Lecture 4

- SGD Convergence continued.
- Adam
- Neural Networks as Generalized Linear Models

Admin

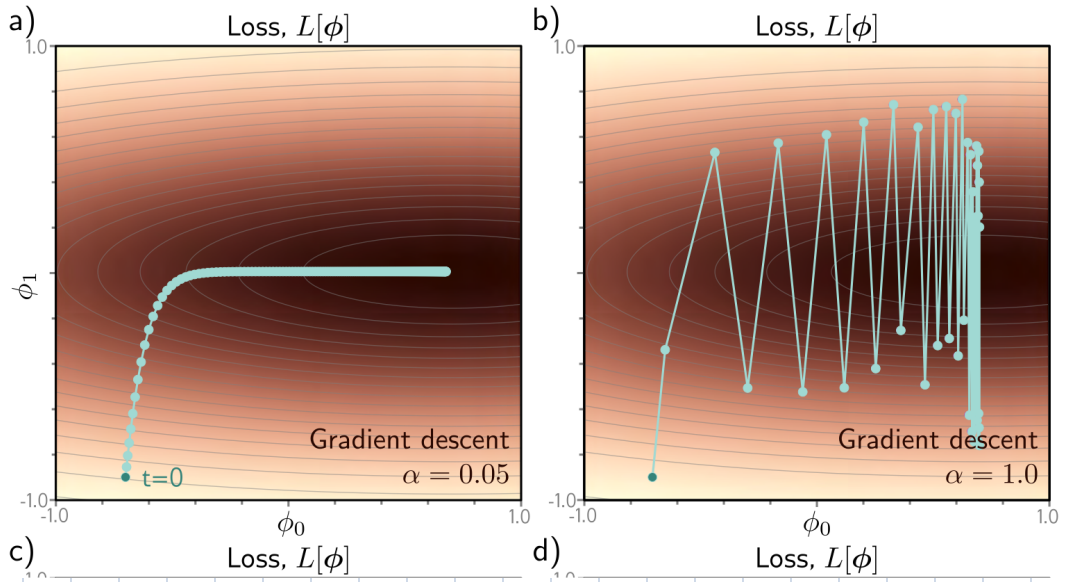
1. Waitlist Cleared!!!
2. 3 New staff
   - Joey, Lance, Kevin.
3. TA office hours start today.
4. 3 new discussion sections added!
5. OH + Study groups

Adam

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

- Advantage: early stopping ↔ 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:

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

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

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

→ $\varepsilon > 0$ to avoid division by zero.

$\sqrt{\vec{v}_{t+1}}$: Take positive squareroot.

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

What's the problem?

Once you are close you might keep overshooting and bouncing.

So can this be smoothed out?

→

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

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

But there can be a bias - Why?

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

too small
Can lead to slow start.

"Bias Correction"

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

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

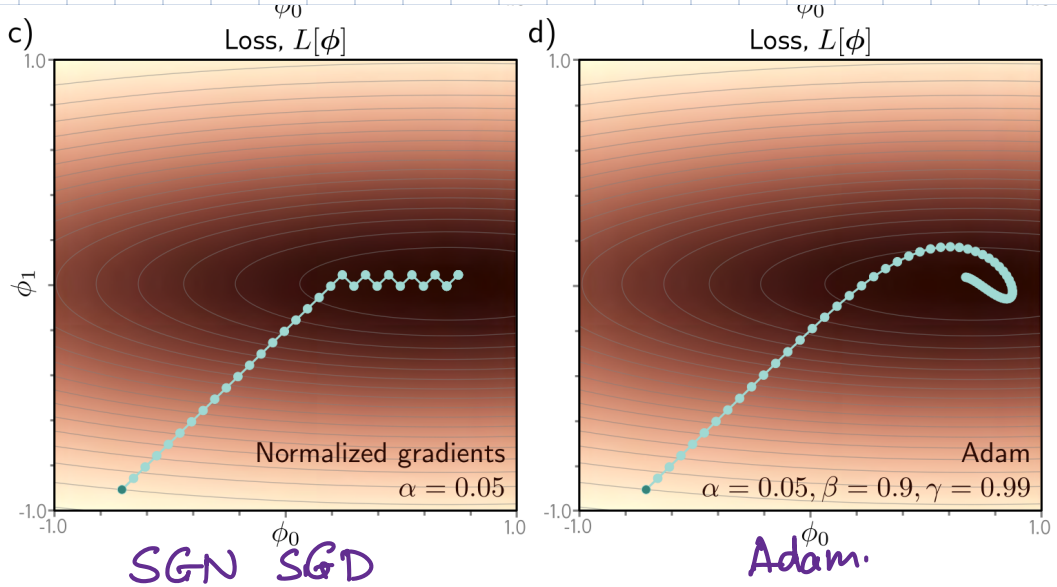
Adam:

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

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.



SGN SGD

Adam.

Since 2025.

Memory:

SGD

SGD + Momentum $\times 2$

Adam $\times 3$

AdamW : Adam with weight decay.

Recall: Ridge regression:

GD update: $\vec{w}_{t+1} = (1-2\eta\lambda)\vec{w}_t - \eta \nabla \mathcal{L}_w(\vec{w}_t)$.

pushing weights down.

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.

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

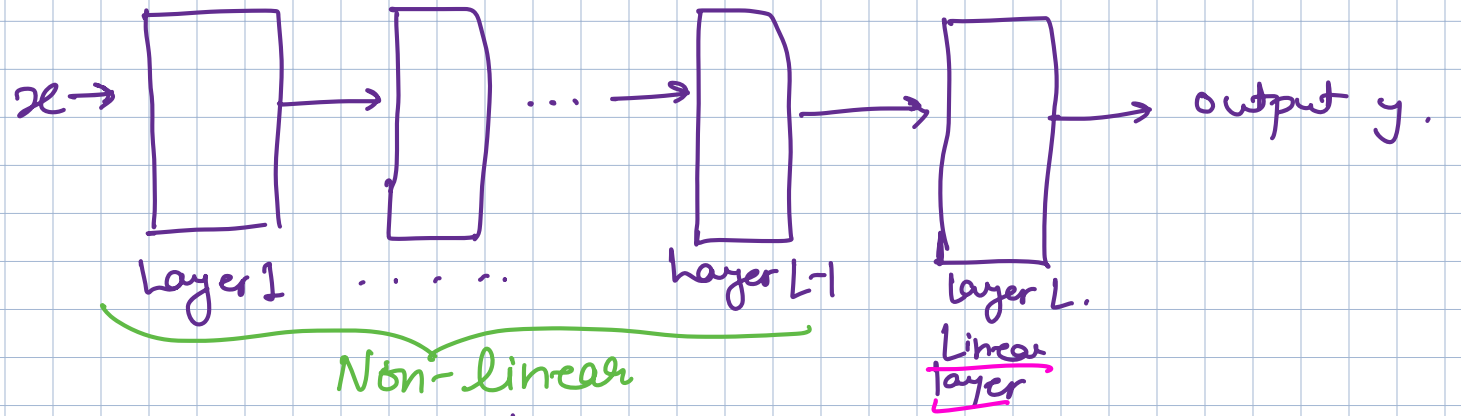
Linearized models

So how can we understand how these algorithms behave on neural networks?

