



Day59; 20221129

| | |
|------|----------------|
| 📅 날짜 | @2022년 11월 29일 |
| 👤 유형 | @2022년 11월 29일 |
| ☰ 태그 | |

GitHub - u8yes/AI

u8yes/AI



You can't perform that action at this time. You signed in with another tab or window. You signed out in another tab or window. Reload to refresh your session. Reload to refresh your session.

⌚ <https://github.com/u8yes/AI>

1 Contributor 0 Issues 0 Stars 0 Forks

[https://s3-us-west-2.amazonaws.com/secure.notion-static.com/a3f06dee-4092-49cc-ad88-c8ae4e5c0ff6/06_%EC%8B%A0%EA%B2%BD%EB%A7%9D\(%EB%94%A5%EB%9F%AC%EB%8B%9D\).pdf](https://s3-us-west-2.amazonaws.com/secure.notion-static.com/a3f06dee-4092-49cc-ad88-c8ae4e5c0ff6/06_%EC%8B%A0%EA%B2%BD%EB%A7%9D(%EB%94%A5%EB%9F%AC%EB%8B%9D).pdf)

기하학(Affine)

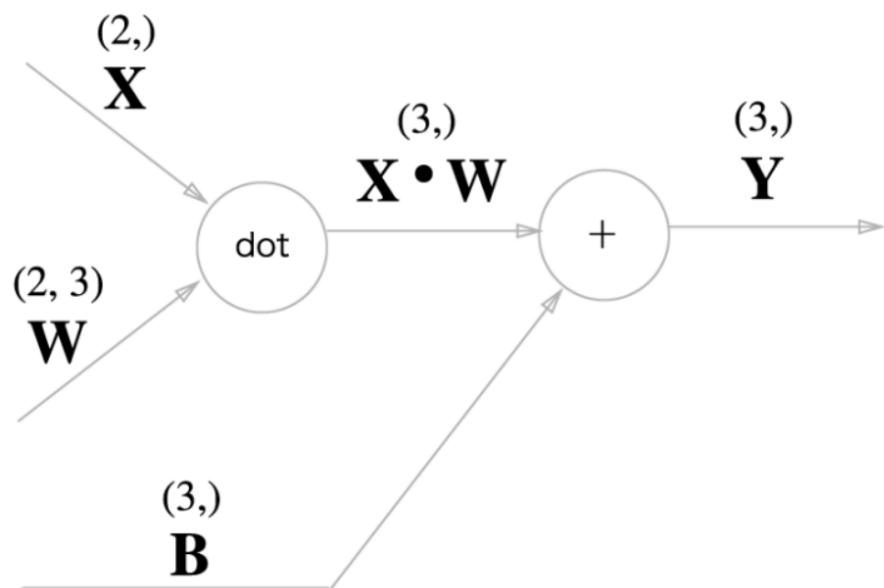
기하학 - 점, 선, 면의 공간좌표

Layer를 Affine Layer라고 부른다. 행렬 내적 계산하는 것을 Affine Transformation.

1차 함수 X 는 tuple 2열(2,) - 2개의 feature, Y 는 tuple 3열(3,) - 3개의 feature

회귀 그래프를 계산식으로 풀어놓은 것.

Affine 계층의 계산 그래프 : 변수가 행렬(다차원 배열)임에 주의



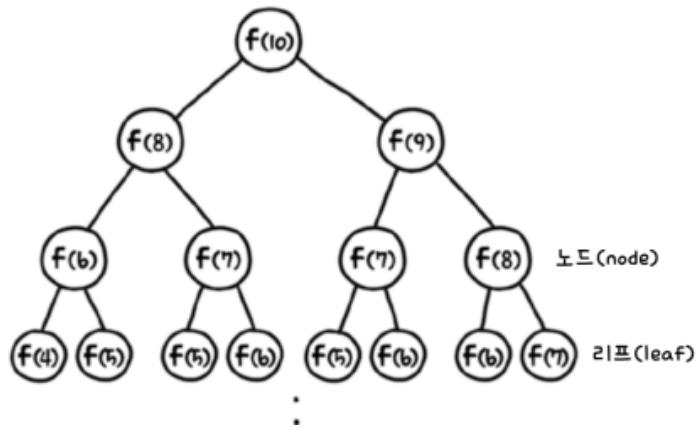
$$W = \begin{pmatrix} W_{11} & W_{21} & W_{31} \\ W_{12} & W_{22} & W_{32} \end{pmatrix}$$

행이 열로 간 것을 전치행렬이라고 한다.

(T : 전치행렬)

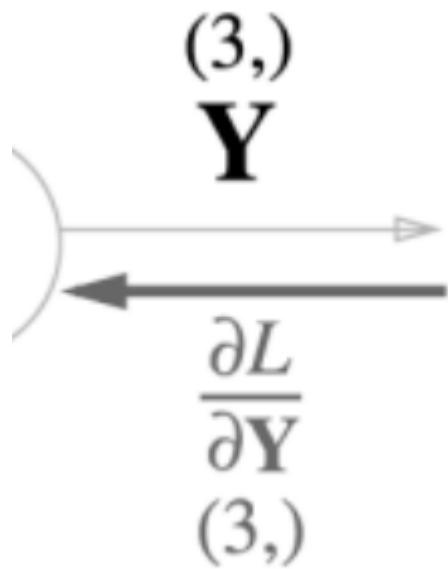
$$\mathbf{W}^T = \begin{pmatrix} W_{11} & W_{12} \\ W_{21} & W_{22} \\ W_{31} & W_{32} \end{pmatrix}$$

이렇게 덧셈 횟수가 기하급수적으로 늘어나는 이유를 그림으로 살펴보겠습니다. 일단 그림을 보기 위해 용어를 정리해 보면 다음과 같은 형태의 그림을 트리^{tree}라고 부릅니다. 트리에 있는 각각의 지점을 노드^{node}, 노드 중에 가장 마지막 단계의 노드를 리프^{leaf}라고 부릅니다.



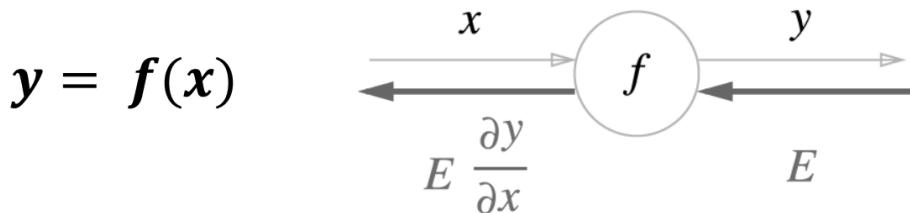
파이썬은 함수 내부에서 함수 외부에 있는 변수를 참조하지 못합니다. 참조라는 말이 조금 어려울 수도 있는데, 변수에 접근하는 것을 참조^{reference}라고 부릅니다. 함수 내부에서 함수 외부에 있는 변수라는 것을 설명하려면 다음과 같은 구문을 사용합니다.

편미분 한 Y에 대한 편미분 Loss



Loss는 0이 될 때까지(근접할 때까지) 반복한다.

