Neural Nets

Schwartz

August 2, 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Upon his death, the titular Reverend Thomas Bayes (1701–1761) of the renowned theorem left an unpublished manuscript deriving the distribution for the parameter of a binomial distribution.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-organizating 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 $\Longrightarrow$ regression
- abstract feature encoding via layers
- backpropegation (gradient decent via chain rule)

- backpropegation 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 $\left\{\eta_j^{(l)}\right\}$

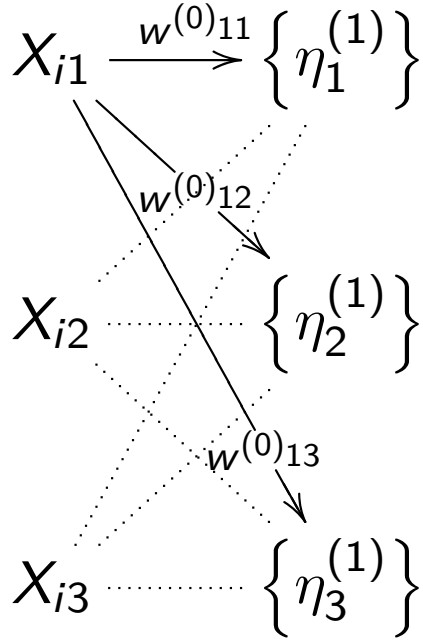
$$\left\{\eta_1^{(1)}\right\}$$

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

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

Artificial Neural Network (NN)

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



Artificial Neural Network (NN)

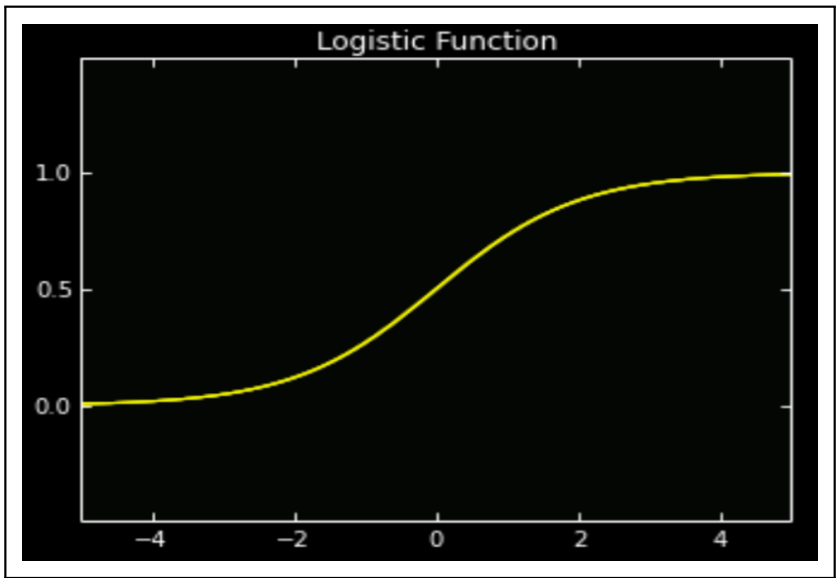
- *NN's* are a collection of nodes $\eta_j^{(l)}$ that take on various states $\left\{\eta_j^{(l)}\right\}$
- Features are embedded into NNs as collections of states

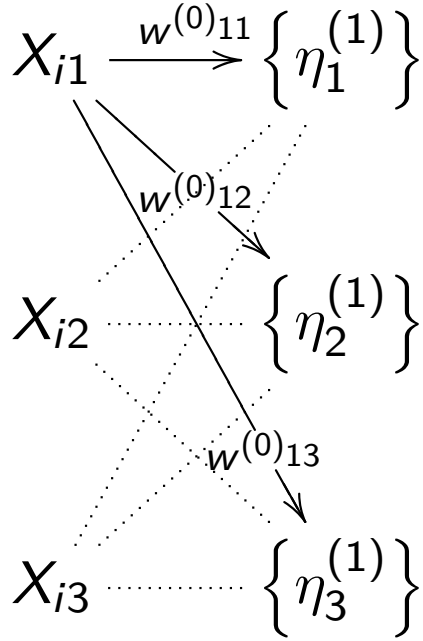
E.g., $\left\{\eta_1^{(1)}\right\} = f\left(X^T w_1^{(0)}\right)$ where

$$X_i = (X_{i1}, X_{i2}, X_{i3})^T$$

$$w_j^{(0)} = \left(w^{(0)}{}_{1j}, w^{(0)}{}_{2j}, w^{(0)}{}_{3j}\right)^T$$

with activation function f





Artificial Neural Network (NN)

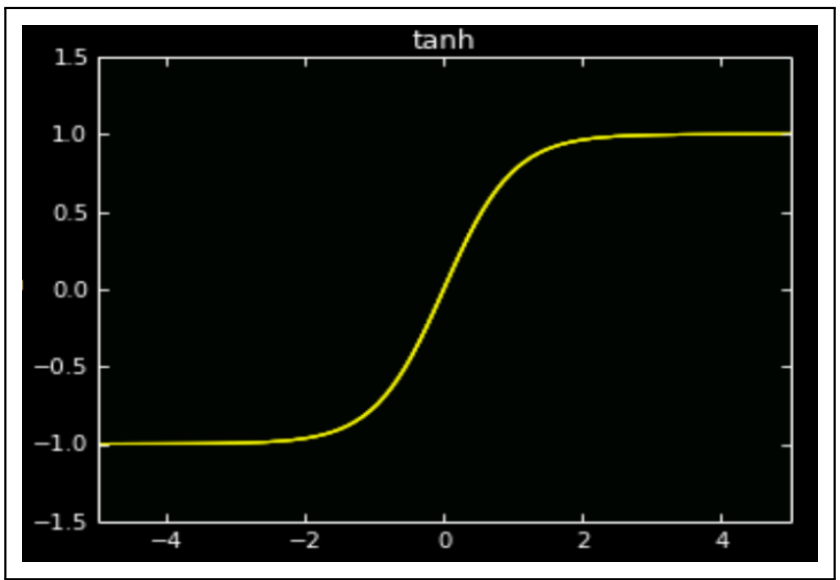
- *NN's* are a collection of nodes $\eta_j^{(l)}$ that take on various states $\left\{\eta_j^{(l)}\right\}$
- Features are embedded into NNs as collections of states

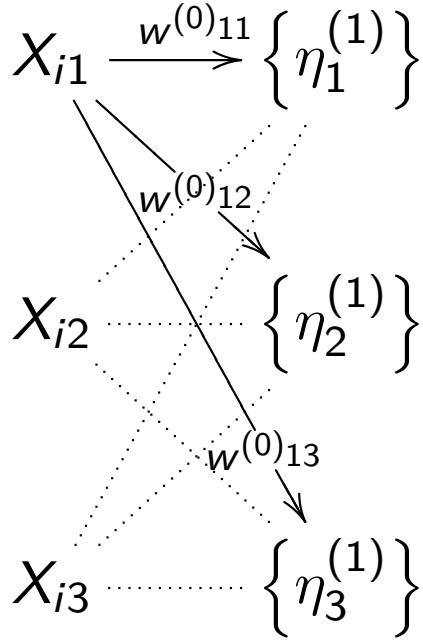
E.g., $\left\{\eta_1^{(1)}\right\} = f\left(X^T w_1^{(0)}\right)$ where

$$X_i = (X_{i1}, X_{i2}, X_{i3})^T$$

$$w_j^{(0)} = \left(w^{(0)}{}_{1j}, w^{(0)}{}_{2j}, w^{(0)}{}_{3j}\right)^T$$

with activation function f





Artificial Neural Network (NN)

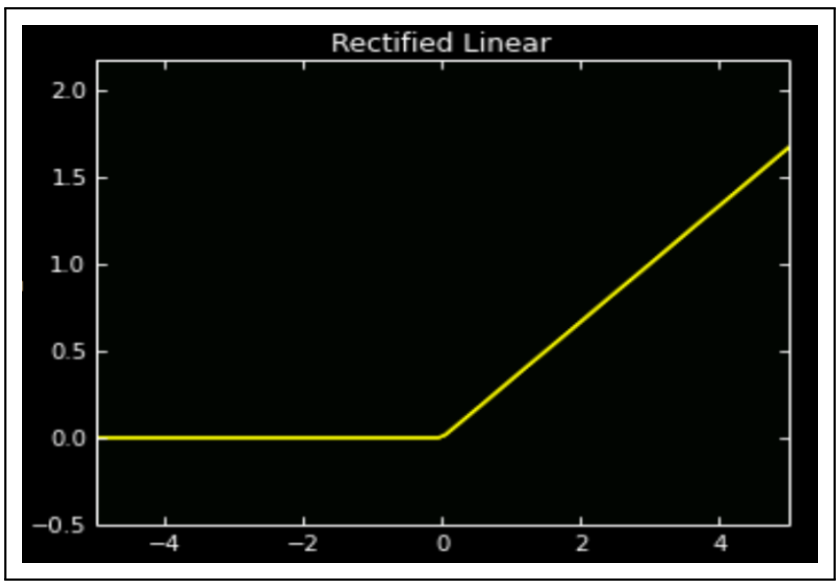
- *NN's* are a collection of nodes $\eta_j^{(l)}$ that take on various states $\left\{\eta_j^{(l)}\right\}$
- Features are embedded into NNs as collections of states

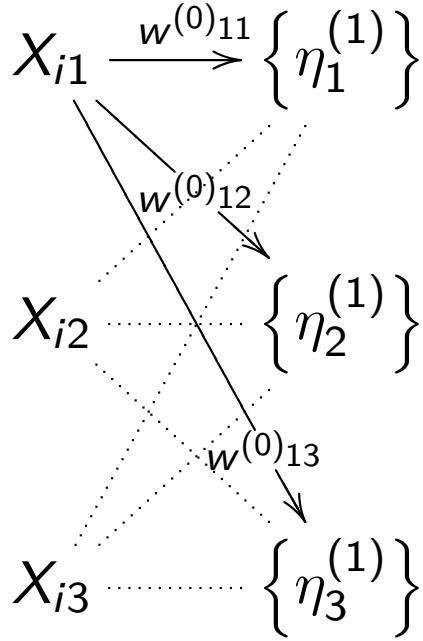
E.g., $\left\{\eta_1^{(1)}\right\} = f\left(X^T w_1^{(0)}\right)$ where

$$X_i = (X_{i1}, X_{i2}, X_{i3})^T$$

$$w_j^{(0)} = \left(w^{(0)}{}_{1j}, w^{(0)}{}_{2j}, w^{(0)}{}_{3j}\right)^T$$

with activation function f





Artificial Neural Network (NN)

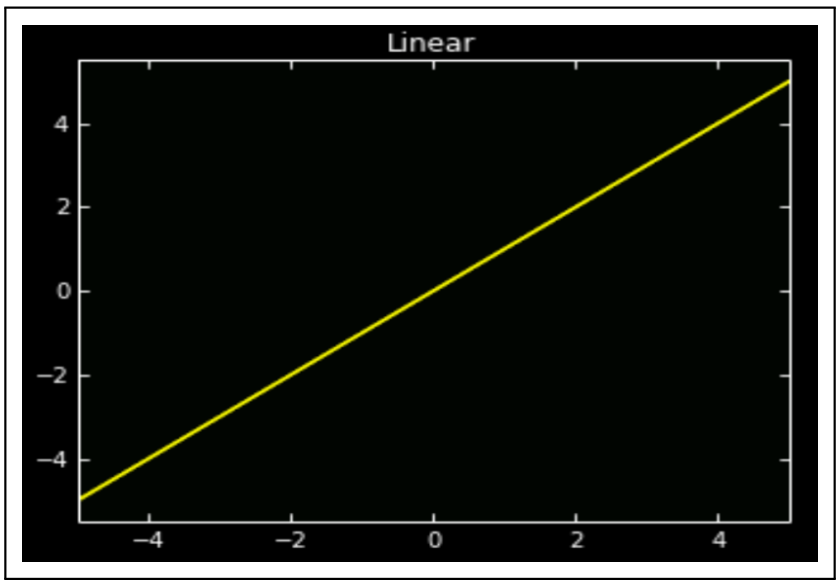
- *NN's* are a collection of nodes $\eta_j^{(l)}$ that take on various states $\left\{\eta_j^{(l)}\right\}$
- Features are embedded into NNs as collections of states

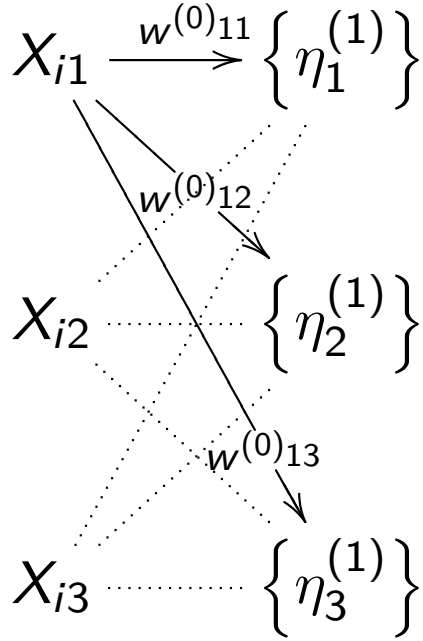
E.g., $\left\{\eta_1^{(1)}\right\} = f\left(X^T w_1^{(0)}\right)$ where

$$X_i = (X_{i1}, X_{i2}, X_{i3})^T$$

$$w_j^{(0)} = \left(w^{(0)}{}_{1j}, w^{(0)}{}_{2j}, w^{(0)}{}_{3j}\right)^T$$

with activation function f

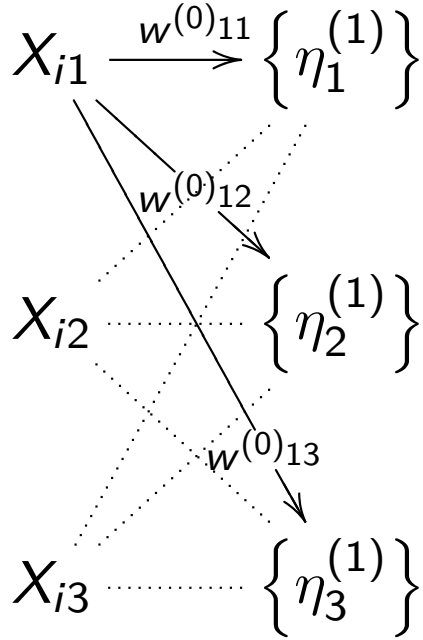




Artificial Neural Network (NN)

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

  Action potentials fire when a neuron (node) is turned "on"

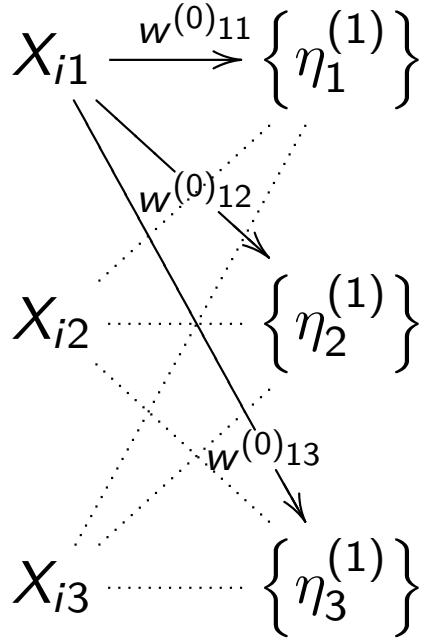


Artificial Neural Network (NN)

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

  Action potentials fire when a neuron (node) is turned "on"

  capturing an "idea" or "concept"

