Deep learning
Episode 2, 2019

Deep learning whereabouts
A catch-all lecture in philosophy, tricks and frameworks

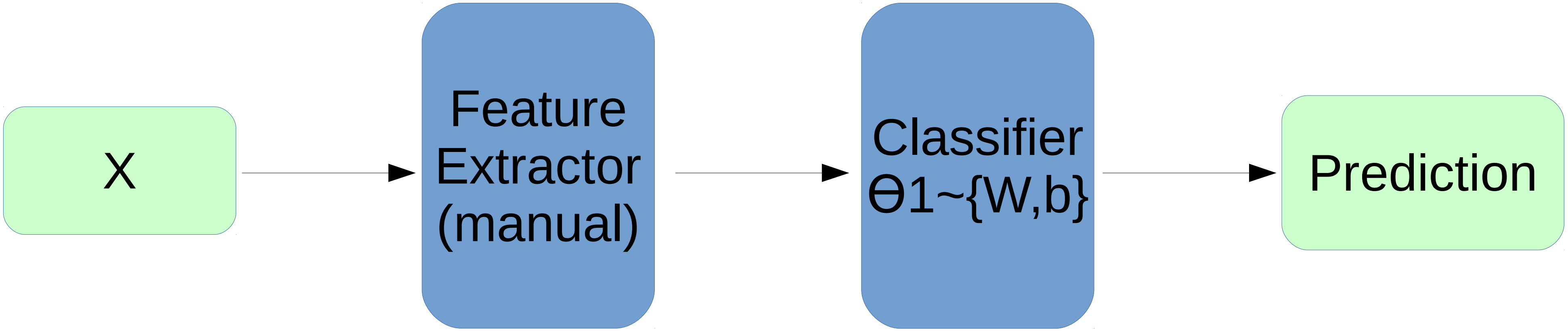
Yandex
Data Factory

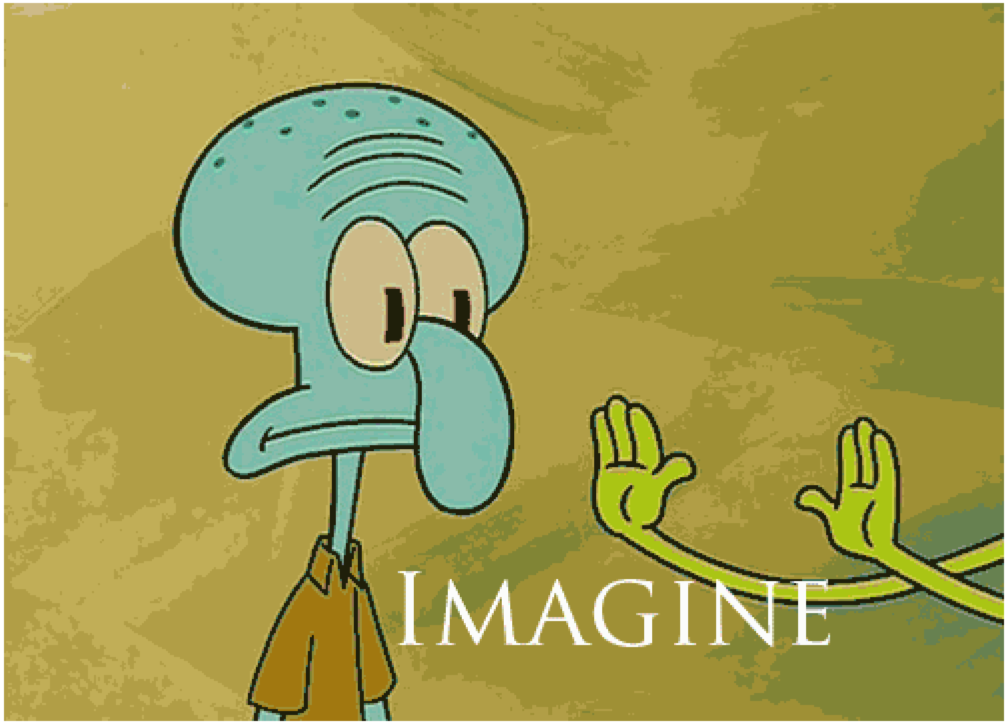
LAMBDA

British Hedgehog
Preservation Society

Previously on deep learning...

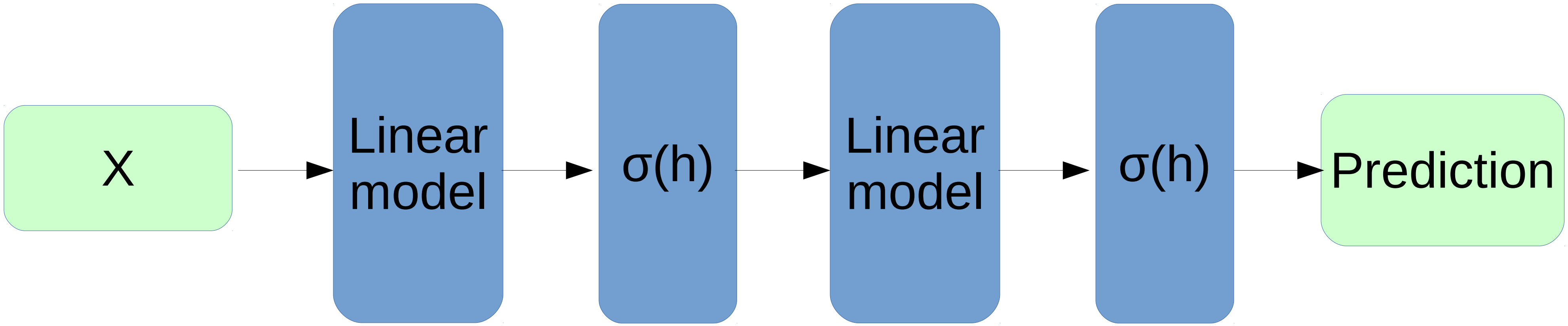
Feature extraction





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*

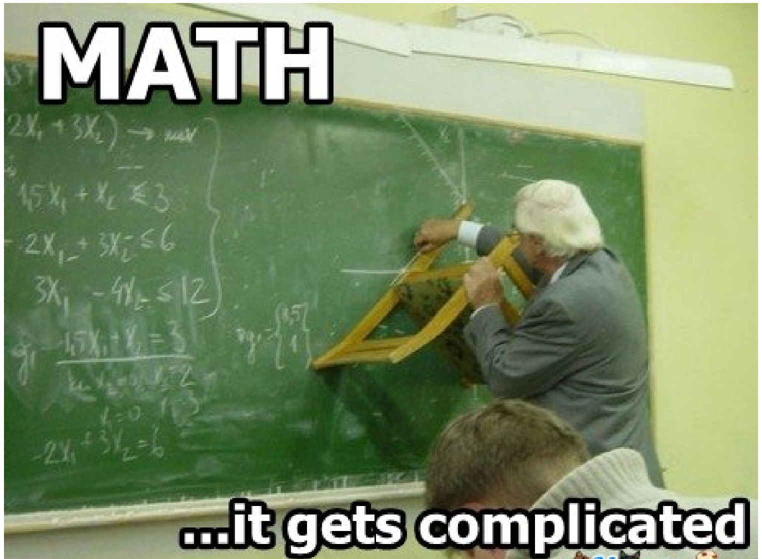
$$\frac{\partial f(g(x))}{\partial x} = \frac{\partial f(g(x))}{\partial g(x)} \cdot \frac{\partial g(x)}{\partial x}$$

Backpropagation

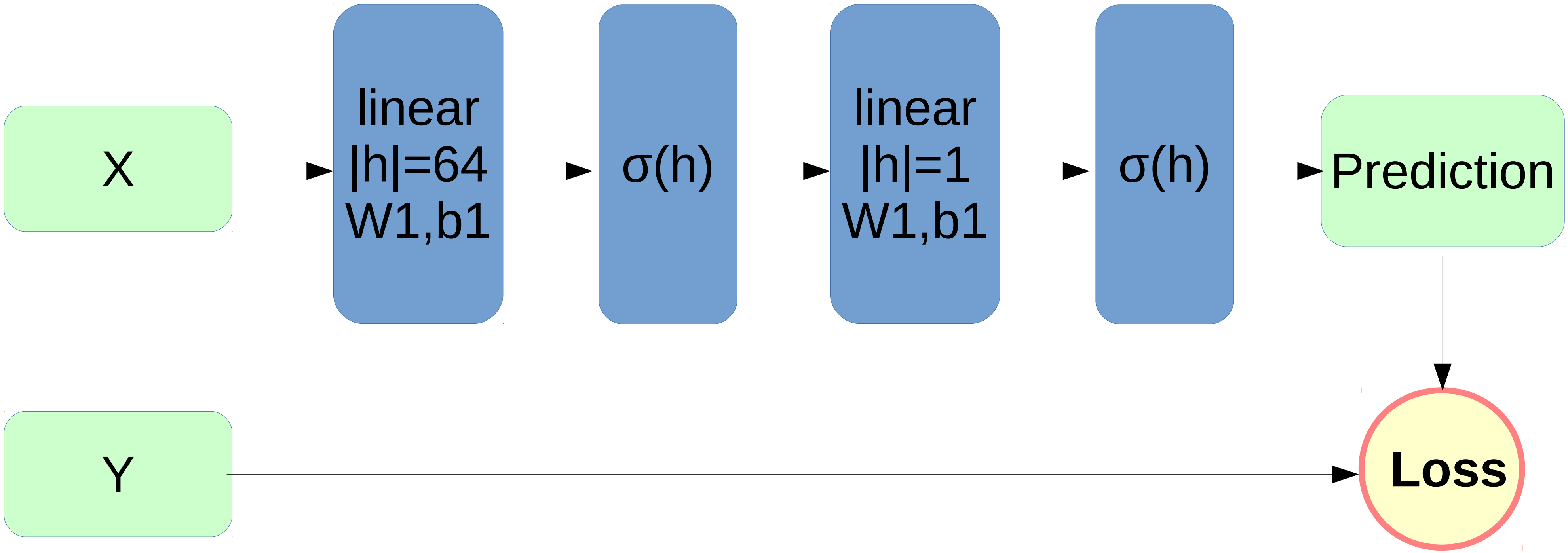
TL;DR: backprop = chain rule*

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

* g and x can be vectors/vectors/tensors

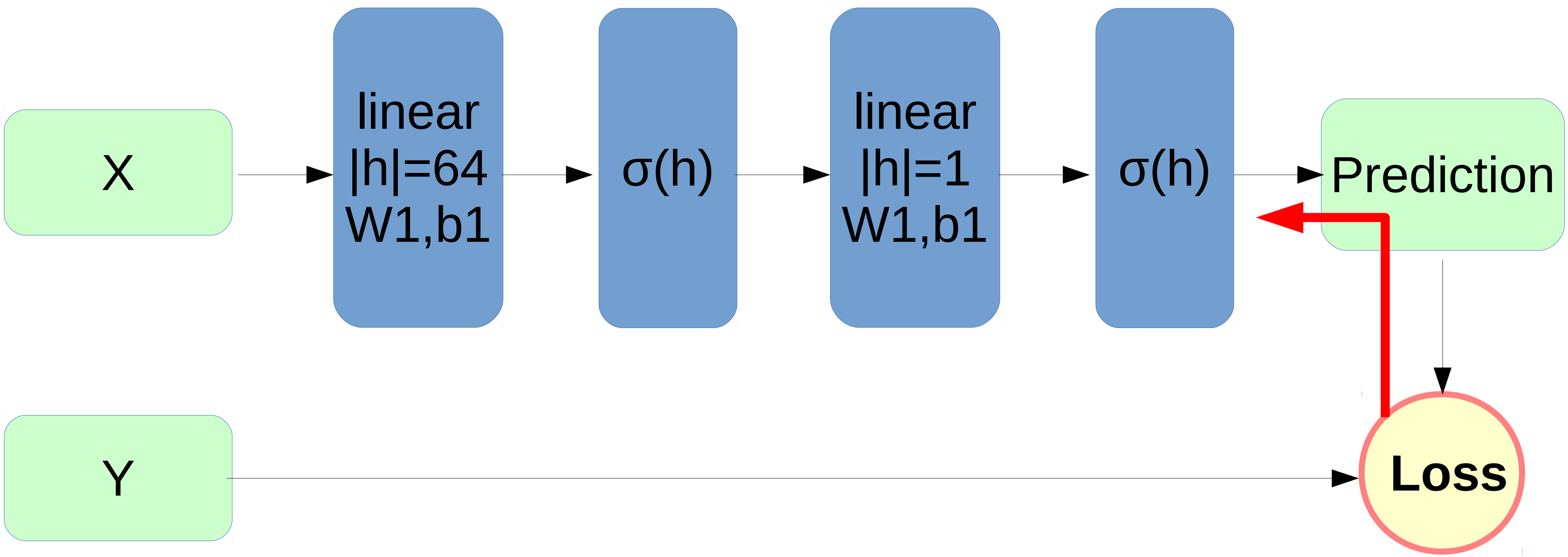


Backpropagation



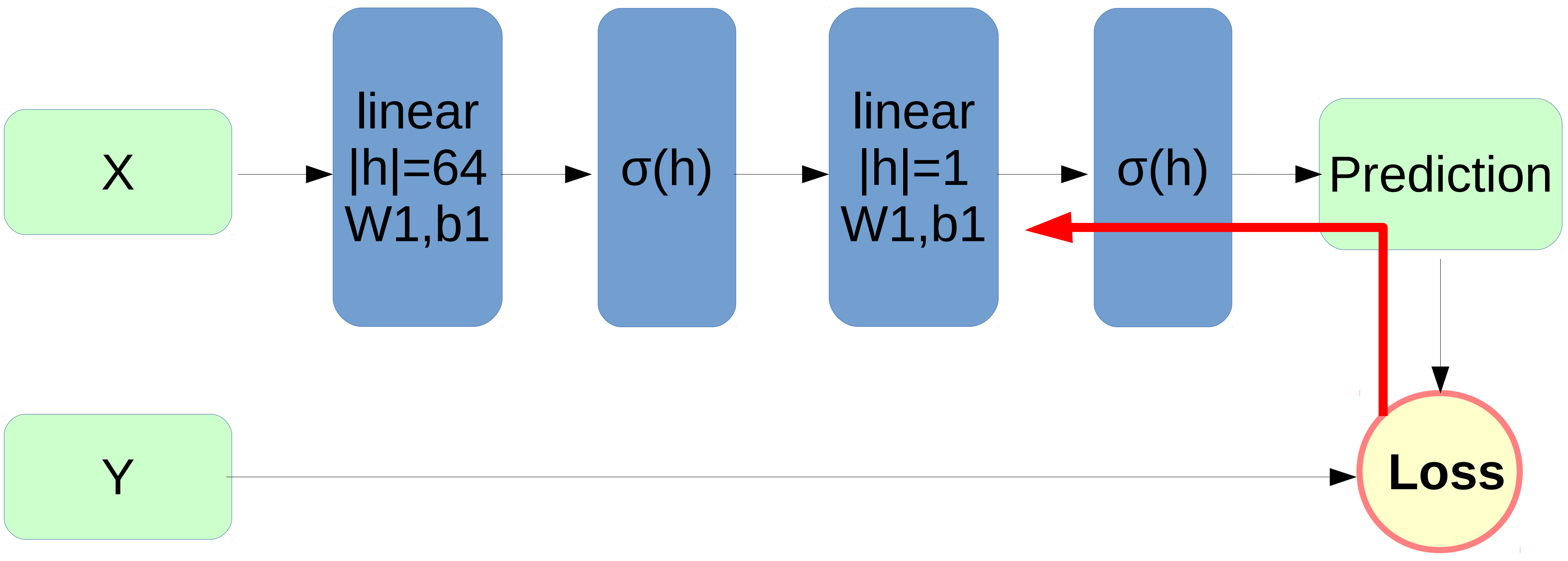
$$\frac{\partial L(\sigma(linear_{w2,b2}(\sigma(linear_{w1,b1}(x)))))}{\partial w1} = \ldots$$

