ML @ URL

Episode 2

Deep learning frameworks

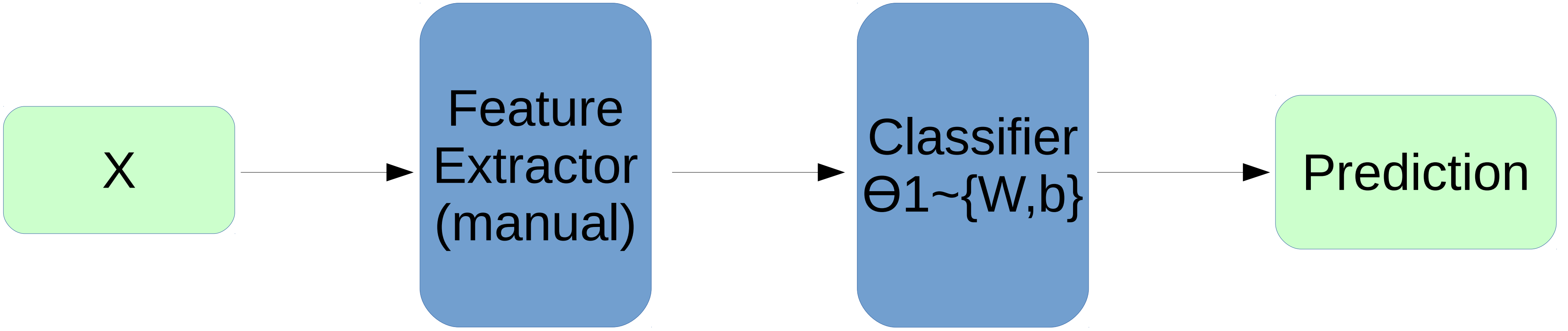
Yandex Data Factory

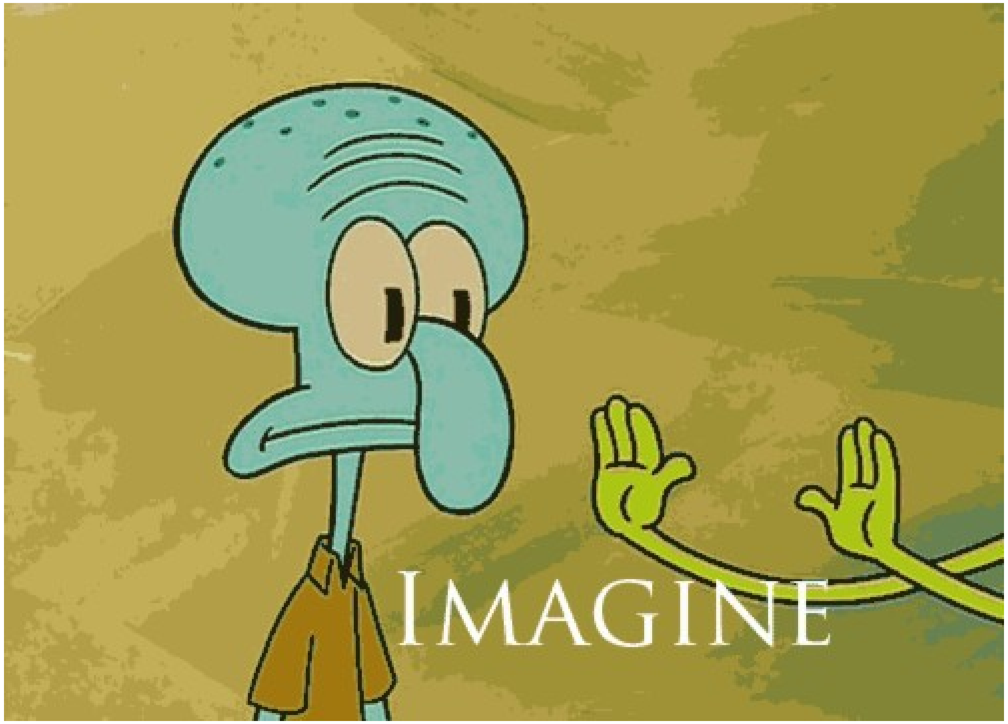
LAMBDA

British Hedgehog Preservation Society

Previously on deep learning...

