

CSCI 1470

Eric Ewing

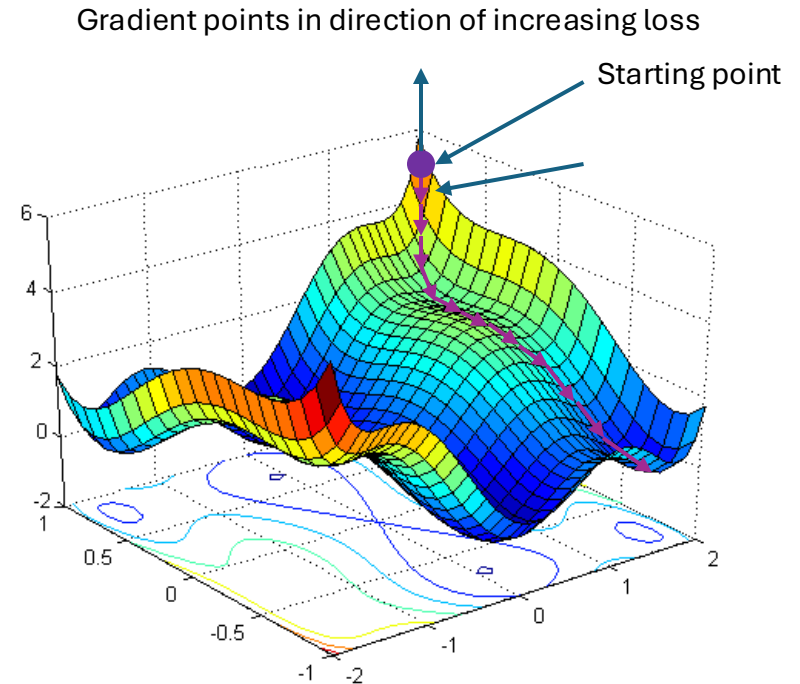
Monday,
2/3/25

Deep Learning

Day 6: Backprop and SGD

Where We left Off: Gradient Descent

- Algorithm to optimize differentiable functions
- Used frequently elsewhere in AI and optimization
- In DL, used to train/fit/optimize network parameters



Goals for Today

(1) How do we find gradients of our neural network?

(1) Backprop (Backpropagation of Errors)

(2) How do we use those gradients to find good solutions?

(1) Stochastic Gradient Descent

Option 2: Gradient Descent

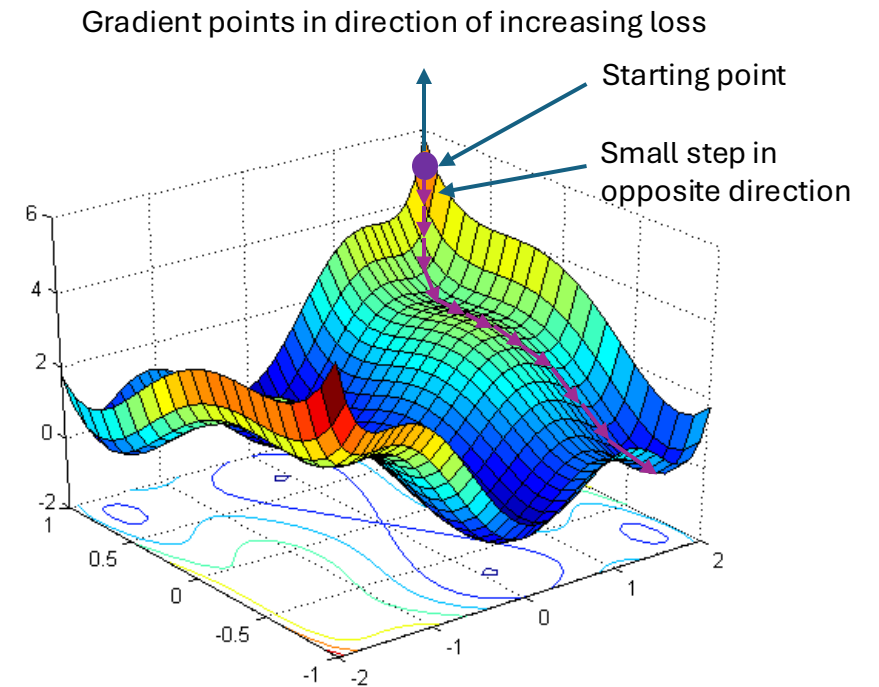
1. Start with some initial set of parameters
2. Take small step in the direction of the negative gradient
3. Repeat 2 until convergence

For N iterations or until $\Delta\theta < \epsilon$:

$$\vec{\theta} \leftarrow \theta - \alpha \nabla f_{\theta}$$



Learning Rate $\alpha \in [0,1]$



Option 2: Gradient Descent

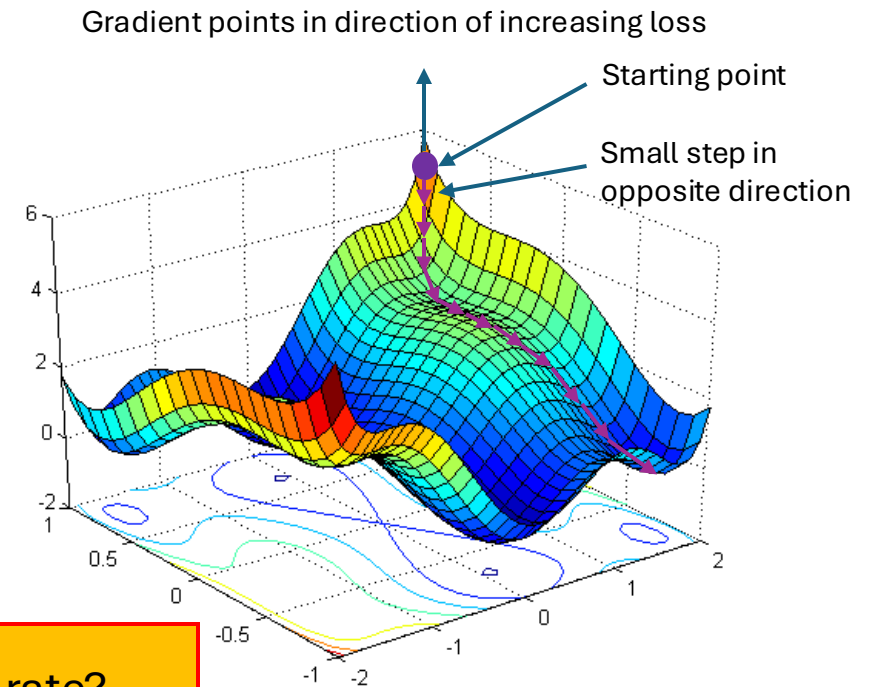
1. Start with some initial set of parameters
2. Take small step in the direction of the negative gradient
3. Repeat 2 until convergence

For N iterations or until $\Delta\theta < \epsilon$:

$$\vec{\theta} \leftarrow \theta - \alpha \nabla f_{\theta}$$

↑
Learning Rate $\alpha \in [0,1]$

Why do we need a learning rate?



Option 2: Gradient Descent

1. Start with some initial set of parameters
2. Take small step in the direction of the negative gradient
3. Repeat 2 until convergence

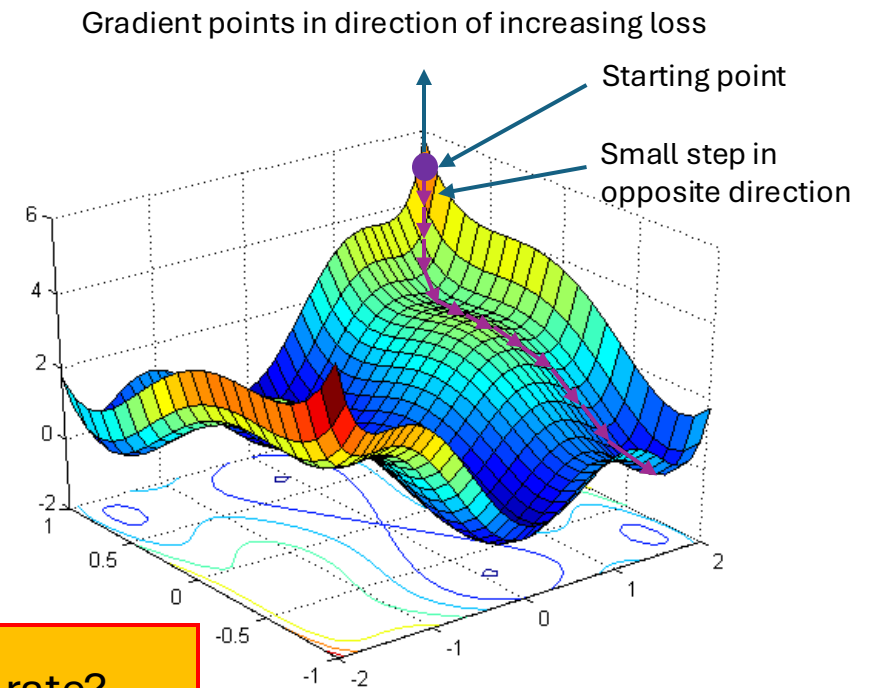
For N iterations or until $\Delta\theta < \epsilon$:

$$\vec{\theta} \leftarrow \theta - \alpha \nabla f_{\theta}$$

↑
Learning Rate $\alpha \in [0,1]$

Why do we need a learning rate?

Derivatives/Gradients only hold locally



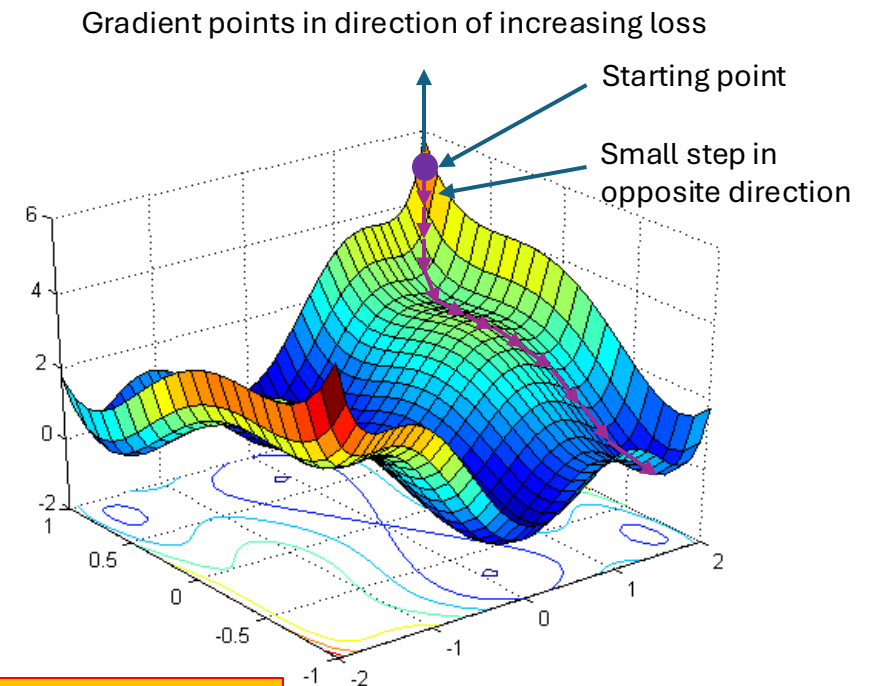
Option 2: Gradient Descent

1. Start with some initial set of parameters
2. Take small step in the direction of the negative gradient
3. Repeat 2 until convergence

