	LOUISIANA STATE UNIVERSITY **College of Agriculture** **School of Plant, Environmental, and Soil Sciences** **HTP in Plant Breeding**	

Data mining and modeling

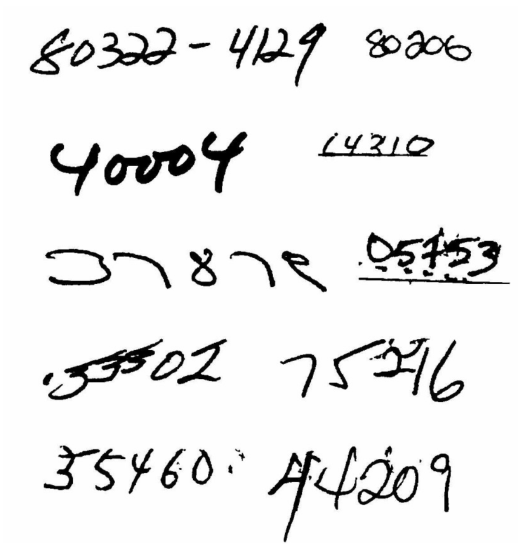
Prof. Roberto Fritsche-Neto

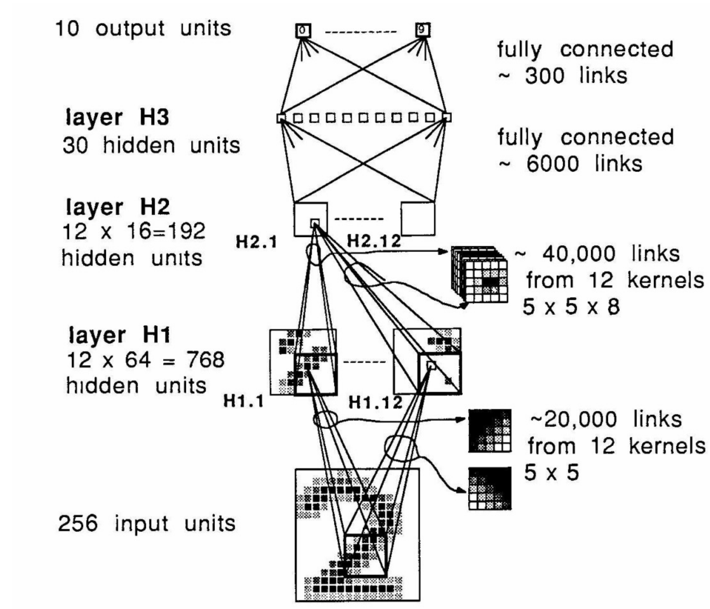
rfneto@agcenter.lsu.edu

Baton Rouge, March 20th, 2024

LeNet 1989

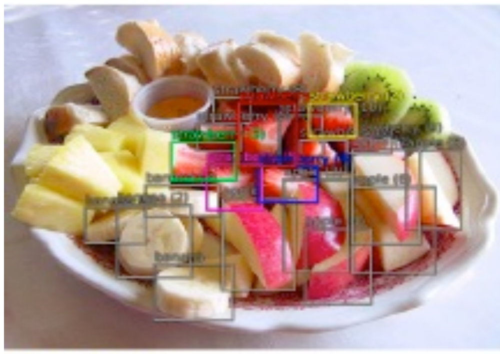
- **Recognize zip codes in US postal service**





- **Around 5% error rate. Good enough to be useful**

CNN: Detection



Groundtruth:
strawberry
strawberry (2)
strawberry (3)
strawberry (4)
strawberry (5)
strawberry (6)
strawberry (7)
strawberry (8)
strawberry (9)
strawberry (10)
apple
apple (2)
apple (3)

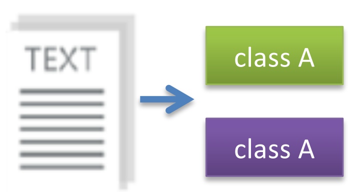
Groundtruth:
tv or monitor
tv or monitor (2)
tv or monitor (3)
person
remote control
remote control (2)

Sermanet, CVPR 2014

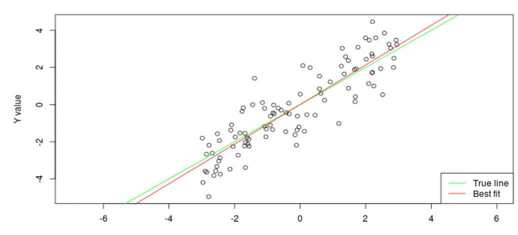
Types of Learning

Supervised: Learning with a labeled training set
Example: seed *classification* based on colors and shape

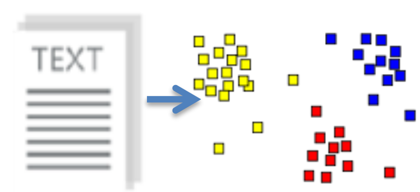
Unsupervised: Discover patterns in unlabeled data
Example: *cluster* images based on color components



