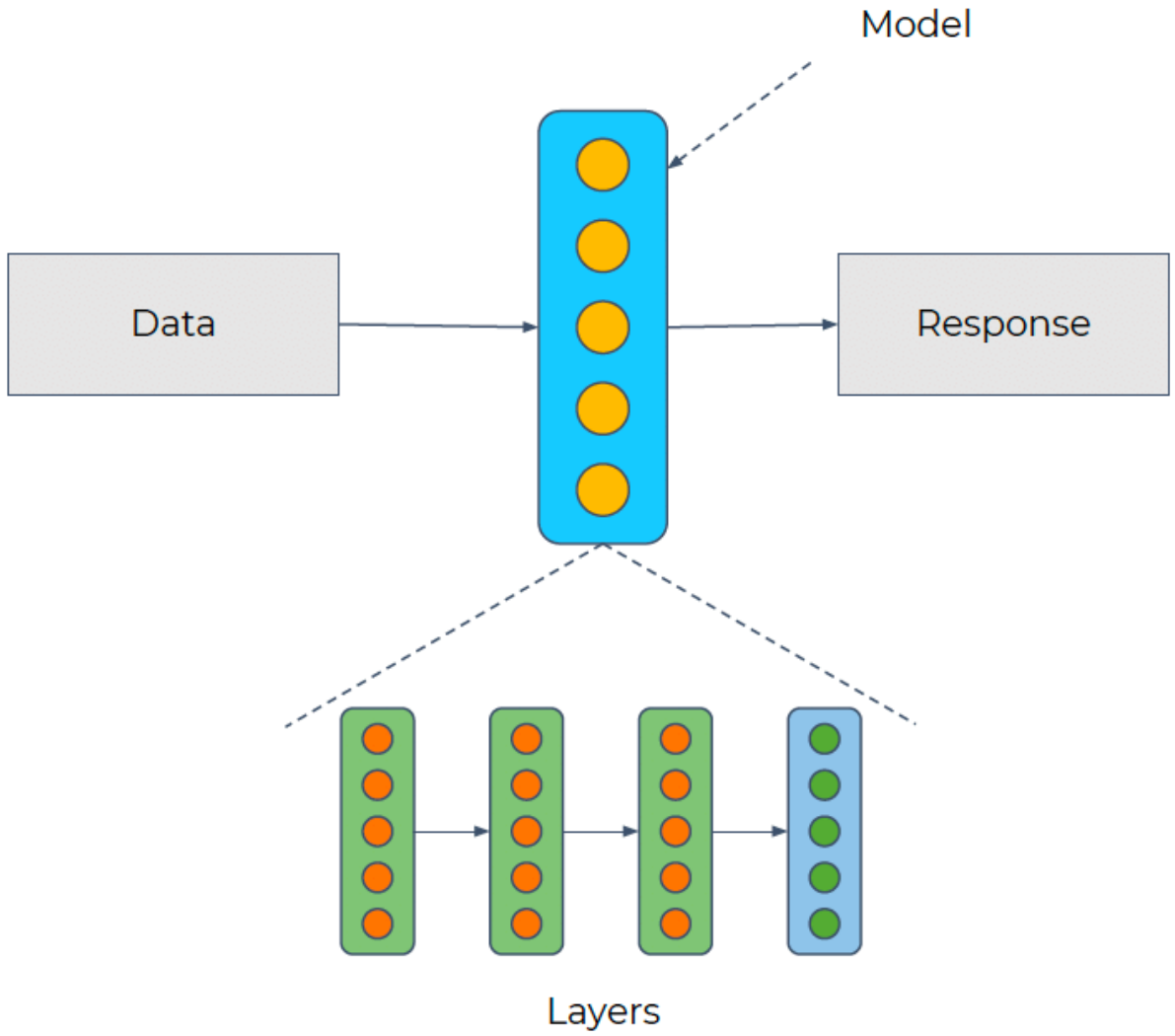
Deep Learning

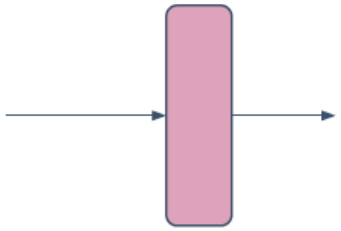
COMP 6721 Introduction of AI

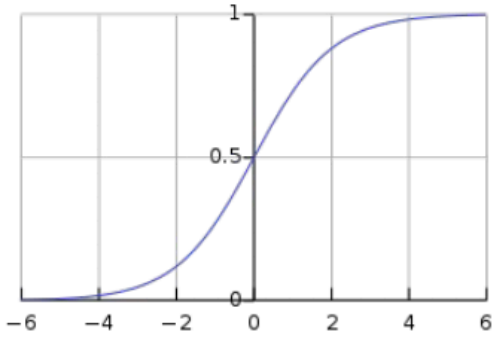
Russell & Norvig – Chapter 23.1 + 23.2 + 23.3

Modular Approach

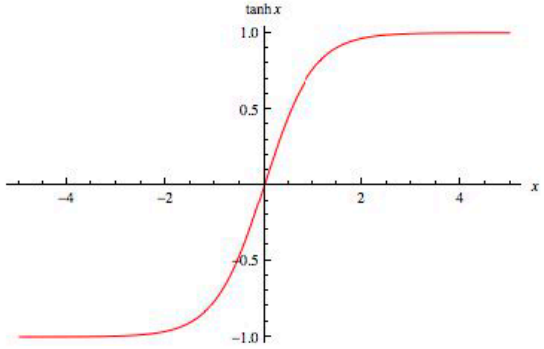


Activation Functions

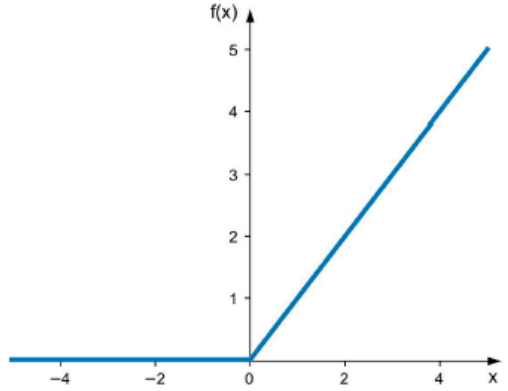




Sigmoid



Tanh



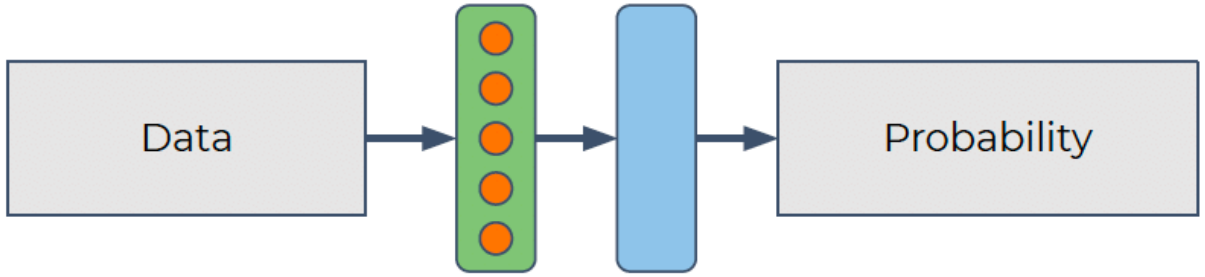
ReLU

Modular Approach

- A module is an elementary building block that you can combine with others to create new modules.
- The output of a module is the input of another one.
- Many elementary modules are implemented efficiently in modern deep learning libraries.

How to build a probabilistic model?

- Binary Classification



Source: Makhmutova Dina, Unsplash

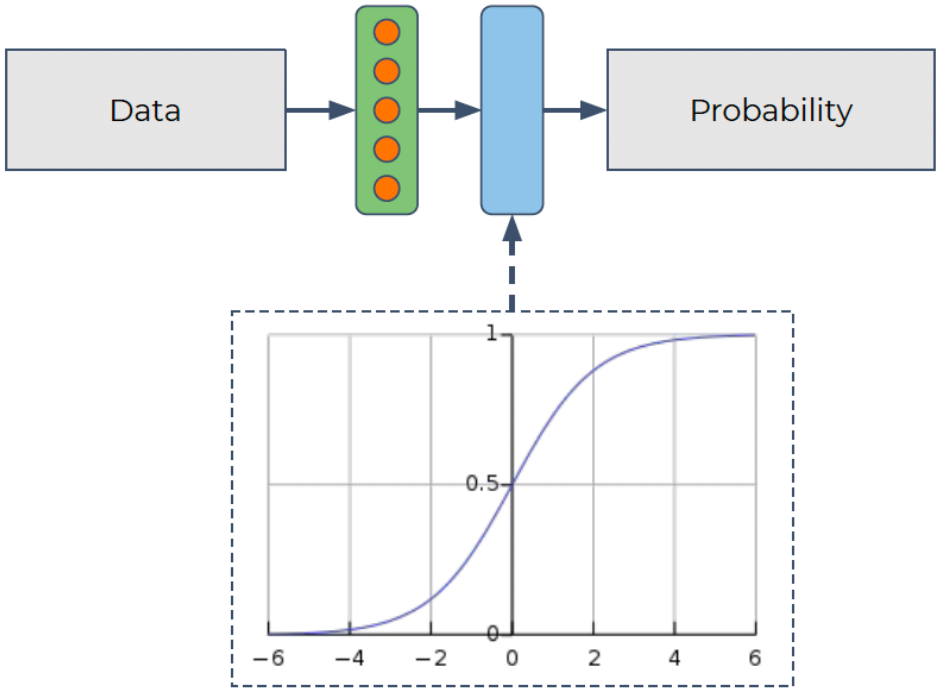
Cat	90%
Not cat	10%

How to build a probabilistic model?

- Binary Classification
- Dense module for mapping to a single scalar.
- A **sigmoid** module to normalize the scalar between 0 and 1.
- Output is the parameter p of a Bernoulli distribution:

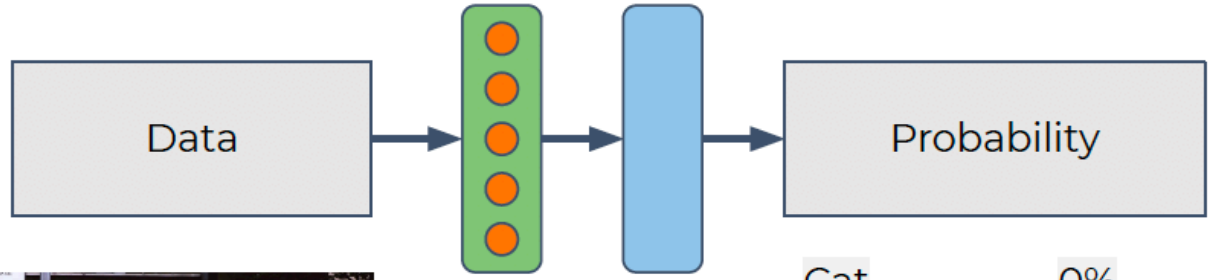
- p is the probability of class 1

- 1-p is the probability of class 0.



How to build a probabilistic model?

- Multi-label Classification

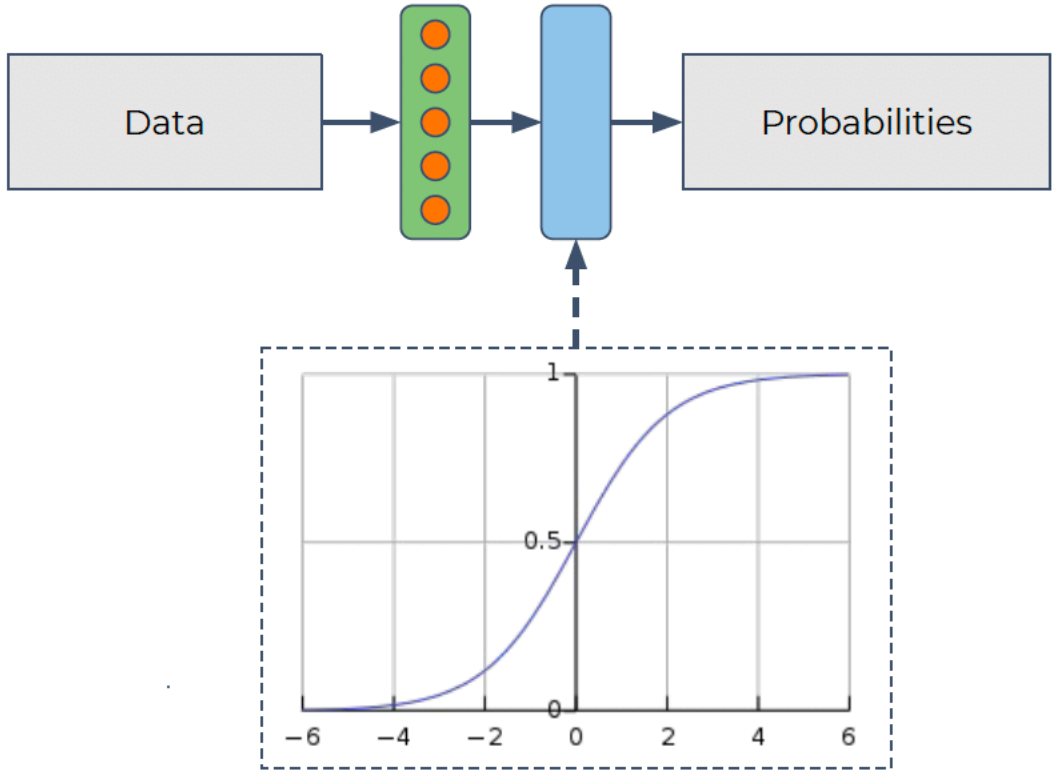


Source: Daryan Shamkhali, Unsplash

Cat	0%
Car	60%
Person	95%
Tree	75%
Dog	0%
Bike	60%

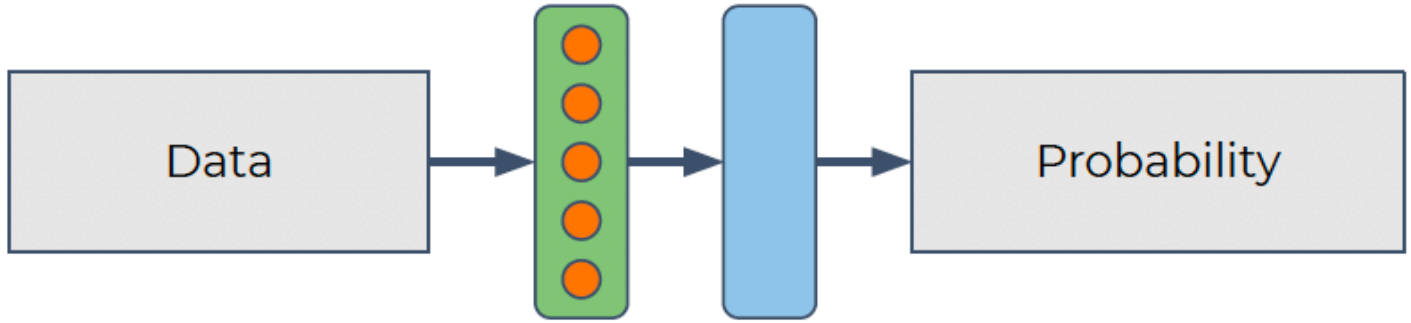
How to build a probabilistic model?

