Wydział Elektroniki i Technik Informacyjnych
Politechnika Warszawska

Zaawansowane zagadnienia sieci neuronowych

Rezydualne uczenie ciągłe klasyfikatorów obrazowych - Dokumentacja końcowa

Kaczmarek Kacper,
Rębacz Gabriel

Warszawa, 12 czerwca 2021

1 Treść zadania

Celem projektu jest implementacja metody uczenia ciągłego określanej jako *residual continual learning* [1] i weryfikacja jej działania w kontekście klasyfikacji obrazów (np. na CIFAR-10/CIFAR-100). W ramach podsumowania proszę porównać efektywność tego podejścia z wybraną metodą typu regularyzacyjnego i metodą typu replay.

1.1 Opis zadania

W artykule [1] przedstawiony został algorytm rezydualnego uczenia ciągłego w aspekcie klasyfikatorów obrazowych. Uczenie ciągłe polega na możliwości sekwencyjnego uczenia modeli tak aby były one w stanie wykonywać kolejne zadania, lecz nie traciły umiejętności wykonywania poprzednich zadań. Celem projektu jest przeprowadzenie procesu uczenia zgodnego z opisanym w artykule, weryfikacja wyników oraz porównanie ich z wynikami benchmarków opisanych w [2] w ramach *Incremental task learning*.

2 Narzędzia

W trakcie implementacji rozwiązania wykorzystaliśmy poniższe narzędzia:

- język: Python ver. 3.8.7,
- PyTorch,
- srodowisko wirtualne: Conda.

3 Opis rozwiązania

3.1 Opis algorytmu

Algorithm 1: Uczenie modelu

Input: $net_s(\cdot;\, \theta_s^*)$ `// source network`
Input: λ `// trade-off parameter`
Input: $(X_t,\, Y_t)$ `// training dataset of target task`
$net_t(\cdot;\, \theta_t) \leftarrow net_s(\cdot;\, \theta_s^*)$
$\theta_t^* \leftarrow argmin_{\theta_t} D_{KL}(Y_t||net_t(X_t;\theta_t)) + \frac{1}{2}\lambda_{\text{dec}}\|\theta_t\|_2^2$
$\hat{Y_s} \leftarrow net_s(X_t;\, \theta_s^*)$
$\hat{Y_t} \leftarrow net_t(X_t;\, \theta_t^*)$
$(\alpha_s, \alpha_t) \leftarrow (-\frac{1}{2}\cdot\mathbb{1}, \frac{1}{2}\cdot\mathbb{1})$
$\theta_t \leftarrow \theta_t^*$
$net_c(\cdot;(\alpha_s, \theta_s^*, \alpha_t, \theta_t), task = \cdot) \leftarrow$
$COMBINE(\alpha_s, net_s(\cdot;\theta_s^*), \alpha_t, net_t(\cdot;\theta_t))$
$(\alpha_s^*, \alpha_t^*, \theta_t^{**}) \leftarrow argmin_{\alpha_s,\alpha_t,\theta_t}\{D_{KL}(\hat{Y_s}||net_c(X_t;(\alpha_s,\theta_s^*), task = s)) + D_{KL}(\hat{Y_t}||net_c(X_t;(\alpha_s,\theta_s^*), task = t)) + \lambda\|(\alpha_s,\alpha_t)]\|_1 + \frac{1}{2}\lambda_{\text{dec}}\|\theta_t\|_2^2\}$
Result: $net_c(\cdot;(\alpha_s^*, \theta_s^*, \alpha_t^*, \theta_t^{**}), task = \cdot)$

3.2 Ważne założenia artykułu

- Nie potrzebne są dane związane z zadaniem źródłowym oprócz samej sieci realizującej zadanie
- Rozmiar sieci nie powiększa się (oprócz dodatkowych ostatnich warstw perceptronów wielowarstwowych pełniących rolę klasyfikatora
- Metoda może być użyta z sieciami CNN. Również z takimi, które wykorzystują normalizację batchy.

3.3 Ważne założenia i uwagi autorów

- W artykule sieci uczone są do rozpoznawania dwóch niezależnych zbiorów danych. My w celu porównania z wybranym benchmarkiem zdecydowaliśmy się podzielić zbiór CIFAR100 na 5 zbiorów każdy zawierający 20 klas, aby uczyć go inkrementalnie.
- Zrezygnowaliśmy z fuzji wartwy konwolucyjnej z batchnormem pokazanej na rysunku 2.

