Using the Shape of Space for Shortcuts

Speeding up regressions on millions of chess positions

Kenneth W. Regan[1]

FWCG, University at Buffalo, 23 October 2015

[1] Joint with Tamal T. Biswas, AAIM 2014, *Theoretical Computer Science* (in press).

The Setting

- Evaluate an **expensive** $f(x) = f(x_1, \ldots, x_\ell)$ at millions of points x.

The Setting

- Evaluate an **expensive** $f(x) = f(x_1, \ldots, x_\ell)$ at millions of points x.
- **Idea**: Precompute & store values $v_u = f(u_1, \ldots, u_\ell)$ at gridpoints u.

The Setting

- Evaluate an **expensive** $f(x) = f(x_1, \ldots, x_\ell)$ at millions of points x.
- **Idea**: Precompute & store values $v_u = f(u_1, \ldots, u_\ell)$ at gridpoints u.
- Estimate $f(x)$ via v_u for nearby gridpoints u.

The Setting

- Evaluate an **expensive** $f(x) = f(x_1, \ldots, x_\ell)$ at millions of points x.
- **Idea**: Precompute & store values $v_u = f(u_1, \ldots, u_\ell)$ at gridpoints u.
- Estimate $f(x)$ via v_u for nearby gridpoints u.

- **Three main options** for doing this:

The Setting

- Evaluate an **expensive** $f(x) = f(x_1, \ldots, x_\ell)$ at millions of points x.
- **Idea**: Precompute & store values $v_u = f(u_1, \ldots, u_\ell)$ at gridpoints u.
- Estimate $f(x)$ via v_u for nearby gridpoints u.

- **Three main options** for doing this:

1. **Interpolate** using v_u for (all) vertices of the cell x is in.

The Setting

- Evaluate an **expensive** $f(x) = f(x_1, \ldots, x_\ell)$ at millions of points x.
- **Idea**: Precompute & store values $v_u = f(u_1, \ldots, u_\ell)$ at gridpoints u.
- Estimate $f(x)$ via v_u for nearby gridpoints u.

- **Three main options** for doing this:

1. **Interpolate** using v_u for (all) vertices of the cell x is in.
   Problem: naive "all" is exponential in the dimension ℓ.

The Setting

- Evaluate an **expensive** $f(x) = f(x_1, \ldots, x_\ell)$ at millions of points x.
- **Idea**: Precompute & store values $v_u = f(u_1, \ldots, u_\ell)$ at gridpoints u.
- Estimate $f(x)$ via v_u for nearby gridpoints u.

- **Three main options** for doing this:

1. **Interpolate** using v_u for (all) vertices of the cell x is in.
   Problem: naive "all" is exponential in the dimension ℓ.
2. Do **Taylor expansion** using one or a few nearby gridpoints u.

The Setting

- Evaluate an **expensive** $f(x) = f(x_1, \ldots, x_\ell)$ at millions of points x.
- **Idea**: Precompute & store values $v_u = f(u_1, \ldots, u_\ell)$ at gridpoints u.
- Estimate $f(x)$ via v_u for nearby gridpoints u.

- **Three main options** for doing this:

1. **Interpolate** using v_u for (all) vertices of the cell x is in.
   Problem: naive "all" is exponential in the dimension ℓ.
2. Do **Taylor expansion** using one or a few nearby gridpoints u.
   Problem: this needs precomputing partials too.

The Setting

- Evaluate an **expensive** $f(x) = f(x_1, \ldots, x_\ell)$ at millions of points x.
- **Idea**: Precompute & store values $v_u = f(u_1, \ldots, u_\ell)$ at gridpoints u.
- Estimate $f(x)$ via v_u for nearby gridpoints u.

- **Three main options** for doing this:

1. **Interpolate** using v_u for (all) vertices of the cell x is in.
   Problem: naive "all" is exponential in the dimension ℓ.
2. Do **Taylor expansion** using one or a few nearby gridpoints u.
   Problem: this needs precomputing partials too.
3. **Cheat.**

Wider Context—Cheating Detection at Chess

- Two activities: **training** and **testing**.

Wider Context—Cheating Detection at Chess

- Two activities: **training** and **testing**.
- First sets for years up to 2009 totaled just over 1 million positions.

Wider Context—Cheating Detection at Chess

- Two activities: **training** and **testing**.
- First sets for years up to 2009 totaled just over 1 million positions.
- New 2010–2014 set has 1.15 million positions. Update each year...

Wider Context—Cheating Detection at Chess

- Two activities: **training** and **testing**.
- First sets for years up to 2009 totaled just over 1 million positions.
- New 2010–2014 set has 1.15 million positions. Update each year...
- *Kaggle* competition set running now $\longrightarrow$ 3 million positions.

Wider Context—Cheating Detection at Chess

- Two activities: **training** and **testing**.
- First sets for years up to 2009 totaled just over 1 million positions.
- New 2010–2014 set has 1.15 million positions. Update each year...
- *Kaggle* competition set running now $\longrightarrow$ 3 million positions.
- Multiply by $\ell = 30$–35 legal moves per position on average and by 10–100 "Newton" or Nelder-Mead iterations per run.

Wider Context—Cheating Detection at Chess

- Two activities: **training** and **testing**.
- First sets for years up to 2009 totaled just over 1 million positions.
- New 2010–2014 set has 1.15 million positions. Update each year...
- *Kaggle* competition set running now $\longrightarrow$ 3 million positions.
- Multiply by $\ell = 30$–35 legal moves per position on average and by 10–100 "Newton" or Nelder-Mead iterations per run.
- Model parameters $s, c, \ldots$ trained to chess **Elo ratings** via nonlinear regression.

Wider Context—Cheating Detection at Chess

- Two activities: **training** and **testing**.
- First sets for years up to 2009 totaled just over 1 million positions.
- New 2010–2014 set has 1.15 million positions. Update each year...
- *Kaggle* competition set running now $\longrightarrow$ 3 million positions.
- Multiply by $\ell = 30$–35 legal moves per position on average and by 10–100 "Newton" or Nelder-Mead iterations per run.
- Model parameters $s, c, \dots$ trained to chess **Elo ratings** via nonlinear regression.
- **Cheating test** regresses on typically 6–9 games, 200–300 positions by one player.

