CS 4803 / 7643: Deep Learning

Topics:
- Optimization
- Computing Gradients

Dhruv Batra
Georgia Tech

Administrativia

- HW1 Reminder
  - Due: 09/09, 11:59pm

Recap from last time

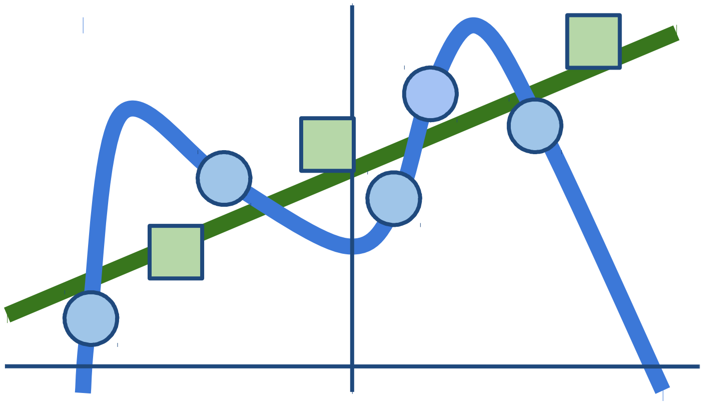
Regularization

λ = regularization strength (hyperparameter)

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)} + \underbrace{\lambda R(W)}$$

Data loss: Model predictions should match training data

Regularization: Prevent the model from doing *too* well on training data



Occam's Razor:
"Among competing hypotheses, the simplest is the best"
William of Ockham, 1285 - 1347

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

Regularization

λ = regularization strength (hyperparameter)

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)} + \underbrace{\lambda R(W)}$$

Data loss: Model predictions should match training data

Regularization: Prevent the model from doing *too* well on training data

Simple examples

L2 regularization: $R(W) = \sum_k \sum_l W_{k,l}^2$

L1 regularization: $R(W) = \sum_k \sum_l |W_{k,l}|$

Elastic net (L1 + L2): $R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$

More complex:

Dropout

Batch normalization

Stochastic depth, fractional pooling, etc

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

So far: Linear Classifiers

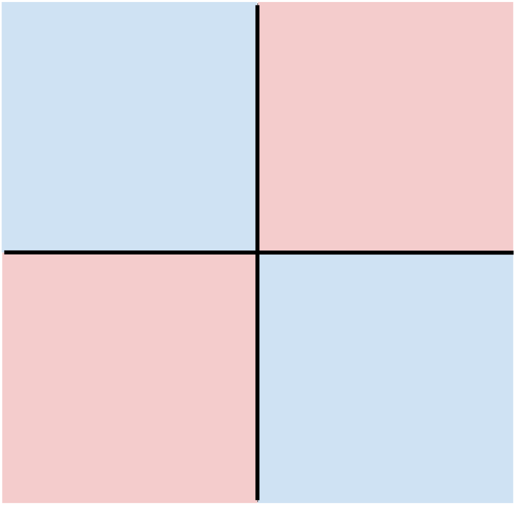
Class scores

$$f(x) = Wx$$

Hard cases for a linear classifier

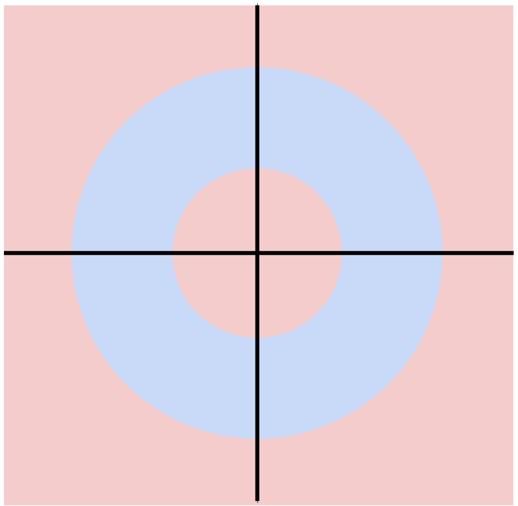
Class 1:
First and third quadrants

Class 2:
Second and fourth quadrants



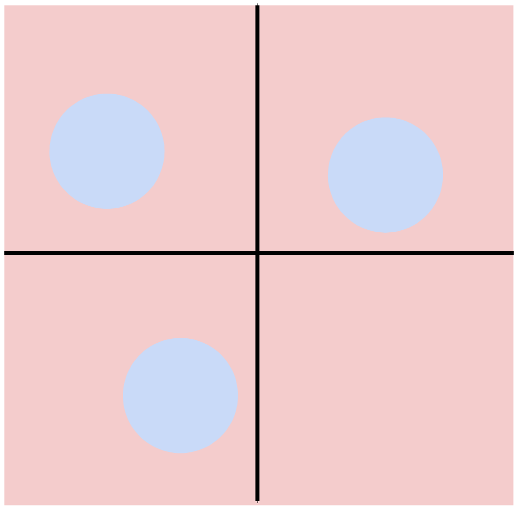
Class 1:
1 <= L2 norm <= 2

Class 2:
Everything else



Class 1:
Three modes

Class 2:
Everything else

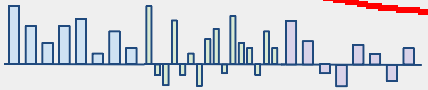


Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Image features vs Neural Nets

$\phi(x)$

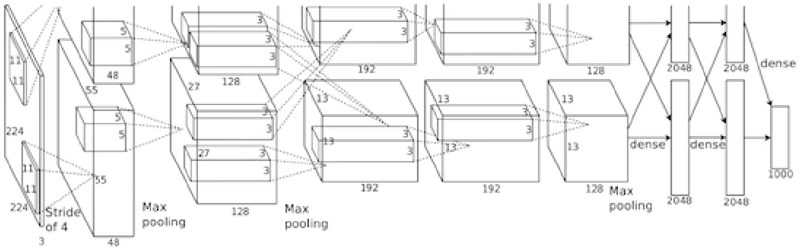
Feature Extraction



$f = w^T \phi(x)$

10 numbers giving scores for classes

training



Krizhevsky, Sutskever, and Hinton, "Imagenet classification with deep convolutional neural networks", NIPS 2012. Figure copyright Krizhevsky, Sutskever, and Hinton, 2012. Reproduced with permission.

10 numbers giving scores for classes

training

Neural networks: without the brain stuff

(**Before**) Linear score function: $f = \underbrace{Wx}$

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

Neural networks: without the brain stuff

(**Before**) Linear score function: $f = \overline{\underline{W}}x$

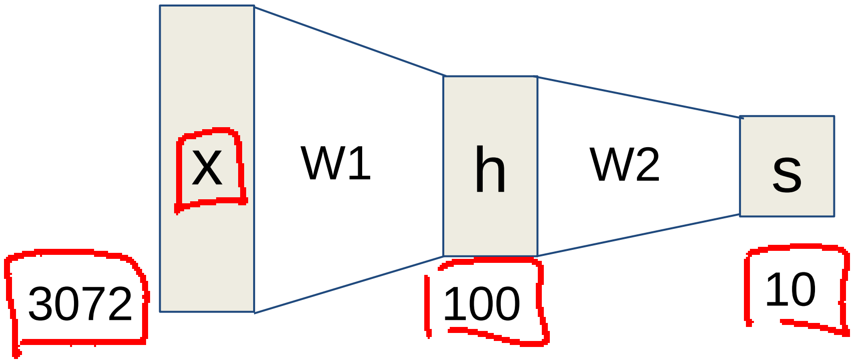
(**Now**) 2-layer Neural Network $f = \underline{W_2}\boxed{\max}(0, \underline{W_1}x)$

