A Neural Network Monte Carlo Evaluation of Withdrawal Benefits in Variable Annuities

Hongjun Ha
(Saint Joseph's University)

ADVANCES IN PREDICTIVE ANALYTICS

December 1, 2017

Introduction

Guaranteed Minimum Withdrawal Benefit (GMWB)

- GMWB: Option to withdraw certain amount every year free of charge, even if account value decreases; in practice offered with step-ups, high-watermark guarantees, fee forgiveness, etc.
- Complex payoff profile. In particular, due to option to withdraw the resulting option valuation problem is non-European – and goes beyond optimal stopping.
- The conventional approach: Numerical methods that require a discretization of the (Markov) state space, such as finite difference schemes. When considering realistic setting?

"Curse of Dimensionality"

Goals

Set up the Problem

- Formulate Pricing Problem of GMWBs
- Specify the Dynamic Optimization Problems

Set up Strategy

- Set up the algorithm to solve the involved dynamic programming
- Explore the feasibility of the Neural Network approach for other (non-surrender) option features (beyond optimal stopping)
- Comparison to grid-based algorithm in Black-Scholes setup

Explore potential limitations...

Set-up

We consider

- d-dimensional Markov process $Y = (Y_t)_{t\in[0,T]}$ drives financial assets and mortality risk
- Possible withdrawal time : t_i, $i = 1, 2, ..., n-1$ where

$$0 = t_0 < t_1 < \cdots < t_{n-1} < t_n = T,$$

  and T is maturity of contract.
- Initial premium P invested in investment funds $S_t = S_t(Y_t)$, and the insurer has a personal (separate) account, X_t.

Set-up (Moenig & Bauer, 2015)

- Law of motion

$$X_{t_i}^{+} = \max\left(0, X_{t_i}^{-} - w_{t_i}\right), \quad i = 1, 2, ...n-1,$$
$$X_{t_i}^{-} = X_{t_{i-1}}^{+} \frac{S_{t_i}}{S_{t_{i-1}}} e^{-\phi(t_i - t_{i-1})}, \quad i = 1, 2...., n,$$
$$X_{t_0}^{+} = P,$$

  ϕ: an option fee, w_{t_i}: amount of withdrawal

- Amount of withdrawal

$$0 \leq w_{t_i} \leq \max\left(X_{t_i}^{-}, \min(g_{t_i}, G_{t_i})\right),$$

  g_{t_i}: guaranteed amount of withdrawal, G_{t_i}: guarantee account.

Set-up

- Motion of guarantee account

$$G_{t_{i+1}} = \begin{cases} \max\left(0, G_{t_i} - w_{t_i}\right), & w_{t_i} \leq g_{t_i} \\ \min\left(\max\left(0, G_{t_i} - w_{t_i}\right), \frac{X^+_{t_i}}{X^-_{t_i}} G_{t_i}\right), & w_{t_i} > g_{t_i} \end{cases}$$

  with $i = 1, 2, ..., n-1$ and $G_{t_1} = P$.

- Cash amount

$$\begin{aligned} C(t_i, w_{t_i}) &= w_{t_i} - \mathsf{fee}^I_{t_i} - \mathsf{fee}^R_{t_i}, \\ \mathsf{fee}^I_{t_i} &= ep_{t_i} \times \max(0, w_{t_i} - \min(g_{t_i}, G_{t_i})), \\ \mathsf{fee}^R_{t_i} &= pg_{t_i} \times (w_{t_i} - \mathsf{fee}^I_{t_i}) \mathbb{1}_{\{x+t_i<59.5\}}. \end{aligned}$$

- Death Benefit

$$D_{t_i} = \max(X^-_{t_i}, G_{t_i}), \quad i = 1, 2, ..., n.$$

- Survival Benefit

$$V(T) = \max\left(X^-_T,\ \min(g_T, G_T)\right).$$

Price

The policyholder maximizes the value of contract by finding "optimal" amounts of withdrawal strategy:

$$
\begin{aligned}
V(0) = \sup_{\mathcal{W}\in\mathcal{A}} \mathbb{E}^{\mathbb{P}\times\mathbb{Q}}\Bigg[& \sum_{i=1}^{(n-1)\wedge\tau} e^{-\int_0^{t_i} r_s ds} e^{-\int_0^{t_i} \mu_x(s) ds} C(t_i, w_{t_i}) \\
& + e^{-\int_0^{T} r_s ds} e^{-\int_0^{T} \mu_x(s) ds} V(T) \mathbb{1}_{\{T\geq\tau\}} \\
& + \sum_{j=1}^{n\wedge\tau} e^{-\int_0^{t_j} r_t dt} e^{-\int_0^{t_{j-1}} \mu_x(t) dt} \times \\
& \left(1 - e^{-\int_0^{t_j - t_{j-1}} \mu_{x+t_{j-1}}(t) dt}\right) D_{t_j} \Bigg| Y_0 \Bigg],
\end{aligned}
$$

where

- $\mathcal{A}$: family of all conceivable withdrawal strategies,
- $\mathbb{P}\times\mathbb{Q}$: the product measure for biometric $\times$ financial events,
- r_t : risk free interest rate, $\mu_x(t)$: force of mortality,
- τ : surrender time.

