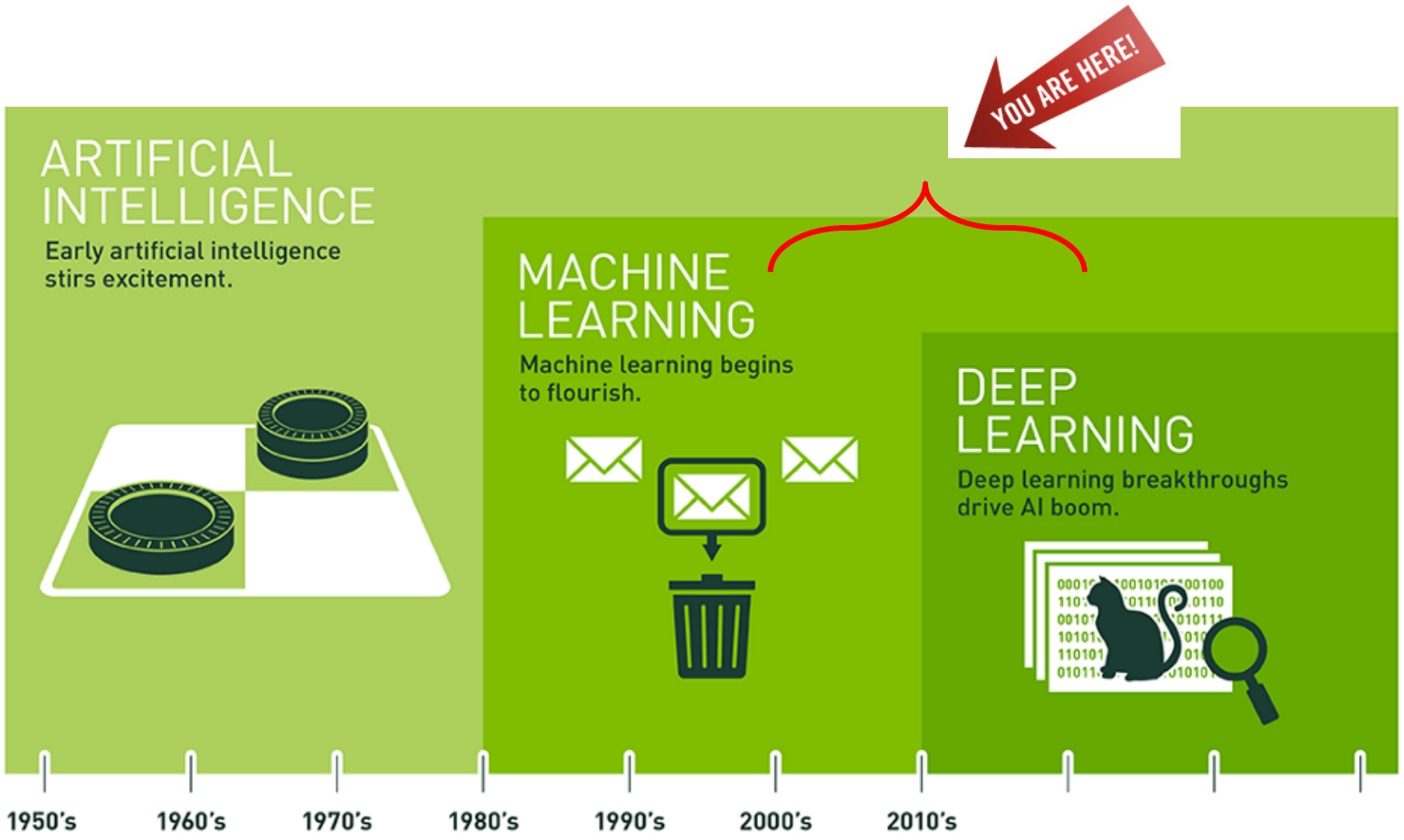
Artificial Intelligence: Deep Learning, CNNs

many slides from: Y. Bengio, A. Ng and Y. LeCun

Today

1. Motivation
2. Feature Learning
3. Early Training of Deep Neural Networks
4. CNNs for Image Processing
5. Conclusion

History of AI



https://www.linkedin.com/pulse/goedels-incompleteness-theorem-emergence-ai-eberhard-schoeneburg/

Deep Learning in the Academic Press (2012-2015)

nature International weekly journal of science

Deep learning

Yann LeCun, Yoshua Bengio & Geoffrey Hinton

Affiliations | Corresponding author

Nature **521**, 436–444 (28 May 2015) | doi:10.1038/nature14539

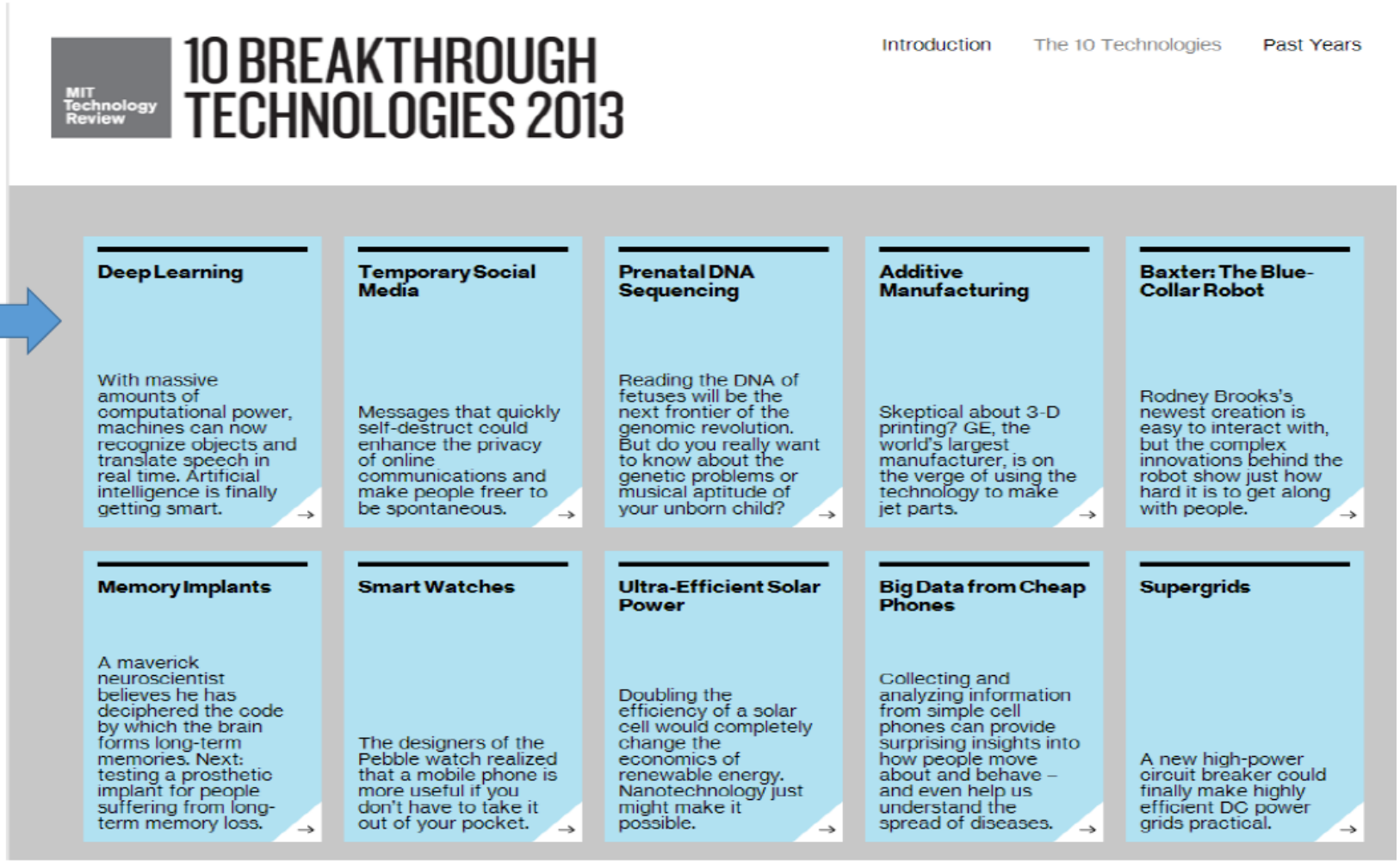
Received 25 February 2015 | Accepted 01 May 2015 | Published online 27 May 2015

Geoffrey Hinton, Li Deng, Dong Yu, George E. Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N. Sainath, and Brian Kingsbury

Deep Neural Networks for Acoustic Modeling in Speech Recognition

The shared views of four research groups

Deep Learning in the News (2013)



Slide from Yoshua Bengio, 2015

Deep Learning in the News (2012-2014)

The New York Tim

Monday, June 25, 2012 Last Update: 11:50 PM ET

EXCLUSIVE

Facebook, Google in 'Deep Learning' Arms Race

Yann LeCun, an NYU artificial intelligence researcher who now works for Facebook. *Photo: Josh Valcarcel/WIRED*

WIRED

NEWS BULLETIN

Google Beat Facebook for DeepMind

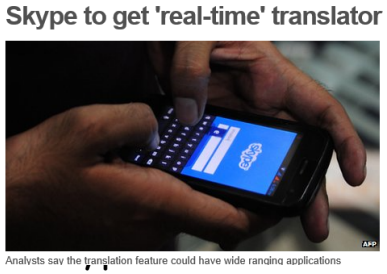
Google Acquires Artificial Intelligence Startup DeepMind For More Than $500M

Posted Jan 26, 2014 by Catherine Shu (@catherineshu)

Slide from Yoshua Bengio, 2015

Major Breakthroughs

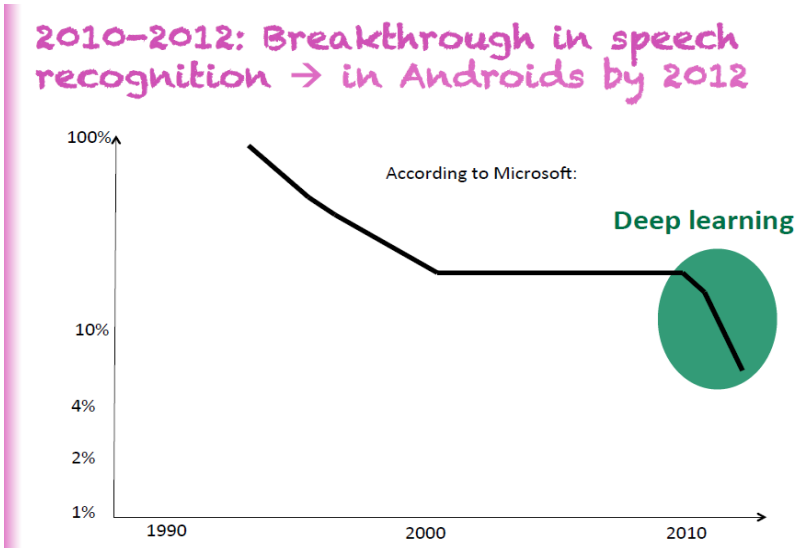
- Speech Recognition & Machine Translation (2010+)



Skype Translator

Google Translate

Google now

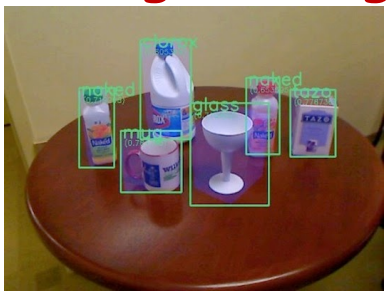


- Image Recognition & Computer Vision (2012+)
- Natural Language Processing (2014+)
- …

Figure from Yoshua Bengio, 2015

Major Breakthroughs

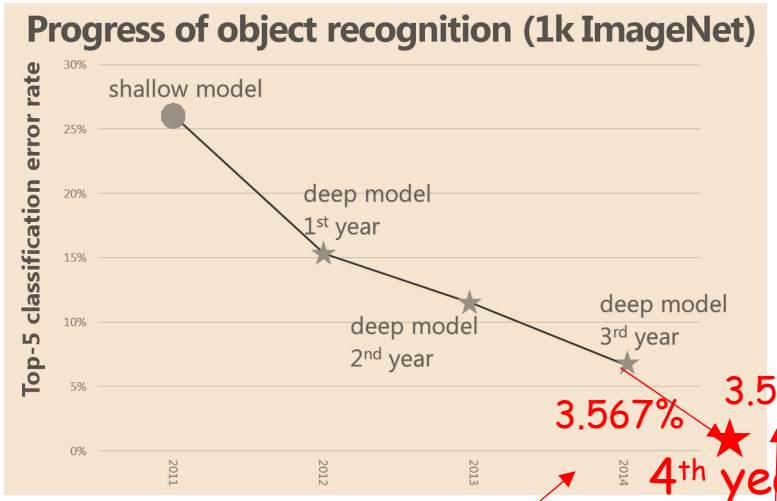
- Speech Recognition & Machine Translation (2010+)
- Image Recognition & Computer Vision (2012+)



Object recognition

Self driving cars

Google Photos



- Natural Language Processing (2014+)
- …

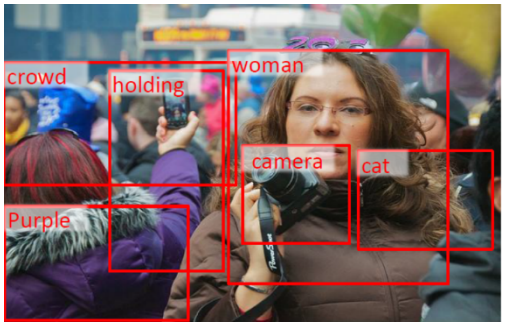
Figure from Yoshua Bengio, 2015

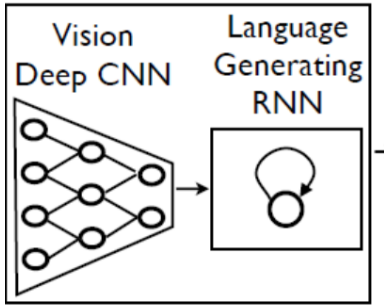
Major Breakthroughs

- Speech Recognition & Machine Translation (2010+)
- Image Recognition & Computer Vision (2012+)
- Natural Language Processing (2014+)

Joe went to the kitchen. Fred went to the kitchen. Joe picked up the milk.
Joe travelled to the office. Joe left the milk. Joe went to the bathroom.
Where is the milk now? A: office
Where is Joe? A: bathroom
Where was Joe before the office? A: kitchen

Question Answering





Machine-generated (but turker prefered)	a bicycle is parked next to a river
Human-annotated (but turker not prefered)	a bike sits parked next to a body of water

Image Captioning (deep vision + deep NLP)

Image Captioning: Better than humans?

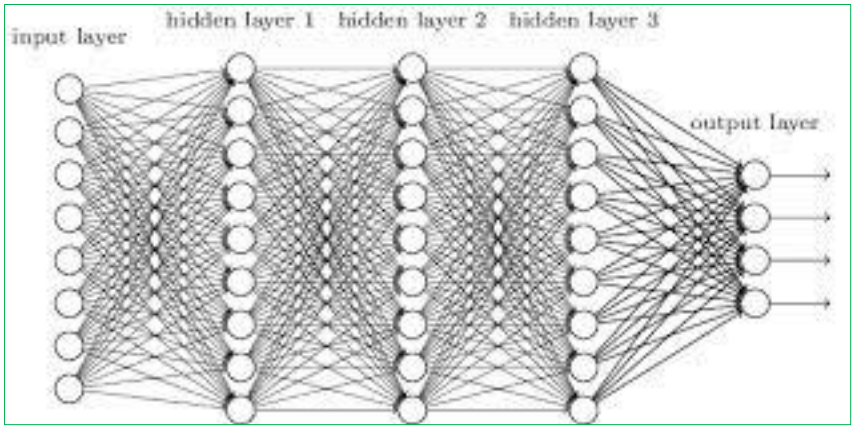
A	a woman in a kitchen preparing food
B	woman working on counter near kitchen sink preparing a meal

