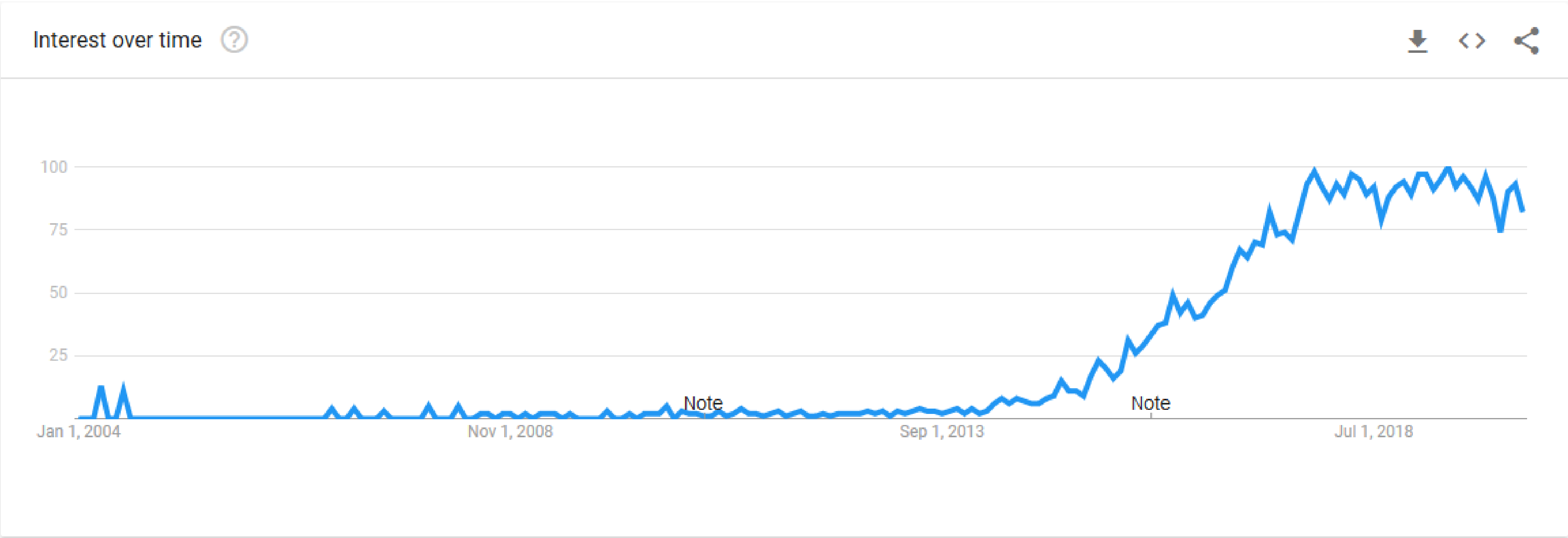
UNIVERSITÄT MANNHEIM

Data Mining II
Neural Networks and Deep Learning

Heiko Paulheim

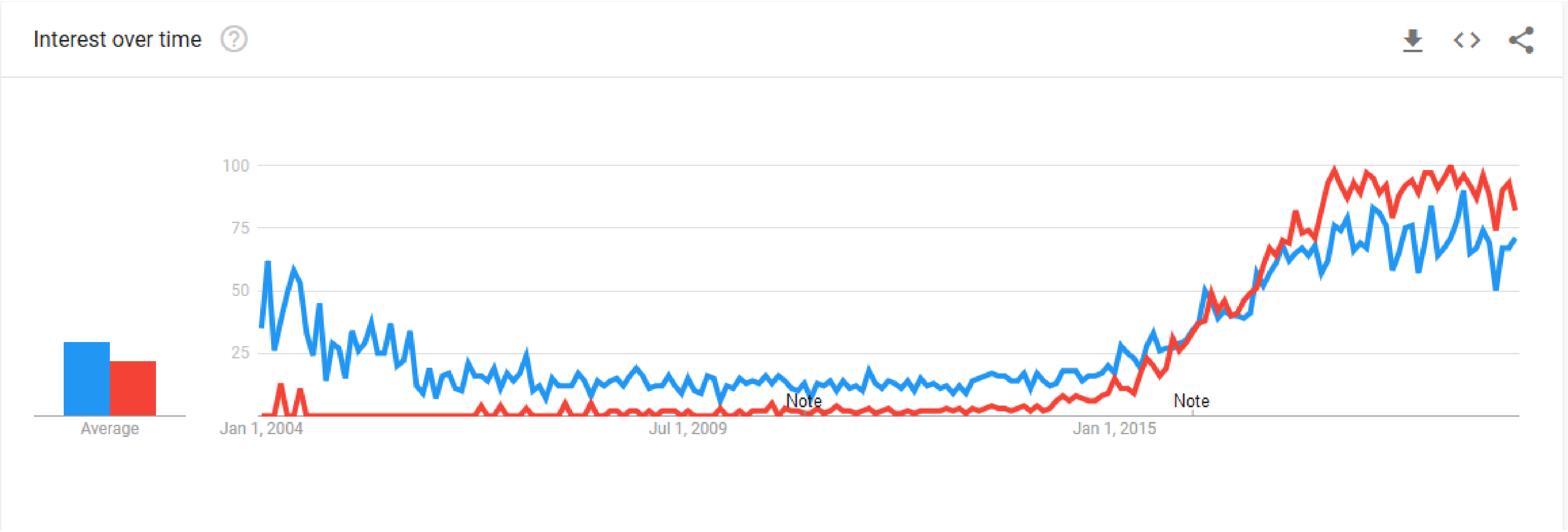
Deep Learning

- A recent hype topic



Deep Learning

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



Deep Learning

- Contents of this Lecture
  - Recap of neural networks
  - The backpropagation algorithm
  - Auto Encoders
  - Deep Learning
  - Network Architectures
  - Transfer Learning with Neural Networks

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

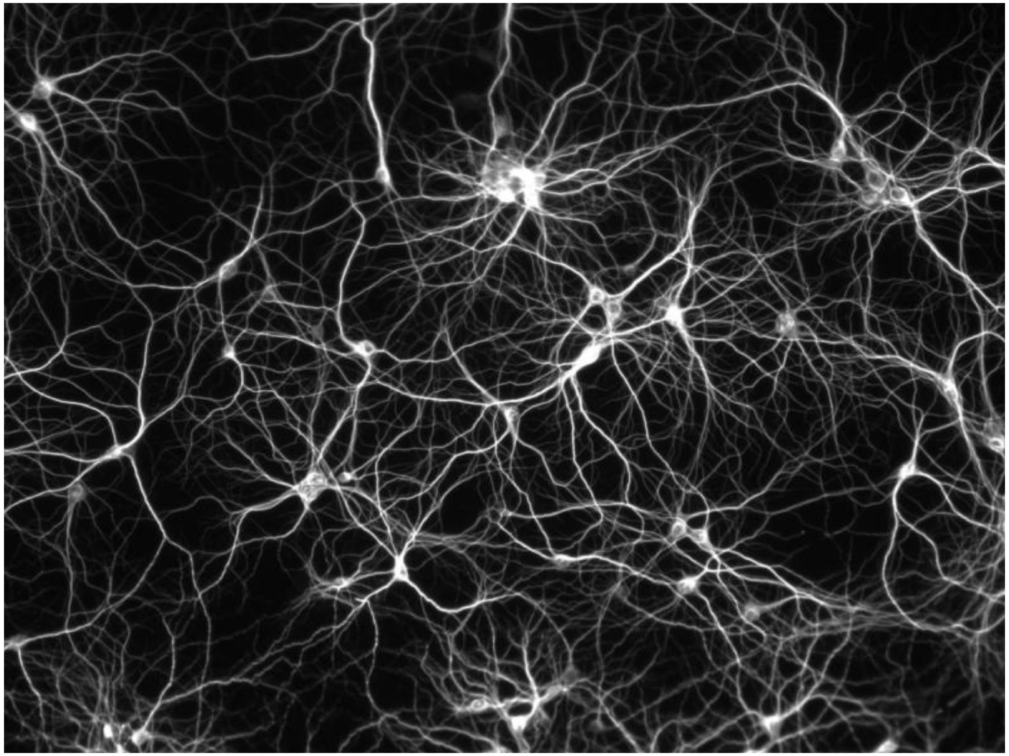
- Smallest model:
  - if at least two of Employed, Owns House, and Balance Account are yes → Get Credit is yes
- As rule set:

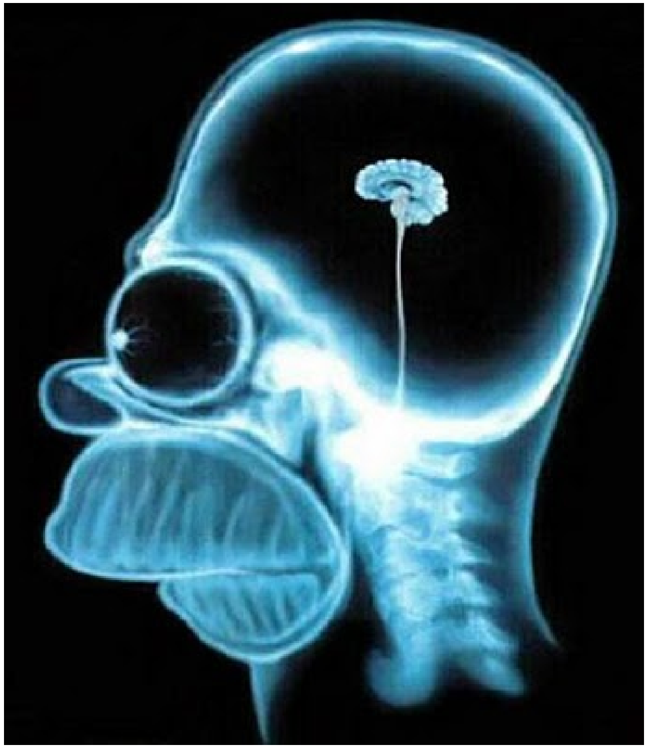
  Employed=yes and OwnsHouse=yes => yes
  Employed=yes and BalanceAccount=yes => yes
  OwnsHouse=yes and BalanceAccount=yes => yes
  => no

- 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 20 attributes need to be yes” requires more than 15,000 rules!

Artificial Neural Networks

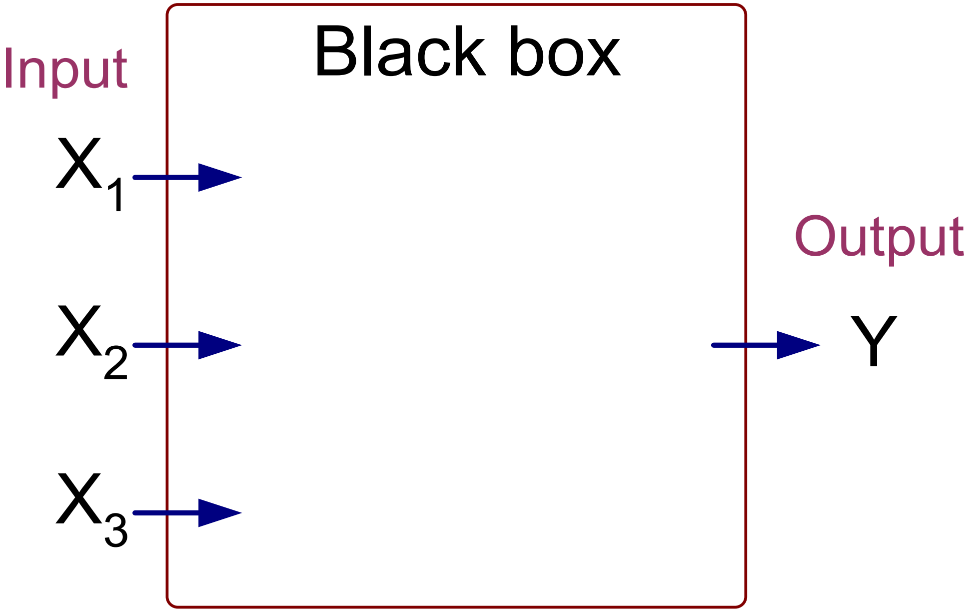
- Inspiration
  - one of the most powerful super computers in the world





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



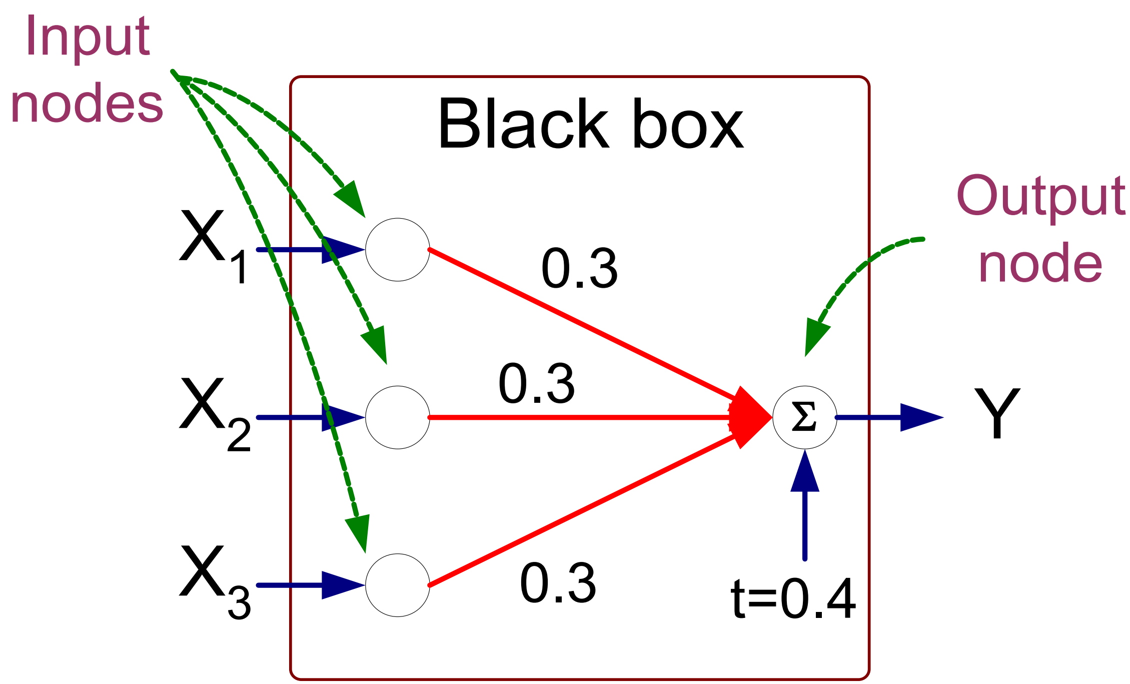
Output Y is 1 if at least two of the three inputs are equal to 1.

