

# Neural Nets

Schwartz

January 7, 2017

# Making machines that think

The perceptron – a single layer feedforward neural network – was invented in 1957 at the Cornell Aeronautical Laboratory by Frank Rosenblatt with funding from the United States Office of Naval Research. In a 1958 press conference organized by the US Navy, based on Rosenblatt's statements, The New York Times reported the perceptron to be “the embryo of an electronic computer that [the Navy] expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence.”

Upon his death, the titular Reverend Thomas Bayes (1701–1761) of the renowned theorem left an unpublished manuscript deriving the distribution for the parameter of a binomial distribution. This found its way to Richard Price, who prior to posthumously reading the manuscript at the Royal Society, dutifully edited the manuscript and added an introduction that forms much of the philosophical basis for Bayesian analysis. Mathematical historians have argued that “by modern standards, we should refer to the Bayes-Price rule. Price discovered Bayes' work, recognized its importance, corrected it, contributed to the article, and found a use for it. The modern convention of employing Bayes' name alone is unfair but so entrenched that anything else makes little sense.” Quite unaware of Bayes' work, the French mathematician Pierre-Simon Laplace reproduced and extended Bayes' results in 1774. It was also suggested that the blind English mathematician Nicholas Saunderson discovered the theorem some time before Bayes, but this is disputed.

The elegant simplicity and effectiveness of Bayes' theorem for “learning” has prompted some psychologists to ask if the human brain itself might be a Bayesian-reasoning machine. They suggest that the Bayesian capacity to draw strong inferences from sparse data could be crucial to the way the mind perceives the world, plans actions, comprehends and learns language, reasons from correlation to causation, and even understands the goals and beliefs of other minds. The key to successful Bayesian reasoning is not in having an extensive, unbiased sample, which is the eternal worry of frequentists, but rather in having an appropriate “prior”. This prior is an assumption about the way the world works. With the correct prior, even a single piece of data can be used to make meaningful Bayesian predictions. By contrast, frequentism is perhaps not well suited to making decisions on the basis of limited information – which is something that people have to do all the time.

It is thought by cognitive neuroscientists that neocortical development occurs in sequential layers, driven by waves of nerve growth factors which result in a self-organizing system. Modern so-called *deep learning* successors of the perceptron use Bayesian model fitting techniques to analogously sequentially train layer upon layer of neurons to produce unsupervised (self-organizing) classification networks.

# Objectives

- ▶ neural networks
- ▶ activating functions
- ▶ neural networks  $\implies$  regression
- ▶ abstract feature encoding via layers
- ▶ backpropagation (gradient decent via chain rule)
  
- ▶ backpropagation parameter tuning
- ▶ images as data
- ▶ convolutions
- ▶ convolutional networks

# Artificial Neural Network (NN)

- ▶ NN's are a collection of nodes  $\eta_j^{(l)}$  that take on various states  $\{\eta_j^{(l)}\}$

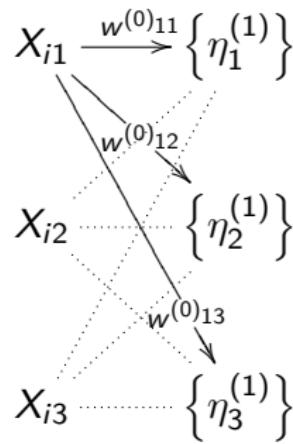
$$\{\eta_1^{(1)}\}$$

$$\{\eta_2^{(1)}\}$$

$$\{\eta_3^{(1)}\}$$

# Artificial Neural Network (NN)

- ▶ NN's are a collection of nodes  $\eta_j^{(l)}$  that take on various states  $\{\eta_j^{(l)}\}$
- ▶ Features are embedded into NNs as collections of states



# Artificial Neural Network (NN)

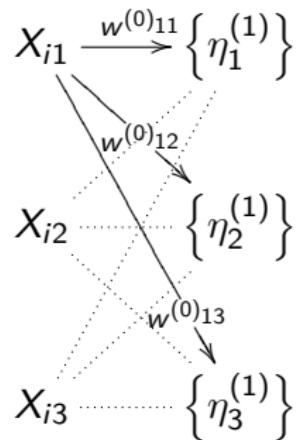
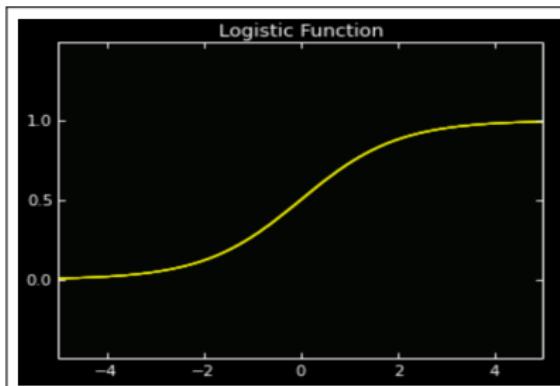
- ▶ NN's are a collection of nodes  $\eta_j^{(l)}$  that take on various states  $\{\eta_j^{(l)}\}$
- ▶ Features are embedded into NNs as collections of states

E.g.,  $\{\eta_1^{(1)}\} = f(X^T W_1^{(0)})$  where

$$X = [X_1, X_2, X_3]^T$$

$$W_j^{(0)} = [w_{1j}^{(0)}, w_{2j}^{(0)}, w_{3j}^{(0)}]^T$$

with activation function  $f$



# Artificial Neural Network (NN)

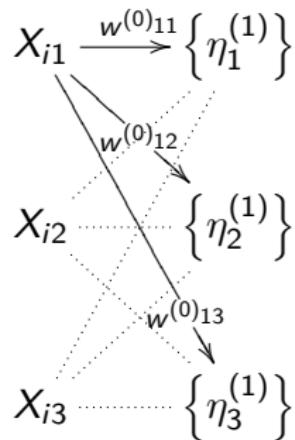
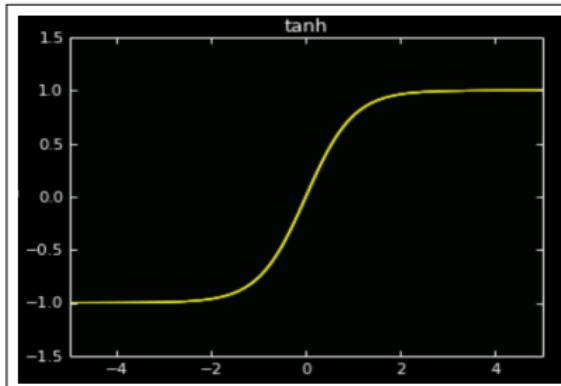
- ▶ NN's are a collection of nodes  $\eta_j^{(l)}$  that take on various states  $\{\eta_j^{(l)}\}$
- ▶ Features are embedded into NNs as collections of states

E.g.,  $\{\eta_1^{(1)}\} = f(X^T W_1^{(0)})$  where

$$X = [X_1, X_2, X_3]^T$$

$$W_j^{(0)} = [w_{1j}^{(0)}, w_{2j}^{(0)}, w_{3j}^{(0)}]^T$$

with activation function  $f$



# Artificial Neural Network (NN)

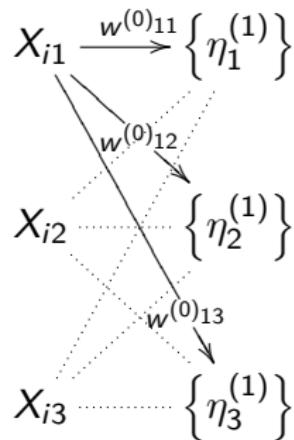
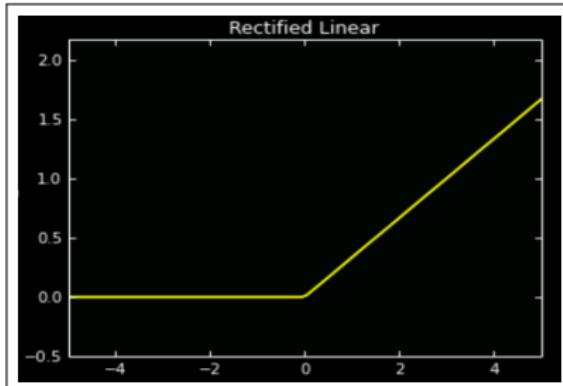
- ▶ NN's are a collection of nodes  $\eta_j^{(l)}$  that take on various states  $\{\eta_j^{(l)}\}$
- ▶ Features are embedded into NNs as collections of states

E.g.,  $\{\eta_1^{(1)}\} = f(X^T W_1^{(0)})$  where

$$X = [X_1, X_2, X_3]^T$$

$$W_j^{(0)} = [w_{1j}^{(0)}, w_{2j}^{(0)}, w_{3j}^{(0)}]^T$$

with activation function  $f$



# Artificial Neural Network (NN)

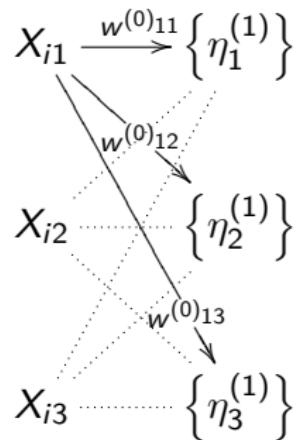
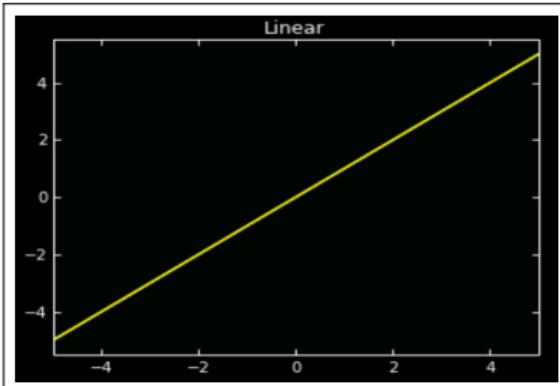
- ▶ NN's are a collection of nodes  $\eta_j^{(l)}$  that take on various states  $\{\eta_j^{(l)}\}$
- ▶ Features are embedded into NNs as collections of states