Classification

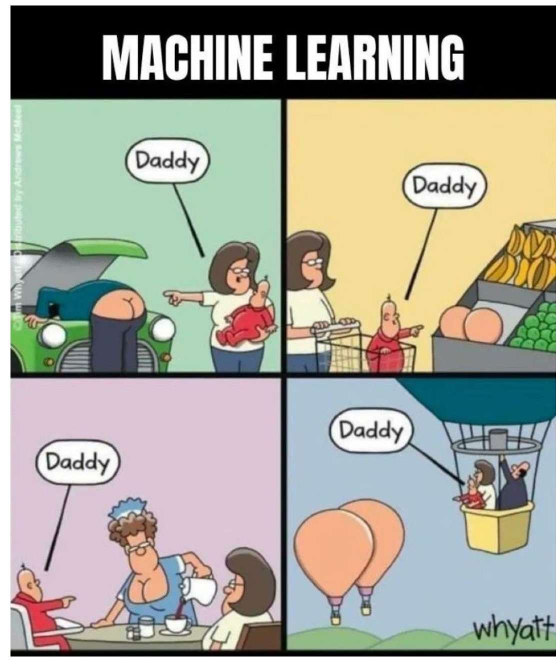


Regression



Clustering

The Curse of Learning



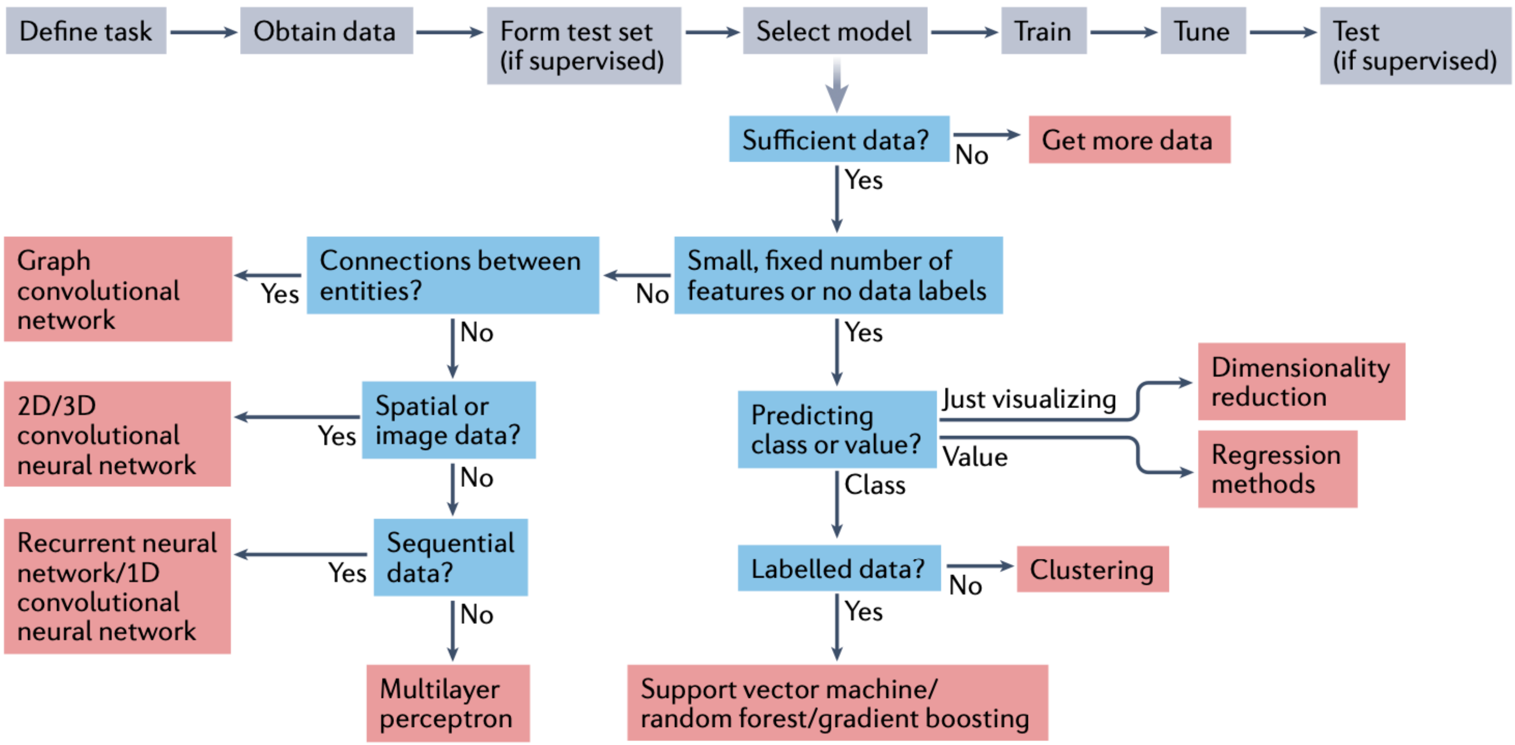
A guide to machine learning for biologists

Joe G. Greener, Shaun M. Kandathil, Lewis Moffat & David T. Jones

Nature Reviews Molecular Cell Biology **23**, 40–55 (2022) | Cite this article

https://www.nature.com/articles/s41580-021-00407-0

The Way of Learning



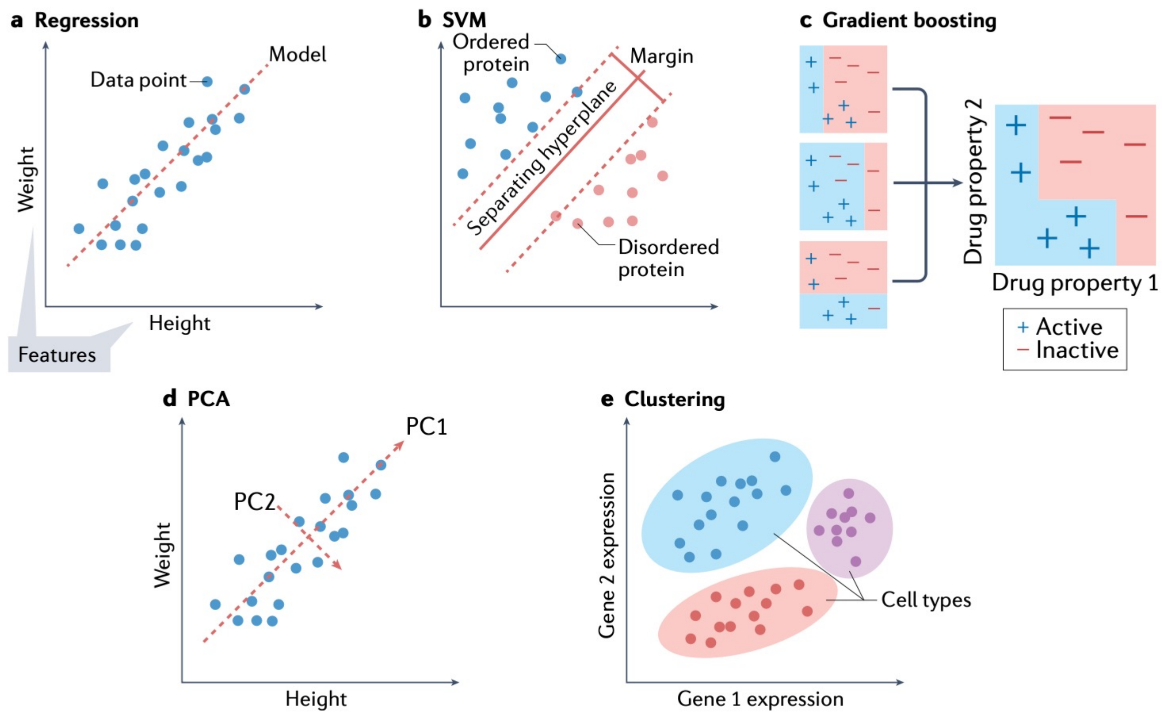
A guide to machine learning for biologists

