

Lecture 3

Gradient Descent Plus

- Weight decay
- Momentum
- Adam

Admin

① TA office hours Tue.

Ridge Regression.

$$\underset{\vec{w}}{\operatorname{argmin}} \mathcal{L}(\vec{w}) + \lambda \|\vec{w}\|^2.$$

↙ hyperparameter

Gradient step : $\vec{w}_{t+1} = \vec{w}_t - \eta (\nabla \mathcal{L}(\vec{w}_t) + \lambda \vec{w}_t).$

$$\vec{w}_{t+1} = \underbrace{(1 - \eta \lambda)}_{\text{weight decay}} \vec{w}_t - \eta \nabla \mathcal{L}(\vec{w}_t)$$

- Shifts weight towards zero.
- Can be beneficial to do this instead of regularization.

Last time:

Gradient Descent:

$$\mathcal{L}(w) = \frac{1}{2} \|X\vec{w} - \vec{y}\|^2.$$

$$\vec{w}_{t+1} = (I - \eta X^T X) \vec{w}_t + \eta X^T \vec{y}$$

Implicit Regularization

$$V^T \vec{w}_{t+1} = V^T \vec{w}_t - \eta V^T (V \Sigma^T U^T) (X \vec{w}_t - \vec{y})$$

$$\tilde{\vec{w}}_{t+1} = \tilde{\vec{w}}_t - \eta \Sigma^T (U^T \Sigma V^T \vec{w}_t - U^T \vec{y})$$

$$= \tilde{\vec{w}}_t - \eta \Sigma^T (\Sigma \tilde{\vec{w}}_t - U^T \vec{y})$$

$$\tilde{w}_{t+1,i} = \tilde{w}_{t,i} - \eta \cdot [\sigma_i^2 \tilde{w}_{t,i} - \sigma_i \tilde{y}_i]$$

$$= \tilde{w}_{t,i} - \eta \cdot \sigma_i [\sigma_i \tilde{w}_{t,i} - \tilde{y}_i]$$

"Early Stopping"

Gradient Descent.

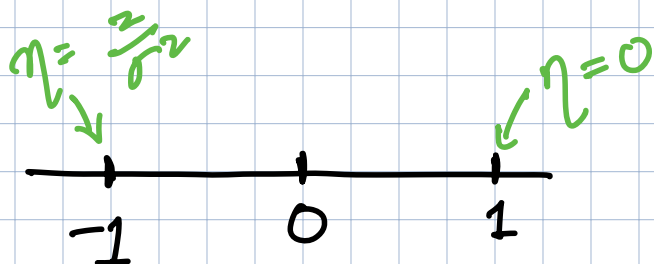
Scalar case: $\sigma w = y$

One direction of SVD basis.

$$\begin{aligned}w_{t+1} &= w_t + \eta \cdot \sigma(y - \sigma w_t) \\&= (1 - \eta \sigma^2) w_t + \eta \cdot \sigma \cdot y.\end{aligned}$$

$$\begin{aligned}\Rightarrow |1 - \eta \sigma^2| &< 1 \\ \Rightarrow -1 &< 1 - \eta \sigma^2 < 1\end{aligned}$$

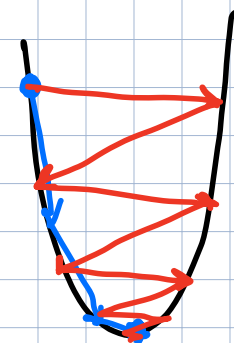
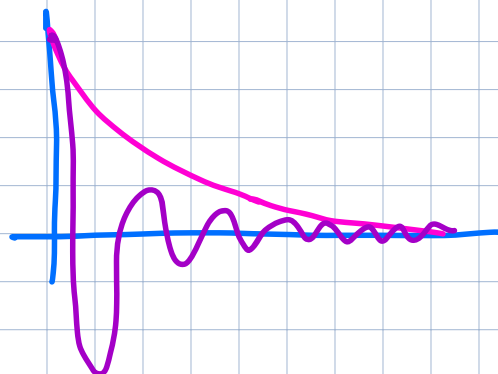
~~or~~

