Play Selection in American Football

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

April 4, 2017

1 Motivation

In the following sections we assume some basic understanding of the American football rules. We use a dynamic programming algorithm to develop an optimal policy for play choice in an American football game. In this context, the policies refer to the different plays an American football team can make at each game situation. At every field postition, the aim of the team is to choose among different possible options, such as run, pass, kick and punt, which will be explained in more detail in the next section, the one which increases the expected reward the most. The reward is the difference in number of points recieved at the end of our possesion and the anticipated points of the opposition from our final field position. Each option gives the team different advantages at different risks. Dynamic programming works backwards to allow us, given any field position, to make the optimal policy choice.
In this problem, there is a large number of possible states, 15250, which leads to computational difficulties. To solve this, we use simulations to replace the exact reward-to-go function with an approximation that makes the optimisation problem feasible to solve. Approximate Policy Iteration (API) and Optimistic Policy Iteration (OPI) to calculate this approximation.

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 optimisation 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 ($\in \{1, 2, 3, 4\}$)

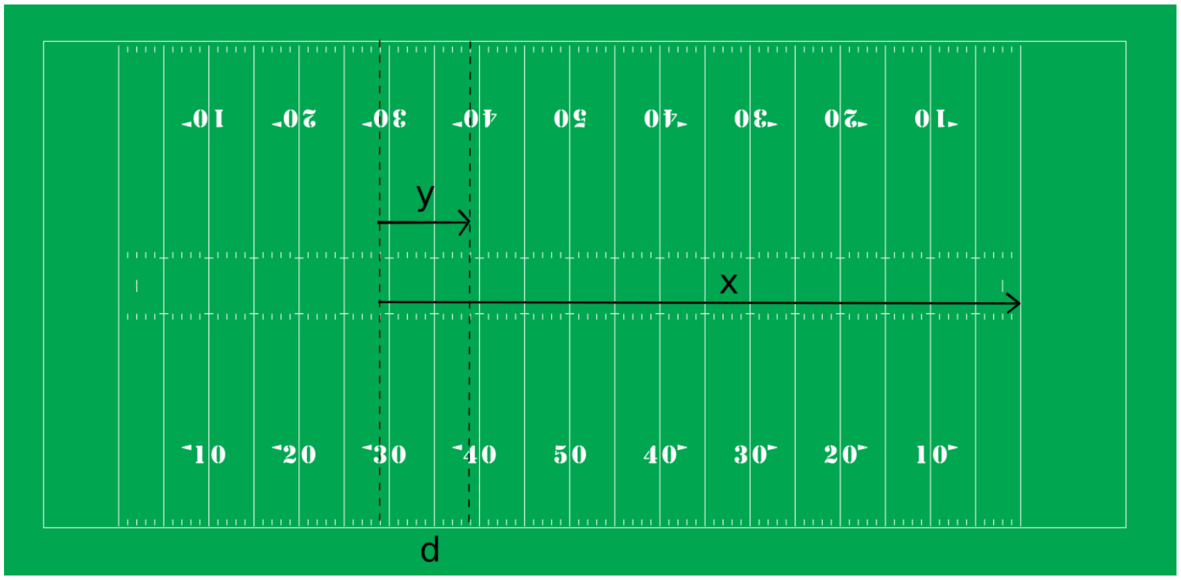

Figure 1: Illustration of state

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 probablility distributions, to model the yards gained. For this section we use the models set out in Bertsekas and Patek's article:

- (R)un: moves $D_p - 2$ yards forward with $D_p \sim$ Poisson(6), with probability 0.05 of a fumble.
- (P)ass: moves $D_p - 2$ yards forward with $D_p \sim$ 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 $D_p \sim$ 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

Once the inputs of the dynamic programming algorithm have been defined, the next step is construct the algorithm itself.

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 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 is computationally tractable, so we can apply 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 been in Figure 1,1 in Bertsekas and Patek. As a benchmark, using this policy from the state $(x, y, d) = (80, 10, 1)$ 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!

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.

