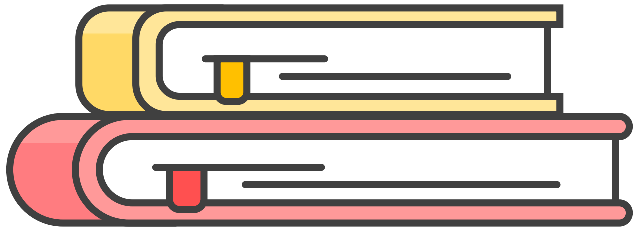
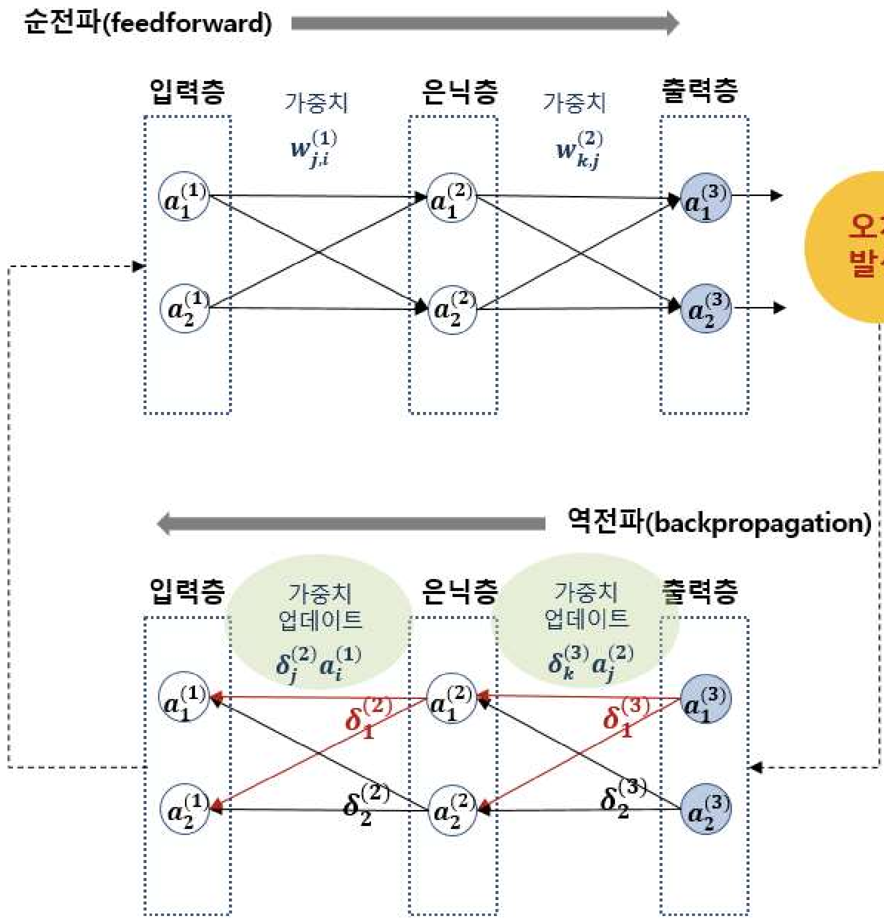


DIVE INTO DEEP LEARNING

2021210088 허지혜

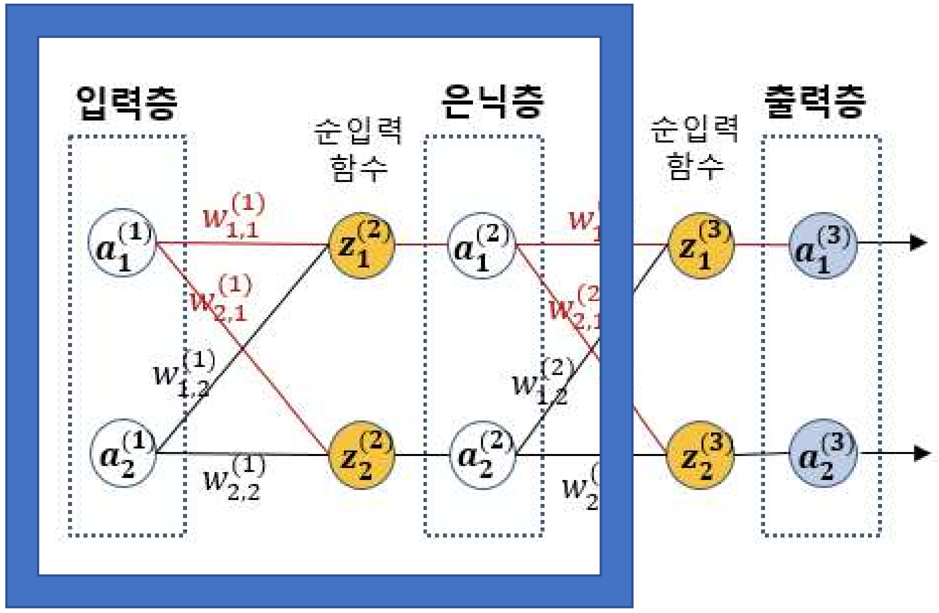
4.7 순전파, 역전파



순서

1. 순전파
2. 오차를 각 가중치로 미분한 값을 기존 가중치에서 뺀다.
3. 2번을 모든 가중치에 적용
4. 학습 횟수 또는 허용오차값에 도달할 때까지 반복

4.7 순전파, 역전파



- 활성화 함수

$$\phi(z) = \frac{1}{1+e^{-z}}$$

- 입력층 -> 은닉층 값

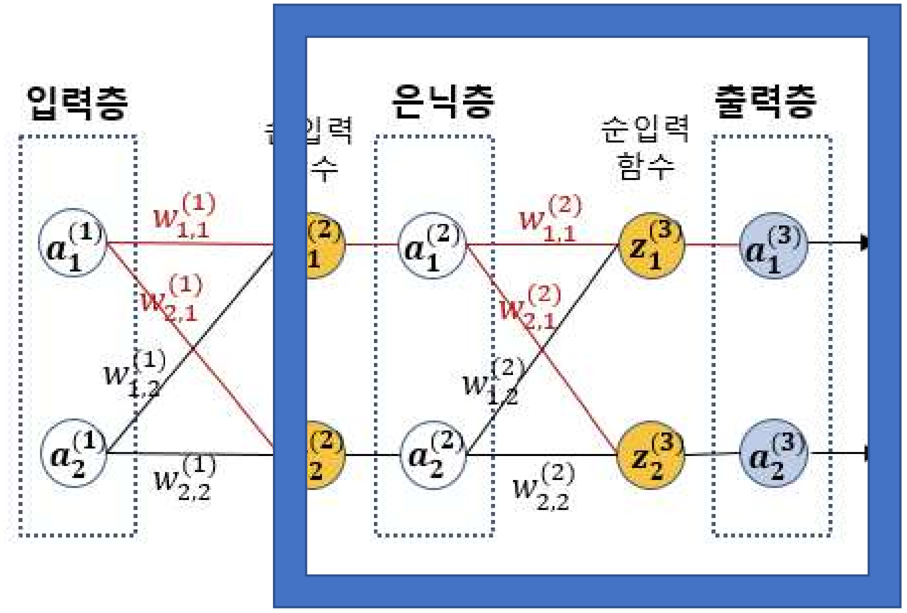
$$z_1^{(2)} = w_{1,1}^{(1)}a_1^{(1)} + w_{1,2}^{(1)}a_2^{(1)}$$

