Neural networks

- Neural networks are made up of **nodes** or **units**, connected by **links**
- Each link has an associated **weight** and **activation level**
- Each node has an **input function** (typically summing over weighted inputs), an **activation function**, and an **output**

Biological Inspiration

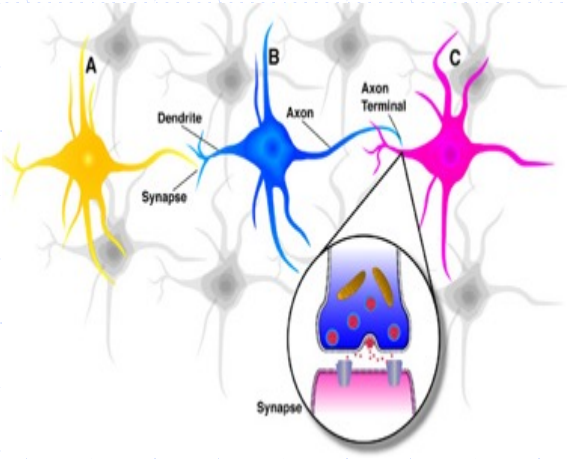
- Idea: To make the computer more robust, intelligent, and learn, …
- Let's model our computer software (and/or hardware) after the brain

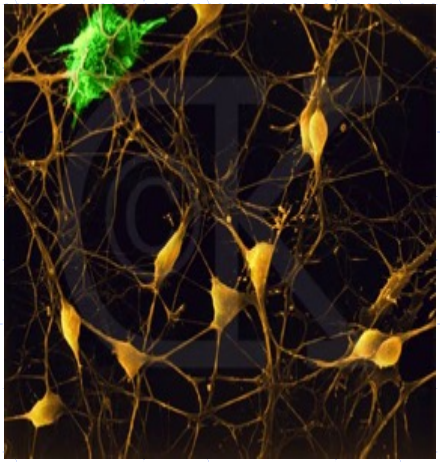
"My brain: It's my second favorite organ."

- Woody Allen, from the movie Sleeper

Neurons in the Brain

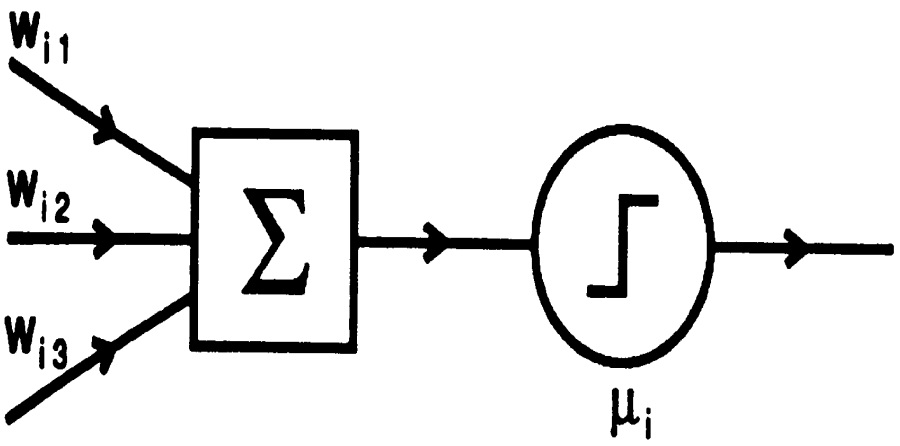
- Although heterogeneous, at a low level the brain is composed of neurons
  - A neuron receives input from other neurons (generally thousands) from its synapses
  - Inputs are approximately summed
  - When the input exceeds a threshold the neuron sends an electrical spike that travels that travels from the body, down the axon, to the next neuron(s)



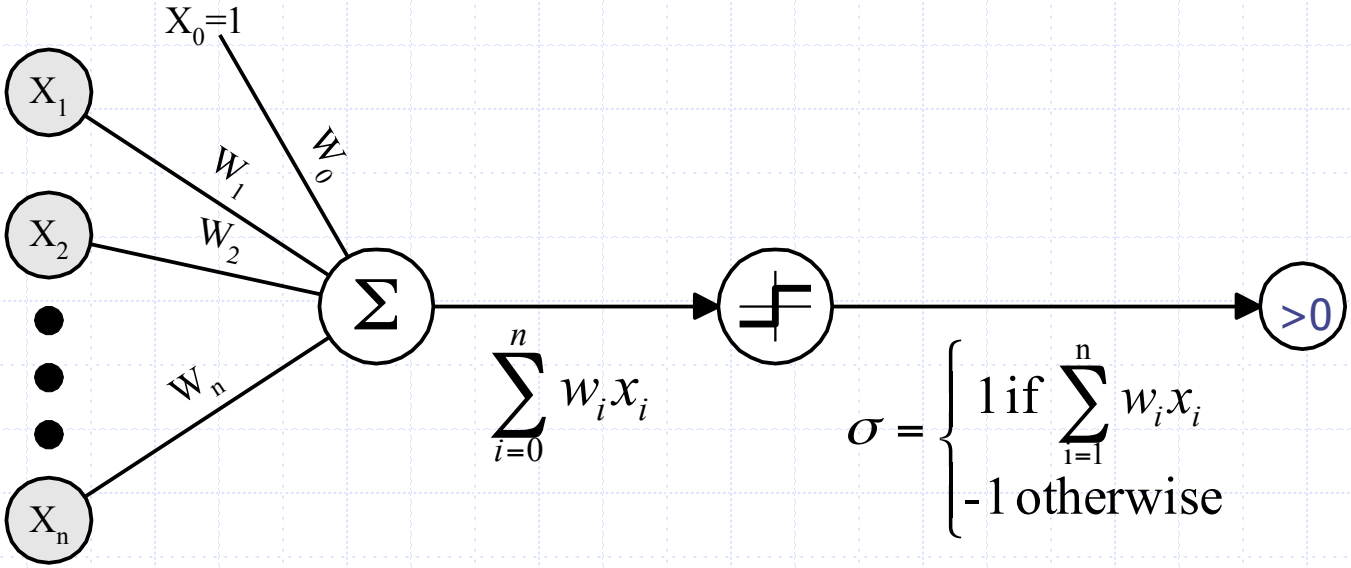


McCulloch–Pitts "neuron" (1943)

- Attributes of neuron
  - m binary inputs and 1 output (0 or 1)
  - Synaptic weights w_{ij}
  - Threshold u_i



Perceptron

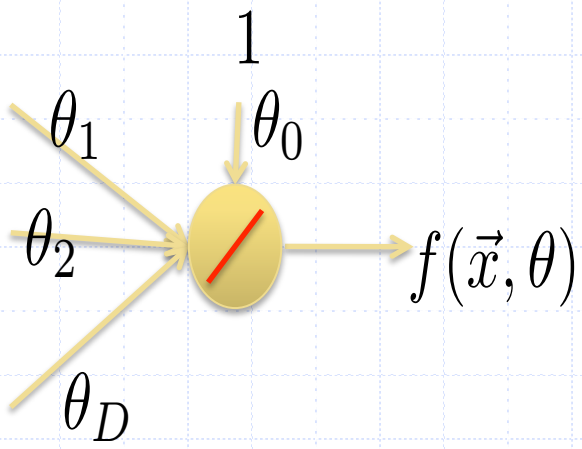


$$o(x_1,\ldots,x_n) = \begin{cases} 1 \text{ if } w_0 + w_1 x_1 + \ldots + w_n x_n > 0 \\ -1 \text{ otherwise} \end{cases}$$

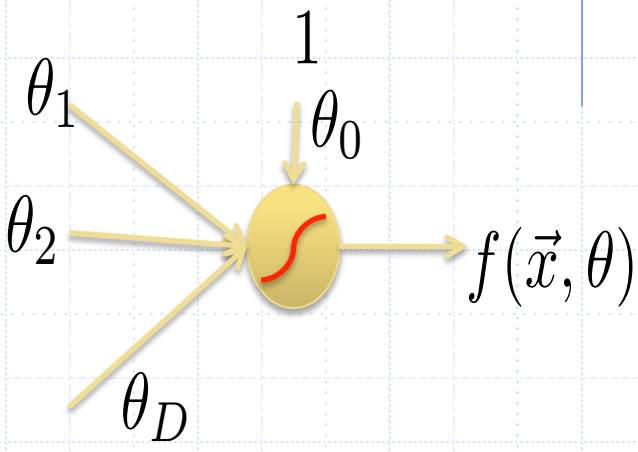
in other words :

$$o(\vec{x}) = \begin{cases} 1 \text{ if } \vec{w} \cdot \vec{x} > 0 \\ -1 \text{ otherwise} \end{cases}$$

