14D006 Stochastic Models and Optimization

Play Selection in American Football

Daniel Bestard, Michael Cameron, Hans-Peter Höllwirth, Akhil Lohia

April 5, 2017

1 Motivation

We consider the dynamic programming framework to compute an optimal policy for play choice in an American football game for one drive from the view of the offensive team. In this context, the policies refer to the different plays an offensive team can make at each game situation. At every field position, the aim of the team is to choose among different possible options (run, pass, kick or punt) such that the expected reward is maximized. The reward is defined as the difference in number of points received at the end of the team's possession and the anticipated points of the opposition from our final field position.

We consider policy iteration algorithms for this task. The large number of possible states in the system (15,250), makes it hard to find the optimal solution numerically. Instead, we consider neuro-dynamic programming algorithms which approximate the optimal solution. In particular, we apply Approximate Policy Iteration (API) and Optimistic Policy Iteration (OPI) to estimate the reward-to-go function from simulated sample drives.

The project is based on the paper "Play Selection in American Football: A Case Study in Neuro-Dynamic Programming" by Patek and Bertsekas (1997). In the following sections we assume some basic understanding of the American football rules.

2 The Football Model

In our American football model we consider the score of one offense drive and add to it the expected points gained by the opposing team from our final field position. More concretely, we want to maximize the expected difference between our drive-score and the opposing team's responding drive-score (which is simply a function of our final field position). We now introduce the elements used by the dynamic programming algorithms to solve the optimization problem.

The **state** of the system $i \in S$ is described by a vector of 3 quantities: $i = [x, y, d]$:

- x = the number of yards to the opposition goal line (discrete value between 1 and 100)
- y = the number of yards to go until the next first down (discrete value between 1 and 100)
- d = down number ($\in \{1, 2, 3, 4\}$)

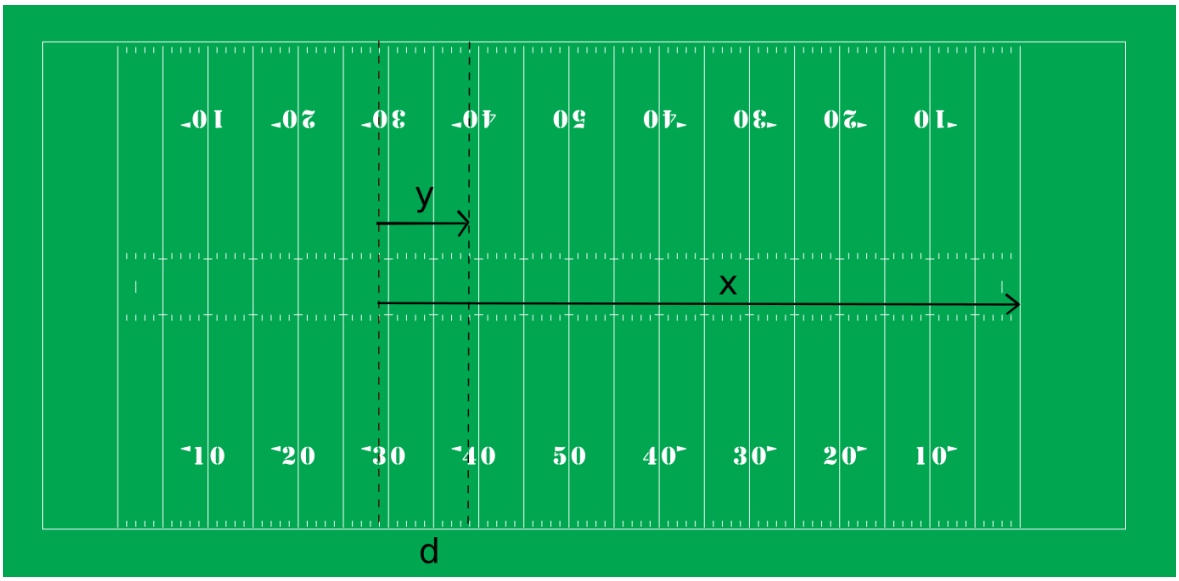

Figure 1: Illustration of state

At each state, the team can choose from one of 4 play options (**actions**) $u \in U$ with $U = \{R, P, U, K\}$. Each play is described using probability distributions, to model the yards gained. For this section we use the models set out in the original paper:

- (R)un: moves $D_p - 2$ yards forward with $D_p \sim \text{Poisson}(6)$, with probability 0.05 of a fumble.
- (P)ass: moves $D_p - 2$ yards forward with $D_p \sim \text{Poisson}(12)$, with probability 0.05 of an interception, 0.45 of incompletion, 0.05 of quarterback sack and 0.45 of completion.
- P(U)nt: moves $6D_p + 6$ yards forward with $D_p \sim \text{Poisson}(10)$
- (K)ick: successful with probability $\max(0, .95 - .95x/60)$

The set of state-action pairs determine the stationary **policy** μ. We look to choose the policy at any given state to maximise the expected reward. The **reward** of the drive is determined by the final state transition:

- Touchdown: 6.8 points (from run or pass)
- Opponent's touchdown: -6.8 points (from run or pass)
- Safety: -2.0 points (from pass or run)
- Field goal: 3.0 points (from kick)
- No score due to fumble (from run), interception (from pass), missed 4th down (from pass or run), missed field-goal (from kick) or punt

2.1 Example

A drive might look like as follows:

$$\begin{bmatrix}25\\10\\1\end{bmatrix} \longrightarrow_0^P \begin{bmatrix}17\\2\\2\end{bmatrix} \longrightarrow_0^R \begin{bmatrix}14\\10\\1\end{bmatrix} \longrightarrow_0^P \begin{bmatrix}10\\6\\2\end{bmatrix} \longrightarrow_0^P \begin{bmatrix}10\\6\\3\end{bmatrix} \longrightarrow_0^R \begin{bmatrix}8\\4\\4\end{bmatrix} \longrightarrow_3^K T$$

In this case, we start from field position $i^* = (x = 25, y = 10, d = 1)$, i.e. with a first-down 25 yards from the endzone. The policy dictates to use a (P)ass play for this particular status which in this realization yields a 8 yard improvement, thus transitioning to $i^1 = (x = 17, y = 2, d = 2)$. We continue with a mix of (P)ass and (R)un games and eventually (K)ick a field-goal at 4^{th}-down which terminates the drive and results in a reward of 3 points.

3 Dynamic Programming Formulation

In this model of the football problem, the solution can be computed directly. However, the problem can quickly become computationally infeasible when introducing other elements of the game, such as time or half yard intervals.

3.1 Optimal Solution

Our original problem has 15250 states. The rules of our model are such that at each first down there is a unique value of y associated with each value of x. Note, that the number of possible combinations is much larger than the 15250 that we claim. There is the additional constraint that it is impossible to have the number of yards to next down greater than yards to the goal line, that is, we cannot have $y > x$. This reduces the number of tates to 15250.

This is still a large problem. In their paper, Bertsekas and Patek, have carried out the computations to find the optimal policy. They have applied the dynamic programming algorithm.

$$\mu^k(i) = arg\max_{u \in U} \Big[\sum_{j \in S} p_{ij}(u)(g(i,u,j) + J^{\mu^{k-1}}(j)) \Big] \quad \forall i \in S$$

This algorithm chooses the policy u which maximises the expected reward. In the formula above, i represents the state we are currently in and j is the state we move to. $p_{ij}(u)$ are the transition probabilities, giving the probability from moving from state i to state j under action u. The reward function is $g(i,u,j)$, which is 0 in every stage except for the terminal state. This is because we only gain rewards after scoring or losing the ball. While $J^{\mu^{k-1}}(j)$ is the reward-to-go function from the state j.

