CS 188: Artificial Intelligence

Deep Learning II

Instructors: Pieter Abbeel & Anca Dragan --- University of California, Berkeley

[These slides were created by Dan Klein, Pieter Abbeel, Anca Dragan for CS188 Intro to AI at UC Berkeley. All CS188 materials are available at http://ai.berkeley.edu.]
Fun Neural Net Demo Site

- Demo-site:
  - http://playground.tensorflow.org/
Objective: Classification Accuracy

\[ l^{acc}(w) = \frac{1}{m} \sum_{i=1}^{m} \left( \text{sign}(w^\top f(x^{(i)})) == y^{(i)} \right) \]

Issue: many plateaus  \( \rightarrow \) how to measure incremental progress?
**Soft-Max**

- Score for $y=1$: $w^T f(x)$  
  Score for $y=-1$: $-w^T f(x)$

- Probability of label:
  \[ p(y = 1|f(x); w) = \frac{e^{w^T f(x^{(i)})}}{e^{w^T f(x^{(i)})} + e^{-w^T f(x^{(i)})}} \]
  \[ p(y = -1|f(x); w) = \frac{e^{-w^T f(x^{(i)})}}{e^{w^T f(x^{(i)})} + e^{-w^T f(x^{(i)})}} \]

- Objective:
  \[ l(w) = \prod_{i=1}^{m} p(y = y^{(i)}|f(x^{(i)}); w) \]

- Log:
  \[ ll(w) = \sum_{i=1}^{m} \log p(y = y^{(i)}|f(x^{(i)}); w) \]
Two-Layer Neural Network

\[ \Sigma f_1 + \Sigma f_2 + \Sigma f_3 \]

\[ z \rightarrow \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \]
N-Layer Neural Network
Multi-class Softmax

- 3-class softmax – classes A, B, C
  - 3 weight vectors: \( w_A, w_B, w_C \)

- Probability of label A: (similar for B, C)
  \[
  p(y = A | f(x); w) = \frac{e^{w_A^T f(x)}}{e^{w_A^T f(x)} + e^{w_B^T f(x)} + e^{w_C^T f(x)}}
  \]

- Objective:
  \[
  l(w) = \prod_{i=1}^{m} p(y = y^{(i)} | f(x^{(i)}; w))
  \]

- Log:
  \[
  ll(w) = \sum_{i=1}^{m} \log p(y = y^{(i)} | f(x^{(i)}; w))
  \]
Multi-class Two-Layer Neural Network

\[ z \rightarrow \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \]
Steepest Descent

- **Idea:**
  - Start somewhere
  - Repeat: Take a step in the steepest descent direction

Figure source: Mathworks
What is the Steepest Descent Direction?

\[
\min_{\Delta : \Delta_1^2 + \Delta_2^2 \leq \epsilon} \quad g(w + \Delta)
\]

- First-Order Taylor Expansion: \( g(w + \Delta) \approx g(w) + \frac{\partial g}{\partial w_1} \Delta_1 + \frac{\partial g}{\partial w_2} \Delta_2 \)

- Steepest Descent Direction:

\[
\min_{\Delta : \Delta_1^2 + \Delta_2^2 \leq \epsilon} \quad \frac{\partial g}{\partial w_1} \Delta_1 + \frac{\partial g}{\partial w_2} \Delta_2
\]

- Recall:

\[
\min_{a : \|a\| \leq \epsilon} \quad a^\top b \quad \Rightarrow \quad a = -b \frac{\epsilon}{\|b\|}
\]

- Hence, solution:

\[
-\nabla g \frac{\epsilon}{\|\nabla g\|}
\]

\[
\nabla g = \begin{bmatrix}
\frac{\partial g}{\partial w_1} \\
\frac{\partial g}{\partial w_2}
\end{bmatrix}
\]
Generally, Steepest Direction

- Steepest Direction = direction of the gradient

\[ \nabla g = \begin{bmatrix} \frac{\partial g}{\partial w_1} \\ \frac{\partial g}{\partial w_2} \\ \vdots \\ \frac{\partial g}{\partial w_n} \end{bmatrix} \]
Optimization Procedure 2: Momentum