Joe G. Greener, Shaun M. Kandathil, Lewis Moffat & David T. Jones

Nature Reviews Molecular Cell Biology **23**, 40–55 (2022) | Cite this article

https://www.nature.com/articles/s41580-021-00407-0

One size does not fit all



A guide to machine learning for biologists

Joe G. Greener, Shaun M. Kandathil, Lewis Moffat & David T. Jones

Nature Reviews Molecular Cell Biology **23**, 40–55 (2022) | Cite this article

https://www.nature.com/articles/s41580-021-00407-0

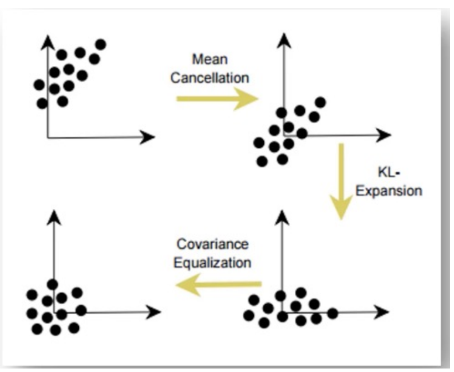
Data structure for supervised learning

- **Classification**

Var1	Var2	Var3	Var4	Var5	Var6	Var7	Var8	Var9	Var10	Var-target
1.2	2.3	4.21	22	1.5	0.2	0.77	2.9	3.4	0.8	**A**
1.5	2.99	5.21	23	41.5	1.2	1.77	2.6	1.4	3.8	**A**
1.9	4.3	4.44	11	8.5	0.1	9.77	6.9	0.4	6.8	**B**

- **Regression**

Var1	Var2	Var3	Var4	Var5	Var6	Var7	Var8	Var9	Var10	Var-target
0.2	2.3	4.23	22	1.5	0.2	0.76	2.9	3.4	0.8	**3.12**
1.5	2.99	5.21	23	41.5	1.2	1.77	2.6	1.4	3.8	**4.88**
1.9	4.3	4.44	11	8.5	0.1	9.79	6.9	0.4	6.8	**9.73**

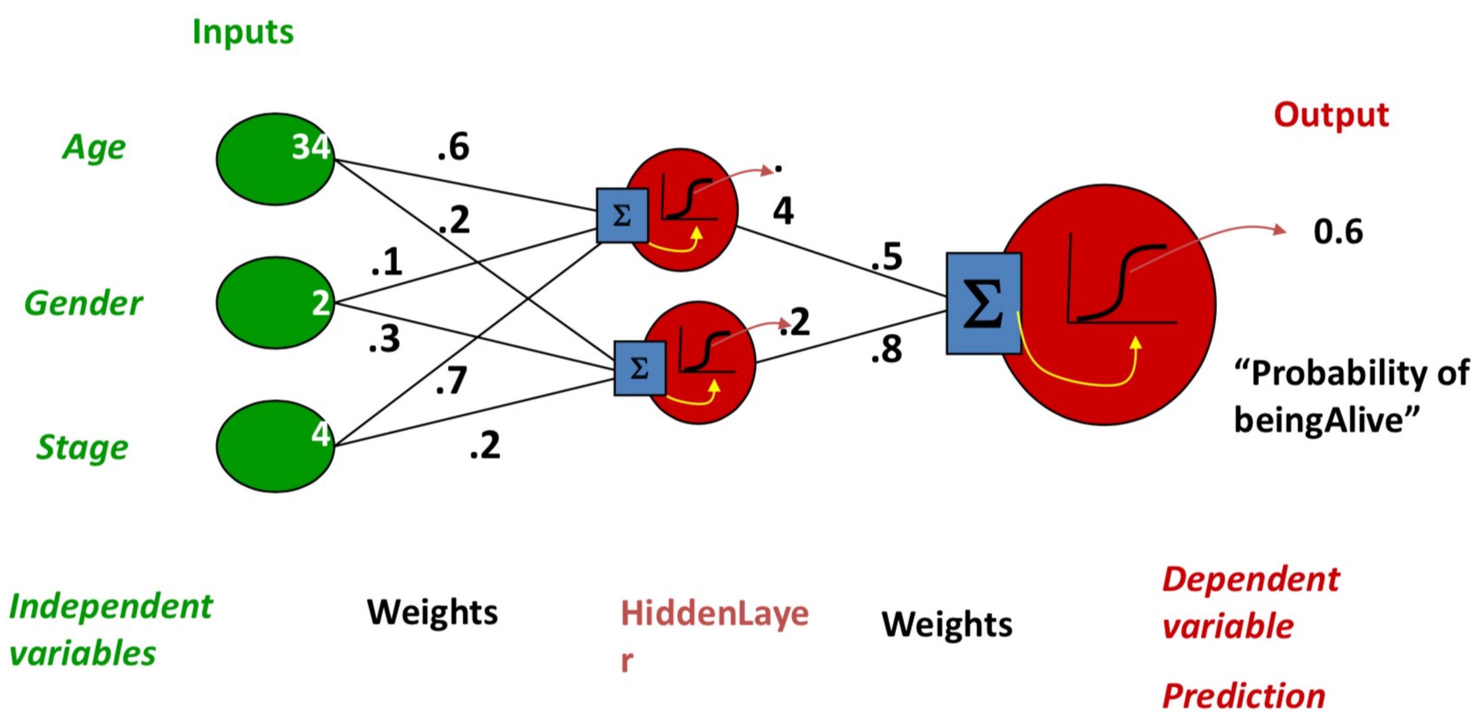


- **Scale / normalize/standardize the data**
- **Balanced classes**
- **Imputation**
- **Create dummy variables**
- **Replace variables that don't bring any information – zip code -> lat and long**
- **Feature selection - *Reduce dimensionality to help to avoid overfitting***
- ***ML models do not work properly with collinearity***

