Lecture 2 Recap
Nearest Neighbor

NN classifier = dog
What is the performance on training data for NN classifier?
What classifier is more likely to perform best on test data?
Linear Regression

• Supervised learning
• Find a linear model that explains a target $y$ given the inputs $X$
Linear Regression

• A linear model is expressed in the form

\[ \hat{y}_i = \sum_{j=1}^{d} x_{ij} \theta_j = x_{i1} \theta_1 + x_{i2} \theta_2 + \cdots + x_{id} \theta_d \]
Logistic Regression

- Loss function

\[ \mathcal{L}(\hat{y}_i, y_i) = y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i) \]

- Cost function

\[ C(\theta) = -\frac{1}{n} \sum_{i=1}^{n} y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i) \]

Minimization

\[ \hat{y}_i = \sigma(x_i \theta) \]
Linear vs Logistic Regression

Prediction can exceed training sample range -> in case of classification [0;1] that's a real issue

Prediction is guaranteed to be within 0 and 1
\( x' = x - \epsilon \nabla_x f(x) \)

\( \nabla_x f(x) \)

Learning rate

\( x^* = \arg \min x f(x) \)
Introduction to Neural Networks
Neural Network

- Linear score function $f = Wx$

<table>
<thead>
<tr>
<th>plane</th>
<th>car</th>
<th>bird</th>
<th>cat</th>
<th>deer</th>
<th>dog</th>
<th>frog</th>
<th>horse</th>
<th>ship</th>
<th>truck</th>
</tr>
</thead>
</table>

On CIFAR-10

On ImageNet

Credit: Li/Karpathy/Johnson
Neural Network

• Linear score function \( f = Wx \)

• Neural network is a nesting of ‘functions’
  – 2-layers: \( f = W_2 \max(0, W_1 x) \)
  – 3-layers: \( f = W_3 \max(0, W_2 \max(0, W_1 x)) \)
  – 4-layers: \( f = W_4 \tanh(W_3, \max(0, W_2 \max(0, W_1 x))) \)
  – 5-layers: \( f = W_5 \sigma(W_4 \tanh(W_3, \max(0, W_2 \max(0, W_1 x)))) \)
  – ... up to hundreds of layers
Neural Network

1-layer network: \( f = Wx \)

2-layer network: \( f = W_2 \max(0, W_1x) \)

128 \times 128 = 16384

10

128 \times 128 = 16384

1000

10
Neurons

Linear function: $Wx + b$
Non-linearity (activation: $f(x)$)

Every neuron computes: $f(Wx + b)$
Net of Neurons

\[ f(W_{0,0}x + b_{0,0}) \]
\[ f(W_{0,1}x + b_{0,1}) \]
\[ f(W_{0,2}x + b_{0,2}) \]
\[ f(W_{0,3}x + b_{0,3}) \]
\[ f(W_{1,0}x + b_{1,0}) \]
\[ f(W_{1,1}x + b_{1,1}) \]
\[ f(W_{1,2}x + b_{1,2}) \]
\[ f(W_{2,0}x + b_{2,0}) \]
Neural Network

input layer

hidden layer

output layer

Credit: Li/Karpathy/Johnson
Neural Network

2-layer network: \( f = W_2 \max(0, W_1 x) \)

- Input Layer: \( 128 \times 128 = 16384 \)
- Hidden Layer: 1000
- Output Layer: 10

Input Layer  Hidden Layer  Output Layer
Neural Network
Neural Network

\[ f = W_3 \cdot (W_2 \cdot (W_1 \cdot x)) \]

Why activation functions?
Why not just concatenate?
Would be much cheaper to compute....
Activation Functions

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

tanh: \( \tanh(x) \)

ReLU: \( \max(0, x) \)

Leaky ReLU: \( \max(0.1x, x) \)

Parametric ReLU: \( \max(ax, x) \)

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

ELU \( f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha(e^x - 1) & \text{if } x \leq 0 \end{cases} \)
Neural Network