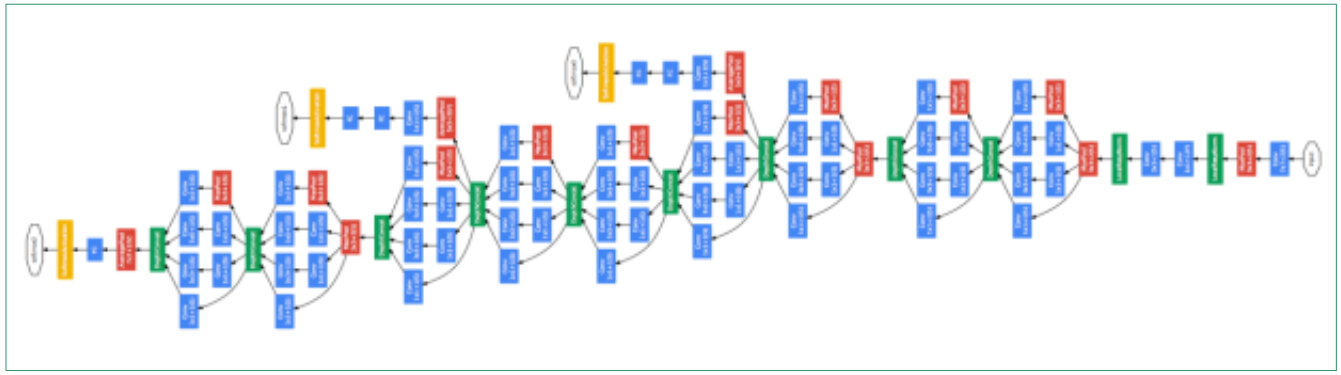
Today

1. Motivation
2. Feature Learning 

3. Early Training of Deep Neural Networks
4. CNNs for Image Processing
5. Deep Learning for NLP
6. Conclusion

A Deep Neural Net





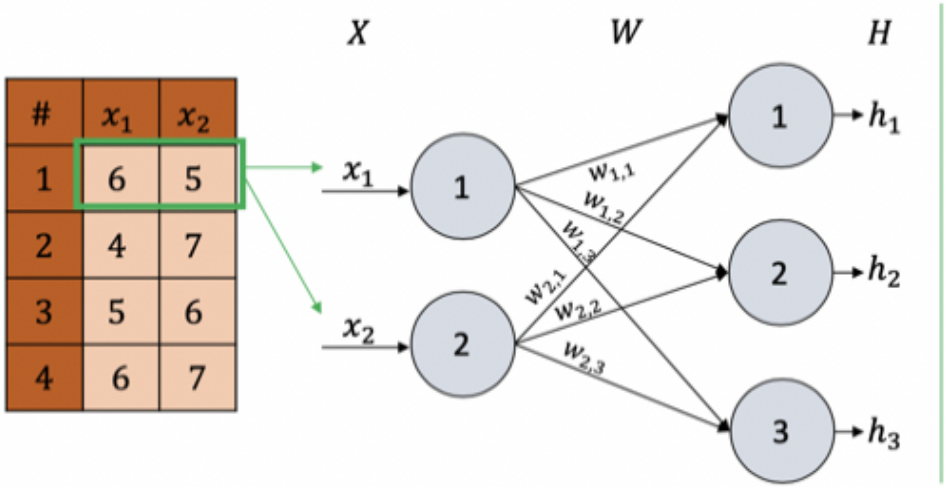
The Google "Inception" deep neural network architecture for image recognition (27 layers)

https://qph.ec.quoracdn.net/main-qimg-970d2b5f57b6b5cd13dc11f5371166b2-c

https://devblogs.nvidia.com/parallelforall/mocha-jl-deep-learning-julia/

Matrix Notation

We now represent inputs *x*, weights *W* and the bias weights *b* as matrices:



$$X \cdot W = H$$

$$\begin{bmatrix} x_1 & x_2 \end{bmatrix} \cdot \begin{bmatrix} W^1_{11} & W^1_{12} & W^1_{13} \\ W^1_{21} & W^1_{22} & W^1_{23} \end{bmatrix} = \begin{bmatrix} h_1 & h_2 & h_3 \end{bmatrix}$$

Input to hidden:

1. Compute net activation $net_h = x \cdot W_{ih} + b_{ih}$
2. Compute activation function (e.g., sigmod): $h = S(net_h)$

Hidden to output:

$$o = S(h \cdot W_{ho} + b_{ho})$$

→ Worksheet #5 ("Matrix Notation")

https://towardsdatascience.com/neural-networks-i-notation-and-building-blocks-817b1d2ea04b

Initial Drawbacks

1. Standard backpropagation with sigmoid activation function does not scale well with multiple layers
   - Weight of early layers change too slowly (no learning)
2. Overfitting
   - Large network -> lots of parameters -> increased capacity to "learn by heart"
3. Multilayered ANNs need lots of labeled data
   - Most data is not labeled :(

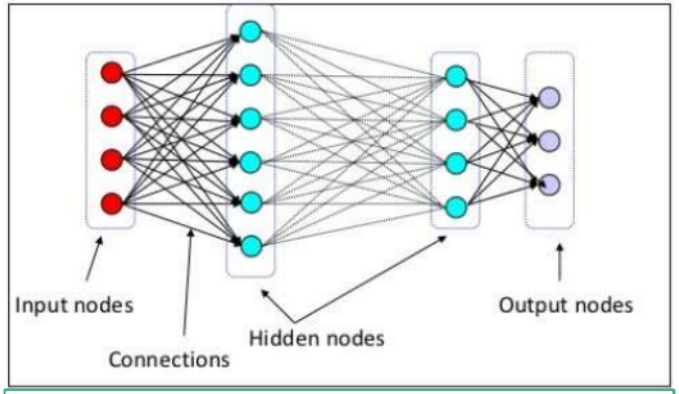
Initial Drawbacks (1)

1. Standard gradient-based backpropagation does not scale well with multiple layers…

When we multiply the gradients many times (for each layer), it can lead to …

a) Vanishing gradient problem:
   - gradients shrink exponentially with the number of layers
   - so weight updates get smaller and smaller
   - and weights of early layers change very slowly and network learns very very slowly

b) Exploding gradient problem:
   - multiplying gradients could also make them grow exponentially.
   - so weight updates get larger and larger
   - and the weights can become so large as to overflow and result in NaN values



$$\delta_h = g'(x_h) \times Err_h$$
$$= O_h\,(1 - O_h) \times \sum_k (w_{hk}\,\delta_k)$$

$$\delta_6 = O_6\,(1 - O_6) \times \sum\ (w_{6,7}\,\delta_7)$$

$$\delta_5 = O_5\,(1 - O_5) \times \sum\ (w_{5,6}\,\delta_6)$$

$$\delta_4 = O_4\,(1 - O_4) \times \sum\ (w_{4,5}\,\delta_5)$$

$$\delta_3 = O_3\,(1 - O_3) \times \sum\ (w_{3,4}\,\delta_4)$$

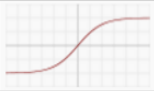
…

Initial Drawbacks (*1*)

To help, we can :

a) Use other activation functions…

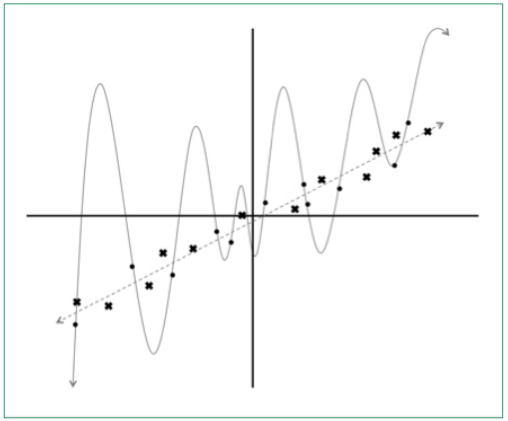
b) Do "gradient clipping" (i.e. set bounds on the gradients)

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1-f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU)[2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU)[3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1+e^x)$	$f'(x) = \frac{1}{1+e^{-x}}$

https://medium.com/towards-data-science/activation-functions-and-its-types-which-is-better-a9a5310cc8f

https://medium.com/towards-data-science/activation-functions-neural-networks-1cbd9f8d91d6

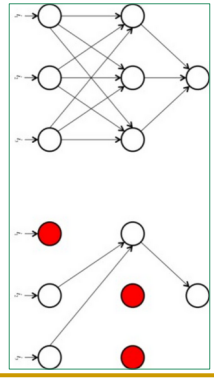
Initial Drawbacks (2)



2. Overfitting
   - Large network -> lots of parameters -> increased capacity to "learn by heart"
   - *Solutions:*
     - *Regularization:*
       - modify the error function that we minimize to penalize large weights.

$$\frac{1}{2}\sum_{i=0}^{n}\frac{(T_i - O_i)^2}{n} + \lambda f(w)$$

       - where *f(w)* grows larger as the weights grow larger and λ is the regularization strength
     - *Dropout:*
       - keep a neuron active with some probability *p* or setting it to zero otherwise.
       - prevents the network from becoming too dependent on any one neuron.



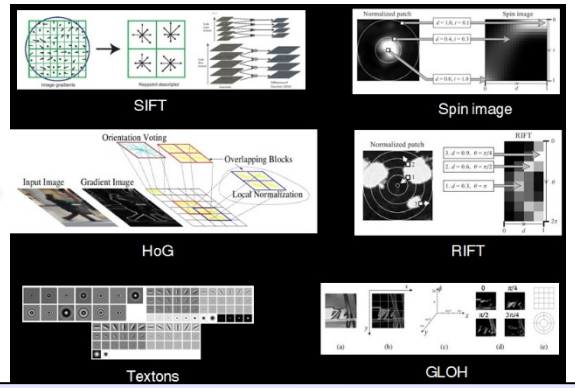
https://www.kdnuggets.com/2015/04/preventing-overfitting-neural-networks.html

Initial Drawbacks (3)

3. Multilayered ANNs need lots of labeled data
   - Solution: "pre-train" the network with features found automatically using unsupervised data
   - i.e. Automatic feature learning… (see next few slides)

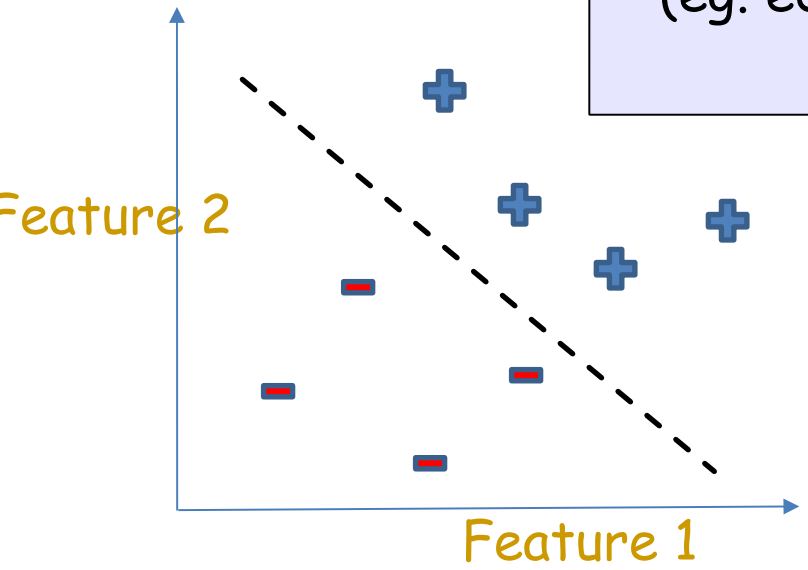
Classic ML

Input



Manual Extraction of Features (eg. edge detection, colors, texture,…)

Learning algorithm



Motorbikes
"Non"-Motorbikes

Classic ML, requires labeled data and hand-crafted features

1. Needs expert knowledge
2. Time-consuming and expensive
3. Does not generalize to other domains

Slide from Y. LeCun

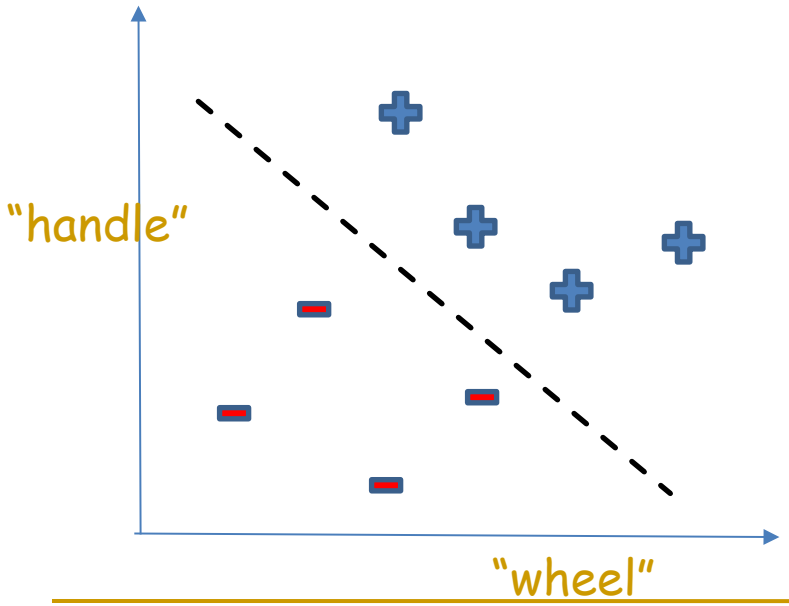
Automatic Feature Learning

Input

Automatic Feature Representation

eg. handle, wheel, …

Learning algorithm



Motorbikes

"Non"-Motorbikes

With Automatic Feature Learning:

1. We feed the network the raw data (not feature-curated)

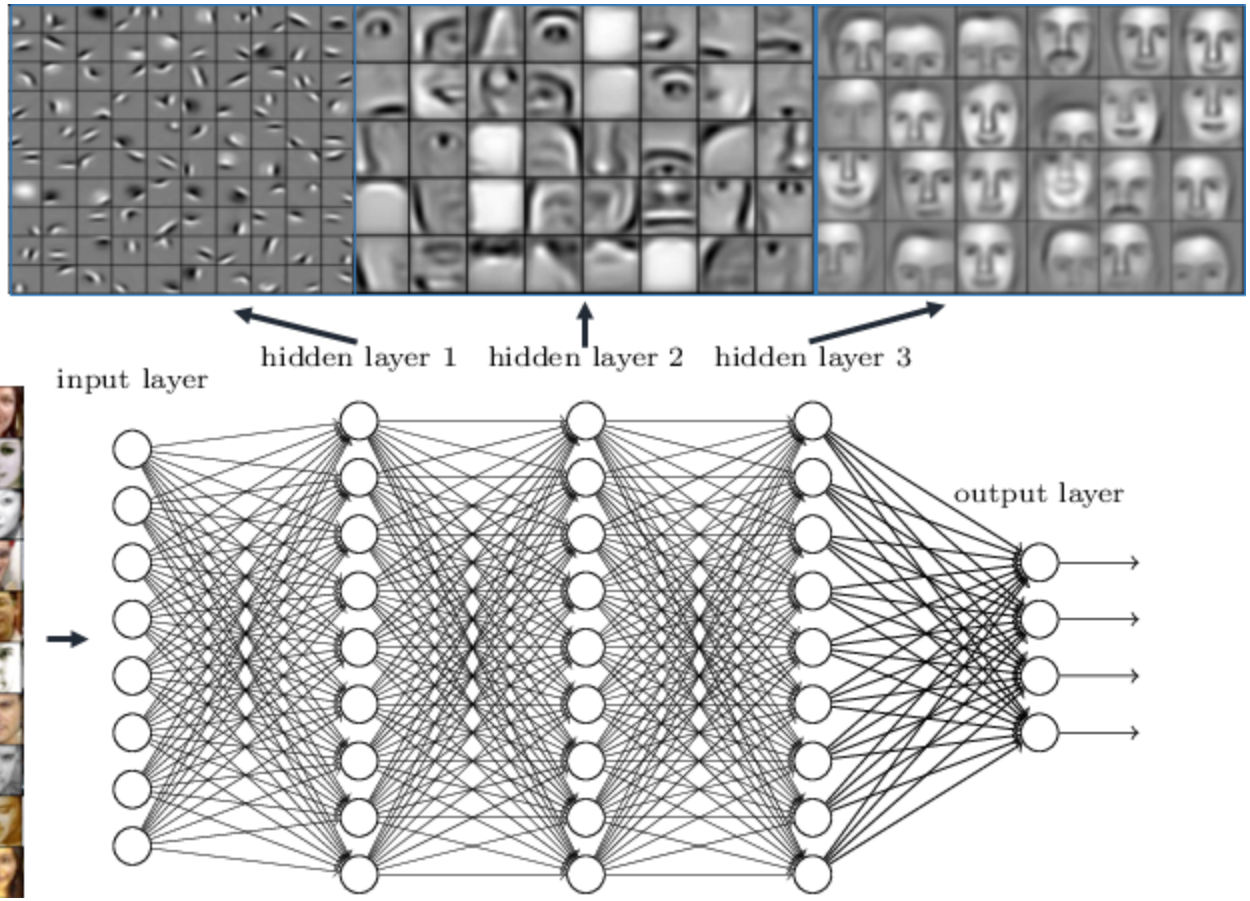
2. The features are learned by the network

3. Features learned can be re-used in similar tasks.

Slide from Y. LeCun

Automatic Feature Learning

Deep neural networks learn hierarchical feature representations



https://www.strong.io/blog-images/movie-posters/Slide6.png

Automatic Feature Learning

Deep Learning = Machine learning algorithms based on learning multiple levels of representation / abstraction.
– Y. Bengio

- Each layer learns more abstract features that are then combined / composed into higher-level features automatically
- Like the human brain …
  - has many layers of neurons which act as feature detectors
  - detecting more and more abstract features as you go up
- E.g. to classify an image of a cat:
  - Bottom Layers: Edge detectors, curves, corners straight lines
  - Middle Layers: Fur patterns, eyes, ears
  - Higher Layers: Body, head, legs
  - Top Layer: Cat or Dog

