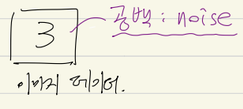
1. Regression : Approximation of target function based on given data.

2. Linear Regression : Regression of "Linear form"
   esl) Linear combination (Linear mapping)

   * Statistics / Deep Learning 의 L.R 차이
     └ noise가 더 ↑ 한 환경.



3. Loss, Cost function

$SE = Loss_{(i)} = (y_i - \hat{y}_i)^2$ * $|y_i - \hat{y}_i|$는 ∨로 미분불가능 해서 적절하지 않음.

$$MSE = Cost = \frac{1}{N}\sum_{i=1}^{n} Loss_{(i)} = \frac{1}{N}\sum^{n}(y_i - \hat{y}_i)^2$$

$$\theta := \theta - \alpha \nabla L(\theta) \quad * \nabla L = \frac{\partial L}{\partial \theta}$$
$$\theta := \theta - \alpha \nabla J(\theta) \quad \nabla J = \frac{\partial J}{\partial \theta}$$

4. gradient descent method

model $\hat{y} = \theta_n x_n + \theta_{n-1}x_{n-1} + \dots + \theta_1 x_1 + \theta_0$

params $(\vec{w}, b) = (\theta_n, \theta_{n-1}, \dots, \theta_1, \theta_0)$

loss $\mathscr{L}^{(i)}(\vec{\theta}) = (y^{(i)} - \hat{y}^{(i)})^2 = (y^{(i)} - (\theta_n x_n^{(i)} + \theta_{n-1}x_{n-1}^{(i)} + \dots + \theta_1 x_1^{(i)} + \theta_0))^2$
n+1 개의 변수 ⇒ gradient가 n+1 개 필요 [———]

cost $J(\vec{\theta}) = \frac{1}{n}\sum_{i=1}^{n}\mathscr{L}^{(i)}$
$= \frac{1}{n}\sum_{i=1}^{n}(y^{(i)} - (\theta_n x_n^{(i)} + \theta_{n-1}x_{n-1}^{(i)} + \dots + \theta_1 x_1^{(i)} + \theta_0))^2$

Fast campus 신경식 강사.

gradient $\nabla_{\vec{\theta}}\mathscr{L}^{(i)}(\vec{\theta}) = (\frac{\partial \mathscr{L}^{(i)}(\vec{\theta})}{\partial \theta_n}, \frac{\partial \mathscr{L}^{(i)}(\vec{\theta})}{\partial \theta_{n-1}}, \dots, \frac{\partial \mathscr{L}^{(i)}(\vec{\theta})}{\partial \theta_1}, \frac{\partial \mathscr{L}^{(i)}(\vec{\theta})}{\partial \theta_0})$

gradient descent $\vec{\theta} := \vec{\theta} - \alpha \nabla_{\vec{\theta}}\mathscr{L}^{(i)}(\vec{\theta})$

$\theta_n := \theta_n - \alpha\frac{\partial \mathscr{L}^{(i)}(\vec{\theta})}{\partial \theta_n}$ 각각 자신의 parameter를 update

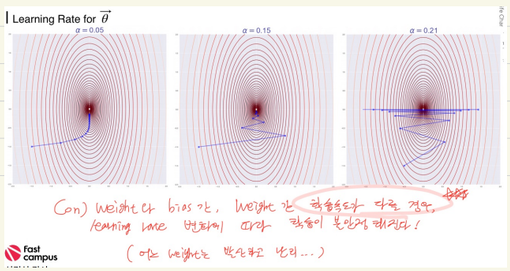
$\theta_{n-1} := \theta_{n-1} - \alpha\frac{\partial \mathscr{L}^{(i)}(\vec{\theta})}{\partial \theta_{n-1}}$ $\theta_0 := \theta_0 - \alpha\frac{\partial \mathscr{L}^{(i)}(\vec{\theta})}{\partial \theta_0}$

$\vdots$

$\theta_1 := \theta_1 - \alpha\frac{\partial \mathscr{L}^{(i)}(\vec{\theta})}{\partial \theta_1}$

5. learning rate

$\theta = \theta - \alpha\frac{\partial L}{\partial \theta}$ ⇒ α↑면, 발산 위험↑ 과 진동을 인한 연산↑ 가능.



Con) weight 와 bias 간, weight 간 학습속도가 다른 경우, learning rate 변화에 따라 학습이 불안정 해진다!
(아는 weight는 빠르게 반영...)