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

Neural networks: without the brain stuff

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

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

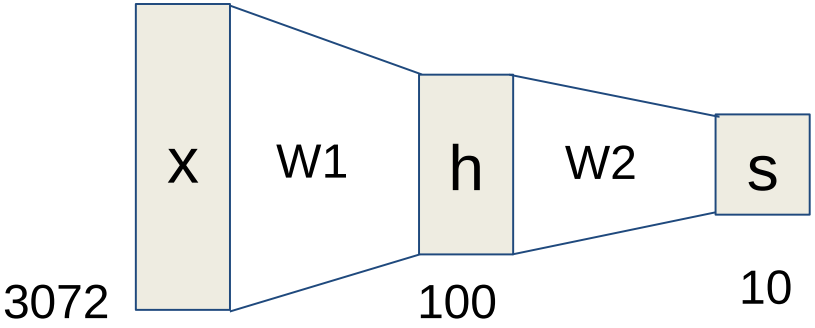


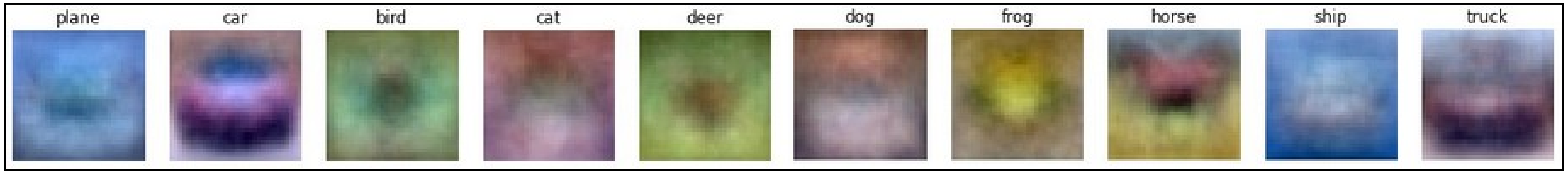
Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Neural networks: without the brain stuff

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

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





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

Neural networks: without the brain stuff

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

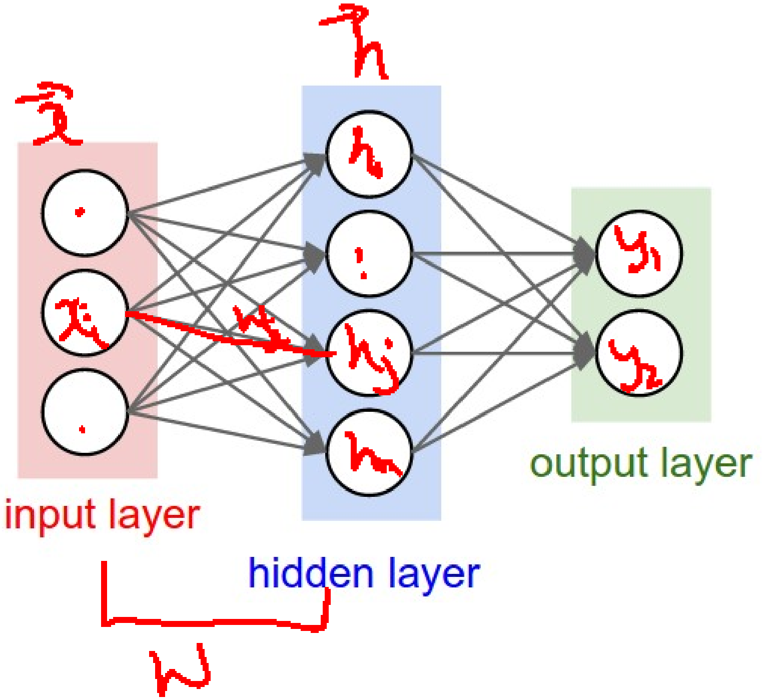
(**Now**) 2-layer Neural Network $f = W_2 \max(0, W_1 x)$
or 3-layer Neural Network

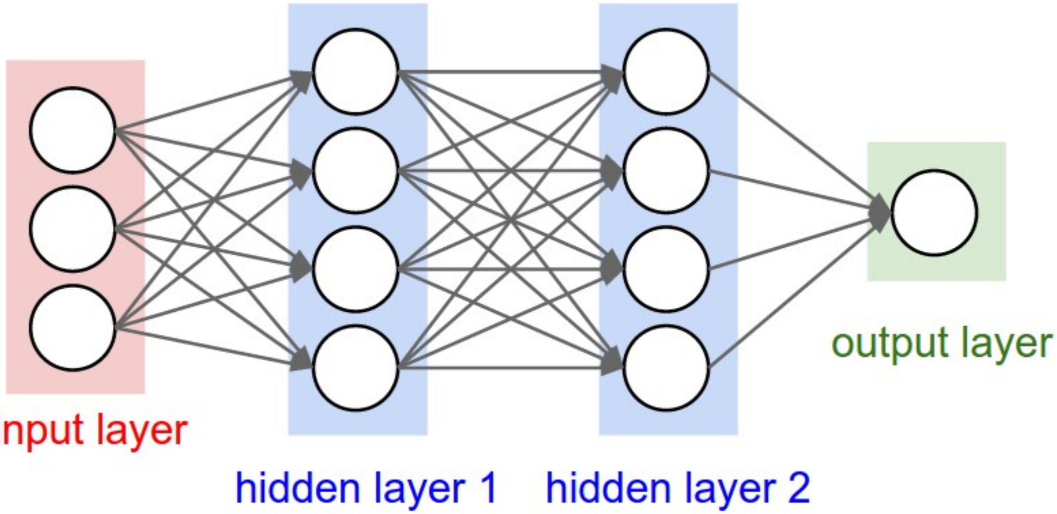
$$f = \underline{W_3 \max}(0, W_2 \underline{\max(0}, W_1 x))$$

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

Multilayer Networks

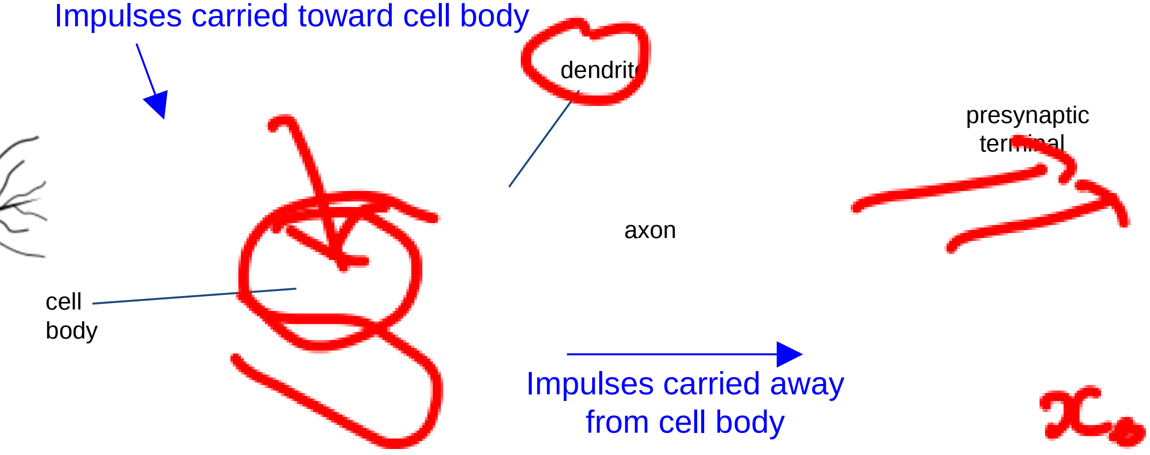
- Cascaded "neurons"
- The output from one layer is the input to the next
- Each layer has its own sets of weights





