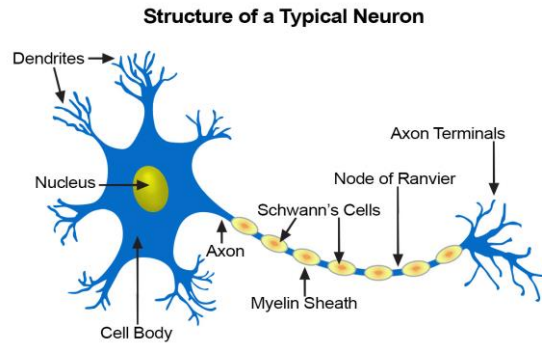


## Neural Networks

---



## Artificial Neural Networks

---

- An Artificial Neural Network is a mathematical model for learning inspired by biological neural networks.
- Artificial neural networks model mathematical functions that map inputs to outputs based on the structure and parameters of the network.
  - In artificial neural networks, the structure of the network is shaped through training on data
- Each artificial neuron is considered to be a **unit** that is connected to other units.
  - Earlier we called simple neurons – perceptrons
- The mathematical function was linear:
  - e.g.  $h(x_1, x_2) = w_0 + w_1x_1 + w_2x_2$ , where  $w_0$  is referred to as bias

## Perceptron Model: Linear separation

n inputs --- separated by a n-1 dimensional plane

Weight Vector  $\vec{w} : (w_0, w_1, w_2)$

Input Vector  $\vec{x} : (1, x_1, x_2)$

$w \bullet x : w_0 + w_1x_1 + w_2x_2$

$\vec{w} \cdot \vec{x} \geq 0$  dot product:  $\vec{w} \cdot \vec{x} = \sum_{i=0}^n w_i * x_i$

$h_w(\vec{x}) = 1$  if  $\vec{w} \cdot \vec{x} \geq 0$  Rain  
0 otherwise Not Rain

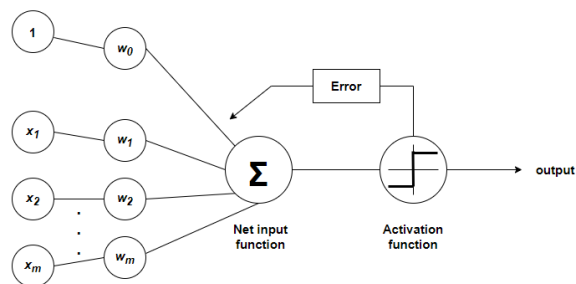
Perceptron Learning Rule: Learn from the data points one at a time to improve h

$w_{i \text{ new}} = w_{i \text{ old}} + \alpha(\text{actual value} - \text{estimated value}) * x_i$

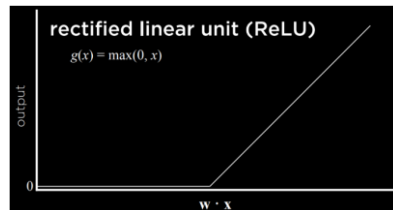
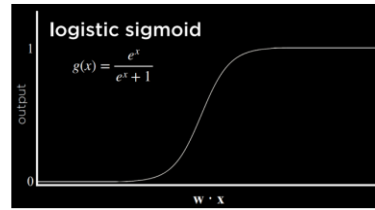
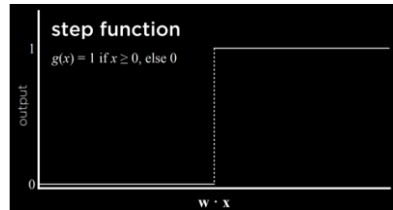
$w_{i \text{ new}} = w_{i \text{ old}} + \alpha(y - h(\vec{x})) * x_i$   $\alpha$ : learning rate

## Perceptron Learning Rule

<https://www.educative.io/answers/what-is-the-perceptron-learning-rule>



## Activation Functions



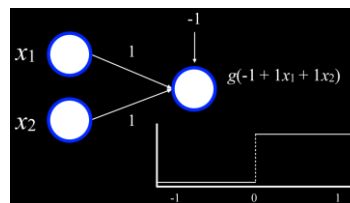
$$\text{softmax}(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^N e^{z_j}}$$

