Topics:
• Neural Networks
• Backpropagation
• **Assignment 1 out!**
  • Due Feb 5\textsuperscript{th}
  • Start now, start now, start now!
  • Start now, start now, start now!
  • Start now, start now, start now!

• **Piazza**
  • Be active!!!
  • Extra credit!

• **Office hours**
  • [Assignment](#) (@41) and [matrix calculus](#) (@46)
Example with an image with **4 pixels**, and **3 classes** (*cat/dog/ship*)

Adapted from slides by Fei-Fei Li, Justin Johnson, Serena Yeung, from CS 231n
We can find the steepest descent direction by computing the derivative (gradient):

\[ f'(a) = \lim_{h \to 0} \frac{f(a + h) - f(a)}{h} \]

Steepest descent direction is the negative gradient

Intuitively: Measures how the function changes as the argument \( a \) changes by a small step size

As step size goes to zero

In Machine Learning: Want to know how the loss function changes as weights are varied

Can consider each parameter separately by taking partial derivative of loss function with respect to that parameter
This idea can be turned into an algorithm (gradient descent)

- Choose a model: $f(x, W) = Wx$
- Choose loss function: $L_i = |y - Wx_i|^2$
- Calculate partial derivative for each parameter: $\frac{\partial L}{\partial w_i}$
- Update the parameters: $w_i = w_i - \frac{\partial L}{\partial w_i}$
- Add learning rate to prevent too big of a step: $w_i = w_i - \alpha \frac{\partial L}{\partial w_i}$
- Repeat (from Step 3)
Often, we only compute the gradients across a small subset of data

- **Full Batch Gradient Descent**
  \[ L = \frac{1}{N} \sum L (f(x_i, W), y_i) \]

- **Mini-Batch Gradient Descent**
  \[ L = \frac{1}{M} \sum L (f(x_i, W), y_i) \]
  Where M is a *subset* of data

- We iterate over mini-batches:
  - Get mini-batch, compute loss, compute derivatives, and take a set
Gradient Descent

- Original $W$
- Negative gradient direction

[Link: http://demonstrations.wolfram.com/VisualizingTheGradientVector/]
For some functions, we can analytically derive the partial derivative.

**Example:**

Function: $f(w, x_i) = w^T x_i$

Loss: $(y_i - w^T x_i)^2$

Dataset: N examples (indexed by $k$)

Update Rule: $w_j \leftarrow w_j + 2\eta \sum_{k=1}^{N} \delta_k x_{kj}$

Derivation of Update Rule:

$L = \sum_{k=1}^{N} (y_k - w^T x_k)^2$

$\frac{\partial L}{\partial w_j} = \sum_{k=1}^{N} \frac{\partial}{\partial w_j} (y_k - w^T x_k)^2$

$= \sum_{k=1}^{N} 2(y_k - w^T x_k) \frac{\partial}{\partial w_j} (y_k - w^T x_k)$

$= -2 \sum_{k=1}^{N} \delta_k \frac{\partial}{\partial w_j} w^T x_k$

...where...

$\delta_k = y_k - w^T x_k$

$= -2 \sum_{k=1}^{N} \delta_k \frac{\partial}{\partial w_j} \sum_{i=1}^{m} w_i x_{ki}$

$= -2 \sum_{k=1}^{N} \delta_k x_{kj}$
If we add a non-linearity (sigmoid), derivation is more complex:

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

First, one can derive that: \( \sigma'(x) = \sigma(x)(1 - \sigma(x)) \)

\[
f(x) = \sigma \left( \sum_k w_k x_k \right)
\]

\[
L = \sum_i \left( y_i - \sigma \left( \sum_k w_k x_{ik} \right) \right)^2
\]

\[
\frac{\partial L}{\partial w_j} = \sum_i 2 \left( y_i - \sigma \left( \sum_k w_k x_{ik} \right) \right) \sigma' \left( \sum_k w_k x_{ik} \right) \frac{\partial}{\partial w_j} \sum_k w_k x_{ik}
\]

\[
= \sum_i -2 \left( y_i - \sigma \left( \sum_k w_k x_{ik} \right) \right) \sigma' \left( \sum_k w_k x_{ik} \right) \frac{\partial}{\partial w_j} \sum_k w_k x_{ik}
\]

\[
= \sum_i -2 \delta_i \sigma(d_i)(1 - \sigma(d_i)) x_{ij}
\]

where \( \delta_i = y_i - f(x_i) \) \( d_i = \sum w_k x_{ik} \)

The sigmoid perception update rule:

\[
w_j \leftarrow w_j + 2\eta \sum_{i=1}^{N} \delta_i \sigma_i(1 - \sigma_i)x_{ij}
\]

where \( \sigma_i = \sigma \left( \sum_{j=1}^{m} w_j x_{ij} \right) \)

\( \delta_i = y_i - \sigma_i \)
A linear classifier can be broken down into:

- Input
- A function of the input
- A loss function

It’s all just one function that can be decomposed into building blocks.

