September 2017

Introduction to Machine Learning

What is Machine Learning?

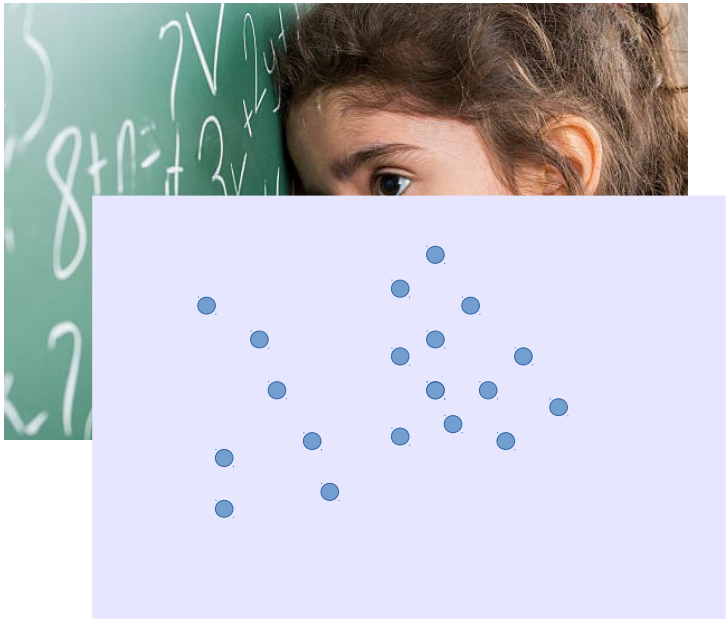
- A learning process allowing a machine to do something without specific programming
- Most machine learning that you hear about these days is *example-based*
  - Mathematically, an *example* is just a real-valued vector
- Given a series of *examples*, can the machine learn something about them and/or can it deduce something about new similar examples that it did not previously experience?
- We normally distinguish two phases of example-based machine learning
  - *Training*: learning from known examples
  - *Testing*: applying learned knowledge to say something about new examples

What is Machine Learning?

Unsupervised
(*unlabeled* examples)

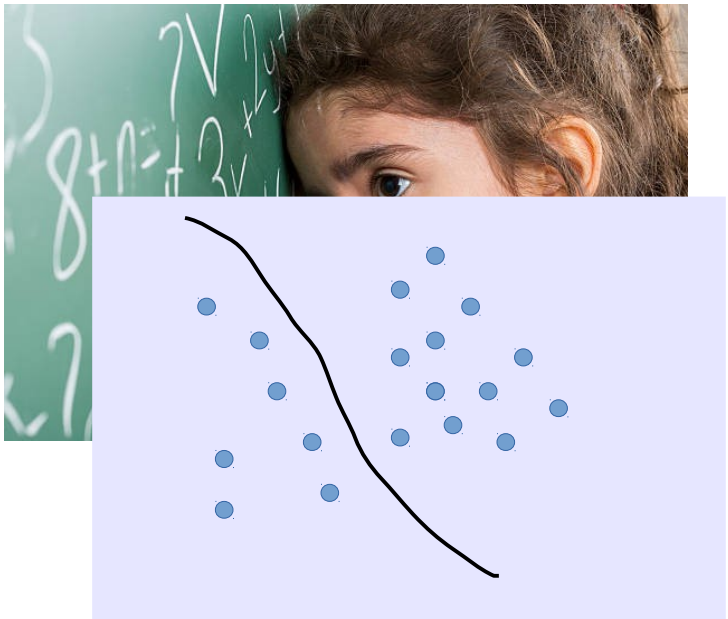
Supervised
(*labeled* examples)

What is Machine Learning?



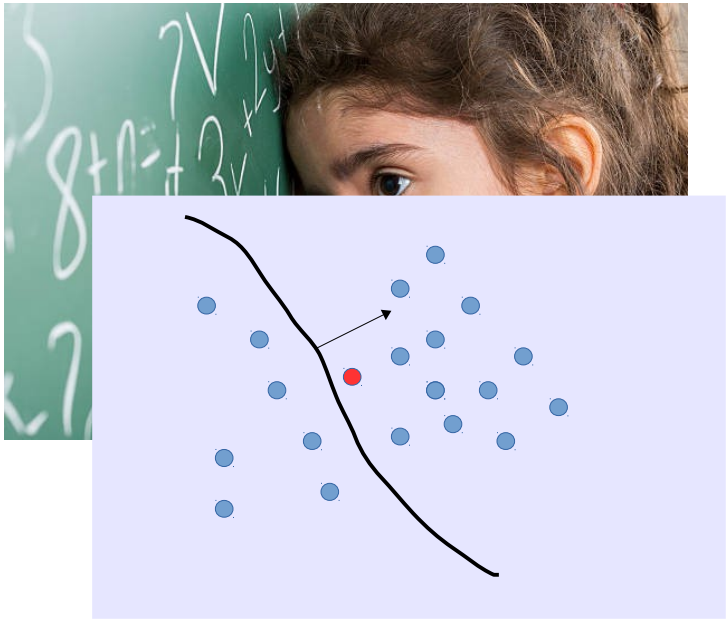
Supervised
(*labeled* examples)

What is Machine Learning?



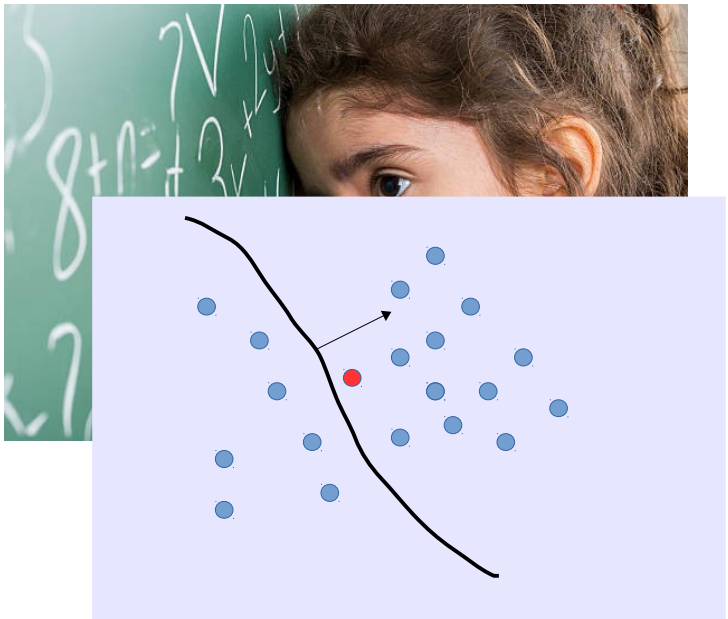
Supervised
(*labeled* examples)