- Multi-label Classification
- **Dense module** for mapping to the number of classes.
- A **sigmoid module** to normalize each component between 0 and 1.



For each component

How to build a probabilistic model?



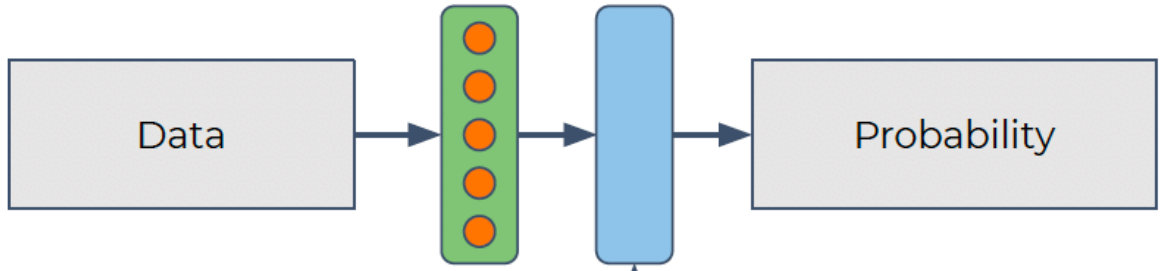
Source: Makhmutova Dina, Unsplash

Cat	90%
Not cat	10%

Cat	90%
Dog	2%
Horse	0%
Tiger	8%
Penguin	0%

How to build a probabilistic model?

- Multiclass Classification:
- **Dense module** for mapping to the number of classes.
- A **softmax module** to normalize each component between 0 and 1 and to have their sum equals to 1.



$$p_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$

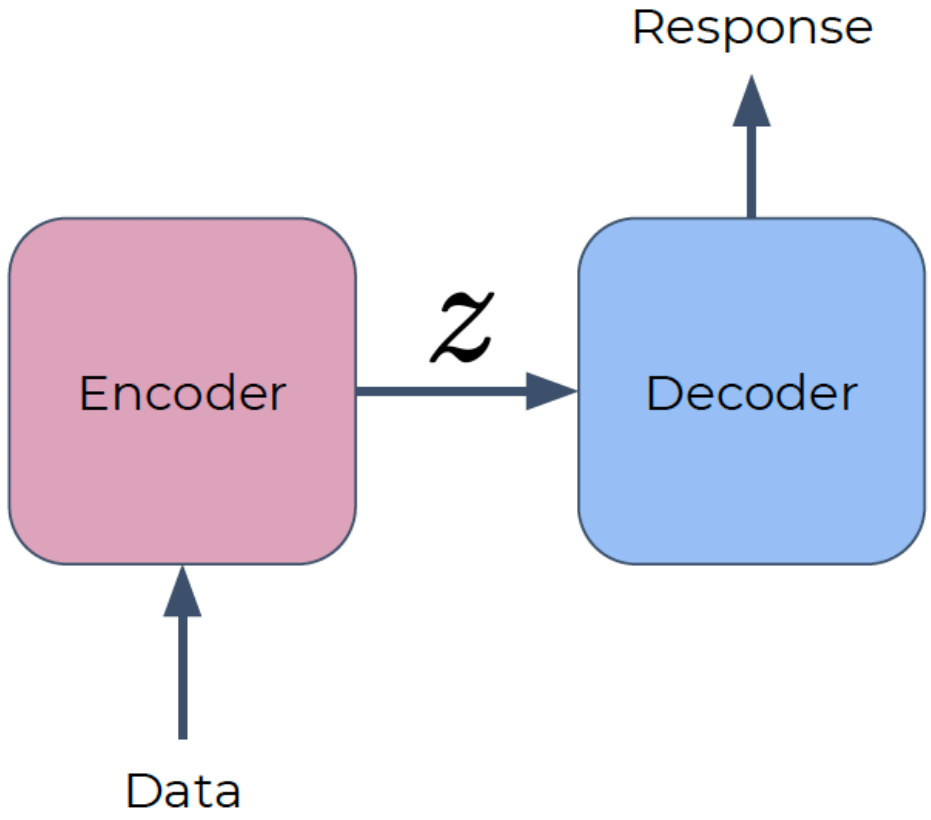
Design Patterns

- Encoder-decoder
  - Multi-input encoder
  - Multi-output decoder
- Teacher-student networks
- Modulation
- Bottom-up approach
- Adversarial
- Auto-regressive, ...

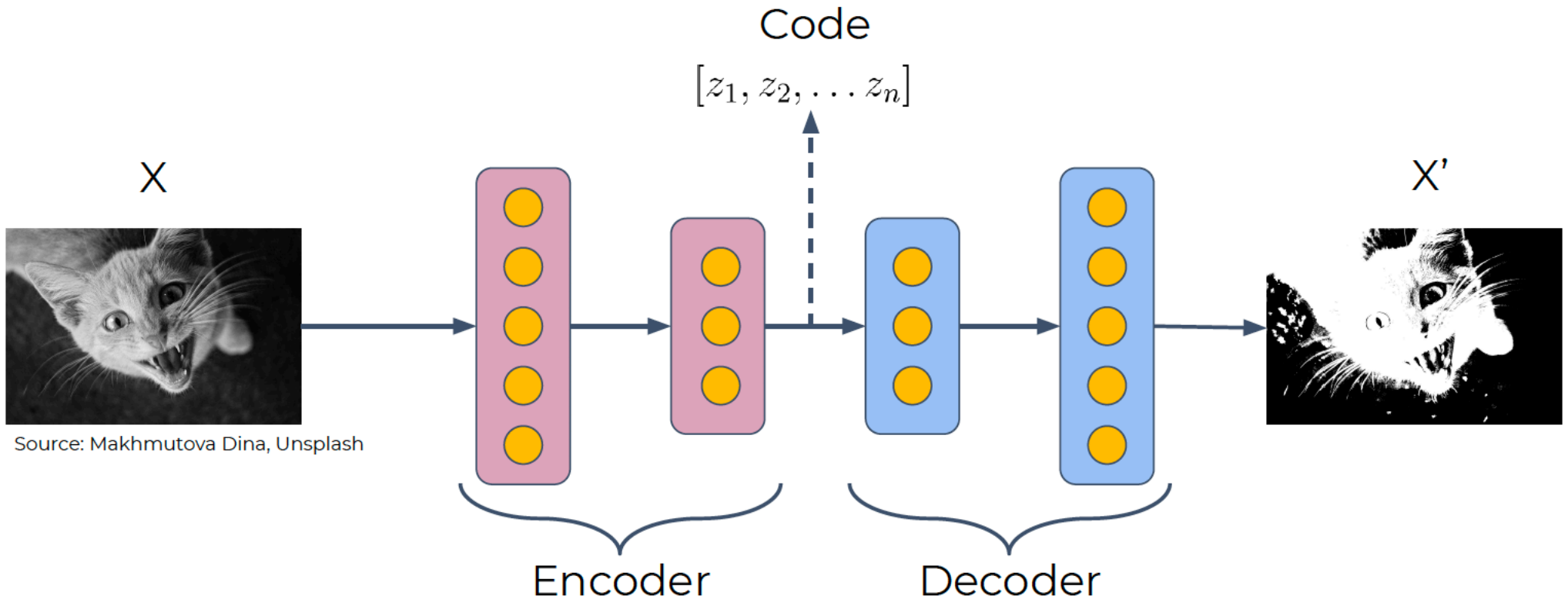
Source: Koushik Chowdavarapu, Unsplash

Encoder-decoder

- Learning a latent representation z
- Uses
  - Autoencoder
  - Seq2seq (RNN)
  - Encoder-decoder network

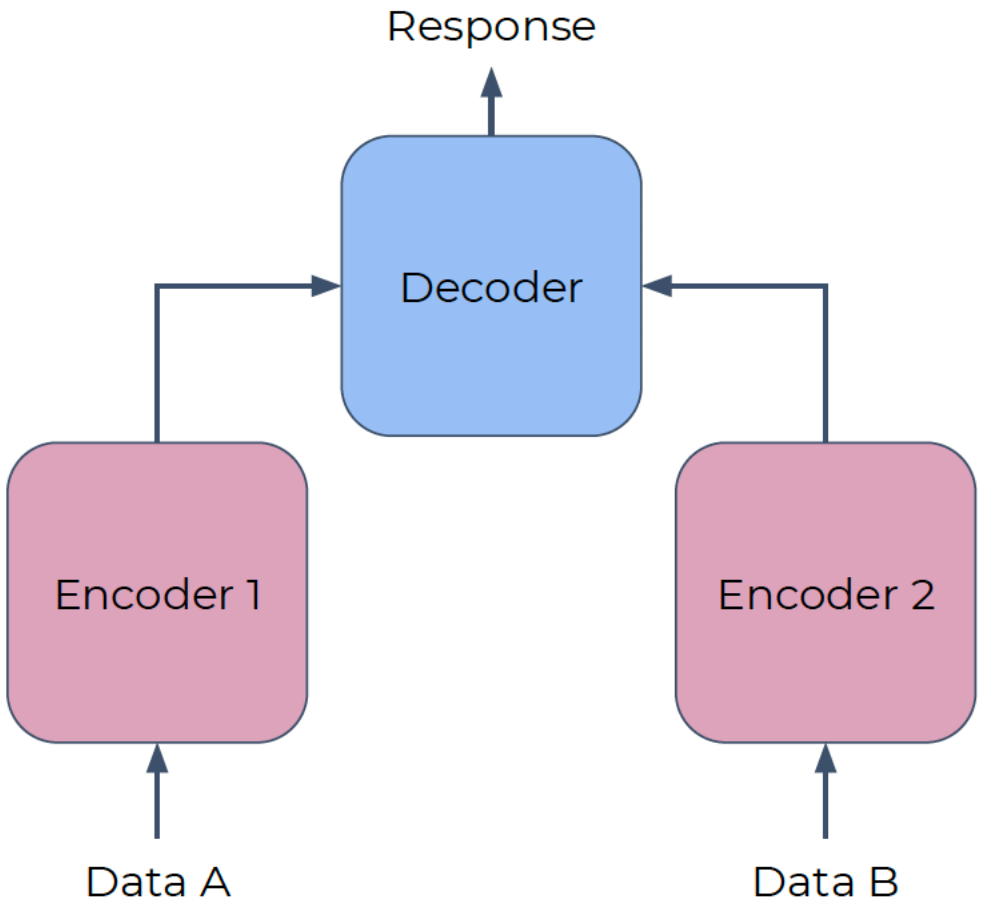


Example: Autoencorder



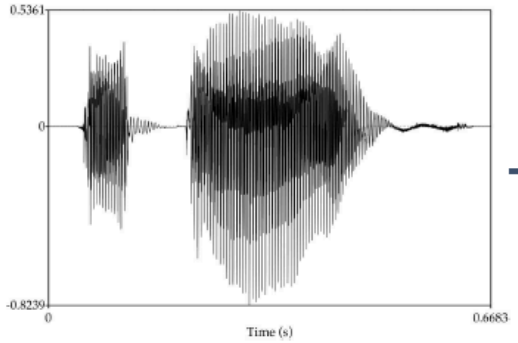
Multi-input Encoder-decoder

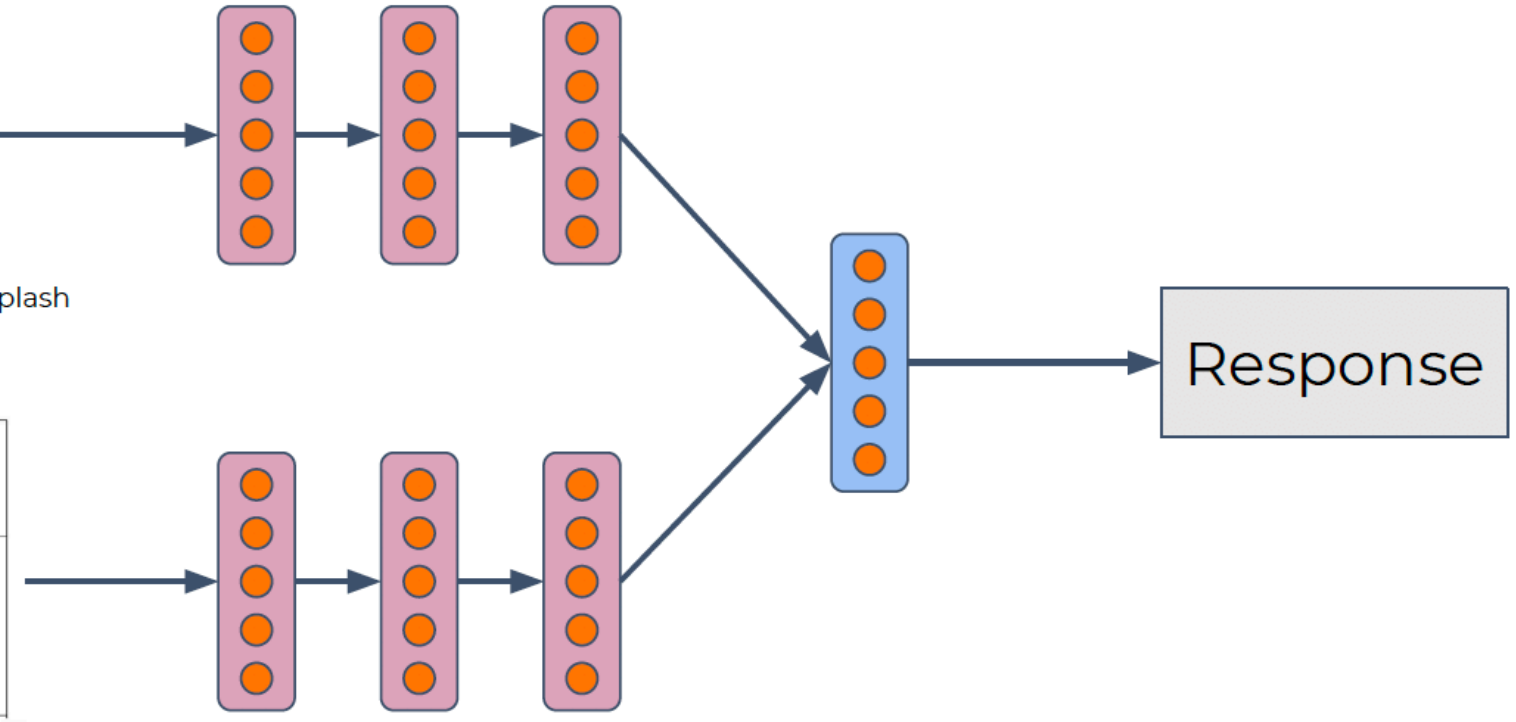
- Encode and merge different information flows
- Uses:
  - Process different modalities
  - Metric learning
  - Relation learning