Why organize a neural network into layers?
Neural Network

• Summary
  – Given a dataset with ground truth training pairs $[x_i; y_i]$,
  
  – Find optimal weights $W$ using stochastic gradient descent, such that the loss function is minimized
  
  – Compute gradients with backpropagation (use batch-mode; more later)
  
  – Iterate many times over training set (SGD; more later)
Artificial neural networks: inspired but not even close to the brain!
It's much more complex than simple linearity + activations
Great for the media and news articles 😊
Google's artificial intelligence computer 'no longer constrained by limits of human knowledge'
Computational Graphs
Computational Graphs

• Neural network is a computational graph
  – It has compute nodes
  – It has edges that connect nodes
  – It is directional
  – It is organized in ‘layers’
Computational Graphs

• $f(x, y, z) = (x + y) \cdot z$
Computational Graphs

Another view of GoogleNet’s architecture.

Prof. Leal-Taixé and Prof. Niessner
Evaluation: Forward Pass

- \( f(x, y, z) = (x + y) \cdot z \)

Initialization \( x = 1, y = -3, z = 4 \)

\[
\begin{align*}
&\text{sum} \quad d = -2 \\
&\times \quad f = -8
\end{align*}
\]
The Flow of Gradients

\[ z = f(x, y) \]
The Flow of Gradients

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

Activations

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

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

generates

f

Activation function

"local gradients"
Backpropagation
Backprop: Forward Pass

- \( f(x, y, z) = (x + y) \cdot z \)

Initialization: \( x = 1, y = -3, z = 4 \)

\( f = -8 \)
Backprop: Backward Pass

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

with \( x = 1, y = -3, z = 4 \)

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

\[ f = d \cdot z \quad \frac{\partial f}{\partial d} = z, \quad \frac{\partial f}{\partial z} = d \]

What is \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)?
Backprop: Backward Pass

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

with \( x = 1, y = -3, z = 4 \)

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

\[ f = d \cdot z \quad \frac{\partial f}{\partial d} = z, \quad \frac{\partial f}{\partial z} = d \]

What is \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)?
Backprop: Backward Pass

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

with \( x = 1, y = -3, z = 4 \)

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

\[ f = d \cdot z \quad \frac{\partial f}{\partial d} = z, \quad \frac{\partial f}{\partial z} = d \]

What is \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} ? \)
Backprop: Backward Pass

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

with \( x = 1, y = -3, z = 4 \)

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

\[ f = d \cdot z \quad \frac{\partial f}{\partial d} = z, \quad \frac{\partial f}{\partial z} = d \]

What is \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)?
$f(x, y, z) = (x + y) \cdot z$

with $x = 1, y = -3, z = 4$

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

\[
\begin{align*}
f &= d \cdot z \\
\frac{\partial f}{\partial d} &= z, \quad \frac{\partial f}{\partial z} = d
\end{align*}
\]

What is $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$?

Chain Rule:

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

\[
\begin{align*}
\frac{\partial f}{\partial y} &= 4 \cdot 1 = 4
\end{align*}
\]
Backprop: Backward Pass

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

with \( x = 1, y = -3, z = 4 \)

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

\[ f = d \cdot z \quad \frac{\partial f}{\partial d} = z \quad \frac{\partial f}{\partial z} = d \]

What is \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)?
The Flow of Gradients

Activation function

\[ z = f(x, y) \]
The Flow of Gradients

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

Activations

Activation function

Prof. Leal-Taixé and Prof. Niessner
Backprop

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

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

\[ f(w_0, x_0, w_1, x_1, b) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + b)}} \]

\[
\begin{align*}
    f(x) &= e^x & \Rightarrow & \quad \frac{\partial f}{\partial x} = e^x \\
    f_a(x) &= ax & \Rightarrow & \quad \frac{\partial f_a}{\partial x} = a \\
    f_c(x) &= c + x & \Rightarrow & \quad \frac{\partial f_c}{\partial x} = 1
\end{align*}
\]
Backprop

\[ f(w_0, x_0, w_1, x_1, b) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + b)}} \]

