



# Deep Learning Tutorial

AIT DL Workshop 2018

Dwijay Bane (R&D Engineer)



# Roadmap

- Supervised Learning with Neural Nets
- Convolutional Neural Networks for Object Recognition
- Recurrent Neural Network
- Other Deep Learning Models

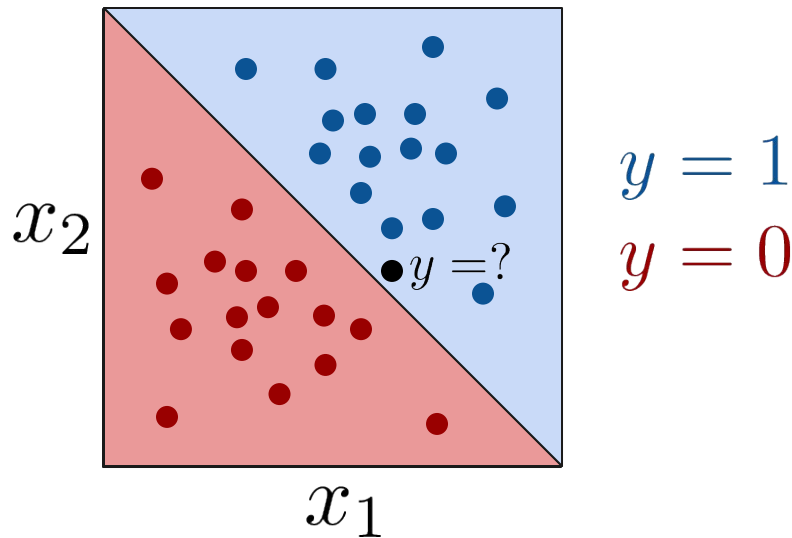
---

# Supervised Learning with Neural Nets

General references:  
Hertz, Krogh, Palmer 1991  
Goodfellow, Bengio, Courville 2016

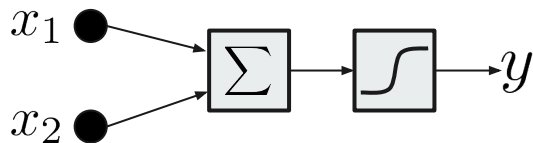
# Supervised learning

Given example input-output pairs  $(X, Y)$ ,  
learn to **predict output Y** from input X



Logistic regression, support vector machines, decision trees, neural networks...

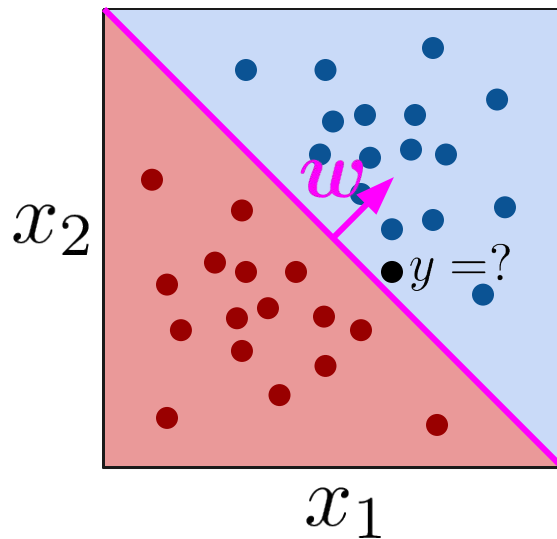
# Binary classification: simple perceptron



$$y = g \left( \sum_i w_i x_i + b \right) = g(\mathbf{w} \cdot \mathbf{x} + b)$$

(McCulloch & Pitts 1943)

$g$  is a nonlinear **activation function**, in this case  $g(z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z > 0 \end{cases}$

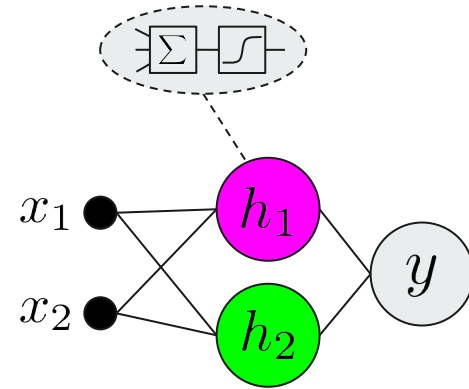
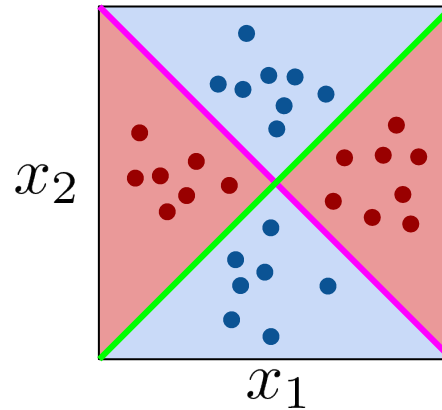
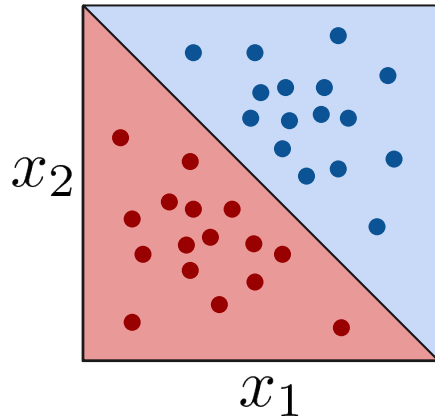


$$y = 1$$
$$y = 0$$

$$\mathbf{w}(k) = \mathbf{w}(k-1) + \eta [t(k) - y(k)] \mathbf{x}(k)$$

Perceptron learning rule  
(Rosenblatt 1962)

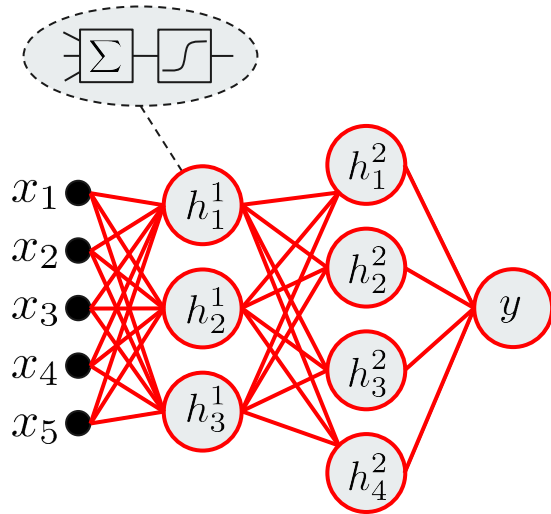
# Linear separability



**Simple perceptrons** can only learn to solve **linearly separable problems** (Minsky and Papert 1969).

We can solve more complex problems by composing many units in **multiple layers**.

# Multilayer perceptron (MLP)



$$h_i^1 = g(\mathbf{w}_i^1 \cdot \mathbf{x} + b_i^1)$$

$$h_i^2 = g(\mathbf{w}_i^2 \cdot \mathbf{h}^1 + b_i^2)$$

$$y = g(\mathbf{w}^3 \cdot \mathbf{h}^2 + b^3)$$

("forward propagation")

MLPs are **universal function approximators** (Cybenko 1989; Hornik 1989).  
(under some assumptions... exercise: show that if  $g$  is linear, this architecture reduces to a simple perceptron)



# Deep vs shallow

Universality: “shallow” MLPs with one hidden layer can represent any continuous function to arbitrary precision, given a large enough number of units. But:

- No guarantee that the number of required units is reasonably small (**expressivity**).
- No guarantee that the desired MLP can actually be found with our chosen learning method (**learnability**).

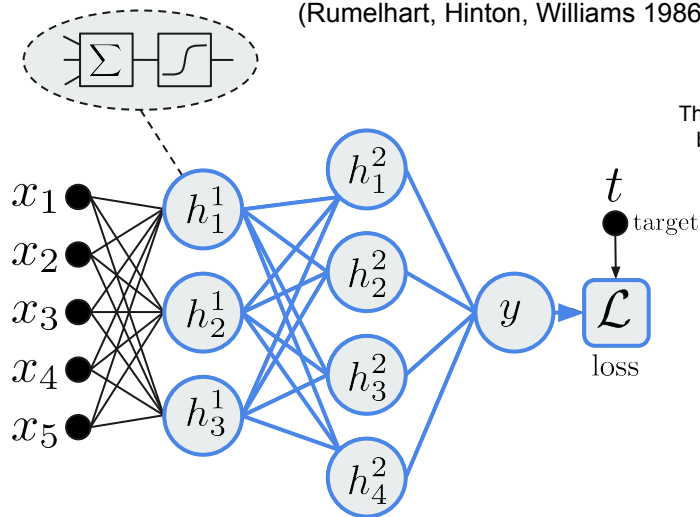
Two motivations for using **deep nets** instead (see Goodfellow et al 2016, section 6.4.1):

- **Statistical:** deep nets are **compositional**, and naturally well suited to representing hierarchical structures where simpler patterns are composed and reused to form more complex ones recursively. It can be argued that many interesting structures in real world data are like this.
- **Computational:** under certain conditions, it can be proved that deep architectures are more **expressive** than shallow ones, i.e. they can learn more patterns for a given total size of the network.



