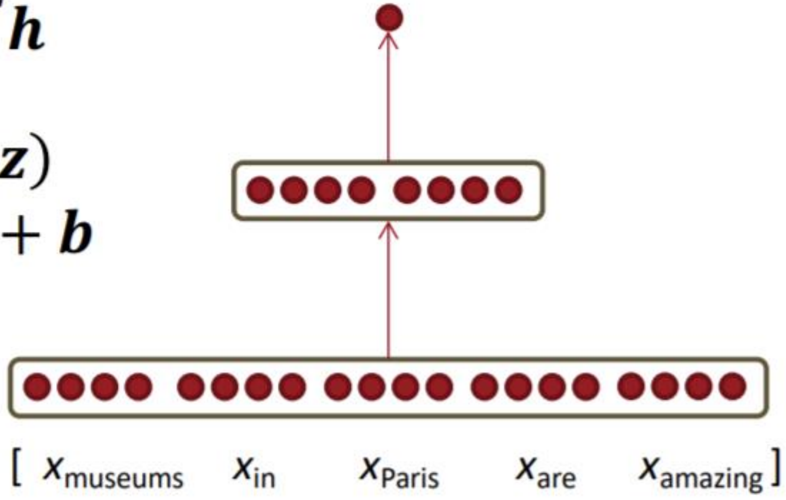
Derivative wrt a weight matrix

- 우리가 계산하고자 하는 것: $\dfrac{\partial s}{\partial \boldsymbol{W}}$
- chain rule을 사용하자!

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

$$\boldsymbol{h} = f(\boldsymbol{z})$$

$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$$



$$\frac{\partial s}{\partial \boldsymbol{W}} = \frac{\partial s}{\partial \boldsymbol{h}} \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{W}}$$

- 앞에서 구한 $\dfrac{\partial s}{\partial \boldsymbol{W}} = \boldsymbol{\delta} \dfrac{\partial \boldsymbol{z}}{\partial \boldsymbol{W}}$로 다시 표현하면,

$$\frac{\partial s}{\partial \boldsymbol{W}} = \boldsymbol{\delta} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{W}} = \boldsymbol{\delta} \frac{\partial}{\partial \boldsymbol{W}} \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b} \qquad \boldsymbol{\delta} = \frac{\partial s}{\partial \boldsymbol{h}} \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}} = \boldsymbol{u}^T \circ f'(\boldsymbol{z})$$

Matrix Gradients for Neural Nets

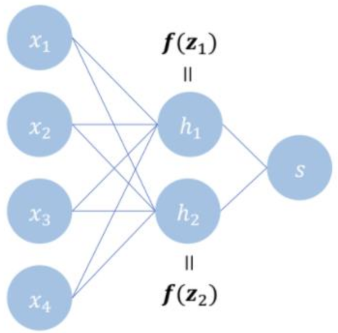
Deriving gradients for backprop

- 이제 아래 식을 W_{ij} 에 대해 미분해보자.

$$\frac{\partial s}{\partial \boldsymbol{W}} = \boldsymbol{\delta} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{W}} = \boldsymbol{\delta} \frac{\partial}{\partial \boldsymbol{W}} \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$$

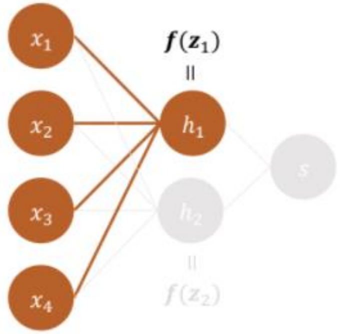
- $\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x}$

$$\begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix}$$



$$z_1 = \sum_{k=1}^{4} w_{1k} x_k$$

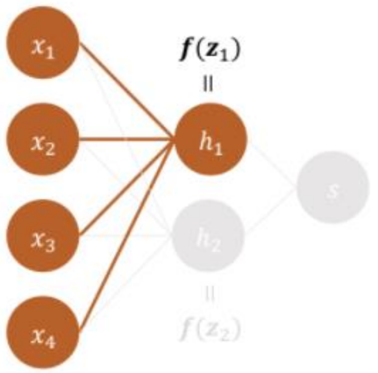
$$\begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix}$$



Matrix Gradients for Neural Nets

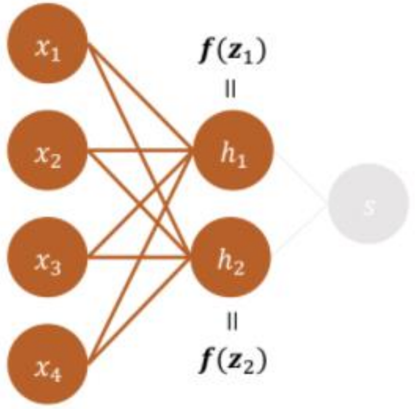
Deriving gradients for backprop

$$\frac{\partial z_1}{\partial w_{1j}} = \frac{\partial \sum_{k=1}^{4} w_{1k} x_k}{\partial w_{1j}} = x_j$$



W_{ij} Only contributes to zi

$$\frac{\partial z_i}{\partial w_{ij}} = \frac{\partial \sum_{k=1}^{4} w_{ik} x_k}{\partial w_{ij}} = x_j$$



Matrix Gradients for Neural Nets

Deriving gradients for backprop

- 그러므로, s를 W_{ij} 에 대해서 미분하면

$$\frac{\partial s}{\partial \boldsymbol{W}} = \boxed{\frac{\partial s}{\partial \boldsymbol{h}} \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}}} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{W}} \qquad \boldsymbol{\delta} = \begin{bmatrix} \delta_1 \\ \delta_2 \end{bmatrix}$$

$$\frac{\partial s}{\partial W_{ij}} = \underbrace{\delta_i}_{\text{Error signal from above}} \underbrace{x_j}_{\text{Local gradient signal}}$$

- 그런데 우리는 모든 W 대한 gradient를 구하고 싶다.
- Overall answer: Outer product

$$\underset{[n \times m]}{\frac{\partial s}{\partial \boldsymbol{W}}} = \underset{[n \times 1]}{\boldsymbol{\delta}^T} \; \underset{[1 \times m]}{\boldsymbol{x}^T}$$

Matrix Gradients for Neural Nets

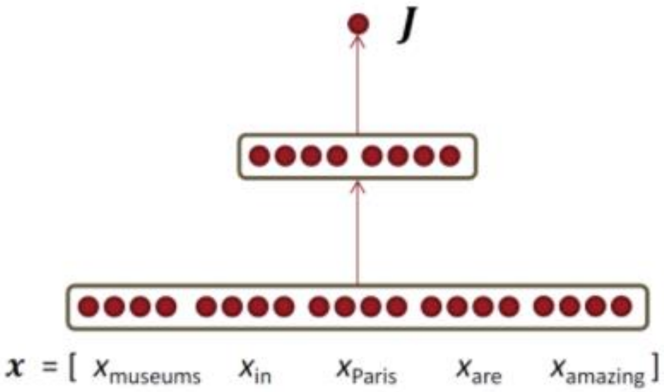
Deriving gradients wrt words for window model

단어 벡터에 도달하고 단어를 업데이트하는 gradient는 각 단어 벡터에 대해 간단히 분할할 수 있다.

$$\nabla_x J = W^T \delta = \delta_{x_{window}}$$

$$x_{window} = [\ x_{museums} \quad x_{in} \quad x_{Paris} \quad x_{are} \quad x_{amazing}\ ]$$