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

\[ f_a(x) = ax \quad \Rightarrow \quad \frac{df_a}{dx} = a \]

\[ f_c(x) = c + x \quad \Rightarrow \quad \frac{df_c}{dx} = 1 \]
Backprop

\[ f(w_0, x_0, w_1, x_1, b) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + b)}} \]

- 1.37² \cdot (1.00) = -0.53

\[
egin{align*}
\frac{\partial f}{\partial w_0} &= x_0 \\
\frac{\partial f}{\partial x_0} &= w_0 \\
\frac{\partial f}{\partial w_1} &= x_1 \\
\frac{\partial f}{\partial x_1} &= w_1 \\
\frac{\partial f}{\partial b} &= -(w_0x_0 + w_1x_1)
\end{align*}
\]

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

Prof. Leal-Taixé and Prof. Niessner
\[ f(w_0, x_0, w_1, x_1, b) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + b)}} \]

**Backprop**

\[ f(x) = e^x \quad \Rightarrow \quad \frac{\partial f}{\partial x} = e^x \]

\[ f_a(x) = ax \quad \Rightarrow \quad \frac{\partial f_a}{\partial x} = a \]

\[ f(x) = \frac{1}{x} \quad \Rightarrow \quad \frac{\partial f}{\partial x} = -\frac{1}{x^2} \]

\[ f_c(x) = c + x \quad \Rightarrow \quad \frac{\partial f_c}{\partial x} = 1 \]
Backprop

\[ f(w_0, x_0, w_1, x_1, b) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + b)}} \]

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

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

\[ f_c(x) = c + x \quad \rightarrow \quad \frac{df_c}{dx} = 1 \]

Prof. Leal-Taixé and Prof. Niessner
Backprop

\[ f(w_0, x_0, w_1, x_1, b) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + b)}} \]

\[ f(x) = e^x \quad \Rightarrow \quad \frac{\partial f}{\partial x} = e^x \]

\[ f_a(x) = ax \quad \Rightarrow \quad \frac{\partial f_a}{\partial x} = a \]

\[ f(x) = \frac{1}{x} \quad \Rightarrow \quad \frac{\partial f}{\partial x} = -\frac{1}{x^2} \]

\[ f_c(x) = c + x \quad \Rightarrow \quad \frac{\partial f_c}{\partial x} = 1 \]
Backprop

\[ f(w_0, x_0, w_1, x_1, b) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + b)}} \]

\[ f(x, y) = x + y \quad \Rightarrow \quad \frac{\partial f}{\partial x} = 1, \frac{\partial f}{\partial y} = 1 \]

\[ 1 \cdot (0.2) = 0.2 \]

\[ 1 \cdot (0.2) = 0.2 \]

\[ f(x) = e^x \quad \Rightarrow \quad \frac{\partial f}{\partial x} = e^x \]

\[ f(x) = \frac{1}{x} \quad \Rightarrow \quad \frac{\partial f}{\partial x} = -\frac{1}{x^2} \]

\[ f_a(x) = ax \quad \Rightarrow \quad \frac{\partial f_a}{\partial x} = a \]

\[ f_c(x) = c + x \quad \Rightarrow \quad \frac{\partial f_c}{\partial x} = 1 \]
Backprop

\[ f(w_0, x_0, w_1, x_1, b) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + b)}} \]

\[ f(x, y) = x + y \quad \Rightarrow \quad \frac{\partial f}{\partial x} = 1, \frac{\partial f}{\partial y} = 1 \]

\[ 1 \cdot (0.2) = 0.2 \]

\[ 1 \cdot (0.2) = 0.2 \]

\[ f(x) = e^x \quad \Rightarrow \quad \frac{\partial f}{\partial x} = e^x \]

