Neural Nets

Schwartz

September 21, 2016

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Upon his death, the titular Reverend Thomas Bayes (1701–1761) of the renowned theorem left an unpublished manuscript deriving the distribution for the parameter of a binomial distribution.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.

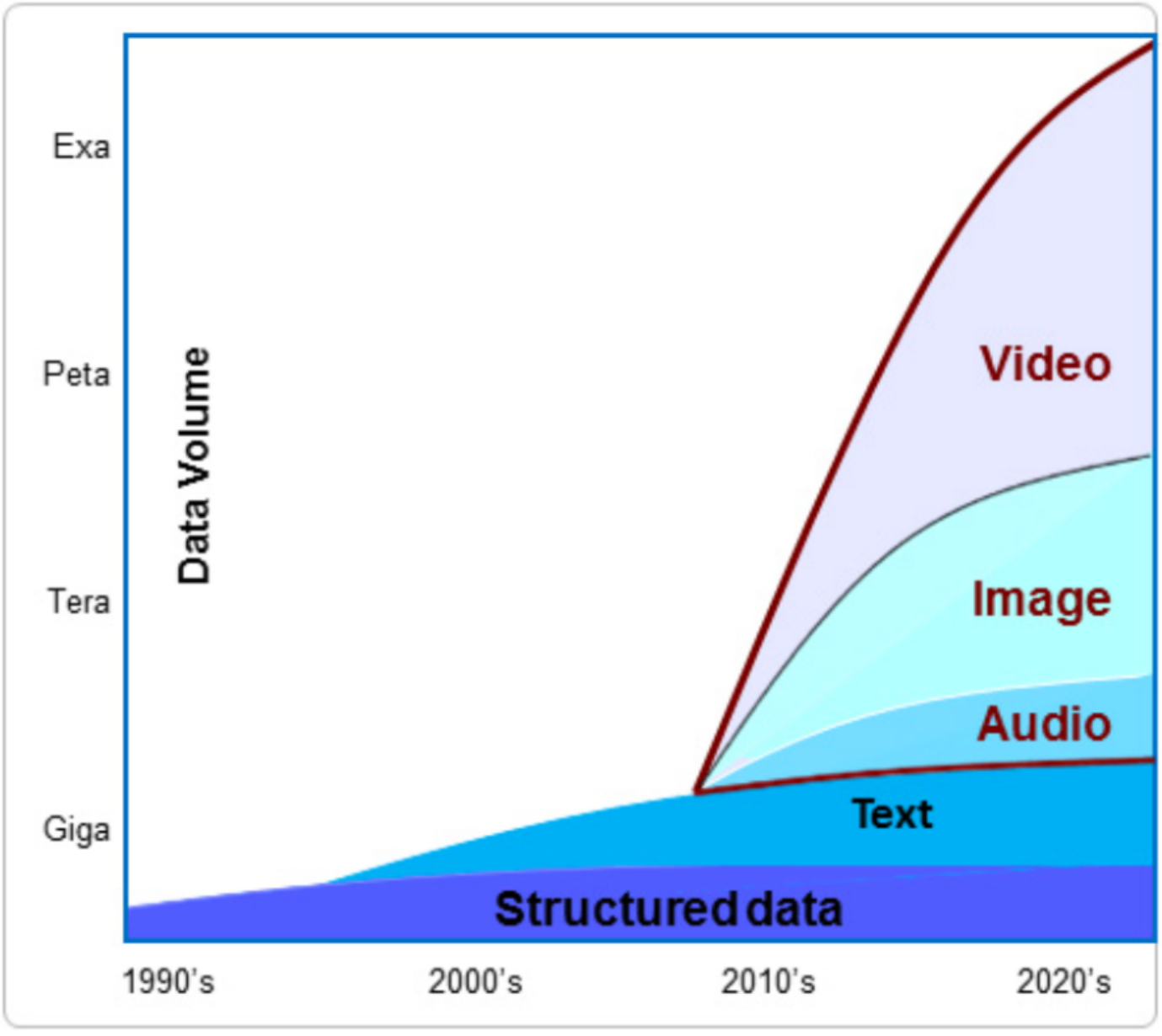
The elegant simplicity and effectiveness of Bayes' theorem for "learning" has prompted some psychologists to ask if the human brain itself might be a Bayesian-reasoning machine. They suggest that the Bayesian capacity to draw strong inferences from sparse data could be crucial to the way the mind perceives 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

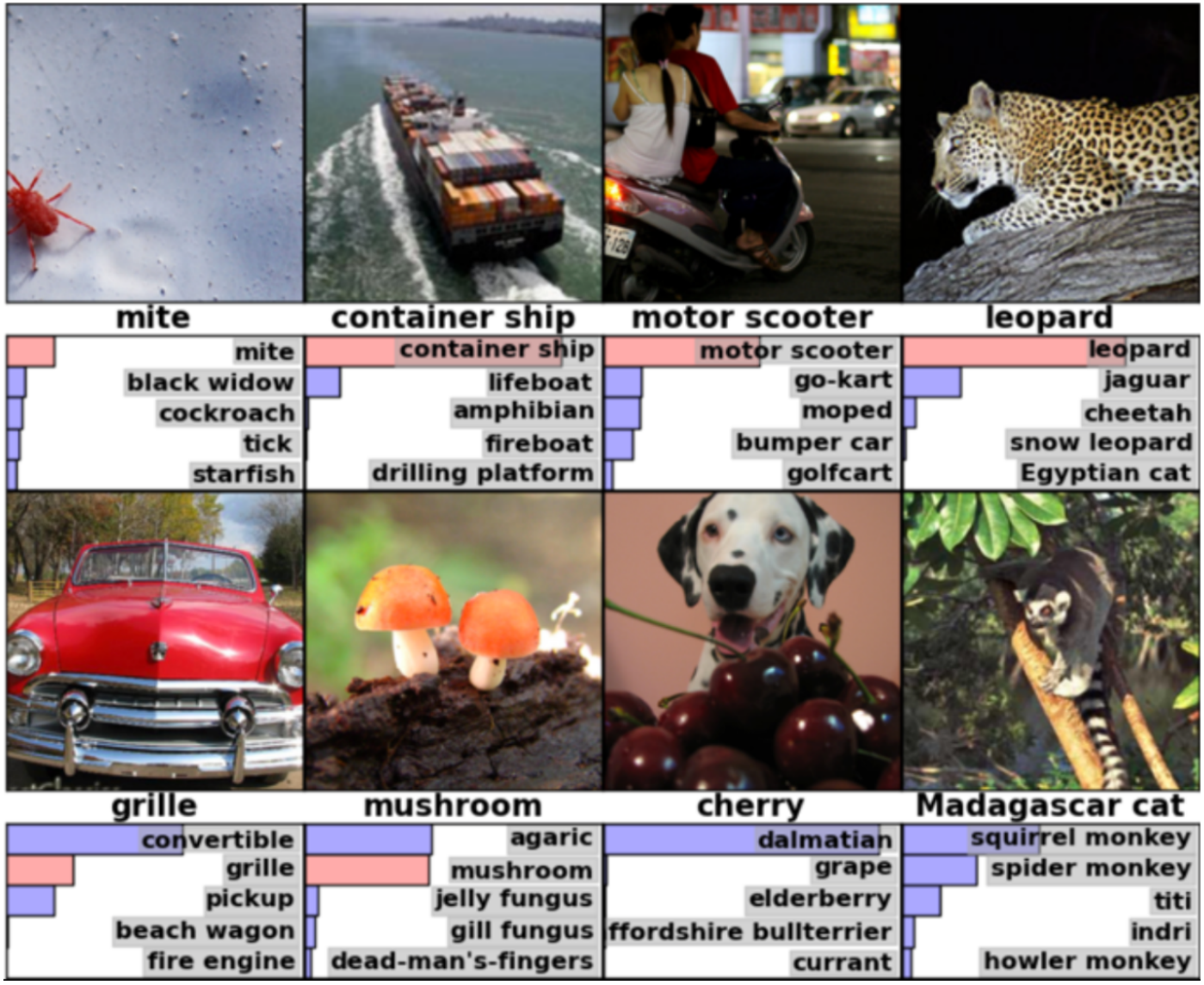
- understand image data
- understand image tools and pipelines
- understand convolutions
- understand neural networks
- understand how they can recapitulate other methodologies
- understand how activating functions provides power
- understand how layers provide abstract feature encoding
- understand how neural networks are trained, i.e., "backpropegation is gradient decent is the chain rule"
- understand that backpropegation doesn't always work well
- understand convolutional networks structure and features
- understand there is new thing called deep learning that can be viewed as an extension of neural networks

Well that's interesting...



What are images?

How can computers do this??



How *might* computers be able to do this?

How *might* computers be able to do this?

- Compress data
- Keep the search simple
- Segment image into objects

Image processing *tasks*

- Read

Image processing *tasks*

- Read

  ./dog_cat

  ./dog_cat/dog ./dog_cat/cat

Image processing *tasks*

- Read

  ./dog_cat

  ./dog_cat/dog ./dog_cat/cat

- image.shape = (width, height) or (width, height,3)

Image processing *tasks*

- Read
  ./dog_cat
  ./dog_cat/dog ./dog_cat/cat

- image.shape = (width, height) or (width, height,3)
  A *tensor* (e.g., a color image) is 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 processing *tasks*

- Resize

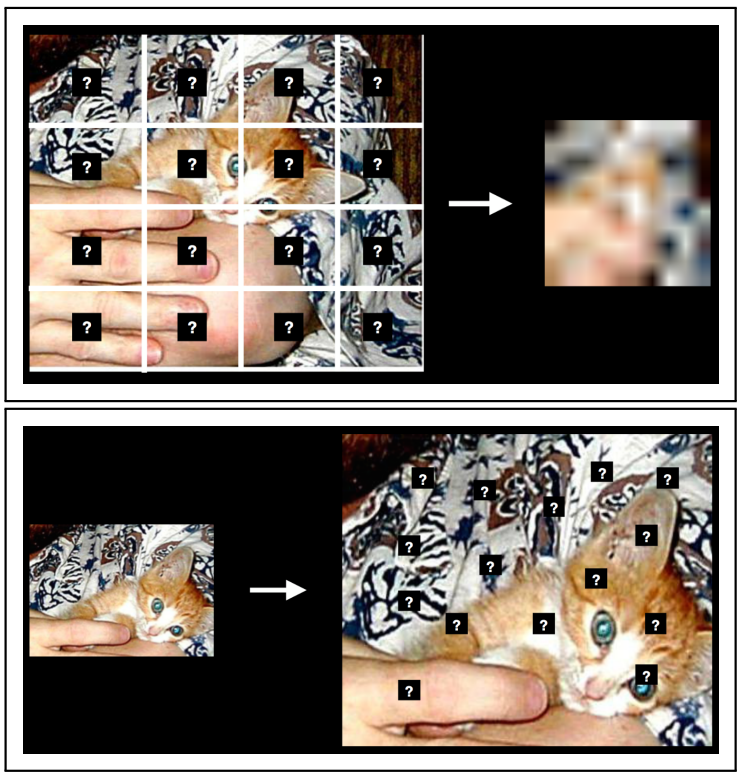
Image processing *tasks*

- Resize to create a uniform feature set

Image processing *tasks*

- Resize to create a uniform feature set

  Down/Upsampling – *not cropping* – bi-linear interpolation



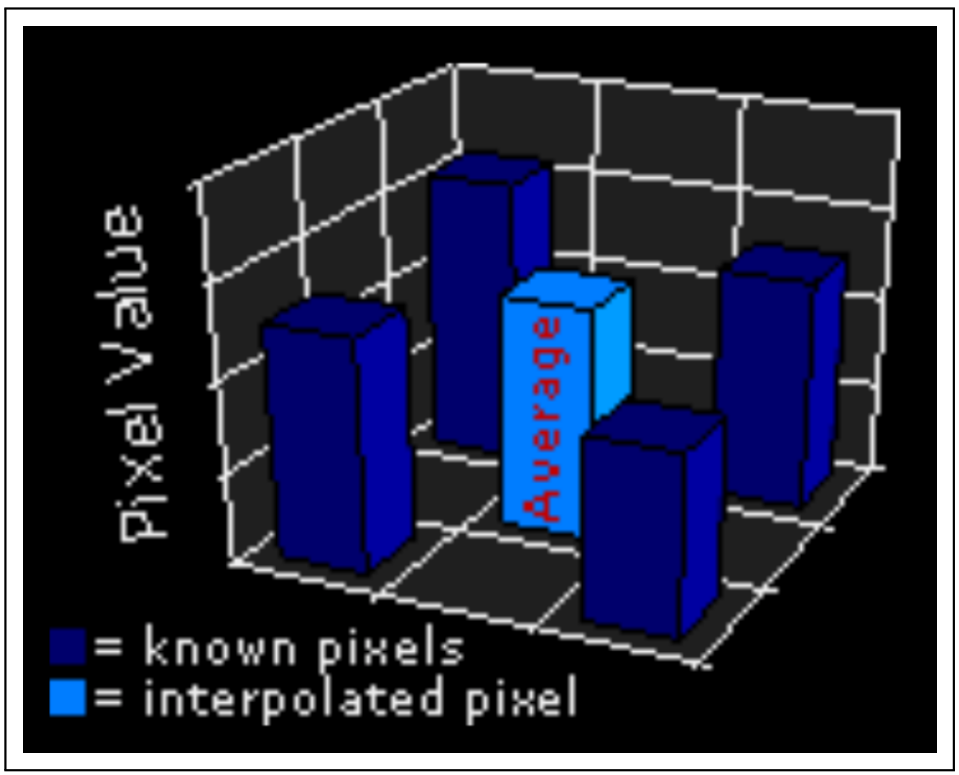
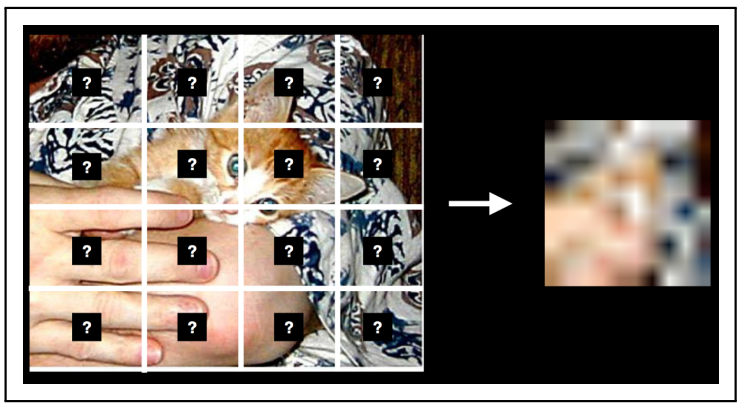
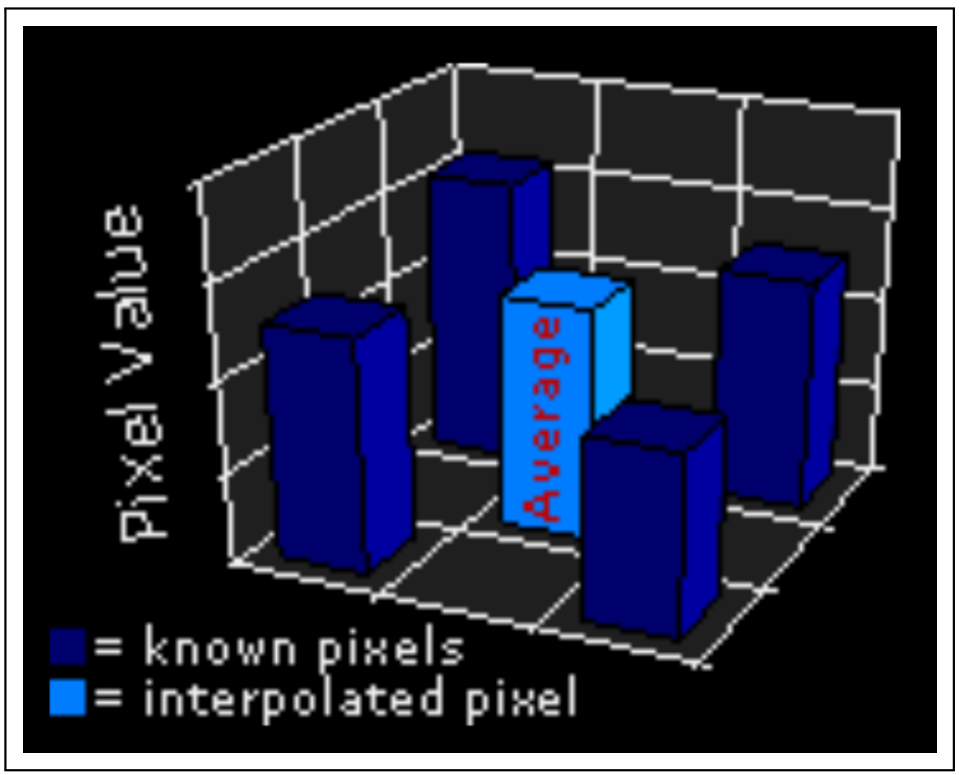


Image processing *tasks*

- Resize to create a uniform feature set

  Down/Upsampling – *not cropping* – bi-linear interpolation



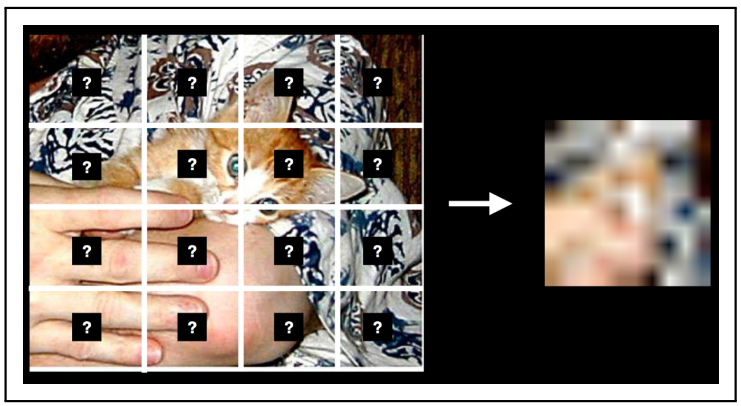


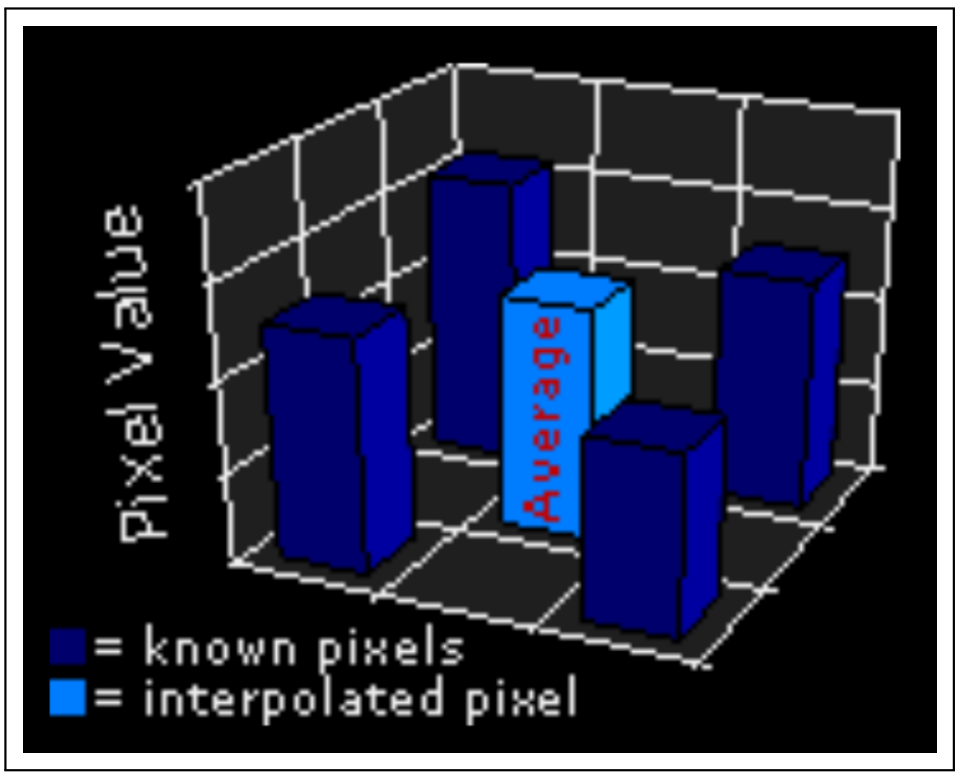
Reduce size/processing time

Image processing *tasks*

- Resize to create a uniform feature set

  Down/Upsampling – *not cropping* – bi-linear interpolation





Reduce size/processing time

Resolution visually checkable

Image processing *tasks*

- Denoise: remove unnecessary details to allow for & better generalization of images to image classes

