Structure of Deep Learning Frameworks: computational graph, autodiff, and optimizers

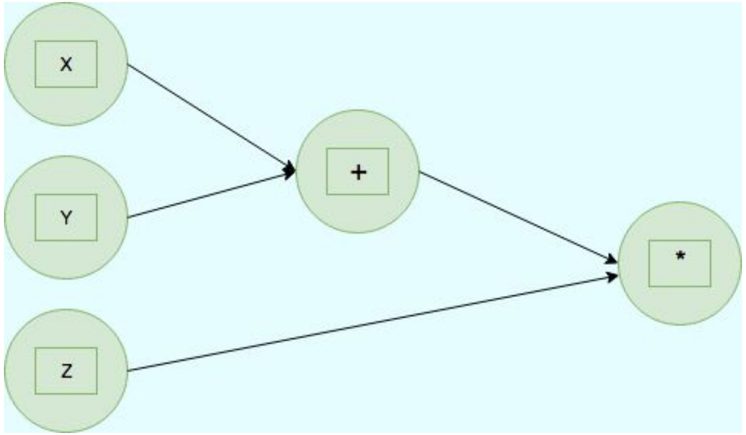
Joris Mollinga
SURF

Computational Graphs

- Represents math in graph format
- Nodes and edges

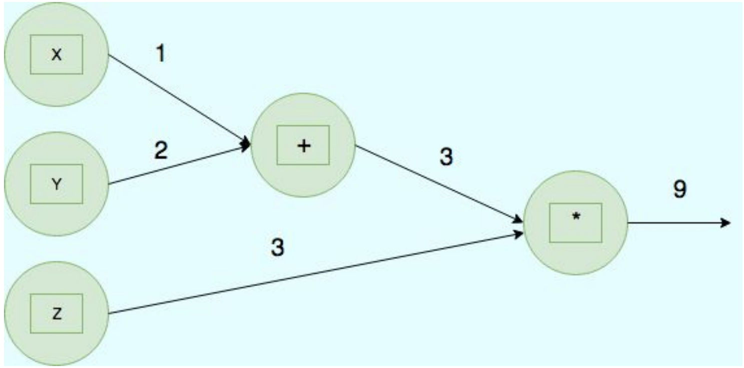
Computational Graphs

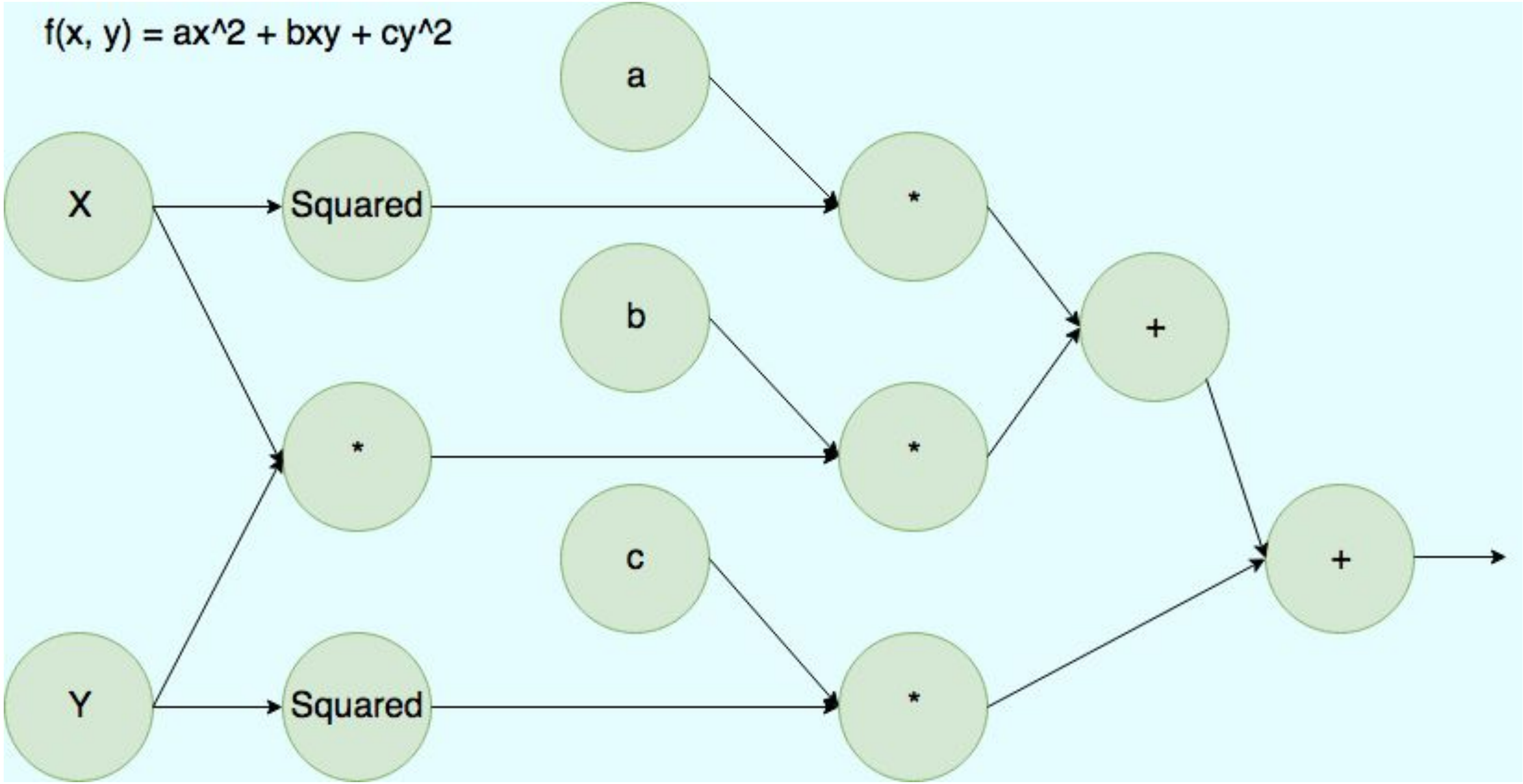
- Represents math in graph format
- Nodes and edges



