

DSC 140B

Representation Learning

Lecture 13 | Part 1

Backpropagation

Gradient of a Network

- ▶ To train a network, we need to compute the gradient of the risk.
- ▶ In the process, we compute the gradient $\nabla_{\vec{w}} H$.
 - ▶ That is, $\partial H / \partial W_{ij}^{(\ell)}$ and $\partial H / \partial b_i^{(\ell)}$ for all valid i, j, ℓ .
- ▶ Last time, we derived **backpropagation**: an algorithm for doing this efficiently.

Backpropagation

Given an input \vec{x} and a current parameter vector \vec{w} :

1. Evaluate the network to compute $z_j^{(\ell)}$ and $a_j^{(\ell)}$ for all nodes.
2. For each layer ℓ from last to first:

▶ Compute $\frac{\partial H}{\partial a_j^{(\ell)}} = \sum_{k=1}^{n_{\ell+1}} \frac{\partial H}{\partial z_k^{(\ell+1)}} W_{jk}^{(\ell+1)}$

▶ Compute $\frac{\partial H}{\partial z_j^{(\ell)}} = \frac{\partial H}{\partial a_j^{(\ell)}} g'(z_j^{(\ell)})$

▶ Compute $\frac{\partial H}{\partial W_{ij}^{(\ell)}} = \frac{\partial H}{\partial z_j^{(\ell)}} a_i^{(\ell-1)}$

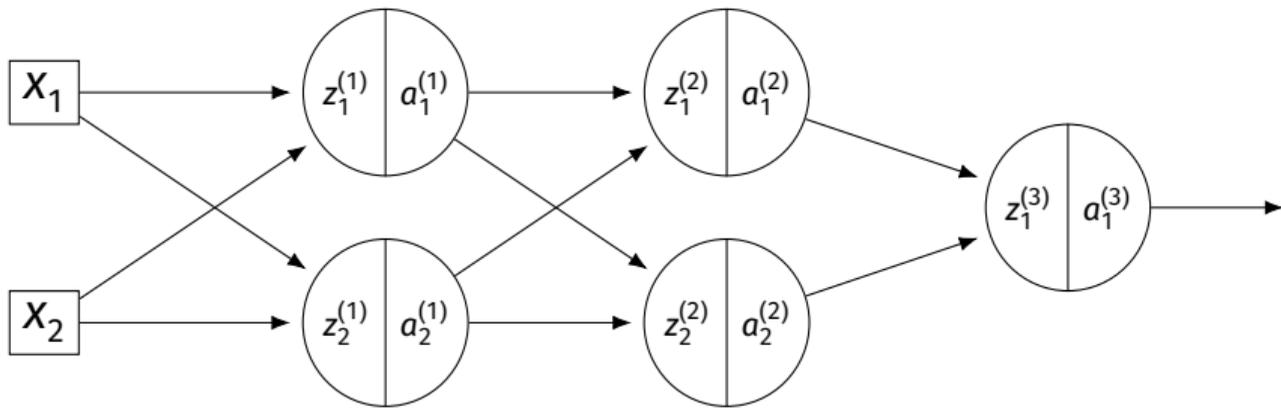
▶ Compute $\frac{\partial H}{\partial b_j^{(\ell)}} = \frac{\partial H}{\partial z_j^{(\ell)}}$

This computes $\nabla_{\vec{w}} H$ and evaluates it at \vec{x} and \vec{w} .