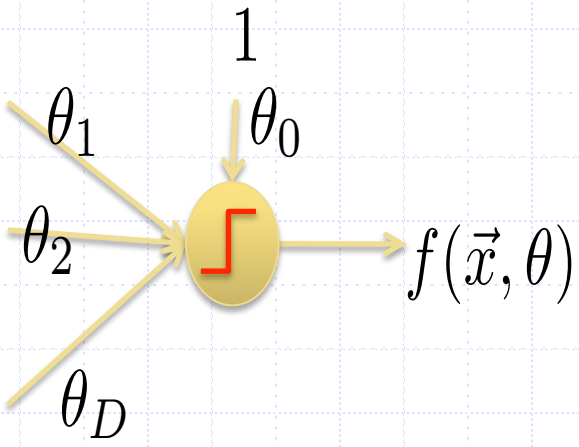
Types of Neurons



Linear Neuron



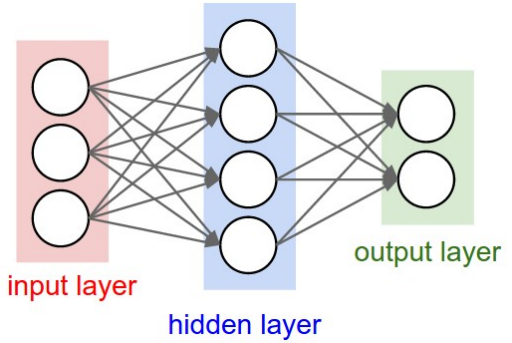
Logistic Neuron



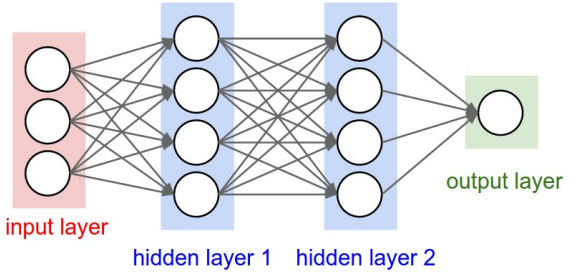
Perceptron

Artificial neural network

- Collection of connected neurons
- Feed-forward network
  - Graph is acyclic

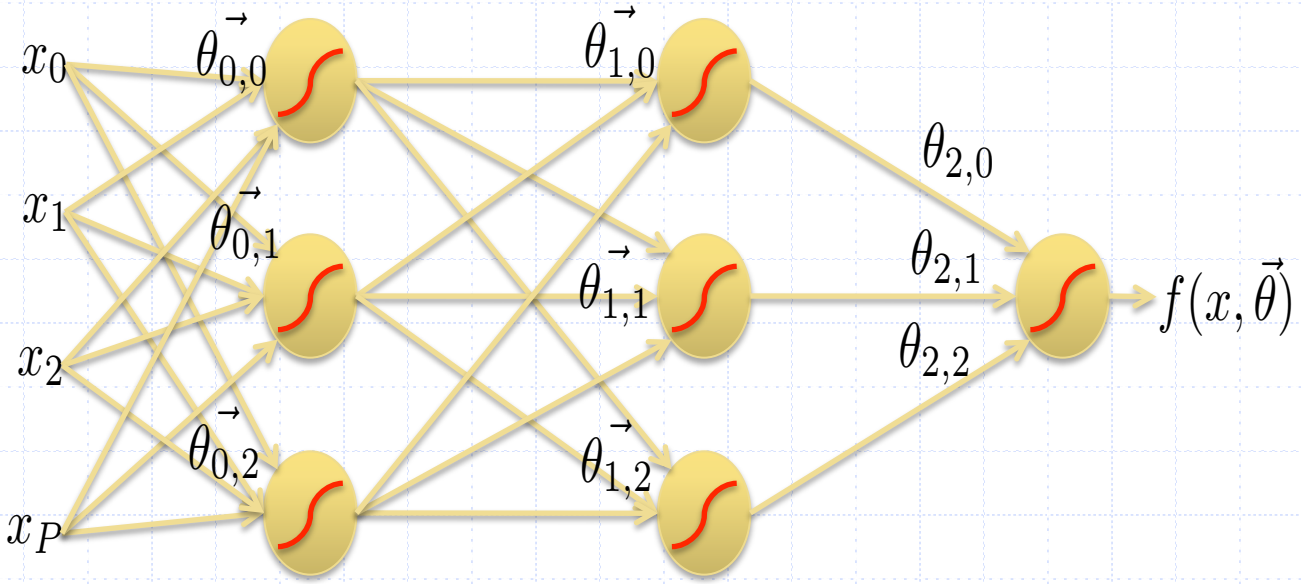


- Architecture? Weights?



- Examples:
  - Multilayer perceptron (MLP)
  - Convolutional neural network (CNN)

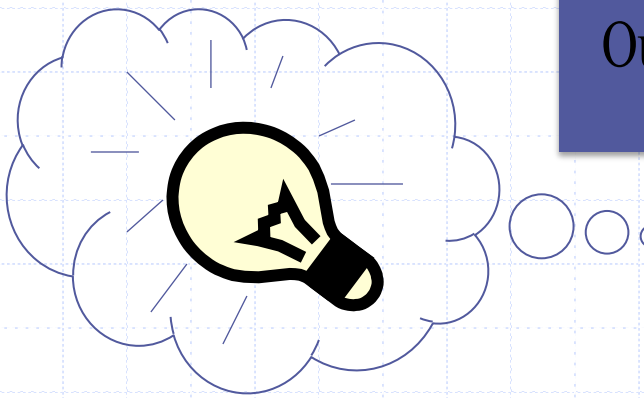
Multilayer Networks



Multilayer Networks

- The class of functions representable by perceptrons is limited

$$\mathrm{Out}(\mathbf{x}) = g\left(\mathbf{w}^{\mathrm{T}}\mathbf{x}\right) = g\left(\sum_j w_j x_j\right)$$

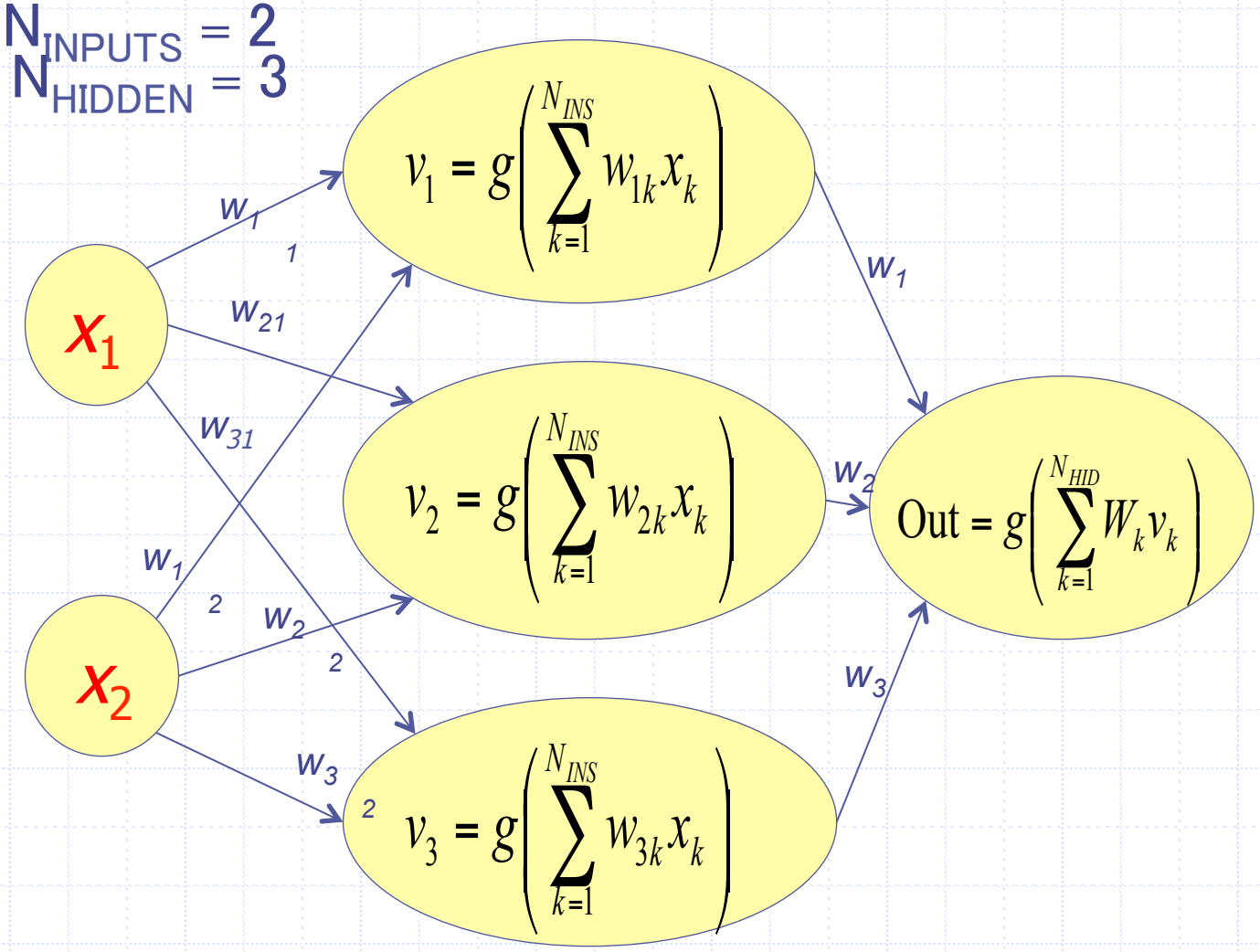


Use a wider representation !

$$\mathrm{Out}(\mathbf{x}) = g\left(\sum_j W_j g\left(\sum_k w_{jk} x_{jk}\right)\right)$$

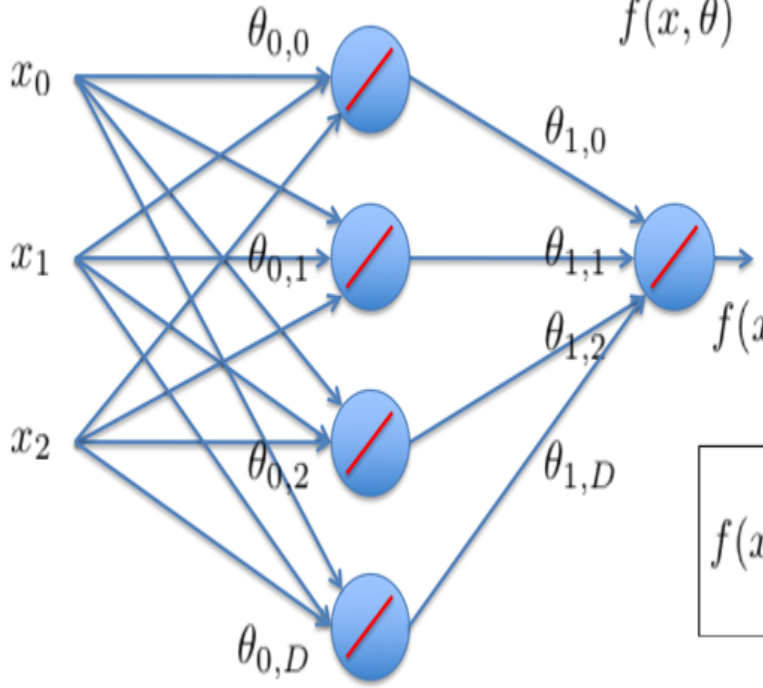
This is a nonlinear function
Of a linear combination
Of nonlinear functions
Of linear combinations
of inputs

A 1-HIDDEN LAYER NET



Linear Regression Neural Networks

- Nothing special happens.
  - The product of two linear transformations is itself a linear transformation.



