CS 412

APRIL 23RD – INTRO TO DEEP LEARNING

Administrivia

HW4 Due Tonight OH 5-7

Midterm almost finished

- Grades back by tonight (late)
- Solution video tomorrow

HW5 Posted Today ← unsupervised learning

slightly shorter

- Due next Thursday

Final Exam

- Current plan per the general final schedule: 24 hour take-home exam on Wednesday May 6th
- Midnight-to-midnight CDT ← expecting the exam to take about 2hrs
- If this doesn't work scheduling-wise, let me know ASAP

→ Course evaluations

especially online class recommendations

Remainder of the course

Deep learning introduction

- Convolutional Neural Networks
- Recurrent Neural Networks

Reinforcement Learning

- Active v. Passive
- Policy setting

Ethics

- Reporting responsibilities
- High impact data science

Next Thursday
Ethics + exam review
+ an additional review

Topics that we missed

Statistical Inference

- Maximum likelihood estimation
  - Frequentist inference model
- Maximum a posteriori estimation
  - Bayesian inference

most common outside of CS

Graphical models

- Naïve Bayes

Other ensemble models

I will post my old slides on the topic to the piazza page for reference, but they are not going to be on the final exam

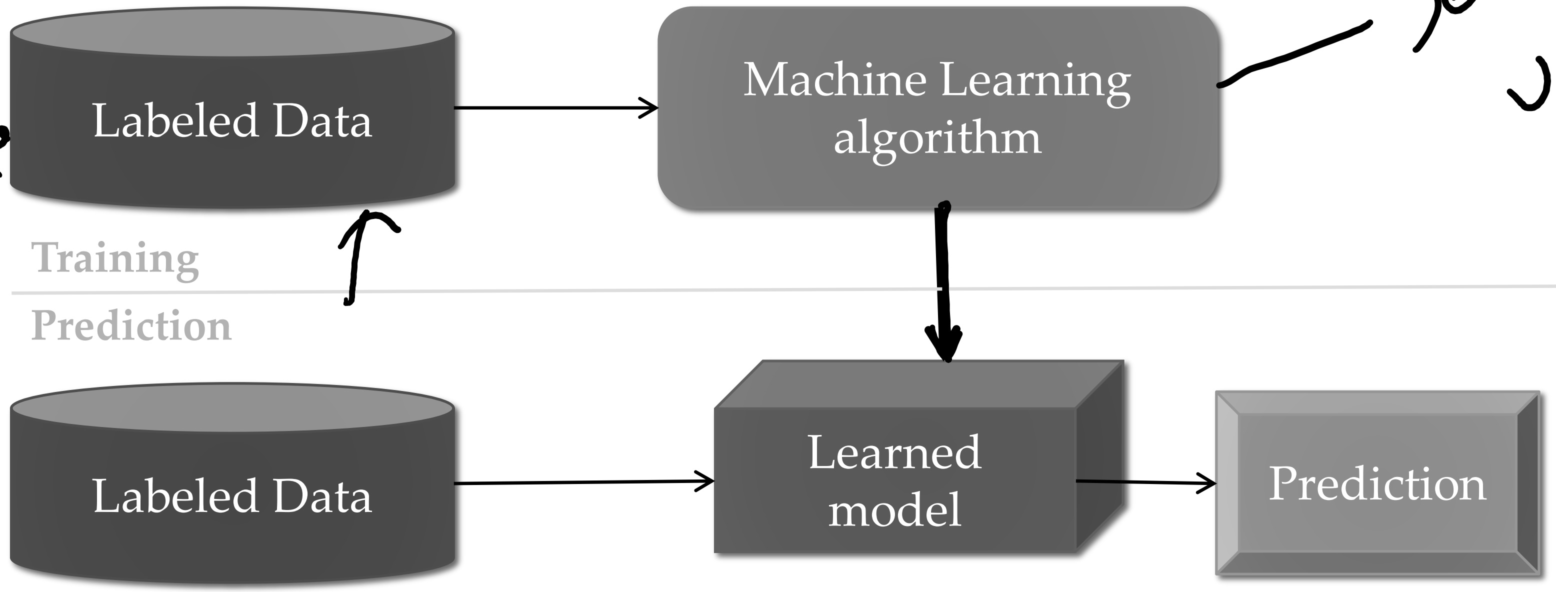
If the videos from a prev semester exist I will post them

Machine Learning Basics

Machine learning is a field of computer science that gives computers the ability to **learn without being explicitly programmed**

Select optimal via ECV

should be pulled from the same population



Methods that can learn from and make predictions on data

new data from world

Deep Learning Today

Advancement in speech recognition in the last ~~2~~ 3 years

- A few long-standing performance records were broken with deep learning methods
- Microsoft and Google have both deployed DL-based speech recognition systems in their products

Advancement in Computer Vision

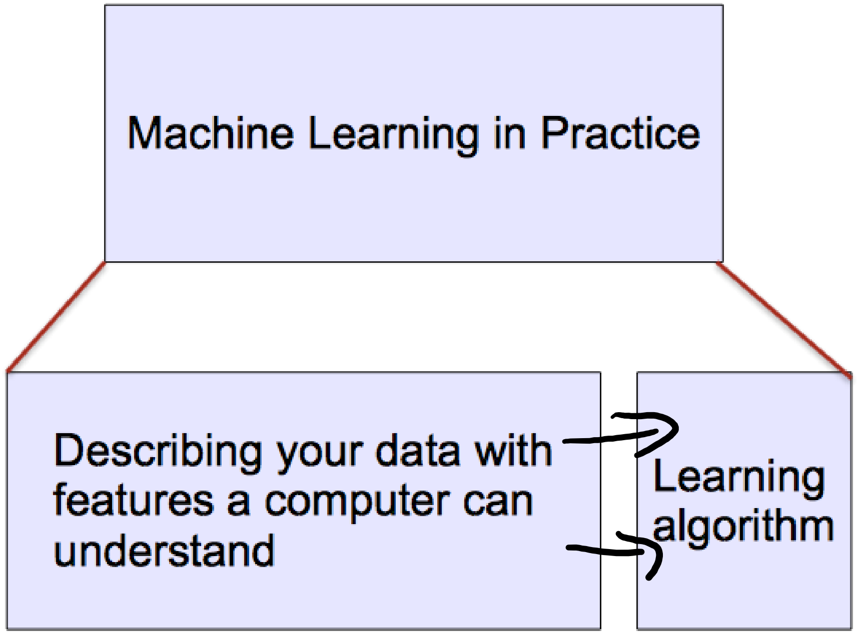
- Feature engineering is the bread-and-butter of a large portion of the CV community, which creates some resistance to feature learning
- But the record holders on ImageNet and Semantic Segmentation are convolutional nets

Advancement in Natural Language Processing

- Fine-grained sentiment analysis, syntactic parsing
- Language model, machine translation, question answering

ML vs. Deep Learning

Most machine learning methods work well because of **human-designed representations** and **input features**
ML becomes just **optimizing weights** to best make a final prediction

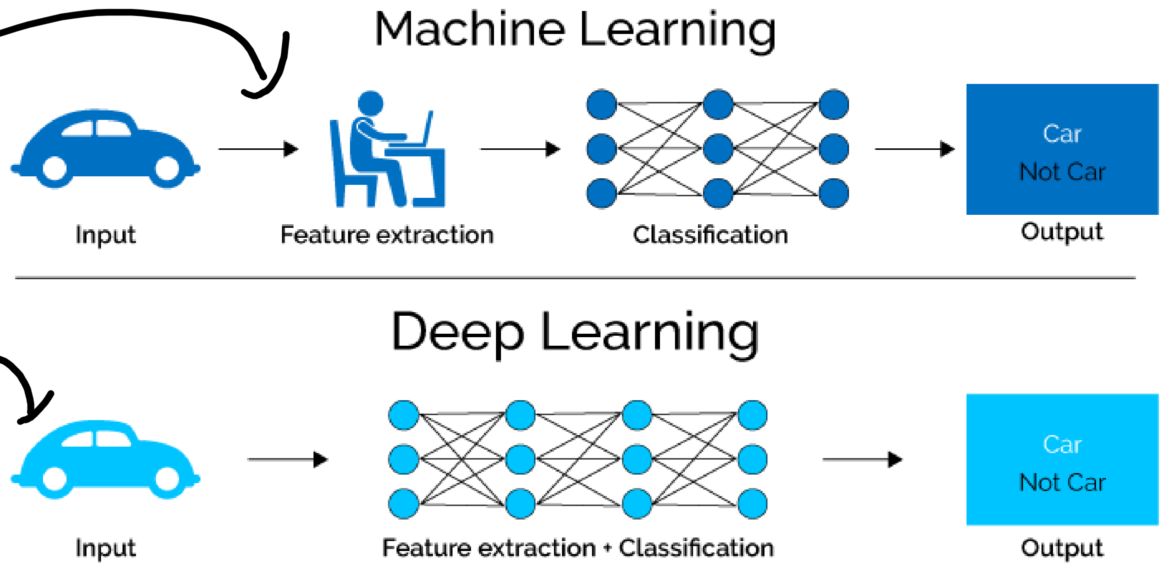


Feature	NER
Current Word	✓
Previous Word	✓
Next Word	✓
Current Word Character n-gram	all
Current POS Tag	✓
Surrounding POS Tag Sequence	✓
Current Word Shape	✓
Surrounding Word Shape Sequence	✓
Presence of Word in Left Window	size 4
Presence of Word in Right Window	size 4

What is Deep Learning (DL) ?

- A machine learning subfield of learning **representations** of data. Exceptional effective at **learning patterns**.
- Deep learning algorithms attempt to learn (multiple levels of) representation by using a **hierarchy of multiple layers**
- If you provide the system **tons of information**, it begins to understand it and respond in useful ways.

intrinsic variance



what we've done in this class

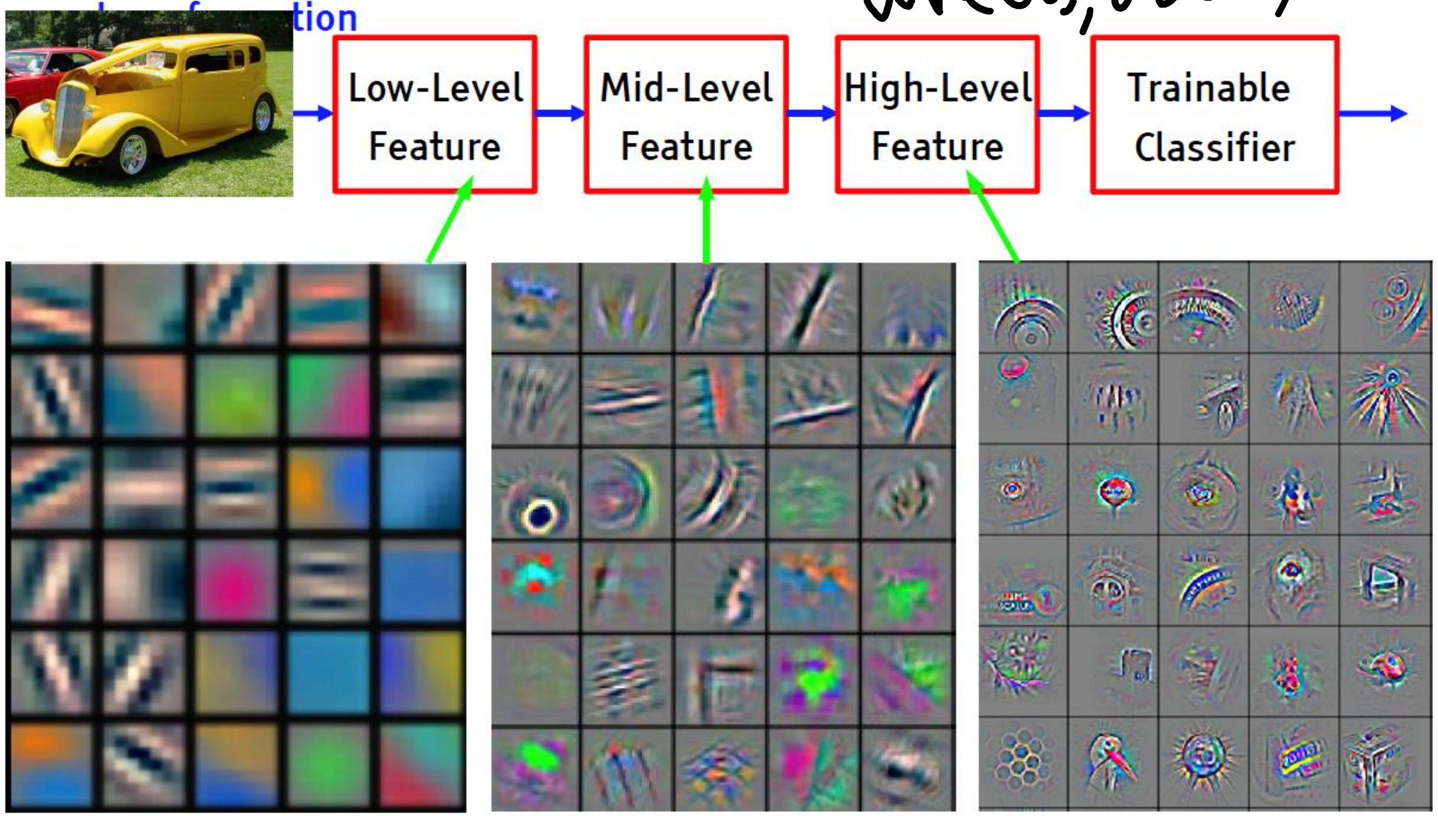
lots of features

features become more high level

Deep Learning = Learning Hierarchical Representations

wheels, doors, windows

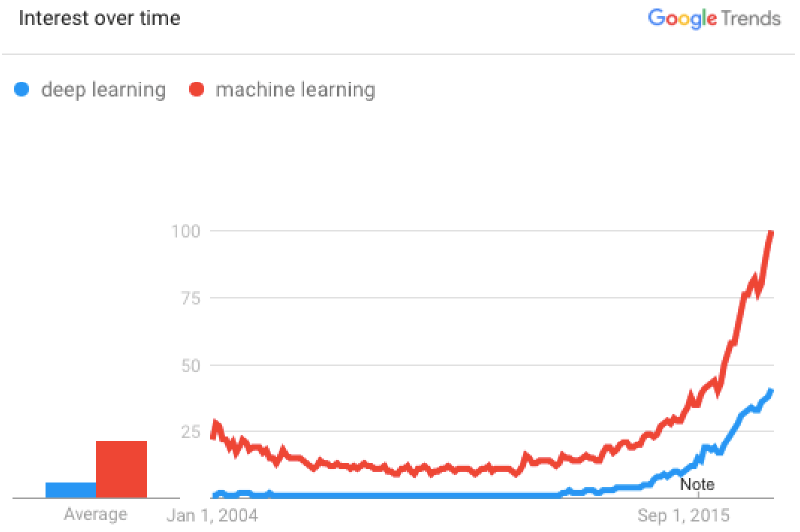
edges + shapes



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Why is DL useful?

- Manually designed features are often **over-specified**, **incomplete** and take a **long time to design** and validate
- Learned Features are **easy to adapt**, **fast** to learn
- Deep learning provides a very **flexible**, (almost?) **universal**, learnable framework for representing world, visual and linguistic information.
- Can learn both unsupervised and supervised
- Effective **end-to-end** joint system learning
- Utilize large amounts of training data



In ~2010 DL started outperforming other ML techniques
first in speech and vision, then NLP

informed by an...

What exactly is deep learning?

- ‘Deep Learning’ means using a neural network
  with several layers of nodes between input and output

- The series of layers between input & output do
  feature identification and processing in a series of stages,
  just as our brains seem to.

- Okay, we’ve done neural networks before, what’s actually new?

we're addn more layers

- makes it harder to fit

NN or gradient descent

on a multi-dimensional

error surface

Limitations of Neural Networks

Random initialization + **densely connected networks** lead to:

each layer is completely connected to the next

High cost
- Each neuron in the neural network can be considered as a logistic regression.
- Training the entire neural network is to train all the interconnected logistic regressions.

Difficult to train as the number of hidden layers increases ← lower quality minima
- Recall that logistic regression is trained by gradient descent.
- In backpropagation, gradient is progressively getting more dilute. That is, below top layers, the correction signal δ_n is minimal. ← diminishing gradient

Stuck in local optima
- The objective function of the neural network is usually not convex.
- The random initialization does not guarantee starting from the proximity of global optima.

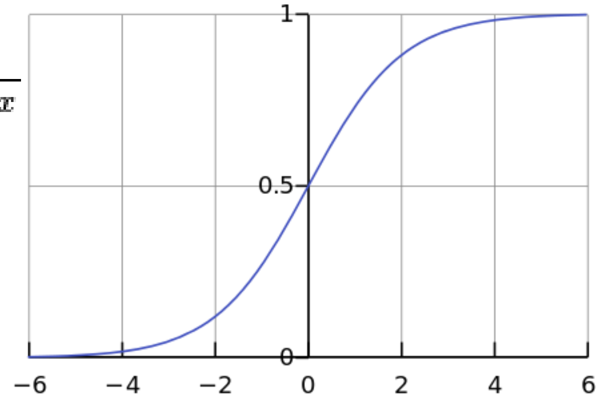
Solution: — we would need to run it more times
- Deep Learning/Learning multiple levels of representation

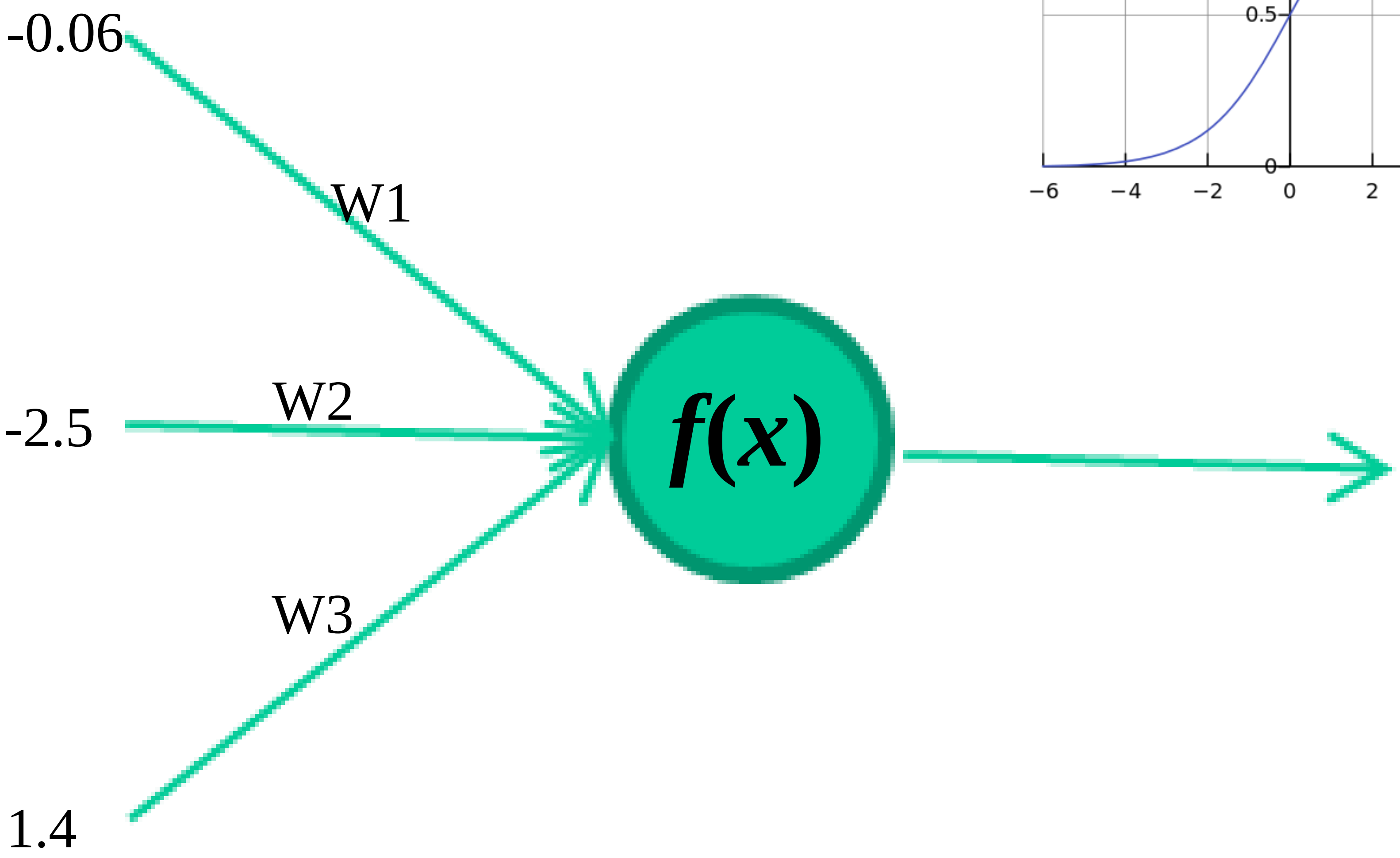
regular neural network only "trains" on error

longer answers

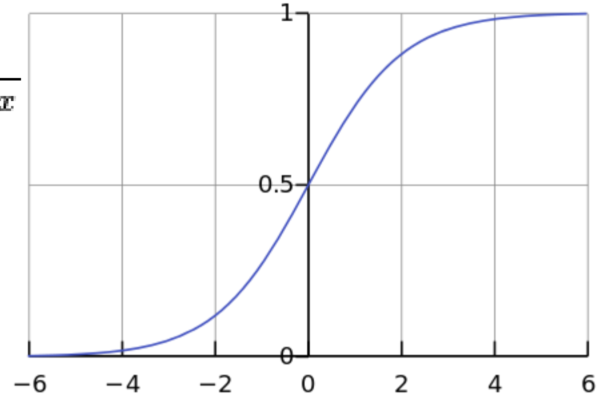
1. reminder/quick-explanation of how neural network weights are learned;
2. the idea of **unsupervised feature learning** (why 'intermediate features' are important for difficult classification tasks, and how NNs seem to naturally learn them)
3. The 'breakthrough' – the simple trick for training Deep neural networks