Automatic Feature Learning

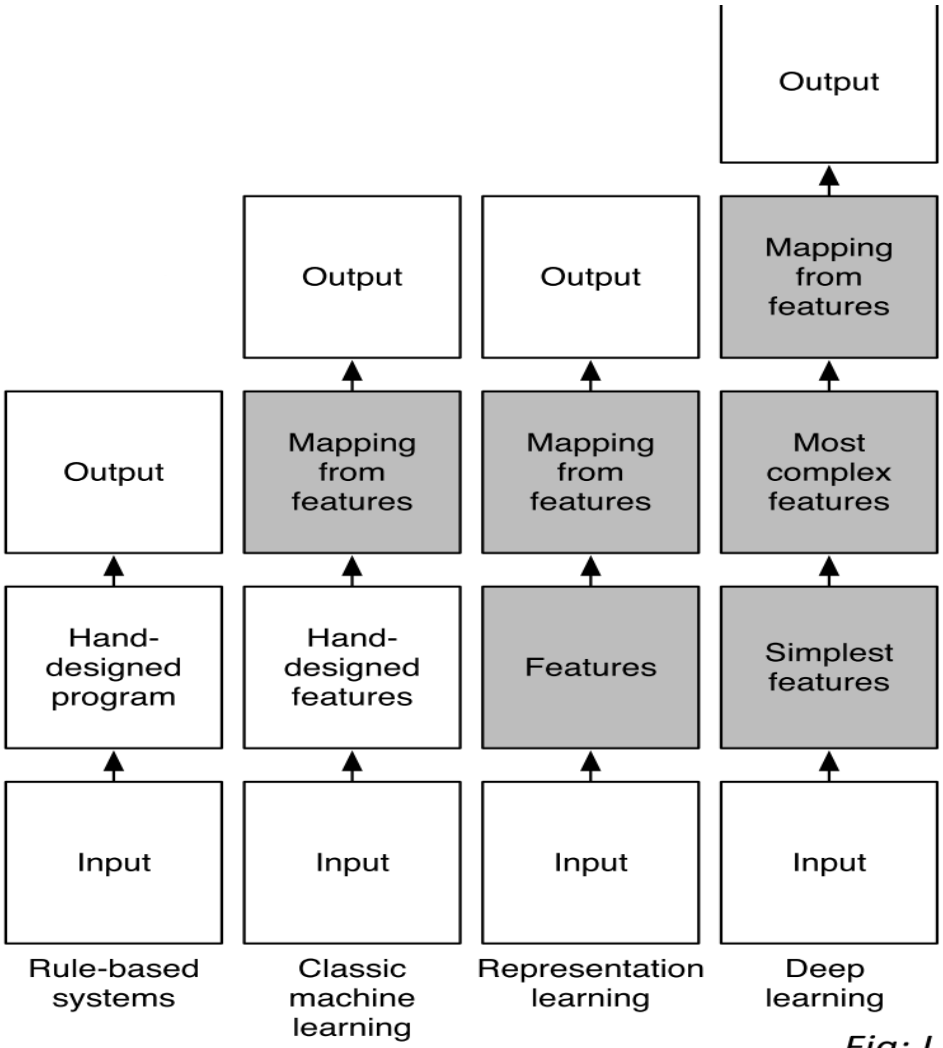
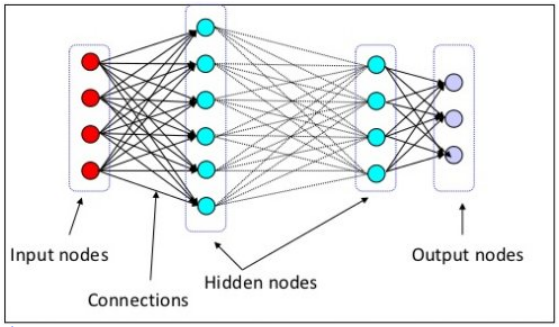
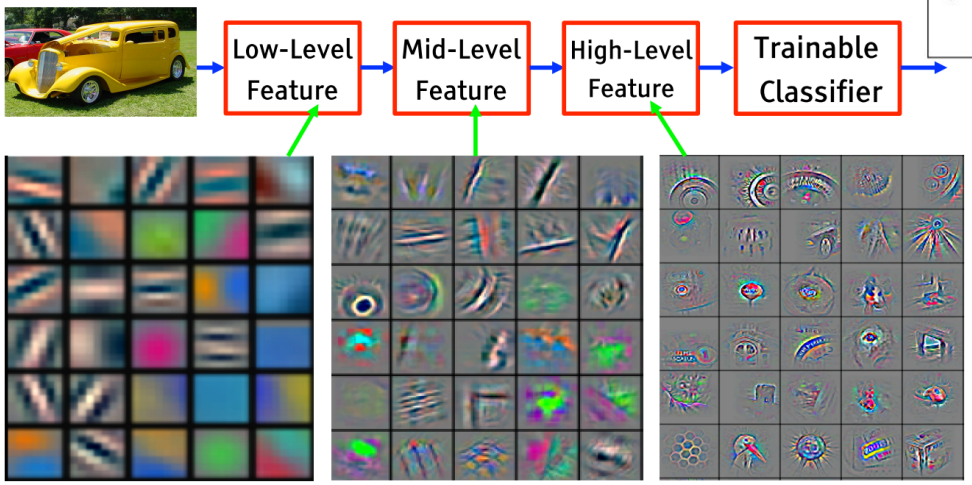


Fig: I. Goodfellow

What Types of Features?



- For image recognition
  - pixel -> edge -> texton -> motif -> part -> object

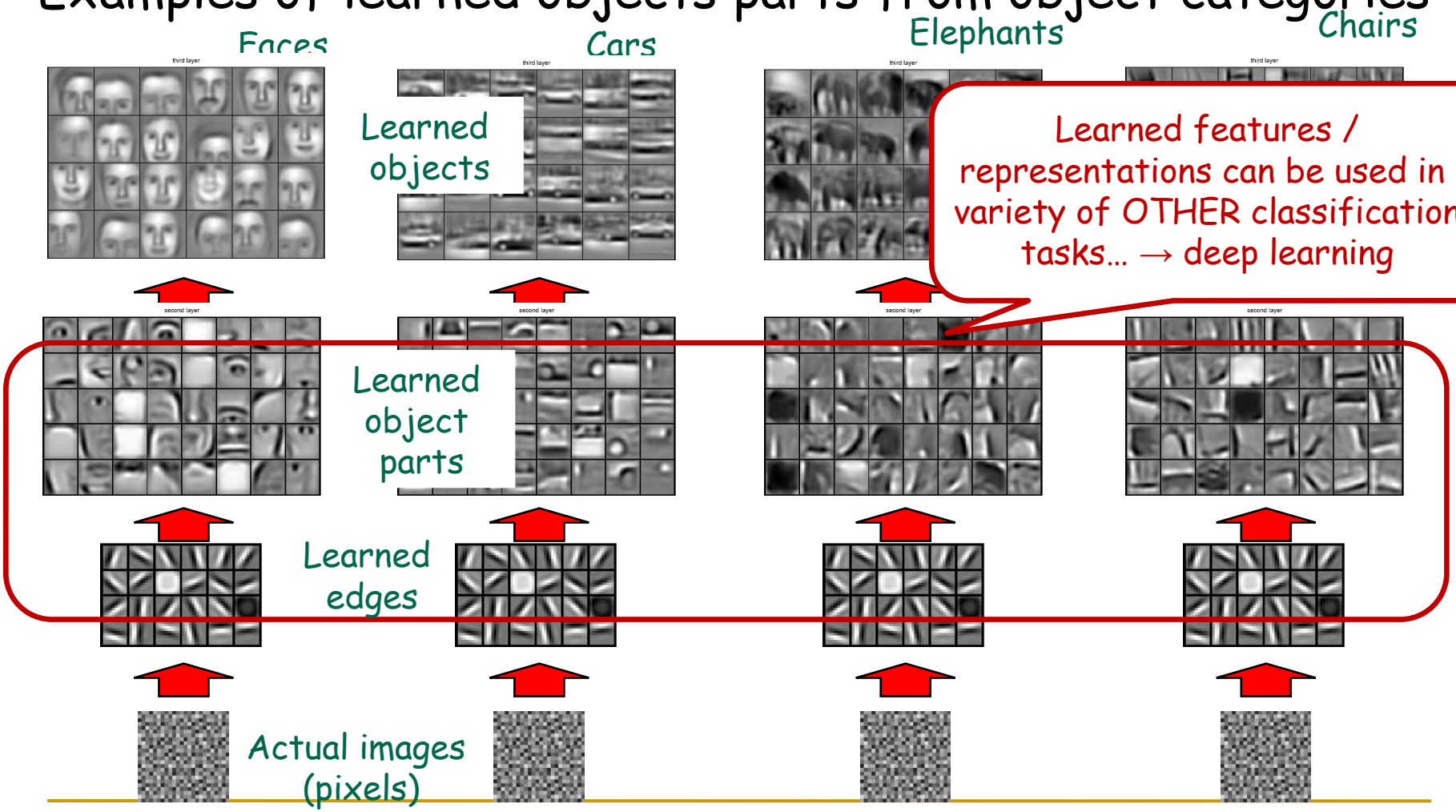


- For NLP
  - character -> word -> constituents -> clause -> sentence -> discourse
- For speech:
  - sample → spectral band -> sound -> … phone -> phoneme -> word

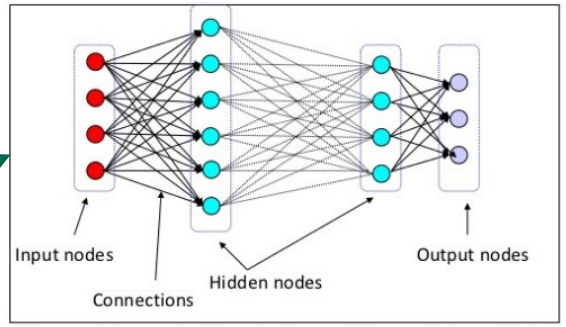
Figure from Y LeCun

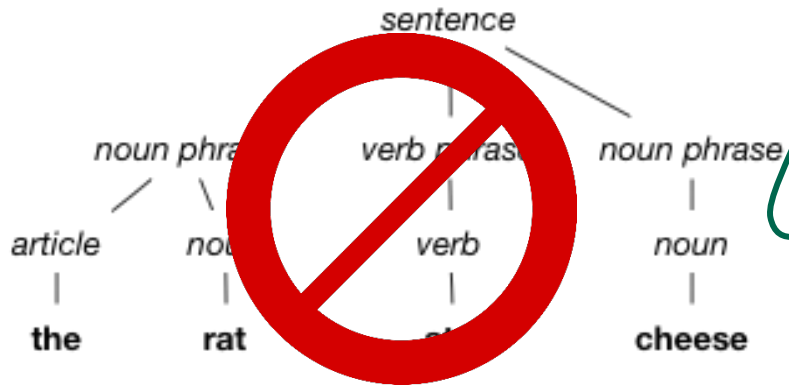
Eg: Learning Image Features

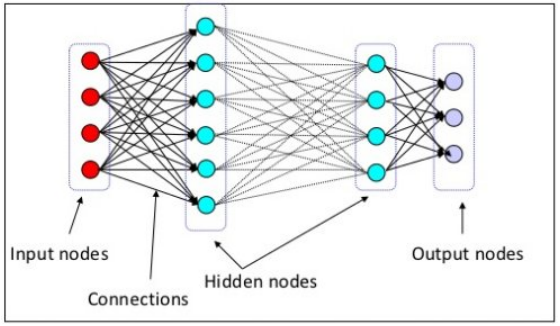
Examples of learned objects parts from object categories



Advantages of Unsupervised Feature Learning







Advantages of Unsupervised Feature Learning

- Much more unlabeled data available than labeled data:
  - Eg. Websites, Books, Videos, Pictures
- Humans learn initially from unlabeled examples
  - Eg. Babies learn to talk/recognize objects without labeled data
- As the features are learned in an unsupervised way from a different and larger dataset, less risk of over-fitting
- **No need for manual feature engineering**
- These features are organized into multiple levels
  - Each level creates new features from combinations of features from the level below
  - Each level is more abstract than the ones below (hierarchy of features)

Today

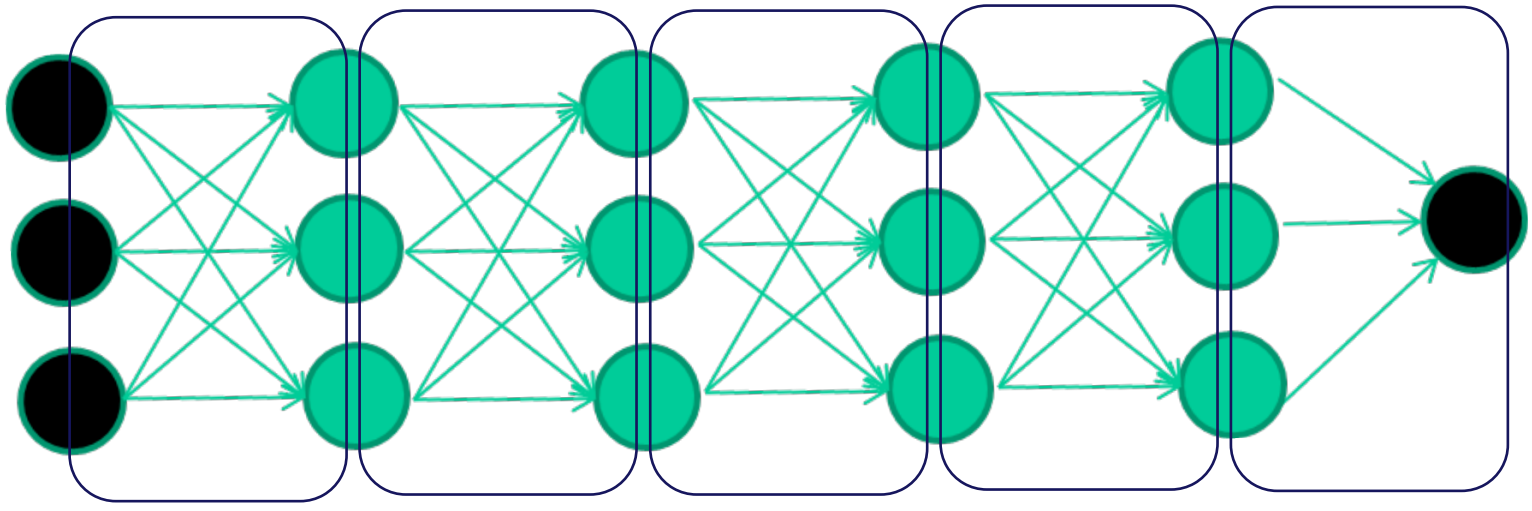
1. Motivation
2. Feature Learning
3. Early Training of Deep Neural Networks
4. CNNs for Image Processing
5. Deep Learning for NLP
6. Conclusion

General Architecture of a Deep Network

1. Unsupervised pre-training of neural network using unlabeled data
   - aka. unsupervised learning of features

2. Supervised training with labeled data using features learned from above with a standard classifier
   - Eg. an ANN, SVM, …

Training a Deep NN



EACH of the (non-output) layers is trained to learn a representation of the data (eg. autoencoder) aka "pretraining the network with unsupervised data"

Use pretrained representations to feed a regular ANN with a smaller set of labelled data.

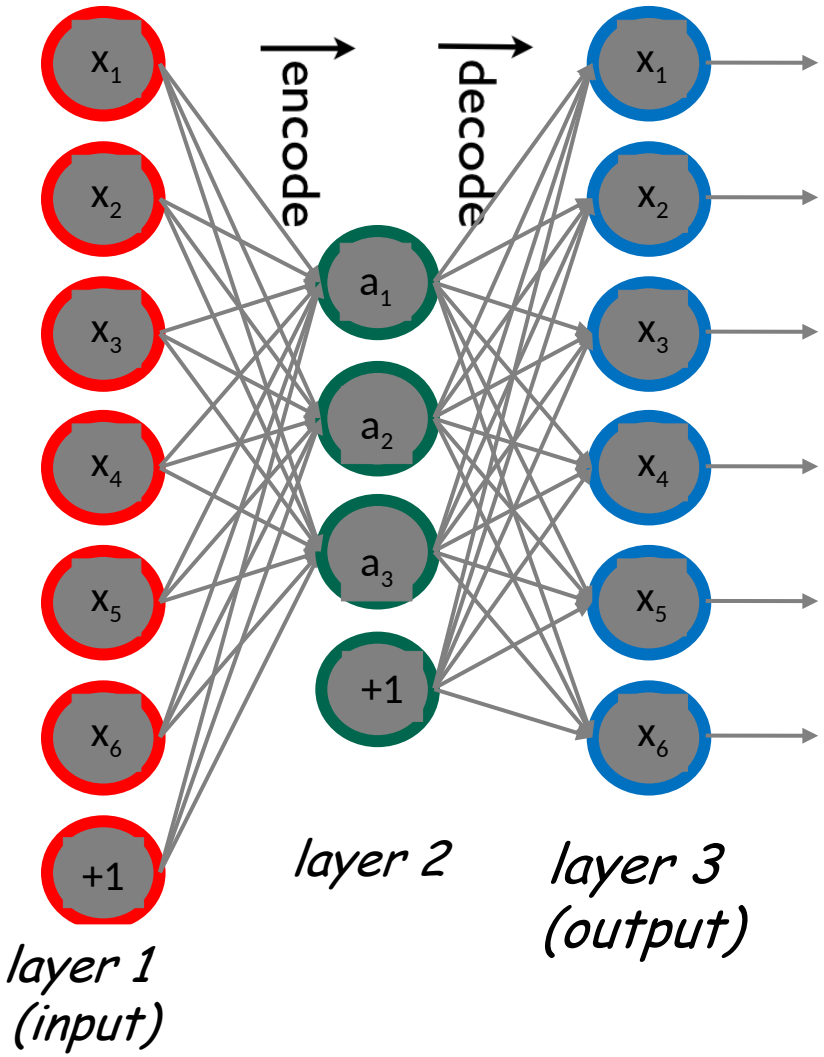
Autoencoders

- To learn a representation of the data, we can use:
  1. Deep Belief Networks
     - Mid 2000's: Geoffrey Hinton trains a deep network by:
       - Stacking Restricted Boltzmann Machines (RBM's) on top of one another – deep belief network
       - Training layer by layer on un-labeled data
       - Then, using back propagation to fine tune weights on labeled data

