Differential Dynamic Programming Neural Optimizer

Sinh viên: Trần Văn Khoa (Mssv: 18020717)
Thầy hướng dẫn: Trần Quốc Long

Trường đại học Công Nghệ, Đại học Quốc Gia Hà Nội

1 Tóm tắt

Tối thiểu hàm mất mát là một nhiệm vụ quan trọng trong các bài toán học máy và học sâu. Các phương pháp tối ưu được nghiên cứu và phát triển và trở thành phần quan trọng trong việc huấn luyện các mô hình học máy. Tuy nhiên các phương pháp tối ưu này vẫn còn những mặt hạn chế, đặc biệt trong những mô hình lớn và phức tạp. Trong bài báo này, việc huấn luyện mạng Deep Neural Networks (DNNs) được xem xét dưới dạng một bài toán điều khiển tối ưu (Optimal Control problem - OCP), qua đó chỉ ra rằng ta có thể liên kết quá trình huấn luyện mạng DNNs với phương pháp tối ưu Differential Dynamic Programming (DDP). Bài báo này đề xuất một phương pháp tối ưu mới, gọi là Differential Dynamic Programming Neural Optimizer (DDPNOpt). Qua thực nghiệm, phương pháp này cho hiệu năng không thua kém gì các phương pháp tốt nhất hiện tại. Ngoài ra, DDPNOpt còn cho thấy kết quả đáng kinh ngạc trong việc ngăn chặn hiện tượng gradient vashing.

2 Giới thiệu

Nghiên cứu này xem xét bài toán điều khiển tối ưu phi tuyến trong miền thời gian rời rạc:

$$\min_{\overline{\boldsymbol{u}}} J(\overline{\boldsymbol{u}}, \boldsymbol{x}_0) = \left[\phi(\boldsymbol{x}_T) + \sum_{t=0}^{T-1} l_t(\boldsymbol{x}_t, \boldsymbol{u}_t)\right] \text{ s.t } \ \boldsymbol{x}_{t+1} = f(\boldsymbol{x}_t, \boldsymbol{u}_t) \tag{1}$$

với $\boldsymbol{x}_t \in \mathbb{R}^n$ và $\boldsymbol{u}_t \in \mathbb{R}^m$ lần lượt là biến trạng thái và biến điều khiển tại thời điểm t. Các hàm f_t, l_t và ϕ lần lượt là hàm phi tuyến chuyển đổi trạng thái, hàm chi phí tức thời và hàm chi phí cuối cùng. OCP hướng tới việc tìm các biến điều khiển tối ưu $\overline{\boldsymbol{u}} \triangleq \{\boldsymbol{u}\}_{t=0}^{T-1}$ nhằm tối thiểu hóa giá trị hàm chi phí J.

Ta có thể coi mạng DNNs được coi như một hệ thống động học phi tuyến thời gian rời rạc, với mỗi lớp được xem như một bước thời gian, với các trọng số tương ứng với các biến điều khiển. Dưới góc nhìn này, các thuật ngữ trong OCP có thể chuyển đổi sang bài toán học sâu được liệt kê trong Bảng 1.

	DNN	OCP
J	Hàm mất mất	Chi phí di chuyển
$\boldsymbol{x}$	Biến đầu vào, đầu ra	Biến trạng thái
$\boldsymbol{u}$	Biến trọng số	Biến điều khiển
f	Lớp	Hàm chuyển đổi trạng thái
ϕ	Hàm mất mát cuối cùng	Hàm chi phí cuối cùng
l	Hàm phân rã trọng số	Hàm chi phí tức thời

Bảng 1: Chuyển đổi thuật ngữ

Đã có nhiều nghiên cứu tối ưu OCP dựa trên Nguyên tắc tối ưu Pontryagin [5] (Pontryagin Maximum Principle - PMP) - đặc trưng cho điều kiện tối ưu bậc nhất. Một phương pháp khác ít được quan tâm hơn là Approximate Dynamic Programming (ADP) [6]. Khác với PMP, ADP hướng tới tối ưu cục bộ ở mỗi bước thời gian. Một

thuật toán phổ biến của ADP là DDP [7] xuất hiện rộng rãi trong các hệ thống tự hành hiện đại để tối ưu hóa quỹ đạo phức tạp. Tuy nhiên việc sử dụng DDP trong huấn luyện DNN vẫn chưa được nghiên cứu sâu rộng.

Bài báo này giới thiệu một phương pháp tối ưu cho DNN mới - DDPNOpt, có sử dụng chính sách phản hồi theo từng lớp, thông qua các thực nghiệm cho thấy cải thiện mạnh mẽ sự hội tụ của hàm mất mát.