Example: Credit Rating

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

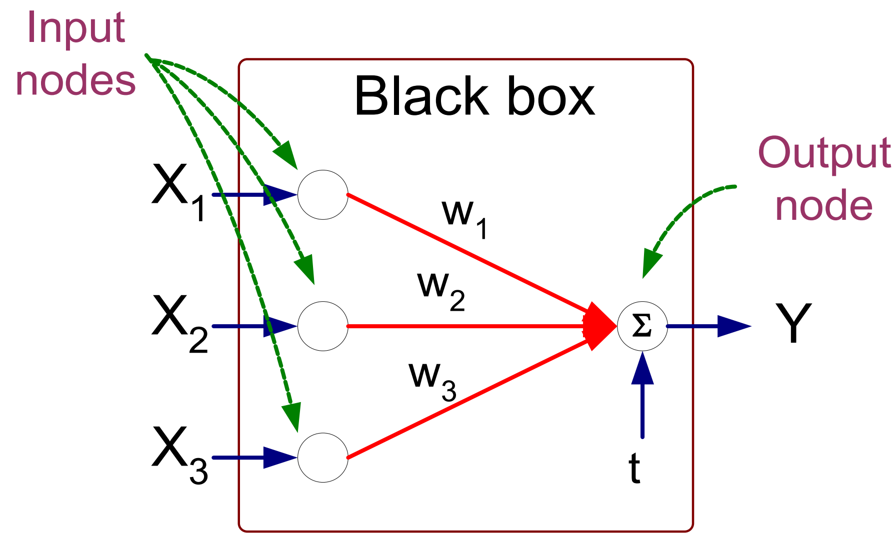


$$Y = I(0.3X_1 + 0.3X_2 + 0.3X_3 - 0.4 > 0)$$

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

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

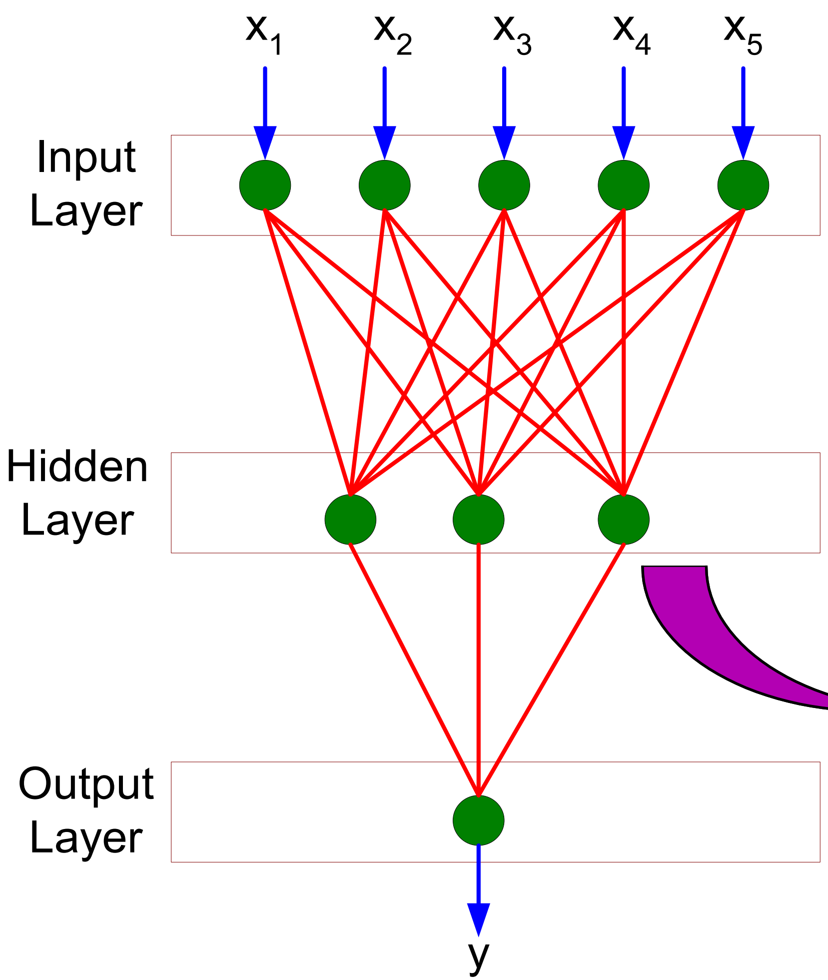


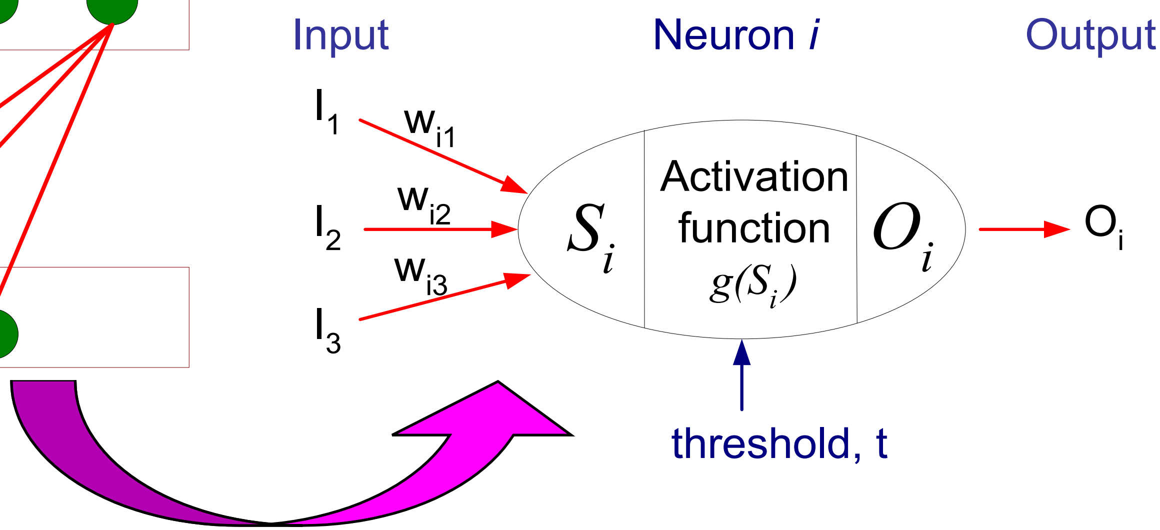
Perceptron Model

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

$$Y = sign(\sum_i w_i X_i - t)$$

General Structure of ANN





Training ANN means learning the weights of the neurons

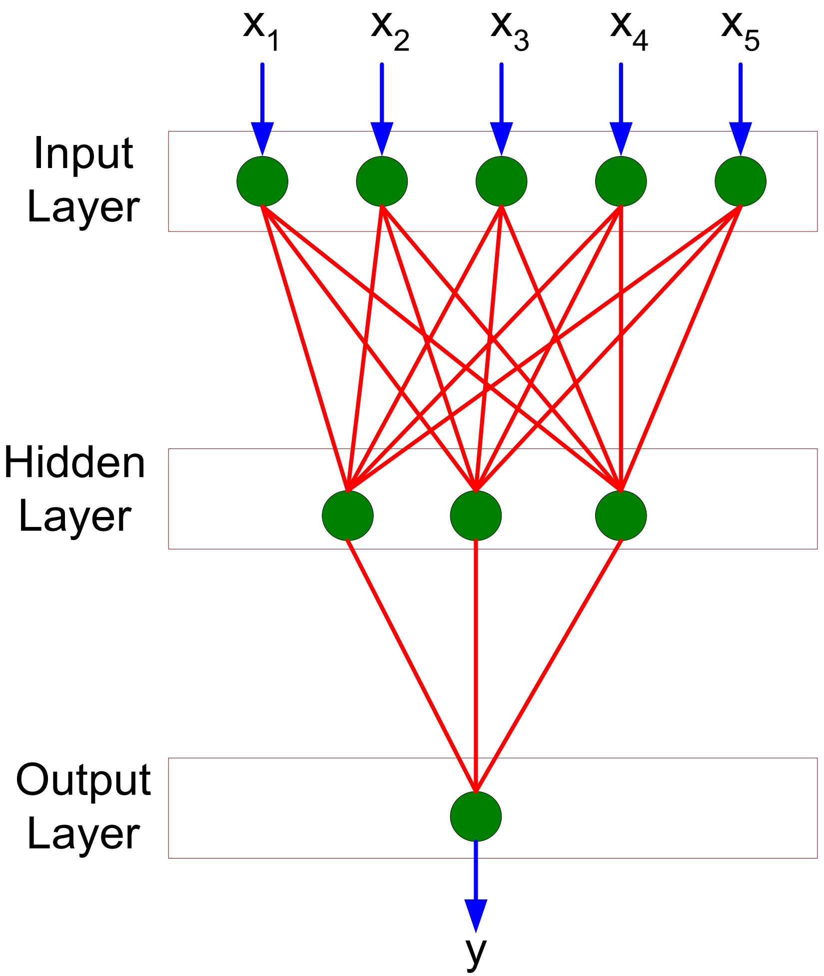
Algorithm for Learning ANN

- Initialize the weights (w_0, w_1, …, w_k), (usually randomly)

- Adjust the weights in such a way that the output of ANN is consistent with class labels of training examples
  - Objective function: $$E = \sum_i \left[Y_i - f(w_i, X_i)\right]^2$$
  - 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:
  $$E = \sum_i \left[Y_i - f(w_i, X_i)\right]^2$$
  - 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

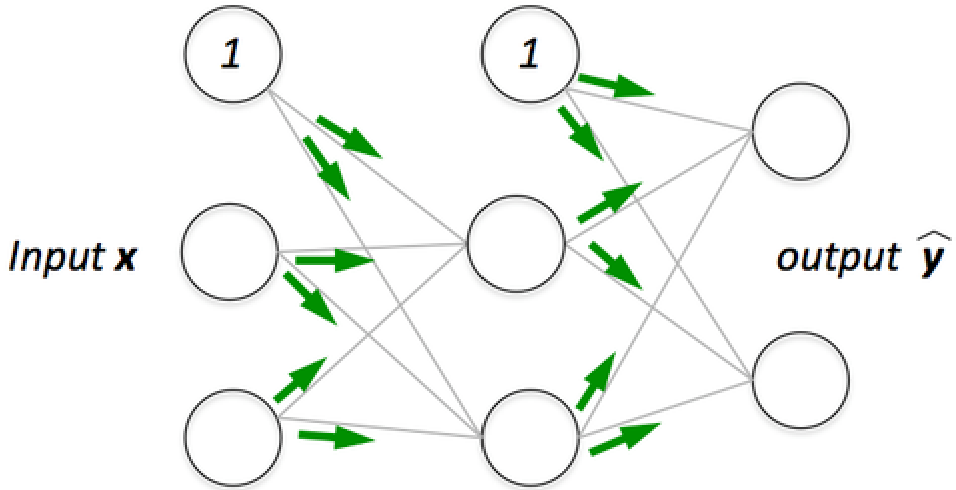
- Sketch of the Backpropagation Algorithm:
  - Present an example to the ANN
  - Compute error at the output layer
  - Distribute error to hidden layer according to weights
    - i.e., the error is distributed according to the contribution of the previous neurons to the result
  - Adjust weights so that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

Training in Batches

- In theory, one could present the examples one after the other
- In practice: use *batches*
  - Neural network gets to see a number of examples (*batch*)
  - All examples in the batch are predicted by the network
  - Errors are accumulated
  - Weights in the neural network are adapted (backpropagated) after predicting all examples in the batch
- Pros: fewer model updates, faster convergence
- Cons: may not find global optimum, harder to handle imbalanced datasets

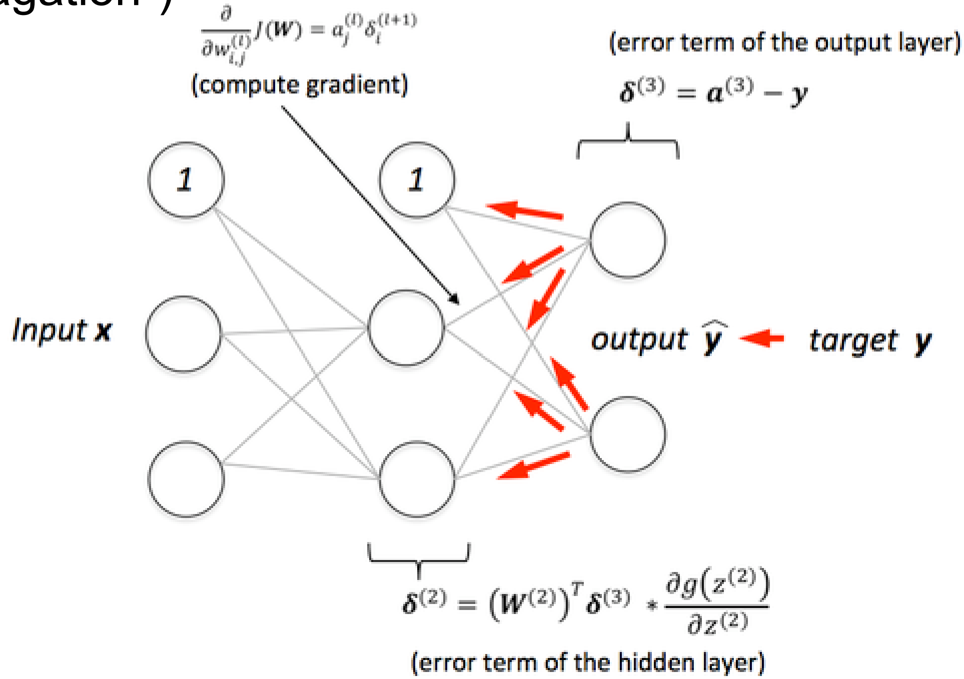
Backpropagation Algorithm

- 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")

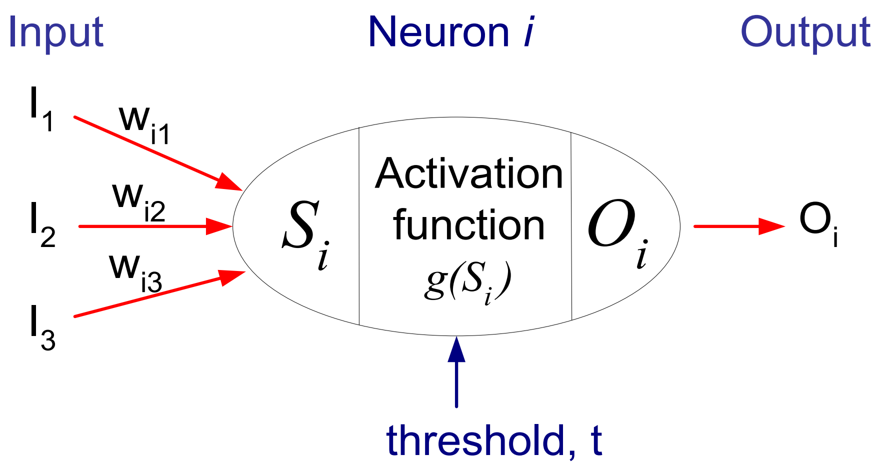


Backpropagation Algorithm – Gradient Descent

- Output of a neuron: $o = g(w_1 i_1 \ldots w_n i_n)$
- Assume the desired output is y, the error is

  $o - y = g(w_1 i_1 \ldots w_n i_n) - y$
- We want to minimize the error, i.e., minimize

  $g(w_1 i_1 \ldots w_n i_n) - y$
- We follow the steepest descent of g, i.e.,
  - the value where g' is maximal



Backpropagation Algorithm – Gradient Descent

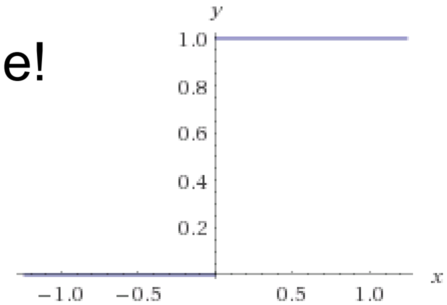
- Hey, wait…
  - the value where g' is maximal

- To find the steepest gradient, we have to differentiate the activation function

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

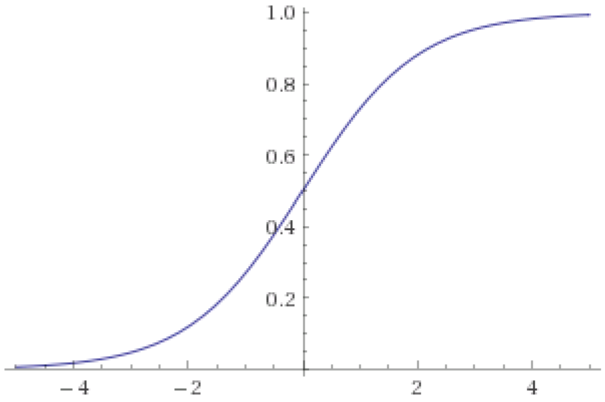
$$\text{where } I(z) = \begin{cases} 1 & \text{if } z \text{ is true} \\ 0 & \text{otherwise} \end{cases}$$

- But I(z) is not differentiable!