# Backpropagation

(Rumelhart, Hinton, Williams 1986)



Problem: compute all  $\partial \mathcal{L} / \partial w_{ik}^l$

Key insights: the loss depends

- on the weights  $w$  of a unit only through that unit's activation  $h$
- on a unit's activation  $h$  only through the activation of those units that are downstream from  $h$ .

The "errors" being backpropagated

$$\frac{\partial \mathcal{L}}{\partial y} \longrightarrow \frac{\partial \mathcal{L}}{\partial w_k^3} = \frac{\partial \mathcal{L}}{\partial y} \frac{\partial y}{\partial w_k^3}$$

$$\frac{\partial \mathcal{L}}{\partial h_i^2} = \frac{\partial \mathcal{L}}{\partial y} \frac{\partial y}{\partial h_i^2} \longrightarrow \frac{\partial \mathcal{L}}{\partial w_{ik}^2} = \frac{\partial \mathcal{L}}{\partial h_i^2} \frac{\partial h_i^2}{\partial w_{ik}^2}$$

$$\frac{\partial \mathcal{L}}{\partial h_i^1} = \sum_k \frac{\partial \mathcal{L}}{\partial h_k^2} \frac{\partial h_k^2}{\partial h_i^1} \longrightarrow \frac{\partial \mathcal{L}}{\partial w_{ik}^1} = \frac{\partial \mathcal{L}}{\partial h_i^1} \frac{\partial h_i^1}{\partial w_{ik}^1}$$

These give the gradient of the loss with respect to the weights, which you can then use with your favorite gradient descent method.

$$\mathcal{L} = \frac{1}{2}(y - t)^2$$

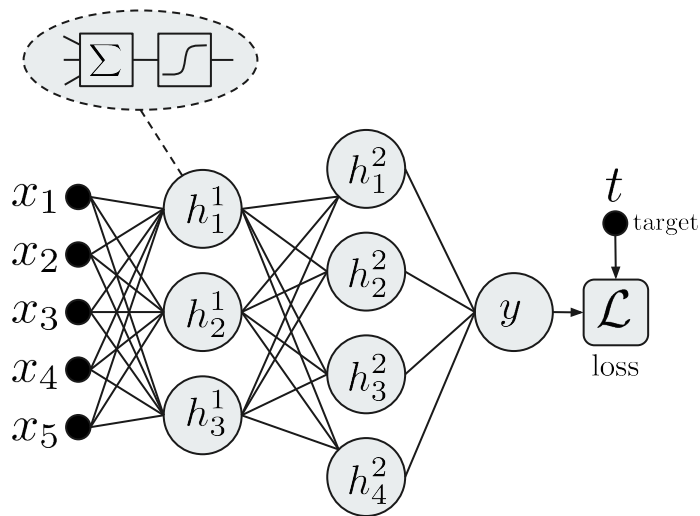
$$h_i^l = \tanh\left(\sum_k w_{ik}^l h_k^{l-1} + b_i^l\right)$$

$$\frac{\partial h_i^l}{\partial h_k^{l-1}} = (1 - (h_i^l)^2) w_{ik}^l$$

$$\frac{\partial h_i^l}{\partial w_{ik}^l} = (1 - (h_i^l)^2) h_k^{l-1}$$


$$\left(\text{recall that } \frac{d}{dz} \tanh(z) = 1 - z^2\right)$$

## Backpropagation - example



$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial y} &= \frac{\partial \mathcal{L}}{\partial y} t \longrightarrow \frac{\partial \mathcal{L}}{\partial w_{ik}^3} = \frac{\partial \mathcal{L}}{\partial y} \frac{\partial y}{\partial w_{ik}^3} = (1 - (h_i^2)^2) h_k^2 \\ \frac{\partial \mathcal{L}}{\partial h_i^2} &= \frac{\partial \mathcal{L}}{\partial y} \left( \pm \frac{\partial \mathcal{L}}{\partial y} \frac{\partial y}{\partial h_i^2} w_{ik}^3 \right) \longrightarrow \frac{\partial \mathcal{L}}{\partial w_{ik}^2} = \frac{\partial \mathcal{L}}{\partial h_i^2} \frac{\partial h_i^2}{\partial w_{ik}^2} = (1 - (h_i^1)^2) h_k^1 \\ \frac{\partial \mathcal{L}}{\partial h_i^1} &= \sum_k \frac{\partial \mathcal{L}}{\partial h_i^2} \frac{\partial h_i^2}{\partial h_i^1} w_{ik}^2 \longrightarrow \frac{\partial \mathcal{L}}{\partial w_{ik}^1} = \frac{\partial \mathcal{L}}{\partial h_i^1} \frac{\partial h_i^1}{\partial w_{ik}^1} = (1 - (h_i^0)^2) x_k \end{aligned}$$

(exercise: derive gradient wrt bias terms  $b$ )



The Navy revealed the embryo of an electronic computer today that it expects **will be able to walk, talk, see, write, reproduce itself and be conscious of its existence** [...]  
Dr. Frank Rosenblatt, a research psychologist at the Cornell Aeronautical Laboratory, Buffalo, said Perceptrons might be fired to the planets as mechanical space explorers.

The New York Times  
July 8th, 1958

The perceptron has shown itself worthy of study despite (and even because of!) its severe limitations. It has many features to attract attention: its linearity; its intriguing learning theorem; its clear paradigmatic simplicity as a kind of parallel computation. **There is no reason to suppose that any of these virtues carry over to the many-layered version.**  
Nevertheless, we consider it to be an important research problem to elucidate (or reject) our intuitive judgement that **the extension to multilayer systems is sterile.**

Minsky and Papert 1969  
(section 13.2)

---

# Convolutional Neural Networks for Object Recognition

General (excellent!) reference:  
“Convolutional Networks for Visual Recognition”, Stanford university  
<http://cs231n.stanford.edu/>

# Traditional Object Detection/Recognition Idea

- Match low-level vision features (e.g. edge, HOG, SIFT, etc)
- Parts-based models

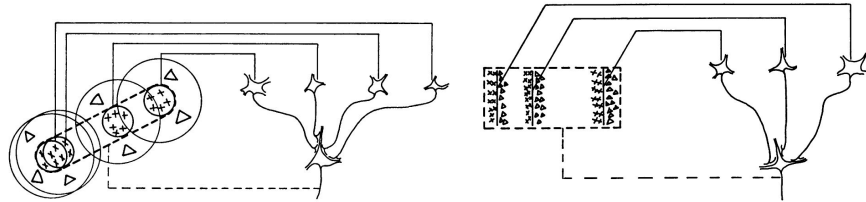


(Lowe 2004)

# Learning the features - inspiration from neuroscience

Hubel and Wiesel:

- Topographic organization of connections
- Hierarchical organization of simple/complex cells



(Hubel and Wiesel 1962)

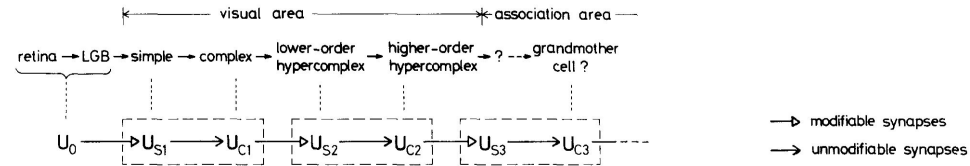


Fig. 1. Correspondence between the hierarchy model by Hubel and Wiesel, and the neural network of the neocognitron

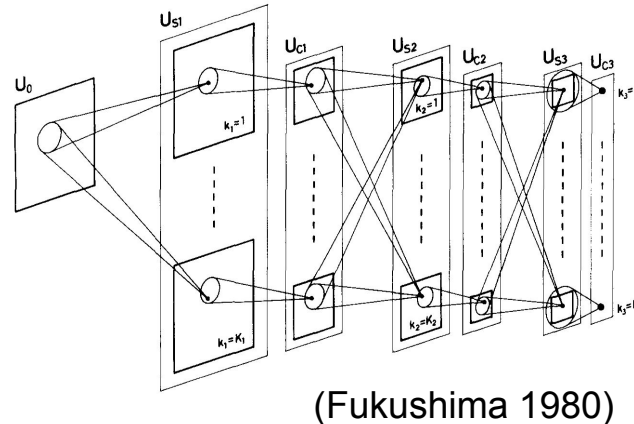


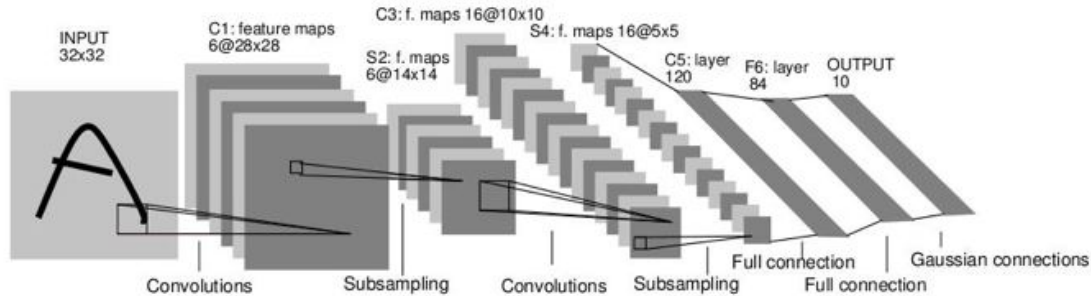
Fig. 2. Schematic diagram illustrating the interconnections between layers in the neocognitron

