

# Data Mining II

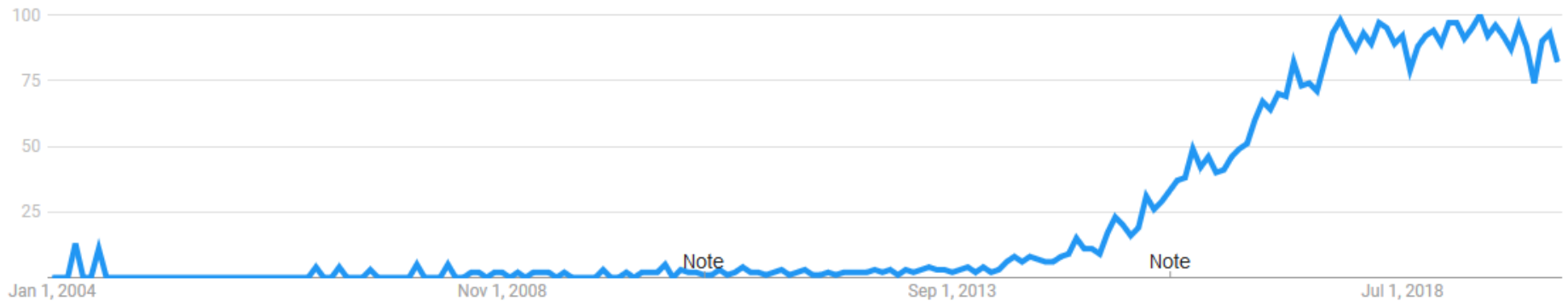
## Neural Networks and Deep Learning



# Deep Learning

- A recent hype topic

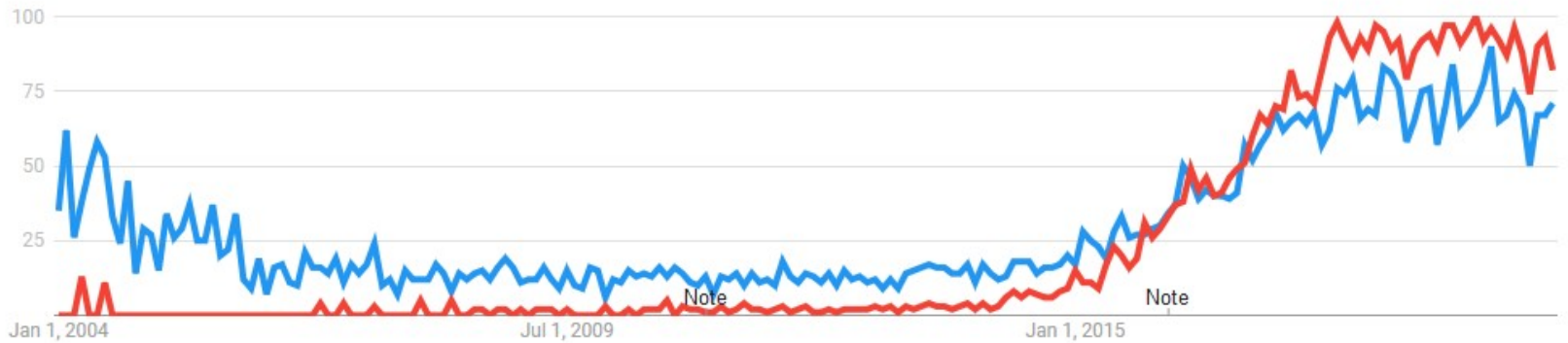
Interest over time 



# Deep Learning

- Just the same as artificial neural networks with a new buzzword?

Interest over time ?



# Deep Learning

- Contents of this Lecture
  - Recap of neural networks
  - The backpropagation algorithm
  - Auto Encoders
  - Deep Learning
  - Network Architectures
  - “Anything2Vec”

# Revisited Example: Credit Rating

- Consider the following example:
  - and try to build a model
  - which is as small as possible (recall: Occam's Razor)

Person	Employed	Owns House	Balanced Account	Get Credit
Peter Smith	yes	yes	no	yes
Julia Miller	no	yes	no	no
Stephen Baker	yes	no	yes	yes
Mary Fisher	no	no	yes	no
Kim Hanson	no	yes	yes	yes
John Page	yes	no	no	no

# Revisited Example: Credit Rating

- Smallest model:
  - if at least two of Employed, Owns House, and Balanced Account are yes  
→ Get Credit is yes
- Not nicely expressible in trees and rule sets
  - as we know them (attribute-value conditions)

Person	Employed	Owns House	Balanced Account	Get Credit
Peter Smith	yes	yes	no	yes
Julia Miller	no	yes	no	no
Stephen Baker	yes	no	yes	yes
Mary Fisher	no	no	yes	no
Kim Hanson	no	yes	yes	yes
John Page	yes	no	no	no

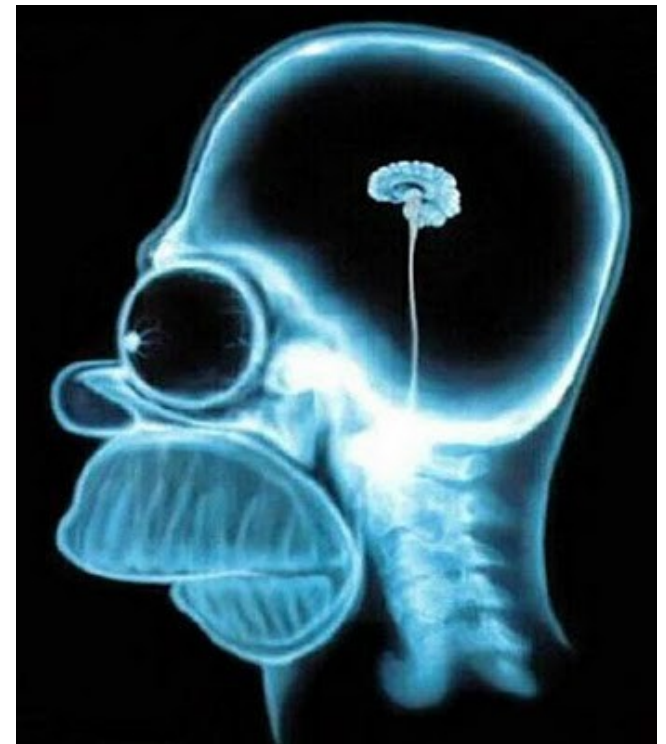
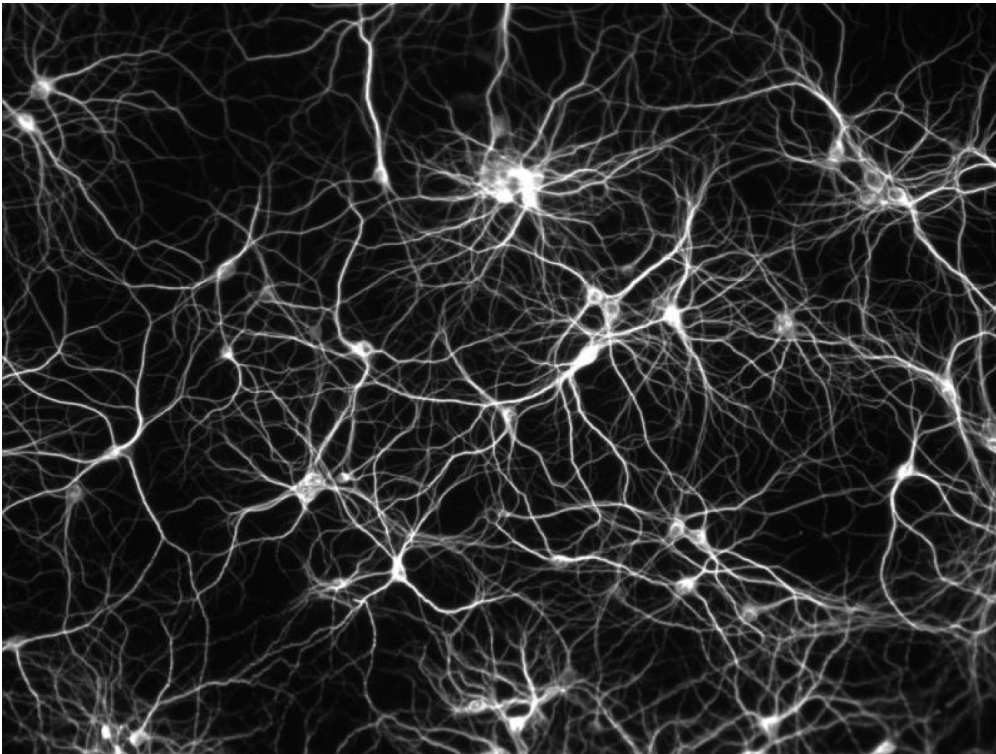
# Revisited Example: Credit Rating

- Smallest model:
  - if at least two of Employed, Owns House, and Balance Account are yes  
→ Get Credit is yes
- As rule set:
  - Employed=yes and OwnsHouse=yes => yes
  - Employed=yes and BalanceAccount=yes => yes
  - OwnsHouse=yes and BalanceAccount=yes => yes
  - => no
- General case:
  - at least m out of n attributes need to be yes => yes
  - this requires  $\binom{n}{m}$  rules, i.e.,  $\frac{n!}{m! \cdot (n-m)!}$
  - e.g., “5 out of 10 attributes need to be yes”  
requires more than 15,000 rules!



# Artificial Neural Networks

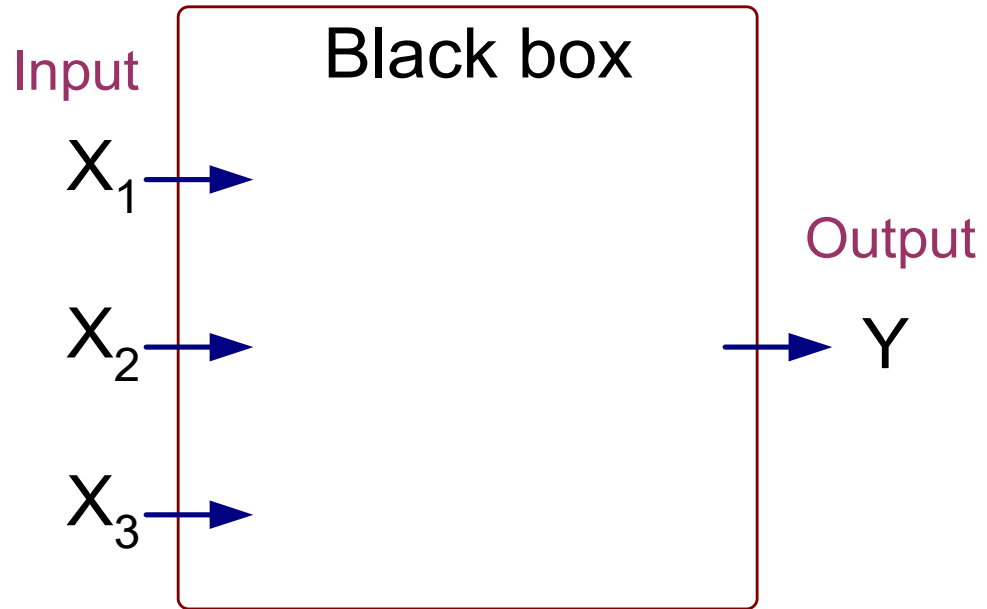
- Inspiration
  - one of the most powerful super computers in the world





# Artificial Neural Networks (ANN)

$X_1$	$X_2$	$X_3$	Y
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1
0	0	1	0
0	1	0	0
0	1	1	1
0	0	0	0



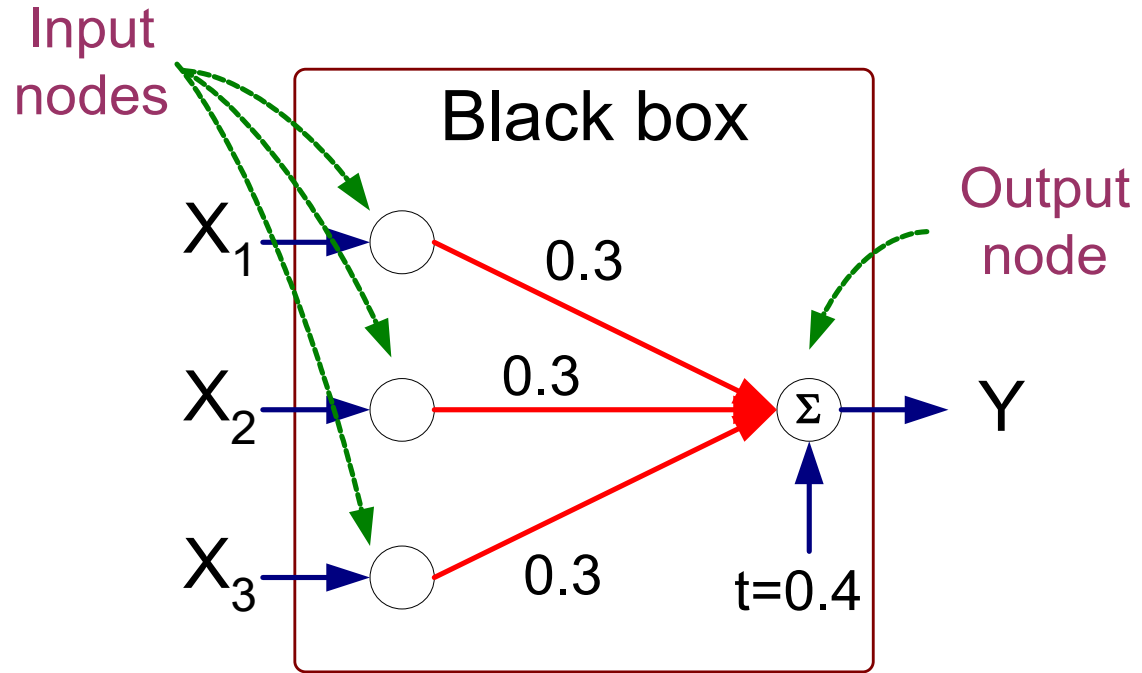
Output  $Y$  is 1 if at least two of the three inputs are equal to 1.