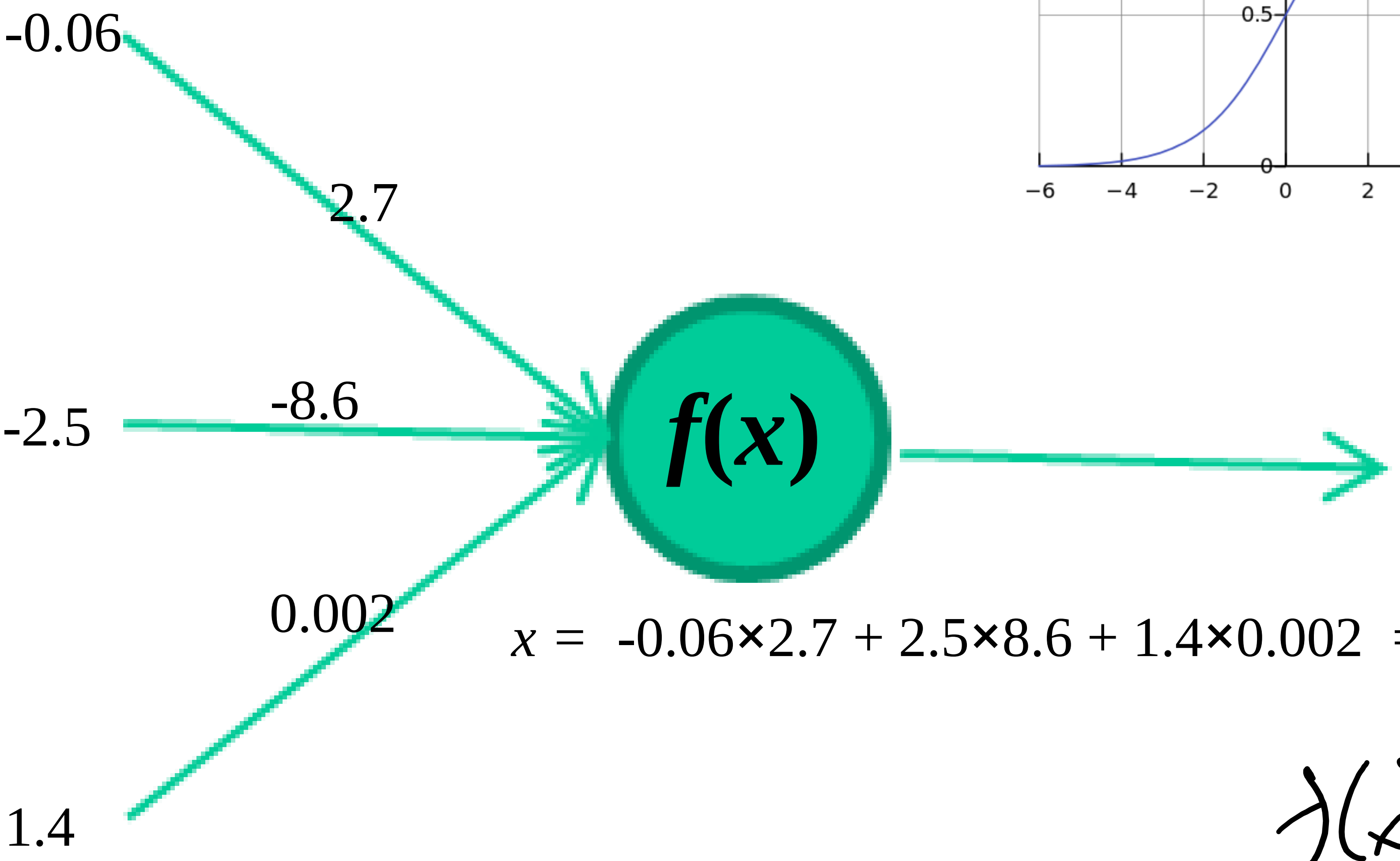
$$f(x) = \frac{1}{1 + e^{-x}}$$





$$f(x) = \frac{1}{1+\mathrm{e}^{-x}}$$



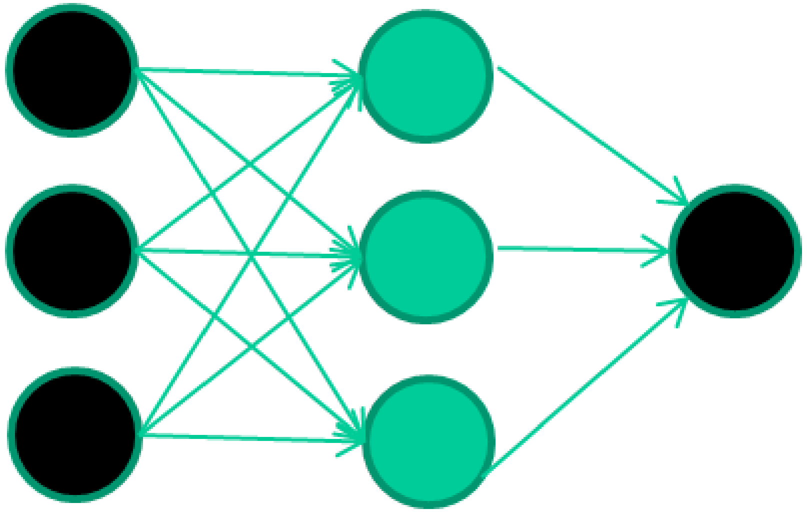


x = -0.06×2.7 + 2.5×8.6 + 1.4×0.002 = 21.34

$f(x) =$ sigmoid(21)

A dataset

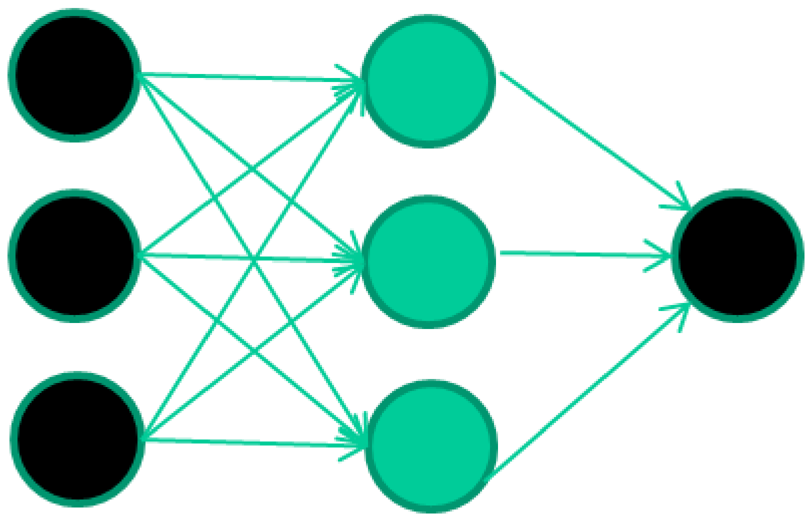
Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			



Training the neural network

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0

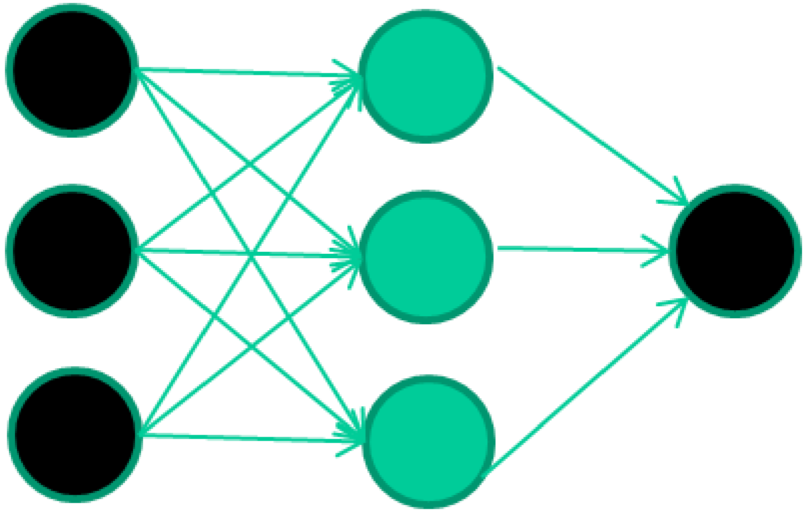
etc …



Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			

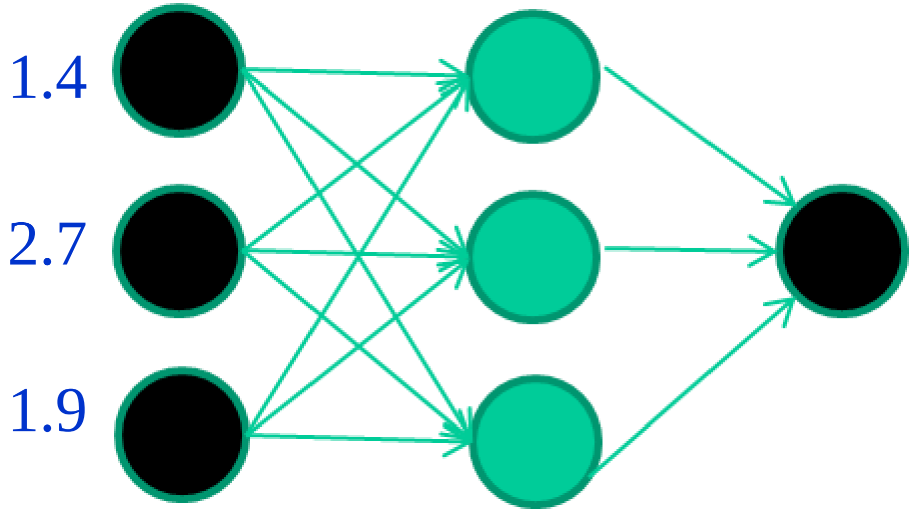
Initialise with random weights



Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			

Present a training pattern

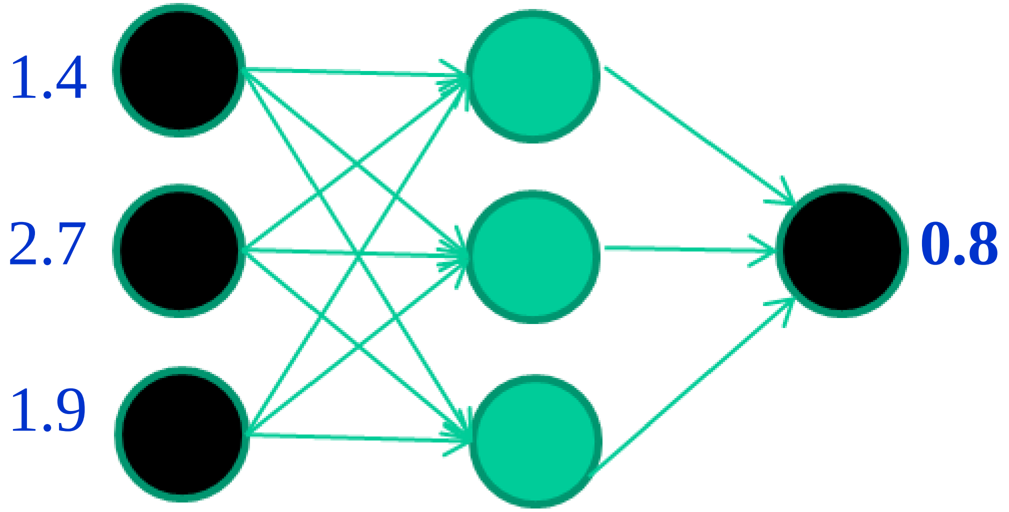


Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0

etc …

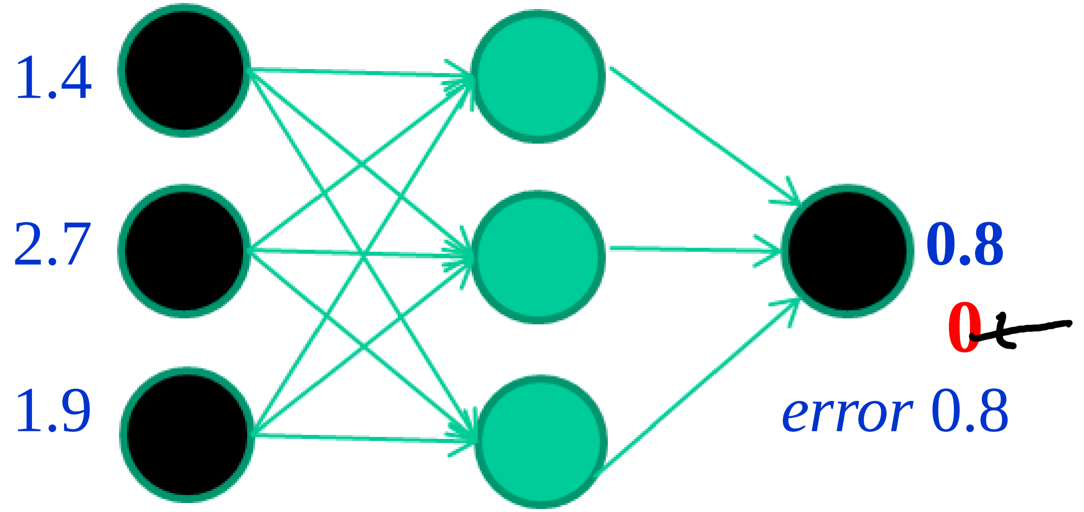
Feed it through to get output



Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			

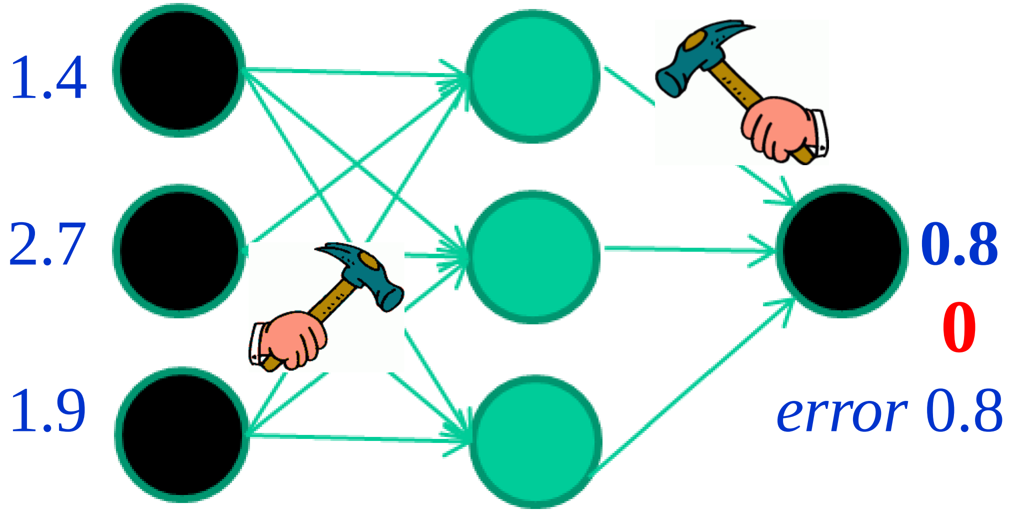
Compare with target output



Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			

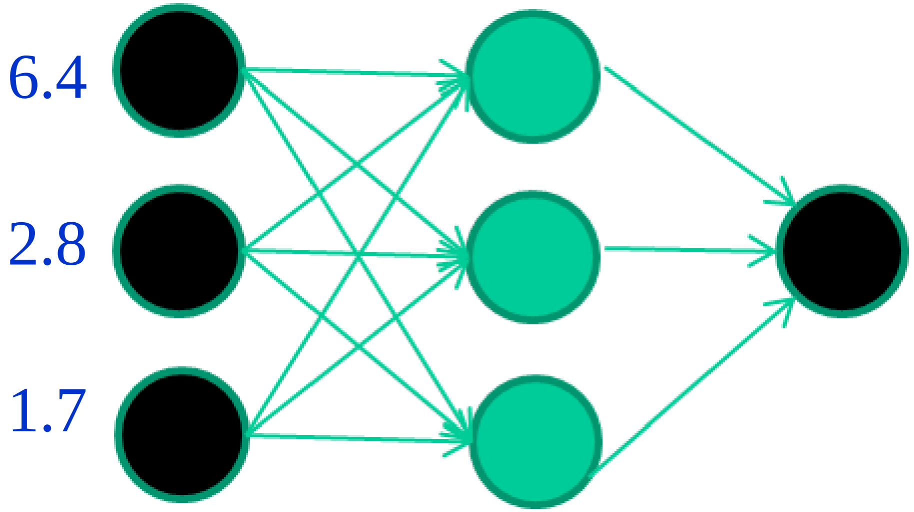
Adjust weights based on error



Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			

Present a training pattern

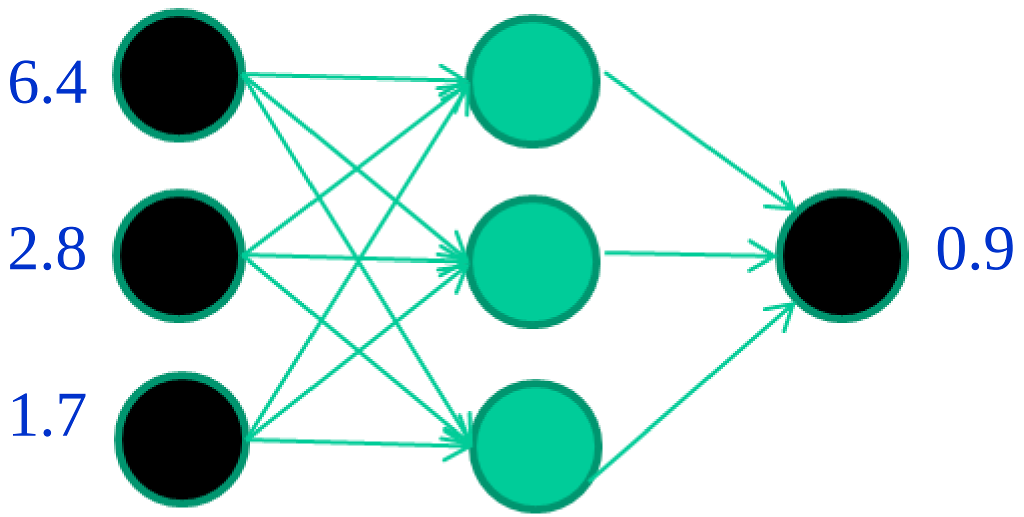


Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0

etc …

Feed it through to get output

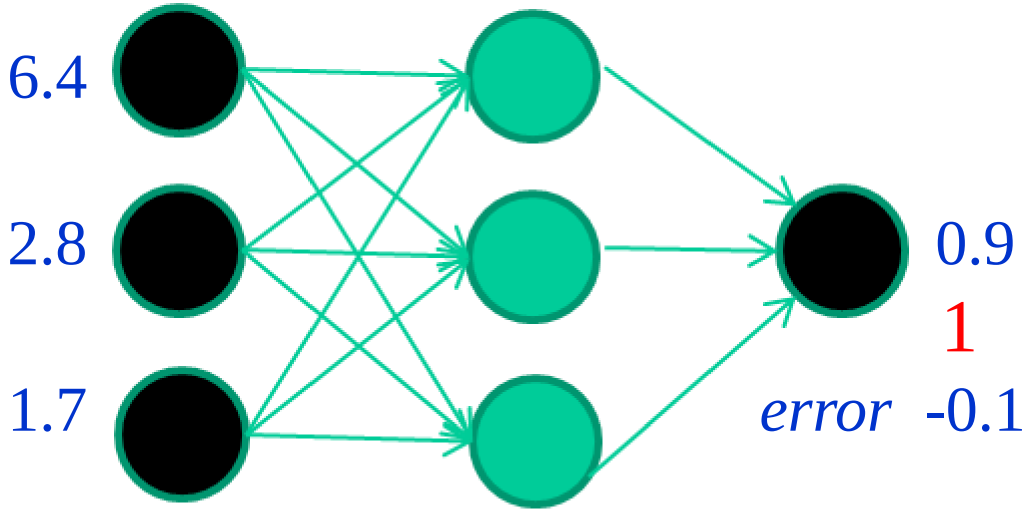


Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0

etc …

Compare with target output

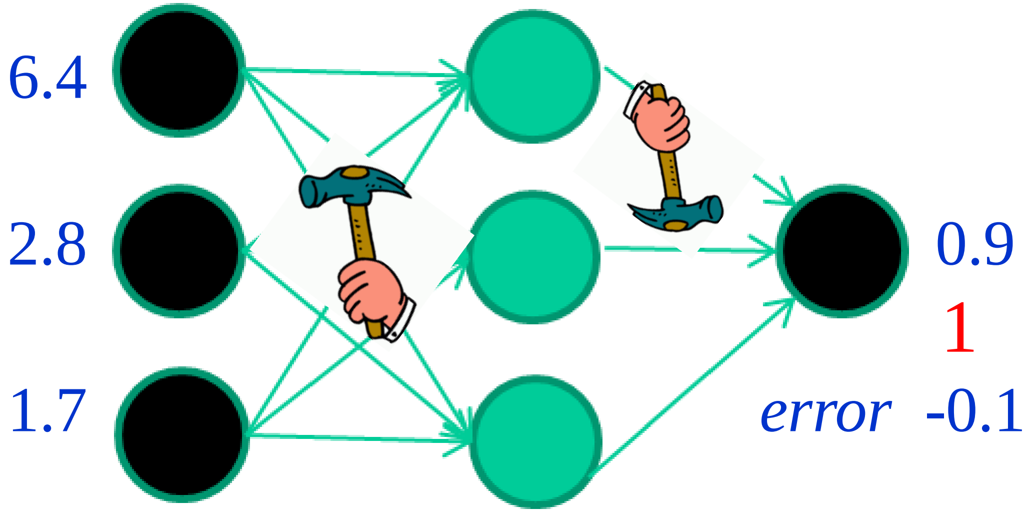


feed forward
back prop NN

Adjust weights based on error

Training data

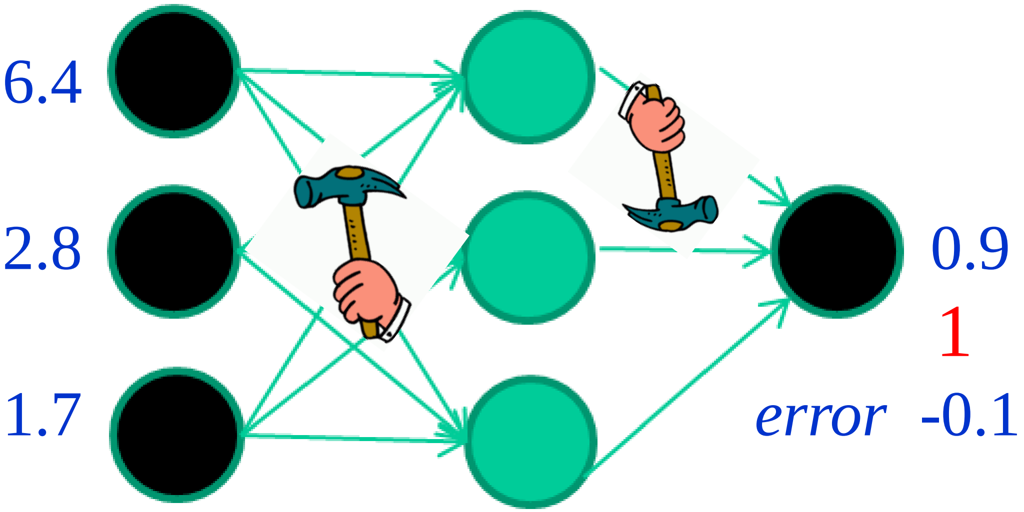
Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			



Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			

And so on ….