(Fukushima 1980)

# “Canonical” CNN structure

INPUT  $\rightarrow$  [[CONV  $\rightarrow$  RELU]\*K  $\rightarrow$  POOL?]\*L  $\rightarrow$  [FC  $\rightarrow$  RELU]\*M  $\rightarrow$  FC

Credit: [cs231n.github.io](https://github.com/cs231n)



(LeCun et al 1998)

Four basic operations:

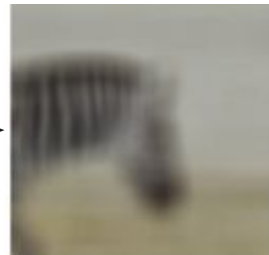
1. Convolution
2. Nonlinearity (ReLU)
3. Pooling
4. Fully connected layers

## 2D Convolution

Example: blurring an image



Replacing each pixel with an average of its neighbors





## 2D Convolution

$$f[m,n] = h \circ g = \sum_{k,l} h[m-k, n-l] g[k,l]$$

1 — 9	1	1	1
	1	1	1
	1	1	1

kernel / filter

0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0	0
0	0	0	90	90	90	90	90	0	0	0
0	0	0	90	90	90	90	90	0	0	0
0	0	0	90	0	90	90	90	0	0	0
0	0	0	90	90	90	90	90	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0

Input image


Output image

## 2D Convolution

$$f[m,n] = h \circ g = \sum_{k,l} h[m-k,n-l]g[k,l]$$

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

kernel / filter

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Input image

0	10								

Output image

## 2D Convolution

$$f[m,n] = h \circ g = \sum_{k,l} h[m-k,n-l]g[k,l]$$

$\frac{1}{9}$	1	1	1
	1	1	1
	1	1	1

kernel / filter

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	90	90	90	90	90	0
0	0	0	0	90	90	90	90	90	0
0	0	0	0	90	90	90	90	90	0
0	0	0	0	90	0	90	90	90	0
0	0	0	0	90	90	90	90	90	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Input image

	0	10	20						

Output image

## 2D Convolution

$$f[m,n] = h \circ g = \sum_{k,l} h[m-k, n-l] g[k,l]$$

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

kernel / filter

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Input image

	0	10	20	30	30				

Output image

# 2D Convolution

$$f[m,n] = h \circ g = \sum_{k,l} h[m-k,n-l]g[k,l]$$

$$\frac{1}{9}$$

1	1	1
1	1	1
1	1	1

kernel / filter

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Input image

	0	10	20	30	30	30	20	10	
	0	20	40	60	60	60	40	20	
	0	30	60	90	90	90	60	30	
	0	30	50	80	80	90	60	30	
	0	30	50	80	80	90	60	30	
	0	20	30	50	50	60	40	20	
	10	20	30	30	30	30	20	10	
	10	10	10	0	0	0	0	0	

Output image

# 2D Convolution

$$f[m,n] = h \circ g = \sum_{k,l} h[m-k,n-l]g[k,l]$$

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

kernel / filter

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Input image

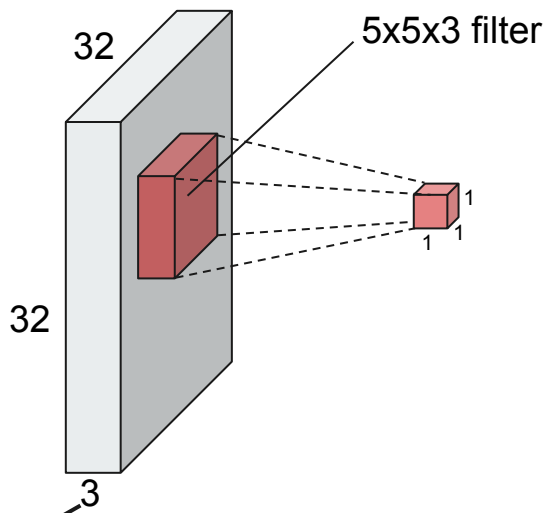
	0	10	20	30	30	30	20	10	
	0	20	40	60	60	60	40	20	
	0	30	60	90	90	90	60	30	
	0	30	50	80	80	90	60	30	
	0	30	50	80	80	90	60	30	
	0	20	30	50	50	60	40	20	
	10	20	30	30	30	30	20	10	
	10	10	10	0	0	0	0	0	

Output image

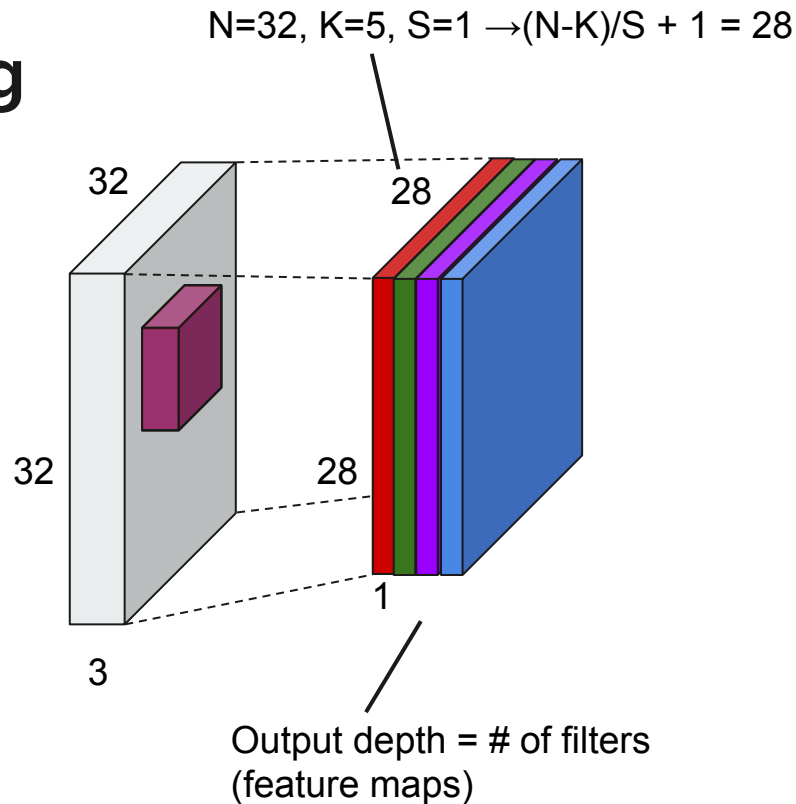
If N=input size, K=filter size, S=stride  
(stride is the size of the step you take  
on the input every time you move by  
one on the output)

$$\text{Output size} = (N-K)/S + 1$$

## More on convolution sizing



Input depth = # of channels in previous layer  
(often 3 **for input layer** (RGB); can be arbitrary  
for deeper layers)