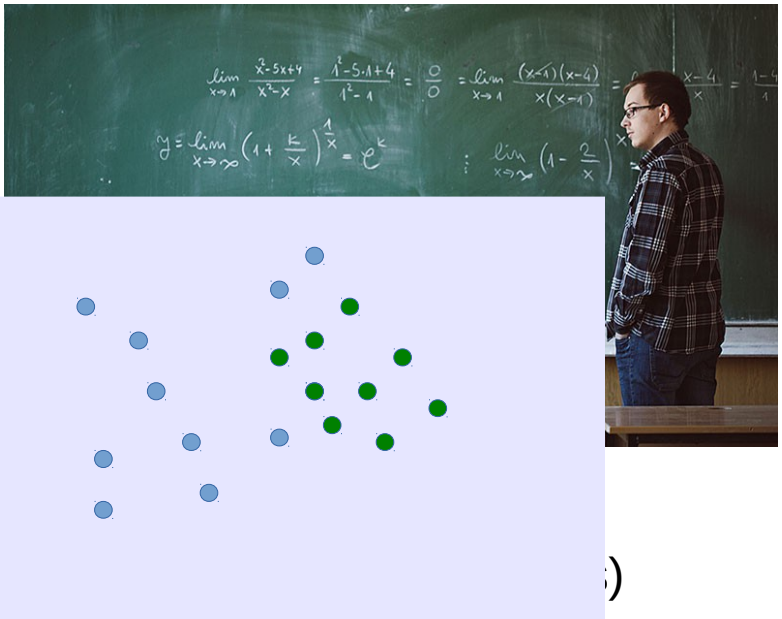
What is Machine Learning?



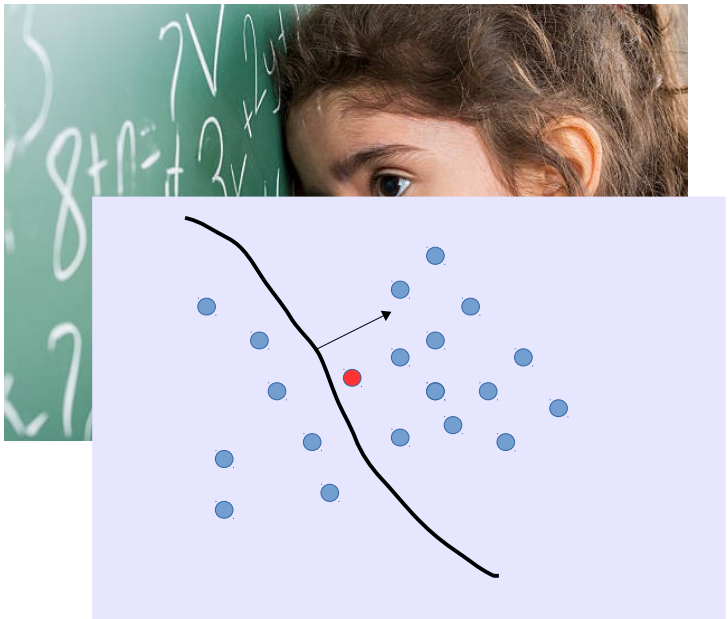
Supervised
(*labeled* examples)

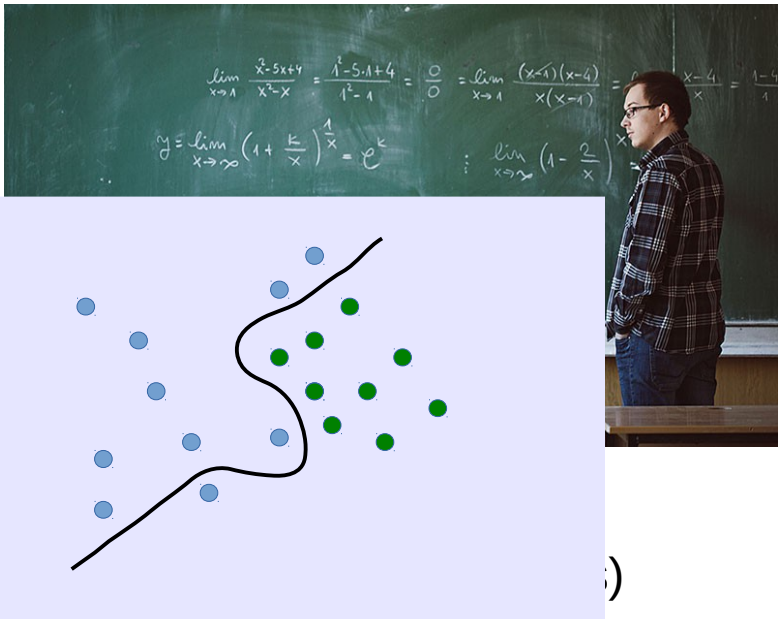
What is Machine Learning?



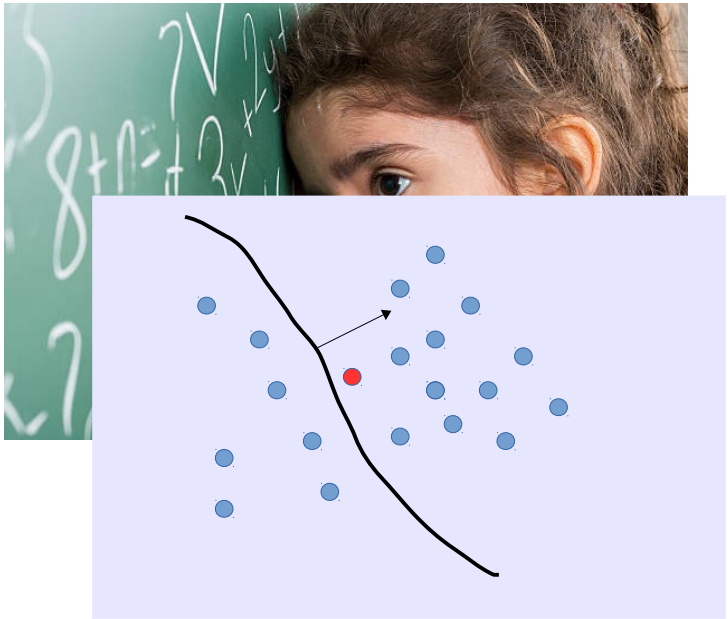


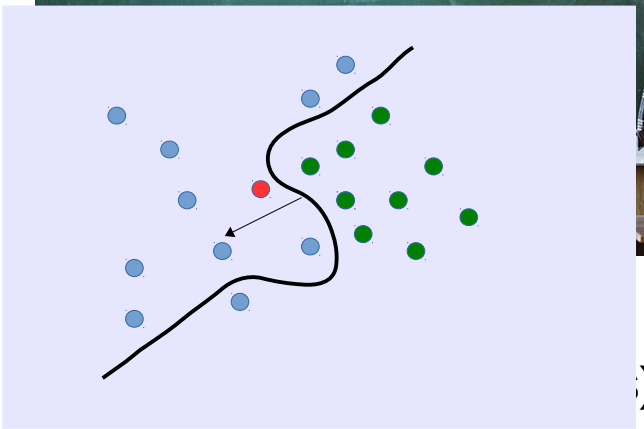
What is Machine Learning?