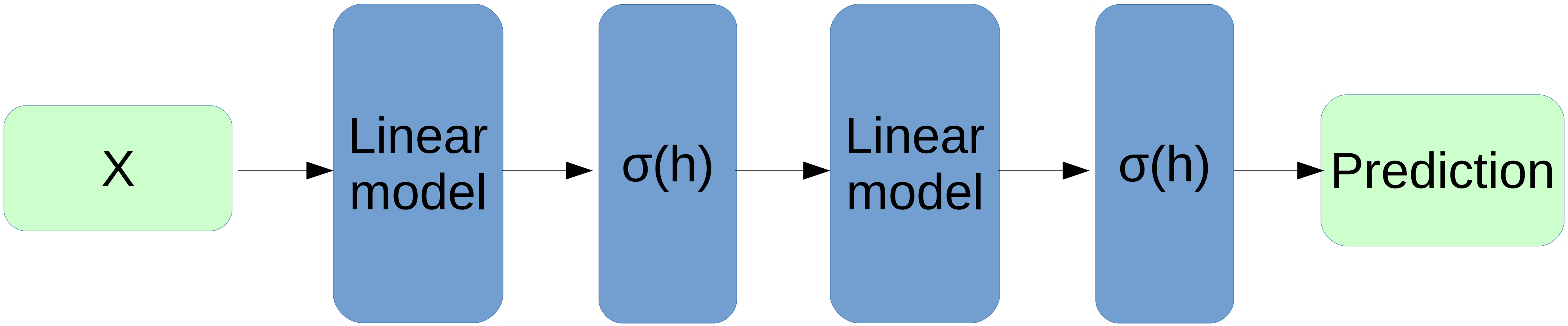
Feature extraction





Features would tune to your problem automatically!

Simple neural network



Trains with stochastic gradient descent!
or momentum/rmsprop/adam/...

Connectionist phrasebook

- Layer – a building block for NNs :
  - “Dense layer”: $f(x) = Wx+b$
  - “Nonlinearity layer”: $f(x) = \sigma(x)$
  - Input layer, output layer
  - A few more we gonna cover later

- Activation – layer output
  - i.e. some intermediate signal in the NN

- Backpropagation – a fancy word for “chain rule”

Backpropagation

TL;DR: backprop = chain rule*

$$\frac{\partial f\left(g\left(x\right)\right)}{\partial x} = \frac{\partial f\left(g\left(x\right)\right)}{\partial g\left(x\right)} \cdot \frac{\partial g\left(x\right)}{\partial x}$$

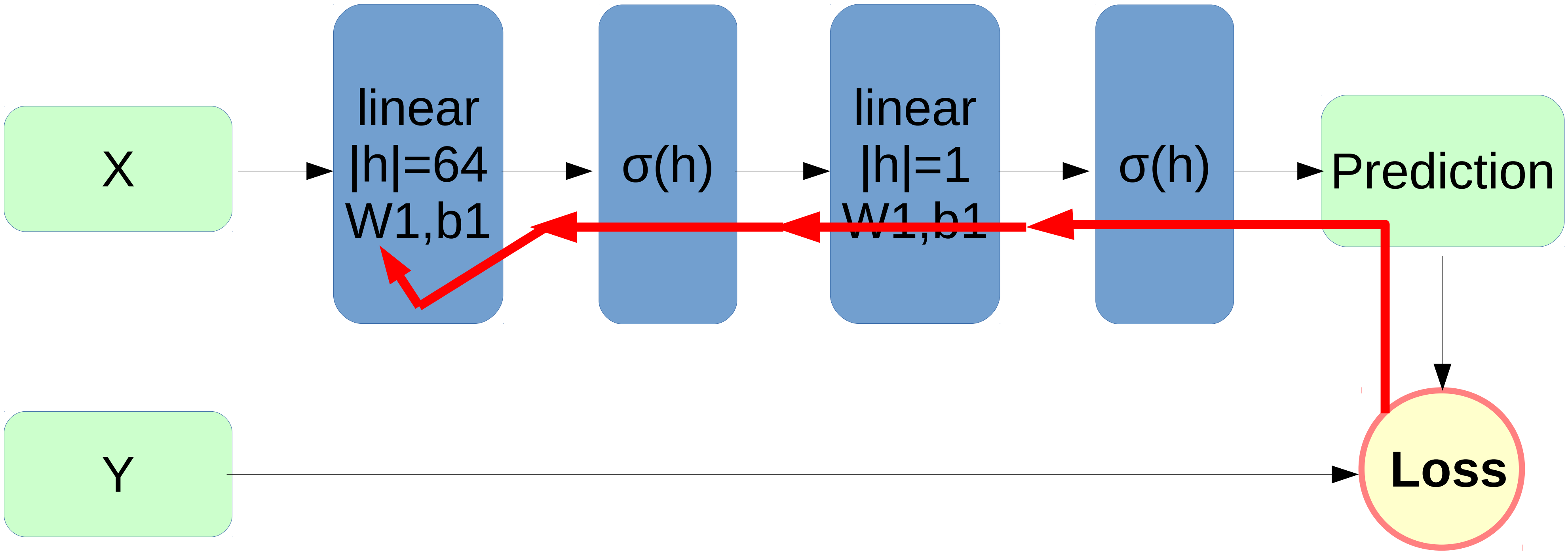
Backpropagation

TL;DR: backprop = chain rule*

$$\frac{\partial f(g(x))}{\partial x} = \frac{\partial f(g(x))}{\partial g(x)} \cdot \frac{\partial g(x)}{\partial x}$$