3 Nền tảng lý thuyết

3.1 Stochastic gradient descent (SGD)

Các bài toán học máy thường hướng tới tìm bộ trọng số θ nhằm tối ưu hàm mất mát $\mathcal{L}_\theta$ bằng cách khởi tạo ngẫu nhiên θ sau đó cập nhật theo quy tắc gradient descent:

$$\theta \coloneqq \theta - \alpha \mathcal{L}_\theta \tag{2}$$

với $\alpha \in \mathbb{R}^+$ là tốc độ học, $\mathcal{L}_\theta$ là đạo hàm bậc nhất của hàm mất mất $\mathcal{L}$ với biến θ.

Phương pháp trên khá đơn giản và thường đạt hiệu năng không tốt trên nhưng mô hình lớn, đặc biệt là dễ xảy ra gradient vashing và bùng nổ gradient. Để hạn chế các hiện tượng này, phương pháp Newton đề xuất sử dụng ma trận Hessian giúp tốc độ học ổn định hơn:

$$\theta \coloneqq \theta - \alpha \mathcal{L}_{\theta\theta}^{-1} \mathcal{L}_\theta. \tag{3}$$

Phần tính toán ma trận Hessian trong các mô hình học sâu sẽ được đề cập ở phần sau.

3.2 Nguyên tắc Bellman

Định nghĩa hàm giá trị tại trạng thái $\boldsymbol{x}_t$ theo nguyên tắc Bellman [8]:

$$V_t(\boldsymbol{x}_t) = \min_{\boldsymbol{u}_t} \underbrace{l_t(\boldsymbol{x}_t, \boldsymbol{u}_t) + V_{t+1}(f_t(\boldsymbol{x}_t, \boldsymbol{u}_t))}_{Q_t(\boldsymbol{x}_t, \boldsymbol{u}_t) \equiv Q_t}, \quad V_T(\boldsymbol{x}_T) = \phi(\boldsymbol{x}_T). \tag{4}$$

Nguyên tắc Bellman hướng tới việc tối thiểu hàm chi phí trên một chuỗi điều khiển bằng cách tối thiểu cục bộ trên mỗi trạng thái. Hàm V_t đại diện cho chi phí nhỏ nhất để đi từ trạng thái hiện tại tới trạng thái đích. Một chính sách $\pi^* = \{\boldsymbol{u}_t\}_{t=0}^{T-1}$ được gọi là tối ưu nếu nó tối thiểu hàm giá trị ở mỗi thời điểm t.

3.3 Differential Dynamic Programming

Việc giải bài toán (4) một cách trực tiếp gần như là bất khả thi, đặc biệt với các vector nhiều chiều. Để giảm bớt chi phí tính toán, ta có thế xấp xỉ hàm này tại $\boldsymbol{x}_t$ and $\boldsymbol{u}_t$ bằng khai triển Taylor:

$$\begin{aligned} Q_t(\delta\boldsymbol{x}_t, \delta\boldsymbol{u}_t) \equiv\ & l_t(\boldsymbol{x}_t + \delta\boldsymbol{x}_t, \boldsymbol{u}_t + \delta\boldsymbol{u}_t) + V_{t+1}(f_t(\boldsymbol{x}_t + \delta\boldsymbol{x}_t, \boldsymbol{u}_t + \delta\boldsymbol{u}_t)) \\ & - l_t(\boldsymbol{x}_t, \boldsymbol{u}_t) - V_{t+1}(f_t(\boldsymbol{x}_t, \boldsymbol{u}_t)) \\ \approx\ & \frac{1}{2} \begin{bmatrix} 1 \\ \delta\boldsymbol{x}_t \\ \delta\boldsymbol{u}_t \end{bmatrix}^\top \begin{bmatrix} 0 & {Q_{\boldsymbol{x}}^t}^\top & {Q_{\boldsymbol{u}}^t}^\top \\ Q_{\boldsymbol{x}}^t & Q_{\boldsymbol{xx}}^t & Q_{\boldsymbol{xu}}^t \\ Q_{\boldsymbol{u}}^t & Q_{\boldsymbol{ux}}^t & Q_{\boldsymbol{uu}}^t \end{bmatrix} \begin{bmatrix} 1 \\ \delta\boldsymbol{x}_t \\ \delta\boldsymbol{u}_t \end{bmatrix} \end{aligned} \tag{5}$$