Image processing *tasks* – more on denoising

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

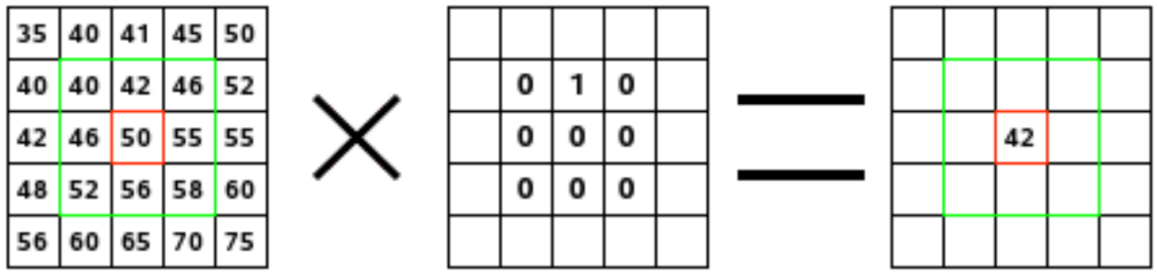
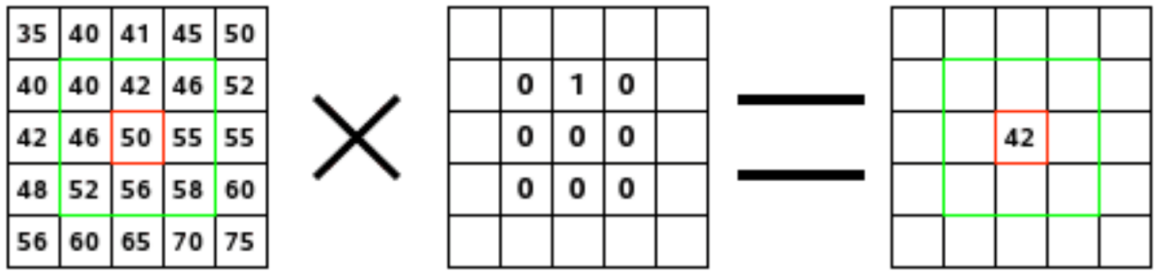


Image processing *tasks* – more on denoising

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



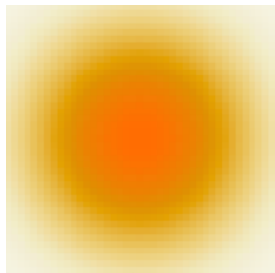
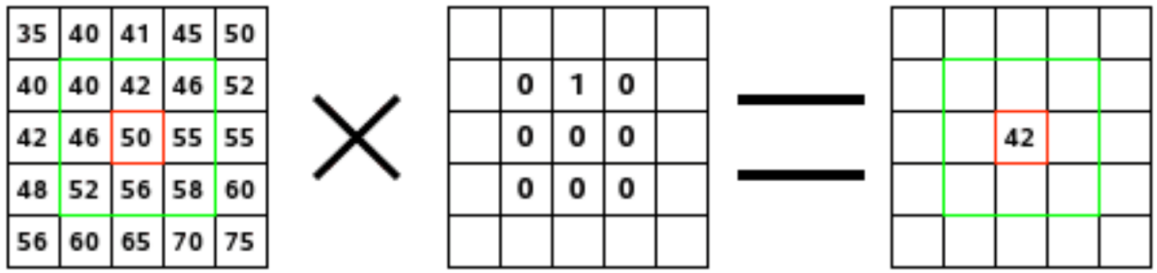


Image processing *tasks* – more on denoising

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



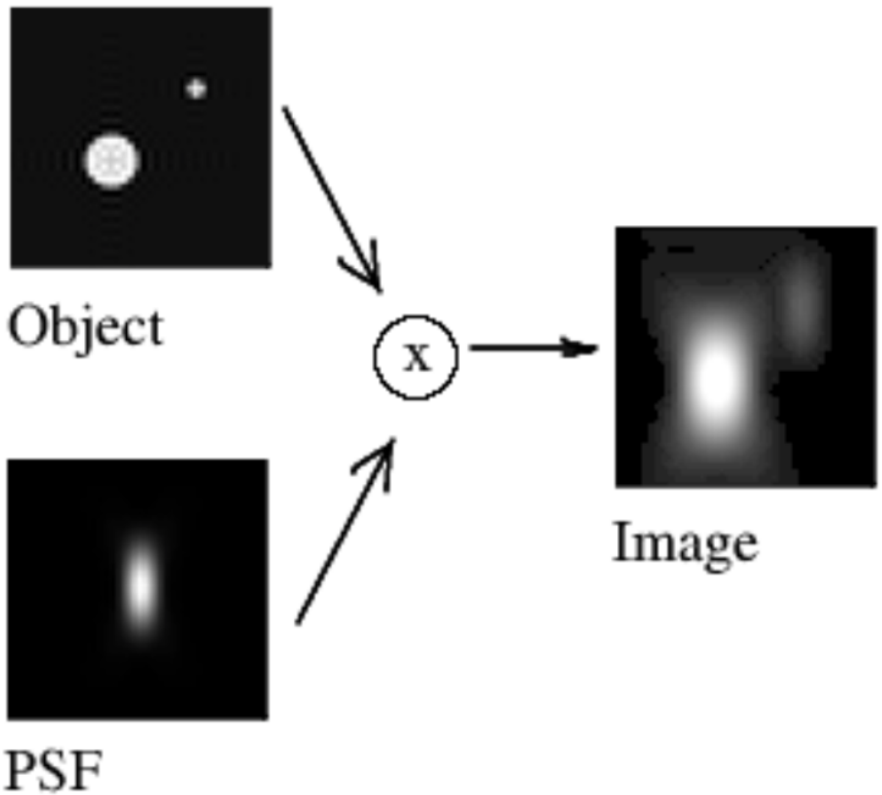
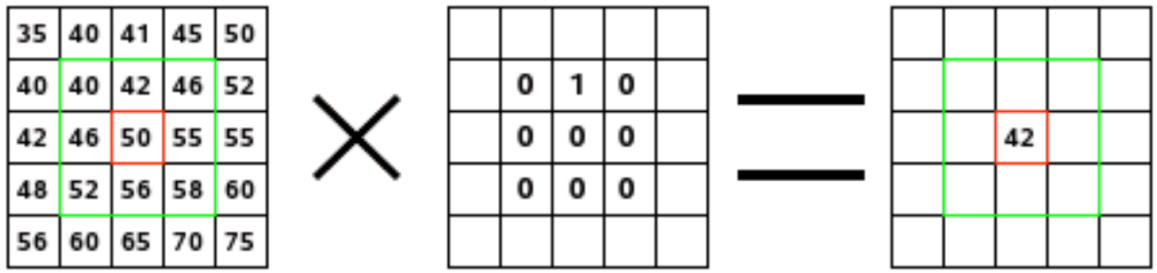


Image processing *tasks* – more on denoising

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



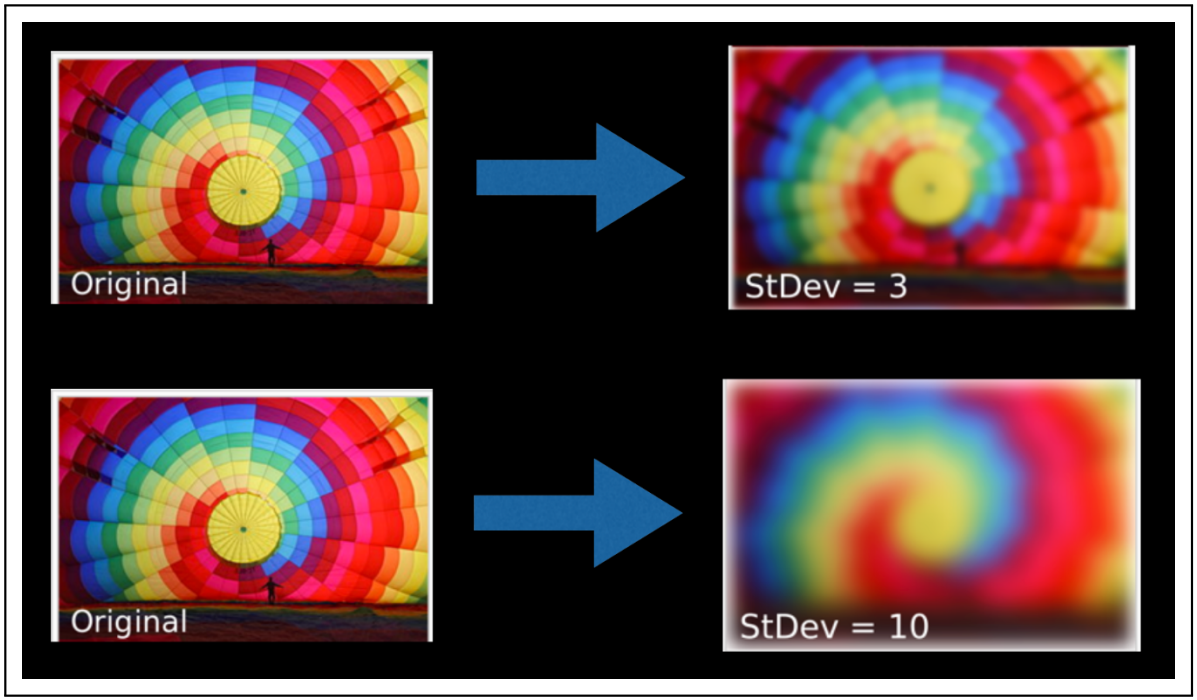


Image processing *tasks* – more on denoising

- Variation Minimization: Variation Fidelity tradeoff

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

Image processing *tasks* – more on denoising

- Variation Minimization: Variation Fidelity tradeoff

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

[What is a "hard" image for this procedure?]

Image processing *tasks*

- Grayscale (luminescence): Tensor $\rightarrow$ Matrix

$$\text{Normalize } 0.2125R + 0.7154G + 0.0721B$$

Canny Filter Sobel Filter

Image processing *tasks*

- Grayscale (luminescence): Tensor $\rightarrow$ Matrix

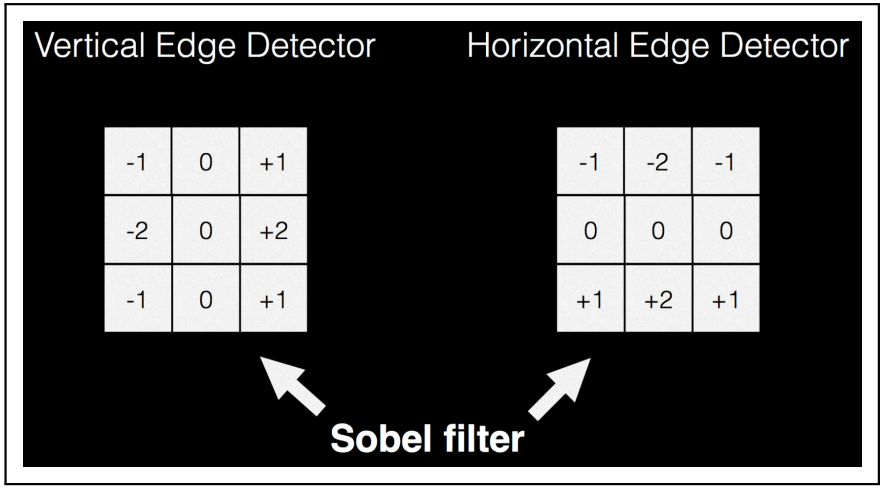
$$\text{Normalize } 0.2125R + 0.7154G + 0.0721B$$

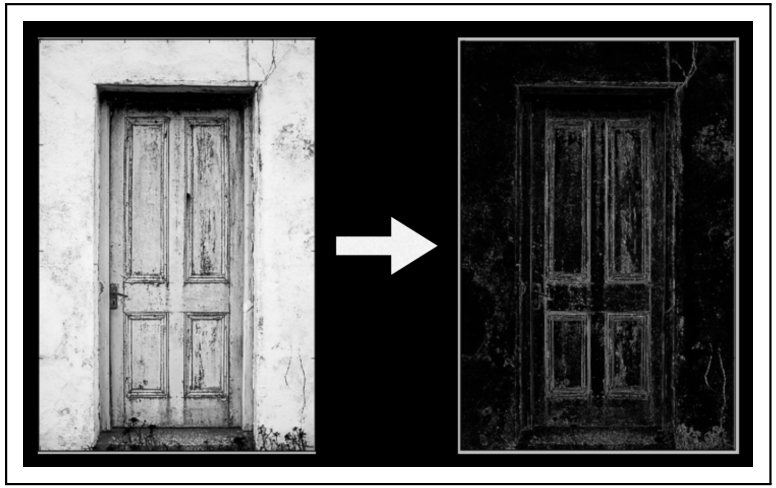
- Edge detection (grayscale): detects large pixel transitions

Canny Filter Sobel Filter

Image processing *tasks* – more on edge-detection

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





We may be able to use intensity gradients for feature ascertainment

Image processing *tasks*

- Image processing libraries
  - Scikit-image (skimage)
  - OpenCV (and dependencies)
  - Python Imaging Library
  - Pillow (a fork of the above)
  - etc.

Image *Color Structure*

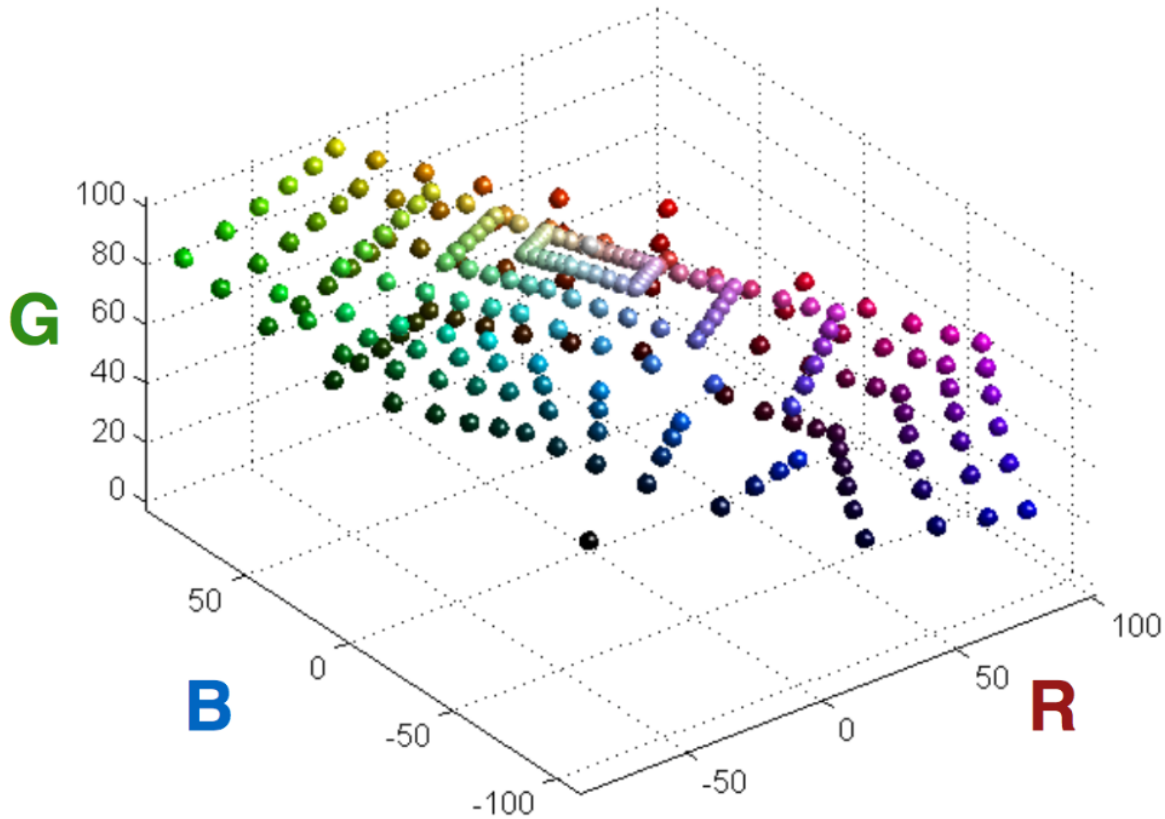
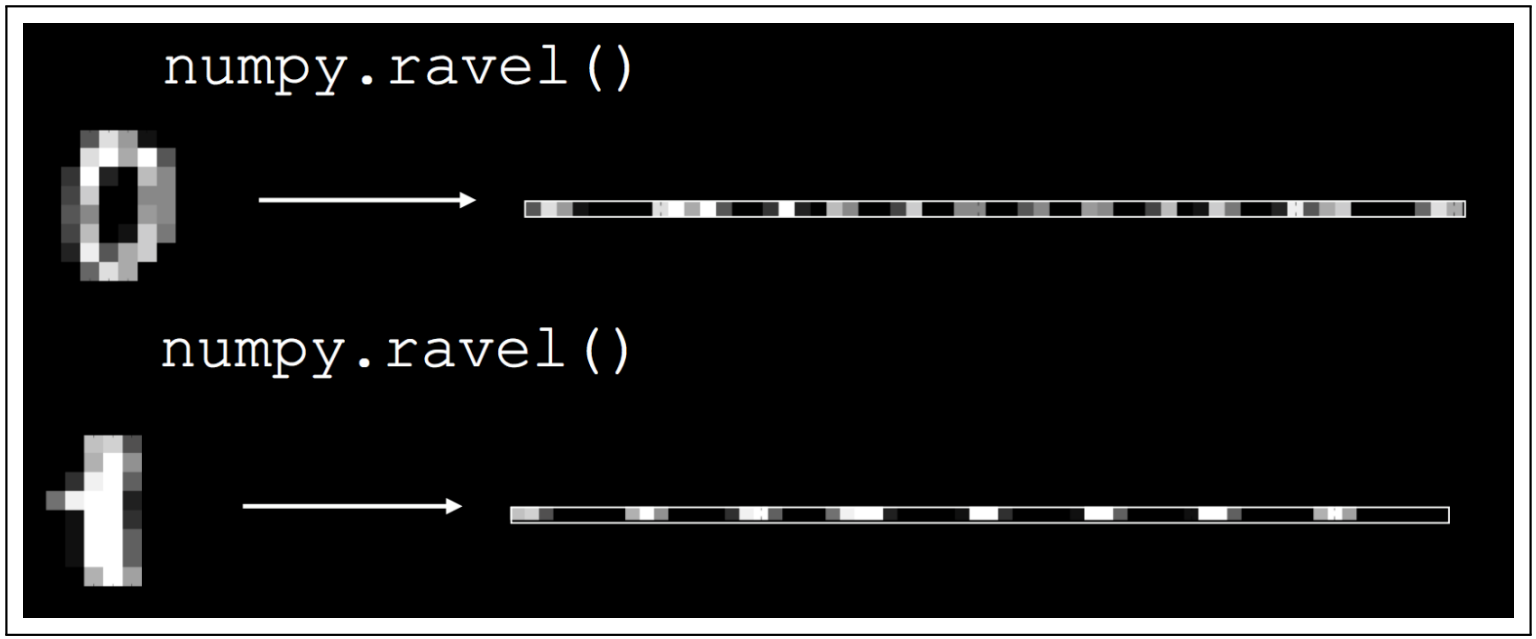
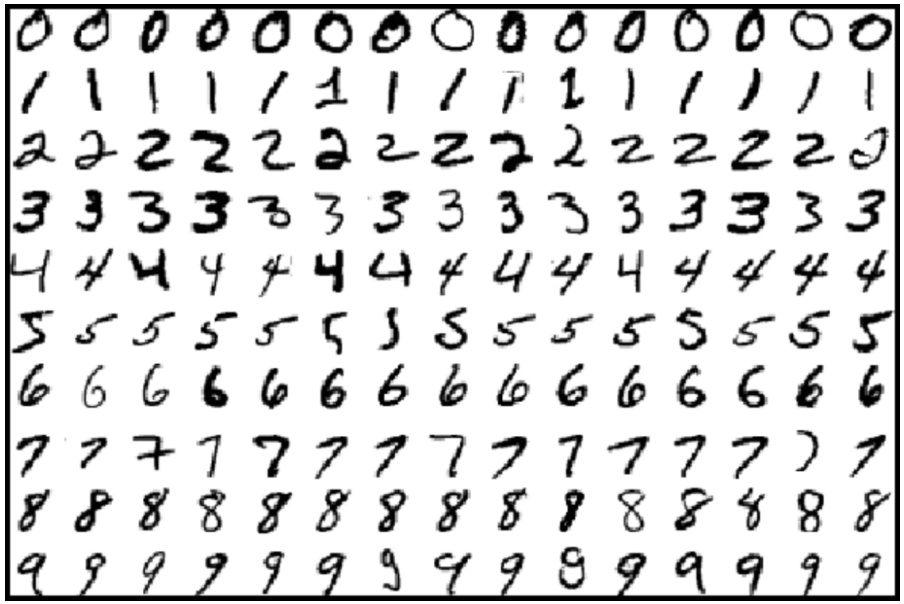


