Topics:
• Backpropagation / Automatic Differentiation
• Jacobians

CS 4644 / 7643-A
ZSOLT KIRA
• **Assignment Due Feb 5th**

• **Resources:**
  - These lectures
  - [Matrix calculus for deep learning](#)
  - [Gradients notes](#) and [MLP/ReLU Jacobian notes](#).
  - [Assignment (@41)](#) and [matrix calculus (@46)](#)

• **Project:** Teaming thread on piazza
To develop a general algorithm for this, we will view the function as a computation graph.

Graph can be any directed acyclic graph (DAG).

- Modules must be differentiable to support gradient computations for gradient descent.

A training algorithm will then process this graph, one module at a time.

Adapted from figure by Marc'Aurelio Ranzato, Yann LeCun.
Directed Acyclic Graphs (DAGs)

• Exactly what the name suggests
  – Directed edges
  – No (directed) cycles
  – Underlying undirected cycles okay
Directed Acyclic Graphs (DAGs)

• Concept
  – Topological Ordering
Directed Acyclic Graphs (DAGs)
\[ f(x_1, x_2) = \ln(x_1) + x_1 x_2 - \sin(x_2) \]
Machine Learning Example

Adapted from figure by Marc'Aurelio Ranzato, Yann LeCun
Backpropagation
Step 1: Compute Loss on Mini-Batch: Forward Pass

Adapted from figure by Marc'Aurelio Ranzato, Yann LeCun
**Step 1:** Compute Loss on Mini-Batch: **Forward Pass**

Layer 1 → Layer 2 → Layer 3

*Adapted from figure by Marc'Aurelio Ranzato, Yann LeCun*
Note that we must store the intermediate outputs of all layers!
- This is because we will need them to compute the gradients (the gradient equations will have terms with the output values in them)
**Step 1:** Compute Loss on Mini-Batch: **Forward Pass**

**Step 2:** Compute Gradients wrt parameters: **Backward Pass**
Step 1: Compute Loss on Mini-Batch: **Forward Pass**

Step 2: Compute Gradients wrt parameters: **Backward Pass**
Step 1: Compute Loss on Mini-Batch: **Forward Pass**

Step 2: Compute Gradients wrt parameters: **Backward Pass**
Step 1: Compute Loss on Mini-Batch: **Forward Pass**

Step 2: Compute Gradients wrt parameters: **Backward Pass**

Step 3: Use **gradient** to update **all parameters** at the end

Backpropagation is the application of gradient descent to a computation graph via the chain rule!

\[ w_i = w_i - \alpha \frac{\partial L}{\partial w_i} \]
We want to compute: \( \left\{ \frac{\partial L}{\partial h^{\ell-1}}, \frac{\partial L}{\partial W} \right\} \)

We will use the *chain rule* to do this:

Chain Rule: \( \frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} \cdot \frac{\partial y}{\partial x} \)
We will use the **chain rule** to compute: \( \left\{ \frac{\partial L}{\partial h^{\ell-1}}, \frac{\partial L}{\partial W} \right\} \)

**Gradient of loss w.r.t. inputs:** \( \frac{\partial L}{\partial h^{\ell-1}} = \frac{\partial L}{\partial h^{\ell}} \frac{\partial h^{\ell}}{\partial h^{\ell-1}} \)

**Gradient of loss w.r.t. weights:** \( \frac{\partial L}{\partial W} = \frac{\partial L}{\partial h^{\ell}} \frac{\partial h^{\ell}}{\partial W} \)

Adapted from figure by Marc'Aurelio Ranzato, Yann LeCun
Backpropagation: a simple example

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

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

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

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

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

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Gradients add at branches
Duality in Fprop and Bprop
Caffe Sigmoid Layer

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

\[(1 - \sigma(x)) \sigma(x) \] * top_diff (chain rule)
Linear Algebra View: Vector and Matrix Sizes
\[
\begin{bmatrix}
  w_{11} & w_{12} & \cdots & w_{1m} & b_1 \\
  w_{21} & w_{22} & \cdots & w_{2m} & b_2 \\
  w_{31} & w_{32} & \cdots & w_{3m} & b_3 \\
\end{bmatrix}
\begin{bmatrix}
  x_1 \\
  x_2 \\
  \vdots \\
  x_m \\
  1
\end{bmatrix}
\]

\[W \times x\]

Sizes: \([c \times (d + 1)] \quad [(d + 1) \times 1]\]

Where \(c\) is number of classes

\(d\) is dimensionality of input
Conventions:

- Size of derivatives for scalars, vectors, and matrices:

