# DSC 190

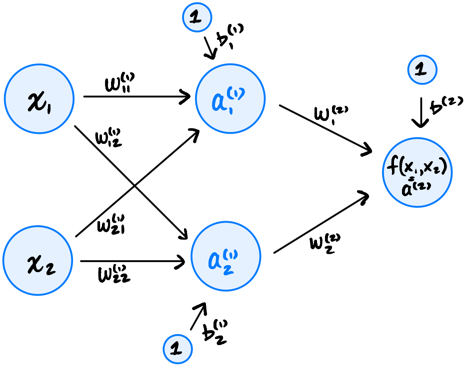
Machine Learning: Representations

Lecture 13 | Part 1

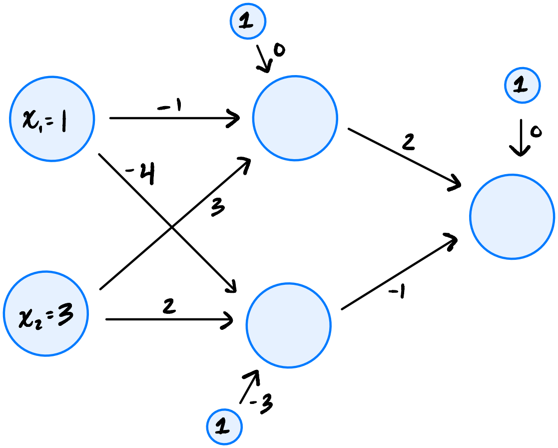
## Convexity in 1-d

# Neural Networks

A NN is just a function: $f(\vec{x};\vec{w})$



# Example



# Learning

- **Given**: a data set $(\vec{x}^{(i)}, y_i)$
- **Find**: weights $\vec{w}$ minimizing some cost function (e.g., expected square loss):

$$C(\vec{w}) = \frac{1}{n} \sum_{i=1}^{n} \left( f(\vec{x}^{(i)}; \vec{w}) - y_i \right)^2$$

- **Problem**: there is no closed-form solution

# Gradient Descent

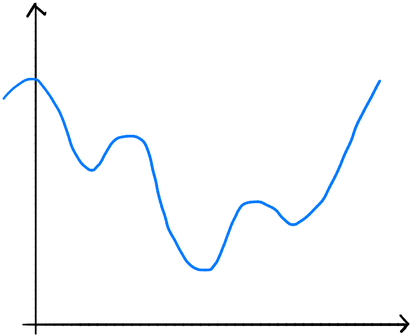
- **Idea**: start at arbitrary $\vec{w}^{(0)}$, walk in direction of gradient:

$$\nabla C = \begin{pmatrix} \frac{\partial C}{\partial w_0} \\ \frac{\partial C}{\partial w_1} \\ \vdots \\ \frac{\partial C}{\partial w_k} \end{pmatrix}$$

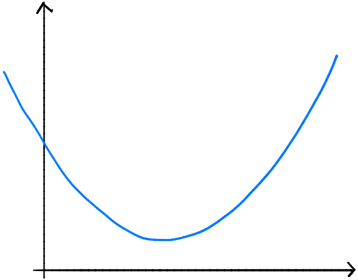
# Question

When is gradient descent guaranteed to work?

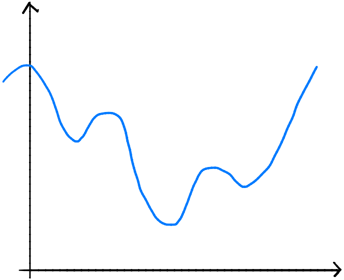
# Not here...



# Convex Functions



Convex



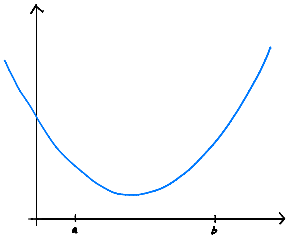
Non-convex

# Convexity: Definition

- $f$ is **convex** if for **every** $a, b$ the line segment between

$$(a, f(a)) \qquad \text{and} \qquad (b, f(b))$$

does not go below the plot of $f$.

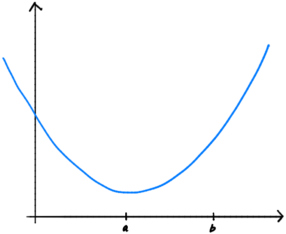


# Convexity: Definition

- $f$ is **convex** if for **every** $a, b$ the line segment between

$$(a, f(a)) \qquad \text{and} \qquad (b, f(b))$$

does not go below the plot of $f$.