$$z_2^{(2)} = w_{2,1}^{(1)}a_1^{(1)} + w_{2,2}^{(1)}a_2^{(1)}$$

$$a_1^{(2)} = \phi(z_1^{(2)})$$

$$a_2^{(2)} = \phi(z_2^{(2)})$$

4.7 순전파, 역전파



- 활성화 함수

$$\phi(z) = \frac{1}{1+e^{-z}}$$

- 은닉층 -> 출력층 값

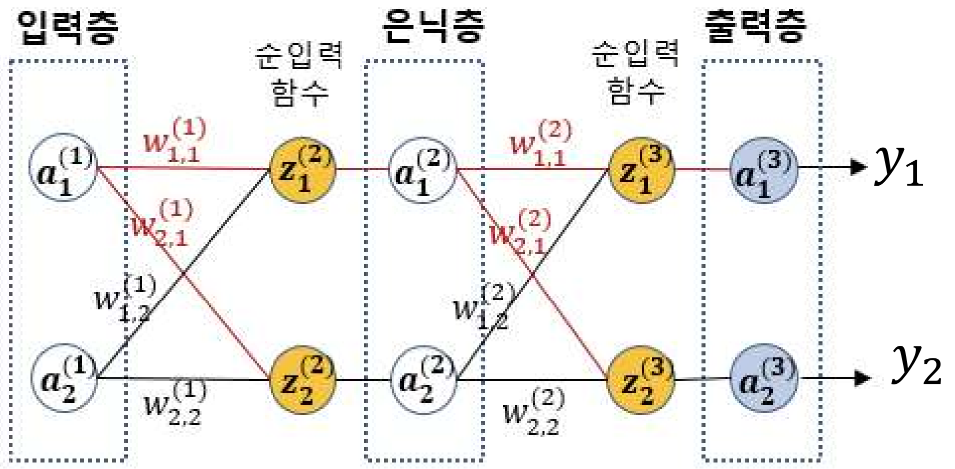
$$z_1^{(3)} = w_{1,1}^{(2)} a_1^{(2)} + w_{1,2}^{(2)} a_2^{(2)}$$

$$z_2^{(3)} = w_{2,1}^{(2)} a_1^{(2)} + w_{2,2}^{(2)} a_2^{(2)}$$

$$a_1^{(3)} = \phi(z_1^{(3)})$$

$$a_2^{(3)} = \phi(z_2^{(3)})$$

4.7 순전파, 역전파



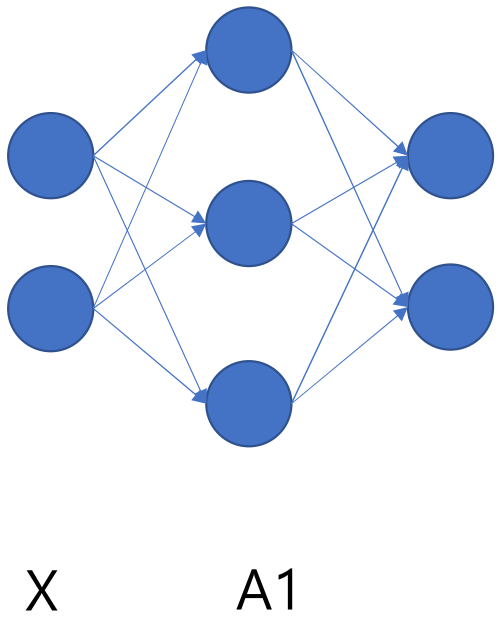
- 목적 함수

$$J_1 = \frac{1}{2}\left(a_1^{(3)} - y_1\right)^2$$

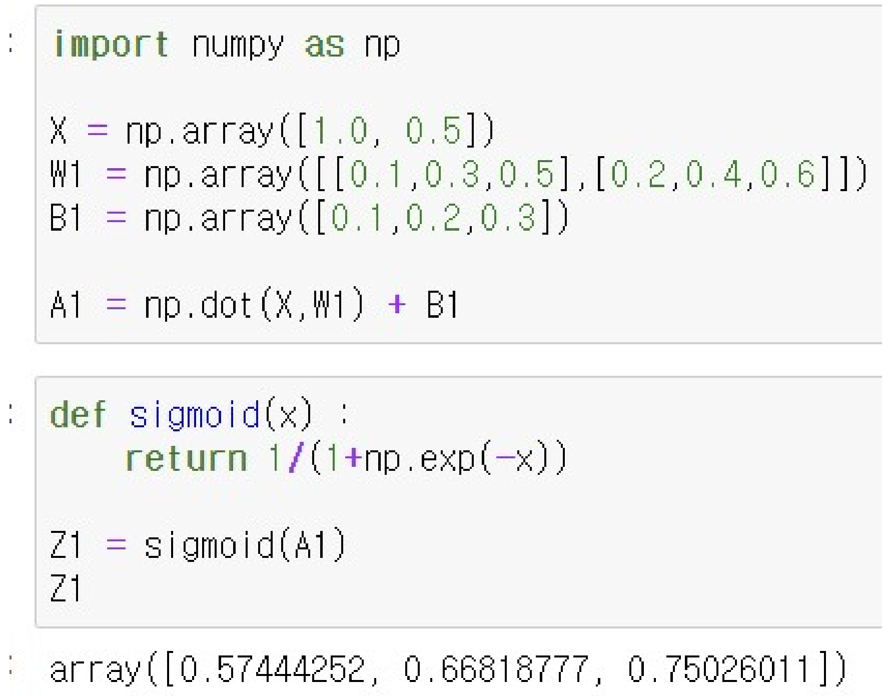
$$J_2 = \frac{1}{2}\left(a_2^{(3)} - y_2\right)^2$$

4.7 순전파, 역전파

- 순전파 코드



INPUT -> HIDDEN1



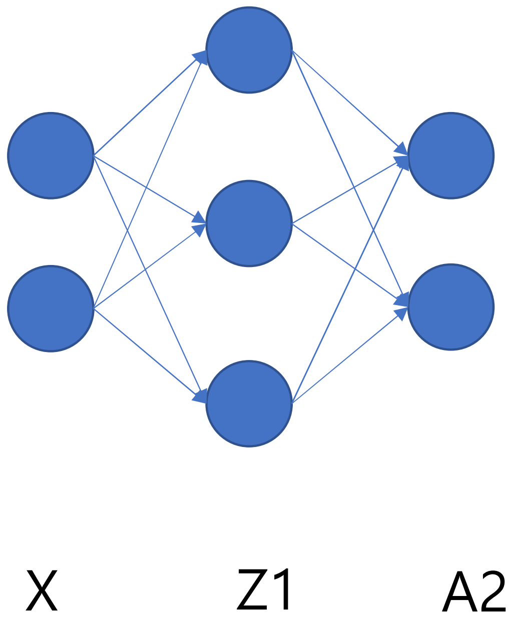
```
import numpy as np

X = np.array([1.0, 0.5])
W1 = np.array([[0.1,0.3,0.5],[0.2,0.4,0.6]])
B1 = np.array([0.1,0.2,0.3])

A1 = np.dot(X,W1) + B1
```

```
def sigmoid(x) :
    return 1/(1+np.exp(-x))

Z1 = sigmoid(A1)
Z1
```

```
array([0.57444252, 0.66818777, 0.75026011])
```