What is Machine Learning?





Some ML terms

- Some ML technologies
  - Neural networks and deep learning
  - Support vector machines
  - Linear regression
  - Decision trees and forests
  - Bayesian networks
  - Clustering
  - Boosting

- Software (free)
  - Scikit-learn (Python), SparkML (Spark), R
- Software (proprietary)
  - Watson ML, Google Prediction API, MS Azure Machine
- Software (deep learning)
  - TensorFlow, Caffe, Torch, Theano

What does a (supervised) ML problem look like?

- Numerous public data sets are available
  - UCI has a lot of classical sets used in the literature
  - Kaggle has regular challenges with data sets
- Generally each data set is contained in a table (CSV file)
  - Each row is an *example*
  - Columns are *features*
  - In supervised learning, one column is called the *target* which is the value you are trying to determine from looking at the features (other columns)

What does a (supervised) ML problem look like?

#Preg	Gl. con.	BP (dia)	Triceps	Serum insulin	BMI	Ped. Fn.	Age	Diabetic?
6	148	72	35	0	33.6	0.627	50	1
1	85	66	29	0	26.6	0.351	31	0
8	183	64	0	0	23.3	0.672	32	1
1	89	66	23	94	28.1	0.167	21	0
0	137	40	35	168	43.1	2.288	33	1
5	116	74	0	0	25.6	0.201	30	0
3	78	50	32	88	31	0.248	26	1

- Numerous public data sets ar
  - UCI has a lot of classi
  - Kaggle has regular ch
- Generally each data set is con
  - Each row is an *example*
  - Columns are *features*
  - In supervised learning, one column is called the *target* which is the value you are trying to determine from looking at the features (other columns)

Regression and classification

- When the target is a real number which has a continuous sense, then the problem is known as a *regression* problem. *e.g.*
  - Estimate the probability of an ad being clicked based on viewing or listening habits
  - Estimate a fair selling price of a house based on location, size, *etc.*

- When the target may only take discrete values and these values don't form a natural order (*i.e.* they represent different classes), the problem is known as a *classification* problem. *e.g.*
  - Determine if there is a cat in the image or not
  - Determine if a tweet is threatening or not
  - Determine a handwritten character (A-Z)

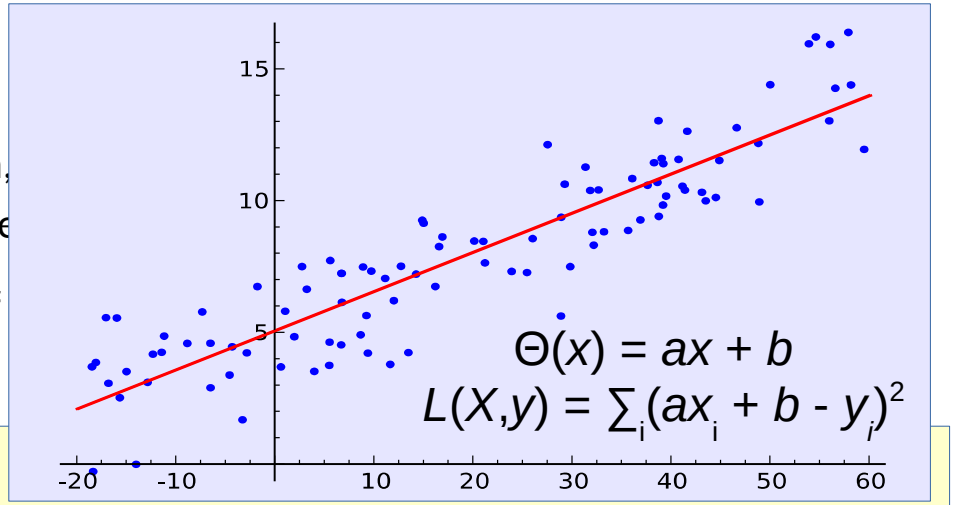
Training (learning from examples)

- Consider there are *N* examples on which to learn, and *M* features per example
- Let *X* be an *N* x *M* real-valued matrix and *y* be a real vector of size *N* (remember the CSV file)
  - Row *i* of *X* contains the *feature values* of example *i*
  - Element *i* of *y* is the *target* for example *i*

- Consider a *regression* problem
  - Produce a function $\theta : R^M \rightarrow R$ which takes an example X_i and produces an evaluation
  - Ideally, we would like to have $\theta(X_i)$ close to y_i for all examples *i*
  - In general, we define a *loss function* $L(\theta, X, y)$ to measure the distance from the ideal

Training (learning from examples)

- Consider there are *N* examples on which to learn,
- Let *X* be an *N* x *M* real-valued matrix and *y* be a re
  - Row *i* of *X* contains the *feature values* of
  - Element *i* of *y* is the *target* for example *i*



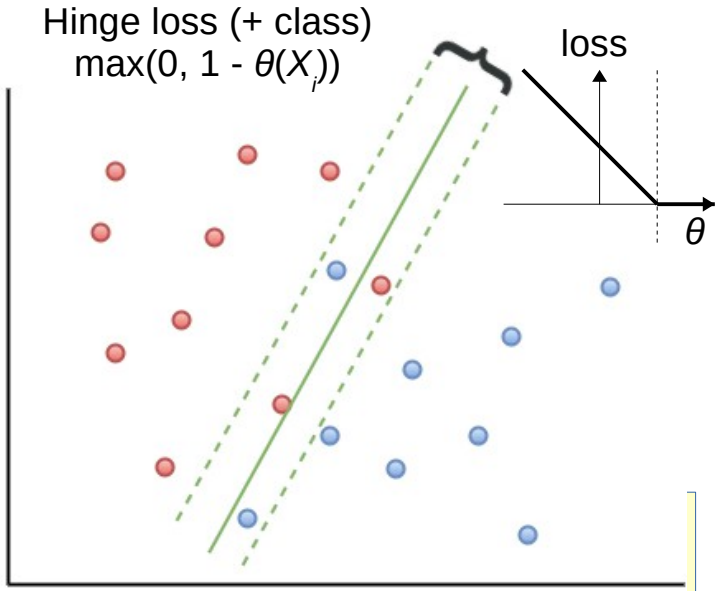
- Consider a *regression* problem
  - Produce a function $\theta : R^M \rightarrow R$ which takes an example X_i and produces an evaluation
  - Ideally, we would like to have $\theta(X_i)$ close to y_i for all examples *i*
  - In general, we define a *loss function* $L(\theta, X, y)$ to measure the distance from the ideal

Training (learning from examples)

- Consider there are *N* examples on which to learn, and *M* features per example
- Let *X* be an *N* x *M* real-valued matrix and *y* be a real vector of size *N* (remember the CSV file)
  - Row *i* of *X* contains the *feature values* of example *i*
  - Element *i* of *y* is the *target* for example *i*