5

## What can be computed?

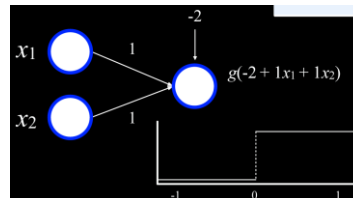
- Logical OR

X	Y	$X \vee Y$
T	T	T
T	F	T
F	T	T
F	F	F



- Logical AND

X	Y	$X \wedge Y$
T	T	T
T	F	F
F	T	F
F	F	F



6

## Loss Functions

---

- Loss Function: A function that quantifies the difference between predicted and actual values in a machine learning model. It guides the optimization process by providing feedback on how well it fits the data.
- Examples:
  - Classification Loss Function: Cross-Entropy/Logistic Loss (CE)  
Measures the distance from the actual class to the predicted value, which is usually a real number between 0 and 1
  - Regression Loss function: Mean Square Error

$$MSE = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2$$

## Gradient Descent in Neural Networks

---

- Gradient descent is an algorithm for minimizing loss when training neural networks.
- The weights along with the loss or activation function determine the behavior of a neural net.
- A neural network is capable of inferring knowledge about the structure of the network itself from the data
- The weights are based on the training data.

## Linear Regression and Classification: Gradient descent

---

- search through a continuous weight space by incrementally modifying the parameters (minimizing loss)
- $\alpha$ : step size/learning rate that can be a fixed constant or decay over time

```

w ← any point in the parameter space
while not converged do
  for each  $w_i$  in w do
     $w_i \leftarrow w_i - \alpha \frac{\partial}{\partial w_i} \text{Loss}(\mathbf{w})$ 

```

$$\frac{\partial}{\partial w_0} \text{Loss}(\mathbf{w}) = -2(y - h_{\mathbf{w}}(x)) \quad \frac{\partial}{\partial w_1} \text{Loss}(\mathbf{w}) = -2(y - h_{\mathbf{w}}(x)) \times x$$

## Using Gradient Descent

---

### Gradient Descent Algorithm

- Start with a random choice of weights. This is the usual situation.
- Repeat:
  - Calculate the gradient based on all data points that will lead to decreasing loss.
  - Update weights according to the gradient.
- But this can be too time consuming instead base calculation on options like:
  - Stochastic Gradient Descent – compute based on a single random point
  - Mini-Batch Gradient Descent – compute based on a few points selected at random
- Need to find a balance between accuracy and computational complexity.
- Lot of different approaches exist

## Multilayer Neural Networks

---

- A multilayer neural network is an artificial neural network with an input layer, an output layer, and at least one hidden layer.
- Inputs and outputs are still used to train the model.
- But each unit in the first hidden layer receives a weighted value as outputs of units in the input layer
- Then the neuron in this layer performs some calculations using these inputs and outputs a value. Each of these values is weighted and further propagated to the next layer.
- Repeating the process until the output layer is reached.
- Through hidden layers, it is possible to model non-linear data.

*In fact, with some assumptions, it has been shown that any mathematical function can be approximated by a neural network*

## Simple Feedforward Networks

---

### Feedforward network

- connections only in one direction (input to output)
- directed acyclic graph with designated input and output nodes (No loops)

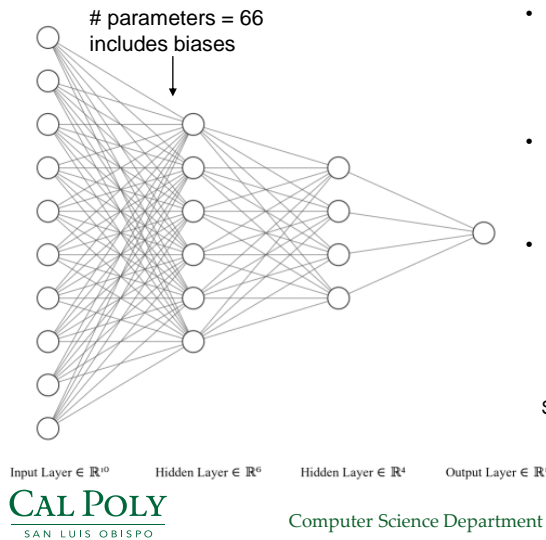
### Networks as complex functions

- Each node within a network is called a unit
- calculates the weighted sum of the inputs from predecessor nodes
- applies a nonlinear function to produce its output

$$a_j = g_j(\sum_i w_{i,j} a_i) \equiv g_j(in_j),$$

- $g_j$  is a nonlinear **activation function**,  $a_j$  denotes the output of unit  $j$  and  $w_{i,j}$  is the weight attached to the link from unit  $i$  to unit  $j$ ;