Assume we have scalar $s \in \mathbb{R}^1$, vector $\mathbf{v} \in \mathbb{R}^m$, i.e. $\mathbf{v} = [v_1, v_2, ..., v_m]^T$ and matrix $\mathbf{M} \in \mathbb{R}^{k \times \ell}$

$$
\begin{array}{ccc}
S & \frac{\partial s_1}{\partial s_2} & \frac{\partial s}{\partial \mathbf{v}} & \frac{\partial s}{\partial \mathbf{M}} \\
\frac{\partial s}{\partial s_2} & \frac{\partial v}{\partial s} & \frac{\partial v_1}{\partial v_2} \\
\frac{\partial v}{\partial s} & \frac{\partial v_1}{\partial v_2} & \text{Tensors}
\end{array}
$$
Conventions:

- Size of derivatives for scalars, vectors, and matrices:
  Assume we have scalar $s \in \mathbb{R}^1$, vector $\mathbf{v} \in \mathbb{R}^m$, i.e. $\mathbf{v} = [v_1, v_2, ..., v_m]^T$ and matrix $\mathbf{M} \in \mathbb{R}^{k \times \ell}$

- What is the size of $\frac{\partial \mathbf{v}}{\partial s}$? $\mathbb{R}^{m \times 1}$ (column vector of size $m$)

- What is the size of $\frac{\partial s}{\partial \mathbf{v}}$? $\mathbb{R}^{1 \times m}$ (row vector of size $m$)

$$\begin{bmatrix}
\frac{\partial v_1}{\partial s} \\
\frac{\partial v_2}{\partial s} \\
\vdots \\
\frac{\partial v_m}{\partial s}
\end{bmatrix}
\quad
\begin{bmatrix}
\frac{\partial s}{\partial v_1} & \frac{\partial s}{\partial v_1} & \cdots & \frac{\partial s}{\partial v_m}
\end{bmatrix}$$
Conventions:

- What is the size of $\frac{\partial v_1}{\partial v_2}$? A matrix:

$$
\begin{bmatrix}
\frac{\partial v_1}{\partial v_2} & \ldots & \ldots & \ldots & \ldots \\
\frac{\partial v_2}{\partial v_1} & \ldots & \ldots & \ldots & \ldots \\
\vdots & \ddots & \vdots & \ddots & \vdots \\
\frac{\partial v_i}{\partial v_1} & \ldots & \frac{\partial v_i}{\partial v_j} & \ldots & \frac{\partial v_i}{\partial v_m} \\
\vdots & \ldots & \vdots & \ddots & \vdots \\
\ldots & \ldots & \ldots & \ldots & \frac{\partial v_i}{\partial v_m} \\
\end{bmatrix}_{m_1 \times m_2}
$$

- This matrix of partial derivatives is called a Jacobian.

(Note this is slightly different convention than on Wikipedia). Also, computationally other conventions are used.
Conventions:

What is the size of $\frac{\partial s}{\partial M}$? A matrix:

$$
\begin{bmatrix}
\frac{\partial s}{\partial m_{[1,1]}} & \cdots & \cdots & \cdots & \cdots & \cdots \\
\cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
\cdots & \cdots & \frac{\partial s}{\partial m_{[i,j]}} & \cdots & \cdots & \cdots \\
\cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
\cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
\cdots & \cdots & \cdots & \cdots & \cdots & \cdots 
\end{bmatrix}
$$
Example 1:
\[ y = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} x \\ x^2 \end{bmatrix} \quad \frac{\partial y}{\partial x} = \begin{bmatrix} 1 \\ 2x \end{bmatrix} \]

Example 2:
\[ y = w^T x = \sum_k w_k x_k \]
\[ \frac{\partial y}{\partial x} = \begin{bmatrix} \frac{\partial y}{\partial x_1} \\ \vdots \\ \frac{\partial y}{\partial x_m} \end{bmatrix} = [w_1, \ldots, w_m] \quad \text{because} \quad \frac{\partial (\sum_k w_k x_k)}{\partial x_i} = w_i \]

= \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} x \\ x^2 \end{bmatrix} \quad \frac{\partial y}{\partial x} = \begin{bmatrix} 1 \\ 2x \end{bmatrix} \]

\[ y = w^T x = \sum_k w_k x_k \]
\[ \frac{\partial y}{\partial x} = \left[ \frac{\partial y}{\partial x_1}, \ldots, \frac{\partial y}{\partial x_m} \right] = [w_1, \ldots, w_m] \quad \text{because} \quad \frac{\partial (\sum_k w_k x_k)}{\partial x_i} = w_i \]

= w^T
Example 3:

\[
\frac{\partial (wA^w)}{\partial w} = 2w^TA \quad \text{(assuming A is symmetric)}
\]

Example 4:

\[
y = Wx \quad \quad \quad \frac{\partial y}{\partial x} = W
\]

\[
\begin{bmatrix}
\frac{\partial y_1}{\partial x_1} \\
\vdots \\
\frac{\partial y_i}{\partial x_j} \\
\vdots \\
\end{bmatrix}
\begin{bmatrix}
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
\vdots & \vdots & w_{ij} & \vdots & \vdots & \vdots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
\end{bmatrix}
= y_i = \sum_j w_{ij}x_j
\]
What is the size of $\frac{\partial L}{\partial W}$?