\[ f(x) = \frac{1}{x} \quad \Rightarrow \quad \frac{\partial f}{\partial x} = -\frac{1}{x^2} \]

\[ f_{a}(x) = ax \quad \Rightarrow \quad \frac{\partial f_{a}}{\partial x} = a \]

\[ f_{c}(x) = c + x \quad \Rightarrow \quad \frac{\partial f_{c}}{\partial x} = 1 \]
Backprop

\[
f(w_0, x_0, w_1, x_1, b) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + b)}}
\]

\[
f(x, y) = x \cdot y \quad \rightarrow \quad \frac{\partial f}{\partial x} = y, \quad \frac{\partial f}{\partial y} = x
\]

\[
2 \cdot (0.2) = 3.9
\]

\[
-1 \cdot (0.2) = -0.2
\]

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

\[
f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{\partial f}{\partial x} = -\frac{1}{x^2}
\]

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

\[
f_c(x) = c + x \quad \rightarrow \quad \frac{\partial f_c}{\partial x} = 1
\]

Prof. Leal-Taixé and Prof. Niessner
Backprop

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

\[ f(x, y) = x \cdot y \quad \Rightarrow \quad \frac{\partial f}{\partial x} = y, \frac{\partial f}{\partial y} = x \]

\[-3.00 \cdot (0.2) = -0.59\]
\[-2.00 \cdot (0.2) = -0.39\]

\[ f(x) = e^x \quad \Rightarrow \quad \frac{\partial f}{\partial x} = e^x \]
\[ f(x) = \frac{1}{x} \quad \Rightarrow \quad \frac{\partial f}{\partial x} = -\frac{1}{x^2} \]

\[ f_a(x) = ax \quad \Rightarrow \quad \frac{\partial f_a}{\partial x} = a \]
\[ f_c(x) = c + x \quad \Rightarrow \quad \frac{\partial f_c}{\partial x} = 1 \]
\[ f(w_0, x_0, w_1, x_1, b) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + b)}} \]

Why use a compute graph in the first place?

\[ \sigma(x) = \frac{1}{1 + e^{-x}} \quad \Rightarrow \quad \frac{\partial \sigma(x)}{\partial x} = \frac{e^{-x}}{(1 + e^{-x})^2} = (1 - \sigma(x))\sigma(x) \]
What happens if there are multiple outputs in a compute node?
What happens if there are loops in the graph?
Computational Graph

Combining nodes:
Linear activation node + hinge loss + regularization

\[ f = Wx \]
\[ L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1) \]
Computational Graph: Logistic Regression

- Loss function

\[ \mathcal{L}(\hat{y}_i, y_i) = y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i) \]

- Cost function

\[ C(\theta) = -\frac{1}{n} \sum_{i=1}^{n} y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i) \]

Minimization

\[ \hat{y}_i = \sigma(x_i \theta) \]

More Next Lecture!
Implementation of Compute Graph

1) forward

2) backwards

class ComputationalGraph(object):
    #...
    def forward(inputs):
        #1. [pass inputs to input nodes]
        #2. forward traverse the computational graph
        for node in self.graph.nodes_topologically_sorted():
            node.forward()
            # forward intermediates / loss
        return loss # final node returns loss
    def backward():
        for node in self.graph.nodes_topologically_sorted_reverse():
            node.backward() # apply chainrule
            # backward intermediate derivatives
        return inputs_gradients
Implementation of Nodes

- Forward and backward pass of MulNode

```
class MulNode(object):
    def forward(x, y):
        z = x * y
        return z
    def backward(dz, x, y):
        dx = y * dz  # [dz/dx * dL/dz]
        dy = x * dz  # [dz/dy * dL/dz]
        return [dx, dy]
```

all values are scalars

Issue?
Implementation of Nodes

- Forward and backward pass of MulNode

\[ x \times y \times z \]

