Deep learning

Episode 2, 2025

Deep learning whereabouts

A catch-all lecture in philosophy, tricks and frameworks

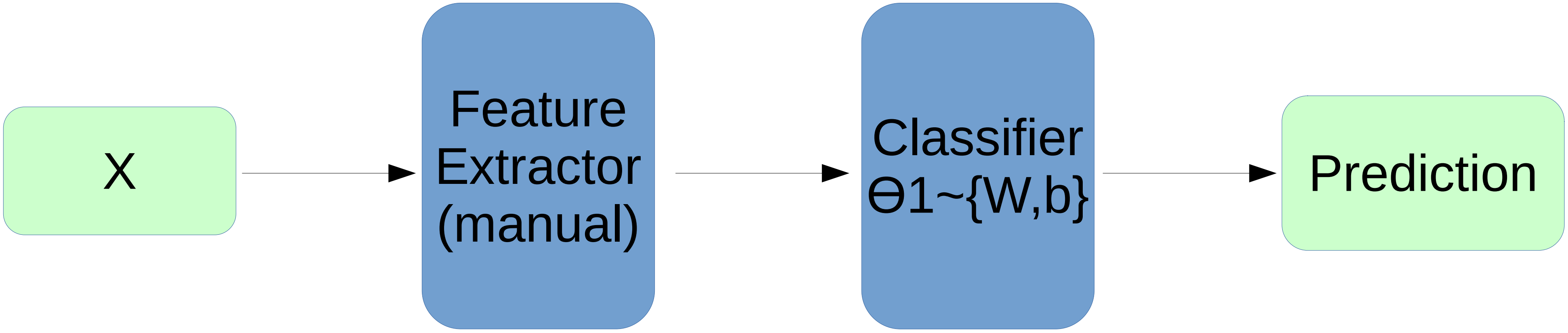
Yandex Data Factory

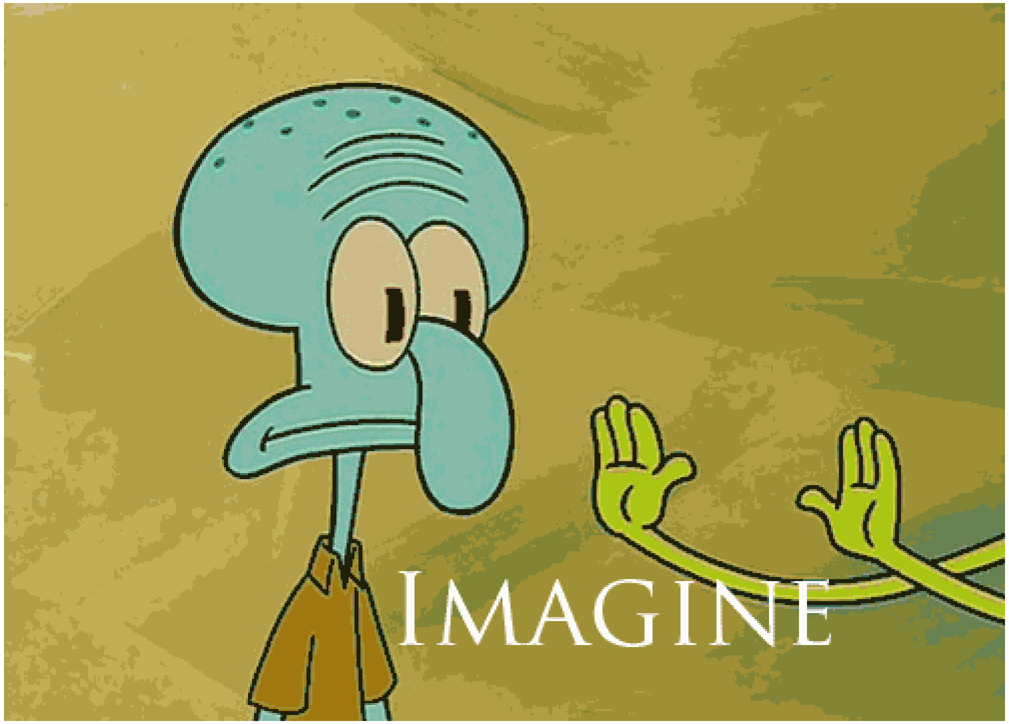
LAMBDA

British Hedgehog Preservation Society

Previously on deep learning...

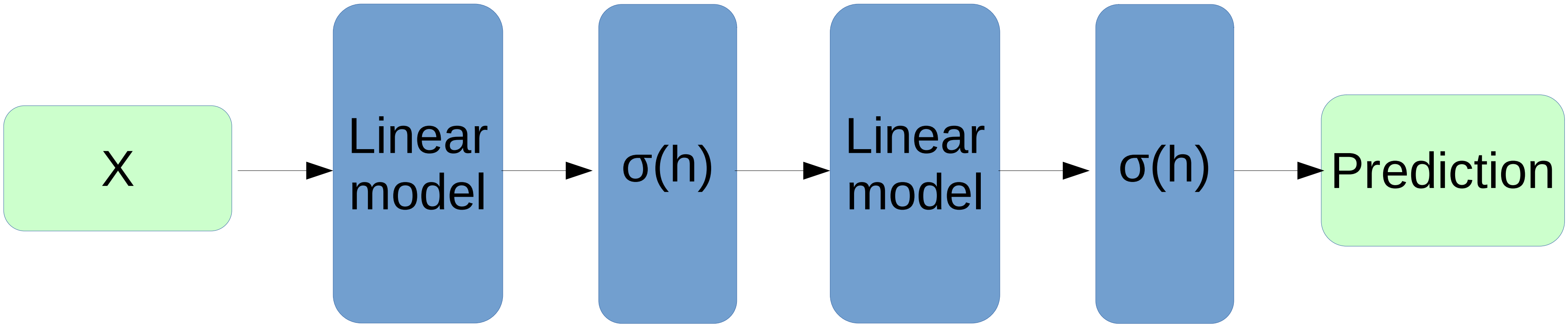
Feature extraction





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*

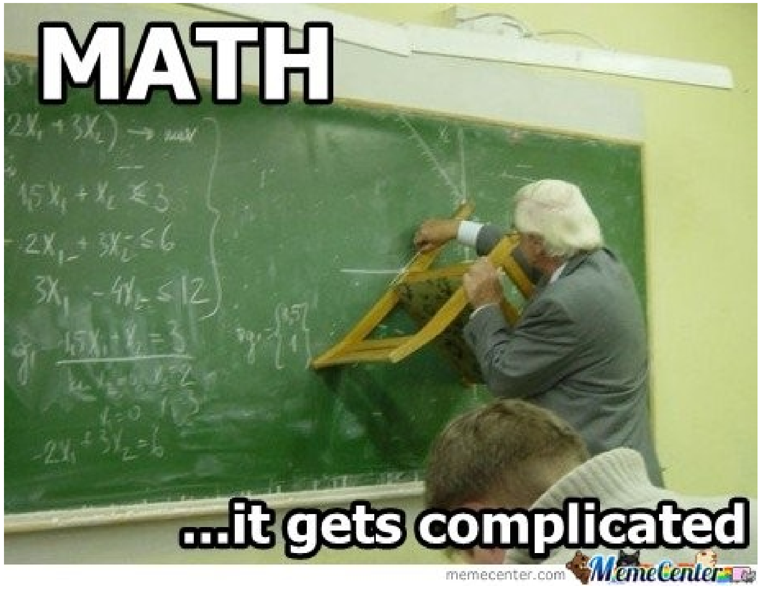
$$\frac{\partial f(g(x))}{\partial x} = \frac{\partial f(g(x))}{\partial g(x)} \cdot \frac{\partial g(x)}{\partial x}$$

Backpropagation

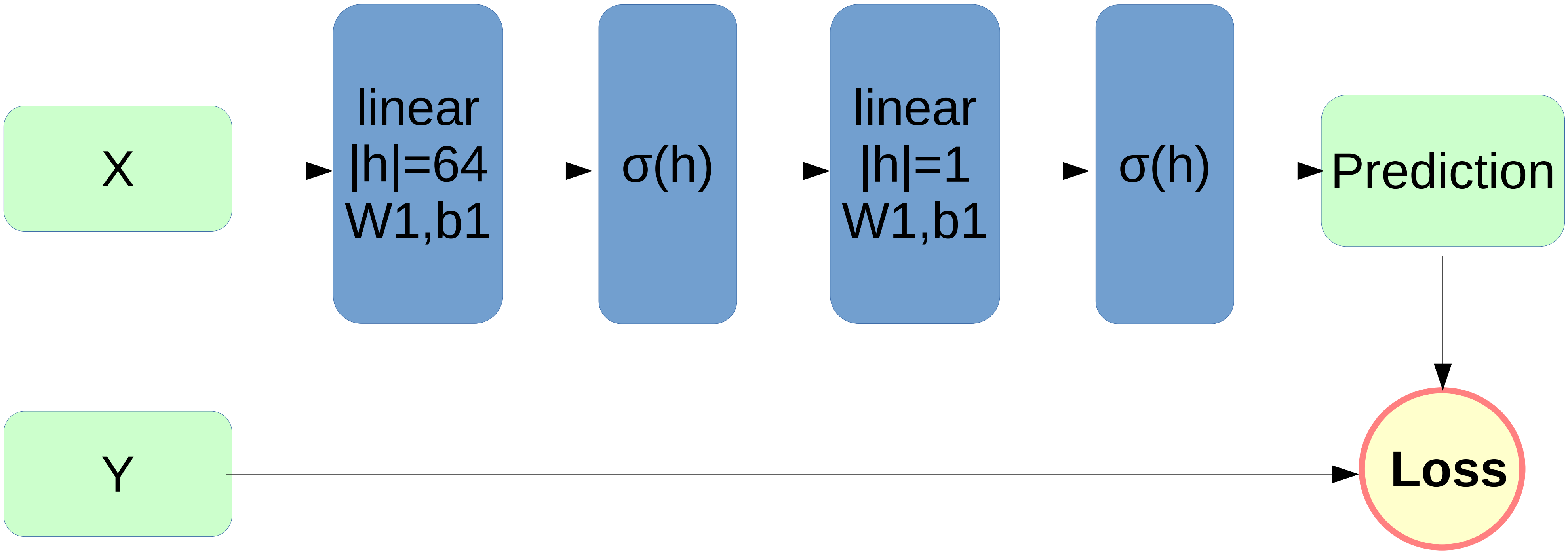
TL;DR: backprop = chain rule*

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

* g and x can be vectors/vectors/tensors

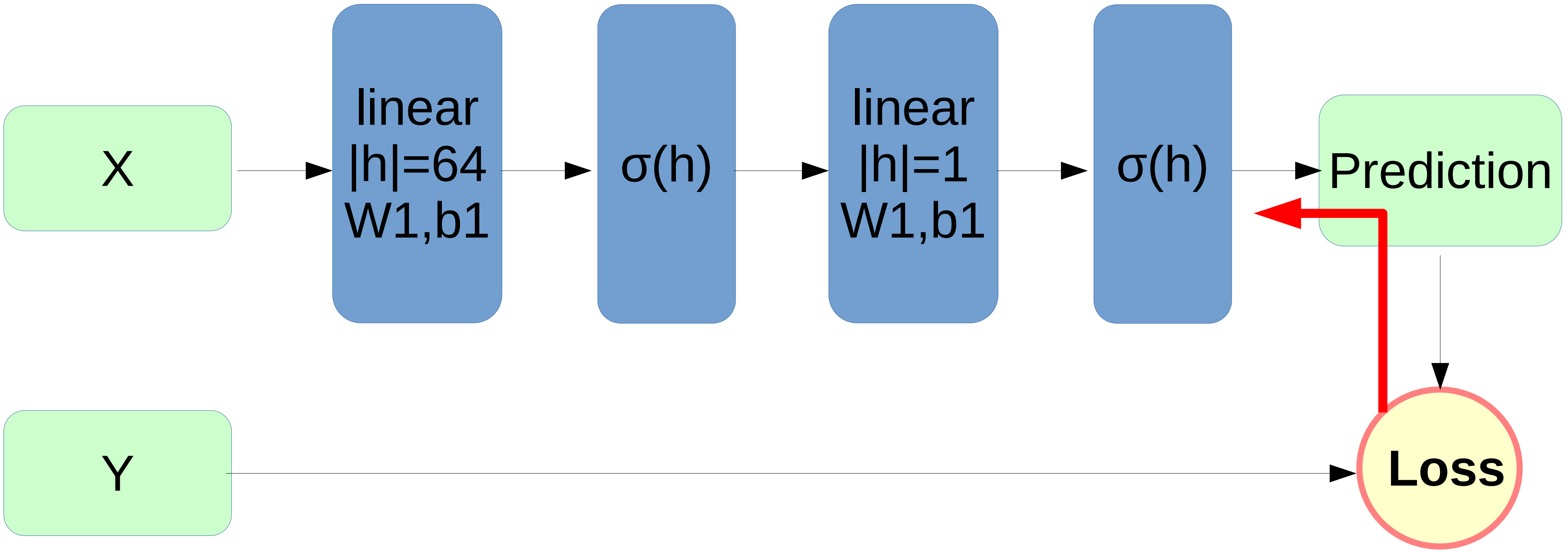


Backpropagation



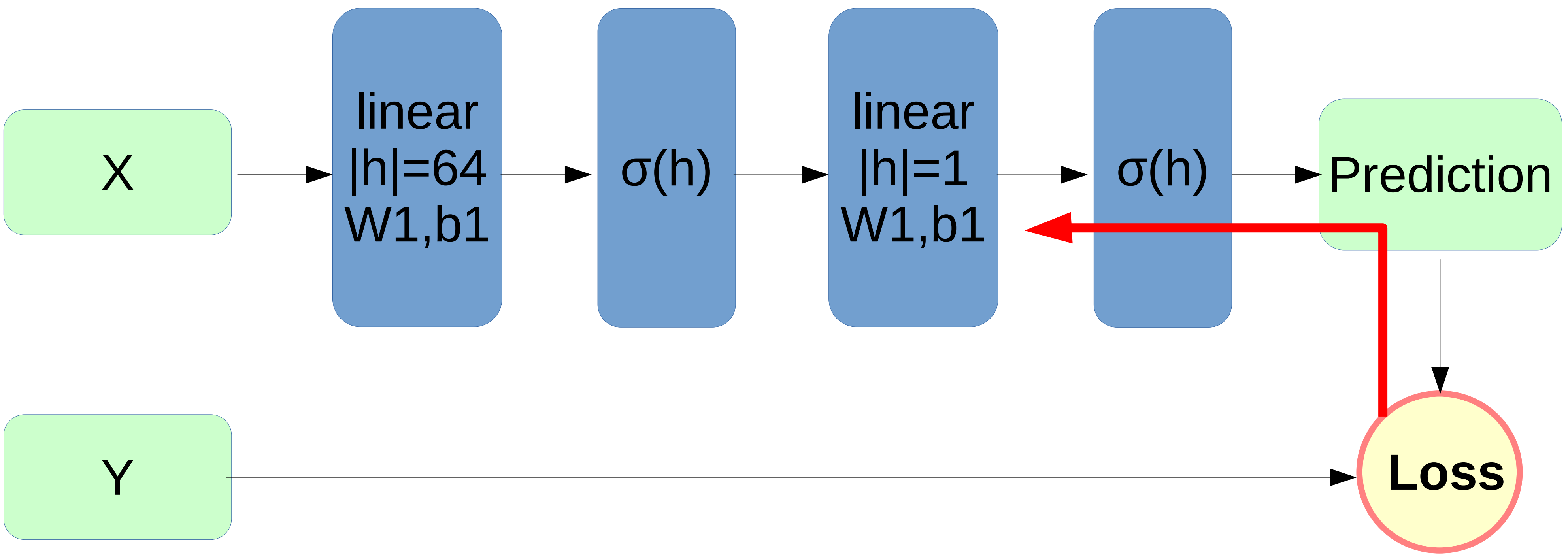
$$\frac{\partial L(\sigma(linear_{w2,b2}(\sigma(linear_{w1,b1}(x)))))}{\partial w1} = \ldots$$