Wider Context—Cheating Detection at Chess

- Two activities: **training** and **testing**.
- First sets for years up to 2009 totaled just over 1 million positions.
- New 2010–2014 set has 1.15 million positions. Update each year...
- *Kaggle* competition set running now $\longrightarrow$ 3 million positions.
- Multiply by $\ell = 30$–35 legal moves per position on average and by 10–100 "Newton" or Nelder-Mead iterations per run.
- Model parameters $s, c, \dots$ trained to chess **Elo ratings** via nonlinear regression.
- **Cheating test** regresses on typically 6–9 games, 200–300 positions by one player. Full accuracy is vital for this test...

Wider Context—Cheating Detection at Chess

- Two activities: **training** and **testing**.
- First sets for years up to 2009 totaled just over 1 million positions.
- New 2010–2014 set has 1.15 million positions. Update each year...
- *Kaggle* competition set running now $\longrightarrow$ 3 million positions.
- Multiply by $\ell = 30$–35 legal moves per position on average and by 10–100 "Newton" or Nelder-Mead iterations per run.
- Model parameters $s, c, \dots$ trained to chess **Elo ratings** via nonlinear regression.
- **Cheating test** regresses on typically 6–9 games, 200–300 positions by one player. Full accuracy is vital for this test...
- ...but not so vital for the training: Large data; approximation OK.

Wider Context—Cheating Detection at Chess

- Two activities: **training** and **testing**.
- First sets for years up to 2009 totaled just over 1 million positions.
- New 2010–2014 set has 1.15 million positions. Update each year...
- *Kaggle* competition set running now $\longrightarrow$ 3 million positions.
- Multiply by $\ell = 30$–35 legal moves per position on average and by 10–100 "Newton" or Nelder-Mead iterations per run.
- Model parameters $s, c, \ldots$ trained to chess **Elo ratings** via nonlinear regression.
- **Cheating test** regresses on typically 6–9 games, 200–300 positions by one player. Full accuracy is vital for this test...
- ...but not so vital for the training: Large data; approximation OK.
- Correspondence e(s,c)–Elo comes out superbly linear under exact runs on smaller data, so—provided the approximations avoid systematic bias across Elo levels—they will help correct each other.

The Problem

1. Space of points $x = (x_1, \ldots, x_\ell)$, where ℓ is not small.

The Problem

1. Space of points $x = (x_1, \ldots, x_\ell)$, where ℓ is not small.
2. Need to evaluate $y = f(x)$ for $M =$ millions of x.

The Problem

1. Space of points $x = (x_1, \ldots, x_\ell)$, where ℓ is not small.
2. Need to evaluate $y = f(x)$ for M = millions of x.
3. Each eval of f is **expensive**. Many repetitive evals [many move situations are similar].

The Problem

1. Space of points $x = (x_1, \ldots, x_\ell)$, where ℓ is not small.
2. Need to evaluate $y = f(x)$ for $M =$ millions of x.
3. Each eval of f is **expensive**. Many repetitive evals [many move situations are similar].
4. However, the target function $\mu(y_1, \ldots, y_M)$ tolerates approximation:

The Problem

1. Space of points $x = (x_1, \ldots, x_\ell)$, where ℓ is not small.
2. Need to evaluate $y = f(x)$ for $M =$ millions of x.
3. Each eval of f is **expensive**. Many repetitive evals [many move situations are similar].
4. However, the target function $\mu(y_1, \ldots, y_M)$ tolerates approximation:
   - Could be linear: $\mu = sum_j a_j y_j$. For mean or quantile statistics.

The Problem

1. Space of points $x = (x_1, \ldots, x_\ell)$, where ℓ is not small.
2. Need to evaluate $y = f(x)$ for $M =$ millions of x.
3. Each eval of f is **expensive**. Many repetitive evals [many move situations are similar].
4. However, the target function $\mu(y_1, \ldots, y_M)$ tolerates approximation:
   - Could be linear: $\mu = sum_j a_j y_j$. For mean or quantile statistics.
   - Could be non-linear but well-behaved, e.g., logistic regression.

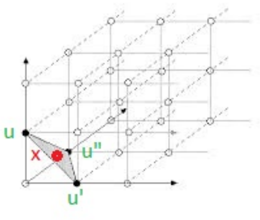
The Problem

1. Space of points $x = (x_1, \ldots, x_\ell)$, where ℓ is not small.
2. Need to evaluate $y = f(x)$ for $M =$ millions of x.
3. Each eval of f is **expensive**. Many repetitive evals [many move situations are similar].
4. However, the target function $\mu(y_1, \ldots, y_M)$ tolerates approximation:
   - Could be linear: $\mu = sum_j a_j y_j$. For mean or quantile statistics.
   - Could be non-linear but well-behaved, e.g., logistic regression.
5. Two other helps: f is smooth and bounded.

The Problem

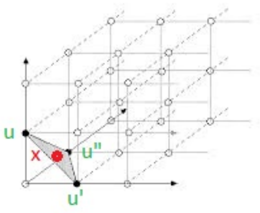
1. Space of points $x = (x_1, \ldots, x_\ell)$, where ℓ is not small.
2. Need to evaluate $y = f(x)$ for $M =$ millions of x.
3. Each eval of f is **expensive**. Many repetitive evals [many move situations are similar].
4. However, the target function $\mu(y_1, \ldots, y_M)$ tolerates approximation:
   - Could be linear: $\mu = sum_j a_j y_j$. For mean or quantile statistics.
   - Could be non-linear but well-behaved, e.g., logistic regression.
5. Two other helps: f is smooth and bounded.
6. And we need good approximation to $\mu(\cdots)$ (only) under distributions $D(x)$ controlled by a few model-specific parameters,

The Euclidean Grid Case



Pre-compute—or memoize—$f(u)$ for gridpoints u. Given x not in the grid, several ideas:

