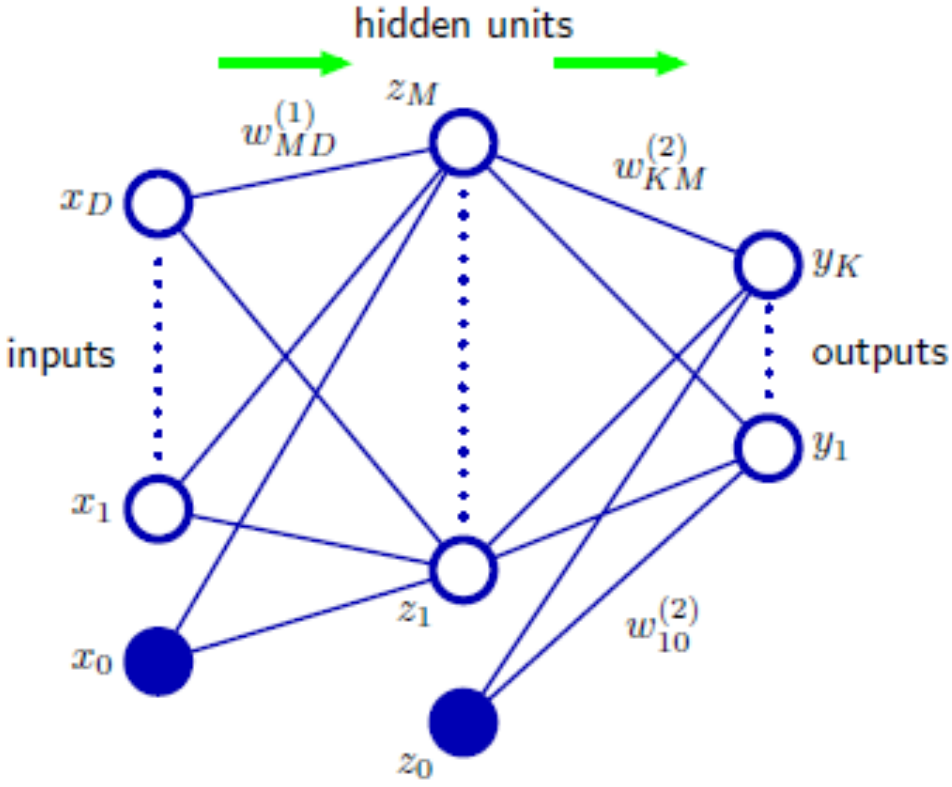
Figure 5.1 Network diagram for the two-layer neural network corresponding to (5.7). The input, hidden, and output variables are represented by nodes, and the weight parameters are represented by links between the nodes, in which the bias parameters are denoted by links coming from additional input and hidden variables x_0 and z_0. Arrows denote the direction of information flow through the network during forward propagation.



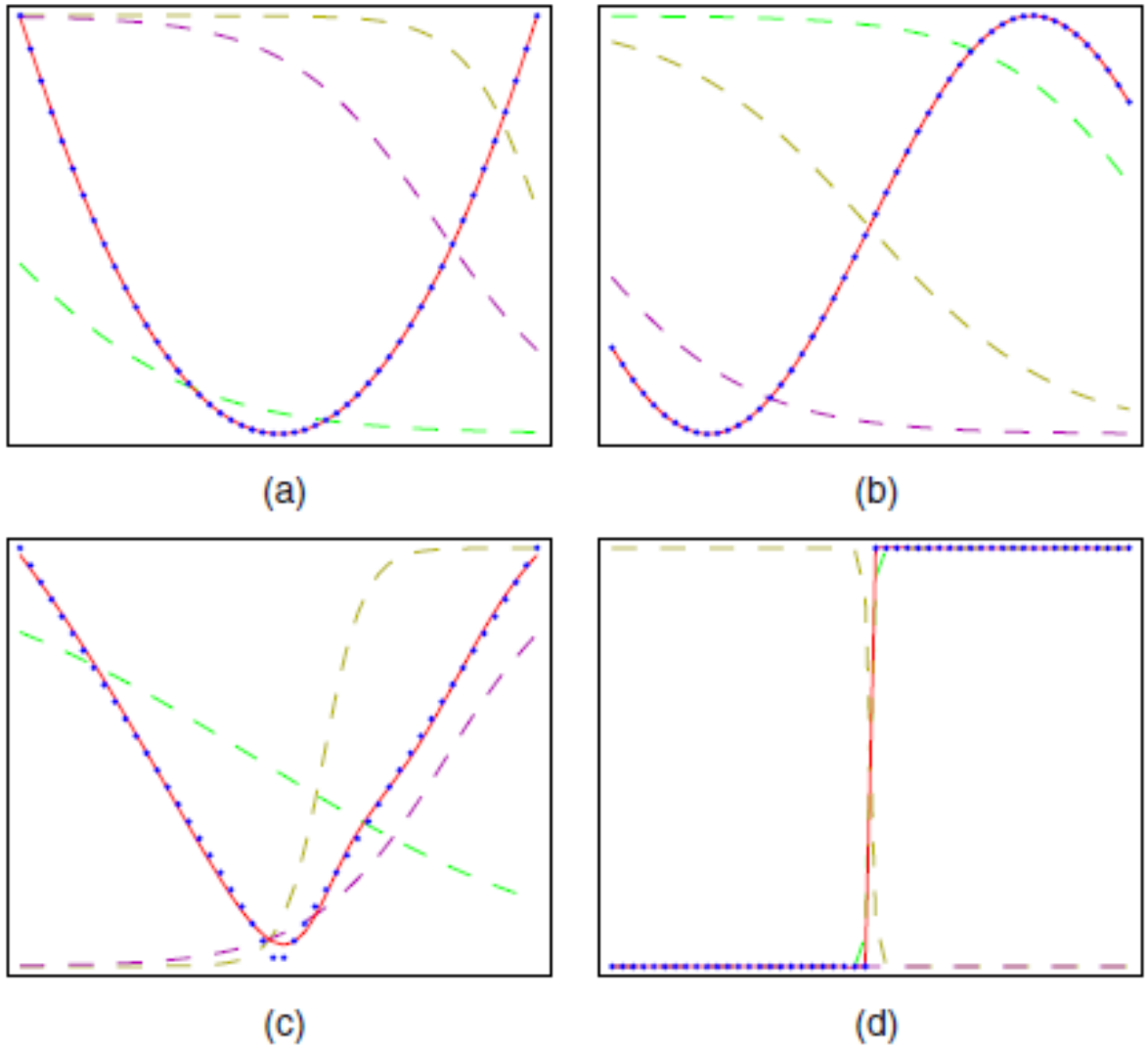
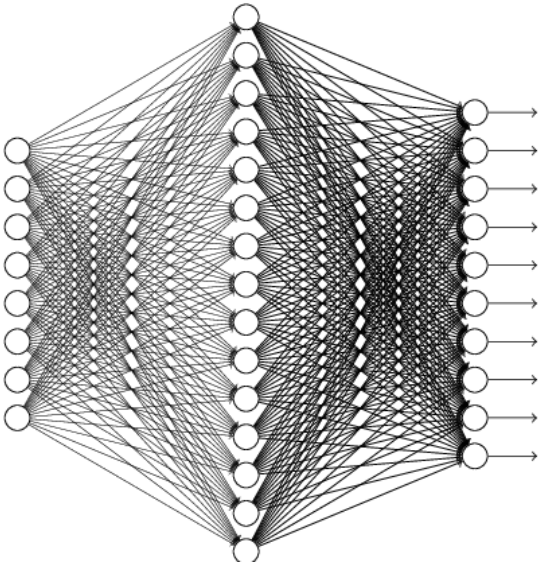
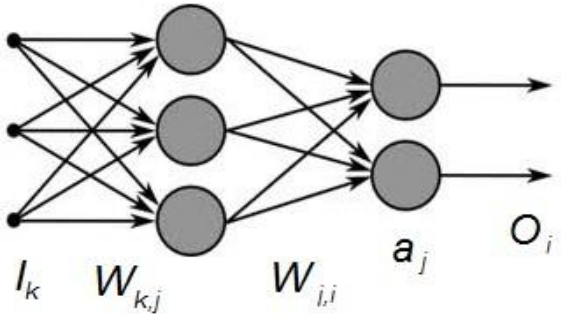
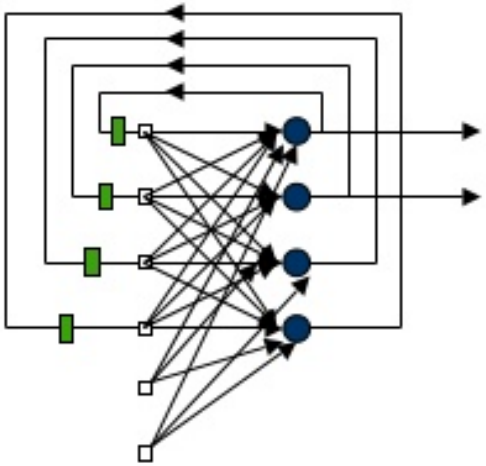


Figure 5.3 Illustration of the capability of a multilayer perceptron to approximate four different functions comprising (a) $f(x) = x^2$, (b) $f(x) = \sin(x)$, (c), $f(x) = |x|$, and (d) $f(x) = H(x)$ where $H(x)$ is the Heaviside step function. In each case, $N = 50$ data points, shown as blue dots, have been sampled uniformly in x over the interval $(-1, 1)$ and the corresponding values of $f(x)$ evaluated. These data points are then used to train a two-layer network having 3 hidden units with 'tanh' activation functions and linear output units. The resulting network functions are shown by the red curves, and the outputs of the three hidden units are shown by the three dashed curves.




I_k
$W_{k,j}$
$W_{j,i}$
a_j
O_i



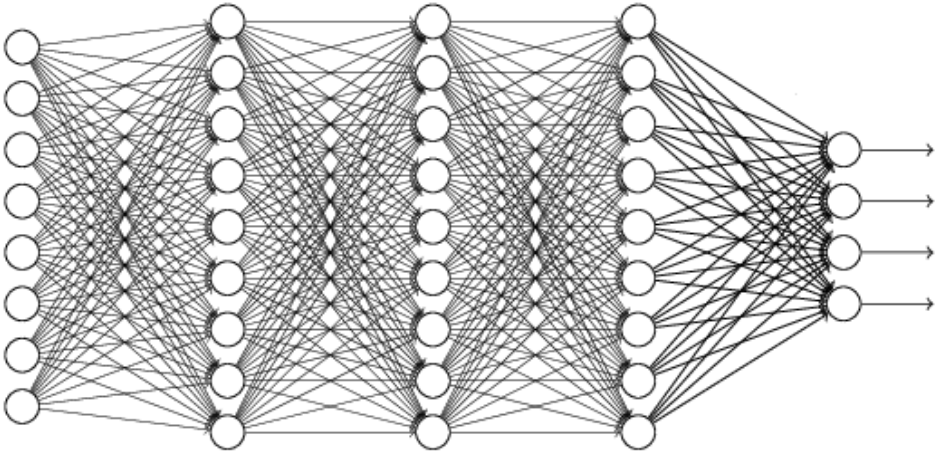


Figure 5.5 Geometrical view of the error function $E(\mathbf{w})$ as a surface sitting over weight space. Point $\mathbf{w}_A$ is a local minimum and $\mathbf{w}_B$ is the global minimum. At any point $\mathbf{w}_C$, the local gradient of the error surface is given by the vector ∇E.