# Example: Credit Rating

- Smallest model:
  - if at least two of Employed, Owns House, and Balance Account are yes  
→ Get Credit is yes
- Given that we represent yes and no by 1 and 0, we want
  - if  $(\text{Employed} + \text{Owns House} + \text{Balance Account}) > 1.5$   
→ Get Credit is yes

# Artificial Neural Networks (ANN)

$X_1$	$X_2$	$X_3$	Y
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1
0	0	1	0
0	1	0	0
0	1	1	1
0	0	0	0

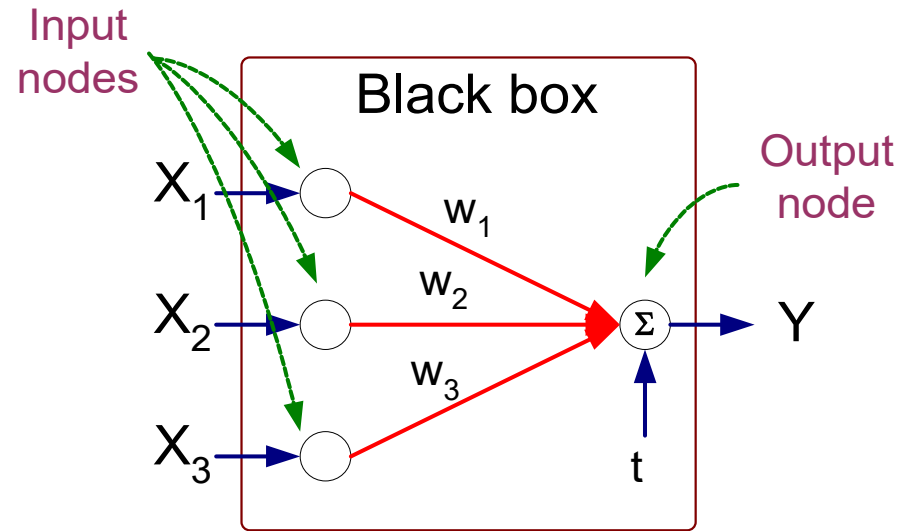


$$Y = I(0.3X_1 + 0.3X_2 + 0.3X_3 - 0.4 > 0)$$

$$\text{where } I(z) = \begin{cases} 1 & \text{if } z \text{ is true} \\ 0 & \text{otherwise} \end{cases}$$

# Artificial Neural Networks (ANN)

- Model is an assembly of inter-connected nodes and weighted links
- Output node sums up each of its input value according to the weights of its links
- Compare output node against some threshold  $t$

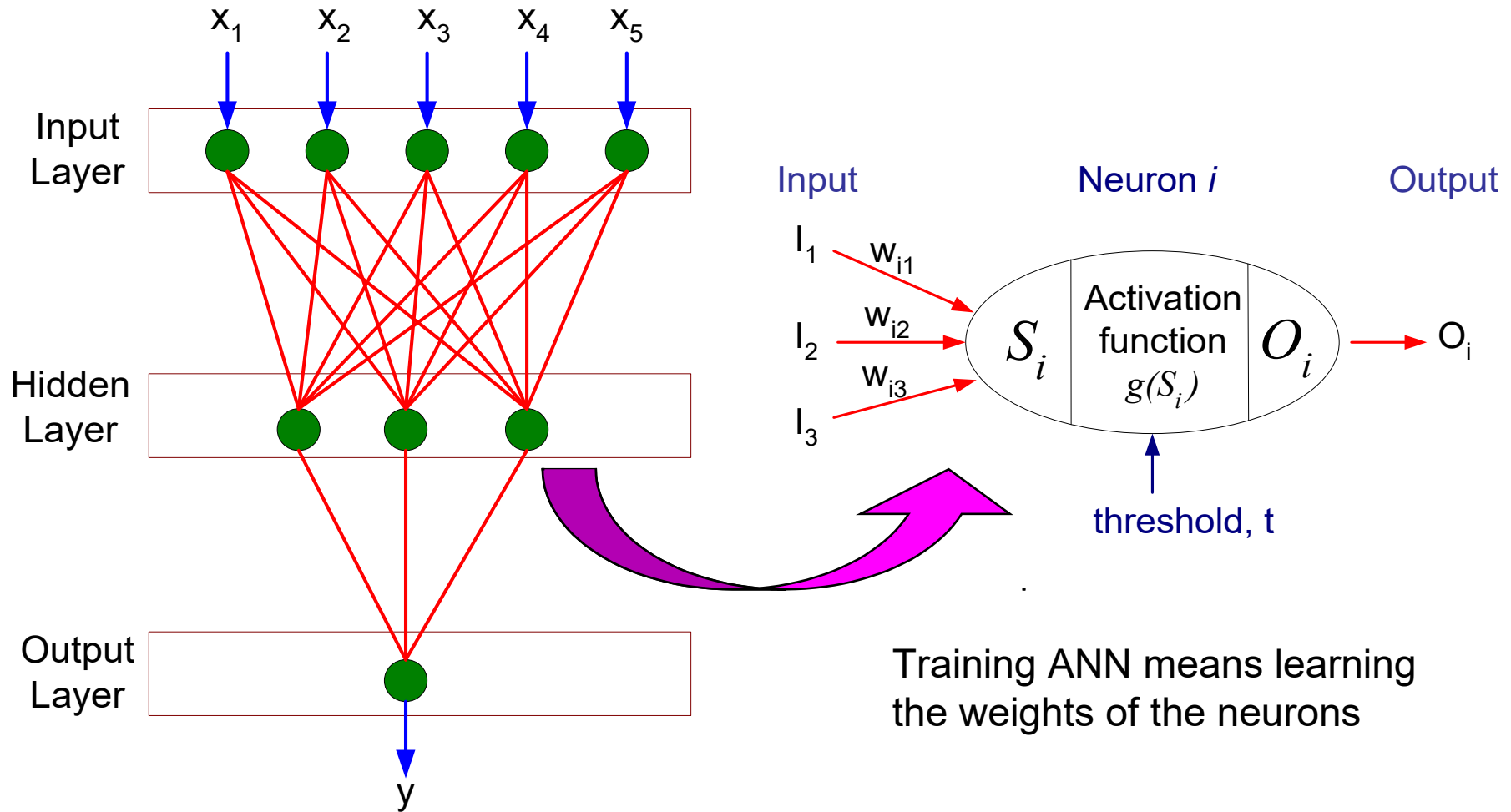


## Perceptron Model

$$Y = I\left(\sum_i w_i X_i - t\right) \quad \text{or}$$

$$Y = \text{sign}\left(\sum_i w_i X_i - t\right)$$

# General Structure of ANN



# Algorithm for Learning ANN

- Initialize the weights ( $w_0, w_1, \dots, w_k$ ), e.g., usually randomly
- Adjust the weights in such a way that the output of ANN is consistent with class labels of training examples
  - Objective function: 
$$E = \sum_i [Y_i - f(w_i, X_i)]^2$$
  - Find the weights  $w_i$ 's that minimize the above objective function

# Backpropagation Algorithm

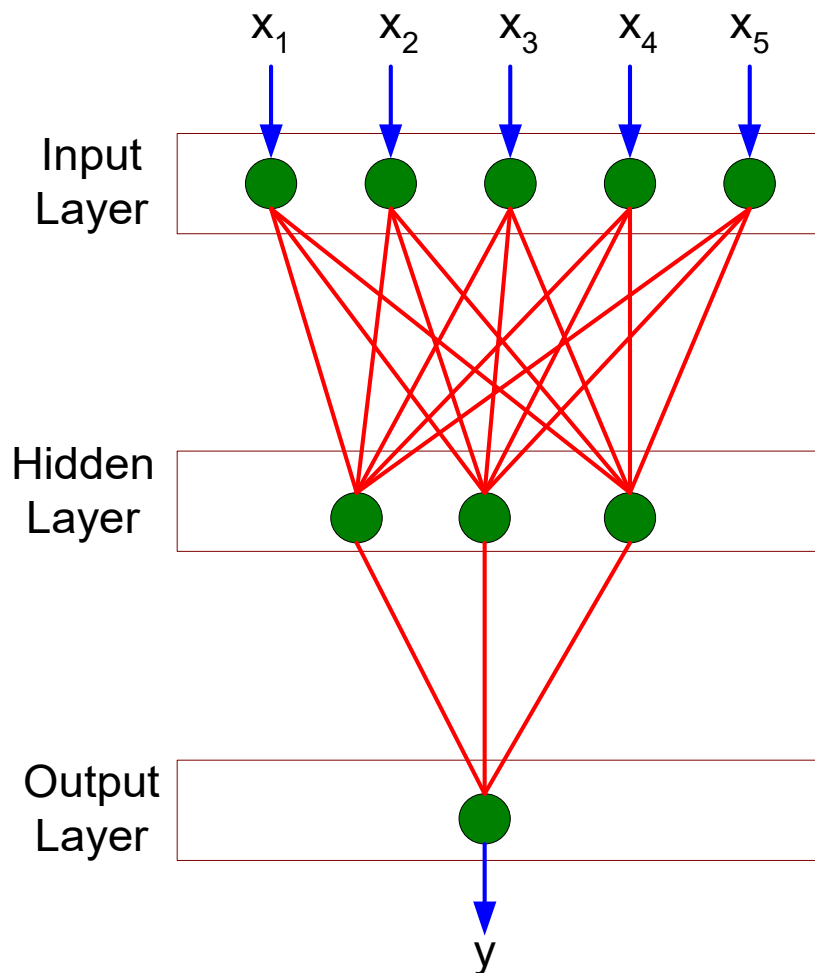
- Adjust the weights in such a way that the output of ANN is consistent with class labels of training examples

- Objective function:

$$E = \sum_i [Y_i - f(w_i, X_i)]^2$$

- Find the weights  $w_i$ 's that minimize the above objective function

- This is simple for a single layer perceptron
- But for a multi-layer network,  $Y_i$  is not known



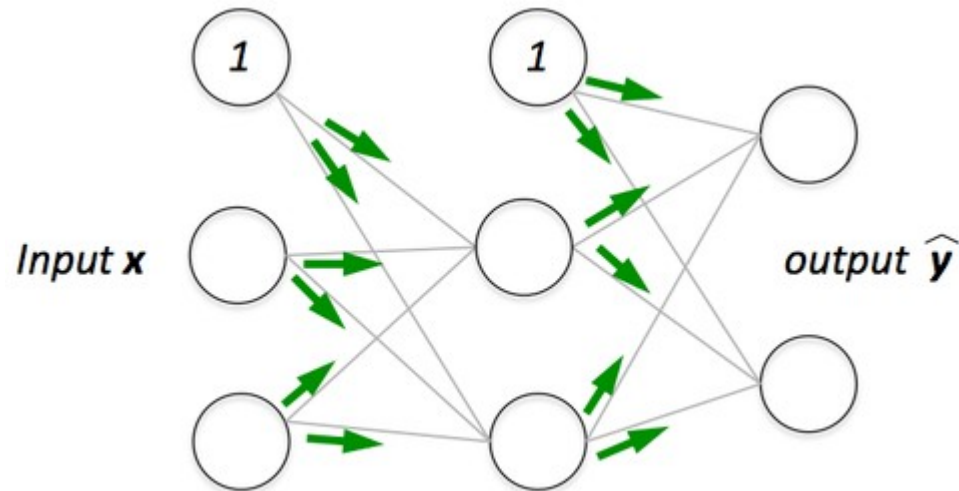


# Backpropagation Algorithm

- Sketch of the Backpropagation Algorithm:
  - Present an example to the ANN
  - Compute error at the output layer
  - Distribute error to hidden layer according to weights
    - i.e., the error is distributed according to the contribution of the previous neurons to the result
  - Adjust weights so that the error is minimized
    - Adjustment factor: learning rate
    - Use gradient descent
  - Repeat until input layer is reached

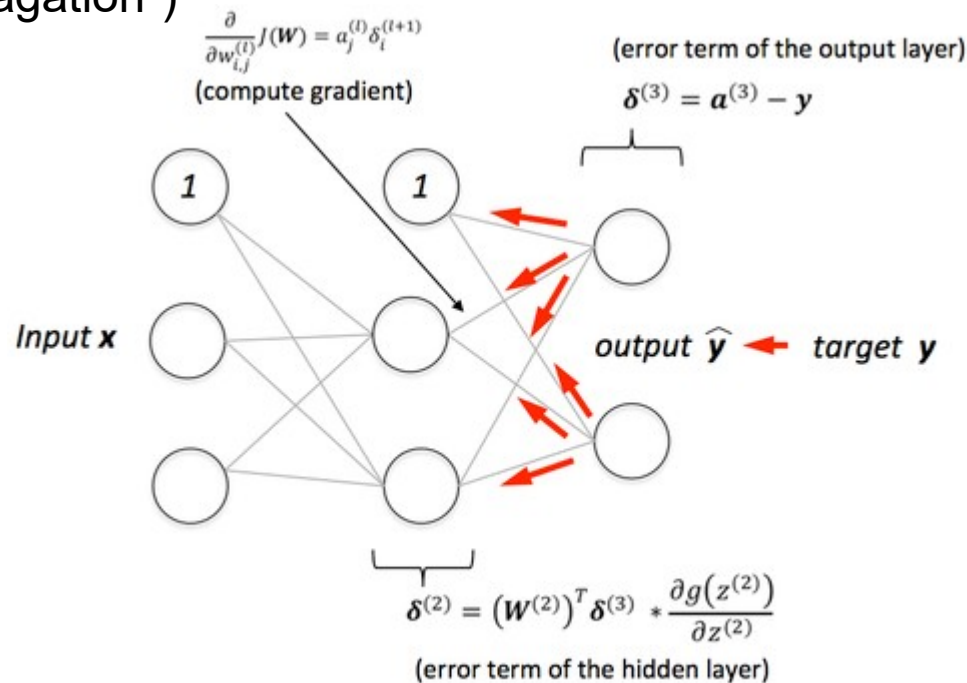
# Backpropagation Algorithm

- Important notions:
  - Predictions are pushed forward through the network (“feed-forward neural network”)
  - Errors are pushed backwards through the network (“backpropagation”)



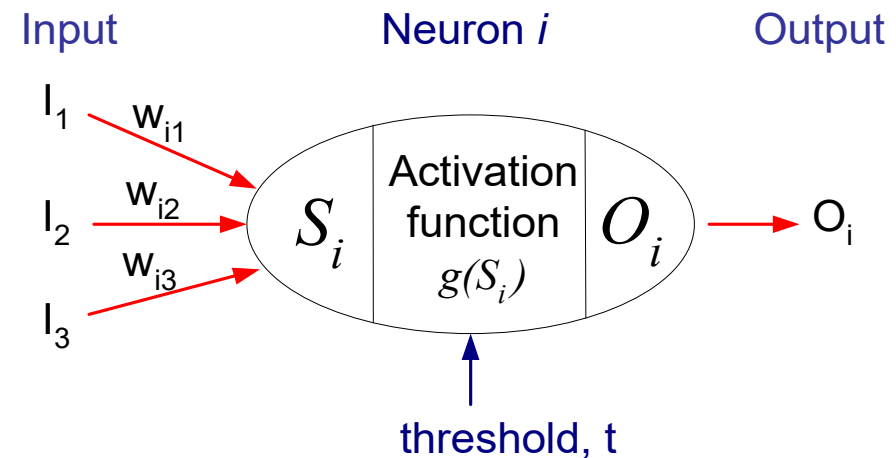
# Backpropagation Algorithm

- Important notions:
  - Predictions are pushed forward through the network (“feed-forward neural network”)
  - Errors are pushed backwards through the network (“backpropagation”)



# Backpropagation Algorithm – Gradient Descent

- Output of a neuron:  $o = g(w_1 i_1 \dots w_n i_n)$
- Assume the desired output is  $y$ , the error is
$$o - y = g(w_1 i_1 \dots w_n i_n) - y$$
- We want to minimize the error, i.e., minimize
$$g(w_1 i_1 \dots w_n i_n) - y$$
- We follow the steepest descent of  $g$ , i.e.,
  - the value where  $g'$  is maximal



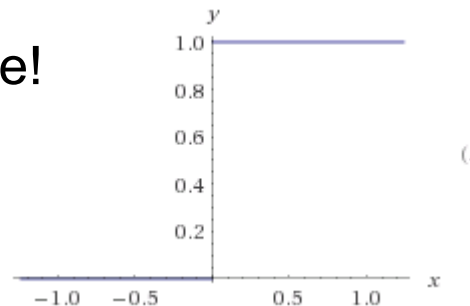
# Backpropagation Algorithm – Gradient Descent

- Hey, wait...
  - the value where  $g'$  is maximal
- To find the steepest gradient, we have to differentiate the activation function

$$Y = I(0.3X_1 + 0.3X_2 + 0.3X_3 - 0.4 > 0)$$

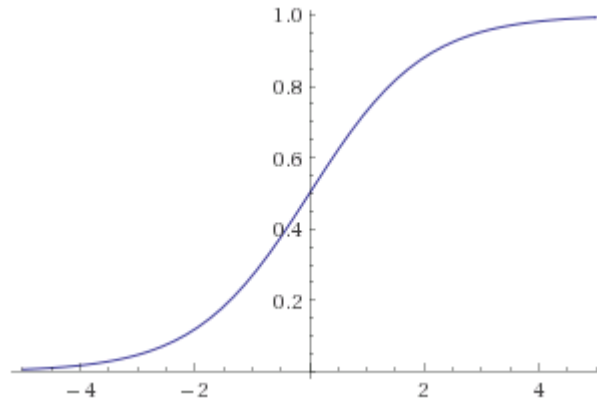
$$\text{where } I(z) = \begin{cases} 1 & \text{if } z \text{ is true} \\ 0 & \text{otherwise} \end{cases}$$

- But  $I(z)$  is not differentiable!