- **Gradient Descent**
  - Init: $w$
  - For $i = 1, 2, ...$
    
    $$w \leftarrow w - \alpha \times \nabla g(w)$$

- **Momentum**
  - Init: $w$
  - For $i = 1, 2, ...$
    
    $$v \leftarrow \mu \times v - \alpha \times \nabla g(w)$$
    $$w \leftarrow w + v$$

- Physical interpretation as ball rolling down the loss function + friction ($\mu$ coefficient).
- $\mu = \text{usually } \sim 0.5, 0.9, \text{ or } 0.99$ (Sometimes annealed over time, e.g. from 0.5 -> 0.99)
Hyperparameters: Random Search vs. Grid Search

Random Search for Hyper-Parameter Optimization
Bergstra and Bengio, 2012
ConvNets are everywhere

Classification

Retrieval

[Krizhevsky 2012]
ConvNets are everywhere

Detection

Segmentation

[Faster R-CNN: Ren, He, Girshick, Sun 2015]

[Farabet et al., 2012]
ConvNets are everywhere

self-driving cars

NVIDIA Tegra X1
ConvNets are everywhere

[Simonyan et al. 2014]

[Goodfellow 2014]
ConvNets are everywhere

[Toshev, Szegedy 2014]

[Mnih 2013]
ConvNets are everywhere

[Ciresan et al. 2013]

[Sermanet et al. 2011]
[Ciresan et al.]
ConvNets are everywhere

[Denil et al. 2014]

[Turaga et al., 2010]
ConvNets are everywhere

*Whale recognition, Kaggle Challenge*

*Mnih and Hinton, 2010*
<table>
<thead>
<tr>
<th>Describes without errors</th>
<th>Describes with minor errors</th>
<th>Somewhat related to the image</th>
<th>Unrelated to the image</th>
</tr>
</thead>
<tbody>
<tr>
<td>A person riding a motorcycle on a dirt road.</td>
<td>Two dogs play in the grass.</td>
<td>A skateboarder does a trick on a ramp.</td>
<td>A dog is jumping to catch a frisbee.</td>
</tr>
<tr>
<td>A group of young people playing a game of frisbee.</td>
<td>Two hockey players are fighting over the puck.</td>
<td>A little girl in a pink hat is blowing bubbles.</td>
<td>A refrigerator filled with lots of food and drinks.</td>
</tr>
<tr>
<td>A herd of elephants walking across a dry grass field.</td>
<td>A close up of a cat laying on a couch.</td>
<td>A red motorcycle parked on the side of the road.</td>
<td>A yellow school bus parked in a parking lot.</td>
</tr>
</tbody>
</table>

[Vinyals et al., 2015]
reddit.com/r/deepdream
Remaining Pieces

- Optimizing machine learning objectives:
  - Stochastic Descent
  - Mini-batches
- Improving generalization
  - Drop-out
- Activation functions
- Initialization and batch normalization
- Computing the gradient $\nabla g(w)$
  - Backprop
  - Gradient checking
Mini-batches and Stochastic Gradient Descent

- Typical objective:

\[
ll(w) = \frac{1}{m} \sum_{i=1}^{m} \log p(y = y^{(i)} | f(x^{(i)}); w)
\]

= average log-likelihood of label given input

\[
\approx \frac{1}{k} \sum_{i=1}^{k} \log p(y = y^{(i)} | f(x^{(i)}); w)
\]

= estimate based on mini-batch 1...k

- Mini-batch gradient descent: compute gradient on mini-batch (+ cycle over mini-batches: 1..k, k+1...2k, ... ; make sure to randomize permutation of data!)

- Stochastic gradient descent: \( k = 1 \)
Remaining Pieces

- Optimizing machine learning objectives:
  - Stochastic Descent
  - Mini-batches

- Improving generalization
  - Drop-out

- Activation functions

- Initialization and batch normalization

- Computing the gradient $\nabla g(w)$
  - Gradient checking
  - Backprop
Regularization: **Dropout**

“randomly set some neurons to zero in the forward pass”

(a) Standard Neural Net  
(b) After applying dropout.  

[Srivastava et al., 2014]
Waaaait a second…
How could this possibly be a good idea?
Waaaait a second…
How could this possibly be a good idea?