E.g.,  $\{\eta_1^{(1)}\} = f(X^T W_1^{(0)})$  where

$$X = [X_1, X_2, X_3]^T$$

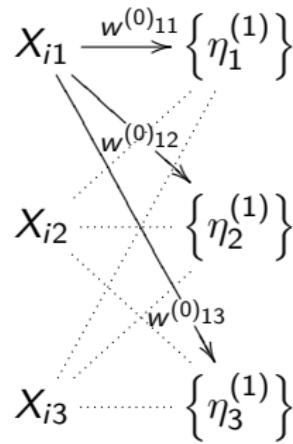
$$W_j^{(0)} = [w_{1j}^{(0)}, w_{2j}^{(0)}, w_{3j}^{(0)}]^T$$

with activation function  $f$



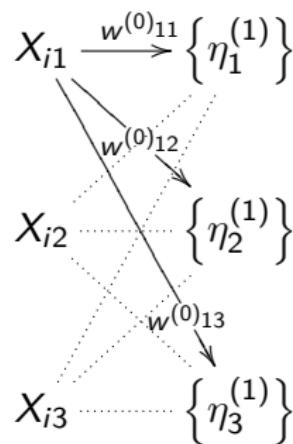
# Artificial Neural Network (NN)

- ▶ NN's are a collection of nodes  $\eta_j^{(l)}$  that take on various states  $\{\eta_j^{(l)}\}$
- ▶ Features are embedded into NNs as collections of states  
*Action potentials fire when a neuron (node) is turned "on"*



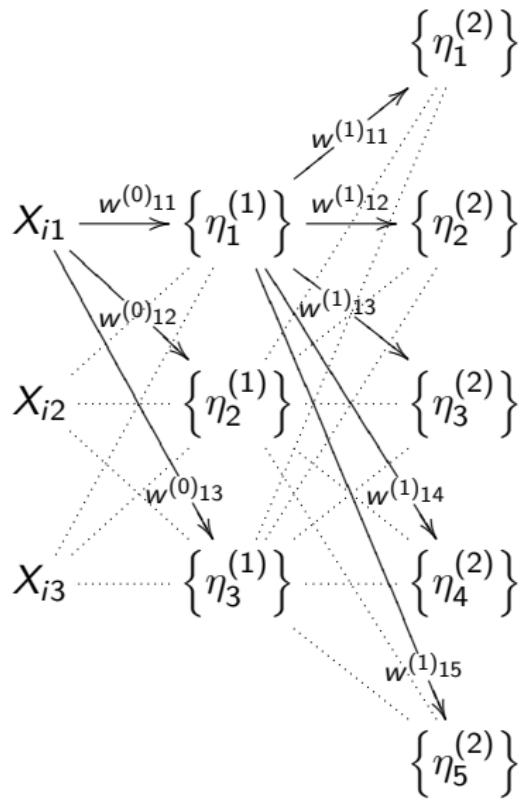
# Artificial Neural Network (NN)

- ▶ NN's are a collection of nodes  $\eta_j^{(l)}$  that take on various states  $\{\eta_j^{(l)}\}$
- ▶ Features are embedded into NNs as collections of states  
*Action potentials fire when a neuron (node) is turned "on" which captures an idea or concept*



# Artificial Neural Network (NN)

- ▶ NN's are a collection of nodes  $\eta_j^{(l)}$  that take on various states  $\{\eta_j^{(l)}\}$
- ▶ Features are embedded into NNs as collections of states  
*Action potentials fire when a neuron (node) is turned "on" which captures an idea or concept*
- ▶ These can be further combined in subsequent layers of states



# Artificial Neural Network (NN)

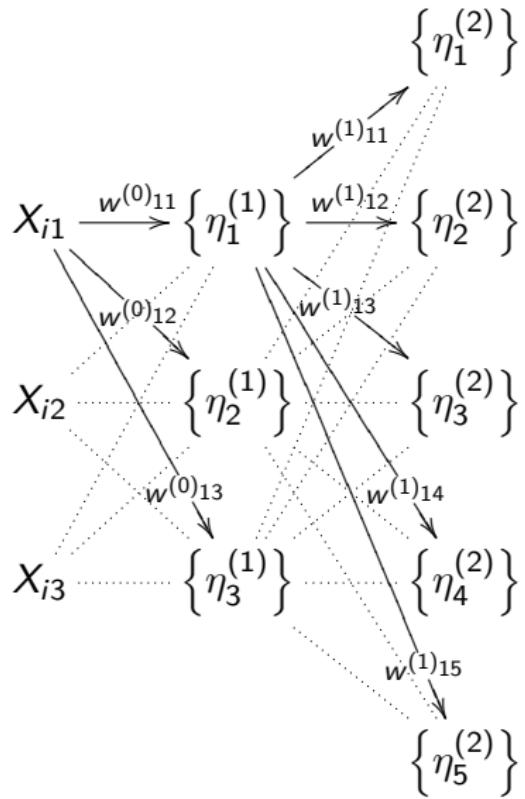
- ▶ NN's are a collection of nodes  $\eta_j^{(l)}$  that take on various states  $\{\eta_j^{(l)}\}$

- ▶ Features are embedded into NNs as collections of states

*Action potentials fire when a neuron (node) is turned "on" which captures an idea or concept*

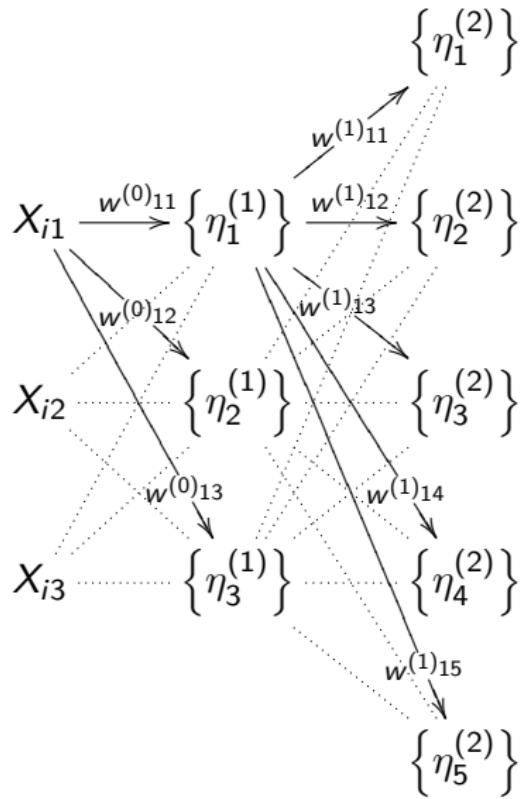
- ▶ These can be further combined in subsequent layers of states

*Each hierarchical layer provides higher orders of abstraction*



# Artificial Neural Network (NN)

- ▶ NN's are a collection of nodes  $\eta_j^{(l)}$  that take on various states  $\{\eta_j^{(l)}\}$
- ▶ Features are embedded into NNs as collections of states  
*Action potentials fire when a neuron (node) is turned "on" which captures an idea or concept*
- ▶ These can be further combined in subsequent layers of states  
*Each hierarchical layer provides higher orders of abstraction*
- ▶ An NN's state at any time is it's current collection of "thoughts"



# Artificial Neural Network (NN)

- ▶ NN's are a collection of nodes  $\eta_j^{(l)}$  that take on various states  $\{\eta_j^{(l)}\}$

- ▶ Features are embedded into NNs as collections of states

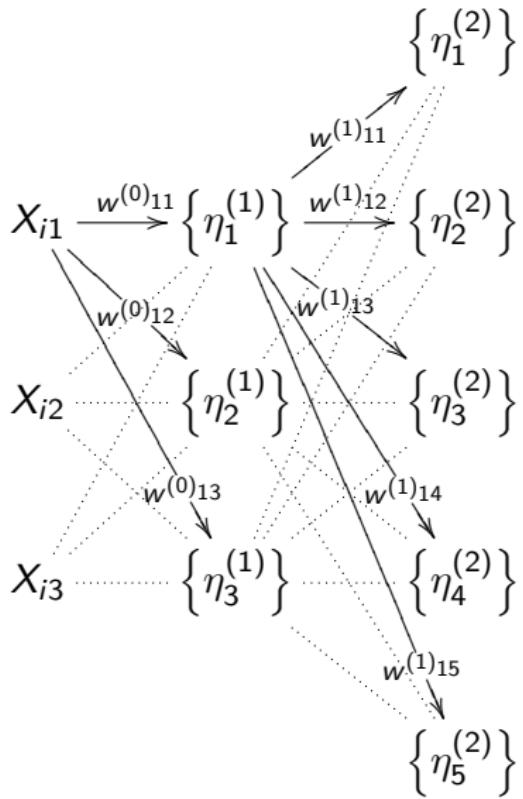
*Action potentials fire when a neuron (node) is turned “on” which captures an idea or concept*