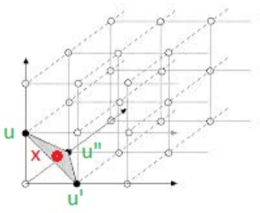
The Euclidean Grid Case



Pre-compute—or memoize—$f(u)$ for gridpoints u. Given x not in the grid, several ideas:

1. Find nearest neighbor u, use $f(u)$ as y. *Not good enough approximation.*

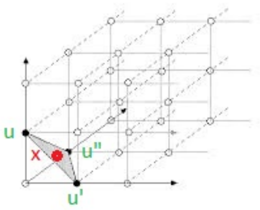
The Euclidean Grid Case



Pre-compute—or memoize—$f(u)$ for gridpoints u. Given x not in the grid, several ideas:

1. Find nearest neighbor u, use $f(u)$ as y. *Not good enough approximation.*
2. Write $x = \sum_k b_k u_k$, use $y = \sum_k b_k f(u_k)$.

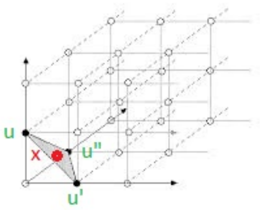
The Euclidean Grid Case



Pre-compute—or memoize—$f(u)$ for gridpoints u. Given x not in the grid, several ideas:

1. Find nearest neighbor u, use $f(u)$ as y. *Not good enough approximation.*
2. Write $x = \sum_k b_k u_k$, use $y = \sum_k b_k f(u_k)$. *Seems better. But the dimension ℓ is not small.*

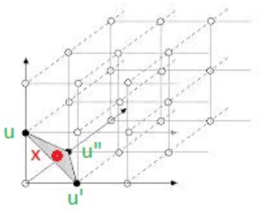
The Euclidean Grid Case



Pre-compute—or memoize—$f(u)$ for gridpoints u. Given x not in the grid, several ideas:

1. Find nearest neighbor u, use $f(u)$ as y. *Not good enough approximation.*
2. Write $x = \sum_k b_k u_k$, use $y = \sum_k b_k f(u_k)$. *Seems better. But the dimension ℓ is not small.*
3. Use any neighbor u and do Taylor expansion.

The Euclidean Grid Case



Pre-compute—or memoize—$f(u)$ for gridpoints u. Given x not in the grid, several ideas:

1. Find nearest neighbor u, use $f(u)$ as y. *Not good enough approximation.*
2. Write $x = \sum_k b_k u_k$, use $y = \sum_k b_k f(u_k)$. *Seems better. But the dimension ℓ is not small.*
3. Use any neighbor u and do Taylor expansion.

Taylor Expansion

$$f(x) = f(u) + \sum_{i=1}^{\ell}(x_i - u_i)\frac{\partial f}{\partial x_i}(u) + \frac{1}{2}\sum_{i,j}(x_i - u_i)(x_j - u_j)\frac{\partial^2 f}{\partial u_i \partial u_j} + \cdots$$

Taylor Expansion

$$f(x) = f(u) + \sum_{i=1}^{\ell} (x_i - u_i) \frac{\partial f}{\partial x_i}(u) + \frac{1}{2} \sum_{i,j} (x_i - u_i)(x_j - u_j) \frac{\partial^2 f}{\partial u_i \partial u_j} + \cdots$$

1. Given that f is fairly smooth as well as bounded, and the grid is fine enough, we can ignore quadratic and higher terms.

Taylor Expansion

$$f(x) = f(u) + \sum_{i=1}^{\ell}(x_i - u_i)\frac{\partial f}{\partial x_i}(u) + \frac{1}{2}\sum_{i,j}(x_i - u_i)(x_j - u_j)\frac{\partial^2 f}{\partial u_i \partial u_j} + \cdots$$

1. Given that f is fairly smooth as well as bounded, and the grid is fine enough, we can ignore quadratic and higher terms.
2. *Problem* is that now we need to memoize all $f_i(u) = \frac{\partial f}{\partial x_i}(u)$. 30-50x data!

Taylor Expansion

$$f(x) = f(u) + \sum_{i=1}^{\ell} (x_i - u_i) \frac{\partial f}{\partial x_i}(u) + \frac{1}{2} \sum_{i,j} (x_i - u_i)(x_j - u_j) \frac{\partial^2 f}{\partial u_i \partial u_j} + \cdots$$

1. Given that f is fairly smooth as well as bounded, and the grid is fine enough, we can ignore quadratic and higher terms.
2. *Problem* is that now we need to memoize all $f_i(u) = \frac{\partial f}{\partial x_i}(u)$. 30-50x data!
3. Main Question: Can we shortcut the partials?

Taylor Expansion

$$f(x) = f(u) + \sum_{i=1}^{\ell}(x_i - u_i)\frac{\partial f}{\partial x_i}(u) + \frac{1}{2}\sum_{i,j}(x_i - u_i)(x_j - u_j)\frac{\partial^2 f}{\partial u_i \partial u_j} + \cdots$$

1. Given that f is fairly smooth as well as bounded, and the grid is fine enough, we can ignore quadratic and higher terms.
2. *Problem* is that now we need to memoize all $f_i(u) = \frac{\partial f}{\partial x_i}(u)$. 30-50x data!
3. Main Question: Can we shortcut the partials?
4. If f were linear, obviously $\frac{\partial f}{\partial x_i} = \text{constant}$

Taylor Expansion

$$f(x) = f(u) + \sum_{i=1}^{\ell}(x_i - u_i)\frac{\partial f}{\partial x_i}(u) + \frac{1}{2}\sum_{i,j}(x_i - u_i)(x_j - u_j)\frac{\partial^2 f}{\partial u_i \partial u_j} + \cdots$$

1. Given that f is fairly smooth as well as bounded, and the grid is fine enough, we can ignore quadratic and higher terms.
2. *Problem* is that now we need to memoize all $f_i(u) = \frac{\partial f}{\partial x_i}(u)$. 30-50x data!
3. Main Question: Can we shortcut the partials?
4. If f were linear, obviously $\frac{\partial f}{\partial x_i}$ = constant = "∂Euclid."

