Deep Learning

Deterministic Neural Networks

Zhirong Wu

MIT Technology Review

10 BREAKTHROUGH TECHNOLOGIES 2013

Deep Learning

With massive amounts of computational power, machines can now recognize objects and translate speech in real time. Artificial intelligence is finally getting smart.

Neural Net Events

IMAGENET

founded by
Warren McCulloch
and Walter Pitts

back propagation by
Rumelhart and Hinton

Microsoft
deep networks in
speech recognition

ImageNet
classification over
millions of images

1943

1969

1986

2006

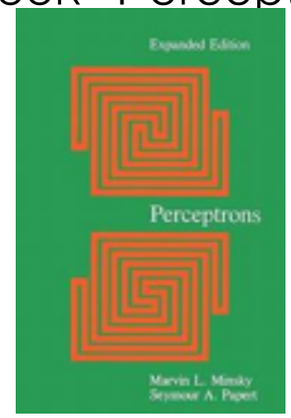
2011

2012

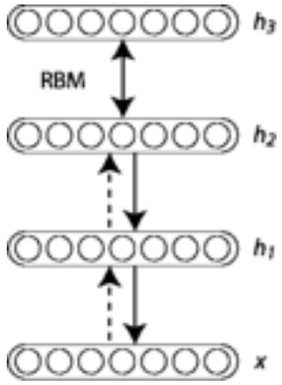
2012

2013

criticism by Minsky in
his book “Perceptron”



deep belief networks
by Hinton

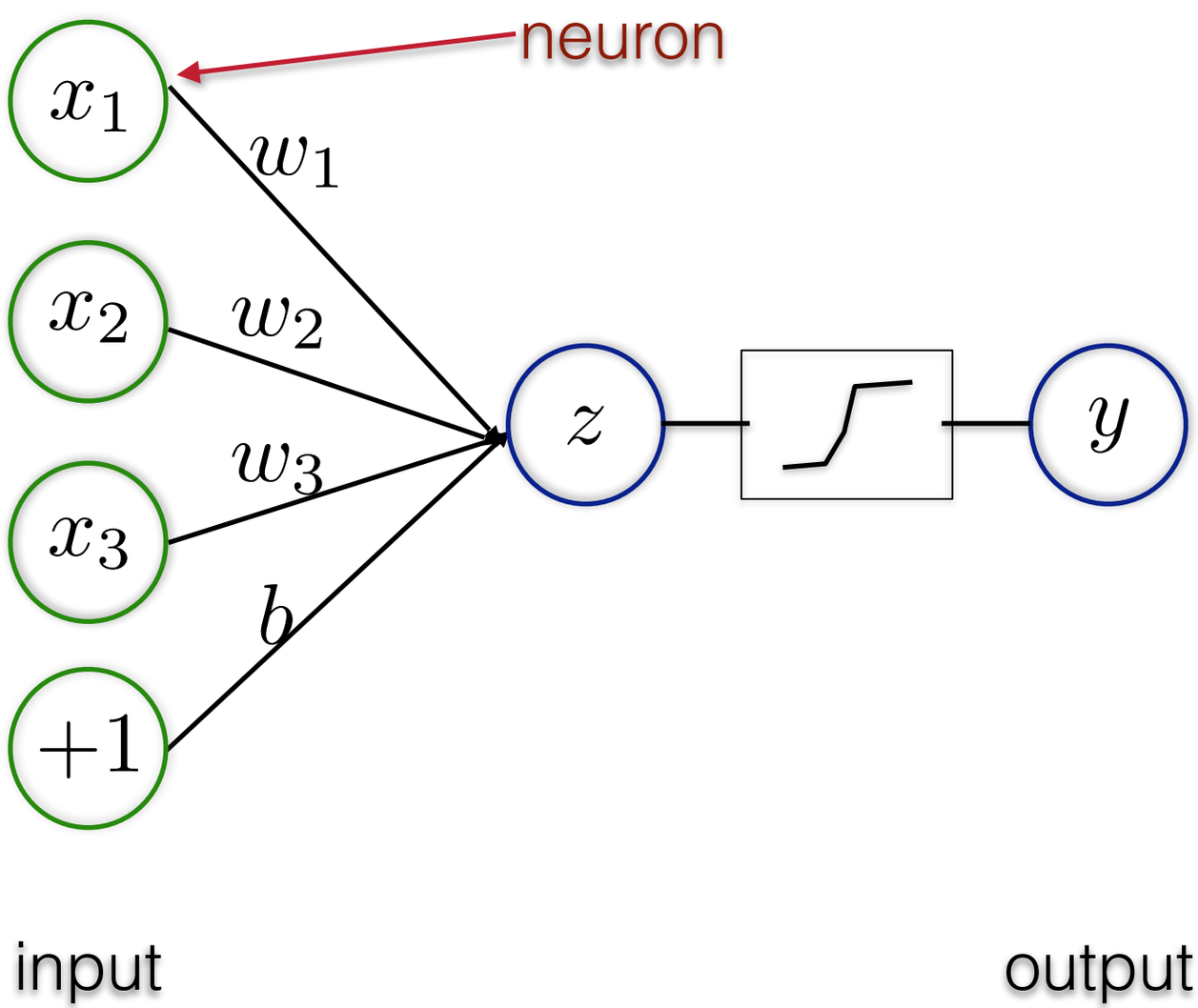


Google
CAT

Facebook established
its deep learning lab

Neural Networks

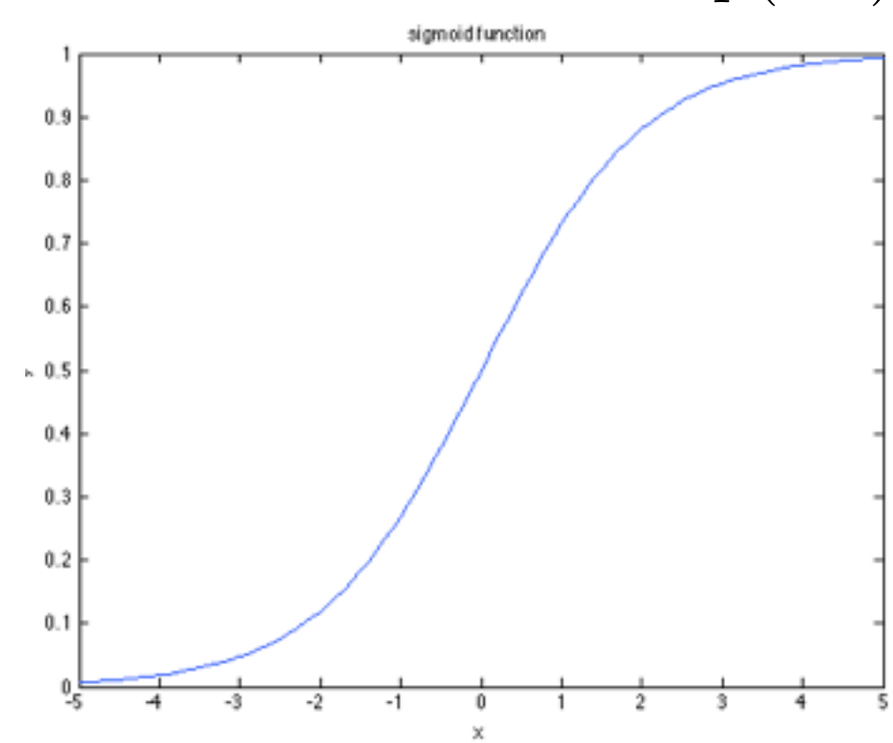
basic building blocks



$$z = \sum_i x_i w_i + b, \quad y = f(z)$$

where f is a activation function:

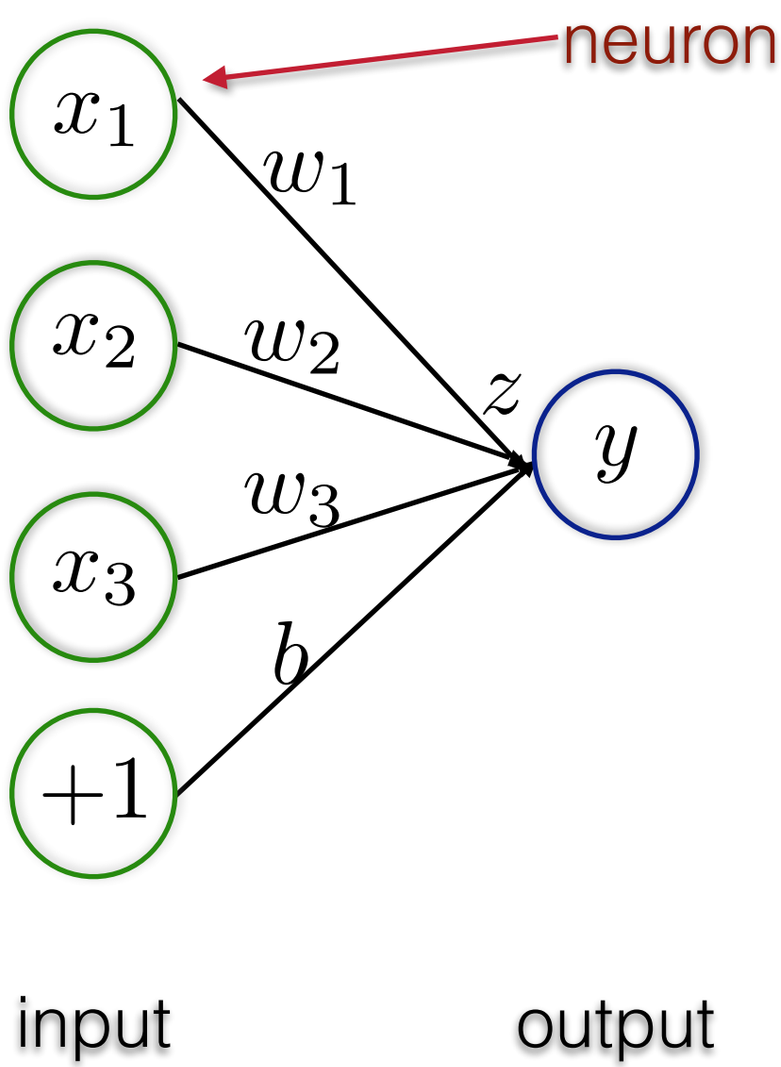
$$f(z) = \sigma(z) = \frac{1}{1 + exp(-z)}$$



