Stochastic dual dynamic programming

Un algoritmo de programación dinámica aproximada

Andrés Ferragut

Universidad ORT Uruguay

Proyecto UTE-FJR-UDELAR-ORT
Optimización en la red eléctrica.

Julio 2017

Contenido

Contenido

Objetivos de la charla

- Analizar el problema de “curse of dimensionality” en Stochastic Dynamic programming.
- Analizar un algoritmo de solución aproximada: Stochastic Dual Dynamic Programming (SDDP).
- Discutir su implementación en software.

Modelo general

- Ventana de tiempo: $k \in \{0, \dots, N\}$.
- En cada k:
  - El estado del sistema es $x_k \in \mathcal{X}$.
  - El control es $u_k \in \mathcal{U}$.
  - Hay una perturbación w_k (aleatoria, independiente en cada k).
- Dinámica del sistema:

$$x_{k+1} = f_k(x_k, u_k, w_k) \qquad k = 0, \dots, N-1.$$

- Costos:
  - En cada paso $g_k(x_k, u_k, w_k)$ (running costs).
  - En el estado terminal $g_N(x_N)$ (terminal cost para horizonte finito).

Problema: stochastic dynamic programming

Mínimo costo medio

Hallar una regla o política $\pi = \{\mu_0, \ldots, \mu_{N-1}\}$, tal que se logre:

$$\min_{\pi} E_w \left[\sum_{k=0}^{N-1} g_k(x_k, \mu_k(x_k), w_k) + g_N(x_N) \right]$$

sujeto a que $x_{k+1} = f_k(x_k, \mu_k(x_k), w_k)$.

Observación: El resultado del problema es una política y no una trayectoria de control. Esto permite reaccionar al estado del sistema y trabajar en "lazo cerrado". Debería ser mejor que una optimización a priori, pero también más costoso computacionalmente.

Ejemplo: control de stock

Empresa que provee productos a clientes, los cuales demandan una cantidad aleatoria.

- Estado: x_k, stock al comienzo del intervalo k ($x_k < 0 \Rightarrow$ demanda insatisfecha).
- Control: $u_k \geqslant 0$, pedido de reposición de stock en k (suponemos entrega inmediata).
- Perturbación: $w_k \geqslant 0$, demanda que llega en el intervalo k.

Dinámica:

$$x_{k+1} = x_k + u_k - w_k = f(x_k, u_k, w_k)$$

Problemas del DP tradicional

Curse of dimensionality

- Para aplicar lo anterior, necesito en general *discretizar* el estado x.
- Esto lleva a que la cantidad (finita) de estados crezca exponencialmente.

Ejemplo [Pereira, 1991]

- Lagos de hydro, cada uno con 2 variables (cota y aportes).
- Discretizo en $m = 20$ niveles cada variable $\Rightarrow m^{2n}$ estados en cada etapa.
  - 1 lago $\longrightarrow 20^2 = 400$ estados.
  - 2 lagos $\longrightarrow 20^4 = 1.6 \times 10^5$ estados.
  - 3 lagos $\longrightarrow 20^6 = 6.4 \times 10^7$ estados.
  - 4 lagos $\longrightarrow 20^8 = 2.5 \times 10^{10}$ estados.
  - 5 lagos $\longrightarrow 20^{10} = 1 \times 10^{13}$ estados.

Approximate dynamic programming

- Idea: construir una aproximación manejable de la función de valor $V(x)$.
- Por ejemplo, si V es convexa, aproximarla por una función lineal a tramos.
- Minimizar esta función en lugar de minimizar V, y mejorar la aproximación en cada paso generando nuevos tramos lineales.

Algoritmo de Kelley

Objetivo: Miniizar $V : \mathbb{R}^n \to \mathbb{R}$ convexa en un conjunto compacto $\mathcal{U}$.