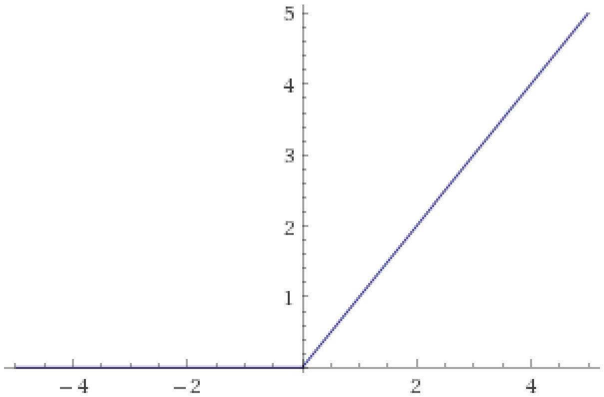


Alternative Activation Functions

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



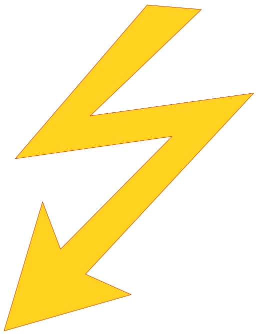
- Rectified Linear Unit (ReLU, since 2010s): max(0,x)



Properties of ANNs and Backpropagation

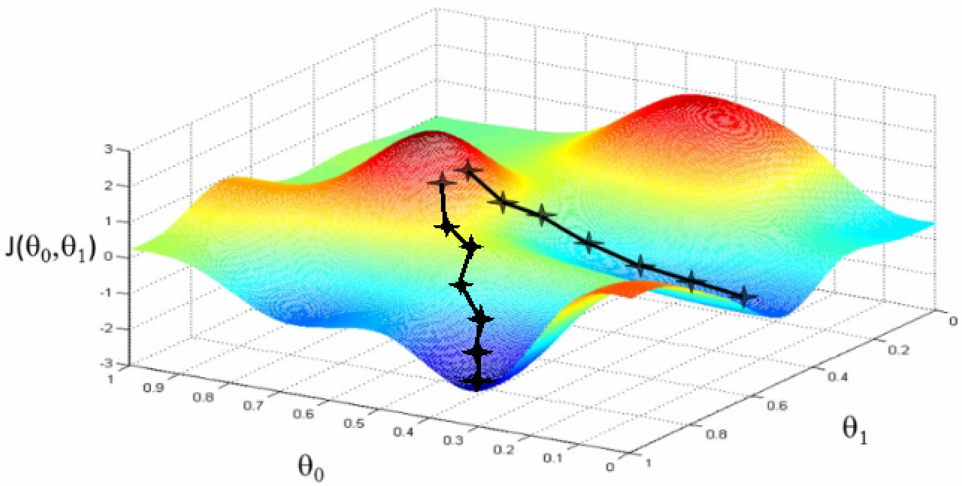
- Non-linear activation function:
  - May approximate any arbitrary function, even with one hidden layer

- 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



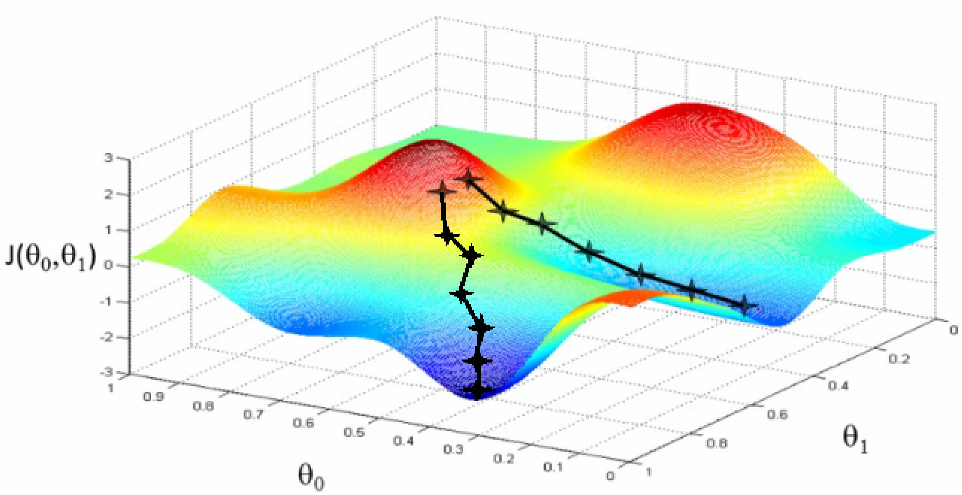
Learning Rate, Momentum, and Local Minima

- 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



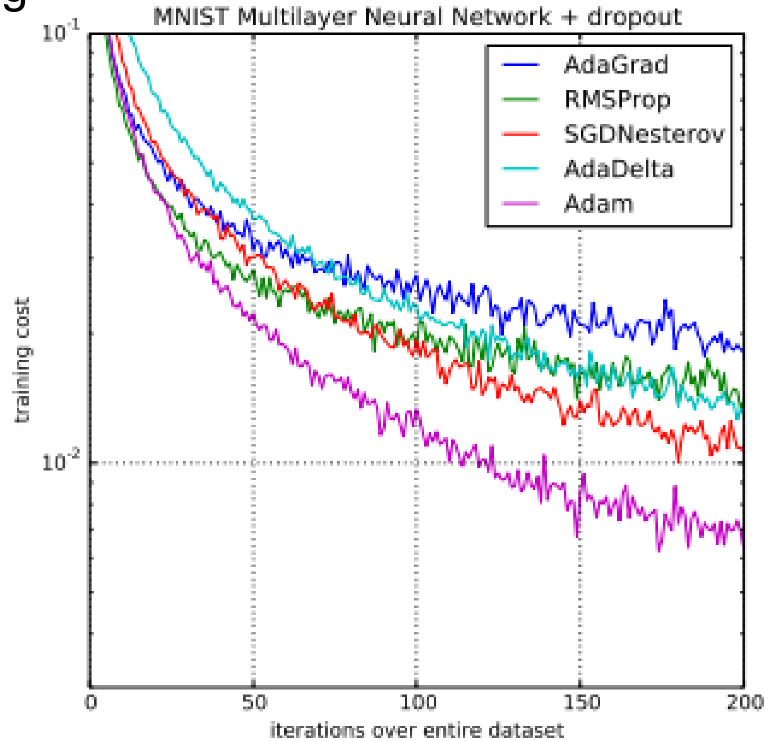
Learning Rate, Momentum, and Local Minima

- Momentum: how much do we change the adaptation of weights
  - Small: allow changes in every direction soon
  - High: keep changing in the same direction for longer
- 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



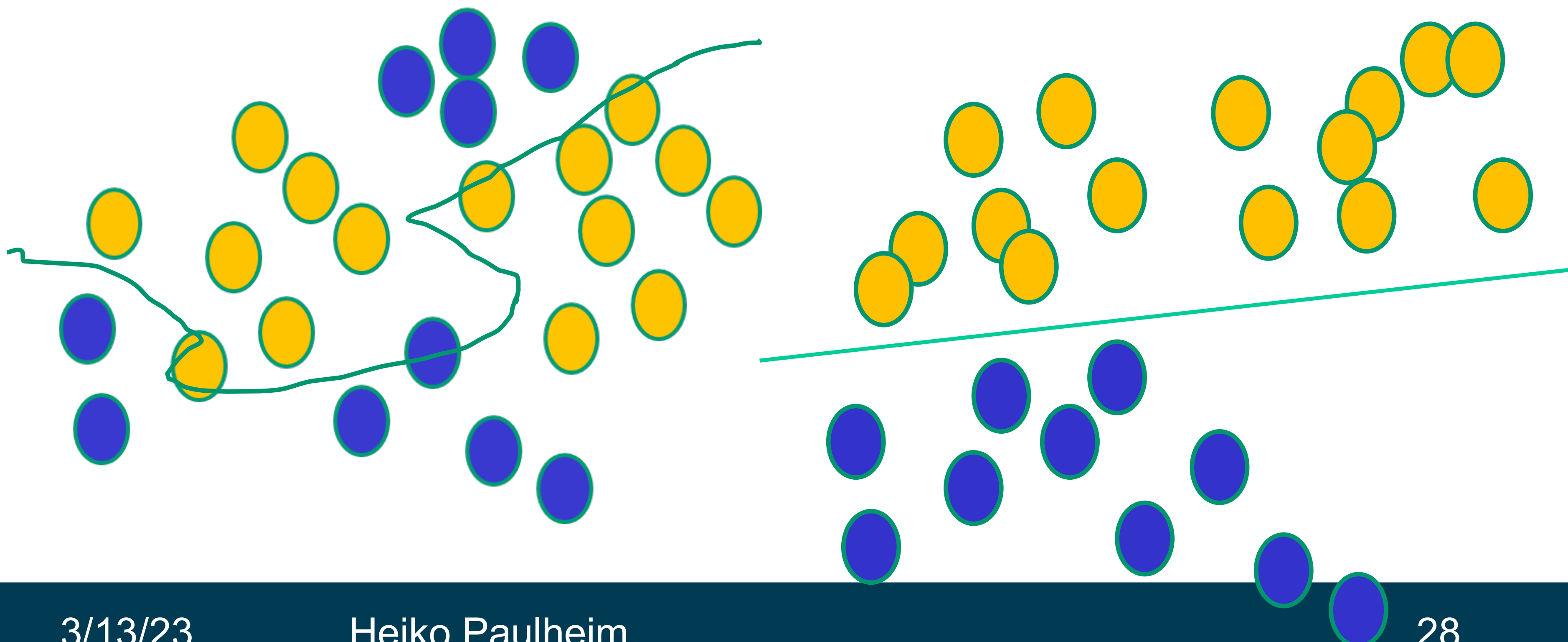
Local Learning Rates

- 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

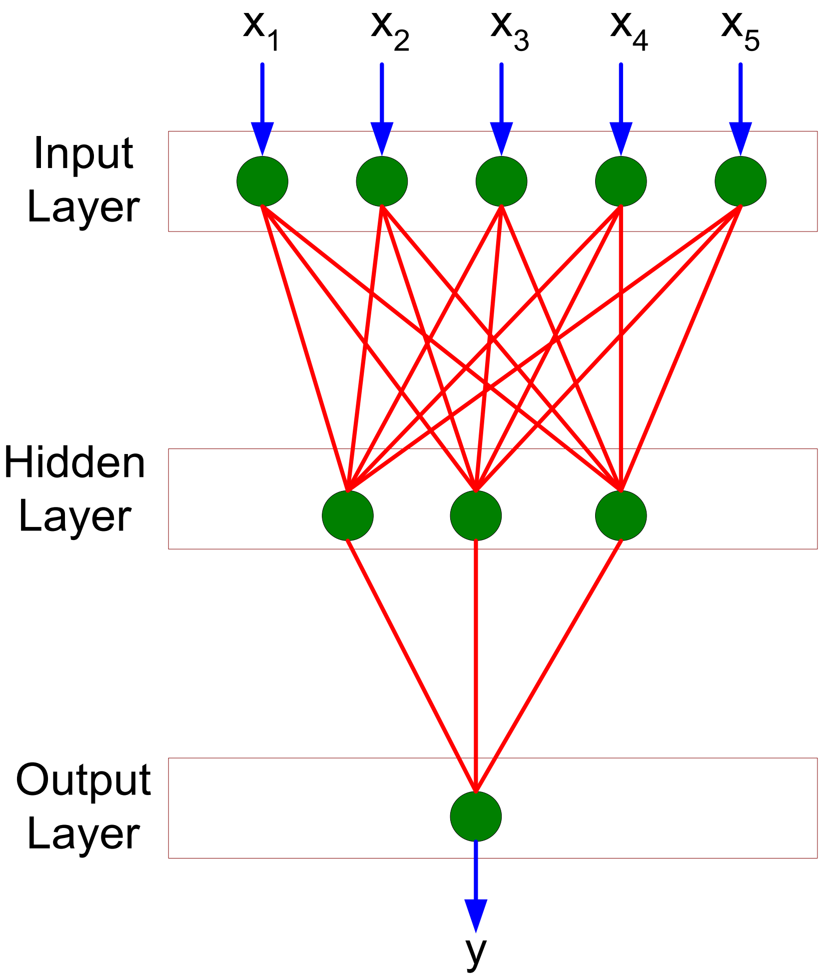
- Idea: reduce the dimensionality of high dimensional data

- 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

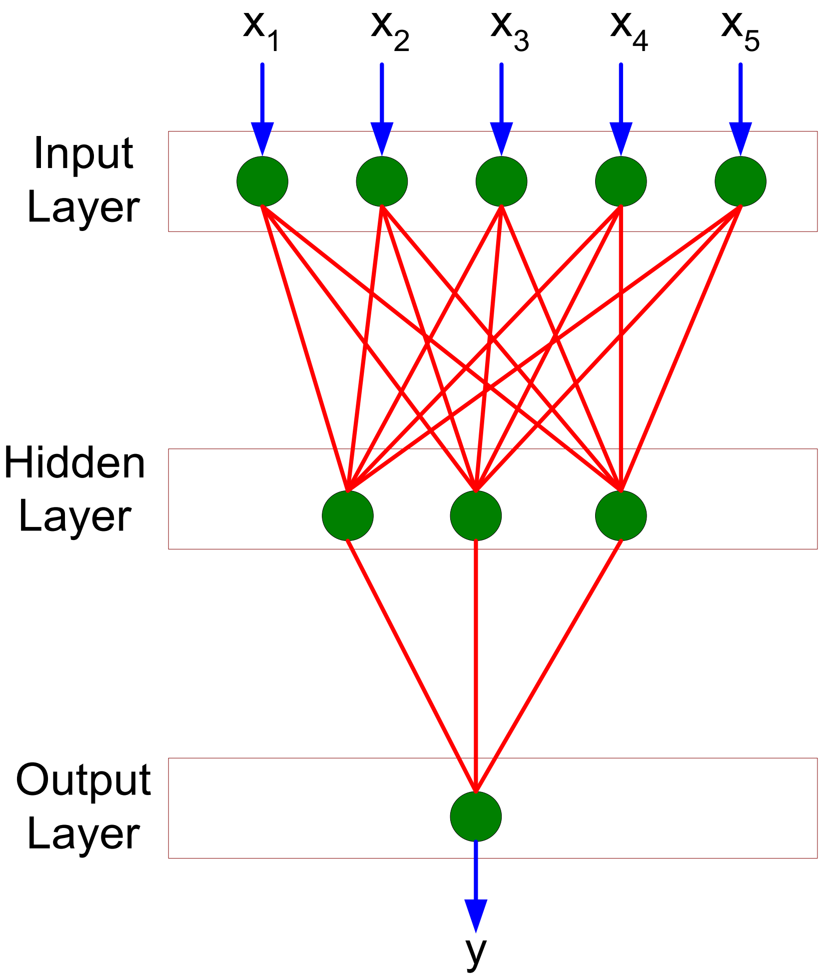
What Happens at the Hidden Layer?

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



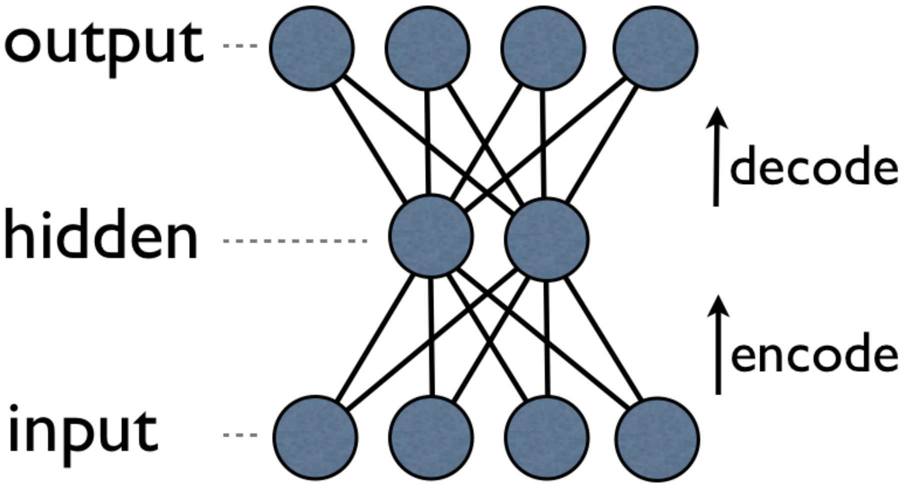
What Happens at the Hidden Layer?

- We create a more compact representation of the dataset
  - Hidden: $h_1 \ldots h_m$
  - 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?



