Lecture 4: Neural Networks and Backpropagation
Administrative: Assignment 1

**Assignment 1** due **Wednesday April 17, 11:59pm**

If using Google Cloud, **you don’t need GPUs** for this assignment!

We will distribute Google Cloud coupons by this weekend
Administrative: Alternate Midterm Time

If you need to request an alternate midterm time:

See Piazza for form, fill it out by 4/25 (two weeks from today)
Administrative: Project Proposal

Project proposal due 4/24 (Two weeks from yesterday)
Administrative: Discussion Section

Discussion section tomorrow (1:20pm in Gates B03):
How to pick a project / How to read a paper
Where we are...

\[ s = f(x; W) = Wx \]  \hspace{1cm} \text{Linear score function}

\[ L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1) \]  \hspace{1cm} \text{SVM loss (or softmax)}

\[ L = \frac{1}{N} \sum_{i=1}^{N} L_i + \lambda \sum_k W_k^2 \]  \hspace{1cm} \text{data loss + regularization}

How to find the best W?
Finding the best W: Optimize with Gradient Descent

# Vanilla Gradient Descent

```python
while True:
    weights_grad = evaluate_gradient(loss_fun, data, weights)
    weights += - step_size * weights_grad # perform parameter update
```
Gradient descent

\[ \frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x + h) - f(x)}{h} \]

Numerical gradient: slow :, approximate (:, easy to write :)
Analytic gradient: fast :), exact :), error-prone :(

In practice: Derive analytic gradient, check your implementation with numerical gradient
Problem: Linear Classifiers are not very powerful

**Visual Viewpoint**

Linear classifiers learn one template per class

**Geometric Viewpoint**

Linear classifiers can only draw linear decision boundaries
One Solution: Feature Transformation

\[ f(x, y) = (r(x, y), \theta(x, y)) \]

Transform data with a cleverly chosen feature transform \( f \), then apply linear classifier

Color Histogram

Histogram of Oriented Gradients (HoG)
Image features vs ConvNets

Feature Extraction

10 numbers giving scores for classes

training

10 numbers giving scores for classes

training

Today: Neural Networks
Neural networks: without the brain stuff

(Before) Linear score function:

\[ f = Wx \]

\[ x \in \mathbb{R}^D, W \in \mathbb{R}^{C \times D} \]
Neural networks: without the brain stuff

(Before) Linear score function:  
\[ f = Wx \]

(Now) 2-layer Neural Network  
\[ f = W_2 \max(0, W_1 x) \]

\[ x \in \mathbb{R}^D, W_1 \in \mathbb{R}^{H \times D}, W_2 \in \mathbb{R}^{C \times H} \]

(In practice we will usually add a learnable bias at each layer as well)
Neural networks: without the brain stuff

(Before) Linear score function: \[ f = Wx \]

(Now) 2-layer Neural Network: \[ f = W_2 \max(0, W_1 x) \]

\[ x \in \mathbb{R}^D, W_1 \in \mathbb{R}^{H \times D}, W_2 \in \mathbb{R}^{C \times H} \]

“Neural Network” is a very broad term; these are more accurately called “fully-connected networks” or sometimes “multi-layer perceptrons” (MLP)

(In practice we will usually add a learnable bias at each layer as well)
Neural networks: without the brain stuff

(Before) Linear score function:

\[ f = W x \]

(Now) 2-layer Neural Network or 3-layer Neural Network

\[ f = W_2 \max(0, W_1 x) \]

or

\[ f = W_3 \max(0, W_2 \max(0, W_1 x)) \]

\[ x \in \mathbb{R}^D, W_1 \in \mathbb{R}^{H_1 \times D}, W_2 \in \mathbb{R}^{H_2 \times H_1}, W_3 \in \mathbb{R}^{C \times H_2} \]

(In practice we will usually add a learnable bias at each layer as well)
Neural networks: without the brain stuff

(Before) Linear score function: \( f = Wx \)

(Now) 2-layer Neural Network: \( f = W_2 \max(0, W_1x) \)

\[ x \in \mathbb{R}^D, W_1 \in \mathbb{R}^{H \times D}, W_2 \in \mathbb{R}^{C \times H} \]
Neural networks: without the brain stuff

(Before) Linear score function: $f = Wx$

(Now) 2-layer Neural Network $f = W_2 \max(0, W_1 x)$
Neural networks: without the brain stuff

(Before) Linear score function: \[ f = W x \]

(Now) 2-layer Neural Network

\[ f = W_2 \max(0, W_1 x) \]

The function \( \max(0, z) \) is called the activation function.

Q: What if we try to build a neural network without one?

\[ f = W_2 W_1 x \]
Neural networks: without the brain stuff

(Before) Linear score function: \[ f = Wx \]

(Now) 2-layer Neural Network \[ f = W_2 \max(0, W_1 x) \]

The function \( \max(0, z) \) is called the activation function.

Q: What if we try to build a neural network without one?

\[ f = W_2 W_1 x \quad W_3 = W_2 W_1 \in \mathbb{R}^{C \times H}, \quad f = W_3 x \]

A: We end up with a linear classifier again!
Activation functions

**Sigmoid**
\[ \sigma(x) = \frac{1}{1+e^{-x}} \]

**tanh**
\[ \tanh(x) \]

**ReLU**
\[ \max(0, x) \]

**Leaky ReLU**
\[ \max(0.1x, x) \]

**Maxout**
\[ \max(w_1^T x + b_1, w_2^T x + b_2) \]

**ELU**
\[ \begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases} \]
Activation functions

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

**tanh**
$$\tanh(x)$$

**ReLU**
$$\max(0, x)$$

**Leaky ReLU**
$$\max(0.1x, x)$$

**Maxout**
$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

**ELU**
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$

ReLU is a good default choice for most problems.
Neural networks: Architectures

"2-layer Neural Net", or "1-hidden-layer Neural Net"

"Fully-connected" layers

"3-layer Neural Net", or "2-hidden-layer Neural Net"
Example feed-forward computation of a neural network

```python
# forward-pass of a 3-layer neural network:
f = lambda x: 1.0/(1.0 + np.exp(-x))  # activation function (use sigmoid)
x = np.random.randn(3, 1)  # random input vector of three numbers (3x1)
h1 = f(np.dot(W1, x) + b1)  # calculate first hidden layer activations (4x1)
h2 = f(np.dot(W2, h1) + b2)  # calculate second hidden layer activations (4x1)
out = np.dot(W3, h2) + b3  # output neuron (1x1)
```
Full implementation of training a 2-layer Neural Network needs ~20 lines:

```python
import numpy as np
from numpy.random import randn

N, D_in, H, D_out = 64, 1000, 100, 10
x, y = randn(N, D_in), randn(N, D_out)
w1, w2 = randn(D_in, H), randn(H, D_out)

for t in range(2000):
    h = 1 / (1 + np.exp(-x.dot(w1)))
y_pred = h.dot(w2)
loss = np.square(y_pred - y).sum()
print(t, loss)

grad_y_pred = 2.0 * (y_pred - y)
grad_w2 = h.T.dot(grad_y_pred)
grad_h = grad_y_pred.dot(w2.T)
grad_w1 = x.T.dot(grad_h * h * (1 - h))
w1 -= 1e-4 * grad_w1
w2 -= 1e-4 * grad_w2
```
Setting the number of layers and their sizes

3 hidden neurons

6 hidden neurons

20 hidden neurons

more neurons = more capacity
Do not use size of neural network as a regularizer. Use stronger regularization instead:

\( \lambda = 0.001 \)  \hspace{1cm} \( \lambda = 0.01 \)  \hspace{1cm} \( \lambda = 0.1 \)

(Web demo with ConvNetJS: [http://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html](http://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html))
Impulses carried toward cell body

dendrite

axon

presynaptic terminal

Impulses carried away from cell body

cell body

This image by Felipe Perucho is licensed under CC-BY 3.0
Impulses carried toward cell body

Impulses carried away from cell body

This image by Felipe Perucho is licensed under CC-BY 3.0
Impulses carried toward cell body

sigmoid activation function
\[
\frac{1}{1 + e^{-x}}
\]

Impulses carried away from cell body

This image by Felipe Perucho is licensed under CC-BY 3.0
Impulses carried toward cell body

Impulses carried away from cell body

class Neuron:
    # ...
    def neuron_tick(inputs):
        """ assume inputs and weights are 1-D numpy arrays and bias is a number """
        cell_body_sum = np.sum(inputs * self.weights) + self.bias
        firing_rate = 1.0 / (1.0 + math.exp(-cell_body_sum))  # sigmoid activation function
        return firing_rate

This image by Felipe Perucho is licensed under CC-BY 3.0
Biological Neurons: Complex connectivity patterns

Neurons in a neural network: Organized into regular layers for computational efficiency
Biological Neurons: Complex connectivity patterns

But neural networks with random connections can work too!