- ▶ These can be further combined in subsequent layers of states

*Each hierarchical layer provides higher orders of abstraction*

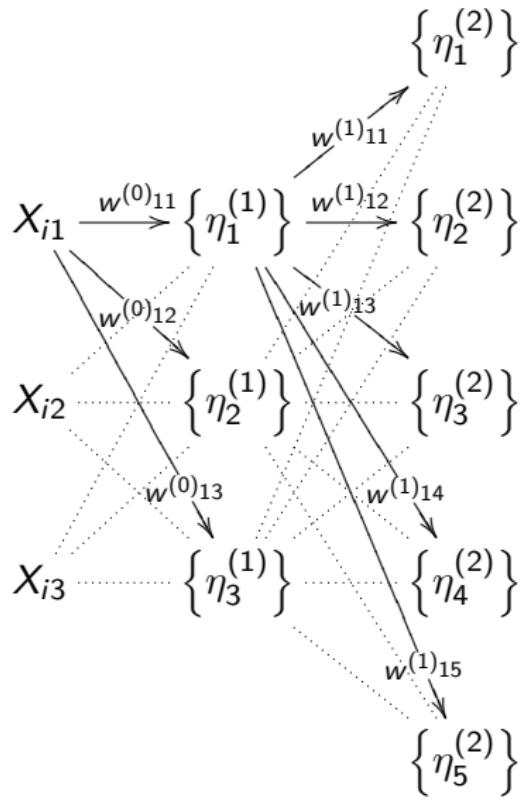
- ▶ An NN's state at any time is it's current collection of “thoughts”

*Similar “thoughts” about similar X's means the NN understands, knows or recognizes what X is*



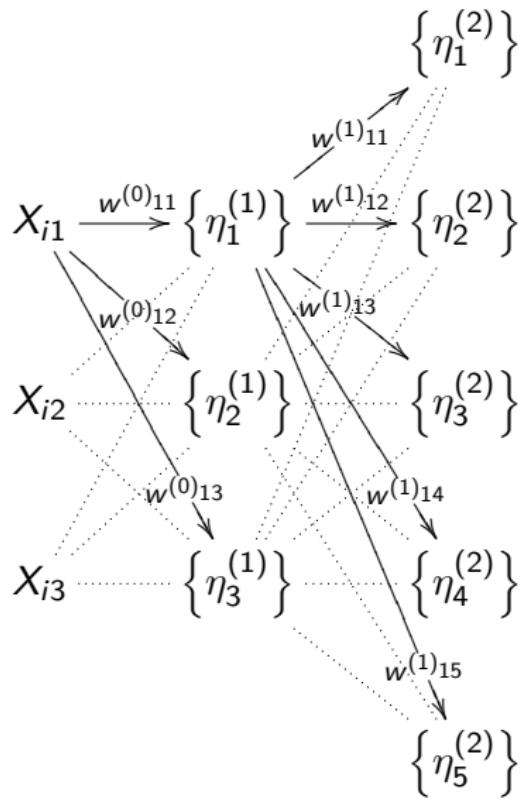
# Artificial Neural Network (NN)

- The (non-linear) activation functions maps, e.g.,  $X^T W_1^{(0)}$   
(and then, e.g.,  $\{\eta^{(1)}\}^T W_1^{(1)}$ )  
onto a set of action potentials



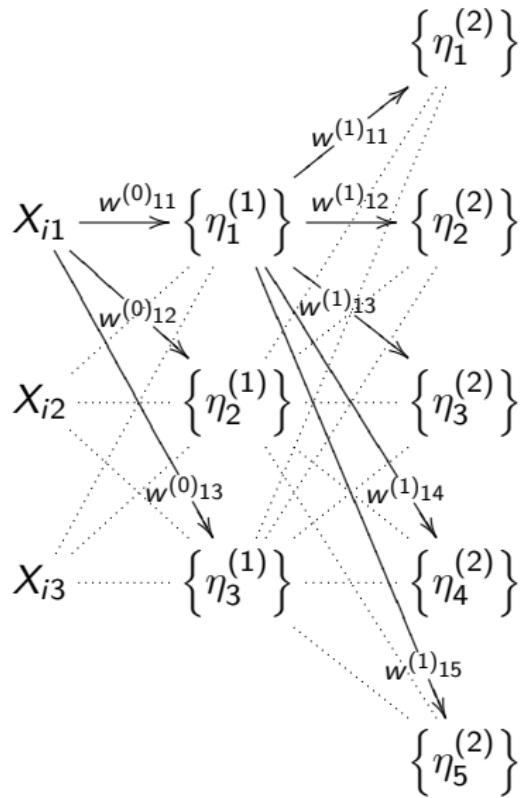
# Artificial Neural Network (NN)

- ▶ The (non-linear) activation functions maps, e.g.,  $X^T W_1^{(0)}$   
(and then, e.g.,  $\{\eta^{(1)}\}^T W_1^{(1)}$ ) onto a set of action potentials
- ▶ The set of (“on” or “off”) states of these activations encode the input sample into the NN as one of its many possible “states”



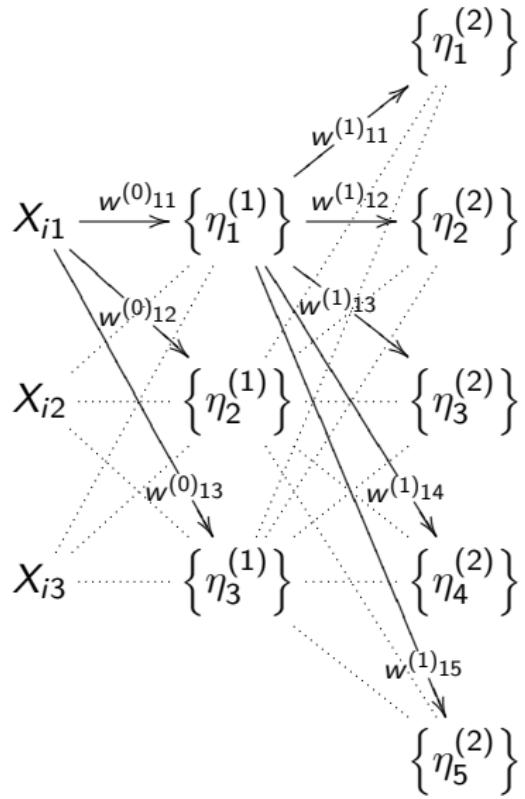
# Artificial Neural Network (NN)

- ▶ The (non-linear) activation functions maps, e.g.,  $X^T W_1^{(0)}$   
(and then, e.g.,  $\{\eta^{(1)}\}^T W_1^{(1)}$ ) onto a set of action potentials
- ▶ The set of (“on” or “off”) states of these activations encode the input sample into the NN as one of its many possible “states”
- ▶ The topology (size and number of layers) of the NN influences the number of “ideas” and level of cumulative abstraction applied



# Artificial Neural Network (NN)

- ▶ The (non-linear) activation functions maps, e.g.,  $X^T W_1^{(0)}$   
(and then, e.g.,  $\{\eta^{(1)}\}^T W_1^{(1)}$ ) onto a set of action potentials
- ▶ The set of (“on” or “off”) states of these activations encode the input sample into the NN as one of its many possible “states”
- ▶ The topology (size and number of layers) of the NN influences the number of “ideas” and level of cumulative abstraction applied
- ▶ The  $w$ 's learn (are trained) to identify patterns in input that map similar inputs to similar NN states

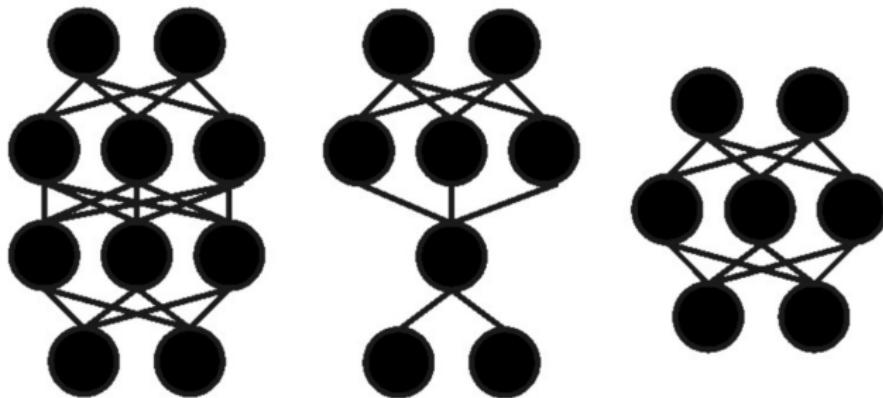


## Using NN's: where the fun ends and the pain begins

- ▶ So patterns are trained (embedded) into an NN...