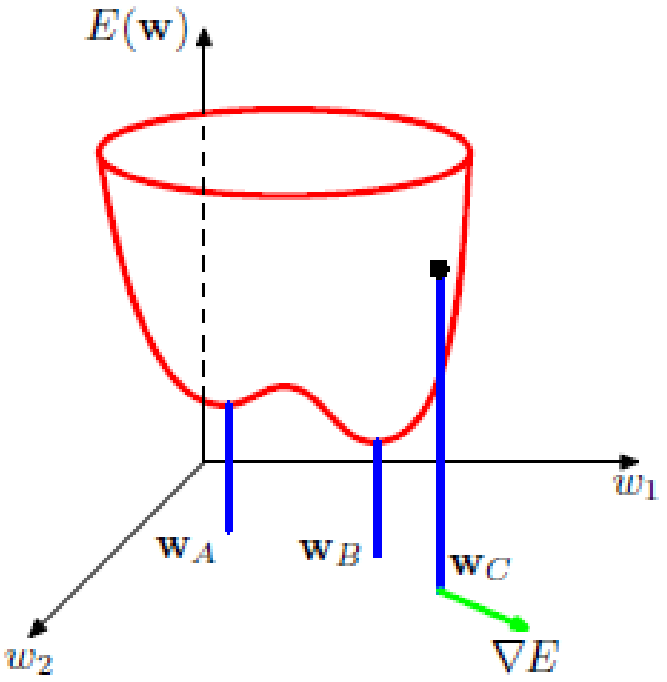
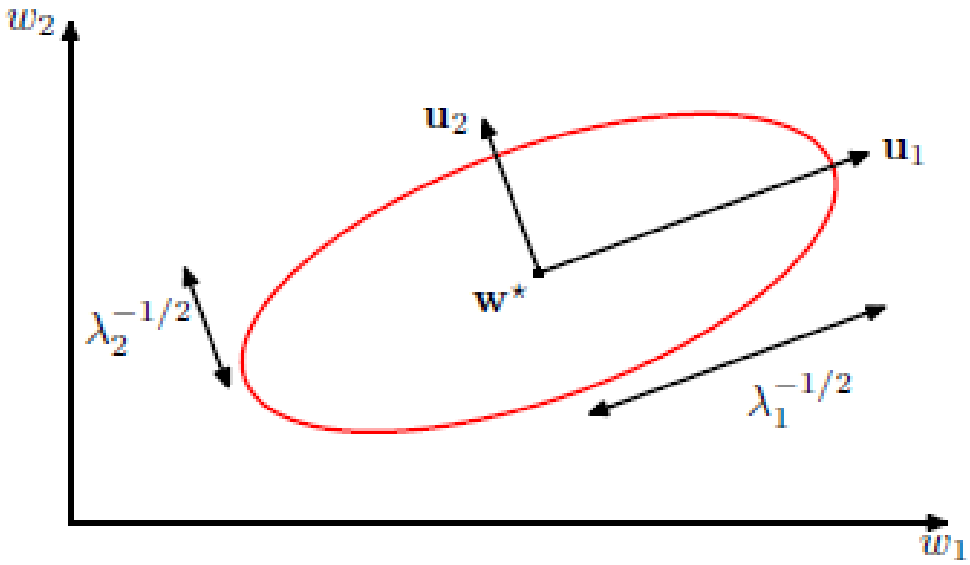


Figure 5.6 In the neighbourhood of a minimum $\mathbf{w}^*$, the error function can be approximated by a quadratic. Contours of constant error are then ellipses whose axes are aligned with the eigenvectors $\mathbf{u}_i$ of the Hessian matrix, with lengths that are inversely proportional to the square roots of the corresponding eigenvectors λ_i.



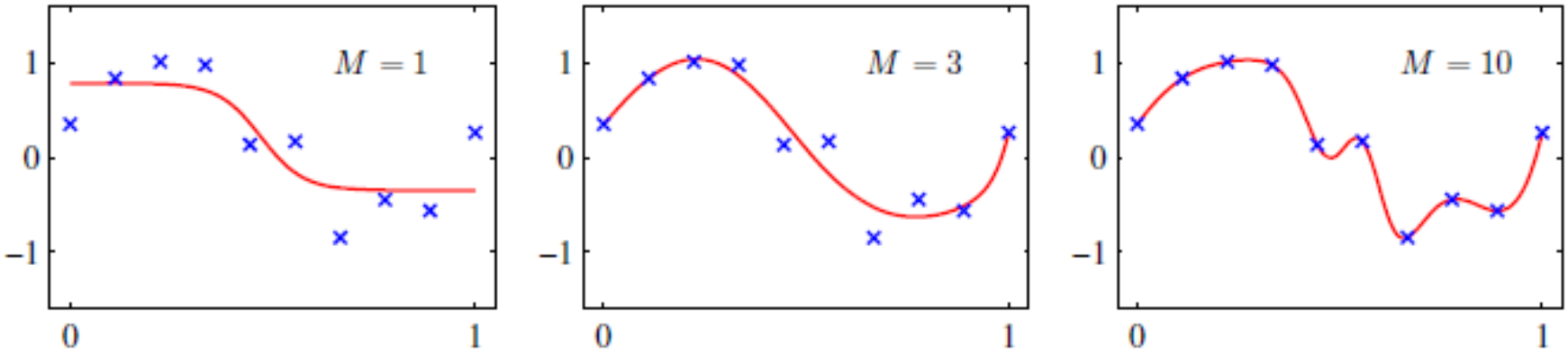
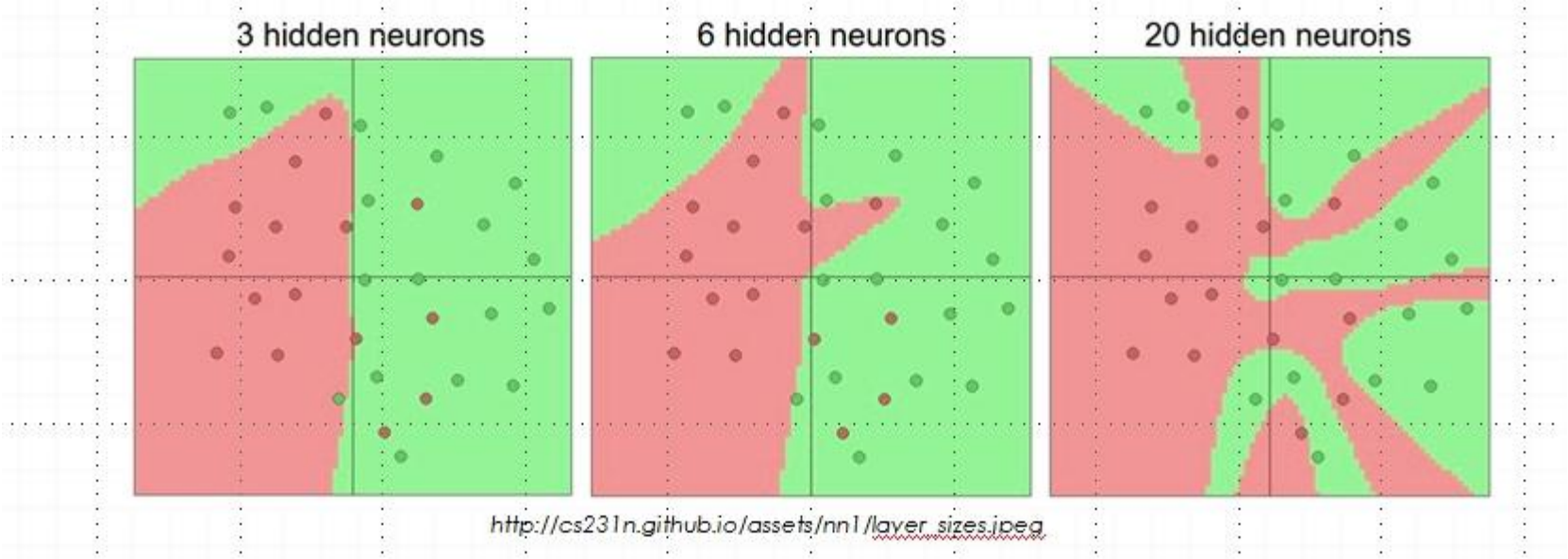


Figure 5.9 Examples of two-layer networks trained on 10 data points drawn from the sinusoidal data set. The graphs show the result of fitting networks having $M = 1$, 3 and 10 hidden units, respectively, by minimizing a sum-of-squares error function using a scaled conjugate-gradient algorithm.

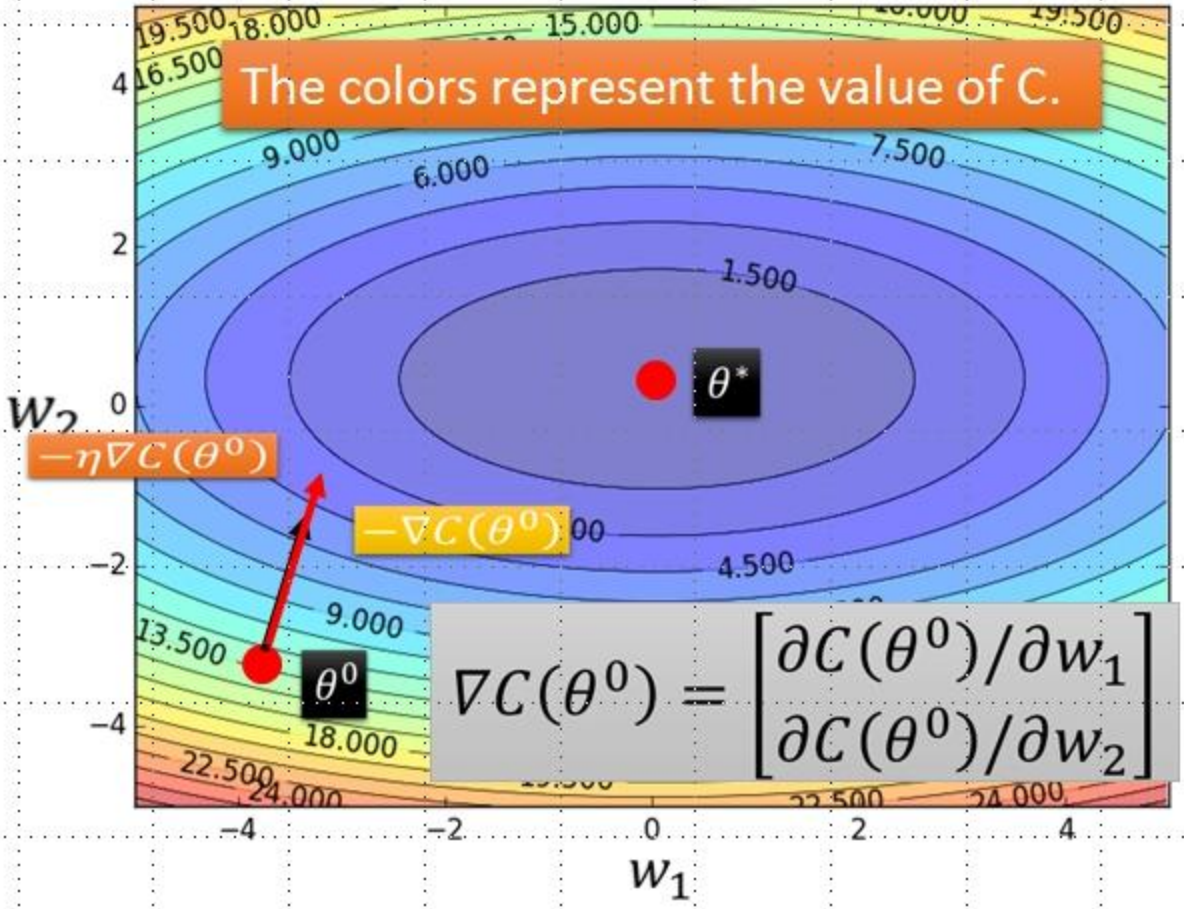


Gradient Descent

Assume there are only two parameters w_1 and w_2 in a network.