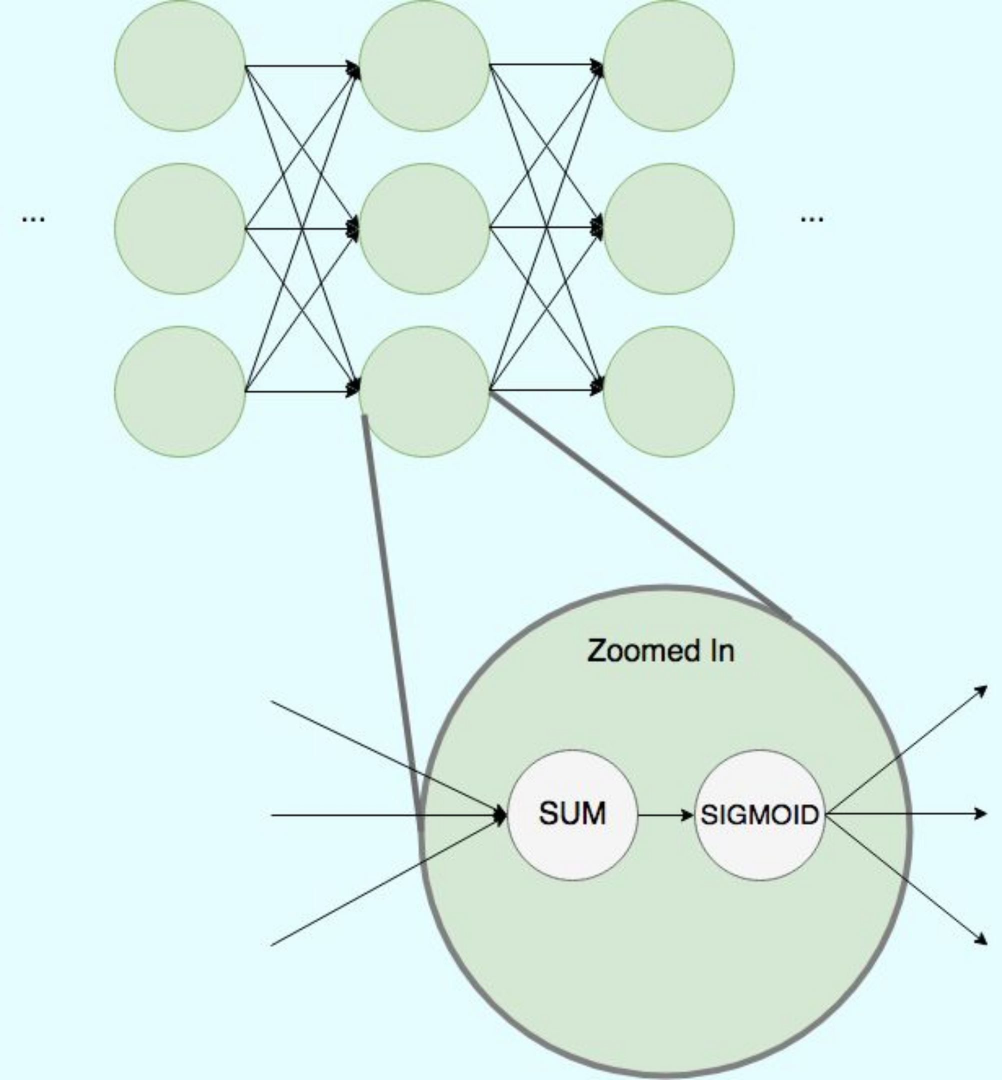
Computational Graphs

