cs231n lecture 6

1. Neural Network Training

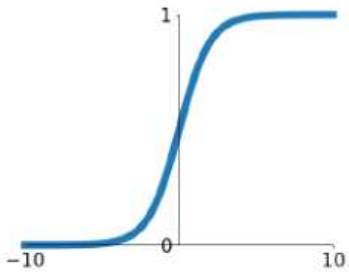
- 모든 data를 가지고 gradient descent Algorithm에 적용하면
  계산량이 많아 SGD(Stochastic Gradient Descent) Algorithm을 이용
  Sample을 뽑아내 Gradient Desscent Algorithm을 사용하는 방법

2. Activation Function

- Sigmoid function

Sigmoid

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

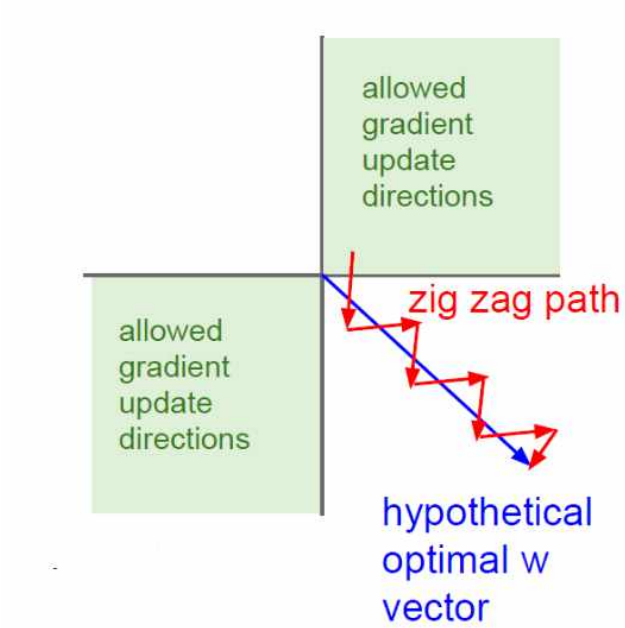


* 출력이 (0,1) 사이의 값이 나오도록 하는 선형 함수

* 단점:
  - Saturated neurons가 Gradient값을 0으로 만든다.
  - 원점 중심이 아니다.
  - 지수함수가 계산량이 많다.

* Gradient의 값이 0이 되는 것이 왜 단점?
Chain Rule에서 Global gradient값, 즉 결과값이 0이 되면
local gradient 값도 0이 돼서
Input에 있는 gradient 값을 구할 수 없음

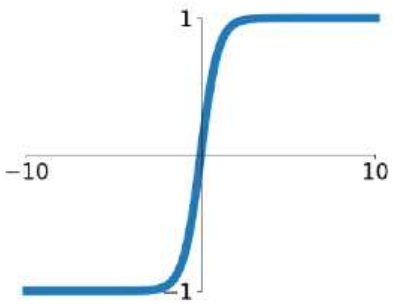
* 원점 중심이 아닌 것은 왜 단점?



w의 경우 제 1사분면과 제 3사분면으로 update가 되기 때문에 원하는 방향으로 update 하기 힘들다.

- tanh(x)

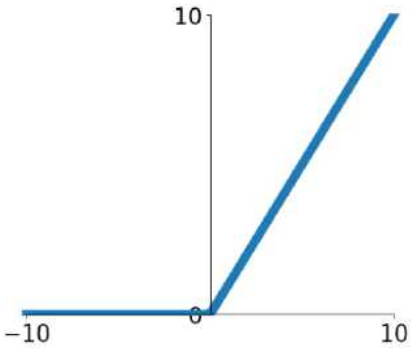
* Sigmoid가 원점 중심이 아닌 것을 보완하기 위해 나온 function



tanh(x)