Cache results of forward pass
- \( \rightarrow \) faster runtime for backward pass

```
class MulNode(object):
    def forward(x, y):
        z = x * y
        self.x = x
        self.y = y
        return z
    def backward(dz):
        dx = self.y * dz  # [dz/dx * dL/dz]
        dy = self.x * dz  # [dz/dy * dL/dz]
        return [dx, dy]
```

All values are scalars
<table>
<thead>
<tr>
<th>File Name</th>
<th>Description</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>LayerNormalization.lua</td>
<td>Adding layer normalization</td>
<td>2 months</td>
</tr>
<tr>
<td>LeakyReLU.lua</td>
<td>Add THNN conversion of ELU, LeakyReLU, LogSigmoid, LogSoftMax, Looku...</td>
<td>a year ago</td>
</tr>
<tr>
<td>Linear.lua</td>
<td>Fix shared function override for specific modules</td>
<td>4 months</td>
</tr>
<tr>
<td>Log.lua</td>
<td>add nn.Inc &amp; nn.Scale</td>
<td>a year ago</td>
</tr>
<tr>
<td>LogSigmoid.lua</td>
<td>lazy init</td>
<td>a year ago</td>
</tr>
<tr>
<td>LogSoftMax.lua</td>
<td>Add THNN conversion of (ELU, LeakyReLU, LogSigmoid, LogSoftMax, Looku...)</td>
<td>a year ago</td>
</tr>
<tr>
<td>LookupTable.lua</td>
<td>Fix shared function override for specific modules</td>
<td>4 months</td>
</tr>
<tr>
<td>MM.lua</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MSECriterion.lua</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MV.lua</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MapTable.lua</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MarginCriterion.lua</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MarginRankingCriterion.lua</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MaskedSelect.lua</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Max.lua</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Maxout.lua</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mean.lua</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Min.lua</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MixtureTable.lua</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Module.lua</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mul.lua</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MulConstant.lua</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MultCriterion.lua</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reshape.lua</td>
<td>Better <em>loststring</em> and cleans formatting</td>
<td>a year ago</td>
</tr>
<tr>
<td>Select.lua</td>
<td>Adds negative dim arguments</td>
<td>11 months</td>
</tr>
<tr>
<td>SelectTable.lua</td>
<td>allow SelectTable to accept input that contains tables of things that...</td>
<td>2 months</td>
</tr>
<tr>
<td>Sequential.lua</td>
<td>Improve error handling</td>
<td>a year ago</td>
</tr>
<tr>
<td>Sigmoid.lua</td>
<td></td>
<td>a year ago</td>
</tr>
<tr>
<td>SmoothL1Criterion.lua</td>
<td></td>
<td>a year ago</td>
</tr>
<tr>
<td>SoftMarginCriterion.lua</td>
<td></td>
<td>a year ago</td>
</tr>
<tr>
<td>SoftMax.lua</td>
<td></td>
<td>a year ago</td>
</tr>
<tr>
<td>SoftMin.lua</td>
<td></td>
<td>a year ago</td>
</tr>
<tr>
<td>SoftPlus.lua</td>
<td></td>
<td>a year ago</td>
</tr>
<tr>
<td>SoftShrink.lua</td>
<td></td>
<td>a year ago</td>
</tr>
<tr>
<td>SoftSign.lua</td>
<td></td>
<td>a year ago</td>
</tr>
<tr>
<td>SparseJacobian.lua</td>
<td>Fix various unused variables in nn</td>
<td>3 years</td>
</tr>
<tr>
<td>SparseLinear.lua</td>
<td>Fixing sparse linear race condition</td>
<td>a year ago</td>
</tr>
<tr>
<td>SpatialAdaptiveAveragePooling.lua</td>
<td>Add SpatialAdaptiveAveragePooling.</td>
<td>4 months</td>
</tr>
<tr>
<td>SpatialAdaptiveMaxPooling.lua</td>
<td>Indices for nn.</td>
<td>7 months</td>
</tr>
<tr>
<td>SpatialAutoCropMSECriterion.lua</td>
<td>fix local / global var leaks</td>
<td>4 months</td>
</tr>
</tbody>
</table>
Torch: MulConstant

\[ f(x) = aX \]