* g and x can be vectors/vectors/tensors

Backpropagation



$$\frac{\partial L}{\partial w1} = \frac{\partial L}{\partial \sigma} \cdot \frac{\partial \sigma}{\partial\, linear_{w2,b2}} \cdot \frac{\partial\, linear_{w2,b2}}{\partial \sigma} \cdot \frac{\partial \sigma}{\partial\, linear_{w1,b1}} \cdot \frac{\partial\, linear_{w1,b1}}{\partial w1}$$

Matrix derivatives we used

$$\text{sigmoid}: \frac{\partial L}{\partial \sigma(x)} \cdot [\sigma(x) \cdot (1 - \sigma(x))]$$

Works for any kind of x
(scalar, vector, matrix, tensor)

$$\text{linear over X}: \frac{\partial L}{\partial W \times X + b} \times W^T$$

$$\text{linear over W}: \frac{1}{\|X\|} \cdot X^T \times \frac{\partial L}{\partial [X \times W + b]}$$

Matrix derivatives we used

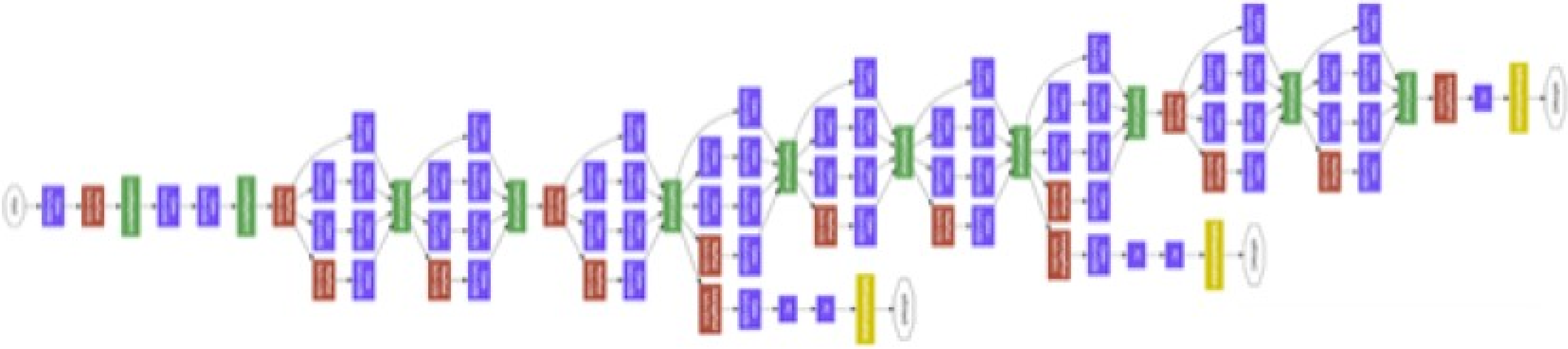
$$\text{sigmoid}: \frac{\partial L}{\partial \sigma(x)} \cdot [\sigma(x) \cdot (1 - \sigma(x))]$$

Works for any kind of x
(scalar, vector, matrix, tensor)

$$\text{linear over X}: \frac{\partial L}{\partial W \times X + b} \times W^T$$

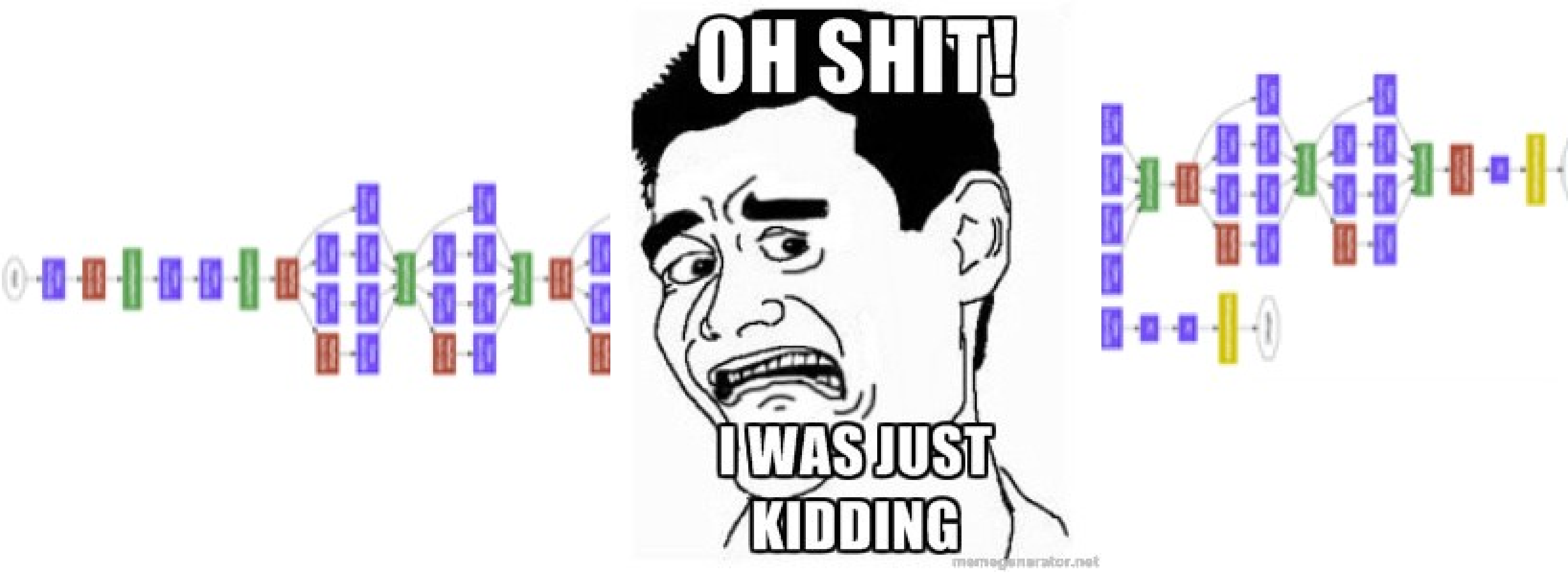
$$\text{linear over W}: \frac{1}{\|X\|} \cdot X^T \times \frac{\partial L}{\partial [X \times W + b]}$$