Backpropagation



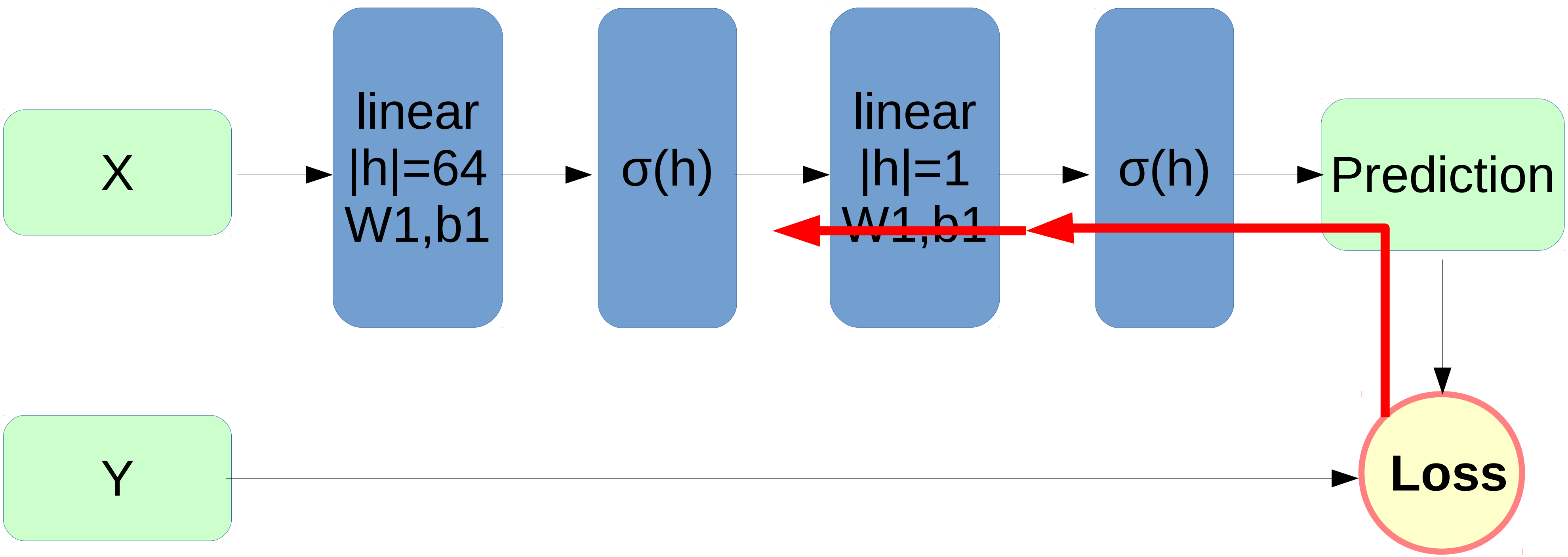
$$\frac{\partial L}{\partial w1} = \frac{\partial L}{\partial \sigma} \cdot$$

Backpropagation



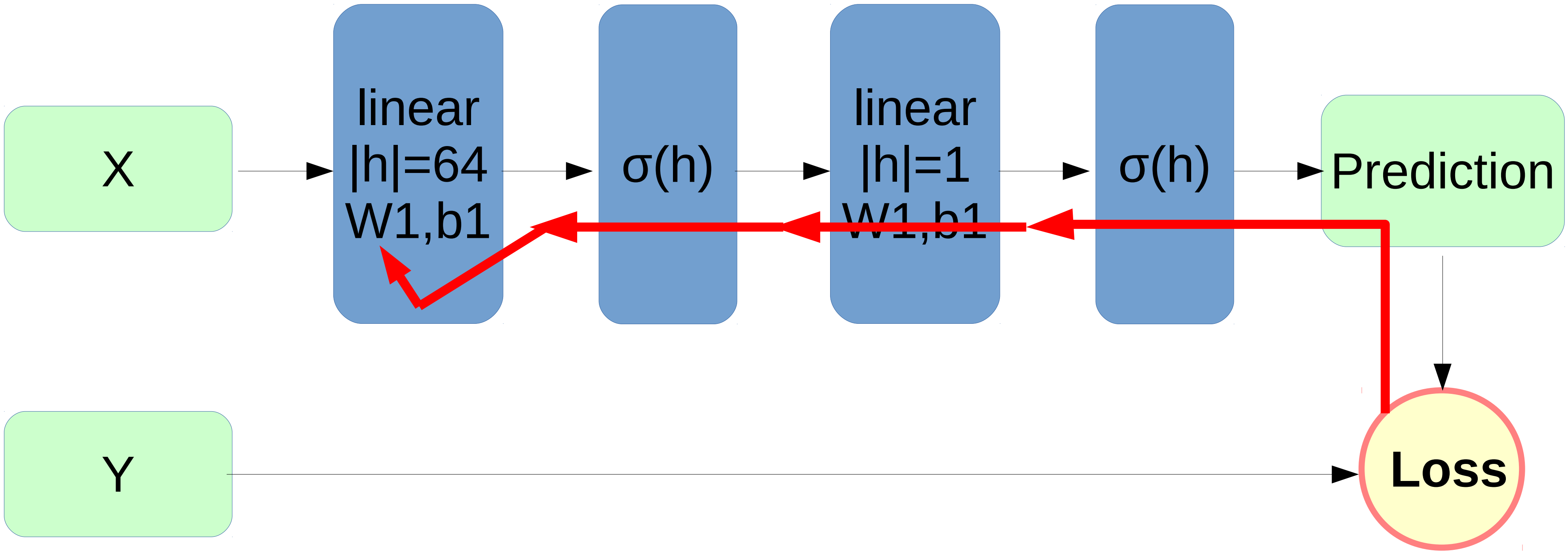
$$\frac{\partial L}{\partial w1} = \frac{\partial L}{\partial \sigma} \cdot \frac{\partial \sigma}{\partial\, linear_{w2,b2}} \cdot$$

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

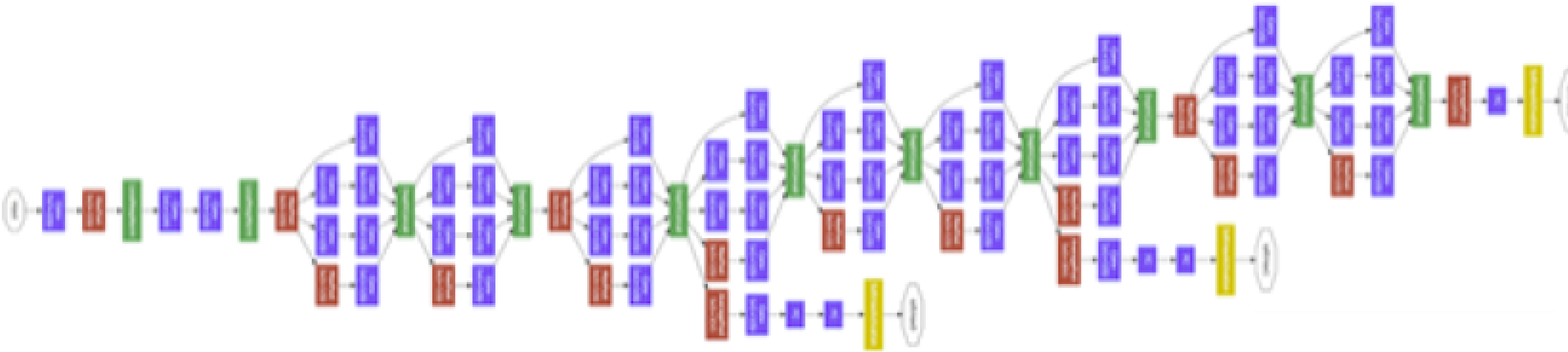
$$\text{sigmoid}: \frac{\partial L}{\partial \sigma(x)} \cdot [\sigma(x) \cdot (1 - \sigma(x))]$$

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

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

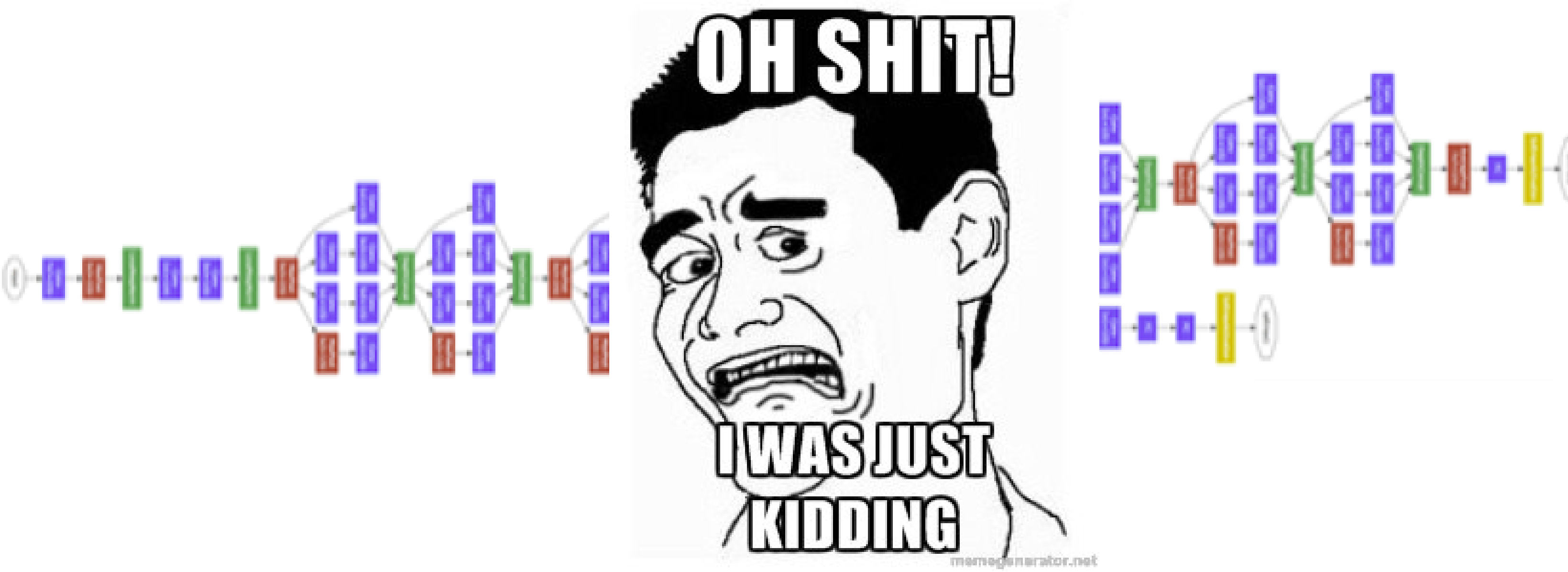
$$\text{linear over W}: \frac{1}{\|X\|} \cdot X^T \times \frac{\partial L}{\partial [X \times W + b]}$$

And now let's differentiate



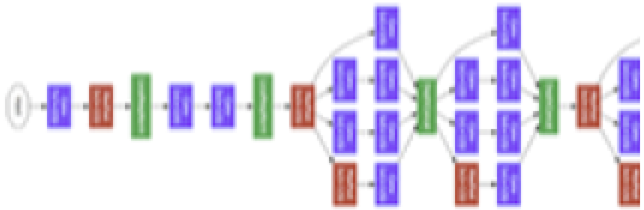
- 5+ types of layers
- each with different dimensions
- parallel branches with independent losses
- several nonlinearities

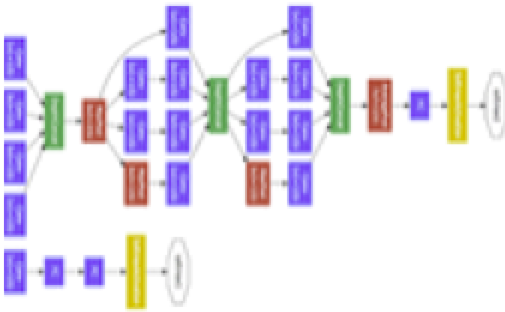
And now let's differentiate



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

Deep learning frameworks

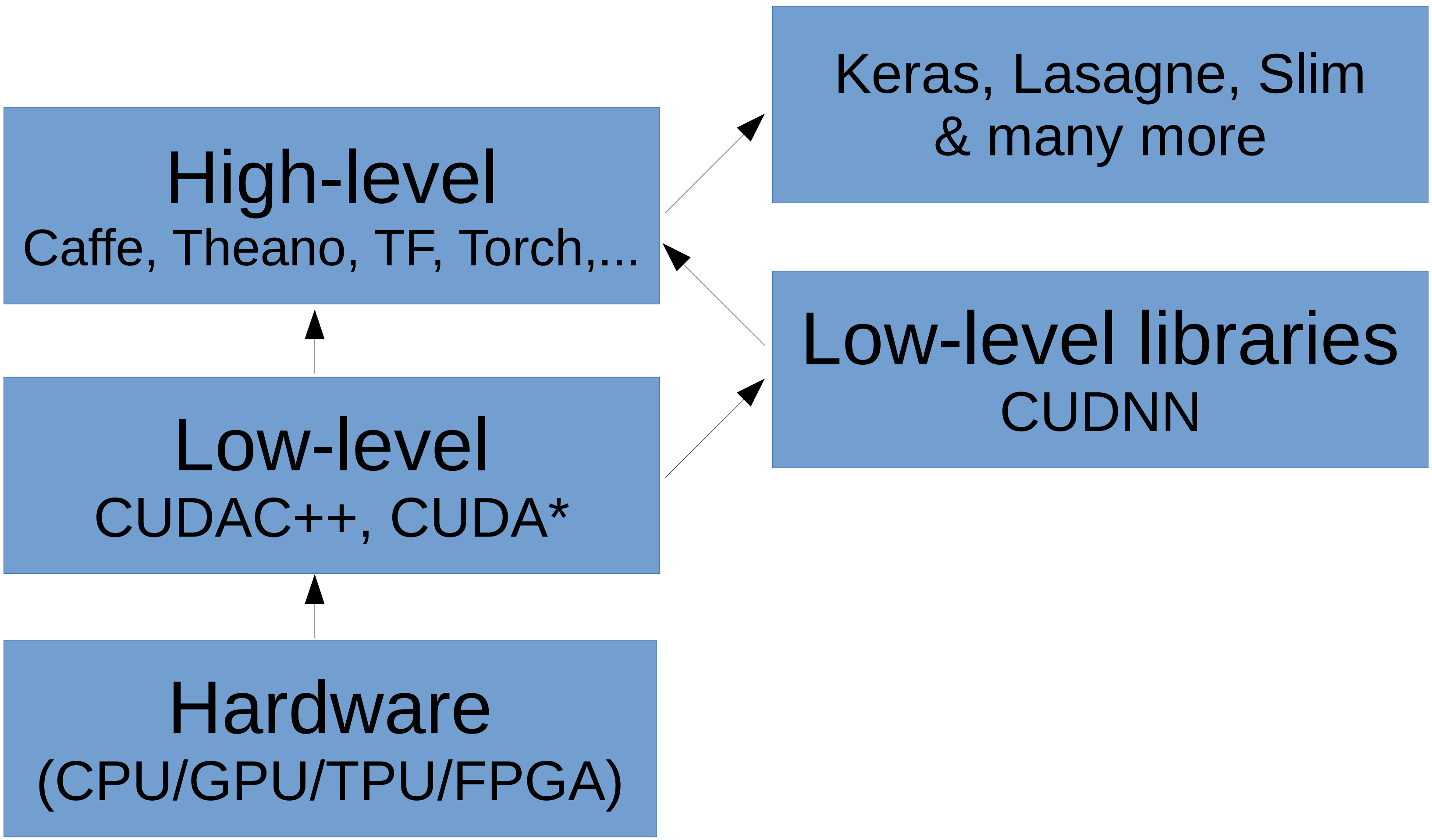




- 5+ types of layers
- each with different dimensions
- parallel branches with independent losses
- several nonlinearities

