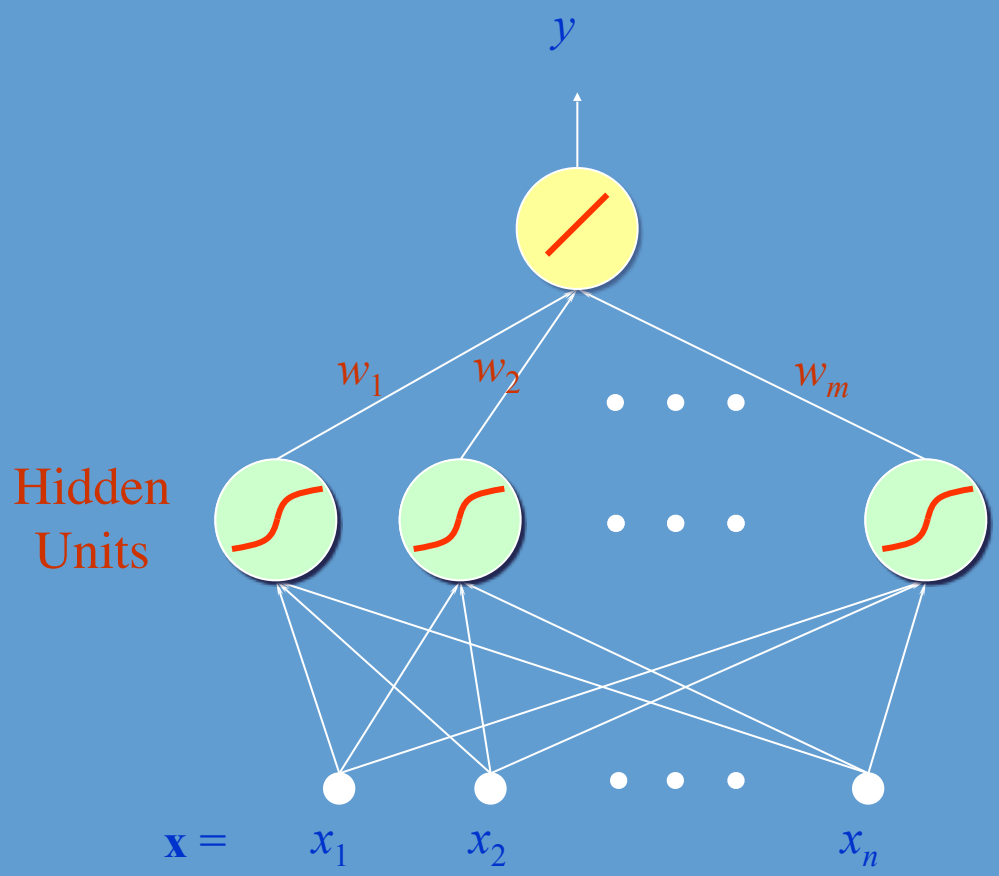
ML-2

More loosely Deep Learning

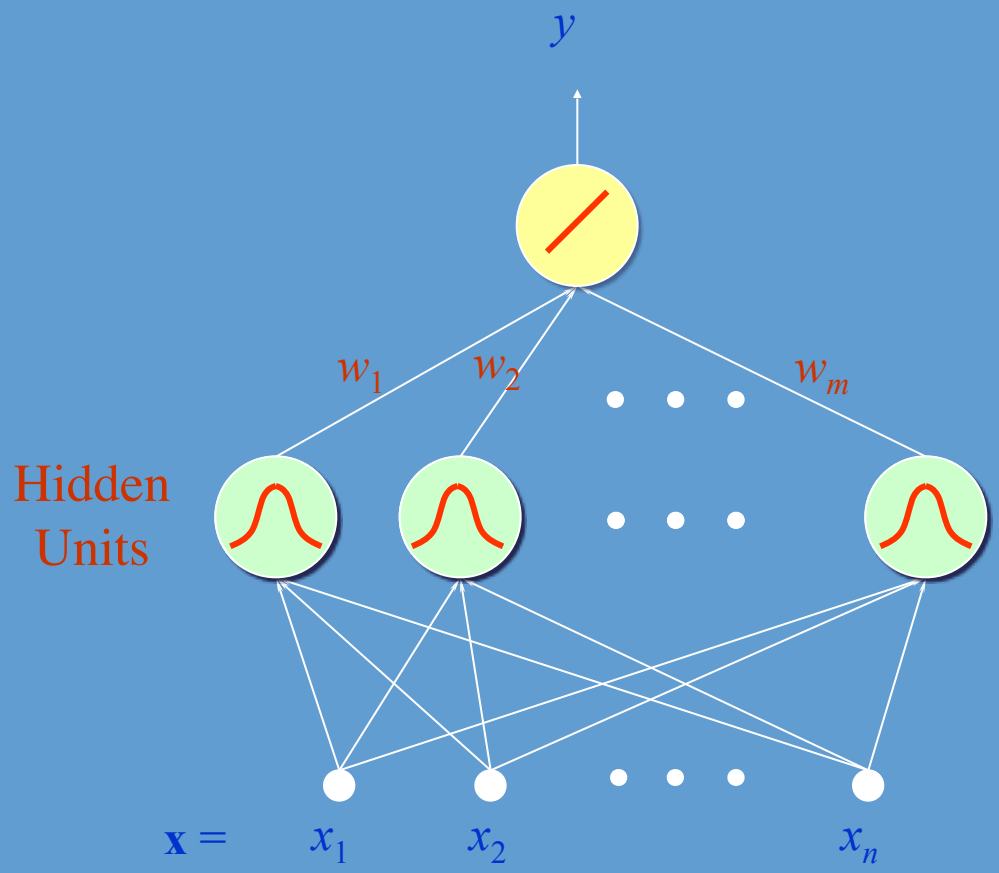
Mayank Vatsa

Single-Hidden Layer NNet



Radial Basis Function Networks

$$f(\mathbf{x}) = \sum_{i=1}^{m} w_i \phi_i(\mathbf{x})$$



Non-Linear Models

$$f(\mathbf{x}) = \sum_{i=1}^{m} w_i \phi_i(\mathbf{x})$$

$$f(\mathbf{x}) = \sum_{i=1}^{m} \overbrace{w_i}^{\text{Weights}} \underbrace{\phi_i(\mathbf{x})}_{\text{Adjusted by the Learning process}}$$

Typical Radial Functions

- Gaussian

$$\phi(r) = e^{-\frac{r^2}{2\sigma^2}} \quad \sigma > 0 \quad \text{and} \quad r \in \Re$$

- Hardy Multiquadratic

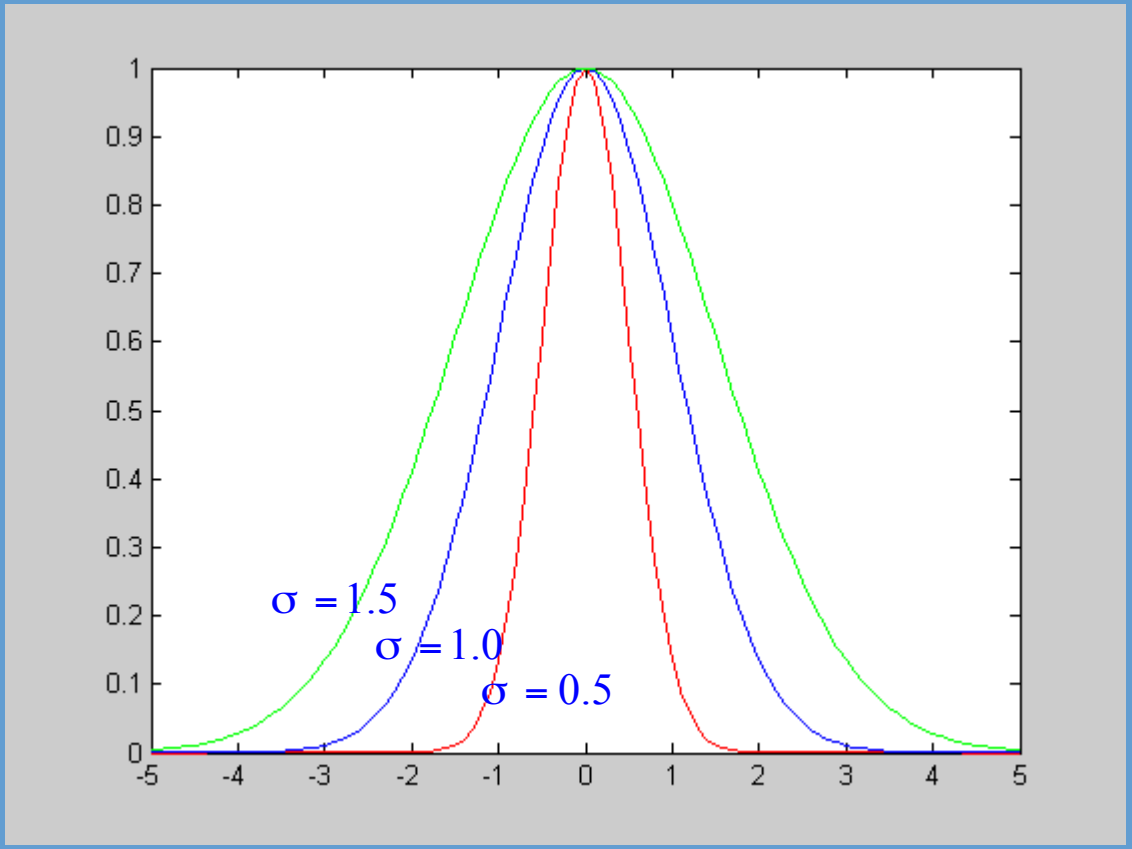
$$\phi(r) = \sqrt{r^2 + c^2}\,/\,c \quad c > 0 \quad \text{and} \quad r \in \Re$$

- Inverse Multiquadratic

$$\phi(r) = c\,/\,\sqrt{r^2 + c^2} \quad c > 0 \quad \text{and} \quad r \in \Re$$

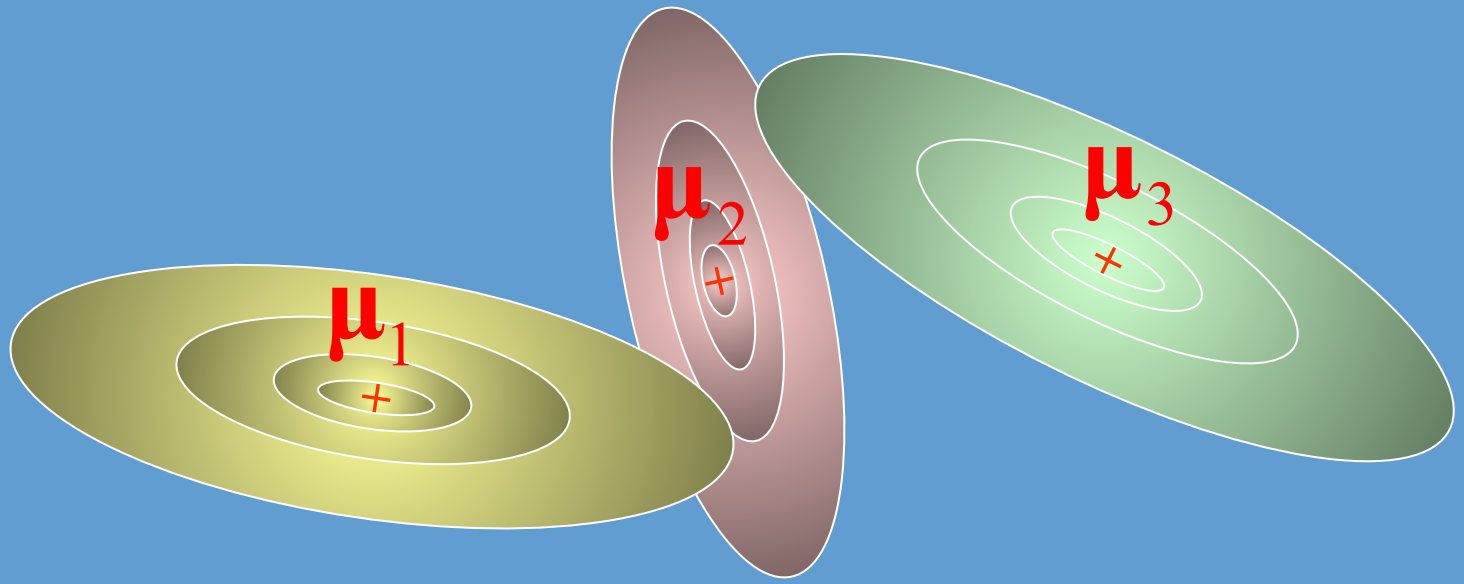
Gaussian Basis Function (σ=0.5,1.0,1.5)

$$\phi(r) = e^{-\frac{r^2}{2\sigma^2}} \quad \sigma > 0 \quad \text{and} \quad r \in \Re$$

