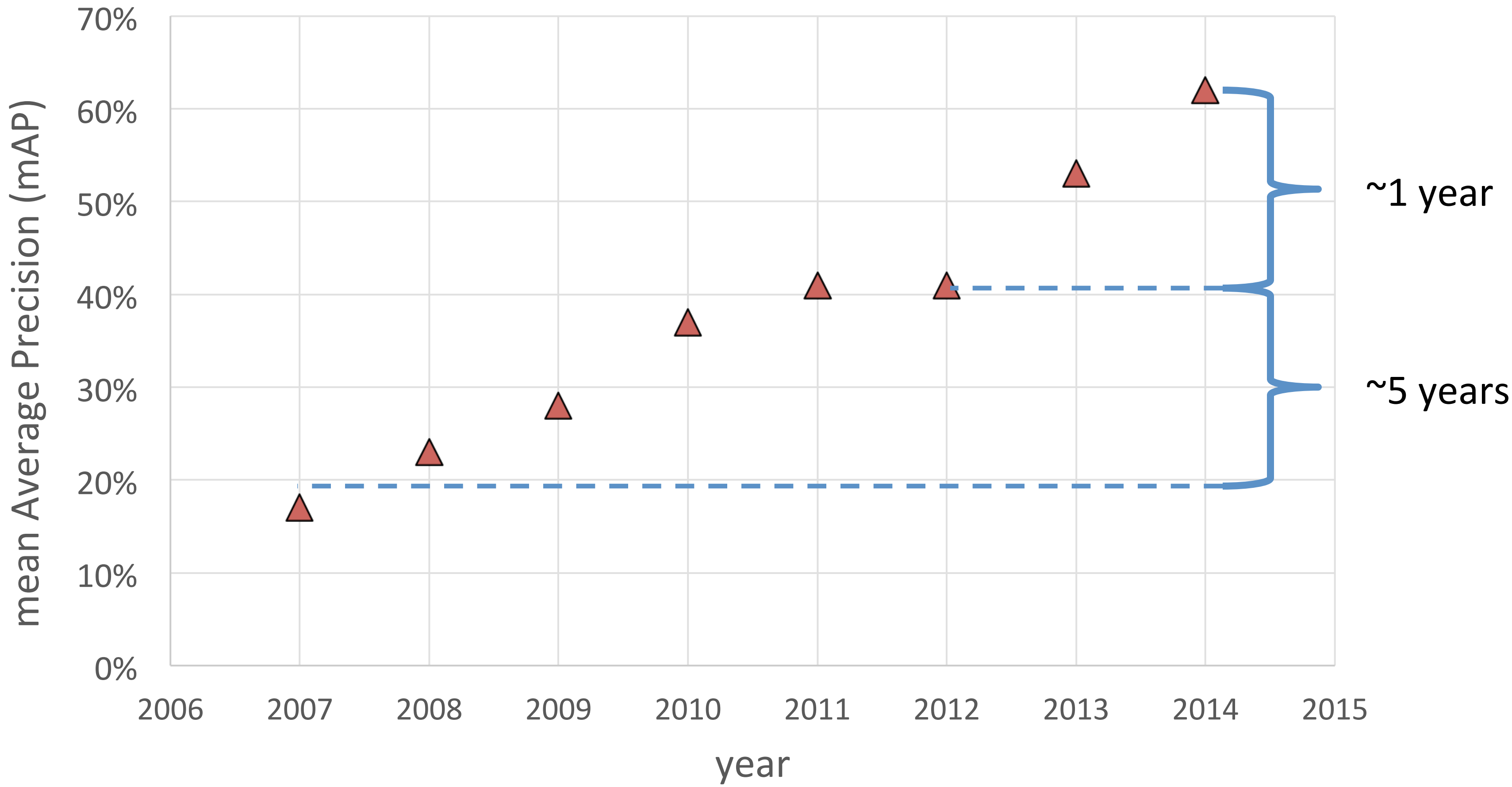
Deep Learning

Ali Farhadi
Mohammad Rastegari
CSE 576

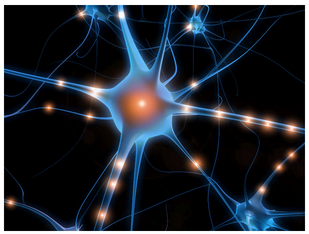
Region-based Convolutional Networks (R-CNNs)

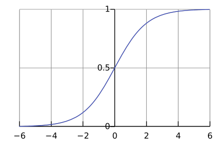


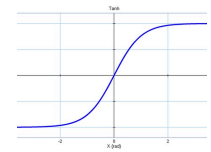
[R-CNN. Girshick et al. CVPR 2014]

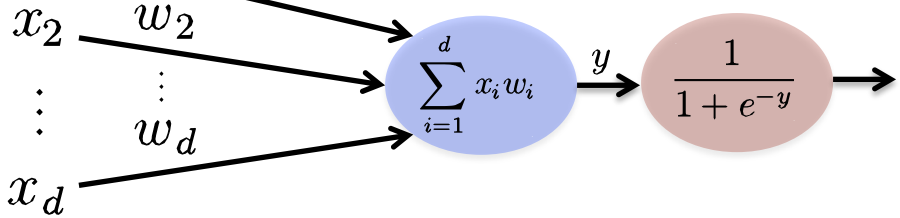
Neural Networks

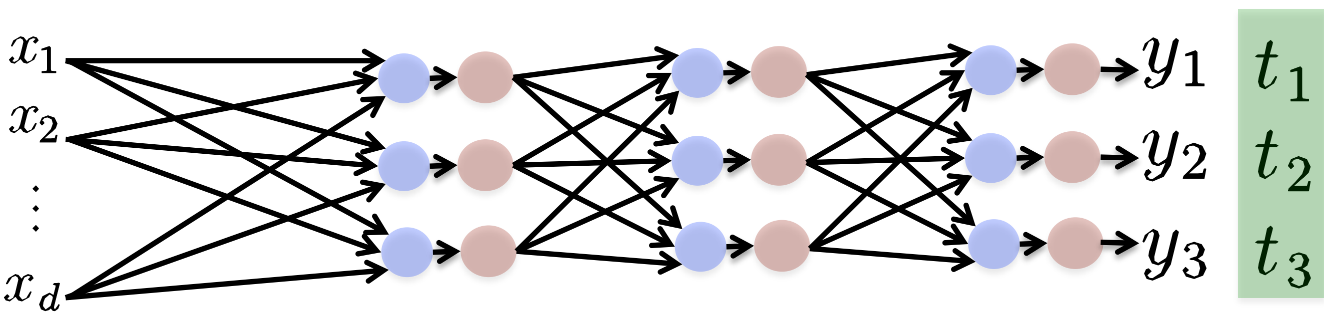
Tanh









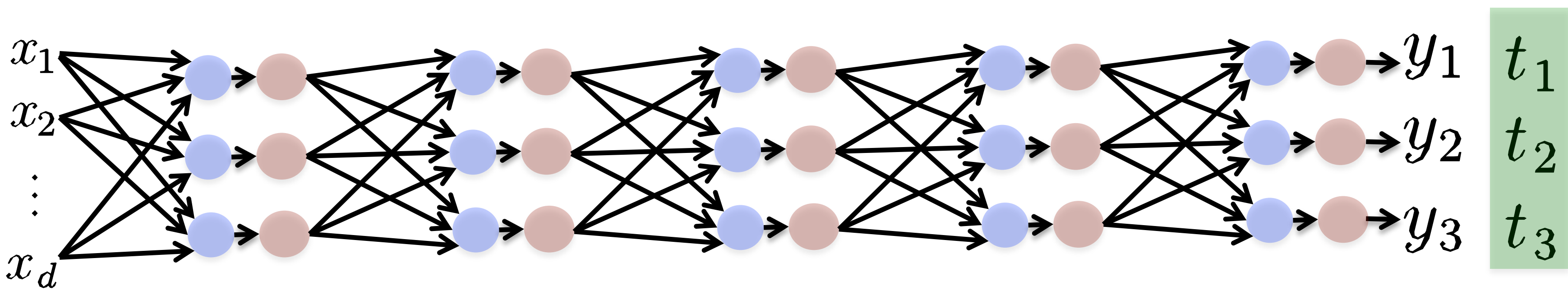


$$L(\mathbf{y}, \mathbf{t}) = \|\mathbf{y} - \mathbf{t}\|^2$$

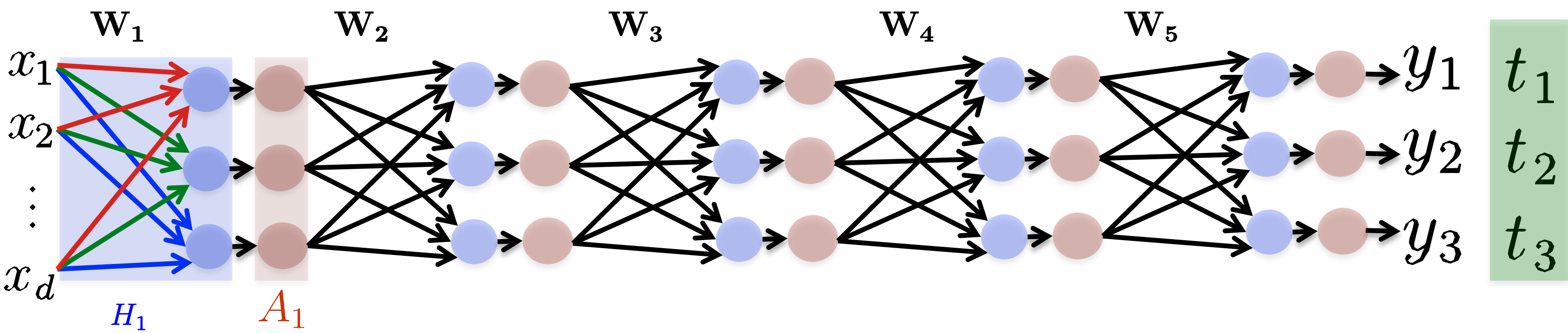
$$\mathbf{x} = [x_1, x_2, \ldots, x_d]$$
$$\mathbf{t} = [t_1, t_2, t_3]$$
$$\mathbf{y} = [y_1, y_2, y_3]$$

$$\mathcal{T} = \{(\mathbf{x_1}, \mathbf{t_1}), (\mathbf{x_2}, \mathbf{t_2}), \ldots, (\mathbf{x_n}, \mathbf{t_n})\}$$

Multi Layer Neural Networks



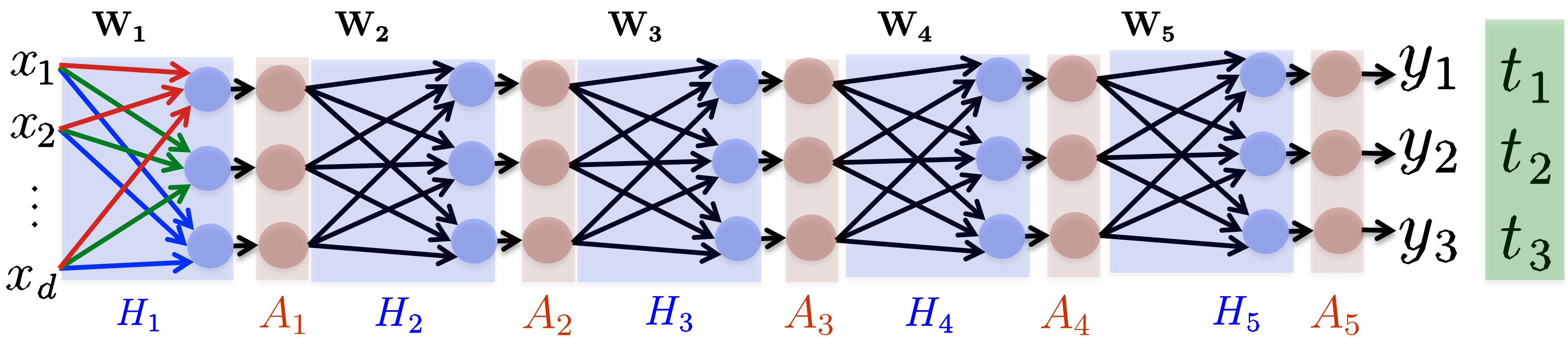
Multi Layer Neural Networks



$$\mathbf{W_1} = \begin{bmatrix} w_{11}, w_{12}, \ldots, w_{1d} \\ w_{21}, w_{22}, \ldots, w_{2d} \\ w_{31}, w_{32}, \ldots, w_{3d} \end{bmatrix}$$

$$H_l(\mathbf{X}) = \mathbf{W}_l\mathbf{X}$$
$$A_l(x_i) = \sigma(x_i)$$

Multi Layer Neural Networks



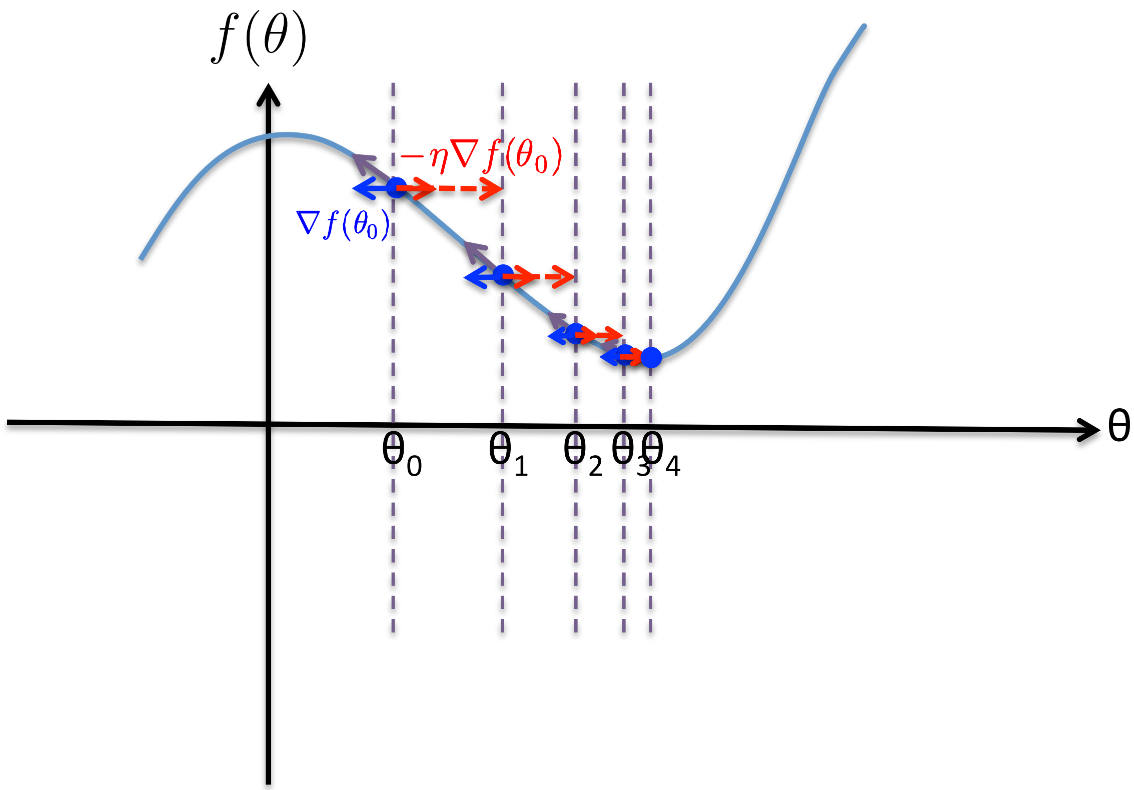
$$\mathbf{y} = A_5(H_5(A_4(H_4(A_3(H_3(A_2(H_2(A_1(H_1(\mathbf{x}))))))))))$$