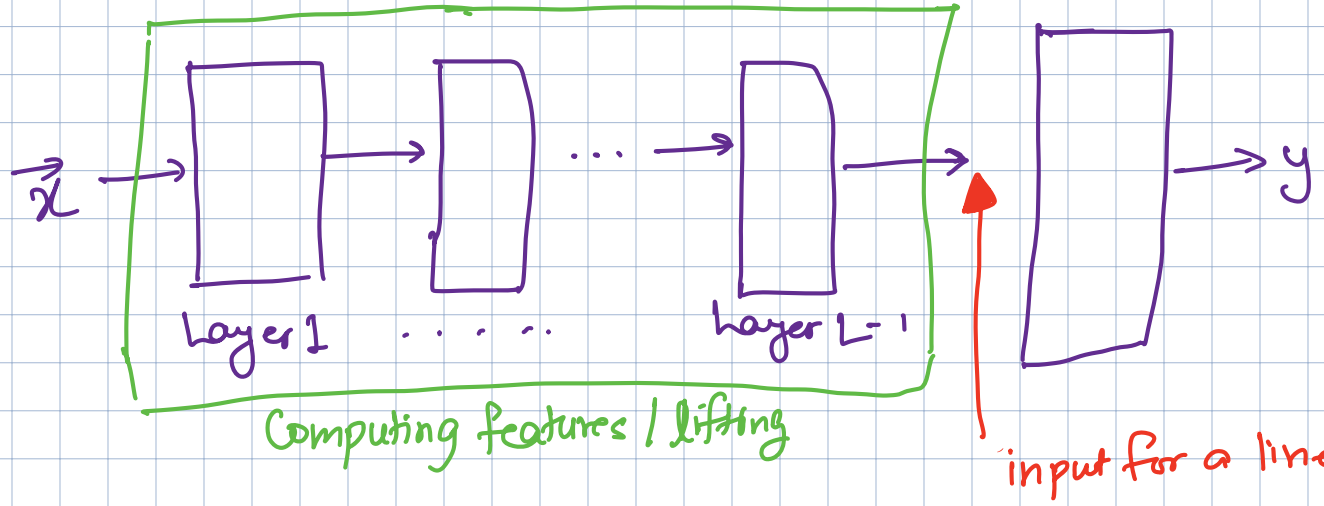
Not nice and convex...

→ Standard approach. Try to find a simpler model that can help understanding

→ Linear models!!



How do I view this as a linear model?



Classical perspective

- Learn the features from data instead of committing to them in advance.
- To reuse my computed features, I can freeze all but the linear layer and retrain → Linear Probing.

Other ideas?

Network: $f(\vec{x}, \vec{\theta})$ $\quad \vec{x}$: data $\quad \vec{\theta}$: parameters $\quad \vec{\theta} = \begin{bmatrix} \vec{w} \\ \vec{b} \end{bmatrix} \in \mathbb{R}^m$

Linearization can come from Taylor expansion!

$\vec{\theta}$ is changing as we train the network.

$f: \mathbb{R}^m \to \mathbb{R}^n$

$\nabla_\theta f \in \mathbb{R}^{m \times n}$ (m × n)

$$f(x, \theta_0 + \Delta\theta) = f(x, \theta_0) + \langle \nabla_\theta f, \Delta\theta \rangle + \cdots$$

$$(\nabla_\theta f)^T = \frac{\partial f}{\partial \theta}$$

$$\langle \nabla_\theta f, \Delta\theta \rangle = \frac{\partial f}{\partial \theta} \cdot \Delta\theta$$

"Linear approximation" of our network.

$\frac{\partial f}{\partial \theta}$ is multiplying the linear term. Those are features

Lazy Training assumption: Weights are moving by very small amounts. ← often true in practice

Why is this reasonable?

This perspective on linearization led to the "Neural Tangent Kernel" perspective.

→ In the limit of infinite width networks, GD will converge to zero training loss in the over-parameterized regime.

Can show: (How NN evolves through training.)

$$\frac{df(x,\theta)}{dt} = -\frac{1}{N}\sum_{i=1}^{N} \underbrace{\nabla_\theta f(x,\theta)^T \nabla_\theta f(x^{(i)},\theta)}_{\text{inner product of features @ } x \text{ with features @ } x^{(i)}} \underbrace{\nabla_f \mathcal{L}(f, y^{(i)})}_{\text{loss @ } y.}$$

$K(x, x^{(i)})$

If Lazy training assumption holds, parameters are not moving much and can show that $K(x, x^{(i)})$ also not changing.

→ Can show GD will converge