계산 그래프의 역전파



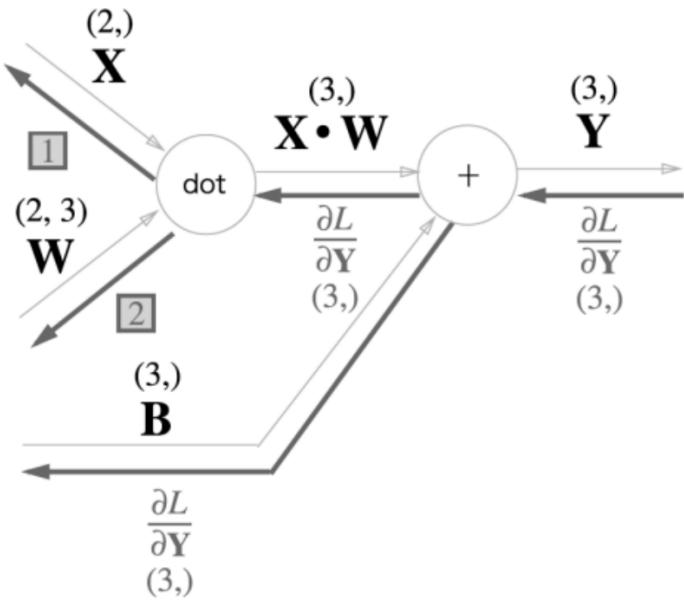
● 역전파의 계산 절차

- 신호 E 에, 노드의 국소적 미분($\frac{\partial y}{\partial x}$)을 곱한 후 다음 노드로 전달 수행.
- 국소적 미분이란 : 순전파 때의 $y = f(x)$ 계산의 미분을 구한다는 것이며, 이는 x 에 대한 y 의 미분($\frac{\partial y}{\partial x}$)을 구한다는 뜻.

Affine 계층의 역전파

$$1 \quad \frac{\partial L}{\partial \mathbf{X}} = \frac{\partial L}{\partial \mathbf{Y}} \mathbf{W}^T \quad (2,) \quad (3,) \quad (3, 2)$$

$$2 \quad \frac{\partial L}{\partial \mathbf{W}} = \mathbf{X}^T \frac{\partial L}{\partial \mathbf{Y}} \quad (2, 3) \quad (2, 1) \quad (1, 3)$$



각 변수의 형상 주의

$$\mathbf{X} = (x_0, x_1, \dots, x_n)$$

$$\frac{\partial L}{\partial \mathbf{X}} = \left(\frac{\partial L}{\partial x_0}, \frac{\partial L}{\partial x_1}, \dots, \frac{\partial L}{\partial x_n} \right)$$

Layer(Dense)

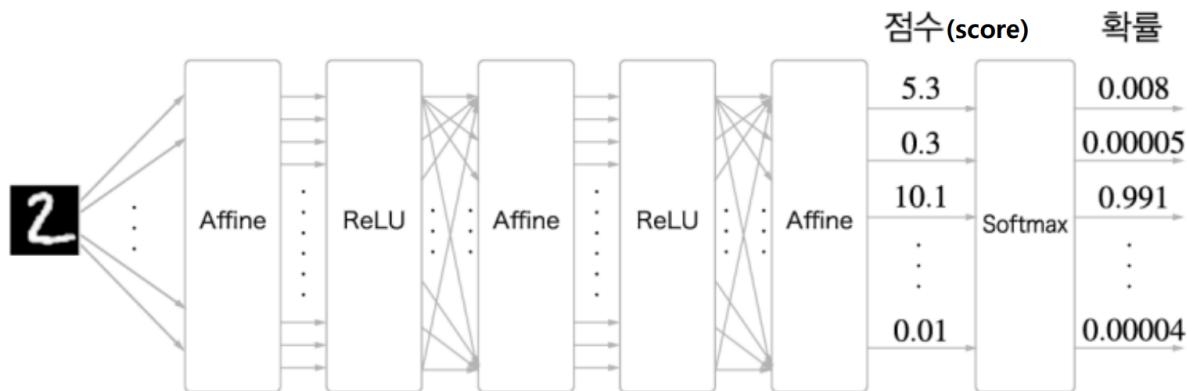
신경망에서는 가운데 affine, relu를 은닉층이라고 부름.

딥러닝은 relu를 많이 씀.

소프트맥스는 최종적으로 1이 된 것을 확률로 나누어줌.

손글씨 숫자 인식에서의 softmax 계층의 출력

- 소프트맥스 함수 : 입력 값을 정규화하여 출력(출력의 합 : 1)
 - 추론할 때는 일반적으로 소프트맥스 계층을 사용하지 않음.



은닉층이 많은 것을 딥러닝이라고 함. 보통은 3개 layer 이상을 기준으로 잡기도 함.

softmax를 통해서 실제로 0~1로 만들어줌. 학습시에만 softmax를 붙여주고 일반적으로 사용하지 않음.

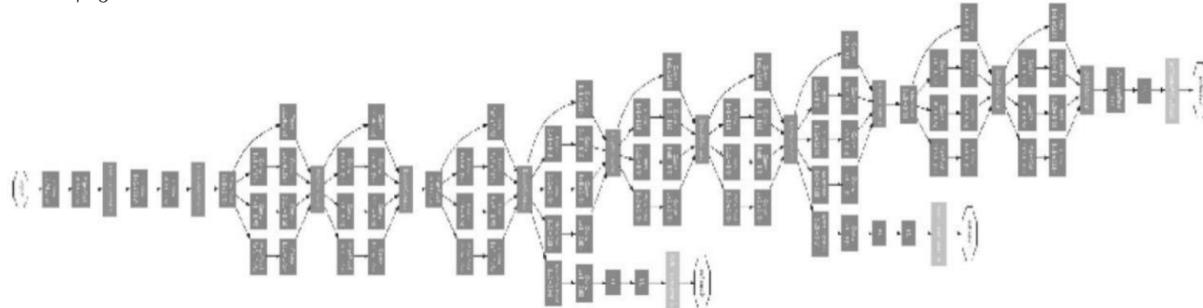
손글씨 숫자 인식에서의 softmax 계층의 출력

- 소프트맥스 함수 : 입력 값을 정규화하여 출력(출력의 합 : 1)
 - 추론할 때는 일반적으로 소프트맥스 계층을 사용하지 않음.

실제 필드에서 적용하고 있는 모델.