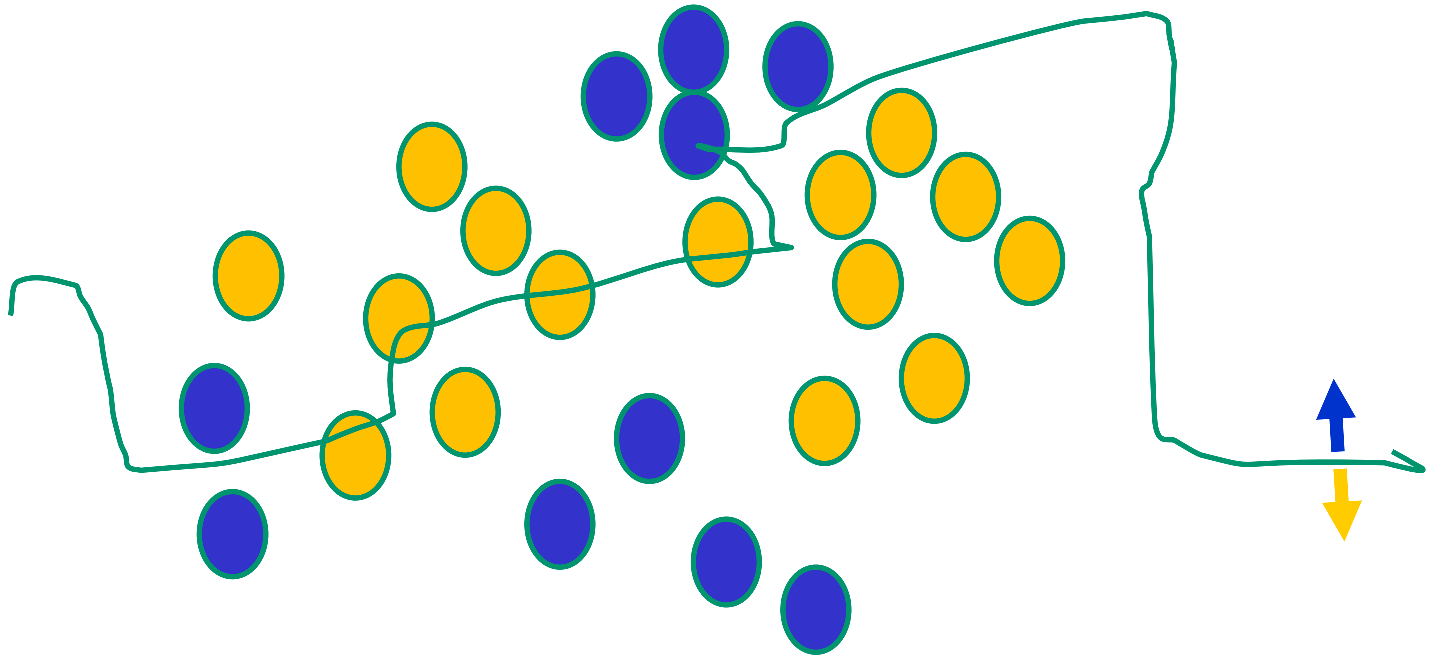
Repeat this thousands, maybe millions of times – each time taking a random training instance, and making slight weight adjustments

Algorithms for weight adjustment are designed to make changes that will reduce the error

sgd

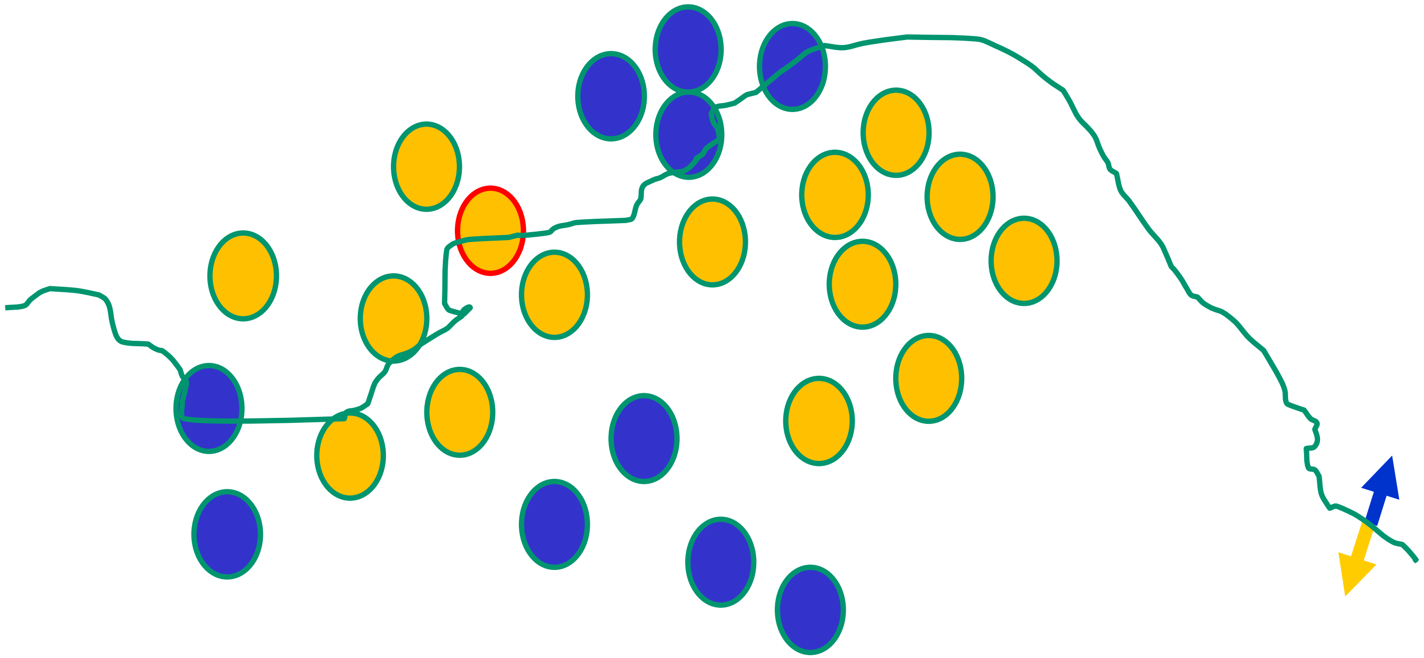
The decision boundary perspective…

Initial random weights



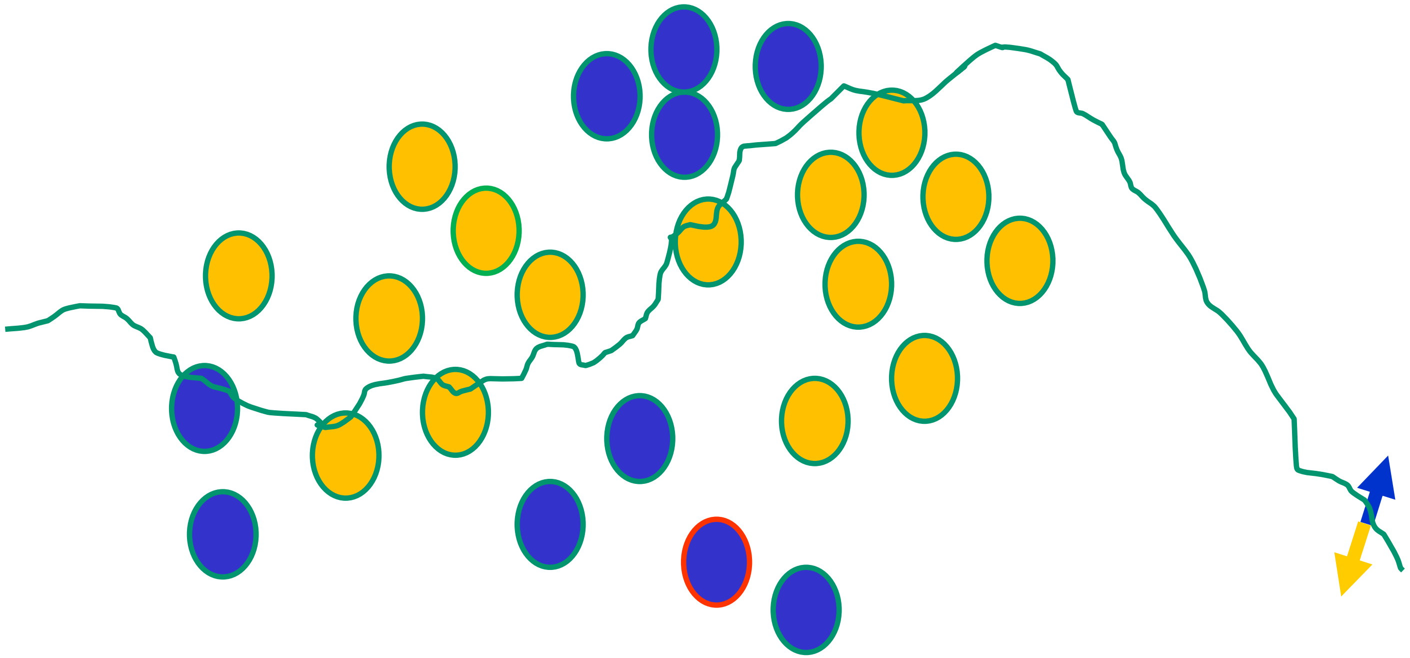
The decision boundary perspective…

Present a training instance / adjust the weights



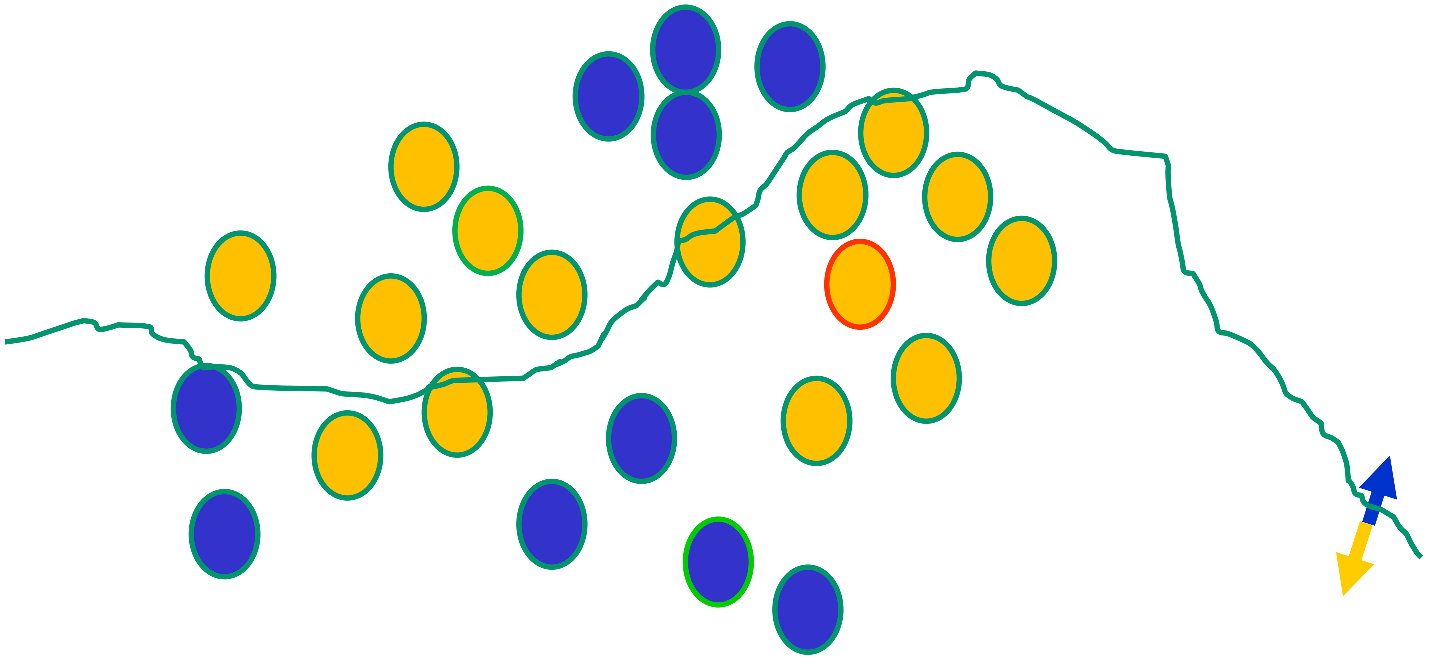
The decision boundary perspective…

Present a training instance / adjust the weights



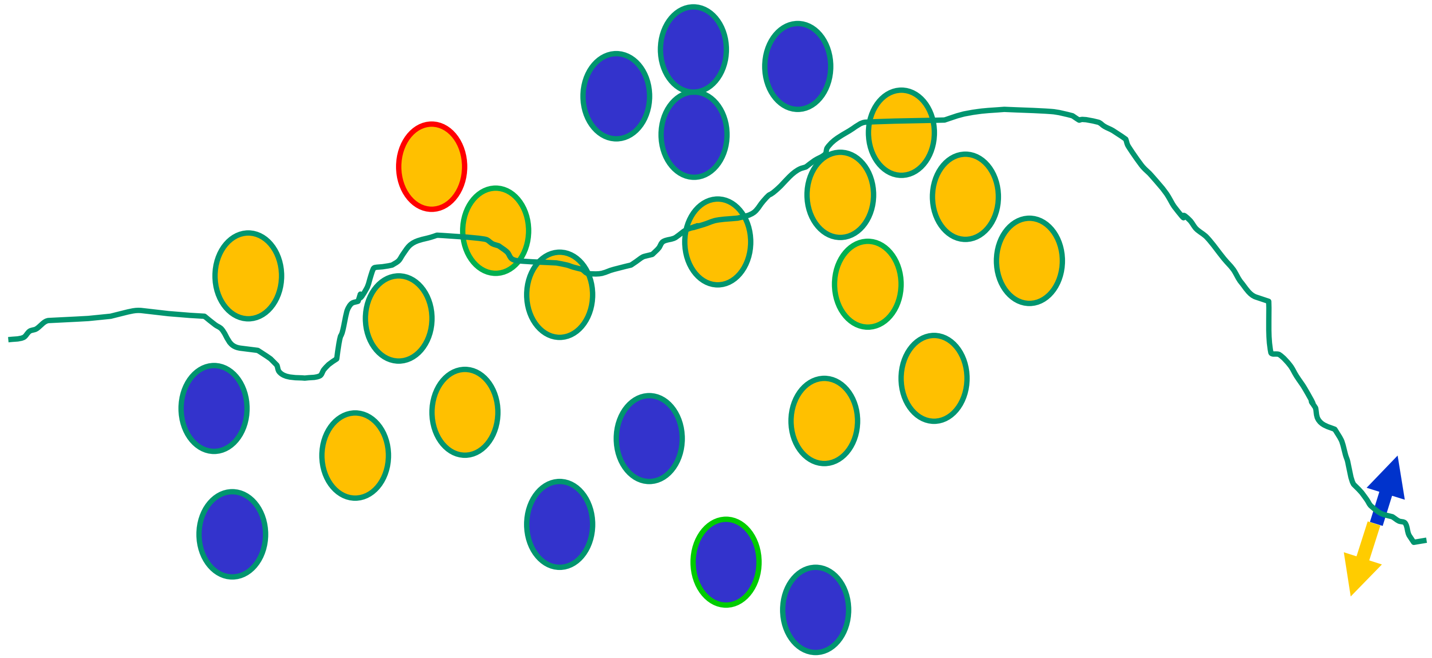
The decision boundary perspective…

Present a training instance / adjust the weights



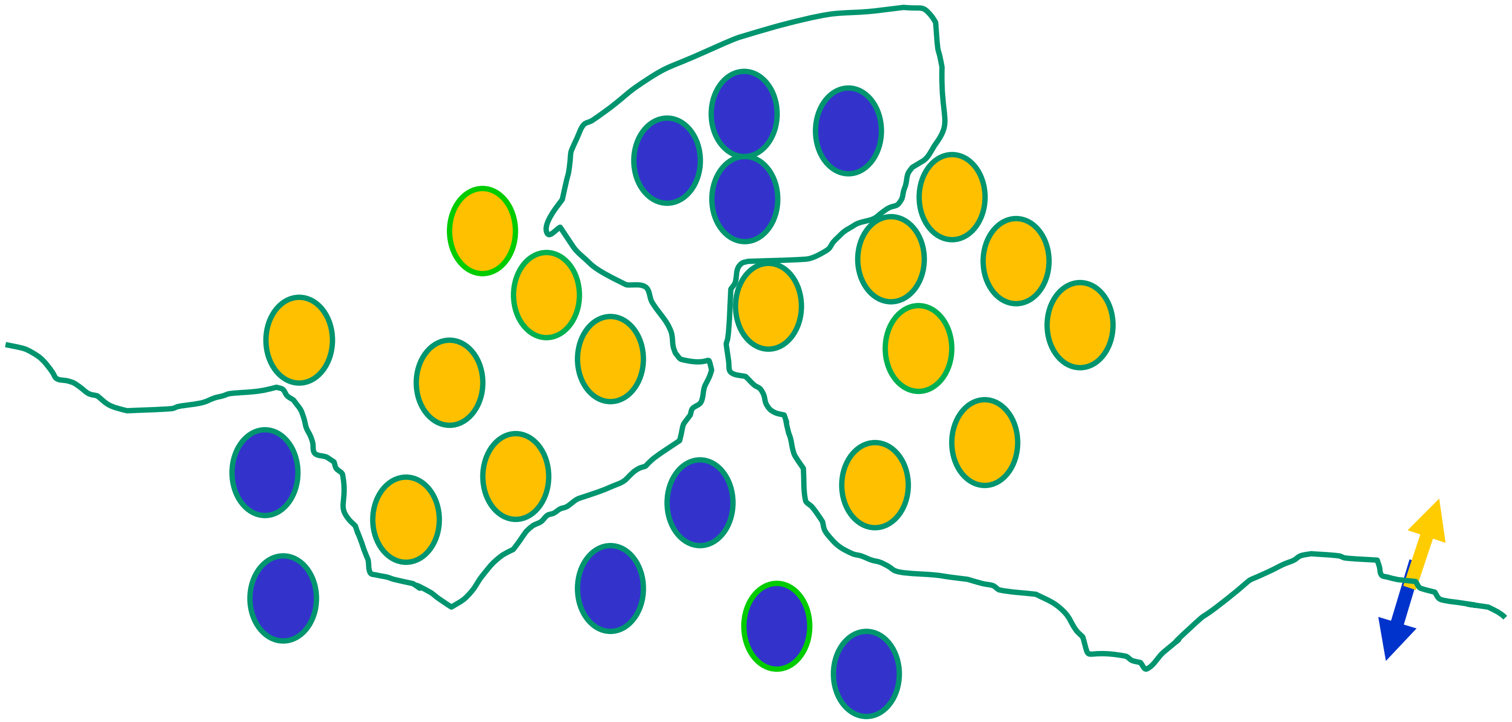
The decision boundary perspective…

Present a training instance / adjust the weights



The decision boundary perspective…

Eventually ….

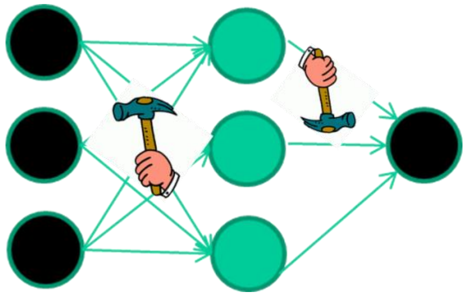


The point I am trying to make

- weight-learning algorithms for NNs are dumb

- they work by making thousands and thousands of tiny adjustments, each making the network do better at the most recent pattern, but perhaps a little worse on many others

- but, by dumb luck, eventually this tends to be good enough to learn effective classifiers for many real applications



Feature detectors

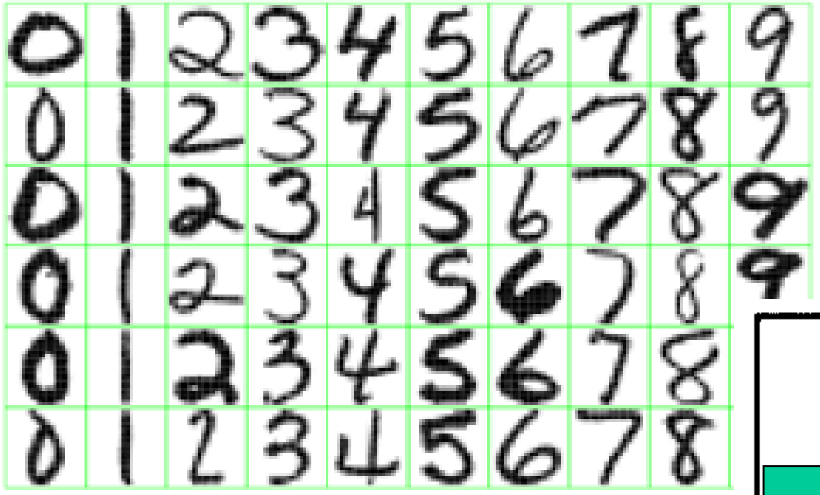
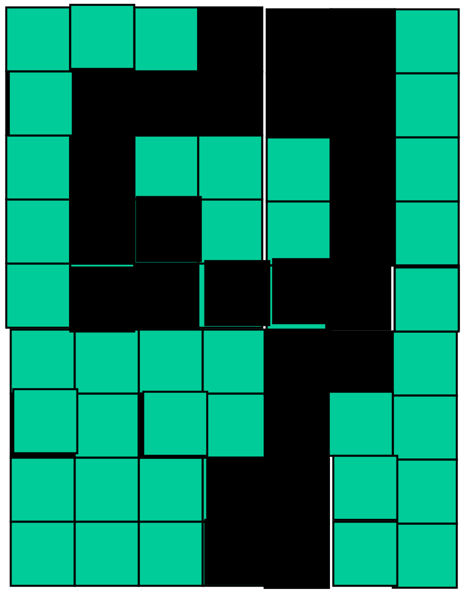
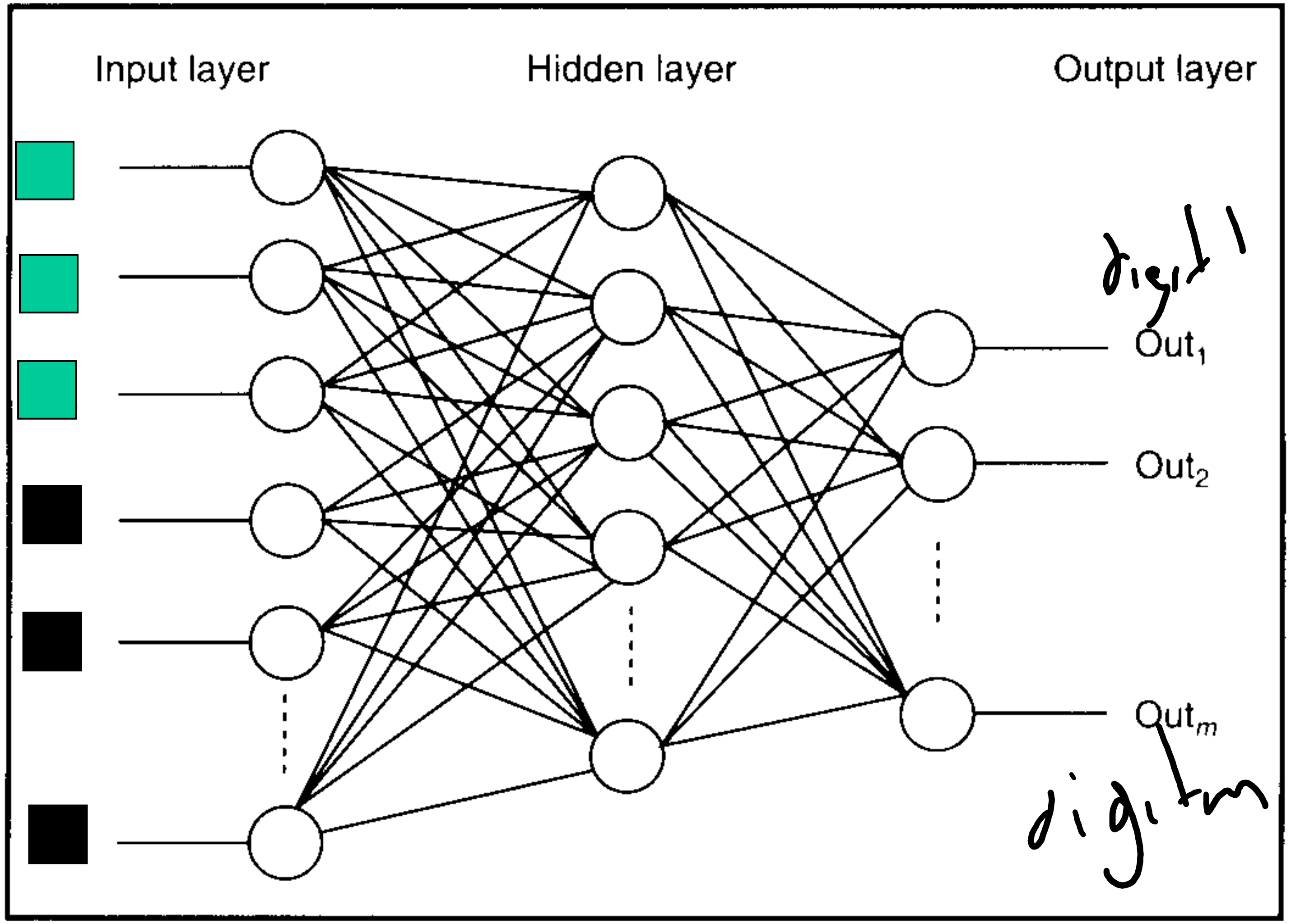


Figure 1.2: *Examples of handwritten digits from postal envelopes.*

binary





digit 1

digit m

what is this unit doing?