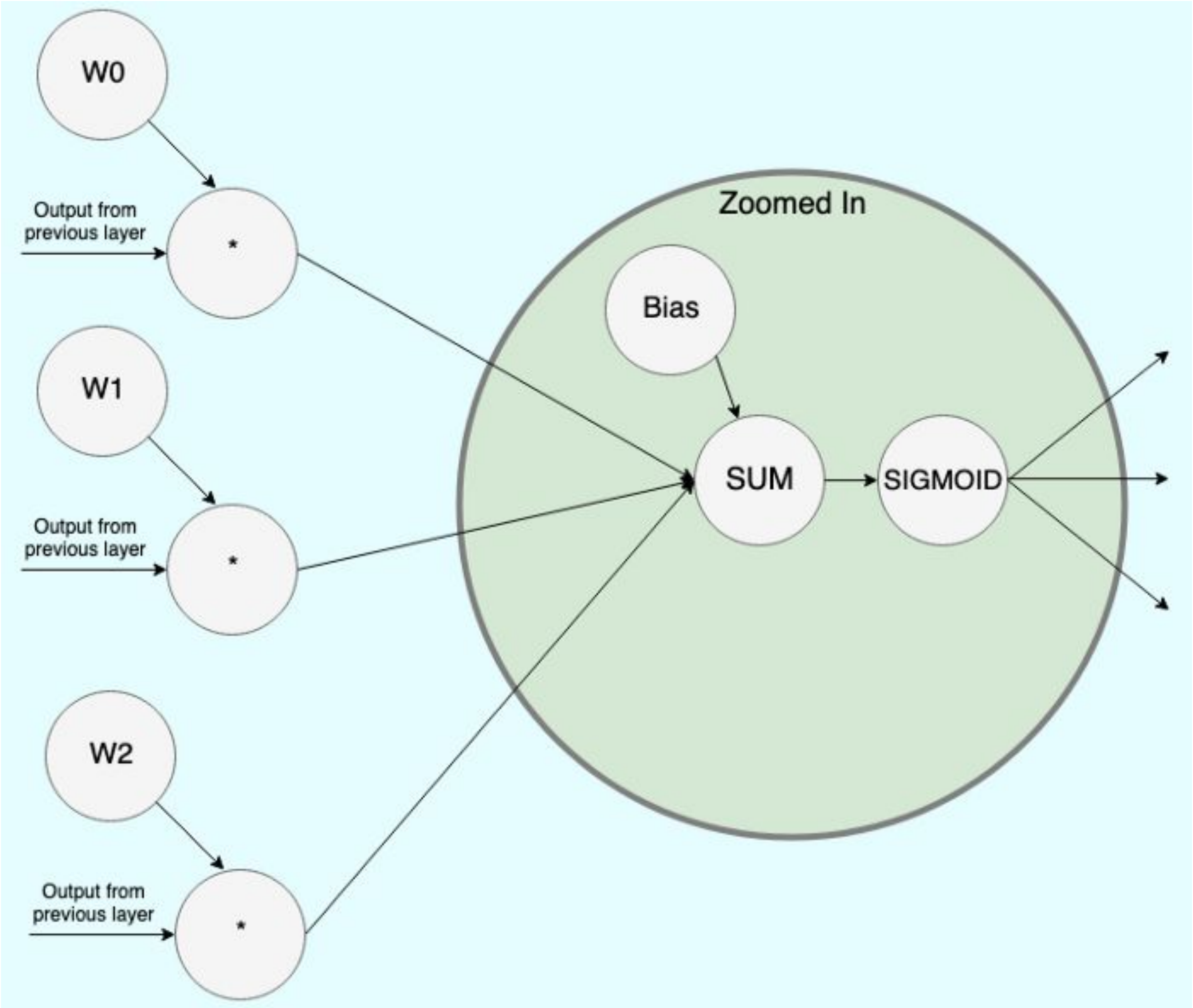
- Represents math in graph format
- Nodes and edges