$$\text{với} \quad \begin{matrix} Q_{\boldsymbol{x}}^t = l_{\boldsymbol{x}}^t + {f_{\boldsymbol{x}}^t}^\top V_{\boldsymbol{x}}^{t+1} \\ Q_{\boldsymbol{u}}^t = l_{\boldsymbol{u}}^t + {f_{\boldsymbol{u}}^t}^\top V_{\boldsymbol{x}}^{t+1} \end{matrix}, \quad \begin{matrix} Q_{\boldsymbol{xx}}^t = l_{\boldsymbol{xx}}^t + {f_{\boldsymbol{x}}^t}^\top V_{\boldsymbol{xx}}^{t+1} f_{\boldsymbol{x}}^t + V_{\boldsymbol{x}}^{t+1} \cdot f_{\boldsymbol{xx}}^t \\ Q_{\boldsymbol{uu}}^t = l_{\boldsymbol{uu}}^t + {f_{\boldsymbol{u}}^t}^\top V_{\boldsymbol{xx}}^{t+1} f_{\boldsymbol{u}}^t + V_{\boldsymbol{x}}^{t+1} \cdot f_{\boldsymbol{uu}}^t \\ Q_{\boldsymbol{ux}}^t = l_{\boldsymbol{ux}}^t + {f_{\boldsymbol{u}}^t}^\top V_{\boldsymbol{xx}}^{t+1} f_{\boldsymbol{x}}^t + V_{\boldsymbol{x}}^{t+1} \cdot f_{\boldsymbol{ux}}^t \end{matrix}. \tag{6}$$

Phân tích phương trình bậc hai (5) ta có nghiệm:

$$\delta\boldsymbol{u}_t^*(\delta\boldsymbol{x}_t) = \boldsymbol{k}_t + \boldsymbol{K}_t \delta\boldsymbol{x}_t \tag{7}$$

Ký hiệu $\cdot$ trong công thức (6) là phép co rút tensor (tensor contraction).

$$\text{với } \boldsymbol{k}_t = -(Q^t_{\boldsymbol{uu}})^{-1}Q^t_{\boldsymbol{u}},\ \boldsymbol{K}_t = -(Q^t_{\boldsymbol{uu}})^{-1}Q^t_{\boldsymbol{ux}}. \tag{8}$$

Thay ngược kết quả này vào phương trình (5), ta có công thức cập nhật cho các giá trị $V^t_{\boldsymbol{x}}$ và $V^t_{\boldsymbol{xx}}$:

$$\begin{cases} V^t_{\boldsymbol{x}} = Q^t_{\boldsymbol{x}} - {Q^t_{\boldsymbol{ux}}}^\top (Q^t_{\boldsymbol{uu}})^{-1} Q^t_{\boldsymbol{u}} \\ V^t_{\boldsymbol{xx}} = Q^t_{\boldsymbol{xx}} - {Q^t_{\boldsymbol{ux}}}^\top (Q^t_{\boldsymbol{uu}})^{-1} Q^t_{\boldsymbol{ux}} \end{cases}. \tag{9}$$

Quá trình tối ưu DDP được trình bày cụ thể trong Algorithm 1. Bắt đầu với đạo hàm bậc một và bậc hai của hàm chi phí ϕ tại biến trạng thái kết thúc, giải thuật tính toán các thông tin đạo hàm Q^t và cập nhật các ma trận V^t để sử dụng cho lớp trước đó. Trong phần cập nhật, ta cập nhật các biến điều khiển từ lớp đầu tiên, sử dụng các thông tin đạo hàm Q^t và sự thay đổi trạng thái $\delta \boldsymbol{x}_t$.

Algorithm 1 Differential Dynamic Programming

Input: $\overline{\boldsymbol{u}} \triangleq \{\boldsymbol{u}_t\}_{t=0}^{T-1}$, $\overline{\boldsymbol{x}} \triangleq \{\boldsymbol{u}_t\}_{t=0}^{T}$
Khởi tạo $V^T_{\boldsymbol{x}} = \nabla_{\boldsymbol{x}}\phi$ và $V^T_{\boldsymbol{xx}} = \nabla_{\boldsymbol{xx}}\phi$
for $t = T-1$ **to** 0 **do**
  Tính $\delta \boldsymbol{u}^*_t(\delta \boldsymbol{x}_t)$ theo công thức (7), (8)
  Tính $V^t_{\boldsymbol{x}}$ và $V^t_{\boldsymbol{xx}}$ theo công thức (9)
end for
Khởi tạo $\hat{\boldsymbol{x}}_0 = \boldsymbol{x}_0$
for $t = 0$ **to** $T-1$ **do**
  $\boldsymbol{u}^*_t = \boldsymbol{u}_t + \delta \boldsymbol{u}^*_t(\delta \boldsymbol{x}_t)$ với $\delta \boldsymbol{x}_t = \hat{\boldsymbol{x}}_t - \boldsymbol{x}_t$
  $\hat{\boldsymbol{x}}_{t+1} = f_t(\hat{\boldsymbol{x}}_t, \boldsymbol{u}^*_t)$
end for
$\overline{\boldsymbol{u}} \leftarrow \{\boldsymbol{u}^*_t\}_{t=0}^{T-1}$