Algoritmo:

1. Condición inicial u_0. $k = 0$, $V_0 = -\infty$.
2. Calculo un subgradiente $\lambda_k \in \partial V(u_k)$.
3. Actualizo la aproximación:

$$V_{k+1}(u) = \max\{V_k(u), V(u_k) + \lambda_k^T(u - u_k)\}.$$

4. Minimizo V_{k+1} y elijo $u_{k+1} = \arg\min_u\{V_{k+1}(u)\}$
5. $k \to k + 1$ y vuelvo al paso 2.

Contenido

SDDP: marco de aplicación

- Problema de optimización multi-etapa con horizonte finito.
- Estado y control continuo, finito-dimensionales ($x \in \mathbb{R}^n$, $u \in \mathbb{R}^m$).
- Costos convexos, dinámica lineal.
- Ruido independiente y discreto (escenarios).

Caso determinístico

Problema (control óptimo determinístico)

$$\min_{u} \sum_{k=0}^{N-1} g_k(x_k, u_k) + g_N(x_N)$$

$$\text{sujeto a:} \quad x_{k+1} = f_k(x_k, u_k)$$

- $x_n \in \mathcal{X}$ es el estado.
- $u_n \in \mathcal{U}$ es el control al comienzo del intervalo.

Hipótesis:

- $(x, u) \to f_k(x, u)$ es afín.
- $\mathcal{U}, \mathcal{X}$ compactos.
- Los costos $g_k(x, u)$ y $g_N(x)$ son convexos.

Caso determinístico: dynamic programming

Algoritmo de Bellman

$$V_N(x) = g_N(x),$$
$$V_k(x) = \min_{u_k}\{g_k(x,u_k) + V_{k+1}(f_k(x,u_k))\} = T_k(V_{k+1})(x).$$

Aquí T_k es el operador de Bellman:

$$[T_k(V)](x) = \min_{u_k}\{g_k(x,u_k) + V(f_k(x,u_k))\}.$$

La política óptima es el u_k que alcanza el mínimo en cada paso del algoritmo.

Propiedades del operador de Bellman

- Monotonía:
$$V(x) \leqslant \tilde{V}(x)\ \forall x \Rightarrow [T(V)](x) \leqslant [T(\tilde{V})](x)\ \forall x.$$
- Convexidad:
Si g es convexa en (x, u), V es convexa y f es afín entonces:
$$x \mapsto [T(V)](x) \quad \text{es convexa.}$$
- Funciones lineales a tramos:
Si V es lineal a tramos, g es lineal a tramos y f es afín entonces:
$$x \mapsto [T(V)](x) \quad \text{es lineal a tramos.}$$

Propiedad de dualidad

- Sea $J : \mathcal{X} \times \mathcal{U} \to \mathbb{R}$ convexa en (x, u) y sea:

$$\phi(x) = \min_u J(x, u)$$

- Puedo obtener un subgradiente $\lambda \in \partial\phi(x_0)$ en $x = x_0$ como el multiplicador de Lagrange de:

$$\begin{aligned} &\min_{x,u} J(x, u) \\ s.t. \quad &x_0 - x = 0. \end{aligned}$$

- En particular:

$$\phi(x) \geqslant \phi(x_0) + \lambda^T (x - x_0).$$

El algoritmo DDP (Dual Dynamic Programming)

Idea:

- Se construye recursivamente una aproximación de cada "cost-to-go" V_k como el máximo de funciones afines (función lineal a tramos).
- En el paso n, tenemos una aproximación $V_k^{(n)}$ de V_k y queremos mejorarla.
- Se sigue una trayectoria óptima $(x_k^{(n)})_{k=1,\dots,N}$ del problema aproximado usando $V_k^{(n)}$ y se agrega un corte para mejorar la aproximación, generando $V_k^{(n+1)}$.
- Ofrece garantías de error en cada paso (cotas del gap).