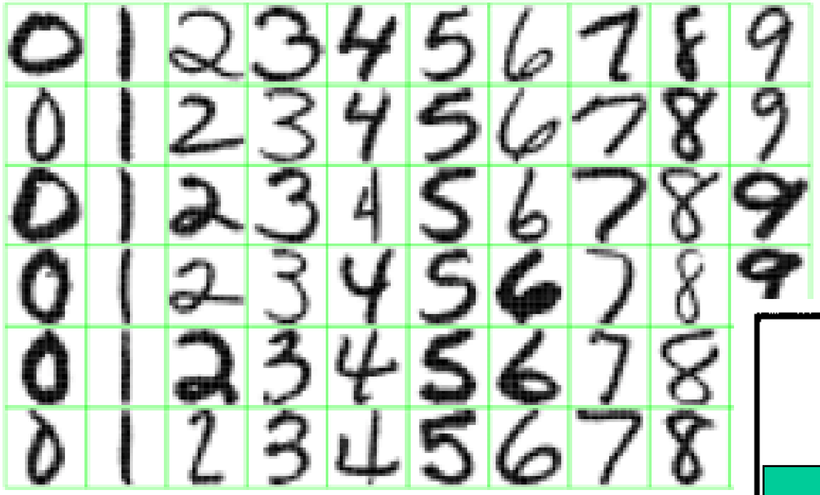
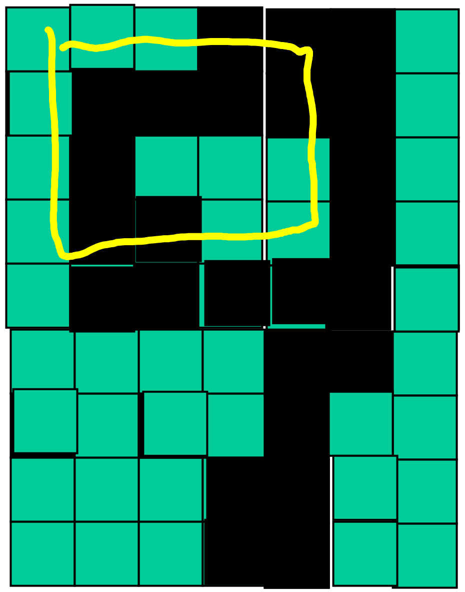
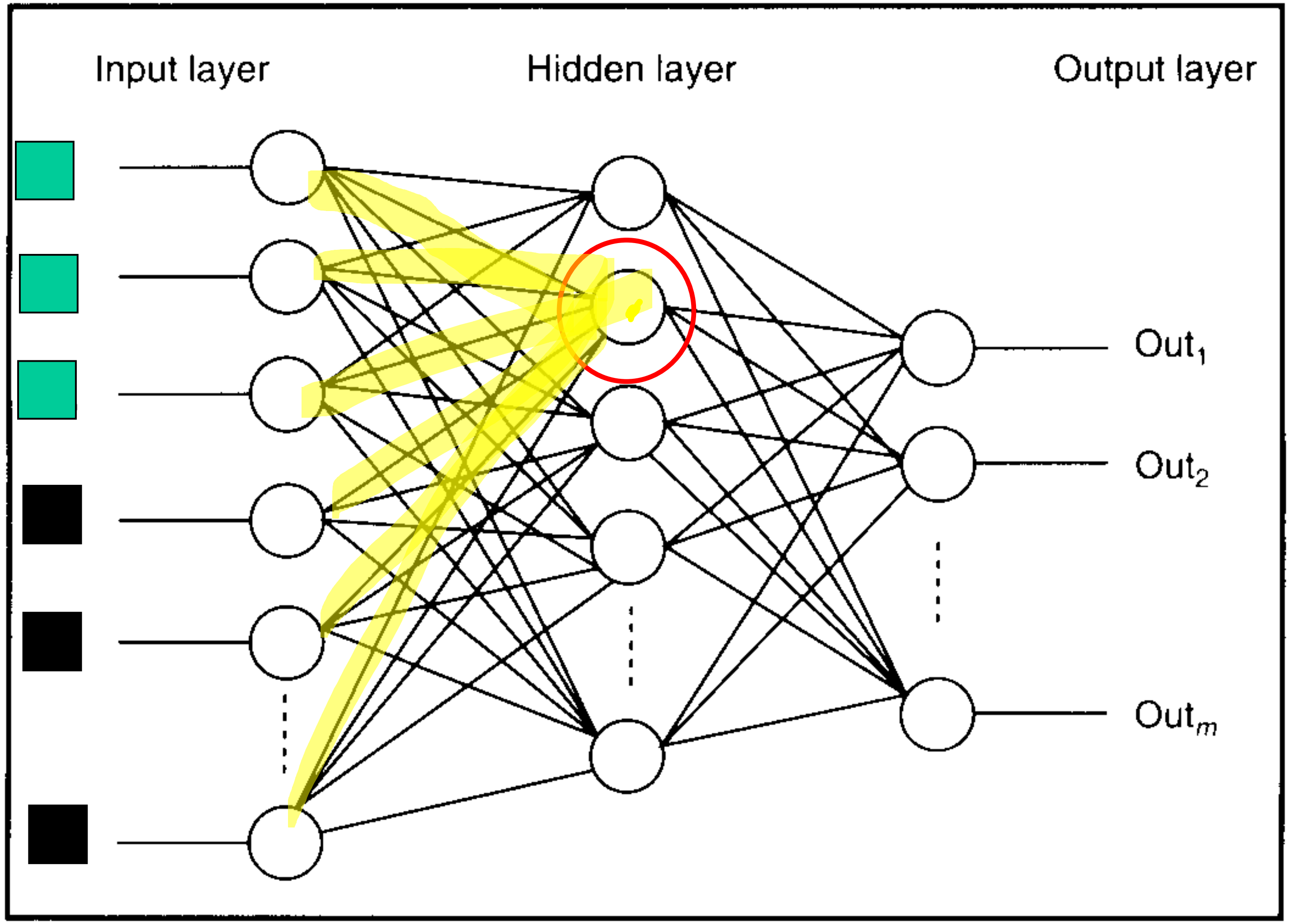
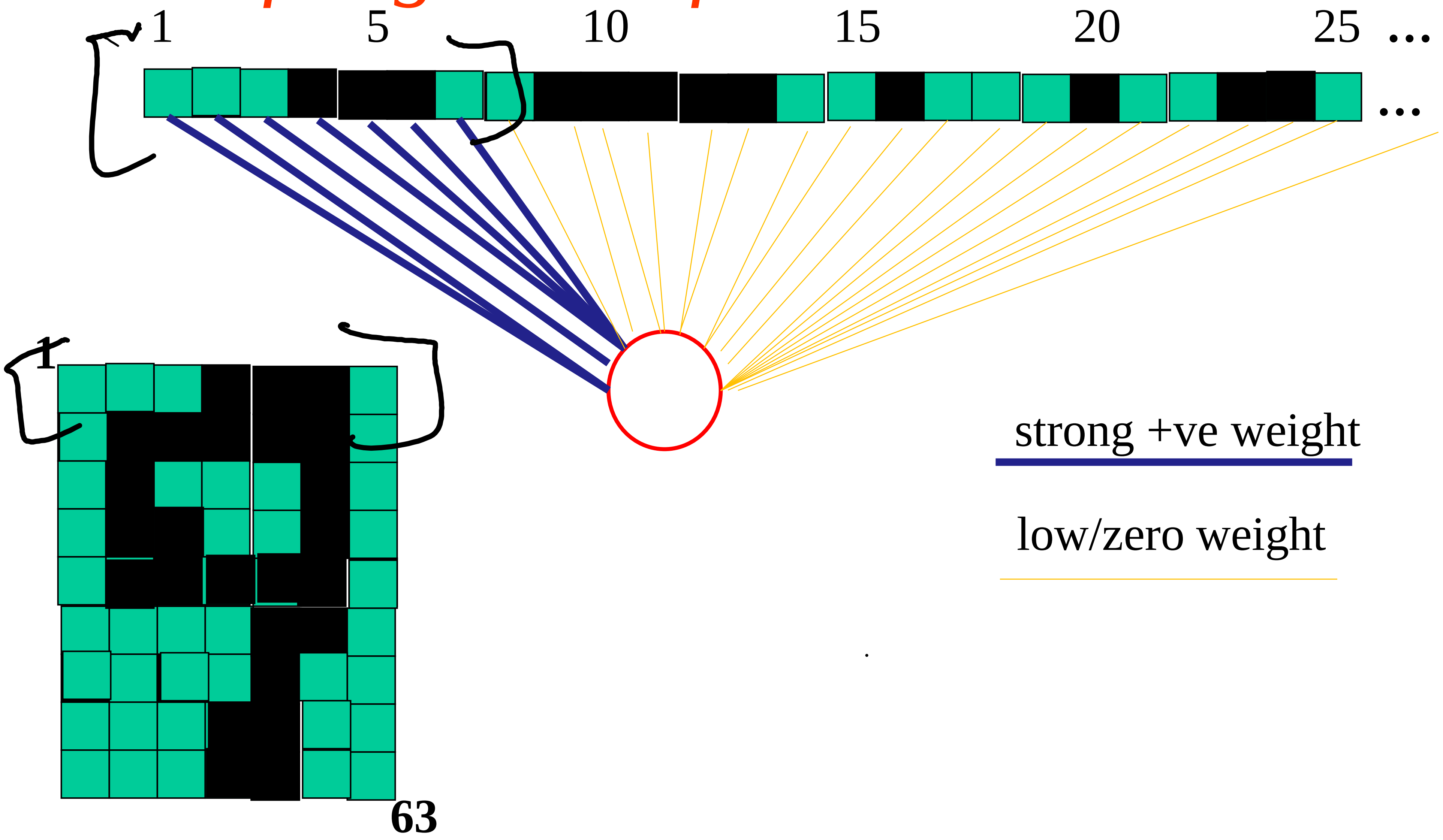


Figure 1.2: *Examples of handwritten digits from postal envelopes.*

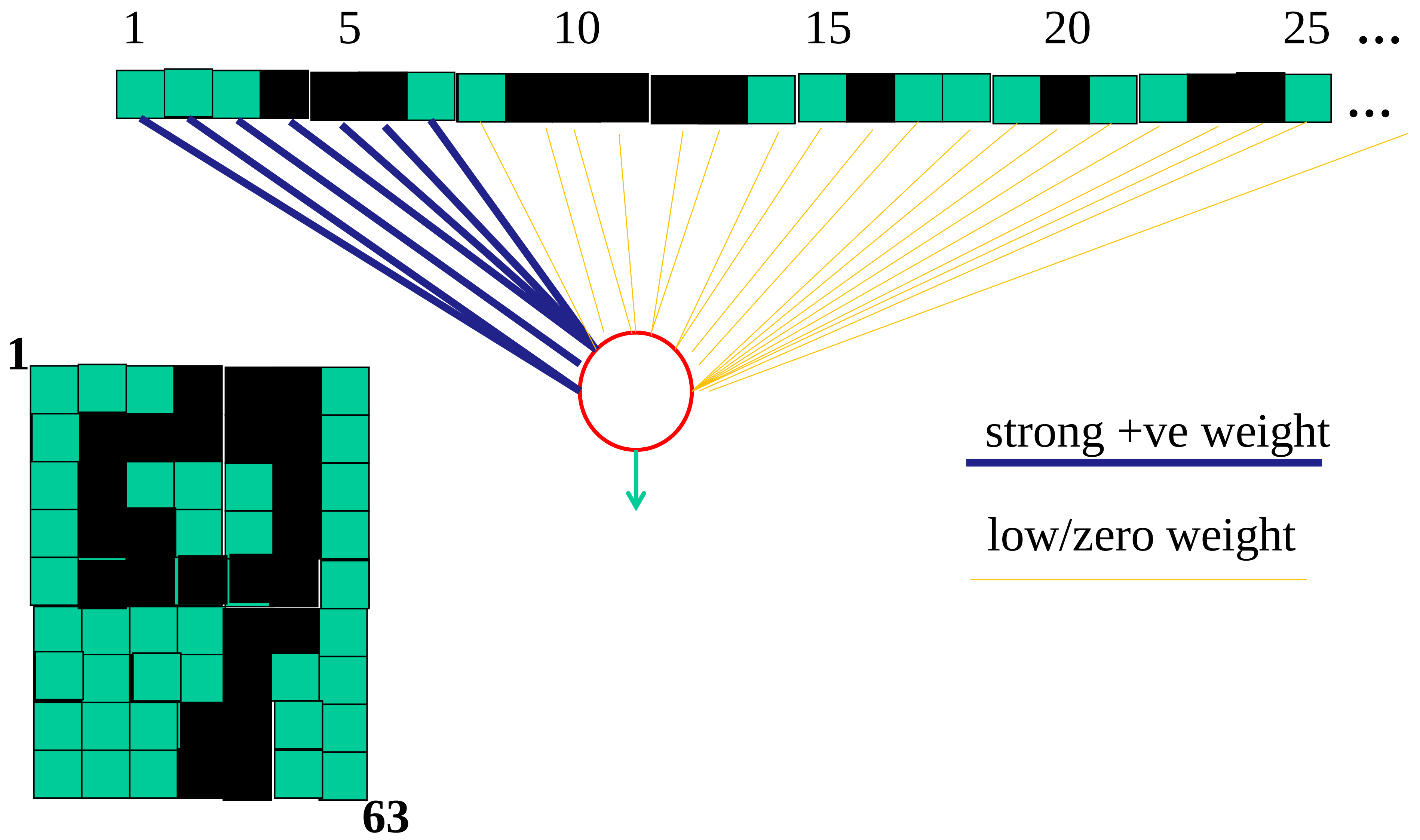




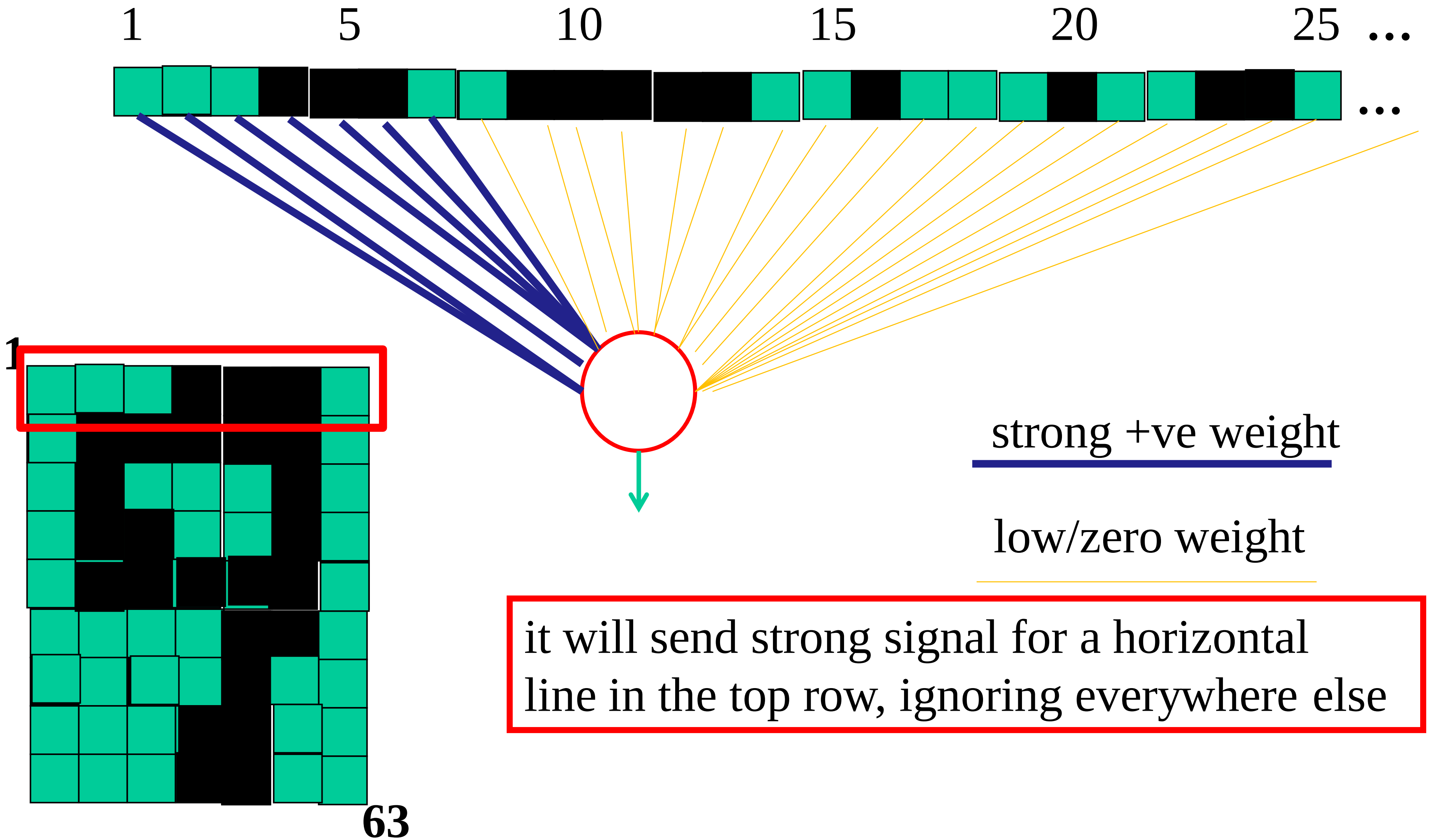
Hidden layer units become *self-organised feature detectors*



What does this unit detect?

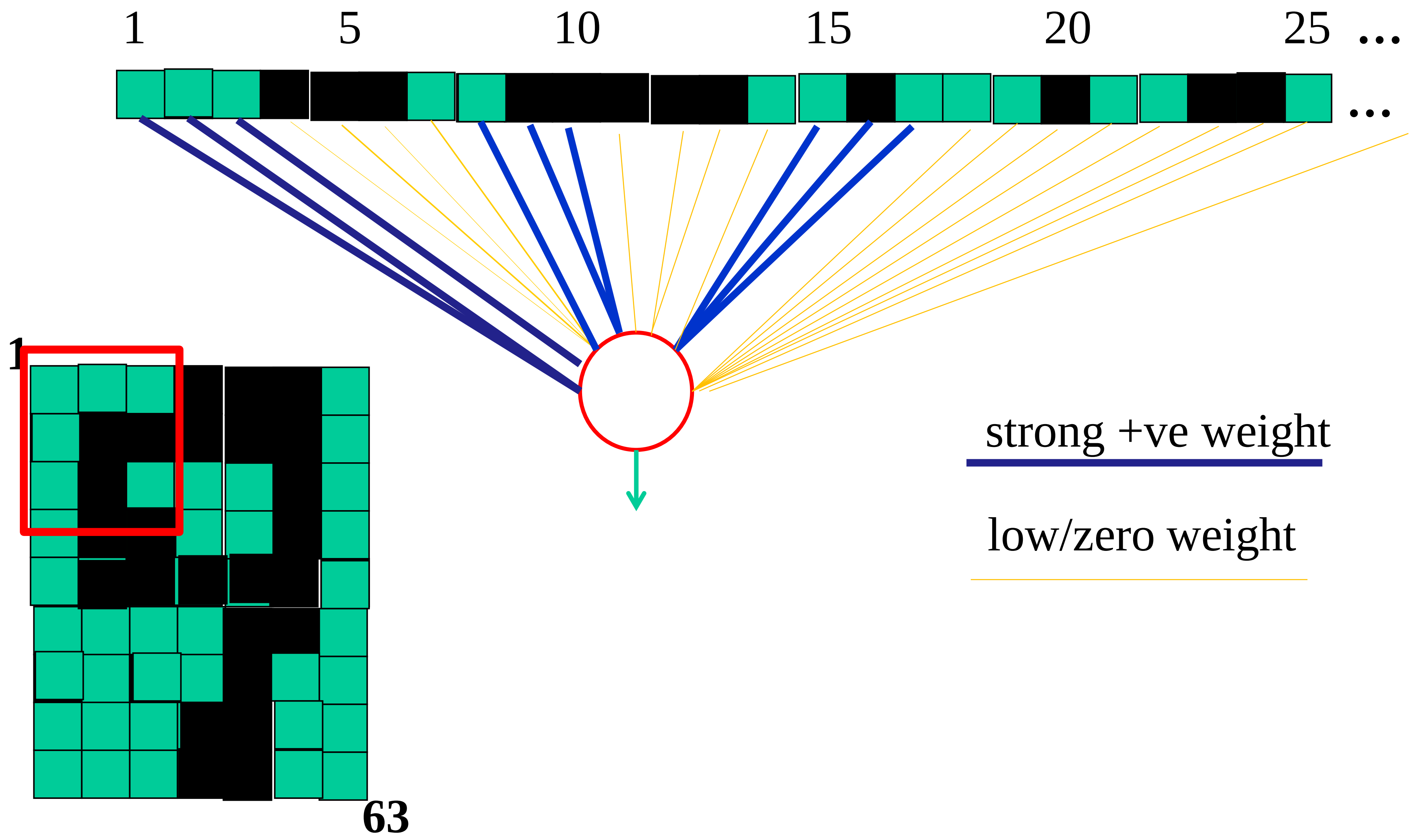


What does this unit detect?

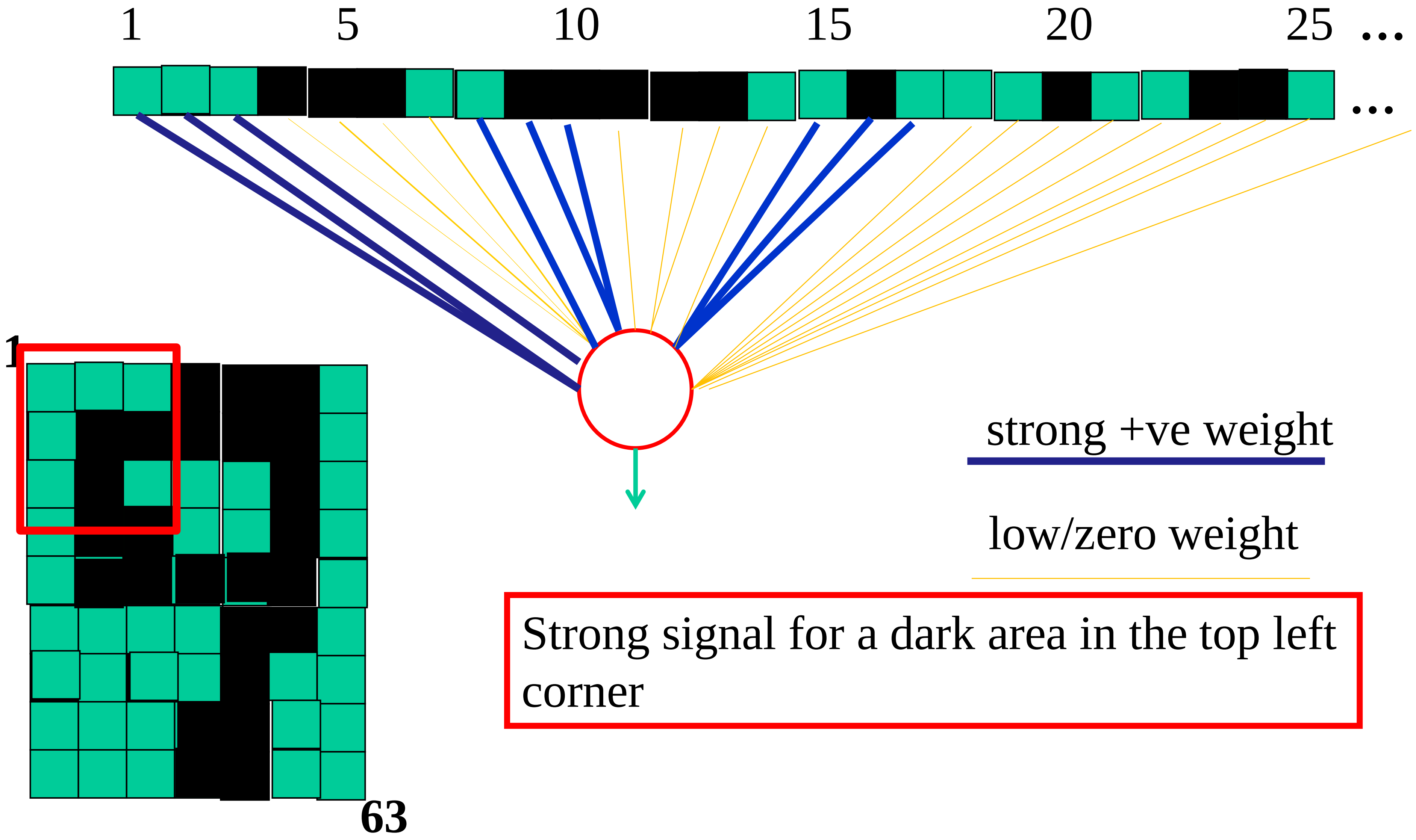


it will send strong signal for a horizontal line in the top row, ignoring everywhere else

What does this unit detect?



What does this unit detect?