For N iterations or until $\Delta\theta < \epsilon$:

$$\vec{\theta} \leftarrow \theta - \alpha \nabla f_{\theta}$$

Gradient Descent does not converge to the global minimum.
It can (and pretty much always does) get stuck in local minima.



Option 2: Gradient Descent

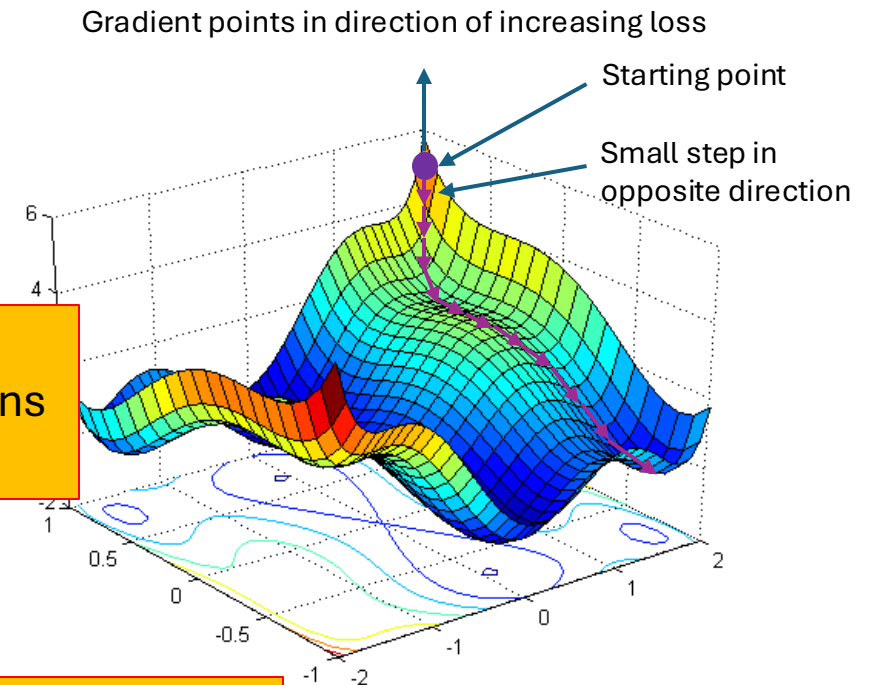
1. Start with some initial set of parameters
2. Take small step in the direction of the negative gradient
3. Repeat 2 until convergence

Understanding gradient descent is the single most important concept in all of Deep Learning. Most decisions in DL are made for reasons related to gradients.

For N iterations or until $\Delta\theta < \epsilon$:

$$\vec{\theta} \leftarrow \theta - \alpha \nabla f_{\theta}$$

Gradient Descent does not converge to the global minimum. It can (and pretty much always does) get stuck in local minima.

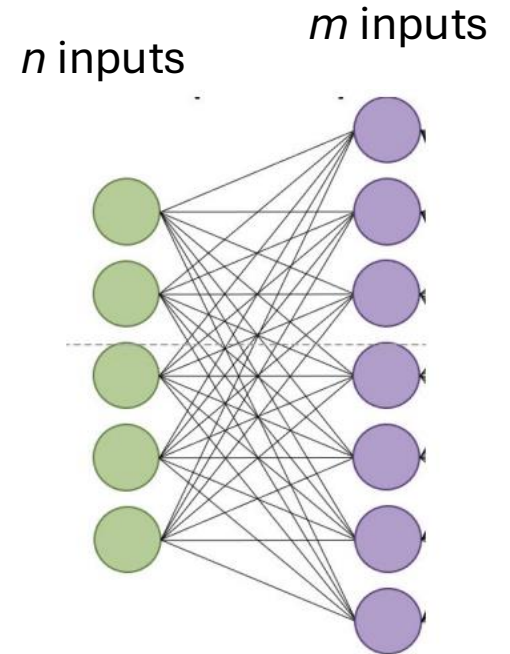


Gradients

Gradient descent needs gradients, how do we actually calculate them?