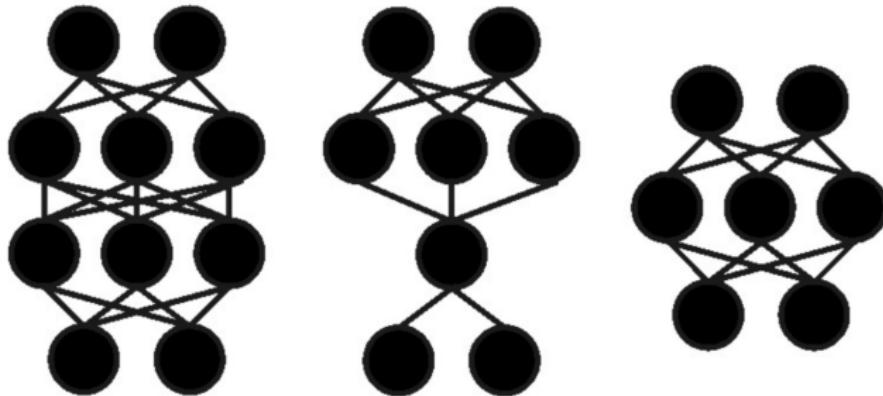
## Using NN's: where the fun ends and the pain begins

- ▶ So patterns are trained (embedded) into an NN...
- ▶ We have to specify the topology



## Using NN's: where the fun ends and the pain begins

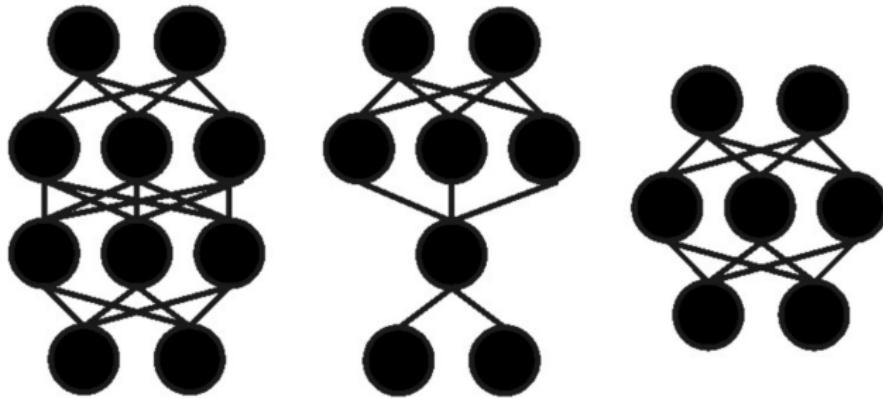
- ▶ So patterns are trained (embedded) into an NN...
- ▶ We have to specify the topology



- ▶ Training patterns into a *Deep NN (DNN)* is challenging...

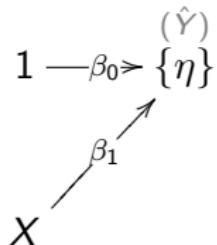
## Using NN's: where the fun ends and the pain begins

- ▶ So patterns are trained (embedded) into an NN...
- ▶ We have to specify the topology

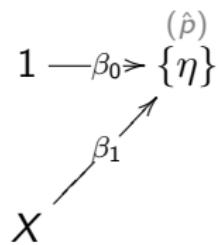


- ▶ Training patterns into a *Deep NN (DNN)* is challenging...  
Let's start simple...

## Example NNs (Regression and Logistic Regression)



$$\begin{aligned}\hat{Y} &= f \left( [1, x] \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix} \right) \\ &= [1, x] \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix} = \beta_0 + \beta_1 x\end{aligned}$$



$$\begin{aligned}\hat{p} &= f \left( [1, x] \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix} \right) \\ &= \frac{1}{1 + e^{-(\beta_0 + \beta_1 x)}}\end{aligned}$$

$$\min_{\hat{Y}} \frac{1}{2} \sum (Y_i - \hat{Y}_i)^2$$

$$\min_p - \prod Y_i \log(p_i) + (1 - Y_i) \log(1 - p_i)$$

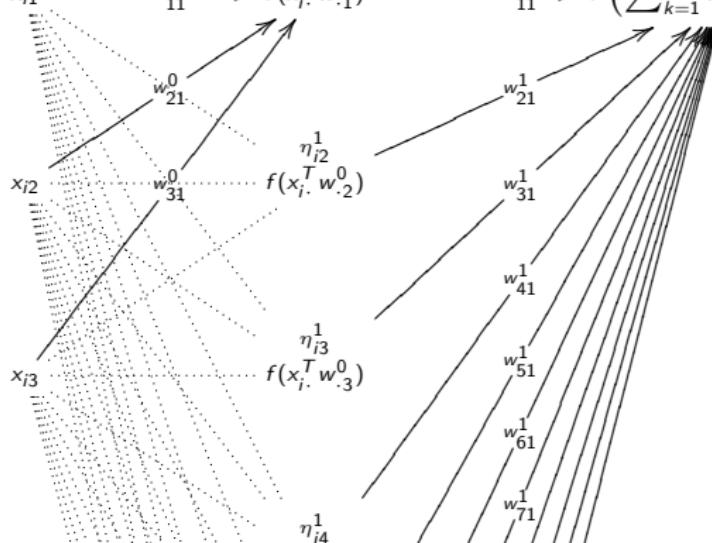
$$\hat{\beta} = \operatorname{argmin}_{\beta} \frac{1}{2} \sum (Y_i - \beta_0 + \beta_1 x_i)^2$$

$$\begin{aligned}\hat{\beta} &= \operatorname{argmin}_{\beta} - \prod Y_i \log \left( \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_i)}} \right) \\ &\quad + (1 - Y_i) \log \left( \frac{1}{1 + e^{\beta_0 + \beta_1 x_i}} \right)\end{aligned}$$

# Forward propagation

$$\hat{Y} = f \left( f \left( \begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ \vdots & \vdots & \vdots \\ x_{n1} & x_{n2} & x_{n3} \end{bmatrix}_{n \times 3} \cdot \begin{bmatrix} w_{11}^0 & w_{12}^0 & \cdots & w_{1p}^0 \\ w_{21}^0 & w_{22}^0 & \cdots & w_{2p}^0 \\ w_{31}^0 & w_{32}^0 & \cdots & w_{3p}^0 \end{bmatrix}_{3 \times p} \right) \cdot \begin{bmatrix} w_{11}^1 \\ w_{21}^1 \\ \vdots \\ w_{n1}^1 \end{bmatrix}_{p \times 1} \right)$$

$$x_{i1} \xrightarrow{w_{11}^0} f(x_i^T w_{.1}^0) \xrightarrow{w_{11}^1} f\left(\sum_{k=1}^p f(x_i^T w_{.k}^0) w_{k1}^1\right) \xrightarrow{\hat{Y}_i} L(\hat{Y}_i) = (\hat{Y}_i - Y_i)^2$$



$$C(\hat{Y}) = \sum_{i=1}^n (Y_i - \hat{Y}_i)^2$$

$$f(A)_{ij} = \frac{1}{1 + e^{-a_{ij}}}$$

# Backpropagation

$$\begin{aligned}\frac{\partial}{\partial a_{ij}} f(A)_{ij} &= \frac{\partial}{\partial a_{ij}} \frac{1}{1 + e^{-a_{ij}}} \\ &= \frac{1}{1 + e^{-a_{ij}}} \left( 1 - \frac{1}{1 + e^{-a_{ij}}} \right)\end{aligned}$$

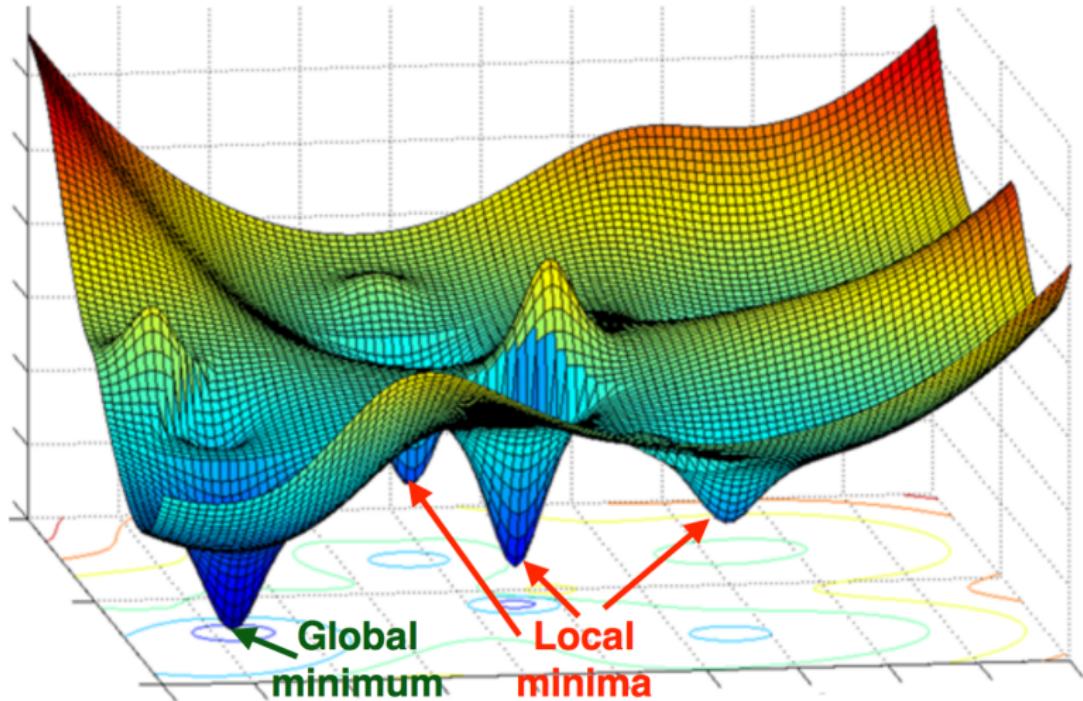
$$\begin{aligned}\frac{\partial}{\partial w_{k1}^1} L(\hat{Y}_i) &= \frac{\partial \hat{Y}_i}{\partial w_{k1}^1} \frac{\partial}{\partial \hat{Y}_i} L(\hat{Y}_i) \\ &= \eta_{ik}^1 \hat{Y}_i (1 - \hat{Y}_i) 2(Y_i - \hat{Y}_i)\end{aligned}$$

$$\begin{aligned}\frac{\partial}{\partial w_{jk}^0} L(\hat{Y}_i) &= \frac{\partial \hat{Y}_i}{\partial w_{jk}^0} \frac{\partial}{\partial \hat{Y}_i} L(\hat{Y}_i) \\ &= \frac{\partial \eta_{ik}^1}{\partial w_{jk}^0} \frac{\partial \hat{Y}_i}{\partial \eta_{ik}^1} \frac{\partial}{\partial \hat{Y}_i} L(\hat{Y}_i) \\ &= x_{ij} \eta_{ik}^1 (1 - \eta_{ik}^1) w_{k1}^1 \hat{Y}_i (1 - \hat{Y}_i) 2(Y_i - \hat{Y}_i)\end{aligned}$$

$$\frac{\partial}{\partial w_{k1}^1} C(\hat{Y}) = \sum_{i=1}^n \frac{\partial}{\partial w_{k1}^1} L(\hat{Y}_i)$$

$$\frac{\partial}{\partial w_{jk}^0} C(\hat{Y}) = \sum_{i=1}^n \frac{\partial}{\partial w_{jk}^0} L(\hat{Y}_i)$$

Sadly, backpropagation (GD) is very weak on DNN's...