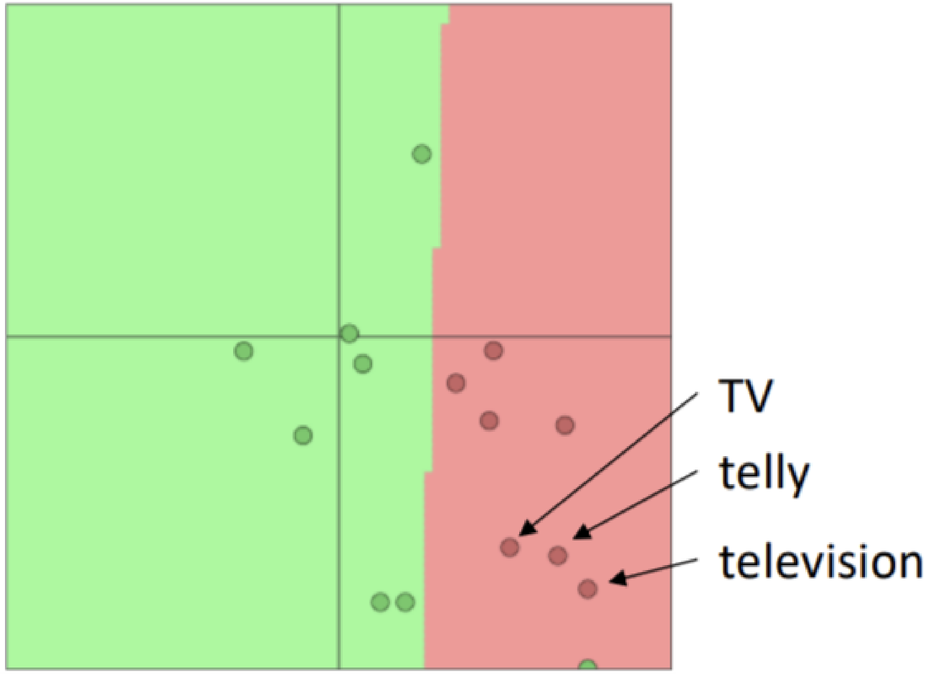
$$\delta_{window} = \begin{bmatrix} \nabla_{x_{museums}} \\ \nabla_{x_{in}} \\ \nabla_{x_{Paris}} \\ \nabla_{x_{are}} \\ \nabla_{x_{amazing}} \end{bmatrix} \in \mathbb{R}^{5d}$$



Matrix Gradients for Neural Nets

A pitfall when retraining word vectors

: single word를 사용하여 영화 감상평에 대한 로지스틱 회귀분류를 훈련하고 있다고 하자.



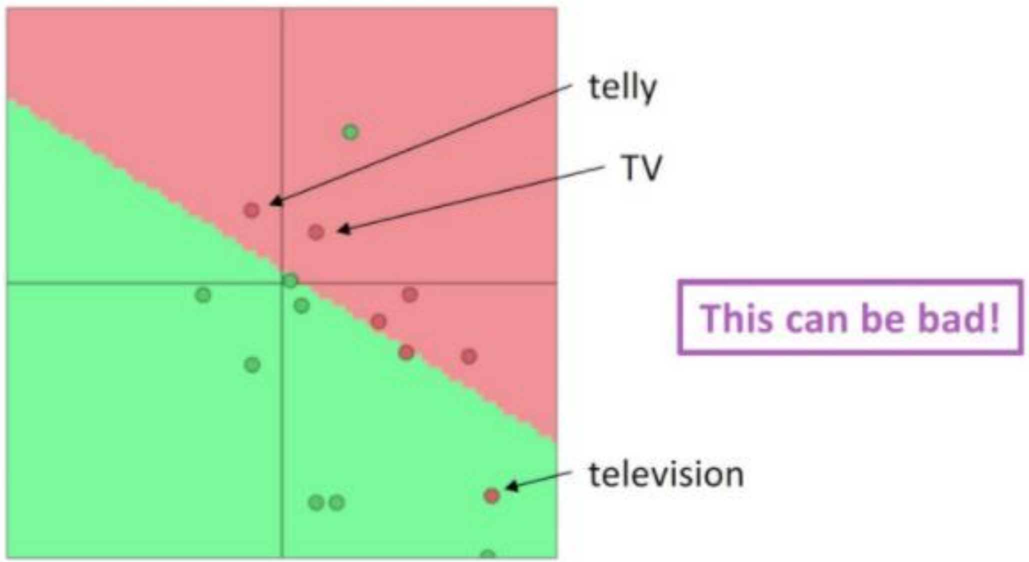
- In the training data : “TV”와 “Telly”가 존재
- In the testing data : “television”만 존재
- pretrained word vector에 세 단어가 비슷한 위치에 존재 :

Question:
word vectors들을 업데이트 하면 어떤 일이 일어날까?

Matrix Gradients for Neural Nets

A pitfall when retraining word vectors

: single word를 사용하여 영화 감상평에 대한 로지스틱 회귀분류를 훈련하고 있다고 하자.



Training set으로 모델을 학습하면
– "TV"와 "telly"는 gradient를 받아 업데이트

Test set에만 있는 "television"은 업데이트되지 않음

Matrix Gradients for Neural Nets

So what should we do?

“pre-trained” word vectors를 사용해야 할까?

- Almost always, yes!
- pre-trained data는 방대한 양에 대해 사람들이 이미 학습을 시켰다
- 따라서 TV, telly, television처럼 train set포함의 유무와 관계 없이 어느 정도 단어간 관계가 형성된다.
- 그러나 데이터가 매우 많다면(100s millions of words of data) 처음부터 학습을 시켜도 무관

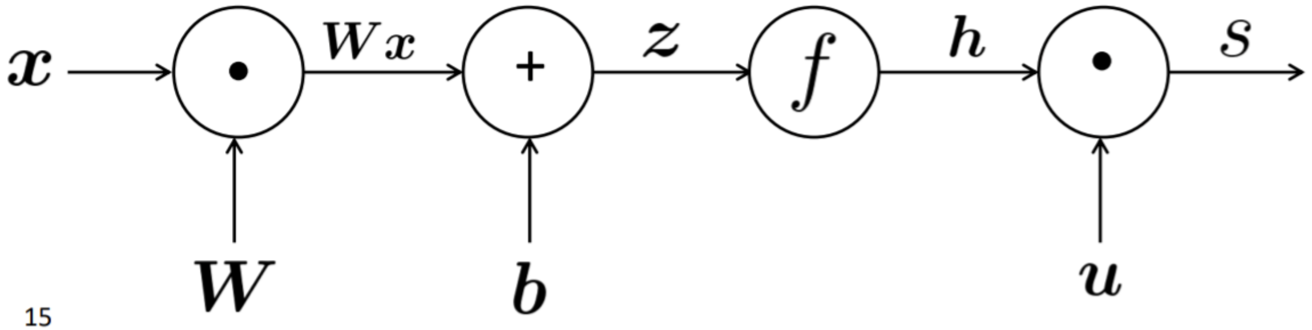
Fine-tune을 해야할까?

- 만약 적은 train set(100,000개 미만)을 갖고 있다면 fine-tuning하지 말 것

- 많은 train set(100만 개 이상)을 갖고 있다면 fine-tune을 하는게 성능 향상에 도움이 된다.