Example: multi-modality

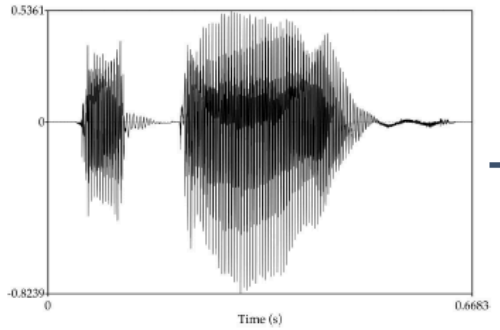
Source: Makhmutova Dina, Unsplash





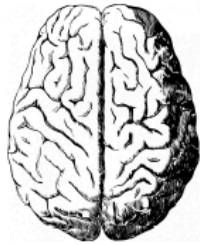
Example: multi-modality

Source: Makhmutova Dina, Unsplash



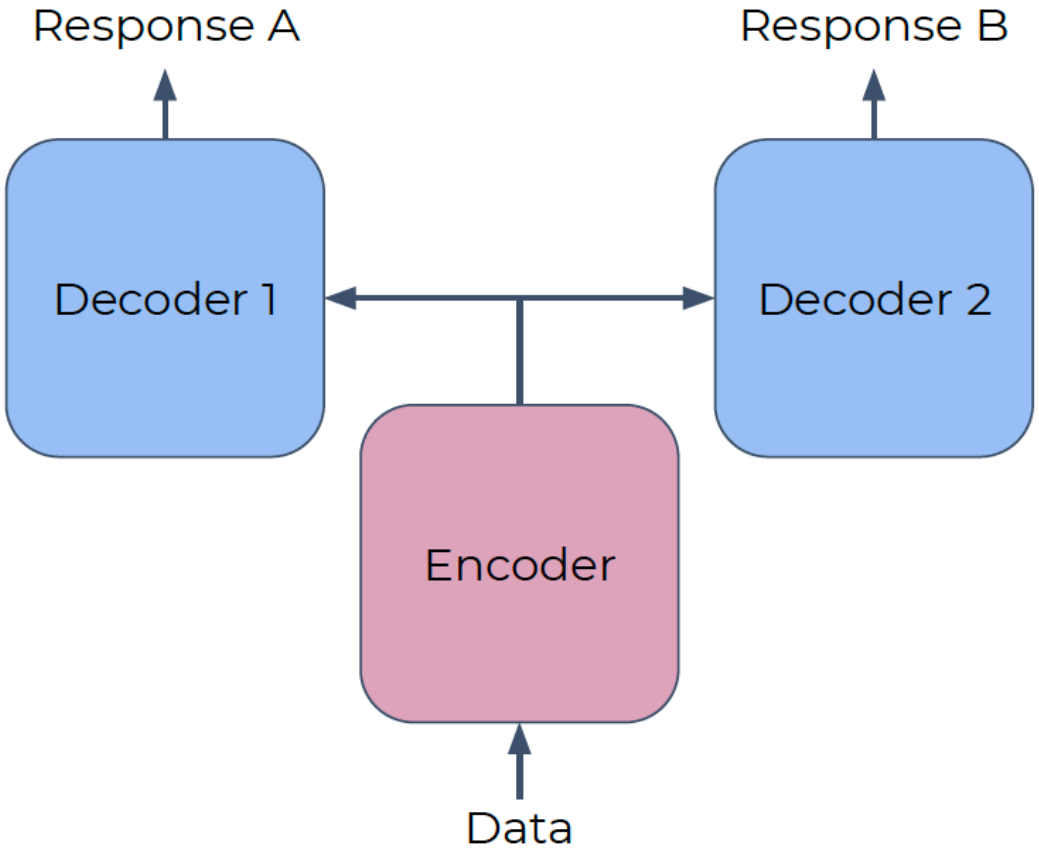
$[z_1, z_2, \ldots z_n]$

Latent space



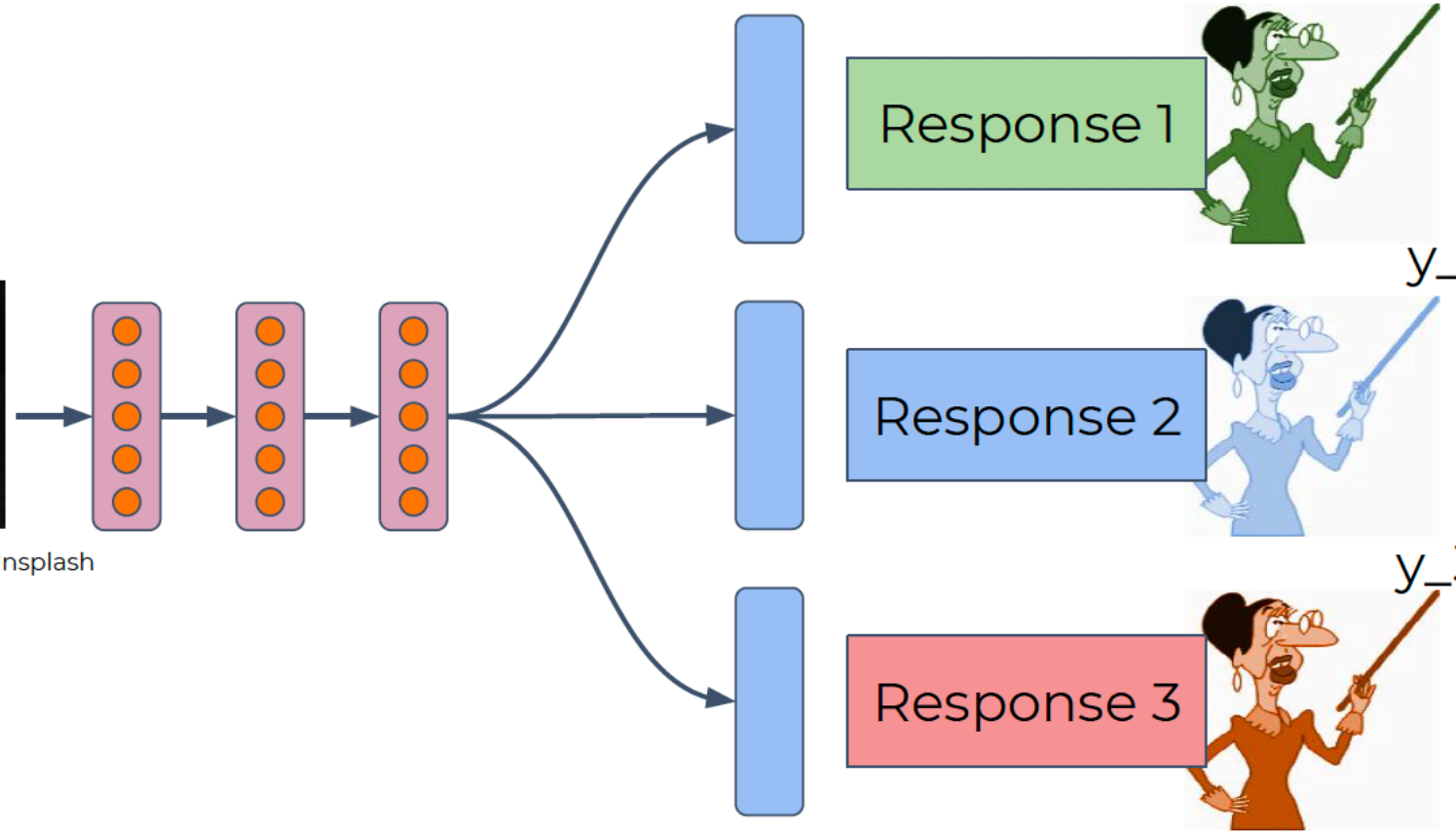
Multi-output Encoder-decoder

- Common representation
- Uses:
  - Multi-task learning



Example: multi-task learning

Source: Makhmutova Dina, Unsplash



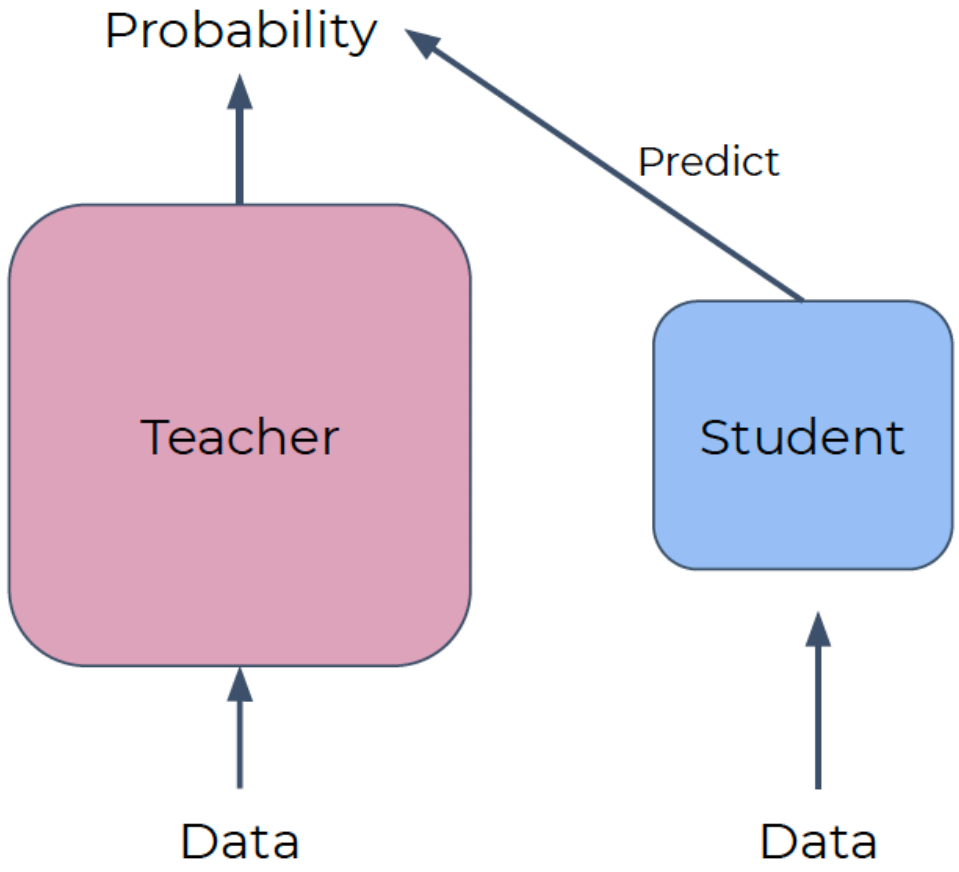
y_1 = Cat

y_2 = Yellow

y_3 = Danger

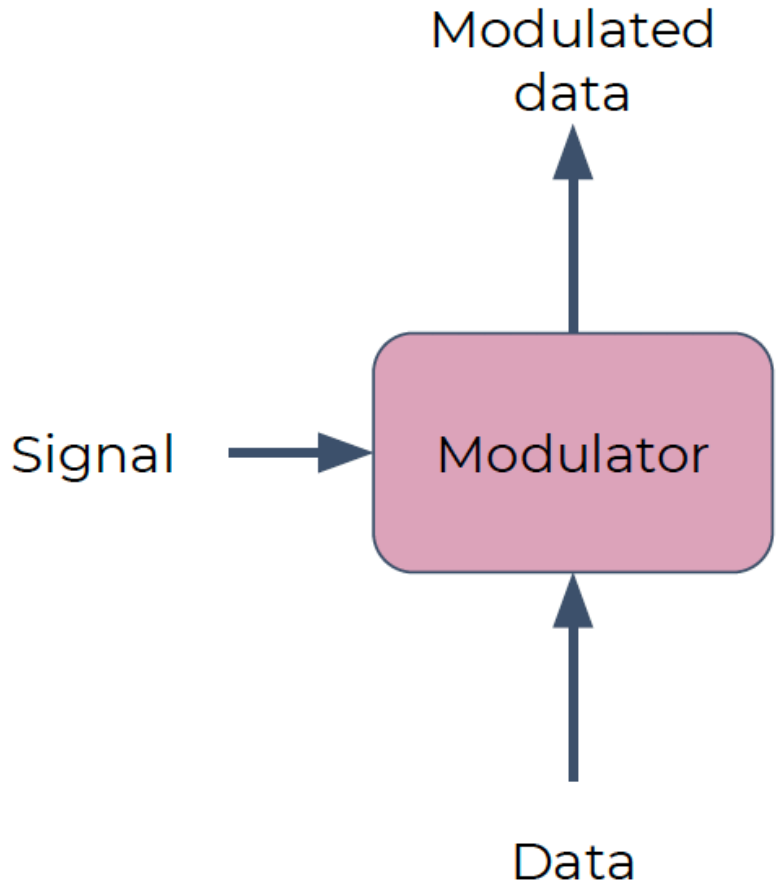
Teacher-student architecture

- Learn to predict the output of another model
- Uses:
  - Model compression

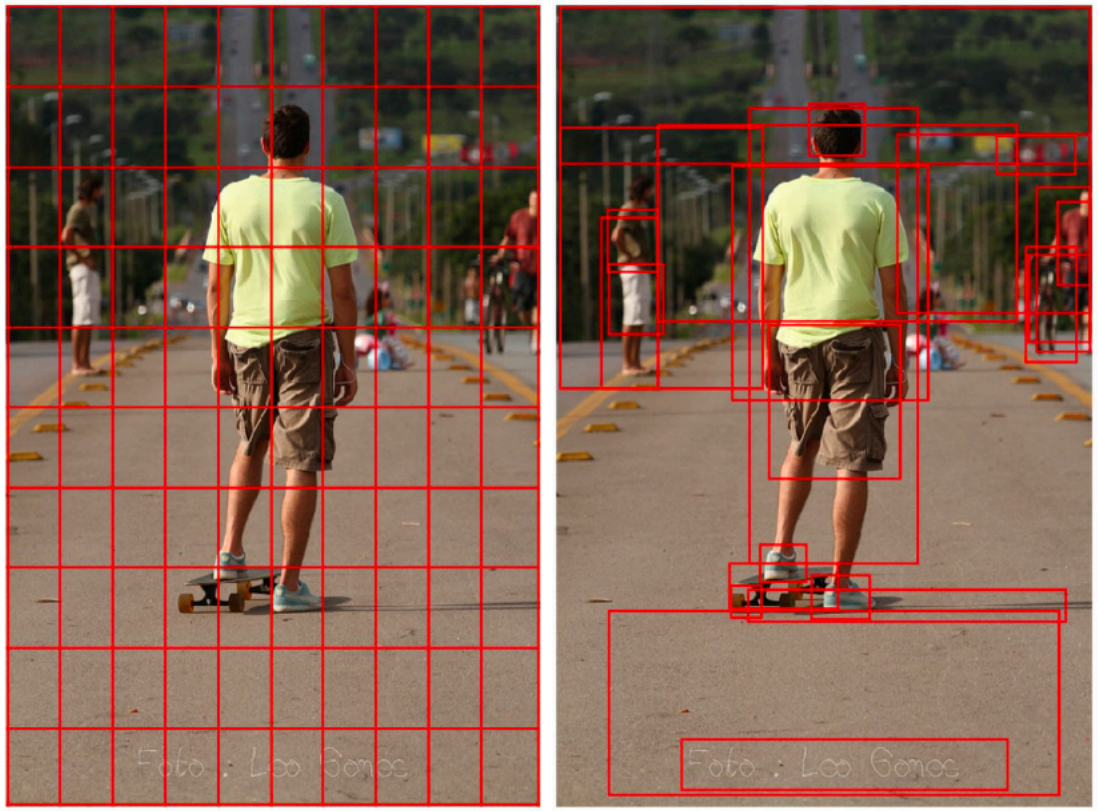


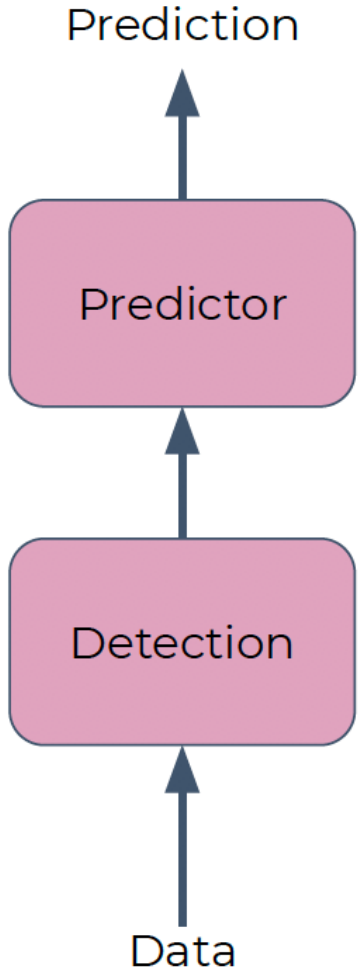
Modulation

- Adaptive modulation of the representation
- Uses:
  - Gating (LSTM)
  - Skip-connections
  - Batch norm
  - FiLM
  - Attention mechanism



Bottom-up Approach





Anderson, Peter, et al. "Bottom-up and top-down attention for image captioning and visual question answering." CVPR. Vol. 3. No. 5. 2018.

