LIKE LION

5강

인공신경망, 어떻게 학습 시키는 거죠?

WHATEVER YOU WANT, MAKE IT REAL.

강사 윤영석

- Multi-Layer Perceptron MLP
  - Single-Layer Perceptron (Linear model) 의 확장
  - 기존의 선형 모델로 해결할 수 없는 문제들을 해결하기 위해 제안
  - 각 층의 사이에 nonlinear activation function을 사용해 모델의 표현력 높임

- Forward Pass
  - 벡터와 벡터의 내적으로 표현되던 선형 모델을 행렬과 행렬의 곱으로 확장

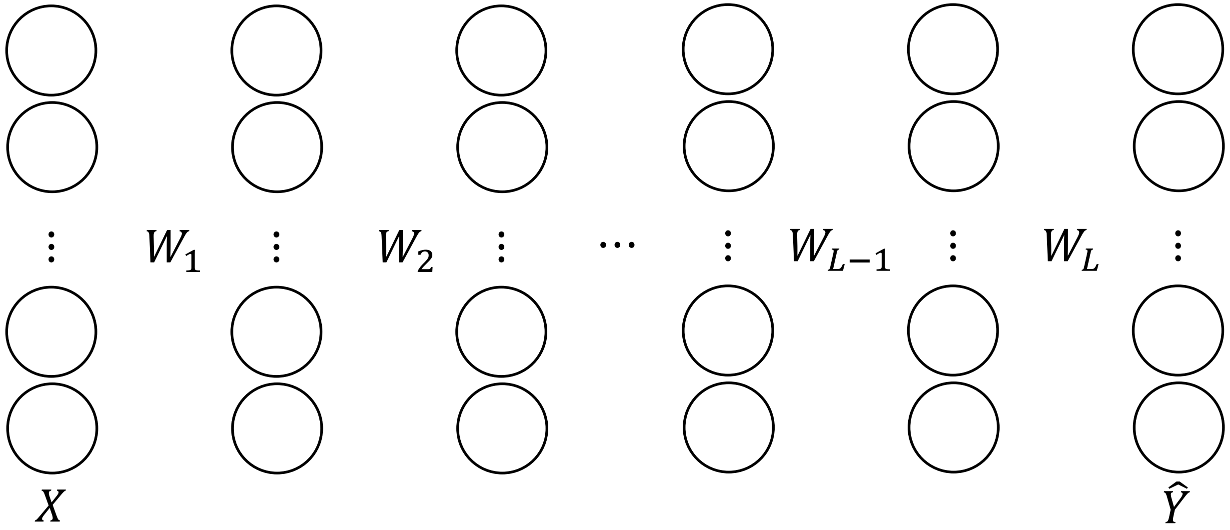
- Activation Function and Loss Function
  - 비선형 함수를 이용해 모델의 표현력을 높임
  - 해결하려는 문제에 맞춰 모델의 성공 정도를 표현해 학습에 사용

1. Backward Pass - 얼마나 학습 시킬까요?
2. Optimizer - 어떻게 학습 시킬까요?
3. 외우지 않고 배우는 모델
4. Batch Normalization
5. (실습) MLP MNIST classification (2)

{ 1. Backward Pass - 얼마나 학습 시킬까요? }

Back Propagation이란?

- MLP는 입력 벡터(or 행렬)과 parameter의 함수로 출력 벡터(or 행렬)를 표현
  - 이렇게 얻어진 출력은 loss function을 통해 해당 MLP가 얼마나 잘 작동하고 있는 지를 표현
  - 이 loss를 줄이는 방향으로 각 parameter를 조절하기 위해 각 parameter에 대한 loss의 편미분 값을 계산하여 이를 이용해 parameter update



$$\hat{Y} = a(W_L a(W_{L-1} \cdots a(W_2 a(W_1 X)) \cdots))$$
$$= f(X, W_1, W_2, \cdots, W_{L-1}, W_L)$$

$$L = loss(Y, \hat{Y})$$

$$w = w - \alpha \frac{\partial L}{\partial w}$$

Partial Derivative이란?

- 편미분
- MLP와 같이 여러 변수로 표현되는 다변수 함수는 각 변수들이 복합적으로 함수 값에 영향을 주기 때문에 다른 변수들의 값을 상수로 둔 상태에서 특정 변수에 대한 도함수 값을 고려
  - 이를 편도함수이라 하며 $\frac{\partial y}{\partial x}$ 로 표현
  - 일변수 함수의 미분과 유사하지만 다른 변수들을 상수로 취급
  - 다변수 함수의 변화량은 변수의 변화 방향을 고려하여 편도함수를 이용해서 표현

$$y = f(x_1, x_2, x_3) \longrightarrow \frac{\partial y}{\partial x_1} = g_1(x_1, x_2, x_3) \quad \frac{\partial y}{\partial x_2} = g_2(x_1, x_2, x_3) \quad \frac{\partial y}{\partial x_3} = g_3(x_1, x_2, x_3)$$

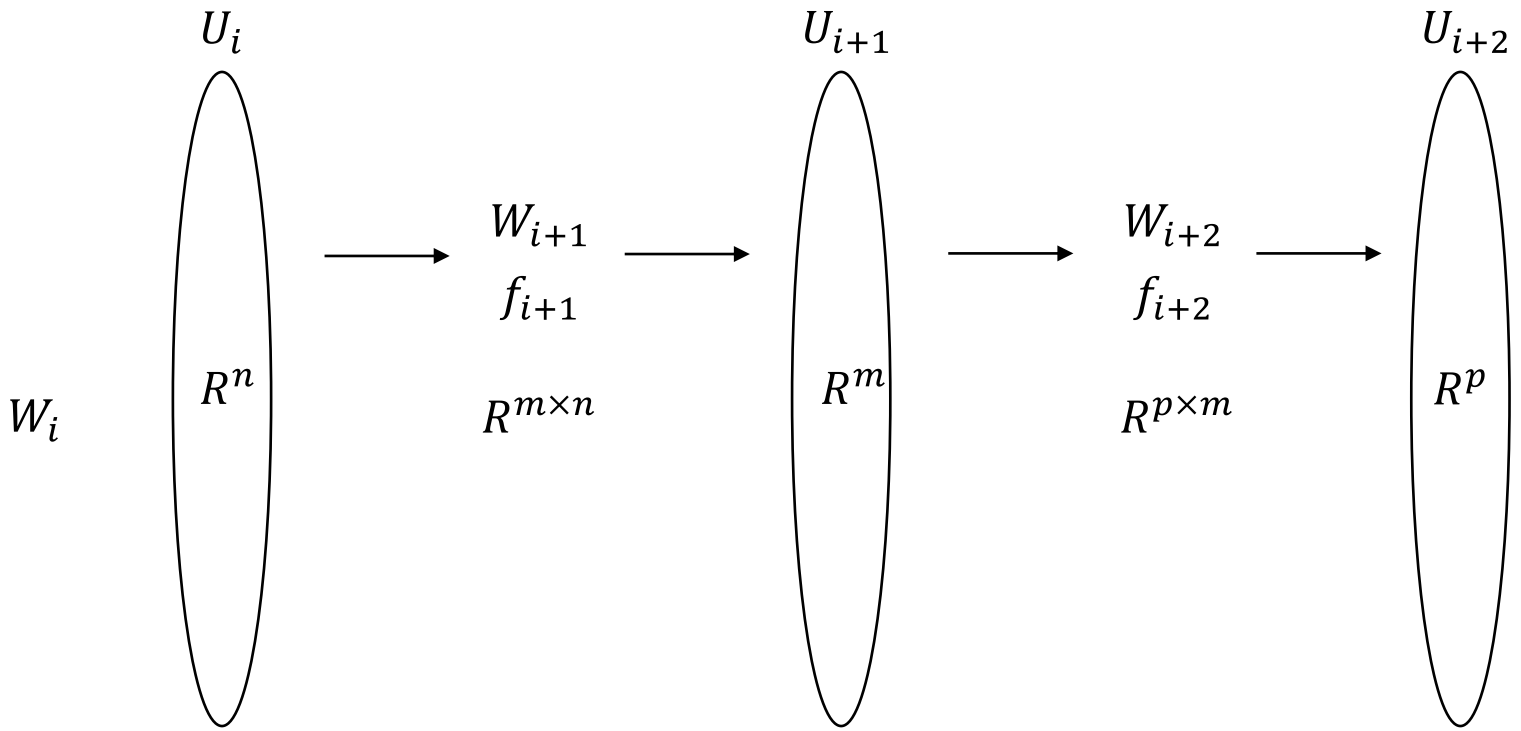
Chain Rule

- 연쇄 법칙
- 합성 함수의 도함수에 대한 공식
    - 다변수 함수에 대한 chain rule을 이용해 MLP에서 각 layer의 parameter에 대한 loss의 편미분 값을 얻음
- MLP와 같이 여러 변수로 표현되는 다변수 함수는 각 변수들이 복합적으로 함수값에 영향을 주기 때문에 편미분 값을 이용

$$y = f(u) = f\big(g(x)\big), \qquad x \in R^n, \qquad u \in R^m, \qquad y \in R^p$$

