Mini_batch optimization

$$X = \Big[\underbrace{\vec{x}^{(1)} \quad \vec{x}^{(2)} \cdots}_{\text{mini-batch 1}} \;\; \underbrace{\quad\cdots\quad}_{\text{mini-batch2}} \cdots \;\; \vec{x}^{(n_s)}\Big]$$

Feed the data one mini-batch at a time.

One full pass through the data is called an "epoch".

```
For i in range(n_epochs):
    For j in range(n_batch):
        Feed_forward
        Back_prop
        update W & b
```

$$n\_batch = \frac{n_s}{batch\_size}$$

If we assume that the time of an epoch is mostly dominated by the n_s and has very little dependence on the # batches, we can roughly take the time of an epoch as the unit of time.

Assignment: Check this!

Plot the time of an epoch vs n_s for diff batch_sizes.

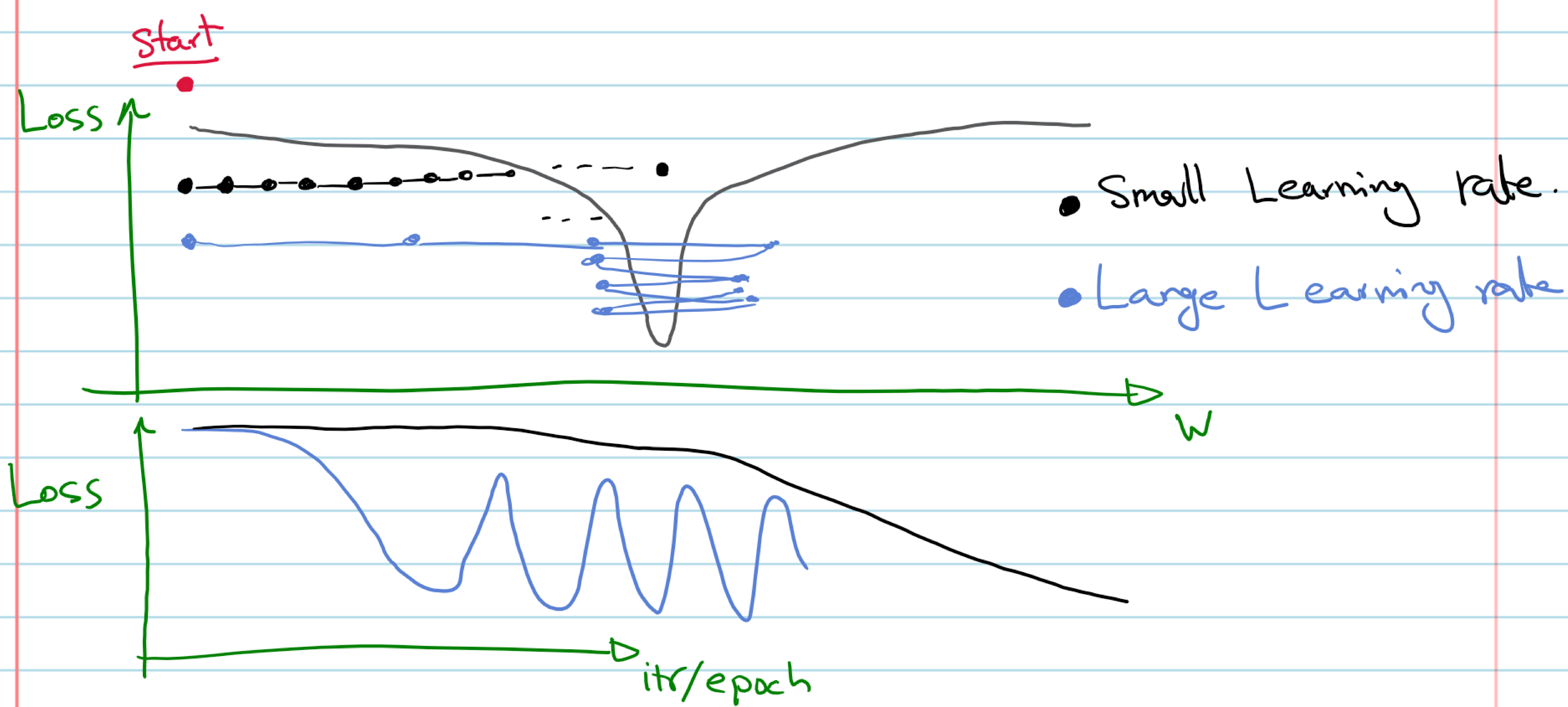
The advantage is that

* First, we use the data more efficiently, i.e. do more GD steps in each epoch.

