

# Intro to Deep Learning

## Lecture 4: Project and Backpropagation

# Experiments Files

The screenshot shows a GitHub repository page for "UCR-EECS228-Introduction-to-Deep-Learning". The repository is public and has 1 branch and 0 tags. The main file listed is "2\_layer\_nn.ipynb", which was added 10 hours ago. Other files listed are "LICENSE" and "README.md", both of which are initial commits from 10 hours ago. The repository description states: "Experiments conducted in UCR's EE/CS 228: Introduction to Deep Learning course". The repository has 1 star, 1 watching, and 0 forks. It also includes sections for Releases, Packages, and Languages.

yinglunz / UCR-EECS228-Introduction-to-Deep-Learning

Code Issues Pull requests Actions Projects Wiki Security Insights Settings

UCR-EECS228-Introduction-to-Deep-Learning Public

main 1 Branch 0 Tags Go to file Add file Code

Yinglun Zhu Add 2-layer-NN experiments on MNIST. 68fd5a4 · 10 hours ago 2 Commits

2\_layer\_nn.ipynb Add 2-layer-NN experiments on MNIST. 10 hours ago

LICENSE Initial commit 10 hours ago

README.md Initial commit 10 hours ago

README MIT license

**UCR-EECS228-Introduction-to-Deep-Learning**

Experiments conducted in UCR's EE/CS 228: Introduction to Deep Learning course

About

Experiments conducted in UCR's EE/CS 228: Introduction to Deep Learning course

Readme MIT license Activity 1 star 1 watching 0 forks

Releases

No releases published [Create a new release](#)

Packages

No packages published [Publish your first package](#)

Languages

Jupyter Notebook 100.0%

<https://github.com/yinglunz/UCR-EECS228-Introduction-to-Deep-Learning>

# Course Project

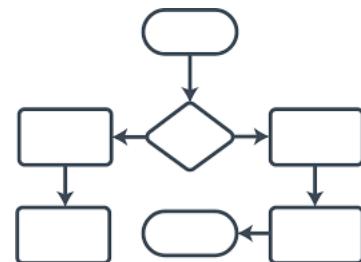
## Expectations

1. A project topic
  - reasonably complex but not too complex
2. Study it in detail
  - Identify challenges
  - Connection to prior art
  - Try different solutions
3. Demonstration
  - Empirical results
  - Something to show off

Show your work,  
and you should  
get your credit!

# Project Proposal

- 1-page proposal Due: Feb 13
- Describes the idea and its feasibility
  - which problem, which dataset, why deep learning
- Suggestion: Add a workflow chart
- Decompose the work
  - Justify each group member will contribute



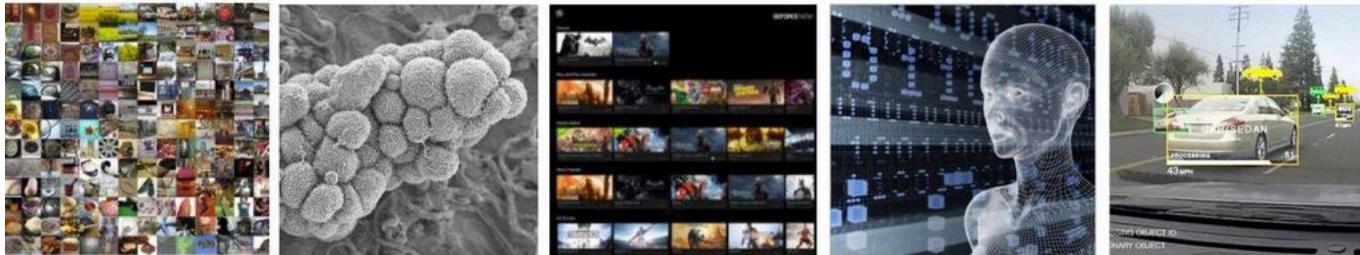
# Project Report Format

- Should resemble an academic paper
- Expected length:
  - Between 6 to 8 pages (not including references)
  - NeurIPS format
  - <https://neurips.cc/Conferences/2023/PaperInformation/StyleFiles>
- Example Structure:
  - Introduction and Summary of Contribution
  - Discussion of Prior art
  - Proposed method
  - Experiments (e.g. from your demo)
  - Discussion & future plans: How would you improve?

# Topics in Deep Learning

Many deep learning research and applications

## DEEP LEARNING EVERYWHERE



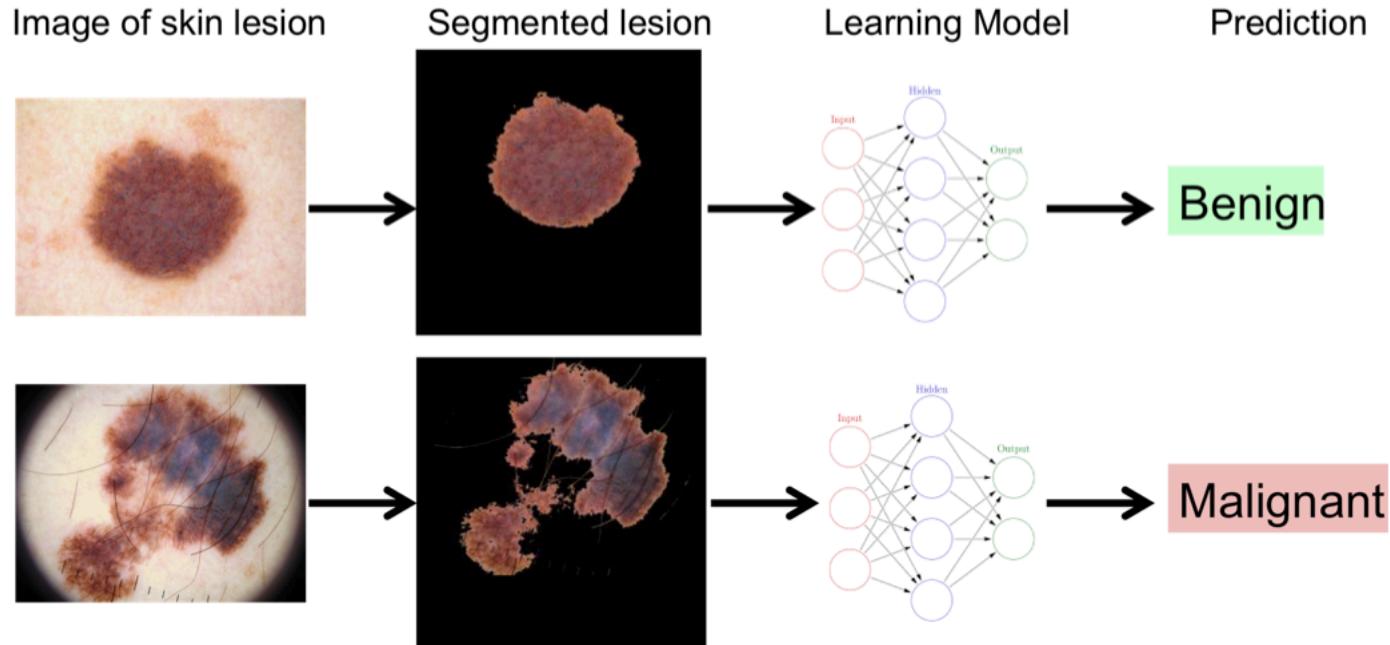
INTERNET & CLOUD	MEDICINE & BIOLOGY	MEDIA & ENTERTAINMENT	SECURITY & DEFENSE	AUTONOMOUS MACHINES
Image Classification Speech Recognition Language Translation Language Processing Sentiment Analysis Recommendation	Cancer Cell Detection Diabetic Grading Drug Discovery	Video Captioning Video Search Real Time Translation	Face Detection Video Surveillance Satellite Imagery	Pedestrian Detection Lane Tracking Recognize Traffic Sign

# Project Resources

- Data: <https://www.kaggle.com/>
- Models: <https://huggingface.co/>
- Papers: <https://arxiv.org/>
- Code: <https://github.com/>



# Computer Vision



Example project from Stanford's class:  
<https://web.stanford.edu/~kalouche/cs229.html>

# Computer Vision

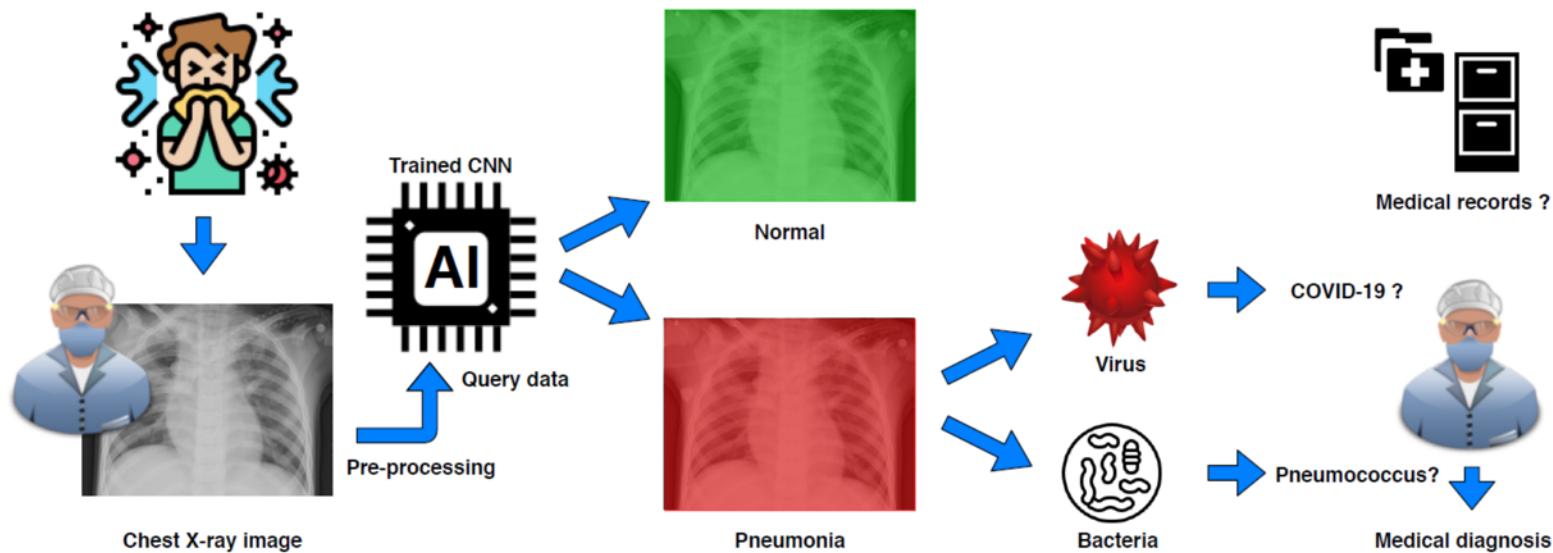
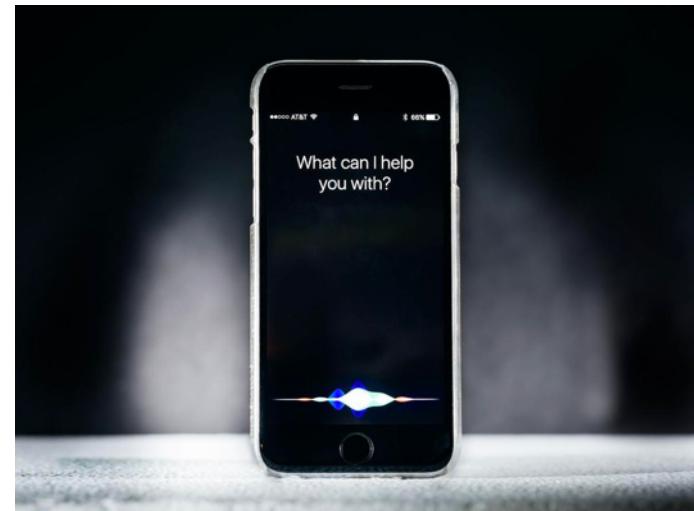
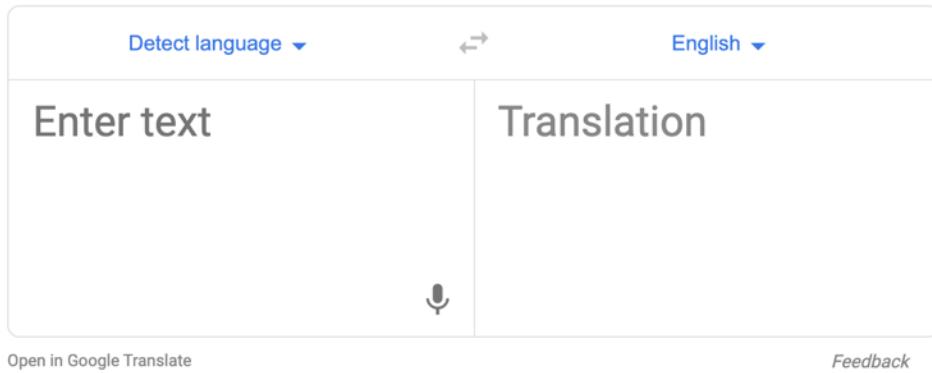