Algorithm 2 Stochastic gradient descent

Input: $\overline{\boldsymbol{u}} \triangleq \{\boldsymbol{u}_t\}_{t=0}^{T-1}$, $\overline{\boldsymbol{x}} \triangleq \{\boldsymbol{u}_t\}_{t=0}^{T}$, tốc độ học η
Khởi tạo $\boldsymbol{p}_T = \nabla_{\boldsymbol{x}_T} J_T = \nabla_{\boldsymbol{x}}\phi$
for $t = T-1$ **to** 0 **do**
  $\delta \boldsymbol{u}^*_t(\delta \boldsymbol{x}_t) = -\eta \nabla_{\boldsymbol{u}_t} J_t = -\eta(l^t_{\boldsymbol{u}} + {f^t_{\boldsymbol{u}}}^\top \boldsymbol{p}_{t+1})$
  $\boldsymbol{p}_t = \nabla_{\boldsymbol{x}_t} J = {f^t_{\boldsymbol{x}}}^\top \boldsymbol{p}_{t+1}$
end for
for $t = 0$ **to** $T-1$ **do**
  $\boldsymbol{u}^*_t = \boldsymbol{u}_t + \delta \boldsymbol{u}^*_t$
end for
$\overline{\boldsymbol{u}} \leftarrow \{\boldsymbol{u}^*_t\}_{t=0}^{T-1}$

4 DDPNOPT - Huấn luyện mạng DNNs như một bài toán tối ưu quỹ đạo

Cách tính toán trong bước feedforward của một lớp trong DNN có dạng:

$$\boldsymbol{x}_{t+1} = \sigma_t(\boldsymbol{h}_t),\ \boldsymbol{h}_t = g_t(\boldsymbol{x}_t, \boldsymbol{u}_t) = \boldsymbol{W}_t \boldsymbol{x}_t + \boldsymbol{b}_t \tag{10}$$

trong đó các hàm σ_t và g_t lần lượt là hàm kích hoạt phi tuyến và hàm biến đổi tuyến tính với biến trọng số $\boldsymbol{u}_t \triangleq [\text{vec}(\boldsymbol{W}_t), \boldsymbol{b}_t]^\top$. Công thức trên có thể xem như một hệ thống động học như trong OCP với $f_t \equiv \sigma_t \circ g_t$.

Để tìm hiểu mối liên hệ giữa DDP và SGD, trước tiên chúng tôi tóm tắt lại chúng trong Algorithm 1 và Algorithm 2. Ở mỗi vòng huấn luyện, chúng tôi coi trọng số như một biến điều khiển $\overline{\boldsymbol{u}}$ biến đổi trạng thái $\overline{\boldsymbol{x}}$. Từ $(\overline{\boldsymbol{x}}, \overline{\boldsymbol{u}})$, cả hai thuật toán bắt đầu tính toán thông tin cập nhật trọng số từ lớp cuối và lan truyền ngược lại về các lớp trước. Xuất phát từ mối liên hệ trên, bài báo đề xuất một phương pháp tối ưu mới - DDPNOpt được xây dựng tuân theo DDP, nhưng có một vài điều chỉnh sau để thuận lợi cho quá trình huấn luyện DNN.

4.1 Tính toán các ma trận đạo hàm thông qua lan truyền ngược

Để đơn giản hóa hàm mất mất, ta có thể loại bỏ thành phần phân rã trọng số và các thành phần đạo hàm bậc hai trong công thức (6) nhưng vẫn đảm bảo sự ổn định cho nghiệm tối ưu của bài toán. Điều này đã được phân tích kỹ lưỡng trong nghiên cứu của Todorov và các cộng sự [4]. Khi đó, các thành phần Q^t được tính toán trong quá trình lan truyền ngược sẽ là:

$$Q^t_{\boldsymbol{x}} = {g^t_{\boldsymbol{x}}}^\top V^t_{\boldsymbol{h}},\ Q^t_{\boldsymbol{u}} = {g^t_{\boldsymbol{u}}}^\top V^t_{\boldsymbol{h}},\ Q^t_{\boldsymbol{xx}} = {g^t_{\boldsymbol{x}}}^\top V^t_{\boldsymbol{hh}} g^t_{\boldsymbol{x}},\ Q^t_{\boldsymbol{ux}} = {g^t_{\boldsymbol{u}}}^\top V^t_{\boldsymbol{hh}} g^t_{\boldsymbol{x}} \tag{11}$$

với $V^t_{\boldsymbol{h}} \triangleq {\sigma^t_{\boldsymbol{h}}}^\top V^{t+1}_{\boldsymbol{x}}$ và $V^t_{\boldsymbol{hh}} \triangleq {\sigma^t_{\boldsymbol{h}}}^\top V^{t+1}_{\boldsymbol{x}} \sigma^t_{\boldsymbol{h}}$.