Taylor Expansion

$$f(x) = f(u) + \sum_{i=1}^{\ell} (x_i - u_i) \frac{\partial f}{\partial x_i}(u) + \frac{1}{2} \sum_{i,j} (x_i - u_i)(x_j - u_j) \frac{\partial^2 f}{\partial u_i \partial u_j} + \cdots$$

1. Given that f is fairly smooth as well as bounded, and the grid is fine enough, we can ignore quadratic and higher terms.
2. *Problem* is that now we need to memoize all $f_i(u) = \frac{\partial f}{\partial x_i}(u)$. 30-50x data!
3. Main Question: Can we shortcut the partials?
4. If f were linear, obviously $\frac{\partial f}{\partial x_i}$ = constant = "∂Euclid."
5. What if the space Γ is warped "similarly" to f? Can we roughly use $\partial\Gamma$ in place of ∂f?

Context: Decision-Making Model at Chess

1. Domain: A set of decision-making situations t.
   Chess game turns

Context: Decision-Making Model at Chess

1. Domain: A set of decision-making situations t.
   Chess game turns
2. Inputs: Values v_i for every option at turn t.
   Computer values of moves m_i

Context: Decision-Making Model at Chess

1. Domain: A set of decision-making situations t.
   Chess game turns
2. Inputs: Values v_i for every option at turn t.
   Computer values of moves m_i
3. Parameters: $s, c, \ldots$ denoting skills and levels.
   Trained correspondence to chess Elo rating E

Context: Decision-Making Model at Chess

1. Domain: A set of decision-making situations t.
   Chess game turns
2. Inputs: Values v_i for every option at turn t.
   Computer values of moves m_i
3. Parameters: $s, c, \ldots$ denoting skills and levels.
   Trained correspondence to chess Elo rating E
4. Defines *fallible agent* $P(s, c, \ldots)$.

Context: Decision-Making Model at Chess

1. Domain: A set of decision-making situations t.
   Chess game turns
2. Inputs: Values v_i for every option at turn t.
   Computer values of moves m_i
3. Parameters: $s, c, \dots$ denoting skills and levels.
   Trained correspondence to chess Elo rating E
4. Defines *fallible agent* $P(s, c, \dots)$.
5. Main Output: Probabilities $p_{t,i}$ for $P(s, c, \dots)$ to select option i at time t.

Context: Decision-Making Model at Chess

1. Domain: A set of decision-making situations t.
   Chess game turns
2. Inputs: Values v_i for every option at turn t.
   Computer values of moves m_i
3. Parameters: $s, c, \dots$ denoting skills and levels.
   Trained correspondence to chess Elo rating E
4. Defines *fallible agent* $P(s, c, \dots)$.
5. Main Output: Probabilities $p_{t,i}$ for $P(s, c, \dots)$ to select option i at time t.
6. Derived Outputs:
   - Aggregate statistics: *move-match* MM, *average error* AE, ...
   - Projected confidence intervals for those statistics.
   - “Intrinsic Performance Ratings” (IPR’s).

How the Model Operates

- Let v_1, v_i stand for the values of the best move m_1 and ith-best move m_i, and p_1, p_i the probabilities that $P(s, c, \dots)$ will play them.

How the Model Operates

- Let v_1, v_i stand for the values of the best move m_1 and ith-best move m_i, and p_1, p_i the probabilities that $P(s, c, \dots)$ will play them.
- Given $s, c, \dots$, the model computes $x_i = g_{s,c}(v_1, v_i) =$ the **perceived inferiority** of m_i by $P(s, c, \dots)$.

How the Model Operates

- Let v_1, v_i stand for the values of the best move m_1 and ith-best move m_i, and p_1, p_i the probabilities that $P(s, c, \dots)$ will play them.
- Given $s, c, \dots$, the model computes $x_i = g_{s,c}(v_1, v_i) =$ the **perceived inferiority** of m_i by $P(s, c, \dots)$.
- Besides g, the model picks a function $h(p_i)$ on probabilities.
- Could be $h(p) = p$ (bad), log (good enough?), $H(p_i)$, *logit*...

How the Model Operates

- Let v_1, v_i stand for the values of the best move m_1 and ith-best move m_i, and p_1, p_i the probabilities that $P(s, c, \dots)$ will play them.
- Given $s, c, \dots$, the model computes $x_i = g_{s,c}(v_1, v_i) =$ the **perceived inferiority** of m_i by $P(s, c, \dots)$.
- Besides g, the model picks a function $h(p_i)$ on probabilities.
- Could be $h(p) = p$ (bad), log (good enough?), $H(p_i)$, *logit*...
- The **Main Equation:**

$$\frac{h(p_i)}{h(p_1)} = 1 - x_i.$$

How the Model Operates

- Let v_1, v_i stand for the values of the best move m_1 and ith-best move m_i, and p_1, p_i the probabilities that $P(s, c, \dots)$ will play them.
- Given $s, c, \dots$, the model computes $x_i = g_{s,c}(v_1, v_i) =$ the **perceived inferiority** of m_i by $P(s, c, \dots)$.
- Besides g, the model picks a function $h(p_i)$ on probabilities.
- Could be $h(p) = p$ (bad), log (good enough?), $H(p_i)$, *logit*...
- The **Main Equation:**

$$\frac{h(p_i)}{h(p_1)} = 1 - x_i.$$

- Ratio not difference on LHS so x_i on RHS has 0-to-1 scale.

How the Model Operates

- Let v_1, v_i stand for the values of the best move m_1 and ith-best move m_i, and p_1, p_i the probabilities that $P(s, c, \dots)$ will play them.
- Given $s, c, \dots$, the model computes $x_i = g_{s,c}(v_1, v_i) =$ the **perceived inferiority** of m_i by $P(s, c, \dots)$.
- Besides g, the model picks a function $h(p_i)$ on probabilities.
- Could be $h(p) = p$ (bad), log (good enough?), $H(p_i)$, *logit*...
- The **Main Equation:**

$$\frac{h(p_i)}{h(p_1)} = 1 - x_i.$$

- Ratio not difference on LHS so x_i on RHS has 0-to-1 scale.
- Given $(x_1, \dots, x_i, \dots, x_\ell)$, fit subject to $\sum_i p_i = 1$ to find p_1. Other p_i follow by $p_i = h^{-1}(h(p_1)(1 - x_i))$.

