Deep Learning as an Engineer: The nuts and bolts and dirty tricks

Jan Schlüter
OFAI, Vienna, Austria

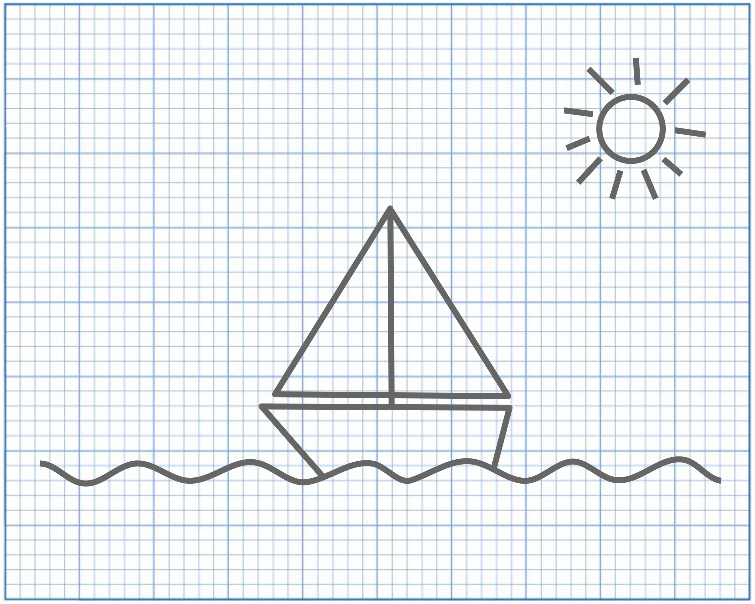
September 11, 2017

Outline

1. Application examples
2. Basic ideas behind deep learning
3. Deep learning in practice

Outline

1. Application examples
2. Basic ideas behind deep learning
3. Deep learning in practice

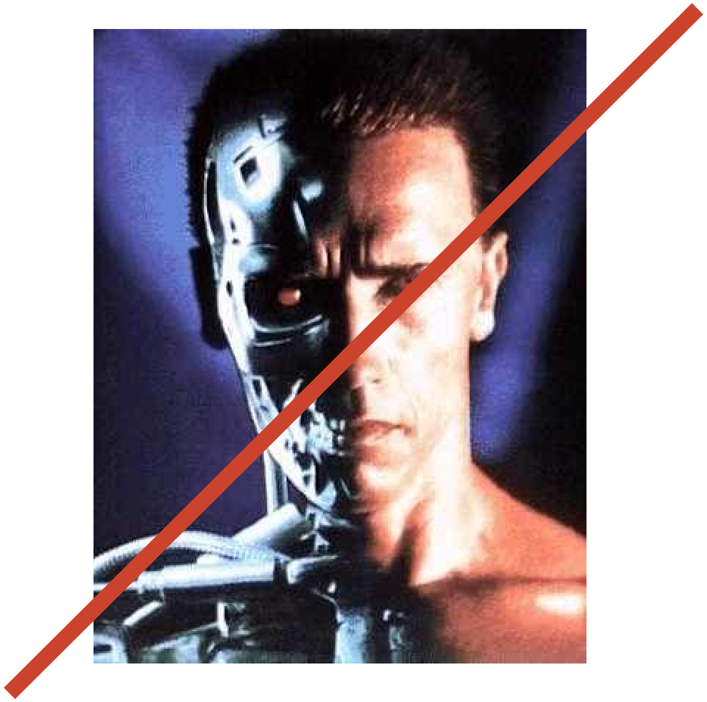


Outline

1. Application examples
2. Basic ideas behind deep learning
3. Deep learning in practice

Application examples

Application examples



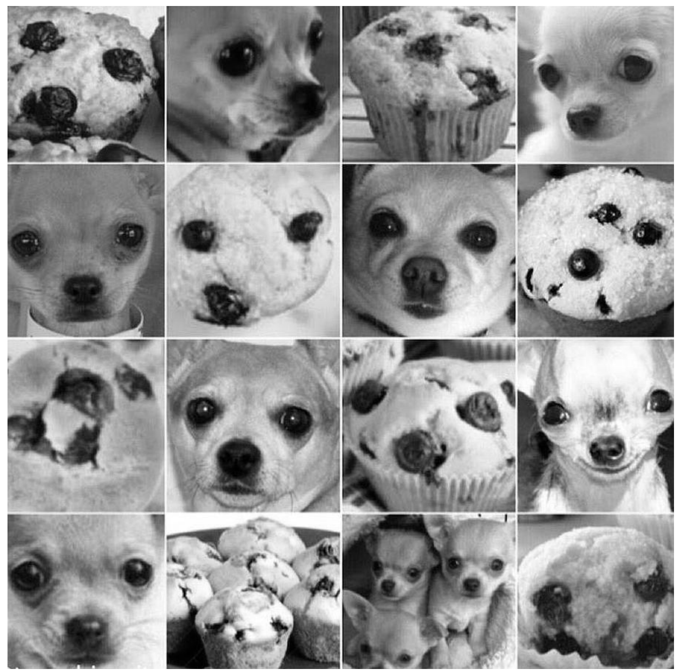
Nonlinear regression

Task: Predict at what force a concrete cylinder bursts, depending on component quantities and age

cement	... kg/m³
blast furnace slag	... kg/m³
fly ash	... kg/m³
water	... kg/m³
superplasticizer	... kg/m³
coarse aggregate	... kg/m³
fine aggregate	... kg/m³
age	... days
compressive strength	?? MPa

Binary image classification

Task: Distinguish grayscale photographs of chihuahuas and blueberry muffins



Categorical image classification

Task: Recognize hand-written digits

Task: Recognize photographed objects
(with a fixed set of possible answers)

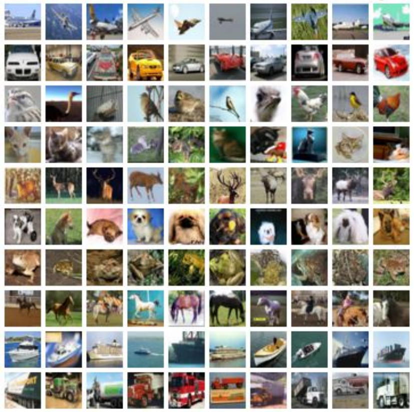
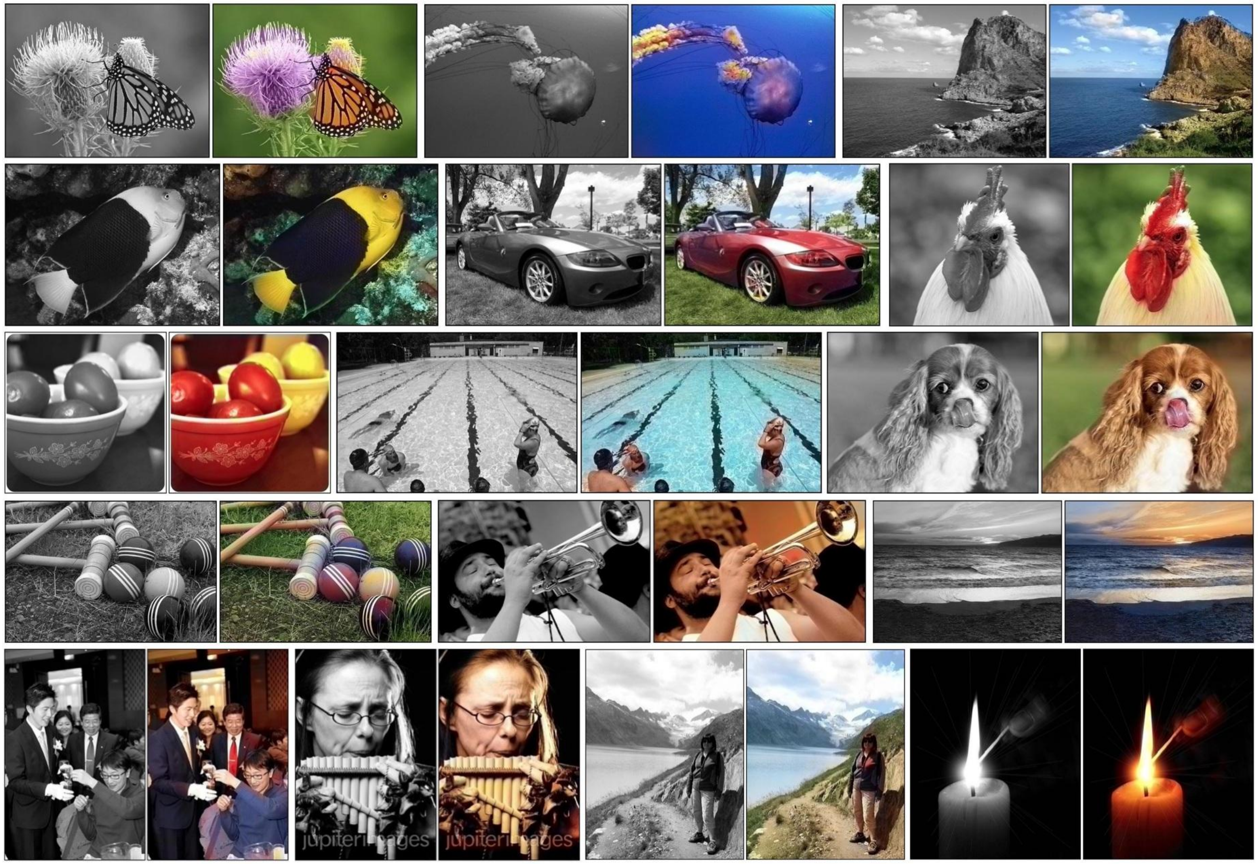


Image colorization

Task: Create colored image from grayscale image



Mar 2016: Colorful Image Colorization, http://arxiv.org/abs/1603.08511, http://richzhang.github.io/colorization/

Image generation

Task: Create colored image from scratch (possibly domain-specific)

Nov 2015: DCGANs, http://arxiv.org/abs/1511.06434, https://github.com/Newmu/dcgan_code

Text generation

Task: Create text from scratch (possibly domain-specific)

PANDARUS:
Alas, I think he shall be come approached and the day
When little srain would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.

Second Senator:
They are away this miseries, produced upon my soul,
Breaking and strongly should be buried, when I perish
The earth and thoughts of many states.

DUKE VINCENTIO:
Well, your wit is in the care of side and that.

May 2015: The Unreasonable Effectiveness of RNNs, http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Text generation

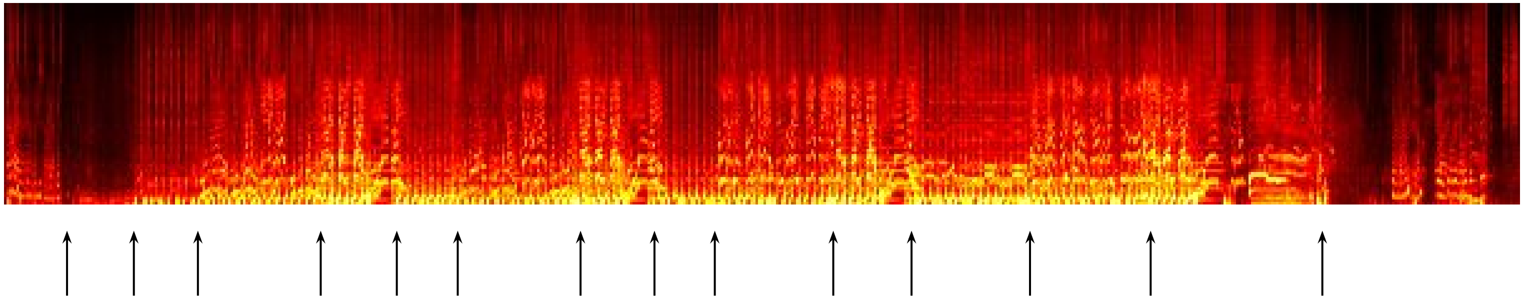
Task: Create text from scratch (possibly domain-specific)

```
static void do_command(struct seq_file *m, void *v)
{
  int column = 32 << (cmd[2] & 0x80);
  if (state)
    cmd = (int)(int_state ^ (in_8(&ch->ch_flags) & Cmd) ? 2 : 1);
  else
    seq = 1;
  for (i = 0; i < 16; i++) {
    if (k & (1 << 1))
      pipe = (in_use & UMXTHREAD_UNCCA) +
        ((count & 0x00000000fffffff8) & 0x000000f) << 8;
    if (count == 0)
      sub(pid, ppc_md.kexec_handle, 0x20000000);
    pipe_set_bytes(i, 0);
  }
  /* Free our user pages pointer to place camera if all dash */
  subsystem_info = &of_changes[PAGE_SIZE];
  rek_controls(offset, idx, &soffset);
  /* Now we want to deliberately put it to device */
  control_check_polarity(&context, val, 0);
  for (i = 0; i < COUNTER; i++)
    seq_puts(s, "policy ");
}
```

May 2015: The Unreasonable Effectiveness of RNNs, http://karpathy.github.io/2015/05/21/rnn-effectiveness/

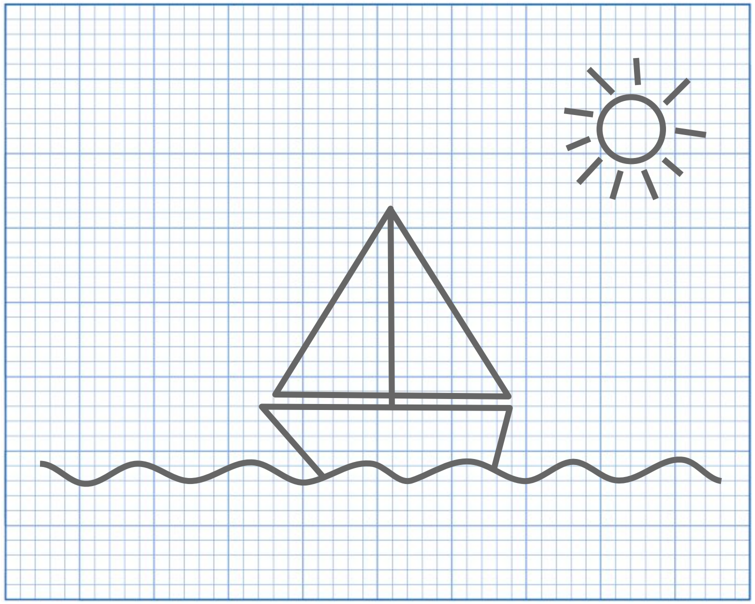
Acoustic event detection

Task: Detect boundaries between different parts of a music piece (e.g., verse → chorus)

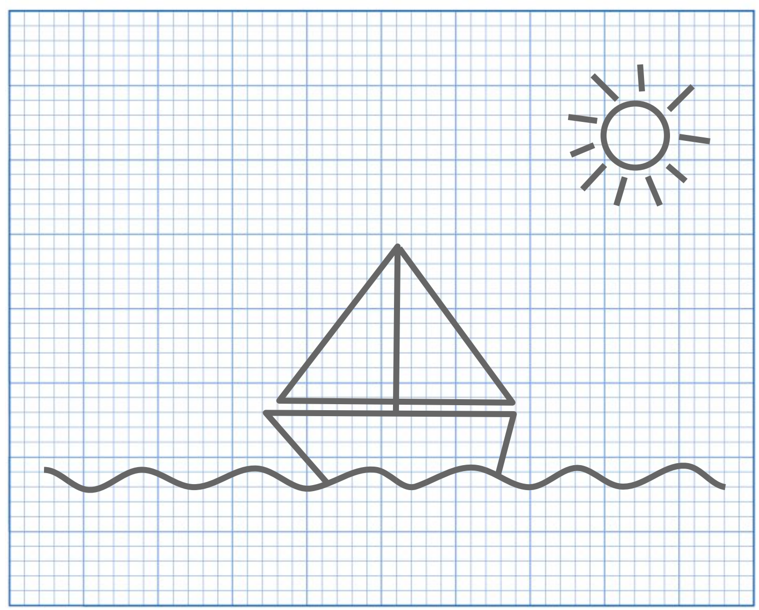


ISMIR 2014: Boundary detection in music structure analysis using convolutional neural networks

Basic ideas behind deep learning



Basic ideas behind ~~deep~~ machine learning



How to solve a task with machine learning

1. **Formalize task** so its solution can be expressed as a function
2. **Define model** as a generic solution with free parameters
3. **Define loss** function measuring how bad the solution is
4. **Optimize** model parameters to minimize loss

Formalize task: regression

Task: Predict at what force a concrete cylinder bursts, depending on component quantities and age

Solution form: y = f(**x**)

Input x: 8-dimensional vector

Output y: scalar

cement	... kg/m³
blast furnace slag	... kg/m³
fly ash	... kg/m³
water	... kg/m³
superplasticizer	... kg/m³
coarse aggregate	... kg/m³
fine aggregate	... kg/m³
age	... days
compressive strength	?? MPa

$\mathbf{x} \in \mathbb{R}^8$

$y \in \mathbb{R}$

Formalize task: binary image classification

Task: Distinguish grayscale photographs of chihuahuas and blueberry muffins

Solution form: y = f($\mathbf{X}$)
Input X: matrix of gray values
Output y: scalar "muffinness"

"0.0"

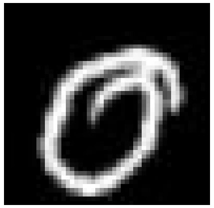
$\mathbf{X} \in [0,1]^{236\times236}$

$y \in [0,1]$

Formalize task: categorical image classification

Task: Recognize hand-written digits

Solution form: $\mathbf{y} = f(\mathbf{X})$
Input X: matrix of gray values
Output y: vector of class probabilities



(1,0,0,0, ... 0)

$\mathbf{X} \in [0,1]^{28\times28}$

$\mathbf{y} \in [0,1]^{10}$; $\sum_i y_i = 1.0$

Task: Recognize photographed objects
(with a fixed set of possible answers)

Solution form: $\mathbf{y} = f(\mathbf{X})$
Input X: 3-tensor of RGB values
Output y: vector of class probabilities

(0,0,1,0, ... 0)

$\mathbf{X} \in [0,1]^{3\times32\times32}$

$\mathbf{y} \in [0,1]^{10}$; $\sum_i y_i = 1.0$

Formalize task: image colorization

Task: Create colored image from grayscale image

Solution form: Y = f(**X**)
Input X: matrix of gray values
Output Y: 3-tensor of RGB values

$\mathbf{X} \in [0,1]^{h \times w}$

$\mathbf{Y} \in [0,1]^{3 \times h \times w}$

Mar 2016: Colorful Image Colorization, http://arxiv.org/abs/1603.08511, http://richzhang.github.io/colorization/

Formalize task: image generation

Task: Create colored image from scratch (possibly domain-specific)

Solution form: $\mathbf{Y} = f(\mathbf{x})$

Input x: vector of random values

Output Y: 3-tensor of RGB values

(0.392, -0.124, ...) $\mathbf{x} \in \mathbb{R}^{100}$

$\mathbf{Y} \in [0,1]^{3\times128\times128}$

Nov 2015: DCGANs, http://arxiv.org/abs/1511.06434, https://github.com/Newmu/dcgan_code

Formalize task: text generation

Task: Create text from scratch (possibly domain-specific)

Solution form: y, h' = f(**x, h**)
Input x: vector encoding of seed or previously emitted character
Input h: vector of initial or previously emitted internal state
Output y: vector of next character probabilities
Output h': vector of next internal state

May 2015: The Unreasonable Effectiveness of RNNs, http://karpathy.github.io/2015/05/21/rnn-effectiveness/

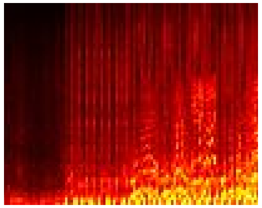
Formalize task: acoustic event detection

Task: Detect boundaries between different parts of a music piece (e.g., verse → chorus)

Solution form: $y = f(\mathbf{X})$

Input X: magnitude spectrogram excerpt

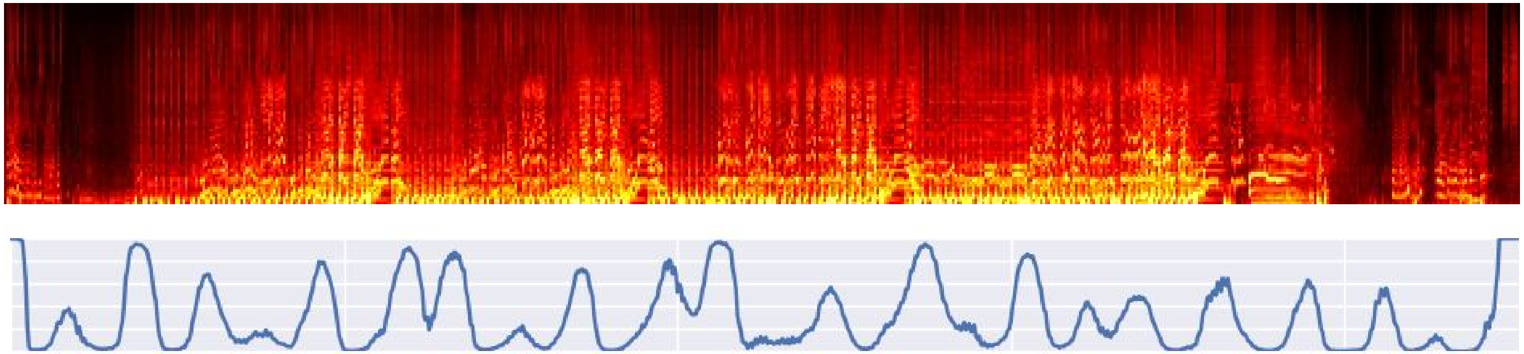
Output y: scalar “boundariness” of excerpt center



$\mathbf{X} \in \mathbb{R}^{115 \times 80}$

“1.0” $y \in [0,1]$

Prediction process: apply f(X) to overlapping excerpts, pick peaks



ISMIR 2014: Boundary detection in music structure analysis using convolutional neural networks

How to solve a task with machine learning

1. **Formalize task** so its solution can be expressed as a function
2. **Define model** as a generic solution with free parameters
3. **Define loss** function measuring how bad the solution is
4. **Optimize** model parameters to minimize loss

$$\mathbf{Y} = f(\mathbf{X})$$

How to solve a task with machine learning

1. **Formalize task** so its solution can be expressed as a function
2. **Define model** as a generic solution with free parameters
3. **Define loss** function measuring how bad the solution is
4. **Optimize** model parameters to minimize loss

$$\mathbf{Y} = f(\mathbf{X}; \theta)$$

How to solve a task with machine learning

1. **Formalize task** so its solution can be expressed as a function
2. **Define model** as a generic solution with free parameters
3. **Define loss** function measuring how bad the solution is
4. **Optimize** model parameters to minimize loss

$$\mathbf{Y} = f(\mathbf{X}; \theta)$$

$$l = L(\theta; f)$$

How to solve a task with machine learning

1. **Formalize task** so its solution can be expressed as a function
2. **Define model** as a generic solution with free parameters
3. **Define loss** function measuring how bad the solution is
4. **Optimize** model parameters to minimize loss

$$\mathbf{Y} = f(\mathbf{X}; \theta)$$

$$l = L(\theta; f, D)$$

How to solve a task with machine learning

1. **Formalize task** so its solution can be expressed as a function
2. **Define model** as a generic solution with free parameters
3. **Define loss** function measuring how bad the solution is
4. **Optimize** model parameters to minimize loss

$$\mathbf{Y} = f(\mathbf{X}; \theta)$$

$$l = L(\theta; f, D) = \mathbf{\Sigma}_{(\mathbf{X}, \mathbf{T}) \in D} J(f(\mathbf{X}; \theta), \mathbf{T})$$

How to solve a task with machine learning

1. **Formalize task** so its solution can be expressed as a function
2. **Define model** as a generic solution with free parameters
3. **Define loss** function measuring how bad the solution is
4. **Optimize** model parameters to minimize loss

$$\mathbf{Y} = f(\mathbf{X}; \theta)$$

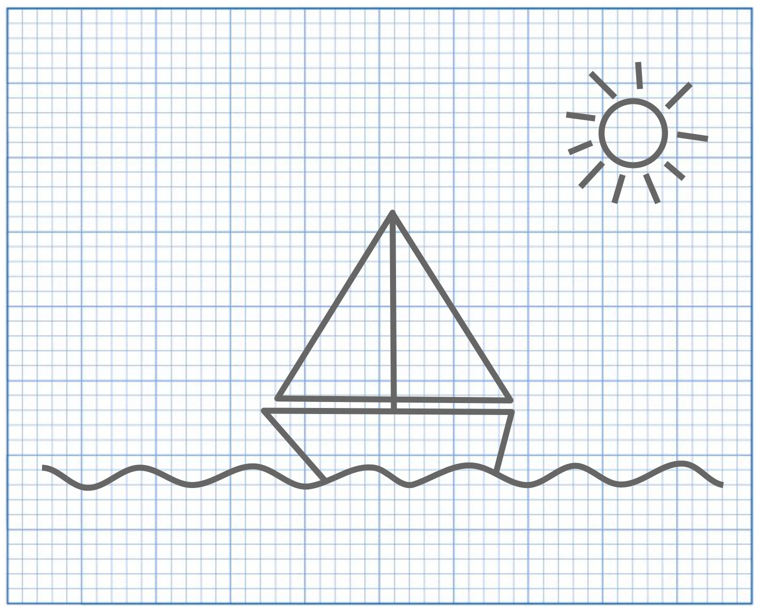
$$l = L(\theta; f, D) = \mathbf{\Sigma}_{(\mathbf{X}, \mathbf{T}) \in D} J(f(\mathbf{X}; \theta), \mathbf{T})$$

$$\theta^* = \min_\theta L(\theta; f, D)$$

deep

~~machine~~

Basic ideas behind ~~deep~~ learning



How to solve a task with deep learning

1. **Formalize task** so its solution can be expressed as a function
2. **Define model** as a generic solution with free parameters
3. **Define loss** function measuring how bad the solution is
4. **Optimize** model parameters to minimize loss

$$\mathbf{Y} = f(\mathbf{X}; \theta)$$

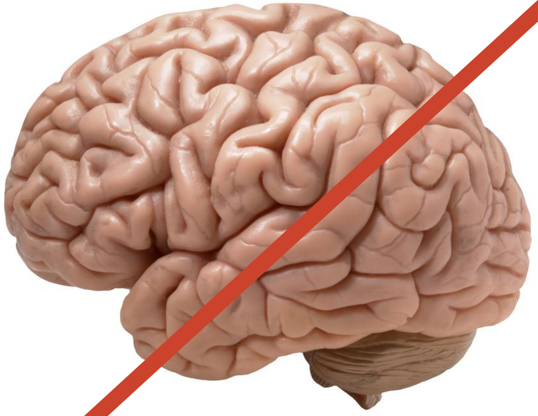
↗

Design choice: make f *deep* (= a composition of multiple nonlinear functions), often an artificial neural network

What are Artificial Neural Networks?

“a simulation of a small brain”

What are Artificial Neural Networks?



“a simulation of a small brain”

What are Artificial Neural Networks?

a fancy name for a family of functions, including:

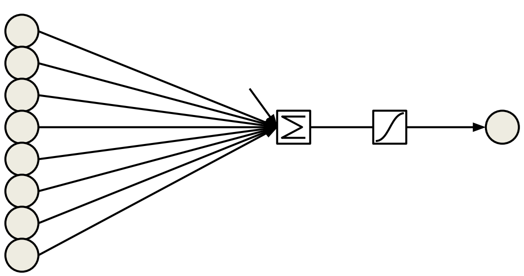
$$y = \sigma(b + \mathbf{w}^T \mathbf{x})$$ (equivalent to logistic regression)

What are Artificial Neural Networks?

a fancy name for a family of functions, including:

$$y = \sigma(b + \mathbf{w}^T\mathbf{x}) \qquad \text{(equivalent to logistic regression)}$$

expression can be visualized as a graph:

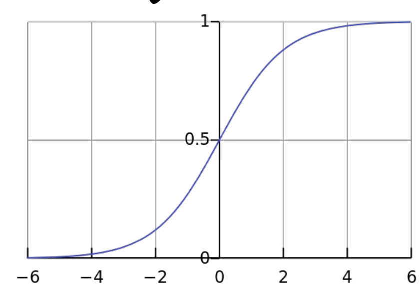


$\mathbf{x}$ $\quad b + \mathbf{w}^T\mathbf{x}$ $\quad y$

Output value is computed as a **weighted sum of its inputs,**

$$b + \mathbf{w}^T\mathbf{x} = b + \Sigma_i w_i x_i$$

followed by a nonlinear function.

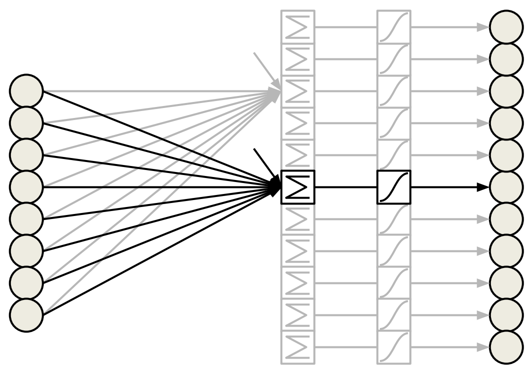


What are Artificial Neural Networks?

a fancy name for a family of functions, including:

$\mathbf{y} = \sigma(\mathbf{b} + \mathbf{W}^T\mathbf{x})$ (multiple logistic regressions)

expression can be visualized as a graph:

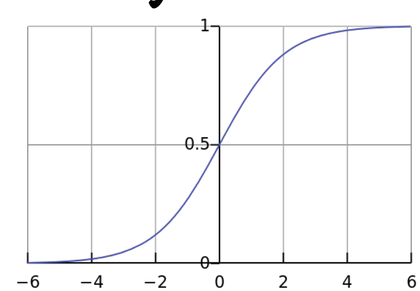


$\mathbf{x}$ $\mathbf{b} + \mathbf{W}^T\mathbf{x}$ $\mathbf{y}$

Output values are computed as **weighted sums of their inputs,**

$$\mathbf{b} + \mathbf{W}^T\mathbf{x} = b_j + \Sigma_i w_{ij} x_i$$

followed by a nonlinear function.

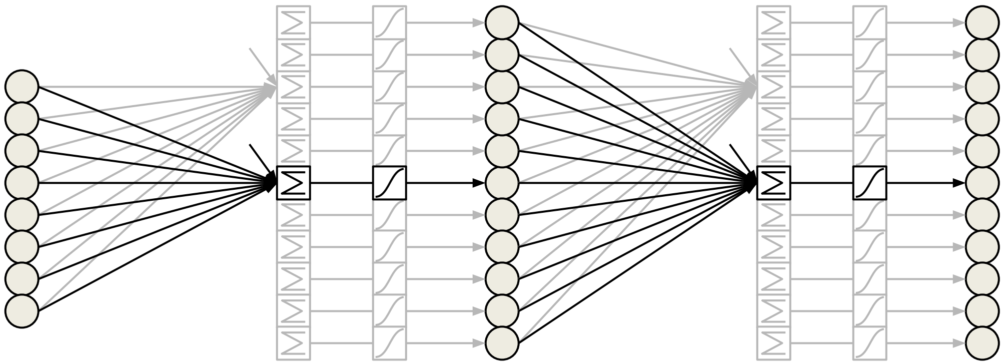


What are Artificial Neural Networks?

a fancy name for a family of functions, including:

$$\mathbf{y} = \sigma(\mathbf{b}_2 + \mathbf{W}_2{}^T \sigma(\mathbf{b}_1 + \mathbf{W}_1{}^T \mathbf{x})) \quad \text{(stacked logistic regressions)}$$

expression can be visualized as a graph:



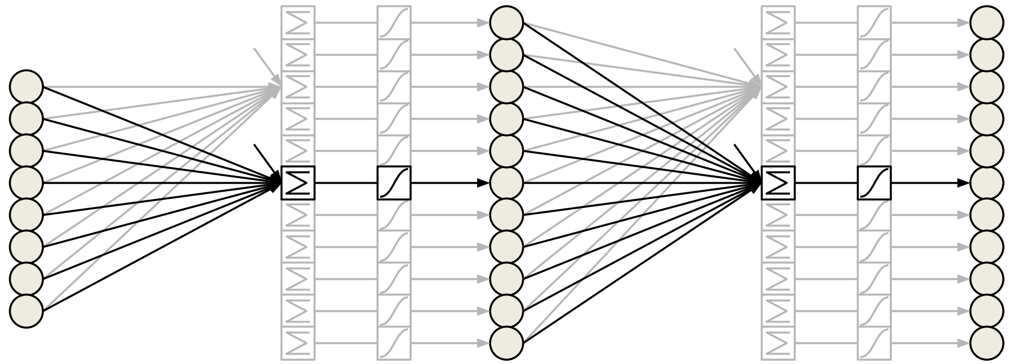
$\mathbf{x}$ $\quad$ $\mathbf{b}_1 + \mathbf{W}_1{}^T\mathbf{x}$ $\quad$ $\mathbf{h}$ $\quad$ $\mathbf{b}_2 + \mathbf{W}_2{}^T\mathbf{h}$ $\quad$ $\mathbf{y}$

What are Artificial Neural Networks?

a fancy name for a family of functions, including:

$$\mathbf{y} = \sigma(\mathbf{b}_2 + \mathbf{W}_2^T \sigma(\mathbf{b}_1 + \mathbf{W}_1^T \mathbf{x})) \qquad \text{(stacked logistic regressions)}$$

expression can be visualized as a graph:



$\mathbf{x}$ $\quad \mathbf{b}_1 + \mathbf{W}_1^T\mathbf{x}$ $\quad \mathbf{h}$ $\quad \mathbf{b}_2 + \mathbf{W}_2^T\mathbf{h}$ $\quad \mathbf{y}$

Universal Approximation Theorem:
This can model any continuous function from $\mathbb{R}^n$ to $\mathbb{R}^m$ arbitrarily well (if $\mathbf{h}$ is made large enough).

Interlude: Why go any deeper than two layers?

A neural network with a single hidden layer of enough units can approximate any continuous function arbitrarily well. In other words, it can solve whatever problem you're interested in!
(Cybenko 1998, Hornik 1991)

But:

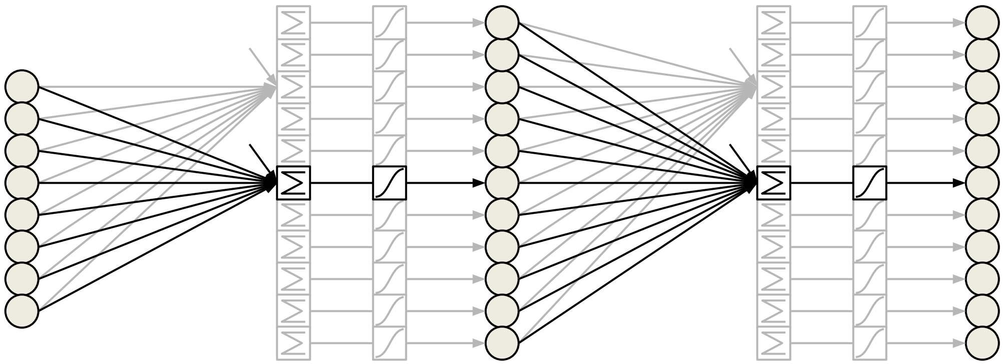
- "Enough units" can be a very large number. There are functions representable with a small, but deep network that would require exponentially many units with a single layer.
  (e.g., Hastad et al. 1986, Bengio & Delalleau 2011)
- The proof only says that a shallow network exists, it does not say how to find it. Evidence indicates that it is easier to train a deep network to perform well than a shallow one.

What are Artificial Neural Networks?

a fancy name for a family of functions, including:

$$\mathbf{y} = \sigma(\mathbf{b}_2 + \mathbf{W}_2^T \sigma(\mathbf{b}_1 + \mathbf{W}_1^T \mathbf{x})) \qquad \text{(stacked logistic regressions)}$$

expression can be visualized as a graph:



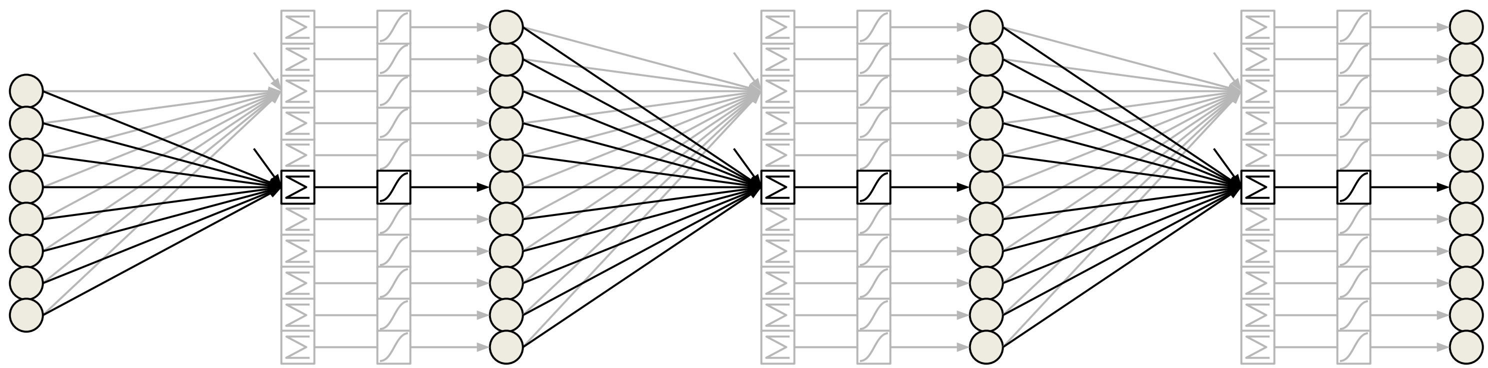
$\mathbf{x}$ $\quad$ $\mathbf{b}_1 + \mathbf{W}_1^T\mathbf{x}$ $\quad$ $\mathbf{h}$ $\quad$ $\mathbf{b}_2 + \mathbf{W}_2^T\mathbf{h}$ $\quad$ $\mathbf{y}$

What are Artificial Neural Networks?

a fancy name for a family of functions, including:

$$\mathbf{y} = \sigma(\mathbf{b}_3 + \mathbf{W}_3^T\sigma(\mathbf{b}_2 + \mathbf{W}_2^T\sigma(\mathbf{b}_1 + \mathbf{W}_1^T\mathbf{x})))$$

expression can be visualized as a graph:



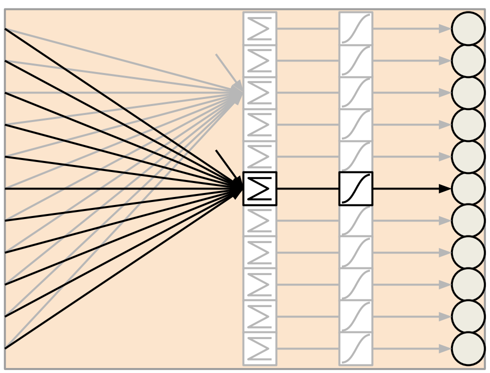
$\mathbf{x}$ $\quad$ $\mathbf{b}_1 + \mathbf{W}_1^T\mathbf{x}$ $\quad$ $\mathbf{h}_1$ $\quad$ $\mathbf{b}_2 + \mathbf{W}_2^T\mathbf{h}_1$ $\quad$ $\mathbf{h}_2$ $\quad$ $\mathbf{b}_3 + \mathbf{W}_3^T\mathbf{h}_2$ $\quad$ $\mathbf{y}$

What are Artificial Neural Networks?