Fig. 1: Global workflow using deep learning for **automatic detection of infection towards supporting COVID-19 screening** from chest X-ray images. In a COVID-19 epidemic context, a detected viral pneumonia can particularly presume a COVID-19 infection.

Paper: <https://arxiv.org/pdf/2004.03399.pdf>

# Natural Language Processing



Neural machine translation: <https://github.com/tensorflow/nmt>

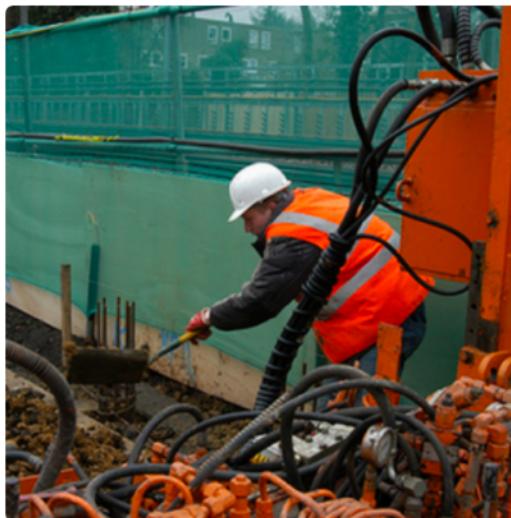
Pretrained language models: <https://github.com/google-research/bert>

# Natural Language Processing

## Application: Image to Sentence



"man in black shirt is playing  
guitar."



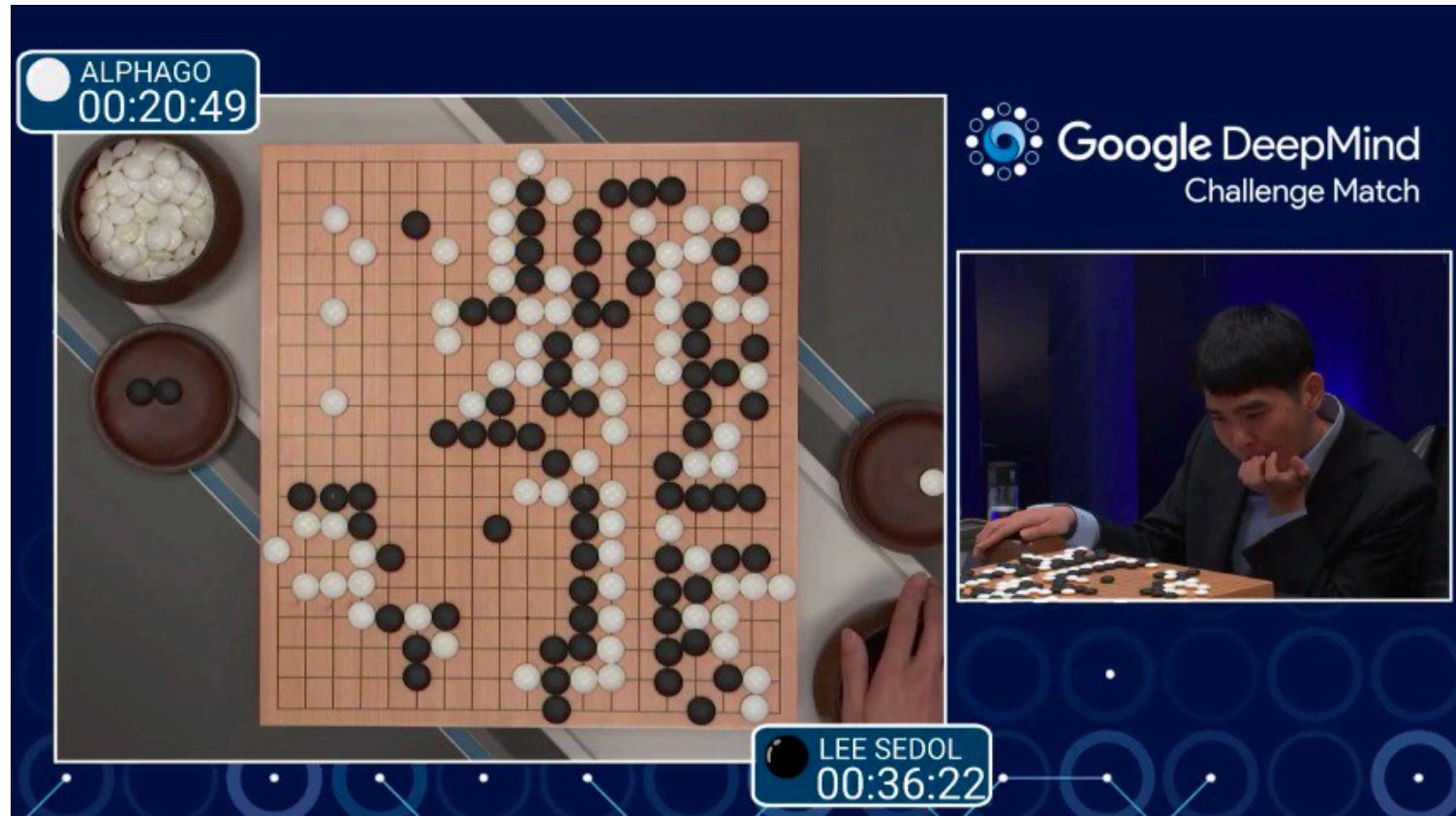
"construction worker in orange  
safety vest is working on road."



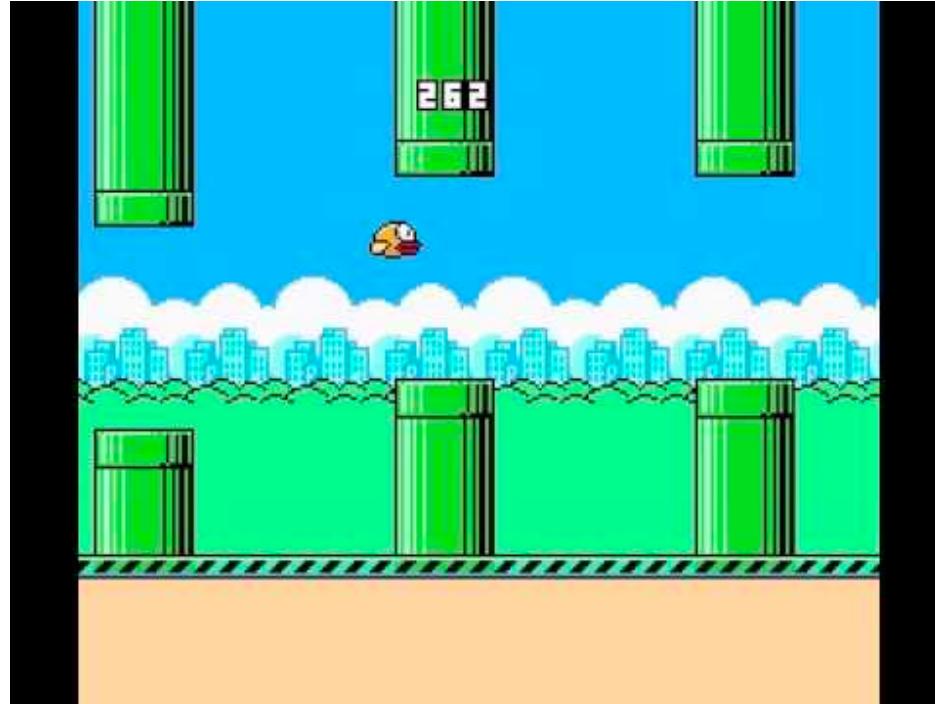
"two young girls are playing with  
lego toy."