Example

Compute the entries of the gradient given:

$$W^{(1)} = \begin{pmatrix} 2 & -3 \\ 2 & 1 \end{pmatrix} \quad W^{(2)} = \begin{pmatrix} 2 & 1 \\ 0 & 1 \end{pmatrix} \quad W^{(3)} = \begin{pmatrix} 3 \\ -2 \end{pmatrix} \quad \vec{x} = (2, 1)^T \quad g(z) = \text{ReLU}$$

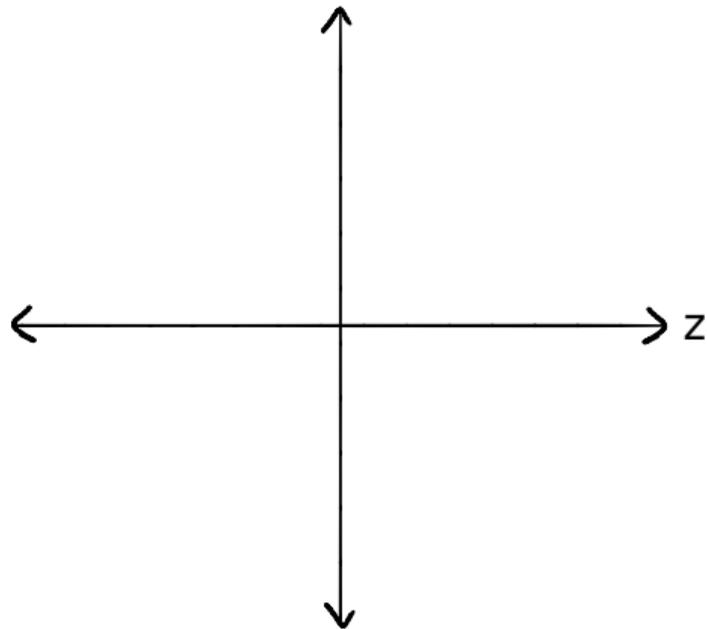


$$\frac{\partial H}{\partial a_j^{(\ell)}} = \sum_{k=1}^{n_{\ell+1}} \frac{\partial H}{\partial z_k^{(\ell+1)}} W_{jk}^{(\ell+1)} \quad \frac{\partial H}{\partial z_j^{(\ell)}} = \frac{\partial H}{\partial a_j^{(\ell)}} g'(z_j^{(\ell)}) \quad \frac{\partial H}{\partial W_{ij}^{(\ell)}} = \frac{\partial H}{\partial z_j^{(\ell)}} a_i^{(\ell-1)}$$

Aside: Derivative of ReLU

$$g(z) = \max\{0, z\}$$

$$g'(z) = \begin{cases} 0, & z < 0 \\ 1, & z > 0 \end{cases}$$

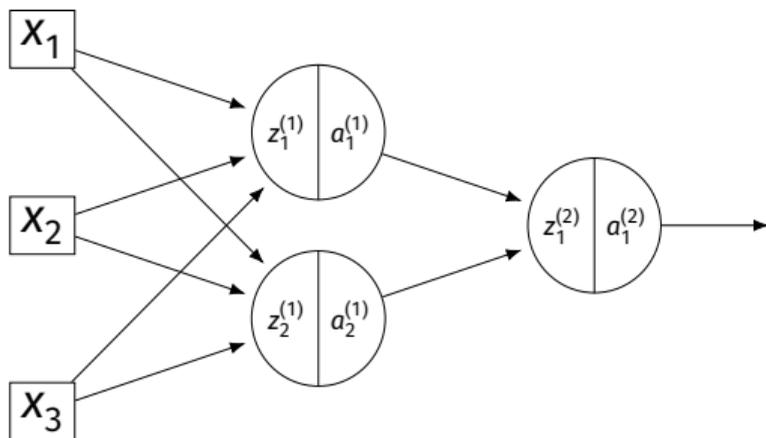


Exercise

What is $\partial H / \partial W_{11}^{(1)}(\vec{x}, \vec{w})$ for the network in the previous example?

Exercise

Suppose $W_{11}^{(1)} = -2, W_{21}^{(1)} = -5, W_{31}^{(1)} = 2$ and $\vec{x} = (3, 2, -2)^T$ and all biases are 0. ReLU activations are used. What is $\partial H / \partial W_{11}^{(1)}(\vec{x}, \vec{w})$?