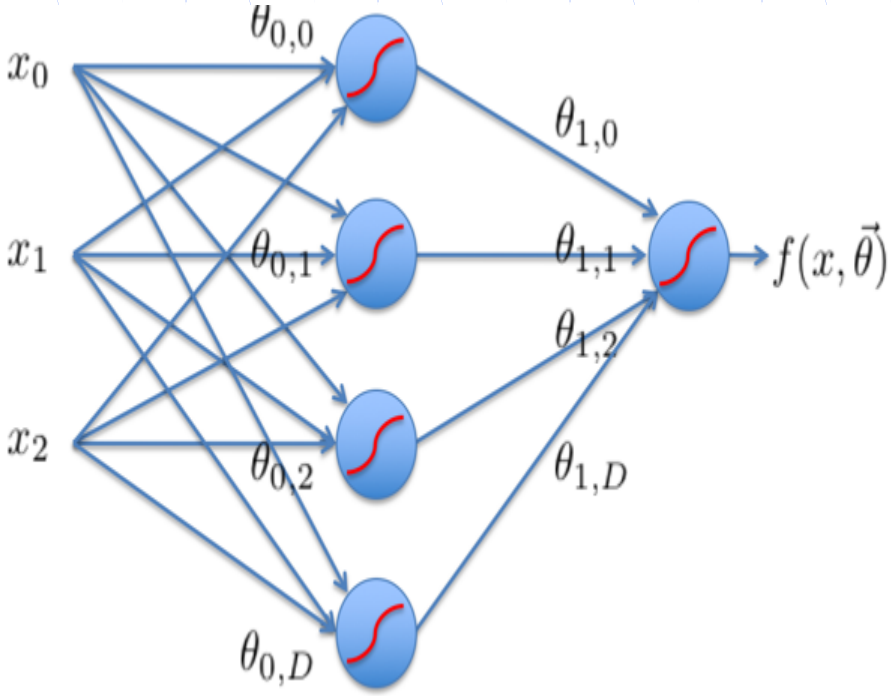
$$f(x,\vec{\theta}) = \sum_{i=0}^{D} \theta_{1,i} \sum_{n=0}^{N-1} \theta_{0,i,n} x_n$$

$$f(x,\vec{\theta}) = \sum_{i=0}^{D} \theta_{1,i} [\theta_{0,i}^T \vec{x}]$$

$$\boxed{f(x,\vec{\theta}) = \sum_{i=0}^{D} [\hat{\theta}_i^T \vec{x}]}$$

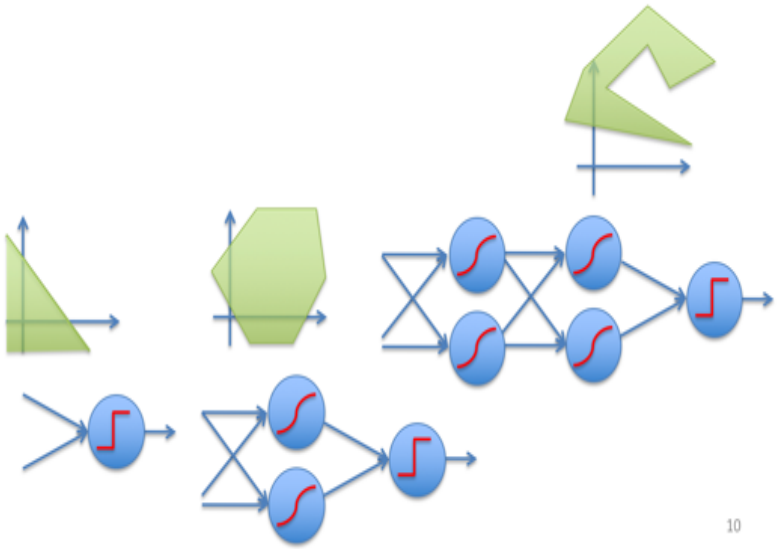
Neural Networks

- We want to introduce non-linearities to the network.
  - Non-linearities allow a network to identify complex regions in space



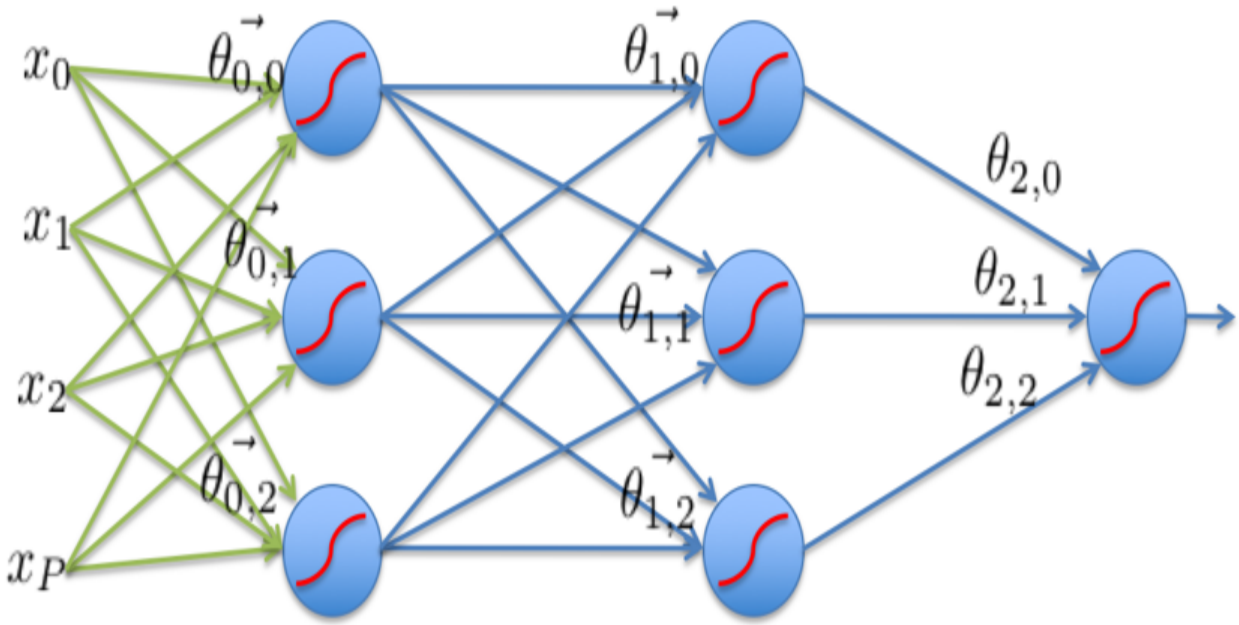
Linear Separability

- More layers can handle more complicated spaces – but require more parameters
- Each node splits the feature space with a hyperplane