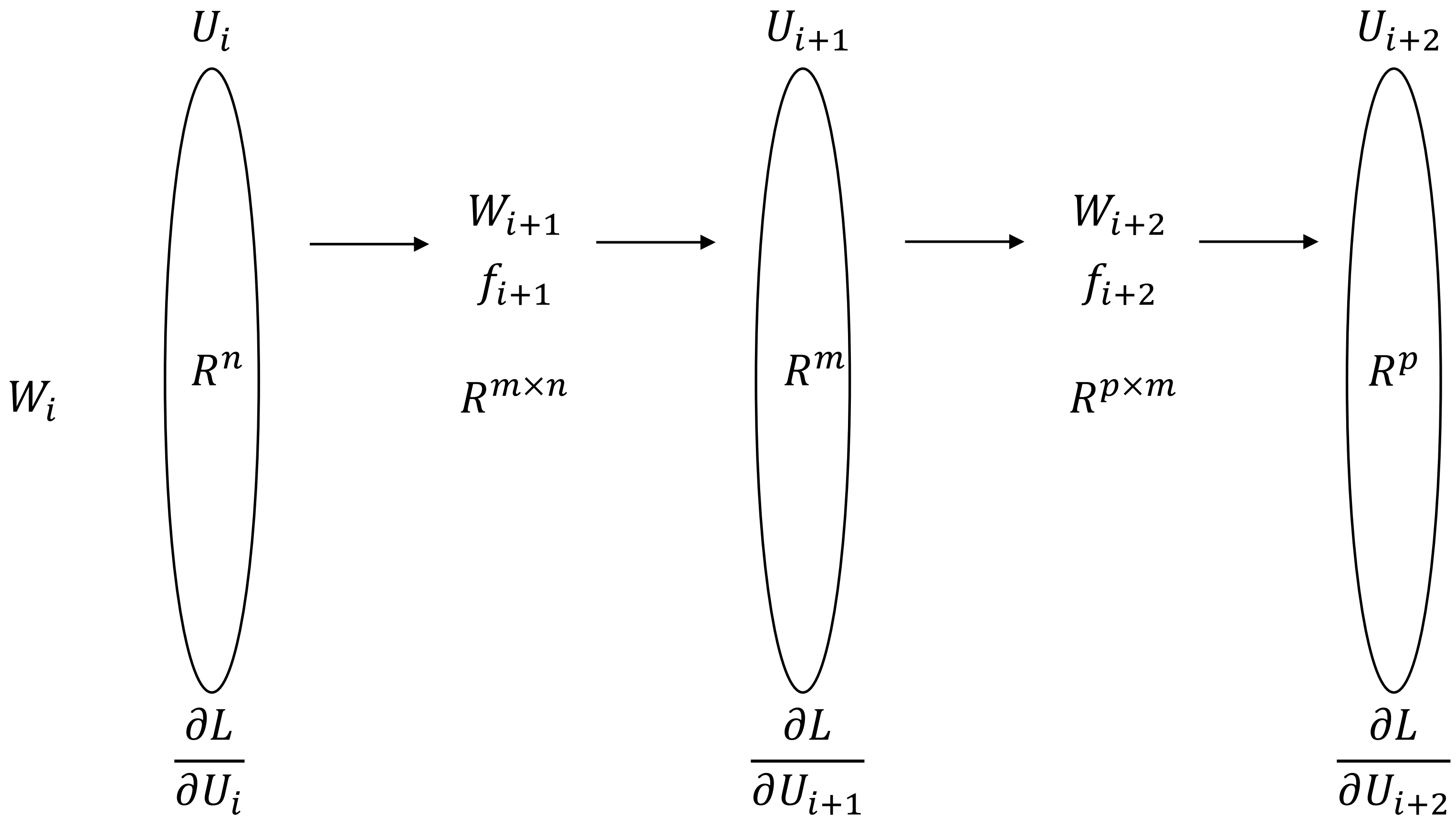
$$\frac{\partial y_i}{\partial x_j} = \frac{\partial y_i}{\partial u_1}\frac{\partial u_1}{\partial x_j} + \cdots + \frac{\partial y_i}{\partial u_m}\frac{\partial u_m}{\partial x_j} = \sum_{k=1}^{m} \frac{\partial y_i}{\partial u_k}\frac{\partial u_k}{\partial x_j}$$

Chain Rule for MLP



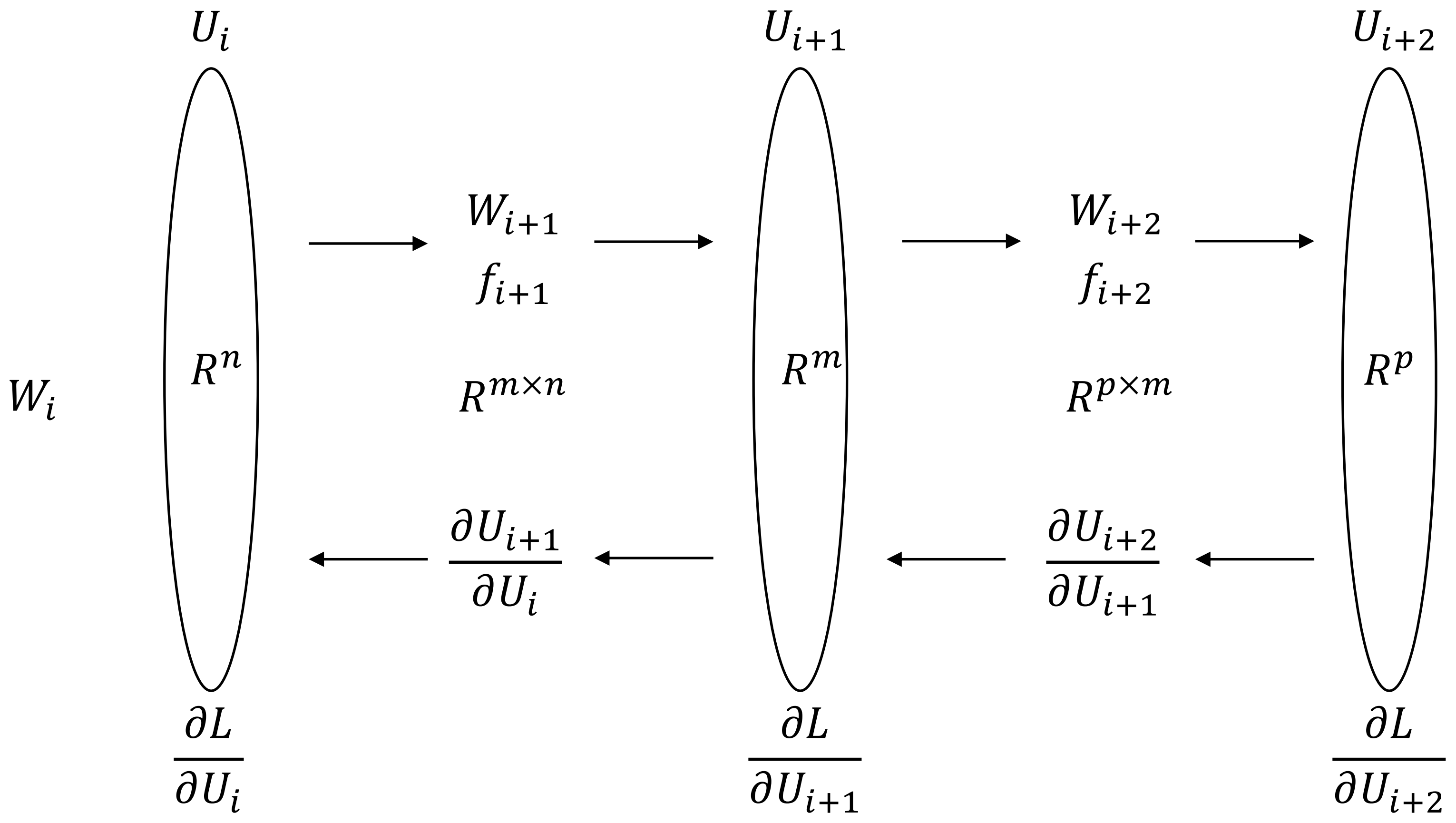
$$
\begin{aligned}
U_{i+2} &= f_{i+2}(U_{i+1}) \\
&= f_{i+2}(f_{i+1}(U_i))
\end{aligned}
$$

Chain Rule for MLP



$$
\begin{aligned}
U_{i+2} &= f_{i+2}(U_{i+1}) \\
&= f_{i+2}(f_{i+1}(U_i))
\end{aligned}
$$

