Intro to Deep Learning

Lecture 5: Backprop and Optimization

Midterm

- Time: Feb. 9 Monday 9:30 - 10:50 am (the usual lecture time)
- Location: Materials Sci and Engineering Room 103 (the usual lecture location)
- Format: 30 multi-choice questions (potentially have multiple correct answers)
- One A4 cheat sheet allowed (hand written!)

- HW1 due this Friday
- HW2 released

Midterm

- Question format:

Problem 1. Which of the following statements regarding course EE/CS 228 are correct?

(A) Lecture time is Tu/Th 8:00 - 9:20 am — MW 9:30 - 10:50

(B) Lecture location is Sproul Hall Room 1340 — Olmsted Room 1212

(C) Midterm is on Nov. 9th Thursday (in class) — Feb 9 in class

(D) EE/CS 228 is a great course

Select all correct answers
Midterm covers everything lectured up to Feb 9th

Recall Stochastic Gradient Descent

1. Pick example (x, y)
2. Pick ML model $f_w(\,.\,)$ with weights w
3. Pick loss function ℓ to minimize (e.g. squared)
4. Optimization becomes:

$$\mathscr{L}(w) = l\left(y, f_w(x)\right)$$

5. Gradient via chain rule

$$\nabla \mathscr{L}(w) = \ell'(y, f_w(x)) \times \nabla f_w(x)$$

Derivative of Loss

Gradient of Model

HOW?

Loss Functions

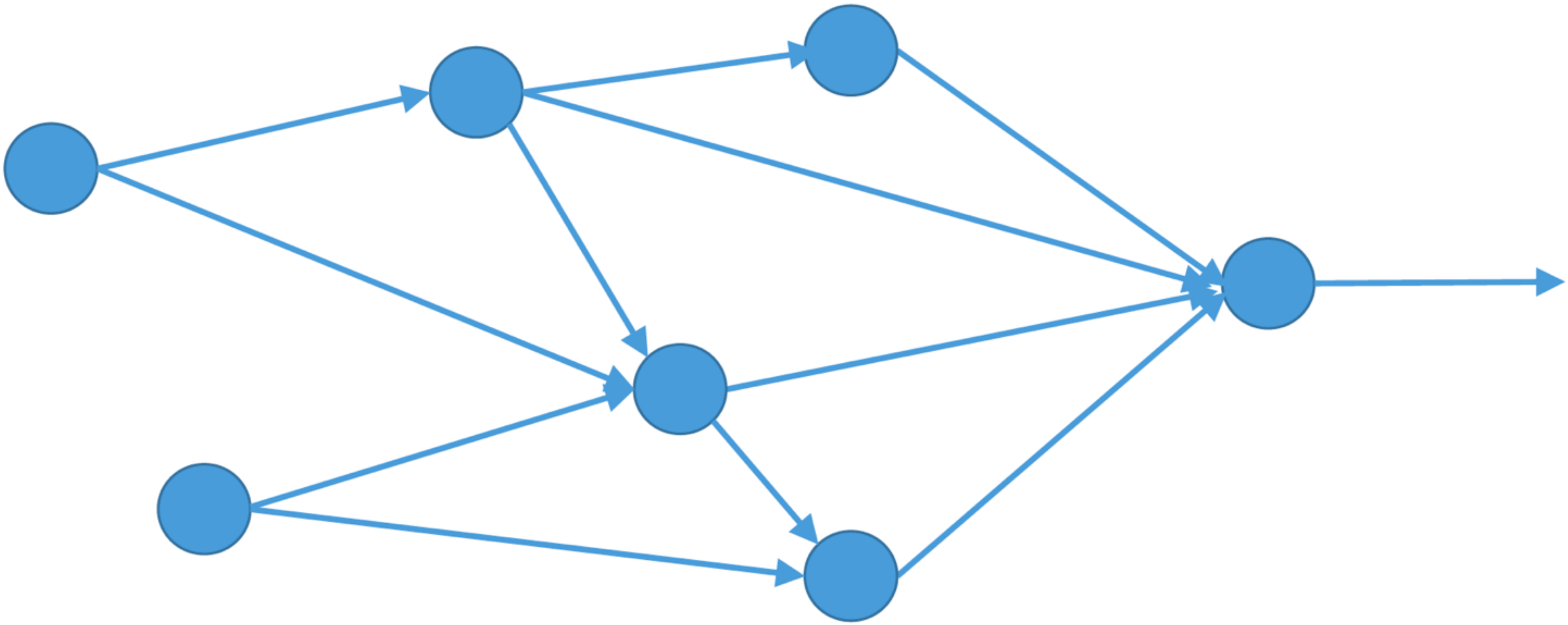
- Square loss: $\ell(y, f(x)) = \frac{1}{2}\big(f(x) - y\big)^2$

$$\ell'(y, f(x)) = \big(f(x) - y\big)$$

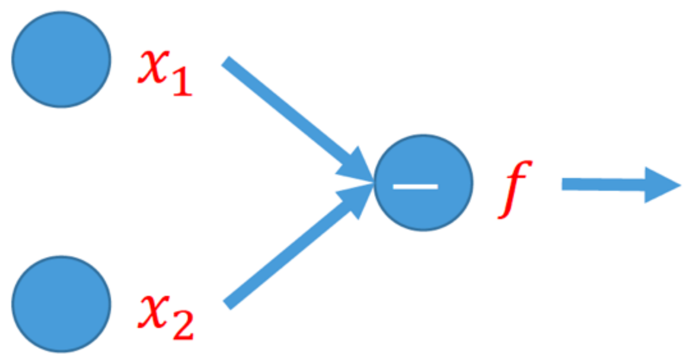
- Cross entropy loss

$$\ell\big(y, f(x)\big) = -\sum_{i \in [k]} y_i \log(p_i), \quad \text{where} \quad p_i = \frac{e^{f_i(x)}}{\sum_{j=1}^{n} e^{f_j(x)}}$$

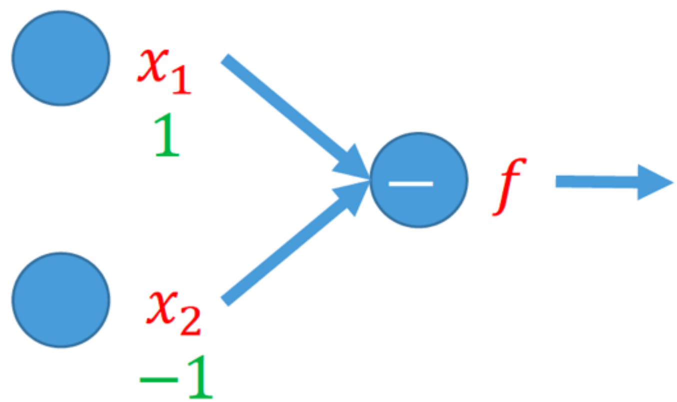
How to differentiate a neural net?