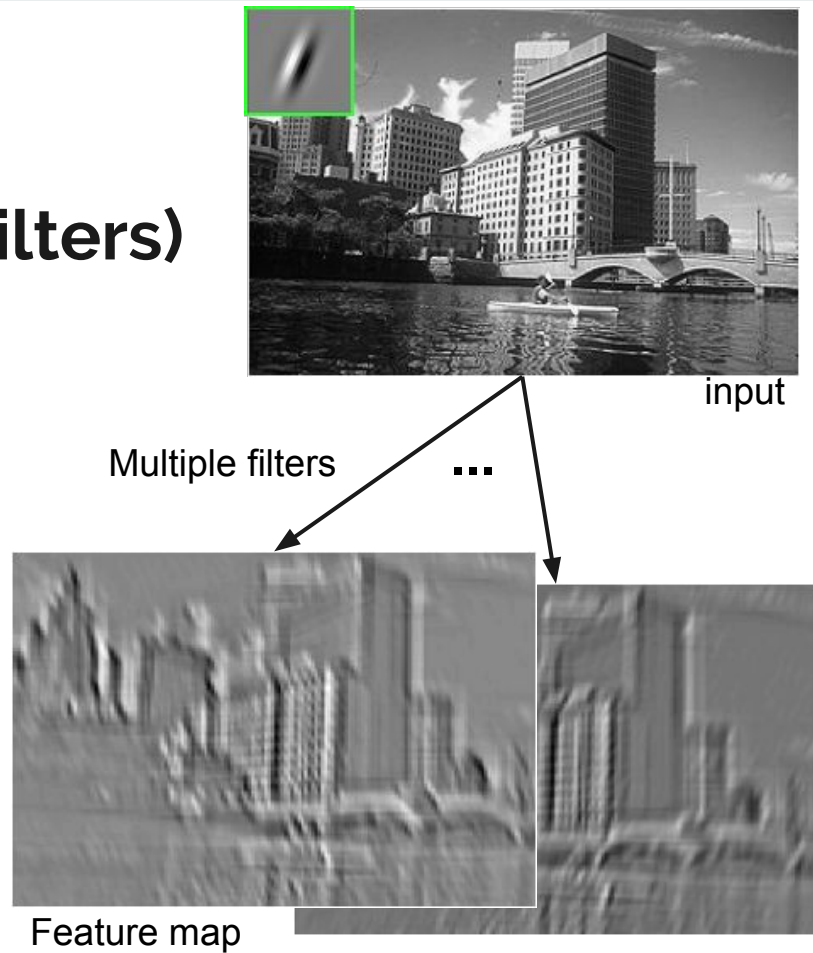
# Convolve with Different Filters



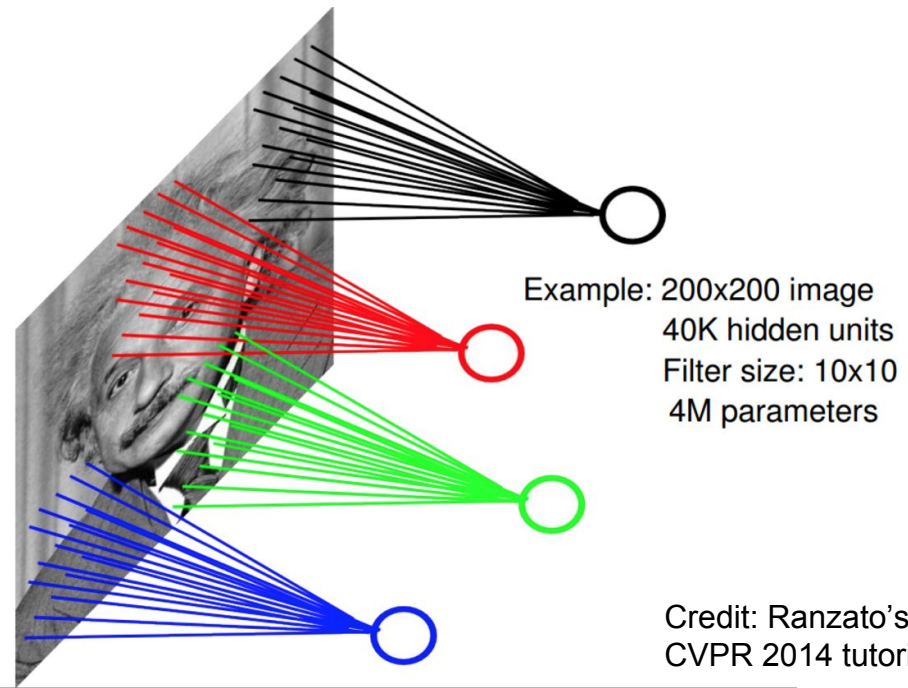
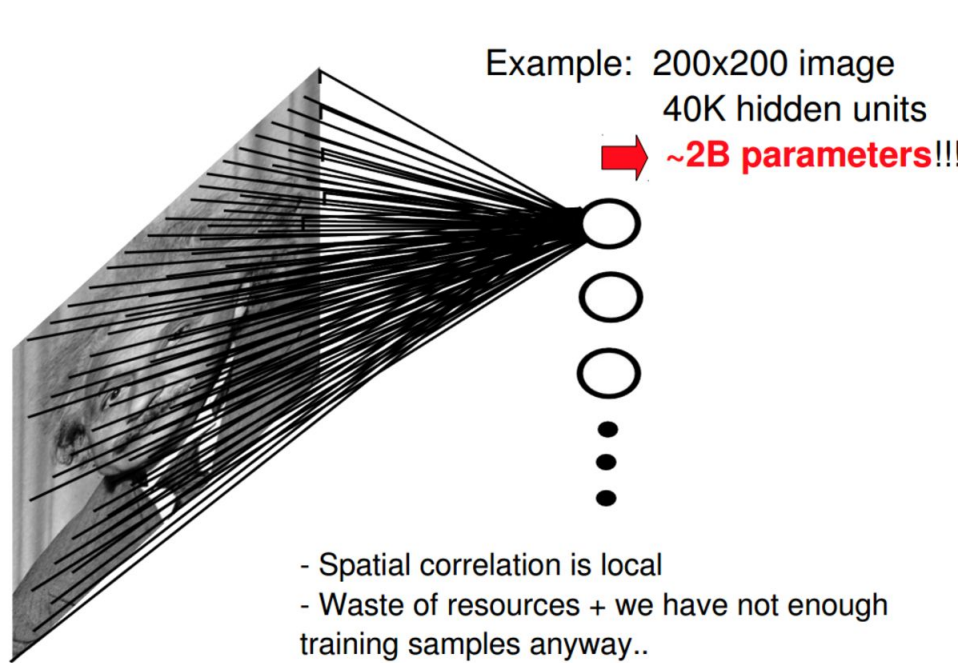


# Convolution (with learned filters)

- Dependencies are local
- Filter has few parameters to learn
  - Share the same parameters across different locations

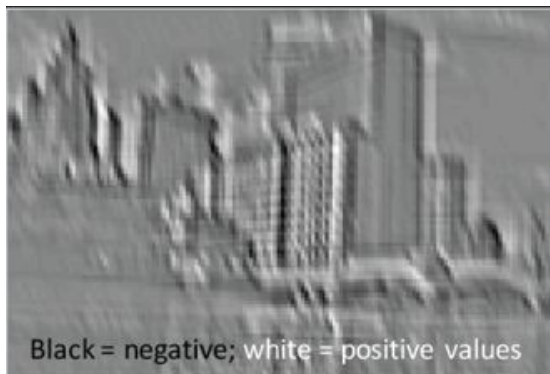
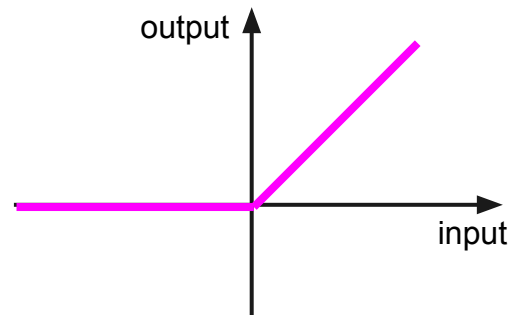


# Fully Connected vs. Locally Connected



# Non-linearity

- Rectified linear function (ReLU)
  - Applied per-pixel,  $\text{output} = \max(0, \text{input})$