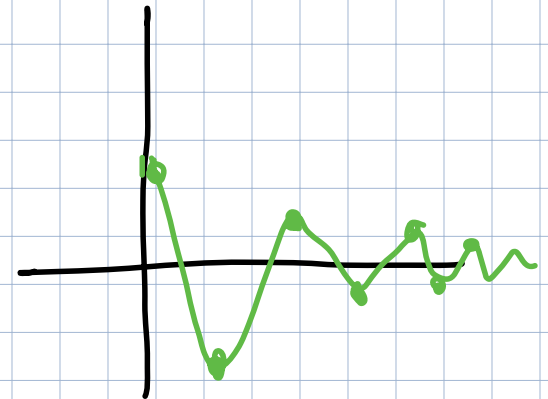
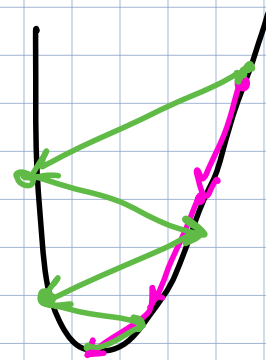
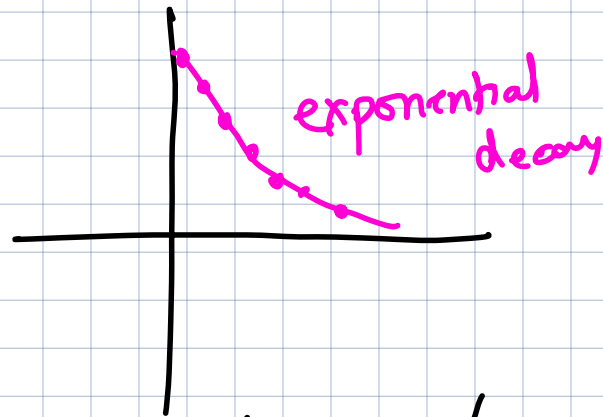
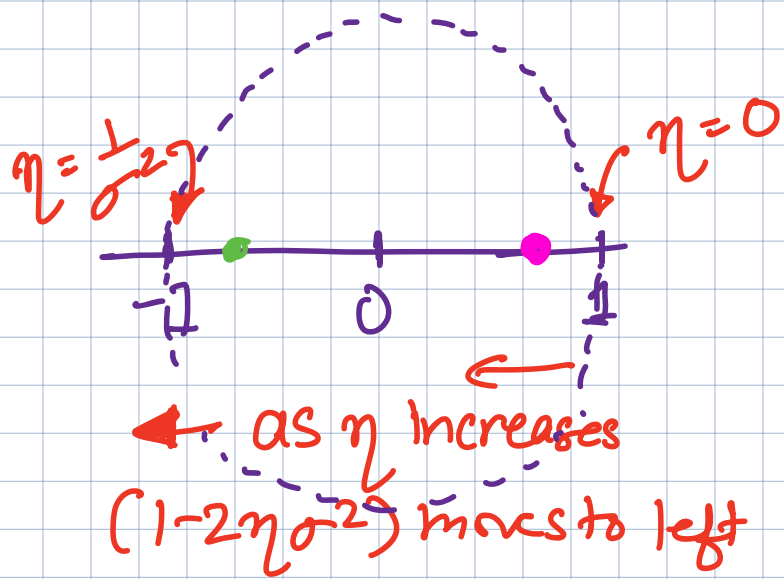


$$\Rightarrow -2 < -\eta \sigma^2$$

$$\Rightarrow \eta < \frac{2}{\sigma^2}$$

$$0 < \eta.$$





Can we smooth out the oscillations?

So how can we smooth out the oscillations?

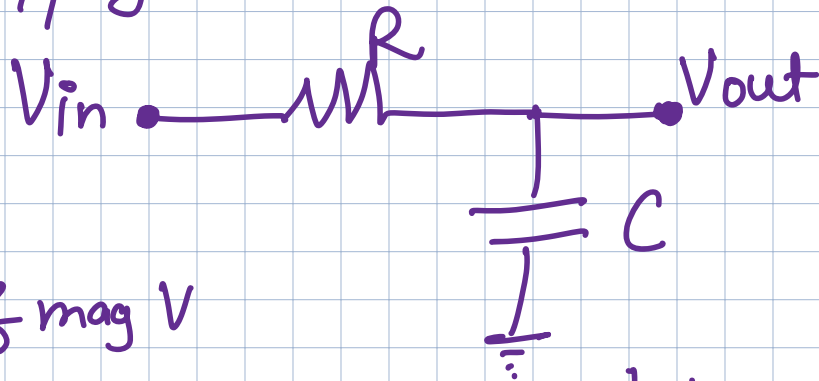
→ Averaging, Low-pass filtering

One strategy: $\frac{1}{t} \sum_{i=1}^t \nabla f(w_i) \rightarrow$ straight average.