Chain Rule for MLP



$$U_{i+2} = f_{i+2}(U_{i+1})$$
$$= f_{i+2}(f_{i+1}(U_i))$$

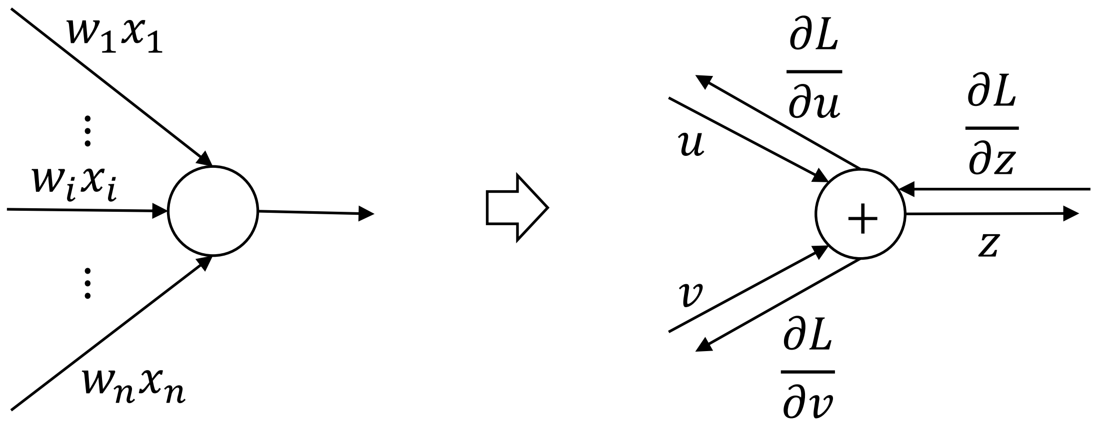
$$\frac{\partial L}{\partial U_i} = \frac{\partial L}{\partial U_{i+1}} \frac{\partial U_{i+1}}{\partial U_i}$$
$$= \frac{\partial L}{\partial U_{i+2}} \frac{\partial U_{i+2}}{\partial U_{i+1}} \frac{\partial U_{i+1}}{\partial U_i}$$

Basic Operations

- MLP의 forward pass에 사용되는 각각의 연산에 대해 chain rule이 어떻게 적용되는 지 분석하여 전체 MLP에 순차적으로 반대 방향으로 적용
  - Backward pass
- MLP의 연산들은 matrix multiplication과 nonlinear activation function으로 구성되어 다음과 같이 구분 가능
  - Addition
  - Multiplication
  - Common variable
  - Nonlinear activation function

Addition Operation

- MLP의 forward pass는 matrix multiplication과 nonlinear activation function으로 구성
- 각 node로 parameter와 이전 층의 값이 곱해져 더해질 때 등장



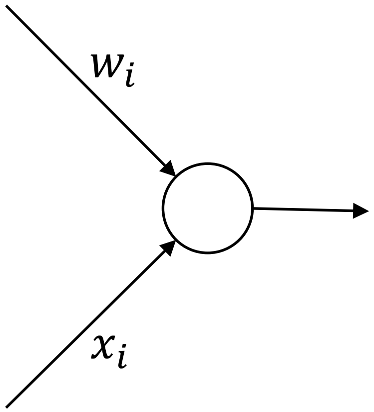
$$z = u + v$$

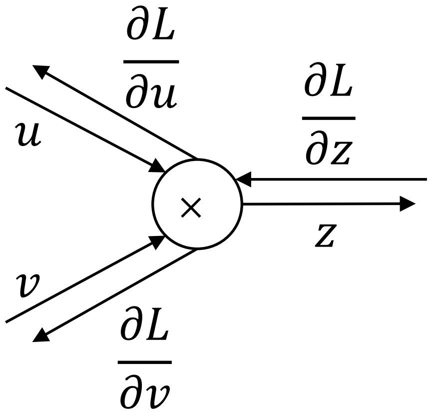
$$\frac{\partial L}{\partial u} = \frac{\partial z}{\partial u}\frac{\partial L}{\partial z} = \frac{\partial L}{\partial z}$$

$$\frac{\partial L}{\partial v} = \frac{\partial z}{\partial v}\frac{\partial L}{\partial z} = \frac{\partial L}{\partial z}$$

Multiplication Operation

- MLP의 forward pass는 matrix multiplication과 nonlinear activation function으로 구성
- 각 layer에서 parameter와 이전 층의 값이 곱해질 때 등장





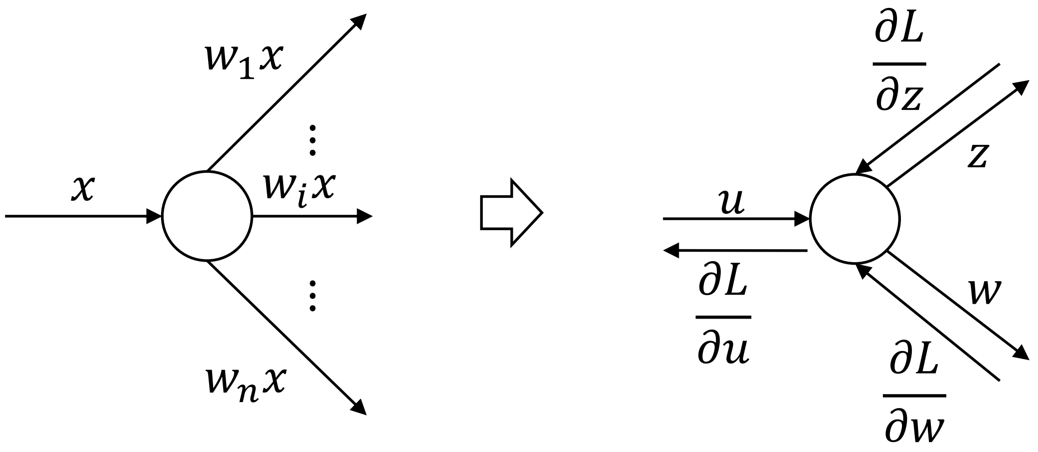
$$z = u \times v$$

$$\frac{\partial L}{\partial u} = \frac{\partial z}{\partial u}\frac{\partial L}{\partial z} = v\frac{\partial L}{\partial z}$$

$$\frac{\partial L}{\partial v} = \frac{\partial z}{\partial v}\frac{\partial L}{\partial z} = u\frac{\partial L}{\partial z}$$

Common Variable

- MLP의 forward pass는 matrix multiplication과 nonlinear activation function으로 구성
- 이전 층의 값이 각 parameter와 곱해질 때 등장

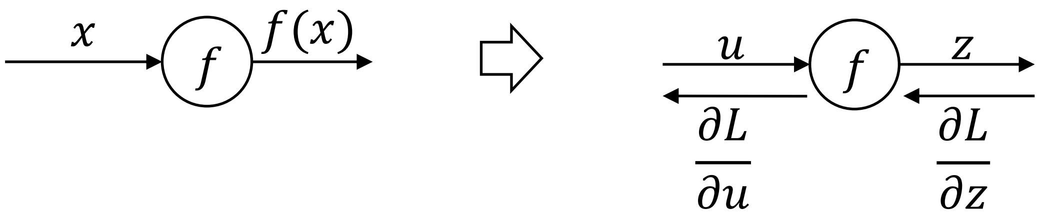


$$z = u$$
$$w = u$$

$$\frac{\partial L}{\partial u} = \frac{\partial z}{\partial u}\frac{\partial L}{\partial z} + \frac{\partial w}{\partial u}\frac{\partial L}{\partial w}$$
$$= \frac{\partial L}{\partial z} + \frac{\partial L}{\partial w}$$

Nonlinear Activation Function

- MLP의 forward pass는 matrix multiplication과 nonlinear activation function으로 구성
- 각 node에서 activation function이 사용될 때 등장



$$z = f(u)$$

$$\frac{\partial L}{\partial u} = \frac{\partial z}{\partial u}\frac{\partial L}{\partial z}$$

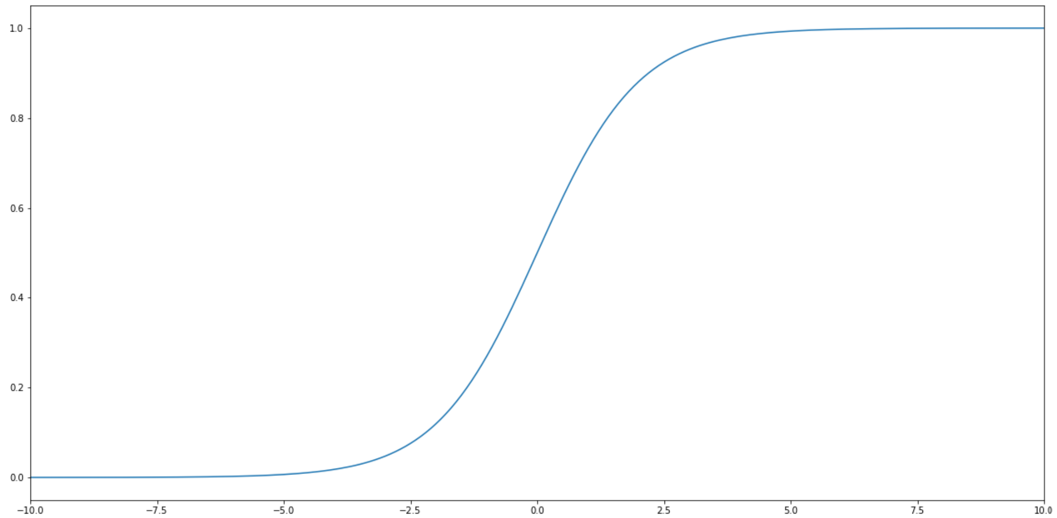
$$= f'(u)\frac{\partial L}{\partial z}$$