4.7 순전파, 역전파

- 순전파 코드



HIDDEN1 -> OUTPUT

```
W2 = np.array([[0.1,0.4],[0.2,0.5],[0.3,0.6]])
B2 = np.array([0.1,0.2])

A2 = np.dot(Z1,W2) + B2
Z2 = sigmoid(A2)
```

```
Z2
```

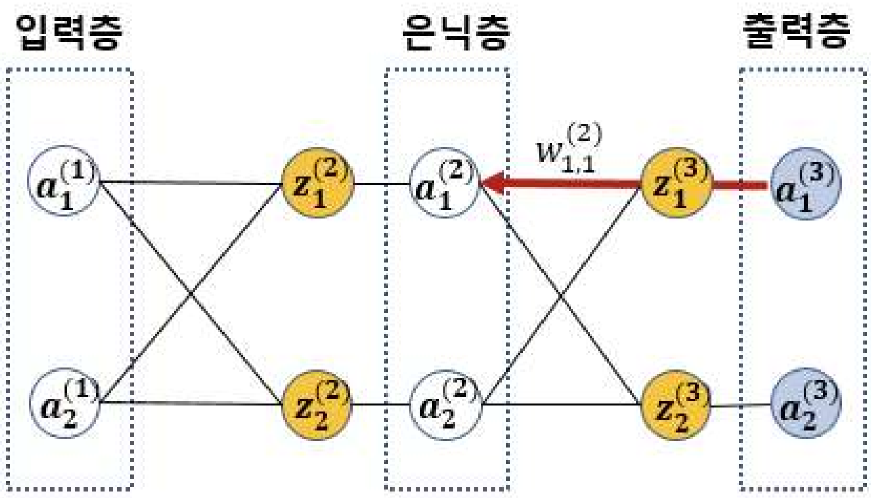
```
array([0.62624937, 0.7710107 ])
```

```
# 분류문제라고 생각 안하고 적용
def identity_function(x):
    return x

W3 = np.array([[0.1,0.3],[0.2,0.4]])
B3 = np.array([0.1,0.2])

A3 = np.dot(Z2,W3) + B3
Y = identity_function(A3)
```

```
print(Y)
```

```
[0.31682708 0.69627909]
```

4.7 순전파, 역전파



- 가중치 업데이트 식

$$\boldsymbol{w_j} = \boldsymbol{w_j} - \boldsymbol{\eta} \frac{\boldsymbol{\partial J_{total}}}{\boldsymbol{\partial w_j}} \qquad \eta : learning\ rate$$

$$w_{1,1}^{(2)} = w_{1,1}^{(2)} - \frac{\partial J_{total}}{\partial w_{1,1}^{(2)}}$$

$$\frac{\partial J_{total}}{\partial w_{1,1}^{(2)}} = \frac{\partial J_1}{\partial a_1^{(3)}} \times \frac{\partial a_1^{(3)}}{\partial z_1^{(3)}} \times \frac{\partial z_1^{(3)}}{\partial w_{1,1}^{(2)}}$$ (Chain rule)

4.7 순전파, 역전파

$$\frac{\partial J_{total}}{\partial w_{1,1}^{(2)}} = \frac{\partial J_1}{\partial a_1^{(3)}} \times \frac{\partial a_1^{(3)}}{\partial z_1^{(3)}} \times \frac{\partial z_1^{(3)}}{\partial w_{1,1}^{(2)}}$$

$$\frac{\partial J_{total}}{\partial a_1^{(3)}} = \frac{\partial J_1}{\partial a_1^{(3)}} = \frac{1}{2}\frac{\partial}{\partial a_1^{(3)}}\left(a_1^{(3)} - y_1\right)^2 = \left(a_1^{(3)} - y_1\right)$$

$$\frac{\partial a_1^{(3)}}{\partial z_1^{(3)}} = \phi\left(z_1^{(3)}\right)\left(1 - \phi\left(z_1^{(3)}\right)\right) = a_1^{(3)}\left(1 - a_1^{(3)}\right)$$

$$\frac{\partial z_1^{(3)}}{\partial w_{1,1}^{(2)}} = a_1^{(2)}$$

4.7 순전파, 역전파

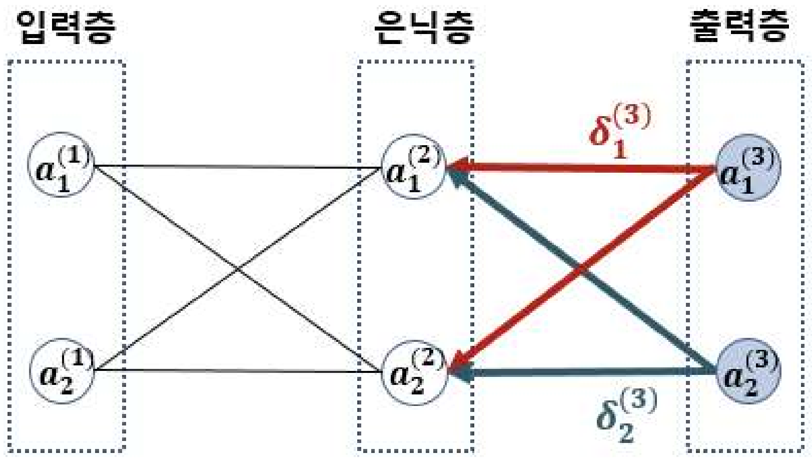
$$\frac{\partial J_{total}}{\partial w_{1,1}^{(2)}} = \frac{\partial J_1}{\partial a_1^{(3)}} \times \frac{\partial a_1^{(3)}}{\partial z_1^{(3)}} \times \frac{\partial z_1^{(3)}}{\partial w_{1,1}^{(2)}} = \left(a_1^{(3)} - y_1\right) \times a_1^{(3)}\left(1 - a_1^{(3)}\right) \times a_1^{(2)}$$

4.7 순전파, 역전파

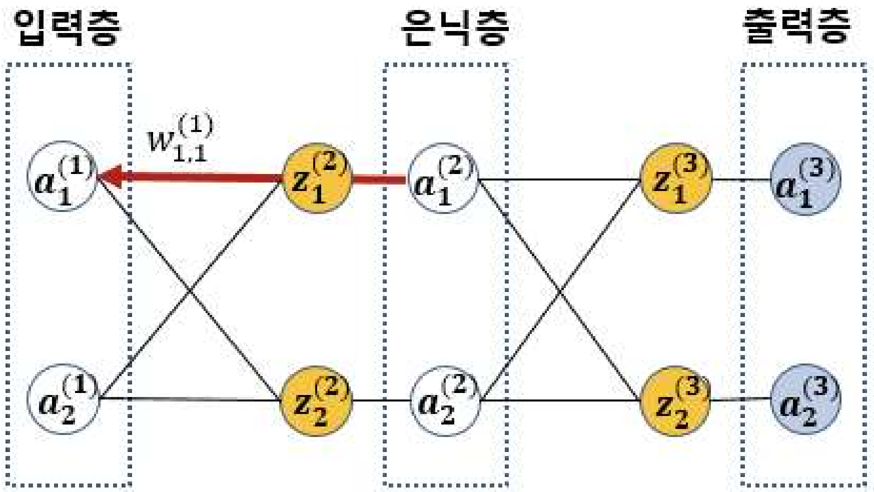
$$\delta_1^{(3)} = \frac{\partial J_1}{\partial z_1^{(3)}} = \left(a_1^{(3)} - y_1\right) \times a_1^{(3)}\left(1 - a_1^{(3)}\right) \qquad \delta_2^{(3)} = \frac{\partial J_2}{\partial z_2^{(3)}} = \left(a_2^{(3)} - y_2\right) \times a_2^{(3)}\left(1 - a_2^{(3)}\right)$$

$$w_{1,1}^{(2)} = w_{1,1}^{(2)} - \delta_1^{(3)} a_1^{(2)} \qquad w_{2,1}^{(2)} = w_{2,1}^{(2)} - \delta_2^{(3)} a_1^{(2)}$$

$$w_{1,2}^{(2)} = w_{1,2}^{(2)} - \delta_1^{(3)} a_2^{(2)} \qquad w_{2,2}^{(2)} = w_{2,2}^{(2)} - \delta_2^{(3)} a_2^{(2)}$$