* 여전히 saturated한 뉴런일 때, gradient값이 0

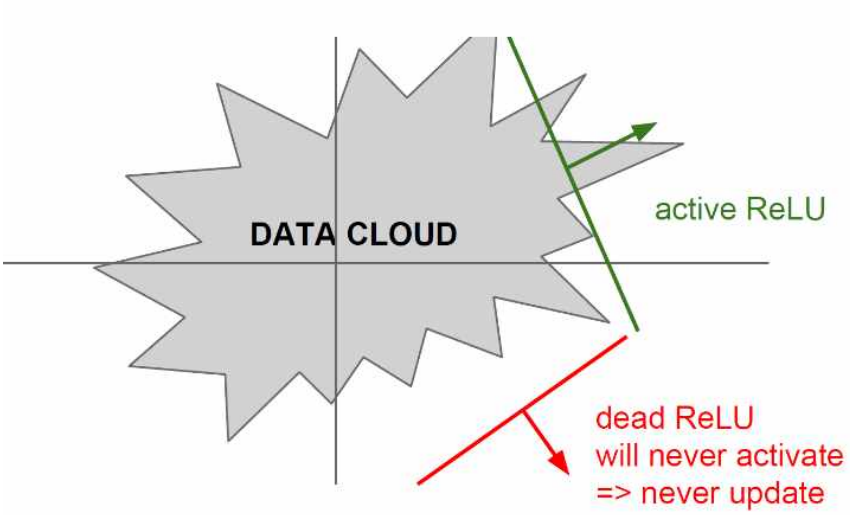
- ReLU



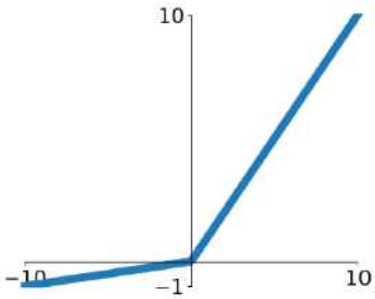
ReLU
(Rectified Linear Unit)

* (+) 영역에서 saturate하지 않고,
  계산 속도도 element-wise 연산이어서
  sigmoid/tanh보다 훨씬 빠름

* 단점 :
(-)의 값은 0으로 만들어 버려 Data의 절반만 activate하게 만듦



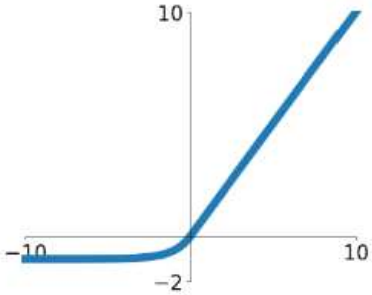
- Leaky ReLU



Leaky ReLU

$$f(x) = \max(0.01x, x)$$

- Exponential Linear Unit



$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha\left(\exp(x) - 1\right) & \text{if } x \leq 0 \end{cases}$$

- Maxout

$$max(w_1^T x + b_1, w_2^T x + b_2)$$

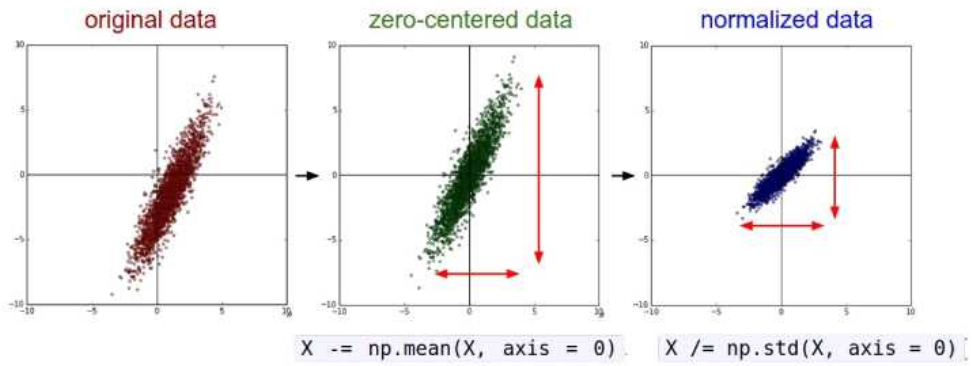
* parameter가 기존 function보다 2배 있어야 함

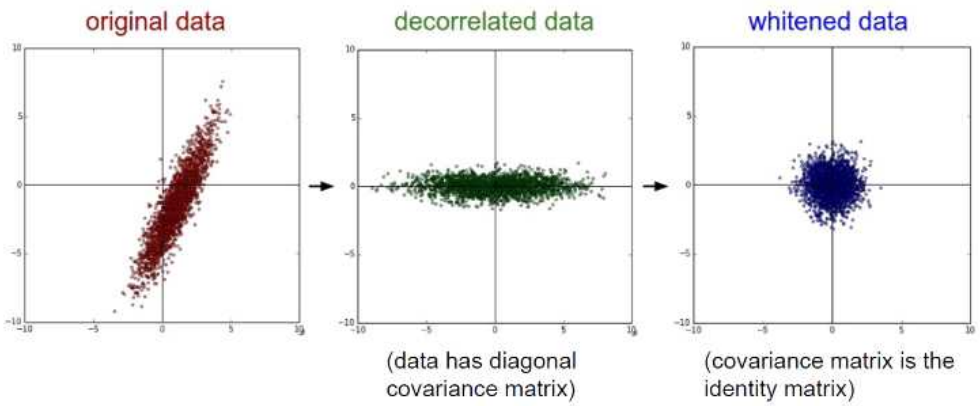
3. Data processing

- 종류 : Zero-centered, Normalized, PCA, Whitening

- Zero-centered, Normalized :
  모든 차원이 동일한 범위에서 전부 동등한 기여를 할 수 있도록 하는 것
  이미지는 Zero-centered 과정만 거침