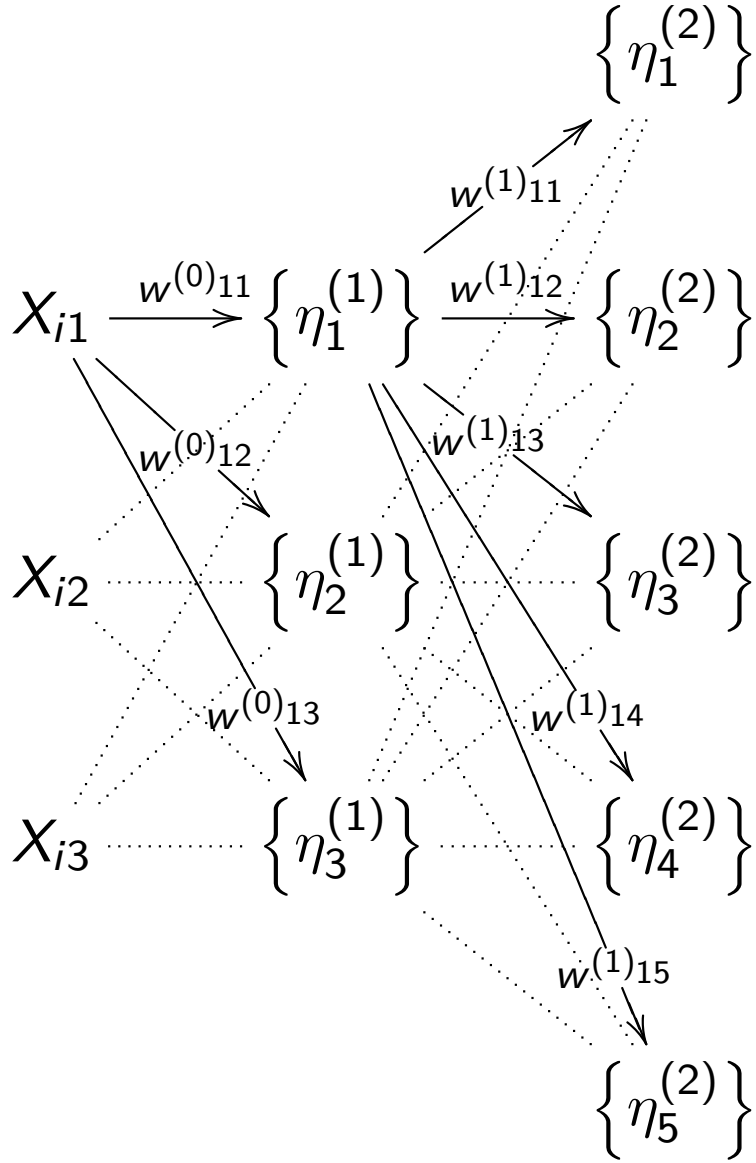


Artificial Neural Network (NN) [thoughts?]

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

  Action potentials fire when a neuron (node) is turned "on"

  capturing an "idea" or "concept"
- These can be further combined in subsequent layers of states



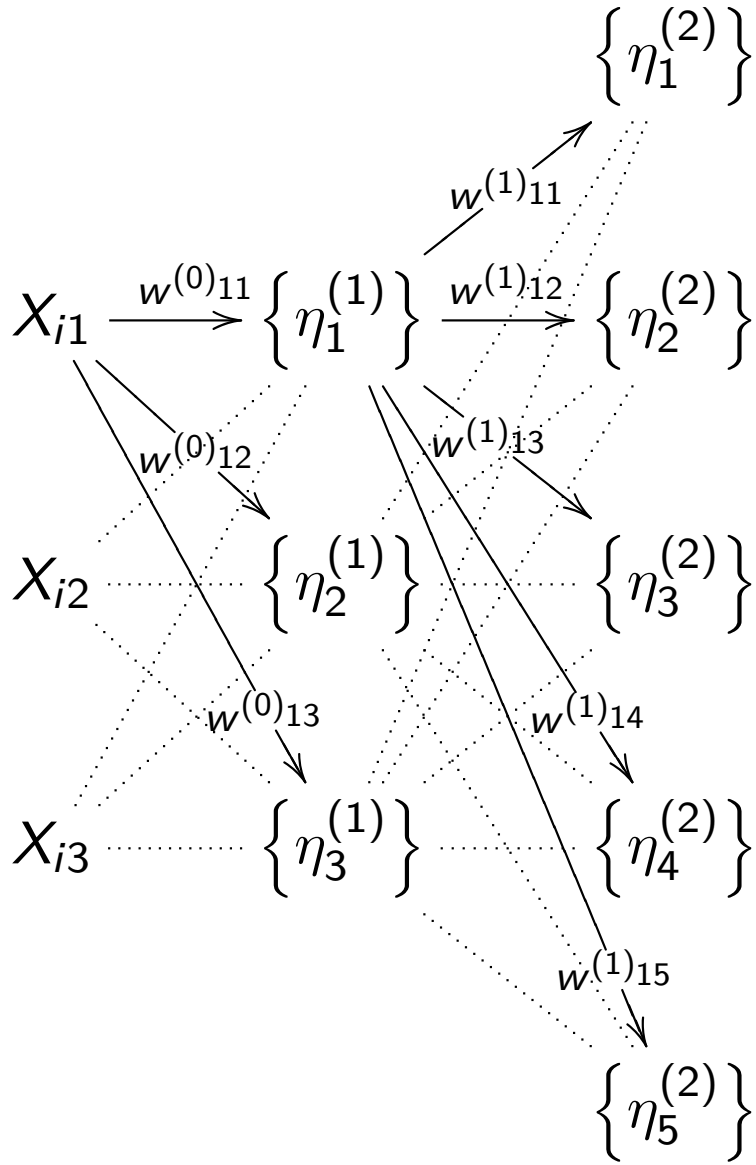
Artificial Neural Network (NN) [thoughts?]

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

  Action potentials fire when a neuron (node) is turned "on"

  capturing an "idea" or "concept"
- These can be further combined in subsequent layers of states

  Each hierarchical layer provides higher orders of abstraction



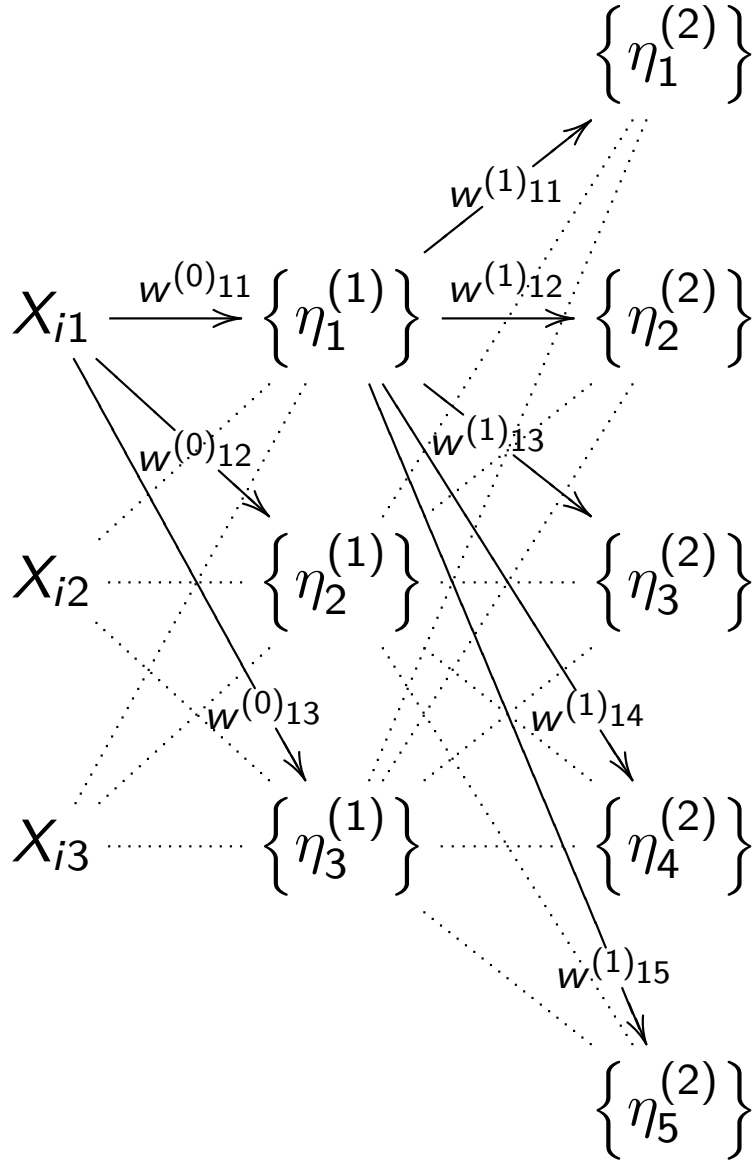
Artificial Neural Network (NN) [thoughts?]

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

  Action potentials fire when a neuron (node) is turned "on"

  capturing 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 representation of data...



Artificial Neural Network (NN) [thoughts?]

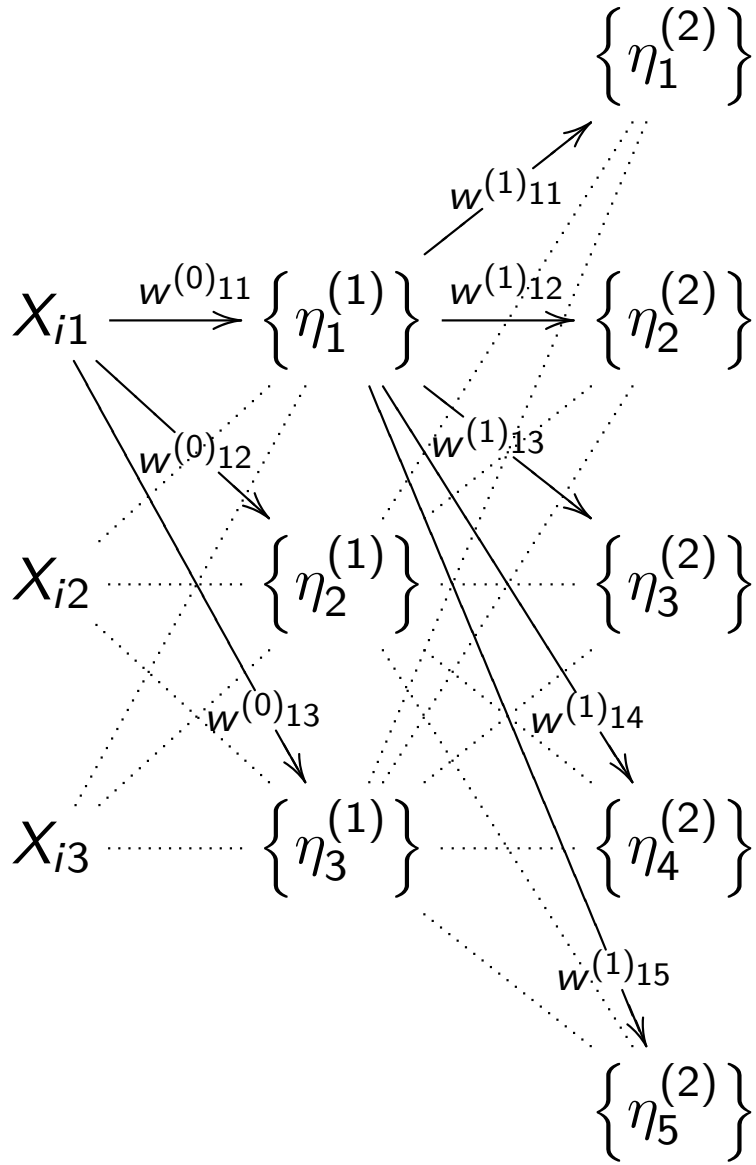
- *NN's* are a collection of nodes $\eta_j^{(l)}$ that take on various states $\left\{\eta_j^{(l)}\right\}$
- Features are embedded into NNs as collections of states

  Action potentials fire when a neuron (node) is turned "on"

  capturing 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 representation of data...

  Similar states for "similar" X_i's means the NN knows, or understands or recognizes X_i

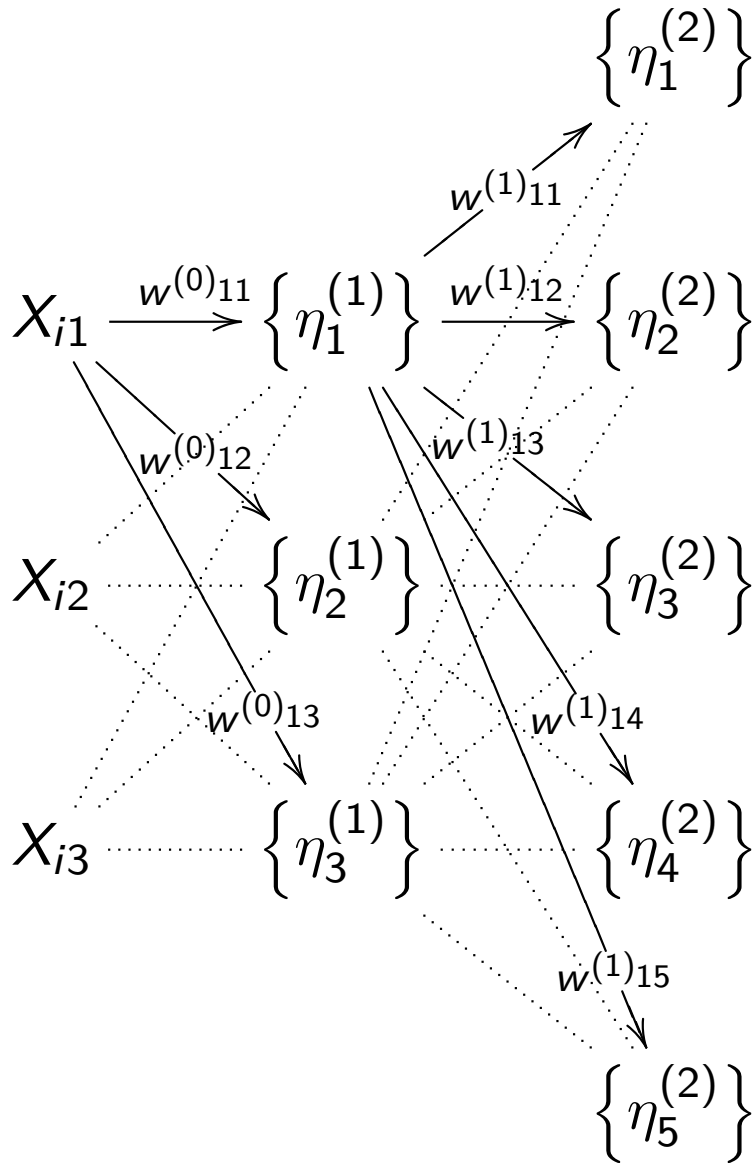


Artificial Neural Network (NN) [labels?]