Sadly, backpropagation (GD) is very weak on DNN's...

- ▶ If you're using backpropagation anyway, then probably

## Sadly, backpropagation (GD) is very weak on DNN's...

- ▶ If you're using backpropagation anyway, then probably
- ▶ Stochastic gradient decent, or mini-batch

## Sadly, backpropagation (GD) is very weak on DNN's...

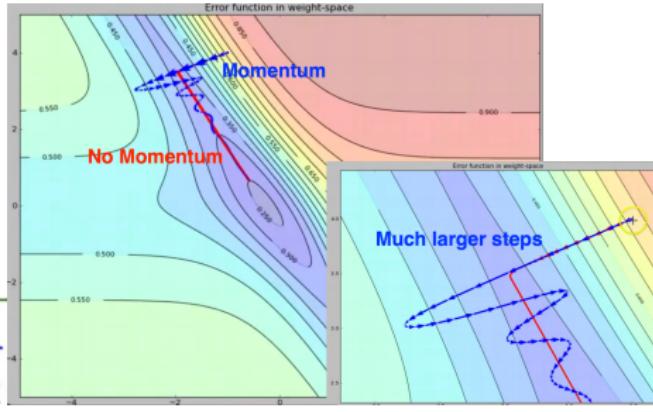
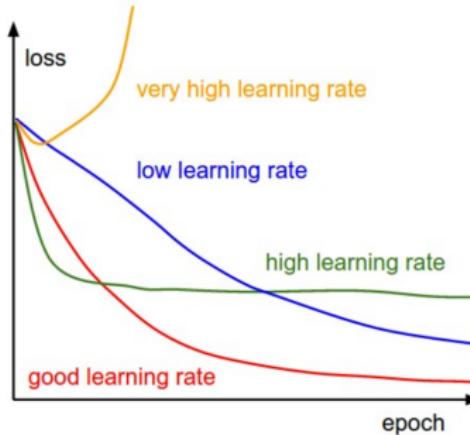
- ▶ If you're using backpropagation anyway, then probably
- ▶ Stochastic gradient decent, or mini-batch
- ▶ Many epochs (iterations through the data)

# Sadly, backpropagation (GD) is very weak on DNN's...

- ▶ If you're using backpropagation anyway, then probably
- ▶ Stochastic gradient decent, or mini-batch
- ▶ Many epochs (iterations through the data)
- ▶ And **momentum** and **velocity**

$$V^{(t)} = \mu V^{(t-1)} - \alpha \nabla Cost(W^{(t-1)})$$

$$W^{(t)} = W^{(t-1)} + V^{(t)}$$



## Unfortunately backpropagation is poor on DNN's...

- ▶ lots of trial and error – no design consensus

## Unfortunately backpropagation is poor on DNN's...

- ▶ lots of trial and error – no design consensus
- ▶ but usually less than 3 hidden layers

## Unfortunately backpropagation is poor on DNN's...

- ▶ lots of trial and error – no design consensus
- ▶ but usually less than 3 hidden layers
- ▶  $2^*p$  neurons (noisy data) to  $30^*p$  neurons (perfect signal)

## Unfortunately backpropagation is poor on DNN's...

- ▶ lots of trial and error – no design consensus
- ▶ but usually less than 3 hidden layers
- ▶  $2^*p$  neurons (noisy data) to  $30^*p$  neurons (perfect signal)
- ▶ fewer outputs than inputs per layer

## Unfortunately backpropagation is poor on DNN's...

- ▶ lots of trial and error – no design consensus
- ▶ but usually less than 3 hidden layers
- ▶  $2^*p$  neurons (noisy data) to  $30^*p$  neurons (perfect signal)
- ▶ fewer outputs than inputs per layer
- ▶ probably rectified linear, perhaps tanh & *maybe* sigmoid AF's

## Unfortunately backpropagation is poor on DNN's...

- ▶ lots of trial and error – no design consensus
- ▶ but usually less than 3 hidden layers
- ▶  $2^*p$  neurons (noisy data) to  $30^*p$  neurons (perfect signal)
- ▶ fewer outputs than inputs per layer
- ▶ probably rectified linear, perhaps tanh & *maybe* sigmoid AF's
- ▶ and tricks:

[research.microsoft.com/pubs/192769/tricks-2012.pdf](http://research.microsoft.com/pubs/192769/tricks-2012.pdf)

## Unfortunately backpropagation is poor on DNN's...

- ▶ lots of trial and error – no design consensus
- ▶ but usually less than 3 hidden layers
- ▶  $2^*p$  neurons (noisy data) to  $30^*p$  neurons (perfect signal)
- ▶ fewer outputs than inputs per layer
- ▶ probably rectified linear, perhaps tanh & *maybe* sigmoid AF's
- ▶ and tricks:

[research.microsoft.com/pubs/192769/tricks-2012.pdf](http://research.microsoft.com/pubs/192769/tricks-2012.pdf)

- ▶ go find a NN paper that does \*something\* like what you're trying to do, get theirs to work, and then adapt it to your data

# What data/information is contained in an image?



What data/information is contained in an image?

A color image is a *tensor*: a matrix of vectors

$$\begin{bmatrix} \textcolor{red}{RGB} & \textcolor{red}{RGB} & \cdots & \textcolor{red}{RGB} \\ \textcolor{red}{RGB} & \textcolor{red}{RGB} & \cdots & \textcolor{red}{RGB} \\ \vdots & \vdots & \ddots & \vdots \\ \textcolor{red}{RGB} & \textcolor{red}{RGB} & \cdots & \textcolor{red}{RGB} \end{bmatrix}$$

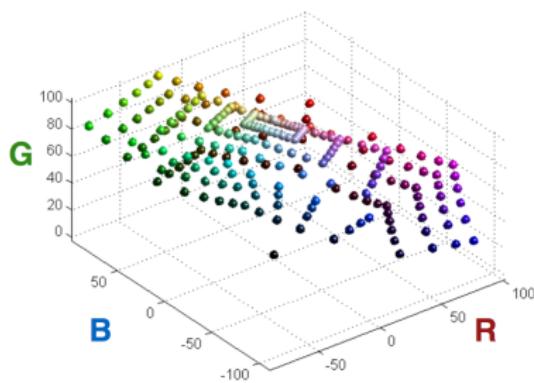
`image.shape = (width, height,3)`

# What data/information is contained in an image?

A color image is a *tensor*: a matrix of vectors

$$\begin{bmatrix} RGB & RGB & \dots & RGB \\ RGB & RGB & \dots & RGB \\ \vdots & \vdots & \ddots & \vdots \\ RGB & RGB & \dots & RGB \end{bmatrix}$$

image.shape = (width, height,3)



What data/information is contained in an image?

A color image is a *tensor*: a matrix of vectors

$$\begin{bmatrix} \textcolor{red}{RGB} & \textcolor{red}{RGB} & \cdots & \textcolor{red}{RGB} \\ \textcolor{red}{RGB} & \textcolor{red}{RGB} & \cdots & \textcolor{red}{RGB} \\ \vdots & \vdots & \ddots & \vdots \\ \textcolor{red}{RGB} & \textcolor{red}{RGB} & \cdots & \textcolor{red}{RGB} \end{bmatrix}$$

`image.shape = (width, height,3)`

Grayscale (luminescence):  $0.2125\textcolor{red}{R} + 0.7154\textcolor{green}{G} + 0.0721\textcolor{blue}{B}$

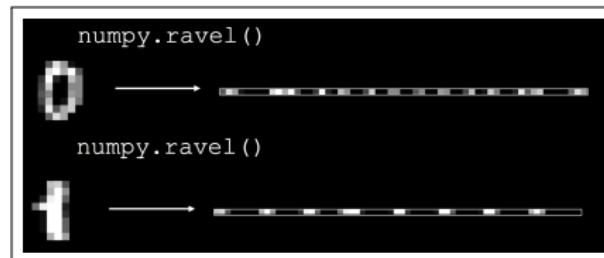
# What data/information is contained in an image?