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

The drive terminates in any of the following events:
The simulation returns N 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

Rhe 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. While in contrast, for OPI, we run much fewer simulations, but over much more policy choices.

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: number of times we train the neural network.

The implementation of the algorithm is as follows:

1. Start an initial policy μ_0.
2. 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 $J_{\mu_k}(i^*)$
3. 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.
4. 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.
5. 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.

5.1 Multilayer Perceptron (MLP)

As stated ealier in the report, we can replace the exact values of the reward-to-go function with approximates. To obtain these estimates we can use a trained neural network. The network has 2 inputs x and y, one hidden layer with 20 nodes, and is trained on the exact output J. We construct 4 of these networks, one for each possible down. For each of these neural networks we input the feature vector $\xi^{\Sigma}(i) = (\sigma_d^x x_i, \sigma_d^y y_i)$, In the football example we set $\sigma_d^x = \sigma_d^y = 0.01$ to ensure that all elements of $\xi^{\Sigma}(i)$ are in $[0, 1]$. This now allows us to get an estimate for the reward-to-go function from any state, by inputting x and y in the feature vectors into the correct neural network depending on the down.

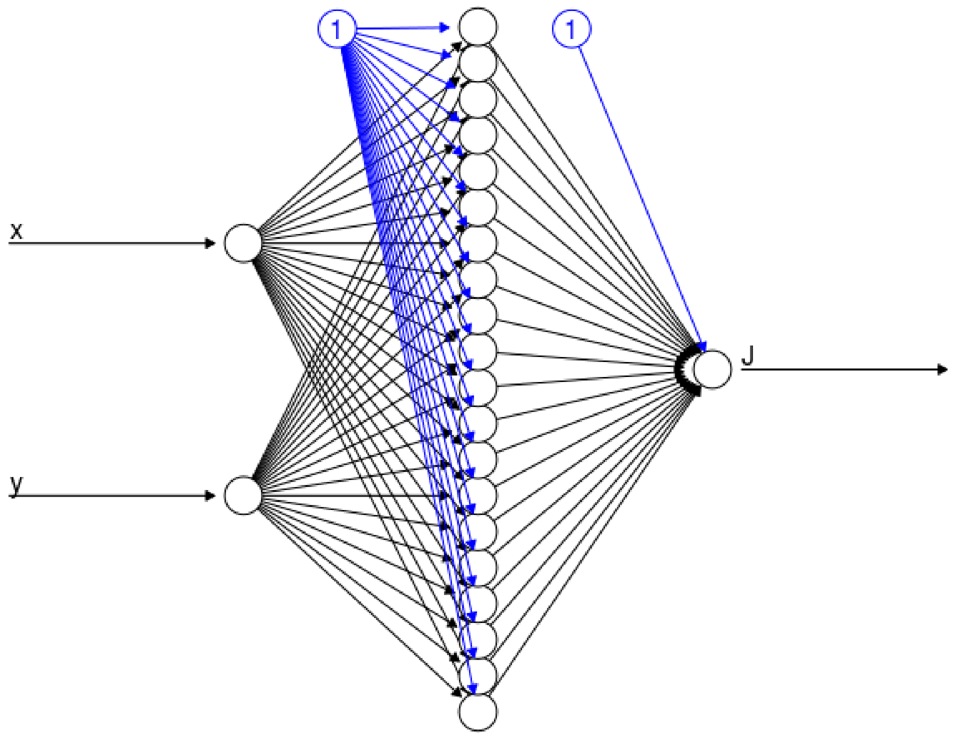

Figure 2: Architecture of neural network with one hidden layer ($R = 20$ nodes)

5.2 Policy Update

Now that we can calculate the approximate reward-to-go function for fixed policies, we can now focus on computing new policies given these approximations. Both API and OPI follow the same outline, given the approximation $\widetilde{J}(i, r^{k-1})$ for a stationary policy μ^{k-1}, compute a new policy μ^k by

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

This is very similar to the dynamic programming algorithm we had seen previously, but now we are using the approximations.
To carry out this update, the transition probabillities must be calculated as well as the approximations.

6 Results