Summary: Backprop

- ▶ **Backprop** is an algorithm for efficiently computing the gradient of a neural network
- ▶ It is not an algorithm **you** need to carry out by hand: your NN library can do it for you.

DSC 140B

Representation Learning

Lecture 13 | Part 2

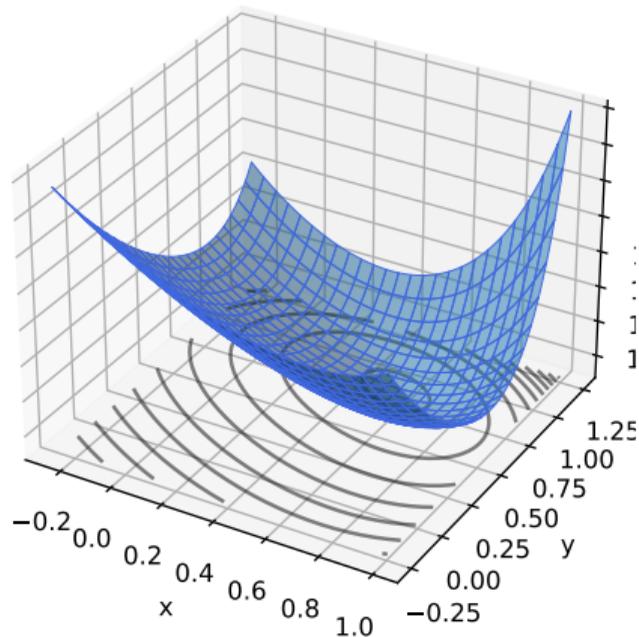
Gradient Descent for NN Training

Training Neural Networks

- ▶ We train NNs by minimizing **empirical risk**, $R(\vec{w})$.
- ▶ **Idea:** compute $\nabla_{\vec{w}}R(\vec{w})$; set to 0; solve for \vec{w} .
- ▶ **Problem:** for most NNs, the $R(\vec{w})$ is **complicated**.
 - ▶ I.e., we can't solve $\nabla_{\vec{w}}R(\vec{w}) = 0$ in closed form.
- ▶ Instead, we use iterative approach: **gradient descent**.

Example

- ▶ Consider $f(x, y) = e^{x^2+y^2} + (x - 2)^2 + (y - 3)^2$.



Example

- ▶ **Goal:** minimize f .
- ▶ **First idea:** solve $\vec{\nabla} f(x, y) = 0$.
- ▶ The gradient is:

$$\vec{\nabla} f(x, y) = \begin{pmatrix} 2xe^{x^2+y^2} + 2(x-2) \\ 2ye^{x^2+y^2} + 2(y-3) \end{pmatrix}$$

- ▶ Can we solve the system?

$$2xe^{x^2+y^2} + 2(x-2) = 0$$

$$2ye^{x^2+y^2} + 2(y-3) = 0$$

Example

- ▶ **Goal:** minimize f .
- ▶ **First idea:** solve $\vec{\nabla} f(x, y) = 0$.
- ▶ The gradient is:

$$\vec{\nabla} f(x, y) = \begin{pmatrix} 2xe^{x^2+y^2} + 2(x-2) \\ 2ye^{x^2+y^2} + 2(y-3) \end{pmatrix}$$