Dynamic Programming

Usually not possible to find $V(0)$ analytically

$\rightarrow$ **Dynamic Programming Principle**: For $i = n-1, ..., 1$,

$$V_{t_i}(Y_{t_i}) = \max_{w_{t_i}} C(t_i, w_{t_i}) + \mathbb{E}^{\mathbb{P}\times\mathbb{Q}}\Bigg[e^{-\int_{t_i}^{t_{i+1}} r_s ds}\Bigg\{ {}_{t_{i+1}-t_i}p_{x+t_i} V_{t_{i+1}}(Y_{t_{i+1}}) + \left(1 - {}_{t_{i+1}-t_i}p_{x+t_i}\right) D_{t_{i+1}}(Y_{t_{i+1}})\Bigg\}\Bigg| Y_{t_i}\Bigg],$$

subject to

- $0 \leq w_{t_i} \leq \max(X^-_{t_i}, \min(g_{t_i}, G_{t_i}))$
- $V_{t_n}(Y_{t_n})^* = V_{t_n}(Y_{t_n}) = \max(X^-_{t_n}, \min(g_{t_n}, G_{t_n}))$

Problem: How to approximate the **Expectation**?

Dynamic Programming

Usually not possible to find $V(0)$ analytically
$\rightarrow$ **Dynamic Programming Principle**: For $i = n-1, ..., 1$,

$$V_{t_i}(Y_{t_i}) = \max_{w_{t_i}} C(t_i, w_{t_i}) + \mathbb{E}^{\mathbb{P}\times\mathbb{Q}}\Bigg[e^{-\int_{t_i}^{t_{i+1}} r_s ds}\Bigg\{ {}_{t_{i+1}-t_i}p_{x+t_i} V_{t_{i+1}}(Y_{t_{i+1}}) + \left(1 - {}_{t_{i+1}-t_i}p_{x+t_i}\right) D_{t_{i+1}}(Y_{t_{i+1}})\Bigg\}\Bigg| Y_{t_i}\Bigg],$$

subject to

- $0 \leq w_{t_i} \leq \max(X^-_{t_i}, \min(g_{t_i}, G_{t_i}))$
- $V_{t_n}(Y_{t_n})^* = V_{t_n}(Y_{t_n}) = \max(X^-_{t_n}, \min(g_{t_n}, G_{t_n}))$

Problem: How to approximate the **Expectation**?

Neural Network

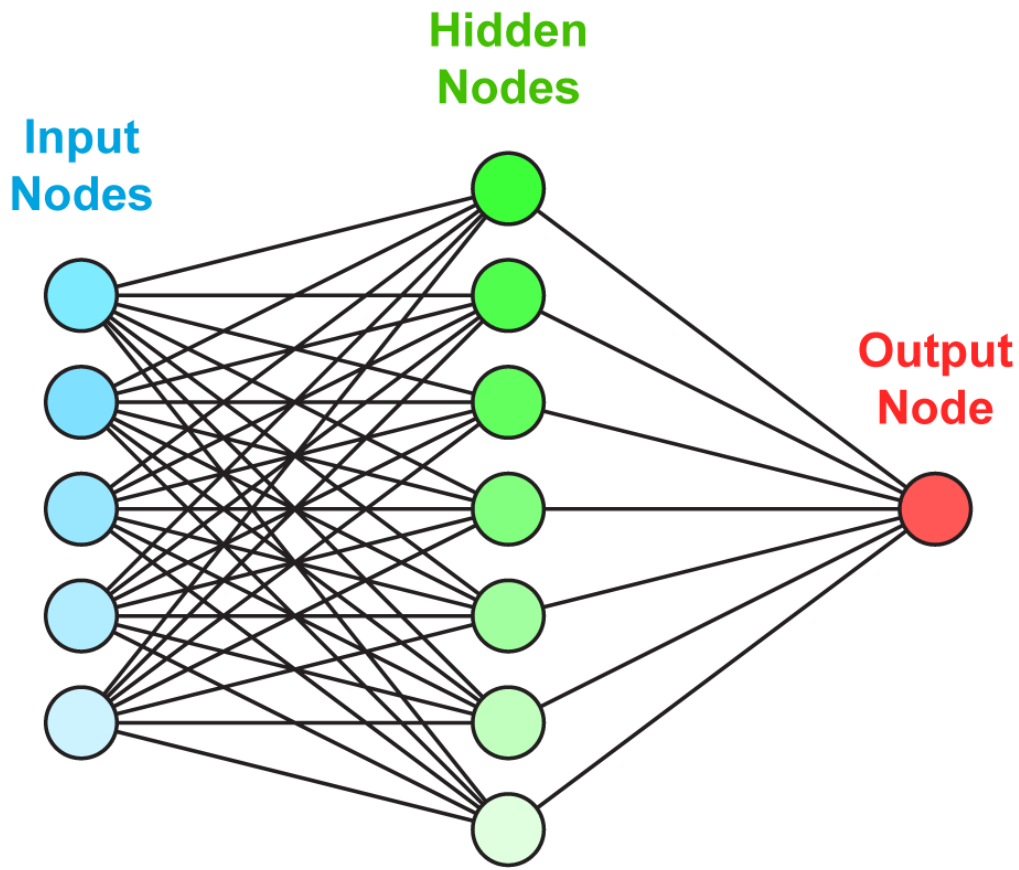

Figure: Simple Neural Networks: the input layer, a hidden layer, and the output layer

Application of NN at t_{n-1}

Assume that

- We have N sample paths of $Y_{t_{n-1}}$ such as fund, interest rate, mortality... and associated V_T and D_T.
- Generate $w_{t_{N-1}}$ which is an element of a **finite** set.
- Generate associated V_T and D_T
- For each $w_{t_{n-1}}$, calculate (noisy) actuarial present values of $V_{t_{n-1}}$ using $w_{t_{n-1}}$, D_T and V_T, denoted by $E_{t_{n-1}}$
- We will use the following set

$$(Y_{t_{n-1}}, E_{t_{n-1}})$$

as a *training set* to fit $E_{t_{n-1}}$ on perceptrons which have Y_{T-1} as inputs.