And now let's differentiate



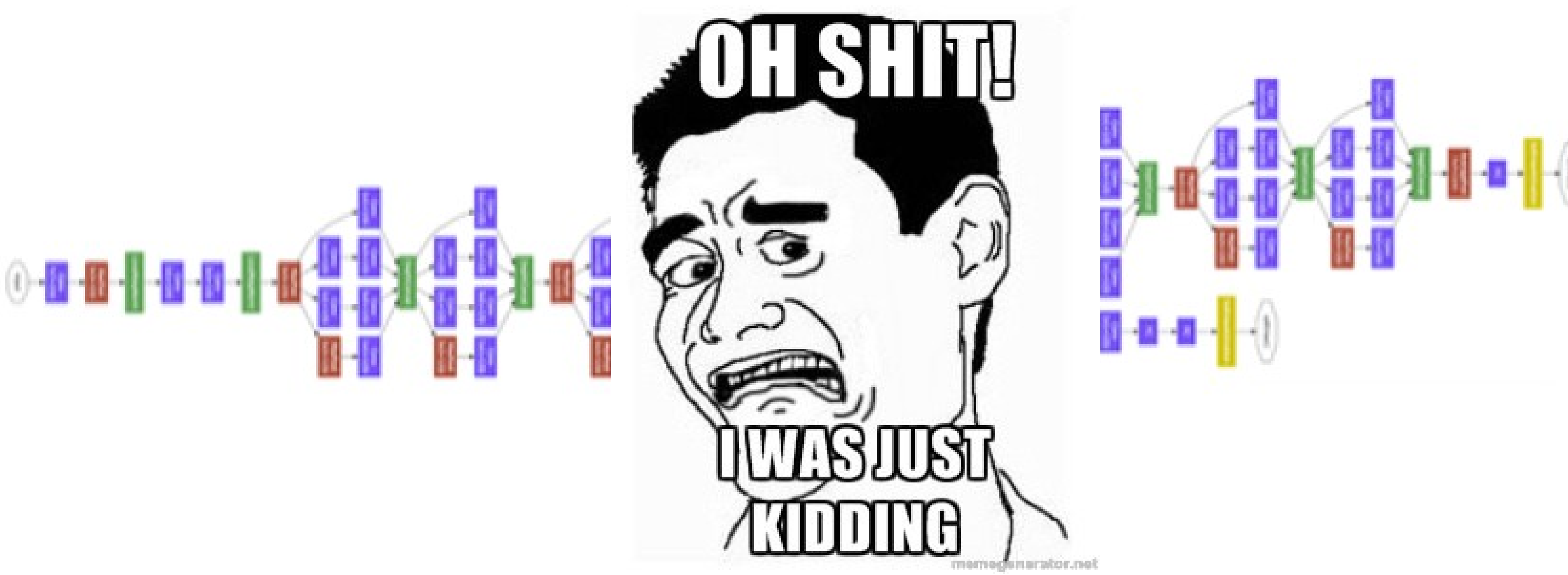
- 5+ types of layers
- each with different dimensions
- parallel branches with independent losses
- several nonlinearities

And now let's differentiate



- 5+ types of layers
- each with different dimensions
- parallel branches with independent losses
- several nonlinearities