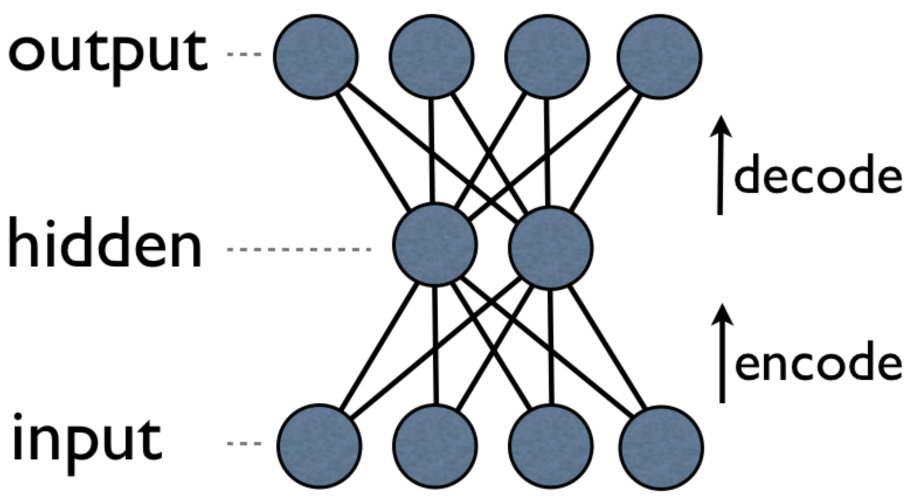
Auto Encoders

- 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



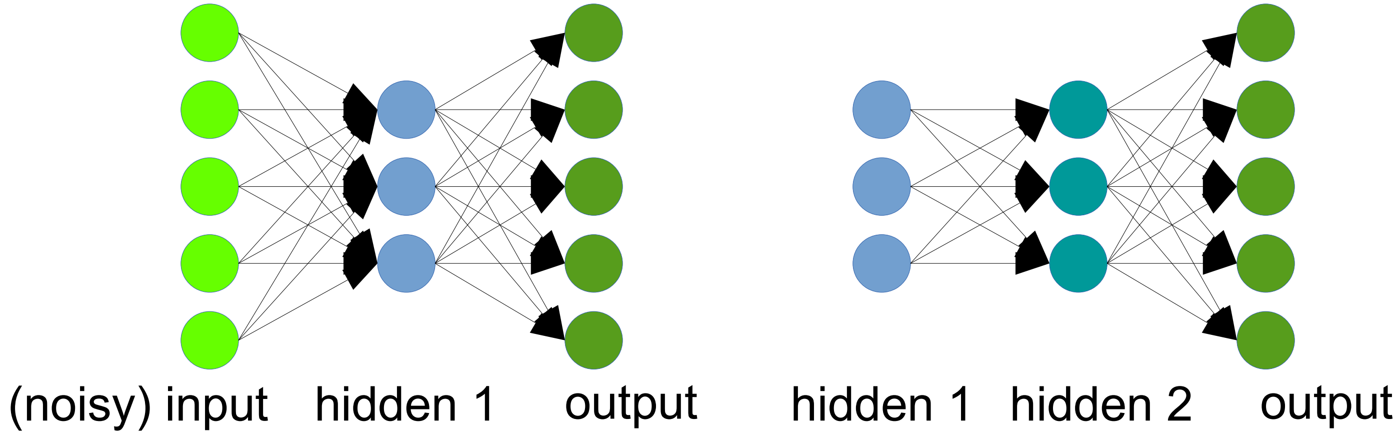
Denoising Auto Encoders

- 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



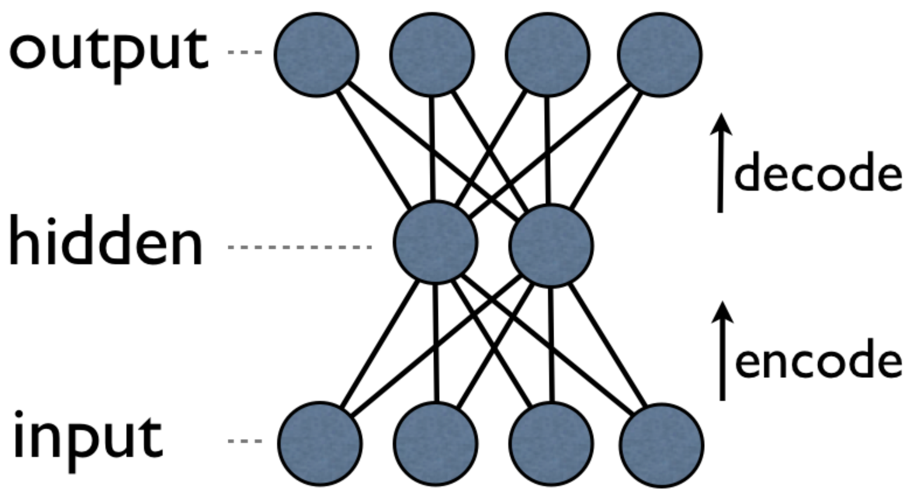
Stacked (Denoising) Auto Encoders

- 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

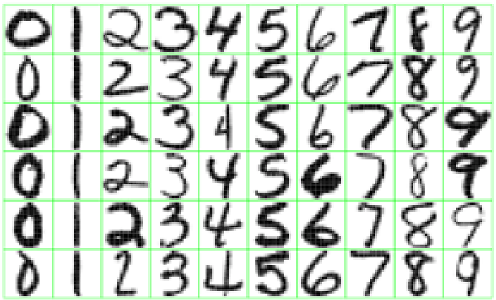
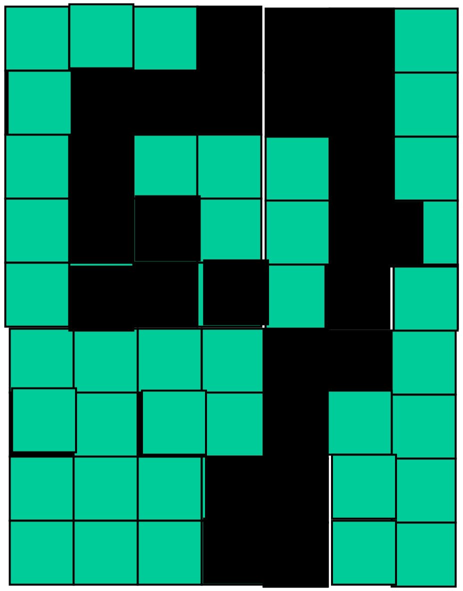
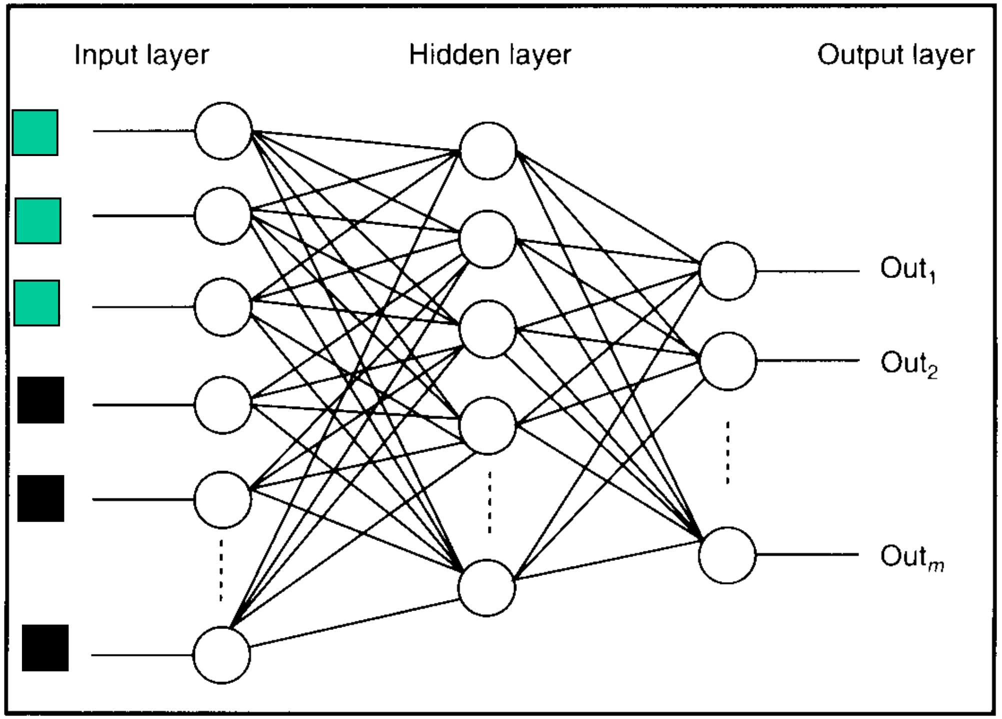


Figure 1.2: *Examples of handwritten digits from U.S. postal envelopes.*



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



From Classifiers to Feature Detectors

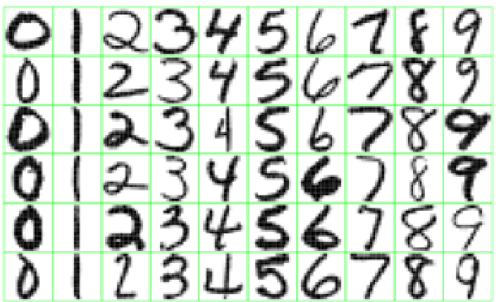
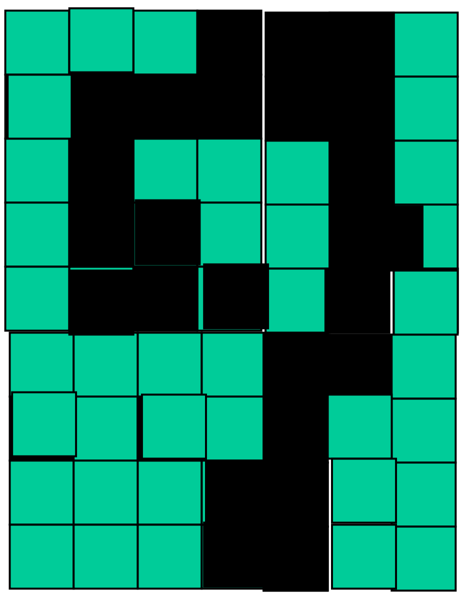
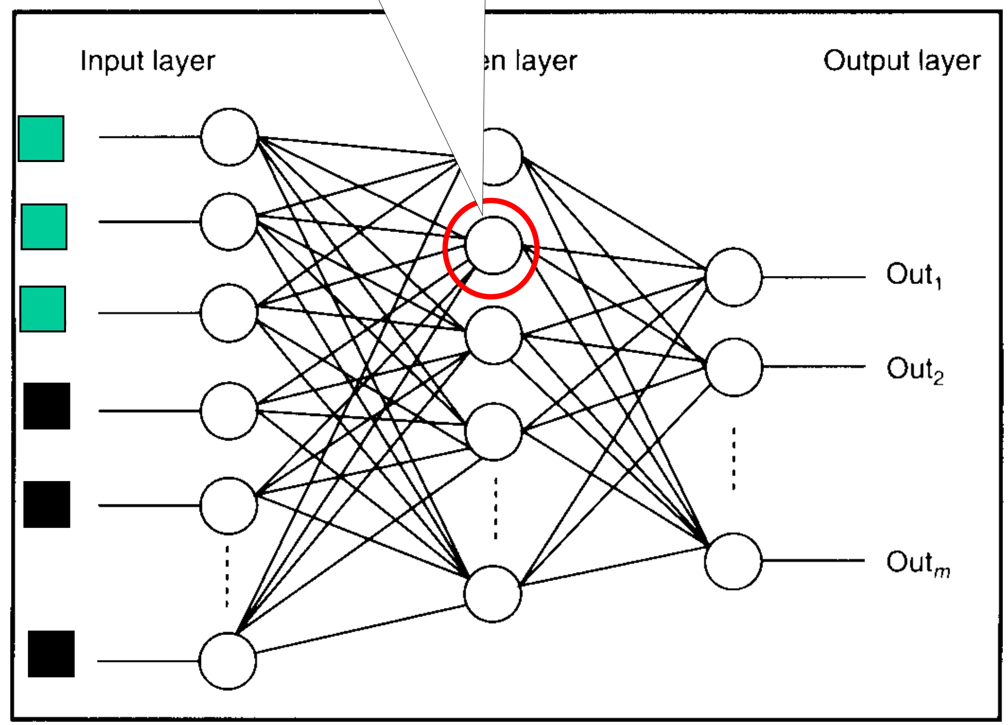


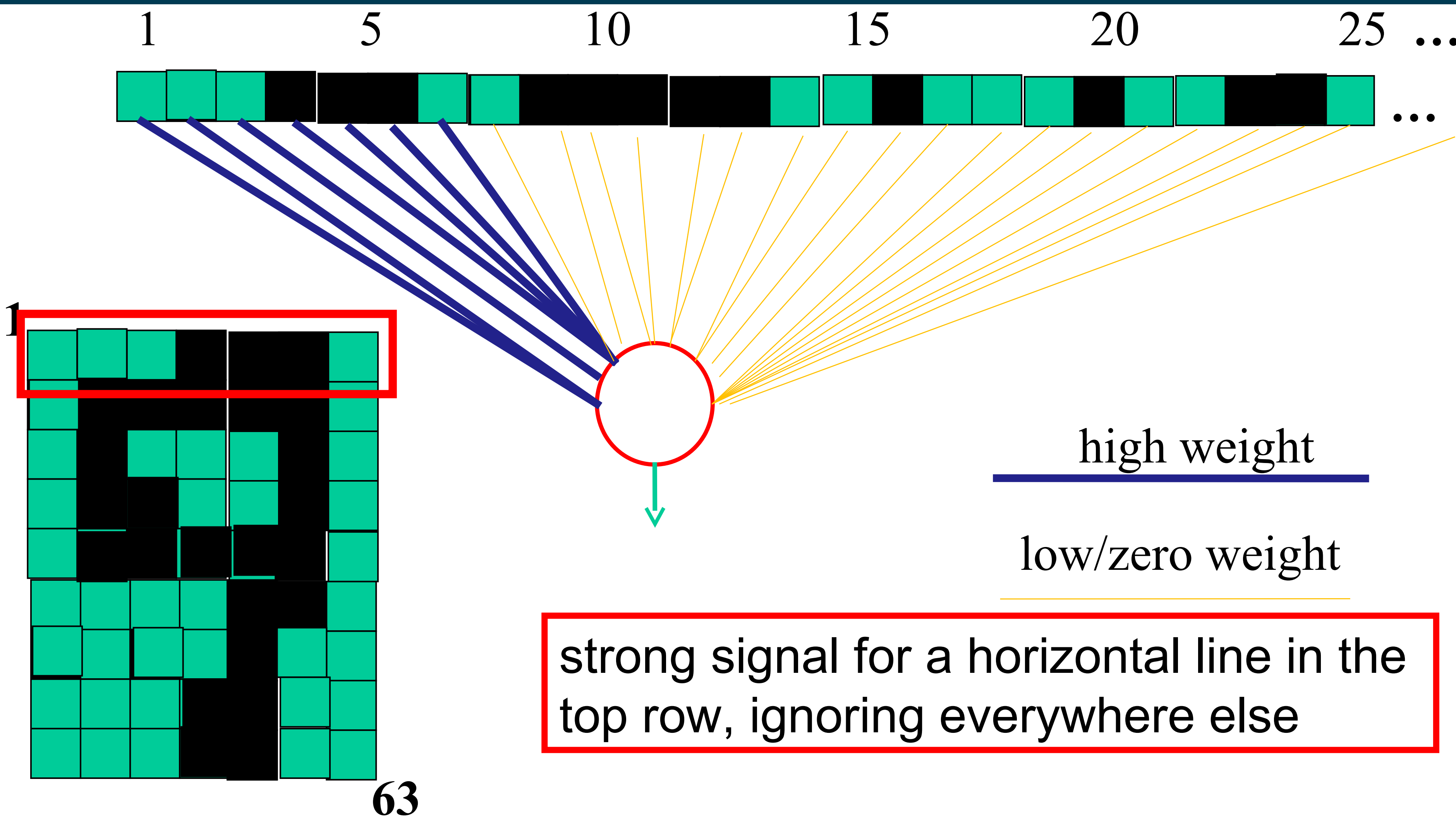
Figure 1.2: *Examples of handwritten digits from U.S. postal envelopes.*



What does a particular neuron do?

