Metoda ważenia gradientu w zejściu gradientowym przez sztuczną sieć neuronową

Jan Miksa
promotor: dr. hab. Przemysław Spurek

Wydział Matematyki i Informatyki
Uniwersytet Jagielloński

Kraków
24 września 2023

Streszczenie

Poniżej, po krótkim przypomnieniu niezbędnych informacji z dziedziny uczenia maszynowego i przedstawieniu powiązanych badań nad algorytmami *uczenia optymalizacji* oraz metodami *hipergradientu*, rozważamy modyfikację algorytmu zejścia gradientowego w uczeniu sztucznych sieci neuronowych.

Zamiast schodzić bezpośrednio w kierunku przeciwnym do gradientu wprowadzone jest dynamiczne *ważenie*, które go modyfikuje. Samo *ważenie* również dostosowuje się w trakcie treningu. Jest to tak naprawdę alternatywne sformułowanie szczególnych przypadków isntniejących już metod.

W kontekście tej modyfikacji wprowadzamy i badamy nową metodę modelowania *ważenia* w oparciu o dodatkową sieć neuronową - **GWNMO**. Zestawiamy ją z już istniejącymi metodami uczenia i przeprowadzamy niezbędne eksperymenty.

Spis treści

1 Preliminaria **2**
- 1.1 Krótkie wprowadzenie do sztucznych sieci neuronowych . . . 2
- 1.2 Optymalizacja . . . 3
  - 1.2.1 Zejście gradientowe . . . 3
- 1.3 Proces uczenia nadzorowanego . . . 6
  - 1.3.1 Stochastyczne zejście gradientowe . . . 7
  - 1.3.2 Adam . . . 7
- 1.4 Sieci konwolucyjne . . . 8
  - 1.4.1 Filtr konwolucyjny . . . 8
  - 1.4.2 Budowa sieci . . . 8
- 1.5 Sieci rekurencyjne . . . 9

2 Uczenie optymalizacji **9**
- 2.1 Wstęp . . . 9
- 2.2 Formalizm w przypadku zejścia gradientowego . . . 10
- 2.3 Realizacja . . . 10

3 Zaawansowane modyfikacje zejścia gradientowego **10**
- 3.1 Metody hipergradientu . . . 10
- 3.2 Metody ważenia gradientu . . . 11

4 GWNMO **11**
- 4.1 Wstęp . . . 11
- 4.2 Sformułowanie . . . 12
- 4.3 Działanie . . . 12
  - 4.3.1 Algorytm uczenia . . . 12
- 4.4 Analiza . . . 13
  - 4.4.1 Złożoność czasowa . . . 13
  - 4.4.2 Lokalne malenie . . . 13
- 4.5 Eksperymenty . . . 14
  - 4.5.1 Architektura . . . 14
  - 4.5.2 Wyniki . . . 14
  - 4.5.3 Implementacja . . . 16
- 4.6 Dalsze prace . . . 16

5 Podsumowanie **16**

1 Preliminaria

1.1 Krótkie wprowadzenie do sztucznych sieci neuronowych

Sztuczne sieci neuronowe to bardzo ciekawe i potężne modele matematyczne o szerokiej gamie zastosowań. Oryginalnie były inspirowane biologicznymi odpowiednikami, choć później *analogie* zostały porzucone.

Intuicyjnie możemy rozumieć je jako modele, które w procesie uczenia tworzą wewnętrzną reprezentację danych i stosują na niej model liniowy.

Definicja 1.1. Sztuczna sieć neuronowa to model matematyczny postaci: $\phi = f_k \circ \ldots \circ f_1$, gdzie:

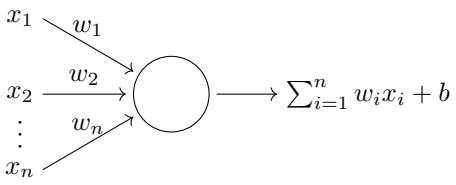
$$f_i(x) = \begin{cases} g(W_i^T x + b_i) & \text{jeśli } i \leq k \\ W_k^T x + b_k & \text{jeśli } i = k \end{cases}$$

Kolejne f_i nazywamy i-tymi *warstwami* sieci. Warstwa pierwsza i ostatnia są nazywane odpowiednio *wejściową* i *wyjściową*, pozostałe są określane mianem *warstw ukrytych*. W_i to macież wag danej warstwy, b_i to wektor wyrazów wolnych, a $g : \mathbb{R} \to \mathbb{R}$ to nieliniowa funkcja aktywacji.