4.7 순전파, 역전파



$$\frac{\partial J_{total}}{\partial w_{1,1}^{(1)}} = \frac{\partial J_{total}}{\partial a_1^{(2)}} \times \frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} \times \frac{\partial z_1^{(2)}}{\partial w_{1,1}^{(1)}}$$

$$\frac{\partial J_{total}}{\partial a_1^{(2)}} = \frac{\partial J_1}{\partial a_1^{(2)}} + \frac{\partial J_2}{\partial a_1^{(2)}} = \frac{\partial J_1}{\partial z_1^{(3)}} \times \frac{\partial z_1^{(3)}}{\partial a_1^{(2)}} + \frac{\partial J_2}{\partial z_2^{(3)}} \times \frac{\partial z_2^{(3)}}{\partial a_1^{(2)}}$$

4.7 순전파, 역전파

$$\frac{\partial J_{total}}{\partial a_1^{(2)}} = \frac{\partial J_1}{\partial a_1^{(2)}} + \frac{\partial J_2}{\partial a_1^{(2)}} = \underset{\delta_1^{(3)}}{\underset{\downarrow}{\frac{\partial J_1}{\partial z_1^{(3)}}}} \times \underset{w_{1,1}^{(2)}}{\underset{\downarrow}{\frac{\partial z_1^{(3)}}{\partial a_1^{(2)}}}} + \underset{\delta_2^{(3)}}{\underset{\downarrow}{\frac{\partial J_2}{\partial z_2^{(3)}}}} \times \underset{w_{2,1}^{(2)}}{\underset{\downarrow}{\frac{\partial z_2^{(3)}}{\partial a_1^{(2)}}}}$$

$$\frac{\partial J_{total}}{\partial a_1^{(2)}} = \delta_1^{(3)} w_{1,1}^{(2)} + \delta_2^{(3)} w_{2,1}^{(2)}$$

4.7 순전파, 역전파

구해야 하는 거 : $\frac{\partial J_{total}}{\partial w_{1,1}^{(1)}} = \frac{\partial J_{total}}{\partial a_1^{(2)}} \times \frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} \times \frac{\partial z_1^{(2)}}{\partial w_{1,1}^{(1)}}$

$$\Downarrow$$

$$\frac{\partial J_{total}}{\partial a_1^{(2)}} = \delta_1^{(3)} w_{1,1}^{(2)} + \delta_2^{(3)} w_{2,1}^{(2)}$$

$$\Downarrow$$

$$\frac{\partial J_{total}}{\partial w_{1,1}^{(1)}} = \underbrace{\left(\delta_1^{(3)} w_{1,1}^{(2)} + \delta_2^{(3)} w_{2,1}^{(2)}\right) \times a_1^{(2)} \left(1 - a_1^{(2)}\right)}_{\delta_1^{(2)}} \times a_1^{(1)}$$

4.7 순전파, 역전파

$$\delta_1^{(2)} = \left(\delta_1^{(3)} w_{1,1}^{(2)} + \delta_2^{(3)} w_{2,1}^{(2)}\right) \times a_1^{(2)} \left(1 - a_1^{(2)}\right)$$

$$\delta_2^{(2)} = \left(\delta_1^{(3)} w_{1,2}^{(2)} + \delta_2^{(3)} w_{2,2}^{(2)}\right) \times a_2^{(2)} \left(1 - a_2^{(2)}\right)$$

$$w_{1,1}^{(1)} = w_{1,1}^{(1)} - \delta_1^{(2)} a_1^{(1)}$$

$$w_{1,2}^{(1)} = w_{1,2}^{(1)} - \delta_1^{(2)} a_2^{(1)}$$

$$w_{2,1}^{(1)} = w_{2,1}^{(1)} - \delta_2^{(2)} a_1^{(1)}$$

$$w_{2,2}^{(1)} = w_{2,2}^{(1)} - \delta_2^{(2)} a_2^{(1)}$$

$$\boldsymbol{w_{j,i}^{(l)} = w_{j,i}^{(l)} - \delta_j^{(l+1)} a_i^{(l)}}$$

$$\boldsymbol{\delta_j^{(3)} = \left(a_j^{(3)} - y_j\right) \times a_j^{(3)} \left(1 - a_j^{(3)}\right)}$$

$$\boldsymbol{\delta_j^{(2)} = \left(\delta_1^{(3)} w_{1,j}^{(2)} + \delta_2^{(3)} w_{2,j}^{(2)}\right) \times a_j^{(2)} \left(1 - a_j^{(2)}\right)}$$

4.7 순전파, 역전파

이렇게
순전파 -> 역전파 -> 가중치 업데이트 -> 순전파 ... 순으로 반복해가면서
오차값이 0과 가까워지게 된다.