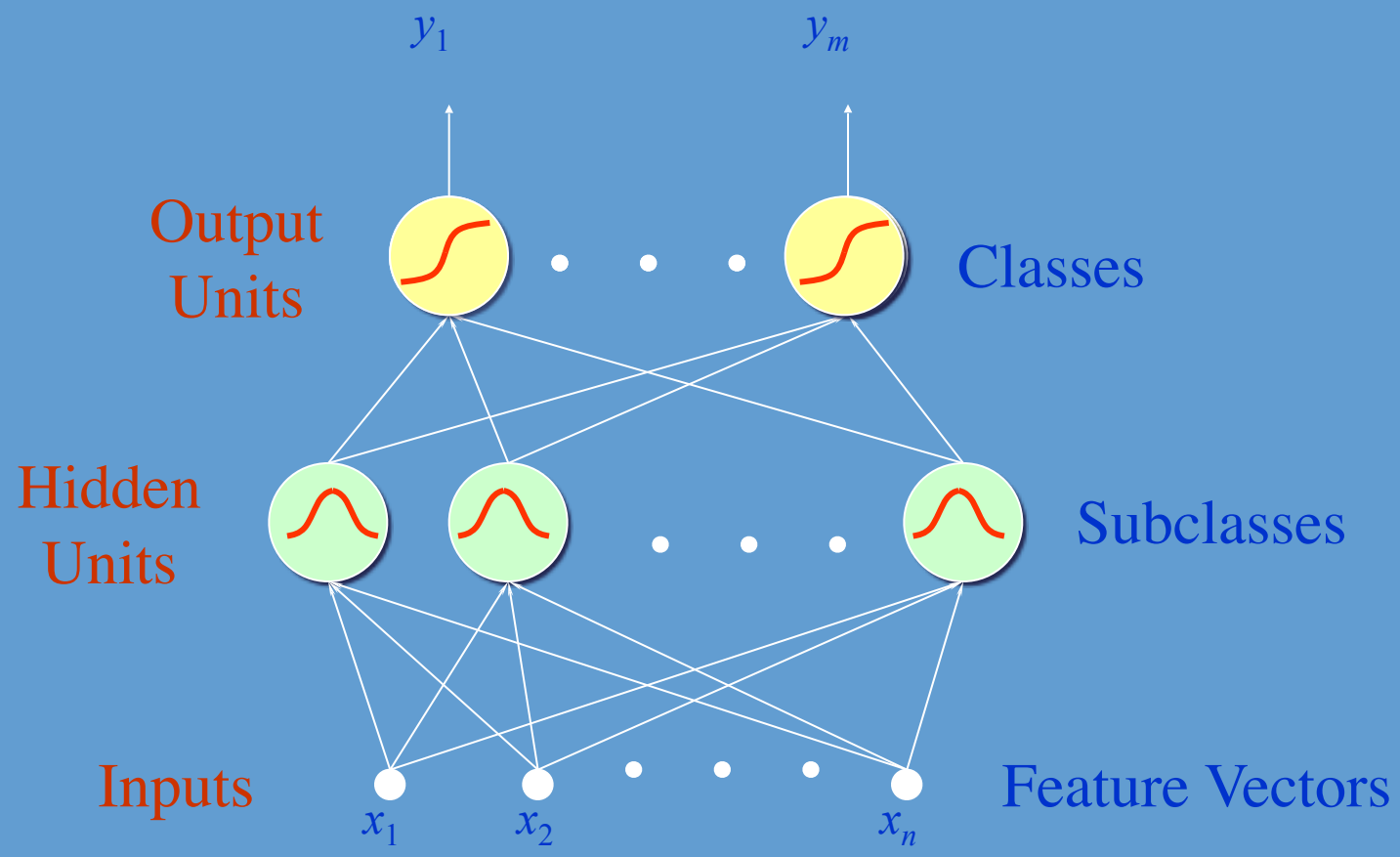


Most General RBF

$$\phi_i(\mathbf{x}) = \phi\left((\mathbf{x}-\boldsymbol{\mu}_i)^T \Sigma^{-1} (\mathbf{x}-\boldsymbol{\mu}_i)\right)$$



The Topology of RBF NNet



Radial Basis Function Networks

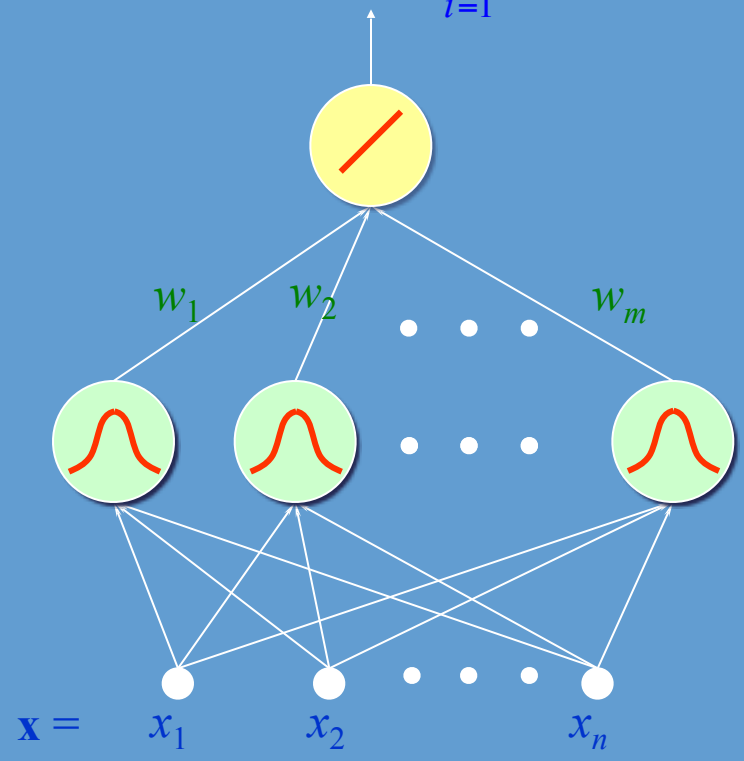
Training set $\mathcal{T} = \left\{\left(\mathbf{x}^{(k)}, y^{(k)}\right)\right\}_{k=1}^{p}$

$$y = f(\mathbf{x}) = \sum_{i=1}^{m} w_i \phi_i(\mathbf{x})$$

Goal $y^{(k)} \approx f\left(\mathbf{x}^{(k)}\right)$ for all k

$$\min E = \frac{1}{2}\sum_{k=1}^{p}\left[y^{(k)} - f\left(\mathbf{x}^{(k)}\right)\right]^2$$

$$= \frac{1}{2}\sum_{k=1}^{p}\left[y^{(k)} - \sum_{i=1}^{m} w_i \phi_i\left(\mathbf{x}^{(k)}\right)\right]^2$$



Learn the Optimal Weight Vector

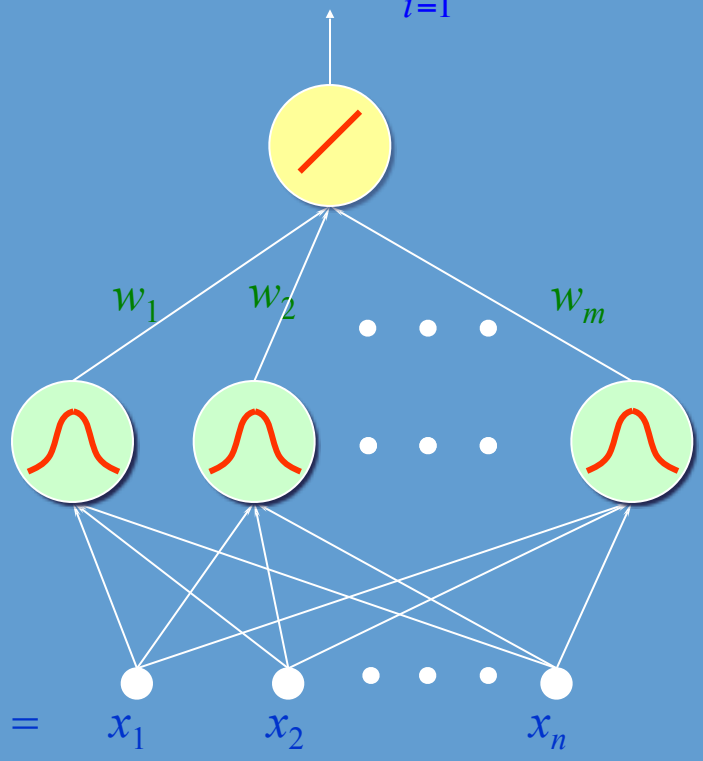
Training set $T = \left\{\left(\mathbf{x}^{(k)}, y^{(k)}\right)\right\}_{k=1}^{p}$

$$y = f(\mathbf{x}) = \sum_{i=1}^{m} w_i \phi_i(\mathbf{x})$$

Goal $y^{(k)} \approx f\left(\mathbf{x}^{(k)}\right)$ for all k

$$\min E = \frac{1}{2}\sum_{k=1}^{p}\left[y^{(k)} - f\left(\mathbf{x}^{(k)}\right)\right]^2$$

$$= \frac{1}{2}\sum_{k=1}^{p}\left[y^{(k)} - \sum_{i=1}^{m} w_i \phi_i\left(\mathbf{x}^{(k)}\right)\right]^2$$



Regularization

$$\lambda_i \geq 0$$

Training set $T = \left\{ \left(\mathbf{x}^{(k)}, y^{(k)} \right) \right\}_{k=1}^{p}$

If regularization is not needed, set $\lambda_i = 0$

Goal $y^{(k)} \approx f\left(\mathbf{x}^{(k)}\right)$ for all k

$$\min C = \frac{1}{2} \sum_{k=1}^{p} \left[y^{(k)} - f\left(\mathbf{x}^{(k)}\right) \right]^2 + \frac{1}{2} \sum_{i=1}^{m} \lambda_i w_i^2$$

$$= \frac{1}{2} \sum_{k=1}^{p} \left[y^{(k)} - \sum_{i=1}^{m} w_i \phi_i\left(\mathbf{x}^{(k)}\right) \right]^2 + \frac{1}{2} \sum_{i=1}^{m} \lambda_i w_i^2$$

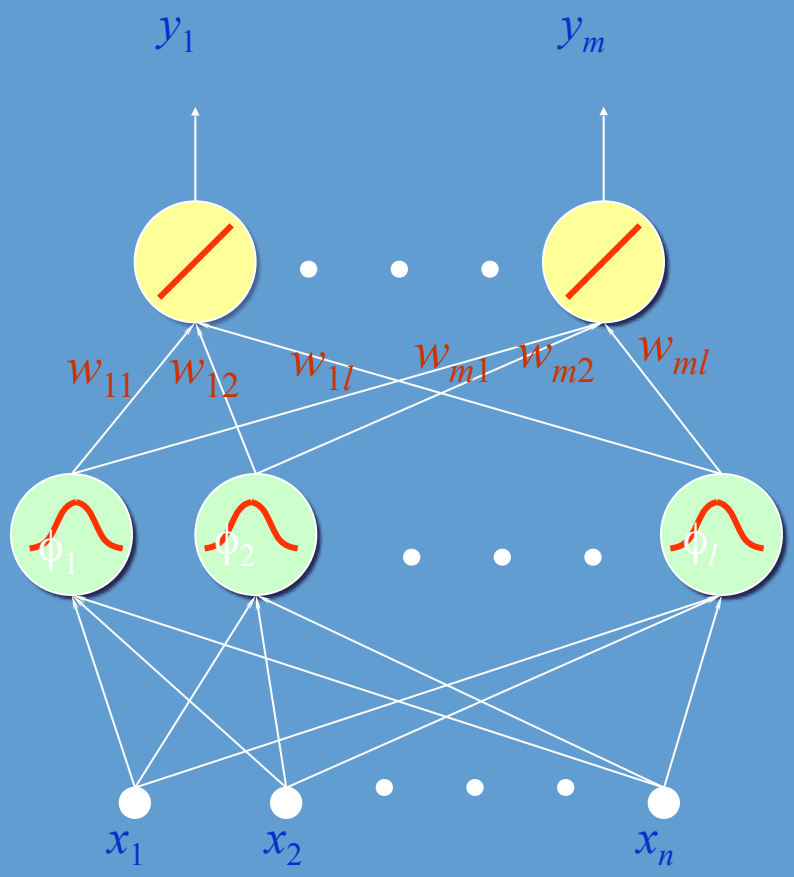
Learn the Optimal Weight Vector

$$f(\mathbf{x}) = \sum_{i=1}^{m} w_i \phi_i(\mathbf{x})$$

Minimize $$C = \frac{1}{2}\sum_{k=1}^{p}\left[y^{(k)} - f\left(\mathbf{x}^{(k)}\right)\right]^2 + \frac{1}{2}\sum_{i=1}^{m}\lambda_i w_i^2$$

$$0 = \frac{\partial C}{\partial w_j} = -\sum_{k=1}^{p}\left[y^{(k)} - f\left(\mathbf{x}^{(k)}\right)\right]\frac{\partial f\left(\mathbf{x}^{(k)}\right)}{\partial w_j} + \lambda_j w_j$$

Learning Kernel Parameters



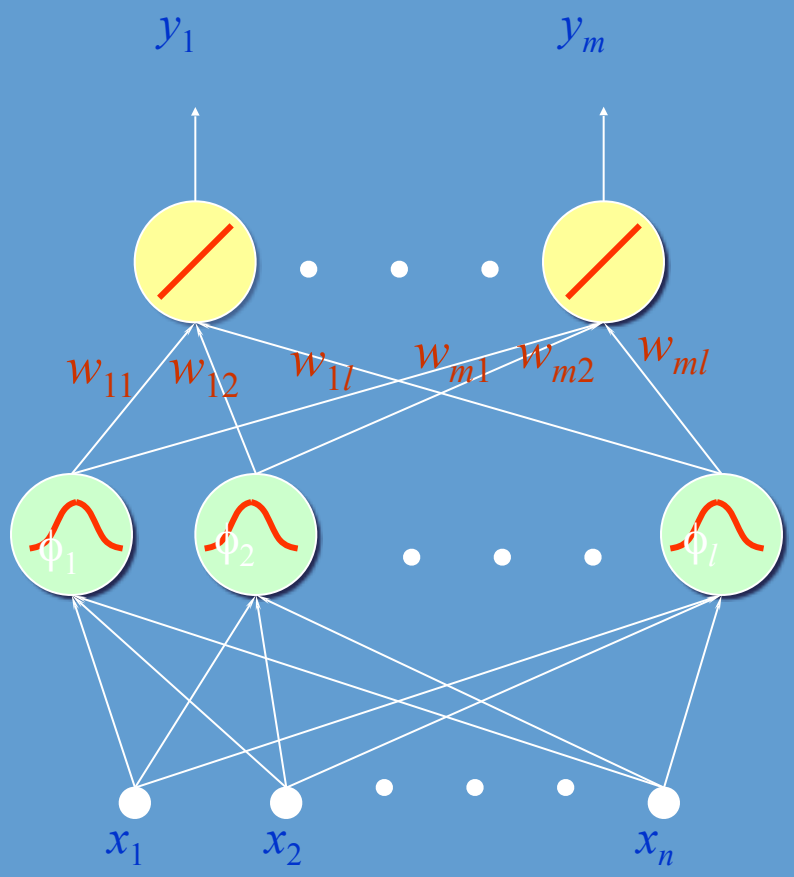
Training set

$$\mathcal{T} = \left\{\left(\mathbf{x}^{(k)}, \mathbf{y}^{(k)}\right)\right\}_{k=1}^{p}$$