4.2 Xấp xỉ ma trận đạo hàm bậc hai

Các thư viện hỗ trợ huấn luyện DNN (Pytorch, Tensorflow ...) hỗ trợ rất tốt việc tính đạo hàm bậc nhất nhưng chưa tính toán được đạo hàm bậc hai. Từ đó, phương pháp EKFAC [2] để xuất tính toàn bộ ma trận bậc hai dựa trên xấp xỉ Gauss-Newton (GN):

$$\nabla^2_{\boldsymbol{u}} J_t \approx \mathbb{E}[J_{\boldsymbol{u}_t} J_{\boldsymbol{u}_t}^\top] = \mathbb{E}[(\boldsymbol{x}_t \otimes J_{\boldsymbol{h}_t})(\boldsymbol{x}_t \otimes J_{\boldsymbol{h}_t})^\top] \approx \mathbb{E}[(\boldsymbol{x}_t \boldsymbol{x}_t^\top)] \otimes \mathbb{E}[(J_{\boldsymbol{h}_t} J_{\boldsymbol{h}_t}^\top)]. \tag{12}$$

Tuy nhiên việc lưu trữ toàn bộ ma trận $Q_{\boldsymbol{uu}}$ trong các mô hình DNN rất tốn bộ nhớ. Vì vậy phương pháp RMSprop [3] đề xuất xấp xỉ dường chéo của ma trận đạo hàm bậc hai dựa trên đạo hàm bậc nhất. Tổng hợp các phương pháp phổ biến nhất được tổng hợp ở Bảng 2.

Phương pháp	Công thức	DDP
SGD	I_t	I_t
RMSprop	$\text{diag}\ (\sqrt{\mathbb{E}[J_{\boldsymbol{u}_t} \odot J_{\boldsymbol{u}_t}]} + \epsilon)$	$\text{diag}\ (\sqrt{\mathbb{E}[Q^t_{\boldsymbol{u}} \odot Q^t_{\boldsymbol{u}}]} + \epsilon)$
KFAC/EKFAC	$\mathbb{E}[\boldsymbol{x}_t \boldsymbol{x}_t^\top] \otimes \mathbb{E}[J_{\boldsymbol{h}_t} J_{\boldsymbol{h}_t}^\top]$	$\mathbb{E}[\boldsymbol{x}_t \boldsymbol{x}_t^\top] \otimes \mathbb{E}[V^t_{\boldsymbol{h}} {V^t_{\boldsymbol{h}}}^\top]$

Bảng 2: Các phương pháp xấp xỉ ma trận Hessian

Việc phải tính toán các thành phần $Q_{\boldsymbol{xx}}$, $Q_{\boldsymbol{ux}}$ và $V_{\boldsymbol{xx}}$ khiến cho quá trình huấn luyện bị chậm đáng kể. Do đó ta có thể xấp xỉ các ma trận này bằng phương pháp GN:

$$Q^t_{\boldsymbol{xx}} = Q^t_{\boldsymbol{x}} \otimes Q^t_{\boldsymbol{x}}, \quad Q^t_{\boldsymbol{ux}} = Q^t_{\boldsymbol{u}} \otimes Q^t_{\boldsymbol{x}}, \quad V^t_{\boldsymbol{xx}} = z^t_{\boldsymbol{x}} \otimes z^t_{\boldsymbol{x}} \tag{13}$$

$$\text{với} \quad z^t_{\boldsymbol{x}} = \sqrt{1 - {Q^t_u}^\top (Q^t_{\boldsymbol{uu}})^{-1} Q^t_{\boldsymbol{u}}} Q^t_{\boldsymbol{x}}. \tag{14}$$

Cuối cùng, phương pháp đề xuất DDPNOpt được trình bày chi tiết trong Algorithm 3.

