Announcements

Reminder: self-grading forms for ps1 and ps2 due 10/5 at midnight (Boston)

- ps3 out on Thursday, due 10/8 (1 week)

- LAB this week: go over solutions for the first two homeworks

Agglomerative Clustering Example
(bottom-up clustering)

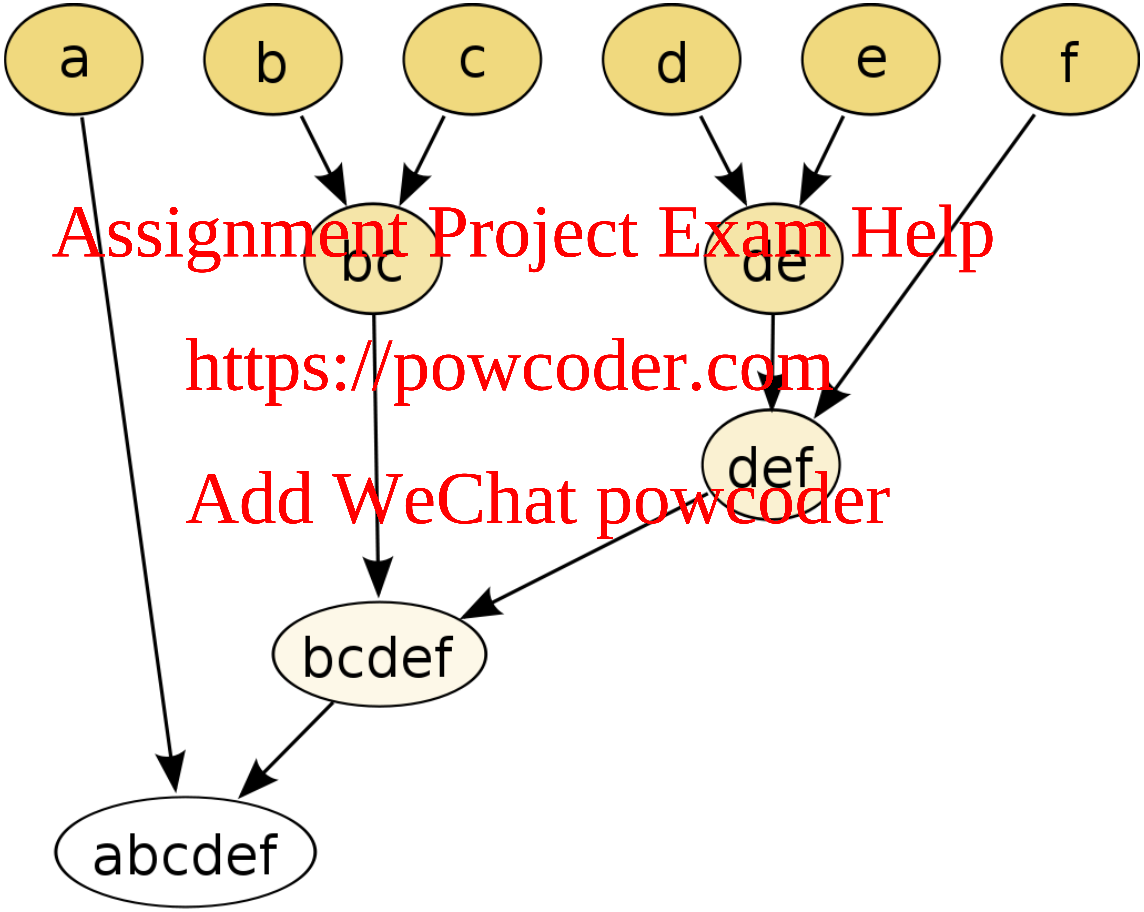
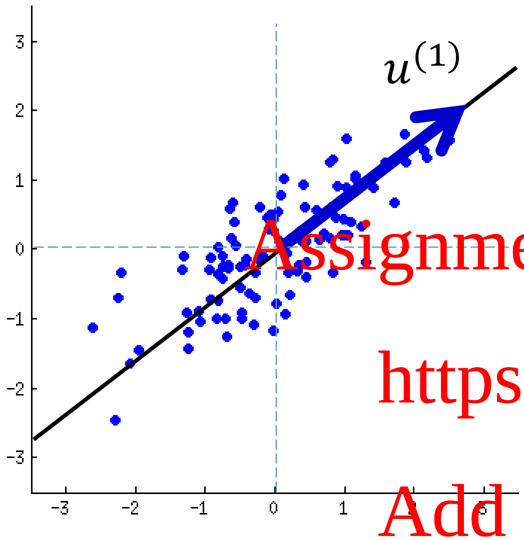


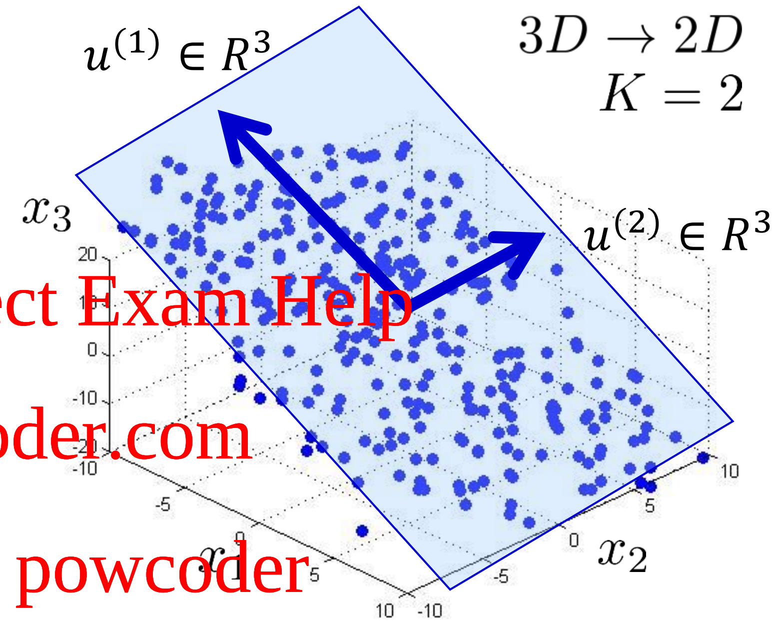
Image source: https://en.wikipedia.org/wiki/Hierarchical_clustering

K-Means for Image Compression

Figure 9.3 Two examples of the application of the K-means clustering algorithm to image segmentation showing the initial images together with their K-means segmentations obtained using various values of K. This also illustrates of the use of vector quantization for data compression, in which smaller values of K give higher compression at the expense of poorer image quality.

Choose subspace with minimal "information loss"





Reduce from 2-dimension to 1-dimension: Find a direction (a vector $u^{(1)}$) onto which to project the data, so as to minimize the projection error.

Reduce from n-dimension to K-dimension: Find K vectors $u^{(1)}, u^{(2)}, \ldots, u^{(K)}$ onto which to project the data so as to minimize the projection error.

PCA Solution

- The solution turns out to be the first K eigenvectors of the data covariance matrix (see Bishop 12.1 for details)

- Closed-form, use Singular Value Decomposition (SVD) on covariance matrix

What features to use?

Edges?

Shapes?

Color?

Today: Outline

- **Neural networks:** artificial neuron, MLP, sigmoid units; neuroscience inspiration, output vs hidden layers; linear vs nonlinear networks;

- **Feed-forward networks**

Intro to Neural Networks

Motivation

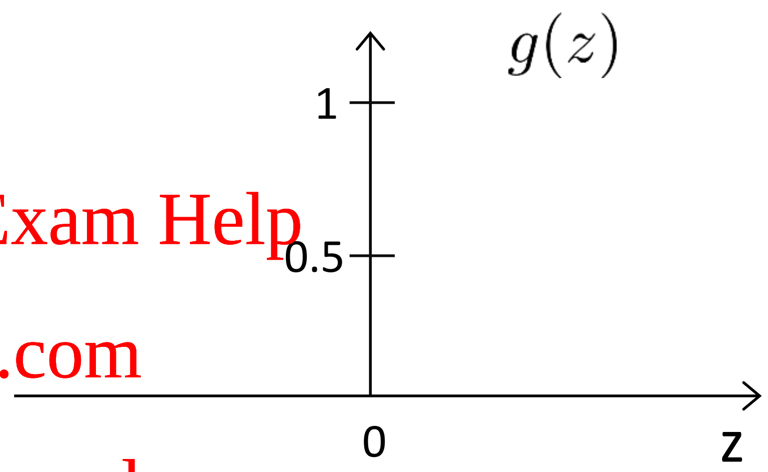
Recall: Logistic Regression

$$0 \leq h_\theta(x) \leq 1$$

$$h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}}$$

$$g(z) = \frac{1}{1 + e^{-z}}$$

sigmoid/logistic function



Output is probability of label 1 given input

$$p(y = 1|x) = \frac{1}{1 + e^{-\theta^T x}}$$

predict "$y = 1$" if $h_\theta(x) \geq 0.5$

predict "$y = 0$" if $h_\theta(x) < 0.5$

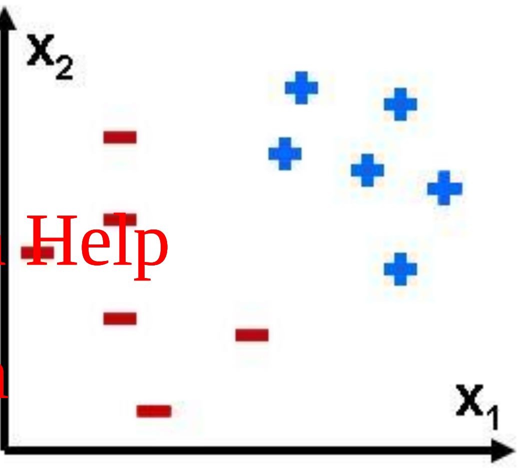
Recall: Logistic Regression Cost

Logistic Regression Hypothesis:

$$h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}}$$

θ: parameters

$D = \{x^i, y^i\}$: data



Logistic Regression Cost Function:

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \mathrm{Cost}(h_\theta(x^{(i)}), y^{(i)})$$

$$= -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log (1 - h_\theta(x^{(i)}))\right]$$

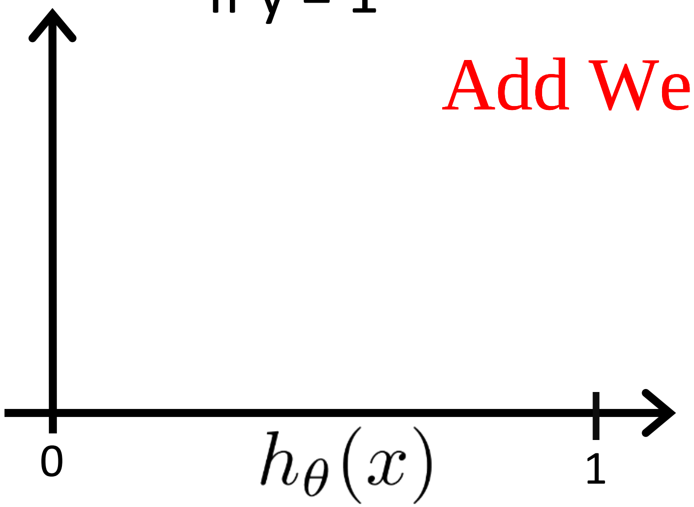
Goal: minimize cost $\quad \min_\theta J(\theta)$

Cost: Intuition

Logistic regression cost function

$$\mathrm{Cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$

If y = 1



$\mathrm{Cost} = 0$ if $y = 1, h_\theta(x) = 1$

But as $h_\theta(x) \rightarrow 0$

$\mathrm{Cost} \rightarrow \infty$

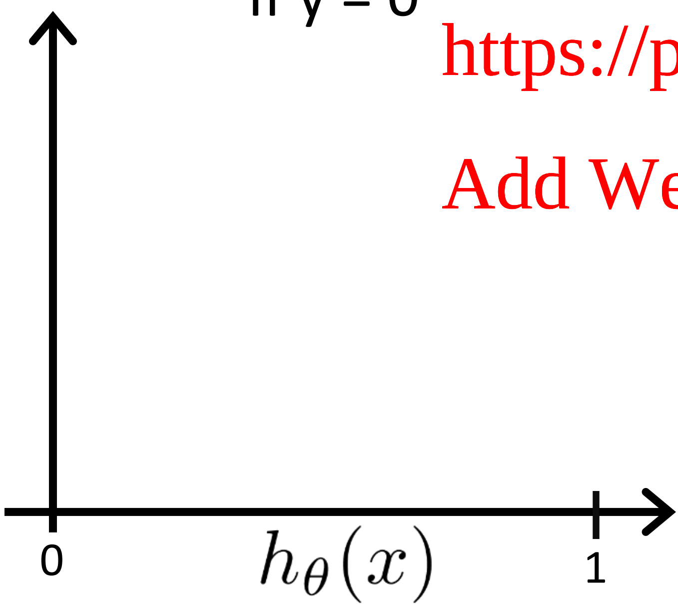
Captures intuition that if $h_\theta(x) = 0$, (predict $P(y = 1|x;\theta) = 0$), but $y = 1$, we'll penalize learning algorithm by a very large cost.

Cost: Intuition

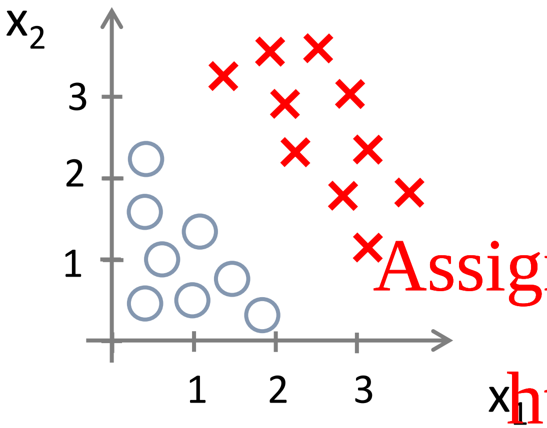
Logistic regression cost function

$$\mathrm{Cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$

If y = 0



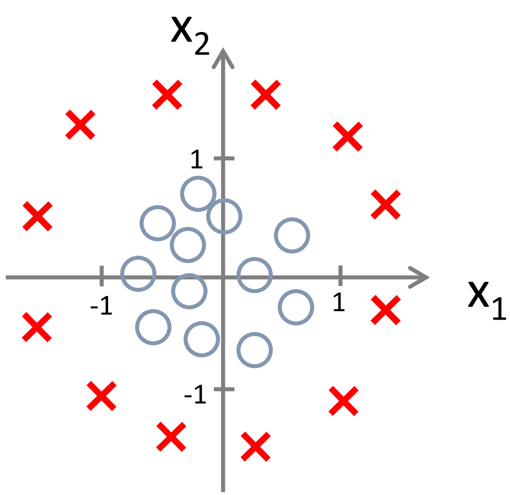
Decision boundary



$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$$

Predict "$y = 1$" if $-3 + x_1 + x_2 \geq 0$

Non-linear decision boundaries



Replace features with non-linear functions
e.g. log, cosine, or polynomial

$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2)$$

Predict "y = 1" if $-1 + x_1^2 + x_2^2 \geq 0$

Limitations of linear models

- Logistic regression and other linear models cannot handle nonlinear decision boundaries
  - Must use non-linear feature transformations
  - Up to designer to specify which one

- Can we instead learn the transformation?
  - Yes, this is what neural networks do!

- A Neural network chains together many layers of "neurons" such as logistic units (logistic regression functions)

Neural Networks learn features

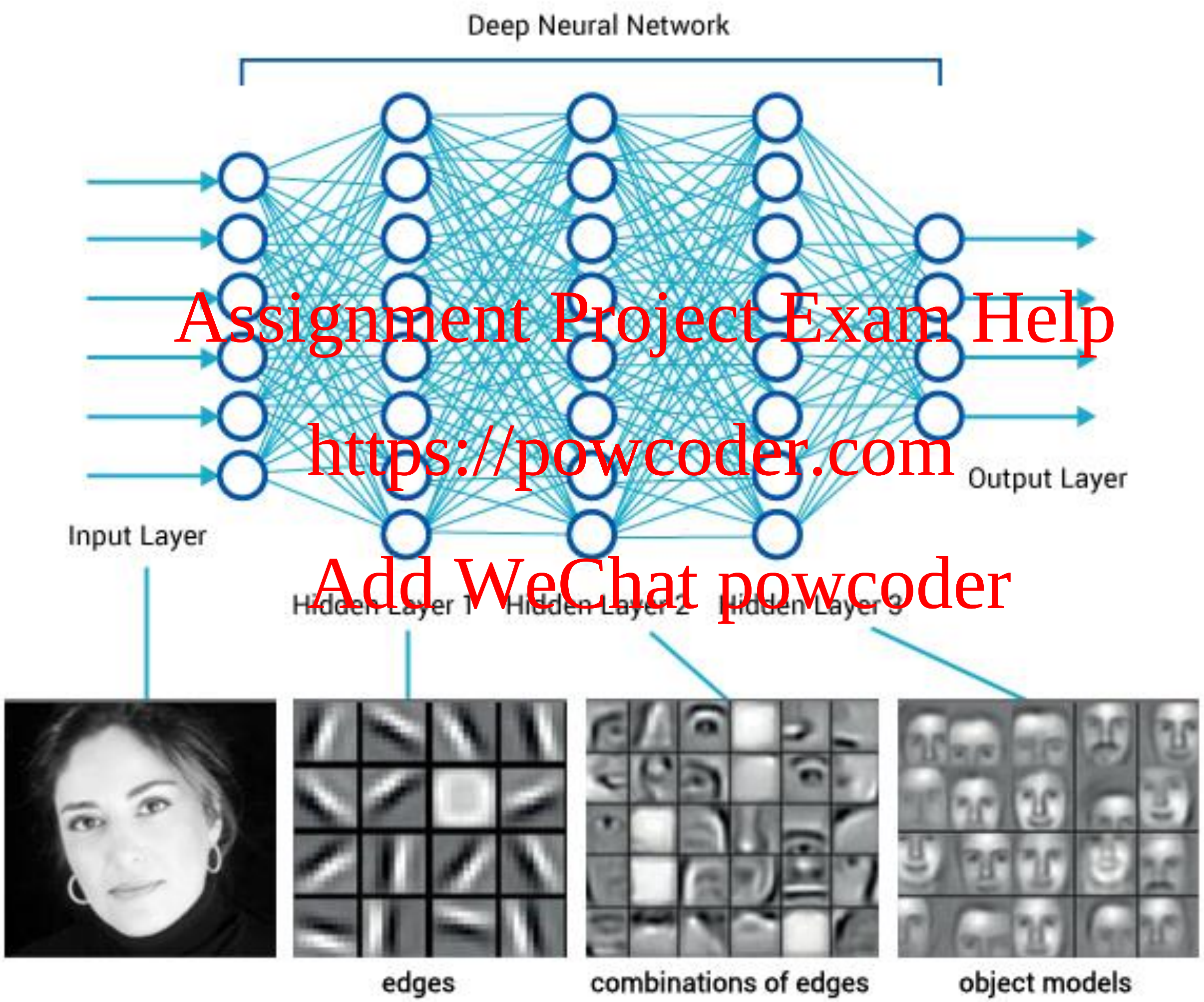
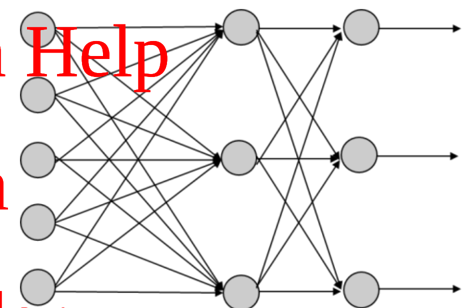