Kernels

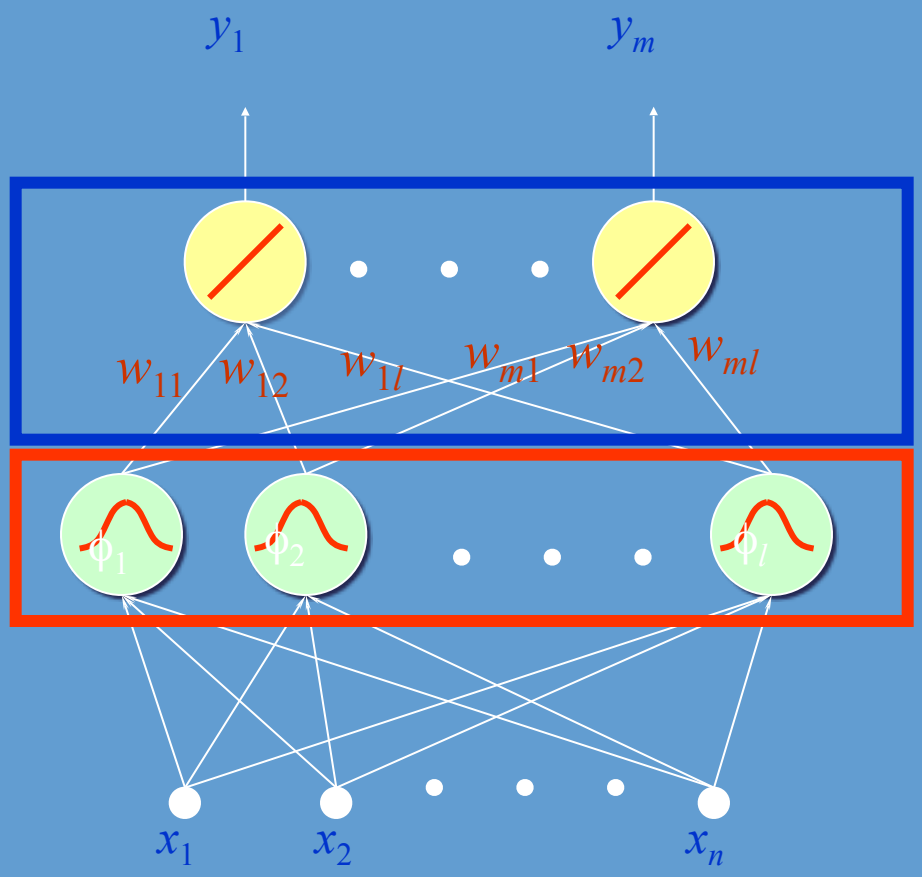
$$\phi_j(\mathbf{x}) = \exp\left[-\frac{\|\mathbf{x} - \quad_j\|^2}{2\sigma_j^2}\right]$$

What to Learn?



- Weights w_{ij}'s
- Centers μ_j's of ϕ_j's
- Widths σ_j's of ϕ_j's

Two-Stage Training



Step 2

Determines w_{ij}'s.

Step 1

Determines

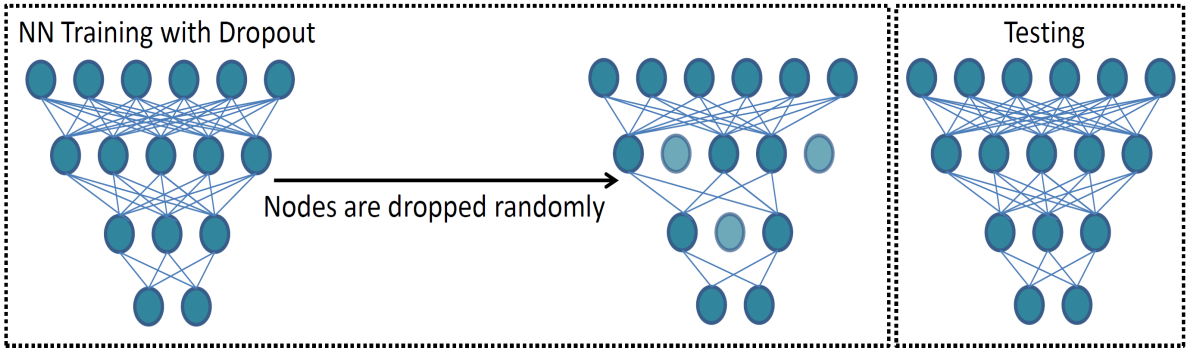
- Centers μ_j's of ϕ_j's.
- Widths σ_j's of ϕ_j's.

ANY OTHER LOSS FUNCTION?

$$H(y, p) = -\sum_i y_i log(p_i)$$

$$-(y \log(p) + (1 - y) \log(1 - p))$$

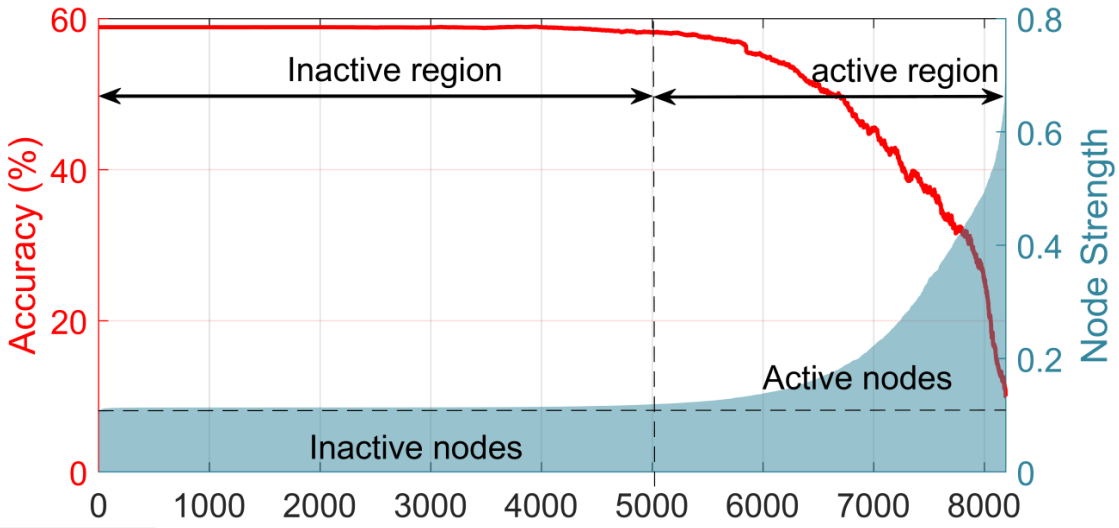
Dropout



- Randomly drops nodes at every iteration of the training.
- Trained model behave as an ensemble of multiple models.

Srivastava, Nitish, et al. "Dropout: a simple way to prevent neural networks from overfitting." *The Journal of Machine Learning Research* 15.1 (2014): 1929-1958.

Active/Inactive region in NN



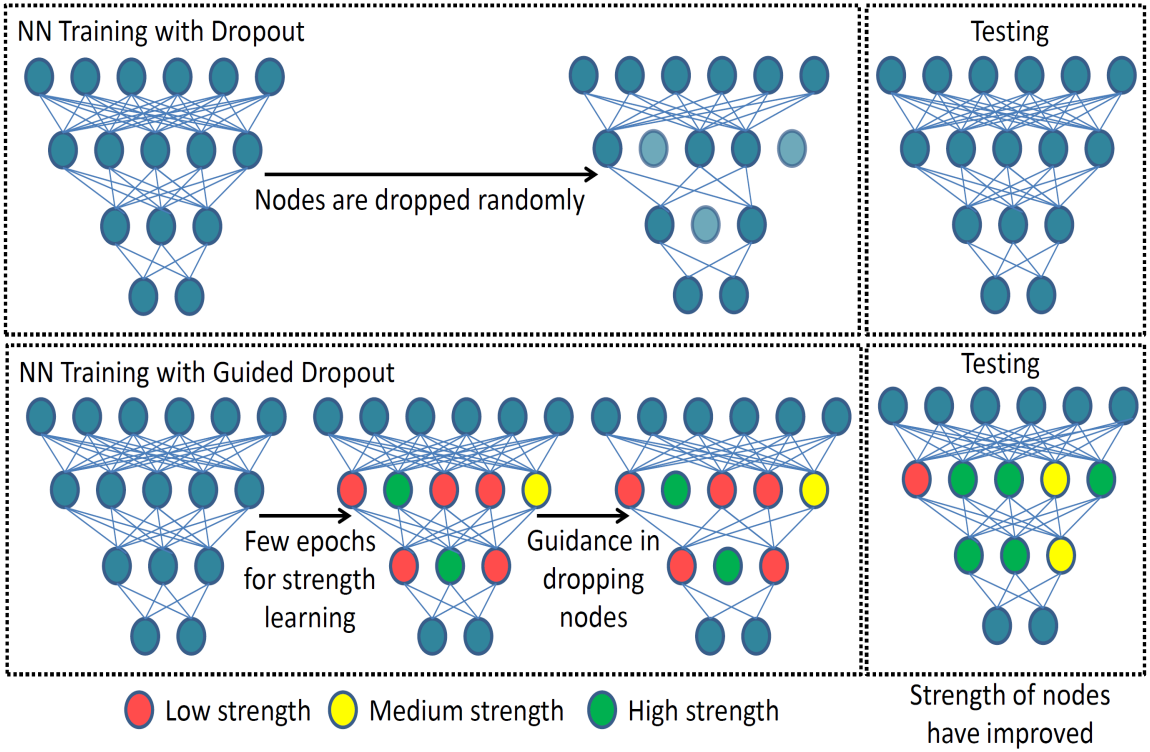
Number of drop nodes in ascending order

Our Work on Guided Dropout

- Our hypothesis is that in absence of high strength nodes during training, low strength nodes can improve their strength and contribute to the performance of NN.

- To achieve this, while training a NN, we drop the high strength nodes in the active region and learn the network with low strength nodes. This is termed as *Guided Dropout*.

Proposed Guided Dropout



From Supervised Classification to Unsupervised feature learning

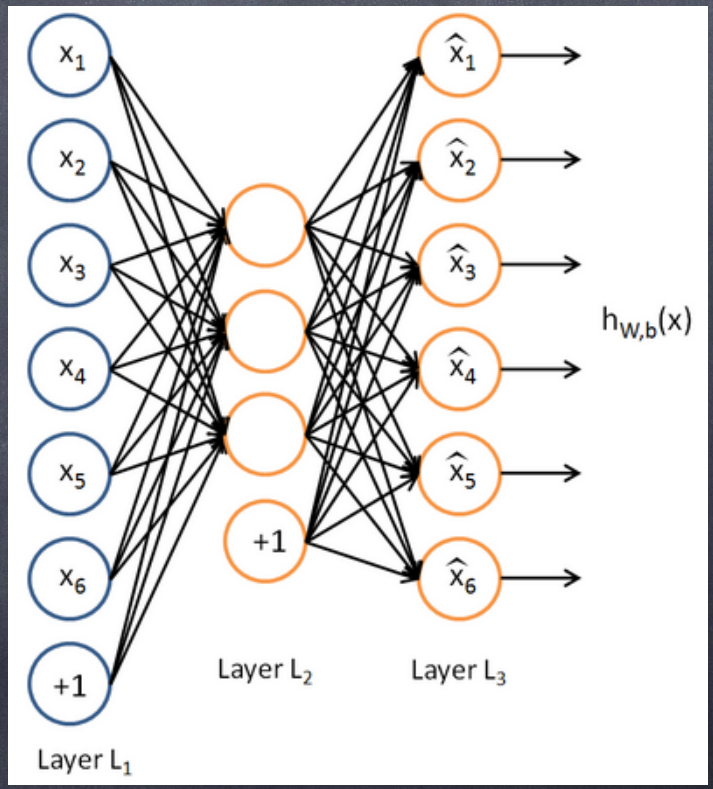
- Traditional neural network, including CNN: supervised approach for classification
- Different layers in the neural network can be viewed as "feature representation" of input data

From Supervised Classification to Unsupervised feature learning

- If input is raw data (huge amount of data available for training) and we "magically" allow our network to learn the "features" in an unsupervised manner

Let us take a look at simple Deep Architecture

- Auto-encoders
- Two parts: encoding and decoding
- Input layer: raw data X
- One hidden layer (encoding): feature learner
- Output layer (decoding): reconstruct $\hat{X}$ such that $||X-\hat{X}||^2$ is minimum

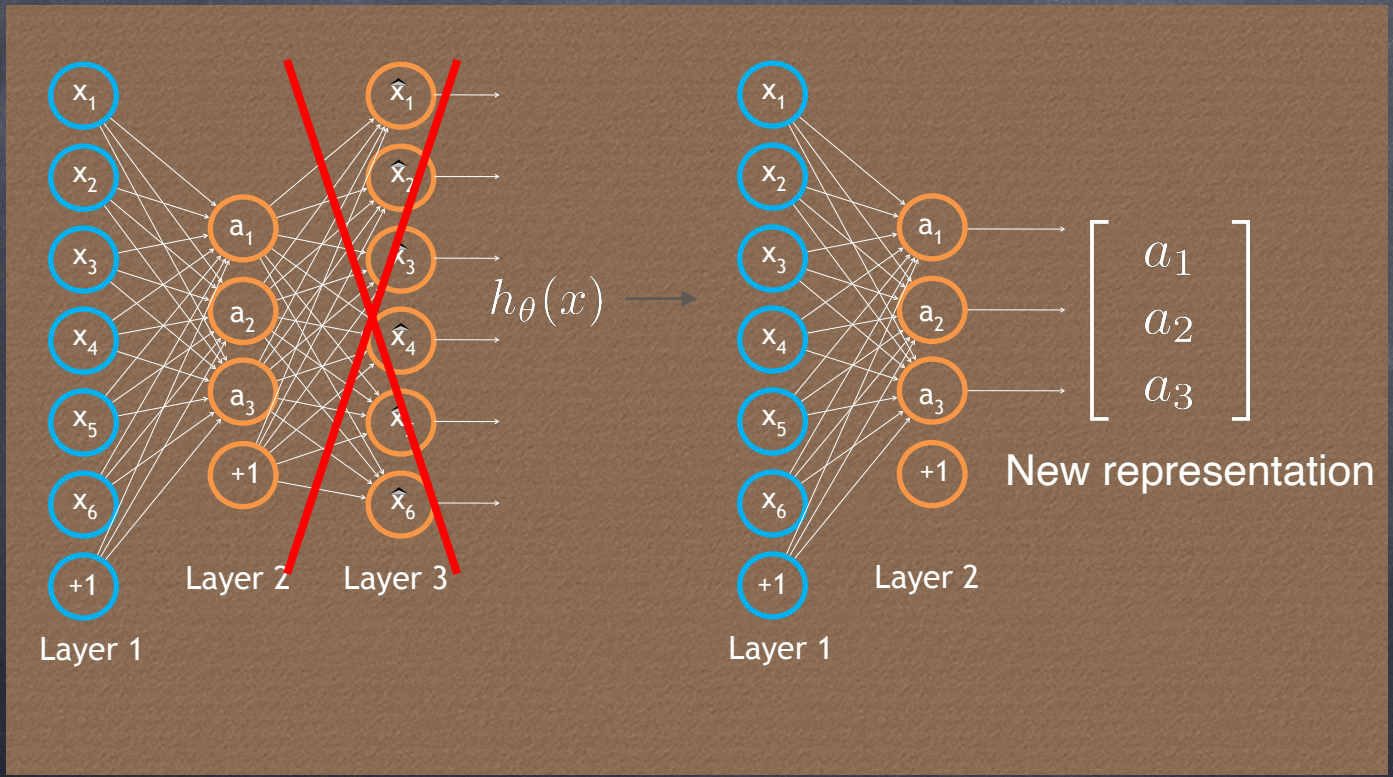


Credit: Andrew Ng

What this network is doing?