Answer: *Backpropagate*

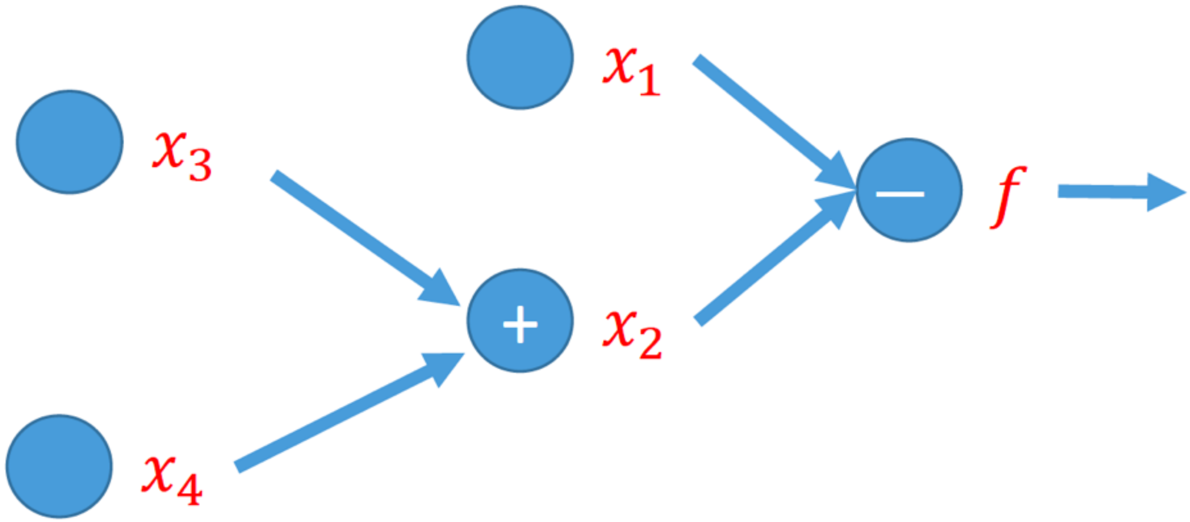


Function: $f = x_1 - x_2$

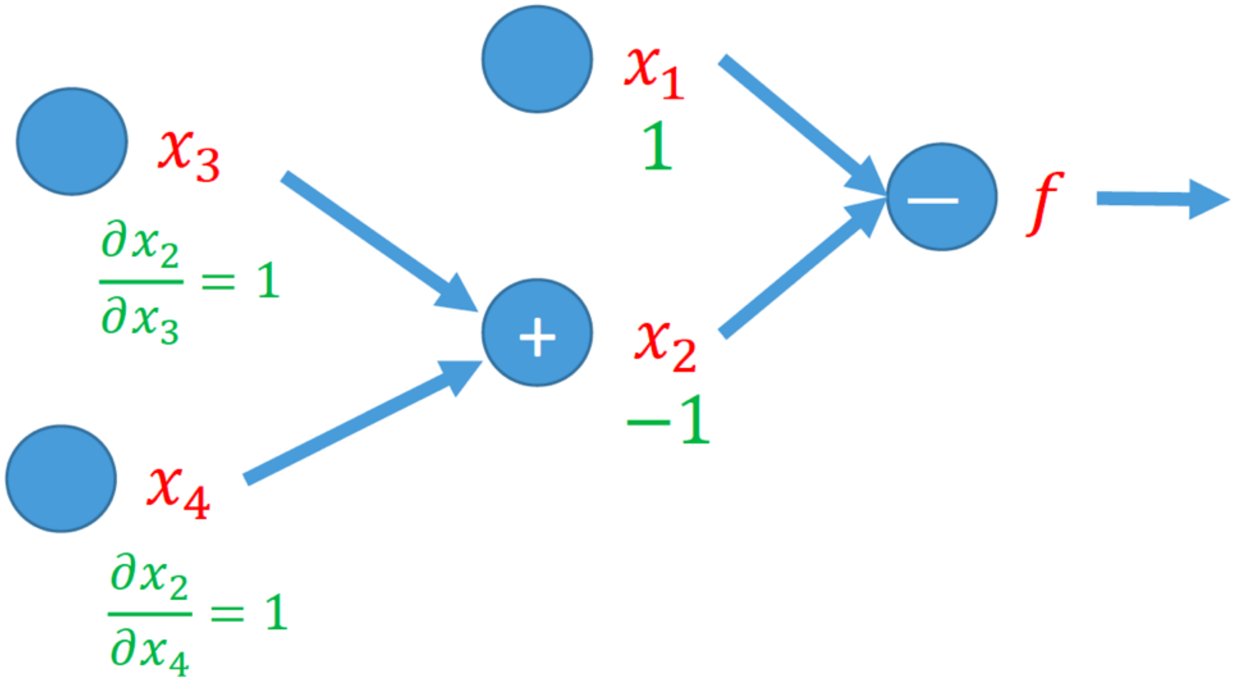


Function: $f = x_1 - x_2$

Gradient: $\frac{\partial f}{\partial x_1} = 1, \frac{\partial f}{\partial x_2} = -1$