Image *Color Structure*

Image *Featurization*



Motivation: MNIST



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

Artificial Neural Network (NN)

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

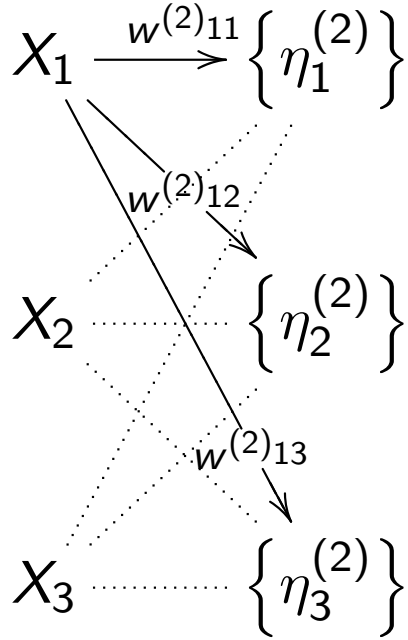
$$\left\{\eta_1^{(2)}\right\}$$

$$\left\{\eta_2^{(2)}\right\}$$

$$\left\{\eta_3^{(2)}\right\}$$

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 can be embedded into NNs as collections of states

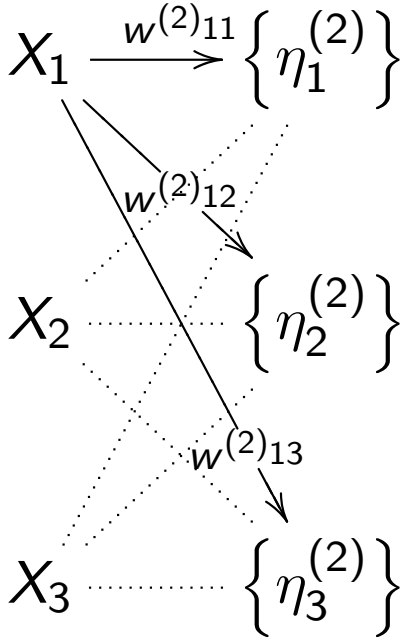


Artificial Neural Network (NN)

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

- Features can be embedded into NNs as collections of states

  Nodes capture an idea or concept



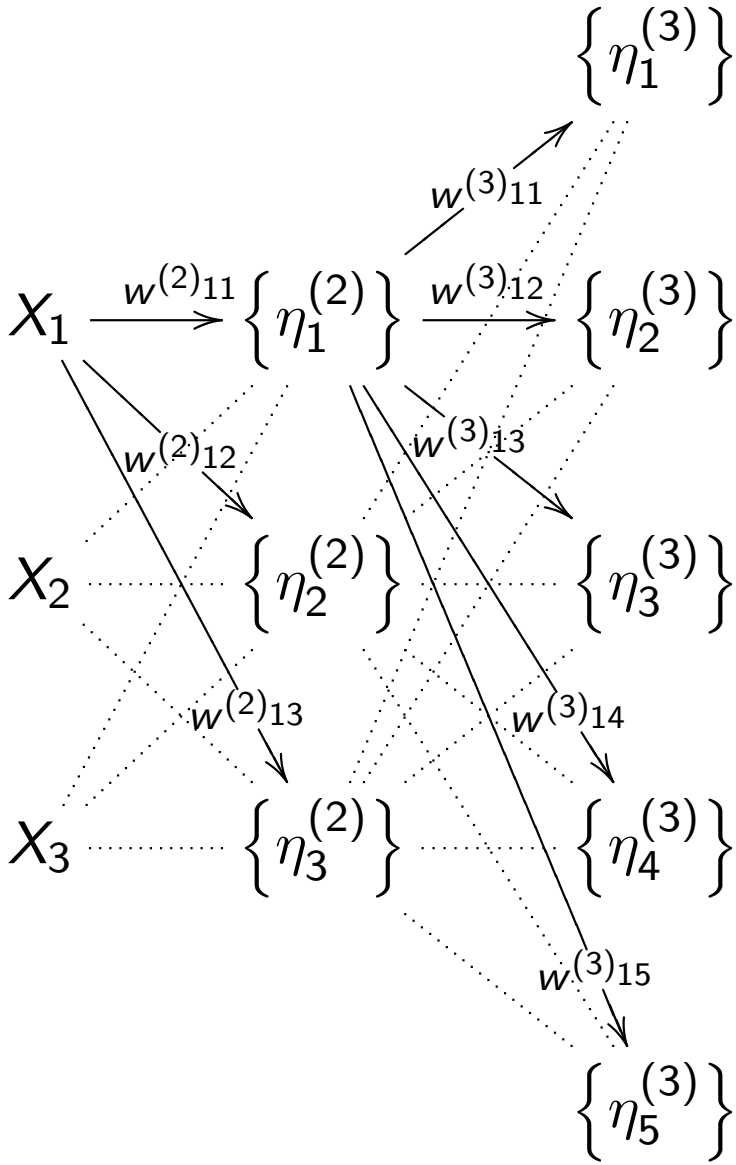
Artificial Neural Network (NN)

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

- Features can be embedded into NNs as collections of states

  Nodes capture an idea or concept

  These states can in turn be embedded in further layers of states



Artificial Neural Network (NN)

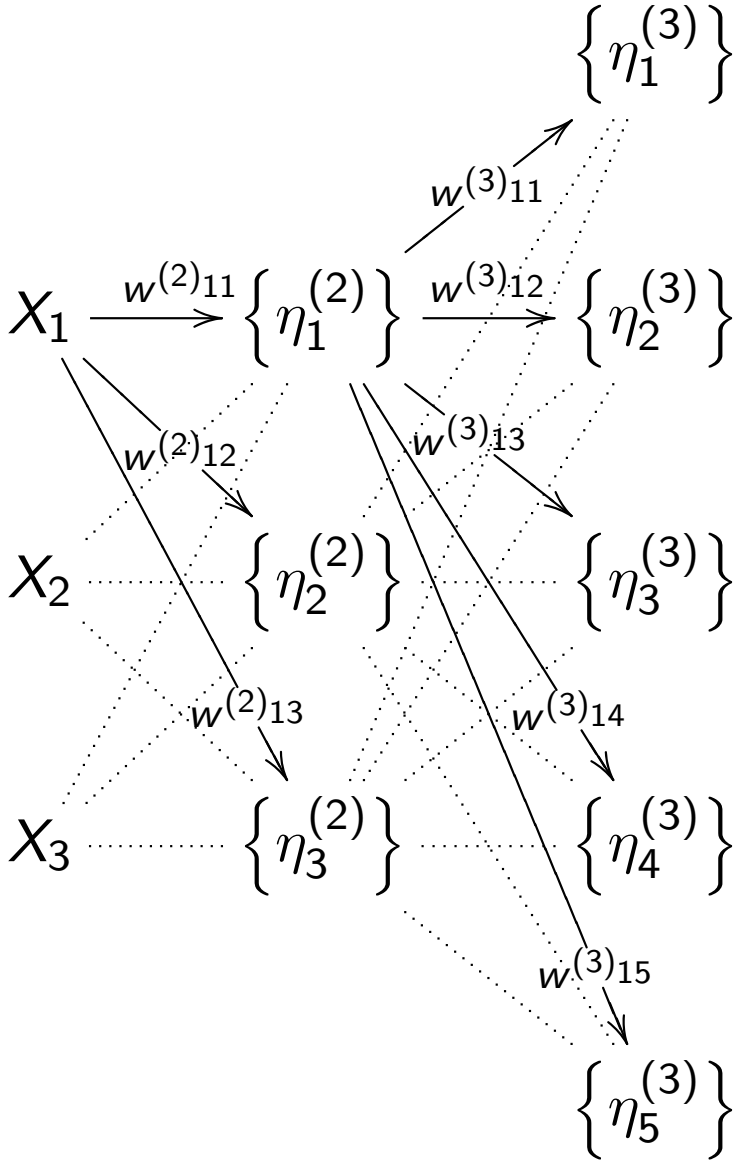
- *NN's* are a collection of nodes $\eta_j^{(l)}$ that take on various states $\left\{\eta_j^{(l)}\right\}$

- Features can be embedded into NNs as collections of states

  Nodes capture an idea or concept

  These states can in turn be embedded in further layers of states

  Each hierarchical layer provides higher orders of abstraction



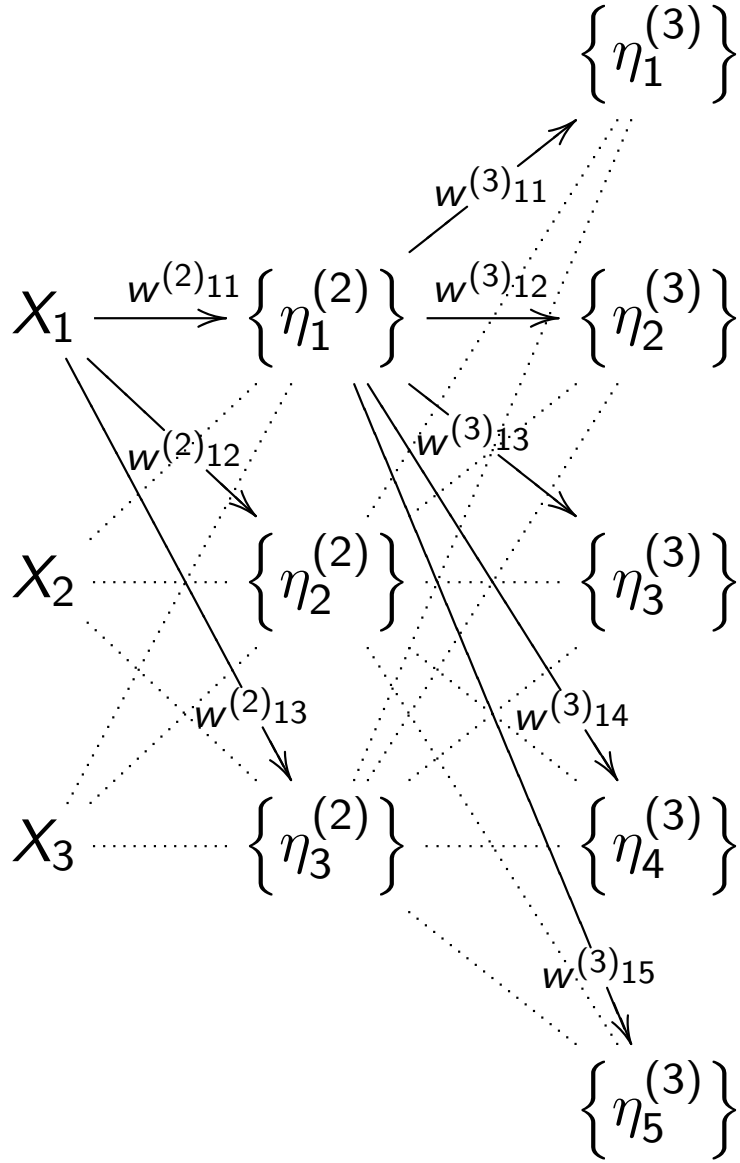
Artificial Neural Network (NN)

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

  Nodes capture an idea or concept

  These states can in turn be embedded in further layers of states

  Each hierarchical layer provides higher orders of abstraction
- The transformation weights $w_{ij}^{(l)}$ and the topology provide the map from "features" to "states"



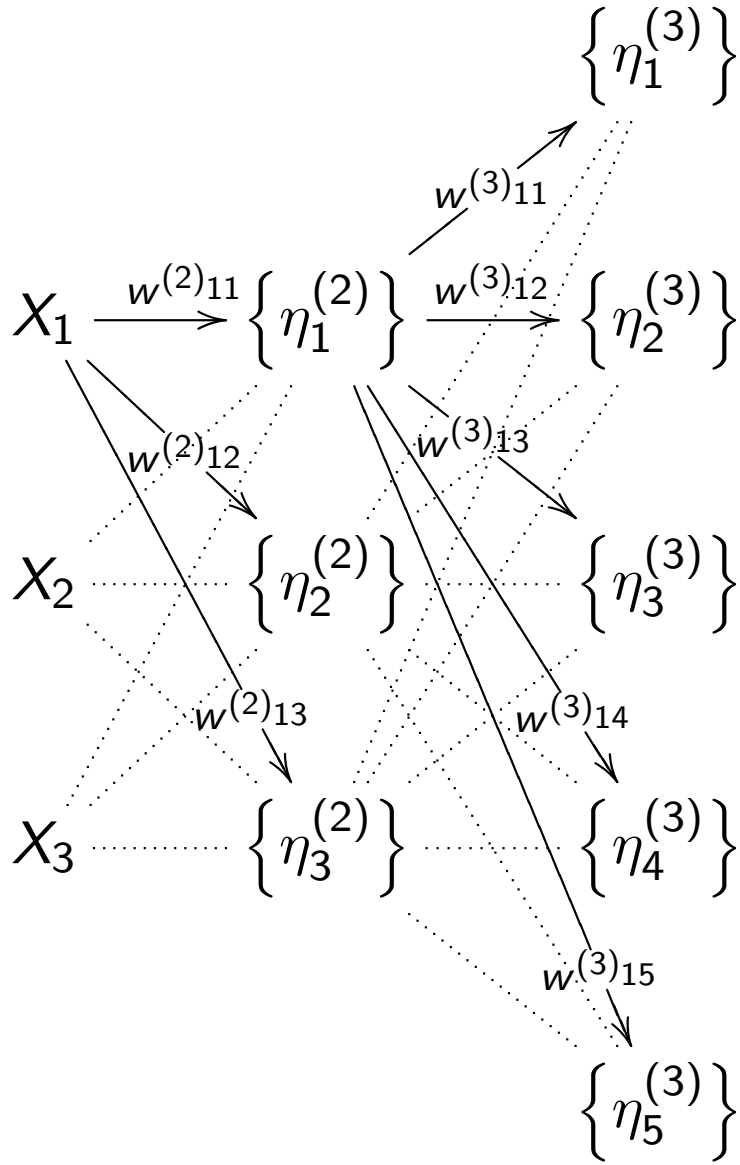
Implementing NN's: *non-linear functions* of *linear models*

E.g., $\left\{\eta_1^{(2)}\right\} = f\left(X^T W_1^{(2)}\right)$ where

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

$$W_j^{(2)} = \left[w^{(2)_{1j}}, w^{(2)_{2j}}, w^{(2)_{3j}}\right]^T$$

with activation function f



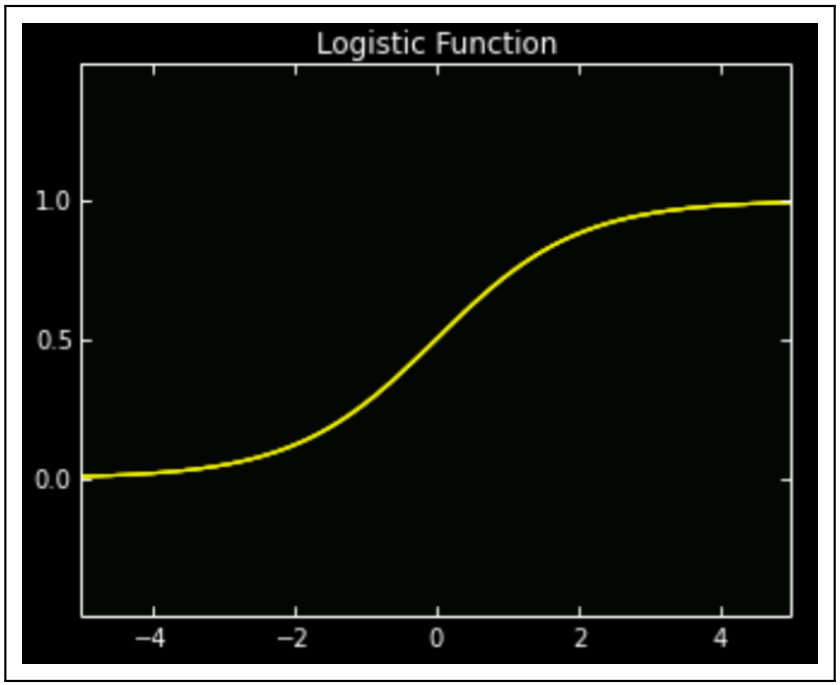
Implementing NN's: *non-linear functions* of *linear models*