Remember that loss is a **scalar** and $W$ is a matrix:

$$ W = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1m} & b_1 \\ w_{21} & w_{22} & \cdots & w_{2m} & b_2 \\ w_{31} & w_{32} & \cdots & w_{3m} & b_3 \end{bmatrix} $$

Jacobian is also a matrix:

$$ \begin{bmatrix} \frac{\partial L}{\partial w_{11}} & \frac{\partial L}{\partial w_{12}} & \cdots & \frac{\partial L}{\partial w_{1m}} & \frac{\partial L}{\partial b_1} \\ \frac{\partial L}{\partial w_{21}} & \frac{\partial L}{\partial w_{12}} & \cdots & \frac{\partial L}{\partial w_{2m}} & \frac{\partial L}{\partial b_2} \\ \frac{\partial L}{\partial w_{31}} & \frac{\partial L}{\partial w_{32}} & \cdots & \frac{\partial L}{\partial w_{3m}} & \frac{\partial L}{\partial b_3} \end{bmatrix} $$
Batches of data are **matrices** or **tensors** (multi-dimensional matrices)

**Examples:**

- Each instance is a vector of size $m$, our batch is of size $[B \times m]$
- Each instance is a matrix (e.g. grayscale image) of size $W \times H$, our batch is $[B \times W \times H]$
- Each instance is a multi-channel matrix (e.g. color image with R,B,G channels) of size $C \times W \times H$, our batch is $[B \times C \times W \times H]$

**Jacobians become tensors which is complicated**

- Instead, flatten input to a vector and get a vector of derivatives!
- This can also be done for partial derivatives between two vectors, two matrices, or two tensors
Define:

\[ h_i = w_i^T h_{\ell-1} \]

\[ h_\ell = W h_{\ell-1} \]

\[
\begin{bmatrix}
|h_\ell| \times 1
\end{bmatrix}
\begin{bmatrix}
|h_\ell| \\
|h_\ell| \\
|h_{\ell-1}|
\end{bmatrix}
\begin{bmatrix}
|w_i^T| \\
|w_i^T| \\
|w_i^T|
\end{bmatrix}
\begin{bmatrix}
|h_{\ell-1}| \times 1
\end{bmatrix}
\]
\[ h^{\ell} = W h^{\ell-1} \]

\[ \frac{\partial h^{\ell}}{\partial h^{\ell-1}} = W \]

Define:

\[ h_i = w_i^T h^{\ell-1} \]

\[ \frac{\partial h_i}{\partial w_i} = h^{(\ell-1),T} \]

\[
\frac{\partial L}{\partial h^{\ell-1}} = \frac{\partial L}{\partial h^{\ell}} \frac{\partial h^{\ell}}{\partial h^{\ell-1}}
\]

\[
1 \times |h^{\ell-1}| \quad 1 \times |h^{\ell}| \quad |h^{\ell}| \times |h^{\ell-1}|
\]
\[ h^\ell = Wh^{\ell-1} \]
\[ \frac{\partial h^\ell}{\partial h^{\ell-1}} = W \]

Define:
\[ h_i = w_i^T h^{\ell-1} \]
\[ \frac{\partial h_i^\ell}{\partial w_i} = h^{(\ell-1),T} \]

Note doing this on full \( W \) matrix would result in Jacobian tensor!

But it is sparse – each output only affected by corresponding weight row.
We can employ any differentiable (or piecewise differentiable) function

A common choice is the **Rectified Linear Unit**

- Provides non-linearity but better gradient flow than sigmoid
- Performed **element-wise**

How many parameters for this layer?

\[ h^\ell = \max(0, h^{\ell-1}) \]
Full Jacobian of ReLU layer is large (output dim x input dim)

- But again it is sparse
- Only **diagonal values non-zero** because it is element-wise
- An output value affected only by corresponding input value

Max function **funnels gradients through selected max**

- Gradient will be **zero** if input <= 0

**Forward**: \( h^\ell = \max(0, h^{\ell-1}) \)

**Backward**: \( \frac{\partial L}{\partial h^{\ell-1}} = \frac{\partial L}{\partial h^\ell} \frac{\partial h^\ell}{\partial h^{\ell-1}} \)

For diagonal

\[
\frac{\partial h^\ell}{\partial h^{\ell-1}} = \begin{cases} 
1 & \text{if } h^{\ell-1} > 0 \\
0 & \text{otherwise}
\end{cases}
\]
Vectorization and Jacobians of Simple Layers
Composition of Functions: \[ f(g(x)) = (f \circ g)(x) \]