Image: http://www.amax.com/blog/wp-content/uploads/2015/12/blog_deeplearning3.jpg

Neurons in the Brain

Inspired "Artificial Neural Networks"

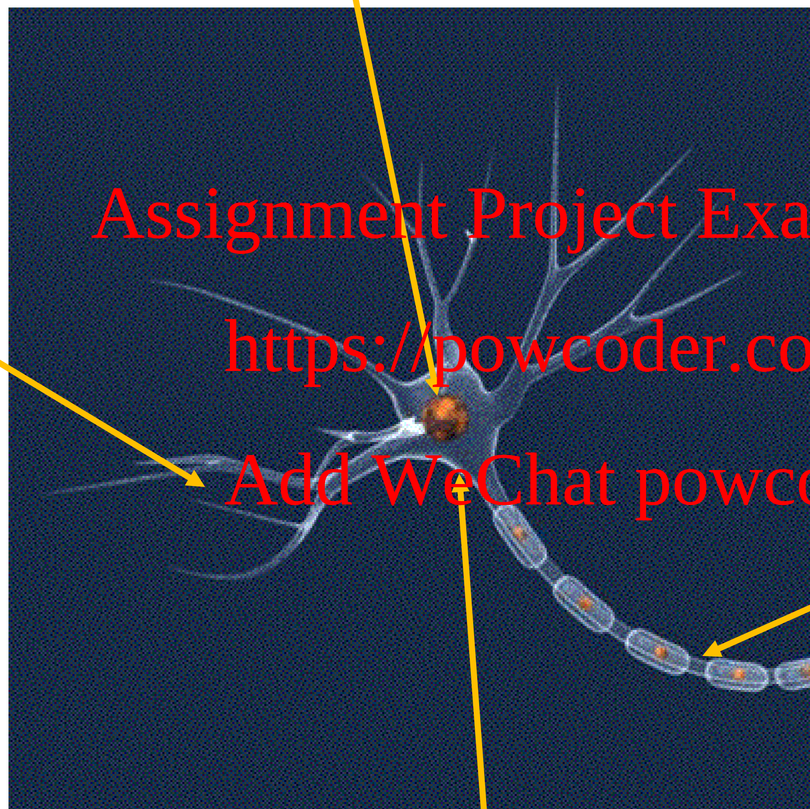


Neurons are cells that process chemical and electrical signals and transmit these signals to neurons and other types of cells

Neuron in the brain

nucleus

dendrites
“Input wire”



Can measure electrical activity (spike) of a single neuron by placing electrodes

axon
“Output wire”

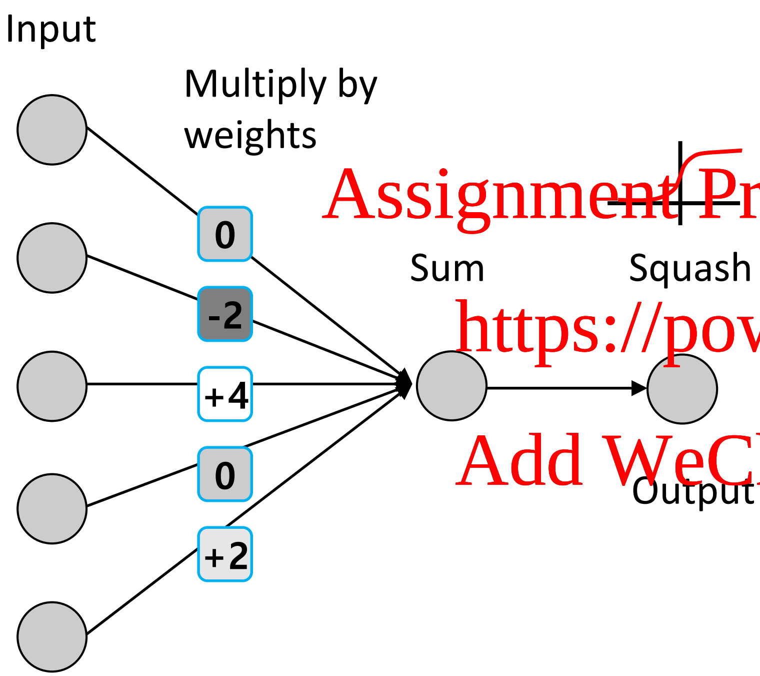
Cell body

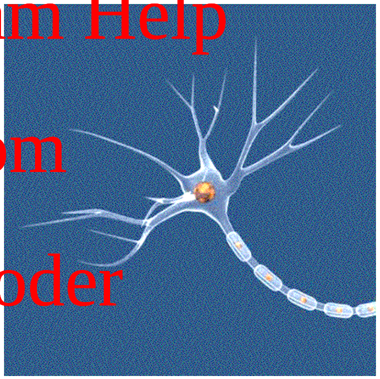
Image: http://webspace.ship.edu/cgboer/neuron.gif

Neural network in the brain

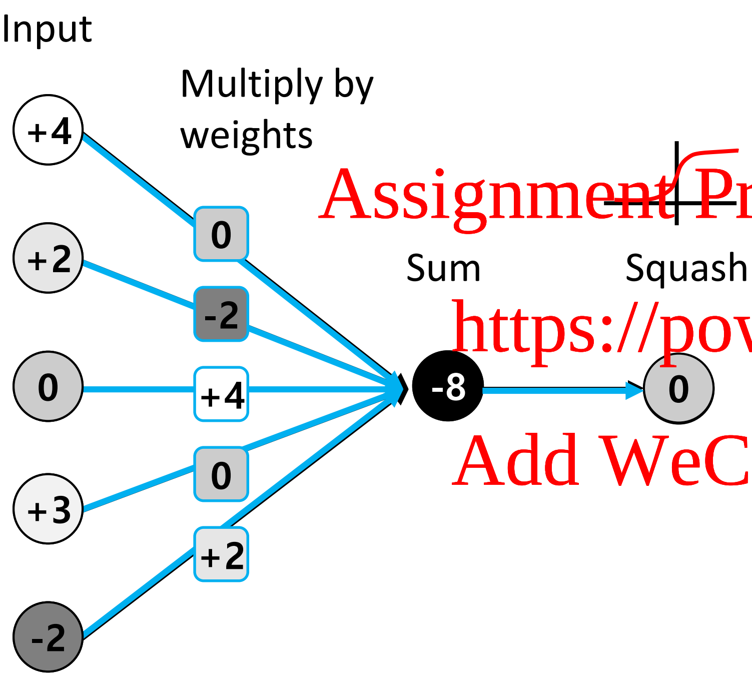
- **Micro networks**: several connected neurons perform sophisticated tasks: mediate reflexes, process sensory information, generate locomotion and mediate learning and memory.
- **Macro networks**: perform higher brain functions such as object recognition and cognition.