- PCA, Whitening:
  더 낮은 차원으로 projection하는 느낌
  이미지 처리에서는 이 전처리 과정은 거치지 않음

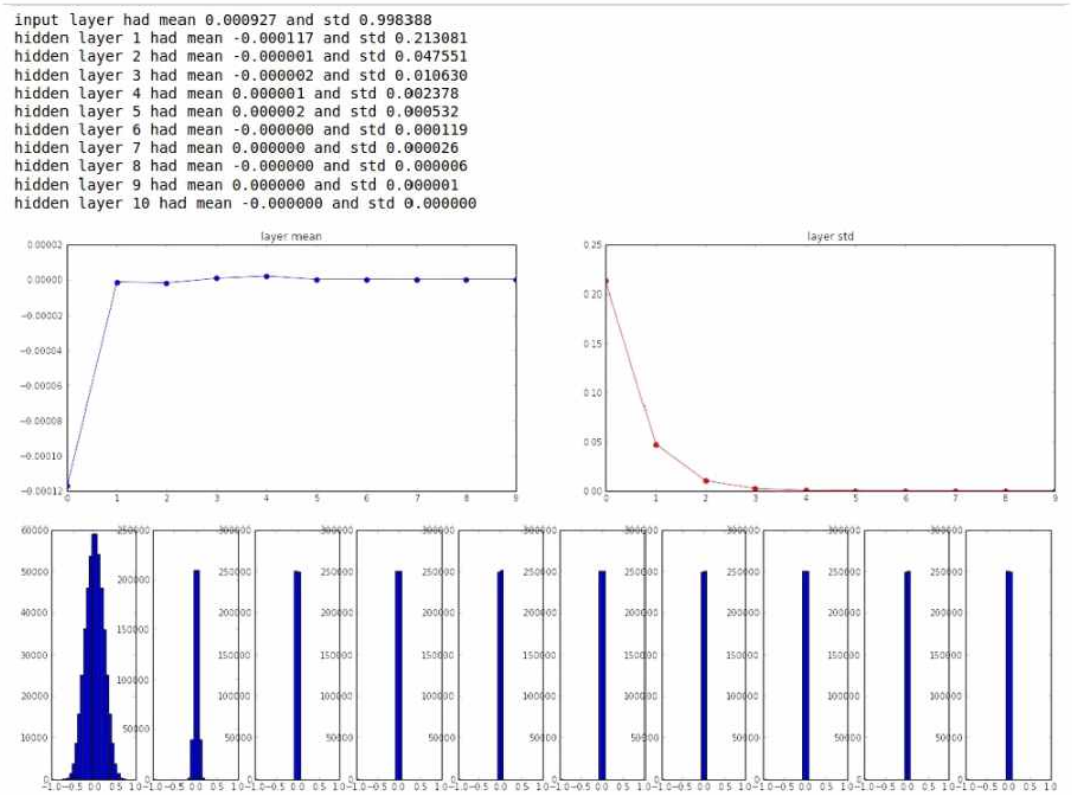




4. Weight Initialization

- 초기값을 0으로 하면 모든 뉴런은 동일한 일을 하게 돼
  모든 gradient의 값이 같게 될 것 -> 의미 x

- 작은 random한 수로 초기화:
  초기 Weight는 표준정규분포에서 sampling
  얕은 network에서는 잘 작동하지만 network가 깊어질 경우 문제가 생김
  network가 깊으면 깊을수록 weight의 값이 너무 작아 0으로 수렴하기 때문

