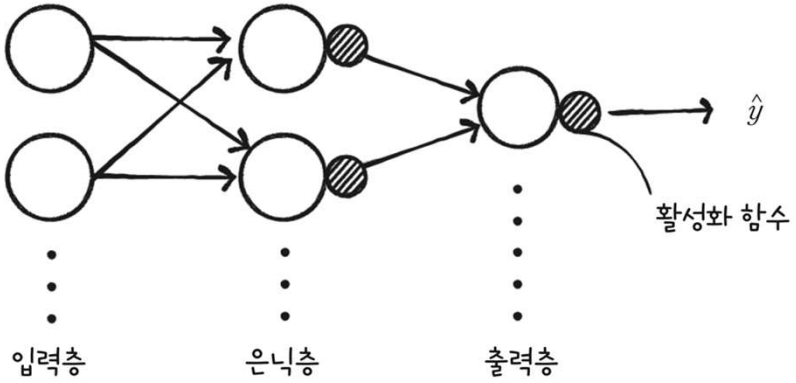
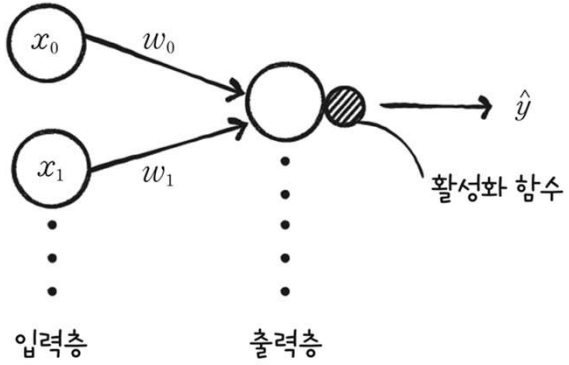
04 분류하는 뉴런을 만듭니다

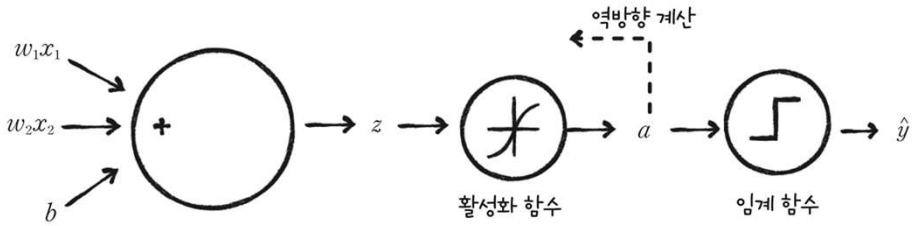
- 이진 분류(binary classification)

04-6 로지스틱 회귀 뉴런으로 단일층 신경망을 만듭니다

이미 단일층 신경망을 구현했습니다! :D







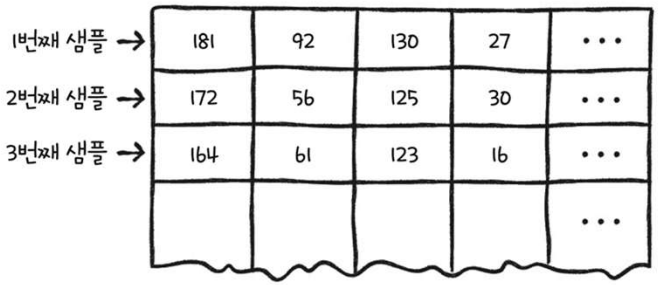
손실 함수 결괏값 저장 기능 추가하기

```
def __init__(self):
    self.w = None
    self.b = None
    self.losses = []
    ...

def fit(self, x, y, epochs=100):
        ...                                          # 이 부분은 잠시 후에 설명합니다.
```