- θ update의 발산 or 진동은 learning rate 에도 input의 값도 관련!

$$\Delta\theta = \alpha \cdot \frac{\partial L}{\partial \theta}$$

$$\frac{\partial L}{\partial \theta} = -2x_{(i)}(y_{(i)} - \theta x_{(i)})$$

만약, x_i 가 $k \cdot x_i$ 가 된다면,

$$\frac{\partial L}{\partial \theta} = -2k^2 x_{(i)}(y_{(i)} - \theta x_{(i)})$$

∴ input이 k배 증가 시,
learning rate(α)의 영향은 $\alpha \cdot 1 \rightarrow \alpha \cdot k^2$ 으로!

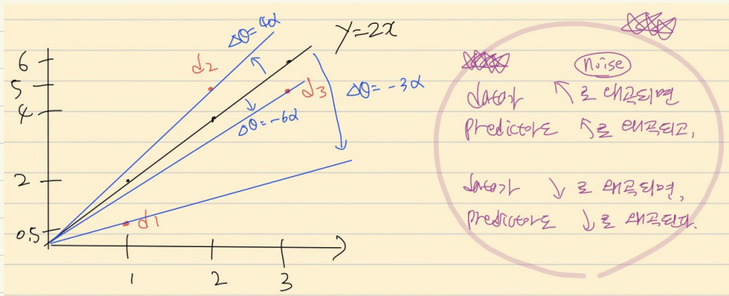
실제로 input이 증가 시, Loss function의 ∪ 그래프가 더 가팔라짐 (Convex ↑)
∵ $Loss_{(i)} = \theta^2 x_{(i)}^2 - 2\theta_{(i)}y_{(i)} + y_{(i)}^2$

- 즉, LR↓ 레도, input이 크면 Δθ 크기가 커져 불안정한 학습 가능성이 있다는 것.
⇒ Cost function 활용! (안정적, 빠르게)

데이터를 Mini batch size(n)로 묶어, 각 데이터 Δθ 크기는 $\frac{1}{n}$를 pooling 하는 효과! * $J(\theta) = \frac{1}{n}\sum L(\theta)$

6. Noise

Noise가 학습에 미치는 영향.



7. Learning – process

Total Learning Process

```
# dataset preparation
dataset_gen = dataset_generator()

dataset_gen.set_coefficient([5,0])
x_data, y_data = dataset_gen.make_dataset()

# model implementation
node1 = nodes.mul_node()

# sqaure error loss implementation
node2 = nodes.minus_node()
node3 = nodes.square_node()

# hyperparameter setting
epochs = 5  # total epoch setting
lr = 0.01   # learning rate setting

th = -1     # arbitrary θ
loss_list = []
th_list = []
```

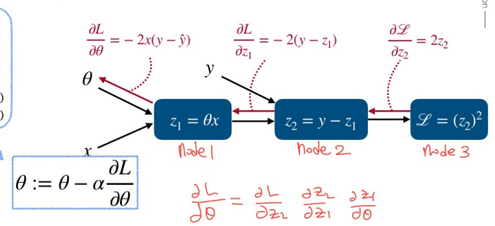
```
for epoch in range(epochs):
    for data_idx in range(len(x_data)):
        x, y = x_data[data_idx], y_data[data_idx]

        z1 = node1.forward(th, x)
        z2 = node2.forward(y, z1)
        l = node3.forward(z2)

        dz2 = node3.backward(1)
        dy, dz1 = node2.backward(dz2)
        dth, dx = node1.backward(dz1)

        th = th - lr*dth

        th_list.append(th)
        loss_list.append(l)
```



8. Hyper parameter tuning

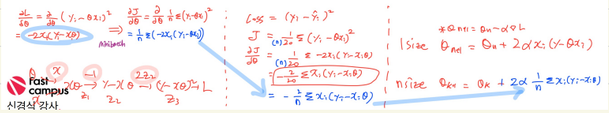
- learning rate
- initial theta
- iteration (mini batch size)
- epoch
- Coefficient.

