Deep Learning

Artificial Intelligence

School of Computer Science
The University of Adelaide

Visual Learning

- Learning the mapping function.

$$f(\mathbf{X}) \to \mathbf{Y}$$

- For example, Image to category, video to action

(assume given set of discrete labels)
{dog, cat, truck, plane, ...}

$\longrightarrow$ cat

Visual Learning

- Learning the mapping function.

$$f(\mathbf{X}) \rightarrow \mathbf{Y}$$

- **Why is it difficult?**

(assume given set of discrete labels)
{dog, cat, truck, plane, ...}

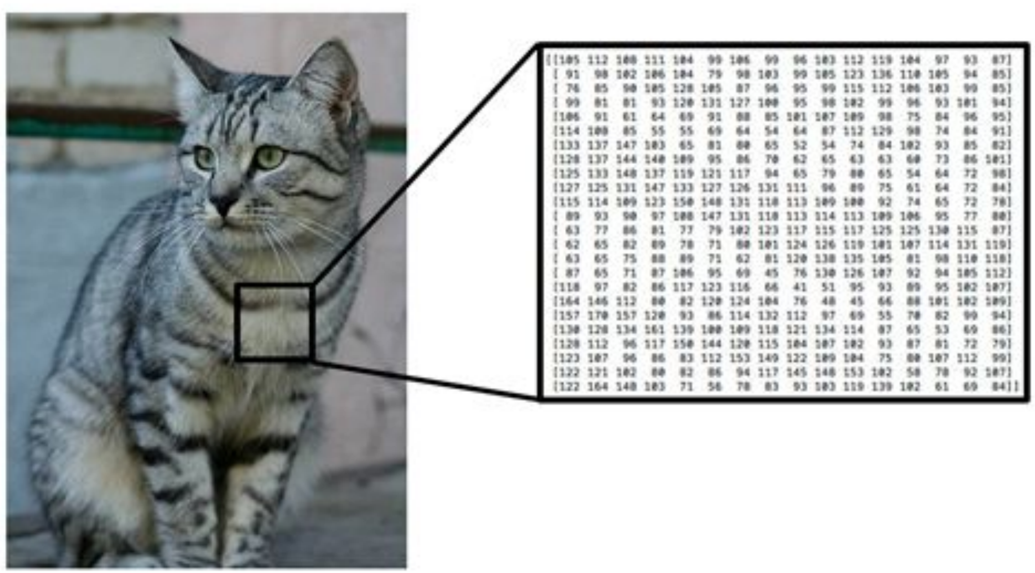
$\longrightarrow$ cat

Visual Learning

- Learning the mapping function.

$$f(\mathbf{X}) \rightarrow \mathbf{Y}$$

- **Why is it difficult?**

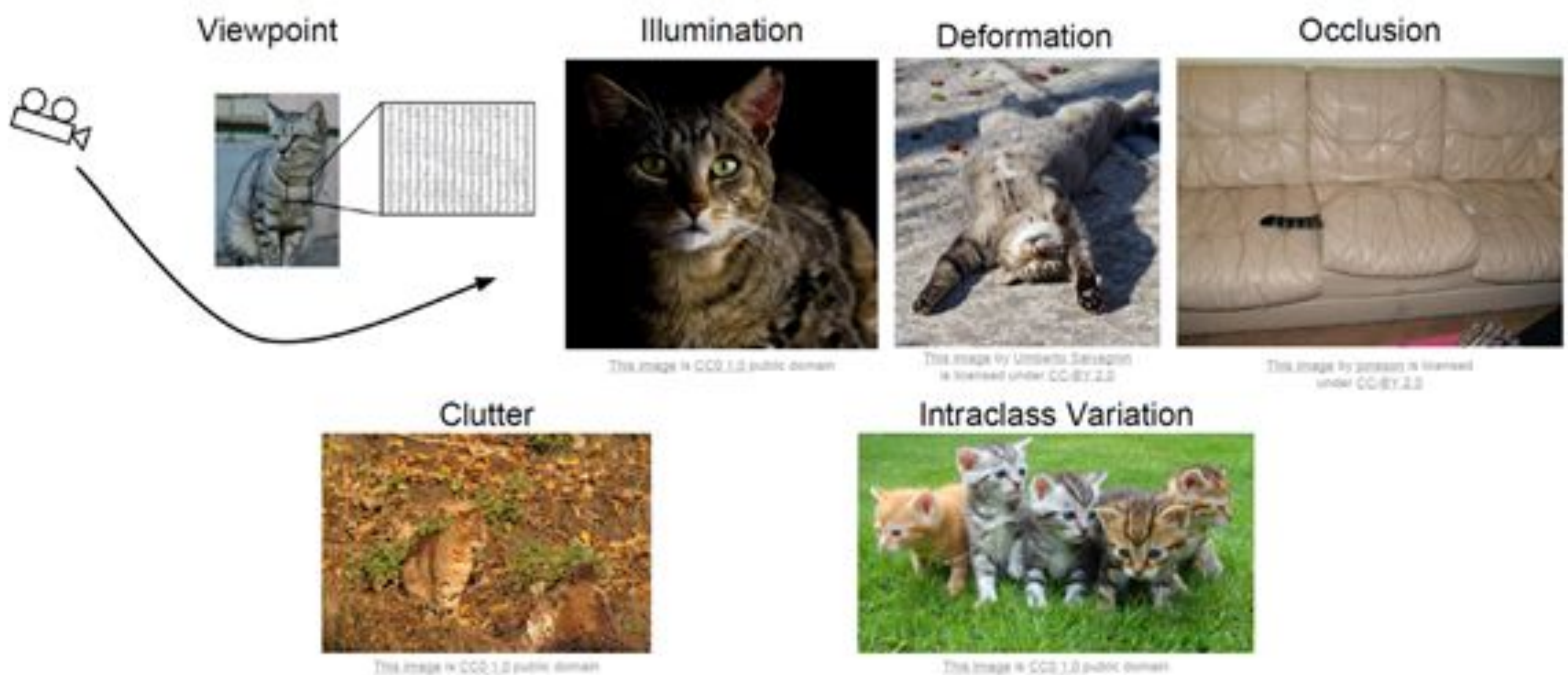


Visual Learning

- Learning the mapping function.