$$\theta = \{w_1, w_2\}$$

Error Surface



Randomly pick a starting point θ^0

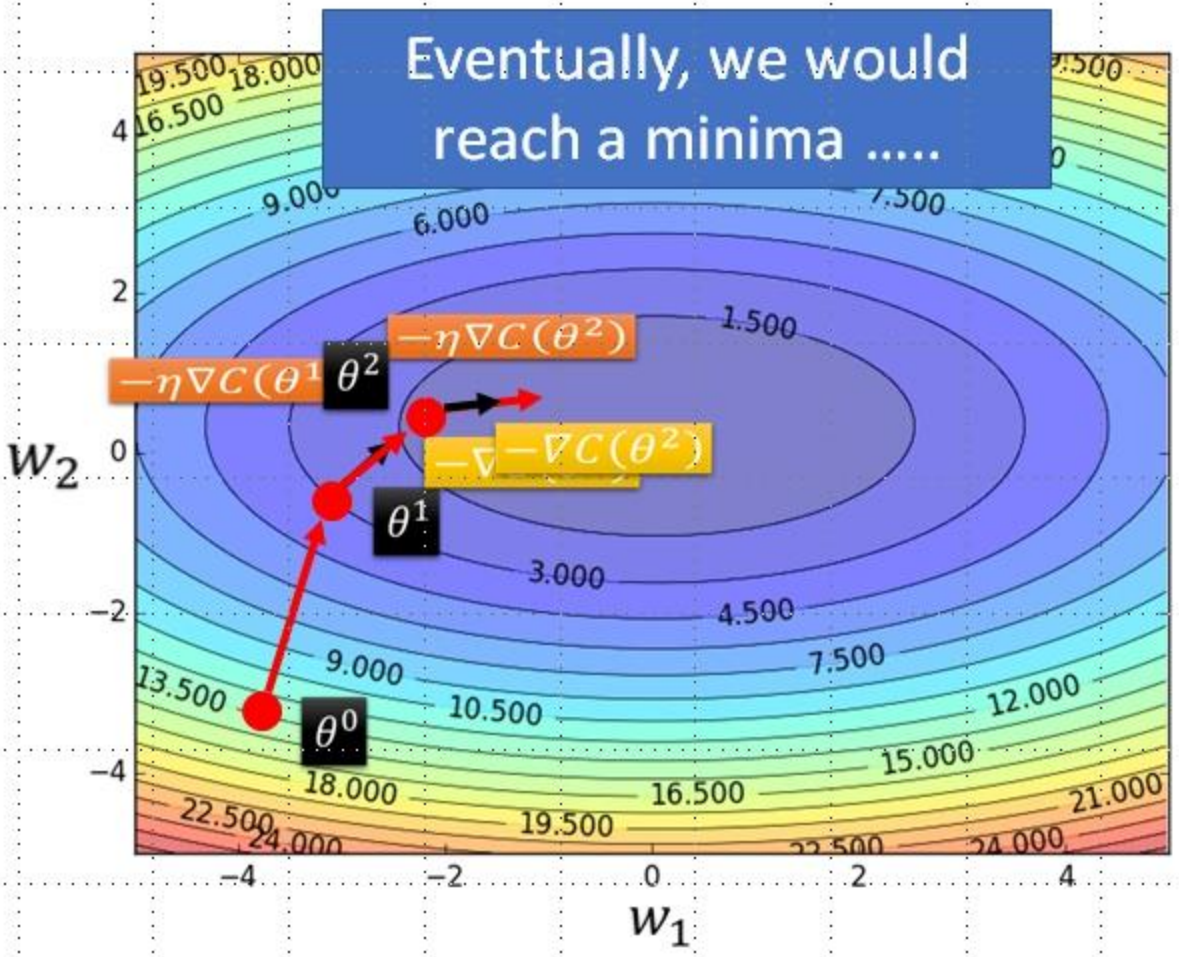
Compute the negative gradient at θ^0

$\rightarrow -\nabla C(\theta^0)$

Times the learning rate η

$\rightarrow -\eta\nabla C(\theta^0)$

Gradient Descent



Randomly pick a starting point θ^0

Compute the negative gradient at θ^0

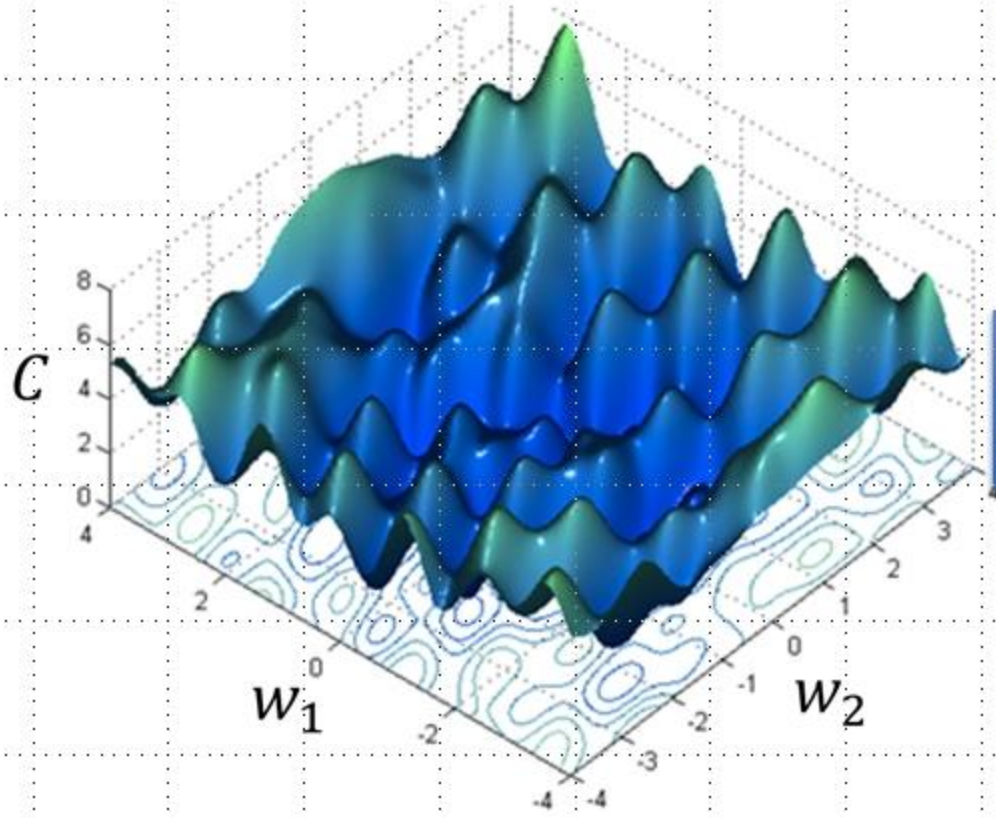
$-\nabla C(\theta^0)$

Times the learning rate η

$-\eta\nabla C(\theta^0)$

Local Minima

- Gradient descent never guarantee global minima



Different initial point θ^0

Reach different minima, so different results

Loss functions and output

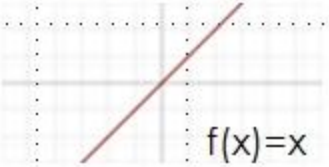
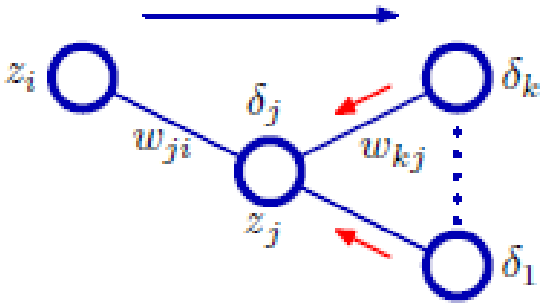
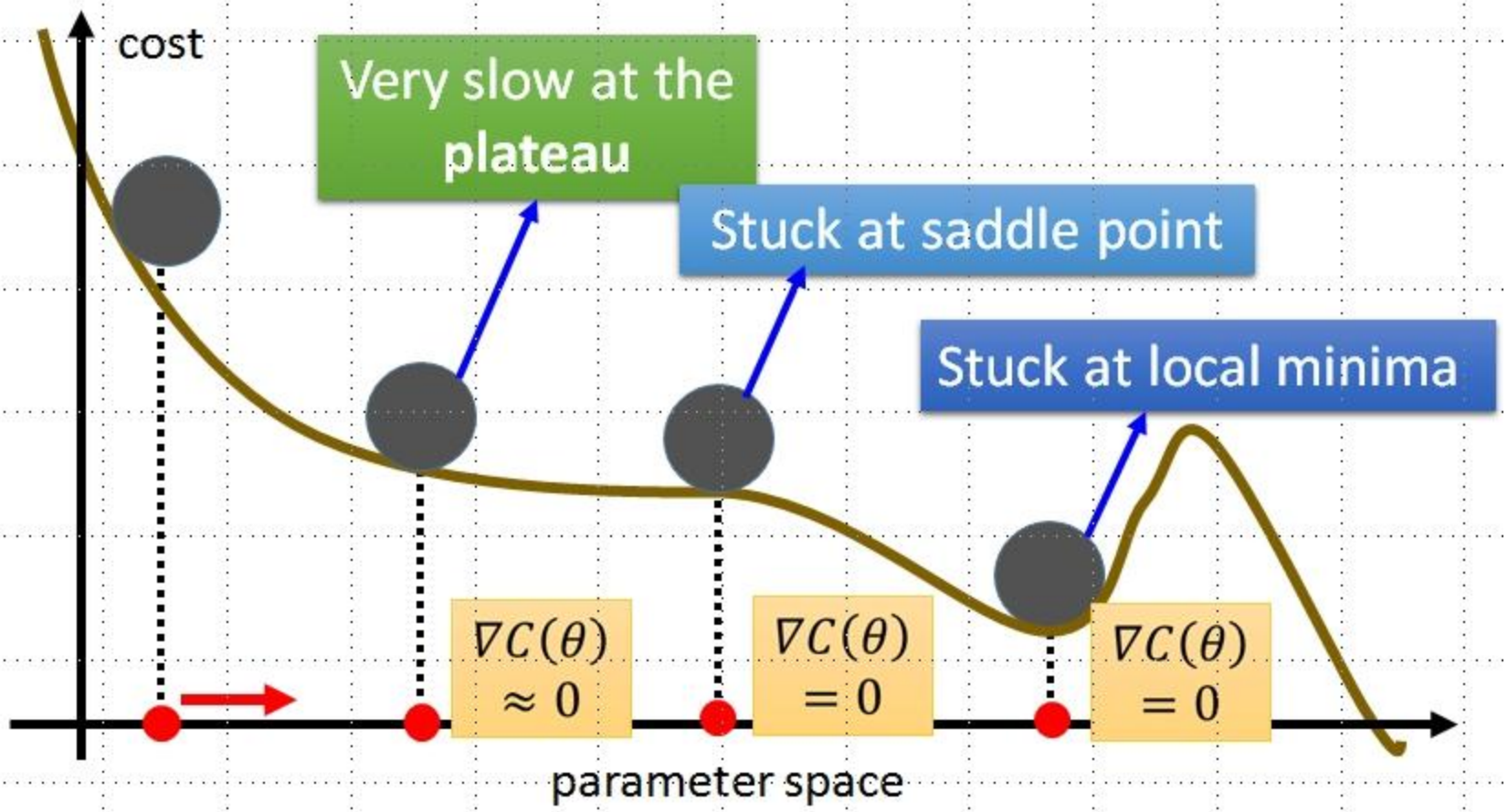
	Classification	Regression
Training examples	R^n x {class_1, ..., class_n} (one-hot encoding)	R^n x R^m
Output Layer	Soft-max [map R^n to a probability distribution] $$P(y = j \mid \mathbf{x}) = \frac{e^{\mathbf{x}^\mathsf{T}\mathbf{w}_j}}{\sum_{k=1}^{K} e^{\mathbf{x}^\mathsf{T}\mathbf{w}_k}}$$	Linear (Identity) or Sigmoid 
Cost (loss) function	Cross-entropy $$J(\theta) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{K}\left[y_k^{(i)} \log \hat{y}_k^{(i)} + \left(1 - y_k^{(i)}\right) \log\left(1 - \hat{y}_k^{(i)}\right)\right]$$	Mean Squared Error $$J(\theta) = \frac{1}{n}\sum_{i=1}^{n}\left(y^{(i)} - \hat{y}^{(i)}\right)^2$$ Mean Absolute Error $$J(\theta) = \frac{1}{n}\sum_{i=1}^{n}\left\lvert y^{(i)} - \hat{y}^{(i)}\right\rvert$$

Figure 5.7 Illustration of the calculation of δ_j for hidden unit j by backpropagation of the δ's from those units k to which unit j sends connections. The blue arrow denotes the direction of information flow during forward propagation, and the red arrows indicate the backward propagation of error information.