## Training a multilayer network - Architecture



- Usually, each neuron in the hidden layer uses an activation function like sigmoid or rectified linear unit (ReLU).
- This helps to capture the non-linear relationship between the inputs and their outputs.
- The neurons in the output layer also use activation functions like sigmoid (for regression) or SoftMax (for classification)

See Neptune.ai for lots of details

13

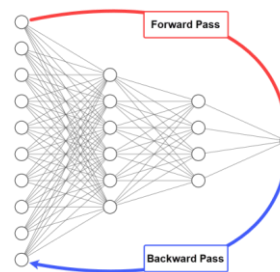
13

## Training a multilayer network - Approach

To train a neural network, there are 2 phases:

### **Forward propagation**

1. propagate data inputs into the input layer
2. Go through the hidden layers
3. Measure the predictions at the output layer and calculate the network error



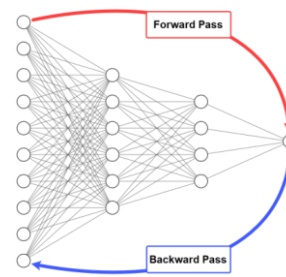
See Neptune.ai for lots of details

14

## Training a multilayer network - Approach

### Backward propagation

1. Calculate the gradient of the loss function with respect to the weights
2. Use the gradient to determine which input weights need to be changed and by how much
3. Use this to measure the error in the  $i-1$  first layer



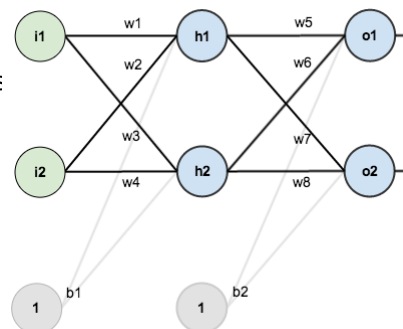
See Neptune.ai for lots of details

15

## Tutorial: Back Propagation

<https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/>

- Example: a neural network with two inputs, two hidden neurons, two output neurons. Additionally, the hidden and output neurons will include a bias.
- Goal: optimize weights to so Neural Net **correctly** maps inputs to outputs

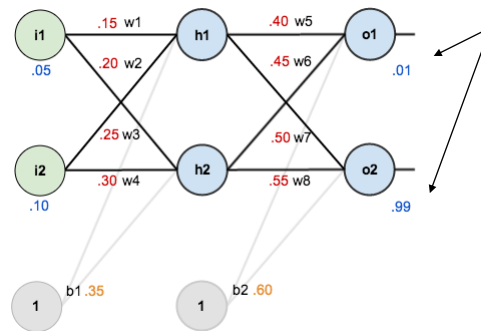


16



## Data: Make the example concrete

- In order to have some numbers to work with, here are the initial weights, the biases, and training inputs/outputs:



17

## Forward Pass: Compute output of hidden layer

- What does this neural network predict given the weights and biases above and inputs of 0.05 and 0.10. To do this we'll feed those inputs forward through the network.

$$net_{h1} = w_1 * i_1 + w_2 * i_2 + b_1 * 1$$

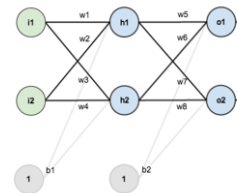
$$net_{h1} = 0.15 * 0.05 + 0.2 * 0.1 + 0.35 * 1 = 0.3775$$

using the logistic function to get the output of  $h_1$ :

$$out_{h1} = \frac{1}{1+e^{-net_{h1}}} = \frac{1}{1+e^{-0.3775}} = 0.593269992$$

Carrying out the same process for  $h_2$  we get:

$$out_{h2} = 0.596884378$$



18

## Output layer output

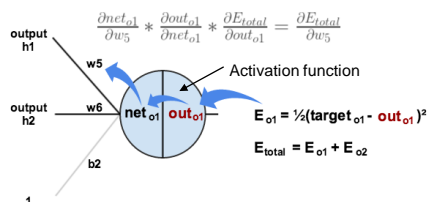
- $O_1$   $net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$   
 $net_{o1} = 0.4 * 0.593269992 + 0.45 * 0.596884378 + 0.6 * 1 = 1.105905967$   
 $out_{o1} = \frac{1}{1+e^{-net_{o1}}} = \frac{1}{1+e^{-1.105905967}} = 0.75136507$
- $O_2$   $out_{o2} = 0.772928465$
- Calculate the error for each output neuron using the squared error function and sum them to get the total error:  
 $E_{total} = \sum \frac{1}{2}(target - output)^2$   
 $E_{o1} = \frac{1}{2}(target_{o1} - out_{o1})^2 = \frac{1}{2}(0.01 - 0.75136507)^2 = 0.274811083$   
 $E_{o2} = 0.023560026$   
 $E_{total} = E_{o1} + E_{o2} = 0.274811083 + 0.023560026 = 0.298371109$