- sigmoid is bounded between 0 and 1
- monotonically increasing
- differentiation: $\sigma'(z) = \sigma(z) * (1 - \sigma(z))$

Neural Networks

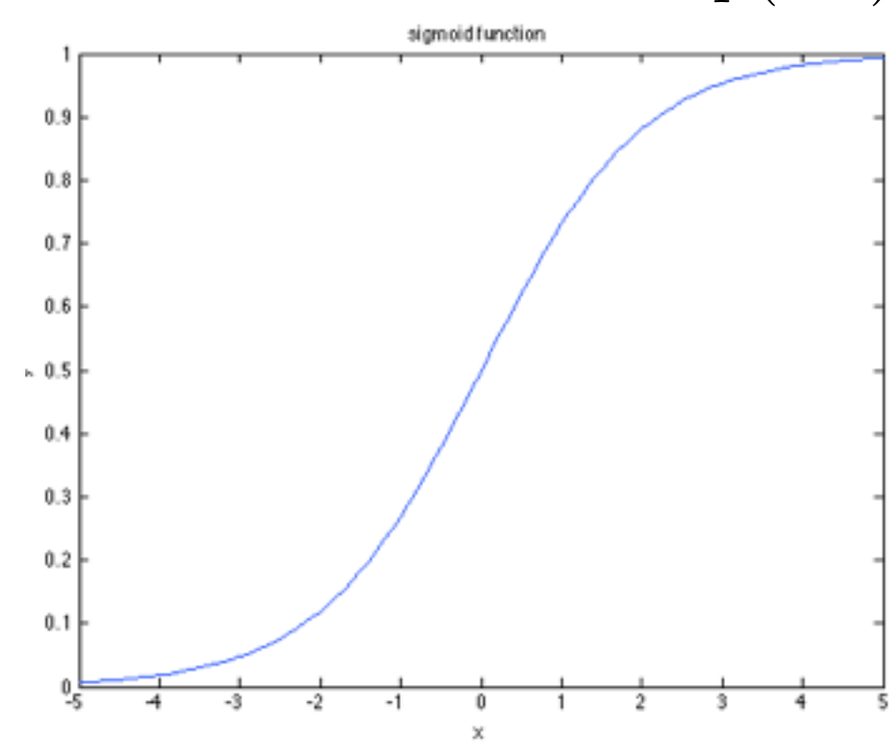
basic building blocks



$$z = \sum_i x_i w_i + b, \quad y = f(z)$$

where f is a activation function:

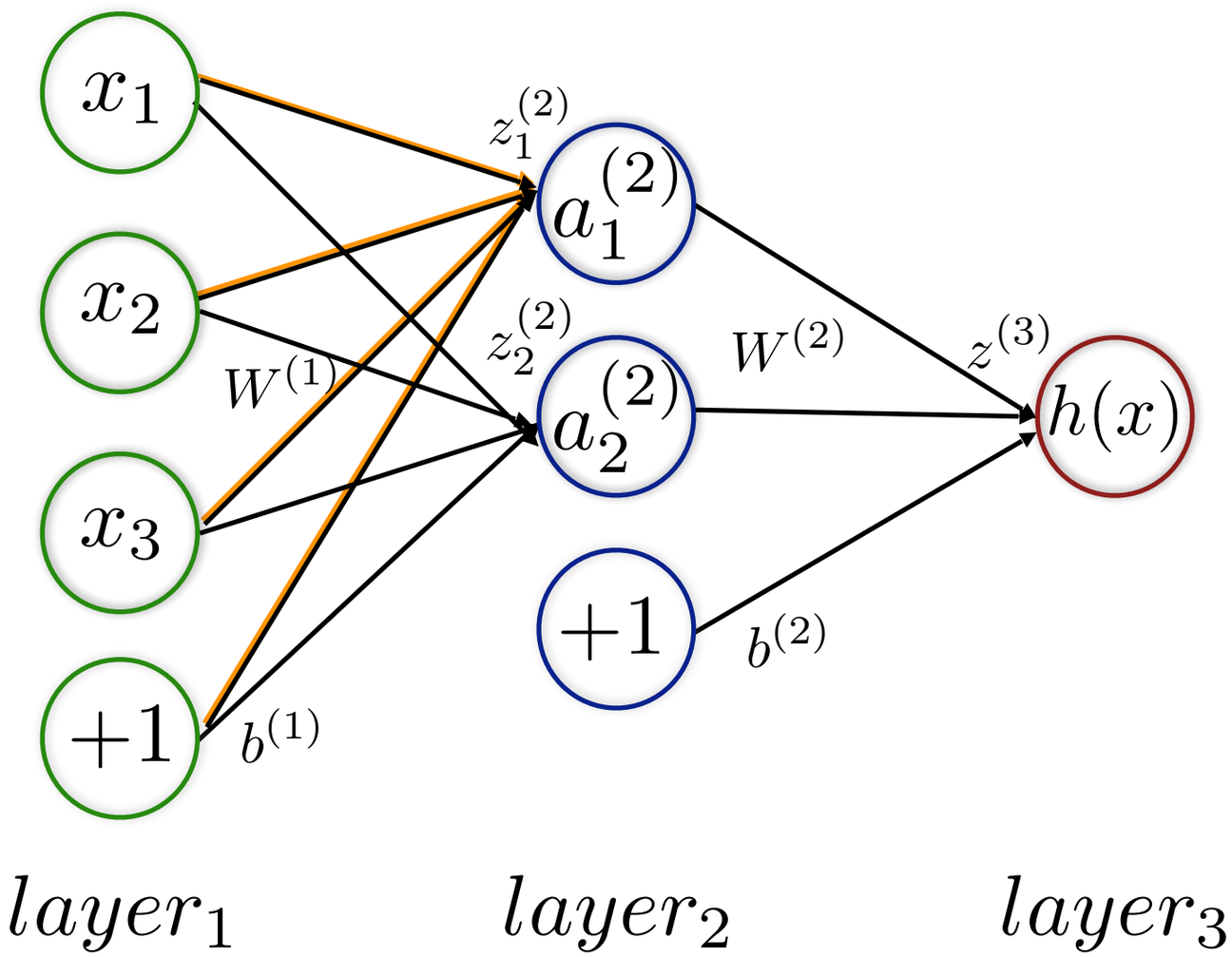
$$f(z) = \sigma(z) = \frac{1}{1 + exp(-z)}$$



- sigmoid is bounded between 0 and 1
- monotonically increasing
- differentiation: $\sigma'(z) = \sigma(z) * (1 - \sigma(z))$

Neural Networks

representation



feed-forward

for each neuron in the next layer:

$$z_i^{(2)} = \sum_{j=1}^{n} w_{ij}^{(1)} x_j + b_i^{(1)}, \quad a_i^{(2)} = f(z_i^{(2)})$$

$$z_1^{(2)} = w_{11}^{(1)} x_1 + w_{21}^{(1)} x_2 + w_{31}^{(1)} x_3 + b_1^{(1)}, \quad a_1^{(2)} = f(z_1^{(2)})$$

compactly:

$$z^{(2)} = W^{(1)} x + b^{(1)}, \quad a^{(2)} = f(z^{(2)})$$
$$z^{(3)} = W^{(2)} x + b^{(2)}, \quad a^{(3)} = f(z^{(3)})$$

globally models a function:

$$\hat{y} = h_{W,b}(x)$$

where W and b are model parameters

Neural Networks

recall: gradient descent

$$min: \quad F(x)$$

if the multivariable function $F(x)$ is defined and differentiable in the neighborhood of a point α, then $F(x)$ decreases fastest if one goes from α in the direction of the negative gradient $-\nabla F(\alpha)$

gradient descent ensures that for sufficiently small γ

$$b = a - \gamma \nabla F(\alpha)$$

we have $F(b) < F(a)$

Neural Networks

supervised learning

- define a loss function over the true label and the model prediction, such as squared error / cross entropy.
- use gradient descent to optimize the model parameter.

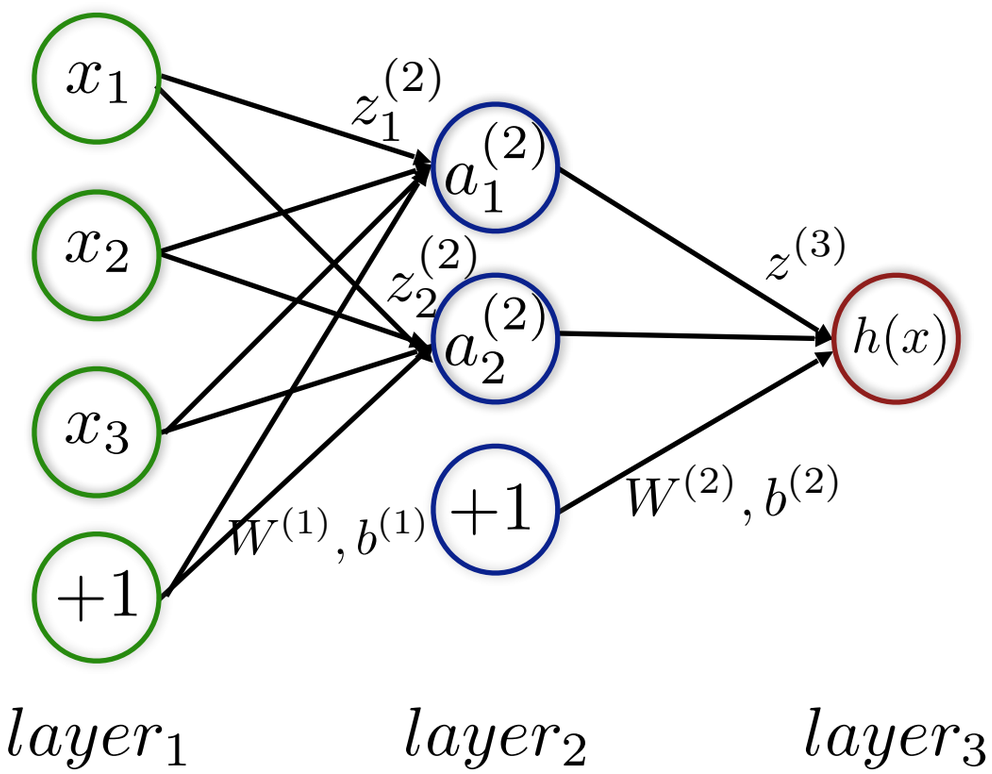
$$w_{ij}^{(l)} := w_{ij}^{(l)} - \alpha \boxed{\frac{\partial}{\partial w_{ij}^{(l)}} J(W, b; x, y)}$$