Forces the network to have a redundant representation.
Another interpretation:

Dropout is training a large ensemble of models (that share parameters).

Each binary mask is one model, gets trained on only ~one datapoint.
At test time....

Ideally:
want to integrate out all the noise

Sampling-based approximation:
do many forward passes with
different dropout masks, average all predictions
At test time….
Can in fact do this with a single forward pass! (approximately)
Leave all input neurons turned on (no dropout).

Q: Suppose that with all inputs present at test time the output of this neuron is $x$.

What would its output be during training time, in expectation? (e.g. if $p = 0.5$)
At test time….
Can in fact do this with a single forward pass! (approximately)
Leave all input neurons turned on (no dropout).

During test: \( a = w_0 x + w_1 y \)

During train:
\[
E[a] = \frac{1}{4} \left( w_0^2 x + w_1^2 y + w_0 x + w_1 y \right)
\]
\[= \frac{1}{4} \left( 2 w_0 x + 2 w_1 y \right)\]
\[= \frac{1}{2} \left( w_0 x + w_1 y \right)\]
At test time….
Can in fact do this with a single forward pass! (approximately)
Leaves all input neurons turned on (no dropout).

During test: \( a = w_0 x + w_1 y \)

During train:
\[
E[a] = \frac{1}{4} \left( w_0^2 + w_1^2 \right) \]
\[
= \frac{1}{2} \left( w_0 x + w_1 y \right)
\]

With \( p=0.5 \), using all inputs in the forward pass would inflate the activations by 2x from what the network was “used to” during training!

\( \Rightarrow \) Have to compensate by scaling the activations back down by \( \frac{1}{2} \).
Remaining Pieces

- Optimizing machine learning objectives:
  - Stochastic Descent
  - Mini-batches
- Improving generalization
  - Drop-out
- Activation functions
- Initialization and batch normalization
- Computing the gradient $\nabla g(w)$
  - Gradient checking
  - Backprop
Activation Functions

Sigmoid

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

\text{tanh} \quad \tanh(x)

\text{ReLU} \quad \max(0, x)

\text{Leaky ReLU} \quad \max(0.1x, x)

\text{Maxout} \quad \max(w_1^Tx + b_1, w_2^Tx + b_2)

\text{ELU} \quad f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha(\exp(x) - 1) & \text{if } x \leq 0 \end{cases}$
Remaining Pieces

- Optimizing machine learning objectives:
  - Stochastic Descent
  - Mini-batches

- Improving generalization
  - Drop-out

- Activation functions

- Initialization and batch normalization

- Computing the gradient $\nabla g(w)$
  - Gradient checking
  - Backprop
Q: what happens when W=0 init is used?
- First idea: **Small random numbers**
  (gaussian with zero mean and 1e-2 standard deviation)

$$W = 0.01* \text{np.random.randn(D,H)}$$
- First idea: **Small random numbers**
  (gaussian with zero mean and 1e-2 standard deviation)

\[
W = 0.01 \times \text{np.random.randn(D,H)}
\]

Works ~okay for small networks, but can lead to non-homogeneous distributions of activations across the layers of a network.
Let's look at some activation statistics.

E.g. 10-layer net with 500 neurons on each layer, using tanh non-linearities, and initializing as described in last slide.
input layer had mean 0.000927 and std 0.998388
hidden layer 1 had mean -0.006117 and std 0.213881
hidden layer 2 had mean -0.000136 and std 0.047551
hidden layer 3 had mean -0.000002 and std 0.018638
hidden layer 4 had mean 0.000661 and std 0.002378
hidden layer 5 had mean 0.000662 and std 0.000533
hidden layer 6 had mean -0.000600 and std 0.000110
hidden layer 7 had mean 0.000600 and std 0.000026
hidden layer 8 had mean -0.000600 and std 0.000060
hidden layer 9 had mean 0.000600 and std 0.000081
hidden layer 10 had mean -0.000600 and std 0.000080
All activations become zero!

Q: What do the gradients look like?
Almost all neurons completely saturated, either -1 and 1. Gradients will be all zero.