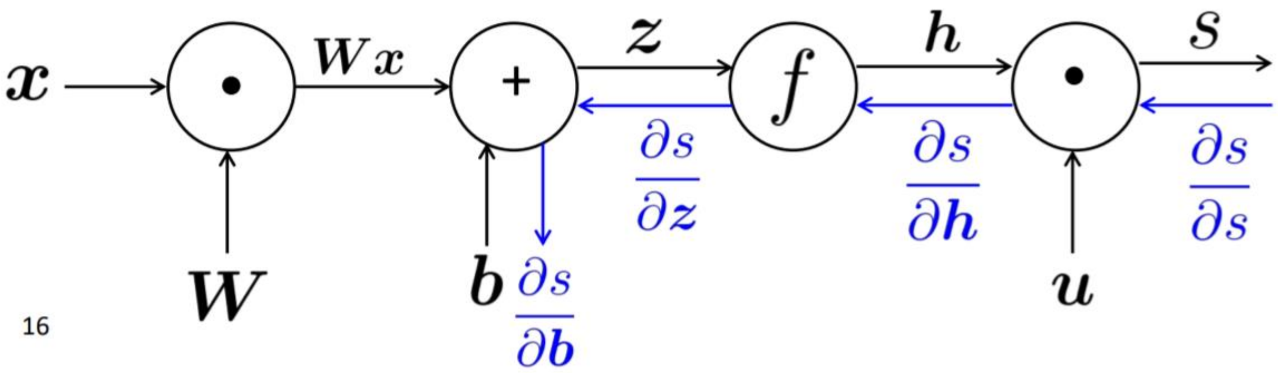
Computation Graphs and Backpropagation

Forward propagation



15

Back propagation



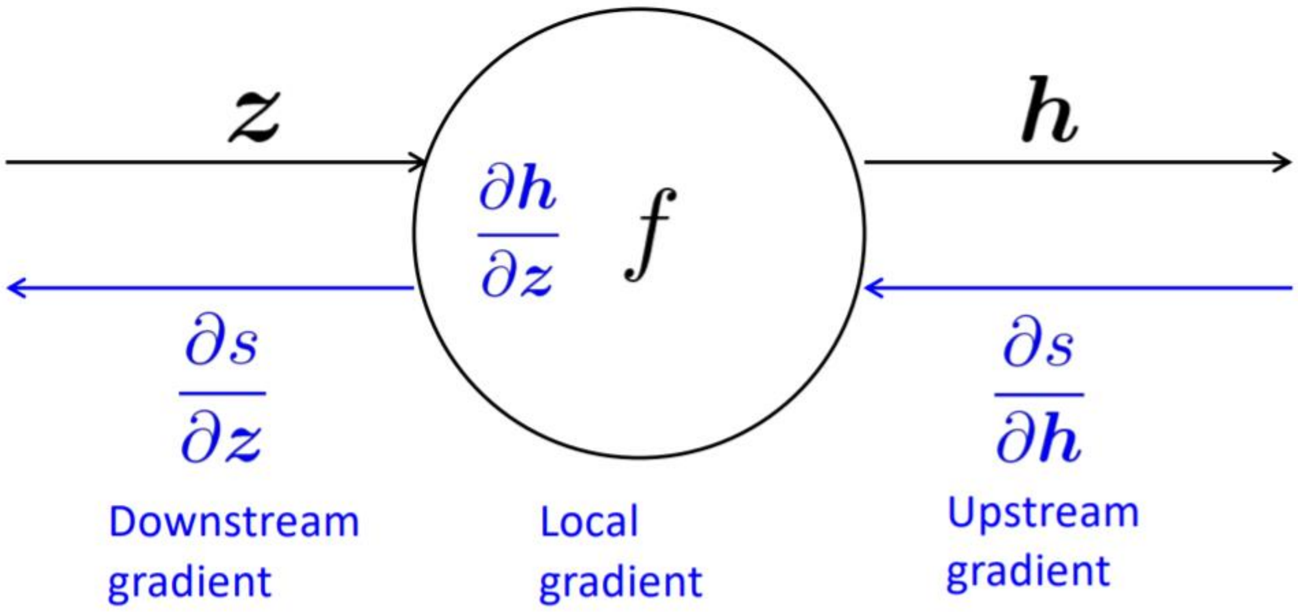
16

$$s = \boldsymbol{u}^T \boldsymbol{h}$$
$$\boldsymbol{h} = f(\boldsymbol{z})$$
$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$$
$$\boldsymbol{x} \quad \text{(input)}$$

Computation Graphs and Backpropagation

Single Node – scalar function example

- 노드가 "upstream gradient"를 받는다.
- 목표는 올바른 "downstream gradient" 을 전달하는 것이다. 각 노드는 local gradient 을 갖고 있다.

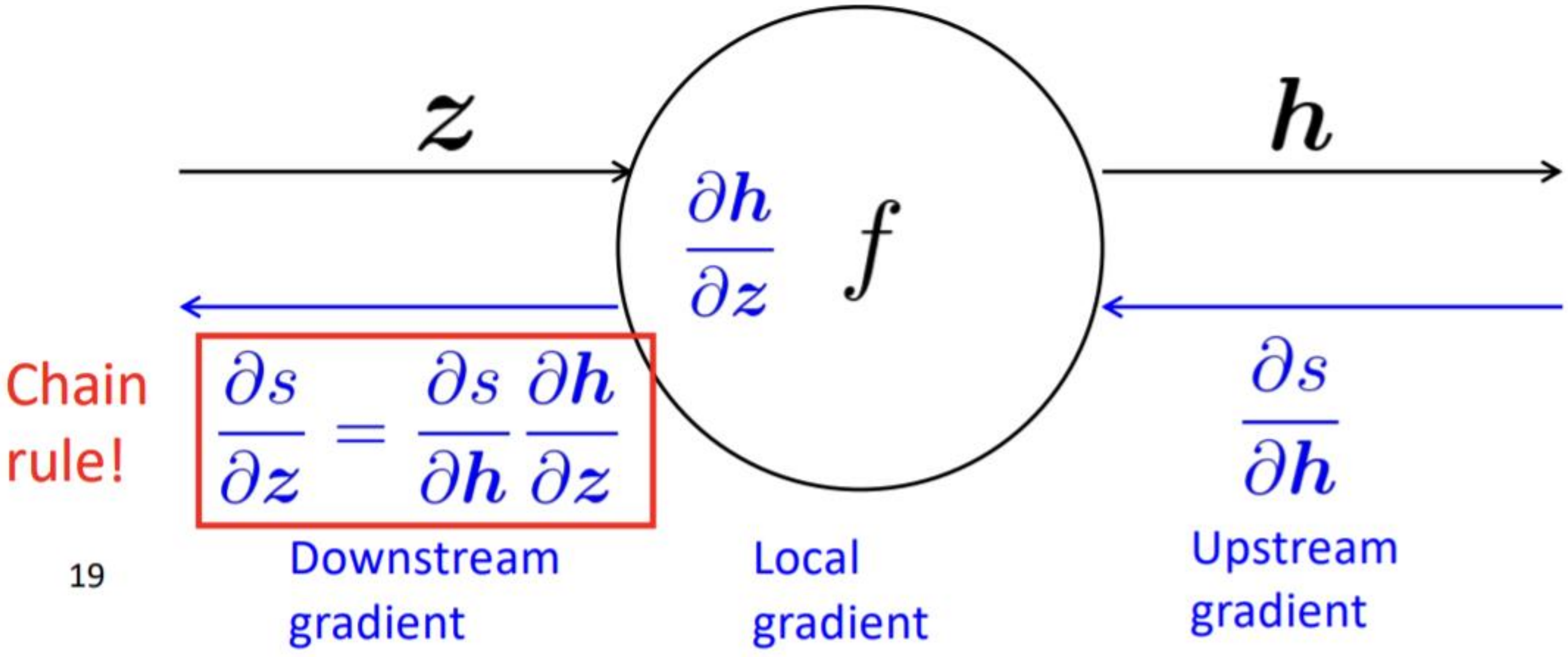


Computation Graphs and Backpropagation

Single Node – scalar function example

- chain rule

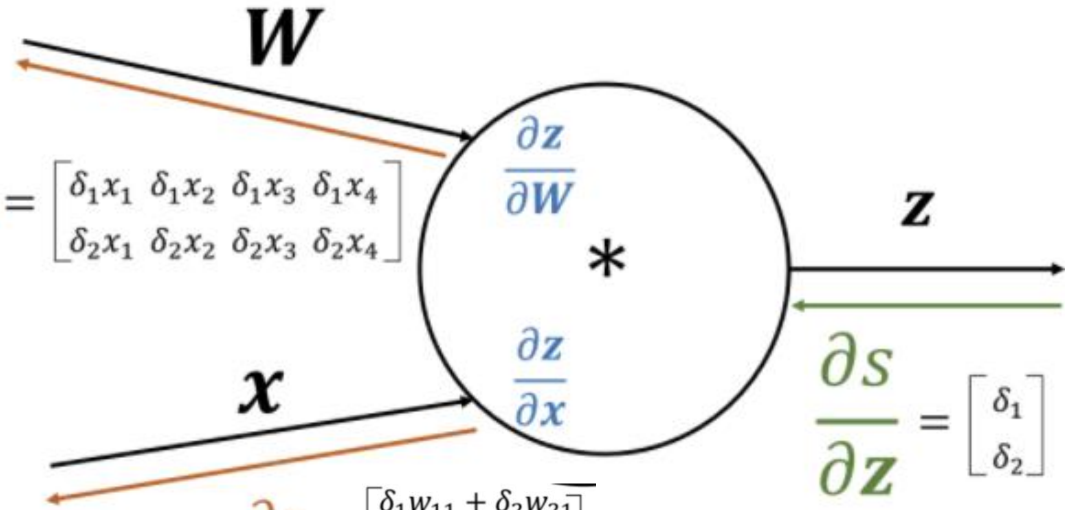
[downstream gradient] = [upstream gradient] x [local gradient]