→ Requires lots of memory ... have to remember all the past gradients.

→ In physics / state-space systems
a state can be used for averaging.

A circuit / physics intuition:



Low-pass filter

If $V_{in} = \text{step of mag } V$

$$V_{out} = V(1 - e^{-\frac{1}{RC}t})$$

Momentum : Another tweak on Gradient Descent. // How to make optimization go faster?
Add memory.

$$\vec{w}_{k+1} = \vec{w}_k - \eta \cdot \vec{z}_{k+1}$$

$$\vec{z}_{k+1} = \beta \cdot \vec{z}_k + (1-\beta) \nabla f(\vec{w}_k), \quad z_0 = 0. \quad \beta \in [0, 1]$$

If $\beta = 0$, $z_{k+1} = \nabla f(w_k)$. i.e same old gradient descent.

$$z_1 = (1-\beta) \nabla f(w_0)$$

$$z_2 = \beta \cdot (1-\beta) \cdot \nabla f(w_0) + (1-\beta) \nabla f(w_1).$$

$$z_3 = \beta^2 \cdot (1-\beta) \cdot \nabla f(w_0) + \beta \cdot (1-\beta) \nabla f(w_1) + (1-\beta) \nabla f(w_2)$$

$$z_{t+1} = (1-\beta) \sum_{k=0}^t \beta^{t-k} \nabla f(w_k)$$

Note on implementation: $(1-\beta)$ factor can be absorbed into learning rate in some libraries