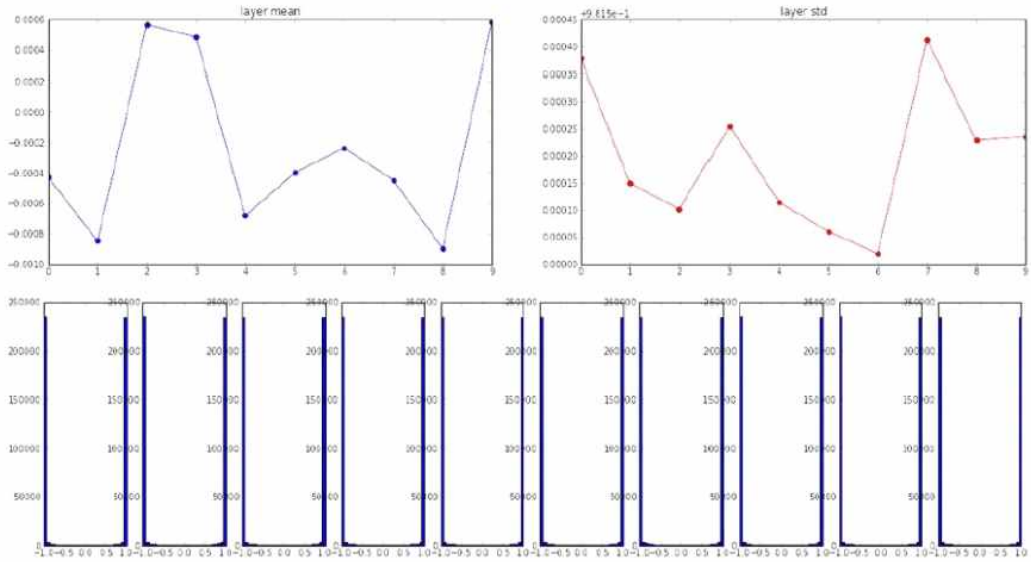


- 표준 편차를 키운다면?
  activaton value의 값이 극단적인 값을 가지게 되고,
  gradient의 값이 모두 0으로 수렴

```
W = np.random.randn(fan_in, fan_out) * 1.0 # layer initialization
```

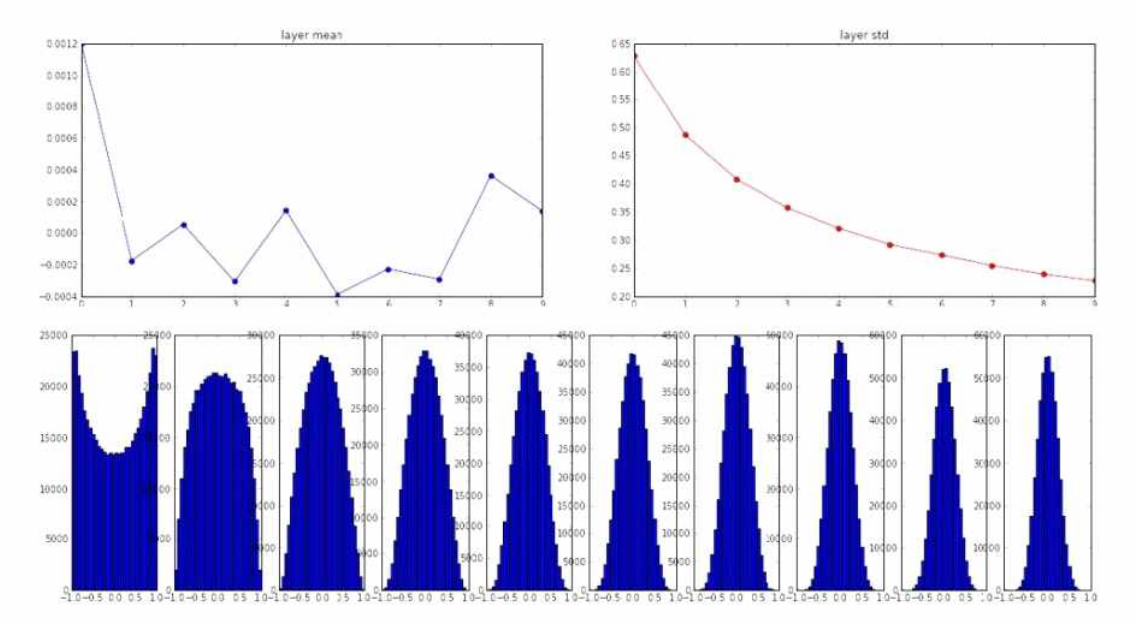
```
input layer had mean 0.001800 and std 1.001311
hidden layer 1 had mean -0.000430 and std 0.981879
hidden layer 2 had mean -0.000849 and std 0.981649
hidden layer 3 had mean 0.000566 and std 0.981601
hidden layer 4 had mean 0.000483 and std 0.981755
hidden layer 5 had mean -0.000682 and std 0.981614
hidden layer 6 had mean -0.000401 and std 0.981560
hidden layer 7 had mean -0.000237 and std 0.981520
hidden layer 8 had mean -0.000448 and std 0.981913
hidden layer 9 had mean -0.000899 and std 0.981728
hidden layer 10 had mean 0.000584 and std 0.981736
```