- Consider a *binary classification* problem (two classes + and -)
  - Produce a function $\theta : R^M \rightarrow R$ which takes an example X_i and produces an evaluation
    - The sign of the evaluation determines the classification
  - Commonly, we try to have have $\theta(X_i)$ more than +1 for class + and less than -1 for class -
  - The loss function $L(\theta, X, y)$ will penalize values outside of the ranges above

Training (learning from examples)



- Consider there are *N* examples on which to learn, and *M* featur
- Let *X* be an *N* x *M* real-valued matrix and *y* be a real vector of si
  - Row *i* of *X* contains the *feature values* of example *i*
  - Element *i* of *y* is the *target* for example *i*

- Consider a *binary classification* problem (two classes + and -)
  - Produce a function $\theta : R^M \rightarrow R$ which takes an example X_i and produces an evaluation
    - The sign of the evaluation determines the classification
  - Commonly, we try to have have $\theta(X_i)$ more than +1 for class + and less than -1 for class -
  - The loss function $L(\theta, X, y)$ will penalize values outside of the ranges above

Machine Learning training as optimization

- With a loss function, we can specify machine learning problems as optimization problems

$$\theta^* = \underset{\theta}{\operatorname{argmin}}\, L(\theta, X, y)$$

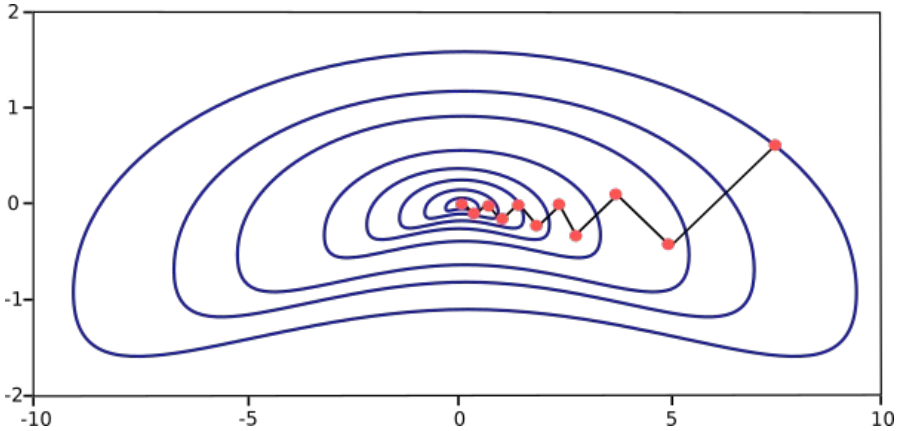
- But: in an optimization problem, the variables have initial domains or ranges
  - What's the domain of θ? All mathematical functions imaginable?

$$\theta \in ?$$

- We have different ways of structuring θ – each way leads us to a different ML technology. *e.g.*
  - Linear regression – specify one coefficient per feature, plus one more constant
  - Support vector machines – specify one coefficient per input example
  - Neural networks – specify weight of each connection and biases of each node

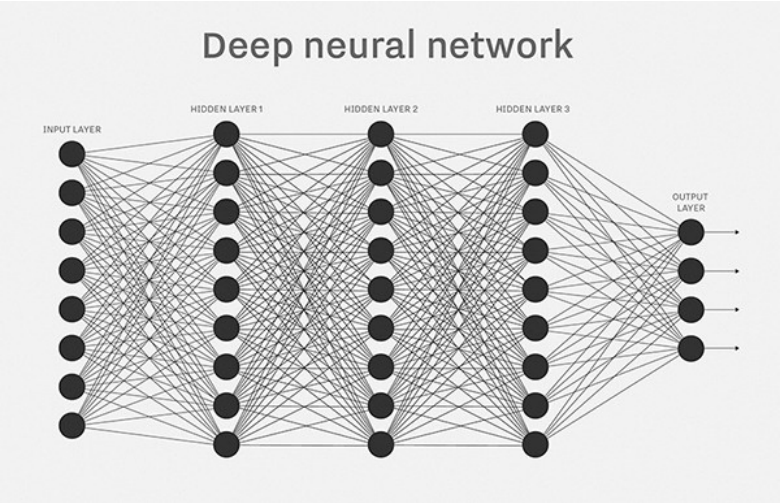
Machine Learning: optimization algorithms

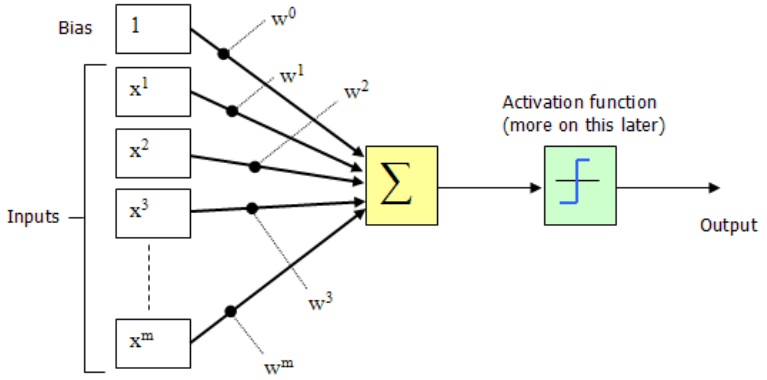
- Although ML problems can be large and dense (in the optimization sense), they tend to have some nice properties which mean that quite efficient algorithms can be employed
- Specifically, the problems can be framed to be continuous and convex, with few or no constraints. Additionally, high precision or guarantees of optimality are not required
  - Local methods related to gradient descent work well and are widespread



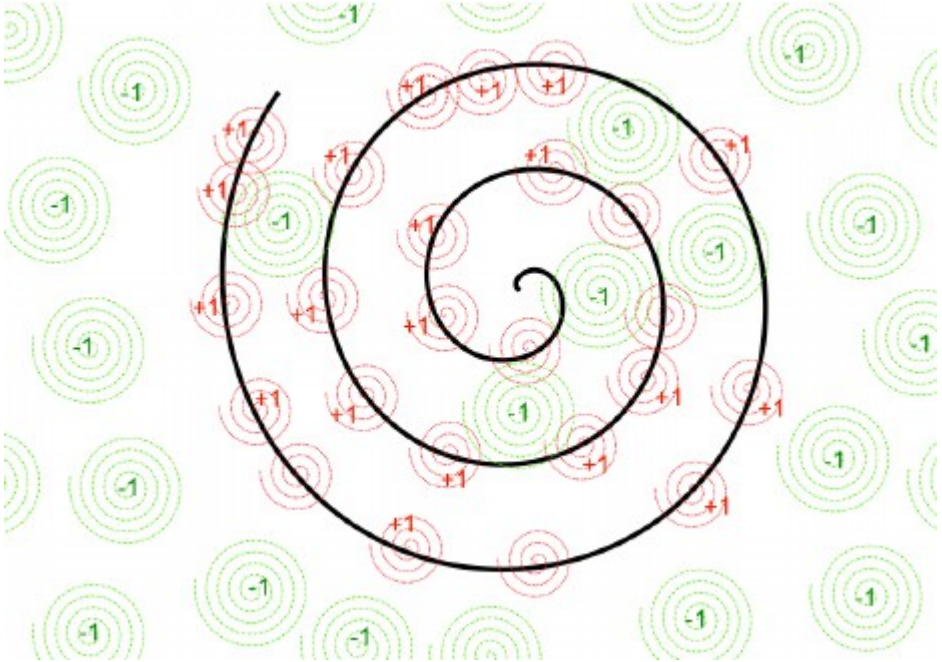
Basic idea is to take the derivative of the cost function and move "downhill". There are a number of local algorithms inspired by this type of technique used in machine learning