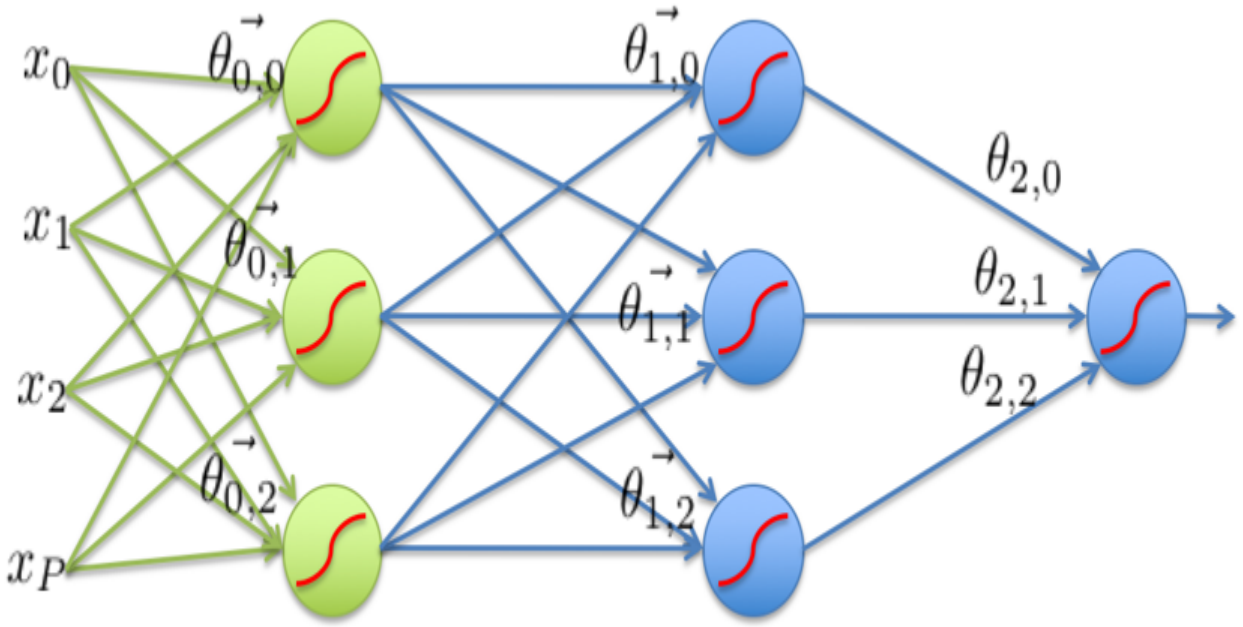
Feed-Forward Networks

- Predictions are fed forward through the network to classify



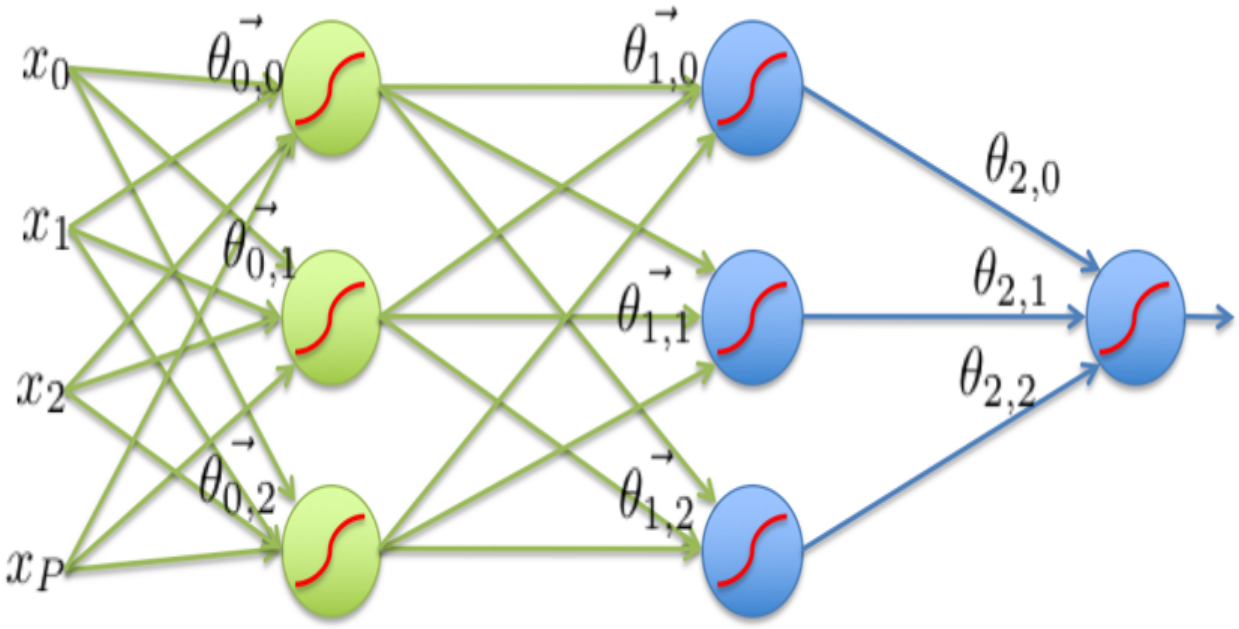
Feed-Forward Networks

- Predictions are fed forward through the network to classify



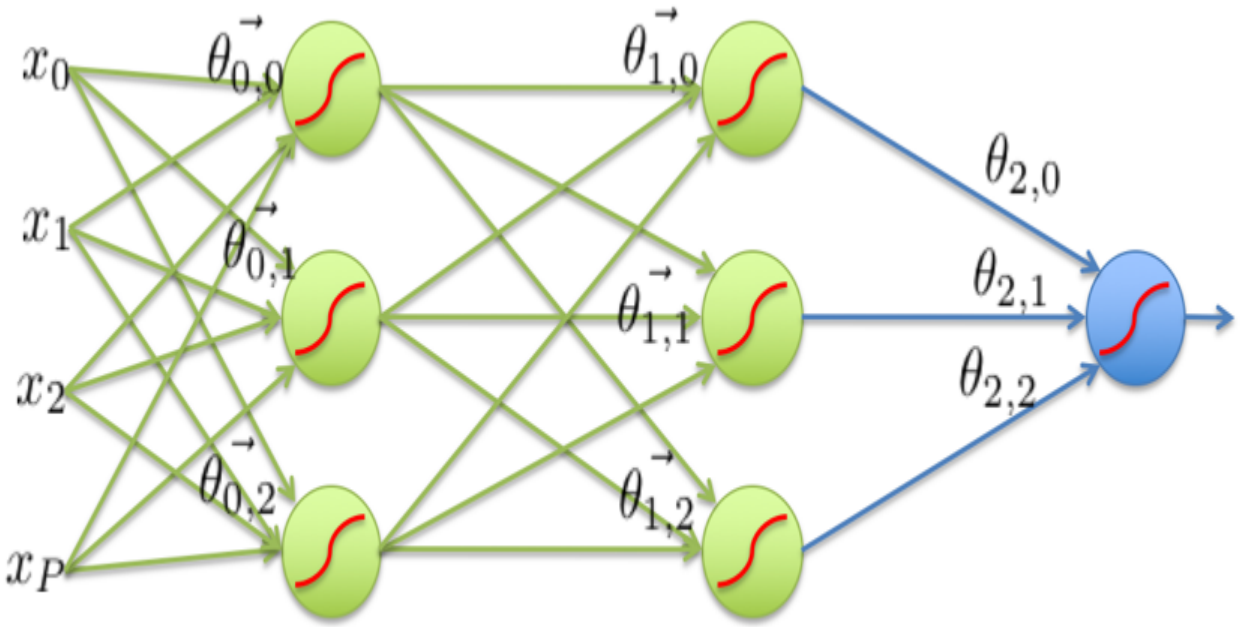
Feed-Forward Networks

- Predictions are fed forward through the network to classify



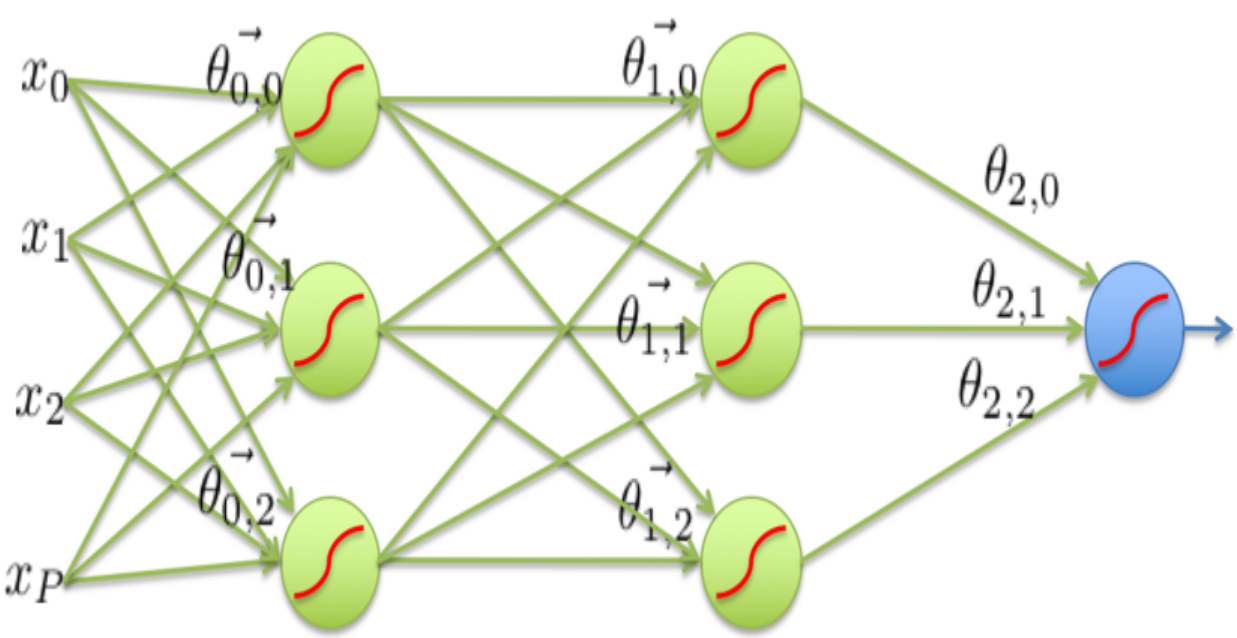
Feed-Forward Networks

- Predictions are fed forward through the network to classify



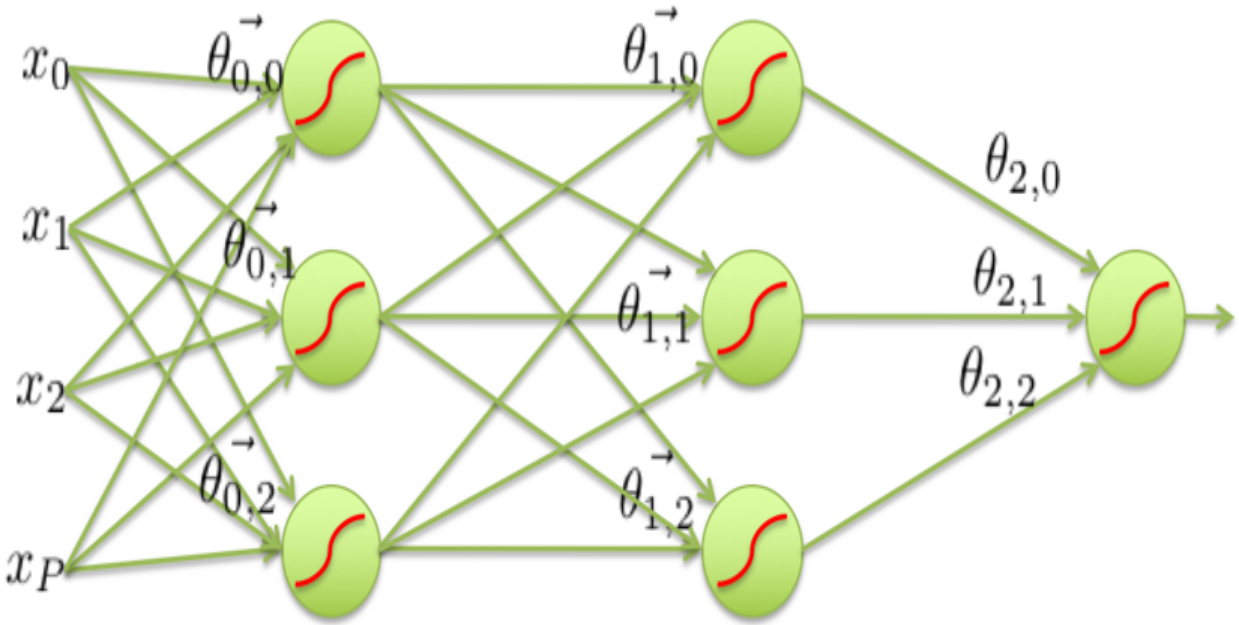
Feed-Forward Networks

- Predictions are fed forward through the network to classify

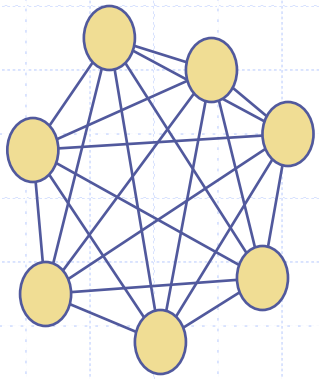


Feed-Forward Networks

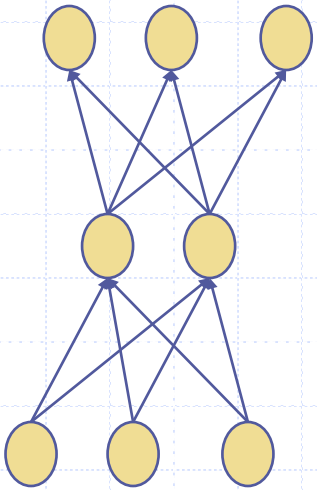
- Predictions are fed forward through the network to classify