(C) Dhruv Batra

Image Credit: Andrej Karpathy, CS231n



Impulses carried toward cell body

dendrite

presynaptic terminal

axon

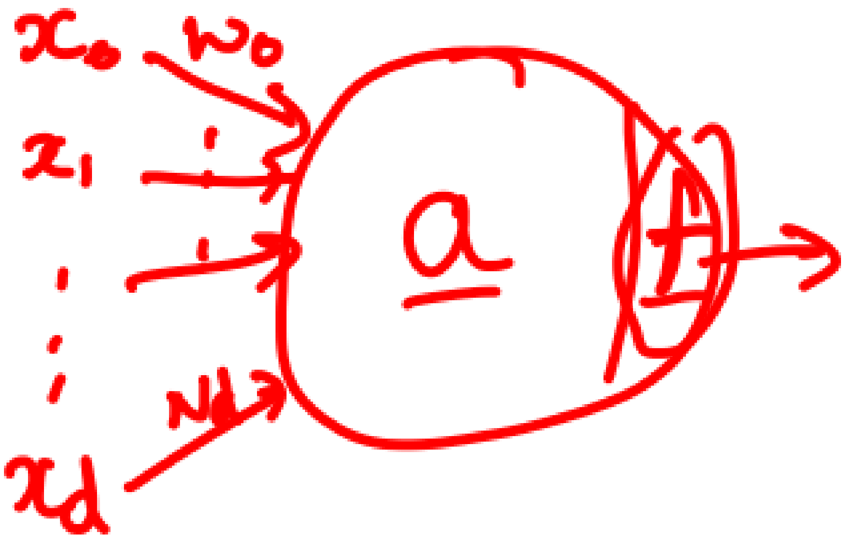
cell body

Impulses carried away from cell body

This image by Felipe Perucho is licensed under CC-BY 3.0

$$a = \sum_j w_j x_j$$

$$= \underline{\vec{w}^T \vec{x}}$$



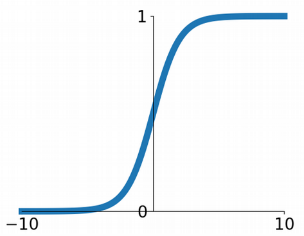
$$y = f(a)$$

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

Activation functions

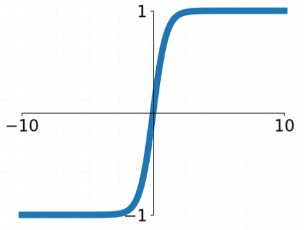
Sigmoid

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



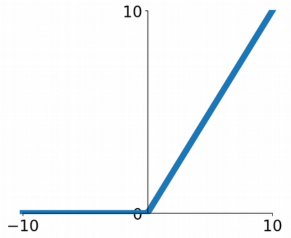
tanh

$\tanh(x)$



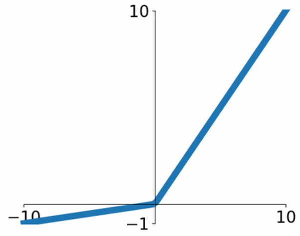
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

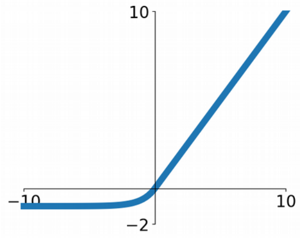


Maxout

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

ELU

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



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

Plan for Today

- Optimization
- Computing Gradients

Optimization

Supervised Learning

- Input: x (images, text, emails…)
- Output: y (spam or non-spam…)

- (Unknown) Target Function
  - f: X ✉ Y (the “true” mapping / reality)

- Data
  - $(x_1,y_1), (x_2,y_2), \ldots, (x_N,y_N)$

- Model / Hypothesis Class
  - {h: X ✉ Y}
  - e.g. $y = h(x) = \text{sign}(w^T x)$

- Loss Function
  - How good is a model wrt my data D?

- Learning = ~~Search in hypothesis space~~
  - Find best h in model class.

→ linear $w^T x$

→ MLP / NN

$L(\vec{w}, D)$

→ Hinge

→ Softmax CE

$$\min_{\vec{w}} L(\vec{w}, D)$$

Demo Time

- https://playground.tensorflow.org

$$\min_{\vec{w}} \; L(\vec{w})$$

Strategy: **Follow the slope**

Gradient Descent

$L(\vec{w})$

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

What is slope?

- In 1-dimension, the derivative of a function:

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

What is slope?

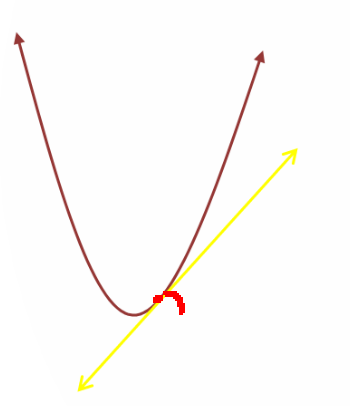
$L(\vec{w})$

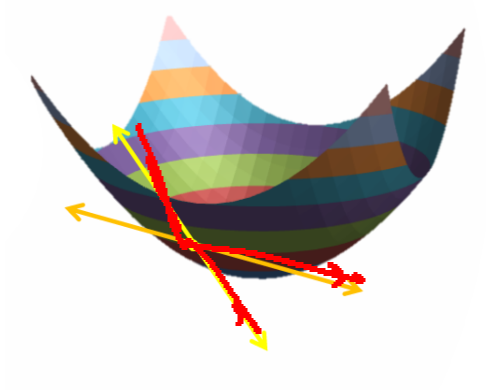
- In d-dimension, recall partial derivatives:

$f : \mathbb{R}^d \rightarrow \mathbb{R}$ $\quad f(\vec{x})$ $\quad \vec{x} \in \mathbb{R}^d$

∂

$$\boxed{\frac{\partial f}{\partial x_i}} = \lim_{h \to 0} \frac{f(x_1, \ldots, \overset{i^{th}}{x_i + h}, \ldots, x_d) - f(x_1, \ldots, x_d)}{h}$$





What is slope?

- The **gradient** is the vector of (partial derivatives) along each dimension

$f: \mathbb{R}^d \rightarrow \mathbb{R}$

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x_1} & \cdots & \frac{\partial f}{\partial x_d} \end{bmatrix}$$

[Caveat: row-vs-col vector convention]

- Properties
  - The direction of steepest descent is the **negative gradient**
  - The slope in any direction is the **dot product** of the direction with the gradient

$\nabla_{\vec{x}} f$

$$\frac{\partial f}{\partial \vec{x}}$$

$\nabla_{\vec{x}} f$

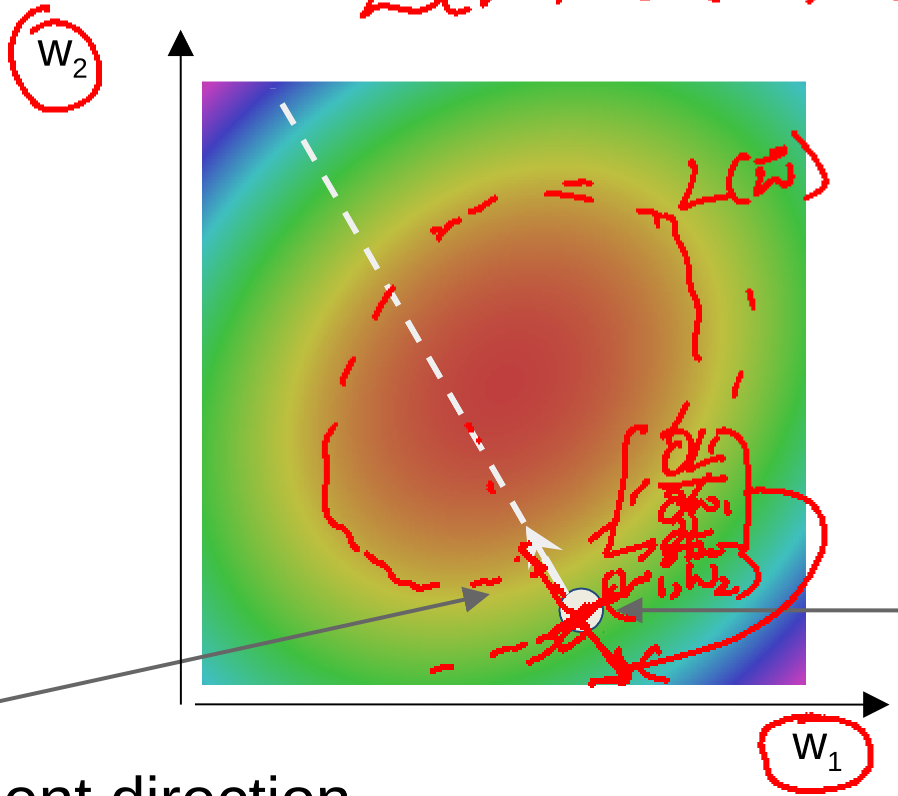
$$\boxed{\nabla_{\vec{w}}} L(\vec{w}, D)$$