What Does a Linear Classifier Consist of?

- **Input**
- **Model**
  \[
  \frac{1}{1 + e^{-u}}
  \]
- **Loss Function**
  \[-\log(p)\]
The same two-layered neural network corresponds to adding another weight matrix.

We will prefer the linear algebra view, but use some terminology from neural networks (& biology).

\[ f(x, W_1, W_2) = \sigma(W_2 \sigma(W_1 x)) \]

*Figure adapted from slides by Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n*
Large (deep) networks can be built by adding more and more layers.

Three-layered neural networks can represent any function.

- The number of nodes could grow unreasonably (exponential or worse) with respect to the complexity of the function.

We will show them without edges:

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

Figure adapted from slides by Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Adding More Layers!
Demo

- http://playground.tensorflow.org
Computation Graphs
Functions can be made **arbitrarily complex** (subject to memory and computational limits), e.g.:

\[ f(x, W) = \sigma(W_5 \sigma(W_4 \sigma(W_3 \sigma(W_2 \sigma(W_1 x)))) \]

We can use **any type of differentiable function (layer)** we want!

- At the end, **add the loss function**

Composition can have **some structure**
The world is **compositional**!

We want our **model** to reflect this.

Empirical and theoretical evidence that it makes **learning complex functions** easier.

Note that **prior state of art engineered features** often had this compositionality as well.

- Pixels -> edges -> object parts -> objects

---

**Compositionality**

*Adapted from figure by Marc'Aurelio Ranzato, Yann LeCun*
We are learning **complex models** with significant amount of parameters (millions or billions)

How do we compute the gradients of the **loss** (at the end) with respect to **internal** parameters?

Intuitively, want to understand how **small changes** in weight deep inside are propagated to affect the **loss function** at the end

\[ \frac{\partial L}{\partial w_i} ? \]
Decomposing a Function

Given a library of simple functions

- \( \sin(x) \)
- \( \log(x) \)
- \( \cos(x) \)
- \( x^3 \)
- \( \exp(x) \)

Compose into a complicate function

\[- \log \left( \frac{1}{1 + e^{-w \cdot x}} \right) \]

\[ w \cdot x \rightarrow u \rightarrow \frac{1}{1 + e^{-u}} \rightarrow p \rightarrow -\log(p) \rightarrow L \]

Adapted from slides by: Marc'Aurelio Ranzato, Yann LeCun
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)
\[ f(x_1, x_2) = \ln(x_1) + x_1 x_2 - \sin(x_2) \]
\[- \log \left( \frac{1}{1 + e^{-w \cdot x}} \right) \]
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*
Step 1: Compute Loss on Mini-Batch: **Forward Pass**

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)

*Adapted from figure by Marc'Aurelio Ranzato, Yann LeCun*
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**

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

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

Adapted from figure by Marc'Aurelio Ranzato, Yann LeCun
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:** \( \frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} \cdot \frac{\partial y}{\partial x} \)
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!

\[ w_i = w_i - \alpha \frac{\partial L}{\partial w_i} \]
Given this computation graph, the training algorithm will:

- Calculate the current model’s outputs (called the **forward pass**)
- Calculate the gradients for each module (called the **backward pass**)

Backward pass is a recursive algorithm that:

- Starts at **loss function** where we know how to calculate the gradients
- Progresses back through the modules
- Ends in the **input layer** where we do not need gradients (no parameters)

This algorithm is called **backpropagation**

---

**Overview of Training**

Adapted from figure by Marc'Aurelio Ranzato, Yann LeCun
In the **backward pass**, we seek to calculate the gradients of the loss with respect to the module’s parameters

- Assume that we have the gradient of the loss with respect to the **module’s outputs** (given to us by upstream module)

- We will also pass the gradient of the loss with respect to the **module’s inputs**

  - This is not required for updating the module’s weights, but passes the gradients back to the previous module

**Problem:**

- We can compute local gradients:
  \[
  \frac{\partial L}{\partial h_{\ell}^l}, \frac{\partial L}{\partial h_{\ell-1}^l}, \frac{\partial L}{\partial W}
  \]

- We are given: \( \frac{\partial L}{\partial h_{\ell}^l} \)

- Compute: \( \left\{ \frac{\partial L}{\partial h_{\ell-1}^l}, \frac{\partial L}{\partial W} \right\} \)

*Adapted from figure by Marc'Aurelio Ranzato, Yann LeCun*
We can compute **local gradients:** \( \left\{ \frac{\partial h^\ell}{\partial h^{\ell-1}}, \frac{\partial h^\ell}{\partial W} \right\} \)

This is just the **derivative of our function** with respect to its parameters and inputs!