$$b_i^{(l)} := b_i^{(l)} - \alpha \boxed{\frac{\partial}{\partial b_i^{(l)}} J(W, b; x, y)}$$

back propagation

learning rate

Neural Networks



back propagation

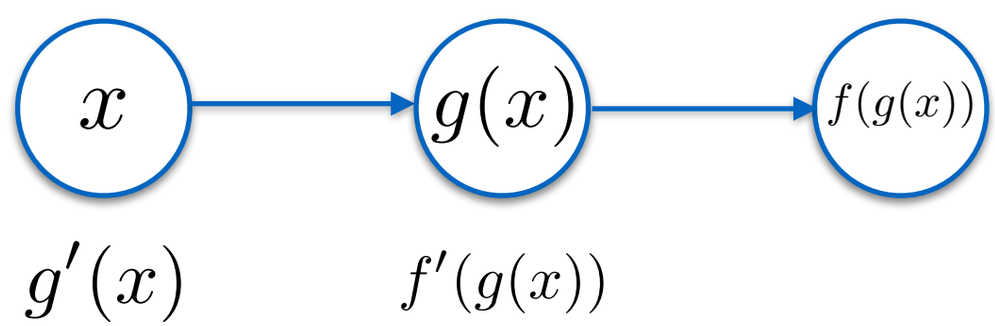
is just the chain rule to derive gradients.

minimize $$J(W,b) = \frac{1}{m}\sum_{i=1}^{m}\frac{1}{2}||h_{W,b}(x^{(i)}) - y^{(i)}||^2$$

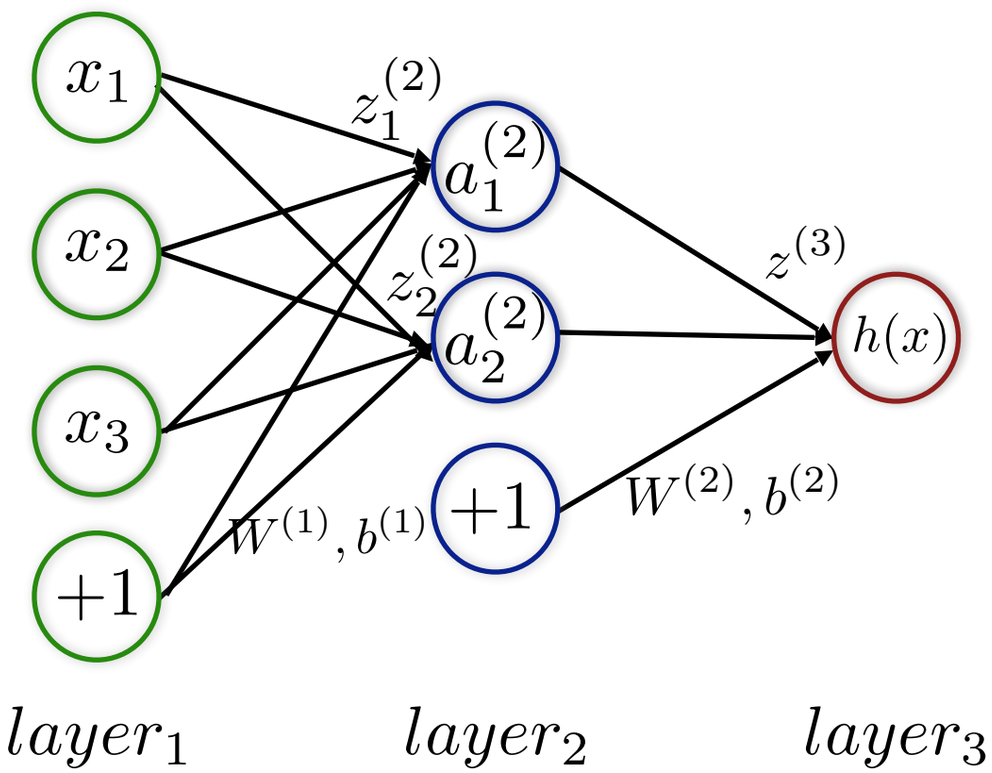
just one sample for illustration: $$J(W,b) = \frac{1}{2}||h(x) - y||^2$$

chain rule: $$y = f(g(x)), \quad y' = f'(g(x)) * g'(x)$$

intuition:



Neural Networks



$$h(x) : \frac{\partial J}{\partial h(x)} = h(x) - y$$

$$z^{(3)} : h(x) = f(z^{(3)})$$

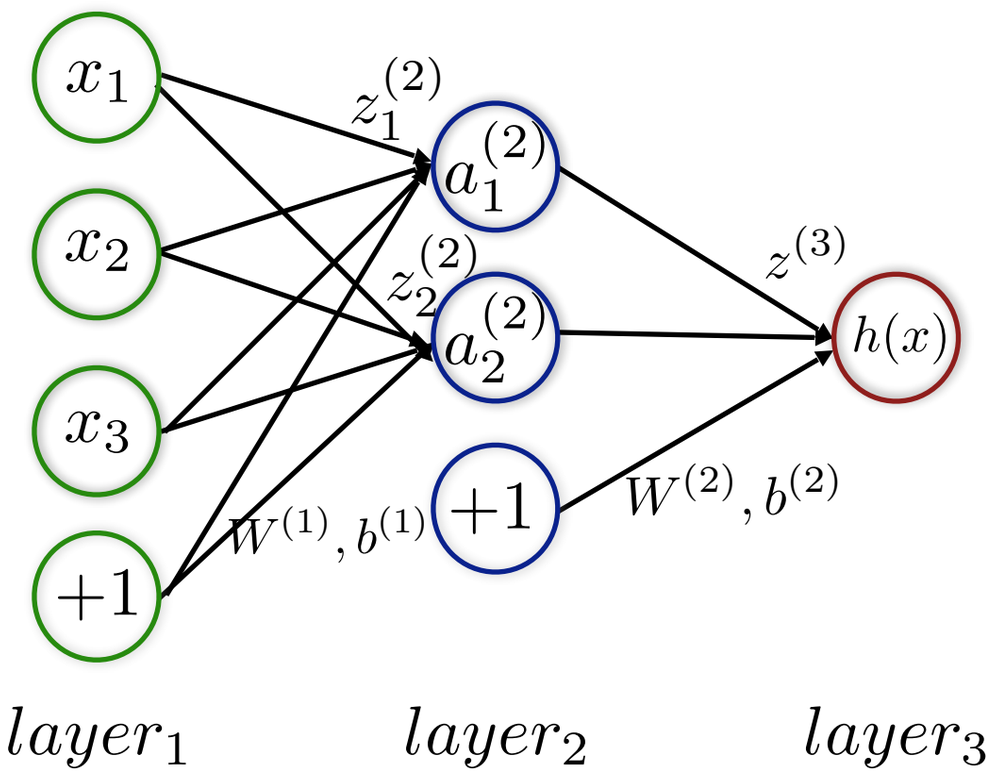
$$\frac{\partial J}{\partial z^{(3)}} = \frac{\partial J}{\partial h(x)} \cdot f'(z^{(3)})$$

$$W^{(2)}, b^{(2)} : \quad z^{(3)} = W^{(2)} a^{(2)} + b^{(2)}$$

$$\frac{\partial J}{\partial w_j^{(2)}} = \frac{\partial J}{\partial z^{(3)}} \cdot a_j^{(2)}$$

$$\frac{\partial J}{\partial b^{(2)}} = \frac{\partial J}{\partial z^{(3)}} \cdot 1$$

Neural Networks



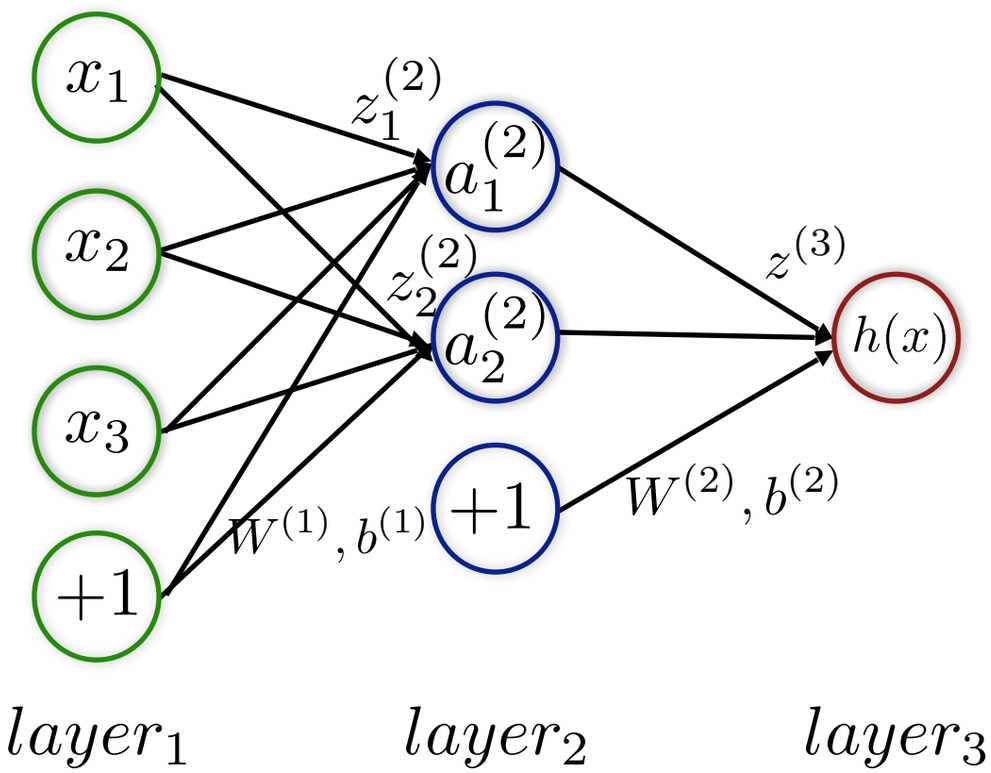
$$a_j^{(2)} : \quad z^{(3)} = W^{(2)} a^{(2)} + b^{(2)}$$

$$\frac{\partial J}{\partial a_j^{(2)}} = \frac{\partial J}{\partial z^{(3)}} \cdot w_j^{(2)}$$

$$z_j^{(2)} : \quad a_j^{(2)} = f(z_j^{(2)})$$

$$\frac{\partial J}{\partial z_j^{(2)}} = \frac{\partial J}{\partial a_j^{(2)}} \cdot f'(z_j^{(2)})$$