The Space and Its Shape

- The points $(x_1, x_2, \ldots, x_\ell)$ satisfy

$$0.0 = x_1 \leq x_2 \leq \cdots \leq x_i \leq x_{i+1} \leq \cdots \leq x_\ell \leq 1.0.$$

The Space and Its Shape

- The points $(x_1, x_2, \ldots, x_\ell)$ satisfy

$$0.0 = x_1 \leq x_2 \leq \cdots \leq x_i \leq x_{i+1} \leq \cdots \leq x_\ell \leq 1.0.$$

- Can treat the points as ordered or say $f(x_1, \ldots, x_\ell)$ is symmetric.

The Space and Its Shape

- The points $(x_1, x_2, \ldots, x_\ell)$ satisfy

$$0.0 = x_1 \leq x_2 \leq \cdots \leq x_i \leq x_{i+1} \leq \cdots \leq x_\ell \leq 1.0.$$

- Can treat the points as ordered or say $f(x_1, \ldots, x_\ell)$ is symmetric.
- Fine to pad all points to the same length ℓ by appending values 1.0.

The Space and Its Shape

- The points $(x_1, x_2, \ldots, x_\ell)$ satisfy

$$0.0 = x_1 \leq x_2 \leq \cdots \leq x_i \leq x_{i+1} \leq \cdots \leq x_\ell \leq 1.0.$$

- Can treat the points as ordered or say $f(x_1, \ldots, x_\ell)$ is symmetric.
- Fine to pad all points to the same length ℓ by appending values 1.0.
- The objective function f is given by

$$f(x) = f(x_1, \ldots, x_\ell) = p_1.$$

The Space and Its Shape

- The points $(x_1, x_2, \ldots, x_\ell)$ satisfy

$$0.0 = x_1 \leq x_2 \leq \cdots \leq x_i \leq x_{i+1} \leq \cdots \leq x_\ell \leq 1.0.$$

- Can treat the points as ordered or say $f(x_1, \ldots, x_\ell)$ is symmetric.
- Fine to pad all points to the same length ℓ by appending values 1.0.
- The objective function f is given by

$$f(x) = f(x_1, \ldots, x_\ell) = p_1.$$

- Each chess position and each $s, c, \ldots$ gives us a point x.

The Space and Its Shape

- The points $(x_1, x_2, \ldots, x_\ell)$ satisfy

$$0.0 = x_1 \leq x_2 \leq \cdots \leq x_i \leq x_{i+1} \leq \cdots \leq x_\ell \leq 1.0.$$

- Can treat the points as ordered or say $f(x_1, \ldots, x_\ell)$ is symmetric.
- Fine to pad all points to the same length ℓ by appending values 1.0.
- The objective function f is given by

$$f(x) = f(x_1, \ldots, x_\ell) = p_1.$$

- Each chess position and each $s, c, \ldots$ gives us a point x.
- We can't avoid the regression to fit $s, c, \ldots$, but we would love to avoid the iterations used to compute $f(x)$.

The Space and Its Shape

- The points $(x_1, x_2, \ldots, x_\ell)$ satisfy

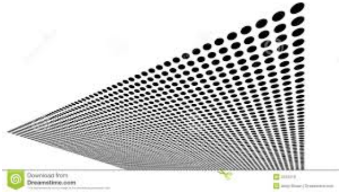
$$0.0 = x_1 \leq x_2 \leq \cdots \leq x_i \leq x_{i+1} \leq \cdots \leq x_\ell \leq 1.0.$$

- Can treat the points as ordered or say $f(x_1, \ldots, x_\ell)$ is symmetric.
- Fine to pad all points to the same length ℓ by appending values 1.0.
- The objective function f is given by

$$f(x) = f(x_1, \ldots, x_\ell) = p_1.$$

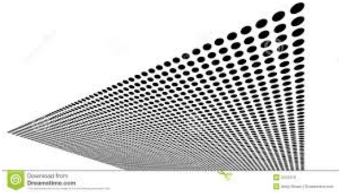
- Each chess position and each $s, c, \ldots$ gives us a point x.
- We can't avoid the regression to fit $s, c, \ldots$, but we would love to avoid the iterations used to compute $f(x)$.
- Influence on $p_1 = f(x)$ comes most from entries with low index i and low value x_i.

The Tapered Grid



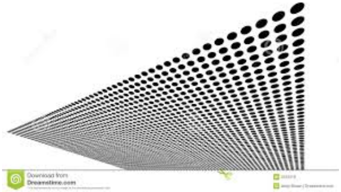
1. Grid needs higher precision near 0.0 in any coordinate, less near 1.0.

The Tapered Grid



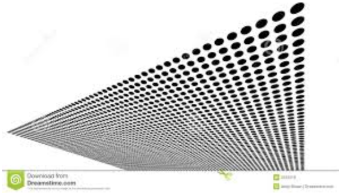
1. Grid needs higher precision near 0.0 in any coordinate, less near 1.0.
2. Grid needs less precision also for higher coordinates i.

The Tapered Grid



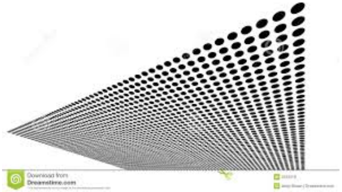
1. Grid needs higher precision near 0.0 in any coordinate, less near 1.0.
2. Grid needs less precision also for higher coordinates i.
3. Can the influence tell us about ∂f **and** help us pick a good grid neighbor u of x?