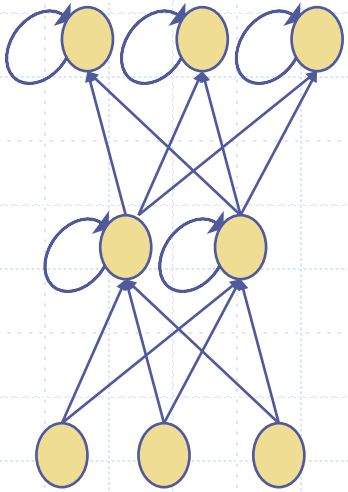
Topologies of Neural Networks

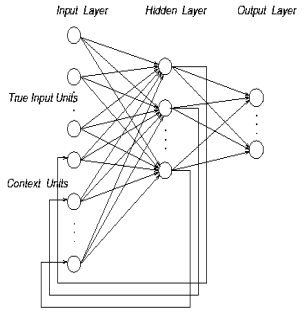


completely connected



feedforward (directed, acyclic)

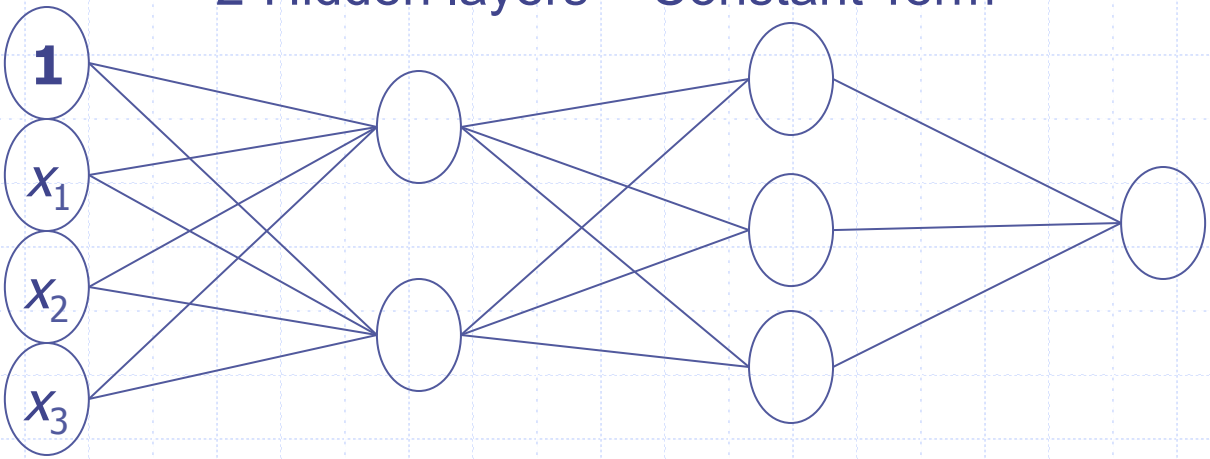




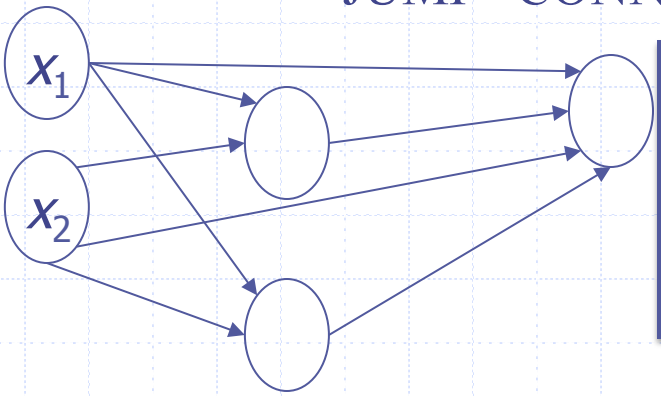
recurrent (feedback connections)

Other Neural Networks

2-Hidden layers + Constant Term



"JUMP" CONNECTIONS



$$\text{Out} = g\left(\sum_{k=1}^{N_{INS}} w_{0k}x_k + \sum_{k=1}^{N_{HID}} W_k v_k\right)$$

Backpropagation

$$\text{Out}(x) = g\left(\sum_j W_j g\left(\sum_k w_{jk} x_k\right)\right)$$

Find a set of weights $\{W_j\}, \{w_{jk}\}$

to minimize

$$\sum_{d \in D} (O_d - t_d)^2$$

by gradient descent.

That's it!

That's the backpropagation algorithm.

Error Gradient for a Sigmoid Unit

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i}\frac{1}{2}\sum_{d\in D}(y_d - t_d)^2$$

$$= \frac{1}{2}\sum_d \frac{\partial}{\partial w_i}(y_d - t_d)^2$$

$$= \frac{1}{2}\sum_d 2(y_d - t_d)\frac{\partial}{\partial w_i}(y_d - t_d)$$

$$= \sum_d (y_d - t_d)\left(-\frac{\partial t_d}{\partial w_i}\right)$$

$$= -\sum_d (y_d - t_d)\frac{\partial t_d}{\partial net_d}\frac{\partial net_d}{\partial w_i}$$

But we know:

$$\frac{\partial y_d}{\partial net_d} = \frac{\partial \sigma\ (net_d)}{\partial net_d} = y_d(1 - y_d)$$

$$\frac{\partial net_d}{\partial w_i} = \frac{\partial(\vec{w}\cdot\vec{x}_d)}{\partial w_i} = x_{i,d}$$

So:

$$\frac{\partial E}{\partial w_i} = -\sum_{d\in D}(y_d - t_d)y_d(1 - y_d)x_{i,d}$$

$\sigma(x)$ is the sigmoid function, $\frac{1}{1+e^{-x}}$

Recall: $\frac{d\sigma(x)}{dx} = \sigma(x)(1-\sigma(x))$

Backpropagation

- We want to evaluate:

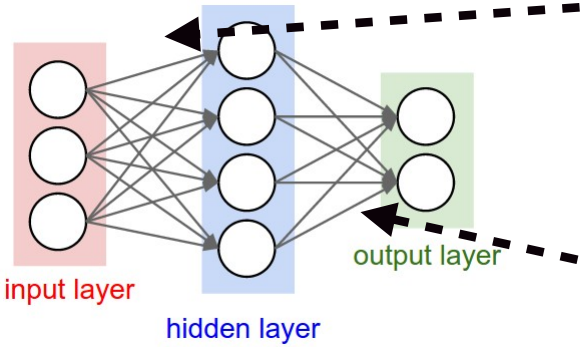
$$\nabla L = [\frac{\partial L}{\partial w_{1,1,1}}, \ldots, \frac{\partial L}{\partial w_{l,k,i}}, \ldots]^T$$

- Utilize the chain rule (calculus):

$$\frac{\partial a}{\partial w} = \frac{\partial a}{\partial b} \cdot \frac{\partial b}{\partial c} \cdot \frac{\partial c}{\partial w}$$

- Example:

$$L = \frac{1}{2} \sum_{i=1}^{2} (y_i - t_i)^2$$

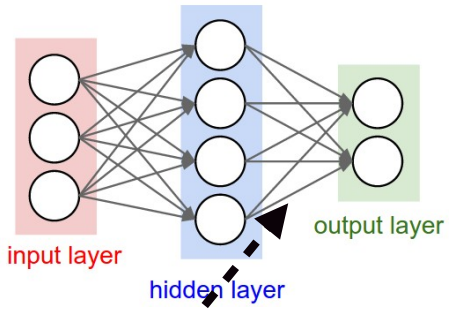


$$\frac{\partial L}{\partial w_{1,1,1}} = ?$$

$$\frac{\partial L}{\partial w_{2,2,4}} = ?$$

Backpropagation

- The latter case:



$$\frac{\partial L}{\partial w_{2,2,4}} = \frac{\partial L}{\partial y_2} \cdot \frac{\partial y_2}{\partial w_{2,2,3}}$$

$$= \frac{\partial \frac{1}{2} \sum_{i=1}^{2} (y_i - t_i)^2}{\partial y_2} \cdot \frac{\partial y_2}{\partial w_{2,2,3}}$$

$$\frac{\partial L}{\partial w_{2,2,4}} = ?$$

$$\frac{\partial L}{\partial y_2} = (y_2 - t_2) \cdot z_4$$

$$= \delta_{2,2} \cdot z_4$$

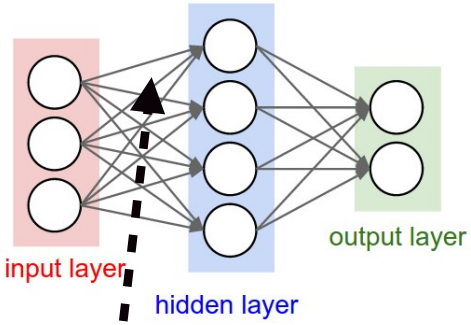
$$y_2 = \sum_{j=1}^{4} w_{2,2,j} \cdot z_j$$

outputs of the hidden layer

*assuming act. fun. for the output layer is identity

Backpropagation

- The former case:



$$\frac{\partial L}{\partial w_{1,1,1}} = ?$$

$$z_1 = f(a_1)$$

$$a_1 = \sum_{j=1}^{3} w_{1,1,j} \cdot x_j$$

$$\frac{\partial L}{\partial w_{1,1,1}} = \frac{\partial L}{\partial a_1} \cdot \frac{\partial a_1}{\partial w_{1,1,1}}$$

$$= \sum_{i=1}^{2} \frac{\partial L}{\partial y_i} \cdot \frac{\partial y_i}{\partial a_1} \cdot x_1 = \delta_{1,1} \cdot x_1$$

$$\delta_{1,1} = \frac{\partial L}{\partial a_1} = \sum_{i=1}^{2} \frac{\partial L}{\partial y_i} \cdot \frac{\partial y_i}{\partial z_1} \cdot \frac{\partial z_1}{\partial a_1}$$