Neural Networks



$$W^{(1)}, b^{(1)} : \quad z^{(2)} = W^{(1)}x + b^{(1)}$$

$$\frac{\partial J}{\partial w_{ij}^{(1)}} = \frac{\partial J}{\partial z_i^{(2)}} \cdot x_j$$

$$\frac{\partial J}{\partial b_i^{(1)}} = \frac{\partial J}{\partial z_i^{(2)}} \cdot 1$$

Neural Networks

learning algorithm for neural nets

initialize: $W = 0.01 * N(0,1); b = 0$

for iter = 1 : epochs

feed_forward(); // computing activations for all layers

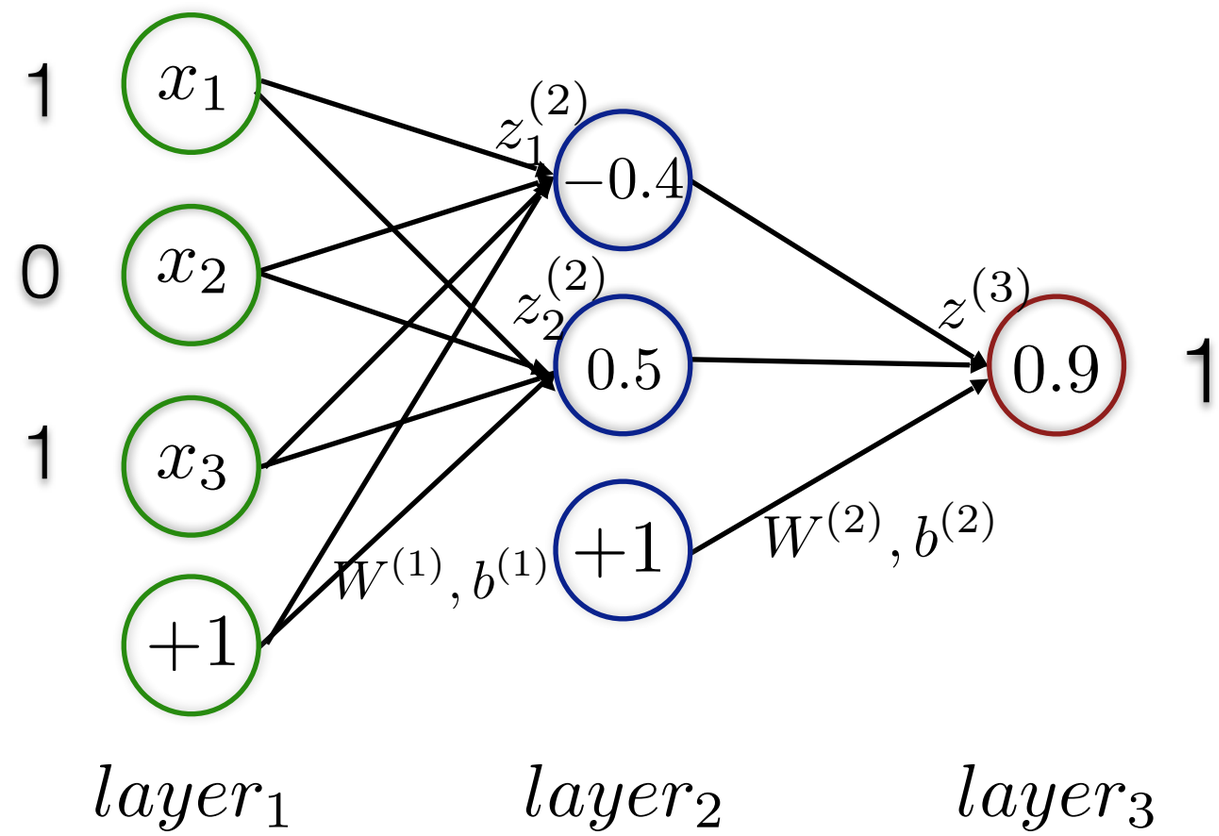
back_prop(); // compute the partial derivatives respect to W, b

$W := W - \alpha \cdot \Delta W; b := b - \alpha \cdot \Delta b$ // update

end

Neural Networks

toy example: (linear identity activation function)

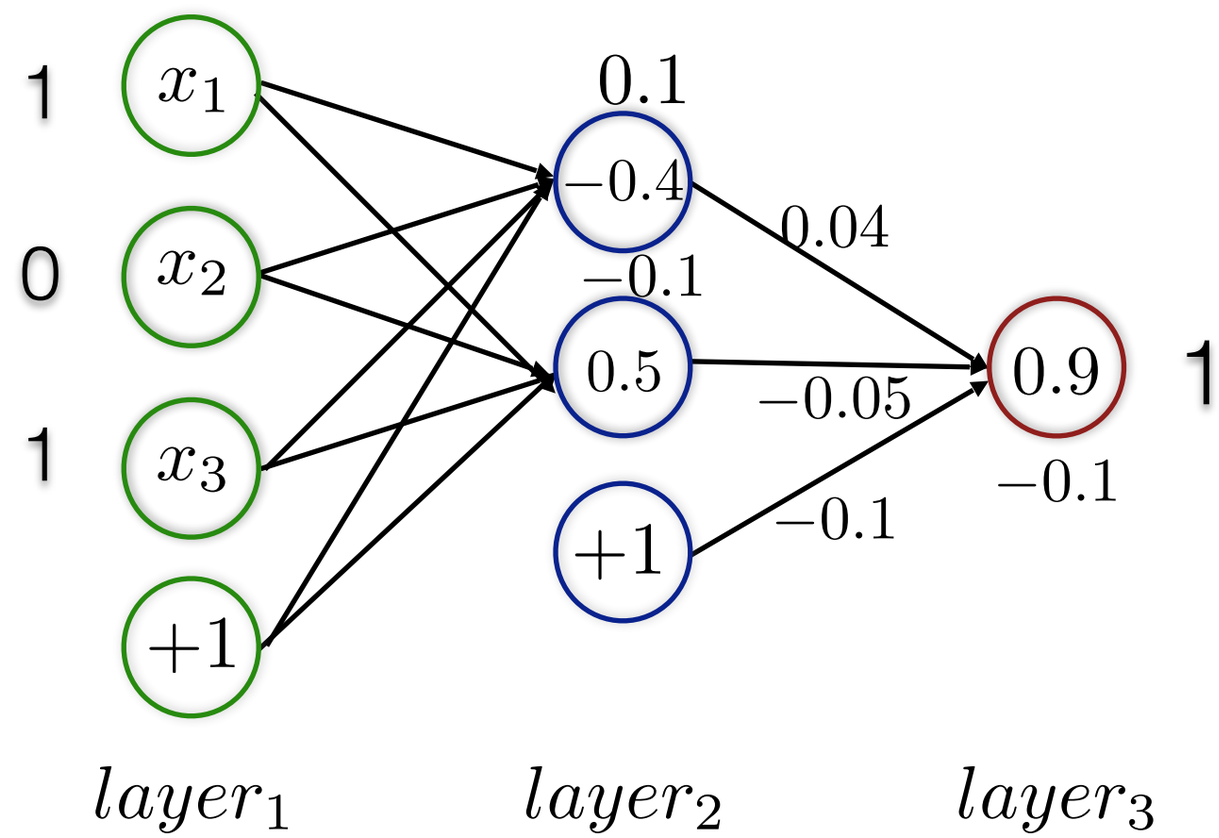


$$W^{(1)} = \begin{bmatrix} 0.1 & 0.3 & 0 \\ -0.2 & 0.5 & 0.2 \end{bmatrix} \qquad b^{(1)} = \begin{bmatrix} -0.5 \\ 0.5 \end{bmatrix}$$

$$W^{(2)} = [-1, 1] \qquad b^{(2)} = [0]$$

Neural Networks

toy example: (linear identity activation function)



$$W^{(1)} = \begin{bmatrix} 0.1 & 0.3 & 0 \\ -0.2 & 0.5 & 0.2 \end{bmatrix} \qquad b^{(1)} = \begin{bmatrix} -0.5 \\ 0.5 \end{bmatrix}$$