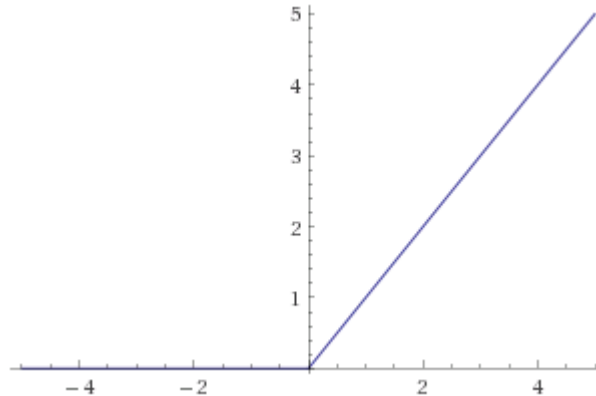


# Alternative Differentiable Activation Functions

- Sigmoid Function (classic ANNs):  $1/(1+e^{(-x)})$

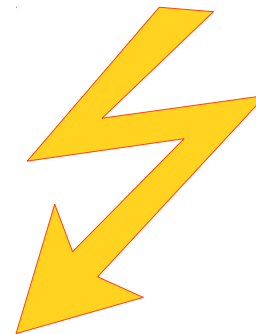


- Rectified Linear Unit (ReLU, since 2010s):  $\max(0,x)$



# Properties of ANNs and Backpropagation

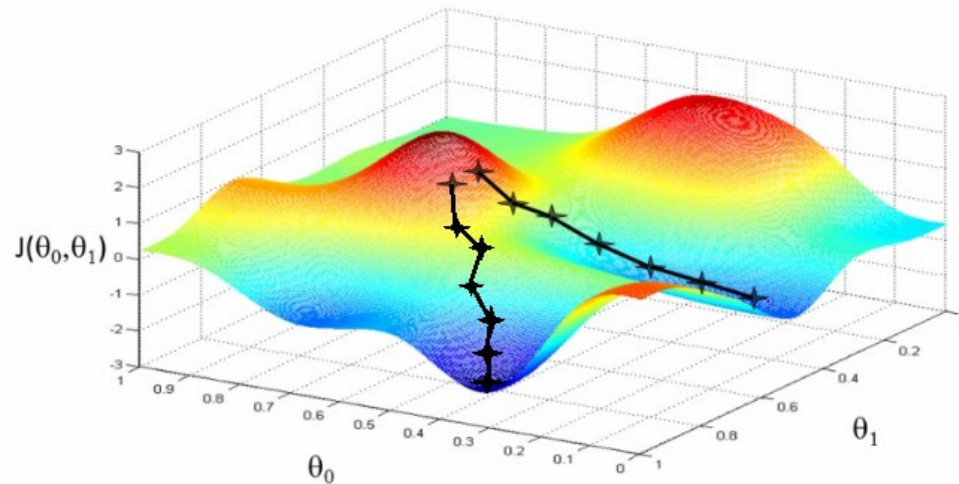
- Non-linear activation function:
  - May approximate any arbitrary function, even with one hidden layer
- Convergence:
  - Convergence may take time
  - Higher learning rate: faster convergence
- Gradient Descent Strategy:
  - Danger of ending in local optima
    - Use momentum to prevent getting stuck
  - Lower learning rate: higher probability of finding global optimum





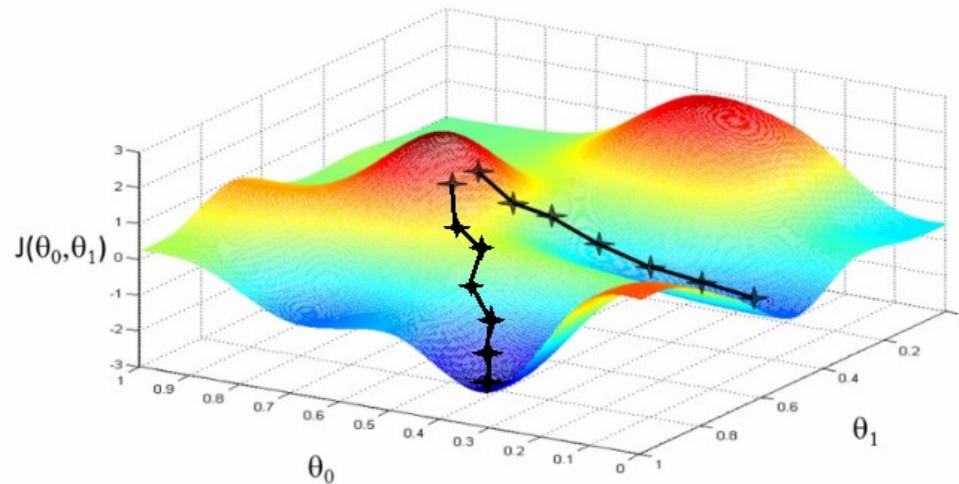
# Learning Rate, Momentum, and Local Minima

- Learning rate: how much do we adapt the weights with each step
  - 0: no adaptation, use previous weight
  - 1: forget everything we have learned so far, simply use weights that are best for current example
- Smaller: slow convergence, less overfitting
- Higher: faster convergence, more overfitting



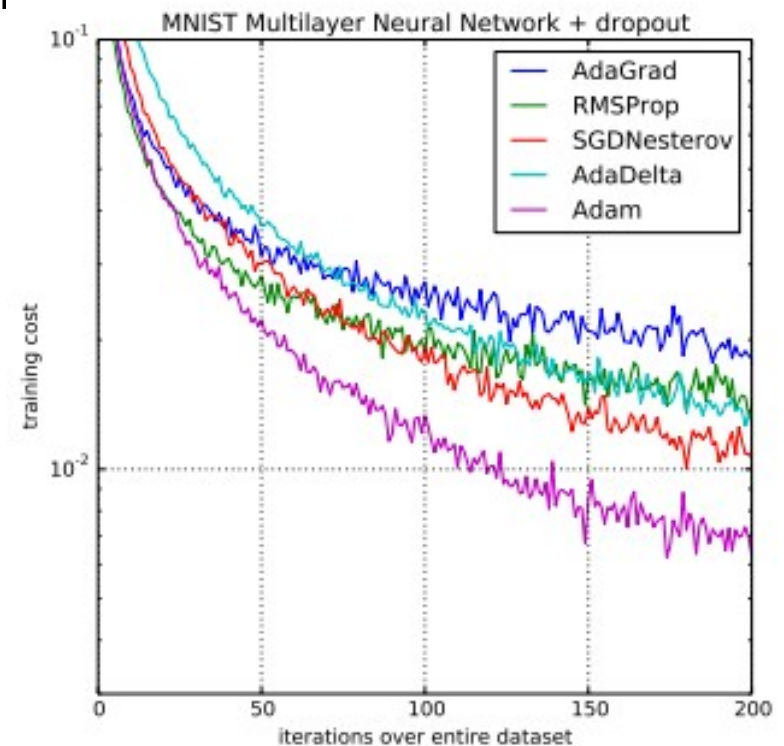
# Learning Rate, Momentum, and Local Minima

- Momentum: how much do we adapt the weights
  - Small: very small steps
  - High: very large steps
- Smaller: better convergence, sticks in local minimum
- Higher: worse convergence, does not get stuck



# Dynamic Learning Rates

- Adapting learning rates over time
  - Search coarse-grained first, fine-grained later
  - e.g., allow bigger jumps in the beginning
- e.g., RMSProp (Hinton, 2014)
  - use decay function for learning rate
- e.g., AdaDelta (Zeiler, 2012)
  - restrict total update for features over windows of time

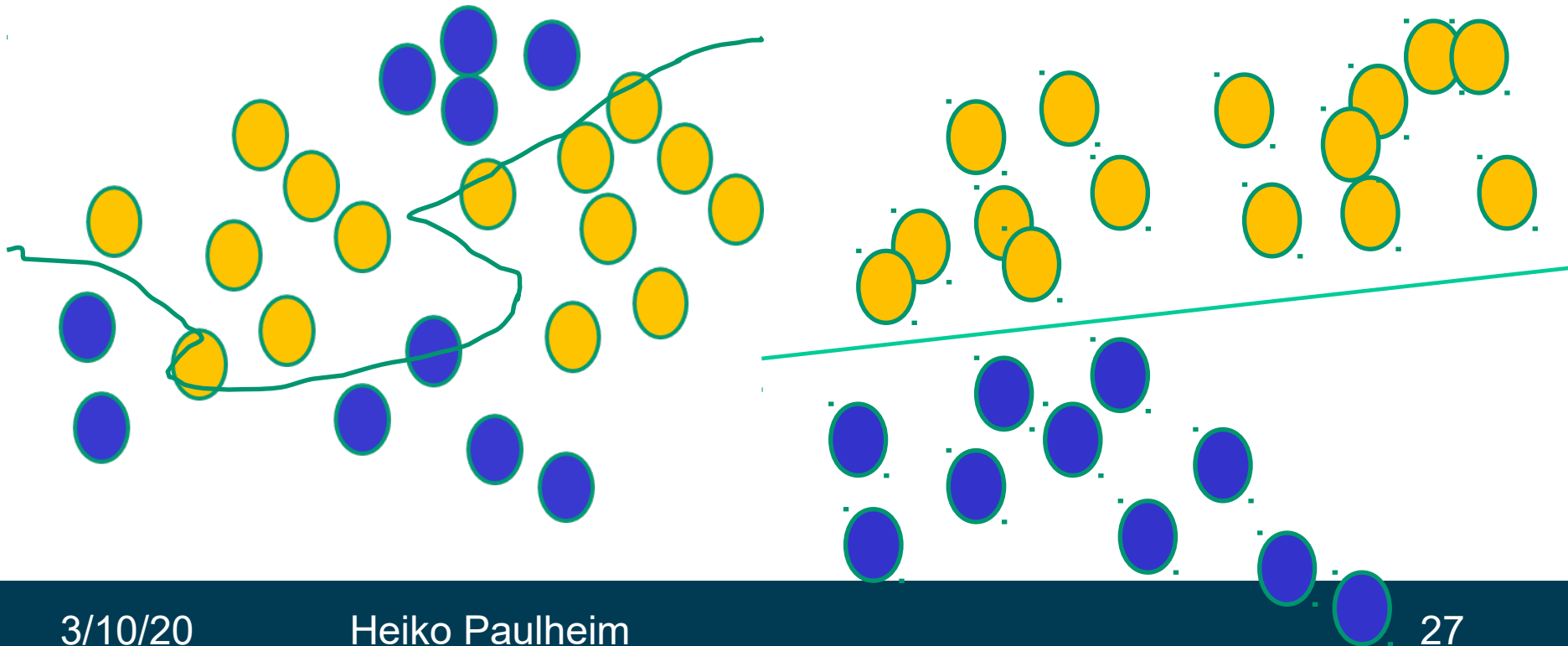


# Local Learning Rates

- Observation
  - not all parameters change equally often
  - e.g., text classification: input neuron weights for infrequent words
- AdaGrad (Duchi et al., 2011)
  - maintain list of gradient changes for each parameter
  - adapt learning rates locally
- AdaDelta (Zeiler, 2012)
  - restrict total updates per parameter
- Bottom line: optimization functions often have a large impact
  - Reading recommendation: <https://ruder.io/optimizing-gradient-descent/>

# ANNs vs. SVMs

- ANNs have arbitrary decision boundaries
  - and keep the data as it is
- SVMs have linear decision boundaries
  - and transform the data first

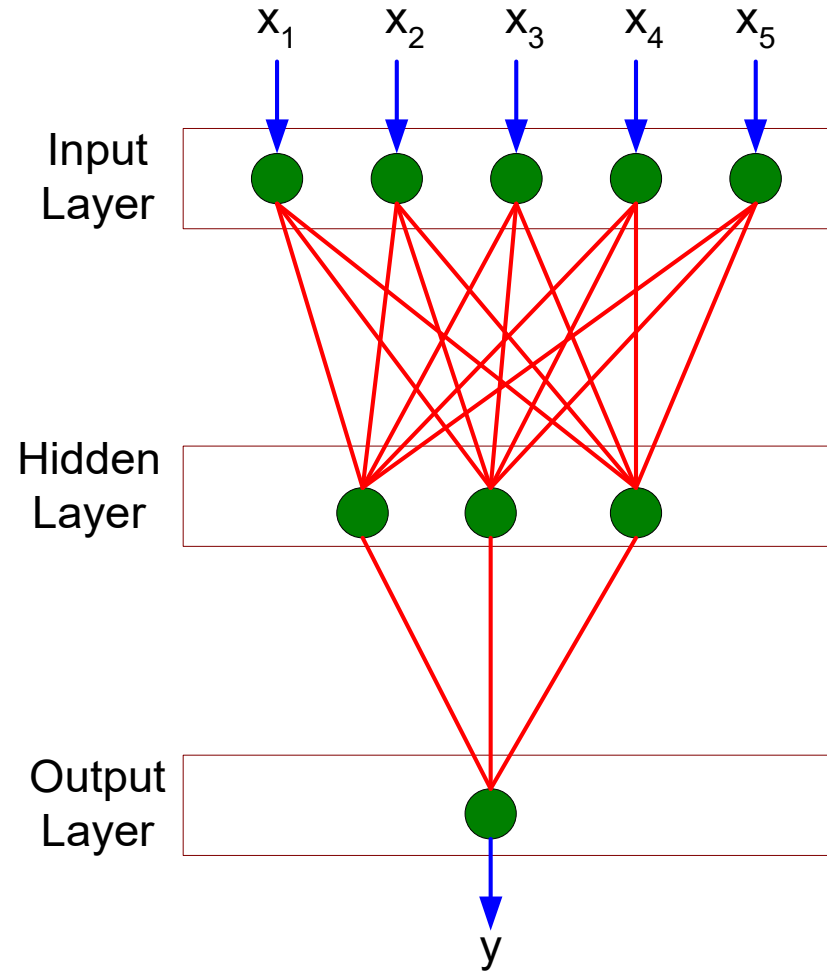


# Recap: Feature Subset Selection & PCA

- Idea: reduce the dimensionality of high dimensional data
- Feature Subset Selection
  - Focus on relevant attributes
- PCA
  - Create new attributes
- In both cases
  - We assume that the data can be described with fewer variables
  - Without losing much information