$$L(\mathbf{y}, \mathbf{t}) = L(A_5, \mathbf{t})$$

$$H_l(\mathbf{X}) = \mathbf{W}_l \mathbf{X}$$

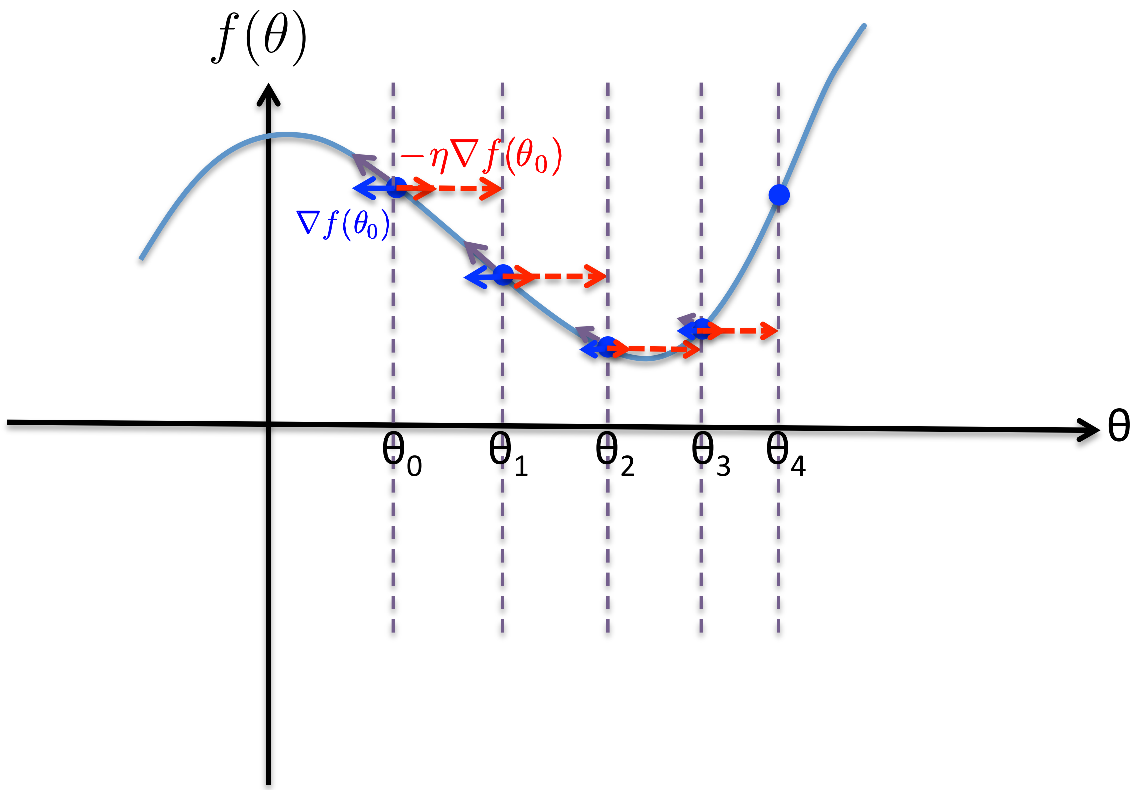
$$A_l(x_i) = \sigma(x_i)$$

Gradient Descend

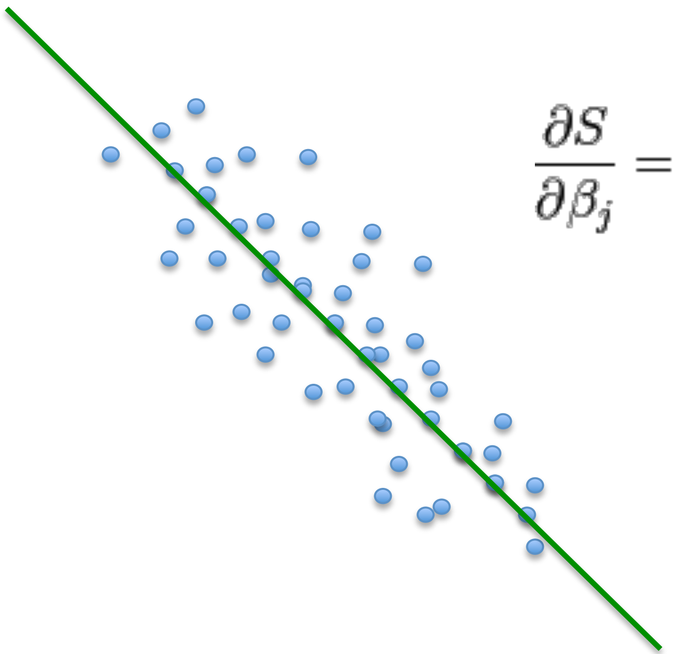


$$\theta_t = \theta_{t-1} - \eta_t \nabla f(\theta_{t-1})$$

Gradient Descend

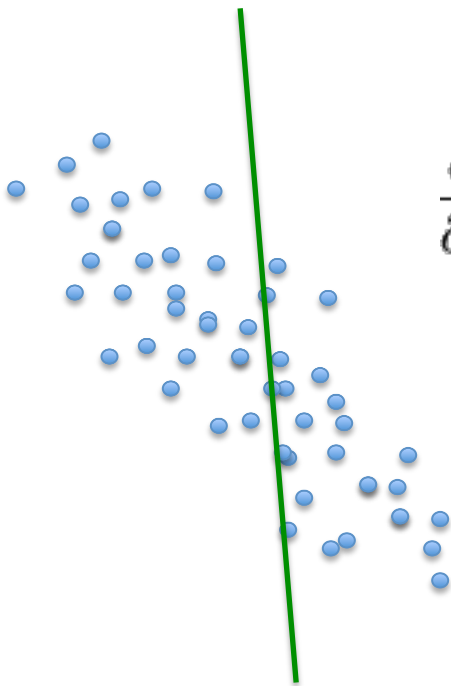


$$\theta_t = \theta_{t-1} - \eta_t \nabla f(\theta_{t-1})$$



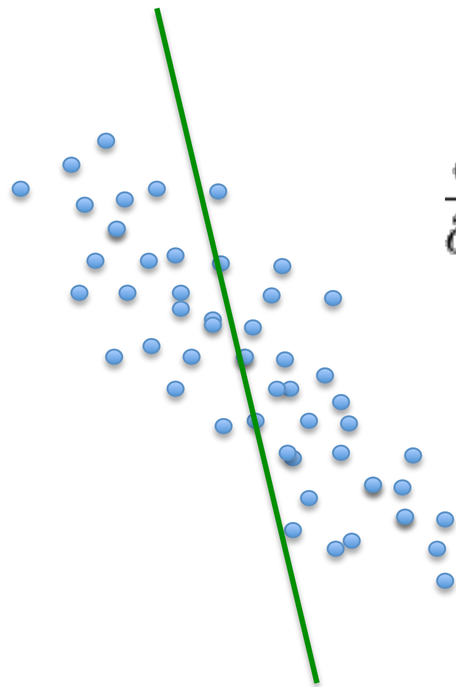
$$S = \min_{\beta} \|\beta X - Y\|$$

$$\frac{\partial S}{\partial \beta_j} = 2\sum_{i=1}^{m}\left(y_i - \sum_{k=1}^{n} X_{ik}\beta_k\right)(-X_{ij})\ (j = 1, 2, \ldots, n).$$



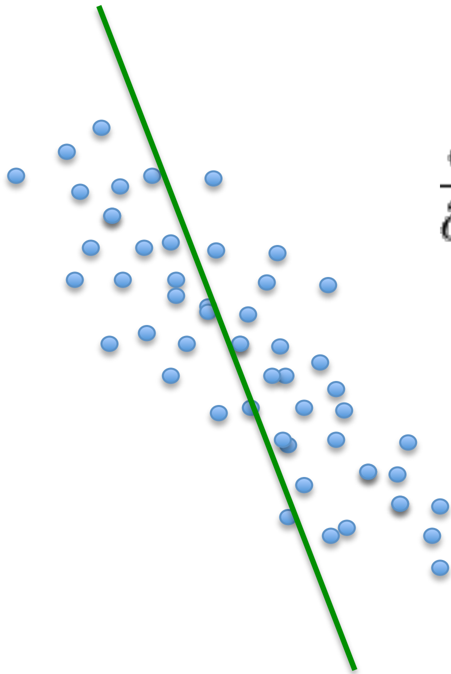
$$S = \min_{\beta} \|\beta X - Y\|$$

$$\frac{\partial S}{\partial \beta_j} = 2\sum_{i=1}^{m}\left(y_i - \sum_{k=1}^{n} X_{ik}\beta_k\right)(-X_{ij})\ (j = 1, 2, \ldots, n).$$



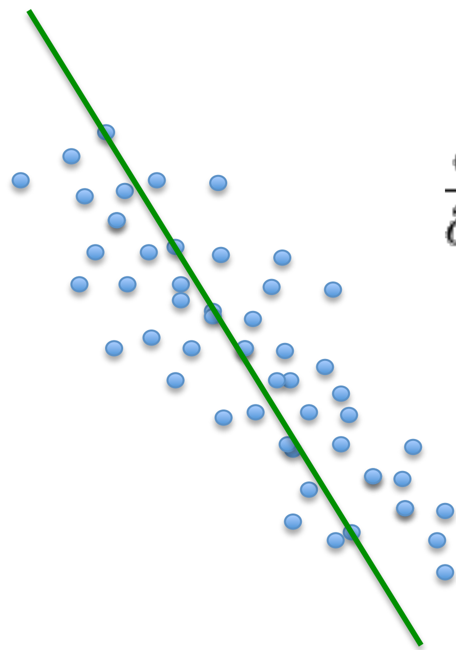
$$S = \min_{\beta} \|\beta X - Y\|$$

$$\frac{\partial S}{\partial \beta_j} = 2\sum_{i=1}^{m}\left(y_i - \sum_{k=1}^{n} X_{ik}\beta_k\right)(-X_{ij}) \ (j = 1, 2, \ldots, n).$$



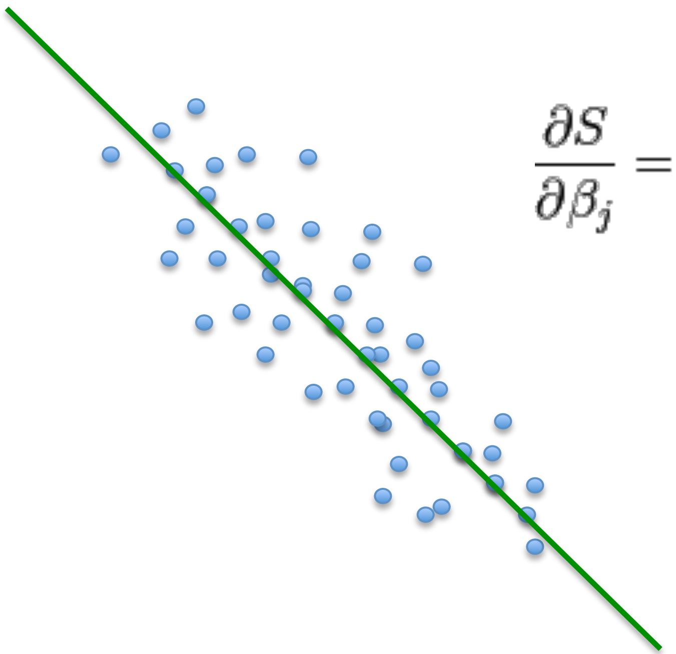
$$S = \min_{\beta} \|\beta X - Y\|$$

$$\frac{\partial S}{\partial \beta_j} = 2\sum_{i=1}^{m}\left(y_i - \sum_{k=1}^{n} X_{ik}\beta_k\right)(-X_{ij}) \ (j = 1, 2, \ldots, n).$$



$$S = \min_{\beta} \|\beta X - Y\|$$

$$\frac{\partial S}{\partial \beta_j} = 2\sum_{i=1}^{m}\left(y_i - \sum_{k=1}^{n} X_{ik}\beta_k\right)(-X_{ij}) \ (j = 1, 2, \ldots, n).$$

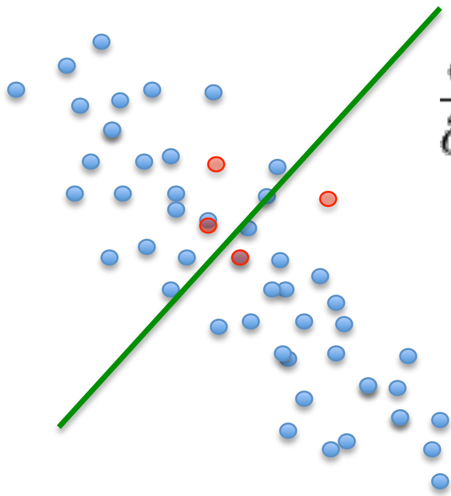


$$S = \min_{\beta} \|\beta X - Y\|$$

$$\frac{\partial S}{\partial \beta_j} = 2\sum_{i=1}^{m}\left(y_i - \sum_{k=1}^{n} X_{ik}\beta_k\right)(-X_{ij})\ (j = 1, 2, \ldots, n).$$

Stochastic Gradient Descend

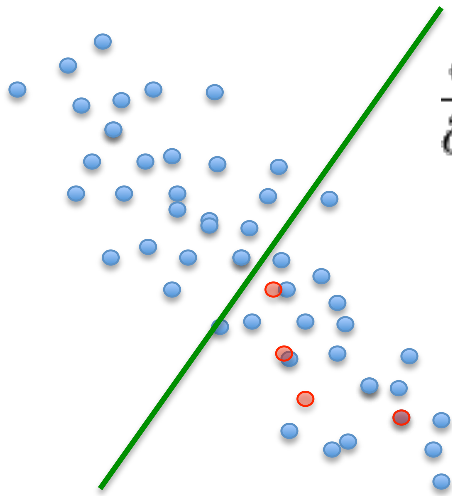
$$S = \min_{\beta} \|\beta X - Y\|$$



$$\frac{\partial S}{\partial \beta_j} = 2\sum_{i=1}^{m}\left(y_i - \sum_{k=1}^{n} X_{ik}\beta_k\right)(-X_{ij}) \; (j = 1, 2, \ldots, n).$$

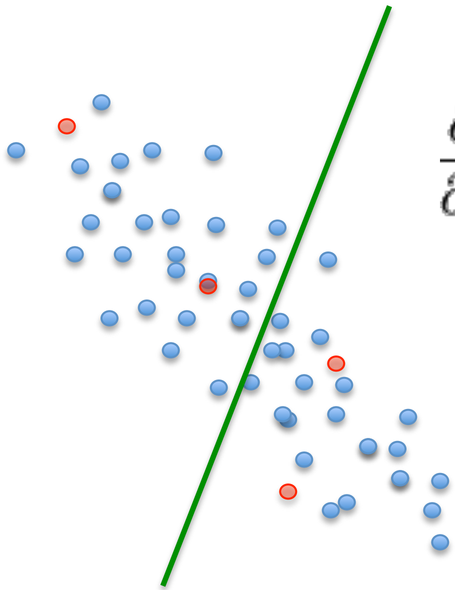
Stochastic Gradient Descend

$$S = \min_{\beta} \|\beta X - Y\|$$



$$\frac{\partial S}{\partial \beta_j} = 2\sum_{i=1}^{m}\left(y_i - \sum_{k=1}^{n} X_{ik}\beta_k\right)(-X_{ij}) \ (j = 1, 2, \dots, n).$$

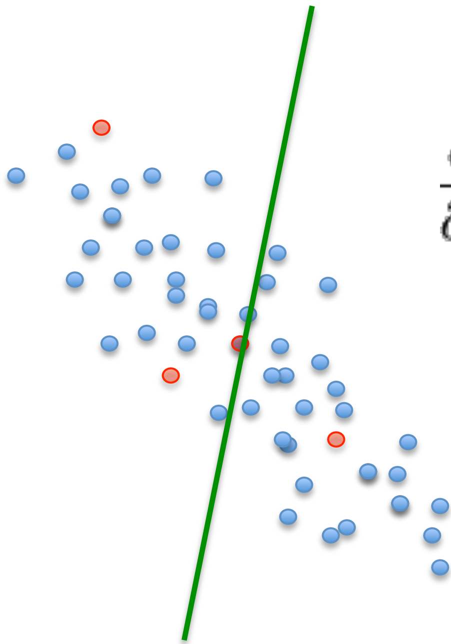
Stochastic Gradient Descend



$$S = \min_{\beta} \|\beta X - Y\|$$

$$\frac{\partial S}{\partial \beta_j} = 2\sum_{i=1}^{m}\left(y_i - \sum_{k=1}^{n} X_{ik}\beta_k\right)(-X_{ij})\ (j = 1, 2, \ldots, n).$$

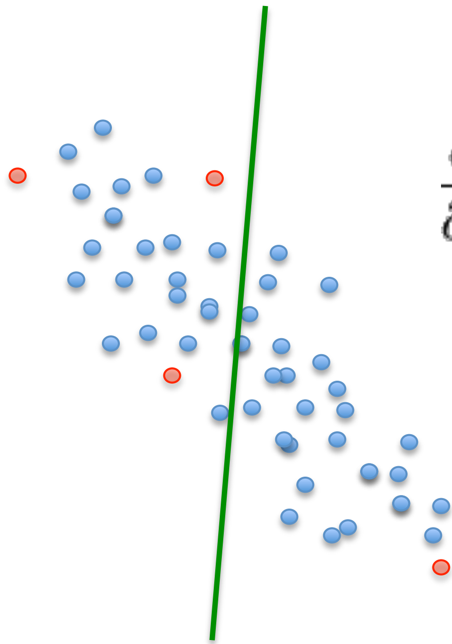
Stochastic Gradient Descend



$$S = \min_{\beta} \|\beta X - Y\|$$

$$\frac{\partial S}{\partial \beta_j} = 2 \sum_{i=1}^{m} \left(y_i - \sum_{k=1}^{n} X_{ik}\beta_k \right) (-X_{ij}) \; (j = 1, 2, \ldots, n).$$

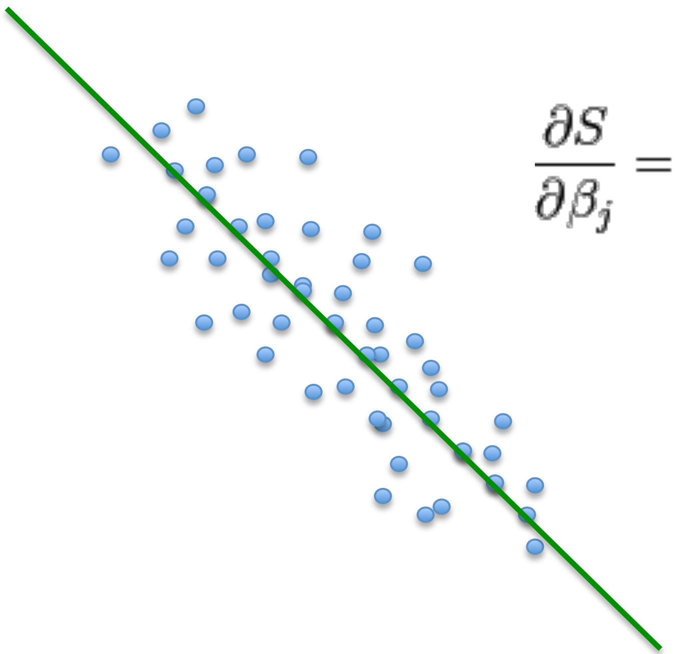
Stochastic Gradient Descend



$$S = \min_{\beta} \|\beta X - Y\|$$

$$\frac{\partial S}{\partial \beta_j} = 2\sum_{i=1}^{m}\left(y_i - \sum_{k=1}^{n} X_{ik}\beta_k\right)(-X_{ij}) \; (j = 1, 2, \ldots, n).$$

Stochastic Gradient Descend

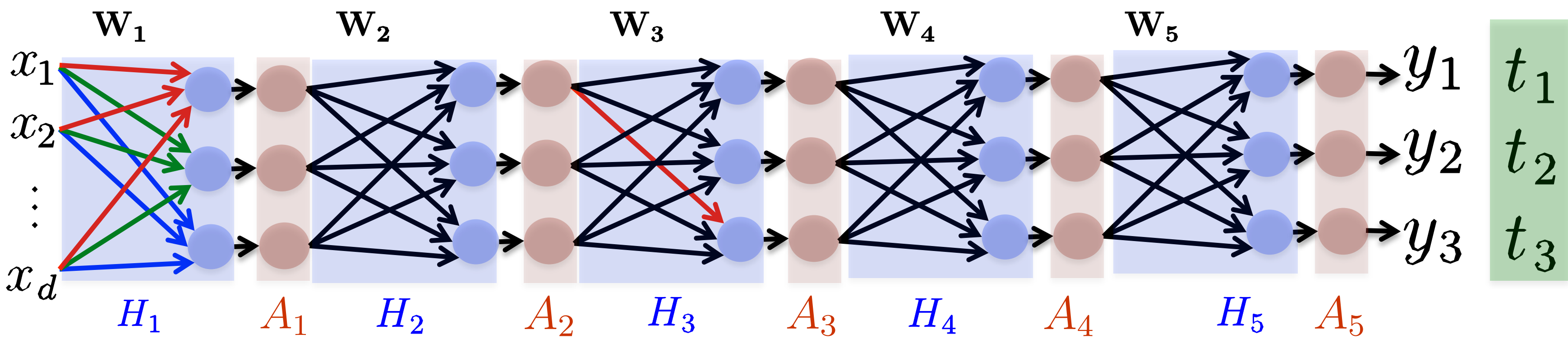


$$S = \min_{\beta} \|\beta X - Y\|$$

$$\frac{\partial S}{\partial \beta_j} = 2\sum_{i=1}^{m}\left(y_i - \sum_{k=1}^{n} X_{ik}\beta_k\right)(-X_{ij}) \ (j = 1, 2, \ldots, n).$$