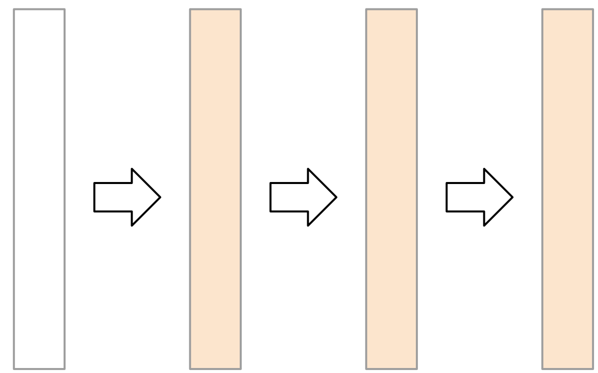
a fancy name for a family of functions, including:

$$f_{\mathbf{W},\mathbf{b}}(\mathbf{x}) = \sigma(\mathbf{b} + \mathbf{W}^T\mathbf{x}) \qquad \mathbf{y} = (f_{W_3,b_3} \circ f_{W_2,b_2} \circ f_{W_1,b_1})(\mathbf{x})$$

expression can be visualized as a graph:



“dense layer”



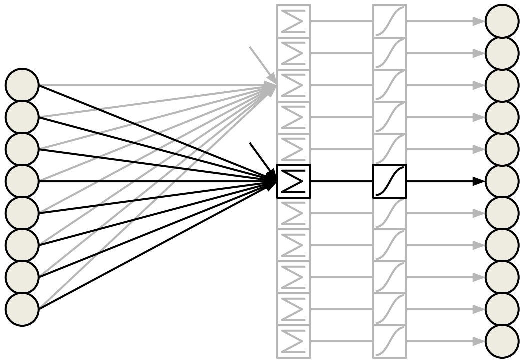
$\mathbf{x}$ $\mathbf{h}_1$ $\mathbf{h}_2$ $\mathbf{y}$

composed of simpler functions, commonly termed “layers”

Why dense layers are great

Fully-connected layer:

Each **input** is a **scalar** value, each **weight** is a **scalar** value, each output is the sum of all inputs **multiplied** by weights.



Consequence: Swapping two inputs does not change the task. We can just swap the weights as well. (Or retrain the network.)

Example task:
Distinguish *iris setosa*, *iris versicolour* and *iris virginica*

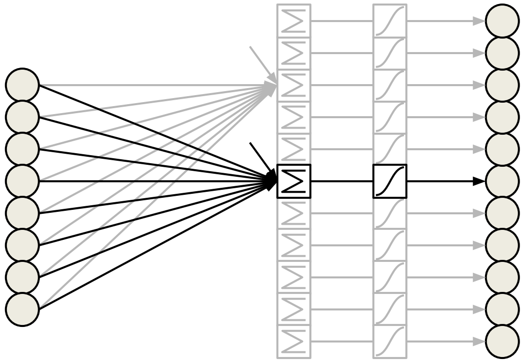
Input: (sepal length, sepal width, petal length, petal width)

Equivalent: (sepal width, petal length, sepal length, petal width)

Why dense layers are great

Fully-connected layer:

Each **input** is a **scalar** value, each **weight** is a **scalar** value, each output is the sum of all inputs **multiplied** by weights.



Same for the targets!

Consequence: Swapping two inputs does not change the task. We can just swap the weights as well. (Or retrain the network.)

Example task:
Distinguish *iris setosa*, *iris versicolour* and *iris virginica*

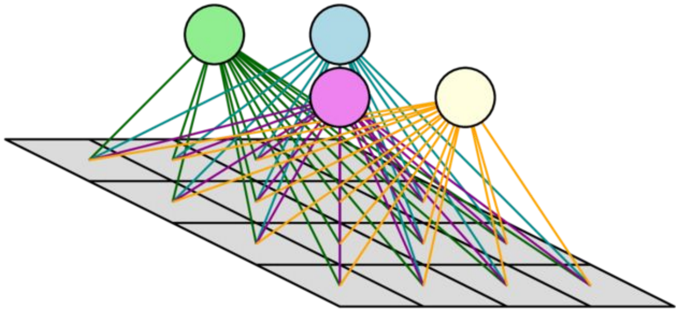
Input: (sepal length, sepal width, petal length, petal width)

Equivalent: (sepal width, petal length, sepal length, petal width)

Why dense layers are great

Fully-connected layer:

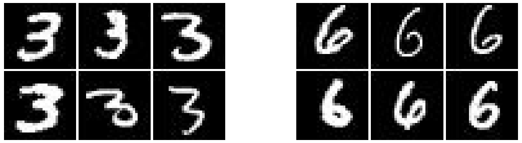
Each **input** is a **scalar** value, each **weight** is a **scalar** value, each output is the sum of all inputs **multiplied** by weights.



Consequence: Swapping two inputs does not change the task. We can just swap the weights as well. (Or retrain the network.)

Example task:
Distinguish 3 and 6

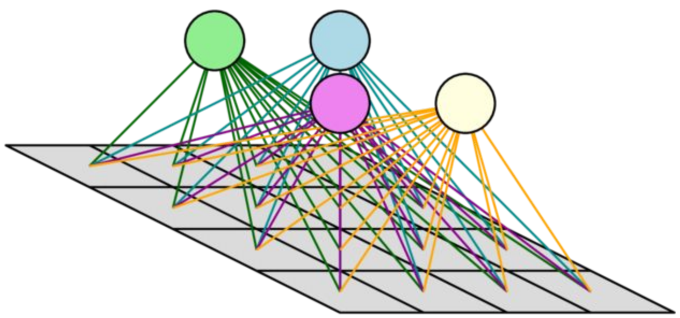
Input:



Why dense layers are ~~great~~ not so great

Fully-connected layer:

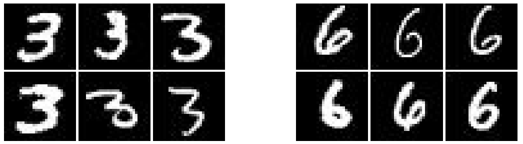
Each **input** is a **scalar** value, each **weight** is a **scalar** value, each output is the sum of all inputs **multiplied** by weights.



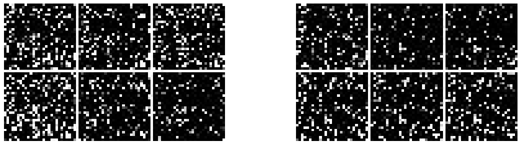
Consequence: Swapping two inputs does not change the task. We can just swap the weights as well. (Or retrain the network.)

Example task:
Distinguish 3 and 6

Input:



Equivalent:



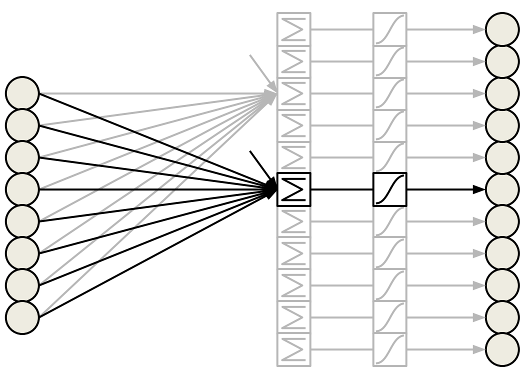
Convolutional layers

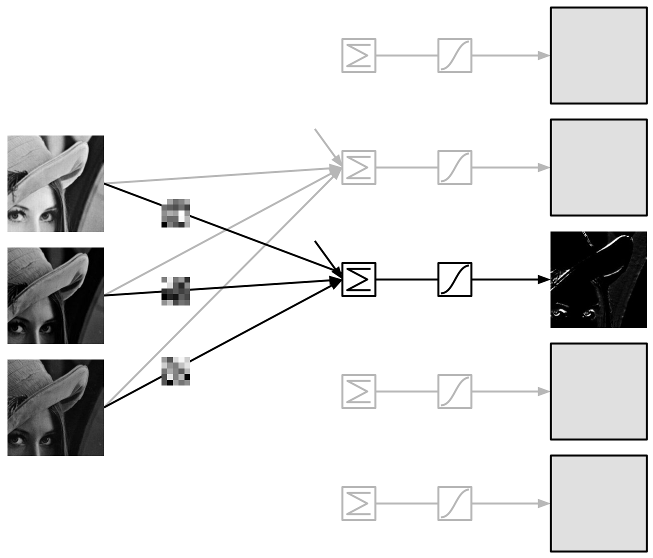
Fully-connected layer:

Each **input** is a **scalar** value,
each **weight** is a **scalar** value,
each output is the sum of
inputs **multiplied** by weights.

Convolutional layer:

Each **input** is a **tensor** (e.g., 2D),
each **weight** is a **tensor**,
each output is the sum of
inputs **convolved** by weights.

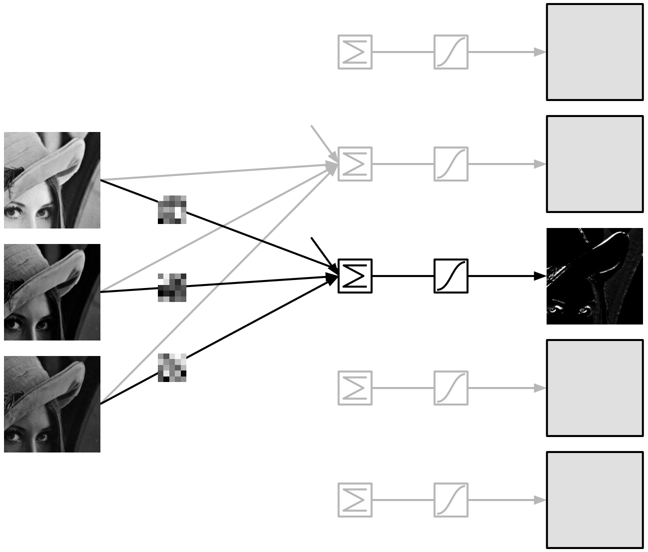




Why convolutional layers are great

Convolutional layer:

Each **input** is a **tensor**,
each **weight** is a **tensor**,
each output is the sum of
inputs **convolved** by weights.



Consequences:

- Input permutation does make a difference now
- Output retains the spatial layout of the input
- Can process large images with few learnable weights
- Weights are required to be applicable at every position

Pooling layers

A **pooling layer** downsamples a tensor.

Max pooling: keep the largest values of local patches

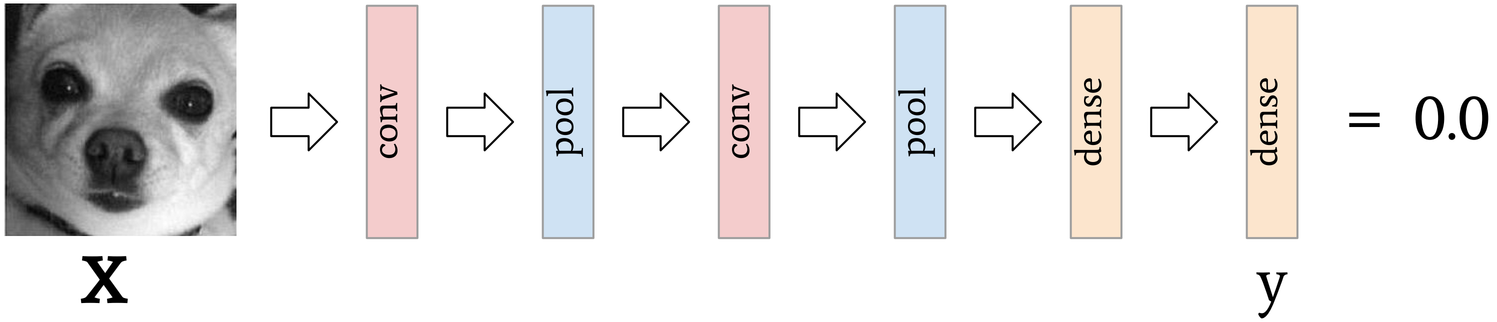
1	**3**	**5**	-3
-2	**3**	2	-1
1	2	-2	**2**
-1	**3**	-2	1

⇨

3	5
3	2

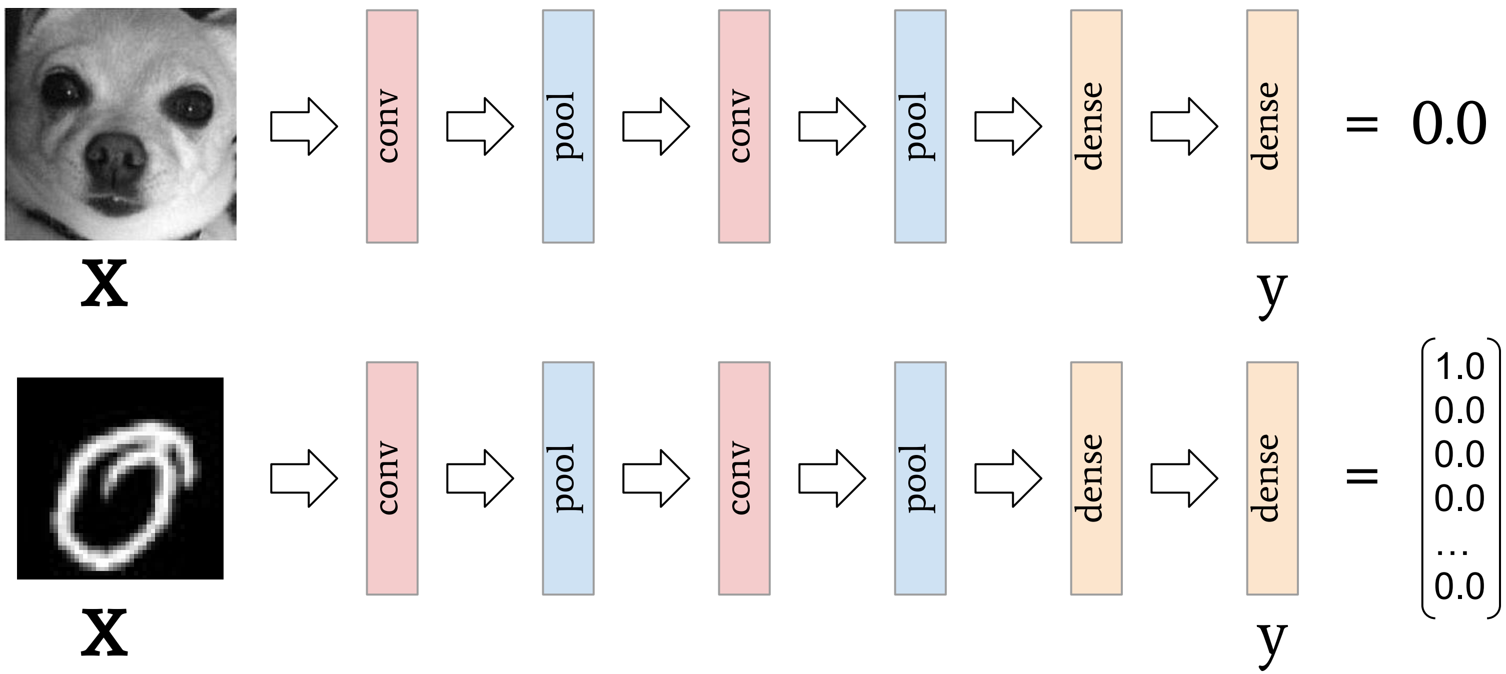
Average pooling: keep the mean values of local patches

Traditional Convolutional Neural Network

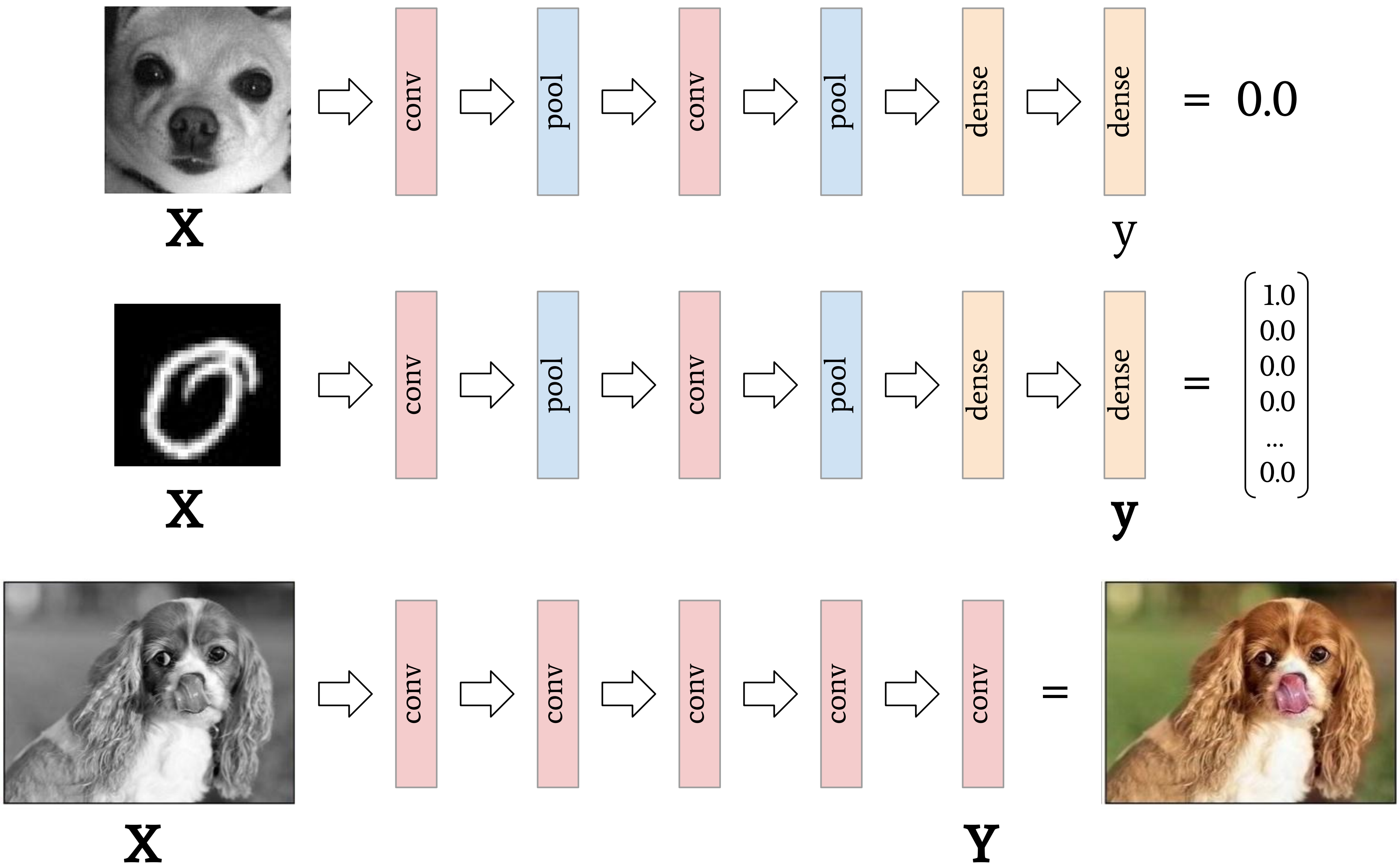


- **Convolutional layers**: local feature extraction
- **Pooling layers**: some translation invariance, data reduction
- **Fully-connected layers**: integrate information over full input

Traditional Convolutional Neural Network



Traditional Convolutional Neural Network



How to solve a task with deep learning

1. **Formalize task** so its solution can be expressed as a function
2. **Define model** as a generic solution with free parameters
3. **Define loss** function measuring how bad the solution is
4. **Optimize** model parameters to minimize loss

$$\mathbf{Y} = f(\mathbf{X}; \theta)$$

↗

Design choice: make f *deep* (= a composition of multiple nonlinear functions), often an artificial neural network

How to solve a task with deep learning

1. **Formalize task** so its solution can be expressed as a function
2. **Define model** as a generic solution with free parameters
3. **Define loss** function measuring how bad the solution is
4. **Optimize** model parameters to minimize loss

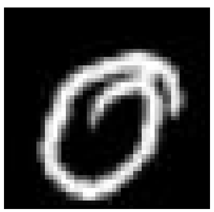
$$\mathbf{Y} = f(\mathbf{X}; \theta)$$

$$l = L(\theta; f, D) = \mathbf{\Sigma}_{(\mathbf{X}, \mathbf{T}) \in D} J(f(\mathbf{X}; \theta), \mathbf{T})$$

Penalty functions

$y = 0.21$ $t = 0.0$ $J(y, t) = -\log(y) \cdot t - \log(1-y) \cdot (1-t)$

"binary cross-entropy"



$\mathbf{y} = \begin{pmatrix} 0.6 \\ 0.0 \\ 0.1 \\ 0.0 \\ \dots \\ 0.1 \end{pmatrix}$ $\mathbf{t} = \begin{pmatrix} 1.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ \dots \\ 0.0 \end{pmatrix}$ $J(\mathbf{y}, \mathbf{t}) = -\boldsymbol{\Sigma}_i \log(y_i) \cdot t_i$

"categorical cross-entropy"

$\mathbf{Y} =$

$\mathbf{T} =$

$J(\mathbf{Y}, \mathbf{T}) = 0.5 \cdot \boldsymbol{\Sigma}_{i,j,k}(Y_{i,j,k} - T_{i,j,k})^2$

"squared error"

How to solve a task with deep learning

1. **Formalize task** so its solution can be expressed as a function
2. **Define model** as a generic solution with free parameters
3. **Define loss** function measuring how bad the solution is
4. **Optimize** model parameters to minimize loss

$$\mathbf{Y} = f(\mathbf{X}; \theta)$$

$$l = L(\theta; f, D) = \boldsymbol{\Sigma}_{(\mathbf{X}, \mathbf{T}) \in D} J(f(\mathbf{X}; \theta), \mathbf{T})$$

$$\theta^* = \min_\theta L(\theta; f, D)$$

Optimization

4. **Optimize** model parameters to minimize loss

$$\theta^* = \min_\theta L(\theta; f, D)$$

Iterative scheme:

0. initialize θ randomly
1. find direction in which L decreases
2. move θ a bit into that direction
3. go to step 1

Optimization

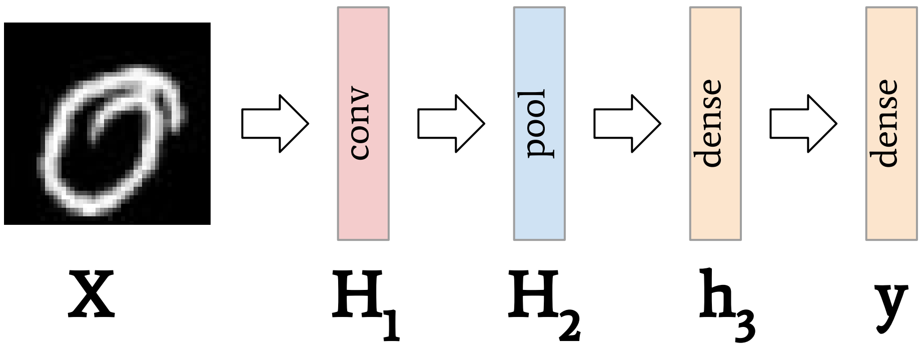
4. **Optimize** model parameters to minimize loss

$$\theta^* = \min_\theta L(\theta; f, D)$$

Iterative scheme:

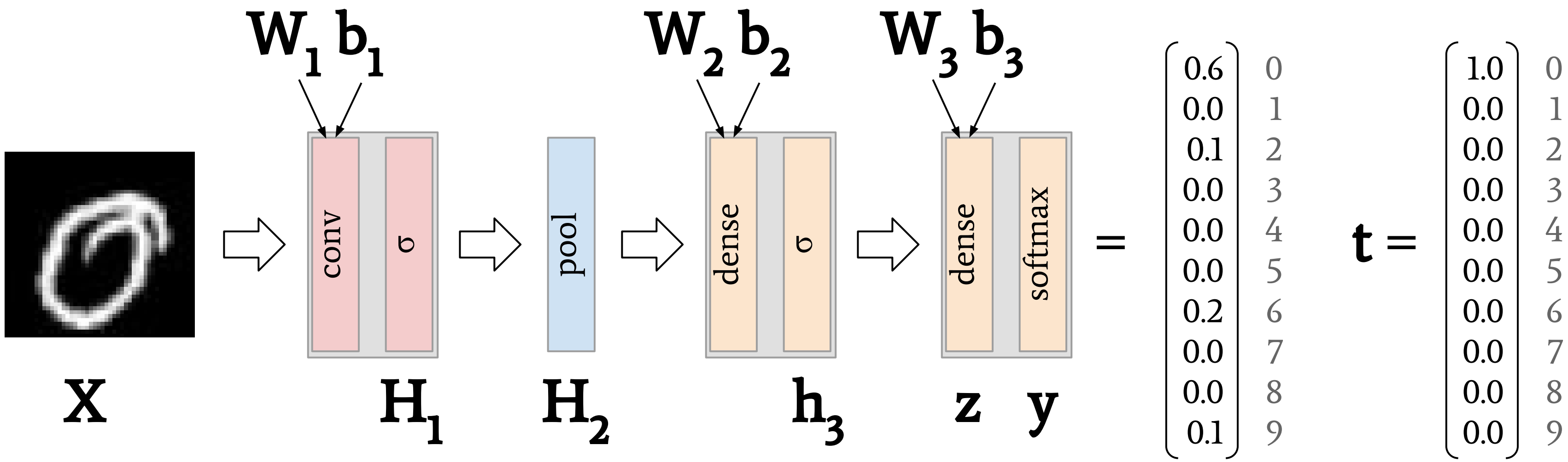
0. initialize θ randomly
1. **find direction in which L decreases**
2. move θ a bit into that direction
3. go to step 1

Find direction in which the loss decreases

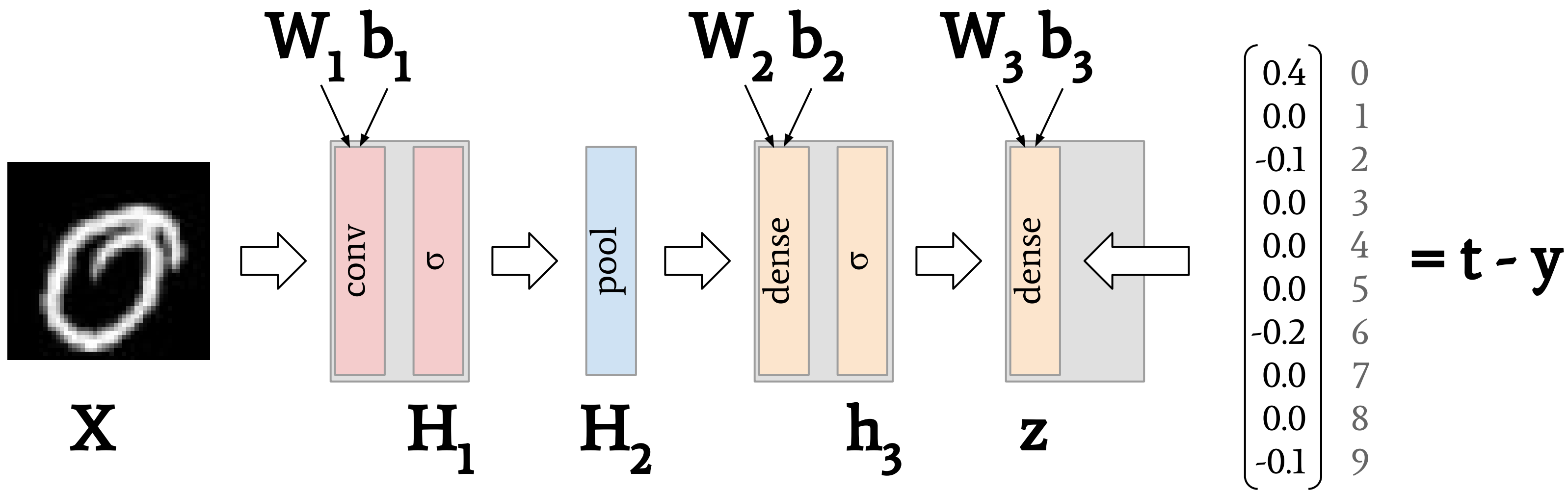


$$\mathbf{y} = \begin{pmatrix} 0.6 \\ 0.0 \\ 0.1 \\ 0.0 \\ 0.0 \\ 0.0 \\ 0.2 \\ 0.0 \\ 0.0 \\ 0.1 \end{pmatrix} \begin{matrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \end{matrix} \qquad \mathbf{t} = \begin{pmatrix} 1.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \end{pmatrix} \begin{matrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \end{matrix}$$

Find direction in which the loss decreases

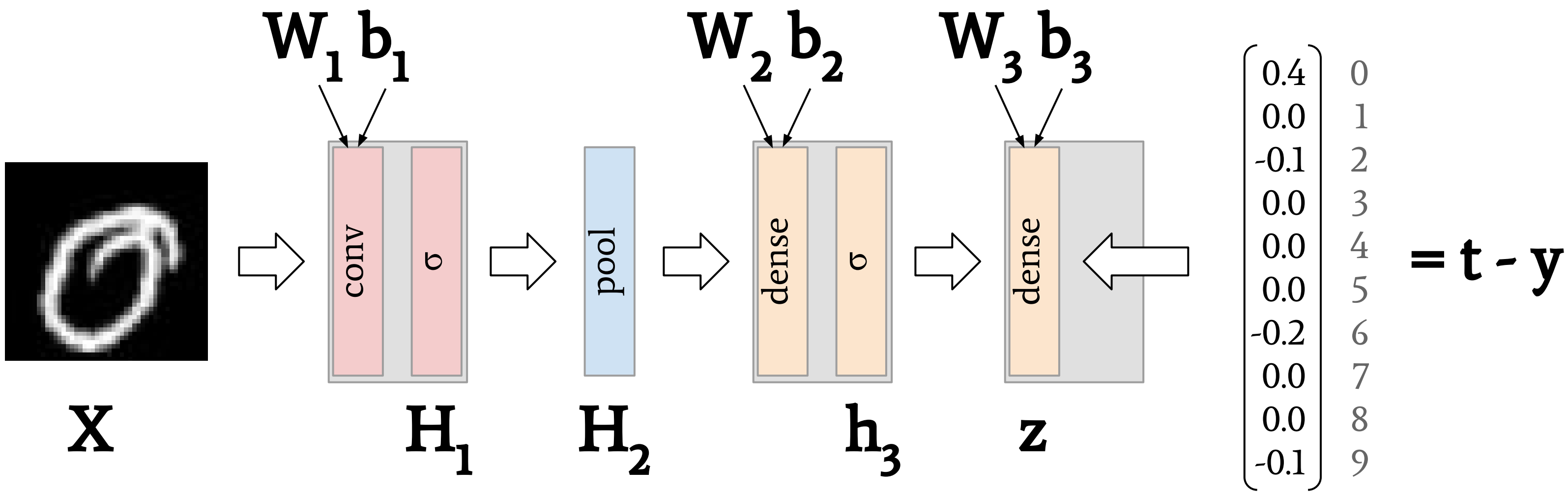


Find direction in which the loss decreases



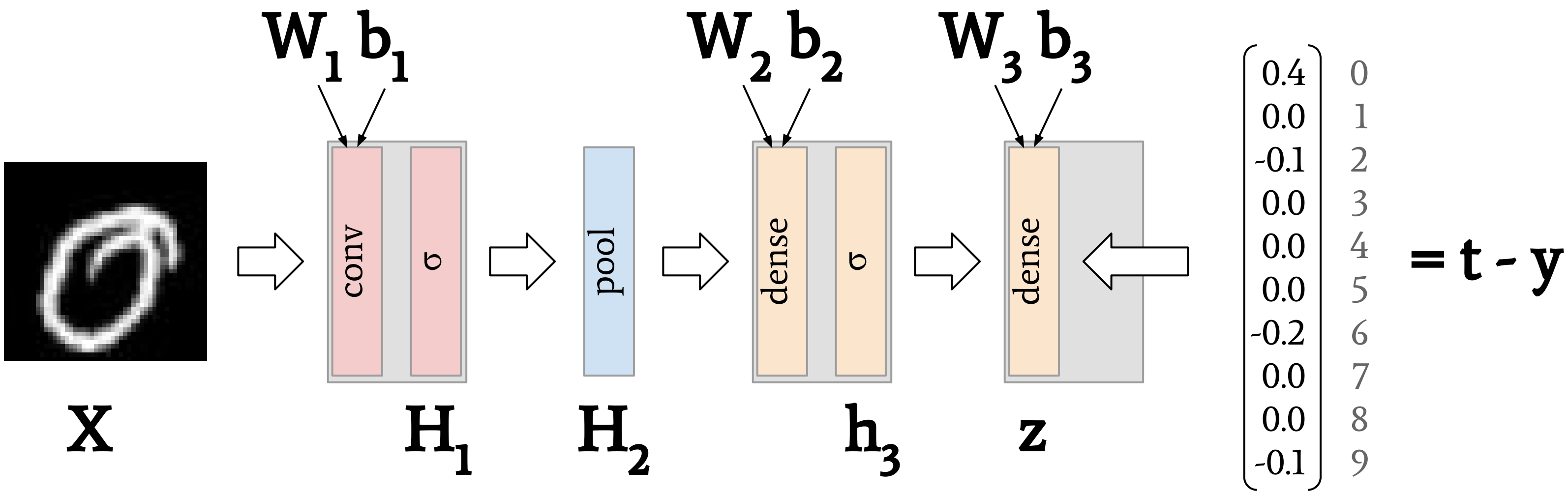
$\nabla z = t - y$

Find direction in which the loss decreases



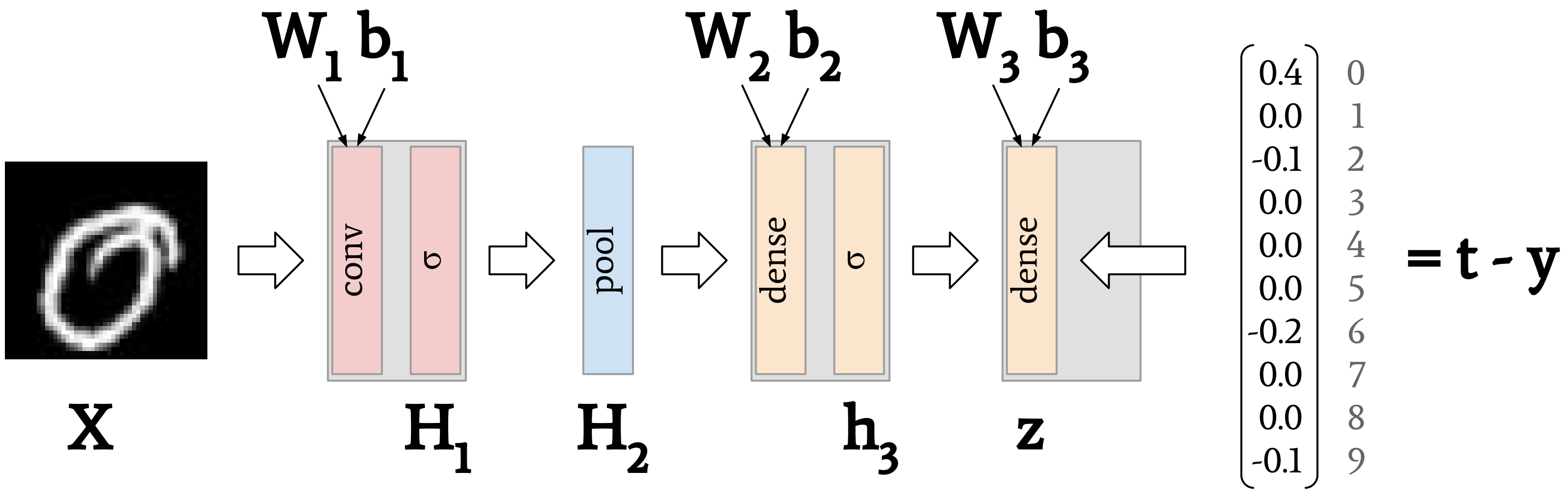
$$\nabla z = t - y$$
$$\nabla b_3 = t - y$$