Algorithm 3 Differential Dynamic Programming Neural Optimizer

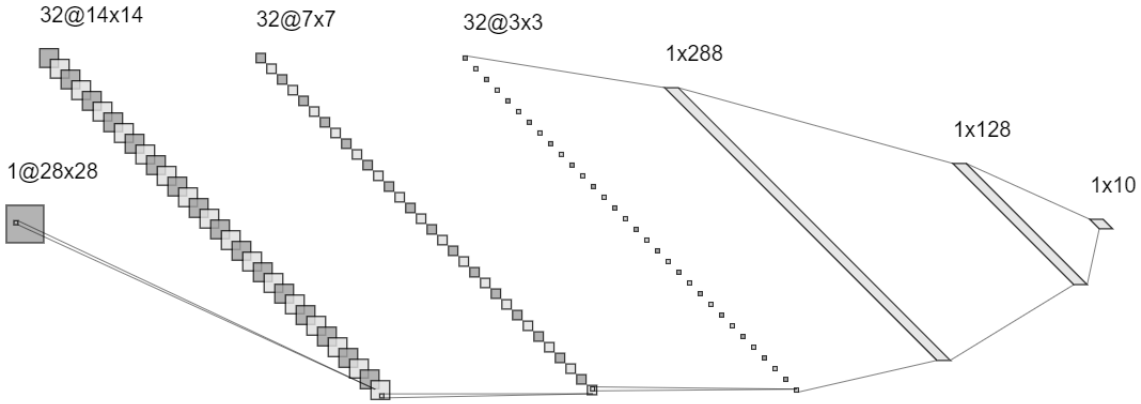
Input: bộ dữ liệu $\mathcal{D}$, tốc độ học η, số vòng lặp huấn luyện K, batch size B
Khởi tạo ngẫu nhiên trọng số $\overline{\boldsymbol{u}}^{(0)}$ của mô hình
for $k = 1$ **to** K **do**
  Lấy mẫu dữ liệu huấn luyện $\boldsymbol{X}_0 \equiv \{\boldsymbol{x}_0^{(i)}\}_{i=1}^B \sim \mathcal{D}$
  Thực hiện feedforward để tạo ra $\boldsymbol{X}_t$
  Khởi tạo $V^T_{\boldsymbol{x}^{(i)}} = \nabla_{\boldsymbol{x}^{(i)}} \phi(\boldsymbol{x}_T^{(i)})$ và $V^T_{\boldsymbol{x}\boldsymbol{x}^{(i)}} = \nabla^2_{\boldsymbol{x}^{(i)}} \phi(\boldsymbol{x}_T^{(i)})$
  for $t = T - 1$ **to** 0 **do**
    Tính $Q^t_{\boldsymbol{u}}$, $Q^t_{\boldsymbol{x}}$, $Q^t_{\boldsymbol{xx}}$, $Q^t_{\boldsymbol{ux}}$ theo công thức (11) hoặc (13)
    Tính $\mathbb{E}[Q^t_{\boldsymbol{uu}}]$ theo một trong các đề xuất trong Bảng 2
    Tính và lưu lại các giá trị $\{\boldsymbol{k}_t^{(i)}, \boldsymbol{K}_t^{(i)}\}_{i=1}^B$
    Tính $V^t_{\boldsymbol{x}^{(i)}}$ và $V^t_{\boldsymbol{x}\boldsymbol{x}^{(i)}}$ theo công thức (9) hoặc (13), (14)
  end for
  Đặt $\hat{\boldsymbol{x}}_0^{(i)} = \boldsymbol{x}_0^{(i)}$
  for $t = 0$ **to** $T - 1$ **do**
    $\boldsymbol{u}_t^* = \boldsymbol{u}_t + \delta \boldsymbol{u}_t^*(\delta \boldsymbol{X}_t)$, với $\delta \boldsymbol{X}_t = \{\hat{\boldsymbol{x}}_t^{(i)} - \boldsymbol{x}_t^{(i)}\}_{i=1}^B$
    $\hat{\boldsymbol{x}}_{t+1}^{(i)} = f_t(\hat{\boldsymbol{x}}_t^{(i)}, \boldsymbol{u}_t^*)$
  end for
  $\overline{\boldsymbol{u}}^{(k+1)} \leftarrow \{\boldsymbol{u}_t^*\}_{t=0}^{T-1}$
end for

Ký hiệu $\otimes$ trong các công thức (12) và (13) là phép nhân Kronecker.

5 Thảo luận và thí nghiệm

5.1 Phân loại ảnh

Dề đánh giá hiệu năng của DDPNOpt, trước tiên chúng tôi cài đặt một thí nghiệm đơn giản cho bài toán phân loại ảnh. Trong bài toán này, chúng tôi sử dụng bộ data MNIST. Đây là tập dữ liệu chữ viết tay từ 0 đến 9. Trong đó mỗi ảnh là ảnh xám có kích thước 28×28. Chúng tôi sử dụng 10000 ảnh để huấn luyện và đánh giá mô hình trên 10000 ảnh khác.