* Second, the mini-batch GD adds stochasity to the optimization process that can sometimes help scape local minimum.

Adaptive Learning Rates

Consider the following optimization problem:



With a small lr, it takes too many itr/epochs to converge to the minimum. Specially for the flat part of the loss, $\left(\frac{dL}{dw} \sim 0\right)$ the training would be slow and the progress would only start after too many itr/epochs.

A large lr would be helpful to get out of the flat part of the loss function but then it cannot converge.

Ideally, we would need to use an Adaptive lr that depending on where it is, it would use a small or large value for the lr.

Learning Rate decay:

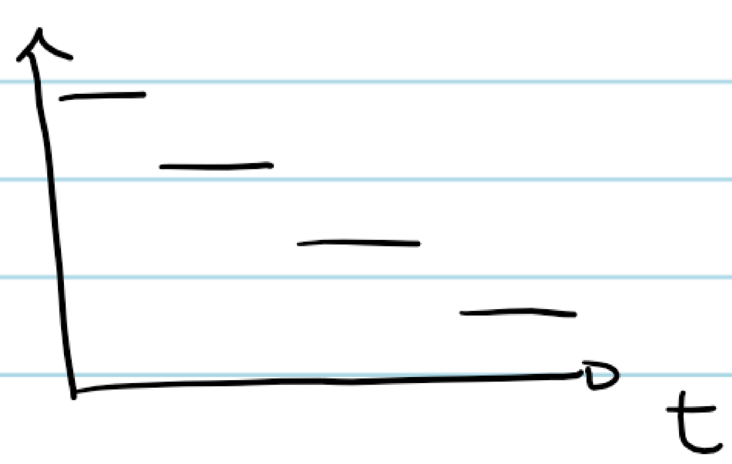
One solution is to decrease the lr.

1) $\alpha = \frac{k}{\sqrt{t}} \alpha_0$ k some const. (hyper param)