Find direction in which the loss decreases



$$\nabla z = t - y$$
$$\nabla b_3 = t - y$$
$$\nabla W_3 = h_3 (t - y)^T$$

Find direction in which the loss decreases



0.4 0.0 -0.1 0.0 0.0 0.0 -0.2 0.0 0.0 -0.1 = $(\mathbf{t} - \mathbf{y})^T$

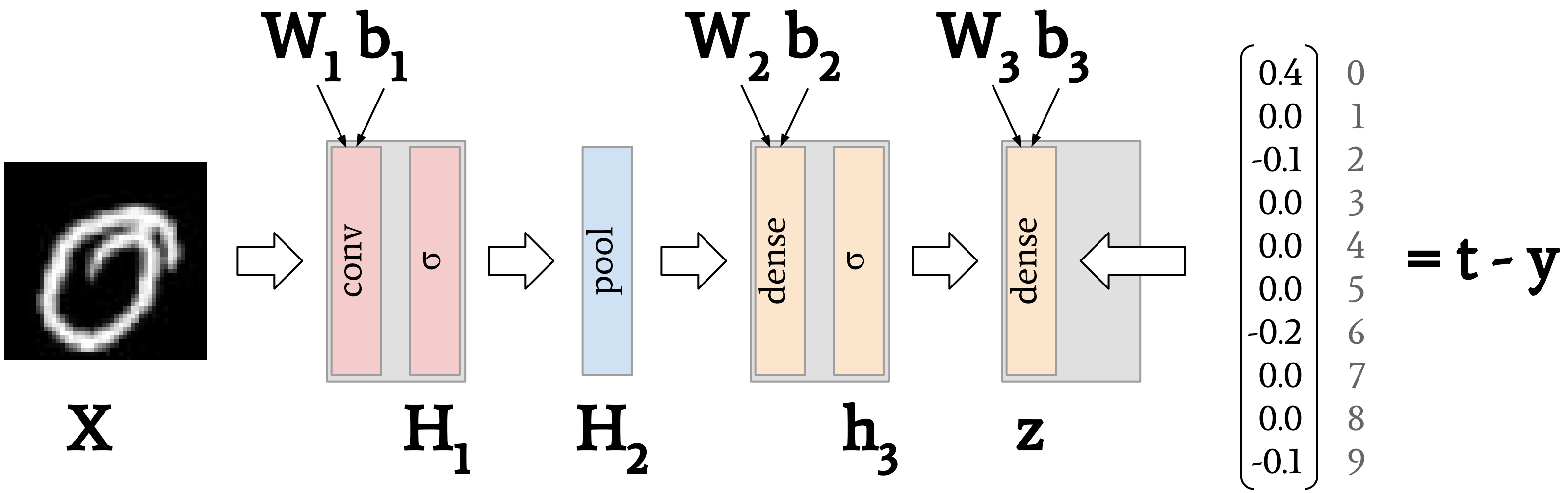
0.9
0.1
0.3
0.0
1.0
0.0
...
= $\mathbf{h}_3$

$$\nabla \mathbf{z} = \mathbf{t} - \mathbf{y}$$

$$\nabla \mathbf{b}_3 = \mathbf{t} - \mathbf{y}$$

$$\nabla \mathbf{W}_3 = \mathbf{h}_3(\mathbf{t} - \mathbf{y})^T$$

Find direction in which the loss decreases



0.4 0.0 -0.1 0.0 0.0 0.0 -0.2 0.0 0.0 -0.1 $= (\mathbf{t} - \mathbf{y})^T$

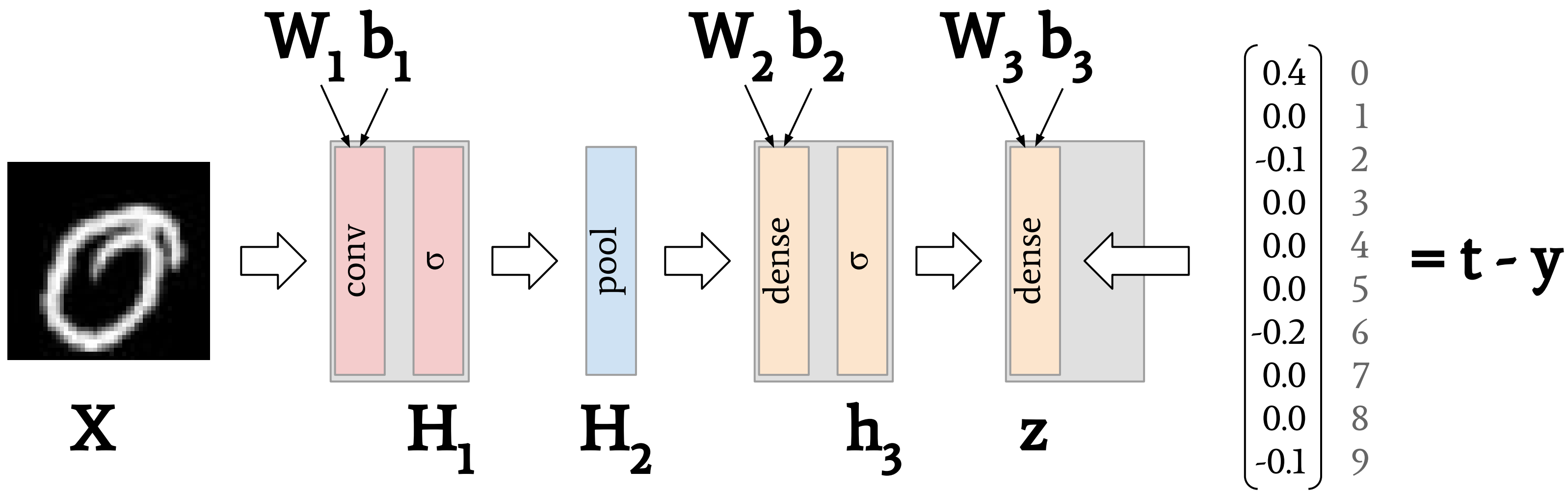
0.9 .36
0.1 .04
0.3 .12
0.0 .0
1.0 .4
0.0 .0
... ...
$= \mathbf{h}_3$

$\nabla \mathbf{z} = \mathbf{t} - \mathbf{y}$

$\nabla \mathbf{b}_3 = \mathbf{t} - \mathbf{y}$

$\nabla \mathbf{W}_3 = \mathbf{h}_3 (\mathbf{t} - \mathbf{y})^T$

Find direction in which the loss decreases

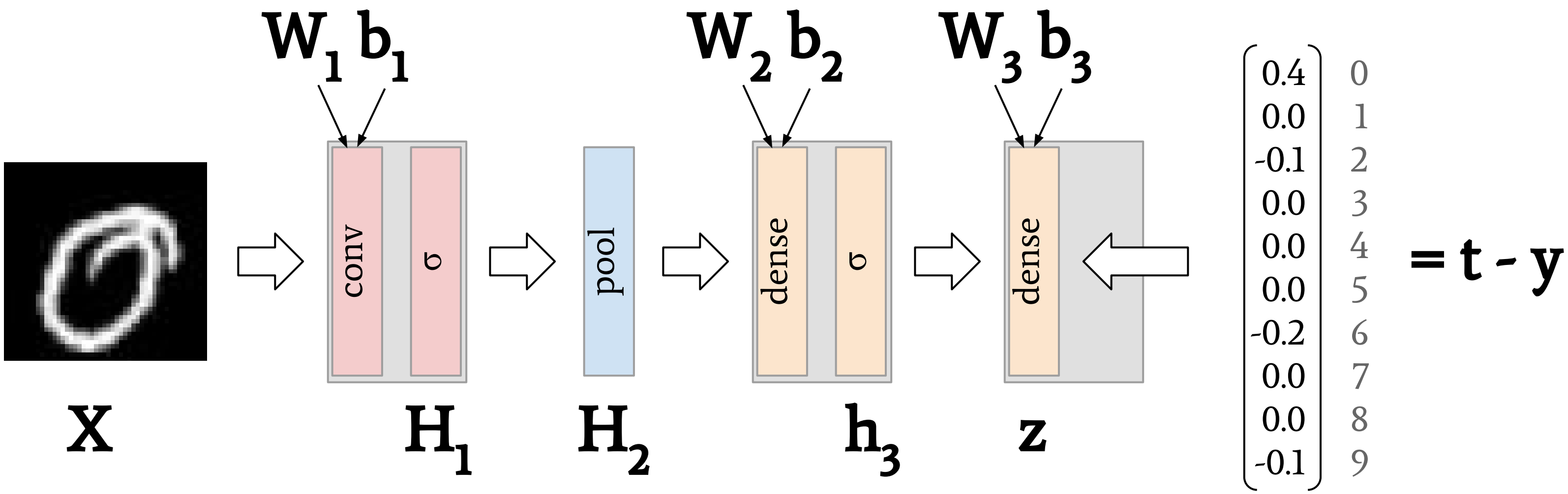


0.4 0.0 -0.1 0.0 0.0 0.0 -0.2 0.0 0.0 -0.1 = $(\mathbf{t}-\mathbf{y})^T$

0.9 .36 .0
0.1 .04 .0
0.3 .12 .0
0.0 .0 .0
1.0 .4 .0
0.0 .0 .0
... ...
= $\mathbf{h}_3$

$$\nabla \mathbf{z} = \mathbf{t} - \mathbf{y}$$
$$\nabla \mathbf{b}_3 = \mathbf{t} - \mathbf{y}$$
$$\nabla \mathbf{W}_3 = \mathbf{h}_3(\mathbf{t} - \mathbf{y})^T$$

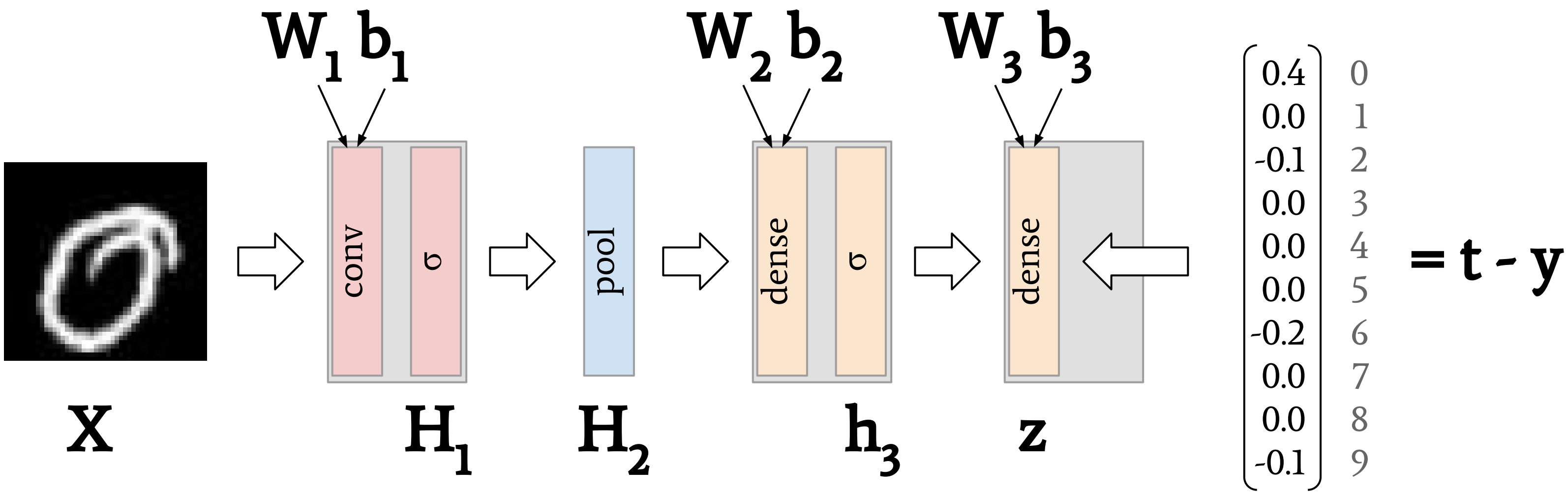
Find direction in which the loss decreases



	0.4 0.0 -0.1 0.0 0.0 0.0 -0.2 0.0 0.0 -0.1 = $(\mathbf{t}-\mathbf{y})^T$			
0.9	.36	.0	-.09	
0.1	.04	.0	-.01	
0.3	.12	.0	-.03	
0.0	.0	.0	.0	
1.0	.4	.0	-.1	
0.0	.0	.0	.0	
...	...	...	...	
= $\mathbf{h}_3$				

$$\nabla \mathbf{z} = \mathbf{t} - \mathbf{y}$$
$$\nabla \mathbf{b}_3 = \mathbf{t} - \mathbf{y}$$
$$\nabla \mathbf{W}_3 = \mathbf{h}_3(\mathbf{t} - \mathbf{y})^T$$

Find direction in which the loss decreases



$= (\mathbf{t} - \mathbf{y})^T$

	0.4	0.0	-0.1	0.0	0.0	0.0	-0.2	0.0	0.0	-0.1
0.9	.36	.0	-.09							
0.1	.04	.0	-.01							
0.3	.12	.0	-.03			…				
0.0	.0	.0	.0							
1.0	.4	.0	-.1							
0.0	.0	.0	.0							
…	…	…	…							

$= \nabla \mathbf{W}_3$

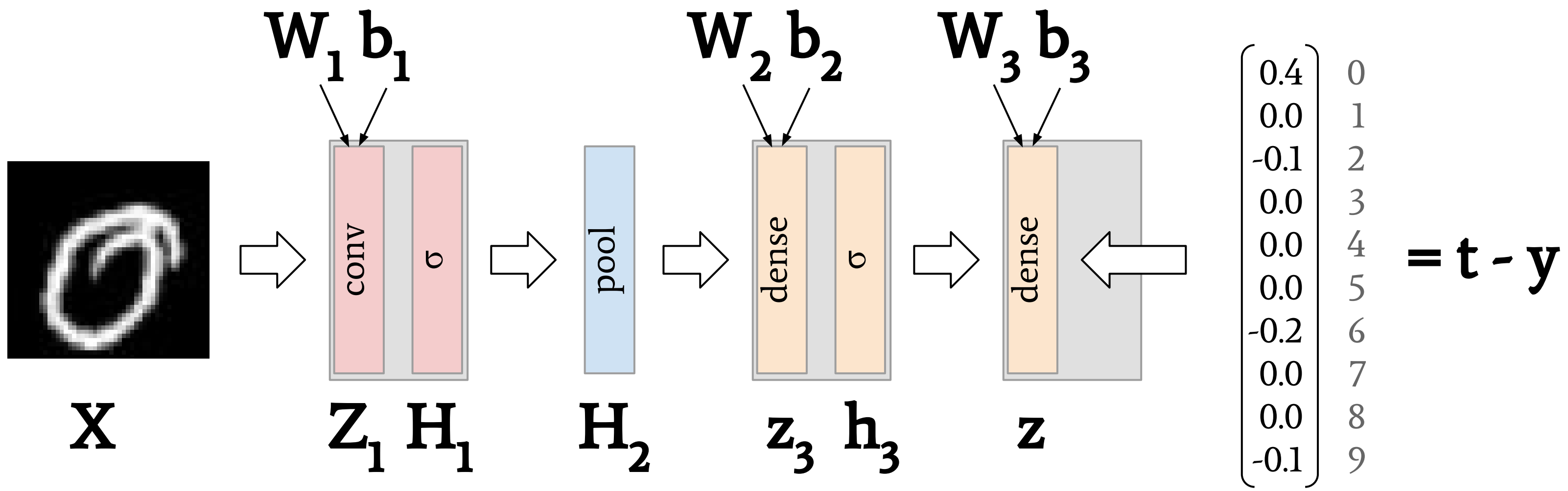
$= \mathbf{h}_3$

$$\nabla \mathbf{z} = \mathbf{t} - \mathbf{y}$$

$$\nabla \mathbf{b}_3 = \mathbf{t} - \mathbf{y}$$

$$\nabla \mathbf{W}_3 = \mathbf{h}_3 (\mathbf{t} - \mathbf{y})^T$$

Find direction in which the loss decreases



$\nabla z = t - y$

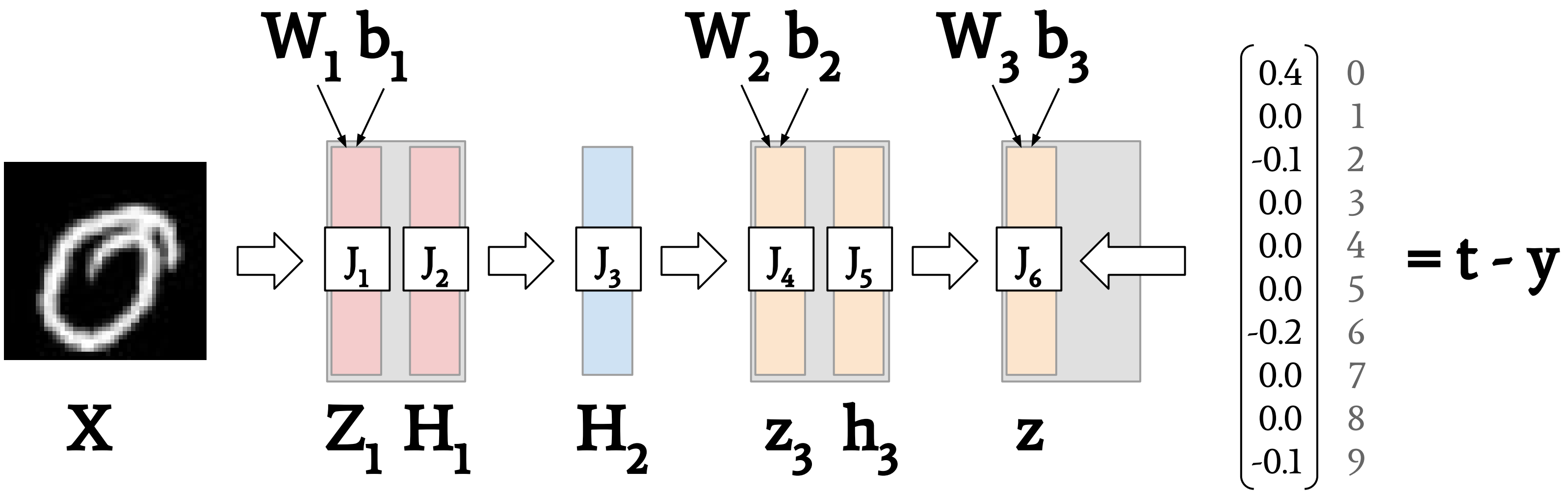
$\nabla b_3 = t - y$

$\nabla W_3 = h_3 (t - y)^T$

$\nabla z_3 = ?$

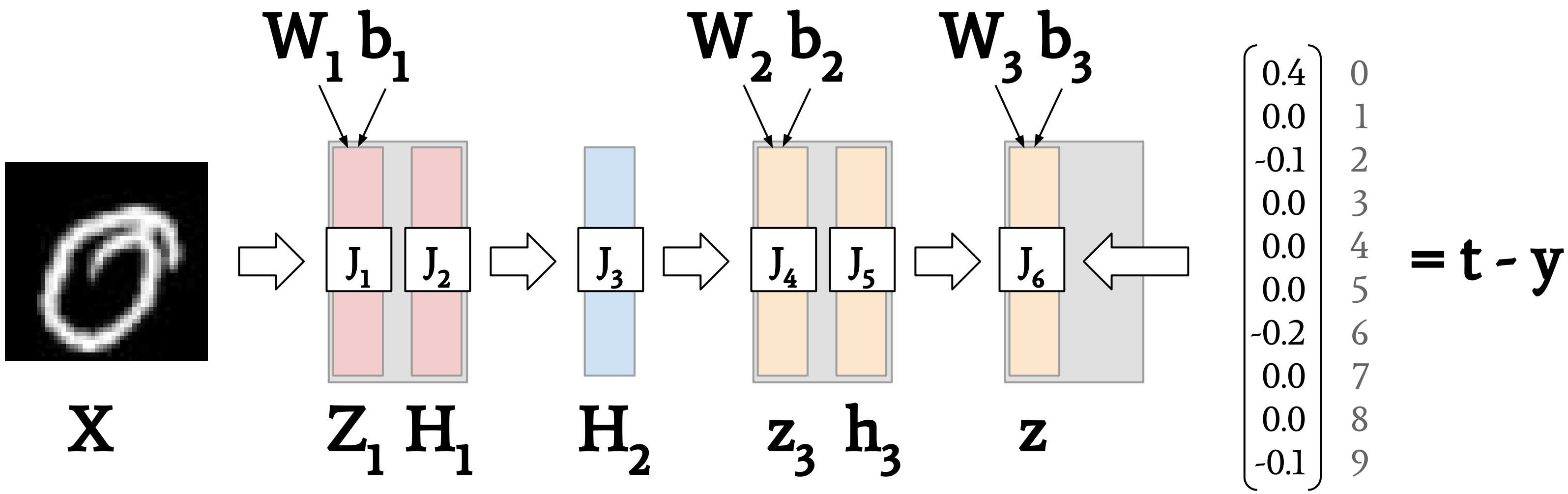
$\nabla Z_1 = ?$

Find direction in which the loss decreases



$$\nabla z = t - y$$
$$\nabla b_3 = t - y$$
$$\nabla W_3 = h_3 (t - y)^T$$
$$\nabla z_3 = ?$$
$$\nabla Z_1 = ?$$

Find direction in which the loss decreases



$J_6 = ?$

$\Delta z = J_6^T \Delta h_3$

$\nabla z = t - y$

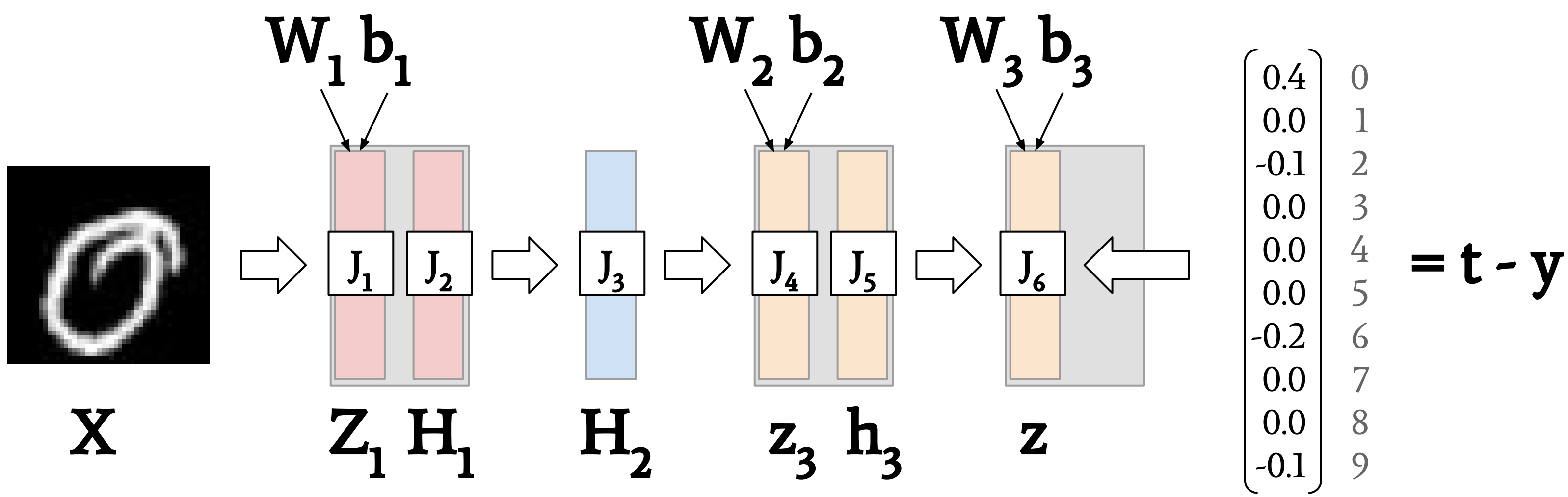
$\nabla b_3 = t - y$

$\nabla W_3 = h_3 (t - y)^T$

$\nabla z_3 = ?$

$\nabla Z_1 = ?$

Find direction in which the loss decreases



$J_6 = ?$

$\Delta z = J_6^T \, \Delta h_3$

$z = W_3^T \, h_3 + b_3$

$\nabla z = t - y$

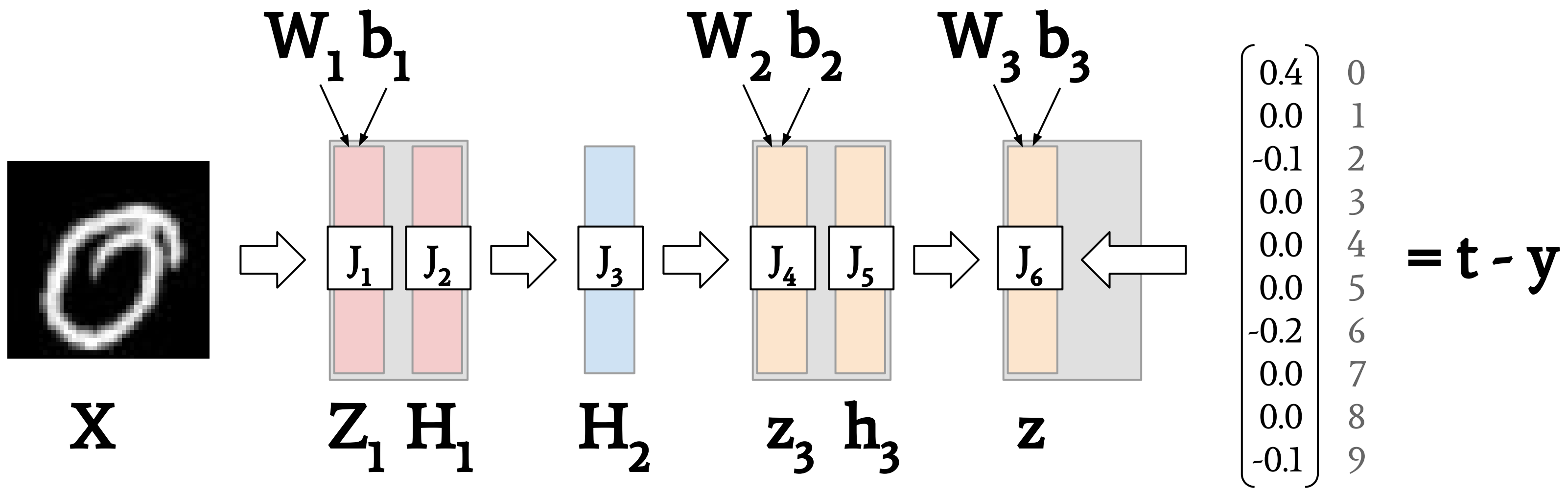
$\nabla b_3 = t - y$

$\nabla W_3 = h_3 (t - y)^T$

$\nabla z_3 = ?$

$\nabla Z_1 = ?$

Find direction in which the loss decreases



$J_6 = W_3$

$\Delta z = J_6^T \Delta h_3$

$z = W_3^T h_3 + b_3$

$\nabla z = t - y$

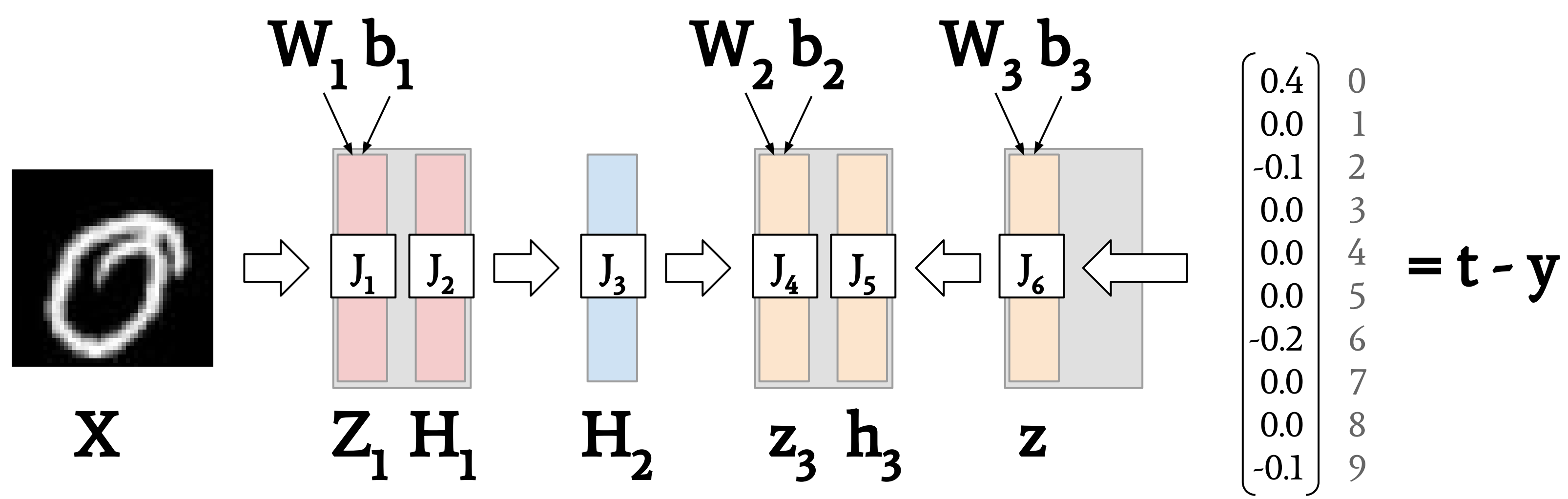
$\nabla b_3 = t - y$

$\nabla W_3 = h_3 (t - y)^T$

$\nabla z_3 = ?$

$\nabla Z_1 = ?$

Find direction in which the loss decreases



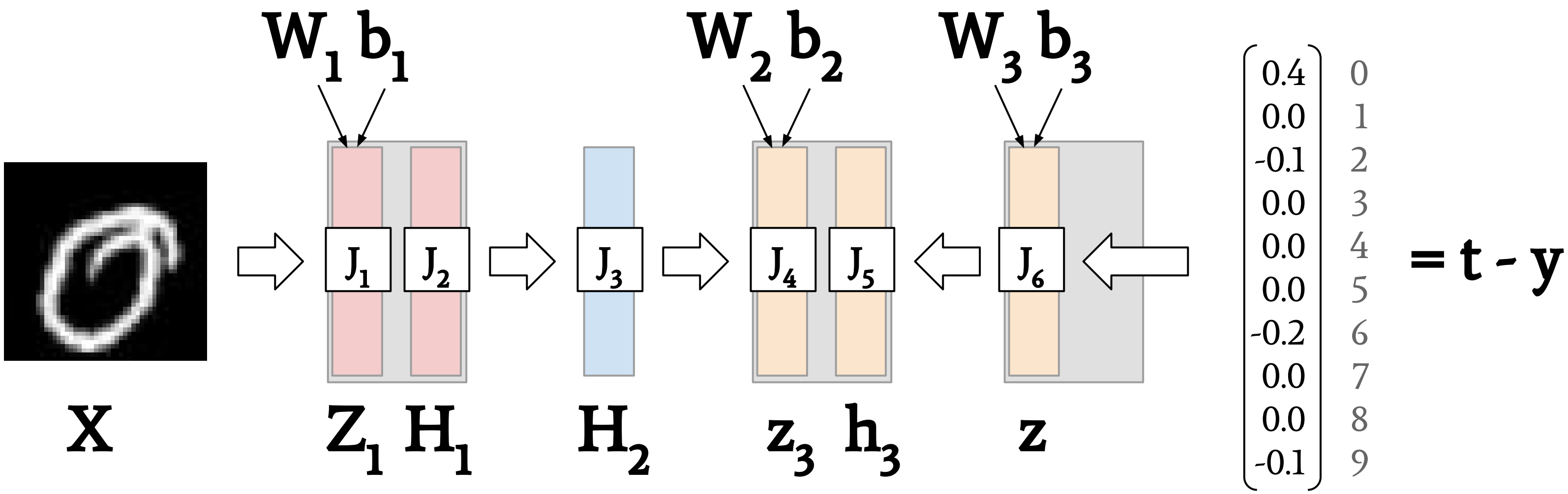
$$J_6 = W_3$$

$$\Delta z = J_6^T \Delta h_3$$

$$\nabla h_3 = J_6 \nabla z$$

$$\nabla z = t - y$$
$$\nabla b_3 = t - y$$
$$\nabla W_3 = h_3 (t - y)^T$$
$$\nabla z_3 = ?$$
$$\nabla Z_1 = ?$$