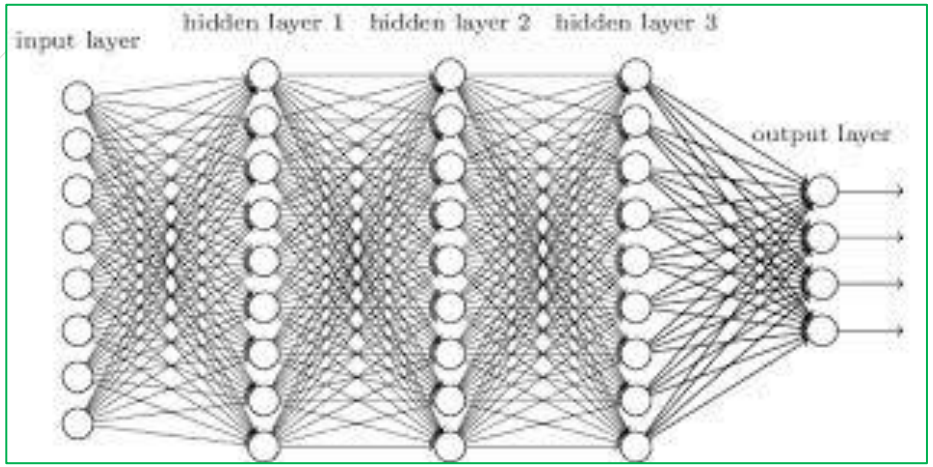
Summary

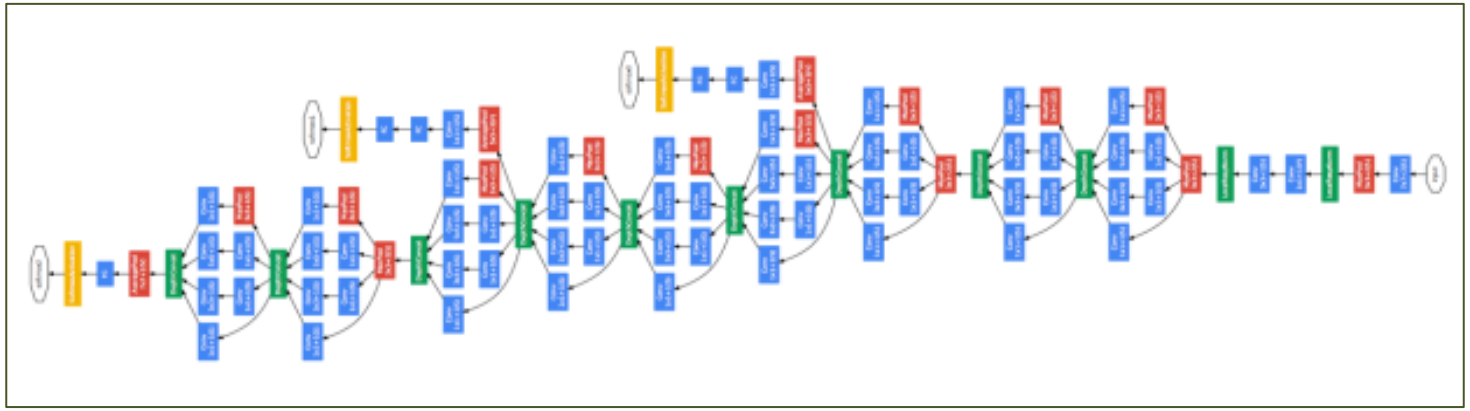
- With deep learning, we can define hypothesis classes with computational graphs (i.e., architectures) easily.
- Creating a graph for a given task and dataset is part of the expertise.
- An architecture often has many layers; each is providing a new distributed representation to the next one that will be easier to process.
- Design patterns for architecture are emerging in the literature.

Topics

- Feature Learning
- Training a Deep Neural Networks

A Deep Neural Network





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/

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 ☹

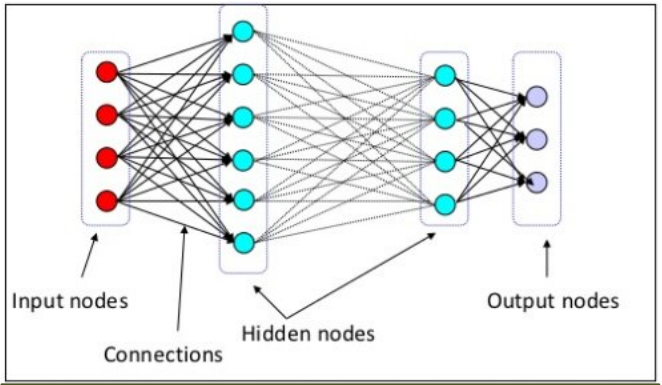
Initial Drawbacks (1)

1. Standard backpropagation with sigmoid activation function 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



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

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

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

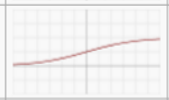
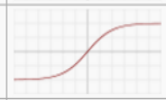
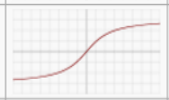
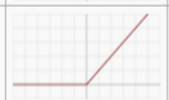
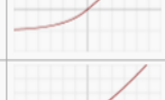
$$\delta_4 = O_4\,(1 - O_4) \times \sum\ (w_{4,5}\,\delta_5)$$

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

…

Initial Drawbacks (1)

- To help, we can :
  a) Use other activation functions…
  b) Do "gradient clipping" (i.e. set bounds on the gradients)

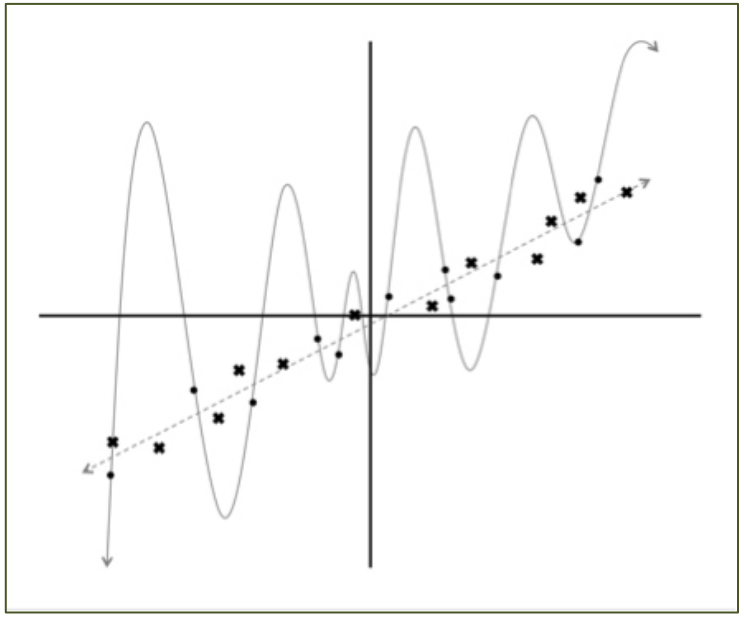
Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1-f(x))$
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”

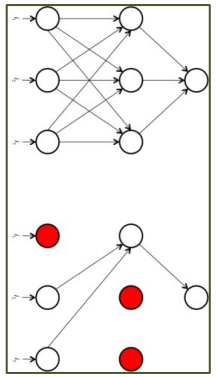


Initial Drawbacks (2)

2. Overfitting
   - *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.

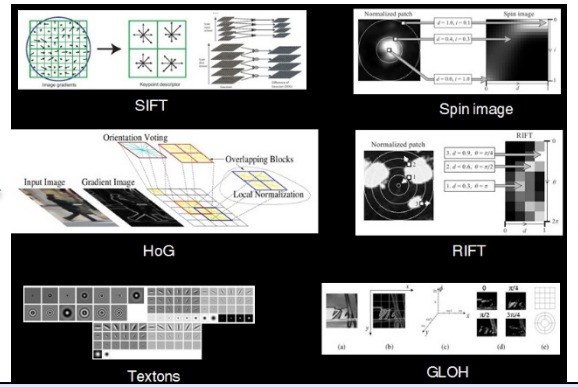


Initial Drawbacks (3)

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

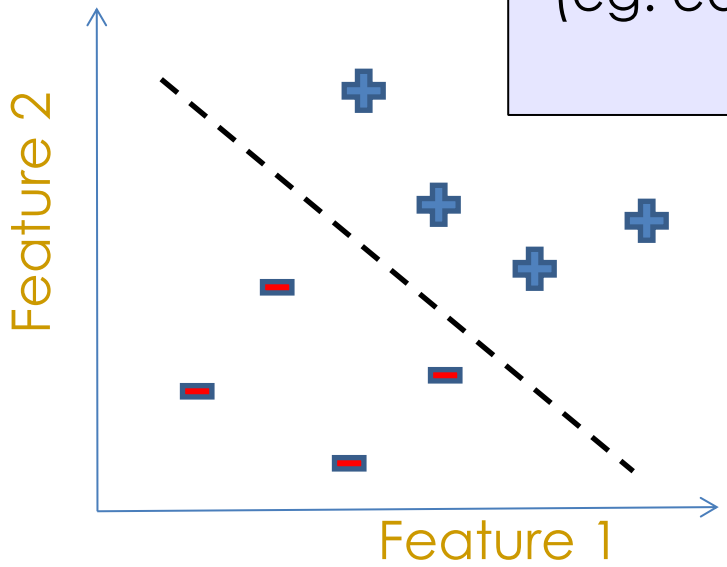
Classic ML

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

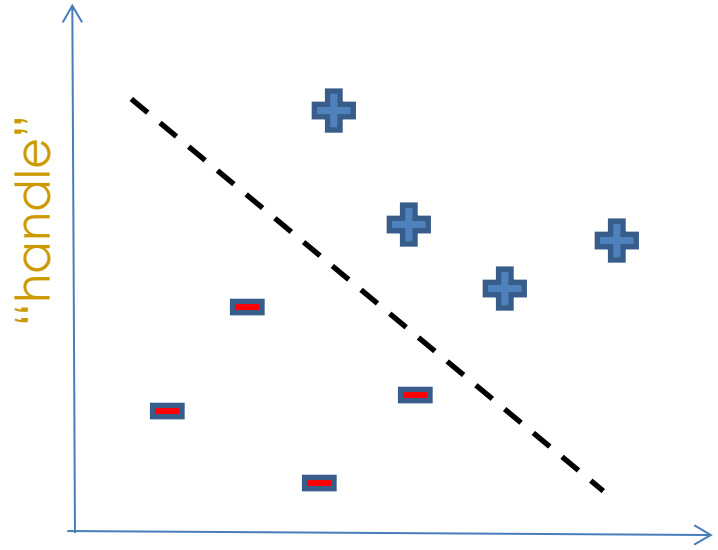
Automatic Feature Learning

Input

Automatic Feature Representation

eg. handle, wheel, …

Learning algorithm



Motorbikes

"Non"-Motorbikes

With Automatic Feature Learning:

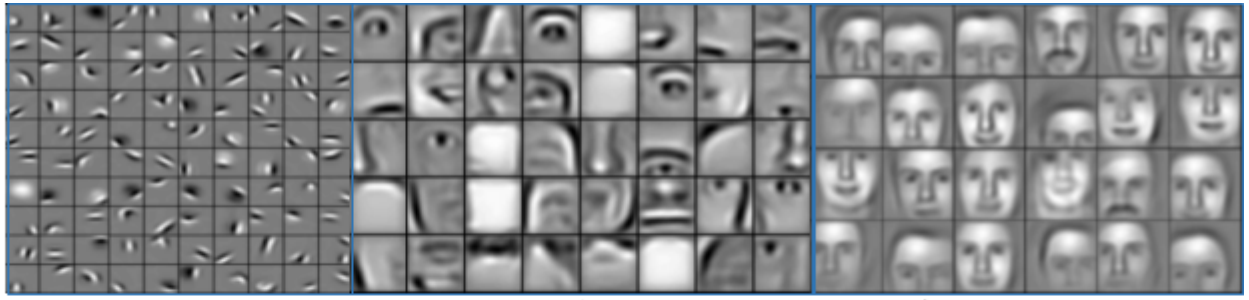
1. We feed the network the raw data (not feature-curated)

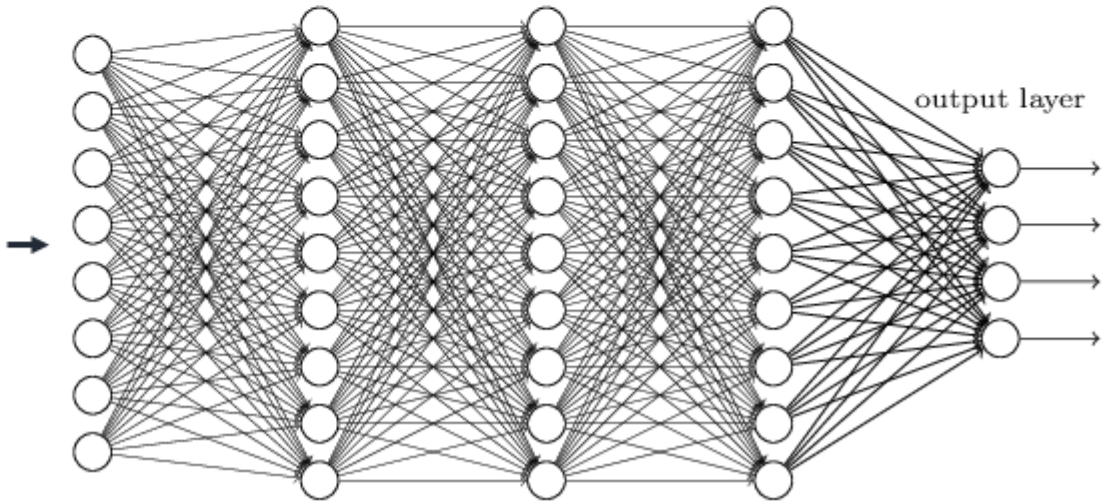
2. The features are learned by the network