  2. Autoencoders
     - 2006: Yoshua Bengio et al. does something similar using auto-encoders instead of RBM's

- Both are 2 layer neural networks that learn to model their inputs

Autoencoder

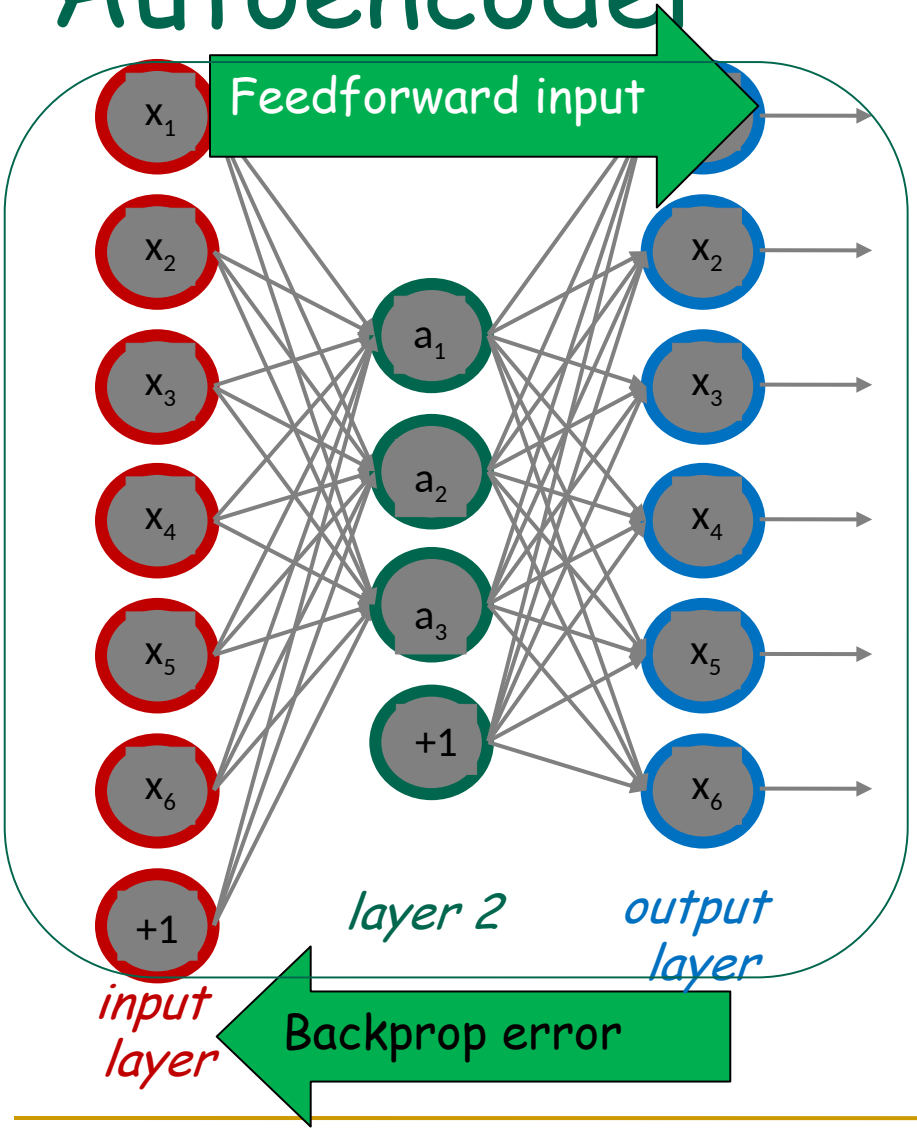


- The network is trained to output the input
- i.e. learn the identity function.

- Trivial… unless, we impose constraints:
  - Nb of units in layer 2 < nb of input units (learn compressed representation)
    OR
  - Constrain layer 2 to be sparse (i.e. many connections are "disabled")