$$= f'(a_1) \cdot \sum_{i=1}^{2} w_{2,i,1} \cdot \delta_{2,i}$$

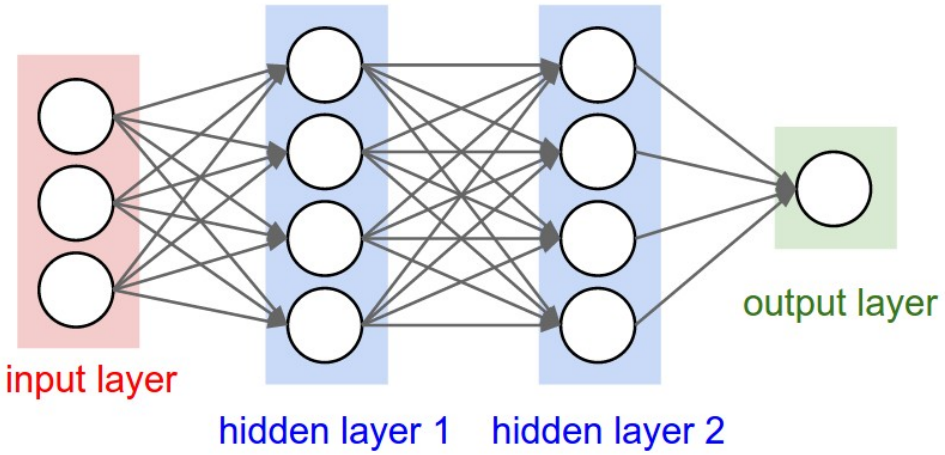
calculated!

Training with BP

Iterate until convergence

Compute activations (a, z, y)

Forward prop



For a batch of M data points X:

Gradient descent

$$w_{l,k,i} = w_{l,k,i} - \eta \cdot \frac{1}{M} \sum_{m=1}^{M} \frac{\partial L_m}{\partial w_{l,k,i}}$$

Back-prop

Compute derivatives

Generalizes to other architectures!

Backpropagation Algorithm

Initialize all weights to small random numbers. Until satisfied, do

- For each training example, do
  1. Input the training example and compute the outputs
  2. For each output unit k

$$\delta_k \leftarrow o_k(1-o_k)(y_k-o_k)$$

  3. For each hidden unit h

$$\delta_h \leftarrow o_h(1-o_h)\sum_{k\in outputs} w_{k,h}\delta_k$$

  4. Update each network weight $w_{i,j}$

$$w_{j,i} \leftarrow w_{j,i} + \Delta w_{j,i}$$

where

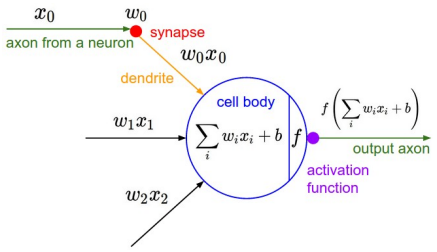
$$\Delta w_{j,i} = \eta\ \delta_j x_{j,i}$$

Examples

- ReLU neuron k in layer l:

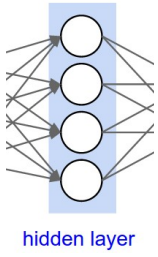
Modeling bias with an extra unit

$$f_{l,k} = \max(0, \mathbf{w}_{l,k} \cdot \mathbf{x})$$



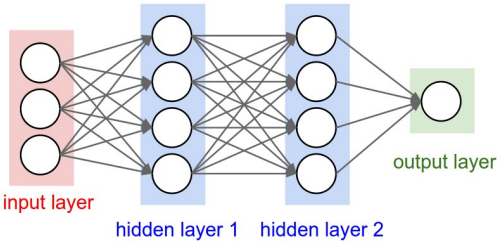
- Layer with ReLU neurons:

$$\mathbf{f}_l = \max(\mathbf{0}, \mathbf{W}_l \cdot \mathbf{x})$$



- MLP: 2 hidden (ReLU) + softmax output

$$y = \sigma(\mathbf{w}_3 \cdot \max(\mathbf{0}, \mathbf{W}_2 \cdot \max(\mathbf{0}, \mathbf{W}_1 \cdot \mathbf{x})))$$