Strong signal for a dark area in the top left corner

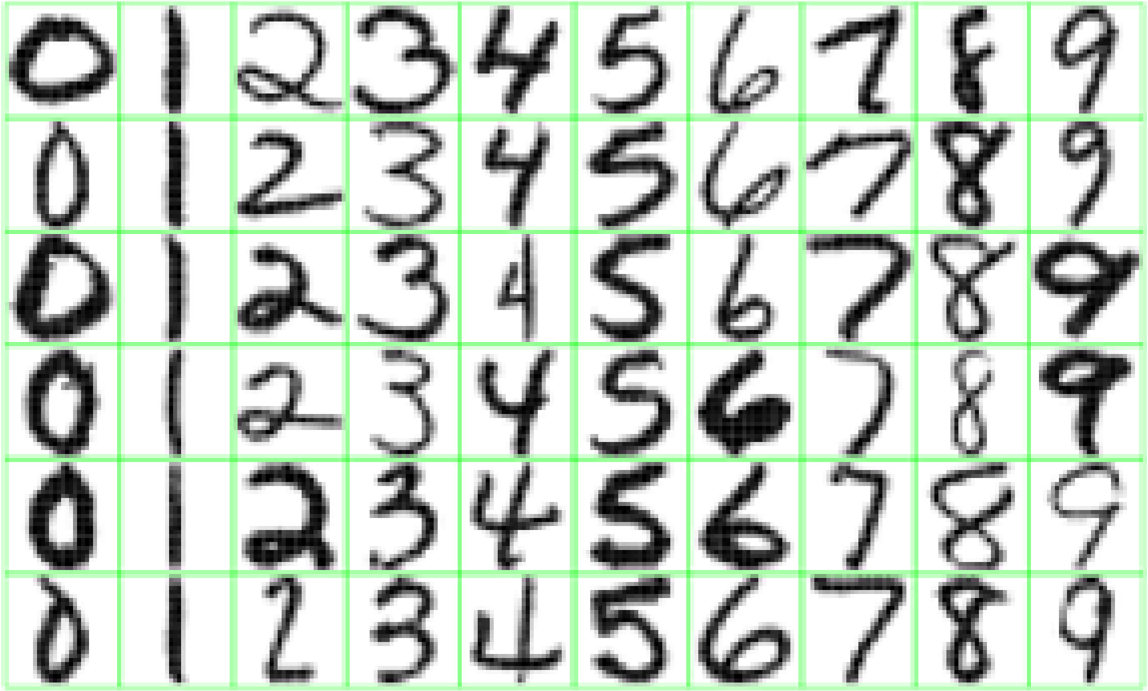


Figure 1.2: *Examples of handwritten digits from U.S. postal envelopes.*

What are the patterns?

What features might you expect a good NN to learn, when trained with data like this?

1

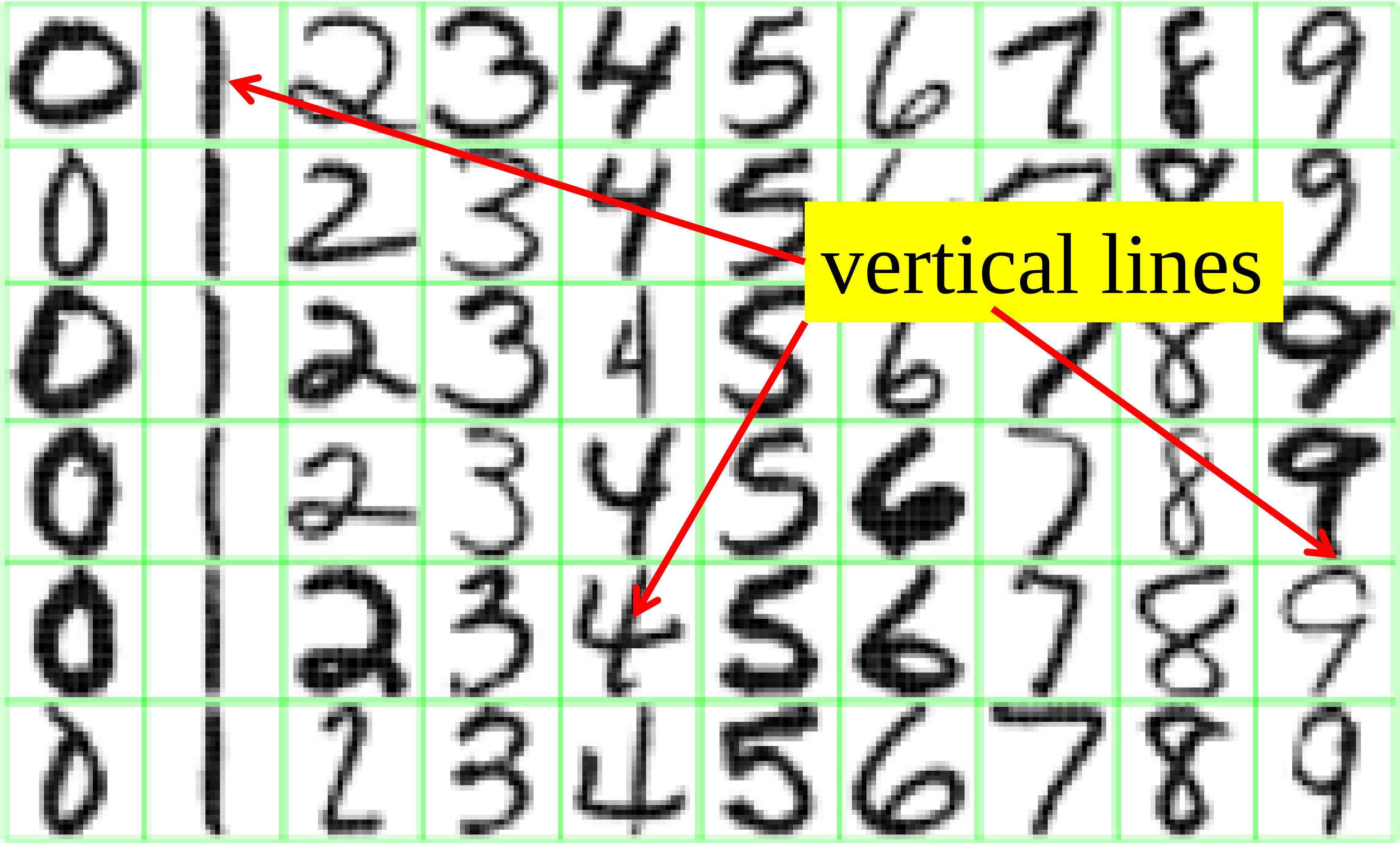


Figure 1.2: *Examples of handwritten digits from U.S. postal envelopes.*

1

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

1

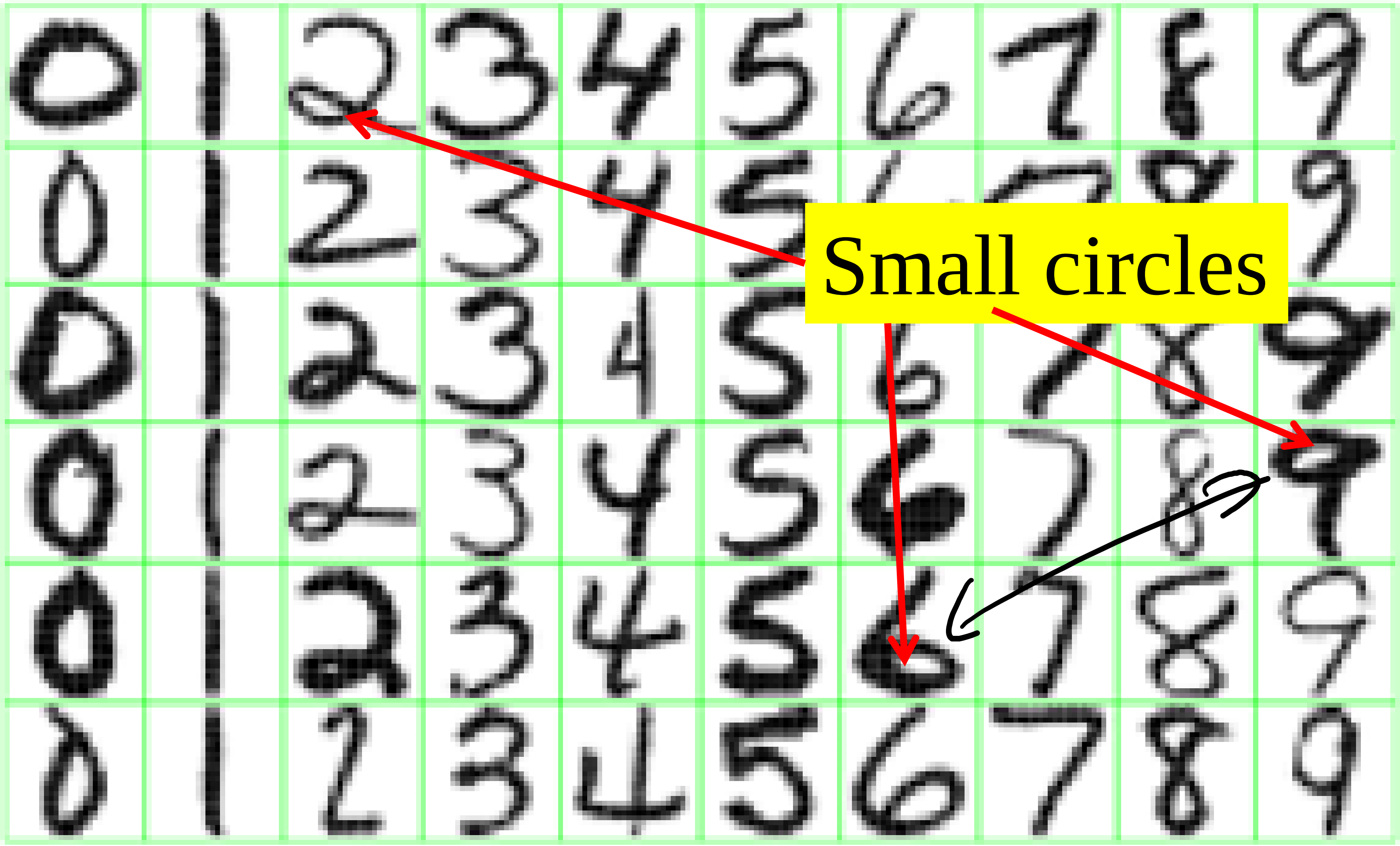
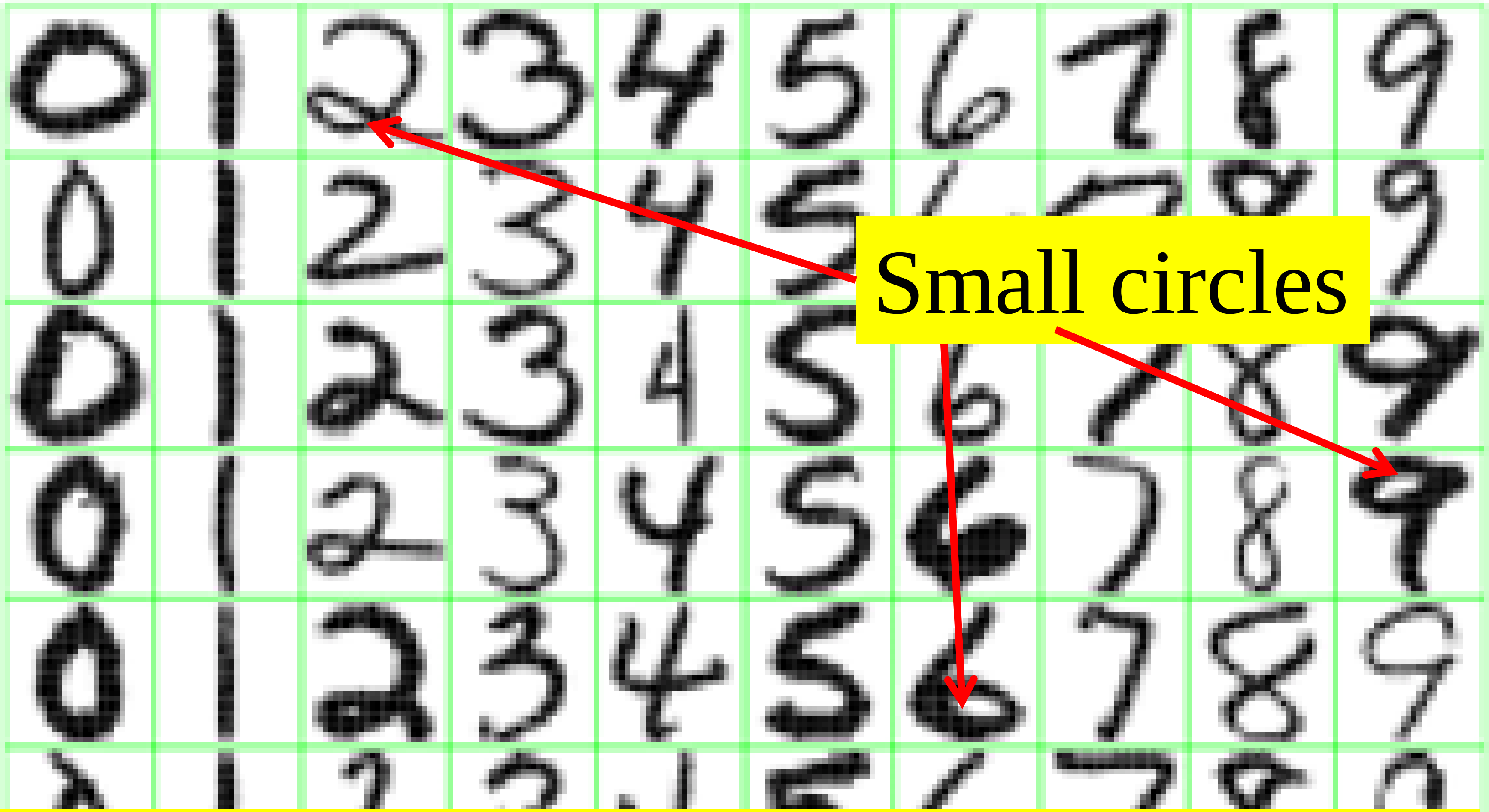


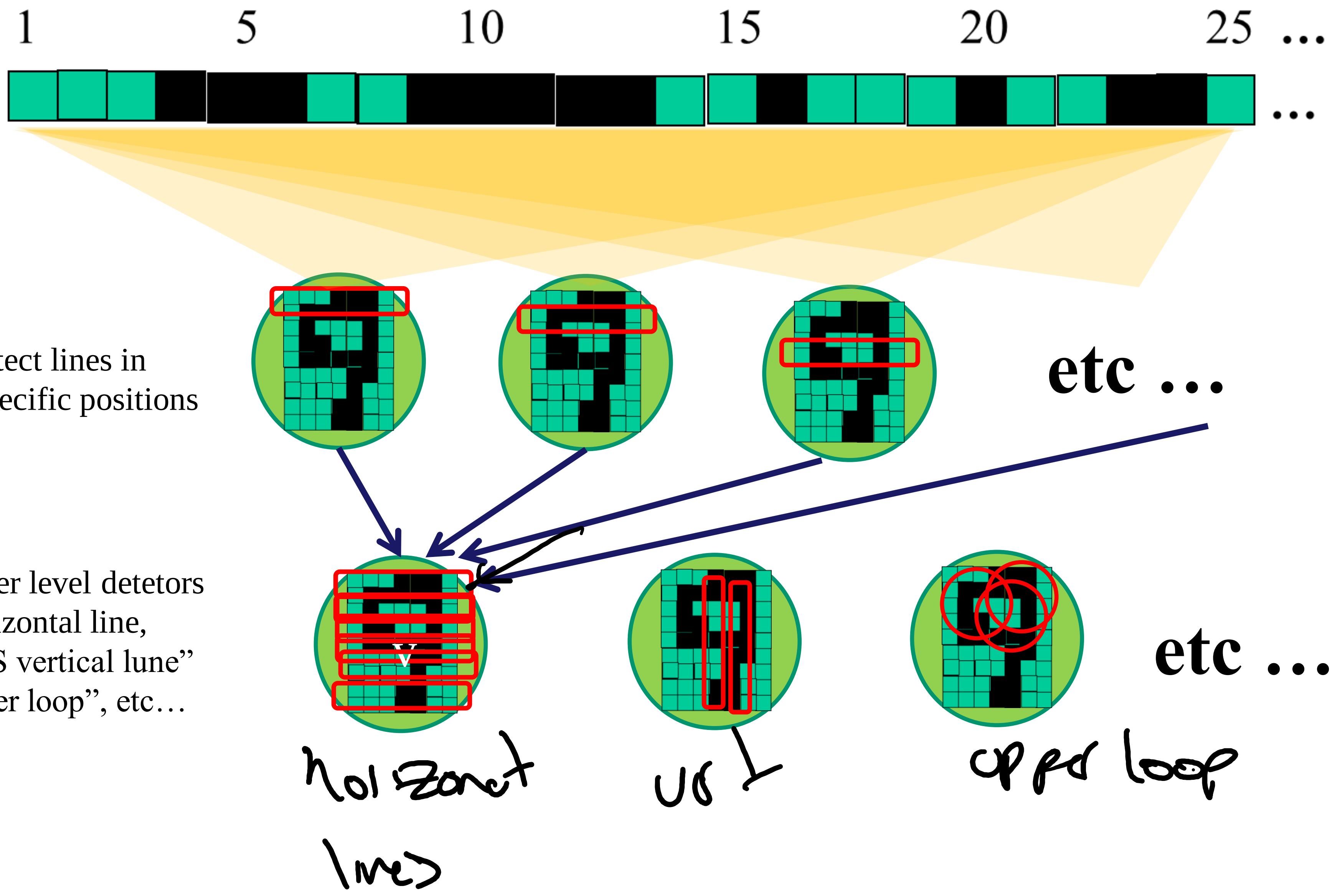
Figure 1.2: *Examples of handwritten digits from U.S. postal envelopes.*



1

But what about position invariance ???
our example unit detectors were tied to
specific parts of the image individual hidden neurons

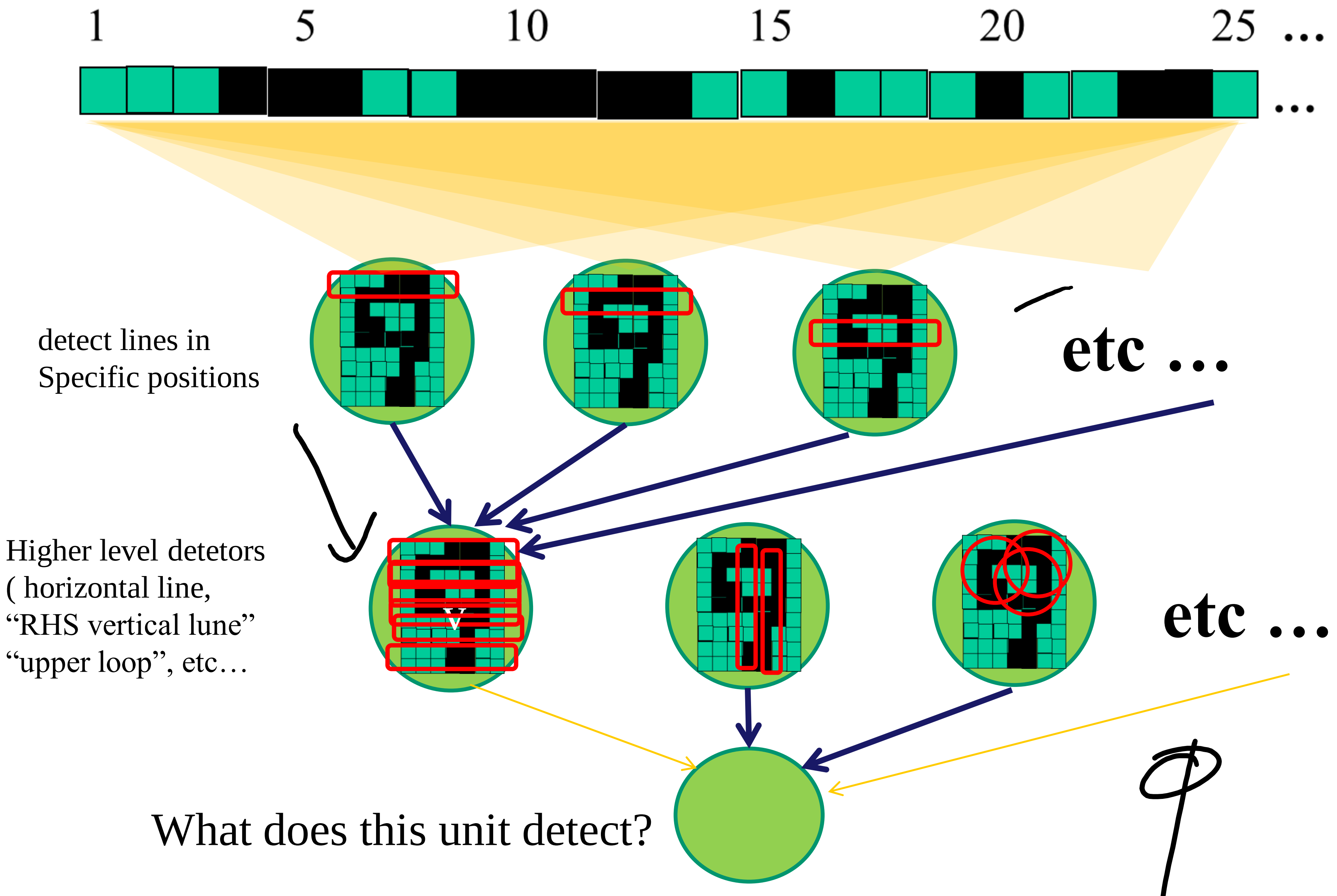
successive layers can learn higher-level features …



