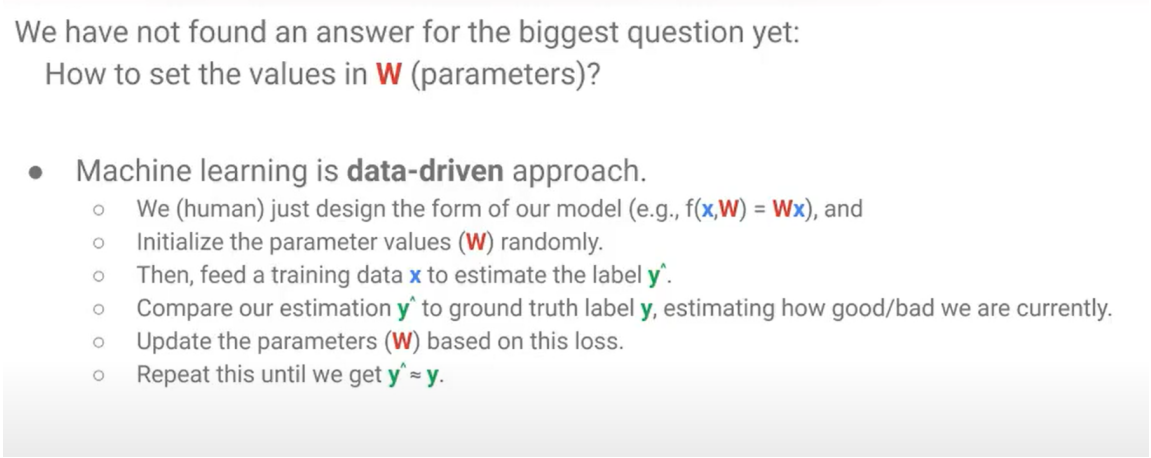
3️⃣

Lecture 3

- weight 찾는 방법



  - 가장 먼저 현재 예측한 가중치가 얼마나 좋은지 판단하는 것을 공부
  - 이후 이 loss를 바탕으로 파라미터 업데이트 하는 과정 공부

Loss function

- Loss function
  - 우리 모델의 예측이 얼마나 맞았는지 혹은 틀렸는지를 계량해주는 function
  - 인풋으로는 y hat과 y
  - 둘이 비슷할수록 fine-tuning → 0이거나 아주 작은 값이 되어야 함

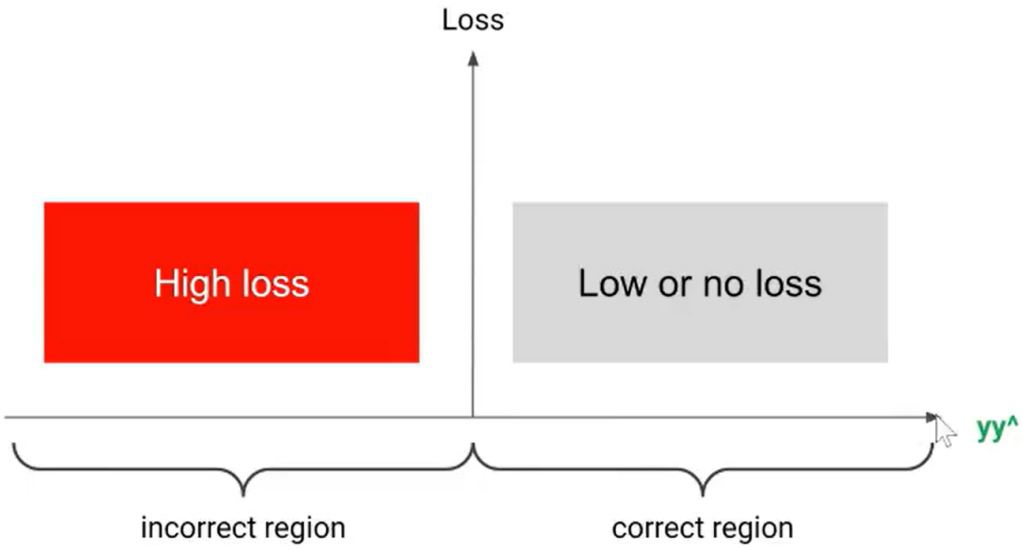
- A function of our prediction **y^** and ground truth label **y**: $\mathcal{L}(\hat{y}, y)$
- Depending on how **y^** and **y** are different, it outputs some positive number to penalize the model.
  - If **y^** = **y**, we do not want to penalize the model, so the loss should be (around) 0.
  - If **y^** ≈ **y**, we might want to penalize the model to fine-tune.
  - If the gap between **y^** and **y** is huge, we should heavily penalize.
  - So, how much **y^** and **y** are different?

- discriminative setting일 때

For **binary classification** problem, the ground truth **y** = {+1, -1}.