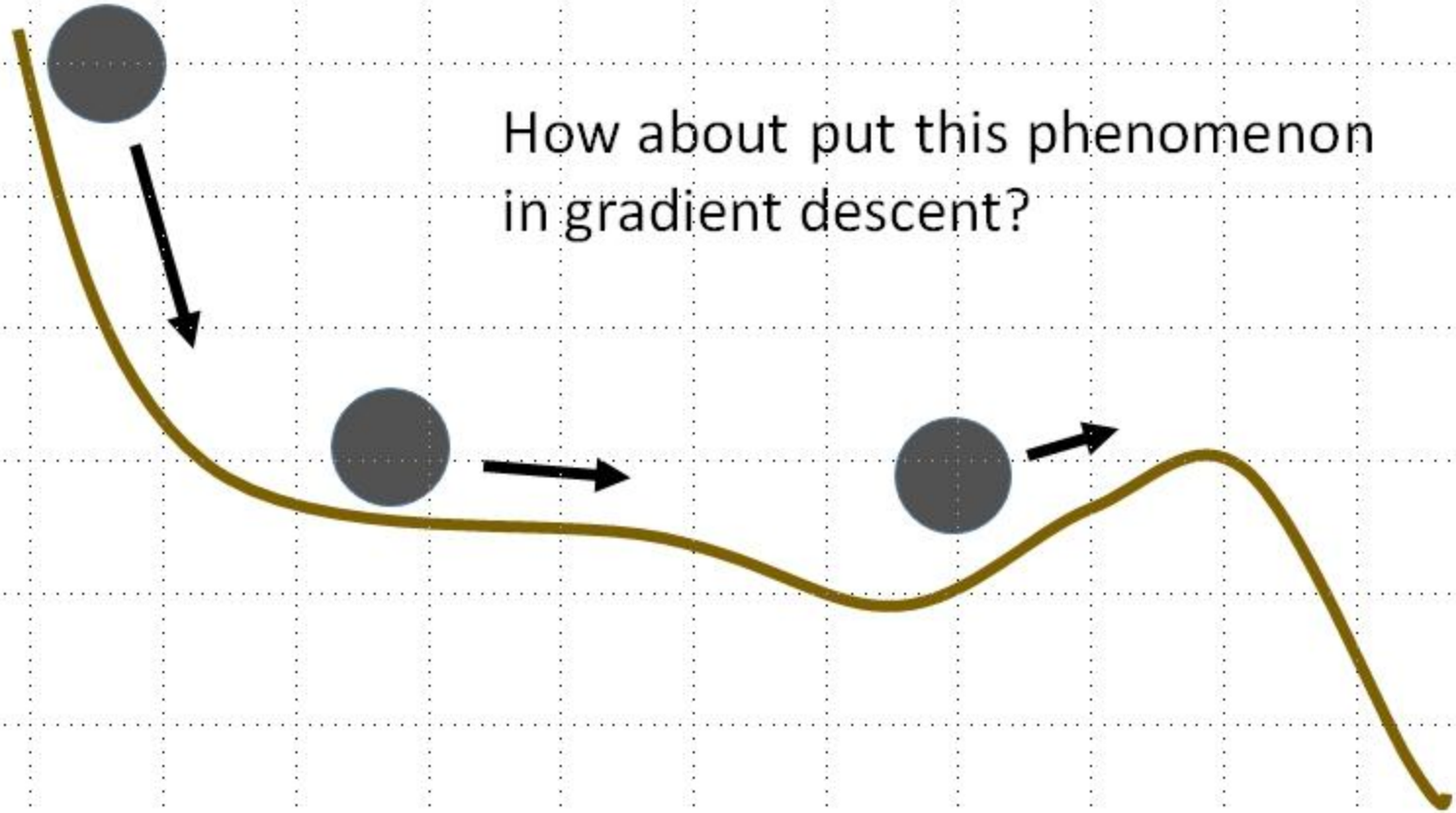


Besides local minima ……



In physical world ……

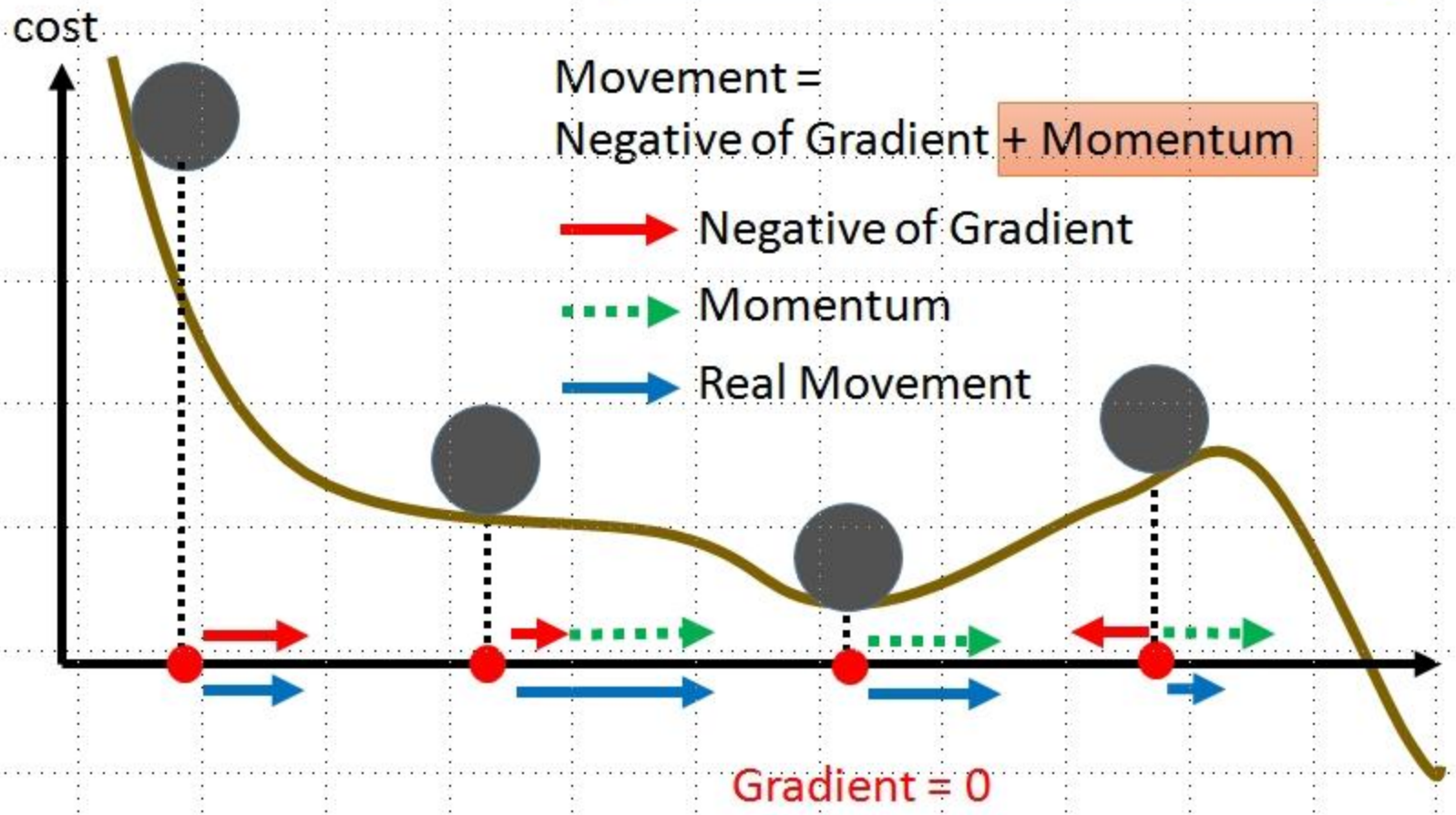
- Momentum



How about put this phenomenon in gradient descent?

Momentum

Still not guarantee reaching global minima, but give some hope ……



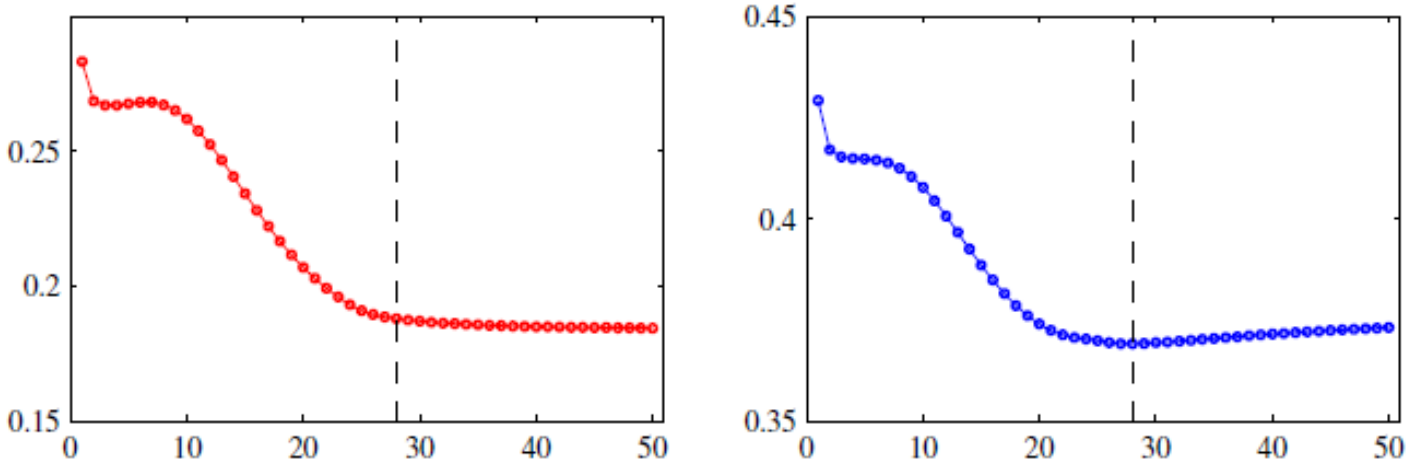
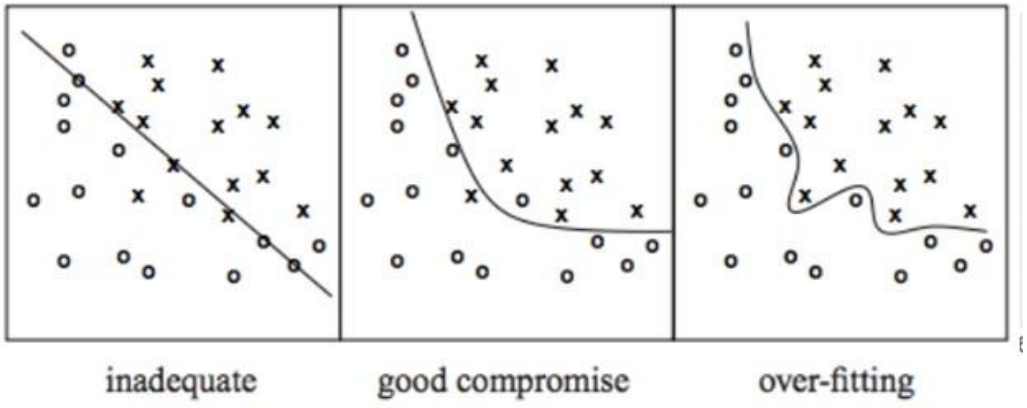
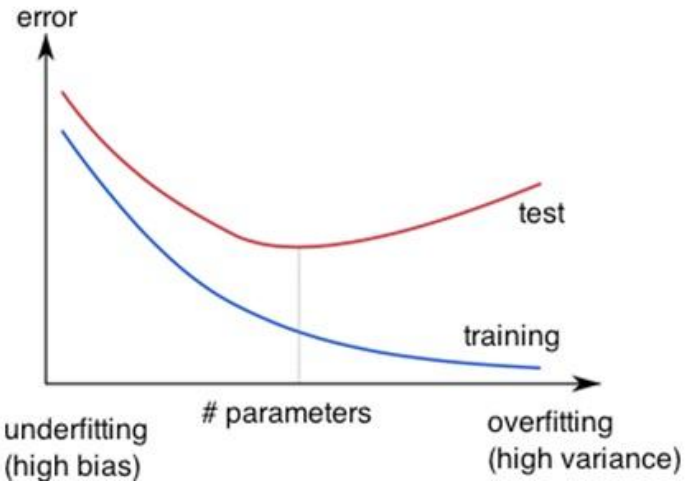
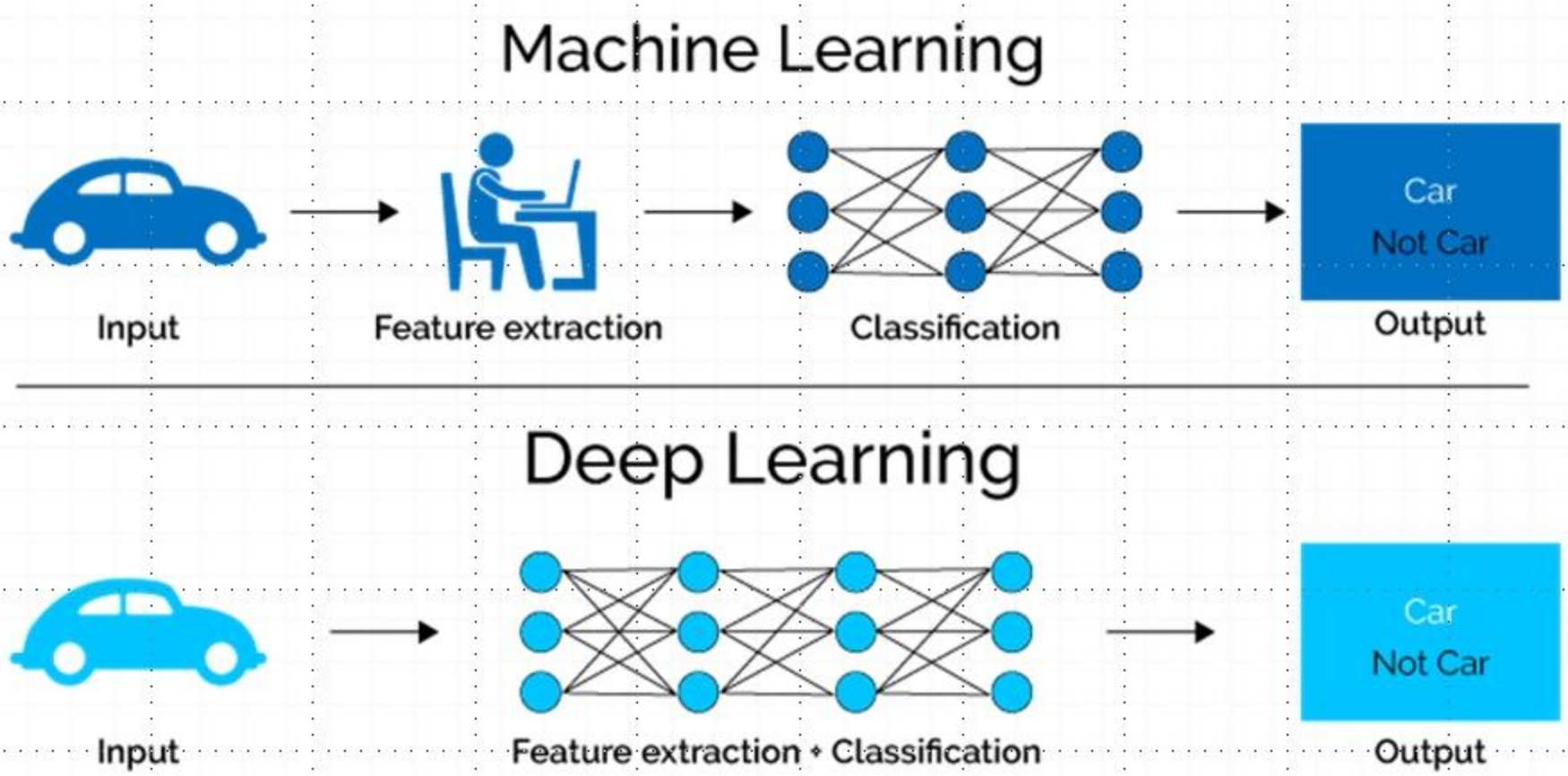