http://demonstrations.wolfram.com/VisualizingTheGradientVector/

$L(\vec{w}) = L(w_1, w_2) = a \cdot w_1^2 + b w_2^2 + c\, w_1 w_2$

blue +ve

red -ve



original W

negative gradient direction

$\min_{\vec{w}} L(\vec{w})$

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

Gradient Descent

$$\min_{\vec{w}} L(\vec{w})$$

```
# Vanilla Gradient Descent

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

$\vec{w}^{(0)} =$ Initialize

for $t = 0, 1, 2, \ldots,$ exhausted

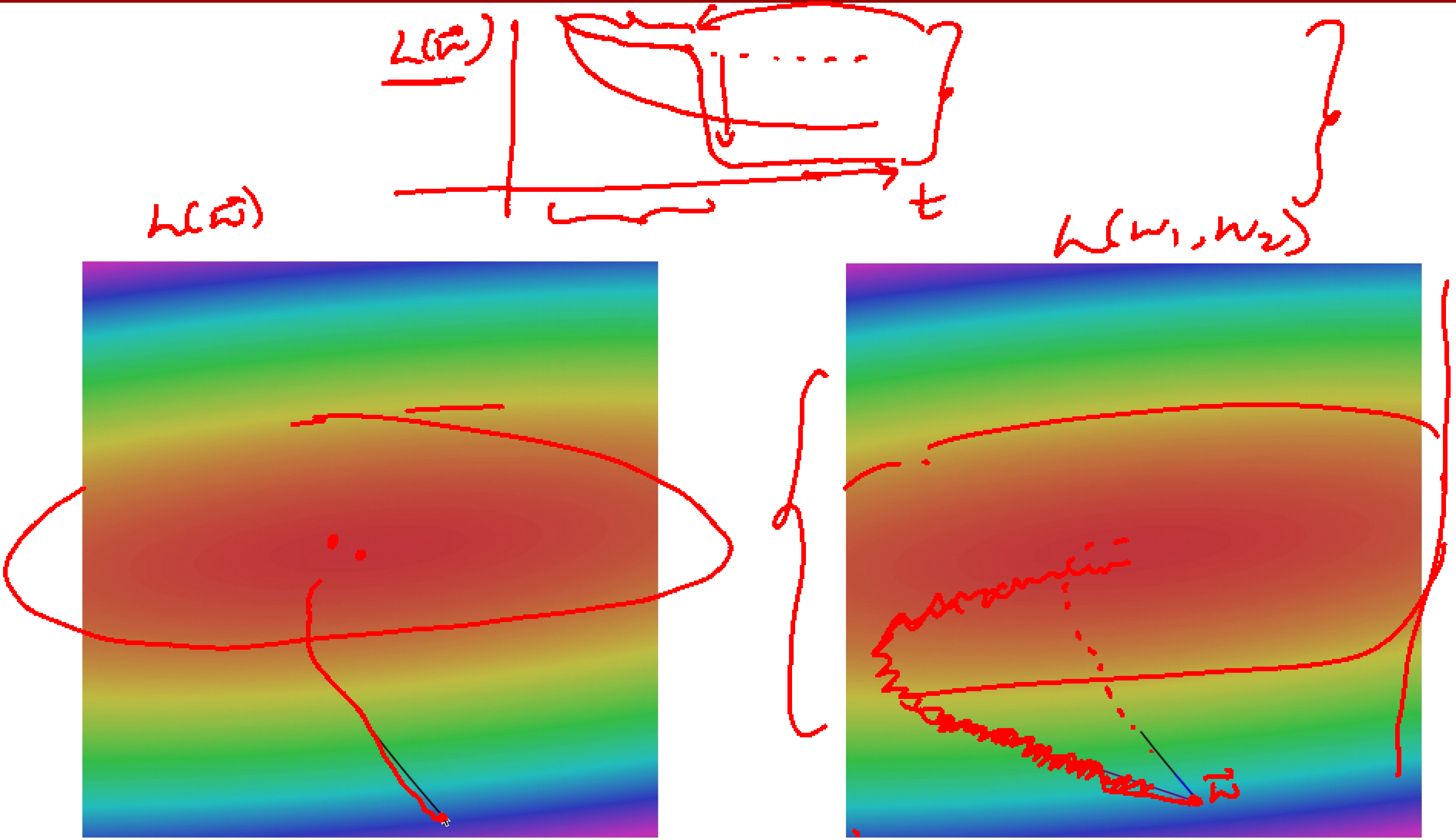
$$w^{(t+1)} = w^{(t)} - \boxed{\eta}\, \nabla_{\vec{w}} L(\vec{w}) \;|$$

0.001

step-size

learning-rate

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



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

Gradient Descent has a problem

$$\underline{L(W)} = \boxed{\frac{1}{N}} \sum_{i=1}^{N} \underline{L_i}(\underline{x_i}, \underline{y_i}, \underline{W}) + \lambda \underline{R(W)}$$

$$\underline{\nabla_W L(W)} = \boxed{\frac{1}{N} \sum_{i=1}^{N}} \nabla_W L_i(x_i, y_i, W) + \lambda \nabla_W R(W)$$

Full sum expensive
when N is large!

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

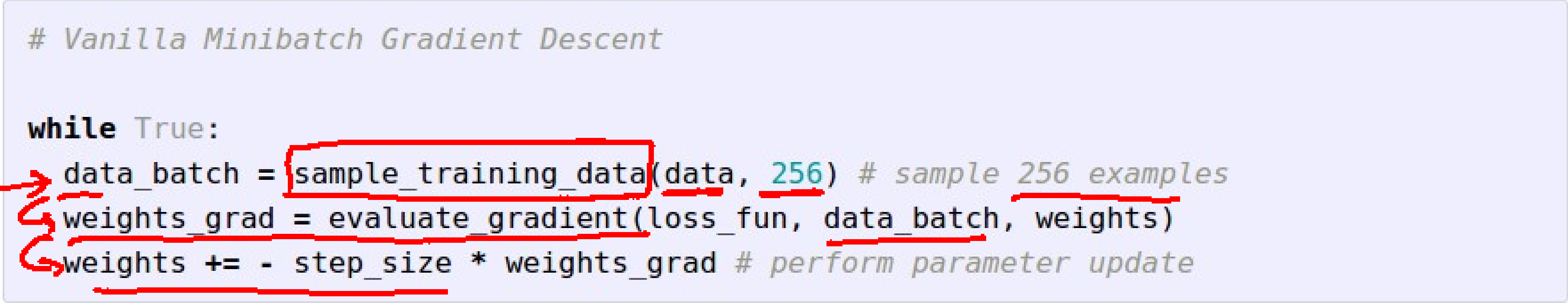
Stochastic Gradient Descent (SGD)

$$L(W) = \frac{1}{N}\sum_{i=1}^{N} L_i(x_i, y_i, W) + \lambda R(W)$$

$$\nabla_W L(W) = \boxed{\frac{1}{N}\sum_{i=1}^{N}} \nabla_W L_i(x_i, y_i, W) + \lambda \nabla_W R(W)$$

Full sum expensive when N is large!

Approximate sum using a **minibatch** of examples
32 / 64 / 128 common