Xie et al, “Exploring Randomly Wired Neural Networks for Image Recognition”, arXiv 2019
Be very careful with your brain analogies!

**Biological Neurons:**
- Many different types
- Dendrites can perform complex non-linear computations
- Synapses are not a single weight but a complex non-linear dynamical system
- Rate code may not be adequate

[Dendritic Computation. London and Hausser]
Problem: How to compute gradients?

\[ s = f(x; W_1, W_2) = W_2 \max(0, W_1 x) \]  \hspace{1cm} \text{Nonlinear score function}

\[ L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1) \] \hspace{1cm} \text{SVM Loss on predictions}

\[ R(W) = \sum_k W_k^2 \] \hspace{1cm} \text{Regularization}

\[ L = \frac{1}{N} \sum_{i=1}^{N} L_i + \lambda R(W_1) + \lambda R(W_2) \] \hspace{1cm} \text{Total loss: data loss + regularization}

If we can compute \( \frac{\partial L}{\partial W_1}, \frac{\partial L}{\partial W_2} \) then we can learn \( W_1 \) and \( W_2 \).
(Bad) Idea: Derive $\nabla_W L$ on paper

$s = f(x; W) = Wx$

$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$

$= \sum_{j \neq y_i} \max(0, W_{j,:} \cdot x + W_{y_i,:} \cdot x + 1)$

$L = \frac{1}{N} \sum_{i=1}^{N} L_i + \lambda \sum_k W_k^2$

$= \frac{1}{N} \sum_{i=1}^{N} \sum_{j \neq y_i} \max(0, W_{j,:} \cdot x + W_{y_i,:} \cdot x + 1) + \lambda \sum_k W_k^2$

$\nabla_W L = \nabla_W \left( \frac{1}{N} \sum_{i=1}^{N} \sum_{j \neq y_i} \max(0, W_{j,:} \cdot x + W_{y_i,:} \cdot x + 1) + \lambda \sum_k W_k^2 \right)$

**Problem:** Very tedious: Lots of matrix calculus, need lots of paper

**Problem:** What if we want to change loss? E.g. use softmax instead of SVM? Need to re-derive from scratch =(

**Problem:** Not feasible for very complex models!
Better Idea: Computational graphs + Backpropagation

\[ f = Wx \]

\[ L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1) \]
Convolutional network (AlexNet)

input image

weights

loss

Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.
Neural Turing Machine

Figure reproduced with permission from a Twitter post by Andrej Karpathy.
Neural Turing Machine

Figure reproduced with permission from a Twitter post by Andrej Karpathy.
Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]
Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]
Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)
Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q \]

Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)
Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q \]

Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)
Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]
e.g. \( x = -2, \ y = 5, \ z = -4 \)

\[
q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1
\]

\[
f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q
\]

Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)
Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]

E.g. \( x = -2, y = 5, z = -4 \)

\[
q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1
\]

\[
f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q
\]

Want:

\[
\frac{\partial f}{\partial x}, \quad \frac{\partial f}{\partial y}, \quad \frac{\partial f}{\partial z}
\]
Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)

\[
q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1
\]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q \]

Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)
Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q \]

Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)
Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)

\[
q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1
\]

\[
f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q
\]

Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)
Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q \]

Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)

Chain rule:

\[ \frac{\partial f}{\partial y} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial y} \]

Upstream gradient

Local gradient
Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)

\[ q = x + y \]
\[ \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \]
\[ \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q \]

Want: \( \frac{\partial f}{\partial x}, \quad \frac{\partial f}{\partial y}, \quad \frac{\partial f}{\partial z} \)

Chain rule:

\[ \frac{\partial f}{\partial y} = \frac{\partial f}{\partial q} \cdot \frac{\partial q}{\partial y} \]

Upstream gradient

Local gradient
Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]
e.g. \( x = -2, \ y = 5, \ z = -4 \)

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q \]

Want: \( \frac{\partial f}{\partial x}, \ \frac{\partial f}{\partial y}, \ \frac{\partial f}{\partial z} \)

Chain rule:

\[
\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x}
\]
Backpropagation: a simple example

\[ f(x, y, z) = (x + y)z \]

e.g. \( x = -2, y = 5, z = -4 \)

\[ q = x + y \quad \frac{\partial q}{\partial x} = 1, \quad \frac{\partial q}{\partial y} = 1 \]

\[ f = qz \quad \frac{\partial f}{\partial q} = z, \quad \frac{\partial f}{\partial z} = q \]

Want: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)
The diagram illustrates the concept of the local gradient in the context of a function $f(x, y)$ that maps from $x$ and $y$ to $z$. The gradient of $f$ at a point is given by the vector of partial derivatives:

$$\nabla f = \left( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right)$$

The partial derivatives $\frac{\partial f}{\partial x}$ and $\frac{\partial f}{\partial y}$ represent the rate of change of $f$ in the direction of $x$ and $y$, respectively. The term “local gradient” emphasizes that the gradient describes the local behavior of the function $f$ at a given point.
"local gradient"

\[
\frac{\partial L}{\partial z}
\]

\[
\frac{\partial z}{\partial x}
\]

\[
\frac{\partial z}{\partial y}
\]
"Downstream gradients" 

\[ \frac{\partial L}{\partial x} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial x} \]

"local gradient"

"Upstream gradient"
"Local gradient"

\[
\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial x}
\]

"Downstream gradients"

\[
\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial y}
\]

"Upstream gradient"
"Downstream gradients"

\[ \frac{\partial L}{\partial x} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial x} \]

"local gradient"

\[ \frac{\partial z}{\partial x} \]

\[ \frac{\partial z}{\partial y} \]

\[ \frac{\partial L}{\partial z} \]

"Upstream gradient"
Another example:

\[ f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}} \]
Another example:

\[ f(w, x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}} \]
Another example:

\[ f(w, x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}} \]

\[
\begin{align*}
 f(x) &= e^x \\
f_a(x) &= ax
\end{align*}
\]

\[
\begin{align*}
 \frac{df}{dx} &= e^x \\
\frac{df}{dx} &= a
\end{align*}
\]

\[
\begin{align*}
 f(x) &= \frac{1}{x} \\
f_c(x) &= c + x
\end{align*}
\]

\[
\begin{align*}
 \frac{df}{dx} &= -\frac{1}{x^2} \\
\frac{df}{dx} &= 1
\end{align*}
\]
Another example: 

\[ f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}} \]

\[
\begin{align*}
  f(x) &= e^x \\
  f_a(x) &= ax
\end{align*}
\]

\[
\begin{align*}
  \frac{df}{dx} &= e^x \\
  \frac{df}{dx} &= a
\end{align*}
\]

\[
\begin{align*}
  f(x) &= \frac{1}{x} \\
  f_c(x) &= c + x
\end{align*}
\]

\[
\begin{align*}
  \frac{df}{dx} &= -\frac{1}{x^2} \\
  \frac{df}{dx} &= 1
\end{align*}
\]
Another example:

\[
f(w, x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}}
\]

For the function \( f(x) = e^x \):
\[
f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x
\]

For the function \( f(x) = ax \):
\[
f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a
\]

For the function \( f(x) = \frac{1}{x} \):
\[
f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -\frac{1}{x^2}
\]

For the function \( f(x) = c + x \):
\[
f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1
\]
Another example:

\[ f(w, x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}} \]

\[
\begin{align*}
  f(x) &= e^x \\
  f_a(x) &= ax \\
  f_c(x) &= c + x
\end{align*}
\]