f(x, y) = ax^2 + bxy + cy^2
a
X
Squared
*
b
+
*
*
c
+
Y
Squared
*


...
...
Zoomed In
SUM
SIGMOID

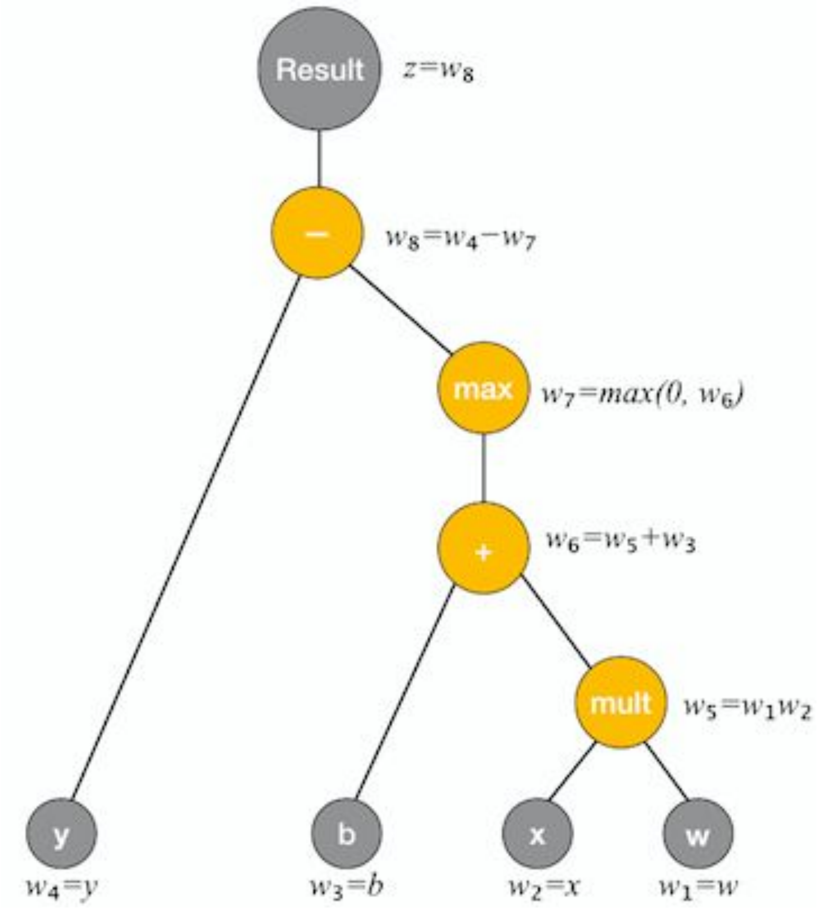


W0
Output from previous layer
*
W1
Output from previous layer
*
W2
Output from previous layer
*
Zoomed In
Bias
SUM
SIGMOID