9. Mini batch ⇒ Cost function 활용 연산!

Iterations/Epochs and Mini-batch

Total # sample:	1000
Mini-batch size:	20
Iteration per epoch:	$\frac{\#\ total\ sample}{mini-batch\ size} = \frac{1000}{20} = 50$
# iterations for 5 epochs:	5 * 50 = 250 Iterations

epoch 당 update 횟수 / 전체데이터 처리횟수 / epoch 당 50 iteration (batch size = 20)



$\nabla L_{(i)} = -2x_{(i)}(y_{(i)} - \theta x_{(i)})$

$\nabla J = \frac{1}{n}\sum -2x_{(i)}(y_{(i)} - \theta x_{(i)})$

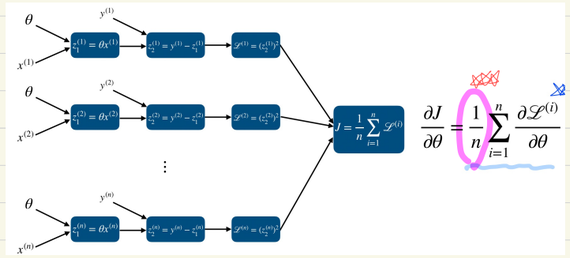
10. 정리.

Review

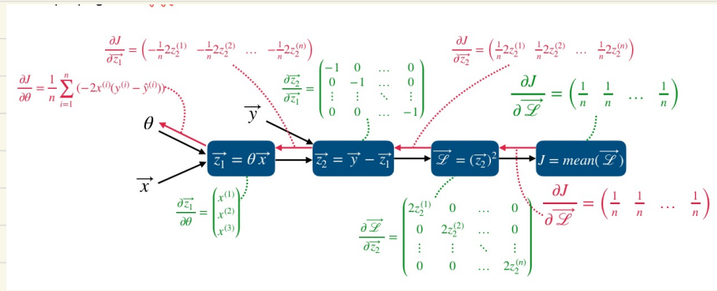
model	$\hat{y} = \theta x$
loss	$\mathcal{L}^{(i)} = (y^{(i)} - \hat{y}^{(i)})^2 = (y^{(i)} - \theta x^{(i)})^2$
cost	$J(\theta) = \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}^{(i)} = \frac{1}{n}\sum_{i=1}^{n} (y^{(i)} - \theta x^{(i)})^2$
gradient	$\frac{\partial J(\theta)}{\partial \theta} = -\frac{1}{n}\sum_{i=1}^{n}\left[2x^{(i)}(y^{(i)} - \theta x^{(i)})\right]$
gradient descent	$\theta := \theta - \alpha \frac{\partial J(\theta)}{\partial \theta}$
parameter update	$\theta := \theta + \frac{\alpha}{n}\sum_{i=1}^{n}\left[2x^{(i)}(y^{(i)} - \theta x^{(i)})\right]$

11. mini-batch의 효과.

① 개별 Loss(cj)의 θ update 시 영향도 pooling ⇒ 보다 Robust한 학습가능.
② 연산 횟수↓ (epoch 당 iteration 수 ↓)

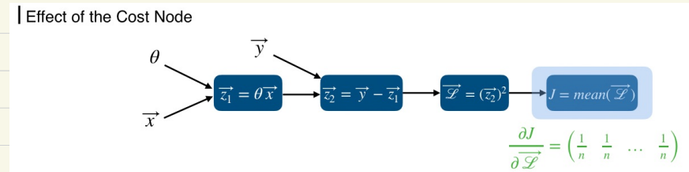


12. Vectorization for several samples $\vec{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$ $\vec{\theta} = \begin{bmatrix} \theta_1 \\ \vdots \\ \theta_t \end{bmatrix}$



13. Effect of Cost node

Effect of the Cost Node



$$\frac{\partial J}{\partial \theta} = \frac{\partial J}{\partial \vec{\mathcal{L}}}\frac{\partial \vec{\mathcal{L}}}{\partial \theta} = \left(\frac{1}{n} \quad \frac{1}{n} \quad \cdots \quad \frac{1}{n}\right)\begin{pmatrix} -2x^{(1)}(y^{(1)} - \hat{y}^{(1)}) \\ -2x^{(2)}(y^{(2)} - \hat{y}^{(2)}) \\ \vdots \\ -2x^{(n)}(y^{(n)} - \hat{y}^{(n)}) \end{pmatrix} = \left(\frac{1}{n} \quad \frac{1}{n} \quad \cdots \quad \frac{1}{n}\right)\begin{pmatrix} \frac{\partial \mathcal{L}^{(1)}(\theta)}{\partial \theta} \\ \frac{\partial \mathcal{L}^{(2)}(\theta)}{\partial \theta} \\ \vdots \\ \frac{\partial \mathcal{L}^{(n)}(\theta)}{\partial \theta} \end{pmatrix}$$

$$= \frac{1}{n}\sum_{i=1}^{n}\frac{\partial \mathcal{L}^{(i)}(\theta)}{\partial \theta}$$