- \[ \frac{df}{dx} = e^x \]
- \[ \frac{df}{dx} = a \]
- \[ \frac{df}{dx} = 1 \]
Another example:

\[ f(w, x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}} \]

\[ f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x \]

\[ f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a \]

\[ f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1 \]
Another example: \[ f(w, x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}} \]

\[
\begin{align*}
  f(x) &= e^x 
  \quad \rightarrow \quad \frac{df}{dx} = e^x \\
  f_a(x) &= ax 
  \quad \rightarrow \quad \frac{df}{dx} = a \\
  f_c(x) &= c + x 
  \quad \rightarrow \quad \frac{df}{dx} = 1 \\
  f(x) &= \frac{1}{x} 
  \quad \rightarrow \quad \frac{df}{dx} = -\frac{1}{x^2}
\end{align*}
\]
Another example:

\[ f(w, x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}} \]

\[ f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x \]
\[ f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a \]
\[ f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -\frac{1}{x^2} \]
\[ f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1 \]
Another example:

\[ f(w, x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}} \]

\[
\begin{align*}
f(x) &= e^x & \Rightarrow & & \frac{df}{dx} &= e^x \\
\frac{d}{dx} f_a(x) &= ax & \Rightarrow & & \frac{df}{dx} &= a \\
\frac{d}{dx} f_c(x) &= c + x & \Rightarrow & & \frac{df}{dx} &= 1 \\
\frac{d}{dx} \frac{1}{x} &\Rightarrow & & \frac{df}{dx} &= -\frac{1}{x^2}
\end{align*}
\]
Another example: 

\[ f(w, x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}} \]

\[ f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x \]

\[ f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a \]

\[ f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1 \]
Another example: \( f(w, x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}} \)

\[
\begin{align*}
  f(x) &= e^x \\
  f_a(x) &= ax \\
  f_c(x) &= c + x
\end{align*}
\]

\[
\begin{align*}
  \frac{df}{dx} &= e^x \\
  \frac{df}{dx} &= a \\
  \frac{df}{dx} &= \frac{1}{x^2}
\end{align*}
\]
Another example:

\[ f(w, x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}} \]

\[ [\text{upstream gradient}] \times [\text{local gradient}] \]
\[ [0.2] \times [1] = 0.2 \]
\[ [0.2] \times [1] = 0.2 \text{ (both inputs!)} \]

\[
\begin{align*}
  f(x) &= e^x \\
  f_a(x) &= ax
\end{align*}
\]

\[
\begin{align*}
  \frac{df}{dx} &= e^x \\
  \frac{df}{dx} &= a
\end{align*}
\]

\[
\begin{align*}
  f(x) &= \frac{1}{x} \\
  f_c(x) &= c + x
\end{align*}
\]

\[
\begin{align*}
  \frac{df}{dx} &= -\frac{1}{x^2} \\
  \frac{df}{dx} &= 1
\end{align*}
\]
Another example:

\[ f(w, x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}} \]

\[
\begin{align*}
f(x) &= e^x & \rightarrow & & \frac{df}{dx} &= e^x \\
f_a(x) &= ax & \rightarrow & & \frac{df}{dx} &= a \\
f_c(x) &= c + x & \rightarrow & & \frac{df}{dx} &= 1
\end{align*}
\]
Another example:

\[ f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2)}} \]

[upstream gradient] x [local gradient]

\[ x_0: [0.2] \times [2] = 0.4 \]
\[ w_0: [0.2] \times [-1] = -0.2 \]

\[
\begin{align*}
  f(x) &= e^x \\
f_a(x) &= ax
\end{align*}
\]

\[
\begin{align*}
  \frac{df}{dx} &= e^x \\
  \frac{df}{dx} &= a
\end{align*}
\]

\[
\begin{align*}
  f(x) &= \frac{1}{x} \\
  f_c(x) &= c + x
\end{align*}
\]

\[
\begin{align*}
  \frac{df}{dx} &= -\frac{1}{x^2} \\
  \frac{df}{dx} &= 1
\end{align*}
\]
Another example:

The Sigmoid function is defined as:

\[ f(w, x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}} \]

Computational graph representation may not be unique. Choose one where local gradients at each node can be easily expressed!
Another example:

$$f(w, x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}}$$

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

Sigmoid function

Computational graph representation may not be unique. Choose one where local gradients at each node can be easily expressed!

Sigmoid local gradient:

$$\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1 + e^{-x})^2} = \left(\frac{1 + e^{-x} - 1}{1 + e^{-x}}\right)\left(\frac{1}{1 + e^{-x}}\right) = (1 - \sigma(x))\sigma(x)$$
Another example:

\[ f(w, x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}} \]

\[ \sigma(x) = \frac{1}{1 + e^{-x}} \]

Sigmoid function

Computational graph representation may not be unique. Choose one where local gradients at each node can be easily expressed!

\[ \text{Sigmoid local gradient:} \quad \frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1 + e^{-x})^2} = \left( \frac{1 + e^{-x} - 1}{1 + e^{-x}} \right) \left( \frac{1}{1 + e^{-x}} \right) = (1 - \sigma(x)) \sigma(x) \]

\[ [\text{upstream gradient}] \times [\text{local gradient}] = [1.00] \times [(1 - 0.73)(0.73)] = 0.2 \]
Patterns in gradient flow

**add** gate: gradient distributor

\[
\begin{array}{c}
3 \\
2 \\
4 \\
2
\end{array}
\quad \rightarrow \quad + \quad \rightarrow \quad \begin{array}{c}
7 \\
2
\end{array}
\]
Patterns in gradient flow

**add** gate: gradient distributor

\[
\begin{align*}
3 & \quad 2 \\
2 & \quad 4 \\
2 & \quad \\
\end{align*}
\]

\[
\begin{align*}
& \quad 7 \\
& \quad 2 \\
\end{align*}
\]

**mul** gate: “swap multiplier”

\[
\begin{align*}
2 & \\
5 & \quad 3 \\
2 & \quad 5 \\
\end{align*}
\]

\[
\begin{align*}
& \quad 6 \\
& \quad 5 \\
\end{align*}
\]

\[
\begin{align*}
5 \times 3 &= 15 \\
2 \times 5 &= 10
\end{align*}
\]
Patterns in gradient flow

**add** gate: gradient distributor

3 → 2 → + → 7 → 2
4 → + → 7
2 → + → 7

**mul** gate: “swap multiplier”

2 → 5*3=15 → 6 → 5
3 → 2*5=10

**copy** gate: gradient adder

7 → 4+2=6
4 → + → 7
7 → + → 7
2 → + → 7

Fei-Fei Li & Justin Johnson & Serena Yeung
Patterns in gradient flow

**add** gate: gradient distributor

```
3
2
4
2

+ 7 2
```

**mul** gate: “swap multiplier”

```
2
5*3=15
3
2*5=10

x 6 5
```

**copy** gate: gradient adder

```
7
4+2=6

7
4
7
2
```

**max** gate: gradient router

```
4
0
5
9

max 5 9
```
Backprop Implementation: “Flat” code

Forward pass: Compute output

```
def f(w0, x0, w1, x1, w2):
    s0 = w0 * x0
    s1 = w1 * x1
    s2 = s0 + s1
    s3 = s2 + w2
    L = sigmoid(s3)
```

Backward pass: Compute grads

```
grad_L = 1.0
grad_s3 = grad_L * (1 - L) * L
grad_w2 = grad_s3
grad_s2 = grad_s3
grad_s0 = grad_s2
grad_s1 = grad_s2
grad_w1 = grad_s1 * x1
grad_x1 = grad_s1 * w1
grad_w0 = grad_s0 * x0
grad_x0 = grad_s0 * w0
```
Backprop Implementation: “Flat” code

Forward pass:
Compute output

Base case

```
def f(w0, x0, w1, x1, w2):
    s0 = w0 * x0
    s1 = w1 * x1
    s2 = s0 + s1
    s3 = s2 + w2
    L = sigmoid(s3)

    grad_L = 1.0
    grad_s3 = grad_L * (1 - L) * L
    grad_w2 = grad_s3
    grad_s2 = grad_s3
    grad_s0 = grad_s2
    grad_s1 = grad_s2
    grad_w1 = grad_s1 * x1
    grad_x1 = grad_s1 * w1
    grad_w0 = grad_s0 * x0
    grad_x0 = grad_s0 * w0
```
Backprop Implementation: “Flat” code

Forward pass:
Compute output

\[
def f(w0, x0, w1, x1, w2):
    s0 = w0 * x0
    s1 = w1 * x1
    s2 = s0 + s1
    s3 = s2 + w2
    L = \text{sigmoid}(s3)
\]