# Deep Reinforcement Learning

**Q:** How can we make optimal **decision** in uncertain environments?



# Deep Reinforcement Learning

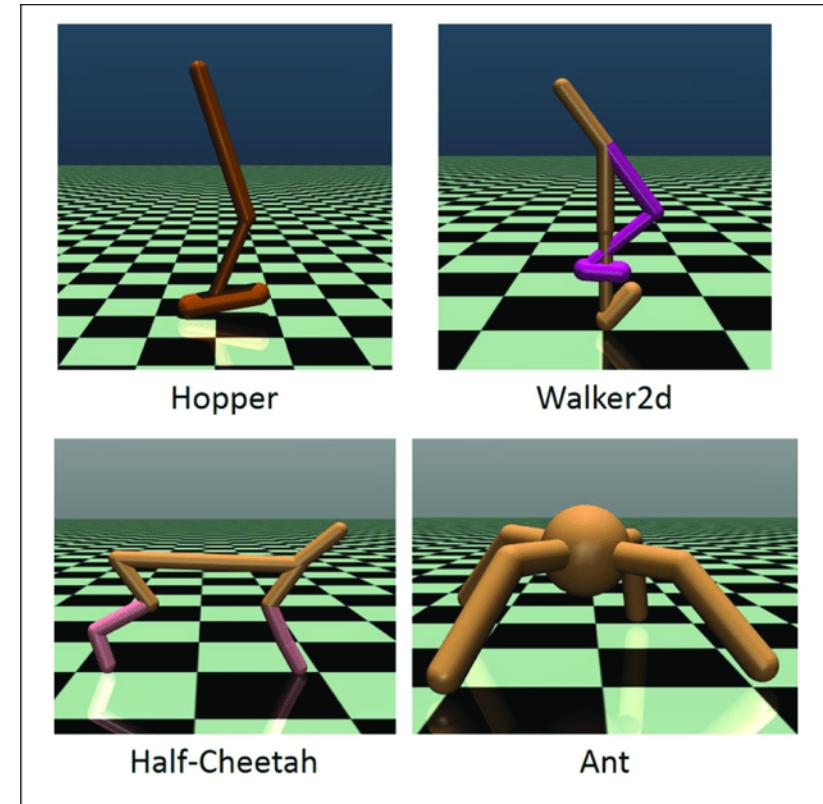


<https://github.com/yenchenlin/DeepLearningFlappyBird>

# Deep Reinforcement Learning

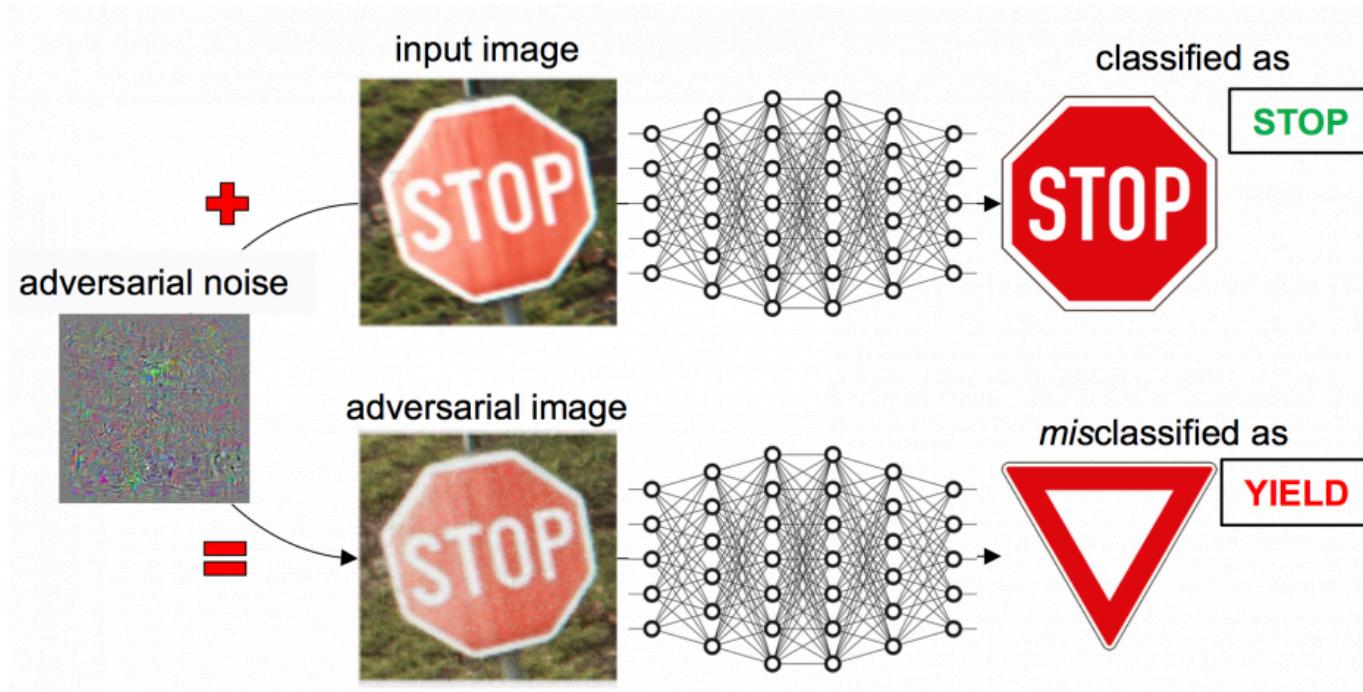
Simulation environments:

- OpenAI Gym
- MuJoCo

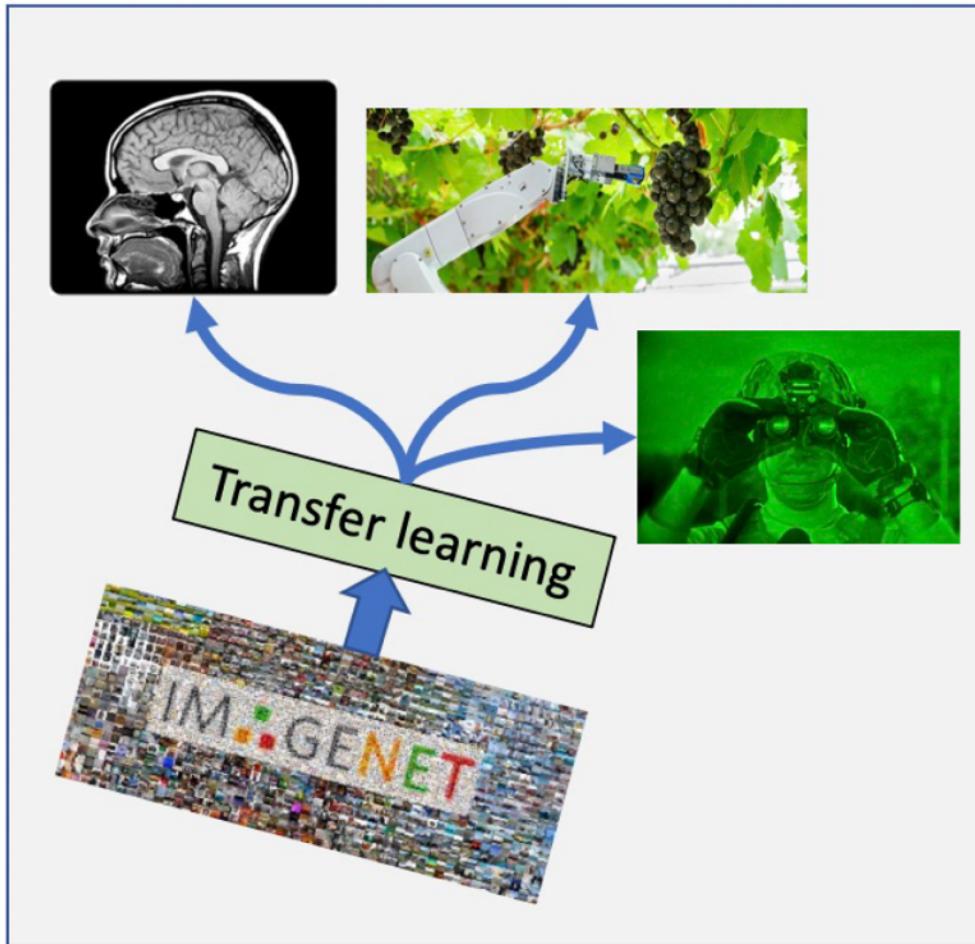


# Adversarial Learning

- Can we mess with an ML model?



# Learning New Tasks

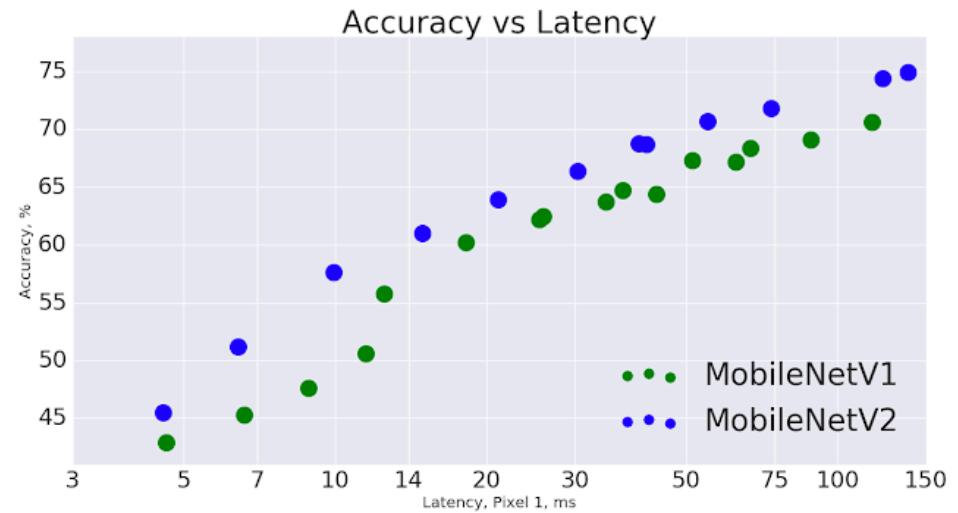
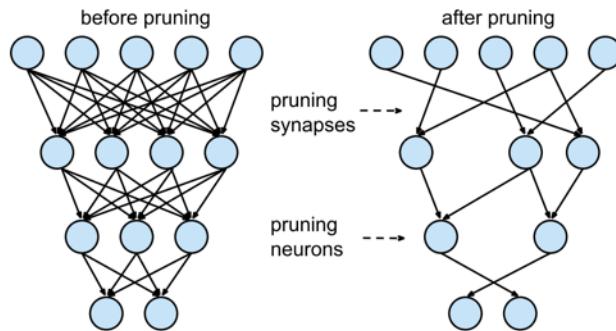


Related keywords:

- Meta-learning
- Domain adaptation
- Multi-task learning

# Mobile Friendly ML

Q: Can we fit deep networks in a smartwatch?



# Generative Modeling

## Open-source models

- Facebook's Llama, Stanford Alpaca, StableDiffusion

## Accessing SOTA models through APIs

- ChatGPT, Gemini

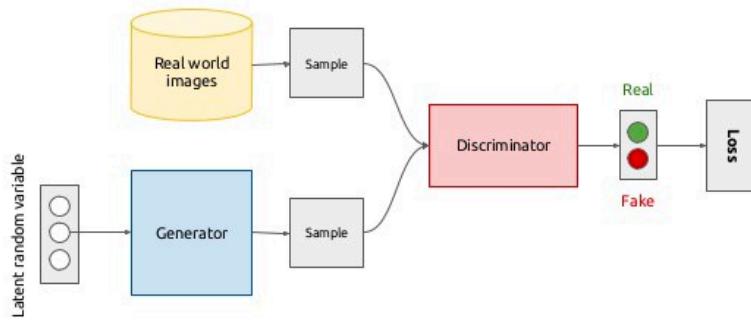
## Chaining Language Model Outputs for Autonomous Agents

✓ <https://github.com/Torantulino/Auto-GPT>

✓ <https://github.com/hwchase17/langchain>

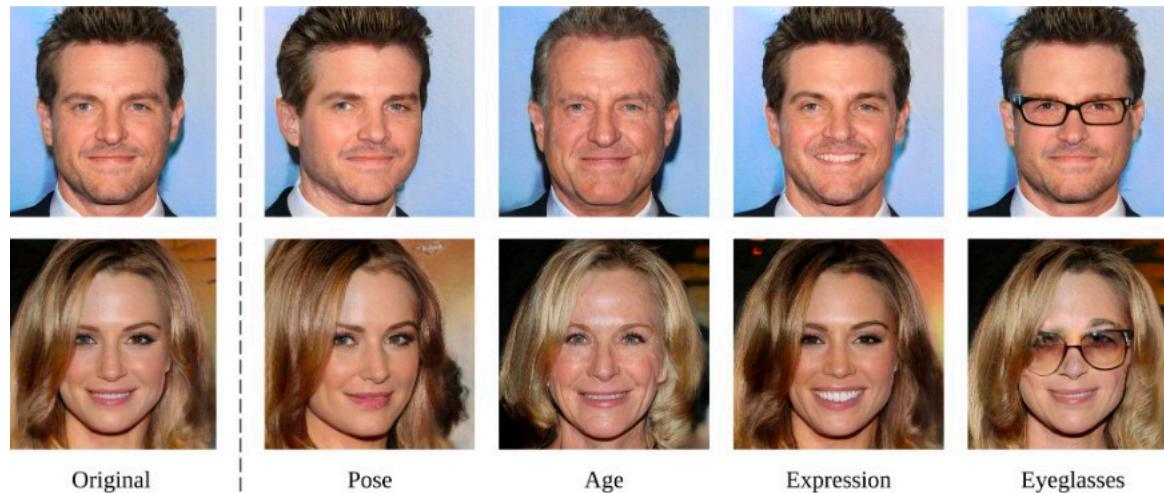
# Generative Models

Generative adversarial networks (conceptual)