- The (non-linear) activation functions maps, e.g., $X^T w_1^{(0)}$

  $\left(\text{and then, e.g., } \left\{ \eta^{(1)} \right\}^T w_1^{(1)} \right)$

  onto a set of action potentials

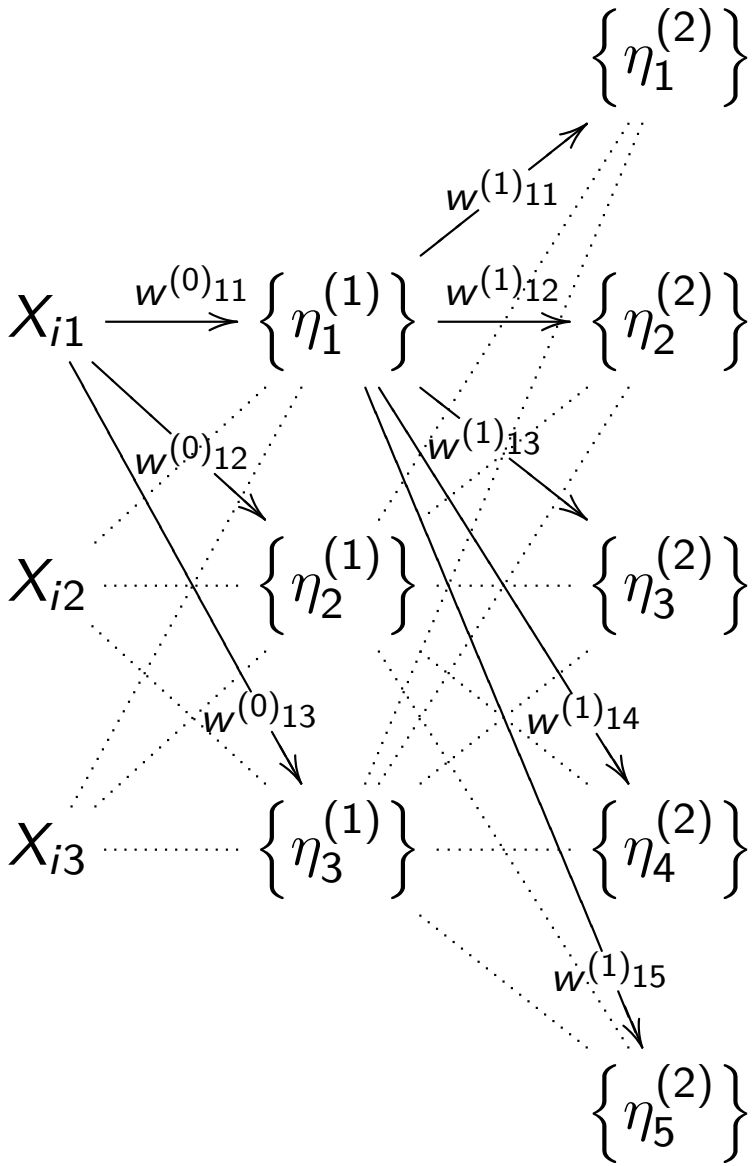


Artificial Neural Network (NN) [labels?]

- The (non-linear) activation functions maps, e.g., $X^T w_1^{(0)}$

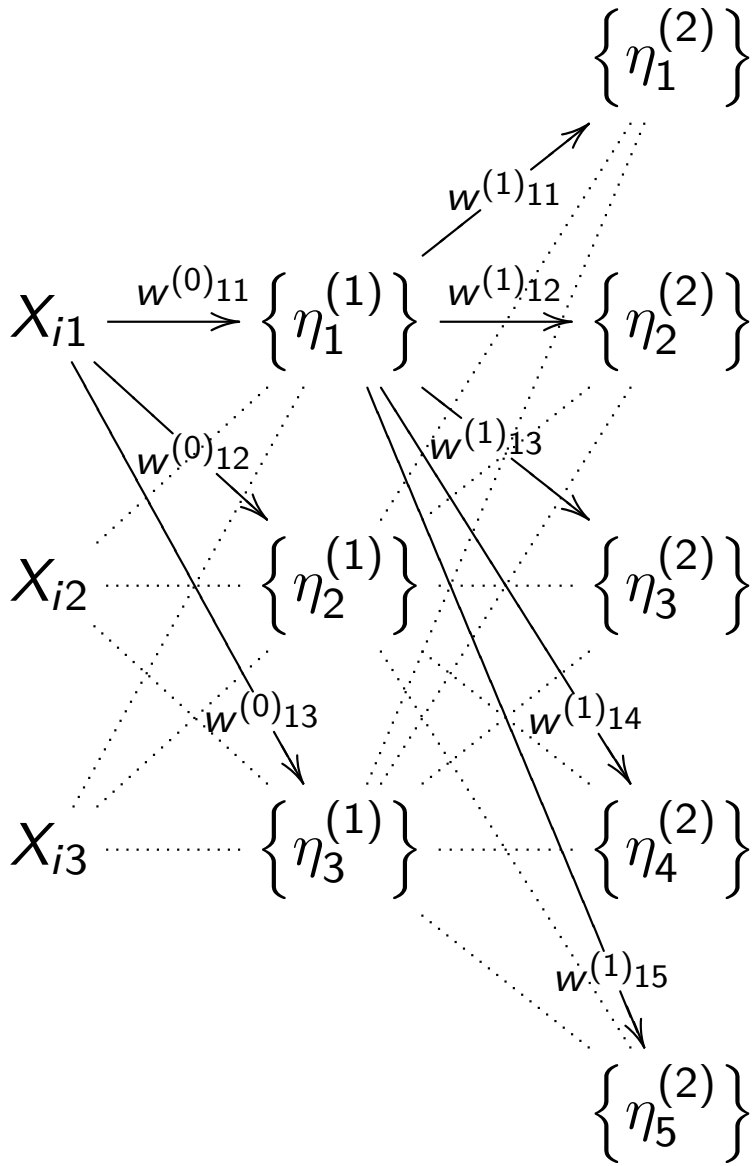
  $\left(\text{and then, e.g., } \left\{ \eta^{(1)} \right\}^T w_1^{(1)} \right)$

  onto a set of action potentials
- Input samples are embedded into NN's as a set of ("on" or "off") activations encoding one of many possible NN states or "thoughts"



Artificial Neural Network (NN) [labels?]

- The (non-linear) activation functions maps, e.g., $X^T w_1^{(0)}$
  $\left(\text{and then, e.g., } \left\{ \eta^{(1)} \right\}^T w_1^{(1)} \right)$
  onto a set of action potentials
- Input samples are embedded into NN's as a set of ("on" or "off") activations encoding one of many possible NN states or "thoughts"
- The topology (size and number of layers) of the NN influences the number of "ideas" and level of cumulative abstraction applied

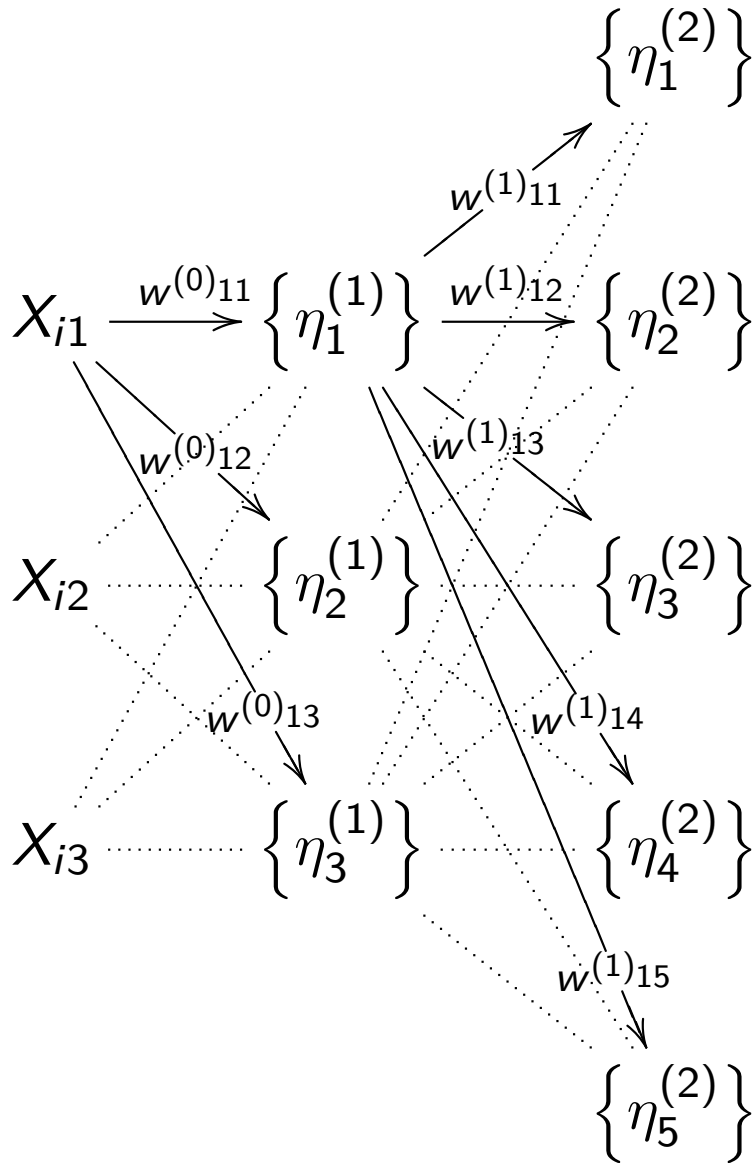


Artificial Neural Network (NN) [labels?]

- The (non-linear) activation functions maps, e.g., $X^T w_1^{(0)}$

  $\left(\text{and then, e.g., } \left\{ \eta^{(1)} \right\}^T w_1^{(1)} \right)$

  onto a set of action potentials
- Input samples are embedded into NN's as a set of ("on" or "off") activations encoding one of many possible NN states or "thoughts"
- The topology (size and number of layers) of the NN influences the number of "ideas" and level of cumulative abstraction applied
- The $w's$ are trained to identify input patterns that should map to similar NN states and "thoughts"...

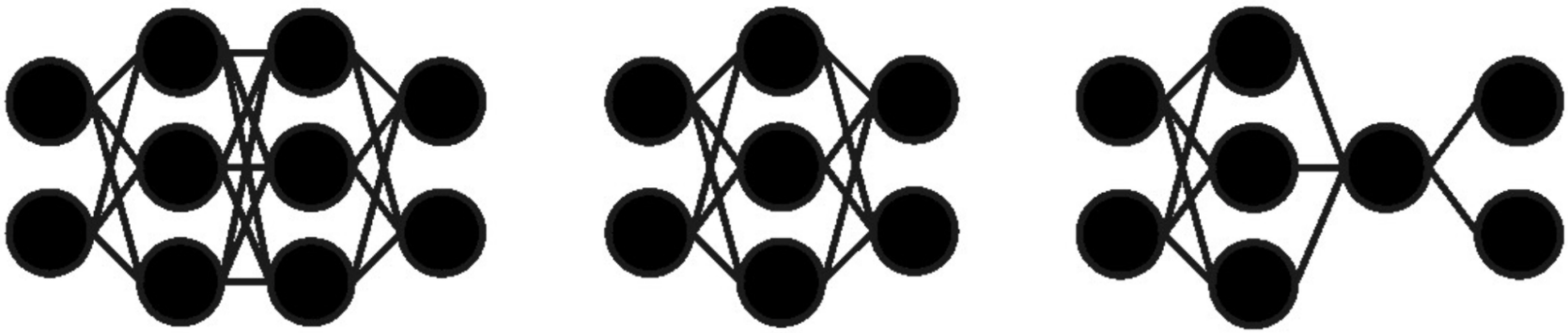


Using NN's: where the fun ends and the pain begins

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

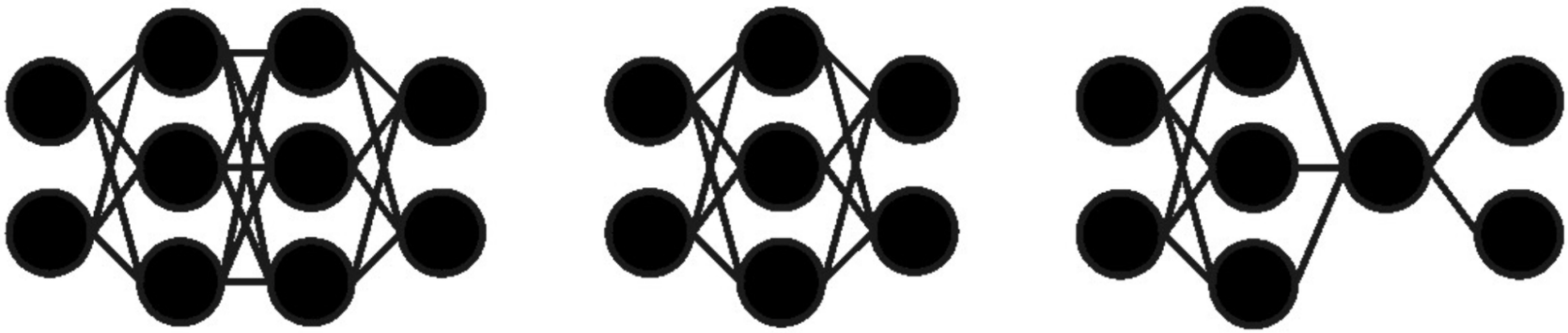
Using NN's: where the fun ends and the pain begins

- 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

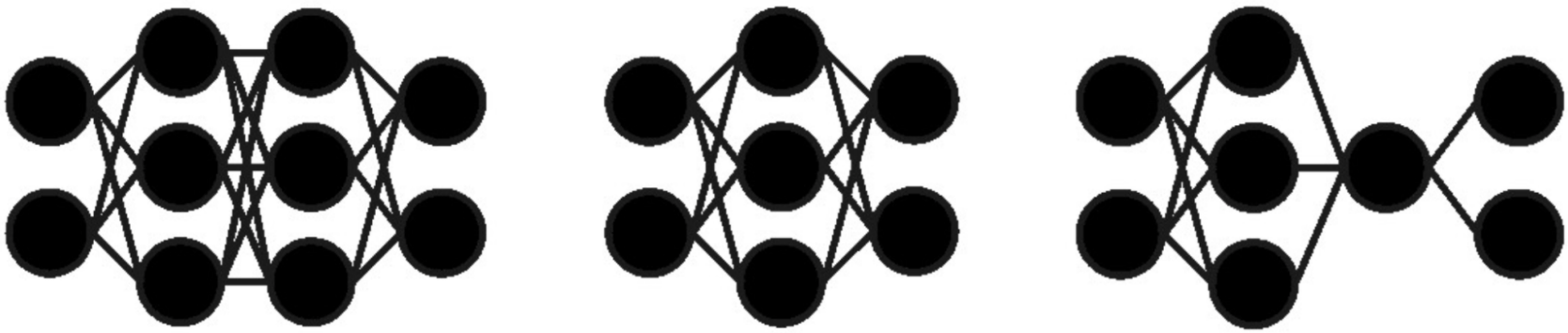
- So patterns are trained (embedded) into an NN...
- We have to specify the topology



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

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

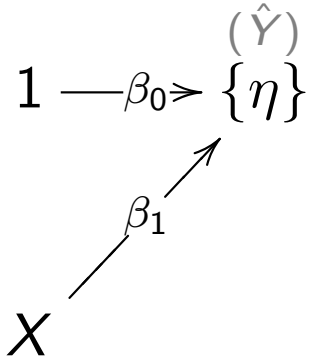
- So patterns are trained (embedded) into an NN...
- We have to specify the topology

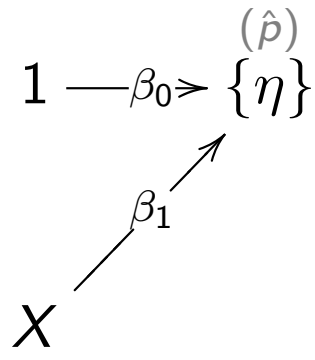


- Training patterns into a *Deep NN (DNN)* is challenging...

Let's start simple...