A lot more iteration

Multi Layer Neural Networks



$$\mathbf{y} = A_5(H_5(A_4(H_4(A_3(H_3(A_2(H_2(A_1(H_1(\mathbf{x}))))))))))$$

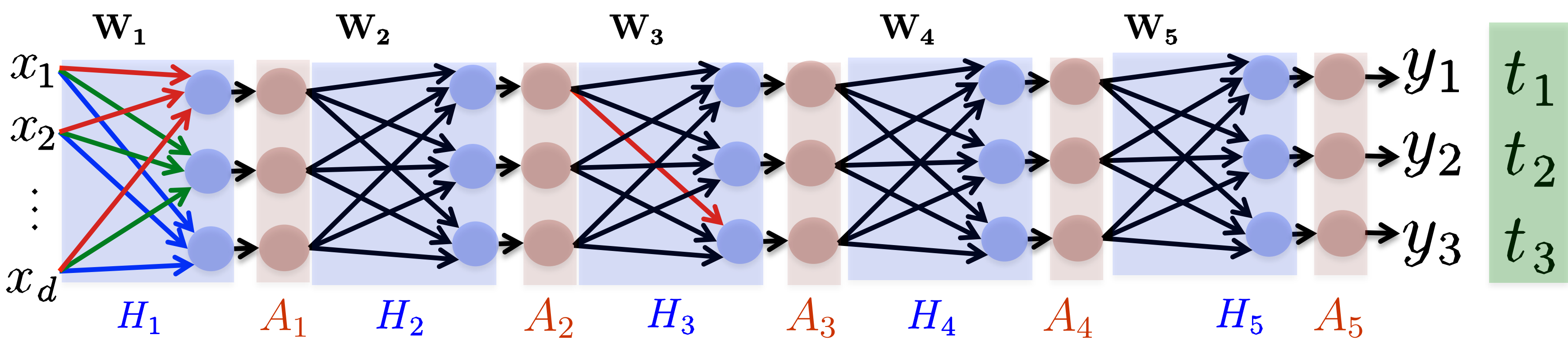
$$L(\mathbf{y}, \mathbf{t}) = L(A_5, \mathbf{t})$$

$$W_{3(ij)}^{t} = W_{3(ij)}^{t-1} - \eta \frac{\partial L}{\partial W_{3(ij)}}$$

$$\frac{\partial L}{\partial W_{3(ij)}} = \frac{\partial L}{\partial A_5} \cdot \frac{\partial A_5}{\partial H_5} \cdot \frac{\partial H_5}{\partial A_4} \cdot \frac{\partial A_4}{\partial H_4} \cdot \frac{\partial H_4}{\partial A_{3(i)}} \cdot \frac{\partial A_{3(i)}}{\partial H_{3(i)}} \cdot \frac{\partial H_{3(i)}}{\partial W_{3(ij)}}$$

$$H_l(\mathbf{X}) = \mathbf{W}_l \mathbf{X}$$
$$A_l(x_i) = \sigma(x_i)$$

Multi Layer Neural Networks



$$\mathbf{y} = A_5(H_5(A_4(H_4(A_3(H_3(A_2(H_2(A_1(H_1(\mathbf{x}))))))))))$$

$$L(\mathbf{y}, \mathbf{t}) = L(A_5, \mathbf{t}) = \frac{1}{2}\left\| A_5 - \mathbf{t} \right\|^2$$

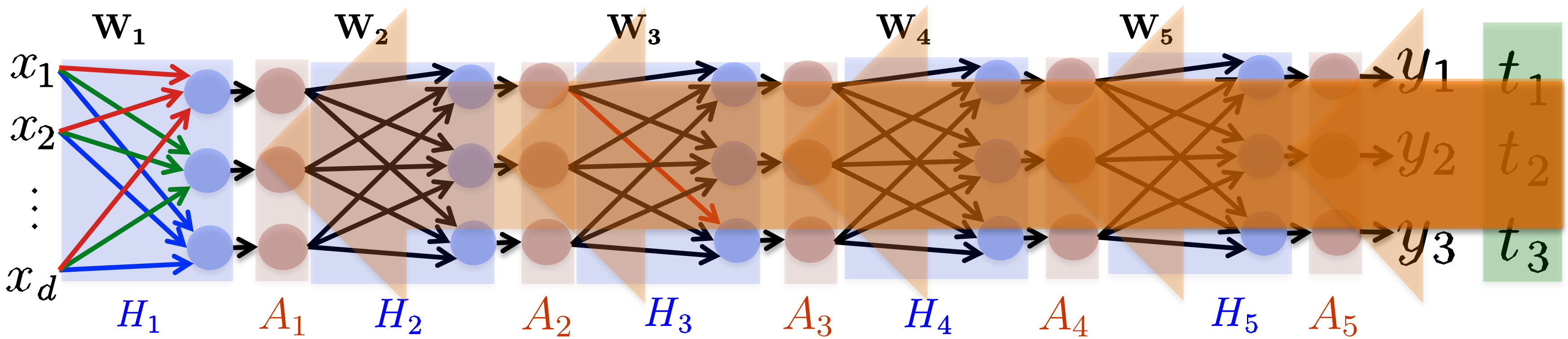
$$\frac{\partial L}{\partial W_{3(ij)}} = \frac{\partial L}{\partial A_5} \cdot \frac{\partial A_5}{\partial H_5} \cdot \frac{\partial H_5}{\partial A_4} \cdot \frac{\partial A_4}{\partial H_4} \cdot \frac{\partial H_4}{\partial A_{3(i)}} \cdot \frac{\partial A_{3(i)}}{\partial H_{3(i)}} \cdot \frac{\partial H_{3(i)}}{\partial W_{3(ij)}}$$

$$(A_5 - \mathbf{t})^\top \odot (A_5(1 - A_5))^\top \cdot W_5 \cdot (A_4(1 - A_4))^\top \cdot W_{3(:i)} \cdot (A_{3(i)}(1 - A_{3(i)}))^\top \cdot A_{2(j)}$$

$$H_l(\mathbf{X}) = \mathbf{W}_l \mathbf{X}$$

$$\sigma' = \sigma(1 - \sigma) \qquad A_l(x_i) = \sigma(x_i)$$

Error Backpropagation



$$E_5 = \frac{\partial L}{\partial A_5} \cdot \frac{\partial A_5}{\partial H_5}$$

$$E_4 = E_5 \cdot \frac{\partial H_5}{\partial A_4} \cdot \frac{\partial A_4}{\partial H_4}$$

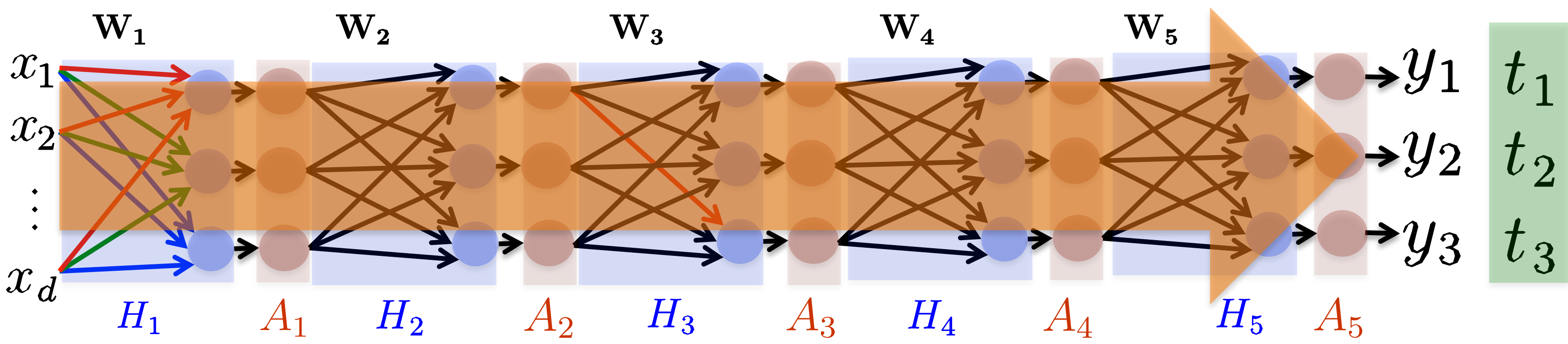
$$E_3 = E_4 \cdot \frac{\partial H_4}{\partial A_3} \cdot \frac{\partial A_3}{\partial H_3}$$

$$E_2 = E_3 \cdot \frac{\partial H_3}{\partial A_2} \cdot \frac{\partial A_2}{\partial H_2}$$

$$E_1 = E_2 \cdot \frac{\partial H_2}{\partial A_1} \cdot \frac{\partial A_1}{\partial H_1}$$

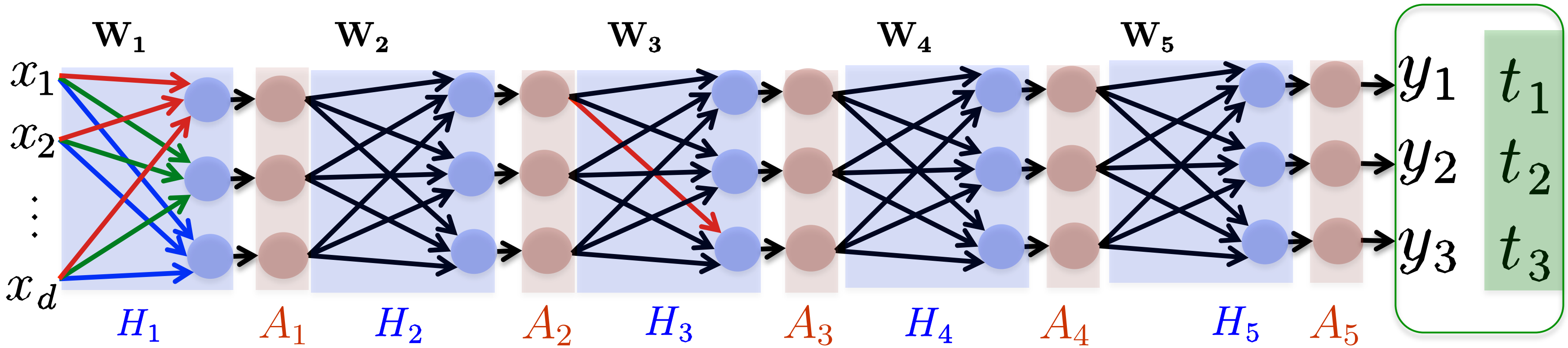
$$\begin{aligned}\frac{\partial L}{\partial W_{l(ij)}} &= E_{l+1} \cdot \frac{\partial H_{l+1}}{\partial A_{l(i)}} \cdot \frac{\partial A_{l(i)}}{\partial H_{l(i)}} \cdot \frac{\partial H_{l(i)}}{\partial W_{l(ij)}} \\ &= E_{l+1} \cdot W_{l(:i)} \cdot (A_{l(i)}(1 - A_{l(i)}))^\top \cdot A_{l-1(j)}\end{aligned}$$

Training



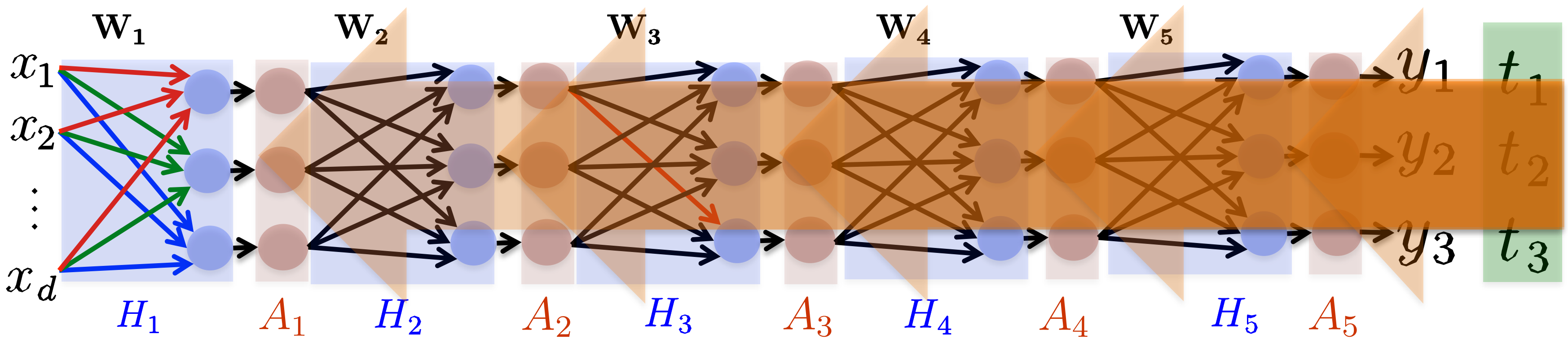
- Loop until no change in loss
  - Y=Forward(X)
  - L = Compute Loss(Y,T)
  - G = Backprop (L)
  - Update Parameters

Training



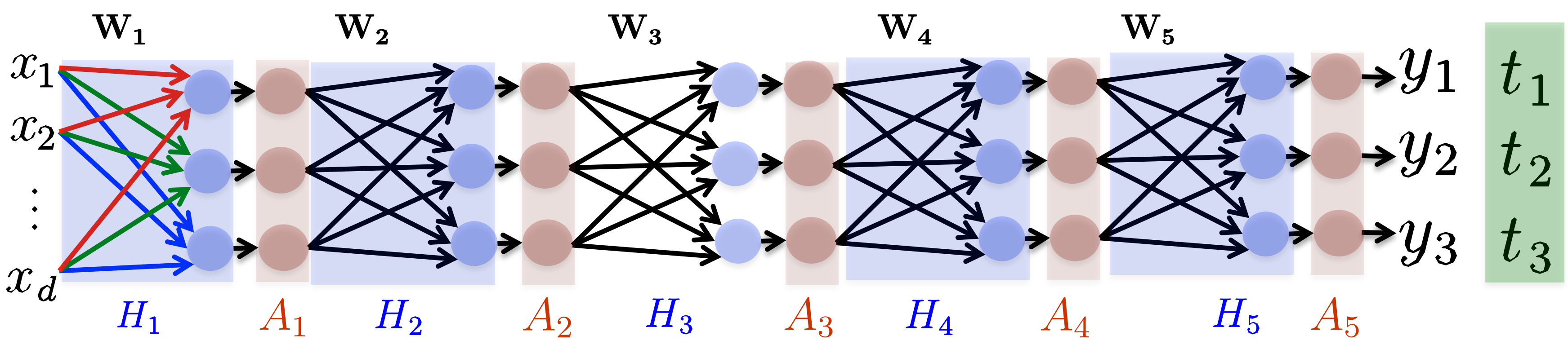
- Loop until no change in loss
  - Y=Forward(X)
  - L = Compute Loss(Y,T)
  - G = Backprop (L)
  - Update Parameters

Training



- Loop until no change in loss
  - Y=Forward(X)
  - L = Compute Loss(Y,T)
  - G = Backprop (L) → $\frac{\partial L}{\partial W_{l(ij)}}$
  - Update Parameters

Training



- Loop until no change in loss
  - Y=Forward(X)
  - L = Compute Loss(Y,T)
  - G = Backprop (L)
  - Update Parameters

$$W^{t+1} = W^t - \eta^t \frac{\partial L}{\partial W_{l(ij)}}$$

Torch

- Torch 7.0
  - Facebook and Google DeepMind
- Based on LUA
- Very easy to design Neural Network
- Very convenient to use GPU

LUA

- Interpreter
- Variables are global by default
- Universal data structure : the table

```
my_table = { 1, 2, 3 }
my_table = { my_var = 'hello', my_other_var = 'bye' }
my_table = { 1, 2, 99, my_var = 'hello' }
my_function = function() print('hello world') end
my_table[my_function] = 'this prints hello world'
my_function()
print(my_table[my_function])
```

```
Torch 7.0  Copyright (C) 2001-2011 Idiap, NEC Labs, NYU
hello world
this prints hello world
```

Torch Tensor

```
  ______             __   |  Torch7
 /_  __/__  ________/ /   |  Scientific computing for Lua.
  / / / _ \/ __/ __/ _ \  |  Type ? for help
 /_/  \___/_/  \__/_//_/  |  https://github.com/torch
                          |  http://torch.ch

th> A= torch.Tensor(3,3)
                                                                      [0.0001s]
th> A
 6.9103e-310  6.9103e-310  8.2681e-317
 8.2682e-317  8.2682e-317  8.2683e-317
 8.2683e-317  8.2684e-317  8.2684e-317
[torch.DoubleTensor of size 3x3]

                                                                      [0.0003s]
th>
```

```
                                                                      [0.0003s]
th> A:fill(1)
 1  1  1
 1  1  1
 1  1  1
[torch.DoubleTensor of size 3x3]

                                                                      [0.0002s]
th>
```

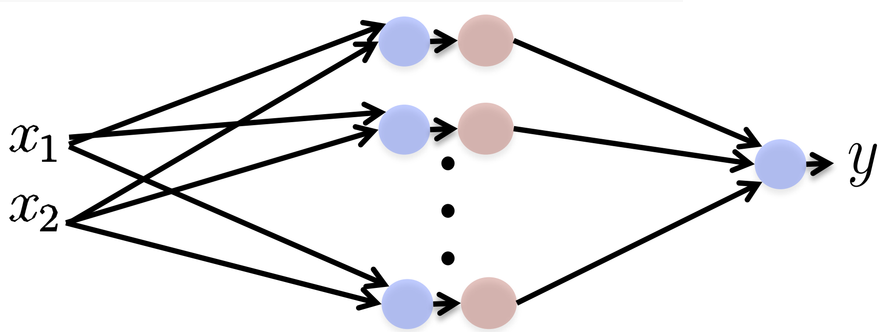
Simple Neural Network Training in Torch

```
require "nn"
mlp = nn.Sequential();  -- make a multi-layer perceptron
inputs = 2; outputs = 1; HUs = 20; -- parameters
mlp:add(nn.Linear(inputs, HUs))
mlp:add(nn.Tanh())
mlp:add(nn.Linear(HUs, outputs))
```

Loss function

We choose the Mean Squared Error criterion.

```
criterion = nn.MSECriterion()
```