Using this exact method we obtain an optimal policy for any state of the 4 downs. These optimal policies can be seen in Figure 1,1 in Bertsekas and Patek. As a benchmark, using this policy from the state $(x, y, d) = (80, 10, 1)$ (a common starting point in football) and running simulations with the optimal policy, the expected net reward is -0.9449 points. This means that if we start each drive in this state, we are expected to lose the game!

Although, this problem is computationally possible. It is still a large problem and can easily be made even bigger by introducing other variables, such as time. This is the basis for our motivation to use approximations to compute the optimal policies. Using these approximation methods it is possible to go on to compute larger problems, whereas finding the exact solution becomes computationally unfeasible easily.

3.2 Approximation with Neuro-Dynamic Programming

An important part of the implentation of this algorithm is the apporximations of the reward-to-go functions. We aim to calculate the approximate reward-to-go function for fixed policies, we can then focus on computing new policies given these approximations. Given the approximation $\widetilde{J}(i, r^{k-1})$ for a stationary policy μ^{k-1}, compute a new policy μ^k by

$$\mu^k(i) = arg\max_{u \in U} \Big[\sum_{j \in S} p_{ij}(u)(g(i,u,j) + \widetilde{J}(i, r^{k-1})) \Big] \quad \forall i \in S$$

This is very similar to the dynamic programming algorithm we had seen previously, but now we are using the approximations.

To obtain these approximations, we first run many simulated drives. Each drive will produce a reward given a randomly assigned starting position. This acts as our data to train the architecture of our approximation model. The architecture can take different forms. For instance Multi-Layered Perceptron (MLP), where a neural network is trained to give the approximate reward-to-go function, given a state. This is the method we implement in this report, but other frameworks are possible, such as the Quadritic architecture or Quadratic with Feature. More information on these two architectures can be found in the original Bertsekas and Patek paper.

4 Simulation

The approximations of the reward-to-go function $J_{u_k}(i^*)$ are obtained from simulated sample trajectories. The simulation routine takes as inputs a policy μ and a starting state i^* and then generates N sample drives. Each generated sample drive represents a realization of the following probabilistic model (probabilities in brackets):

- (R)un attempts result in either
  - (0.05) fumble
  - (0.95) run with movement $D_r - 2$ with $D_r \sim$ Poisson(6)
- (P)ass attempts result in either
  - (0.05) pass interception with movement $D_p - 2$ with $D_p \sim$ Poisson(12)
  - (0.05) sack with movement D_s with $D_s \sim$ Poisson(6)
  - (0.45) pass incomplete
  - (0.45) pass complete with movement $D_p - 2$ with $D_p \sim$ Poisson(12)
- P(U)nt attempts always result in turn-over with movement $6D_p + 6$ with $D_p \sim$ Poisson(10)

- (K)ick attempts result in either
  - (max(0,(0.95-0.95x/60)) successful field goal
  - (otherwise) missed field goal

No reward is earned in any state transition, except for the case where the drive terminates. The drive terminates in any of the following events:

- touchdown (reward: 6.8)
- successful field-goal (reward: 3.0)
- missed field-goal, punt, fumble, interception, missed 4^{th}-down (reward: 0.0)
- safety (reward: -2.0)
- opponent's touchdown (=interception in opponent's endzone) (reward: -6.8)

Once the drive is terminated, the final field position x of the drive determines the expected points gained by the opposing team: $6.8x/100$. The simulation returns N such sample drives. Each drive l is described by its status sequence (i_t^l) for $t = 1, ..., T^l$ and reward $g_{T^i}^l$.

4.1 Expected Reward

Given a set of N simulated sample trajectories for a specific policy μ, we can estimate the expected reward of this policy from a starting state i^*:

$$\widetilde{J}_{u_k}(i^*) = \frac{1}{N} \sum_{l=1}^{N} g_{T^i}^l$$

where $g_{T^i}^i$ denotes the reward of the i^{th} sample trajectory with drive length T^i.