Example NNs (Regression and Logistic Regression)





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

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

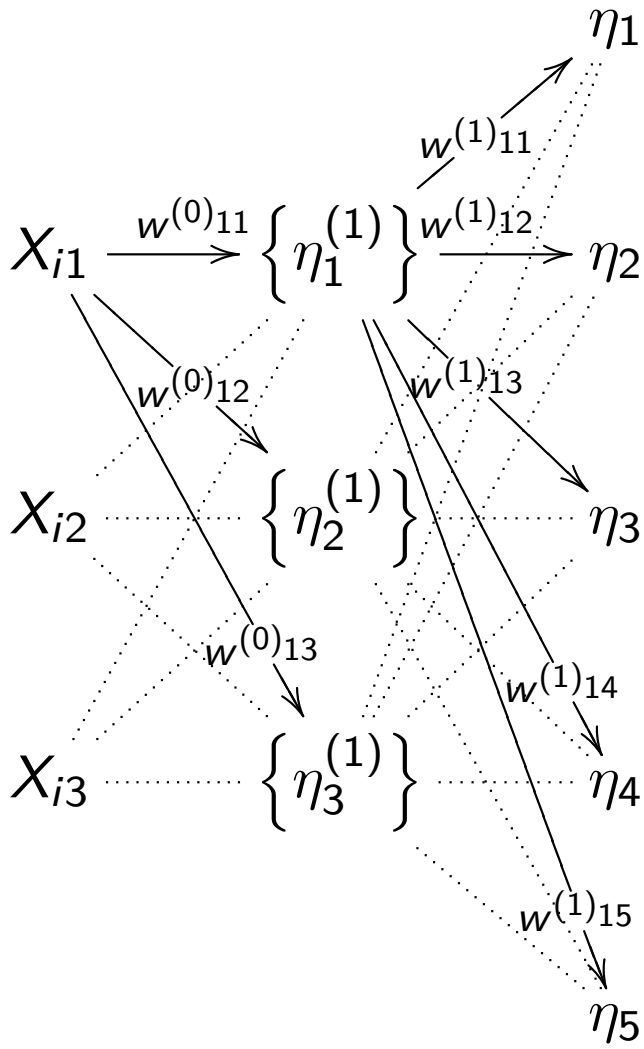
$$\min_{\hat{Y}} \frac{1}{2}\sum (Y_i - \hat{Y}_i)^2$$

$$\min_{\boldsymbol{p}} -\prod Y_i \log(p_i) + (1 - Y_i)\log(1 - p_i)$$

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

$$\hat{\beta} = \underset{\beta}{\operatorname{argmin}} -\prod Y_i \log\left(\frac{1}{1 + e^{-(\beta_0 + \beta_1 x_i)}}\right)$$
$$+ (1 - Y_i)\log\left(\frac{1}{1 + e^{\beta_0 + \beta_1 x_i}}\right)$$

Softmax



Choose the class with the largest probability

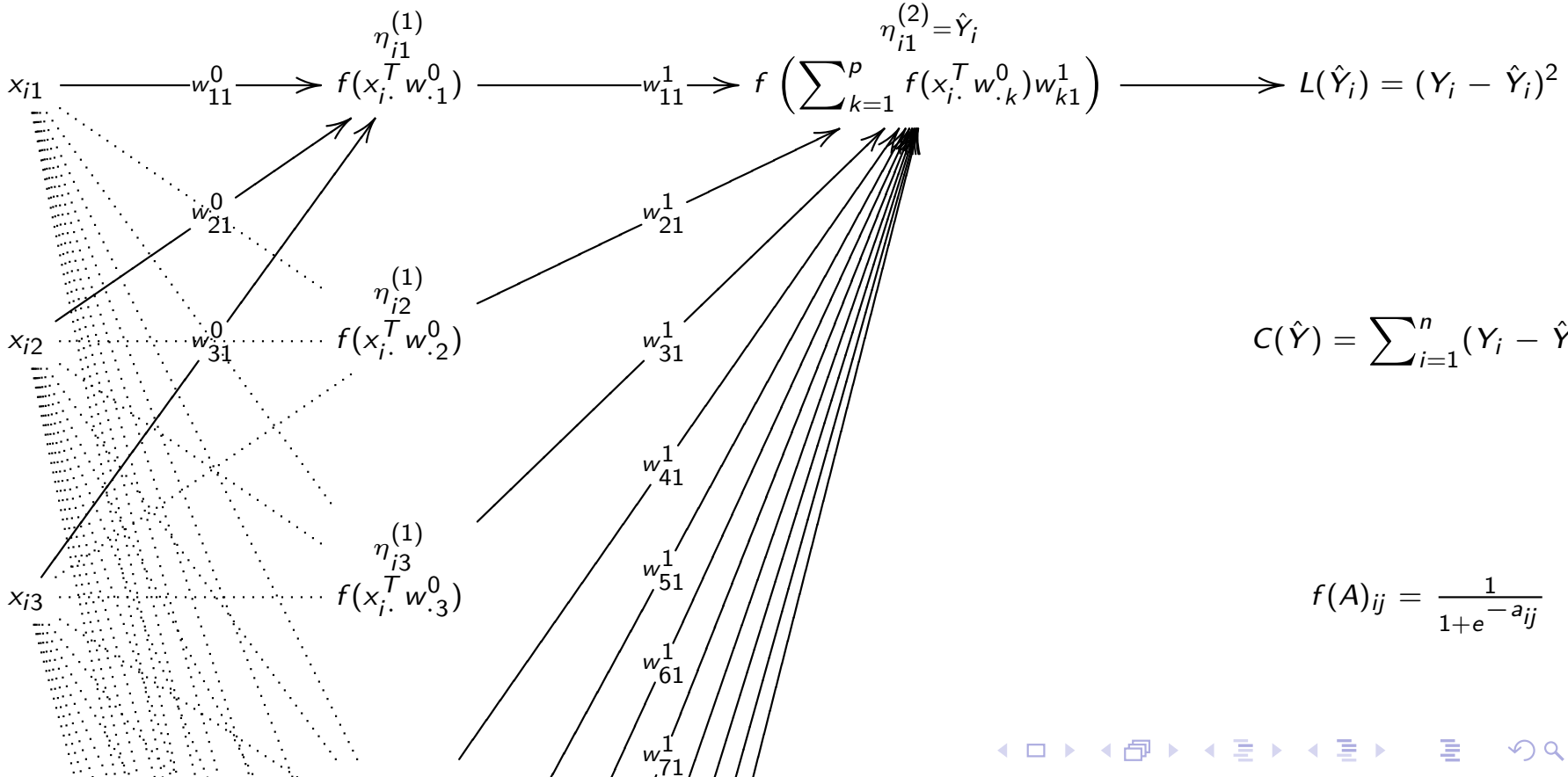
$$exp(\eta_k)/\sum_{k'=1}^{K} exp(\eta_{k'})$$

Define a loss function on those predictions

And Optimize!

Forward propagation [And how to update parameters?]

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



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

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

Backpropagation (Chain Rule Extraordinaire)

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

$$\frac{\partial}{\partial w^1_{k1}} L(\hat{Y}_i) = \frac{\partial \hat{Y}_i}{\partial w^1_{k1}} \frac{\partial}{\partial \hat{Y}_i} L(\hat{Y}_i)$$
$$= \eta^1_{ik} \hat{Y}_i (1 - \hat{Y}_i) 2(Y_i - \hat{Y}_i)$$

$$\frac{\partial}{\partial w^0_{jk}} L(\hat{Y}_i) = \frac{\partial \hat{Y}_i}{\partial w^0_{jk}} \frac{\partial}{\partial \hat{Y}_i} L(\hat{Y}_i)$$
$$= \frac{\partial \eta^1_{ik}}{\partial w^0_{jk}} \frac{\partial \hat{Y}_i}{\partial \eta^1_{ik}} \frac{\partial}{\partial \hat{Y}_i} L(\hat{Y}_i)$$
$$= x_{ij} \eta^1_{ik} (1 - \eta^1_{ik}) w^1_{k1} \hat{Y}_i (1 - \hat{Y}_i) 2(Y_i - \hat{Y}_i)$$

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

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

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

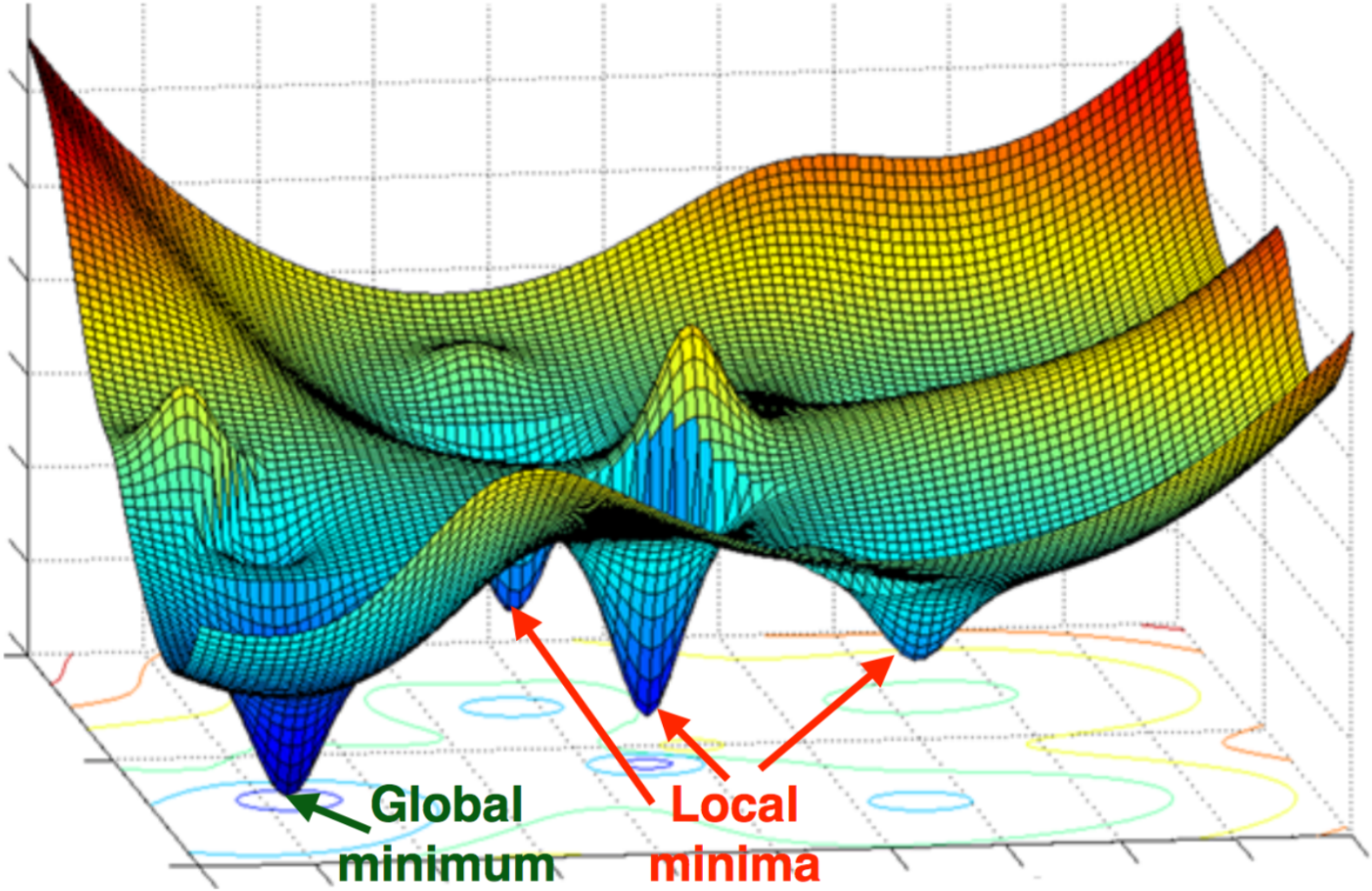
$$\eta^1_{ik} = f(x_{i\cdot}^T w^0_{\cdot k})$$

$$\hat{Y}_i = f\left(\sum_{k=1}^{p} f(x_{i\cdot}^T w^0_{\cdot k}) w^1_{k1}\right)$$

$$L(\hat{Y}_i) = (Y_i - \hat{Y}_i)^2$$

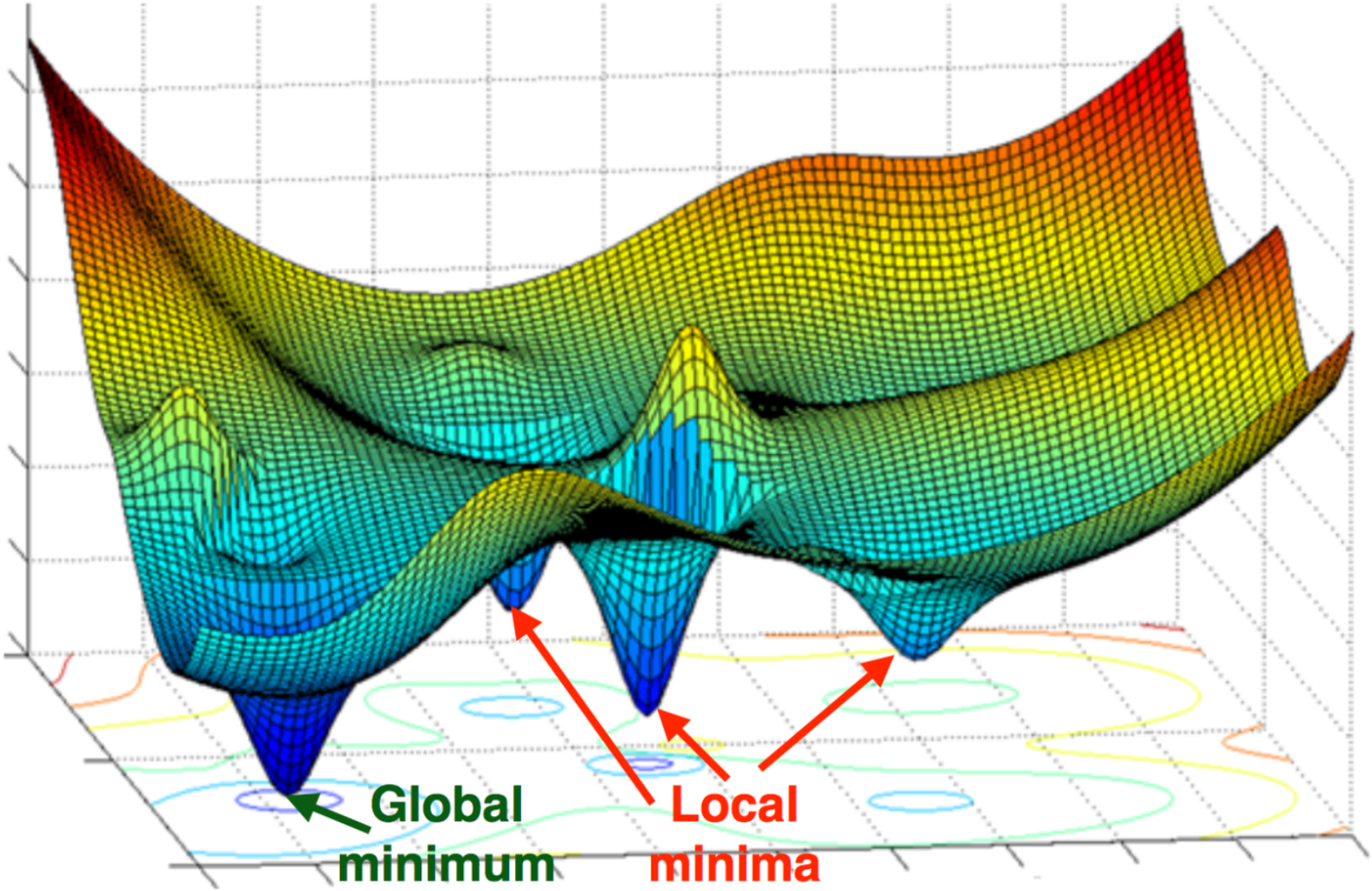
$$C(\hat{Y}) = \sum_{i=1}^{n} (Y_i - \hat{Y}_i)^2$$

Sadly, backpropagation (GD) is very weak on Deep NN's



Sadly, backpropagation (GD) is very weak on Deep NN's

Because there's so, many w's... and thus so many local minima...