Find direction in which the loss decreases



$J_5 = ?$

$\Delta h_3 \approx J_5^T \Delta z_3$

$\nabla z = t - y$

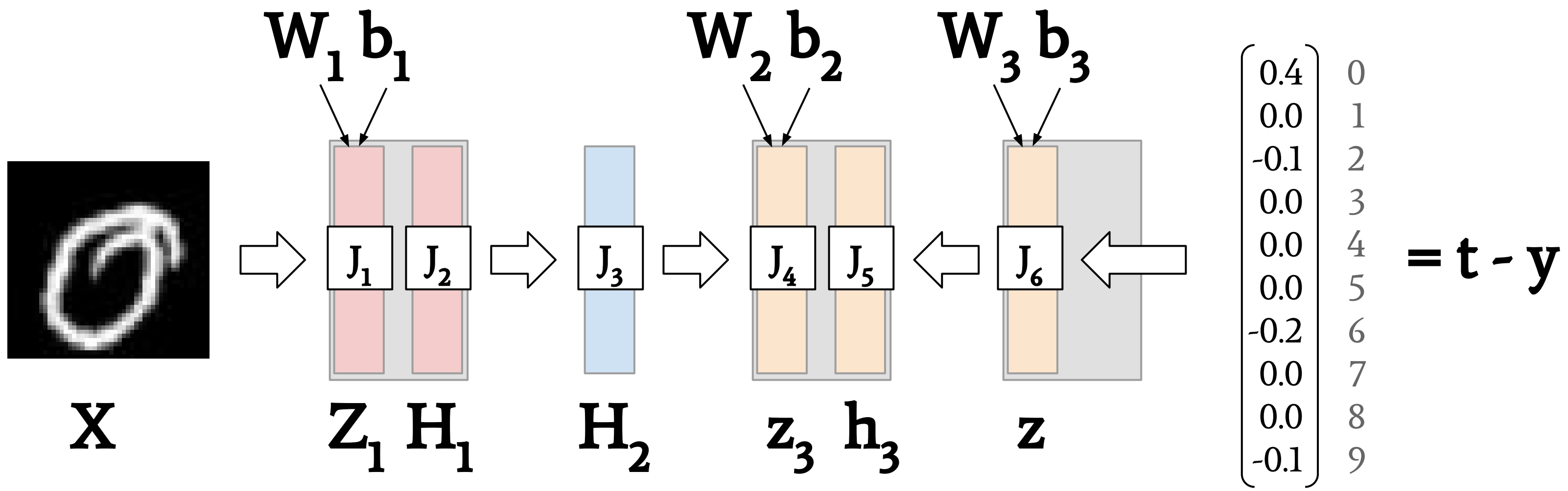
$\nabla b_3 = t - y$

$\nabla W_3 = h_3 (t - y)^T$

$\nabla z_3 = ?$

$\nabla Z_1 = ?$

Find direction in which the loss decreases



$J_5 = ?$

$\Delta h_3 \approx J_5^T \Delta z_3$

$(h_3)_i = \sigma((z_3)_i)$

$\nabla z = t - y$

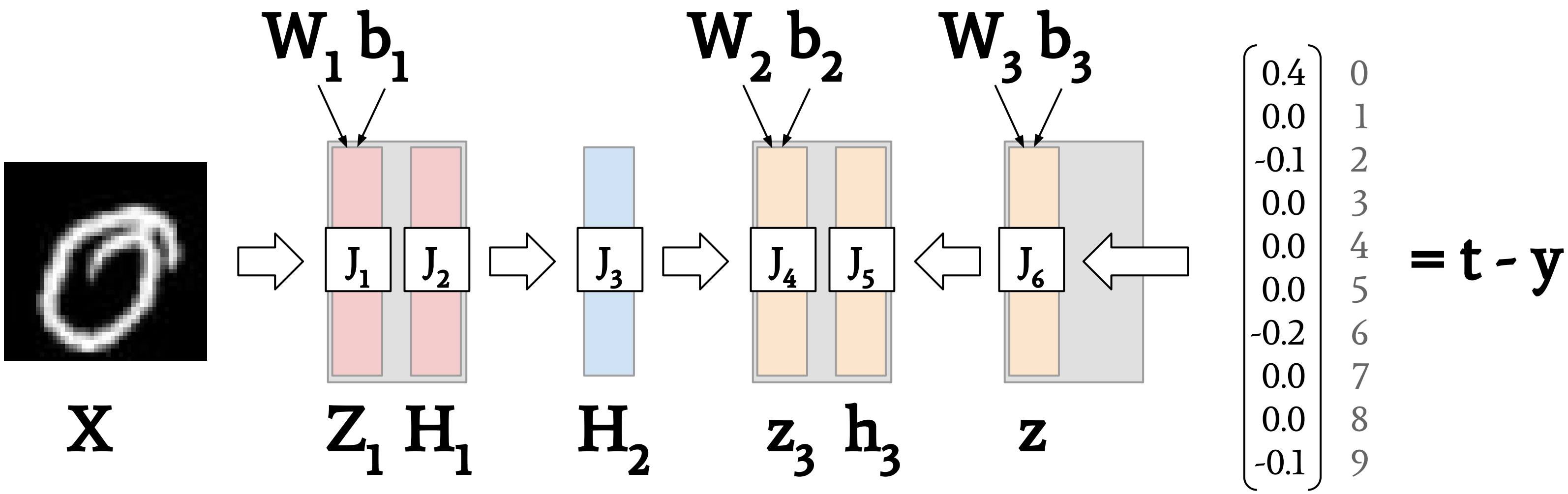
$\nabla b_3 = t - y$

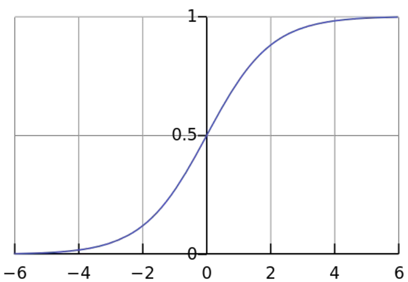
$\nabla W_3 = h_3 (t - y)^T$

$\nabla z_3 = ?$

$\nabla Z_1 = ?$

Find direction in which the loss decreases





$J_5 = ?$

$\Delta h_3 \approx J_5^T \Delta z_3$

$(h_3)_i = \sigma((z_3)_i)$

$\nabla z = t - y$

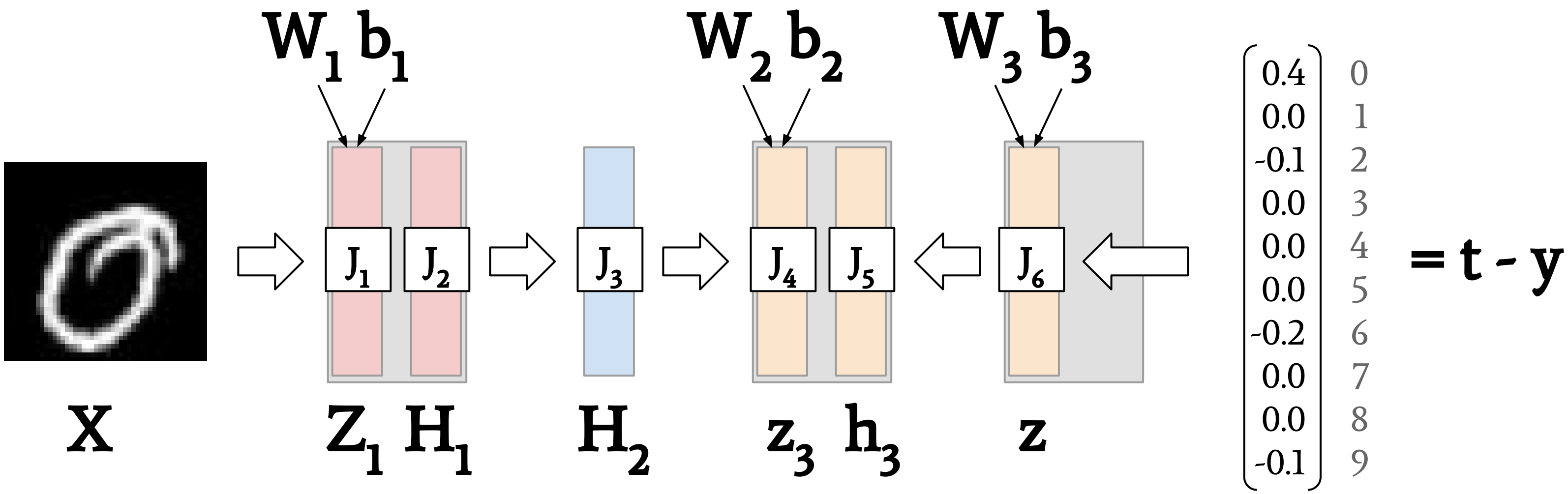
$\nabla b_3 = t - y$

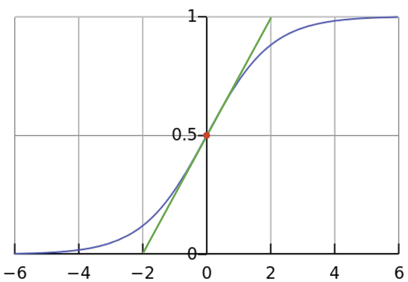
$\nabla W_3 = h_3 (t - y)^T$

$\nabla z_3 = ?$

$\nabla Z_1 = ?$

Find direction in which the loss decreases





$J_5 = ?$

$\Delta h_3 \approx J_5^T \, \Delta z_3$

$(h_3)_i = \sigma((z_3)_i)$

$\nabla z = t - y$

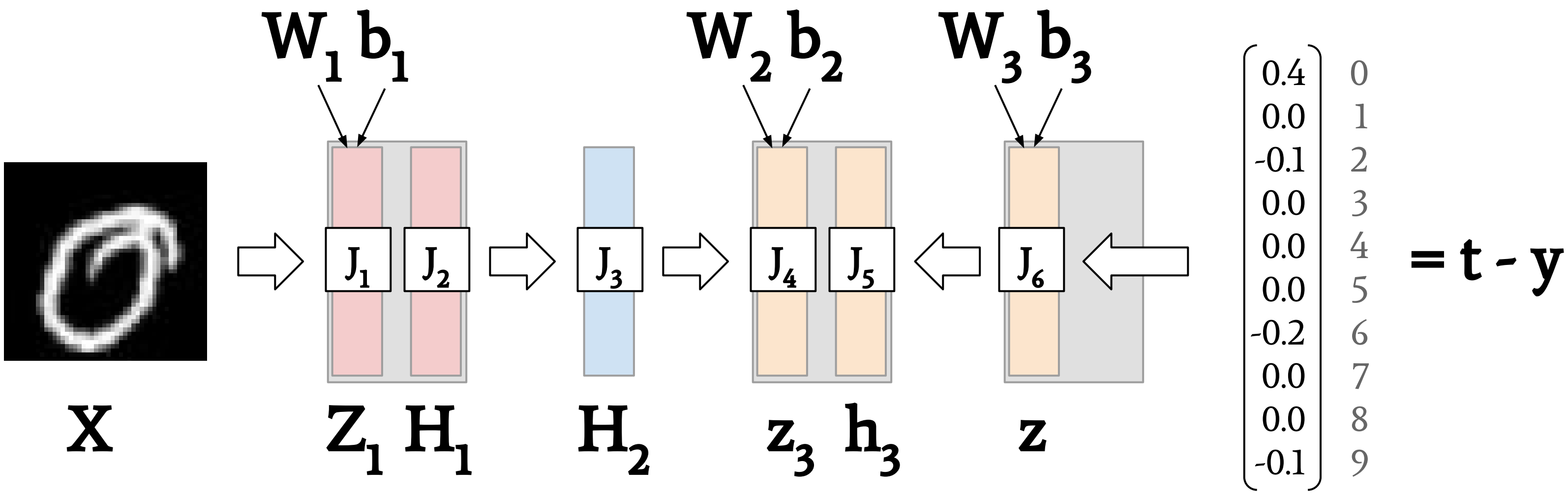
$\nabla b_3 = t - y$

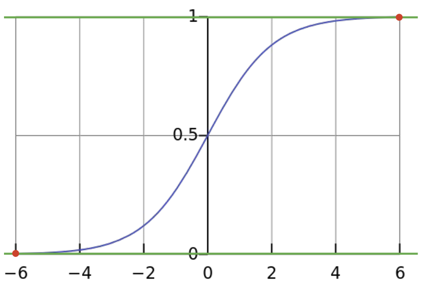
$\nabla W_3 = h_3 (t - y)^T$

$\nabla z_3 = ?$

$\nabla Z_1 = ?$

Find direction in which the loss decreases





$J_5 = ?$

$\Delta h_3 \approx J_5^T \Delta z_3$

$(h_3)_i = \sigma((z_3)_i)$

$\nabla z = t - y$

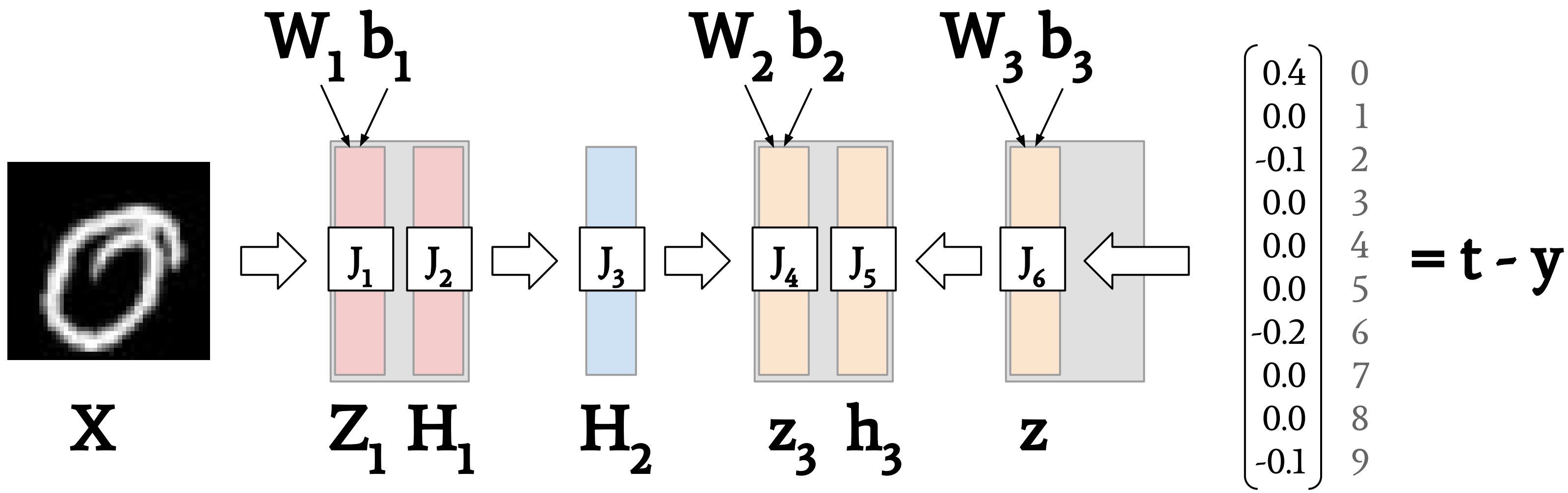
$\nabla b_3 = t - y$

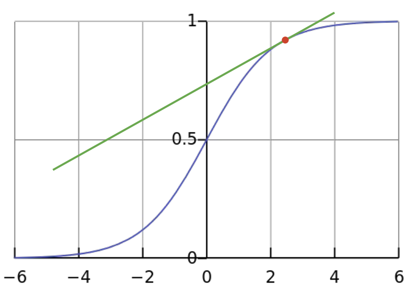
$\nabla W_3 = h_3 (t - y)^T$

$\nabla z_3 = ?$

$\nabla Z_1 = ?$

Find direction in which the loss decreases





$J_5 = ?$

$\Delta h_3 \approx J_5^T \, \Delta z_3$

$(h_3)_i = \sigma((z_3)_i)$

$\nabla z = t - y$

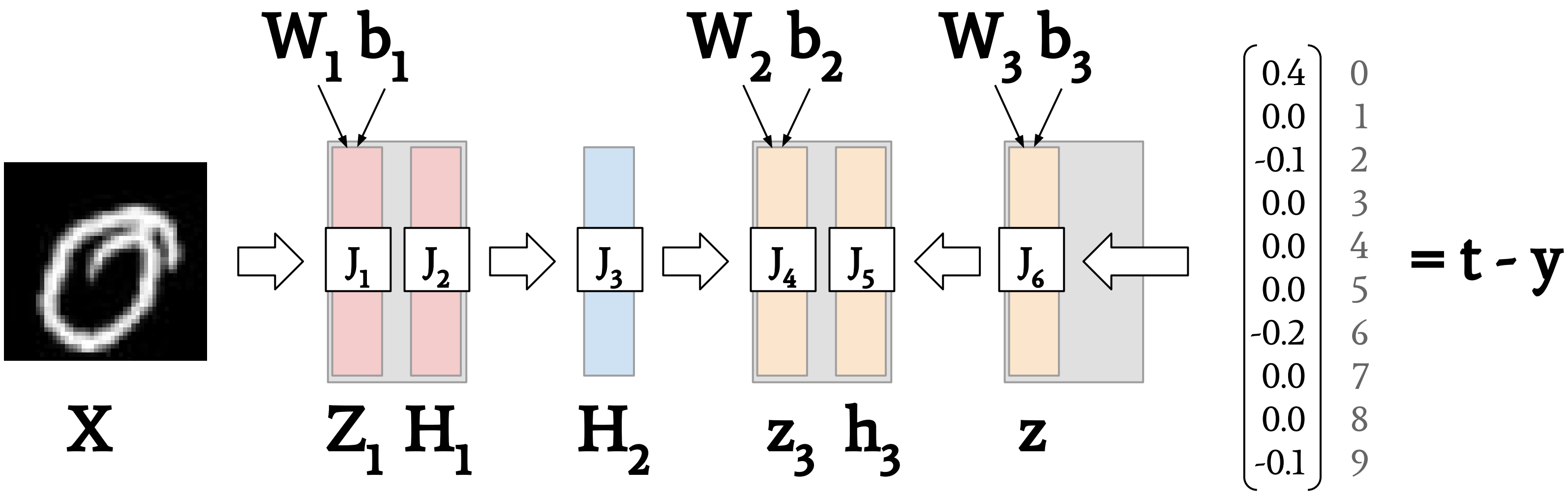
$\nabla b_3 = t - y$

$\nabla W_3 = h_3 (t - y)^T$

$\nabla z_3 = ?$

$\nabla Z_1 = ?$

Find direction in which the loss decreases



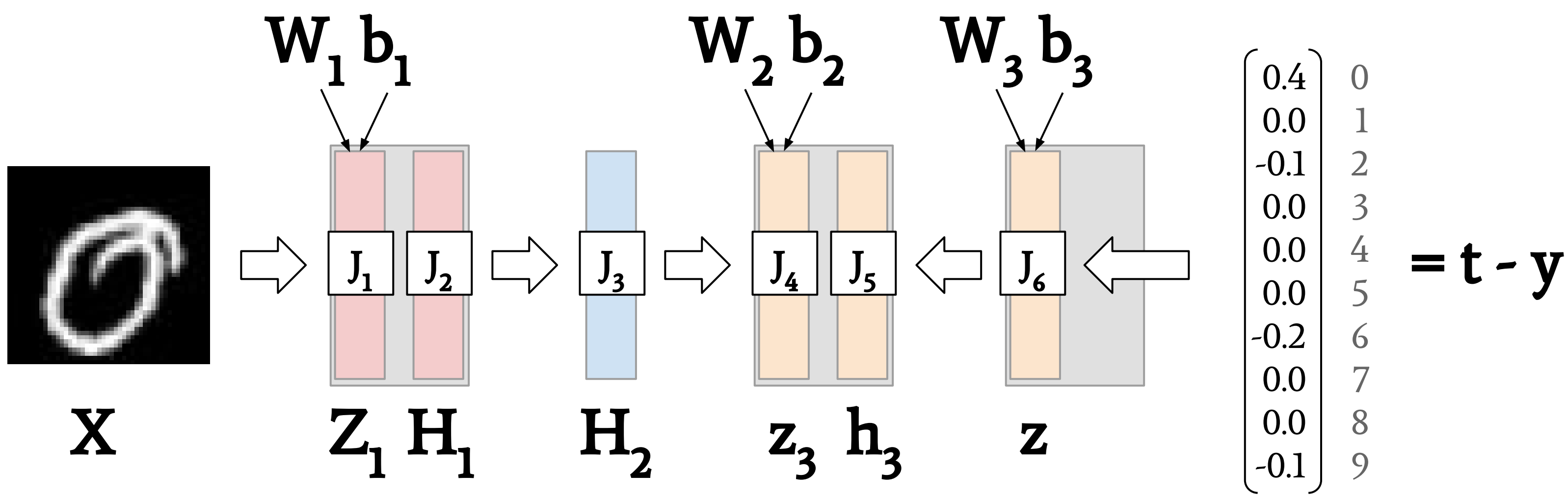
$$(J_5)_{i,i} = \sigma'((z_3)_i)$$

$$\Delta h_3 \approx J_5^T \, \Delta z_3$$

$$(h_3)_i = \sigma((z_3)_i)$$

$$\nabla z = t - y$$
$$\nabla b_3 = t - y$$
$$\nabla W_3 = h_3 (t - y)^T$$
$$\nabla z_3 = ?$$
$$\nabla Z_1 = ?$$

Find direction in which the loss decreases



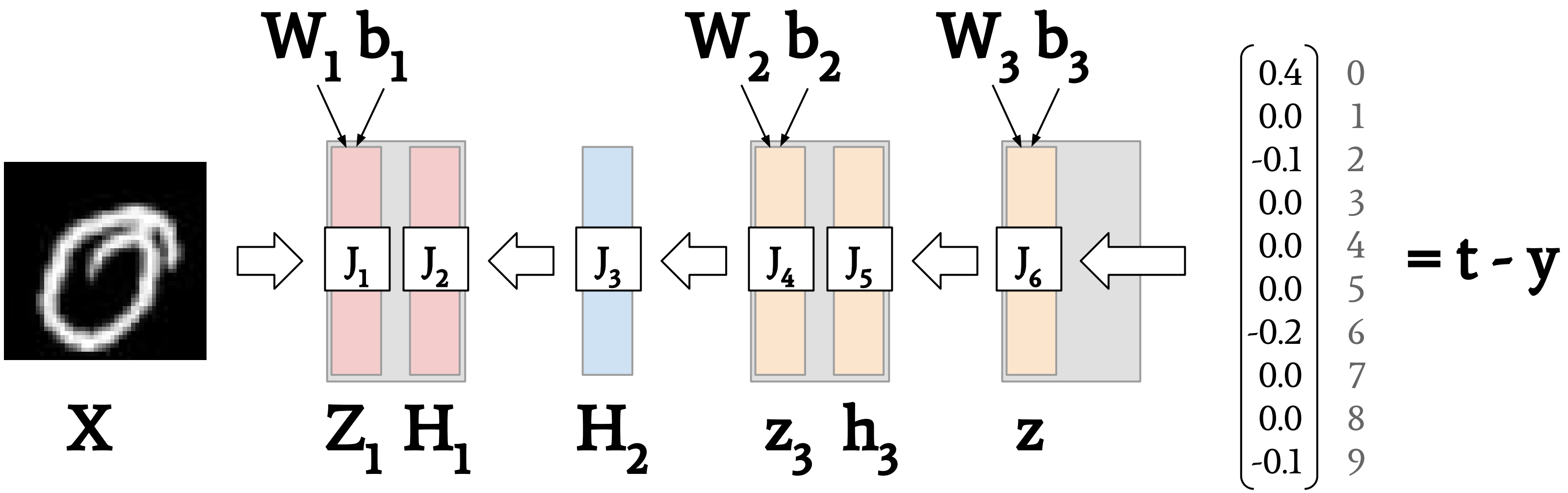
$$(J_5)_{i,i} = \sigma'((z_3)_i)$$

$$\Delta h_3 \approx J_5^T \Delta z_3$$

$$\nabla z_3 = J_5 \nabla h_3$$

$$\nabla z = t - y$$
$$\nabla b_3 = t - y$$
$$\nabla W_3 = h_3 (t - y)^T$$
$$\nabla z_3 = J_5 J_6 (t - y)$$
$$\nabla Z_1 = ?$$

Find direction in which the loss decreases



$$\nabla z = t - y$$

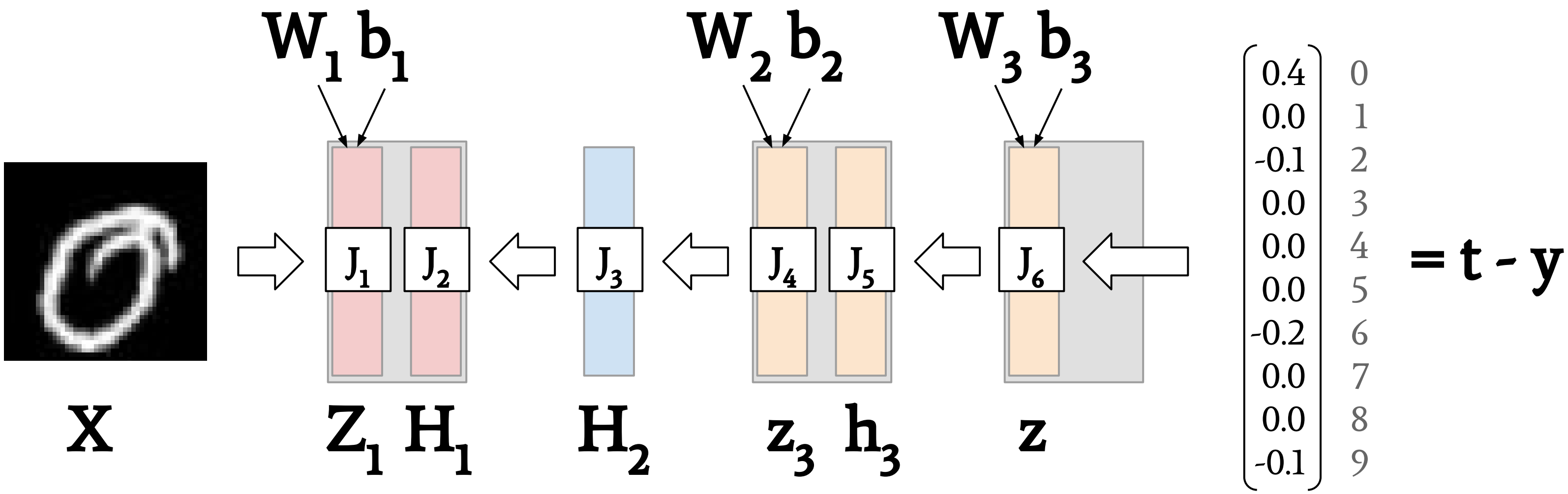
$$\nabla b_3 = t - y$$

$$\nabla W_3 = h_3 (t - y)^T$$

$$\nabla z_3 = J_5 \, J_6 \, (t - y)$$

$$\nabla Z_1 = J_2 \, J_3 \, J_4 \, J_5 \, J_6 \, (t - y)$$

Find direction in which the loss decreases



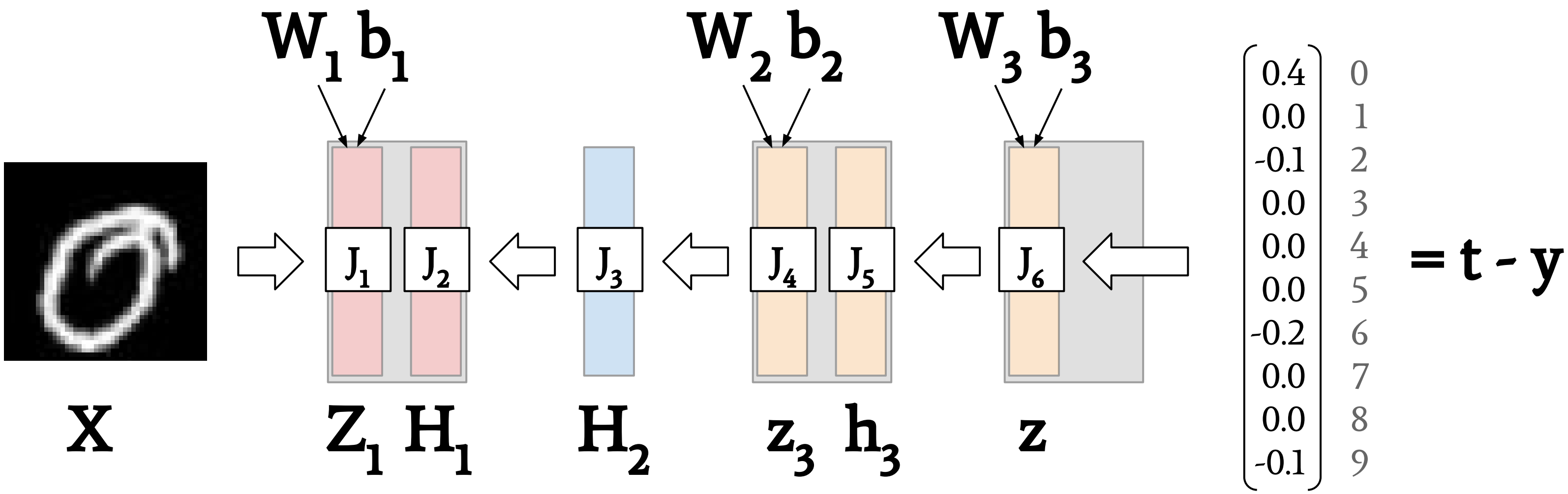
1	**3**	**5**	-3
-2	**3**	2	-1
1	2	-2	**2**
-1	**3**	-2	1

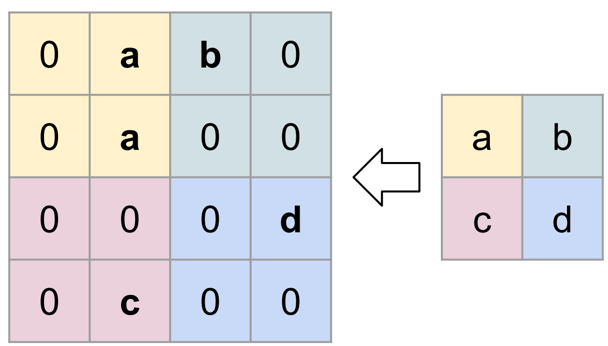
→

3	5
3	2

$$\nabla z = t - y$$
$$\nabla b_3 = t - y$$
$$\nabla W_3 = h_3 (t - y)^T$$
$$\nabla z_3 = J_5 \, J_6 \, (t - y)$$
$$\nabla Z_1 = J_2 \, J_3 \, J_4 \, J_5 \, J_6 \, (t - y)$$

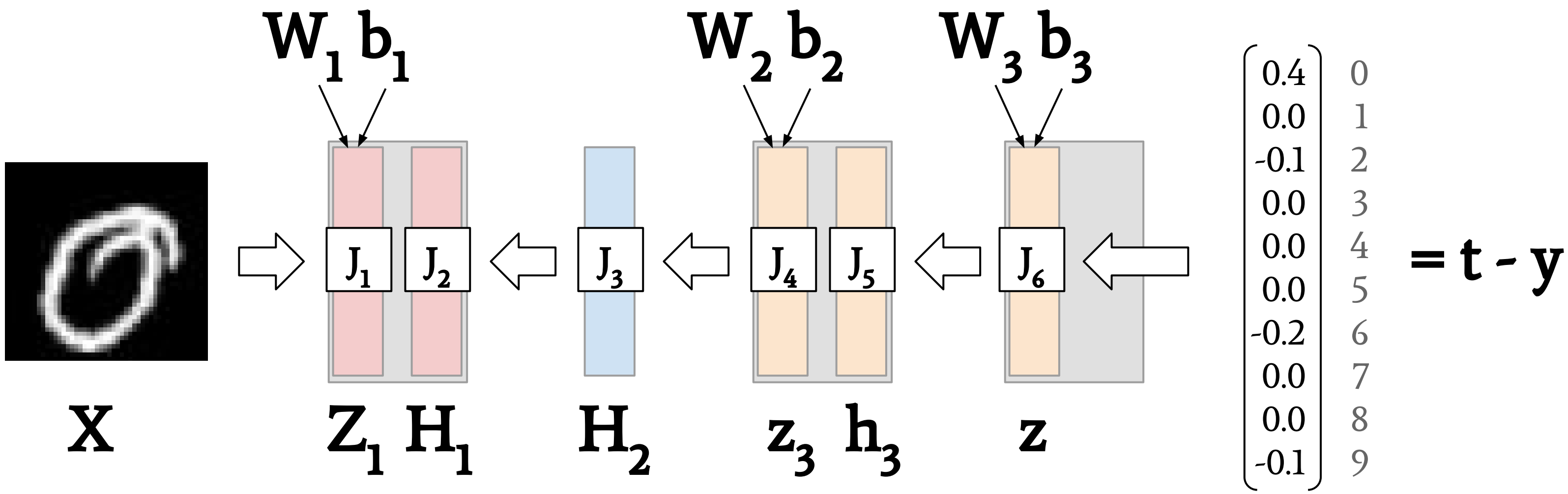
Find direction in which the loss decreases

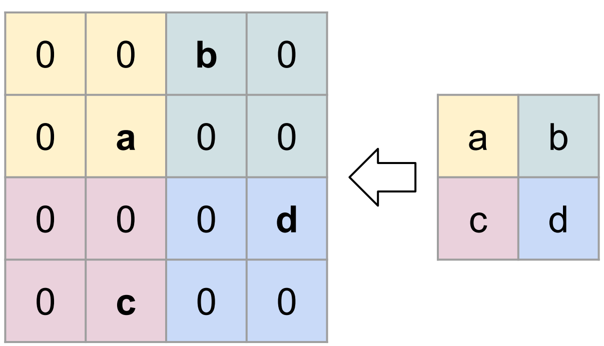




$$\nabla z = t - y$$
$$\nabla b_3 = t - y$$
$$\nabla W_3 = h_3 (t - y)^T$$
$$\nabla z_3 = J_5 J_6 (t - y)$$
$$\nabla Z_1 = J_2 J_3 J_4 J_5 J_6 (t - y)$$

Find direction in which the loss decreases





$$\nabla z = t - y$$

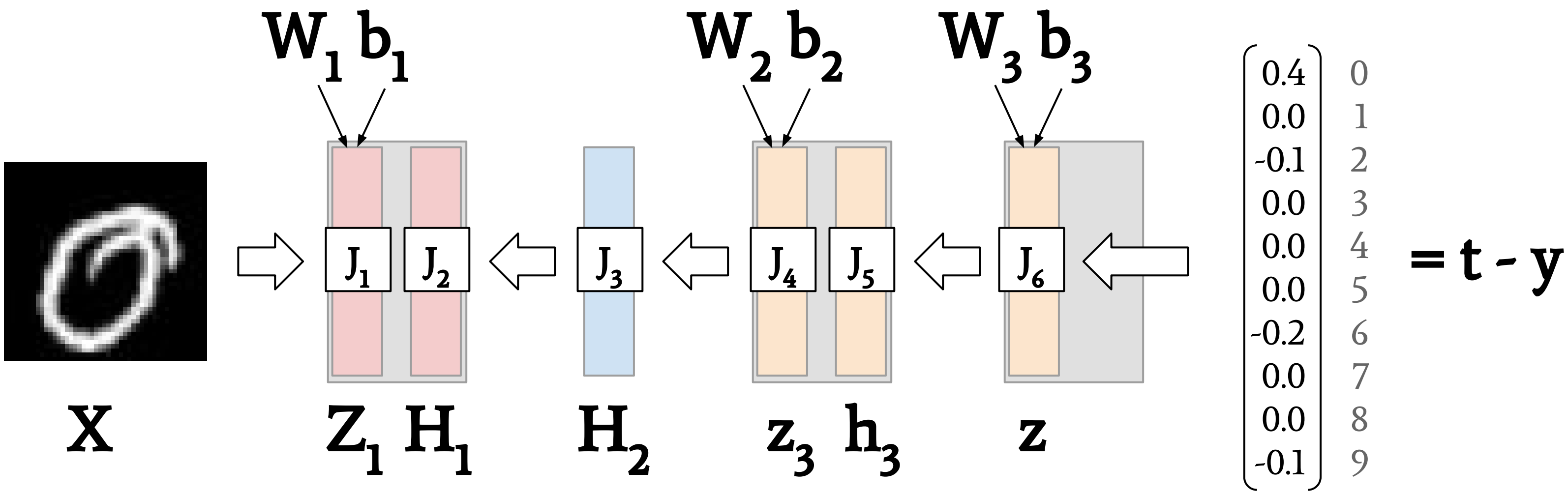
$$\nabla b_3 = t - y$$

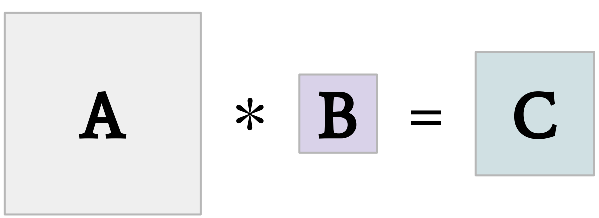
$$\nabla W_3 = h_3 (t - y)^T$$

$$\nabla z_3 = J_5 J_6 (t - y)$$

$$\nabla Z_1 = J_2 J_3 J_4 J_5 J_6 (t - y)$$

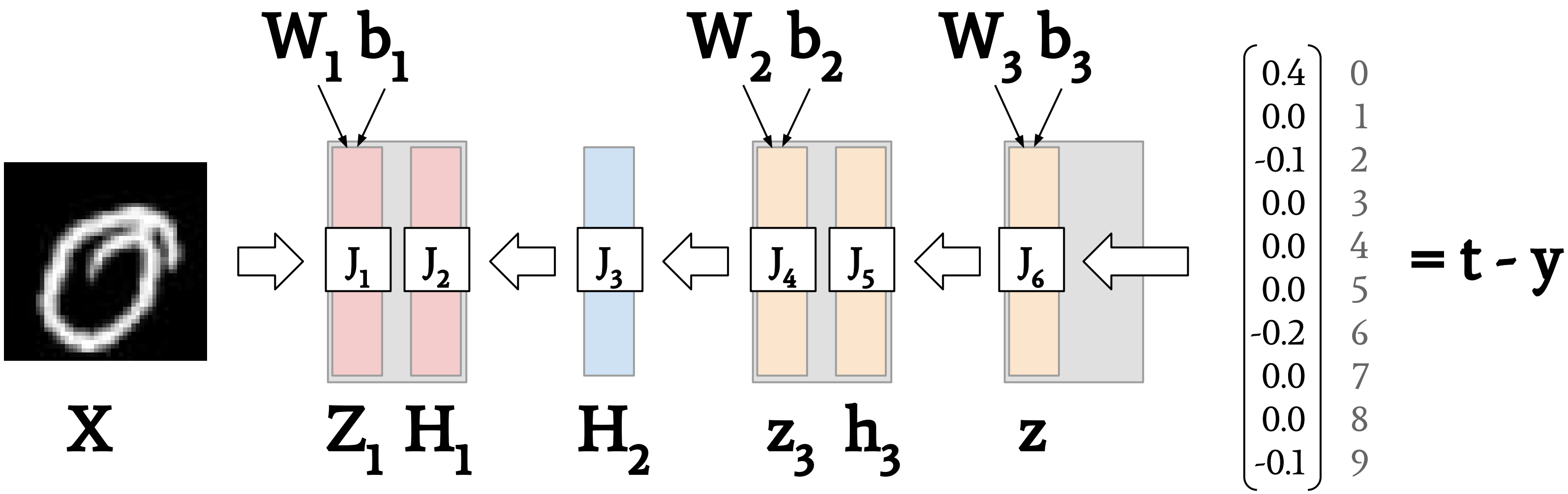
Find direction in which the loss decreases

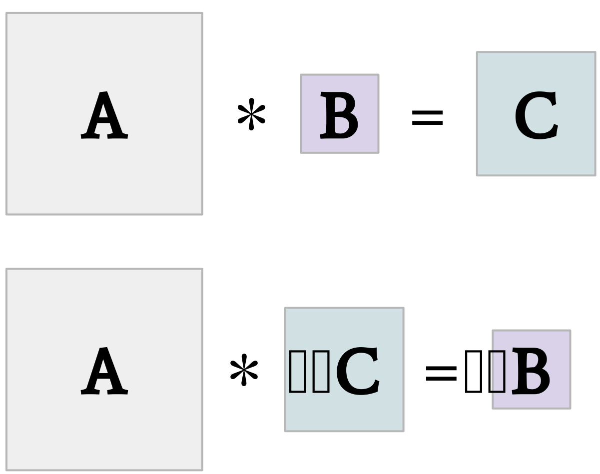




$$\nabla z = t - y$$
$$\nabla b_3 = t - y$$
$$\nabla W_3 = h_3 (t - y)^T$$
$$\nabla z_3 = J_5 J_6 (t - y)$$
$$\nabla Z_1 = J_2 J_3 J_4 J_5 J_6 (t - y)$$

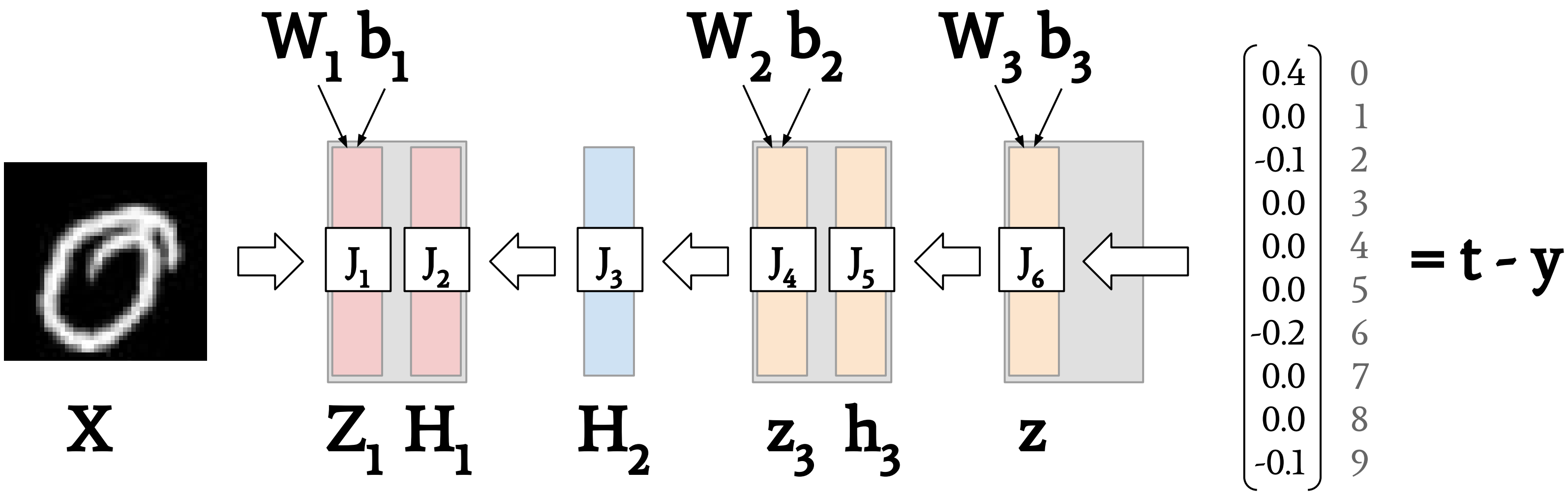
Find direction in which the loss decreases

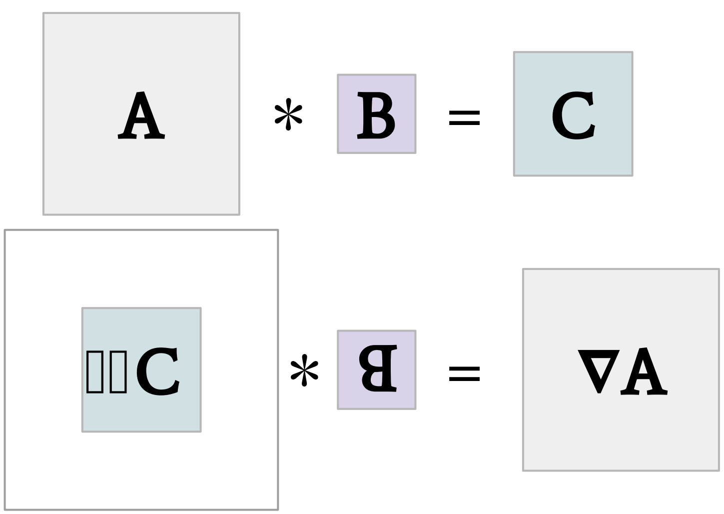




$$\nabla z = t - y$$
$$\nabla b_3 = t - y$$
$$\nabla W_3 = h_3 (t - y)^T$$
$$\nabla z_3 = J_5 J_6 (t - y)$$
$$\nabla Z_1 = J_2 J_3 J_4 J_5 J_6 (t - y)$$

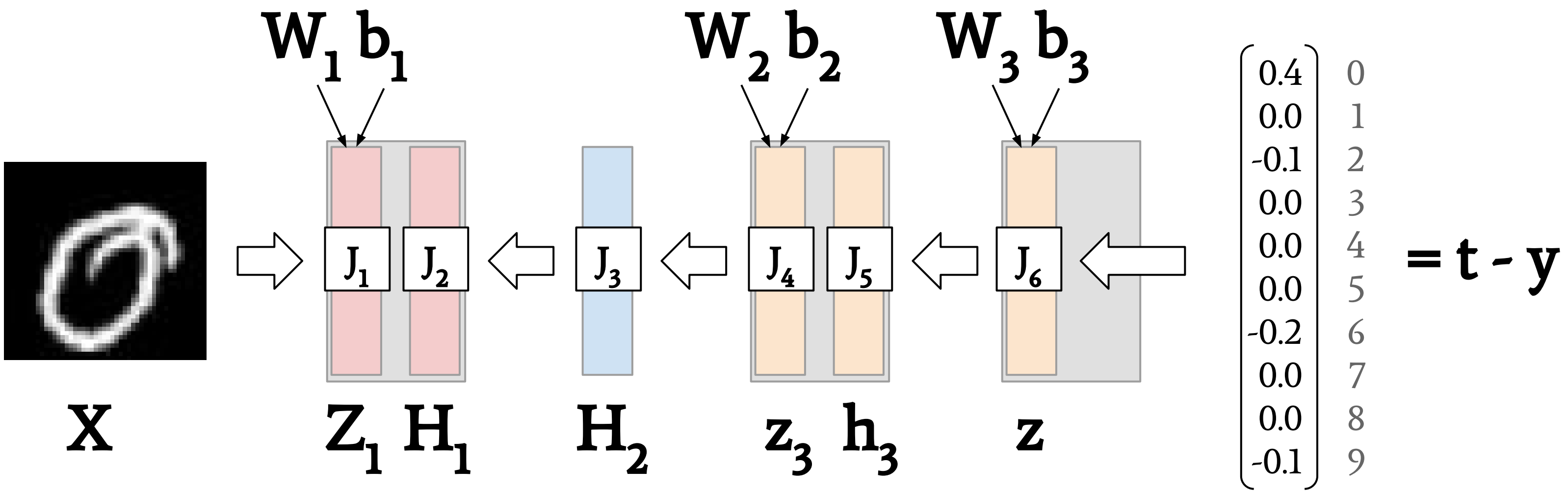
Find direction in which the loss decreases





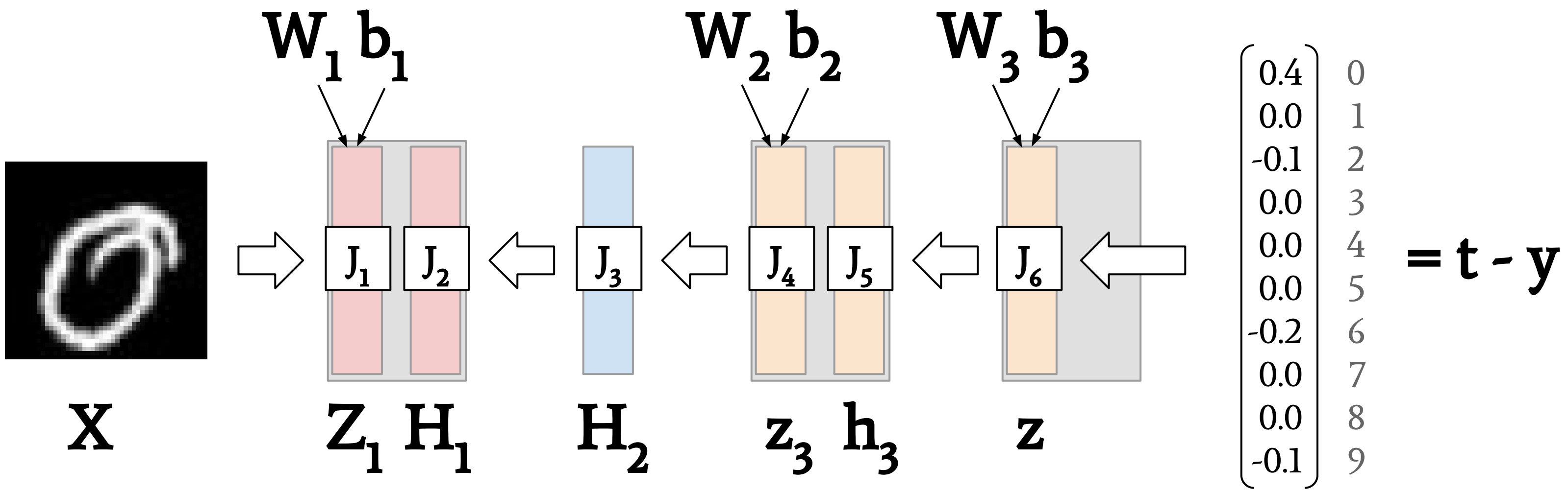
$$\nabla z = t - y$$
$$\nabla b_3 = t - y$$
$$\nabla W_3 = h_3 (t - y)^T$$
$$\nabla z_3 = J_5 J_6 (t - y)$$
$$\nabla Z_1 = J_2 J_3 J_4 J_5 J_6 (t - y)$$