# What Happens at the Hidden Layer?

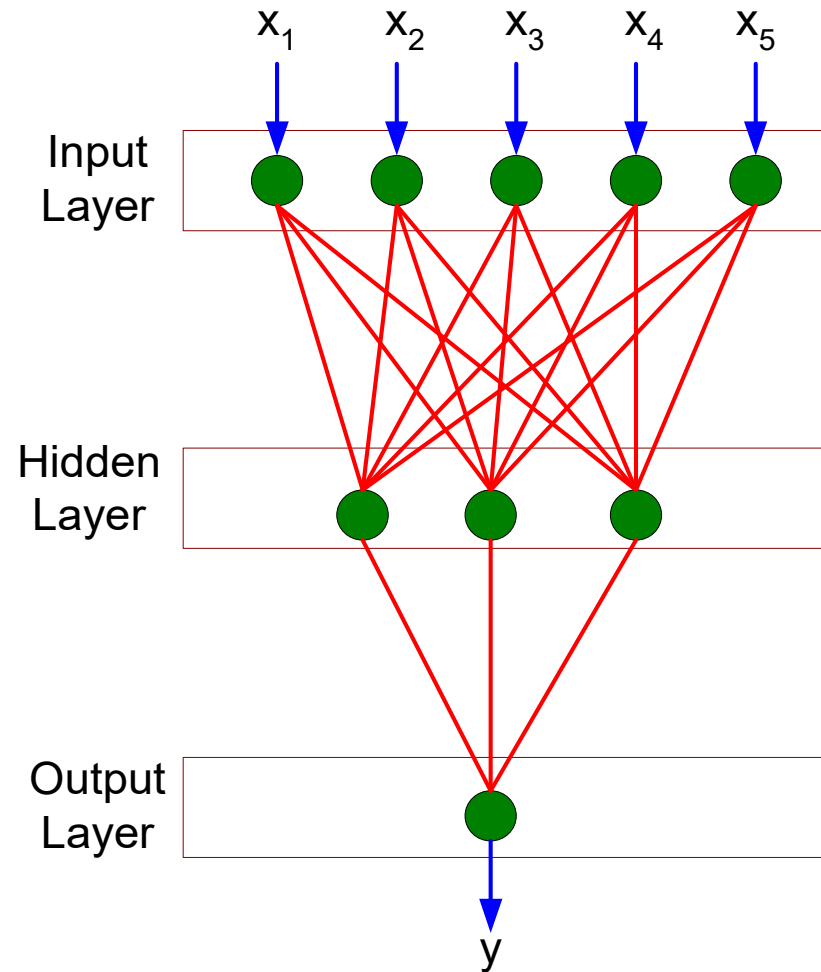
- Usually, the hidden layer is *smaller* than the input layer
  - Input:  $x_1 \dots x_n$
  - Hidden:  $h_1 \dots h_m$
  - $n > m$
- The output can be predicted from the values at the hidden layer
- Hence:
  - $m$  features should be sufficient to predict  $y$ !





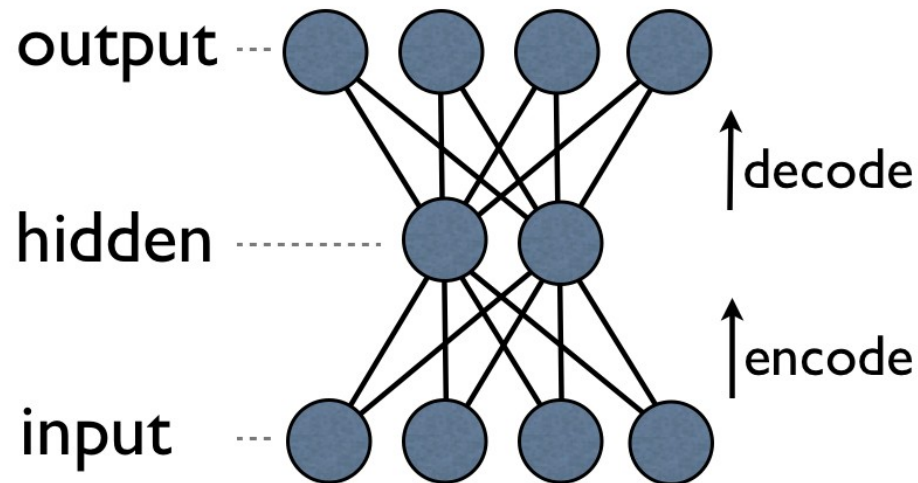
# What Happens at the Hidden Layer?

- We create a more compact representation of the dataset
  - Hidden:  $h_1 \dots h_m$
  - Which still conveys the information needed to predict  $y$
- Particularly interesting for sparse datasets
  - The resulting representation is usually dense
- But what if we don't know  $y$ ?



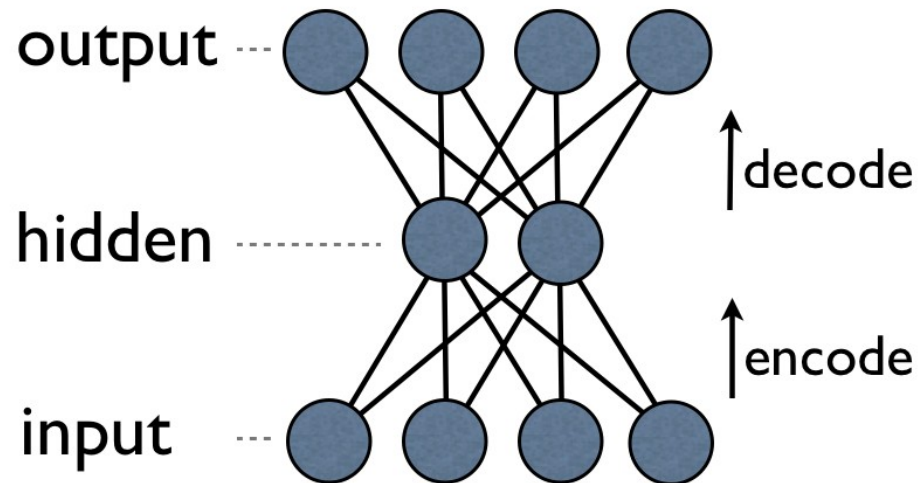
# Auto Encoders

- Auto encoders use the same example as input and output
  - i.e., they train a model for predicting an example from itself
  - using fewer variables
- Similar to PCA
  - But PCA provides only a linear transformation
  - ANNs can also create non-linear parameter transformations



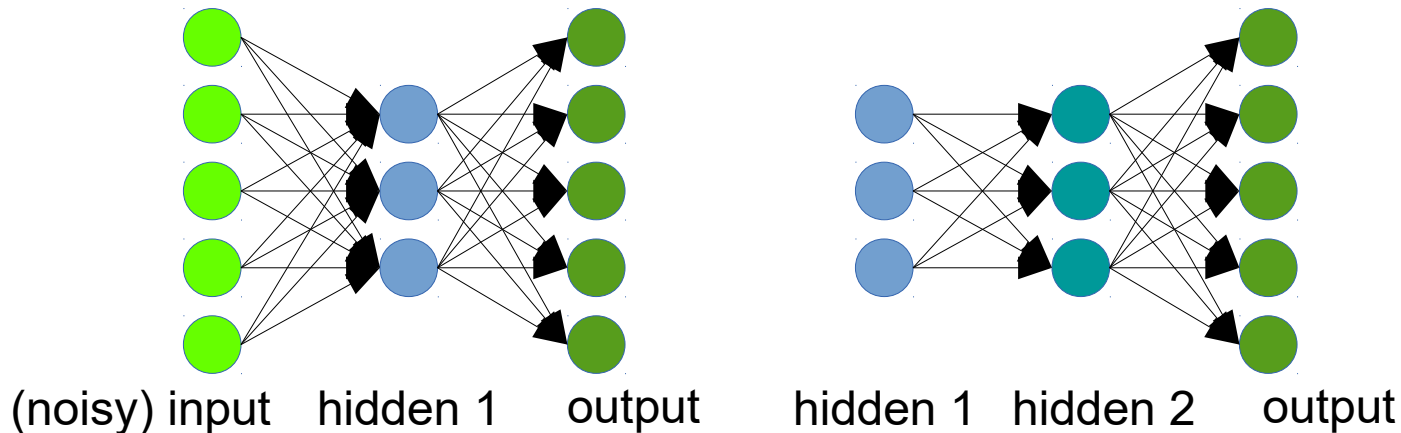
# Denoising Auto Encoders

- Instead of training with the same input and output
  - Add random noise to input
  - Keep output clean
- Result
  - A model that learns to remove noise from an instance



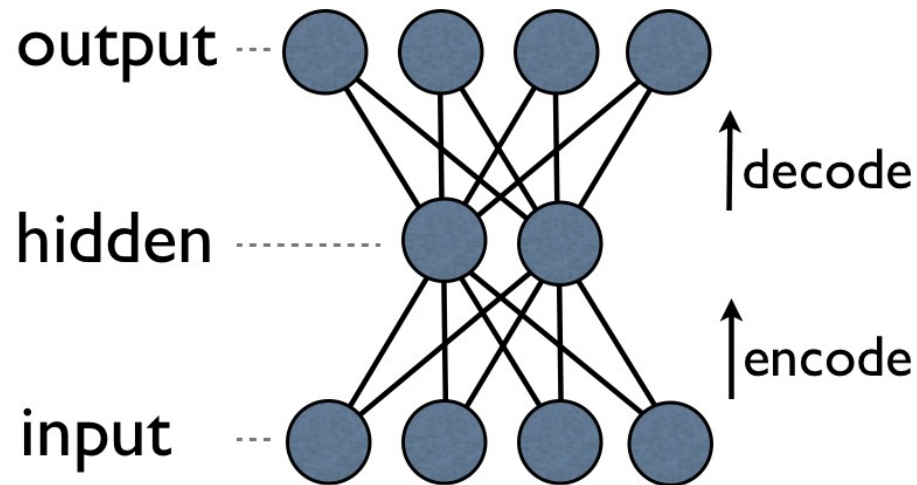
# Stacked (Denoising) Auto Encoders

- Stacked Auto Encoders contain several hidden layers
  - Hidden layers capture more complex hidden variables and/or denoising patterns
  - They are often trained consecutively:
    - First: train an auto encoder with one hidden layer
    - Second: train a second one-layer neural net:
      - first hidden layer as input
      - original as output



# Footnote: Auto Encoders for Outlier Detection

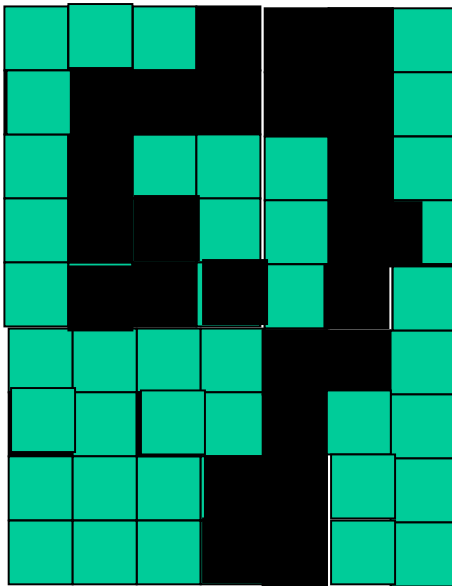
- Also known as Replicator Neural Networks  
(Hawkins et al., 2002)
- Train an autoencoder
  - That captures the patterns in the data
- Encode and decode each data point, measure deviation
  - Deviation is a measure for outlier score



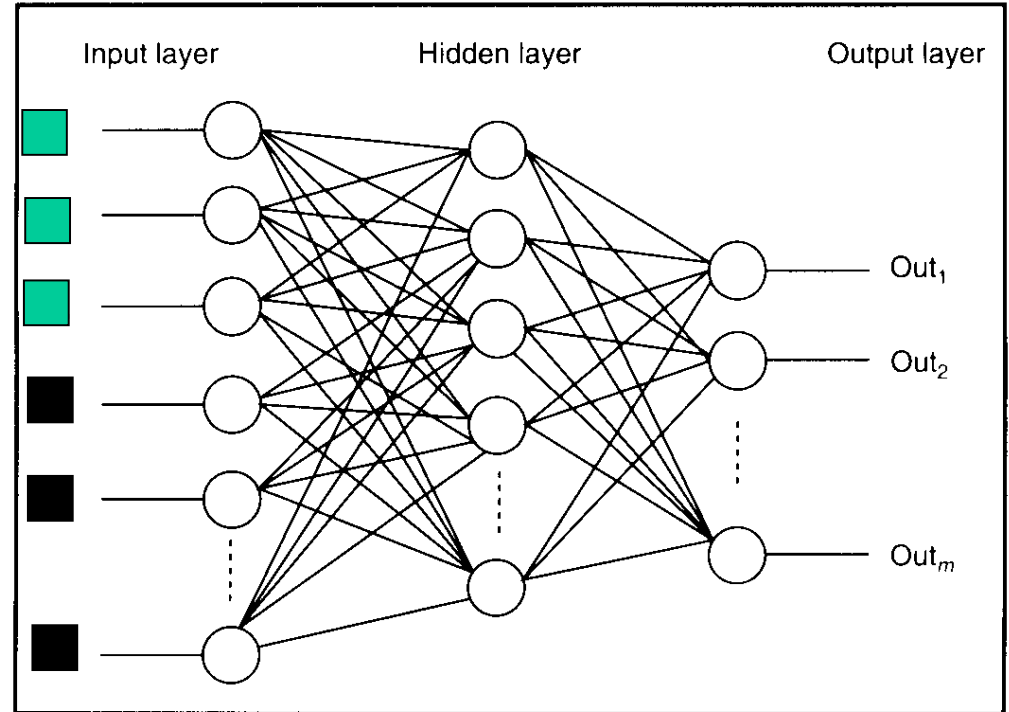
# From Classifiers to Feature Detectors



Figure 1.2: Examples of handwritten digits from U.S. postal envelopes.