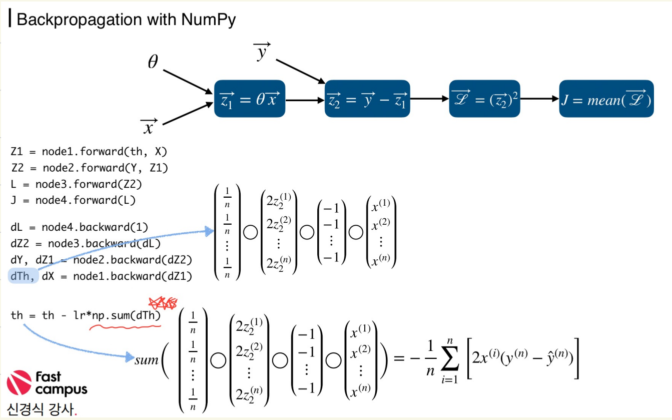
• Cost의 역할
[forward ⇒ 평균값 구해줘
backward ⇒ $\frac{1}{n}$ 보정위한 Vectorization

Back propagation (학습) 에선,
Cost Node는 $\frac{1}{n}$ 보정/위한 Update시, 개별 Loss 영향도에 Vectorization 의미!

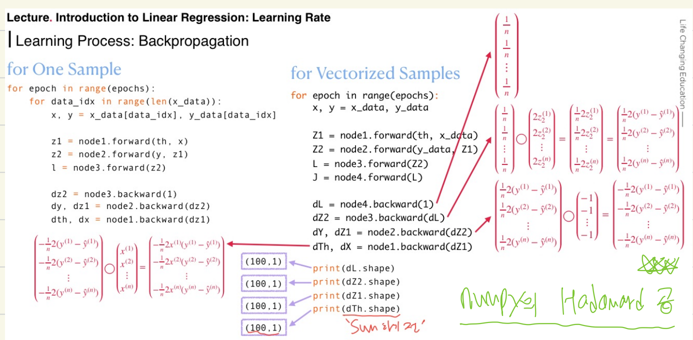
※ forward propagation 에선, 개별 Loss의 평균값 연산.

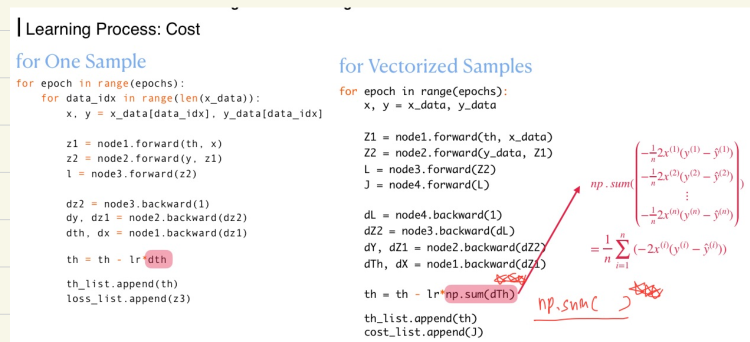
14. Backpropagation with numpy.

- numpy는 element wise 연산 제공. 특히 곱셈이라면 $a \odot b = \begin{bmatrix} a_1 b_1 \\ a_2 b_2 \\ \vdots \\ a_n b_n \end{bmatrix}$ Hadamard 곱 기본연산!



$\therefore$ np.sum(a*b) = $a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$





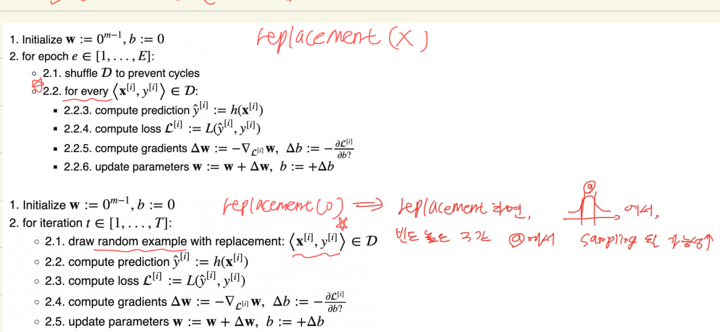
15. batch, stochastic, mini-batch gradient descent.