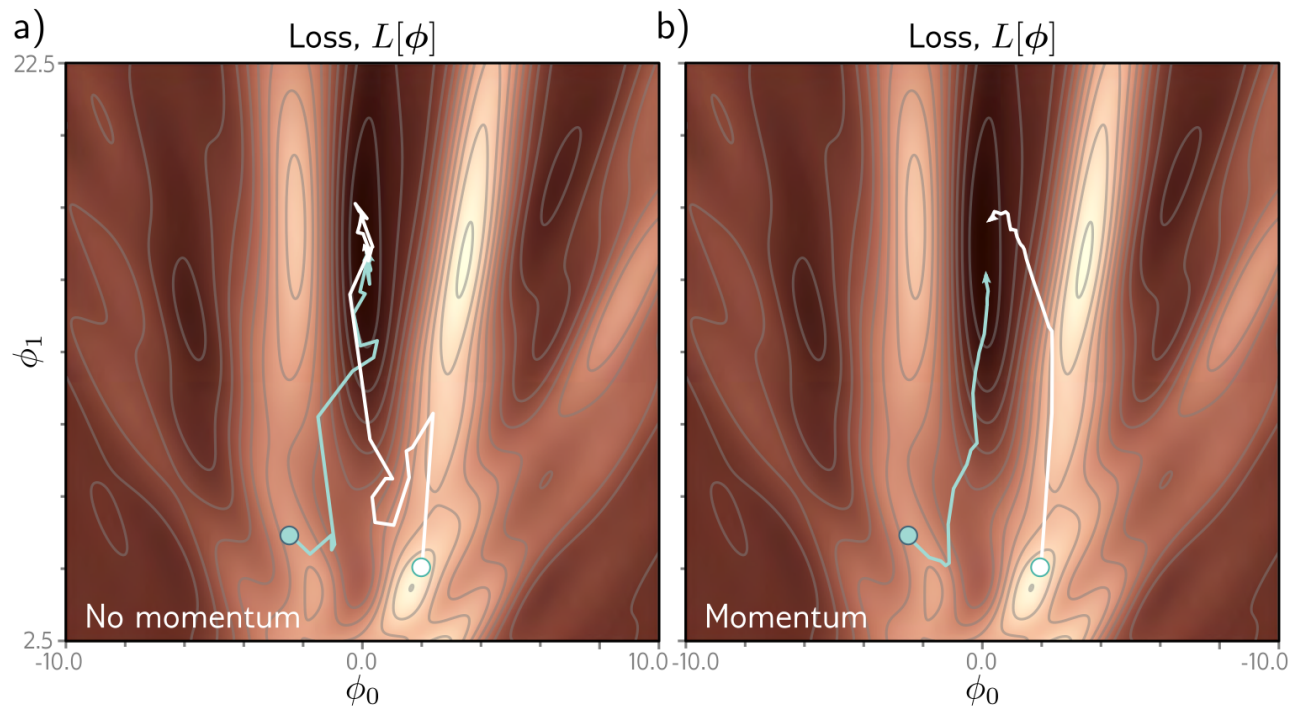


Figure 6.7 Stochastic gradient descent with momentum. a) Regular stochastic descent takes a very indirect path toward the minimum. b) With a momentum term, the change at the current step is a weighted combination of the previous change and the gradient computed from the batch. This smooths out the trajectory and increases the speed of convergence.

Example:

$$\text{So } \nabla f(w_t) = (-1)^t \cdot c$$

$$\frac{\|z_{t+1}\|}{\|c\|} = \frac{1-\beta}{1+\beta}$$

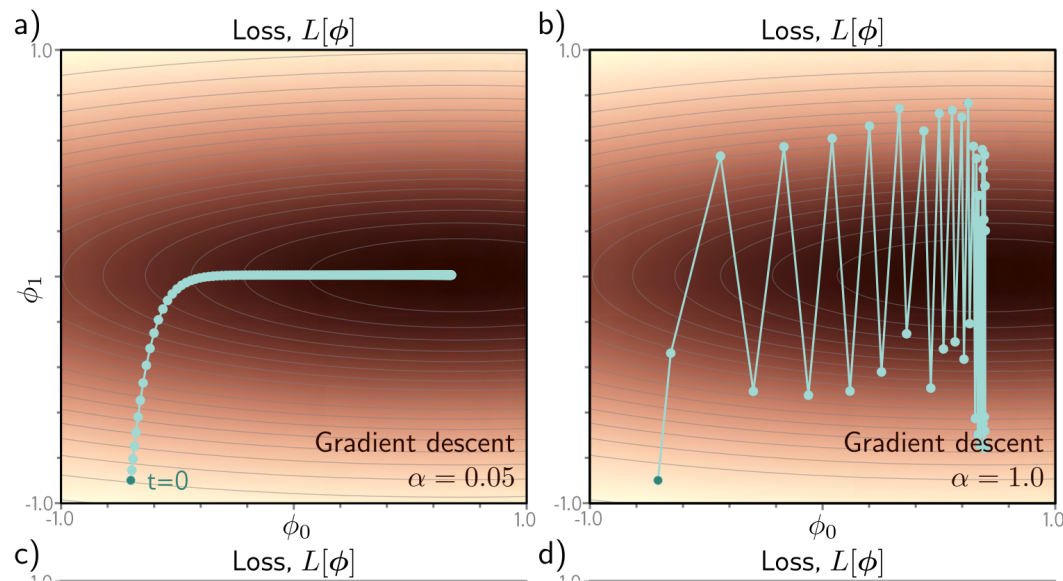
$$\sum_{i=1}^{\infty} \alpha^i = \frac{1}{1-\alpha}$$

Adam

Gradient Descent has a feature/bug $\rightarrow$ takes bigger steps in directions with large singular values and smaller steps in directions with small singular values.

- Advantage: early stopping $\leftrightarrow$ Implicit regularization.
- Disadvantage: hard to choose a learning rate that
 - (a) makes enough progress in all directions
 - (b) doesn't blow up.

So how do you solve this? ... Brainstorm.



Prince
2025

• Ideal world: compute SVD and normalize.

→ Too expensive ☹

→ Normalize the gradients

$$\vec{w}_{t+1} = \vec{w}_t - \eta \cdot \nabla_{\vec{w}} \mathcal{L}(\vec{w}) \Big|_{\vec{w} = \vec{w}_t}$$

Regular GD.

Instead:

$$\vec{m}_t \leftarrow \nabla_{\vec{w}} \mathcal{L}(\vec{w}). \quad \text{gradient}$$

$$\vec{v}_t \leftarrow \left(\nabla_{\vec{w}} \mathcal{L}(\vec{w}) \right)^2 \quad \text{squared gradient.}$$

$$\frac{\vec{m}_t}{\sqrt{\vec{v}_t}} = \frac{\vec{m}_t}{\sqrt{\vec{m}_t^2}}$$

$$\vec{w}_{t+1} = \vec{w}_t - \eta \cdot \frac{\vec{m}_{t+1}}{\sqrt{\vec{v}_{t+1}} + \epsilon}$$

→ $\epsilon > 0$ to avoid division by zero.
 $\sqrt{\vec{v}_{t+1}}$: Take positive square root.

$$\frac{\vec{m}_{t+1}}{\sqrt{\vec{v}_{t+1}}} = \text{sgn} \left(\nabla_{\vec{w}} \mathcal{L}(\vec{w}) \right) \quad \text{"sign SGD"}$$

What's the problem?