Then,

- The predicted value of $E_{t_{n-1}}$ based on a trained NN becomes an approximation of $V_{t_{n-1}}$.
- Solve the dynamic programming at t_{n-1} based on the approximated $V_{t_{n-1}}$

Algorithm

1. At $t_i, i = 1, 2, ..., n$, generate K sample paths of state variables $Y_{t_i}^{(k)}, k = 1, 2, ..., K$ and generate *random* $w_{t_i}^{(k)}$ based on $Y_{t_i}^{(k)}$.
2. At $t_i, i = n-1, n-2, ..., 1$,
   1. Divide and collect sample paths which have the same admissible set.
   2. At each set, collect sample paths which have the same withdrawal strategy.
   3. Based on the categorized training set, calculate
   $$E^{(k)} = d_{t_i}^{(k)} \left(p_{t_i}^{(k)} \tilde{V}_{t_{i+1}}^{(k)}(Y_{t_n}^{(k)}) + \left(1 - p_{t_i}^{(k)}\right) \tilde{D}_{t_{i+1}}^{(k)} \right).$$
   4. Fit $E^{(k)}$ on perceptronss and save fitting results.
   5. Solve the dynamic programming until the maturity and set
   $$\tilde{V}_{t_i}^{(k)} = \sum_{h=t_i}^{t_{n-1}} d_{t_i,h}^{(k)} p_{t_i,h}^{(k)} w_{t_h}^{*} + \sum_{h=t_{i+1}}^{t_n} d_{t_i,h}^{(k)} (1 - p_{t_i,h}^{(k)}) \tilde{D}(Y_h^{(k)})$$
   $$+ d_{t_i,t_n}^{(k)} p_{t_i,t_n}^{(k)} \tilde{V}_T^{(k)}(Y_{t_n}^{(k)})$$

Discretized solution

When implementing NN,
$\mathcal{A}_{t_i} = \{w_{t_i} | 0 \leq w_{t_i} \leq \max\left(X^-_{t_i}, \min(g_{t_i}, G_{t_i})\right)\}$ need to be discretized. We consider:

- $X^-_{t_i} \leq G_{t_i} \leq g_{t_i} \rightarrow A_{t_i} = \{0, X^-_{t_i}, G_{t_i}\}$,
- $g_{t_i} \leq G_{t_i} \leq X^-_{t_i}$:
  - $\sum_{s=t_i}^{t_n} g_s < G_{t_i} \rightarrow A_{t_i} = \left\{0, G_{t_i} - g, X^-_{t_i}\right\}$, where $g = G_{t_i} - \sum_{s=t_{i+1}}^{t_n} g_s$
  - $\sum_{s=t_i}^{t_n} g_s \geq G_{t_i} \rightarrow A_{t_i} = \{0, g_{t_i}, X^-_{t_i}\}$,
- $g \leq X^-_t \leq G_t$:
  - $\sum_{s=t_i}^{t_n} g_s < G_{t_i} \rightarrow A_{t_i} = \left\{0, g, X^-_{t_i}\right\}$, where $g = X^-_{t_i} - X^-_{t_i}/G_{t_i}\left(\sum_{s=t_{i+1}}^{t_n} g_s\right)$,
  - $g(T-t) \geq G_t \rightarrow A_{t_i} = \{0, g_{t_i}, X^-_{t_i}\}$.

Contract Under the Black-Scholes Assumption

GMWB contract	
Maturity	15
Number of withdrawal per year	1
Initial Premium (P_0)	15
ep_{t_i}	$(8\%, 7\%, ..., 2\%, 1\%, 0\%, \cdots, 0\%)$
pg_{t_i}	10%
g	1
Policyholder	
Age	55
μ	0.01
Financial Market	
Risk free rate	5%
Volatility	15%

Table: Description of GMWB contract under Black-Scholes Assumption

Considered model under Black-Scholes Assumption

State Variables: X_t and G_t

NN model: A skip-layer feedforward neural network:

$$\hat{E}(Y_t) = \alpha + \sum_{j=1}^{p} \beta_j Y_{t,j} + \sum_h \varphi_h \left(\alpha_h + \sum_{j=1}^{p} w_{j,h} Y_{t,j} \right)$$

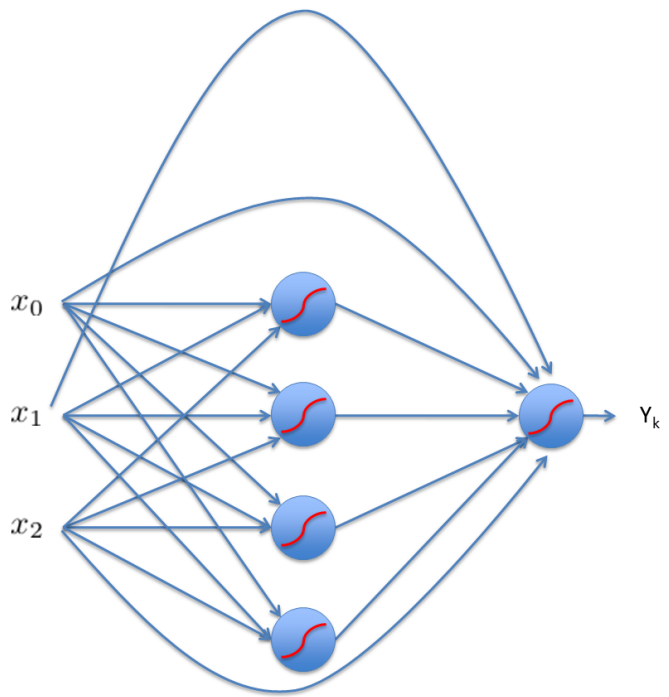

Figure: Input layers are directly sent to the output layer.