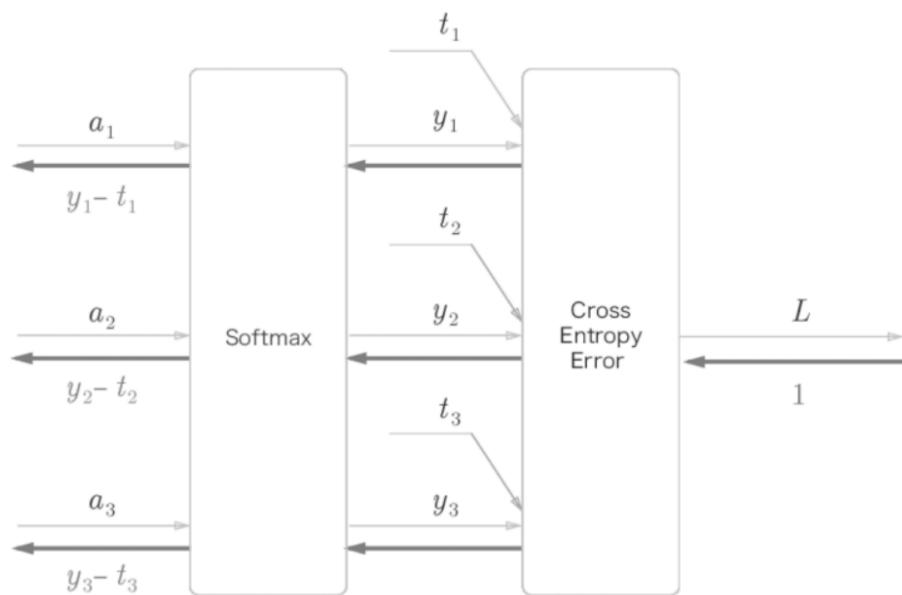
GoogLeNet

- 구성

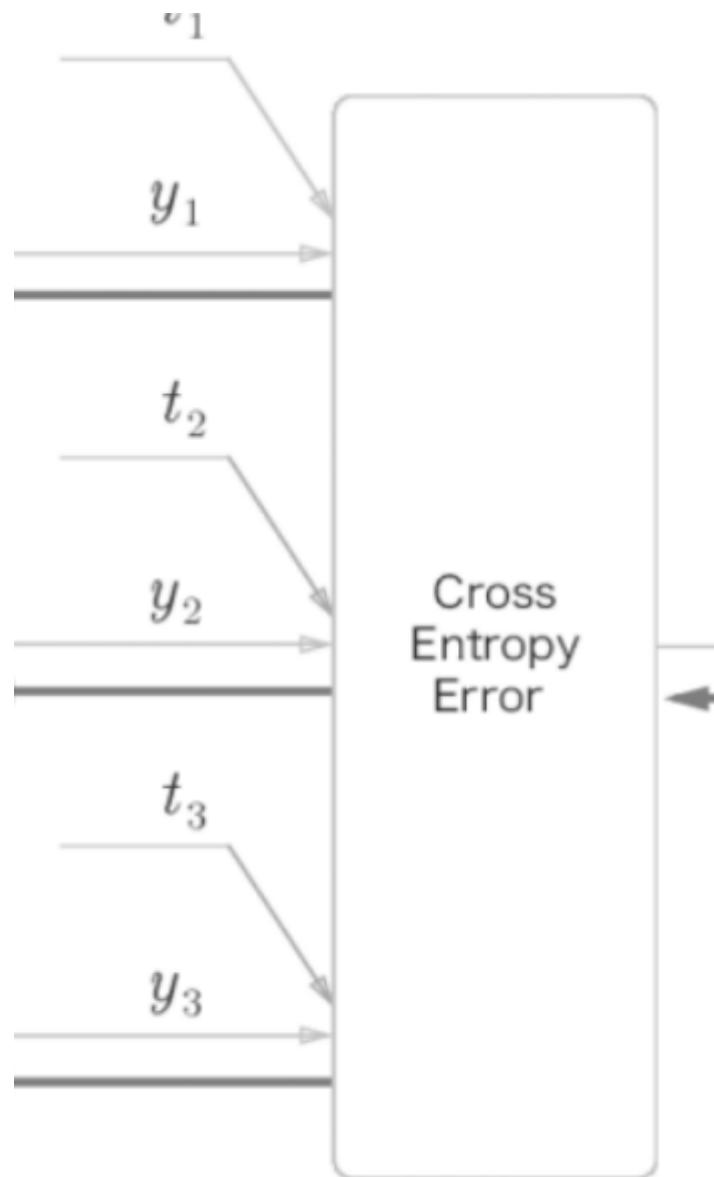


t 는 bias값

간소화한 Softmax-with-Loss 계층의 계산 그래프



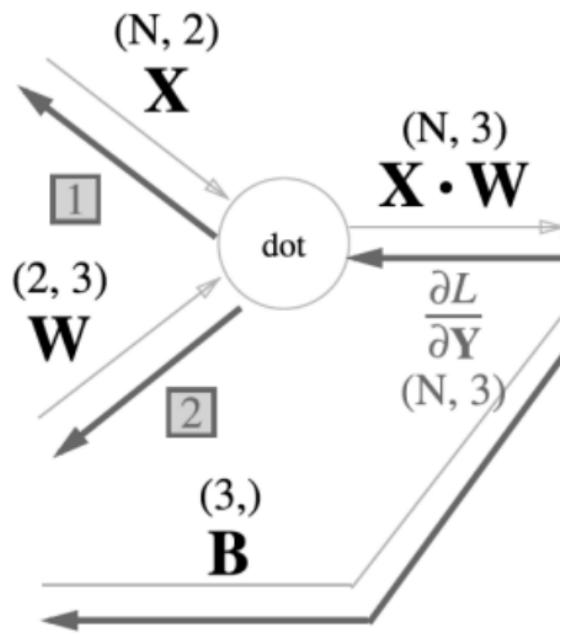
여기서는 우리가 계산한 부분이라서 업데이트는 위 그래프에 있는 $y-t$ 부터 하면 된다.



전치 행렬

$$\boxed{2} \quad \frac{\partial L}{\partial \mathbf{W}} = \mathbf{X}^T \cdot \frac{\partial L}{\partial \mathbf{Y}}$$

$(2, 3) \quad (2, N) \quad (N, 3)$



학습 관련 기술들

매개변수 갱신

신경망 학습의 목적 : 손실 함수의 값을 가능한 한 낮추는 매개변수를 찾는 것.

확률적 경사 하강법(SGD : Stochastic Gradient descent)

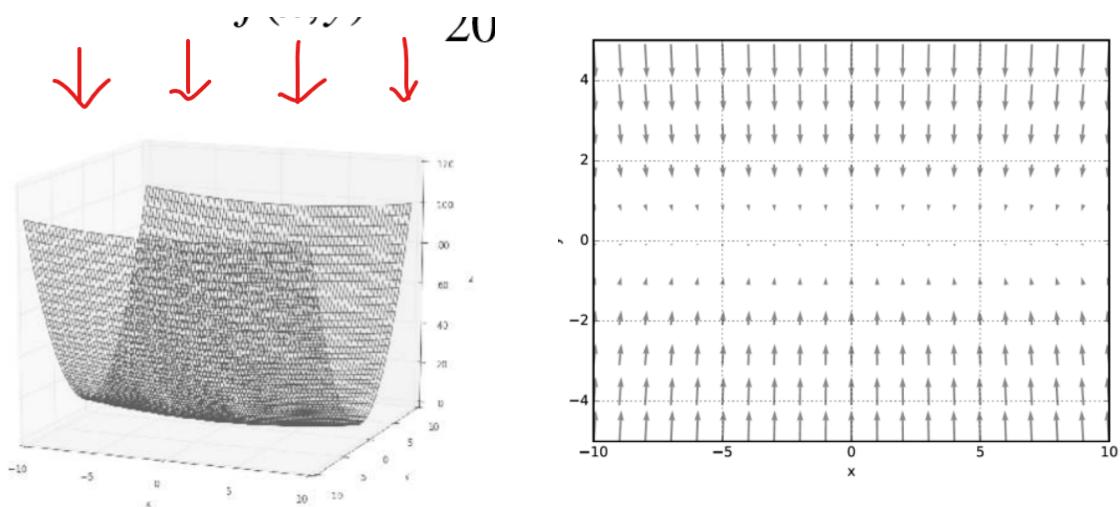
- 최적의 매개변수 값을 찾는 단서로 매개변수의 기울기(미분)를 이용했음.