A color image is a *tensor*: a matrix of vectors

$$\begin{bmatrix} \textcolor{red}{RGB} & \textcolor{red}{RGB} & \cdots & \textcolor{red}{RGB} \\ \textcolor{red}{RGB} & \textcolor{red}{RGB} & \cdots & \textcolor{red}{RGB} \\ \vdots & \vdots & \ddots & \vdots \\ \textcolor{red}{RGB} & \textcolor{red}{RGB} & \cdots & \textcolor{red}{RGB} \end{bmatrix}$$

image.shape = (width, height, 3)

Grayscale (luminescence):  $0.2125R + 0.7154G + 0.0721B$

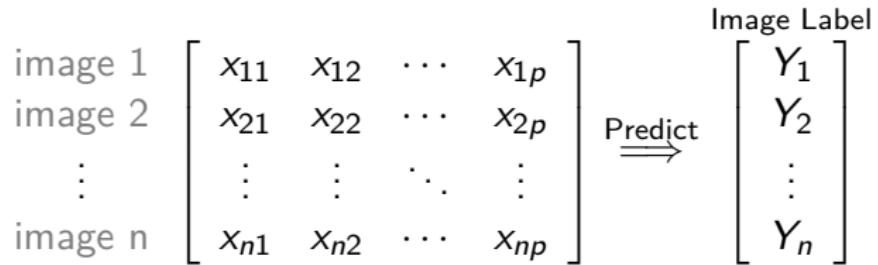


featurized.shape = (width·height·3,1)

# What data/information is contained in an image?

$$\begin{array}{l} \text{image 1} \\ \text{image 2} \\ \vdots \\ \text{image } n \end{array} \left[ \begin{array}{cccc} x_{11} & x_{12} & \cdots & x_{1p} \\ x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{np} \end{array} \right] \xrightarrow{\text{Predict}} \begin{array}{l} \text{Image Label} \\ \left[ \begin{array}{c} Y_1 \\ Y_2 \\ \vdots \\ Y_n \end{array} \right] \end{array}$$

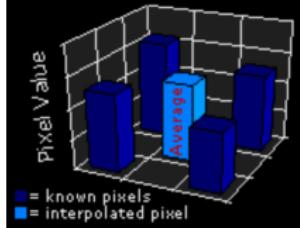
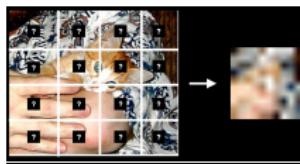
# What data/information is contained in an image?



mite	container ship	motor scooter	leopard
black widow cockroach tick starfish	lifeboat amphibian fireboat drilling platform	motor scooter go-kart moped bumper car golfcart	leopard jaguar cheetah snow leopard Egyptian cat
grille	mushroom	cherry	Madagascar cat
convertible grille pickup beach wagon fire engine	agaric mushroom jelly fungus gill fungus dead-man's-fingers	dalmatian grape elderberry ffordshire bulterrier currant	squirrel monkey spider monkey titi indri howler monkey

# What data/information is contained in an image?

$$\begin{array}{l} \text{image 1} \\ \text{image 2} \\ \vdots \\ \text{image } n \end{array} \left[ \begin{array}{cccc} x_{11} & x_{12} & \cdots & x_{1p} \\ x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{np} \end{array} \right] \xrightarrow{\text{Predict}} \left[ \begin{array}{c} Y_1 \\ Y_2 \\ \vdots \\ Y_n \end{array} \right]$$



mite	container ship	motor scooter	leopard
black widow	lifeboat	motor scooter	jaguar
cockroach	amphibian	go-kart	cheetah
tick	fireboat	moped	snow leopard
starfish	drilling platform	bumper car	Egyptian cat



grille	mushroom	cherry	Madagascar cat
convertible	agaric	dalmatian	squirrel monkey
grille	mushroom	grape	spider monkey
pickup	jelly fungus	elderberry	titi
beach wagon	gill fungus	ffordshire bullterrier	indri
fire engine	dead-man's-fingers	currant	howler monkey

# Image Processing

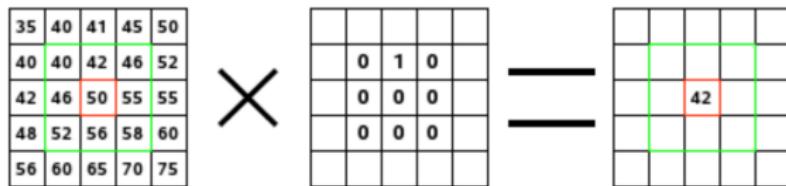
- ▶ Remove unnecessary details to allow for better generalization of images to image classes

$$x_i \rightarrow y_i : \operatorname{argmin}_y \quad \frac{1}{2} \sum_{Fidelity} (x_i - y_i)^2 + \lambda \sum_{Variation} |y_{i+1} - y_i|$$



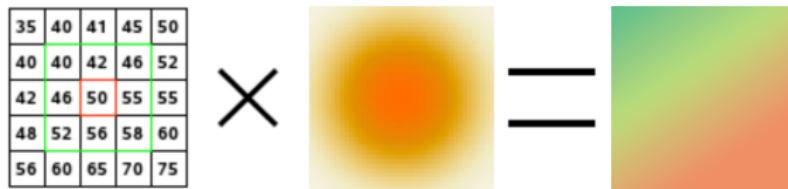
# Convolutional Kernel

- ▶ Spatial proximity and pixel agreement (*bi-lateral*)



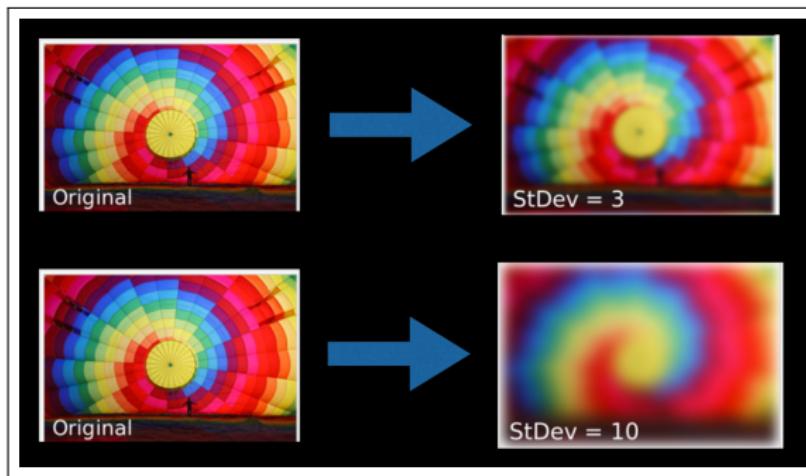
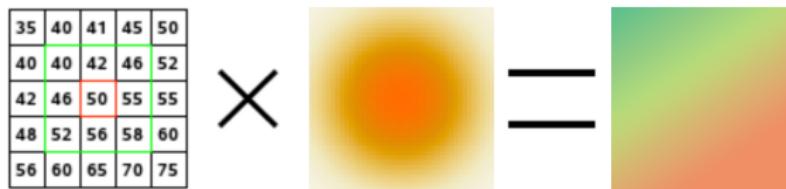
# Convolutional Kernel

- ▶ Spatial proximity *and* pixel agreement (*bi-lateral*)



# Convolutional Kernel

- Spatial proximity *and* pixel agreement (*bi-lateral*)

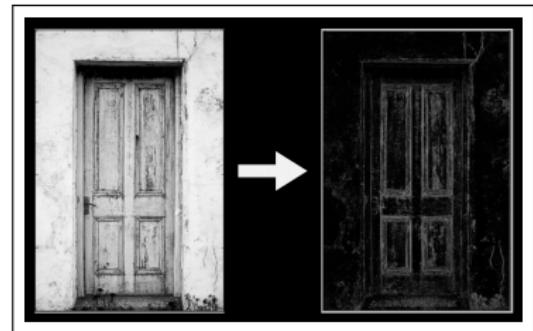


# Convolutional Kernel

It may be easier to identify objects if we just consider the edges

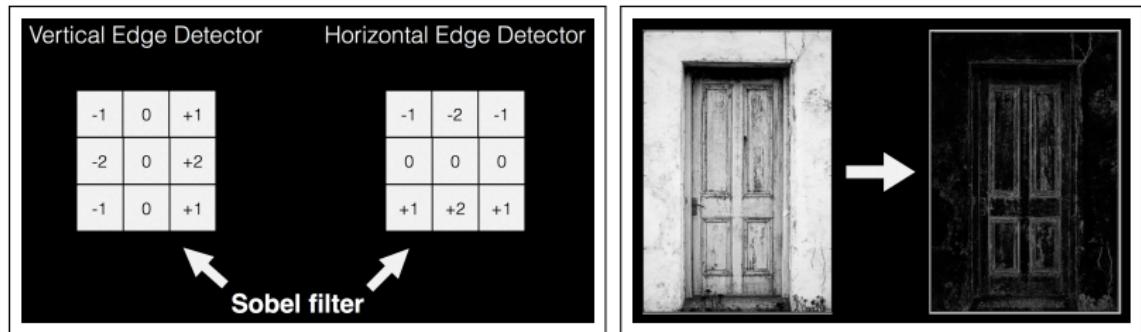
Vertical Edge Detector	Horizontal Edge Detector
$\begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}$	$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix}$