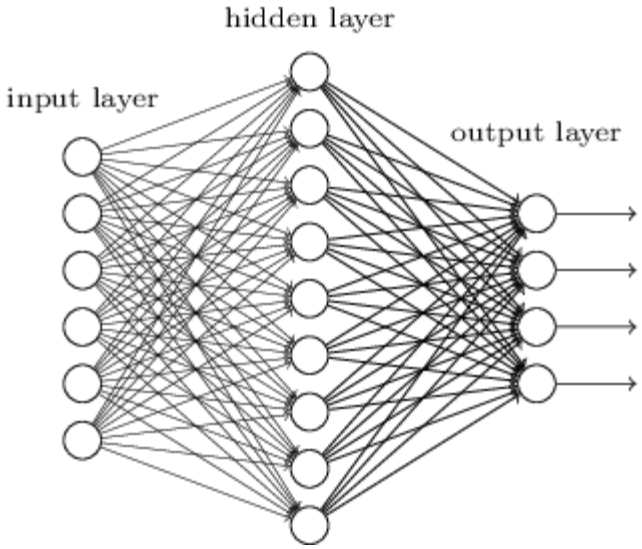
Figure 5.12 An illustration of the behaviour of training set error (left) and validation set error (right) during a typical training session, as a function of the iteration step the best generalization performance suggests that training dashed lines, corresponding to the minimum of the validati

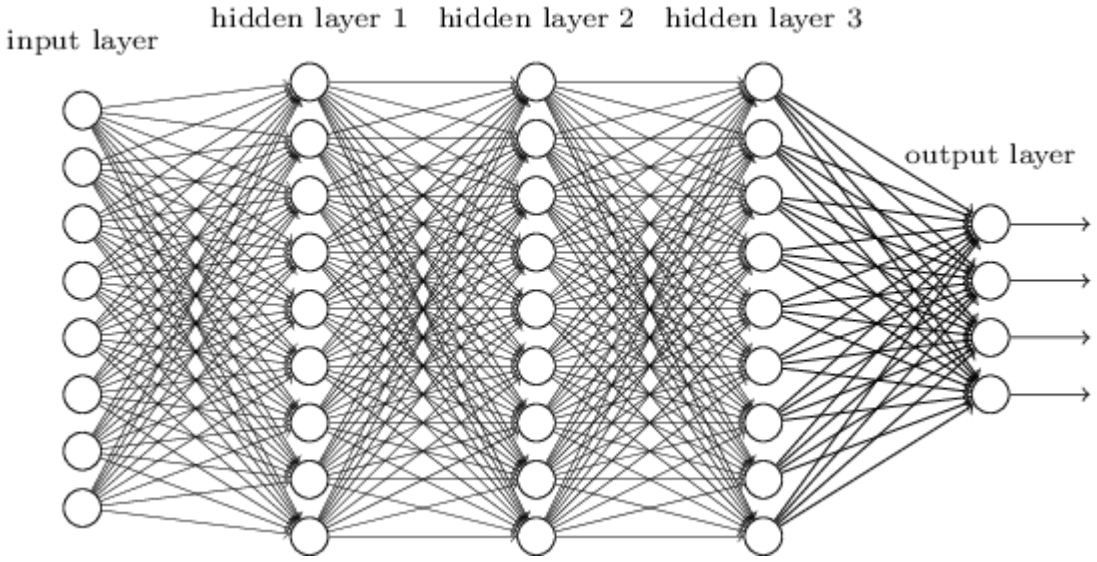




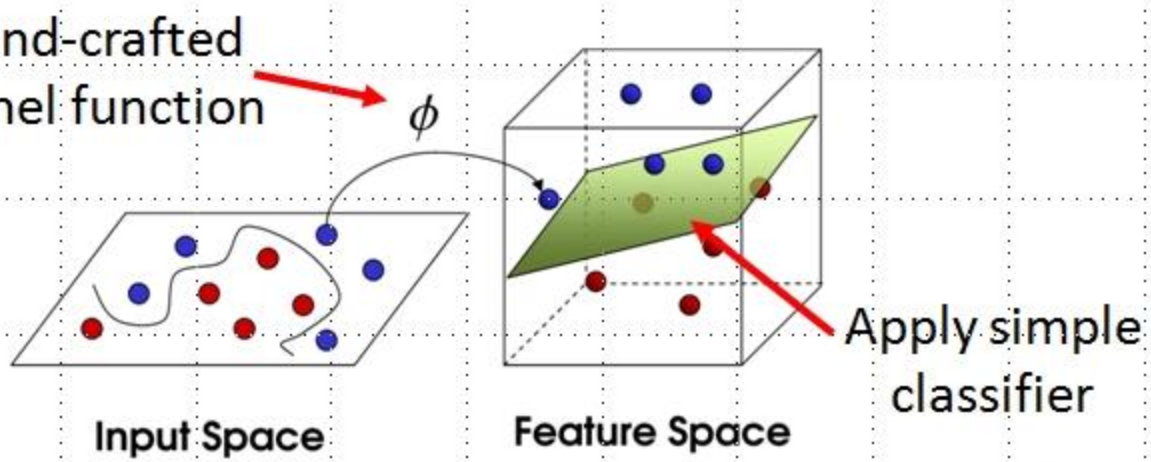

Machine Learning
Input
Feature extraction
Classification
Car
Not Car
Output
Deep Learning
Input
Feature extraction + Classification
Car
Not Car
Output


@teenybis


hidden layer
input layer
output layer

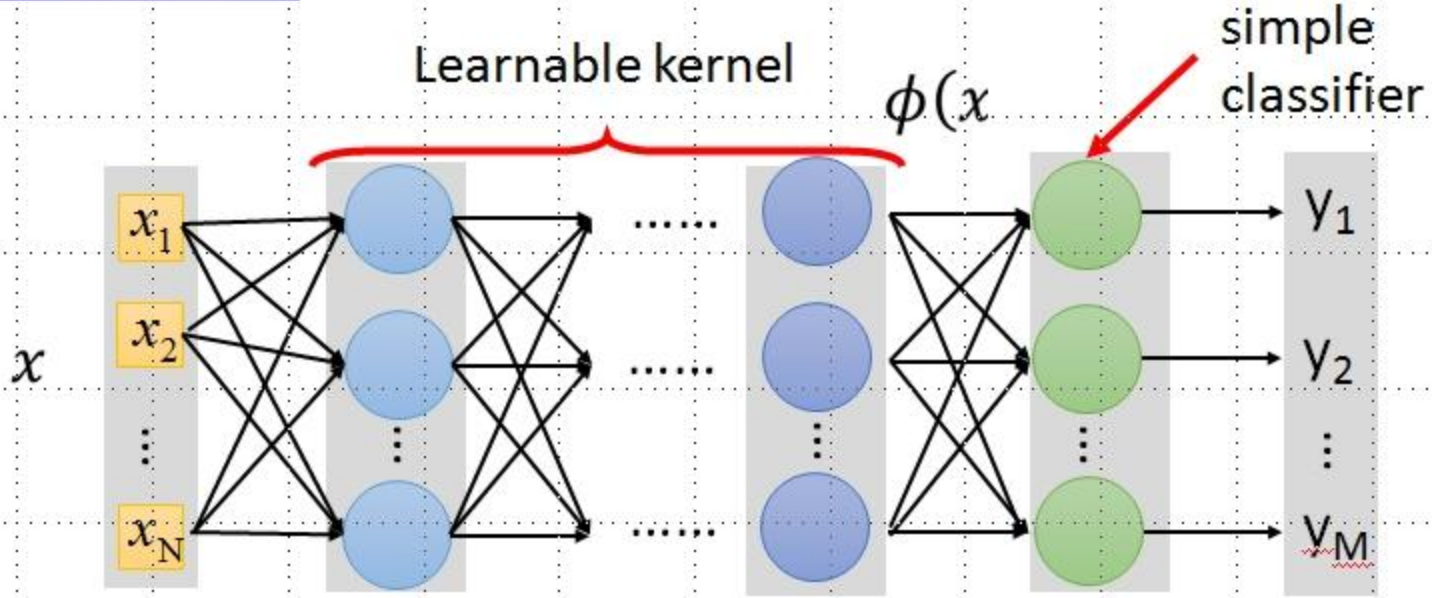

input layer
hidden layer 1
hidden layer 2
hidden layer 3
output layer

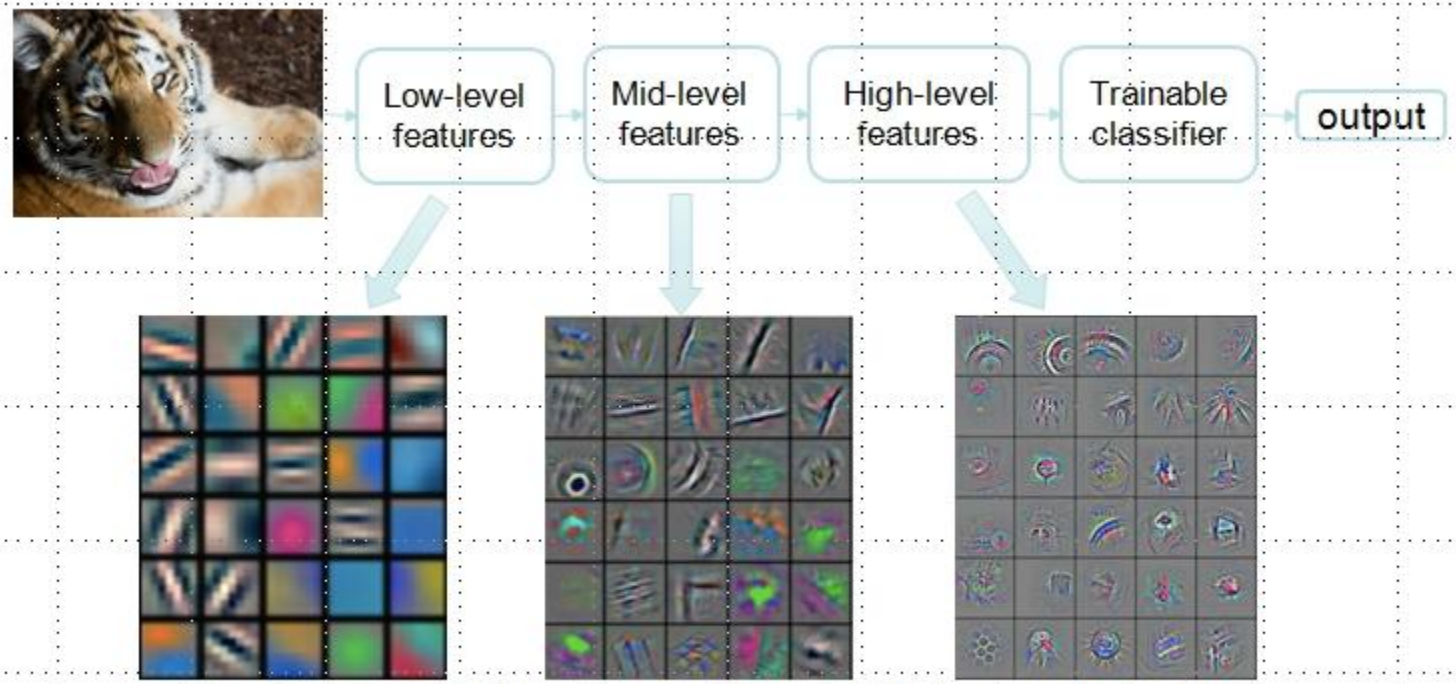
SVM



Source of image: http://www.gipsa-lab.grenoble-inp.fr/transfert/seminaire/455_Kadri2013Gipsa-lab.pdf