The Tapered Grid



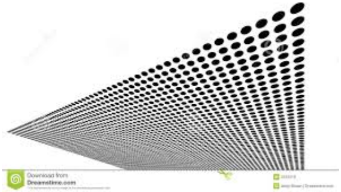
1. Grid needs higher precision near 0.0 in any coordinate, less near 1.0.
2. Grid needs less precision also for higher coordinates i.
3. Can the influence tell us about ∂f **and** help us pick a good grid neighbor u of x?
4. Given $x = (x_1, \ldots, x_\ell)$, how to define "nearest" gridpoint(s) $u = (u_1, \ldots, u_\ell)$?

The Tapered Grid



1. Grid needs higher precision near 0.0 in any coordinate, less near 1.0.
2. Grid needs less precision also for higher coordinates i.
3. Can the influence tell us about ∂f **and** help us pick a good grid neighbor u of x?
4. Given $x = (x_1, \ldots, x_\ell)$, how to define "nearest" gridpoint(s) $u = (u_1, \ldots, u_\ell)$?
5. How to define a good bounding set $u, v, \ldots$?

The Tapered Grid



1. Grid needs higher precision near 0.0 in any coordinate, less near 1.0.
2. Grid needs less precision also for higher coordinates i.
3. Can the influence tell us about ∂f **and** help us pick a good grid neighbor u of x?
4. Given $x = (x_1, \ldots, x_\ell)$, how to define "nearest" gridpoint(s) $u = (u_1, \ldots, u_\ell)$?
5. How to define a good bounding set $u, v, \ldots$?
6. How to make the computation of nearby gridpoints efficient?

Side Note and Pure-Math Problem

1. The simple function $h(p) = p$, so $p_i = \frac{a_i}{\sum_i a_i}$, works poorly.

Side Note and Pure-Math Problem

1. The simple function $h(p) = p$, so $p_i = \frac{a_i}{\sum_i a_i}$, works poorly.
2. Much better is $h(p) = \frac{1}{\log(1/p)}$.

Side Note and Pure-Math Problem

1. The simple function $h(p) = p$, so $p_i = \frac{a_i}{\sum_i a_i}$, works poorly.
2. Much better is $h(p) = \frac{1}{\log(1/p)}$.
3. Gives $p_i = p_1^{b_i}$, where $b_i = 1/a_i = \frac{1}{1-x_i}$.

Side Note and Pure-Math Problem

1. The simple function $h(p) = p$, so $p_i = \frac{a_i}{\sum_i a_i}$, works poorly.
2. Much better is $h(p) = \frac{1}{\log(1/p)}$.
3. Gives $p_i = p_1^{b_i}$, where $b_i = 1/a_i = \frac{1}{1-x_i}$.
4. **Problem:** Given y and $b_1, \ldots, b_\ell$, find p such that

$$p^{b_1} + p^{b_2} + \cdots + p^{b_\ell} = y.$$

Side Note and Pure-Math Problem

1. The simple function $h(p) = p$, so $p_i = \frac{a_i}{\sum_i a_i}$, works poorly.
2. Much better is $h(p) = \frac{1}{\log(1/p)}$.
3. Gives $p_i = p_1^{b_i}$, where $b_i = 1/a_i = \frac{1}{1-x_i}$.
4. **Problem:** Given y and $b_1, \ldots, b_\ell$, find p such that

$$p^{b_1} + p^{b_2} + \cdots + p^{b_\ell} = y.$$

5. For $\ell = 1$, simply $p = \sqrt[b]{y}$. So this generalizes taking roots.

Side Note and Pure-Math Problem

1. The simple function $h(p) = p$, so $p_i = \frac{a_i}{\sum_i a_i}$, works poorly.
2. Much better is $h(p) = \frac{1}{\log(1/p)}$.
3. Gives $p_i = p_1^{b_i}$, where $b_i = 1/a_i = \frac{1}{1-x_i}$.
4. **Problem:** Given y and $b_1, \ldots, b_\ell$, find p such that

$$p^{b_1} + p^{b_2} + \cdots + p^{b_\ell} = y.$$

5. For $\ell = 1$, simply $p = \sqrt[b]{y}$. So this generalizes taking roots.
6. We have $y = 1$ and $b_1 = 1$. Can this be solved **without** iteration?

Side Note and Pure-Math Problem

1. The simple function $h(p) = p$, so $p_i = \frac{a_i}{\sum_i a_i}$, works poorly.
2. Much better is $h(p) = \frac{1}{\log(1/p)}$.
3. Gives $p_i = p_1^{b_i}$, where $b_i = 1/a_i = \frac{1}{1-x_i}$.
4. **Problem:** Given y and $b_1, \ldots, b_\ell$, find p such that

$$p^{b_1} + p^{b_2} + \cdots + p^{b_\ell} = y.$$

5. For $\ell = 1$, simply $p = \sqrt[b]{y}$. So this generalizes taking roots.
6. We have $y = 1$ and $b_1 = 1$. Can this be solved **without** iteration?
7. Simplest case $\ell = 2$: does $g(b) = p$ such that $p + p^b = 1$ have a closed form?

Axioms and Properties

1. Suppose $x = (0.0, 0.0, 0.0, 0.0, 1.0, 1.0, \ldots, 1.0)$.
2. This means four equal-optimal moves, all others lose instantly.

Axioms and Properties

1. Suppose $x = (0.0, 0.0, 0.0, 0.0, 1.0, 1.0, \ldots, 1.0)$.
2. This means four equal-optimal moves, all others lose instantly.
3. The model will give $p_1 = p_2 = p_3 = p_4 = 0.25$, all other $p_i = 0$.