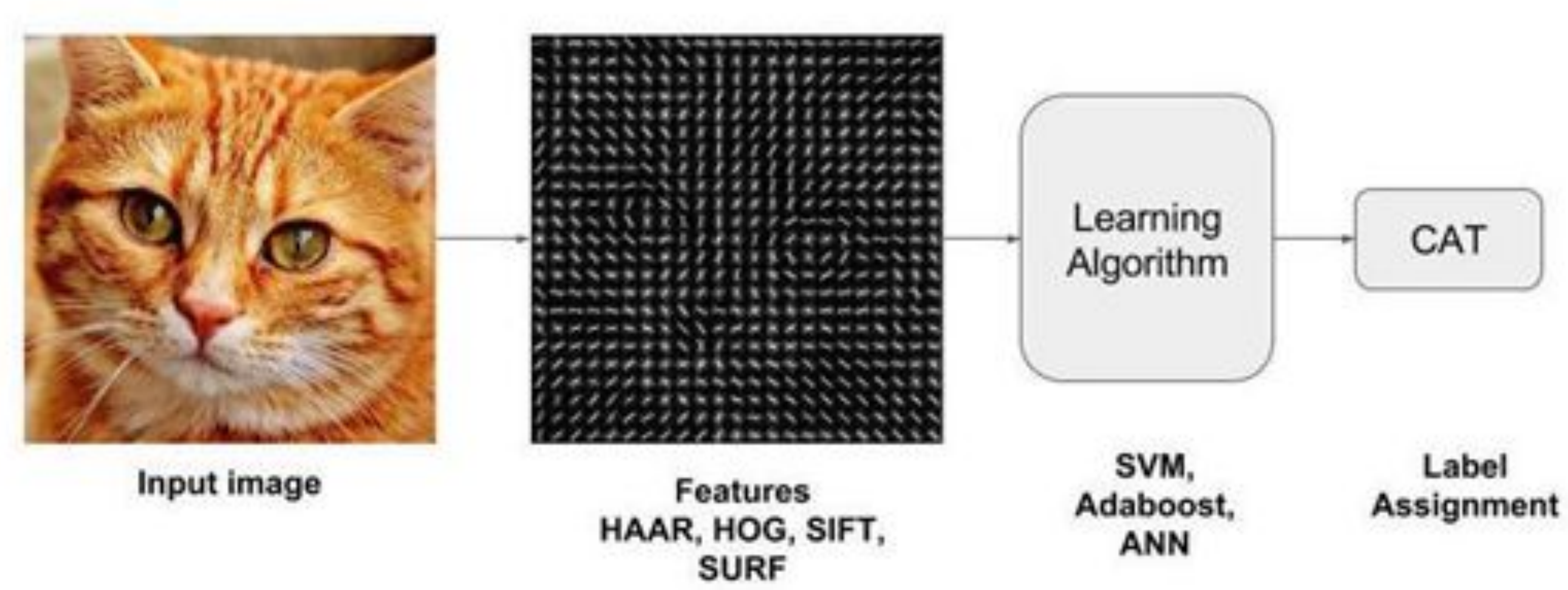
$$f(\mathbf{X}) \rightarrow \mathbf{Y}$$

- **Why is it difficult?**



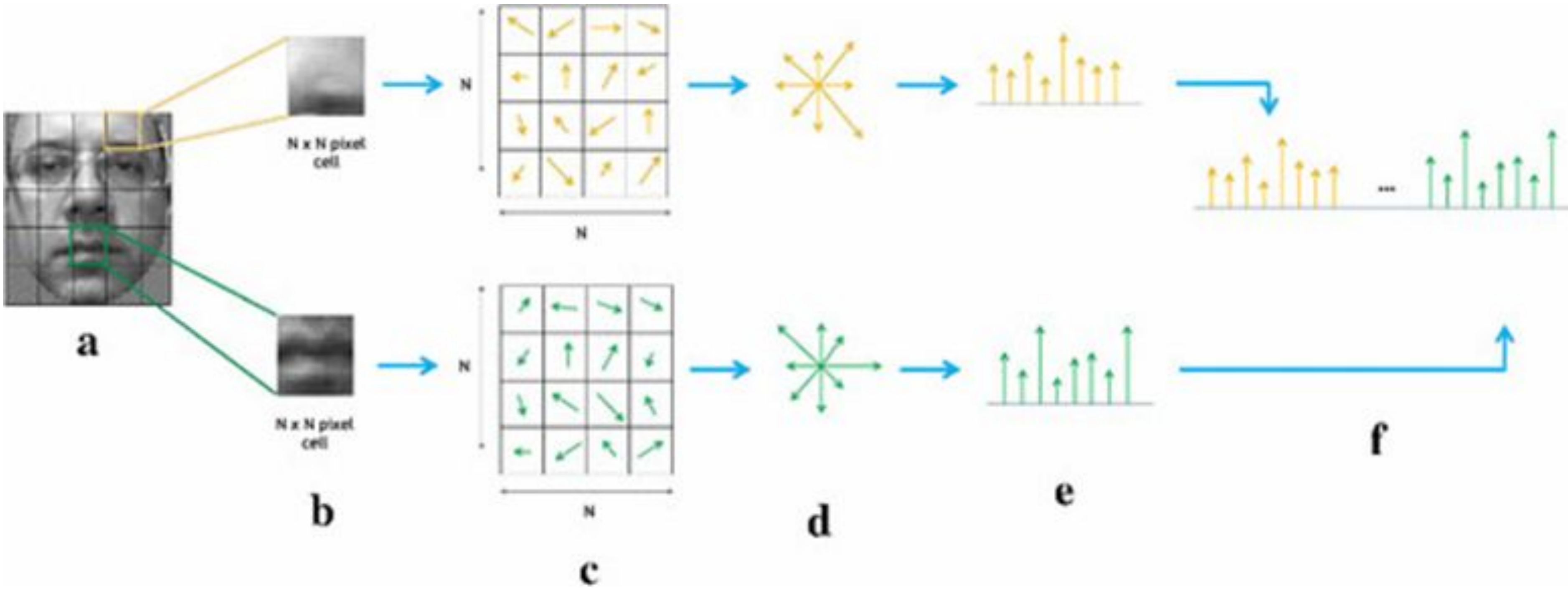
Traditional Learning

- Decompose the problem into two parts:



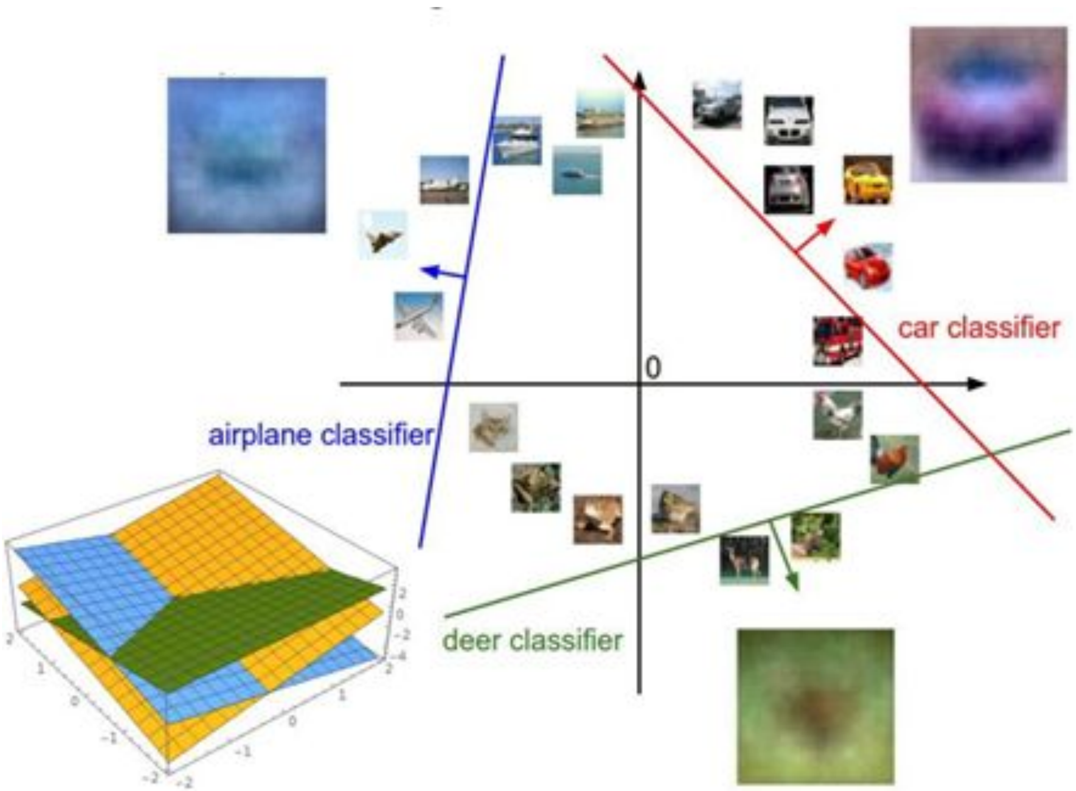
- Hand crafting features and learning a classifier.

Example of Hand-Crafted Feature (HOG)



- Engineering features with desired invariances is difficult.
- You can easily lose important information.

Example of Linear Classifier



$$f(x,W) = Wx + b$$

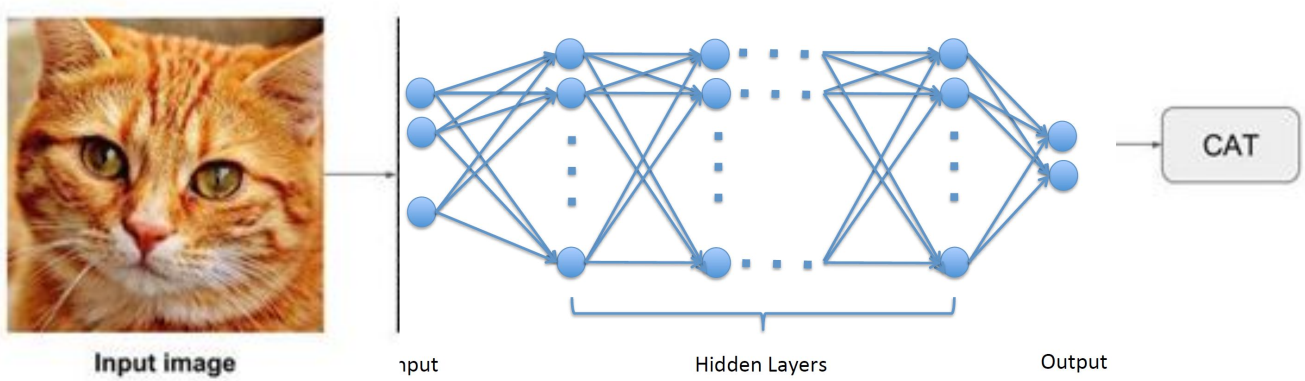
Array of **32x32x3** numbers
(3072 numbers total)

- Classification = learning parameters W and b.
- What shape is the classification boundary?
- How to optimize?