$$W^{(2)} = [-1, 1] \qquad b^{(2)} = [0]$$

Neural Networks

batch gradient descent & stochastic gradient descent

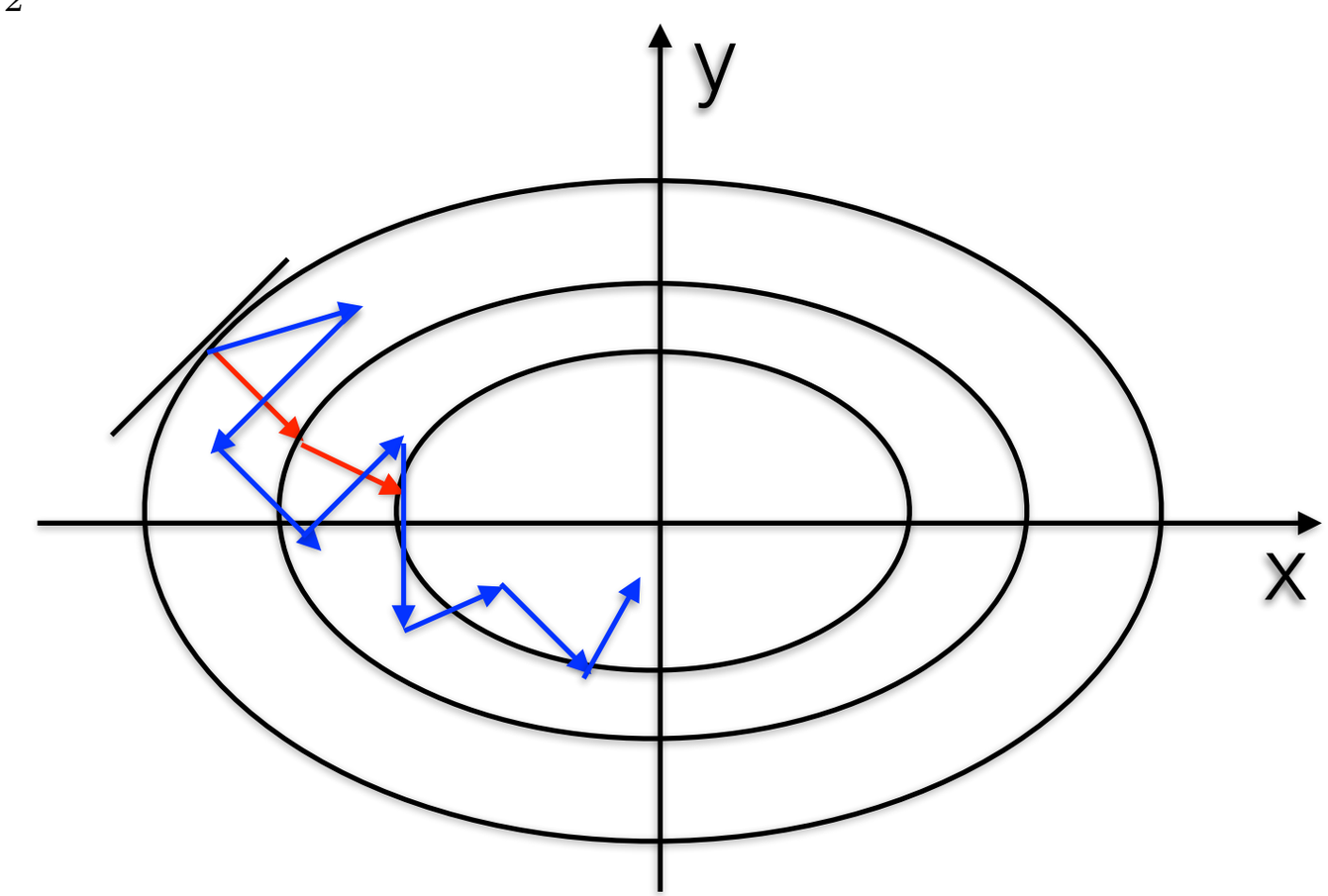
use a mini-batch instead of all samples
to evaluate one gradient step
e.x. 100 training data, 5 mini-batch, each 20 data

recall

minimize $J(W,b) = \frac{1}{m}\sum_{i=1}^{m}\frac{1}{2}||h_{W,b}(x^{(i)}) - y^{(i)}||^2$

we have to compute and average all the training sample gradients in order to perform one gradient descent step

too costly!
each sample has already provide some information where to go



shuffle the data

Neural Networks

stochastic gradient descent algorithm

```
initialize: W = 0.01 * N(0,1); b = 0
for iter = 1 : epochs
    shuffle the data
    for batch = 1 : num_batch
        feed_forward();  // computing activations for all layers
        back_prop(); // compute the partial derivatives respect to W, b
        W := W - α · ΔW; b := b - α · Δb        // update
    end
end
```

Neural Networks

stochastic gradient descent practical issues:

1. how to initialize W,b?

   small values. e.x. W to be 0.01*N(0,1), b to be 0

2. how to set the learning rate?

   set the learning rate so that:

$$\alpha * \frac{\partial}{\partial w} \approx 10^{-3} \cdot w$$

   linearly decreasing when necessary

3. how to set the size of mini-batch?

   cannot be too big, 20 - 50 is fine

4. problem: gradient diffusion.

   choose larger learning rate for the front layers.

Neural Networks

more sophisticated way to update parameters

$$\theta := \theta - \alpha \cdot \Delta\theta$$

$\lambda \approx 10^{-4} - 10^{-6}$

weight decay: $$\hat{J}(\theta) = J(\theta) + \frac{\lambda}{2}||\theta||^2 = \frac{1}{2}||h(x) - y||^2 + \frac{\lambda}{2}||\theta||^2$$

$$\Delta\theta = \frac{\partial(J)}{\partial\theta} + \lambda\theta$$

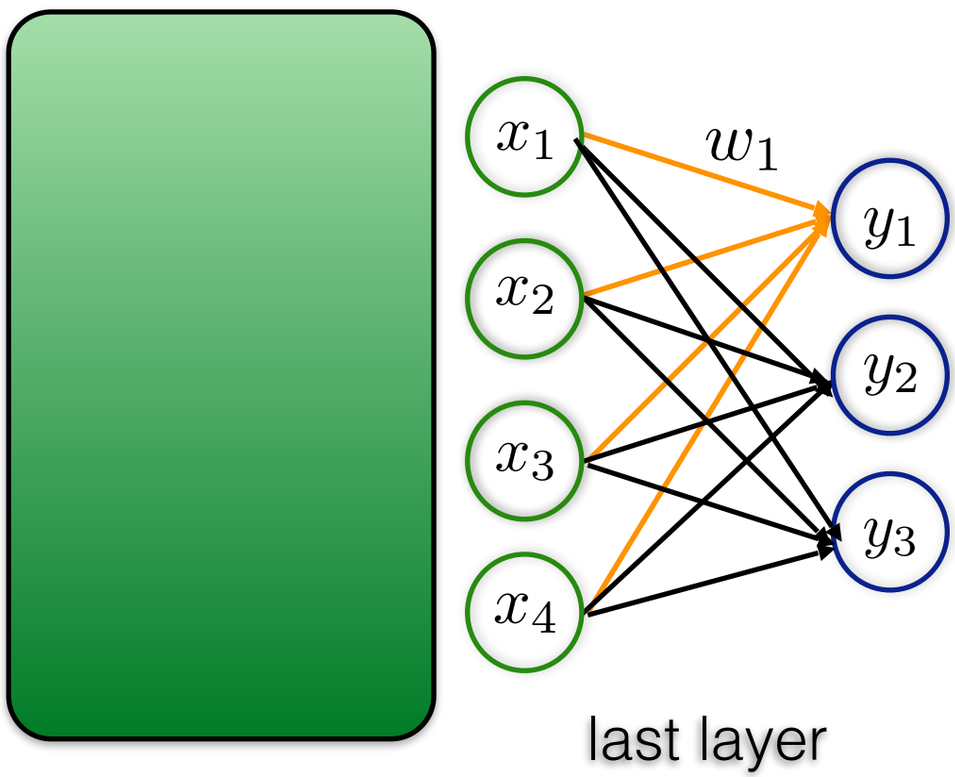
$m \approx 0.9$

momentum: $$\Delta\theta_{new} = m \cdot \Delta\theta_{old} + (1 - m) \cdot \Delta\theta$$

brings robustness to stochastic gradient descent

Neural Networks

softmax classifier



label is in 1-k coding scheme

$$\hat{y}_i = \frac{exp(w_i x)}{\sum_{j=1}^{k} exp(w_j x)} = p(y_i = 1)$$

cost function:

$$J(W) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} log(\hat{y}^{(i)}) + (1 - y^{(i)}) log(1 - \hat{y}^{(i)}))\right]$$

cross entropy:
maximum likelihood of bernouli distribution

derivatives: (back prop)

$$\frac{\partial}{\partial w_j} J(W) = -\frac{1}{m}\sum_{i=1}^{m}\left[x^{(i)}(y_j^{(i)} - \hat{y}_j^{(i)})\right]$$

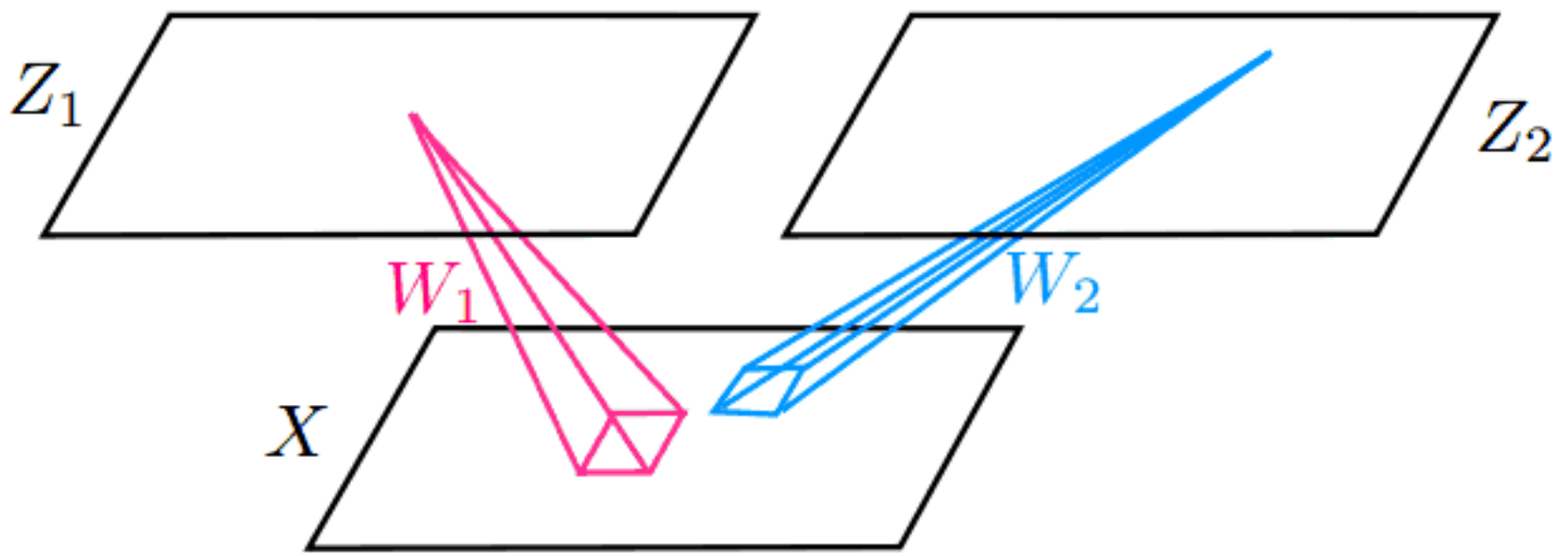
Convolutional Neural Networks

why convolutional?