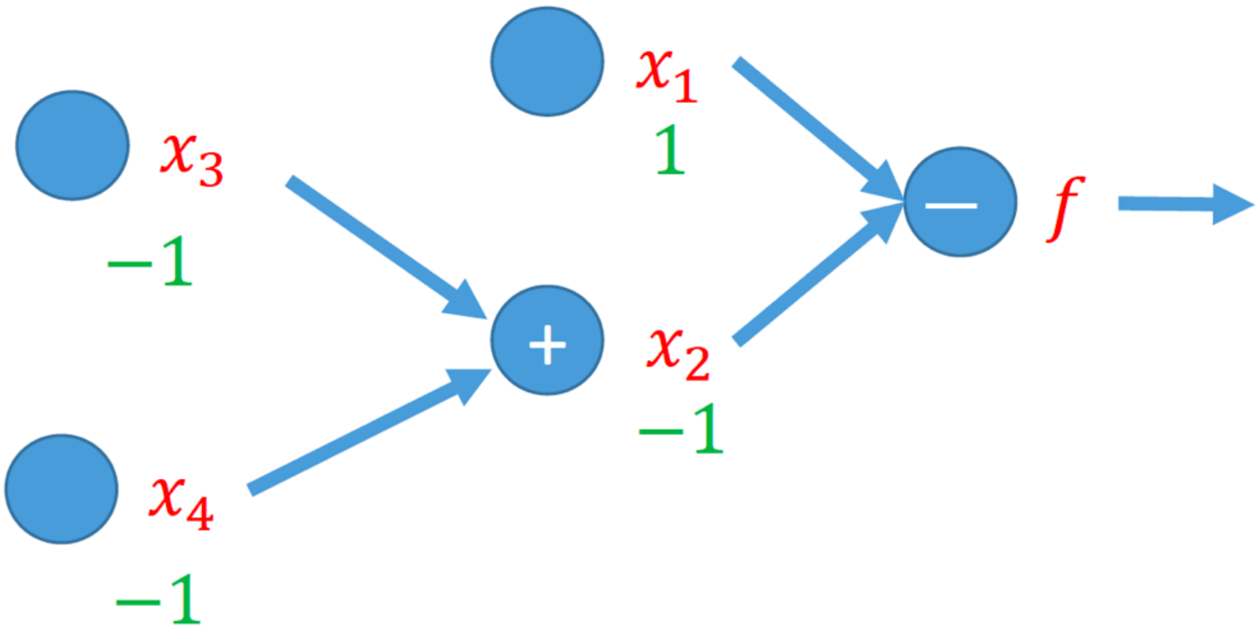


Function: $f = x_1 - x_2 = x_1 - (x_3 + x_4)$



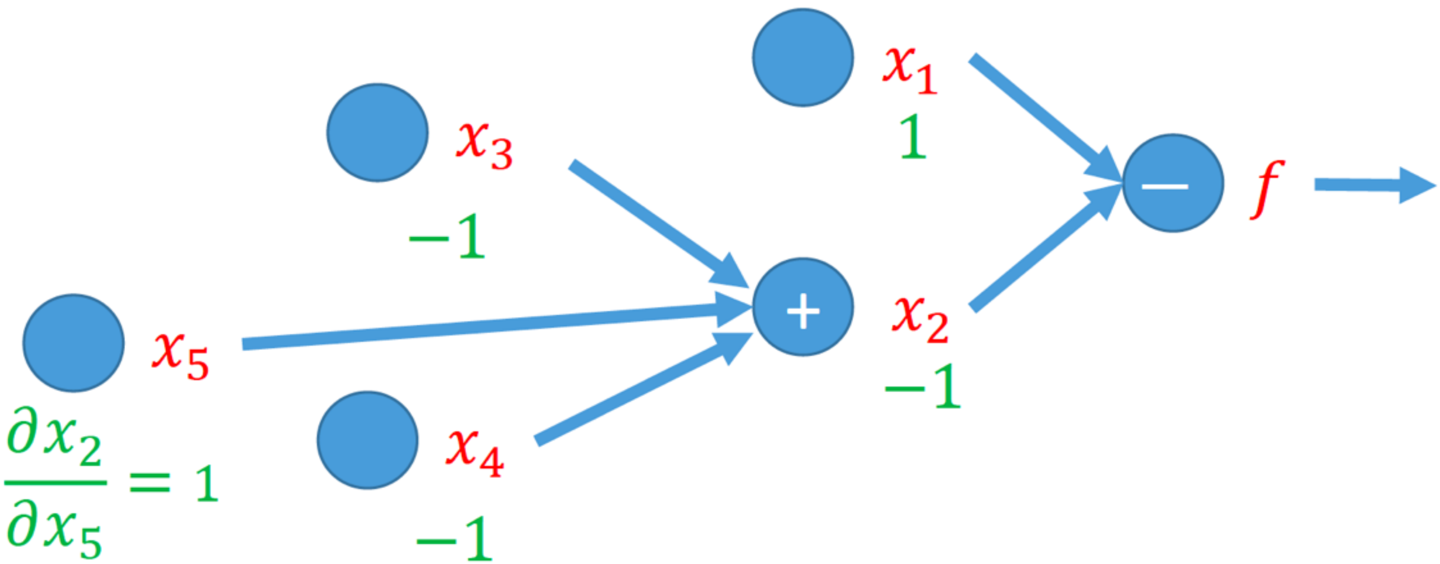
Function: $f = x_1 - x_2 = x_1 - (x_3 + x_4)$

Gradient: $\frac{\partial x_2}{\partial x_3} = 1, \frac{\partial x_2}{\partial x_4} = 1$. What about $\frac{\partial f}{\partial x_3}$?



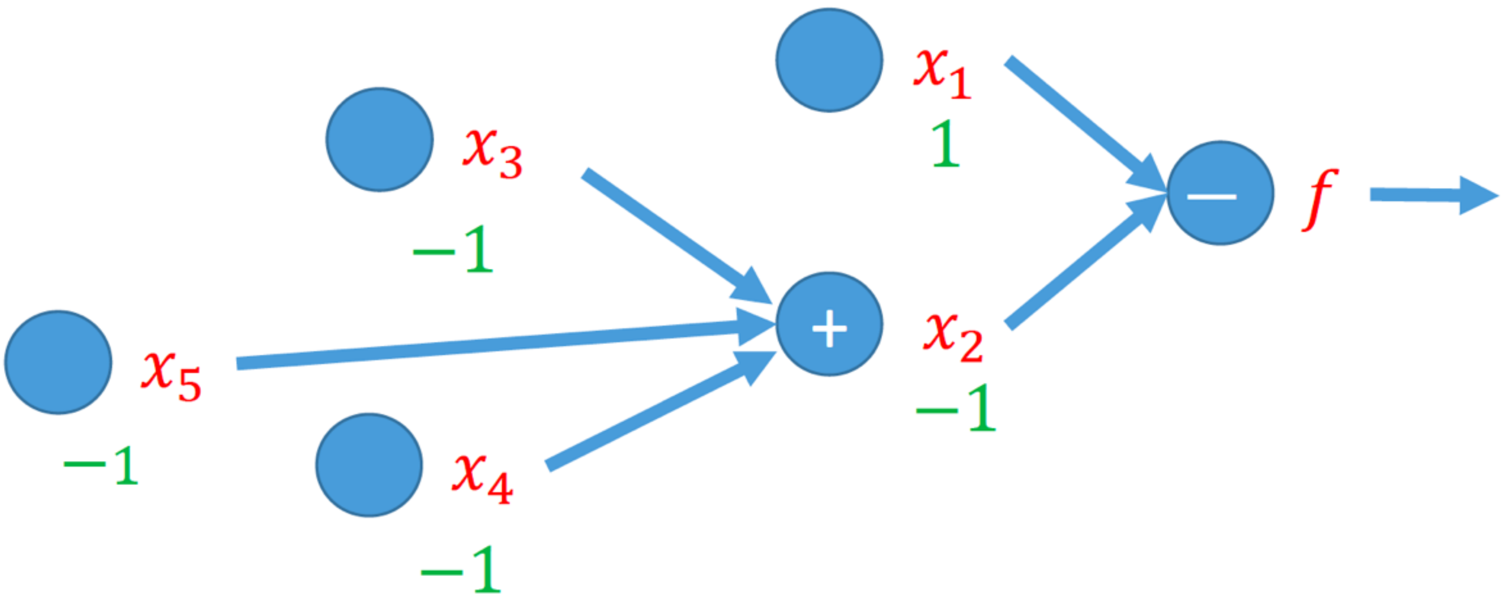
Function: $f = x_1 - x_2 = x_1 - (x_3 + x_4)$

Gradient: $\frac{\partial f}{\partial x_3} = \frac{\partial f}{\partial x_2}\frac{\partial x_2}{\partial x_3} = -1$