```
# Vanilla Minibatch Gradient Descent

while True:
  data_batch = sample_training_data(data, 256) # sample 256 examples
  weights_grad = evaluate_gradient(loss_fun, data_batch, weights)
  weights += - step_size * weights_grad # perform parameter update
```

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

Stochastic Gradient Descent (SGD)

$$L(W) = \frac{1}{N}\sum_{i=1}^{N} L_i(x_i, y_i, W)$$

$$\boxed{\underline{\nabla_W L(W)}} = \underline{\left|\frac{1}{N}\sum_{i=1}^{N}\right|} \nabla_W L_i(x_i, y_i, W) = \text{mean-over-}N\left(\nabla_W L_i\right)_{i=1}^{N}$$

$$\approx \text{mean-over-}B\left(\cdot\right)$$

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

Stochastic Gradient Descent (SGD)

"Actually care about"

$$\boxed{L(W)} = \boxed{\frac{1}{N}\sum_{i=1}^{N}} L_i(x_i, y_i, W) \overset{N}{\approx} E_{\boxed{X,Y \sim P^*(\,)}}\left[L(w,x,y)\right]$$

$$\nabla_W L(W) = \frac{1}{N}\sum_{i=1}^{N} \nabla_W L_i(x_i, y_i, W)$$

$$\nabla_w E[\;] \quad \boxed{E_{x,y}\left[\nabla_w L(w,x,y)\right]}$$

$$\boxed{\overset{\approx}{B}\text{sample}}$$

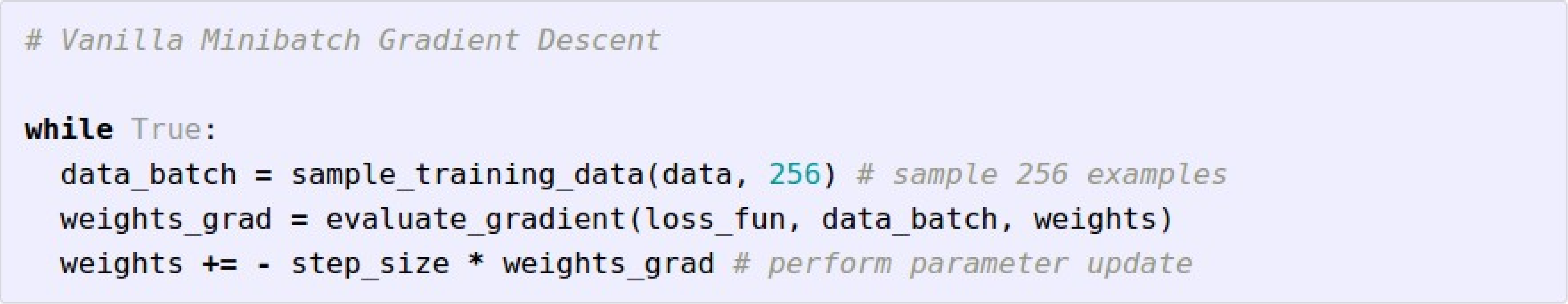
Stochastic Gradient Descent (SGD)

$$L(W) = \frac{1}{N}\sum_{i=1}^{N} L_i(x_i, y_i, W) \boxed{+ \lambda R(W)}$$

$$\nabla_W L(W) = \frac{1}{N}\sum_{i=1}^{N} \underline{\nabla_W L_i}(x_i, y_i, W) \boxed{+ \cancel{\lambda} \underline{\nabla_W} R(W)}$$

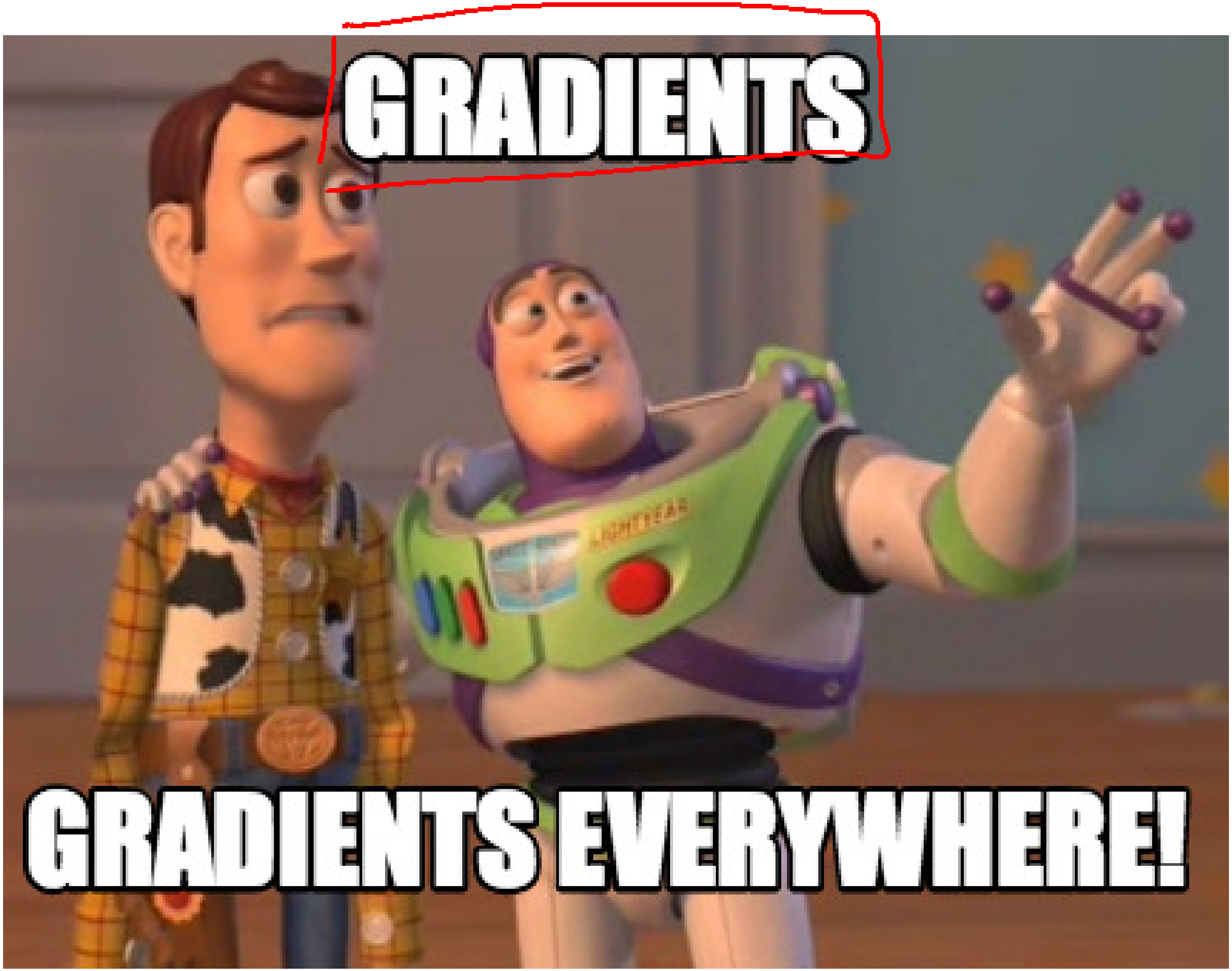
Full sum expensive when N is large!

Approximate sum using a **minibatch** of examples
32 / 64 / 128 common



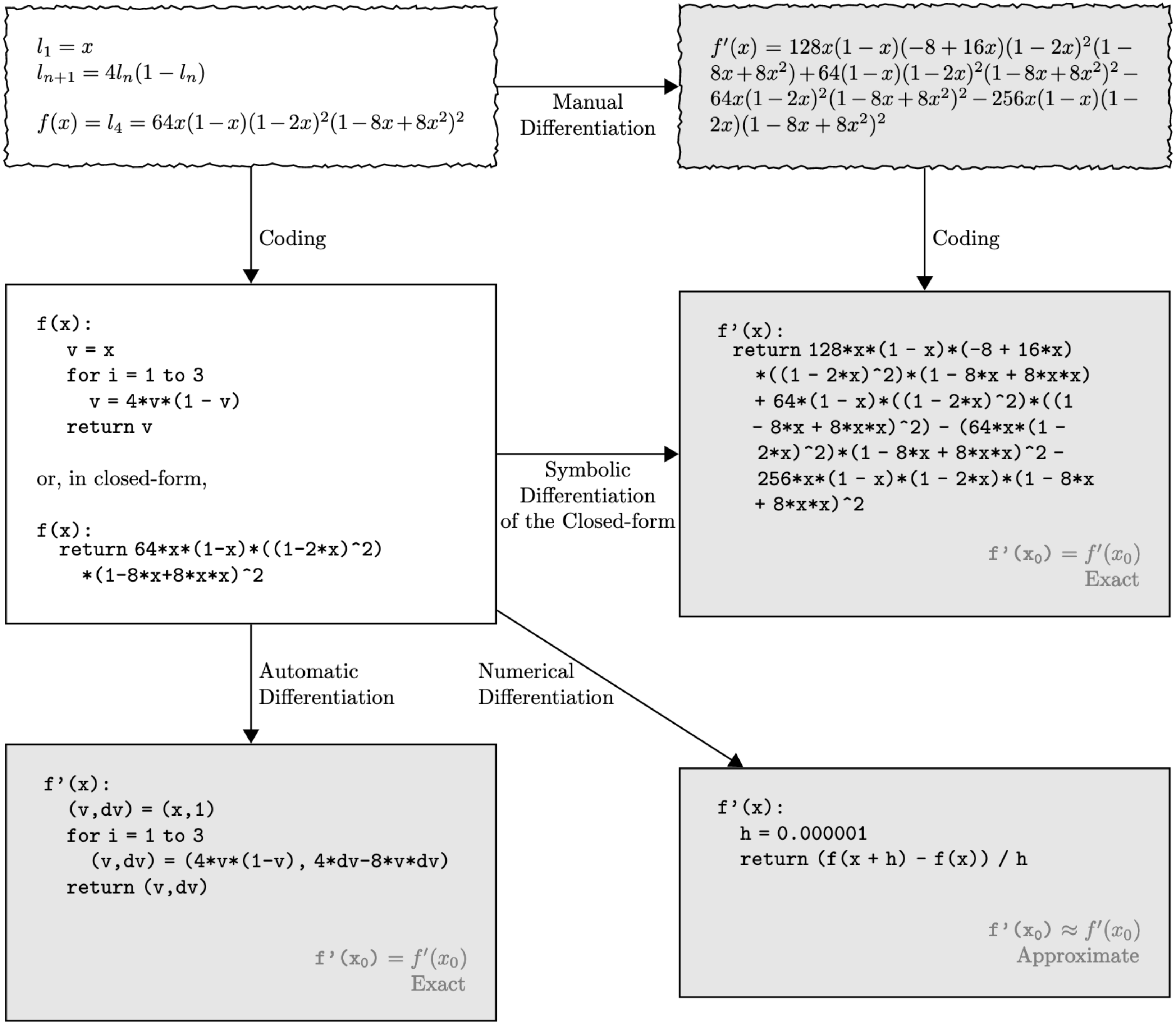
```
# Vanilla Minibatch Gradient Descent