Weight Matrix for a Layer of Neurons

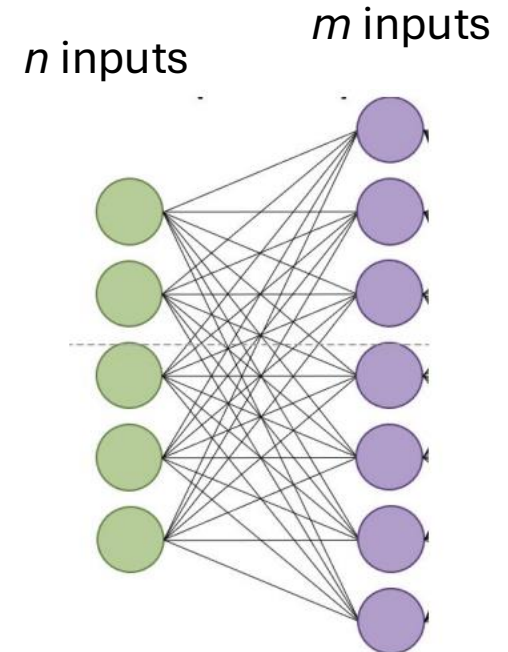
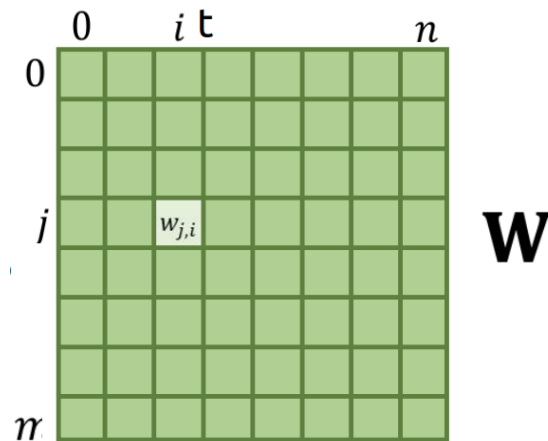
- We have an input of size n and we want an output vector of size m .
- We will represent our weights as a matrix.
 - What should the dimensions of our matrix be?



Weight Matrix

- We have an input of size n and we want an output vector of size m .
- We will represent our weights as a matrix.
 - What should the dimensions of our matrix be?

$w_{j,i}$ is the j^{th} row and the i^{th} column of our matrix, or the weight multiplied by the i^{th} index of the input which is used to create the j^{th} index in the output



Jacobians

- Gradients are for functions with multiple inputs and one output
- Hidden layers in our neural networks have multiple outputs
- The Jacobian matrix is the matrix of all partial derivatives

$$\begin{array}{l} f: \mathbb{R}^n \rightarrow \mathbb{R}^m \\ \text{Output Variables:} \\ [f_1, f_2, \dots, f_m] \end{array} \quad \begin{array}{l} \text{Input Variables:} \\ [x_1, x_2, \dots, x_n] \\ \left[\begin{array}{ccc} \frac{\partial f_1}{\partial x_1} & \dots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \dots & \frac{\partial f_m}{\partial x_n} \end{array} \right] \end{array}$$

Chain rule

If f and g are both differentiable and $F(x)$ is the composite function defined by $F(x) = f(g(x))$ then F is differentiable and F' is given by the product

$$F'(x) = f'(g(x)) g'(x)$$

Differentiate
outer function



Differentiate
inner function

Applying Chain rule [Example]

$$f(x) = x^2 \qquad g(x) = (2x^2 + 1)$$

$$F(x) = f(g(x))$$

$$F(x) = (2x^2 + 1)^2$$

Backprop Derivation

We will use the following notation:

$\mathbb{X} \in \mathbb{R}^{n \times d}$: Data matrix

$W^{(l)}$: Weights matrix for layer l

$z^{(l)}$: output of neurons at layer l before activation

$\sigma^{(l)}$: activation function of neurons at layer l

$a^{(l)}$: output of neurons at layer l after activation

\hat{y} : output of neural network

$L(y, \hat{y})$: Loss Function

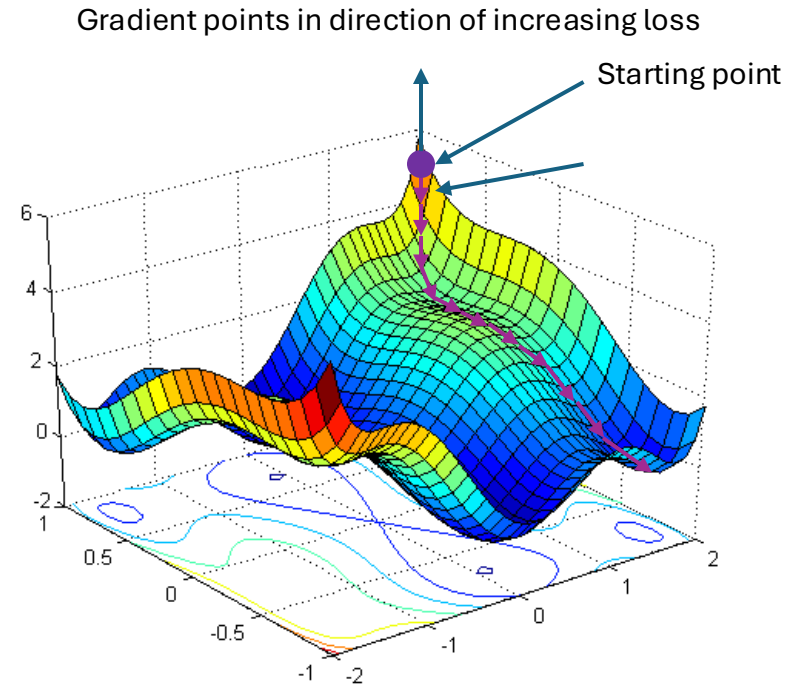
Backprop Derivation (Blackboard)