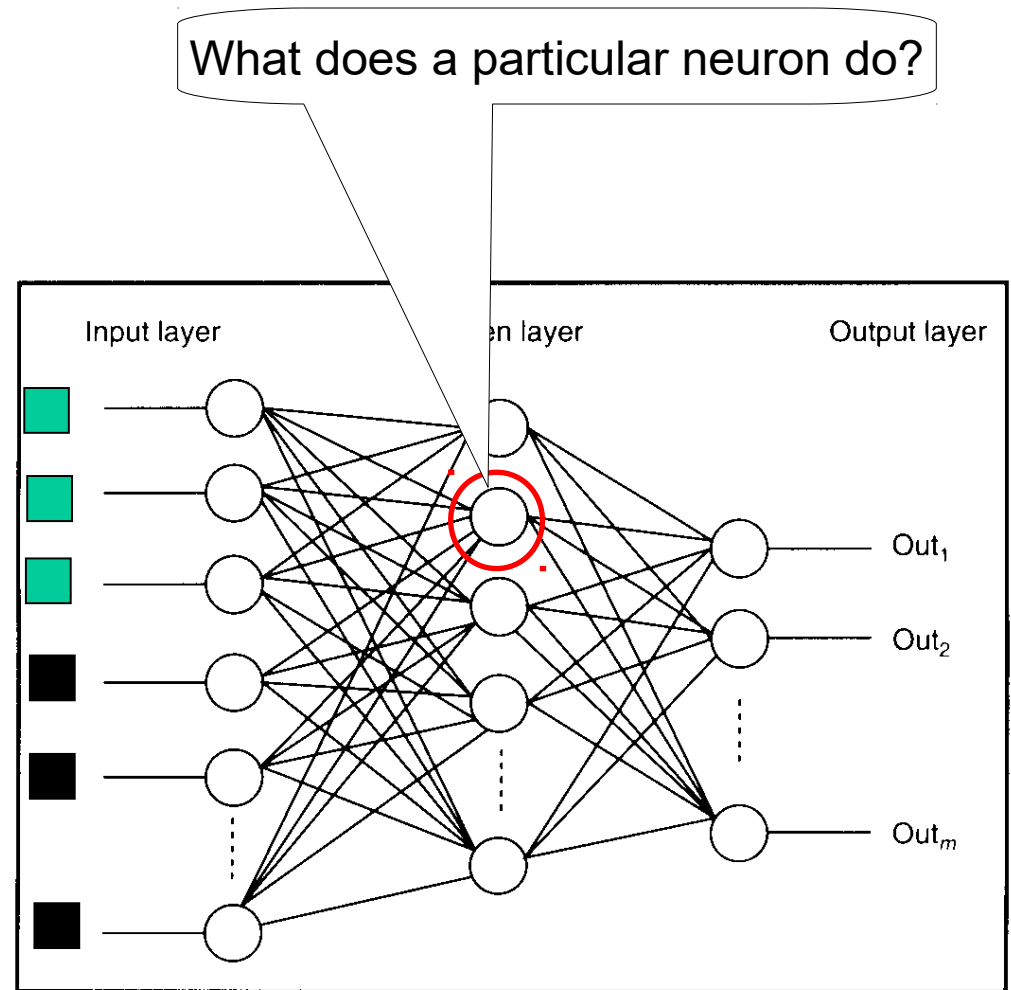
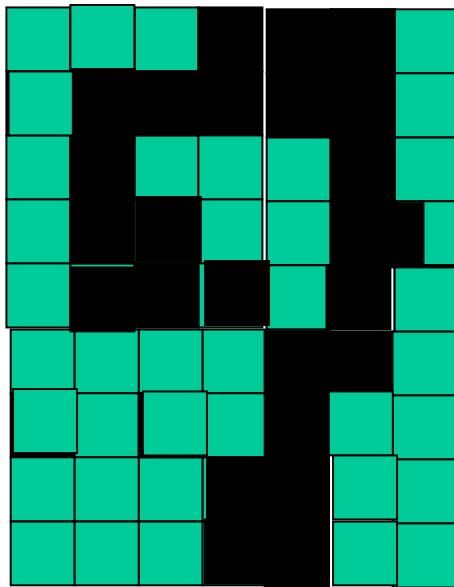
Some of the following slides are borrowed from <https://www.macs.hw.ac.uk/~dwcorne/Teaching/>



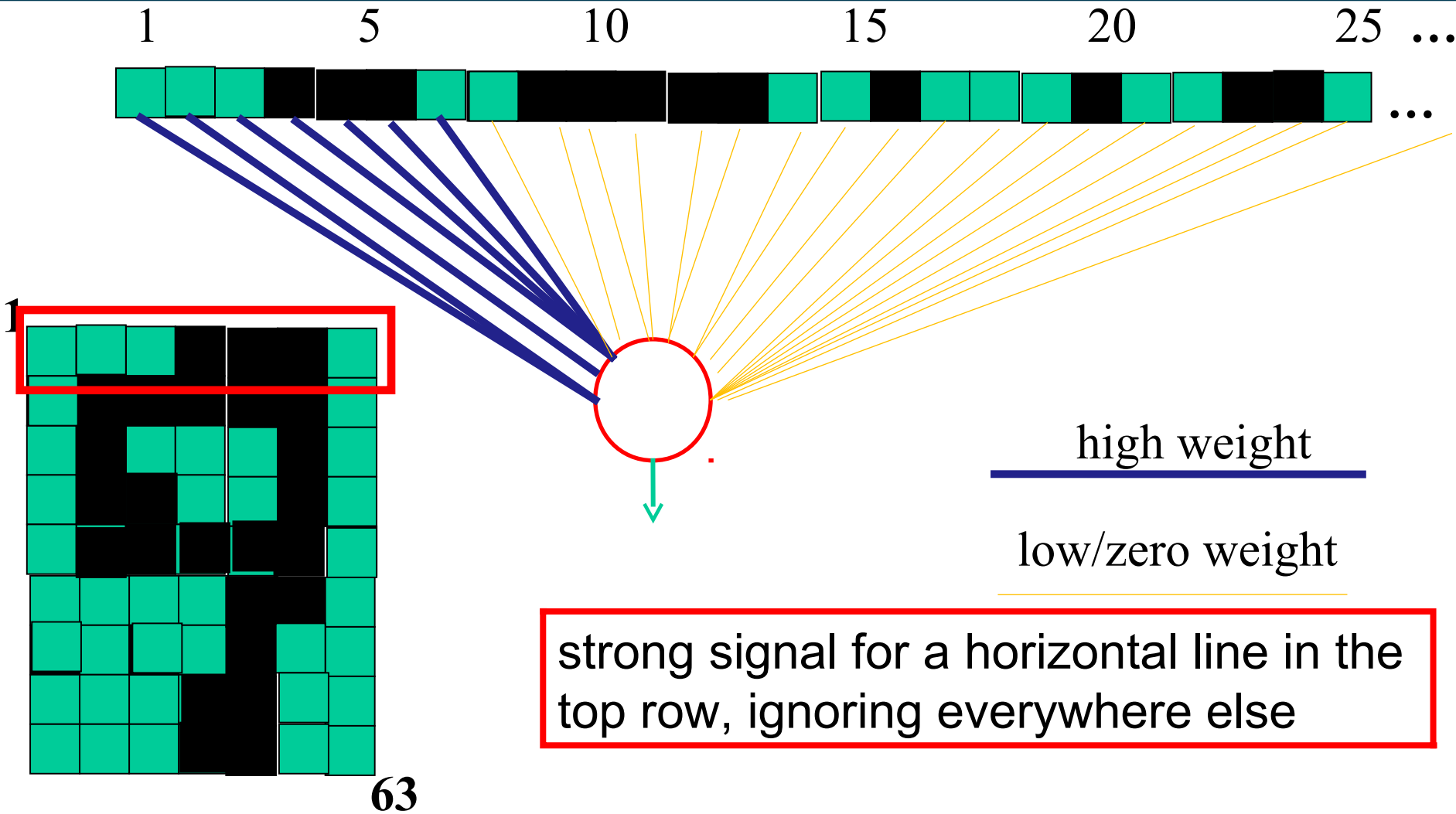
# From Classifiers to Feature Detectors



Figure 1.2: Examples of handwritten digits from U.S. postal envelopes.

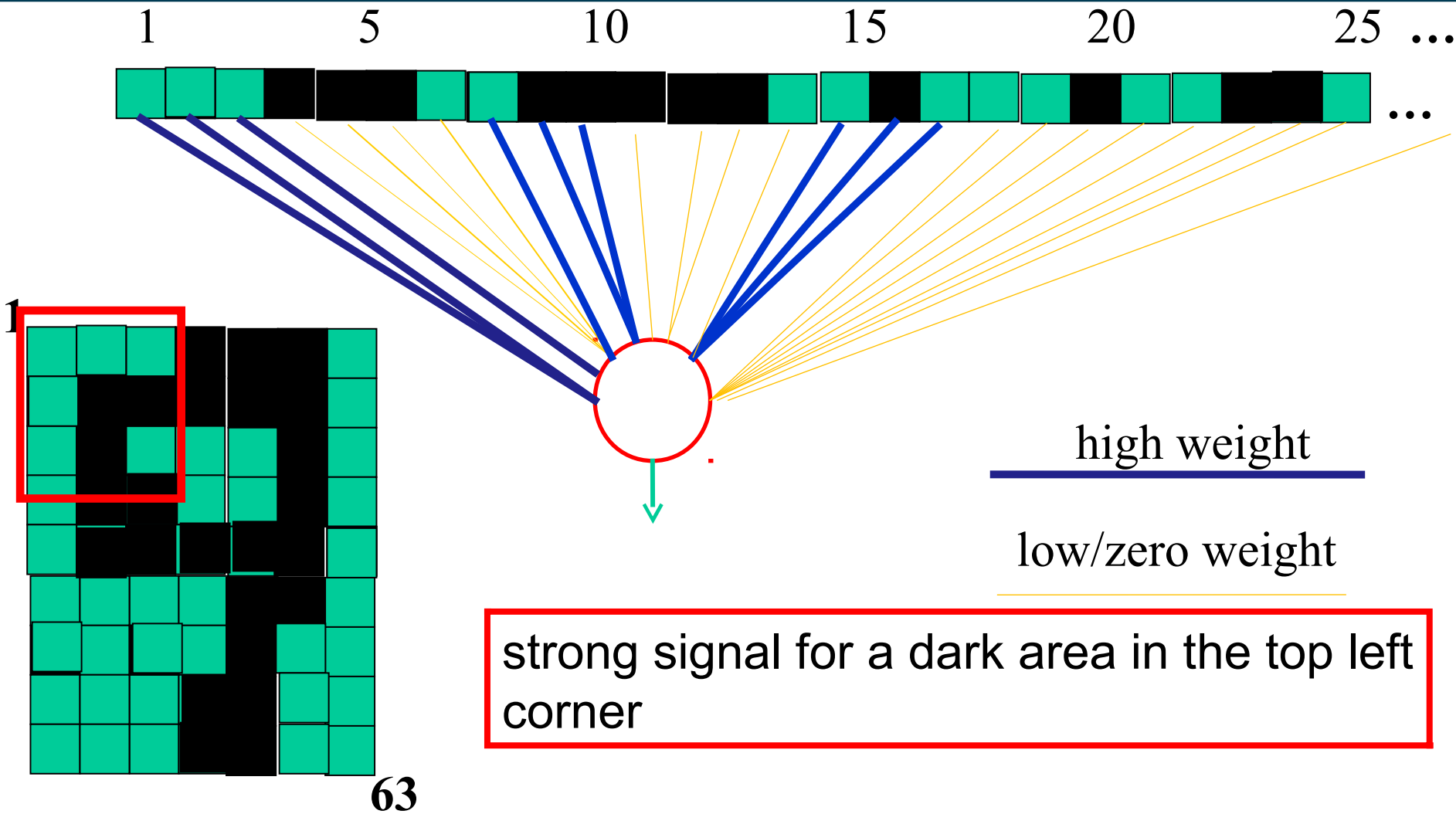


# What Happens at the Hidden Layer?





# What Happens at the Hidden Layer?



# Is that enough? What Features do we Need?



Figure 1.2: *Examples of handwritten digits from U.S. postal envelopes.*

# Is that enough? What Features do we Need?

- What we have
  - Line at the top
  - Dark area in the top left corner
  - ...
- What we want
  - Vertical Line
  - Horizontal Line
  - Circle
- Challenges
  - Positional variance
  - Color variance

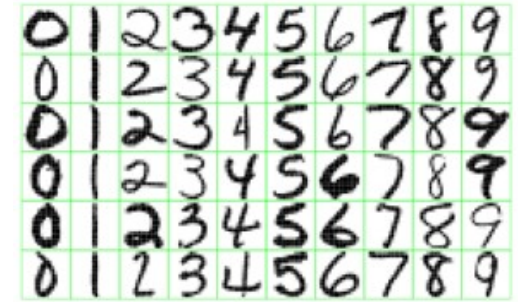


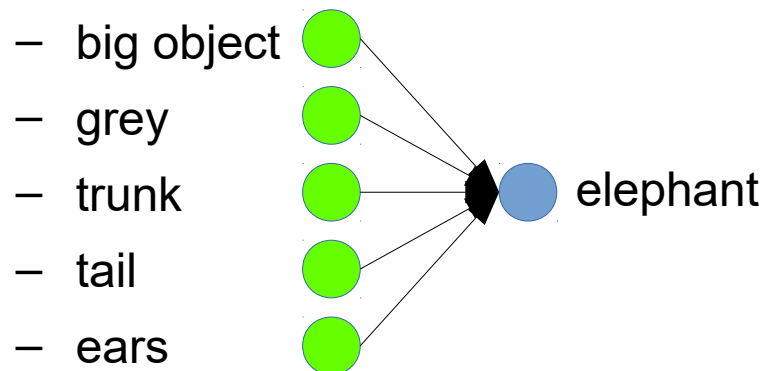
Figure 1.2: Examples of handwritten digits from U.S. postal envelopes.



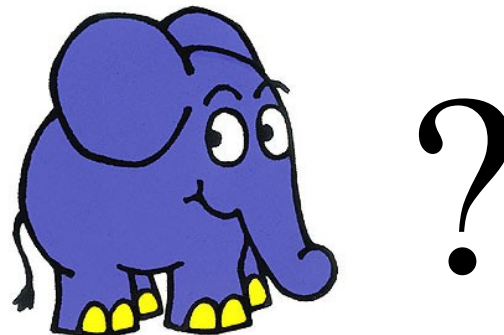
# Regularization with Dropout

- ANNs, and in particular Deep ANNs, tend to overfitting
- Example: image classification

- Elephant: five features in the highest level layer



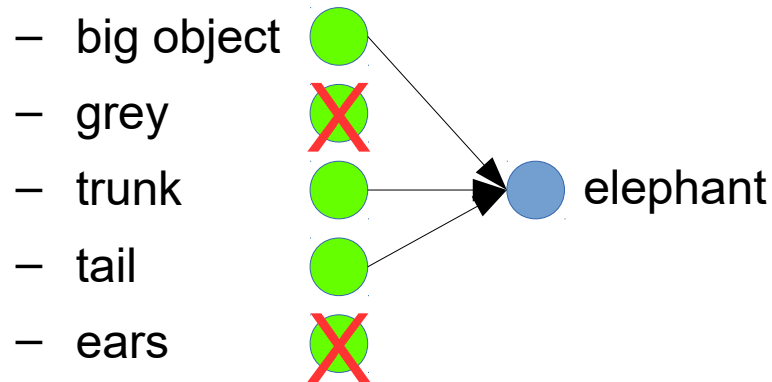
- Possible tendency to overfit:
  - expect all five to fire



# Regularization with Dropout

- Regularization
  - Randomly deactivate hidden neurons when training an example
  - E.g., factor  $\alpha=0.4$ : deactivate neurons randomly with probability 0.4

- Example:



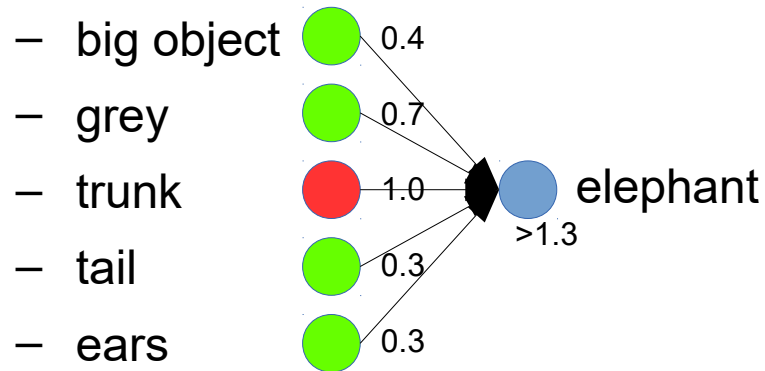
# Regularization with Dropout

- Regularization
  - Randomly deactivate hidden neurons when training an example
  - E.g., factor  $\alpha=0.4$ : deactivate neurons randomly with probability 0.4
- Result:
  - Learned model is more robust, less overfit
- For classification:
  - use all hidden neurons
- Problem: activation levels will be higher!
  - Multiply each output with  $1/(1+\alpha)$