Sigmoid

\[
\text{grad}_L = 1.0
\]

\[
\text{grad}_s3 = \text{grad}_L \times (1 - L) \times L
\]

\[
\text{grad}_w2 = \text{grad}_s3
\]

\[
\text{grad}_s2 = \text{grad}_s3
\]

\[
\text{grad}_s0 = \text{grad}_s2
\]

\[
\text{grad}_s1 = \text{grad}_s2
\]

\[
\text{grad}_w1 = \text{grad}_s1 \times x1
\]

\[
\text{grad}_x1 = \text{grad}_s1 \times w1
\]

\[
\text{grad}_w0 = \text{grad}_s0 \times x0
\]

\[
\text{grad}_x0 = \text{grad}_s0 \times w0
\]
Backprop Implementation: “Flat” code

Forward pass:
Compute output

```
def f(w0, x0, w1, x1, w2):
    s0 = w0 * x0
    s1 = w1 * x1
    s2 = s0 + s1
    s3 = s2 + w2
    L = sigmoid(s3)

    grad_L = 1.0
    grad_s3 = grad_L * (1 - L) * L
    grad_w2 = grad_s3
    grad_s2 = grad_s3
    grad_s0 = grad_s2
    grad_s1 = grad_s2
    grad_w1 = grad_s1 * x1
    grad_x1 = grad_s1 * w1
    grad_w0 = grad_s0 * x0
    grad_x0 = grad_s0 * w0
```
Backprop Implementation: “Flat” code

Forward pass:
Compute output

```
def f(w0, x0, w1, x1, w2):
    s0 = w0 * x0
    s1 = w1 * x1
    s2 = s0 + s1
    s3 = s2 + w2
    L = sigmoid(s3)
```

Add gate

```
grad_L = 1.0
grad_s3 = grad_L * (1 - L) * L
grad_w2 = grad_s3
grad_s2 = grad_s3
grad_s0 = grad_s2
grad_s1 = grad_s2
grad_w1 = grad_s1 * x1
grad_x1 = grad_s1 * w1
grad_w0 = grad_s0 * x0
grad_x0 = grad_s0 * w0
```
Backprop Implementation:
“Flat” code

Forward pass:
Compute output

```
def f(w0, x0, w1, x1, w2):
    s0 = w0 * x0
    s1 = w1 * x1
    s2 = s0 + s1
    s3 = s2 + w2
    L = sigmoid(s3)

    grad_L = 1.0
    grad_s3 = grad_L * (1 - L) * L
    grad_w2 = grad_s3
    grad_s2 = grad_s3
    grad_s0 = grad_s2
    grad_s1 = grad_s2
    grad_w1 = grad_s1 * x1
    grad_x1 = grad_s1 * w1
    grad_w0 = grad_s0 * x0
    grad_x0 = grad_s0 * w0
```
Backprop Implementation: "Flat" code

Forward pass:
Compute output

```
def f(w0, x0, w1, x1, w2):
    s0 = w0 * x0
    s1 = w1 * x1
    s2 = s0 + s1
    s3 = s2 + w2
    L = sigmoid(s3)

    grad_L = 1.0
    grad_s3 = grad_L * (1 - L) * L
    grad_w2 = grad_s3
    grad_s2 = grad_s3
    grad_s0 = grad_s2
    grad_s1 = grad_s2
    grad_w1 = grad_s1 * x1
    grad_x1 = grad_s1 * w1
    grad_w0 = grad_s0 * x0
    grad_x0 = grad_s0 * w0
```
“Flat” Backprop: Do this for assignment 1!

Stage your forward/backward computation!

E.g. for the SVM:

```python
# receive W (weights), X (data)
# forward pass (we have 6 lines)
scores = #...
margins = #...
data_loss = #...
reg_loss = #...
loss = data_loss + reg_loss
# backward pass (we have 5 lines)
dmargins = # ... (optionally, we go direct to dscores)
dscores = #...
dW = #...
```

\[ f = WX \]

\[ L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1) \]
“Flat” Backprop: Do this for assignment 1!

E.g. for two-layer neural net:

```python
# receive W1,W2,b1,b2 (weights/biases), X (data)
# forward pass:
h1 = #... function of X,W1,b1
scores = #... function of h1,W2,b2
loss = #... (several lines of code to evaluate Softmax loss)
# backward pass:
dscores = #...
dh1,dW2,db2 = #...
dW1,db1 = #...
```
Backprop Implementation: Modularized API

Graph (or Net) object  *(rough pseudo code)*

```python
class ComputationalGraph(object):
    #...

def forward(inputs):
    # 1. [pass inputs to input gates...]
    # 2. forward the computational graph:
    for gate in self.graph.nodes_topologically_sorted():
        gate.forward()
    return loss  # the final gate in the graph outputs the loss

def backward():
    for gate in reversed(self.graph.nodes_topologically_sorted()):
        gate.backward()  # little piece of backprop (chain rule applied)
    return inputs_gradients
```
Modularized implementation: forward / backward API

Gate / Node / Function object: Actual PyTorch code

```python
class Multiply(torch.autograd.Function):
    @staticmethod
    def forward(ctx, x, y):
        ctx.save_for_backward(x, y)
        z = x * y
        return z
    @staticmethod
    def backward(ctx, grad_z):
        x, y = ctx.saved_tensors
        grad_x = y * grad_z  # dz/dx * dL/dz
        grad_y = x * grad_z  # dz/dy * dL/dz
        return grad_x, grad_y
```

(x, y, z are scalars)
Example: PyTorch operators
The PyTorch sigmoid layer forward function is defined as:

\[
\sigma(x) = \frac{1}{1 + e^{-x}}
\]

The forward function is implemented as follows in C code:

```c
void THNN_(Sigmoid_updateOutput)(
    THNNState *state,
    THTensor *input,
    THTensor *output)
{
    THTensor_(sigmoid)(output, input);
}
```

And the update function is:

```c
void THNN_(Sigmoid_updateGradInput)(
    THNNState *state,
    THTensor *gradOutput,
    THTensor *gradInput,
    THTensor *output)
{
    THNN_CHECK_NELEMENT(output, gradOutput);
    THTensor_(resizeAs)(gradInput, output);
    TH_TENSOR_APPLY3(scalar_t, gradInput, scalar_t, gradOutput, scalar_t, output,
        scalar_t z = *output_data;
        *gradInput_data = *gradOutput_data * (1.0 - z) * z;
    );
}
```
PyTorch sigmoid layer

\[ \sigma(x) = \frac{1}{1 + e^{-x}} \]

```c
static void sigmoid_kernel(TensorIterator& iter) {
    AT_DISPATCH_FLOATING_TYPES(iter.dtype(), "sigmoid_cpu", [&]() {
        unary_kernel_vec(
            iter,
            [=](scalar_t a) -> scalar_t {
                return (1 / (1 + std::exp(-a))));
            },
            [=](Vec256<scalar_t> a) {
                a = Vec256<scalar_t>((scalar_t)(0)) - a;
                a = a.exp();
                a = Vec256<scalar_t>((scalar_t)(1)) + a;
                a = a.reciprocal();
                return a;
            });
    });
}
```

Forward

Forward actually defined elsewhere...
PyTorch sigmoid layer

Forward
\[
\sigma(x) = \frac{1}{1 + e^{-x}}
\]

Backward
\[
(1 - \sigma(x)) \sigma(x)
\]

Forward actually defined elsewhere...
So far: backprop with scalars

What about vector-valued functions?
Recap: Vector derivatives

Scalar to Scalar

\[ x \in \mathbb{R}, \ y \in \mathbb{R} \]

Regular derivative:

\[ \frac{\partial y}{\partial x} \in \mathbb{R} \]

If \( x \) changes by a small amount, how much will \( y \) change?
Recap: Vector derivatives

Scalar to Scalar

\[ x \in \mathbb{R}, \ y \in \mathbb{R} \]

Regular derivative:

\[ \frac{\partial y}{\partial x} \in \mathbb{R} \]

If \( x \) changes by a small amount, how much will \( y \) change?