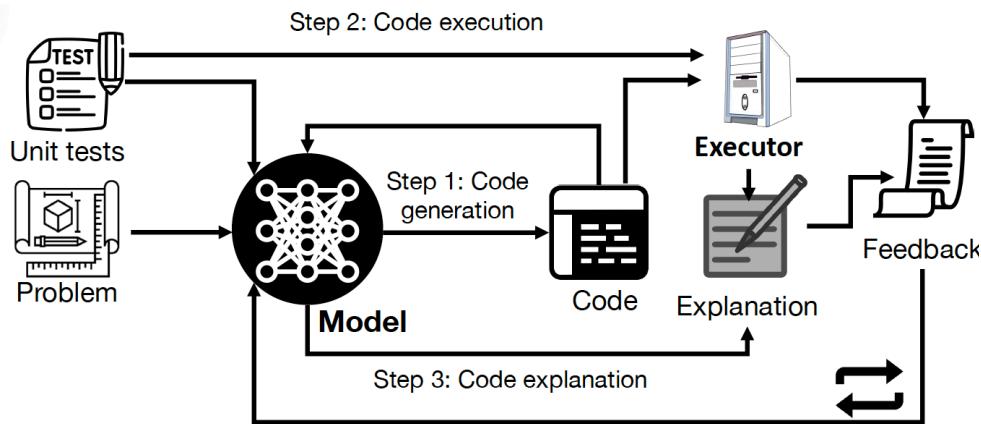
Q: Can we learn to generate fake stuff?

5



# Self-debugging language models

## Teaching Large Language Models to Self-Debug



<pre>C++</pre> <pre>int remainder_7_large_numbers ( string num ) {     int series [ ] = {         1, 3, 2, -1, -3, -2 };     int series_index = 0;     int result = 0;     for ( int i = num . size ( ) - 1;     i &gt;= 0; i -- ) {         int digit = num [ i ] - '0';         result += digit * series [             series_index ];         series_index = ( series_index +             1 ) % 6;         result %= 7;     }     if ( result &lt; 0 ) result = (         result + 7 ) % 7;     return result; }</pre>	<pre>Original Python</pre> <pre>def remainder_7_large_numbers(num):     series = [1, 3, 2, -1, -3, -2]     series_index = 0     result = 0     for i in range((len(num) - 1), -1, -1):         digit = (num[i] - '0')         result += (digit * series[series_index])         series_index = ((series_index + 1) % 6)         result %= 7     if (result &lt; 0):         result = ((result + 7) % 7)     return result</pre>
---	--

### Self-debugging with UT feedback

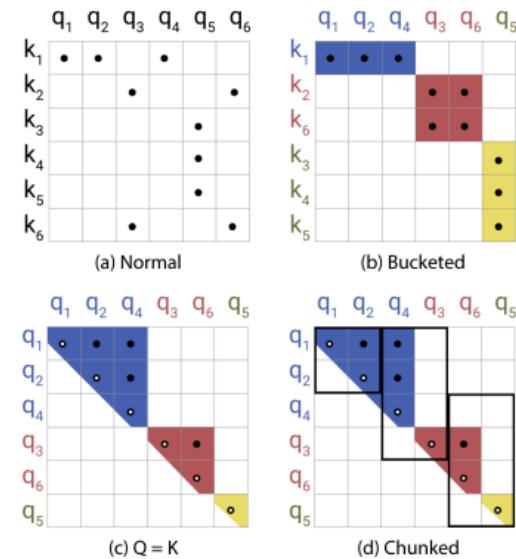
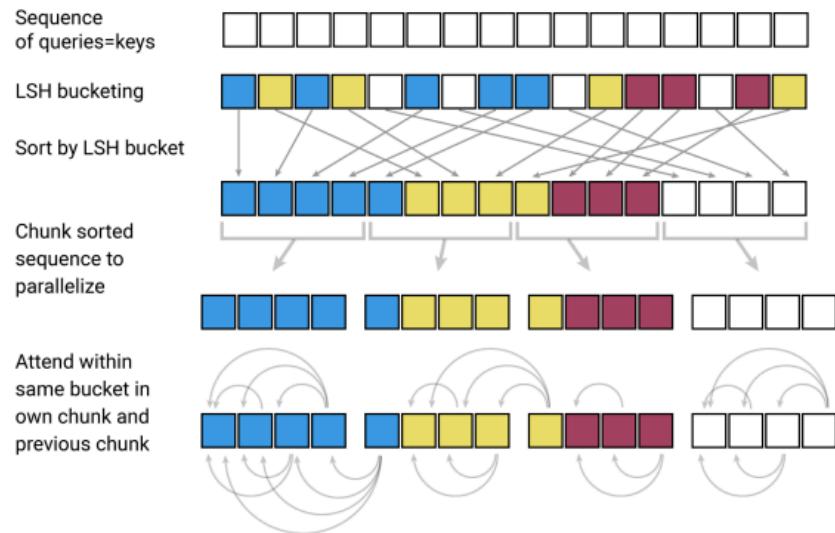
The Python translation does not do the same thing as the C++ code. These are the results of one failed unit test that tests whether the Python translation's outputs match the C++ program's outputs:

```
Failed: assert remainder_7_large_numbers('K') == 6
Actual Result: Python runtime error: Traceback (most recent call last):
  File <filename>, line 29, in <module>
    print(remainder_7_large_numbers('K'))
  File <filename>, line 22, in remainder_7_large_numbers
    digit = (num[i] - '0')
TypeError: unsupported operand type(s) for -: 'str' and 'str'
Correct the Python translation.
```

```
def remainder_7_large_numbers(num):
    series = [1, 3, 2, -1, -3, -2]
    series_index = 0
    result = 0
    for i in range((len(num) - 1), -1, -1):
        digit = (ord(num[i]) - ord('0'))
        result += (digit * series[series_index])
        series_index = ((series_index + 1) % 6)
        result %= 7
    if (result < 0):
        result = ((result + 7) % 7)
    return result
```

Figure 9: An example where SELF-DEBUGGING with unit test feedback fixes the code translation error, while the simple feedback fails.

# Efficient Attention Computation



Reduce computation from  $O(n^2)$  to  $O(n)$

# The Learning Problem

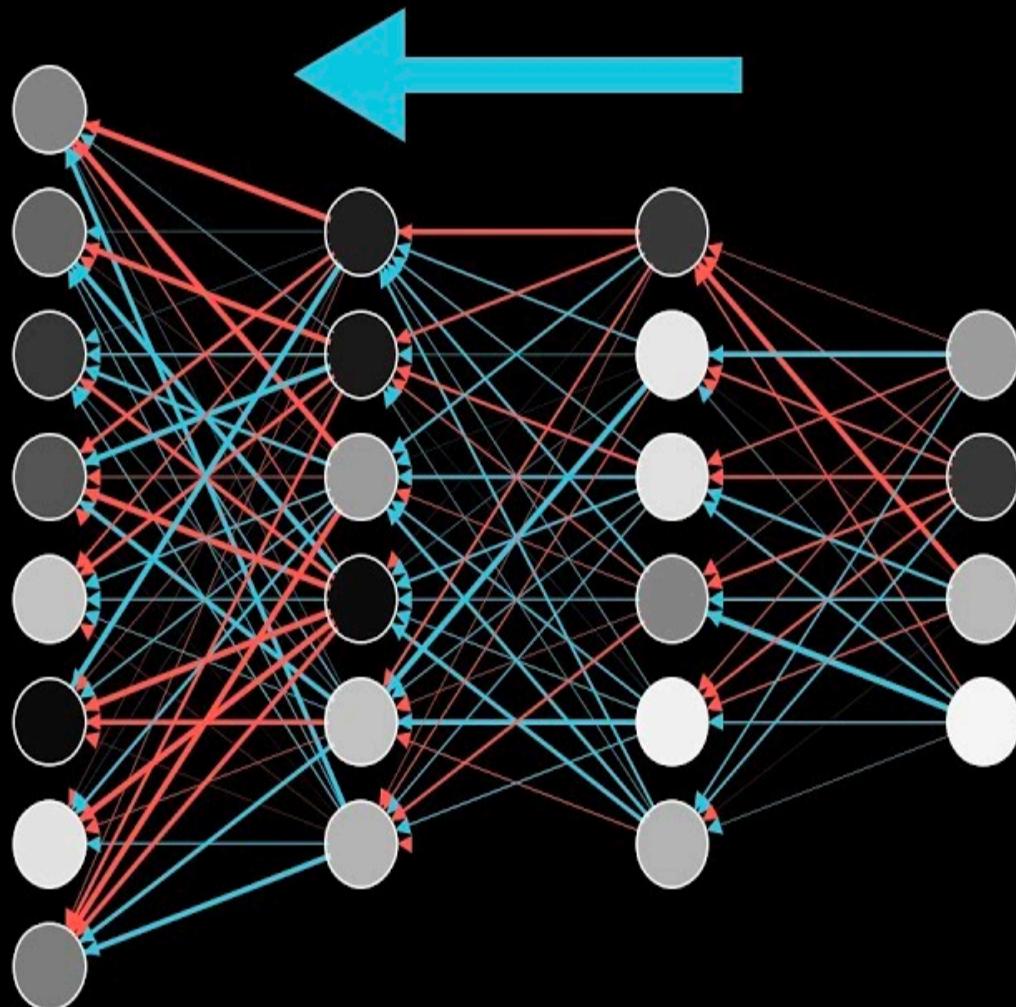
1. Data:  $(x_i, y_i)_{i=1}^n$ ,  $x_i \in \mathbb{R}^d$ ,  $y_i \in \mathbb{R}^K$
2. Model: neural networks with trainable weight matrices  $W_1, W_2, \dots, W_L, W_{\text{out}}$
3. Loss function depending on the task

Last lecture

4. Optimization

This lecture

# Backpropagation



# Chain Rule

- Suppose  $x \rightarrow u \rightarrow y$

- Then  $\frac{dy}{dx} = \frac{dy}{du} \frac{du}{dx}$

# Recall Stochastic Gradient Descent

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

$$\mathcal{L}(w) = \ell(y, f_w(x))$$

5. Gradient via chain rule

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

Derivative of Loss

Gradient of Model

HOW?