Deep learning frameworks

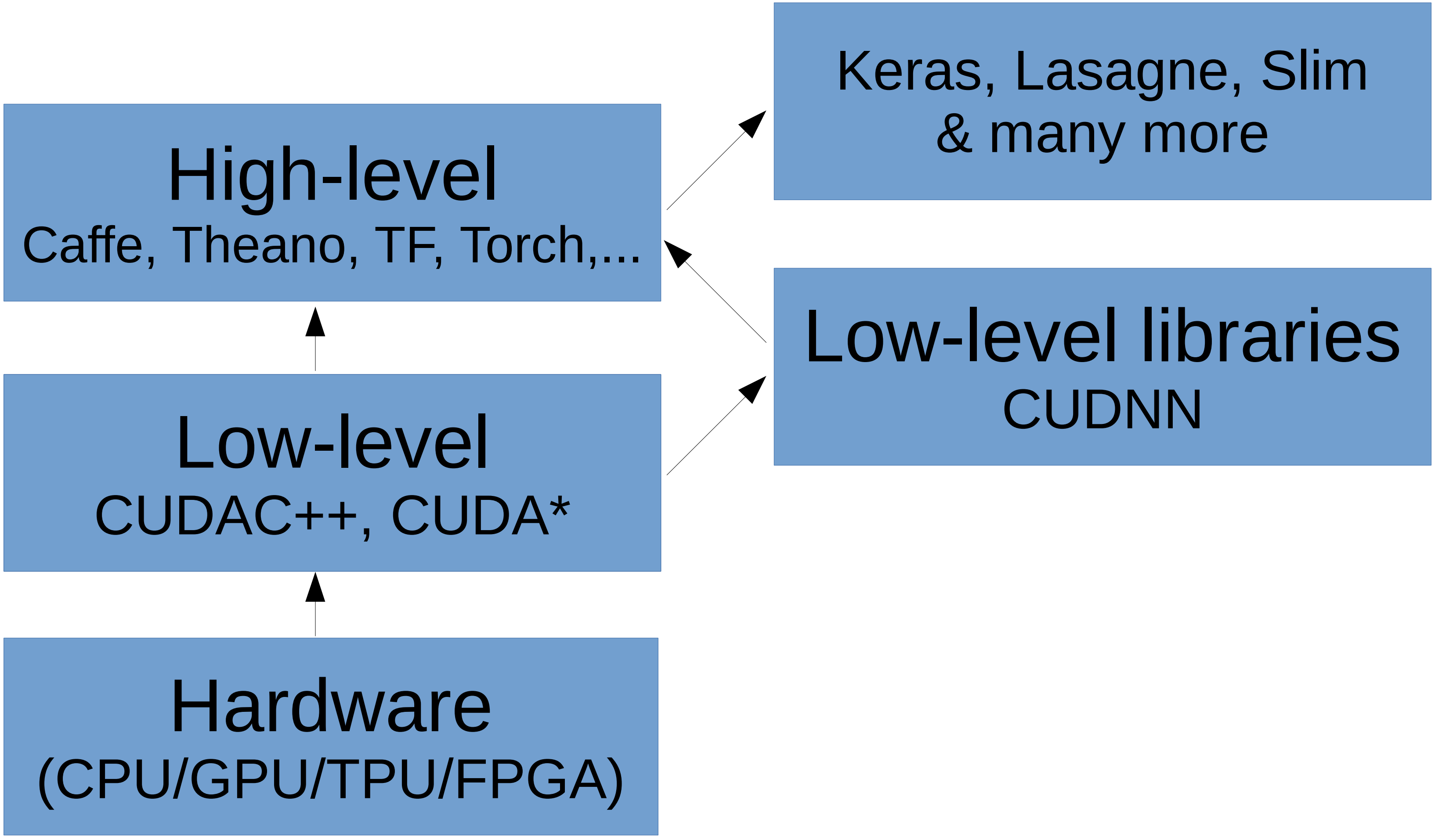


Dream deep learning framework

- Automatic gradients
- Pre-implemented popular “layers”
- Optimized computation (GPU & multi-CPU)