Nonlinear Activation Function

- Activation function의 backward pass를 위해서는 해당 node에서 각 함수의 미분값이 필요
  - Activation function은 element-wise로 계산이 이루어지기 때문에 벡터나 행렬을 고려하지 않고 간단하게 처리

Nonlinear Activation Function

- Activation function의 backward pass를 위해서는 해당 node에서 각 함수의 미분 값이 필요
  - Activation function은 element-wise로 계산이 이루어지기 때문에 벡터나 행렬을 고려하지 않고 간단하게 처리



$$f(x) = \sigma(x) = \frac{1}{1+e^{-x}}$$

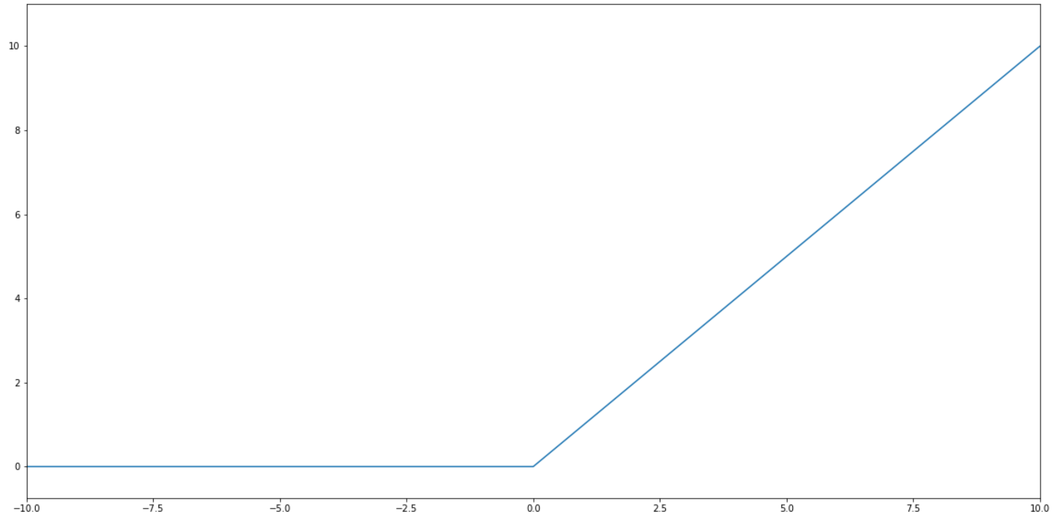
$$f'(x) = \left(\frac{1}{1+e^{-x}}\right)^2 (e^{-x})$$

$$= \frac{1}{1+e^{-x}} \cdot \frac{e^{-x}}{1+e^{-x}} = \sigma(x)(1-\sigma(x))$$

LIKE LION

Nonlinear Activation Function

- Activation function의 backward pass를 위해서는 해당 node에서 각 함수의 미분값이 필요
  - Activation function은 element-wise로 계산이 이루어지기 때문에 벡터나 행렬을 고려하지 않고 간단하게 처리



$$f(x) = \max(0, x)$$

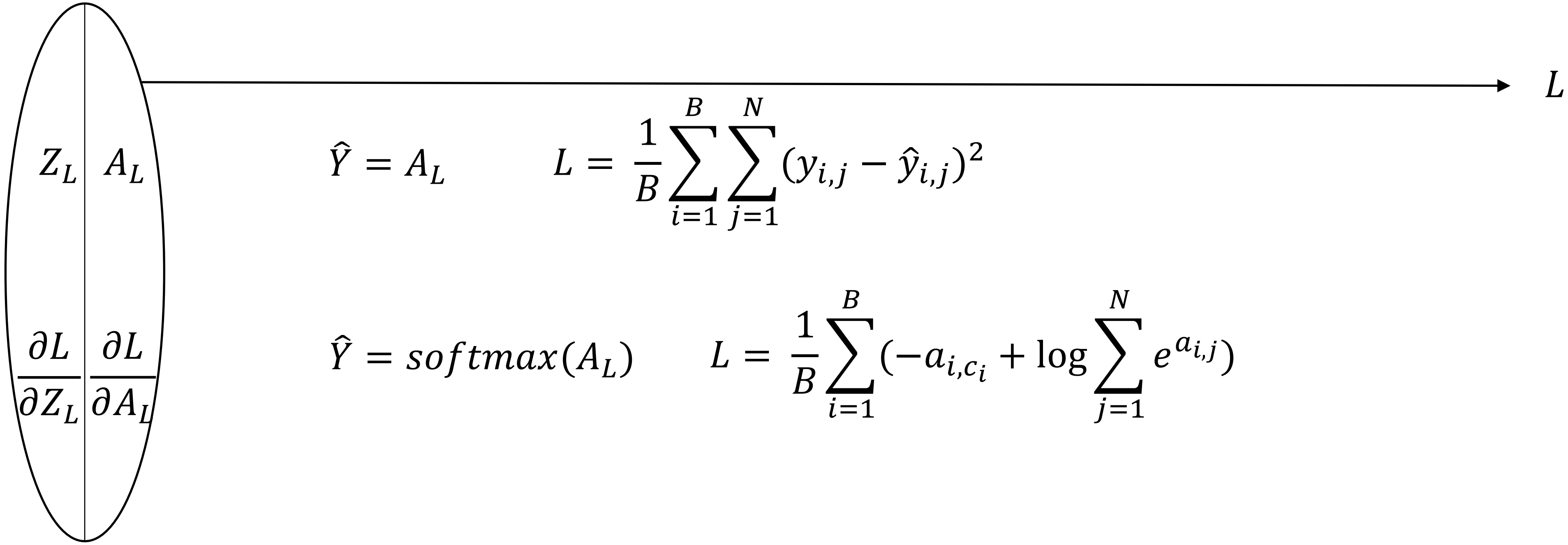
$$f'(x) = \begin{cases} 1 & if\ x > 0 \\ 0 & else \end{cases}$$

Last layer의 Backward pass

$A_L = f(Z_L)$

$Z_L, A_L \in R^{N \times B}$

Last layer의 Backward pass



$A_L = f(Z_L)$

$Z_L, A_L \in R^{N \times B}$

$\hat{Y} = A_L \qquad L = \frac{1}{B}\sum_{i=1}^{B}\sum_{j=1}^{N}(y_{i,j} - \hat{y}_{i,j})^2$

$\hat{Y} = softmax(A_L) \qquad L = \frac{1}{B}\sum_{i=1}^{B}(-a_{i,c_i} + \log\sum_{j=1}^{N} e^{a_{i,j}})$

Last layer의 Backward pass

$Z_L \quad A_L$

$\frac{\partial L}{\partial Z_L} \quad \frac{\partial L}{\partial A_L}$

L

$A_L = f(Z_L)$

$Z_L, A_L \in R^{N \times B}$

$\hat{Y} = A_L \qquad L = \frac{1}{B}\sum_{i=1}^{B}\sum_{j=1}^{N}(y_{i,j} - \hat{y}_{i,j})^2$

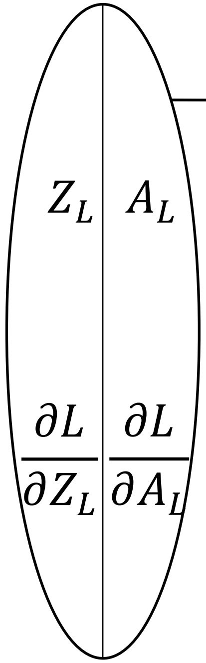
$\hat{Y} = softmax(A_L) \qquad L = \frac{1}{B}\sum_{i=1}^{B}(-a_{i,c_i} + \log\sum_{j=1}^{N} e^{a_{i,j}})$

Recall

$$L = \sum_{i=1}^{n} l_i = -\sum_{i=1}^{n} y_i \log \hat{y}_i = -\log \hat{y}_c = -\log \frac{e^{o_c}}{\sum_{i=1} e^{o_i}} = -o_c + \log \sum_{i=1} e^{o_i}$$

LIKE LION

Last layer의 Backward pass



L

$$A_L = f(Z_L)$$

$$Z_L, A_L \in R^{N\times B}$$

$$\hat{Y} = A_L \qquad L = \frac{1}{B}\sum_{i=1}^{B}\sum_{j=1}^{N}(y_{i,j} - \hat{y}_{i,j})^2 \qquad \frac{\partial L}{\partial A_L} = 2\left(\hat{Y} - Y\right) = 2(A_L - Y)$$