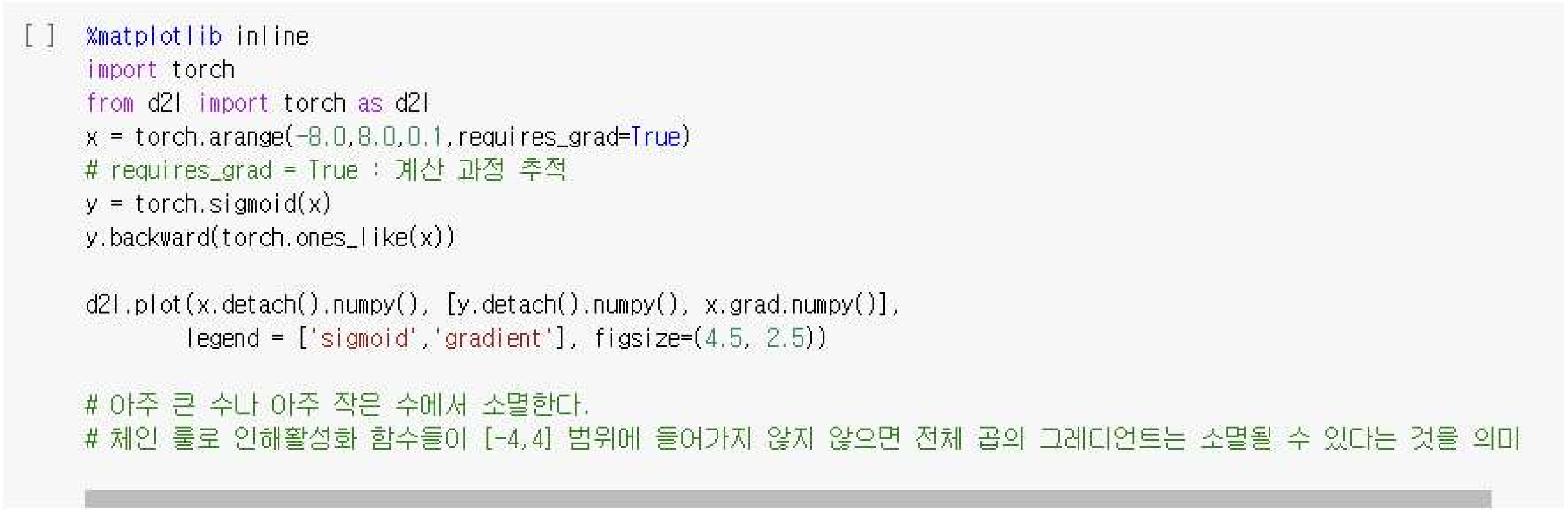
4.8 수치적 안정성 및 초기화

활성화 함수 선택이 얕은 네트워크에서는 크게 중요하지 않지만
딥 네트워크에서는 비선형성과 초기화 선택이 최적화 알고리즘을 빠르게
수렴시키는데 중요한 역할을 한다.

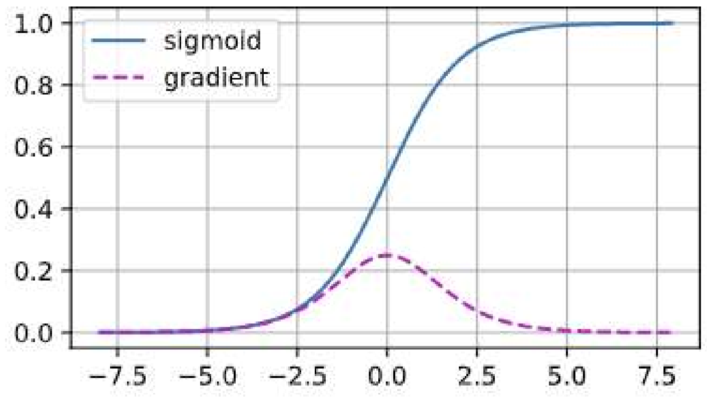
위를 중요하게 생각하지 않으면
그레디언트 소실과 폭발이 발생할 수 있다.

4.8 수치적 안정성 및 초기화

- 기울기 소실



```
[ ]  %matplotlib inline
     import torch
     from d2l import torch as d2l
     x = torch.arange(-8.0,8.0,0.1,requires_grad=True)
     # requires_grad = True : 계산 과정 추적
     y = torch.sigmoid(x)
     y.backward(torch.ones_like(x))

     d2l.plot(x.detach().numpy(), [y.detach().numpy(), x.grad.numpy()],
              legend = ['sigmoid','gradient'], figsize=(4.5, 2.5))

     # 아주 큰 수나 아주 작은 수에서 소멸한다.
     # 체인 룰로 인해활성화 함수들이 [-4,4] 범위에 들어가지 않지 않으면 전체 곱의 그레디언트는 소멸될 수 있다는 것을 의미
```



- 층을 많이 사용하는 경우,
이 현상이 어떤 층에서 일어날 가능성이 높다.

4.8 수치적 안정성 및 초기화

- 기울기 폭발

```python
M = torch.normal(0, 1, size=(4, 4))
print('a single matrix \n', M)
for i in range(100):
    M = torch.mm(M, torch.normal(0, 1, size=(4, 4)))

print('after multiplying 100 matrices\n', M)

## mm : matrix multiplication 행렬 곱
## 우리가 선택한 스케일링으로 행렬의 곱이 너무 커짐
# 그러면 알고리즘을 수렴하게 만들기 어려워진다.
```

```
a single matrix
 tensor([[ 0.0913,  1.0248, -0.6124, -0.3784],
        [ 1.3457, -0.0501, -0.9427, -0.1643],
        [ 0.2765,  0.3791, -1.9563, -0.2200],
        [ 0.2264, -0.7368, -0.0854, -1.0450]])
after multiplying 100 matrices
 tensor([[-4.0315e+22,  9.2344e+21, -7.7860e+21,  2.1396e+22],
        [-7.2791e+22,  1.6673e+22, -1.4058e+22,  3.8631e+22],
        [-6.9486e+22,  1.5916e+22, -1.3420e+22,  3.6877e+22],
        [-3.3651e+22,  7.7079e+21, -6.4990e+21,  1.7859e+22]])
```

4.8 수치적 안정성 및 초기화

위에서 제시한 문제들을 해결하거나 완화하는 방법 중 하나는 신중한 초기화이다.
최적화 및 적절한 정규화 추가로 주의를 기울이면 안정성을 향상시킬 수 있다.

4.8 수치적 안정성 및 초기화

- Default initialization

가중치 값을 초기화하기 위해 정규 분포 방법을 지정하였다.
기본 임의 초기화 방법이다.

4.8 수치적 안정성 및 초기화

- Xavier initialization

Xavier initialization 또는 Global initialization은 이전 노드와 다음 노드의 개수에 의존하는 방법이다.
Uniform 분포를 따르는 방법과 normal 분포를 따르는 방법이 있다.

위 초기화 방법은 비선형 함수에서 효과적인 결과를 보여주지만
ReLU 함수 사용시 출력값이 0으로 수렴하게 되는 현상을 확인할 수 있다.