t time → itr/epoch

2) $\alpha = e^{-kt} \alpha_0$ → Exponentially decaying

3) α : Stair case

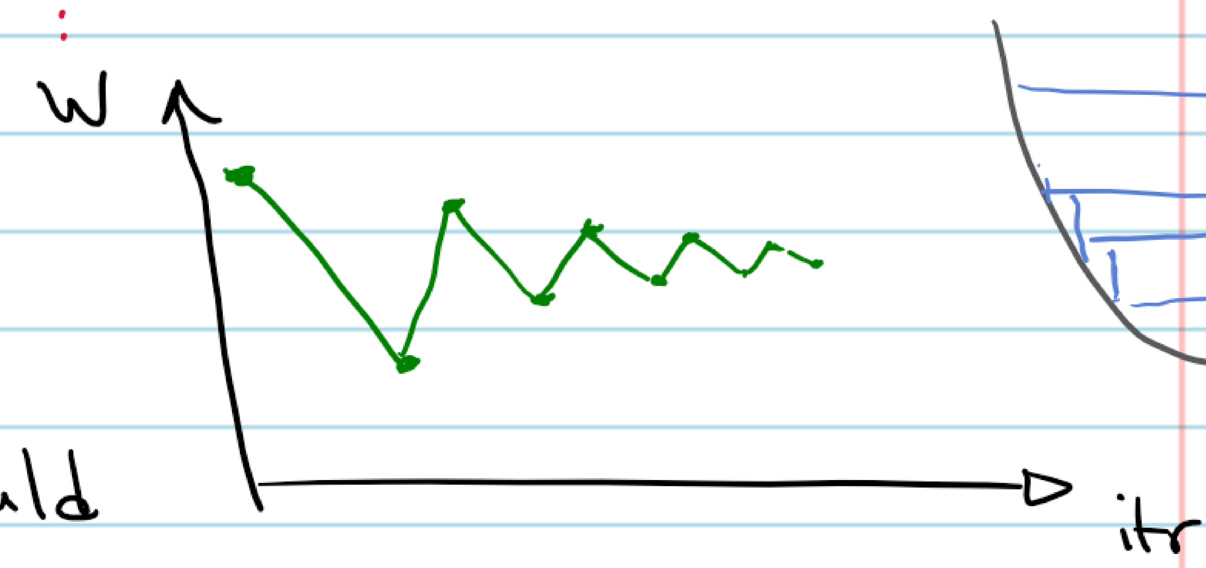


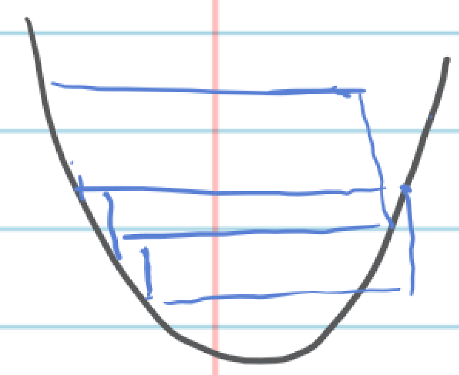
But, this does not fully resolve the problem:

* This still gives the same lr for all the params in the problem and they can be on diff. scales.

* Whether the lr should decrease or not depends on the loss and there could be situations where this does not help.

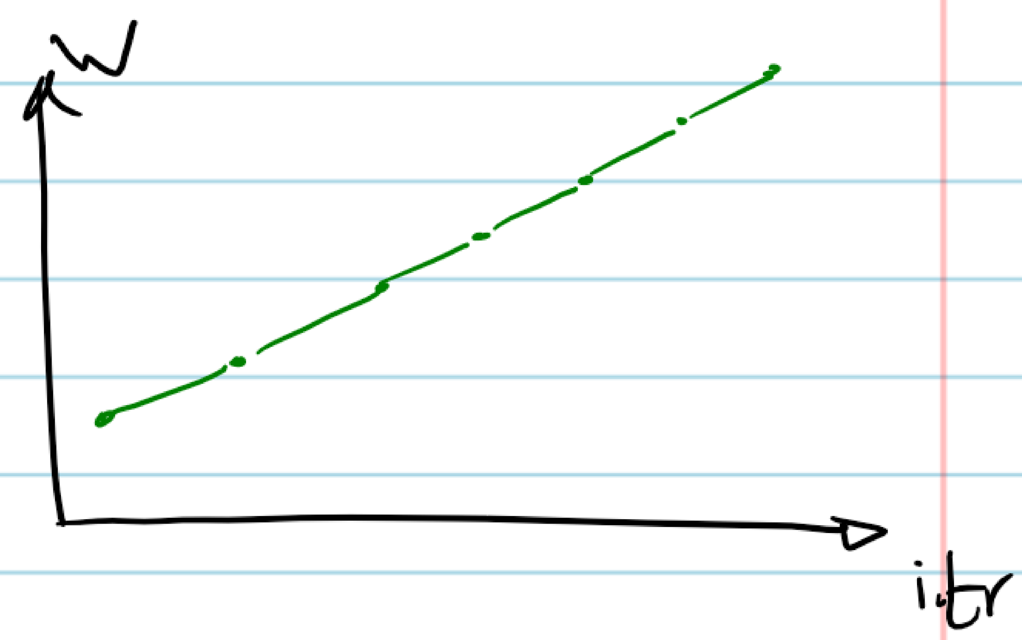
Using the history for adaptive lr:





Consider this situation.

From the oscillations in W, one could guess that the lr should decrease.



Also if, we can guess that a larger lr maybe helpful.