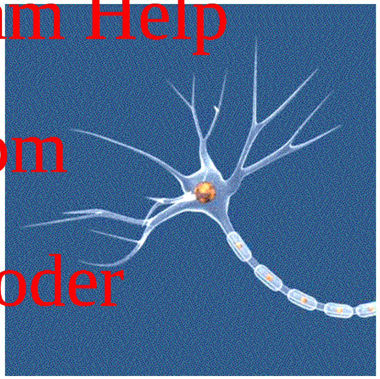
Logistic Unit as Artificial Neuron



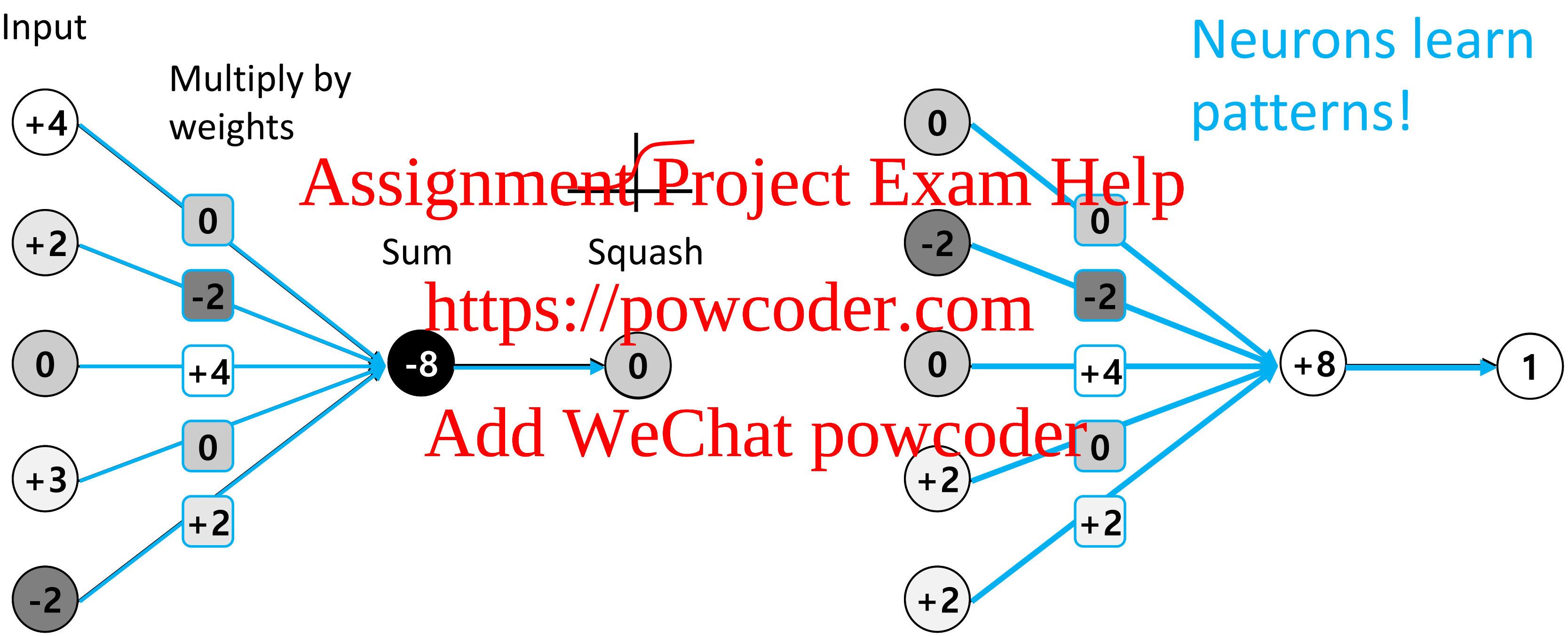


Logistic Unit as Artificial Neuron



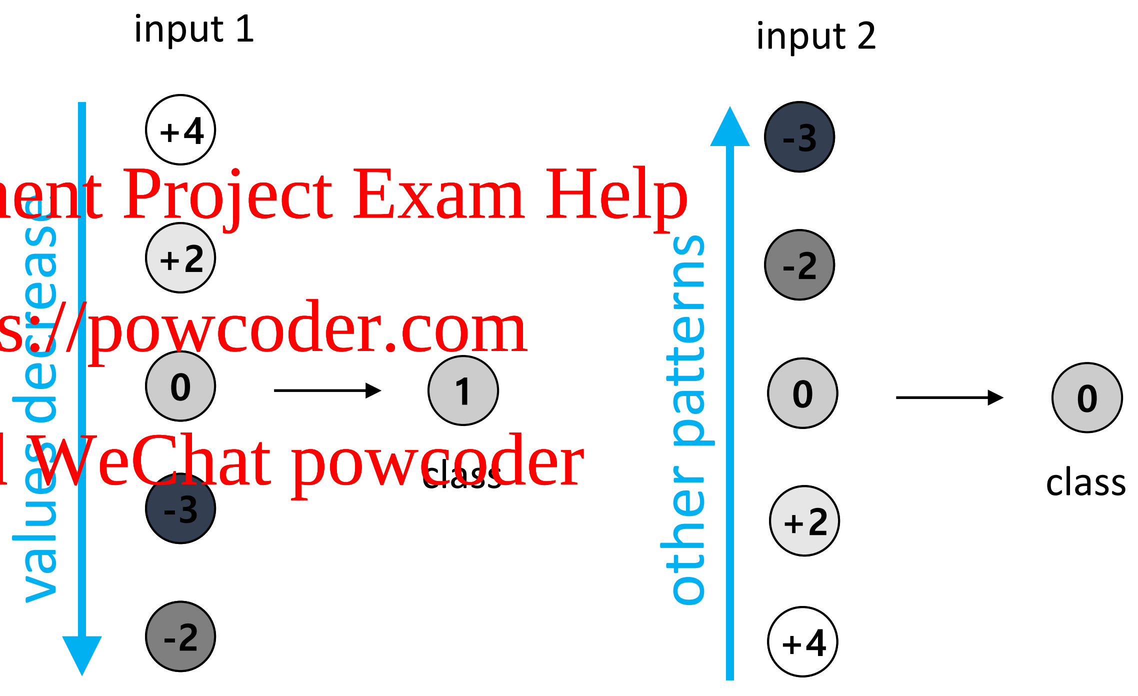


Logistic Unit as Artificial Neuron



Artificial Neuron Learns Patterns

- Classify input into class 0 or 1
- Teach neuron to predict correct class label
- Detect presence of a simple "feature"