**Init()**

```python
function MulConstant: _init(constant_scalar, ip)
    parent._init(self)
    assert(type(constant_scalar) == 'number', 'input is not scalar!')
    self.constant_scalar = constant_scalar
    -- default for inplace is false
    self.inplace = ip or false
    if (ip and type(ip) == 'boolean') then
        error('in-place flag must be boolean')
    end
    end
```

**Forward()**

```python
function MulConstant: updateOutput(input)
    if self.inplace then
        input:mul(self.constant_scalar)
        self.output:set(input)
    else
        self.output:resizeAs(input)
        self.output:copy(input)
        self.output:mul(self.constant_scalar)
    end
    return self.output
end
```

**Backward()**

```python
function MulConstant: updateGradInput(input, gradOutput)
    if self.gradInput then
        if self.inplace then
            gradOutput:mul(self.constant_scalar)
            self.gradInput:set(gradOutput)
            -- restore previous input value
            input:div(self.constant Scalar)
        else
            self.gradInput:resizeAs(gradOutput)
            self.gradInput:copy(gradOutput)
            self.gradInput:mul(self.constant Scalar)
        end
    end
    return self.gradInput
end
```
Caffee: Layers (GitHub)

- absval_layer.cpp
- absval_layer.cu
- accuracy_layer.cpp
- argmax_layer.cpp
- base_conv_layer.cpp
- base_data_layer.cpp
- batch_norm_layer.cpp
- batch_reindex_layer.cpp
- bnll_layer.cpp
- bnll_layer.cu
- concat_layer.cpp
- concat_layer.cu
- contrastive_loss_layer.cpp
- contrastive_loss_layer.cu
- conv_layer.cpp
- pooling_layer.cpp
- pooling_layer.cu
- power_layer.cpp
- power_layer.cu
- prelu_layer.cpp
- prelu_layer.cu
- reduction_layer.cpp
- reduction_layer.cu
- relu_layer.cpp
- relu_layer.cu
- reshape_layer.cpp
- sigmoid_cross_entropy_loss_layer.cpp
- sigmoid_cross_entropy_loss_layer.cu
- sigmoid_layer.cpp
- sigmoid_layer.cu
- softmax_layer.cpp
- softmax_layer.cu
- slice_layer.cpp
- slice_layer.cu
Caffe: Sigmoid Layer

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

Forward()

\[ \sigma'(x) = (1 - \sigma(x)) \sigma(x) \]

Backward()
Vectorized Operations

What if \( x, y, z \), are vectors?

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

Activations

x

y

z

Activation function

"local gradients"

gradients

Prof. Leal-Taixé and Prof. Niessner
Vectorized Operations

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

\[ x = [x_1, x_2, ..., x_n] \]

\[ y = [y_1, y_2, ..., y_n] \]

\[ z = [z_1, z_2, ..., z_n] \]

Activations

Activation function

“local gradients”

These are now vectors

gradients

Prof. Leal-Taixé and Prof. Niessner
Vectorized Operations

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

Activation function

Gradients are now also vectors!

\[ x = [x_1, x_2, \ldots, x_n] \]

\[ y = [y_1, y_2, \ldots, y_n] \]

\[ z = [z_1, z_2, \ldots, z_n] \]
Vectorized Operations

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

\[x = [x_1, x_2, \ldots, x_n]\]

\[y = [y_1, y_2, \ldots, y_n]\]

Activations

Activation function

Need derivative of every output element w.r.t. every input element!

\[z = [z_1, z_2, \ldots, z_n]\]

"local gradients"
Vectorized Operations

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

\[ x = [x_1, x_2, \ldots, x_n] \]
\[ y = [y_1, y_2, \ldots, y_n] \]
\[ z = [z_1, z_2, \ldots, z_n] \]