Artificial Neural Networks (ANN)

Many layers of staked multiple regressions

The regression type depends on the activation function



Algorithm for learning ANN

- **Initialize the weights ($w_0, w_1, \ldots, w_k$)**

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

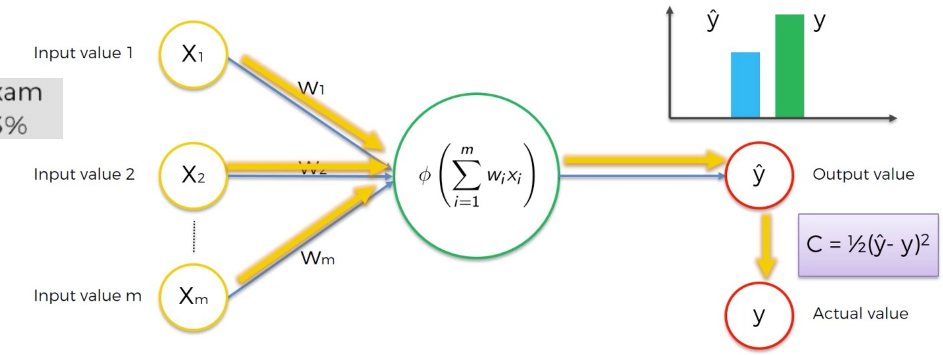
  Error function:

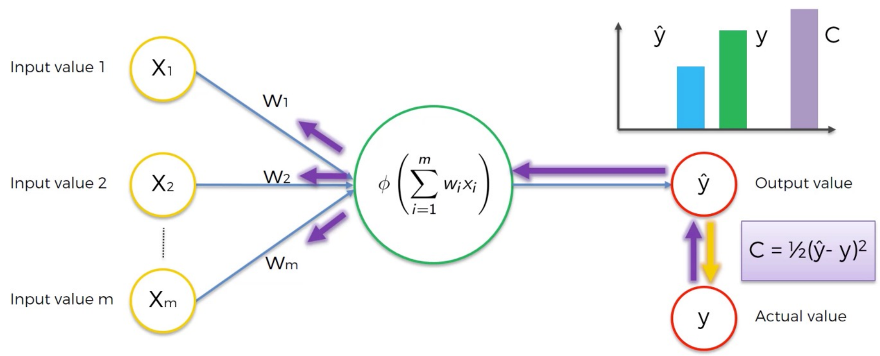
$$E = \sum_i \left[Y_i - f(w_i, X_i) \right]^2$$

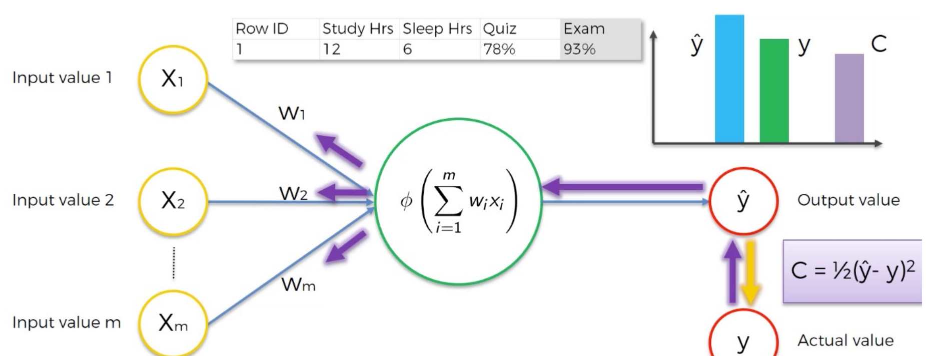
- **Find the weights w_i's that minimize the error function**

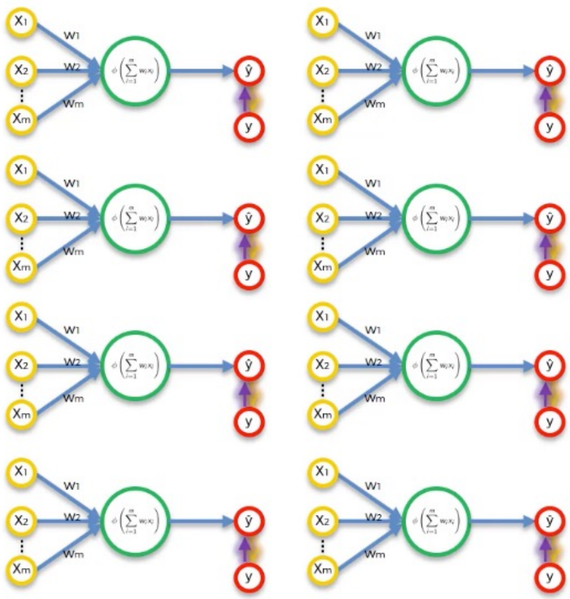
  e.g., gradient descent, backpropagation algorithm

Row ID	Study Hrs	Sleep Hrs	Quiz	Exam
1	12	6	78%	93%









Row ID	Study Hrs	Sleep Hrs	Quiz	Exam
1	12	6	78%	93%
2	22	6.5	24%	68%
3	115	4	100%	95%
4	31	9	67%	75%
5	0	10	58%	51%
6	5	8	78%	60%
7	92	6	82%	89%
8	57	8	91%	97%

$$C = \sum \tfrac{1}{2}(\hat{y} - y)^2$$

Algorithm for learning ANN

STEP 1: Randomly initialise the weights to small numbers close to 0 (but not 0).

STEP 2: Input the first observation of your dataset in the input layer, each feature in one input node.