Training

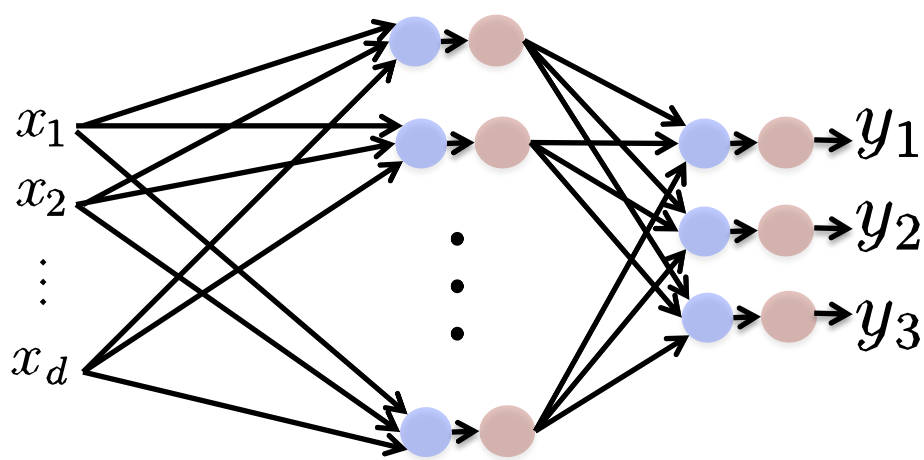
We create data *on the fly* and feed it to the neural network.

```
for i = 1,2500 do
  -- random sample
  local input= torch.randn(2);     -- normally distributed example in 2d
  local output= torch.Tensor(1);
  if input[1]*input[2] > 0 then  -- calculate label for XOR function
    output[1] = -1
  else
    output[1] = 1
  end

  -- feed it to the neural network and the criterion
  criterion:forward(mlp:forward(input), output)

  -- train over this example in 3 steps
  -- (1) zero the accumulation of the gradients
  mlp:zeroGradParameters()
  -- (2) accumulate gradients
  mlp:backward(input, criterion:backward(mlp.output, output))
  -- (3) update parameters with a 0.01 learning rate
  mlp:updateParameters(0.01)
end
```

```
> x = torch.Tensor(2)
> x[1] =  0.5; x[2] =  0.5; print(mlp:forward(x))

-0.6140
[torch.Tensor of dimension 1]

> x[1] =  0.5; x[2] = -0.5; print(mlp:forward(x))

 0.8878
[torch.Tensor of dimension 1]

> x[1] = -0.5; x[2] =  0.5; print(mlp:forward(x))

 0.8548
[torch.Tensor of dimension 1]

> x[1] = -0.5; x[2] = -0.5; print(mlp:forward(x))

-0.5498
[torch.Tensor of dimension 1]
```

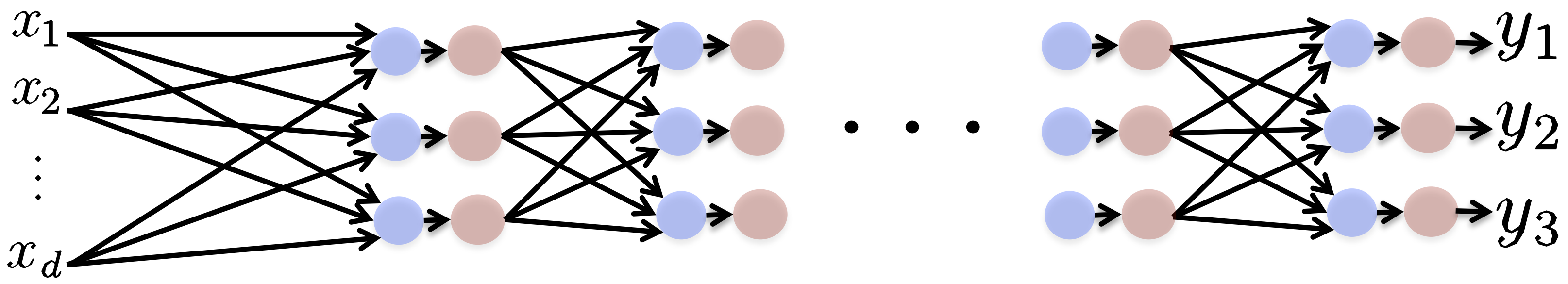
Universal Approximation Theorem

A network with a single hidden layer containing a finite number of neurons (i.e., a multilayer perceptron), can approximate continuous functions.

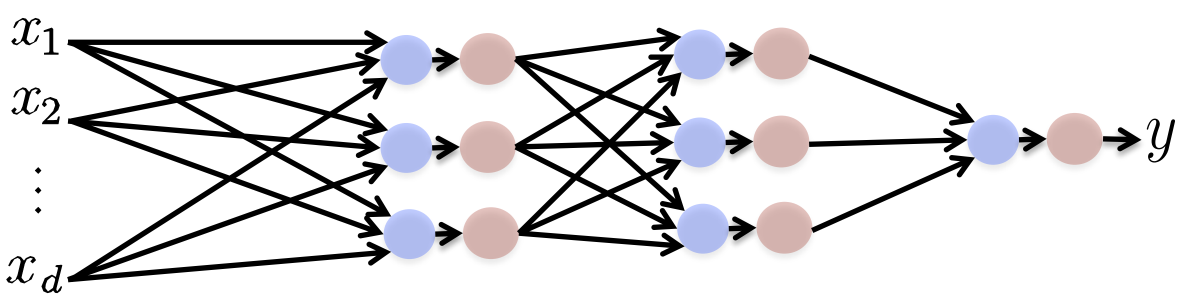


Why Deep Network?

- Feature hierarchy (We will see later)



Multi-Class Classification

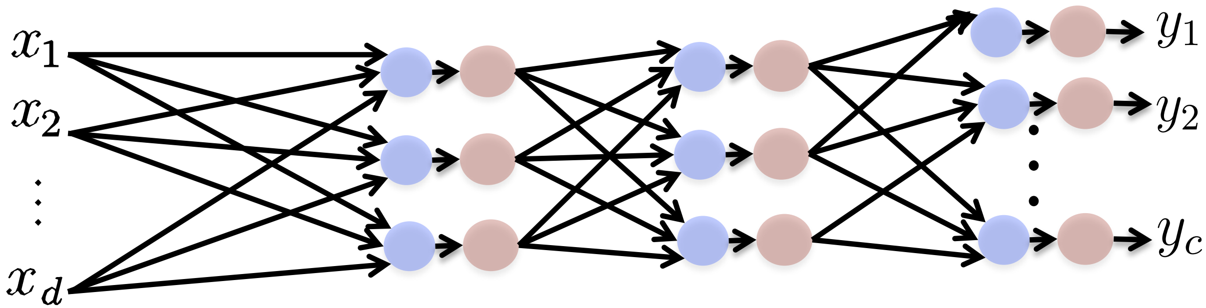


$$L(\mathbf{y}, \mathbf{t}) = \|\mathbf{y} - \mathbf{t}\|^2$$

Binary Class : $t \in \{-1, +1\}$

Multi Class : $t \in \{1, 2, 3, \ldots, c\}$ Unbalanced penalization

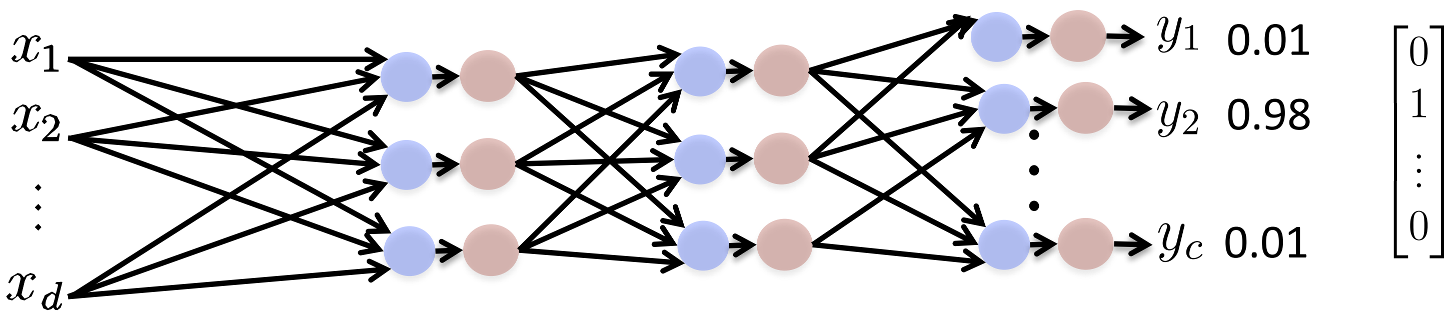
Multi-Class Classification



$$L(\mathbf{y}, \mathbf{t}) = \|\mathbf{y} - \mathbf{t}\|^2$$

Multi Class : $\mathrm{t} \in \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix} \cdots \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}$

Soft Max

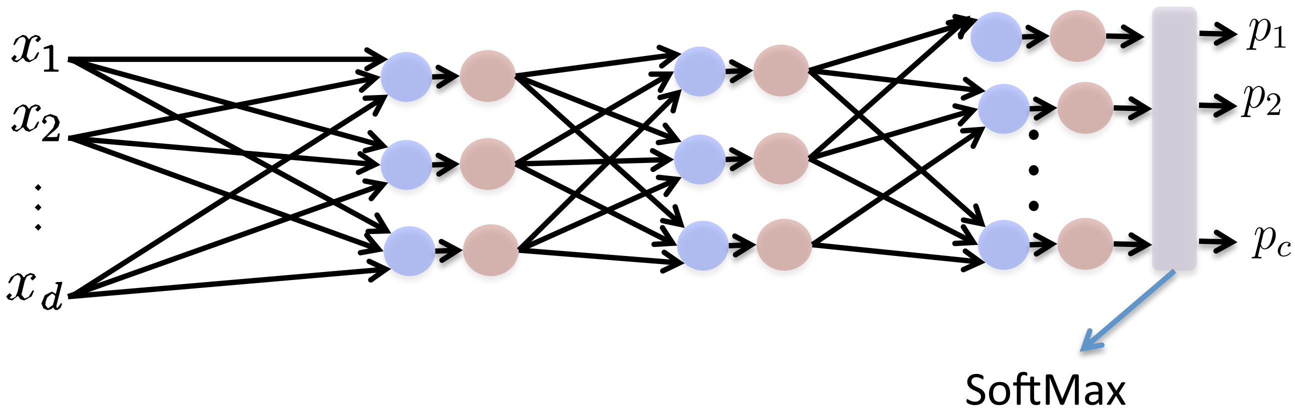


$$L(\mathbf{y}, \mathbf{t}) = \|\mathbf{y} - \mathbf{t}\|^2$$

Multi Class : $\mathbf{t} \in \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix} \cdots \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}$

$$P(\mathbf{y}) = \frac{e^{y_i}}{\sum_{j=1}^{c} e^{y_j}}$$

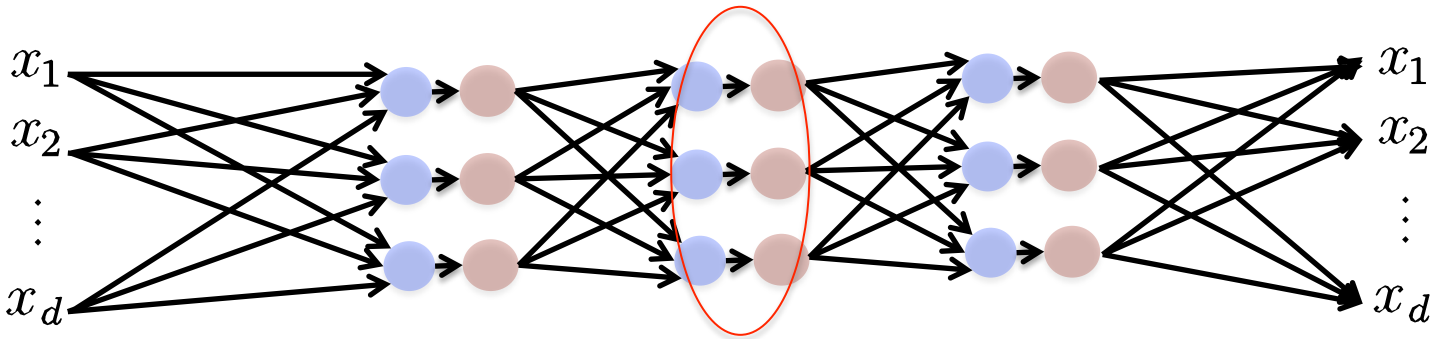
Cross Entropy Loss



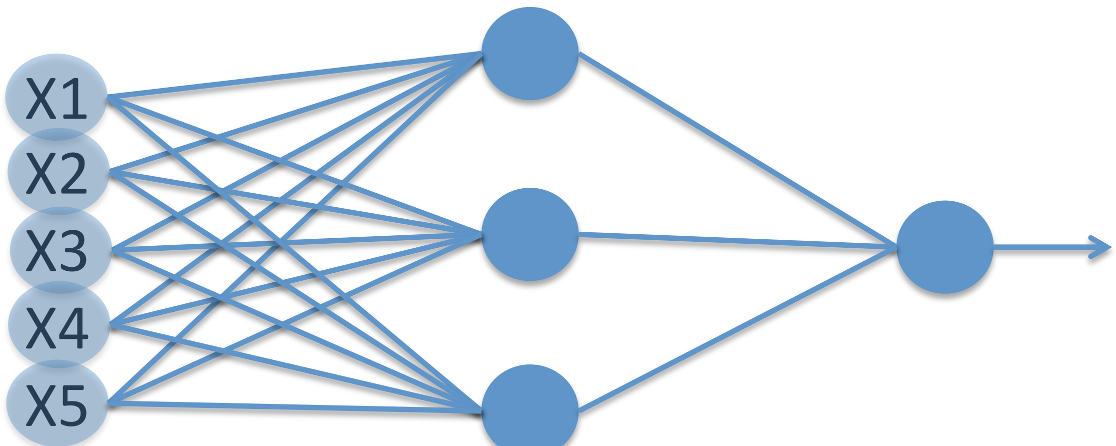
Multi Class : $t \in \{1, 2, 3, \ldots, c\}$

$$L(P, t) = -\log p_t$$

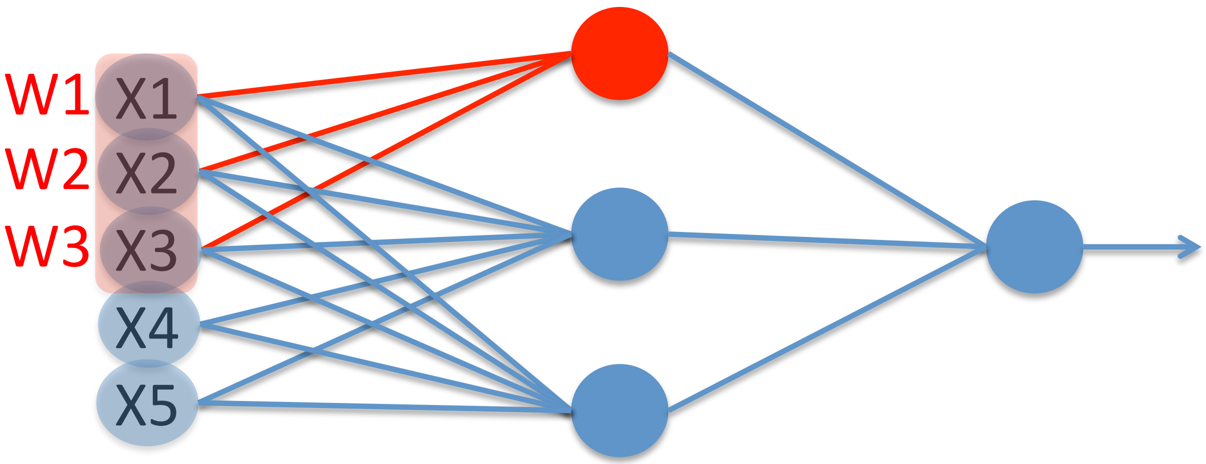
Auto Encoder



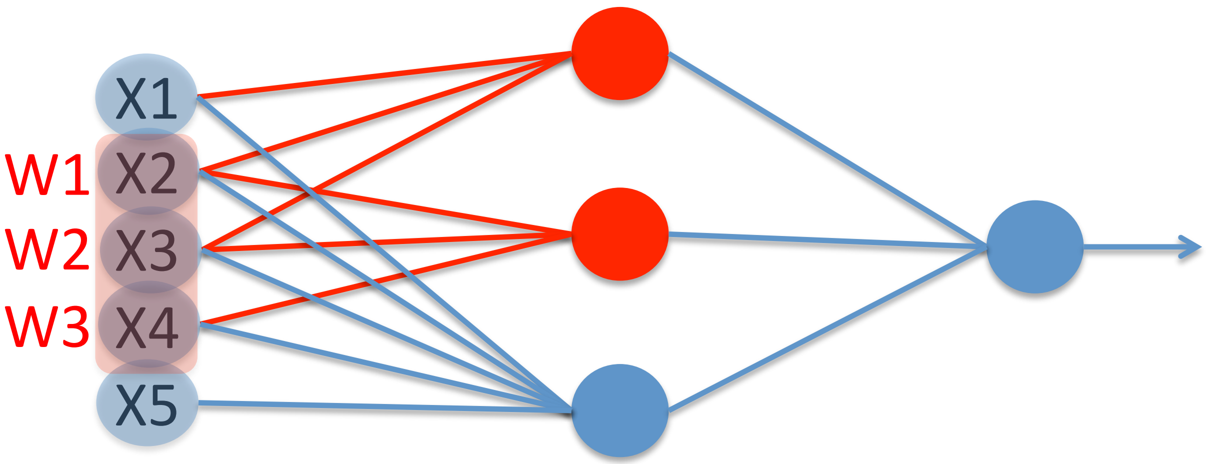
Convolutional Neural Networks (CNN)



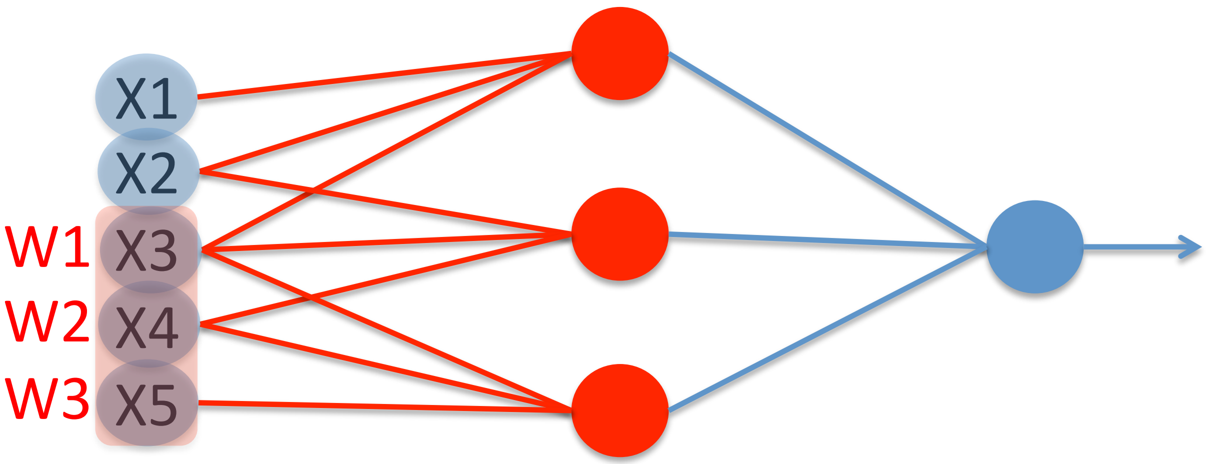
Convolutional Neural Networks (CNN)