Estimation

Usually, the NN algorithm has a over-fitting problem. To find a good number of perceptrons and avoid over–fitting,

- we construct a grid of number of perceptrons and regularization parameters.
- at t_i, the network is trained at each grid point and the best combination is chosen based on RMSE.
- of course, it is time consuming. Is it possible to determine a regularization parameter automatically? (Cherkassky & Ma, 2004 for SVM)

Results

- Number of Simulation: 50,000
- Number of perceptrons and regularization parameters vary at each time
- Option Fee: 0.22%
- Grid : 14.9748
- NNMC: 15.0865
- At certain t_i close to $t_1 = 1$, NNMC seems not to be able to solve the dynamic programming due to overfitting...

Results

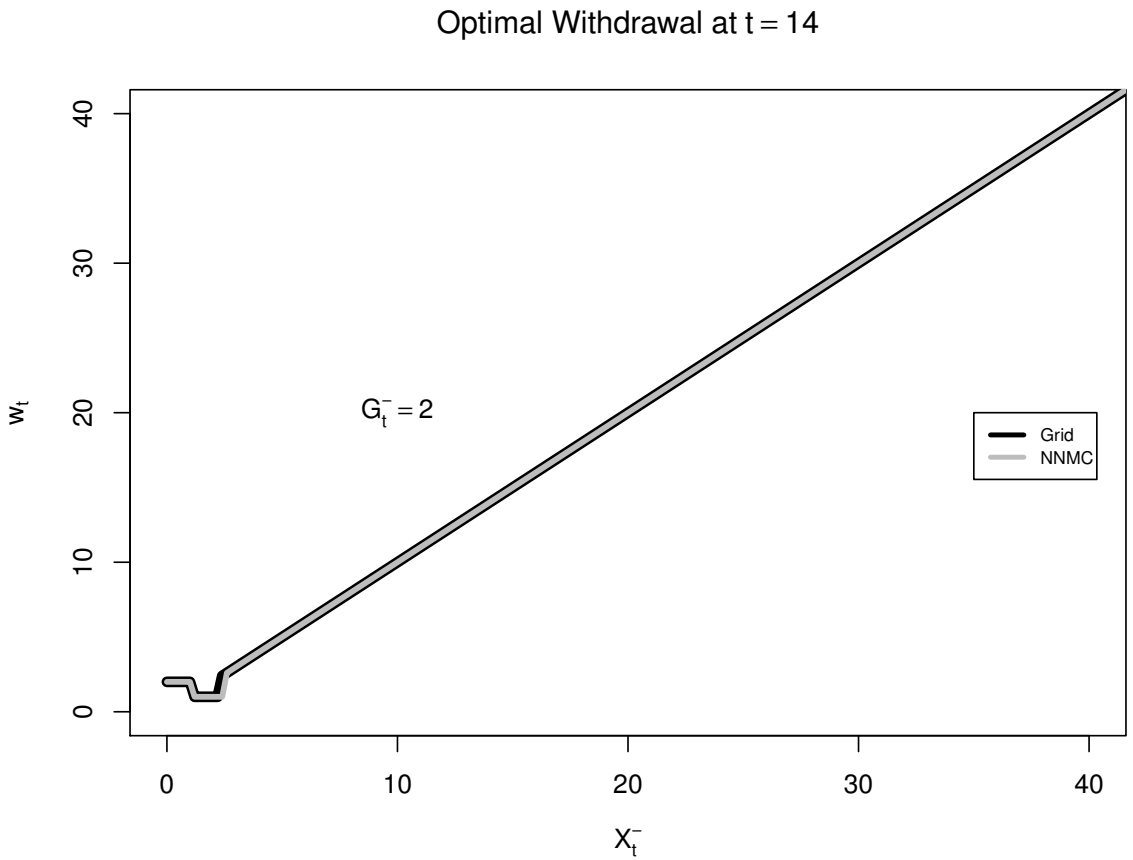

Figure: Optimal Solution via MCNN at $t = 14$ with $G_{14} = 2$. Also, GAM, SVM, MARS are good, but polynomial regressions with regularization are not useful.