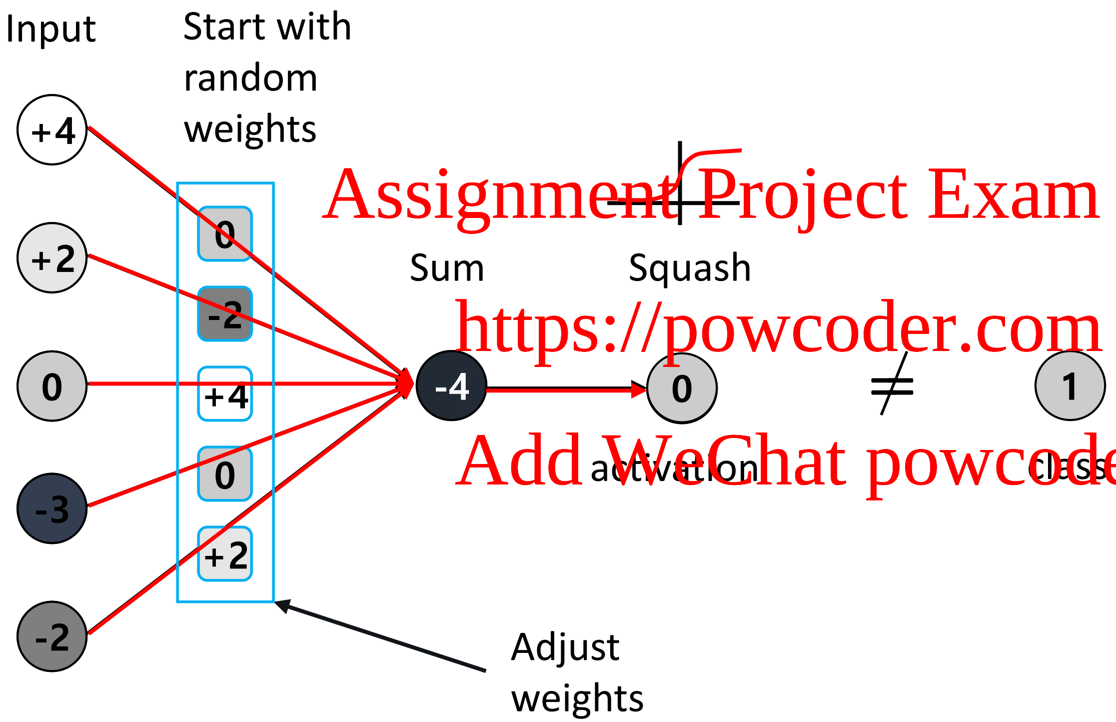


Example

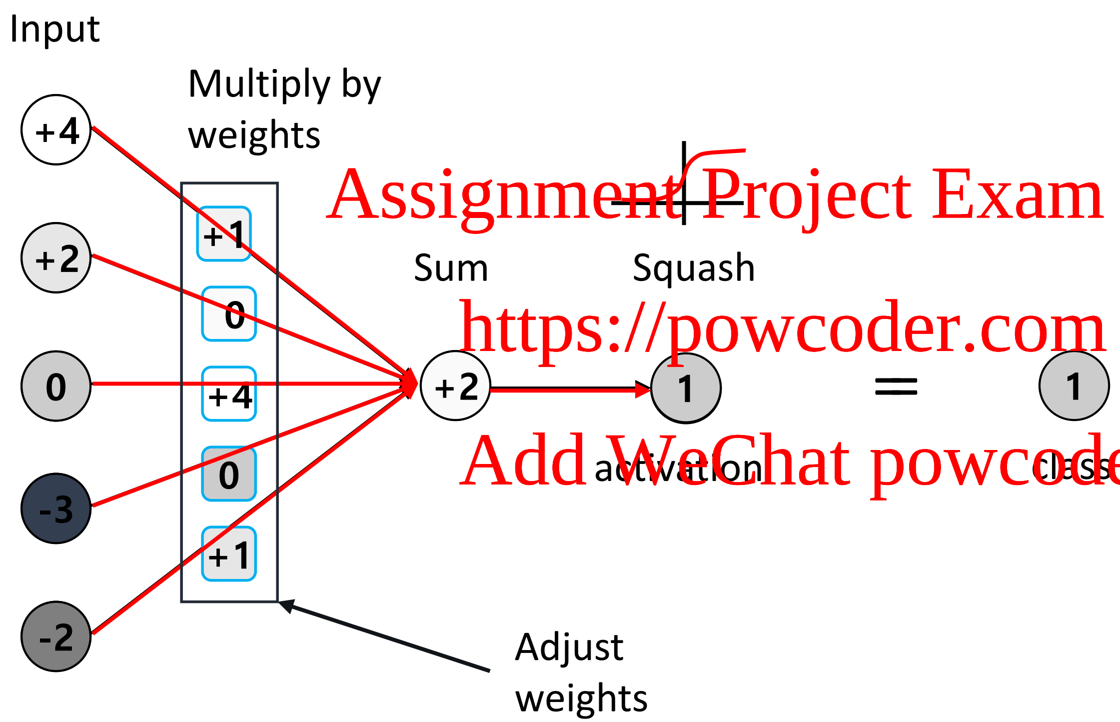
Neural Networks: Learning

Intuition

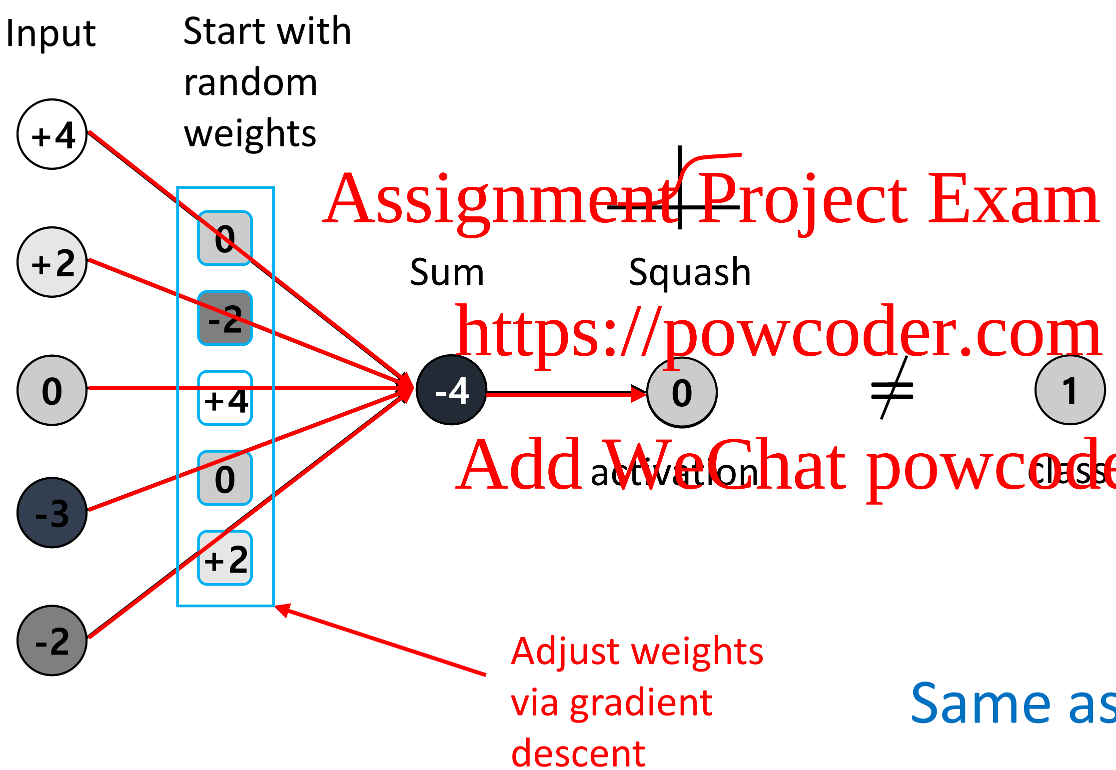
Artificial Neuron: Learning



Artificial Neuron: Learning



Artificial Neuron: Learning

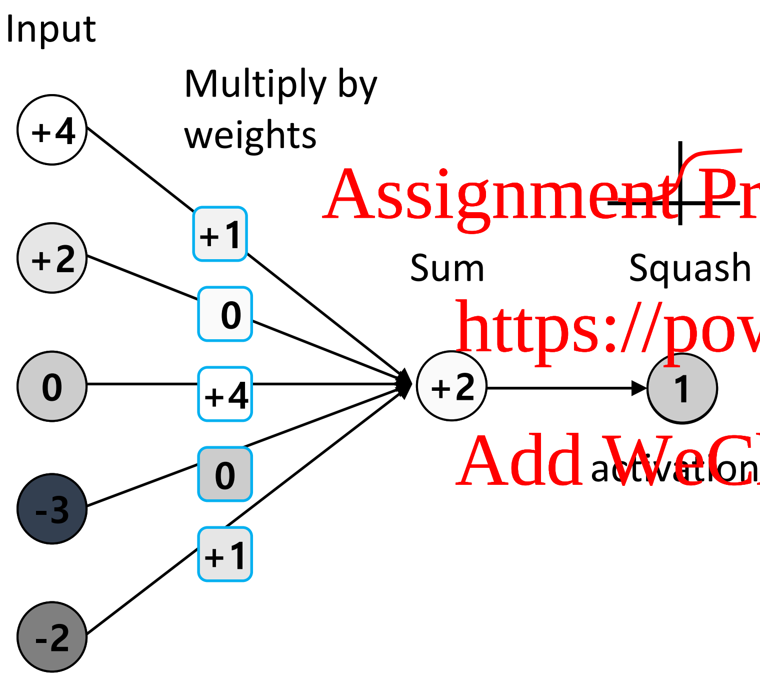


Same as in logistic regression

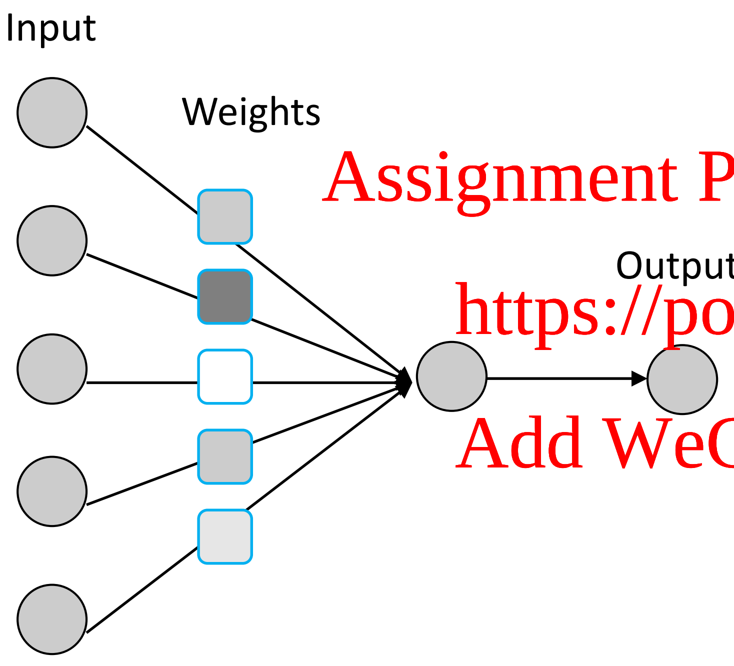
Neural Networks: Learning