Axioms and Properties

1. Suppose $x = (0.0, 0.0, 0.0, 0.0, 1.0, 1.0, \ldots, 1.0)$.
2. This means four equal-optimal moves, all others lose instantly.
3. The model will give $p_1 = p_2 = p_3 = p_4 = 0.25$, all other $p_i = 0$.
4. **Axiom:** Influence of *poor* moves tapers off:

$$\text{As } x_i \longrightarrow 1.0, \quad \frac{\partial f}{\partial x_i} \longrightarrow 0.$$

Axioms and Properties

1. Suppose $x = (0.0, 0.0, 0.0, 0.0, 1.0, 1.0, \ldots, 1.0)$.
2. This means four equal-optimal moves, all others lose instantly.
3. The model will give $p_1 = p_2 = p_3 = p_4 = 0.25$, all other $p_i = 0$.
4. **Axiom:** Influence of *poor* moves tapers off:

$$\text{As } x_i \longrightarrow 1.0, \quad \frac{\partial f}{\partial x_i} \longrightarrow 0.$$

5. **Axiom:** Influence of *lower-ranked* moves becomes less:

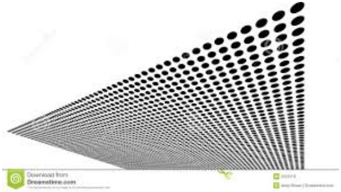
$$\text{For } i < j,\ \frac{\partial f}{\partial x_j} < \frac{\partial f}{\partial x_i}.$$

(Not quite what was meant...)

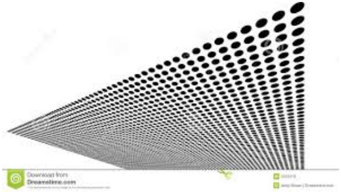
Axioms and Properties

1. Suppose $x = (0.0, 0.0, 0.0, 0.0, 1.0, 1.0, \ldots, 1.0)$.
2. This means four equal-optimal moves, all others lose instantly.
3. The model will give $p_1 = p_2 = p_3 = p_4 = 0.25$, all other $p_i = 0$.
4. **Axiom:** Influence of *poor* moves tapers off:
$$\text{As } x_i \longrightarrow 1.0, \quad \frac{\partial f}{\partial x_i} \longrightarrow 0.$$
5. **Axiom:** Influence of *lower-ranked* moves becomes less:
$$\text{For } i < j, \; \frac{\partial f}{\partial x_j} < \frac{\partial f}{\partial x_i}.$$
(Not quite what was meant. . .)
6. **"Universal Guess"**: In the first Taylor term, use
$$\frac{\partial f}{\partial x_i} \approx \frac{1}{i} a_i = \frac{1}{i}(1 - x_i).$$

Tailoring the Grid

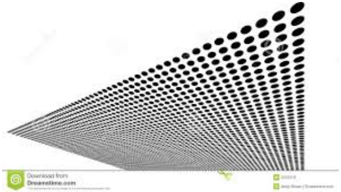


Tailoring the Grid



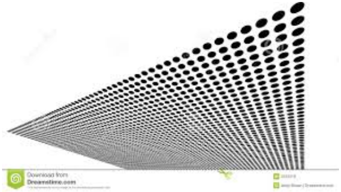
1. Define grid values u_i on each coordinate i by "tapering" (data structures by T. Biswas).

Tailoring the Grid



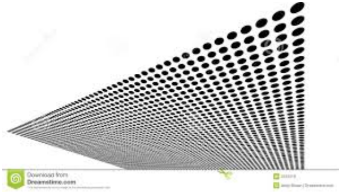
1. Define grid values u_i on each coordinate i by "tapering" (data structures by T. Biswas).
2. Given $x = (x_1, \ldots, x_\ell)$, how to define a "neighborhood" from U?

Tailoring the Grid



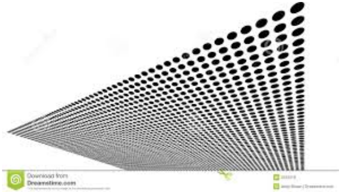
1. Define grid values u_i on each coordinate i by "tapering" (data structures by T. Biswas).
2. Given $x = (x_1, \ldots, x_\ell)$, how to define a "neighborhood" from U?
3. Define x^+ by rounding each entry x_i to next-higher u_i.

Tailoring the Grid



1. Define grid values u_i on each coordinate i by "tapering" (data structures by T. Biswas).
2. Given $x = (x_1, \ldots, x_\ell)$, how to define a "neighborhood" from U?
3. Define x^+ by rounding each entry x_i to next-higher u_i.
4. And x^- by rounding each entry x_i to next-lower u_i.

Tailoring the Grid



1. Define grid values u_i on each coordinate i by “tapering” (data structures by T. Biswas).
2. Given $x = (x_1, \ldots, x_\ell)$, how to define a “neighborhood” from U?
3. Define x^+ by rounding each entry x_i to next-higher u_i.
4. And x^- by rounding each entry x_i to next-lower u_i.
5. Neighborhood drawn from entries of x^+ and x^-.

Strategies

Given $x = (x_1, x_2, \ldots, x_\ell)$,

1. **Bounds** x^+ and x^- are well-defined by rounding each coordinate up/down to a gridpoint.

Strategies

Given $x = (x_1, x_2, \ldots, x_\ell)$,

1. **Bounds** x^+ and x^- are well-defined by rounding each coordinate up/down to a gridpoint.
2. **“Nearest Neighbor”** is defined by a *nondecreasing* sequence using only values from x^+ and x^-. Always $u_1 = 0.0 = x_1^+ = x_1^-$.

Strategies

Given $x = (x_1, x_2, \ldots, x_\ell)$,

1. **Bounds** x^+ and x^- are well-defined by rounding each coordinate up/down to a gridpoint.
2. **"Nearest Neighbor"** is defined by a *nondecreasing* sequence using only values from x^+ and x^-. Always $u_1 = 0.0 = x_1^+ = x_1^-$.
3. Start with x^-, but "round up" when the rounding-down deficiency exceeds some weighted threshold.

Strategies

Given $x = (x_1, x_2, \ldots, x_\ell)$,

1. **Bounds** x^+ and x^- are well-defined by rounding each coordinate up/down to a gridpoint.
2. **"Nearest Neighbor"** is defined by a *nondecreasing* sequence using only values from x^+ and x^-. Always $u_1 = 0.0 = x_1^+ = x_1^-$.
3. Start with x^-, but "round up" when the rounding-down deficiency exceeds some weighted threshold.
4. Once you have "rounded up," you can use same gridpoint value, but cannot "round down" again until x^- values come above it.