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

Automatic Feature Learning

Deep neural networks learn hierarchical feature representations





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

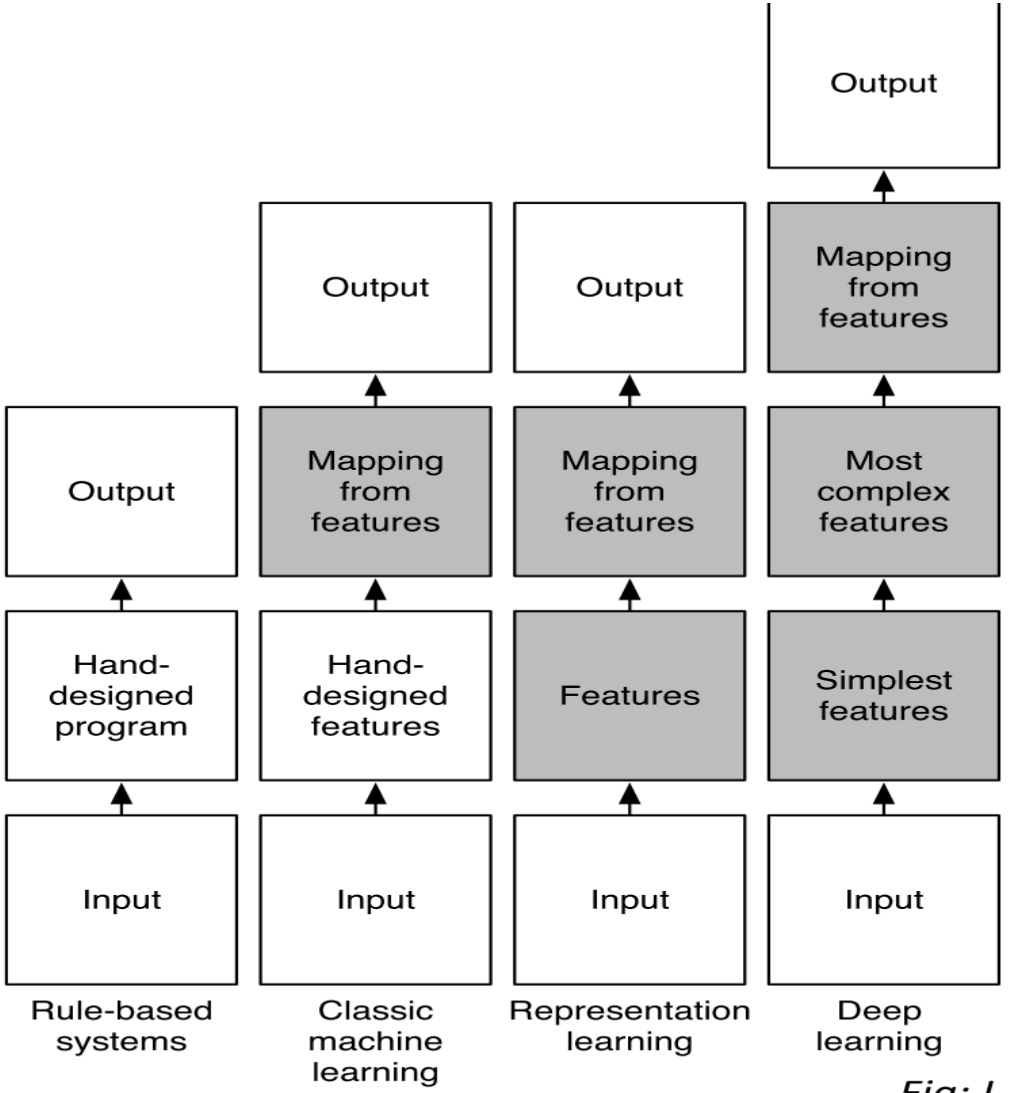
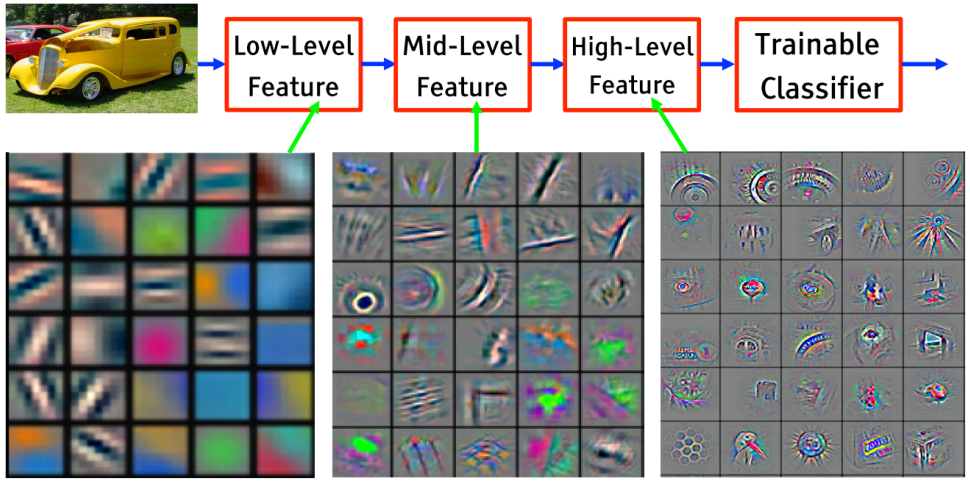
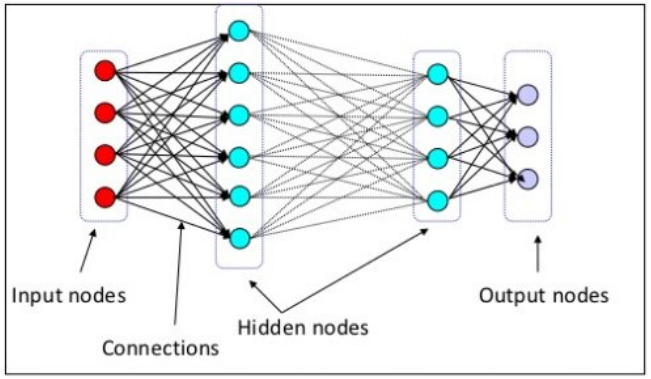


Fig: I. Goodfellow

What types of features?

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

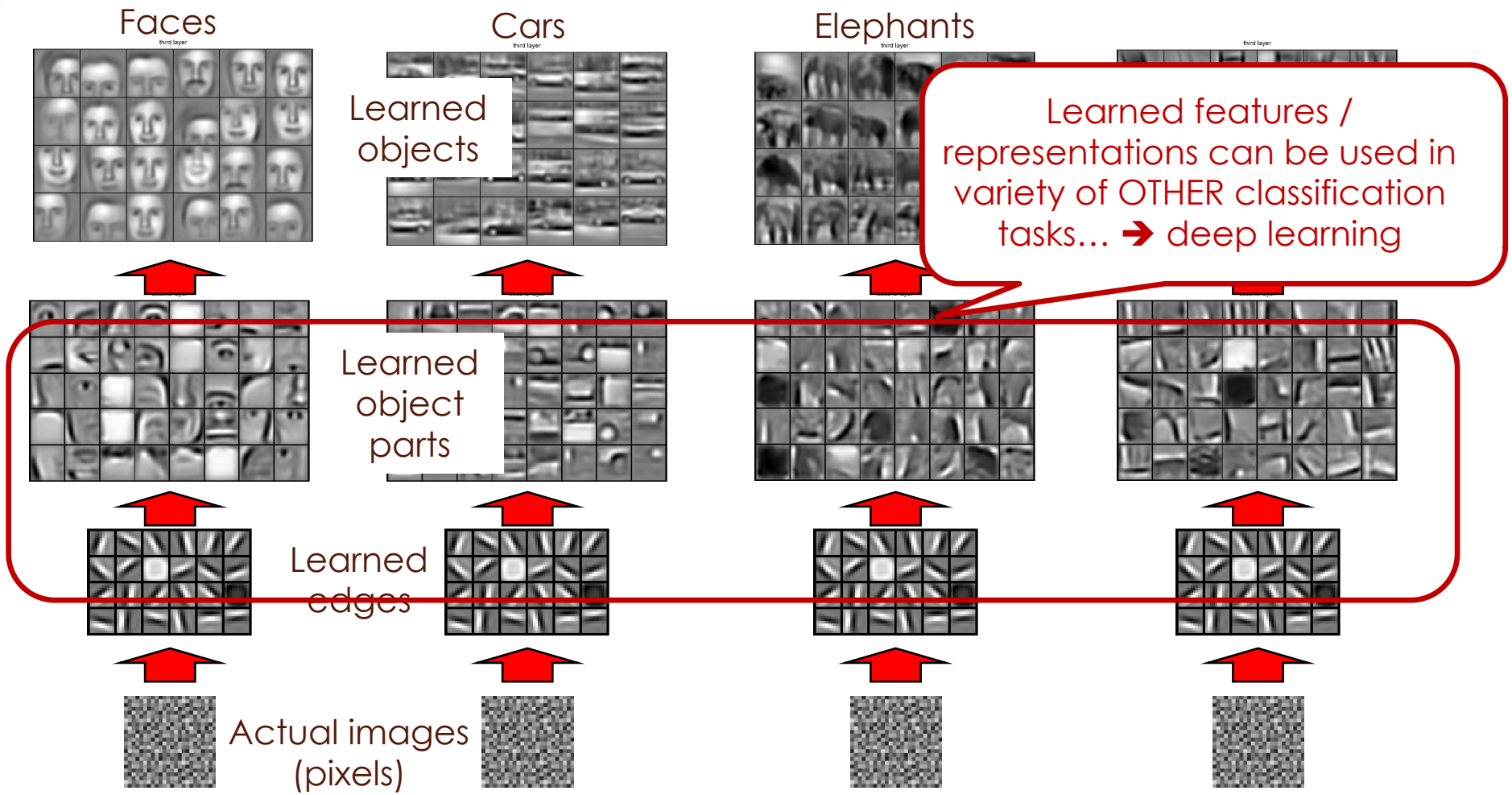




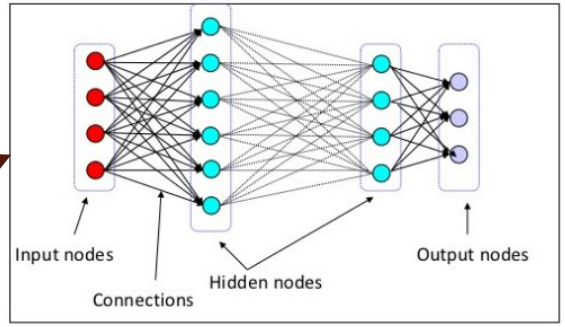
- For NLP
  - character -> word -> constituents -> clause -> sentence -> discourse
- For speech:
  - sample ->spectral band -> sound -> … phone -> phoneme -> word

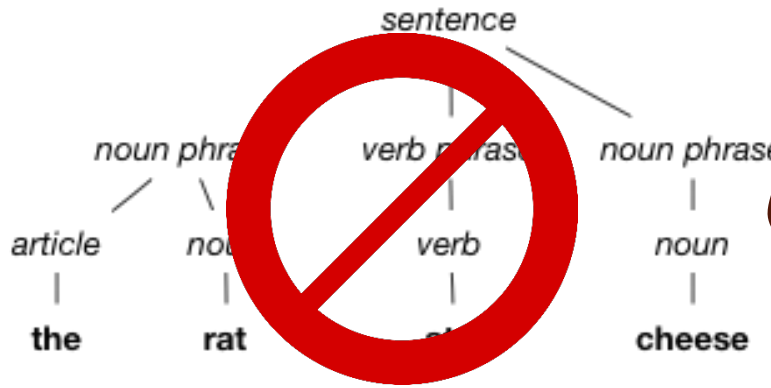
Example: Learning Image Features

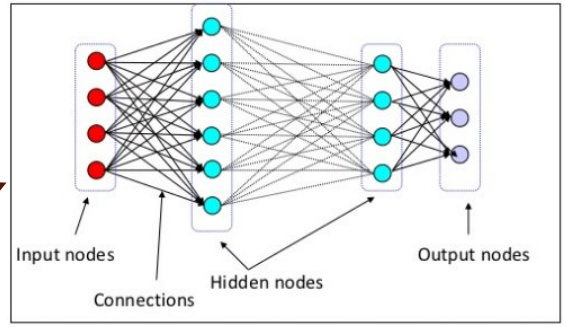
Examples of learned objects parts from object categories



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

Advantages of Unsupervised Feature Learning

- **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)

Topics

- ***Feature Learning***
- Training a Deep Neural Networks

General Architecture of Deep Network

1. Unsupervised pre-training of neural network using unlabeled data
   - unsupervised learning of features
   - Done via auto-encoders
2. Supervised training with labeled data using features learned from above with a standard classifier
   - Eg. Support Vector Machine, a feed forward ANN with backpropagration, …

