COMPSCI 367 Tutorial 3

Logic: another thing that penguins aren't very good at.

Overview

- Machine Learning?
- Well posed learning problems
- Designing a learning system (Checkers)

Machine Learning

- Material taken from “Machine Learning” - Tom M. Mitchell. Course textbook.

&

- Carl Schultz's 367 tutorial notes from 2008.

Machine Learning

- Learn: to improve automatically with experience
- Machine Learning: programs that improve automatically with experience
  - e.g,
    - successfully recognise spoken words
    - play better checkers

Well Posed Learning Problems

- *Definition:* A computer program is said to **learn** from experience *E* with respect to some class of tasks *T* and performance measure *P*, if its performance at tasks in *T*, as measured by *P*, improves with experience *E*.

Well Posed Learning Problems

- *T*: class of tasks that we want computer program to do
- *P*: measure of performance for how well computer did
- *E*: some experience program has with task

Well Posed Learning Problems

- A checkers learning problem
  - *T*: playing checkers
  - *P*: percent of games won against opponents
  - *E* playing practice games against itself

Well Posed Learning Problems

- A handwriting recognition learning problem
  - *T*: recognising and classifying handwritten words within images
  - *P*: percent of words correctly classified
  - *E*: a database of handwritten words with given classifications

Well Posed Learning Problems

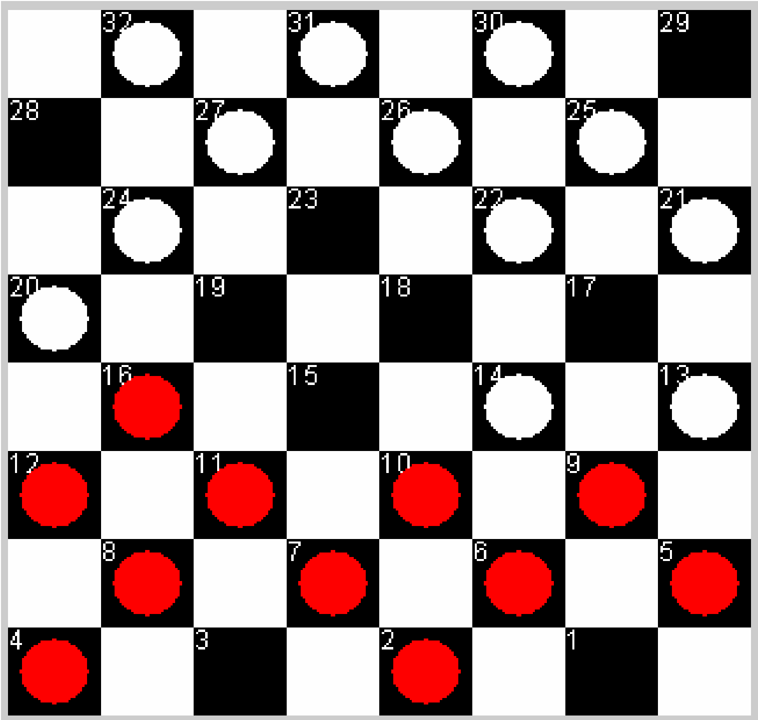
- A robot driving learning problem
  - *T*: driving on public motorways using vision sensors
  - *P*: average distance travelled before an error (as judged by human overseer)
  - *E*: a sequence of images and steering commands recorded while observing a human driver

Designing a Learning System

- Goal: Design a system to learn how to play checkers and enter it into the world checkers tournament.
  - 1) Choose the training experience
  - 2) Choose the target function
  - 3) Choose a representation for the target function
  - 4) Choose a function approximation algorithm