- for images, too many model parameters
- locality and stationarity of features

Convolutional Neural Networks

detection layers



convolution

$$Z_i = \sigma(W_i * X)$$

Convolutional Neural Networks

pooling :

- reduce the size of representations
- allow small translation invariance.

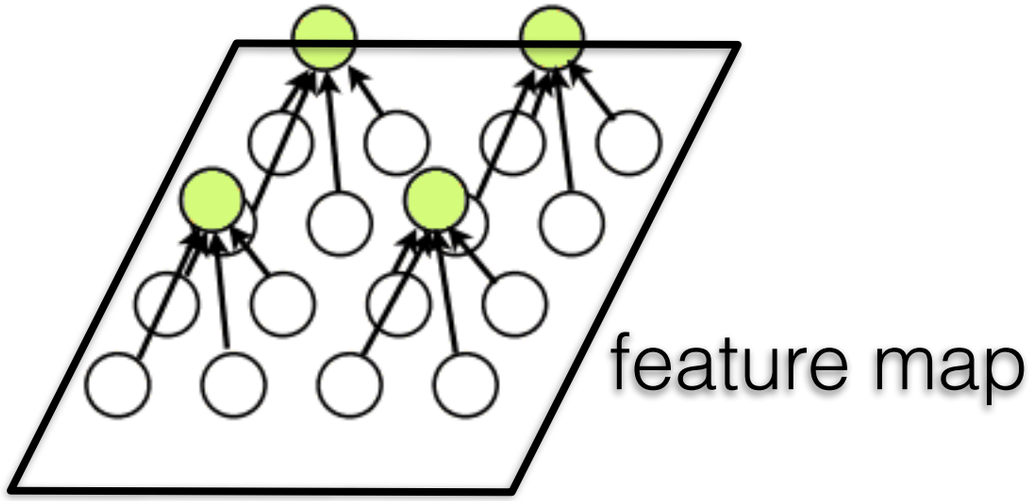
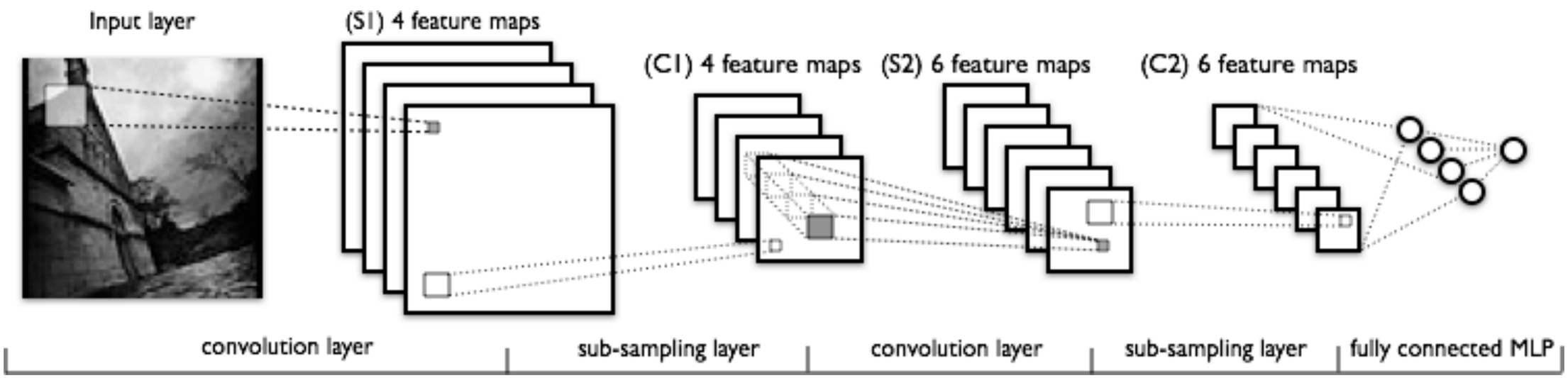


figure from roger grosse tutorial

common pooling techniques:
max pooling, average pooling

Convolutional Neural Networks

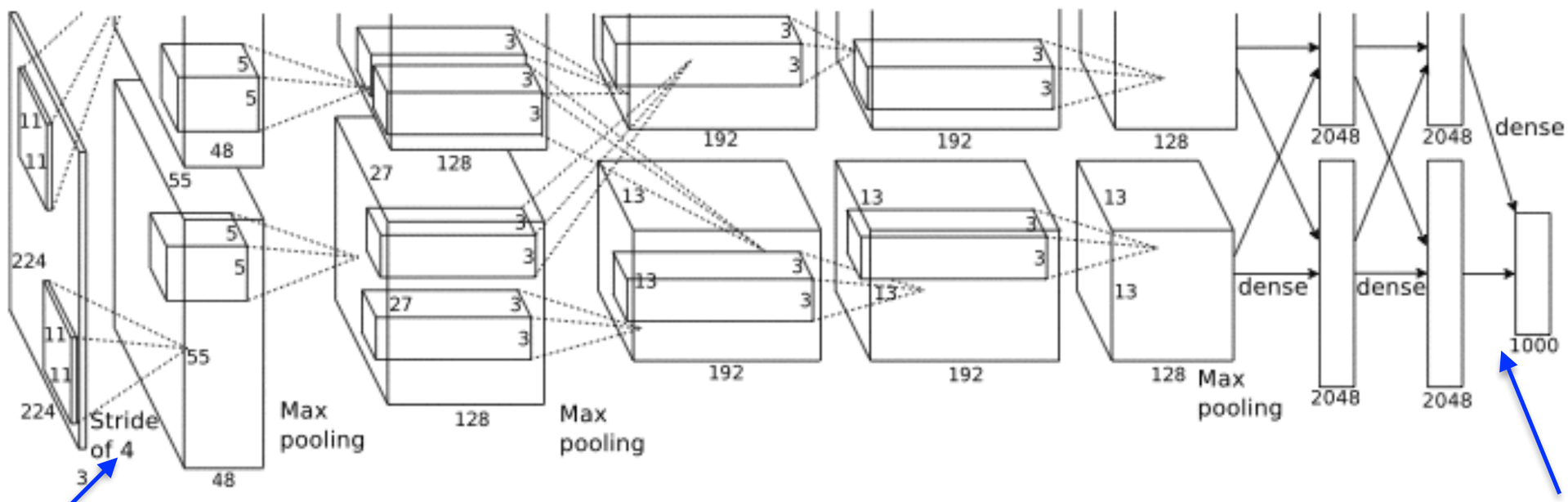
convolution network is just a combination of convolution layers, pooling layers and fully connected layers



(figure from LeNet tutorial, http://deeplearning.net/tutorial/lenet.html)

Convolutional Neural Networks

practical example: ImageNet Classification



stride

softmax
classifier

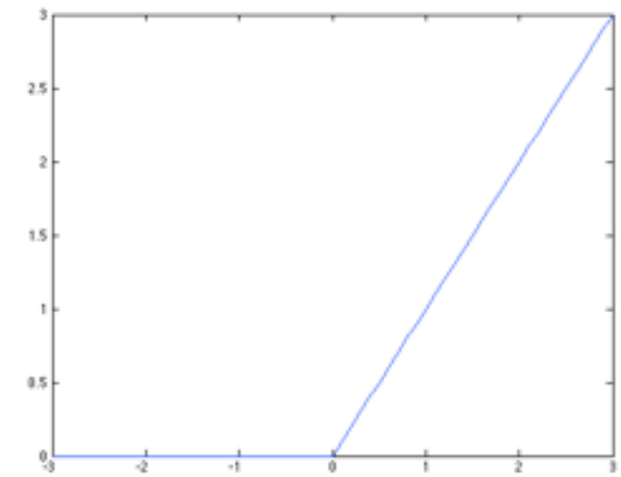
architecture:
8 layers in total
first 5 are convolutional
last 3 are fully connected
max-pooling in the 1,2,5th layer

(figure from Krizhevsky, Alex et al 2012)

Convolutional Neural Networks

details:

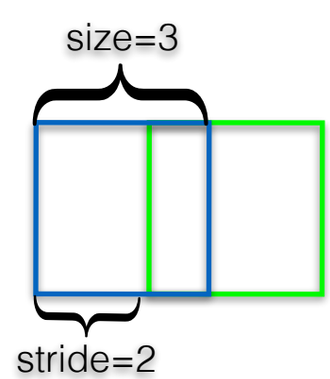
activation function : ReLU $f(z) = max(0, z)$



local normal contrast: $$b^i_{x,y} = a^i_{x,y} / \left(k + \alpha \sum_{j=\max(0,i-n/2)}^{\min(N-1,i+n/2)} (a^j_{x,y})^2 \right)^\beta$$

normalize at the same spatial location
over different feature maps

overlapping pooling: pooling size 3 with stride 2



Convolutional Neural Networks

reduce over-fitting:

training 1.2 million images for 60 million parameters

data augmentation:

- extracting random 224*224 patches from the 256*256 images and horizontal refections. This results in an increase by a factor of 2048.

- altering the intensities of the RGB channels, object identity is invariant to the illumination. Add each RGB pixel in each training image by

$$[\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3][\alpha_1\lambda_1, \alpha_2\lambda_2, \alpha_3\lambda_3]^T$$

where p_i are the 3 - 1 eigenvectors and λ_i are eigen values of 3 - 3 covariance matrix of RGB pixels, α_i are random numbers

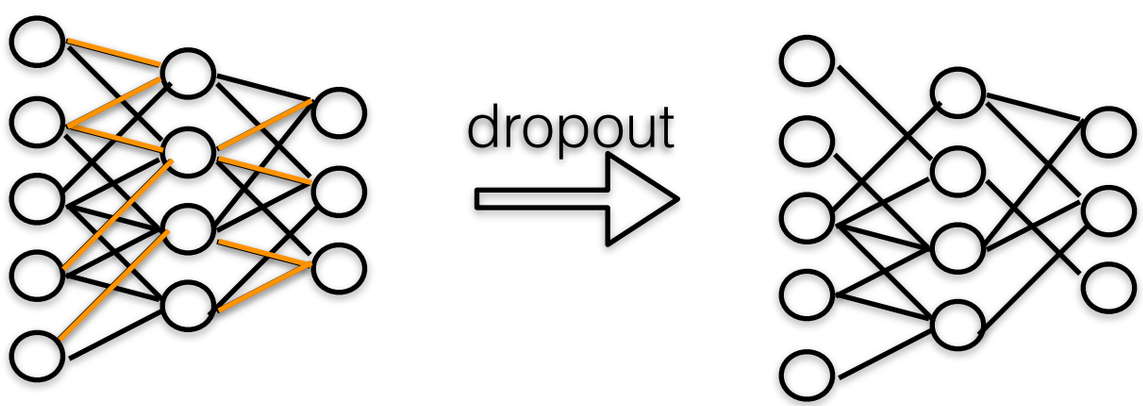
Convolutional Neural Networks

reduce over-fitting:

dropout:

an efficient way to combining different model predictions.