Strategies

Given $x = (x_1, x_2, \ldots, x_\ell)$,

1. **Bounds** x^+ and x^- are well-defined by rounding each coordinate up/down to a gridpoint.
2. **“Nearest Neighbor”** is defined by a *nondecreasing* sequence using only values from x^+ and x^-. Always $u_1 = 0.0 = x_1^+ = x_1^-$.
3. Start with x^-, but “round up” when the rounding-down deficiency exceeds some weighted threshold.
4. Once you have “rounded up,” you can use same gridpoint value, but cannot “round down” again until x^- values come above it.
5. Like a heuristic for solving Knapsack problems.

Strategies

Given $x = (x_1, x_2, \ldots, x_\ell)$,

1. **Bounds** x^+ and x^- are well-defined by rounding each coordinate up/down to a gridpoint.
2. **"Nearest Neighbor"** is defined by a *nondecreasing* sequence using only values from x^+ and x^-. Always $u_1 = 0.0 = x_1^+ = x_1^-$.
3. Start with x^-, but "round up" when the rounding-down deficiency exceeds some weighted threshold.
4. Once you have "rounded up," you can use same gridpoint value, but cannot "round down" again until x^- values come above it.
5. Like a heuristic for solving Knapsack problems.
6. Refinements which we have not yet fully explored include working backward from $i = N$ (too).

Strategies

Given $x = (x_1, x_2, \ldots, x_\ell)$,

1. **Bounds** x^+ and x^- are well-defined by rounding each coordinate up/down to a gridpoint.
2. **"Nearest Neighbor"** is defined by a *nondecreasing* sequence using only values from x^+ and x^-. Always $u_1 = 0.0 = x_1^+ = x_1^-$.
3. Start with x^-, but "round up" when the rounding-down deficiency exceeds some weighted threshold.
4. Once you have "rounded up," you can use same gridpoint value, but cannot "round down" again until x^- values come above it.
5. Like a heuristic for solving Knapsack problems.
6. Refinements which we have not yet fully explored include working backward from $i = N$ (too).
7. Combine with "universal gradient" idea, or even ignore said idea.

Results So Far. . .

1. Basic story: looks good on random data, not sure yet on real-world data...

Results So Far...

1. Basic story: looks good on random data, not sure yet on real-world data...
2. We ran experiments under a randomized distribution D_ϵ in which $r \in [x_{i-1}, 1]$ is sampled uniformly and

$$x_i = x_{i-1} + \epsilon(r - x_{i-1}) \quad \text{(capped at } x_i = 1.0\text{)}.$$

Results So Far...

1. Basic story: looks good on random data, not sure yet on real-world data...
2. We ran experiments under a randomized distribution D_ϵ in which $r \in [x_{i-1}, 1]$ is sampled uniformly and

$$x_i = x_{i-1} + \epsilon(r - x_{i-1}) \quad \text{(capped at } x_i = 1.0\text{)}.$$

3. That is, we make each move randomly slightly inferior to the previous one. We choose ϵ according to N, to make expectation $x_i \approx 1.0$ as i nears N.

Results So Far. . .

1. Basic story: looks good on random data, not sure yet on real-world data...
2. We ran experiments under a randomized distribution D_ϵ in which $r \in [x_{i-1}, 1]$ is sampled uniformly and
$$x_i = x_{i-1} + \epsilon(r - x_{i-1}) \quad \text{(capped at } x_i = 1.0\text{)}.$$
3. That is, we make each move randomly slightly inferior to the previous one. We choose ϵ according to N, to make expectation $x_i \approx 1.0$ as i nears N.
4. Results under D_ϵ are *good*: 3-place precision on $\mu(\dots)$ given 2-place to 1-place precision on grid.

Results So Far...

1. Basic story: looks good on random data, not sure yet on real-world data...
2. We ran experiments under a randomized distribution D_ϵ in which $r \in [x_{i-1}, 1]$ is sampled uniformly and

$$x_i = x_{i-1} + \epsilon(r - x_{i-1}) \quad \text{(capped at } x_i = 1.0\text{)}.$$

3. That is, we make each move randomly slightly inferior to the previous one. We choose ϵ according to N, to make expectation $x_i \approx 1.0$ as i nears N.
4. Results under D_ϵ are *good*: 3-place precision on $\mu(\dots)$ given 2-place to 1-place precision on grid.
5. Results on real chess data...

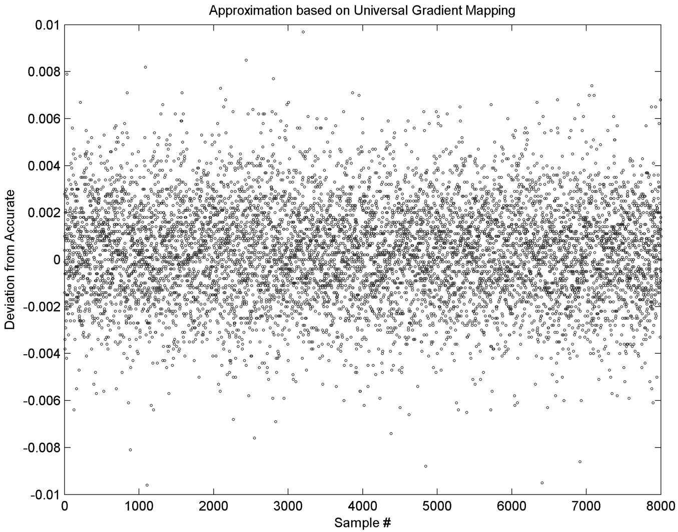
Results So Far...

1. Basic story: looks good on random data, not sure yet on real-world data...
2. We ran experiments under a randomized distribution D_ϵ in which $r \in [x_{i-1}, 1]$ is sampled uniformly and

$$x_i = x_{i-1} + \epsilon(r - x_{i-1}) \quad \text{(capped at } x_i = 1.0).$$

3. That is, we make each move randomly slightly inferior to the previous one. We choose ϵ according to N, to make expectation $x_i \approx 1.0$ as i nears N.
4. Results under D_ϵ are *good*: 3-place precision on $\mu(\dots)$ given 2-place to 1-place precision on grid.
5. Results on real chess data...still a work in progress.

Results for NN+UG



Results for Just NN