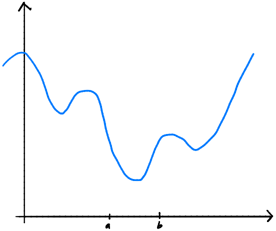


# Convexity: Definition

- $f$ is **convex** if for **every** $a, b$ the line segment between

$$(a, f(a)) \qquad \text{and} \qquad (b, f(b))$$

does not go below the plot of $f$.

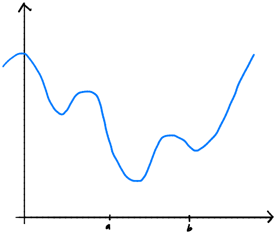


# Convexity: Definition

- $f$ is **convex** if for **every** $a, b$ the line segment between

$$(a, f(a)) \qquad \text{and} \qquad (b, f(b))$$

does not go below the plot of $f$.



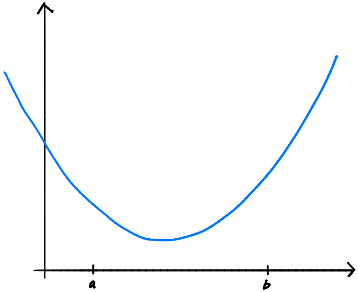
# Other Terms

- If a function is not convex, it is **non-convex**.
- **Strictly convex**: the line lies strictly above curve.
- **Concave**: the line lines on or below curve.

# Convexity: Formal Definition

- A function $f : \mathbb{R} \to \mathbb{R}$ is **convex** if for every choice of $a, b \in \mathbb{R}$ and $t \in [0, 1]$:

$$(1 - t)f(a) + tf(b) \geq f((1 - t)a + tb).$$

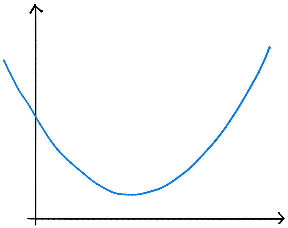


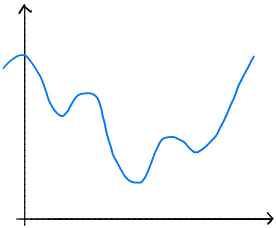
# Example

Is $f(x) = |x|$ convex?

# Another View: Second Derivatives

- If $\frac{d^2 f}{dx^2}(x) \geq 0$ for all $x$, then $f$ is convex.
- Example: $f(x) = x^4$ is convex.
- **Warning!** Only works if $f$ is twice differentiable!



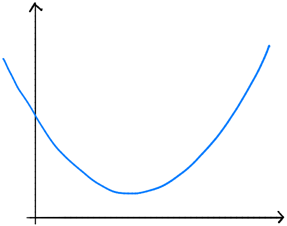


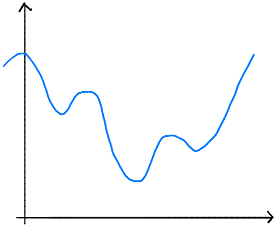
# Another View: Second Derivatives

- "Best" straight line at $x_0$:
  - $h_1(z) = f'(x_0) \cdot z + b$
- "Best" parabola at $x_0$:
  - At $x_0$, $f$ looks likes $h_2(z) = \frac{1}{2} f''(x_0) \cdot z^2 + f'(x_0) z + c$
  - Possibilities: upward-facing, downward-facing.

# Convexity and Parabolas

- Convex if for **every** $x_0$, parabola is upward-facing.
  - That is, $f''(x_0) \geq 0$.





# Convexity and Gradient Descent

- Convex functions are (relatively) easy to optimize.
- **Theorem**: if $R(x)$ is convex and differentiable[1][2] then gradient descent converges to a **global optimum** of $R$ *provided* that the step size is small enough[3].

---

[1] and its derivative is not too wild

[2] actually, a modified GD works on non-differentiable functions

[3] step size related to steepness.

# Nonconvexity and Gradient Descent

- Nonconvex functions are (relatively) hard to optimize.
- Gradient descent can still be useful.
- But not guaranteed to converge to a global minimum.