Backpropagation



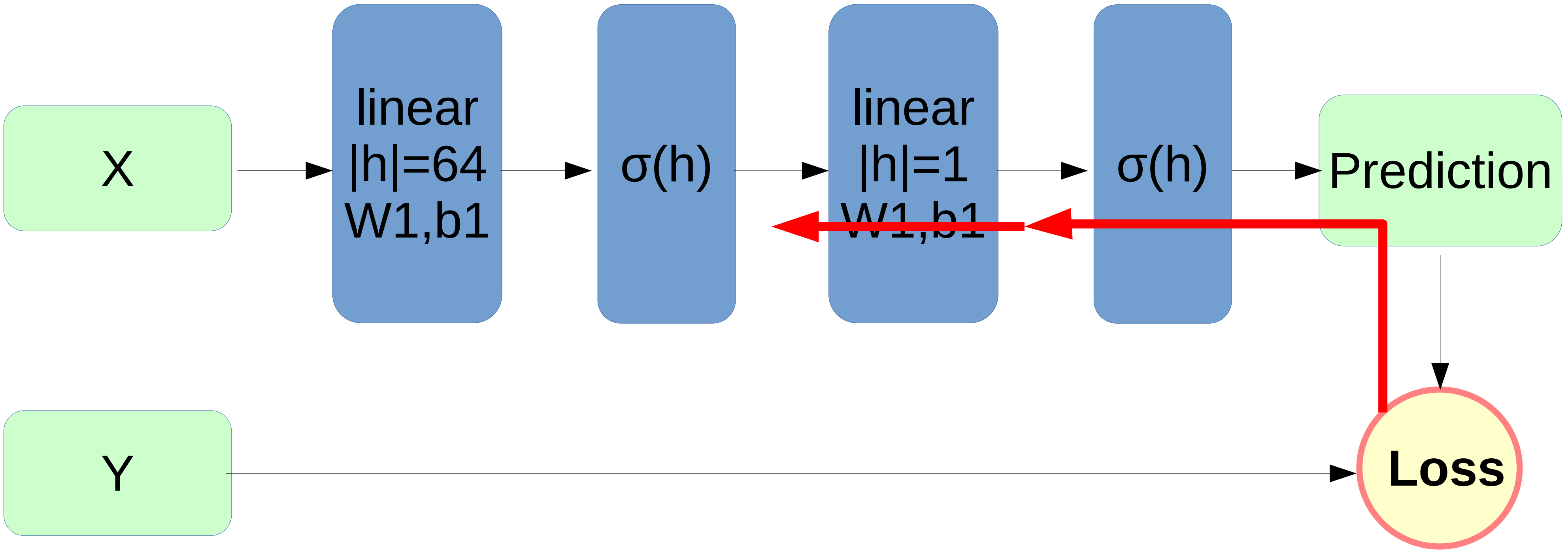
$$\frac{\partial L}{\partial w1} = \frac{\partial L}{\partial \sigma} \cdot$$

Backpropagation



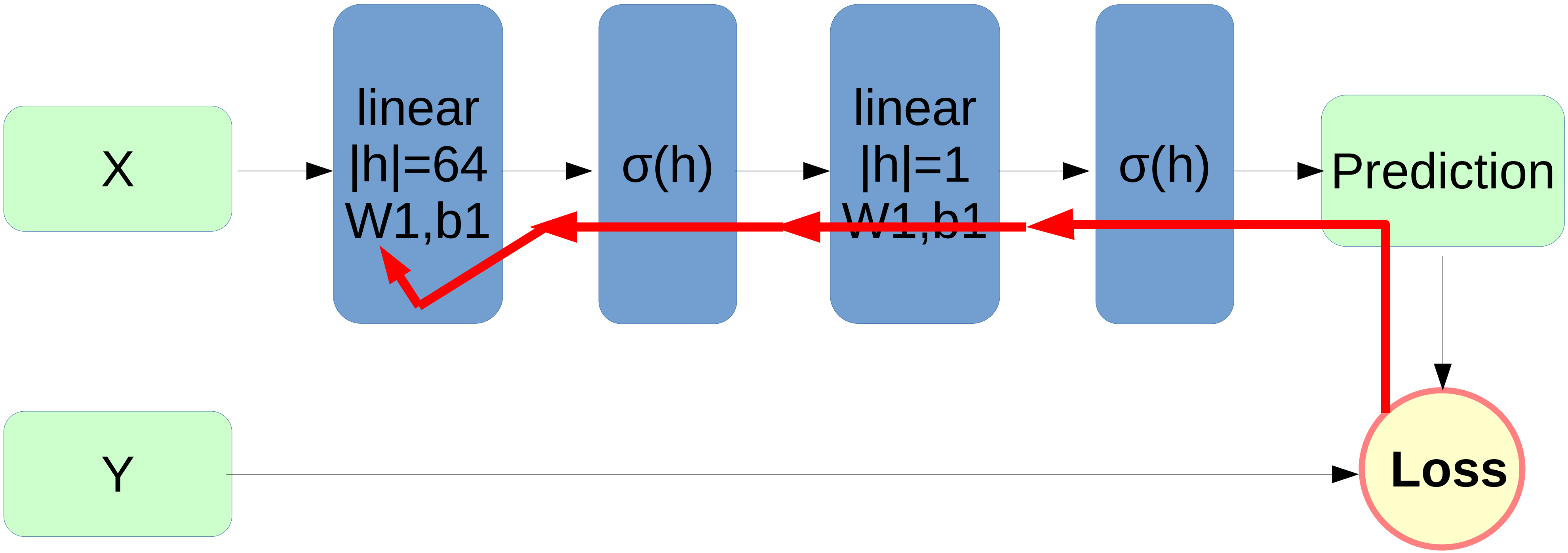
$$\frac{\partial L}{\partial w1} = \frac{\partial L}{\partial \sigma} \cdot \frac{\partial \sigma}{\partial \, linear_{w2,b2}} \cdot$$

Backpropagation



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

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

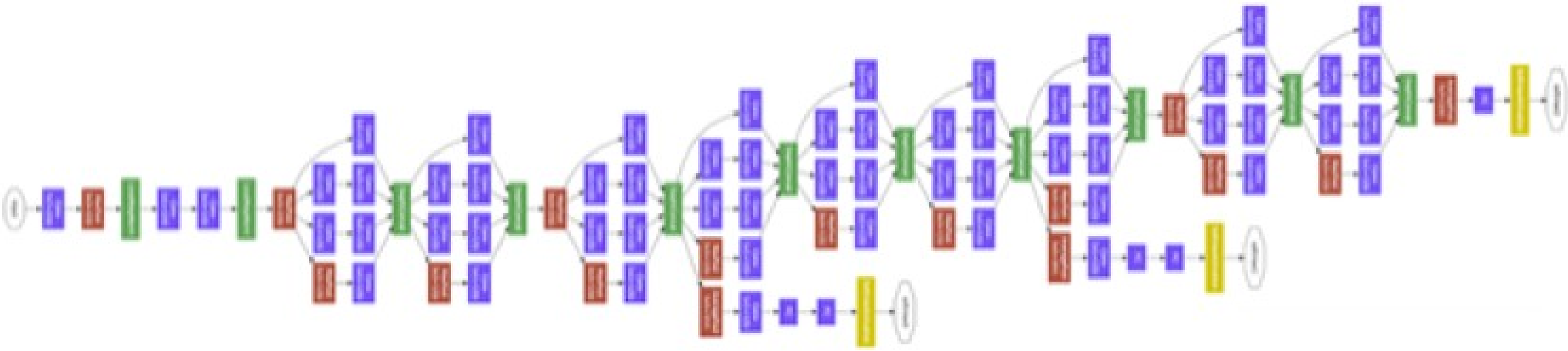
$$\text{sigmoid}: \frac{\partial L}{\partial \sigma(x)} \cdot [\sigma(x) \cdot (1 - \sigma(x))]$$

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

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

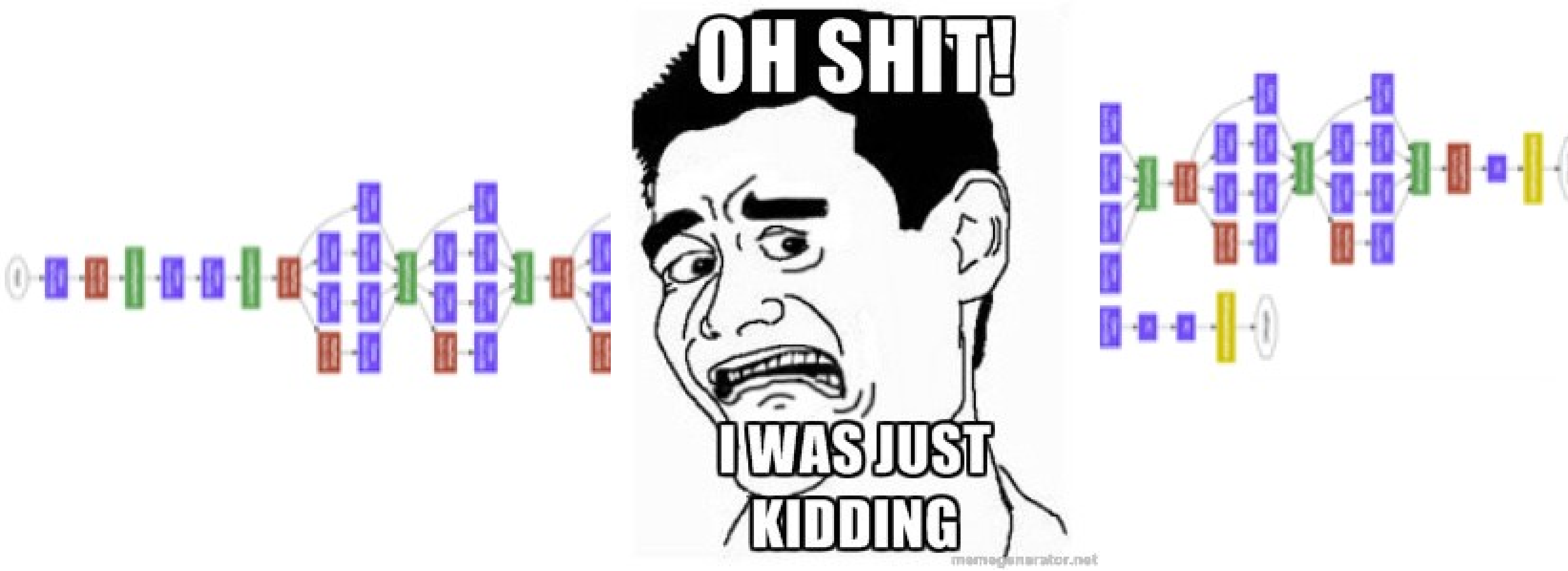
$$\text{linear over W}: \frac{1}{\|X\|} \cdot X^T \times \frac{\partial L}{\partial [X \times W + b]}$$

And now let's differentiate



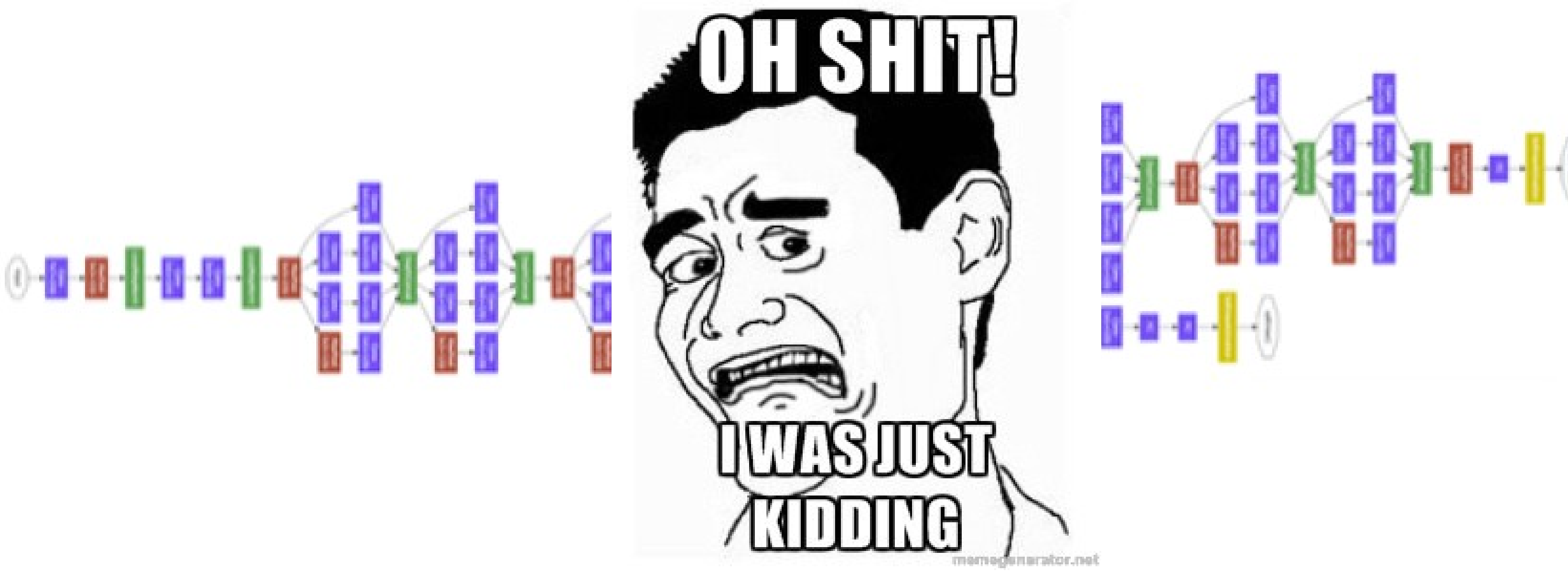
- 5+ types of layers
- each with different dimensions
- parallel branches with independent losses
- several nonlinearities