Deep Network

- Decompose the problem into multiple parts:

$$
\begin{aligned}
\mathbf{Z}_1 &= f_1(\mathbf{X}) \\
\mathbf{Z}_2 &= f_2(\mathbf{Z}_1) \\
&\cdots \\
\mathbf{Z}_t &= f_t(\mathbf{Z}_{t-1}) \\
\mathbf{Y} &= f_y(\mathbf{Z}_t)
\end{aligned}
$$

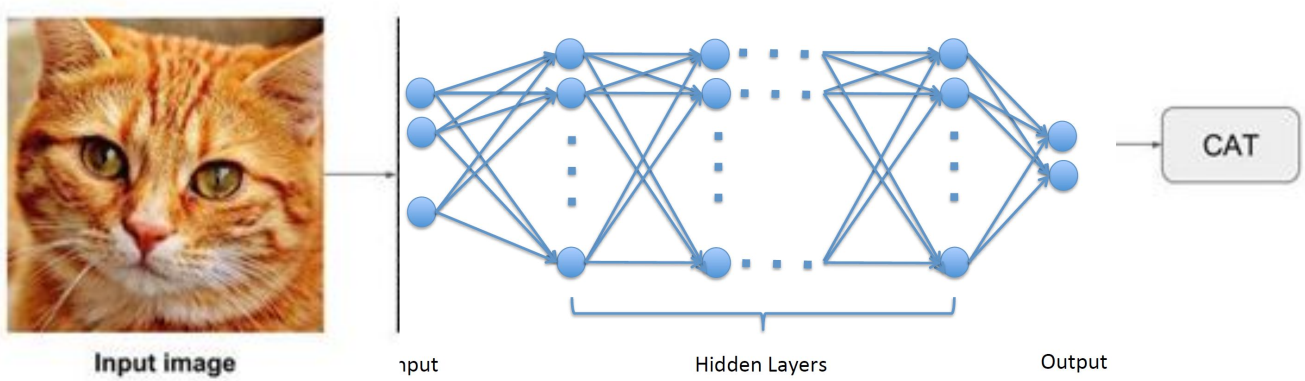


- Both features and classifier are learned

Deep Network

- Decompose the problem into multiple parts:

$$\mathbf{Z}_1 = f_1(\mathbf{X})$$
$$\mathbf{Z}_2 = f_2(\mathbf{Z}_1)$$
$$\cdots$$
$$\mathbf{Z}_t = f_t(\mathbf{Z}_{t-1})$$
$$\mathbf{Y} = f_y(\mathbf{Z}_t)$$



- Both features and classifier are learned
- Researchers have hypothesized that the number of layers in an MLP correlates well with high-level information.

MLP Based Deep Learning

- Unmanageable number of parameters
  - Example, let's take MNIST problem with 28x28=784 input images
  - 1 hidden layer with 1000 nodes and an output layer with 10 nodes: # weights = 784*1000 + 1000*10 = 794000 weights
  - 2 hidden layers with 1000 nodes and an output layer with 10 nodes: # weights = 784*1000 + 1000*1000 + 1000*10 = 1794000 weights
  - Prone to overfitting

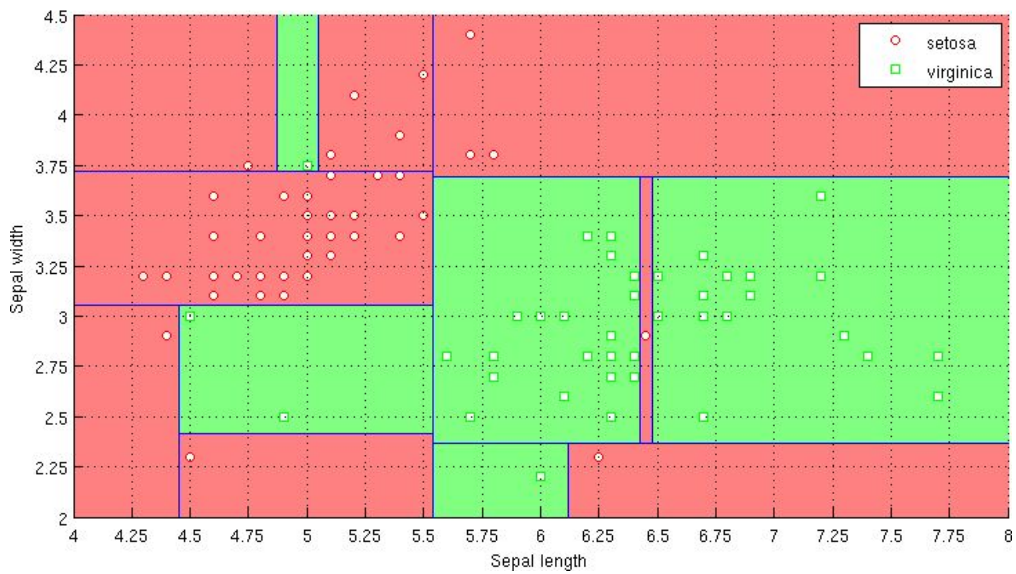
- Gradient descent didn't work beyond a couple of hidden layers
  - Magnitude kept reducing as the gradient flowed back to the input layer (vanishing gradient problem [Hochreiter91])
  - Convergence issues

- Machines (at the time – 80s and 90s) couldn't cope with datasets bigger than a few thousand samples and models with more than a few thousand weights

Learning Neural Networks

- Optimizing a loss function to learn parameters

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)}_{\text{Fitting to data}}$$