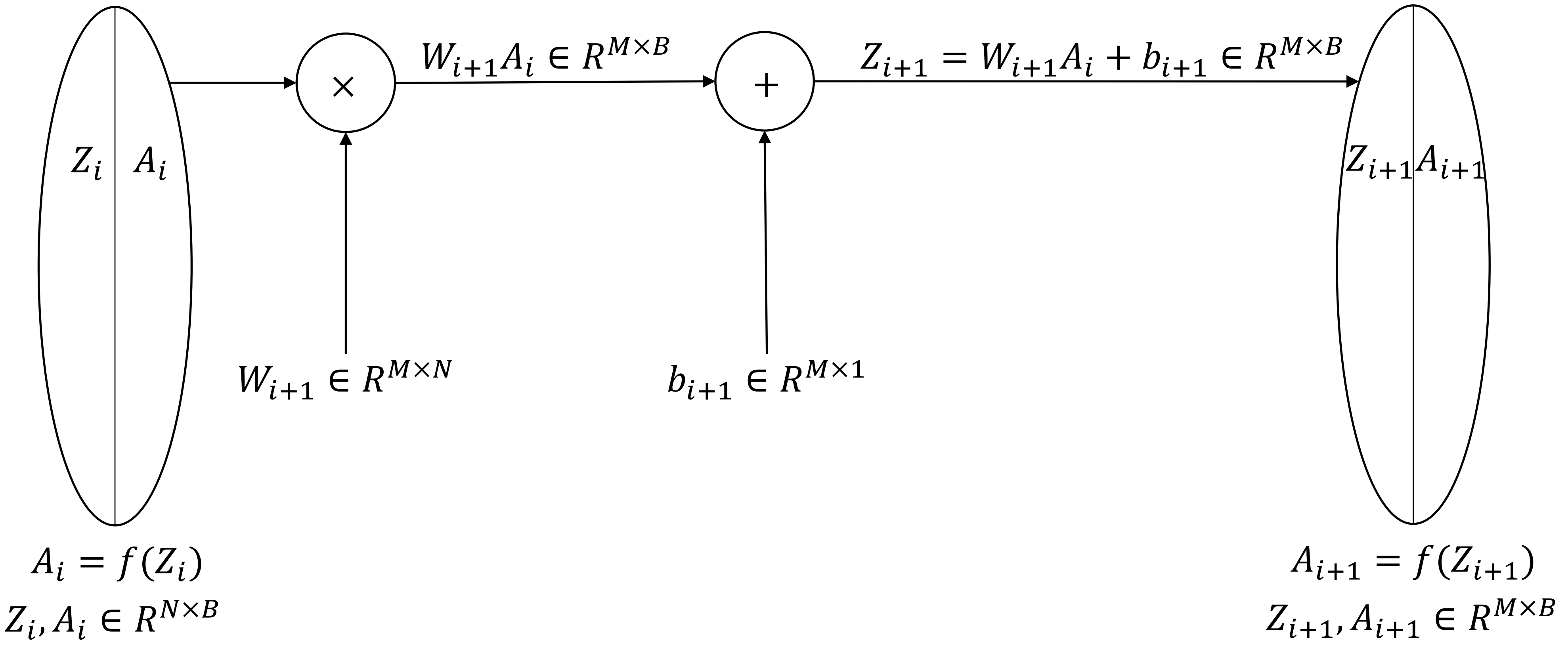
$$\hat{Y} = softmax(A_L) \qquad L = \frac{1}{B}\sum_{i=1}^{B}(-a_{i,c_i} + \log\sum_{j=1}^{N} e^{a_{i,j}}) \qquad \frac{\partial L}{\partial A_L} = \left(\hat{Y} - Y\right)$$

Recall

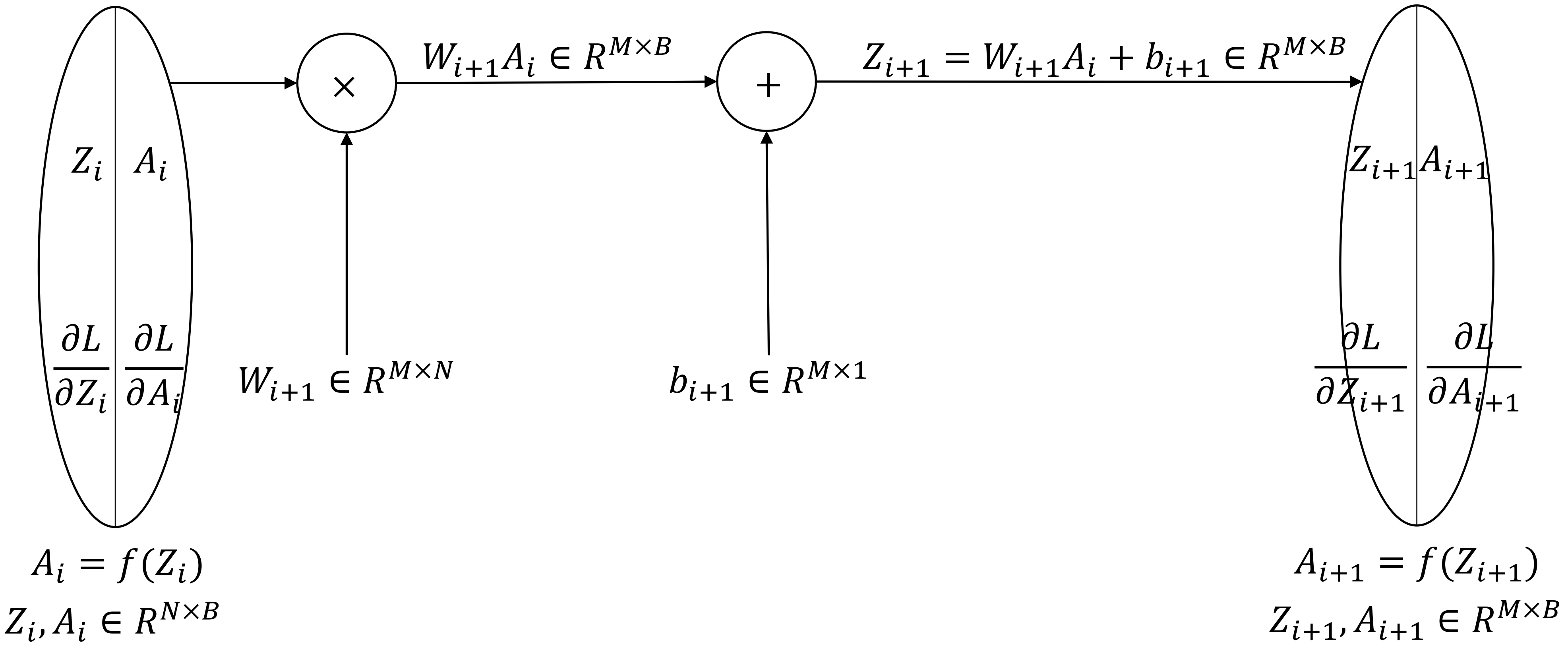
$$L = \sum_{i=1}^{n} l_i = -\sum_{i=1}^{n} y_i \log \hat{y}_i = -\log \hat{y}_c = -\log \frac{e^{o_c}}{\sum_{i=1} e^{o_i}} = -o_c + \log \sum_{i=1} e^{o_i}$$