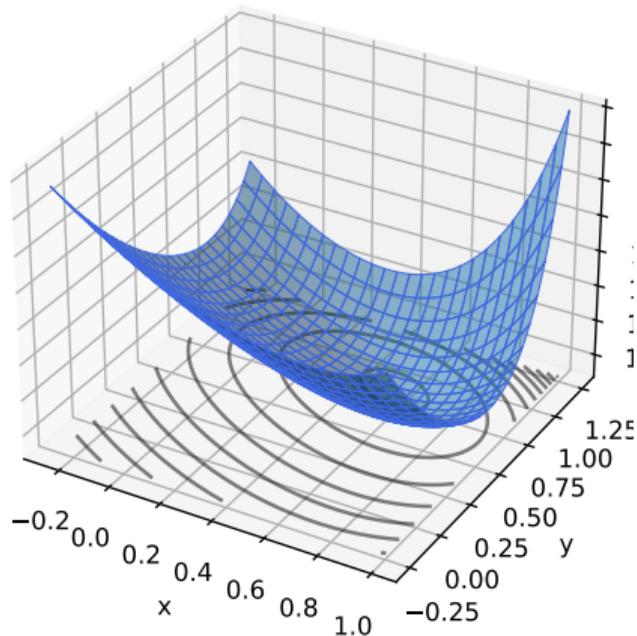
- ▶ Can we solve the system? **Not in closed form.**

$$2xe^{x^2+y^2} + 2(x-2) = 0$$

$$2ye^{x^2+y^2} + 2(y-3) = 0$$

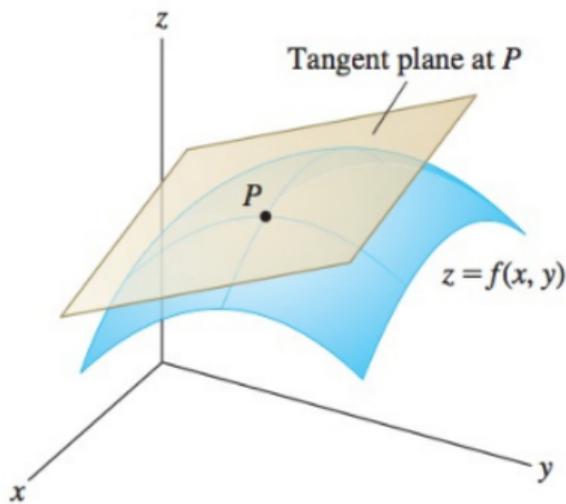
Idea

- ▶ Apply an **iterative** approach.
- ▶ Start at an arbitrary location.
- ▶ “Walk downhill”, towards minimum.



Which way is down?

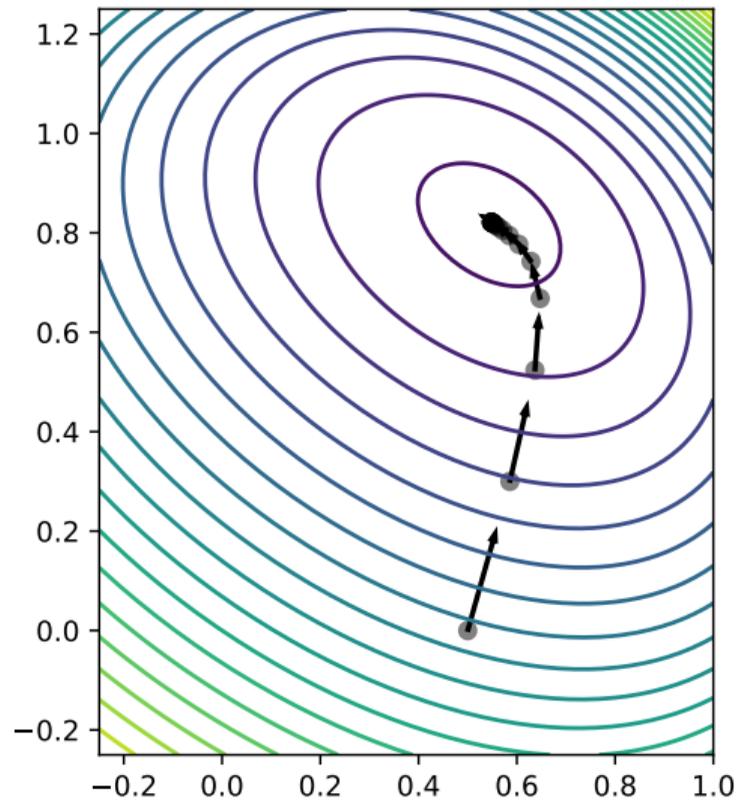
- ▶ **Claim:** $\vec{\nabla}f(\vec{x})$ points in direction of **steepest ascent**.
- ▶ It follows that $-\vec{\nabla}f(\vec{x})$ points in direction of **steepest descent**.
- ▶ $\|\vec{\nabla}f(\vec{x})\|$ measures the rate of change.