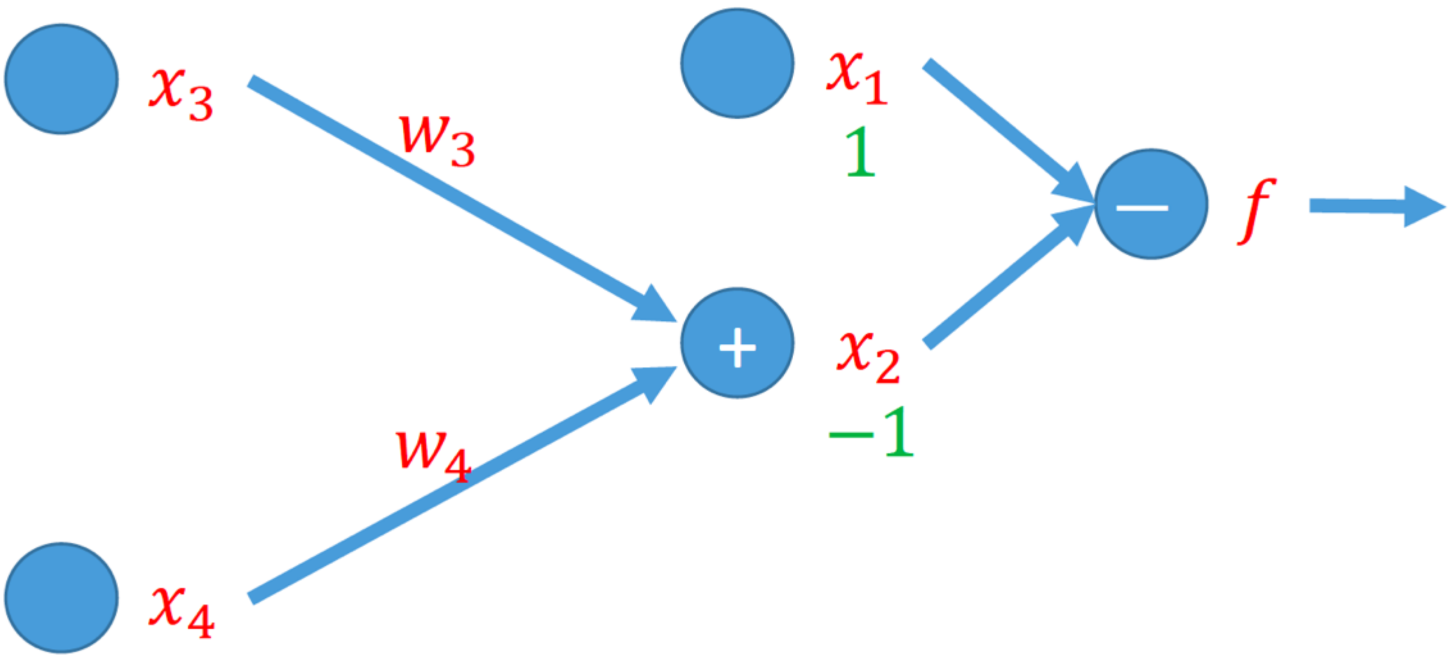
Function: $f = x_1 - x_2 = x_1 - (x_3 + x_5 + x_4)$

Gradient: $\frac{\partial x_2}{\partial x_5} = 1$

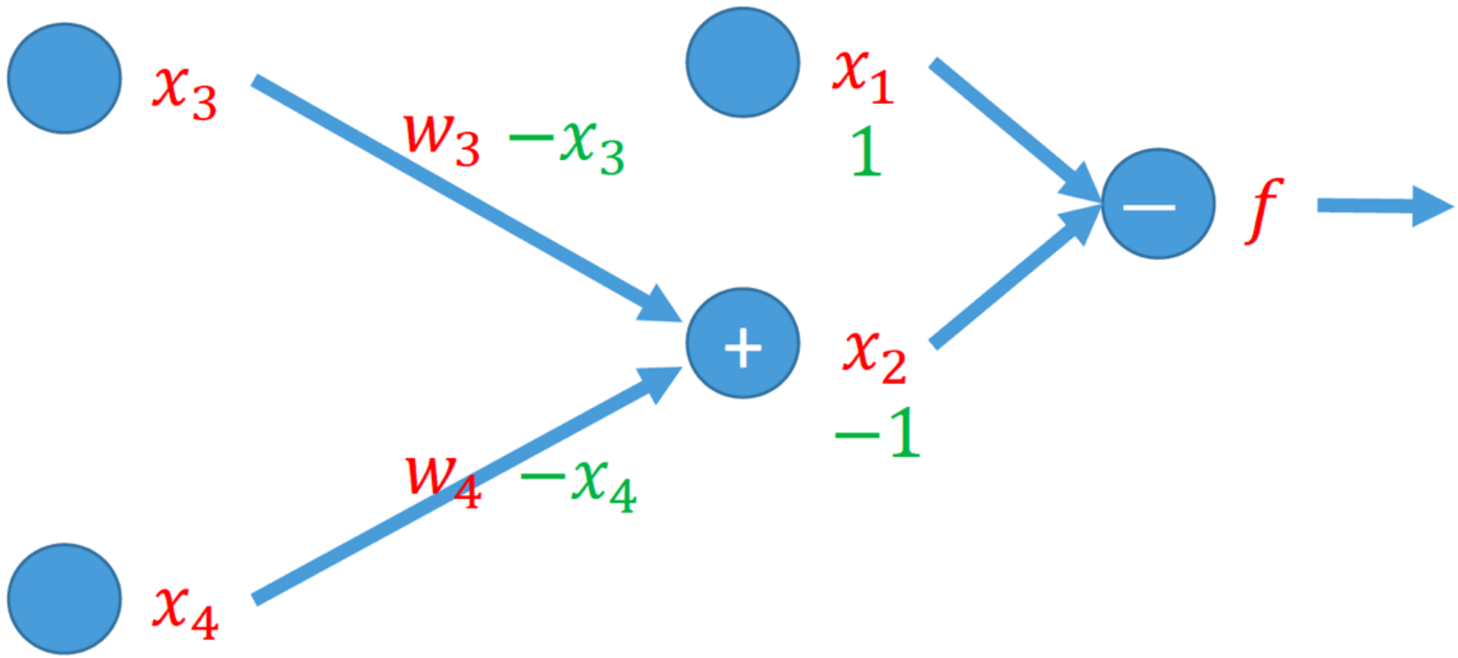


Function: $f = x_1 - x_2 = x_1 - (x_3 + x_5 + x_4)$

Gradient: $\frac{\partial f}{\partial x_5} = \frac{\partial f}{\partial x_2}\frac{\partial x_2}{\partial x_5} = 1$