Vector to Scalar

\[ x \in \mathbb{R}^N, \ y \in \mathbb{R} \]

Derivative is Gradient:

\[ \frac{\partial y}{\partial x} \in \mathbb{R}^N \]

\[ \left( \frac{\partial y}{\partial x} \right)_n = \frac{\partial y}{\partial x_n} \]

For each element of \( x \), if it changes by a small amount then how much will \( y \) change?
## Recap: Vector derivatives

<table>
<thead>
<tr>
<th></th>
<th>Scalar to Scalar</th>
<th>Vector to Scalar</th>
<th>Vector to Vector</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>x ∈ R, y ∈ R</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Regular derivative:</strong></td>
<td><strong>Derivative is Gradient:</strong></td>
<td><strong>Derivative is Jacobian:</strong></td>
</tr>
<tr>
<td></td>
<td>( \frac{\partial y}{\partial x} \in \mathbb{R} )</td>
<td>( \frac{\partial y}{\partial x} \in \mathbb{R}^N ) ( \left( \frac{\partial y}{\partial x} \right)_n = \frac{\partial y}{\partial x_n} )</td>
<td>( \frac{\partial y}{\partial x} \in \mathbb{R}^{N \times M} ) ( \left( \frac{\partial y}{\partial x} \right)_{n,m} = \frac{\partial y_m}{\partial x_n} )</td>
</tr>
<tr>
<td>If x changes by a small amount, how much will y change?</td>
<td>For each element of x, if it changes by a small amount then how much will y change?</td>
<td>For each element of x, if it changes by a small amount then how much will each element of y change?</td>
<td></td>
</tr>
</tbody>
</table>
Backprop with Vectors

\[ \frac{\partial L}{\partial x} = \frac{\partial z}{\partial x} \frac{\partial L}{\partial z} \]

\[ \frac{\partial L}{\partial y} = \frac{\partial z}{\partial y} \frac{\partial L}{\partial z} \]

"local gradients"

"Downstream gradients"

"Upstream gradient"

Loss L still a scalar!
Backprop with Vectors

**Downstream gradients**

\[
\frac{\partial L}{\partial x} = \frac{\partial z}{\partial x} \frac{\partial L}{\partial z}
\]

**Upstream gradients**

\[
\frac{\partial L}{\partial y} = \frac{\partial z}{\partial y} \frac{\partial L}{\partial z}
\]

Loss L still a scalar!

For each element of z, how much does it influence L?
Backprop with Vectors

For each element of \( z \), how much does it influence \( L \)?

Loss \( L \) still a scalar!

“local gradients”

\[
\begin{bmatrix}
\frac{\partial L}{\partial x} \\
\frac{\partial L}{\partial y} \\
\frac{\partial L}{\partial z}
\end{bmatrix} = \begin{bmatrix}
\frac{\partial z}{\partial x} \\
\frac{\partial z}{\partial y} \\
\frac{\partial z}{\partial z}
\end{bmatrix}
\]

Jacobian matrices

\[
[D_x \times D_z] \quad f \quad [D_y \times D_z]
\]
Backprop with Vectors

\[ \frac{\partial L}{\partial x} = \frac{\partial z}{\partial x} \frac{\partial L}{\partial z} \]

“Downstream gradients”

Matrix-vector multiply

\[ \frac{\partial z}{\partial y} = \frac{\partial z}{\partial y} \frac{\partial L}{\partial z} \]

“Upstream gradient”

For each element of \( z \), how much does it influence \( L \)?

Loss \( L \) still a scalar!

Jacobian matrices

\[ [D_x \times D_z] \]

\[ [D_y \times D_z] \]
Backprop with Vectors

4D input $x$: 

\[
\begin{bmatrix}
1 \\
-2 \\
3 \\
-1
\end{bmatrix}
\]

4D output $y$: 

\[
\begin{bmatrix}
1 \\
0 \\
3 \\
0
\end{bmatrix}
\]

$f(x) = \max(0,x)$ 

*(elementwise)*
Backprop with Vectors

4D input $x$:  
\[
\begin{bmatrix}
  1 \\
  -2 \\
  3 \\
  -1
\end{bmatrix}
\]

$f(x) = \max(0, x)$ (elementwise)

4D output $y$:  
\[
\begin{bmatrix}
  1 \\
  0 \\
  3 \\
  0
\end{bmatrix}
\]

4D $dL/dy$:  
\[
\begin{bmatrix}
  4 \\
  -1 \\
  5 \\
  9
\end{bmatrix}
\]  

Upstream gradient
Backprop with Vectors

4D input $x$:

\[
\begin{bmatrix}
1 \\
-2 \\
3 \\
-1
\end{bmatrix}
\]

$f(x) = \max(0, x)$

(elementwise)

4D output $y$:

\[
\begin{bmatrix}
1 \\
0 \\
3 \\
0
\end{bmatrix}
\]

Jacobian $dy/dx$

\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0
\end{bmatrix}
\]

4D $dL/dy$:

\[
\begin{bmatrix}
4 \\
-1 \\
5 \\
9
\end{bmatrix}
\]

Upstream gradient
Backprop with Vectors

4D input $x$:

$$
\begin{bmatrix}
  1 \\
  -2 \\
  3 \\
  -1
\end{bmatrix}
$$

$f(x) = \max(0,x)$ (elementwise)

4D output $y$:

$$
\begin{bmatrix}
  1 \\
  0 \\
  3 \\
  0
\end{bmatrix}
$$

4D $\frac{dL}{dy}$:

$$
\begin{bmatrix}
  4 \\
  -1 \\
  5 \\
  9
\end{bmatrix}
$$

$\frac{dy}{dx}$ $\frac{dL}{dy}$

$$
\begin{bmatrix}
  1 & 0 & 0 & 0 \\
  0 & 0 & 0 & 0 \\
  0 & 0 & 1 & 0 \\
  0 & 0 & 0 & 0
\end{bmatrix}
\begin{bmatrix}
  4 \\
  -1 \\
  5 \\
  9
\end{bmatrix}
$$

Upstream gradient
Backprop with Vectors

4D input $x$:

\[
\begin{bmatrix}
1 \\
-2 \\
3 \\
-1
\end{bmatrix}
\]

$f(x) = \max(0,x)$ (elementwise)

4D output $y$:

\[
\begin{bmatrix}
1 \\
0 \\
3 \\
0
\end{bmatrix}
\]

4D $dL/dx$:

\[
\begin{bmatrix}
4 \\
0 \\
5 \\
0
\end{bmatrix}
\]

4D $dL/dy$:

\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0
\end{bmatrix}
\begin{bmatrix}
4 \\
-1 \\
5 \\
9
\end{bmatrix}
\]

Upstream gradient
Backprop with Vectors

4D input \( x \):

\[
\begin{bmatrix}
1 \\
-2 \\
3 \\
-1
\end{bmatrix}
\]

4D output \( y \):

\[
\begin{bmatrix}
1 \\
0 \\
3 \\
0
\end{bmatrix}
\]

4D \( \frac{dL}{dx} \):

\[
\begin{bmatrix}
4 \\
0 \\
5 \\
0
\end{bmatrix}
\]

4D \( \frac{dL}{dy} \):

\[
\begin{bmatrix}
1 0 0 0 \\
0 0 0 0 \\
0 0 1 0 \\
0 0 0 0
\end{bmatrix}
\]

\[
\begin{bmatrix}
4 \\
-1 \\
5 \\
9
\end{bmatrix}
\]

Jacobian is \textbf{sparse}: off-diagonal entries always zero! Never \textbf{explicitly} form Jacobian -- instead use \textbf{implicit} multiplication.

Upstream gradient
Backprop with Vectors

4D input $x$: 
\[
\begin{bmatrix}
1 \\
-2 \\
3 \\
-1
\end{bmatrix}
\]

$f(x) = \max(0,x)$ (elementwise)

4D output $y$: 
\[
\begin{bmatrix}
1 \\
0 \\
3 \\
0
\end{bmatrix}
\]

4D $dL/dy$: 
\[
\begin{bmatrix}
4 \\
-1 \\
5 \\
0
\end{bmatrix}
\]

$\frac{dy}{dx} \cdot \frac{dL}{dy}$

4D $dL/dx$: 
\[
\begin{bmatrix}
4 \\
0 \\
5 \\
0
\end{bmatrix}
\]

Jacobian is sparse: off-diagonal entries always zero! Never explicitly form Jacobian -- instead use implicit multiplication

Upstream gradient

$\frac{\partial L}{\partial x_i} = \begin{cases} 
\frac{\partial L}{\partial y_i} & \text{if } x_i > 0 \\
0 & \text{otherwise}
\end{cases}$
Backprop with Matrices (or Tensors)

For each element of \( x \), how much does it influence each element of \( z \)?