(stochastic: with replacement / without; mini-batch: with replacement / without)

- batch size (n, 1, mini n)에 따라 gradient descent 학습모습이 다르게 된다.
- batch-size ↓ 일수록 overfitting. 연산시간 ↑ 문제 가능.
  (stochastical)

- Stochastic GD

replacement (X)



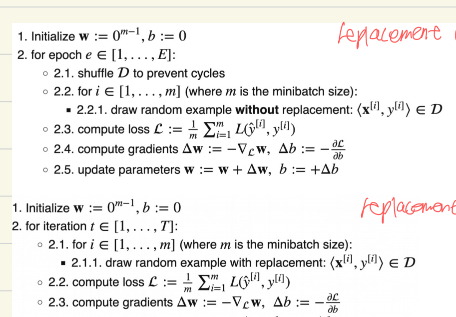
1. Initialize $\mathbf{w} := 0^{m-1}, b := 0$
2. for epoch $e \in [1, \ldots, E]$:
   - 2.1. shuffle $\mathcal{D}$ to prevent cycles
   - 2.2. for every $\langle \mathbf{x}^{[i]}, y^{[i]} \rangle \in \mathcal{D}$:
     - 2.2.3. compute prediction $\hat{y}^{[i]} := h(\mathbf{x}^{[i]})$
     - 2.2.4. compute loss $\mathcal{L}^{[i]} := L(\hat{y}^{[i]}, y^{[i]})$
     - 2.2.5. compute gradients $\Delta \mathbf{w} := -\nabla_{\mathcal{L}^{[i]}} \mathbf{w}$, $\Delta b := -\frac{\partial \mathcal{L}^{[i]}}{\partial b}$
     - 2.2.6. update parameters $\mathbf{w} := \mathbf{w} + \Delta \mathbf{w}$, $b := +\Delta b$

replacement (O) ⟹ replacement 하면, 뽑은 후 공을 다시 넣어서 sampling 된 경우

1. Initialize $\mathbf{w} := 0^{m-1}, b := 0$
2. for iteration $t \in [1, \ldots, T]$:
   - 2.1. draw random example with replacement: $\langle \mathbf{x}^{[i]}, y^{[i]} \rangle \in \mathcal{D}$
   - 2.2. compute prediction $\hat{y}^{[i]} := h(\mathbf{x}^{[i]})$
   - 2.3. compute loss $\mathcal{L}^{[i]} := L(\hat{y}^{[i]}, y^{[i]})$
   - 2.4. compute gradients $\Delta \mathbf{w} := -\nabla_{\mathcal{L}^{[i]}} \mathbf{w}$, $\Delta b := -\frac{\partial \mathcal{L}^{[i]}}{\partial b}$
   - 2.5. update parameters $\mathbf{w} := \mathbf{w} + \Delta \mathbf{w}$, $b := +\Delta b$

- mini-batch GD

replacement (X)



1. Initialize $\mathbf{w} := 0^{m-1}, b := 0$
2. for epoch $e \in [1, \ldots, E]$:
   - 2.1. shuffle $\mathcal{D}$ to prevent cycles
   - 2.2. for $i \in [1, \ldots, m]$ (where m is the minibatch size):
     - 2.2.1. draw random example **without** replacement: $\langle \mathbf{x}^{[i]}, y^{[i]} \rangle \in \mathcal{D}$
   - 2.3. compute loss $\mathcal{L} := \frac{1}{m} \sum_{i=1}^{m} L(\hat{y}^{[i]}, y^{[i]})$
   - 2.4. compute gradients $\Delta \mathbf{w} := -\nabla_{\mathcal{L}} \mathbf{w}$, $\Delta b := -\frac{\partial \mathcal{L}}{\partial b}$
   - 2.5. update parameters $\mathbf{w} := \mathbf{w} + \Delta \mathbf{w}$, $b := +\Delta b$

replacement (O)