$$\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial L}{\partial \mathbf{W}}$$

- \mathbf{W} : 갱신할 가중치 매개변수
- $\frac{\partial L}{\partial \mathbf{W}}$: \mathbf{W} 에 대한 손실 함수의 기울기
- η : 학습률(0.01 혹은 0.001등)

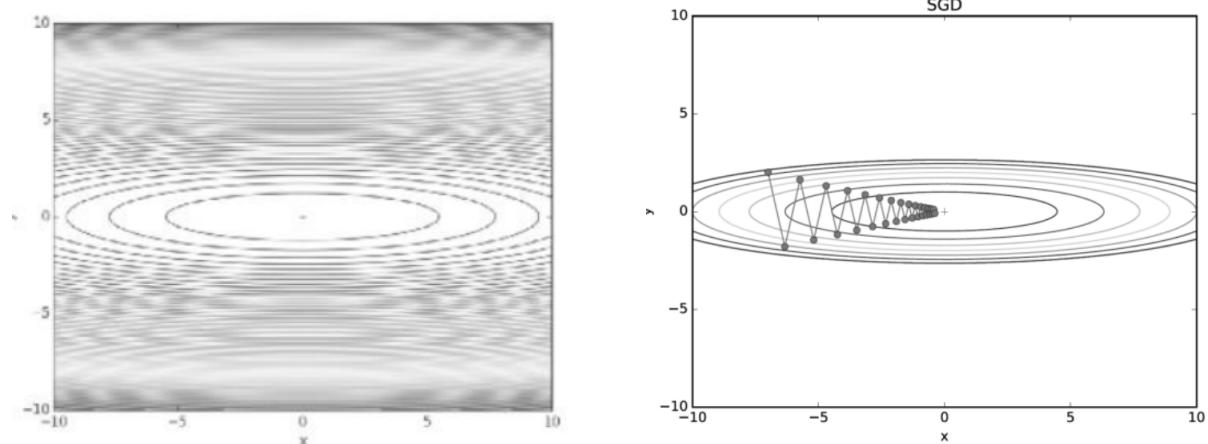
SGD 단점

위에서 보면 다음 그래프 모양이 나옴.

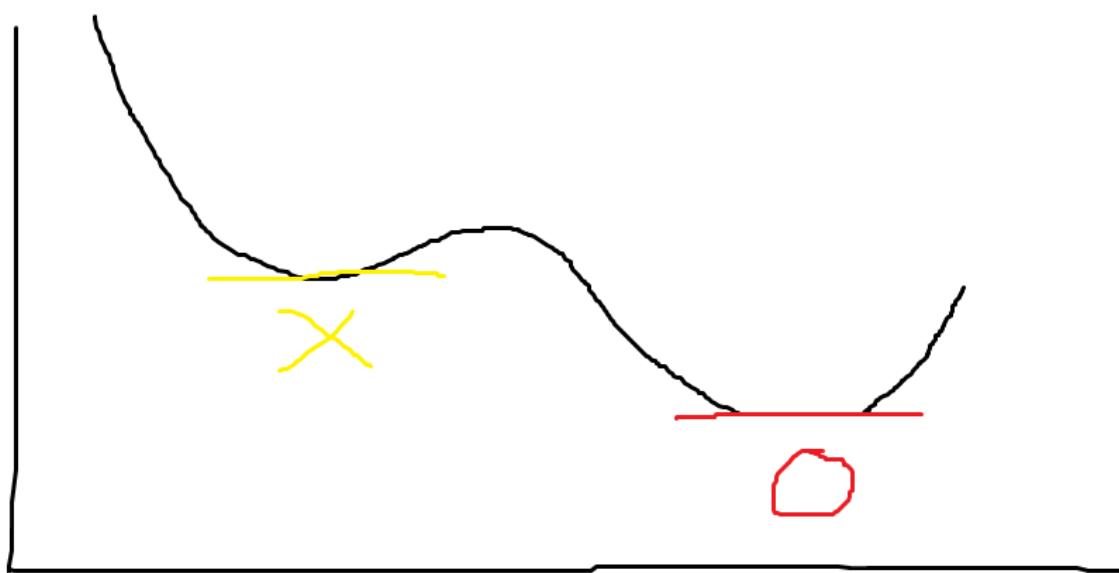


$$f(x,y) = \frac{1}{20}x^2 + y^2 \text{ 의 기울기}$$

등고선 최소점으로 안 가고 지그재그로 가게 됨(Worse)



최저점을 못 찾음, 왼쪽(+위쪽) 최저점을 최저점이라 판단함.



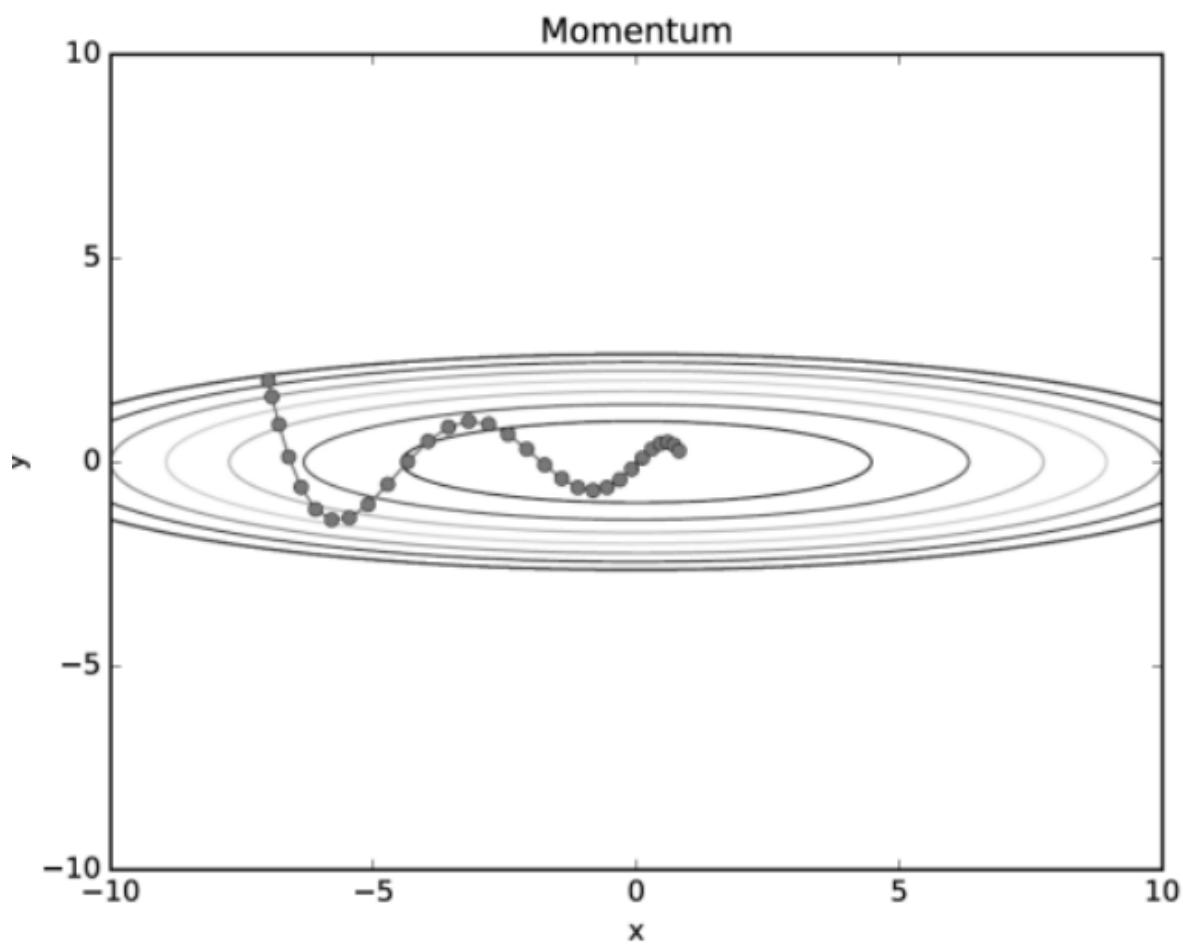
모멘텀(Momentum)