Sadly, backpropagation (GD) is very weak on Deep NN's

- But if you're gonna, then consider stochastic or mini-batch

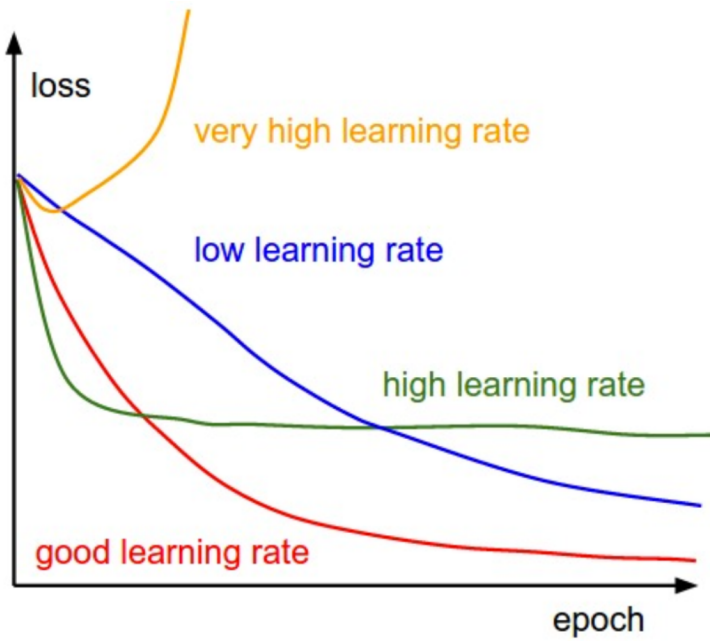
Sadly, backpropagation (GD) is very weak on Deep NN's

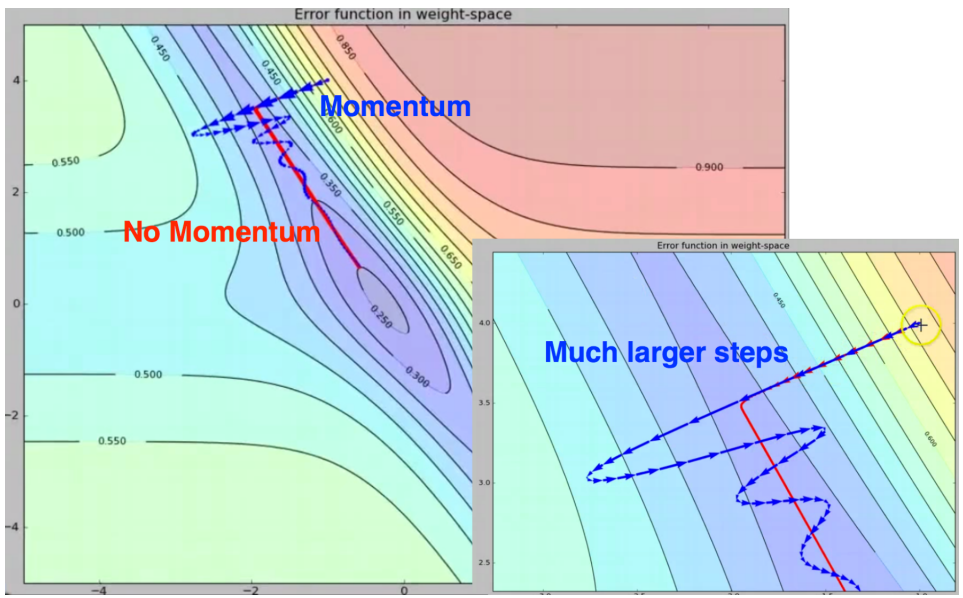
- But if you're gonna, then consider stochastic or mini-batch
- You'll be doing many *epochs* (iterations through the data)

Sadly, backpropagation (GD) is very weak on Deep NN's

- But if you're gonna, then consider stochastic or mini-batch
- You'll be doing many *epochs* (iterations through the data)
- And you might wanna explore using momentum and velocity

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





Sadly, backpropagation (GD) is very weak on Deep NN's

- Not much consensus on designs – lots of trial and error...

Sadly, backpropagation (GD) is very weak on Deep NN's

- Not much consensus on designs – lots of trial and error...
- but usually less that 3 hidden layers

Sadly, backpropagation (GD) is very weak on Deep NN's

- Not much consensus on designs – lots of trial and error...
- but usually less that 3 hidden layers
- fewer outputs than inputs per layer

Sadly, backpropagation (GD) is very weak on Deep NN's

- Not much consensus on designs – lots of trial and error...
- but usually less that 3 hidden layers
- fewer outputs than inputs per layer
- 2*p neurons (noisy data) to 30*p neurons (perfect signal)

Sadly, backpropagation (GD) is very weak on Deep NN's

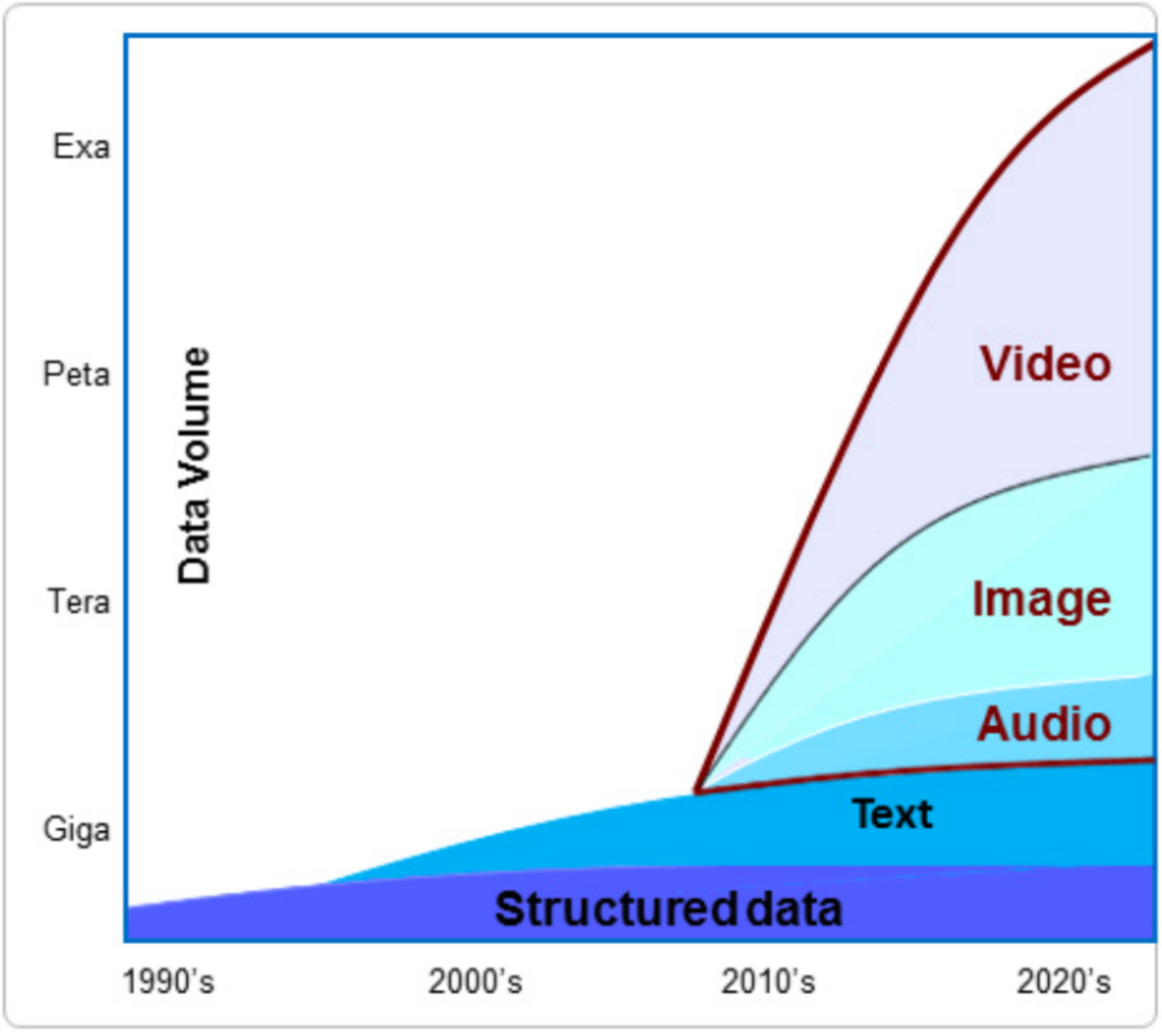
- Not much consensus on designs – lots of trial and error...
- but usually less that 3 hidden layers
- fewer outputs than inputs per layer
- 2*p neurons (noisy data) to 30*p neurons (perfect signal)
- probably rectified linear, perhaps tanh & *maybe* sigmoid AF's

Sadly, backpropagation (GD) is very weak on Deep NN's

- Not much consensus on designs – lots of trial and error...
- but usually less that 3 hidden layers
- fewer outputs than inputs per layer
- 2*p neurons (noisy data) to 30*p neurons (perfect signal)
- probably rectified linear, perhaps tanh & *maybe* sigmoid AF's

Here's what you'll actually do:
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

The new type of 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

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

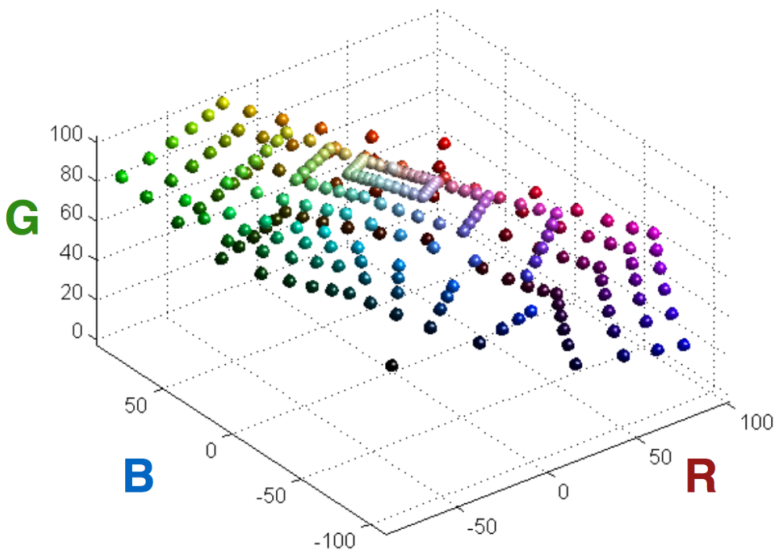
image.shape = (width, height,3)

What data/information is contained in an image?

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

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

image.shape = (width, height,3)



What data/information is contained in an image?

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

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

image.shape = (width, height,3)

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

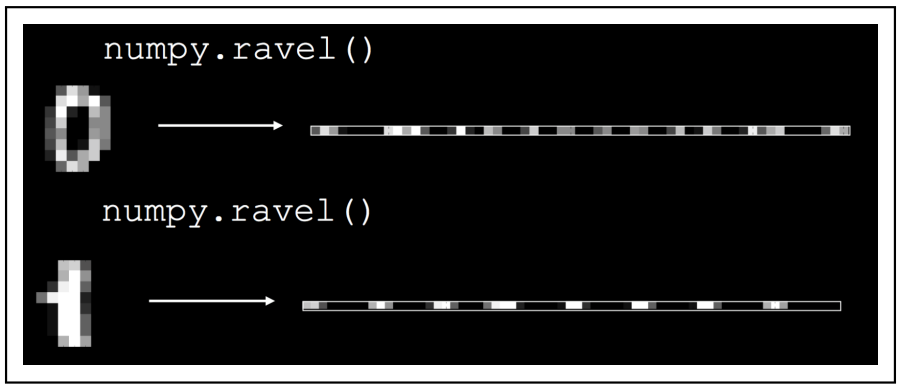
What data/information is contained in an image?

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

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

image.shape = (width, height,3)

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



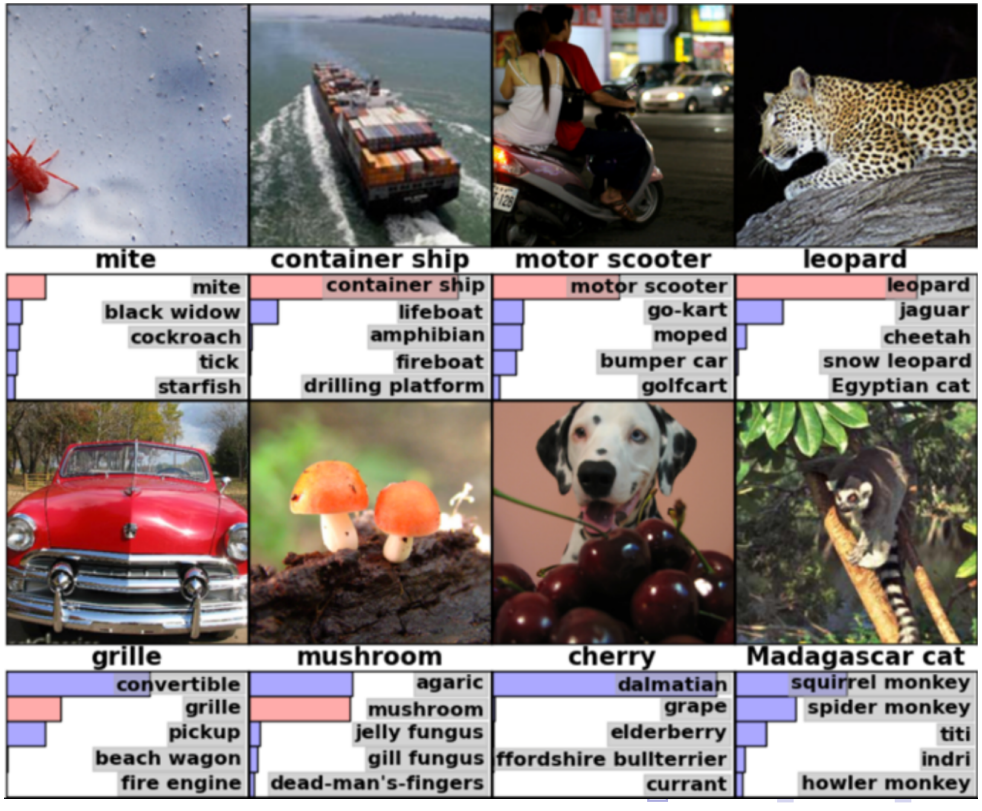
featurized.shape = (width·height·3,1)

What data/information is contained in an image?

$$\begin{array}{c} \\ \text{image 1} \\ \text{image 2} \\ \vdots \\ \text{image n} \end{array} \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1p} \\ x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{np} \end{bmatrix} \xRightarrow{\text{Predict}} \begin{array}{c} \text{Image Label} \\ \begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_n \end{bmatrix} \end{array}$$

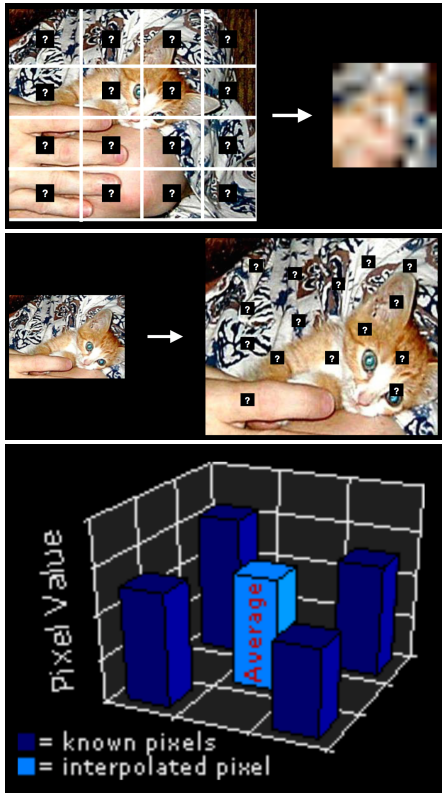
What data/information is contained in an image?