LIKE LION

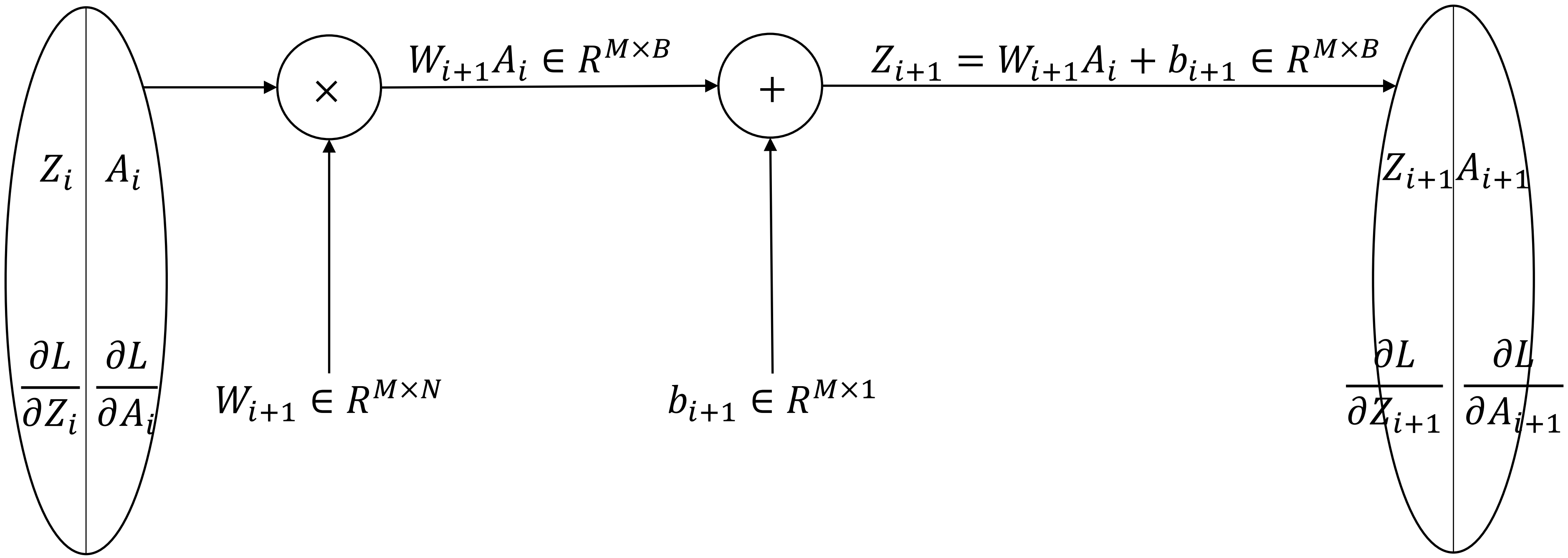
Layer 사이의 Backward pass



Layer 사이의 Backward pass



Layer 사이의 Backward pass



$A_i = f(Z_i)$

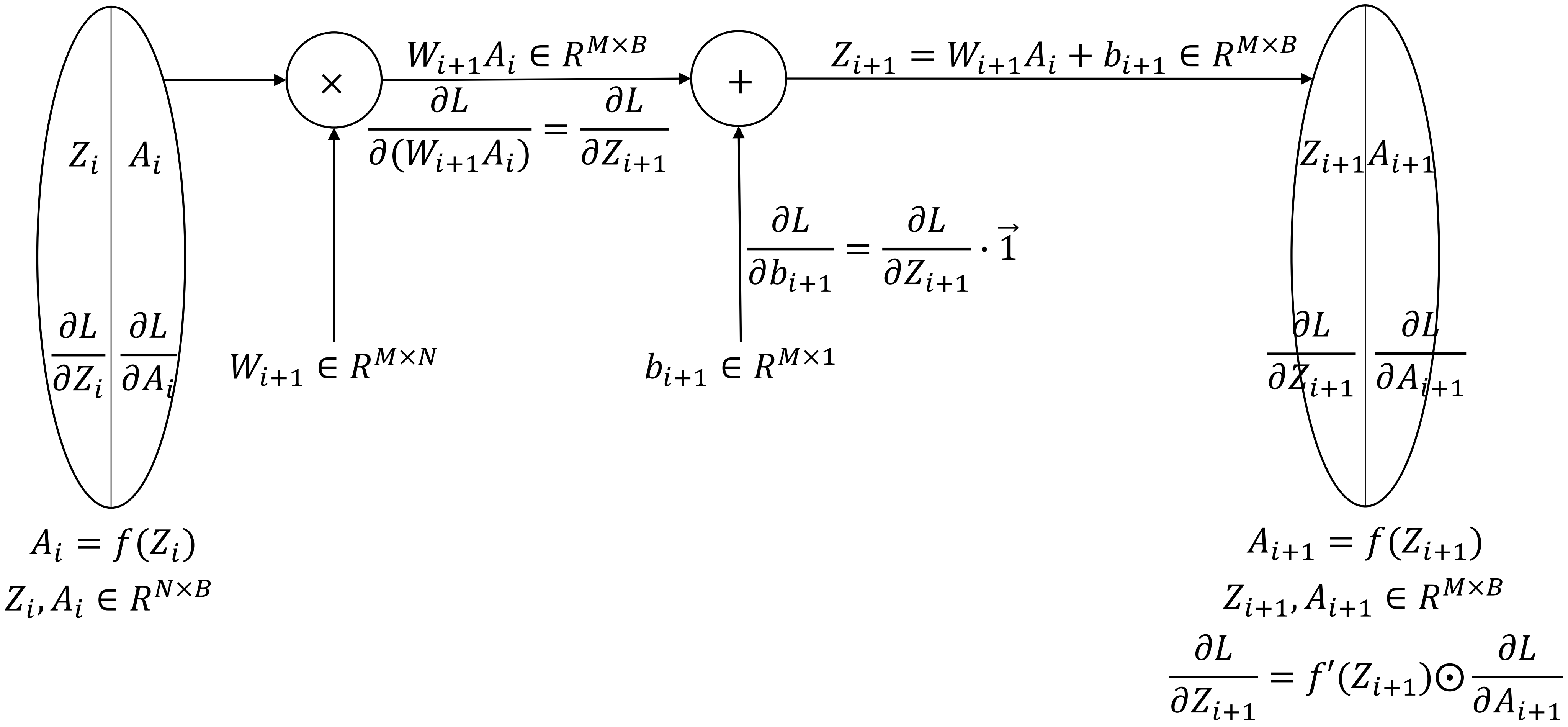
$Z_i, A_i \in R^{N\times B}$

$A_{i+1} = f(Z_{i+1})$

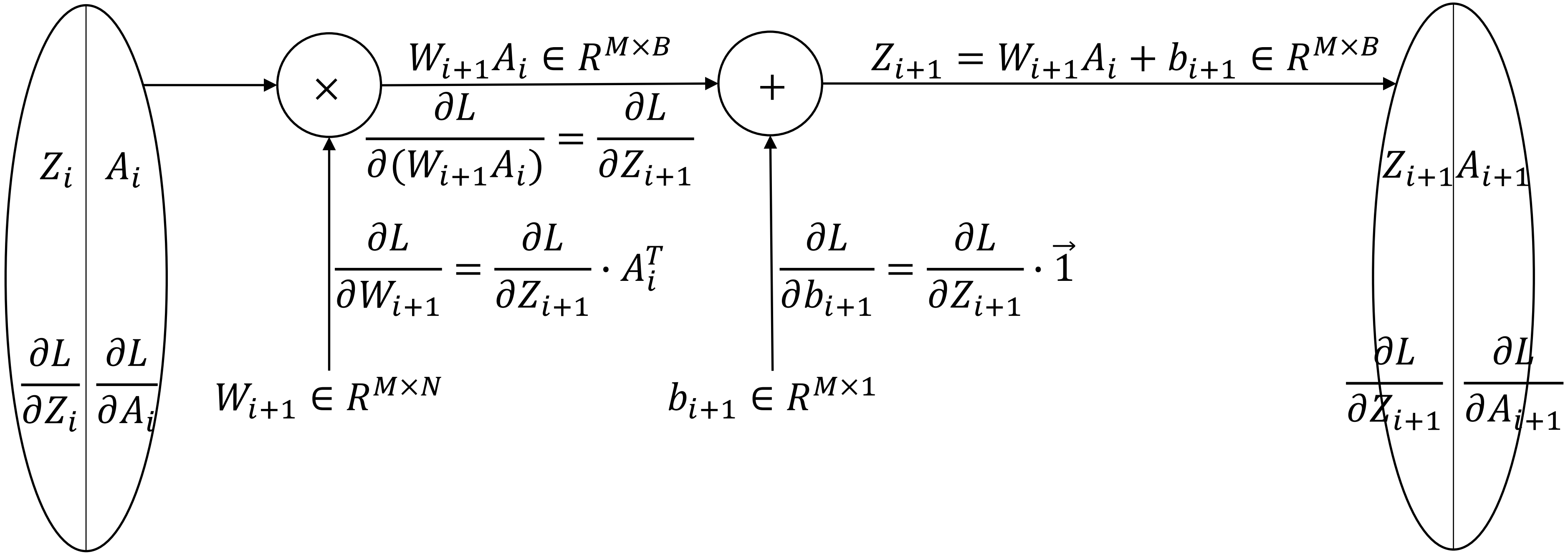
$Z_{i+1}, A_{i+1} \in R^{M\times B}$

$$\frac{\partial L}{\partial Z_{i+1}} = f'(Z_{i+1}) \odot \frac{\partial L}{\partial A_{i+1}}$$

Layer 사이의 Backward pass



Layer 사이의 Backward pass



$A_i = f(Z_i)$

$Z_i, A_i \in R^{N\times B}$

$$\frac{\partial L}{\partial A_i} = W_{i+1}^T \cdot \frac{\partial L}{\partial Z_{i+1}}$$

$A_{i+1} = f(Z_{i+1})$

$Z_{i+1}, A_{i+1} \in R^{M\times B}$

$$\frac{\partial L}{\partial Z_{i+1}} = f'(Z_{i+1}) \odot \frac{\partial L}{\partial A_{i+1}}$$

{ 2. Optimizer - 어떻게 학습 시킬까요? }

LIKE LION

Nonconvex Optimization: Parameter Updates

- Closed form solution을 얻기 어려운 문제들은 gradient descent를 통한 parameter update로 해결
  - 엄청나게 넓고 복잡한 parameter space에서 optimal point를 찾기 위해 다양한 알고리즘 등장
  - 기본적인 stochastic gradient descent를 바탕으로 momentum과 history를 사용하는 안정적인 알고리즘 주로 사용
    - SGD, momentum, AdaGrad, RMSprop, Adam, etc.