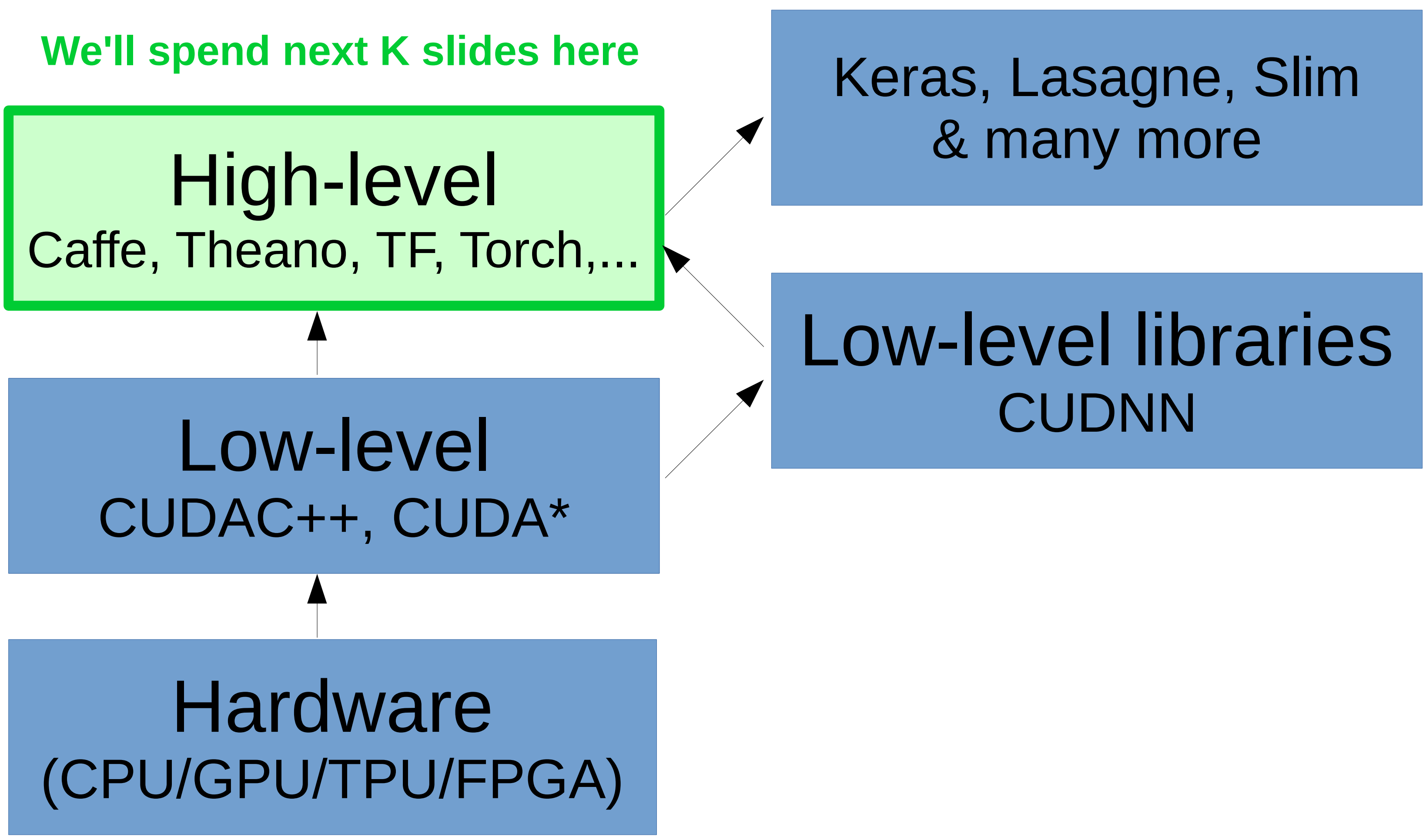
Deep learning frameworks

- Core idea: helps you define and train neural nets



Deep learning frameworks

- Core idea: helps you define and train neural nets



Deep learning frameworks

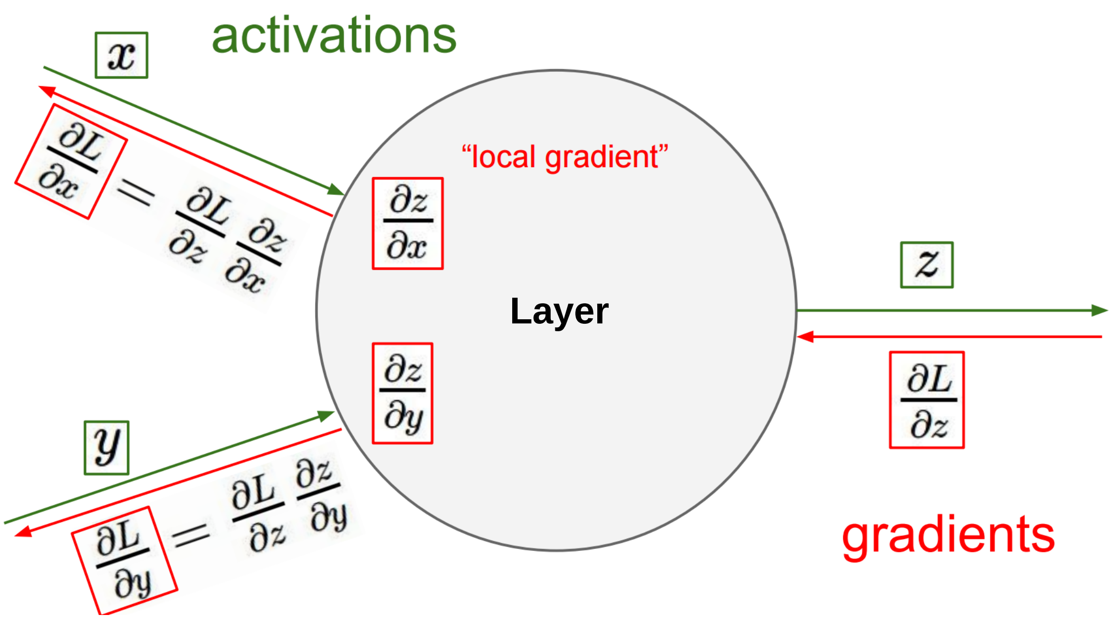
Layer-based frameworks:
  Same idea as in our hand-made neural net

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_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

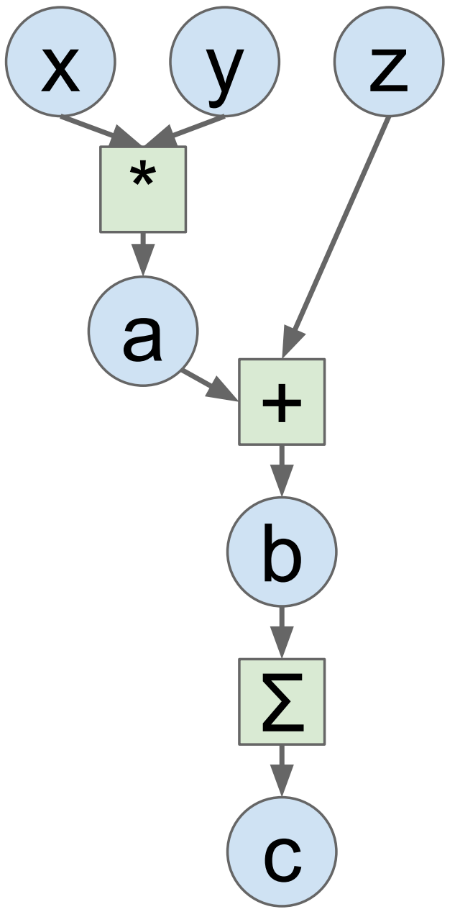
Symbolic graphs

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

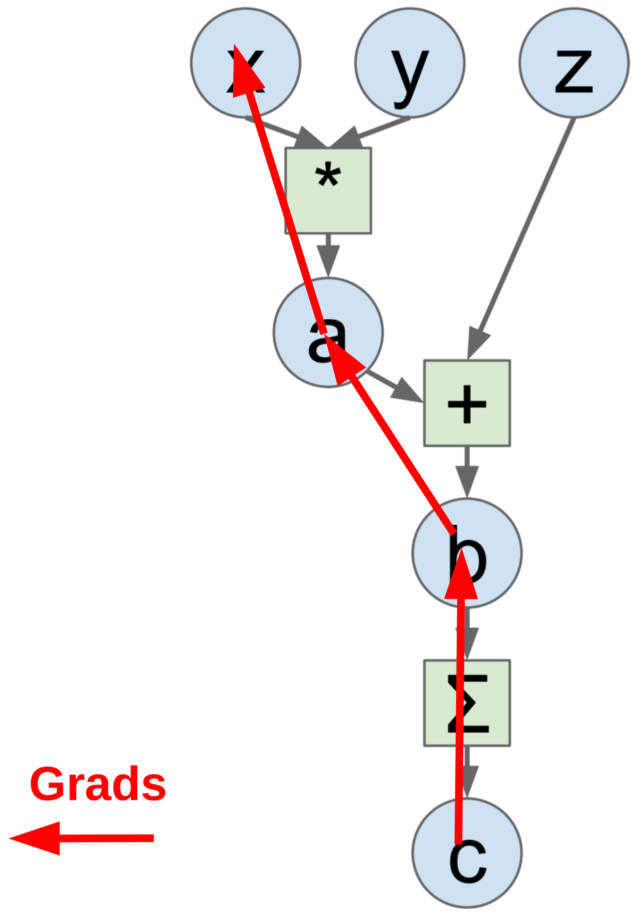



```
a = x * y
b = a + z
c = np.sum(b)
```

Idea: let's define
this graph explicitly!