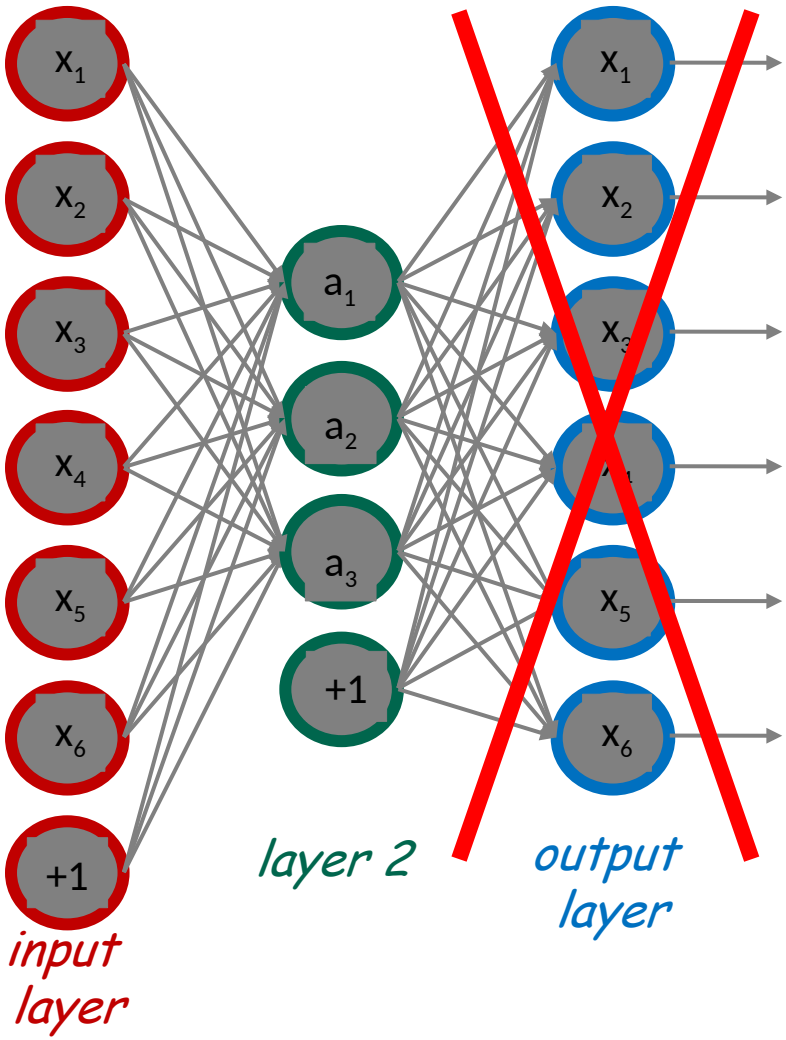
→ **Worksheet #5 ("Autoencoder")**

Autoencoder

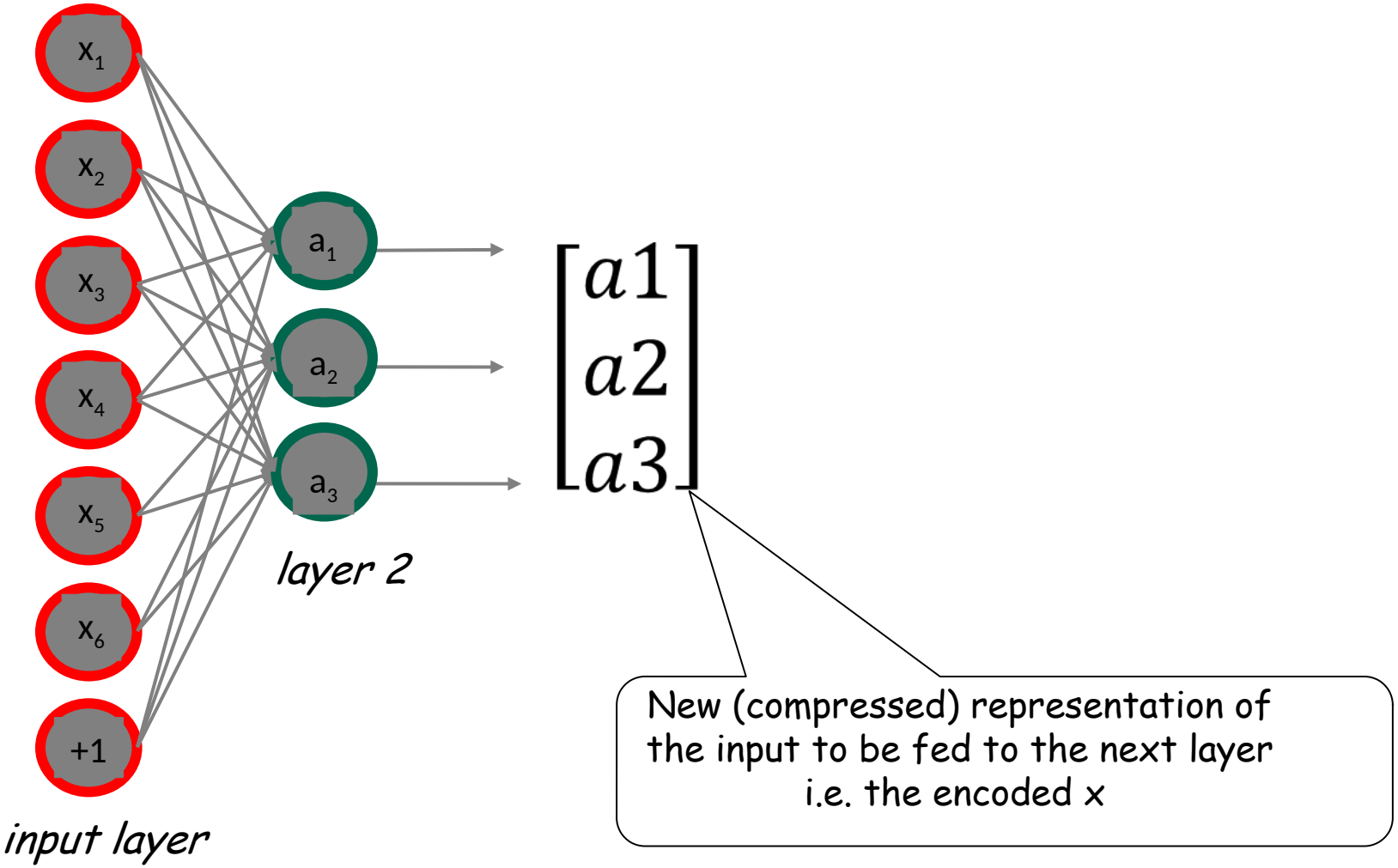


→ **Worksheet #5 ("Autoencoder Activation")**

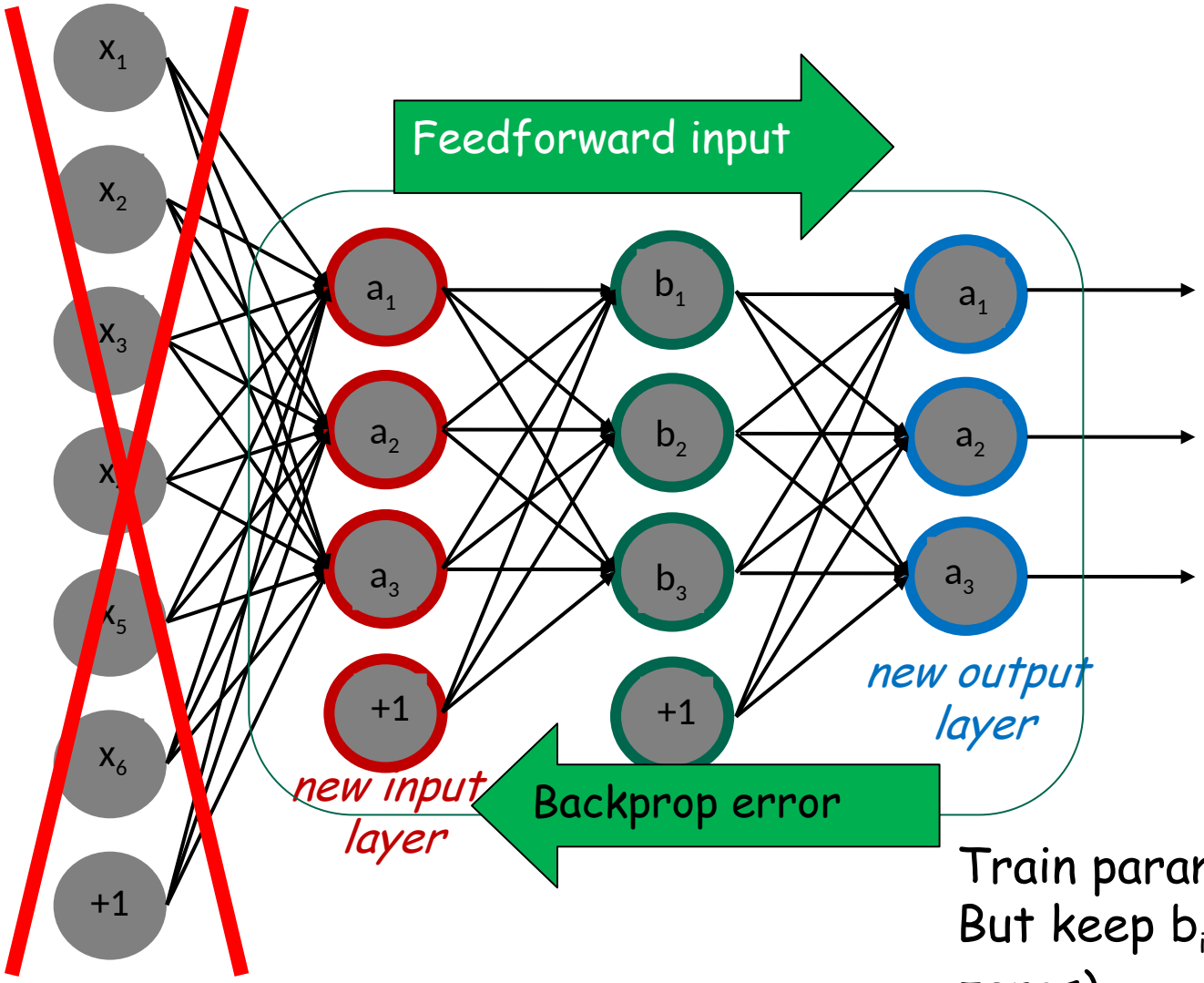
Autoencoder



Autoencoder

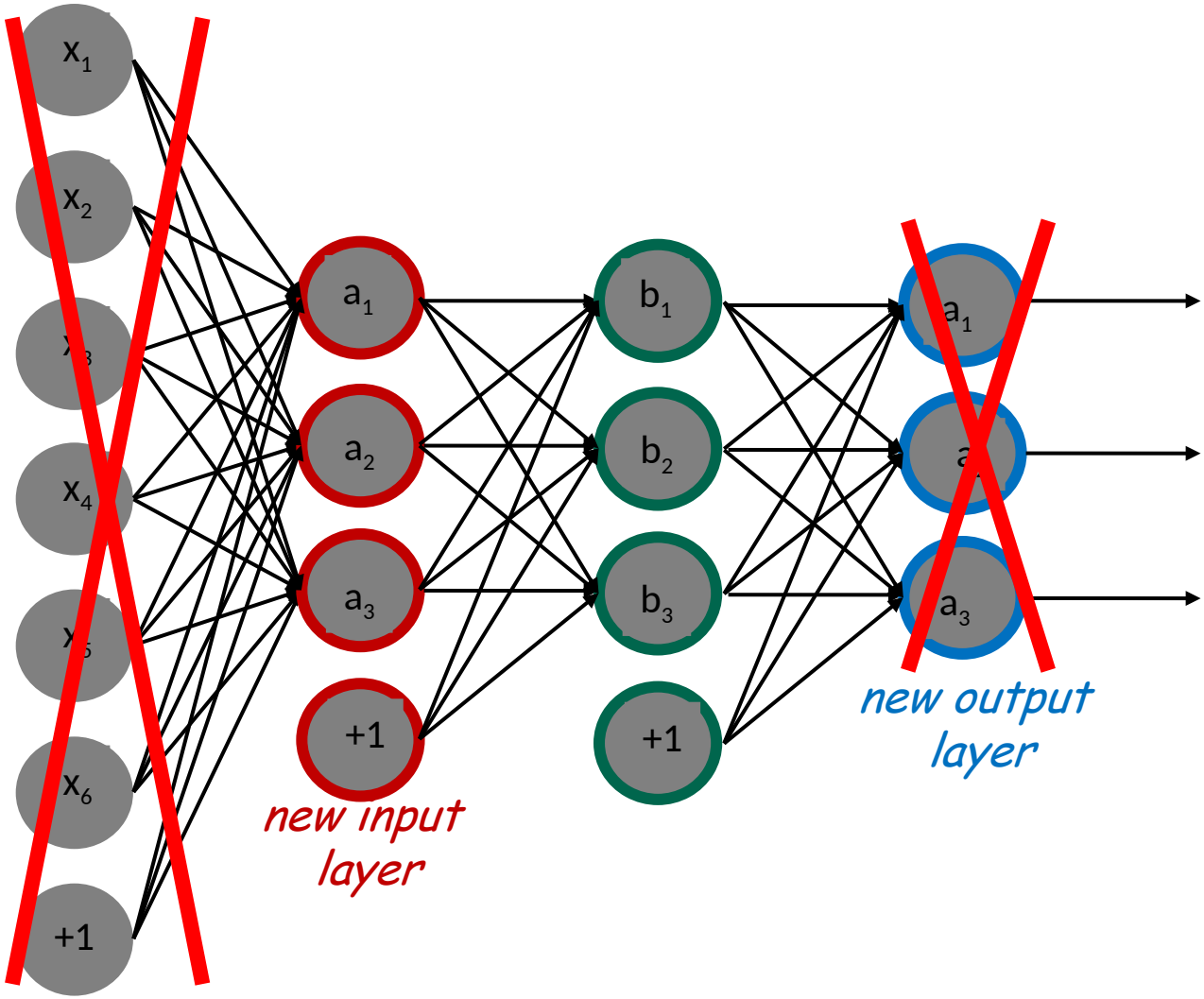


Autoencoder

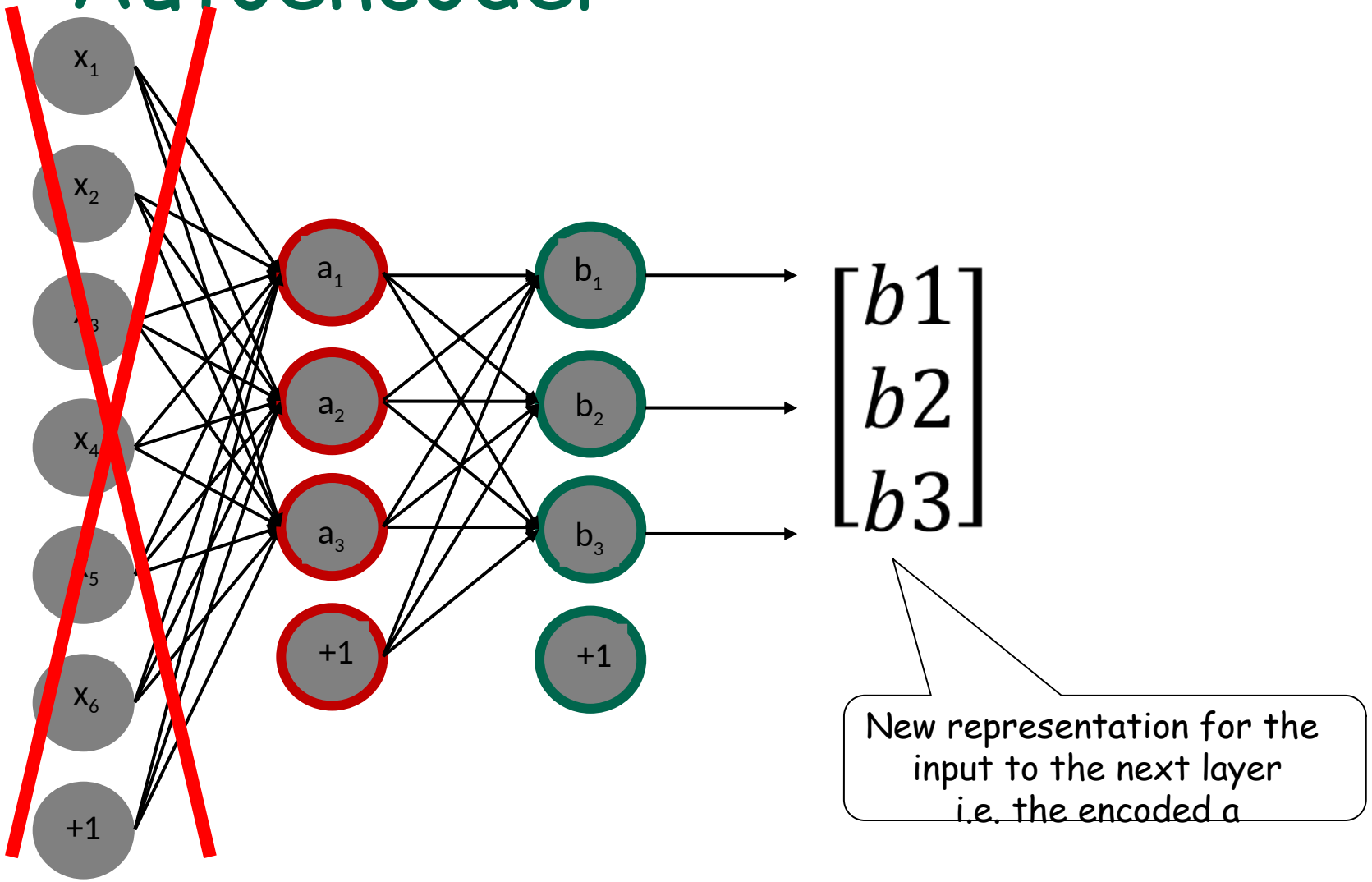


Train parameters (weights)
But keep b_i's sparse (ie. many zeros).

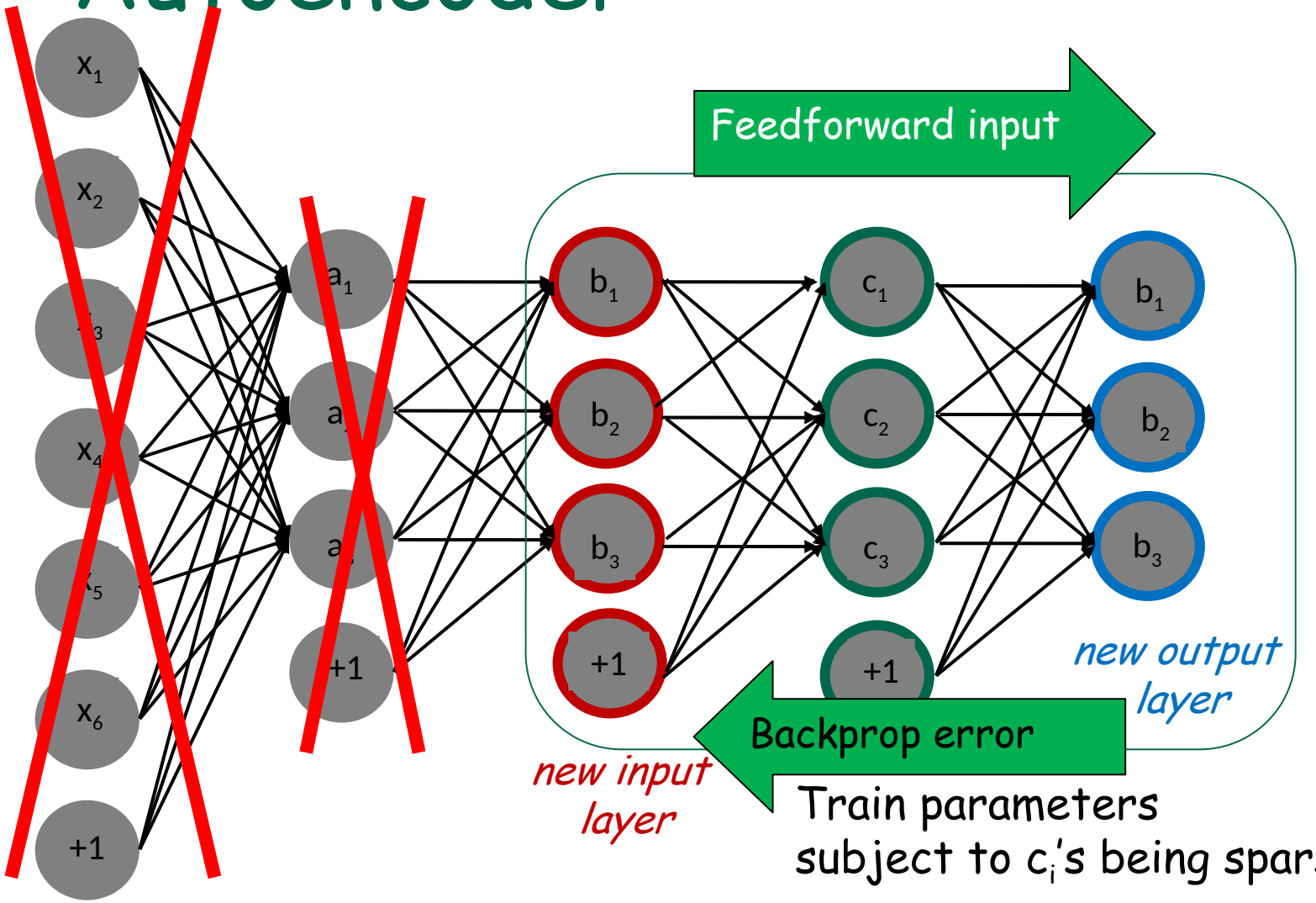
Autoencoder



Autoencoder

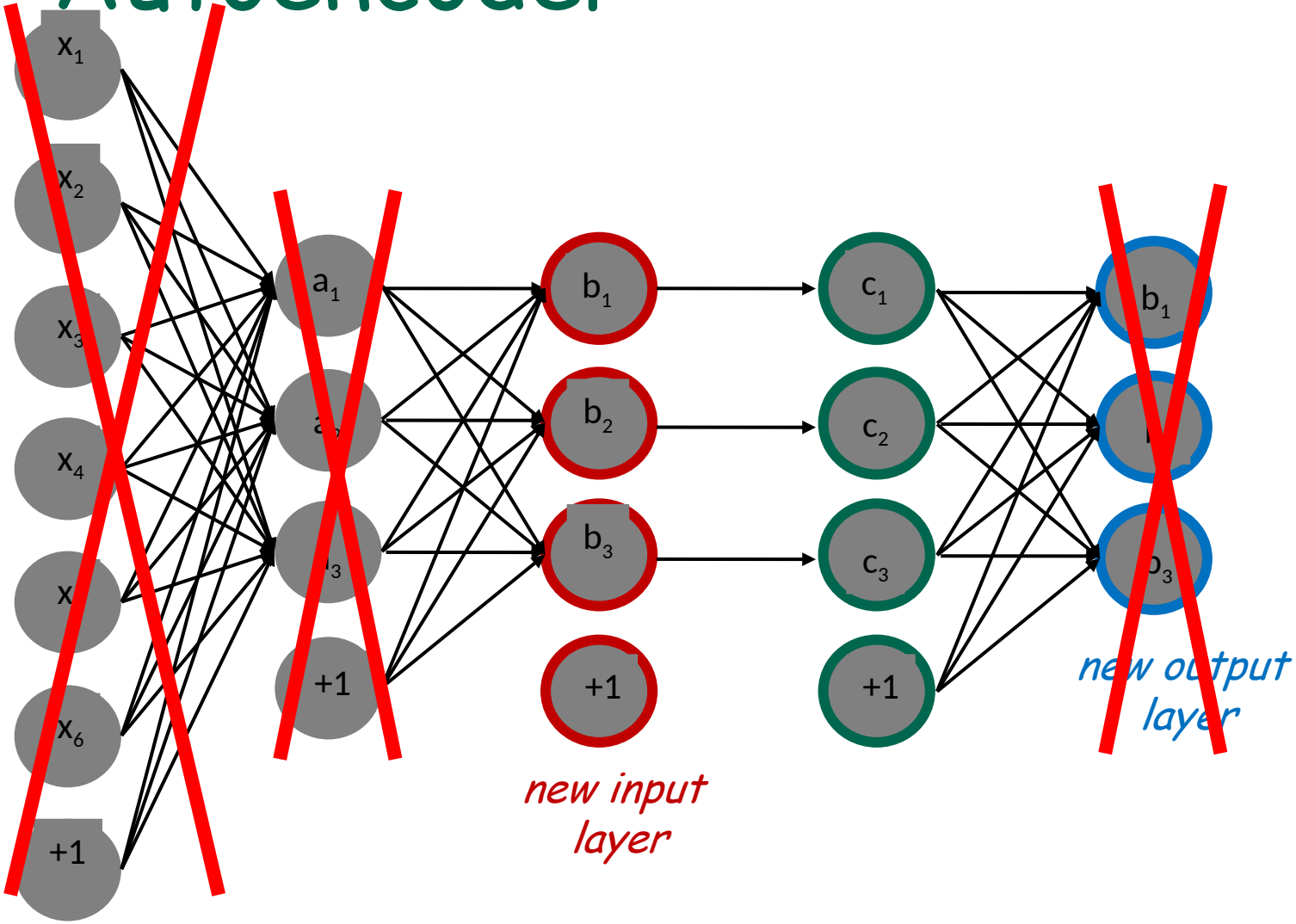


Autoencoder

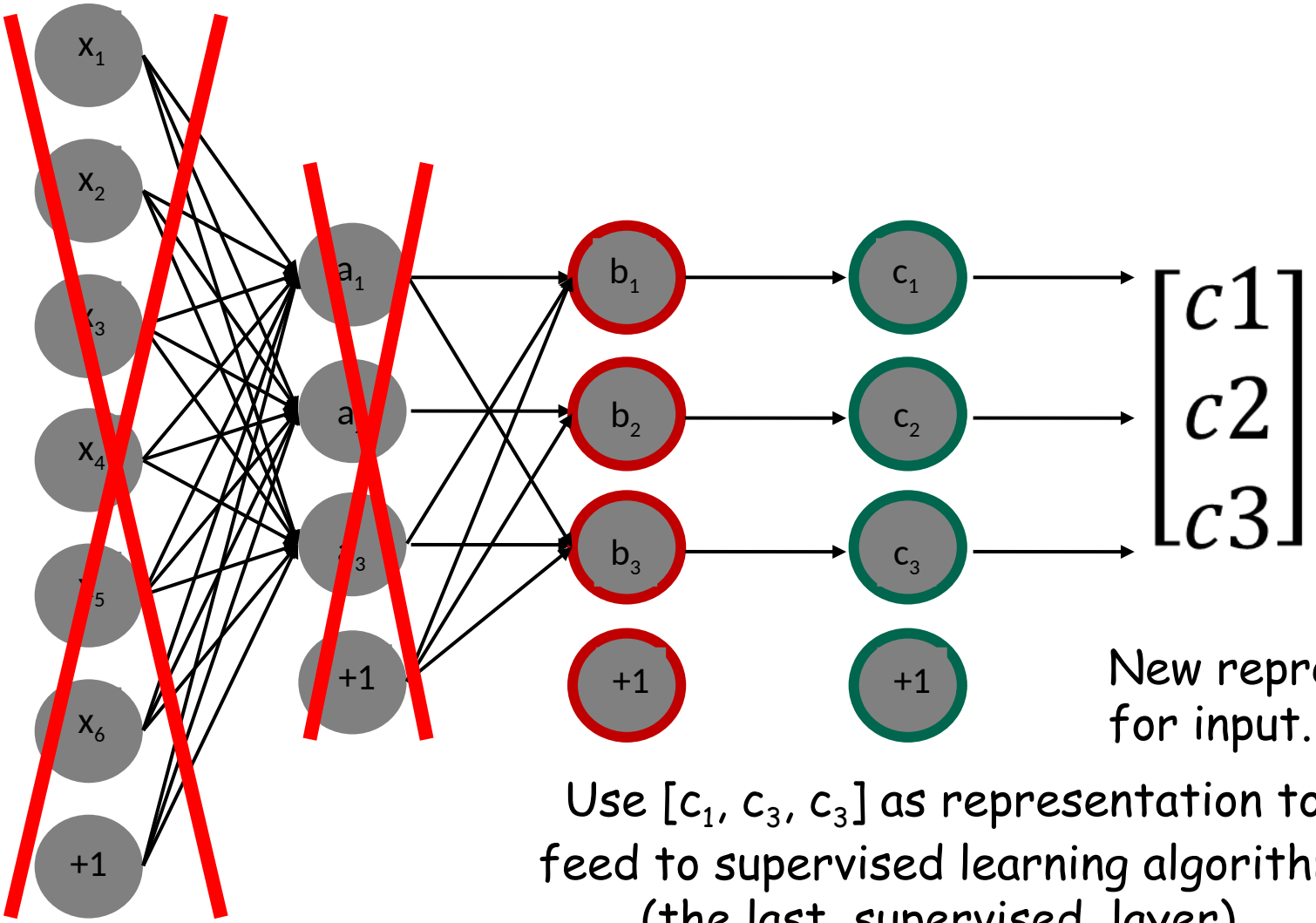


Train parameters subject to c_i's being sparse.

Autoencoder



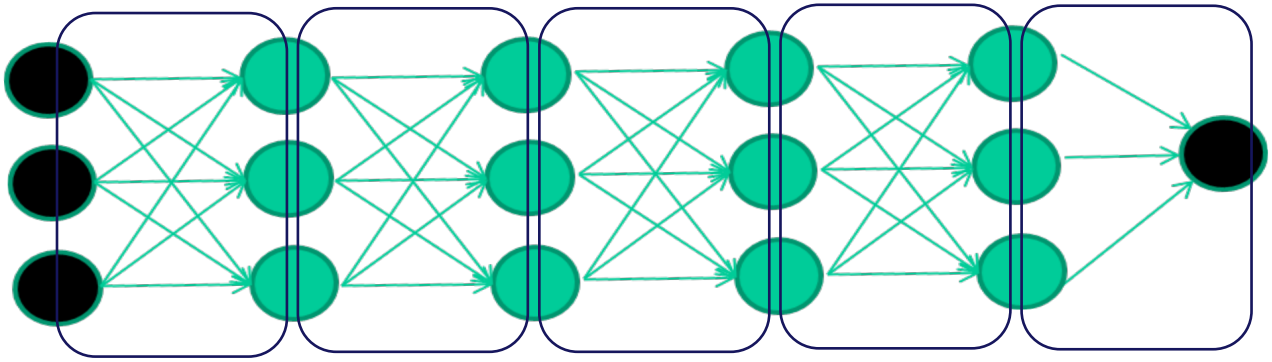
Autoencoder



New representation for input.

Use $[c_1, c_3, c_3]$ as representation to feed to supervised learning algorithm (the last, supervised, layer).