1) Choose the training experience

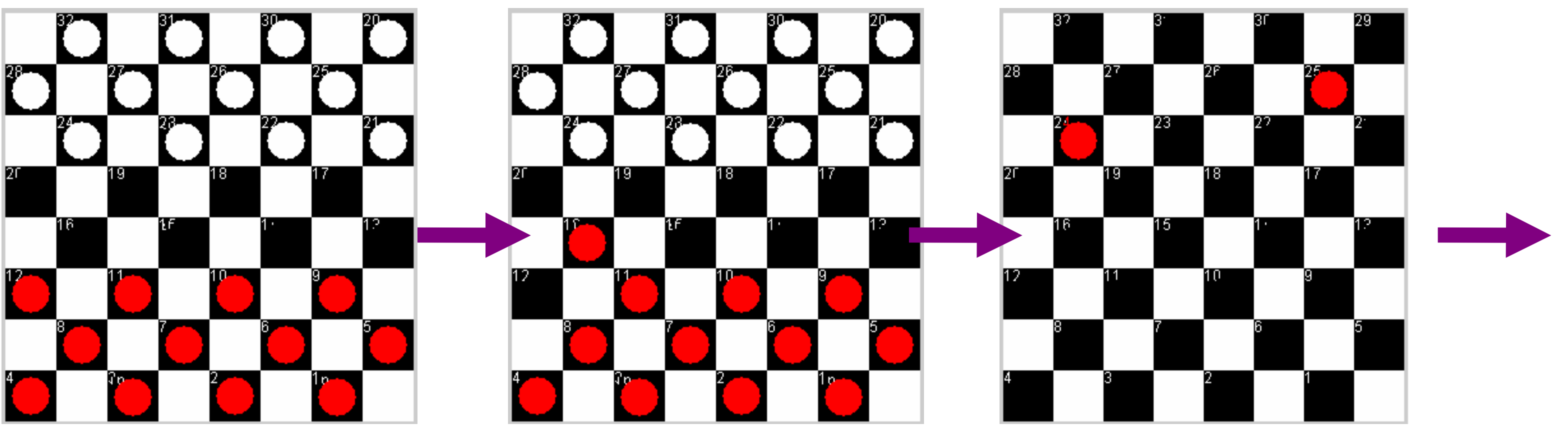
Direct



16 → 19

1) Choose the training experience

Indirect

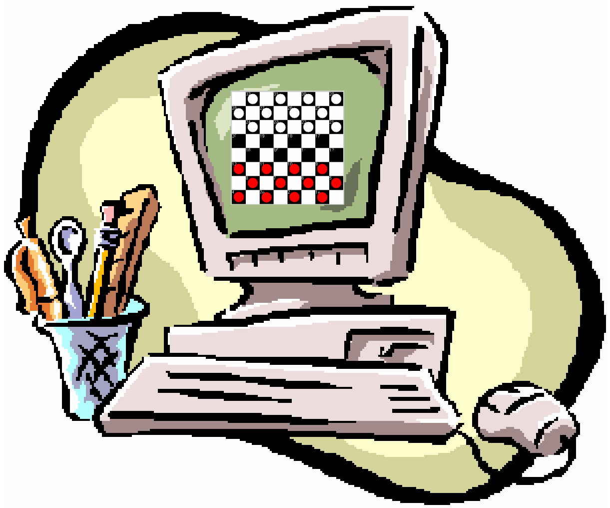


WIN

Credit Assignment Problem!

1) Choose the training experience

- Self-play experiments



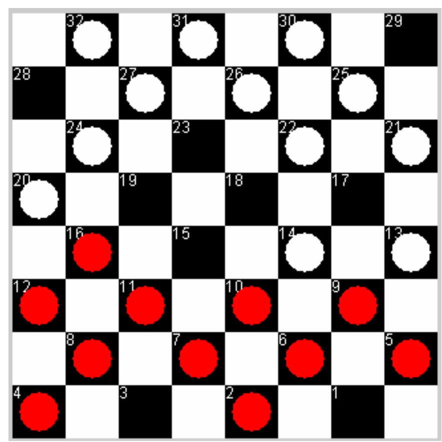
- No need for trainer.