Multi-layer network

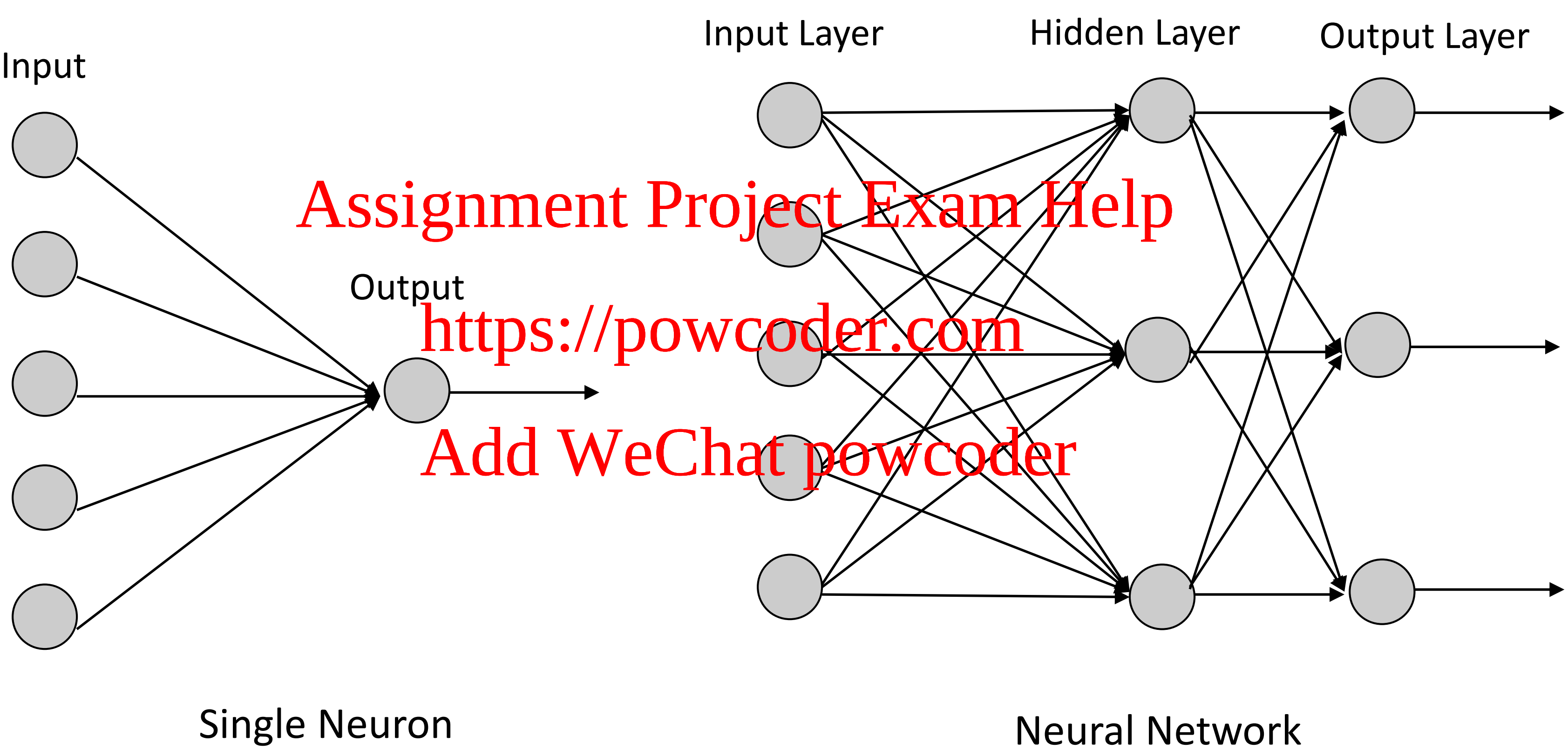
Artificial Neuron: simplify



Artificial Neuron: simplify



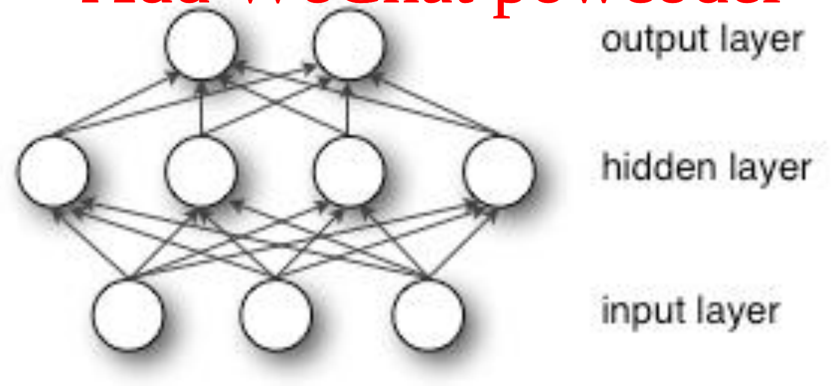
Artificial Neural Network



Deep Network: many hidden layers

Multi-layer perceptron (MLP)

- Just another name for a feed-forward neural network
- Logistic regression is a special case of the MLP with no hidden layer and sigmoid output