STEP 3: Forward-Propagation: from left to right, the neurons are activated in a way that the impact of each neuron's activation is limited by the weights. Propagate the activations until getting the predicted result y.

STEP 4: Compare the predicted result to the actual result. Measure the generated error.

STEP 5: Back-Propagation: from right to left, the error is back-propagated. Update the weights according to how much they are responsible for the error. The learning rate decides by how much we update the weights.

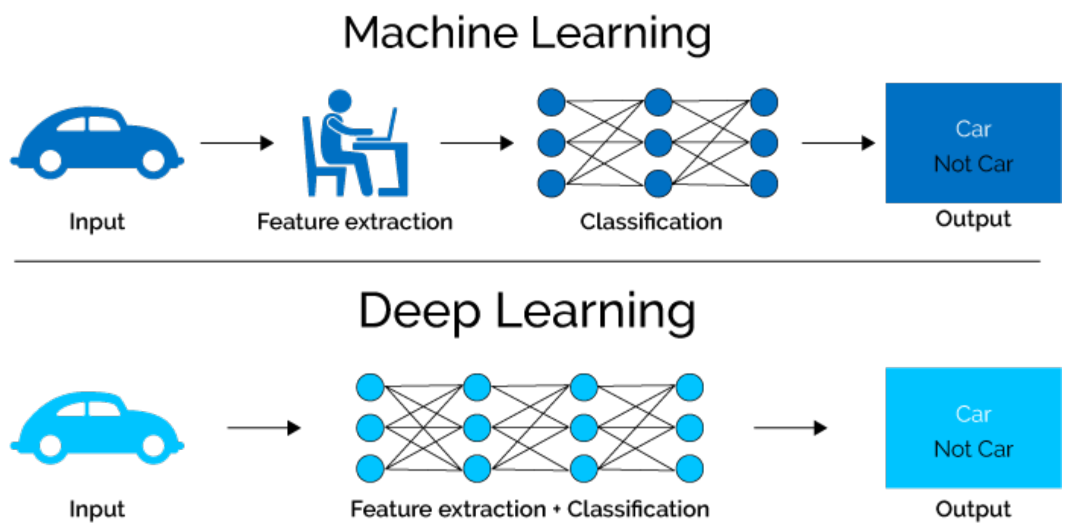
STEP 6: Repeat Steps 1 to 5 and update the weights after each observation (Reinforcement Learning). Or:
Repeat Steps 1 to 5 but update the weights only after a batch of observations (Batch Learning).

STEP 7: When the whole training set passed through the ANN, that makes an epoch. Redo more epochs.

Batch: sample (fold) used for training
Epoch: iteration

ML *vs.* Deep Learning

- DL is a machine learning subfield of learning representations of data exceptionally 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

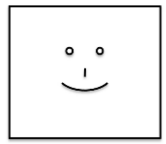


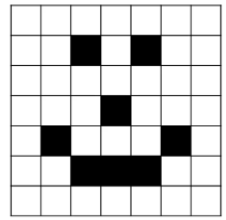
STEP 1: Convolution

STEP 2: Max Pooling

STEP 3: Flattening

STEP 4: Full Connection





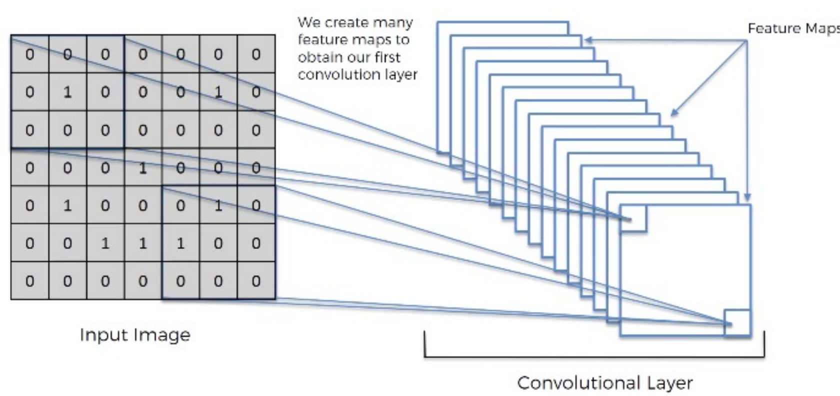
$$\begin{bmatrix}0&0&0&0&0&0&0\\0&1&0&0&0&1&0\\0&0&0&0&0&0&0\\0&0&0&1&0&0&0\\0&1&0&0&0&1&0\\0&0&1&1&1&0&0\\0&0&0&0&0&0&0\end{bmatrix}$$

$$\underbrace{\begin{bmatrix}0&0&0&0&0&0&0\\0&1&0&0&0&1&0\\0&0&0&0&0&0&0\\0&0&0&1&0&0&0\\0&1&0&0&0&1&0\\0&0&1&1&1&0&0\\0&0&0&0&0&0&0\end{bmatrix}}_{\text{Input Image}} \otimes \underbrace{\begin{bmatrix}0&0&1\\1&0&0\\0&1&1\end{bmatrix}}_{\text{Feature Detector}} = \underbrace{\begin{bmatrix}0&&&&\\&&&&\\&&&&\\&&&&\\&&&&\end{bmatrix}}_{\text{Feature Map}}$$

$$\underbrace{\begin{bmatrix}0&0&0&0&0&0&0\\0&1&0&0&0&1&0\\0&0&0&0&0&0&0\\0&0&0&1&0&0&0\\0&1&0&0&0&1&0\\0&0&1&1&1&0&0\\0&0&0&0&0&0&0\end{bmatrix}}_{\text{Input Image}} \otimes \underbrace{\begin{bmatrix}0&0&1\\1&0&0\\0&1&1\end{bmatrix}}_{\text{Feature Detector}} = \underbrace{\begin{bmatrix}0&1&&&\\&&&&\\&&&&\\&&&&\\&&&&\end{bmatrix}}_{\text{Feature Map}}$$