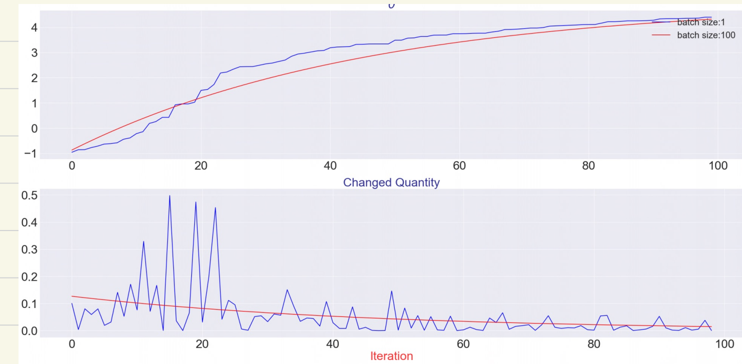
1. Initialize $\mathbf{w} := 0^{m-1}, b := 0$
2. for iteration $t \in [1, \ldots, T]$:
   - 2.1. for $i \in [1, \ldots, m]$ (where m is the minibatch size):
     - 2.1.1. draw random example with replacement: $\langle \mathbf{x}^{[i]}, y^{[i]} \rangle \in \mathcal{D}$
   - 2.2. compute loss $\mathcal{L} := \frac{1}{m} \sum_{i=1}^{m} L(\hat{y}^{[i]}, y^{[i]})$
   - 2.3. compute gradients $\Delta \mathbf{w} := -\nabla_{\mathcal{L}} \mathbf{w}$, $\Delta b := -\frac{\partial \mathcal{L}}{\partial b}$
   - 2.4. update parameters $\mathbf{w} := \mathbf{w} + \Delta \mathbf{w}$, $b := +\Delta b$

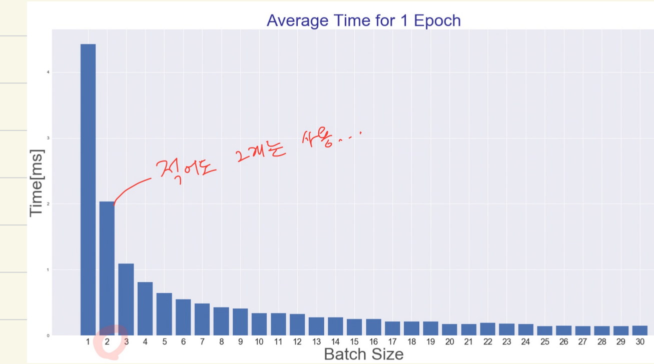
- replacement 경우, epoch 보다는 iteration 이 적절한 표현.
  ↳ 전체 데이터를 순차적으로 1번 학습하는 것이 epoch 인데,
  이 경우는 replacement 로 인해 epoch가 어렵고, 쉽게 epoch인지도 확인하기 어렵다.

16. Batch-size ↑ 효과 (안정적, 빠르게)

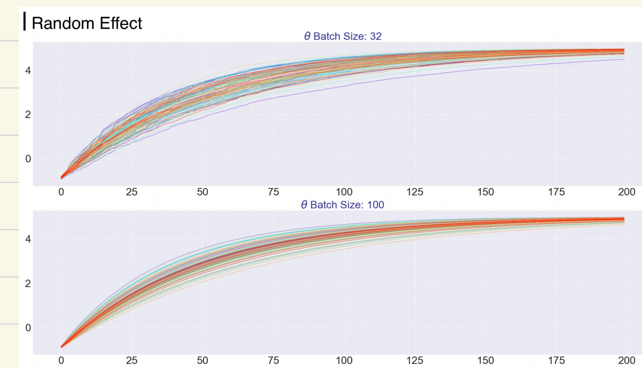
Outlier, random 등 데이터 특성에 대해 robust하게! ≒ Learning rate ↓ 효과와 유사! (Robust 측면)

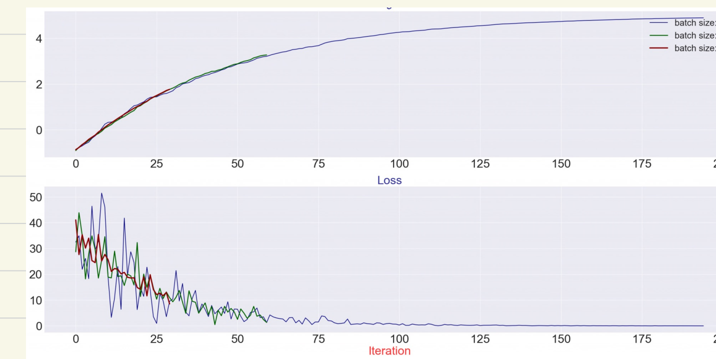
1) outlier 영향이 pooling으로 학습이 보다 안정적. 2) Epoch 당 연산 시간 ↓