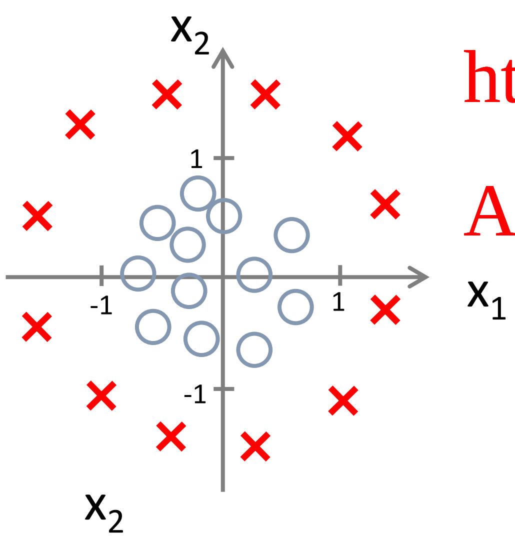


Neural Networks Learn Features

logistic regression unit == artificial neuron

chain several units together == neural network

“earlier” units learn non-linear feature transformation

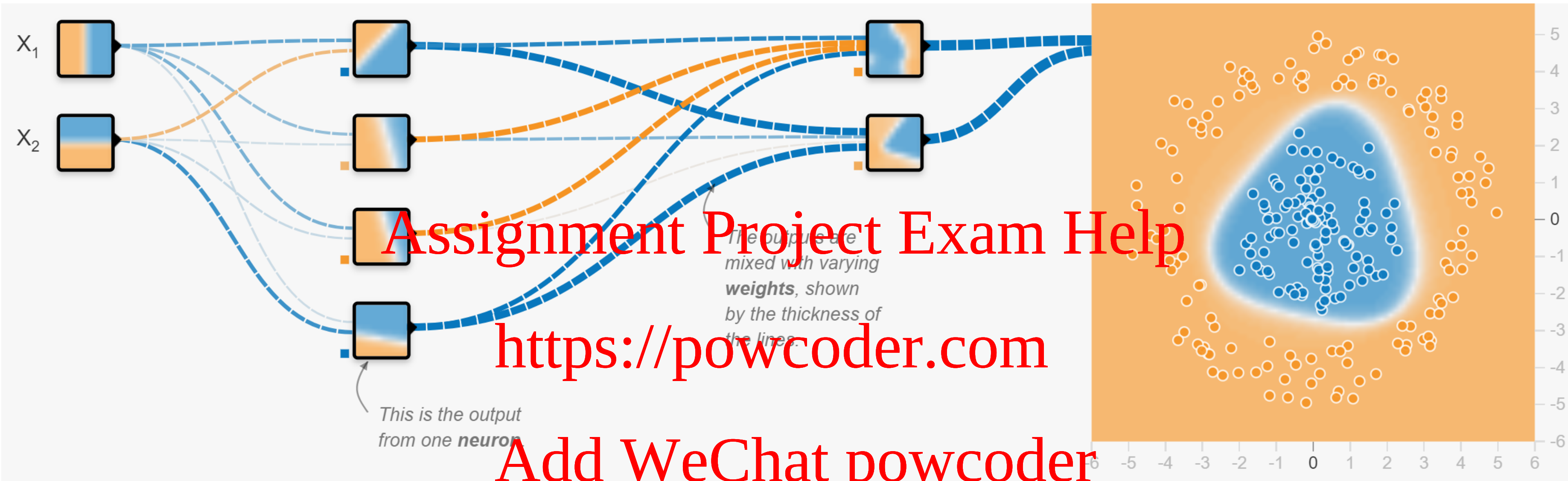


$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$$

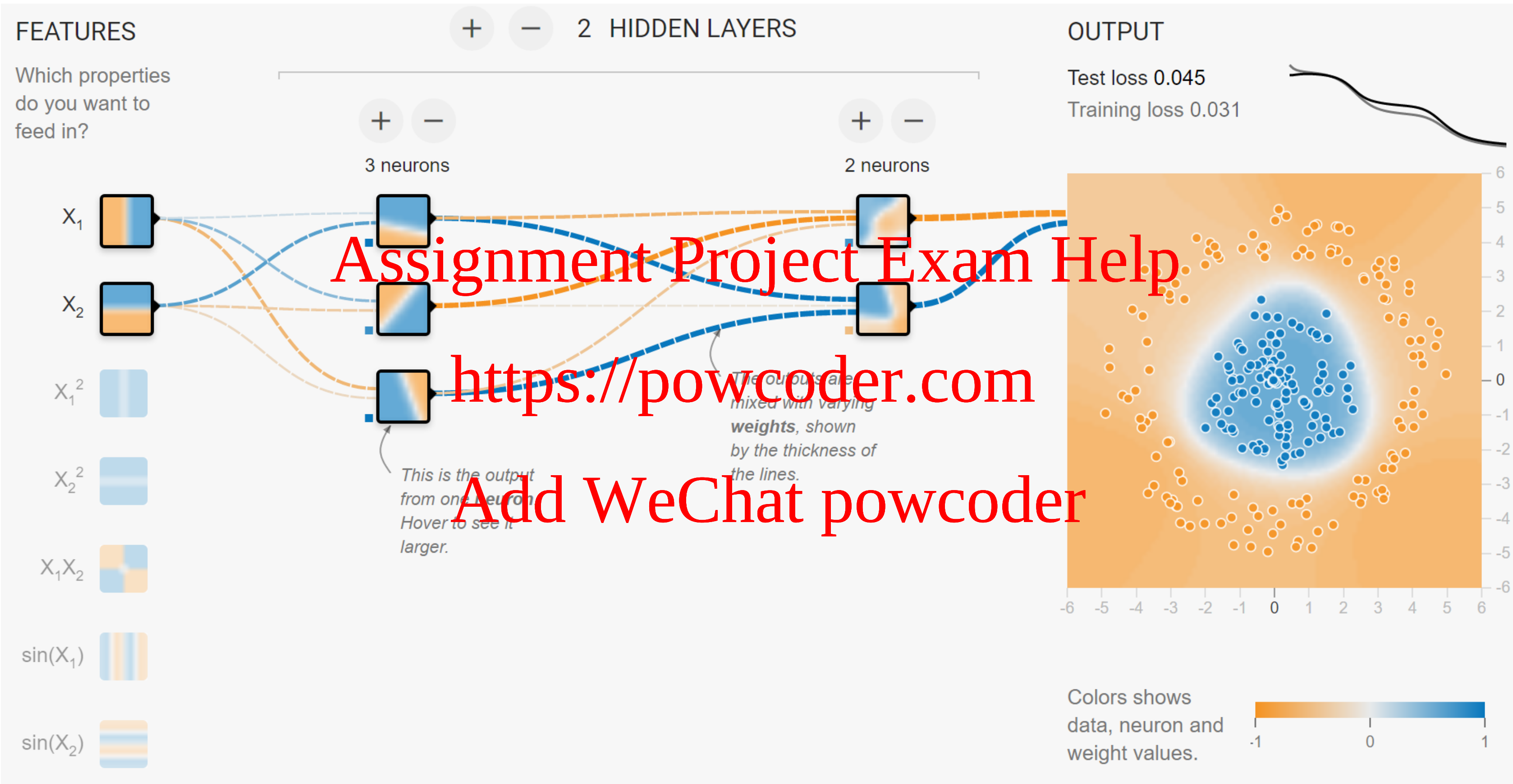
simple neural network

$$h(x) = g(\theta + \theta_1 h^{(1)}(x) + \theta_2 h^{(2)}(x) + \theta_3 h^{(3)}(x))$$

Example


X1
X2
This is the output from one neuron.
The outputs are mixed with varying weights, shown by the thickness of the lines.
5 4 3 2 1 0 -1 -2 -3 -4 -5 -6
-6 -5 -4 -3 -2 -1 0 1 2 3 4 5 6

Training a neural net: Demo



[Tensorflow playground]

Artificial Neural Network:

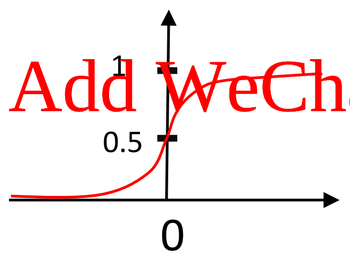
general notation

input $x = \begin{bmatrix} x_1 \\ \cdots \\ x_5 \end{bmatrix}$

hidden layer activations

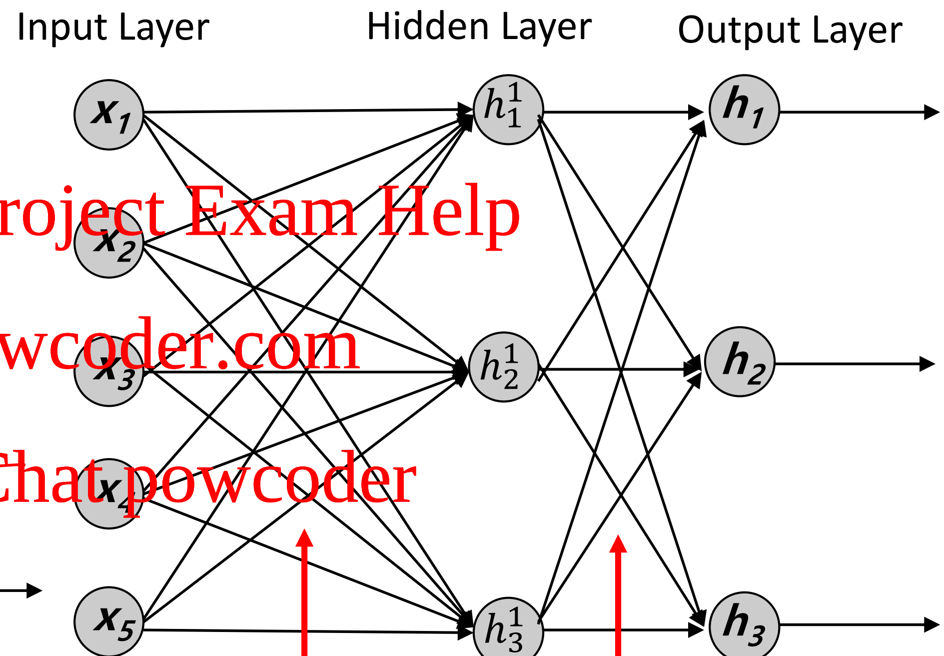
$$h^i = g(\Theta^{(i)}x)$$

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



output

$$h_\Theta(\mathrm{x}) = g(\Theta^{(2)}a)$$



weights $\Theta^{(1)} = \begin{pmatrix} \theta_{11} & \cdots & \theta_{15} \\ \vdots & \ddots & \vdots \\ \theta_{31} & \cdots & \theta_{35} \end{pmatrix}$ $\Theta^{(2)} = \begin{pmatrix} \theta_{11} & \cdots & \theta_{13} \\ \vdots & \ddots & \vdots \\ \theta_{31} & \cdots & \theta_{33} \end{pmatrix}$

Cost function

Neural network: $h_\Theta(x) \in \mathbb{R}^K \quad (h_\Theta(x))_i = i^{th} \text{ output}$

training error

$$J(\Theta) = \boxed{-\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log(h_\Theta(x^{(i)}))_k + (1 - y_k^{(i)}) \log(1 - (h_\Theta(x^{(i)}))_k)\right]}$$

$$\boxed{+\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_{ji}^{(l)})^2}$$

regularization

Gradient computation

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log h_\theta(x^{(i)})_k + (1-y_k^{(i)})\log(1-h_\theta(x^{(i)})_k)\right]$$

$$+\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_{ji}^{(l)})^2$$

Cover next time!

$$\min_{\Theta} J(\Theta)$$

Need code to compute:

- $J(\Theta)$
- $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$

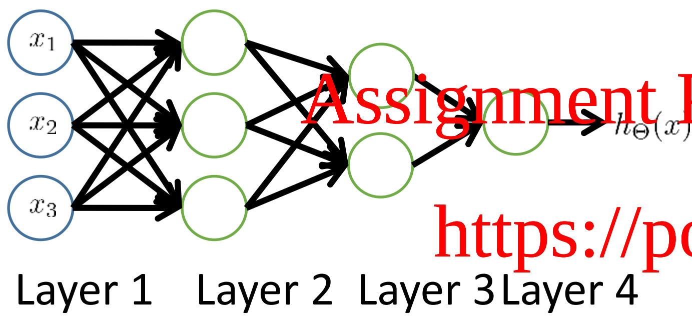
Use "Backpropagation algorithm"

- Efficient way to compute $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$
- Computes gradient incrementally by "propagating" backwards through the network