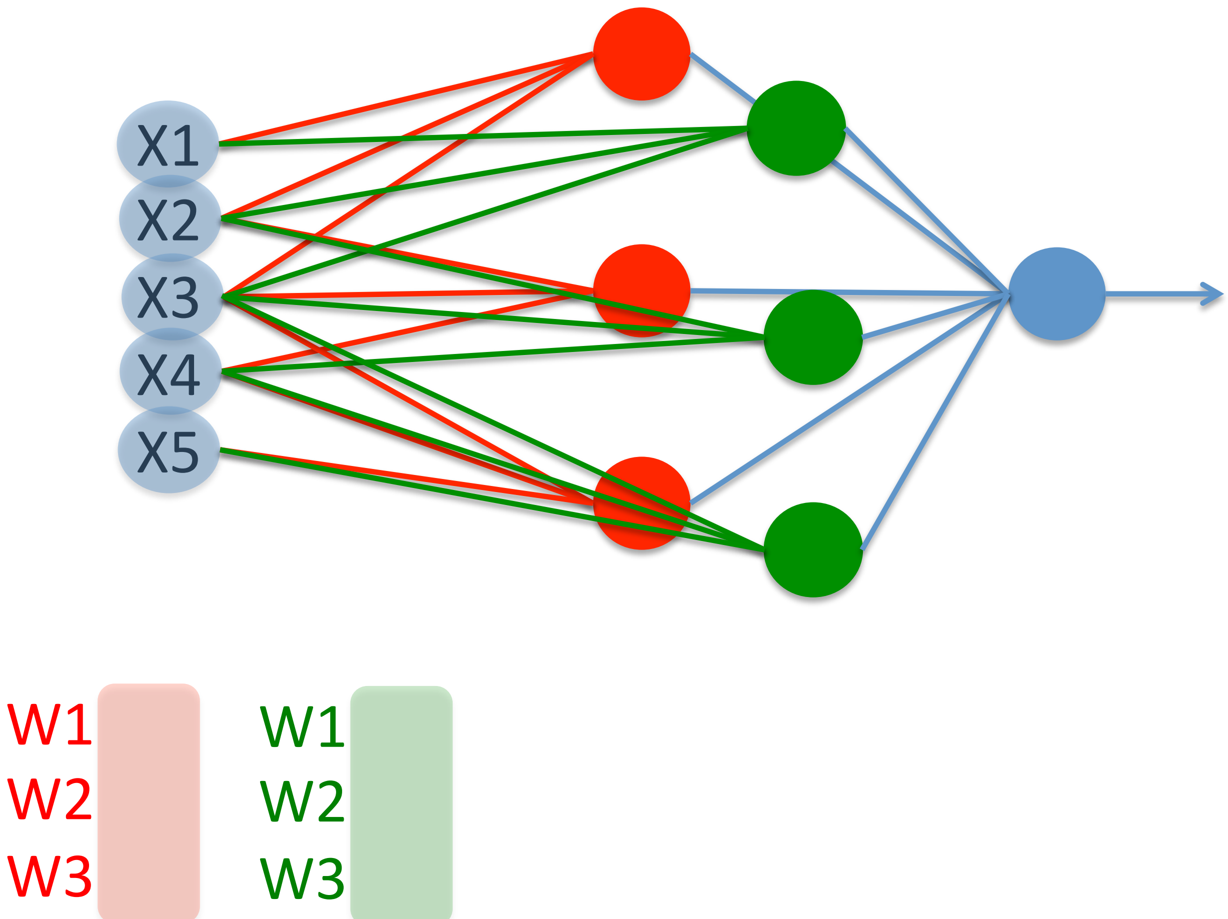
Convolutional Neural Networks (CNN)

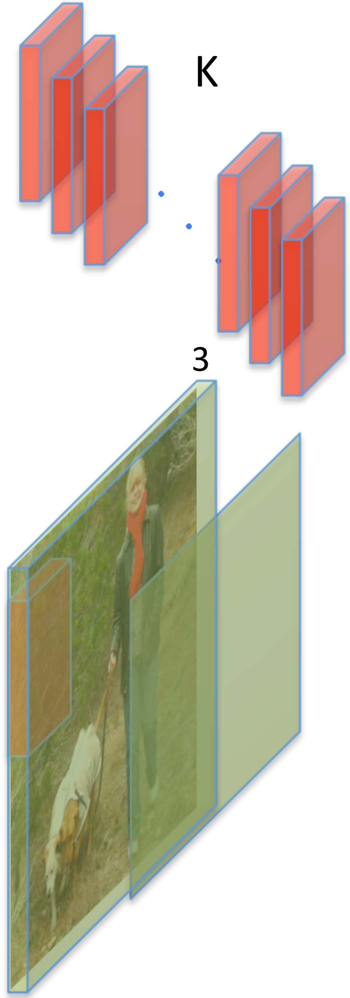


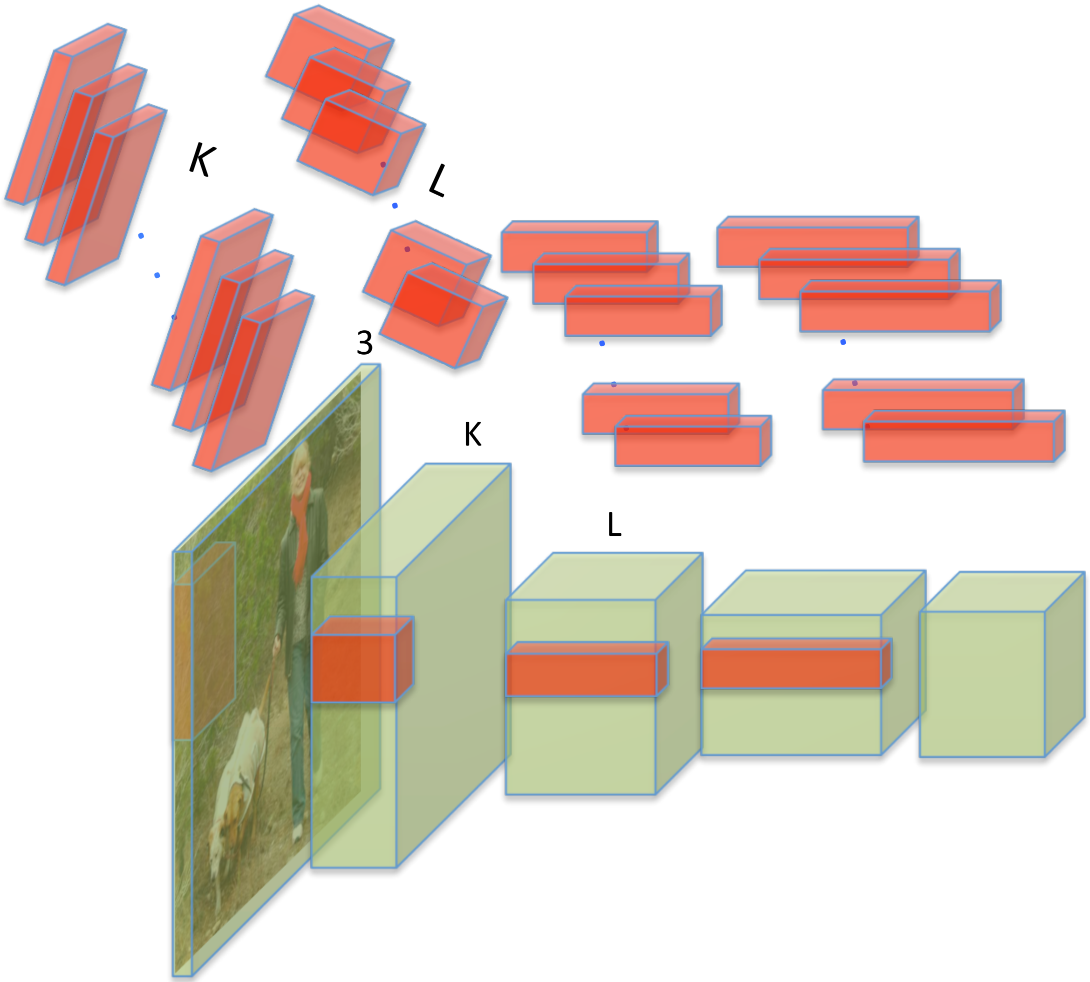
Convolutional Neural Networks (CNN)

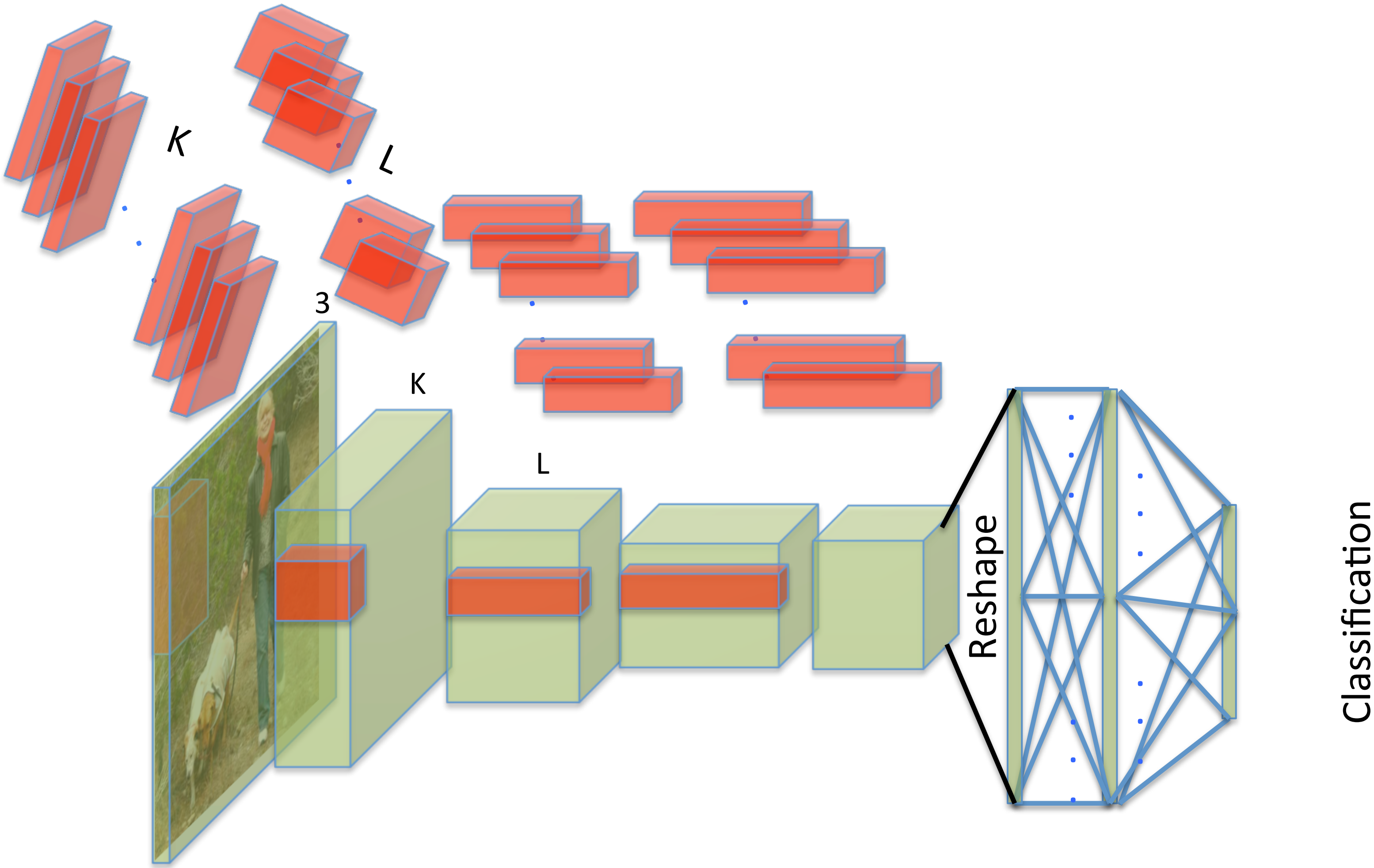


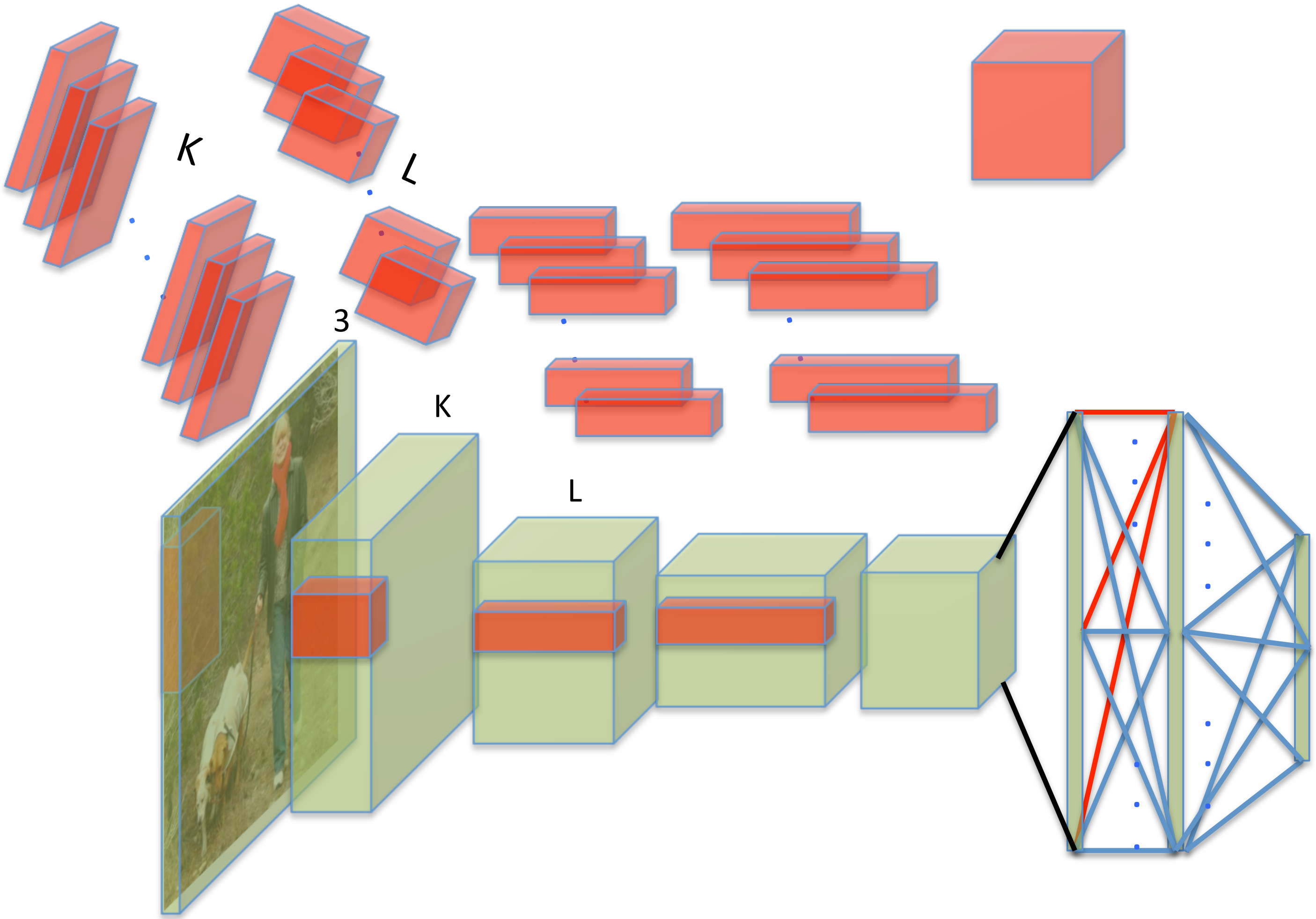
Convolutional Neural Networks (CNN)