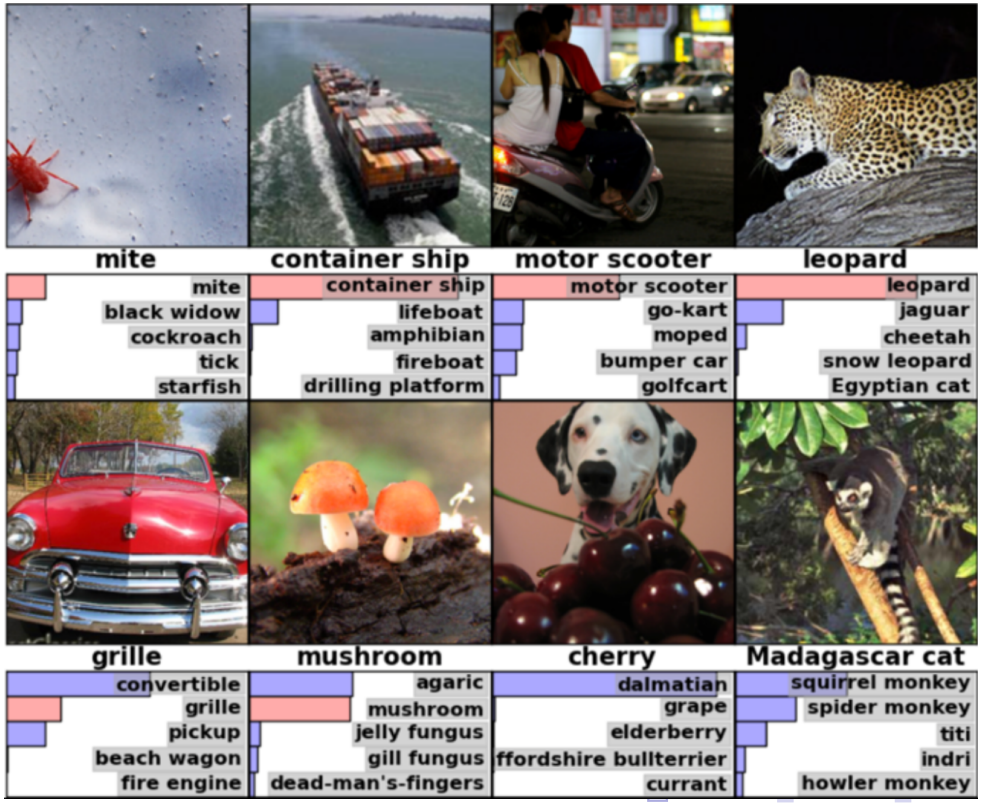
$$\begin{array}{c} \\ \text{image 1} \\ \text{image 2} \\ \vdots \\ \text{image n} \end{array} \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1p} \\ x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{np} \end{bmatrix} \xrightarrow{\text{Predict}} \begin{array}{c}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



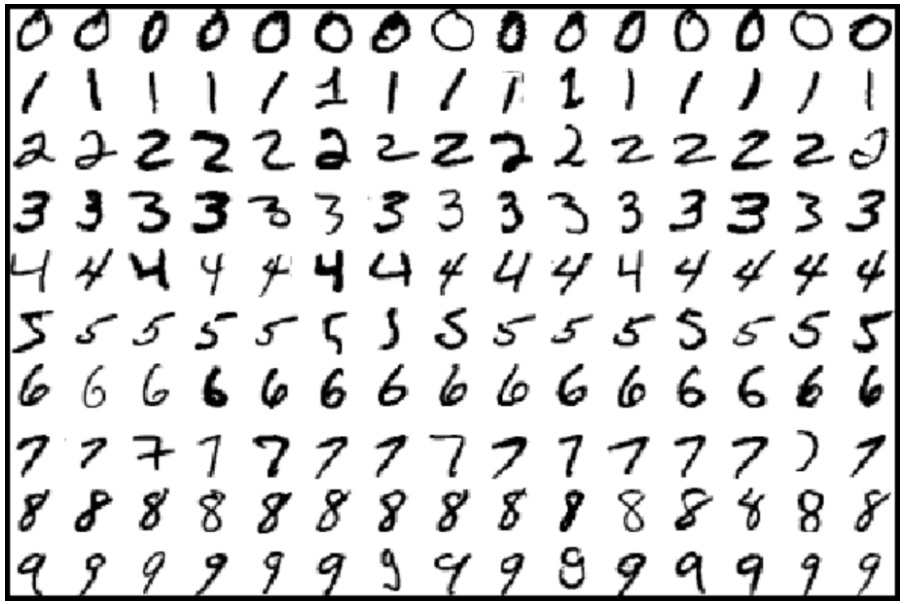
What data/information is contained in an image?

$$\begin{array}{c} \\ \text{image 1} \\ \text{image 2} \\ \vdots \\ \text{image n} \end{array} \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1p} \\ x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{np} \end{bmatrix} \xRightarrow{\text{Predict}} \begin{array}{c} \text{Image Label} \\ \begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_n \end{bmatrix} \end{array}$$





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

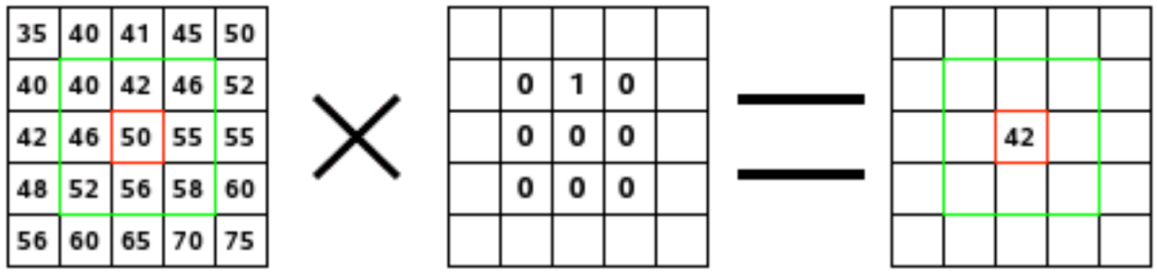
Image Processing

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

$$x_i \to y_i : \underset{\boldsymbol{y}}{\operatorname{argmin}} \quad \frac{1}{2}\underbrace{\sum (x_i - y_i)^2}_{\textit{Fidelity}} + \lambda \underbrace{\sum |y_{i+1} - y_i|}_{\textit{Variation}}$$

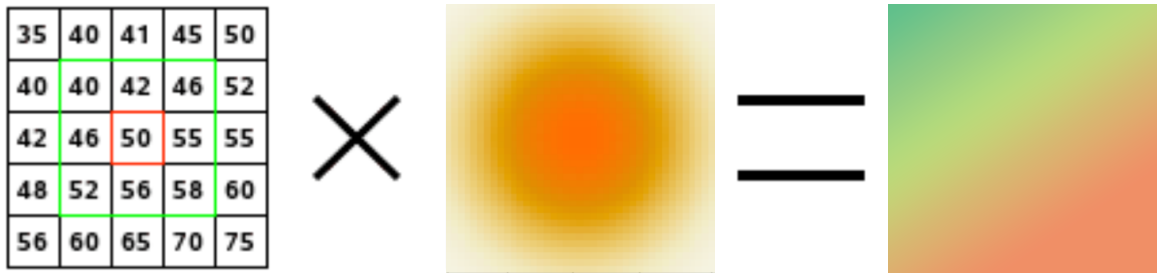
Convolutional Kernel

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



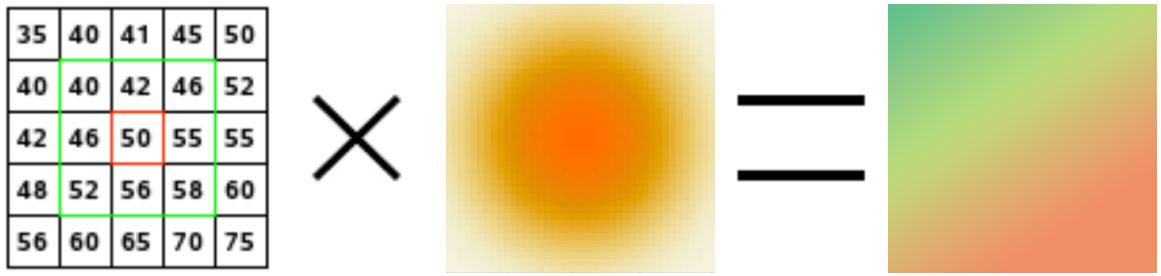
Convolutional Kernel

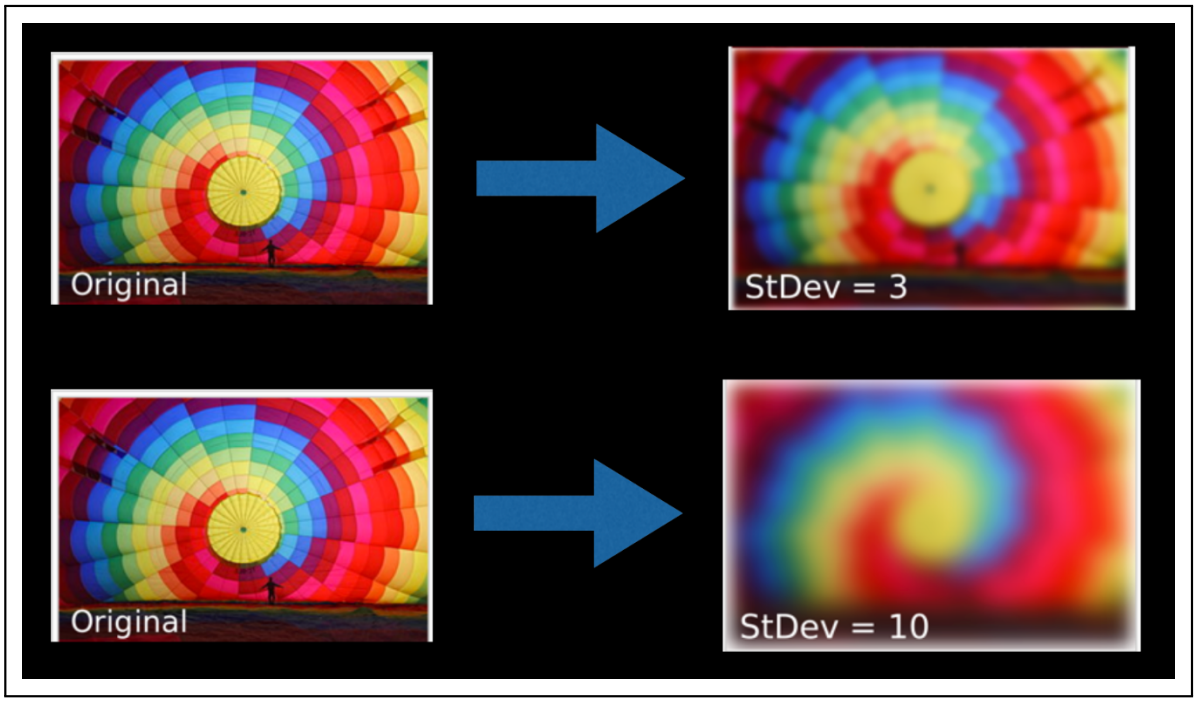
- Spatial proximity *and pixel agreement (bi-lateral)*



Convolutional Kernel

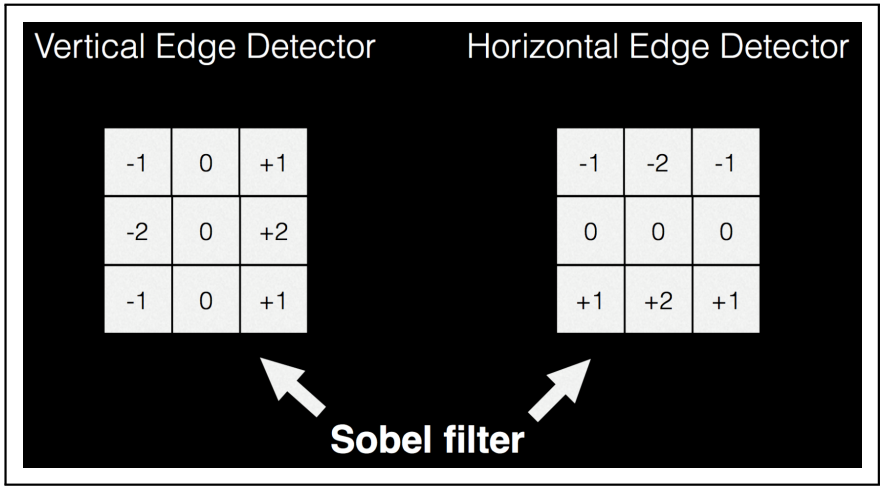
- Spatial proximity *and pixel agreement (bi-lateral)*

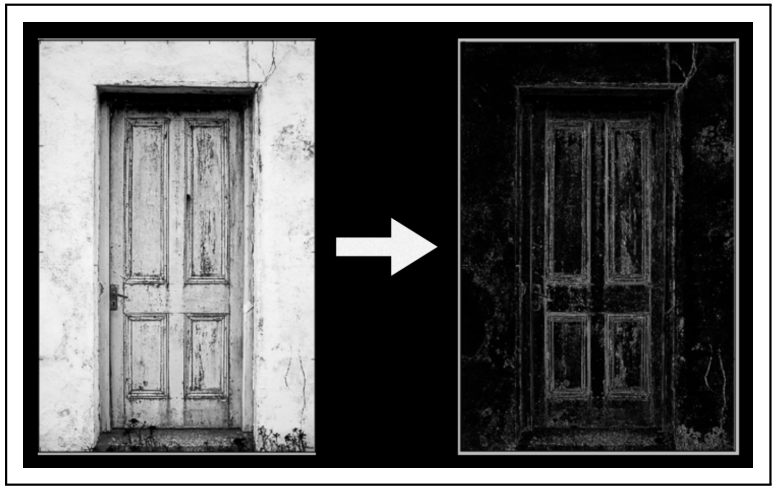




Convolutional Kernel

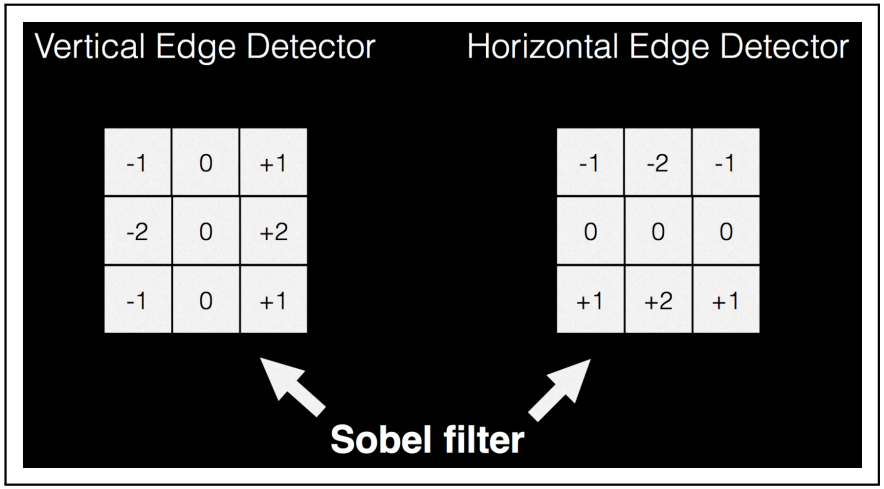
It may be easier to identify objects if we just consider the edges