- for each forward pass and each back propagation, randomly defunctions every neuron with probability 0.5. (setting the activation to zero)
- testing: half all the weights, and use all of the neurons to predict



Convolutional Neural Networks

practical implementation tips:
(do not contain activation functions)

feed forward:

convolution:

```
output = conv2(input,  rot90(kernel), 'valid');
```

max & average pooling:

```
patches = im2col(input,[pool_size, pool_size]);
patches = patches(:,1:pool_size:end);
output = mean(patches); output = max(patches);
output = reshape(output,layer_size);
```

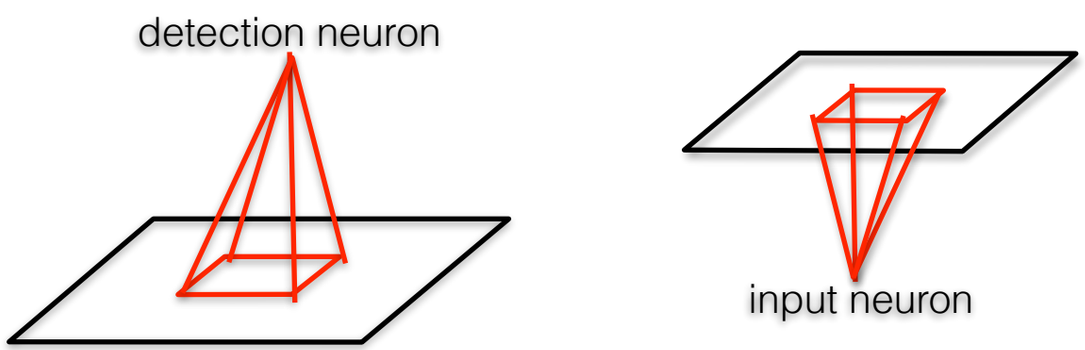
Convolutional Neural Networks

practical implementation tips:
(do not contain activation functions)

back prop:

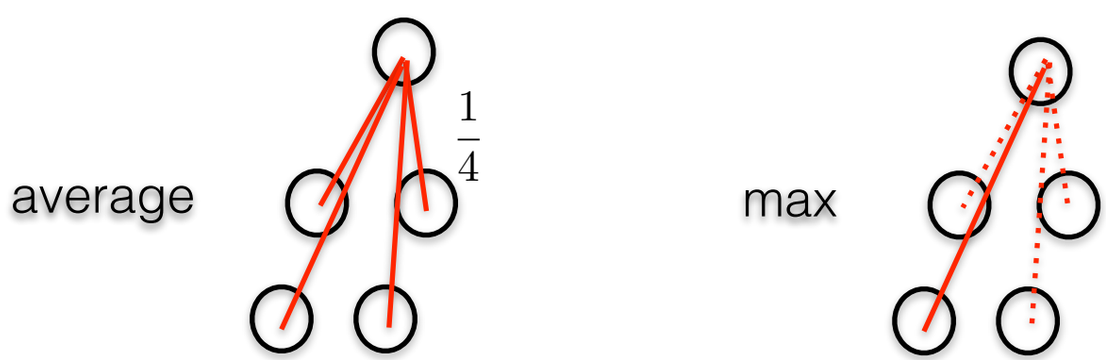
convolution:

```
grd_pre_layer = conv2(grd_latter_layer, kernel, 'full');
grd_w = conv2(pre_layer_activation, rot90(grd_latter_layer), 'valid');
```



max & average pooling:

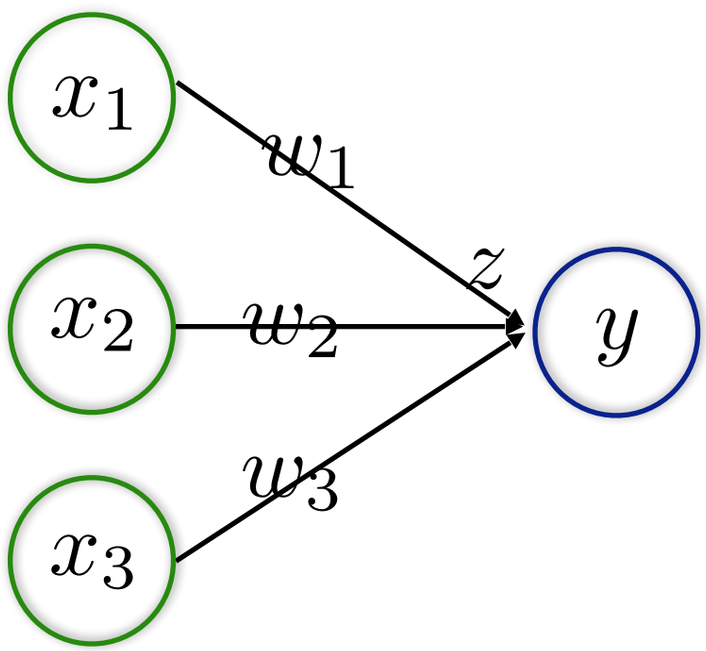
```
grd_unpooled= kron( grd_pooled, ones(scale) ) ./ scale ^ 2);
%max_index
grd_unpooled(max_index) = grd_pooled;
```



Convolutional Neural Networks

filter visualization:

first layer: what pattern is most likely to turn on this neuron?



$$max \quad w \cdot x$$

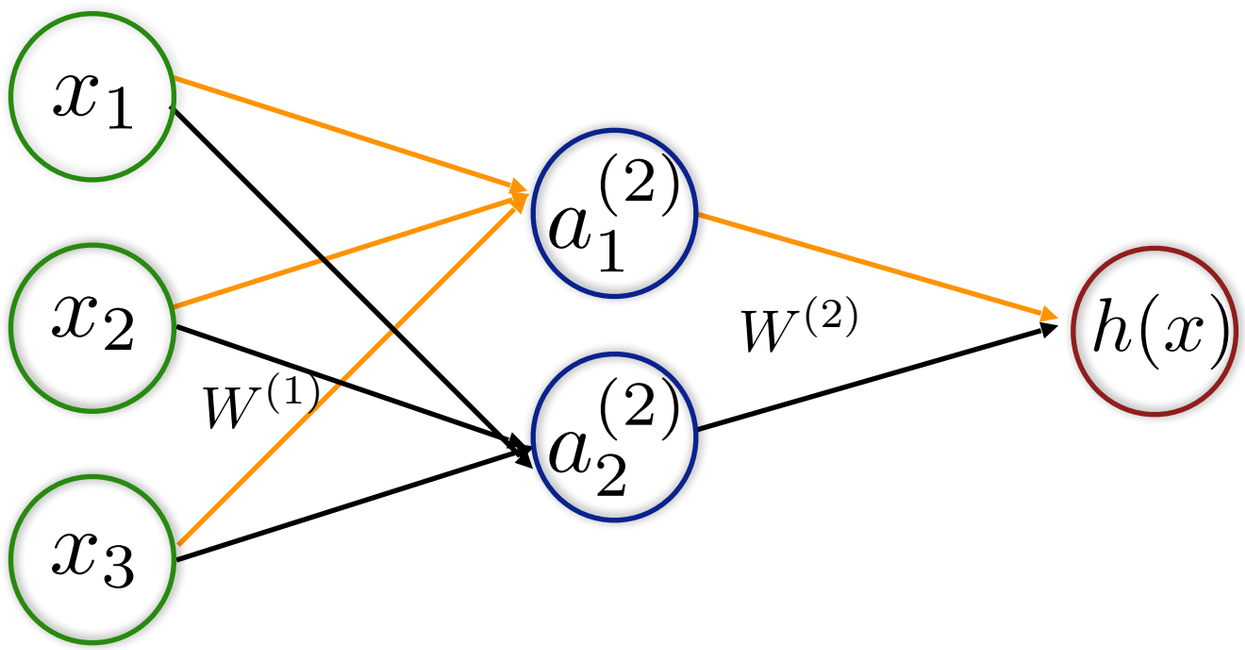
$$s.t. \quad ||x|| = 1$$

$$x = \frac{w}{||w||}$$

Convolutional Neural Networks

filter visualization:

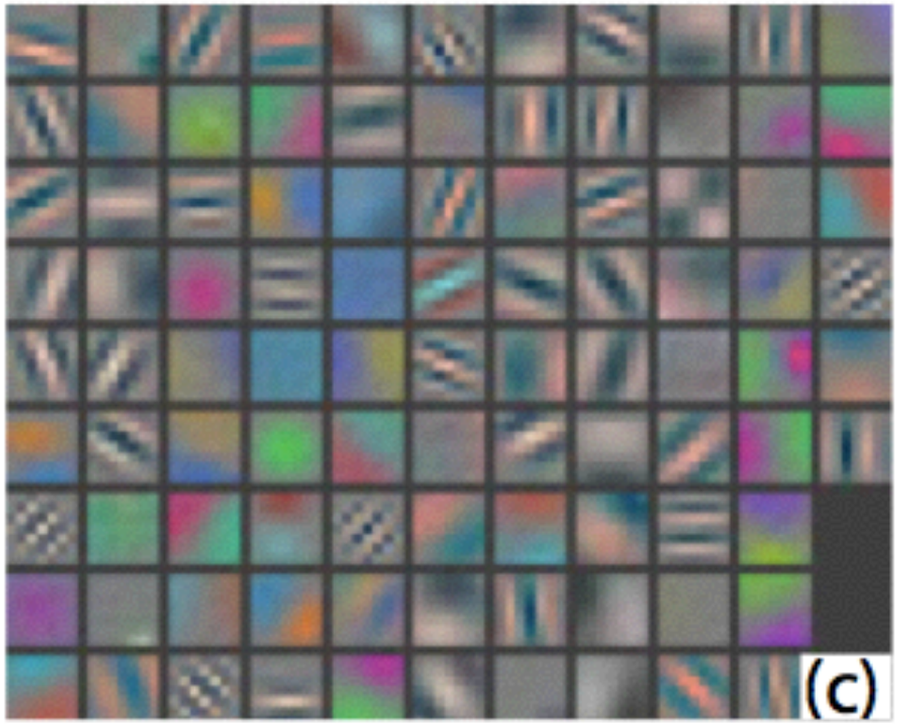
higher layers: difficult, a linear combination of previous layers.