Summary (so far)

- Decisions made
  - *T*: playing checkers
  - *P*: percent of games won in the world tournament
  - *E*: games played against itself
- Decisions yet to be made
  - The exact type of knowledge to be learned
  - A representation for this target knowledge
  - A learning mechanism

2) Choose the target function

- *ChooseMove*(*B*) → *M*

ChooseMove (

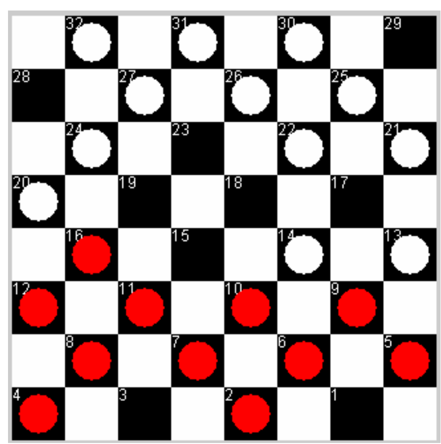
) = 16 → 19

Difficult, given our training experience

2) Choose the target function

- $V(b) \rightarrow R$
  - Where *V* is an evaluation function that maps board, *b*, to some real number *R.*

V



= 30 points

Easier to learn

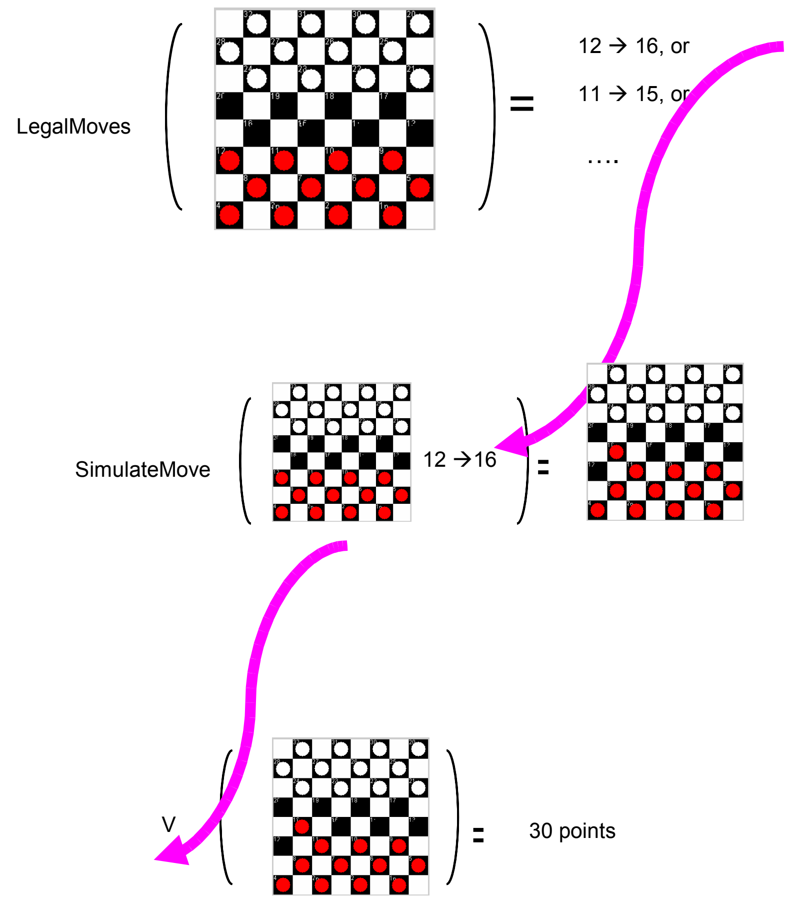
2) Choose the target function

For all legal moves:

- simulate that move

- check the value of the result

Pick the best move



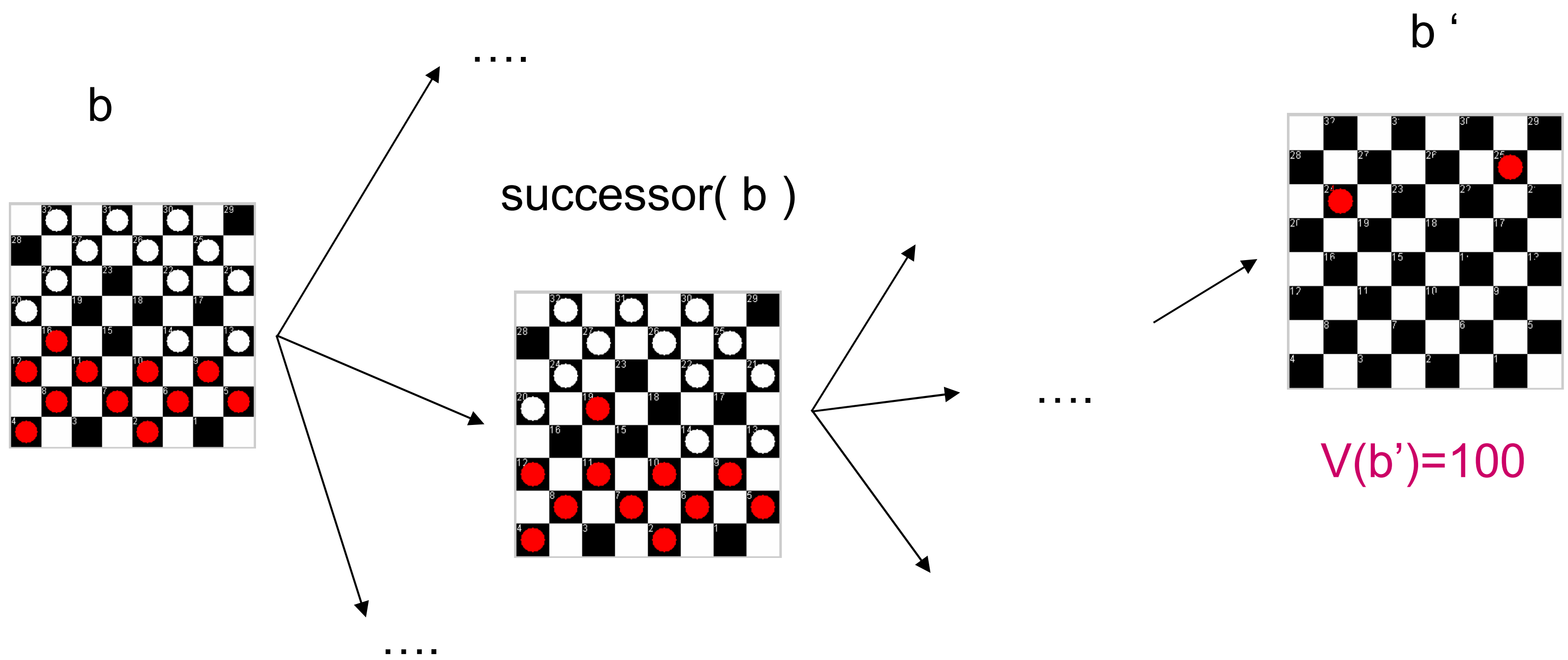
2) Choose the target function

- What values should the target function, *V*, produce?
  - $V(b) = 100$, if b is a final board state that is won
  - $V(b) = -100$, if b is a final board state that is lost
  - $V(b) = 0$, if b is a final board state that is a draw

…but what about boards in the middle of a game?

2) Choose the target function

- V(b) = V(b´), if b is not a final state where b´ is the best final board state starting from b assuming both players play optimally



Not efficiently computable!

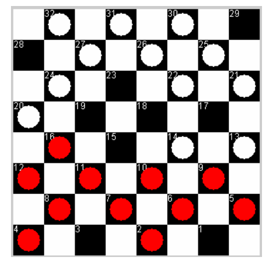
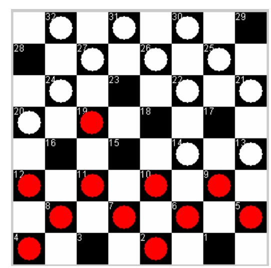
2) Choose the target function

- *V* is too hard to learn
- *Function approximation*
  - Learn an approximation to ideal target function *V*

$$\hat{V}(b) \simeq V(b)$$

3) Choose a representation

1) big table?

input: board	*score*
	50
	90
....	