Hình 1: Mô hình cho bài toán phân loại ảnh

Mô hình được sử dụng được mô tả như trong Hình 1. Đầu tiên chúng tôi chuẩn hoá dữ liệu đầu vào bằng cách chuẩn hóa giá trị các pixel về khoảng -1 đến 1. Chúng tôi xây dựng 3 khối convolution với các bộ lọc kích thước 3x3 và sử dụng padding bằng 1. Giữa các khối convolution chúng tôi dùng hàm kích hoạt Tanh cùng với một lớp MaxPolling. Để phân loại ra được ảnh thuộc 1 trong 10 lớp ứng với các số từ 0 đến 9, chúng tôi trải phẳng đầu ra của các lớp convolution, kế tiếp dùng các lớp fully connected với đầu ra là vector 10×1 được cho qua hàm softmax. Hàm chi phí được sử dùng là cross entropy. Chúng tôi sử dụng các hàm tối ưu phổ biến như RMSprop và Adam để so sánh với DDPNOpt.

	RMSprop	Adam	DDPNOpt
F1-score	0.9849	0.984768	**0.986**

Bảng 3: F1-score

Sau khi thử nghiệm, chúng tôi tính được kết quả f1-score như Bảng 3. Khi dùng DDPNOpt thì đạt kết quả tốt nhất với f1-score là 0.986, trong khi hai phương pháp RMSprop và Adam thì thấp hơn một chút với f1-score lần lượt là 0.9849 và 0.984768.

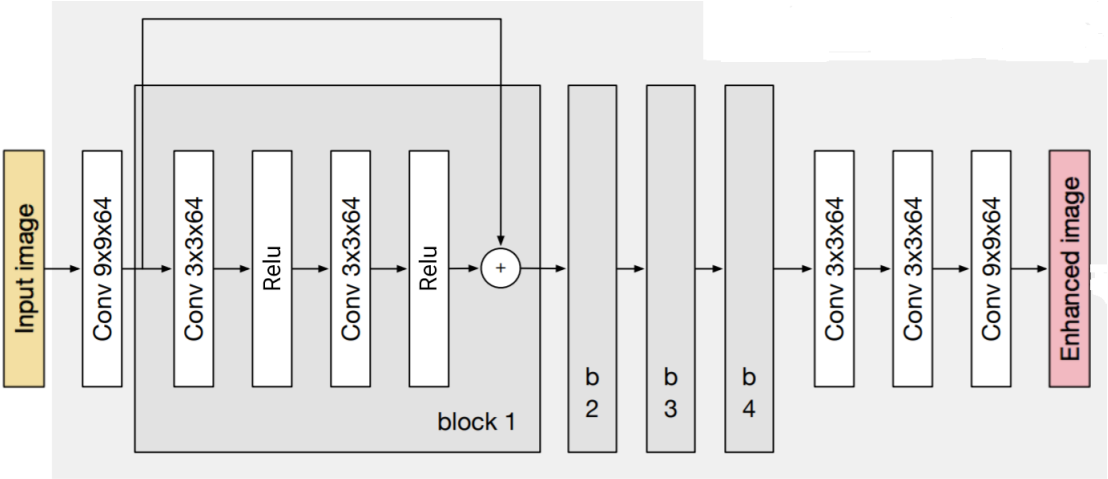
5.2 Bài toán xóa nhiễu cho ảnh

Mục tiêu của bài toán này là xây dựng mô hình có thể xóa nhiễu cho ảnh. Chúng tôi tạo dữ liệu bằng cách thêm nhiễm vào các ảnh không có nhiễu và coi ảnh nhiễu nà như một đầu vào cho model và ảnh ban đầu như một label, như ví dụ mẫu ở Hình 2. Ở đây thì chúng tôi sử dụng 2975 ảnh huấn luyện và 500 ảnh đánh giá, mỗi ảnh có kích thước là $256 \times 256 \times 3$.
Chúng tôi sử dụng mô được được đề xuất trong bài báo của Ignatov [10] như trong Hình 3. Đầu tiên, chúng tôi chuẩn hóa dữ liệu đầu vào bằng cách chuẩn hóa giá trị pixel về khoảng -1 đến 1. Mô hình ở đây sử dụng các khối convolution, hàm kích hoạt ReLU ở các lớp giữa và hàm kích hoạt Tanh ở cuối cùng. Hàm chi phí được sử dụng là mean square error.

(a) Đầu vào

(b) Đầu ra

Hình 2: Ví dụ của một điểm dữ liệu



Hình 3: Mô hình được sử dụng

	RMSprop	Adam	DDPNOpt
PSNR	32.245	32.079	**32.45**

Bảng 4: Peak signal to noise ratio

Để đánh giá kết quả, ở đây chúng tôi sử dụng độ đo PSNR. Với độ đo này thì ảnh có giá trị gần giống với ảnh không nhiễu thì kết quả càng lớn. Kết quả được trình bày trong Bảng 4. Với phương pháp được đề xuất, điểm PSNR của mô hình có chút nhỉnh hơn.