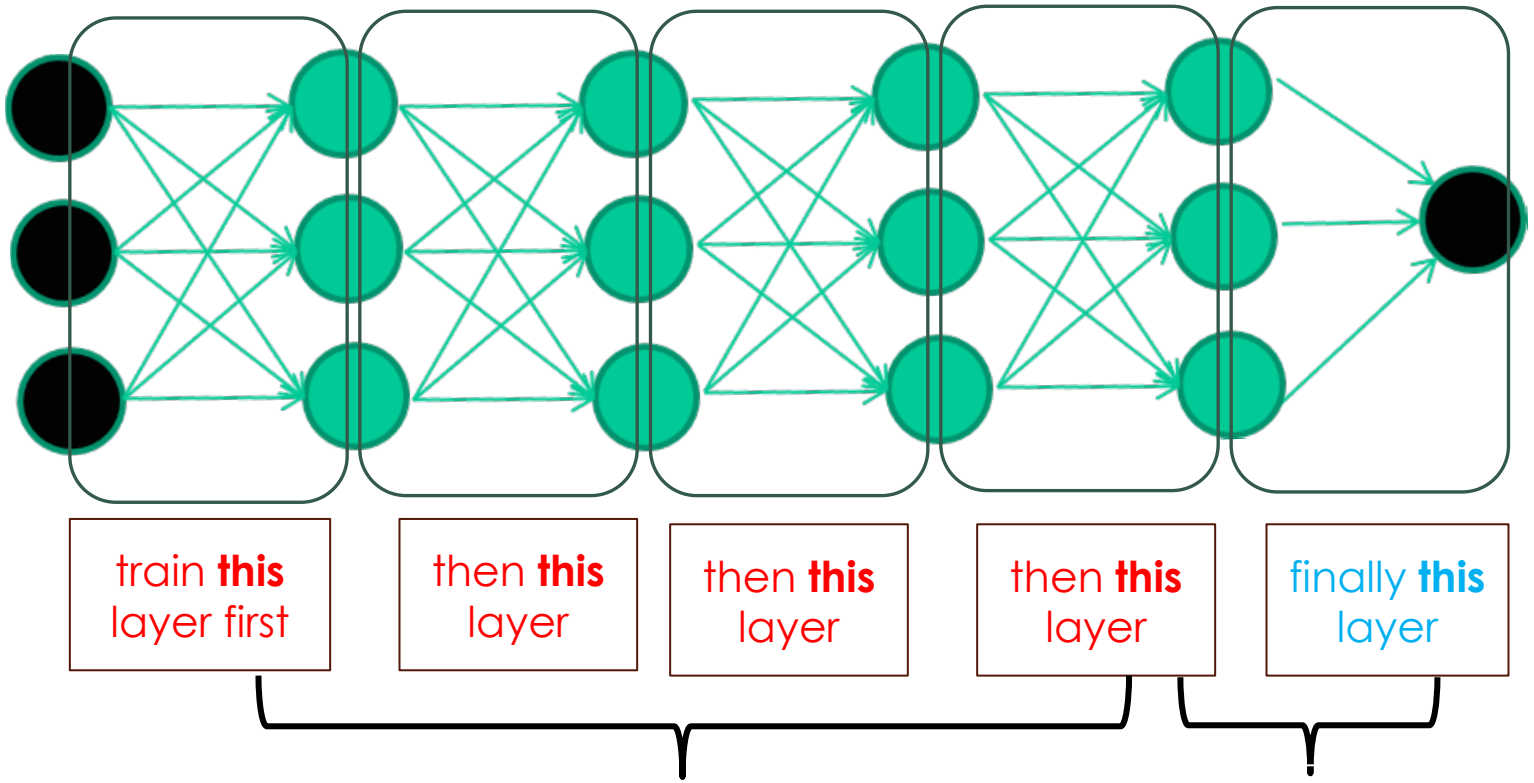
Major Deep Learning Models

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

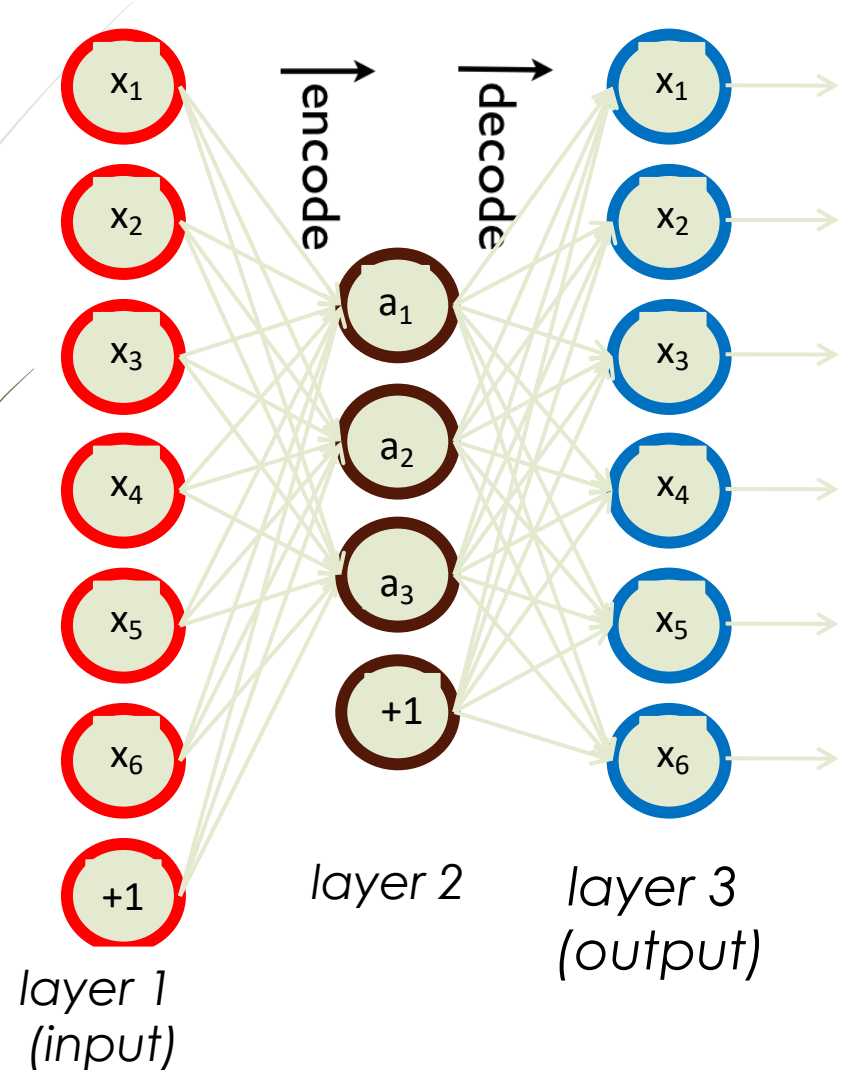
Training a Deep Neural Network



EACH of the (non-output) layers is trained to be an RBM or an autoencoder which is "pretraining the network with unsupervised data"

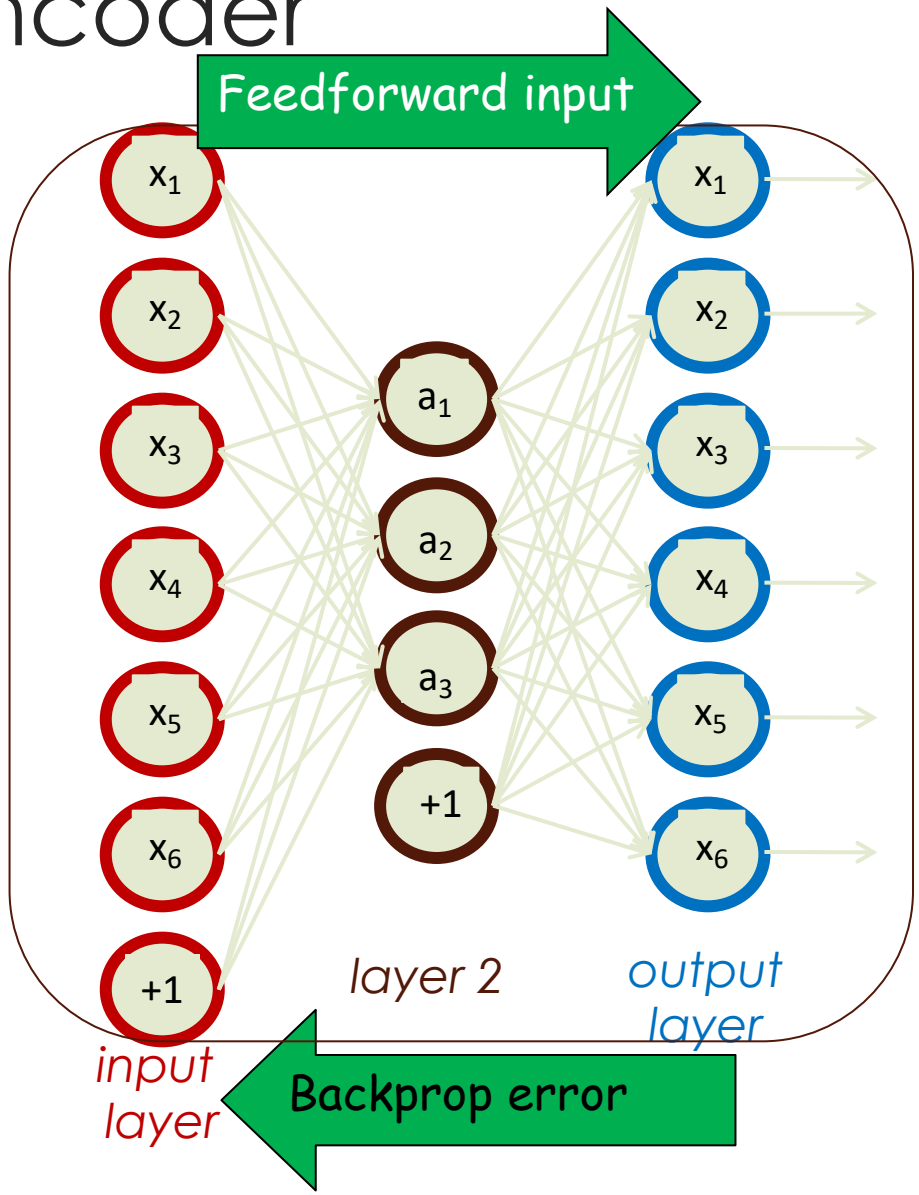
Use pretrained representations to feed a regular ANN with a smaller set of labelled data.

AutoEncoder

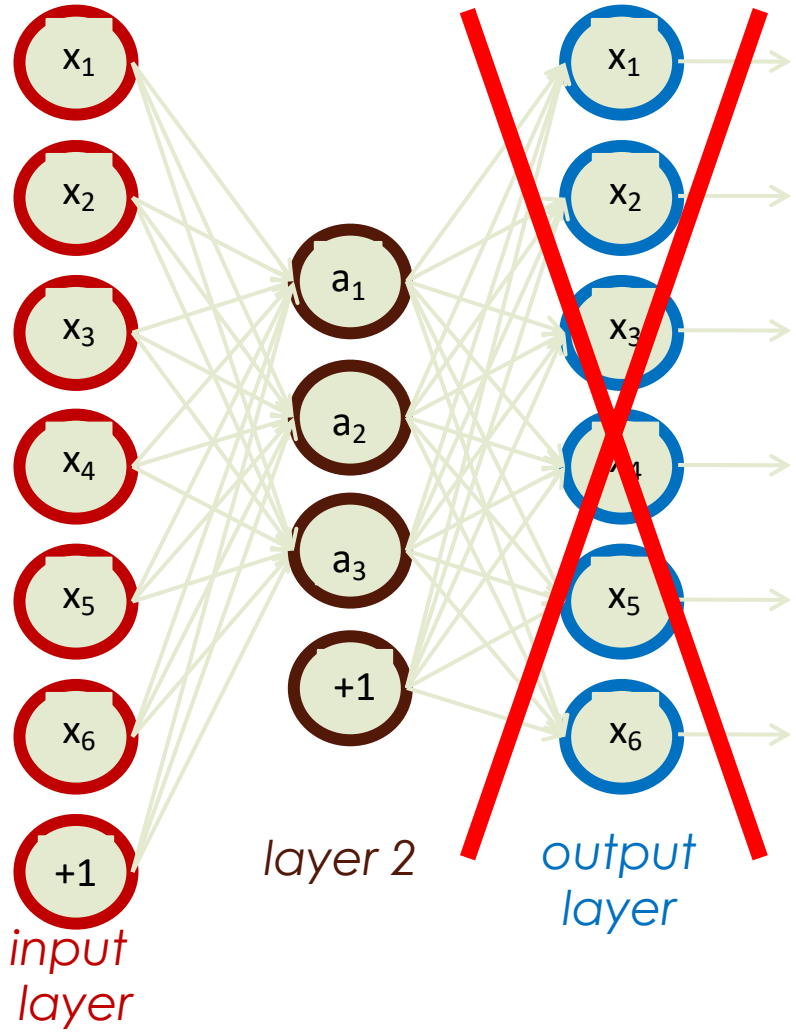


- The network is trained to output the input
- i.e. learn the identity function.
- Trivial… unless, we impose constraints:
  - Number of units in layer 2 < number of input units (learn compressed representation)
    OR
  - Constrain layer 2 to be sparse (i.e. many connections are "disabled")