# Regularization with Dropout

- For classification:
  - use all hidden neurons
- Problem: activation levels will be higher!
  - Correction: multiply each output with  $1/(1+\alpha)$

- Example:



without correction:  $0.4+0.7+0.3+0.3 = 1.7 > 1.3$

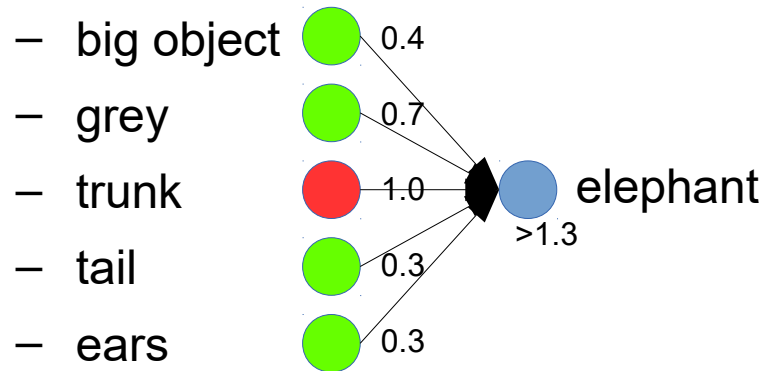




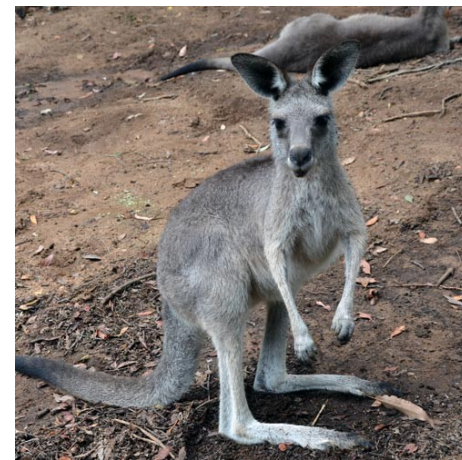
# Regularization with Dropout

- For classification:
  - use all hidden neurons
- Problem: activation levels will be higher!
  - Correction: multiply each output with  $1/(1+\alpha)$

- Example:



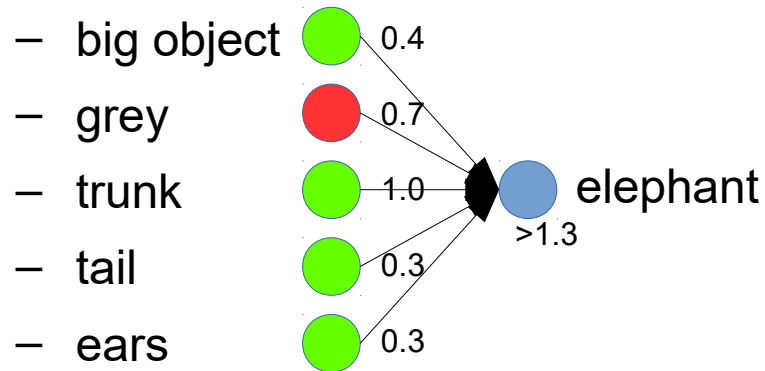
With correction:  $(5/7) * (0.4 + 0.7 + 0.3 + 0.3) = 1.21 < 1.3$



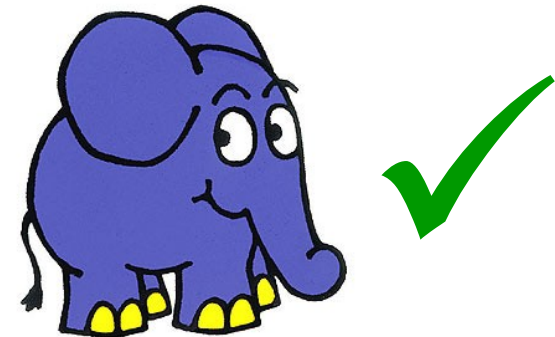
# Regularization with Dropout

- For classification:
  - use all hidden neurons
- Problem: activation levels will be higher!
  - Correction: multiply each output with  $1/(1+\alpha)$

- Example:



$$(5/7) * (0.4 + 1.0 + 0.3 + 0.3) = 1.43 > 1.3$$



# Architectures: Convolutional Neural Networks

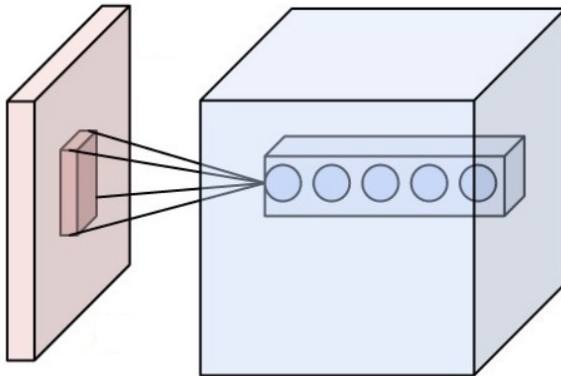
- Special architecture for image processing
- Problem: imagine a 4k resolution picture (3840x2160)
  - Treating each pixel as an input: 8M input neurons
  - Connecting that to a hidden layer of the same size:  
 $8M^2 = 64$  trillion weights to learn
  - This is hardly practical...
- Solution:
  - Convolutional neural networks

# Architectures: Convolutional Neural Networks

- Two parts:
  - Convolution layer
  - Pooling layer
- Stacks of those are usually used

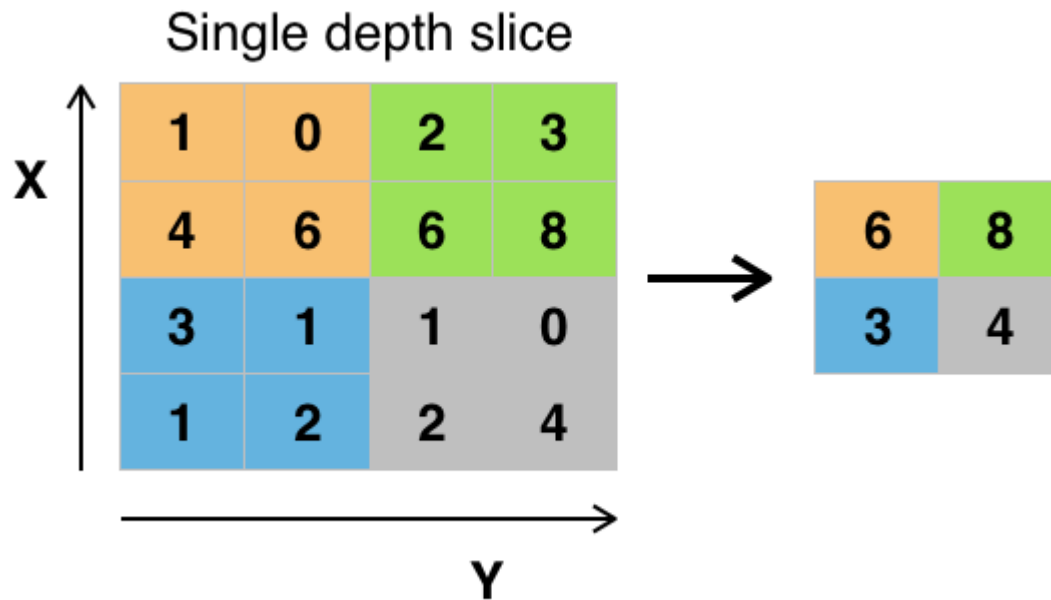
# Architectures: Convolutional Neural Networks

- Convolution layer
  - Each neuron is connected to a small  $n \times n$  square of the input neurons
  - i.e., number of connections is linear, not quadratic
- Use different neurons for detecting different features
  - They can share their weights
  - (intuition: a horizontal line looks the same everywhere)



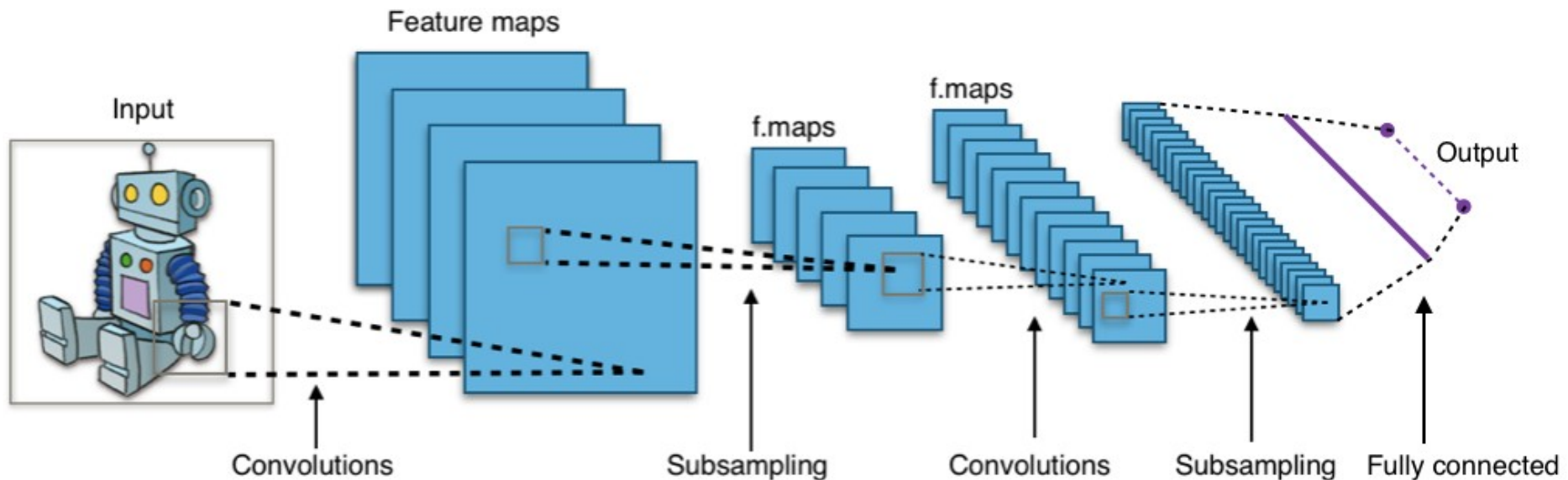
# Architectures: Convolutional Neural Networks

- Pooling layer (aka as *subsampling layer*)
  - Use only the maximum value of a neighborhood of neurons
  - Think: downsizing a picture
  - Number of neurons is divided by four with each pooling layer



# Architectures: Convolutional Neural Networks

- The big picture
  - With each pooling/subsampling step: 4 times less neurons
  - After a few layers, we have a decent number of inputs
  - Feed those into a fully connected ANN for the actual classification



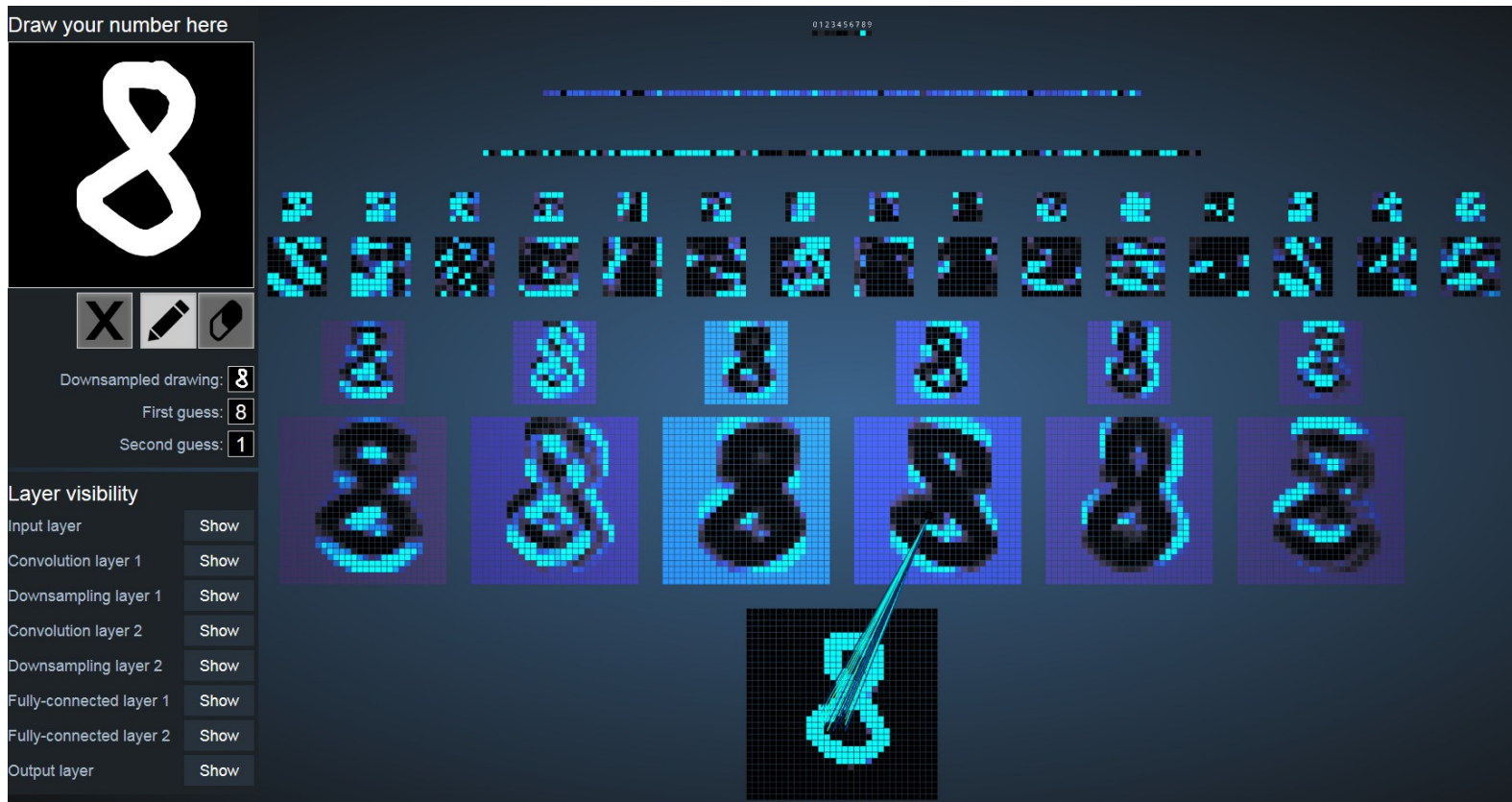
# Architectures: Convolutional Neural Networks