*1.0 instead of *0.01



- 이런 초기값 문제에 대해서 'Xavier initialization'이라는 논문이 제시

- activation function이 linear하다는 가정하에 다음과 같은 식을 사용하여 weight의 값을 초기화하고, 입/출력의 분산을 맞춰줄 수 있게 됨



- activation function이 ReLU인 경우, 출력의 분산이 반토막 나기 때문에 이 식이 성립하지 않음

- activation function이 ReLU인 경우에는 He Initialization을 사용

5. Weight Initialization

- Batch에서 계산한 mean과 variance를 이용하여 정규화를 해주는 과정을 Model에 추가해주는 것

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

- 각 layer에서 Weight가 지속적으로 곱해져서 생기는 Bad Scaling의 효과를 상쇄

- unit gaussian으로 바꿔주고 분산과 평균을 이용해 Normalized를 좀 더 유연하게 할 수 있음

$$y^{(k)} = \gamma^{(k)}\widehat{x}^{(k)} + \beta^{(k)}$$

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

- Regularization의 역할 (Overfitting 방지)

- weight의 초기화 의존성에 대한 문제 줄임

- Test할 때 minibatch의 평균과 표준편차를 구할 수 없어 Training하면서 구한 평균의 이동평균을 이용해 고정된 Mean과 Std를 사용