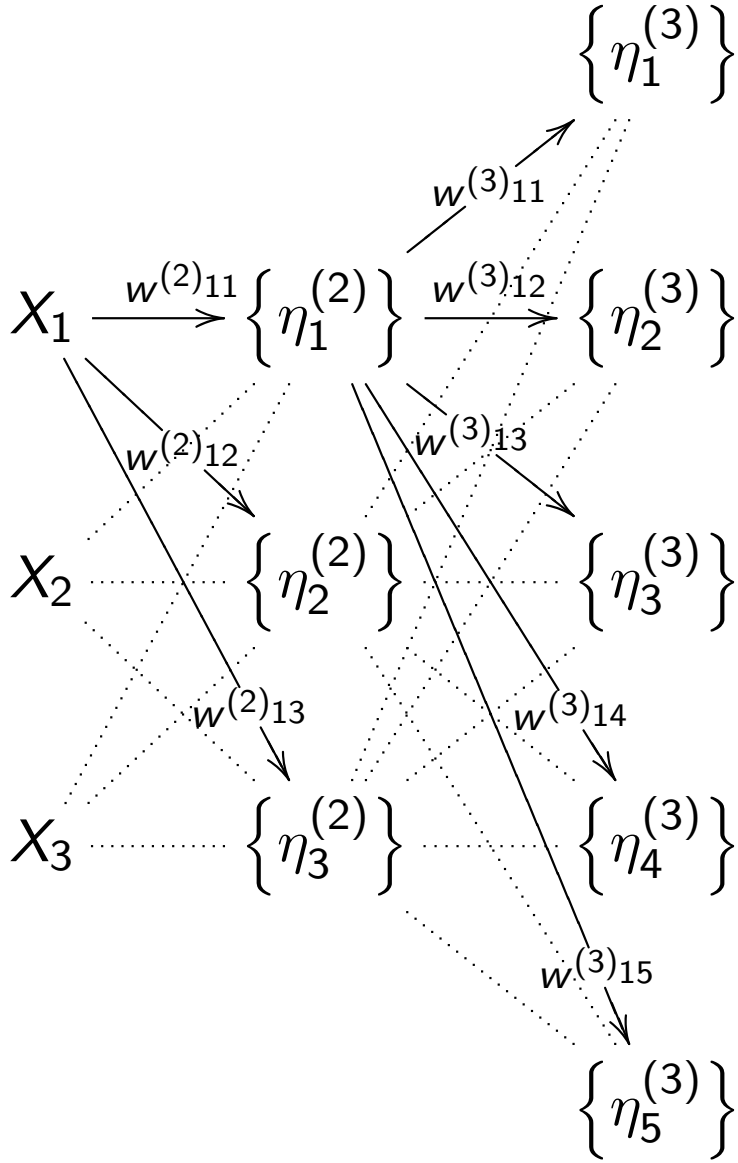
E.g., $\left\{\eta_1^{(2)}\right\} = f\left(X^T W_1^{(2)}\right)$ where

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

$$W_j^{(2)} = \left[w^{(2)}{}_{1j}, w^{(2)}{}_{2j}, w^{(2)}{}_{3j}\right]^T$$

with activation function f





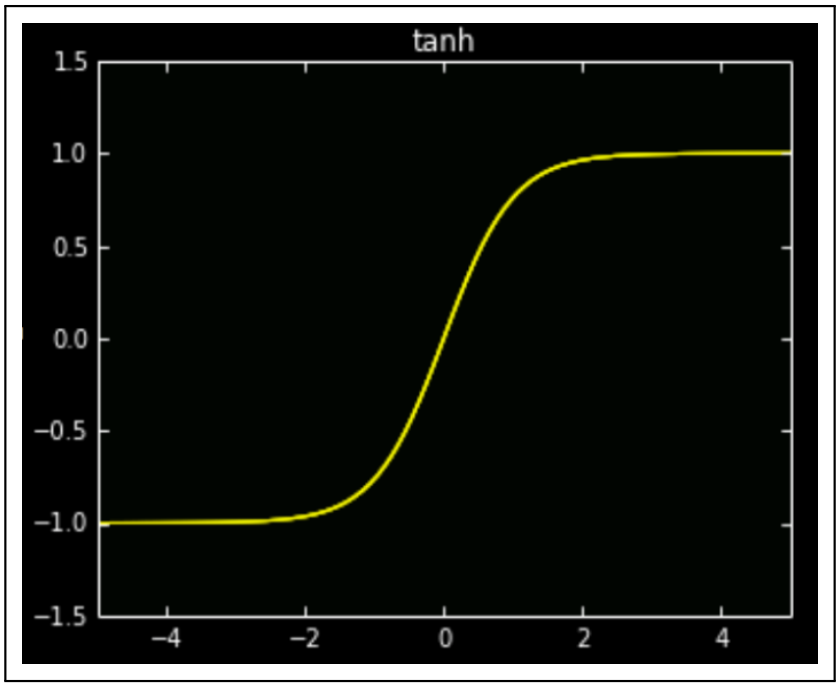
Implementing NN's: *non-linear functions* of *linear models*

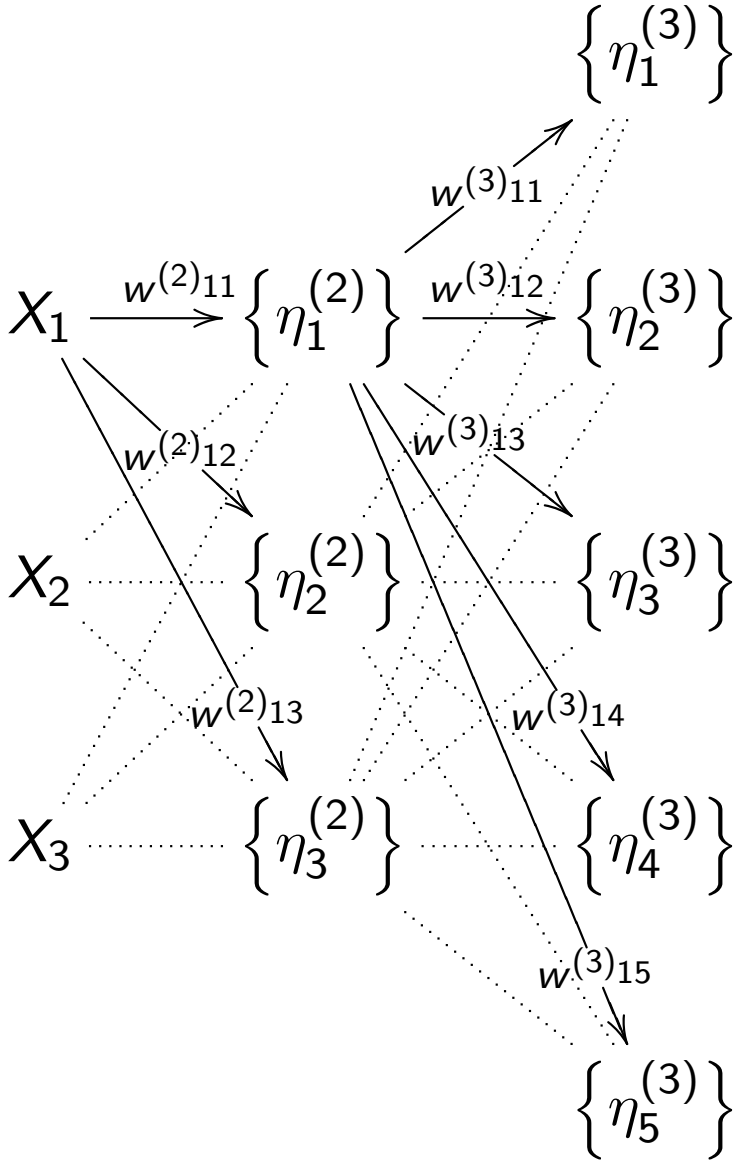
E.g., $\left\{\eta_1^{(2)}\right\} = f\left(X^T W_1^{(2)}\right)$ where

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

$$W_j^{(2)} = \left[w^{(2)}{}_{1j}, w^{(2)}{}_{2j}, w^{(2)}{}_{3j}\right]^T$$

with activation function f





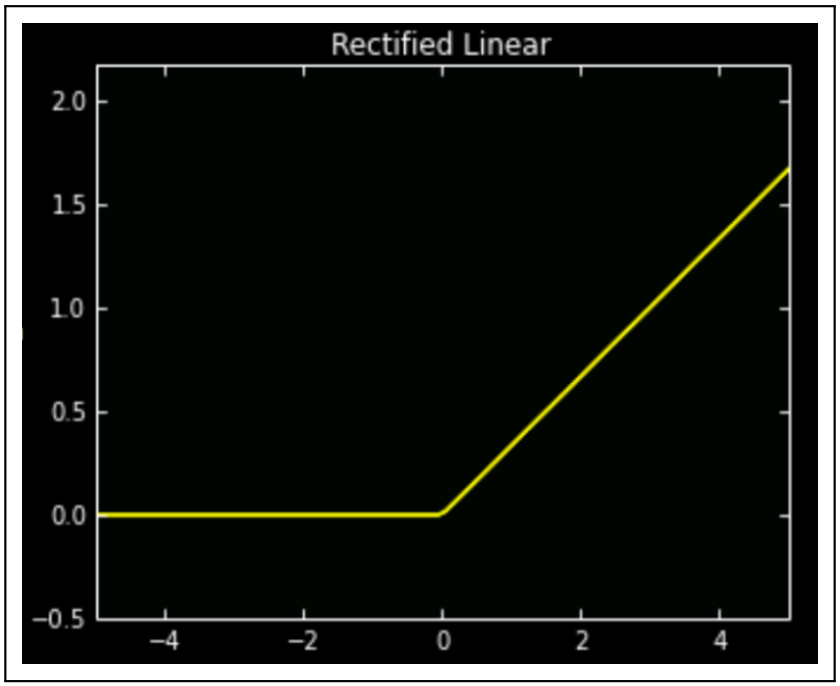
Implementing NN's: *non-linear functions* of *linear models*

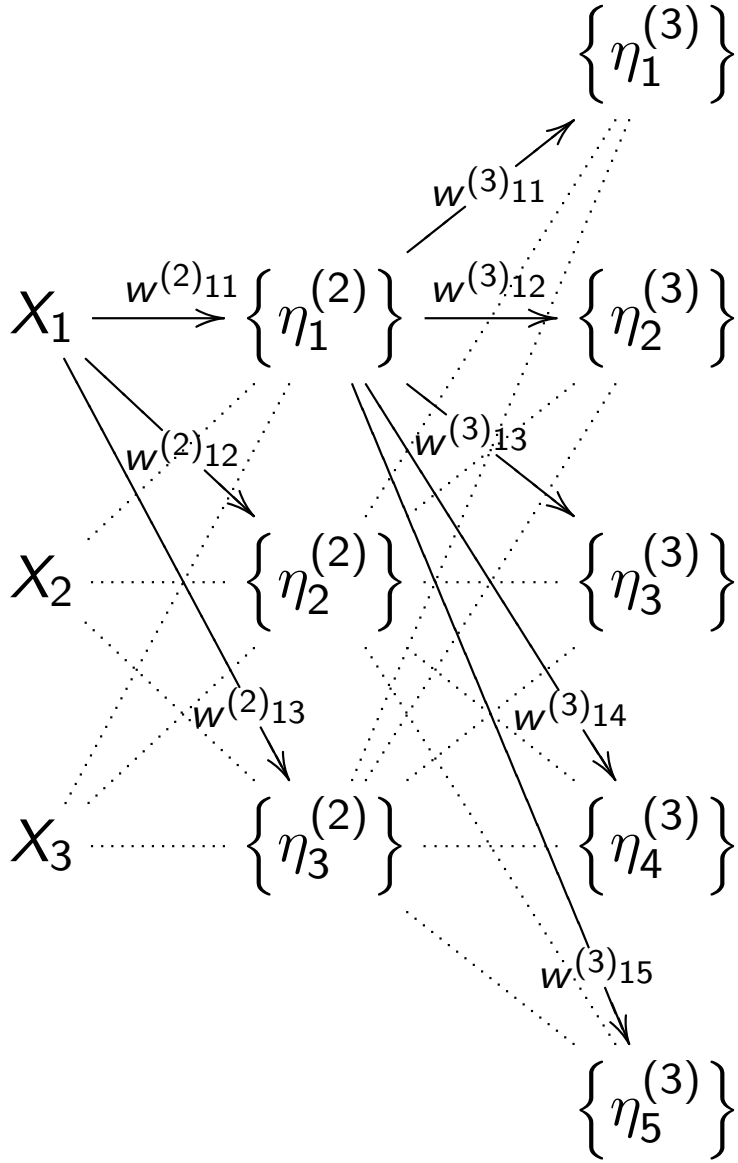
E.g., $\left\{\eta_1^{(2)}\right\} = f\left(X^T W_1^{(2)}\right)$ where

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

$$W_j^{(2)} = \left[w^{(2)}{}_{1j}, w^{(2)}{}_{2j}, w^{(2)}{}_{3j}\right]^T$$

with activation function f





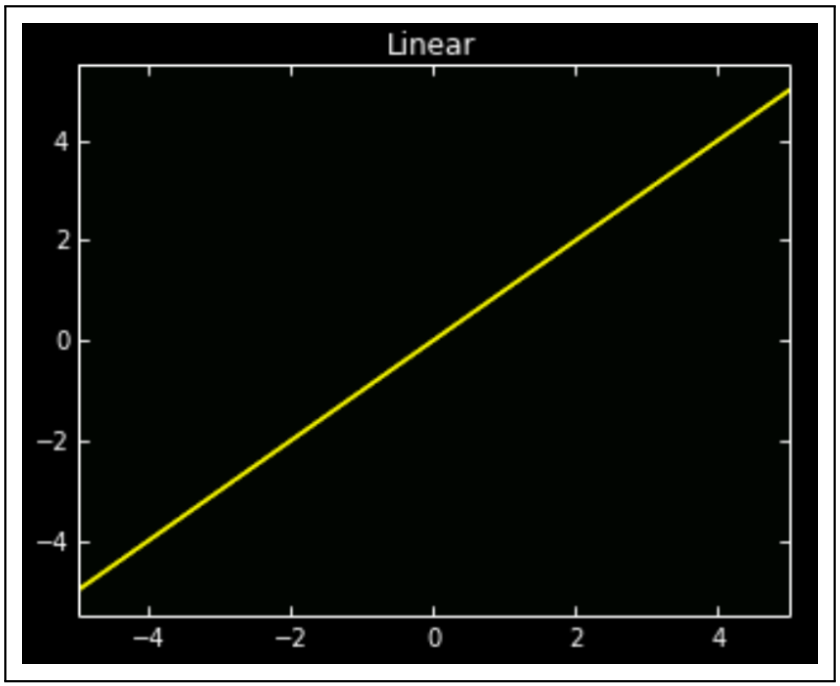
Implementing NN's: *non-linear functions* of *linear models*

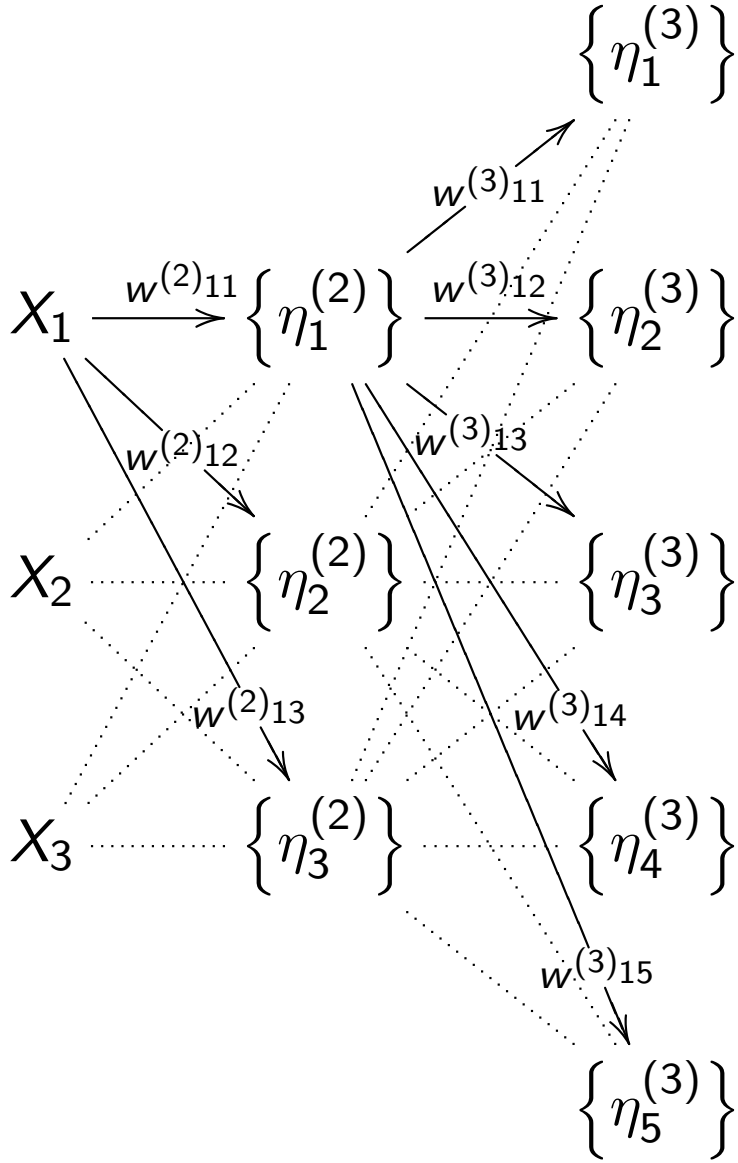
E.g., $\left\{\eta_1^{(2)}\right\} = f\left(X^T W_1^{(2)}\right)$ where