And now let's differentiate



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

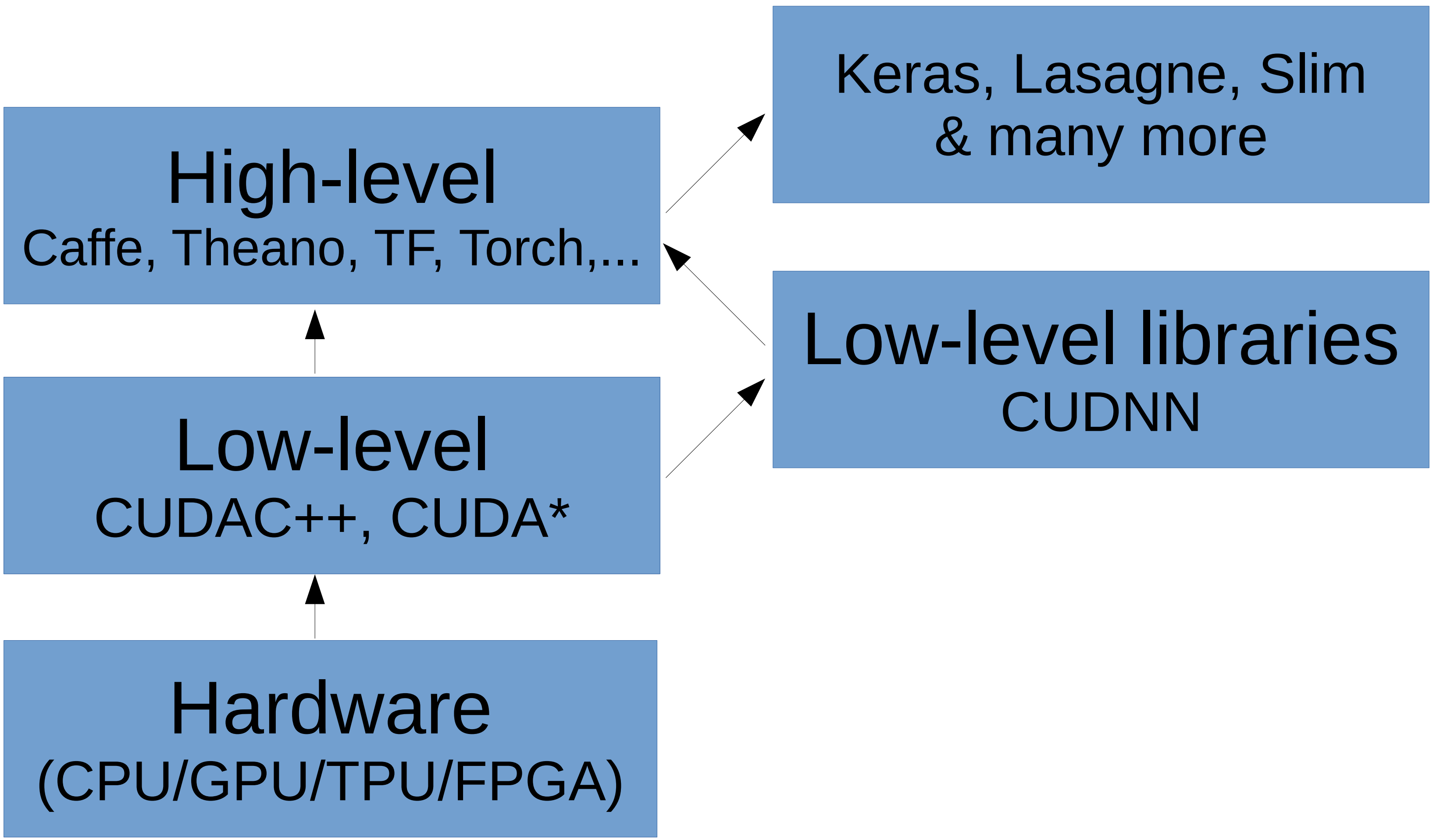
Deep learning frameworks



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

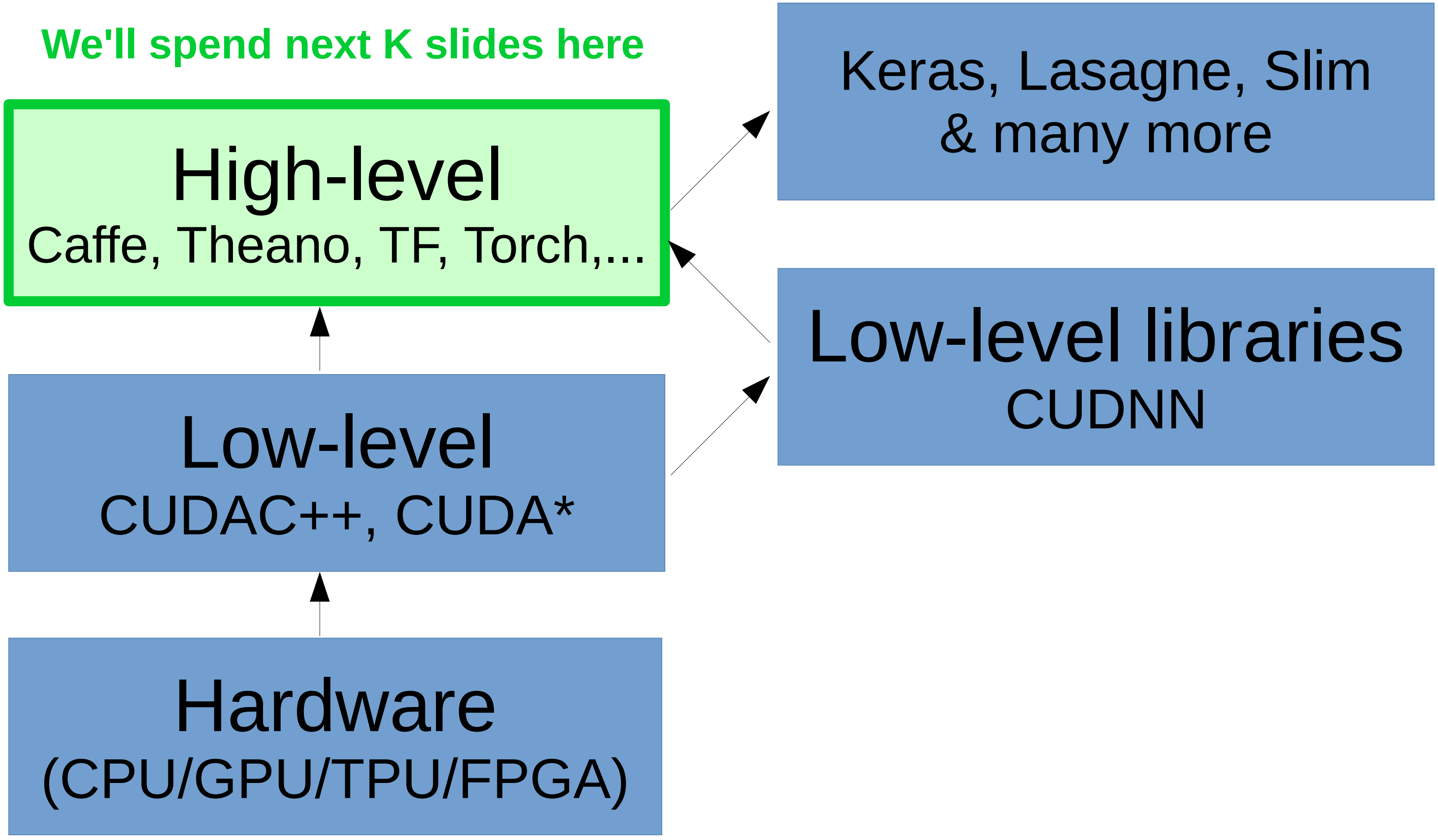
Deep learning frameworks

- 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:

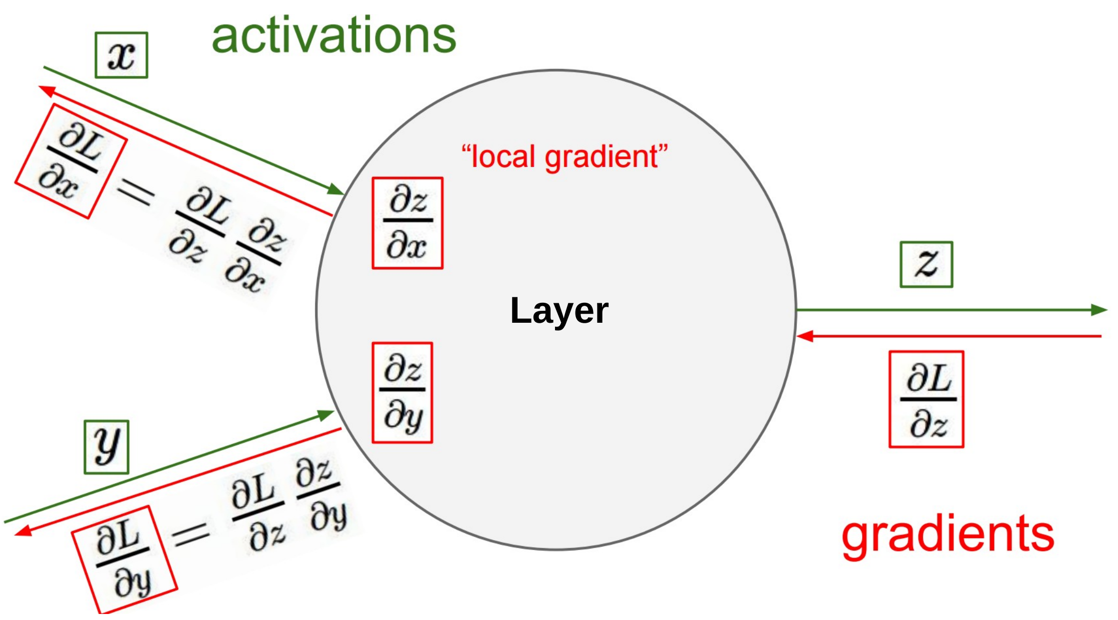
Same idea as in our hand-made neural net

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_solve
r.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

Still used in some legacy codebases

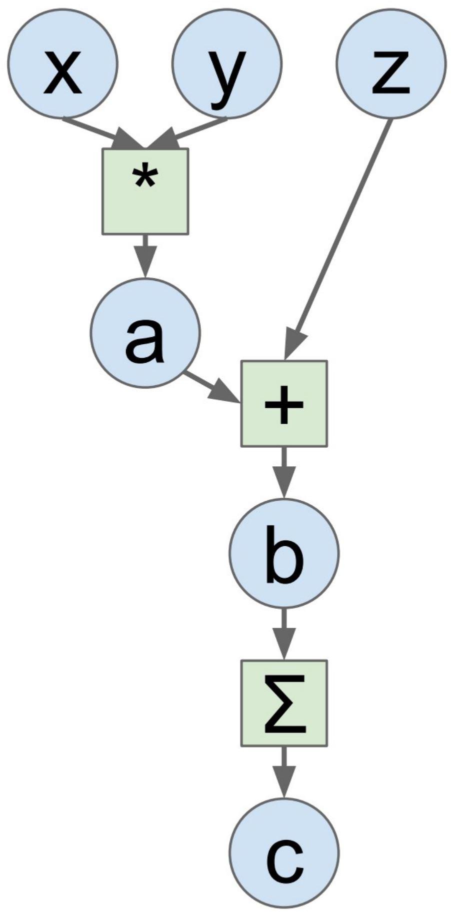
Symbolic graphs

What will your CPU do
when you write this?

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


This many further slides taken from Ars Ashukha @deepbayes2017

Symbolic graphs

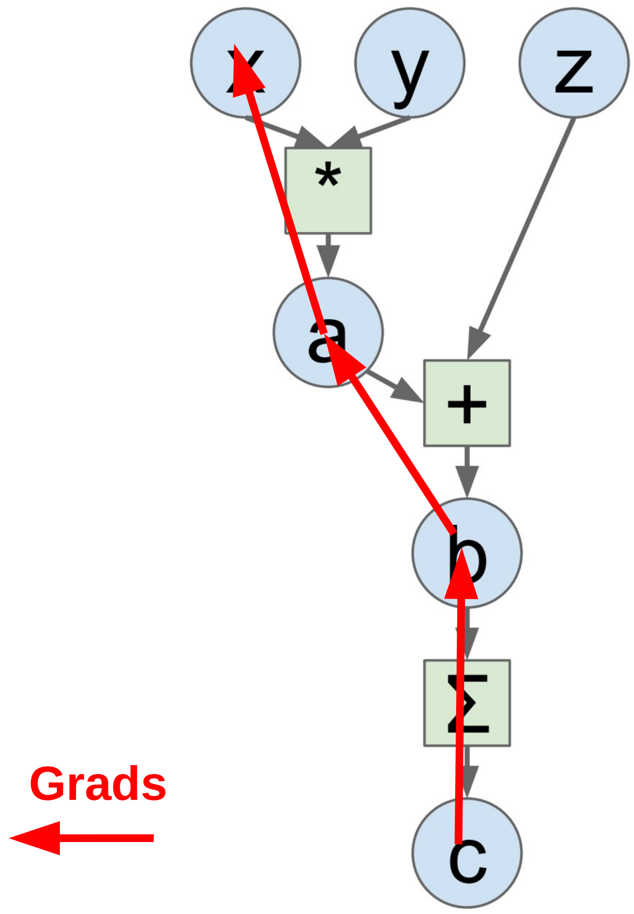



```
a = x * y
b = a + z
c = np.sum(b)
```

Idea: let's define
this graph explicitly!