$$\hat{y} = \sigma^k (W^k \dots \sigma^2 (W^2 \sigma^1 (W^1 \mathbb{X})))$$

$$\text{Need: } \frac{dL}{dW^1}, \frac{dL}{dW^2}, \dots$$

Gradient Descent Revisited

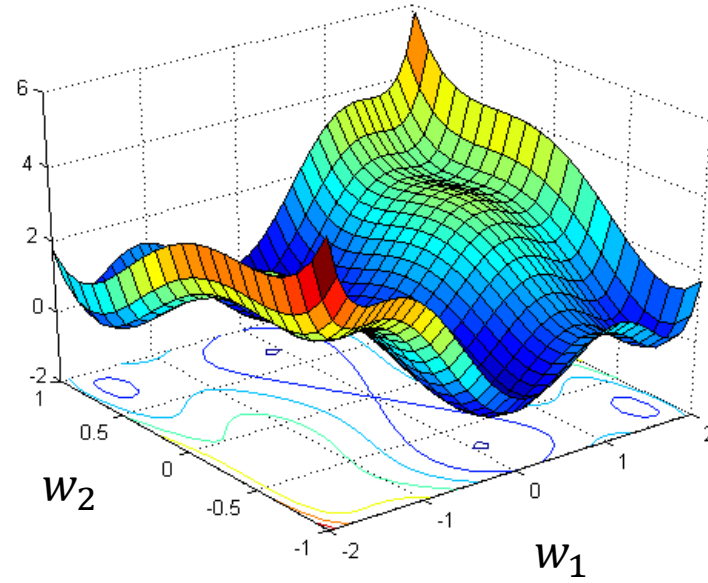
- We can calculate gradients efficiently, therefore we can run gradient descent
- But what about all of its issues...



Non-Convex Functions

MSE is **not** convex with respect to network parameters when non-linear activations are involved.

Multiple local minima



Saddle points

Local maxima

Stochastic Gradient Descent

- (full) Gradient Descent is slow, need to calculate gradient with many training examples
- Alternative: only use part of data at any given time (a **batch**)
 - Sample small batch from dataset, compute gradient only using the batch

For N iterations or until $\Delta\theta < \epsilon$:

For each *batch* $(\mathbb{X}_b, \mathbb{y}_b)$ in (\mathbb{X}, \mathbb{y})

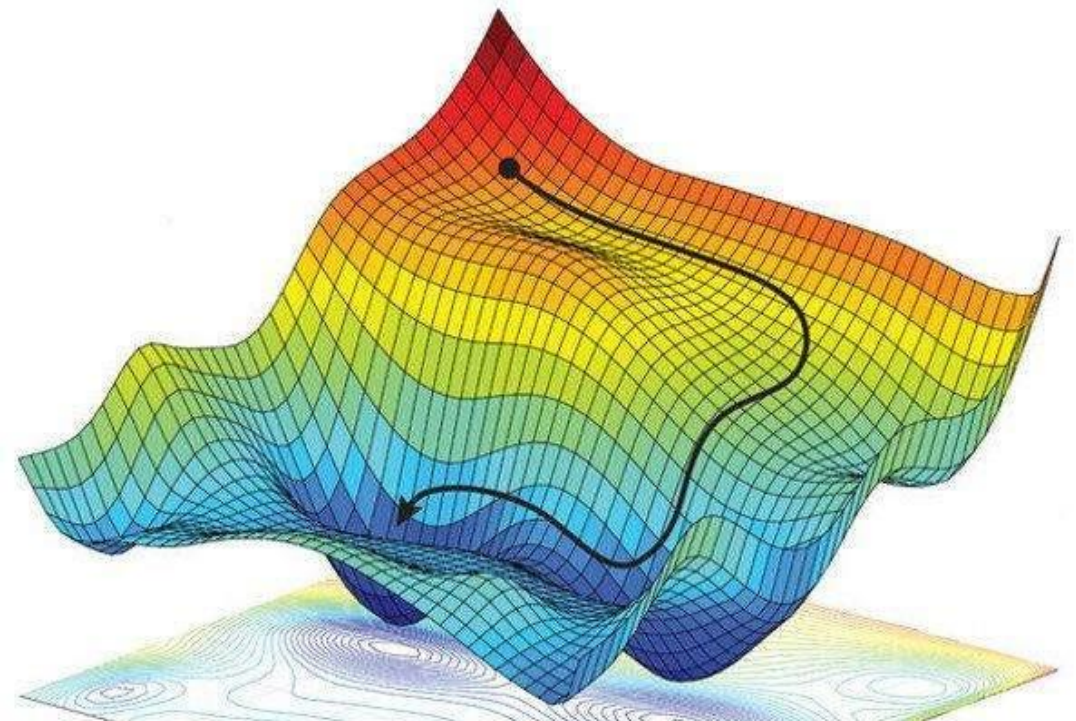
Compute Loss $L = \text{loss}_{fn}(f_{\theta}(\mathbb{X}_b), \mathbb{y}_b)$

$$\vec{\theta} \leftarrow \theta - \alpha \frac{dL}{d\theta}$$

← Compute with Backprop!