- Każdy z taksów posiada swoją własną sieć klasyfikującą na końcu sieci ResNet oraz jest ona przełączana w zależności od zadania jakie sieć rozwiązuje.
- Dodatkowo po każdym procesie uczenia z opisanym w algorytmie 1 dostrajana jest warstwa końcowa, w celu uzyskania lepszych wyników.
- Rozszerzyliśmy kombinowanie warstw także na wstępną warstwę konwolucyjną, shortcuty oraz ostatnią warstwę normalizującą batch, które nie są uwzględnione w artykule, ale występują w naszej implementacji sieci ResNet.
- W przypadku dostrajania sieci target, funkcja straty została zamieniona z dywergencji Kullbacka-Leiblera na Cross-entropy loss w celu ułatwienia implementacji.
- Wagi sieci source i target muszą pozostać zamrożone w momencie uczenia sieci combined w celu poprawnego działania algorytmu.
- Wagi ostatnich warstw sieci combined nie są mrożone na żadnym etapie algorytmu, w trakcie eksperymentów okazało się że jest to bardzo istotne aby sieć zachowywała zdolność wykonywania poprzednich zadań.

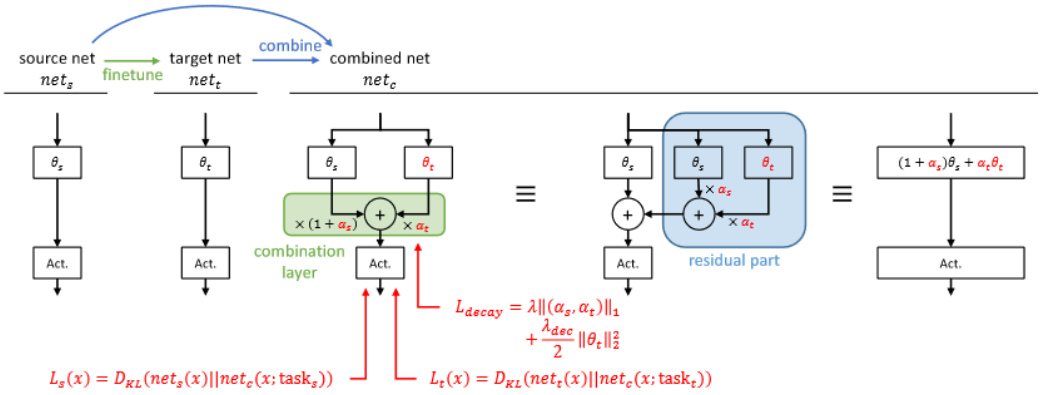
3.4 Typ modelu

Model zostanie wyuczony zgodnie z instrukcjami opisanymi w [1]. Główny algorytm tworzenia powyższego modelu opisuje Algorytm 1.

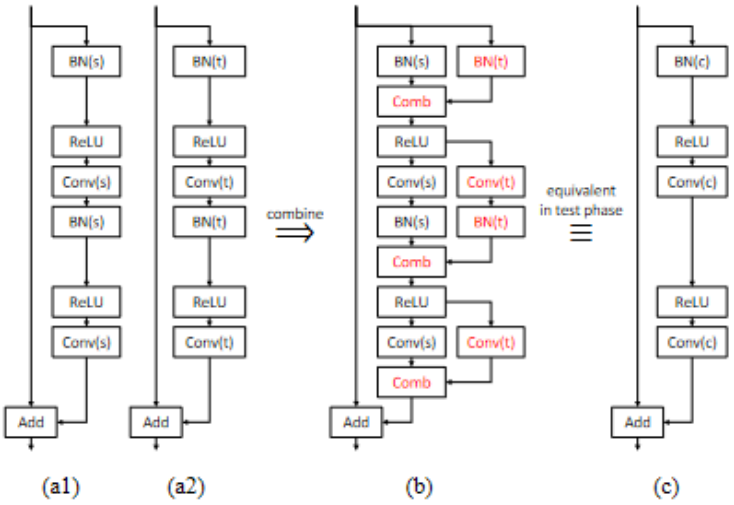
W pierwszym kroku tworzona/pobrana jest sieć źródłowa na zbiorze źródłowym o architekturze zgodnej z [3], której kod implementacji znajduje się w [4]. Nasz wybór padł na *WideResNet_28_2_cifar*, która posiada warstwę konwolucyjną 16x3, potem trzy "stage", w każdym po cztery bloki, a dla każdego stage filtry mają rozmiary kolejno 32x32, 64x64, 128x128 na końcu warstwę batchnorm oraz klasyfikator zrealizowany za pomocą perceptronu wielowarstwowego. Wybór ten został podyktowany użyciem wyżej opisanych benchmarków, które korzystają z właśnie tej implementacji, zatem porównanie metod będzie wiarygodne. Następnie kopiowana jest sieć źródłowa (source network) jako sieć docelowa (target network) i zostaje ona trenowana pod kątem nowego zadania. Tworzona jest końcowa sieć powstała z połączenia sieci źródłowej oraz docelowej zgodnie z wytycznymi opisanymi na Rysunku 1 i Rysunku 2. W sieci docelowej uczone są tylko parametry dobory cech α_s i α_t oraz wagi skopiowanej sieci docelowej θ_t. Do wyznaczania gradientów wykorzystywana

jest funkcja kosztu stanowiąca sumę trzech funkcji kosztów opisanych na Rysunku 1. Warto zauważyć, że każde z zadań powinno mieć dopasowaną własną warstwę neuronów końcowych. Warstwa ta jest przełączana w zależności od rozwiązywanego zadania.