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

$$W_j^{(2)} = \left[w^{(2)}{}_{1j}, w^{(2)}{}_{2j}, w^{(2)}{}_{3j}\right]^T$$

with activation function f



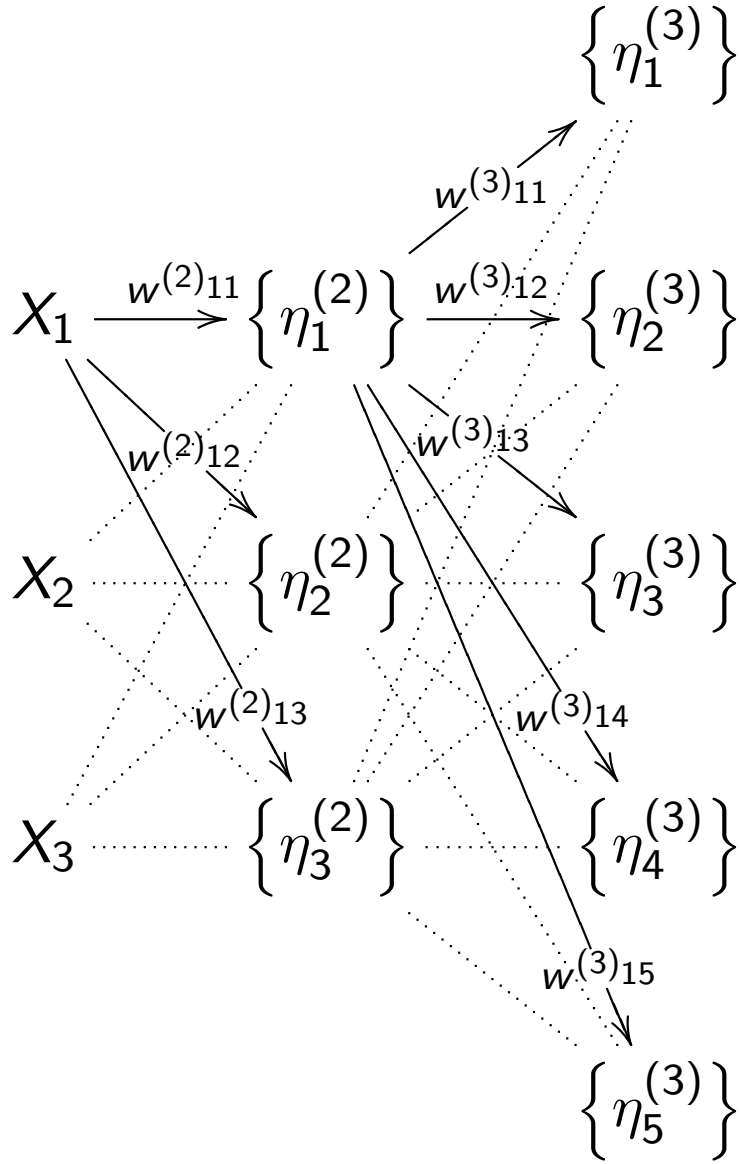


Interpreting NN's: a model of mind

- The (non-linear) activation function f maps, e.g., $X^T W_1^{(2)}$

  $\left(\text{and then, e.g., } \left\{\eta^{(2)}\right\}^T W_1^{(3)} \right)$

  onto an action potential



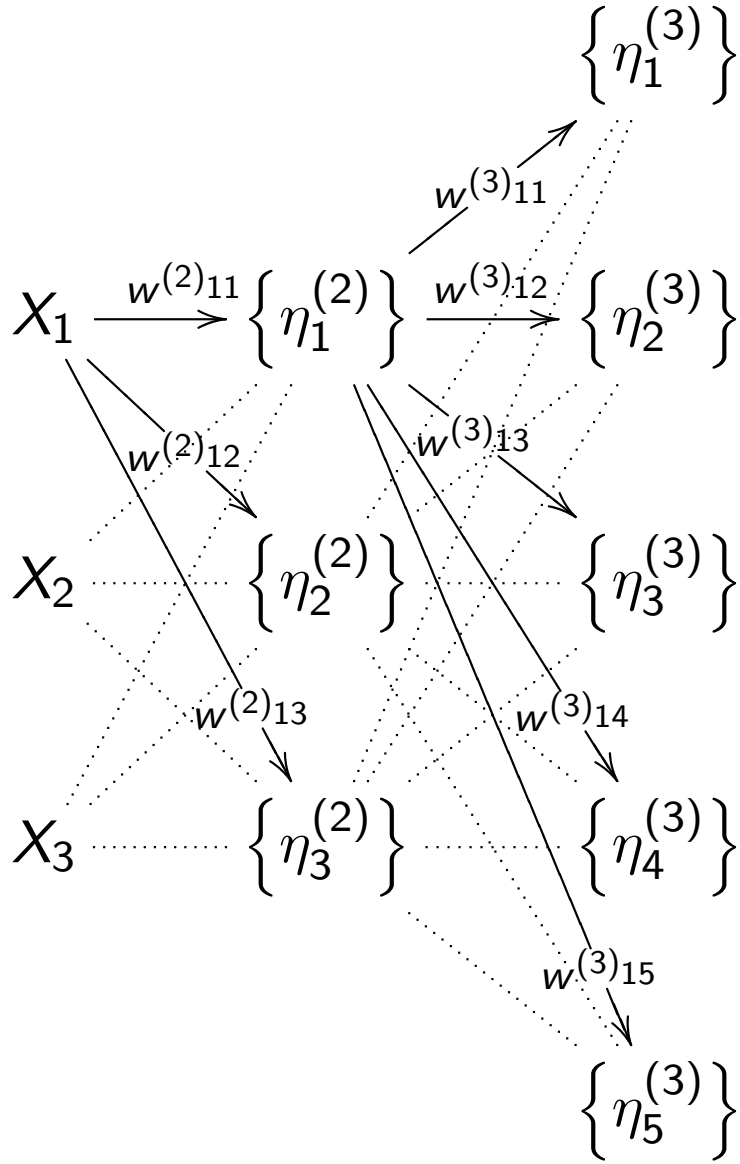
Interpreting NN's: a model of mind

- The (non-linear) activation function f maps, e.g., $X^T W_1^{(2)}$

  $\left(\text{and then, e.g., } \left\{\eta^{(2)}\right\}^T W_1^{(3)} \right)$

  onto an action potential

  When a threshold is passed the neuron (node) is on!



Interpreting NN's: a model of mind

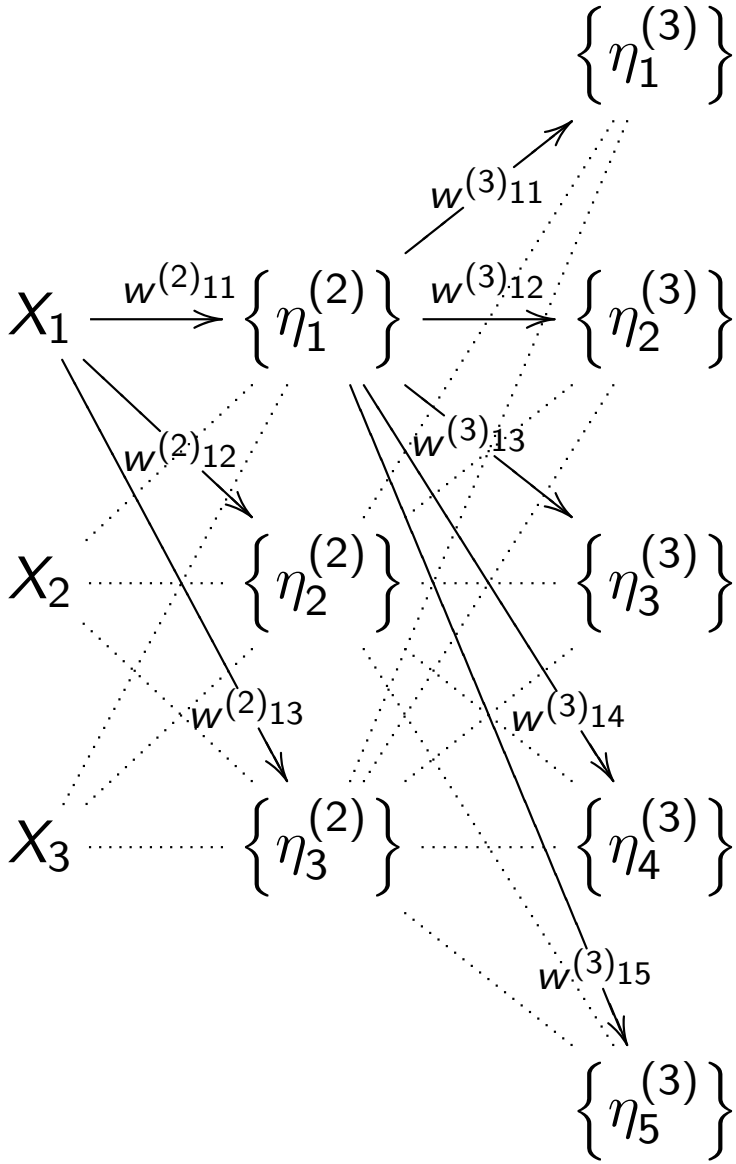
- The (non-linear) activation function f maps, e.g., $X^T W_1^{(2)}$

  $\left(\text{and then, e.g., } \left\{\eta^{(2)}\right\}^T W_1^{(3)} \right)$

  onto an action potential

  When a threshold is passed the neuron (node) is on!

  An "on" neuron is an "on" "idea"



Interpreting NN's: a model of mind

- The (non-linear) activation function f maps, e.g., $X^T W_1^{(2)}$

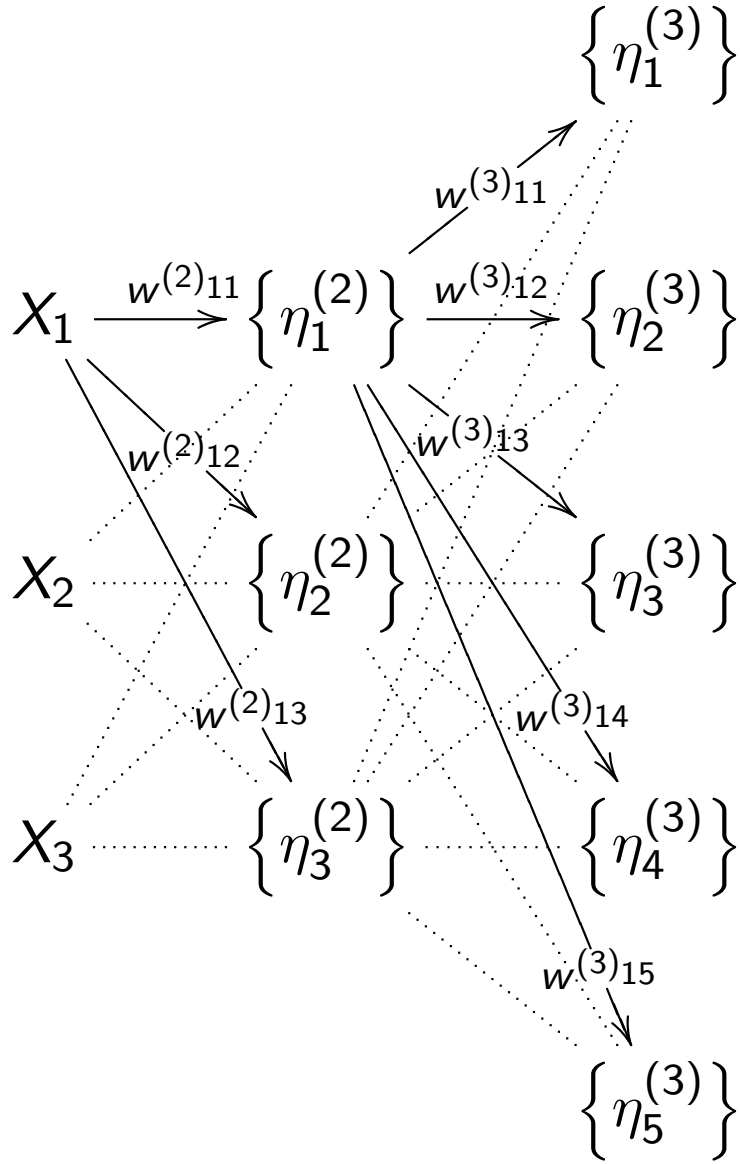
  $\left(\text{and then, e.g., } \left\{\eta^{(2)}\right\}^T W_1^{(3)} \right)$

  onto an action potential

  When a threshold is passed the neuron (node) is on!

  An "on" neuron is an "on" "idea"

  An NN's state at any time is it's current collection of "thoughts"



Interpreting NN's: a model of mind

- The (non-linear) activation function f maps, e.g., $X^T W_1^{(2)}$

  $\left(\text{and then, e.g., } \left\{\eta^{(2)}\right\}^T W_1^{(3)} \right)$

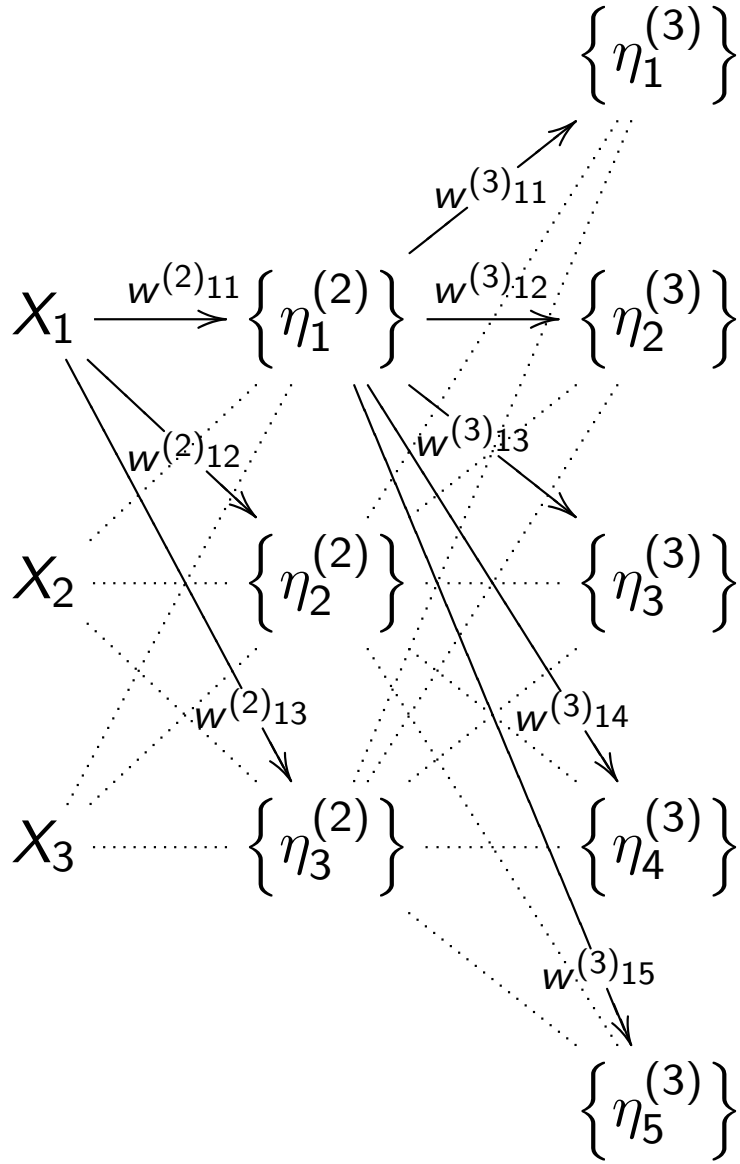
  onto an action potential

  When a threshold is passed the neuron (node) is on!

  An "on" neuron is an "on" "idea"

  An NN's state at any time is it's current collection of "thoughts"

- Now we "train" the $w's$ so similar X's produce similar NN states



Interpreting NN's: a model of mind

- The (non-linear) activation function f maps, e.g., $X^T W_1^{(2)}$

  $\left(\text{and then, e.g., } \left\{\eta^{(2)}\right\}^T W_1^{(3)} \right)$

  onto an action potential

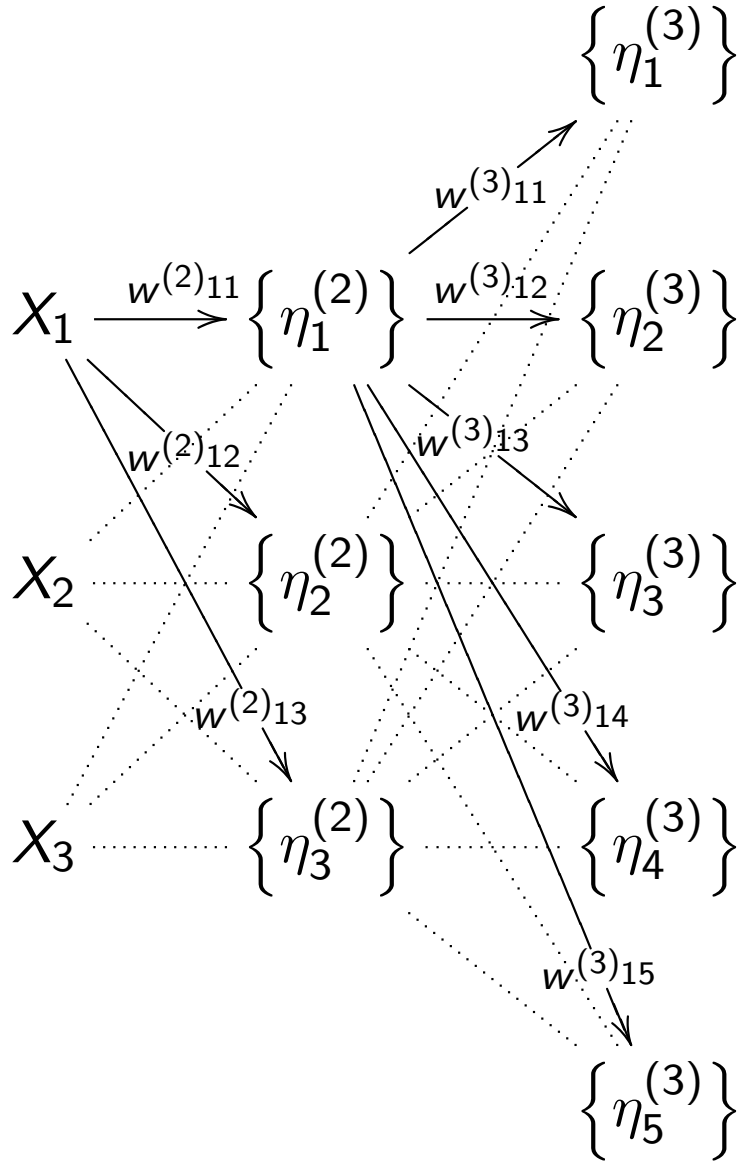
  When a threshold is passed the neuron (node) is on!