El algoritmo DDP (Dual Dynamic Programming)

Algoritmo (forward in time):

1. Inicializamos $t = 0$ y $x_0^{(n)} = x_0$.
2. Para cada k resolvemos:

$$\min_{x,u}\{g_k(x,u) + V_k^{(n)}(f_k(x,u))\},$$
$$\text{s.t:} \quad x = x_k^{(n)} \quad [\lambda_k^{(n+1)}]$$

y definimos:

$$\beta_k^{(n+1)} := \text{valor del óptimo}$$
$$\lambda_k^{(n+1)} := \text{multiplicador de la restricción } x = x_k^{(n)}$$
$$u_k^{(n)} := \text{control óptimo aproximado}$$

El algoritmo DDP (Dual Dynamic Programming)

3. Por construcción se tiene que:

$$\beta_k^{(n+1)} = [T_k(V_{k+1}^{(n)})](x_k^{(n)}),$$
$$\lambda_k^{(n+1)} \in \partial[T_k(V_{k+1}^{(n)})](x_k^{(n)}).$$

4. Se tiene entonces que:

$$\beta_k^{(n+1)} + (\lambda_k^{(n+1)})^T(x - x_k^{(n)}) \leqslant [T_k(V_{k+1}^{(n)})](x) \leqslant [T_k(V_{k+1})](x) \leqslant V_k(x)$$

El algoritmo DDP (Dual Dynamic Programming)

5. Por lo tanto, la función afín $x \mapsto \beta_k^{(n+1)} + (\lambda_k^{(n+1)})^T(x - x_k^{(n)})$ es una cota inferior nueva de la función de valor. La agregamos como corte.

$$V_k^{(n+1)} = \max\{V_k^{(n)}(x), \beta_k^{(n+1)} + (\lambda_k^{(n+1)})^T(x - x_k^{(n)})\}.$$

   Se mantiene la convexidad y el ser cota inferior.

6. Elijo $x_{k+1}^{(n)} = f_k(x_k^{(n)}, u_k^{(n)})$ hasta llegar a $k = N$ donde se completa una iteración forward.

El algoritmo DDP (Dual Dynamic Programming)

Inicialización:

- Para inicializar el algoritmo, se requiere $V_k^{(0)}$, cota inferior de la función de valor para cada k. Se pueden computar en una pasada "backwards" a partir $g_N(x)$ usando una trayectoria cualquiera y calculando un corte en cada paso como antes.

Criterio de parada:

- En cada paso se tiene un candidato de solución $u^{(n)}$ y una cota inferior de la función de valor, entonces:

$$\sum_{k=0}^{N-1} g(x_k^{(n)}, u_k^{(n)}) + g_N(x_N^{(n)}) \text{ es una cota superior del costo,}$$

$$V_0^{(n)}(x_0) \text{ es una cota inferior del costo.}$$

- Comparando ambas tengo un criterio de parada con garantías de optimality gap.

El caso estocástico

Supongamos que ahora tenemos perturbaciones w_k en cada paso de tiempo.
Complicaciones:

- Se requiere un modelo probabilítstico de las w_k: e.g. variables iid.
- Para cada iteración n hay que hacer dos pasados:
  - Una hacia adelante (forward) siguiendo una realización de las perturbaciones.
  - Una hacia atrás (backward) que genere nuevos "cortes".
  - No se tendrá una cota superior exacta sino que habrá que estimarla vía Monte Carlo.

Stochastic dynamic programming

Problema

$$\min_{\pi} E\left[\sum_{k=0}^{N-1} g_k(x_k, u_k, w_k) + g_N(x_N)\right],$$

$$\text{sujeto a: } x_{k+1} = f_k(x_k, u_k, w_k)$$
$$u_k = \pi_k(x_k, w_k).$$

con la hipótesis de que w_k son v.a. *iid*.

Observaciones:

- En esta formulación, estamos dejando que u_k dependa de la perturbación actual w_k.
- Puede llevarse a la formulación anterior tomando como estado $y_k = (x_k, w_k)$.
- Cambio importante: esto conmuta el mínimo con la esperanza.