This many 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

This 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

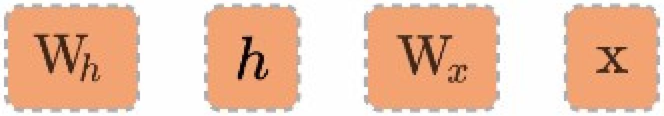
Dynamic graphs

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))
```



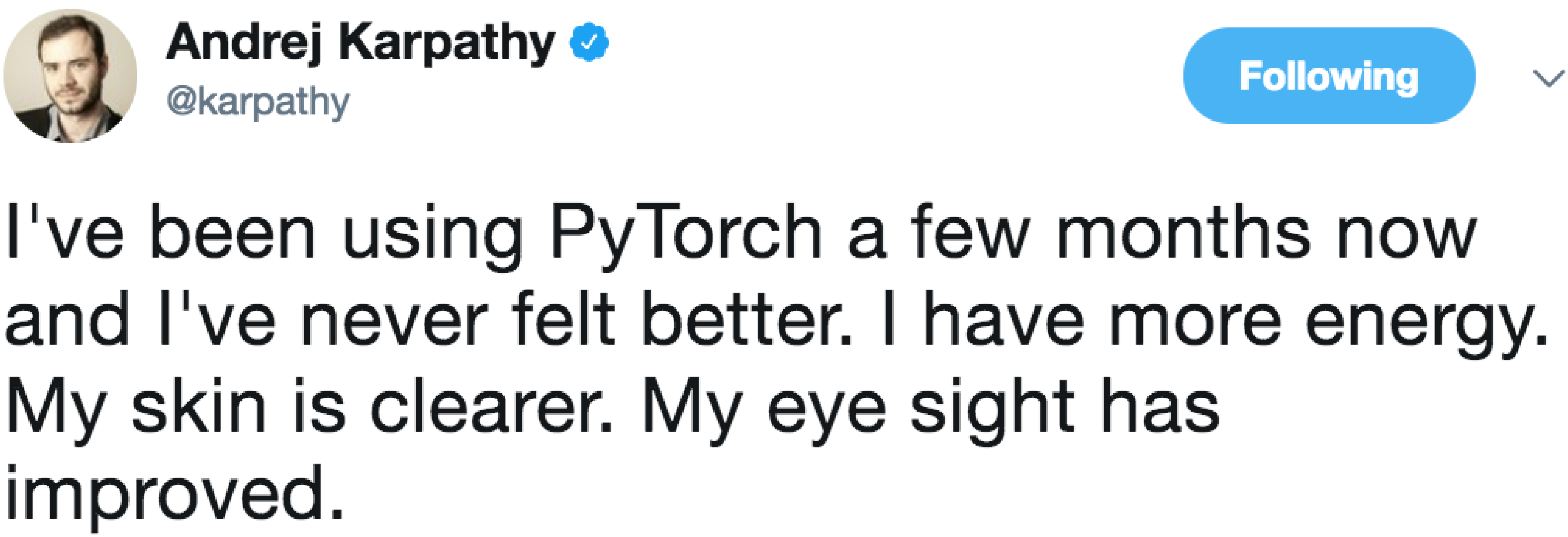
Dynamic graphs

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!

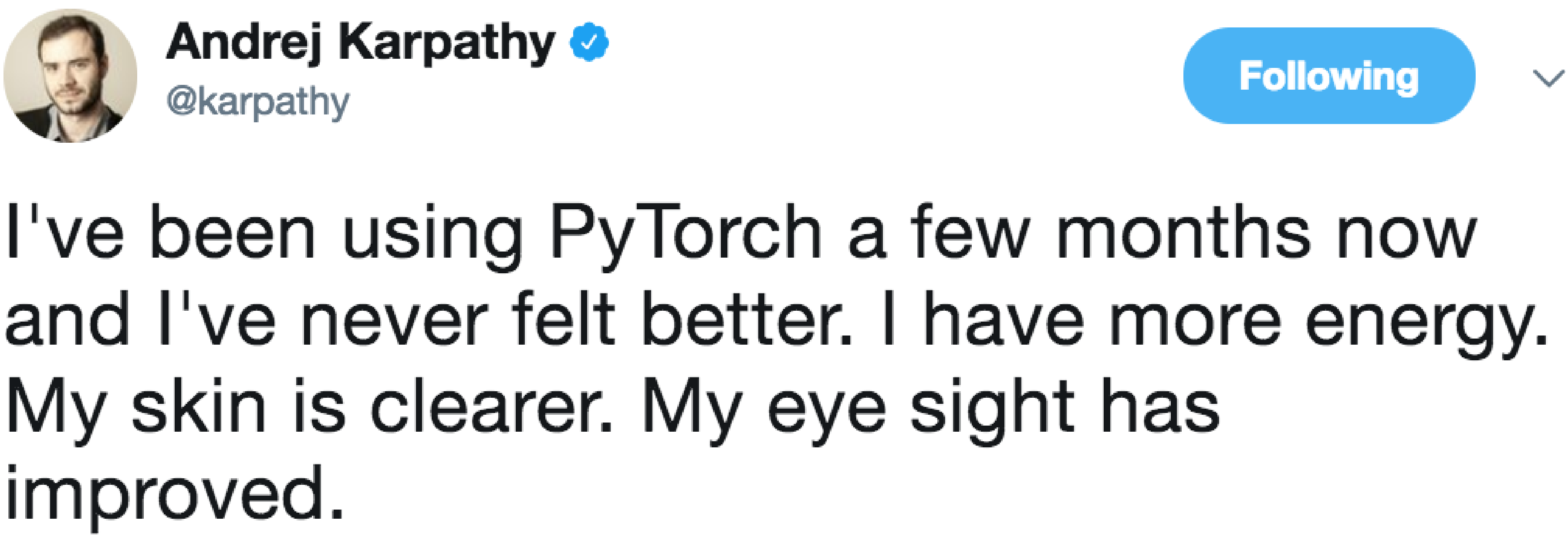
Dynamic graphs


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!

Dynamic graphs

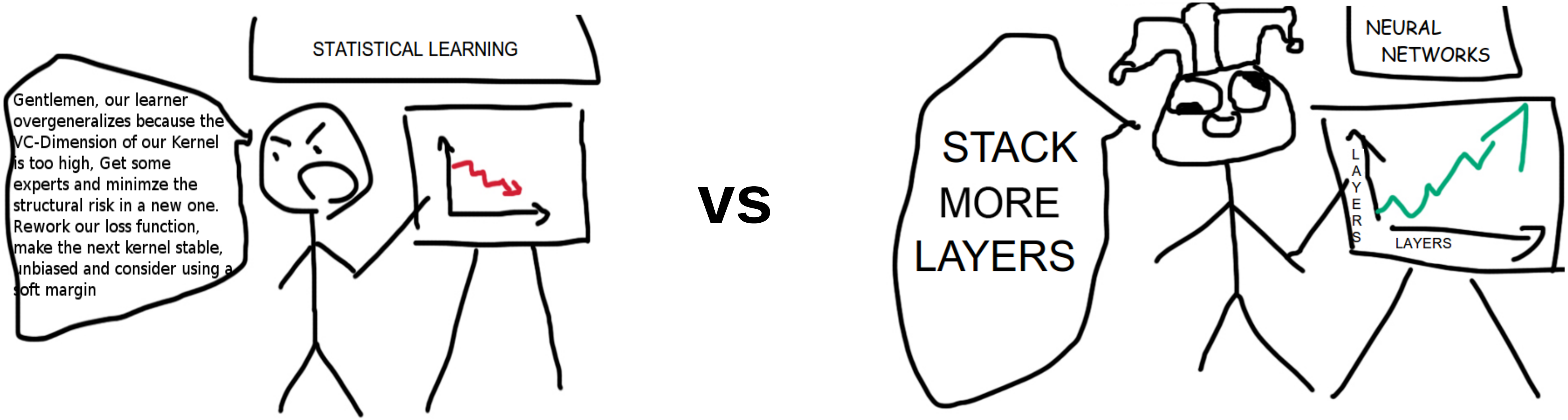


Researchers love them!

Lemme grumble a bit

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

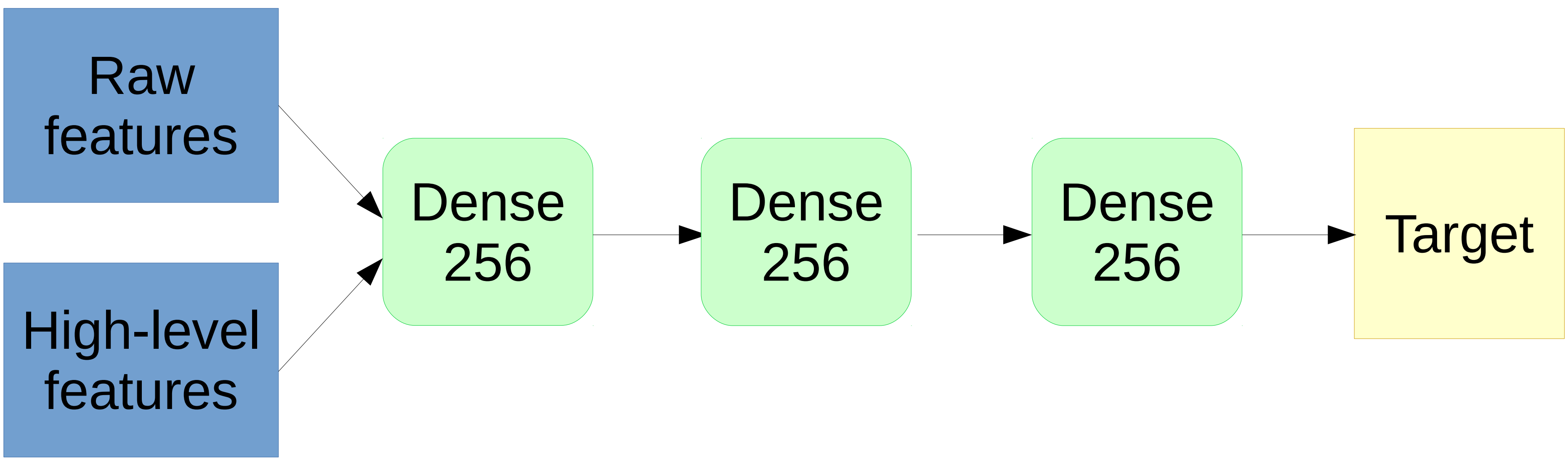
Deep learning is a language

Say, you train classifier on two sets of features

Raw features	Car photo (image pixels)	Car price	Target
High-level features	Car brand, model, age, blemishes		

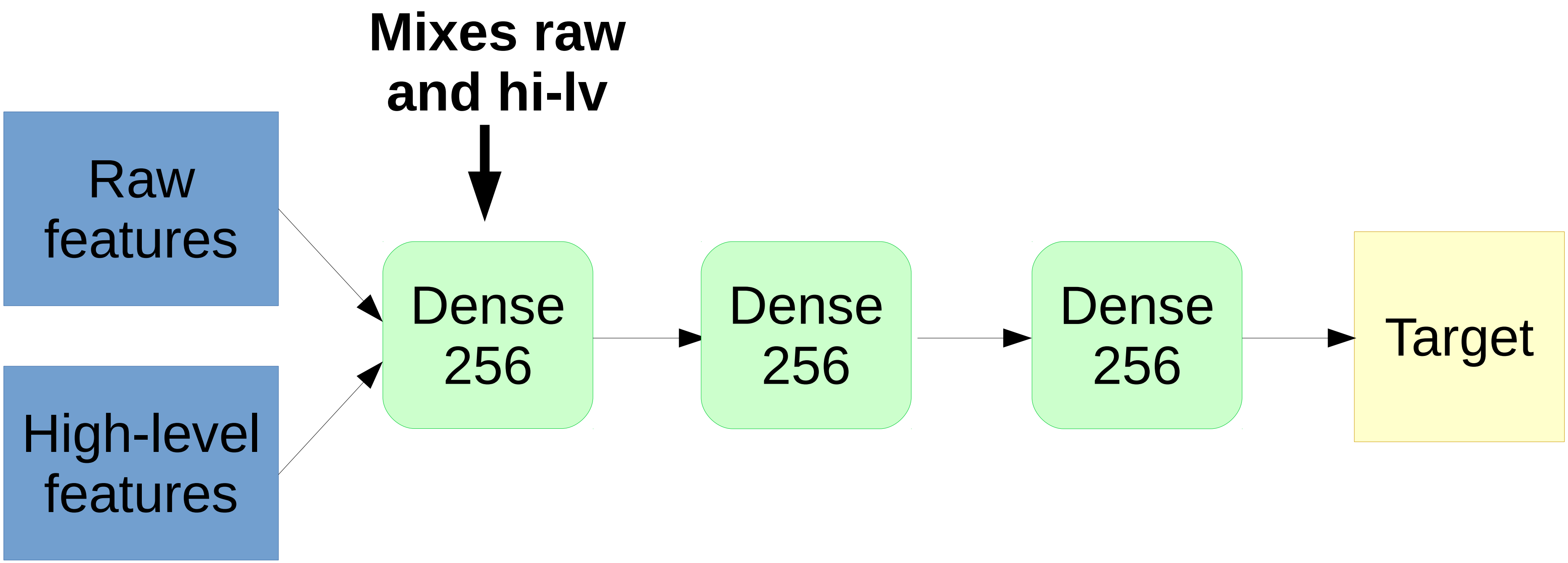
Deep learning is a language

Naive approach



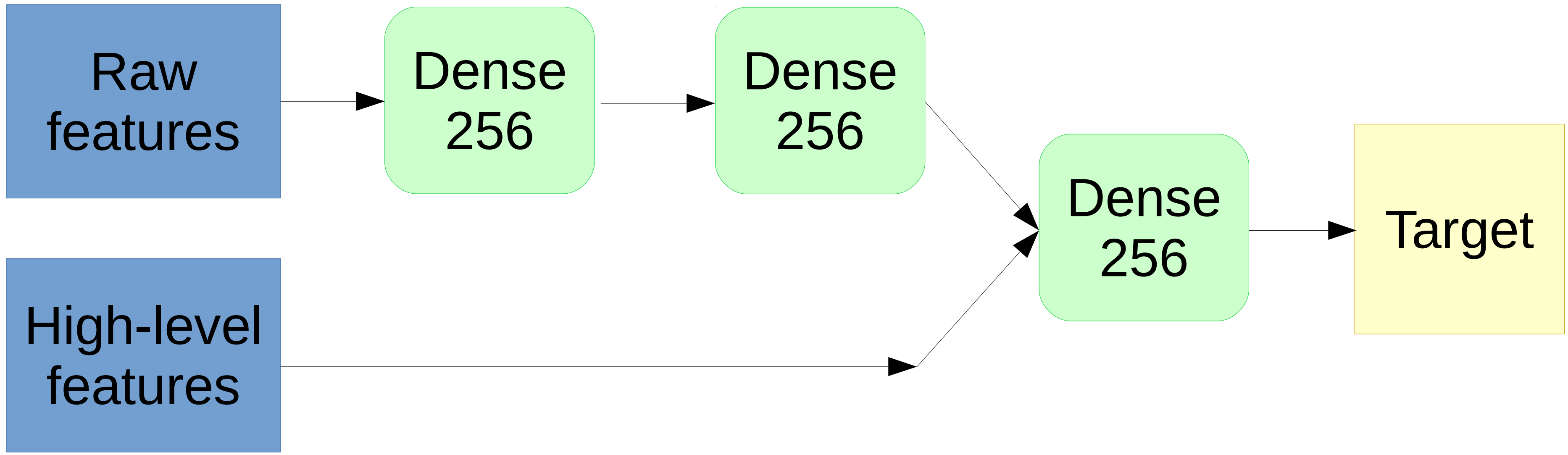
Deep learning is a language

Naive approach



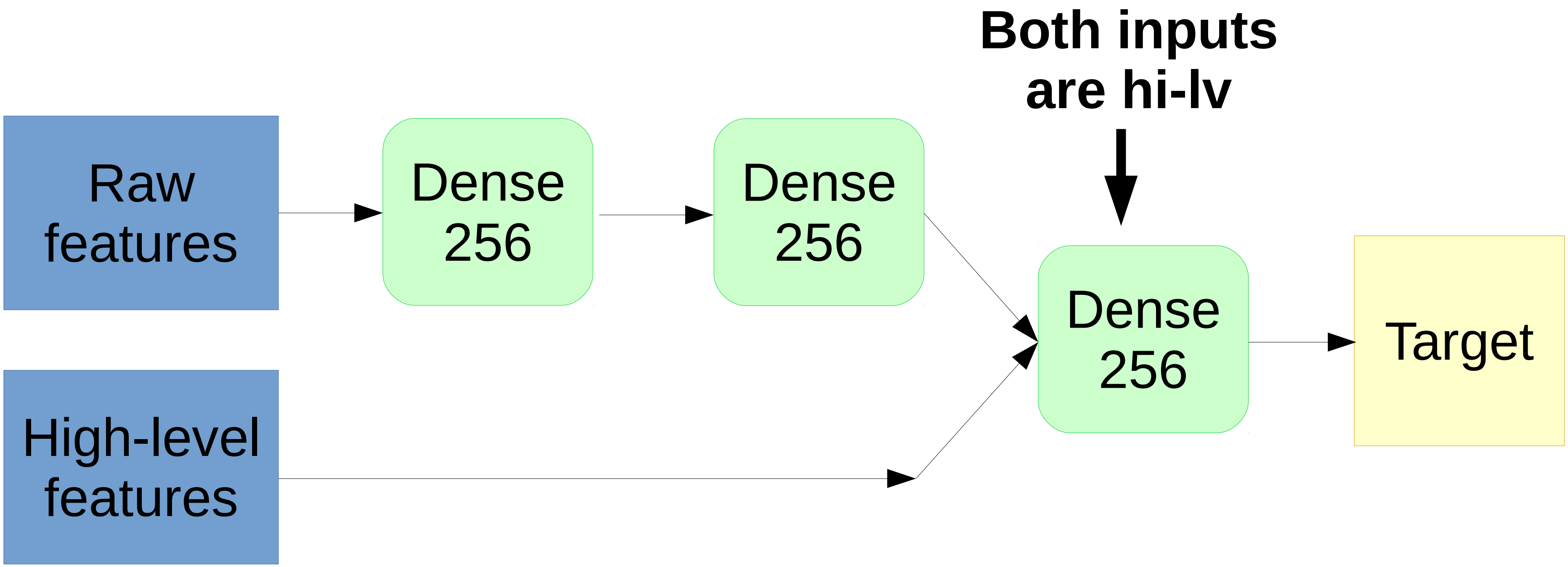
Deep learning is a language

Less naïve approach



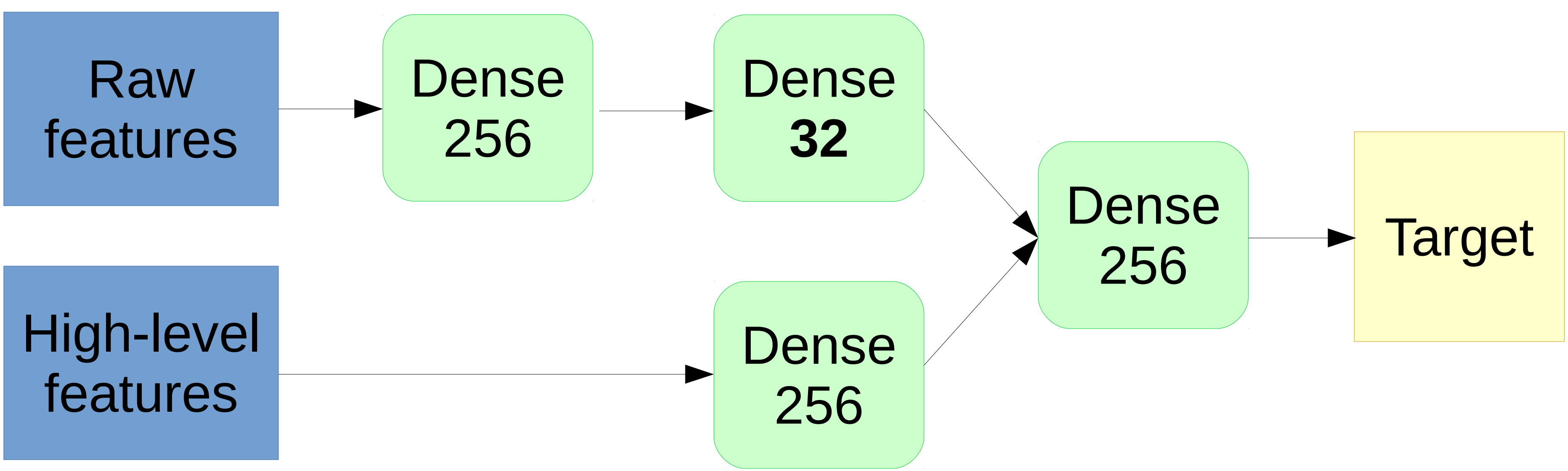
Deep learning is a language

Less naïve approach



Deep learning is a language

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