Results

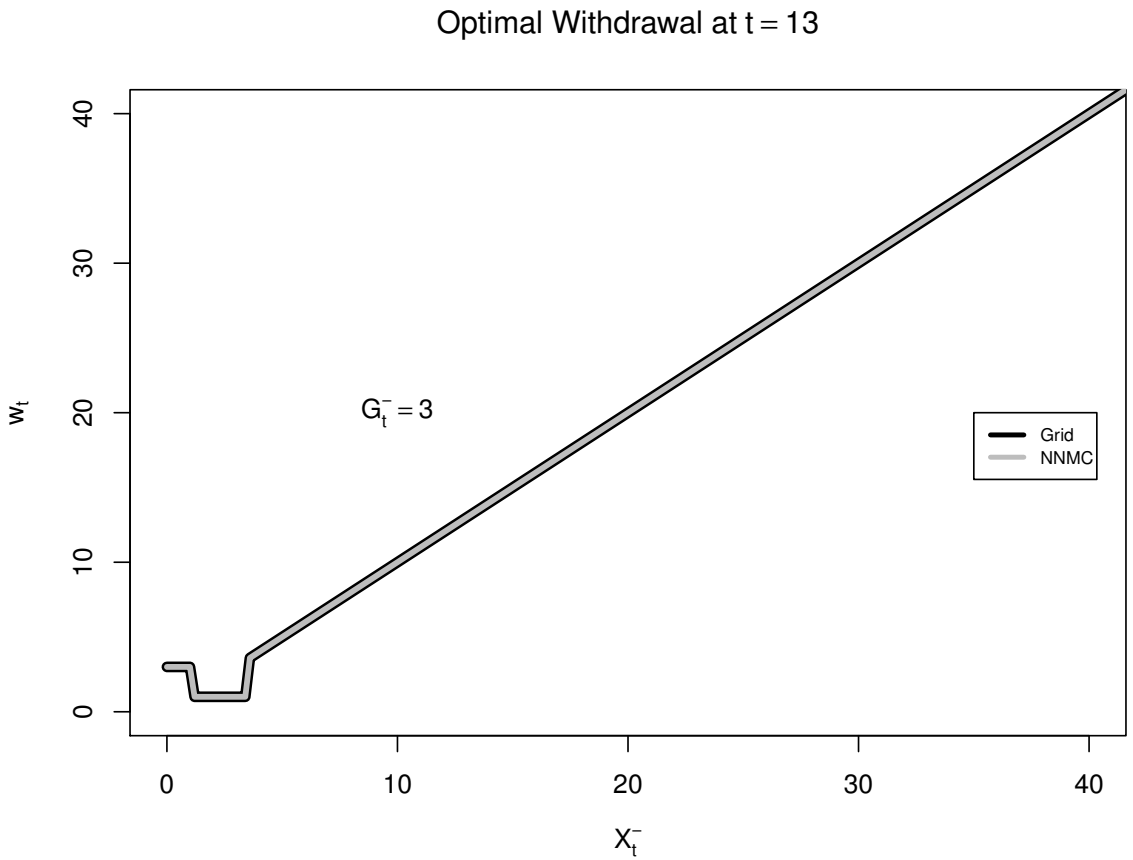

Figure: Optimal Solution via MCNN at $t = 13$ with $G_{13} = 3$. Also, GAM, SVM, MARS are good, but polynomial regressions with regularization are not useful.

Results

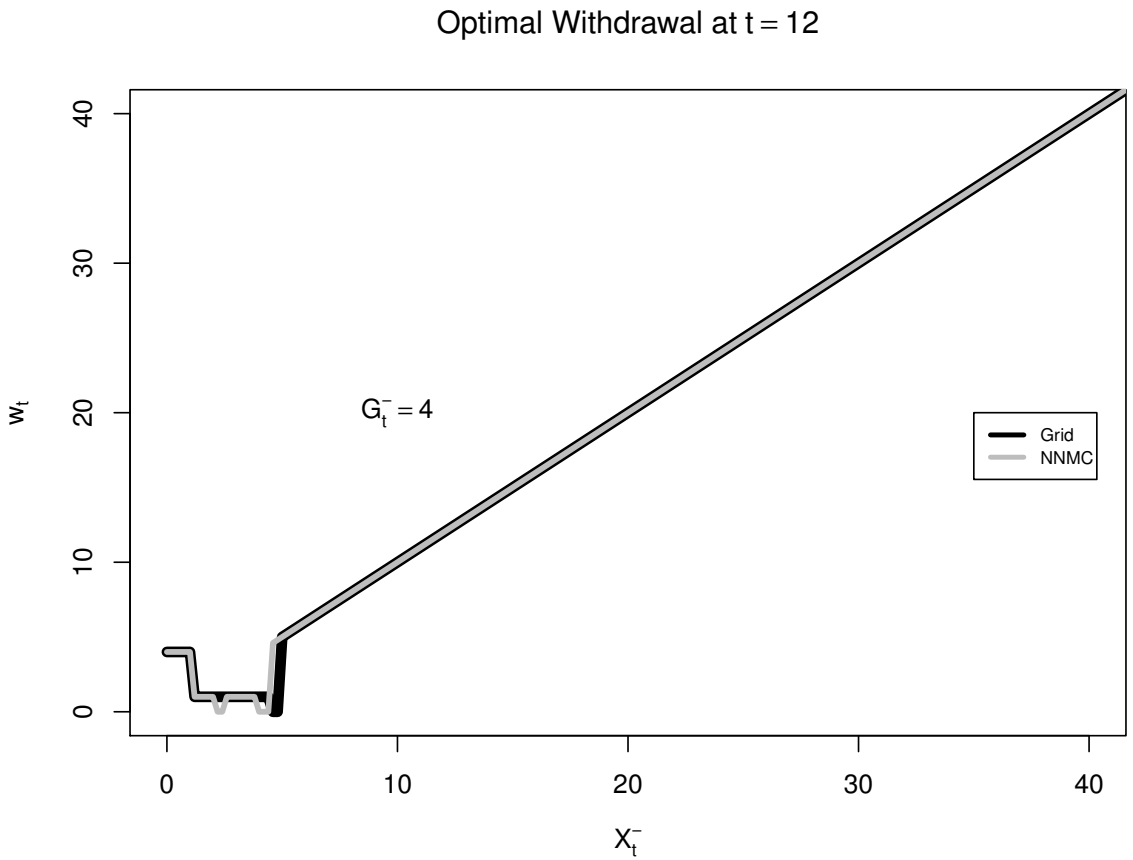

Figure: Optimal Solution via MCNN at $t = 12$ with $G_{12} = 4$. Also, GAM, SVM, MARS are good, but polynomial regressions with regularization are not useful.