Computation Graphs and Backpropagation

Single Node : Multiple Input -Matrix-Vector example

$$z = Wx \qquad \begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix}$$



$$\frac{\partial s}{\partial W} = \begin{bmatrix} \delta_1 x_1 & \delta_1 x_2 & \delta_1 x_3 & \delta_1 x_4 \\ \delta_2 x_1 & \delta_2 x_2 & \delta_2 x_3 & \delta_2 x_4 \end{bmatrix}$$

$$\frac{\partial s}{\partial z} = \begin{bmatrix} \delta_1 \\ \delta_2 \end{bmatrix}$$

$$\frac{\partial s}{\partial x} = \begin{bmatrix} \delta_1 w_{11} + \delta_2 w_{21} \\ \delta_1 w_{12} + \delta_2 w_{22} \\ \delta_1 w_{13} + \delta_2 w_{23} \\ \delta_1 w_{14} + \delta_2 w_{24} \end{bmatrix}$$

$$\frac{\partial s}{\partial \boldsymbol{W}} = \frac{\partial s}{\partial \boldsymbol{z}} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{W}} = \boldsymbol{\delta} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{W}}$$

$$\frac{\partial s}{\partial w_{ij}} = \sum_{k=1}^{2} \delta_k \frac{\partial z_k}{\partial w_{ij}} = \delta_i x_j$$

$$\frac{\partial s}{\partial \boldsymbol{W}} = \boldsymbol{\delta} \cdot \boldsymbol{x}^{\boldsymbol{T}} = \begin{bmatrix} \delta_1 \\ \delta_2 \end{bmatrix} \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \end{bmatrix}$$

$$\frac{\partial s}{\partial \boldsymbol{x}} = \frac{\partial s}{\partial \boldsymbol{z}} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}} = \boldsymbol{\delta} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}}$$

$$\frac{\partial s}{\partial x_j} = \sum_{k=1}^{2} \delta_k \frac{\partial z_k}{\partial x_j} = \delta_1 w_{1j} + \delta_2 w_{2j}$$

$$\frac{\partial s}{\partial \boldsymbol{x}} = \boldsymbol{W}^{\boldsymbol{T}} \cdot \boldsymbol{\delta} = \begin{bmatrix} w_{11} & w_{21} \\ w_{12} & w_{22} \\ w_{13} & w_{23} \\ w_{14} & w_{24} \end{bmatrix} \begin{bmatrix} \delta_1 \\ \delta_2 \end{bmatrix}$$

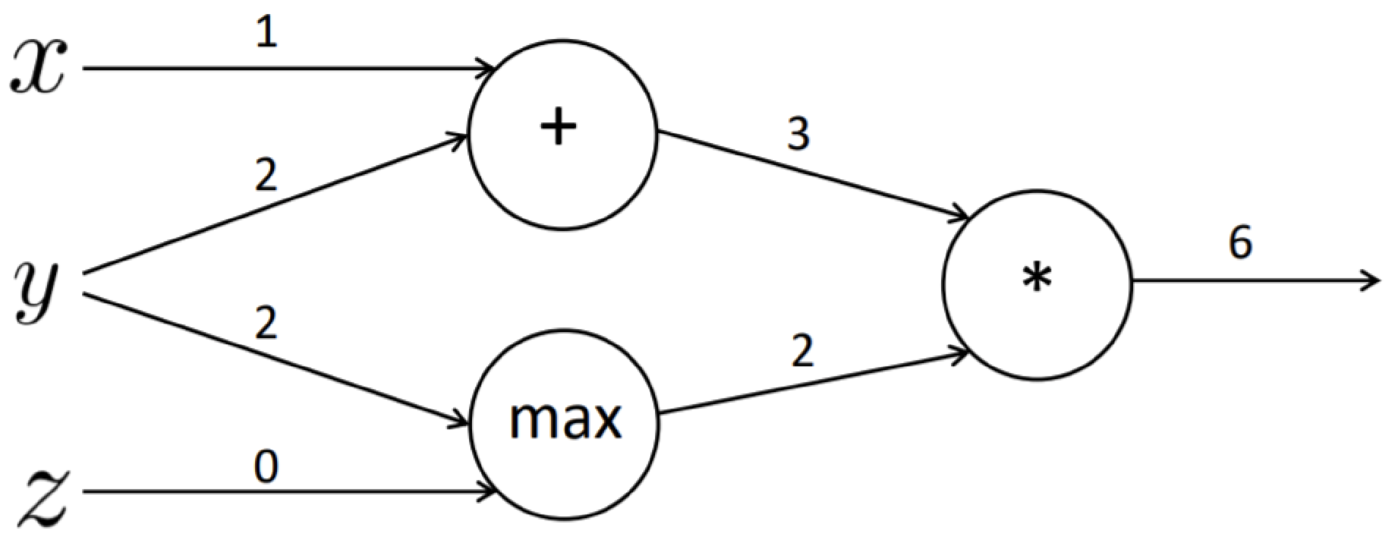
Computation Graphs and Backpropagation

An Example

$$f(x, y, z) = (x + y)\max(y, z)$$
$$x = 1, y = 2, z = 0$$

$$a = x + y$$
$$b = \max(y, z)$$
$$f = ab$$

Forward prop steps



Computation Graphs and Backpropagation

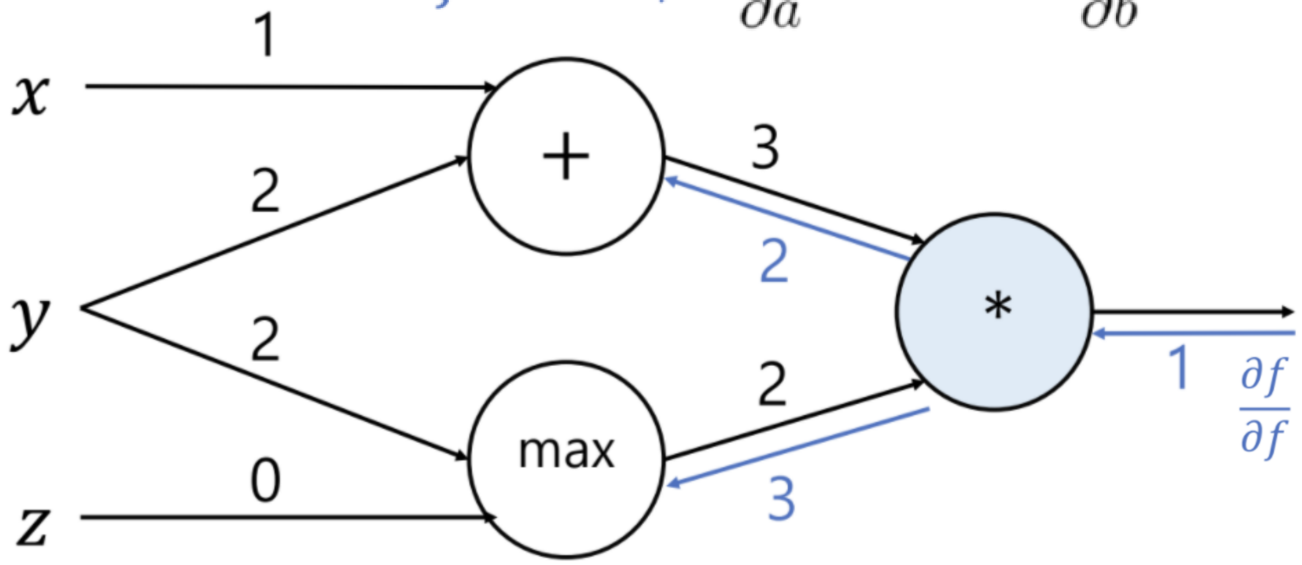
An Example

$$f(x, y, z) = (x + y)\max(y, z)$$
$$x = 1, y = 2, z = 0$$

$$a = x + y$$
$$b = \max(y, z)$$
$$f = ab$$

Back prop steps

$f = ab$, $\frac{\partial f}{\partial a} = b = 2 \quad \frac{\partial f}{\partial b} = a = 3$