For each element of \( y \), how much does it influence each element of \( z \)?

**“Local gradients”**

\[
\begin{align*}
\frac{\partial L}{\partial x} &= \frac{\partial z}{\partial x} \frac{\partial L}{\partial z} \\
\frac{\partial L}{\partial y} &= \frac{\partial z}{\partial y} \frac{\partial L}{\partial z}
\end{align*}
\]

Loss \( L \) still a scalar!

d\( L \)/d\( x \) always has the same shape as \( x \)!

Jacobian matrices

Matrix-vector multiply

\[
\begin{align*}
[D_x \times M_x] \\
[D_y \times M_y] \\
[D_z \times M_z]
\end{align*}
\]
Backprop with Matrices (or Tensors)

Loss $L$ still a scalar!

$dL/dx$ always has the same shape as $x$!

For each element of $y$, how much does it influence each element of $z$?

For each element of $z$, how much does it influence $L$?

Jacobian matrices

For each element of $z$, how much does it influence $L$?

For each element of $y$, how much does it influence each element of $z$?

Matrix-vector multiply

"Downstream gradients"

"Local gradients"

"Upstream gradient"

\[
\frac{\partial L}{\partial x} = \frac{\partial z}{\partial x} \frac{\partial L}{\partial z} \]

\[
\frac{\partial L}{\partial y} = \frac{\partial z}{\partial y} \frac{\partial L}{\partial z} \]

\[
\frac{\partial z}{\partial x} \times (D_z \times M_z) \]

\[
(D_y \times M_y) \times (D_z \times M_z) \]

\[
(D_x \times M_x) \times (D_z \times M_z) \]

\[
(D_y \times M_y) \times (D_z \times M_z) \]

\[
[D_x \times M_x] \]

\[
[D_z \times M_z] \]

\[
[D_z \times M_z] \]

\[
[D_y \times M_y] \]

\[
[D_y \times M_y] \]
### Backprop with Matrices

**x**: \([N \times D]\)

\[
\begin{bmatrix}
2 & 1 & -3 \\
-3 & 4 & 2 
\end{bmatrix}
\]

**w**: \([D \times M]\)

\[
\begin{bmatrix}
3 & 2 & 1 & -1 \\
2 & 1 & 3 & 2 \\
3 & 2 & 1 & -2 
\end{bmatrix}
\]

**y**: \([N \times M]\)

\[
\begin{bmatrix}
13 & 9 & -2 & -6 \\
5 & 2 & 17 & 1 
\end{bmatrix}
\]

**dL/dy**: \([N \times M]\)

\[
\begin{bmatrix}
2 & 3 & -3 & 9 \\
-8 & 1 & 4 & 6 
\end{bmatrix}
\]

**Matrix Multiply**

\[
y_{n,m} = \sum_{d} x_{n,d} w_{d,m}
\]

Also see derivation in the course notes:

Backprop with Matrices

\[ x: [N \times D] \]
\[
\begin{bmatrix}
2 & 1 & -3 \\
-3 & 4 & 2
\end{bmatrix}
\]

\[ w: [D \times M] \]
\[
\begin{bmatrix}
3 & 2 & 1 & -1 \\
2 & 1 & 3 & 2 \\
3 & 2 & 1 & -2
\end{bmatrix}
\]

Matrix Multiply

\[ y_{n,m} = \sum_d x_{n,d}w_{d,m} \]

\[ y: [N \times M] \]
\[
\begin{bmatrix}
13 & 9 & -2 & -6 \\
5 & 2 & 17 & 1
\end{bmatrix}
\]

\[ dL/dy: [N \times M] \]
\[
\begin{bmatrix}
2 & 3 & -3 & 9 \\
-8 & 1 & 4 & 6
\end{bmatrix}
\]

Jacobians:
\[
\frac{dy}{dx}: [(N \times D) \times (N \times M)]
\]
\[
\frac{dy}{dw}: [(D \times M) \times (N \times M)]
\]

For a neural net we may have
\[ N=64, D=M=4096 \]
Each Jacobian takes 256 GB of memory!
Must work with them implicitly!
Backprop with Matrices

\[ x: [N \times D] \]
\[
\begin{bmatrix}
2 & 1 & -3 \\
-3 & 4 & 2 
\end{bmatrix}
\]

\[ w: [D \times M] \]
\[
\begin{bmatrix}
3 & 2 & 1 & -1 \\
2 & 1 & 3 & 2 \\
3 & 2 & 1 & -2 
\end{bmatrix}
\]

Matrix Multiply

\[ y_{n,m} = \sum_d x_{n,d} w_{d,m} \]

\[ y: [N \times M] \]
\[
\begin{bmatrix}
13 & 9 & -2 & -6 \\
5 & 2 & 17 & 1 
\end{bmatrix}
\]

\[ dL/dy: [N \times M] \]
\[
\begin{bmatrix}
2 & 3 & -3 & 9 \\
-8 & 1 & 4 & 6 
\end{bmatrix}
\]

Q: What parts of \( y \) are affected by one element of \( x \)?
Backprop with Matrices

\[ x: [N \times D] \]
\[
\begin{bmatrix}
2 & 1 & -3 \\
-3 & 4 & 2
\end{bmatrix}
\]

\[ w: [D \times M] \]
\[
\begin{bmatrix}
3 & 2 & 1 & -1 \\
2 & 1 & 3 & 2 \\
3 & 2 & 1 & -2
\end{bmatrix}
\]

\[ y: [N \times M] \]
\[
\begin{bmatrix}
13 & 9 & -2 & -6 \\
5 & 2 & 17 & 1
\end{bmatrix}
\]

\[ dL/dy: [N \times M] \]
\[
\begin{bmatrix}
2 & 3 & -3 & 9 \\
-8 & 1 & 4 & 6
\end{bmatrix}
\]

**Q:** What parts of \( y \) are affected by one element of \( x \)?

**A:** \( x_{n,d} \) affects the whole row \( y_n \).

\[
\frac{\partial L}{\partial x_{n,d}} = \sum_m \frac{\partial L}{\partial y_{n,m}} \frac{\partial y_{n,m}}{\partial x_{n,d}}
\]
Backprop with Matrices

\[ x: [N \times D] \]
\[
\begin{bmatrix}
2 & 1 \\
-3 & 4 & 2
\end{bmatrix}
\]

\[ w: [D \times M] \]
\[
\begin{bmatrix}
3 & 2 & 1 & -1 \\
2 & 1 & 3 & 2 \\
3 & 2 & 1 & -2
\end{bmatrix}
\]

\[ y: [N \times M] \]
\[
\begin{bmatrix}
13 & 9 & -2 & -6 \\
5 & 2 & 17 & 1
\end{bmatrix}
\]

\[ dL/dy: [N \times M] \]
\[
\begin{bmatrix}
2 & 3 & -3 & 9 \\
-8 & 1 & 4 & 6
\end{bmatrix}
\]

Q: What parts of \( y \) are affected by one element of \( x \)?
A: \( x_{n,d} \) affects the whole row \( y_n \).