Results

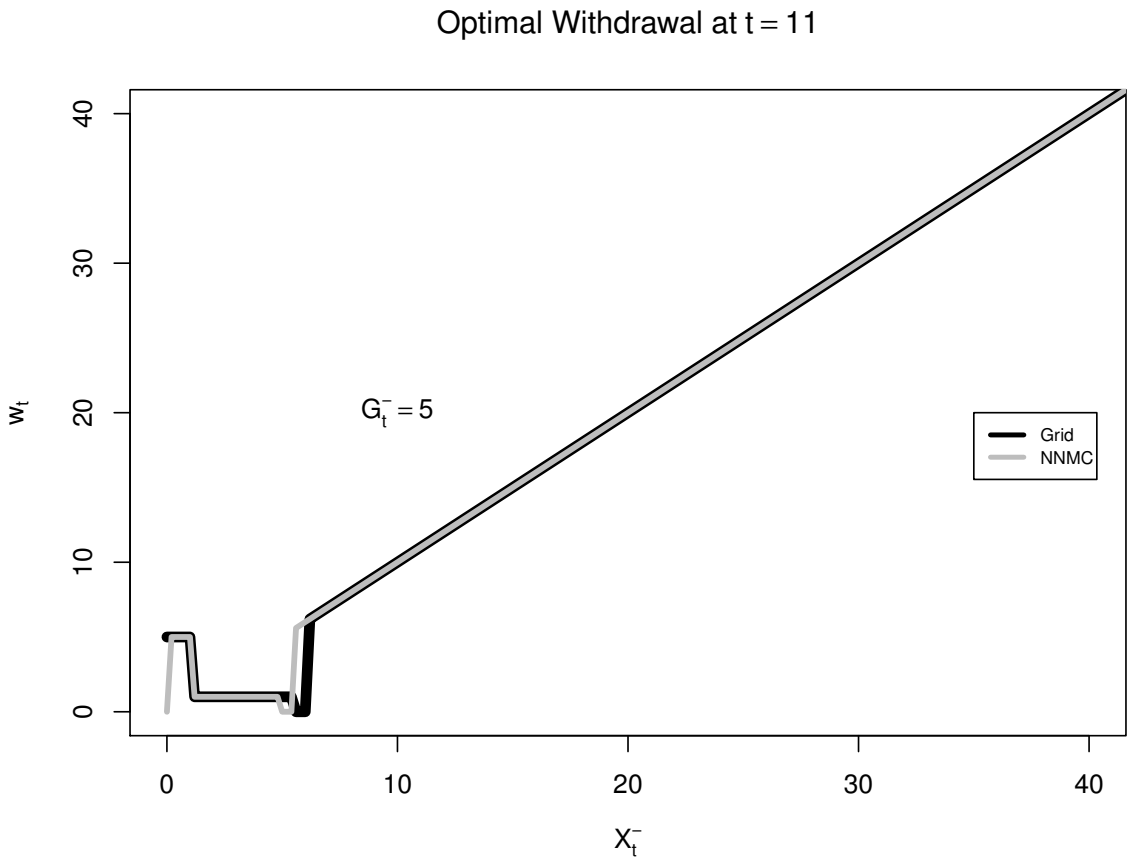

Figure: Optimal Solution via MCNN at $t = 11$ with $G_{11} = 5$. Also, GAM, SVM, MARS are good, but polynomial regressions with regularization are not useful.

Results

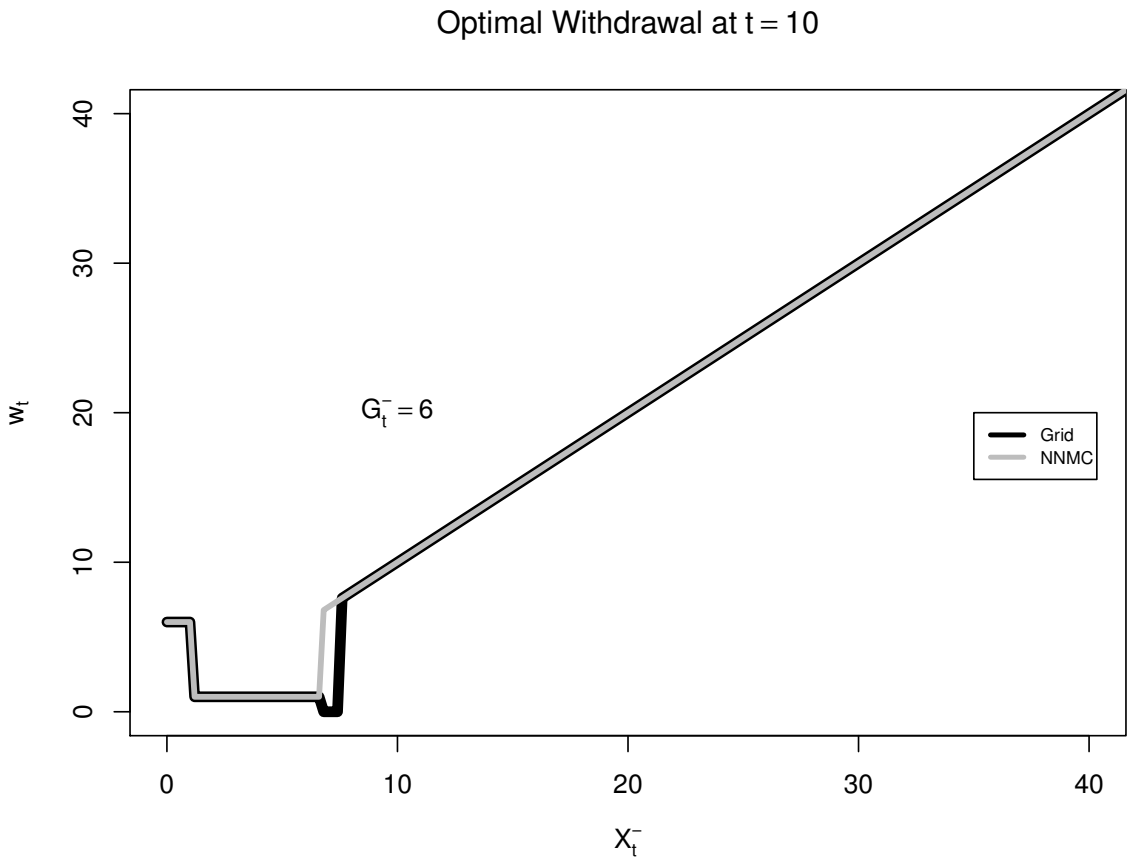

Figure: Optimal Solution via MCNN at $t = 10$ with $G_{10} = 6$. Also, GAM, SVM, MARS are good, but polynomial regressions with regularization are not useful.