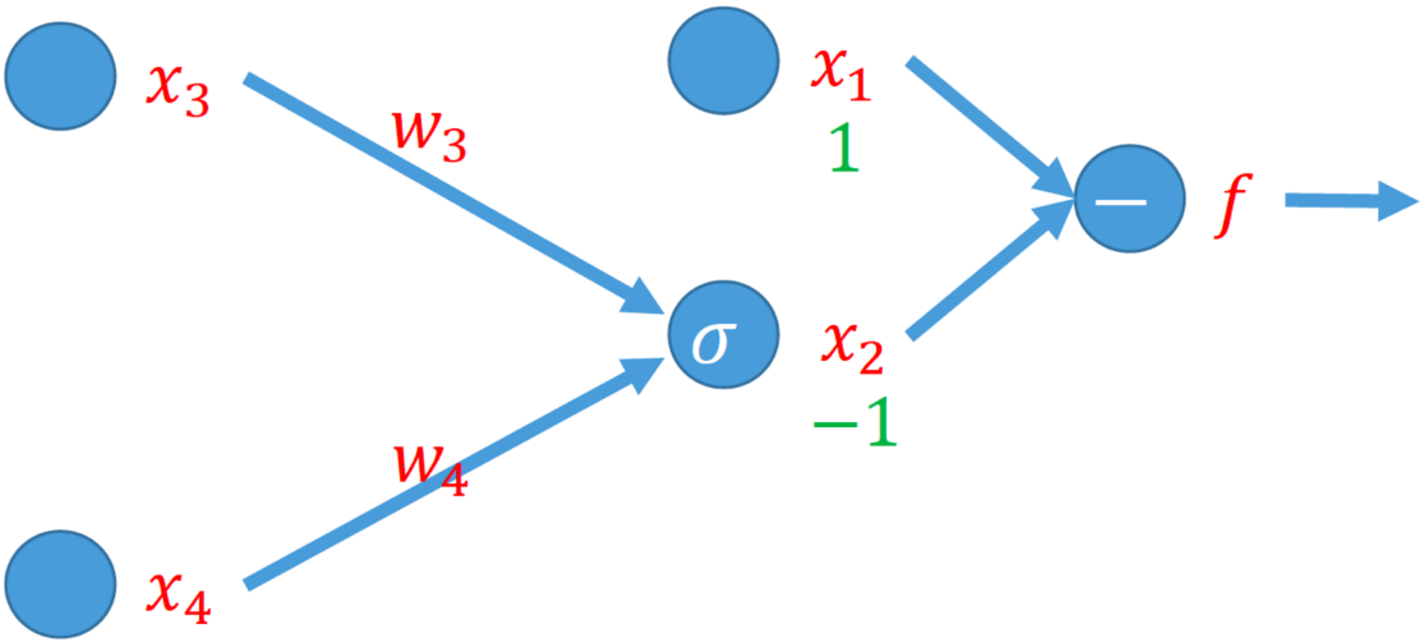


Function: $f = x_1 - x_2 = x_1 - (w_3x_3 + w_4x_4)$

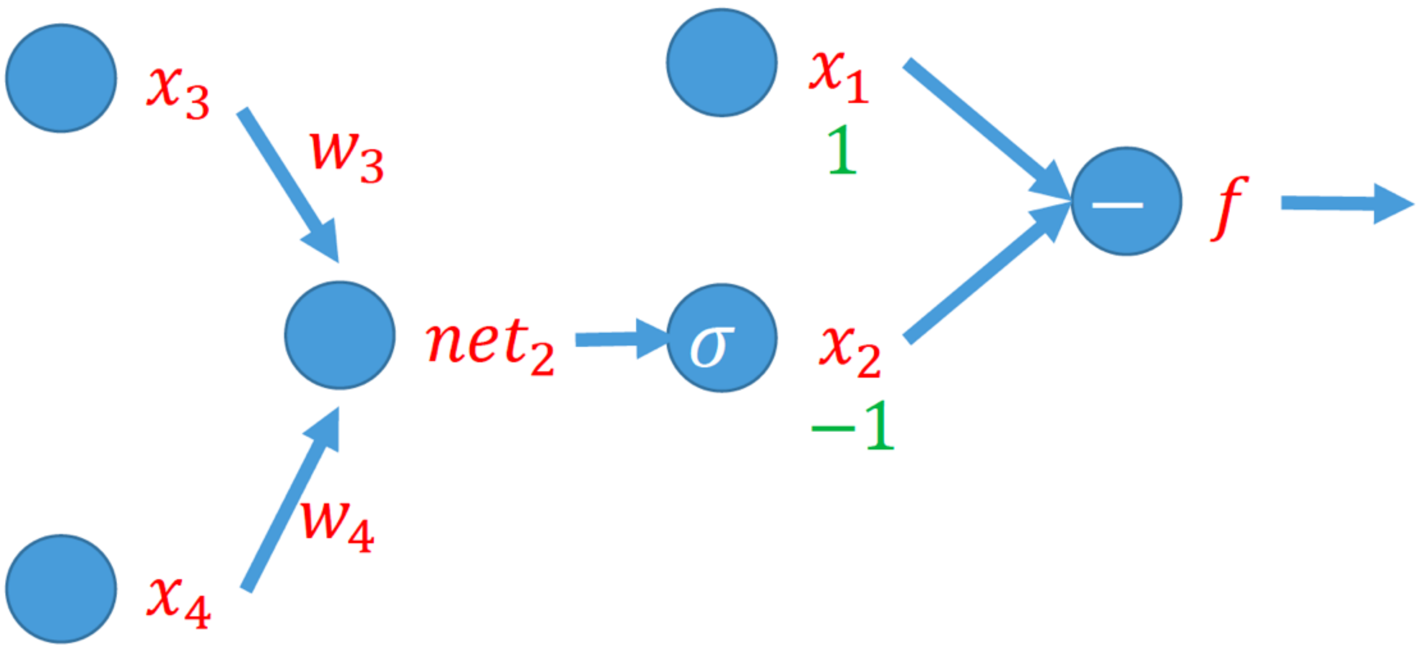


Function: $f = x_1 - x_2 = x_1 - (w_3 x_3 + w_4 x_4)$

Gradient: $\frac{\partial f}{\partial w_3} = \frac{\partial f}{\partial x_2} \frac{\partial x_2}{\partial w_3} = -1 \times x_3 = -x_3$