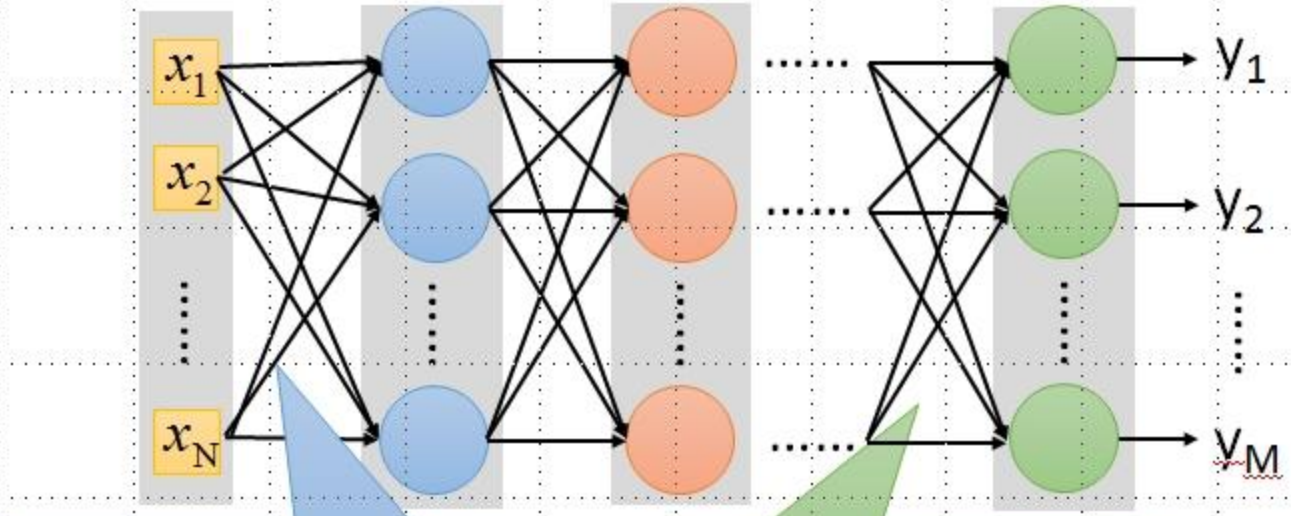
Deep Learning





Feature visualization of convolutional net trained on ImageNet (Zeiler and Fergus, 2013)