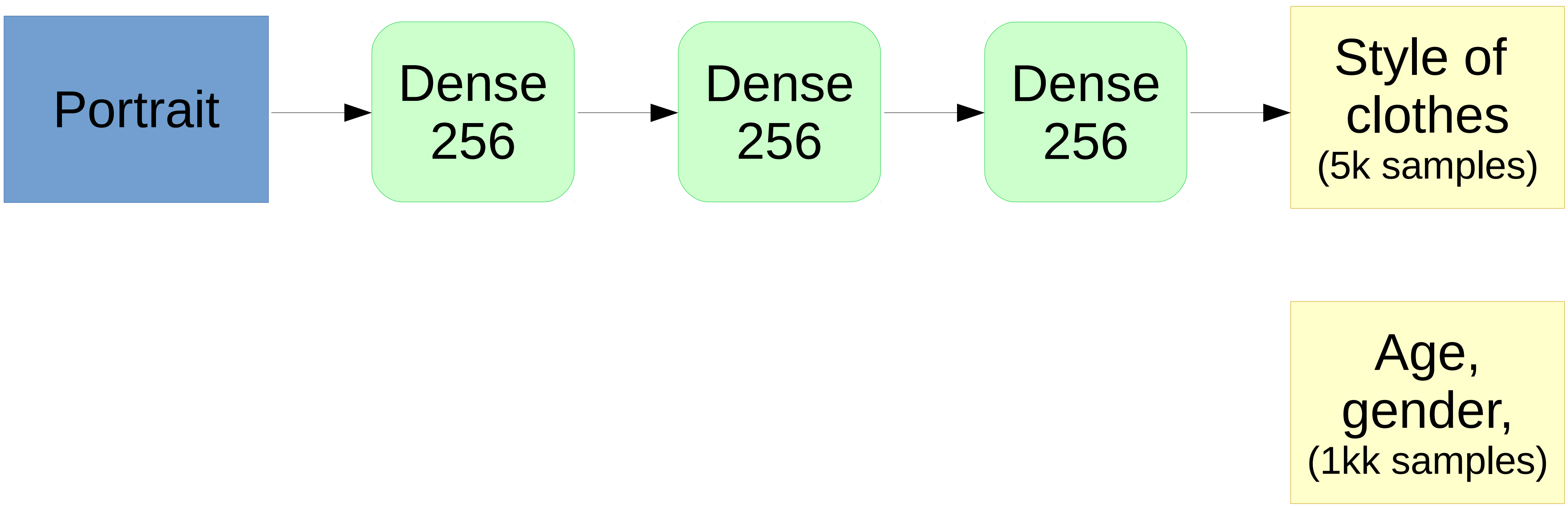


Deep learning is a language

You have a small dataset

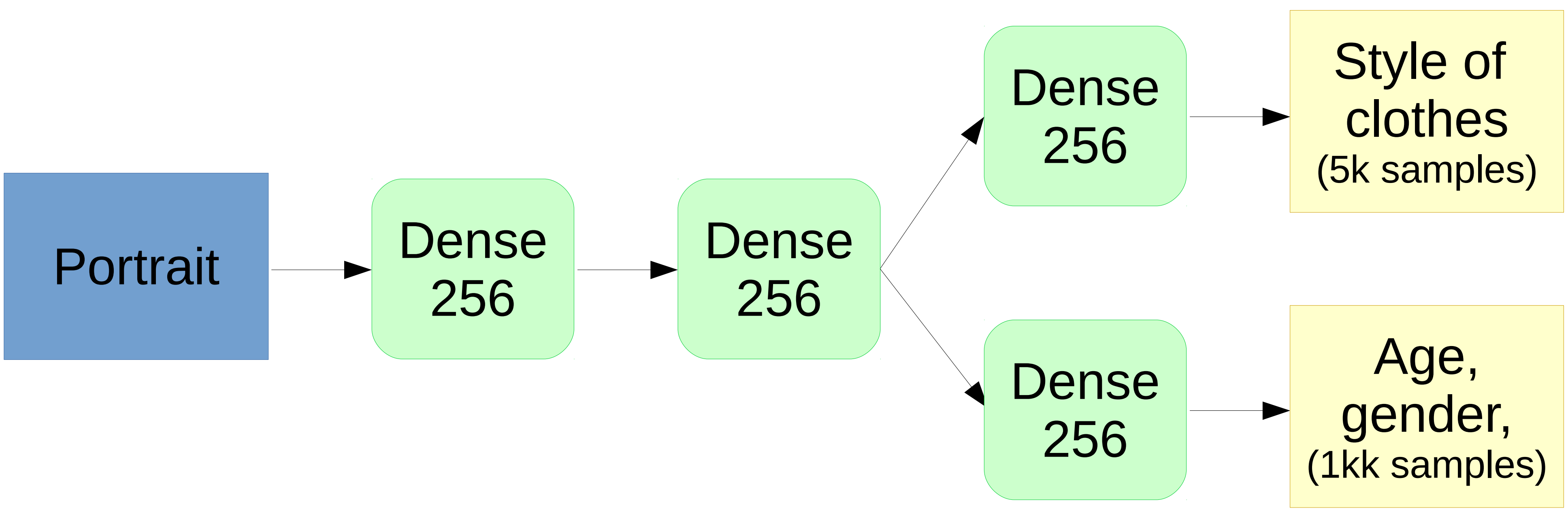
Deep learning is a language

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



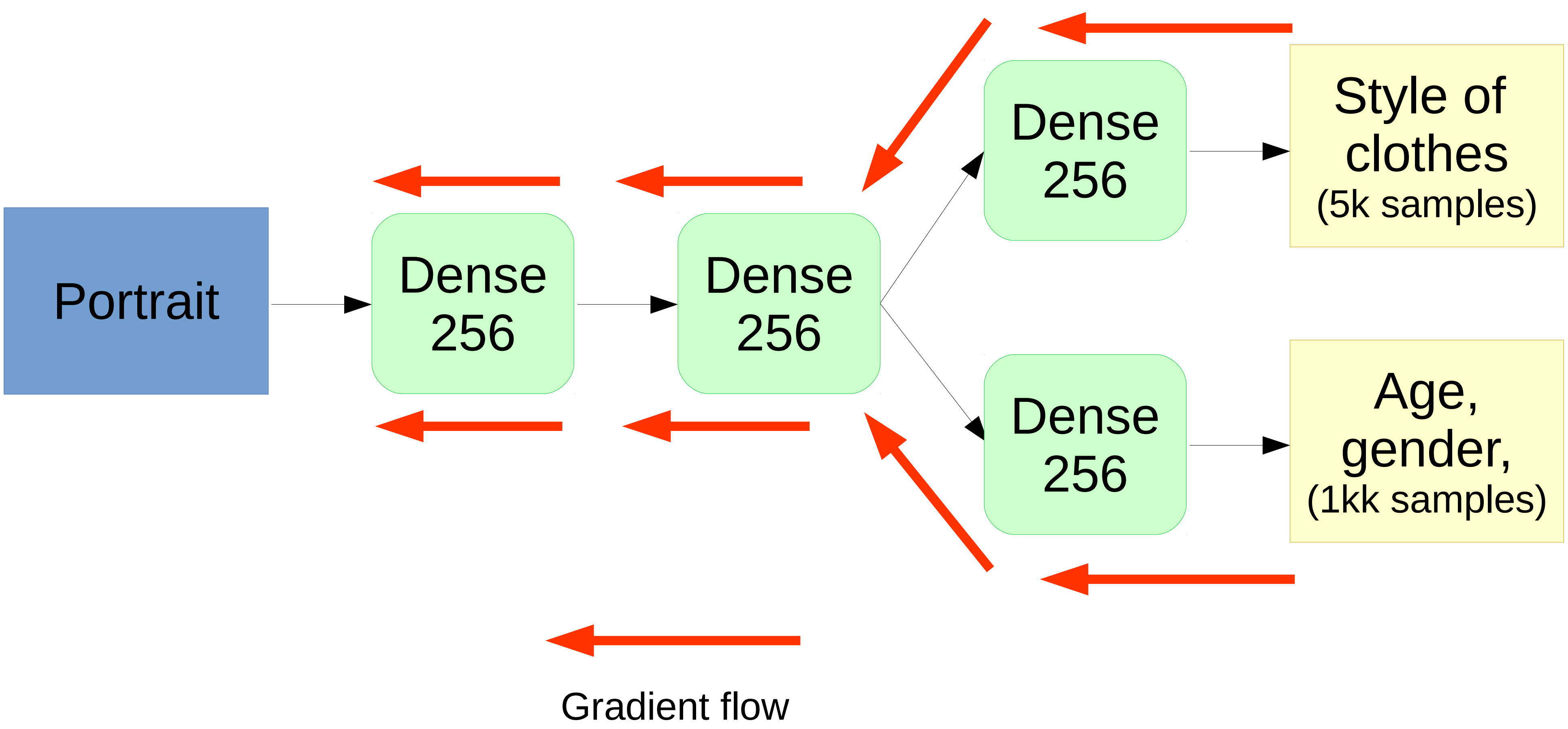
Deep learning is a language

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”
- “I don't want model to be indifferent to small shifts”

For texts:
- “Model should reconstruct the underlying process”

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**

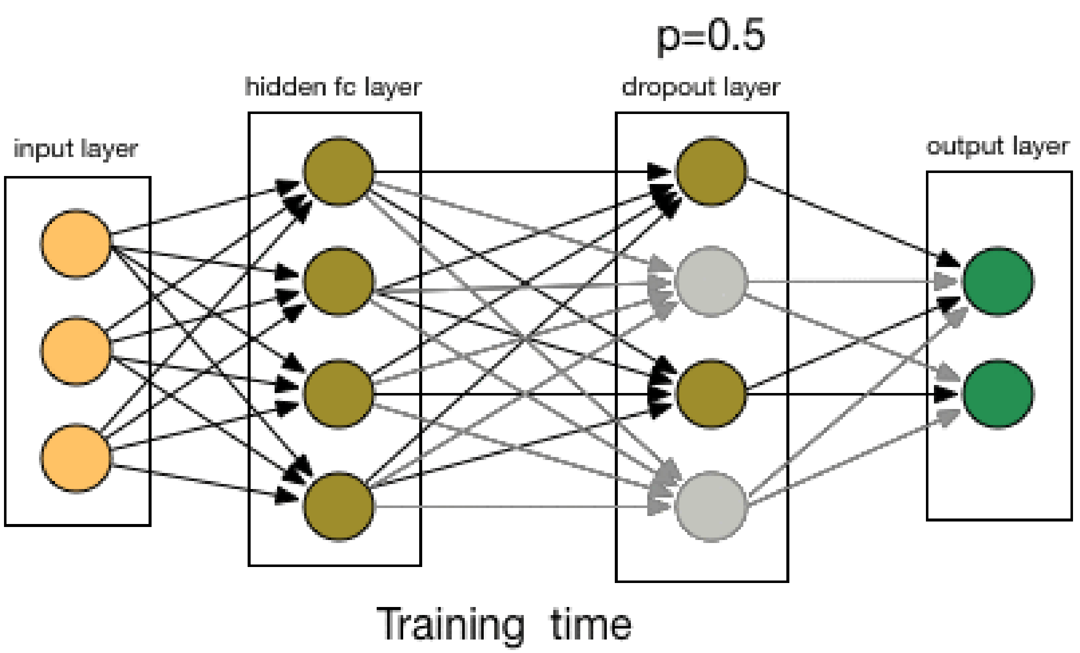
multiply neurons by zero!

- Scale up the remaining neurons to keep average the same

Regularization

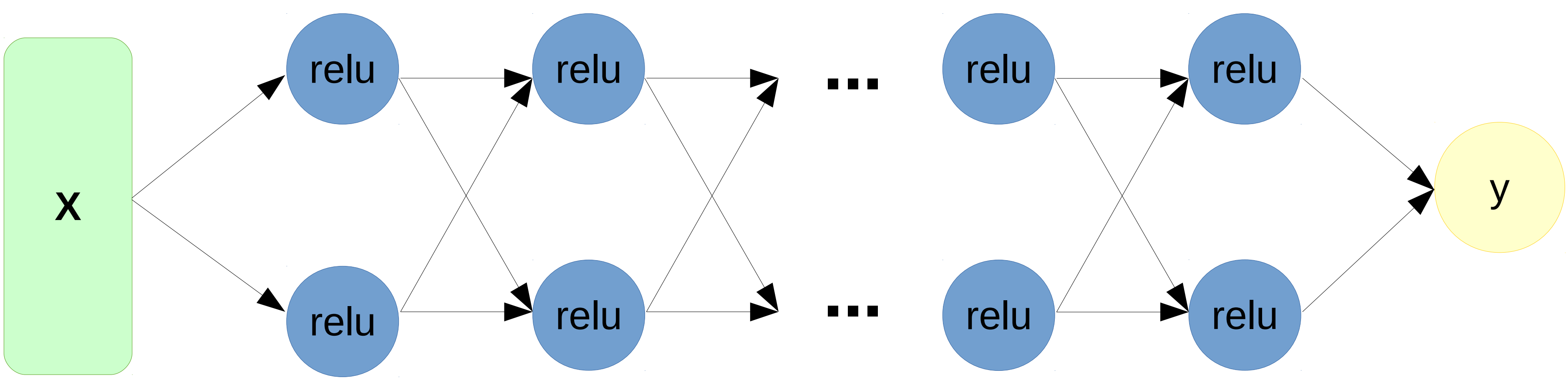
- Dropout:

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



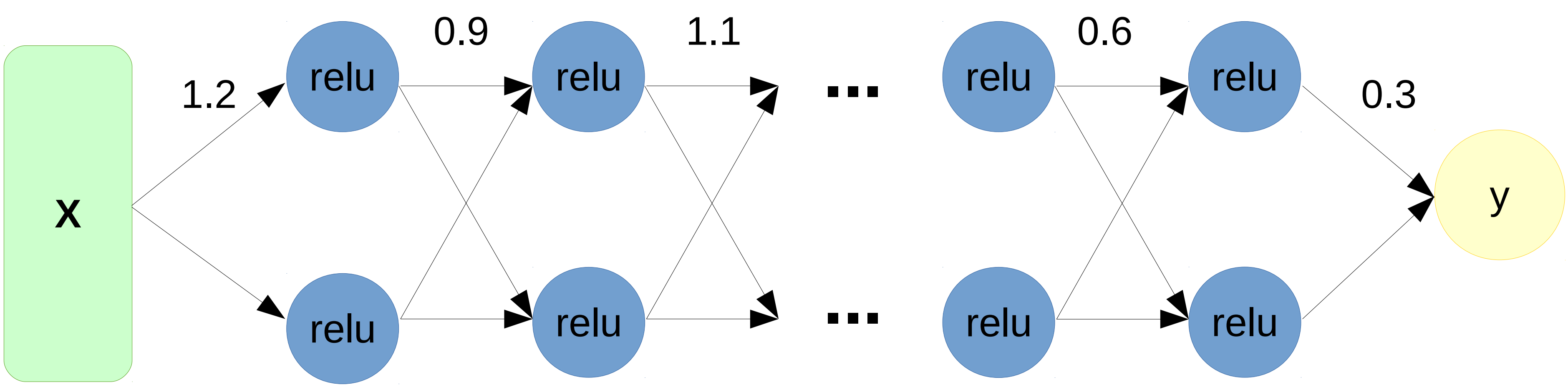
The problem with deep networks

- Imagine a 100-layer network with ReLU



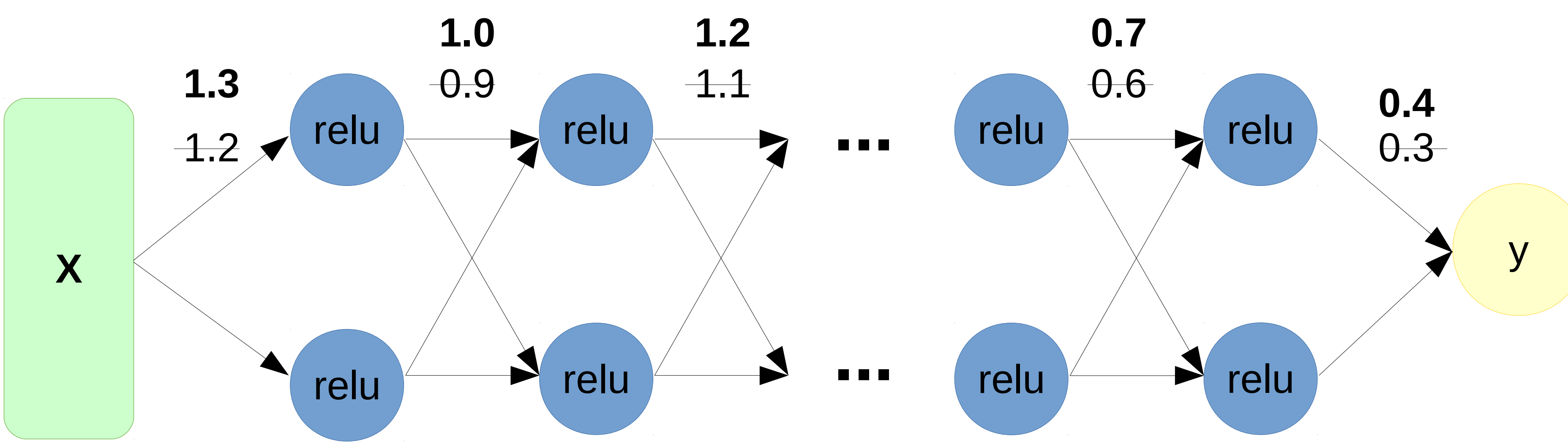
The problem with deep networks

- Imagine a 100-layer network with ReLU



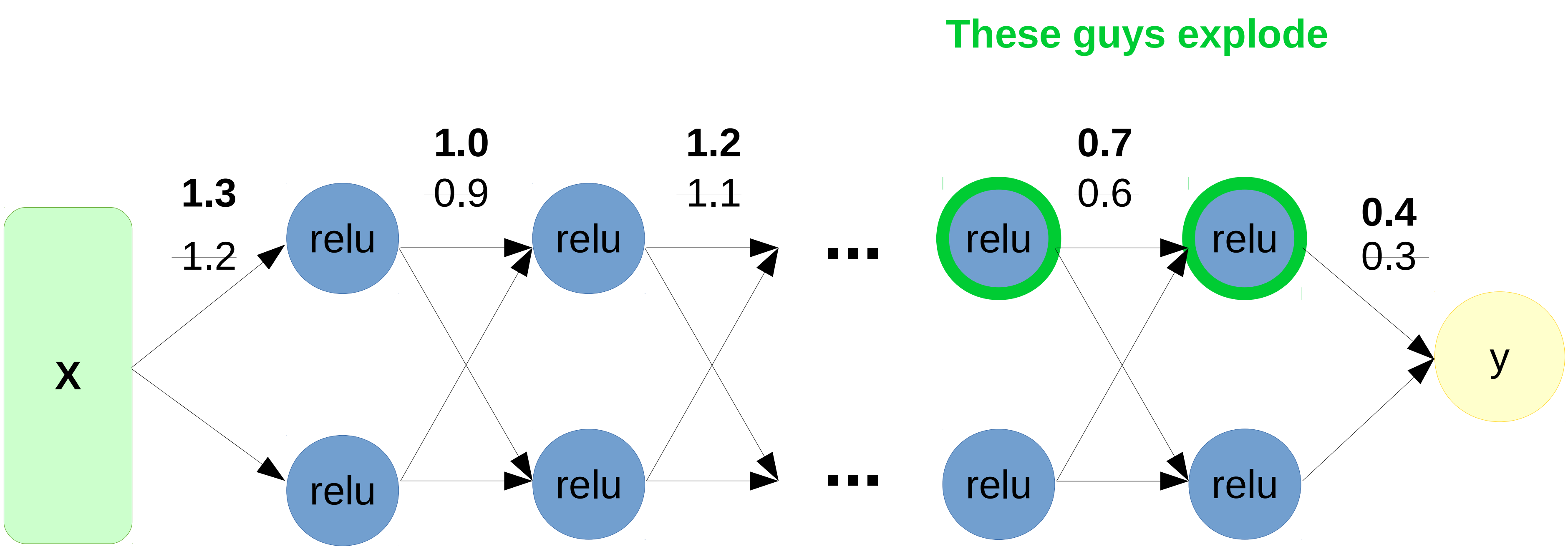
The problem with deep networks

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



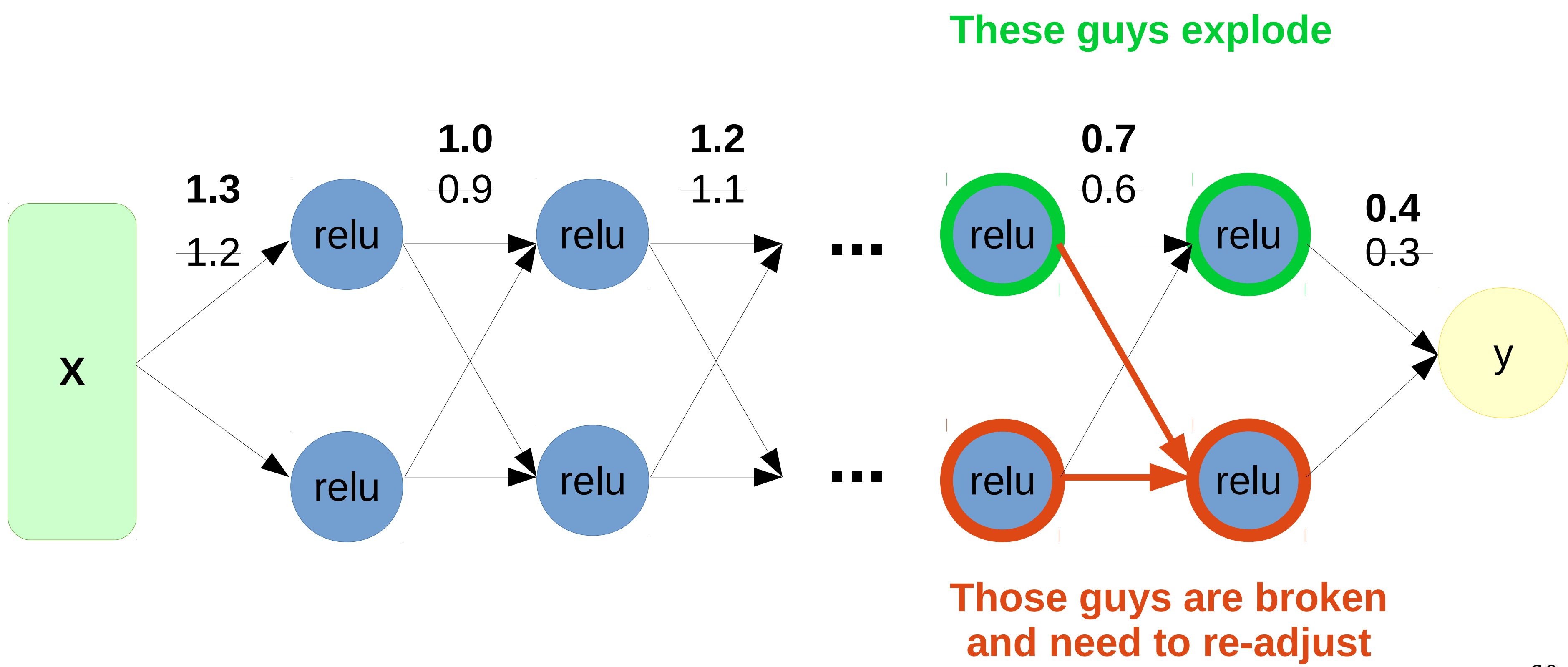
The problem with deep networks

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



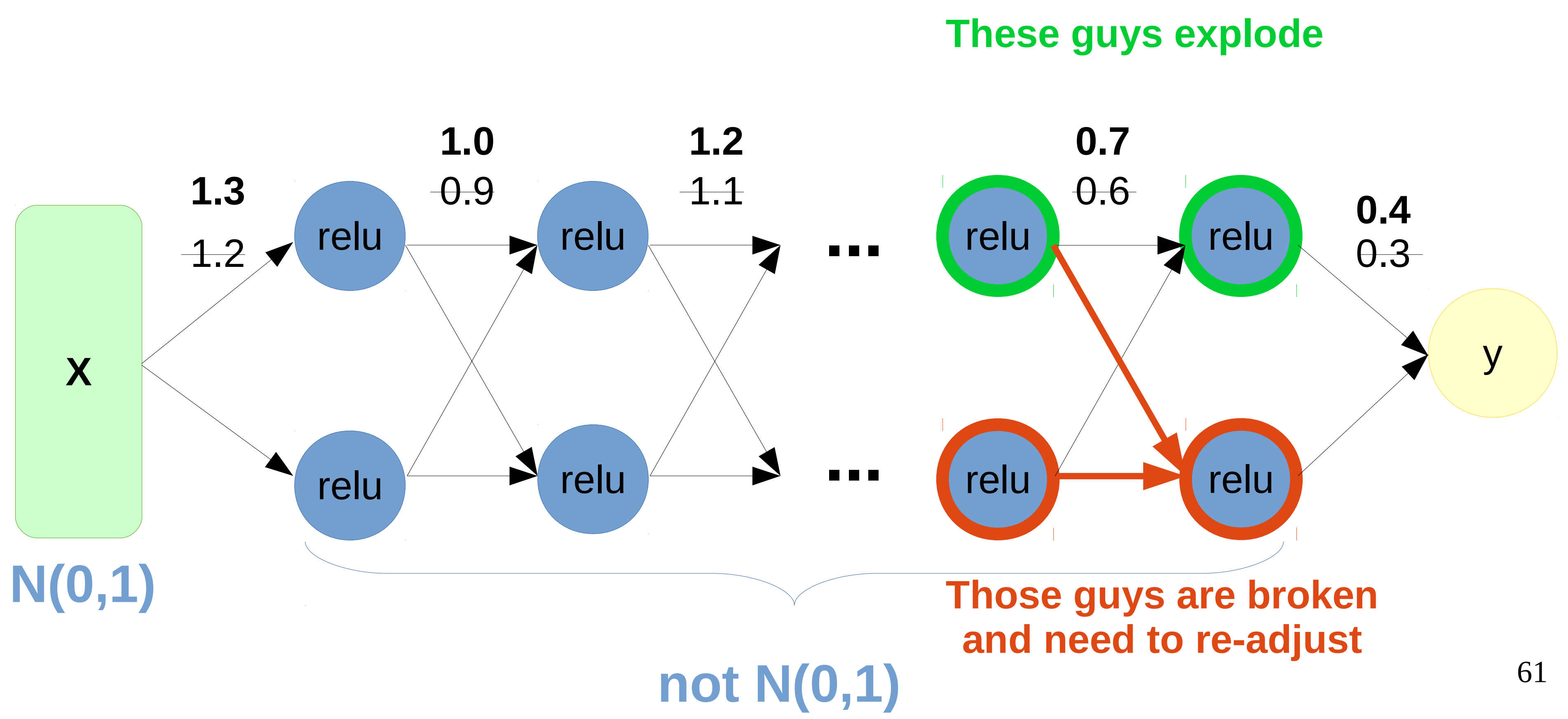