$$W \sim N(0, Var(W))$$

$$Var(W) = \sqrt{\frac{2}{n_{in} + n_{out}}}$$

Uniform 함수

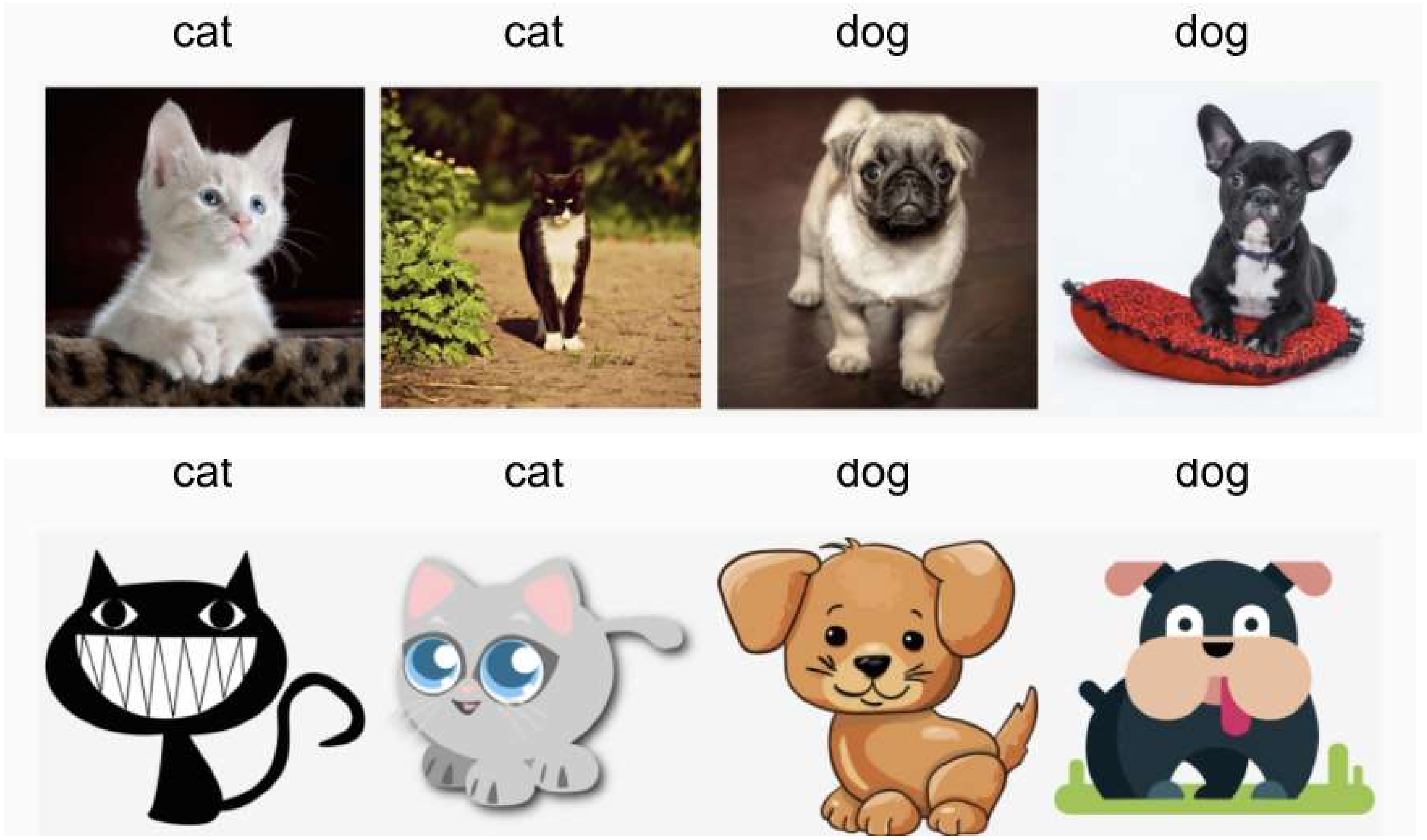
$$W \sim U(-\sqrt{\frac{6}{n_{in} + n_{out}}}, \ +\sqrt{\frac{6}{n_{in} + n_{out}}})$$

4.9 환경과 분포 이동

- Covariate Shift

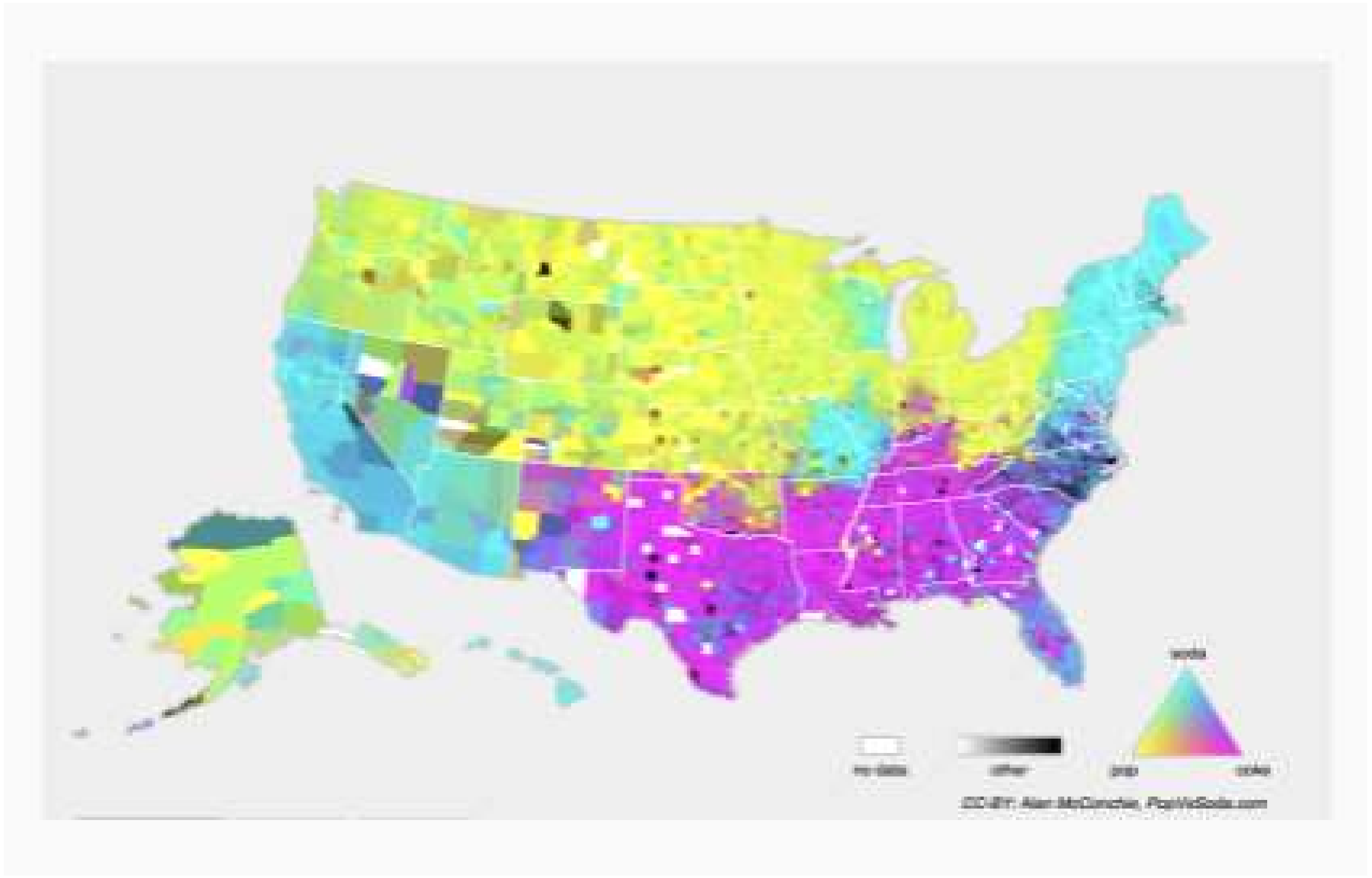
Training set

Test set



4.9 환경과 분포 이동

- Concept Shift



데이터의 출처를 지리별로 이동시키면 그림과 같이 나온다.