4.2 Heuristic Benchmark

The simulations and the expected reward function can be used to yield a heuristic benchmark for the optimal play-selection policy. We use the suggested heuristic policies from the paper in order to establish the correctness of the simulation algorithm. In particular, we consider the following heuristic policies:

1. If $d = 1$ (first down): (**P**)ass
2. If $d = 2$ (second down):
   (a) If $y < 3$ (less than 3 yards to next first down): (**R**)un
   (b) If $y \geq 3$ (3 or more yards to next first down): (**P**)ass

3. If $d = 3$ (third down):
   (a) If $x < 41$ (less than 41 yards to goal):
      i. If $y < 3$ (less than 3 yards to next first down): (**P**)ass or (R)un
      ii. If $y \geq 3$ (3 or more yards to next first down): (**P**)ass or (R)un
   (b) If $x \geq 41$ (less than 41 yards to goal):
      i. If $y < 3$ (less than 3 yards to next first down): (P)ass or (**R**)un
      ii. If $y \geq 3$ (3 or more yards to next first down): (**P**)ass or (R)un
4. If $d = 4$ (forth down):
   (a) If $x < 41$ (less than 41 yards to goal):
      i. If $y < 3$ (less than 3 yards to next first down): (P)ass or (**R**)un or (K)ick
      ii. If $y \geq 3$ (3 or more yards to next first down): (P)ass or (R)un or (**K**)ick
   (b) If $x \geq 41$ (less than 41 yards to goal):
      i. If $y < 3$ (less than 3 yards to next first down): (P)ass or (**R**)un or P(U)nt
      ii. If $y \geq 3$ (3 or more yards to next first down): (P)ass or (R)un or P(**U**)nt

We estimated the expected reward for each of the $2^4 * 3^4 = 1296$ heuristic policy combinations from starting position $i^* = (x = 80, y = 10, d = 1)$ (one of the most likely starting positions in football) and found the best heuristic expected reward-to-go $J_\mu(i^*) = -1.26$. The associated policy to this reward is highlighted in bold. Note that the reward-to-go matches the heuristic result of the underlying paper, thus establishing the correctness of the simulation algorithm.

5 Approximate and Optimistic Policy Iteration

The aim now is to approximate the reward-to-go function to be used in the policy update algorithm, We have two different approaches to this problem, API and OPI. In API, we run many drive simulations over few different policies. In contrast, for OPI, we run much fewer simulations per iteration, but need more iterations to converge to a good policy choice.

We now describe in detail how the algorithms API and OPI are implemented in practice. Let N_p, N_e , N_s and N_t denote the parameters of the algorithm which are set by the user in advance. Even thought the definition of these parameters can be inferred from the algorithm, it is more clear to define them separately:

- N_p: if, for example, $N_p = 20$, then in every 20th iteration we compute the expected reward-to-go.

- N_e: number of sample trajectories that we use to compute the estimate of the reward-to-go function.
- N_s: number of sample trajectories in the in training set D_k (see later).
- N_t: maximum number of training cycles for neural network.

The implementation of the algorithm is as follows:

1. Start with an initial policy μ_0.
2. Iteratively compute
   (a) Given μ_k, if $k \in \{j \cdot N_p | j = 0, 1, 2, \}$, then generate N_e sample trajectories, each starting from i^*, to obtain an estimate of $J_{\mu_k}(i^*)$
   (b) Given the probabilistic rule for picking initial conditions, generate N_s sample trajectories and record D_k, where the last element is simply an object that stores the reward and the sequence of the vector of states. D_k is created because we use its elements in the next step of the algorithm.
   (c) We fit the neural networks to estimate the reward-to-go function and save the object in a variable that we call r^k which uses the elements of D_k. The neural networks cycle through the data N_t times.
   (d) Compute a new policy $\mu_{k+1} := G(r^k)$, where G is the greedy operator which chooses the actions at each state that are best with respect to the reward-to-go approximation given by r^k.