Gradient Descent

- ▶ Pick arbitrary starting point $\vec{x}^{(0)}$, **learning rate** parameter $\eta > 0$.
- ▶ Until convergence, repeat:
 - ▶ Compute gradient of f at $\vec{x}^{(i)}$; that is, compute $\vec{\nabla}f(\vec{x}^{(i)})$.
 - ▶ Update $\vec{x}^{(i+1)} = \vec{x}^{(i)} - \eta \vec{\nabla}f(\vec{x}^{(i)})$.
- ▶ When do we stop?
 - ▶ When difference between $\vec{x}^{(i)}$ and $\vec{x}^{(i+1)}$ is negligible.
 - ▶ I.e., when $\|\vec{x}^{(i)} - \vec{x}^{(i+1)}\|$ is small.

```
def gradient_descent(
    gradient, x, learning_rate=.01,
    threshold=.1e-4
):
    while True:
        x_new = x - learning_rate * gradient(x)
        if np.linalg.norm(x - x_new) < threshold:
            break
        x = x_new
    return x
```



Gradient Descent for NN Training

- ▶ Now let's use gradient descent to train a NN.

Empirical Risk Minimization

0. Collect a training set, $\{(\vec{x}^{(i)}, y_i)\}$
1. Pick the form of the prediction function, H .
 - ▶ E.g., a neural network, H .
2. Pick a loss function.
3. Minimize the empirical risk w.r.t. that loss.

In General

- ▶ Let ℓ be the loss function, let $H(\vec{x}; \vec{w})$ be the prediction function.
- ▶ The empirical risk is:

$$R(\vec{w}) = \frac{1}{n} \sum_{i=1}^n \ell(H(\vec{x}^{(i)}; \vec{w}), y_i)$$

- ▶ We want to minimize it using gradient descent.

Gradient of the Risk

$$\nabla_{\vec{w}} R(\vec{W}) = \nabla_{\vec{w}} \left(\frac{1}{n} \sum_{i=1}^n \ell(H(\vec{x}^{(i)}; \vec{W}), y_i) \right)$$

Gradient of the Risk

$$\begin{aligned}\nabla_{\vec{w}} R(\vec{w}) &= \nabla_{\vec{w}} \left(\frac{1}{n} \sum_{i=1}^n \ell(H(\vec{x}^{(i)}; \vec{w}), y_i) \right) \\ &= \frac{1}{n} \sum_{i=1}^n \nabla_{\vec{w}} \ell(H(\vec{x}^{(i)}; \vec{w}), y_i)\end{aligned}$$

Gradient of the Risk

$$\begin{aligned}\nabla_{\vec{w}} R(\vec{w}) &= \nabla_{\vec{w}} \left(\frac{1}{n} \sum_{i=1}^n \ell(H(\vec{x}^{(i)}; \vec{w}), y_i) \right) \\ &= \frac{1}{n} \sum_{i=1}^n \nabla_{\vec{w}} \ell(H(\vec{x}^{(i)}; \vec{w}), y_i)\end{aligned}$$

Using the chain rule...