- Our model predicts one score **y^**.
- If **y^** > 0, we classify it as the positive class, otherwise as the negative class.
- **Margin-based loss**:
  - The loss is decided based on **yy^**.
  - If they have same sign (that is, classification is correct), loss gets smaller or 0.
  - If they have different sign (wrong classification), loss gets larger.

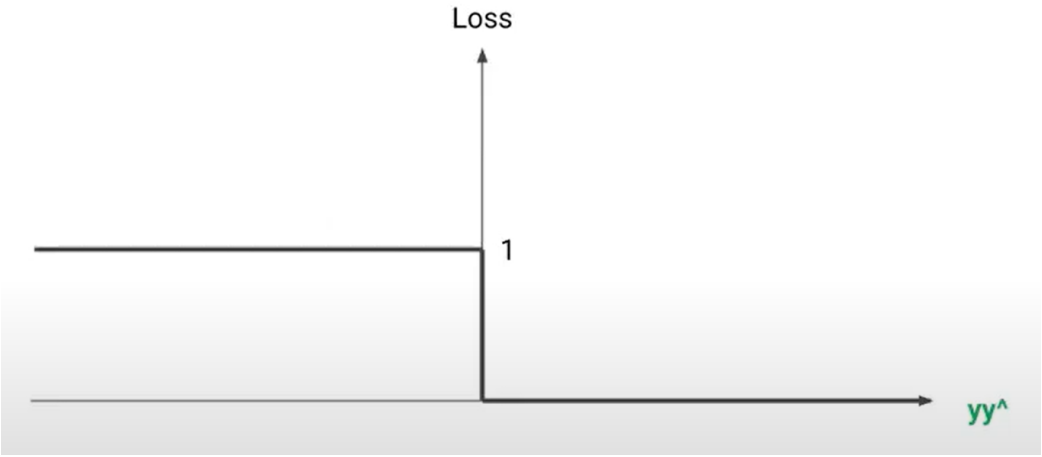
  - 다음과 같은 margin-based loss 모델을 갖고 싶음



→ 오른쪽으로 갈수록 우리는 잘한 것

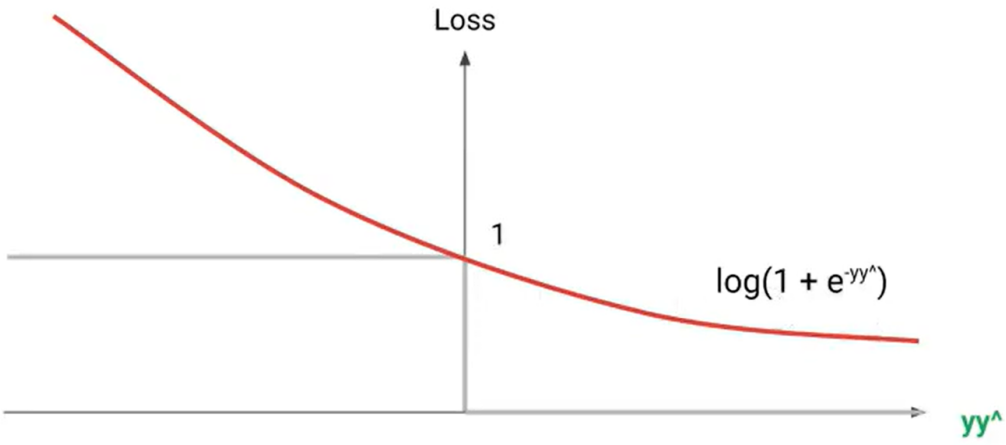
→ loss를 낮게 주거나 안주고 싶음

→ negative로 갈수록 실제 부호랑 다른단 의미니까 loss를 많이 주고 싶음

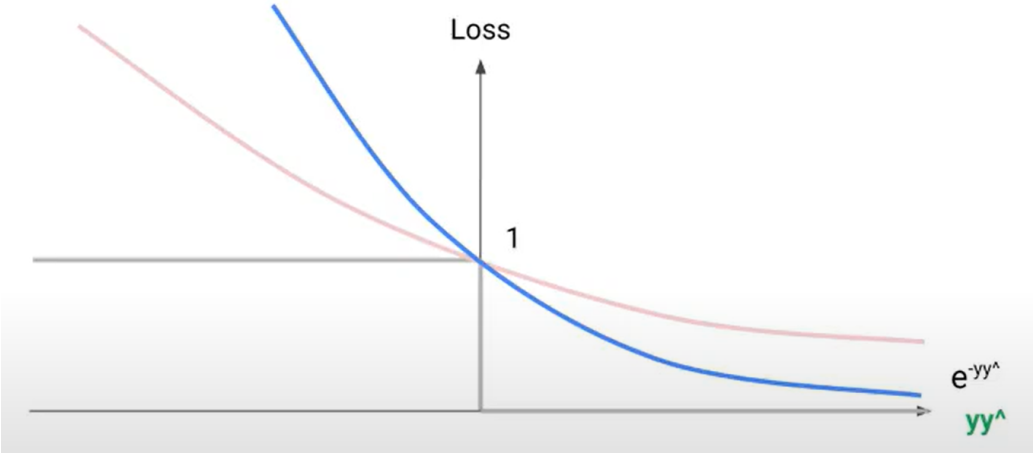


but 문제점이 있음 ; 미분이 안된다!! + 레이블 자체가 틀린 경우를 잡아내지 못한다!!