## Backwards Pass

- The goal with backpropagation is to update each of the weights in the network so that they cause the actual output to be closer the target output, thereby minimizing the error for each output neuron and the network as a whole.
- Consider  $w_5$ : How much does a change in  $w_5$  affect total error  
 $\frac{\partial E_{total}}{\partial w_5}$  “the partial derivative of  $E_{total}$  with respect to  $w_5$ ” or  
 “the gradient with respect to  $w_5$ ”

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5}$$

Applying the chain rule



## Backwards Pass

Applying the chain rule  $\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5}$

$$\frac{\partial E_{total}}{\partial out_{o1}} = 2 * \frac{1}{2} (target_{o1} - out_{o1})^{2-1} * -1 + 0$$

$$\frac{\partial E_{total}}{\partial out_{o1}} = -(target_{o1} - out_{o1}) = -(0.01 - 0.75136507) = 0.74136507$$

$$out_{o1} = \frac{1}{1 + e^{-net_{o1}}}$$

$$\frac{\partial out_{o1}}{\partial net_{o1}} = out_{o1}(1 - out_{o1}) = 0.75136507(1 - 0.75136507) = 0.186815602$$

$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$

$$\frac{\partial net_{o1}}{\partial w_5} = 1 * out_{h1} * w_5^{(1-1)} + 0 + 0 = out_{h1} = 0.593269992$$

Finally

$$\frac{\partial E_{total}}{\partial w_5} = 0.74136507 * 0.186815602 * 0.593269992 = 0.082167041$$

21

To decrease the error: subtract from the current weight (perhaps multiplied by a learning rate)

Old Weights:  $w_5 = .40$ ,  $w_6 = .45$ ,  $w_7 = .50$ ,  $w_8 = .55$ ,

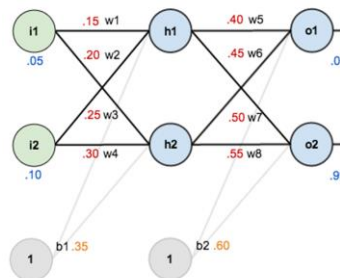
$$w_5^+ = w_5 - \eta * \frac{\partial E_{total}}{\partial w_5} = 0.4 - 0.5 * 0.082167041 = 0.35891648$$

$$w_6^+ = 0.408666186$$

$$w_7^+ = 0.511301270$$

$$w_8^+ = 0.561370121$$

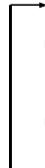
Lots more in the article



22

## Training Cycle

---

- Start with a random choice of weights
- 
- Forward propagation
  - Backward propagation

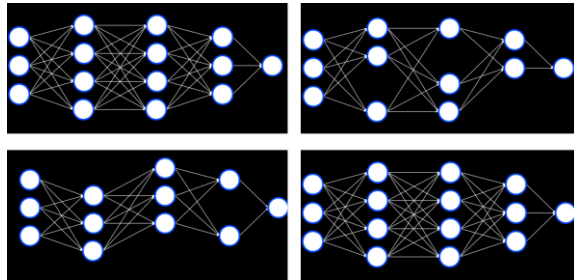
## Deep Neural Networks: Overfitting

---

- Deep neural networks, which are neural networks that have more than one hidden layer.
- Very effective but lots of care must be taken in training
- Major issue is **Overfitting**: Model is too specific to the training data. E.g. Some parameters specific to a few data points may be having too large an influence.
- The model will fail to generalize to new data

## Overfitting: Dropout in training

- Dropout temporarily remove units that we select at random during the learning phase. This way, we try to prevent over-reliance on any one unit in the network. Throughout training, the neural network will assume different forms, each time dropping some other units and then using them again.