# DSC 190

## Machine Learning: Representations

Lecture 13 | Part 2

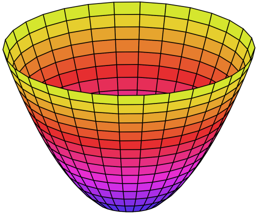
# Convexity in Many Dimensions

# Convexity: Definition

- $f(\vec{x})$ is **convex** if for **every** $\vec{a}, \vec{b}$ the line segment between

$$(\vec{a}, f(\vec{a})) \qquad \text{and} \qquad (\vec{b}, f(\vec{b}))$$

does not go below the plot of $f$.



# Convexity: Formal Definition

- A function $f : \mathbb{R}^d \to \mathbb{R}$ is **convex** if for every choice of $\vec{a}, \vec{b} \in \mathbb{R}^d$ and $t \in [0, 1]$:

$$(1 - t)f(\vec{a}) + tf(\vec{b}) \geq f((1 - t)\vec{a} + t\vec{b}).$$

# The Second Derivative Test

- For 1-d functions, convex if second derivative $\geq 0$.
- For 2-d functions, convex if ???

# The Hessian Matrix

- Create the **Hessian** matrix of second derivatives:

$$H(\vec{x}) = \begin{pmatrix} \frac{\partial f^2}{\partial x_1^2}(\vec{x}) & \frac{\partial f^2}{\partial x_1 x_2}(\vec{x}) \\ \frac{\partial f^2}{\partial x_2 x_1}(\vec{x}) & \frac{\partial f^2}{\partial x_2^2}(\vec{x}) \end{pmatrix}$$

# In General

- If $f : \mathbb{R}^d \to \mathbb{R}$, the **Hessian** at $\vec{x}$ is:

$$H(\vec{x}) = \begin{pmatrix} \frac{\partial f^2}{\partial x_1^2}(\vec{x}) & \frac{\partial f^2}{\partial x_1 x_2}(\vec{x}) & \cdots & \frac{\partial f^2}{\partial x_1 x_d}(\vec{x}) \\ \frac{\partial f^2}{\partial x_2 x_1}(\vec{x}) & \frac{\partial f^2}{\partial x_2^2}(\vec{x}) & \cdots & \frac{\partial f^2}{\partial x_2 x_d}(\vec{x}) \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial f^2}{\partial x_d x_1}(\vec{x}) & \frac{\partial f^2}{\partial x_d^2}(\vec{x}) & \cdots & \frac{\partial f^2}{\partial x_d^2}(\vec{x}) \end{pmatrix}$$

# The Second Derivative Test

- A function $f : \mathbb{R}^d \to \mathbb{R}$ is **convex** if for any $\vec{x} \in \mathbb{R}^d$, the Hessian matrix $H(\vec{x})$ is **positive semi-definite**.
- That is, all eigenvalues are $\geq 0$

# DSC 190

## Machine Learning: Representations

Lecture 13 | Part 3

# Basic Backpropagation

# Learning

- **Given**: a data set $(\vec{x}^{(i)}, y_i)$
- **Find**: weights $\vec{w}$ minimizing some cost function (e.g., expected square loss):

$$C(\vec{w}) = \frac{1}{n}\sum_{i=1}^{n}\left(f(\vec{x}^{(i)}; \vec{w}) - y_i\right)^2$$

- **Problem**: there is no closed-form solution

# Gradient Descent

- **Idea**: start at arbitrary $\vec{w}^{(0)}$, walk in direction of gradient:

$$\nabla C = \begin{pmatrix} \frac{\partial C}{\partial w_0} \\ \frac{\partial C}{\partial w_1} \\ \vdots \\ \frac{\partial C}{\partial w_k} \end{pmatrix}$$

# Computing the Gradient

- To train a neural network, we can use gradient descent.
- Involves computing the gradient of the cost function.
- **Backpropagation** is one method for efficiently computing the gradient.