*1.0 instead of *0.01
"Xavier initialization"
[Glorot et al., 2010]

Reasonable initialization.
(Mathematical derivation assumes linear activations)
but when using the ReLU nonlinearity it breaks.
He et al., 2015
(note additional /2)
He et al., 2015 (note additional \(1/2\))
Proper initialization is an active area of research…

*Understanding the difficulty of training deep feedforward neural networks* by Glorot and Bengio, 2010

*Exact solutions to the nonlinear dynamics of learning in deep linear neural networks* by Saxe et al., 2013

*Random walk initialization for training very deep feedforward networks* by Sussillo and Abbott, 2014

*Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification* by He et al., 2015

*Data-dependent Initializations of Convolutional Neural Networks* by Krähenbühl et al., 2015

*All you need is a good init*, Mishkin and Matas, 2015

…
Batch Normalization

“you want unit gaussian activations? just make them so.”

consider a batch of activations at some layer. To make each dimension unit gaussian, apply:

\[ \hat{x}^{(k)} = \frac{x^{(k)} - \mathbb{E}[x^{(k)}]}{\sqrt{\text{Var}[x^{(k)}]}} \]

this is a vanilla differentiable function...
Batch Normalization

“you want unit gaussian activations? just make them so.”

1. compute the empirical mean and variance independently for each dimension.

\[ \hat{x}(k) = \frac{x^{(k)} - E[x^{(k)}]}{\sqrt{\text{Var}[x^{(k)}]}} \]

[ioffe and Szegedy, 2015]
Batch Normalization

 Usually inserted after Fully Connected / (or Convolutional, as we’ll see soon) layers, and before nonlinearity.

Problem: do we necessarily want a unit gaussian input to a tanh layer?

\[
\hat{x}(k) = \frac{x(k) - E[x(k)]}{\sqrt{Var[x(k)]}}
\]

[ioffe and szegedy, 2015]
Batch Normalization

Normalize:

$$\tilde{x}^{(k)} = \frac{x^{(k)} - E[x^{(k)}]}{\sqrt{\text{Var}[x^{(k)}]}}$$

And then allow the network to squash the range if it wants to:

$$y^{(k)} = \gamma^{(k)} \tilde{x}^{(k)} + \beta^{(k)}$$

Note, the network can learn:

$$\gamma^{(k)} = \sqrt{\text{Var}[x^{(k)}]}$$
$$\beta^{(k)} = E[x^{(k)}]$$

to recover the identity mapping.

[Ioffe and Szegedy, 2015]
Batch Normalization

**Input:** Values of $x$ over a mini-batch: $B = \{x_1...m\}$; Parameters to be learned: $\gamma, \beta$

**Output:** $\{y_i = \text{BN}_{\gamma,\beta}(x_i)\}$

\[
\begin{align*}
\mu_B & \leftarrow \frac{1}{m} \sum_{i=1}^{m} x_i & \text{// mini-batch mean} \\
\sigma_B^2 & \leftarrow \frac{1}{m} \sum_{i=1}^{m} (x_i - \mu_B)^2 & \text{// mini-batch variance} \\
\hat{x}_i & \leftarrow \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} & \text{// normalize} \\
y_i & \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma,\beta}(x_i) & \text{// scale and shift}
\end{align*}
\]

- Improves gradient flow through the network
- Allows higher learning rates
- Reduces the strong dependence on initialization
- Acts as a form of regularization in a funny way, and slightly reduces the need for dropout, maybe

[Ioffe and Szegedy, 2015]
Batch Normalization

Input: Values of $x$ over a mini-batch: $B = \{x_1...m\}$; Parameters to be learned: $\gamma, \beta$

Output: $\{y_i = \text{BN}_{\gamma,\beta}(x_i)\}$

$\mu_B \leftarrow \frac{1}{m} \sum_{i=1}^{m} x_i$ // mini-batch mean

$\sigma^2_B \leftarrow \frac{1}{m} \sum_{i=1}^{m} (x_i - \mu_B)^2$ // mini-batch variance

$\hat{x}_i \leftarrow \frac{x_i - \mu_B}{\sqrt{\sigma^2_B + \epsilon}}$ // normalize

$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma,\beta}(x_i)$ // scale and shift

Note: at test time BatchNorm layer functions differently:

The mean/std are not computed based on the batch. Instead, a single fixed empirical mean of activations during training is used.