- The 4K picture revisited (3840x2160):
  - Treating each pixel as an input: 8M input neurons
  - Connecting that to a hidden layer of the same size:  
 $8M^2 = 64$  trillion weights to learn
- Number of connections (weights to be learned) in the first convolutional layer:
  - Assume each hidden neuron is connected to a 16x16 square
  - and we learn 256 hidden features (i.e., 256 layers of convolutional neurons)
  - $16 \times 16 \times 256 \times 8M =$  still 526 billion weights
- But: neurons for the same hidden feature share their weight
  - Thus, it's just  $16 \times 16 \times 256 = 65k$  weights



# Architectures: Convolutional Neural Networks

- Nice play around visualization for handwritten number detection



<http://scs.ryerson.ca/~aharley/vis/conv/flat.html>

# Architectures: Convolutional Neural Networks

- In practice, several layers are used
- Picture on the right
  - Google's GoogLeNet (Inception)
  - Current state of the art in image classification
- Can be used as a pre-trained network



# What does an Artificial Neural Network Learn?





# What does an Artificial Neural Network Learn?



# What does an Artificial Neural Network Learn?

- Image recognition networks can be attacked
  - changing small pixels barely noticed by humans



$x$

“panda”

57.7% confidence

+ .007 ×



$\text{sign}(\nabla_x J(\theta, x, y))$

“nematode”

8.2% confidence

=



$x +$

$\epsilon \text{sign}(\nabla_x J(\theta, x, y))$

“gibbon”

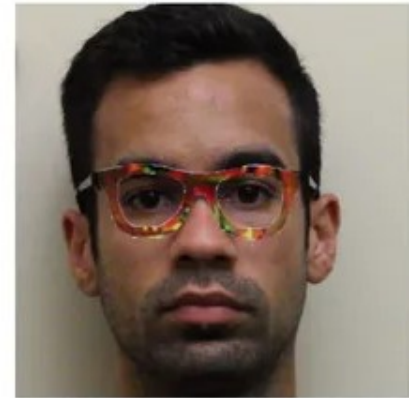
99.3 % confidence

Goodfellow et al.: Explaining and Harnessing Adversarial Examples, 2015



# Possible Implications

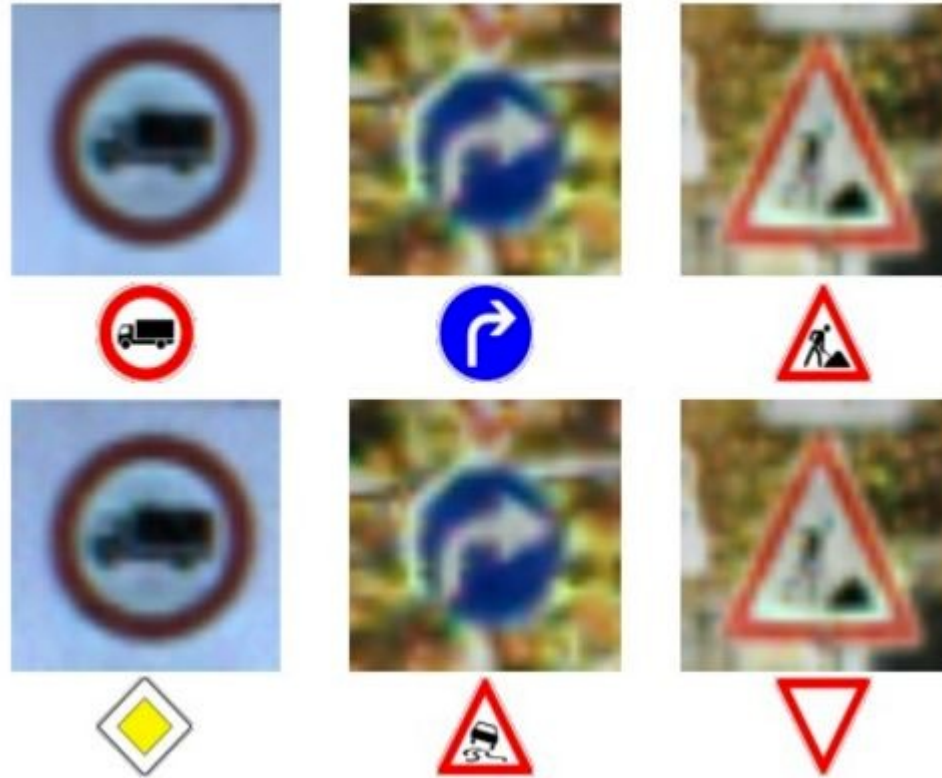
- Face Detection



Sharif et al.: Accessorize to a Crime: Real and Stealthy Attacks on State-of-the-Art Face Recognition, 2016

# Possible Implications

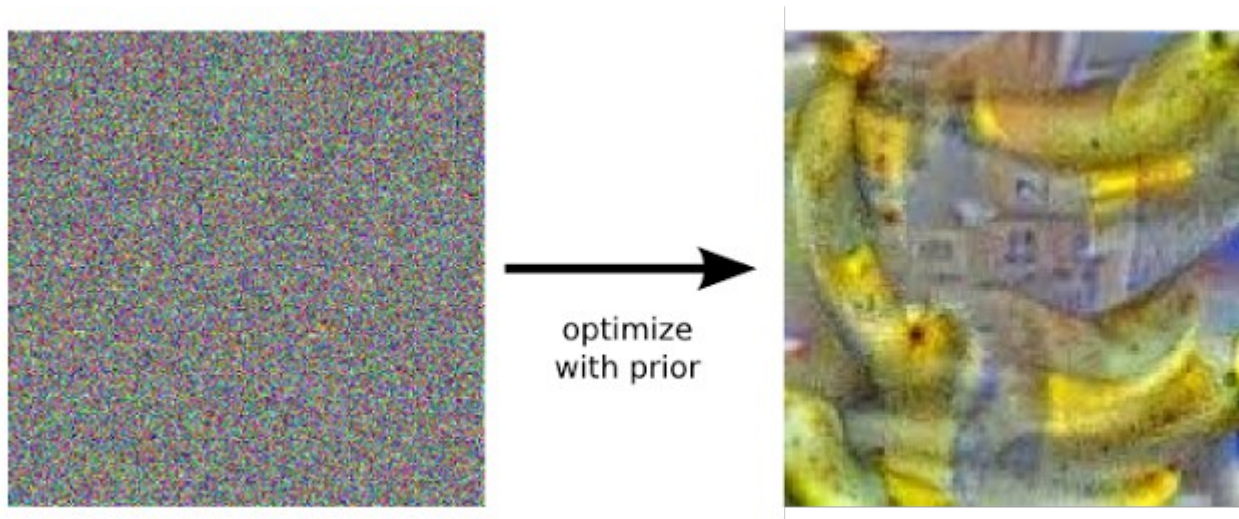
- Autonomous Driving



Papernot et al.: Practical Black-Box Attacks against Machine Learning, 2017

# Turning a Neural Network Upside Down

- Assume you have a neural network trained for image classification
  - Reverse application: given label, synthesize image
  - Additional constraint (prior): neighboring pixels correlate



<https://research.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html>



# Turning a Neural Network Upside Down

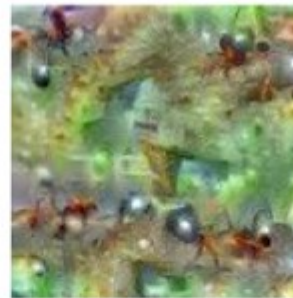
- Asking for prototype pictures of certain labels



Hartebeest



Measuring Cup



Ant



Starfish



Anemone Fish



Banana



Parachute

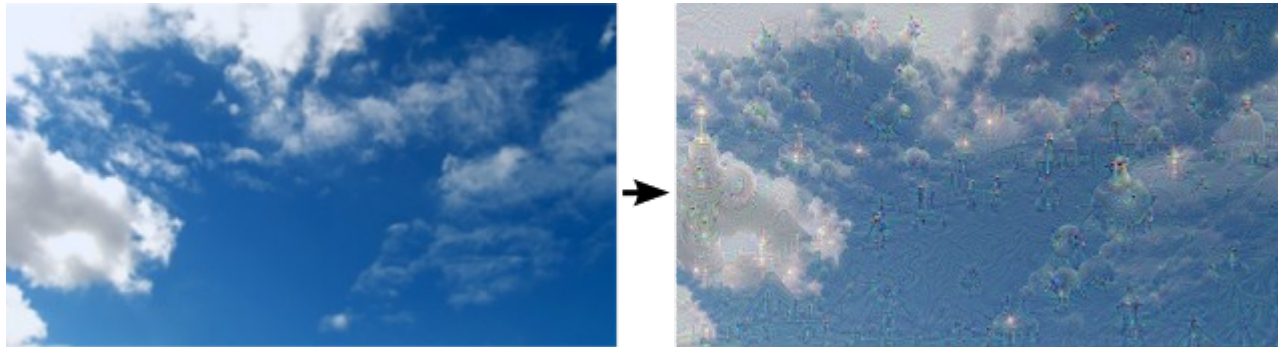


Screw

<https://research.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html>

# Making a Neural Network Daydream

- First step: classify an image
- Second step: amplify (i.e., use pair of input image and predicted label as additional training example)



<https://research.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html>

# Making a Neural Network Daydream

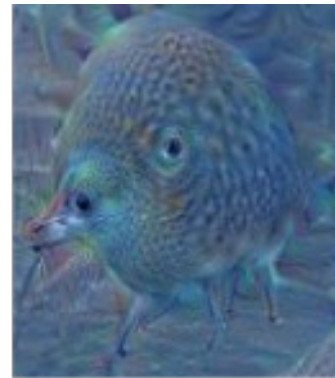
- First step: classify an image
- Second step: amplify (i.e., use pair of input image and predicted label as additional training example)



"Admiral Dog!"



"The Pig-Snail"



"The Camel-Bird"

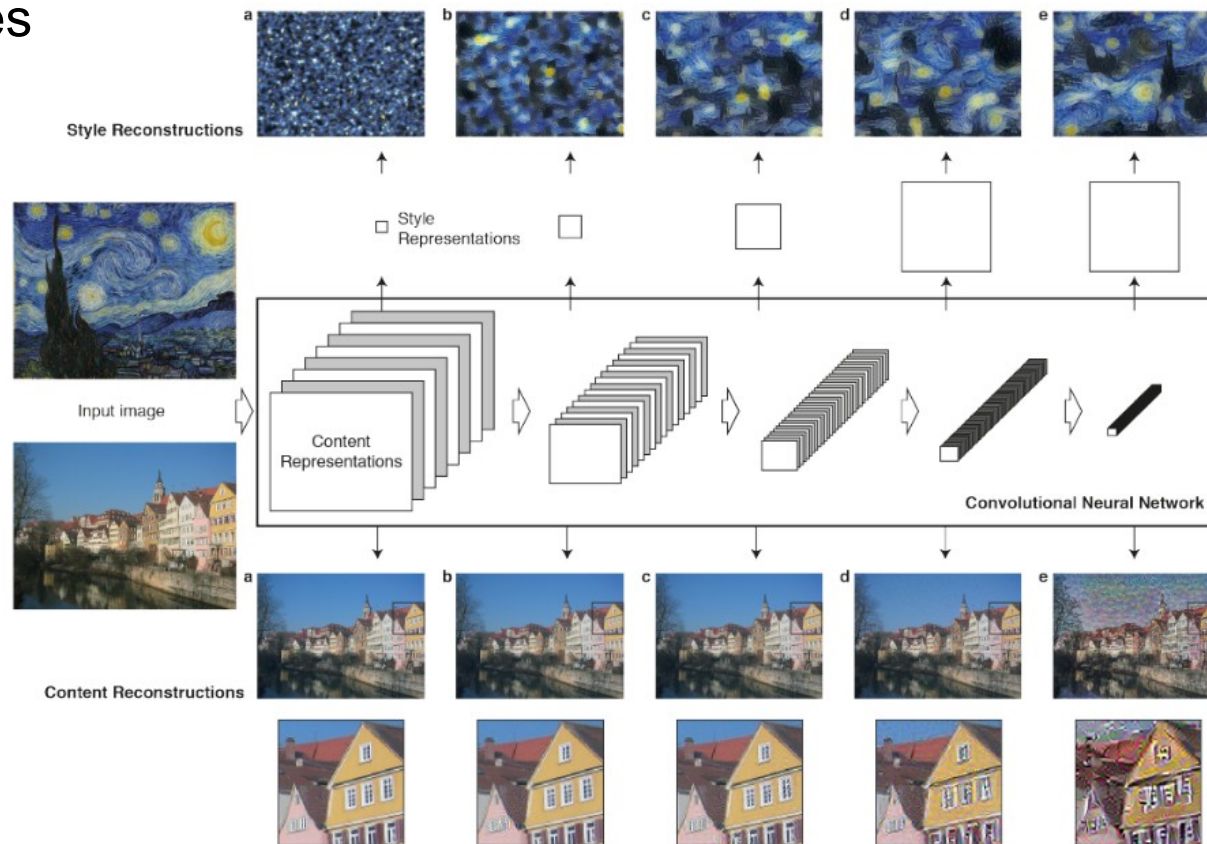


"The Dog-Fish"

<https://research.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html>

# Neural Networks for Arts

- Train a neural network that extracts both artistic as well as content features



<https://arxiv.org/pdf/1508.06576.pdf>



# Neural Networks for Arts

- Then: generate picture with a given set of contents and style

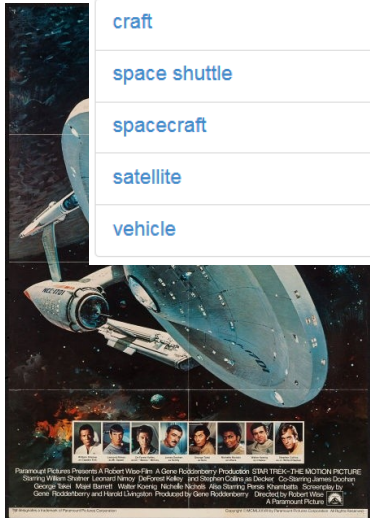


<https://arxiv.org/pdf/1508.06576.pdf>

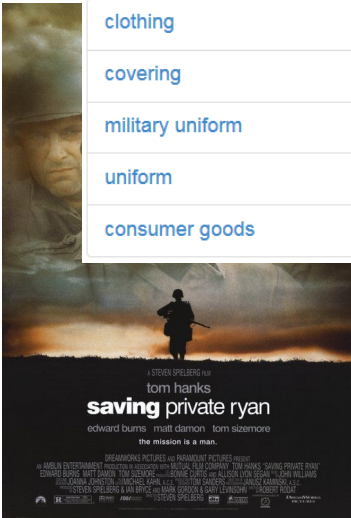
# Reusing Pre-trained Networks

- The output of a network can be used as an input to yet another classifier (neural network or other)
- Think: a multi-label image classifier as an auto-encoder
- Example: predict movie genre from poster
  - Using an image classifier trained for object recognition

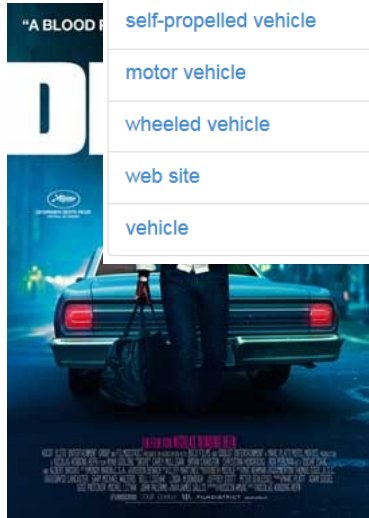
Maximally accurate	Maximally specific
craft	0.82347
space shuttle	0.82341
spacecraft	0.81132
satellite	0.79922
vehicle	0.60857