Q: How much does \( x_{n,d} \) affect \( y_{n,m} \)?

\[
\frac{\partial L}{\partial x_{n,d}} = \sum_m \frac{\partial L}{\partial y_{n,m}} \frac{\partial y_{n,m}}{\partial x_{n,d}}
\]
Backprop with Matrices

\[ x: [N \times D] \]
\[
\begin{bmatrix}
2 & 1 & -3 \\
-3 & 4 & 2
\end{bmatrix}
\]

\[ w: [D \times M] \]
\[
\begin{bmatrix}
3 & 2 & 1 & -1 \\
2 & 1 & 3 & 2 \\
3 & 2 & 1 & -2
\end{bmatrix}
\]

\[ y: [N \times M] \]
\[
\begin{bmatrix}
13 & 9 & -2 & -6 \\
5 & 2 & 17 & 1
\end{bmatrix}
\]

\[ dL/dy: [N \times M] \]
\[
\begin{bmatrix}
2 & 3 & -3 & 9 \\
-8 & 1 & 4 & 6
\end{bmatrix}
\]

**Q:** What parts of \( y \) are affected by one element of \( x \)?

**A:** \( x_{n,d} \) affects the whole row \( y_{n,} \).

\[ \frac{\partial L}{\partial x_{n,d}} = \sum_m \frac{\partial L}{\partial y_{n,m}} \frac{\partial y_{n,m}}{\partial x_{n,d}} = \sum_m \frac{\partial L}{\partial y_{n,m}} w_{d,m} \]
### Backprop with Matrices

**x**: $[N \times D]$

\[
\begin{bmatrix}
  2 & 1 & -3 \\
  -3 & 4 & 2 \\
\end{bmatrix}
\]

**w**: $[D \times M]$

\[
\begin{bmatrix}
  3 & 2 & 1 & -1 \\
  2 & 1 & 3 & 2 \\
  3 & 2 & 1 & -2 \\
\end{bmatrix}
\]

**y**: $[N \times M]$

\[
\begin{bmatrix}
  13 & 9 & -2 & -6 \\
  5 & 2 & 17 & 1 \\
\end{bmatrix}
\]

**dL/dy**: $[N \times M]$

\[
\begin{bmatrix}
  2 & 3 & -3 & 9 \\
  -8 & 1 & 4 & 6 \\
\end{bmatrix}
\]

**Matrix Multiply**

\[
y_{n,m} = \sum_d x_{n,d} w_{d,m}
\]

Q: What parts of $y$ are affected by one element of $x$?

A: $x_{n,d}$ affects the whole row $y_n$.

Q: How much does $x_{n,d}$ affect $y_{n,m}$?

A: $w_{d,m}$

\[
\frac{\partial L}{\partial x} = \left( \frac{\partial L}{\partial y} \right)^T w
\]

\[
\frac{\partial L}{\partial x_{n,d}} = \sum_m \frac{\partial L}{\partial y_{n,m}} \frac{\partial y_{n,m}}{\partial x_{n,d}} = \sum_m \frac{\partial L}{\partial y_{n,m}} w_{d,m}
\]
Backprop with Matrices

\[ x: [N\times D] \]
\[
\begin{bmatrix}
2 & 1 & -3 \\
-3 & 4 & 2
\end{bmatrix}
\]

\[ w: [D\times M] \]
\[
\begin{bmatrix}
3 & 2 & 1 & -1 \\
2 & 1 & 3 & 2 \\
3 & 2 & 1 & -2
\end{bmatrix}
\]

\[ y: [N\times M] \]
\[
\begin{bmatrix}
13 & 9 & -2 & -6 \\
5 & 2 & 17 & 1
\end{bmatrix}
\]

\[ dL/dy: [N\times M] \]
\[
\begin{bmatrix}
2 & 3 & -3 & 9 \\
-8 & 1 & 4 & 6
\end{bmatrix}
\]

Matrix Multiply

\[ y_{n,m} = \sum_d x_{n,d} w_{d,m} \]

By similar logic:

These formulas are easy to remember: they are the only way to make shapes match up!
Summary for today:

- **(Fully-connected) Neural Networks** are stacks of linear functions and nonlinear activation functions; they have much more representational power than linear classifiers.
- **backpropagation** = recursive application of the chain rule along a computational graph to compute the gradients of all inputs/parameters/intermediates.
- Implementations maintain a graph structure, where the nodes implement the `forward()` / `backward()` API.
- **forward**: compute result of an operation and save any intermediates needed for gradient computation in memory.
- **backward**: apply the chain rule to compute the gradient of the loss function with respect to the inputs.
Next Time: Convolutional Networks!
A vectorized example: $f(x, W) = \|W \cdot x\|^2 = \sum_{i=1}^{n} (W \cdot x)_{i}^2$
A vectorized example: $f(x, W) = \|W \cdot x\|^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$

$\in \mathbb{R}^n \in \mathbb{R}^{n \times n}$
A vectorized example: $f(x, W) = \|W \cdot x\|^2 = \sum_{i=1}^{n} (W \cdot x)^2_i$
A vectorized example: \( f(x, W) = \|W \cdot x\|^2 = \sum_{i=1}^{n} (W \cdot x)_i^2 \)

\[
W = \begin{bmatrix}
0.1 & 0.5 \\
-0.3 & 0.8
\end{bmatrix}
\]

\[
x = \begin{bmatrix}
0.2 \\
0.4
\end{bmatrix}
\]

\[q = W \cdot x = \begin{pmatrix}
W_{1,1}x_1 + \cdots + W_{1,n}x_n \\
\vdots \\
W_{n,1}x_1 + \cdots + W_{n,n}x_n
\end{pmatrix}
\]

\[f(q) = \|q\|^2 = q_1^2 + \cdots + q_n^2\]
A vectorized example: 

\[
f(x, W) = \|W \cdot x\|^2 = \sum_{i=1}^{n} (W \cdot x)_i^2
\]

\[
W = \begin{bmatrix}
0.1 & 0.5 \\
-0.3 & 0.8
\end{bmatrix}
\]

\[
x = \begin{bmatrix}
0.2 \\
0.4
\end{bmatrix}
\]

\[
q = W \cdot x = \begin{pmatrix}
W_{1,1}x_1 + \cdots + W_{1,n}x_n \\
\vdots \\
W_{n,1}x_1 + \cdots + W_{n,n}x_n
\end{pmatrix}
\]

\[
f(q) = \|q\|^2 = q_1^2 + \cdots + q_n^2
\]
A vectorized example: $f(x, W) = \|W \cdot x\|^2 = \sum_{i=1}^{n} (W \cdot x)^2_i$

\[ W = \begin{bmatrix} 0.1 & 0.5 \\ -0.3 & 0.8 \end{bmatrix}, \quad x = \begin{bmatrix} 0.2 \\ 0.4 \end{bmatrix} \]

\[ q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix} \]

\[ f(q) = \|q\|^2 = q_1^2 + \cdots + q_n^2 \]
A vectorized example: 

\[ f(x, W) = \|W \cdot x\|^2 = \sum_{i=1}^{n} (W \cdot x)^2_i \]

\[
\begin{bmatrix}
0.1 & 0.5 \\
-0.3 & 0.8
\end{bmatrix}
\]

\[
\begin{bmatrix}
0.2 \\
0.4
\end{bmatrix}
\]

\[
\begin{bmatrix}
0.22 \\
0.26
\end{bmatrix}
\]

\[
L2
\]

\[
q = W \cdot x = \begin{pmatrix}
W_{1,1}x_1 + \cdots + W_{1,n}x_n \\
\vdots \\
W_{n,1}x_1 + \cdots + W_{n,n}x_n
\end{pmatrix}
\]

\[
\frac{\partial f}{\partial q_i} = 2q_i
\]

\[
\nabla_q f = 2q
\]

\[
f(q) = \|q\|^2 = q_1^2 + \cdots + q_n^2
\]
A vectorized example: \[ f(x, W) = \|W \cdot x\|^2 = \sum_{i=1}^{n} (W \cdot x)_i^2 \]

\[ W = \begin{pmatrix} 0.1 & 0.5 \\ -0.3 & 0.8 \end{pmatrix} \]

\[ x = \begin{pmatrix} 0.2 \\ 0.4 \end{pmatrix} \]

\[ \begin{pmatrix} 0.22 \\ 0.26 \end{pmatrix} \]

\[ \begin{pmatrix} 0.44 \\ 0.52 \end{pmatrix} \]

\[ q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix} \]

\[ \frac{\partial f}{\partial q_i} = 2q_i \]

\[ \nabla_q f = 2q \]

\[ f(q) = \|q\|^2 = q_1^2 + \cdots + q_n^2 \]
A vectorized example: \( f(x, W) = \|W \cdot x\|^2 = \sum_{i=1}^{n} (W \cdot x)_{i}^2 \)

\[
\begin{pmatrix}
0.1 & 0.5 \\
-0.3 & 0.8
\end{pmatrix}
\]

\[
\begin{pmatrix}
0.2 \\
0.4
\end{pmatrix}
\]

\[
\begin{pmatrix}
0.22 \\
0.26
\end{pmatrix}
\]

\[
\begin{pmatrix}
0.44 \\
0.52
\end{pmatrix}
\]