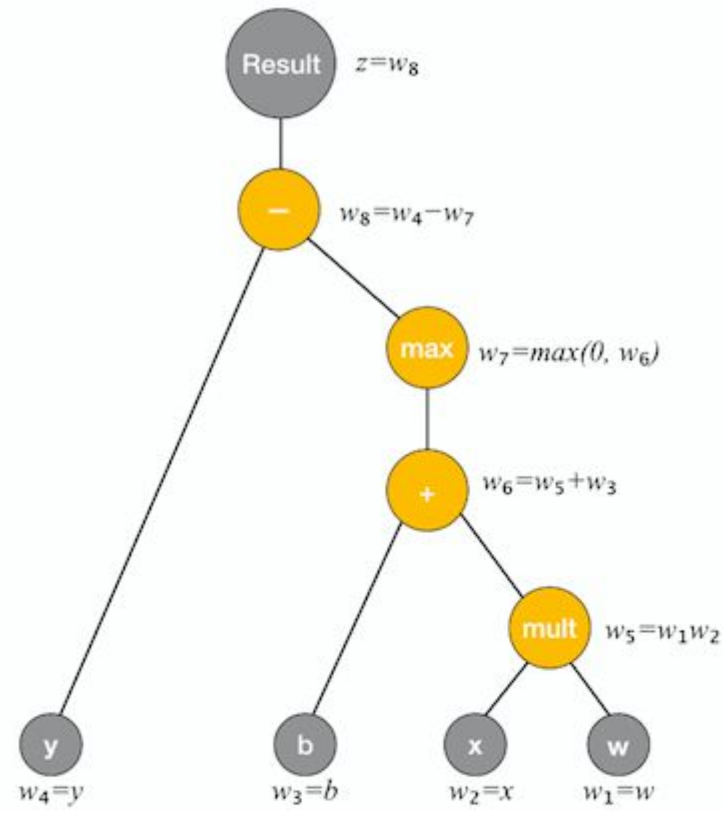
Autodiff

$$C(y, w, x, b) = y - max(0, w \cdot x + b)$$

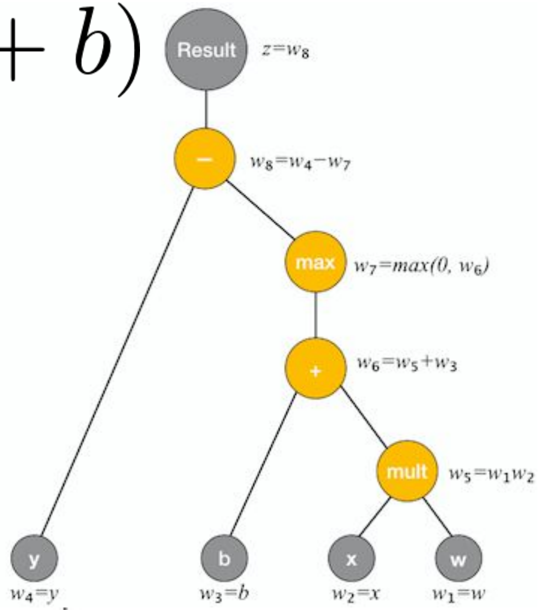
$$C(y, w, x, b) = y - max(0, w \cdot x + b)$$



Node	Expression	Value
w_1	w	2
w_2	x	1
w_3	b	1
w_4	y	5
w_5	$w_1 \cdot w_2$	2
w_6	$w_5 + w_3$	3
w_7	$\max(0, w_6)$	3
w_8	$w_4 - w_7$	2
z	w_8	2

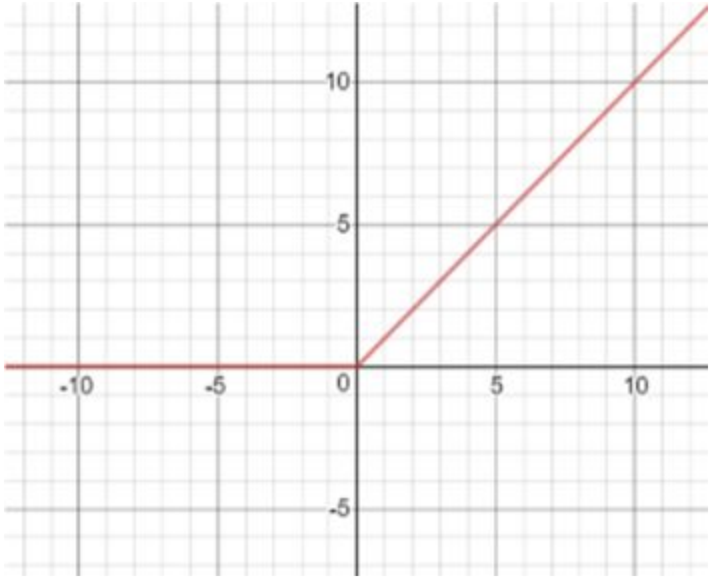


$$C(y, w, x, b) = y - max(0, w \cdot x + b)$$



$$\frac{\delta z}{\delta w_1} = \frac{\delta z}{\delta w_8} \times \frac{\delta w_8}{\delta w_7} \times \frac{\delta w_7}{\delta w_6} \times \frac{\delta w_6}{\delta w_5} \times \frac{\delta w_5}{\delta w_1}$$