- $\mathbf{v} \leftarrow \alpha \mathbf{v} - \eta \frac{\partial L}{\partial \mathbf{W}}$ \mathbf{v} : 속도(Velocity) - 기울기 방향으로 힘을 받아 물체가 가속된다는 물리 법칙
- $\mathbf{W} \leftarrow \mathbf{W} + \mathbf{v}$



- 모멘텀의 이미지 : 공이 그릇의 곡면(기울기)을 따라 구르듯 움직인다.

속도는 방향성까지 고려됨. (속력은 방향성 없음)



- 모멘텀에 의한 최적화 갱신 경로

decay

미국·영국 [dɪ'keɪ] ↗ 영국식 ↗

(명사)

1 부패, 부식

tooth decay ↗

총치

2 (사회제도 등의) 쇠퇴[퇴락]

economic/moral/urban decay ↗

경제적/도덕적/도시의 퇴락

(동사)

1 부패하다, 썩다; 부패시키다, 썩게 만들다 (=rot)

decaying leaves / teeth / food ↗

썩어 가는 낙엽들/치아들/음식물

2 (건물지역이) 퇴락하다

decaying inner city areas ↗

퇴락하고 있는 도심지들

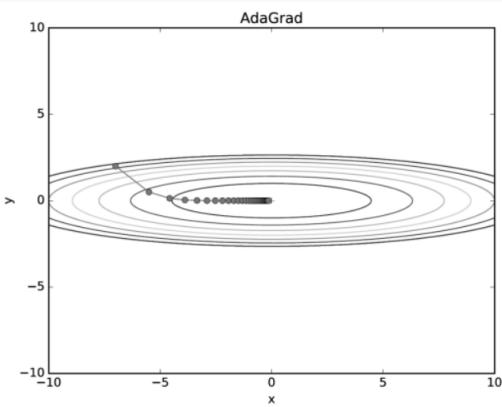
여러 사이트 다른 뜻 ↗

AdaGrad : 각각의 매개변수에 맞춤형 값을 만들어 주는 방식(⊙ : 행렬의 원소별 곱셈).

$$\mathbf{h} \leftarrow \mathbf{h} + \frac{\partial L}{\partial \mathbf{W}} \odot \frac{\partial L}{\partial \mathbf{W}} , \quad \mathbf{W} \leftarrow \mathbf{W} - \eta \frac{1}{\sqrt{\mathbf{h}}} \frac{\partial L}{\partial \mathbf{W}}$$

넓어지는 폭이 점점 좁아져가는 형태.

AdaGrad

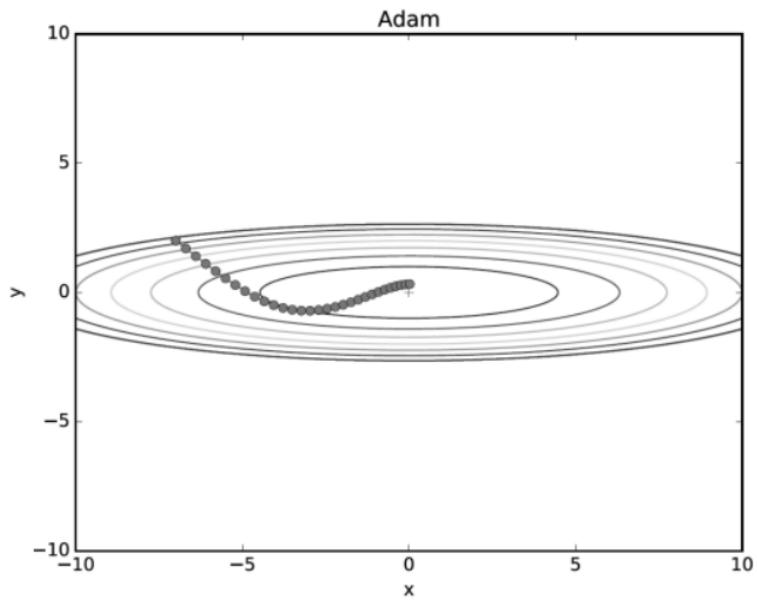


- 최소값을 향해 효율적으로 이동.
- y축 방향은 기울기가 커서 처음에는 크게 움직이지만, 그 큰 움직임에 비례해 갱신 정도도 큰 폭으로 작아지도록 조정.
- 따라서 y축 방향으로 갱신 강도가 빠르게 약해지고, 지그재그 움직임이 줄어듬.

촘촘하게 들어감

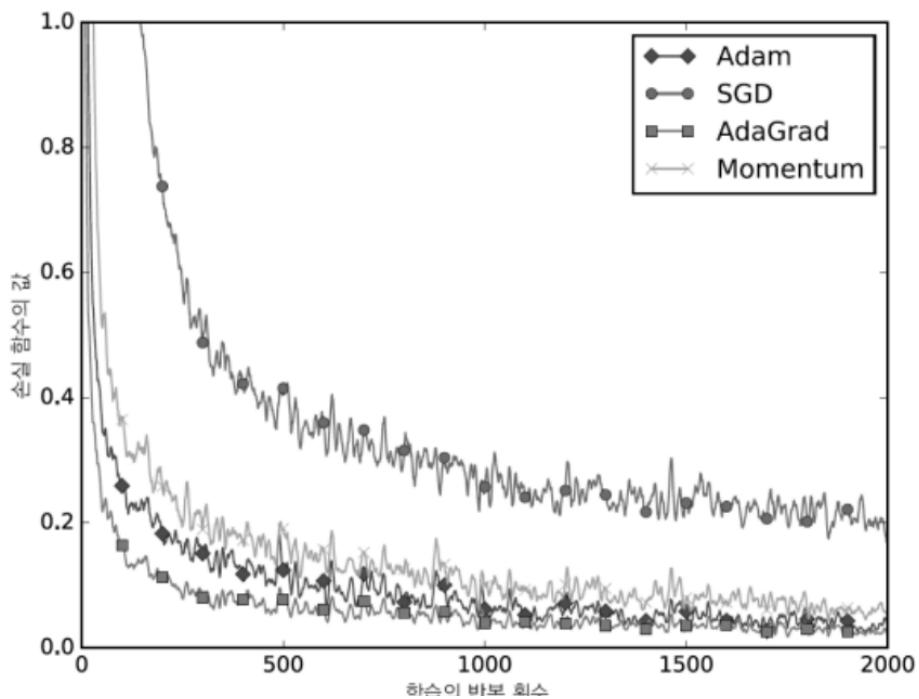
Adam

- 모멘텀과 AdaGrad를 융합한 듯한 방법으로 2015년에 제안된 새로운 방법.