We set μ_0 to an heuristic policy. Step (a) and the underlying simulation function are described in chapter 4.

5.1 Generating Sample Drives

At each iteration we generate N_S sample drives by simulation. In contrast to the sample simulation for estimating the expected reward, we choose the initial state of the drive randomly. The random mechanism ensures that we encounter the most likely states with high probability. The mechansim looks as follows:

1. Set $d = 1$ (first down) with probability 0.10
   - With probability 0.25 x is chosen uniformly from 1 to 75.
   - With probability 0.75 x is chosen uniformly from 76 to 100.
   - If $x < 10$,choose $y = x$. Else choose $y = 10$
2. Set $d = 2$ (second down) with probability 0.25

- With probability 0.25 x is chosen uniformly from 1 to 50.
- With probability 0.75 x is chosen uniformly from 51 to 100.
- Yards to next first down, y, is chosen uniformly from 1 to x.

3. Set $d = 3$ (third down) with probability 0.30
   - Yards to go to goal line, x, is chosen uniformly from 1 to 100.
   - Yards to next first down, y, is chosen uniformly from 1 to x.
4. Set $d = 4$ (fourth down) with probability 0.35
   - Yards to go to goal line, x, is chosen uniformly from 1 to 100.
   - Yards to next first down, y, is chosen uniformly from 1 to x.

5.2 Multilayer Perceptron (MLP)

In our implementation, we use the multilayer perceptron (MLP) architecture to approximate the reward-to-go function $J_\mu(i)$ for policy μ from state i. We train 4 neural networks, one for each possible down d. Each neural network consists of 2 input nodes (scaled x and y:), one hidden layer with 20 nodes, and one output node (J). The architecture is depicted in figure 1. We scale the input values by a factor of $\sigma_d^x = \sigma_d^y = 0.01$ for all d to obtain values in the range $[0, 1]$. The neural networks then essentially find the least-squares estimates to the reward-to-go values $J_\mu(i)$. We use the R library *nnet* to train the neural networks. The number of training cycles is limited to N_t.

The trained neural networks will then be used to predict the reward-to-go values for all possible states $i = (x, y, d)$. We compute the predictions for all state combinations and store them for the policy update step.

5.3 Policy Update

Finally, we use the predictions for the reward-to-go function $J_\mu(i)$ from the trained neural network r^{k-1} to update the policy:

$$\mu^k(i) = G(r^k)(i) = arg\max_{u \in U} \Big[\sum_{j \in S} p_{ij}(u)(g(i,u,j) + \widetilde{J}(i, r^{k-1})) \Big] \quad \forall i \in S$$

We set the immediate reward function $g(i, u, j) = 0$ for all (i, j) pairs reward is only earned at the termination of the drive. Furthermore, we take the estimates for the reward-to-go $\widetilde{J}(i, r^{k-1})$ from the neural network output.

To compute the expected reward, all that is left is finding the transition probabilities $p_{ij}(u)$ from state i to state j with action u. Note that the transition matrix $P(u) = (p_{ij}(u)$