(a) Ảnh đầu vào

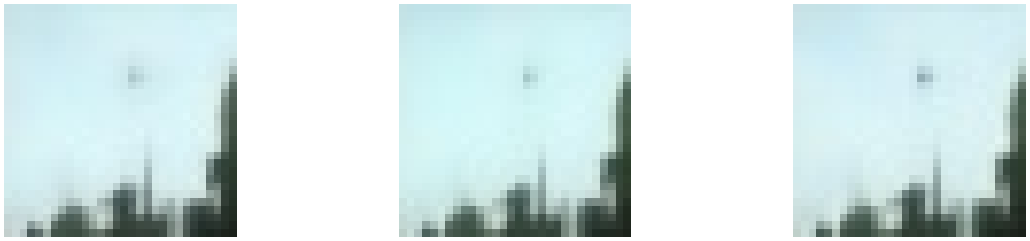
(b) Adam

(c) RMSprop

(d) DDPNOPT

Hình 4: Ảnh đầu vào và các kết quả khi sử dụng các phương pháp

Hình 4 là một vài ví dụ sau huấn luyện với các phương pháp khác nhau. Chúng tôi thấy thì ảnh đã loại bỏ được nhiễm nhưng chất lượng ảnh bị giảm đi đôi chút. Nhìn qua thì thấy không có sự khác biệt giữa các phương pháp, nhưng nhìn kĩ thì thấy phần chấm nhỏ trong ảnh sử dụng mô hình được huấn luyện với DDPNOpt rõ hơn so với mô hình được huấn luyện với hai phương pháp còn lại. Chi tiết được trình bày trong Hình 5.



(a) Adam (b) RMSprop (c) DDPNOPT

Hình 5: Cận cảnh phần khác biệt giữa các phương pháp ở hình 4

6 Kết luận

Trong bài báo này, nghiên cứu đề xuất DDPNOpt, một phương pháp tối ưu hóa mới liên hệ giữa quy trình huấn luyện DNN với điều khiển tối ưu và tối ưu hóa quỹ đạo. DDPNOpt có các chính sách phản hồi theo lớp giúp cải thiện sự hội tụ và độ mạnh mẽ của hàm mất mát so với các phương pháp tối ưu hóa hiện có. Nghiên cứu này cung cấp một cái nhìn sâu sắc về thuật toán mới và là cầu nối giữa học sâu và điều khiển tối ưu.
Bằng các thực nghiệm, chúng tôi thấy DDPNOpt tốt hơn chút so với các phương pháp phổ biến hiện tại. Tuy nhiên việc phải tính toán các thành phần phản hồi khiến cho quá trình huấn luyện lâu hơn khoảng hai lần và gây tốn bộ nhớ hơn.

Tài liệu

[1] Liu, Guan-Horng et al. “DDPNOpt: Differential Dynamic Programming Neural Optimizer.” ICLR (2021).

[2] George, Thomas et al. “Fast Approximate Natural Gradient Descent in a Kronecker-factored Eigenbasis.” NeurIPS (2018).

[3] T. Tieleman and G. Hinton. Lecture 6.5 - rmsprop: Divide the gradient by a running average of its recent magnitude, 2012.

[4] Emanuel Todorov and Weiwei Li. A generalized iterative lqg method for locally-optimal feedback control of constrained nonlinear stochastic systems. In Proceedings of the 2005, American Control Conference, 2005., pp. 300–306. IEEE, 2005.

[5] Vladimir Grigor’evich Boltyanskii, Revaz Valer’yanovich Gamkrelidze, and Lev Semenovich Pontryagin. The theory of optimal processes. i. the maximum principle. Technical report, TRW SPACE TECHNOLOGY LABS LOS ANGELES CALIF, 1960

[6] Dimitri P Bertsekas, Dimitri P Bertsekas, Dimitri P Bertsekas, and Dimitri P Bertsekas. Dynamic programming and optimal control. Athena scientific Belmont, MA, 1995.

[7] D.H. Jacobson and D.Q. Mayne. Differential Dynamic Programming. Modern analytic and computational methods in science and mathematics. American Elsevier Publishing Company, 1970. URL https://books.google.com/books?id=tA-oAAAAIAAJ

[8] Richard Bellman. The theory of dynamic programming. Technical report, Rand corp santa monica ca, 1954

[9] Kingma, Diederik P. and Jimmy Ba. “Adam: A Method for Stochastic Optimization.” CoRR abs/1412.6980 (2015): n. pag.

[10] Ignatov, A., Kobyshev, N., Timofte, R., Vanhoey, K. and Van Gool, L., 2017. Dslr-quality photos on mobile devices with deep convolutional networks. In Proceedings of the IEEE International Conference on Computer Vision (pp. 3277-3285).