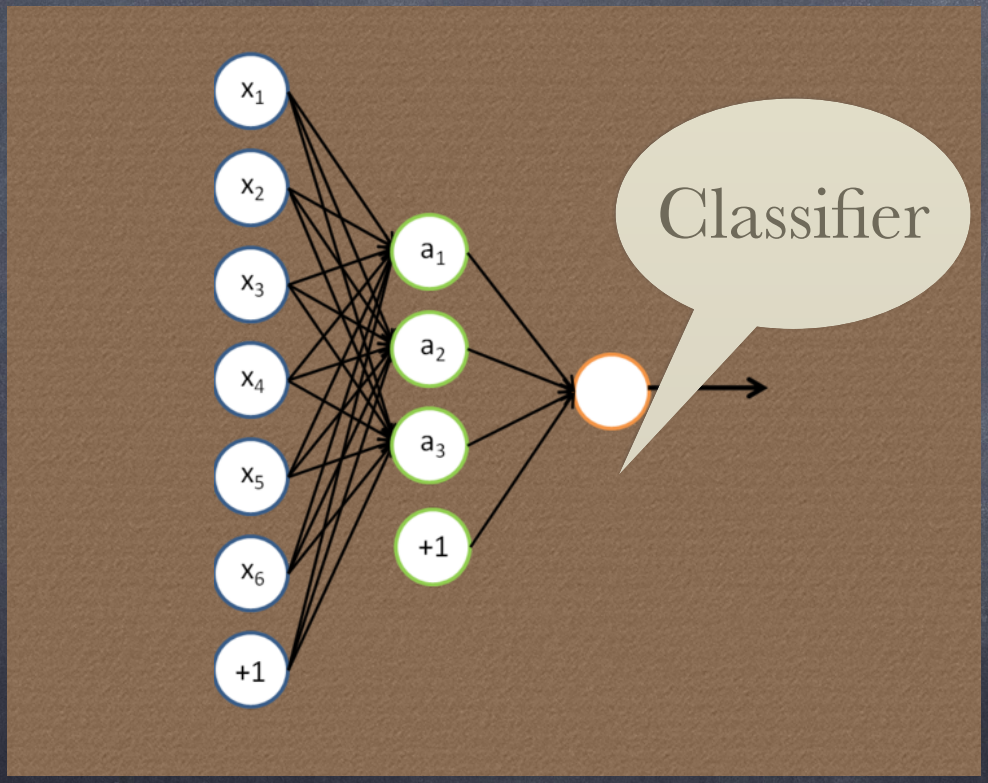
- a1 = sum(w_i*x_i)
- xcap = sum(w_1i*a_i)
- xcap = sum(w_1i*(sum(w_i*x_i))
- ||x - xcap||_2

Autoencoder



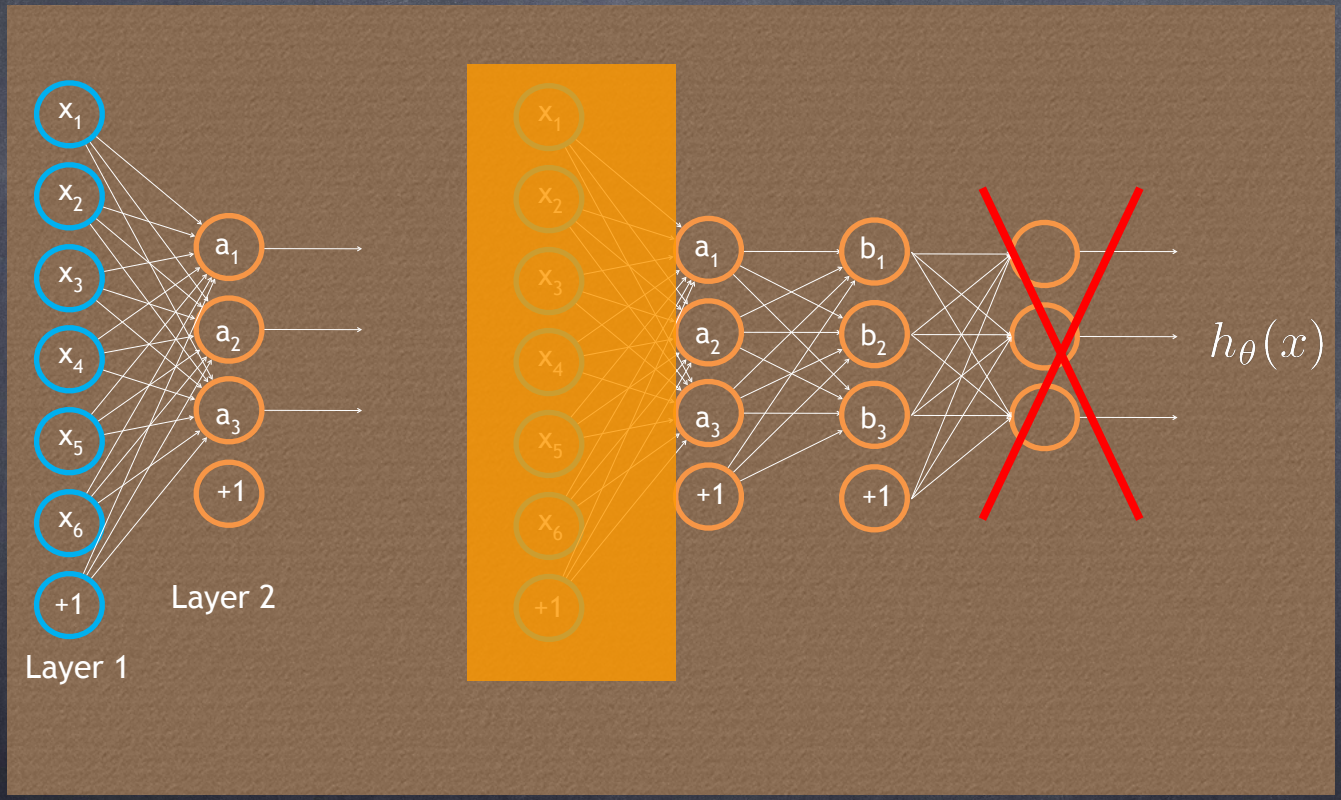
Credit: Andrew N

Autoencoder



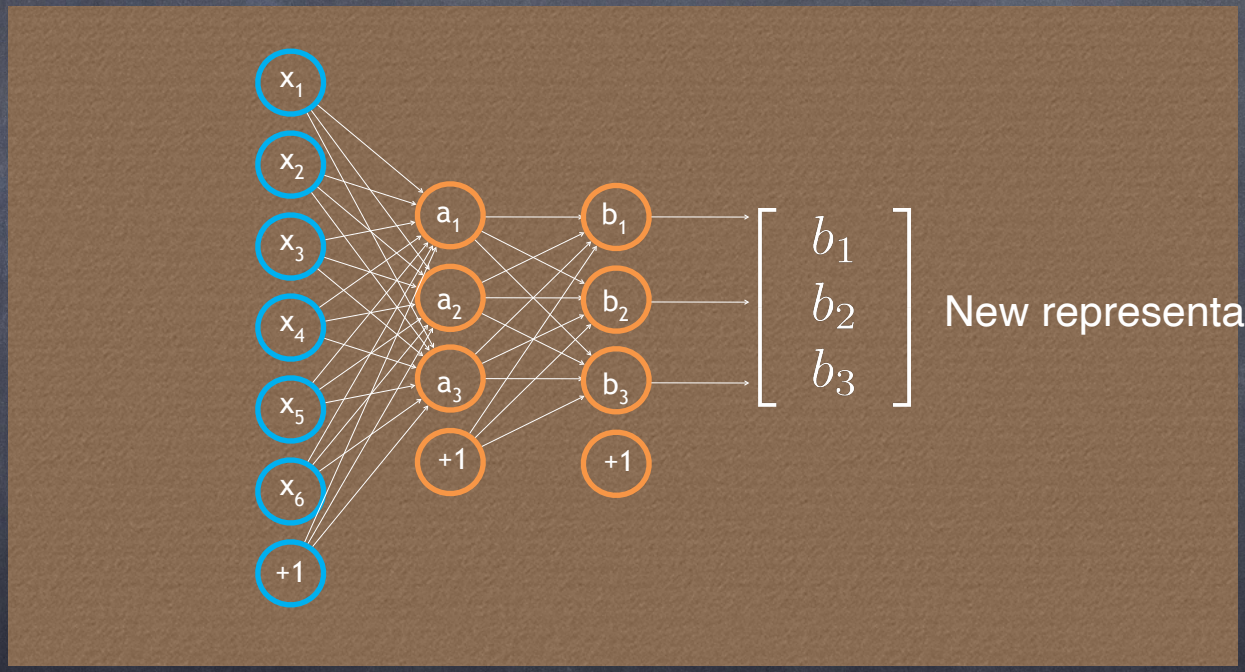
Credit: Andrew N

Auto-encoder



Credit: Andrew N

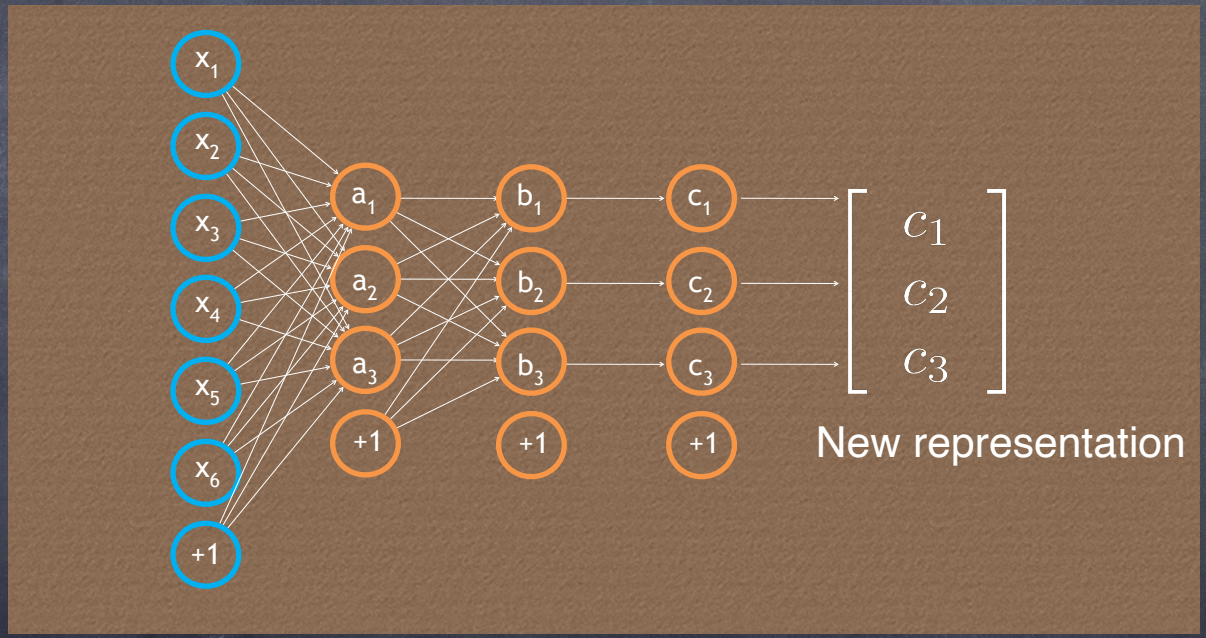
Auto-encoder



Greedy Layer by Layer Training

Credit: Andrew N

Multilayer Auto-encoder: Deep Auto-encoder



Credit: Andrew N

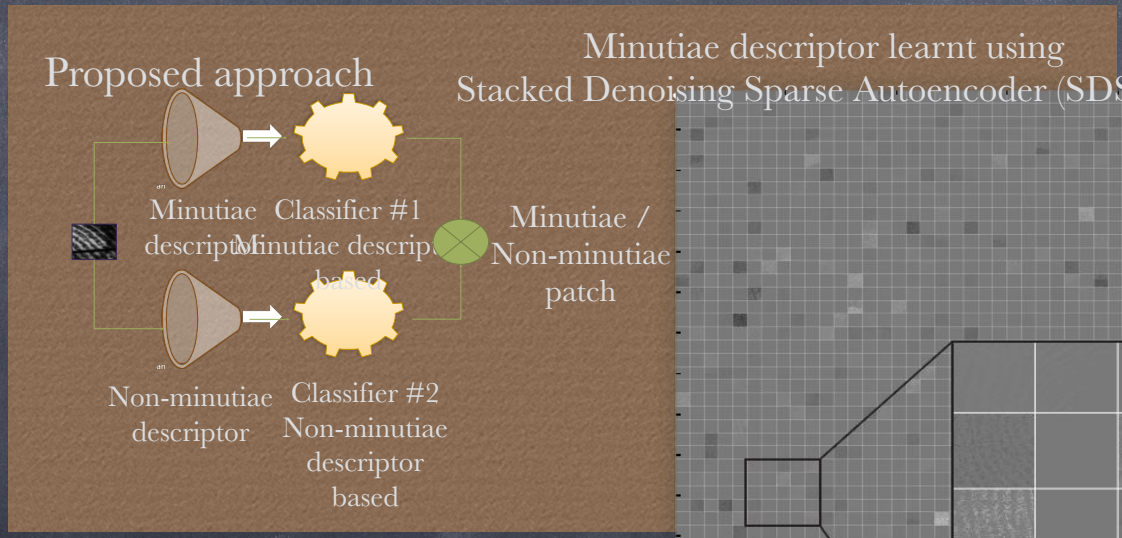
Multilayer Auto-encoder: Deep Auto-encoder

$$\underset{\mathbf{W}}{\operatorname{argmin}} ||\mathbf{X} - gof(\mathbf{X})||_F^2 + R(\mathbf{W}, \mathbf{X}) \quad \text{for all } \mathbf{W}$$