This and some further slides taken from Ars Ashukha @deepbayes2017

Symbolic graphs




```
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

Symbolic graphs

Static graph frameworks
- Purely static is legacy

Theano (deprecated)
TensorFlow (before 2.0)

- Static in modern frameworks
 torch.jit.trace/script, compile
  jax.jit / tensorflow.function

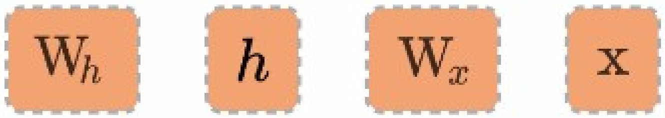
Dynamic graphs

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

+ 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

Andrej Karpathy
@karpathy

Following

I've been using PyTorch a few months now and I've never felt better. I have more energy. My skin is clearer. My eye sight has improved.

Researchers love them!

Advanced: GPU kernels w/o CUDA

https://openai.com/index/triton/

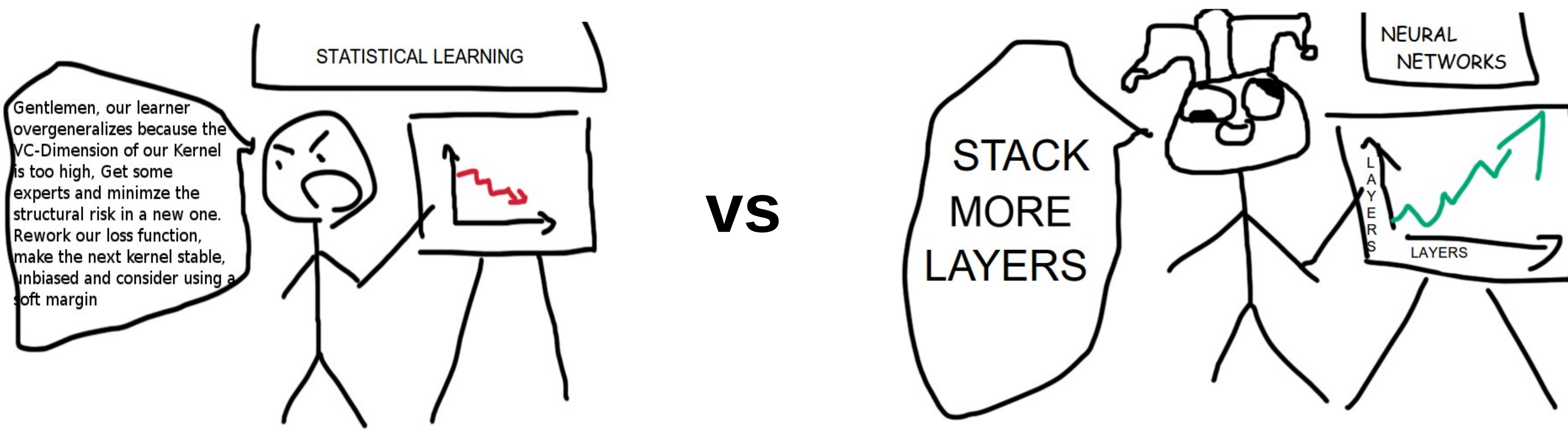
Python