**Example:** If \( h^\ell = W h^{\ell-1} \)

then \( \frac{\partial h^\ell}{\partial h^{\ell-1}} = W \)

and \( \frac{\partial h^\ell}{\partial w_i} = h^{\ell-1,i} \) (a sparse matrix with in the \( i \)-th row)
We want to compute: \( \frac{\partial L}{\partial h^{\ell-1}} , \frac{\partial L}{\partial W} \)

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

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

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

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

---

**Computing the Gradients of Loss**

*Adapted from figure by Marc'Aurelio Ranzato, Yann LeCun*
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. \( x = -2, y = 5, z = -4 \)

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

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

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

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

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

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

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

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

\[
\begin{align*}
q &= x + y \\
\frac{\partial q}{\partial x} &= 1, \\
\frac{\partial q}{\partial y} &= 1
\end{align*}
\]

\[
\begin{align*}
f &= qz \\
\frac{\partial f}{\partial q} &= z, \\
\frac{\partial f}{\partial z} &= q
\end{align*}
\]

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

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

\[ e.g. \ x = -2, \ y = 5, \ z = -4 \]

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

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

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

Chain rule:

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

Upstream gradient \quad Local gradient
Backpropagation: a simple example

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

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

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

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

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

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

Chain rule:

Upstream gradient
Local gradient
Backpropagation: a simple example

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

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

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

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

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

Chain rule:

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

Diagram:

- Upstream gradient:
- Local gradient:

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
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:

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

Upstream gradient

Local gradient

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Backpropagation: a simple example
Backpropagation: a simple example
Patterns in backward flow
Q: What is an **add** gate?
Patterns in backward flow

**add gate**: gradient distributor
Patterns in backward flow

**Q:** What is a **max** gate?

**add gate:** gradient distributor

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Patterns in backward flow

\textbf{add} gate: gradient distributor

\textbf{max} gate: gradient router
Patterns in backward flow

**Q:** What is a *mul* gate?
Patterns in backward flow

**add** gate: gradient distributor

**max** gate: gradient router

**mul** gate: gradient switcher
Gradients add at branches
Duality in Fprop and Bprop

\[
\text{SUM} \quad \text{BPROP} \\
\text{COPY} \quad \text{COPY}
\]
Deep Learning = Differentiable Programming

• Computation = Graph
  – Input = Data + Parameters
  – Output = Loss
  – Scheduling = Topological ordering

• What do we need to do?
  – Generic code for representing the graph of modules
  – Specify modules (both forward and backward function)
Modularized implementation: forward / backward 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

\[(x, y, z \text{ are scalars})\]
Modularized implementation: forward / backward API

(x, y, z are scalars)

```python
class MultiplyGate(object):
    def forward(self, x, y):
        z = x * y
        self.x = x  # must keep these around!
        self.y = y
        return z
    def backward(self, dz):
        dx = self.y * dz  # [dz/dx * dL/dz]
        dy = self.x * dz  # [dz/dy * dL/dz]
        return [dx, dy]
```
Example: Caffe layers