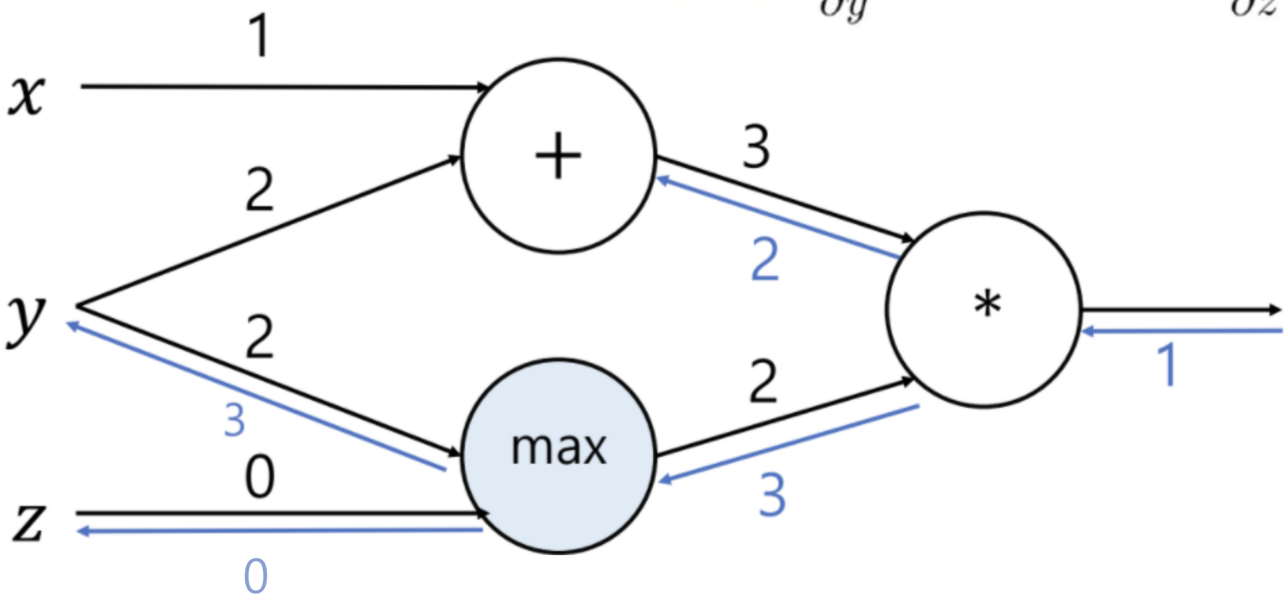
An Example

$$f(x, y, z) = (x + y)\max(y, z)$$
$$x = 1, y = 2, z = 0$$

$$a = x + y$$
$$b = \max(y, z)$$
$$f = ab$$

Back prop steps

$b = \max(y,z)$, $\frac{\partial b}{\partial y} = \mathbf{1}(y > z) = 1$ $\frac{\partial b}{\partial z} = \mathbf{1}(z > y) = 0$



Computation Graphs and Backpropagation

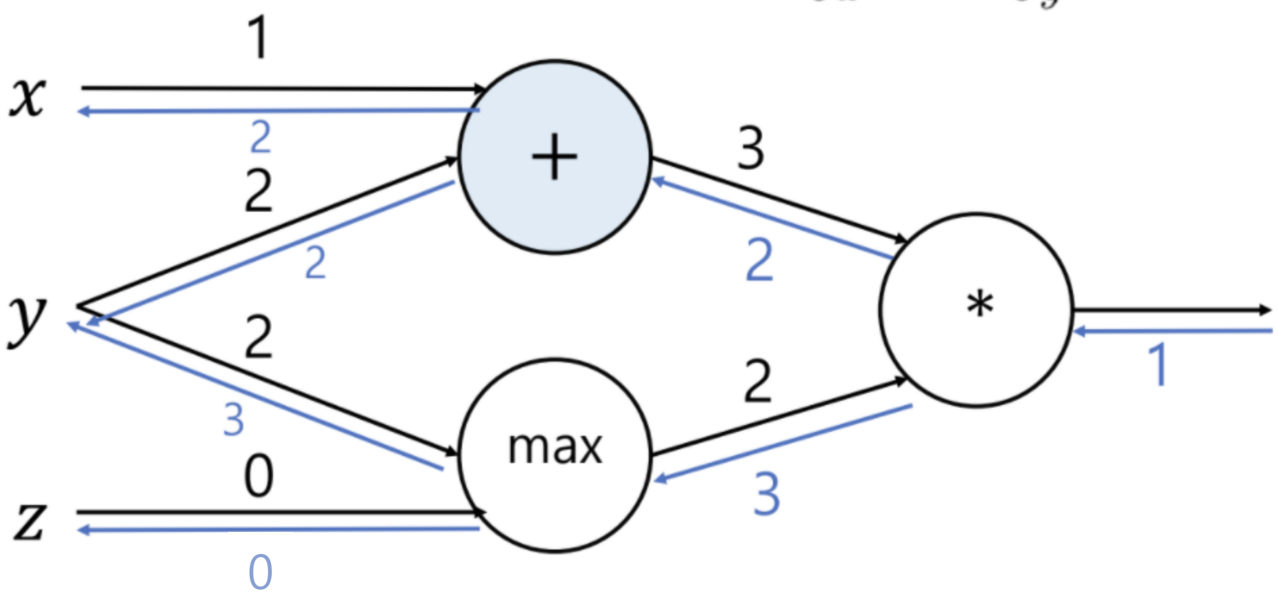
An Example

$$f(x, y, z) = (x + y)\max(y, z)$$
$$x = 1, y = 2, z = 0$$

$$a = x + y$$
$$b = \max(y, z)$$
$$f = ab$$

Back prop steps

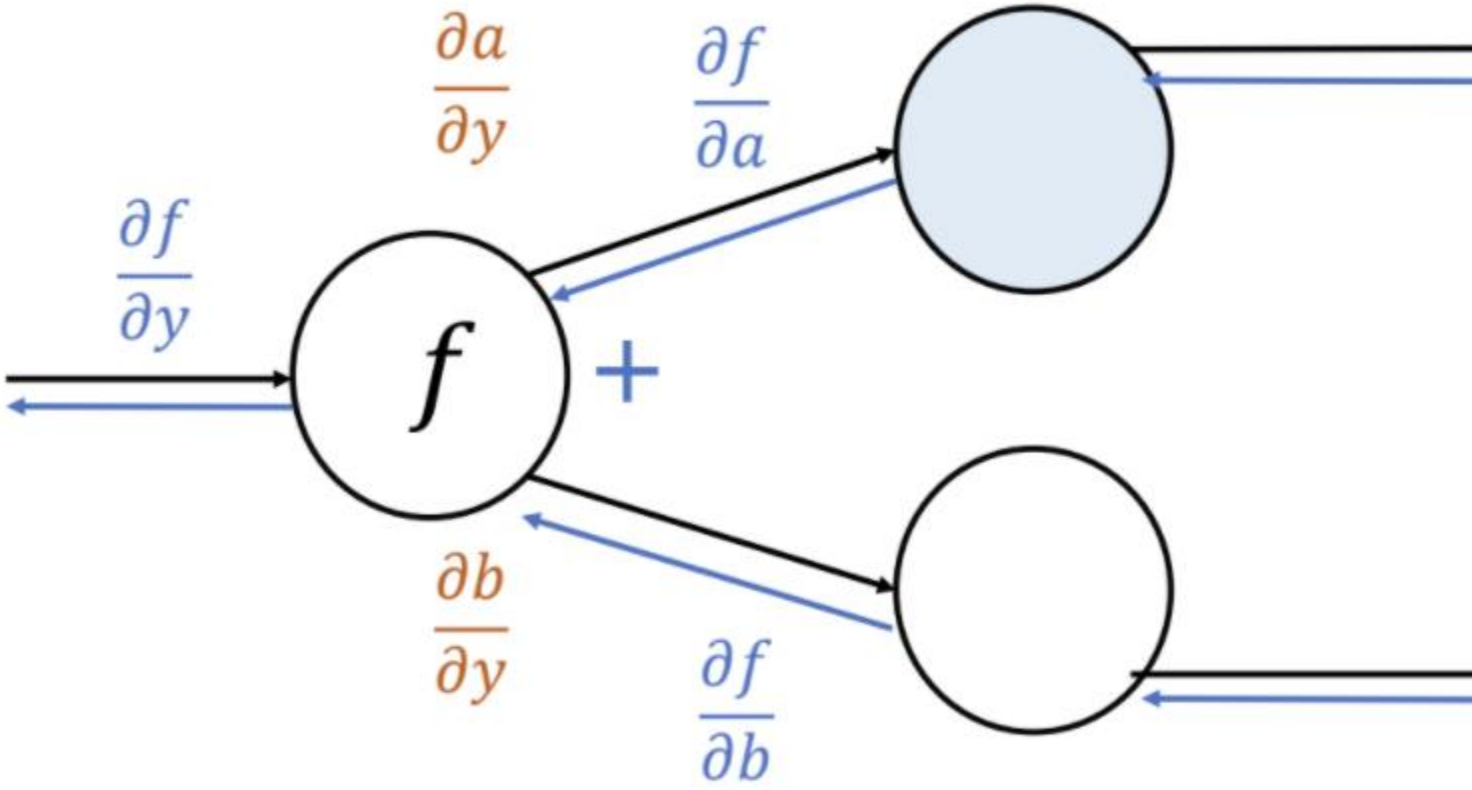
$$a = x + y, \quad \frac{\partial a}{\partial x} = 1 \quad \frac{\partial a}{\partial y} = 1$$