Once you are close you might keep overshooting and bouncing.

So can this be smoothed out?

→

$$\begin{cases} \vec{m}_{t+1} = \beta \cdot \vec{m}_t + (1-\beta) \nabla_w \mathcal{L}(\vec{w}_t) \\ \vec{v}_{t+1} = \gamma \cdot \vec{v}_t + (1-\gamma) (\nabla_w \mathcal{L}(\vec{w}_t))^2 \\ \vec{w}_{t+1} = \vec{w}_t - \eta \frac{\vec{m}_{t+1}}{\sqrt{\vec{v}_{t+1}} + \epsilon} \end{cases}$$

But there can be a bias - Why?

Say $\beta = 0.9$. Then $\vec{m}_1 = 0.9[0] + \underbrace{(0.1) \cdot \nabla_w \mathcal{L}(\vec{w})}_{\text{too small}} \Big|_{\vec{w}=\vec{w}_0}$.

too small
Can lead to slow start.

"Bias Correction"

$$\vec{m}_{t+1}^{\sim} = \frac{\vec{m}_{t+1}}{1 - \beta^{t+1}}$$

$$\vec{v}_{t+1}^{\sim} = \frac{\vec{v}_{t+1}}{1 - \gamma^{t+1}}$$

If all gradients had mean μ

$$E[m_t] = (1 - \beta^t) \mu.$$

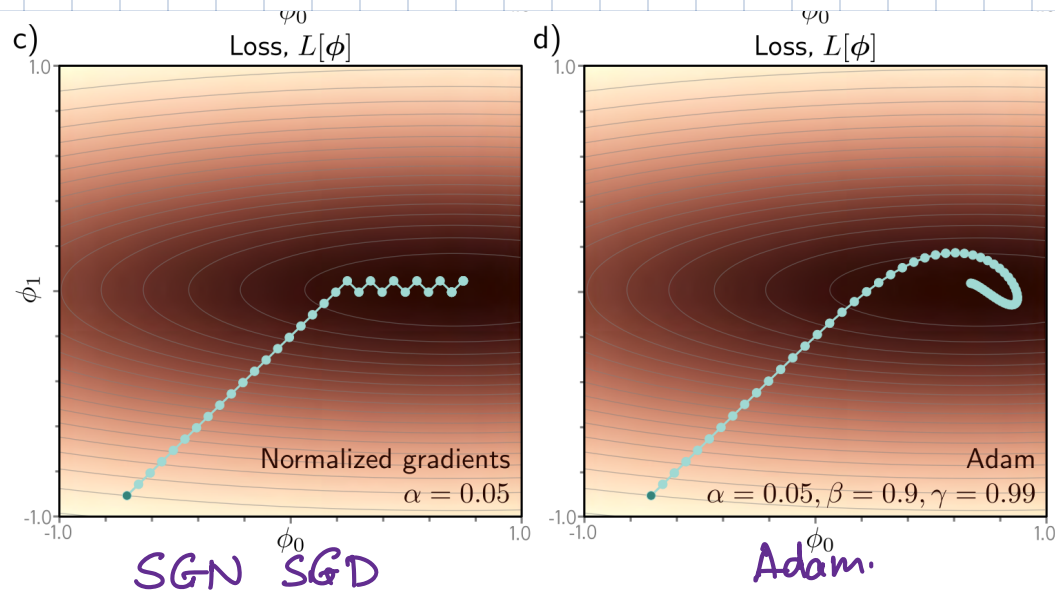
Adam:

$$\vec{w}_{t+1} = \vec{w}_t - \eta \cdot \frac{\vec{m}_{t+1}^{\sim}}{\sqrt{\vec{v}_{t+1}^{\sim} + \epsilon}}$$

Usually: Used with SGD, so compute $\vec{m}, \vec{v}$ from mini-batches

Normalization helps with exploding / vanishing gradients

→ Even if the gradient is getting tiny, the size of the step is largely dictated by the learning rate.



since 2025.