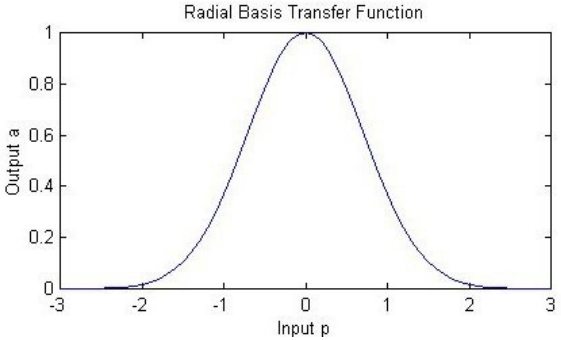
Some classifiers: Neural networks





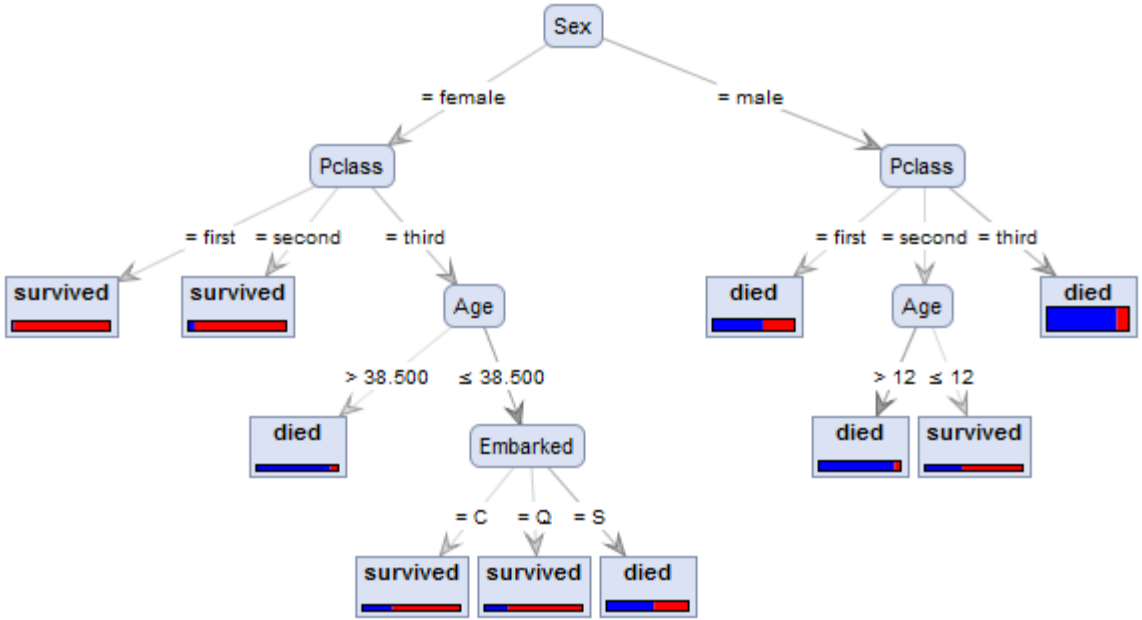
Some classifiers: Support vector machine





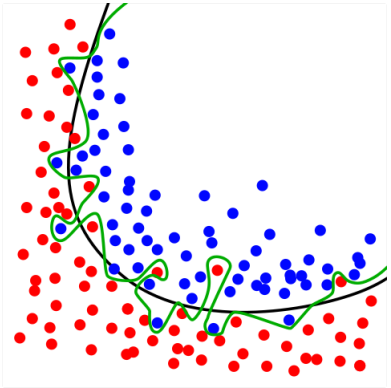
(Fast training of Support Vector Machines with Gaussian Kernel – M Fischetti, Discrete Optimization 2015)

Some classifiers: Decision trees and forests



The trouble with Machine Learning is…

- …That your brand new shiny function θ^* needs to predict future unseen examples well. We don't actually care about the quality of θ^* on training set – it's a guide
  - A function θ^* produced naively normally performs a lot worse on new examples
  - So long as the training and future examples are drawn from the same distribution, the reason is normally overfitting



(discovers overfitting)

Overfitting and dealing with it

With four parameters I can fit an elephant,
and with five I can make him wiggle his trunk.
—John von Neumann
Attributed to von Neumann by Enrico Fermi, as quoted by Freeman Dyson in "A meeting with Enrico Fermi" in Nature 427 (22 January 2004) p. 297

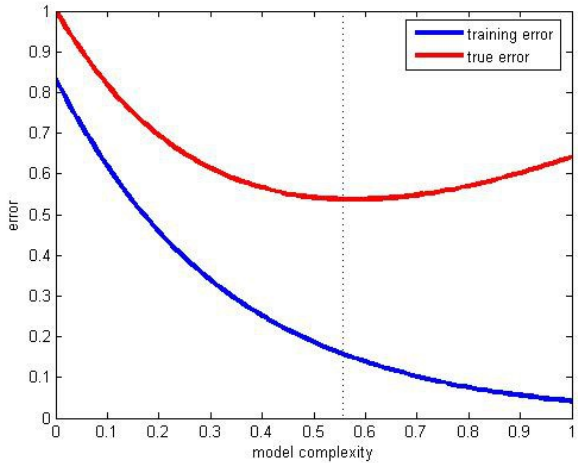
- Overfitting occurs when θ has the ability to encode so much information that it can, to some degree of approximation, "remember" the training set, hence it loses its ability to generalize. To deal with it, one can:
  - Increase the size of the training set (not always possible)
  - Fundamentally limit the model in some way (*e.g.* limit the maximum value of coefficients)
  - Introduce a "regularization term" $\Omega(\theta)$ into the optimization problem. Ω is meant to measure the "complexity" of θ in some way

- Regularization term: popular choices are the sums of the squares of coefficients of θ (L2-norm) or the sums of the absolute values of the coefficients of θ (L1-norm).

$$\theta^* = \underset{\theta}{\operatorname{argmin}} \left[L(\theta, X, y) + \lambda \, \Omega(\theta) \right]$$

Now we have λ to worry about!