$\frac{\delta w_5}{\delta w_1} = \frac{\delta w_1 w_2}{\delta w_1} = w_2$	$\frac{\delta w_5}{\delta w_2} = \frac{\delta w_1 w_2}{\delta w_2} = w_1$
$\frac{\delta w_6}{\delta w_5} = \frac{\delta w_5 + w_3}{\delta w_5} = 1$	$\frac{\delta w_6}{\delta w_3} = \frac{\delta w_5 + w_3}{\delta w_3} = 1$
$\frac{\delta w_7}{\delta w_6} = \frac{\delta \max(0, w_6)}{\delta w_6} = \begin{cases} 0, & x < 0 \\ 1, & x > 0 \end{cases}$	$\frac{\delta w_8}{\delta w_7} = \frac{\delta w_4 - w_7}{\delta w_7} = -1$
$\frac{\delta w_8}{\delta w_4} = \frac{\delta w_4 - w_7}{\delta w_4} = 1$	$\frac{\delta z}{\delta w_8} = 1$


10
5
-5
-10
-5
0
5
10

$$\frac{\delta z}{\delta w_1} = \frac{\delta z}{\delta w_8} \times \frac{\delta w_8}{\delta w_7} \times \frac{\delta w_7}{\delta w_6} \times \frac{\delta w_6}{\delta w_5} \times \frac{\delta w_5}{\delta w_1} = 1 \times (-1) \times \begin{cases} 0, & x < 0 \\ 1, & x > 0 \end{cases} \times 1 \times w_2 = \begin{cases} 0, & x < 0 \\ 1, & x > 0 \end{cases}$$

Frameworks

- Caffe
- NVCaffe
- IntelCaffe
- PyTorch
- PyTorch Lightning
- Tensorflow
- Horovod (DL distribution framework only)

PyTorch

- Probably the most popular package nowadays
- Multi-GPU support (more than one node)
- Supports cuDNN and MKL-DNN
- Suppports various communication backends: Gloo, MPI, NCCL

Tensorflow

- Also very popular
- Multi-GPU in one node through *tf.device*
- Parallelism across nodes with *tf.distribute*
- Support for TPU
- Supports cuDNN and MKL-DNN
- Supports NCCL allreduce
- For CPU, build from source

Tensorflow optimization tips

- Use *TFRecords* to prevent I/) bottlenecks
- Overlap computation and data preparation using *tf.data.Dataset.prefetch*
- Parallelize data transformation using *tf.data.Dataset.map*
- If data fits in memory, use *tf.data.Dataset.cache*

Horovod

Is a distribution framework for deep learning (not a deep learning framework itself). Design goals:

- Minimal code changes to make serial program distributed
- High performance distribution

Horovod

- Support for Keras, MXNet, Tensorflow and PyTorch
- Supports MPI and MCCL as communication backends
- Requires about 6 lines of code changes
- Has it’s own profiling ability, making it easy to assess communication overhead.

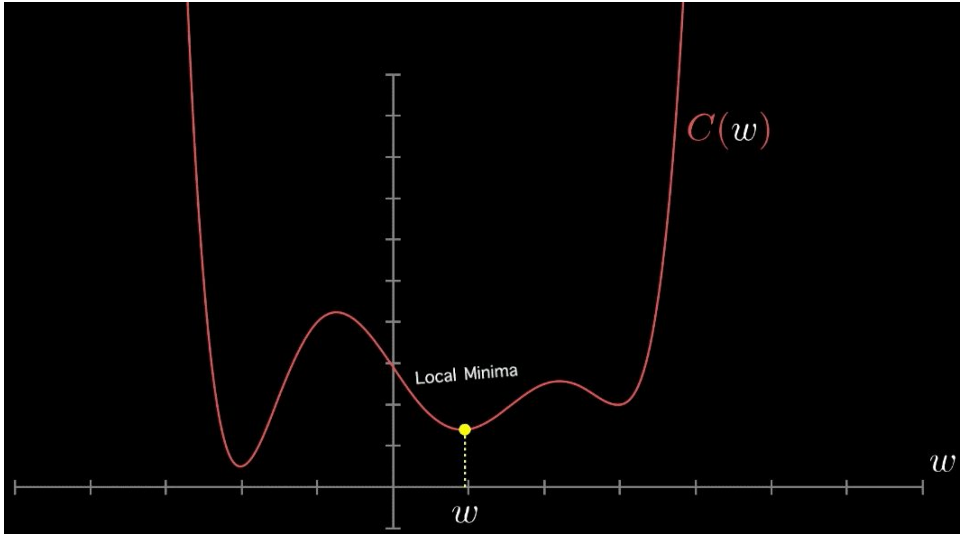
PyTorch Lightning

- PyTorch wrapper for high performance AI research
- Supports various distribution strategies (horovod, data parallel, model parallel)
- Has other convenient features too.