Find direction in which the loss decreases



$$\nabla\theta = -\frac{\partial}{\partial\theta} J(f(\mathbf{X};\theta), \mathbf{t})$$

Find direction in which the loss decreases



$$\nabla\theta = -\frac{\partial}{\partial\theta} J(f(\mathbf{X};\theta), \mathbf{t})$$

$$-\frac{\partial}{\partial\theta} L(\theta; f, D) = -\boldsymbol{\Sigma}_{(\mathbf{X},\mathbf{T}) \in D}\frac{\partial}{\partial\theta} J(f(\mathbf{X};\theta), \mathbf{T})$$

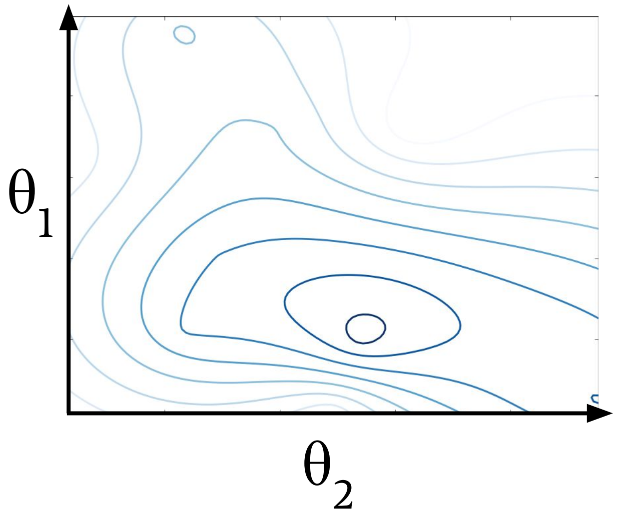
Optimization

4. **Optimize** model parameters to minimize loss

$$\theta^* = \min_\theta L(\theta; f, D)$$

Iterative scheme:

0. initialize θ randomly
1. find direction in which L decreases
2. move θ a bit into that direction
3. go to step 1



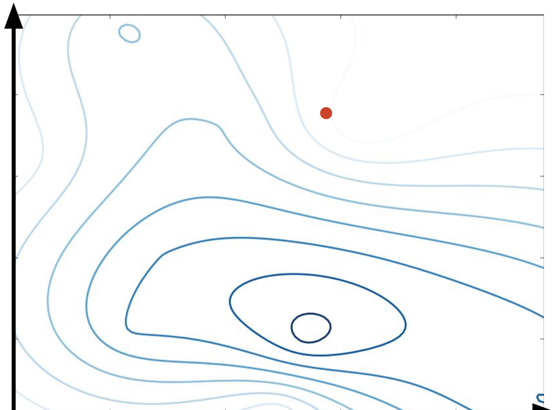
Optimization

4. **Optimize** model parameters to minimize loss

$$\theta^* = \min_\theta L(\theta; f, D)$$

Iterative scheme:

→ 0. initialize θ randomly
1. find direction in which L decreases
2. move θ a bit into that direction
3. go to step 1



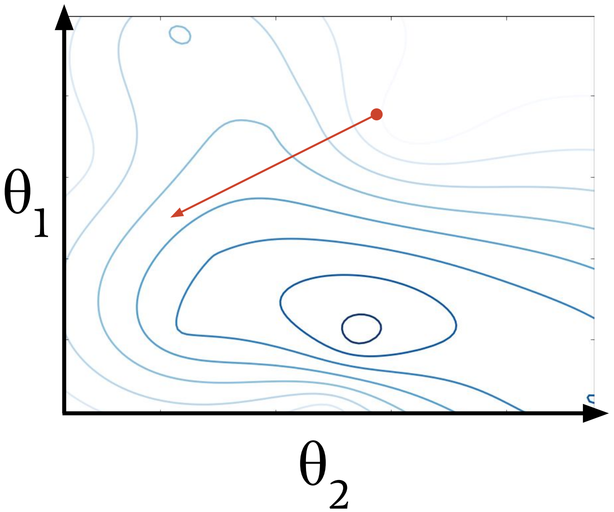
Optimization

4. **Optimize** model parameters to minimize loss

$$\theta^* = \min_\theta L(\theta; f, D)$$

Iterative scheme:

0. initialize θ randomly
1. find direction in which L decreases
2. move θ a bit into that direction
3. go to step 1



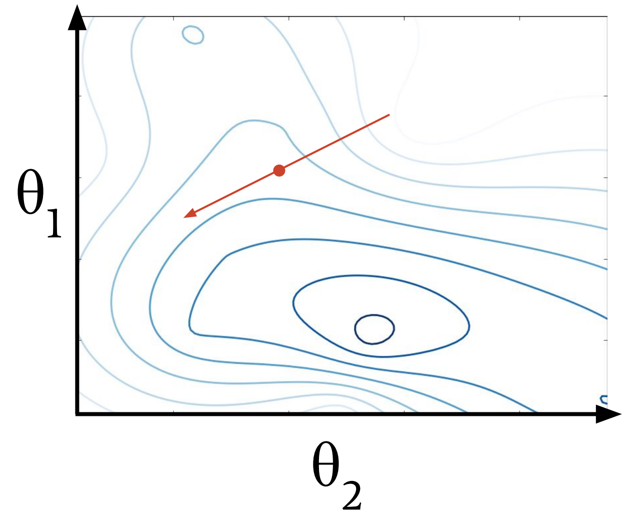
Optimization

4. **Optimize** model parameters to minimize loss

$$\theta^* = \min_\theta L(\theta; f, D)$$

Iterative scheme:

0. initialize θ randomly
1. find direction in which L decreases
2. ➔ move θ a bit into that direction
3. go to step 1



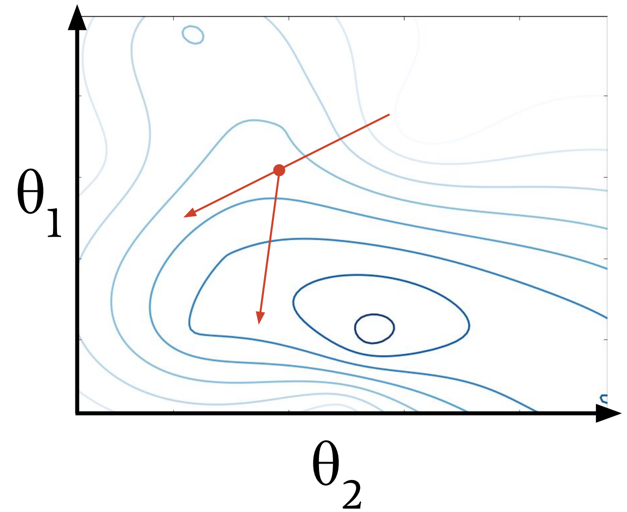
Optimization

4. **Optimize** model parameters to minimize loss

$$\theta^* = \min_\theta L(\theta; f, D)$$

Iterative scheme:

0. initialize θ randomly
1. find direction in which L decreases (←)
2. move θ a bit into that direction
3. go to step 1



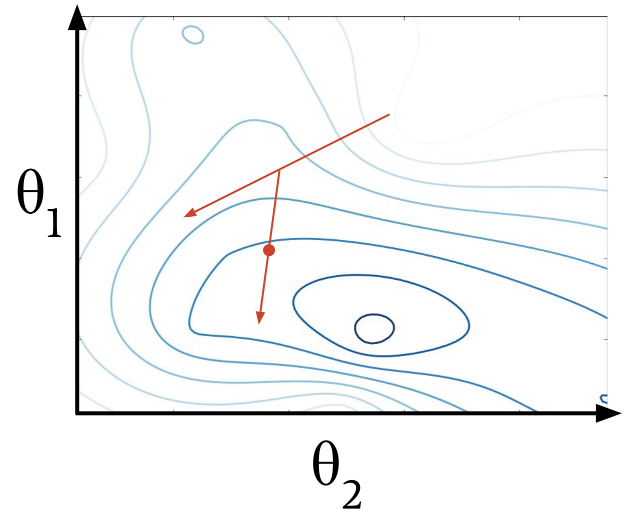
Optimization

4. **Optimize** model parameters to minimize loss

$$\theta^* = \min_\theta L(\theta; f, D)$$

Iterative scheme:

0. initialize θ randomly
1. find direction in which L decreases
2. ➔ move θ a bit into that direction
3. go to step 1



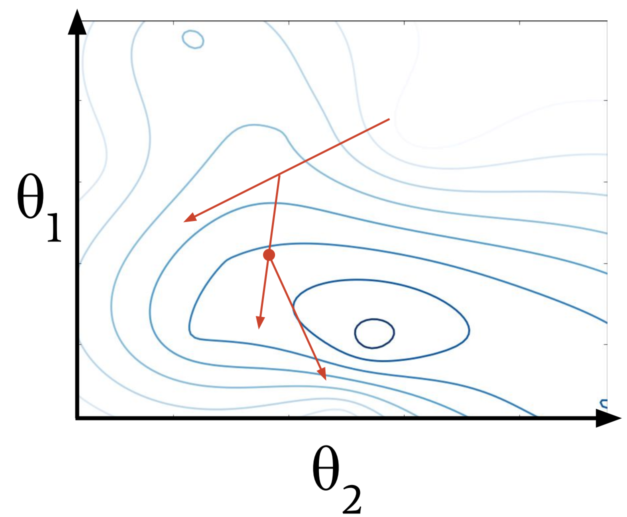
Optimization

4. **Optimize** model parameters to minimize loss

$$\theta^* = \min_\theta L(\theta; f, D)$$

Iterative scheme:

0. initialize θ randomly
1. find direction in which L decreases
2. move θ a bit into that direction
3. go to step 1



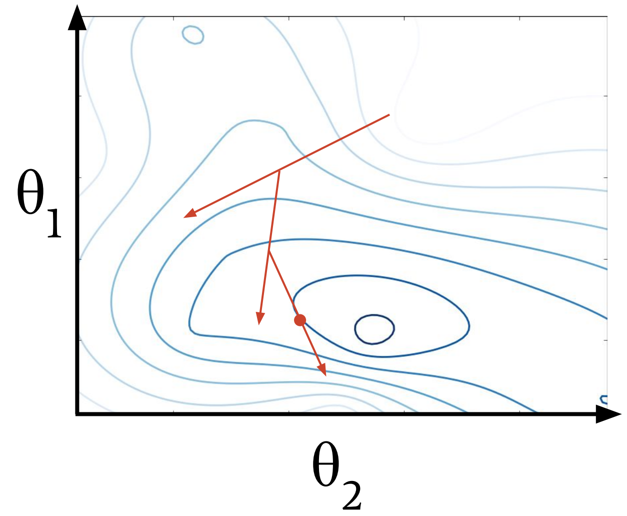
Optimization

4. **Optimize** model parameters to minimize loss

$$\theta^* = \min_\theta L(\theta; f, D)$$

Iterative scheme:

0. initialize θ randomly
1. find direction in which L decreases
2. ➔ move θ a bit into that direction
3. go to step 1



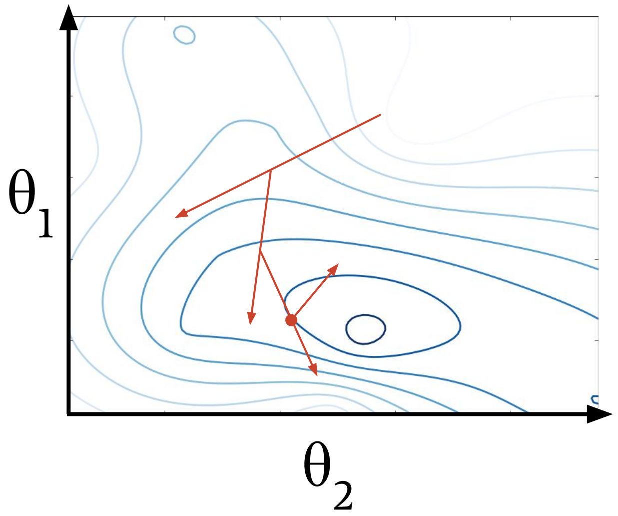
Optimization

4. **Optimize** model parameters to minimize loss

$$\theta^* = \min_\theta L(\theta; f, D)$$

Iterative scheme:

0. initialize θ randomly
1. ➜ find direction in which L decreases
2. move θ a bit into that direction
3. go to step 1



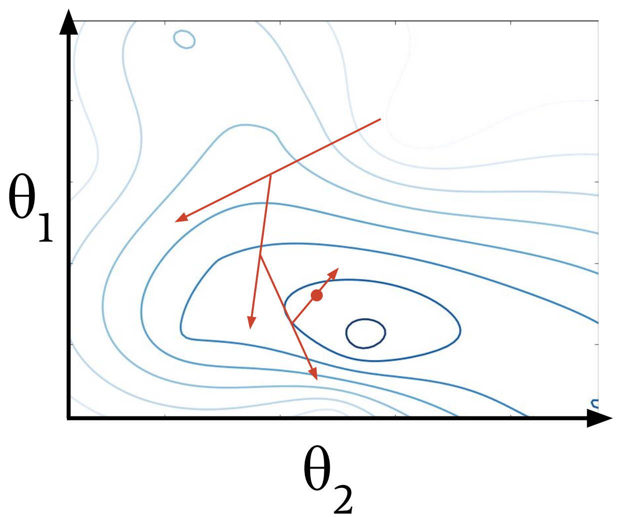
Optimization

4. **Optimize** model parameters to minimize loss

$$\theta^* = \min_\theta L(\theta; f, D)$$

Iterative scheme:

0. initialize θ randomly
1. find direction in which L decreases
2. move θ a bit into that direction ←
3. go to step 1



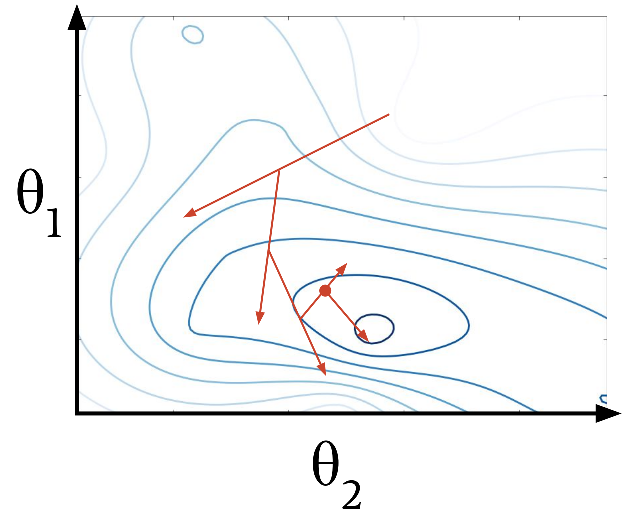
Optimization

4. **Optimize** model parameters to minimize loss

$$\theta^* = \min_\theta L(\theta; f, D)$$

Iterative scheme:

0. initialize θ randomly
1. find direction in which L decreases
2. move θ a bit into that direction
3. go to step 1



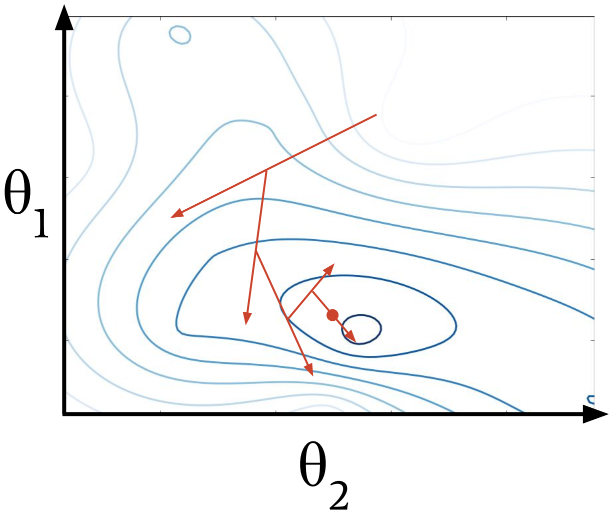
Optimization

4. **Optimize** model parameters to minimize loss

$$\theta^* = \min_\theta L(\theta; f, D)$$

Iterative scheme:

0. initialize θ randomly
1. find direction in which L decreases
2. ➜ move θ a bit into that direction
3. go to step 1



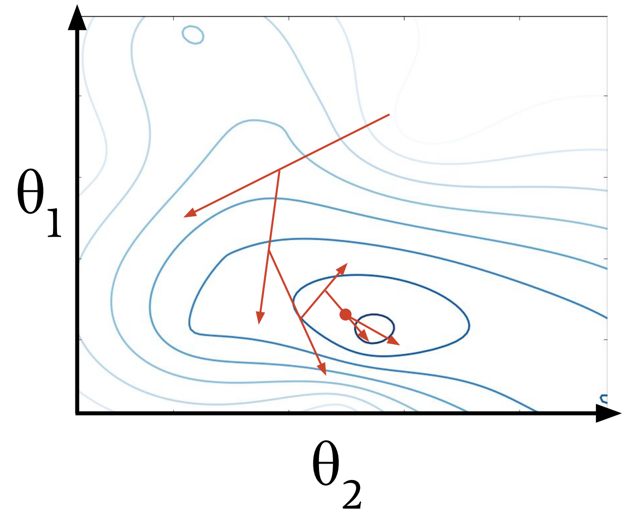
Optimization

4. **Optimize** model parameters to minimize loss

$$\theta^* = \min_\theta L(\theta; f, D)$$

Iterative scheme:

0. initialize θ randomly
1. find direction in which L decreases ←
2. move θ a bit into that direction
3. go to step 1



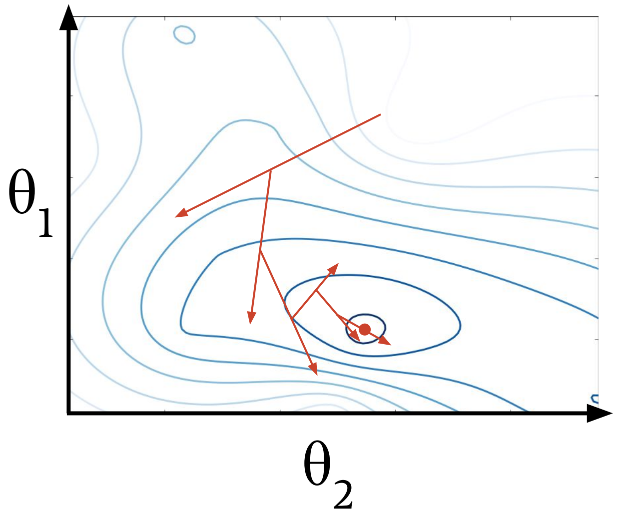
Optimization

4. **Optimize** model parameters to minimize loss

$$\theta^* = \min_\theta L(\theta; f, D)$$

Iterative scheme:

0. initialize θ randomly
1. find direction in which L decreases
2. ➡ move θ a bit into that direction
3. go to step 1



How to solve a task with deep learning

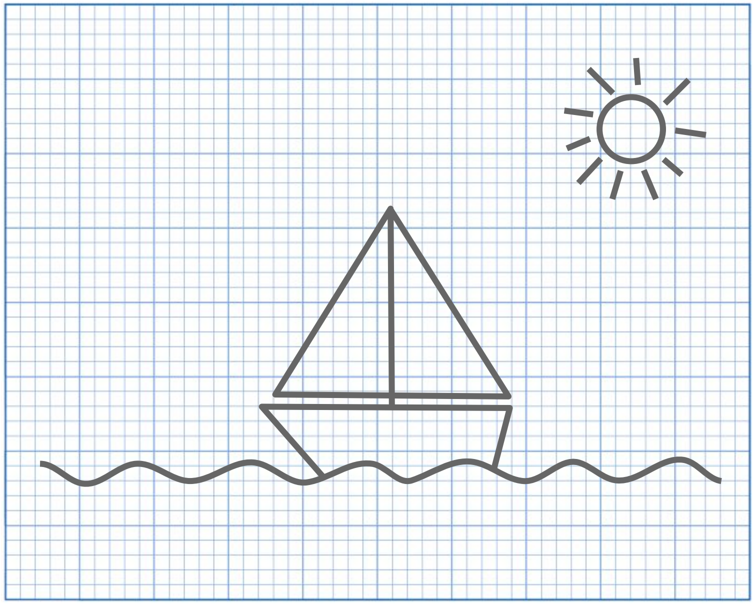
1. **Formalize task** so its solution can be expressed as a function
2. **Define model** as a generic solution with free parameters
3. **Define loss** function measuring how bad the solution is
4. **Optimize** model parameters to minimize loss

$$\mathbf{Y} = f(\mathbf{X}; \theta)$$

$$l = L(\theta; f, D) = \mathbf{\Sigma}_{(\mathbf{X}, \mathbf{T}) \in D} J(f(\mathbf{X}; \theta), \mathbf{T})$$

$$\theta^* = \min_\theta L(\theta; f, D)$$

Basic ideas behind ~~deep~~ ~~machine~~ deep learning



Deep learning in practice

Deep learning in practice

Optimization

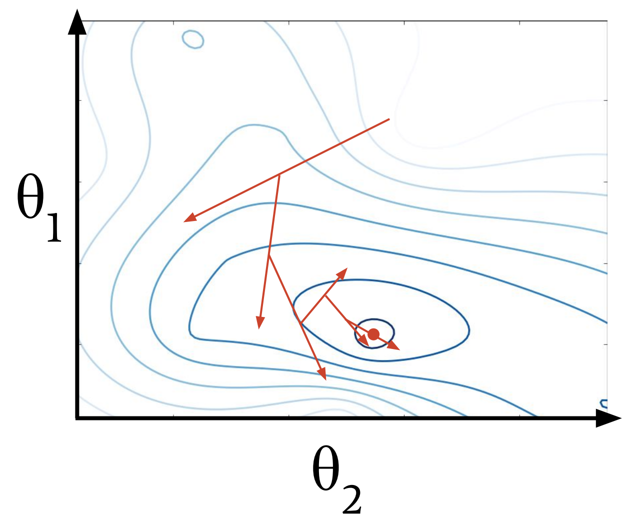
Optimization

4. **Optimize** model parameters to minimize loss

$$\theta^* = \min_\theta L(\theta; f, D)$$

Iterative scheme:

0. initialize θ randomly
1. find direction in which L decreases
2. move θ a bit into that direction
3. go to step 1

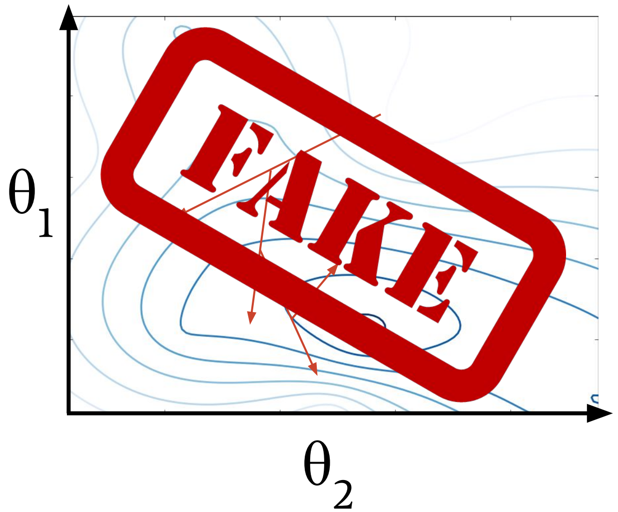


4. **Optimize** model parameters to minimize loss

$$\theta^* = \min_\theta L(\theta; f, D)$$

Iterative scheme:

0. initialize θ randomly
1. find direction in which L decreases
2. move θ a bit into that direction
3. go to step 1



Optimization

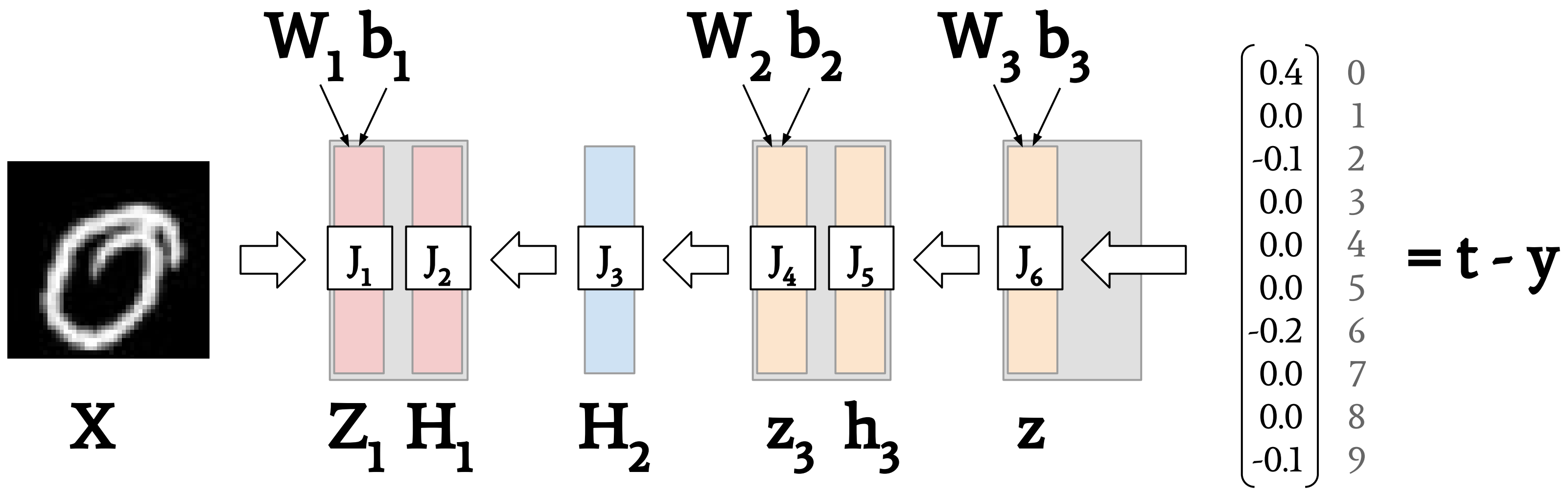
4. **Optimize** model parameters to minimize loss

$$\theta^* = \min_\theta L(\theta; f, D)$$

Iterative scheme:

0. initialize θ randomly
1. **find direction in which L decreases**
2. move θ a bit into that direction
3. go to step 1

Find direction in which the loss decreases



$\nabla z = t - y$

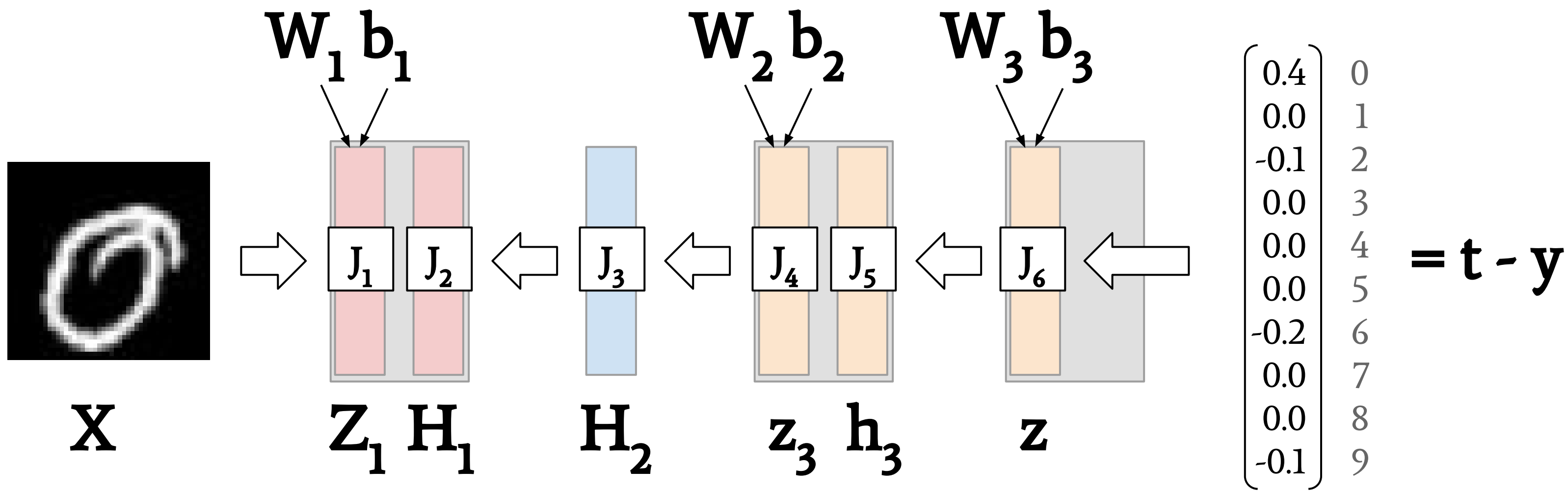
$\nabla b_3 = t - y$

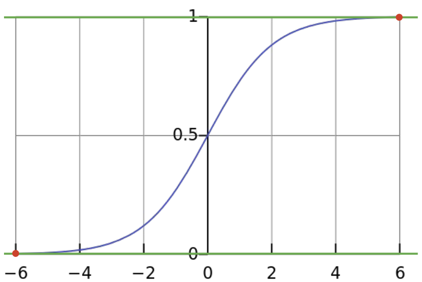
$\nabla W_3 = h_3 (t - y)^T$

$\nabla z_3 = J_5 J_6 (t - y)$

$\nabla Z_1 = J_2 J_3 J_4 J_5 J_6 (t - y)$

Find direction in which the loss decreases

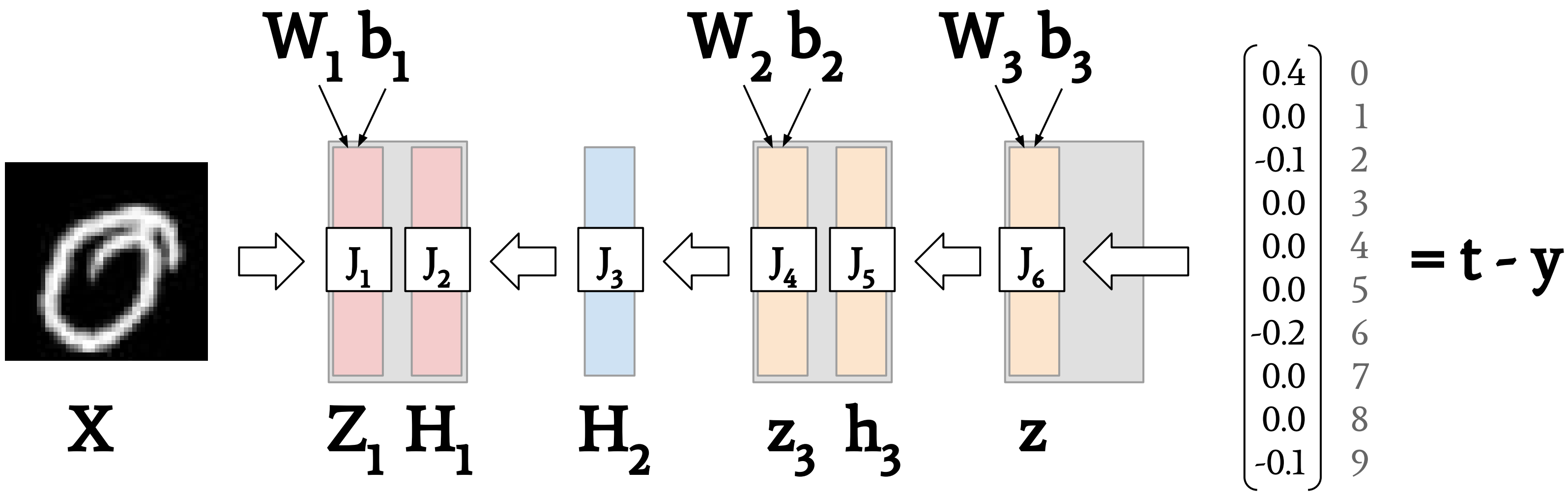


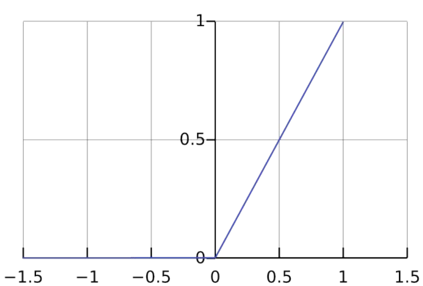


$$(J_5)_{i,i} = \sigma'((z_3)_i)$$

$$\nabla z = t - y$$
$$\nabla b_3 = t - y$$
$$\nabla W_3 = h_3 (t - y)^T$$
$$\nabla z_3 = J_5\, J_6\, (t - y)$$
$$\nabla Z_1 = J_2\, J_3\, J_4\, J_5\, J_6\, (t - y)$$

Find direction in which the loss decreases



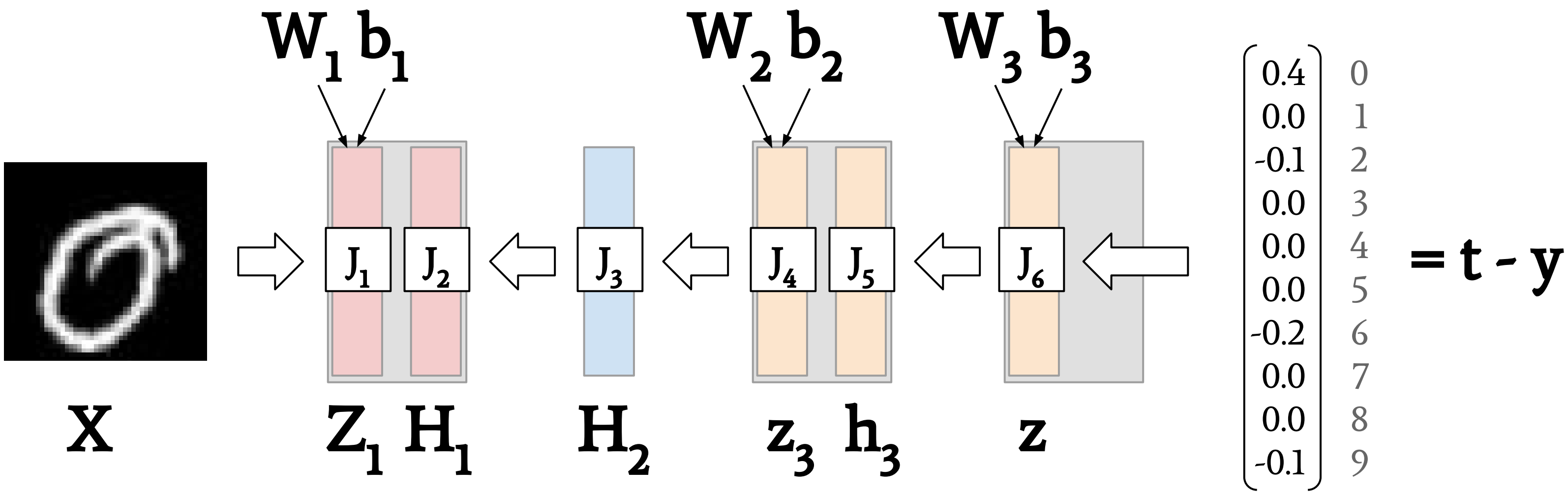


ReLU: max(x, 0)

$$(J_5)_{i,i} = [(z_3)_i > 0]$$

$$\nabla z = t - y$$
$$\nabla b_3 = t - y$$
$$\nabla W_3 = h_3 (t - y)^T$$
$$\nabla z_3 = J_5 J_6 (t - y)$$
$$\nabla Z_1 = J_2 J_3 J_4 J_5 J_6 (t - y)$$

Find direction in which the loss decreases



$J_6 = W_3$

$J_4 = W_2$

J_1 = "mumble W_1 mumble mumble"

$\nabla z = t - y$

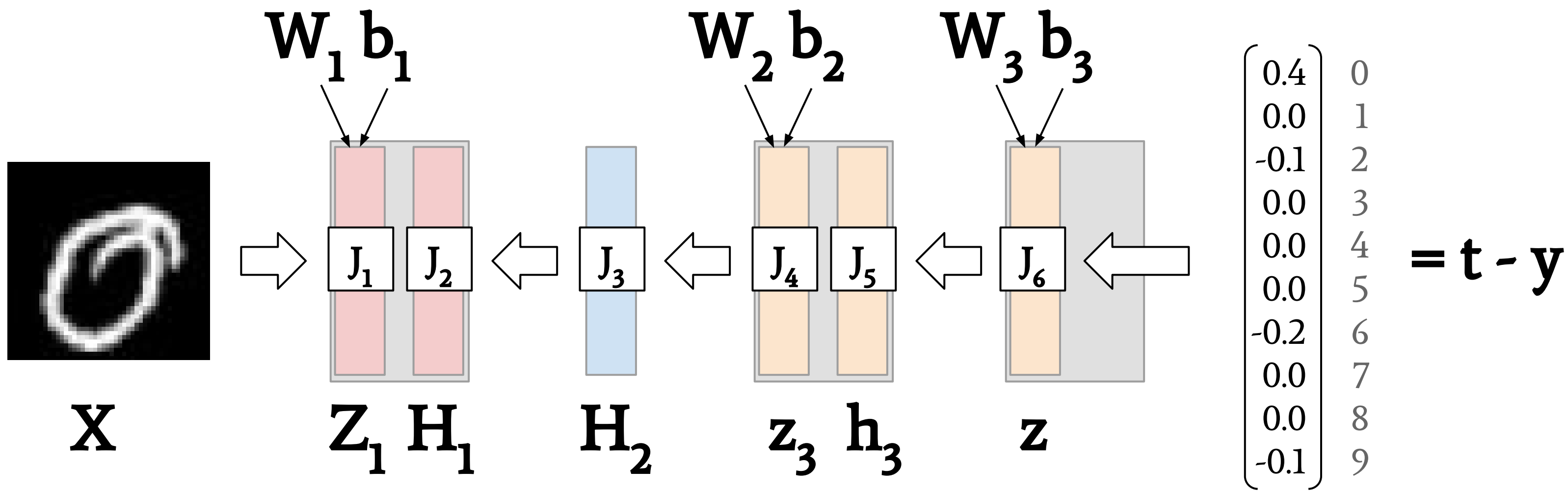
$\nabla b_3 = t - y$

$\nabla W_3 = h_3 (t - y)^T$

$\nabla z_3 = J_5 J_6 (t - y)$

$\nabla Z_1 = J_2 J_3 J_4 J_5 J_6 (t - y)$

Find direction in which the loss decreases



Problem:
Depending on $\mathbf{W}_1$, $\mathbf{W}_2$, $\mathbf{W}_3$,
$\nabla\mathbf{Z}_1$ may become very small
("vanishing gradient")
or large ("exploding gradient")

$\nabla\mathbf{z} = \mathbf{t} - \mathbf{y}$

$\nabla\mathbf{b}_3 = \mathbf{t} - \mathbf{y}$

$\nabla\mathbf{W}_3 = \mathbf{h}_3(\mathbf{t} - \mathbf{y})^T$

$\nabla\mathbf{z}_3 = J_5\, J_6\, (\mathbf{t} - \mathbf{y})$

$\nabla\mathbf{Z}_1 = J_2\, J_3\, J_4\, J_5\, J_6\, (\mathbf{t} - \mathbf{y})$

Initialization

Problem:
Depending on θ, $-\frac{\partial}{\partial\theta} J(f(\mathbf{X};\theta),\mathbf{t})$ may become very small ("vanishing gradient") or large ("exploding gradient").