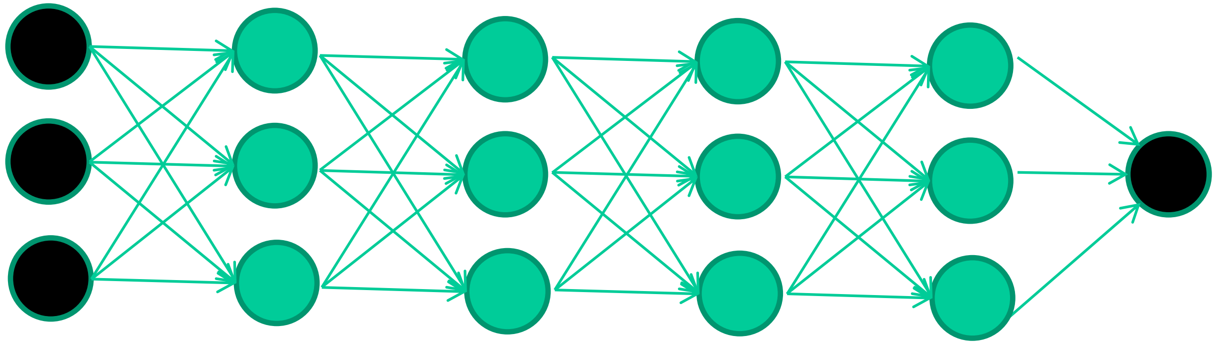
detect lines in
Specific positions

Higher level detetors
(horizontal line,
“RHS vertical lune”
“upper loop”, etc…

successive layers can learn higher-level features …

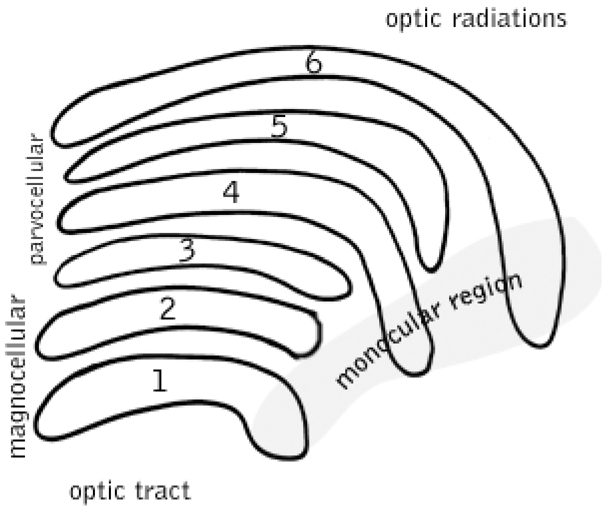


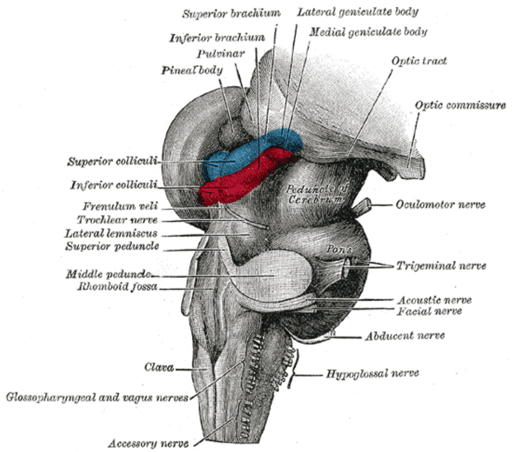
So: *multiple layers make sense*



So: *multiple layers make sense*

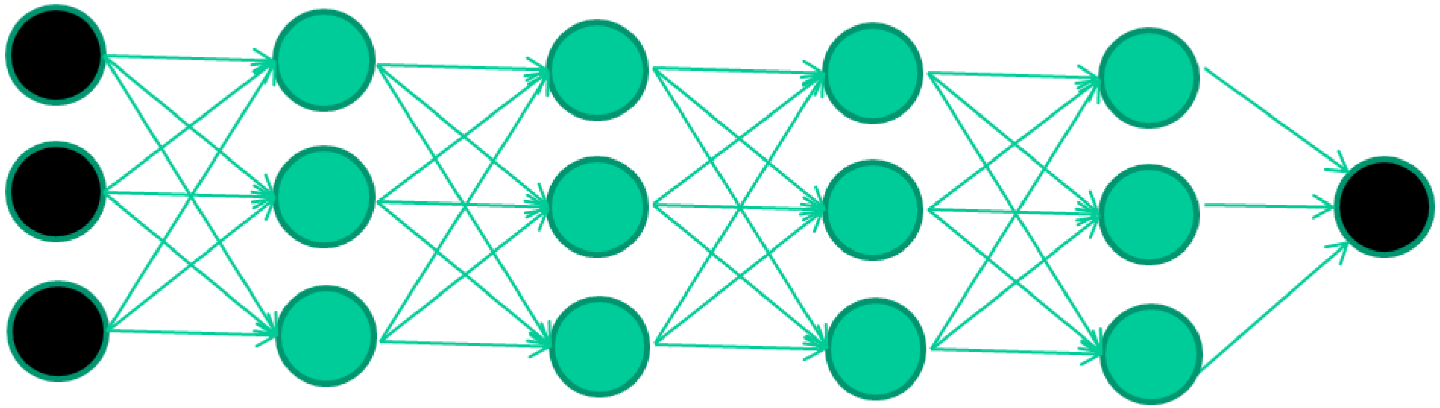
Your brain works that way



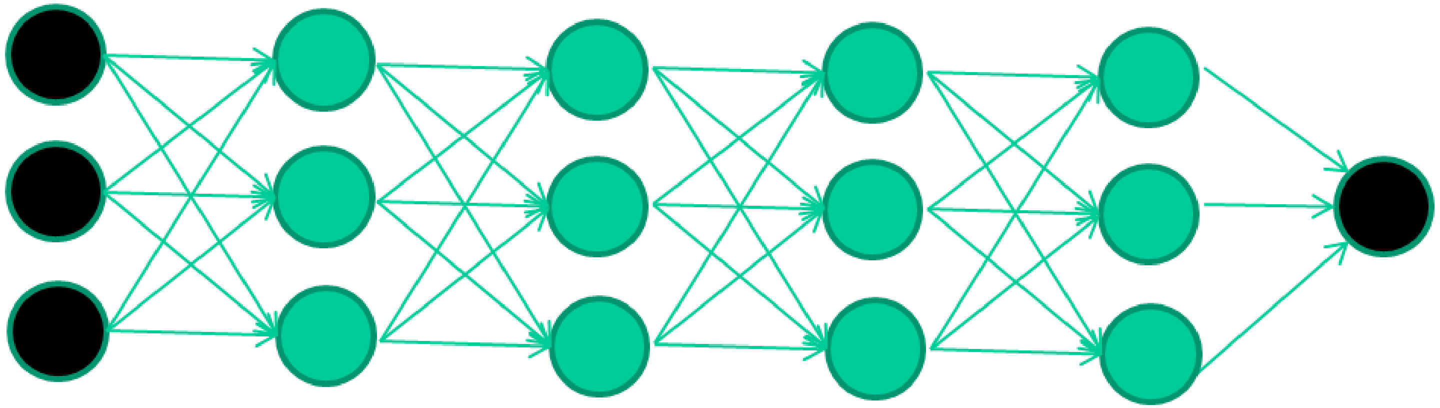


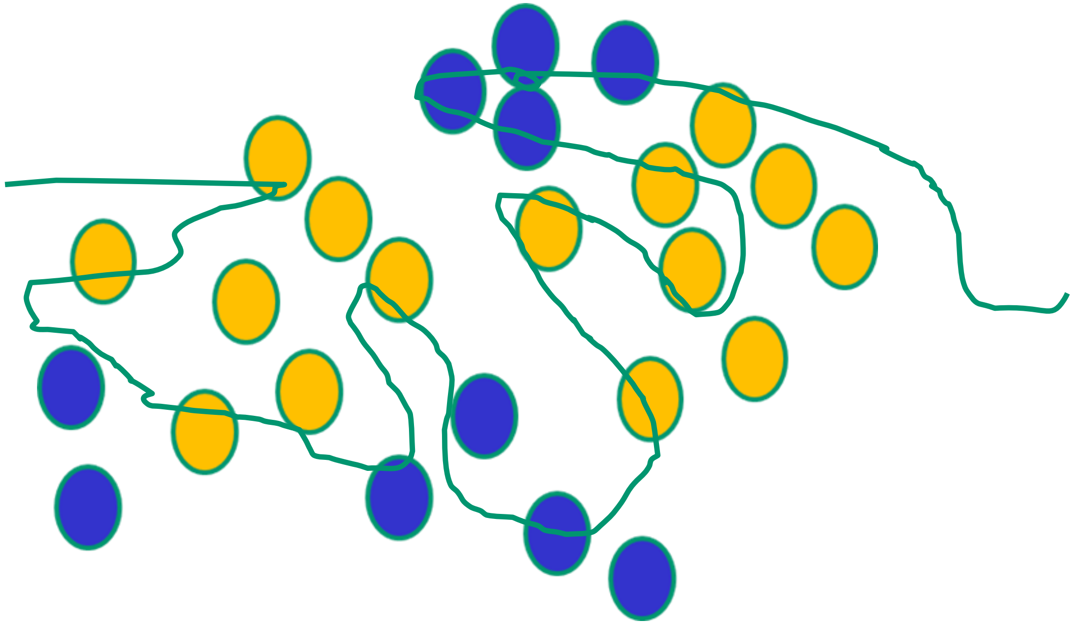
So: *multiple layers make sense*

Many-layer neural network architectures should be capable of learning the true underlying features and 'feature logic', and therefore generalise very well …



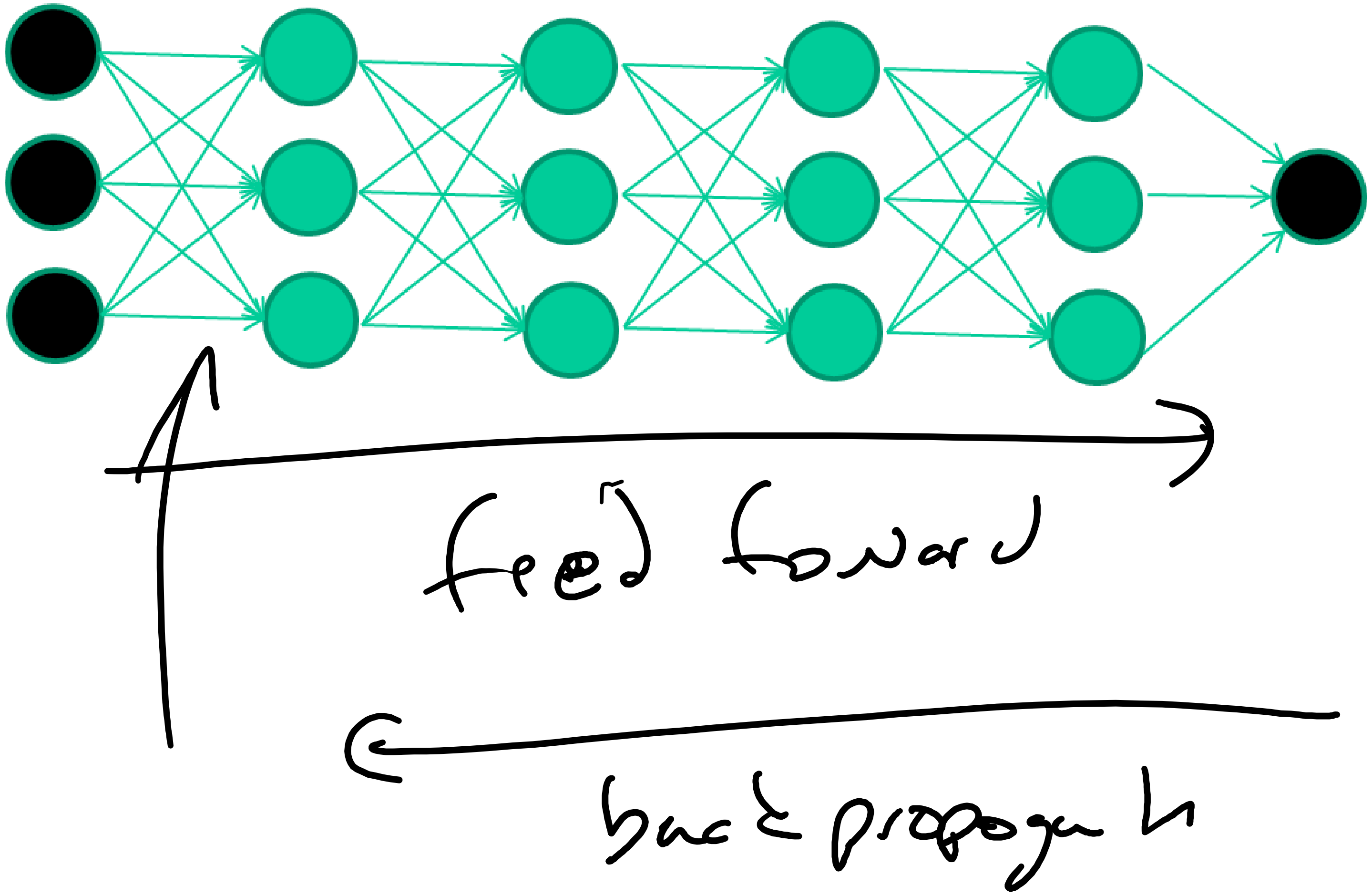
But, until very recently, our weight-learning algorithms simply did not work on multi-layer architectures



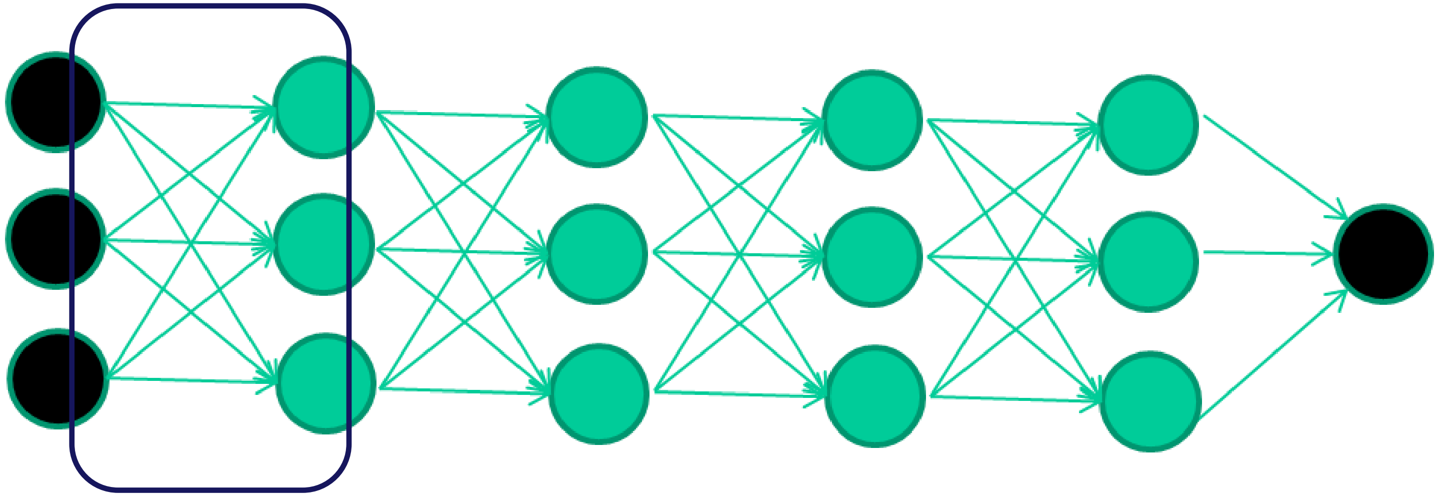


Along came deep learning …

The new way to train multi-layer NNs…

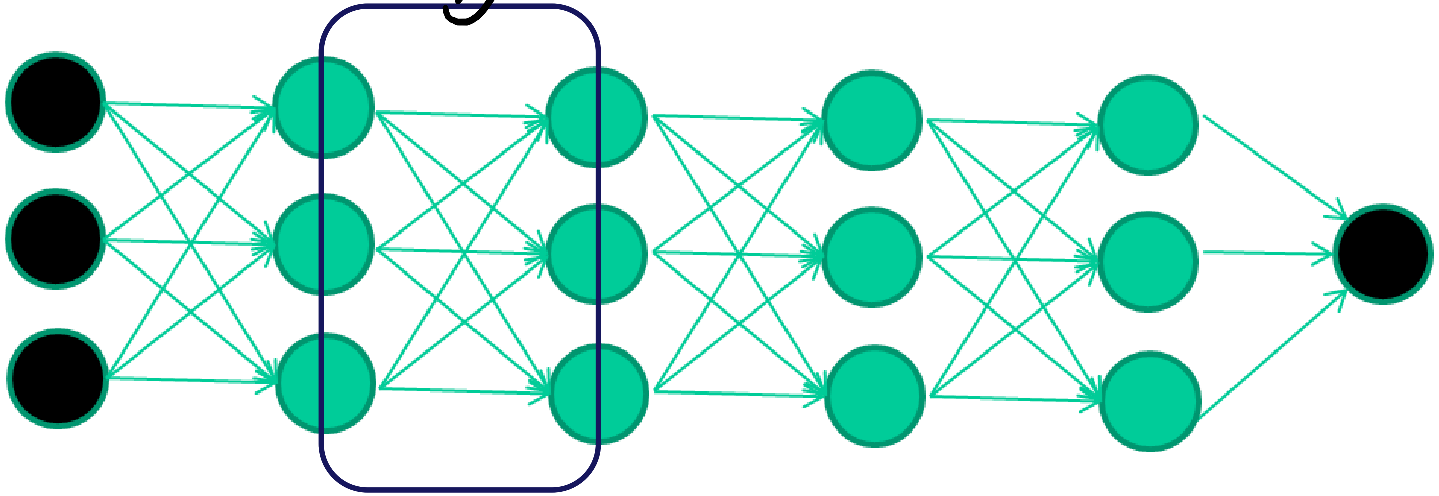


The new way to train multi-layer NNs…



Train **this** layer first

The new way to train multi-layer NNs…



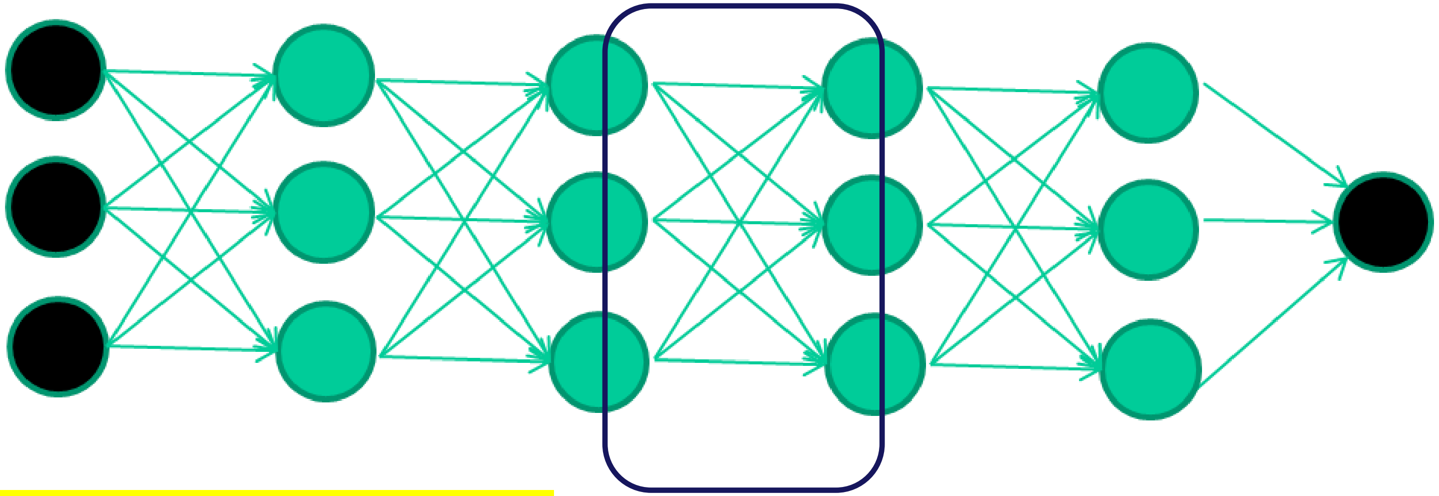
o

Train **this** layer first

then **this** layer

how to train here?
we can't know the "correct" value

The new way to train multi-layer NNs…

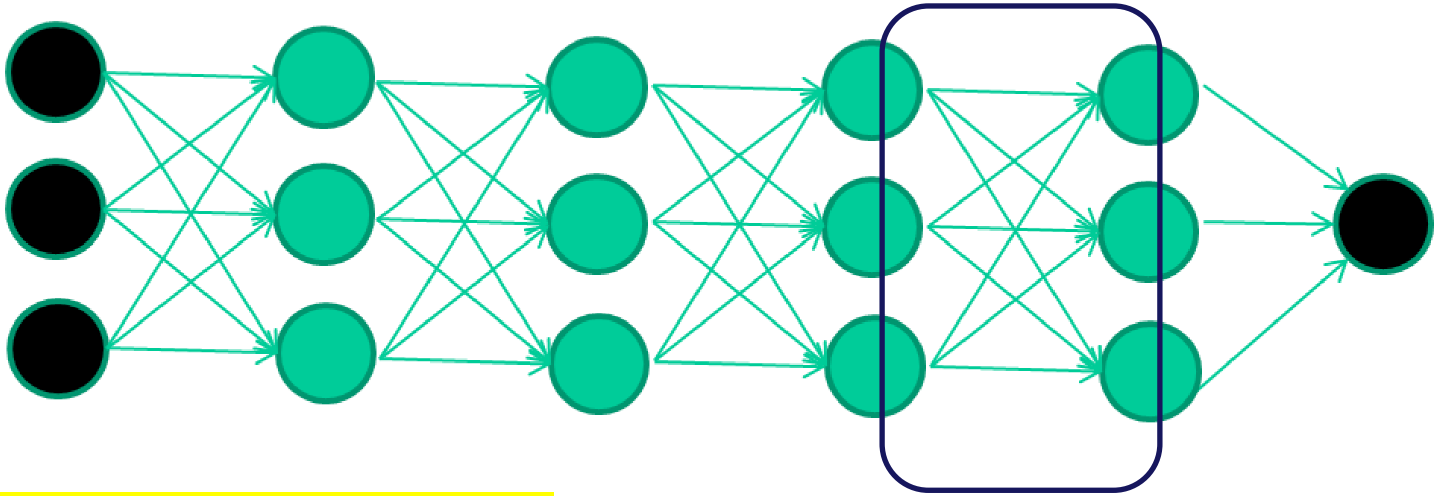


Train **this** layer first

then **this** layer

then **this** layer

The new way to train multi-layer NNs…



Train **this** layer first

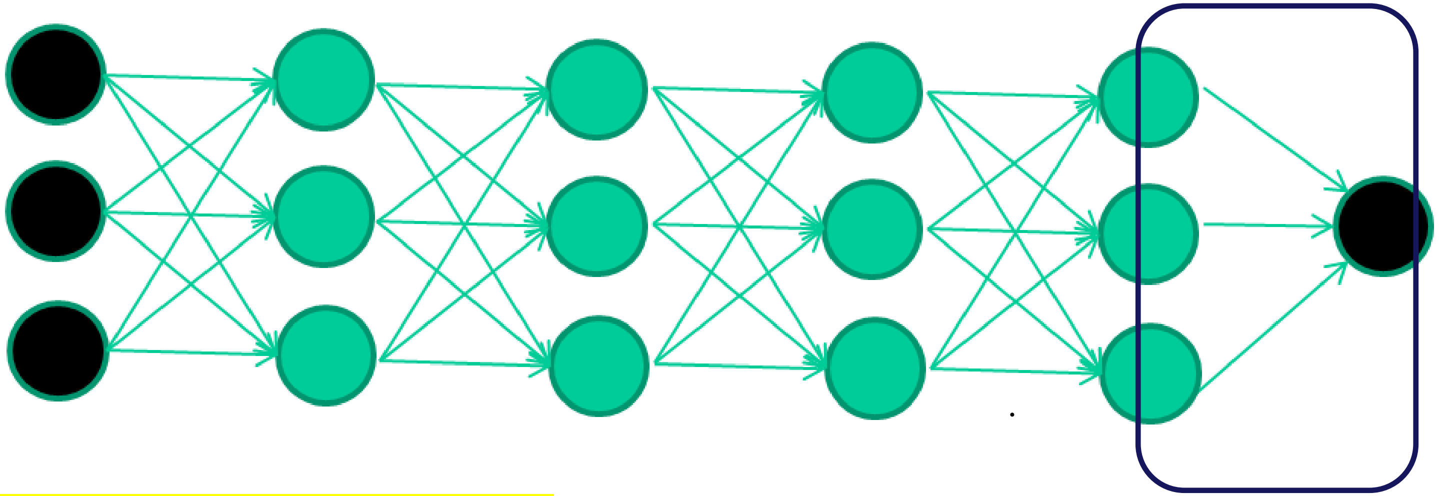
then **this** layer

then **this** layer

then **this** layer

The new way to train multi-layer NNs…

want to enforce independence



— this is our ML model NN

Train **this** layer first

then **this** layer

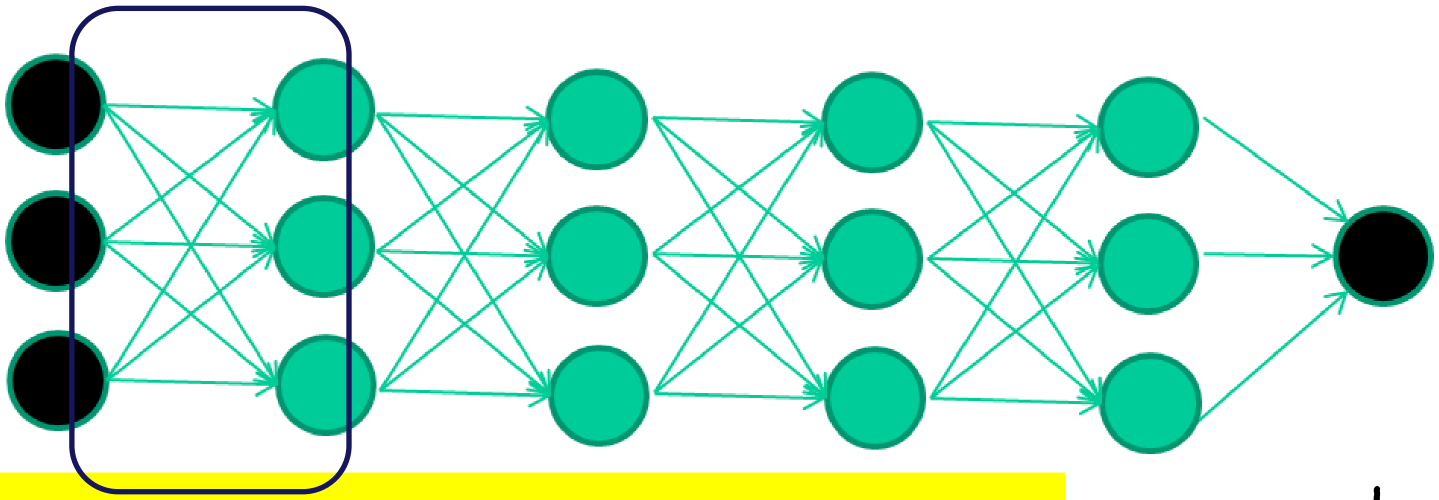
then **this** layer

then **this** layer

finally **this** layer

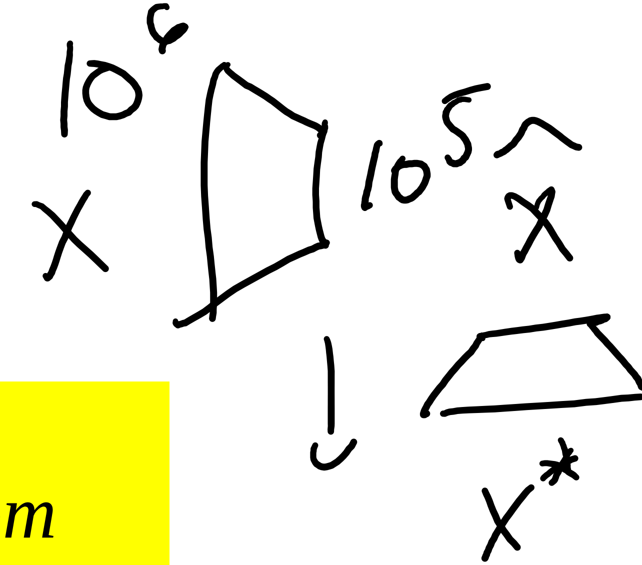
100

The new way to train multi-layer NNs…



EACH of the (non-output) layers is trained to be an ***auto-encoder***

Basically, it is forced to learn good features that describe what comes from the previous layer



X is the data

produce y s.t. $|y| < |x|$

and

Auto Encoders

The *auto encoder* idea is motivated by the concept of a good representation.