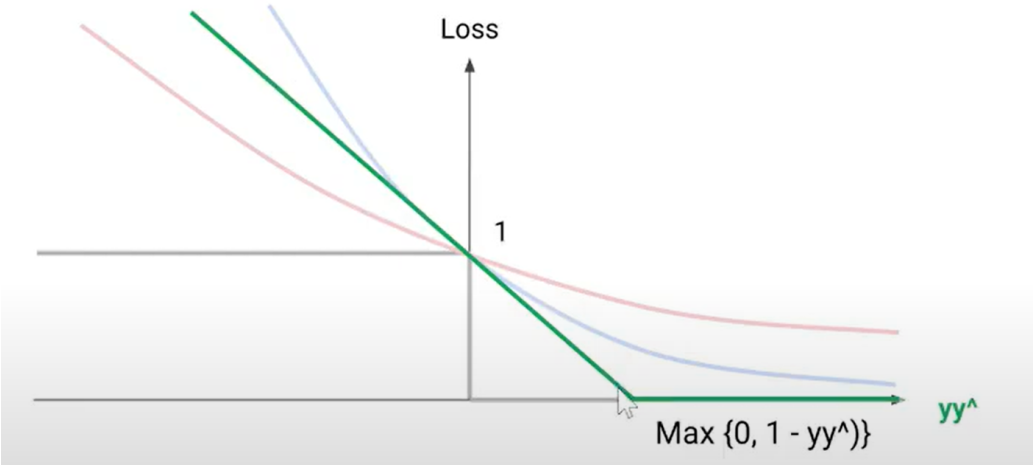
- 이 문제점을 model 등장
  - log loss



  - exponential loss



- Hinge loss



한 점을 제외한 모든 점에서는 constant라 딱히 미분하지 않아도 된다

→ 계산량이 빨라서 많이 쓰임

- 정리

  - Exponential loss is severely affected by outliers, as it assigns extremely large loss to them.
    → Less suitable for noisy data
  - Hinge loss and log loss are widely used.
  - Hinge loss (SVM) is computationally more efficient.
  - Log loss (logistic regression) is more interpretable, as its output can be seen as p(y|x).

- probabilistic setting

- For a **binary classification** problem, the ground truth **y** = {0, 1}.
  - Our model predicts one score **y^**, probability of one class.
  - The other class probability is 1 **y^**.
  - Taking sigmoid function on the score difference is an example.
- Naturally extends to K > 2 classes:
  - Ground truth is represented as an one-hot vector: **y** = [0, 0, 0, 0, 1, 0, 0]
  - The model predicts K - 1 scores, leaving the last one as 1 - sum.
  - **Softmax loss** is an example: predictions are between 0 and 1, and sum to 1.
- In this setting, the loss function basically **compares** two (GT and predicted) **probability distributions**.

- cross entropy
  - 모든 데이터 포인트를 모든 데이터 개수에 대해 -를 붙이고 y와 예측값의 log값을 곱함

From information theory: Measures the average number of bits needed to identify an event drawn from the set.

- General definition:

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K} y_{ik}\log(\hat{y}_{ik})$$

- Binary classification:

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N}[y_i\log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)]$$

  - 이걸 왜 loss function으로 사용하는가
    - 우리가 예측한 값에 대한 점수에 로그를 씌워 마이너스를 붙인 것
    - 우리가 쓰는 영역은 아래 노란색 상자
      - 정확하게 예측하면 0이 나옴 / 틀릴수록 loss 값이 커짐 (이상적인 function)

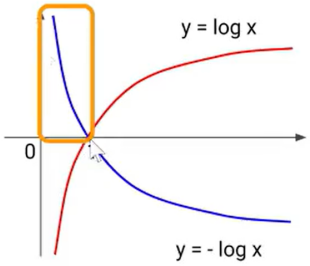
- Observation: Even if it sums over all classes (K), only one term survives, as we have y_{ik} = 1 for only one k.