Jacobian Matrix:

\[
\begin{bmatrix}
\frac{\partial z_1}{\partial x_1} & \ldots & \frac{\partial z_1}{\partial x_n} \\
\vdots & \ddots & \vdots \\
\frac{\partial z_n}{\partial x_1} & \ldots & \frac{\partial z_n}{\partial x_n}
\end{bmatrix}
\]

"local gradients"

Prof. Leal-Taixé and Prof. Niessner
Vectorized Operations

$$\frac{\partial L}{\partial x} = \frac{\partial z}{\partial x} \frac{\partial L}{\partial z}$$

$$x = [x_1, x_2, ..., x_n]$$

$$y = [y_1, y_2, ..., y_n]$$

Activation function

"local gradients"

$$z = [z_1, z_2, ..., z_n]$$

Jacobian Matrix:

$$\begin{bmatrix}
\frac{\partial z_1}{\partial y_1} & \cdots & \frac{\partial z_1}{\partial y_n} \\
\vdots & \ddots & \vdots \\
\frac{\partial z_n}{\partial y_1} & \cdots & \frac{\partial z_n}{\partial y_n}
\end{bmatrix}$$

$$\frac{\partial L}{\partial z}$$

gradients
Vectorized Operations

Assuming input and output $\in \mathbb{R}^{4096}$

\[ x = [x_1, x_2, \ldots, x_n] \]
\[ x \in \mathbb{R}^{4096} \]

\[ z = [z_1, z_2, \ldots, z_n] \]
\[ z \in \mathbb{R}^{4096} \]

What is the size of the Jacobian?

\[ \text{dim}(J) = 4096 \times 4096 \]

Jacobian Matrix:

\[
\begin{bmatrix}
\frac{\partial z_1}{\partial y_1} & \ldots & \frac{\partial z_1}{\partial y_n} \\
\frac{\partial z_2}{\partial y_1} & \ldots & \frac{\partial z_2}{\partial y_n} \\
\vdots & \ddots & \vdots \\
\frac{\partial z_n}{\partial y_1} & \ldots & \frac{\partial z_n}{\partial y_n}
\end{bmatrix}
\]
Vectorized Operations

Jacobian Matrix:

\[ \begin{bmatrix}
\frac{\partial z_1}{\partial y_1} & \cdots & \frac{\partial z_1}{\partial y_n} \\
\frac{\partial z_2}{\partial y_1} & \cdots & \frac{\partial z_2}{\partial y_n} \\
\vdots & \ddots & \vdots \\
\frac{\partial z_n}{\partial y_1} & \cdots & \frac{\partial z_n}{\partial y_n}
\end{bmatrix} \]

How efficient is that:
- \( \text{dim}(J) = 4096 \times 4096 = 16.78 \text{ mio} \)
- Assuming floats (i.e., 4 bytes / elem)
- \( \to 64 \text{ MB} \)

Typically, networks are run in batches:
- Assuming mini-batch size of 16
- \( \to \text{dim}(J) = (16 \cdot 4096) \times (16 \cdot 4096) = 4295 \text{ mio} \)
- \( \to 16.384 \text{MB} = \textbf{16GB} \)

How to handle this?

Prof. Leal-Taixé and Prof. Niessner
Neural Networks
Neural Networks

• Are compute graphs
• Goal: for given train set, find optimal weights

• Optimization using gradient-based solvers
  – Many options (more in the next lectures)

• Gradients are computed via backpropagation
  – Nice because can easily modularize complex functions
Administrative Things

• **This week: tutorial on April 26\textsuperscript{th}**
  – Python tutorial
  – Introduction to submission system

• **Next Lecture on April 30\textsuperscript{th}**
  – Optimization and Regularization
  – More on neural networks 😊

• **Next week: first exercise**
  – May 3\textsuperscript{rd} -> start of exercise 1
See you next week!