The problem with deep networks

- 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)

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

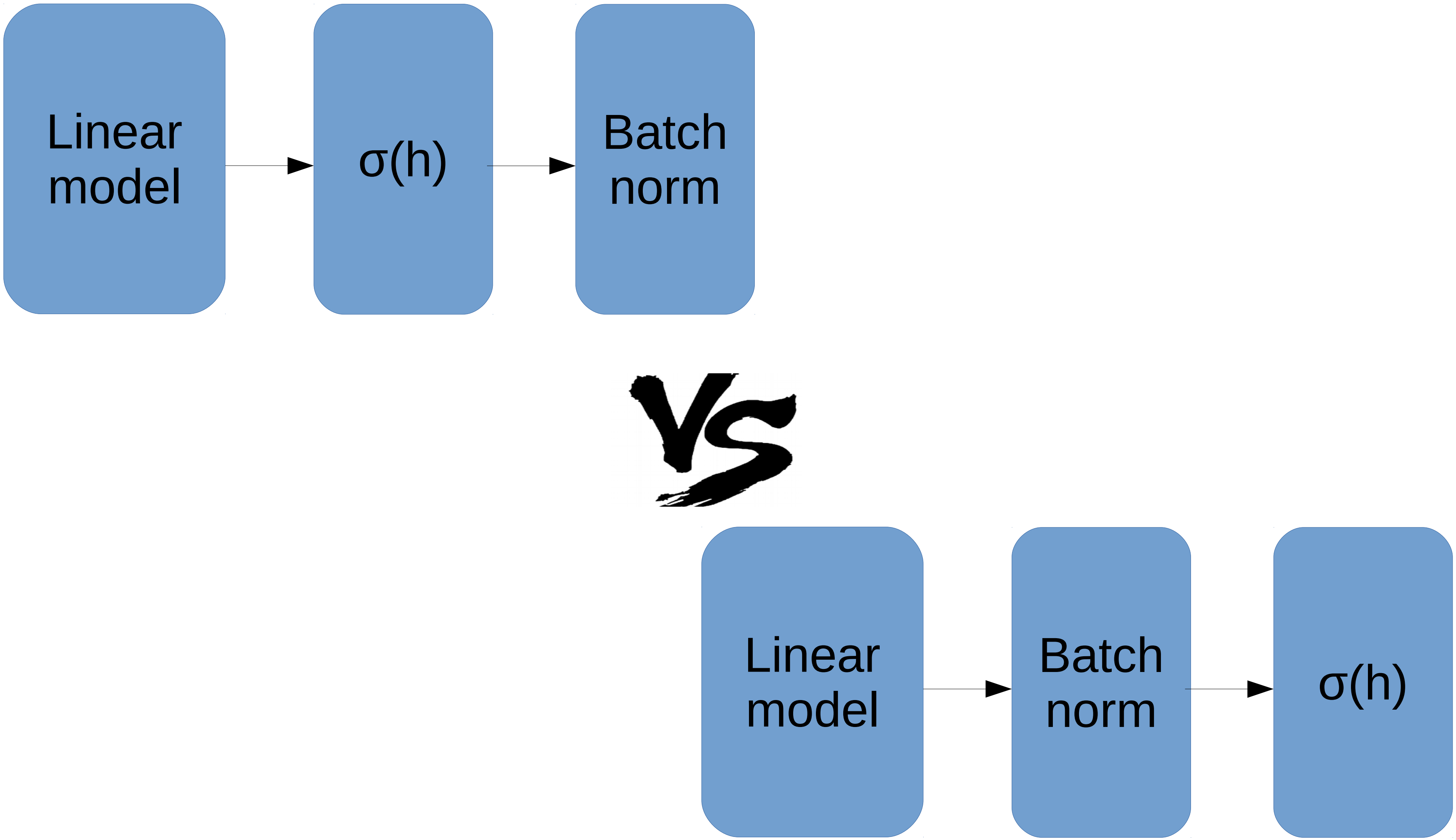
i stands for i-th neuron

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

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

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

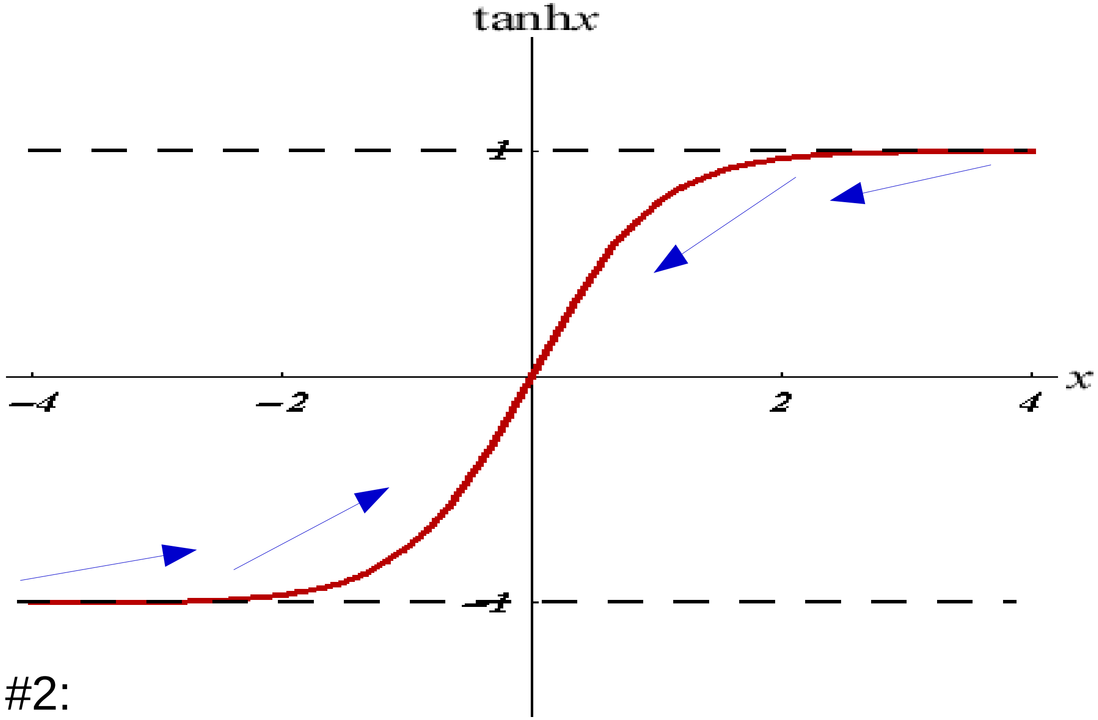
Batch normalization



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

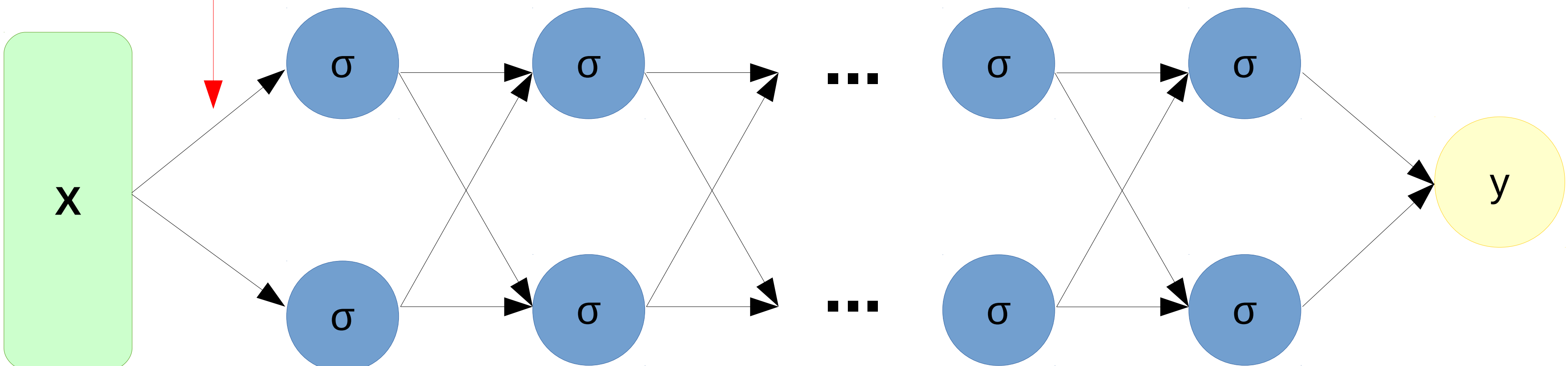
Self-normalizing neural networks (SELU)

other

The problem with deep networks

- Imagine a 100-layer network with ReLU

w consider this weight

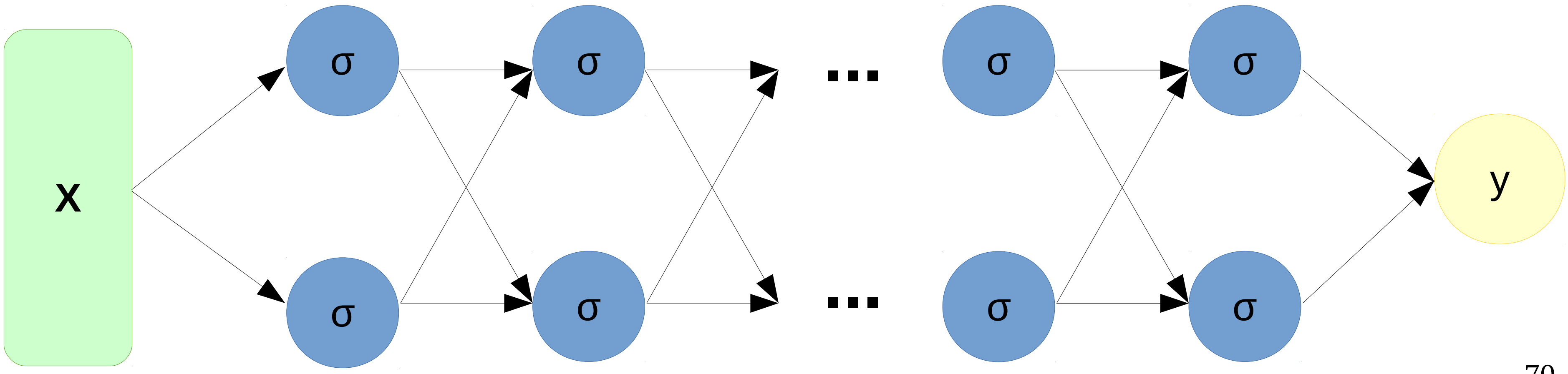


The problem with deep networks

other

- Imagine a 100-layer network with ReLU

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial h_N} \cdot \frac{\partial h_N}{\partial h_{N-1}} \cdot \ldots \cdot \frac{\partial h_1}{\partial w}$$



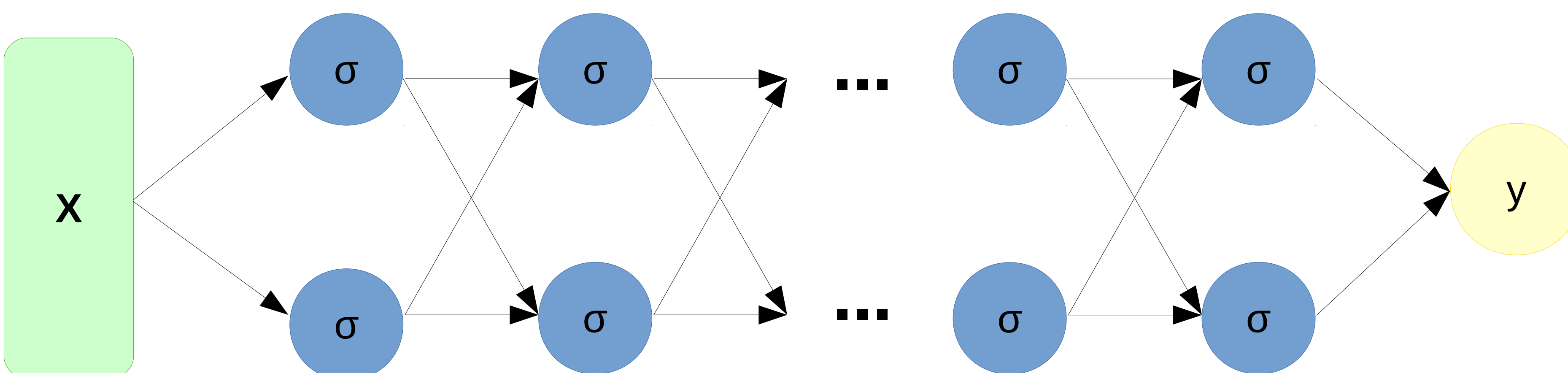
other

The problem with deep networks

- Imagine a 100-layer network with ReLU

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial h_N} \cdot \frac{\partial h_N}{\partial h_{N-1}} \cdot \ldots \cdot \frac{\partial h_1}{\partial w}$$

$$\frac{\partial h_{i+1}}{\partial h_i} < 1 \qquad \Rightarrow \qquad \text{gradients approach 0}$$