Initialization

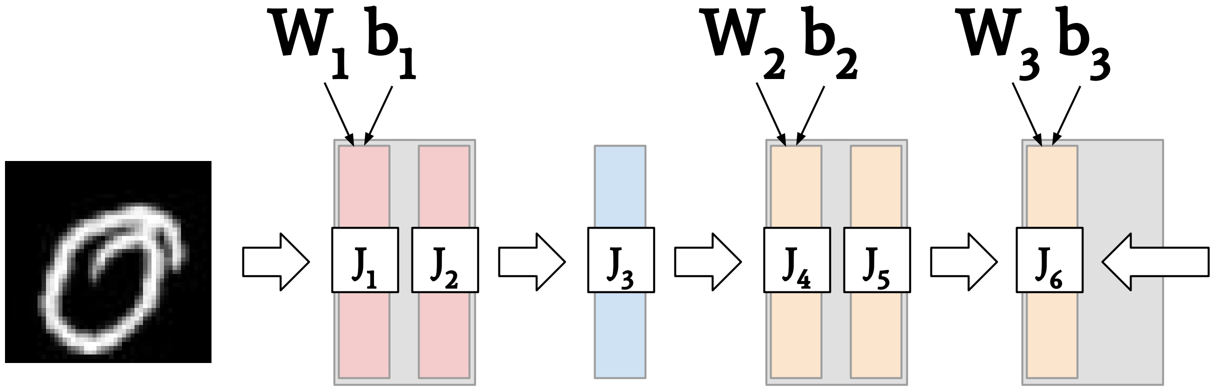
Problem:

Depending on θ, $-\frac{\partial}{\partial\theta} J(f(\mathbf{X};\theta),\mathbf{t})$ may become very small ("vanishing gradient") or large ("exploding gradient").

Iterative scheme:

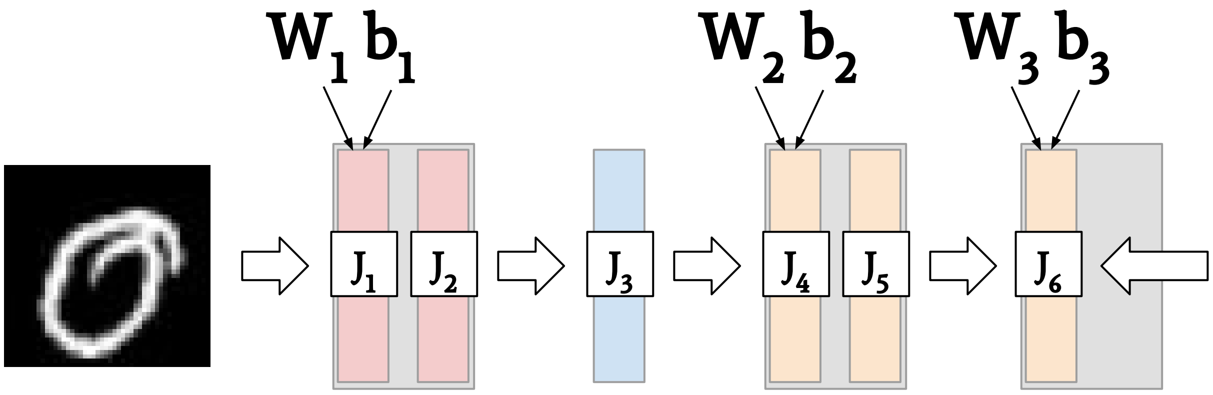
0. **initialize θ randomly**
1. find direction in which L decreases
2. move θ a bit into that direction
3. go to step 1

Initialization



2006: Initialize weights with unsupervised pretraining

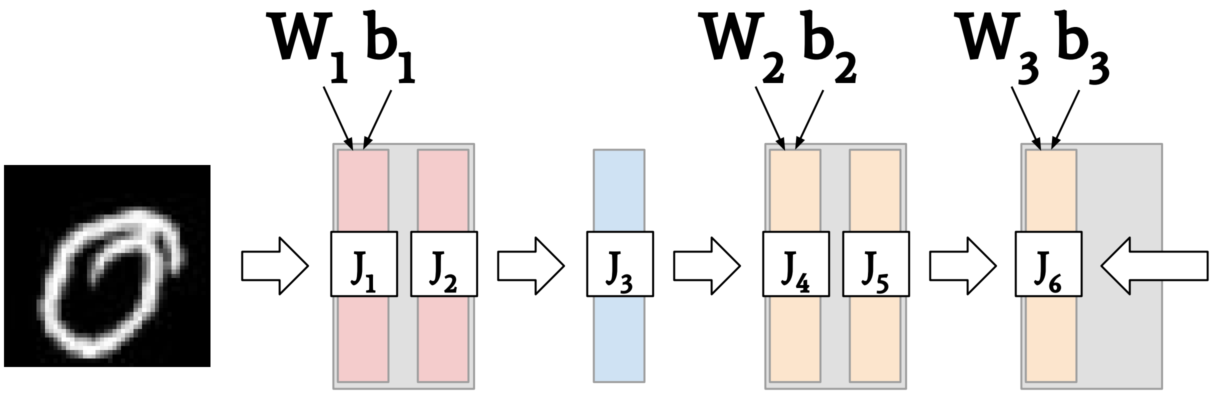
Initialization



2006: Initialize weights with unsupervised pretraining

2010: Initialize randomly, scaled to preserve variance of Gaussian inputs and/or gradients (Glorot 2010; He 2015)

Initialization

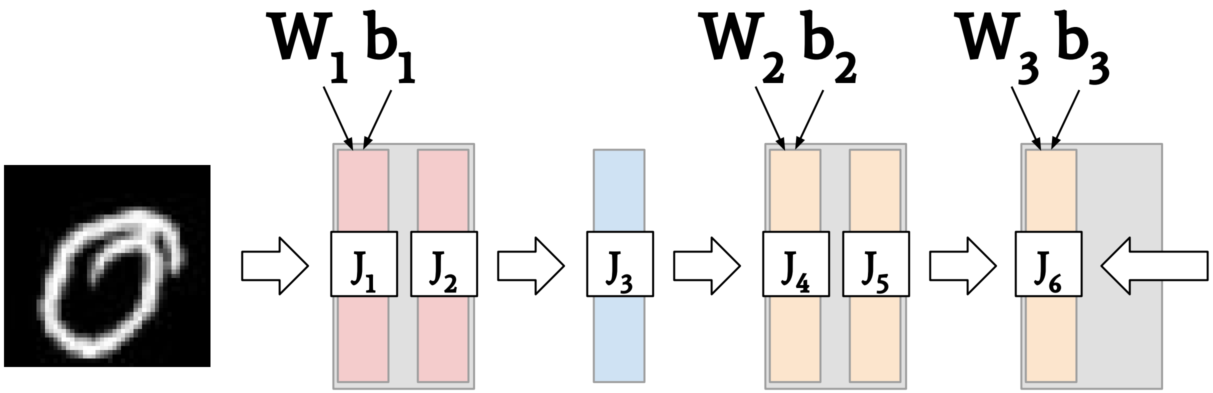


2006: Initialize weights with unsupervised pretraining

2010: Initialize randomly, scaled to preserve variance of Gaussian inputs and/or gradients (Glorot 2010; He 2015)

2014: Random, variance-preserving, orthogonal (against skewed distribution of singular values of Jacobian; Saxe 2014)

Initialization



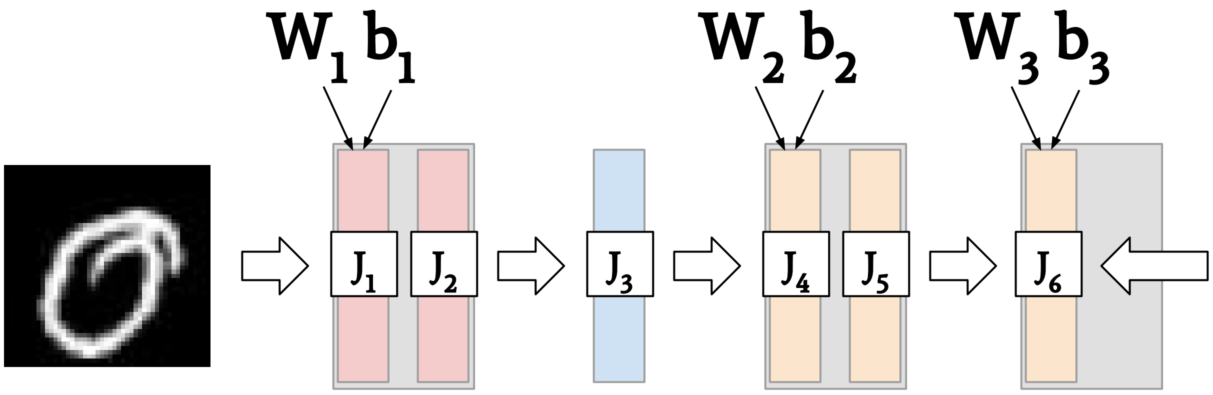
2006: Initialize weights with unsupervised pretraining

2010: Initialize randomly, scaled to preserve variance of Gaussian inputs and/or gradients (Glorot 2010; He 2015)

2014: Random, variance-preserving, orthogonal (against skewed distribution of singular values of Jacobian; Saxe 2014)

2016: Initialize randomly, scaled by observed variance of actual training data at each layer (Krähenbühl; Mishkins; Salima)

Initialization



simple formula, most common

2006: Initialize weights with unsupervised pretraining

2010: Initialize randomly, scaled to preserve variance of Gaussian inputs and/or gradients (Glorot 2010; He 2015)

2014: Random, variance-preserving, orthogonal (against skewed distribution of singular values of Jacobian; Saxe 2014)

2016: Initialize randomly, scaled by observed variance of actual training data at each layer (Krähenbühl; Mishkins; Salima)

Optimization

4. **Optimize** model parameters to minimize loss

$$\theta^* = \min_\theta L(\theta; f, D)$$

Iterative scheme:

0. **initialize θ randomly**
1. find direction in which L decreases
2. move θ a bit into that direction
3. go to step 1

Optimization

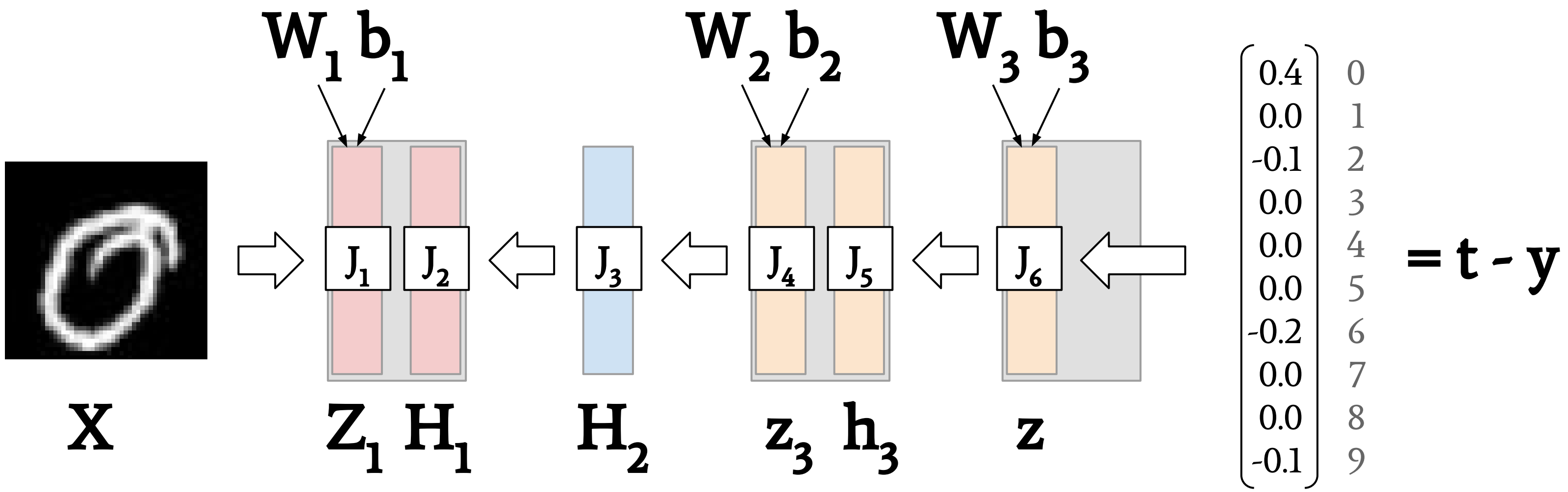
4. **Optimize** model parameters to minimize loss

$$\theta^* = \min_\theta L(\theta; f, D)$$

Iterative scheme:

0. initialize θ randomly
1. **find direction in which L decreases**
2. move θ a bit into that direction
3. go to step 1

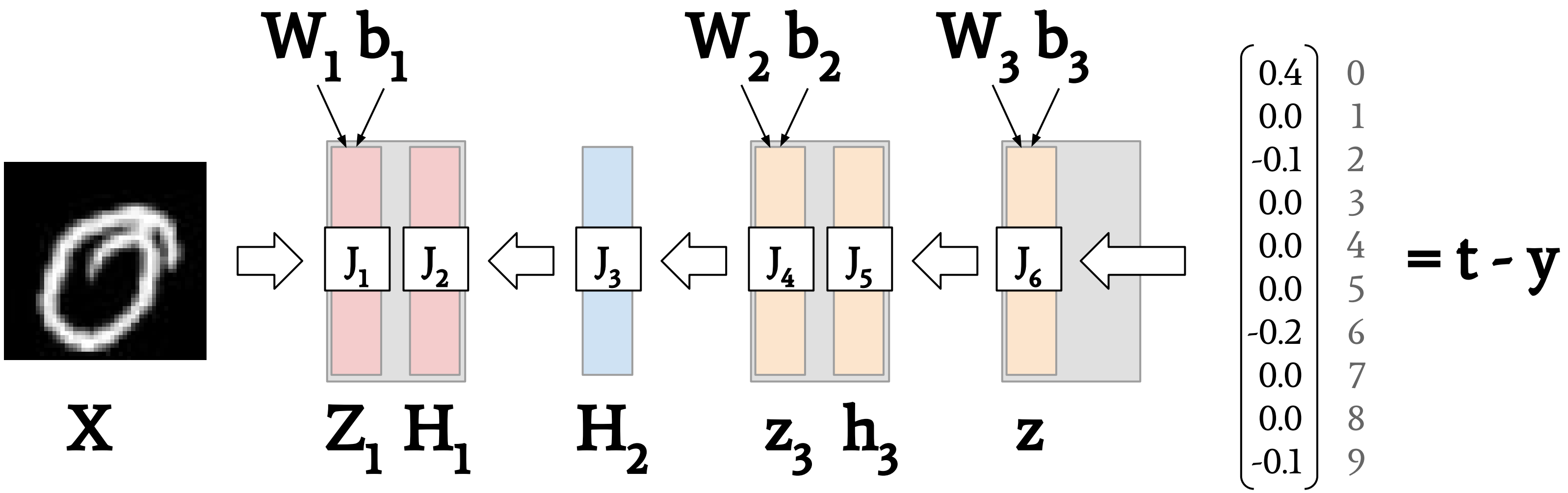
Find direction in which the loss decreases



$$\nabla\theta = -\frac{\partial}{\partial\theta} J(f(\mathbf{X};\theta), \mathbf{t})$$

$$-\frac{\partial}{\partial\theta} L(\theta; f, D) = -\boldsymbol{\Sigma}_{(\mathbf{X},\mathbf{T}) \in D} \frac{\partial}{\partial\theta} J(f(\mathbf{X};\theta), \mathbf{T})$$

Find direction in which the loss decreases



$$\nabla\theta = -\frac{\partial}{\partial\theta} J(f(\mathbf{X};\theta),\mathbf{t})$$

$$-\frac{\partial}{\partial\theta} L(\theta; f, D) \approx -\sum_{(\mathbf{X},\mathbf{T})\in D'} \frac{\partial}{\partial\theta} J(f(\mathbf{X};\theta),\mathbf{T}) \quad \text{where } D' \subset D$$

Optimization

4. **Optimize** model parameters to minimize loss

$$\theta^* = \min_\theta L(\theta; f, D)$$

Iterative scheme:

0. initialize θ randomly
1. **find direction in which L decreases**
2. move θ a bit into that direction
3. go to step 1

Optimization

4. **Optimize** model parameters to minimize loss

$$\theta^* = \min_\theta L(\theta; f, D)$$

Iterative scheme:

0. initialize θ randomly
1. find direction in which L decreases
2. **move θ a bit into that direction**
3. go to step 1

SGD+

Stochastic Gradient Descent (SGD):

$$\theta \leftarrow \theta - \eta \frac{\partial L}{\partial \theta}$$

Take small step in direction of negative gradient.

Analogy: Somebody walking among hills,
always in direction of steepest descent.

How far to move?

SGD+

Stochastic Gradient Descent (SGD):

$$\theta \leftarrow \theta - \eta \frac{\partial L}{\partial \theta}$$

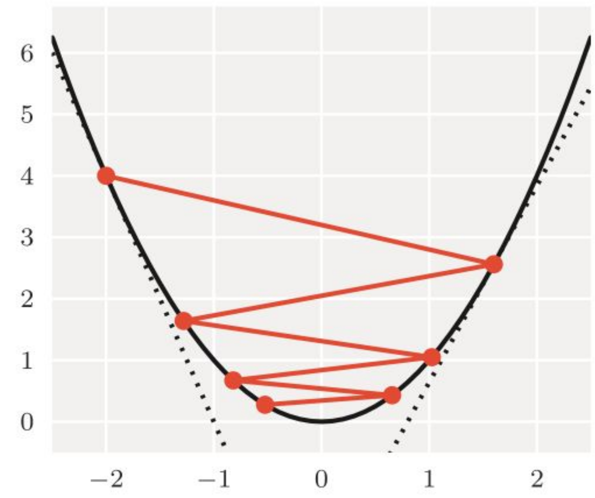
Take small step in direction of negative gradient.

Analogy: Somebody walking among hills, always in direction of steepest descent.

How far to move?

Too small η: slow progress

Too large η: oscillation or divergence



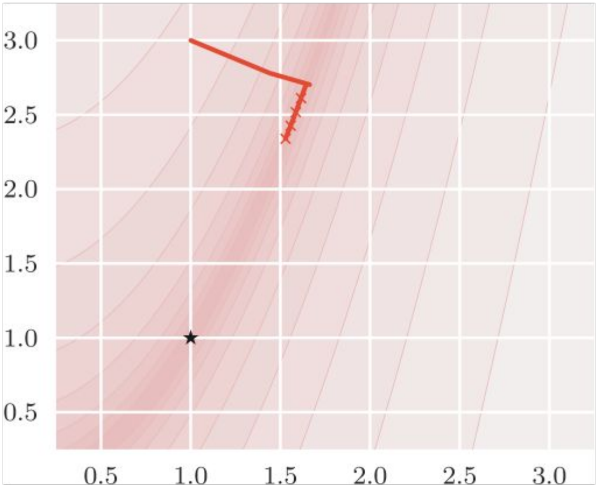
SGD+

Stochastic Gradient Descent (SGD):

$$\theta \leftarrow \theta - \eta \frac{\partial L}{\partial \theta}$$

Take small step in direction of negative gradient.

Analogy: Somebody walking among hills, always in direction of steepest descent.



How far to move?

Too small η: slow progress

Too large η: oscillation or divergence

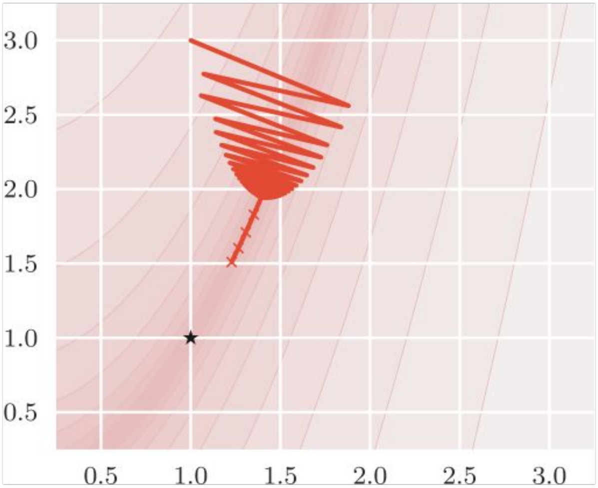
SGD+

Stochastic Gradient Descent (SGD):

$$\theta \leftarrow \theta - \eta \frac{\partial L}{\partial \theta}$$

Take small step in direction of negative gradient.

Analogy: Somebody walking among hills, always in direction of steepest descent.



How far to move?

Too small η: slow progress

Too large η: oscillation or divergence

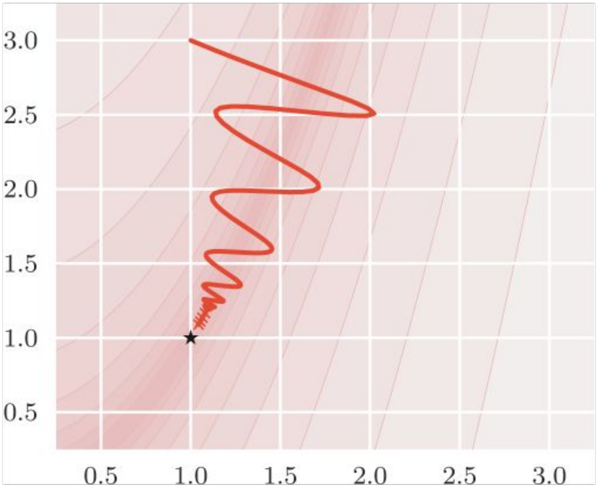
SGD+

Stochastic Gradient Descent (SGD) with Momentum:

$$v \leftarrow \alpha v - \eta \frac{\partial L}{\partial \theta}$$

$$\theta \leftarrow \theta + v$$

Dampen velocity according to friction coefficient α (e.g., 0.9).
Increase velocity in direction of negative gradient.
Move according to velocity.



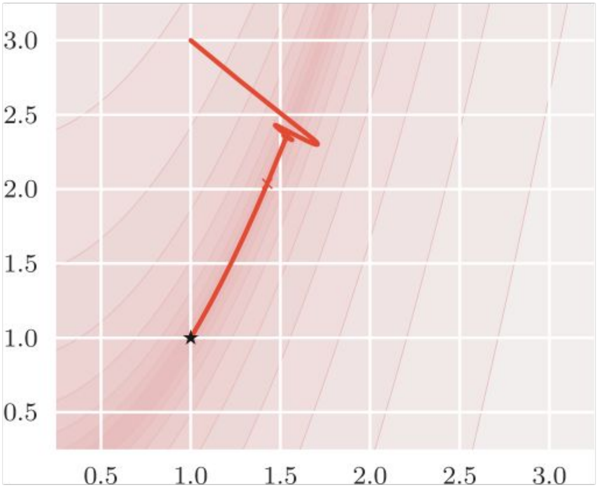
Analogy: Ball rolling down hills.

SGD+

Adam (Adaptive Moment Estimation):

- Compute **velocity (first moment)**:
  exponential moving average over past gradients (as before)
- Compute **second moment estimate**:
  exponential moving average over past gradient magnitudes
- Move according to velocity, **divided by second moment**

Intuition: counter notoriously small gradients by upscaling, and large gradients by downscaling, separately for each weight



ICLR 2015: Adam: A Method for Stochastic Optimization

Optimization

4. **Optimize** model parameters to minimize loss

$$\theta^* = \min_\theta L(\theta; f, D)$$

Iterative scheme:

0. initialize θ randomly
1. find direction in which L decreases
2. **move θ a bit into that direction**
3. go to step 1

Optimization

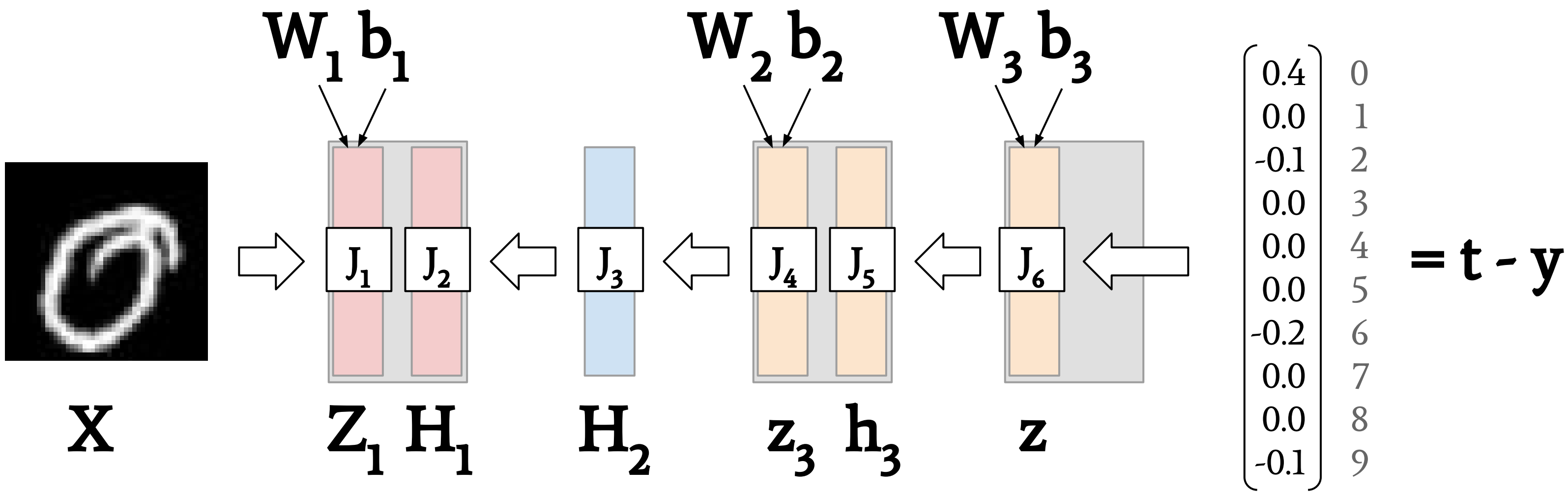
4. **Optimize** model parameters to minimize loss

$$\theta^* = \min_\theta L(\theta; f, D)$$

Iterative scheme:

0. initialize θ randomly
1. **find direction in which L decreases**
2. move θ a bit into that direction
3. go to step 1

Find direction in which the loss decreases



Problem:
Depending on $\mathbf{W_1}$, $\mathbf{W_2}$, $\mathbf{W_3}$,
$\nabla \mathbf{Z_1}$ may become very small
("vanishing gradient")
or large ("exploding gradient")

$\nabla \mathbf{z} = \mathbf{t} - \mathbf{y}$

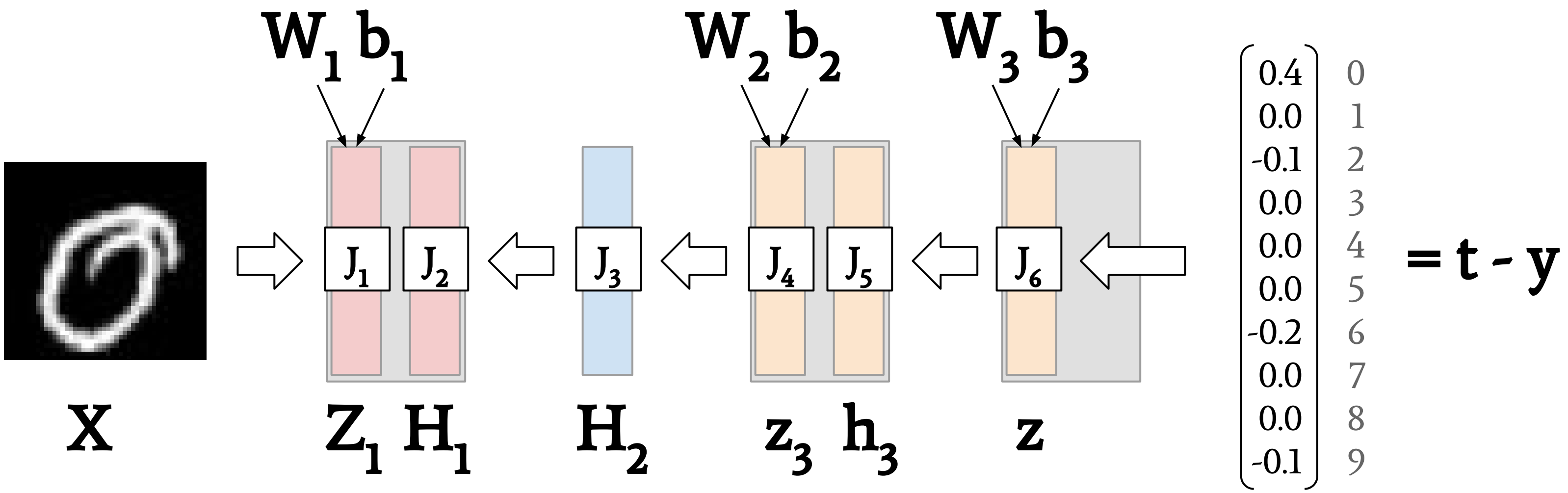
$\nabla \mathbf{b_3} = \mathbf{t} - \mathbf{y}$

$\nabla \mathbf{W_3} = \mathbf{h_3}(\mathbf{t} - \mathbf{y})^T$

$\nabla \mathbf{z_3} = J_5 J_6 (\mathbf{t} - \mathbf{y})$

$\nabla \mathbf{Z_1} = J_2 J_3 J_4 J_5 J_6 (\mathbf{t} - \mathbf{y})$

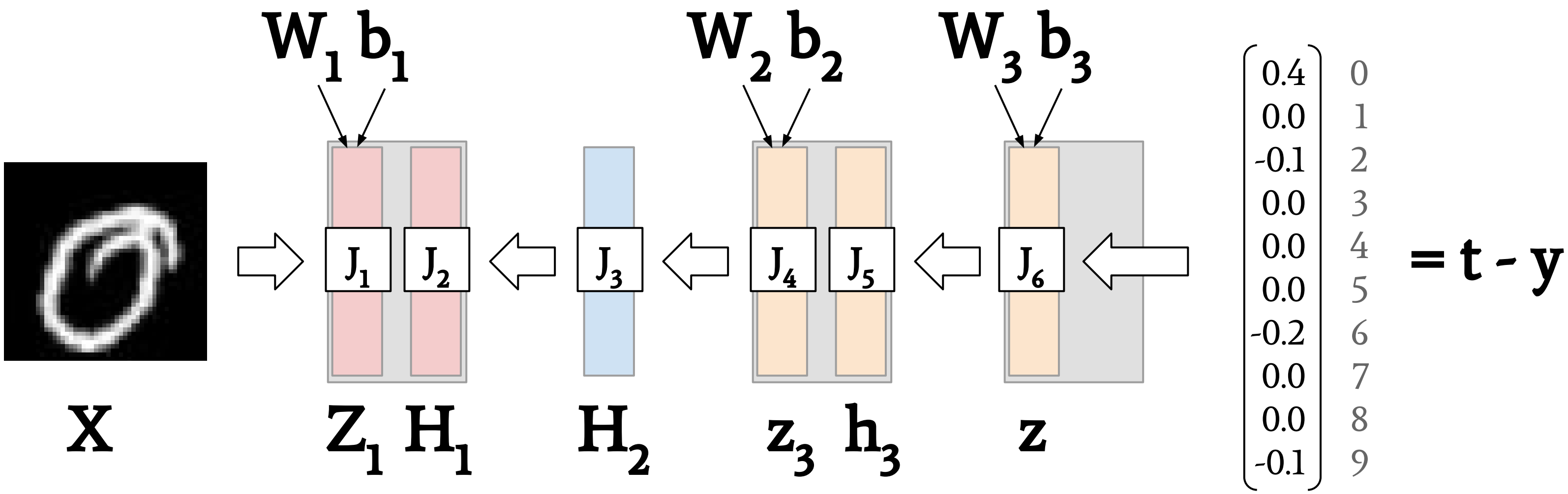
Gradient clipping



Possible solution:
Scale/clip ∇z, ∇h_3, ∇z_3, ∇H_1, ∇Z_1 when they become too large.

$$\nabla z = t - y$$
$$\nabla b_3 = t - y$$
$$\nabla W_3 = h_3 (t - y)^T$$
$$\nabla z_3 = J_5 J_6 (t - y)$$
$$\nabla Z_1 = J_2 J_3 J_4 J_5 J_6 (t - y)$$

Unitary weights

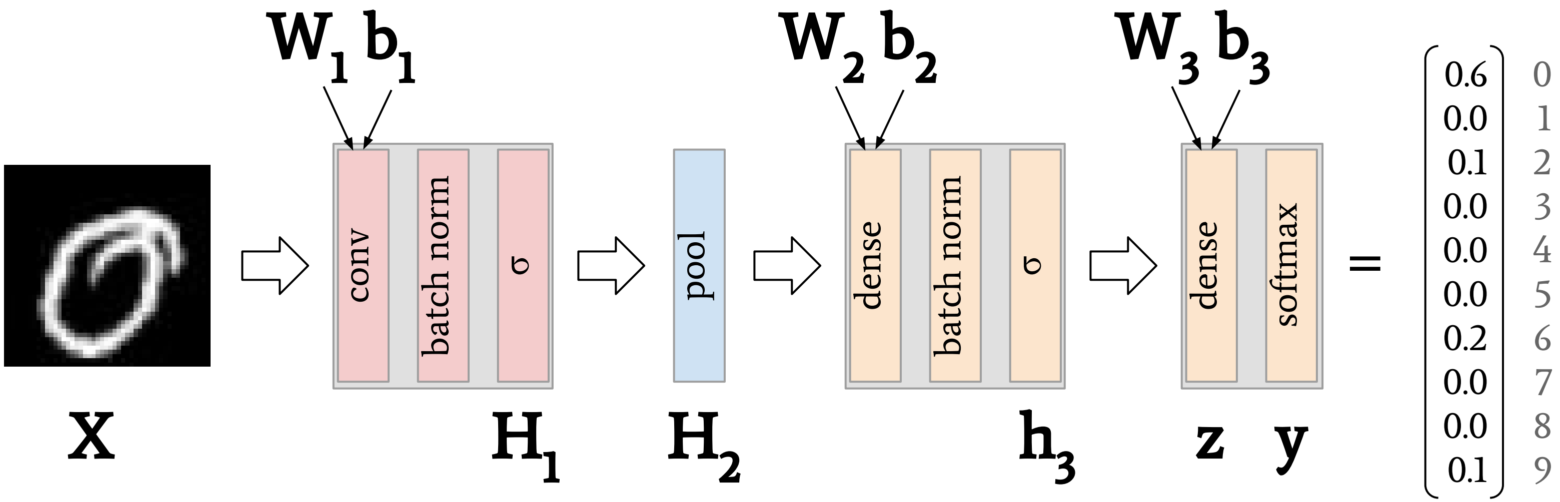


Possible solution:
Parameterize $\mathbf{W_1}$, $\mathbf{W_2}$, $\mathbf{W_3}$ such that they always stay orthogonal matrices.

$$\nabla \mathbf{z} = \mathbf{t} - \mathbf{y}$$
$$\nabla \mathbf{b_3} = \mathbf{t} - \mathbf{y}$$
$$\nabla \mathbf{W_3} = \mathbf{h_3}(\mathbf{t} - \mathbf{y})^T$$
$$\nabla \mathbf{z_3} = J_5 \, J_6 \, (\mathbf{t} - \mathbf{y})$$
$$\nabla \mathbf{Z_1} = J_2 \, J_3 \, J_4 \, J_5 \, J_6 \, (\mathbf{t} - \mathbf{y})$$

abs/1707.09520: Orthogonal Recurrent Neural Networks with Scaled Cayley Transform

Batch normalization



Possible solution:

Normalize to zero mean / unit variance after every layer

- learn scale and bias on top to not lose expressivity
- estimate mean / variance on minibatch, not full dataset
- use fixed statistics after training
- backpropagate error through mean / variance computation

Optimization

4. **Optimize** model parameters to minimize loss

$$\theta^* = \min_\theta L(\theta; f, D)$$

Iterative scheme:

0. initialize θ randomly
1. find direction in which L decreases
2. move θ a bit into that direction
3. go to step 1

Deep learning in practice

Batch normalization

Initialization

SGD+

Optimization

Deep learning in practice

Batch normalization

Initialization — SGD+ — Batch normalization → Optimization

Generalization

Generalization

4. **Optimize** model parameters to minimize loss

$$\theta^* = \min_\theta L(\theta; f, D)$$

What we get:

$f(\mathbf{X}; \theta) \approxeq \mathbf{T}$ for all $(\mathbf{X}, \mathbf{T}) \in D$

Generalization

4. **Optimize** model parameters to minimize loss

$$\theta^* = \min_\theta L(\theta; f, D)$$

What we get:

$f(\mathbf{X}; \theta) \approxeq \mathbf{T}$ for all $(\mathbf{X}, \mathbf{T}) \in D$

What we wanted:

$f(\mathbf{X}; \theta) \approxeq \mathbf{T}$ for all $(\mathbf{X}, \mathbf{T}) \notin D$ (but from the same task)

Generalization

4. **Optimize** model parameters to minimize loss

$$\theta^* = \min_\theta L(\theta; f, D)$$

What we get:

$f(\mathbf{X}; \theta) \approxeq \mathbf{T}$ for all $(\mathbf{X}, \mathbf{T}) \in D$

What we wanted:

$f(\mathbf{X}; \theta) \approxeq \mathbf{T}$ for all $(\mathbf{X}, \mathbf{T}) \notin D$ (but from the same task)

Problem:

There exist θ that fulfil the first, but not the second.