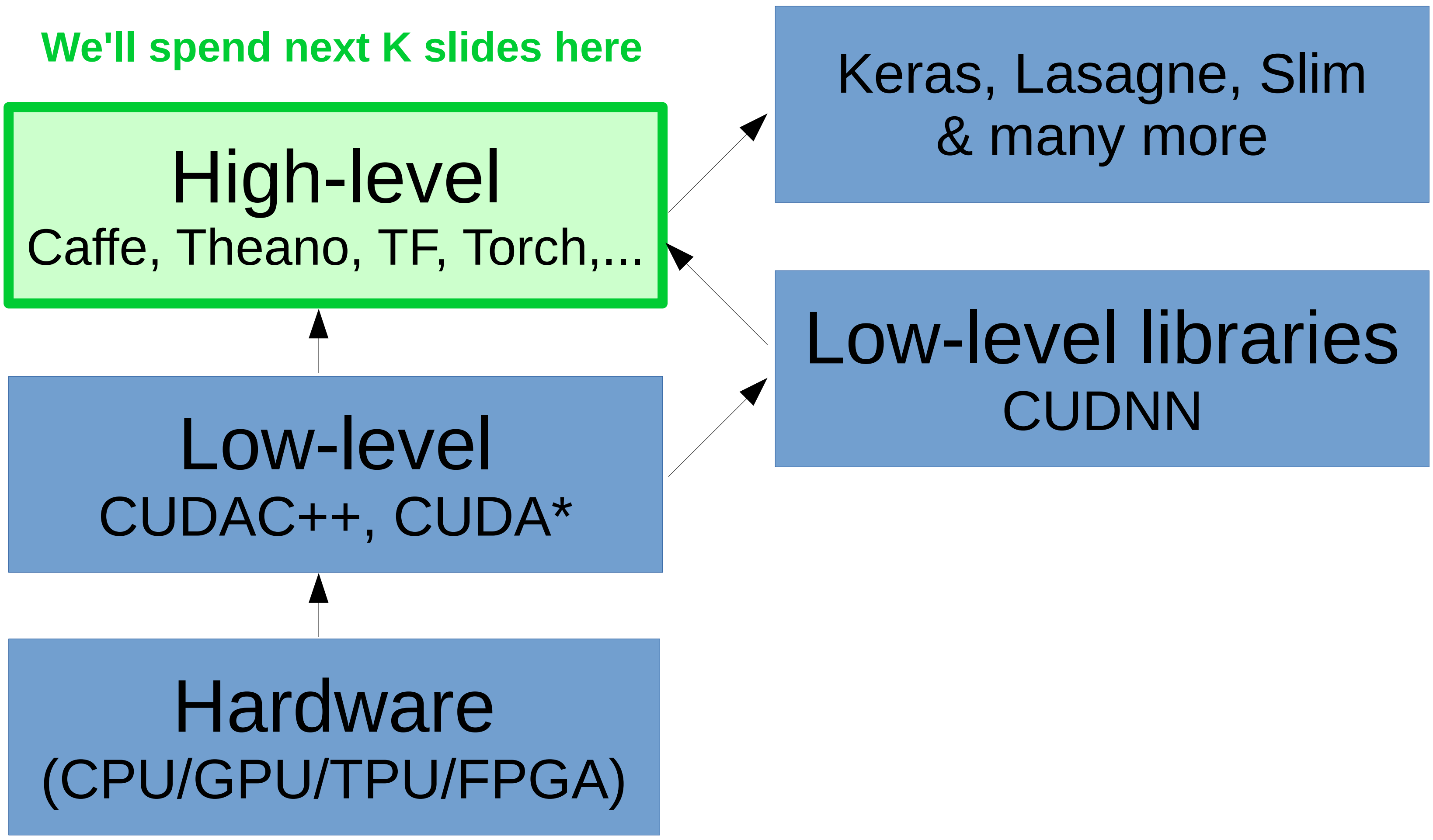
Deep learning frameworks

- Core idea: helps you define and train neural nets



Deep learning frameworks

- Core idea: helps you define and train neural nets



Deep learning frameworks

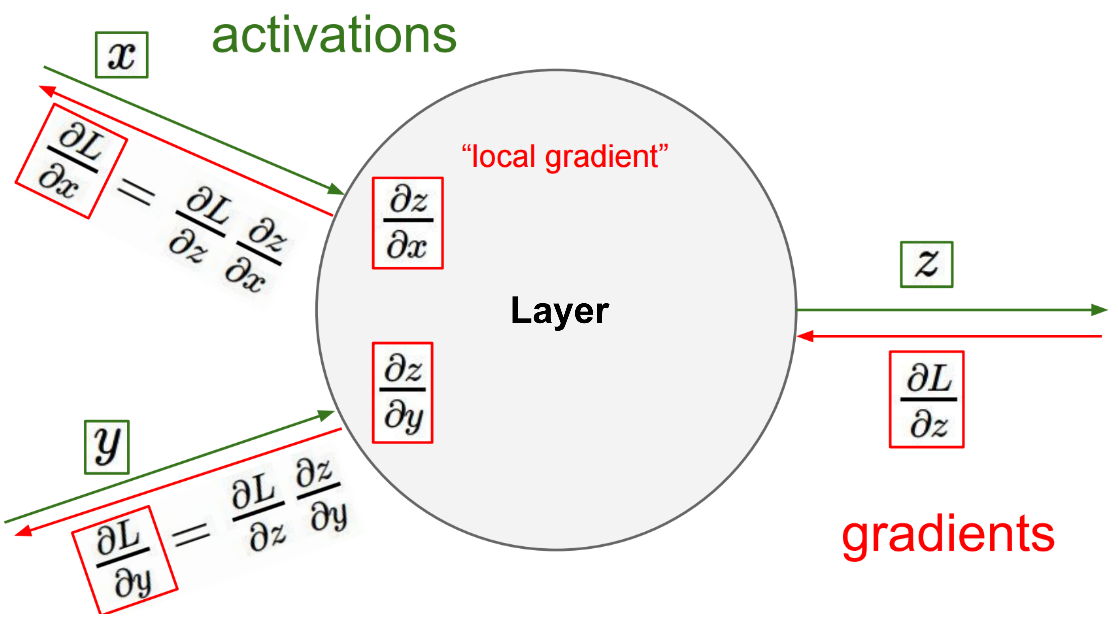
Layer-based frameworks:
  Same idea as in our hand-made neural net

Deep learning frameworks

Layer-based frameworks:

Same idea as in our hand-made neural net

this one - http://bit.ly/2w9kAHm



Deep learning frameworks

Caffe

```
name: "LeNet"
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  param {lr_mult: 1}
  param {lr_mult: 2}
  convolution_param {
    num_output: 20
    kernel_size: 5
    stride: 1
    weight_filler {
      type: "xavier"
    }}}
....
```

130 lines

You define model in config file by stacking layers.