while True:
  data_batch = sample_training_data(data, 256) # sample 256 examples
  weights_grad = evaluate_gradient(loss_fun, data_batch, weights)
  weights += - step_size * weights_grad # perform parameter update
```

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


GRADIENTS
GRADIENTS EVERYWHERE!

How do we compute gradients?

- Analytic or “Manual” Differentiation “pen & paper”
- Symbolic Differentiation
- Numerical Differentiation
- Automatic Differentiation
  - Forward mode AD
  - Reverse mode AD
    - aka “backprop”



$l_1 = x$

$l_{n+1} = 4l_n(1 - l_n)$

$f(x) = l_4 = 64x(1-x)(1-2x)^2(1-8x+8x^2)^2$

Manual Differentiation

$f'(x) = 128x(1-x)(-8+16x)(1-2x)^2(1-8x+8x^2)+64(1-x)(1-2x)^2(1-8x+8x^2)^2 - 64x(1-2x)^2(1-8x+8x^2)^2 - 256x(1-x)(1-2x)(1-8x+8x^2)^2$

Coding

Coding

```
f(x):
  v = x
  for i = 1 to 3
    v = 4*v*(1 - v)
  return v
```

or, in closed-form,

```
f(x):
  return 64*x*(1-x)*((1-2*x)^2)
    *(1-8*x+8*x*x)^2
```

Symbolic Differentiation of the Closed-form

```
f'(x):
  return 128*x*(1 - x)*(-8 + 16*x)
    *((1 - 2*x)^2)*(1 - 8*x + 8*x*x)
    + 64*(1 - x)*((1 - 2*x)^2)*((1
    - 8*x + 8*x*x)^2) - (64*x*(1 -
    2*x)^2)*(1 - 8*x + 8*x*x)^2 -
    256*x*(1 - x)*(1 - 2*x)*(1 - 8*x
    + 8*x*x)^2
```

f'(x_0) $= f'(x_0)$
Exact

Automatic Differentiation

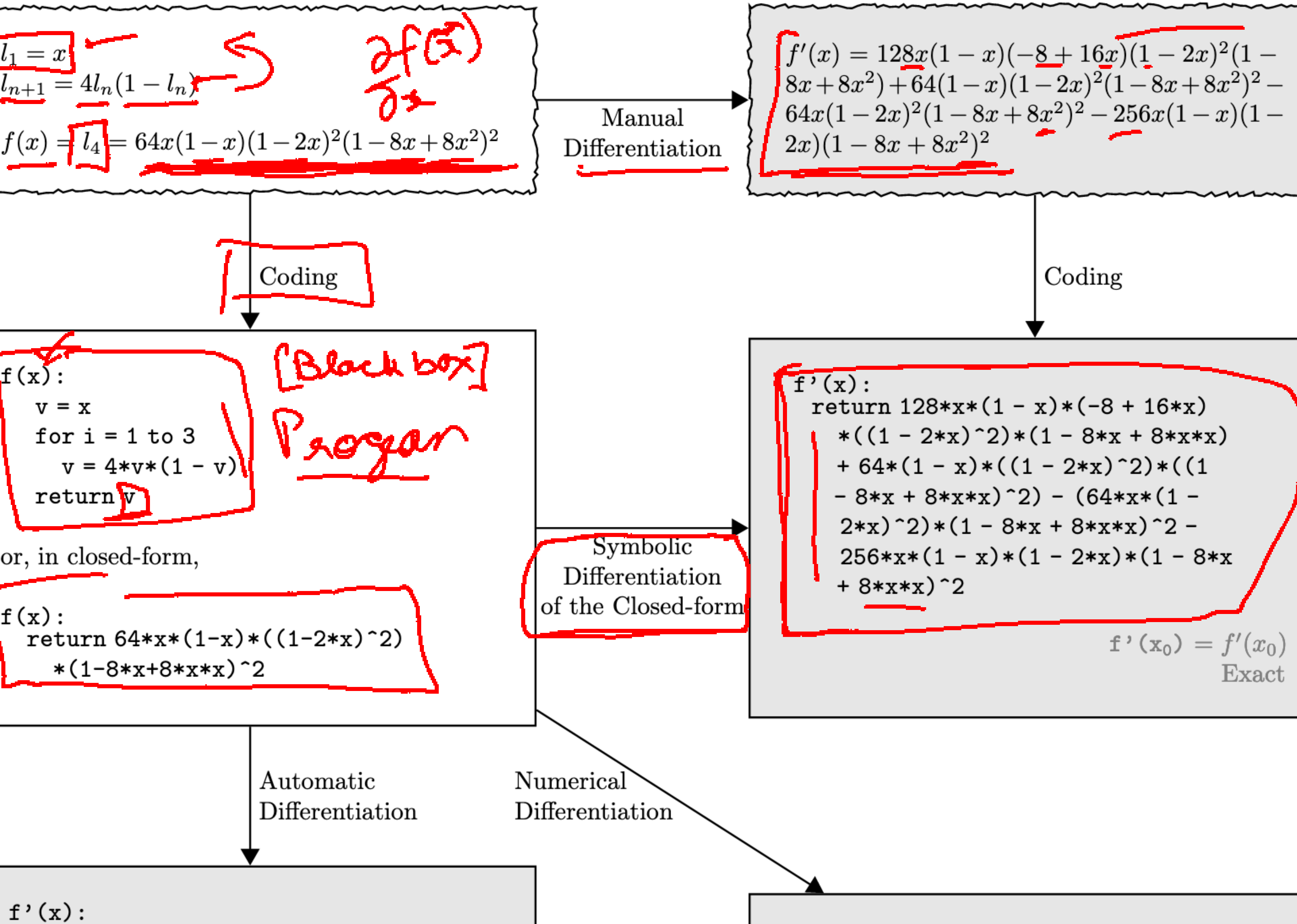
Numerical Differentiation

```
f'(x):
  (v,dv) = (x,1)
  for i = 1 to 3
    (v,dv) = (4*v*(1-v), 4*dv-8*v*dv)
  return (v,dv)
```

f'(x_0) $= f'(x_0)$
Exact

```
f'(x):
  h = 0.000001
  return (f(x + h) - f(x)) / h
```

f'(x_0) $\approx f'(x_0)$
Approximate



$l_1 = x$

$l_{n+1} = 4l_n(1 - l_n)$

$f(x) = l_4 = 64x(1-x)(1-2x)^2(1-8x+8x^2)^2$

$\frac{\partial f(\bar{x})}{\partial x}$

Manual Differentiation

$f'(x) = 128x(1-x)(-8+16x)(1-2x)^2(1-8x+8x^2)+64(1-x)(1-2x)^2(1-8x+8x^2)^2-64x(1-2x)^2(1-8x+8x^2)^2-256x(1-x)(1-2x)(1-8x+8x^2)^2$

Coding

Coding

```
f(x):
  v = x
  for i = 1 to 3
    v = 4*v*(1 - v)
  return v
```

[Black box]

Program

or, in closed-form,

```
f(x):
 return 64*x*(1-x)*((1-2*x)^2)
   *(1-8*x+8*x*x)^2
```

Symbolic Differentiation of the Closed-form