Second-Order Optimization Methods

- Gradient-based method보다 더 정확하지만 Hessian matrix의 역행렬 계산을 필요로 함
  - 현실적으로 parameter space에서 계산이 너무 오래 걸리기 때문에 deep learning 모델의 학습에 사용하기 어려움
  - Newton method, Quasi-Newton method, BFGS, L-BFGS, etc.

$$J(\theta) \approx J(\theta_0) + (\theta - \theta_0)^T \nabla_\theta J(\theta_0) + \frac{1}{2}(\theta - \theta_0)^T H(\theta - \theta_0)$$

$$\theta^* = \theta_0 - H^{-1} \nabla_\theta J(\theta_0)$$

Parameter space in Deep Learning

- 실제 deep learning 모델의 parameter space는 차원이 굉장히 크기 때문에 (~1M) global optimal point를 찾는 것이 불가능.
    - 때문에 gradient descent method들은 saddle points을 피하고 local minima를 찾는 데 목표로 함
    - 일반적으로 local minima들은 비슷한 함수 값을 가짐

Gradient-based Methods

- First-order optimization methods
- Parameter를 loss function gradient의 반대 방향으로 update 하여 loss function 이 더 작은 parameter를 얻음
  - Gradient를 이용해 실제 update를 어떻게 하느냐가 중요
  - Mini-batch를 이용해 전체 데이터 셋의 일부를 random sample 하여 update
    - Mini-batch의 size가 증가해도 실제 computation time에 거의 영향 없음
  - 이론적인 분석에 비해 실제 데이터에서 더 잘 작동

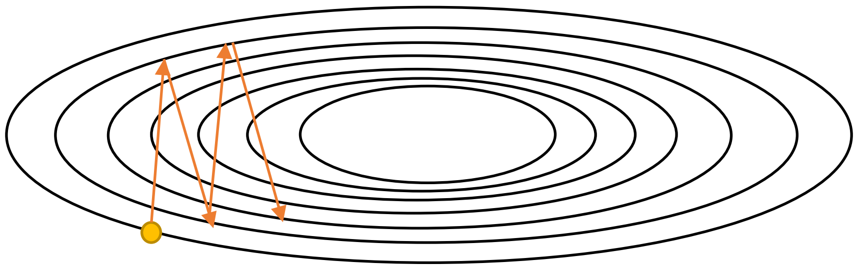
LIKE LION

Stochastic Gradient Descent SGD

- Parameter를 gradient의 반대 방향으로 update
- 장점
  - 가장 빠른 gradient-based method
  - 가장 기본적인 방법으로 쉽게 적용 가능
- 단점
  - Local minima와 saddle points에 빠지기 쉬움
  - Gradient에 noise가 많이 발생
  - Parameter space에서 update 방향이 진동하기 쉬움

$$\theta_t = \theta_{t-1} - \alpha \frac{\partial L}{\partial \theta}$$

α : $Learning\ rate$



Momentum

- 역학의 운동량 개념으로 gradient가 빠르게 변하는 것을 막으며 일관된 방향으로의 update 유도
- Hyper-parameter로 momentum factor가 추가됨

$$\theta_t = \theta_{t-1} - \alpha \frac{\partial L}{\partial \theta}$$

$\alpha : Learning\ rate$

$\Longrightarrow$

$$v_t = \mu v_{t-1} - \alpha \frac{\partial L}{\partial \theta}$$

$$\theta_t = \theta_{t-1} + v_t$$

$$= \theta_{t-1} - \alpha\left(\frac{\partial L}{\partial \theta} - \frac{\mu}{\alpha} v_{t-1}\right)$$

$\mu : Momentum\ factor$

Nesterov Momentum

- 이번 time-step의 gradient를 구하는 위치를 변경
  - Lookahead gradient step
- 장점
  - 기존 momentum에 비해 더 강한 이론적인 converge guarantee 제공
  - 일반적으로 더 잘 작동함
- 단점
  - 실제로 parameter update를 두 번 진행

$$v_t = \mu v_{t-1} - \alpha \frac{\partial L}{\partial \theta}$$

$$\theta_t = \theta_{t-1} + v_t$$

$$= \theta_{t-1} - \alpha\left(\frac{\partial L}{\partial \theta} - \frac{\mu}{\alpha} v_{t-1}\right)$$

$$\Downarrow$$

$$v_t = \mu v_{t-1} - \alpha \frac{\partial L(\theta + \mu v_{t-1})}{\partial \theta}$$

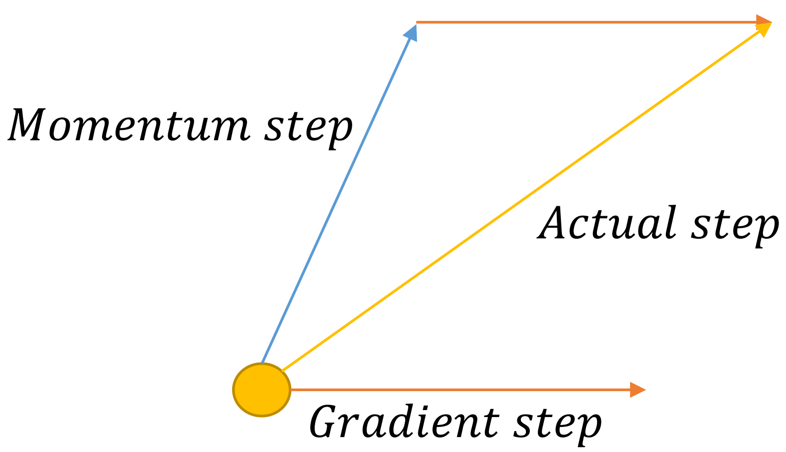
$$\theta_t = \theta_{t-1} + v_t$$

$$= \theta_{t-1} - \alpha\left(\frac{\partial L(\theta + \mu v_{t-1})}{\partial \theta} - \frac{\mu}{\alpha} v_{t-1}\right)$$

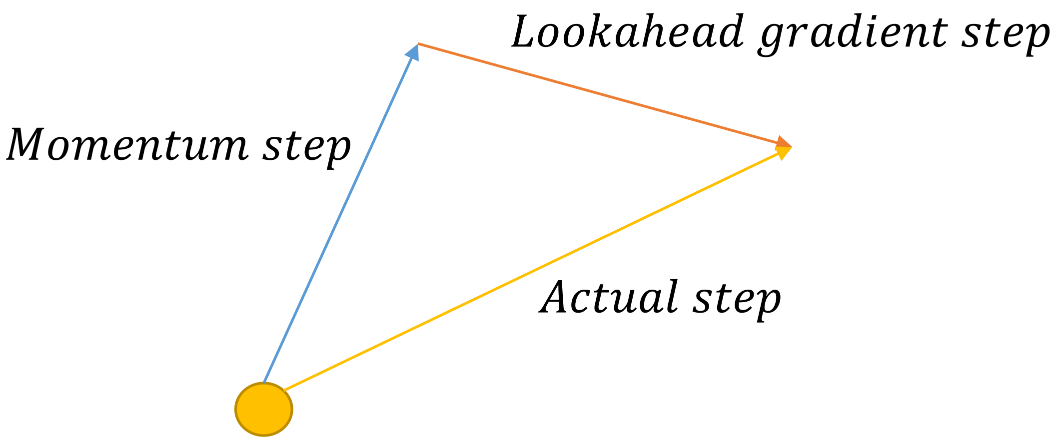
LIKE LION

Nesterov Momentum

Standard momentum update



Nesterov momentum update



AdaGrad

- Update 방향이 과하게 진동하는 문제를 해결하기 위해 parameter-wise update history를 통해 parameter-wise learning rate 적용
  - Per-parameter adaptive learning rate
- 장점
  - Update 양이 많은 parameter의 update를 줄이고 그동안 update가 많이 진행되지 않은 parameter의 update를 늘림
- 단점
  - Adaptive learning rate이 계속 감소

$$S_{t-1} = \sum_{i=1}^{t-1} \left(\frac{\partial L}{\partial \theta_i}\right)^{\circ 2} \qquad \theta_t = \theta_{t-1} - \alpha \frac{\partial L}{\partial \theta} / \left(\sqrt{S_{t-1}} + \epsilon\right)$$