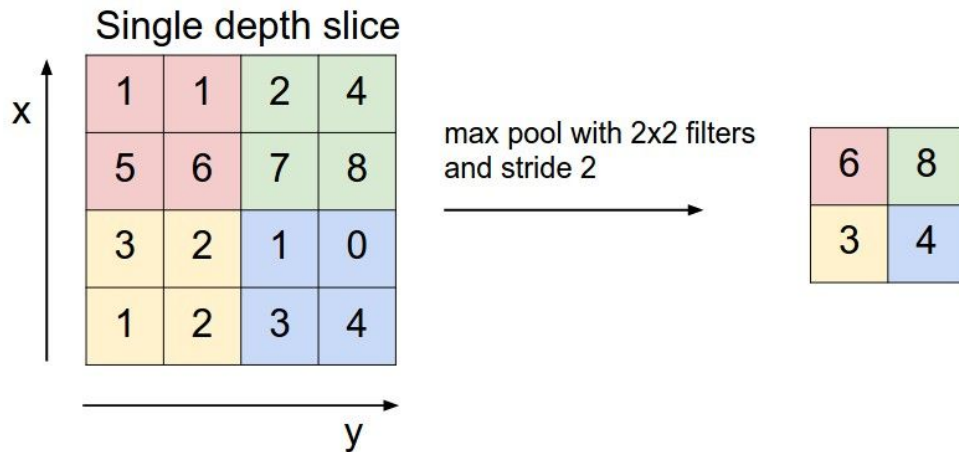
Input feature map



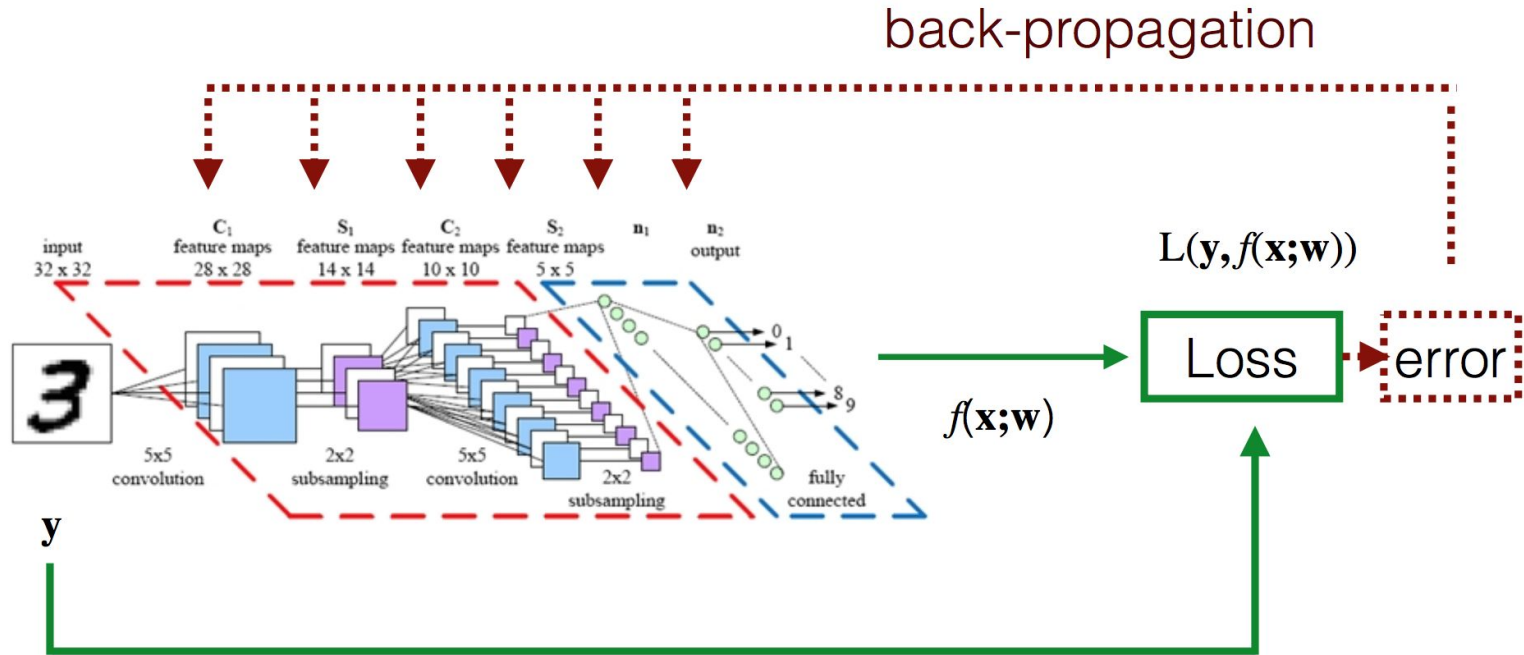
Output feature map

# Pooling

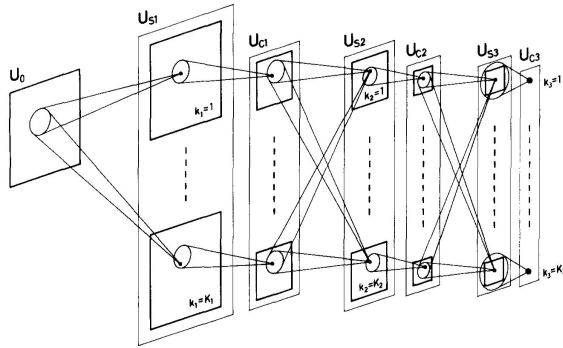
- Reduce size of representation in following layers
- Introduce some invariance to small translation



# Learning

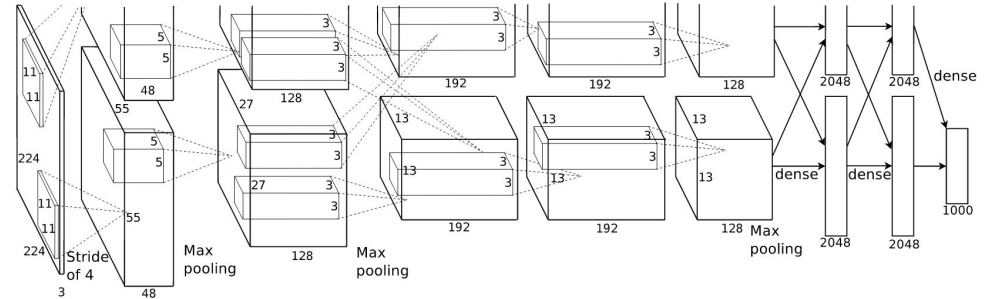
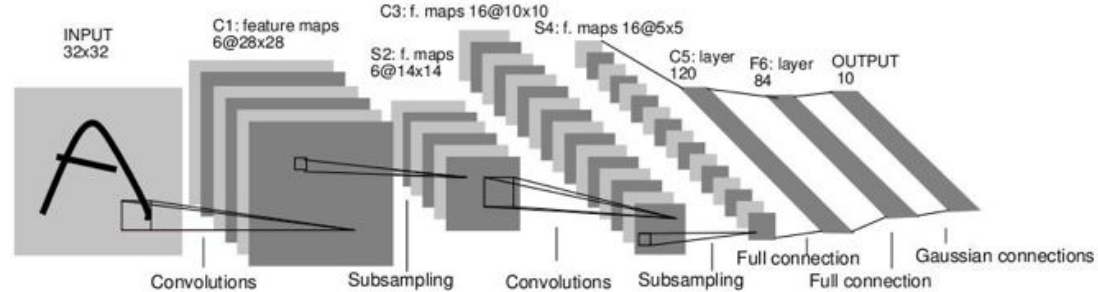


## Key evolutionary steps



**Neocognitron - Fukushima 1980**  
Inspired by Hubel and Wiesel  
“Convolutional” structure,  
alternating “pooling” layers

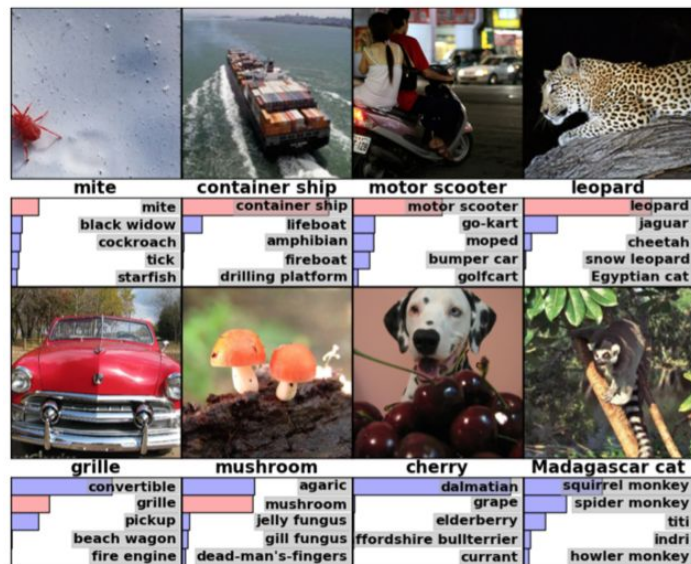
**LeNet - LeCun et al 1998**  
Backpropagation, gradient descent



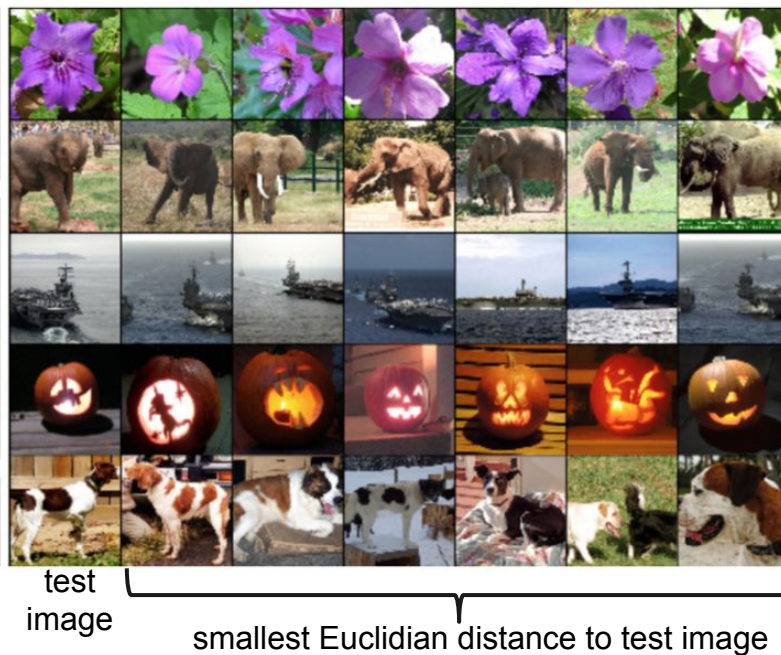
**AlexNet - Krizhevsky et al 2012**  
Larger, deeper network ( $\sim 10^7$  params), much more data (ImageNet -  $\sim 10^6$  images), more compute (incl. GPUs), better regularization (Dropout)



## Image classification



## Image retrieval



But also object detection, image segmentation, captioning...