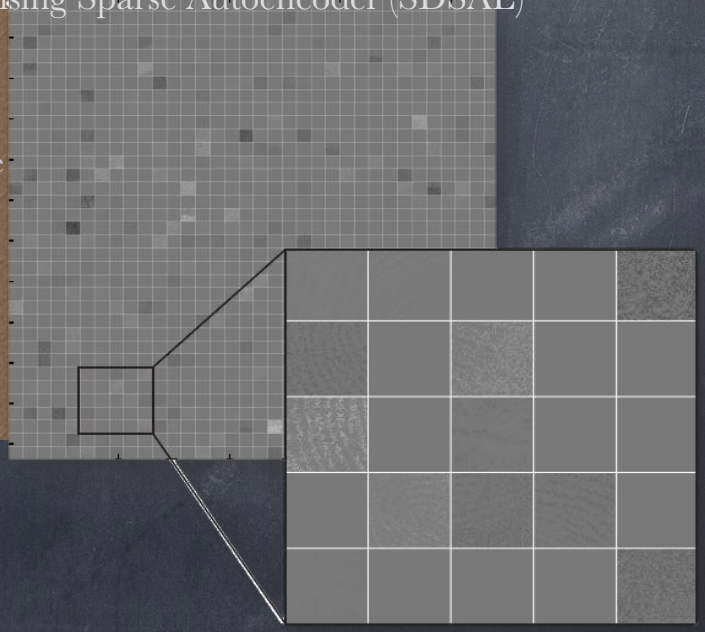
$$g = \mathbf{W_1'}\ \phi(\mathbf{W_2'}...\phi(\mathbf{W_L'}(f(\mathbf{X}))))$$

$$f = \phi(\mathbf{W_L}\ \phi(\mathbf{W_{L-1}}...\phi(\mathbf{W_1}(\mathbf{X}))))$$

Latent Fingerprint Representation using AE



Minutiae descriptor learnt using Stacked Denoising Sparse Autoencoder (SDSAE)

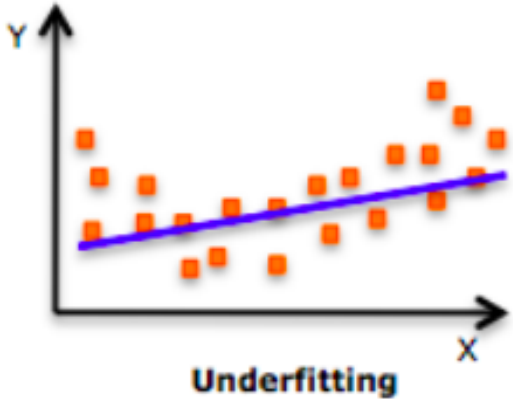


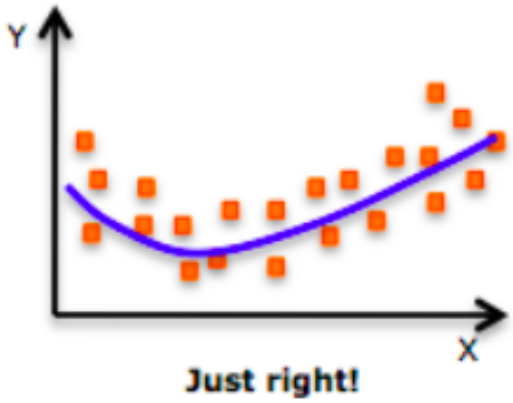
Sankaran et al. BTAS2

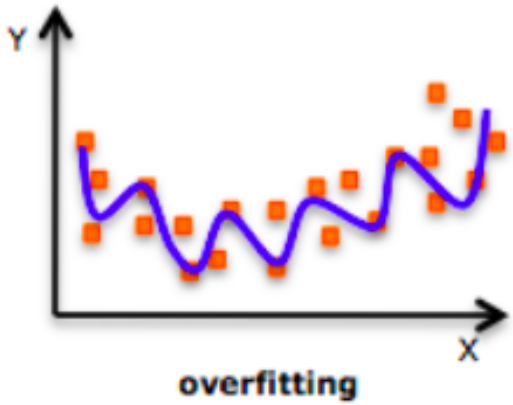
Autoencoder

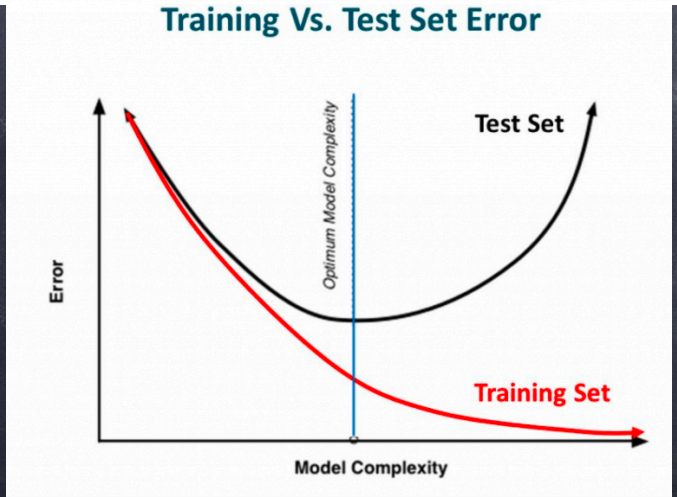
- Initial layers learn low level features
- Deeper layers learn high level features
- Sample AE architecture
  - Input size = mxn
  - First hidden layer = m/2 x n/2
  - Second hidden layer = m/4 x n/4