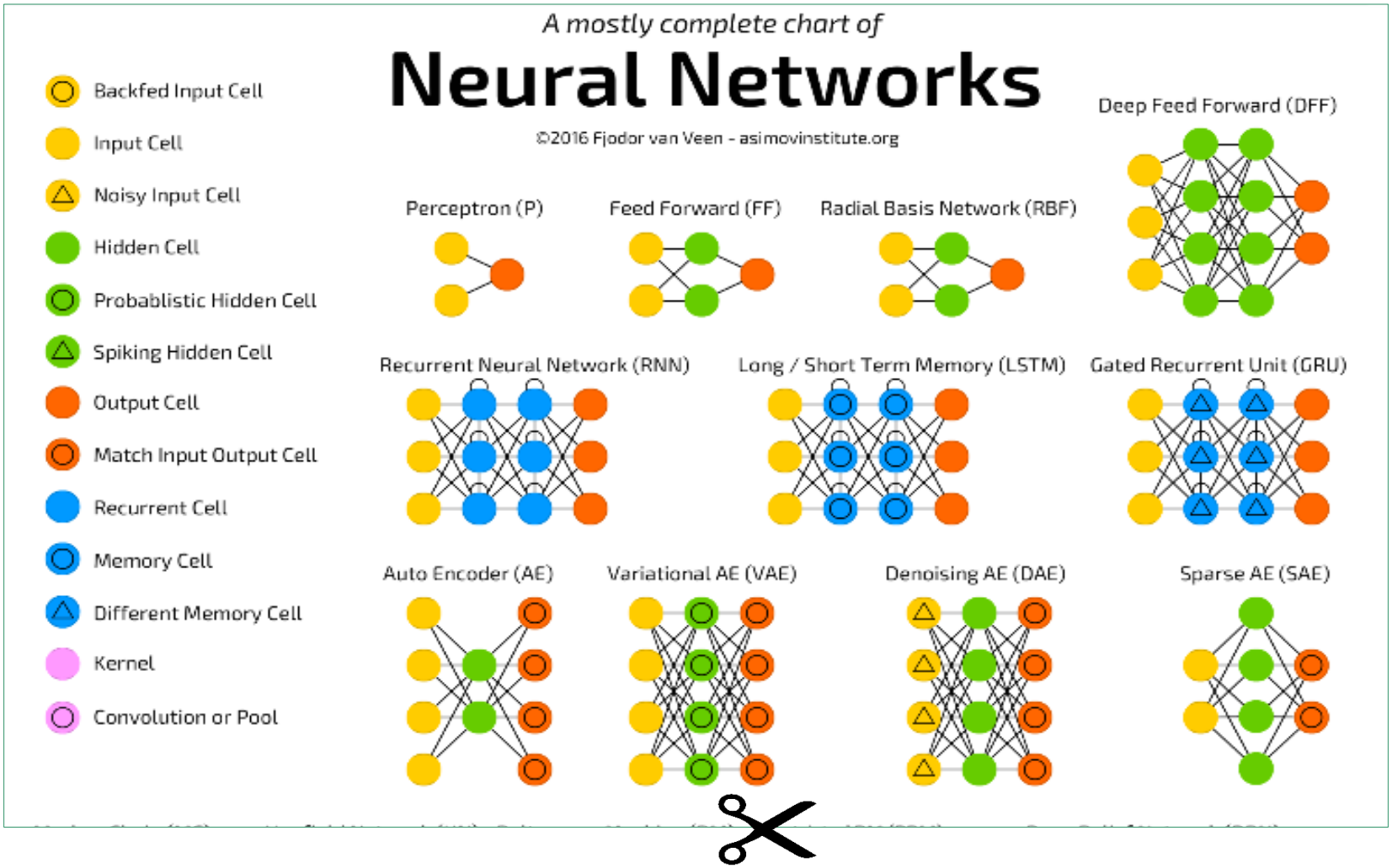
Training a Deep NN



Use of unlabelled data to "pretrain" the network i.e. learn more and more abstract feature representations

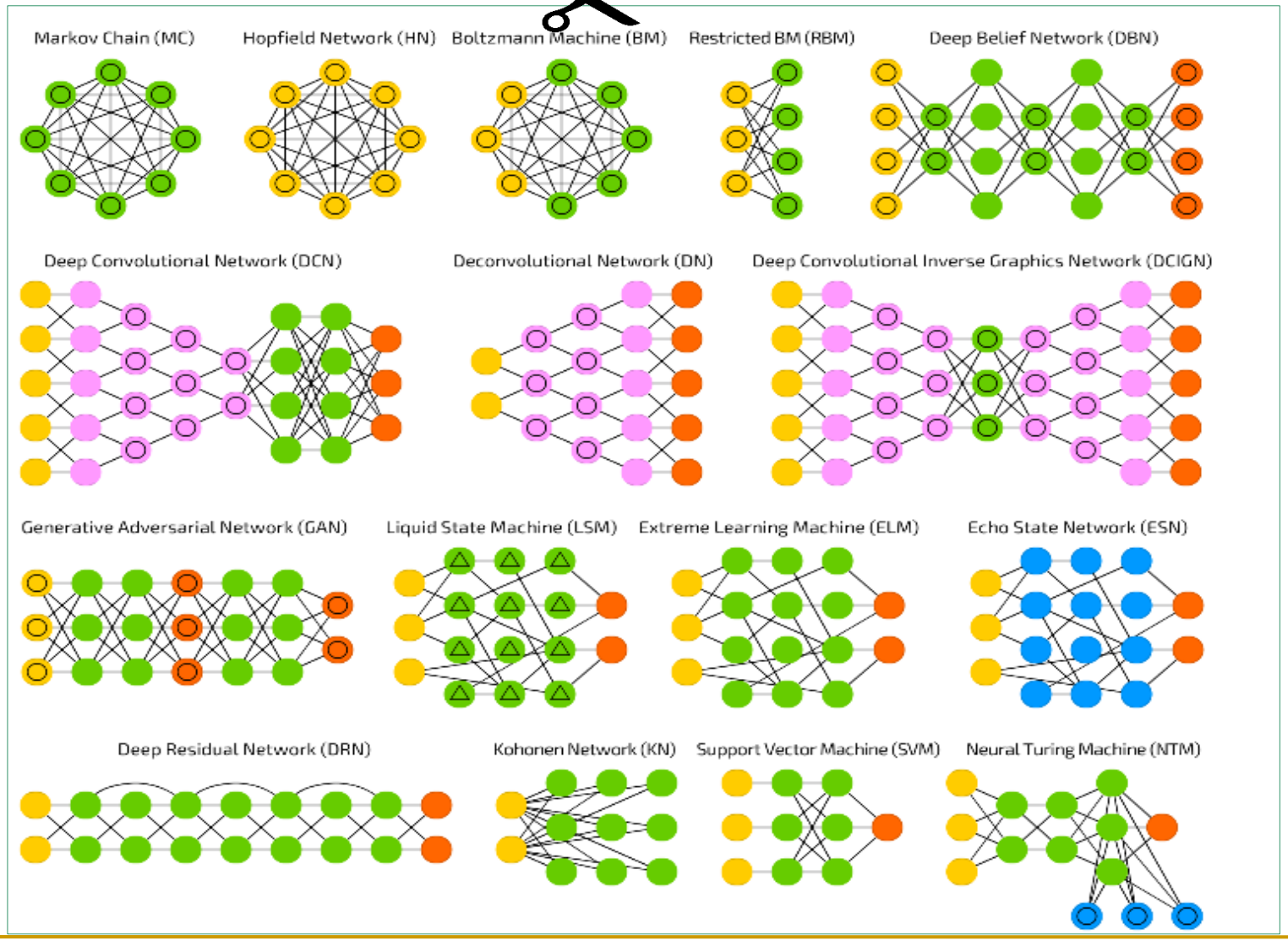
Use pretrained representations to feed a regular ANN with a smaller set of labelled data.

Many Types of Neural Networks



https://becominghuman.ai/cheat-sheets-for-ai-neural-networks-machine-learning-deep-learning-big-data-678c51b4b463

Many Types of Deep Networks (con't)



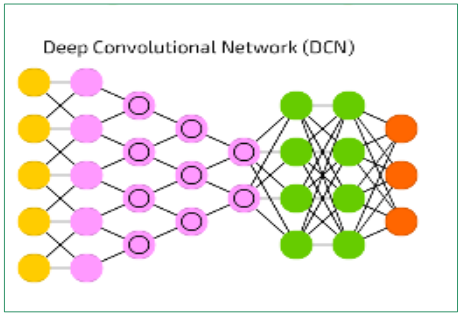
https://becominghuman.ai/cheat-sheets-for-ai-neural-networks-machine-learning-deep-learning-big-data-678c51b4b463

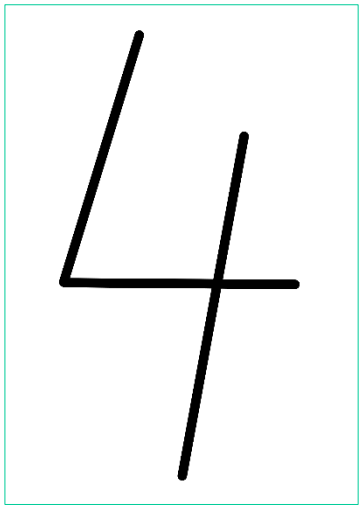
Today

1. Motivation
2. Feature Learning
3. Early Training of Deep Neural Networks
4. CNNs for Image Processing
5. Conclusion

CNNs for Image Processing

CNNs = Convolutional Neural Networks





25	2	1	44
223	7	6	60
196	8	2	148
249	1	3	40
60	7	1	154
59	1	7	213
214	7	3	163
89	182	219	13
74	146	113	72
89	18	244	85
1	4	8	97
3	4	2	121
2	1	2	131
7	6	8	47
3	5	5	126
7	6	8	121
5	3	1	237

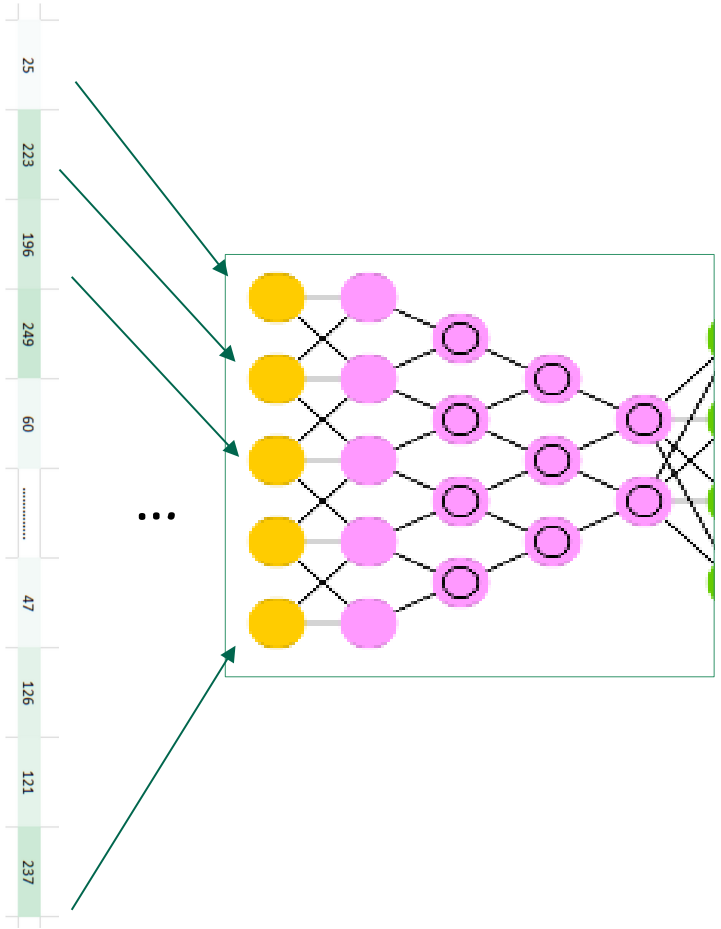
Image of a 4 in grey scale
Value = 0-> white …. 255->black

https://www.analyticsvidhya.com/blog/2017/06/architecture-of-convolutional-neural-networks-simplified-demystified/

CNNs for Image Processing

- Standard input of the image in the ANN:
  1. 1 pixel = 1 input.
  2. Eg. color image of 200x200x 3channels (RGB)
     --> in a fully connected ANN, a neuron of the input layer has 200*200*3 = 120,000 weights
     --> huge number of parameters, can easily overfit
  2. We linearize the image ==> We lose spatial information

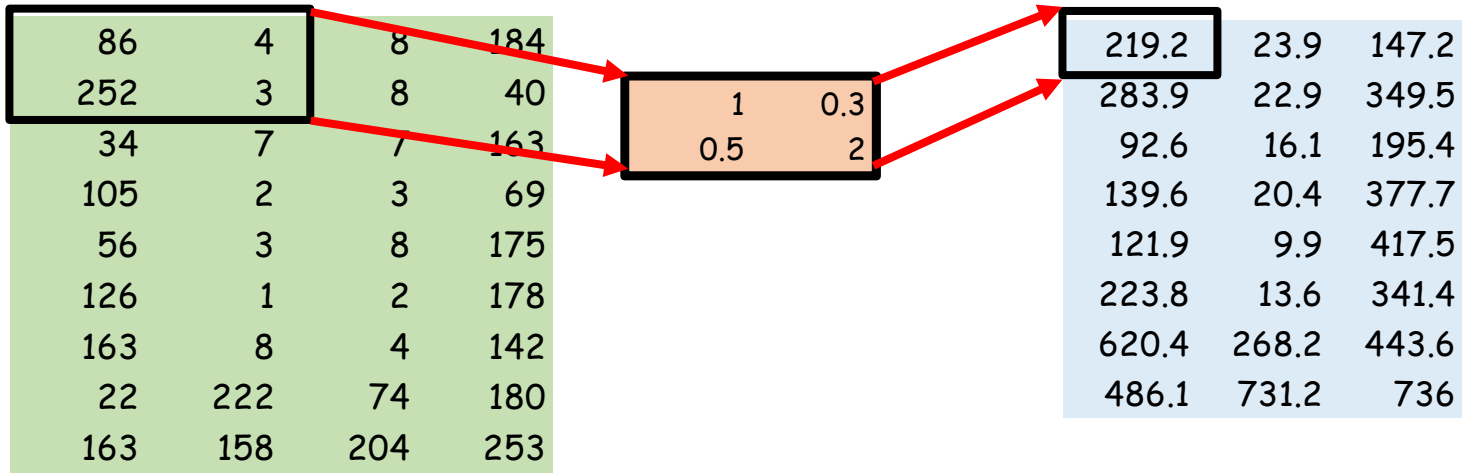
25	2	1	44
223	7	6	60
196	8	2	148
249	1	3	40
60	7	1	154
59	1	7	213
214	7	3	163
89	182	219	13
74	146	113	72
89	18	244	85
1	4	8	97
3	4	2	121
2	1	2	131
7	6	8	47
3	5	5	126
7	6	8	121
5	3	1	237



https://www.analyticsvidhya.com/blog/2017/06/architecture-of-convolutional-neural-networks-simplified-demystified/

Convolutional Layer

- Use a filter (aka kernel) that "convolves" on the image
- Filter = small weight matrix to learn

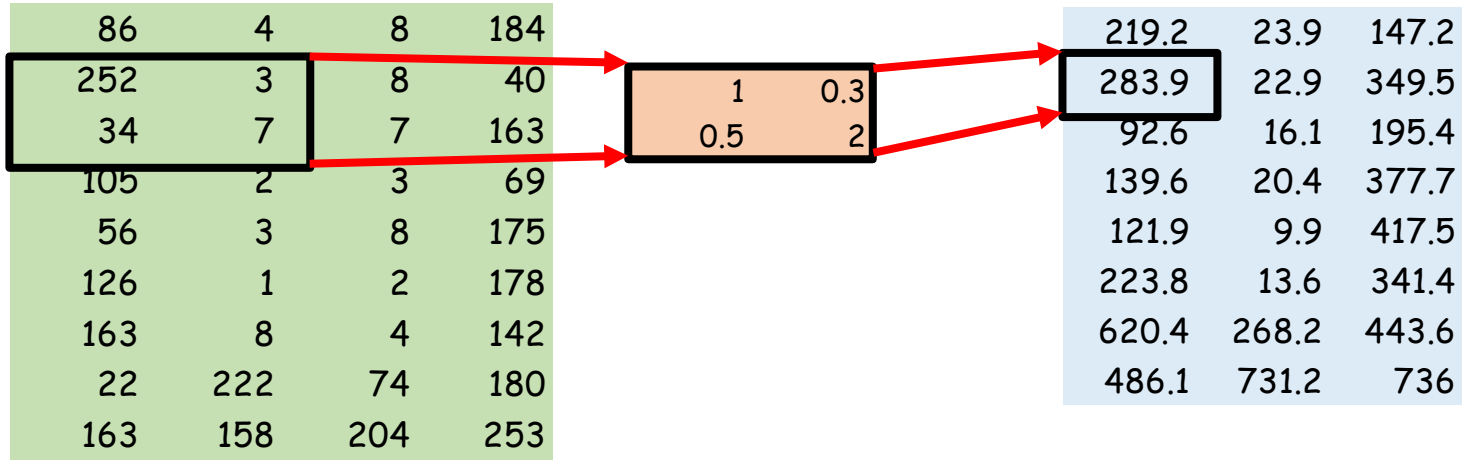


I(original image) W (filter) C (activation map)

$$C_{11} = (I_{11} \times W_{11}) + (I_{12} \times W_{12}) + (I_{21} \times W_{21}) + (I_{22} \times W_{22})$$
$$= 86 \times 1 + 4 \times 0.3 + 252 \times 0.5 + 3 \times 2 = 219.2$$

Convolutional Layer

- Use a filter (aka kernel) that "convolves" on the image
- Filter = small weight matrix to learn