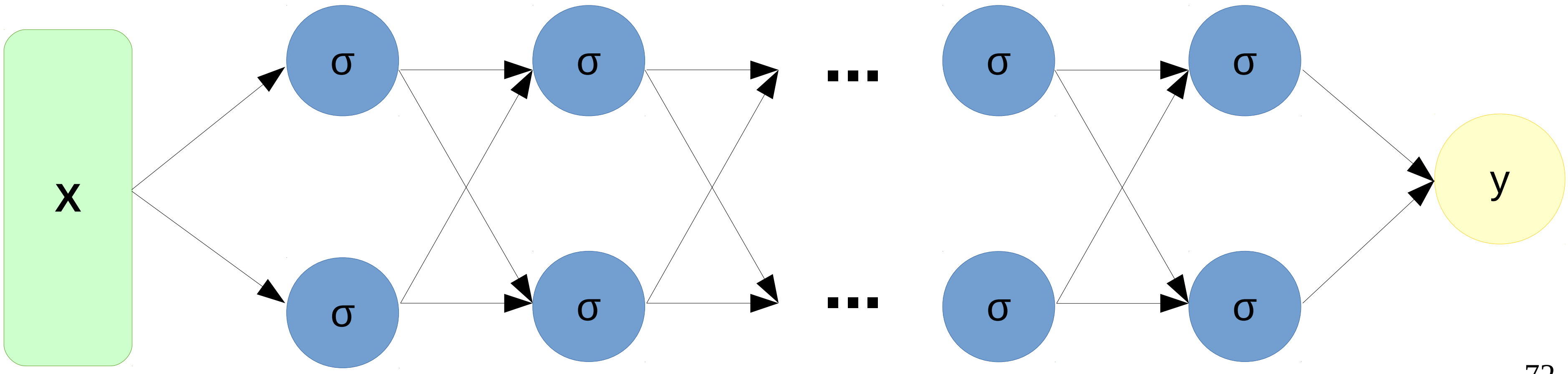
other

The problem with deep networks

- Imagine a 100-layer network with ReLU

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial h_N} \cdot \frac{\partial h_N}{\partial h_{N-1}} \cdot \ldots \cdot \frac{\partial h_1}{\partial w}$$

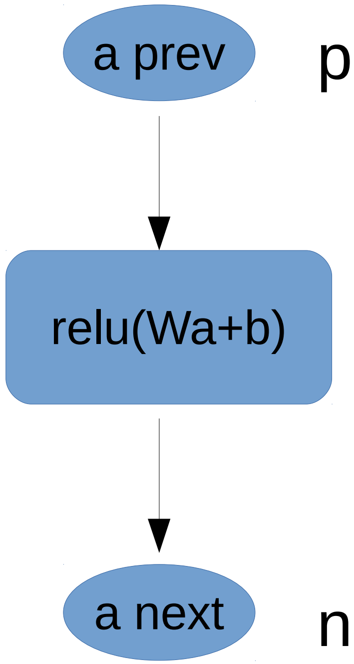
Q: Imagine that each $\frac{\partial h_{i+1}}{\partial h_i} \approx 0.5$; see any problems?



Architectures: residual network

Idea: let's create a shortcut for gradients

Normal layer



previous layer activation

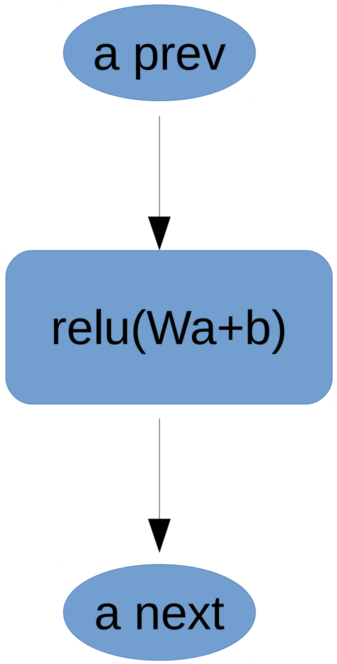
next layer activation

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

Architectures: residual network

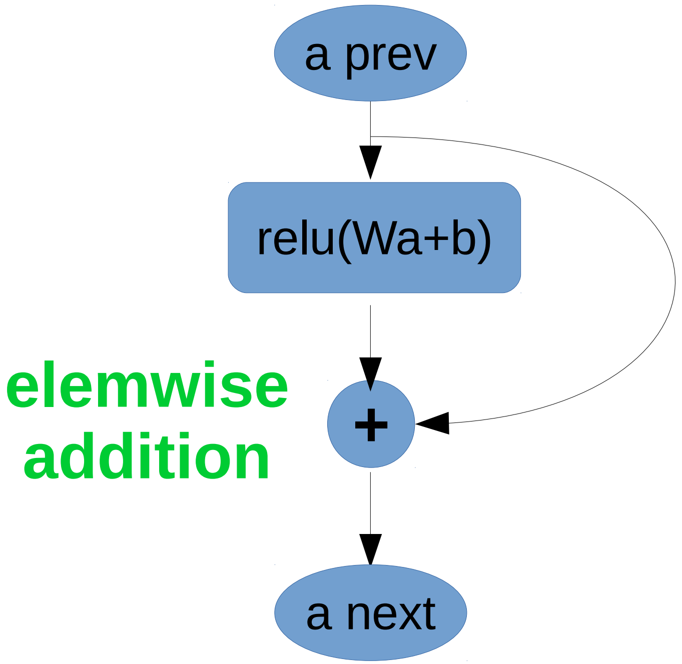
Idea: let's create a shortcut for gradients

Normal layer



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

Residual layer

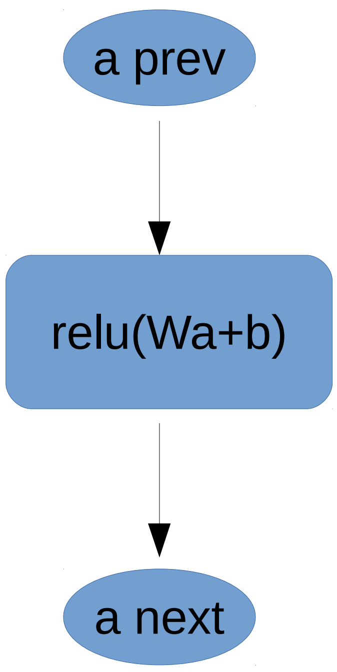


$$f_{w,b}(x) = relu(W \cdot a + b) + X$$

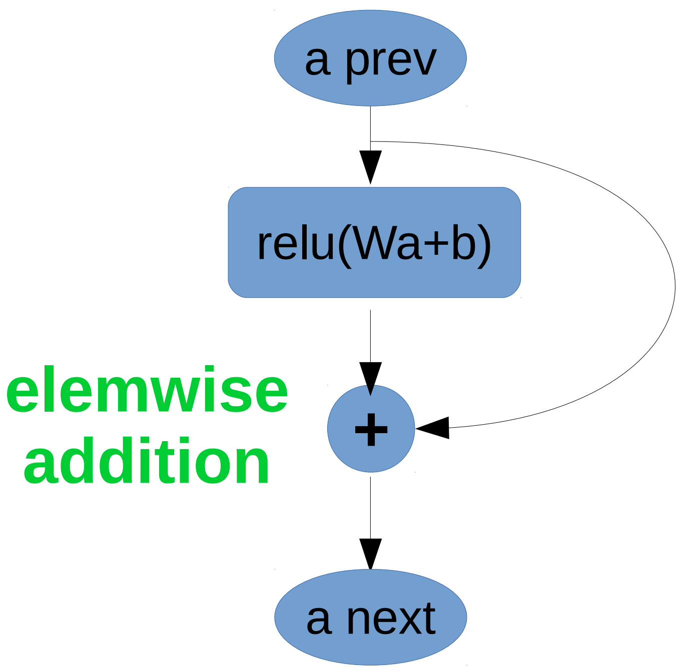
Architectures: residual network

Idea: let's create a shortcut for gradients

Normal layer



Residual layer



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

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

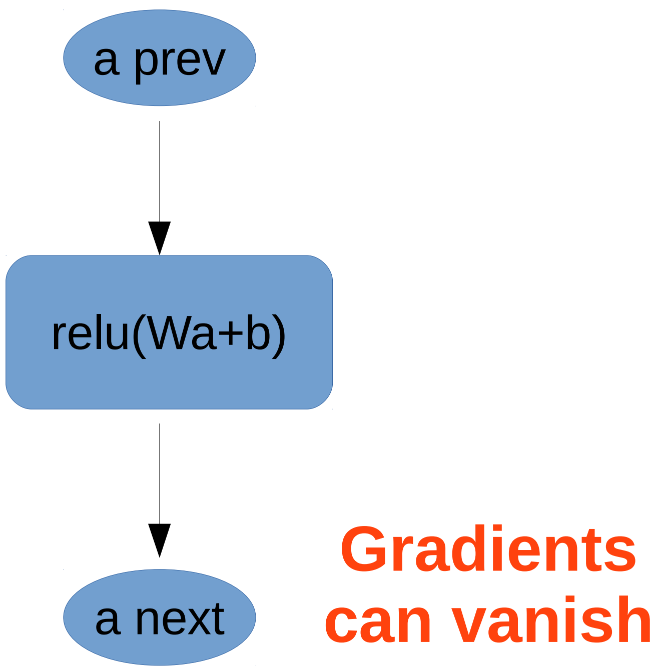
$$f_{w,b}(x) = relu(W \cdot a + b) + X$$

???