Too many Parameters

=

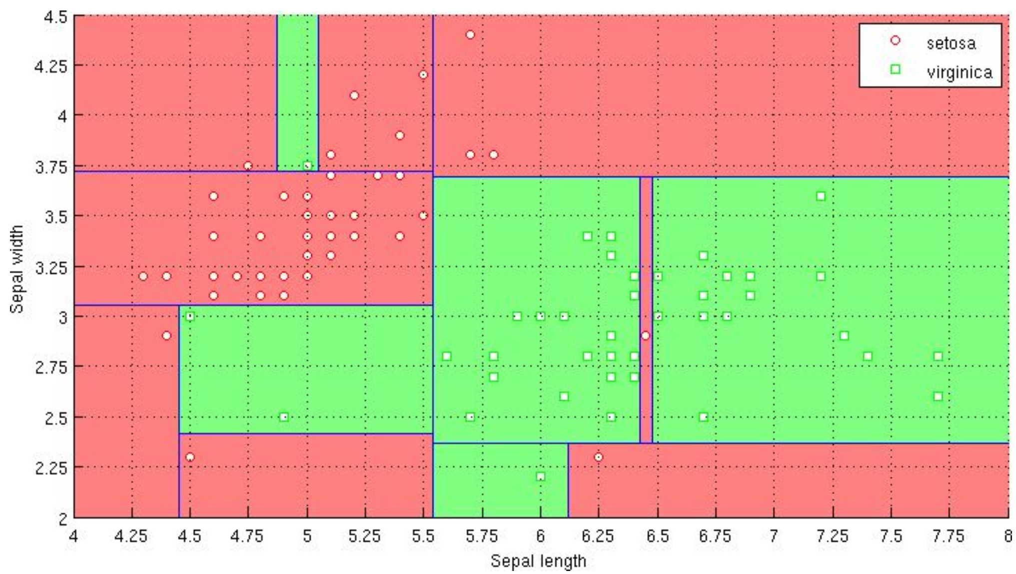
Overfitting!!!

Regularization

- Optimizing a loss function to learn parameters

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)}_{\text{Fitting to data}} + \underbrace{\lambda R(W)}_{\text{Choose the simplest model}}$$

Overfitting



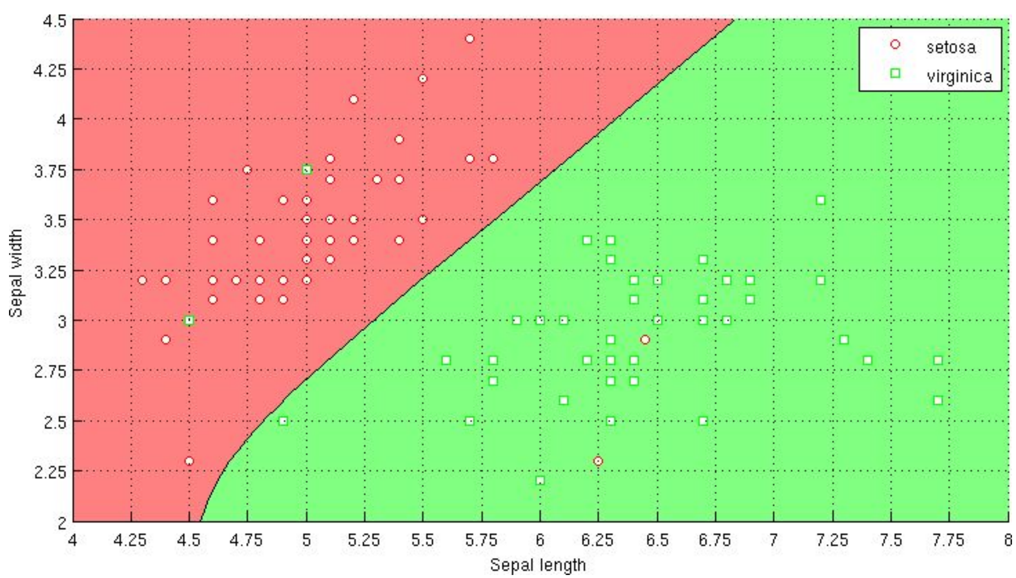
Remember Occam's Razor !!!

Usually L2 or L1 sum of the network's weights are minimized to select simplest hypothesis.

Learning of Neural Network

- Optimizing a loss function to learn parameters

$$L(W) = \frac{1}{N}\underbrace{\sum_{i=1}^{N} L_i(f(x_i, W), y_i)}_{\text{Fitting to data}} + \underbrace{\lambda R(W)}_{\text{Choose the simplest model}}$$

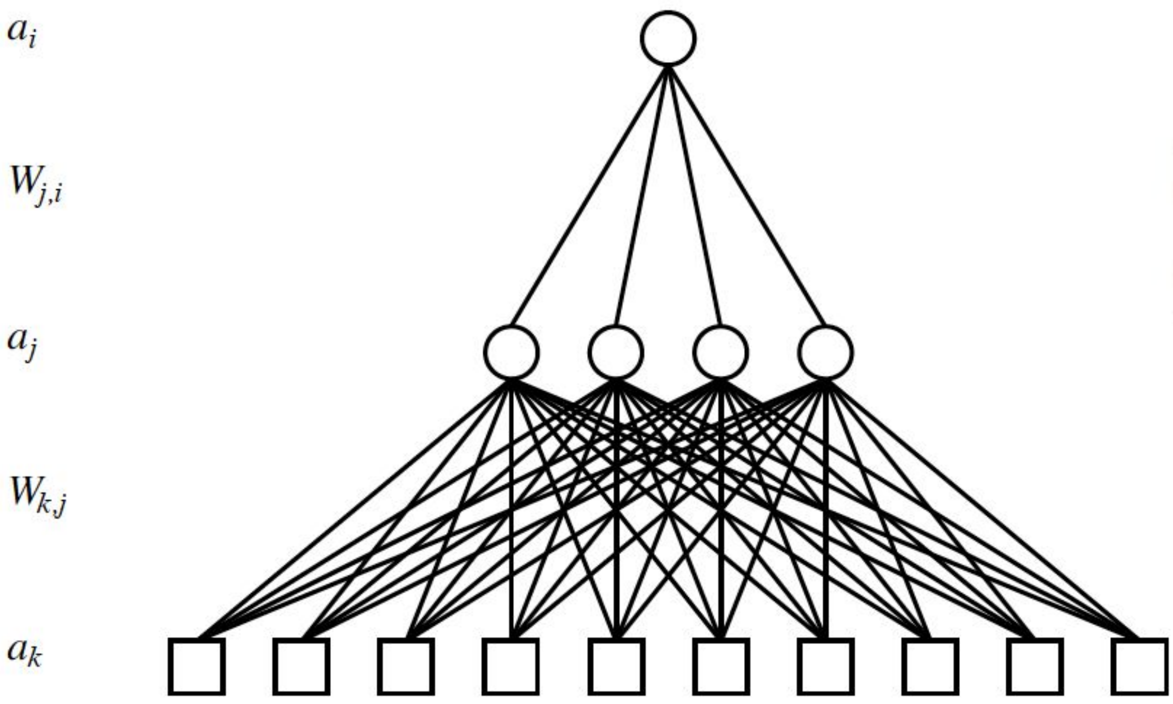


Overfitting

Remember Occam's Razor !!!