Results

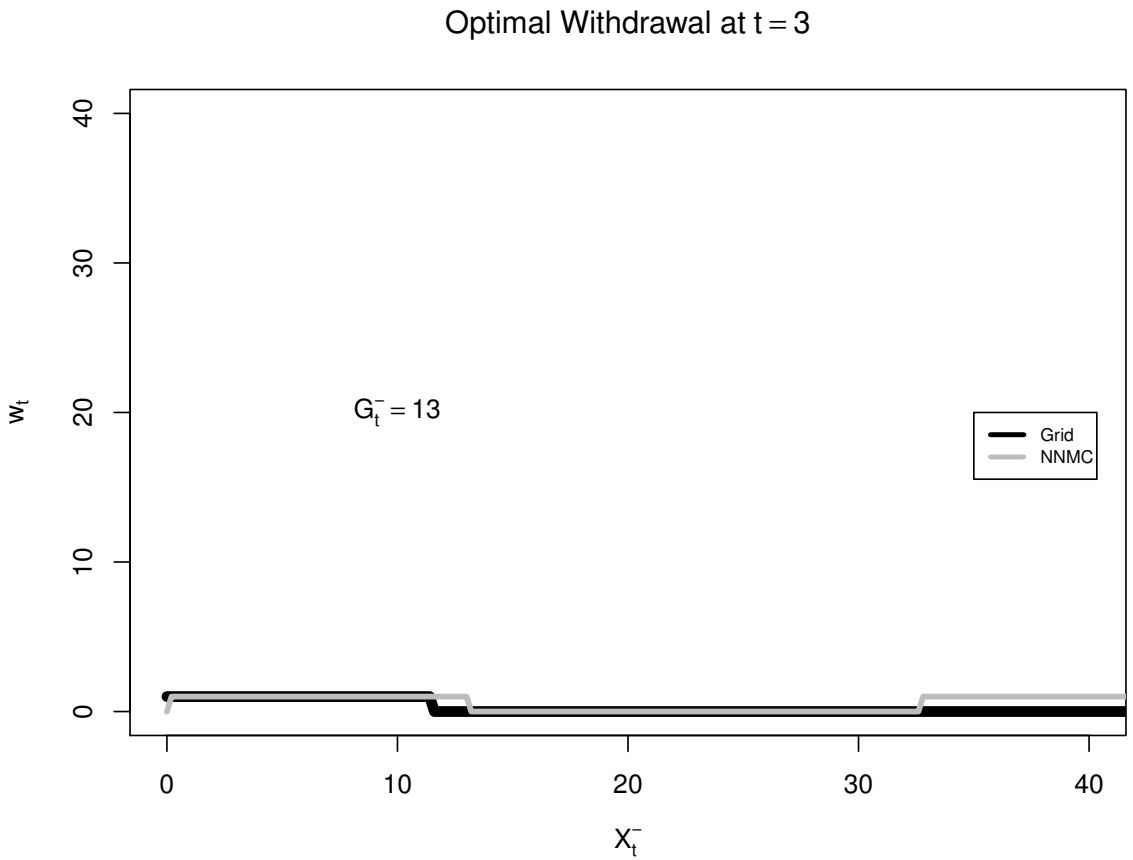

Figure: Optimal Solution via MCNN at $t = 3$ with $G_3 = 13$. We found that the Ridge regression with second order performs better.