Computation Graphs and Backpropagation

Gradients sum at outward branches

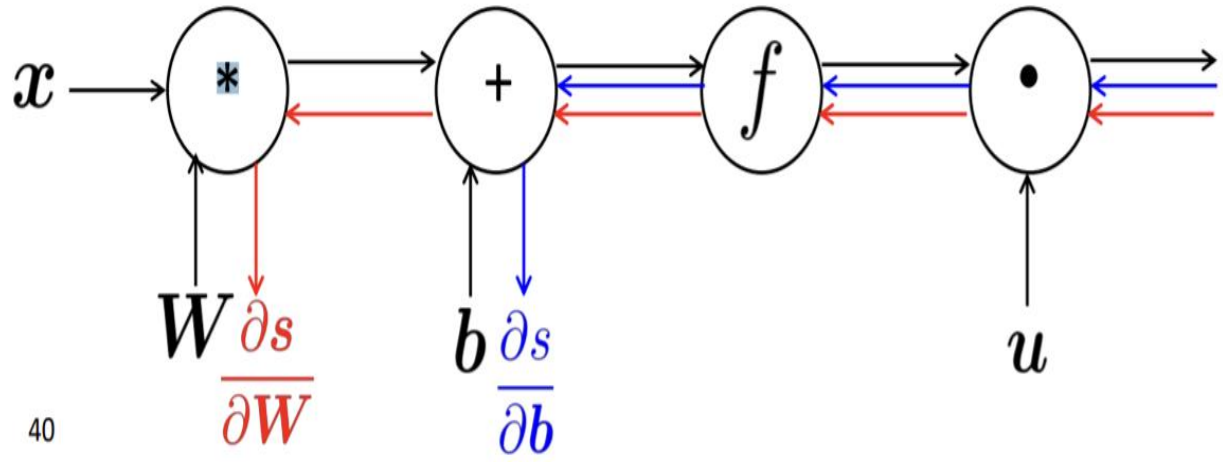
$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial a}\frac{\partial a}{\partial y} + \frac{\partial f}{\partial b}\frac{\partial b}{\partial y}$$



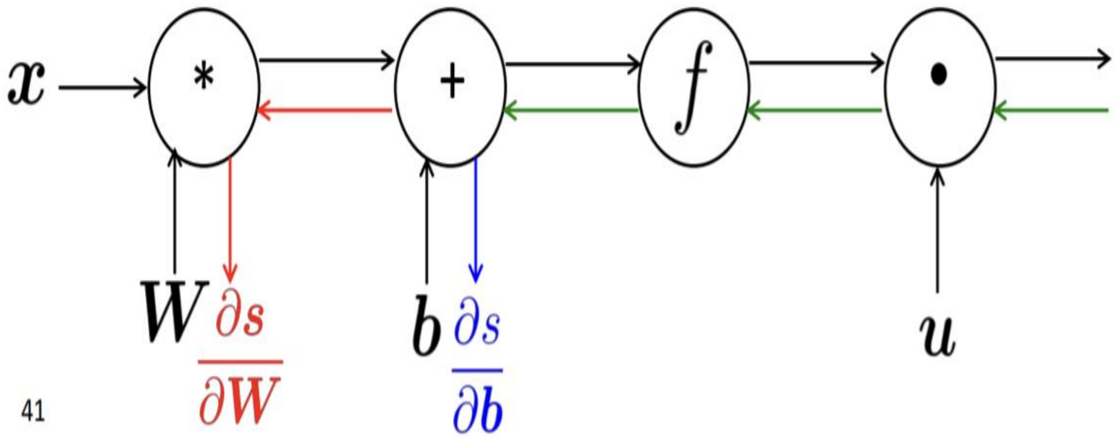
Computation Graphs and Backpropagation

Efficiency: compute all gradients at once

Incorrect way of doing backprop:

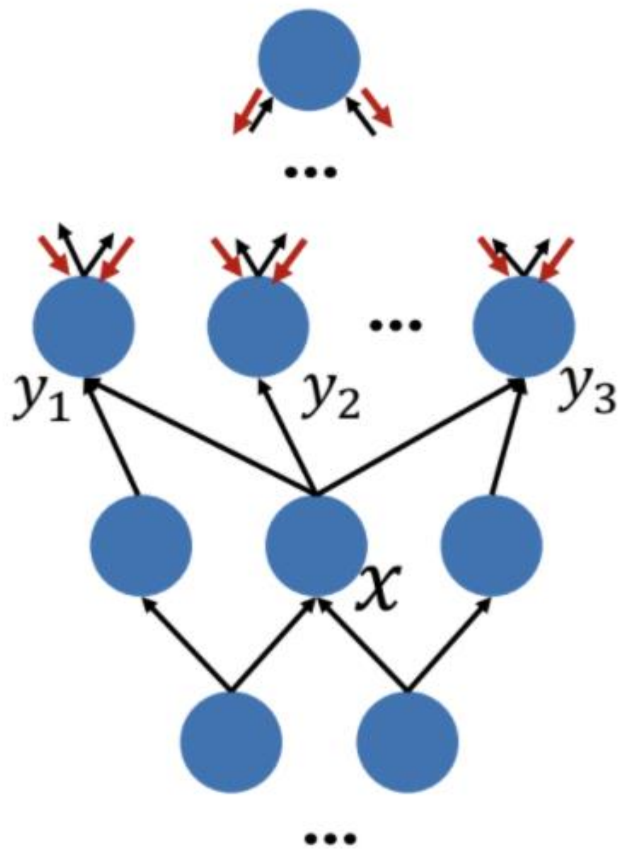


Correct way:



Computation Graphs and Backpropagation

Back-Prop in General Computation Graph



1. Fprop: visit nodes in topological sort order - Compute value of node given predecessors

2. Bprop:
- initialize output gradient = 1
- visit nodes in reverse order: Compute gradient wrt each node using gradient wrt successors

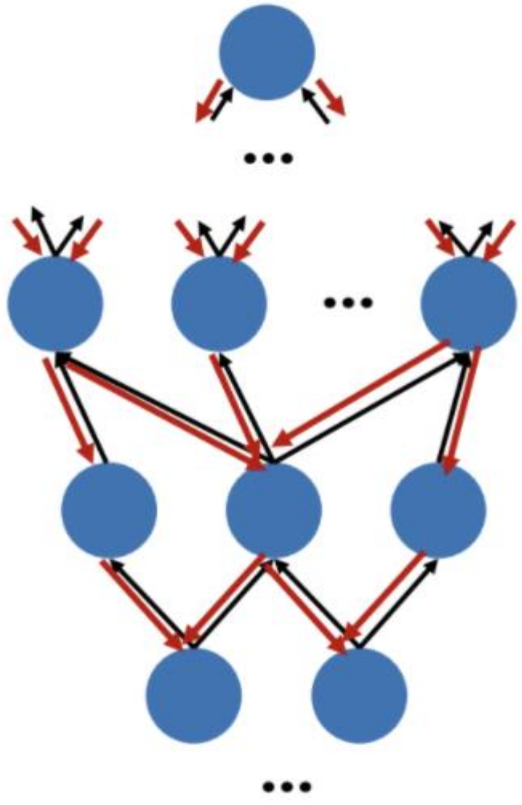
$\{y_1, y_2, \ldots y_n\}$ = successors of x

$$\frac{\partial z}{\partial x} = \sum_{i=1}^{n} \frac{\partial z}{\partial y_i} \frac{\partial y_i}{\partial x}$$

In general our nets have regular layer-structure and so we can use matrices and Jacobians…

Computation Graphs and Backpropagation

Back-Prop in General Computation Graph



- The gradient computation can be automatically inferred from the symbolic expression of the fprop
- Each node type needs to know how to compute its output and how to compute the gradient wrt its inputs given the gradient wrt its output
- Modern DL frameworks (Tensorflow, PyTorch, etc.) do backpropagation for you but mainly leave layer/node writer to hand-calculate the local derivative