(e.g. can be estimated during training with running averages)
Remaining Pieces

- Optimizing machine learning objectives:
  - Stochastic Descent
  - Mini-batches

- Improving generalization
  - Drop-out

- Activation functions

- Initialization and batch normalization

- Computing the gradient $\nabla g(w)$
  - Gradient checking
  - Backprop
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
Computational Graph

\[ f = Wx \]

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

input image
weights

loss
Neural Turing Machine

input tape

loss
Neural Turing Machine
\[ f(x, y, z) = (x + y)z \]
e.g. \( x = -2, y = 5, z = -4 \)
\[ f(x, y, z) = (x + y)z \]
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: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)
\[ f(x, y, z) = (x + y)z \]
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: \(
\frac{\partial f}{\partial x}, \quad \frac{\partial f}{\partial y}, \quad \frac{\partial f}{\partial z}
\)
\[ f(x, y, z) = (x + y)z \]

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: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)
\[ f(x, y, z) = (x + y)z \]

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: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)
\[ f(x, y, z) = (x + y)z \]
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: \( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \)
\[ f(x, y, z) = (x + y)z \]

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

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:

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial y}$$
$f(x, y, z) = (x + y)z$

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: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$
\[ f(x, y, z) = (x + y)z \]
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:
\[ \frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x} \]
activations

$$x$$

$$y$$

$$z$$
activations

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

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

"local gradient"
The diagram illustrates the flow of activations and gradients in a neural network. It shows the function $f$ with inputs $x$ and $y$, leading to intermediate value $z$, and finally to the loss $L$. The local gradient is indicated by $\frac{\partial z}{\partial x}$ and $\frac{\partial z}{\partial y}$, while the overall gradient with respect to $z$ is $\frac{\partial L}{\partial z}$. The term "local gradient" is highlighted to emphasize the concept of gradient propagation through the network.
 activations

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

"local gradient"

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

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

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

gradients
activations

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

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

"local gradient"
activations

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

"local gradient"

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

gradients
Another example:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

\[
\begin{align*}
  \frac{df}{dx} &= -\frac{1}{x^2} \\
  \frac{df}{dx} &= 1
\end{align*}
\]
Another example: \[ f(w, x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}} \]

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

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

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

\[ (1)(-0.53) = -0.53 \]
Another example:

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

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

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

\[ (e^{-1})(-0.53) = -0.20 \]

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

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

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

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

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

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

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

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

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

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

\[ (-1) \times (-0.20) = 0.20 \]

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

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

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

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

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

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

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

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

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

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

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

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

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

[local gradient] x [its gradient]

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

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

\[
\begin{align*}
  f(x) &= \frac{1}{x} \\
  \frac{df}{dx} &= -\frac{1}{x^2}
\end{align*}
\]
\[
f(w, x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}}
\]

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

\[
\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1 + e^{-x})^2} = \left(\frac{1 + e^{-x} - 1}{1 + e^{-x}}\right) \left(\frac{1}{1 + e^{-x}}\right) = (1 - \sigma(x)) \sigma(x)
\]

sigmoid function

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

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

\[ \frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1 + e^{-x})^2} = \left( \frac{1 + e^{-x} - 1}{1 + e^{-x}} \right) \left( \frac{1}{1 + e^{-x}} \right) = (1 - \sigma(x)) \sigma(x) \]
Patterns in backward flow

**add** gate: gradient distributor

**max** gate: gradient router

**mul** gate: gradient…?

![Diagram showing patterns in backward flow](image)
Gradients add at branches
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
```
Implementation: forward/backward API

\[(x, y, z)\text{ are scalars)}\]

```
class MultiplyGate(object):
    def forward(x, y):
        z = x*y
        return z
    def backward(dz):
        # dx = ... #todo
        # dy = ... #todo
        return [dx, dy]
```

\[
\frac{\partial L}{\partial x} \quad \frac{\partial L}{\partial z}
\]
Implementation: forward/backward API

```
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]
```

(x, y, z are scalars)
Deep Learning Frameworks

TensorFlow (in your Project 6!)
Theano
Torch
CAFFE
Computation Graph Toolkit (CGT)
That’s it for ML

Up next: Advanced Applications!