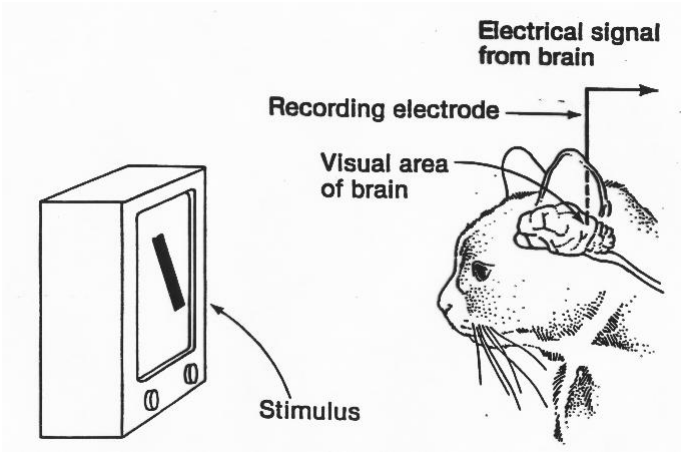
In Keras

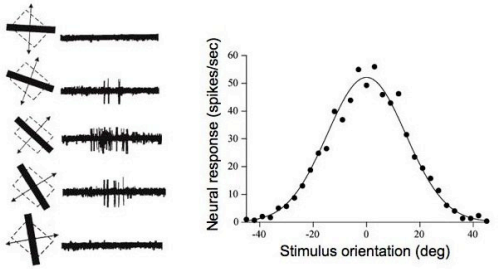
```
#imports and data loading
#. . .
#. . .

#constructing a model
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(784,)))
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))
model.summary()
model.compile(loss='categorical_crossentropy',  optimizer=RMSprop(), metrics=['accuracy'])

#training
history = model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs, \
  verbose=1, validation_data=(x_test, y_test))

#evaluating
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

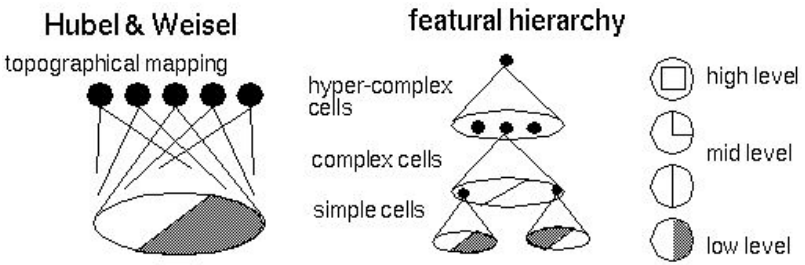
Learning time: 20 epochs Accuracy: 98.40%

Convolutional Neural Networks

- Sensibly ‘adjust’ network architecture:
  - what would be a good network architecture for visual imagery?
- Hubel & Wiesel (1959):
  - importance of hierarchical organization

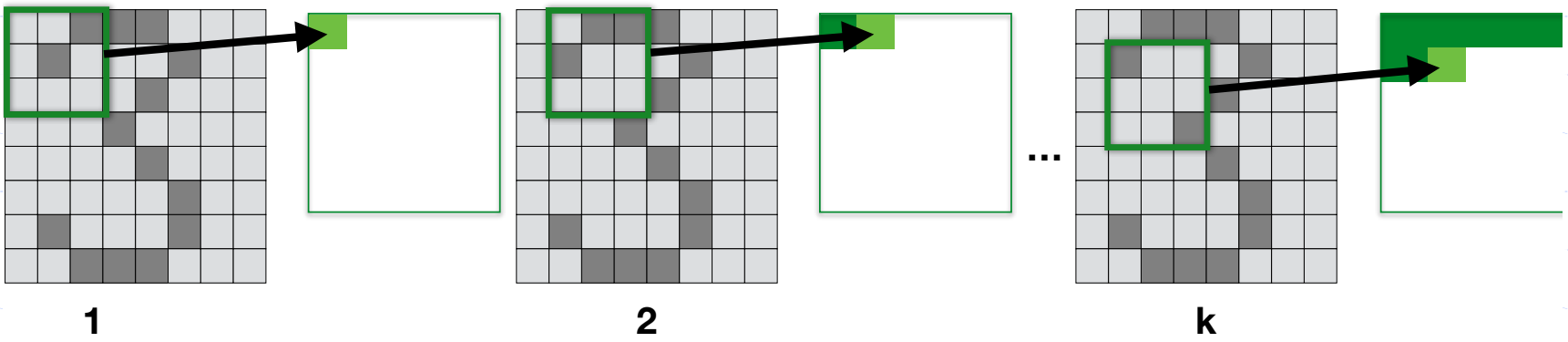






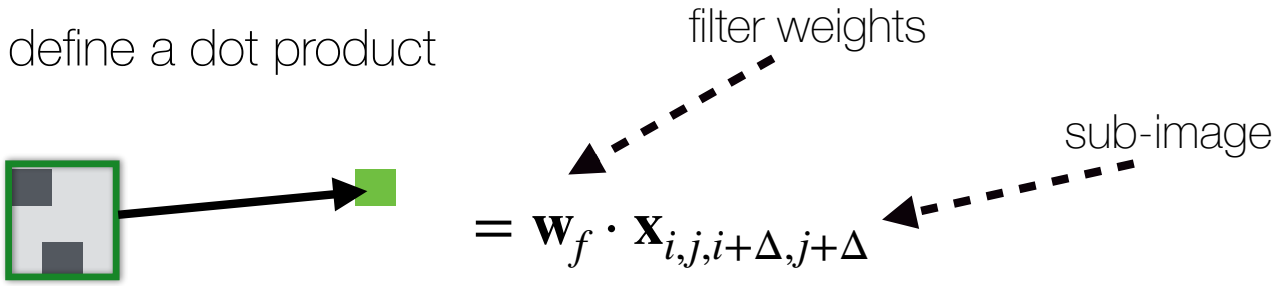
Convolutional Neural Networks

- Parameter sharing:
  - Utilize convolution in order to reduce the number of parameters
  - Basic idea:
    - Init. layer: input (e.g. image)
    - Define filter -> typically smaller size
    - Convolve filter with the layer

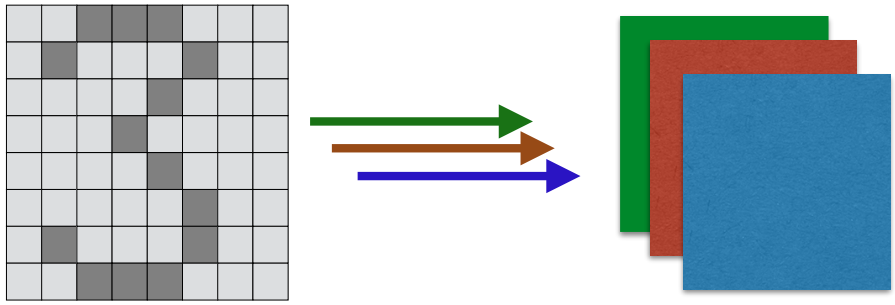


Convolutional layer

- Filters define a dot product



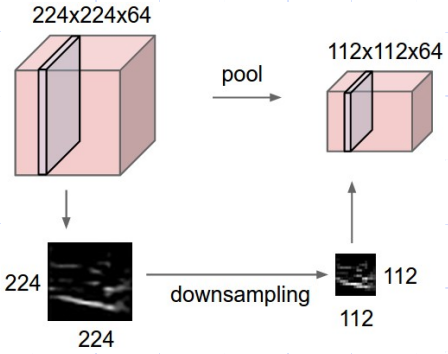
- After convolution, apply element-wise non-linear activation
  - Typically ReLU
- Multiple-filters:

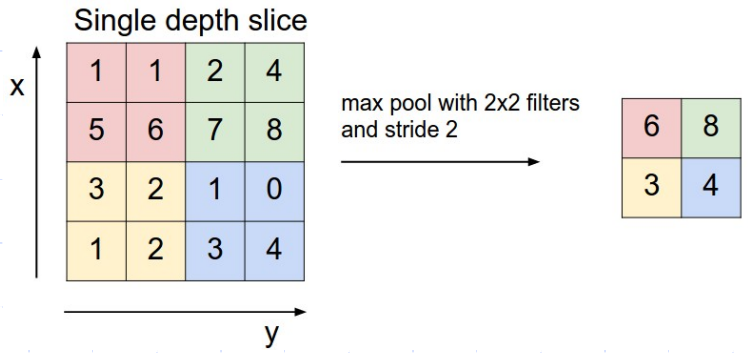


- Important: filters' depths should be the same as the input volume.

Stride, padding, pooling

- Stride: controls the shift of the filter
  - e.g. stride = 2 means that we jump 2 pixels at a time
- Padding: adding zeros around the border
  - convenient for preserving the input size
- Pooling layer: progressively reducing the spacial size





Putting together

- Convolutional neural networks (CNN):
  - 'Head' - convolutional and pooling layers:
    - feature learning
  - 'Tail' - application dependent:
    - e.g. for classification: fully connected layers

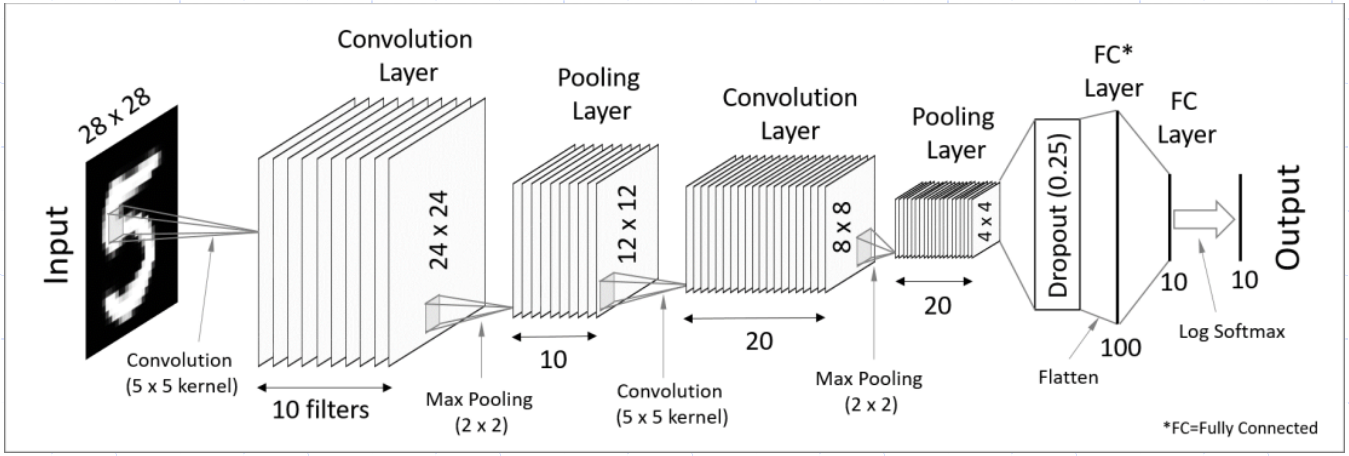


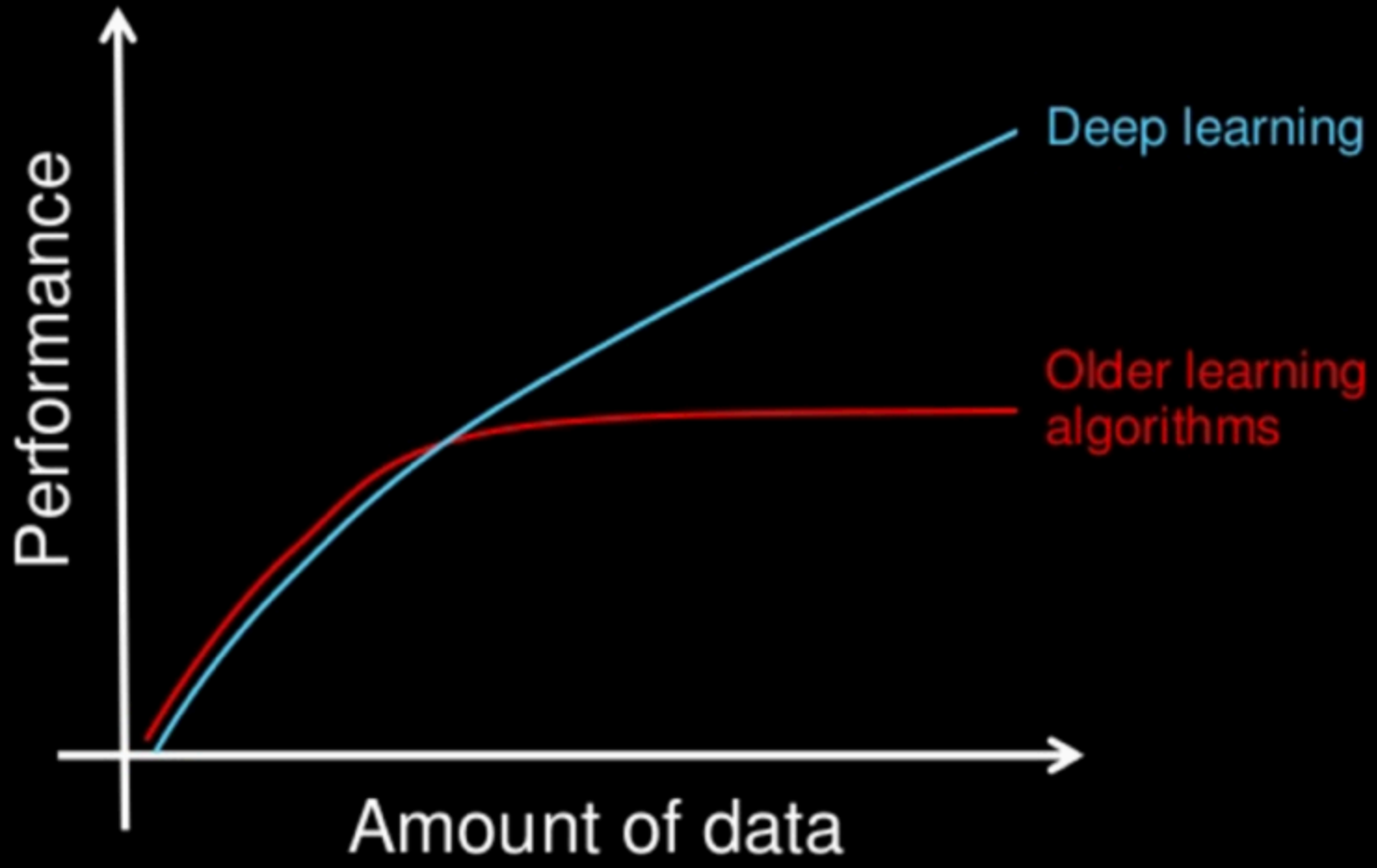
*Image from codetolight.wordpress.com

CNN in Keras

```
#imports and data loading
#. . .
#. . .