SGD

- SGD does not always take steps in the direction of the full gradient
 - Sometimes those steps make total loss go up (i.e., can escape local minima)



Stochastic Gradient Descent

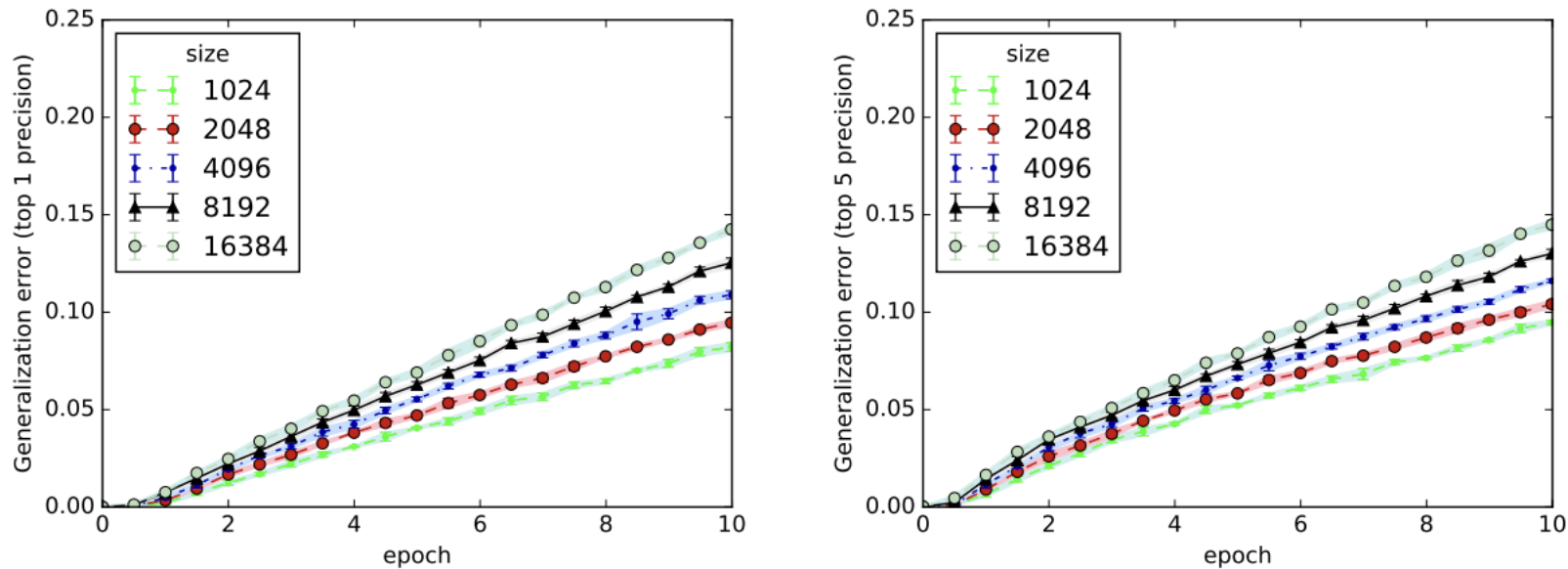


Figure 8: Left: Generalization error in terms of top 1 precision for varying model size on Imagenet. Right: The same with top 5 precision.

SGD generalizes better than vanilla Gradient Descent.
(Generalization is performance on test set)

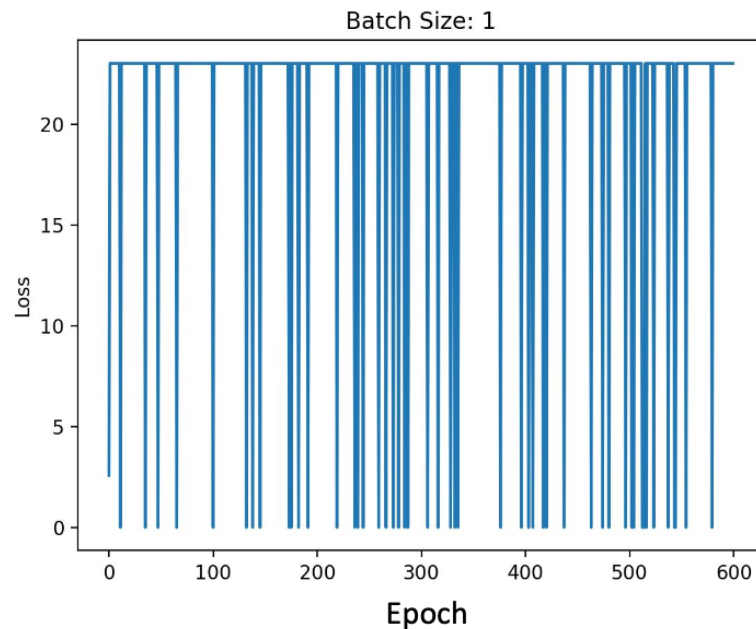
Train faster, generalize better: Stability of stochastic gradient descent. Hardt et al.

What size should the batch be?

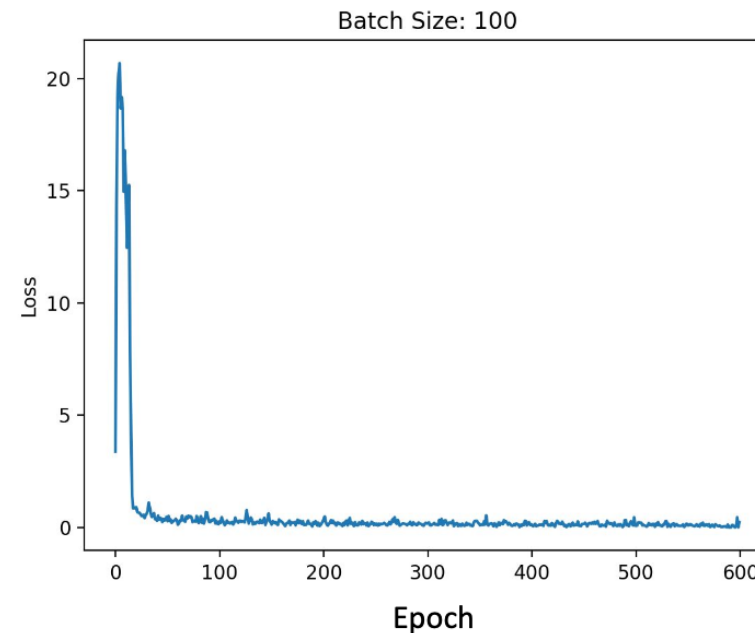
Any questions?