- For example, for a classifier, a good representation can be defined as one that will yield a better performing classifier.

An *encoder* is a deterministic mapping f_θ that transforms an input vector $\boldsymbol{x}$ into hidden representation $\boldsymbol{y}$

- $\theta = \{\mathbf{W}, b\}$, where $\mathbf{W}$ is the weight matrix and b is bias (an offset vector)

A *decoder* maps back the hidden representation $\boldsymbol{y}$ to the reconstructed input $\boldsymbol{z}$ via g_θ.

Auto encoding: compare the reconstructed input z to the original input $\boldsymbol{x}$ and try to minimize this error to make $\boldsymbol{z}$ as close as possible to $\boldsymbol{x}$. ←

Train these two together

$X \rightarrow y \rightarrow$ ML $\rightarrow$ Label

De-noising Auto Encoders

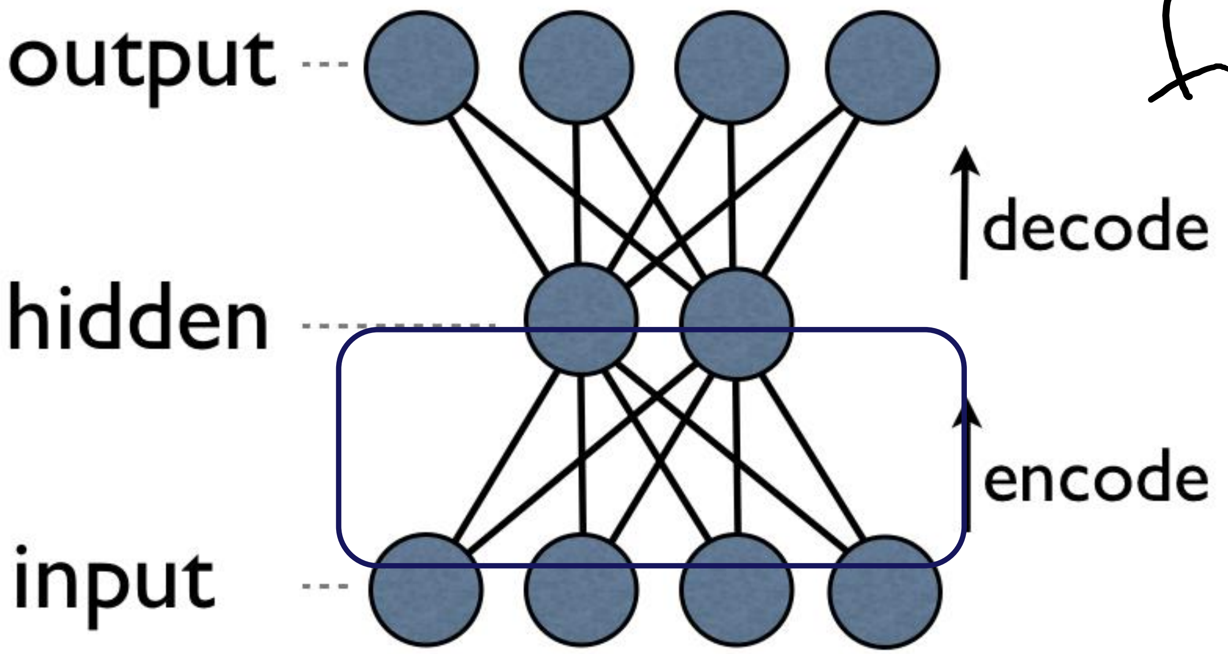
In Vincent et al. (2010), *"a **good representation** is one that can be obtained **robustly** from a **corrupted input** and that will be useful for **recovering** the corresponding **clean input**."*

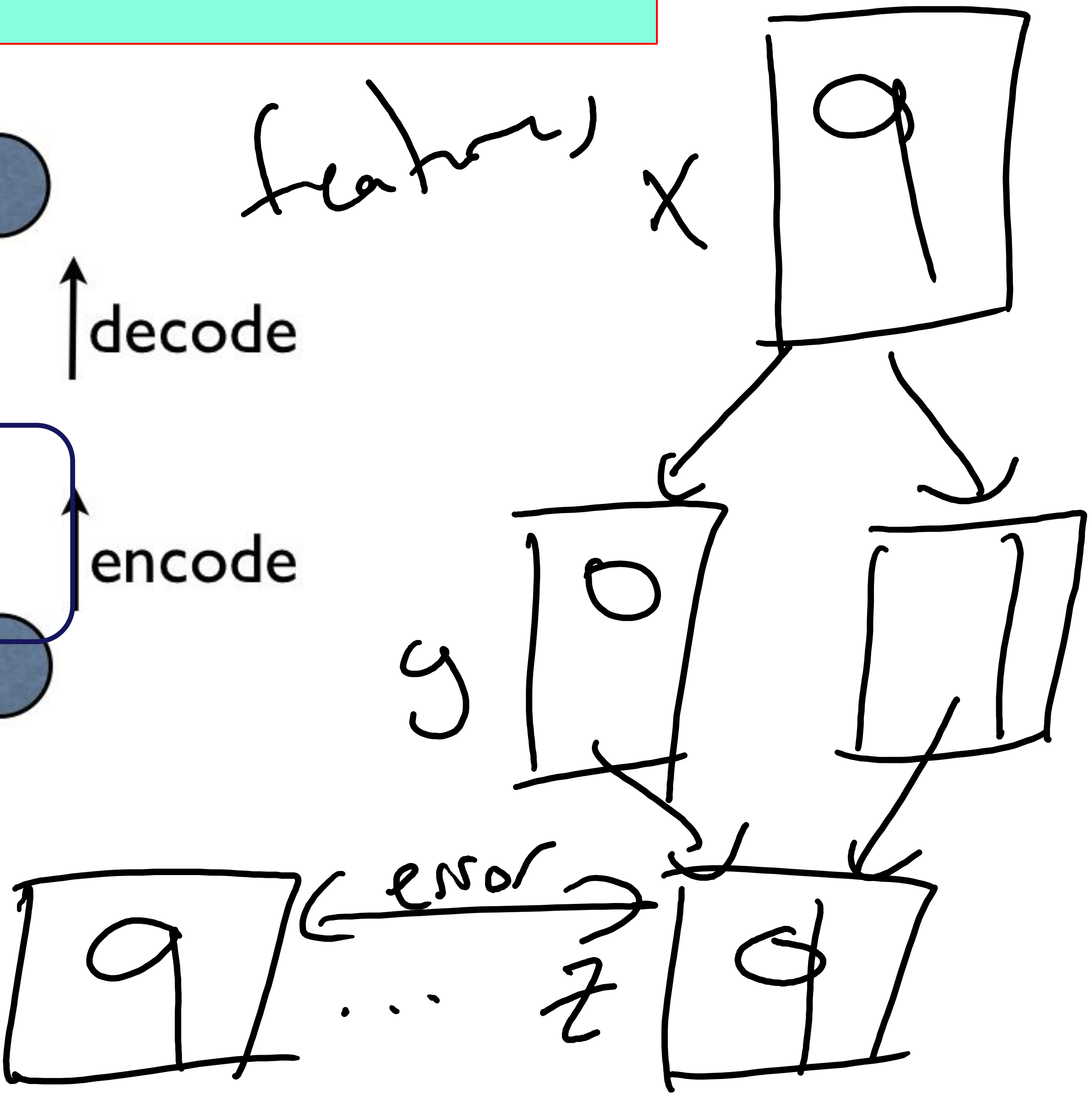
- The higher level representations are relatively stable and robust to input corruption.
- It is necessary to extract features that are useful for representation of the input distribution.

need the decoder to produce labelled data

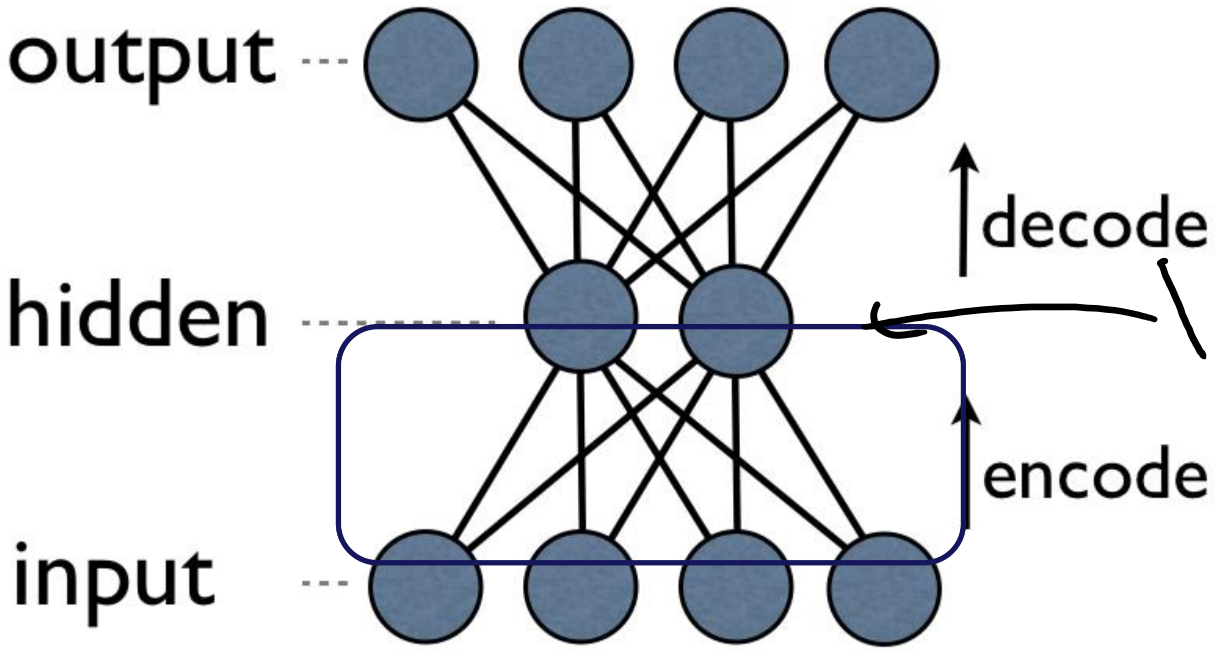
In de-noising auto encoders, the partially *corrupted* output is cleaned (de-noised). algorithm

an auto-encoder is trained, with an absolutely standard weight-adjustment algorithm to reproduce the input





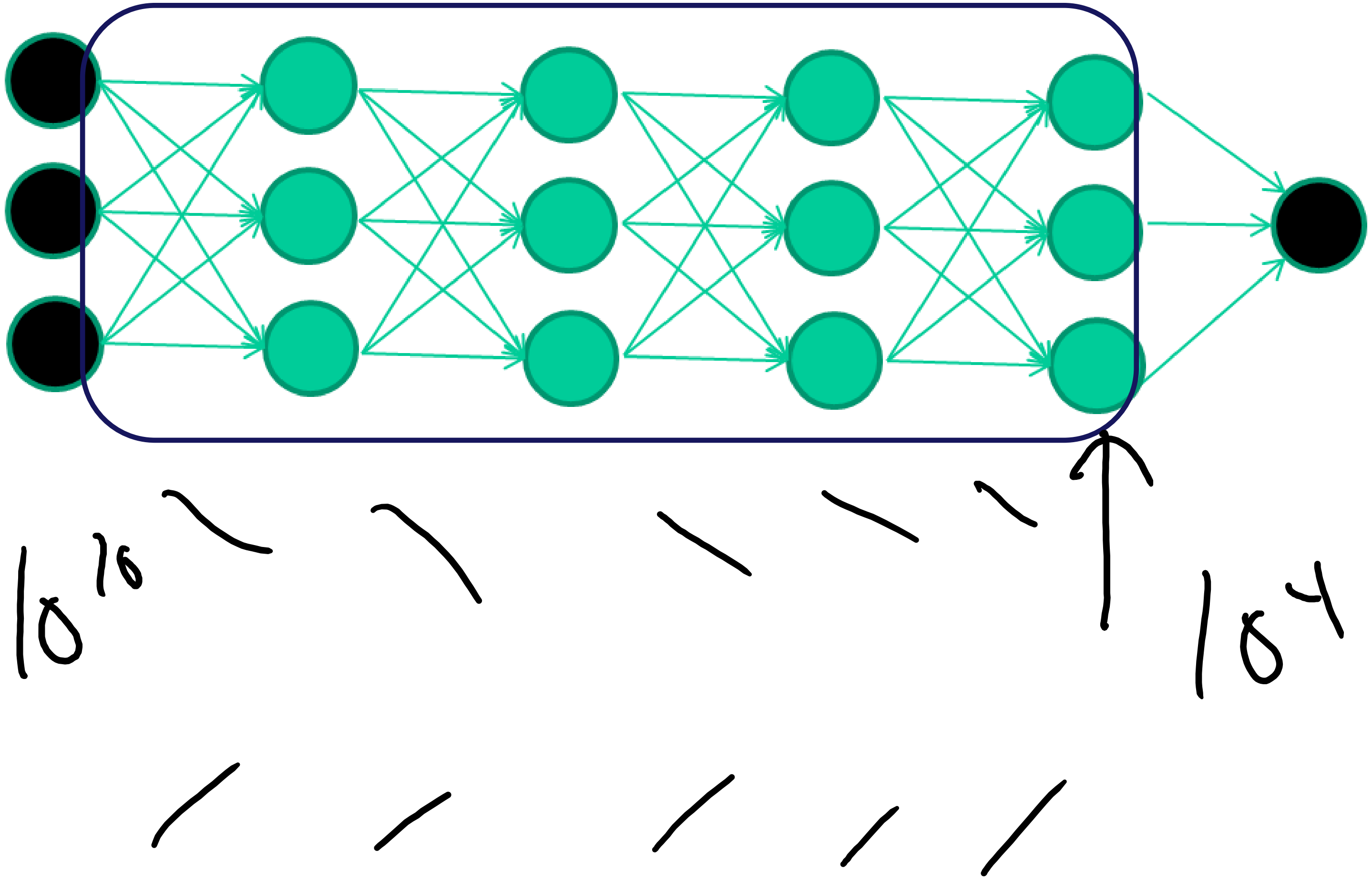
an auto-encoder is trained, with an absolutely standard weight-adjustment algorithm to <u>reproduce the input</u>



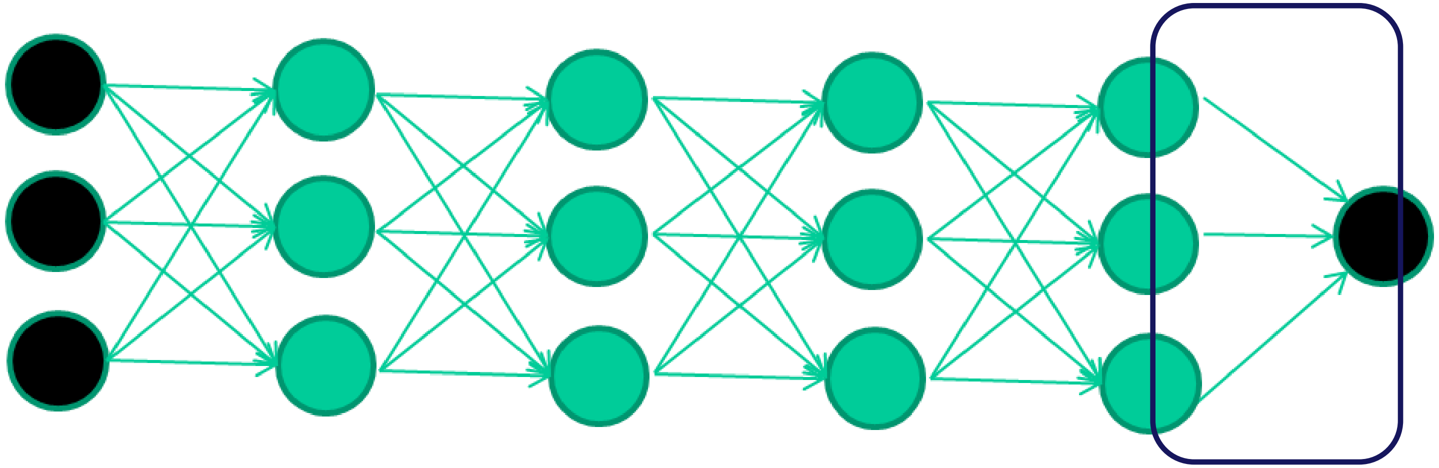
learned extractors

By making this happen with (many) fewer units than the inputs, this forces the ‘hidden layer’ units to become good feature detectors

intermediate layers are each trained to be auto encoders (or similar)



Final layer trained to predict class based on outputs from previous layers

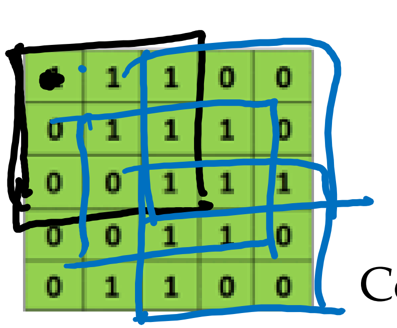


Convolutional Neural Networks (CNNs)

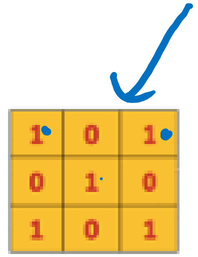
Main CNN idea for text:
Compute vectors for n-grams and group them afterwards

nlp

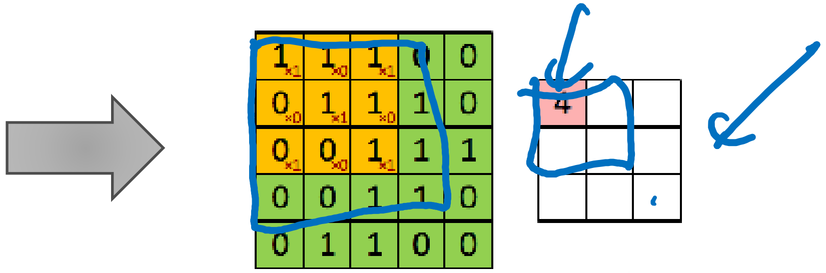
Example: “this takes too long” compute vectors for:
This takes, takes too, too long, this takes too, takes too long, this takes too long



Input matrix



Convolutional 3x3 filter



Image

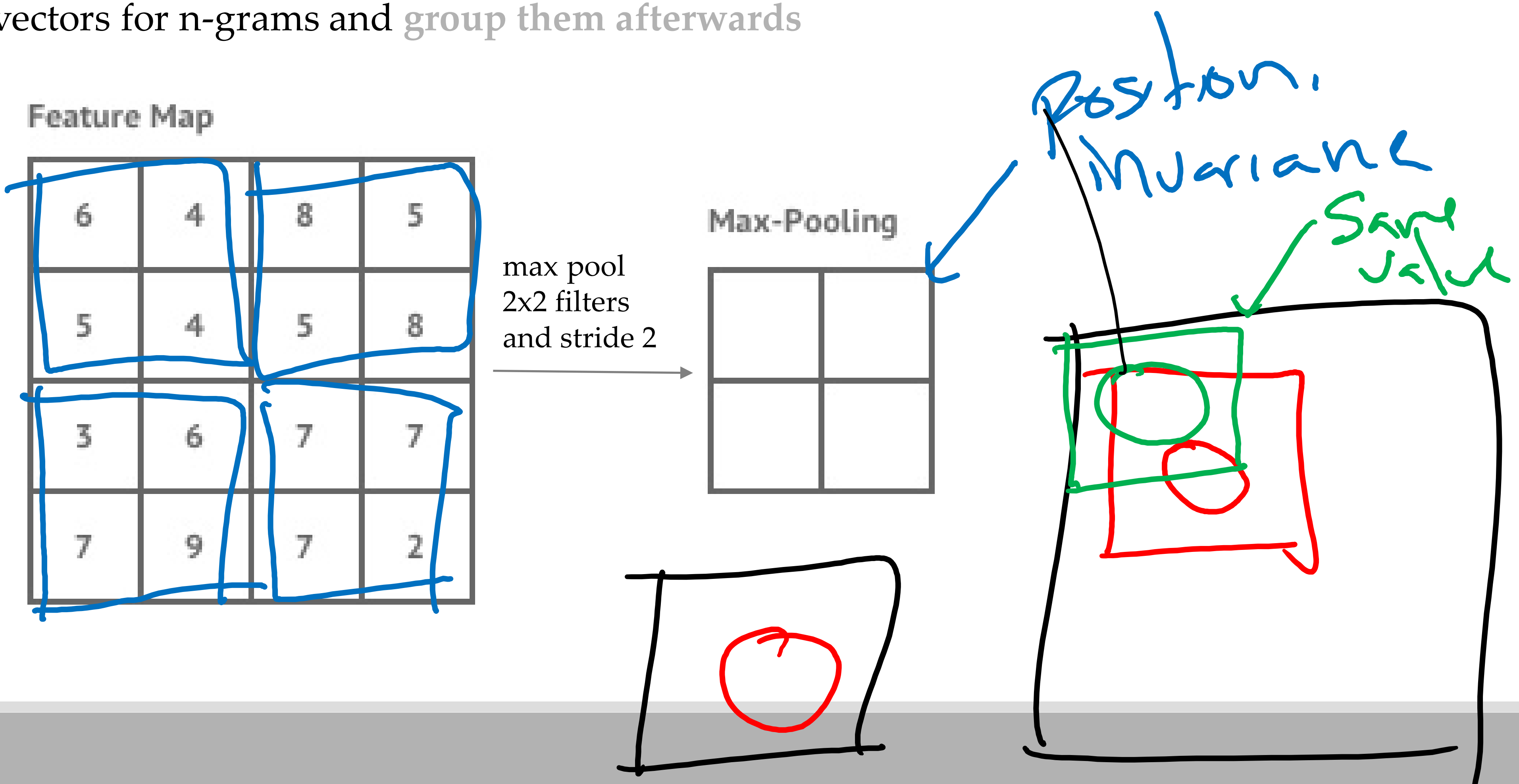
Convolved Feature

5x5 → 3x3?