#constructing a model
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu',input_shape=input_shape))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

#training
history = model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs, \
  verbose=1, validation_data=(x_test, y_test))

#evaluating
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

Learning time: 12 epochs Accuracy: 99.25%

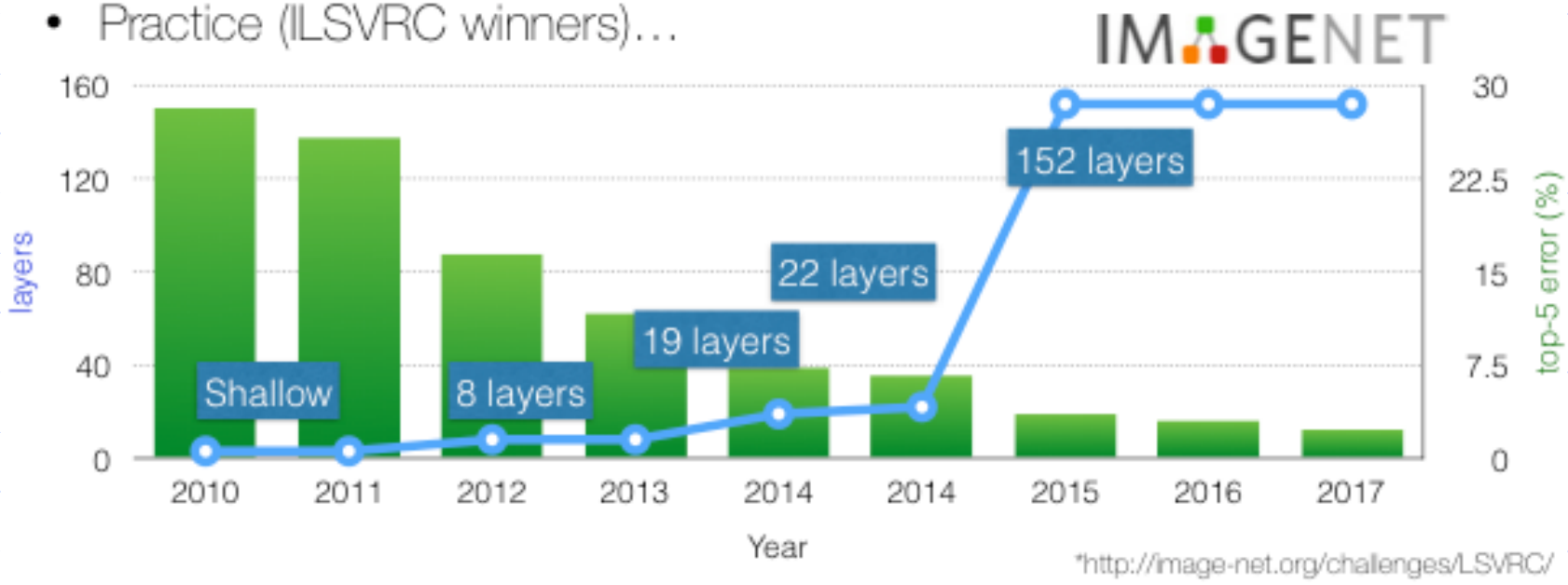
Why deep learning



How do data science techniques scale with amount of data?

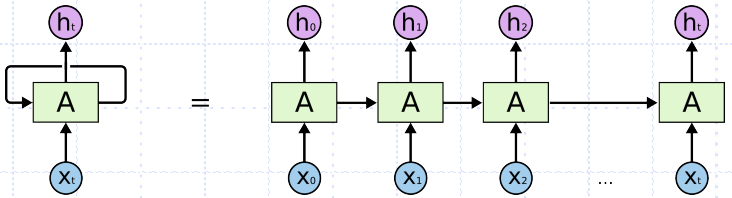
Is being deep important

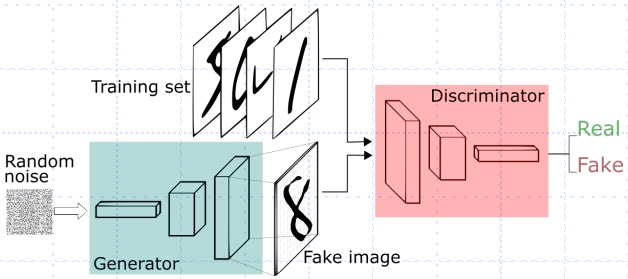
- *Universality theorem*, Cybenko 1989
  - Holds for 'shallow' networks (1 hidden layer)
  - Caveat, no conditioning on the number of the hidden neurons
- Practice (ILSVRC winners)…



More on DL

- Many other models… Examples:
  - Sequence processing:
    - Recurrent neural networks, LSTM
    - Residual Net
  - Generative models:
    - Variational autoencoder
    - Generative adversarial networks





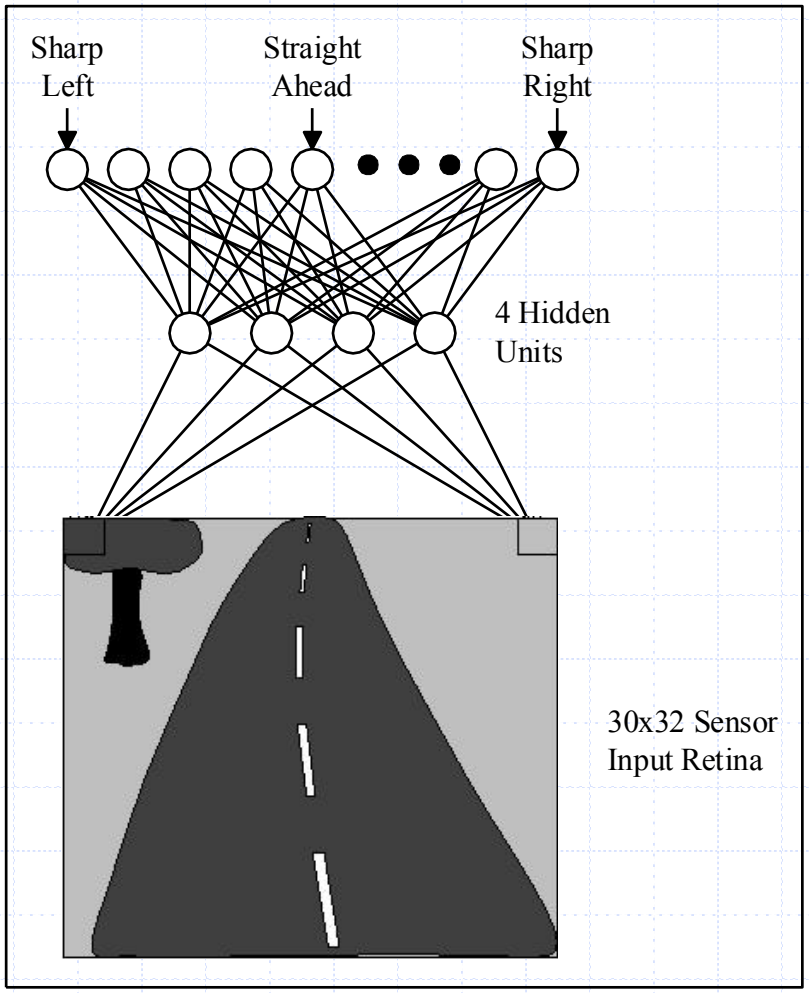
- Some drawbacks:
  - Data and compute time
  - Interpretability
  - Theory?

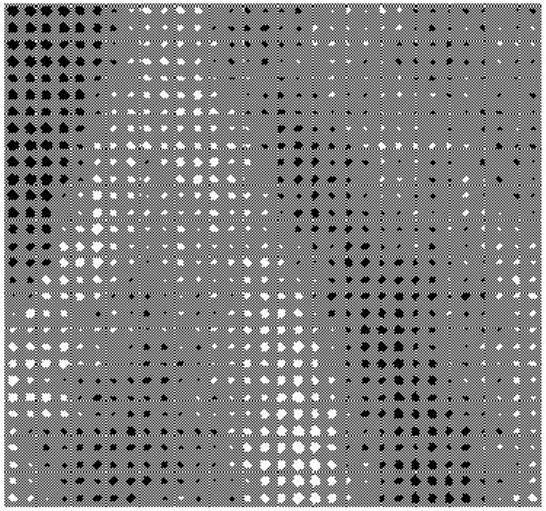
*GAN image from skymind.ai

“Executing” neural networks

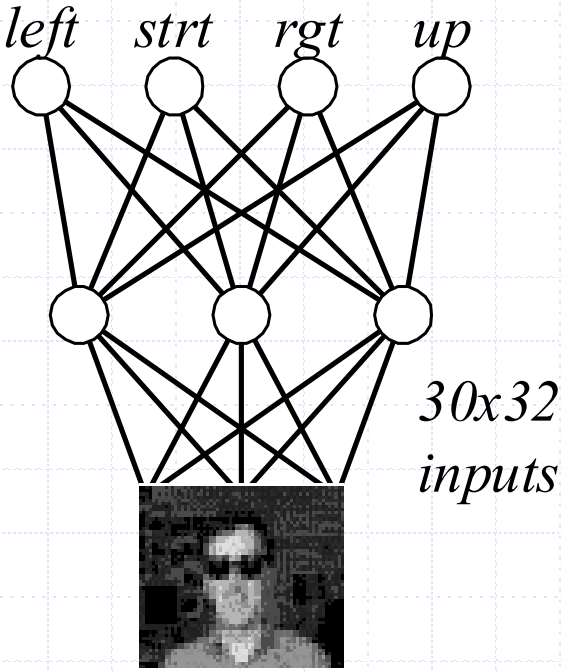
- Input units are set by some exterior function (think of these as sensors), which causes their output links to be activated at the specified level
- Working forward through the network, the input function of each unit is applied to compute the input value
  - Usually this is just the weighted sum of the activation on the links feeding into this node
- The activation function transforms this input function into a final value
  - Typically this is a nonlinear function, often a sigmoid function corresponding to the “threshold” of that node

ALVINN drives 70 mph on highways

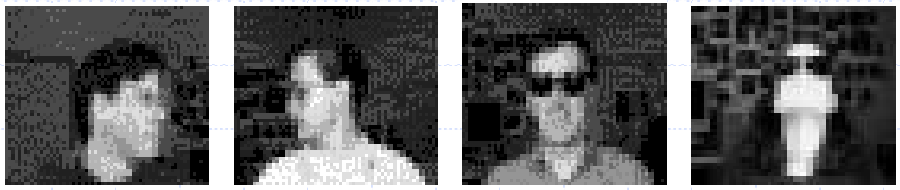




Neural Nets for Face Recognition



90% accurate learning head pose, and recognizing 1-of-20 faces



Typical Input Images

NNs for Handwritten Character Recognition

- LeNet, Yann LeCun et al.
- Convolutional Neural Networks
  - Uses backprop
  - Representation optimized for pixel processing; handling extreme variability, and robust to distortions and simple transformations
- http://yann.lecun.com/exdb/lenet/

- Lenet Video

Deep Learning

- NN with at least 2 hidden layers
- Different hidden layers often represent different abstraction levels
- Networks with loops -- recurrent neural networks
- Major successes in recent years:
  - Computer vision
  - NLP
  - Go

Summary: Perceptrons & NN

- Perceptrons
  - Simple classifier
  - Power depends on features
  - Basis of many more sophisticated algorithms
- NN
  - Networks of simple classifiers
  - Capable of solving complex real-world problems