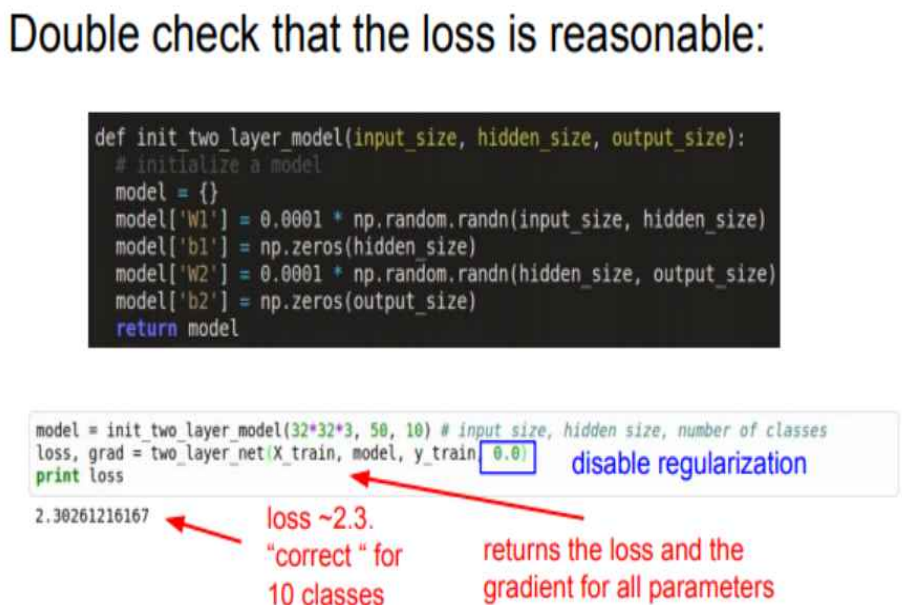
- 학습 속도 개선 가능

6. Babysitting the Learning Process

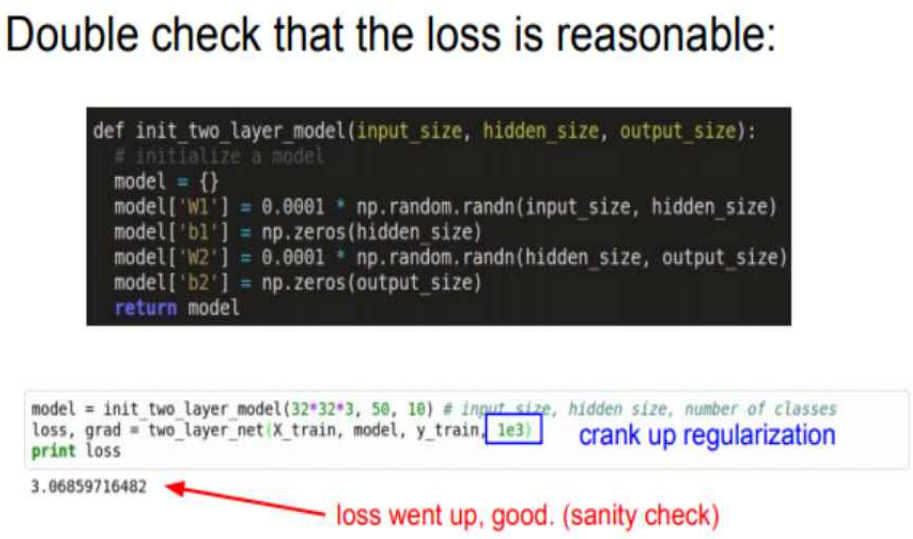
- 전처리 과정 : 이미지의 경우 zero-centered

- 사용할 neural architecture를 정해줌

- loss 가 적절하게 나오는지 확인

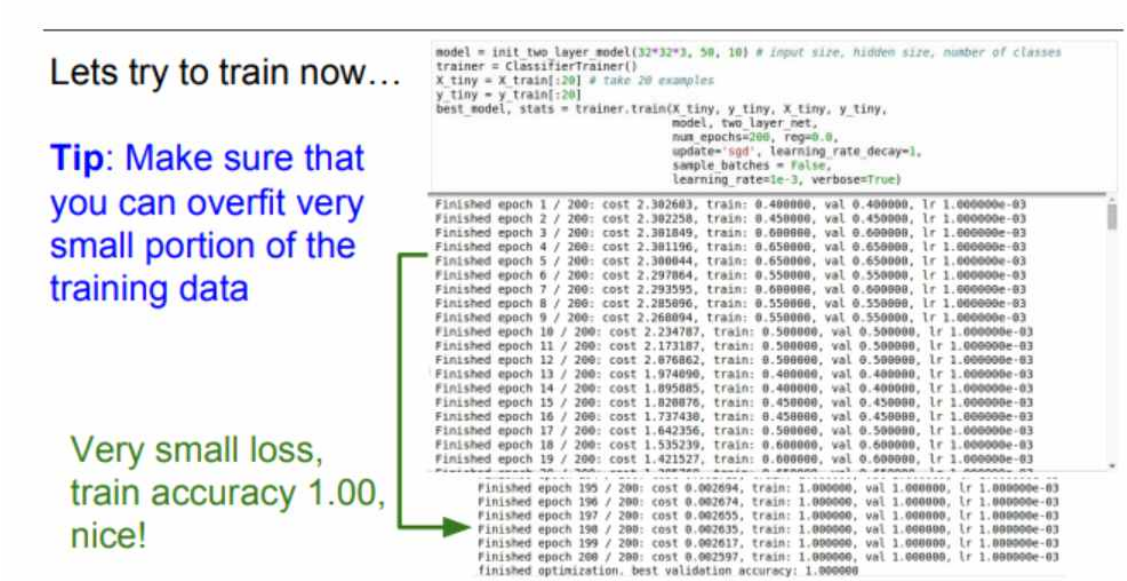


- 규제 값을 올렸을 때 loss가 증가, network가 잘 동작하는지 확인