  An "on" neuron is an "on" "idea"

  An NN's state at any time is it's current collection of "thoughts"

- Now we "train" the $w's$ so similar X's produce similar NN states

  The NN "knows/recognizes" X's

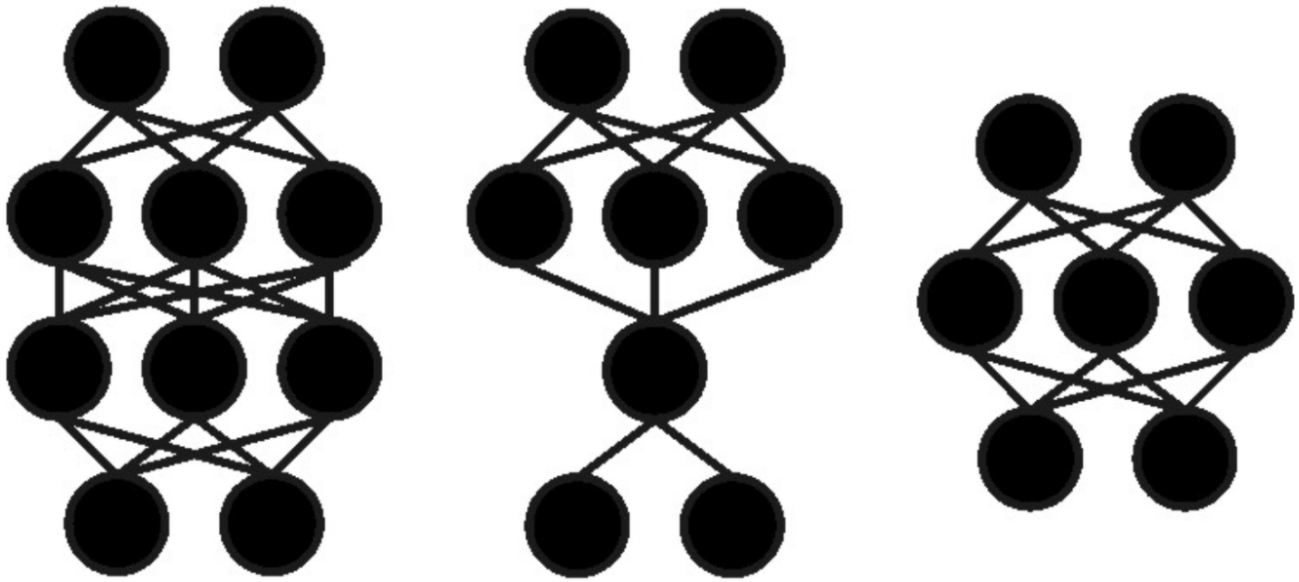


Using NN's: where the fun ends and the pain begins

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

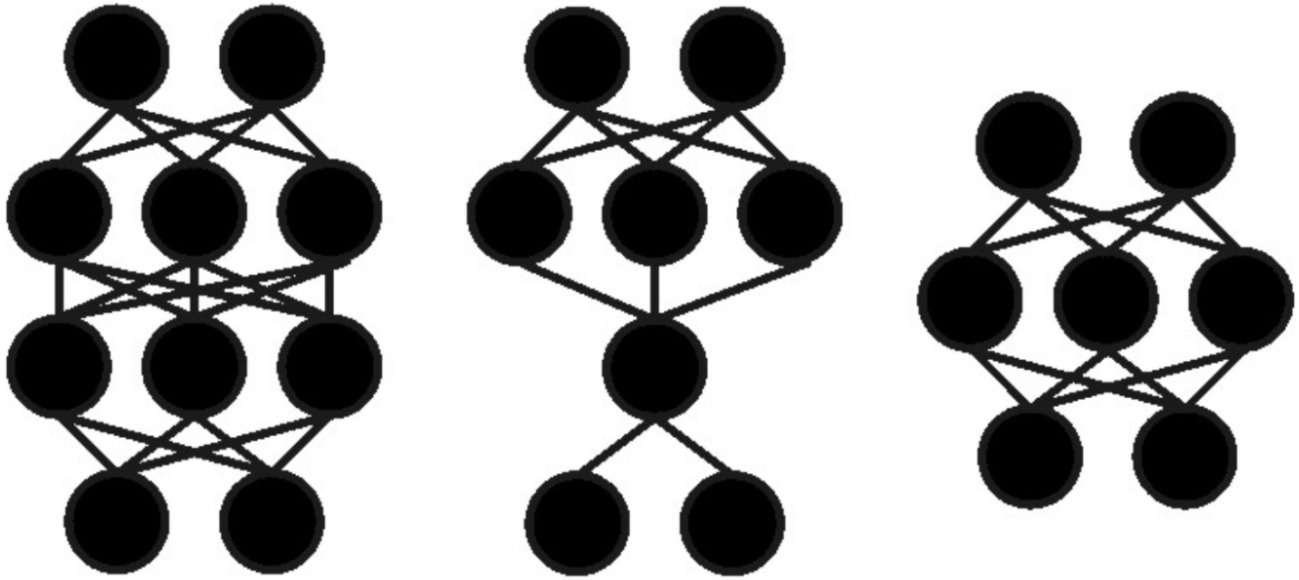
Using NN's: where the fun ends and the pain begins

- So patterns are trained (embedded) into an NN...
- Wait... what's the NN again?



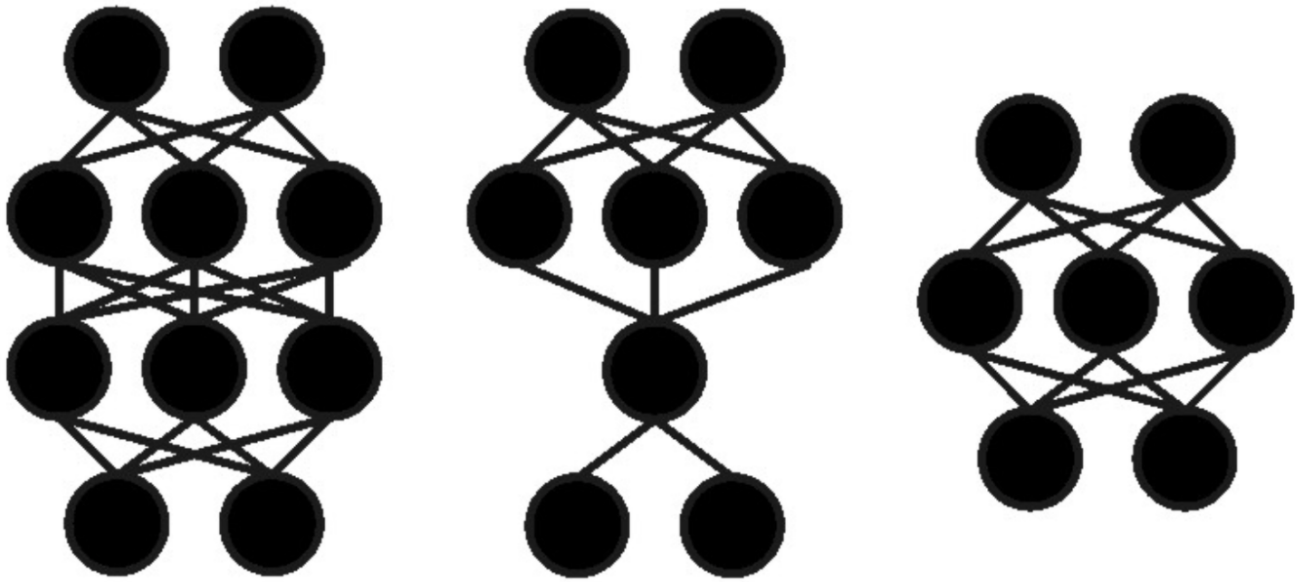
Using NN's: where the fun ends and the pain begins

- So patterns are trained (embedded) into an NN...
- Wait... what's the NN again? We have to specify topology



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

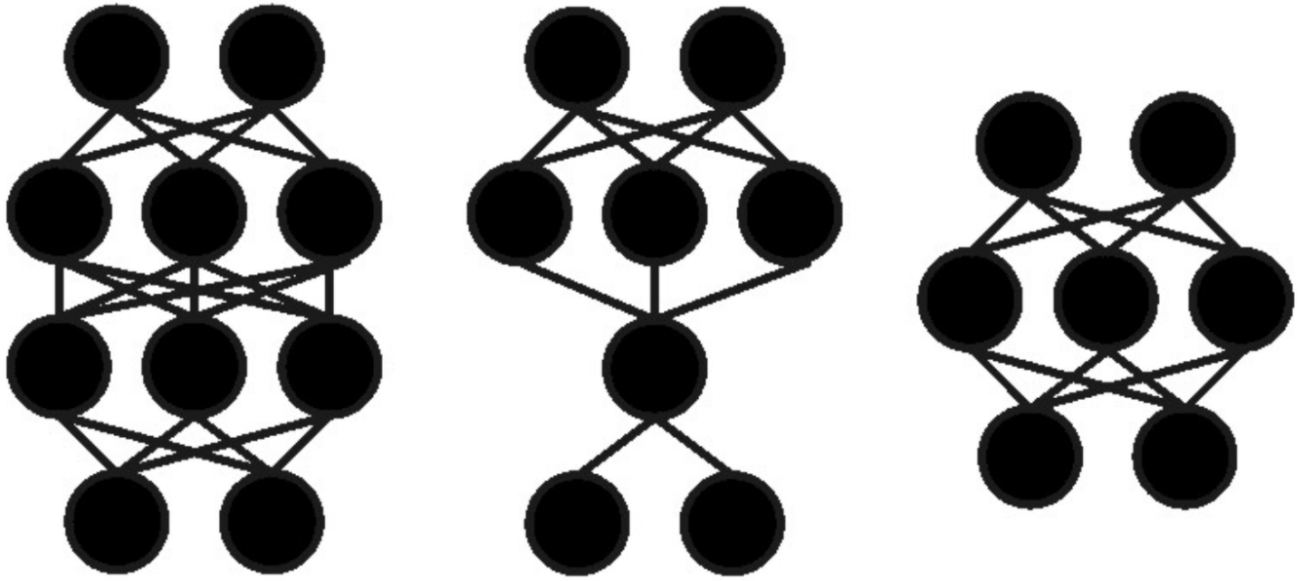
- So patterns are trained (embedded) into an NN...
- Wait... what's the NN again? We have to specify topology



- 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...
- Wait... what's the NN again? We have to specify topology



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

A first NN

We observe feature X. Subsequently, we observe an associated continuous valued Y. Similar X's seem to produce similar Y's.

We want to "understand" what Y will be upon observing $X = x$.

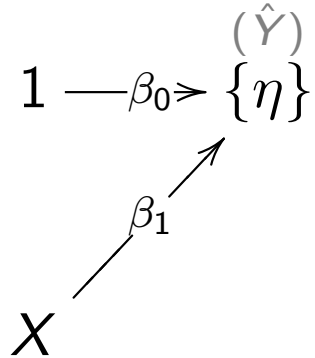
Let's design an NN with a node η that captures the "idea" of $Y|X$
This will be gauged by the NN's ability to predict Y ($\hat{Y}$) from X.
We will use a linear (i.e., identity) activation function to produce $\hat{Y}$

A first NN

We observe feature X. Subsequently, we observe an associated continuous valued Y. Similar X's seem to produce similar Y's.

We want to "understand" what Y will be upon observing $X = x$.

Let's design an NN with a node η that captures the "idea" of $Y|X$
This will be gauged by the NN's ability to predict Y ($\hat{Y}$) from X.
We will use a linear (i.e., identity) activation function to produce $\hat{Y}$

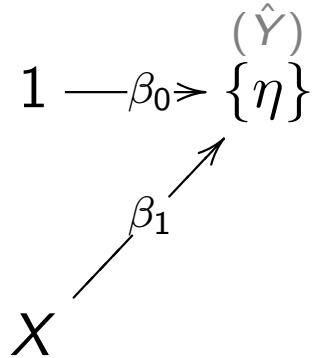


A first NN

We observe feature X. Subsequently, we observe an associated continuous valued Y. Similar X's seem to produce similar Y's.

We want to "understand" what Y will be upon observing $X = x$.

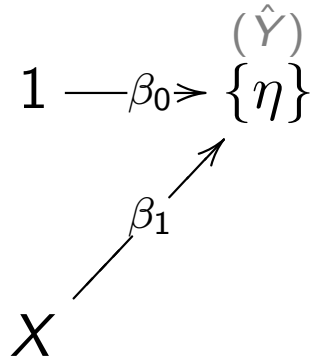
Let's design an NN with a node η that captures the "idea" of $Y|X$
This will be gauged by the NN's ability to predict Y ($\hat{Y}$) from X.
We will use a linear (i.e., identity) activation function to produce $\hat{Y}$



We'll train the NN with a squared loss cost function $\frac{1}{2}\sum(Y_i - \hat{Y}_i)^2$

A first NN

We'll train the NN with a squared loss cost function $\frac{1}{2}\sum(Y_i - \hat{Y}_i)^2$



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

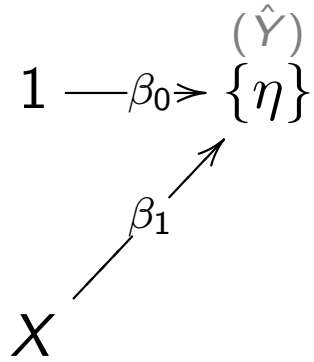
$$= [1, x]\begin{bmatrix}\beta_0 \\ \beta_1\end{bmatrix} = \beta_0 + \beta_1 x$$

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

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

A first NN

We'll train the NN with a squared loss cost function $\frac{1}{2}\sum(Y_i - \hat{Y}_i)^2$



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

$$= [1, x]\begin{bmatrix}\beta_0 \\ \beta_1\end{bmatrix} = \beta_0 + \beta_1 x$$

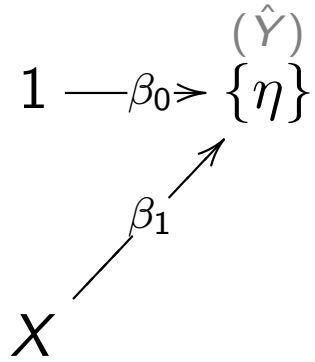
$$\min_{\hat{Y}} \frac{1}{2}\sum(Y_i - \hat{Y}_i)^2$$

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

Oh my doesn't that look familiar...

A first NN

We'll train the NN with a squared loss cost function $\frac{1}{2}\sum(Y_i - \hat{Y}_i)^2$



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

$$= [1, x]\begin{bmatrix}\beta_0 \\ \beta_1\end{bmatrix} = \beta_0 + \beta_1 x$$

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

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

Oh my doesn't that look familiar... okay let's try again

A second NN

We observe feature X. Subsequently, we observe an associated binary outcome Y. The outcomes Y seem to be associated with X.

We want to "understand" what Y will be upon observing $X = x$.

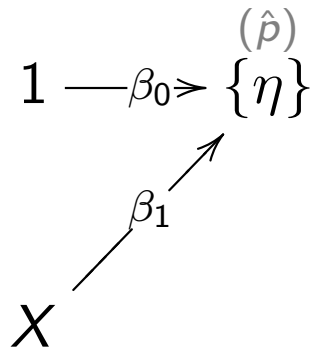
Let's design an NN with a node η capturing the "idea" of $\Pr(Y|X)$
This will be gauged by the NN's ability to predict Y ($\hat{p}$) from X.
We will use a logistic (sigmoid) activation function to produce $\hat{p}$

A second NN

We observe feature X. Subsequently, we observe an associated binary outcome Y. The outcomes Y seem to be associated with X.

We want to "understand" what Y will be upon observing $X = x$.

Let's design an NN with a node η capturing the "idea" of $\Pr(Y|X)$
This will be gauged by the NN's ability to predict Y ($\hat{p}$) from X.
We will use a logistic (sigmoid) activation function to produce $\hat{p}$

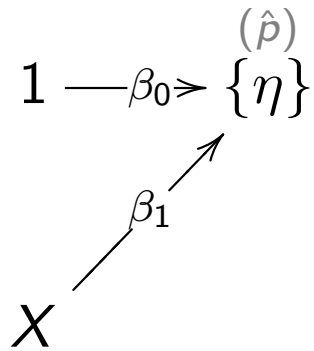


A second NN

We observe feature X. Subsequently, we observe an associated binary outcome Y. The outcomes Y seem to be associated with X.

We want to "understand" what Y will be upon observing $X = x$.

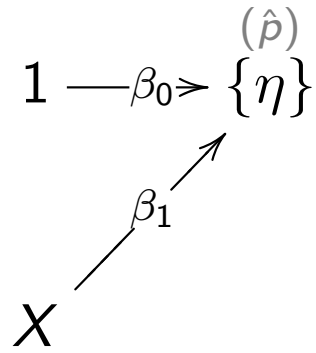
Let's design an NN with a node η capturing the "idea" of $\Pr(Y|X)$
This will be gauged by the NN's ability to predict Y ($\hat{p}$) from X.
We will use a logistic (sigmoid) activation function to produce $\hat{p}$



We'll train the NN using the log loss cost function, $-\log L(\hat{\boldsymbol{Y}}|\boldsymbol{Y})$

A second NN

We'll train the NN using the log loss cost function, $-\log L(\hat{\boldsymbol{Y}}|\boldsymbol{Y})$



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

$$= \frac{1}{1 + e^{-(\beta_0+\beta_1 x)}}$$

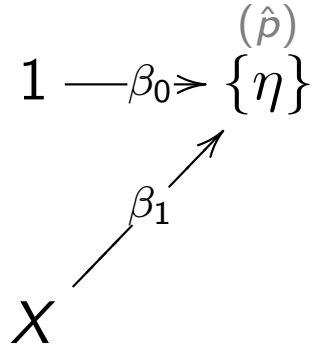
$$\min_{\boldsymbol{p}} -\prod Y_i \log(p_i) + (1 - Y_i)\log(1 - p_i)$$

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

$$+ (1 - Y_i)\log\left(\frac{1}{1 + e^{\beta_0+\beta_1 x_i}}\right)$$

A second NN

We'll train the NN using the log loss cost function, $-\log L(\hat{\boldsymbol{Y}}|\boldsymbol{Y})$



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

$$= \frac{1}{1 + e^{-(\beta_0+\beta_1 x)}}$$

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

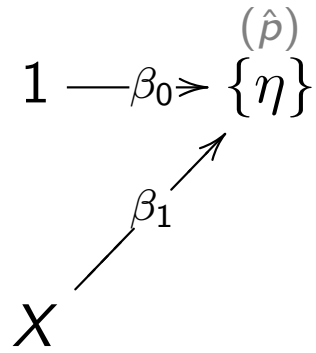
$$\hat{\beta} = \underset{\beta}{\text{argmin}} -\prod Y_i \log\left(\frac{1}{1 + e^{-(\beta_0+\beta_1 x_i)}}\right)$$

$$+ (1 - Y_i)\log\left(\frac{1}{1 + e^{\beta_0+\beta_1 x_i}}\right)$$

Oh... we've also already done that...

A second NN

We'll train the NN using the log loss cost function, $-\log L(\hat{\boldsymbol{Y}}|\boldsymbol{Y})$



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

$$= \frac{1}{1 + e^{-(\beta_0+\beta_1 x)}}$$

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

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

$$+ (1 - Y_i)\log\left(\frac{1}{1 + e^{\beta_0+\beta_1 x_i}}\right)$$

Oh... we've also already done that... okay, what gives

A general approach to fitting NN's

NN's can be fit using gradient decent

A general approach to fitting NN's

NN's can be fit using gradient decent

- *Forward propagation* generates NN states from input features

A general approach to fitting NN's

NN's can be fit using gradient decent

- *Forward propagation* generates NN states from input features
- *Backward propagation* is gradient decent on NN cost functions

A general approach to fitting NN's

NN's can be fit using gradient decent

- *Forward propagation* generates NN states from input features
- *Backward propagation* is gradient decent on NN cost functions

Why is is it called *backpropagation*?

A general approach to fitting NN's

NN's can be fit using gradient decent

- *Forward propagation* generates NN states from input features
- *Backward propagation* is gradient decent on NN cost functions

Why is is it called *backpropagation*?

[linear regression/af with squared loss cost function]

A general approach to fitting NN's

NN's can be fit using gradient decent

- *Forward propagation* generates NN states from input features
- *Backward propagation* is gradient decent on NN cost functions

Why is is it called *backpropagation*?

[linear regression/af with squared loss cost function]

$$-\frac{\partial}{\partial W_j^{(3)}}\frac{1}{2}\left(Y-\left(X^T W^{(2)}\right)W^{(3)}\right)^2$$

$$=-\frac{\partial}{\partial W_j^{(3)}}g_2\left(g_1\left(W^{(2)}\right)\right)=-\frac{\partial}{\partial g_1\left(W^{(2)}\right)}\frac{\partial g_1\left(W^{(2)}\right)}{\partial W_j^{(2)}}g_2\left(g_1\left(W^{(2)}\right)\right)$$

$$=\left(Y-\left(X^T W^{(2)}\right)W^{(3)}\right)X^T W_j^{(2)}=\textit{error}\times\textit{culpability}$$

A general approach to fitting NN's

NN's can be fit using gradient decent

- *Forward propagation* generates NN states from input features
- *Backward propagation* is gradient decent on NN cost functions

Why is is it called *backpropagation*?