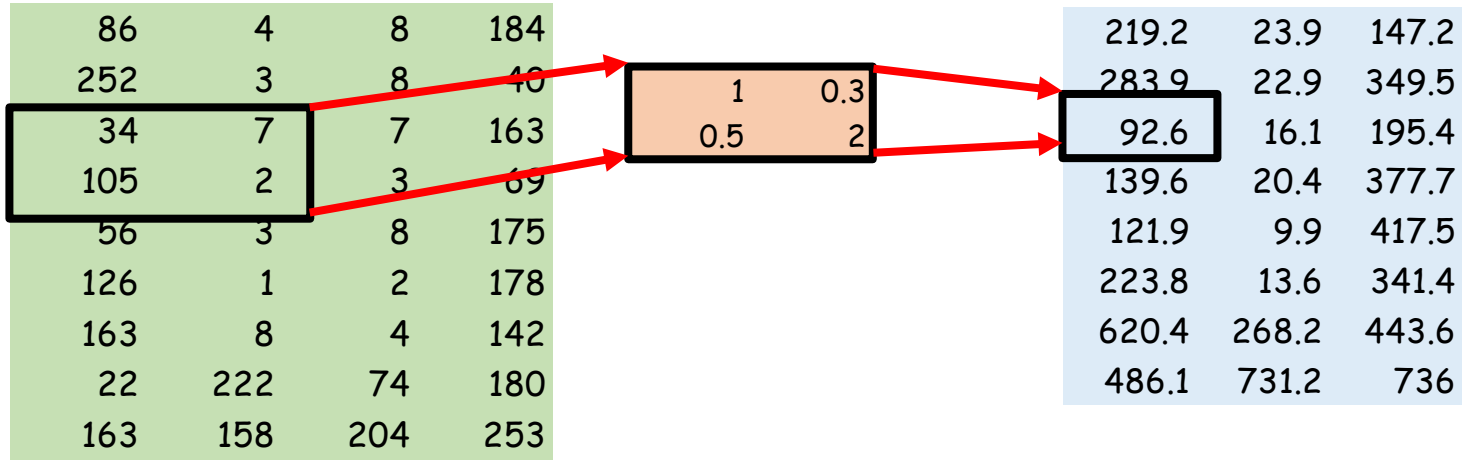


I(original image) W (filter) C (activation map)

$$C_{12} = (I_{12} \times W_{11}) + (I_{13} \times W_{12}) + (I_{22} \times W_{21}) + (I_{23} \times W_{22})$$
$$= 252 \times 1 + 3 \times 0.3 + 34 \times 0.5 + 7 \times 2 = 283.9$$

Convolutional Layer

- Use a filter (aka kernel) that "convolves" on the image
- Filter = small weight matrix to learn

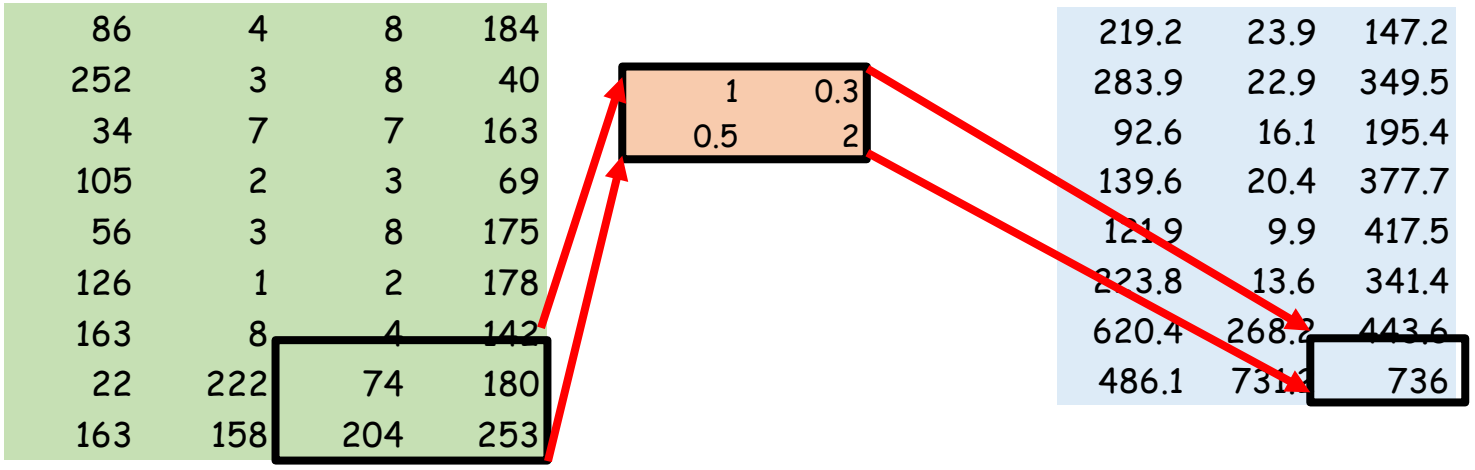


I (original image) W (filter) C (activation map)

$$C_{13} = (I_{13} \times W_{11}) + (I_{14} \times W_{12}) + (I_{23} \times W_{21}) + (I_{24} \times W_{22})$$
$$= 34 \times 1 + 7 \times 0.3 + 105 \times 0.5 + 2 \times 2 = 92.6$$

Convolutional Layer

- Use a filter (aka kernel) that "convolves" on the image
- Filter = small weight matrix to learn



I (original image) W (filter) C (activation map)

$$C_{38} = (I_{37} \times W_{11}) + (I_{47} \times W_{12}) + (I_{38} \times W_{21}) + (I_{37} \times W_{22})$$
$$= 74 \times 1 + 180 \times 0.3 + 204 \times 0.5 + 253 \times 2 = 736$$

Learn the Filters

18	54	51	239	244	188
55	121	75	78	95	88
35	24	204	113	109	221
3	154	104	235	25	130
15	253	225	159	78	233
68	85	180	214	245	0

$I\ (6 \times 6)$

1	0	1
0	1	0
1	0	1

$W\ (3 \times 3)$

429	505	686	856
261	792	412	640
633	653	851	751
608	913	713	657

$C\ (4 \times 4)$

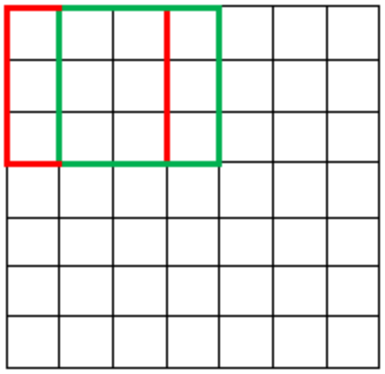
- The weight matrix (filter/kernel) behaves like a filter
- The network learns the values of the filter(s) that activate when they "see" some visual feature that is useful to identify the object (the final classification)
  - Ex. a horizontal line, a blotch of some color, a circle…

https://www.analyticsvidhya.com/blog/2017/06/architecture-of-convolutional-neural-networks-simplified-demystified/

Convolution Hyper-parameters

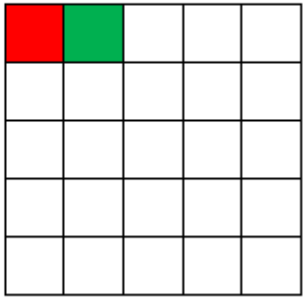
1. Stride
2. Padding

Stride

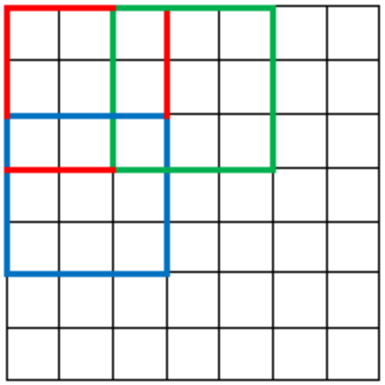


(7×7)

W (3×3) with stride =1

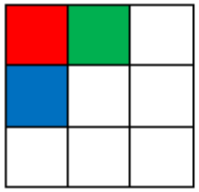


C (5×5)



(7×7)

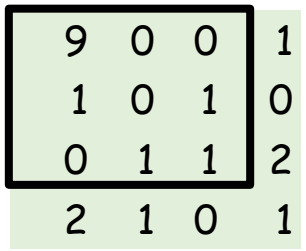
W (3×3) with stride =2

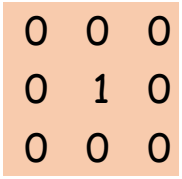


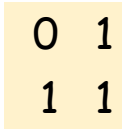
C (3×3)

→ **Worksheet #5 ("CNN Activation Map")**

Padding

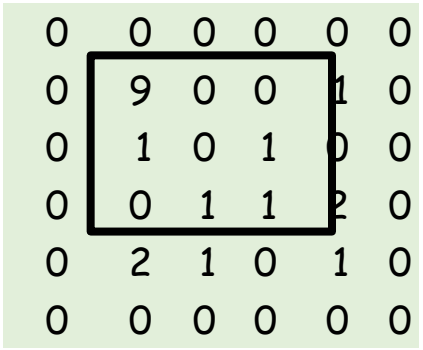


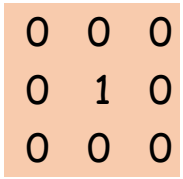


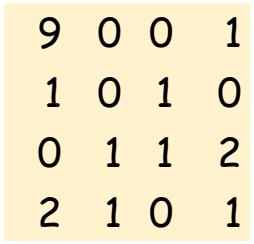


9 not picked up ;-(

filter should pick up high values surrounded by low values



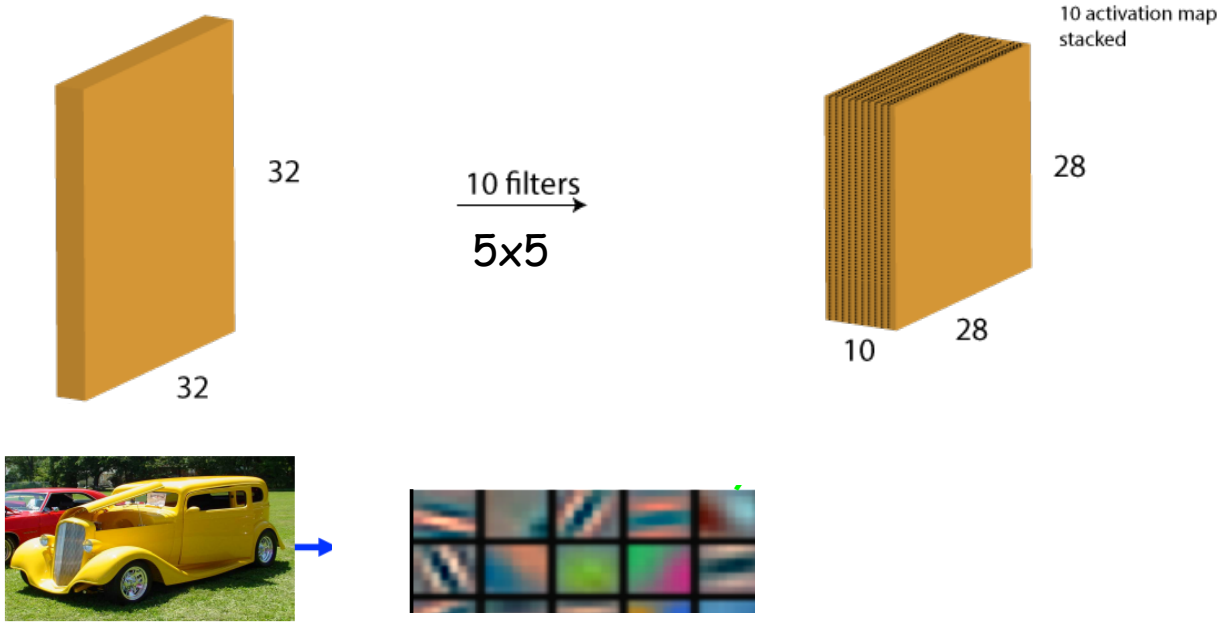




9 picked up ;-)

Learn Several Filters

- So we create 1 activation map per filter



https://www.analyticsvidhya.com/blog/2017/06/architecture-of-convolutional-neural-networks-simplified-demystified/

Pooling Layer

- Used to:
  - To reduce the size of the activation maps
  - So that we reduce the number of parameters of the network and hence avoid overfitting.
- Several strategies:
  - Max pooling

429	505	686	856
261	792	412	640
633	653	851	751
608	913	713	657

→

792	856
913	851

  - Average pooling
  - …

429	505	686	856
261	792	412	640
633	653	851	751
608	913	713	657

→

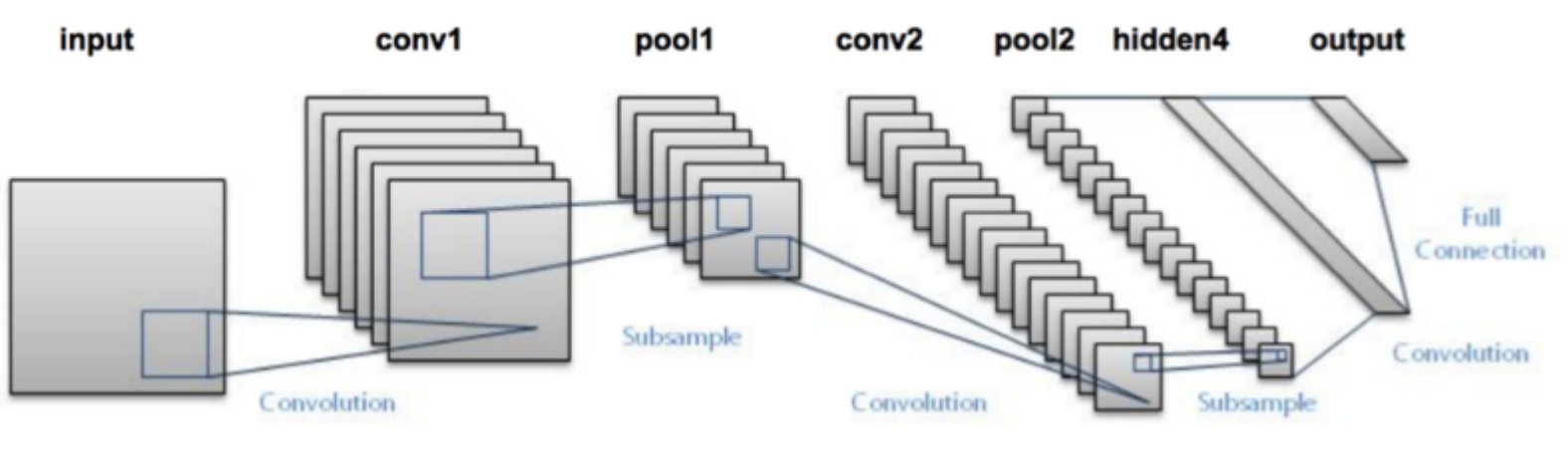
496.8	648.5
701.8	743

→ Worksheet #5 ("Pooling Layer")

https://www.analyticsvidhya.com/blog/2017/06/architecture-of-convolutional-neural-networks-simplified-demystified/

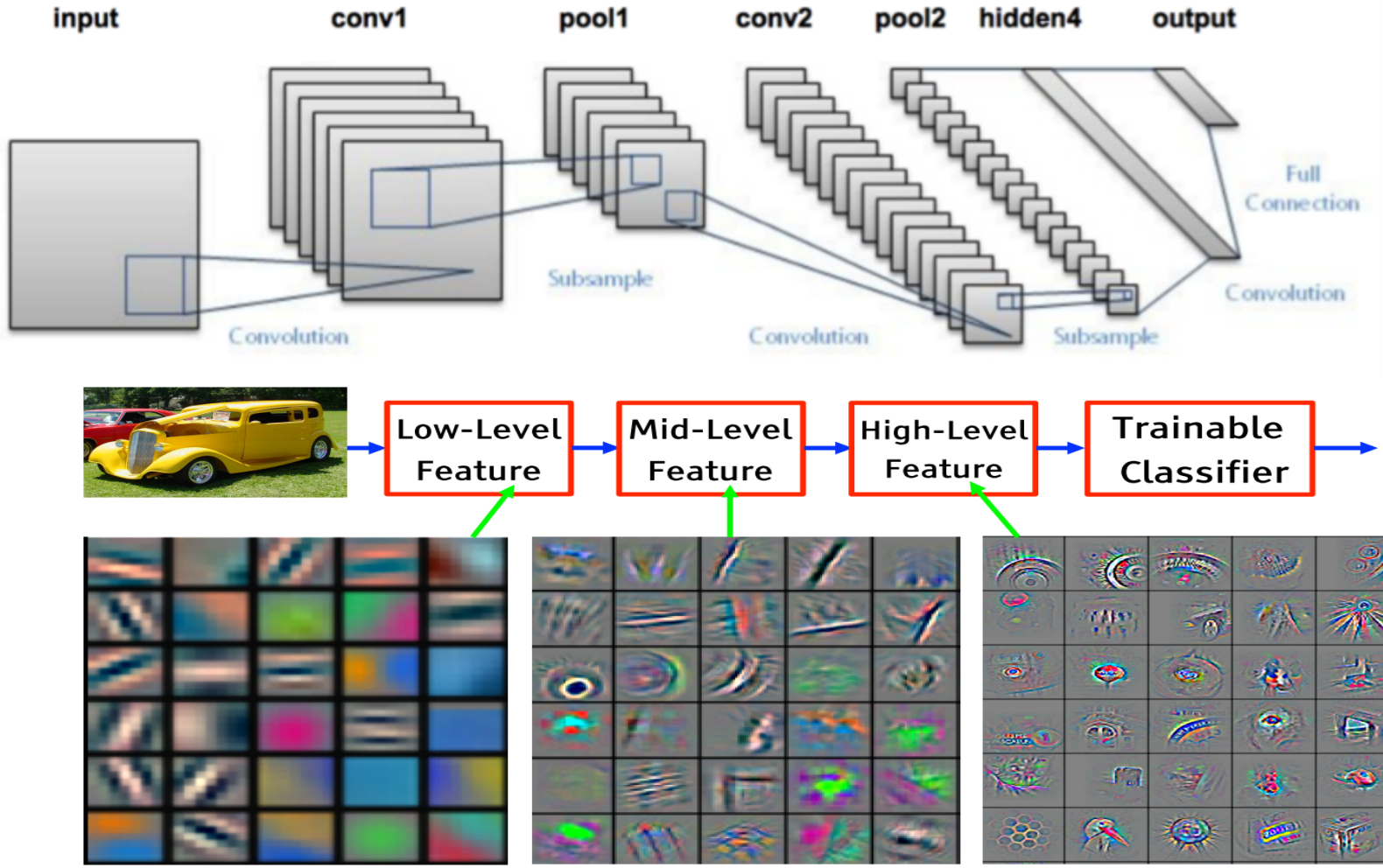
Architecture of a CNN

- Stack:
  - Convolutional Layers
  - Pooling Layers
- Finish off with:
  - A fully connected layer at the end for the final classification