Maximally accurate	Maximally specific
clothing	0.67786
covering	0.67213
military uniform	0.61569
uniform	0.60664
consumer goods	0.60306



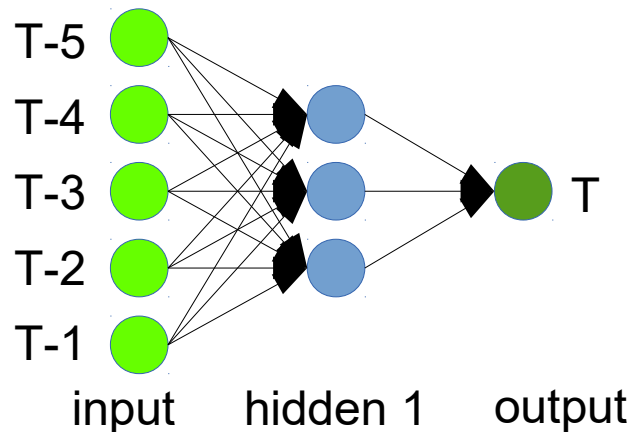
Maximally accurate	Maximally specific
self-propelled vehicle	0.56750
motor vehicle	0.54098
wheeled vehicle	0.53425
web site	0.51618
vehicle	0.50493



<http://demo.caffe.berkeleyvision.org/>

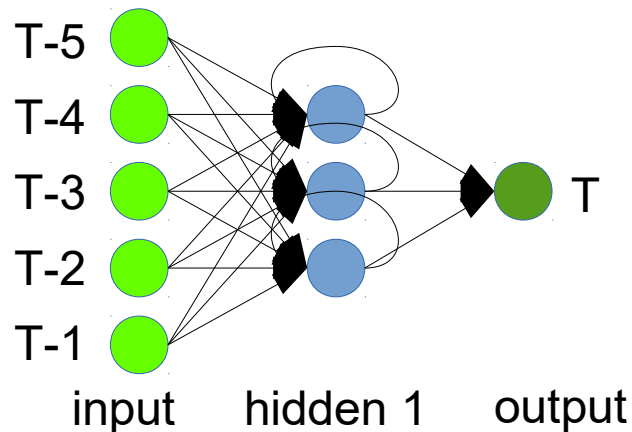
# Using ANNs for Time Series Prediction

- Last week, we have learned about time series prediction
  - Long term trends
  - Seasonal effects
  - Random fluctuation
  - ...
- Scenario: predict the continuation of a time series
  - let's use the last five values as features (3-window)



# Using ANNs for Time Series Prediction

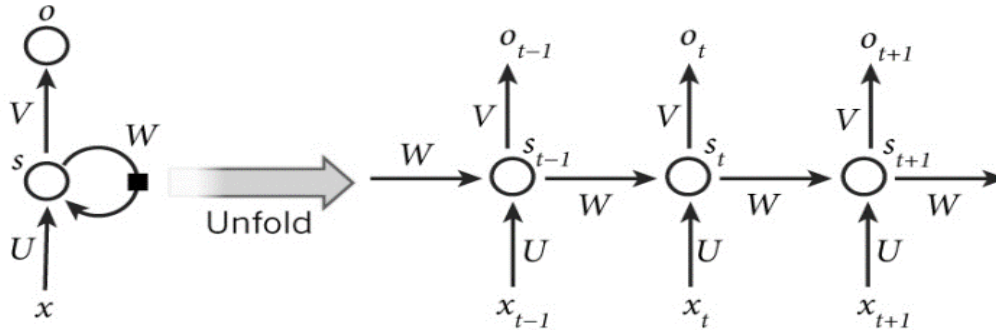
- Assume that this is running continuously
  - we will always just use the last five examples
  - we cannot detect longer term trends
- Solution
  - introduce a memory
  - Implementation: backward loops



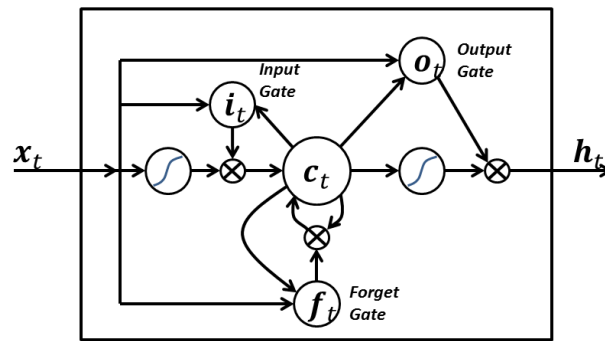


# Long Short Term Memory Networks (LSTM)

- Notion of a recursive neural network
  - A folded deep neural network
  - Note: influence of the past decays over time

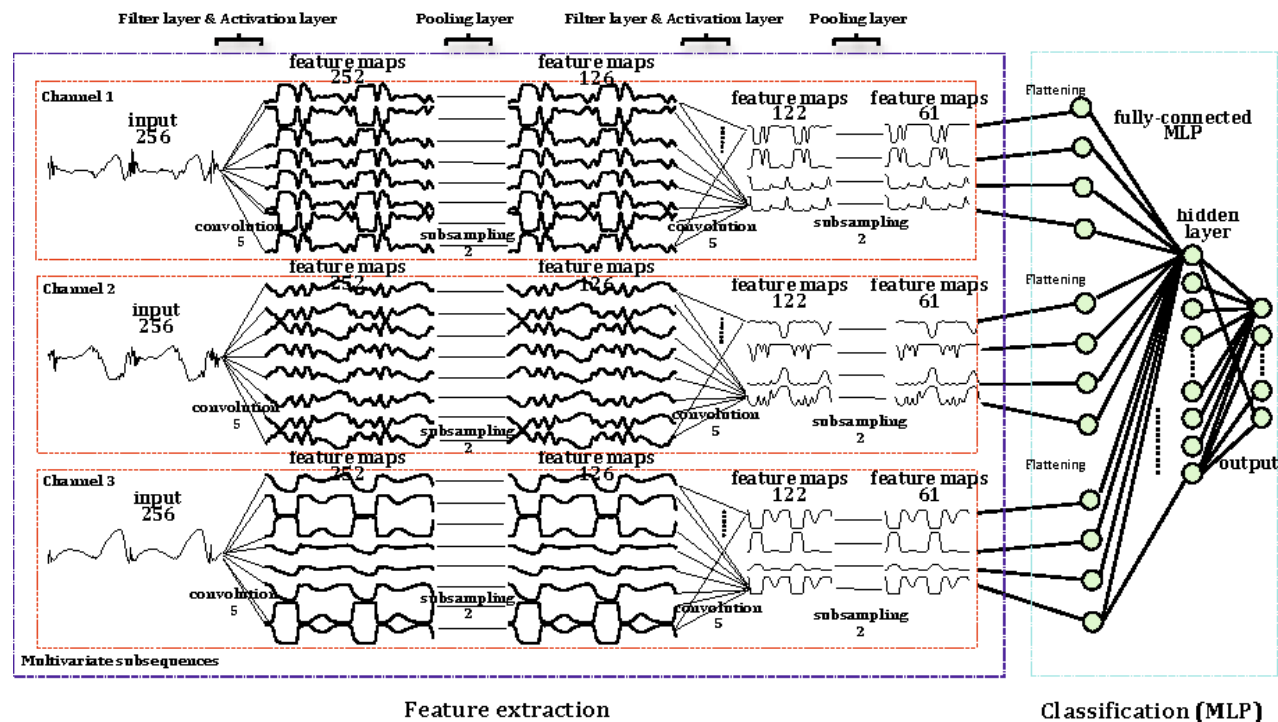


- LSTMs are special recursive neural networks



# CNNs for Time Series Prediction

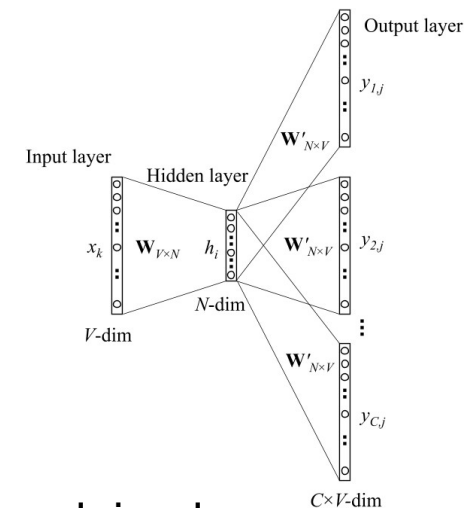
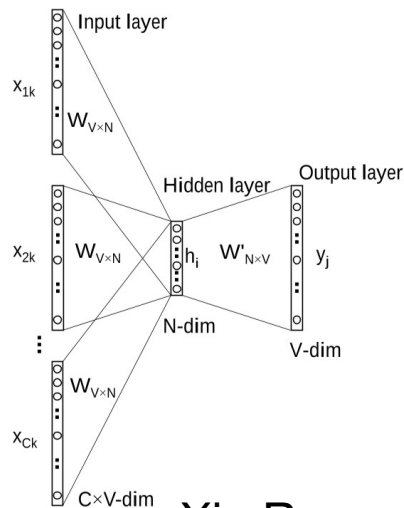
- Notion: time series also have typical features
  - Think: trends, seasonal variation, ...



Zheng et al.: Time Series Classification Using Multi-Channels Deep Convolutional Neural Networks, 2014

# word2vec

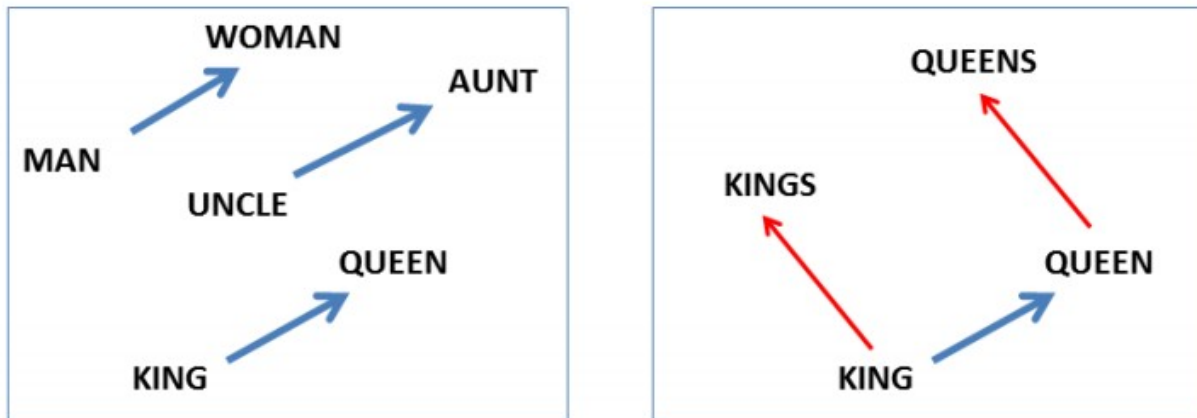
- word2vec is similar to an auto encoder for words
- Training set: a text corpus
- Training task variants:
  - Continuous bag of words (CBOW): predict a word from the surrounding words
  - Skip-Gram: predicts surrounding words of a word



Xin Rong: word2vec parameter learning explained

# word2vec

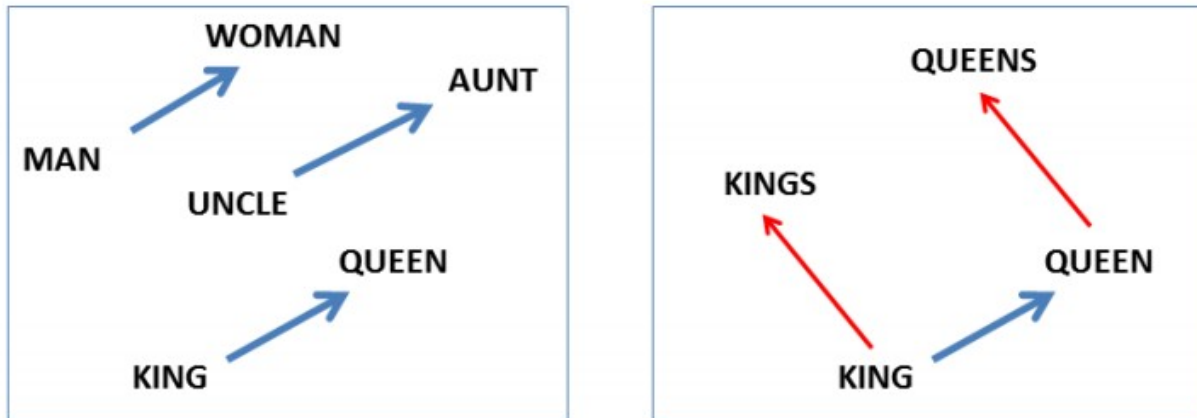
- word2vec creates an n-dimensional vector for each word
- Each word becomes a point in a vector space
- Properties:
  - Similar words are positioned to each other
  - Relations have the same direction



(Mikolov et al., NAACL HLT, 2013)

# word2vec

- Arithmetics are possible in the vector space
  - king – man + woman  $\approx$  queen
- This allows for finding analogies:
  - king:man  $\leftrightarrow$  queen:woman
  - knee:leg  $\leftrightarrow$  elbow:forearm
  - Hillary Clinton:democrat  $\leftrightarrow$  Donald Trump:Republican



(Mikolov et al., NAACL HLT, 2013)

# word2vec

- Pre-trained models exist
  - e.g., on Google News Corpus or Wikipedia
- Can be downloaded and used instantly

```
172ms [{"Nine_Inch_Nails",0.6071341037750244},{"Reznor",0.5817075371742249},{"NIN",0.5102616548538208},  
["Radiohead",0.4629957675933838},{"Metallica",0.45992764830589294}]
```

If you don't get "queen" back, something went wrong and baby SkyNet cries.

Try more examples too: "he" is to "his" as "she" is to ?, "Berlin" is to "Germany" as "Paris" is to ? (click to fill in).

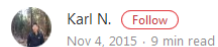
Till\_Lindemann is to Rammstein as Trent\_Reznor is to

**Nine\_Inch\_Nails**

Computer\_Science is to Computers as Philosophy is to **Books**

# From word2vec to anything2vec

- Vector space embeddings have recently become en vogue
  - Basically, everything that can be expressed as sequences can be processed by the word2vec pipeline
- There are vector space embeddings for...
  - Graph2vec (social graphs)
  - Doc2vec (entire documents)
  - RDF2Vec (RDF graphs)
  - Chord2Vec (music chords)
  - Audio2vec
  - Video2vec
  - Gene2vec (Amino acid sequences)
  - Emoji2vec
  - ...



## Anything2Vec

### 2Vec or Not 2Vec?

This might be old news to you, but if you're considering the use of word embeddings, our suggestion: just take the plunge. We've read a "few" studies documenting their effectiveness, not the least of which is our personal favorite:

# Summary

- Artificial Neural Networks
  - Are a powerful learning tool
  - Can approximate universal functions / decision boundaries
- Deep neural networks
  - ANNs with multiple hidden layers
  - Hidden layers learn to identify relevant features
  - Many architectural variants exist
- Pre-trained models
  - e.g., for image recognition
  - word embeddings
  - ...



# Questions?