**Sobel filter**



# Convolutional Kernel

It may be easier to identify objects if we just consider the edges

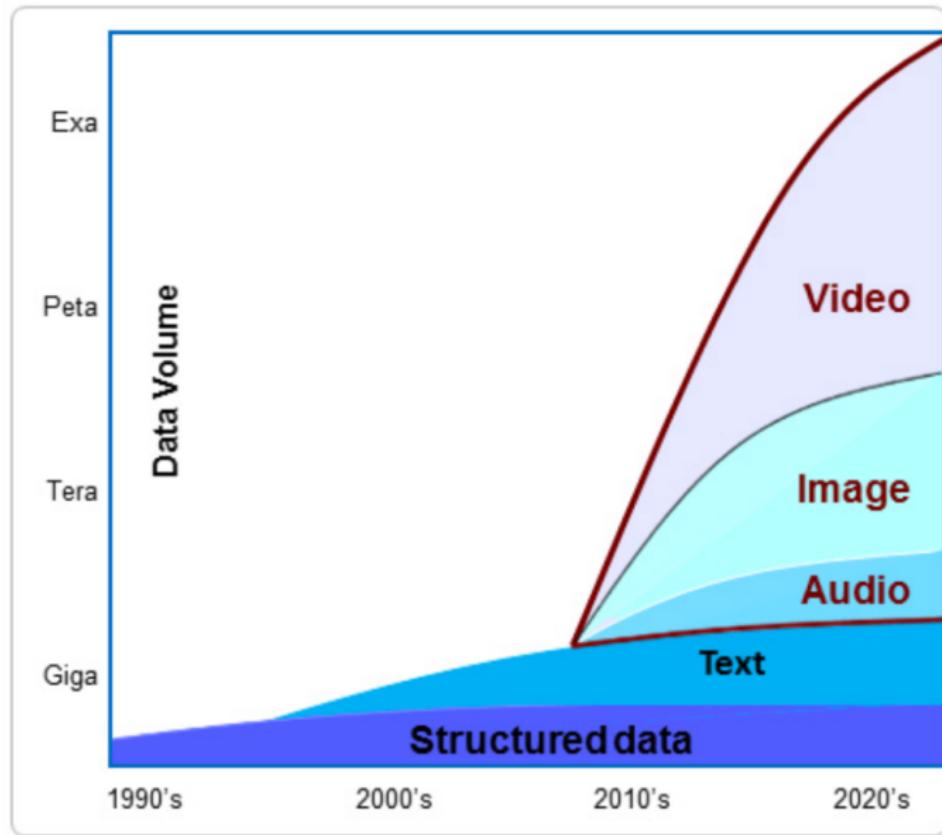


Canny Filter

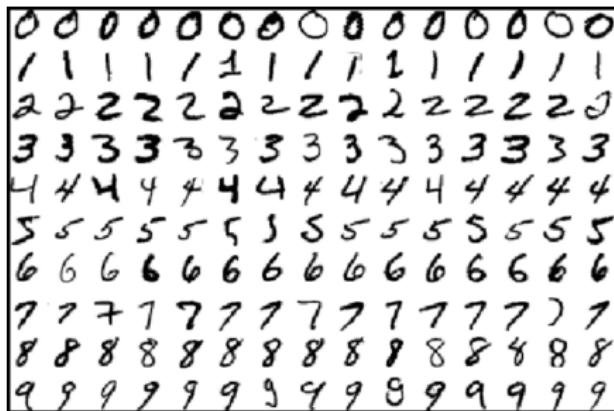


Sobel Filter

# The new type of data



## NNs are the best at identifying patterns in complex data

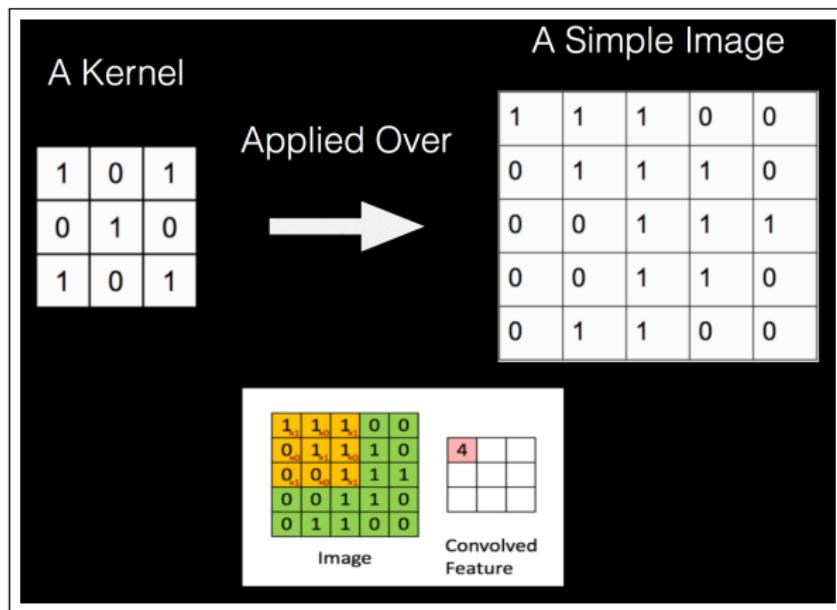


Classifier	Test Error Rate
Large and Deep Convolutional Network	0.33%
SVM with degree 9 polynomial kernal	0.66%
Gradient boosted stumps on Haar features	0.87%

Neural networks can recognize patterns from complex, heavy-tailed inputs such as image, audio, video, text, and human speech data

## Returning to Convolutions...

Recall that a convolution matrix (or kernel or mask) in image processing is a small matrix that can be used to blur, sharpen, emboss, detect edges, etc. through *convolutions* with an image



# Returning to Convolutions...

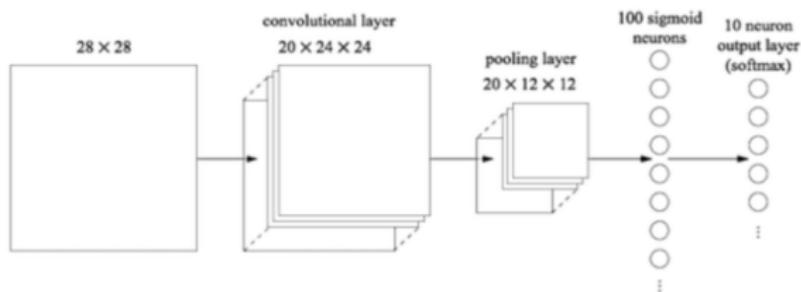


idea: *denovo* kernels

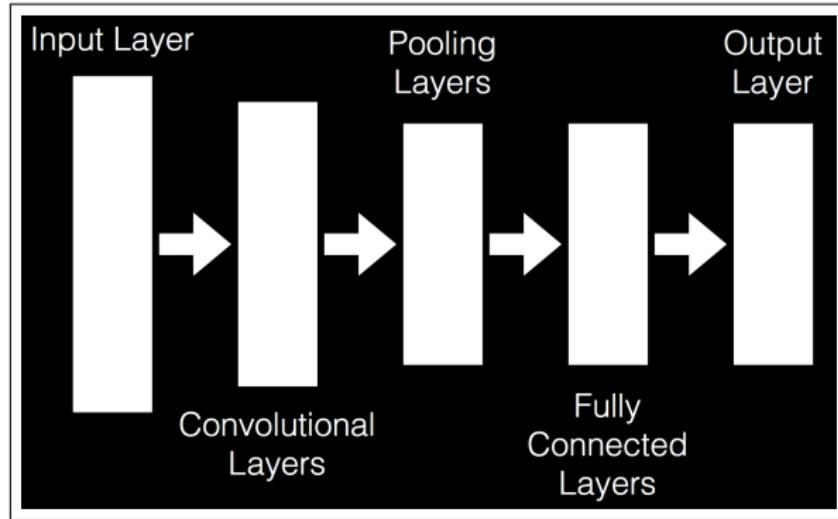
Can NN's learn their own convolution matrices?

idea: *denovo* kernels

Can NN's learn their own convolution matrices? **Yep.**

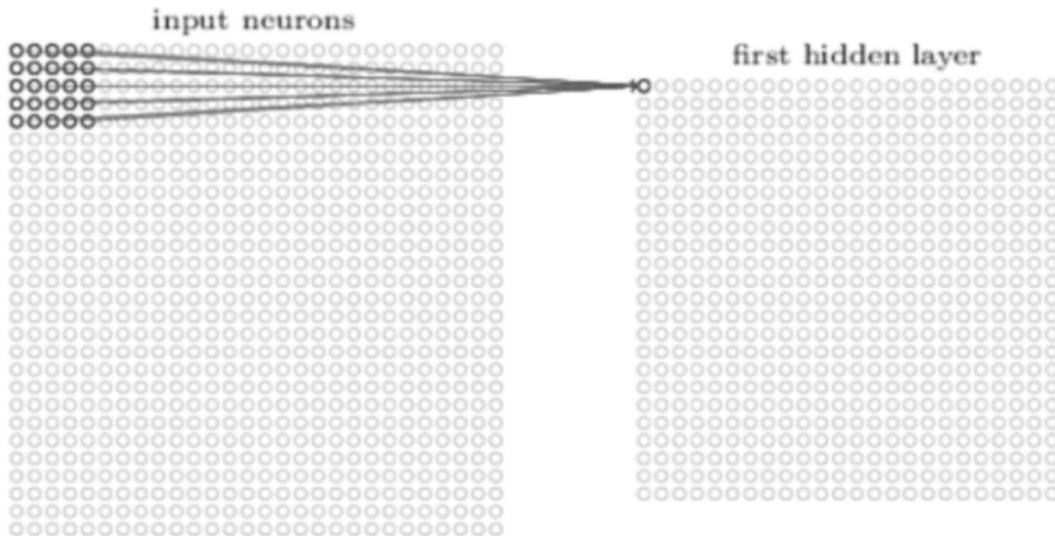


# Convolutional Neural Network's – CovNets, or CNN's



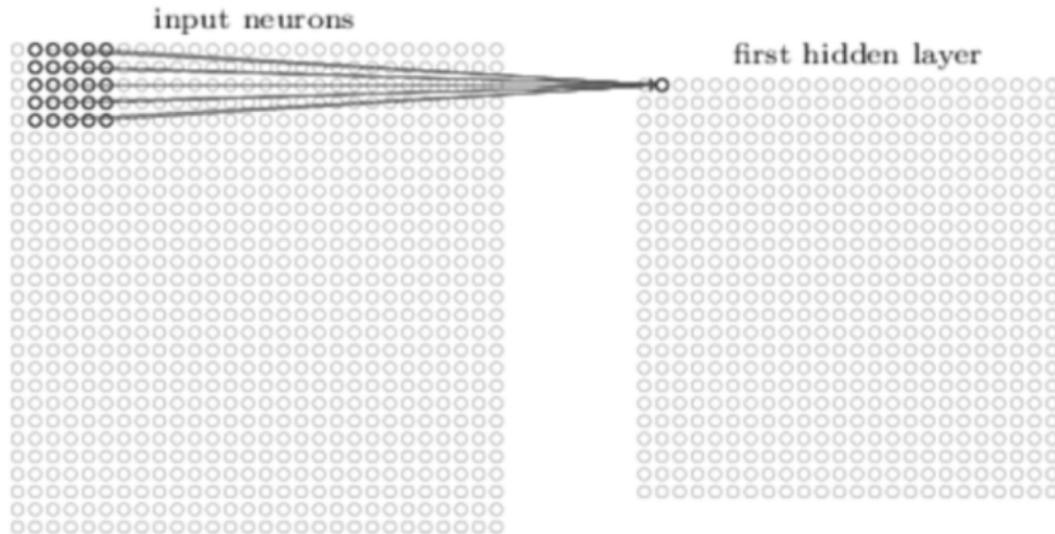
There are three key features that make this NN structure actually work: (1) local receptive fields, (2) shared weights, and (3) pooling