Algoritmo de Bellman

$$
\begin{aligned}
V_N(x) &= g_N(x), \\
\hat{V}_k(x, w) &= \min_{u_k} \{ g_k(x, u_k, w) + V_{k+1}(f_k(x, u_k, w)) \}, \\
V_k(x) &= E\left[\hat{V}_k(x, w_k) \right]
\end{aligned}
$$

Política óptima:

$$
\pi_k(x, w) = \arg\min_{u_k} \{ g_k(x, u_k, w) + V_{k+1}(f_k(x, u_k, w)) \}.
$$

Operador de Bellman

- Para cada k y una función $V : \mathcal{X} \times \mathcal{W} \to \mathbb{R}$ se define:

$$[\hat{T}_k(V)](x, w) := \min_u \{g_k(x, u, w) + V(f(x, u, w))\}.$$

- Con esta notación la iteración de Bellman resulta:

$$\begin{aligned} V_N(x) &= g_N(x), \\ V_k(x) &= T_k(V_{k+1}(x)) := E\left[\hat{T}(V_{k+1})(x, w_k)\right] \end{aligned}$$

- Este operador preserva las mismas propiedades del caso determinístico (monotonía, convexidad, etc.)

Paso de dualidad

- Supongamos que tenemos una cota inferior $V_{k+1}^{(n+1)} \leqslant V_{k+1}$ y un estado $x_k^{(n)}$. Resolvemos:

$$\hat{\beta}_k^{(n+1)}(w) = \min_{x,u} \left\{ g_k(x,u,w) + V_{k+1}^{(n+1)}(f_k(x,u,w)) \right\},$$
$$s.t. \quad x = x_k^{(n)} \quad [\hat{\lambda}_k^{(n+1)}(w)].$$

- Por lo tanto:

$$\hat{\beta}_k^{(n+1)}(w) = \hat{T}_k(V_{k+1}^{(n+1)})(x_k^{(n)}, w),$$
$$\hat{\lambda}_k^{(n+1)}(w) \in \partial_x \hat{T}_k(V_{k+1}^{(n+1)})(x_k^{(n)}, w).$$

- Para cada perturbación entonces:

$$\hat{\beta}_k^{(n+1)}(w) + (\hat{\lambda}_k^{(n+1)})^T(x - x_k^{(n)}) \leqslant \hat{T}_k(V_{k+1}^{(n+1)})(x,w) \leqslant \hat{V}_k(x,w).$$

Paso de dualidad

- Promediando sobre los escenarios, tenemos una cota inferior para la función de valor:

$$\beta_k^{(n+1)} + (\lambda_k^{(n+1)})^T (x - x_k^{(n)}) \leqslant E(\hat{V}_k(x, w)) = V_k(x).$$

- En cada paso definimos:

$$\begin{aligned}
\beta_k^{(n+1)} &= E[\hat{\beta}_k^{(n+1)}(w)] = T_k(V_{k+1}^{(n+1)})(x), \\
\lambda_k^{(n+1)} &= E[\hat{\lambda}_k^{(n+1)}(w)] \in \partial_x T_k(V_{k+1}^{(n+1)})(x).
\end{aligned}$$

Algoritmo SDDP

Al comienzo de cada paso: disponemos de una aproximación $V_k^{(n)}$ de V_k tal que:

- $V_k^{(n)} \leqslant V_k$.
- $V_N^{(n)} = g_N$, el costo terminal.
- $V_k^{(n)}$ es convexa (mejor aún, lineal a tramos...)

Algoritmo SDDP

Forward iteration:

- Seleccionamos un escenario al azar $w_0, \ldots, w_{N-1}$.
- Construimos la trayectoria $x_k^{(n)}$ siguiendo la dinámica:

$$u_k^{(n)} = \arg\min_u \left\{ g_k(x_k^{(n)}, u, w_k) + V_{k+1}^{(n)}(f(x_k^{(n)}, u, w_k) \right\},$$
$$x_{k+1}^{(n)} = f_k(x_k^{(n)}, u_k^{(n)}, w_k).$$

- Esto da la trayectoria óptima si las funciones de cost-to-go se sustituyen por sus aproximaciones $V_k^{(n)}$.