Putting it together: how to train 101



- Get the input data. Randomly split it into 25% test, 25% validation, and 50% training data
- If there are not enough input examples to do this, there are ways around using “cross validation”
- For all possible values of λ we wish to explore
  - Find θ^* according to the current value of λ by optimizing over the *training set*
  - Evaluate the performance of θ^* on the *validation set* according to your preferred metric (see next slides)
  - If the performance on the validation set is the best seen so far, set $\theta_{best} = \theta^*$
- Evaluate the performance of θ_{best} on the *testing set*. This is a final evaluation of the quality of the constructed classification or regression model

Evaluating a classifier: confusion matrix

- Thanks to over-excited statisticians, there are many many ways to evaluate the quality of a classifier, depending on what you are interested in. Most of the raw information is in the "confusion matrix"

	p' (Predicted)	n' (Predicted)
p (Actual)	True Positive	False Negative
n (Actual)	False Positive	True Negative

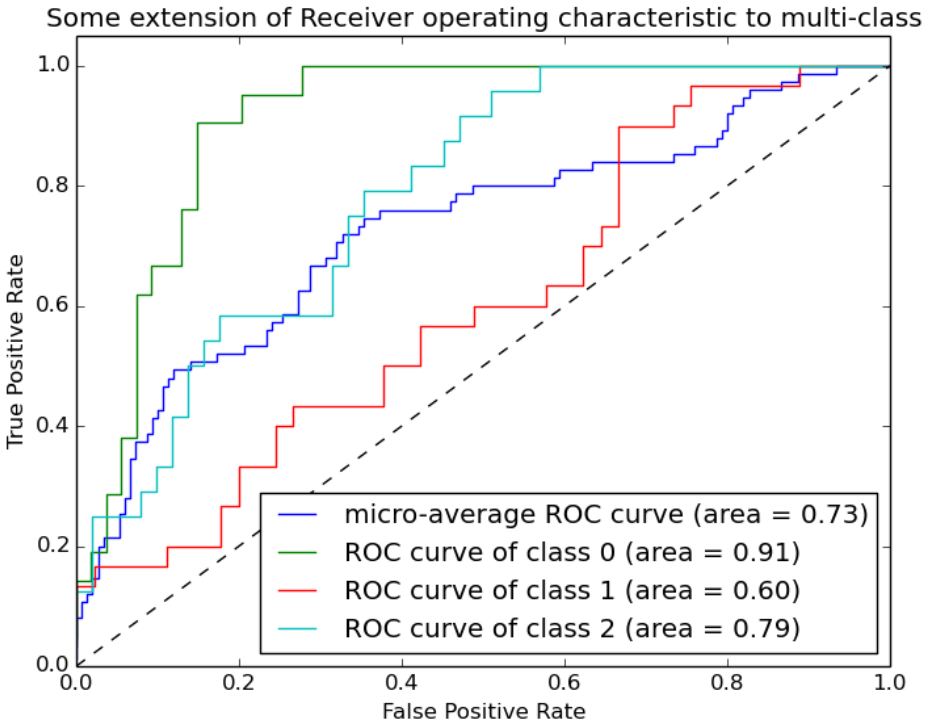
- Example

	Spam (Predicted)	Non-Spam (Predicted)	Accuracy
Spam (Actual)	27	6	81.81
Non-Spam (Actual)	10	57	85.07
Overall Accuracy			83.44

Evaluating a classifier: metrics

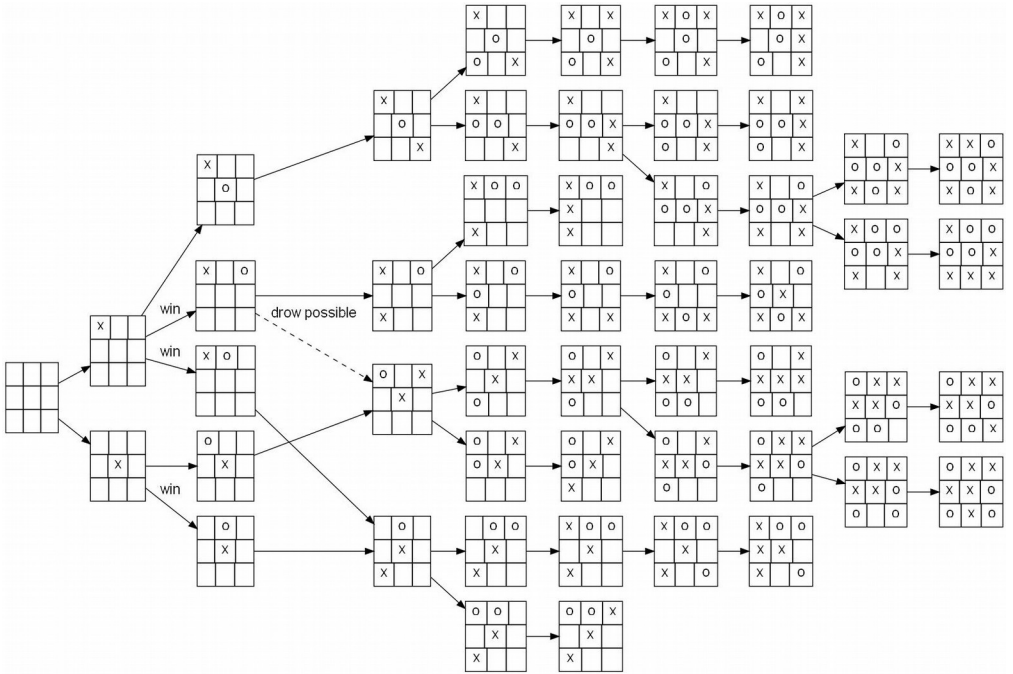
- Recall: *TP* / (*TP* + *FN*)
  - Proportion of positives found
- Specificity: *TN* / (*TN* + *FP*)
  - Proportion of negatives found
- Precision: *TP* / (*TP* + *FP*)
  - Proportion declared positive which were actually positive
- Fall-out: *FP* / (*TN* + *FP*)
  - Proportion of negatives declared positive
- NPV: TN / (TN + FN)
  - Neg. predictive value
- FDR: FP / (FP + TP)
  - False discovery rate
- FOR: FN / (FN + TN)
  - False omission rate
- Accuracy: *TP* + *TN* / (*TP* + *TN* + *FP* + *FN*)
  - Proportion correct
- F1: 2 prec. recall / (prec. + recall)
  - Harmonic mean of precision and recall

Evaluating a classifier: ROC-AUC (Receiver Operator Characteristic – Area Under Curve)