# 1. Local Receptive Field



- ▶ A group of pixels are a *local receptive field*
- ▶ Defined by the size of the kernel
- ▶ The image is transformed into the set of local receptive fields

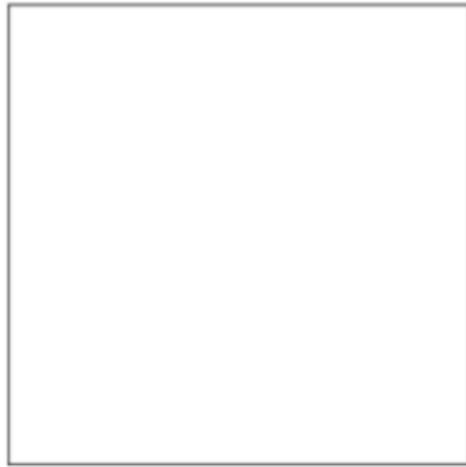
# 1. Local Receptive Field



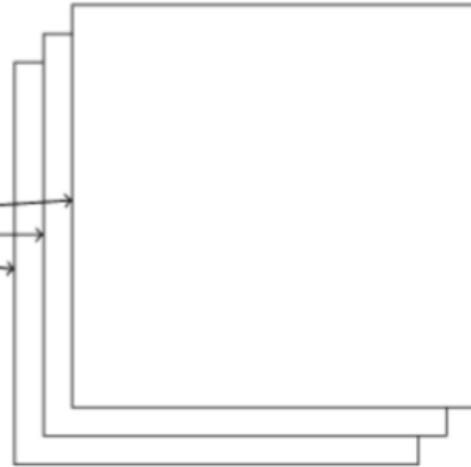
- ▶ A group of pixels are a *local receptive field*
- ▶ Defined by the size of the kernel
- ▶ The image is transformed into the set of local receptive fields

## 2. Shared Weights

$28 \times 28$  input neurons



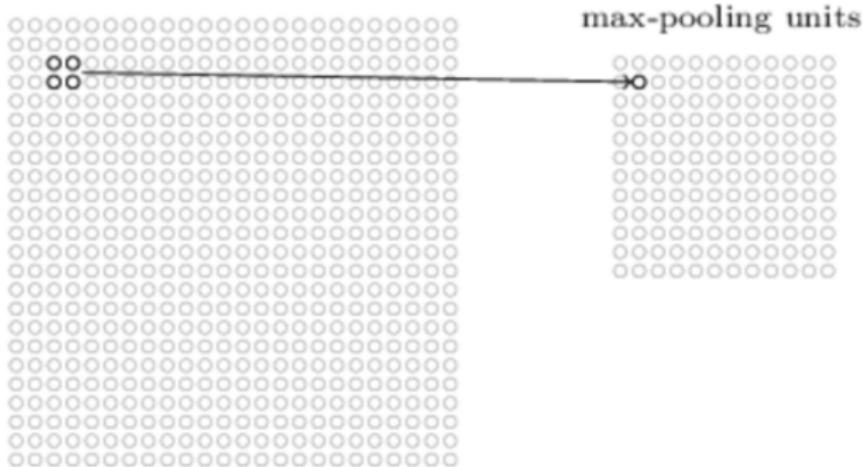
first hidden layer:  $3 \times 24 \times 24$  neurons



- ▶ Multiple convolutions are learned/used
- ▶ Weights within a convolution are (obviously) shared

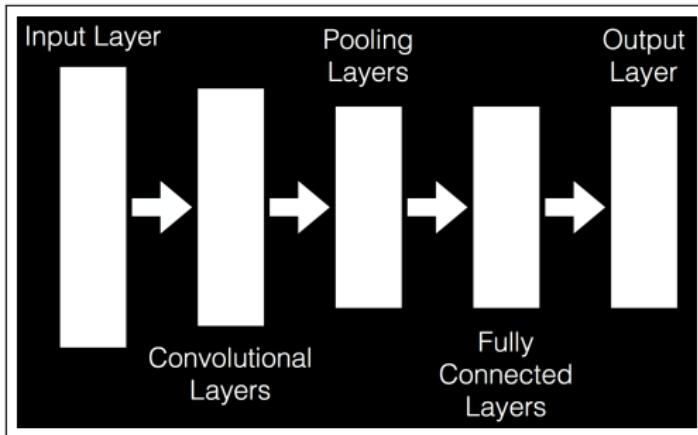
### 3. Pooling

hidden neurons (output from feature map)



- ▶ Convolutional layers are simplified using, e.g., *max pooling*
- ▶ This reduces computational complexity in downstream layers
- ▶ In addition it provides a form of translational invariance

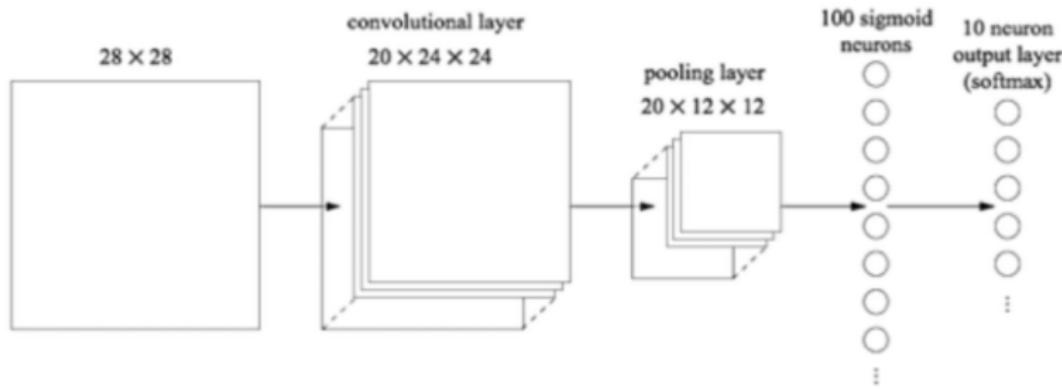
# Fully Connected and Output Layers



- ▶ Fully connected layers hierarchically aggregate learned features
  - they produce higher order features in standard NN manner
- ▶ The output layer estimates the one-versus-all class probability
- ▶ This will not be quite normalized, but (the multi-armed bandit) *softmax* can provide complete class probabilities

$$\frac{e^{\text{Pr}(Y_k=1)}}{\sum_{k=1}^K e^{\text{Pr}(Y_k=1)}}$$

So a final CNN might look something like this...



# Recurrent Neural Network (RNN)

- ▶ A RNN is a NN where a hidden layer feeds back into itself
- ▶ This allows the NN to exhibit dynamic temporal behavior
- ▶ RNN's provide “internal memory” for sequence processing
- ▶ They are very applicable to handwriting/speech recognition

<https://github.com/SirDudeness/BassGenerator>

# Mores

[github.com/sallamander/neural-networks-intro](https://github.com/sallamander/neural-networks-intro)

[Provides introductions in NumPy, Theano, Tensorflow, & Keras]

[neuralnetworksanddeeplearning.com](http://neuralnetworksanddeeplearning.com) [General introduction]

[cs231n.github.io/](https://cs231n.github.io/) [Convolution NN's for visual recognition]

[deeplearning.net/tutorial/lenet.html](http://deeplearning.net/tutorial/lenet.html) [Mathematics]

[arxiv.org/abs/1503.02531](https://arxiv.org/abs/1503.02531) [Interpretation detractions]

[Projects]

[lasagne.readthedocs.org/en/latest/index.html](http://lasagne.readthedocs.org/en/latest/index.html)

(also, see nolearn for an scikit learn type of interface)

[cbinsights.com/blog/python-tools-machine-learning](http://cbinsights.com/blog/python-tools-machine-learning)

[deeplearning.net/tutorial/lenet.html](http://deeplearning.net/tutorial/lenet.html)

[deeplearning4j.org](http://deeplearning4j.org)

[caffe.berkeleyvision.org](http://caffe.berkeleyvision.org)

“How does deep learning work and how is it different from normal neural net works applied with SVM” on quora.com differentiates deep learning from other methods. Hinton’s 2007 Google talk “The Next Generation of Neural Networks” introduces deep learning

# Fine

NN's do not **currently** make machine learning any *easier*...