Algoritmo SDDP

Backward iteration:

- En cada k queremos mejorar la estimación de $V_k^{(n)}$.
- Resolvemos, en cada k, para todo posible w:

$$\hat{\beta}_k^{(n+1)}(w) = \min_{x,u}\{g_k(x,u,w) + V_{k+1}^{(n+1)}(f_k(x,u,w))\},$$
$$s.t. \quad x = x_k^{(n)} \quad [\hat{\lambda}_k^{(n+1)}(w)].$$

- Calculamos el subgradiente y óptimo promedio:

$$\beta_k^{(n+1)} = E[\hat{\beta}_k^{(n+1)}(w)], \qquad \lambda_k^{(n+1)} = E[\hat{\lambda}_k^{(n+1)}(w)].$$

- Agregamos el corte a la estimación:

$$V_k^{(n+1)}(x) = \max\{V_k^{(n)}(x), \beta_k^{(n+1)} + (\lambda_k^{(n+1)})^T(x - x_k^{(n)})\}$$

- Retrocedemos de k a $k-1$ y al llegar a 0 se completa la pasada.

- $V_0^{(n)}(x_0)$ sigue siendo una cota inferior exacta del costo.
- El valor observado sobre la trayectoria forward es solo una estimación del costo promedio alcanzable.
- Para tener una cota superior, simulación Monte Carlo:
  - Sorteo varias trayectorias forward.
  - Calculo el costo sobre cada una de ellas y promedio para estimar la esperanza.
  - Si el promedio más dos desvíos esta cerca de la cota inferior, termina la corrida.
  - Se puede hacer solo en algunos n para ahorrar cálculo.

Otros tipos de algoritmos SDDP

- SDDP es en realidad una familia de algoritmos.
- El presentado aquí es DOASA: Dynamic Outer Approximation Sampling Algorithm. Usa solo un escenario para el forward y backward.
- El SDDP clásico ([Pereira, 1991]) elige N escenarios en paralelo, y agrega N cortes en la pasada backward.
- CUPPS (Cutting-Plane and Partial-Sampling) es otra variante.
- Hay otros trucos numéricos que pueden aplicarse. Gran literatura reciente sobre estos temas después de [Pereira, 1991].

Contenido

Julia

- Julia es un lenguaje de programación matemática de desarrollo reciente.
- Código abierto, impulsado por investigadores del MIT.
- Alta optimización en tiempos de cómputo y posibilidades de paralelismo.
- Comunidad internacional de desarrollo muy dedicada.

JuMP ∈ JuliaOpt

- JuliaOpt es el subgrupo encargado de paquetes de optimización.
- Generaron un módulo JuMP (Julia Mathematical Programming) que incorpora los solvers más modernos para resolver todo tipo de problemas de optimización.
- Mantienen otro módulo (Convex) que es el equivalente en Julia de CVX.
- Presentan interfaz con los solvers comerciales más destacados.

mosek

GUROBI OPTIMIZATION

SDDP en Julia

- Dentro de la comunidad JuliaOpt existe un grupo trabajando en la implementación de SDDP.
- La biblioteca y doc se pueden ver en: https://github.com/JuliaOpt/StochDynamicProgramming.jl
- Comunidad muy activa. Uno de los desarrolladores es la fuente de esta presentación.
- Ejemplos de uso incluyen optimización de manejo de represas.

Contenido

Conclusiones

- Presentamos un algoritmo de la clase SDDP para resolver problemas de programación dinámica estocástica de forma aproximada.
- El algoritmo:
  - Explota la convexidad.
  - No requiere discretización del espacio de estados.
  - Construye aproximaciones a la función de valor "en los puntos correctos".
- Tiene garantías de optimalidad y convergencia.
- Se dispone de una implementación moderna con una comunidad de soporte creciente.