3) Choose a representation

2) CLIPS rules?

IF my piece is near a side edge

THEN score = 80

IF my piece is near the opponents edge

THEN score = 90

....

3) Choose a representation

3) polynomial function? we define some variables…

X_1 = number of white pieces on board

X_2 = number of red pieces

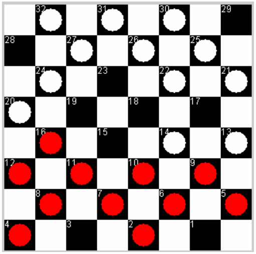
X_3 = number of white kings

X_4 = number of red kings

X_5 = number of white pieces threatened by red (can be captured on red's next turn)

X_6 = number of red pieces threatened by white

e.g.



$X_1 = 12$ $X_3 = 0$ $X_5 = 1$

$X_2 = 11$ $X_4 = 0$ $X_6 = 0$

3) Choose a representation

Quick maths revision

- **polynomial**. expression with linear (+ and –) combination of terms (constants × variables) where exponent is non-negative integer (x^4 is okay, but $x^{3/4}$ or x^{-2} not polynomial), e.g. these are polynomial expressions:
  - $x^5 + 3$
  - $x^2 + 5x + 1$
  - $x^3 + 3x^0$
  - $f(x)=x^3 + 2$ …is a polynomial function
  - $0=x^3 + 2$ …is a polynomial equation
- **degree**. take a term, sum the exponents of the variables in that term
  - x^5 has degree 5
  - xy has degree 2
- **degree of a polynomial**. is the highest degree of any of the terms – polynomials with degree 1 to 5 are given special names
  - linear. has degree 1
  - quadratic. has degree 2
  - cubic. has degree 3
  - quartic. has degree 4
  - quintic. has degree 5
- **quadratic polynomial**. has degree 2, e.g.
  - x^2
  - $10x + 3 + x^2$
- *NB: in many cases, people say "quadratic" and mean that the highest* ***allowable*** *degree is 2 (not necessarily exactly 2), i.e. the degree might be less*

3) Choose a representation

- Weighted linear combination:

Computer program will change the values of the weights – it will *learn* what the weights should be to give correct score for each board

$$\hat{V}(b) = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_4 + w_5 x_5 + w_6 x_6$$

4) Choose a function approximation algorithm

- Require a set of training examples
- Ordered pair: $<b, V_{train}(b)>$
- e.g.
  - Black has won (red has no pieces left):

$$<x_1=3, x_2=0, x_3=1, x_4=0, x_5=0, x_6=0>, +100>$$

X_1 = number of white pieces on board

X_2 = number of red pieces

X_3 = number of white kings

X_4 = number of red kings

X_5 = number of white pieces threatened by red (can be captured on red's next turn)