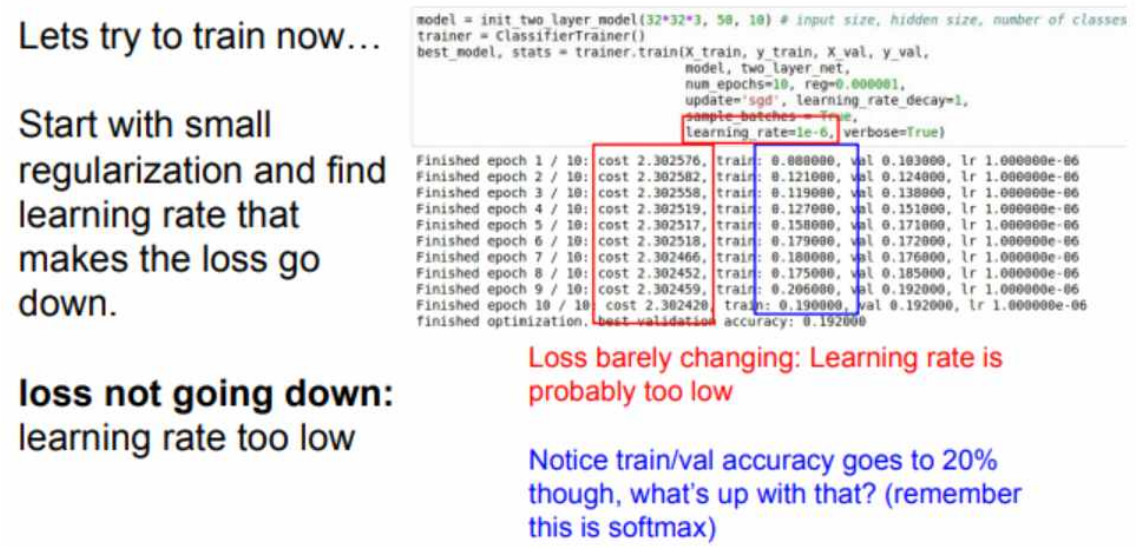


- 작은 데이터 셋을 이용해서 훈련

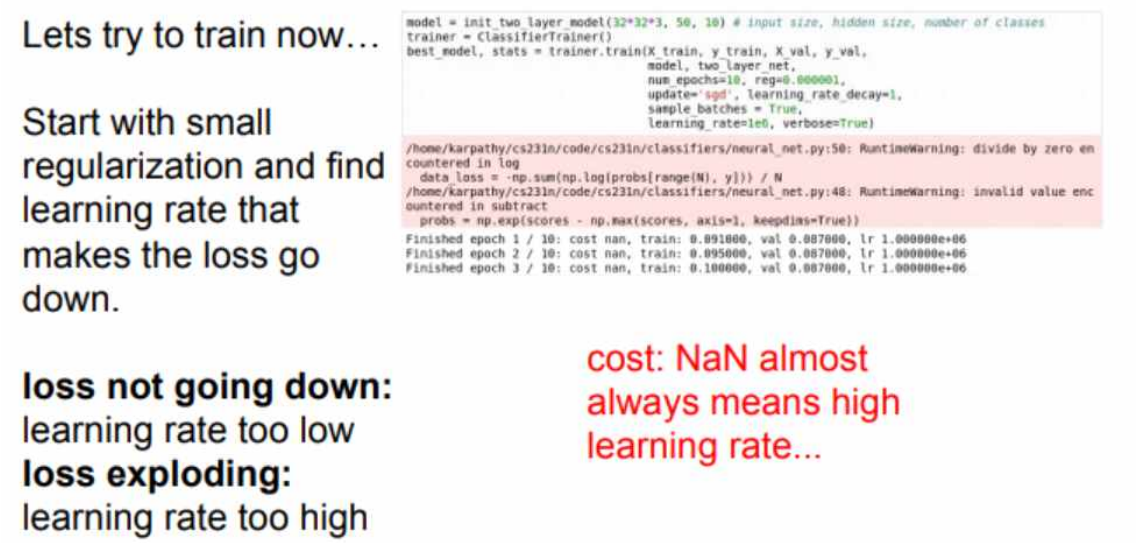
- Data 수가 작아 overfitting이 발생, train acc가 100%
  overfitting이 나온다는 것은 모델이 제대로 동작하고 있다는 의미



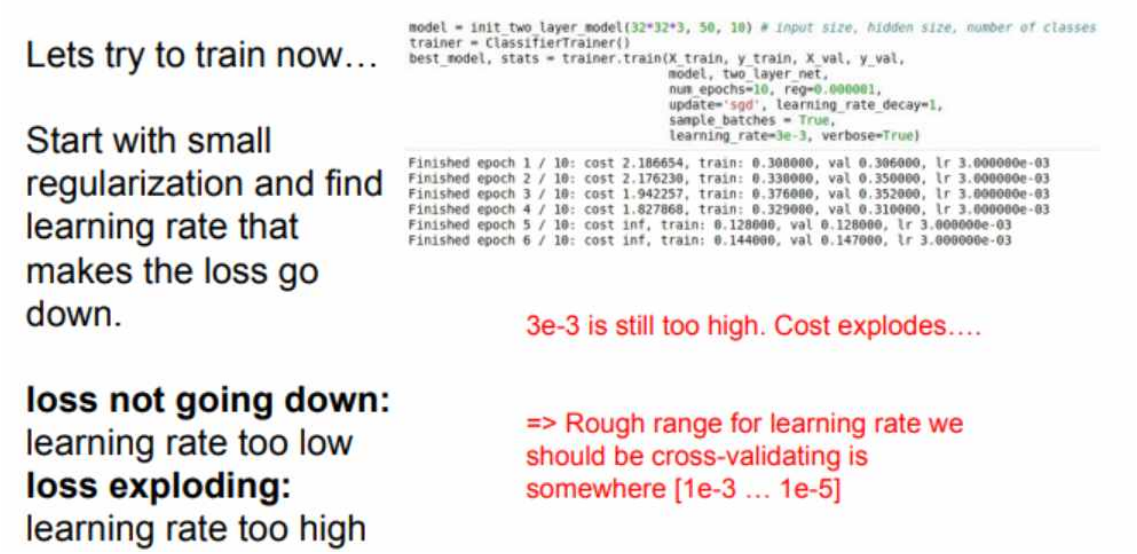
- hyper parameter 값들(regularization, learning rate)을 설정