Memory:

SGD

SGD + Momentum

x2

Adam

x3

AdamW: Adam with weight decay.

Recall: Ridge regression:

GD update: $\bar{w}_{t+1} = \underbrace{(1 - 2\eta\lambda)}_{\text{pushing weights down}} \bar{w}_t - \eta \nabla \mathcal{L}_w(\bar{w}_t).$

In Adam: magnitude of gradient keeps being normalized

→ So adding a regularizer to the loss doesn't work as well

AdamW = Adam with weight decay.

$$\text{AdamW: } \vec{w}_{t+1} = (1-\lambda)\vec{w}_t - \eta \cdot \frac{\tilde{m}_{t+1}}{\sqrt{\tilde{v}_{t+1} + \epsilon}}$$

$$\begin{aligned}\vec{m}_{t+1} &= \beta \cdot \vec{m}_t + (1-\beta)(\nabla_w L(\vec{w}_t) + \lambda \vec{w}_t) \\ \vec{v}_{t+1} &= \gamma \cdot \vec{v}_t + (1-\gamma)(\nabla_w L(\vec{w}_t) + \lambda \vec{w}_t)^2\end{aligned}$$

Stepping back

Why do we standardize data in ML?

→ Want features on similar orders of magnitude. → else large features will dominate.
↳ Better conditioning → Ice-cream flavor prediction

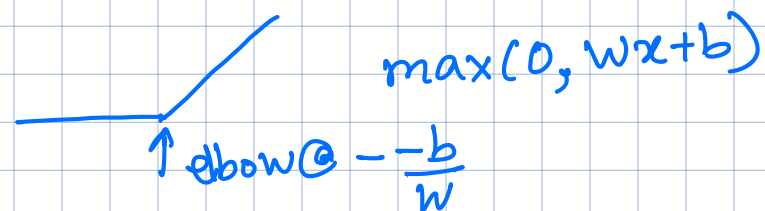
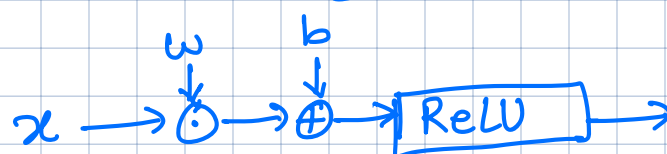
→ Numerical issues and numerical precision.

→ Want to be able to "use" the non-linearity for the expressive power of neural nets.

What happens with non-standardized data?

Consider ReLU.

Scalar case



Say we initialize

$$b \sim N(0,1)$$

$$w \sim N(0,1)$$

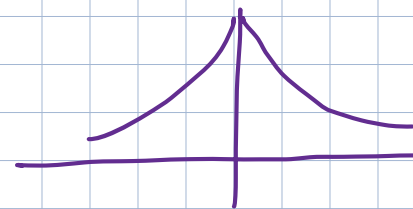
then $-\frac{b}{w} \sim$ Cauchy distribution.

$$\text{PDF: } \frac{1}{\pi} \left[\frac{1}{x^2+1} \right]$$

Properties: "0-mean"

∞ variance.

But mostly around 0.



50% of probability mass in -1 to 1

70% -2 to +2

80% -3 to +3

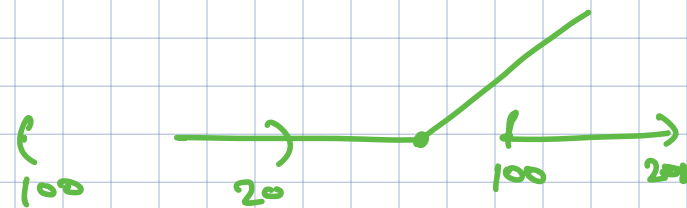
90% -7 to +7

We want our values to be where the non-linearity is interesting

→ True for other non-linearities too Tanh, sigmoid

What happens if we don't do this?

Suppose our data was in the range 100 to 200.