- Adam에 의한 최적화 갱신 경로

MNIST 데이터셋으로 본 갱신 방법 비교

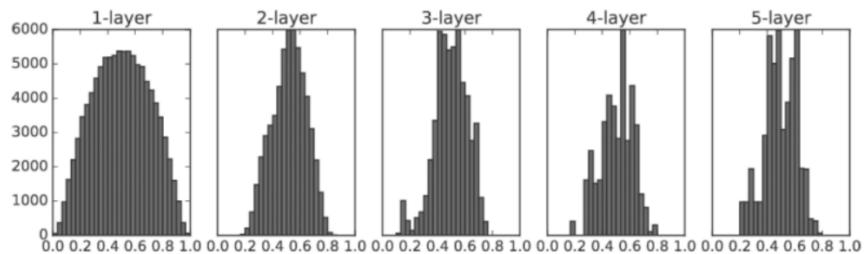


AdaGrad가 결과가 제일 좋음. Mnist에서는

활성화 함수로 sigmoid를 사용할 때 Xavier 초기값(표준편차가 $1 / \sqrt{n}$),

활성화 함수로 ReLU를 사용할 때 He 초기값 ($\sqrt{2/n}$)

은닉층의 활성화값 분포



- 가중치의 초기값으로 Xavier 초기값을 이용할 때의 각 층의 활성화값 분포.
- 각 층의 활성화 값들을 광범위하게 분포시킬 목적으로 가중치의 적절한 분포를 찾고자 함.
- 앞 계층의 노드가 n개라면 표준편차가 $1 / \sqrt{n}$ 인 분포를 사용하면 된다는 결론.
- 활성화 함수로 sigmoid를 사용할 때 Xavier 초기값 적용.
- 활성화 함수로 ReLU를 사용할 때 He 초기값($\sqrt{2/n}$) 적용.

딥러닝

03_fashionMNIST_DNN

```
4]: from tensorflow import keras  
  
(train_input, train_target), (test_input, test_target) #  
= keras.datasets.fashion_mnist.load_data()  
  
6]: print(train_input.shape, train_target.shape)  
print(test_input.shape, test_target.shape)  
  
(60000, 28, 28) (60000,)  
(10000, 28, 28) (10000,)
```

```

3]: from sklearn.model_selection import train_test_split

train_scaled = train_input / 255.0
train_scaled = train_scaled.reshape(-1, 28*28)

train_scaled, val_scaled, train_target, val_target =
    = train_test_split(train_scaled, train_target, test_size=0.2, random_state=42)

1]: dense1 = keras.layers.Dense(units=100, activation='sigmoid', input_shape=(28*28,))
dense2 = keras.layers.Dense(units=10, activation='softmax')

```

변수로 담아서 생성자에서 실행

```

2]: # 실증 신경망 만들기
model = keras.Sequential([dense1, dense2])

```

```

: model.summary()

Model: "sequential_2"
-----  

Layer (type)          Output Shape         Param #
-----  

dense_2 (Dense)      (None, 100)           78500  

dense_3 (Dense)      (None, 10)            1010  

-----  

Total params: 79,510
Trainable params: 79,510
Non-trainable params: 0
-----
```

```

: # 층을 추가하는 다른 방법2
model = keras.Sequential([
    keras.layers.Dense(units=100, activation='sigmoid', input_shape = (28 * 28,), name='hidden'),
    keras.layers.Dense(units=10, activation='softmax', name='output')
], name='패션 MNIST 모델')

```

```

: model.summary()

Model: "패션 MNIST 모델"
-----  

Layer (type)          Output Shape         Param #
-----  

hidden (Dense)        (None, 100)           78500  

output (Dense)        (None, 10)            1010  

-----  

Total params: 79,510
Trainable params: 79,510
Non-trainable params: 0
-----
```

```
21]: # 층을 추가하는 다른 방법3
model = keras.Sequential()
model.add(keras.layers.Dense(units=100, activation='sigmoid', input_shape=(28 * 28,), name='hidden'))
model.add(keras.layers.Dense(units=10, activation='softmax', name='output'))
```

```
22]: model.summary()
```

```
Model: "sequential_3"
```

| Layer (type) | Output Shape | Param # |
|--------------------------|--------------|---------|
| hidden (Dense) | (None, 100) | 78500 |
| output (Dense) | (None, 10) | 1010 |
| <hr/> | | |
| Total params: 79,510 | | |
| Trainable params: 79,510 | | |
| Non-trainable params: 0 | | |

```
: model.compile(optimizer='sgd', loss='sparse_categorical_crossentropy', metrics='accuracy')
```

```
: model.fit(train_scaled, train_target, epochs=100)
```

```
Epoch 97/100
1500/1500 [=====] - 2s 1ms/step - loss: 0.3303 - accuracy: 0.8819
Epoch 98/100
1500/1500 [=====] - 2s 1ms/step - loss: 0.3294 - accuracy: 0.8820
Epoch 99/100
1500/1500 [=====] - 2s 1ms/step - loss: 0.3286 - accuracy: 0.8824
Epoch 100/100
1500/1500 [=====] - 2s 1ms/step - loss: 0.3280 - accuracy: 0.8835
Epoch 96/100
1500/1500 [=====] - 2s 1ms/step - loss: 0.3274 - accuracy: 0.8831
Epoch 97/100
1500/1500 [=====] - 2s 1ms/step - loss: 0.3264 - accuracy: 0.8833
Epoch 98/100
1500/1500 [=====] - 4s 3ms/step - loss: 0.3258 - accuracy: 0.8840
Epoch 99/100
1500/1500 [=====] - 3s 2ms/step - loss: 0.3251 - accuracy: 0.8838
Epoch 100/100
1500/1500 [=====] - 2s 1ms/step - loss: 0.3243 - accuracy: 0.8842
```

```
: <keras.callbacks.History at 0x159d1363048>
```

```
: model = keras.Sequential()
model.add(keras.layers.Flatten(input_shape=(28, 28)))
model.add(keras.layers.Dense(units=100, activation='relu'))
model.add(keras.layers.Dense(units=10, activation='softmax'))
```

```
: model.summary()
```

```
Model: "sequential_8"
```