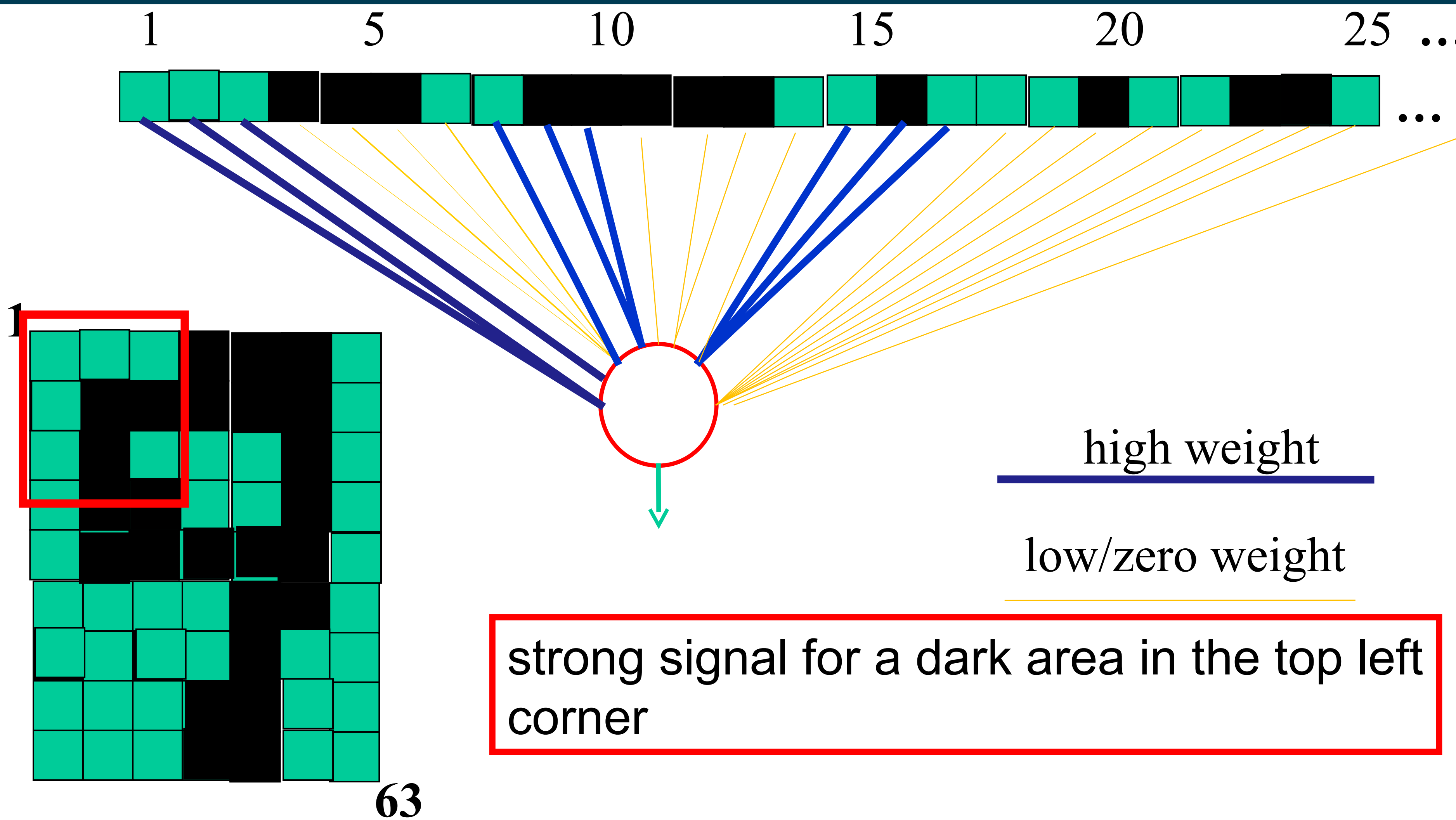


What Happens at the Hidden Layer?



strong signal for a horizontal line in the top row, ignoring everywhere else

What Happens at the Hidden Layer?



strong signal for a dark area in the top left corner

Is that enough? What Features do we Need?

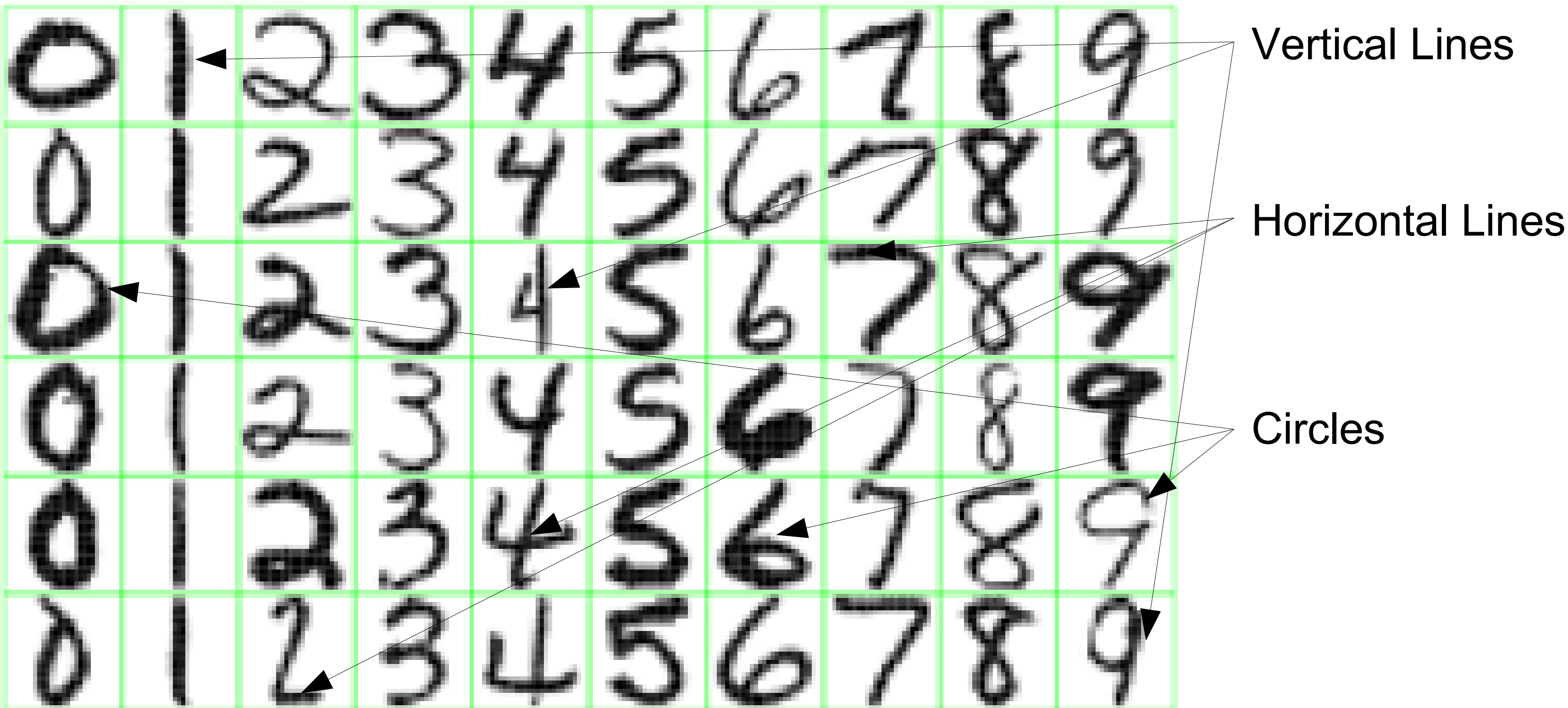


Figure 1.2: *Examples of handwritten digits from U.S. postal envelopes.*

Is that enough? What Features do we Need?

- What we have
  - Line at the top
  - Dark area in the top left corner
  - …

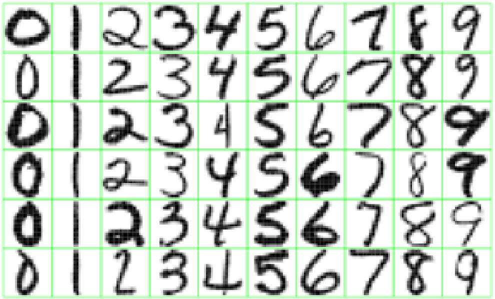
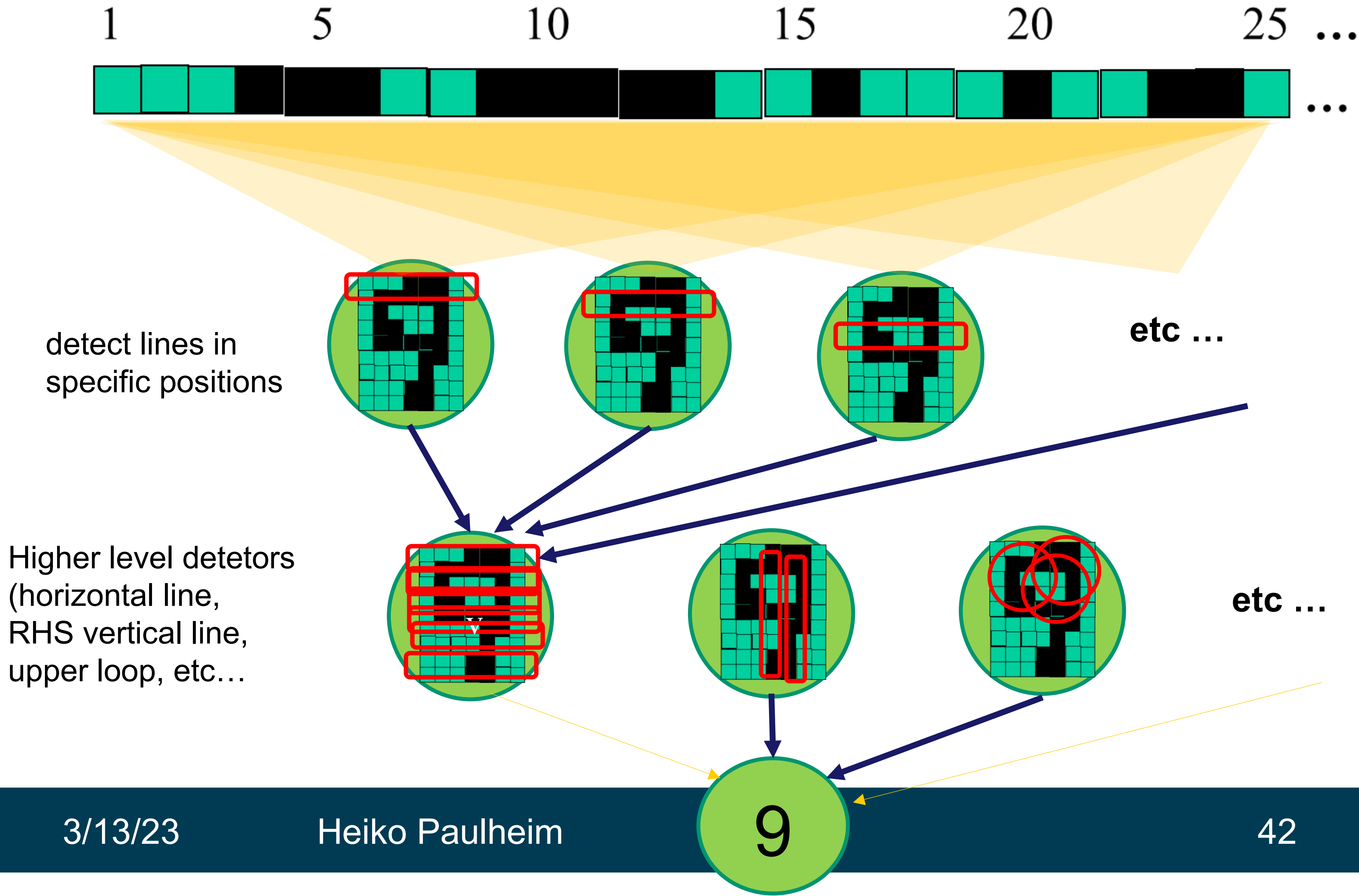


Figure 1.2: *Examples of handwritten digits from U.S. postal envelopes.*

- What we want
  - Vertical Line
  - Horizontal Line
  - Circle

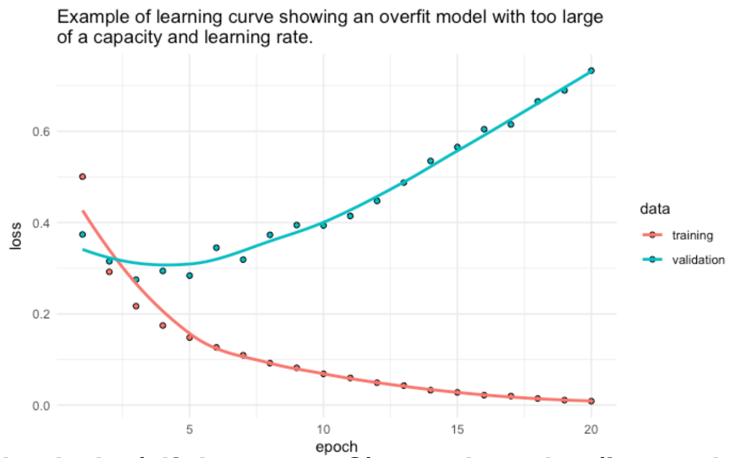
- Challenges
  - Positional variance
  - Color variance

On the Quest for Higher Level Features



Memorizing vs. Learning

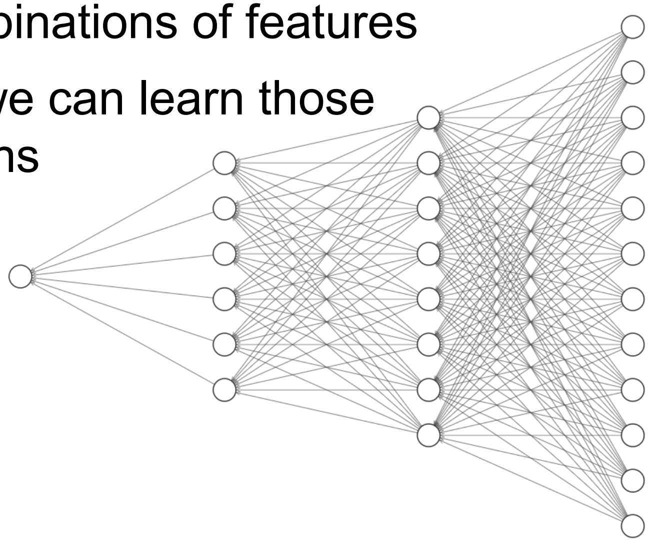
- The goal of training a neural network is learning a *generalized* model
  - should classify unseen examples
- The opposite of generalization is memorization
  - Model learns training examples “by heart”
  - Lesser performance on unseen examples
  - Indicator: performance increase on training set, decrease on test set



https://rstudio-conf-2020.github.io/dl-keras-tf/notebooks/learning-curve-diagnostics.nb.html

Memorizing vs. Learning

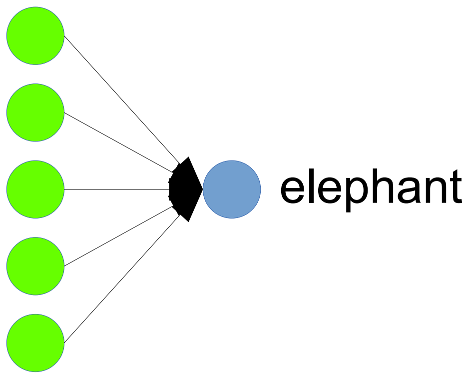
- Hidden layers define the *capacity* of a neural network
  - Roughly: how complex can the patterns be that are stored
- Too low capacity: underfitting
  - Patterns that, e.g., separate classes have a certain complexity
  - We need enough hidden neuron connections to “store” those patterns
- Too high capacity: overfitting
  - Examples can be identified by certain combinations of features
  - With enough hidden neuron connections, we can learn those combinations instead of generalized patterns



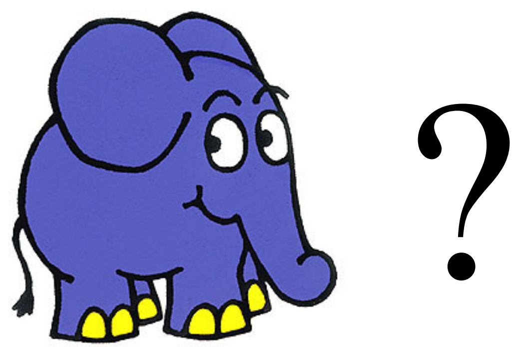
Regularization with Dropout

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

- Elephant: five features in the highest level layer
  - big object
  - grey
  - trunk
  - tail
  - ears

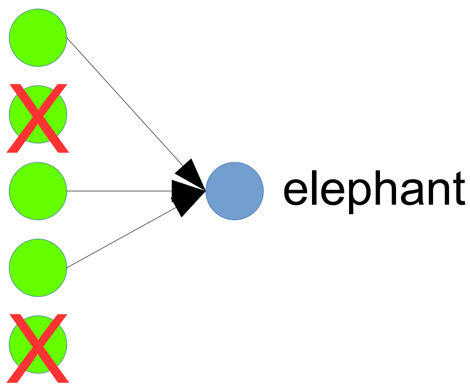


- Possible tendency to overfit:
  - expect all five to fire



Regularization with Dropout

- Regularization
  - Randomly deactivate hidden neurons when training an example
  - E.g., factor α=0.4: deactivate neurons randomly with probability 0.4
- Example:
  - big object
  - grey
  - trunk
  - tail
  - ears