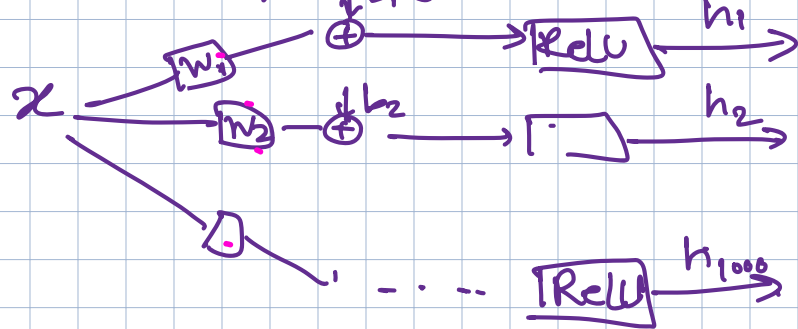


Probability of a corner landing in that range is about 0.6%.

So only 1-2 units out of 1000 may have corners there.

Lose expressive power ;)

Mathematically, why is this bad?



if no corners:

$$h_i = \begin{cases} 0 & \forall x \in (100, 200) \\ \alpha x + \beta & \forall x \in (100, 200) \end{cases}$$

$$\begin{bmatrix} \alpha_1 x_1 + \beta_1 \\ \alpha_1 x_2 + \beta_1 \\ \vdots \\ \alpha_1 x_n + \beta_1 \end{bmatrix}$$

$$\alpha_{1000} x_1 + \beta_{1000}$$

$$\alpha_{1000} x_n + \beta_{1000}$$

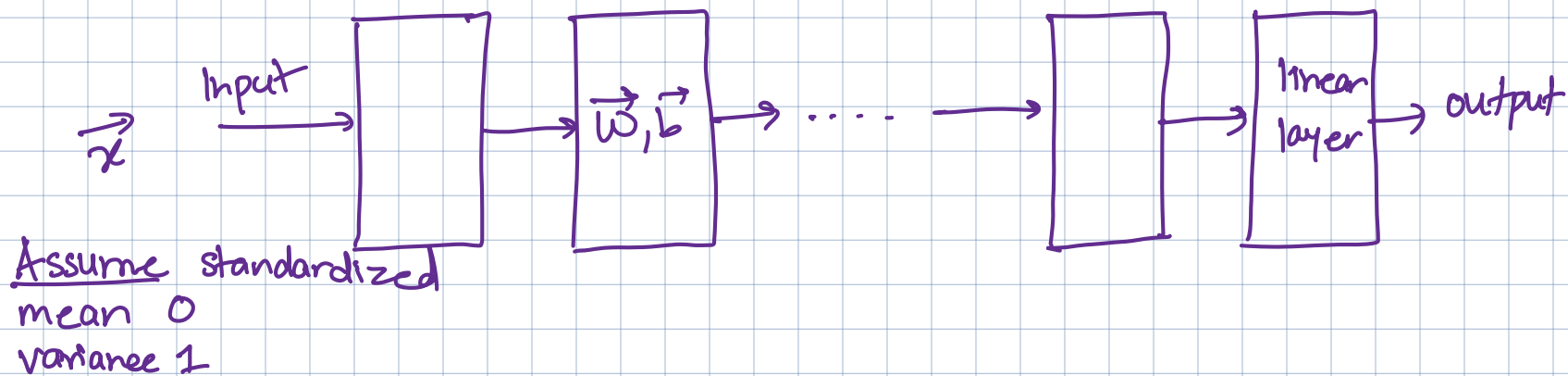
$$\begin{bmatrix} \alpha_1 x_1 & \dots & \alpha_{1000} x_1 \\ \vdots & & \vdots \\ \alpha_1 x_n & \dots & \alpha_{1000} x_n \end{bmatrix}$$

→ sum of two rank one matrices

$$\begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_{1000} \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_{1000} \end{bmatrix} + \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_{1000} \end{bmatrix} \begin{bmatrix} \alpha_1 x_1 \\ \alpha_1 x_2 \\ \vdots \\ \alpha_1 x_n \end{bmatrix}$$

Debugging tip: Inspect rank of feature matrix
or
singular
values

Initialization



How to initialize?