## Future Attractions:

- Computer Vision: finding features, edge detection
- Convolution Neural Nets
- Classification using NNs: Digit recognition
- Recurrent NNs: Image captioning
- Language
  - Legal Sentences: Context Free Grammars, Parsing
  - Bag of Words: Sentiment Analysis
  - Word2Vec
  - Word Embedding
  - Attention
  - Transformer Architecture
  - Positional Encoding

## Computer Vision: Images and Information

---

- “Vision is based on inference”
- An image can be represented as a matrix of numbers
  - We do not process  $10^5$  to  $10^6$  numbers
- What makes us perceive objects in images?
  - Hypothesis: process images at low level
  - Extract “features”
  - Combine features with prior knowledge to classify objects in the image at a high-level
- What features should (or do) we use?

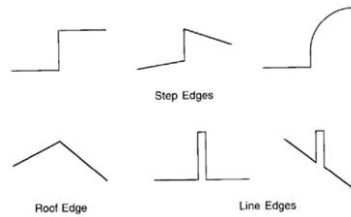
## Image Features

---

- Edges
- Corners
- Texture
- Distribution of light
- Features across multiple images (flow)
- ...

## Edge detection

1. Detection of short linear edge segments (**edgels**)
2. Aggregation of **edgels** into extended edges
3. Possibly combine the edges



Edge is where change occurs - where biggest derivative!

## Idea: Gradient measures greatest change

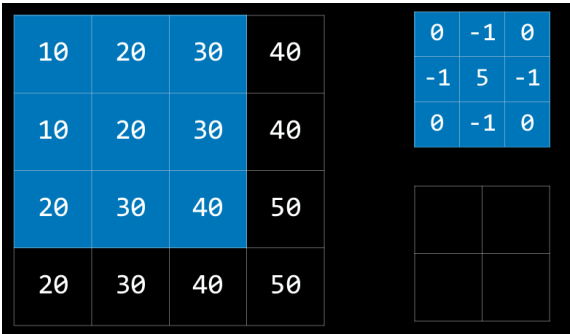
- How can we measure the gradient from a matrix of numbers
- E.g. Sobel Operators
  - Compute derivatives in the x and y directions ("gradient like")

$$\frac{1}{8} \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad \frac{1}{8} \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

$s_x$                        $s_y$

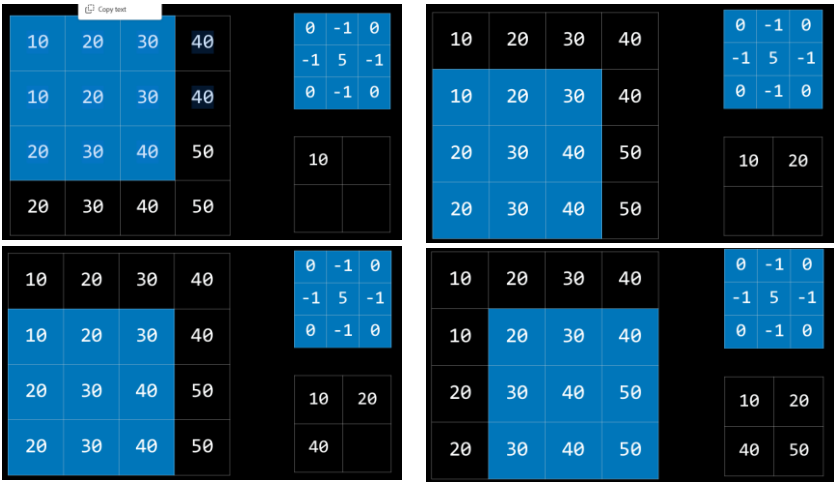
# Image Convolution

- applying a filter that adds each pixel value of an image to its neighbors, weighted according to a kernel matrix



31

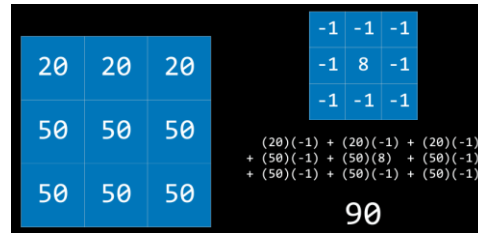
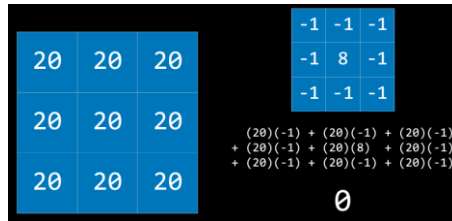
# Computer Vision