<table>
<thead>
<tr>
<th>Branch</th>
<th>master</th>
<th>Directory</th>
<th>Commit Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>develop</td>
<td>layer.cpp</td>
<td>diamante layer headers</td>
<td>a year ago</td>
</tr>
<tr>
<td>develop</td>
<td>layer.cu</td>
<td>diamante layer headers</td>
<td>a year ago</td>
</tr>
<tr>
<td>develop</td>
<td>layer.e</td>
<td>diamante layer headers</td>
<td>a year ago</td>
</tr>
<tr>
<td>develop</td>
<td>layer.nin</td>
<td>diamante layer headers</td>
<td>a year ago</td>
</tr>
<tr>
<td>develop</td>
<td>layer rebate</td>
<td>diamante layer headers</td>
<td>a year ago</td>
</tr>
<tr>
<td>develop</td>
<td>layer rebate</td>
<td>enable dilated deserialization</td>
<td>a year ago</td>
</tr>
<tr>
<td>develop</td>
<td>layer rebate</td>
<td>Line default from jinuo for patch</td>
<td>a year ago</td>
</tr>
<tr>
<td>develop</td>
<td>layer rebate</td>
<td>Subclass multi GPU for NCC</td>
<td>a year ago</td>
</tr>
<tr>
<td>develop</td>
<td>layer rebate</td>
<td>Add missing layers besides equal signs in batch_norm_layer.cpp</td>
<td>a year ago</td>
</tr>
<tr>
<td>develop</td>
<td>layer rebate</td>
<td>diamante layer headers</td>
<td>a year ago</td>
</tr>
<tr>
<td>develop</td>
<td>layer rebate</td>
<td>diamante layer headers</td>
<td>a year ago</td>
</tr>
<tr>
<td>develop</td>
<td>layer rebate</td>
<td>diamante layer headers</td>
<td>a year ago</td>
</tr>
<tr>
<td>develop</td>
<td>layer rebate</td>
<td>diamante layer headers</td>
<td>a year ago</td>
</tr>
<tr>
<td>develop</td>
<td>layer rebate</td>
<td>diamante layer headers</td>
<td>a year ago</td>
</tr>
<tr>
<td>develop</td>
<td>layer rebate</td>
<td>Remove incorrect call of gmm in src in跨越式</td>
<td>a year ago</td>
</tr>
<tr>
<td>develop</td>
<td>layer rebate</td>
<td>Separation and specialization of ConvolutionalFolderLayer into ScanLayer</td>
<td>a year ago</td>
</tr>
<tr>
<td>develop</td>
<td>layer rebate</td>
<td>diamante layer headers</td>
<td>a year ago</td>
</tr>
<tr>
<td>develop</td>
<td>layer rebate</td>
<td>diamante layer headers</td>
<td>a year ago</td>
</tr>
<tr>
<td>develop</td>
<td>layer rebate</td>
<td>diamante layer headers</td>
<td>a year ago</td>
</tr>
<tr>
<td>develop</td>
<td>layer rebate</td>
<td>diamante layer headers</td>
<td>a year ago</td>
</tr>
<tr>
<td>develop</td>
<td>layer rebate</td>
<td>diamante layer headers</td>
<td>a year ago</td>
</tr>
<tr>
<td>develop</td>
<td>layer rebate</td>
<td>diamante layer headers</td>
<td>a year ago</td>
</tr>
<tr>
<td>develop</td>
<td>layer rebate</td>
<td>diamante layer headers</td>
<td>a year ago</td>
</tr>
<tr>
<td>develop</td>
<td>layer rebate</td>
<td>diamante layer headers</td>
<td>a year ago</td>
</tr>
<tr>
<td>develop</td>
<td>layer rebate</td>
<td>diamante layer headers</td>
<td>a year ago</td>
</tr>
<tr>
<td>develop</td>
<td>layer rebate</td>
<td>diamante layer headers</td>
<td>a year ago</td>
</tr>
<tr>
<td>develop</td>
<td>layer rebate</td>
<td>diamante layer headers</td>
<td>a year ago</td>
</tr>
<tr>
<td>develop</td>
<td>layer rebate</td>
<td>diamante layer headers</td>
<td>a year ago</td>
</tr>
<tr>
<td>develop</td>
<td>layer rebate</td>
<td>diamante layer headers</td>
<td>a year ago</td>
</tr>
<tr>
<td>develop</td>
<td>layer rebate</td>
<td>diamante layer headers</td>
<td>a year ago</td>
</tr>
<tr>
<td>develop</td>
<td>layer rebate</td>
<td>diamante layer headers</td>
<td>a year ago</td>
</tr>
<tr>
<td>develop</td>
<td>layer rebate</td>
<td>diamante layer headers</td>
<td>a year ago</td>
</tr>
<tr>
<td>develop</td>
<td>layer rebate</td>
<td>diamante layer headers</td>
<td>a year ago</td>
</tr>
<tr>
<td>develop</td>
<td>layer rebate</td>
<td>diamante layer headers</td>
<td>a year ago</td>
</tr>
</tbody>
</table>

Caffe is licensed under BSD 2-Clause

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Caffe Sigmoid Layer

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

\[(1 - \sigma(x)) \sigma(x) \] * top_diff (chain rule)

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
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!
- In practice, figure out Jacobians for simpler items (scalars, vectors), figure out pattern, and *slice* or index appropriate elements to create Jacobians
**Fully Connected (FC) Layer: Forward Function**

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

- Input: \( h^{\ell-1} \)
- Function: \( W h^{\ell-1} \)
- Output: \( h^\ell \)

**Parameters**
- \( W \)

**Dimensions**
- \( |h^\ell| \times 1 \)
- \( |h^\ell| \times |h^{\ell-1}| \)
- \( |h^{\ell-1}| \times 1 \)
Fully Connected (FC) Layer

Note doing this on full $W$ matrix would result in Jacobian tensor!

But it is sparse – each output only affected by corresponding weight row.

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

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