Vanishing Gradient Problem



Smaller gradients

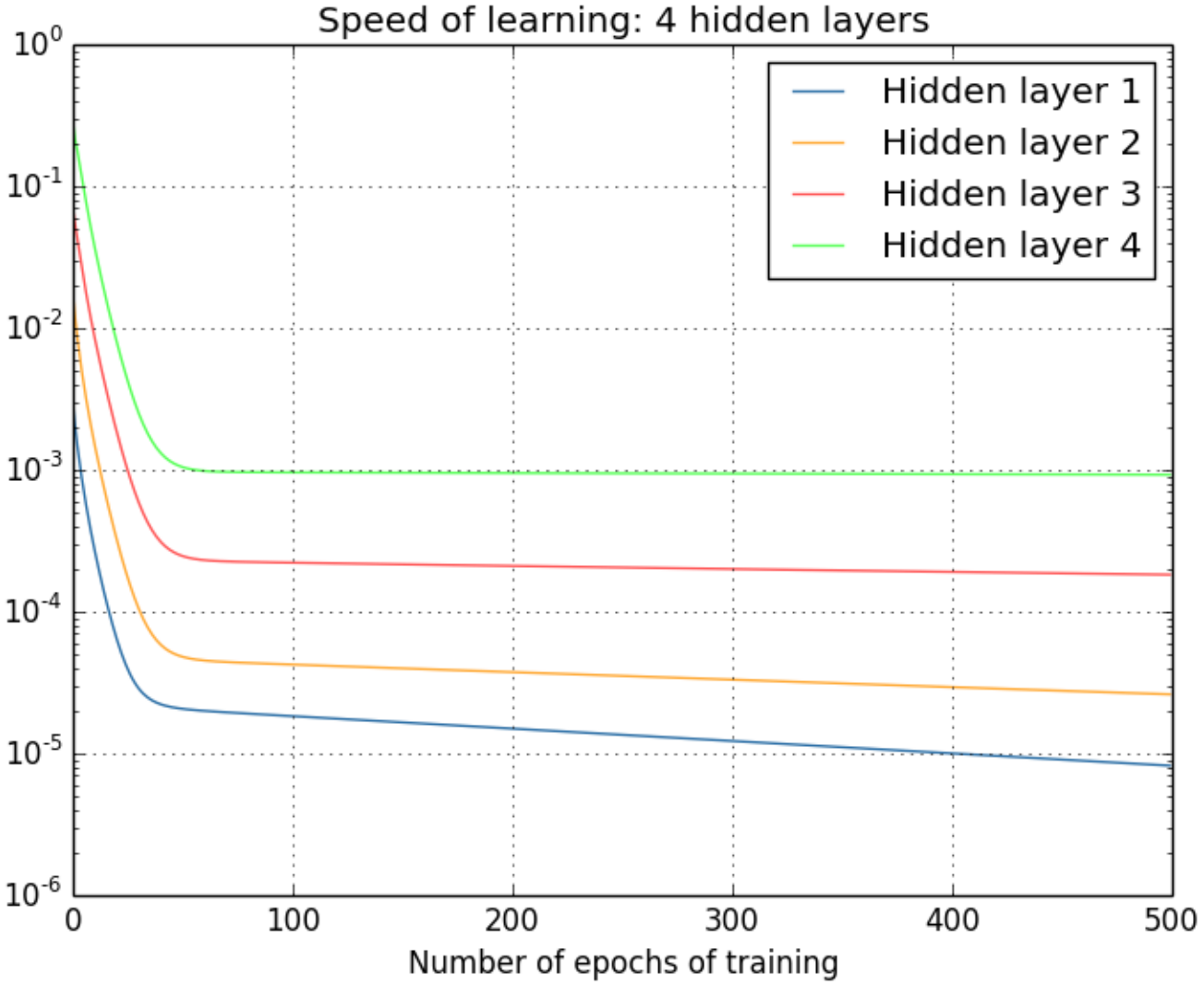
Learn very slow

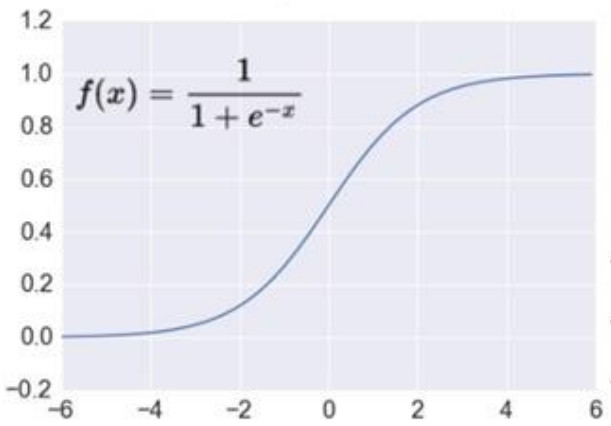
Almost random

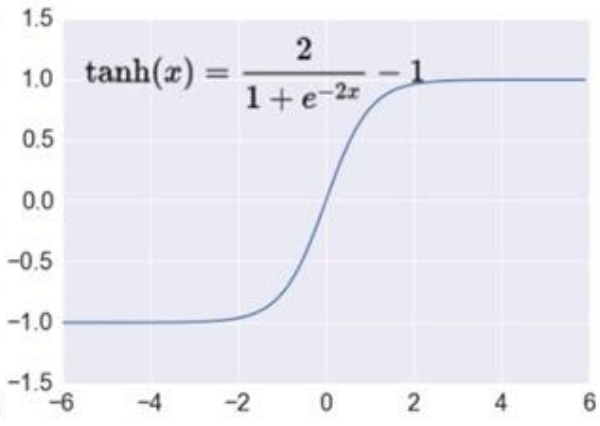
Larger gradients

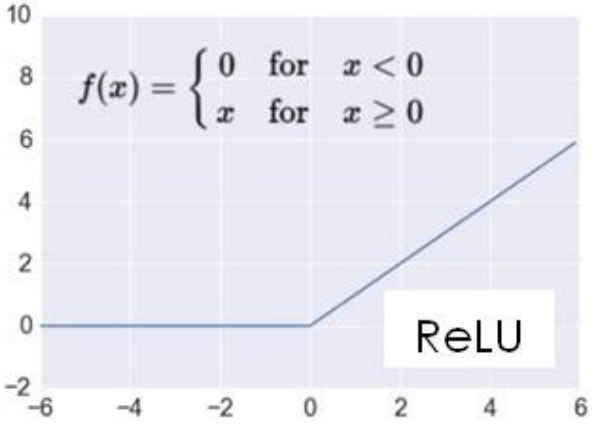
Learn very fast

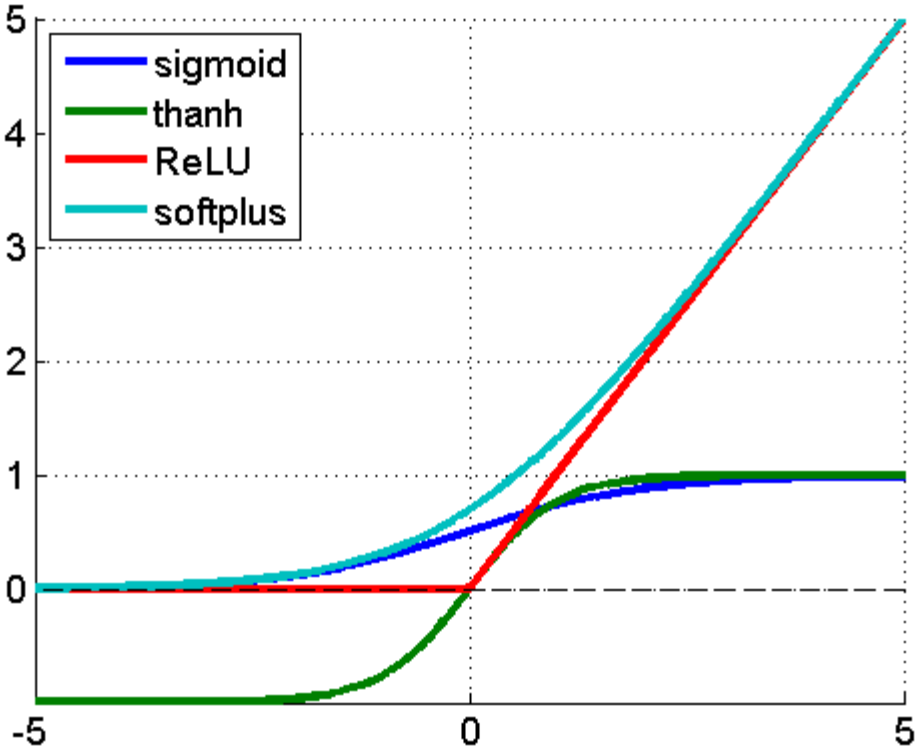
Already converge

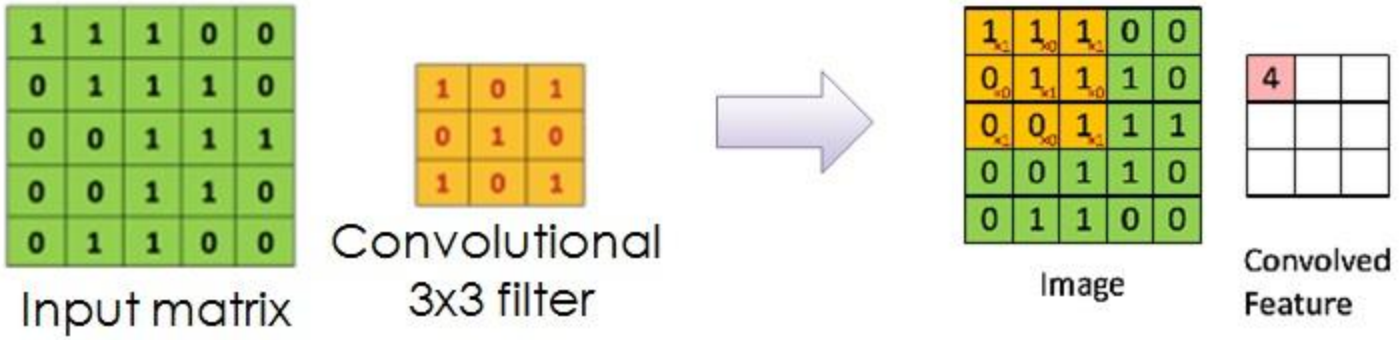

Speed of learning: 4 hidden layers
Hidden layer 1
Hidden layer 2
Hidden layer 3
Hidden layer 4
10^{0}
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
0
100
200
300
400
500
Number of epochs of training

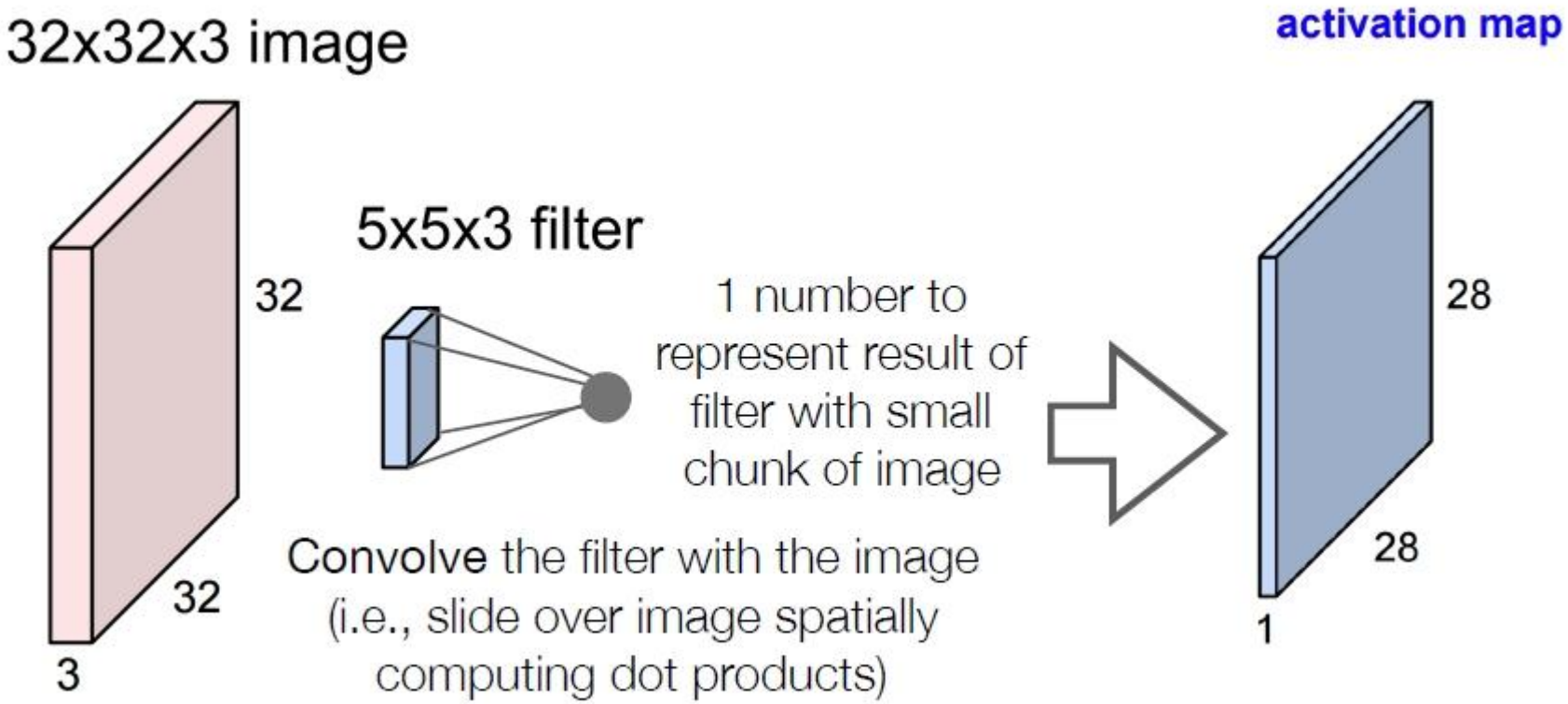

$f(x) = \frac{1}{1+e^{-x}}$


$\tanh(x) = \frac{2}{1+e^{-2x}} - 1$


$f(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases}$
ReLU


sigmoid
thanh
ReLU
softplus


Input matrix
Convolutional
3x3 filter
Image
Convolved
Feature


32x32x3 image
activation map
5x5x3 filter
32
32
3
1 number to
represent result of
filter with small
chunk of image
28
28
1
Convolve the filter with the image
(i.e., slide over image spatially
computing dot products)

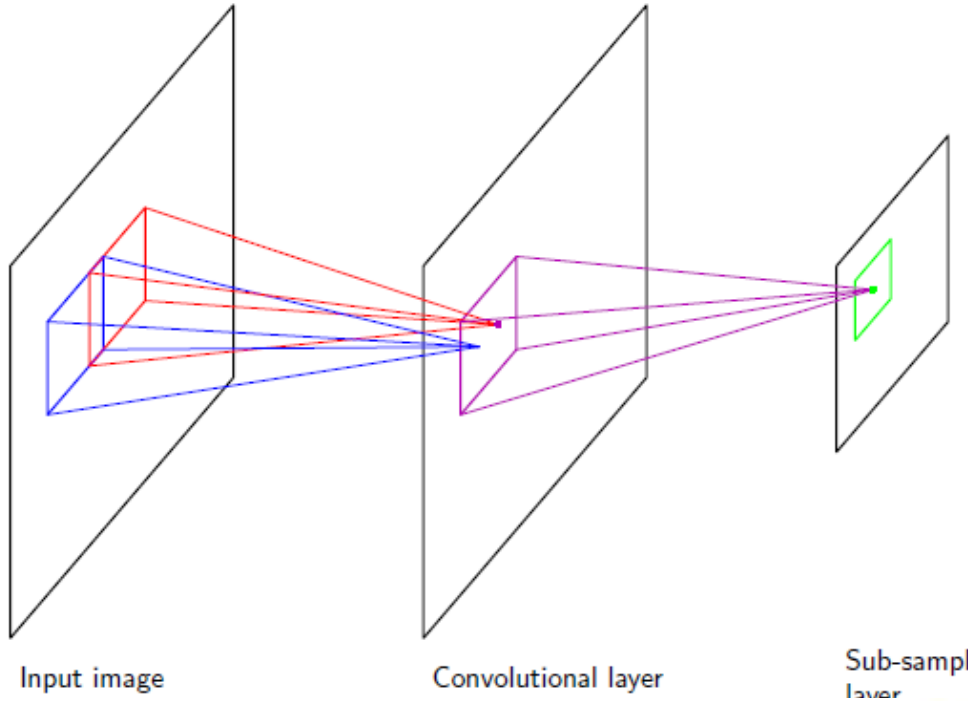
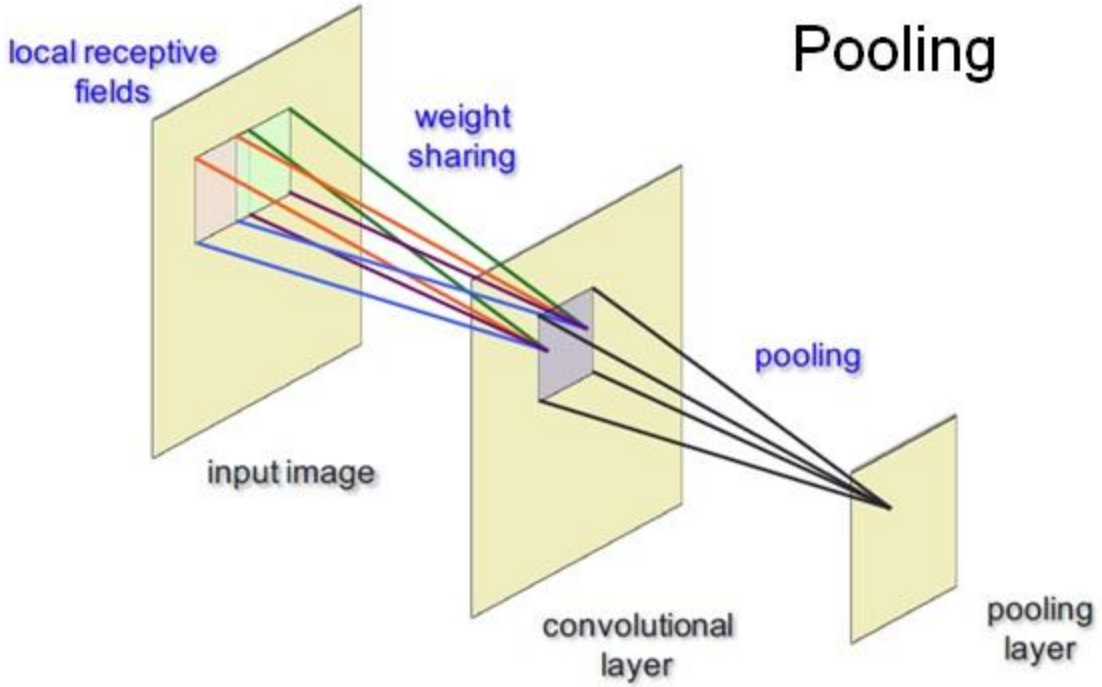
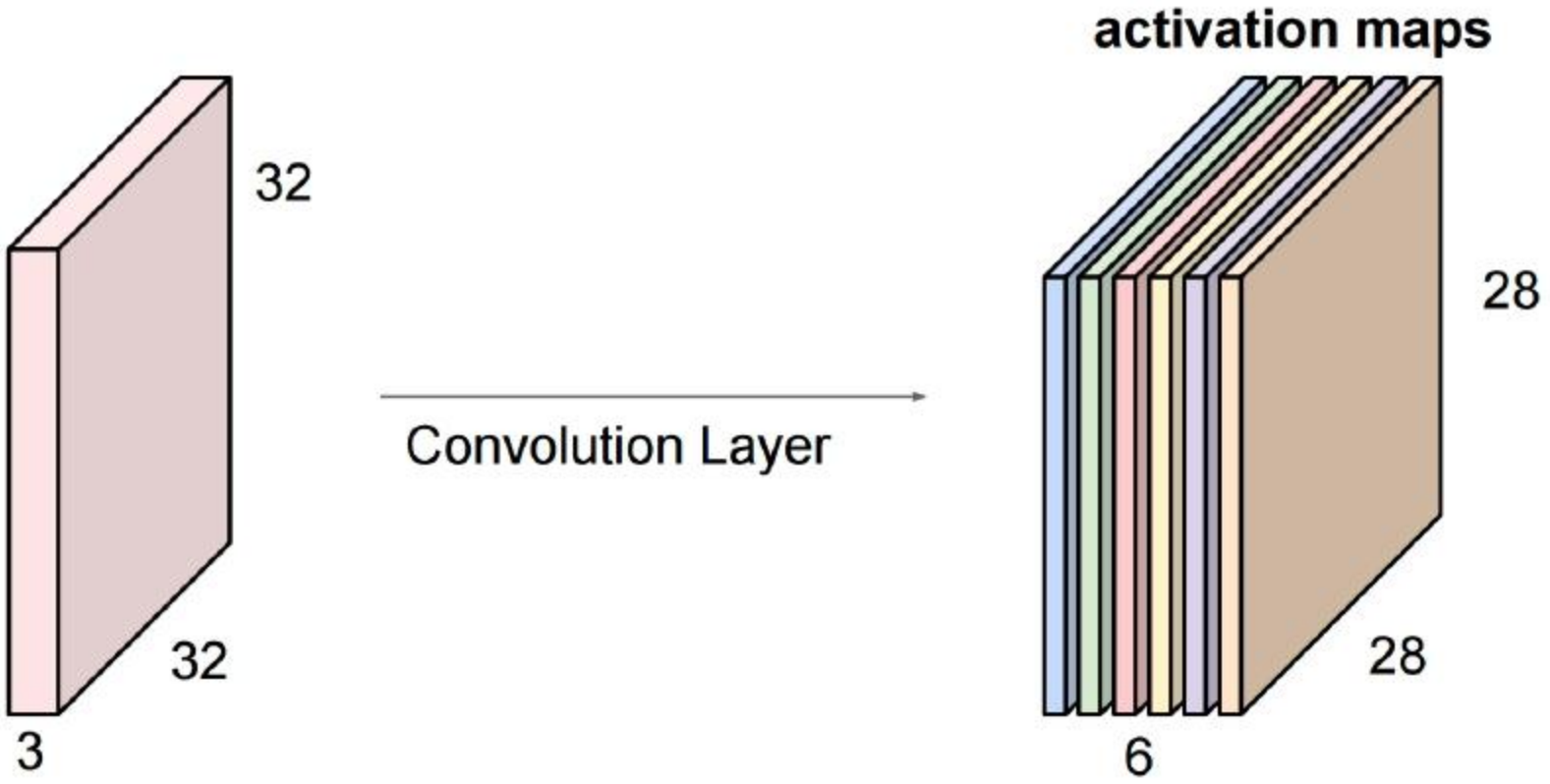