Computation Graphs and Backpropagation

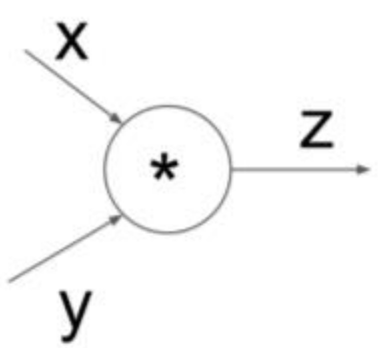
Backprop Implementations

```python
class ComputationalGraph(object):
    #...
    def forward(inputs):
        # 1. [pass inputs to input gates...]
        # 2. forward the computational graph:
        for gate in self.graph.nodes_topologically_sorted():
            gate.forward()
        return loss # the final gate in the graph outputs the loss
    def backward():
        for gate in reversed(self.graph.nodes_topologically_sorted()):
            gate.backward() # little piece of backprop (chain rule applied)
        return inputs_gradients
```

Computation Graphs and Backpropagation

Backprop Implementations

Forward / backward



(x,y,z are scalars)

```python
class MultiplyGate(object):
    def forward(x,y):
        z = x*y
        self.x = x # must keep these around!
        self.y = y
        return z
    def backward(dz):
        dx = self.y * dz # [dz/dx * dL/dz]
        dy = self.x * dz # [dz/dy * dL/dz]
        return [dx, dy]
```

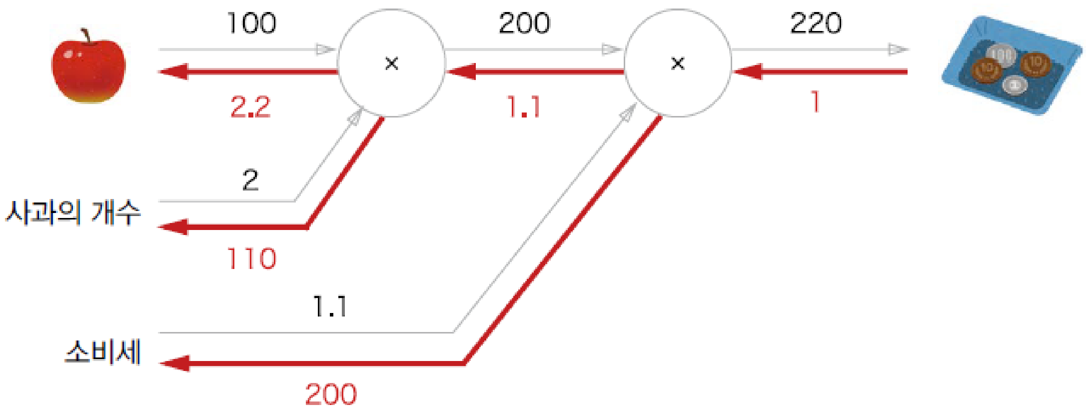
Add의 경우
z = x+y

Max의 경우
z = max(x, y)

Computation Graphs and Backpropagation

Backprop Implementations

단순한 layer 구현하기



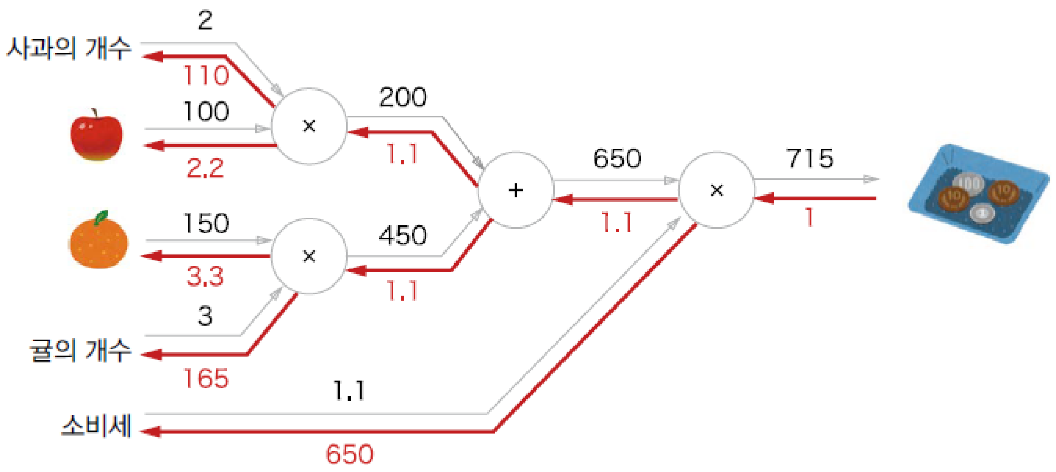

```
apple = 100
apple_num = 2
tax = 1.1

# 계층들
mul_apple_layer = MulLayer()
mul_tax_layer = MulLayer()

# 순전파
apple_price = mul_apple_layer.forward(apple, apple_num)
price = mul_tax_layer.forward(apple_price, tax)

print('%d' % price)

# 역전파
dprice = 1
dapple_price, dtax = mul_tax_layer.backward(dprice)
dapple, dapple_num = mul_apple_layer.backward(dapple_price)

print("%.1f, %d, %d" % (dapple, dapple_num, dtax))
```

```
220
2.2, 110, 200
```

Computation Graphs and Backpropagation

Backprop Implementations

단순한 layer 구현하기




```
apple = 100
apple_num = 2
orange = 150
orange_num = 3
tax = 1.1

# 계층들
mul_apple_layer = MulLayer()
mul_orange_layer = MulLayer()
add_apple_orange_layer = AddLayer()
mul_tax_layer = MulLayer()

# 순전파
apple_price = mul_apple_layer.forward(apple, apple_num)
orange_price = mul_orange_layer.forward(orange, orange_num)
all_price = add_apple_orange_layer.forward(apple_price, orange_price)
price = mul_tax_layer.forward(all_price, tax)

# 역전파
dprice = 1
dall_price, dtax = mul_tax_layer.backward(dprice)
dapple_price, dorange_price = add_apple_orange_layer.backward(dall_price)
dorange, dorange_num = mul_orange_layer.backward(dorange_price)
dapple, dapple_num = mul_apple_layer.backward(dapple_price)

print('%d' % price)
print("%d, %.1f, %.1f, %d, %d" % (dapple_num, dapple, dorange, dorange_num, dtax))
```

```
715
49500, 990.0, 660.0, 33000, 650
```

Computation Graphs and Backpropagation

Gradient Checking: Numerical Gradient

For small h (≈ 1e−4), $f'(x) \approx \dfrac{f(x+h) - f(x-h)}{2h}$

- 미분 공식을 통해 gradient를 쉽게 계산 가능
- 적용할 때마다 f를 새롭게 계산해야 하기 때문에 computational graph 방식에 비해 매우 느림
- 따라서 일부에 대해서만 gradient가 제대로 계산되었는지 확인하는 용도

Snippet 2.1

```
def eval_numerical_gradient(f, x):
  """
  a naive implementation of numerical gradient of f at x
  - f should be a function that takes a single argument
  - x is the point (numpy array) to evaluate the gradient
  at
  """

  fx = f(x) # evaluate function value at original point
  grad = np.zeros(x.shape)
  h = 0.00001

  # iterate over all indexes in x
  it = np.nditer(x, flags=['multi_index'],
                 op_flags=['readwrite'])
  while not it.finished:

    # evaluate function at x+h
    ix = it.multi_index
    old_value = x[ix]
    x[ix] = old_value + h # increment by h
    fxh_left = f(x) # evaluate f(x + h)
    x[ix] = old_value - h # decrement by h
    fxh_right = f(x) # evaluate f(x - h)
    x[ix] = old_value # restore to previous value (very
        important!)

    # compute the partial derivative
    grad[ix] = (fxh_left - fxh_right) / (2*h) # the slope
    it.iternext() # step to next dimension
  return grad
```