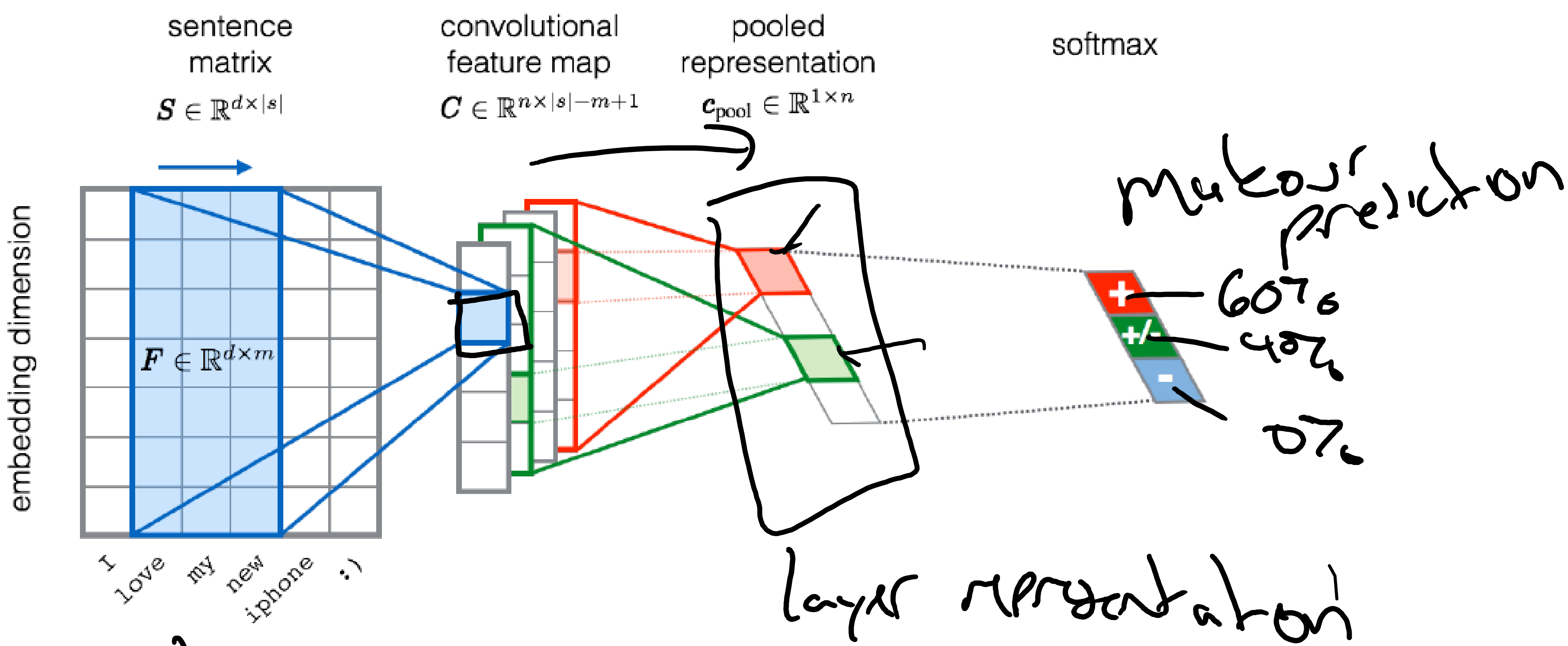
Same weights

Convolutional Neural Networks (CNNs)

Main CNN idea for text:
Compute vectors for n-grams and group them afterwards



Convolutional Neural Networks (CNNs)

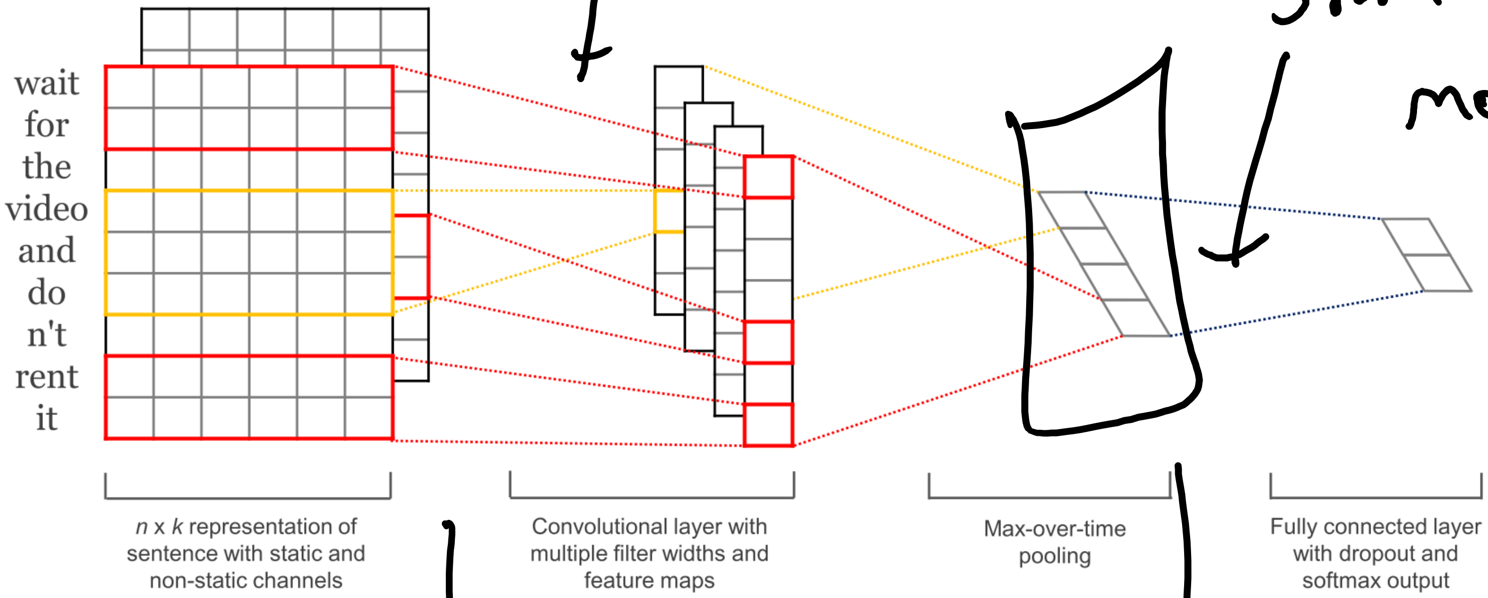


nlp

Convolutional Neural Networks (CNNs)

weights we need to learn

standard ML model



we can repeat this multiple times

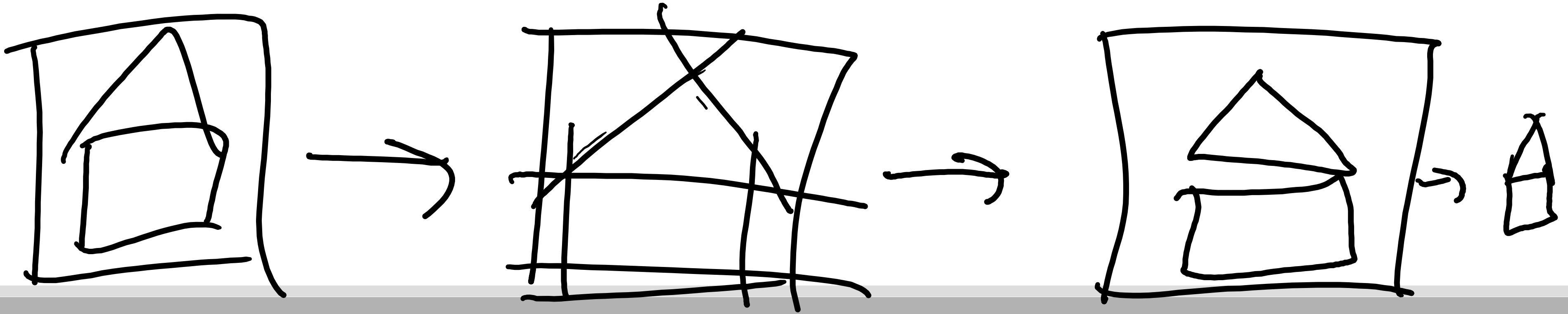
CNN Architecture

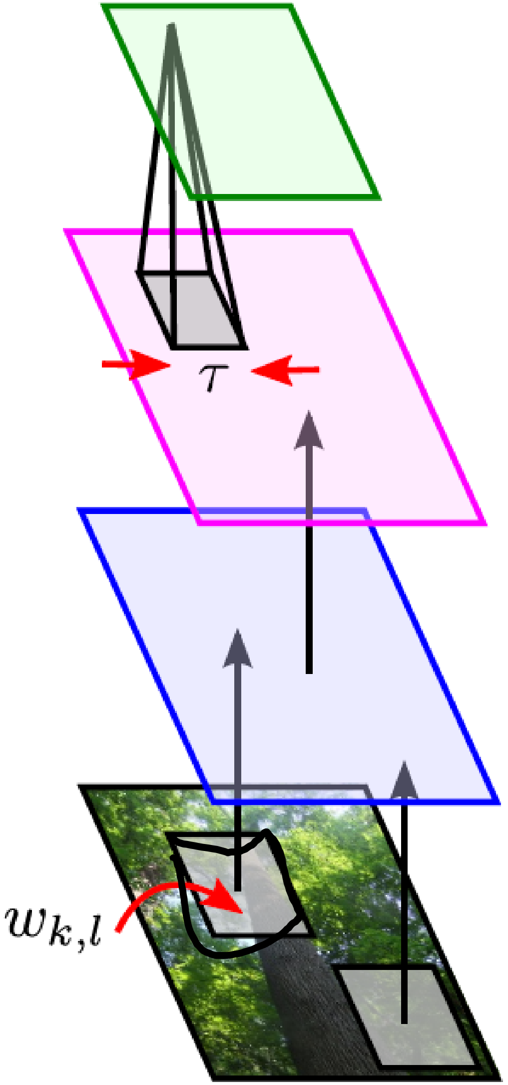
Intuition: Neural network with specialized connectivity structure,
- Stacking multiple layers of feature extractors
- Low-level layers extract local features.
- High-level layers extract learn global patterns.

regional relevance
+ locality

A CNN is a list of layers that transform the input data into an output class/prediction.

There are a few distinct types of layers:
- Convolutional layer
- Non-linear layer
- Pooling layer





$$x_{i,j} = \max_{|k|<\tau, |l|<\tau} y_{i-k,j-l}$$

mean or subsample also used

pooling stage

Feature maps of a larger region are combined.

$$y_{i,j} = f(a_{i,j})$$

e.g. $f(a) = [a]_+$

$f(a) = \text{sigmoid}(a)$ ReLU

non-linear stage

Feature maps are trained with neurons.

$$a_{i,j} = \sum_{k,l} w_{k,l} z_{i-k,j-l}$$

only parameters

convolutional stage

Each sub-region yields a feature map, representing its feature.

Shared weights

$z_{i,j}$

input image

Images are segmented into sub-regions.

CNN Architecture: Convolutional Layer

e.g.
circle
horizontal line

The core layer of CNNs

The convolutional layer consists of a set of filters.

- Each filter covers a spatially small portion of the input data.

Each filter is convolved across the dimensions of the input data, producing a multidimensional feature map.

- As we convolve the filter, we are computing the dot product between the parameters of the filter and the input.

have a higher value

Intuition: the network will learn filters that activate when they see some specific type of feature at some spatial position in the input.

The key architectural characteristics of the convolutional layer is local connectivity and shared weights.

CNN Convolutional Layer: Local Connectivity

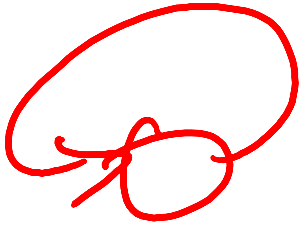
Neurons in layer m are only connected to 3 adjacent neurons in the m-1 layer.

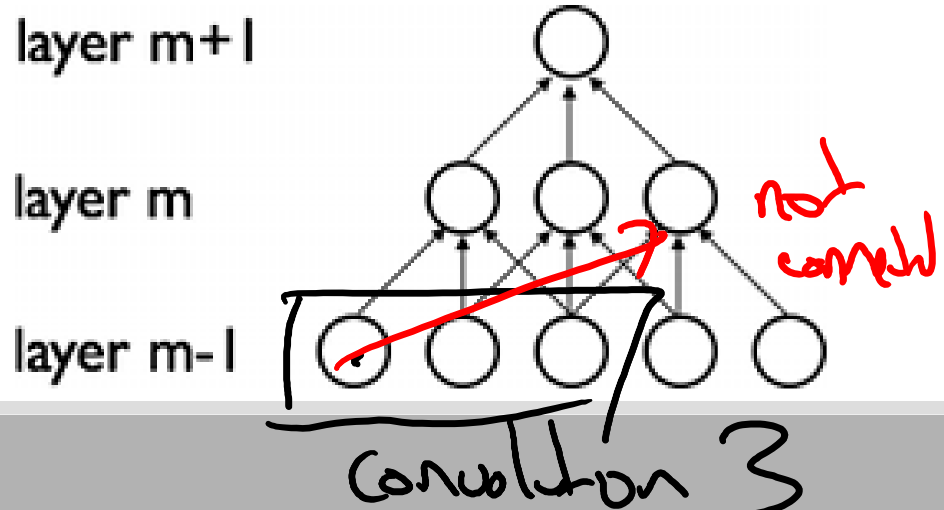
Neurons in layer m+1 have a similar connectivity with the layer below.

Each neuron is unresponsive to variations outside of its receptive field with respect to the input.

- Receptive field: small neuron collections which process portions of the input data

The architecture thus ensures that the learnt feature extractors produce the strongest response to a spatially local input pattern.





CNN Convolutional Layer: Shared Weights

We show 3 hidden neurons belonging to the same feature map (the layer right above the input layer).

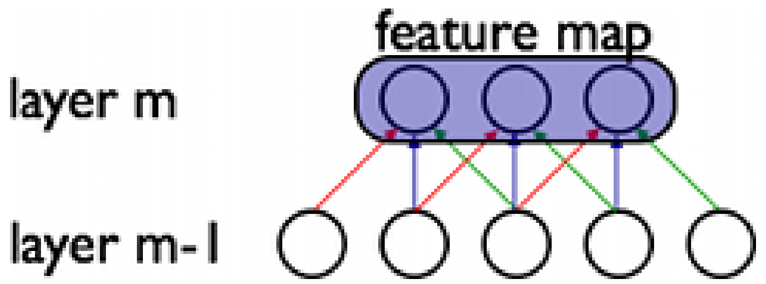
Weights of the same color are shared—constrained to be identical.

Gradient descent can still be used to learn such shared parameters, with only a small change to the original algorithm.

The gradient of a shared weight is simply the sum of the gradients of the parameters being shared.

Replicating neurons in this way allows for features to be detected regardless of their position in the input.

Additionally, weight sharing increases learning efficiency by greatly reducing the number of free parameters being learnt.



CNN Architecture: Non-linear Layer

Intuition: Increase the nonlinearity of the entire architecture without affecting the receptive fields of the convolution layer

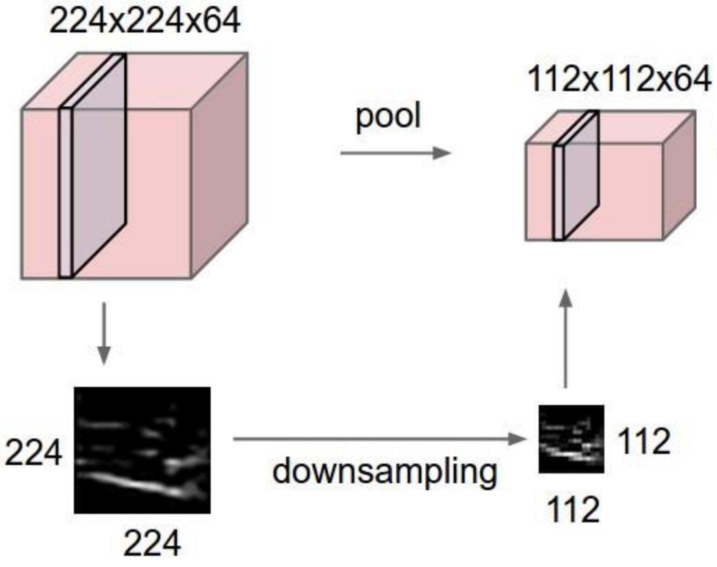
A layer of neurons that applies the non-linear activation function, such as,

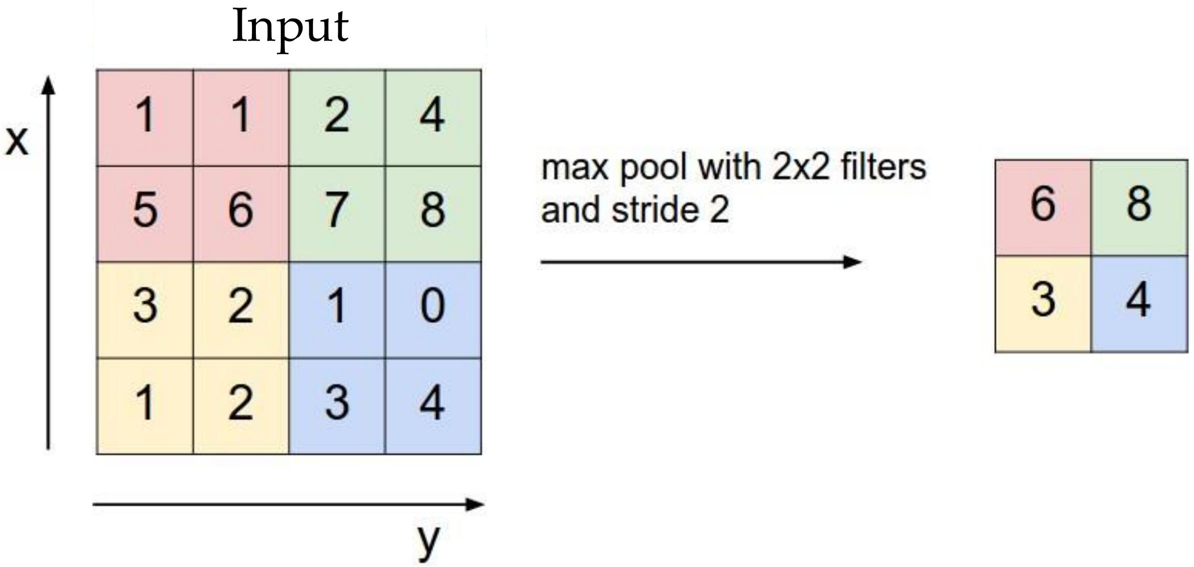
- $f(x) = \max(0, x)$
- $f(x) = \tanh x$
- $f(x) = |\tanh x|$
- $f(x) = (1 + e^{-x})^{-1}$

CNN Architecture: Pooling Layer

Intuition: to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network, and hence to also control overfitting

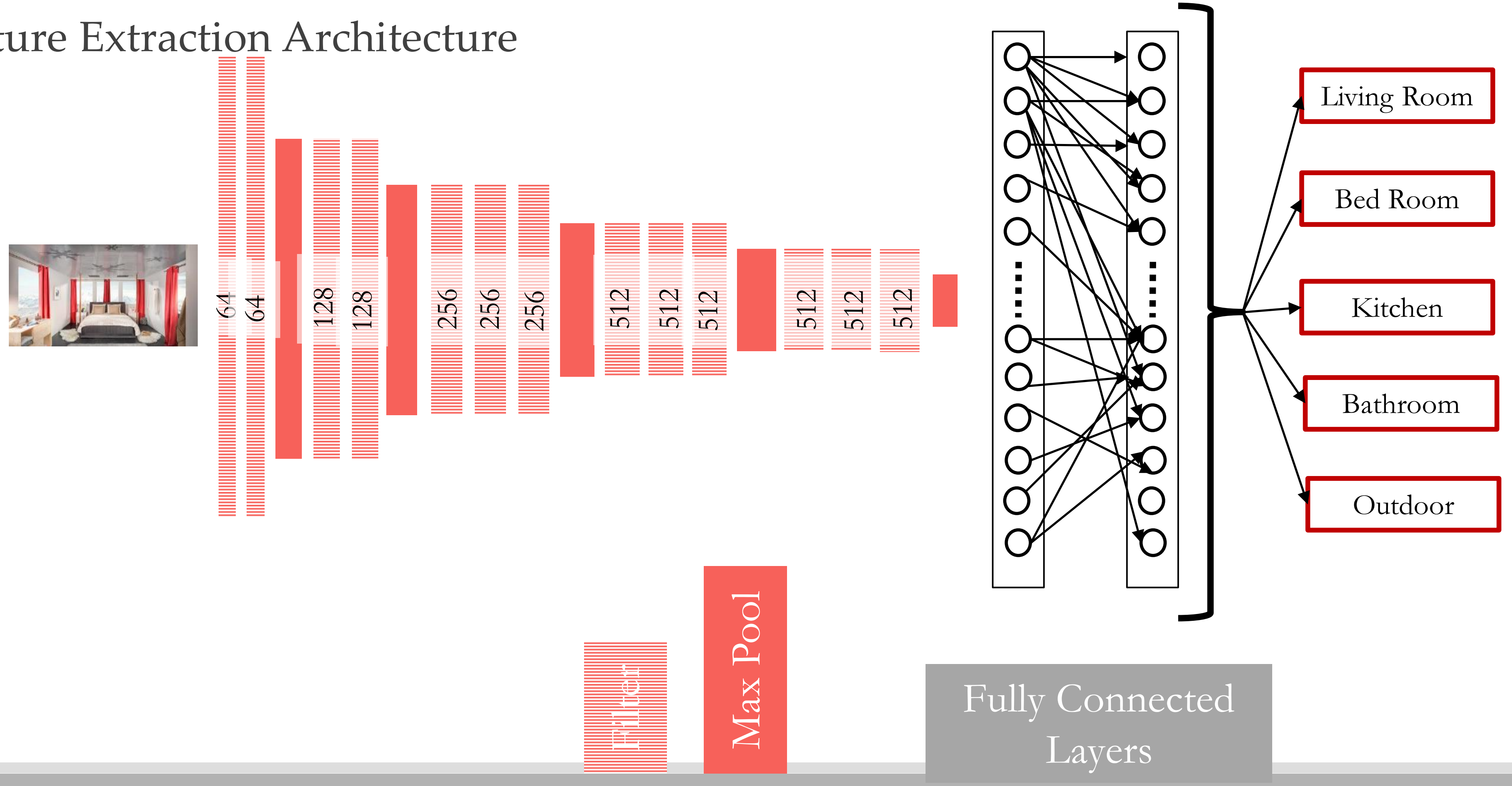
Pooling partitions the input image into a set of non-overlapping rectangles and, for each such sub-region, outputs the maximum value of the features in that region.





Convolutional Neural Network

Feature Extraction Architecture



Conclusion

Deep learning = Learning Hierarchical Representations

Deep learning is thriving in big data analytics, including image processing, speech recognition, and natural language processing.

Deep learning has matured and is very promising as an artificial intelligence method.

Still has room for improvement:

- Scaling computation
- Optimization
- Bypass intractable marginalization
- More disentangled abstractions
- Reasoning from incrementally added facts

Package Resources

Name	Language	Link	Note
Pylearn2	Python	http://deeplearning.net/software/pylearn2/	A machine learning library built on Theano
Theano	Python	http://deeplearning.net/software/theano/	A python deep learning library
Caffe	C++	http://caffe.berkeleyvision.org/	A deep learning framework by Berkeley
Torch	Lua	http://torch.ch/	An open source machine learning framework
Overfeat	Lua	http://cilvr.nyu.edu/doku.php?id=code:start	A convolutional network image processor
Deeplearning 4j	Java	http://deeplearning4j.org/	A commercial grade deep learning library
Word2vec	**C**	**https://code.google.com/p/word2vec/**	**Word embedding framework**
GloVe	C	http://nlp.stanford.edu/projects/glove/	Word embedding framework
Doc2vec	C	https://radimrehurek.com/gensim/models/doc2vec.html	Language model for paragraphs and documents
StanfordNLP	**Java**	**http://nlp.stanford.edu/**	**A deep learning-based NLP package**
TensorFlow	**Python**	**http://www.tensorflow.org**	**A deep learning based python library**