Learning with hierarchical features

David Silver
Outline

- Introduction to hierarchical features
- Learning issues with hierarchical features
- Experimental setup
- Preliminary results
- Scaling up to Go
Hierarchical features

- Binary features are either on or off in any state.
- Define partial ordering between features.
- If feature A => feature B, then A is a child of B.
Example: Go shapes
Example: logic

A

~A

B

~B

A^B

A^C

A^~B

~B^C

~B^~C
Example: bit features
Terminology

- If feature $A \Rightarrow feature\ B$
  - $B$ is more general than $A$
  - $A$ is more specific than $B$
- A strict hierarchy has a strict partial order
  - If feature $A \Rightarrow feature\ B$
  - then feature $B \not\Rightarrow feature\ A$
Terminology

- In an **exclusive** hierarchy
  - If parent is on, exactly one child is on
  - A single path through hierarchy is on
Motivation

- More general features learn quickly
- More specific features learn accurately
- How to combine features to get the best of both worlds?
Linear function approximation

- Prediction is a linear combination of features
  \[ y = \sum w_j x_j \]
- One weight for each feature
- Find weights that minimise squared error between prediction and target
  \[ w = \text{argmin } \mathbb{E}[(z-y)^2] \]
Hierarchical function approximation

- Many solutions exist
- In general a large subspace of solutions
- Internal nodes of the hierarchy are 'redundant'
- Leaves are sufficient to describe any solution
Cascade learning

Define a particular set of weights as optimal:

- Weight for feature A minimises error given all features more general than A
- Weight for feature A minimises total magnitude of children of A
Two algorithms

- Online cascade
  - Gradient descent using error of all more general features
- Floating average
  - Gradient descent at leaves
  - Average weights up hierarchy
Step-size control

“All features are equal. But some are more equal than others”

Frequencies of hierarchical features span many orders of magnitude

Don’t learn all features at the same rate
Step-sizes and uncertainty

- Each weight $w_j$ can be viewed as an estimate of the optimal weight $w_j^*$
- There is an uncertainty associated with this estimate: the estimation error
- Estimation error should determine step-size.
- Uncertain features $\Rightarrow$ high step-size
- Certain features $\Rightarrow$ low step-size
Measuring uncertainty

- Observed error = estimation error + approximation error + noise
- How to separate out estimation error?
  - Kalman filter
  - Linear regression
  - Gradient descent
  - Hierarchical bounds
Hierarchical bounds

- If feature B is a child of feature A (B => A)
  - Approximation error at A > B
  - Estimation error at B > A
  - Noise at B = A
- Use these bounds to track error intervals
Experimental setup

The world is binary, e.g. 01101000

Features recognise binary patterns

1-bit features: e.g. **1******, *****0**

2-bit features: e.g. **10***** , *****00**

3-bit features: e.g. *110***** , ***100**
Using uncertainty

- Diagonal Kalman filter assigns step-size in proportion to uncertainty
- Are there better strategies for hierarchies?
  - Weighted uncertainty
  - Resource pot
- Does floating average resolve this issue?
Experimental setup

- Target really is linear in features
  \[ z = \sum w_j^* x_j \]
  \[ w_j^* \sim \mathcal{N}(0, 2^{-l_j}) \]
- No added noise
- States (binary sequences) selected with uniform probability
Experimental setup

- 32 bit world
- All 1-bit to 8-bit features used in target
- Different subsets of features used during learning:
  - = n-bit features
  - ≥ n-bit features
  - ≤ n-bit features
Some results
Scaling up to Computer Go

- Same form of features, same issues
- Temporal sequences
- Non-stationary target
- Less structured target
- Location invariant features