- lr = 1e6로 설정하니 값이 너무 커서 cost 값이 Nan



- lr = 3e-3로 수정해도 inf, 몇 번의 실험을 통해 1e-3~1e-5 값이 적절한 값이라는 것을 추측

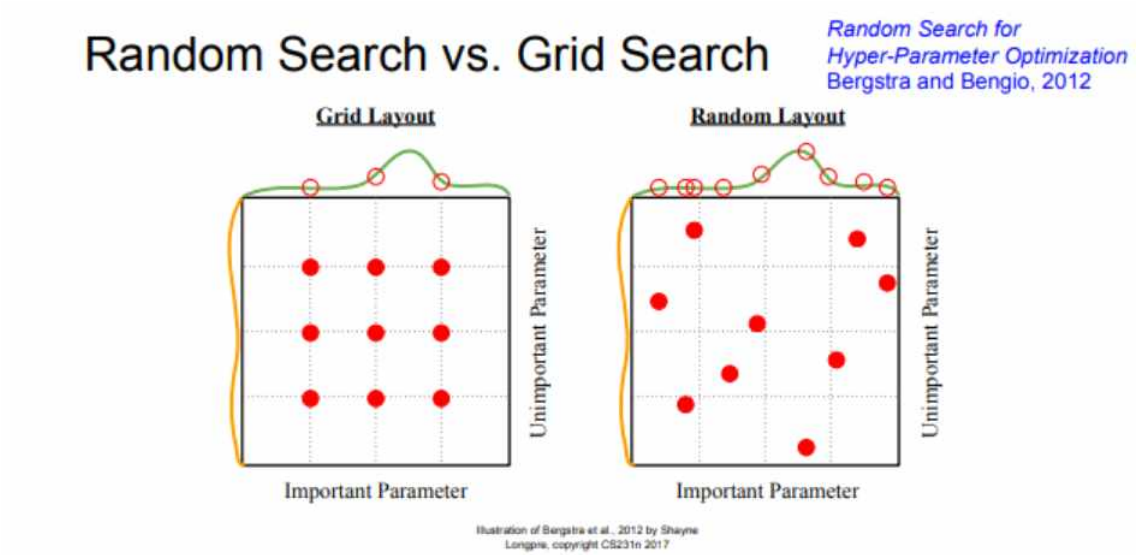


7. Hyperparameter Optimization

- 종류 : Grid Search, Random Search

- Grid Search :
  탐색의 대상이 되는 특정 구간 내의 후보 hyperparameter 값들을
  일정한 간격을 두고 선정
  각각에 대하여 측정한 성능 결과를 기록한 후
  가장 높은 성능을 발휘했던 hyperparameter 값을 선정하는 방법

- Random Search :
  Grid Search와 큰 맥락은 유사
  탐색 대상 구간 내의 후보 hyperparameter 값들을
  random sampling으로 선정하는 것이 차이
  Grid Search에 비해 불필요한 반복 수행 횟수를 대폭 줄임
  정해진 간격(grid) 사이에 위치한 값들에 대해서도 확률적으로 탐색이 가능
  최적 hyperparameter 값을 더 빨리 찾을 수 있음
  Random Search를 더 많이 사용



- 과정 요약 :
  * Hyperparameter 값을 설정
  * 위에서 정한 범위 내에서 파라미터 값을 무작위로 추출
  * Validation Set을 이용하여 평가
  * 특정 횟수를 반복하여 그 정확도를 보고 Hyperparameter 범위를 좁힘

- Hyperparameter를 정할 때 loss curve를 보고 이 hyperparameter가 적합한지 아닌지 평가를 하는 경우가 많음

- loss curve가 초기에 평평하다면 초기화가 잘못될 가능성이 큰 것

- training accuracy와 validation accuracy가 gap이 클 경우 overfitting이 된 가능성이 매우 높은 것

- gap이 없을 경우 model capacity를 늘리는 것을 고려
  training한 dataset이 너무 작은 경우일 수도 있음