---

# Recurrent Neural Network



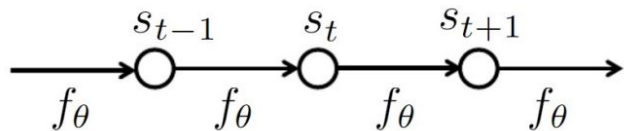


# Handling Sequential Information

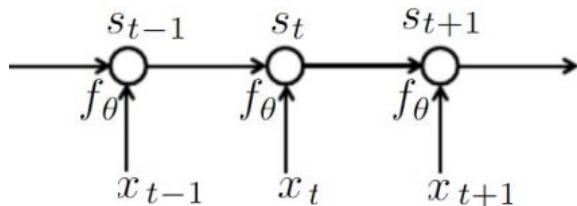
- **Natural language processing:** sentences, translations
- **Speech / Audio:** signal processing, speech recognition
- **Video:** action recognition, captioning
- **Sequential decision making / Planning**
- **Time-series data**
- **Biology / Chemistry:** protein sequences, molecule structures
- ...

# Dynamic System / Hidden Markov Model

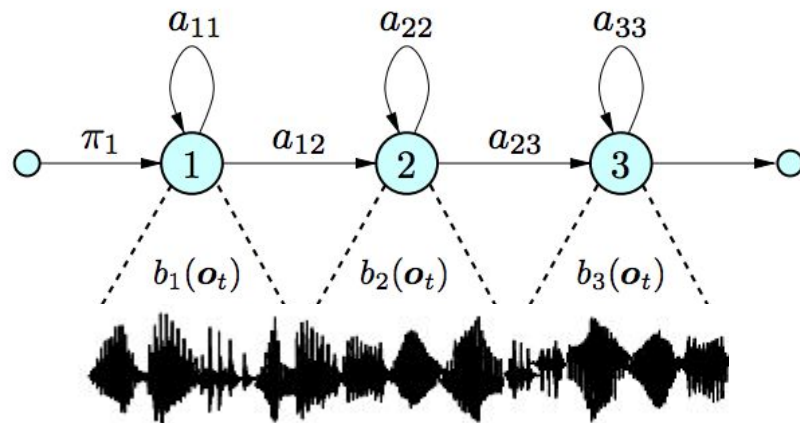
Classical form of a dynamic system



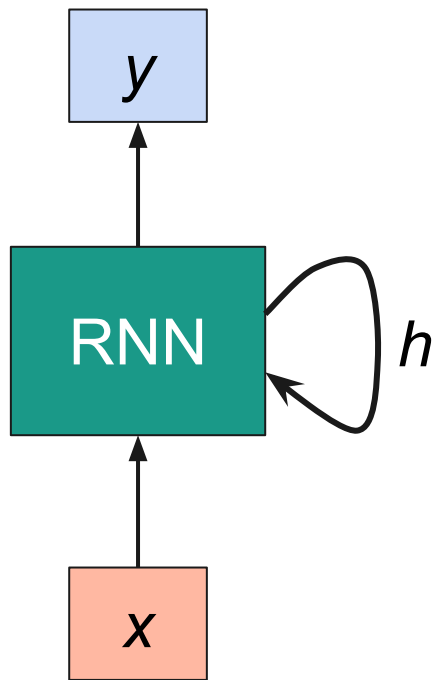
With an external signal  $x$



Hidden Markov Model



# Recurrent Network / RNN

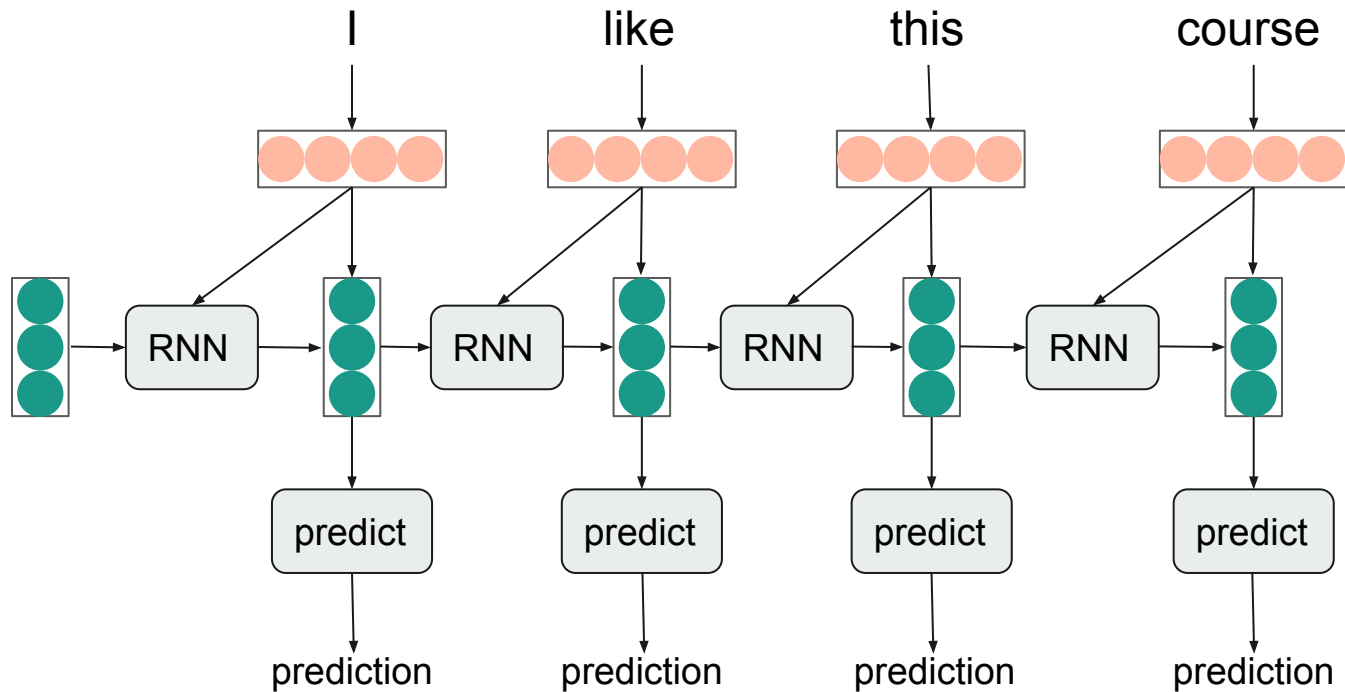


- A general form to process a sequence.
  - Applying a recurrence formula at each time step
- The state consists of a vector  $h$ .  
It summarizes input up to time  $t$ .

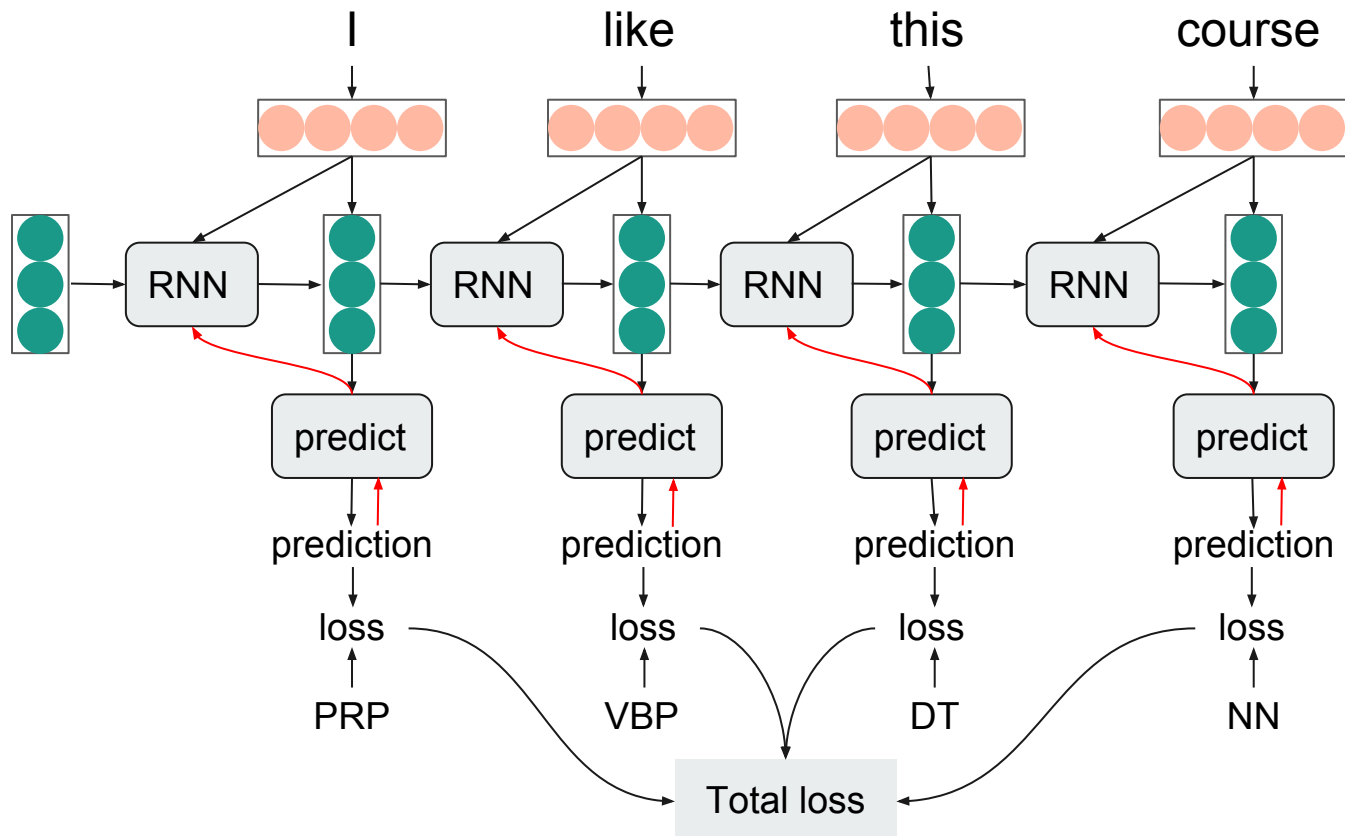
$$\underbrace{h_t}_{\text{New state}} = \underbrace{f_W}_{\text{A function with parameter } W}(\underbrace{h_{t-1}}_{\text{Old state}}, \underbrace{x_t}_{\text{Input at time } t})$$