Optimizers

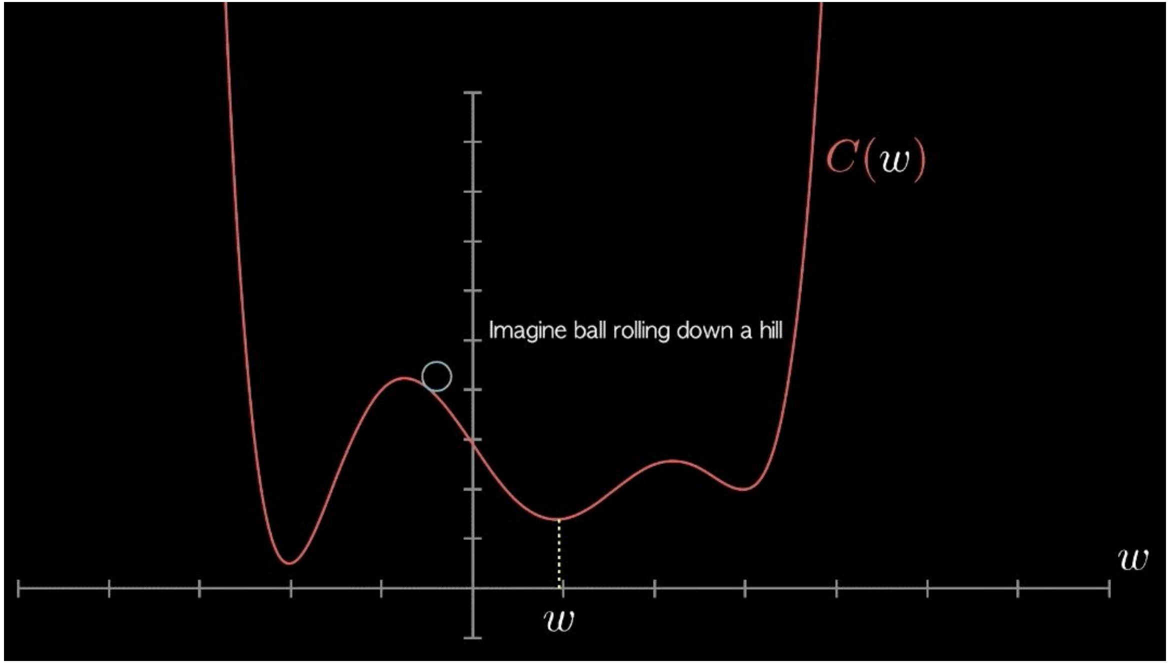
SGD

$$\theta_j = \theta_j - \eta \cdot \overbrace{\frac{\partial C}{\partial \theta_j}}^{\text{Backprop}}$$