Small batch size:
Fast, jittery updates



Large batch size:
Slower, stable updates

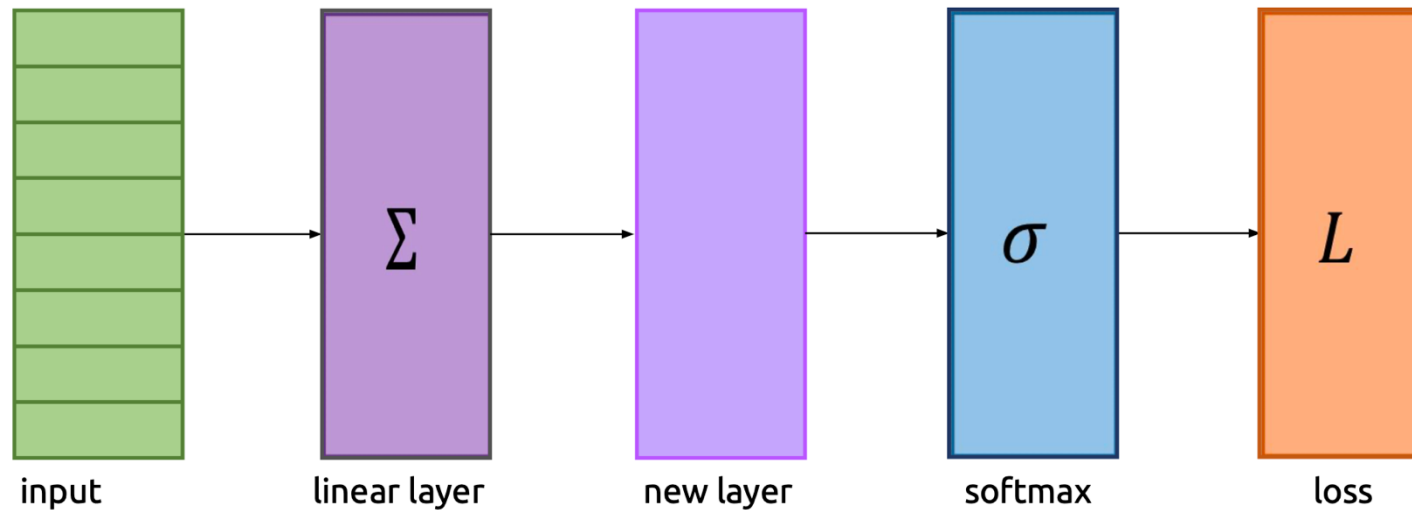


Generalizing Backpropagation

- What if we want to add another layer to our model?
- Calculating derivatives by hand *again* is a lot of work 😞

Can the computers do this for us?