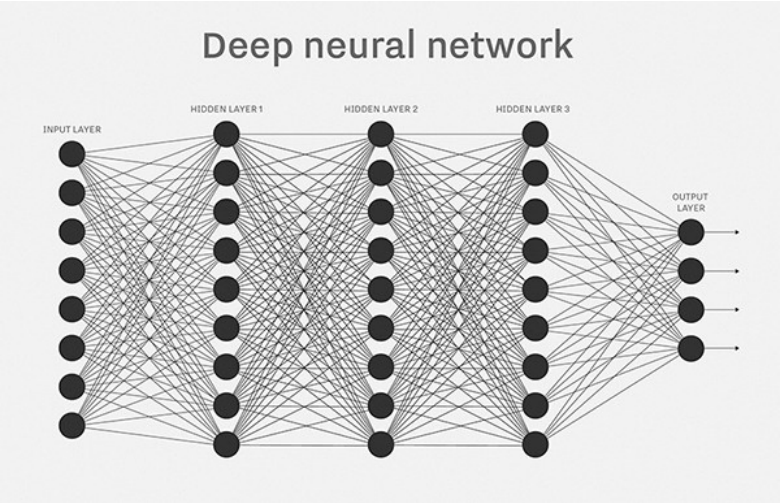
AlphaGo: A Machine Learning success

- Does not use AlphaBeta-based algorithms, but Monte Carlo tree search
  - Machine plays against itself making a ‘probable move’ at each step (called a “roll out”)
  - “Goodness” of moves made on the winning side is slightly increased after simulation
  - Used by all good Go engines
  - Innovation is to add deep learning for position and move evaluation



AlphaGo: A Machine Learning success

- Value network
  - Given a position, deliver a single probability value that white (black) will win
- Policy network
  - Given a position, deliver a value for each move square (these values sum to 1) representing move quality
- Fast policy network
  - A smaller, less accurate faster version of the Policy Network



AlphaGo: A Machine Learning success

- Value network
  - Given a position, deliver a single probability value that white (black) will win
- Policy network
  - Given a position, deliver a value for each move square (these values sum to 1) representing move quality
- Fast policy network
  - A smaller, faster, less accurate version of the Policy Network

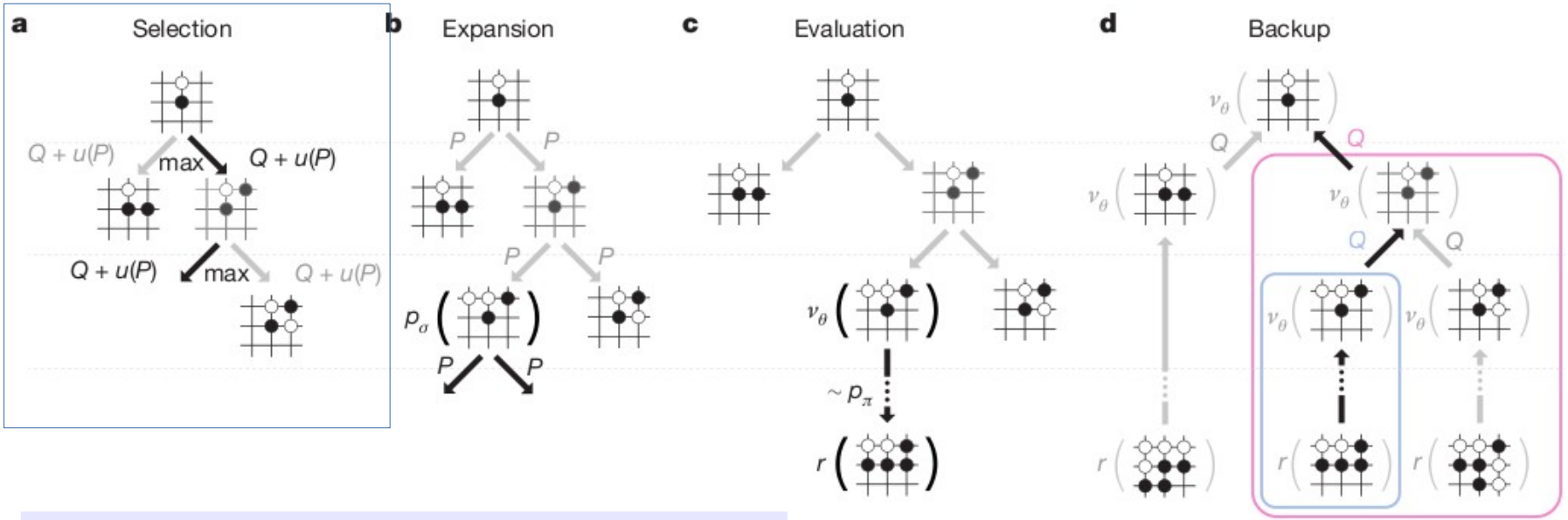
- Policy network
  - Trained by studying master games – can find the master move 57% of the time
  - Improved by playing against previous versions of itself
  - PN *alone with no search* won 85% of games against *Pachi*
- Fast policy network
  - 2 microseconds to evaluate

AlphaGo: A Machine Learning success

- Policy network
  - Given a position, deliver a single probability value that white (black) will win
- Policy network
  - Given a position, deliver a value for each move square (these values sum to 1) representing move quality
- Fast policy network
  - A smaller, faster, less accurate version of the Policy Network

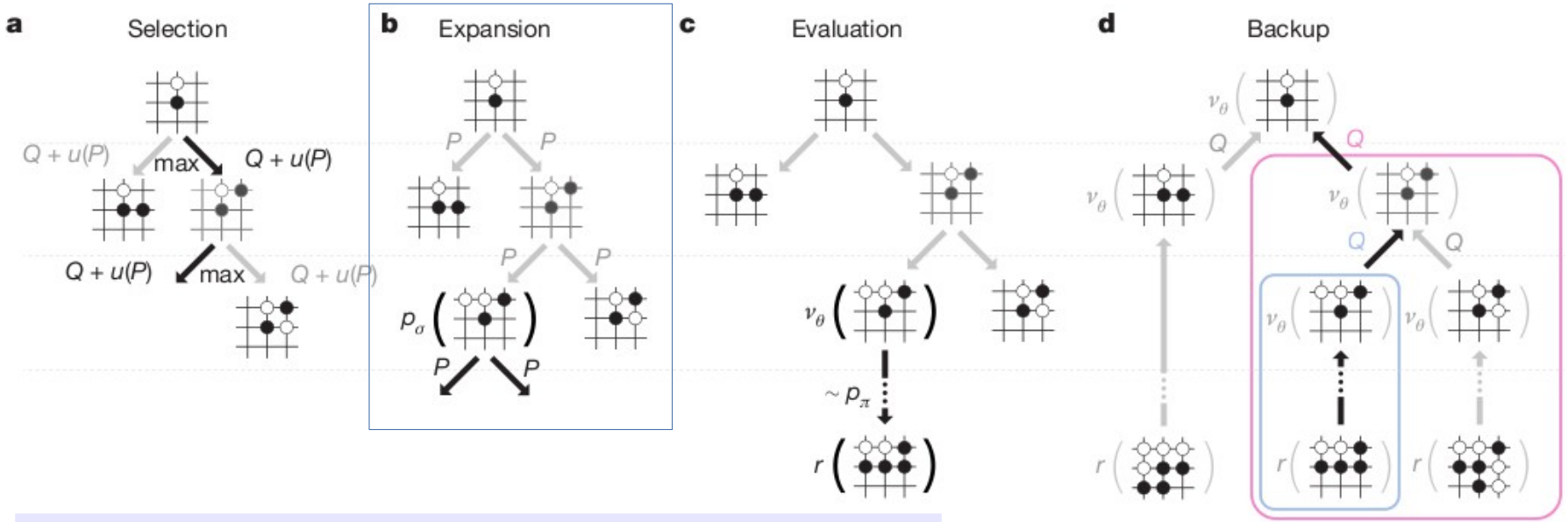
- Value network
  - Trained by simulating games using the PN from over 30 million positions
  - Final win/loss result from a position is used to control a gradient descent method