X_6 = number of red pieces threatened by white

4) Choose a function approximation algorithm

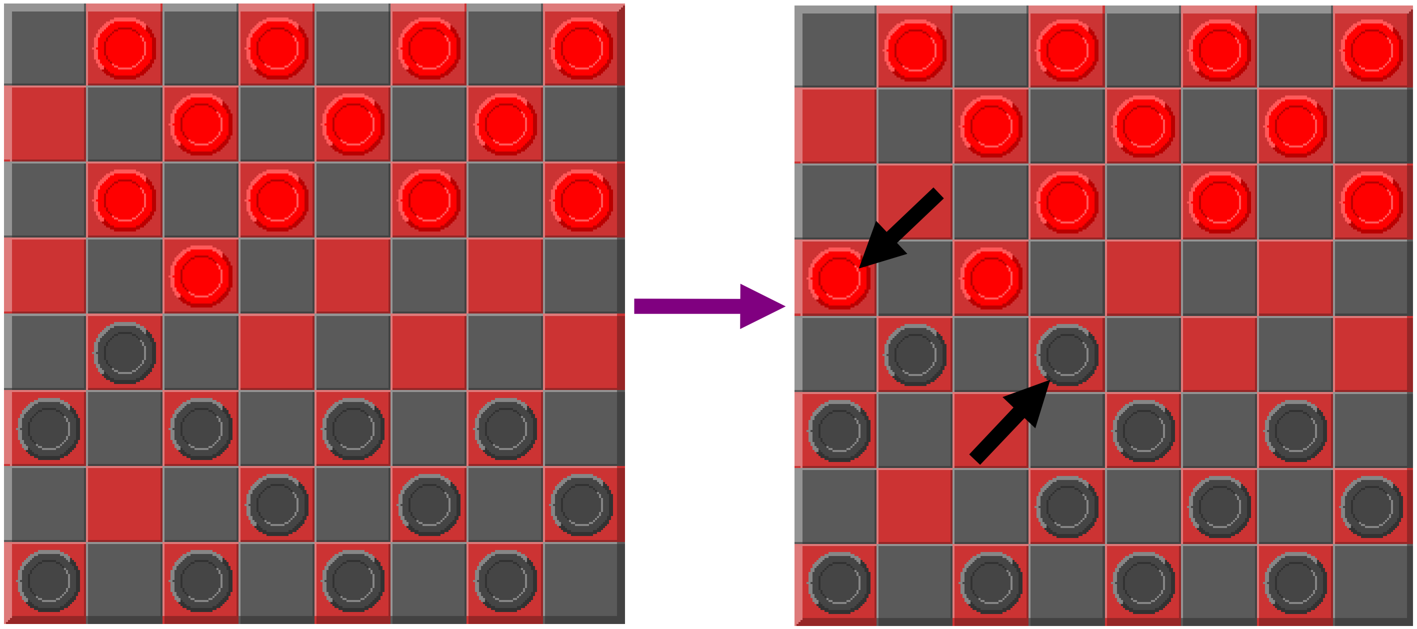
- What about intermediate board states?

$$V_{train}(b) \leftarrow \widehat{V}(Successor(b))$$

Where *Successor(b)* is the next board state following *b*, which it is again the program's turn to move (i.e. the board state following the program's move and the opponents response)

4) Choose a function approximation algorithm

- *b:*
- *Successor(b)*:



4) Choose a function approximation algorithm

- Now, we have the training data
- Need an algorithm to adjust the weights to *best fit* this training data.
- Common approach: best set of weights will minimise the squared error, *E*, between training values and value predicted by $\widehat{V}$

4) Choose a function approximation algorithm

$$E \equiv \sum_{\langle b, V_{train}(b) \rangle \in training\ examples} \left(V_{train}(b) - \hat{V}(b) \right)^2$$

- We wish to minimise *E*, for the observed training examples

4) Choose a function approximation algorithm

- Need algorithm that will incrementally refine weights as more training examples become available
- Needs to be robust to errors in training data
- LMS training rule: will adjust weights a small amount in the direction that reduces the error

4) Choose a function approximation algorithm

- LMS weight update rule.
  - For each training example <*b*, $V_{train}(b)$>
    - Use the current weights to calculate $\hat{V}(b)$
    - For each weight w_i, update it as

$$w_i \leftarrow w_i + \mu \left(V_{train}(b) - \hat{V}(b) \right) x_i$$

where,

$$V_{train}(b) \leftarrow \hat{V}(Successor(b))$$

4) Choose a function approximation algorithm

$$w_i \leftarrow w_i + \mu \left(\boxed{\hat{V}(Successor(b)) - \hat{V}(b)} \right) x_i$$

error

weight to update

learning rate

modify weight in proportion to size of attribute value

Final Algorithm

learning process – playing and learning at same time

computer will play against itself

initialise $\widehat{V}$ with random weights (w_0=23 etc.)