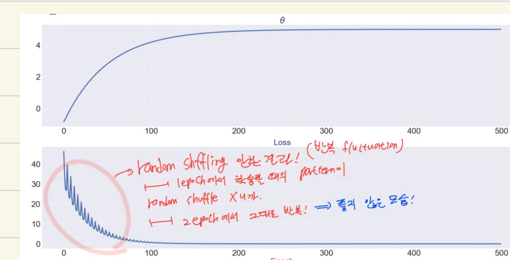


3) Random effect ↓↓ (sampling random으로 해도 안정적인 학습) 4) Iteration 횟수 ↓↓

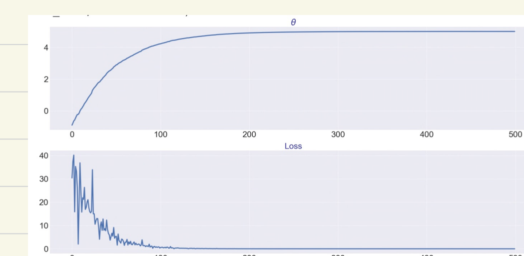




※ Sampling random으로 해야하는 이유.



⟹

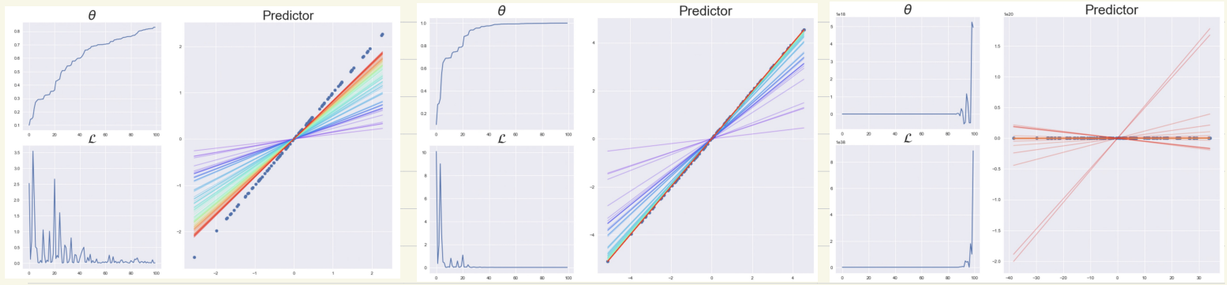


Epoch 마다 학습패턴이 반복되는 모습!

17. Learning with various Standard deviation

- 데이터셋의 표준편차가 클수록, 파라미터 학습의 오독이 넓어짐. (심플게 σ가 크면, θ가 발산함!)
- 따라서, 표준편차가 큰 데이터를 대상으로 할 때, Learning rate를 ↓↓ 할 필요가 있음.

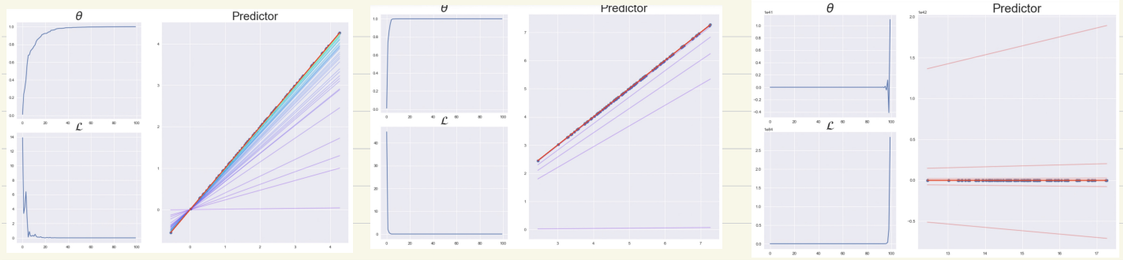
$\sigma = 1$ $\sigma = 2$ $\sigma = 15$



18. Learning with various mean

- 데이터셋 mean의 절대값이 클수록 θ 학습 진동이 커지는 모습!
- std와 마찬가지로, 데이터셋의 (mean)이 클수록 learning rate는 작게 잡을 필요가 있음!

$\mu = 2$ $\mu = 5$ $\mu = 15$



※ 응수도 마찬가지 모습 (-15에서 0과 발산)