A complex function (e.g. defined by a neural network):

\[
\begin{align*}
    f(x) &= g_\ell (g_{\ell-1}(... g_1(x))) \\
    f(x) &= g_\ell \circ g_{\ell-1} ... \circ g_1(x)
\end{align*}
\]

(Many of these will be parameterized)

(Note you might find the opposite notation as well!)
Jacobian View of Chain Rule
Chain Rule: Cascaded
- Input: $x \in \mathbb{R}^D$
- Binary label: $y \in \{-1, +1\}$
- Parameters: $w \in \mathbb{R}^D$
- Output prediction: $p(y = 1|x) = \frac{1}{1 + e^{-w^T x}}$
- Loss: $L = \frac{1}{2} \|w\|^2 - \lambda \log(p(y|x))$

Adapted from slide by Marc'Aurelio Ranzato
We have discussed **computation graphs for generic functions**

Machine Learning functions (input -> model -> loss function) is also a computation graph

We can use the **computed gradients from backprop/automatic differentiation** to update the weights!
Example Gradient Computations

We can do this in a combined way to see all terms together:

\[ \bar{w} = \frac{\partial L}{\partial p} \frac{\partial p}{\partial u} \frac{\partial u}{\partial w} = -\frac{1}{\sigma(w^T x)} \sigma(w^T x) (1 - \sigma(w^T x)) x^T \]

This effectively shows gradient flow along path from \( L \) to \( w \).
The chain rule can be computed as a series of scalar, vector, and matrix linear algebra operations.

Extremely efficient in graphics processing units (GPUs)

\[
\bar{w} = -\frac{1}{\sigma(w^T x)} \sigma(w^T x)(1 - \sigma(w^T x))x^T
\]

Vectorized Computations
Automatic differentiation:
- Carries out this procedure for us on arbitrary graphs
- Knows derivatives of primitive functions
- As a result, we just define these (forward) functions and don’t even need to specify the gradient (backward) functions!

\[
\begin{align*}
L &= 1 \\
\bar{p} &= \frac{\partial L}{\partial p} = -\frac{1}{p} \\
\text{where } p &= \sigma(w^T x) \text{ and } \sigma(x) = \frac{1}{1+e^{-x}} \\
\bar{u} &= \frac{\partial L}{\partial u} = \frac{\partial L}{\partial p} \frac{\partial p}{\partial u} = \bar{p} \sigma(1 - \sigma) \\
\bar{w} &= \frac{\partial L}{\partial w} = \frac{\partial L}{\partial p} \frac{\partial p}{\partial u} \frac{\partial u}{\partial w} = \bar{u} x^T \\
\text{We can do this in a combined way to see all terms together:} \\
\bar{w} &= \frac{\partial L}{\partial p} \frac{\partial p}{\partial u} \frac{\partial u}{\partial w} = -\frac{1}{\sigma(w^T x)} \sigma(w^T x)(1 - \sigma(w^T x))x^T \\
&= -\left(1 - \sigma(w^T x)\right)x^T \\
\text{This effectively shows gradient flow along path from } L \text{ to } w
\end{align*}
\]

Example Gradient Computations
Computation Graph / Global View of Chain Rule

\[
L = \frac{1}{p} \quad \bar{p} = \frac{\partial L}{\partial p} = -\frac{1}{p}
\]

where \( p = \sigma(w^T x) \) and \( \sigma(x) = \frac{1}{1 + e^{-x}} \)

\[
\begin{align*}
\bar{u} &= \frac{\partial L}{\partial u} = \frac{\partial L}{\partial p} \frac{\partial p}{\partial u} = \bar{p} \sigma(1 - \sigma) \\
\bar{w} &= \frac{\partial L}{\partial w} = \frac{\partial L}{\partial u} \frac{\partial u}{\partial w} = \bar{u} x^T
\end{align*}
\]

We can do this in a combined way to see all terms together:

\[
\bar{w} = \frac{\partial L}{\partial p} \frac{\partial p}{\partial u} \frac{\partial u}{\partial w} = -\frac{1}{\sigma(w^T x)} \sigma(w^T x)(1 - \sigma(w^T x)) x^T
\]

This effectively shows gradient flow along path from \( L \) to \( w \)

Different Views of Equivalent Ideas
• **Backpropagation**: Recursive, modular algorithm for chain rule + gradient descent

• **When we move to vectors and matrices:**
  • Composition of functions (scalar)
  • Composition of functions (vectors/matrices)
  • Jacobian view of chain rule
  • Can view entire set of calculations as linear algebra operations (matrix-vector or matrix-matrix multiplication)

• **Automatic differentiation:**
  • Reduction of modules to simple operations we know (simple multiplication, etc.)
  • Automatically build computation graph in background as write code
  • Automatically compute gradients via backward pass