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K} y_{ik}\log(\hat{y}_{ik}) \longrightarrow \mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N}\log(\hat{y}_{iT_i})$$

T_i: ground truth index for example i

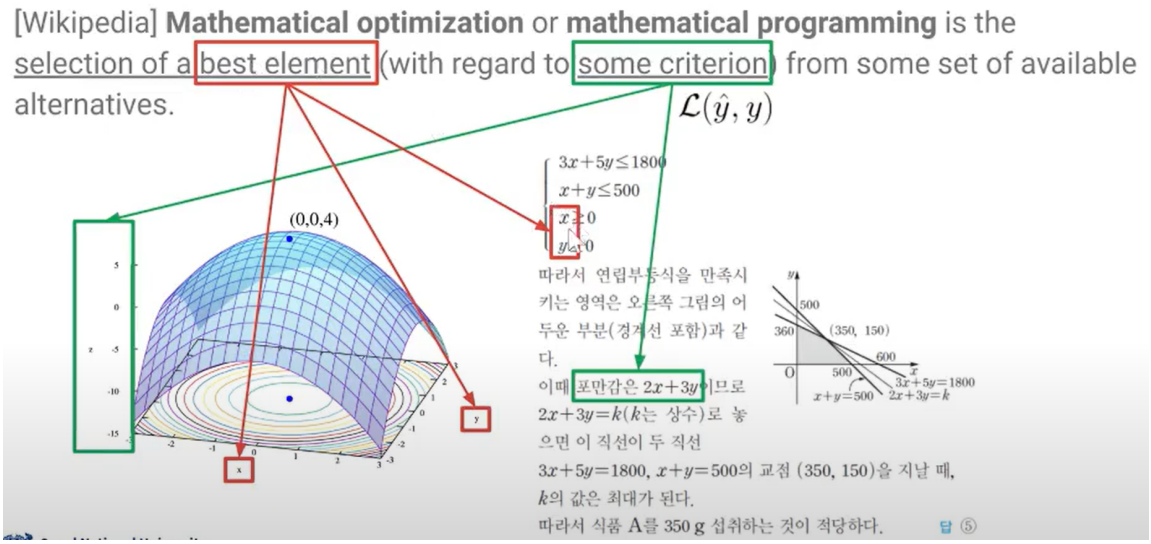
→ This is the sum of **-log("our predicted probability for the true class")** over all samples.

- Since "our predicted probability" is between 0 and 1, it corresponds to the blue range shown left.
  → If our estimate gets closer to 1, the loss converges to 0. If our estimates gets closer to 0, the loss increases.



Optimization

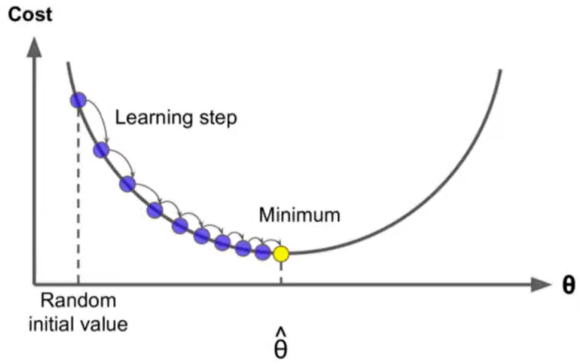
- Optimization



  - 방법
    - exhausetive search 모두 다 해보는 것
    - random search 아무 가중치를 넣어봄
    - visualization 위와 같이 그래프를 그려봄
    - greedy search

- 우리가 쓰는 방법

- We have a **cost function J(Θ)** we want to minimize.
- Idea: for the current value of Θ, calculate **gradient** of J(Θ), then take small step in direction of negative gradient. Repeat.
  - Recall that gradient (or derivative) of a function at a point is the **best linear approximation** to the function at that point.



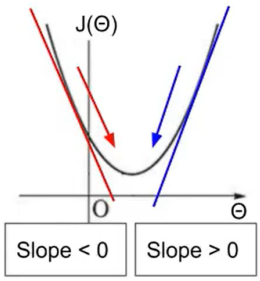
- gradient descent

- Update equation (in matrix notation):

$$\Theta^{new} = \Theta^{old} - \alpha \nabla_{\Theta} J(\Theta)$$

- Update equation (for a single parameter):

$$\theta_i^{new} = \theta_i^{old} - \alpha \frac{\partial}{\partial \theta_i^{old}} J(\Theta)$$




```
theta = rand(vector)
while true:
    theta_grad = evaluate_gradient(J, data, theta)
    theta = theta - alpha * theta_grad
    if (norm(theta_grad) <= beta) break
```

여기서 J(theta)가 loss function

→ 문제점이 있음 ; convex하지 않을 수 있음. loss function이 미분가능하지 않을 가능성

- stochastic gradient descent

  아주 작은 데이터셋에서 진행해도 어느 정도 맞는 방향으로 간다

  → 미니배치