4.10 Predicting House Prices on Kaggle

- Downloading and Caching Datasets

```python
import hashlib
import os
import tarfile
import zipfile
import requests


DATA_HUB = dict()
DATA_URL = 'http://d2l-data.s3-accelerate.amazonaws.com/'
```

```python
def download(name, cache_dir=os.path.join('..', 'data')):
    """Download a file inserted into DATA_HUB, return the local filename."""
    assert name in DATA_HUB, f"{name} does not exist in {DATA_HUB}."
    url, sha1_hash = DATA_HUB[name]
    os.makedirs(cache_dir, exist_ok=True)
    fname = os.path.join(cache_dir, url.split('/')[-1])
    if os.path.exists(fname):
        sha1 = hashlib.sha1()
        with open(fname, 'rb') as f:
            while True:
                data = f.read(1048576)
                if not data:
                    break
                sha1.update(data)
        if sha1.hexdigest() == sha1_hash:
            return fname  # Hit cache
    print(f'Downloading {fname} from {url}...')
    r = requests.get(url, stream=True, verify=True)
    with open(fname, 'wb') as f:
        f.write(r.content)
    return fname
```

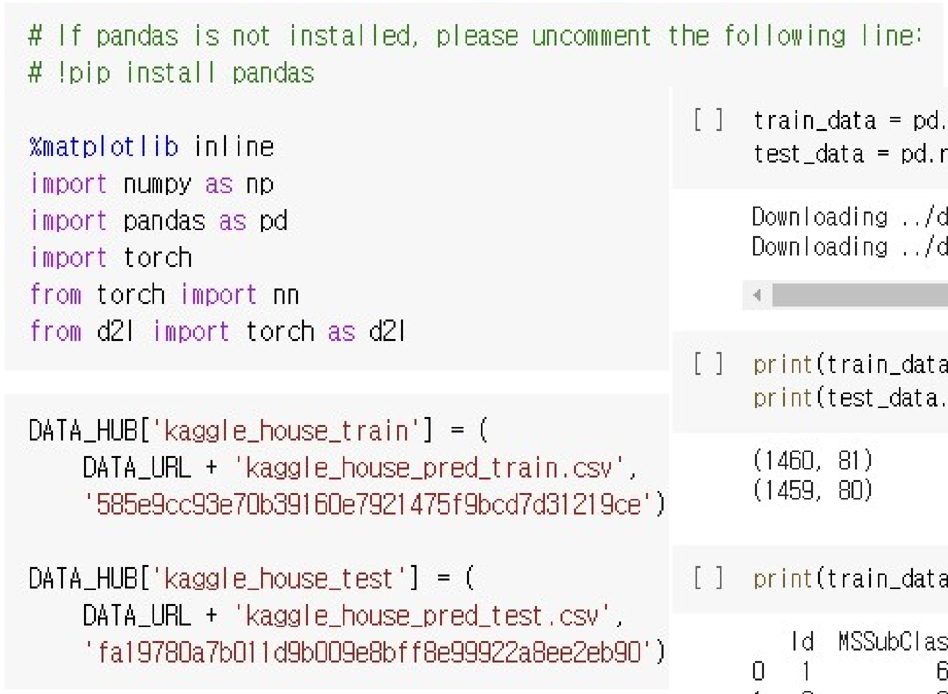
4.10 Predicting House Prices on Kaggle

- Downloading and Caching Datasets

```python
def download_extract(name, folder=None):
    """Download and extract a zip/tar file."""
    fname = download(name)
    base_dir = os.path.dirname(fname)
    data_dir, ext = os.path.splitext(fname)
    if ext == '.zip':
        fp = zipfile.ZipFile(fname, 'r')
    elif ext in ('.tar', '.gz'):
        fp = tarfile.open(fname, 'r')
    else:
        assert False, 'Only zip/tar files can be extracted.'
    fp.extractall(base_dir)
    return os.path.join(base_dir, folder) if folder else data_dir

def download_all():
    """Download all files in the DATA_HUB."""
    for name in DATA_HUB:
        download(name)
```