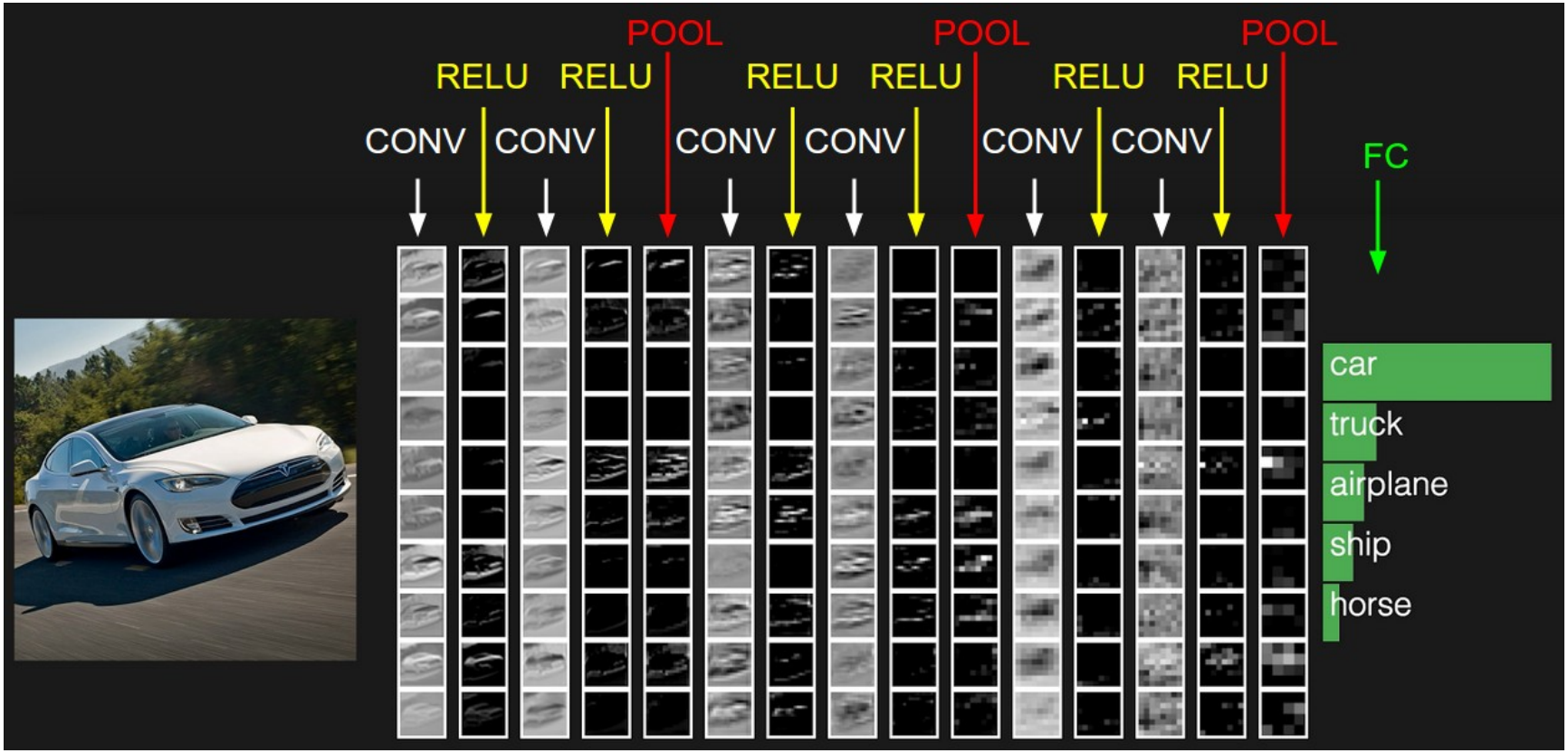
https://www.analyticsvidhya.com/blog/2017/06/architecture-of-convolutional-neural-networks-simplified-demystified/

Learning a Hierarchy of Features



https://www.analyticsvidhya.com/blog/2017/06/architecture-of-convolutional-neural-networks-simplified-demystified/

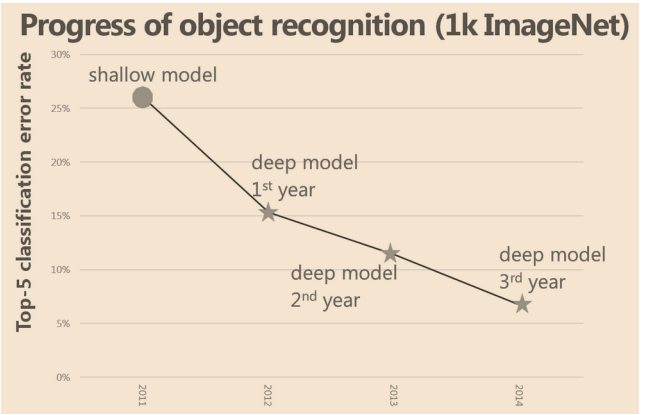
Example of a CNN



http://cs231n.github.io/convolutional-networks/

Successful CNN Networks

- LeNet
  - First successful applications of CNNs
  - Developed by Yann LeCun in the 1990's
  - used to read zip codes, digits, etc.
- AlexNet
  - First work that popularized CNNs for computer vision
  - developed by Alex Krizhevsky, Ilya Sutskever and Geoff Hinton (U. of Toronto)
  - In 2012 significantly outperformed all teams at the ImageNet ILSVRC challenge



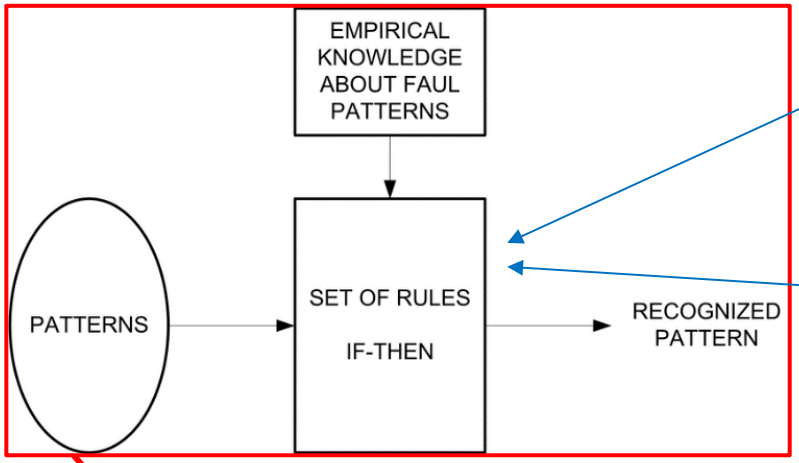
http://cs231n.github.io/convolutional-networks/

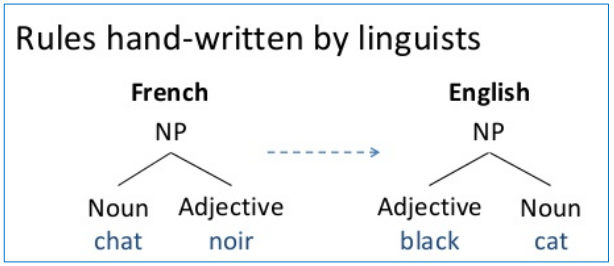
Why now?

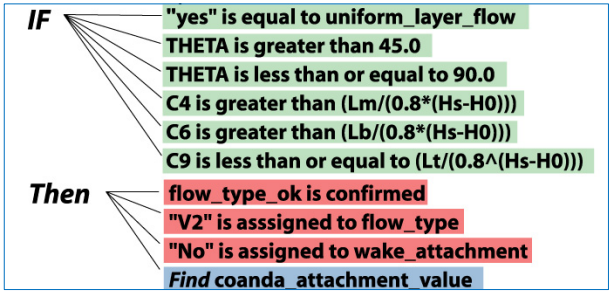
1. Basic science
   - Backpropagation did not work / overfitting…
   - *now*: developed method for training, better activation functions, better architectures….
   - Need for lots training data…
   - *now*: we have massive amounts + unsupervised pre-training
2. GPU computing
   - Neural networks take very very long to train… (days, weeks)
   - *now:* use of GPU's which are optimized for very fast matrix multiplication
3. Open Access to resources
   - *now* : Access to DL methods, code and frameworks
   - *now* : Fast turnaround from idea to implementation

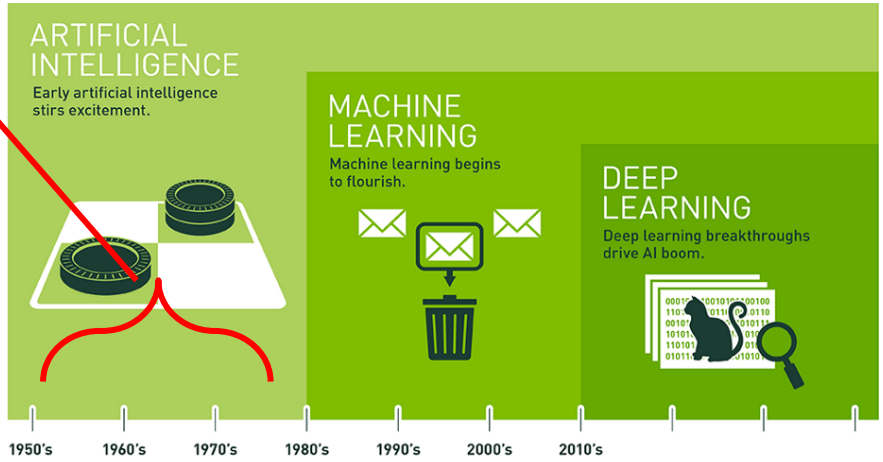
History of AI

Rules written by experts ☹
(eg. linguistics, medical doctors,…)







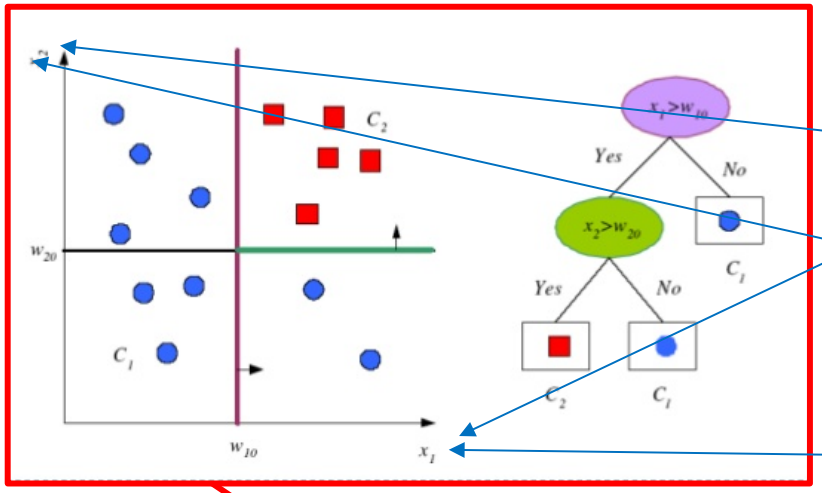


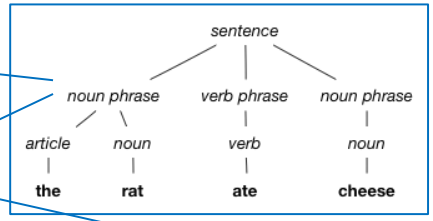
https://www.researchgate.net/profile/Dubravko_Miljkovic/publication/268239364/figure/fig30/AS:394719407427587@147…

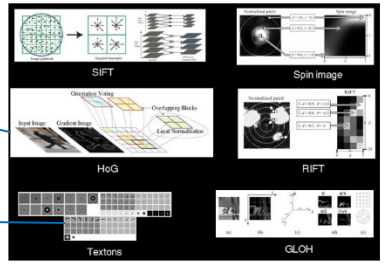
http://www.cormix.info/images/RuleTreeExample.jpg

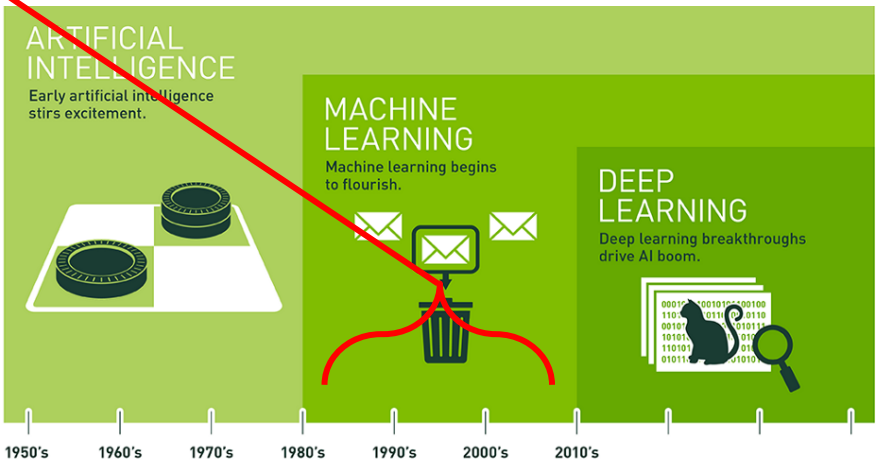
History of AI

Rules learns via the data ;-)
But: features identified by the experts
(eg. linguistics, medical doctors,…)



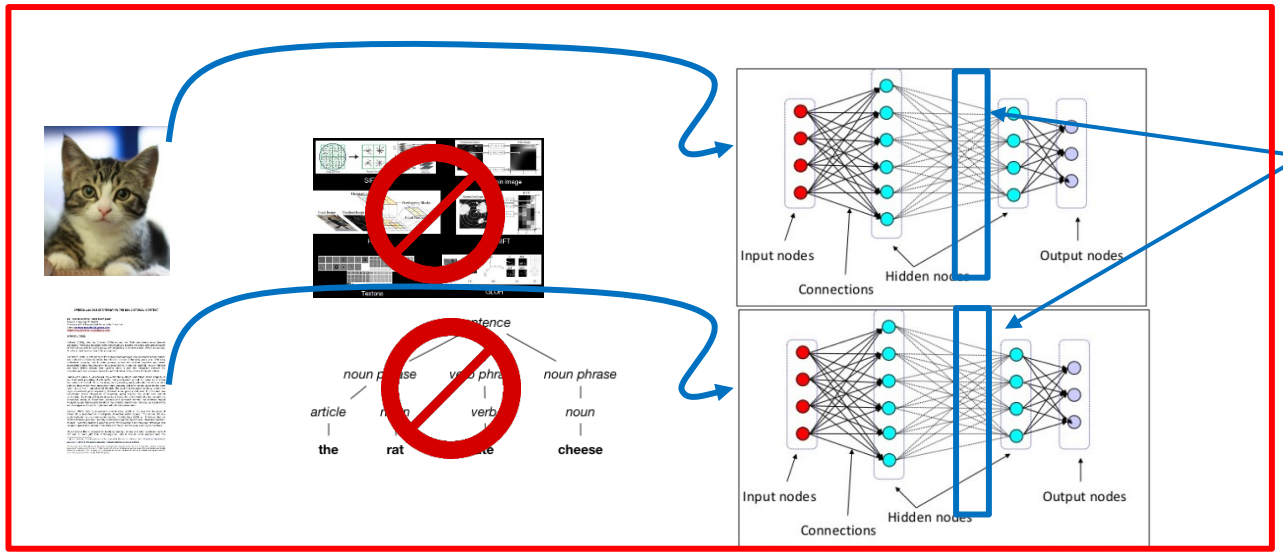




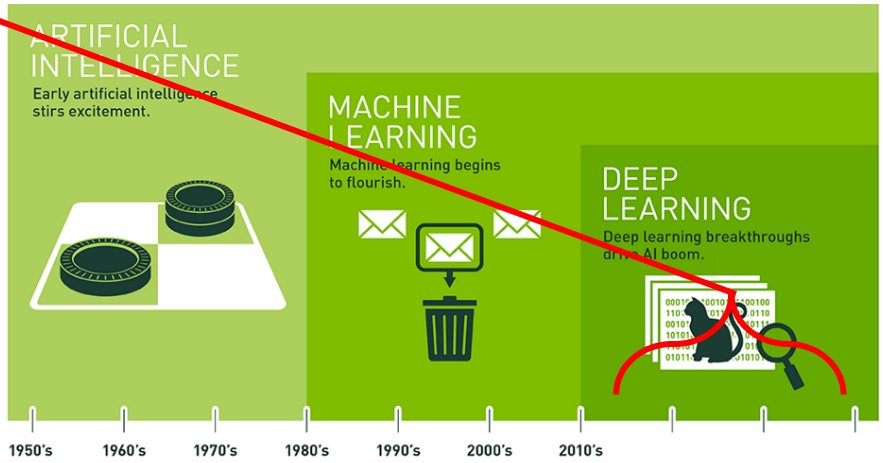


https://www.linkedin.com/pulse/goedels-incompleteness-theorem-emergence-ai-eberhard-schoeneburg/

History of AI



Rules AND features learned from the data



https://www.linkedin.com/pulse/goedels-incompleteness-theorem-emergence-ai-eberhard-schoeneburg/

Conclusion

- Deep Learning is thriving !
  - *vision*
  - *image processing*
  - *speech recognition*
  - *natural language processing*
  - …
- Canada is a world leader in Deep Learning
  1. Montreal: (Bengio et al.) MILA
  2. Toronto: (Hinton et al.) Vector Institute
  3. Edmonton: (Sutton et al.) AMII