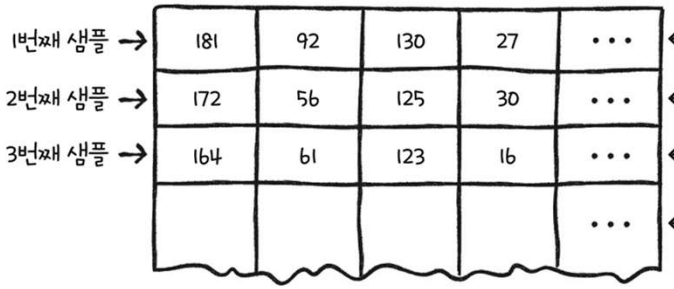
```
        for i in index:                              # 모든 샘플에 대해 반복합니다.
            z = self.forpass(x[i])                   # 정방향 계산
            a = self.activation(z)                   # 활성화 함수 적용
            err = -(y[i]  -  a)                      # 오차 계산
            w_grad, b_grad = self.backprop(x[i], err)  # 역방향 계산
            self.w -= w_grad                         # 가중치 업데이트
            self.b -= b_grad                         # 절편 업데이트
            # 안전한 로그 계산을 위해 클리핑한 후 손실을 누적합니다.
            a = np.clip(a, 1e-10, 1-1e-10)
            loss += -(y[i]*np.log(a)+(1-y[i])*np.log(1-a))
                                                     # 에포크마다 평균 손실을 저장합니다.
        self.losses.append(loss/len(y))
```

여러가지 경사 하강법



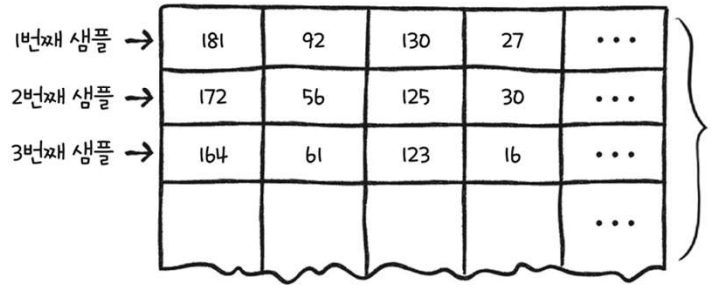
1개의 샘플을 중복되지 않도록 무작위로 선택 → 그레이디언트 계산

확률적 경사 하강법



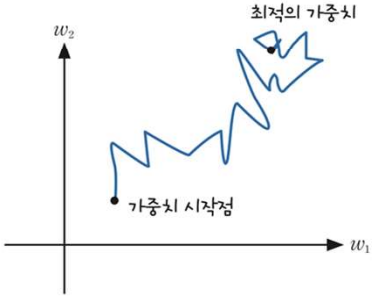
전체 샘플 중 몇 개의 샘플을 중복되지 않도록 무작위로 선택 → 그레이디언트 계산

미니 배치 경사 하강법

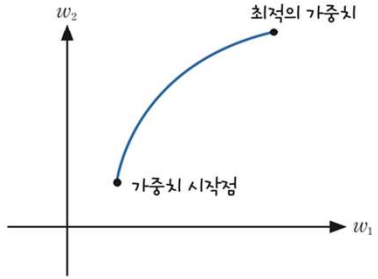


전체 샘플들 모두 선택 → 그레이디언트 계산(에포크)