# Loss Functions

- Square loss:  $\ell(y, f(x)) = \frac{1}{2}(f(x) - y)^2$   
 $\ell'(y, f(x)) = (f(x) - y)$
  - Cross entropy loss
- $$\ell(y, f(x)) = - \sum_{i \in [k]} y_i \log(p_i), \quad \text{where} \quad p_i = \frac{e^{f_i(x)}}{\sum_{j=1}^n e^{f_j(x)}}$$

# Break

# Recall Stochastic Gradient Descent

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

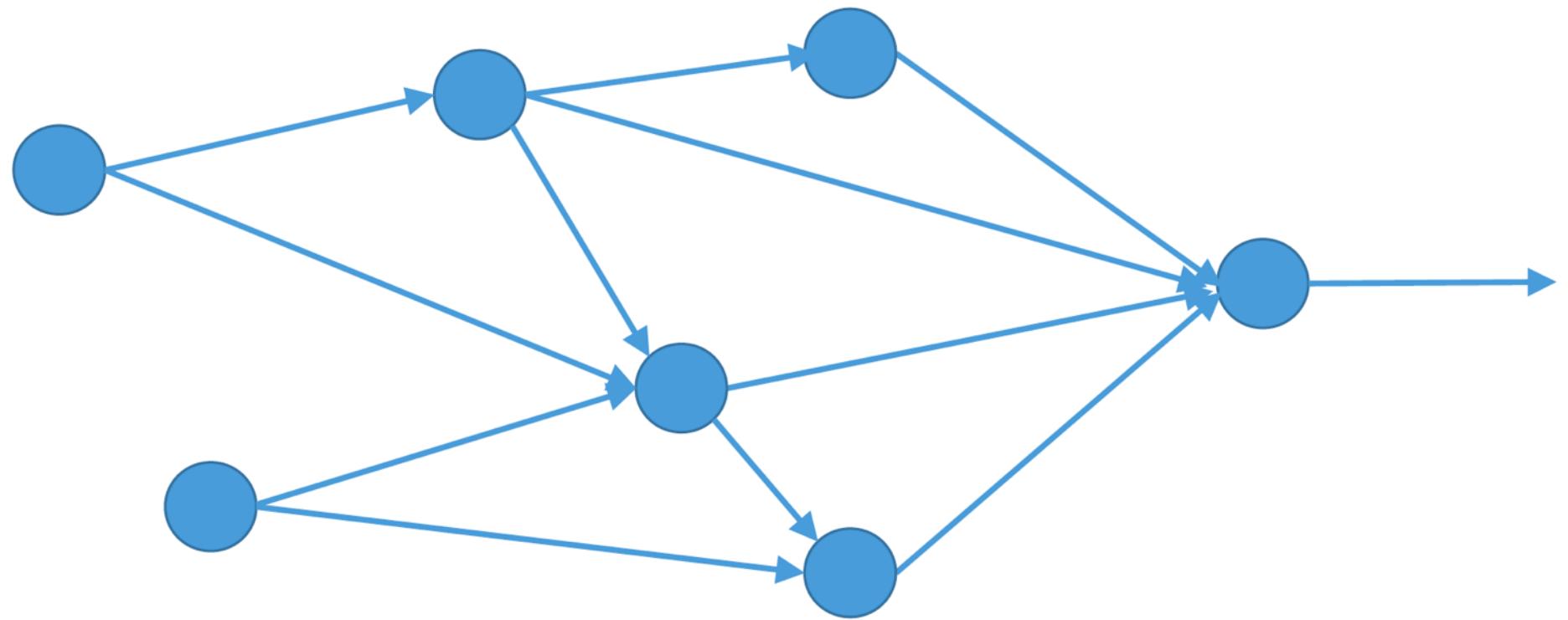
$$\mathcal{L}(w) = \ell(y, f_w(x))$$

5. Gradient via chain rule

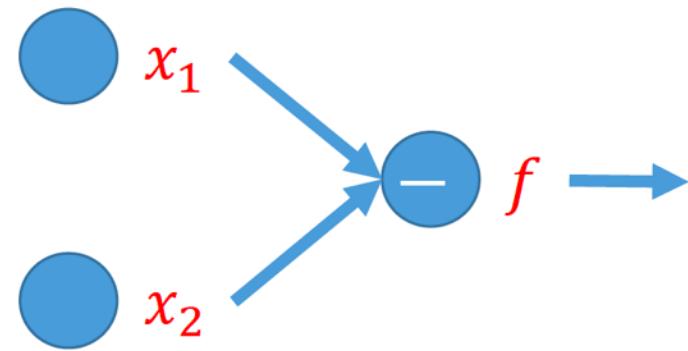
$$\nabla \mathcal{L}(w) = \ell'(y, f_w(x)) \times \nabla f_w(x)$$



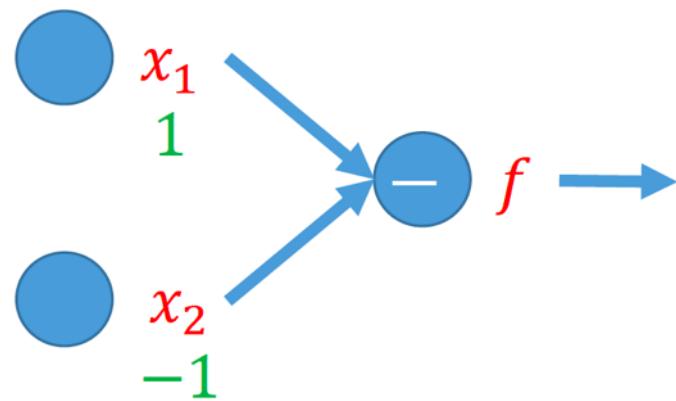
# How to differentiate a neural net?