$$\underbrace{\begin{bmatrix}0&0&0&0&0&0&0\\0&1&0&0&0&1&0\\0&0&0&0&0&0&0\\0&0&0&1&0&0&0\\0&1&0&0&0&1&0\\0&0&1&1&1&0&0\\0&0&0&0&0&0&0\end{bmatrix}}_{\text{Input Image}} \otimes \underbrace{\begin{bmatrix}0&0&1\\1&0&0\\0&1&1\end{bmatrix}}_{\text{Feature Detector}} = \underbrace{\begin{bmatrix}0&1&0&0&0\\0&1&1&1&0\\1&0&1&2&1\\1&4&2&1&0\\0&0&1&2&1\end{bmatrix}}_{\text{Feature Map}}$$

$$* \begin{bmatrix}1&0&-1\\2&0&-2\\1&0&-1\end{bmatrix}$$



STEP 1: Convolution

STEP 2: Max Pooling

STEP 3: Flattening

STEP 4: Full Connection

0	1	0	0	0
0	1	1	1	0
1	0	1	2	1
1	4	2	1	0
0	0	1	2	1

Feature Map

Max Pooling →

1		

Pooled Feature Map

0	1	0	0	0
0	1	1	1	0
1	0	1	2	1
1	4	2	1	0
0	0	1	2	1

Feature Map

Max Pooling →

1	1	

Pooled Feature Map

0	1	0	0	0
0	1	1	1	0
1	0	1	2	1
1	4	2	1	0
0	0	1	2	1

Feature Map

Max Pooling →

1	1	0
4	2	1

Pooled Feature Map

0	1	0	0	0
0	1	1	1	0
1	0	1	2	1
1	4	2	1	0
0	0	1	2	1

Feature Map

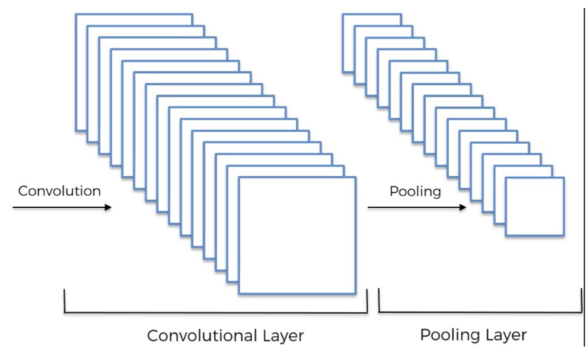
Max Pooling →

1	1	0
4	2	1
0	2	1

Pooled Feature Map

0	0	0	0	0	0	0
0	1	0	0	0	1	0
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	1	0	0	0	1	0
0	0	1	1	1	0	0
0	0	0	0	0	0	0