How can we implement our intuition?

We need to look at the history, but probably the more recent history. To do this, we apply a weighted average to the history (of changes, i.e. dW) and take that for the update:

$$W = W - \eta\, dW \longrightarrow W = W - \eta\, \overline{dW}$$

Exponentiall-weighted average is a good choice, i.e. $\frac{\omega_{t-j}}{\omega_{t-j+1}} = \beta, 0<\beta<1$

At each step t+1, we calculate dW_{t+1} (from the loss func).
We also keep the $\{dW_t\}$ list which has the full history of dW, then calculate

$$\overline{dW}_{t+1} = \sum_{t'=0}^{t} \omega_{t'} dW_{t'} + \omega_{t+1} dW_{t+1}$$

$$W = W - \eta\, \overline{dW}$$

For the $\overset{exp}{\text{}}$ weights, we really need to specify the ballence between the history and the current value. This is b/c

$\overline{dW_t}$ has all the history with the right ratio.

$$\overline{dW}_{t+1} = \beta \overline{dW}_t + (1-\beta) dW_{t+1} \ .$$

With the recurssion, we get

$$\overline{dW}_{t+1} = \beta^2 \overline{dW}_{t-1} + \beta(1-\beta) dW_t + (1-\beta) dW_{t+1}$$

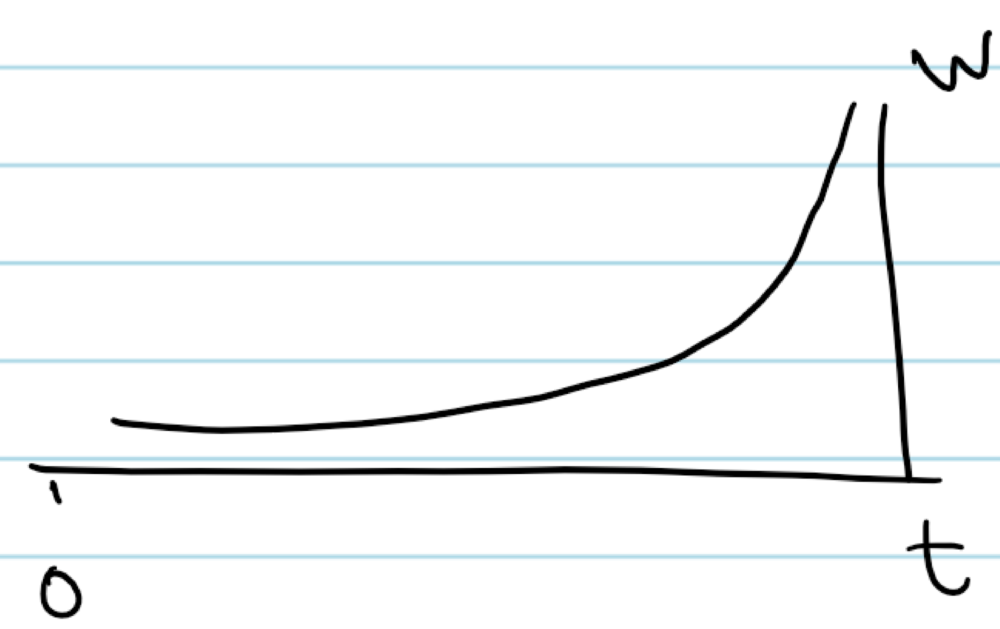
$$\vdots$$

$$=$$

$$\underbrace{\overline{dW}_0}_{0} + (1-\beta)\beta^t dW_1 + \beta^{t-1}(1-\beta) dW_2 + \cdots \beta^{t-k}(1-\beta) dW_{k+1} + \cdots (1-\beta) dW_{t+1}$$