| Layer (type) | Output Shape | Param # |
|--------------------------|--------------|---------|
| flatten_3 (Flatten) | (None, 784) | 0 |
| dense_12 (Dense) | (None, 100) | 78500 |
| dense_13 (Dense) | (None, 10) | 1010 |
| <hr/> | | |
| Total params: 79,510 | | |
| Trainable params: 79,510 | | |
| Non-trainable params: 0 | | |

```
: from sklearn.model_selection import train_test_split
(train_input, train_target), (test_input, test_target) # 
    = keras.datasets.fashion_mnist.load_data()

train_scaled = train_input / 255.0

train_scaled, val_scaled, train_target, val_target # 
    = train_test_split(train_scaled, train_target, test_size=0.2, random_state=42)

model.compile(optimizer='sgd', loss='sparse_categorical_crossentropy', metrics='accuracy')

model.fit(train_scaled, train_target, epochs=50)
1500/1500 [=====] - 13s 9ms/step - loss: 0.0428 - accuracy: 0.9868
Epoch 5/50
1500/1500 [=====] - 13s 8ms/step - loss: 0.0426 - accuracy: 0.9868
Epoch 6/50
1500/1500 [=====] - 14s 10ms/step - loss: 0.0423 - accuracy: 0.9868
Epoch 7/50
1500/1500 [=====] - 14s 9ms/step - loss: 0.0423 - accuracy: 0.9869
Epoch 8/50
1500/1500 [=====] - 13s 9ms/step - loss: 0.0421 - accuracy: 0.9868
Epoch 9/50
1500/1500 [=====] - 13s 9ms/step - loss: 0.0421 - accuracy: 0.9869
Epoch 10/50
1500/1500 [=====] - 12s 8ms/step - loss: 0.0417 - accuracy: 0.9868
Epoch 11/50
1500/1500 [=====] - 15s 10ms/step - loss: 0.0417 - accuracy: 0.9872
Epoch 12/50
1500/1500 [=====] - 14s 9ms/step - loss: 0.0417 - accuracy: 0.9871
Epoch 13/50
1467/1500 [=====]>.] - ETA: 0s - loss: 0.0415 - accuracy: 0.9875

: model.evaluate(val_scaled, val_target)
375/375 [=====] - 1s 928us/step - loss: 0.3285 - accuracy: 0.8837
: [0.32853275537490845, 0.8836666941642761]
```

옵티마이저

```
[1]: model.compile(optimizer='sgd', loss='sparse_categorical_crossentropy', metrics='accuracy')

[1]: sgd = keras.optimizers.SGD(learning_rate=0.1)
      model.compile(optimizer=sgd, loss='sparse_categorical_crossentropy', metrics='accuracy')

[2]: sgd = keras.optimizers.SGD(momentum=0.9)

[3]: adagrad = keras.optimizers.Adagrad()
      model.compile(optimizer=adagrad, loss='sparse_categorical_crossentropy', metrics='accuracy')

[4]: rmsprop = keras.optimizers.RMSprop()
      model.compile(optimizer=rmsprop, loss='sparse_categorical_crossentropy', metrics='accuracy')

[5]: model = keras.Sequential()
      model.add(keras.layers.Flatten(input_shape=(28, 28)))
      model.add(keras.layers.Dense(units=100, activation='relu'))
      model.add(keras.layers.Dense(units=10, activation='softmax'))
```



```
[1]: # adam = keras.optimizers.Adam()

      model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics='accuracy')
      model.fit(train_scaled, train_target, epochs=50)

Epoch 42/50
1500/1500 [=====] - 2s 1ms/step - loss: 0.1179 - accuracy: 0.9556
Epoch 43/50
1500/1500 [=====] - 3s 2ms/step - loss: 0.1136 - accuracy: 0.9572
Epoch 44/50
1500/1500 [=====] - 2s 1ms/step - loss: 0.1132 - accuracy: 0.9585
Epoch 45/50
1500/1500 [=====] - 2s 1ms/step - loss: 0.1107 - accuracy: 0.9581
Epoch 46/50
1500/1500 [=====] - 2s 1ms/step - loss: 0.1126 - accuracy: 0.9579
Epoch 47/50
1500/1500 [=====] - 2s 1ms/step - loss: 0.1084 - accuracy: 0.9596
Epoch 48/50
1500/1500 [=====] - 2s 1ms/step - loss: 0.1048 - accuracy: 0.9611
Epoch 49/50
1500/1500 [=====] - 2s 1ms/step - loss: 0.1047 - accuracy: 0.9610
Epoch 50/50
1500/1500 [=====] - 2s 1ms/step - loss: 0.1018 - accuracy: 0.9619

[1]: <keras.callbacks.History at 0x22ae89e8488>

[1]: model.evaluate(val_scaled, val_target)

375/375 [=====] - 1s 4ms/step - loss: 0.5525 - accuracy: 0.8892

[1]: [0.5524566769599915, 0.8892499804496765]
```

04_fashionMNIST_Xavier

```
|: from tensorflow import keras
|(train_input, train_target), (test_input, test_target) # 
|= keras.datasets.fashion_mnist.load_data()

|: print(train_input.shape, train_target.shape, test_input.shape, test_target.shape)
|(60000, 28, 28) (60000,) (10000, 28, 28) (10000,)

|: from sklearn.model_selection import train_test_split
|train_scaled = train_input / 255.0
|train_scaled = train_scaled.reshape(-1, 28 * 28)
|train_scaled, val_scaled, train_target, val_target # 
|= train_test_split(train_scaled, train_target, test_size=0.2, random_state=42)

|: model = keras.Sequential()
|model.add(keras.layers.Dense(units=512, activation='relu', #
|                           input_shape=(28*28), #
|                           kernel_initializer='glorot_normal')) # api에서 제공해주는 것을 kernel이라고 함.
|model.add(keras.layers.Dense(units=512, activation='relu', #
|                           kernel_initializer='glorot_normal'))
|model.add(keras.layers.Dense(units=512, activation='relu', #
|                           kernel_initializer='glorot_normal'))
|model.add(keras.layers.Dense(units=512, activation='relu', #
|                           kernel_initializer='glorot_normal'))
|model.add(keras.layers.Dense(units=10, activation='softmax', #
|                           kernel_initializer='glorot_normal'))

|: model.compile(optimizer=keras.optimizers.Adam(learning_rate=0.001), loss='sparse_categorical_crossentropy', metrics='accuracy')
```

```
|: model.summary()
```

Model: "sequential_3"

| Layer (type) | Output Shape | Param # |
|------------------|--------------|---------|
| dense_9 (Dense) | (None, 512) | 401920 |
| dense_10 (Dense) | (None, 512) | 262656 |
| dense_11 (Dense) | (None, 512) | 262656 |
| dense_12 (Dense) | (None, 512) | 262656 |
| dense_13 (Dense) | (None, 10) | 5130 |

```
=====
Total params: 1,195,018
Trainable params: 1,195,018
Non-trainable params: 0
=====
```

```

: model.fit(train_scaled, train_target, epochs=50, batch_size=100)
Epoch 42/50
480/480 [=====] - 3s 7ms/step - loss: 0.0819 - accuracy: 0.9687
Epoch 43/50
480/480 [=====] - 3s 7ms/step - loss: 0.0762 - accuracy: 0.9711
Epoch 44/50
480/480 [=====] - 3s 7ms/step - loss: 0.0791 - accuracy: 0.9705
Epoch 45/50
480/480 [=====] - 3s 7ms/step - loss: 0.0724 - accuracy: 0.9720
Epoch 46/50
480/480 [=====] - 3s 7ms/step - loss: 0.0803 - accuracy: 0.9699
Epoch 47/50
480/480 [=====] - 3s 7ms/step - loss: 0.0674 - accuracy: 0.9737
Epoch 48/50
480/480 [=====] - 3s 7ms/step - loss: 0.0660 - accuracy: 0.9753
Epoch 49/50
480/480 [=====] - 3s 7ms/step - loss: 0.0731 - accuracy: 0.9720
Epoch 50/50
480/480 [=====] - 3s 7ms/step - loss: 0.0690 - accuracy: 0.9741

: <keras.callbacks.History at 0x1f6c2625888>

: model.evaluate(val_scaled, val_target)
375/375 [=====] - 1s 2ms/step - loss: 0.5528 - accuracy: 0.8916
: [0.5527626276016235, 0.8915833234786987]