What is regularization

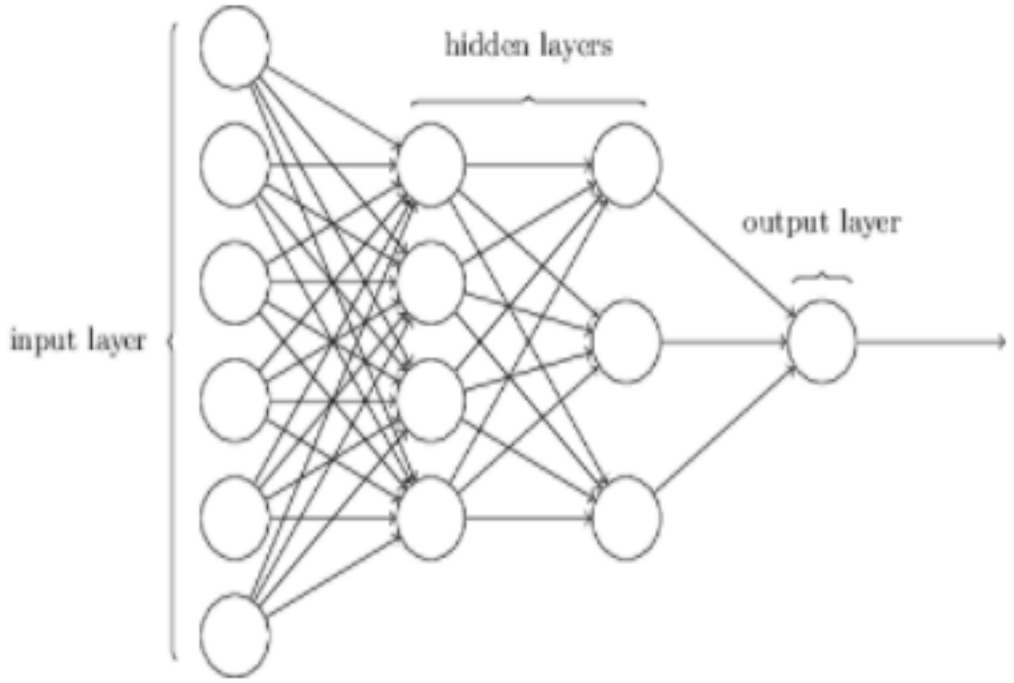


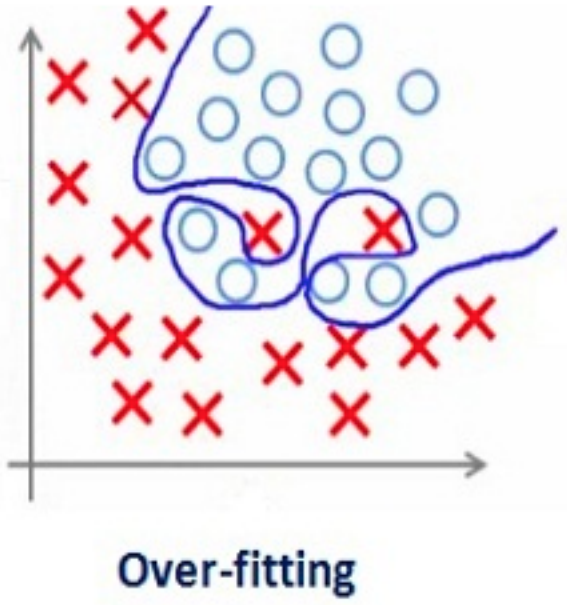




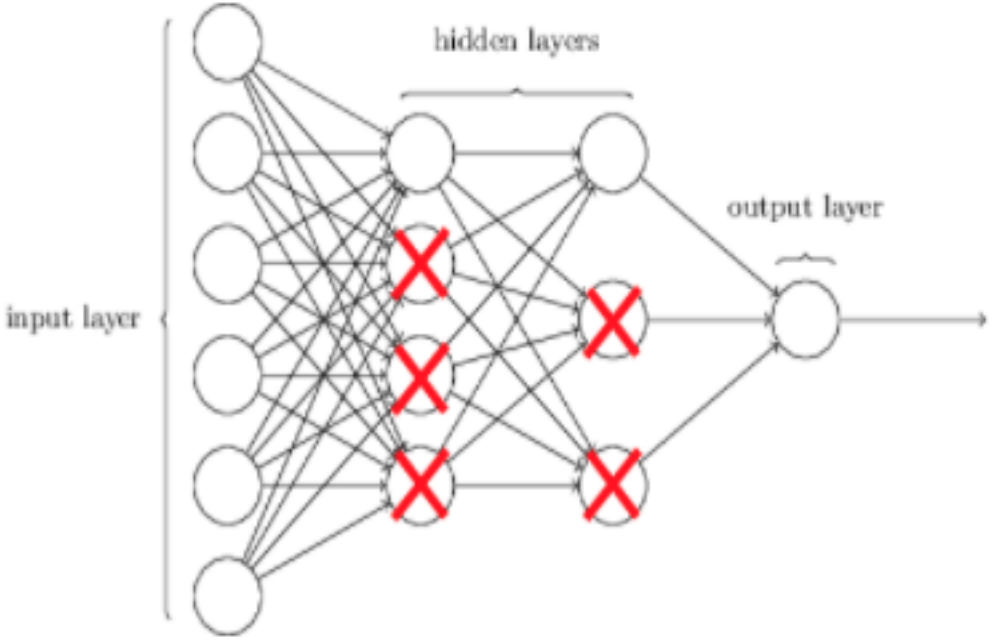


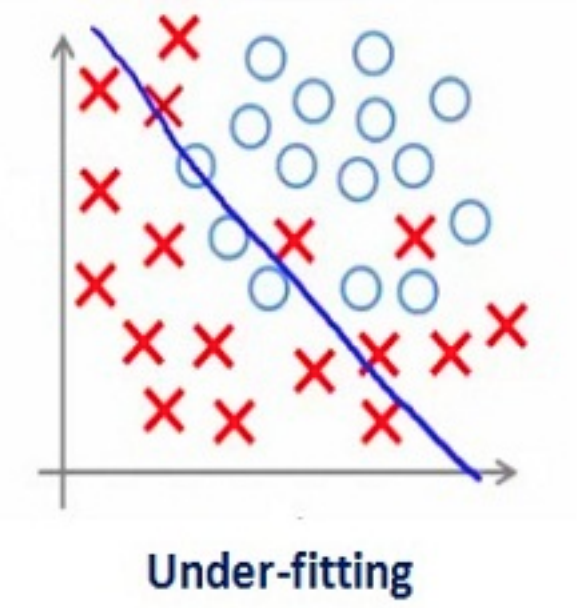
Overfitting

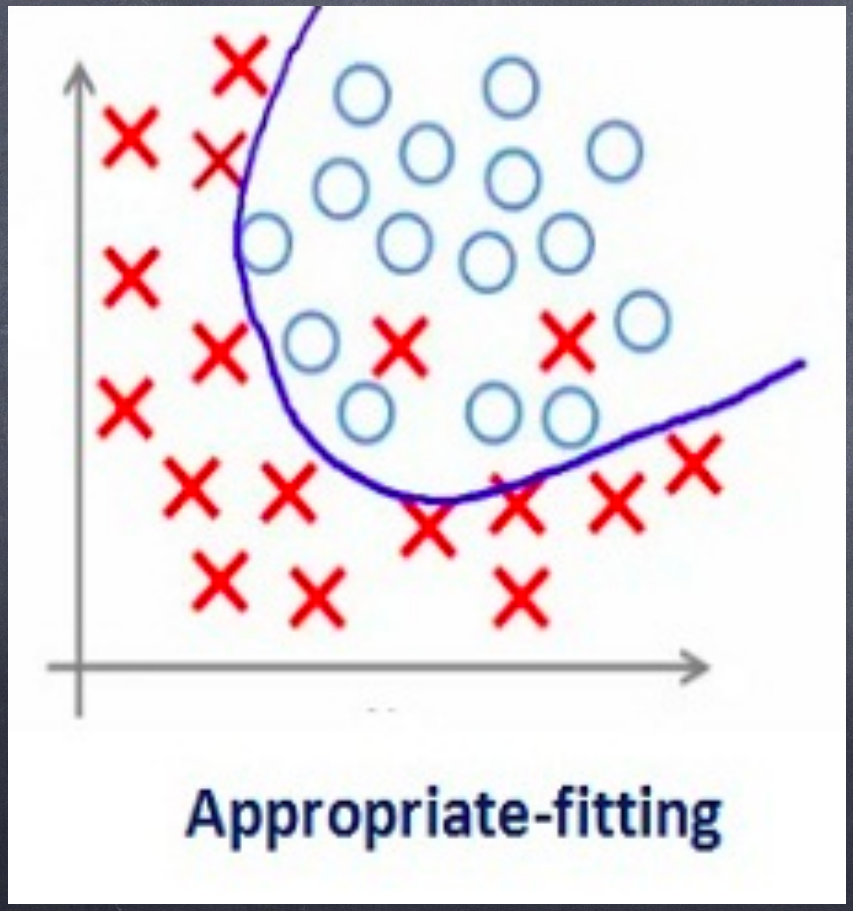




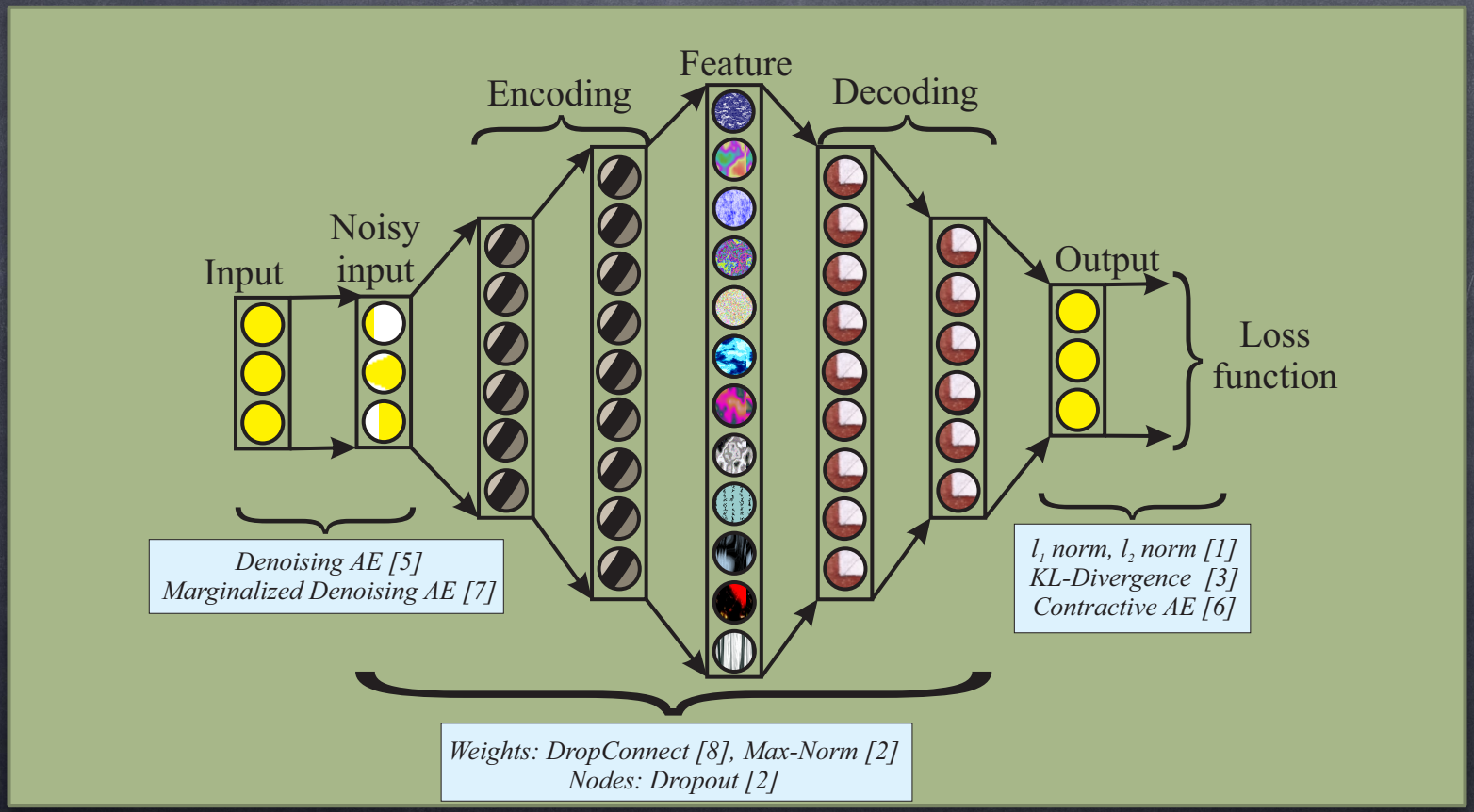
Under-fitting






Appropriate-fitting

Multilayer Auto-encoder: Regularization Approaches



More on Monday

- First programming assignment will be released

Assignment 1 Released

- For Q1 and Q2, you need to use IRIS dataset available at - https://archive.ics.uci.edu/ml/datasets/iris

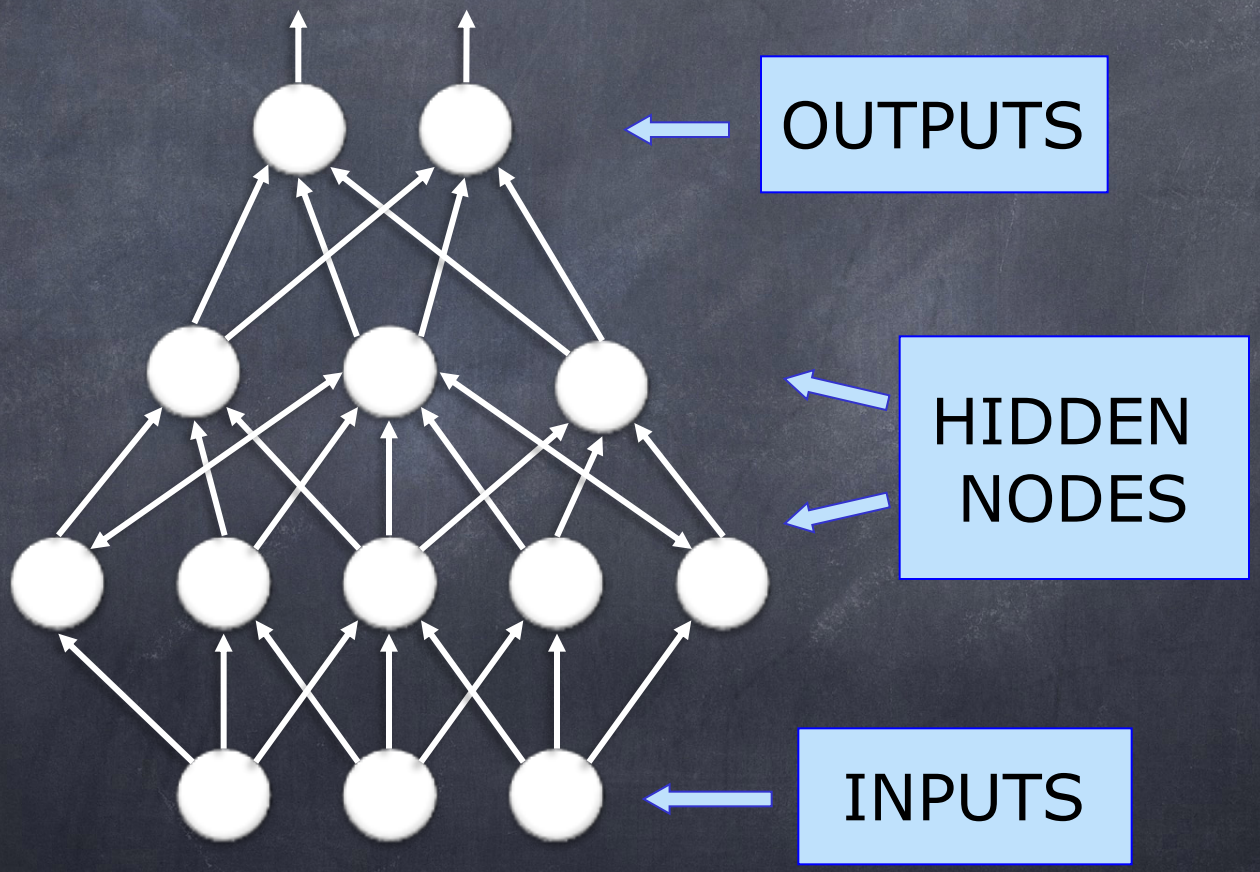
Q1. Implement two perceptron model to perform classification on iris database from scratch. You should not be using any inbuilt function for this implementation (except reading the data).
(a) vary the learning rate and show the best learning rate value when your run it for 50 epochs.
(b) vary the number of epochs from 10 to 100 in a step of 10 and show the loss value curve (using the best learning rate obtained from (a))

Q2. Implement a 3-class backpropagation NNet on your own to classify iris data, i.e. from scratch. You should not be using any inbuilt function for this implementation (except reading the data).
(a) vary the learning rate and show the best learning rate value when your run it for 50 epochs.
(b) vary the number of epochs from 10 to 100 in a step of 10 and show the loss value curve (using the best learning rate obtained from (a))
(c) add L2 regularization - show the comparisons with and without this regularization and analyze your results.

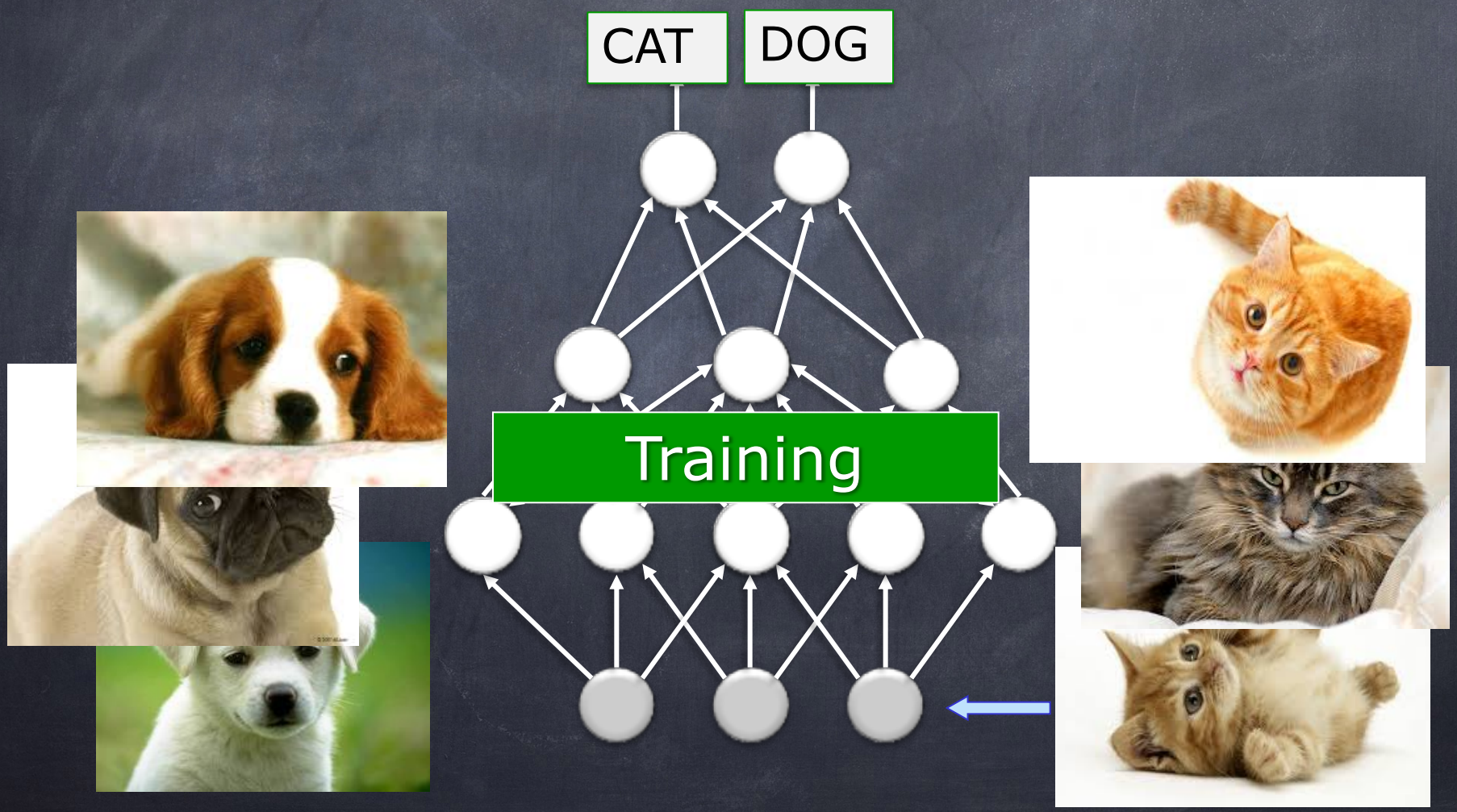
Q3. Use any toolbox in python and implement RBF NNet to solve one of the problems/databases (of your choice from the UCI ML database Repo). Analyze your results with respect to varying learning rate and epochs. You are not allowed to use someone's code available online. UCI databases: https://archive.ics.uci.edu/ml/datasets.php

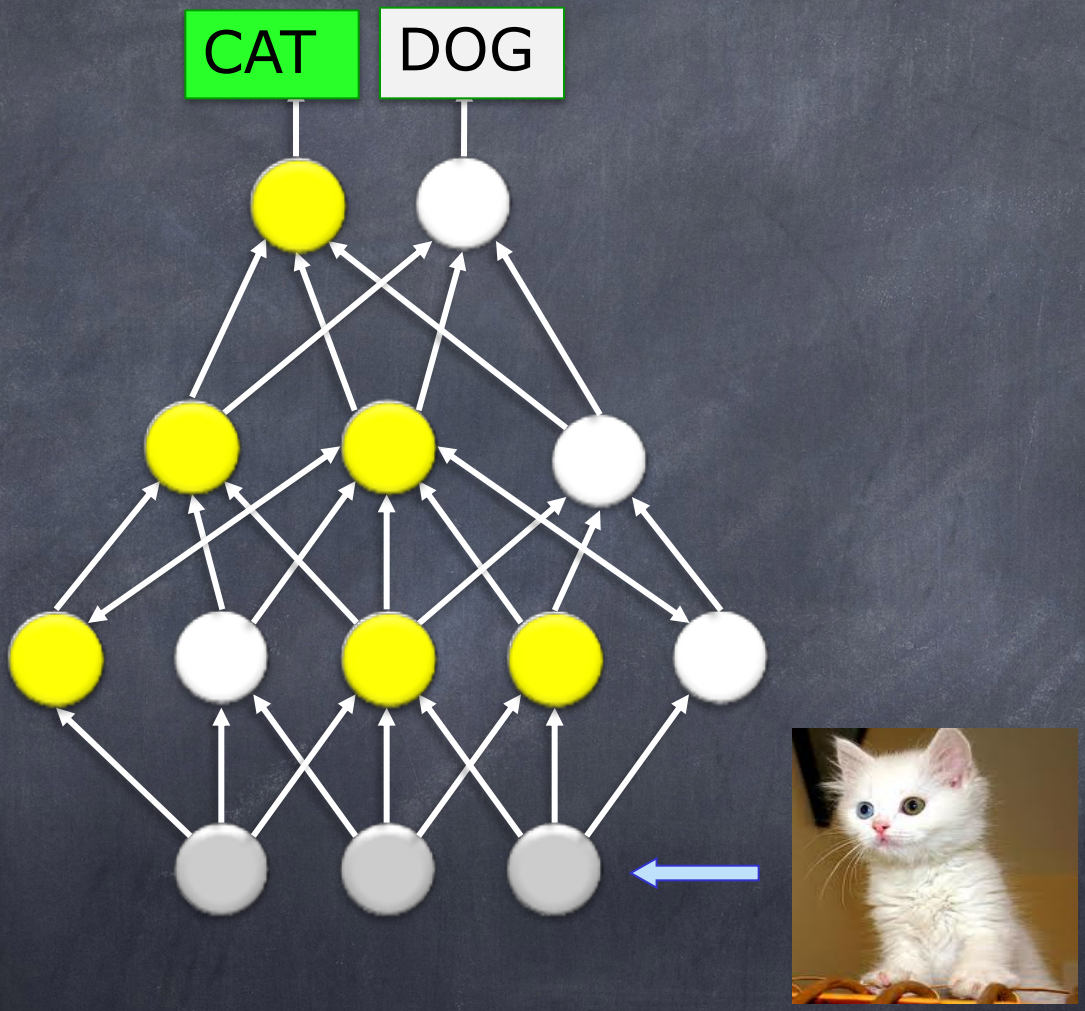
Q4. Using MNIST database, code Autoencoder model with three encoding and three decoding layers. Show the visualization of the feature maps. On the features, add a classifier to perform 10-class classification and show the training loss curve and test accuracy.