```python
@triton.jit
def matmul(A, B, C, M, N, K, stride_am, stride_ak,
            stride_bk, stride_bn, stride_cm, stride_cn,
            **META):
    # extract metaparameters
    BLOCK_M, GROUP_M = META['BLOCK_M'], META['GROUP_M']
    BLOCK_N = META['BLOCK_N']
    BLOCK_K = META['BLOCK_K']
    # programs are grouped together to improve L2 hit rate
    _pid_m = tl.program_id(0)
    _pid_n = tl.program_id(1)
    pid_m = _pid_m // GROUP_M
    pid_n = (_pid_n * GROUP_M) + (_pid_m % GROUP_M)
    # rm (resp. rn) denotes a range of indices
    # for rows (resp. col) of C
    rm = pid_m * BLOCK_M + tl.arange(0, BLOCK_M)
    rn = pid_n * BLOCK_N + tl.arange(0, BLOCK_N)
    # rk denotes a range of indices for columns
    # (resp. rows) of A (resp. B)
    rk = tl.arange(0, BLOCK_K)
    # the memory addresses of elements in the first block of
    # A and B can be computed using numpy-style broadcasting
    A = A + (rm[:, None] * stride_am + rk[None, :] * stride_ak)
    B = B + (rk [:, None] * stride_bk  + rn[None, :] * stride_bn)
    # initialize and iteratively update accumulator
    acc = tl.zeros((BLOCK_M, BLOCK_N), dtype=tl.float32)
```

[Short break, then practice, then the rest of the class]

Not magic!

Don't expect deep learning to solve all your problems for free. For it won't.



https://i.warosu.org/data/sci/img/0073/62/1435656449422.png

Not magic

Book of grudges

- No core theory
    - Relies on intuitive reasoning

Not magic

Book of grudges

- No core theory
  - Relies on intuitive reasoning

- Needs tons of data
  - You need either large dataset or heavy wizardry

Not magic

Book of grudges

- No core theory
  - Relies on intuitive reasoning

- Needs tons of data
  - You need either large dataset or heavy wizardry

- Computationally heavy
  - Running on mobiles/embedded is a challenge

Not magic

Book of grudges

- No core theory
  - Relies on intuitive reasoning
- Needs tons of data
  - You need either large dataset or heavy wizardry
- Computationally heavy
  - Running on mobiles/embedded is a challenge
- Pathologically overhyped
  - People expect of it to make wonders

Deep learning is a language

Deep learning is a language

in which you can hint your model
on what you want it to learn

Deep learning is a language

Say, you train classifier on two sets of features

Deep learning is a language

Say, you train classifier on two sets of features

Raw features