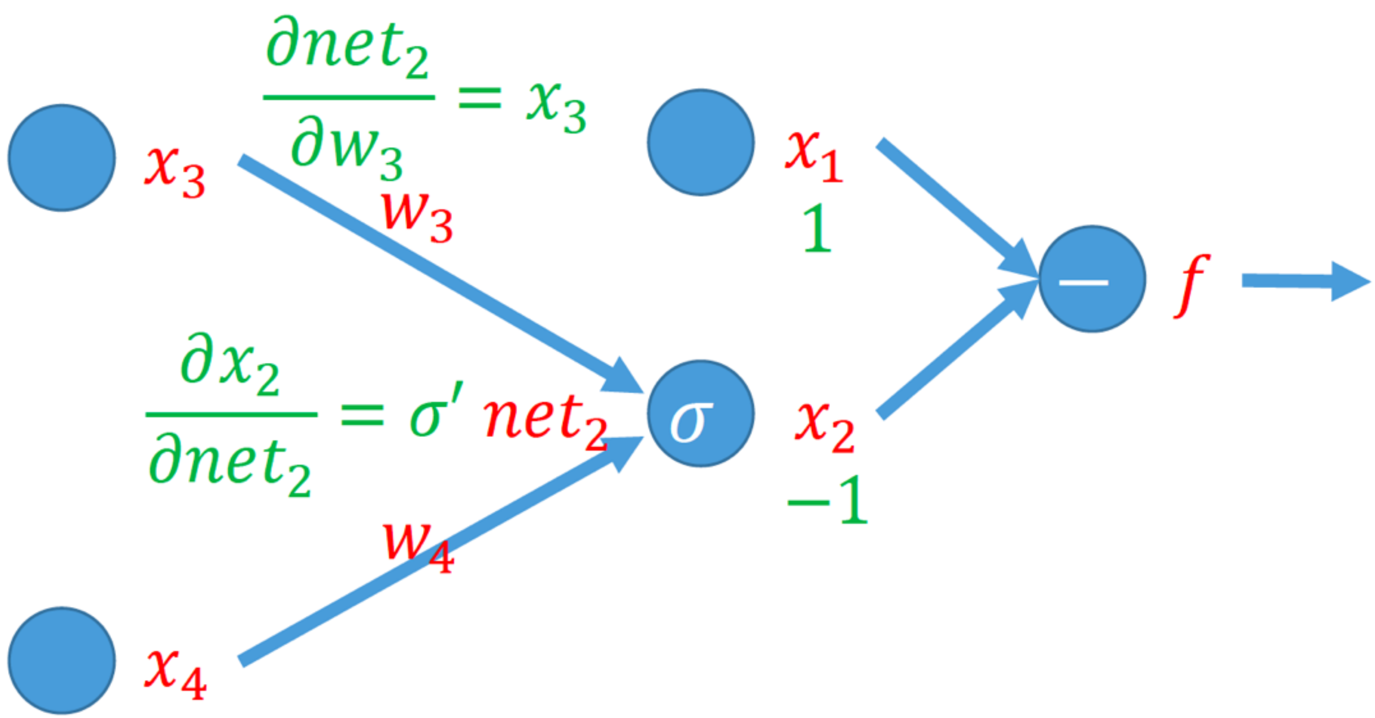


Function: $f = x_1 - x_2 = x_1 - \sigma(w_3 x_3 + w_4 x_4)$



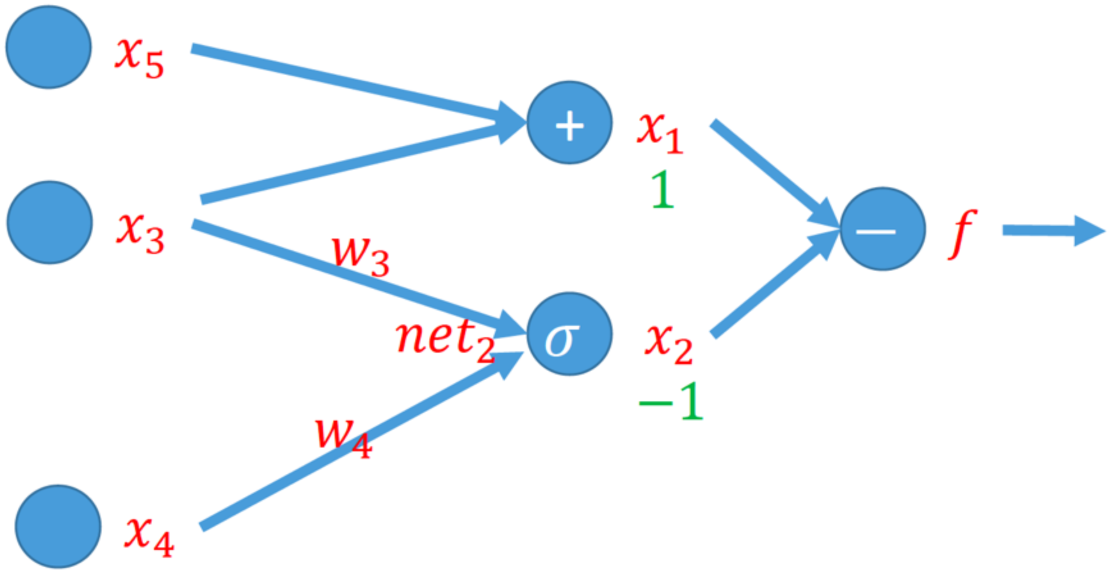
Function: $f = x_1 - x_2 = x_1 - \sigma(w_3 x_3 + w_4 x_4)$

Let $net_2 = w_3 x_3 + w_4 x_4$

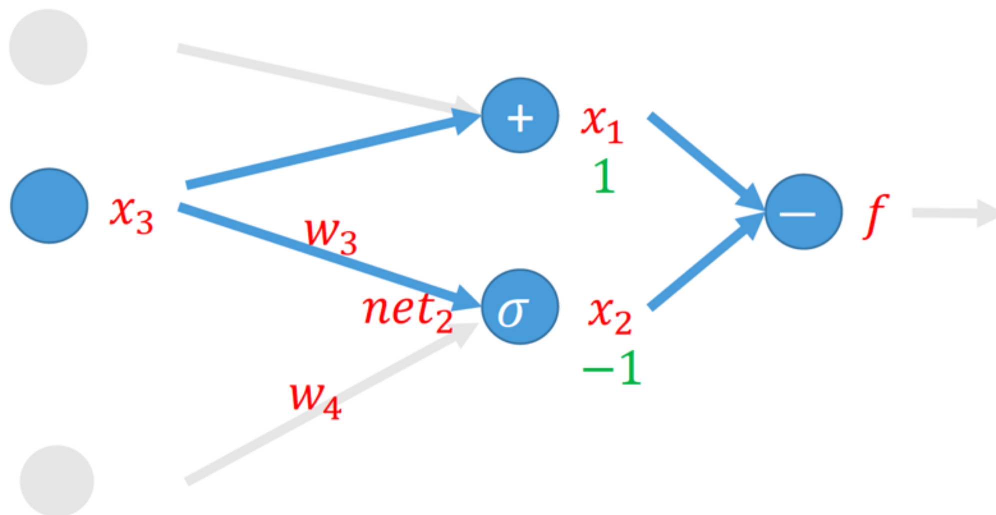


Function: $f = x_1 - x_2 = x_1 - \sigma(w_3 x_3 + w_4 x_4)$

Gradient: $\frac{\partial f}{\partial w_3} = \frac{\partial f}{\partial x_2}\frac{\partial x_2}{\partial net_2}\frac{\partial net_2}{\partial w_3} = -1 \times \sigma' \times x_3 = -\sigma' x_3$



Function $f = x_1 - x_2 = (x_3 + x_5) - \quad \sigma(w_3 x_3 + w_4 x_4)$

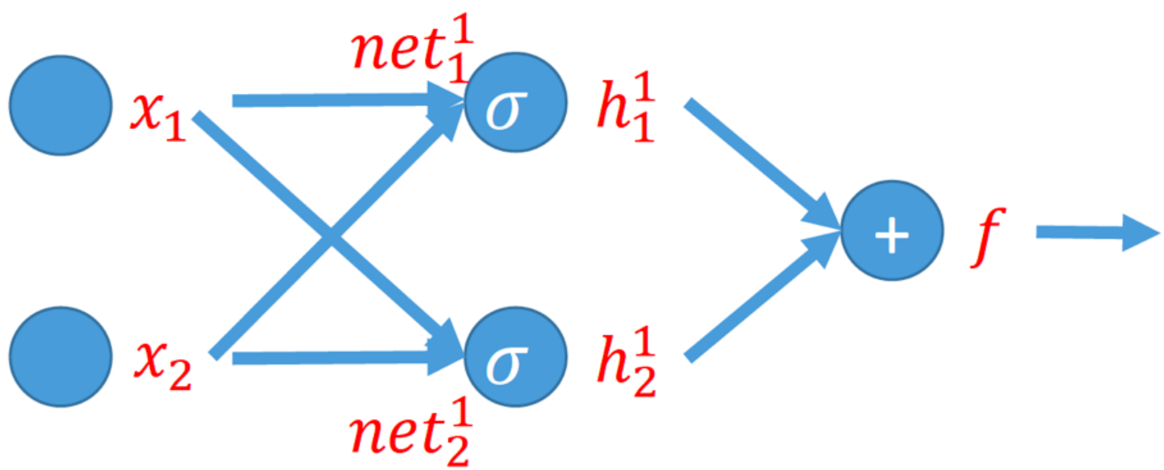


Function: $f = x_1 - x_2 = (x_3 + x_5) - \sigma(w_3 x_3 + w_4 x_4)$

Gradient: $\frac{\partial f}{\partial x_3} = \frac{\partial f}{\partial x_2}\frac{\partial x_2}{\partial net_2}\frac{\partial net_2}{\partial x_3} + \frac{\partial f}{\partial x_1}\frac{\partial x_1}{\partial x_3} = -1 \times \sigma' \times w_3 + 1 \times 1 = -\sigma' w_3 + 1$