[linear regression/af with squared loss cost function]

$$-\frac{\partial}{\partial W_j^{(3)}}\frac{1}{2}\left(Y-\left(X^T W^{(2)}\right)W^{(3)}\right)^2$$

$$=-\frac{\partial}{\partial W_j^{(3)}}g_2\left(g_1\left(W^{(2)}\right)\right)=-\frac{\partial}{\partial g_1\left(W^{(2)}\right)}\frac{\partial g_1\left(W^{(2)}\right)}{\partial W_j^{(2)}}g_2\left(g_1\left(W^{(2)}\right)\right)$$

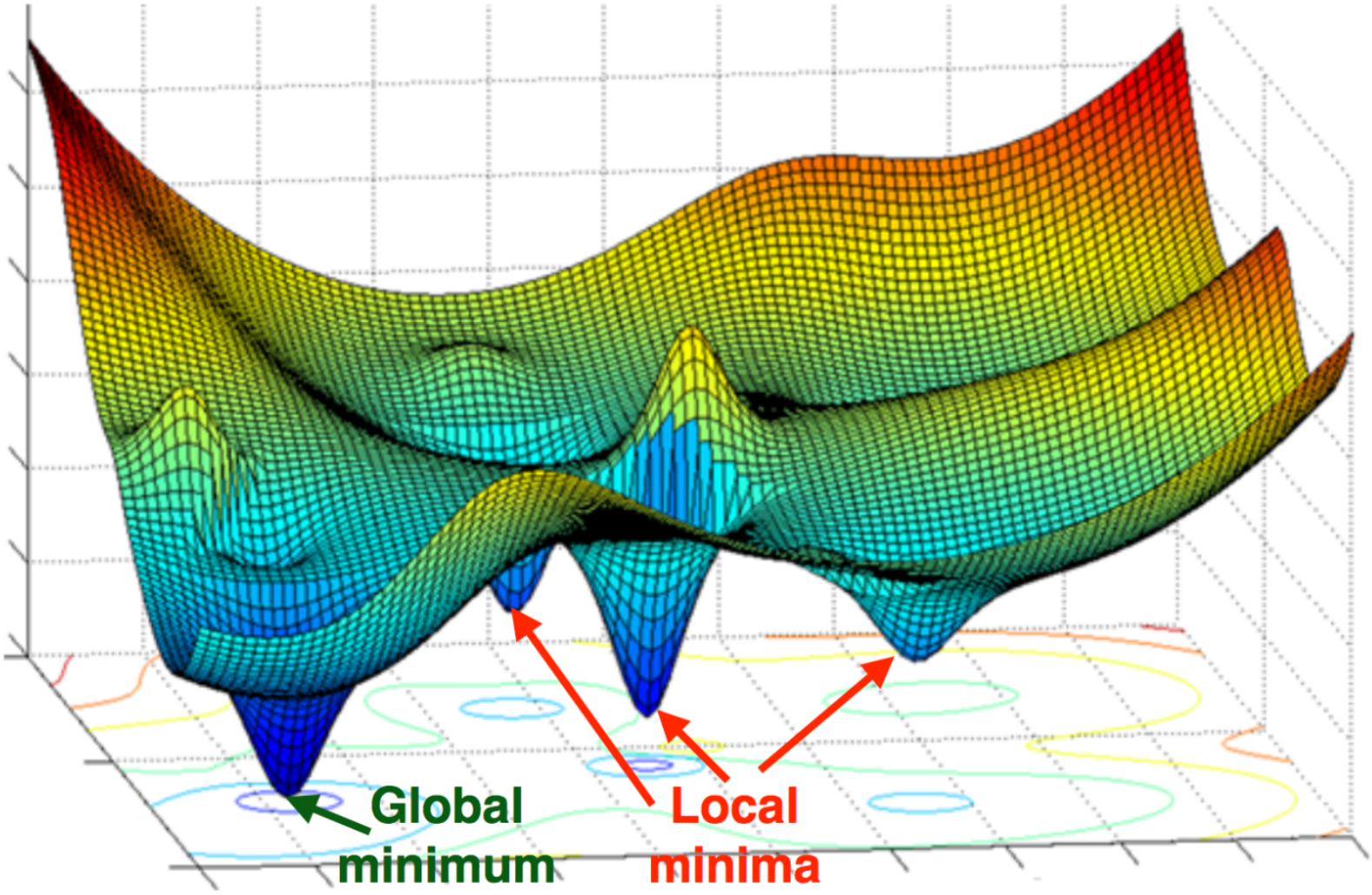
$$=\left(Y-\left(X^T W^{(2)}\right)W^{(3)}\right)X^T W_j^{(2)}=\textit{error}\times\textit{culpability}$$

$$-\frac{\partial}{\partial W_{ij}^{(2)}}\frac{1}{2}\left(Y-\left(X^T W^{(2)}\right)W^{(3)}\right)^2=-\frac{\partial}{\partial W^{(2)}}g_3\left(g_2\left(g_1\left(W^{(2)}\right)\right)\right)$$

$$=-\frac{\partial}{\partial g_2\left(g_1\left(W^{(2)}\right)\right)}\frac{\partial g_2\left(g_1\left(W^{(2)}\right)\right)}{\partial g_1\left(W^{(2)}\right)}\frac{\partial g_1\left(W^{(2)}\right)}{\partial W_{ij}^{(2)}}g_3\left(g_2\left(g_1\left(W^{(2)}\right)\right)\right)$$

$$=\left(Y-\left(X^T W^{(2)}\right)W^{(3)}\right)W_j^{(3)}X_i=\textit{error}\times\textit{culpability}$$

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

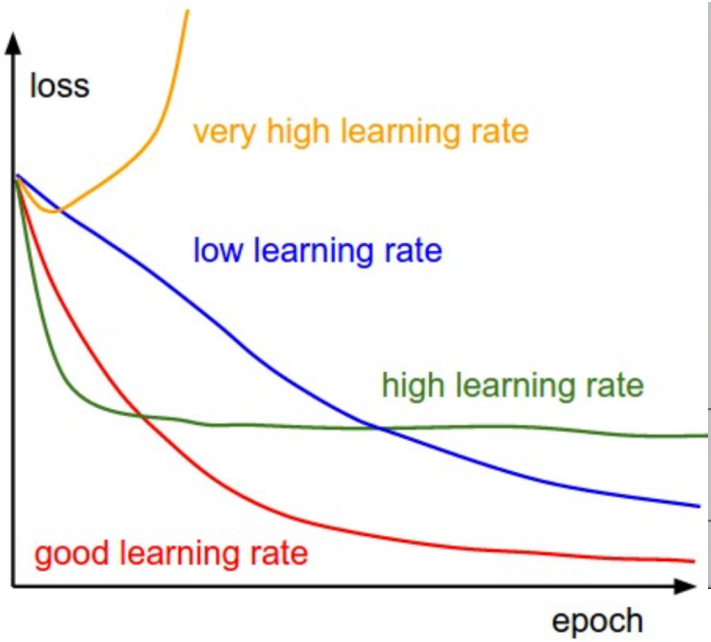
Sadly, backpropagation (GD) is very weak on DNN's...

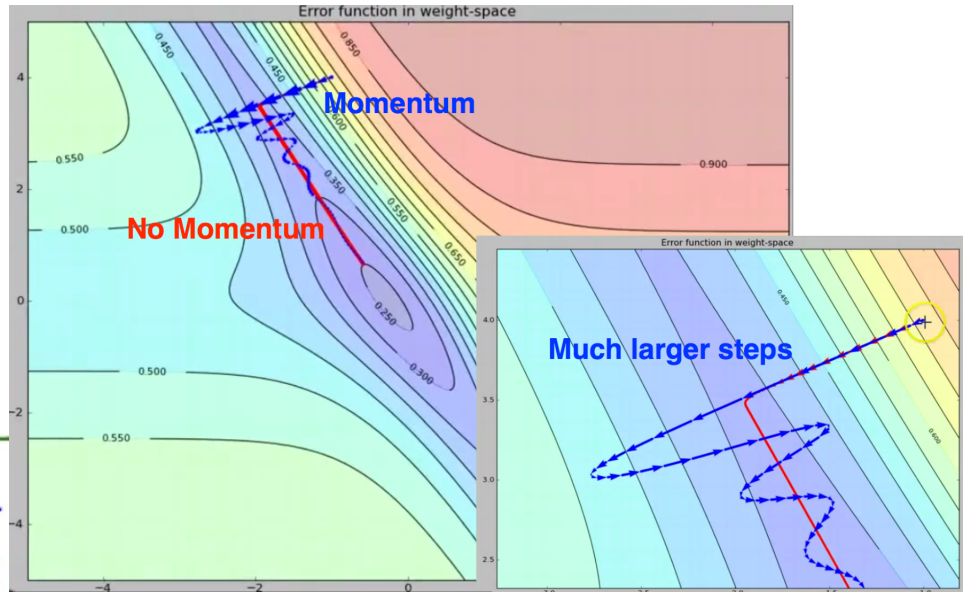
- If you're using backpropagation anyway, then probably
- Stochastic gradient decent, or mini-batch
- Many epochs (iterations through the data)

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 that 3 hidden layers

Unfortunately backpropagation is poor on DNN's...

- lots of trial and error – no design consensus
- but usually less that 3 hidden layers
- 2*n neurons (noisy data) to 30*n neurons (perfect signal)

Unfortunately backpropagation is poor on DNN's...

- lots of trial and error – no design consensus
- but usually less that 3 hidden layers
- 2*n neurons (noisy data) to 30*n 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 that 3 hidden layers
- 2*n neurons (noisy data) to 30*n 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 that 3 hidden layers
- 2*n neurons (noisy data) to 30*n 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`

Unfortunately backpropagation is poor on DNN's...

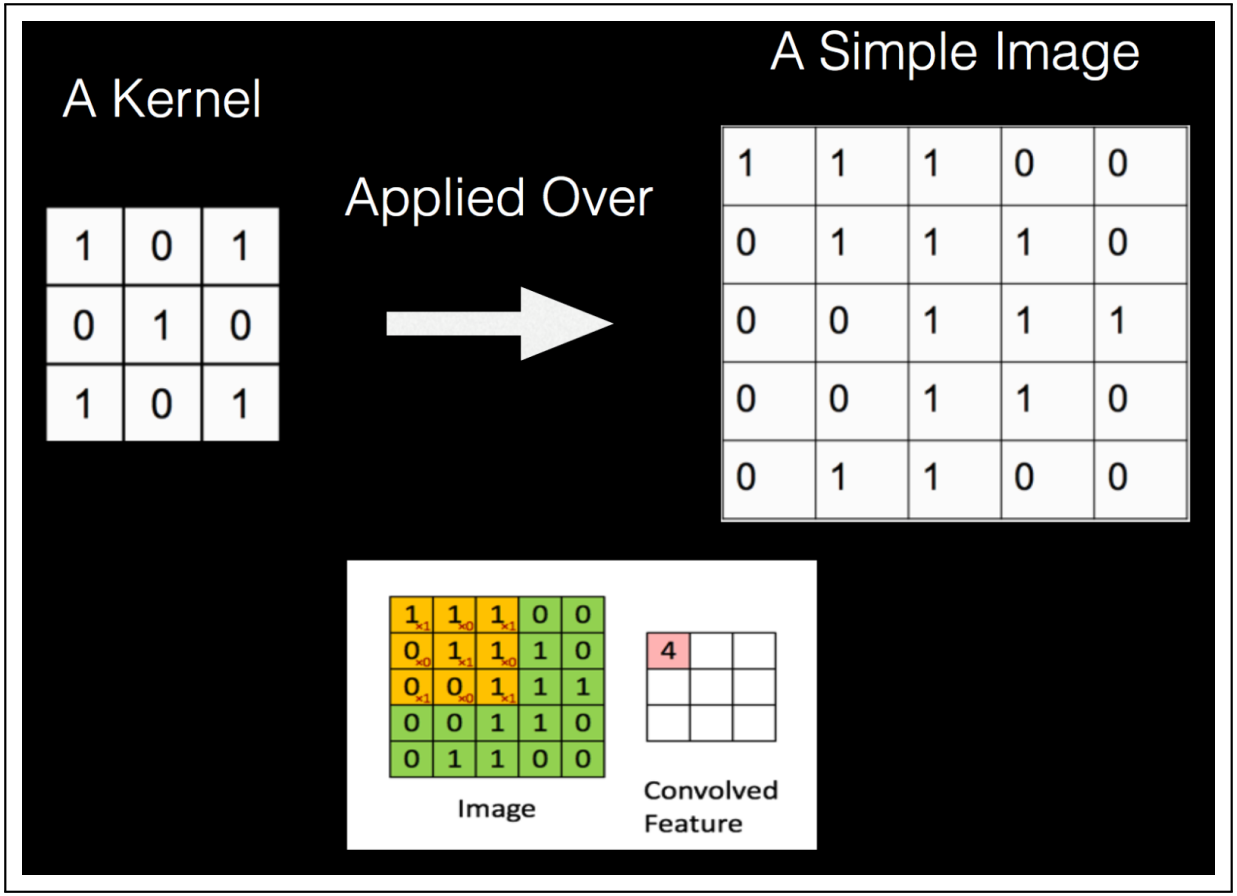
- lots of trial and error – no design consensus
- but usually less that 3 hidden layers
- 2*n neurons (noisy data) to 30*n neurons (perfect signal)
- fewer outputs than inputs per layer
- probably rectified linear, perhaps tanh & *maybe* sigmoid AF's
- and tricks:

`research.microsoft.com/pubs/192769/tricks-2012.pdf`

- 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

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

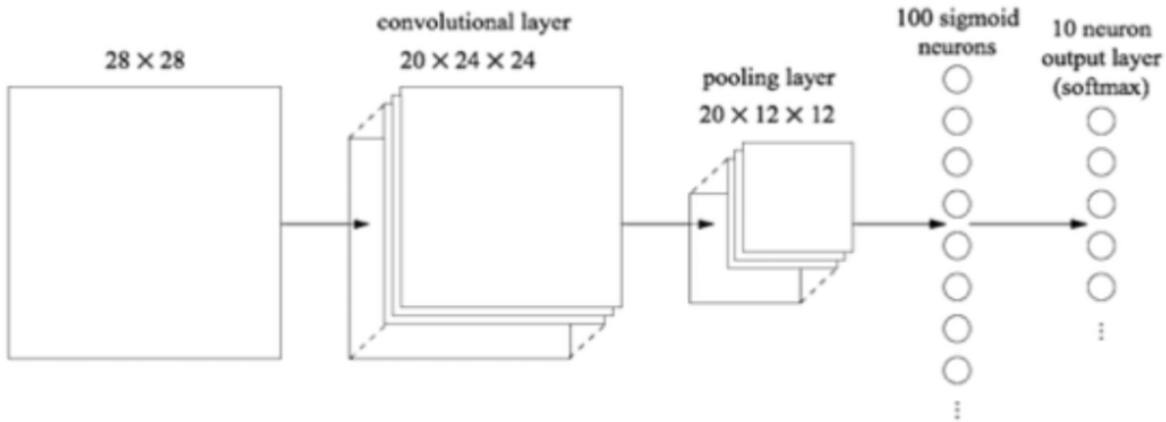


Returning to Convolutions...

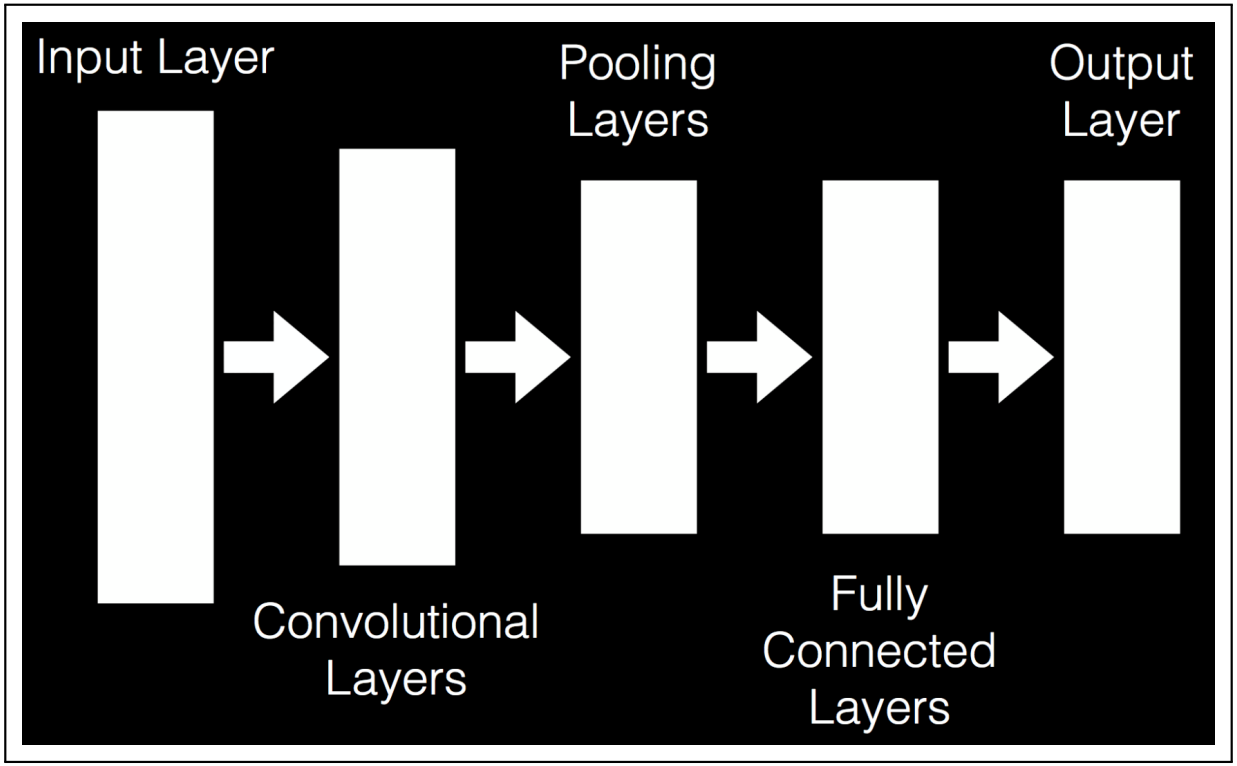
Can NN's learn their own convolution matrices?