Car photo (image pixels)

High-level features

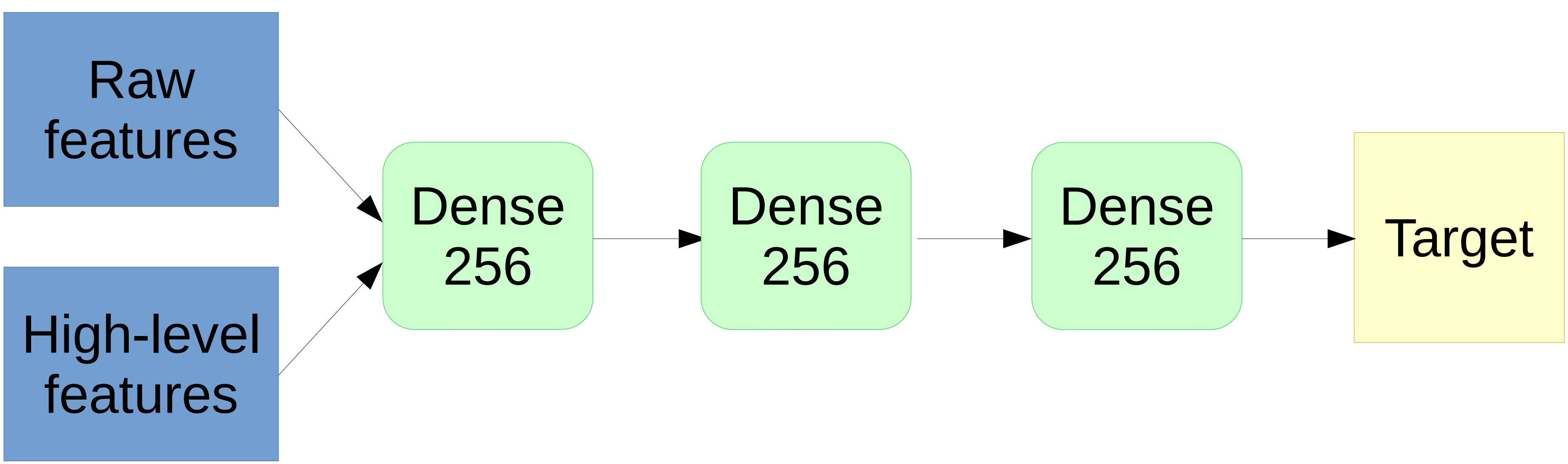
Car brand, model, age, blemishes

Car price

Target

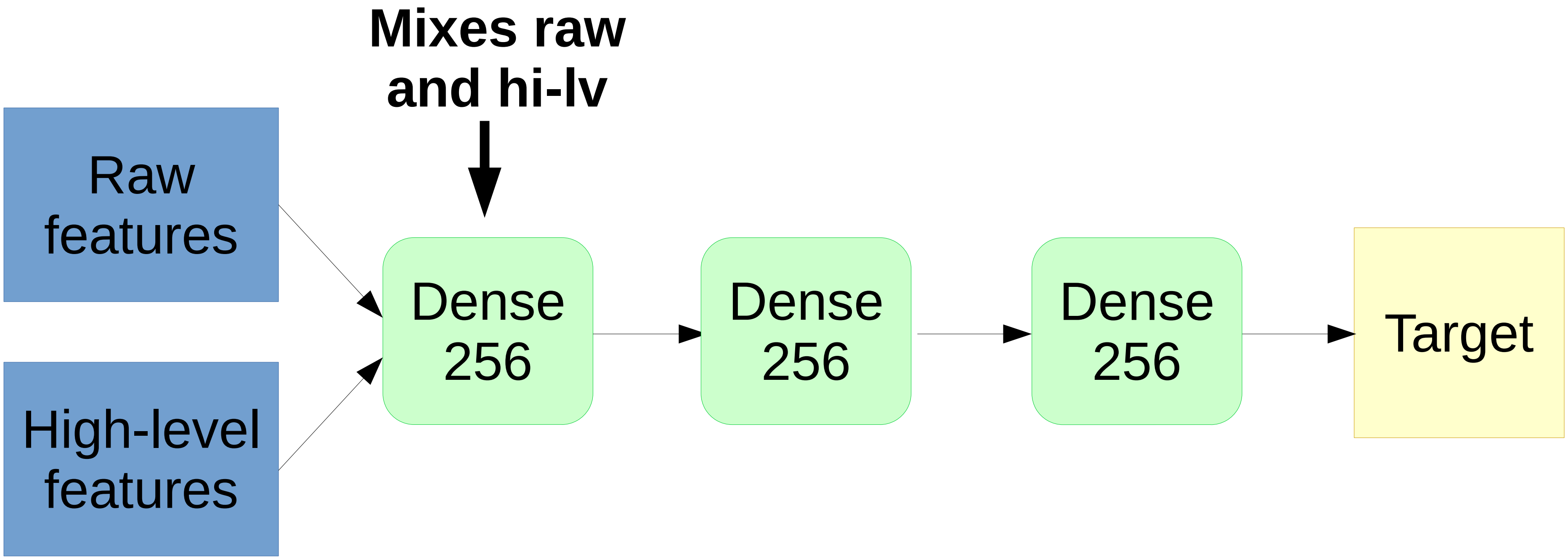
Deep learning is a language

Naive approach



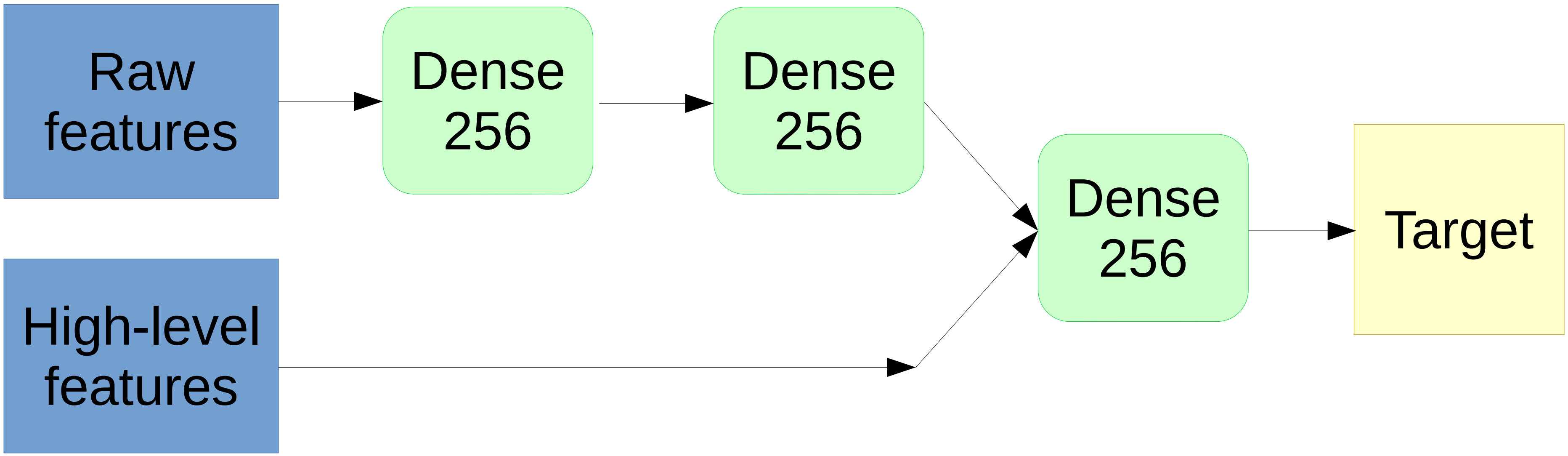
Deep learning is a language

Naive approach



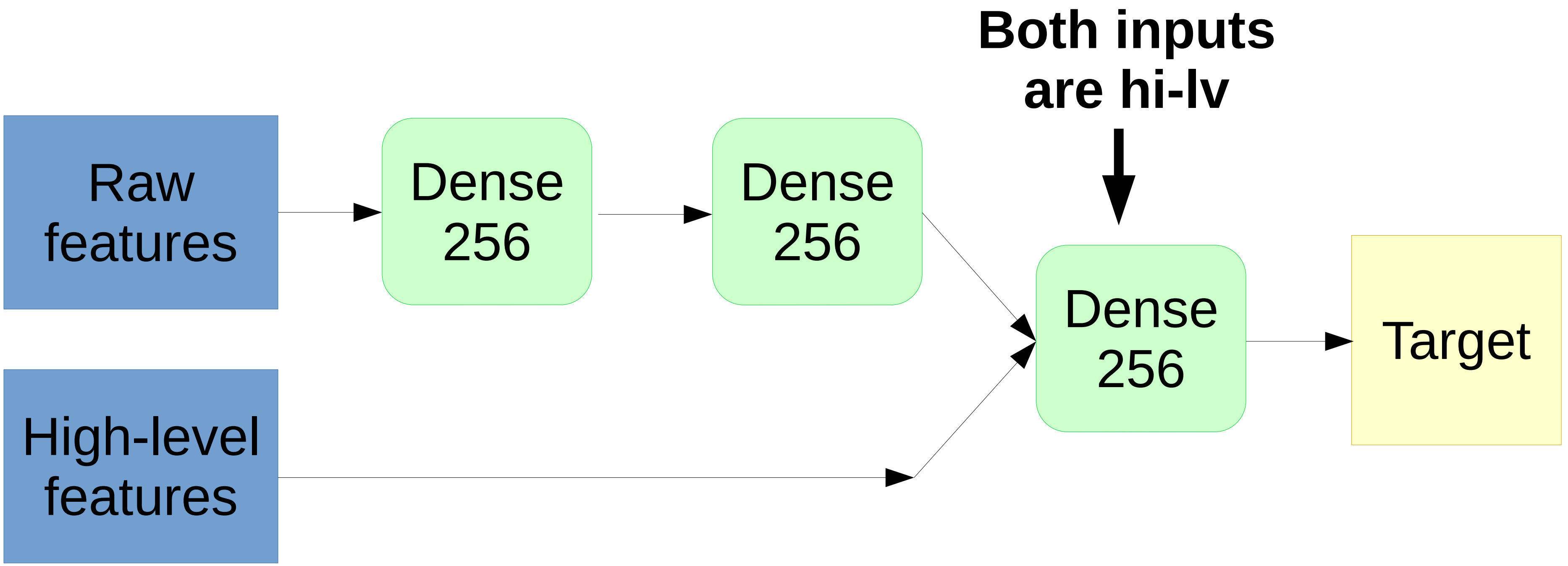
Deep learning is a language

Less naïve approach



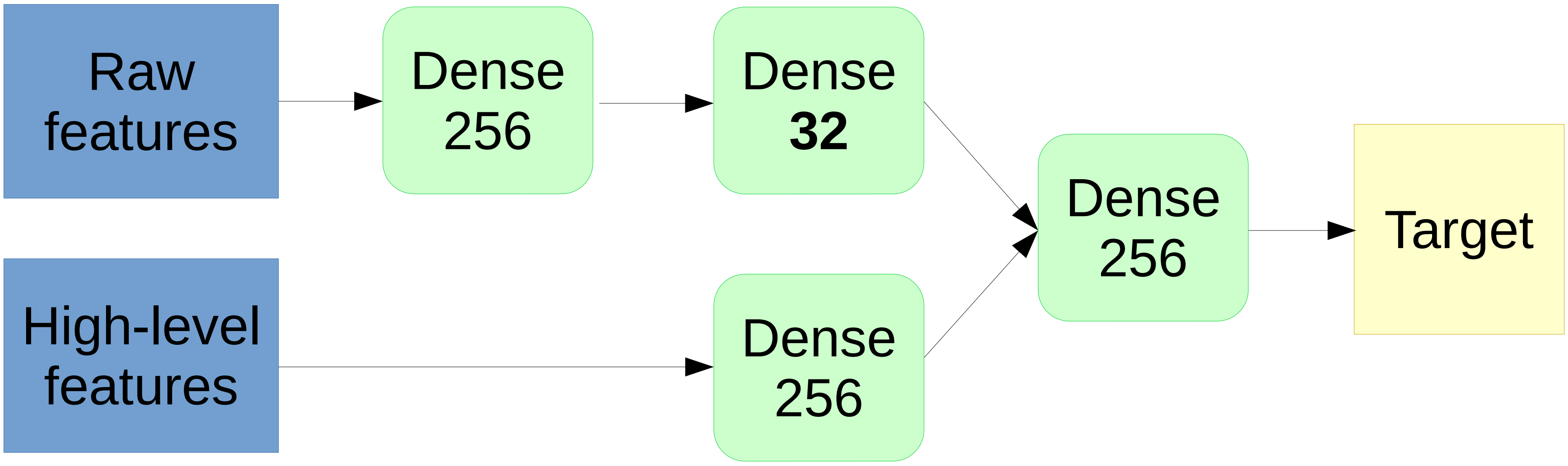
Deep learning is a language

Less naïve approach



Deep learning is a language

“Image features should be less important”
if that's what you want to say

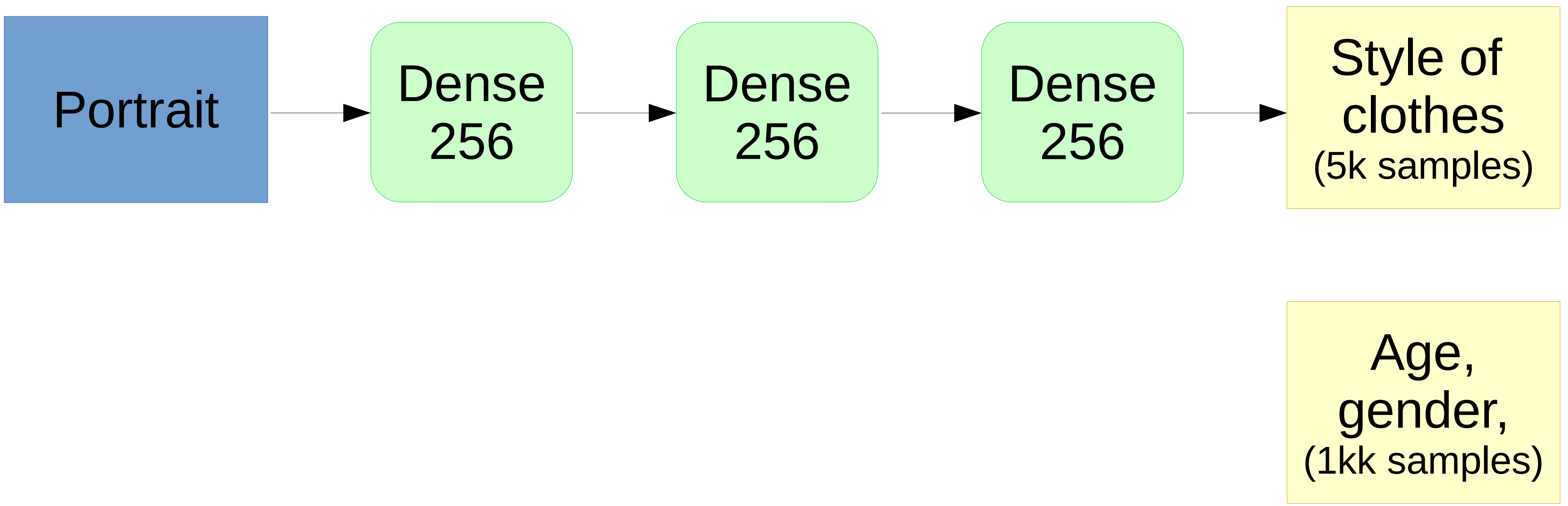


Deep learning is a language

You have a small dataset

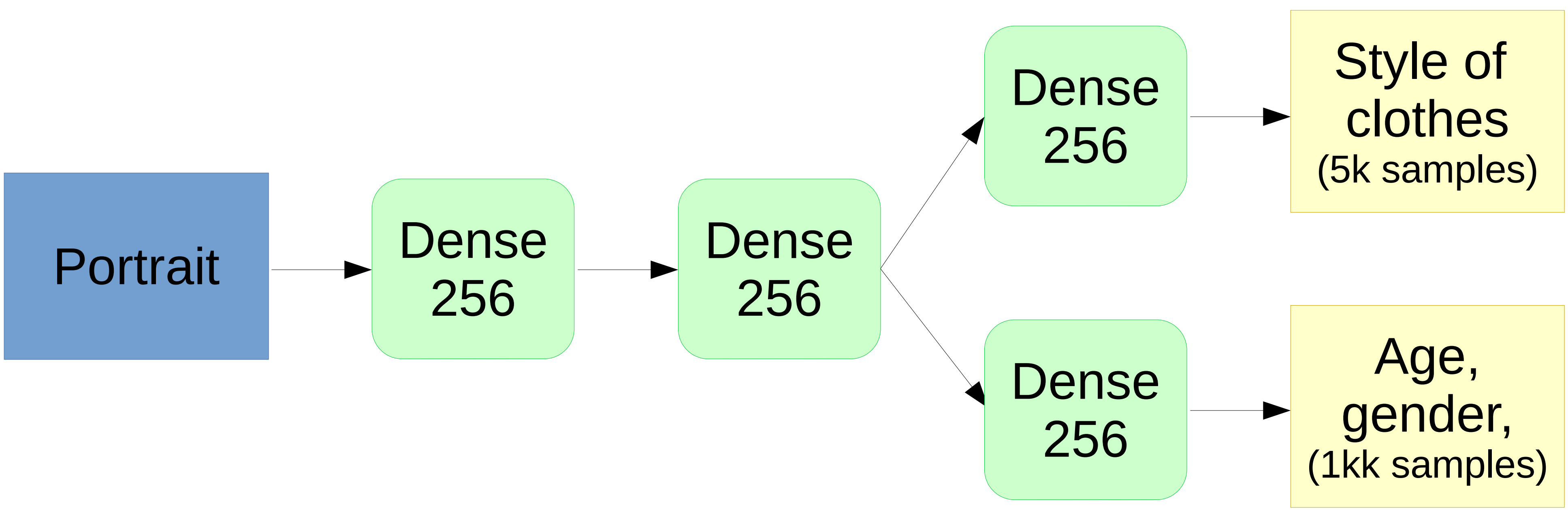
Deep learning is a language

You have a small dataset
and a larger dataset with similar task



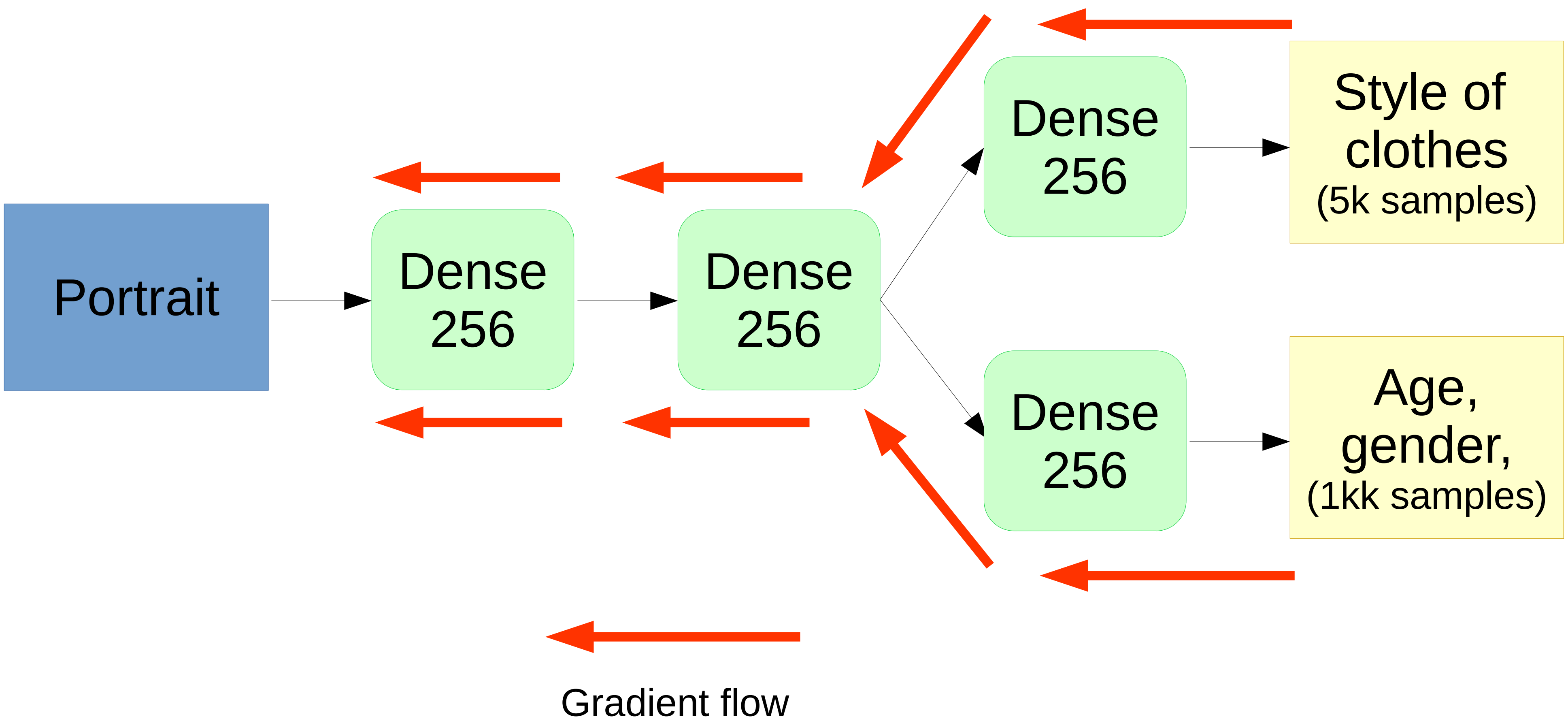
Deep learning is a language

You have a small dataset
and a larger dataset with similar task



Deep learning is a language

I want to learn features for style classification that also help determine age & gender



Deep learning is a language

For images:

- “I want to classify cats regardless where they are”
- “A cat shifted by 3 pixels is still a cat”

For texts:

- “People read and write texts left to right”

In general:

- “I don't want model to trust single feature too much”
- “I want my features to be sparse”

Let's see a few more “words”

Regularization

- Neural networks overfit like nothing else.

Gotta regularize!

- We can use L1/L2 like usual, but there's more!

Regularization

- Dropout:

“I don't my network to trust
any single neuron too much”

- Idea:

At training time, with probability **p**

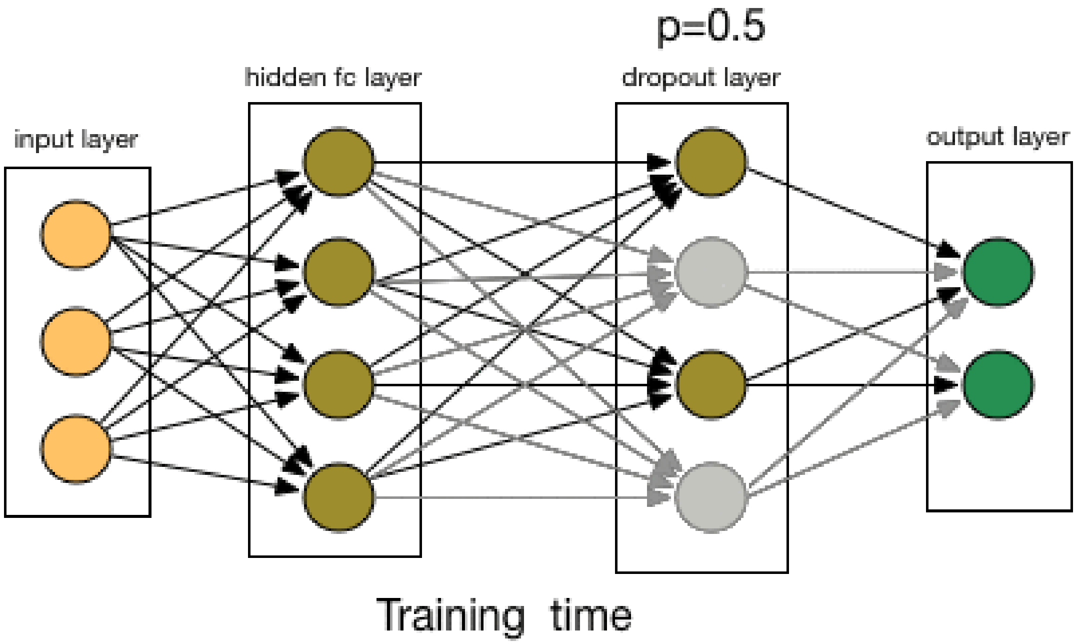
multiply neurons by zero!

- Scale up the remaining neurons to keep average the same

Regularization

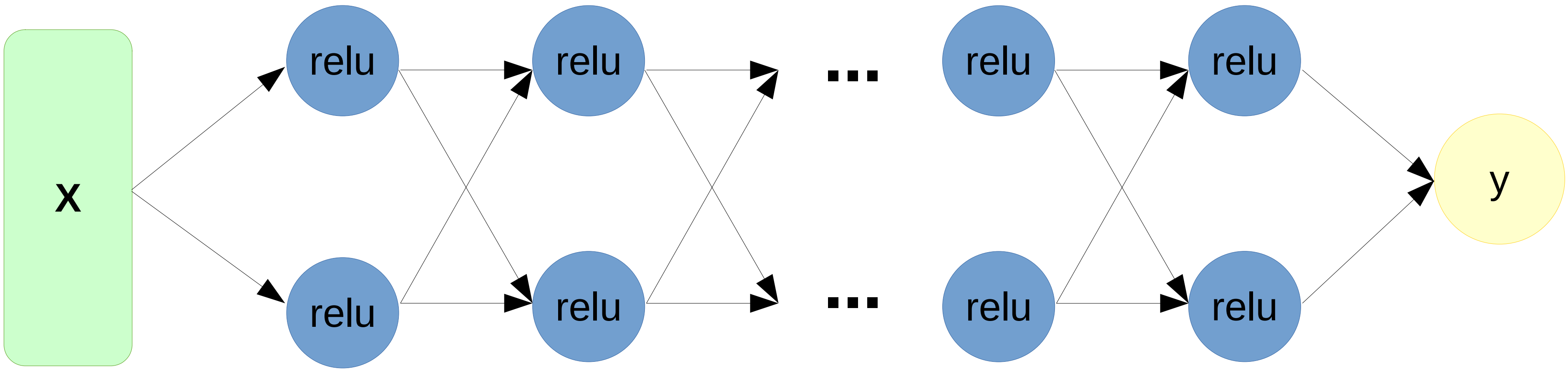
- Dropout:

"I don't my network to trust any single neuron too much"



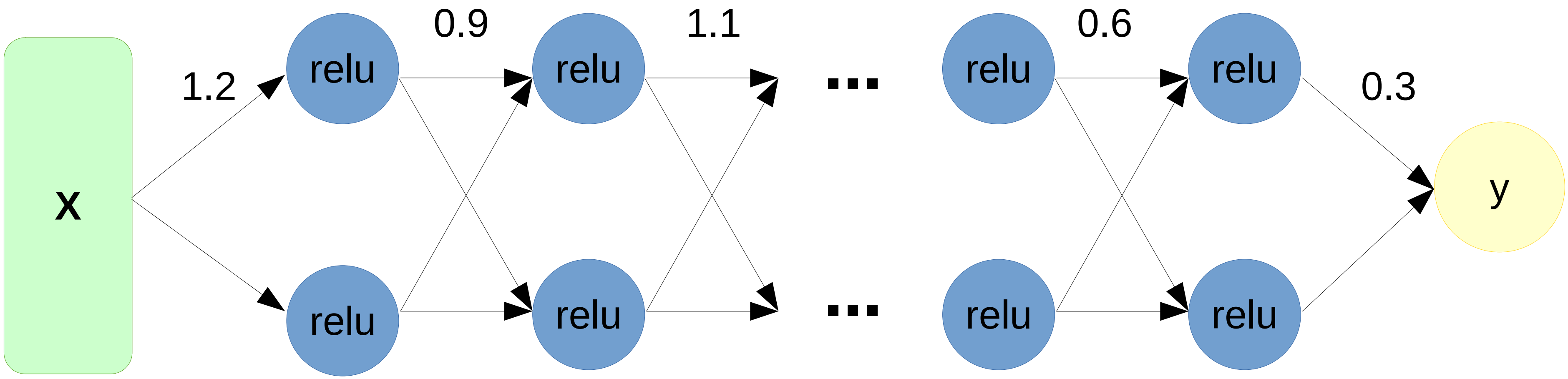
The problem with deep networks

- Imagine a 100-layer network with ReLU



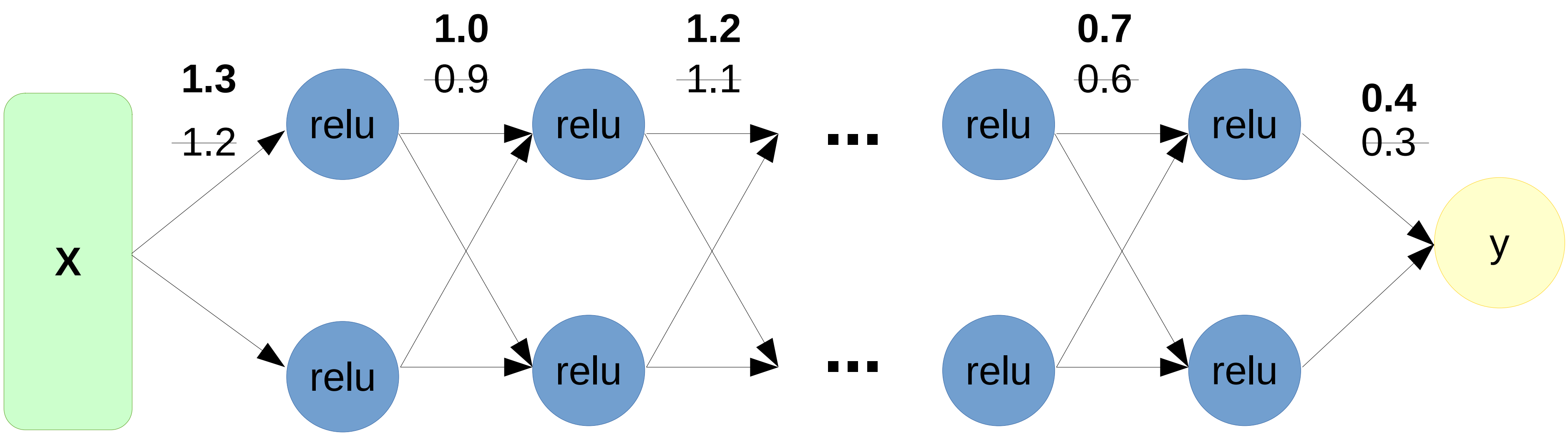
The problem with deep networks

- Imagine a 100-layer network with ReLU



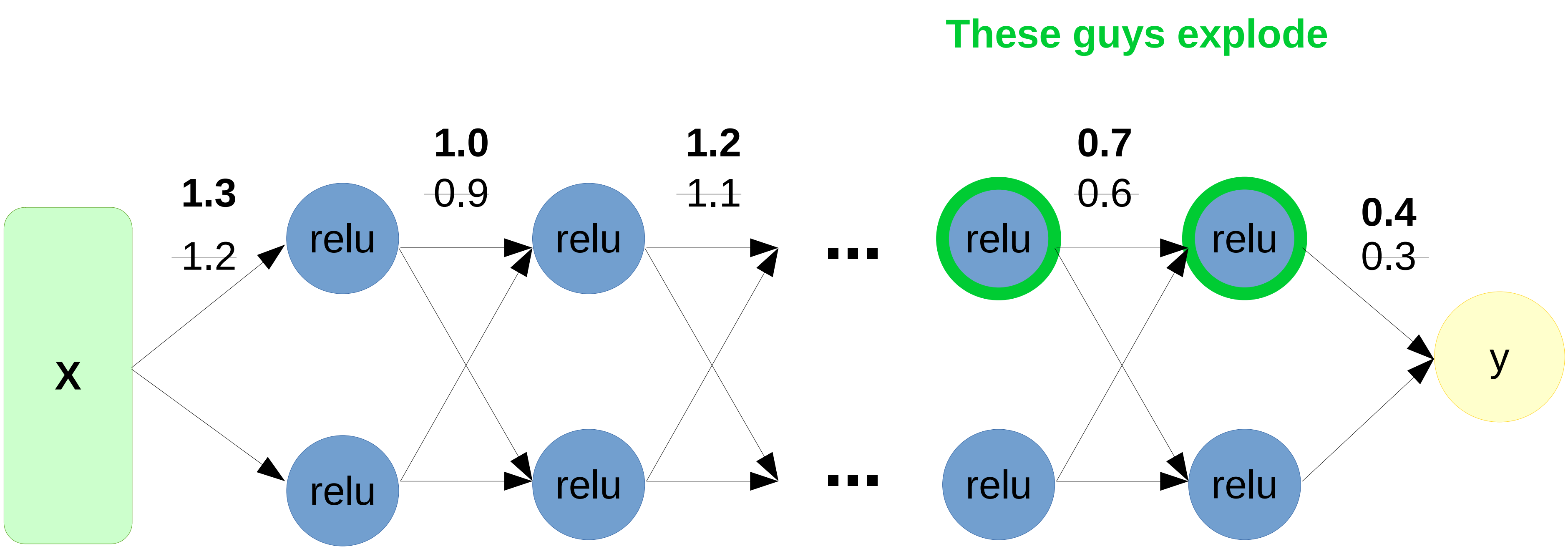
The problem with deep networks

- Imagine a 100-layer network with ReLU
- Single gradient step...



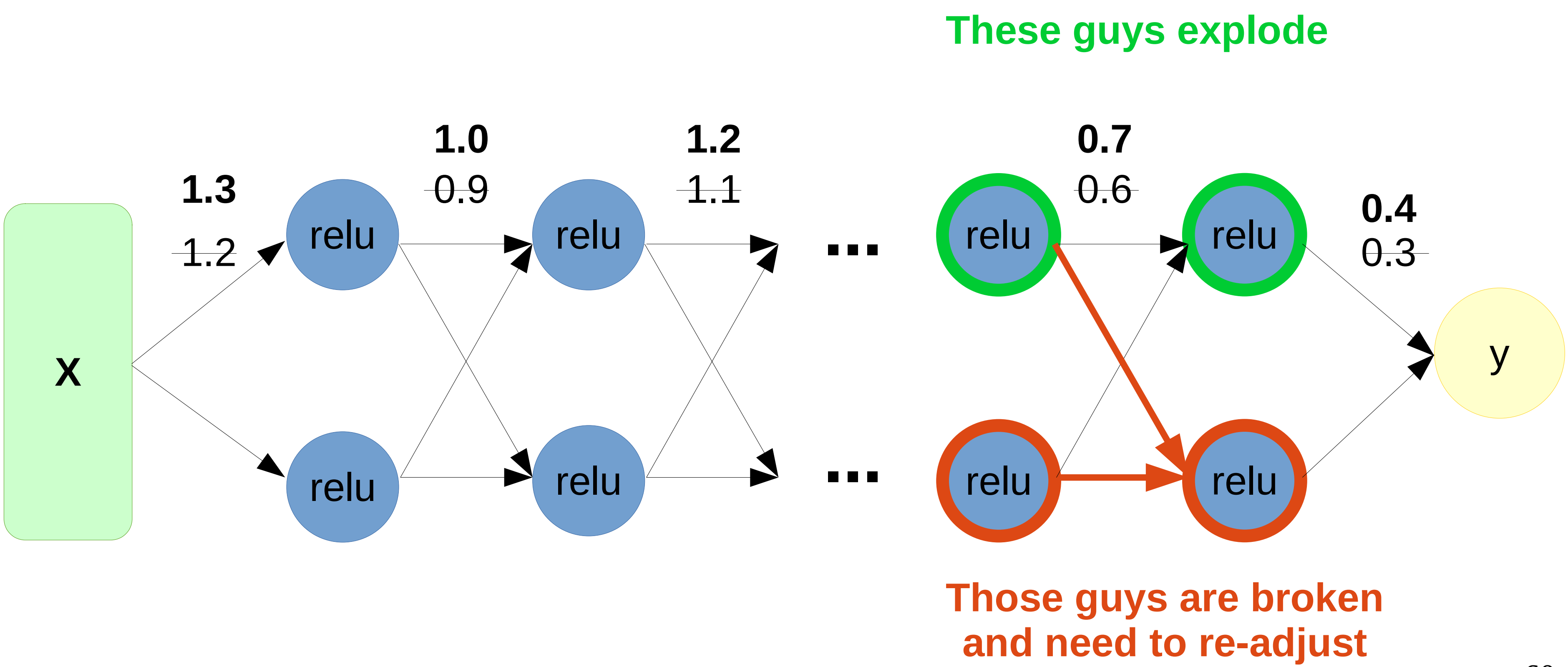
The problem with deep networks