**Answer:** *Backpropagate*

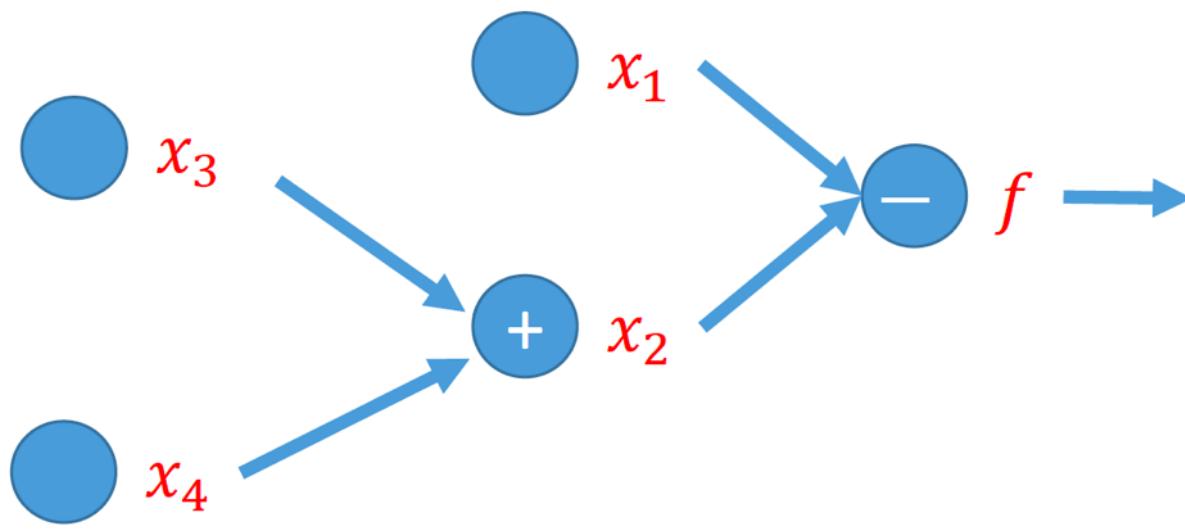


Function:  $f = x_1 - x_2$

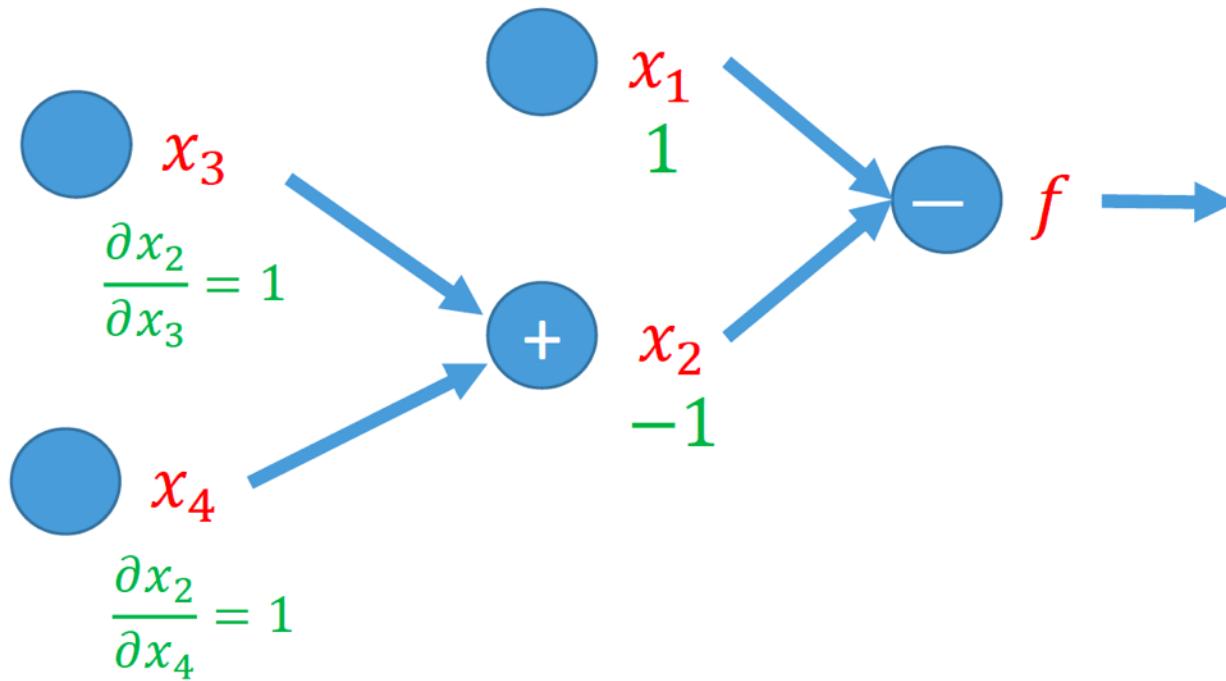


Function:  $f = x_1 - x_2$

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

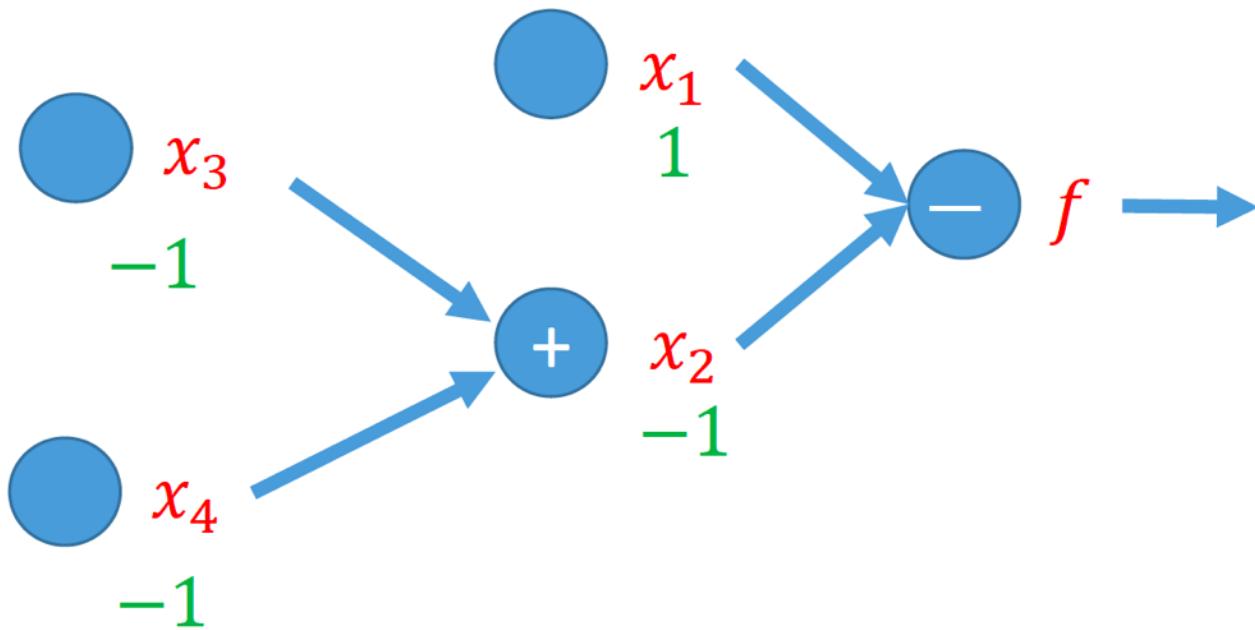


Function:  $f = x_1 - x_2 = x_1 - (x_3 + x_4)$



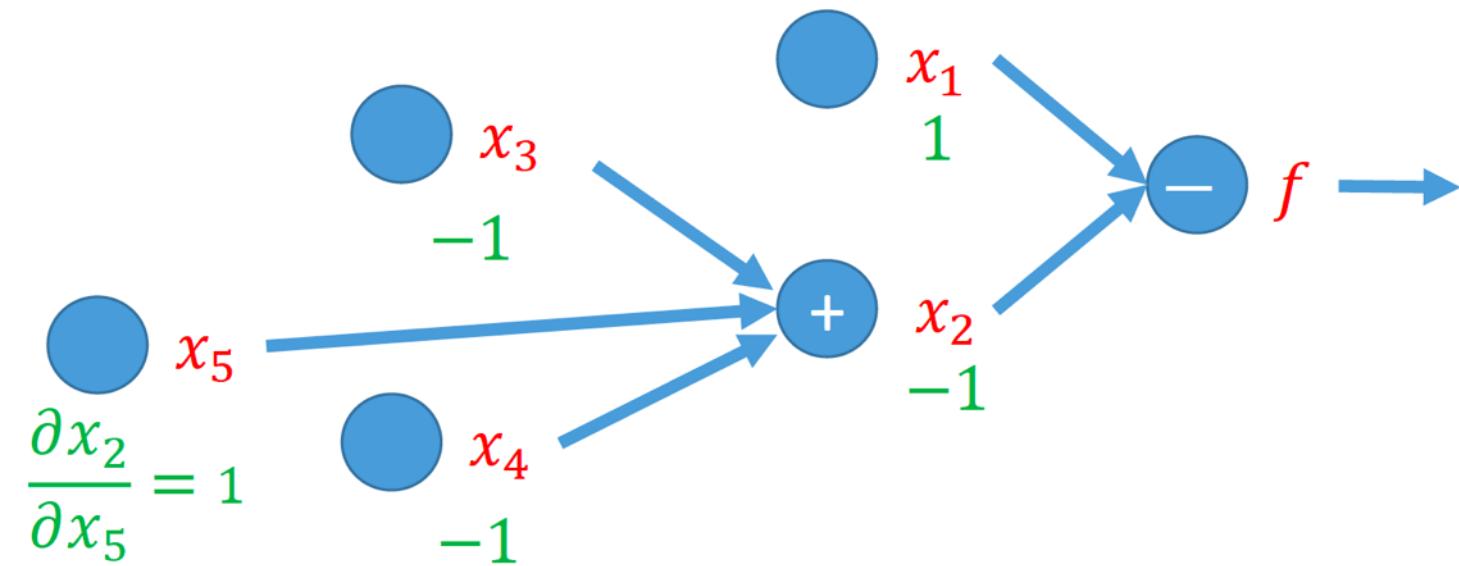
Function:  $f = x_1 - x_2 = x_1 - (x_3 + x_4)$

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



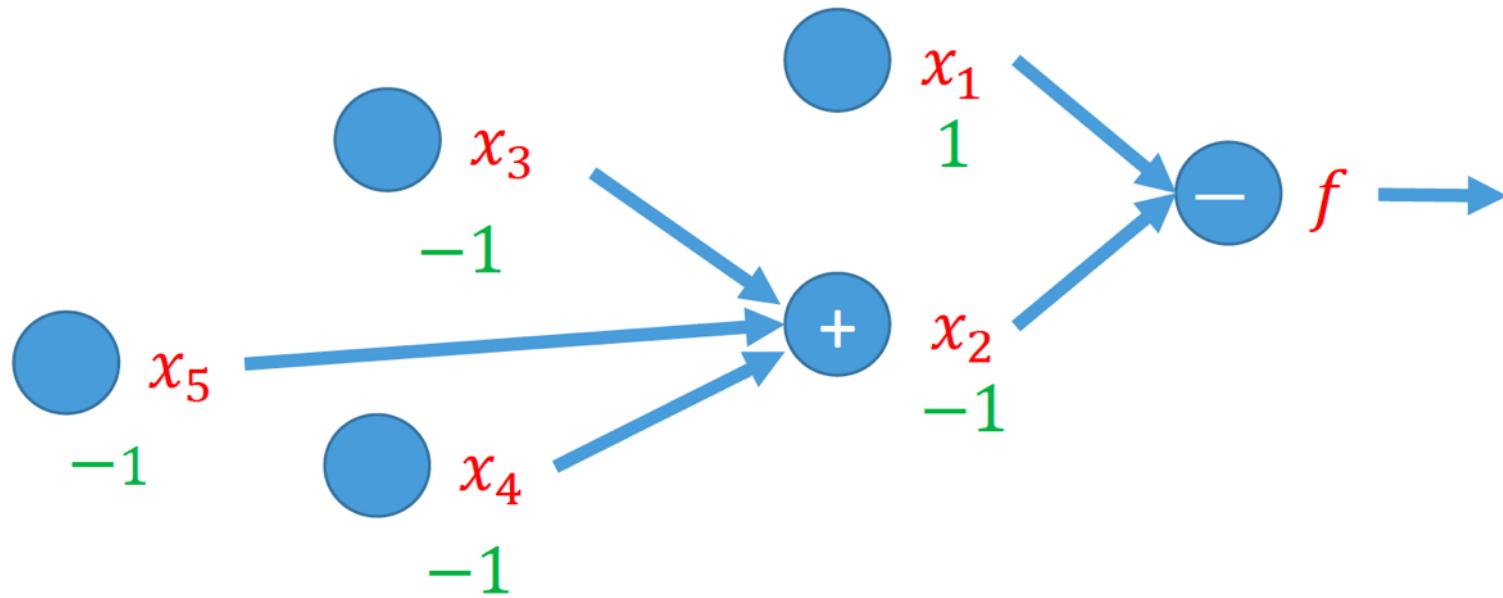
Function:  $f = x_1 - x_2 = x_1 - (x_3 + x_4)$

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



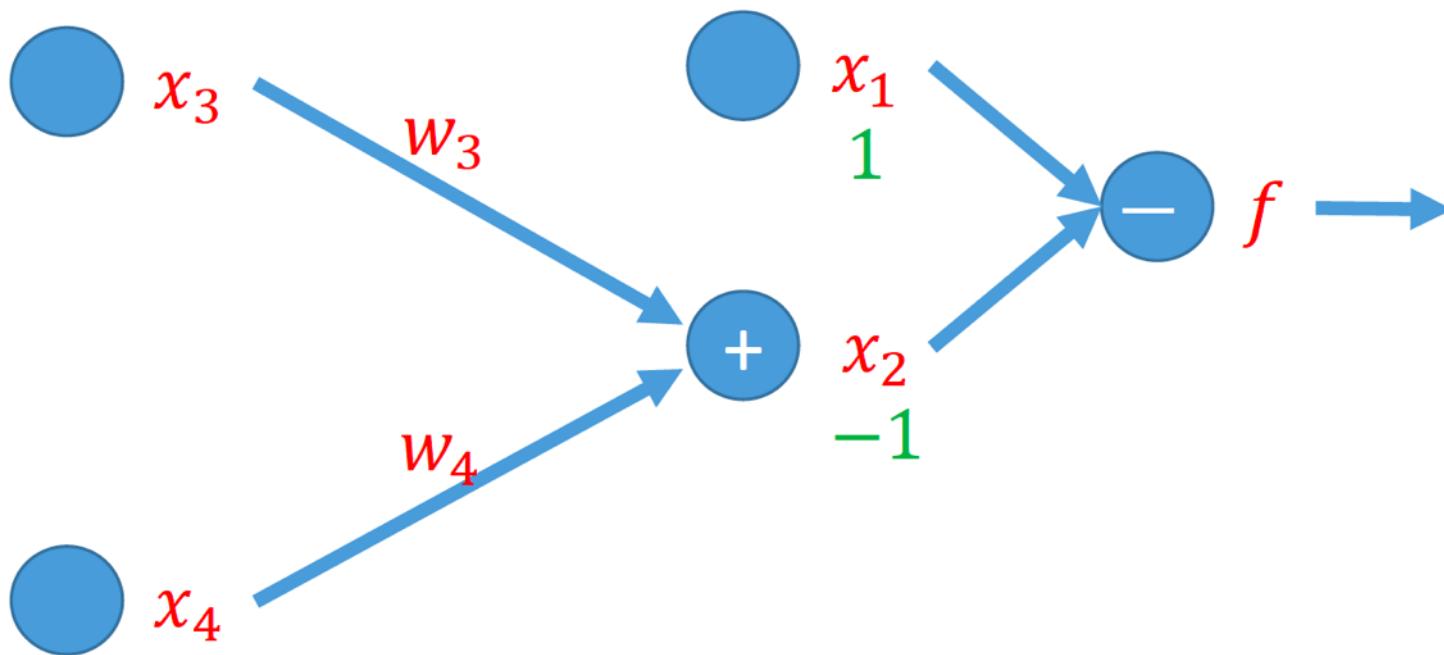
Function:  $f = x_1 - x_2 = x_1 - (x_3 + x_5 + x_4)$