Then train like this:

```
caffe train -solver examples/mnist/lenet_solver.prototxt
```

Deep learning frameworks

Caffe

```
name: "LeNet"
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  param {lr_mult: 1}
  param {lr_mult: 2}
  convolution_param {
    num_output: 20
    kernel_size: 5
    stride: 1
    weight_filler {
      type: "xavier"
    }}}
....
```

130 lines

+ Easy to deploy (C++)
+ A lot of pre-trained models (model zoo)
- Model as protobuf
- Hard to build new layers
- Hard to debug

Industry standard for computer vision

Symbolic graphs

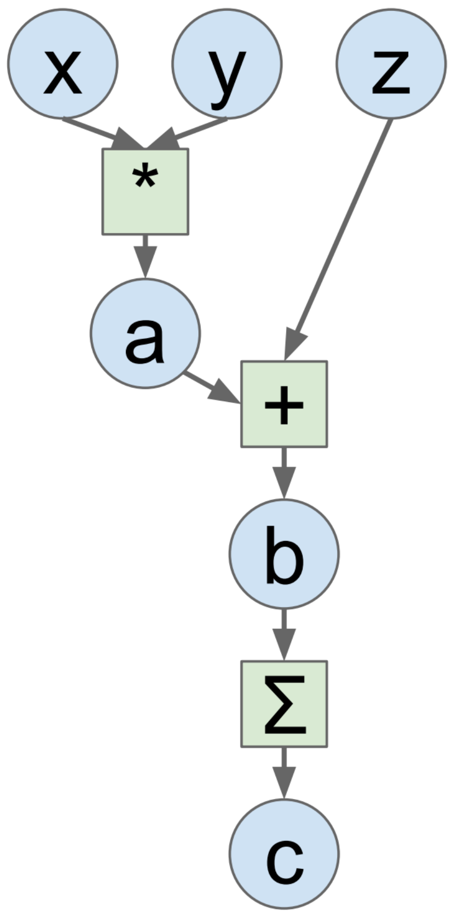
What will your CPU do
when you write this?

```
N, D = 3, 4
x = np.random.randn(N, D)
y = np.random.randn(N, D)
z = np.random.randn(N, D)

a = x * y
b = a + z
c = np.sum(b)
```


This and many further slides taken from Ars Ashukha @deepbayes2017

Symbolic graphs



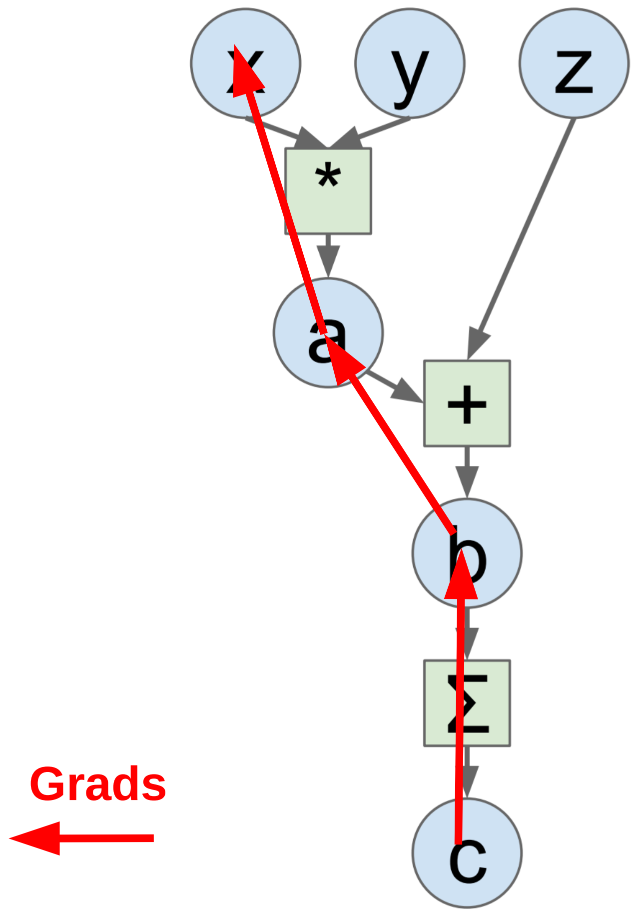

```
N, D = 3, 4
x = np.random.randn(N, D)
y = np.random.randn(N, D)
z = np.random.randn(N, D)

a = x * y
b = a + z
c = np.sum(b)
```

Idea: let's define this graph explicitly!