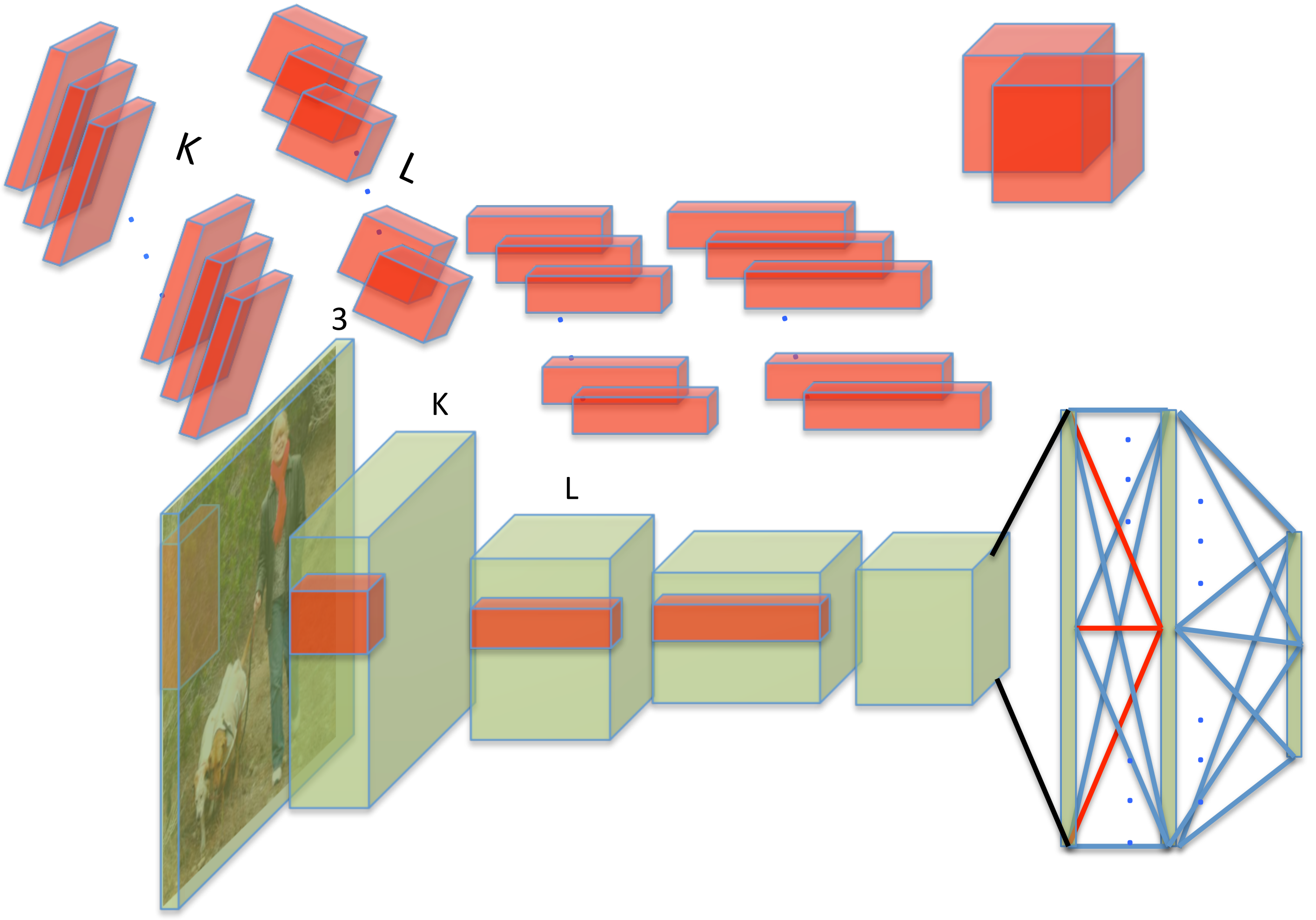


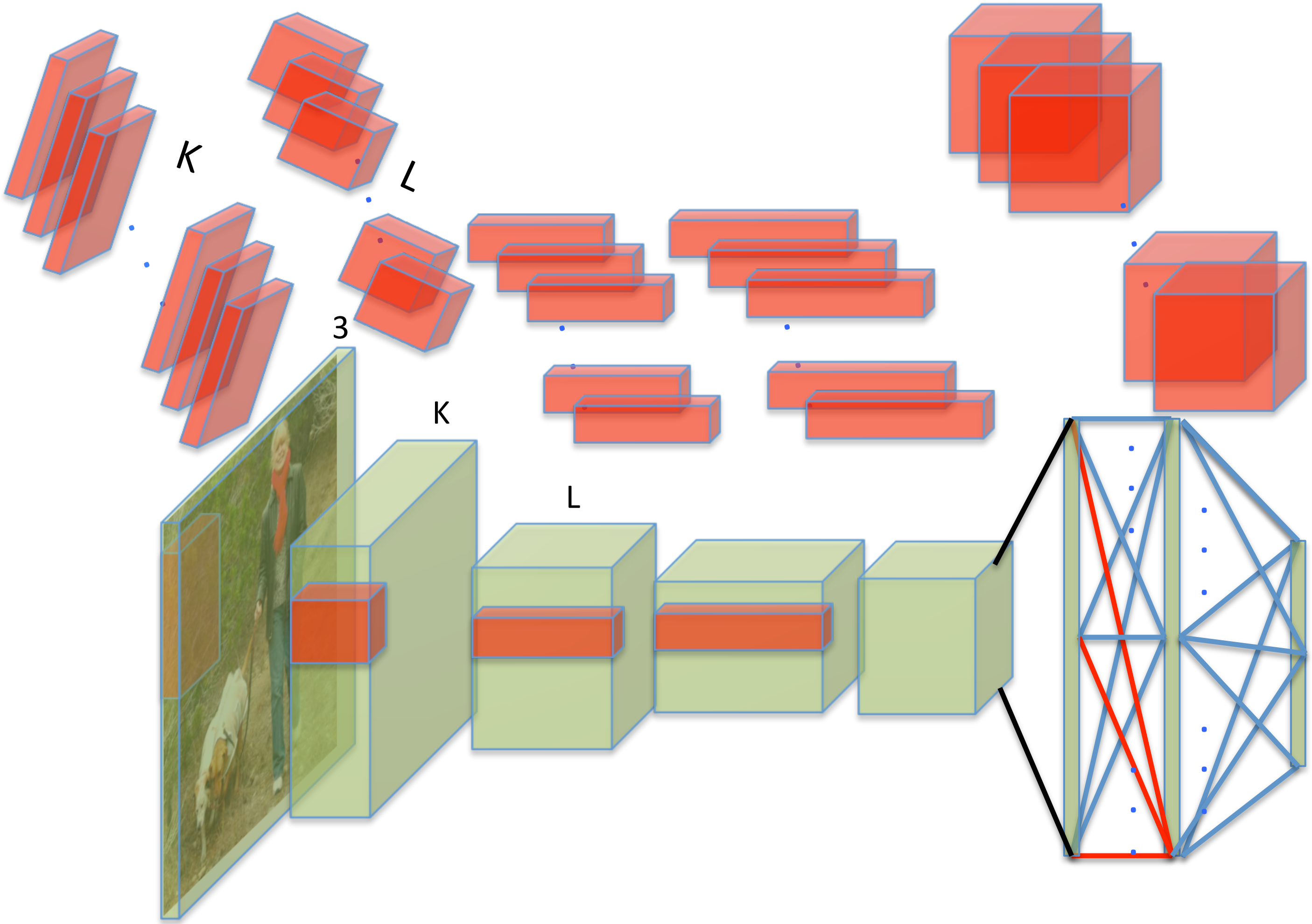

K
3


K
L
3
K
L

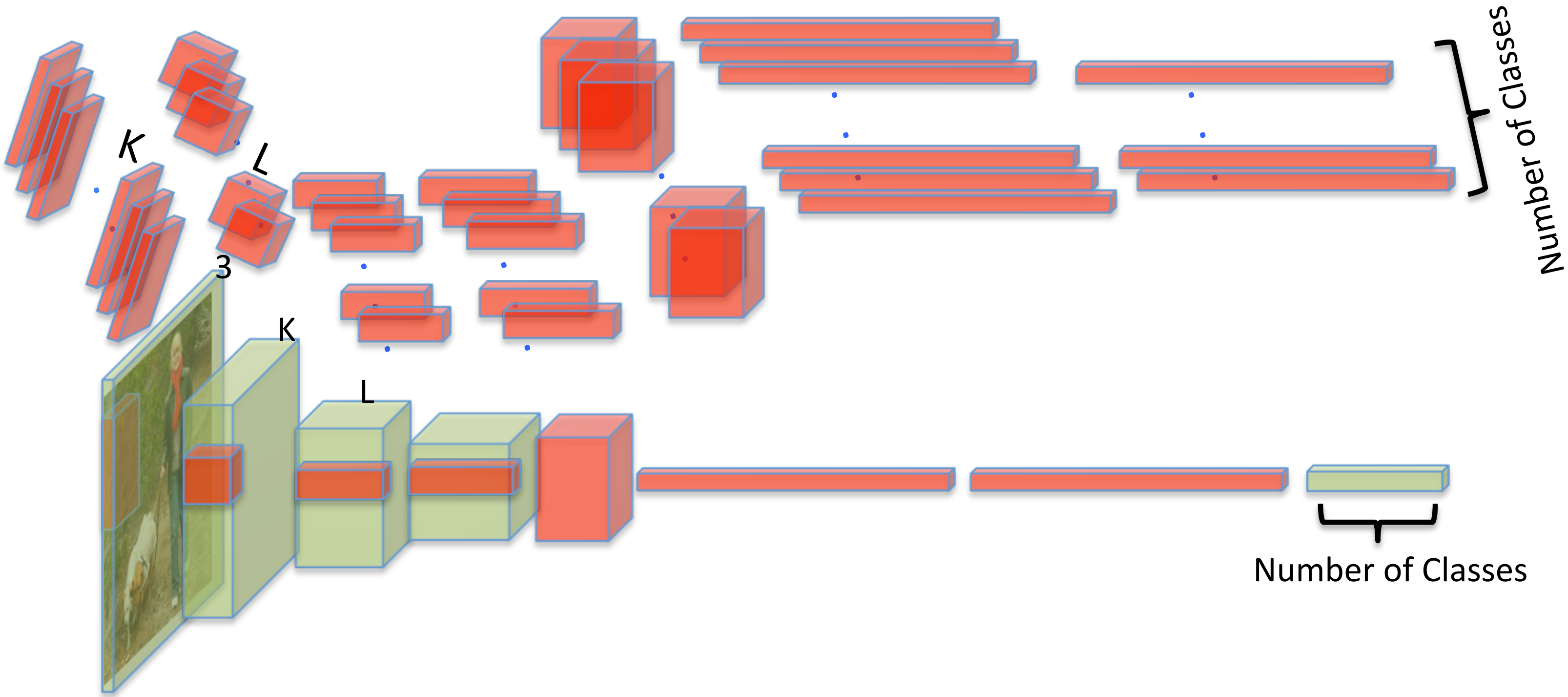

K
L
3
K
L
Reshape
Classification


K
L
3
K
L

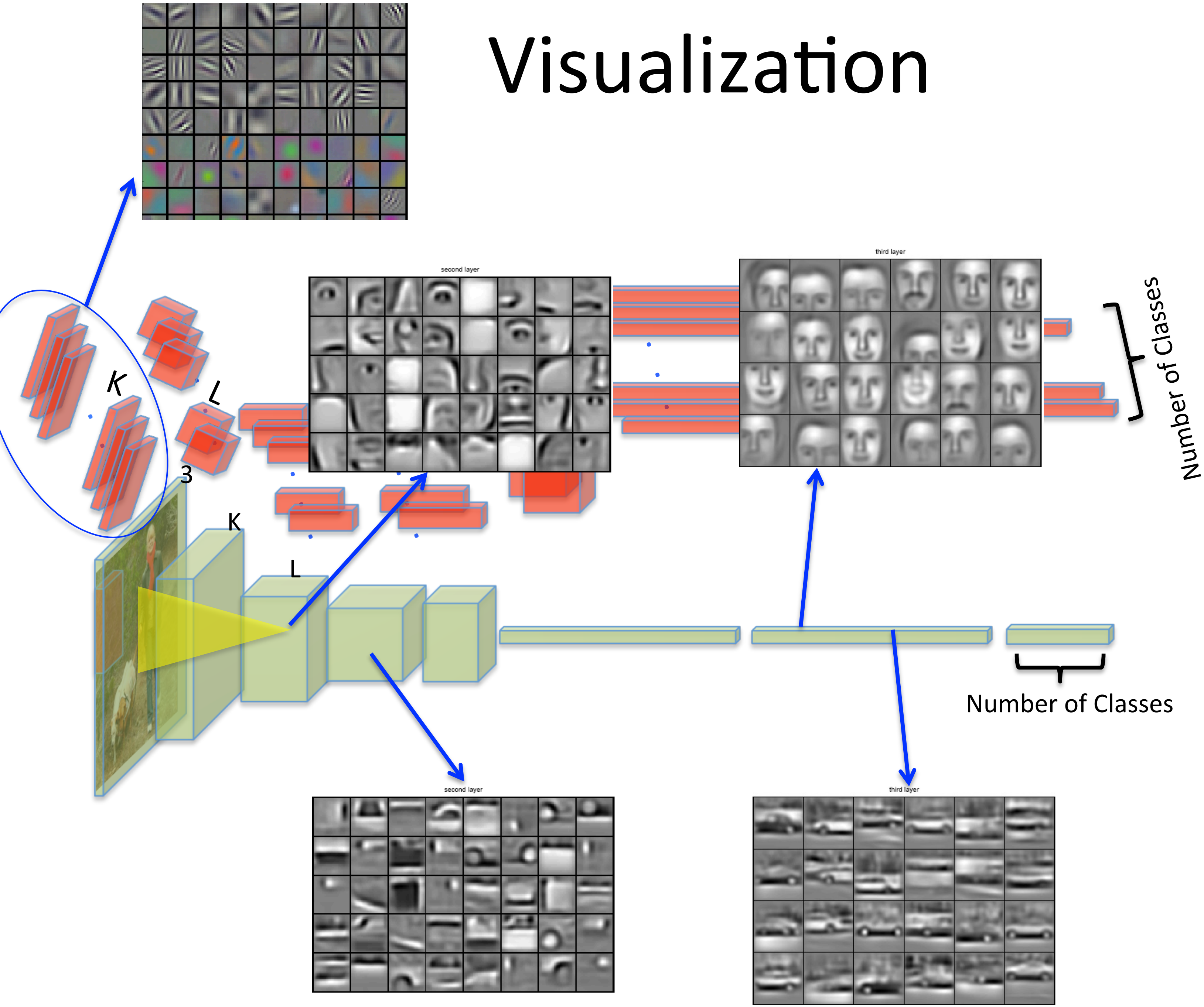

K
L
3
K
L


K
L
3
K
L

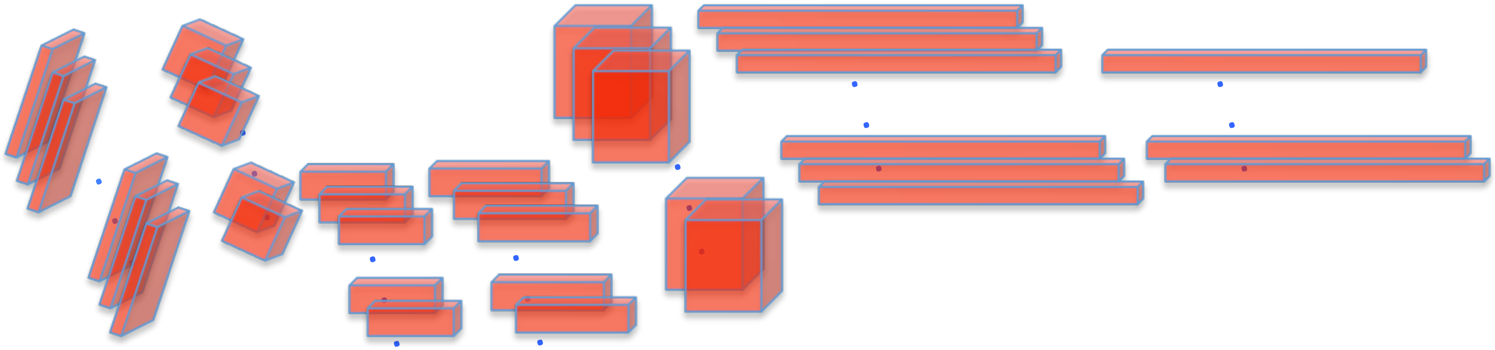
All The Way Convolutional



Visualization



CNNs are expensive: Memory



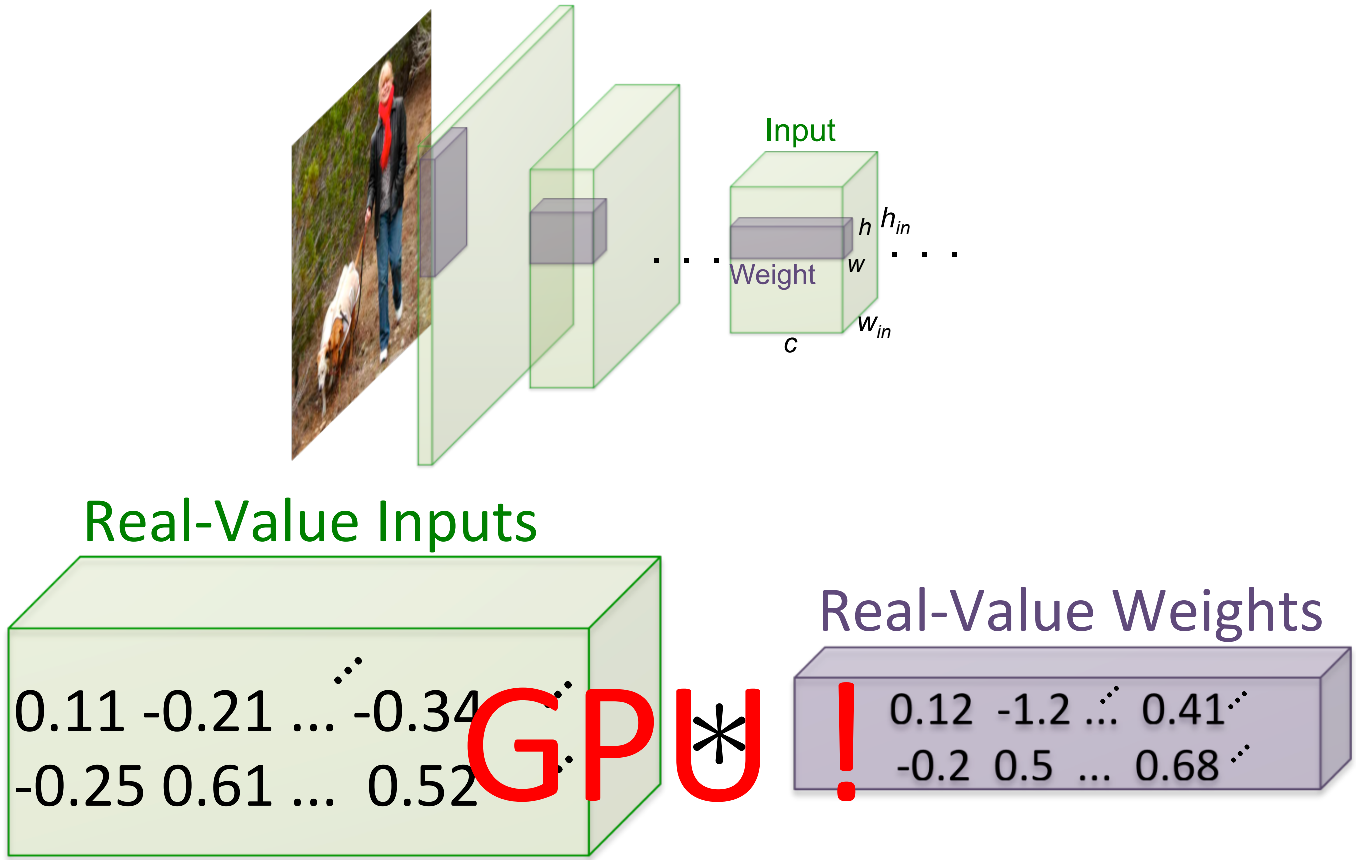
Number of parameters to learn:

- 60 M
- 140 M

Memory :

- 475 M
- 1.1 GB

CNNs are expensive: Computation



Number of Operations :