\[
\frac{\partial q_k}{\partial W_{i,j}} = 1_{k=i}x_j
\]

\[
q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix}
\]

\[
f(q) = \|q\|^2 = q_1^2 + \cdots + q_n^2
\]
A vectorized example: \( f(x, W) = \|W \cdot x\|^2 = \sum_{i=1}^{n} (W \cdot x)^2_i \)

\[
\begin{bmatrix}
0.1 & 0.5 \\
-0.3 & 0.8
\end{bmatrix}
\]

\[
\begin{bmatrix}
0.22 \\
0.26
\end{bmatrix}
\]

\[
\begin{bmatrix}
0.44 \\
0.52
\end{bmatrix}
\]

\[
0.116
\]

\[
1.00
\]

\[
q = W \cdot x = \begin{pmatrix}
W_{1,1}x_1 + \cdots + W_{1,n}x_n \\
\vdots \\
W_{n,1}x_1 + \cdots + W_{n,n}x_n
\end{pmatrix}
\]

\[
f(q) = \|q\|^2 = q_1^2 + \cdots + q_n^2
\]

\[
\frac{\partial q_k}{\partial W_{i,j}} = 1_{k=i}x_j
\]

\[
\frac{\partial f}{\partial W_{i,j}} = \sum_k \frac{\partial f}{\partial q_k} \frac{\partial q_k}{\partial W_{i,j}}
\]

\[
= \sum_k (2q_k)(1_{k=i}x_j)
\]

\[
= 2q_i x_j
\]
A vectorized example: \( f(x, W) = \|W \cdot x\|^2 = \sum_{i=1}^{n} (W \cdot x)_i^2 \)

\[
W = \begin{bmatrix}
0.1 & 0.5 \\
-0.3 & 0.8 \\
0.088 & 0.176 \\
0.104 & 0.208 \\
0.2 & 0.4 \\
\end{bmatrix}
\]

\[
x = \begin{bmatrix}
0.22 \\
0.26 \\
0.44 \\
0.52 \\
\end{bmatrix}
\]

\[
q = W \cdot x = \begin{pmatrix}
W_{1,1}x_1 + \cdots + W_{1,n}x_n \\
\vdots \\
W_{n,1}x_1 + \cdots + W_{n,n}x_n \\
\end{pmatrix}
\]

\[
f(q) = \|q\|^2 = q_1^2 + \cdots + q_n^2
\]

\[
\frac{\partial q_k}{\partial W_{i,j}} = 1_{k=i}x_j
\]

\[
\frac{\partial f}{\partial W_{i,j}} = \sum_k \frac{\partial f}{\partial q_k} \frac{\partial q_k}{\partial W_{i,j}}
\]

\[
= \sum_k (2q_k)(1_{k=i}x_j)
\]

\[
= 2q_i x_j
\]
A vectorized example:  

\[ f(x, W) = \|W \cdot x\|^2 = \sum_{i=1}^{n} (W \cdot x)_i^2 \]

\[ \nabla_W f = 2q \cdot x^T \]

\[
W = \begin{bmatrix}
0.1 & 0.5 \\
-0.3 & 0.8 \\
0.088 & 0.176 \\
0.104 & 0.208 \\
0.2 & 0.4
\end{bmatrix}
\]

\[
q = W \cdot x = \begin{pmatrix}
W_{1,1}x_1 + \cdots + W_{1,n}x_n \\
\vdots \\
W_{n,1}x_1 + \cdots + W_{n,n}x_n
\end{pmatrix}
\]

\[
f(q) = \|q\|^2 = q_1^2 + \cdots + q_n^2
\]

\[
\frac{\partial q_k}{\partial W_{i,j}} = 1_{k=i} x_j
\]

\[
\frac{\partial f}{\partial W_{i,j}} = \sum_k \frac{\partial f}{\partial q_k} \frac{\partial q_k}{\partial W_{i,j}}
\]

\[
= \sum_k (2q_k) (1_{k=i} x_j)
\]

\[
= 2q_i x_j
\]
A vectorized example:

\[ f(x, W) = \|W \cdot x\|^2 = \sum_{i=1}^{n} (W \cdot x)_i^2 \]

The gradient with respect to a variable should have the same shape as the variable.

\[ \nabla_W f = 2q \cdot x^T \]

\[ q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix} \]

\[ f(q) = \|q\|^2 = q_1^2 + \cdots + q_n^2 \]

\[ \frac{\partial q_k}{\partial W_{i,j}} = 1_{k=i}x_j \]

\[ \frac{\partial f}{\partial W_{i,j}} = \sum_k \frac{\partial f}{\partial q_k} \frac{\partial q_k}{\partial W_{i,j}} = \sum_k (2q_k)(1_{k=i}x_j) \]

Always check: The gradient with respect to a variable should have the same shape as the variable.
A vectorized example: \( f(x, W) = \| W \cdot x \|^2 = \sum_{i=1}^{n} (W \cdot x)_i^2 \)

\[
\begin{bmatrix}
-0.3 & 0.5 \\
0.088 & 0.176 \\
0.104 & 0.208 \\
0.2 & 0.4
\end{bmatrix}
\]

\[
W
\]

\[
\begin{bmatrix}
0.22 \\
0.26 \\
0.44 \\
0.52
\end{bmatrix}
\]

\[
x
\]

\[
\star
\]

\[
L2
\]

\[
\frac{\partial q_k}{\partial x_i} = W_{k,i}
\]

\[
q = W \cdot x = \begin{pmatrix}
W_{1,1}x_1 + \cdots + W_{1,n}x_n \\
\vdots \\
W_{n,1}x_1 + \cdots + W_{n,n}x_n
\end{pmatrix}
\]

\[
f(q) = \| q \|^2 = q_1^2 + \cdots + q_n^2
\]
A vectorized example: \[ f(x, W) = \|W \cdot x\|^2 = \sum_{i=1}^{n} (W \cdot x)_i^2 \]

\[
\begin{pmatrix}
0.1 & 0.5 \\
-0.3 & 0.8 \\
0.088 & 0.176 \\
0.104 & 0.208 \\
0.2 & 0.4
\end{pmatrix}
\]

\[
\begin{pmatrix}
0.22 \\
0.26 \\
0.44 \\
0.52
\end{pmatrix}
\]

\[ x \]

\[
W
\]

\[ L2 \]

\[ q = W \cdot x = \begin{pmatrix}
W_{1,1}x_1 + \cdots + W_{1,n}x_n \\
\vdots \\
W_{n,1}x_1 + \cdots + W_{n,n}x_n
\end{pmatrix}
\]

\[
f(q) = \|q\|^2 = q_1^2 + \cdots + q_n^2
\]

\[
\frac{\partial q_k}{\partial x_i} = W_{k,i}
\]

\[
\frac{\partial f}{\partial x_i} = \sum_k \frac{\partial f}{\partial q_k} \frac{\partial q_k}{\partial x_i} = \sum_k 2q_k W_{k,i}
\]
A vectorized example: \( f(x, W) = \|W \cdot x\|^2 = \sum_{i=1}^{n} (W \cdot x)^2_i \)

\[
W = \begin{bmatrix}
0.1 & 0.5 \\
-0.3 & 0.8 \\
0.088 & 0.176 \\
0.104 & 0.208 \\
0.2 & 0.4 \\
-0.112 & 0.636
\end{bmatrix}
\]

\[
x = \begin{bmatrix}
0.22 \\
0.26 \\
0.44 \\
0.52
\end{bmatrix}
\]

\[
q = W \cdot x = \begin{pmatrix}
W_{1,1}x_1 + \cdots + W_{1,n}x_n \\
\vdots \\
W_{n,1}x_1 + \cdots + W_{n,n}x_n
\end{pmatrix}
\]

\[
f(q) = \|q\|^2 = q_1^2 + \cdots + q_n^2
\]

\[
\nabla_x f = 2W^T \cdot q
\]
In discussion section: A matrix example...

\[ z_1 = XW_1 \]

\[ h_1 = \text{ReLU}(z_1) \]

\[ \hat{y} = h_1 W_2 \]

\[ L = \|\hat{y}\|_2^2 \]

\[ \frac{\partial L}{\partial W_2} = ? \]

\[ \frac{\partial L}{\partial W_1} = ? \]