If we plot these weight, they would make an exp function

$$W_t \propto \beta^{-t}$$



So, this would mean for each step

Feed Forward

Backprop $\longrightarrow dW_{t+1}$

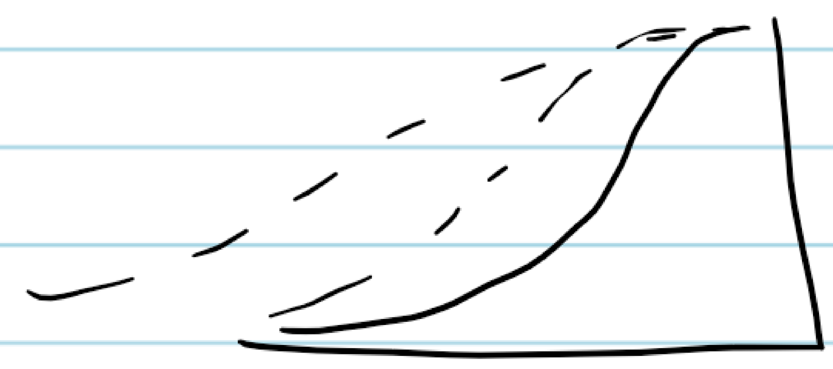
$\overline{dW} = \beta \overline{dW} + (1-\beta) dW_{t+1}$

$W \mathrel{-}= \eta \overline{dW}$

β would be a hyperparameter that determines how much we would want to weigh the history.

$\beta = 0 \rightarrow$ No history

$(\beta \rightarrow 1) \rightarrow$ Slower decay



remember $\frac{w_{t-1}}{w_t} = \beta$.

With this, we can pick a large lr & if it starts fluctuating, the weighted average would be suppress the large values of dW.

$\rightarrow$ See the example in Jupyter NB.

Example:

$dW: \{ 0.5, -0.45, 0.42, -0.4 \}, \quad dW_{t+1} = 0.38$

$\underline{\beta = 0.9}$ $\quad (0.5) \sim ? \rightarrow \overline{dW}$ 0.05

$(0.5)(0.9) + (0.45)(0.1) \rightarrow 0$

$(0.45)(0.9) + (0.42)(0.1) \rightarrow 0.64$

$\vdots$

$\rightarrow$ Needs some work.

RMS Prop:

Alternatively, one could change the lr based on the variance of the dW.

A small var, indicates that the lr is good, on the other hand, a large var, indicates signals that the lr may be large & we may need to decrease it.

It means

$$W \longrightarrow W - \eta \frac{dW}{\sqrt{\overline{Var}}} \qquad \overline{Var}_{t+1} = \beta\, \overline{Var}_t + (1-\beta)\, \underset{dW^2}{\underset{\downarrow}{Var_{t+1}}}$$

We add ε to avoid divergence:

$$W - \eta \frac{dW}{\sqrt{\overline{Var} + \varepsilon}}$$

Similar to the momentum, we use exp-weighted averaging to calculate the variance.

Adam

This combines the rms prop & momentum:

In step $(t+1)$

$$\overline{dW} = \left(\beta_1 \overline{dW} + (1-\beta_1)\, dW_{t+1}\right) / (1-\beta_1^t)$$

This is to compensate for the bias in the first iterations.