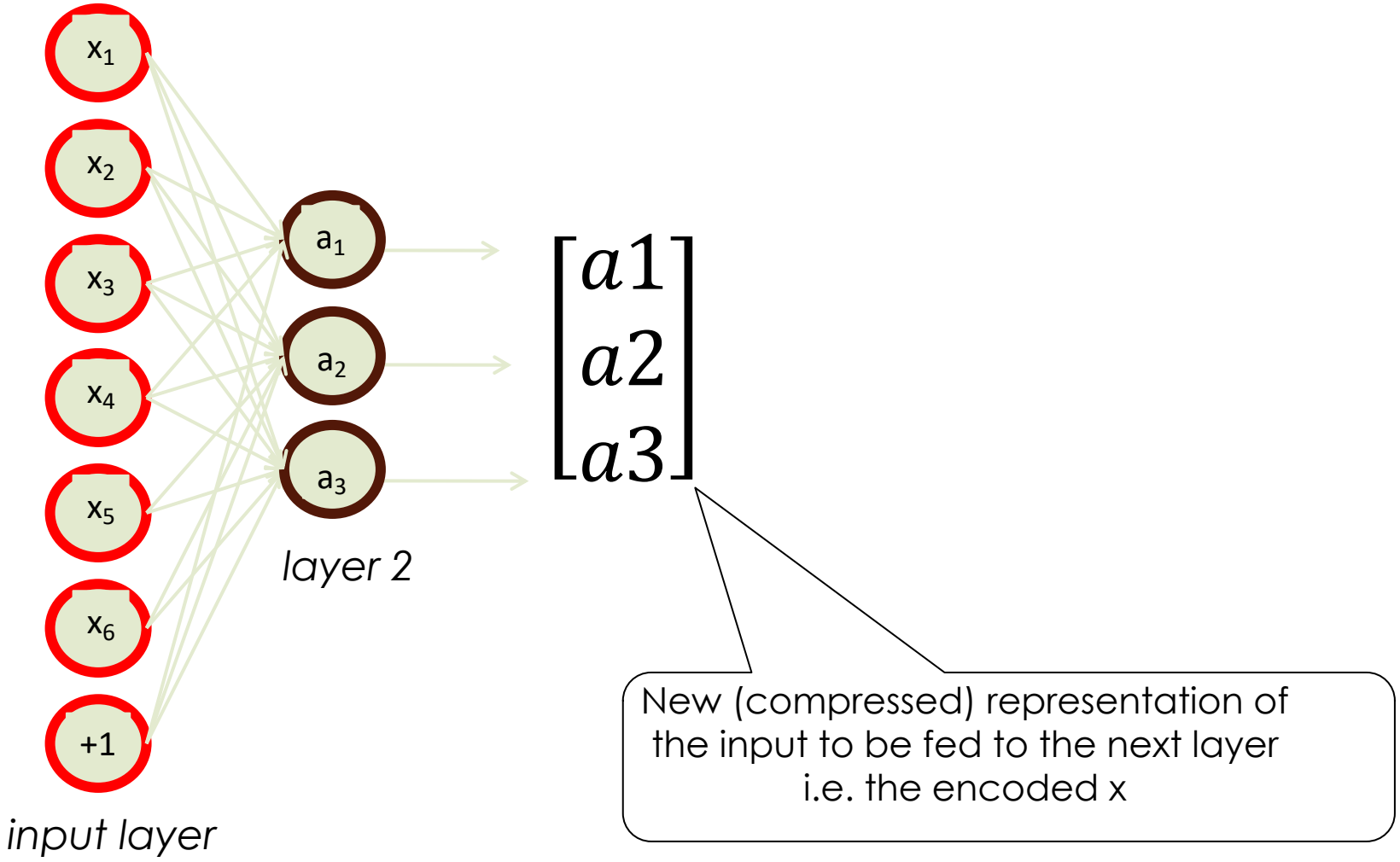
AutoEncoder



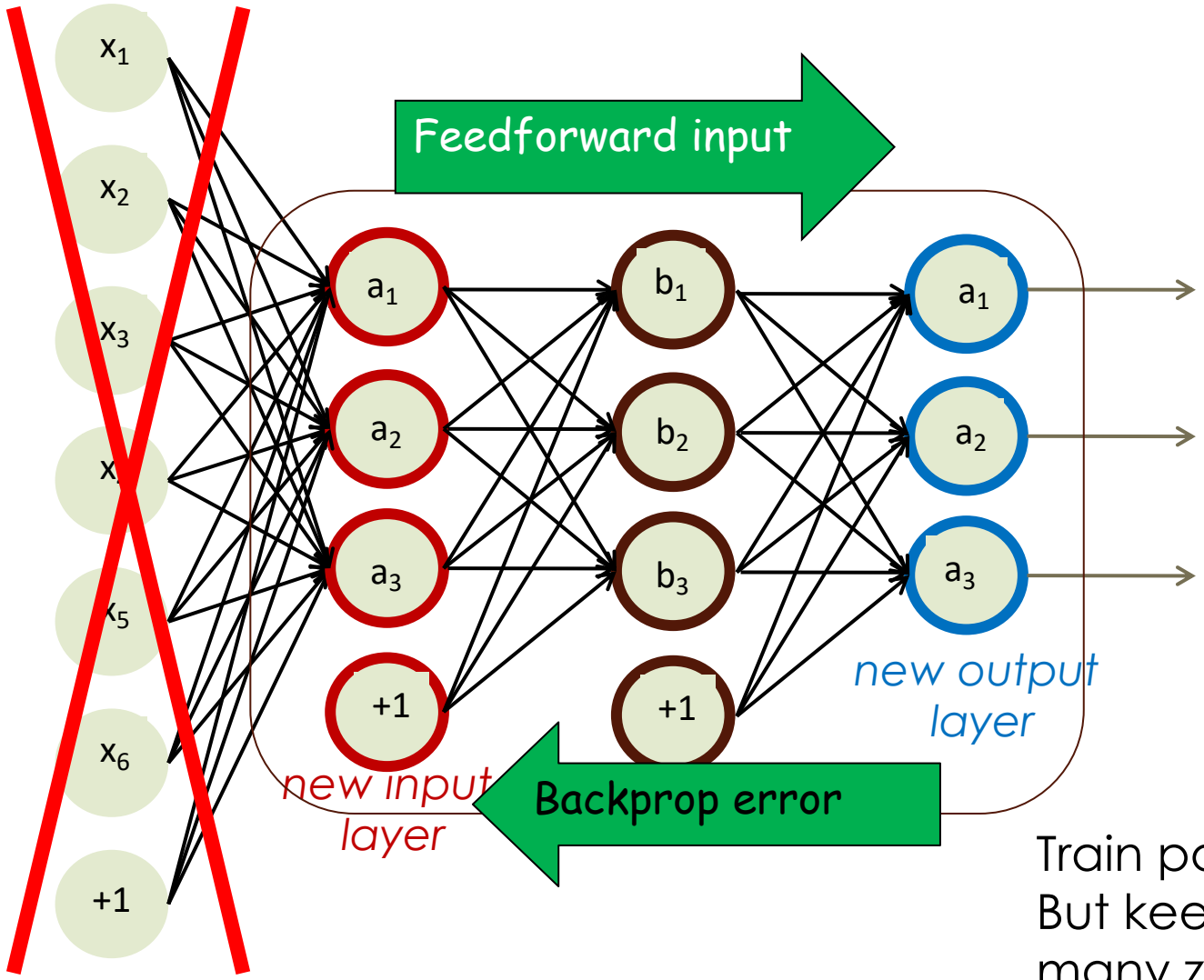
AutoEncoder



AutoEncoder

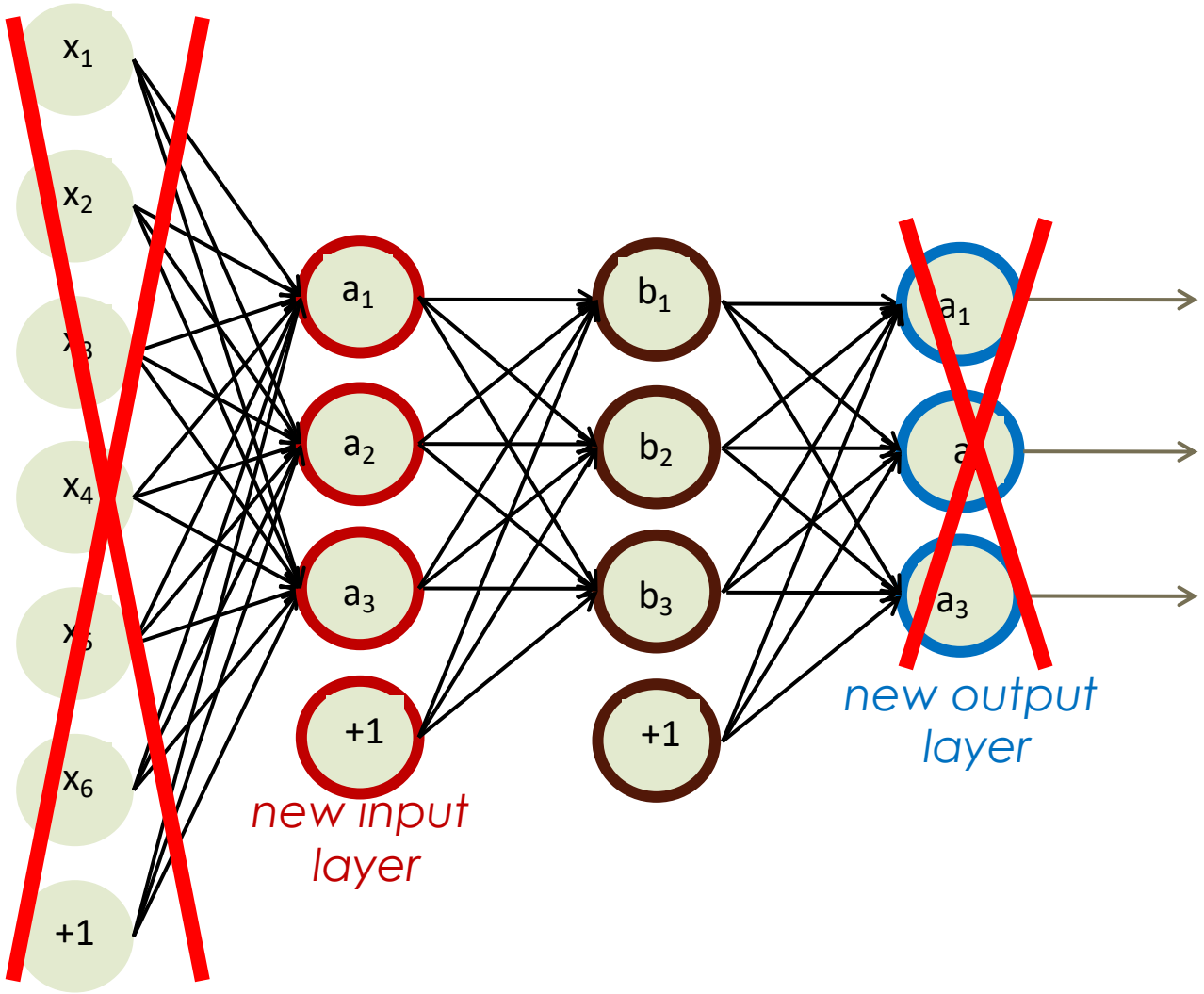


AutoEncoder

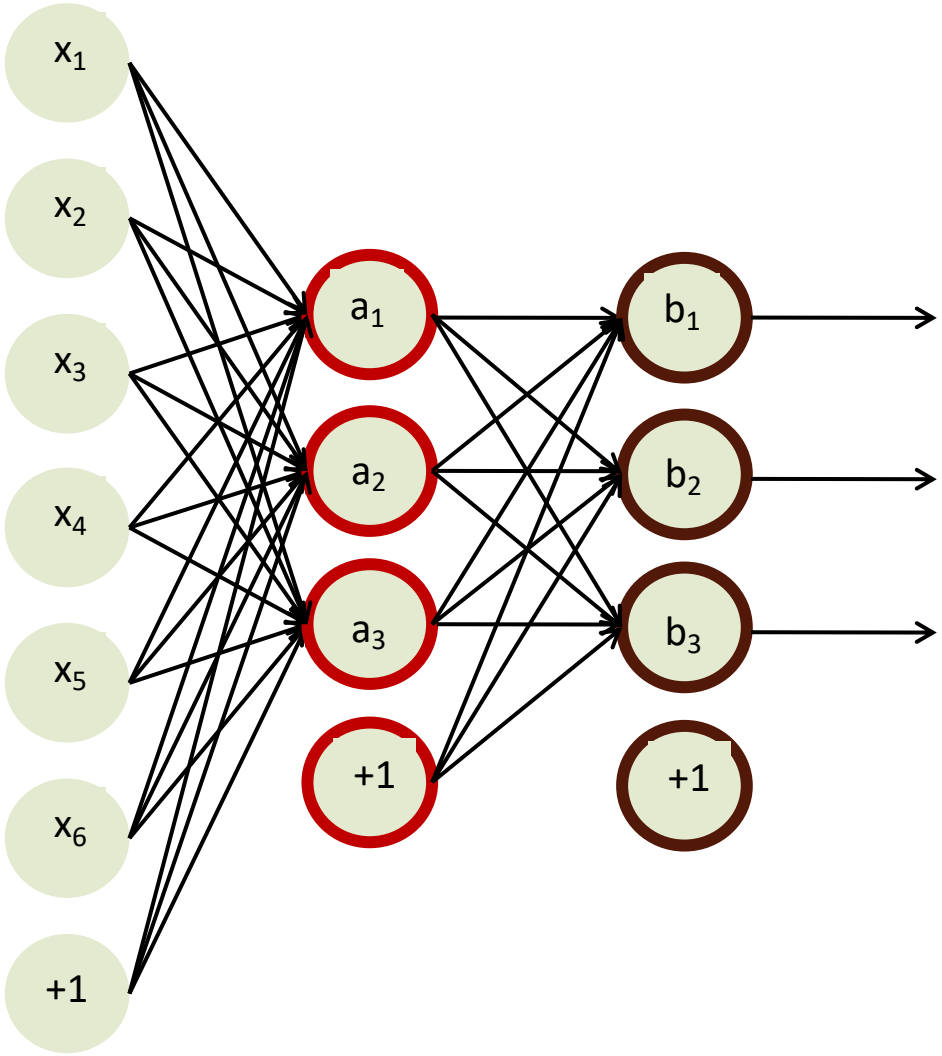


Train parameters (weights)
But keep b_i's sparse (ie. many zeros).

AutoEncoder



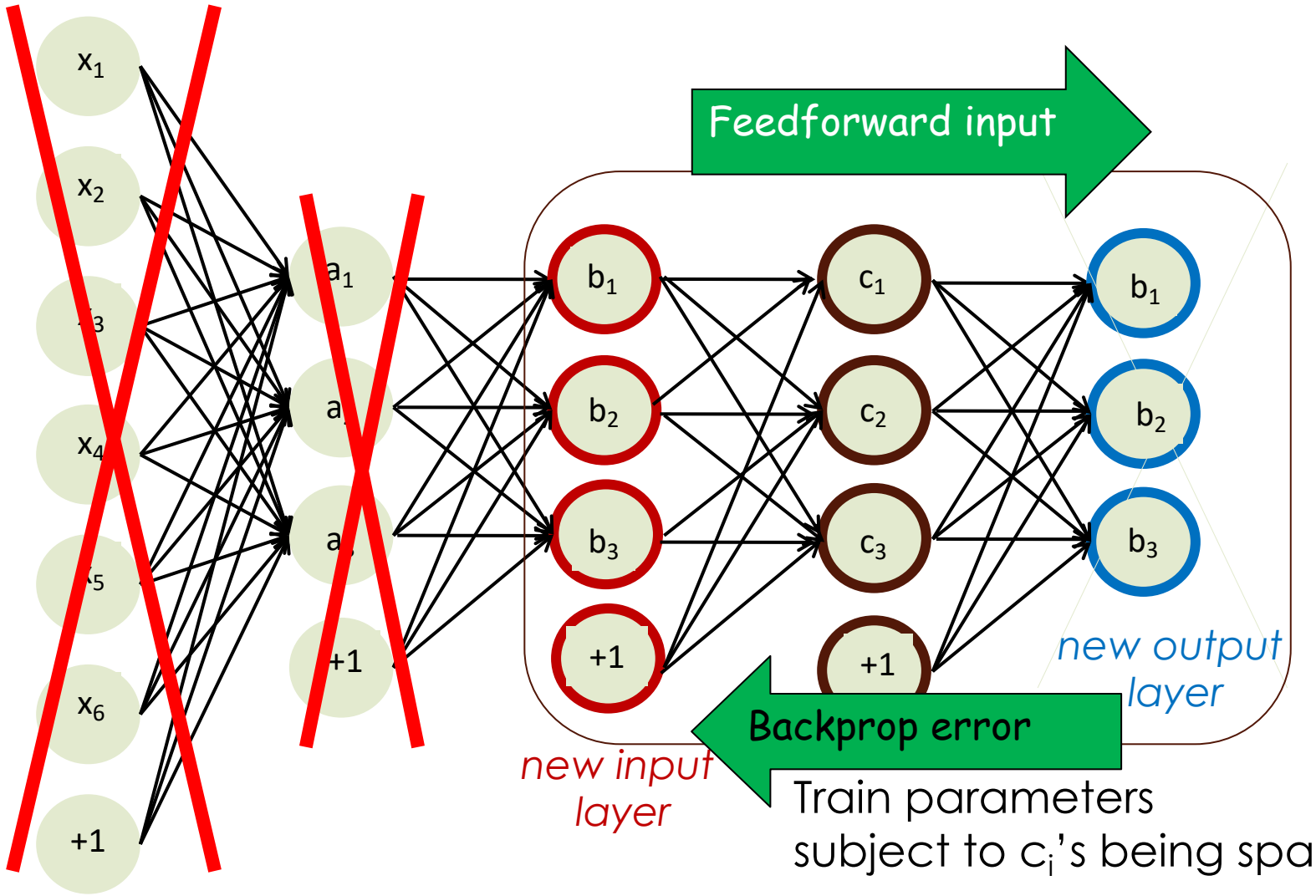
AutoEncoder



$$\begin{bmatrix} b1 \\ b2 \\ b3 \end{bmatrix}$$

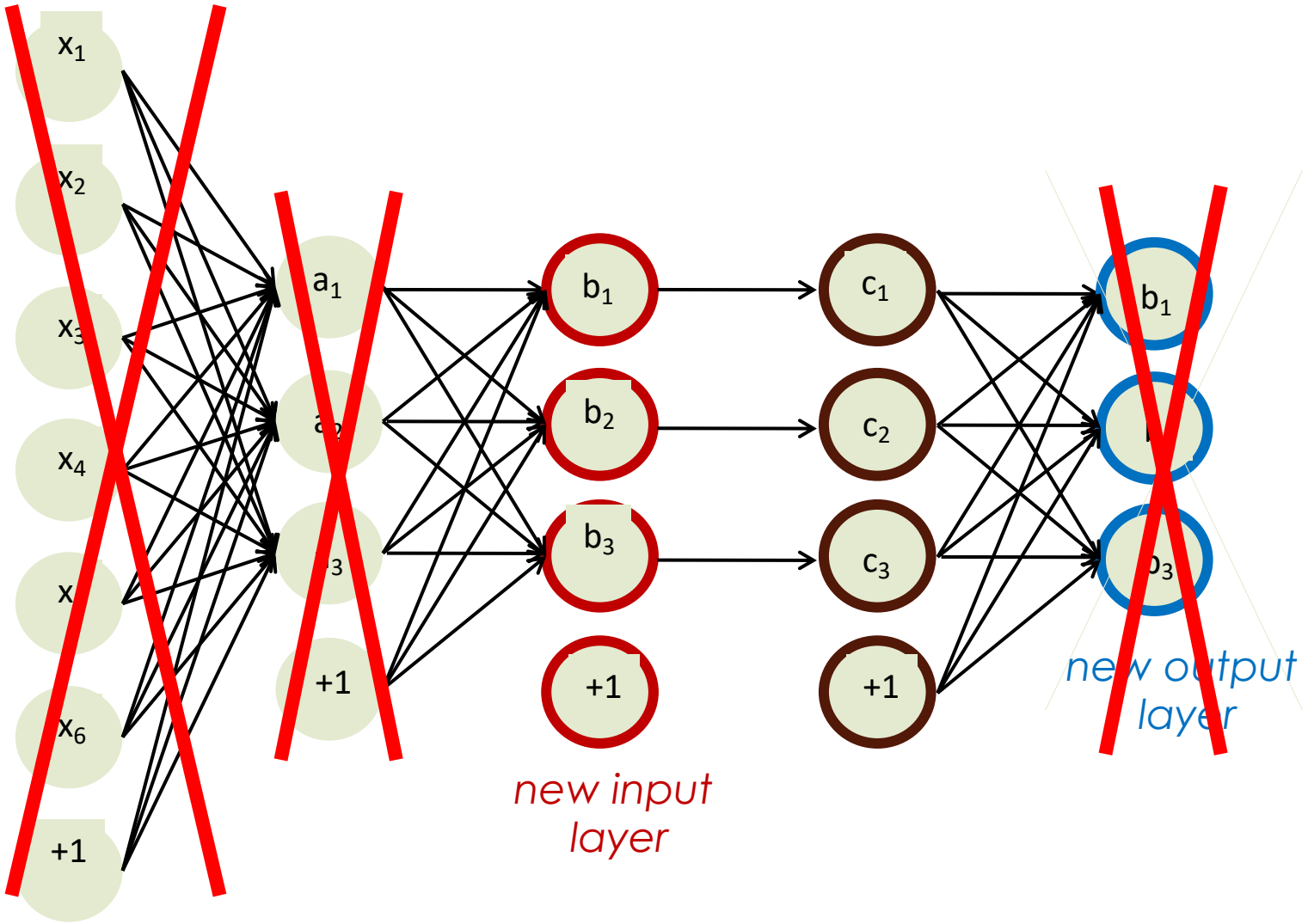
New representation for the input to the next layer i.e. the encoded a