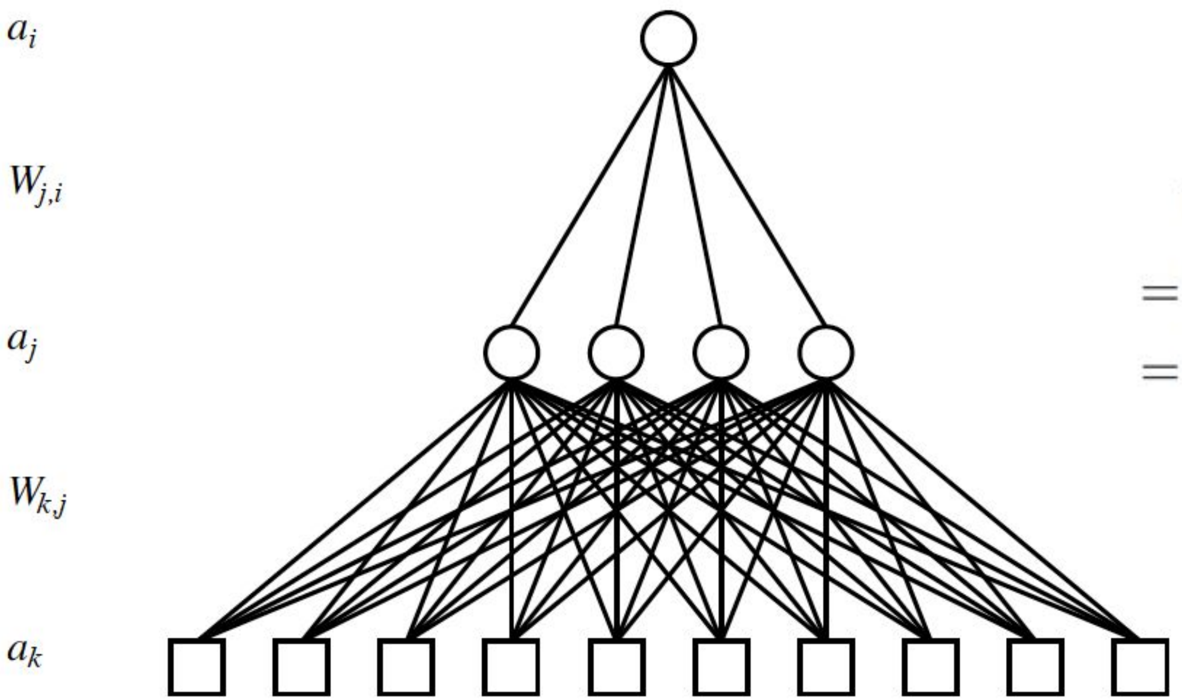
Regularizing decision boundaries in this manner help avoiding overfitting.

Non-Linear Activations



$$\mathbf{h} = s(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1)$$
$$\hat{\mathbf{y}} = \mathbf{W}_2 \mathbf{h} + \mathbf{b}_2$$

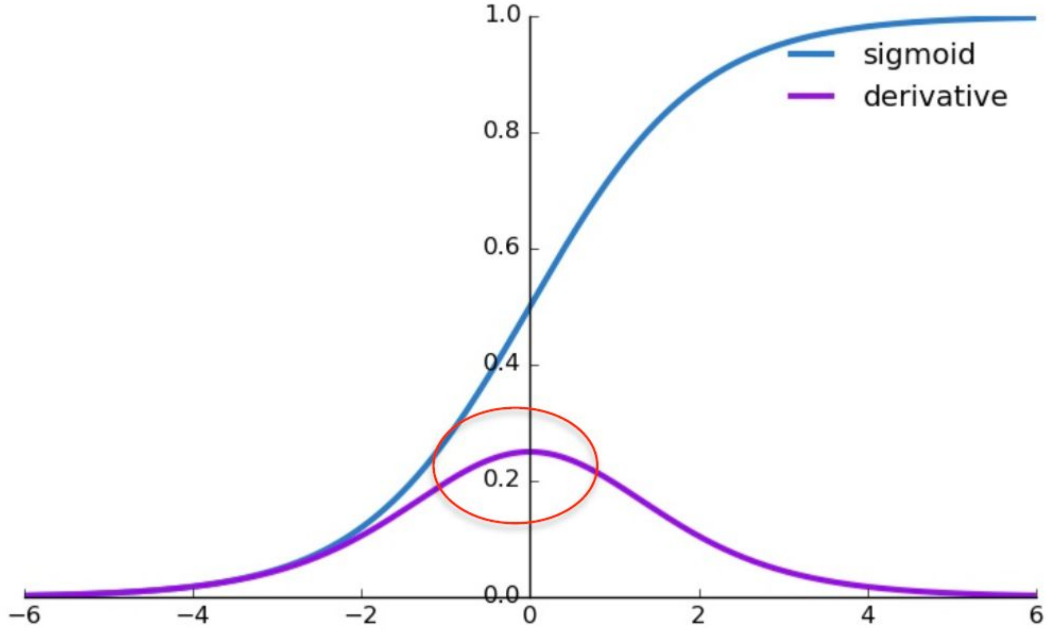
Non Linearities are Important



$$
\begin{aligned}
&\mathbf{W}_2(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2 \\
&= \mathbf{W}_2\mathbf{W}_1\mathbf{x} + \mathbf{W}_2\mathbf{b}_1 + \mathbf{b}_2 \\
&= \mathbf{W}\mathbf{x} + \mathbf{b}
\end{aligned}
$$