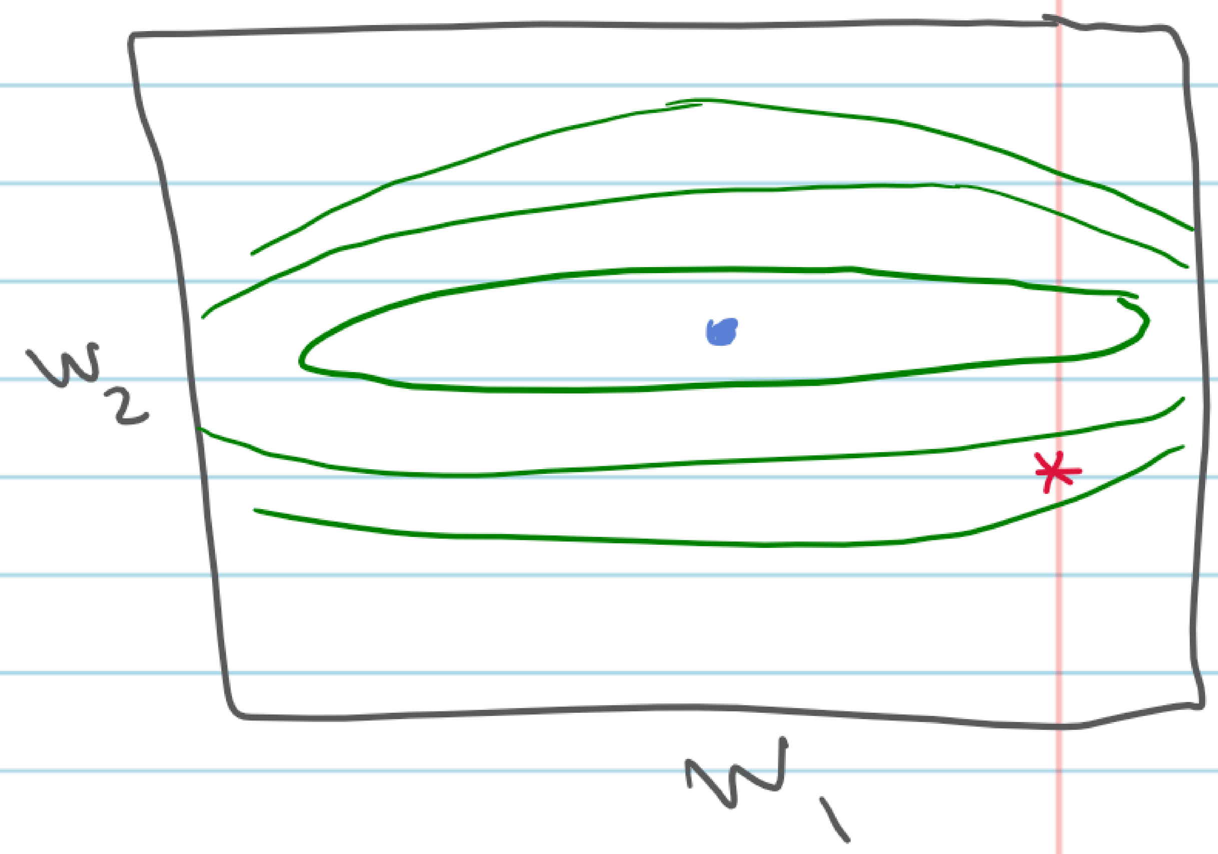
$$\overline{Var} = \left(\beta_2 \overline{Var} + (1-\beta_2)\, dW_{t+1}^{**2}\right) / (1-\beta_2^t)$$

$$W \mathrel{-}= \eta \frac{\overline{dW}}{\sqrt{\overline{Var}} + \varepsilon}$$

The hyperparams are $\{\eta, \underset{0.9}{\beta_1}, \underset{0.99}{\beta_2}, \underset{10^{-8}}{\varepsilon}\}$.