# The Gradient

$$\begin{aligned}
\nabla_{\vec{w}} C(\vec{w}) &= \nabla_{\vec{w}} \frac{1}{n} \sum_{i=1}^{n} \left( f(\vec{x}^{(i)}; \vec{w}) - y_i \right)^2 \\
&= \frac{1}{n} \sum_{i=1}^{n} \nabla_{\vec{w}} \left( f(\vec{x}^{(i)}; \vec{w}) - y_i \right)^2 \\
&= \frac{1}{n} \sum_{i=1}^{n} 2 \left( f(\vec{x}^{(i)}; \vec{w}) - y_i \right) \nabla_{\vec{w}} f(\vec{x}^{(i)}; \vec{w})
\end{aligned}$$

# Interpreting the Gradient

$$\nabla_{\vec{w}} C(\vec{w}) = \frac{1}{n} \sum_{i=1}^{n} 2 \left( f(\vec{x}^{(i)}; \vec{w}) - y_i \right) \nabla_{\vec{w}} f(\vec{x}^{(i)}; \vec{w})$$

- The gradient has one term for each training example, $(\vec{x}^{(i)}, y_i)$
- If prediction for $\vec{x}^{(i)}$ is good, contribution to gradient is small.
- $\nabla_{\vec{w}} f(\vec{x}^{(i)}; \vec{w})$ captures how sensitive $f(\vec{x}^{(i)})$ is to value of each parameter.

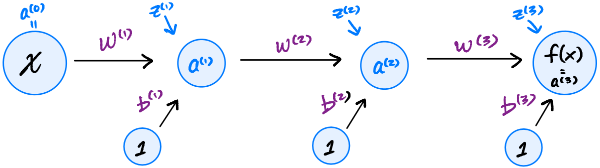
# The Chain Rule

- Recall the **chain rule** from calculus.
- Let $f, g : \mathbb{R} \to \mathbb{R}$
- Then:
$$\frac{d}{dx} f(g(x)) = f'(g(x)) \cdot g(x)$$
- Alternative notation: $\frac{d}{dx} f(g(x)) = \frac{df}{dg} \frac{dg}{dx}(x)$

# Example

- $f(x) = x^2$; $g(x) = 2x + 1$

# The Chain Rule for NNs

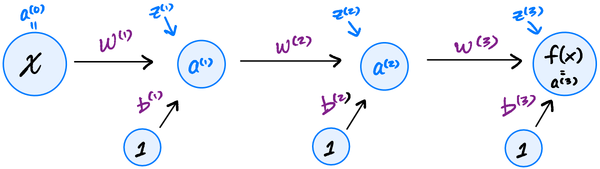



$$z^{(\ell)} = w^{(\ell)} a^{(\ell-1)} + b^{(\ell)}$$

$$a^{(\ell)} = \sigma\left(z^{(\ell)}\right)$$

$$\vec{w} = \left(w^{(1)}, w^{(2)}, w^{(3)}, b^{(1)}, b^{(2)}, b^{(3)}\right)$$

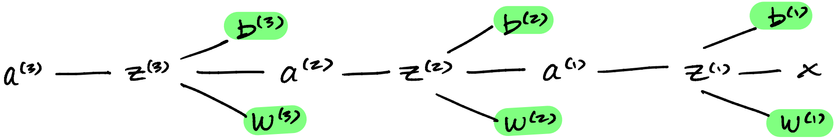
# Computation Graphs



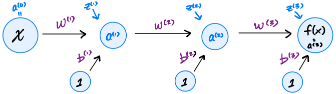

$$z^{(\ell)} = W^{(\ell)} a^{(\ell-1)} + b^{(\ell)}$$

$$a^{(\ell)} = \sigma(z^{(\ell)})$$

$$\vec{w} = \left(W^{(1)}, W^{(2)}, W^{(3)}, b^{(1)}, b^{(2)}, b^{(3)}\right)$$



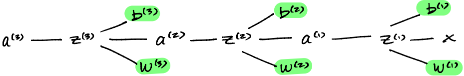
# Example




$z^{(l)} = w^{(l)} a^{(l-1)} + b^{(l)}$

$a^{(l)} = \sigma(z^{(l)})$

$\vec{w} = (w^{(1)}, w^{(2)}, w^{(3)}, b^{(1)}, b^{(2)}, b^{(3)})$

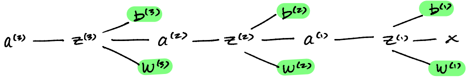


# General Formulas

- Derivatives are defined recursively
- Easy to compute derivatives for early layers if we have derivatives for later layers.
- This is **backpropagation**.