$$y_t = W'h_t$$

# Processing a Sequence: Unrolling in Time

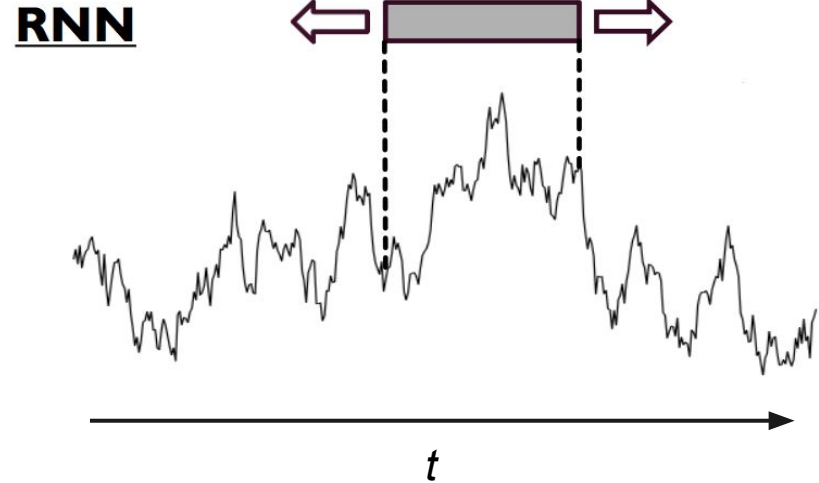
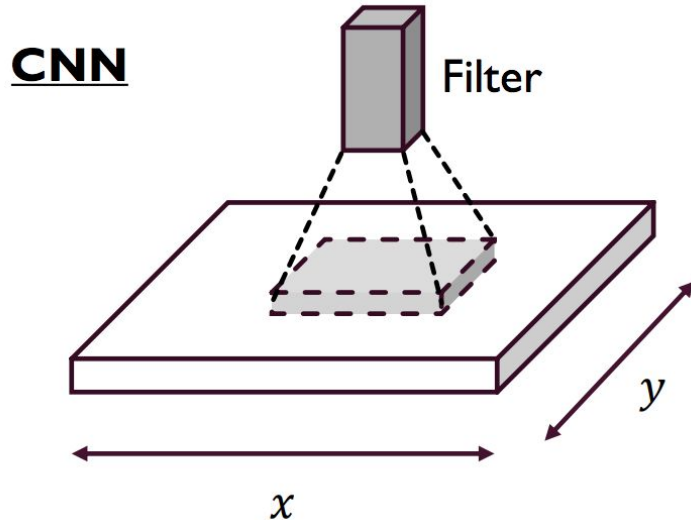


# Training: Backpropagation Through Time

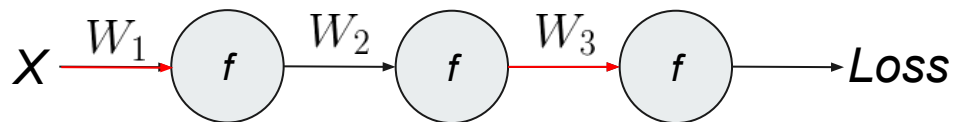


# Parameter Sharing Across Time

- The parameters are shared and derivatives are accumulated.
- Make it possible to generalize to sequences of different lengths.



# Vanishing Gradient



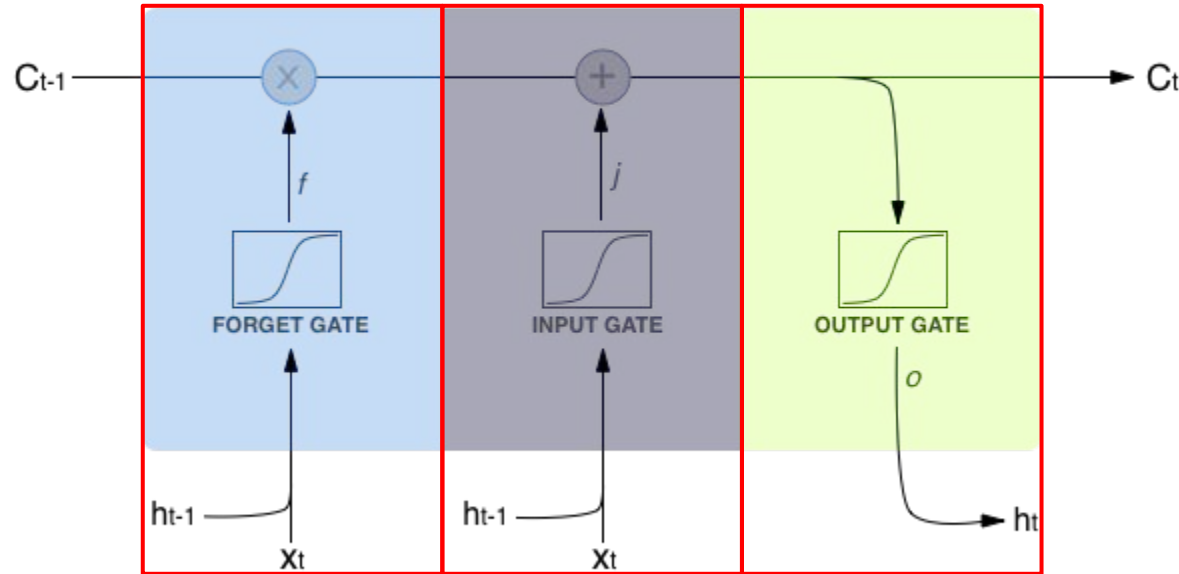
$$\frac{\partial Loss}{\partial W_3} = \frac{\partial Loss}{\partial f(x_3)} \frac{\partial f(x_3)}{\partial W_3}$$

$$\frac{\partial Loss}{\partial W_1} = \frac{\partial Loss}{\partial f(x_3)} \frac{\partial f(x_3)}{\partial f(x_2)} \frac{\partial f(x_2)}{\partial f(x_1)} \frac{\partial f(x_1)}{\partial W_1}$$

- $\frac{\partial f(x_t)}{\partial f(x_{t-1})}$  expanded quickly!
  - $|\cdot| > 1$ , gradient explodes → clipping gradients
  - $|\cdot| < 1$ , gradient vanishes → introducing memory via LSTMs, GRUs
- Have problem in learning long-term dependency.

# Long Short Term Memory (LSTM)

LSTM Memory Cell



Forget the irrelevant part of previous state

Selected update cell state values

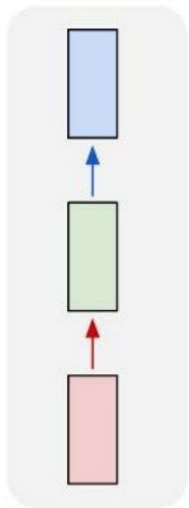
Output certain parts of the cell state

- Introducing gates to optionally let information flow through.
  - An LSTM cell has three gates to protect and control the **cell state**.