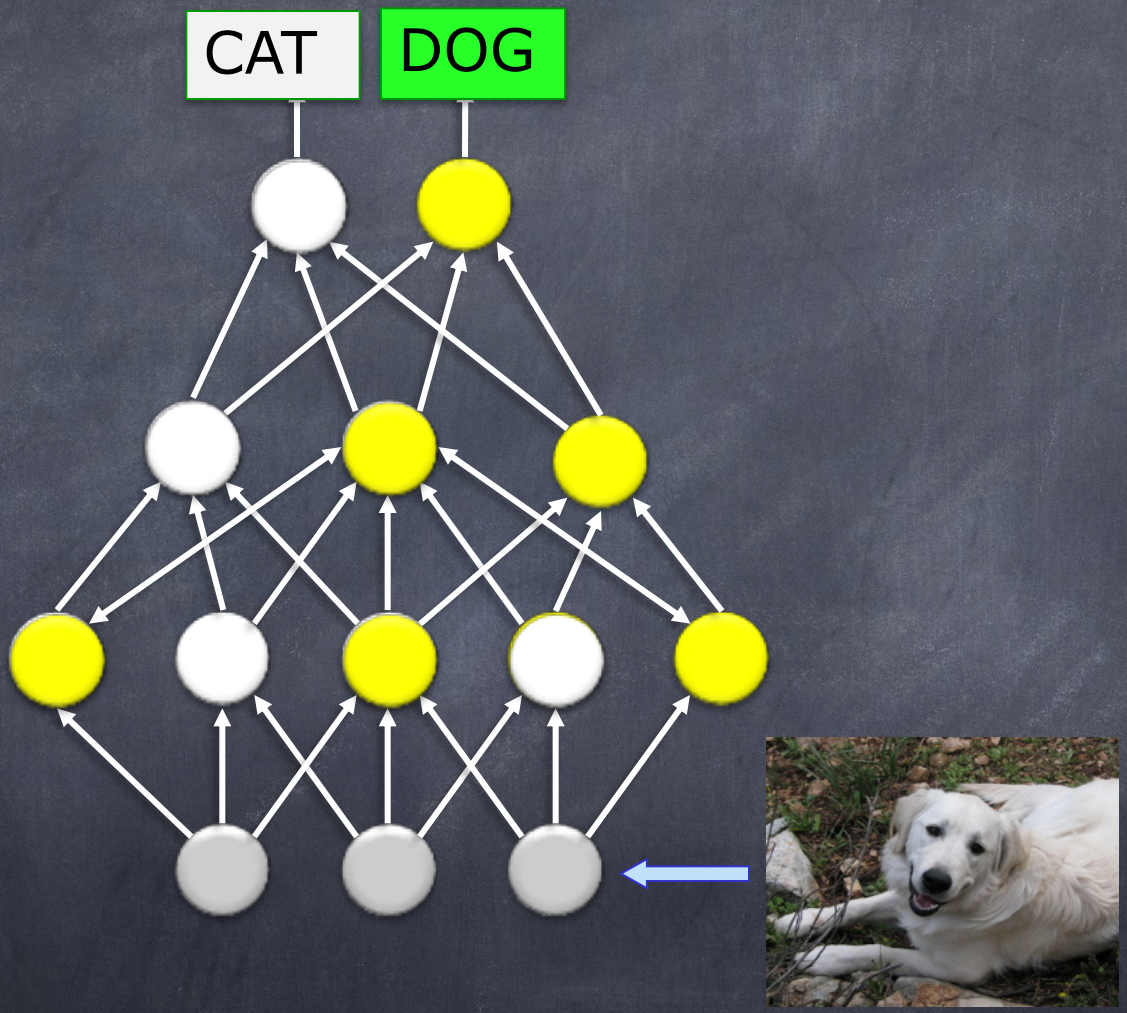
Some useful things to keep in mind while doing the coding



Two Class Classification




CAT
DOG


CAT
DOG

Learning Rate

- Recall GDR ...

Derivation of GDR …

The vector of $\frac{\partial E}{\partial w_i}$ derivatives that form the gradient can be obtained by differentiating E

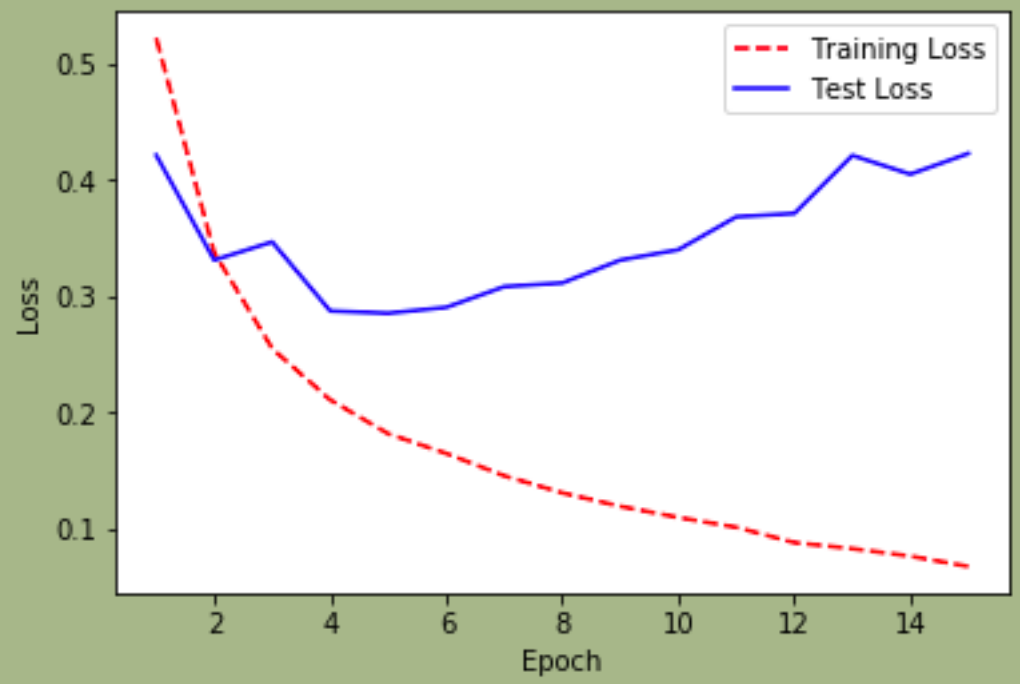
$$\begin{aligned}
\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i}\frac{1}{2}\sum_{d\in D}(t_d - o_d)^2 \\
&= \frac{1}{2}\sum_{d\in D}\frac{\partial}{\partial w_i}(t_d - o_d)^2 \\
&= \frac{1}{2}\sum_{d\in D}2(t_d - o_d)\frac{\partial}{\partial w_i}(t_d - o_d) \\
&= \sum_{d\in D}(t_d - o_d)\frac{\partial}{\partial w_i}(t_d - \vec{w}\cdot\vec{x}_d) \\
\frac{\partial E}{\partial w_i} &= \sum_{d\in D}(t_d - o_d)(-x_{id})
\end{aligned}$$

The weight update rule for standard gradient descent can be summarized as

$$\Delta w_i = -\eta\frac{\partial E}{\partial w_i}$$

$$\Delta w_i = \eta\sum_{d\in D}(t_d - o_d)x_{id}$$

No of epochs and training error

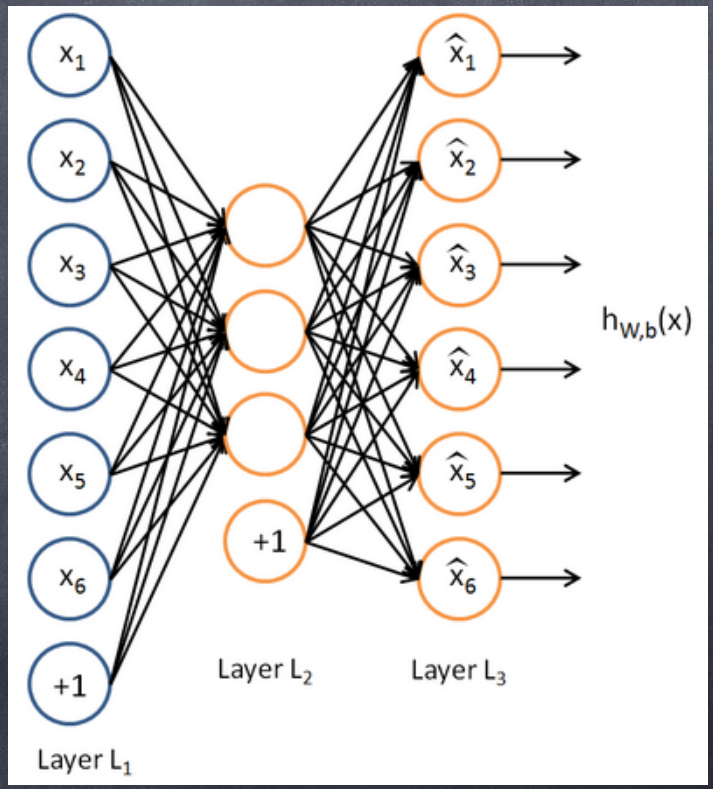


What is batch size here?

Lets go back to AEs

Let us take a look at simple Deep Architecture

- Auto-encoders
- Two parts: encoding and decoding
- Input layer: raw data X
- One hidden layer (encoding): feature learner
- Output layer (decoding): reconstruct $\hat{X}$ such that $||X-\hat{X}||^2$ is minimum

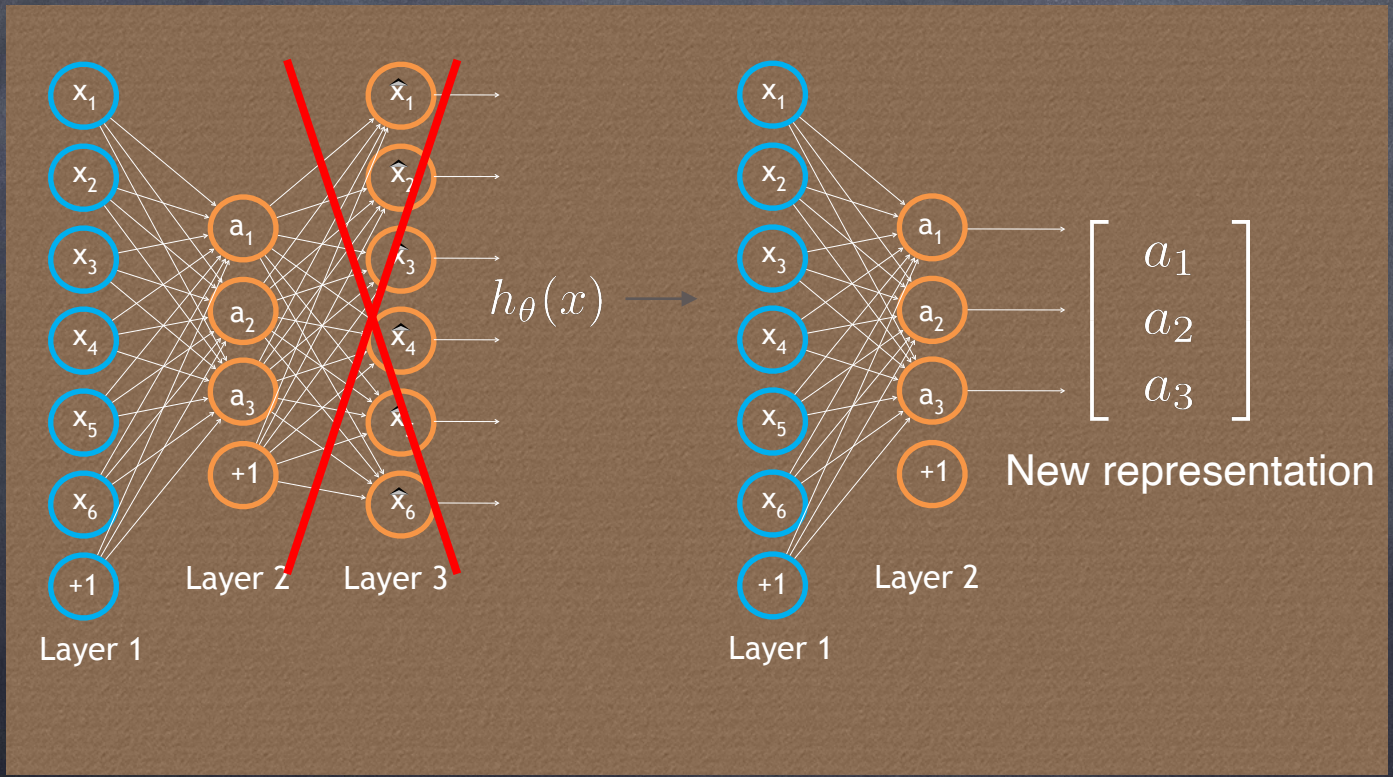


Credit: Andrew Ng

What this network is doing?