This and many further slides taken from Ars Ashukha @deepbayes2017

Symbolic graphs




```
N, D = 3, 4
x = np.random.randn(N, D)
y = np.random.randn(N, D)
z = np.random.randn(N, D)

a = x * y
b = a + z
c = np.sum(b)
```

+ Automatic gradients!
+ Easy to build new layers
+ We can optimize the Graph
- Graph is static during training
- Need time to compile/optimize
- Hard to debug

This and many further slides taken from Ars Ashukha @deepbayes2017

60 seconds of holywar

theano **and** TensorFlow™

- Graph optimization
- Numpy-like interface
- Great for RNNs
- Inconvenient randomness
- Worse multi-gpu support
- Yet another argument

- Easier to deploy
- Graph visualization
- Google! (and hype)
- Worse optimization
- Sessions, graphs
- Yet another argument

60 seconds of coding

Note to self: start coding!

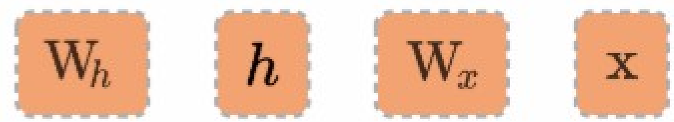
Dynamic graphs

Chainer, DyNet, Pytorch

A graph is created on the fly

```
from torch.autograd import Variable

x = Variable(torch.randn(1, 10))
prev_h = Variable(torch.randn(1, 20))
W_h = Variable(torch.randn(20, 20))
W_x = Variable(torch.randn(20, 10))
```



Dynamic graphs

Chainer, DyNet, Pytorch

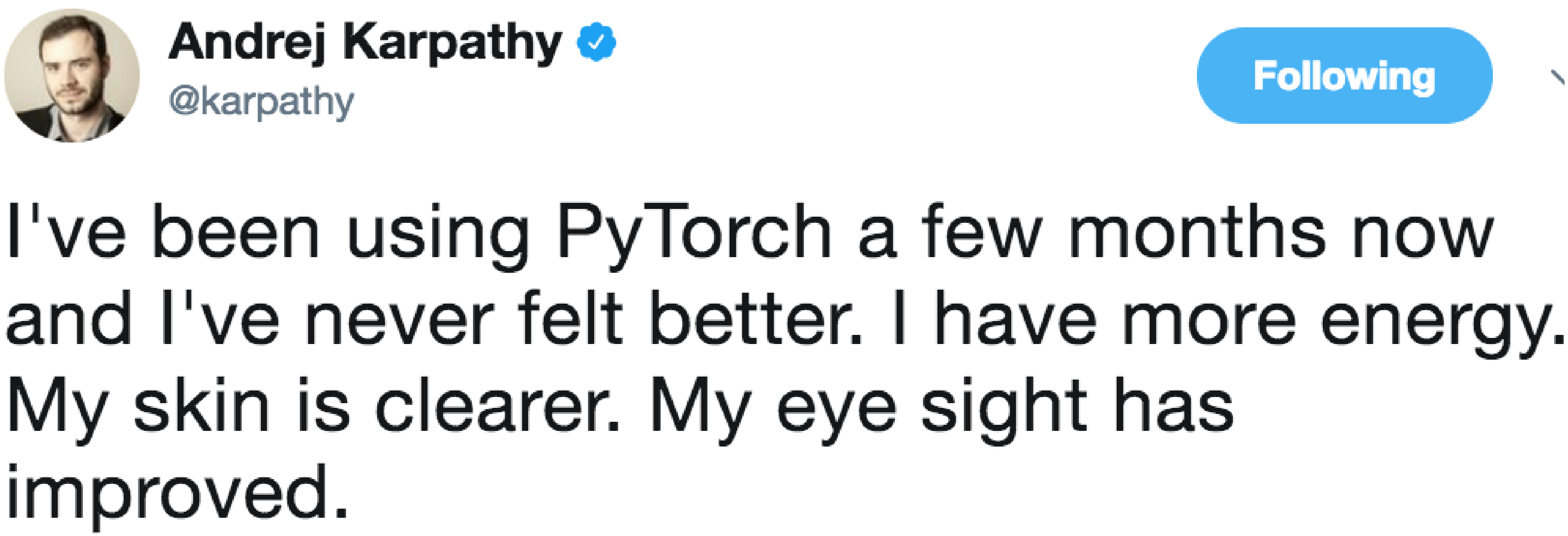
PYTORCH

Chainer

+ Can change graph on the fly
+ Can get value of any tensor at any time (easy debugging)
- Hard to optimize graphs (especially large graphs)
- Still early development

Researchers love them!

Dynamic graphs



Researchers love them!

We gonna be using pytorch...

PYTORCH