$$\frac{\partial f}{\partial w^{(\ell)}} = \frac{\partial f}{\partial a^{(\ell)}} \cdot \frac{\partial a^{(\ell)}}{\partial z^{(\ell)}} \cdot \frac{\partial z^{(\ell)}}{\partial w^{(\ell)}}$$

$$\frac{\partial f}{\partial a^{(\ell)}} = \frac{\partial f}{\partial a^{(\ell+1)}} \cdot \frac{\partial a^{(\ell+1)}}{\partial z^{(\ell+1)}} \cdot \frac{\partial z^{(\ell+1)}}{\partial a^{(\ell)}}$$

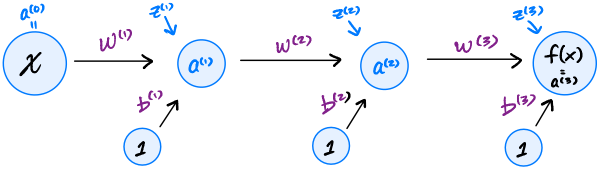


# Warning

- The derivatives depend on the network **architecture**
  - Number of hidden nodes / layers
- Backprop is done automatically by your NN library

# Backpropagation

Compute the derivatives for the last layers first; use them to compute derivatives for earlier layers.




$$z^{(l)} = w^{(l)} a^{(l-1)} + b^{(l)}$$

$$a^{(l)} = \sigma(z^{(l)})$$

$$\vec{w} = (w^{(1)}, w^{(2)}, w^{(3)}, b^{(1)}, b^{(2)}, b^{(3)})$$

# DSC 190

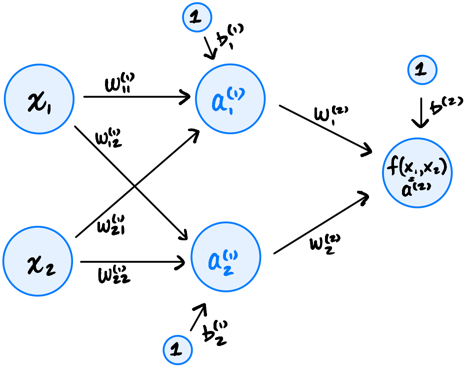
## Machine Learning: Representations

Lecture 13 | Part 4

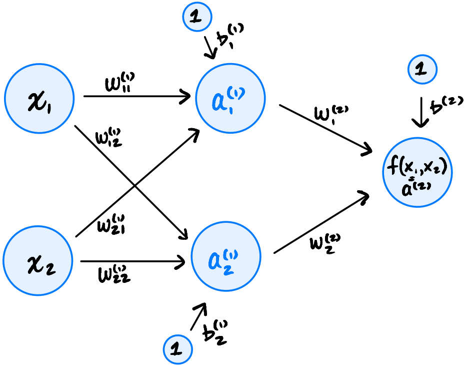
## A More Complex Example

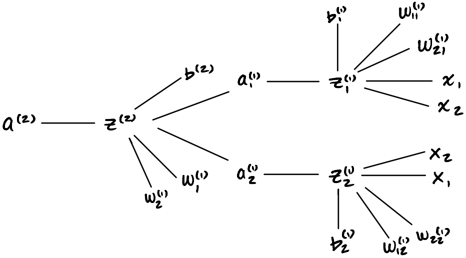
# Complexity

- The strategy doesn't change much when each layer has more nodes.



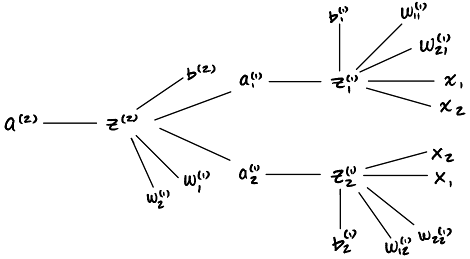
# Computational Graph





# Example

# General Formulas



$$\frac{\partial f}{\partial w_{ij}^{(\ell)}} = \frac{\partial f}{\partial a^{(\ell)}} \cdot \frac{\partial a^{(\ell)}}{\partial z^{(\ell)}} \cdot \frac{\partial z^{(\ell)}}{\partial w_{ij}^{(\ell)}}$$

$$\frac{\partial f}{\partial a^{(\ell)}} = \frac{\partial f}{\partial a^{(\ell+1)}} \cdot \frac{\partial a^{(\ell+1)}}{\partial z^{(\ell+1)}} \cdot \frac{\partial z^{(\ell+1)}}{\partial a^{(\ell)}}$$

# DSC 190

## Machine Learning: Representations

Lecture 13 | Part 5

## Intuition Behind Backprop

# Intuition