```
f'(x):
 return 128*x*(1 - x)*(-8 + 16*x)
    *((1 - 2*x)^2)*(1 - 8*x + 8*x*x)
    + 64*(1 - x)*((1 - 2*x)^2)*((1
    - 8*x + 8*x*x)^2) - (64*x*(1 -
    2*x)^2)*(1 - 8*x + 8*x*x)^2 -
    256*x*(1 - x)*(1 - 2*x)*(1 - 8*x
    + 8*x*x)^2
```

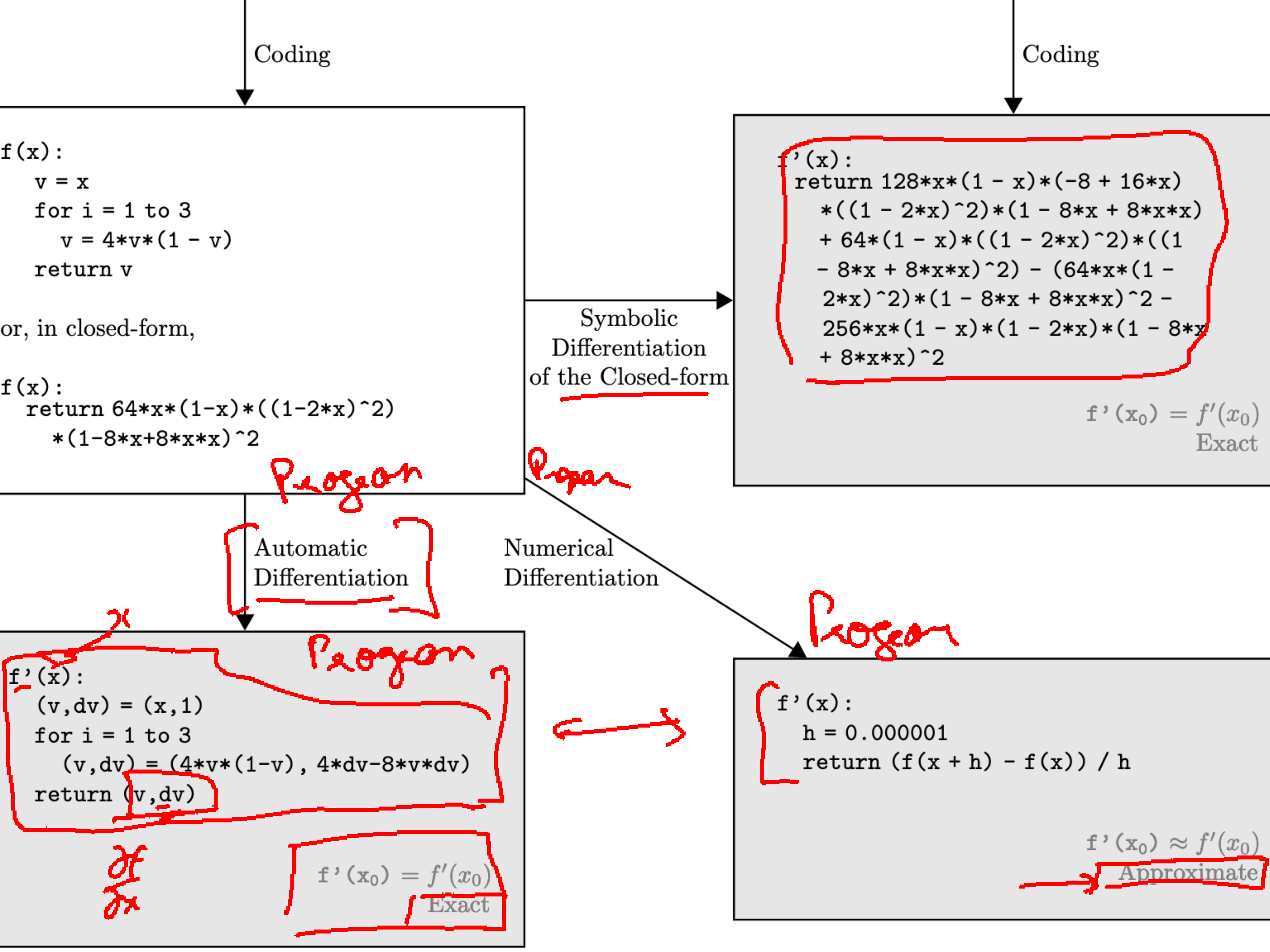
f'(x_0) = $f'(x_0)$

Exact

Automatic Differentiation

Numerical Differentiation

```
f'(x):
```



```
f(x):
   v = x
   for i = 1 to 3
     v = 4*v*(1 - v)
   return v
```

or, in closed-form,

```
f(x):
  return 64*x*(1-x)*((1-2*x)^2)
    *(1-8*x+8*x*x)^2
```

```
f'(x):
return 128*x*(1 - x)*(-8 + 16*x)
   *((1 - 2*x)^2)*(1 - 8*x + 8*x*x)
   + 64*(1 - x)*((1 - 2*x)^2)*((1
   - 8*x + 8*x*x)^2) - (64*x*(1 -
    2*x)^2)*(1 - 8*x + 8*x*x)^2 -
    256*x*(1 - x)*(1 - 2*x)*(1 - 8*x
   + 8*x*x)^2
```

$\texttt{f'(x}_0\texttt{)} = f'(x_0)$
Exact

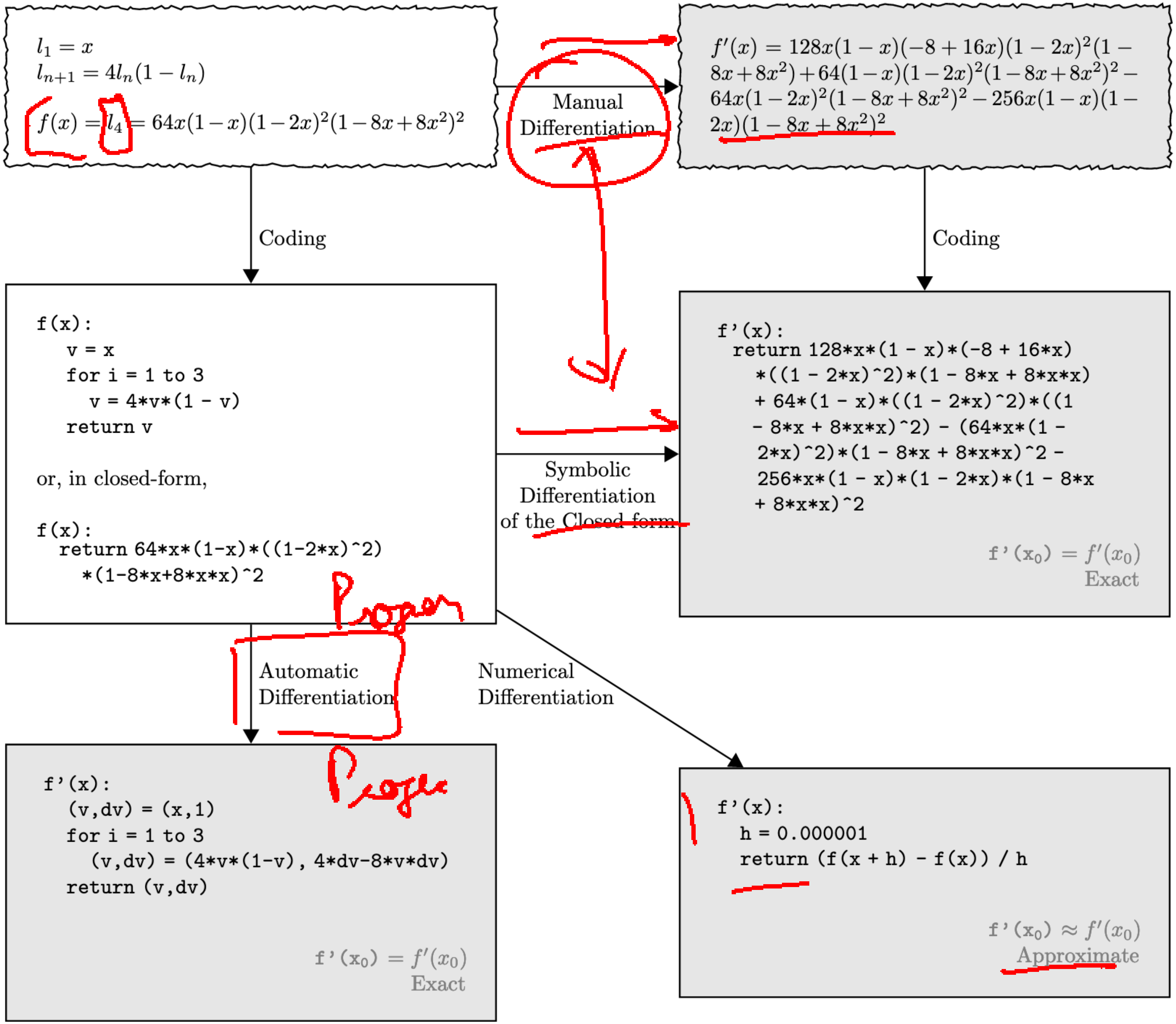
```
f'(x):
  (v,dv) = (x,1)
  for i = 1 to 3
    (v,dv) = (4*v*(1-v), 4*dv-8*v*dv)
  return (v,dv)
```

$\frac{\partial f}{\partial x}$

$\texttt{f'(x}_0\texttt{)} = f'(x_0)$
Exact