Generalization

4. **Optimize** model parameters to minimize loss

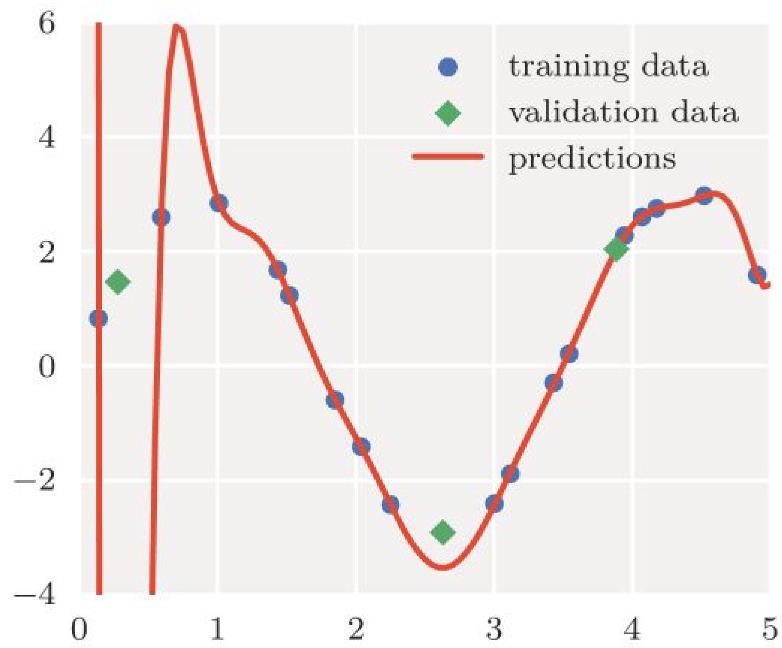
$$\theta^* = \min_\theta L(\theta; f, D)$$

What we get:

$f(\mathbf{X}; \theta) \approxeq \mathbf{T}$ for all $(\mathbf{X}, \mathbf{T}) \in D$

What we wanted:

$f(\mathbf{X}; \theta) \approxeq \mathbf{T}$ for all $(\mathbf{X}, \mathbf{T}) \notin D$



Problem:

There exist θ that fulfil the first, but not the second. → *overfitting*

Generalization

4. **Optimize** model parameters to minimize loss

$$\theta^* = \min_\theta L(\theta; f, D)$$

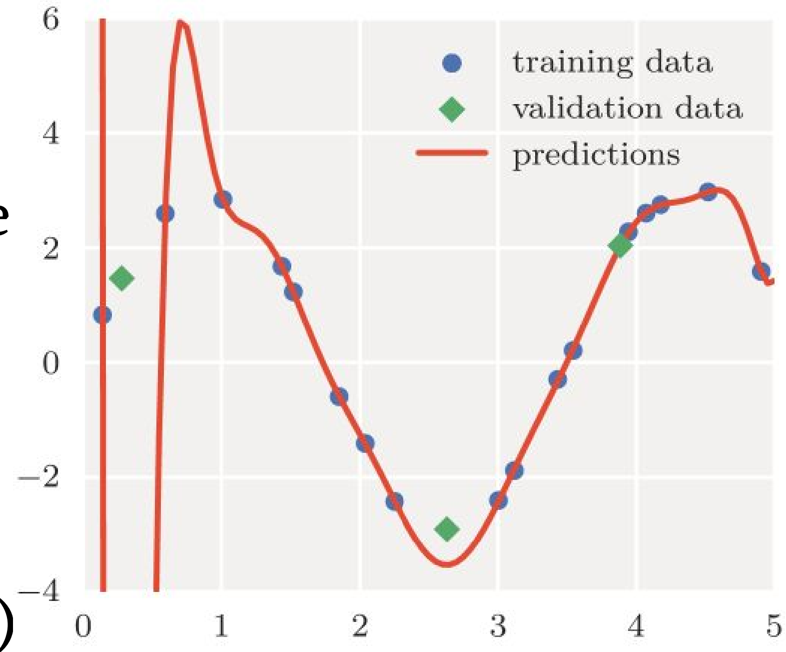
Goal:

Modify optimization to avoid solutions θ that only match the training examples.

Weight decay

Goal: Modify optimization to avoid solutions θ that only match the training examples.

Observation: Learning examples by heart often requires large jumps in the function = large gradients = large coefficients multiplied with inputs

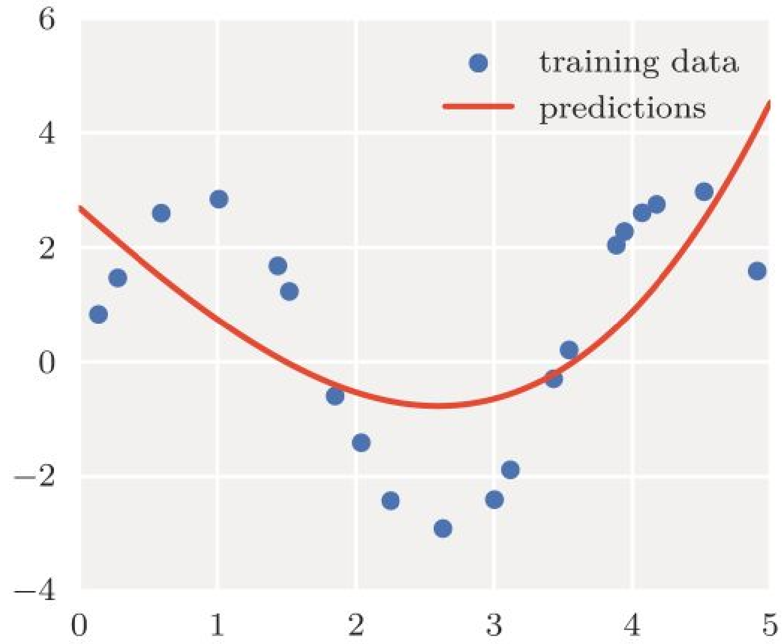


Countermeasure: Shrink weights after each update (= L2 decay), or whenever too large (weight clipping)

Early stopping

Goal: Modify optimization to avoid solutions θ that only match the training examples.

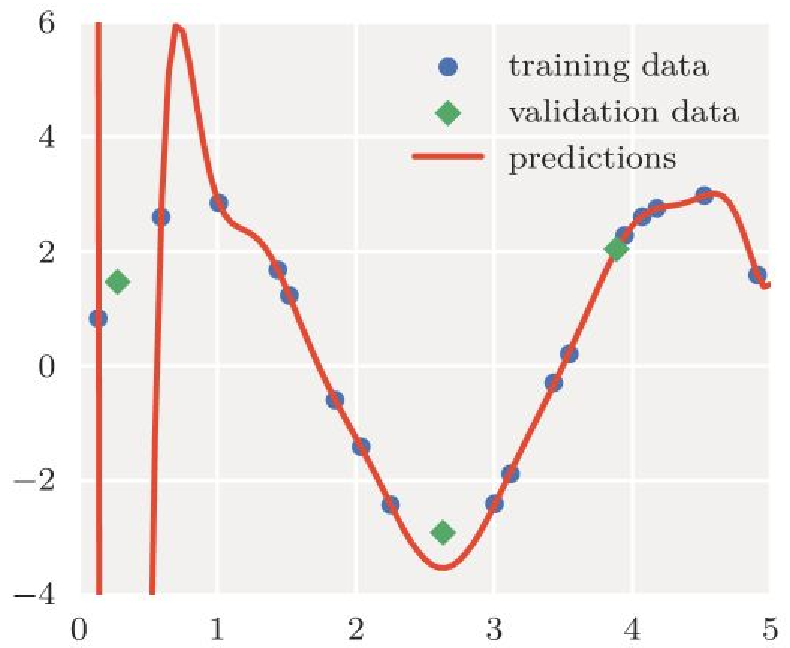
Observation: Training is iterative.
Initial model underfits.



Early stopping

Goal: Modify optimization to avoid solutions θ that only match the training examples.

Observation: Training is iterative.
Initial model underfits.
Final model overfits.

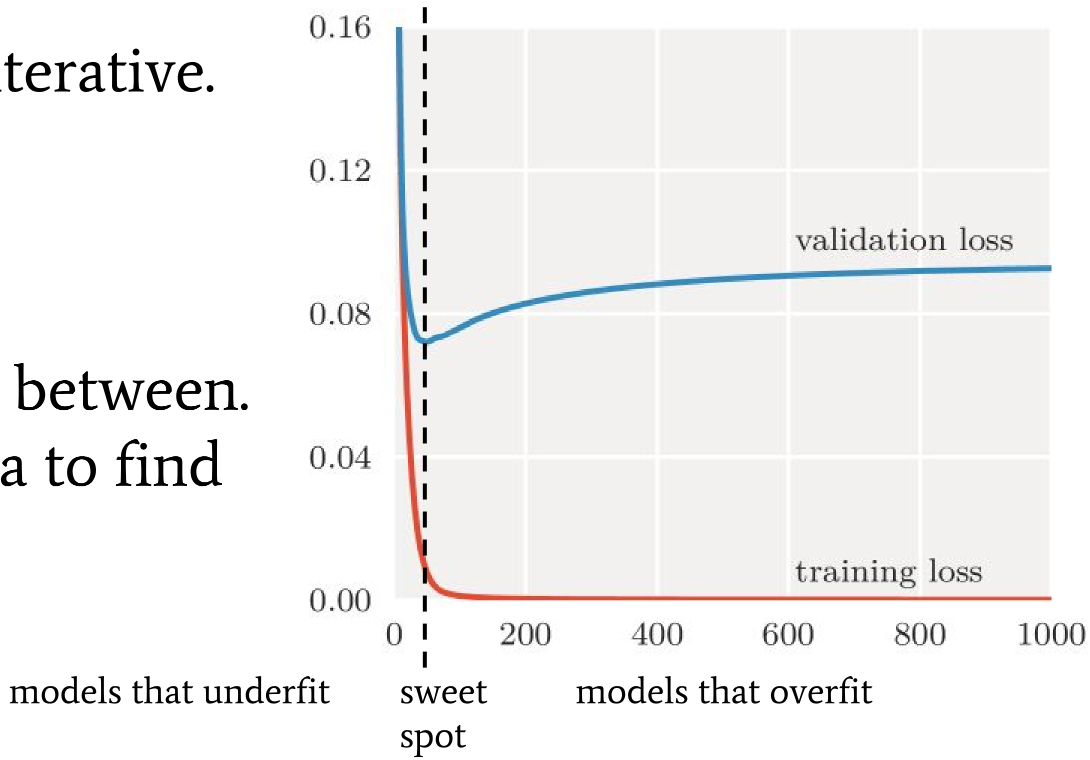


Early stopping

Goal: Modify optimization to avoid solutions θ that only match the training examples.

Observation: Training is iterative.
Initial model underfits.
Final model overfits.

Solution: Stop training in between.
Monitor loss on extra data to find sweet spot.



Data augmentation

Goal: Modify optimization to avoid solutions θ that only match the training examples.

Observation: Overfitting may mean the solution depends on irrelevant properties of the input.

cat facing left

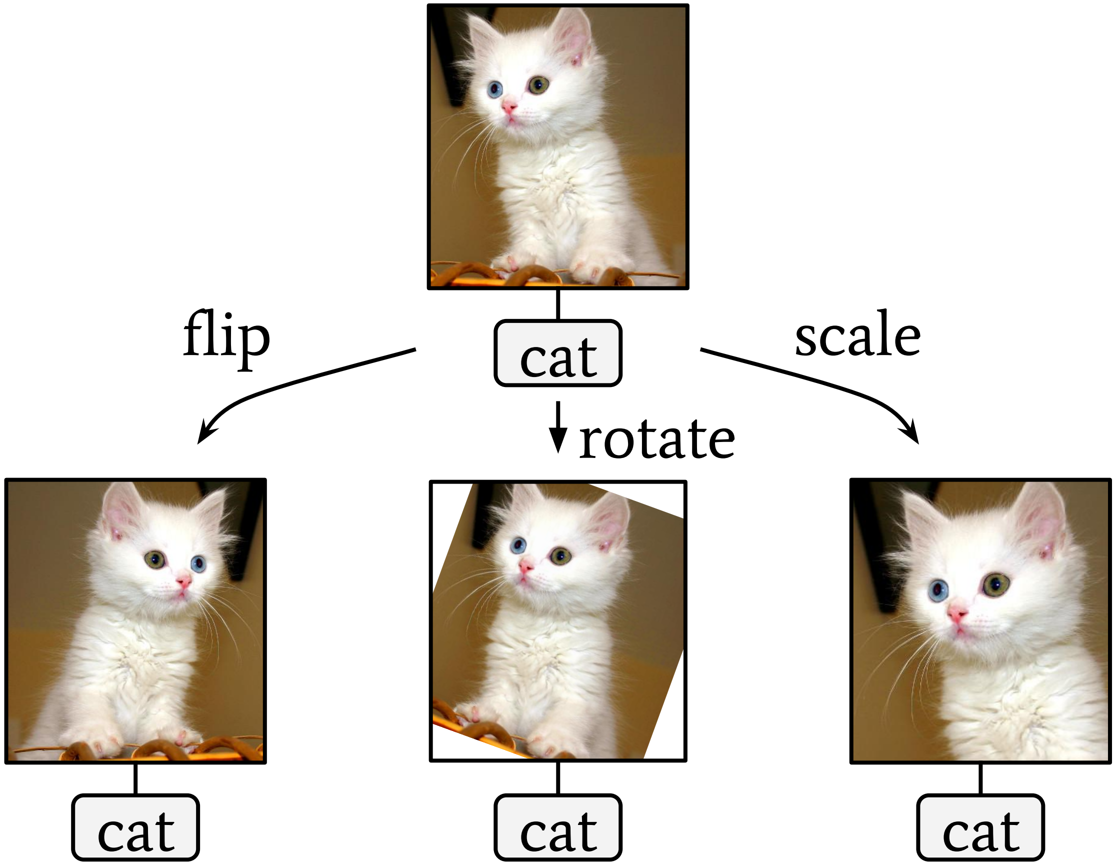
cat facing right

Possible solutions:

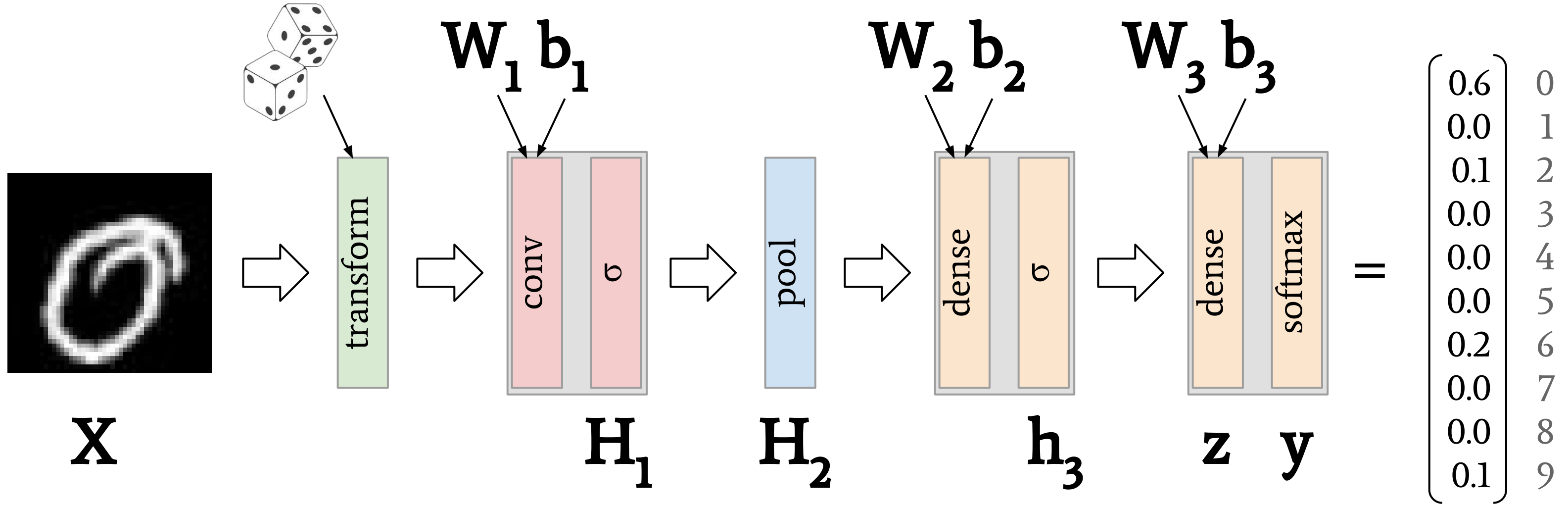
- More data
- Design invariant model
- **Data augmentation**

Data augmentation

Data augmentation:
Transform training data, let classifier learn to ignore it.



Data augmentation:
Transform training data, let classifier learn to ignore it.



Typical transformations:

- For images: horizontal flip, scale, rotation, color, contrast
- For audio: time stretching, pitch shifting, equalizer

Dropout

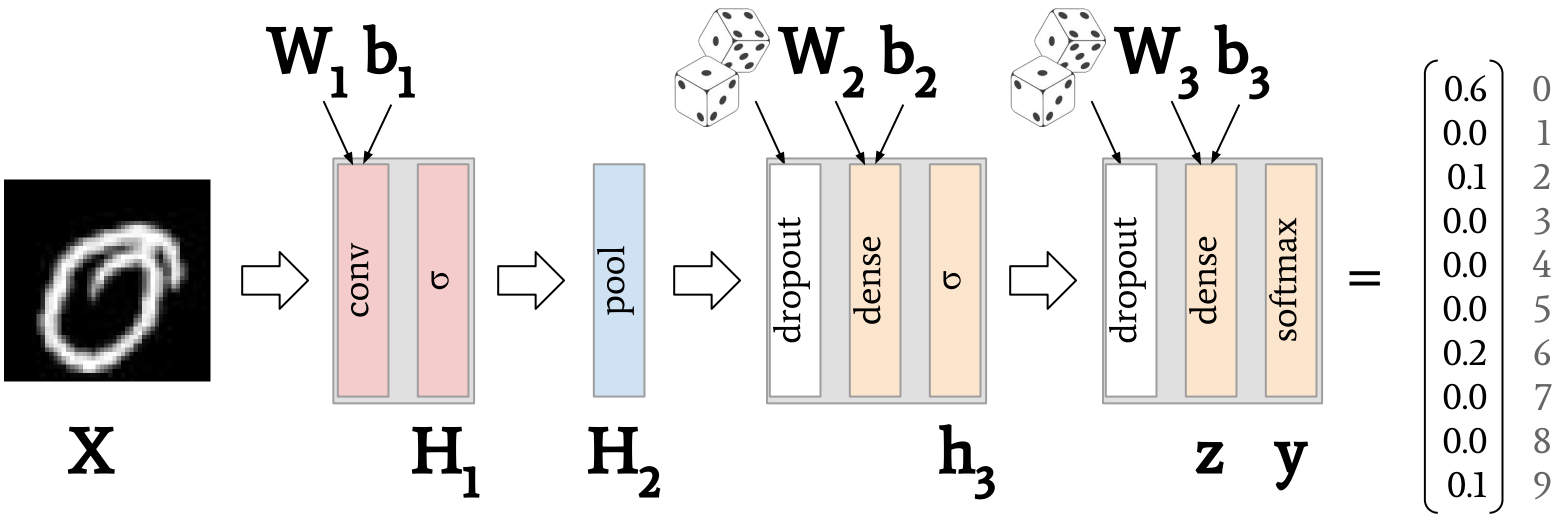
Goal: Modify optimization to avoid solutions θ that only match the training examples.

Observation: Units can learn to focus on few units in previous layer to distinguish training examples.

Solution: Drop 50% of hidden units for each training example. Scale up weights by 2.0 to compensate.

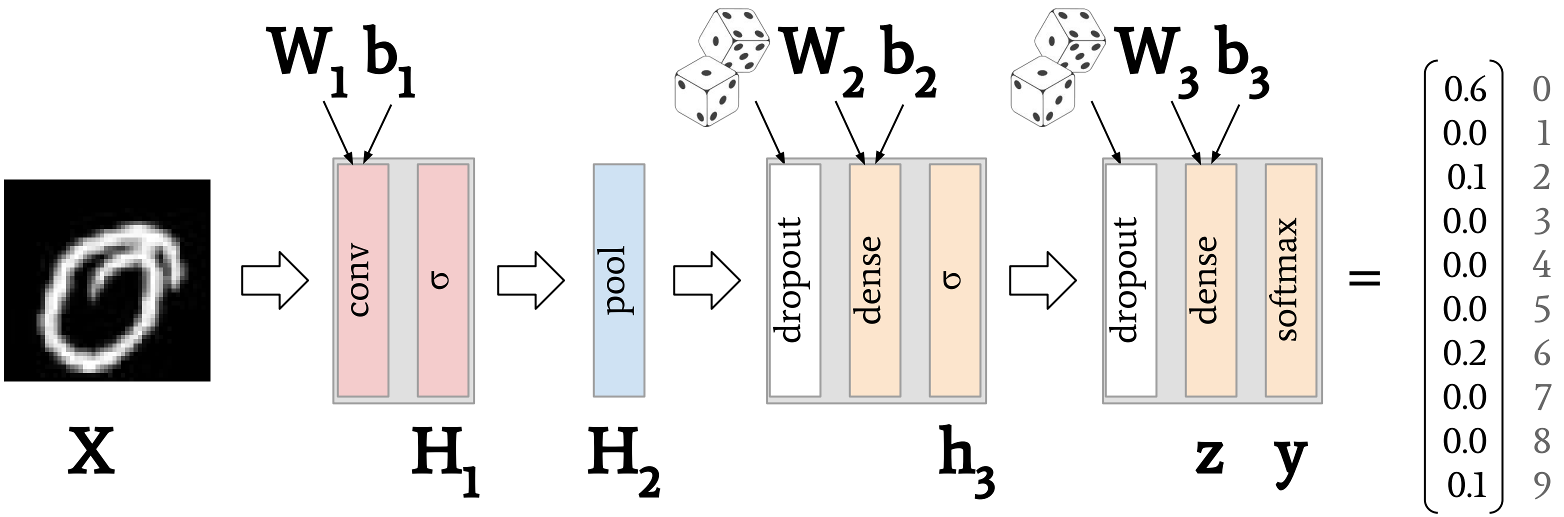
Dropout

Solution: Drop 50% of hidden units for each training example.
Scale up weights by 2.0 to compensate.



Dropout

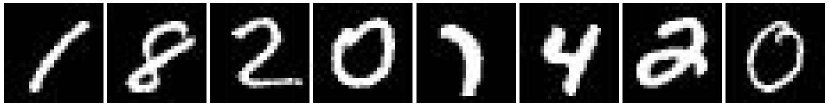
Solution: Drop 50% of hidden units for each training example.
Scale up weights by 2.0 to compensate.



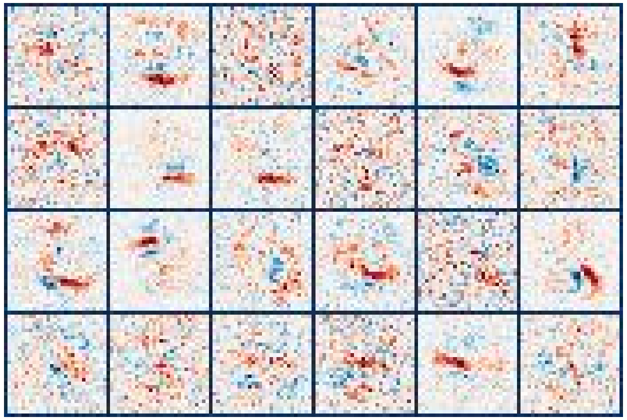
At test time, do not drop any units (and do not scale up weights).
Can be interpreted as an ensemble of 2^N networks trained simultaneously with shared weights.

Dropout

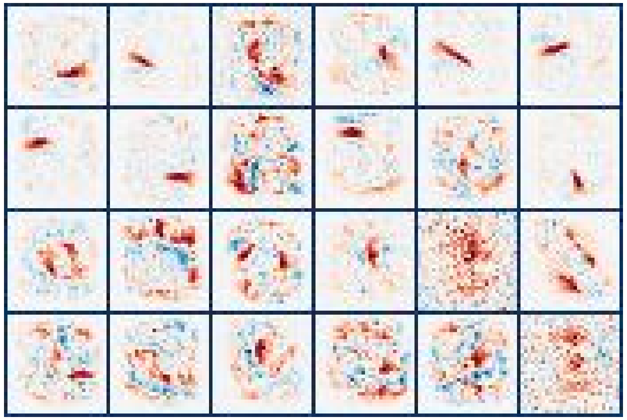
Solution: Drop 50% of hidden units for each training example.
Scale up weights by 2.0 to compensate.

MNIST digit recognition: 

First-layer features after training:



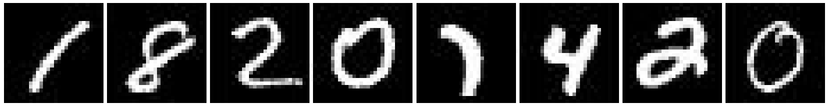
No dropout:
noisy, possibly overfit to training set

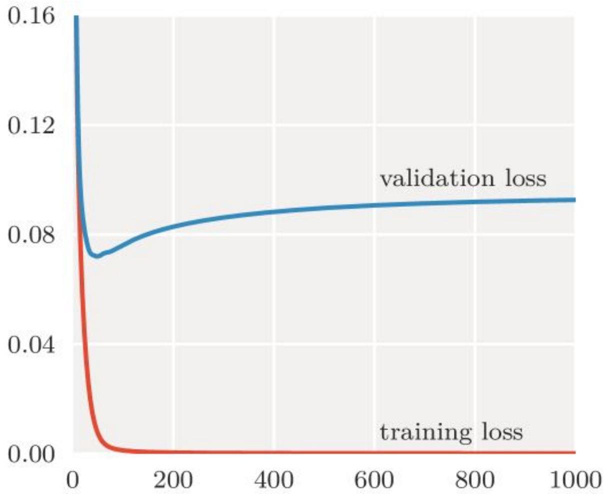


20% input, 50% hidden dropout:
cleaner global features, more general

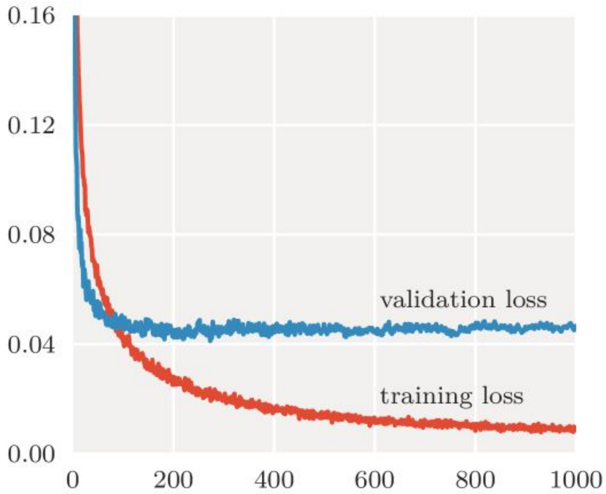
Dropout

Solution: Drop 50% of hidden units for each training example.
Scale up weights by 2.0 to compensate.

MNIST digit recognition: 



No dropout:
quick overfitting, 169 test errors



20% input, 50% hidden dropout:
validation error plateaus, 99 test errors

Deep learning in practice

Initialization — SGD+ — Batch normalization → **Optimization**

Weight decay — Early stopping — Data augmentation — Dropout → **Generalization**

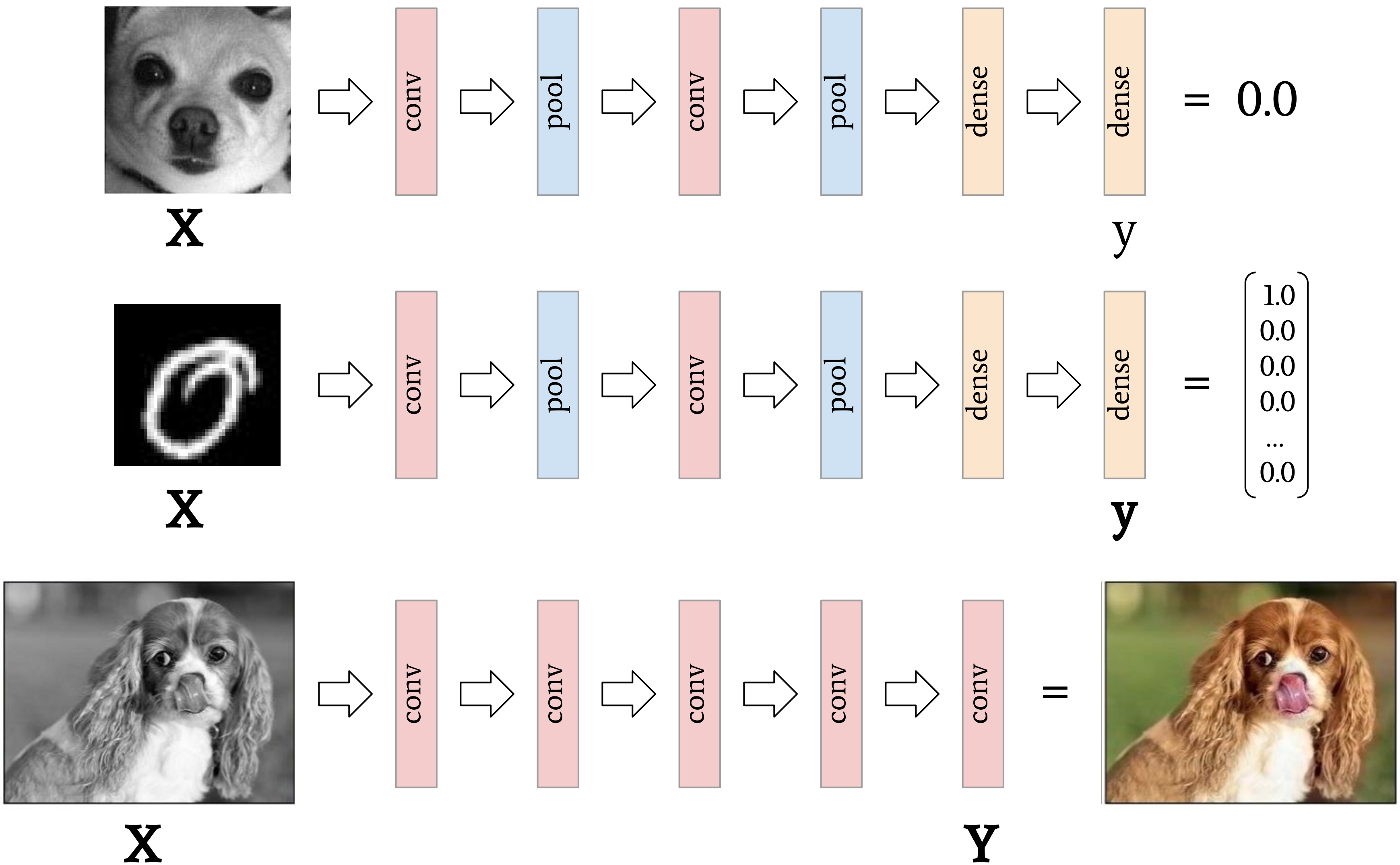
Deep learning in practice

Initialization — SGD+ — Batch normalization → **Optimization**

Weight decay — Early stopping — Data augmentation — Dropout → **Generalization**

Architectures

Traditional Convolutional Neural Network



Going Deeper

How many layers to use?

- Single hidden layer enough for universal approximation
- More hidden layers can express functions more compactly

ImageNet Large Scale Visual Recognition Challenge:

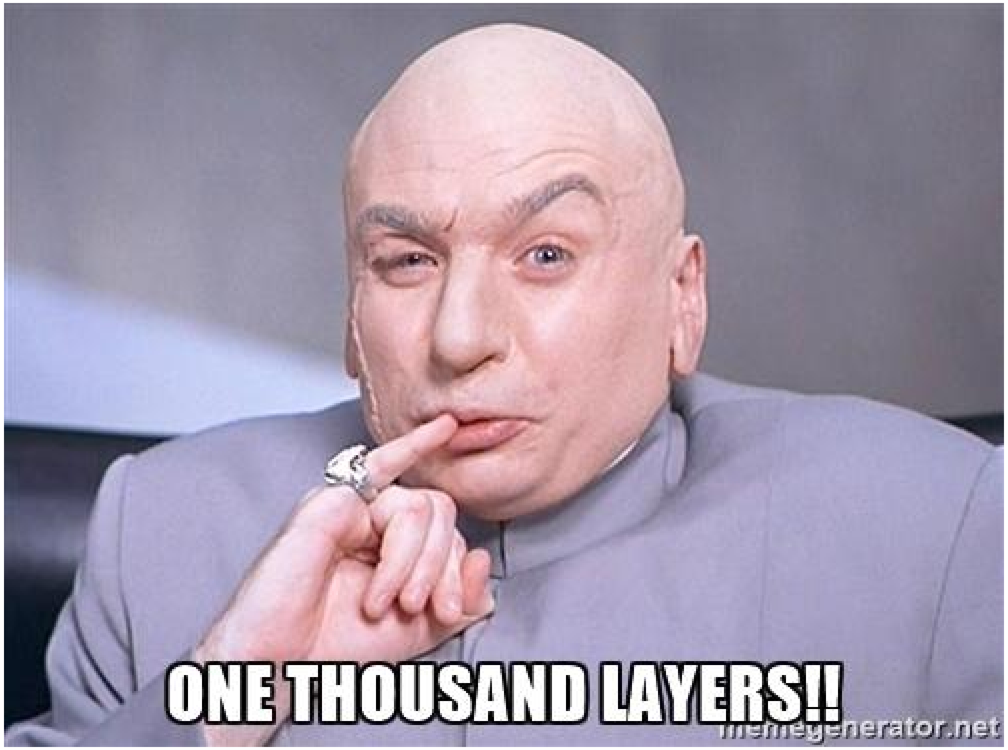
1.2 million training images of 1000 classes (incl. 120 dog breeds)

- 2012: AlexNet, 16.4% top-5 error, 8 layers.
- 2013: ZFNet, 11.2% top-5 error, 8 layers.
- 2014: GoogLeNet: 6.7% top-5 error, 22 layers.
- 2015: ResNets: 3.6% top-5 error, 152 layers.

Going Deeper

How many layers to use?

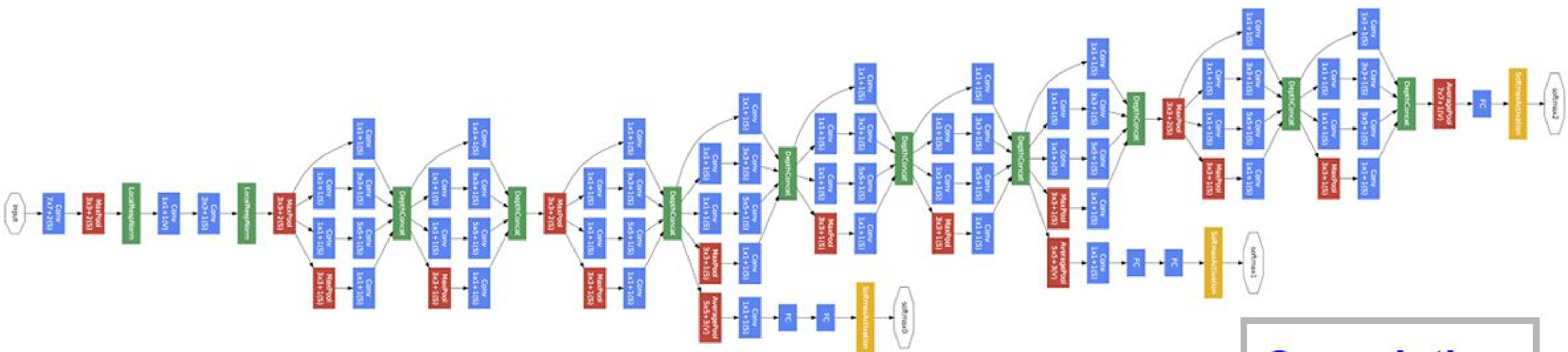
- Single hidden layer enough for universal approximation
- More hidden layers can express functions more compactly



Inception

~~How many layers to use?~~ How to use many layers?

GoogLeNet: 22 layers, **auxiliary classifiers**



Convolution
Pooling
Softmax
Other

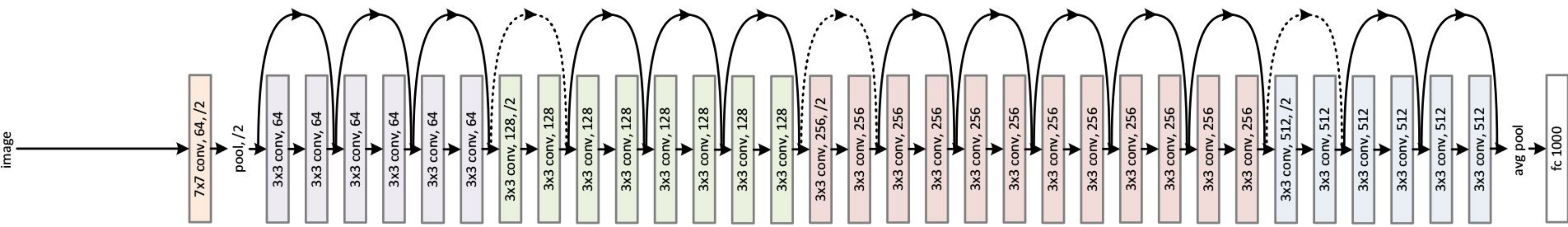
Idea: Provide better gradient information to lower layers via additional classification layers

Sep 2014: Going Deeper with Convolutions, http://arxiv.org/abs/1409.4842

ResNet

~~How many layers to use?~~ How to use many layers?

ResNet: 152 layers (38 shown here), **shortcut connections**



Idea: Provide better gradient information to lower layers via bypasses. Input directly connected to output, learns residuals.

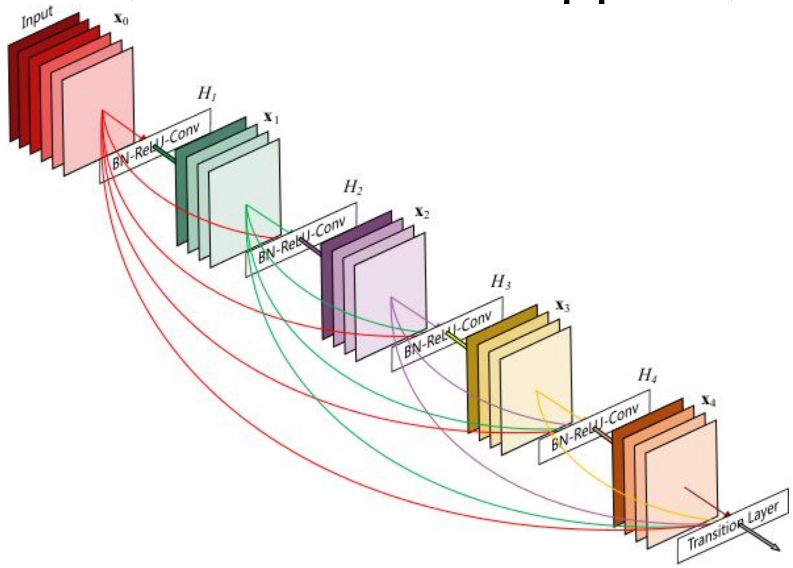
Shown to learn networks of 1001 layers. But: seems to behave like an ensemble of many shallow networks, not a single deep one.