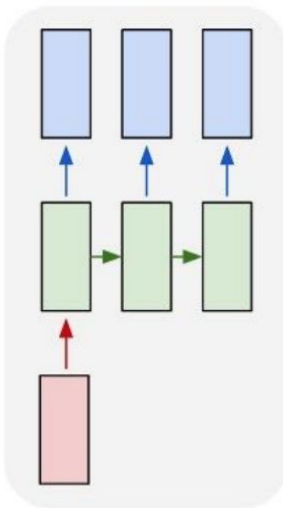


# Flexibility of RNNs

one to one

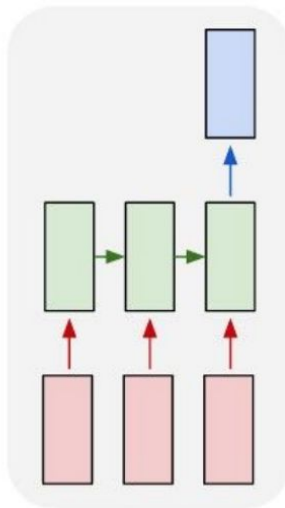


one to many



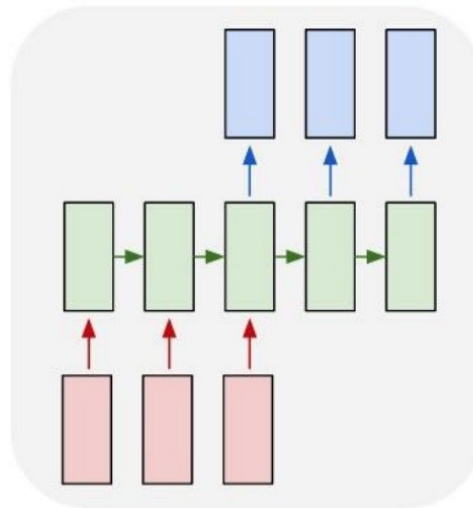
**Image  
Captioning**

many to one



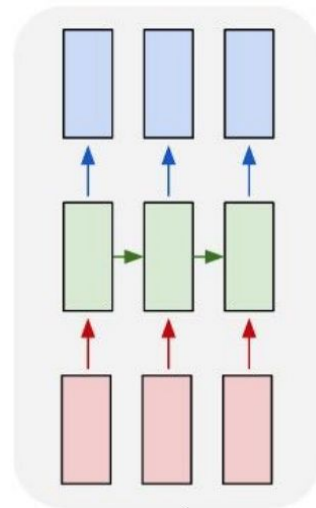
**Sentiment  
Classification**

many to many



**Machine  
Translation**

many to many



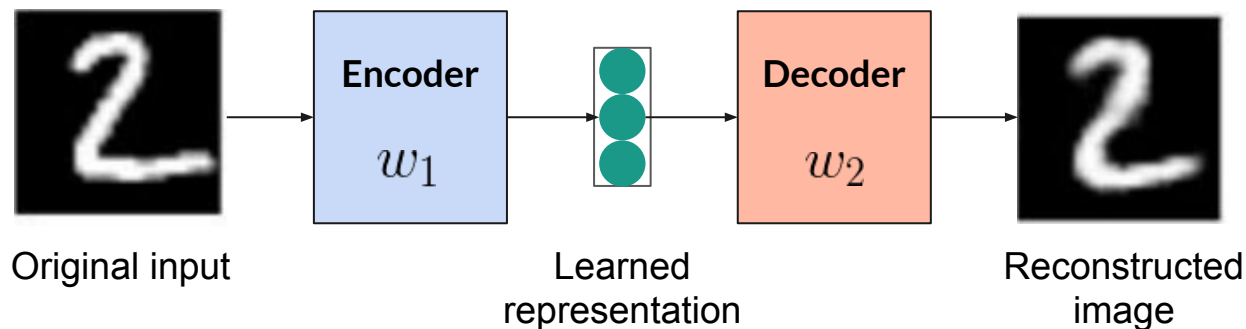
**POS  
Tagging**

---

# Other Deep Learning Models

# Auto-encoder

- Learning representations
  - a good representation should keep the information well



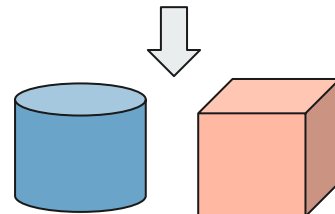
- → objective: minimize reconstruction error

$$\min_{w_1, w_2} \|x - \underbrace{f(g(x; w_1); w_2)}_{\text{reconstruction}}\|_2^2$$

# Generative Models

- What are the learned representations?
  - One view: latent variables to generate the observed data
- Goal of learning a generative model: to recover  $p(\mathbf{x})$  from data

latent variables:  
color, shape, position, ...



observed data

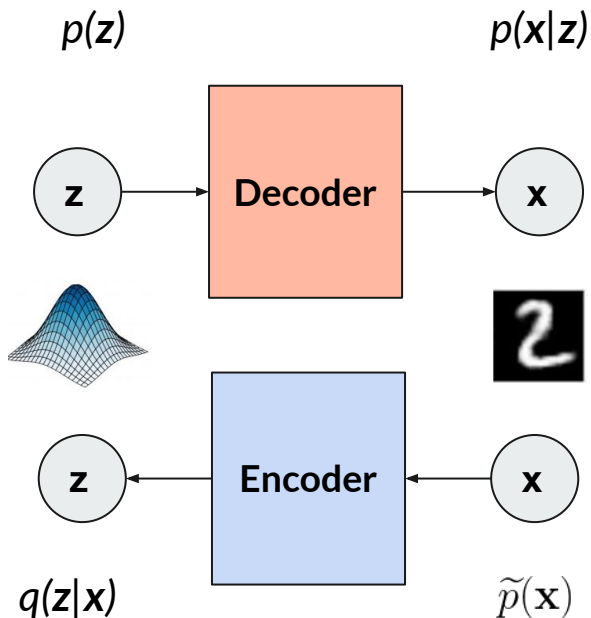
## Desirable properties

Sampling new data  
Evaluating likelihood of data  
Extracting latent features

## Problem

Directly computing  
$$p(\mathbf{x}) = \int p_{\theta}(\mathbf{x}|\mathbf{z})p_{\theta}(\mathbf{z})d\mathbf{z}$$
  
is intractable!

# Variational Autoencoder (VAE)



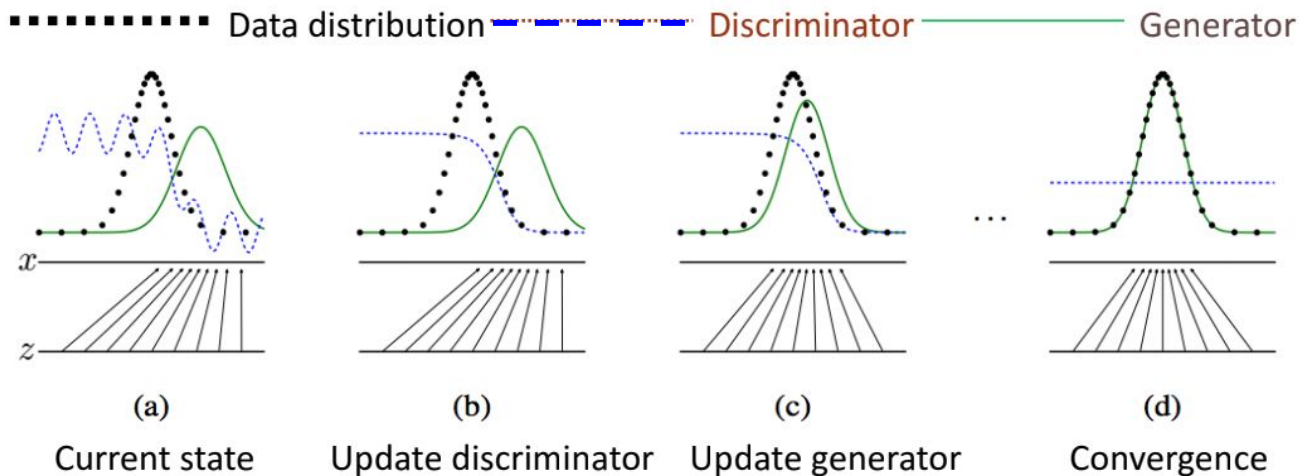
- Idea: approximate  $p(z|x)$  with a simpler, tractable  $q(z|x)$
- Learning objective

$$l_i(\theta, \phi) = -\underbrace{\mathbb{E}_{z \sim q_\phi(z|x_i)} [\log(p_\theta(x_i|z))]}_{\text{Reconstruction error}}$$

$$+ \underbrace{KL(q_\phi(z|x_i) \parallel p(z))}_{\text{Measure how close } q \text{ is to } p}$$

# Generative Adversarial Network (GAN)

- An implicit generative model, formulated as a minimax game.
  - The discriminator is trying to distinguish real and fake samples.
  - The generator is trying to generate fake samples to fool the discriminator.





# Thanks & Questions?

- 
- Hands-on session !!!

Dwijay Bane ([dwijay@edutechlearning.com](mailto:dwijay@edutechlearning.com))

Edutech Learning Solutions Pvt. Ltd. ([info@edutechlearning.com](mailto:info@edutechlearning.com))