Regularization with Dropout

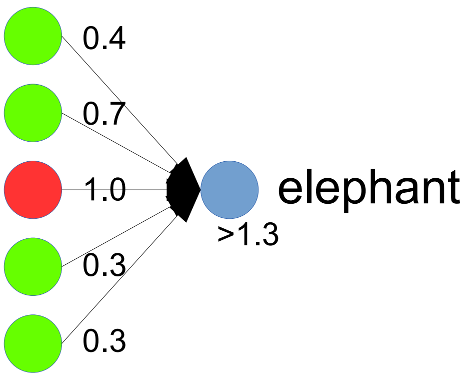
- Regularization
  - Randomly deactivate hidden neurons when training an example
  - E.g., factor α=0.4: deactivate neurons randomly with probability 0.4
- Result:
  - Learned model is more robust, less overfit

- 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+α)

- Example:
  - big object
  - grey
  - trunk
  - tail
  - ears

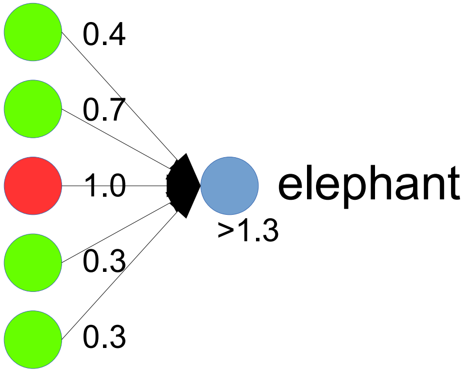


without correction: 0.4+0.7+0.3+0.3 = 1.7>1.3

Regularization with Dropout

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

- Example:
  - big object
  - grey
  - trunk
  - tail
  - ears



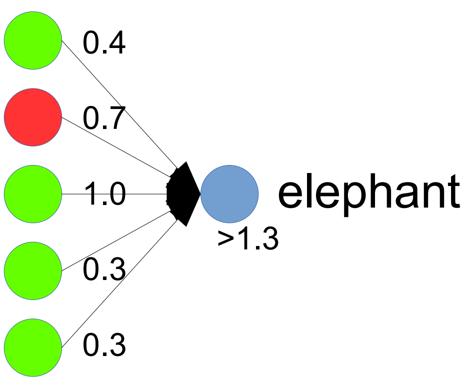
With correction: (5/7)*(0.4+0.7+0.3+0.3) = 1.21<1.3

X

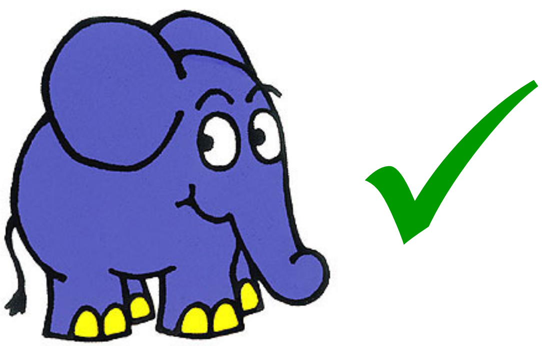
Regularization with Dropout

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

- Example:
  - big object
  - grey
  - trunk
  - tail
  - ears



(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

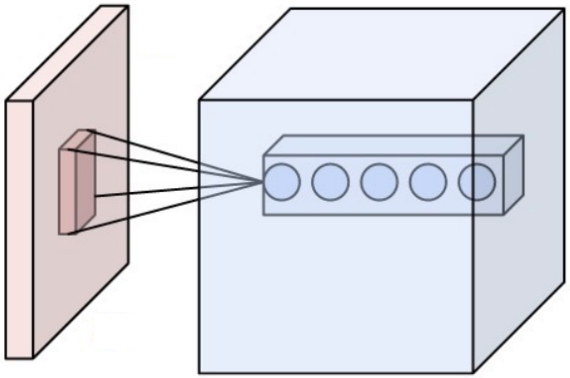
Architectures: Convolutional Neural Networks

- Two parts:
  - Convolution layer
  - Pooling layer

- Stacks of those are usually used

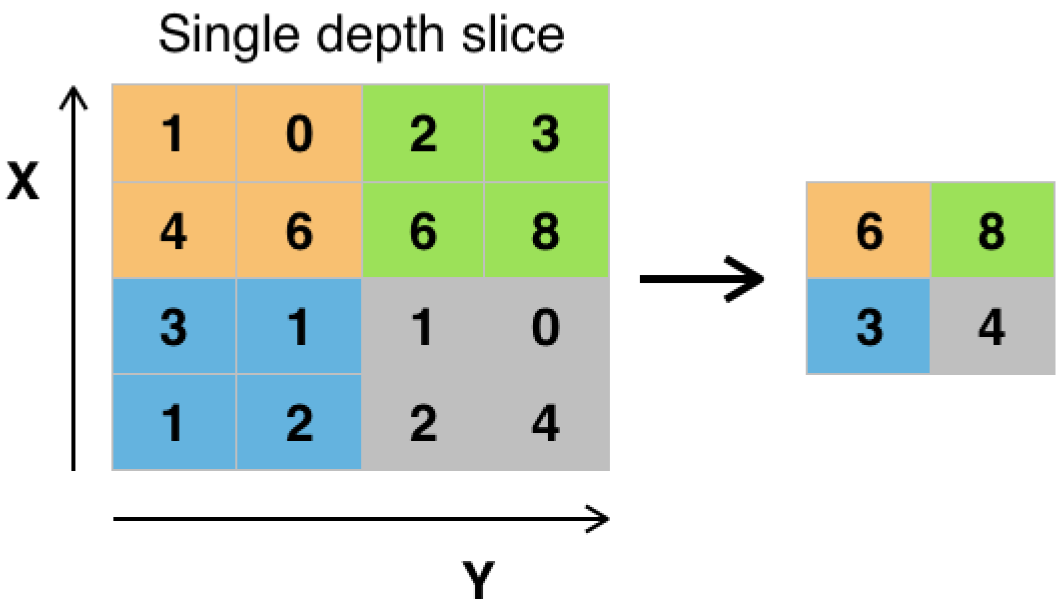
Architectures: Convolutional Neural Networks

- Convolution layer
  - Each neuron is connected to a small n x 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)



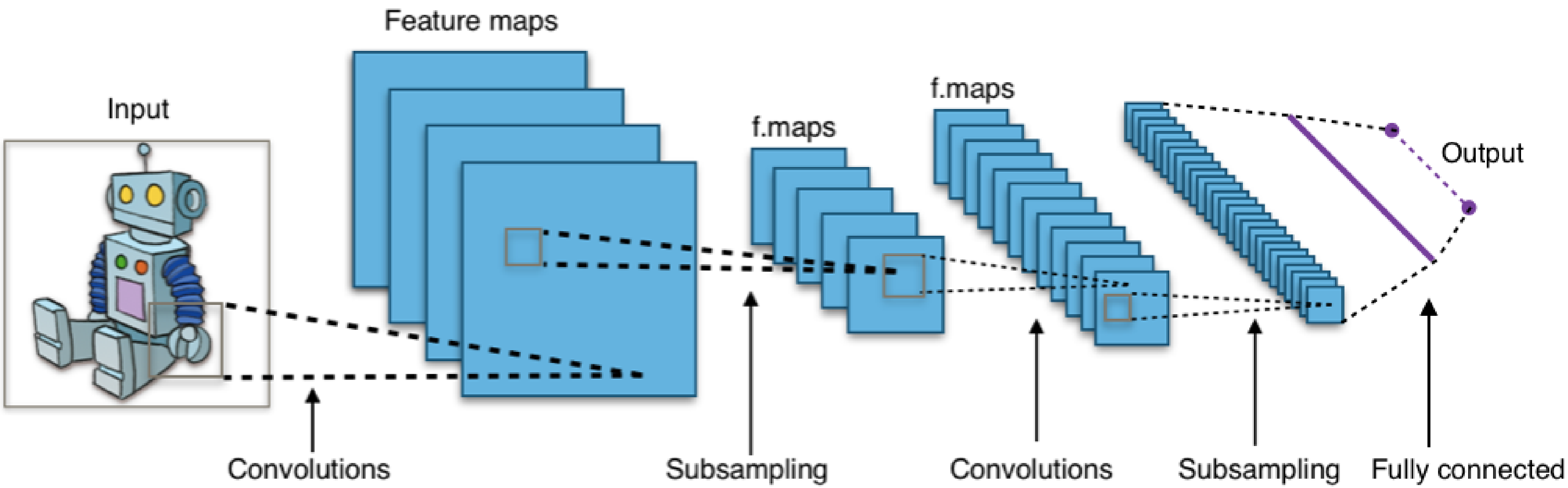
Architectures: Convolutional Neural Networks

- Pooling layer (aka *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



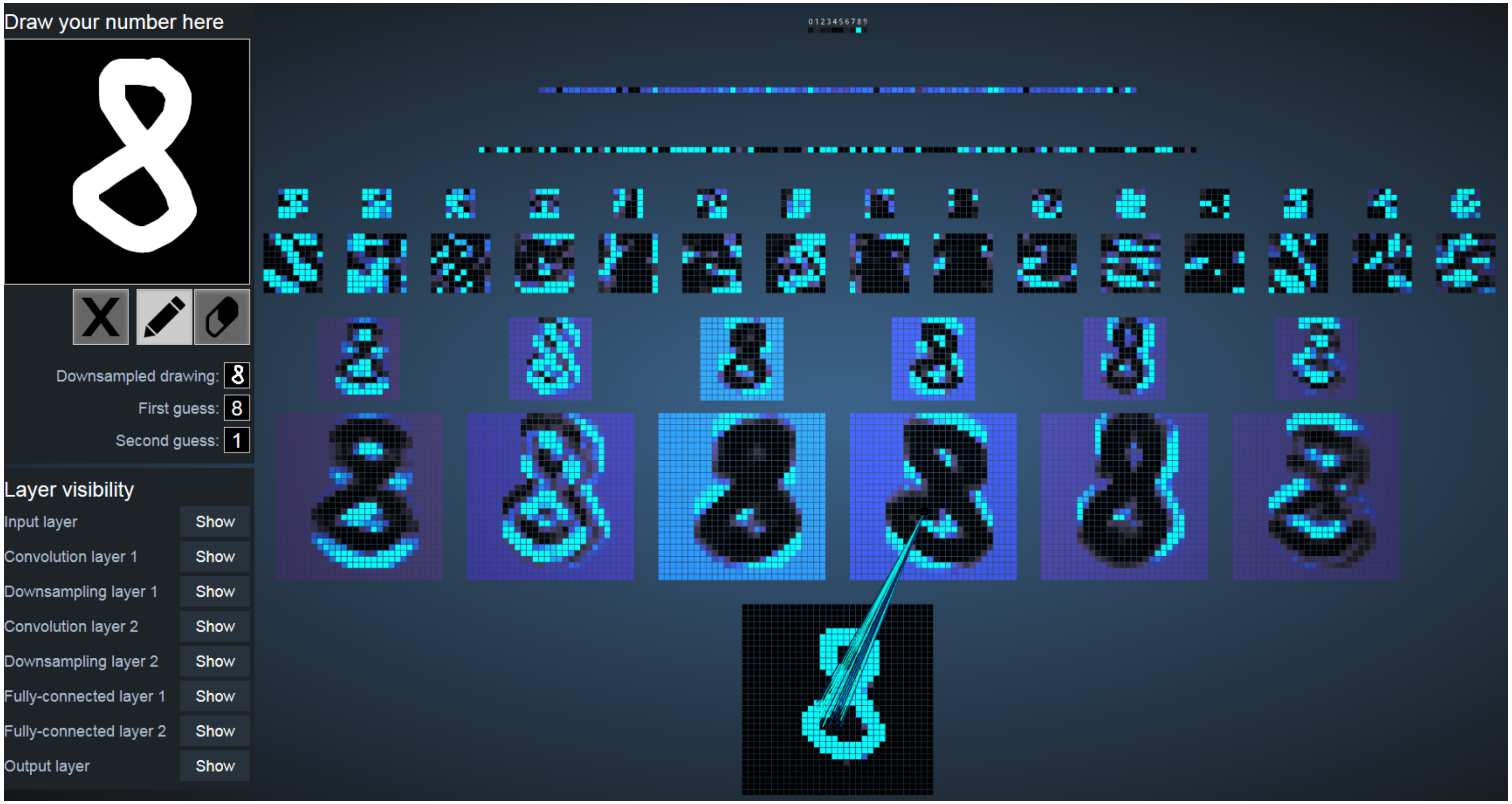
Architectures: Convolutional Neural Networks

- 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)
  - 16x16x256x8M = still 526 billion weights
- But: neurons for the same hidden feature share their weight
  - Thus, it's just 16x16x256 = 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

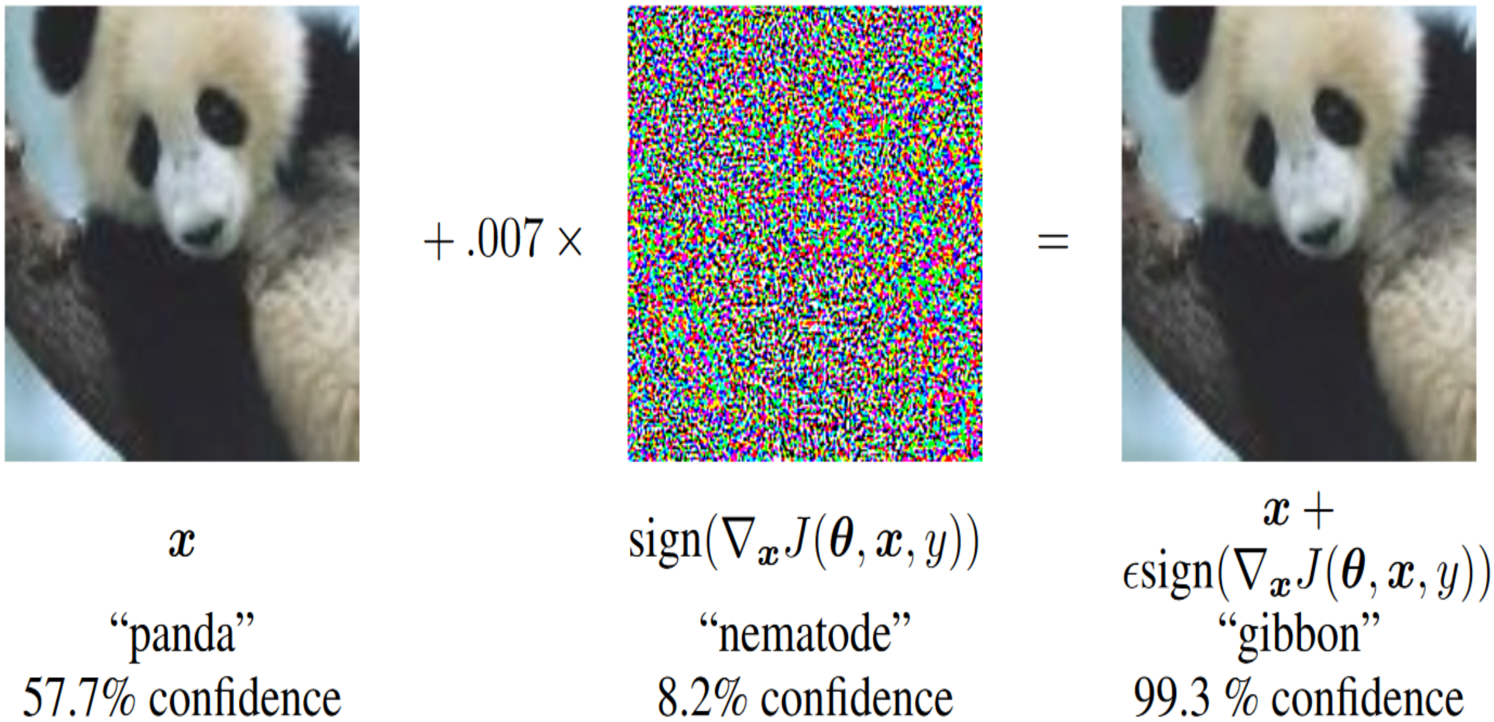
- Can be used as a pre-trained network

What does an Artificial Neural Network Learn?

What does an Artificial Neural Network Learn?