- Imagine a 100-layer network with ReLU
- Single gradient step...



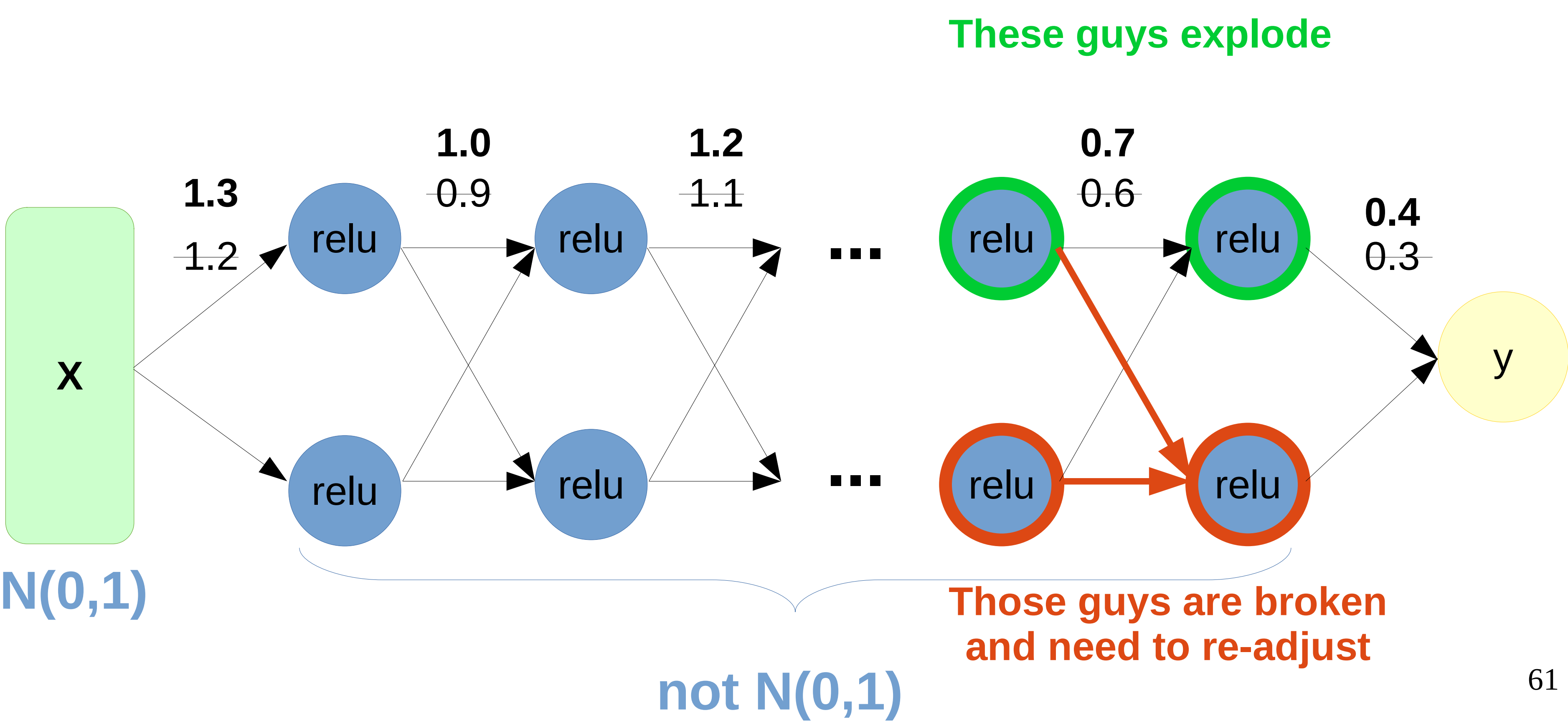
The problem with deep networks

- Imagine a 100-layer network with ReLU
- Single gradient step...



The problem with deep networks

- Imagine a 100-layer network with ReLU
- Single gradient step...



Batch normalization

TL;DR:

- It's usually a good idea to normalize linear model inputs

(c) Every machine learning lecturer, ever

Batch normalization

Idea:

- We normalize activation of a hidden layer

  (zero mean unit variance)

$$h_i = \frac{h_i - \mu_i}{\sqrt{\sigma_i^2}}$$

- Update μ_i, σ_i^2 with moving average while training

$$\mu_i := \alpha \cdot mean_{batch} + (1-\alpha) \cdot \mu_i$$

$$\sigma_i^2 := \alpha \cdot variance_{batch} + (1-\alpha) \cdot \sigma_i^2$$

Batch normalization

Idea:

- We normalize activation of a hidden layer

  (zero mean unit variance)

$$h_i = \frac{h_i - \mu_i}{\sqrt{\sigma_i^2}}$$

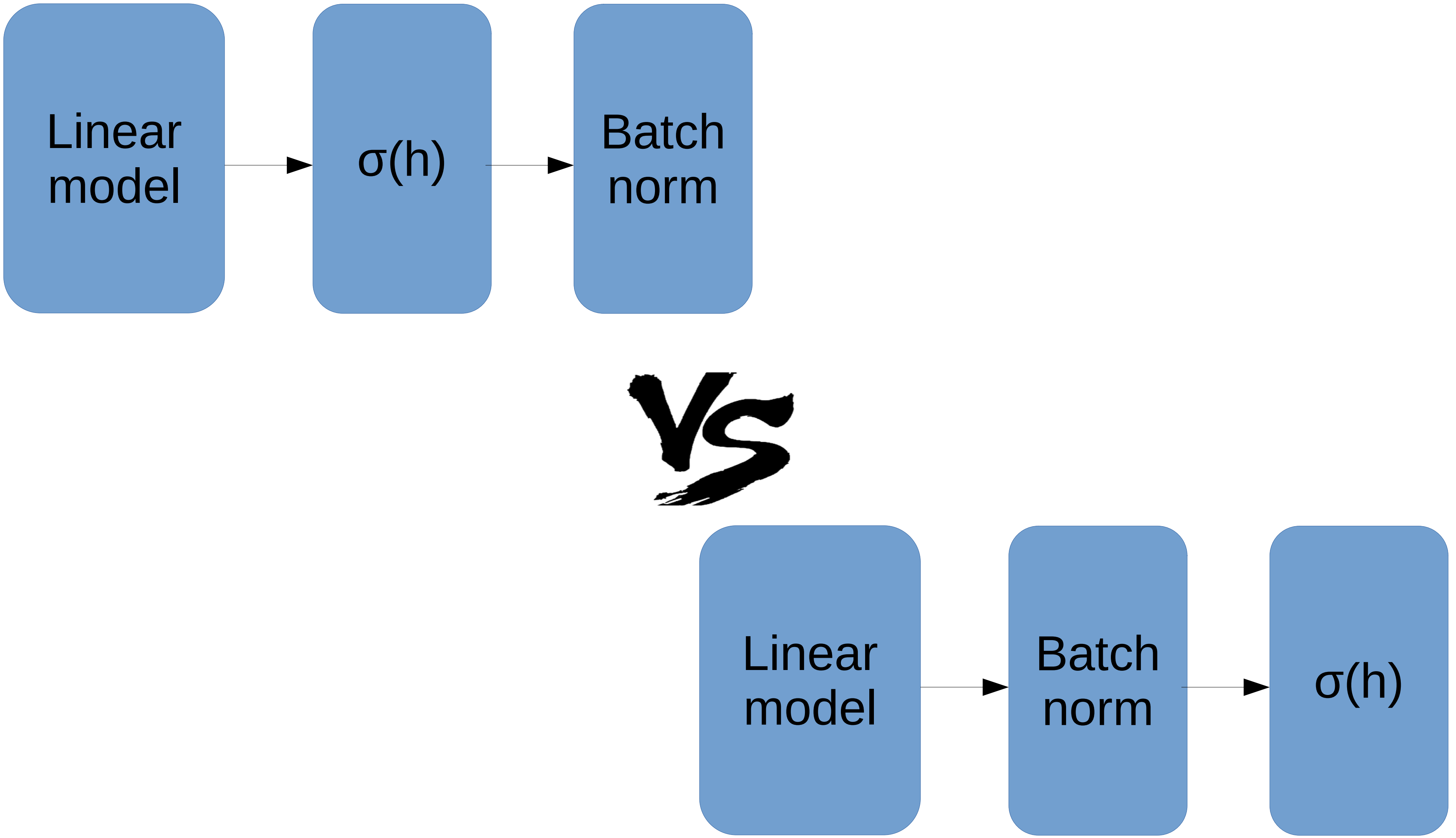
i stands for i-th neuron

- Update μ_i, σ_i^2 with moving average while training

$$\mu_i := \alpha \cdot mean_{batch} + (1 - \alpha) \cdot \mu_i$$

$$\sigma_i^2 := \alpha \cdot variance_{batch} + (1 - \alpha) \cdot \sigma_i^2$$

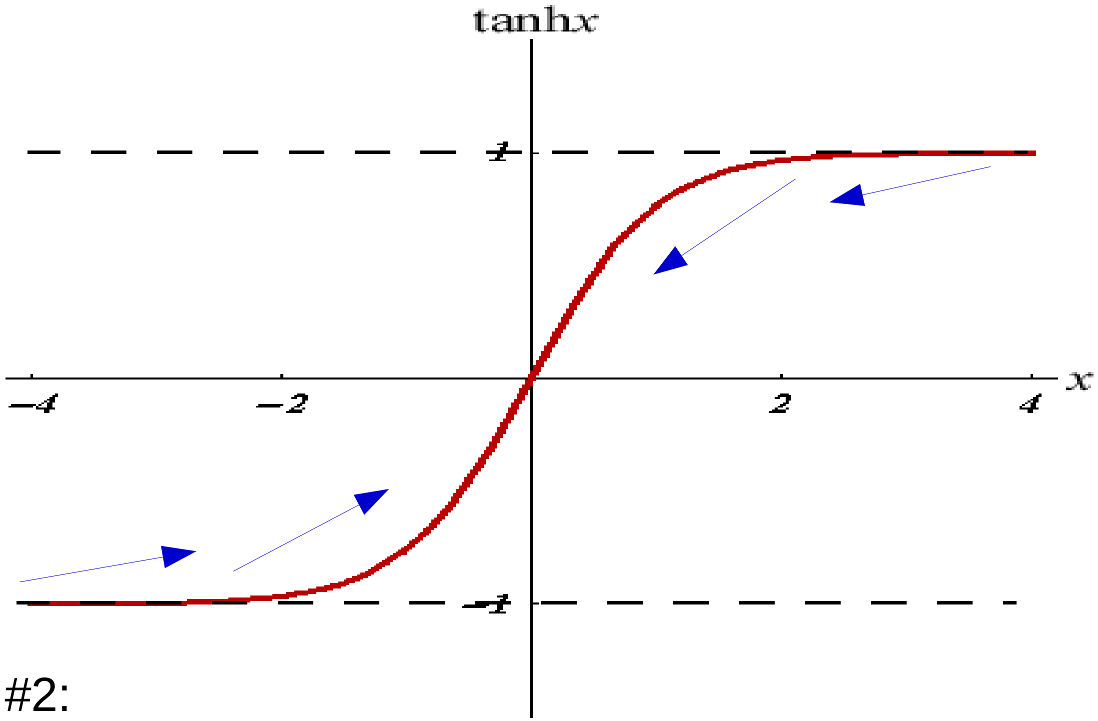
Batch normalization



Batch normalization

Good side effect #1:

- Vanishing gradient less a problem for sigmoid-like nonlinearities



Good side effect #2:

- We no longer need to train bias (+b term in Wx+b)

Weight normalization

Same problem, different solution

- Learn separate “direction” w and “length” l

$$\hat{w} \stackrel{\text{def}}{=} \frac{w}{\|w\|} \cdot l$$

- Much simpler, but requires good init

More normalization

Layer/Instance normalization

– Like batchnorm, but normalizes over different axes

Normprop

– A special training algorithm

Self-normalizing neural networks (SELU)

More normalization

Layer/Instance normalization

- Like batchnorm, but normalizes over different axes

Normprop

- A special training algorithm

Self-normalizing neural networks (SELU)

Architectures: residual network

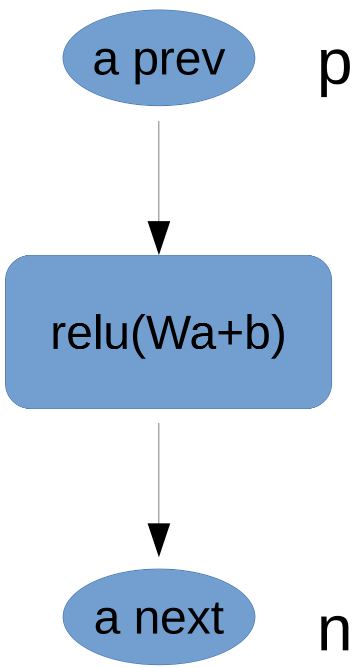
Problem: very deep networks are hard to train

Gradients w.r.t. first layers are volative

Architectures: residual network

Idea: let's create a shortcut for gradients

Normal layer



previous layer activation

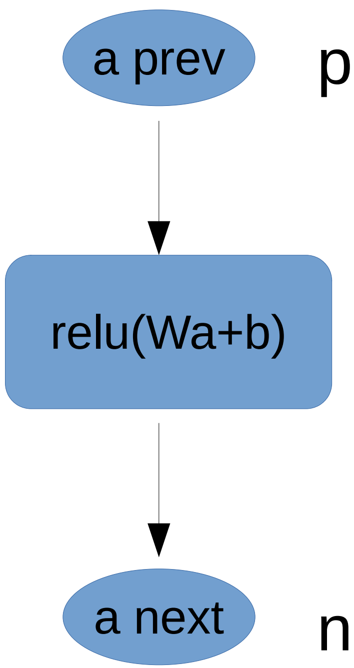
next layer activation

$$f_{w,b}(x) = relu(W \cdot a + b)$$