Rysunek 1: Schemat sposobu tworzenia połączenia



Rysunek 2: Schemat miejsc połączeń warstw



4 Badania

4.1 Założenia

Postanowiliśmy wykorzystać zbiór danych CIFAR-100, który posiada obrazki 32x32 przedstawiające 100 różnych klas. Na potrzeby zadania podzielimy go na 5 rożnych zbiorów, z których każdy posiada 20 klas. Architektura sieci będzie dokładnie taka sama jak w benchmarku[2] co pozwoli w prosty sposób zaimplementować na nich algorytm *residual continual learning*, a implementacja będzie rozszerzeniem kodu implementującego benchmark w celu jak najwierniejszego oddania zasad panujących przy przebiegach, z którymi będą porównywane nasze wyniki.

Sprawdzona zostanie jakość uzyskanych wyników wykorzystując najpierw sieć wytrenowaną na podzbiorze CIFAR-100 zawierającym tylko 20 klas jako sieć źródłową i następnie sekwencyjnie model będzie uczony ciągle z wykorzystaniem następnych podzbiorów CIFAR-100, które zawierają kolejne 20 podklas. Struktura sieci będzie rozszerzana o kolejne warstwy liniowe klasyfikujące dla danego zadania, jednak rozmiar sieci przed klasyfikatorem pozostanie ten sam. W ten sposób sieć nauczy się rozpoznawać wszystkie 100 klas inkrementalne. Kolejne sieci wyjściowe będą stawały się siecią źródłową, a sieć docelowa będzie uczona kolejnego zadania, aby ją połączyć w sieć źródłową w następnej iteracji.

4.2 Wyniki

	Algorithm	Incremental Task Learning score
Baselines	Adam	30.53 ± 0.58
	SGD	43.77 ± 1.15
	Adagrad	36.27 ± 0.43
	L2	51.73 ± 1.30
	Naive rehearsal	70.20 ± 0.17
	Naive rehearsal-C	78.41 ± 0.37
Continual learning methods	EWC	61.11 ± 1.43
	Online EWC	63.22 ± 0.97
	SI	64.81 ± 1.00
	MAS	64.77 ± 0.78
	ResCL	**77.83** ± 1.79
	Offline (Adam)	84.28 ± 0.42
	Offline (SGD)	86.79 ± 0.17

Tablica 1: Wyniki accuracy dla poszczególnych algorytmów wraz z naszym dla porównania

5 Wnioski

- Metoda zaprezentowana w artykule[1] prezentuje bardzo dobre wyniki w porównaniu do innych metod *Continual learning methods* z tabeli 1 opisanych w benchmarku[2].
- Algorytm można parametryzować zmieniając parametr lambda (λ). Wartość przy jakiej zostaliśmy to 0.0001. Została ona wybrana eksperymentalnie i może nie być optymalna. Możliwe, że zoptymalizowanie tego parametru pod zadanie mogłoby dać jeszcze lepsze wyniki uczenia.
- W artykule nie wszystko odpowiadało naszej budowie sieci i część założeń musiała być poczyniona przez nas. Naszym dodatkowym pomysłem jest dostrajanie ostatniej warstwy co mogło przyczynić się do lepszych wyników.
- Możliwe, że model jest przystosowany do *incremental class learning*, jednak ostatecznie porzuciliśmy ten temat na rzecz *incremental task learning*.

Bibliografia

[1] Janghyeon Lee i in. *Residual Continual Learning*. 2020. arXiv: 2002.06774 [cs.LG].

[2] Yen-Chang Hsu i in. „Re-evaluating Continual Learning Scenarios: A Categorization and Case for Strong Baselines". W: *NeurIPS Continual learning Workshop*. 2018. URL: https://arxiv.org/abs/1810.12488.

[3] Kaiming He i in. *Identity Mappings in Deep Residual Networks*. 2016. arXiv: 1603.05027 [cs.CV].

[4] Kaiming He i in. *Deep Residual Networks with 1K Layers*. [Strona internetowa, repozytorium; data odwiedzin: 26.03.2021]. URL: https://github.com/KaimingHe/resnet-1k-layers.