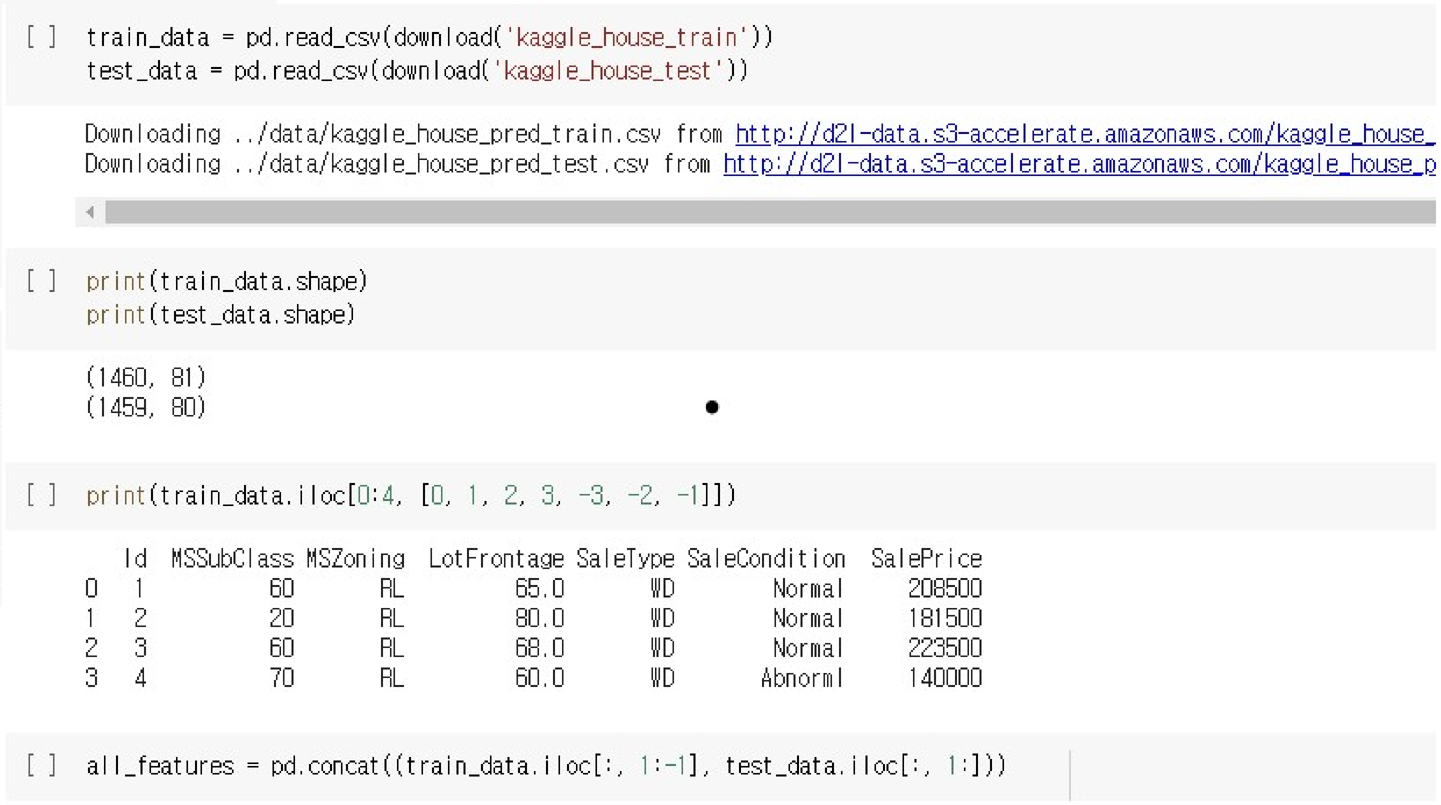
4.10 Predicting House Prices on Kaggle

- Accessing and Reading the Dataset



```
# If pandas is not installed, please uncomment the following line:
# !pip install pandas

%matplotlib inline
import numpy as np
import pandas as pd
import torch
from torch import nn
from d2l import torch as d2l
```

```
DATA_HUB['kaggle_house_train'] = (
    DATA_URL + 'kaggle_house_pred_train.csv',
    '585e9cc93e70b39160e7921475f9bcd7d31219ce')

DATA_HUB['kaggle_house_test'] = (
    DATA_URL + 'kaggle_house_pred_test.csv',
    'fa19780a7b011d9b009e8bff8e99922a8ee2eb90')
```



```
[ ]  train_data = pd.read_csv(download('kaggle_house_train'))
     test_data = pd.read_csv(download('kaggle_house_test'))
```

```
Downloading ../data/kaggle_house_pred_train.csv from http://d2l-data.s3-accelerate.amazonaws.com/kaggle_house_
Downloading ../data/kaggle_house_pred_test.csv from http://d2l-data.s3-accelerate.amazonaws.com/kaggle_house_p
```

```
[ ]  print(train_data.shape)
     print(test_data.shape)
```

```
(1460, 81)
(1459, 80)
```

```
[ ]  print(train_data.iloc[0:4, [0, 1, 2, 3, -3, -2, -1]])
```

```
   Id  MSSubClass MSZoning  LotFrontage SaleType SaleCondition  SalePrice
0   1          60       RL         65.0       WD        Normal     208500
1   2          20       RL         80.0       WD        Normal     181500
2   3          60       RL         68.0       WD        Normal     223500
3   4          70       RL         60.0       WD       Abnorml     140000
```

```
[ ]  all_features = pd.concat((train_data.iloc[:, 1:-1], test_data.iloc[:, 1:]))
```

4.10 Predicting House Prices on Kaggle

- Data Preprocessing

```
# 정규화
# If test data were inaccessible, mean and standard deviation could be
# calculated from training data
numeric_features = all_features.dtypes[all_features.dtypes != 'object'].index
all_features[numeric_features] = all_features[numeric_features].apply(
    lambda x: (x - x.mean()) / (x.std()))
# After standardizing the data all means vanish, hence we can set missing
# values to 0
all_features[numeric_features] = all_features[numeric_features].fillna(0)
```

```
# `Dummy_na=True` considers "na" (missing value) as a valid feature value, and
# creates an indicator feature for it
all_features = pd.get_dummies(all_features, dummy_na=True)
all_features.shape
```

```
(2919, 331)
```

```
n_train = train_data.shape[0]
train_features = torch.tensor(all_features[:n_train].values,
                              dtype=torch.float32)
test_features = torch.tensor(all_features[n_train:].values,
                             dtype=torch.float32)
train_labels = torch.tensor(train_data.SalePrice.values.reshape(-1, 1),
                            dtype=torch.float32)
```

4.10 Predicting House Prices on Kaggle

-Training

```
loss = nn.MSELoss()
in_features = train_features.shape[1]

def get_net():
    net = nn.Sequential(nn.Linear(in_features, 1))
    return net
```

```
def log_rmse(net, features, labels):
    # To further stabilize the value when the logarithm is taken, set the
    # value less than 1 as 1
    clipped_preds = torch.clamp(net(features), 1, float('inf'))
    rmse = torch.sqrt(loss(torch.log(clipped_preds), torch.log(labels)))
    return rmse.item()
```

```
def train(net, train_features, train_labels, test_features, test_labels,
          num_epochs, learning_rate, weight_decay, batch_size):
    train_ls, test_ls = [], []
    train_iter = d2l.load_array((train_features, train_labels), batch_size)
    # The Adam optimization algorithm is used here
    optimizer = torch.optim.Adam(net.parameters(), lr=learning_rate,
                                 weight_decay=weight_decay)
    for epoch in range(num_epochs):
        for X, y in train_iter:
            optimizer.zero_grad()
            l = loss(net(X), y)
            l.backward()
            optimizer.step()
        train_ls.append(log_rmse(net, train_features, train_labels))
        if test_labels is not None:
            test_ls.append(log_rmse(net, test_features, test_labels))
    return train_ls, test_ls
```

-K-Fold Cross-Validation

```
def get_k_fold_data(k, i, X, y):
    assert k > 1
    fold_size = X.shape[0] // k
    X_train, y_train = None, None
    for j in range(k):
        idx = slice(j * fold_size, (j + 1) * fold_size)
        X_part, y_part = X[idx, :], y[idx]
        if j == i:
            X_valid, y_valid = X_part, y_part
        elif X_train is None:
            X_train, y_train = X_part, y_part
        else:
            X_train = torch.cat([X_train, X_part], 0)
            y_train = torch.cat([y_train, y_part], 0)
    return X_train, y_train, X_valid, y_valid
```

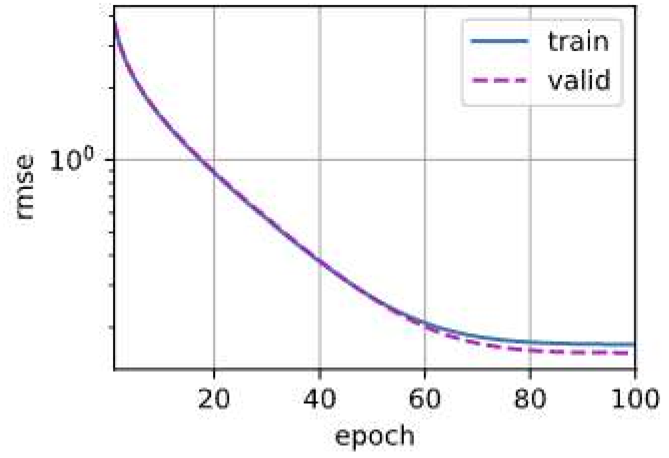
```
def k_fold(k, X_train, y_train, num_epochs, learning_rate, weight_decay,
           batch_size):
    train_l_sum, valid_l_sum = 0, 0
    for i in range(k):
        data = get_k_fold_data(k, i, X_train, y_train)
        net = get_net()
        train_ls, valid_ls = train(net, *data, num_epochs, learning_rate,
                                   weight_decay, batch_size)
        train_l_sum += train_ls[-1]
        valid_l_sum += valid_ls[-1]
        if i == 0:
            d2l.plot(list(range(1, num_epochs + 1)), [train_ls, valid_ls],
                     xlabel='epoch', ylabel='rmse', xlim=[1, num_epochs],
                     legend=['train', 'valid'], yscale='log')
        print(f'fold {i + 1}, train log rmse {float(train_ls[-1]):f}, '
              f'valid log rmse {float(valid_ls[-1]):f}')
    return train_l_sum / k, valid_l_sum / k
```

4.10 Predicting House Prices on Kaggle

-Model Selection

```
k, num_epochs, lr, weight_decay, batch_size = 5, 100, 5, 0, 64
train_l, valid_l = k_fold(k, train_features, train_labels, num_epochs, lr,
                          weight_decay, batch_size)
print(f'{k}-fold validation: avg train log rmse: {float(train_l):f}, '
      f'avg valid log rmse: {float(valid_l):f}')
```

```
fold 1, train log rmse 0.169588, valid log rmse 0.156506
fold 2, train log rmse 0.162381, valid log rmse 0.190267
fold 3, train log rmse 0.164189, valid log rmse 0.168431
fold 4, train log rmse 0.168283, valid log rmse 0.154608
fold 5, train log rmse 0.163852, valid log rmse 0.183156
5-fold validation: avg train log rmse: 0.165659, avg valid log rmse: 0.170594
```



Reference

순전파, 역전파 설명
https://m.blog.naver.com/samsjang/221033626685?view=img_75

Dataset shift, Covariate shift 설명
https://data-newbie.tistory.com/355