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

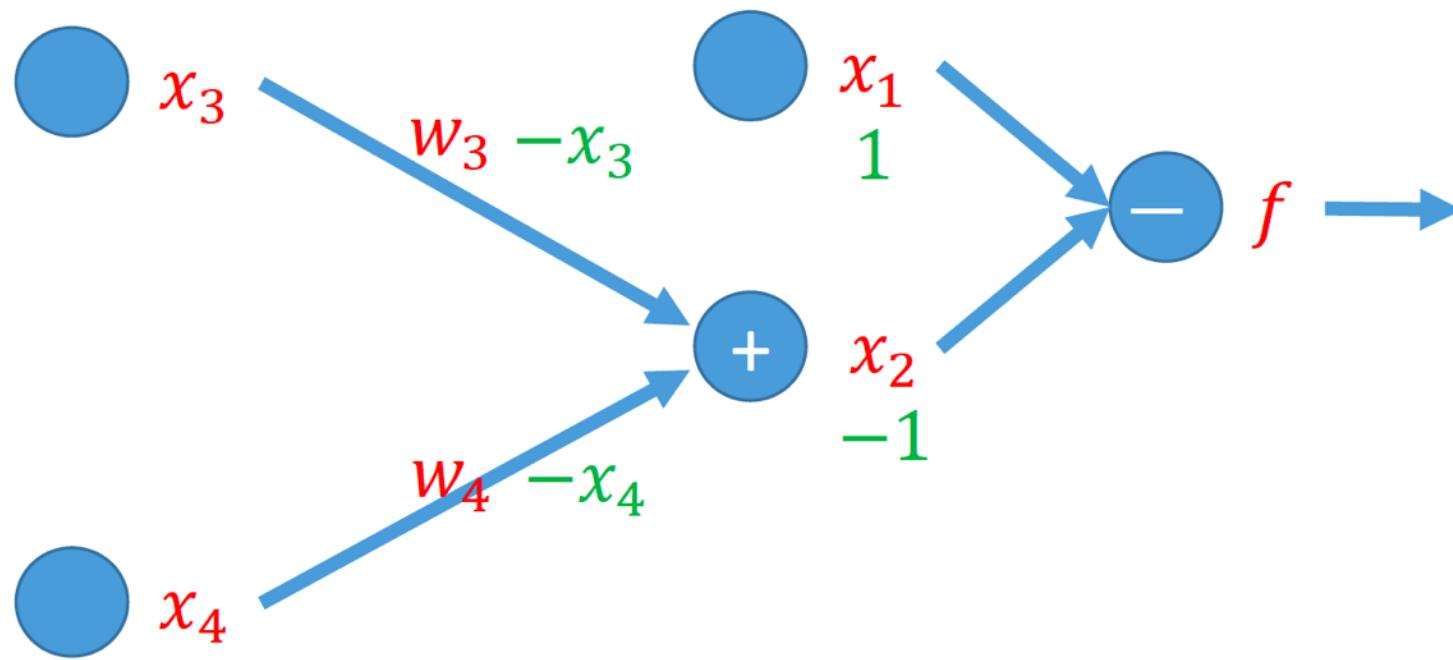


Function:  $f = x_1 - x_2 = x_1 - (x_3 + x_5 + x_4)$

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

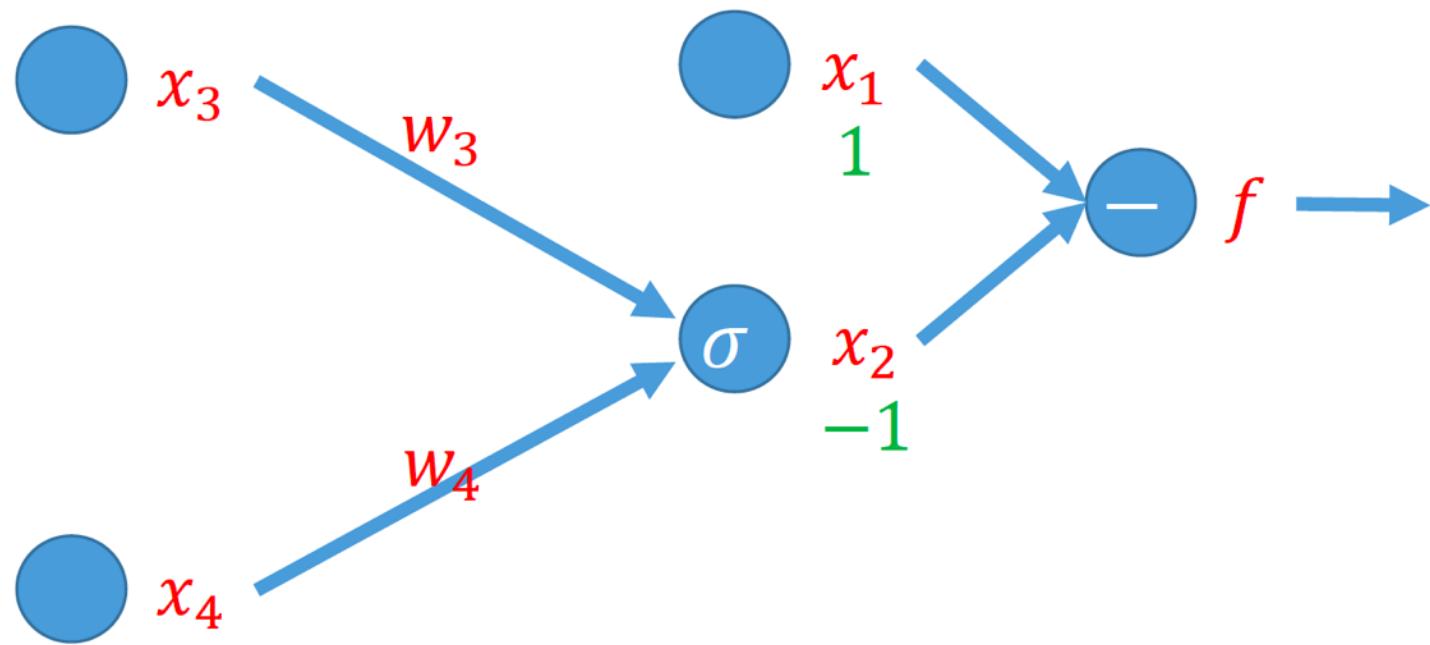


Function:  $f = x_1 - x_2 = x_1 - (w_3 x_3 + w_4 x_4)$

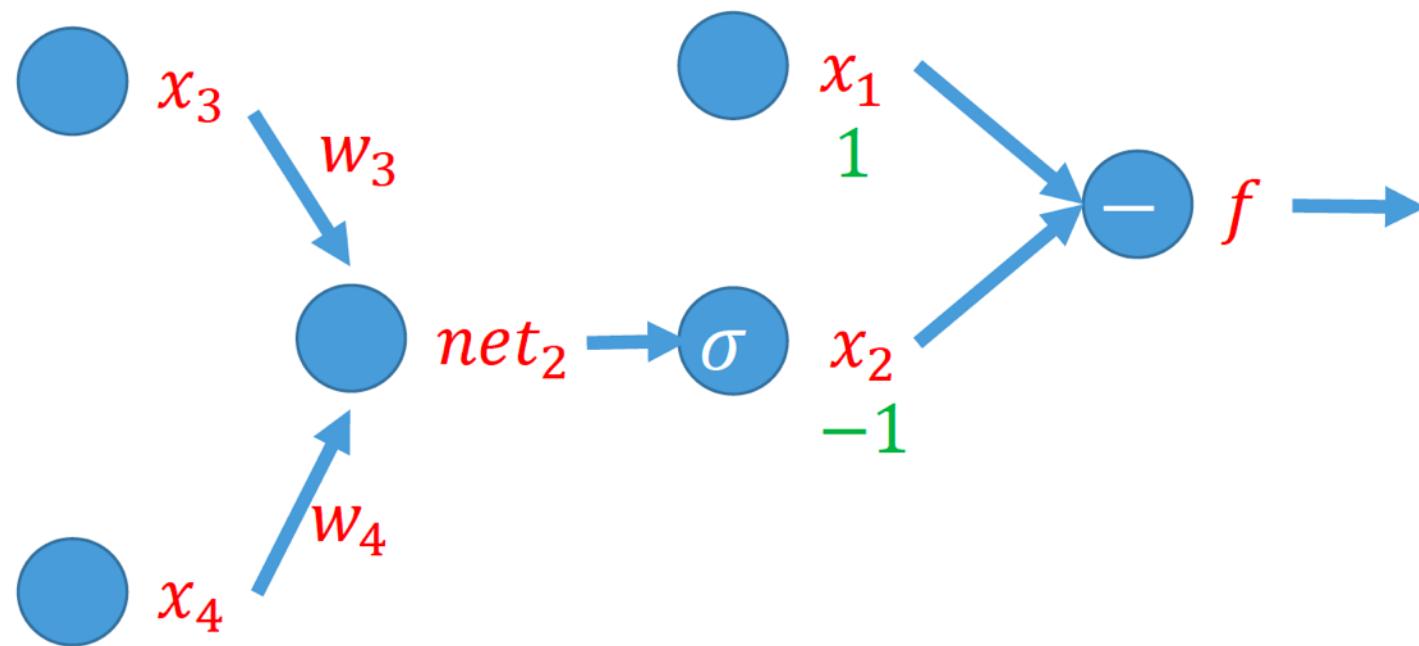


Function:  $f = x_1 - x_2 = x_1 - (w_3 x_3 + w_4 x_4)$

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

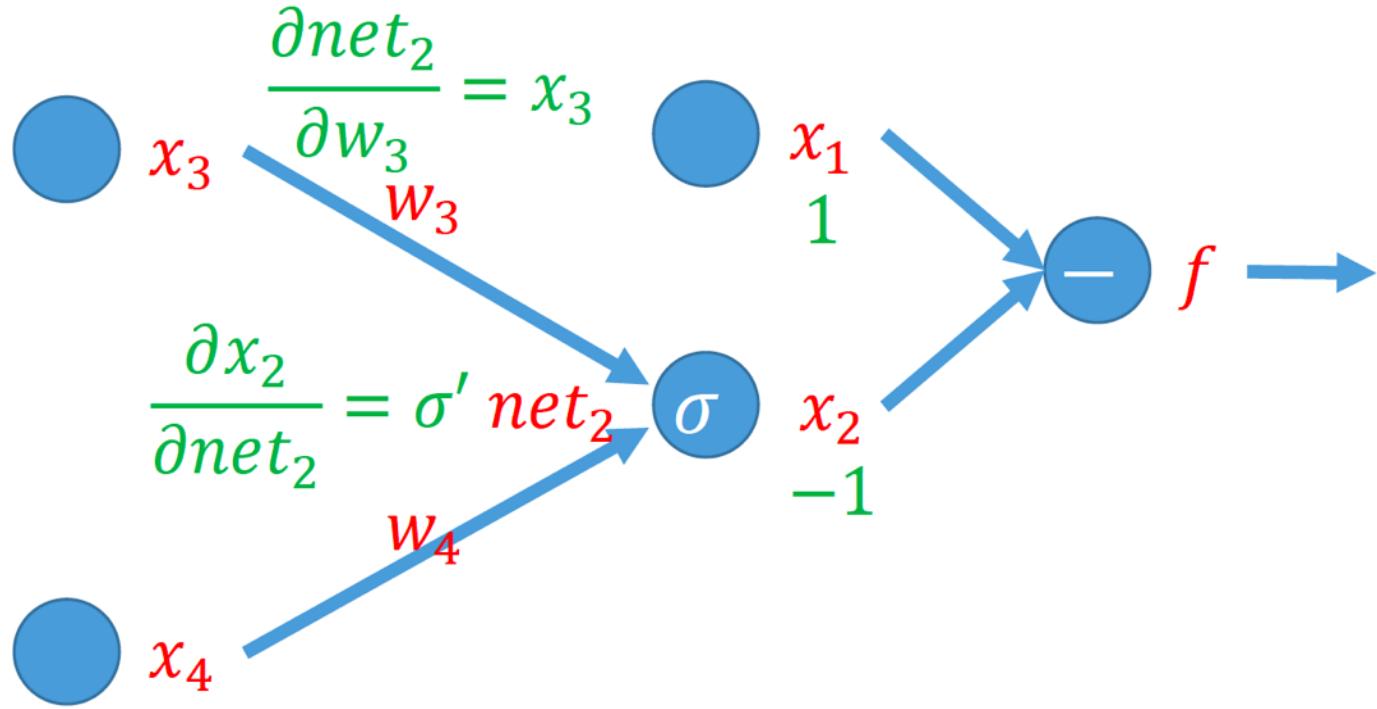


Function:  $f = x_1 - x_2 = x_1 - \sigma(w_3x_3 + w_4x_4)$



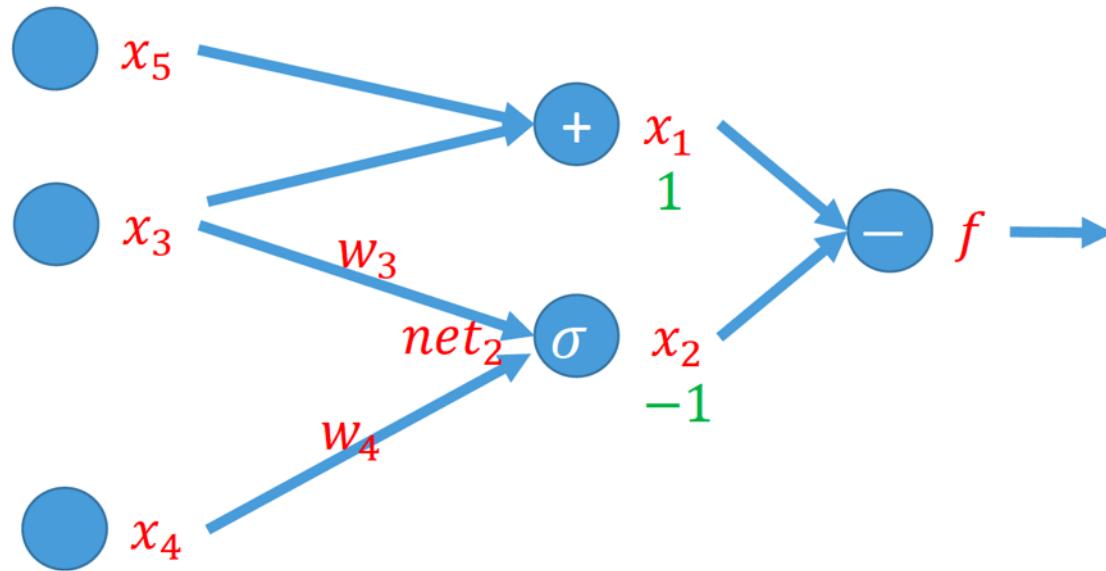
Function:  $f = x_1 - x_2 = x_1 - \sigma(w_3x_3 + w_4x_4)$

Let  $net_2 = w_3x_3 + w_4x_4$

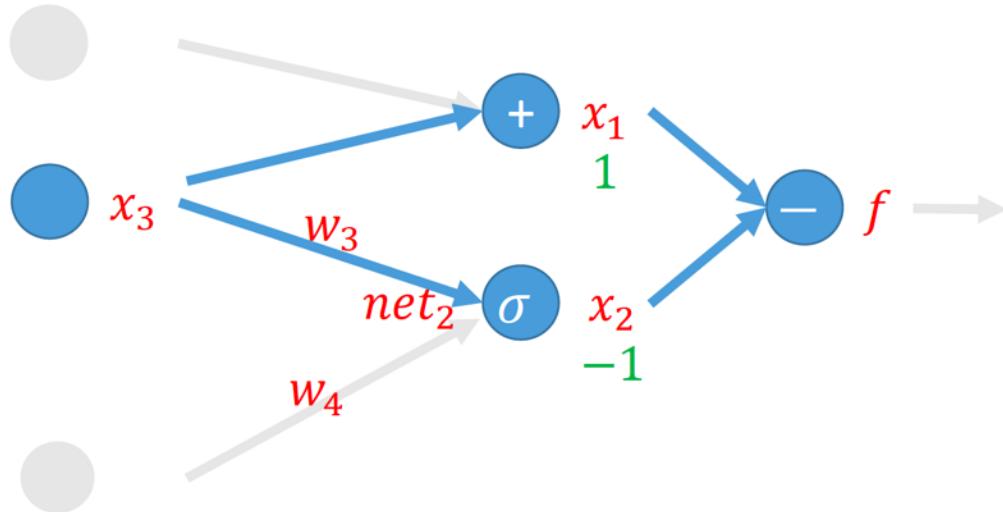


Function:  $f = x_1 - x_2 = x_1 - \sigma(w_3 x_3 + w_4 x_4)$

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

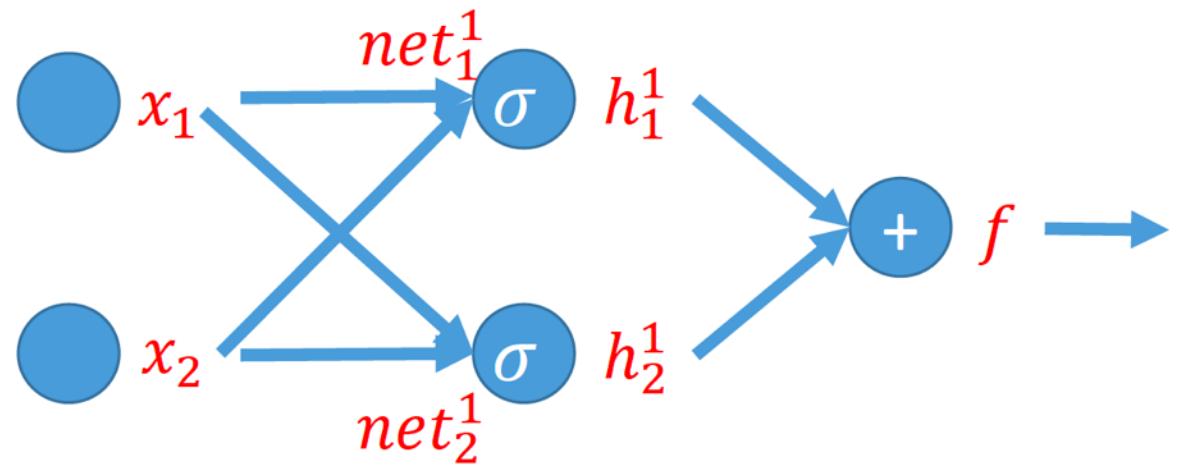


Function:  $f = x_1 - x_2 = (x_3 + x_5) - \sigma(w_3 x_3 + w_4 x_4)$

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

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

Sub-gradient  
used at  $x = 0$

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

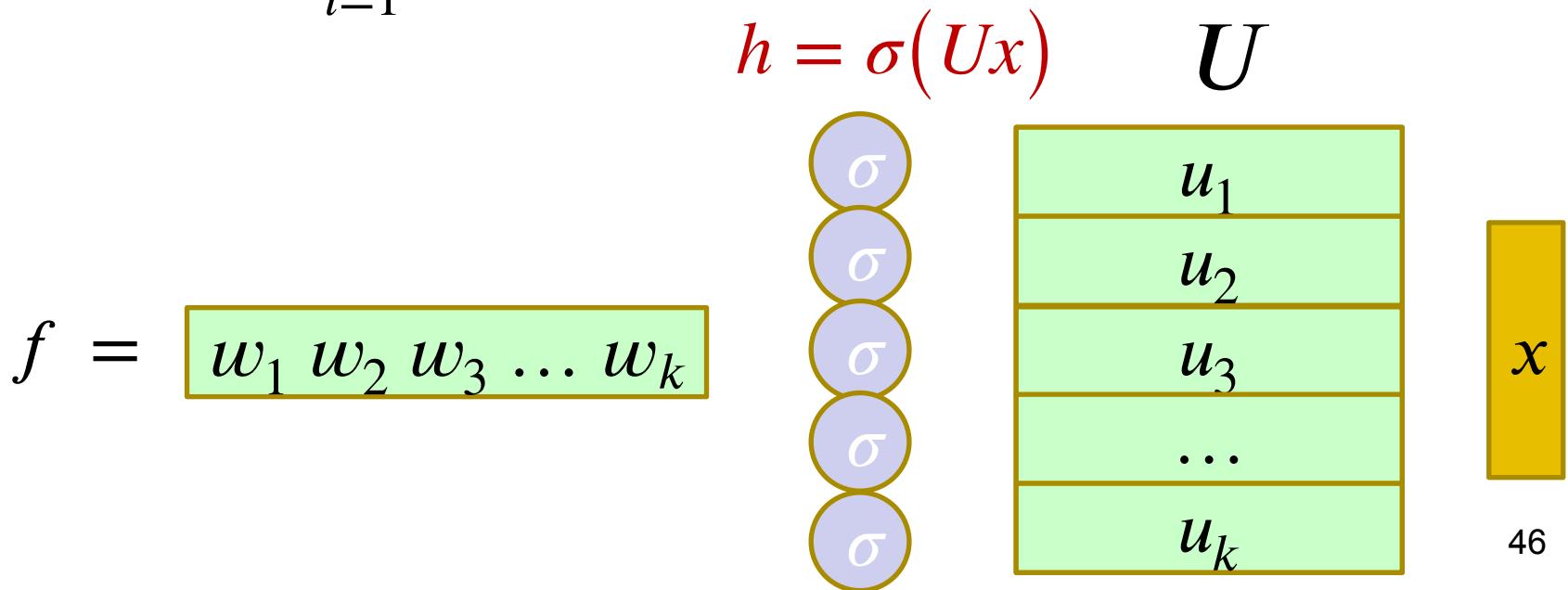
$$\sigma'(x) = \sigma(x) \cdot (1 - \sigma(x))$$

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

# 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

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