Batch normalization

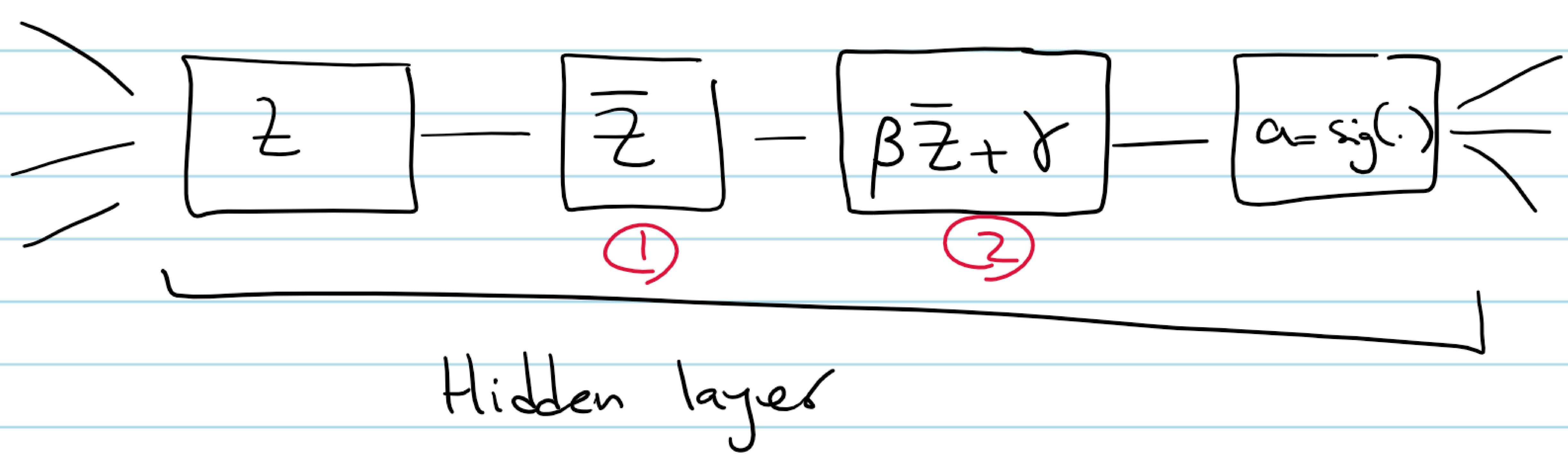
Consider the following situation. This would be hard b/c a large step with w_1 would be small for w_2.



This is why we usually normalize / Standardize our inputs.

But this still could happen for a NN since the hidden layers are not constrained.

But this can be done:



1. Standardize z

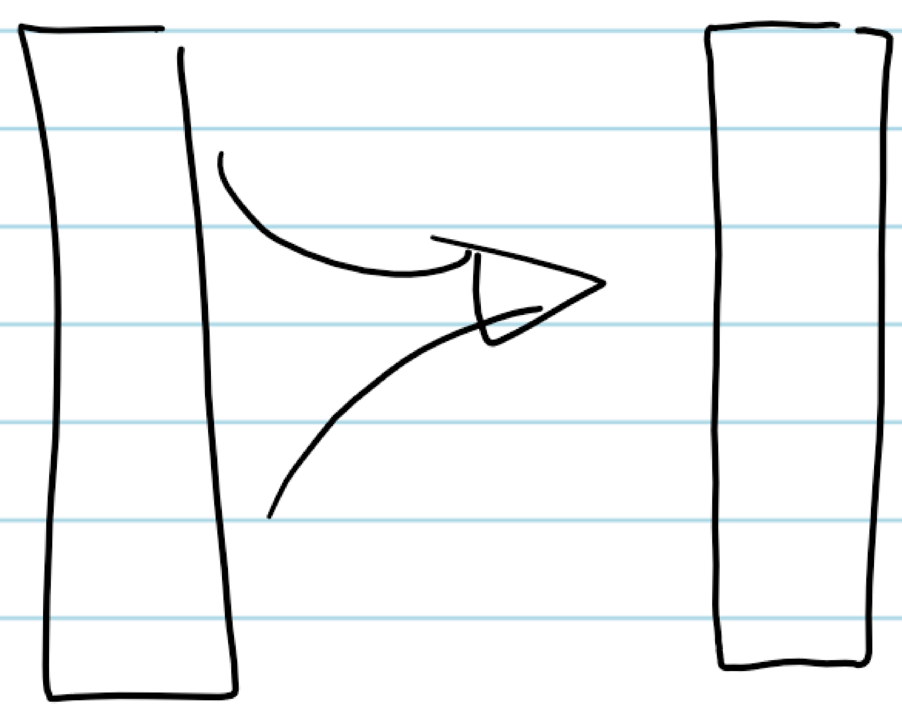
2. Rescale with β & Shift by γ

$\rightarrow$ Params: $\{W, \beta, \gamma\}$ $\rightarrow$ For optimization.

Advantages

* In principle should be easier/faster to optimize/train.

* Quasi-indep. training of hidden layers



W/o the input to the next layer could drastically change in each iteration.
But w normalization, it would have the same mean & var.