Dec 2015: Deep Residual Learning for Image Recognition, http://arxiv.org/abs/1512.03385
Mar 2016: Identity Mappings in Deep Residual Networks, http://arxiv.org/abs/1603.05027

DenseNet

~~How many layers to use?~~ How to use many layers?

DenseNet: like ResNet, but shortcuts append, not add features



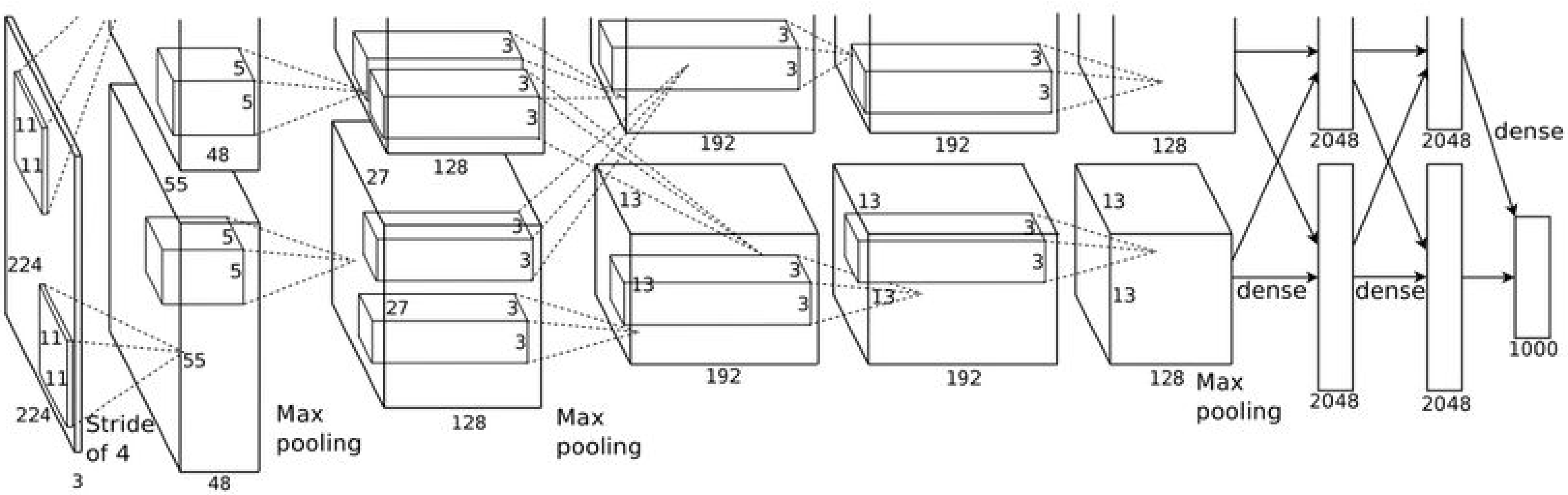
Idea: Each layer expands the set of available feature maps.
Avoids redundant features as learned in ResNet.

Aug 2016, abs/1608.06993: Densely Connected Convolutional Networks

Grouped convolution

Three dimensions: Depth, Width, **Multiplicity**

Can be advantageous to have separate processing chains.



AlexNet: Two chains of identical structure joined in the end.

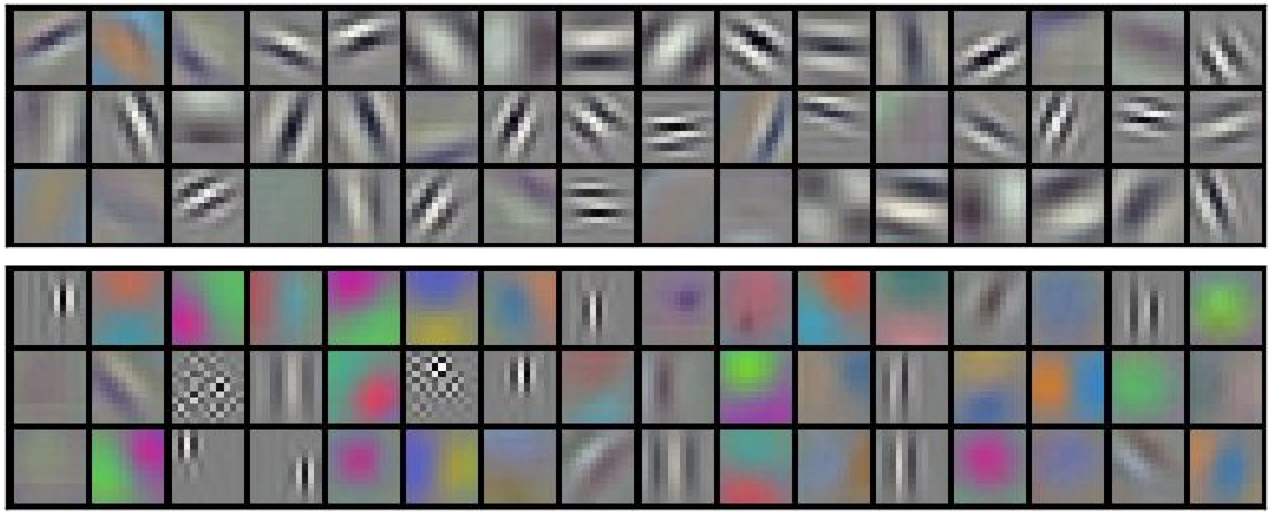
Originally for technical reasons, later shown to improve results.

NIPS 2012: ImageNet Classification with Deep Convolutional Neural Networks

Grouped convolution

Three dimensions: Depth, Width, **Multiplicity**

Can be advantageous to have separate processing chains.



AlexNet: Two chains of identical structure joined in the end.

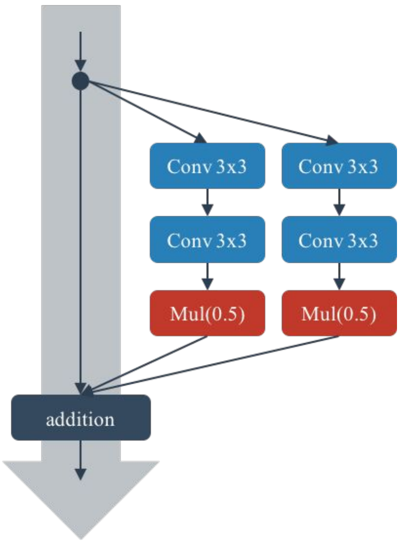
Originally for technical reasons, later shown to improve results.

NIPS 2012: ImageNet Classification with Deep Convolutional Neural Networks

Shake-shake

Three dimensions: Depth, Width, **Multiplicity**

Can be advantageous to have separate processing chains.



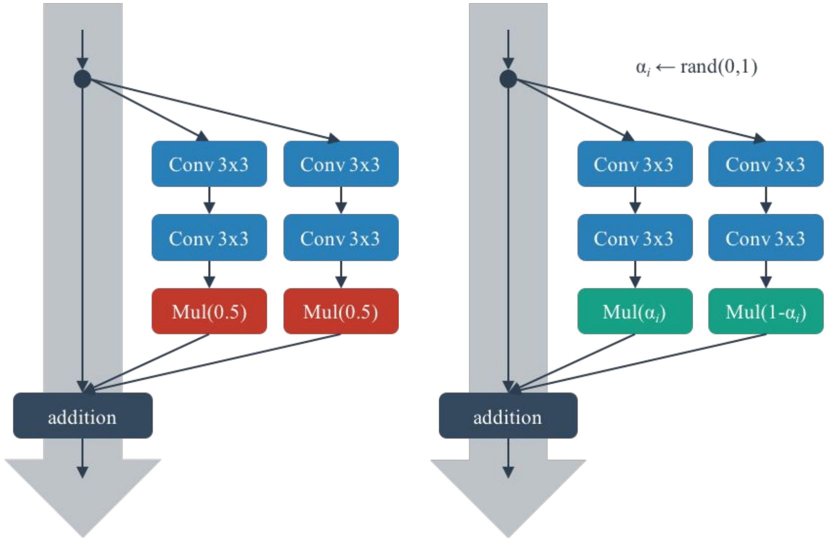
Shake-Shake: Two parallel processing steps averaged.

May 2017, abs/1705.07485: Shake-Shake regularization

Shake-shake

Three dimensions: Depth, Width, **Multiplicity**

Can be advantageous to have separate processing chains.



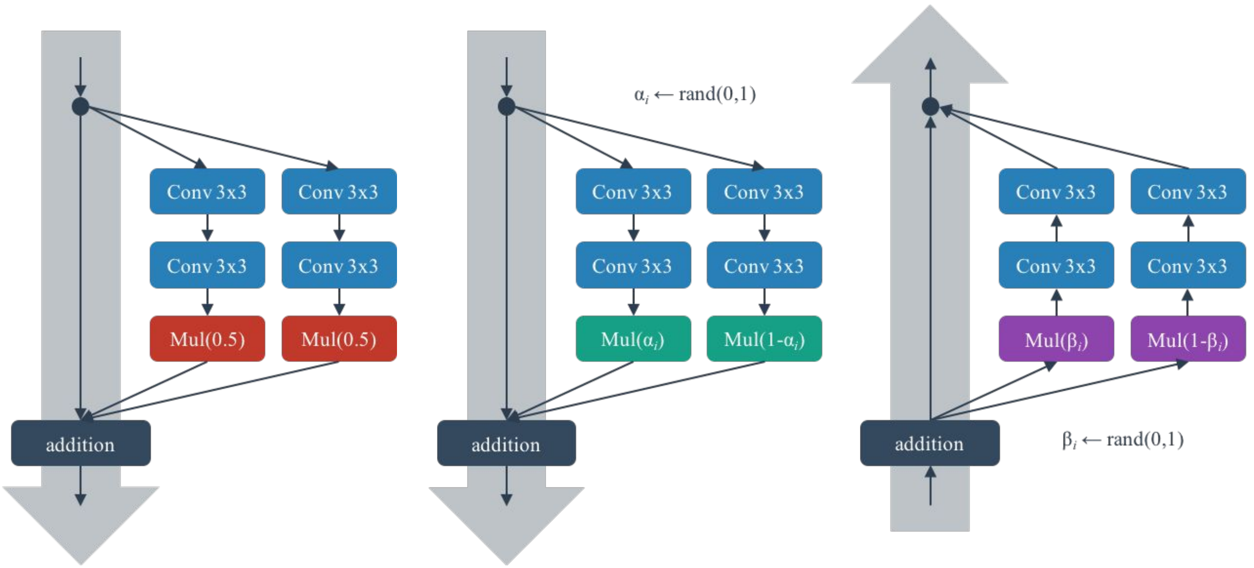
Shake-Shake: Two parallel processing steps ~~averaged.~~ randomly combined.

May 2017, abs/1705.07485: Shake-Shake regularization

Shake-shake

Three dimensions: Depth, Width, **Multiplicity**

Can be advantageous to have separate processing chains.



Shake-Shake: Two parallel processing steps ~~averaged.~~ randomly combined, with different coefficients in forward/backward pass.

May 2017, abs/1705.07485: Shake-Shake regularization

Deep learning in practice

Initialization
SGD+
Batch normalization

Optimization

Weight decay
Early stopping
Data augmentation
Dropout

Generalization

Shake-shake

Architectures

Inception
ResNet
DenseNet
Grouped convolution

Deep learning in practice

Initialization
SGD+
Batch normalization

Optimization

Weight decay
Early stopping
Data augmentation
Dropout
Shake-shake

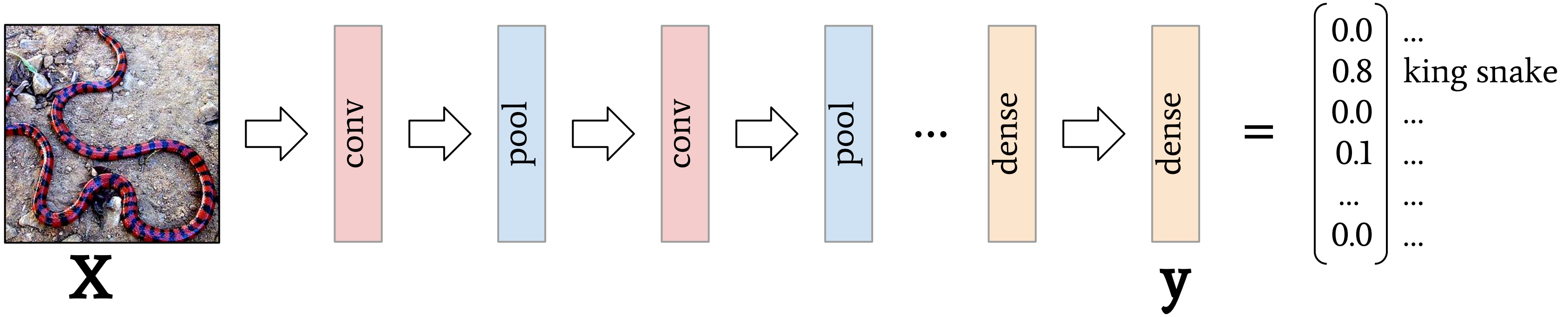
Generalization

Architectures

Inception
ResNet
DenseNet
Grouped convolution

Inspection

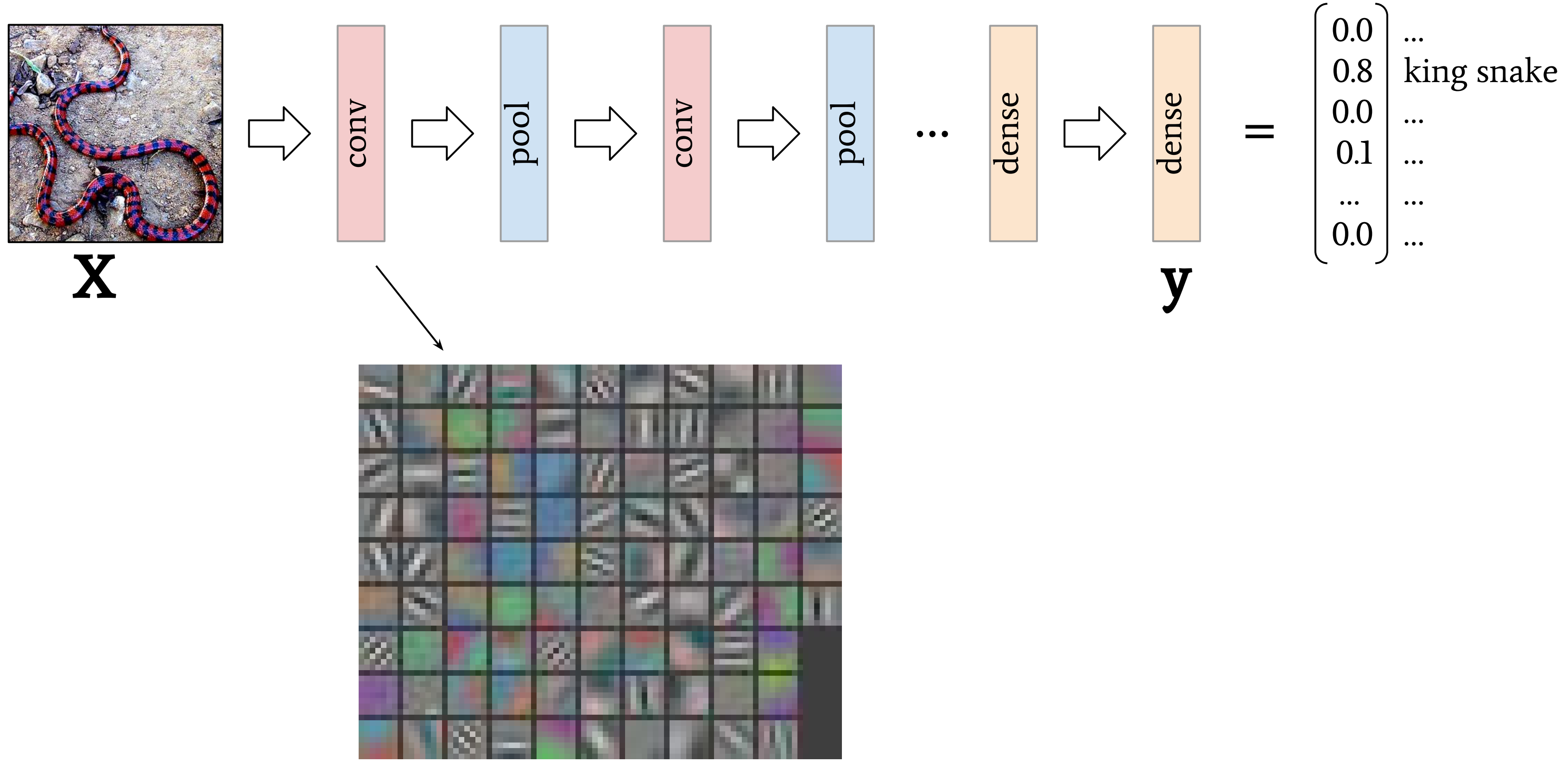
Inspection



$$= \begin{pmatrix} 0.0 \\ 0.8 \\ 0.0 \\ 0.1 \\ \ldots \\ 0.0 \end{pmatrix} \begin{matrix} \ldots \\ \text{king snake} \\ \ldots \\ \ldots \\ \ldots \\ \ldots \end{matrix}$$

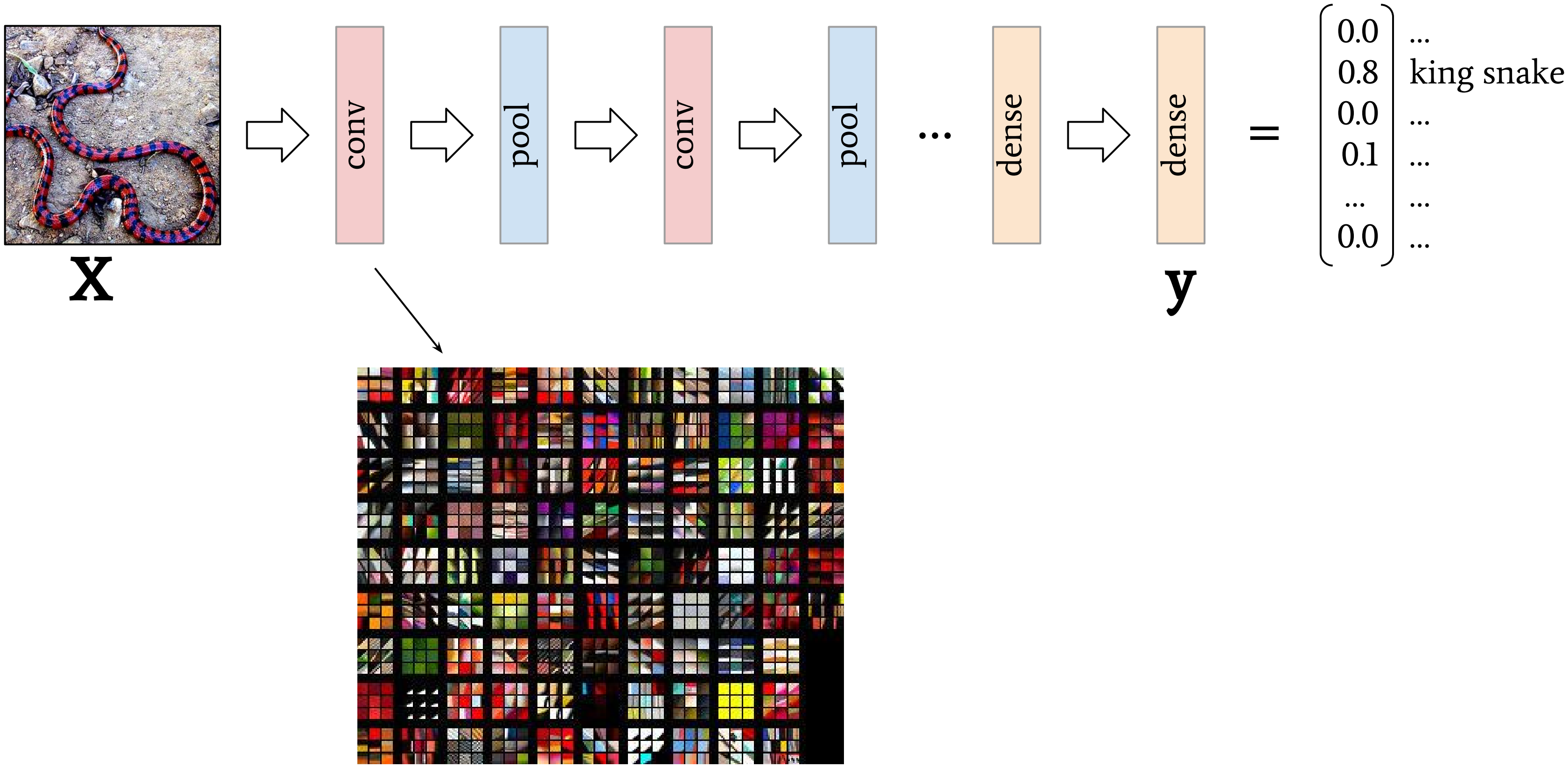
Visualize filters

Method: Show convolution kernels in pixel space. Only possible for first layer.



Visualize data

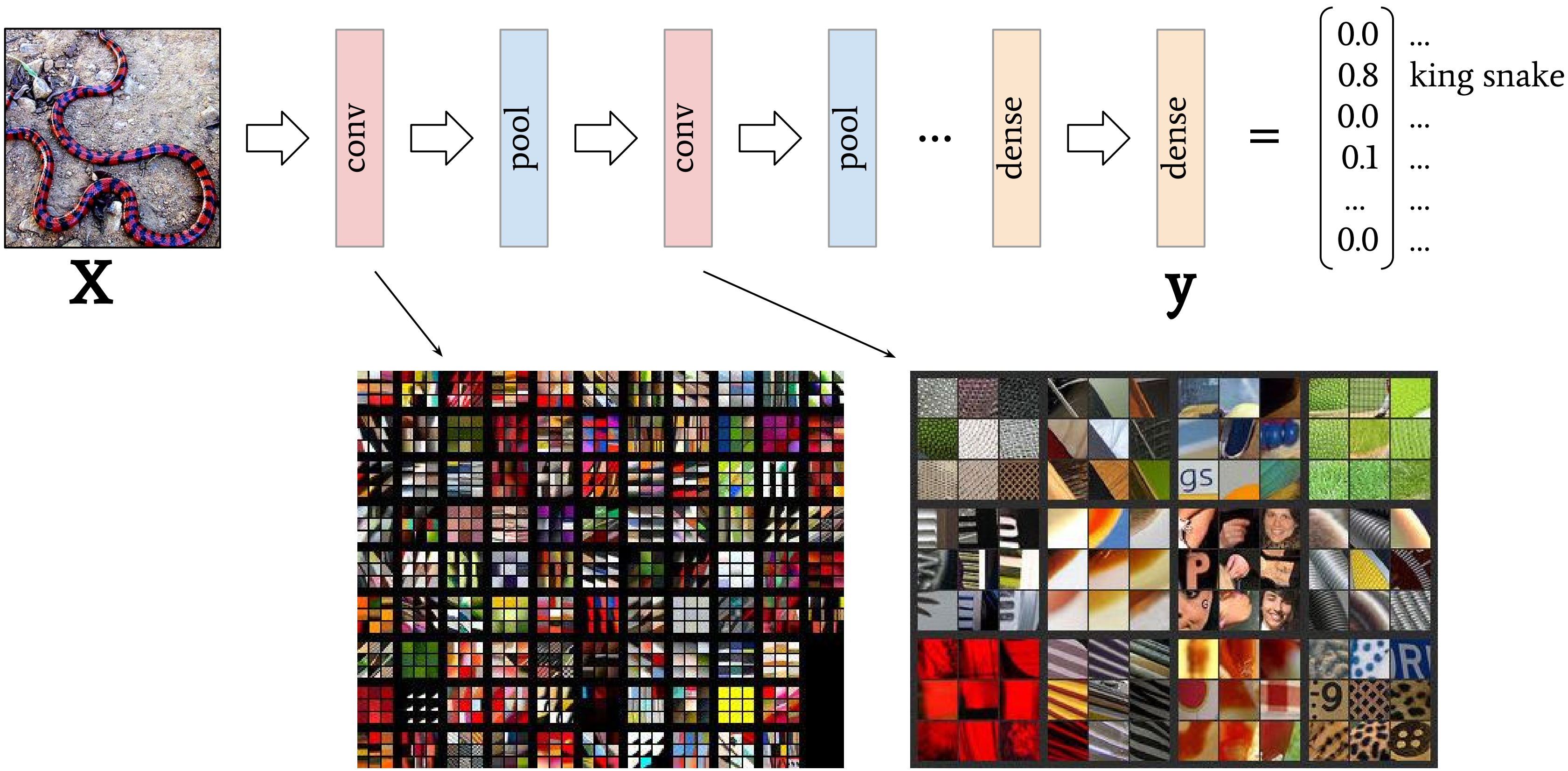
Method: Show training patches that maximally activate some unit.



Nov 2013, abs/1311.2901: Visualizing and Understanding Convolutional Networks

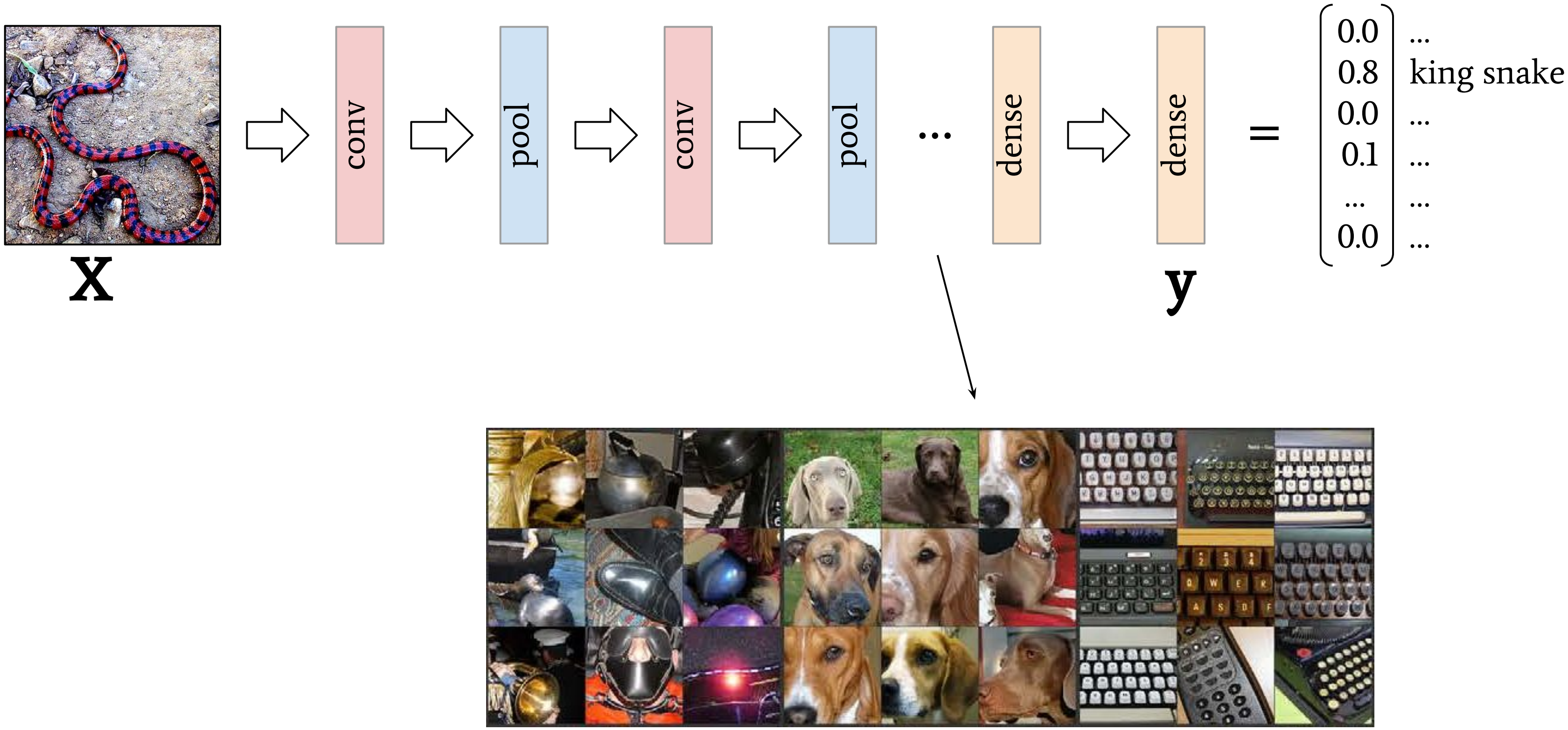
Visualize data

Method: Show training patches that maximally activate some unit.



Nov 2013, abs/1311.2901: Visualizing and Understanding Convolutional Networks

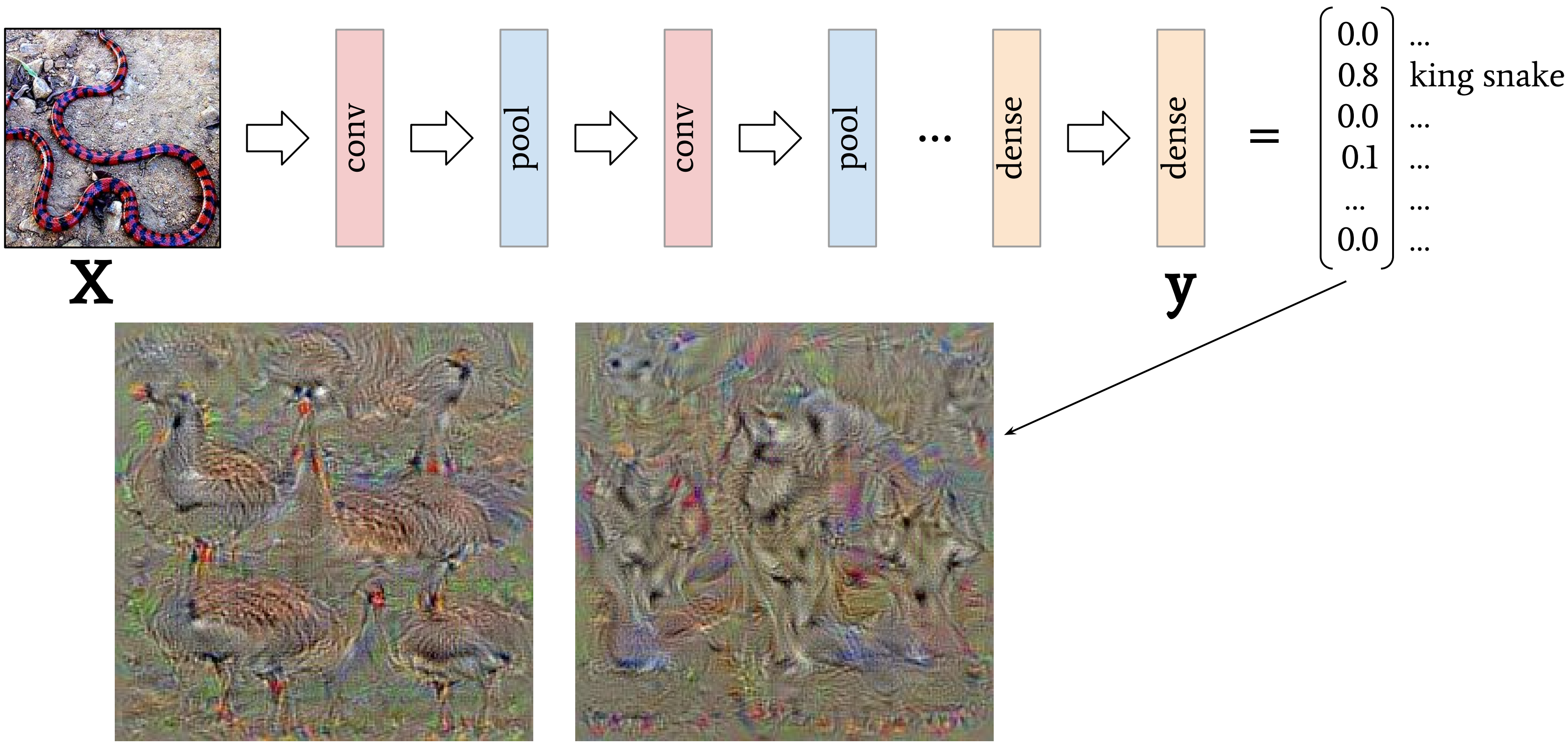
Method: Show training patches that maximally activate some unit.



Nov 2013, abs/1311.2901: Visualizing and Understanding Convolutional Networks

Generate data

Method: Generate patches that maximally activate some unit.

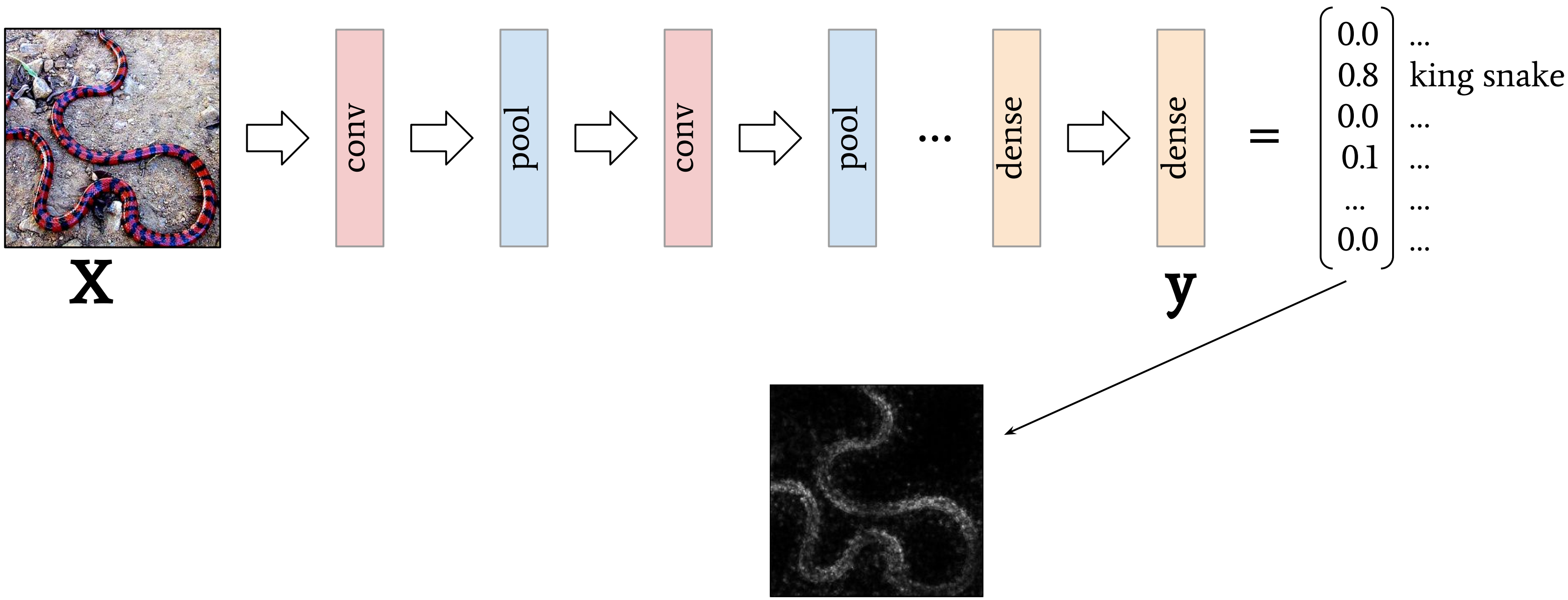


goose husky

Dec 2013, abs/1312.6034: Deep Inside Conv. Networks: Visualising Image Classification Models and Saliency Maps

Guided backpropagation

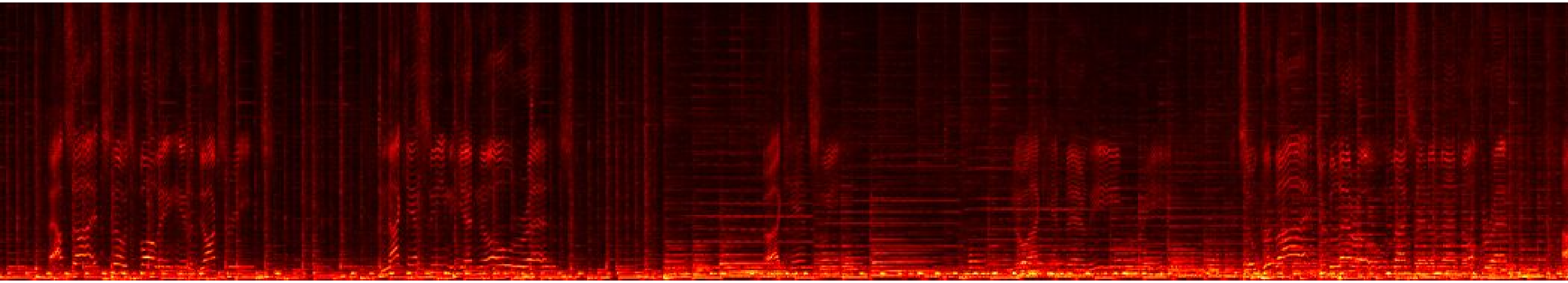
Method: Show gradient of some unit wrt. input example (modified backward pass).

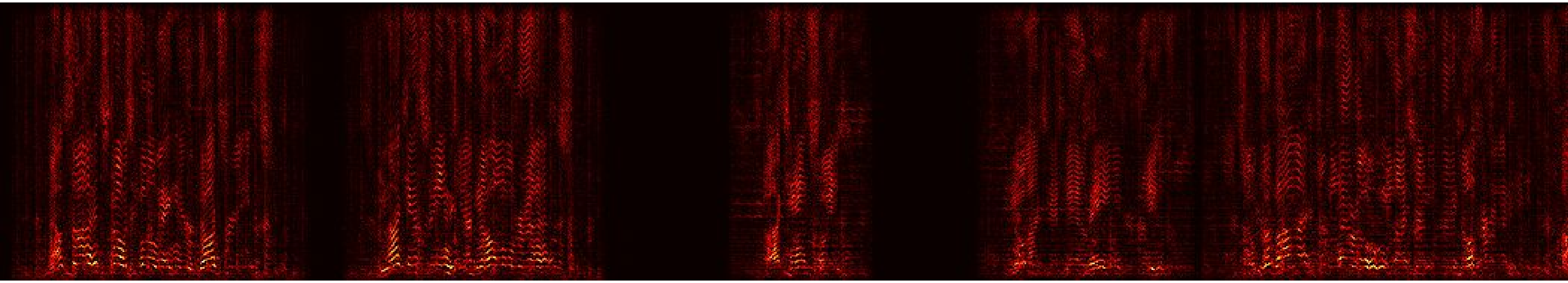


Dec 2014, abs/1412.6806: Striving for Simplicity: The All Convolutional Net

Guided backpropagation

Method: Show gradient of some unit wrt. input example (modified backward pass).





ISMIR 2016: Learning to Pinpoint Singing Voice from Weakly Labeled Examples

Deep learning in practice

Optimization: Initialization, SGD+, Batch normalization

Generalization: Weight decay, Early stopping, Data augmentation, Dropout, Shake-shake

Architectures: Inception, ResNet, DenseNet, Grouped convolution

Inspection: Show filters, Show data, Generate data, Guided backpropagation

Deep learning in practice