Vanishing Gradient Problem

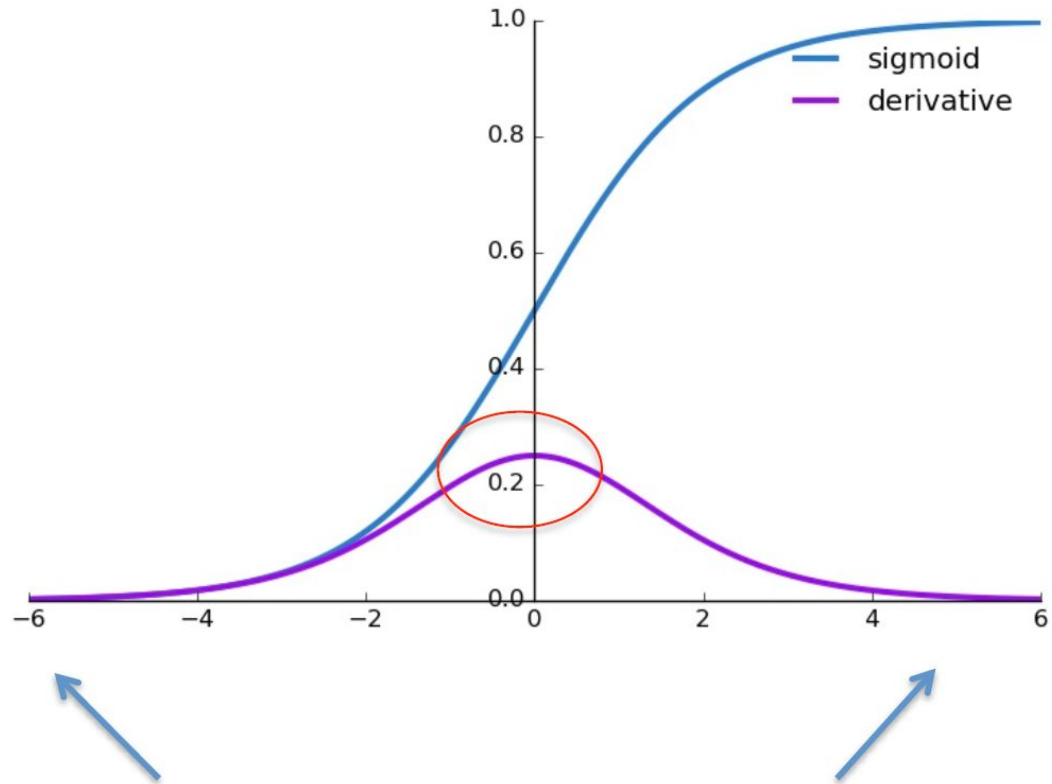
- Non-linearity based on sigmoid.



$$g_{sig}(in) = \frac{1}{1 + e^{-in}}$$

Vanishing Gradient Problem

- Non-linearity based on sigmoid.

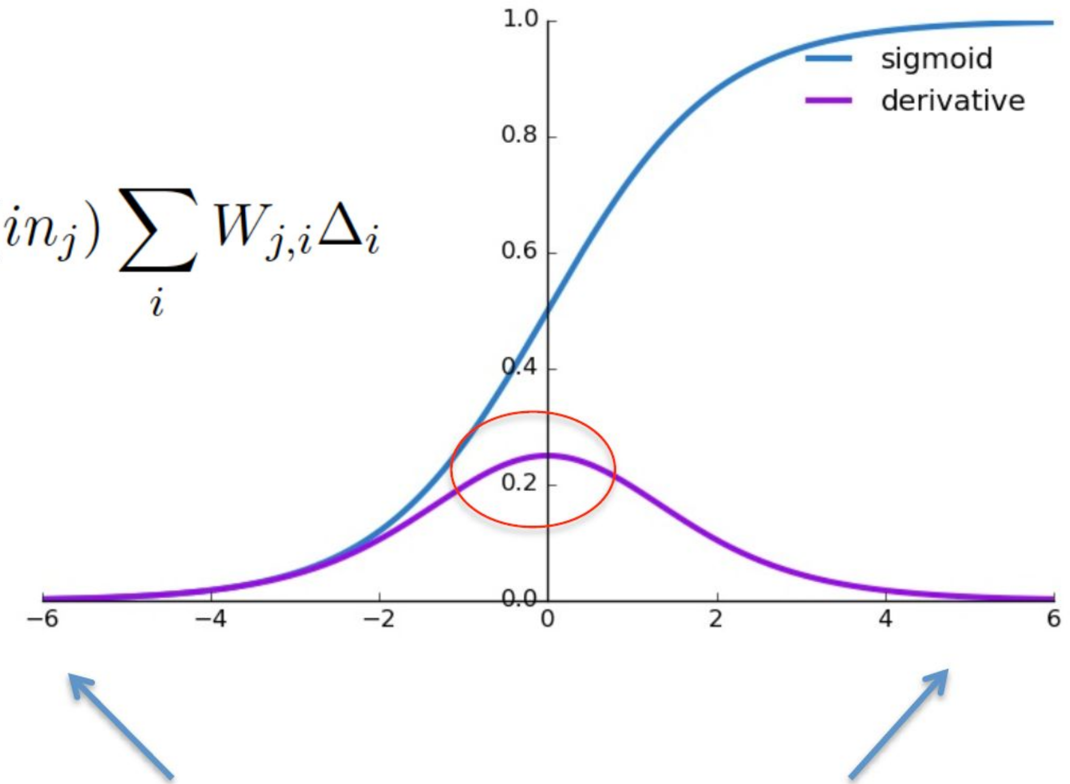


Small gradient magnitude, particularly at the tails

Vanishing Gradient Problem

- After several multiplications over the layers, the grad mag will become insignificant.