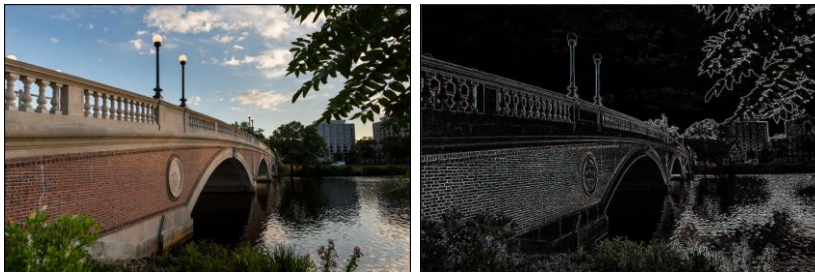
32



## Different Kernels – pull out different aspects



## Edge detection kernel at work

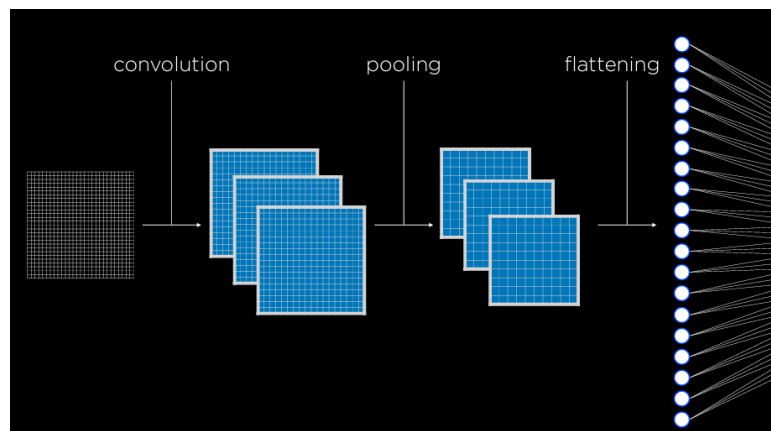


## Pooling and down-sampling

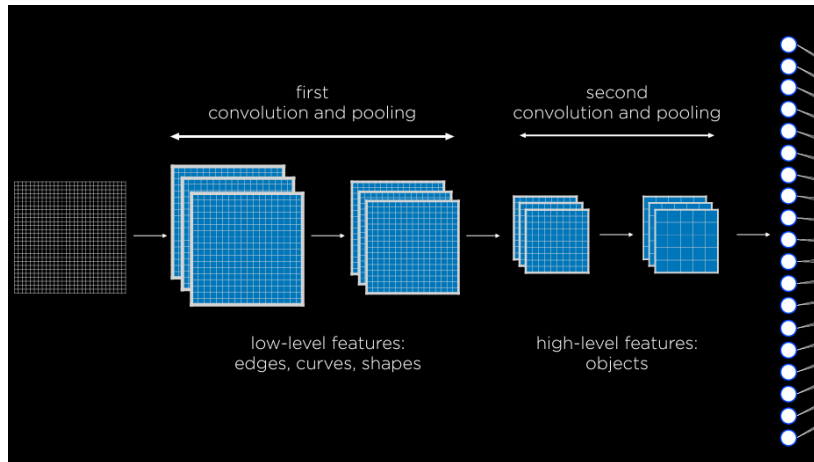
- One issue in large deep neural networks is the computational load. Images contain many pixels.
- A pooling layer in a neural network summarizes a set of adjacent units from the preceding layer with a single value. Common forms of pooling:
  - Average-pooling: computes the average value of its  $l$  inputs
    - coarsen the resolution of the image, downsample by a factor of  $s$ .
    - Ideally, classifier can still recognize in features in the pooled image: that is it facilitates multiscale recognition
  - Max-pooling: computes the maximum value of its  $l$  inputs.
    - Can be used for downsampling
    - logical disjunction, saying that a feature exists somewhere in the unit's receptive field.

## Convolutional Neural Network

- neural networks that use convolution, usually for analyzing images



## Layers in network allow steps in detecting higher level features



37

## Example: Digit recognition using tensor flow (1:14:30 in CS50 – 5)

- Read CS50 Neural Network notes and follow the digit recognition example which starts at 1:14:30 in CS50 lecture 5 on Neural Nets

38

## Recall Feed-Forward

- Input data is provided to the network, which eventually produces some output



- Need a different architecture for NN to deal with sequences of either inputs or outputs.
  - Text
  - Audio
  - Video
  - ...

## Recurrent Neural Networks

- Allows the network to have **state**:
  - Units may take as input values that are computed from their own output at earlier time steps
  - Allows results from an earlier step to affect the remaining computation/outputs