Yes 😊



Computer-based Derivatives

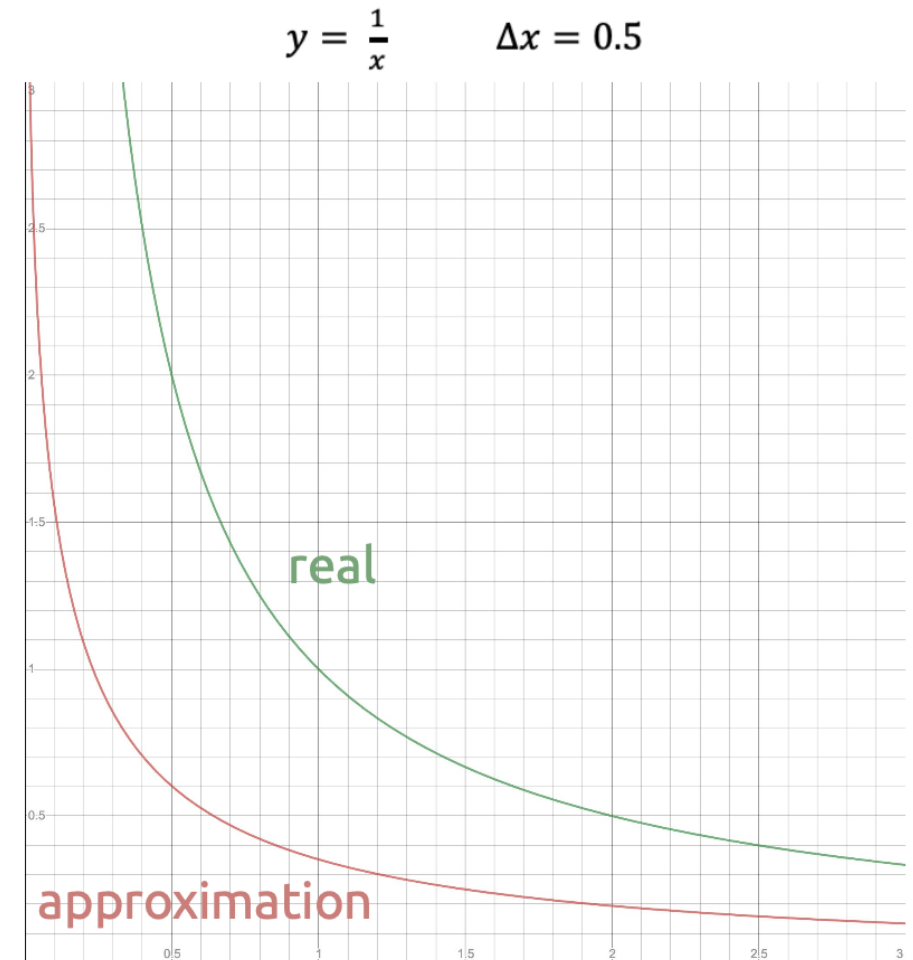
- **Numeric differentiation**

- $\frac{df}{dx} \approx \frac{f(x+\Delta x) - f(x)}{\Delta x}$
- Pick a small step size Δx
- Also called “finite differences”

Computer-based Derivatives

• Numeric differentiation

- $\frac{df}{dx} \approx \frac{f(x+\Delta x) - f(x)}{\Delta x}$
- Pick a small step size Δx
- Also called “finite differences”
- Easy to implement
- Arbitrarily inaccurate/unstable



Computer-based Derivatives

- Numeric differentiation
- **Symbolic differentiation**
 - Computer “does algebra” and simplifies expressions
 - What Wolfram Alpha does
<https://www.wolframalpha.com/>


$$d/dx (2x + 3x^2 + x(6 - 2))$$

 Extended Keyboard

 Upload

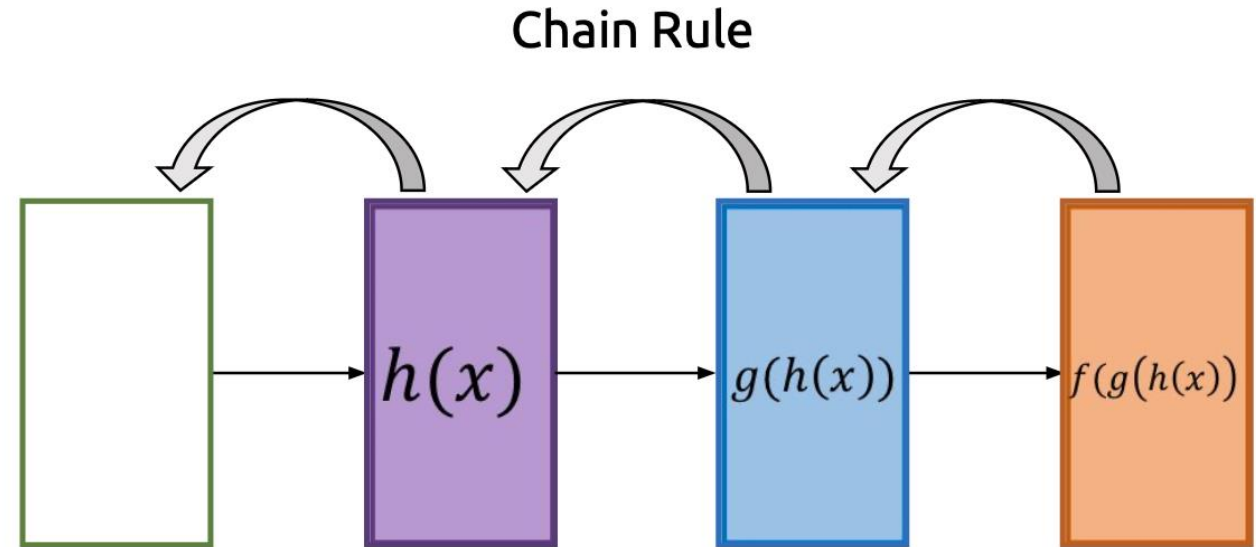
Derivative:

$$\frac{d}{dx} (2x + 3x^2 + x(6 - 2)) = 6(x + 1)$$

$$\frac{d}{dx} (6x + 3x^2)$$


Computer-based Derivatives

- Numeric differentiation
- Symbolic differentiation
- **Automatic differentiation**
 - Use the chain rule at runtime



Computer-based Derivatives

- Numeric differentiation
- Symbolic differentiation
- **Automatic differentiation**
 - Use the chain rule at runtime
 - Gives exact results
 - Handles dynamics (loops, etc.)
 - Easier to implement
 - Can't simplify expressions
- $\sin^2 x + \cos^2 x \Rightarrow 1$
- Automatic differentiation doesn't know this identity, will end up evaluating the entire expression on the left hand side

Two Main “Flavors” of Autodiff

- **Forward Mode Autodiff**

- Compute derivatives alongside the program as it is running

- **Reverse Mode Autodiff**

- Run the program, then compute derivatives (in reverse order)

Computer-based Derivatives

- Numeric differentiation
- **Symbolic differentiation**
 - Computer “does algebra” and simplifies expressions
 - What Wolfram Alpha does
 - Exact (no approximation error)
 - Complex to implement
 - Only handles static expressions (what about e.g. loops?)

- Example:

```
while abs(x) > 5:  
    x = x / 2
```

- This loop could run once or 100 times, it's impossible to know

Recap

Backprop is an efficient algorithm for computing gradients of neural networks

Stochastic Gradient Descent (SGD) is an optimization algorithm that finds good network parameters given a gradient

SGD uses batch learning updates for both **efficiency** reasons and because it produces **better solutions**