# One-hidden layer network

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

Overall size  
 $k \times d$

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

$k \times 1$  vector with  
entries  $w_i \sigma'(u_i^\top x)$

$1 \times d$  input features

- $\odot$  is Hadamard product (entry-wise multiplication)<sup>48</sup>

# One-hidden layer network

- $\frac{\partial f}{\partial w} = \sigma(Ux)$
- $\frac{\partial f}{\partial U} = (w \odot \sigma'(Ux))x^\top$
- Consider  $\mathcal{L}(\theta) = \frac{1}{2} (w^\top \sigma(Ux) - y)^2$   
 $\frac{\partial \mathcal{L}}{\partial w} = (w^\top \sigma(Ux) - y) \sigma(Ux)$   
 $\frac{\partial \mathcal{L}}{\partial w} = (w^\top \sigma(Ux) - y) (w \odot \sigma'(Ux))x^\top$

# Training a one-hidden layer net

Initialize  $w_0, U_0$

Learning rate  $\eta > 0$ , duration  $END$

for  $0 \leq t \leq END - 1$

1. Randomly pick example  $(x, y)$  from training data
2. Update weights

- $w_{t+1} = w_t - \eta (w^\top \sigma(Ux) - y) \sigma(Ux)$
- $U_{t+1} = U_t - \eta (w^\top \sigma(Ux) - y) (w \odot \sigma'(Ux)) x^\top$

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

# Acknowledgements

- A lot of content from online resources
  - Yingyu Liang's slides <https://www.cs.princeton.edu/courses/archive/spring16/cos495/>
  - Goodfellow's slides <https://www.deeplearningbook.org/>