Results

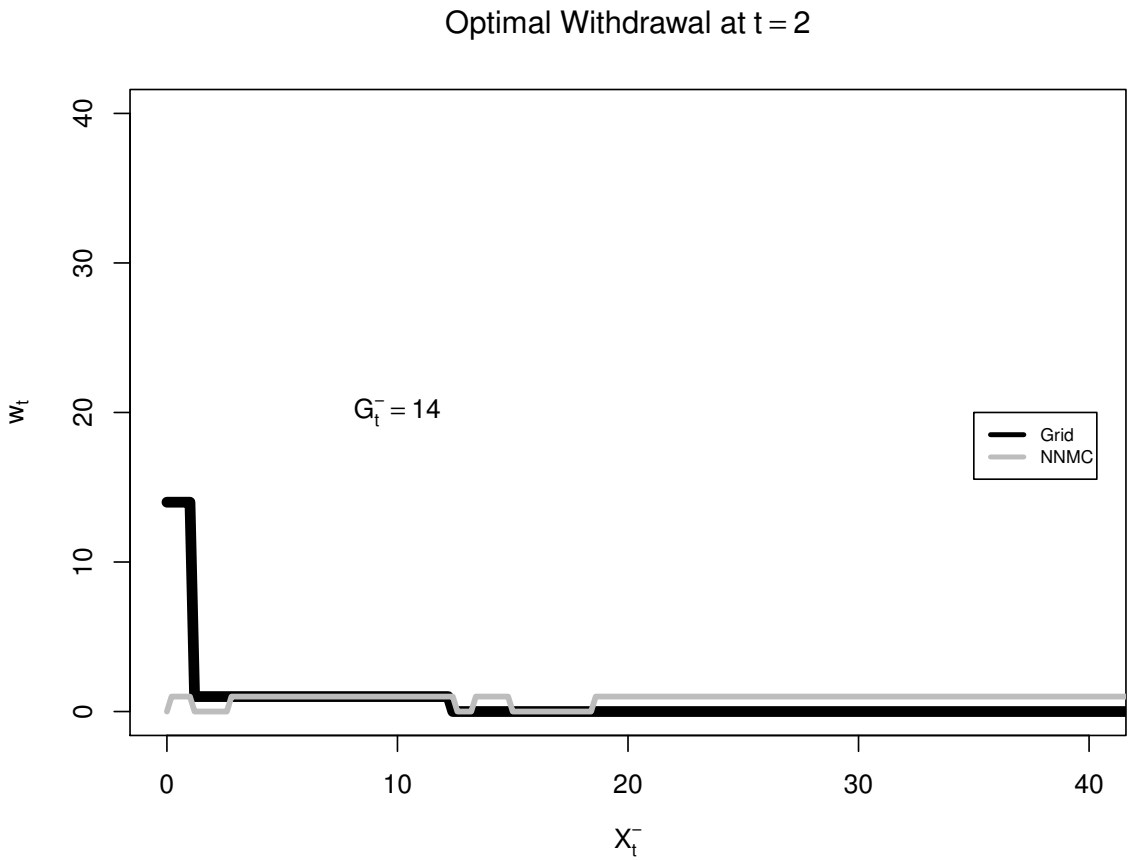

Figure: Optimal Solution via MCNN at $t = 2$ with $G_2 = 14$. We found that the Ridge regression with second order performs better.

Results

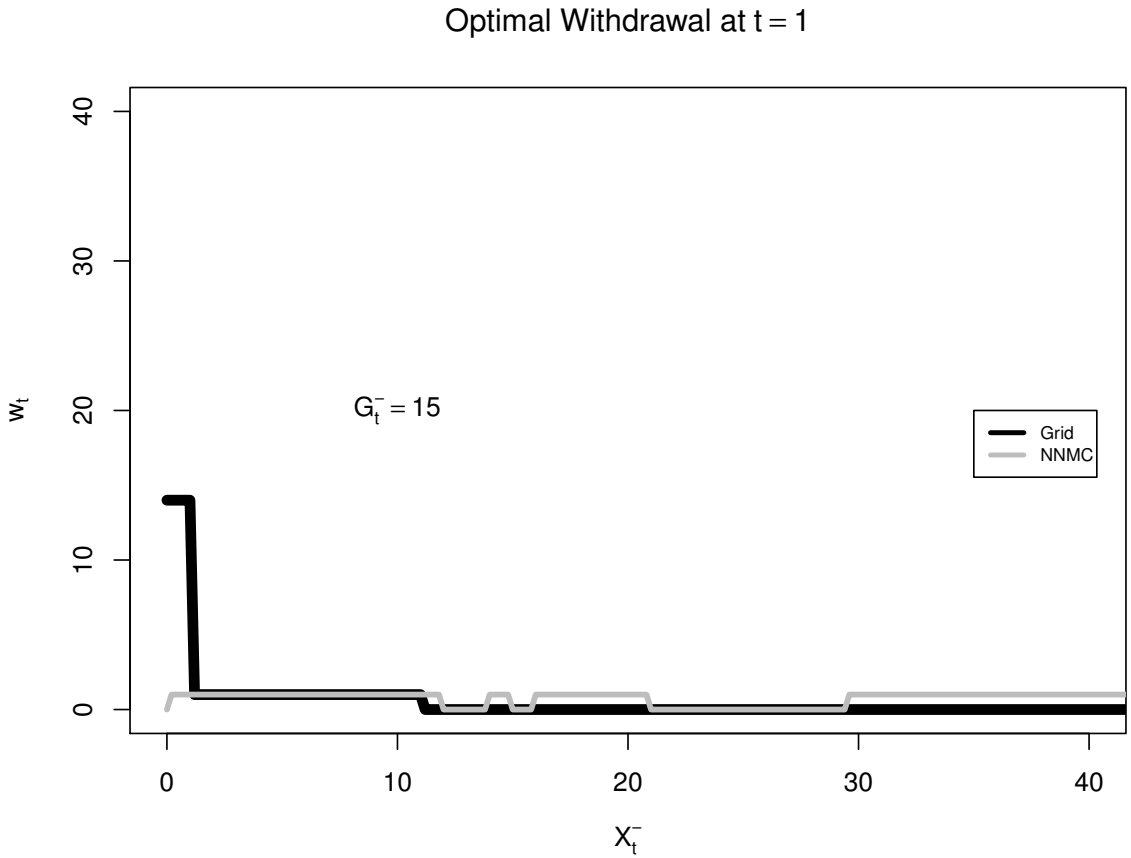

Figure: Optimal Solution via MCNN at $t = 1$ with $G_1 = 14$. We found that the Ridge regression with second order performs better.

Conclusion

- NNMC gives viable results if there is not much noise.
- How can one obtain a robust predictive model?
- *Ensembles* may be a solution to have a right prediction.
- I am implementing *ensembles* with NN, GBM, GAM and LASSO (any suggestions?) to solve dynamic programming based on h2o platform:

  `https://www.h2o.ai/`

- It is essential to consider stochastic volatility, stochastic interest rate and stochastic mortality (nontrivial application of machine learning methods).

Contact

Hongjun Ha
hha@sju.edu

Saint Joseph's University

https://sites.sju.edu/math/

Thank you!