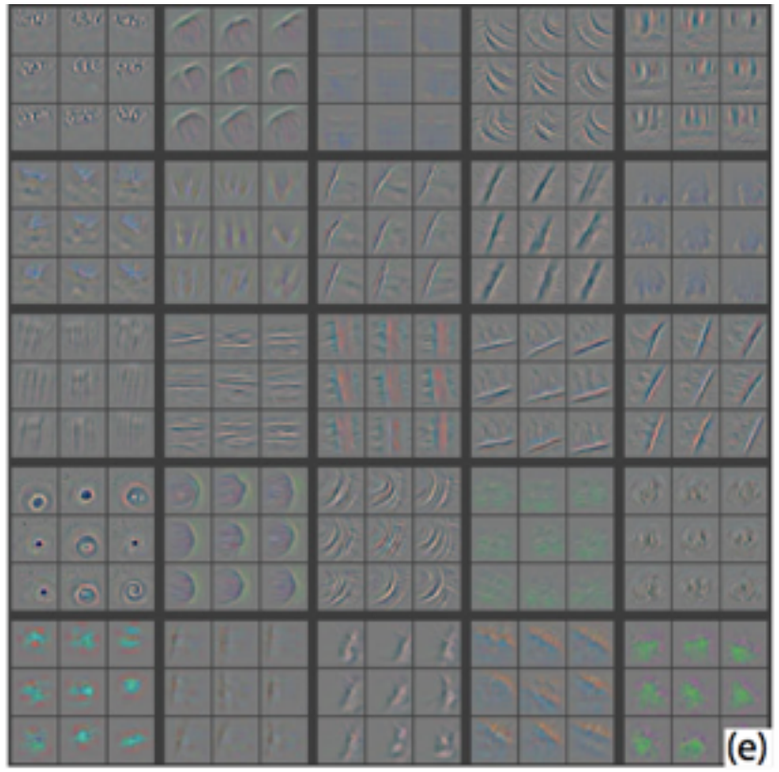


$$f_1 \cdot W_1^{(2)} + f_2 \cdot W_2^{(2)}$$

Convolutional Neural Networks

filter visualization:



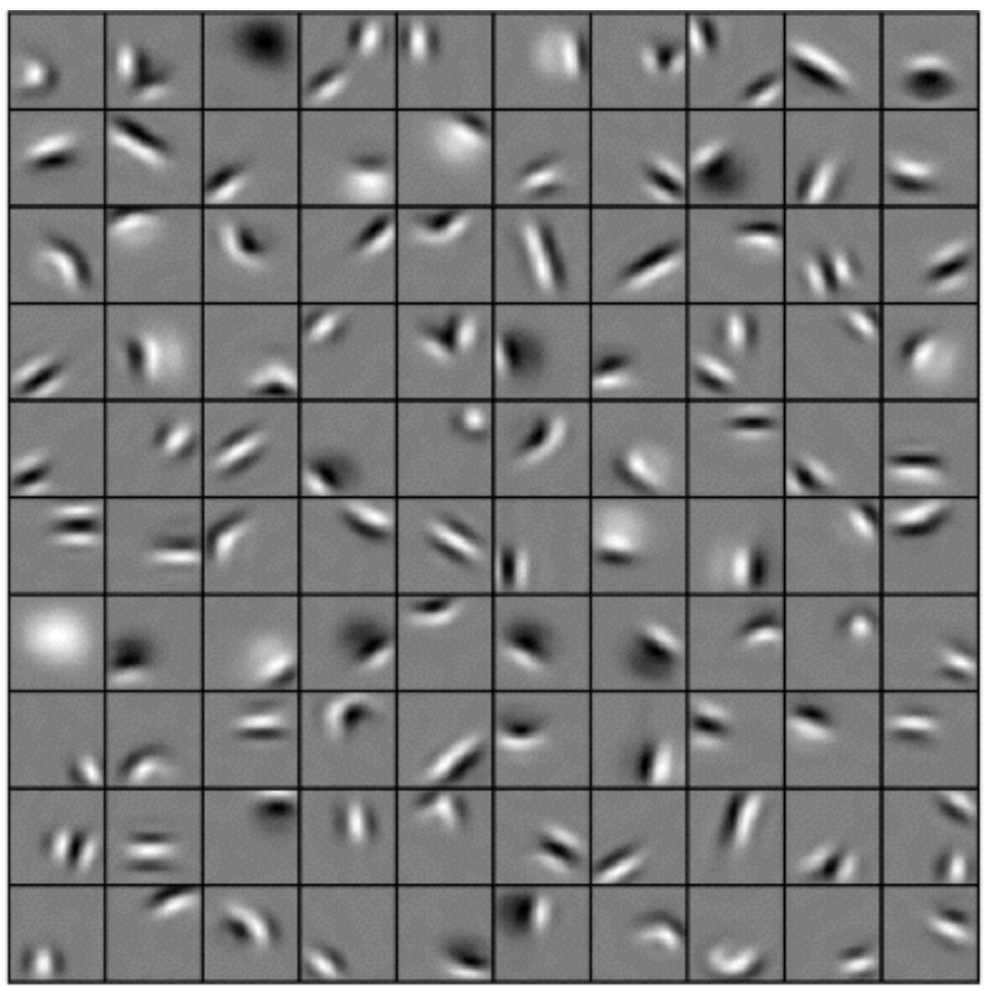


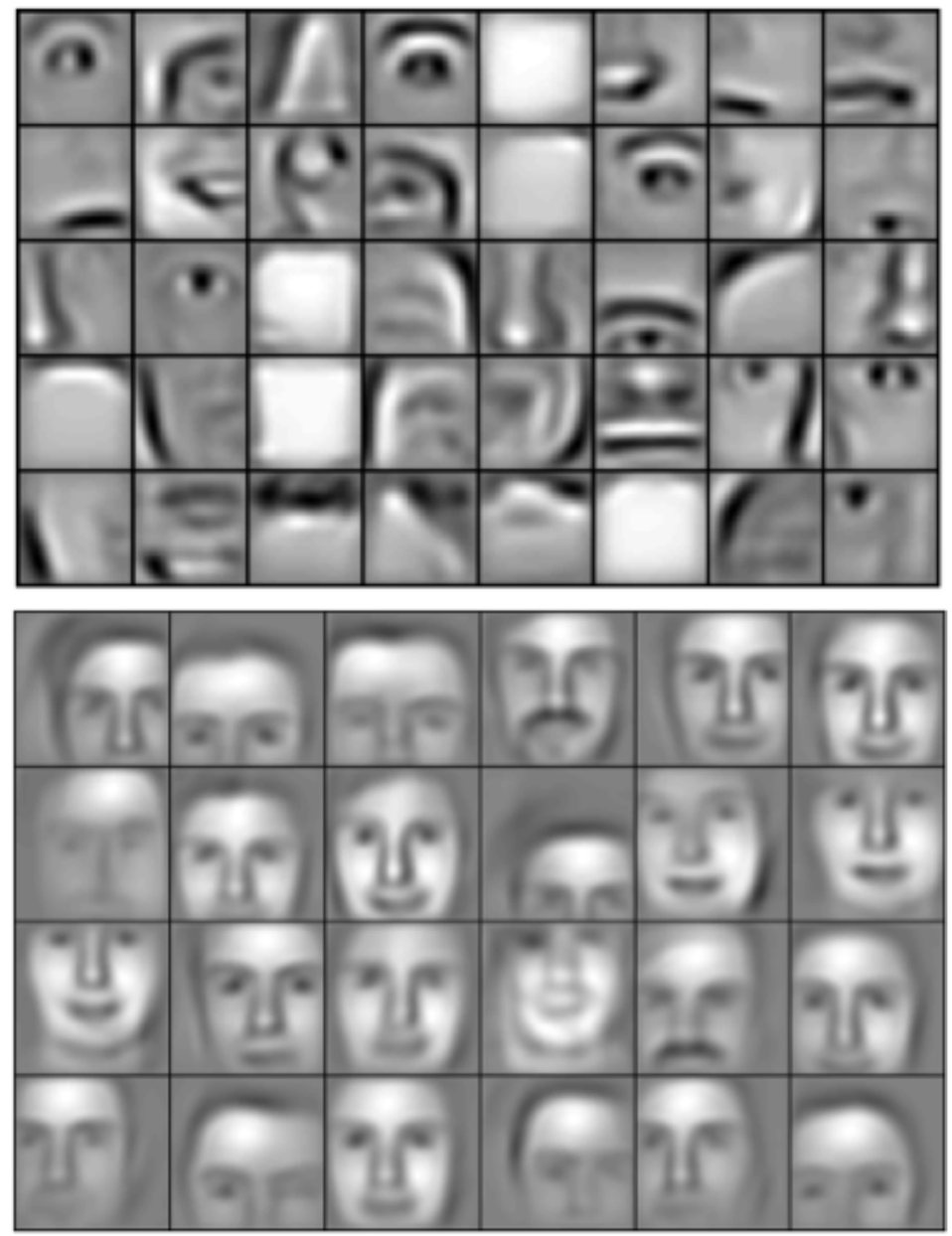
filters learned

(Matthew D. Zeiler et al 2013)

Convolutional Neural Networks

filter visualization:





(Honglak Lee et al 2009)

Convolutional Neural Networks

online web demo:

Convolutional Neural Networks

matlab code demo