idea: *denovo* kernels

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

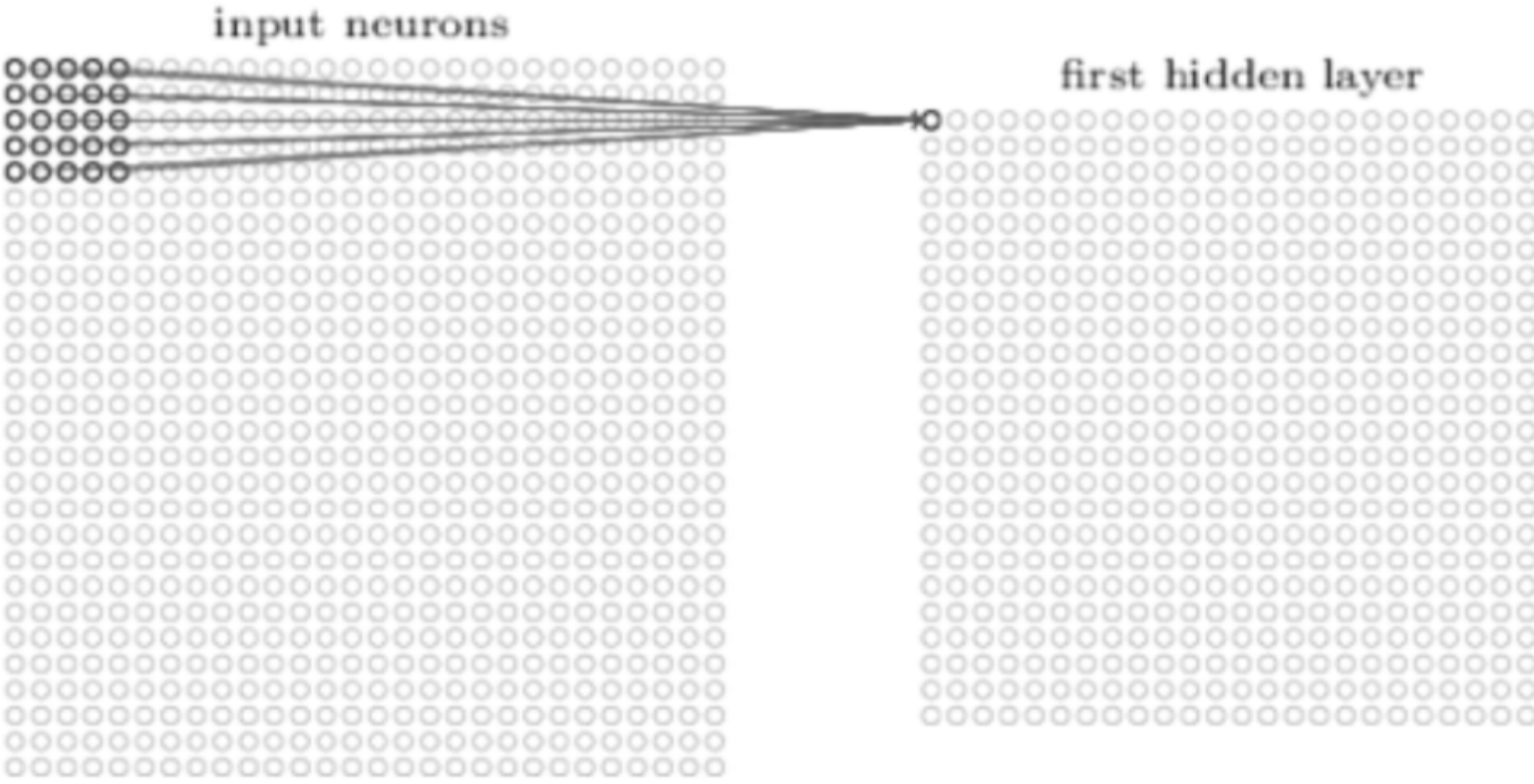


Convolutional Neural Network's – CovNets, or CNN's



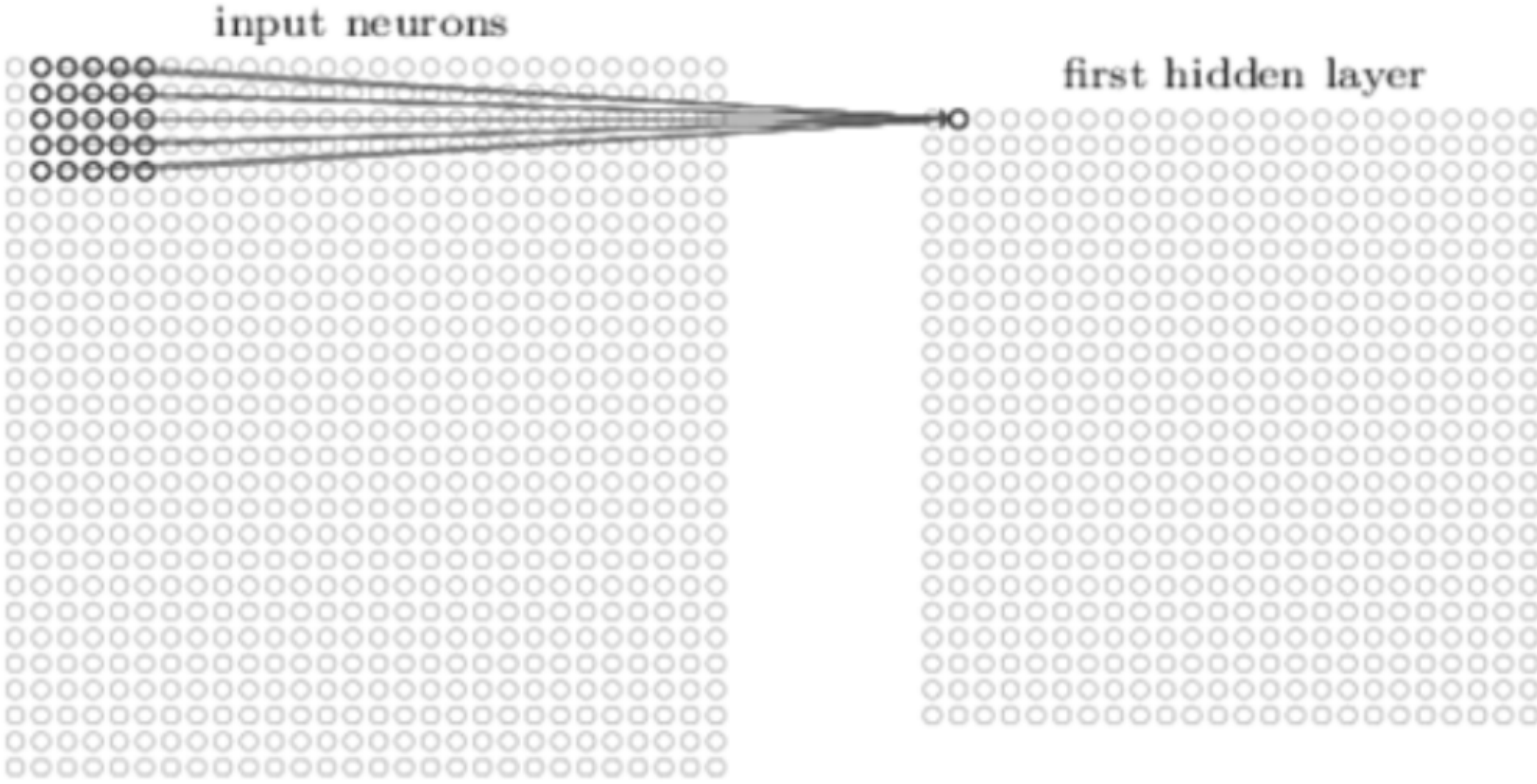
There are three key features that make this NN structure actually work: (1) local receptive fields, (2) shared weights, and (3) pooling

1. Local Receptive Field



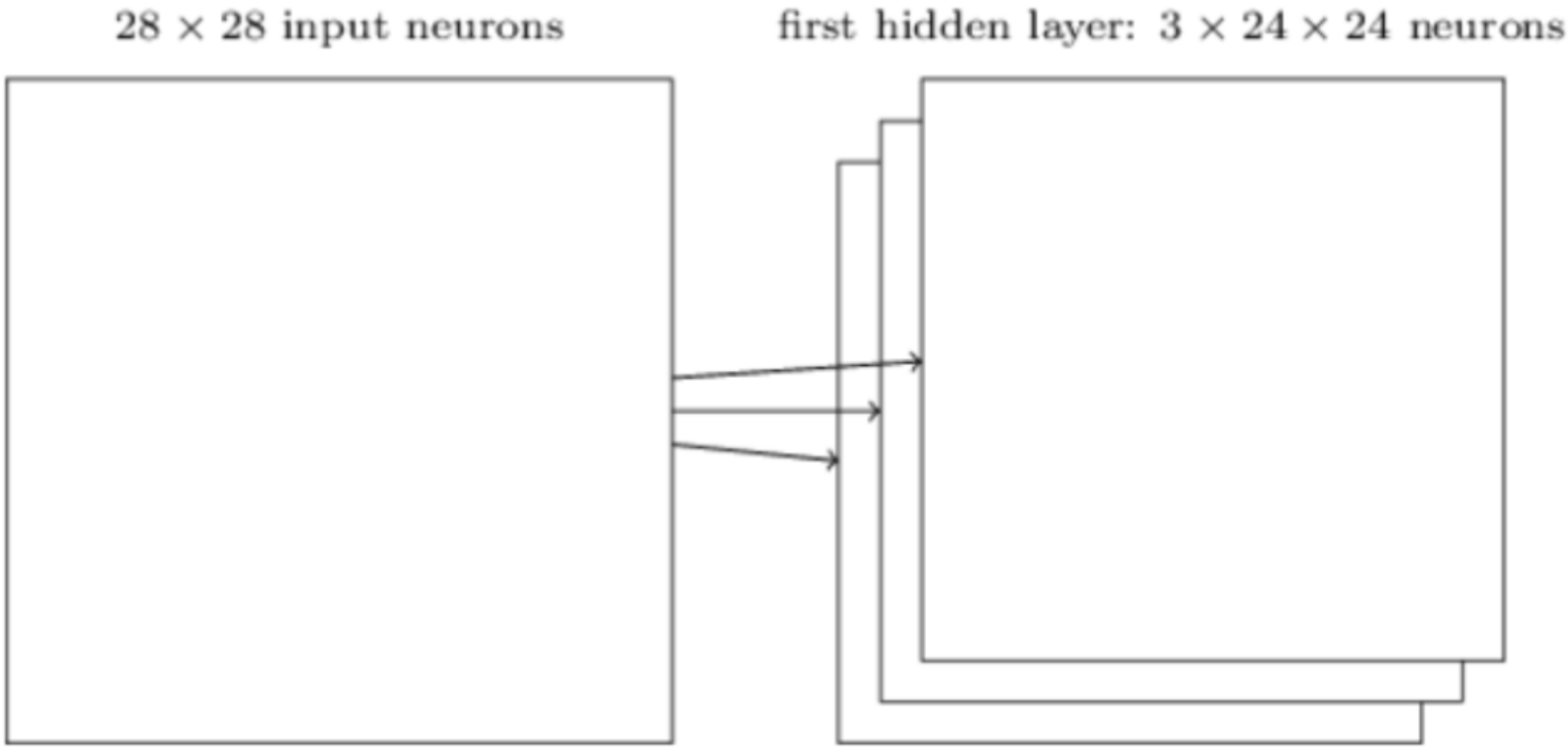
- A group of pixes are a *local receptive field*
- Defined by the size of the kernel
- The image is transformed into the set of local receptive fields

1. Local Receptive Field



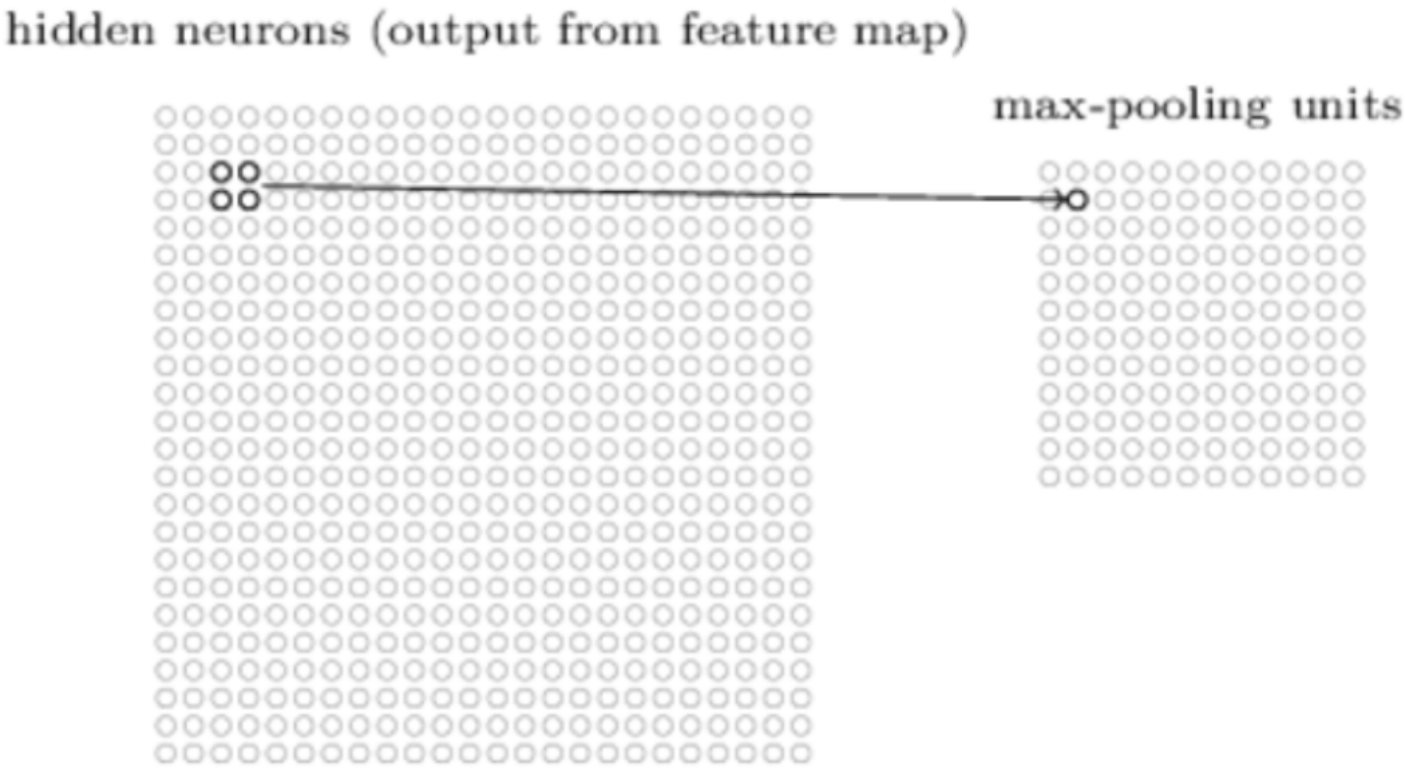
- A group of pixes are a *local receptive field*
- Defined by the size of the kernel
- The image is transformed into the set of local receptive fields

2. Shared Weights



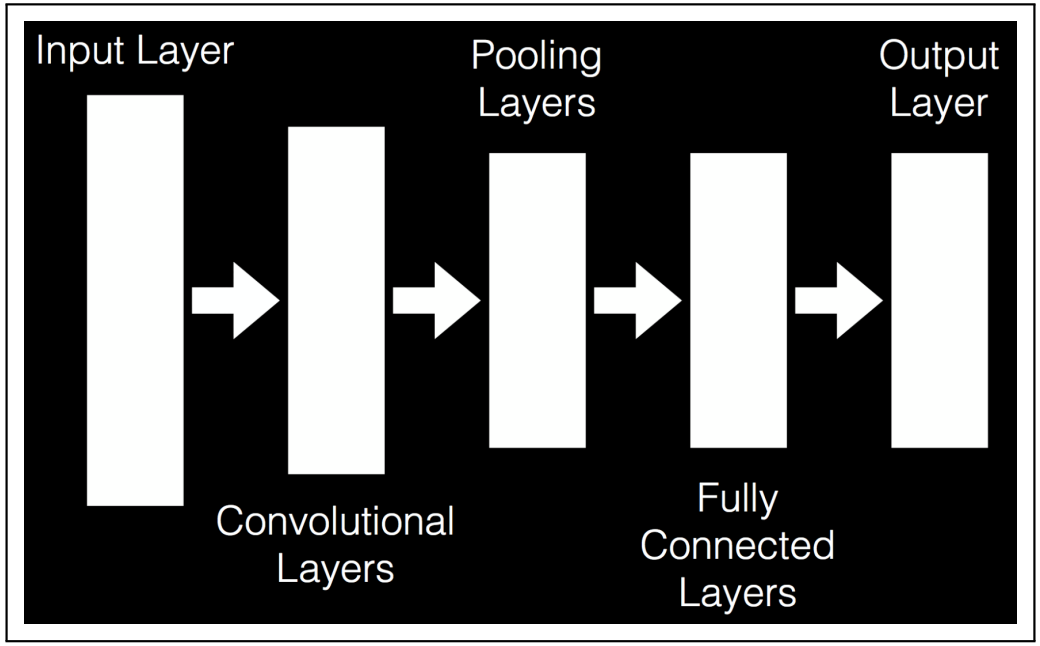
- Multiple convolutions are learned/used
- Weights within a convolution are (obviously) shared

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

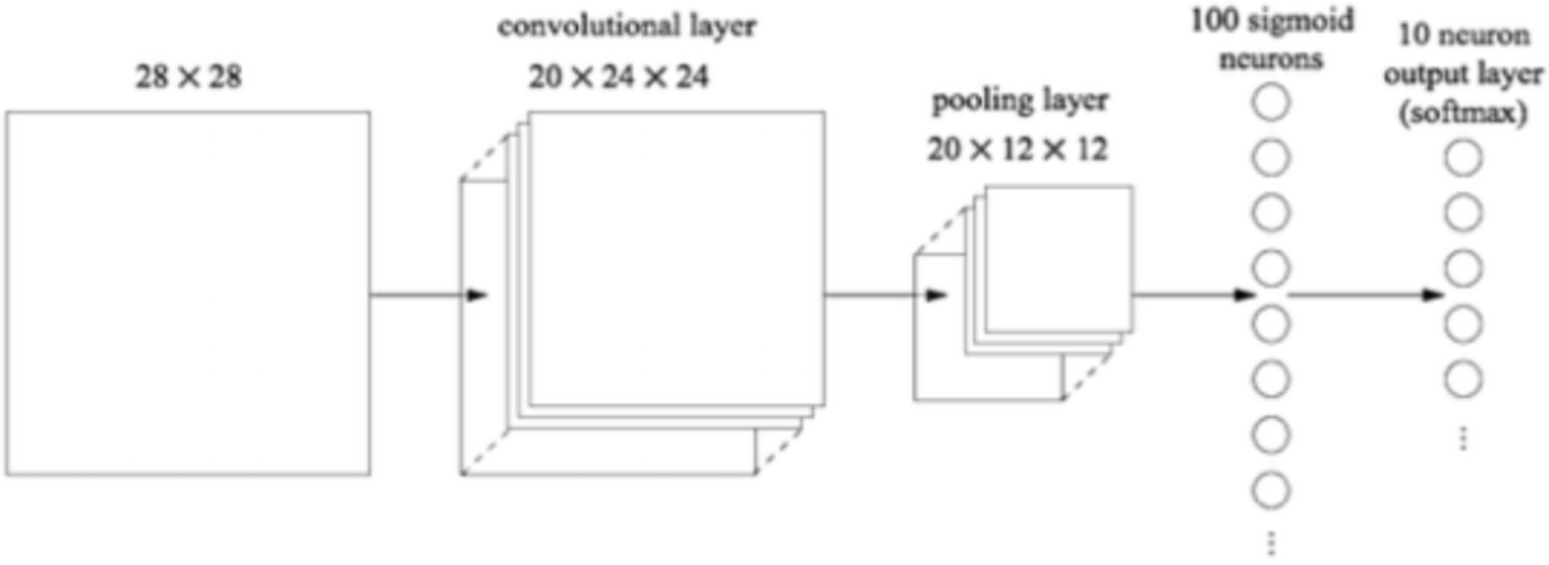
Fully Connected and Output Layers



- Fully connected layers hierarchically aggregate learned features – they produces higher order features in standard NN manner
- 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

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

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
[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*...