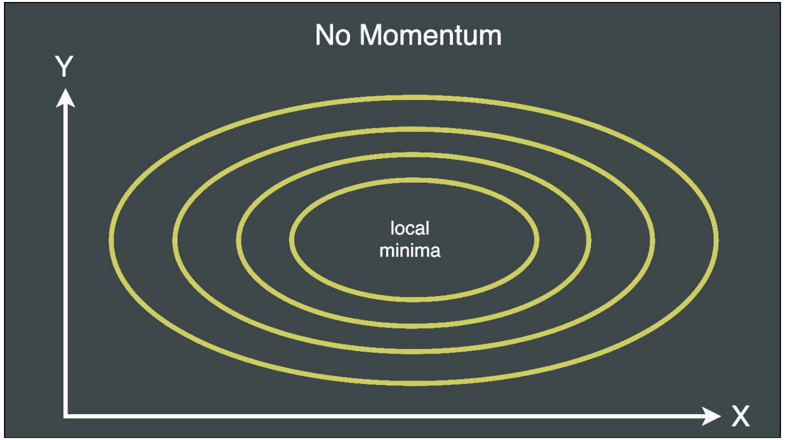


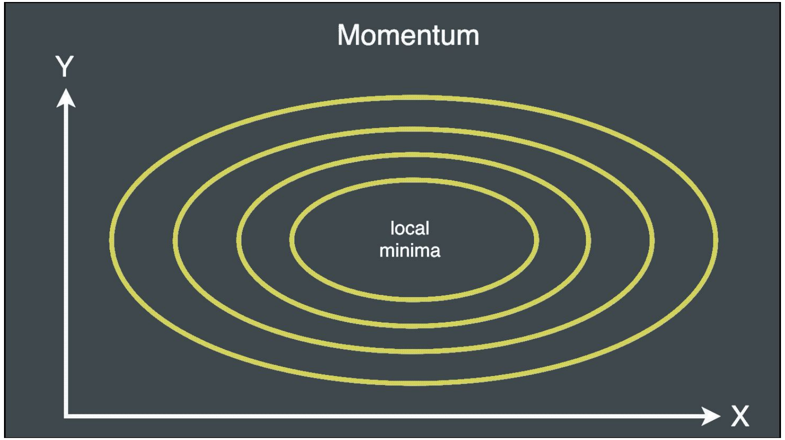
Momentum

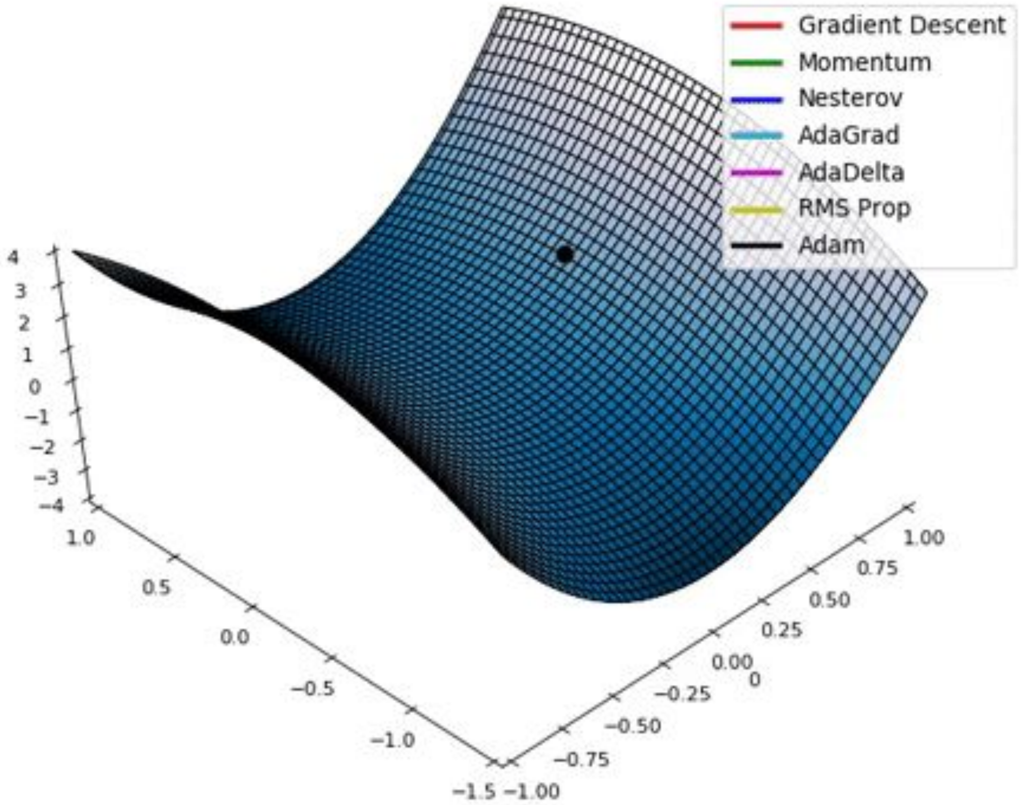
$$\theta = \theta - \eta \nabla J(\theta) + \gamma v_t$$



Momentum







Gradient Descent
Momentum
Nesterov
AdaGrad
AdaDelta
RMS Prop
Adam
4
3
2
1
0
−1
−2
−3
−4
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.00
−0.75
−0.50
−0.25
0.00
0
0.25
0.50
0.75
1.00

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while θ_t not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return θ_t (Resulting parameters)

What optimizer to pick?

Adam is always a safe bet

Thank you!