AutoEncoder

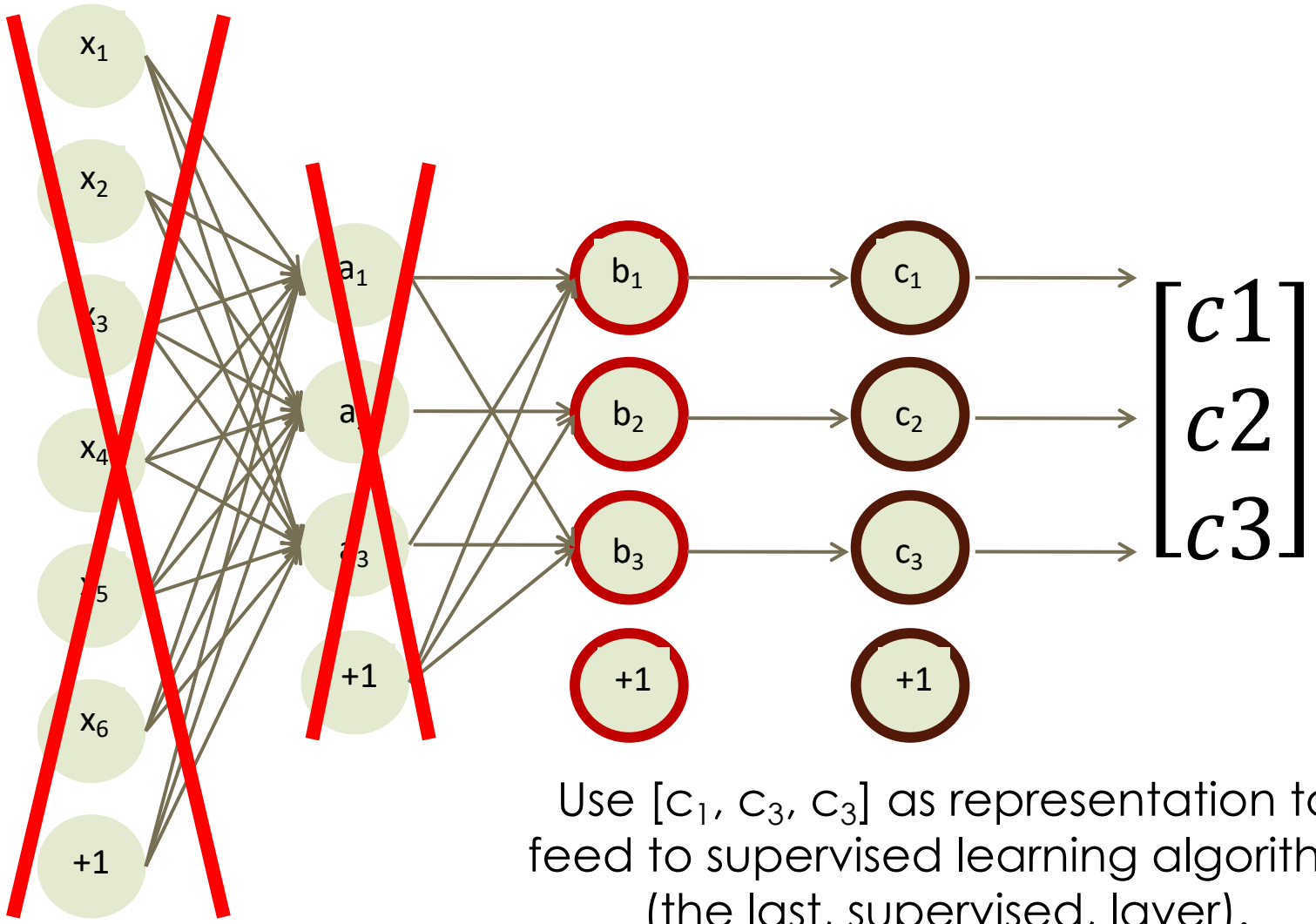


Train parameters subject to c_i's being sparse.

AutoEncoder

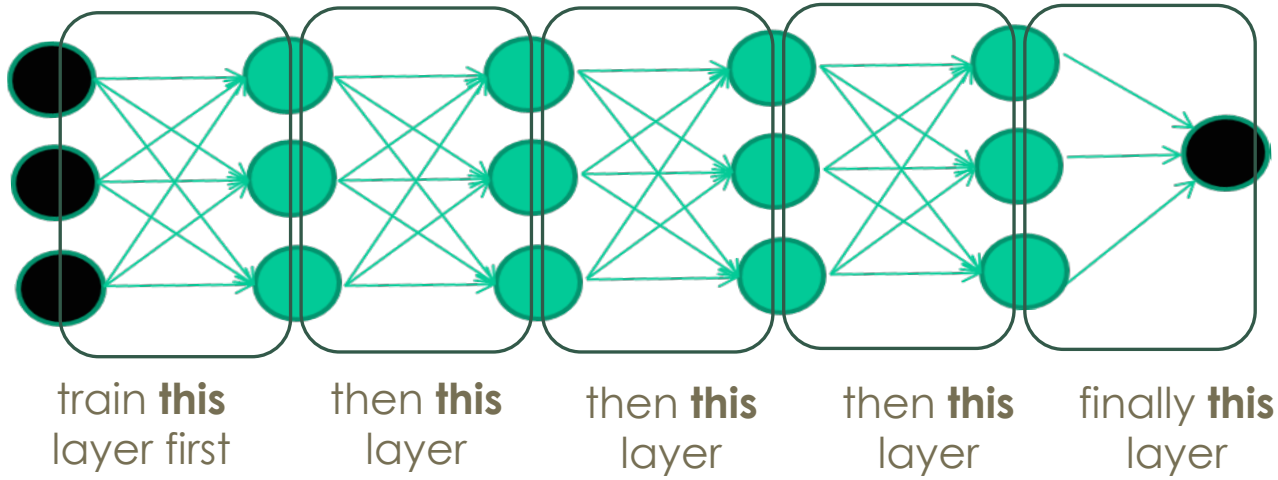


AutoEncoder



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

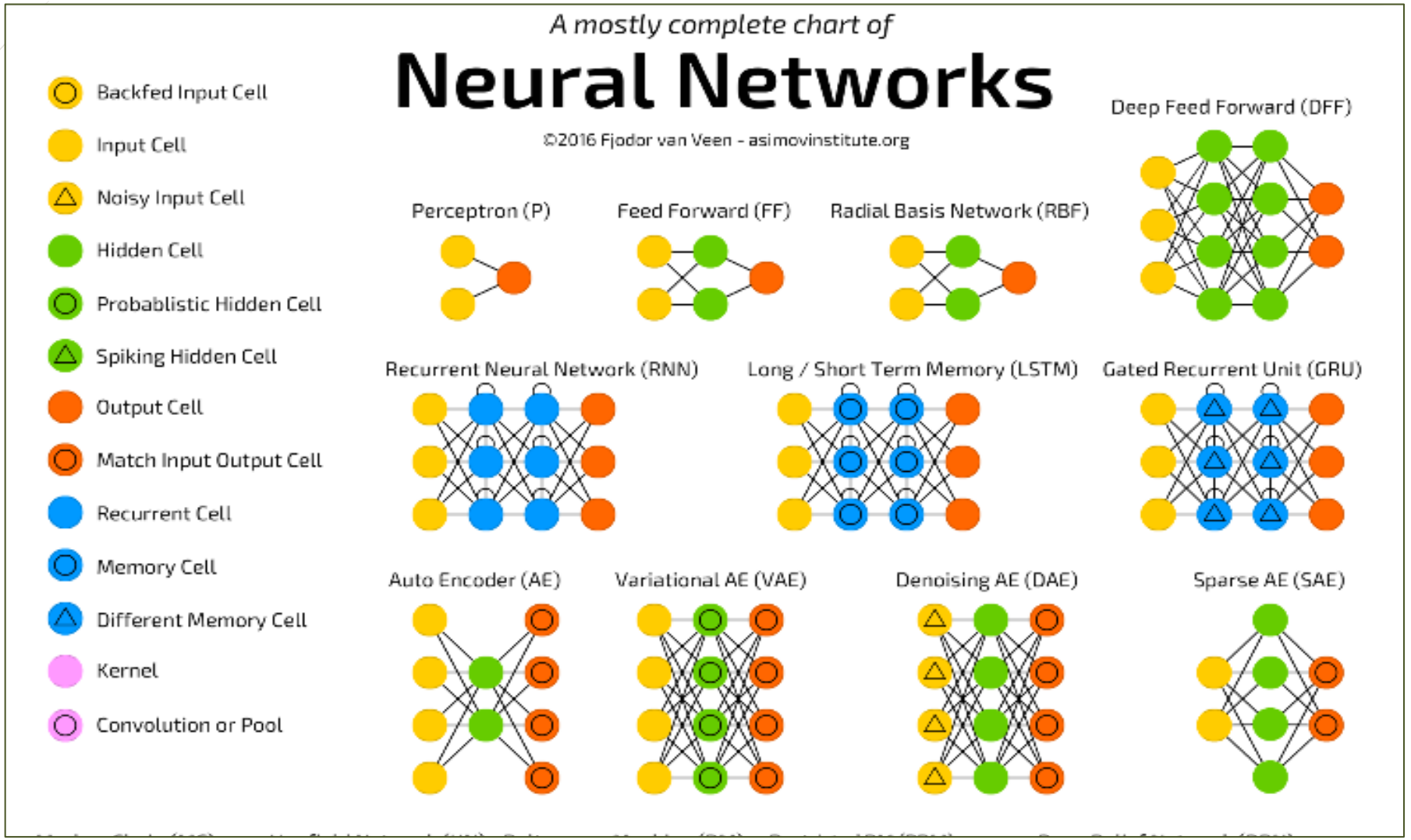
Training a Deep Neural Network



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

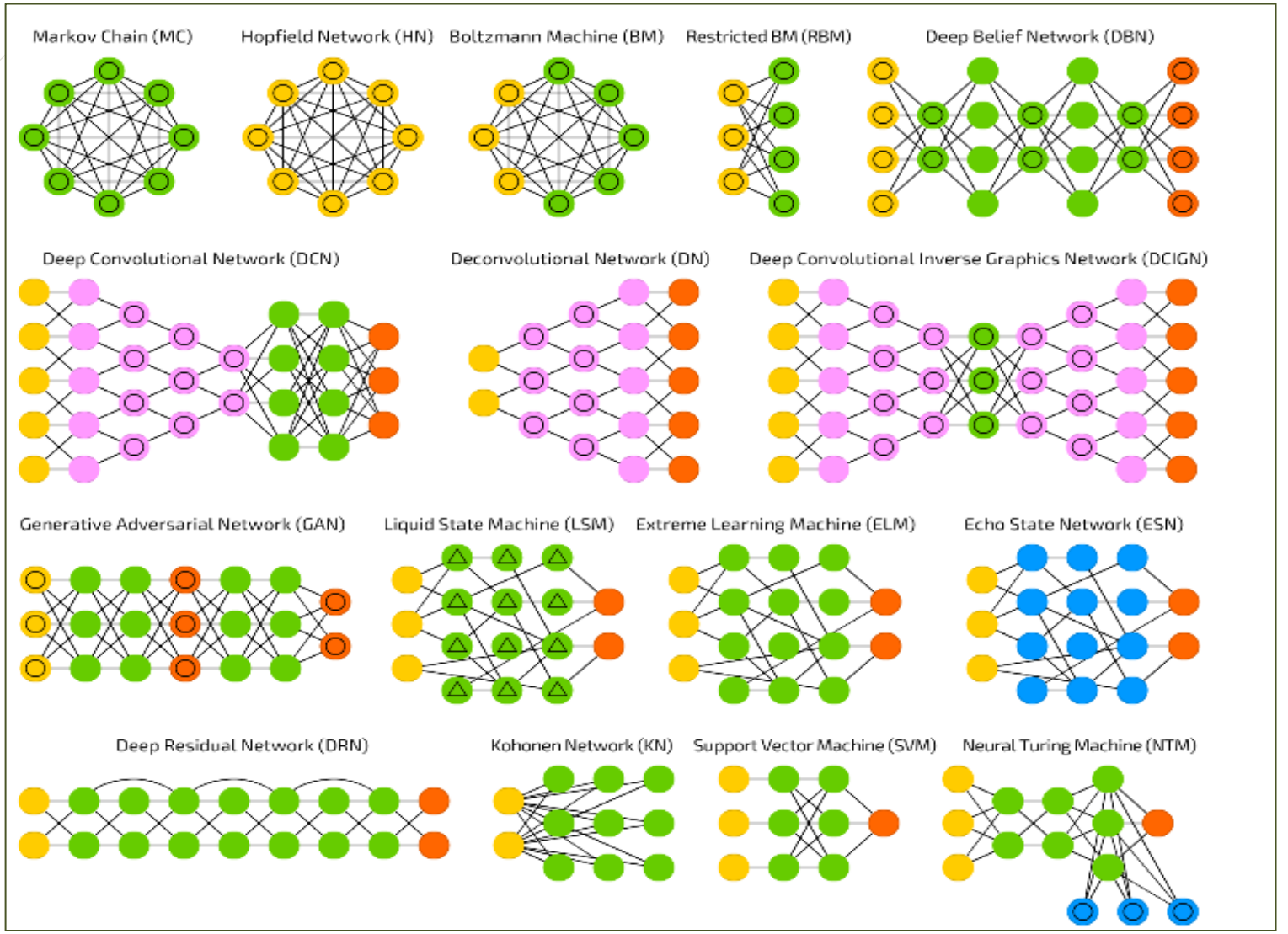
Use pretrained representations to feed a regular ANN with a smaller set of labelled data.

Types of Neural Networks



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

Types of Neural Networks



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

The End