배치 경사 하강법

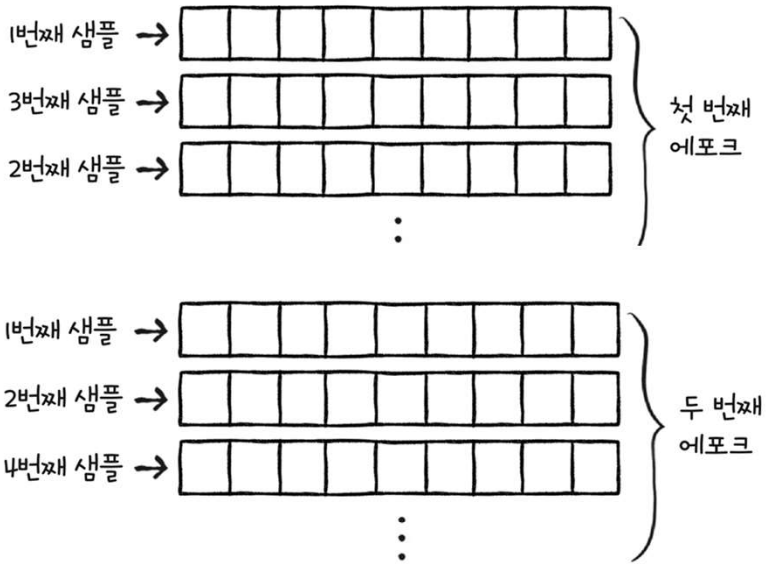


확률적 경사 하강법



배치 경사 하강법

에포크마다 훈련 샘플 섞기




```
def fit(self, x, y, epochs=100):
    self.w = np.ones(x.shape[1])                    # 가중치를 초기화합니다.
    self.b = 0                                      # 절편을 초기화합니다.
    for i in range(epochs):                         # epochs만큼 반복합니다.
        loss = 0
        indexes = np.random.permutation(np.arange(len(x)))    # 인덱스를 섞습니다.
        for i in indexes:                           # 모든 샘플에 대해 반복합니다.
            z = self.forpass(x[i])                  # 정방향 계산
            a = self.activation(z)                  # 활성화 함수 적용
            err = -(y[i] - a)                       # 오차 계산
            w_grad, b_grad = self.backprop(x[i], err) # 역방향 계산
            self.w -= w_grad                        # 가중치 업데이트
            self.b -= b_grad                        # 절편 업데이트
            a = np.clip(a, 1e-10, 1-1e-10)  # 안전한 로그 계산을 위해 클리핑한 후 손실을 누적합니다.
            loss += -(y[i]*np.log(a)+(1-y[i])*np.log(1-a)) # 에포크마다 평균 손실을 저장합니다.
        self.losses.append(loss/len(y))
```

score() 메서드 추가하고 단일층 신경망 훈련하기

활성화 함수를 뺐습니다

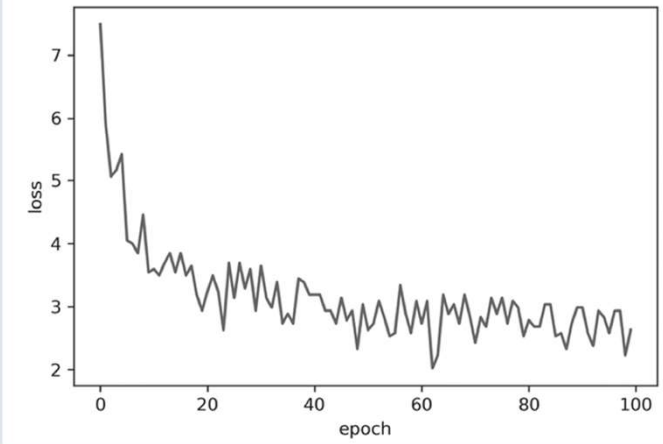
```
def predict(self, x):
    z = [self.forpass(x_i) for x_i in x]  # 정방향 계산
    return np.array(z) > 0                # 계단 함수 적용

def score(self, x, y):
    return np.mean(self.predict(x) == y)
```

```
layer = SingleLayer( )
layer.fit(x_train, y_train)
layer.score(x_test, y_test)
```

```
0.9298245614035088
```

```
plt.plot(layer.losses)
plt.xlabel('epoch')
plt.ylabel('loss')
plt.show( )
```



04-7 사이킷런으로 로지스틱 회귀를 수행합니다

로지스틱 손실 함수 지정

```
sgd = SGDClassifier(loss='log', max_iter=100, tol=1e-3, random_state=42)
```

회귀는 SGDRegressor

```
sgd.fit(x_train, y_train)
sgd.score(x_test, y_test)
0.8333333333333334
```

```
sgd.predict(x_test[0:10])
array([0, 1, 0, 0, 0, 0, 1, 0, 0, 0])
```