start a new game - for each board

(a) calculate $\widehat{V}$ on all possible legal moves

(b) pick the successor with the highest score

(c) evaluate error

(d) modify each weight to correct error

First time round $\widehat{V}$ is essentially random (because we set the weights as random) – as it learns $\widehat{V}$ should pick better successors

1 million games later and our $\widehat{V}$ *might* now predict a useful score for any board that we might see

Final Algorithm

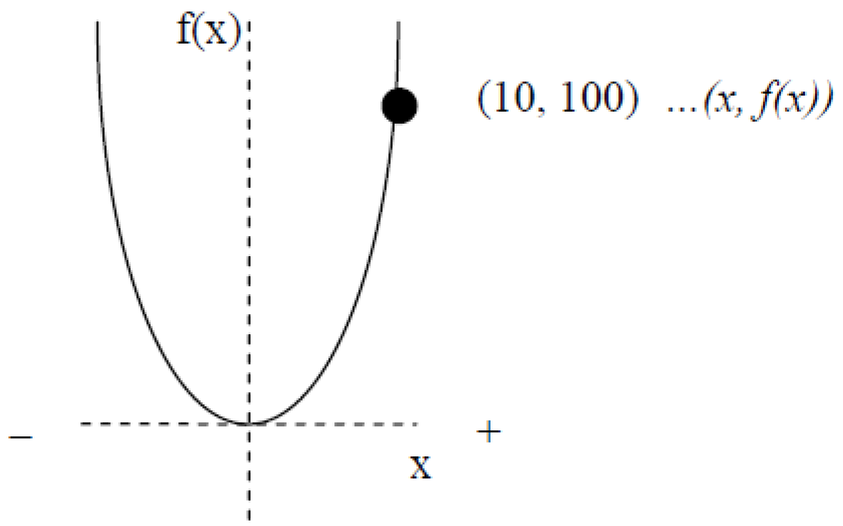
- $(w_0 = 25.0, w_1 = 15.5, w_2 = 19.3, ..., w_6 = 54.4)$
- $(w_0 = 82.0, w_1 = 15.3, w_2 = 9.9, ..., w_6 = 100.0)$
- $(w_0 = 67.3, w_1 = 0.5, w_2 = 3.5, ..., w_6 = 30.2)$
- …
- $(w_0 = 3.23, w_1 = 1.04, w_2 = 4.55, ..., w_6 = 0.78)$
- Searching hypothesis space for best fit to observed training data.

Quick maths revision

- **gradient-descent**. finds the local minimum of a function – does this by moving in direction *negative* of gradient at the current point
    - **a** is current point
    - **b** is next point
    - f '(**a**) is gradient of function at point **a**
    - η is the size of the step that we'll take (must be +ve)