Summary

- Forward to compute f
- Backward to compute the gradients



Activation Functions

- ReLU $\mathrm{ReLU}(x) = \max\{x, 0\}$

$$\mathrm{ReLU}'(x) = \begin{cases} 0, & x \le 0 \\ 1, & x > 0 \end{cases}$$

Sub-gradient used at $x = 0$

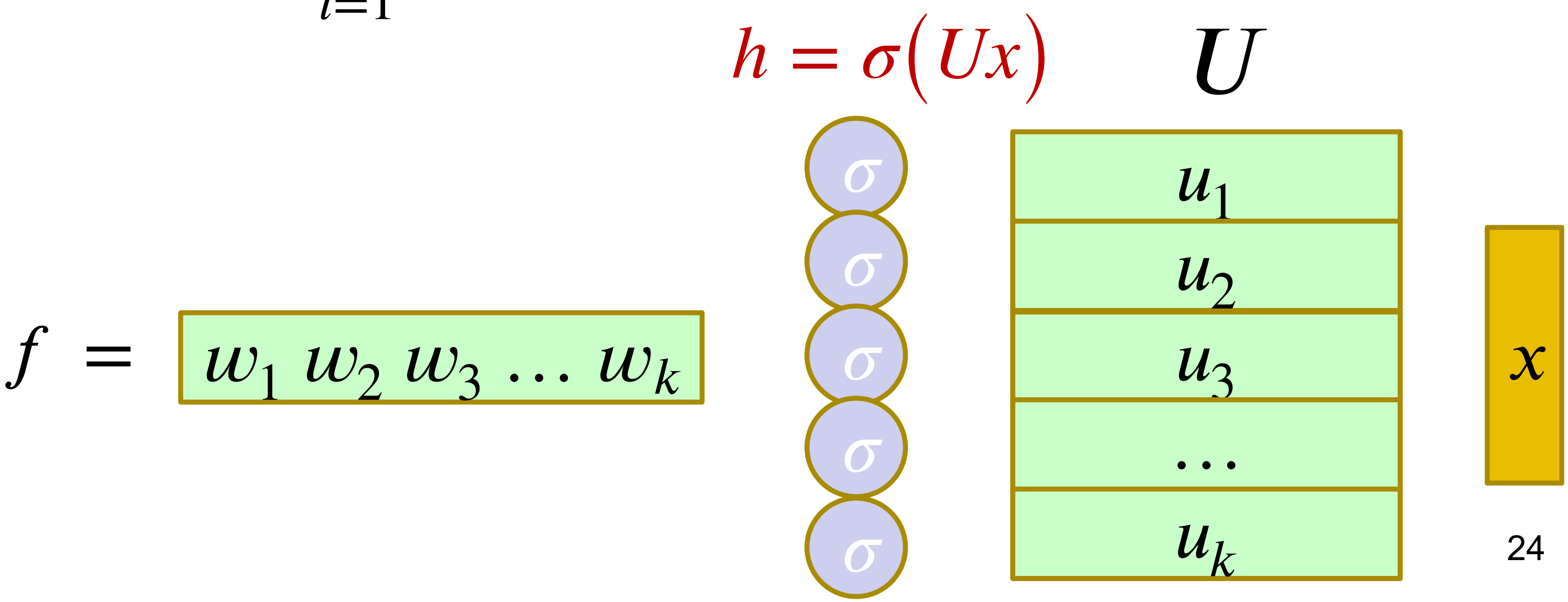
- Sigmoid $\sigma(x) = \dfrac{1}{1 + e^{-x}}$

$$\sigma'(x) = \sigma(x) \cdot \big(1 - \sigma(x)\big)$$

- $\tanh(x) = \dfrac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$

One-hidden layer network

- $f_\theta(x) = w^\top \sigma(Ux)$ where $\theta = (w, U)$
- $f_\theta(x) = \sum_{i=1}^{k} w_i \, \sigma(u_i^\top x)$



One-hidden layer network

- $f_\theta(x) = \sum_{i=1}^{k} w_i \sigma(u_i^\top x)$
- $\nabla f_\theta(x) = \left(\frac{\partial f}{\partial w}, \frac{\partial f}{\partial U}\right)$
- $\frac{\partial f}{\partial w} = h = \sigma\left(Ux\right)$
- $\frac{\partial f}{\partial U} = ?$

One-hidden layer network

- $$\frac{\partial f}{\partial u_i} = \frac{\partial w_i \sigma\left(u_i^\top x\right)}{\partial u_i} = w_i \sigma'\left(u_i^\top x\right) x$$

Overall size $k \times d$

$$\frac{\partial f}{\partial U} = \left(w \odot \sigma'(Ux)\right) x^\top$$

$k \times 1$ vector with entries $w_i \sigma'\left(u_i^T x\right)$

$1 \times d$ input features

$\odot$ is Hadamard product (entry-wise multiplication)

One-hidden layer network

- $\dfrac{\partial f}{\partial w} = \sigma\left(Ux\right)$
- $\dfrac{\partial f}{\partial U} = \left(w \odot \sigma'(Ux)\right)x^\top$

- Consider $\mathscr{L}(\theta) = \dfrac{1}{2}\left(w^\top \sigma(Ux) - y\right)^2$

$$\frac{\partial \mathscr{L}}{\partial w} = \left(w^\top \sigma(Ux) - y\right)\sigma(Ux)$$

$$\frac{\partial \mathscr{L}}{\partial w} = \left(w^\top \sigma(Ux) - y\right)\left(w \odot \sigma'(Ux)\right)x^\top$$

Training a one-hidden layer net

Initialize w_0, U_0

Learning rate $\eta > 0$, duration END

for $0 \leq t \leq \mathrm{END} - 1$

1. Randomly pick example (x, y) from training data
2. Update weights
   - $w_{t+1} = w_t - \eta \left(w^\top \sigma(Ux) - y \right) \sigma(Ux)$
   - $U_{t+1} = U_t - \eta \left(w^\top \sigma(Ux) - y \right) \left(w \odot \sigma'(Ux) \right) x^\top$

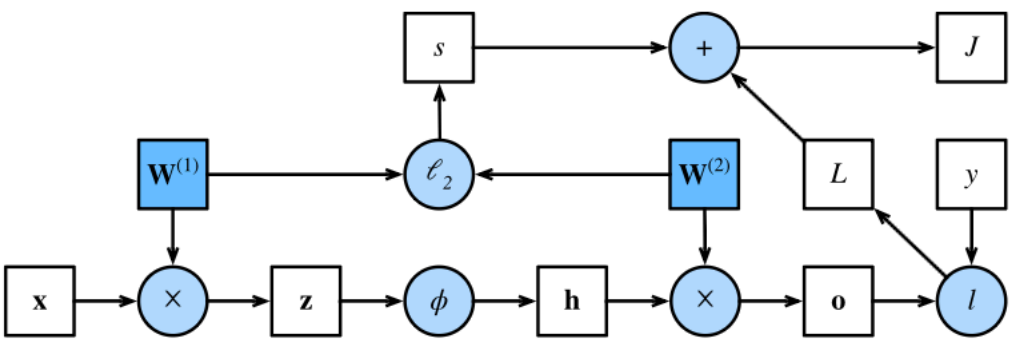
Return final model $\theta_{\mathrm{FINAL}} = \left(w_{\mathrm{END}}, U_{\mathrm{END}} \right)$

For HW2, you will train your own neural network from scratch.