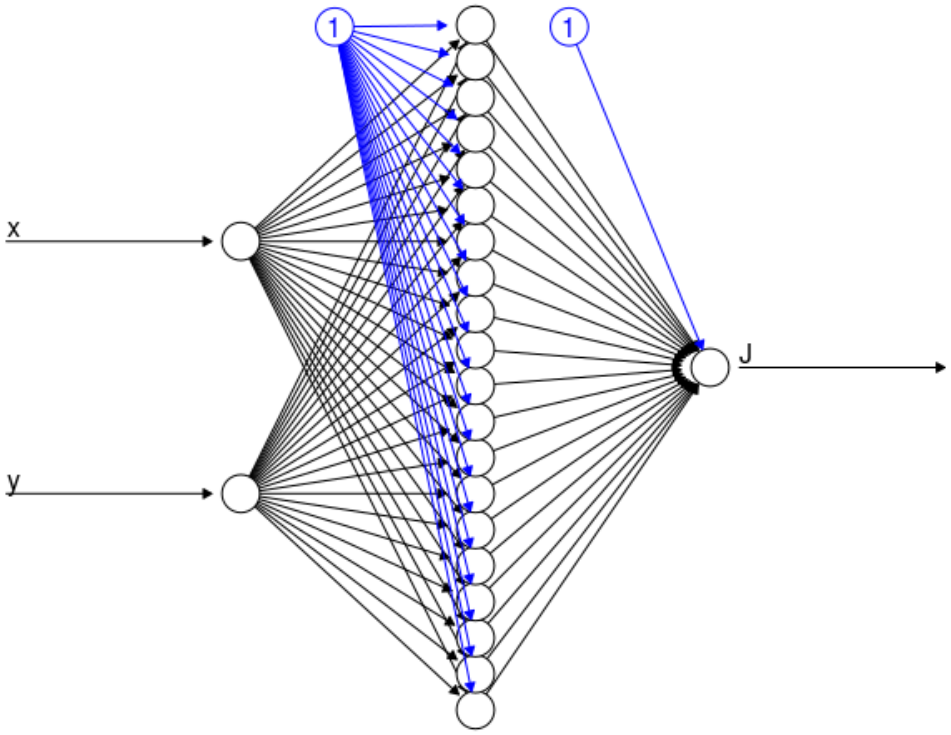

Figure 2: Architecture of neural network with one hidden layer ($R = 20$ nodes)

is of dimension $15,250 \times 15,250$! Unfortunately, the paper does not mention how to derive the transition probabilities. We decided to use the generated sample data to empirically estimate the transition probabilities. Note that this approach gives us a sparse transition matrix, even for large sample sizes N_S. However, this sparsity allows us to compute the sum over all possible transitions effectively.

In a final step, we compare the expected reward of a state i with each action $u \in \{P, R, U, K\}$ and set $\mu^k(i)$ to the action that gives the maximum expected reward.

6 Results

The football case study provides an effective testbed to evaluate the NDP framework. We implement the API and OPI algorithms using the Multilayer Perceptron architecture. In this case, the multilayer perceptron basically boils down to a neural network with one hidden layer of 20 nodes. The neural network trained on simulated drives from random starting points allow to approximate the reward-to-go from a given state instead of having

to take a recursive path of calculating the expected reward. This is the major idea of the study which differentiates the reinforcement learning approach from usual dynamic programming.

The football model represents a challenge for the approximate methods and seems to have characteristics of truly large scale examples. Apart from that, since we know the optimal results, it allows to evaluate the performance of our model. Finally, football is intuitive. This aids in the implementation and de-bugging of the algorithms and also provides a means for interpreting the results.

6.1 Heuristic Policy Results

We evaluate a large set of heuristic policies from a given starting point to find the one which gives the best results based on a large number of simulated drives. The best heuristic policy gives an expected reward of -1.2653 which is about .3 points worse than the optimal results.

6.2 API and OPI Results

We try a wide range of number of samples and iterations to get the updated policies for a given starting point. Unfortunately we do not achieve a convergence in terms of the expected reward from the given starting state. We obtain noisy expected scores in the same ballpark as Patek and Bertsekas albiet with quite a bit of variation. Possible ways we could think of to gain convergence include the following:

- Continuously update the same neural network using new generated samples instead of training new models at every iteration. This should probably provide us with better approximates of the expected reward to go.
- The calculation of transition probabilities can be done in a more sophisticated manner. Since we know the probabilistic results of every action taken at a given state, we should be able to calculate the expected state to which we will transition instead of considering the possibilities of all possible transitions.
- We can modify the architecture of the model we use to approximate the reward to go from a given state. Currently we use a neural network with 1 hidden layer and 20 nodes. A deeper network with more layers or a different architecture like the quadratic or recursive one discussed by Patek and Bertsekas in their could result in less noisy approximate rewards from a given state.