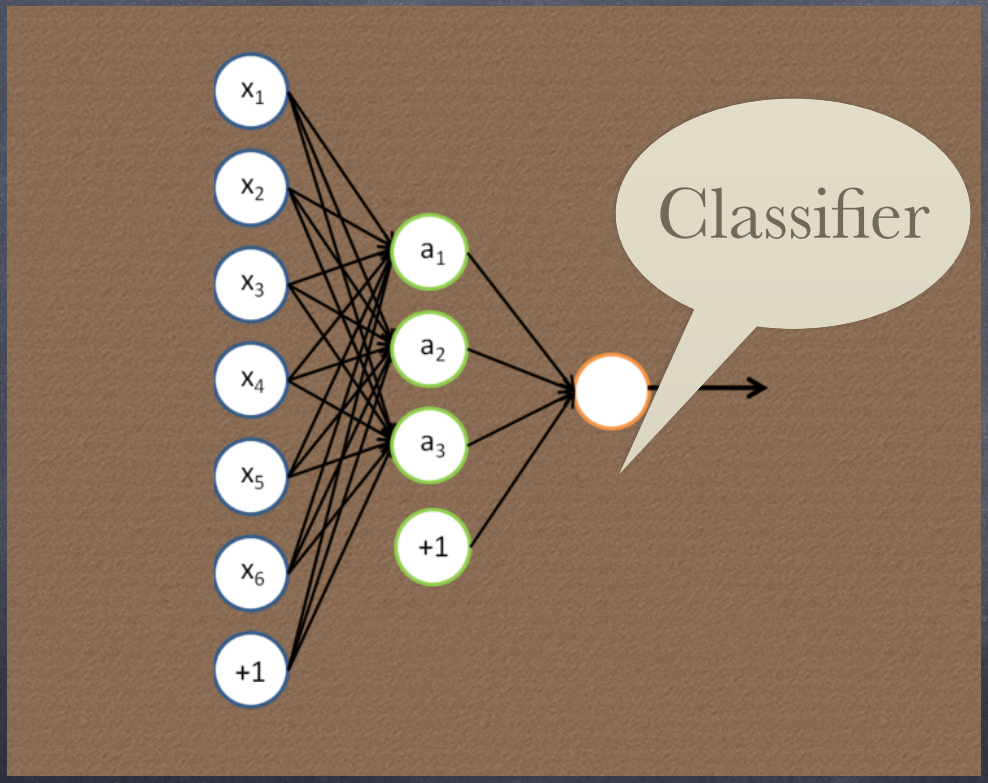
- a1 = sum(w_i*x_i)
- xcap = sum(w_1i*a_i)
- xcap = sum(w_1i*(sum(w_i*x_i))
- ||x - xcap||_2

Autoencoder



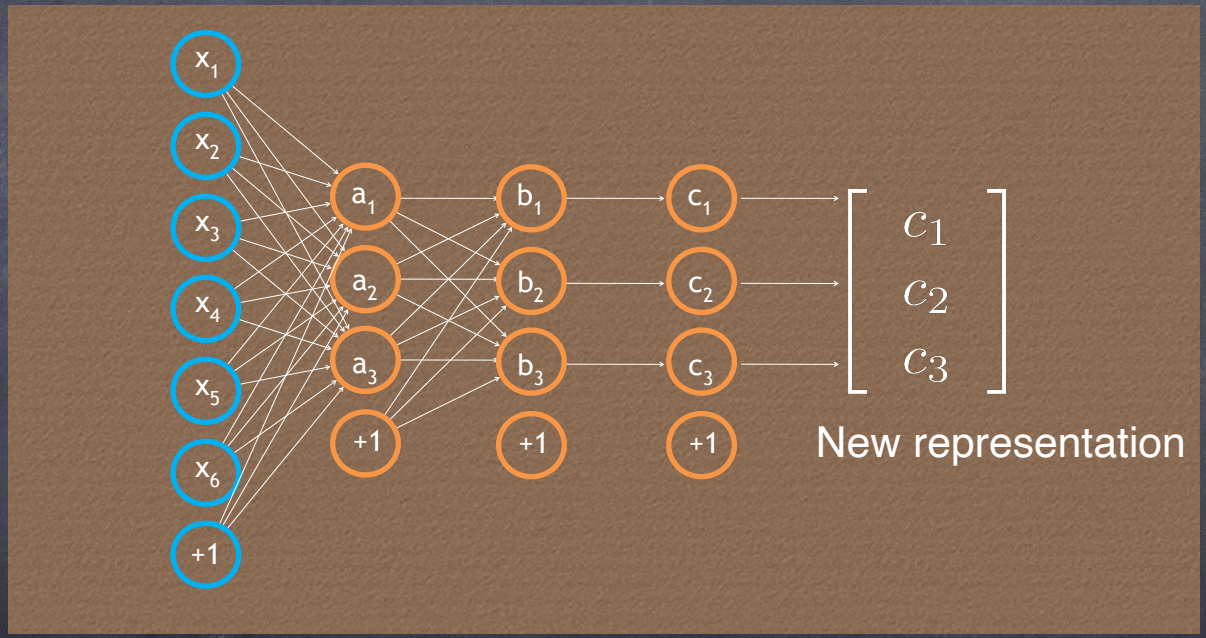
Credit: Andrew N

Autoencoder



Credit: Andrew N

Multilayer Auto-encoder: Deep Auto-encoder



Credit: Andrew N

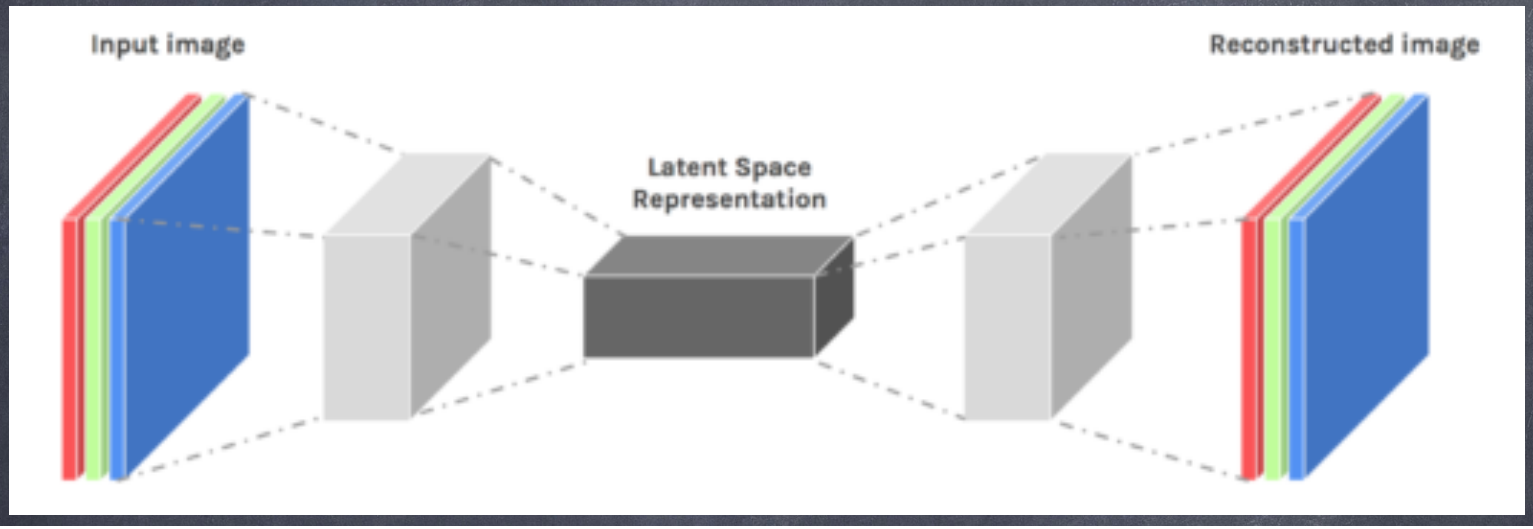
Multilayer Auto-encoder: Deep Auto-encoder

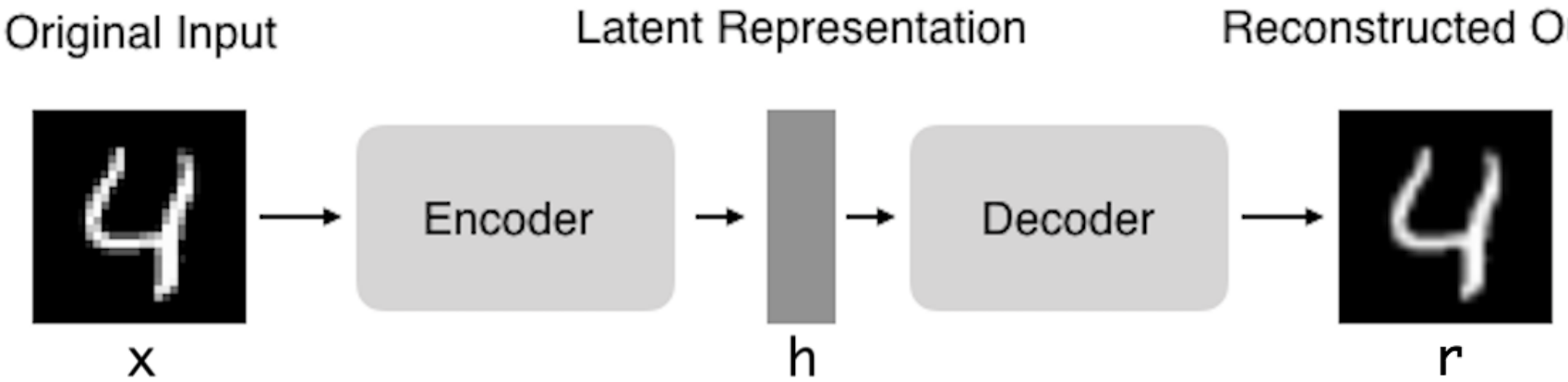
$$\underset{\mathbf{W}}{\operatorname{argmin}} ||\mathbf{X} - gof(\mathbf{X})||_F^2 + R(\mathbf{W}, \mathbf{X}) \quad \text{for all } \mathbf{W}$$

$$g = \mathbf{W_1'}\ \phi(\mathbf{W_2'}...\phi(\mathbf{W_L'}(f(\mathbf{X}))))$$

$$f = \phi(\mathbf{W_L}\ \phi(\mathbf{W_{L-1}}...\phi(\mathbf{W_1}(\mathbf{X}))))$$

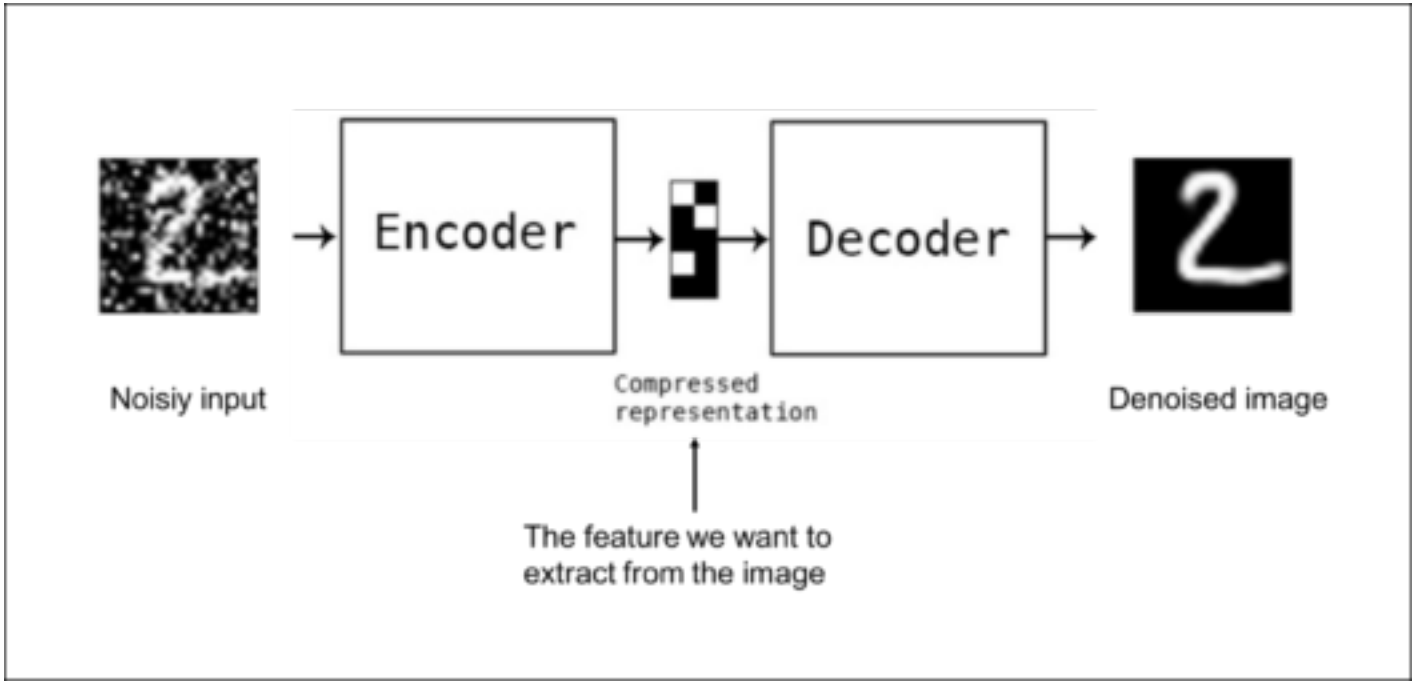
AE in Images


Input image
Latent Space Representation
Reconstructed image


Original Input
Latent Representation
Reconstructed Output
Encoder
Decoder
x
h
r

Denoising AE

- We use AE for denoising output



AE in Keras

```
•input_img = Input(shape=(784,))
•encoding_dim = 32
•encoded = Dense(encoding_dim, activation='relu')(input_img)
•decoded = Dense(784, activation='sigmoid')(encoded)
•autoencoder = Model(input_img, decoded)
```

- Maps 28x28 images into a 32 dimensional vector
- Unfortunately, you cannot use Keras in your assignment (use pytorch)