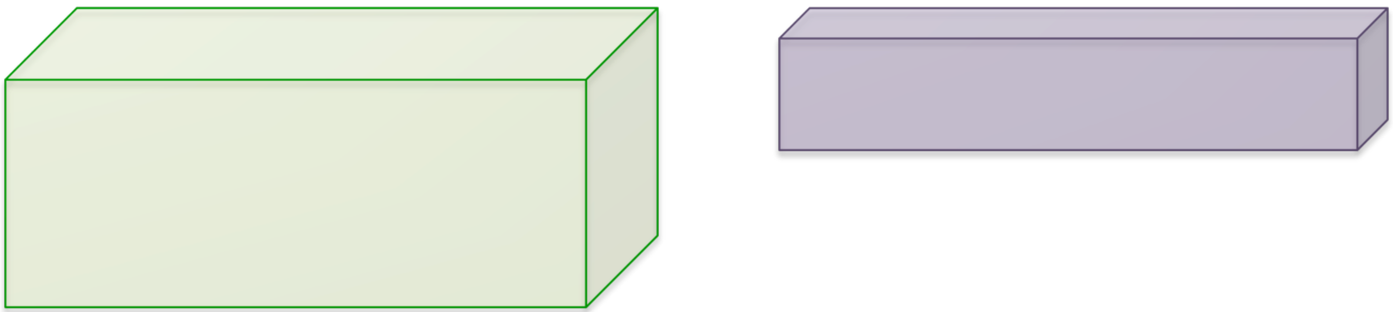
- AlexNet → 1.5B FLOPs
- VGG → 19.6B FLOPs

Inference time on CPU :

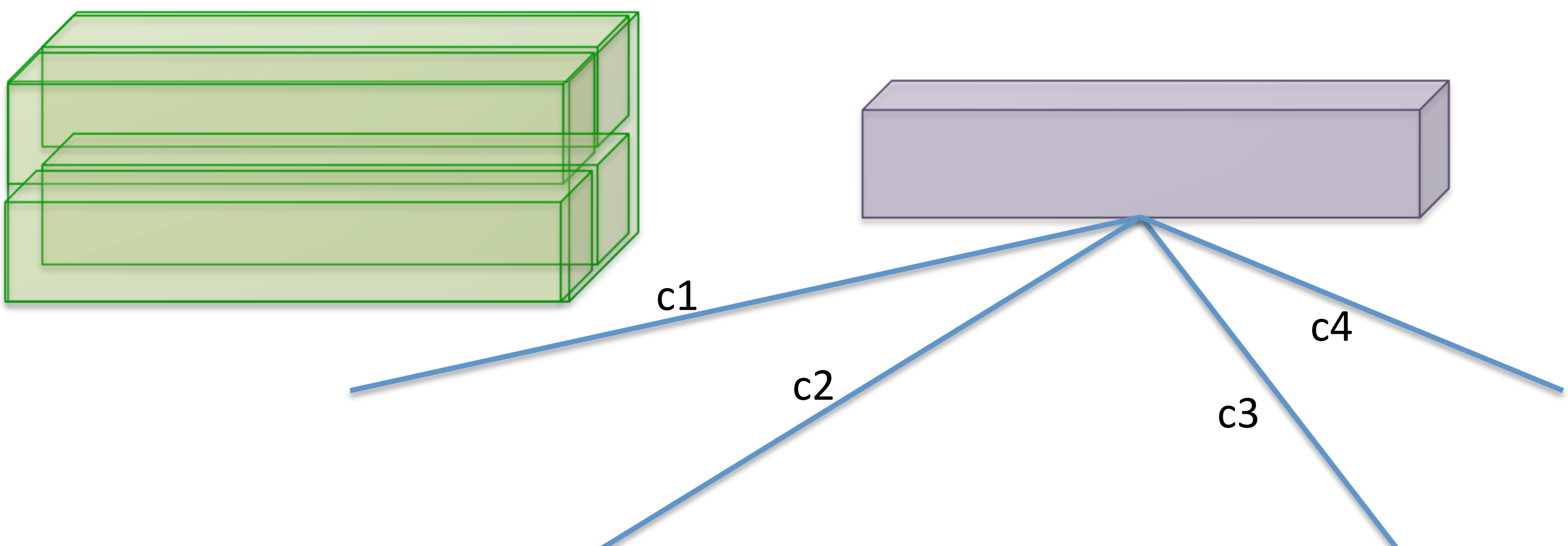
- AlexNet → ~3 fps
- VGG → ~0.25 fps

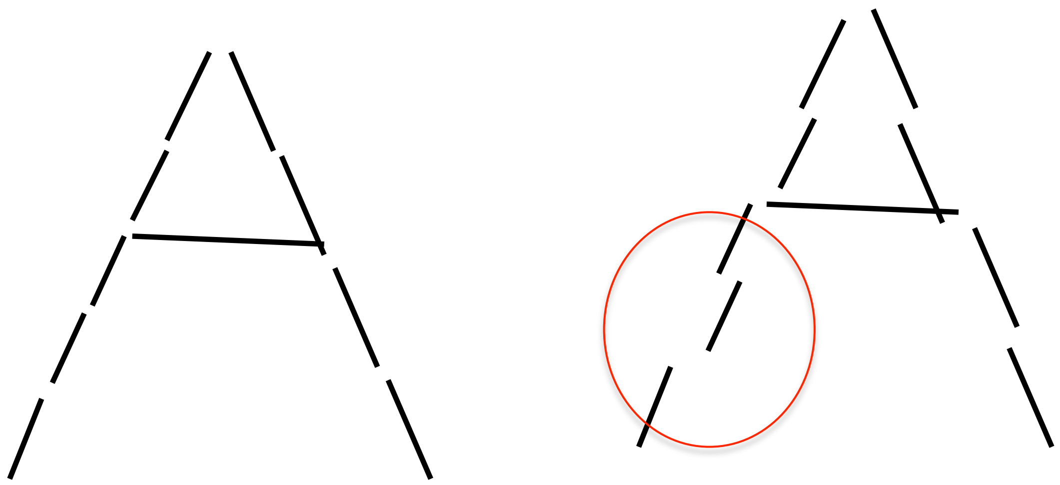
Convolution

- **CPU**



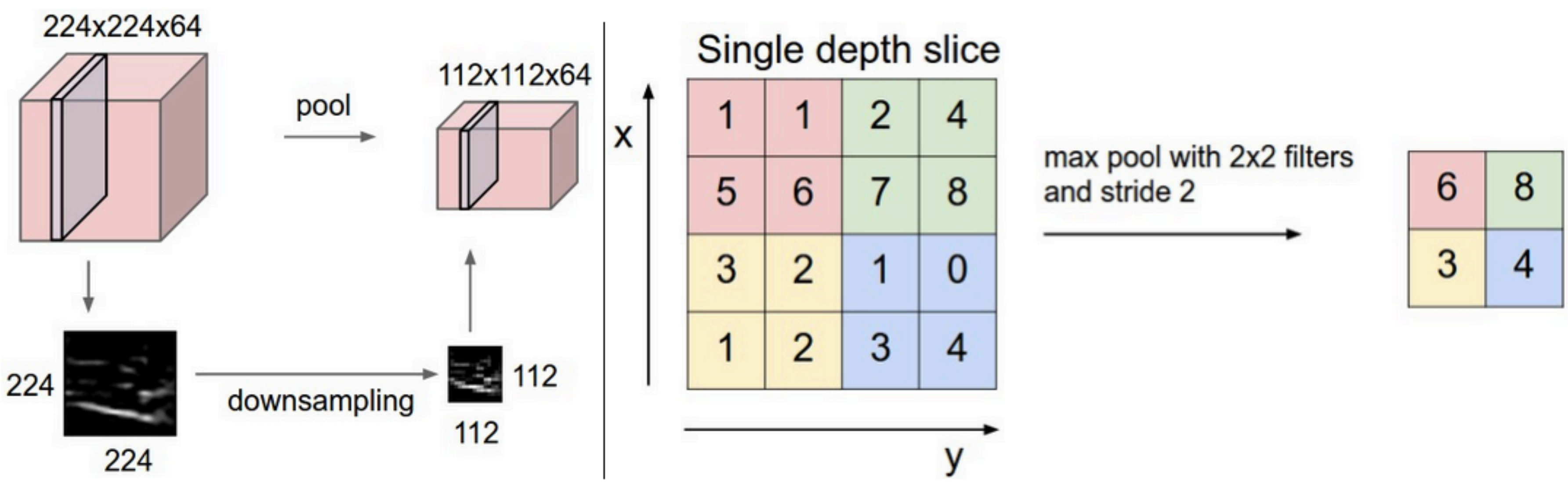
- **GPU**



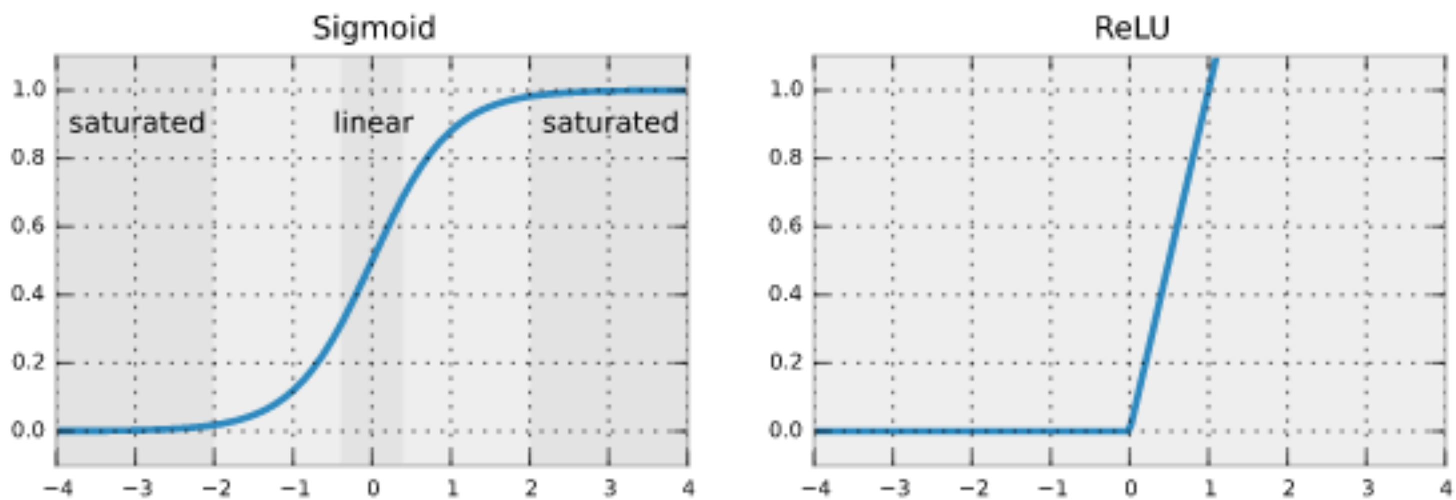


Max Pooling

- Adds more non-linearity
- Robust against small spatial variations



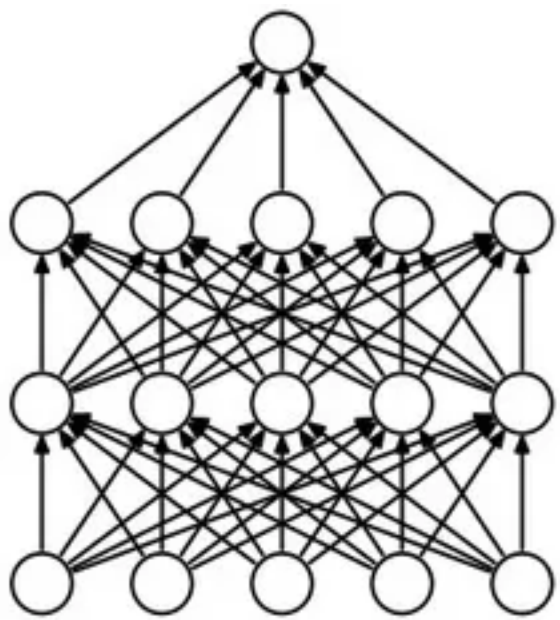
Rectified Linear Unit
ReLU



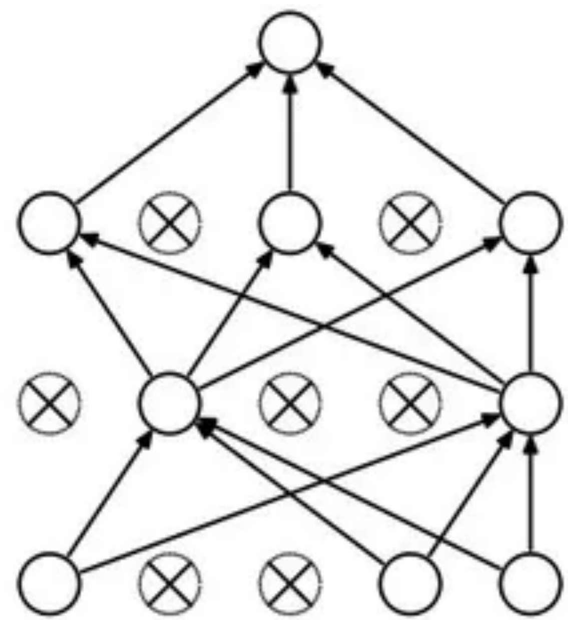
$$f(x) = \max(0, x)$$

$$g_f = \begin{cases} g_x = 1 & \text{if } x > 0 \\ g_x = 0 & \text{if } x \leq 0 \end{cases}$$

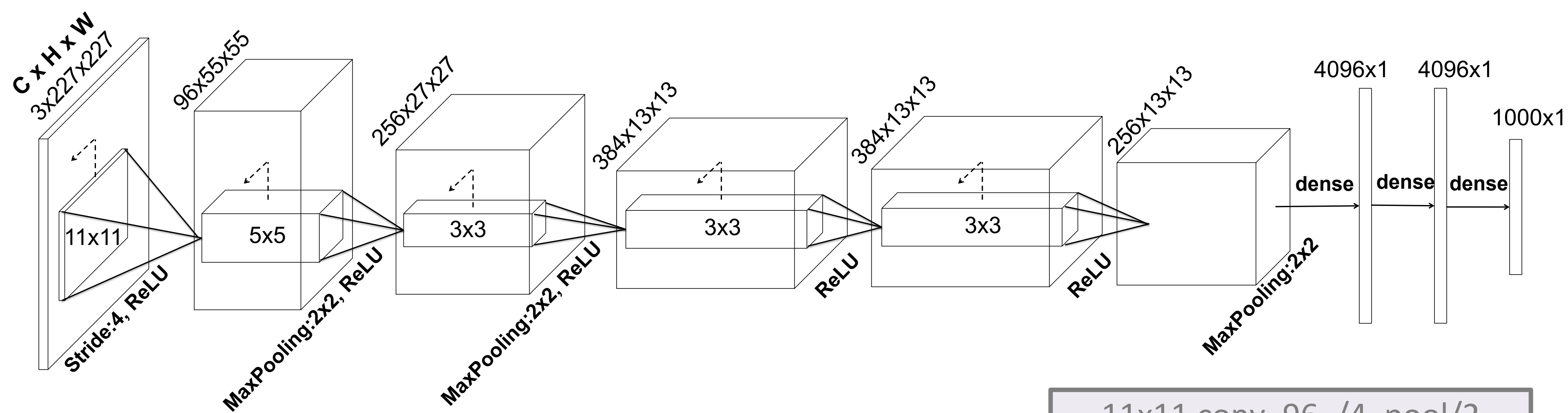
Generalization by Dropout



(a) Standard Neural Net

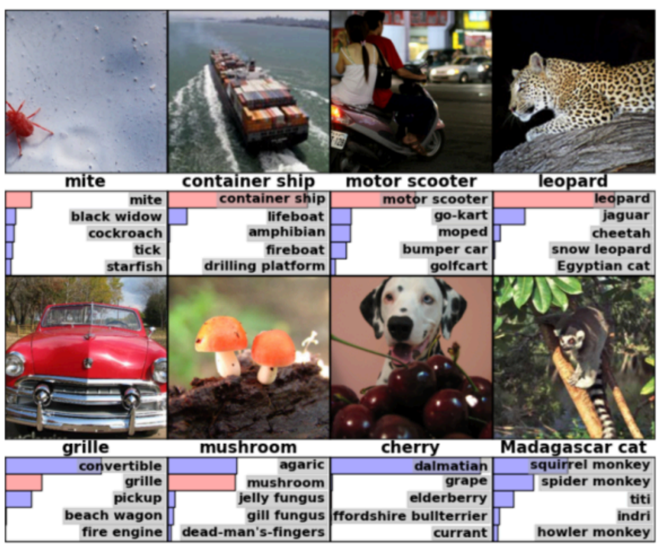


(b) After applying dropout.