Tips and Tricks of Neural Network

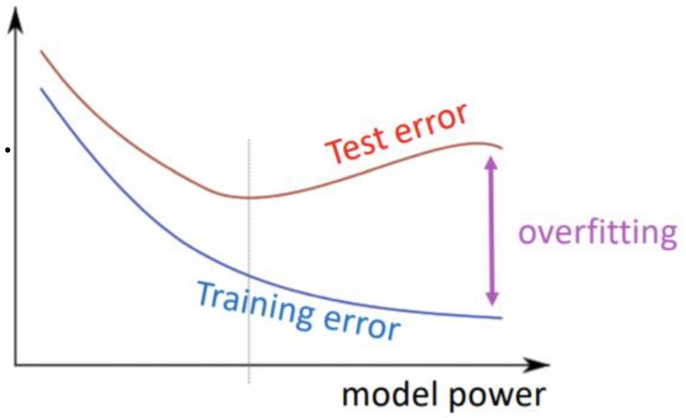
Tips and Tricks of Neural Network

Regularization

$$J(\theta) = \frac{1}{N}\sum_{i=1}^{N} -\log\left(\frac{e^{f_{y_i}}}{\sum_{c=1}^{C} e^{f_c}}\right) \qquad J(\theta) = \frac{1}{N}\sum_{i=1}^{N} -\log\left(\frac{e^{f_{y_i}}}{\sum_{c=1}^{C} e^{f_c}}\right) \boxed{+\lambda\sum_{k}\theta_k^2}$$

손실 함수를 그냥 쓰면 training set에 과적합 될 수 있다

- 이를 방지하기 위해 실제 full loss function에는 모든 parameter에 대해 Regularization을 한다
- 보통 L2 Regularization을 많이 이용
- feature가 많은 깊은 모델(e.g. deel model)일수록 뛰어난 효과를 보인다.



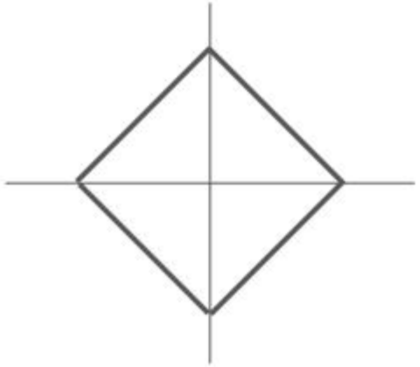
Tips and Tricks of Neural Network

Regularization

L1 Regularization / L2 Regularization

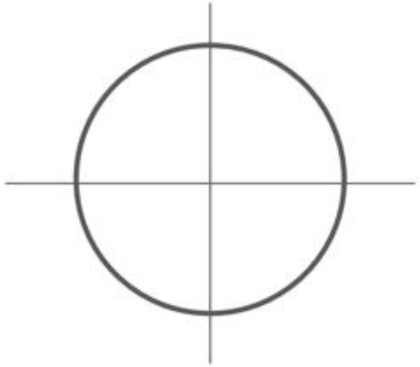
L1 (Manhattan) distance

$$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$



L2 (Euclidean) distance

$$d_2(I_1, I_2) = \sqrt{\sum_p \left(I_1^p - I_2^p\right)^2}$$



L1 Regularization

$$\text{Cost} = \sum_{i=0}^{N} (y_i - \sum_{j=0}^{M} x_{ij} W_j)^2 + \lambda \sum_{j=0}^{M} |W_j|$$

L2 Regularization

$$\text{Cost} = \underbrace{\sum_{i=0}^{N} (y_i - \sum_{j=0}^{M} x_{ij} W_j)^2}_{\text{Loss function}} + \lambda \underbrace{\sum_{j=0}^{M} W_j^2}_{\text{Regularization Term}}$$

Tips and Tricks of Neural Network

Vectorization

- looping over word vectors
- versus
- concatenating them all into one large matrix and then multiplying the softmax weights with that matrix

1000 loops, best of 3: 639 µs per loop
10000 loops, best of 3: 53.8 µs per loop

The (10x) faster method is using a C x N matrix • Always try to use vectors and matrices rather than for loops!

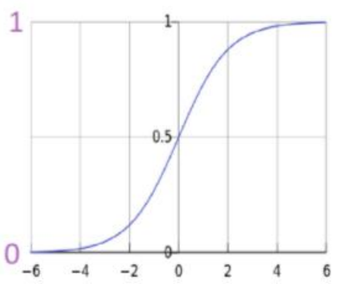
```
from numpy import random
N = 500 # number of windows to classify
d = 300 # dimensionality of each window
C = 5 # number of classes
W = random.rand(C,d)
wordvectors_list = [random.rand(d,1) for i in range(N)]
wordvectors_one_matrix = random.rand(d,N)

%timeit [W.dot(wordvectors_list[i]) for i in range(N)]
%timeit W.dot(wordvectors_one_matrix)
```

Tips and Tricks of Neural Network

Non-linearities: The starting points

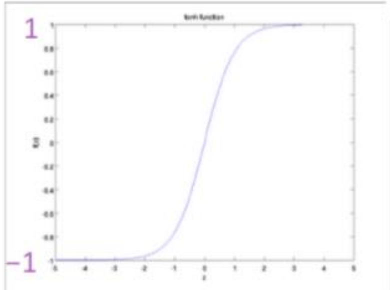
logistic (“sigmoid”)

$$f(z) = \frac{1}{1+\exp(-z)}.$$



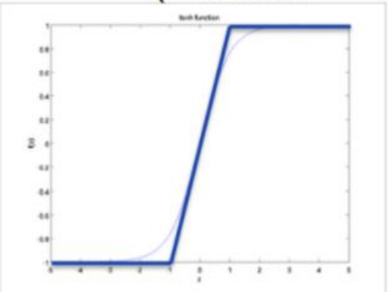
tanh

$$f(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}},$$



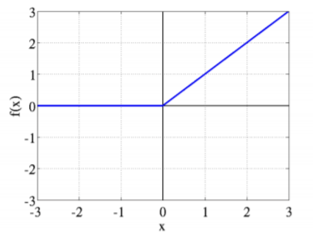
hard tanh

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



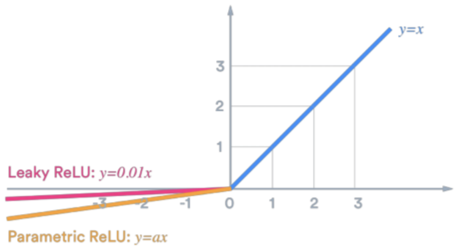
ReLU (rectified linear unit) hard tanh

$$\mathrm{rect}(z) = \max(z,0)$$



Leaky ReLU

Parametric ReLU



tanh is just a rescaled and shifted sigmoid
(2 as steep, [−1,1]):

Sigmoid와 tanh는 여전히 많이 쓰이지만 deep한 구조에서는 쓰이지 않음
> why? : vanishing gradient

- Deep network에서는 ReLU (좌)를 항상 첫번째로 고려할 것
- 간단할뿐만 아니라 성능도 좋게 나옴
- Leaky ReLU, Parametric ReLU, GeLU 등 다양한 Variation이 존재

Tips and Tricks of Neural Network

Parameter Initialization

일반적으로 작은 random value로 parameter의 초기값을 줘야함
- ✓ 히든 레이어의 bias term은 0으로 초기값을 줘야함
- ✓ 다른 모든 weight들은 Uniform(–r, r)에서 sampling할 것
- ✓ r은 매우 작거나 매우 크면 안됨

Xavier initialization을 자주 이용하기도 함
- ✓ previous layer size와 next layer size에 맞게 weight variance를 조절

$$\mathrm{Var}(W_i) = \frac{2}{n_{\text{in}} + n_{\text{out}}}$$

Tips and Tricks of Neural Network

Optimizer

보통은 그냥 SGD를 써도 최적화가 잘 된다.

- ✓ 하지만 더 좋은 값을 얻기 위해서는 learning rate를 반드시 튜닝해야 한다.

복잡한 신경망을 학습할 때는 "adaptive" optimizer가 성능이 좋음

- ✓ Adaptive optimizer는 gradient 정보를 계속 축적하며 이를 통해 gradient를 조절하는 방법이다.
  - Adagrad
  - RMSprop
  - Adam <- A fairly good, safe place to begin in many cases
  - SparseAdam
  - …

Tips and Tricks of Neural Network

Learning Rates

– 0.001정도의 일정한 learning rate를 일반적으로 씀
It must be order of magnitude right – try powers of 10
- Too big: model may diverge or not converge
- Too small: your model may not have trained by the deadline

– 학습이 진행될수록 learning rate를 감소시키는게 보통 성능이 뛰어남
- By hand: k epoch마다 반으로 줄이기
- By a formula: k epoch마다 lr = $lr_0 e^{-kt}$ 와 같이 증가시키기
- cyclic learning rate와 같이 fancy한 방법들이 존재