```
f'(x):
  h = 0.000001
  return (f(x + h) - f(x)) / h
```

$\texttt{f'(x}_0\texttt{)} \approx f'(x_0)$
Approximate



$l_1 = x$

$l_{n+1} = 4l_n(1 - l_n)$

$f(x) = l_4 = 64x(1-x)(1-2x)^2(1-8x+8x^2)^2$

Manual Differentiation

$f'(x) = 128x(1-x)(-8+16x)(1-2x)^2(1-8x+8x^2)+64(1-x)(1-2x)^2(1-8x+8x^2)^2-64x(1-2x)^2(1-8x+8x^2)^2-256x(1-x)(1-2x)(1-8x+8x^2)^2$

Coding

```
f(x):
   v = x
   for i = 1 to 3
     v = 4*v*(1 - v)
   return v
```

or, in closed-form,

```
f(x):
  return 64*x*(1-x)*((1-2*x)^2)
    *(1-8*x+8*x*x)^2
```

Coding

```
f'(x):
  return 128*x*(1 - x)*(-8 + 16*x)
    *((1 - 2*x)^2)*(1 - 8*x + 8*x*x)
    + 64*(1 - x)*((1 - 2*x)^2)*((1
    - 8*x + 8*x*x)^2) - (64*x*(1 -
    2*x)^2)*(1 - 8*x + 8*x*x)^2 -
    256*x*(1 - x)*(1 - 2*x)*(1 - 8*x
    + 8*x*x)^2
```

`f'(x0)` $= f'(x_0)$
Exact

Symbolic Differentiation of the Closed-form

Proper

Automatic Differentiation

Numerical Differentiation

Proper

```
f'(x):
  (v,dv) = (x,1)
  for i = 1 to 3
    (v,dv) = (4*v*(1-v), 4*dv-8*v*dv)
  return (v,dv)
```

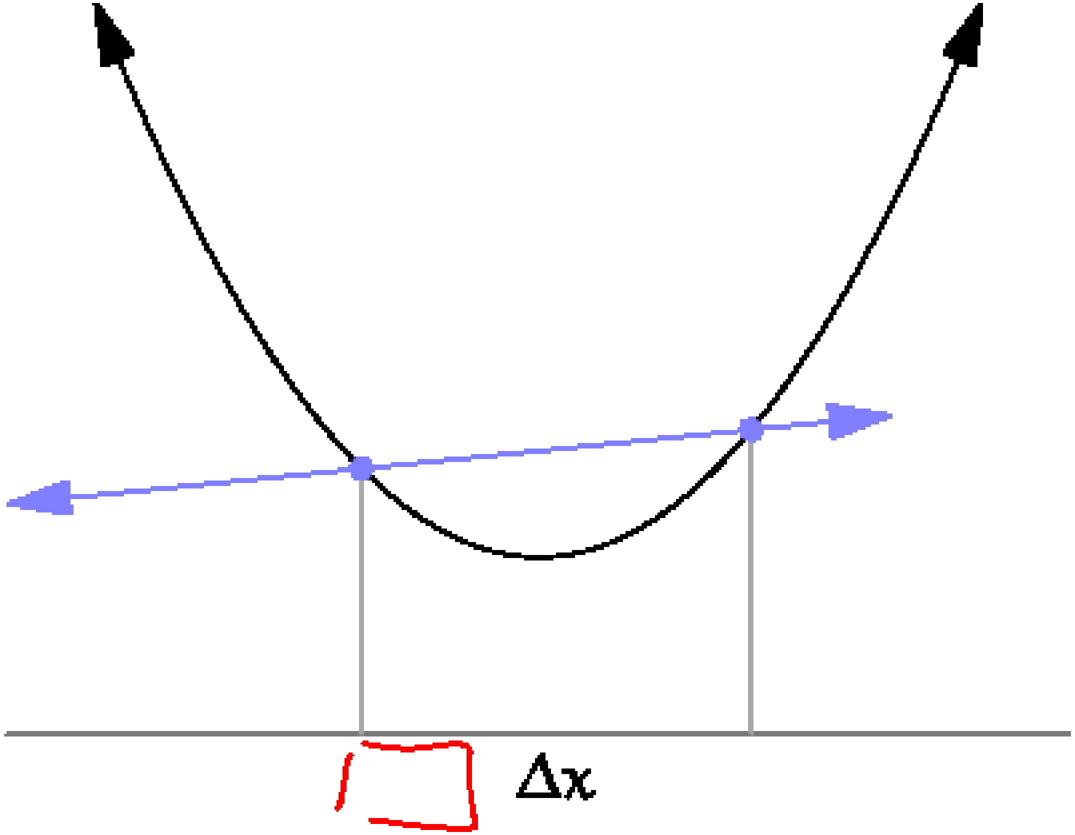
`f'(x0)` $= f'(x_0)$
Exact

```
f'(x):
  h = 0.000001
  return (f(x + h) - f(x)) / h
```

`f'(x0)` $\approx f'(x_0)$
Approximate

How do we compute gradients?

- Analytic or “Manual” Differentiation
- Symbolic Differentiation
- Numerical Differentiation
- Automatic Differentiation
  - Forward mode AD
  - Reverse mode AD
    - aka “backprop”


Δx

current W:

$\vec{W}$

[0.34,
-1.11,
0.78,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,…]

loss $\boxed{1.25347}$

$L(\vec{W})$

gradient dW:

[?,
?,
?,
?,
?,
?,
?,
?,
?,…]

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

current W:	**W + h** (first dim)**:**	**gradient dW:**
[0.34,	[0.34 + $\overset{h}{\boxed{\mathbf{0.0001}}}$	[?,
-1.11,	-1.11,	?,
0.78,	0.78,	?,
0.12,	0.12,	?,
0.55,	0.55,	?,
2.81,	2.81,	?,
-3.1,	-3.1,	?,
-1.5,	-1.5,	?,
0.33,…]	0.33,…]	?,…]
loss 1.25347	**loss 1.25322**	

$L(w_1+h, w_2, \ldots, w_d)$

$$\frac{\partial L}{\partial w_1} = \frac{L(w_1+h, \ldots, w_d) - L(w_1, \ldots, w_d)}{h}$$

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

current W:

[0.34,
-1.11,
0.78,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,…]
loss 1.25347

W + h (first dim)**:**

[0.34 + **0.0001**,
-1.11,
0.78,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,…]
loss 1.25322

gradient dW:

[**-2.5**,
?,
?,

(1.25322 - 1.25347)/0.0001
= -2.5

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

?,
?,…]

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

current W:

[0.34,
-1.11,
0.78,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,…]
loss 1.25347

W + h (second dim)**:**

[0.34,
-1.11 + **0.0001**,
0.78,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,…]
loss 1.25353

gradient dW:

[-2.5,
?,
?,
?,
?,
?,
?,
?,
?,…]

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

current W:

[0.34,
-1.11,
0.78,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,…]
loss 1.25347

W + h (second dim)**:**

[0.34,
-1.11 + **0.0001**,
0.78,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,…]
loss 1.25353

gradient dW:

[-2.5,
0.6,
?,
~~?,~~

(1.25353 - 1.25347)/0.0001
= 0.6

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

?,…]

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

current W:	**W + h** (third dim)**:**	**gradient dW:**
[0.34,	[0.34,	[-2.5,
-1.11,	-1.11,	0.6,
0.78,	0.78 + **0.0001**,	?,
0.12,	0.12,	?,
0.55,	0.55,	?,
2.81,	2.81,	?,
-3.1,	-3.1,	?,
-1.5,	-1.5,	?,
0.33,…]	0.33,…]	?,…]
loss 1.25347	**loss 1.25347**	

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

current W:

[0.34,
-1.11,
0.78,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,…]

loss 1.25347

W + h (third dim):

[0.34,
-1.11,
0.78 + **0.0001**,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,…]

loss 1.25347

gradient dW:

[-2.5,
0.6,
0,
?,
?,…]

(1.25347 - 1.25347)/0.0001
= 0

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

$\boxed{L(\vec{w})}$

$\vec{w} \in \mathbb{R}^d$

$d = 1B$

Numerical vs Analytic Gradients

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

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

In practice: Derive analytic gradient, check your implementation with numerical gradient.
This is called a **gradient check.**

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

How do we compute gradients?

- Analytic or “Manual” Differentiation
- Symbolic Differentiation ✗
- Numerical Differentiation
- Automatic Differentiation
  - Forward mode AD
  - Reverse mode AD
    - aka “backprop”

$$\frac{\partial L}{\partial \vec{w}} = \underset{\cancel{d \times d}\ 1 \times d}{\left[\qquad \right]}$$

$$\frac{\partial L}{\partial \vec{w}\, \partial \vec{w}}$$