$$\mathbf{b} = \mathbf{a} - \eta \cdot f'(\mathbf{a})$$

- $f(x) = x^2$

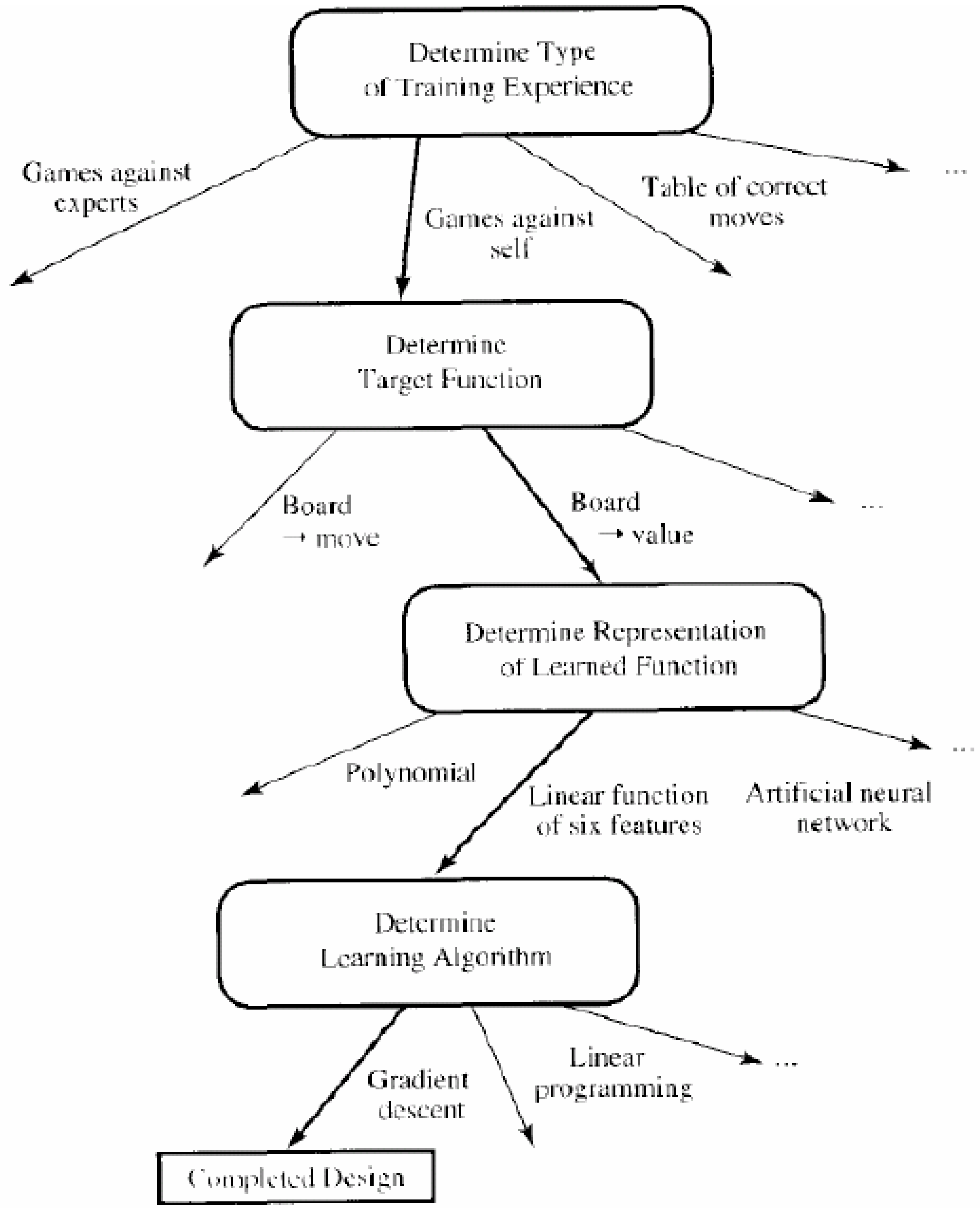


Quick maths revision

- $f'(x) = 2x$ …derivative of our function used to get the gradient at our point
    - $f'(10) = 20$ *...formula says we move negative to gradient, so says move left along graph, which seems sensible*
    - $\mathbf{a} = 10$
    - $\eta = 0.1$ *...something small*
    - $\mathbf{b} = 10 - 0.1 \times 20 = 8$ *...8 < 10, so we are moving towards the minimum*

- can work for functions with arbitrary number of variables
    - $f(x, \ldots, z)$

- to do this, take partial derivatives of each variable in turn
    - **partial derivative**. differentiate function for one variable, and fix all other variables (treat as constants)
    - in terms of *learning*, this means we modify each weight in turn


Determine Type
of Training Experience
Games against
experts
Games against
self
Table of correct
moves
...
Determine
Target Function
Board
→ move
Board
→ value
...
Determine Representation
of Learned Function
Polynomial
Linear function
of six features
Artificial neural
network
...
Determine
Learning Algorithm
Gradient
descent
Linear
programming
...
Completed Design