Gradient of the Risk

$$\begin{aligned}\nabla_{\vec{w}} R(\vec{w}) &= \nabla_{\vec{w}} \left(\frac{1}{n} \sum_{i=1}^n \ell(H(\vec{x}^{(i)}; \vec{w}), y_i) \right) \\ &= \frac{1}{n} \sum_{i=1}^n \nabla_{\vec{w}} \ell(H(\vec{x}^{(i)}; \vec{w}), y_i) \\ &= \frac{1}{n} \sum_{i=1}^n \frac{\partial \ell}{\partial H}(\vec{x}^{(i)}, y_i, \vec{w}) \nabla_{\vec{w}} H(\vec{x}^{(i)}; \vec{w})\end{aligned}$$

Gradient of the Risk

- ▶ In general:

$$\nabla_{\vec{w}} R(\vec{w}) = \frac{1}{n} \sum_{i=1}^n \frac{\partial \ell}{\partial H}(\vec{x}^{(i)}, y_i, \vec{w}) \nabla_{\vec{w}} H(\vec{x}^{(i)}; \vec{w})$$

- ▶ When H is a neural network, we compute $\nabla_{\vec{w}} H$ using **backpropagation**.
- ▶ But to move forward, we need to pick loss, ℓ .

Example: Square Loss

- ▶ Recall, the square loss is:

$$\ell(H(\vec{x}^{(i)}; \vec{w}), y_i) = (H(\vec{x}^{(i)}; \vec{w}) - y_i)^2$$

- ▶ The partial derivative of the loss w.r.t. H is:

$$\begin{aligned} \frac{\partial \ell}{\partial H}(\vec{x}^{(i)}, y_i, \vec{w}) &= \frac{\partial}{\partial H} (H(\vec{x}^{(i)}; \vec{w}) - y_i)^2 \\ &= 2(H(\vec{x}^{(i)}; \vec{w}) - y_i) \end{aligned}$$

Example: Square Loss

- ▶ If we use square loss, the risk is the **mean squared error**.
- ▶ The gradient of the risk is:

$$\nabla_{\vec{w}} R(\vec{w}) = \frac{2}{n} \sum_{i=1}^n (H(\vec{x}^{(i)}; \vec{w}) - y_i) \nabla_{\vec{w}} H(\vec{x}^{(i)}; \vec{w})$$

- ▶ We compute it using backpropagation.

Gradient Descent for NNs

- ▶ Pick starting weights $\vec{w}^{(0)}$, learning rate parameter $\eta > 0$, loss function ℓ .
- ▶ Until convergence, repeat:
 - ▶ Compute $\nabla_{\vec{w}} R(\vec{w}^{(0)})$ using backpropagation.
 - ▶ Update $\vec{w}^{(i+1)} = \vec{w}^{(i)} - \eta \nabla_{\vec{w}} R(\vec{w}^{(i)})$.
- ▶ When do we stop?
 - ▶ When difference between $\vec{w}^{(i)}$ and $\vec{w}^{(i+1)}$ is negligible.
 - ▶ I.e., when $\|\vec{w}^{(i)} - \vec{w}^{(i+1)}\|$ is small.

Interpretation

- ▶ We use **backpropagation** to compute $\nabla_{\vec{w}} H$.
 - ▶ And therefore to compute $\nabla_{\vec{w}} R$.
- ▶ Recall: entry i of $\nabla_{\vec{w}} H$ measures how much H changes when we change w_i .
 - ▶ I.e., how much “influence” w_i has on H .
- ▶ Intuition: if prediction is wrong, weights with greatest influence should be changed the most.

Interpretation

- ▶ This is essentially what backpropagation does.

$$\nabla_{\vec{w}} R(\vec{w}) = \frac{2}{n} \sum_{i=1}^n \underbrace{(H(\vec{x}^{(i)}) - y_i)}_{\text{Error}} \underbrace{\nabla_{\vec{w}} H(\vec{x}^{(i)}; \vec{w})}_{\text{Blame}}$$

- ▶ Backprop propagates errors “backwards” through the network, assigning blame to each weight.

DSC 140B

Representation Learning

Lecture 13 | Part 3

PyTorch

Training NNs in Python

- ▶ Python is **the** language of ML.
- ▶ There are several popular NN libraries for Python, but nowadays **PyTorch** is the most popular.

Demo Notebook

- ▶ A demo notebook is linked at dsc140b.com