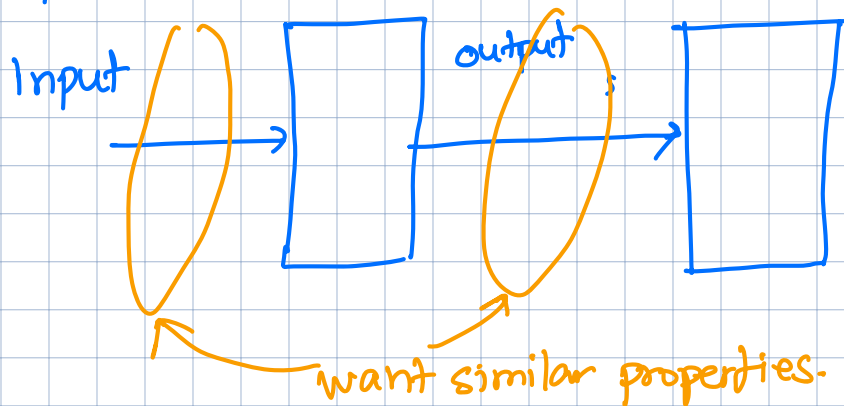
→ Set everything to 0?

But then all inputs from the previous layer get multiplied by zero

Gradients are zero, weights never move.

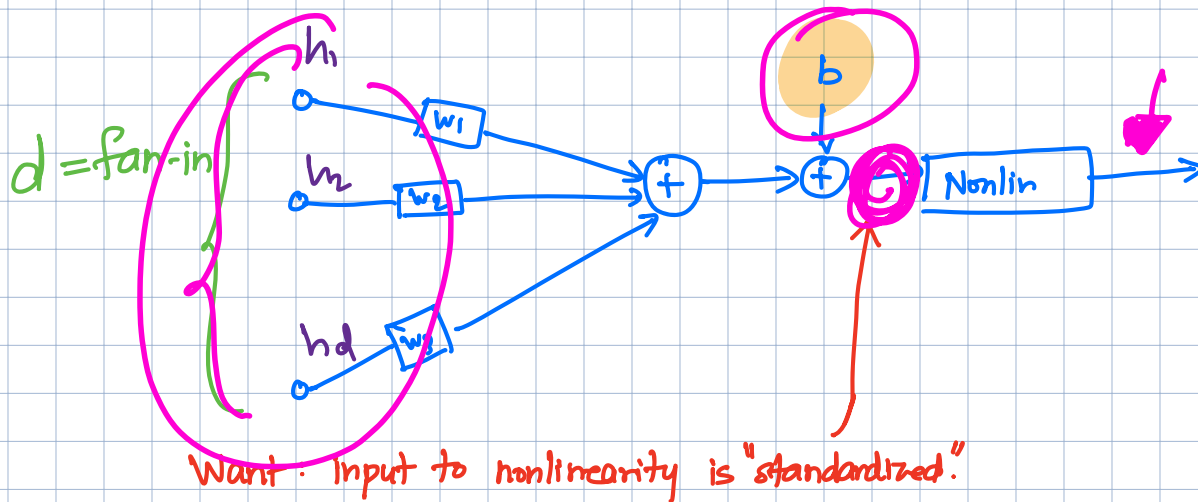
→ So what do we do?

Key observation: Output activations from one layer are inputs to the next layer.



Goal: Can each layer have inputs that start "standardized"?

Xavier initialization (weights, i.e. parameters that multiply data)



$$E[h_i w_i] = E[h_i] E[w_i]$$

Approx "0-mean"

"unit-variance"

Adding d terms: $w_i \sim N(0, \frac{1}{d})$ ← variance.

$$\text{var}(\sum w_i h_i)$$

$$= E[(\sum w_i h_i)^2] - (E(\sum w_i h_i))^2$$

$$= \sum_d E[w_i^2] \cdot E[h_i^2] \quad \text{because independence and zero-mean.}$$

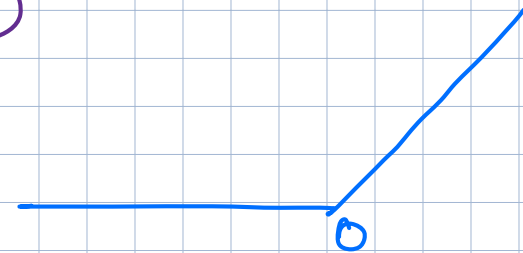
$$= \sum_d \frac{1}{d} E[h_i^2]$$

He Initialization

ReLU non-linearities. on previous layer

Observation: Output of a ReLU is zero half the time.

$$w_i \sim N(0, \frac{1}{d/2}) = N(0, \frac{2}{d})$$



Bias Initialization

- (a) Initialize at 0.
- (b) Use other small number 0.01.
- (c) Treat as Xavier with initialization d_{in} .

Glorot

$$N(0, \frac{2}{d_{in} + d_{out}})$$