RMSprop

- AdaGrad의 gradient accumulation S_t의 momentum을 적용
  - 너무 먼 과거의 gradient의 효과를 줄임
- 장점
  - 실제 deep learning 모델에 적용하기 어려운 AdaGrad를 적용 가능하도록 개선

$$S_{t-1} = \sum_{i=1}^{t-1} \left(\frac{\partial L}{\partial \theta_i}\right)^{\circ 2}$$

$$\theta_t = \theta_{t-1} - \alpha \frac{\partial L}{\partial \theta} / \left(\sqrt{S_{t-1}} + \epsilon\right)$$

$\Longrightarrow$

$$S_{t-1} = \mu S_{t-2} + (1-\mu)\left(\frac{\partial L}{\partial \theta_{t-1}}\right)^{\circ 2}$$

$$\theta_t = \theta_{t-1} - \alpha \frac{\partial L}{\partial \theta} / \left(\sqrt{S_{t-1}} + \epsilon\right)$$

LIKE LION

Adam

- RMSprop과 momentum의 조합
  - Gradient에 대한 momentum과 parameter-wise learning rate, gradient accumulation에 대한 momentum까지 모두 포함
  - Bias correction이라는 기법을 통해 각 momentum이 초반에 불안정하게 작동하는 걸 막음

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\frac{\partial L}{\partial \theta_{t-1}} \qquad \tilde{m}_t = \frac{m_t}{1-\beta_1^t}$$

$$S_t = \beta_2 S_{t-1} + (1-\beta_2)\left(\frac{\partial L}{\partial \theta_{t-1}}\right)^{\circ 2} \qquad \tilde{S}_t = \frac{S_t}{1-\beta_2^t}$$

$$\theta_t = \theta_{t-1} - \alpha \tilde{m}_t \Big/ \left(\sqrt{\tilde{S}_t} + \epsilon\right)$$

Learning Rate Scheduling

- 모델의 학습이 진행될수록 parameter가 최적 값으로 다가가기 때문에 learning rate을 점점 줄여 더 정확한 수렴을 시도
  - Adaptive learning rate의 개념이 없는 SGD와 momentum method에서 중요
  - AdaGrad, RMSProp과 Adam에서도 추가적으로 많이 사용
- 일반적으로 학습 정도 (epoch, step)에 따라 감소시키며 다양한 방법 존재
  - Linear decay, step decay, exponential decay, 1/t decay
  - 초기 learning rate, decay hyper-parameter는 모두 직접 조절해야 하는 요소

Parameter Initialization

- 모델이 깊어짐에 따라 안정적으로 학습이 진행되지 않고 중간에 gradient가 전달이 안 되거나 activation 값이 0이 되는 문제 발생
  - Nonlinear activation function을 sigmoid 계열에서 ReLU 계열로 바꾸는 것과 더불어 모델의 초기 parameter 설정의 중요성 대두
  - 2010년에 Xavier initialization, 15년에 He initialization이 제안되며 안정적인 학습을 위한 기반이 마련됨
    - 모든 parameter의 초기값을 Gaussian distribution 가정

LIKE LION

{ 3. 외우지 않고 배우는 모델 }

Regularization

- 모델이 학습되기를 기대하는 pattern보다 더 자세하게 데이터를 학습하여 발생하는 overfitting을 막기 위한 기법
- Overfitting
  - 개념적으로는 모델의 complexity가 너무 커 데이터에 존재하는 noise까지 학습함에 따라 학습 데이터가 아닌 데이터에 대해 정확한 추론을 하지 못 하는 현상
- Regularization techniques
  - Norm regularizations, early stopping, ensemble methods, dropout, label smoothing, data augmentation, mix-up, multi-task learning, etc.

Norm Regularizations

- 모델을 학습시키는 loss function에 모델 parameter에 대한 restriction으로 L1 norm이나 L2 norm을 포함
  - 일반적으로 모델의 complexity를 제한하는 효과
  - 기존의 machine learning 기법에서 많이 사용되던 방법으로 deep learning 에서는 다른 regularization 기법들에 효과가 묻히기 때문에 비교적 덜 사용하는 추세

LIKE LION

Early Stopping

- Validation set을 이용한 성능을 모니터링 하며 성능 향상이 더 이상 나타나지 않을 때 학습을 멈추는 기법
  - Training set에 대한 성능은 계속 증가하지만 validation set의 성능이 안 좋아지는 지점을 overfitting이 시작되는 시점으로 이해
  - 모델 자체에 추가적인 변화를 주지 않기 때문에 굉장히 대중적으로 사용하는 기법
- 실제 데이터를 사용하여 학습을 하는 경우 validation 성능이 한참동안 오르지 않다가 갑자기 오르는 경우가 있어 주의 필요

Ensemble Methods

- Deep learning 모델들은 다양한 hyper-parameter를 조절할 뿐만 아니라 다양한 이유에 의해 randomness가 포함되며 학습되기 때문에 모두 다른 모델이 된다고 가정
  - 이로 인해 같은 데이터로 학습을 진행하여도 모두 다른 모델이 학습되어 모델 averaging이 generalization 성능에 도움을 줌
  - 일반적으로 2% 정도의 성능 향상을 기대

Dropout

- 매번 forward pass를 할 때마다 전체 parameter 중 일부를 0으로 대체
  - 모델의 전체 parameter 중 일부를 이용해서도 좋은 성능을 얻을 수 있도록 유도
  - Deep learning은 학습에 오랜 시간이 걸리기 때문에 효과적으로 ensemble을 하는데 한계가 있는데, 이를 개념적으로 보완 가능
  - 각 parameter에 random mask를 적용
    - 실제로 전체 가능한 값 중 일부만 사용되기 때문에 전체 값을 $\frac{1}{p_{drop}}$ 만큼 키워 줌

LIKE LION

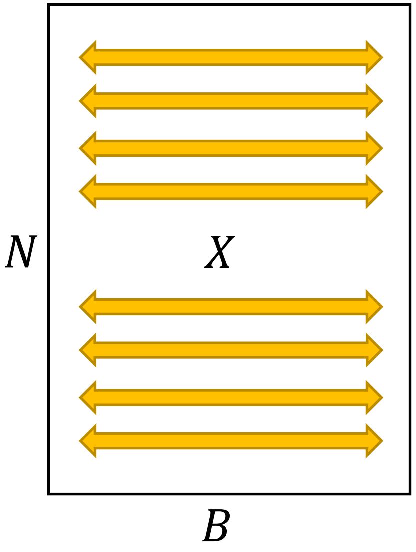
{ 4. Batch Normalization }

Activation Distribution Assumption

- 모델 자체에 대한 분석과 여러 유용한 알고리즘들은 대부분 activation과 parameter의 분포에 대해 Gaussian을 가정
  - 그렇지만 실제 데이터로부터 얻은 activation의 분포는 전혀 다른 분포를 가져 모델의 행동이 이론적인 분석과 크게 달라짐
  - 또한 일반적으로 parameter의 분포도 uniform이나 Gaussian을 가정하고 초기화 하기 때문에 activation도 같은 분포를 가져야 각 layer를 지나면서도 분포가 유지됨
  - 이를 위해 mini-batch 단위로 activation은 normalize 하여 원하는 분포로 만들어주는 batch normalization 사용

Batch Normalization

- 각 layer의 activation이 Gaussian 분포를 이루기를 단순히 기대하는 것이 아니라 batch 단위로 normalize를 하여 원하는 분포로 만들어 줌
  - 보통은 pre-activation을 normalize 하여 사용하지만, 어디에 batch normalization layer를 추가하는 것이 더 좋은 지에 대한 정답은 아직 없음



$X \in R^{N\times B}$ $\quad N : feature\ size, B : Batch\ size$

$$\hat{x}^k = \frac{x^k - E[x^k]}{\sqrt{Var[x^k]}}, k = 1, \cdots, N$$

$$y^k = \gamma \hat{x}^k + \beta \equiv BN_{\gamma,\beta}(x^k) \qquad Learable\ parameters\ \gamma, \beta$$

Batch Normalization

- 학습 과정에서는 mini-batch 전체의 정보를 이용해 batch-statistics를 계산하여 normalize에 사용
- 추론 과정에서는 mini-batch 단위의 데이터를 얻는 것이 아니라 데이터 하나를 다룬다고 가정하기 때문에 batch-statistics를 사용할 수 없음
  - 때문에 이 때는 학습 과정에서 기억해 둔 training-statistics를 사용하여 normalize 진행

Summary

- 편미분 partial derivative와 연쇄 법칙 chain rule을 이용해 MLP의 각 parameter에 대한 loss function의 gradient를 얻어내는 backward pass를 통해 모델 학습
- Backward pass를 통해 얻어진 gradient를 사용해 모델을 학습 시키는 다양한 알고리즘
  - SGD, momentum, Nesterov momentum, AdaGrad, RMSProp, Adam
- 모델이 학습 데이터에 overfitting 되는 문제를 해결하기 위한 다양한 regularization 방법
  - L1, L2 penalty, early stopping, ensemble methods, dropout
- 모델 activation의 분포를 조정해 학습 과정의 안정도를 높이고 빠른 학습을 돕는 batch normalization

{ 5. (실습) MLP MNIST classification (2) }