```

05_fashionMNIST_Dropout

```

[2]: from tensorflow import keras
      (train_input, train_target), (test_input, test_target) # 
      = keras.datasets.fashion_mnist.load_data()

[3]: print(train_input.shape, train_target.shape, test_input.shape, test_target.shape)
(60000, 28, 28) (60000,) (10000, 28, 28) (10000,)

[4]: from sklearn.model_selection import train_test_split
      train_scaled = train_input / 255.0
      train_scaled = train_scaled.reshape(-1, 28 * 28)
      train_scaled, val_scaled, train_target, val_target # 
      = train_test_split(train_scaled, train_target, test_size=0.2, random_state=42)

```

```

model = keras.Sequential()
model.add(keras.layers.Dense(units=512, activation='relu', #
                            input_shape=(28*28,), #
                            kernel_initializer='glorot_normal')) # api에서 제공해주는 것을 kernel이라고 함.
model.add(keras.layers.Dropout(0.3)) # 현재 유닛에 30%를 dropou트해버림. 70% 데이터만 사용함.
model.add(keras.layers.Dense(units=512, activation='relu', #
                            kernel_initializer='glorot_normal'))
model.add(keras.layers.Dropout(0.3))
model.add(keras.layers.Dense(units=512, activation='relu', #
                            kernel_initializer='glorot_normal'))
model.add(keras.layers.Dropout(0.3))
model.add(keras.layers.Dense(units=512, activation='relu', #
                            kernel_initializer='glorot_normal'))
model.add(keras.layers.Dropout(0.3))
model.add(keras.layers.Dense(units=10, activation='softmax', #
                            kernel_initializer='glorot_normal'))

model.compile(optimizer=keras.optimizers.Adam(learning_rate=0.001), loss='sparse_categorical_crossentropy', metrics='accuracy')

```

[7]: model.summary()

Model: "sequential"

| Layer (type) | Output Shape | Param # |
|-----------------------------|--------------|---------|
| <hr/> | | |
| dense (Dense) | (None, 512) | 401920 |
| dropout (Dropout) | (None, 512) | 0 |
| dense_1 (Dense) | (None, 512) | 262656 |
| dropout_1 (Dropout) | (None, 512) | 0 |
| dense_2 (Dense) | (None, 512) | 262656 |
| dropout_2 (Dropout) | (None, 512) | 0 |
| dense_3 (Dense) | (None, 512) | 262656 |
| dropout_3 (Dropout) | (None, 512) | 0 |
| dense_4 (Dense) | (None, 10) | 5130 |
| <hr/> | | |
| Total params: 1,195,018 | | |
| Trainable params: 1,195,018 | | |
| Non-trainable params: 0 | | |
| <hr/> | | |

```
] model.fit(train_scaled, train_target, epochs=50, batch_size=100)
Epoch 42/50
480/480 [=====] - 4s 9ms/step - loss: 0.2356 - accuracy: 0.9137
Epoch 43/50
480/480 [=====] - 4s 9ms/step - loss: 0.2389 - accuracy: 0.9131
Epoch 44/50
480/480 [=====] - 4s 9ms/step - loss: 0.2369 - accuracy: 0.9138
Epoch 45/50
480/480 [=====] - 4s 9ms/step - loss: 0.2319 - accuracy: 0.9166
Epoch 46/50
480/480 [=====] - 4s 9ms/step - loss: 0.2266 - accuracy: 0.9184
Epoch 47/50
480/480 [=====] - 4s 9ms/step - loss: 0.2331 - accuracy: 0.9153
Epoch 48/50
480/480 [=====] - 5s 10ms/step - loss: 0.2315 - accuracy: 0.9153
Epoch 49/50
480/480 [=====] - 5s 10ms/step - loss: 0.2251 - accuracy: 0.9175
Epoch 50/50
480/480 [=====] - 4s 9ms/step - loss: 0.2269 - accuracy: 0.9167
] <keras.callbacks.History at 0x26738b51c48>
] model.evaluate(val_scaled, val_target)
375/375 [=====] - 1s 2ms/step - loss: 0.3236 - accuracy: 0.8963
] [0.32358816266059875, 0.8962500095367432]
```

이미지에 사용하는 것은 CNN.

RNN

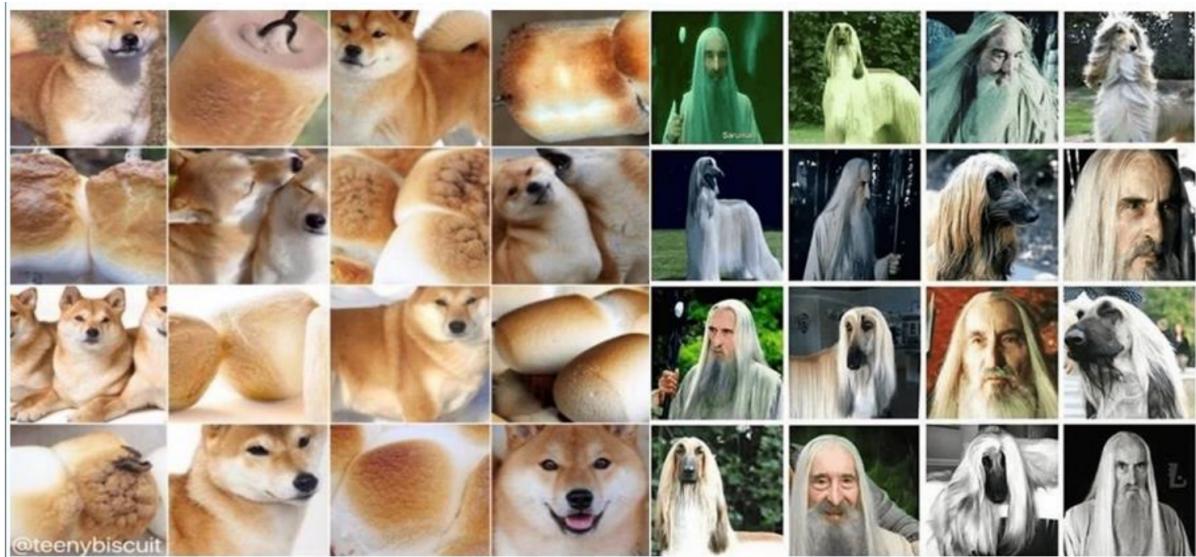
Convolutional Neural Network(합성곱 신경망).

Affine은 행렬 곱해주는 곱셈.(내적곱)

그 내적곱이 Convolution으로 부를 뿐.

Convolutional 은 이미지 처리하는 합성곱 연산이다.





데이터로 pixel을 훑어놓는 것이 아니라 이미지의 특징을 살려주는 CNN.

합성곱 계층

> 완전 연결 계층의 문제점.

- 데이터의 형상이 무시.
- 입력 데이터가 이미지인 경우, 이미지는 3차원(가로, 세로, 채널(색상))으로 구성된 데이터이나 1차원으로 평탄화 해줘야 함.
 - MNIST 데이터셋(1채널, 가로: 28픽셀, 세로: 28픽셀).
 - 형상을 무시하고 모든 입력 데이터를 동등한 뉴런(같은 차원의 뉴런)으로 취급하여 형상에 담긴 정보를 살릴 수 없음.