Input Image

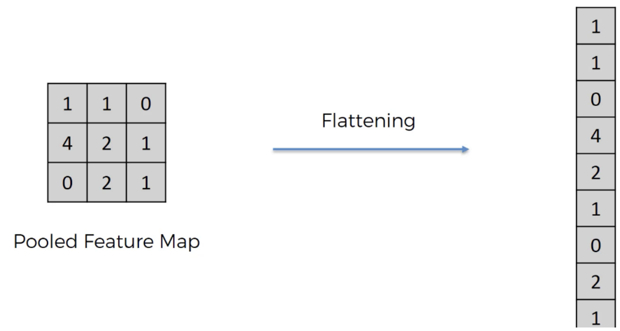


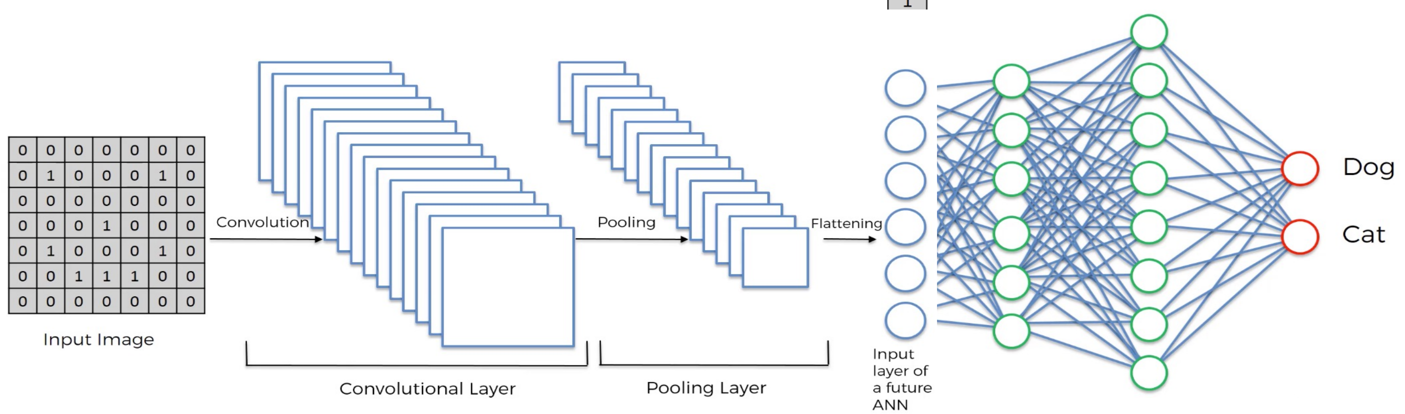
STEP 1: Convolution

STEP 2: Max Pooling

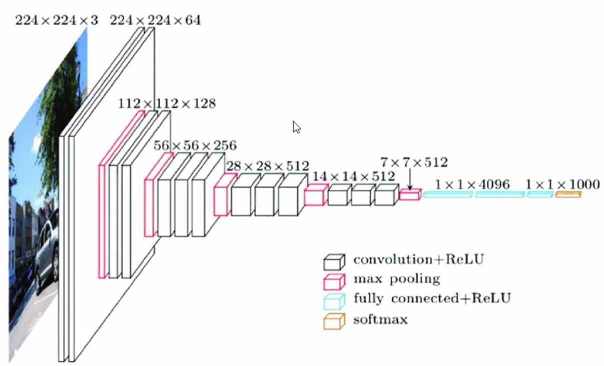
STEP 3: Flattening

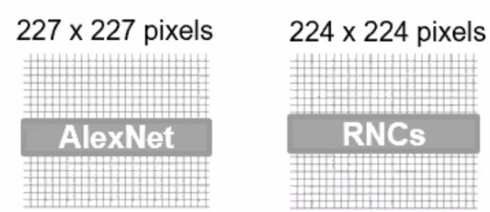
STEP 4: Full Connection



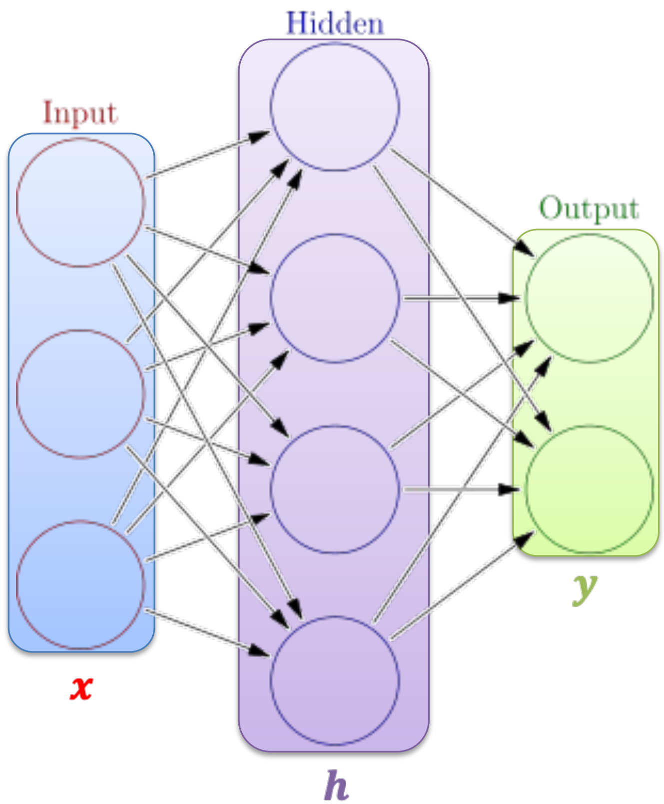


- AlexNet
- VGG-16
- VGG-19
- GoogLeNet
- ResNet-18
- ResNet-50
- ResNet-101





Learnable parameters



Weights

$$h = \sigma(W_1 x + b_1)$$

$$y = \sigma(W_2 h + b_2)$$

Activation functions

How do we train?

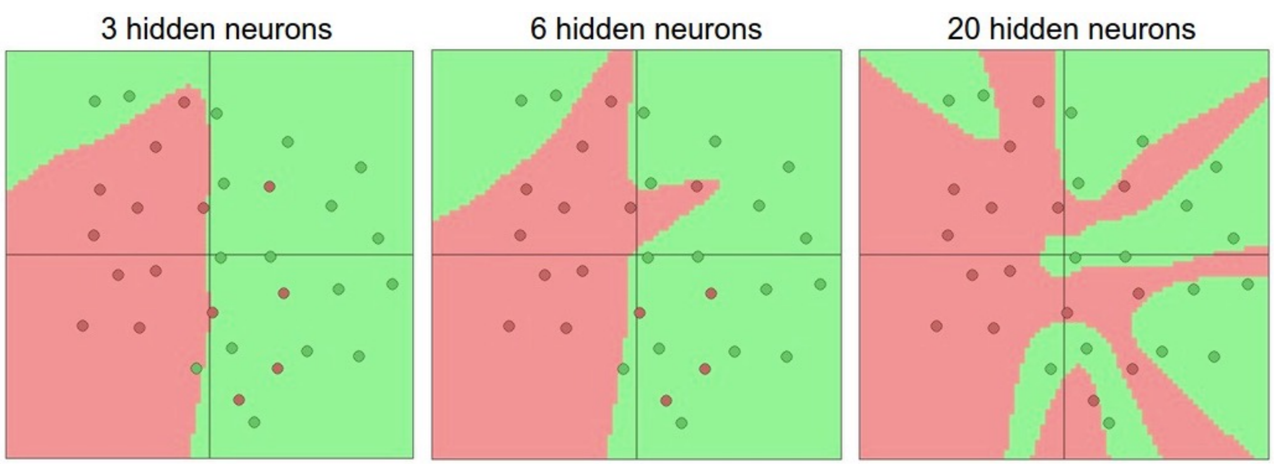
4 + 2 = 6 neurons (not counting inputs)
[3 x 4] + [4 x 2] = 20 weights
4 + 2 = 6 biases

---

26 learnable **parameters**

Number of neurons and layers

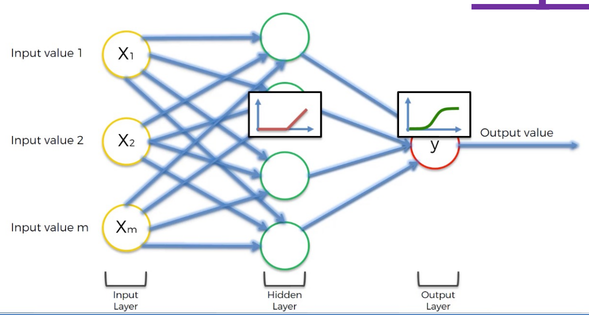
- **The number of neurons in the input layer is the number of explanatory variables**
- **Non-linearities needed to learn complex (non-linear) representations of data**
- **Otherwise, the ANN would be just a linear function**



- **More hidden layers and neurons can approximate more complex functions**

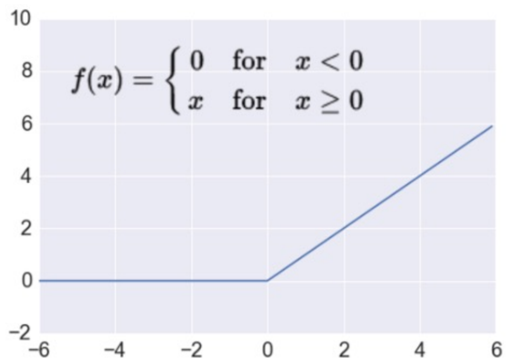
Activation functions

- The learned information extracted from the training data is stored and captured by the weight values of the connections between the layers
- The final output can be continuous, binary, ordinal, or count, which is controlled for the activation function in the output layer