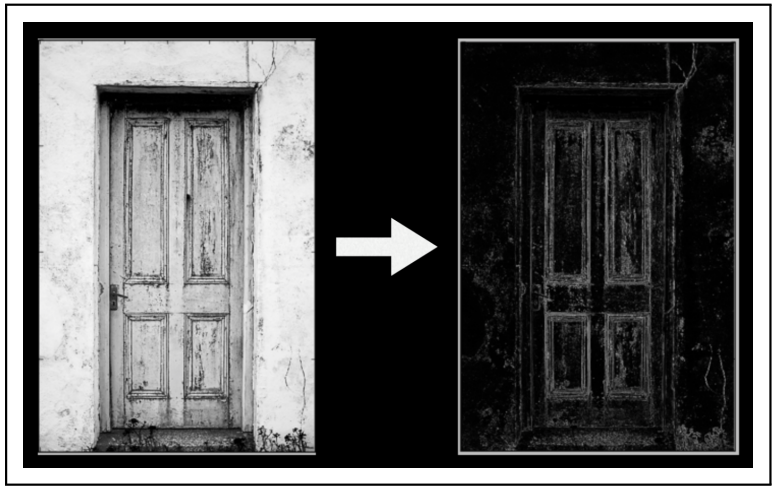




Convolutional Kernel

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





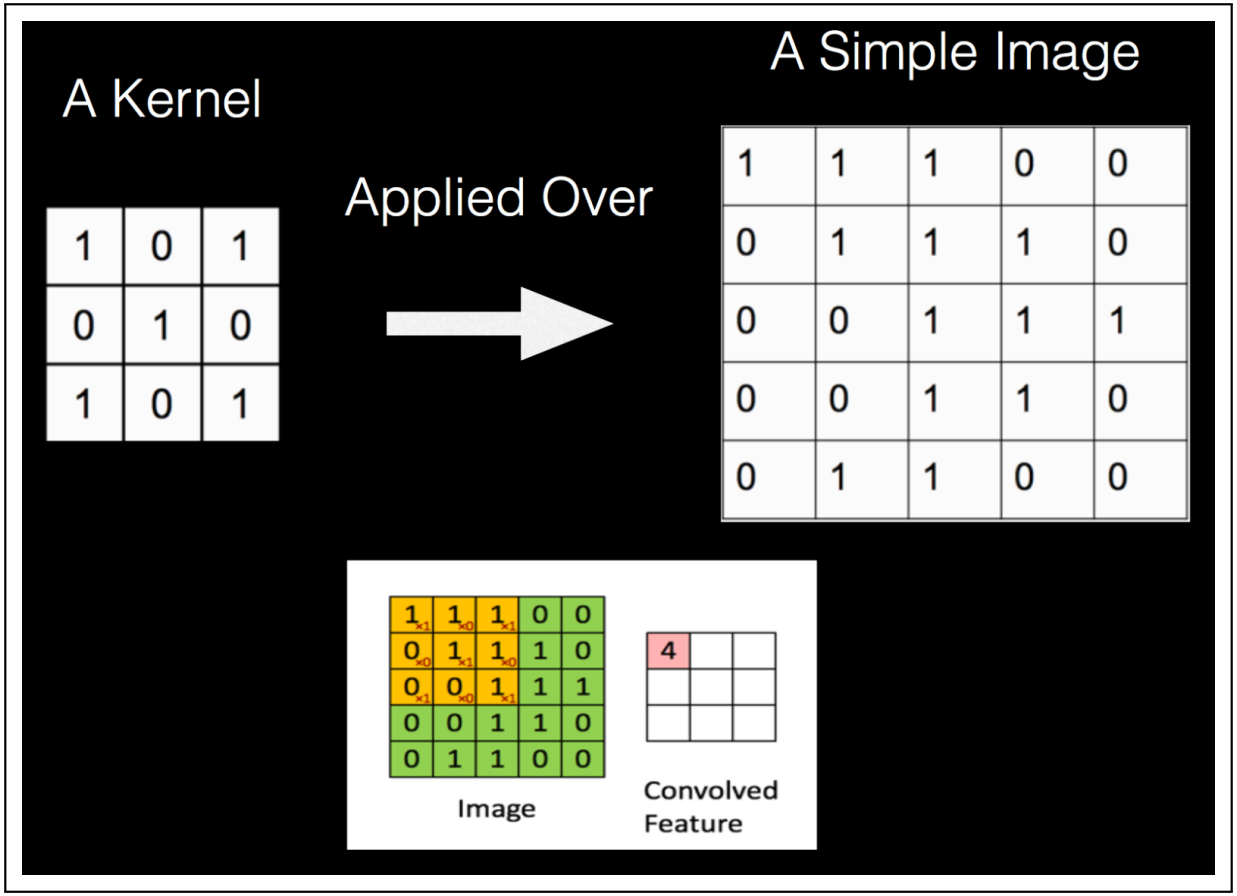
Canny Filter

Sobel Filter

Convolutional Kernel

Convolutional Kernel

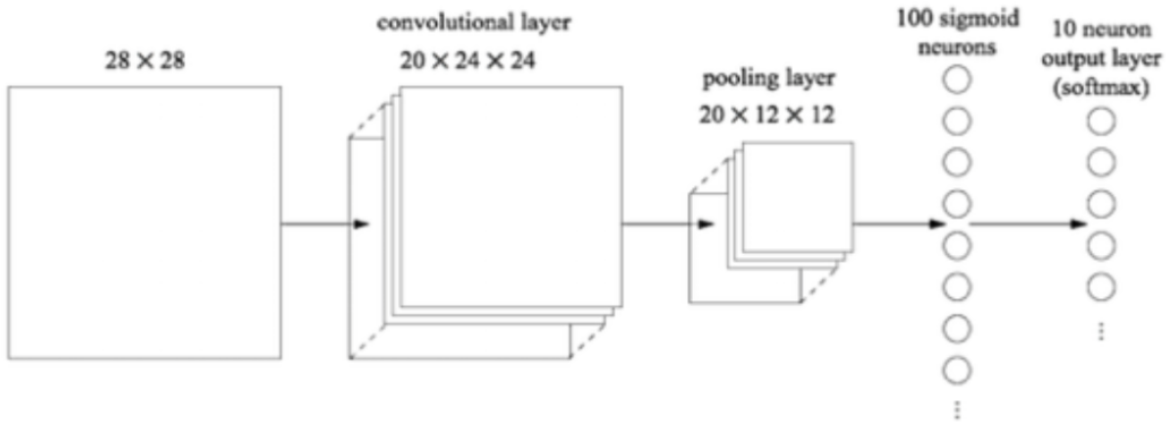
Recall that a convolution matrix (or kernel or mask) in image processing is a small matrix that can be used to blur, sharpen, emboss, detect edges, etc. through *convolutions* with an image



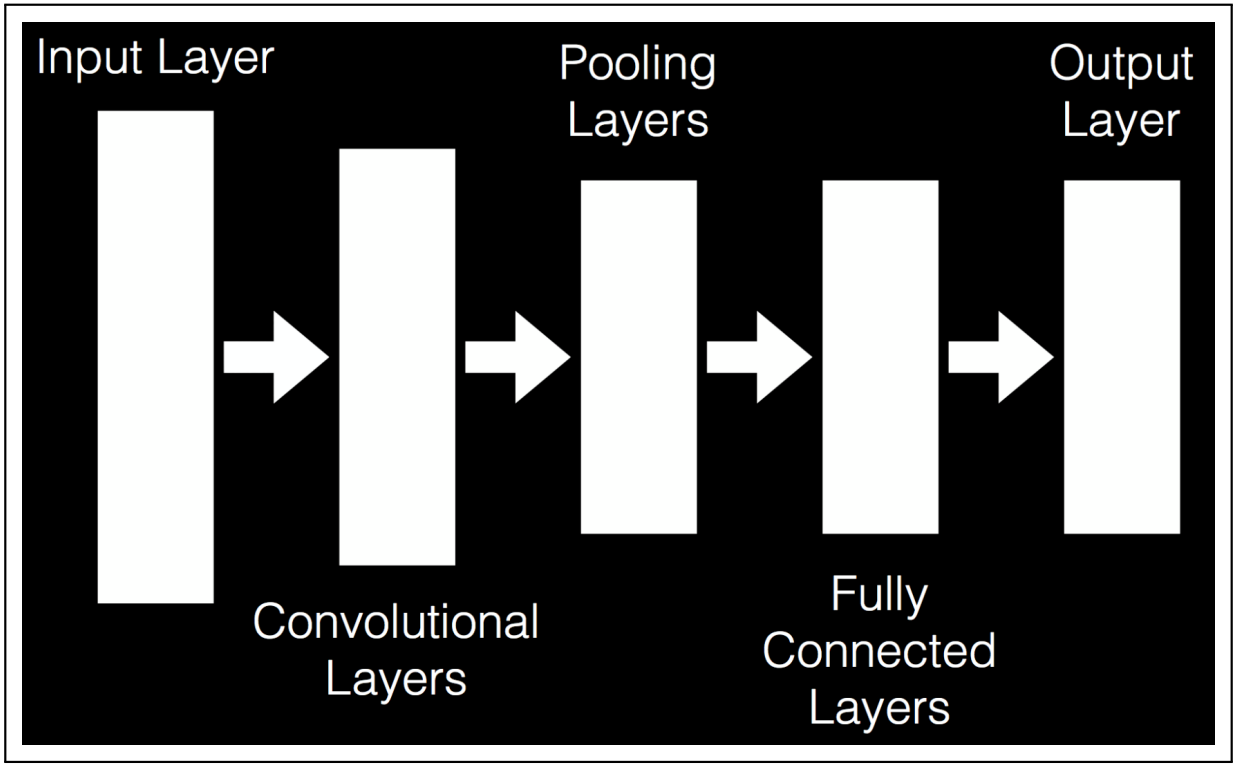
Can NN's learn their own convolution matrices?

idea: *denovo* kernels

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

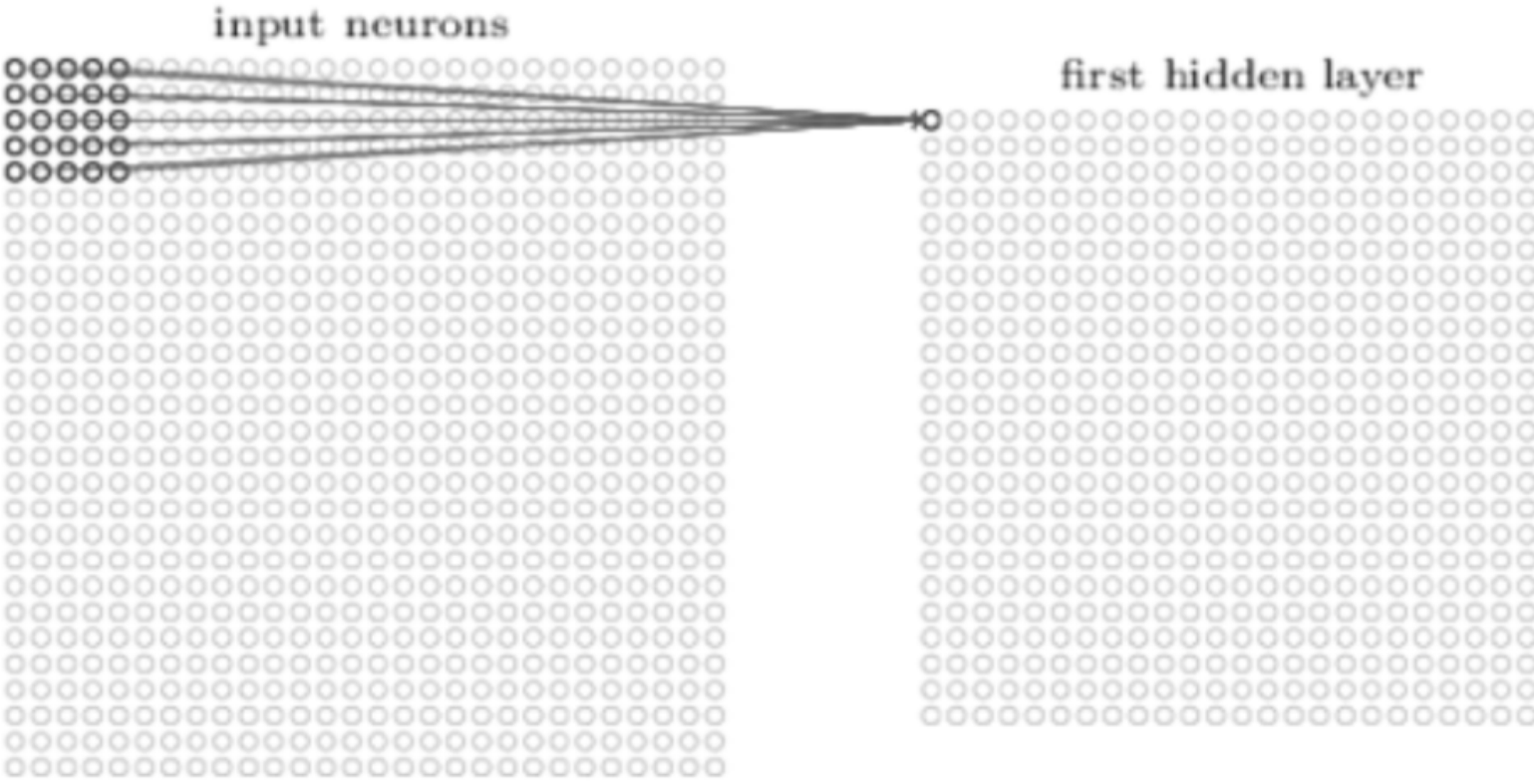


Convolutional Neural Network (CNN)