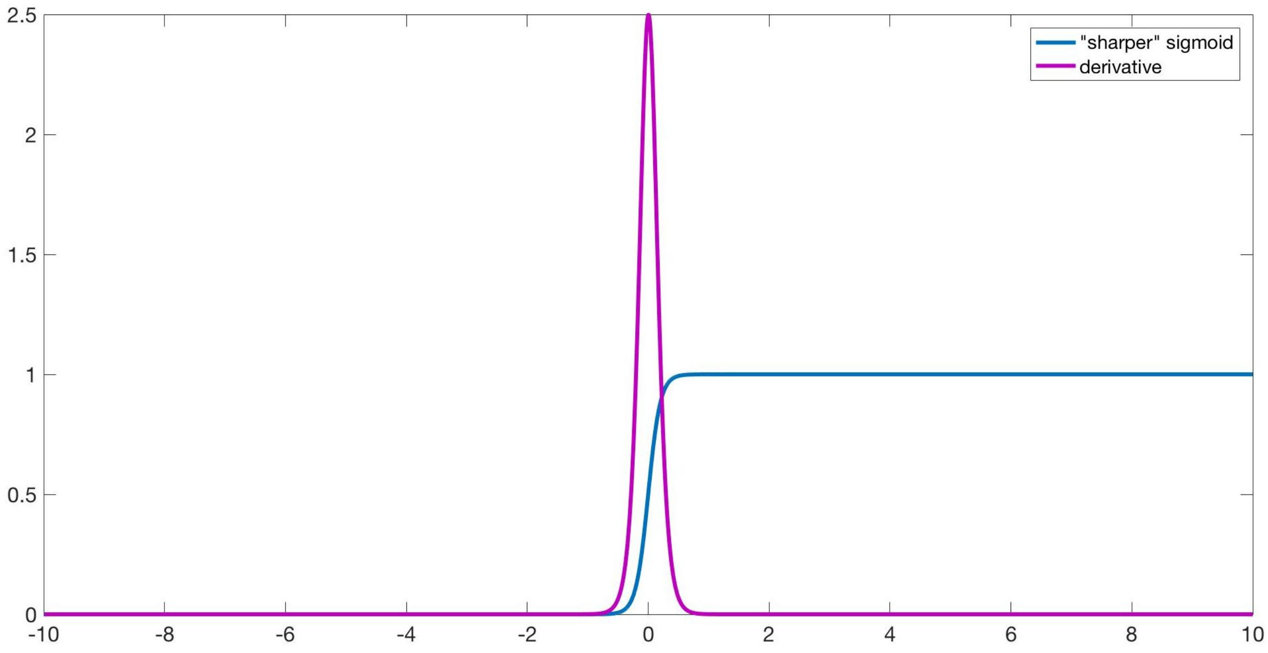
$$\Delta_j = g'(in_j) \sum_i W_{j,i} \Delta_i$$



Small gradient magnitude, particularly at the tails

Vanishing Gradient Problem

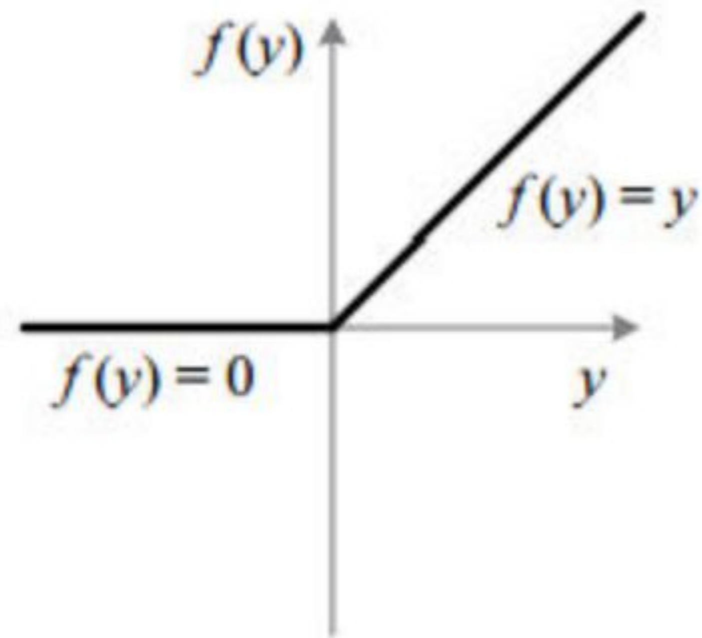
- Note that we could try to fix the problem by using non-linear activations that have gradients of larger magnitude
  - But if the magnitude gets larger than 1, then we have the problem of exploding gradients



Rectified Linear Units (RELU)

Nair & Hinton (2010)

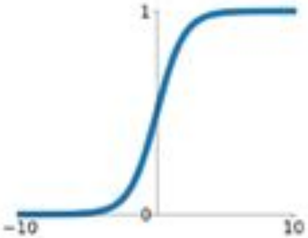
- Maximum gradient magnitude is 1
- Large region where this magnitude is 1
- Still non-linear
- Gradient shape?



Some Prevalent Activation Functions

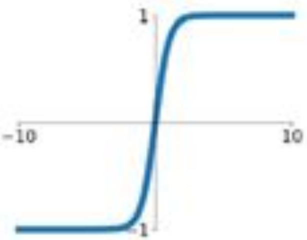
Sigmoid

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



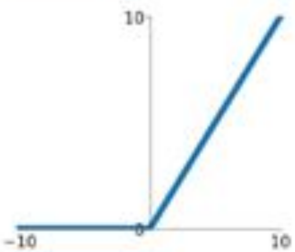
tanh

$\tanh(x)$



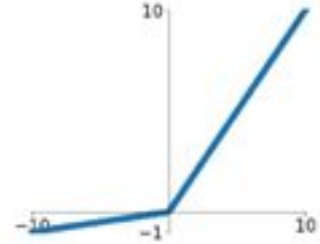
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

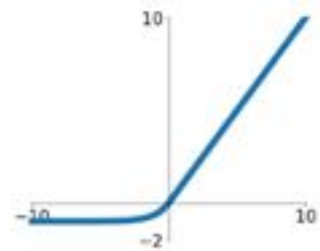


Maxout

$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Biggest Influence in Visual Perception Change in NN Architecture Deep Convolutional Neural Networks