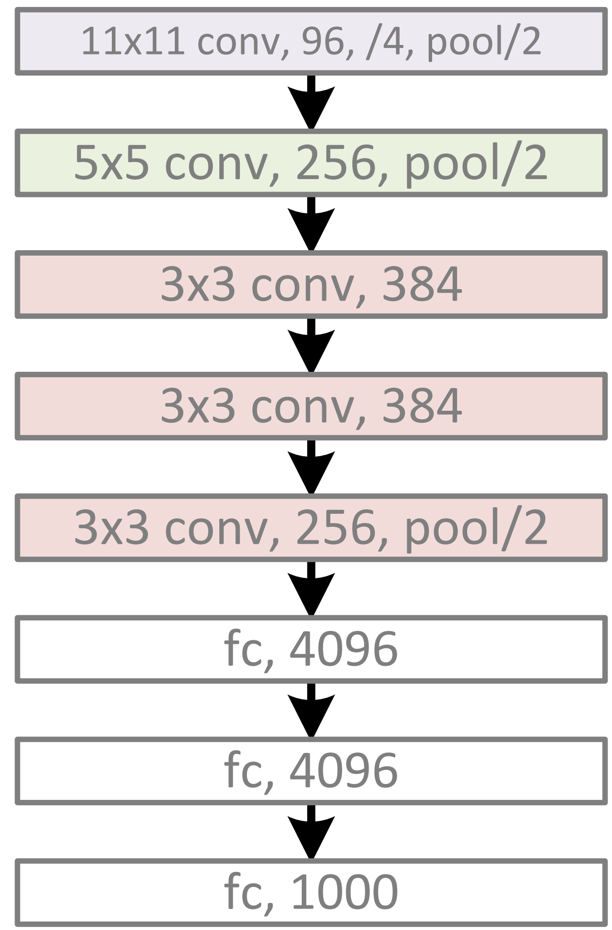


[Alex Krizhevsky, NIPS 2012] AlexNet

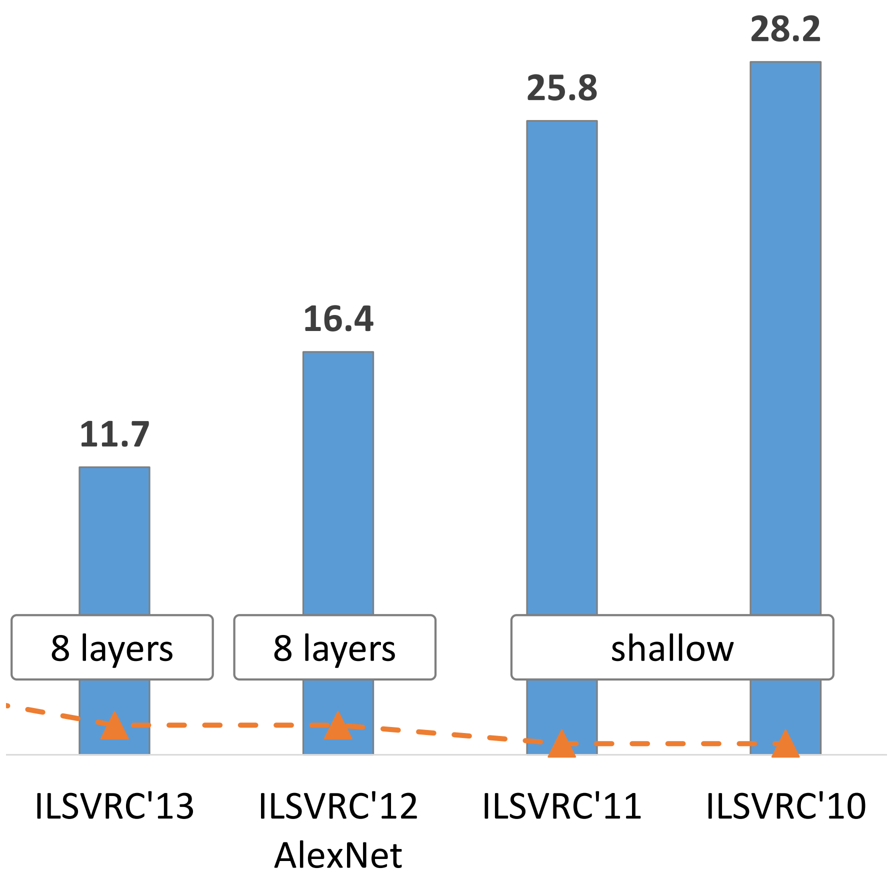
- 1000 Category
- 1.2M Training Images
- 150K Test Images

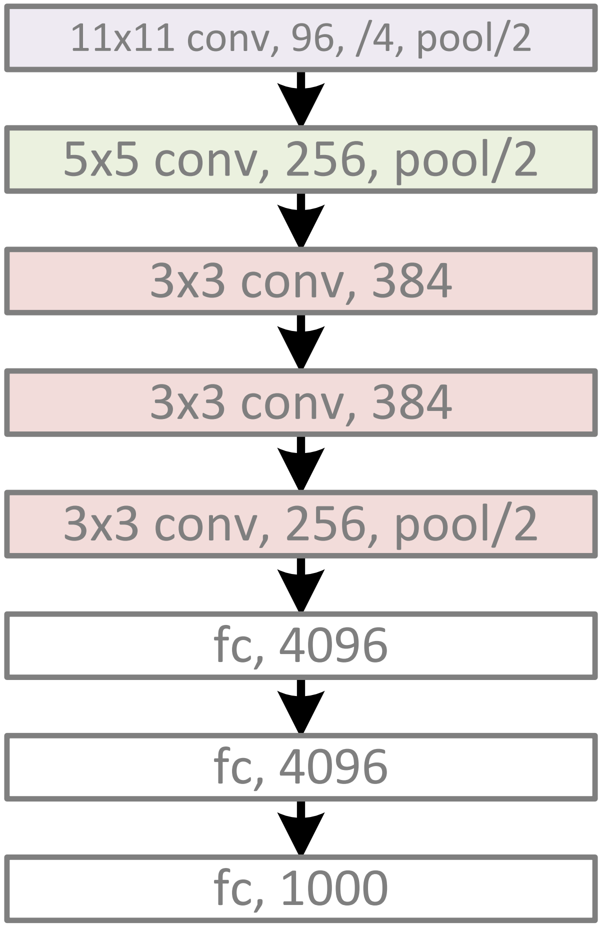


Top – 5 Error



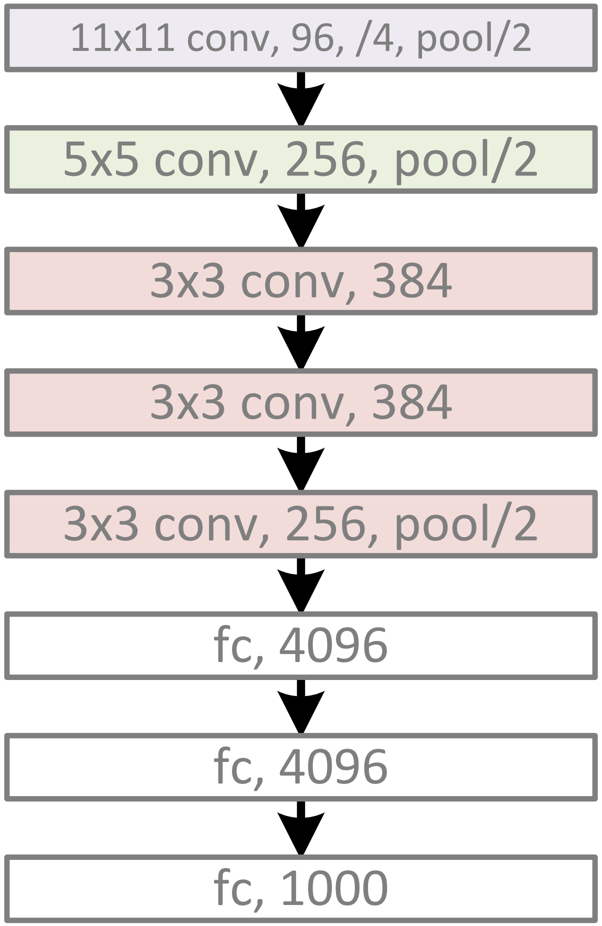
AlexNet 2012





Alex Net in Torch

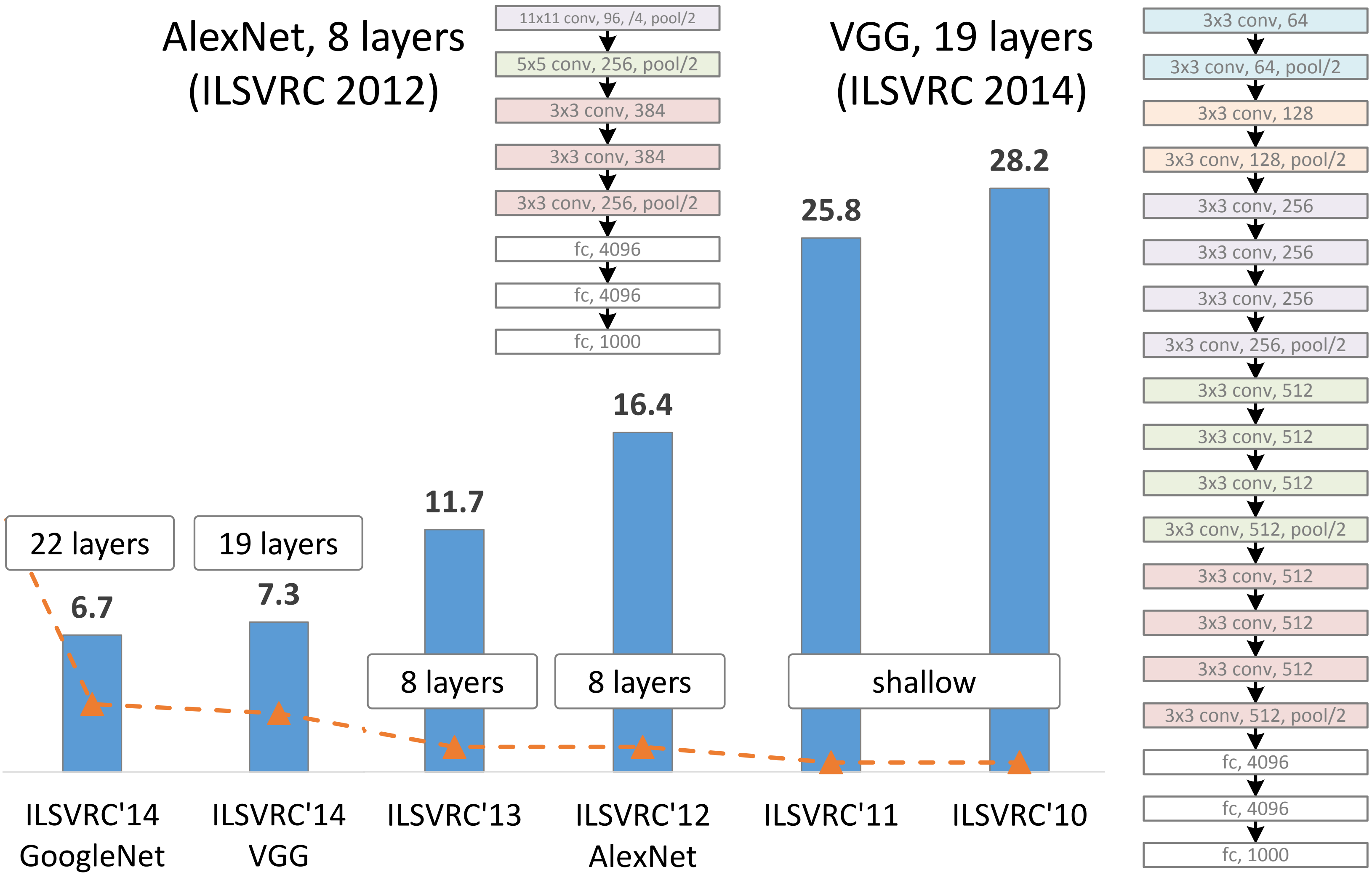
```
model = nn.Sequential()
model:add(cudnn.SpatialConvolution(3,96,11,11,4,4,2,2))
model:add(cudnn.ReLU())
model:add(nn.SpatialMaxPooling(3,3,2,2))
model:add(cudnn.SpatialConvolution(96,256,5,5,1,1,2,2))
model:add(cudnn.ReLU())
model:add(nn.SpatialMaxPooling(3,3,2,2))
model:add(cudnn.SpatialConvolution(256,384,3,3,1,1,1,1))
model:add(cudnn.ReLU())
model:add(cudnn.SpatialConvolution(384,384,3,3,1,1,1,1))
model:add(cudnn.ReLU())
model:add(cudnn.SpatialConvolution(384,256,3,3,1,1,1,1))
model:add(nn.ReLU())
model:add(nn.SpatialMaxPooling(3,3,2,2))

model:add(nn.View(256*6*6))
model:add(nn.Linear(256*6*6, 4096))
model:add(cudnn.ReLU())
model:add(nn.Dropout(0.5))
model:add(nn.Linear(4096, 4096))
model:add(cudnn.ReLU())
model:add(nn.Dropout(0.5))
model:add(nn.Linear(4096, 1000)
```



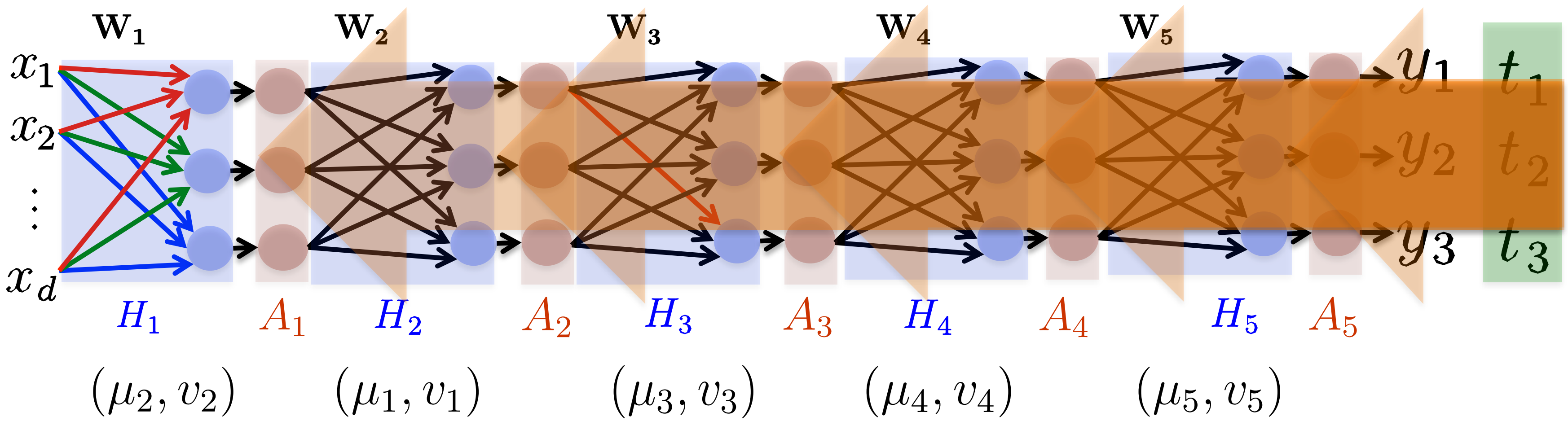
Going on GPU is easy in Torch

```
model = model:cuda()
```

Revolution of Depth



Vanishing Gradients

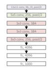


$$\frac{\partial L}{\partial W_{3(ij)}} = \frac{\partial L}{\partial A_5} \cdot \frac{\partial A_5}{\partial H_5} \cdot \frac{\partial H_5}{\partial A_4} \cdot \frac{\partial A_4}{\partial H_4} \cdot \frac{\partial H_4}{\partial A_{3(i)}} \cdot \frac{\partial A_{3(i)}}{\partial H_{3(i)}} \cdot \frac{\partial H_{3(i)}}{\partial W_{3(ij)}}$$

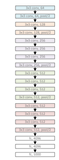
Bach Normalization: $$BN_l(x_i) = \frac{x_i - \mu_l}{v_l}$$

Revolution of Depth

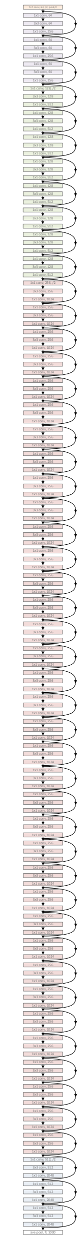
AlexNet, 8 layers
(ILSVRC 2012)

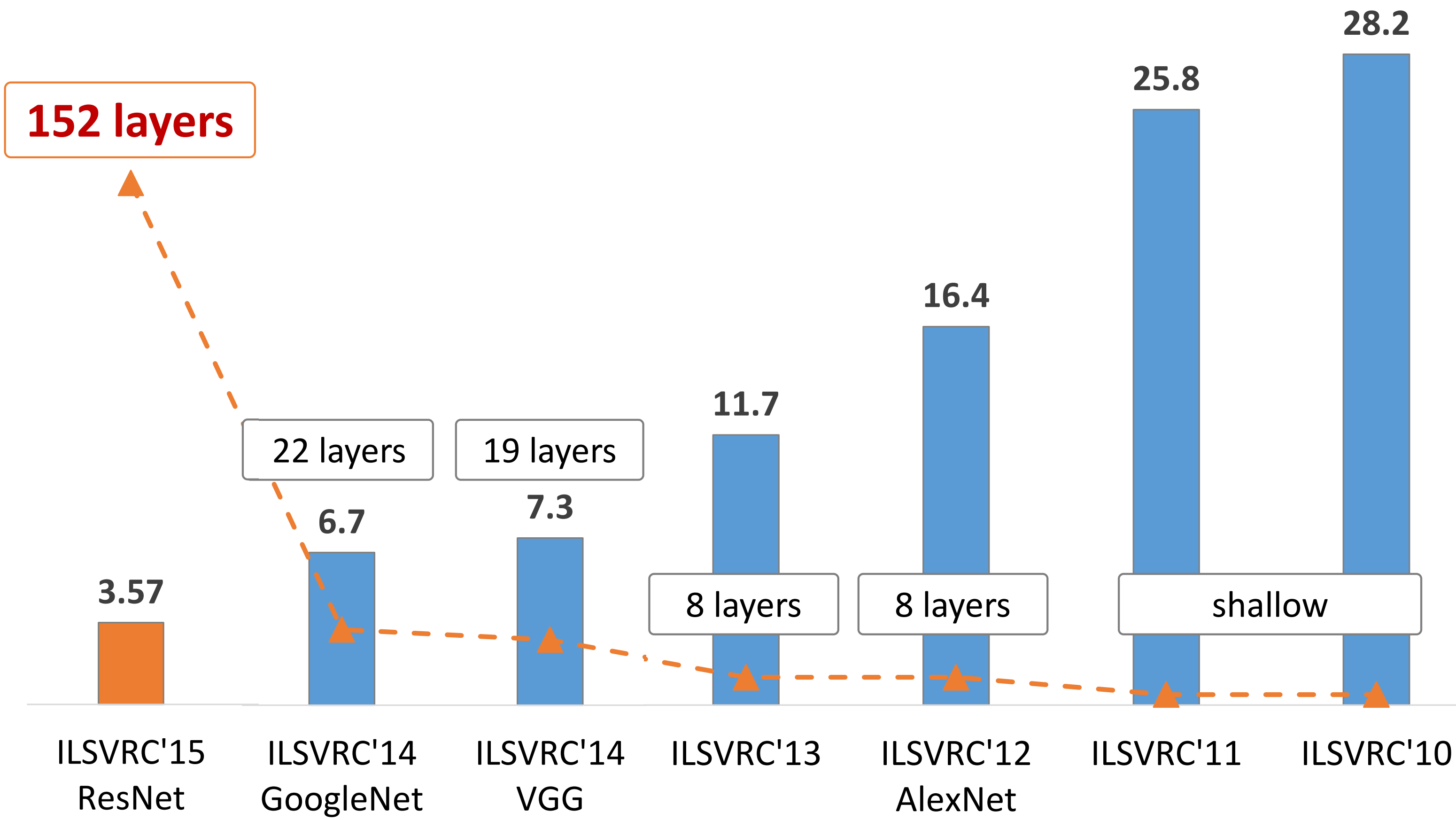


VGG, 19 layers
(ILSVRC 2014)



ResNet, 152 layers
(ILSVRC 2015)





Next:
Object Detection with CNNs