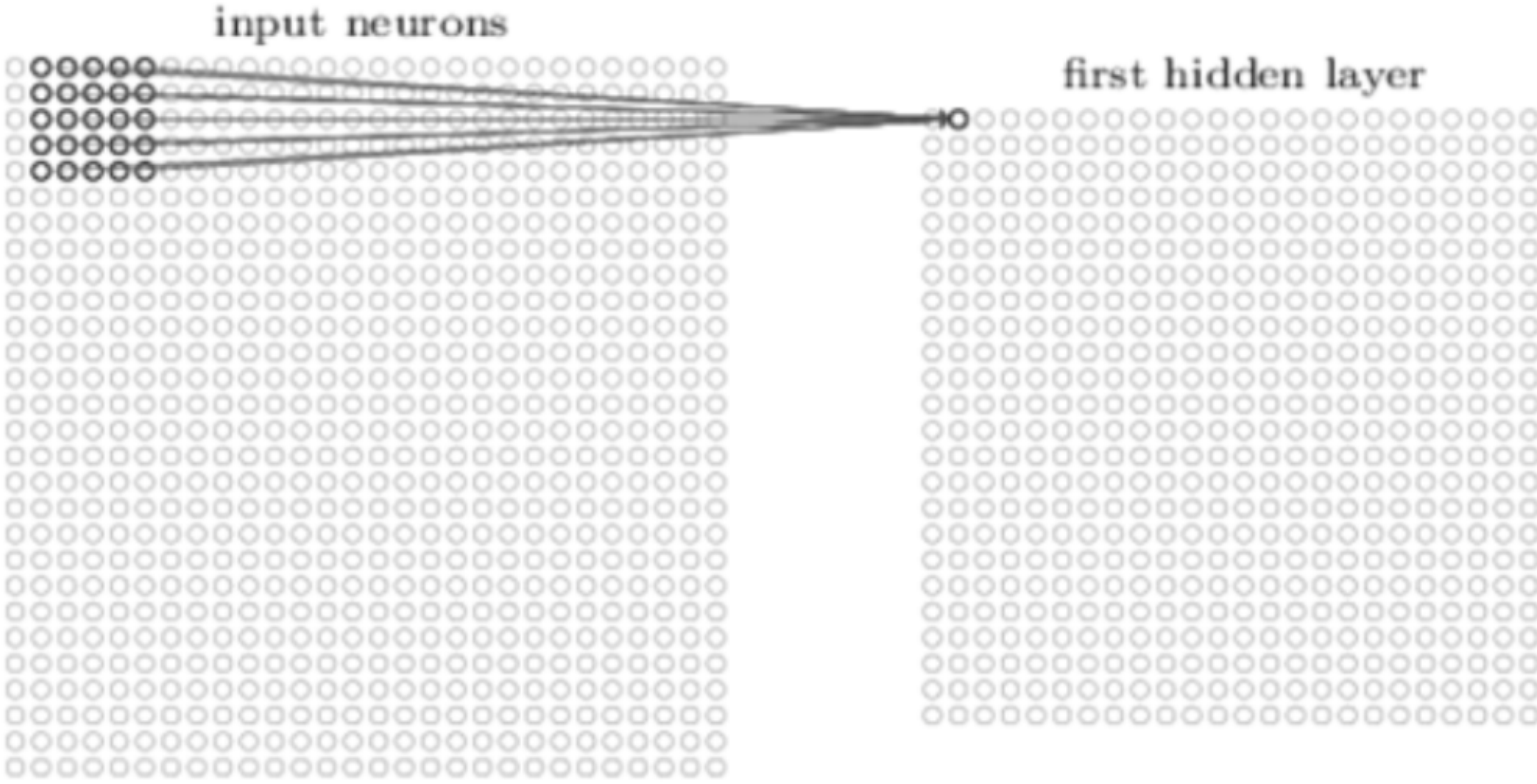
There are three key features that make this CNN structure actually work: (1) local receptive fields, (2) shared weights, and (3) pooling

1. Local Receptive Field



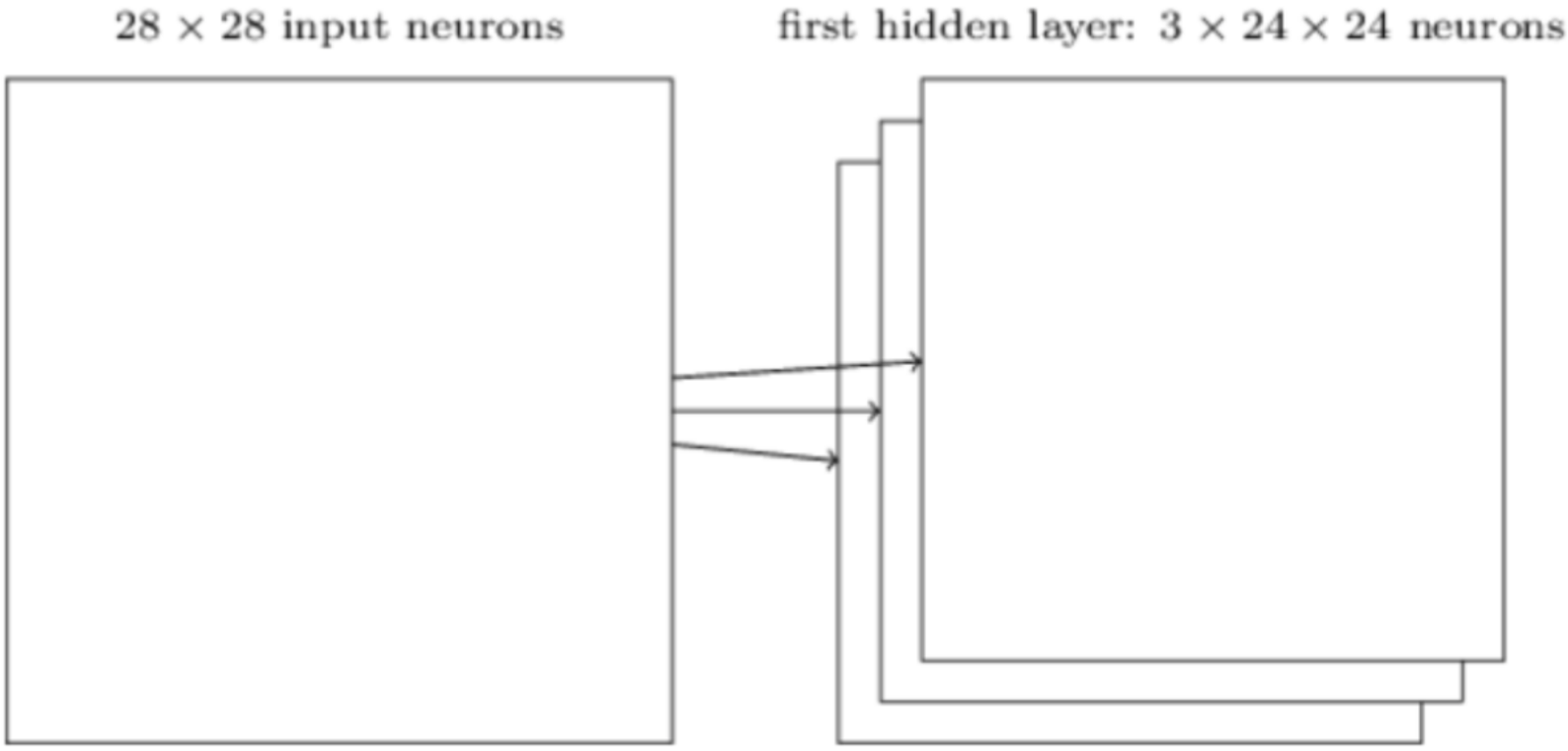
- A group of pixes are a *local receptive field*
- Defined by the size of the kernel
- The image is transformed into the set of local receptive fields

1. Local Receptive Field



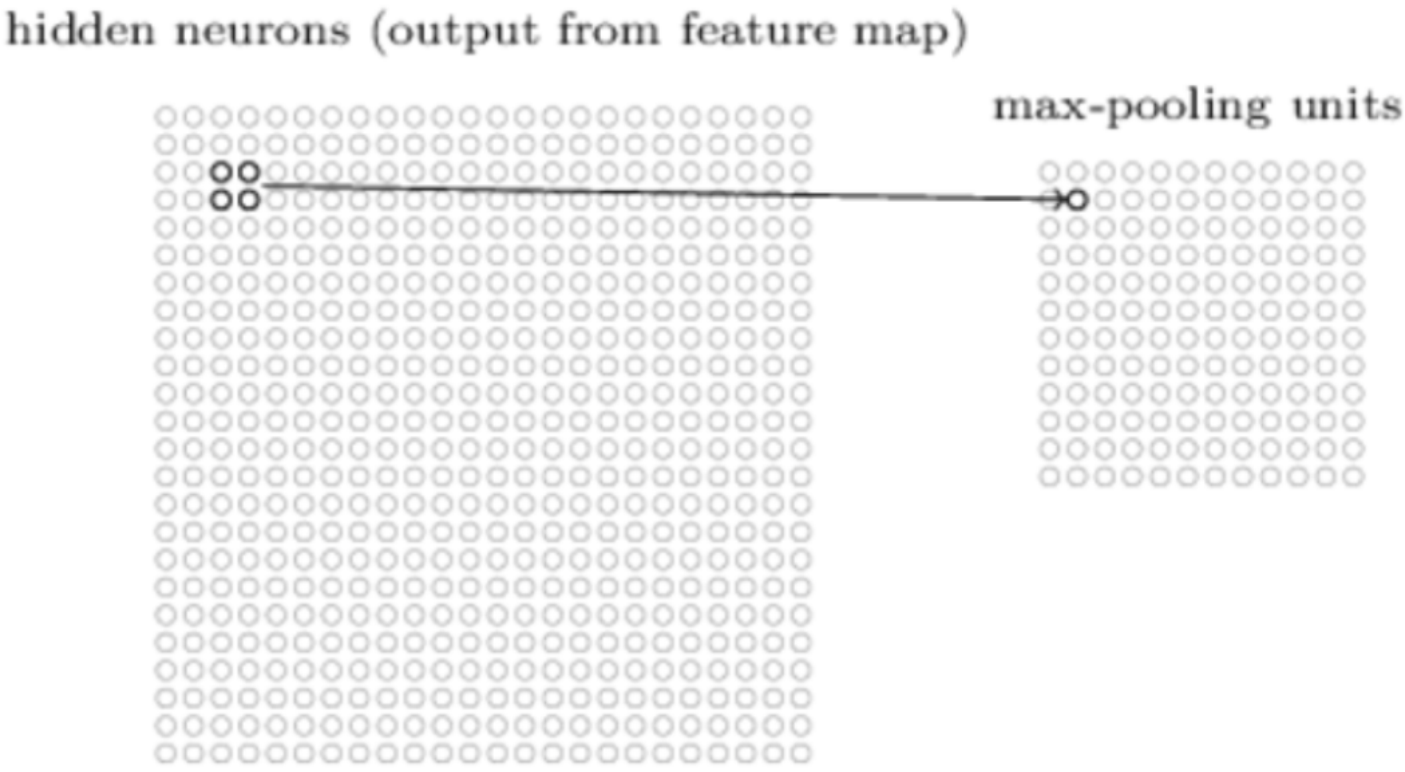
- A group of pixes are a *local receptive field*
- Defined by the size of the kernel
- The image is transformed into the set of local receptive fields

2. Shared Weights



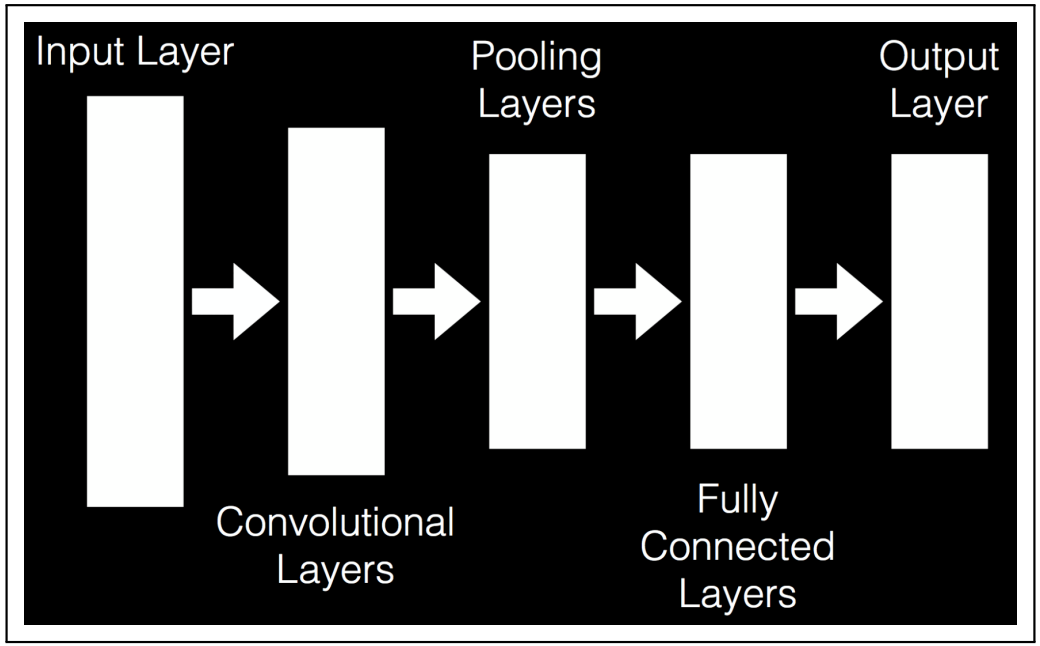
- Multiple convolutions are learned/used
- Weights within a convolution are (obviously) shared

3. Pooling



- 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

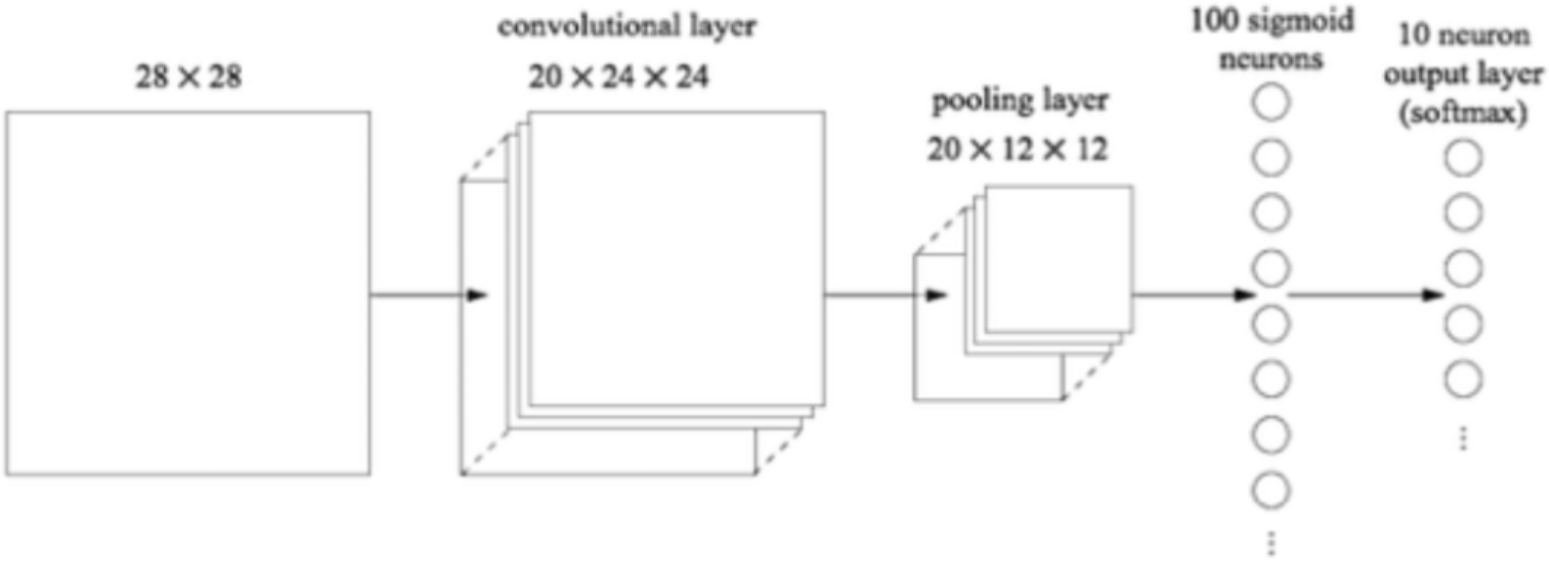
Fully Connected and Output Layers



- Fully connected layers hierarchically aggregate learned features – they produces higher order features in standard NN manner
- Softmax can be applied to the output layer $\eta_k, k = 1, \cdots K$ to estimate the one-versus-all class probabilities of K classes:

$$\frac{e^{\eta_k}}{\sum_{k'=1}^{K} e^{\eta_{k'}}}$$

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
[Provides introductions in NumPy, Theano, Tensorflow, & Keras]
neuralnetworksanddeeplearning.com [General introduction]
cs231n.github.io/ [Convolution NN's for visual recognition]
deeplearning.net/tutorial/lenet.html [Mathematics]
arxiv.org/abs/1503.02531 [Interpretation detractions]
[Projects]
lasagne.readthedocs.org/en/latest/index.html
(also, see nolearn for an scikit learn type of interface)
cbinsignts.com/blog/python-tools-machine-learning
deeplearning.net/tutorial/lenet.html
deeplearning4j.org
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*...