What does an Artificial Neural Network Learn?

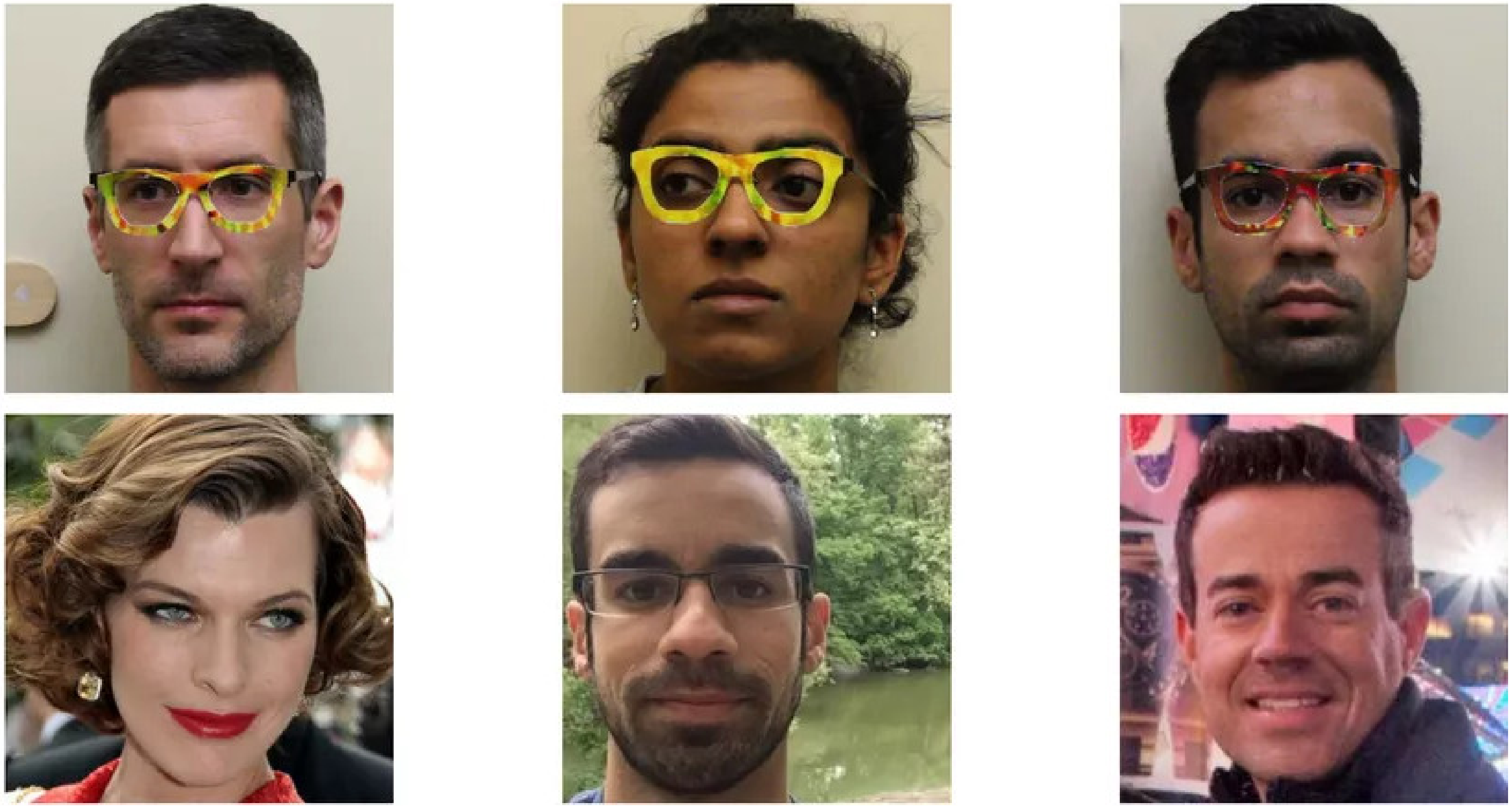
- Image recognition networks can be attacked
  - changing small pixels barely noticed by humans



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

Possible Implications

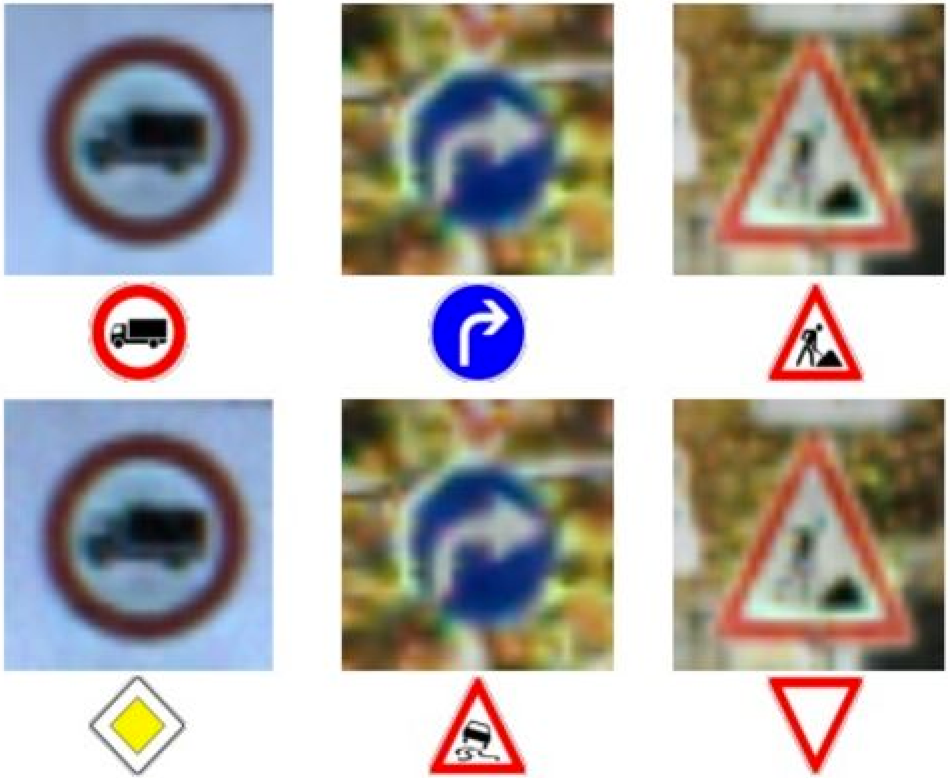
- Face Detection



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

Possible Implications

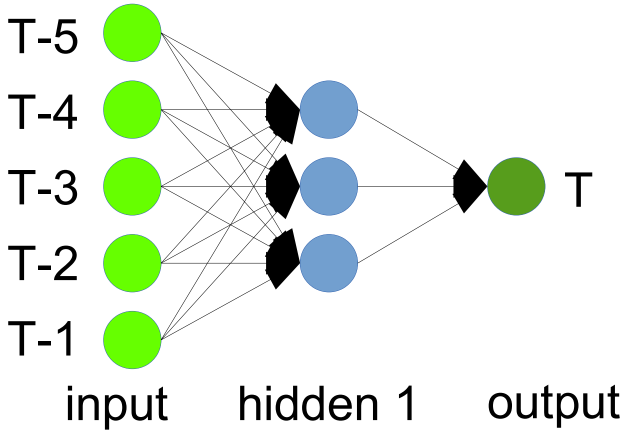
- Autonomous Driving



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

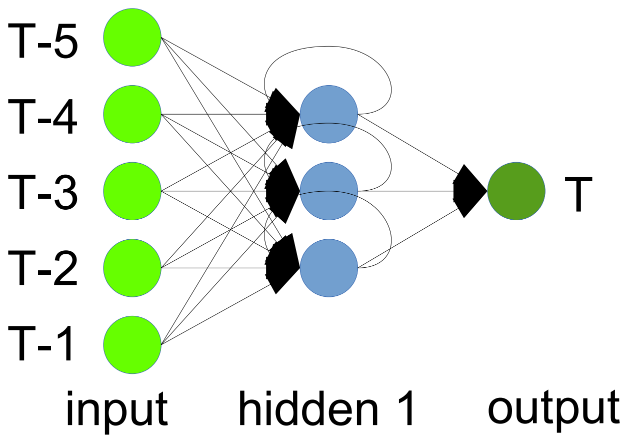
Using ANNs for Time Series Prediction

- 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)



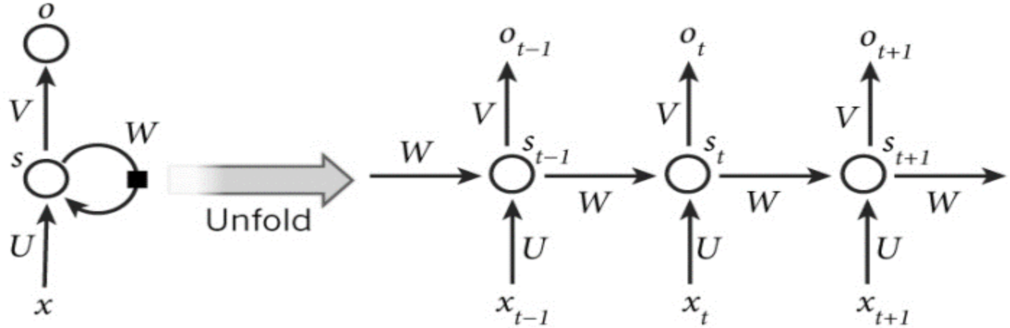
Using ANNs for Time Series Prediction

- 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

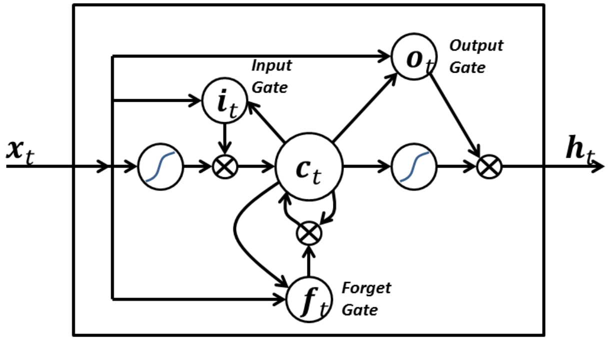


Long Short Term Memory Networks (LSTM)

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

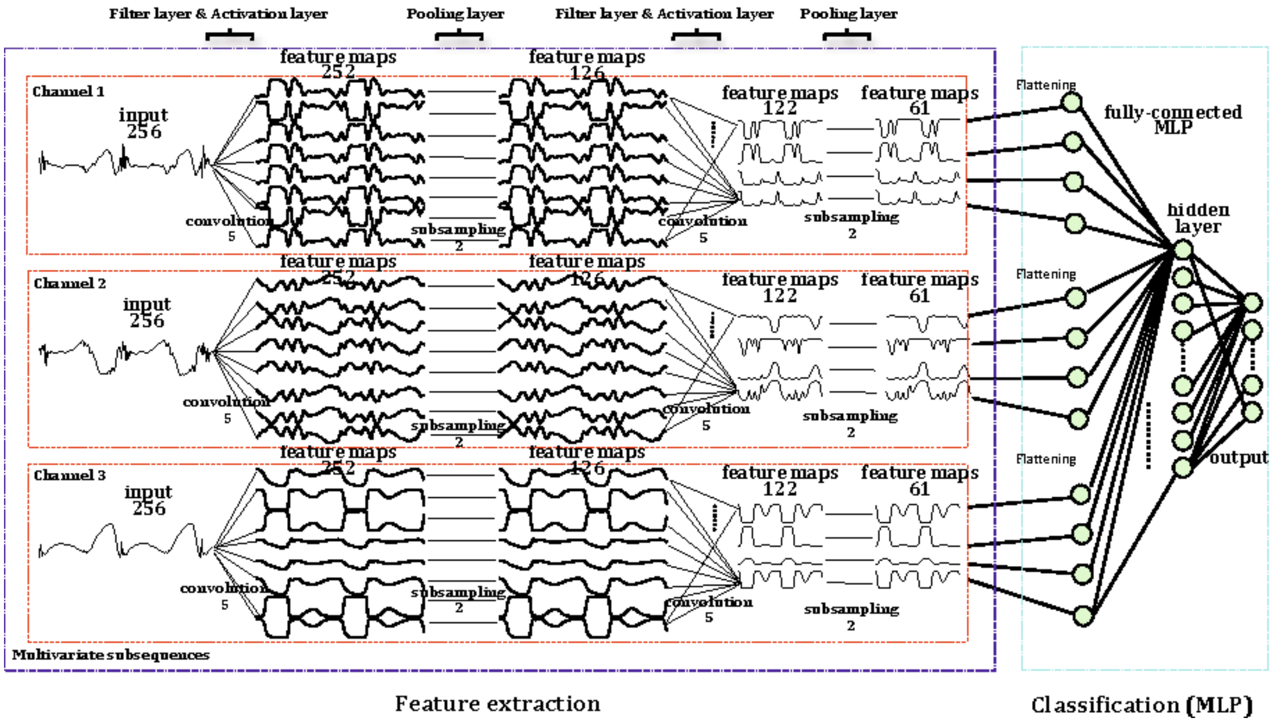


- LSTMs are special recurrent neural networks



CNNs for Time Series Prediction

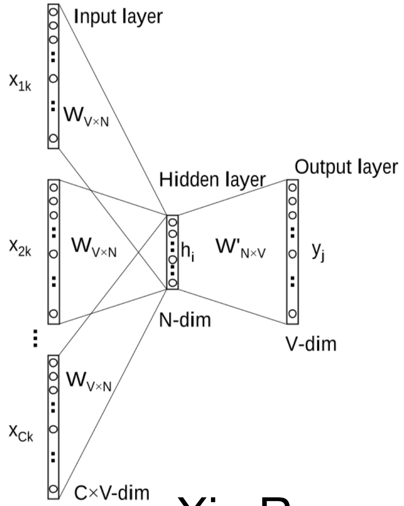
- Notion: time series also have typical features
  - Think: trends, seasonal variation, ...

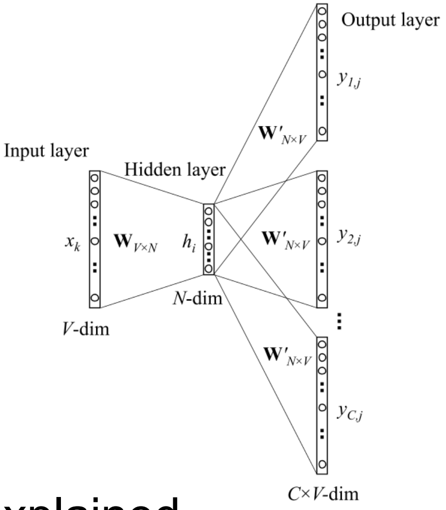


Zheng et al.: Time Series Classification Using Multi-Channels Deep Convolutional Neural Networks, 2014