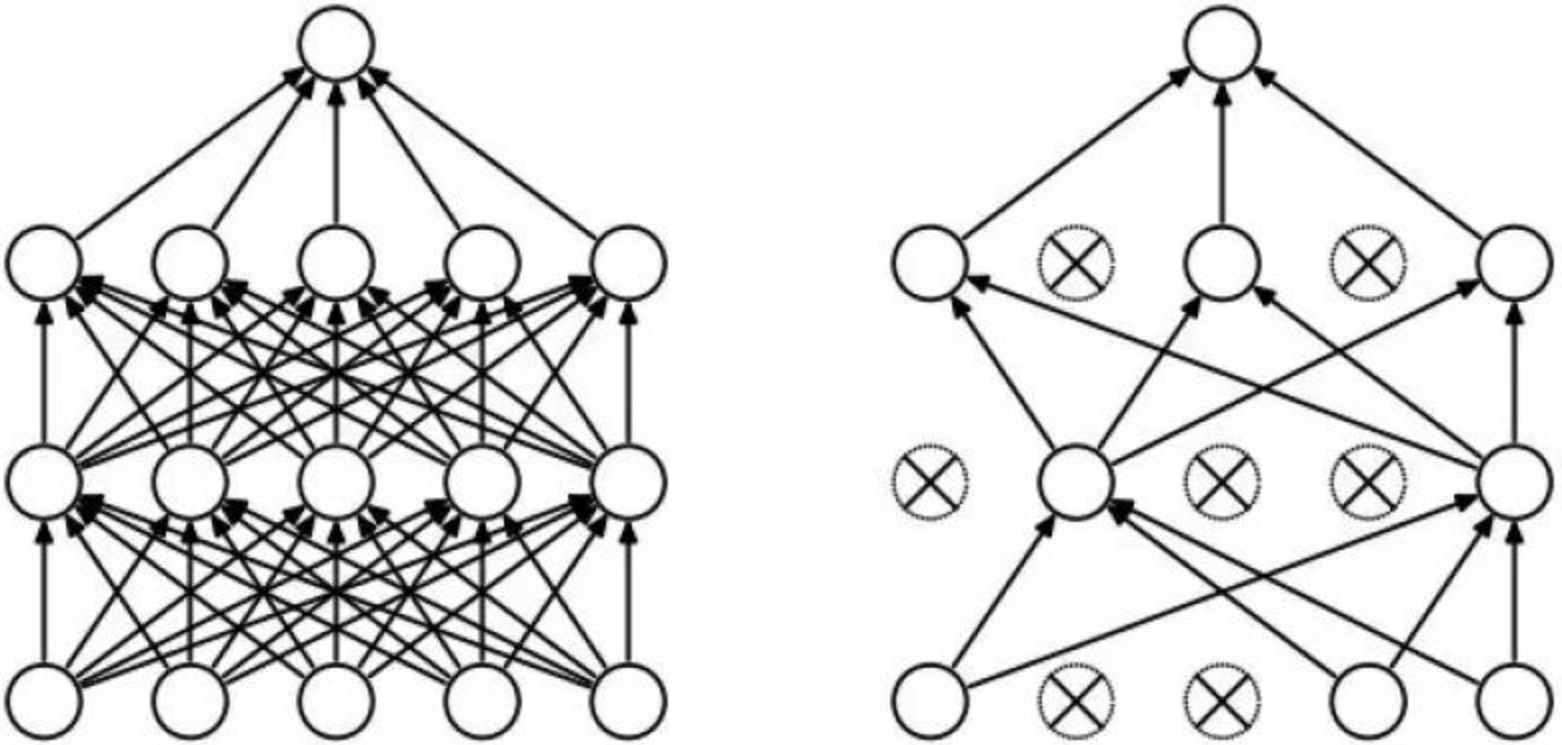
Figure 5.17 Diagram illustrating part of a convolutional neura
tional units followed by a layer of subsampling u
layers may be used.



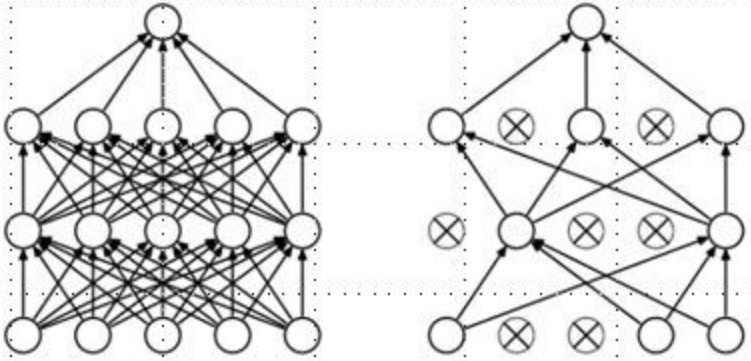


Stacking up multiple filters yields “new image”

Dropout



Regularization



Dropout

- Randomly drop units (along with their connections) during training
- Each unit retained with fixed probability p, independent of other units
- Hyper-parameter p to be chosen (tuned)

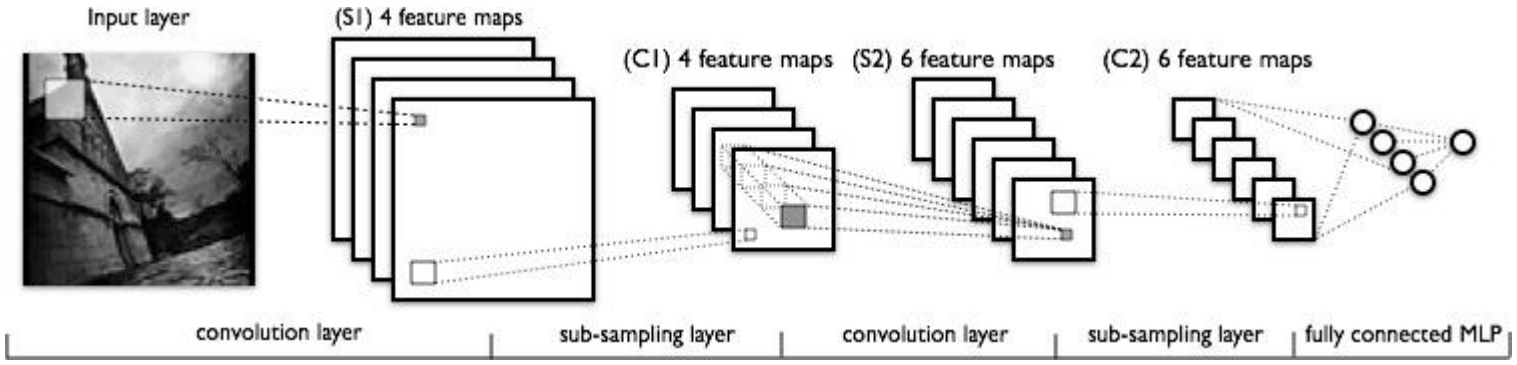
L2 = weight decay

- Regularization term that penalizes big weights, added to the objective
- Weight decay value determines how dominant regularization is during gradient computation
- Big weight decay coefficient → big penalty for big weights

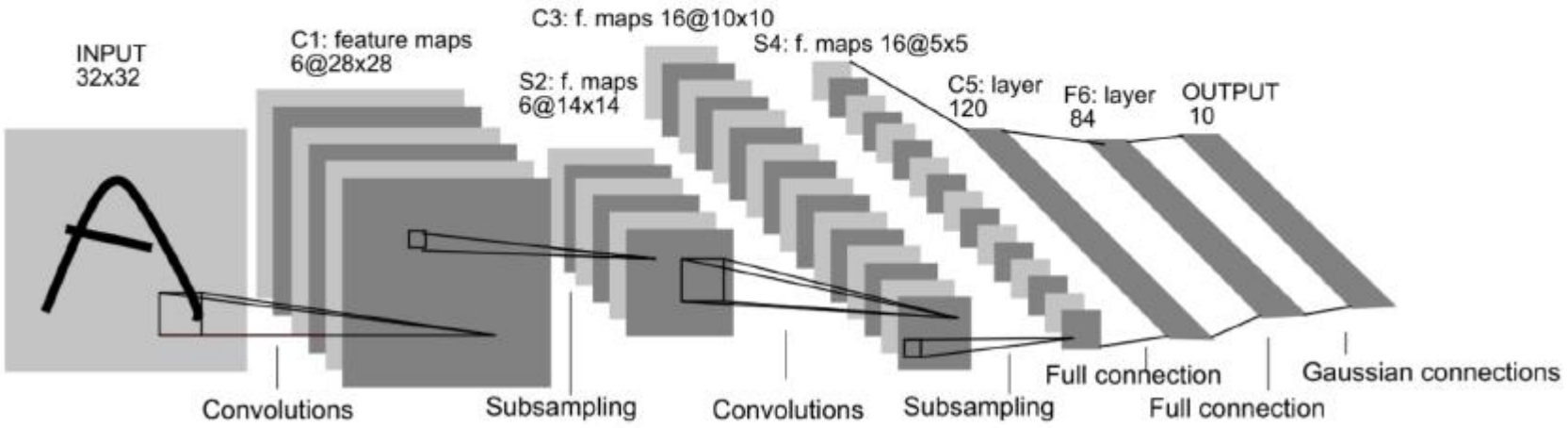
$$J_{reg}(\theta) = J(\theta) + \lambda \sum_k \theta_k^2$$

Early-stopping

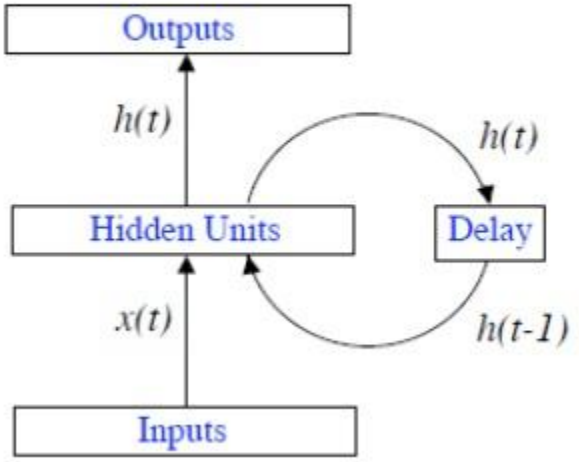
- Use validation error to decide when to stop training
- Stop when monitored quantity has not improved after n subsequent epochs



LeNet 5 [LeCun et al., 1998]



Recurrent neural network



Back Propagation Through Time (BPTT)