AlphaGo: A Machine Learning success



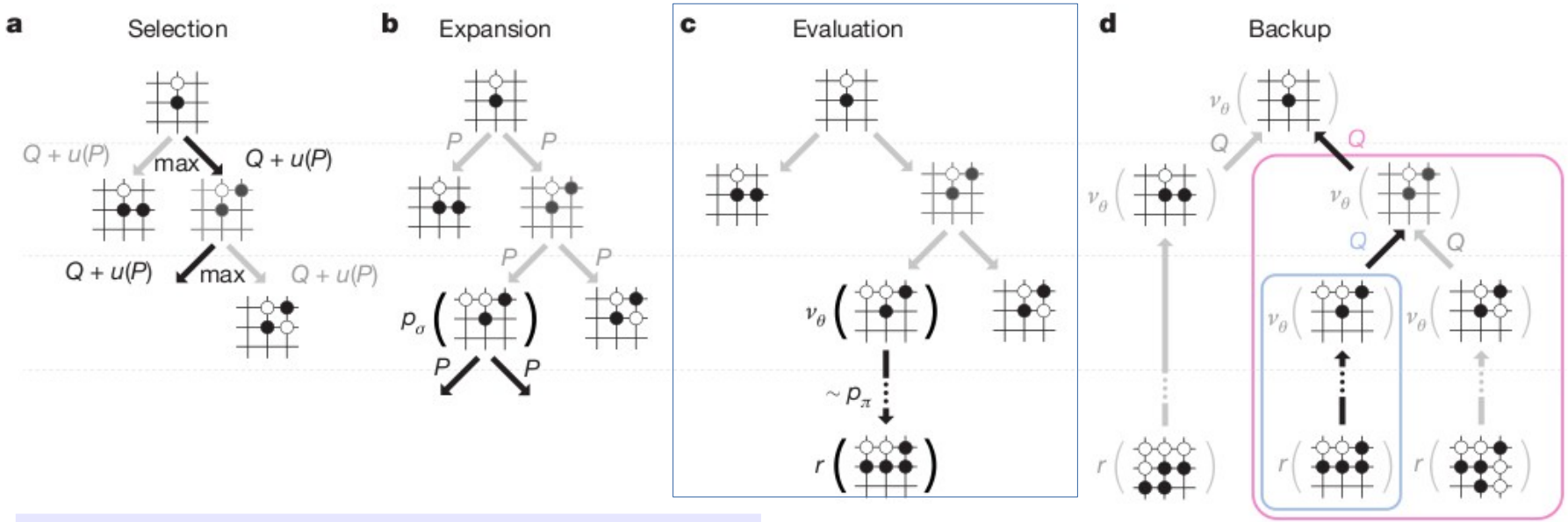
Selection: move evaluations are known
"Roughly" follow best moves to a leaf node

AlphaGo: A Machine Learning success



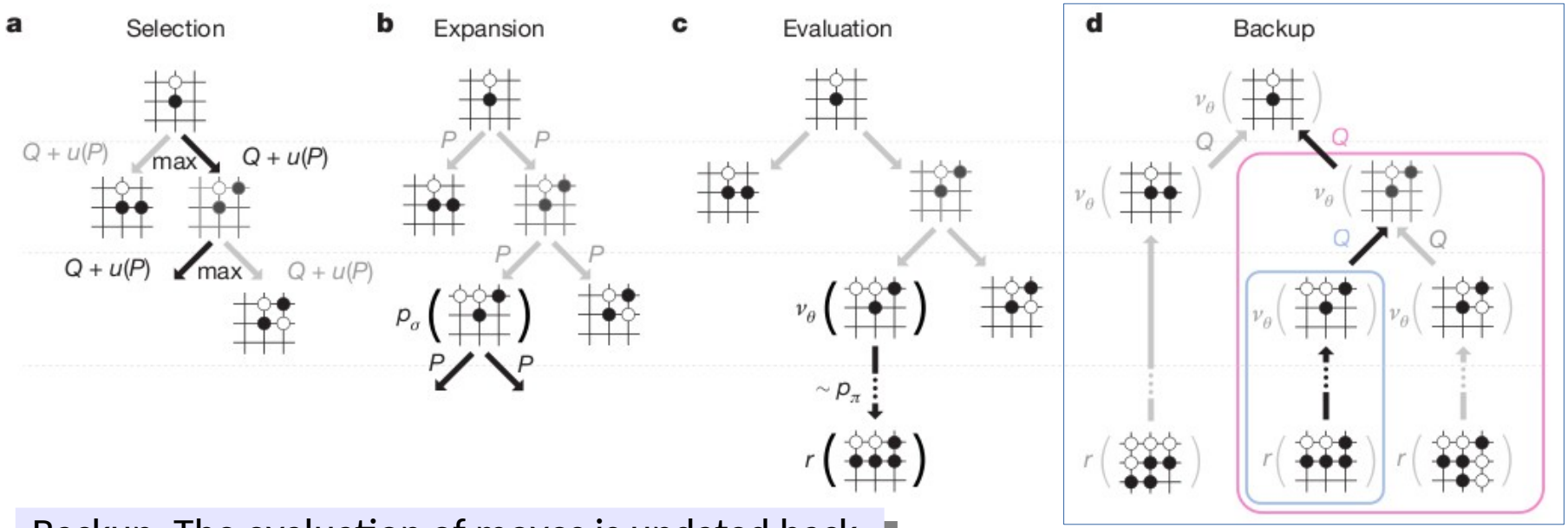
Expansion: The leaf node is expanded by computing moves from that position with their evaluations (PN)

AlphaGo: A Machine Learning success



Evaluation: The quality of moves from the (ex-)leaf node are evaluted (FPN + VN)

AlphaGo: A Machine Learning success



Backup: The evaluation of moves is updated back up the the tree to influence subsequent selection

AlphaGo: A Machine Learning success

- Parallel Architecture
  - 48 CPUs and 8 GPUs
  - Neural network evaluations can use vectorized operations
  - Different simulations using MCTS can be done in parallel
- Distributed Architecture
  - 1202 CPUs, 176 GPUs
- Results
  - Defeated Lee Sedol, 18 times world champion in a 5 game match, 4-1
  - ELO rating well over 3000