Break

Computation Graph

- Optimize
  $$J = \ell(W^{(2)}\phi(W^{(1)}x), y) + \|W^{(1)}\|_F^2 + \|W^{(2)}\|_F^2$$




```
# Compute prediction and loss
pred = model(X)
loss = loss_fn(pred, y)

# Backpropagation
loss.backward()
optimizer.step()
optimizer.zero_grad()
```

Computation Graph

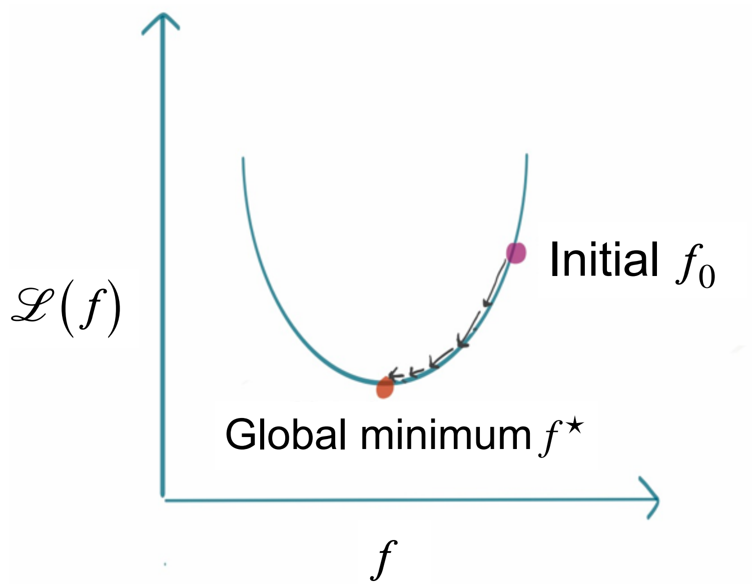
- Forward pass:
  - Generate dynamic graph (at least for PyTorch)
  - Store a, z lists
- Backward pass:
  - loss.backward() calculate gradients
  - Release memory
- Inference: torch.no_grad()
  - No graph will be generated (and no need to store intermediate values)

Gradient Descent Revisit

- Gradient descent iterations

$$f_{i+1} = f_i - \eta \nabla \mathscr{L}(f_i)$$

New model — Current model — Learning rate — Gradient



Linear regression

1. Data: $\left(x_i, y_i\right)_{i=1}^{n}$, $x_i \in \mathbb{R}^d, y_i \in \mathbb{R}$
2. Model: Linear model $f(x; w) = x^\top w$
3. Squared Loss: $\ell(f(x), y) = \dfrac{1}{2}\left(f(x) - y\right)^2$
4. Optimize

$$\widehat{w} = \arg\min_{w \in \mathbb{R}^d} \widehat{\mathscr{L}}(w) \quad \text{where} \quad \widehat{\mathscr{L}}(w) = \frac{1}{2n}\sum_{i=1}^{n}(x_i^\top w - y_i)^2$$

Linear regression

- Let us move to matrix notation

$$X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix}$$

$n \times d$
matrix

$$y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{bmatrix}$$

$n \times 1$
vector

- The loss function becomes

$$\widehat{\mathscr{L}}(w) = \frac{1}{2n} \sum_{i=1}^{n} (x_i^\top w - y_i)^2 = \frac{1}{2n} \| y - Xw \|_2^2$$

GD for Linear regression

Optimization becomes

$$w^\star = \arg\min_{w \in \mathbb{R}^d} \frac{1}{2} \| y - Xw \|_2^2$$

- Calculate the gradient

$$\nabla \mathscr{L}(w) = X^\top X w - X^\top y$$

- Gradient iterations are given by

$$w_{t+1} = w_t - \eta \nabla \mathscr{L}(w_t)$$

GD for Linear regression

Plugging in the gradient formula

$$w_{t+1} = w_t - \eta\left(X^\top X w_t - X^\top y\right)$$

Questions:

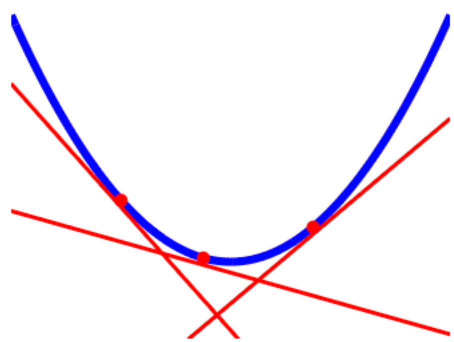
- Which learning rate to use?
- How many iterations needed?
- Can we speed up further?

Answers are important for **deep learning** as well.

Convex Optimization

- A (differentiable) function $f : \mathbb{R}^d \to \mathbb{R}$ is convex iff, for any $x_1, x_2 \in \mathbb{R}^d$, we have

$$f(x_1) \geq f(x_2) + \nabla f(x_2)^\top (x_1 - x_2)$$



Convex Optimization

- A function $f : \mathbb{R}^d \rightarrow \mathbb{R}$ is L-smooth if for any $x_1, x_2 \in \mathbb{R}^d$

$$\|\nabla f(x_1) - \nabla f(x_2)\|_2 \leq L\|x_1 - x_2\|_2$$

- L-smooth implies that for any $x_1, x_2 \in \mathbb{R}^d$

$$f(x_1) \leq f(x_2) + \nabla f(x_2)^\top (x_1 - x_2) + \frac{L}{2}\|x_1 - x_2\|_2^2$$

GD for Convex Functions

- Let $f : \mathbb{R}^d \to \mathbb{R}$ be a convex function that is also L-smooth
- Let $w^\star := \arg\min_w f(w)$ — the minimizer we try to find
- Consider the following gradient descent update

$$w_{t+1} = w_t - \eta \nabla f(w_t)$$

Step I: The Descend

- If $\eta \leq 1/L$, we then have

$$f(w_{t+1}) \leq f(w_t) - \frac{\eta}{2}\|\nabla f(w_t)\|_2^2$$

Function value decreases strictly unless $\nabla f(w_t) = 0$

Step II: Convergence Rate

We have

- $f(w_{t+1}) - f(w^\star) \leq \frac{1}{2\eta}\left(\|w_t - w^\star\|_2^2 - \|w_{t+1} - w^\star\|_2^2\right)$

- $f(w_T) - f(w^\star) \leq \frac{\|w_0 - w^\star\|_2^2}{2\eta T}$
  - To achieve ε error, we need $O(\frac{1}{\eta\varepsilon}) = O(\frac{L}{\varepsilon})$ number of iterations

GD for Linear regression

Optimization becomes

$$f(w) = \frac{1}{2} \| y - Xw \|_2^2$$

- f is convex (actually quadratic)
- It's gradient equals to

$$\nabla f(w) = X^\top X w - X^\top y$$

GD for Linear regression

- It's gradient equals to

$$\nabla f(w) = X^\top X w - X^\top y$$

- $\nabla f(w)$ is $\lambda_{\max}\left(X^\top X\right)$-Lipschitz

$$|\nabla f(w_1) - \nabla f(w_2)| = |X^\top X(w_1 - w_2)| \leq \lambda_{\max}(X^\top X) \cdot \|w_1 - w_2\|_2$$

Select the learning rate as $\eta = 1/\lambda_{\max}(X^\top X)$ and run for $O\left(\lambda_{\max}(X^\top X)/\varepsilon\right)$ iterations ensures convergence up to an ε-optimal minimizer.