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

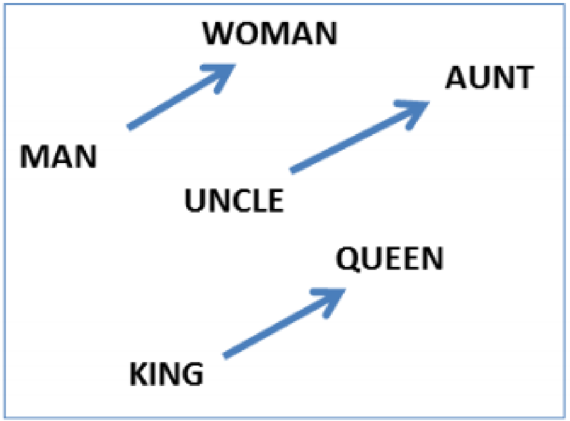


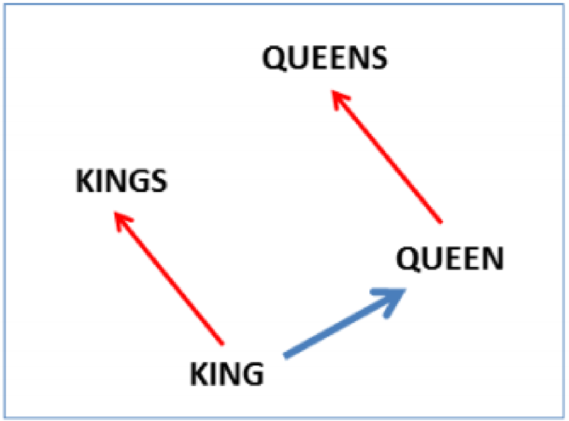


Xin Rong: word2vec parameter learning explained

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

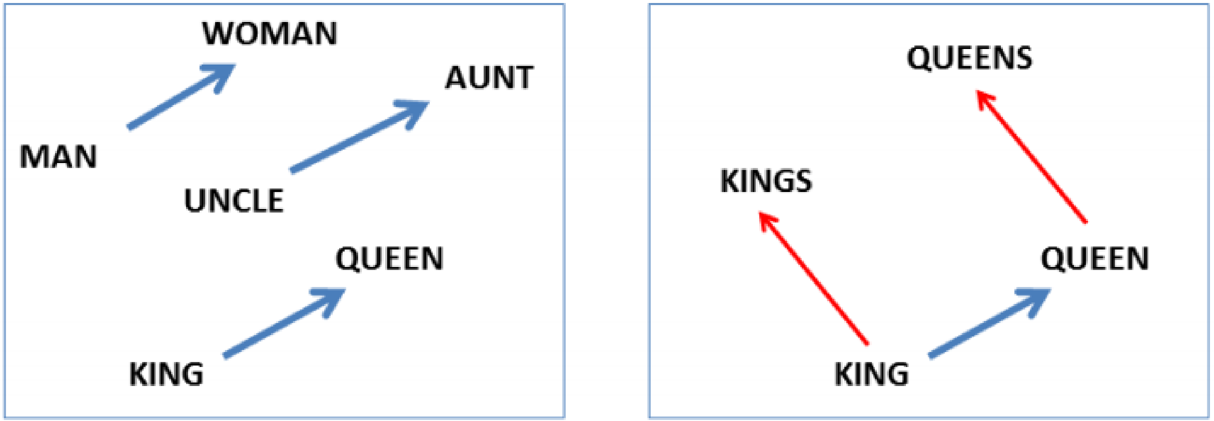




(Mikolov et al., NAACL HLT, 2013)

word2vec

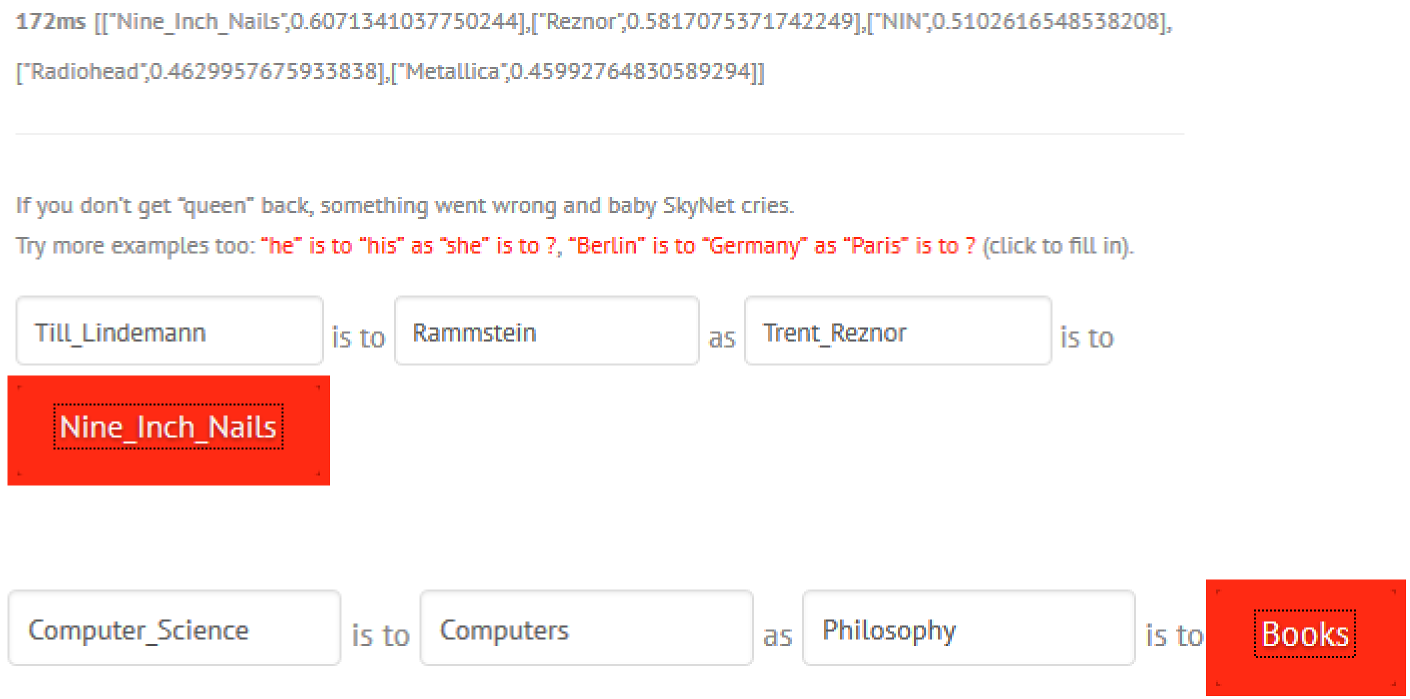
- Arithmetics are possible in the vector space
  - king – man + woman ≈ queen
- This allows for finding analogies:
  - king:man ↔ queen:woman
  - knee:leg ↔ elbow:forearm
  - Hillary Clinton:democrat ↔ Donald Trump:Republican



(Mikolov et al., NAACL HLT, 2013)

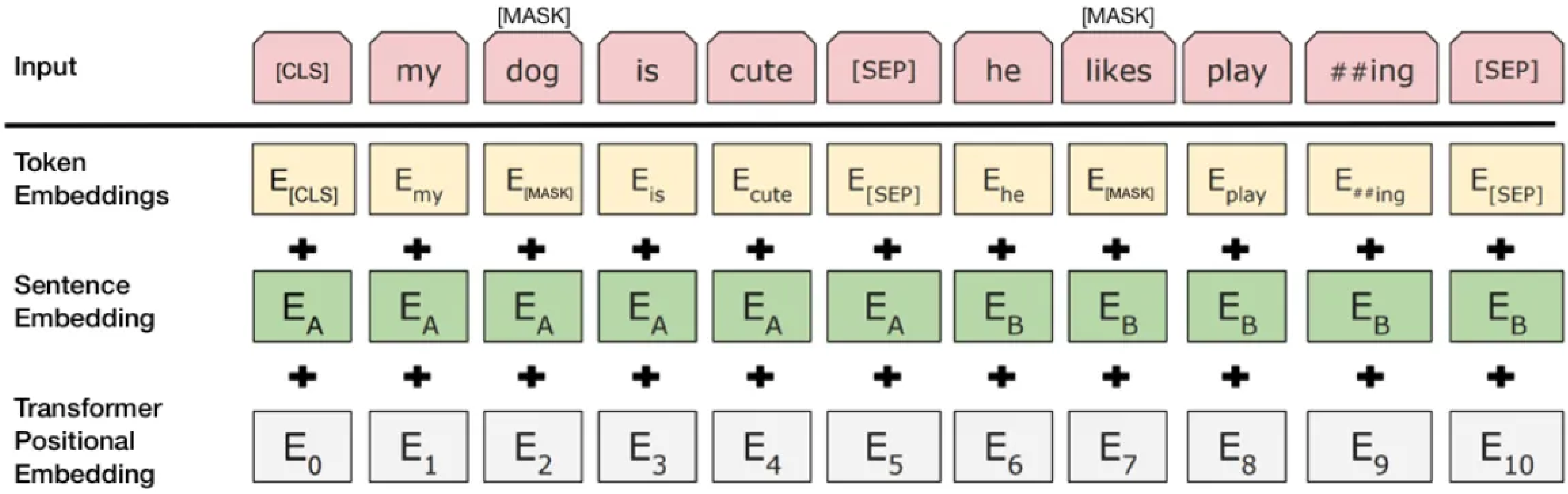
word2vec

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



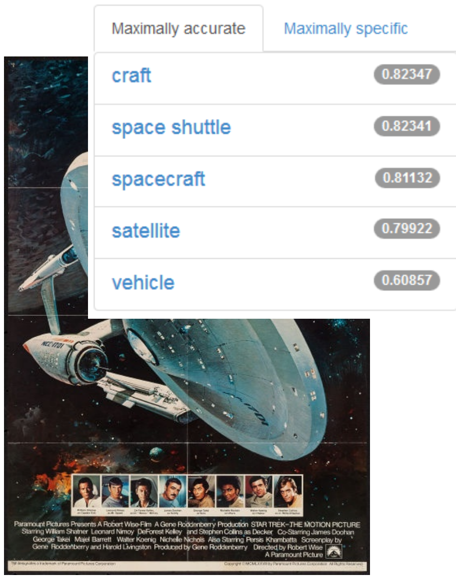
BERT

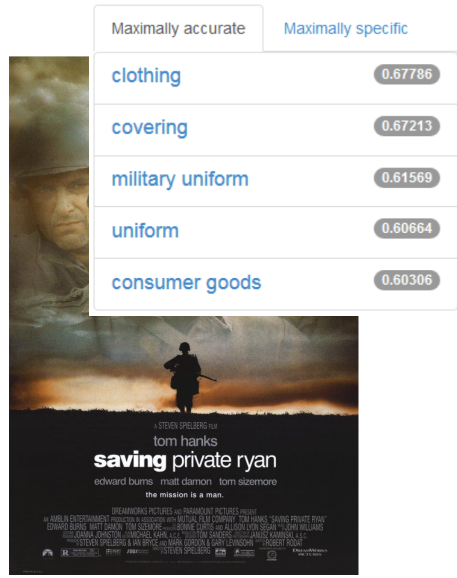
- Learns a representation of words in context
  - Unlike word2vec: one fixed representation per word
  - Larger training corpus required

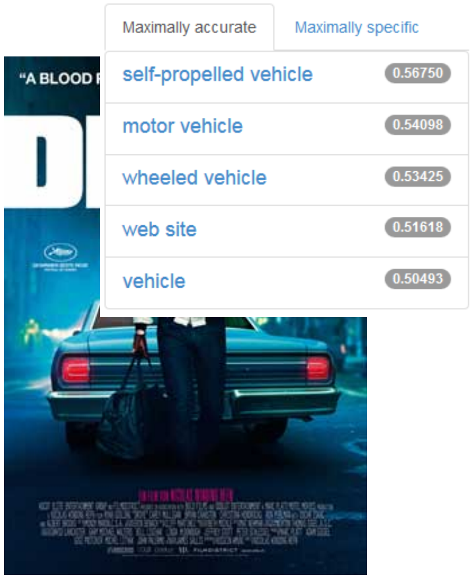


Reusing Pre-trained Networks

- 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



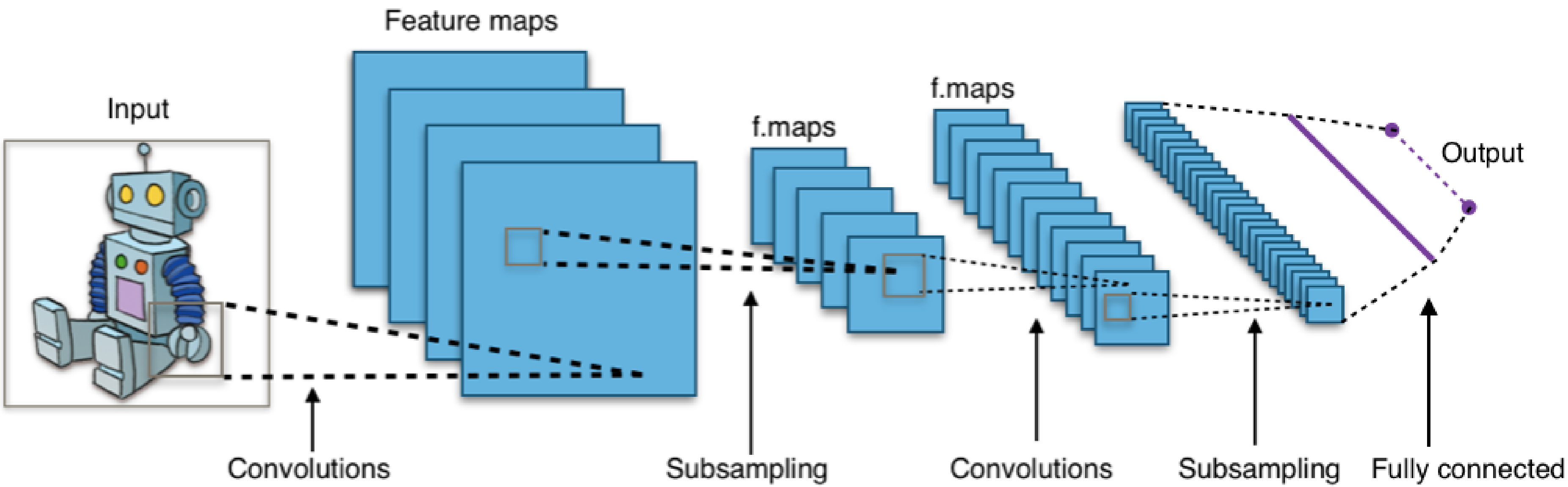




http://demo.caffe.berkeleyvision.org/

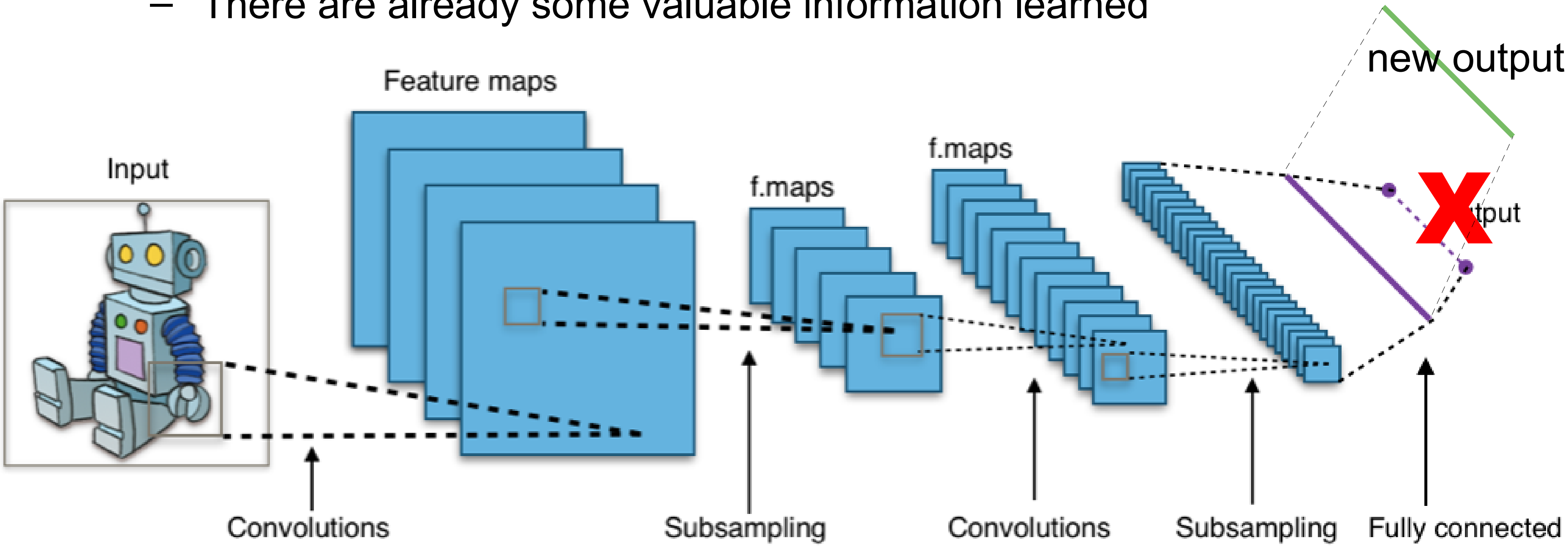
Reusing Pre-trained Networks

- In many cases, the last or second-to last layer are reused
- Fine-tuning on a task at hand often leads to advantages
  - i.e., use the trained network, add a new classification layer, and present examples
  - Referred to as “transfer learning”



Transfer Learning with Pretrained Networks

- Pre-trained neural networks can be (re)used for other tasks
  - They can also be retrained, using the pre-training as initialization
  - Sometimes, different layers are frozen
- Rationale:
  - There are already some valuable information learned



Summary

- Artificial Neural Networks
  - Powerful learning tool, approximates arbitrary functions or boundaries
  - Lots of hyperparameters: learning rate, batch size, drop out, ...
- 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?