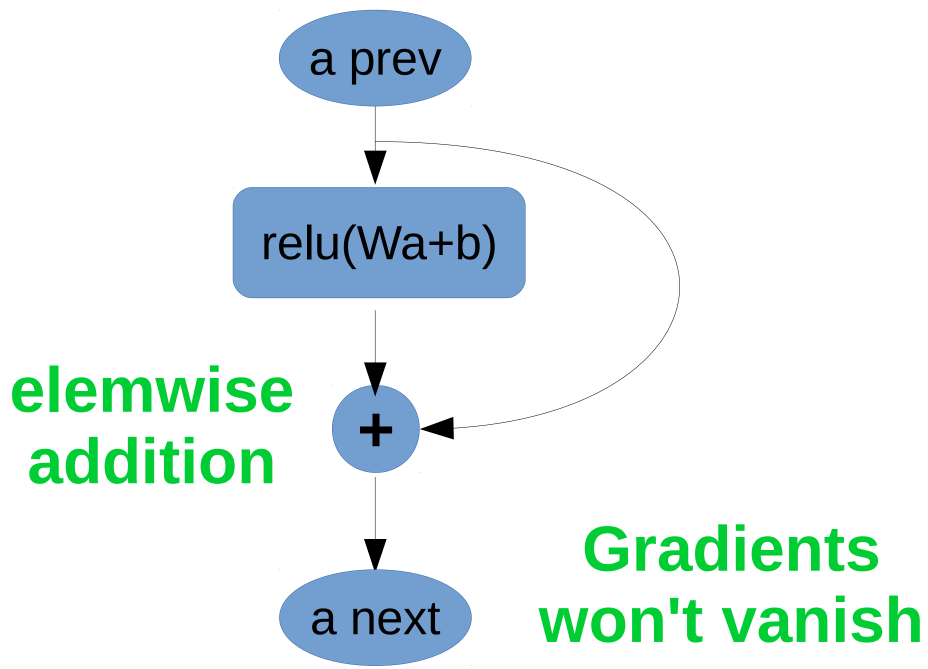
Architectures: residual network

Idea: let's create a shortcut for gradients

Normal layer



Residual layer



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

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

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

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

Visualizing loss surfaces

https://arxiv.org/abs/1712.09913

Visualize loss function in two random projections

$$f(\alpha, \beta) = L(\theta^* + \alpha\delta + \beta\eta)$$

Note: projection is scaled so that each “neuron” has the same norm

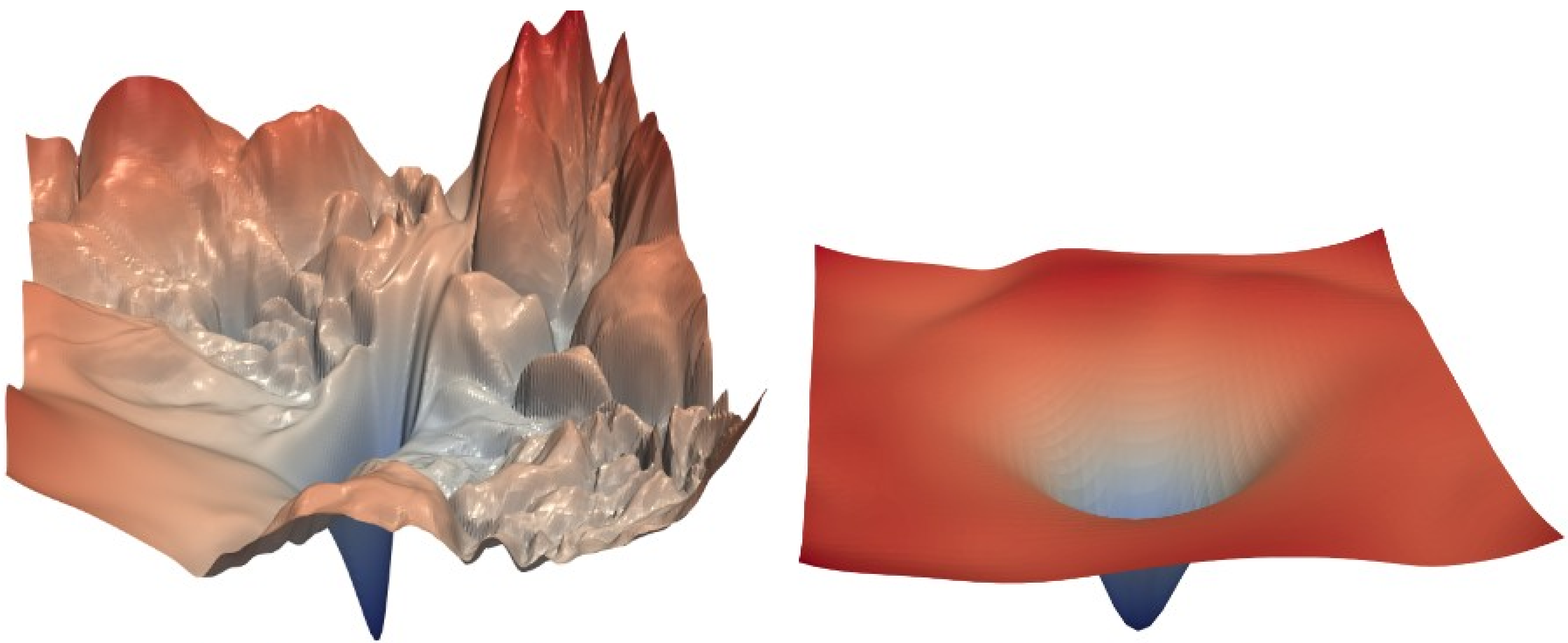
$$d_{i,j} \leftarrow \frac{d_{i,j}}{||d_{i,j}||}||\theta_{i,j}||$$

Visualizing loss surfaces

https://arxiv.org/abs/1712.09913

Visualize loss function in two random projections

$$f(\alpha, \beta) = L(\theta^* + \alpha\delta + \beta\eta)$$



(a) without skip connections

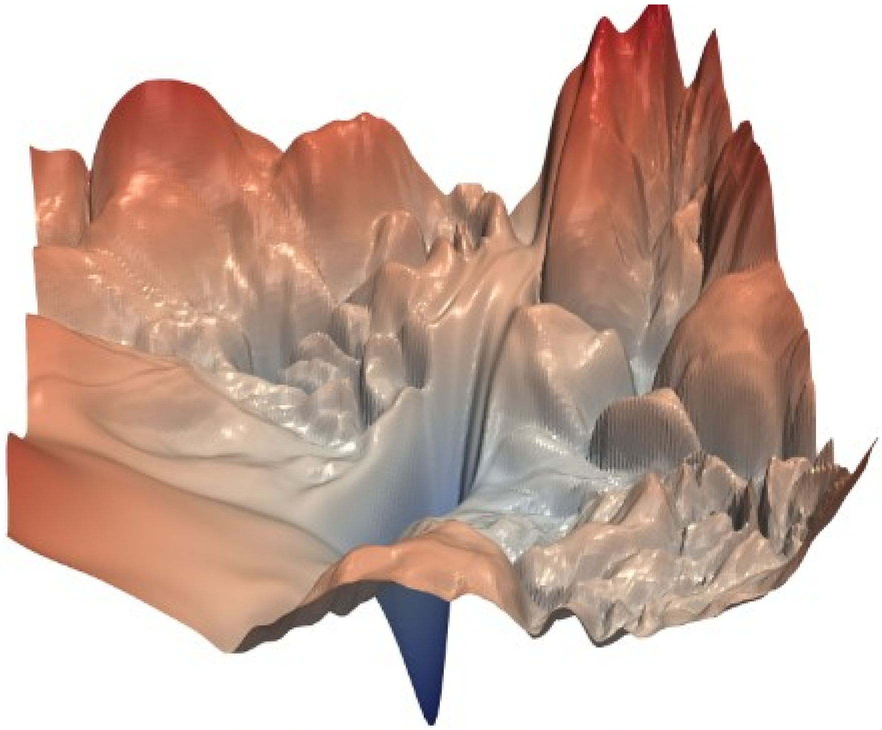
(b) with skip connections

Visualizing loss surfaces

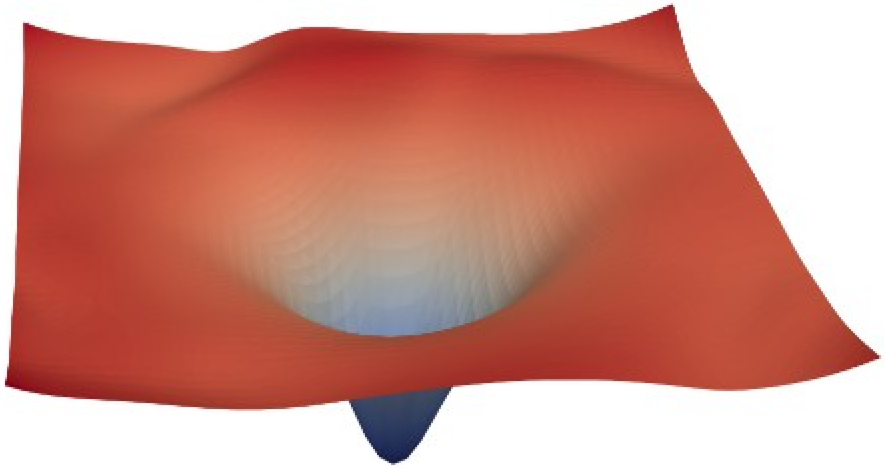
https://arxiv.org/abs/1712.09913

Visualize loss function in two random projections

$$f(\alpha, \beta) = L(\theta^* + \alpha\delta + \beta\eta)$$



(a) without skip connections



(b) with skip connections

Nuff

What did we learn?

Nuff

Coding time!