Activation functions	g(z)	String	Use	Data
rectifier linear unit	Max(0, Z)	relu	Positive values	CONTINUOS BUT POSITIVE
sigmoid	$(1+e^{-z})^{-1}$	sigmoid	converts independent variables of near infinite range into simple probabilities between 0 and 1	BINARY
Tanh	tanh(z)	tanh	deal more easily with negative numbers	CONTINUOS FROM NEGATIVE TO POSITIVE
softmax $g(z_j) = \frac{\exp(z_j)}{1+\sum_{c=1}^{C}\exp(z_c)}, j=1,\ldots,C$		softmax	sigmoid activation function that handles multinomial labeling systems, that is, it is appropriate for categorical outcomes	MORE THAN TWO CLASSES

Activation: ReLU



http://adilmoujahid.com/images/activation.png

Takes a real-valued number and thresholds it at zero $f(x) = \max(0, x)$

$$R^n \rightarrow R^n_+$$

Trains much faster

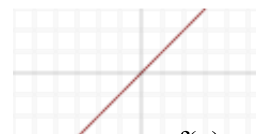
- **accelerates the convergence**

Less expensive operations

- **compared to sigmoid/tanh (exponentials etc.)**
- **implemented by simply thresholding a matrix at zero**

Loss or cost functions and output

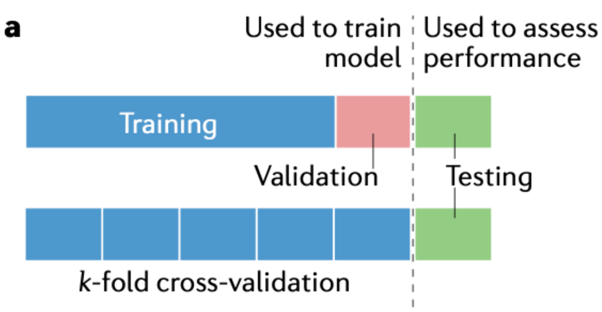
Measure the amount of 'disagreement' between the obtained and ideal outputs

	Classification	Regression
Training examples	R^n x {class_1, ..., class_n} (one-hot encoding)	R^n x R^m
Output Layer	Softmax [map R^n to a probability distribution] $P(y=j \mid \mathbf{x}) = \frac{e^{\mathbf{x}^\mathsf{T}\mathbf{w}_j}}{\sum_{k=1}^{K} e^{\mathbf{x}^\mathsf{T}\mathbf{w}_k}}$	Tanh, linear or Sigmoid  f(x)=x
Cost (loss) function	Cross-entropy $J(\theta) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{K}\left[y_k^{(i)}\log\hat{y}_k^{(i)} + \left(1-y_k^{(i)}\right)\log\left(1-\hat{y}_k^{(i)}\right)\right]$	Mean Squared Error $J(\theta) = \frac{1}{n}\sum_{i=1}^{n}\left(y^{(i)} - \hat{y}^{(i)}\right)^2$ Mean Absolute Error $J(\theta) = \frac{1}{n}\sum_{i=1}^{n}\left\lvert y^{(i)} - \hat{y}^{(i)}\right\rvert$

Most them weigh more the values near the average, but in breeding, our concern is the extreme values

Cross-validation and model selection

- **Randomly select two portions of the data to be used for training and validation**
- **Then, run the model on the test set to see how it performs**



Confusion Matrix

Classified As

Actual	Blue	Red
Blue	7	1
Red	0	5

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + TN + FN}$$

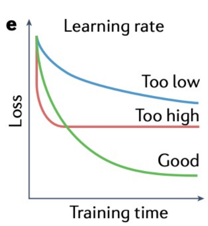
$$\text{Sensitivity} = \frac{TP}{TP + FN}$$

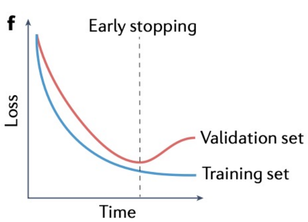
$$\text{Specificity} = \frac{TN}{TN + FP}$$

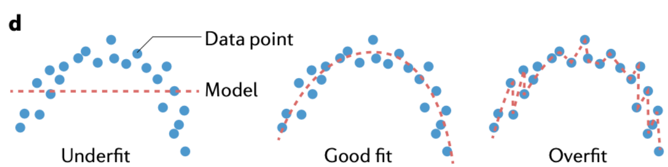
$$\text{FPR} = \frac{FP}{FP + TN}$$

$$\text{F1 score} = 2 \times \frac{\text{sensitivity} \times \text{precision}}{\text{sensitivity} + \text{precision}}$$

$$\text{AUC} = \frac{1}{2}\left(\frac{TP}{TP + FN} + \frac{TN}{TN + FP}\right)$$

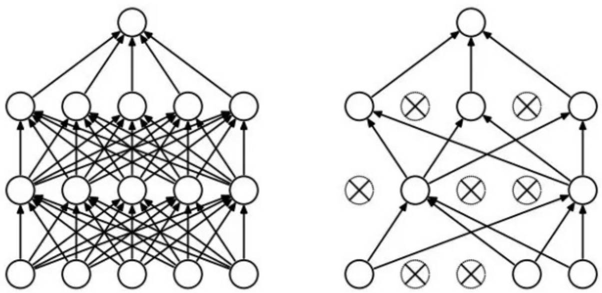






- **Learn the training data very well, even outliers (noise)**
- **Fail to generalize to new examples (test data)**

Regularization



Dropout

- Randomly drop units (along with their connections) during training
- Each unit retained with fixed probability p, independent of other units
- **Hyper-parameter** p to be chosen (tuned)

L2 = weight decay

- Regularization term that penalizes big weights, added to the objective
- Weight decay value determines how dominant regularization is during gradient computation
- Big weight decay coefficient → big penalty for big weights

$$J_{reg}(\theta) = J(\theta) + \lambda \sum_k \theta_k^2$$

Early-stopping

- Use validation error to decide when to stop training
- Stop when monitored quantity has not improved after n subsequent epochs
- n is called patience