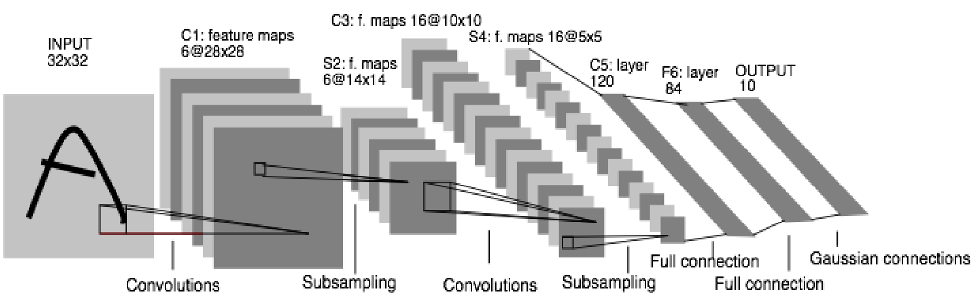
Network architectures

Feed-forward

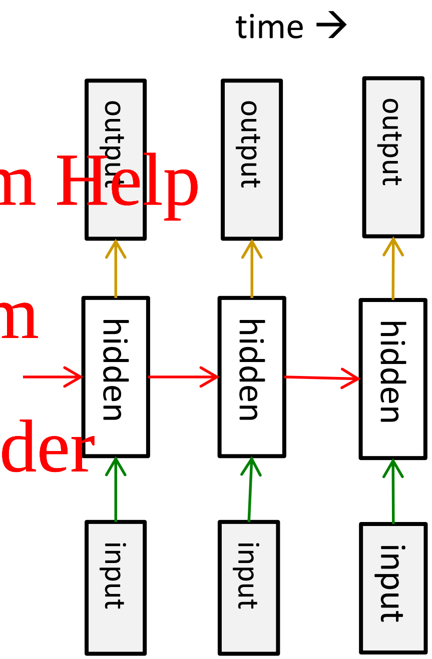
Fully connected



Convolutional

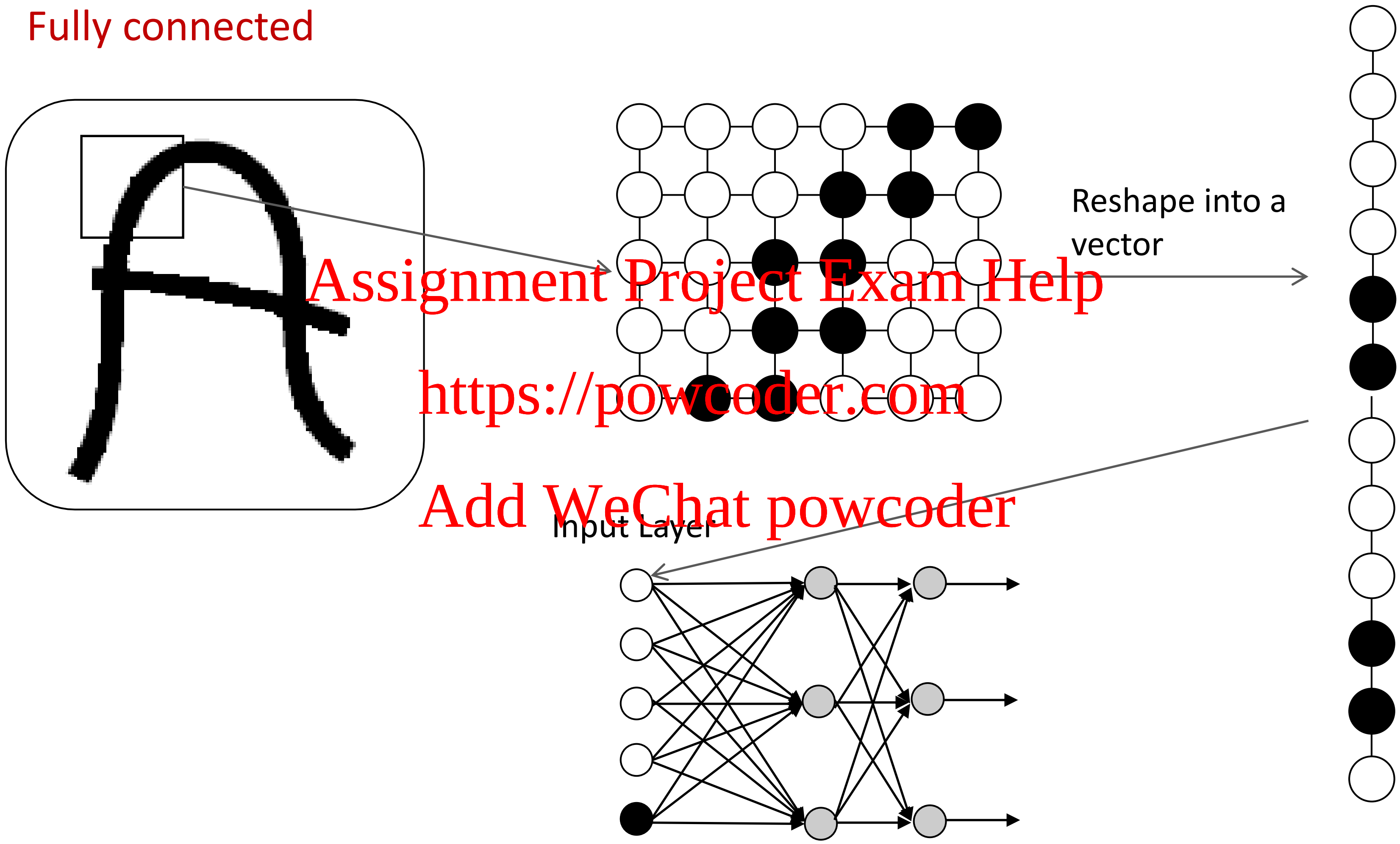


Recurrent



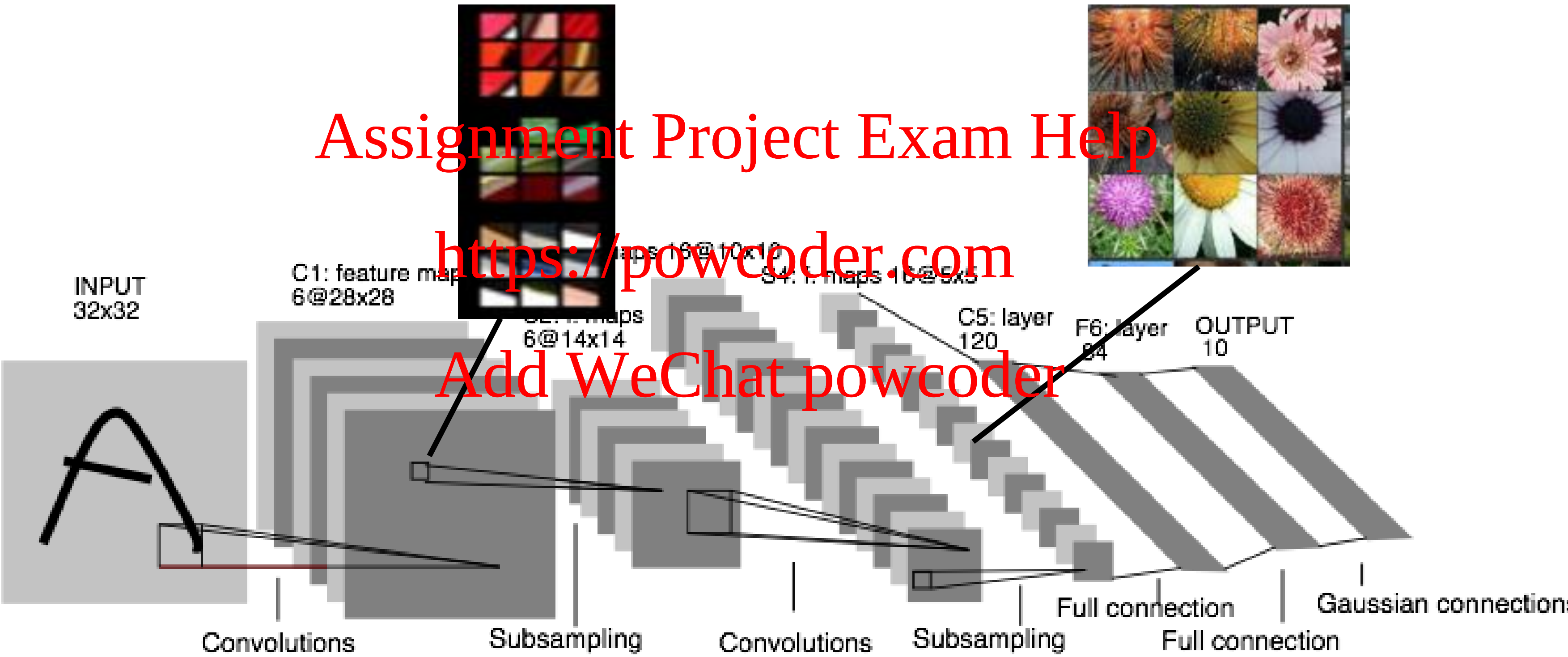
Representing images

Fully connected



Convolutional Neural Network

A better architecture for 2d signals



LeNet

Why Deep Learning?

The Unreasonable Effectiveness of Deep Features

Maximal activations of $pool_5$ units [R-CNN]

$conv_5$ DeConv visualization [Zeiler-Fergus]

Rich visual structure of features deep in hierarchy.

Summary so far

- Neural network chains together many layers of “neurons” such as logistic units

- Hidden neurons learn more and more abstract non-linear features

Next Class

Neural Networks I: Learning:

Learning via gradient descent; computation graphs, backpropagation algorithm.

Reading: Bishop Ch 5.1-5.3