Architectures: residual network

Idea: let's create a shortcut for gradients

Normal layer



previous layer activation

next layer activation

$$f_{w,b}(x)=relu(W\cdot a+b)$$

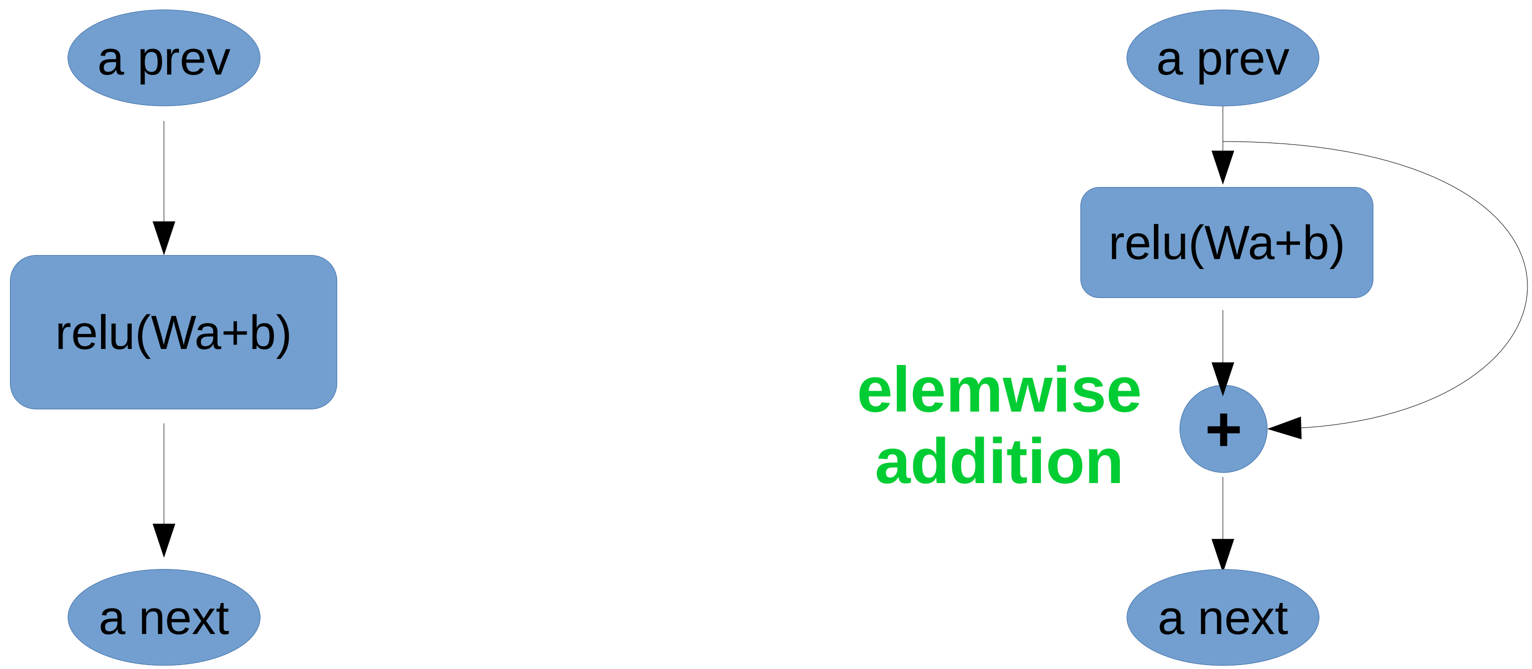
$$\nabla f_{w,b}(x)=\underline{\nabla relu(W\cdot a+b)}$$

Gradients can vanish if relu < 0

Architectures: residual network

Idea: let's create a shortcut for gradients

Normal layer | Residual layer



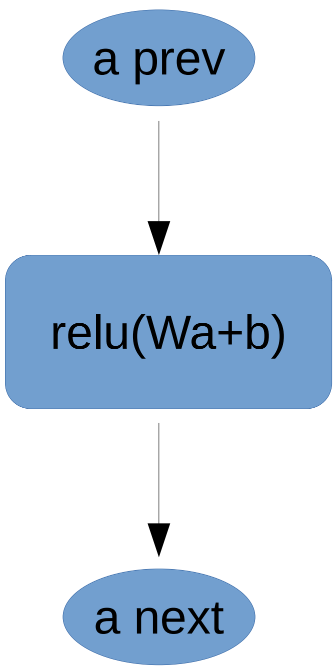
$$f_{w,b}(x) = relu(W \cdot a + b)$$

$$f_{w,b}(x) = relu(W \cdot a + b) + X$$

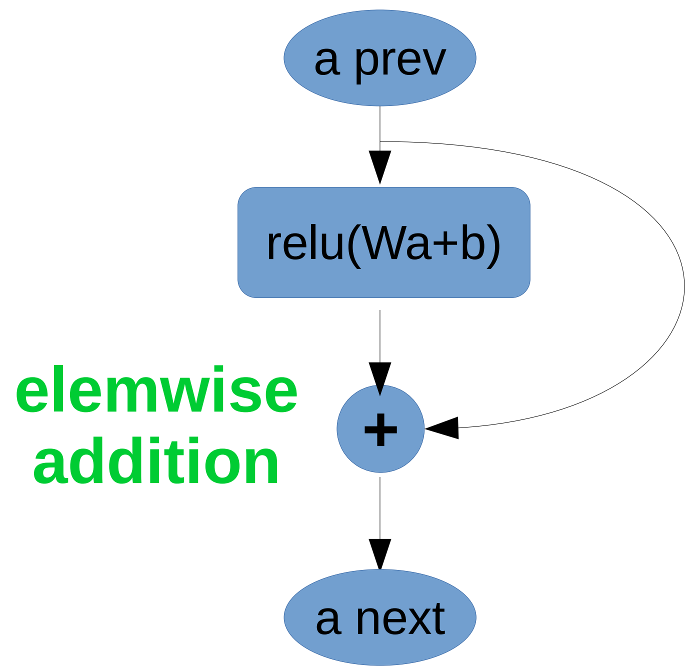
Architectures: residual network

Idea: let's create a shortcut for gradients

Normal layer



Residual layer



$$f_{w,b}(x) = relu(W \cdot a + b)$$

$$\nabla f_{w,b}(x) = \nabla relu(W \cdot a + b)$$

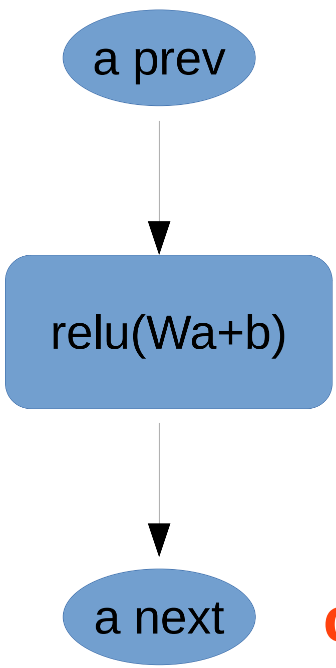
$$f_{w,b}(x) = relu(W \cdot a + b) + X$$

???

Architectures: residual network

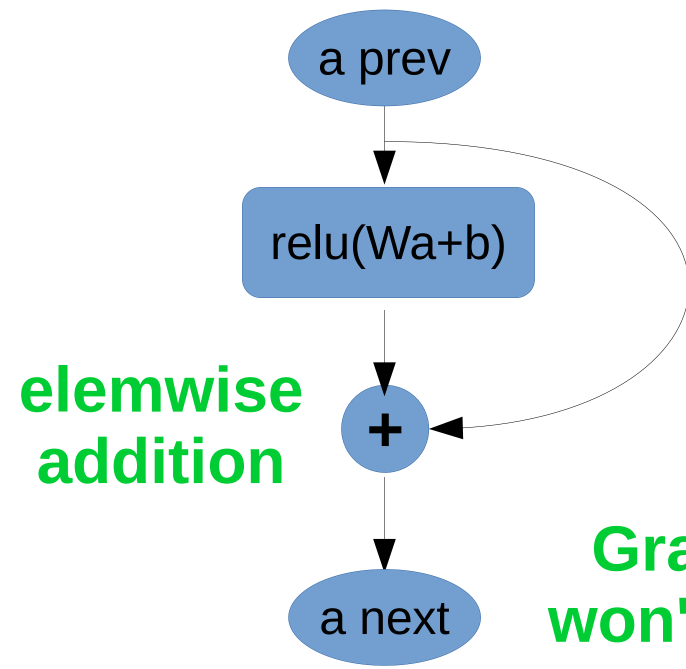
Idea: let's create a shortcut for gradients

Normal layer



Gradients can vanish

Residual layer



Gradients won't vanish

$$f_{w,b}(x) = relu(W \cdot a + b)$$

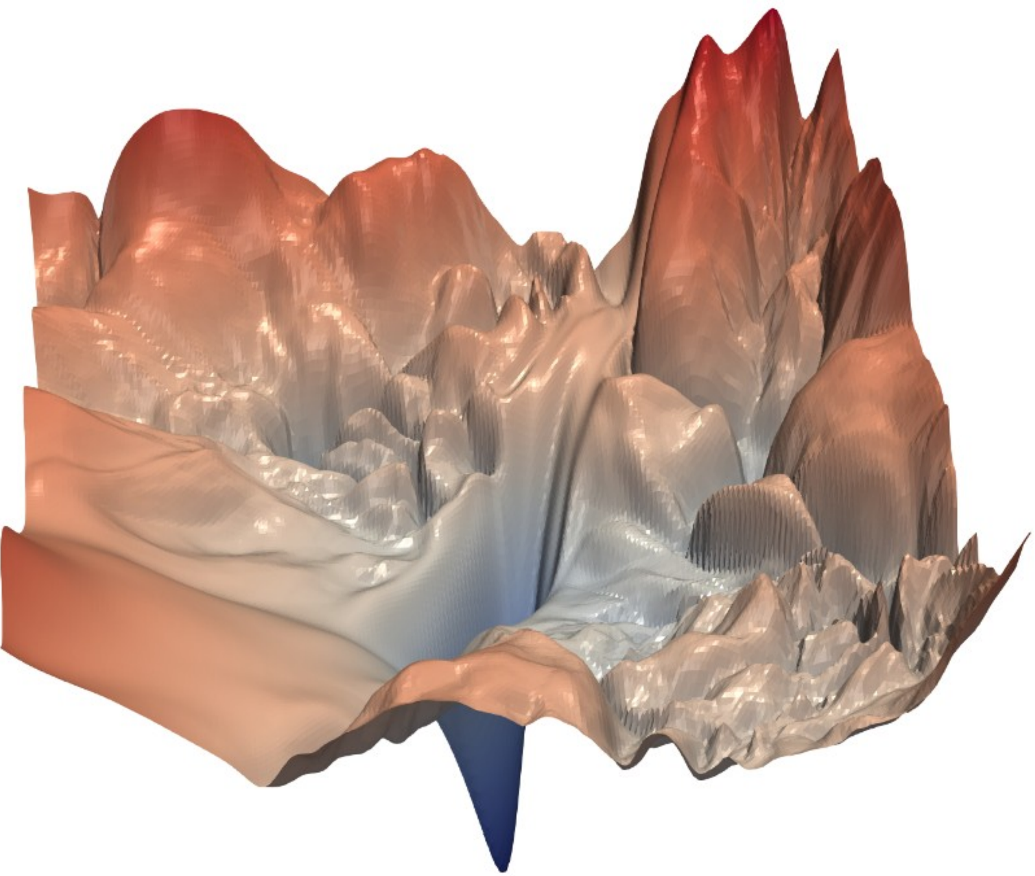
$$\nabla f_{w,b}(x) = \underline{\nabla relu(W \cdot a + b)}$$

$$f_{w,b}(x) = relu(W \cdot a + b) + X$$

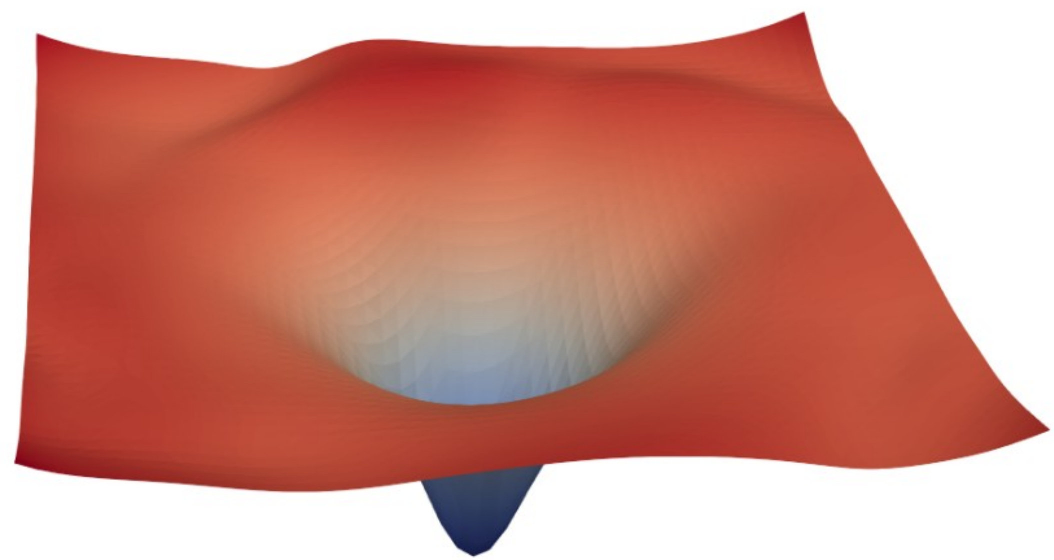
$$\nabla f_{w,b}(x) = \underline{\nabla relu(W \cdot a + b)} + \underline{\vec{1}}$$

Loss Surfaces

Idea: https://arxiv.org/abs/1712.09913



(a) without skip connections



(b) with skip connections

TL;DR today

Didn’t like writing backward code?

TL;DR today

Didn’t like writing backward code?

Autograd (pytorch, jax, tf, etc)

Multiple types of input data?

e.g. image + tabular

TL;DR today

Didn’t like writing backward code?

Autograd (pytorch, jax, tf, etc)

Multiple types of input data?
e.g. image + tabular

Concat / add branches

Model overfits too quickly?

TL;DR today

Didn’t like writing backward code?

Autograd (pytorch, jax, tf, etc)

Multiple types of input data?
e.g. image + tabular

Concat / add branches

Model overfits too quickly?

Dropout or similar

Many layers hard to train?

TL;DR today

Didn’t like writing backward code?

Autograd (pytorch, jax, tf, etc)

Multiple types of input data?
e.g. image + tabular

Concat / add branches

Model overfits too quickly?

Dropout or similar

Many layers hard to train?

BatchNorm, Residual connections

Nuff

Coding time!