Definicja 1.2. Wynik pojedynczej warstwy $f(x) = W^T x + b$ to wektor:

$$f_j(x) = \sum_k W_k j x_k + b_j$$

Parę $(W_{\cdot,j}, b_j)$ interpretowaną jako pojedyncza jednostka nazywamy **sztucznym neuronem**. Ilość kolumn W nazywamy *szerokością* warstwy.



Rysunek 1: Schemat sztucznego neuronu

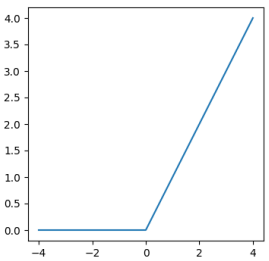
W praktyce jako nieliniowej funkcji aktywacji zwykle używa się funkcji ReLU [*ang. rectified linear unit*] zdefiniowanej poniżej.

Definicja 1.3. Funkcja aktywacji **ReLU**:

$$ReLU(x) = max(0, x)$$

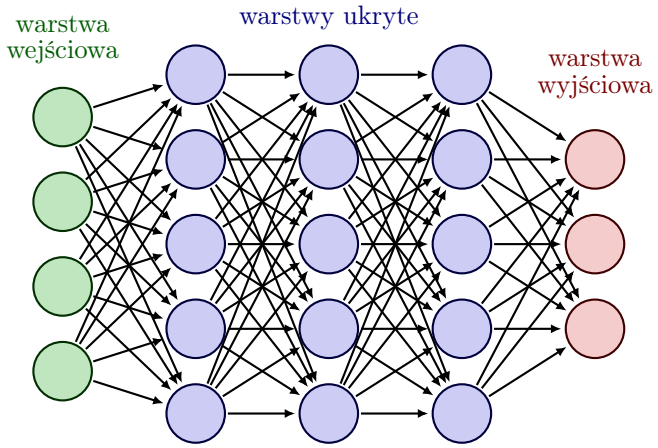
Jej pochodną określamy jako:

$$\frac{d}{dx} ReLU(x) = \begin{cases} 0 & \text{jeśli } x \leq 0 \\ 1 & \text{w przeciwnym wypadku} \end{cases}$$



Rysunek 2: ReLU

Mianem *architektury* sztucznej sieci neuronowej określamy ilość warstw oraz ich szerokości.



Rysunek 3: Schemat architektury sztucznej sieci neuronowej. Źródło: [Neu], TikZ.net

1.2 Optymalizacja

Zanim przejdziemy do procesu *uczenia* opisanych wyżej struktur, najpierw musimy spojrzeć ogólniej. Poniżej wprowadzamy zejście gradientowe wspierając się publikacją [GG23].

Definicja 1.4. Mówimy, że $f : \mathbb{R}^N \to \mathbb{R} \cup +\infty$ jest **wypukła** jeśli dla każdych $x, y \in \mathbb{R}^N$ i $t \in [0, 1]$ zachodzi:

$$f(tx + (1-t)y) \leq tf(x) + (1-t)f(y)$$

Definicja 1.5. Niech $f : \mathbb{R}^N \to \mathbb{R}$ i $L \in \mathbb{R}^+$. Mówimy, że f jest L-gładka jeśli jest różniczkowalna i $\nabla f : \mathbb{R}^N \to \mathbb{R}^N$ jest L-Lipschitzowski:

$$\forall_{x,y \in \mathbb{N}} \|\nabla f(x) - \nabla f(y)\| \leq L\|x - y\|$$

Chcemy minimalizować funkcję różniczkowalną $f(x) : \mathbb{R}^N \to \mathbb{R}$. Aby problem był dobrze postawiony wymagamy żeby $\arg\min f \neq \emptyset$. Dla uproszenia poniżej zakładamy dodatkowo, że f jest wypukła oraz L-gładka.

1.2.1 Zejście gradientowe

Niech $\alpha > 0$ będzie *wielkością kroku*. Zejście gradientowe przy danym x_0 generuje ciąg $(x_t)_{t \in \mathbb{N}}$ spełniający:

$$x_{t+1} = x_t - \alpha \nabla f(x_t)$$

Twierdzenie 1.1. (Lokalne malenie funkcji w zejściu gradientowym)
Przy powyższej definicji x_t. Nawet przy osłabionych założeniach: $f : \mathbb{R}^N \to \mathbb{R}$, $f \in \mathcal{C}^1$ oraz $\forall_{x \in \mathbb{R}^N} \nabla f(x) \neq 0$

$$\exists_{\alpha > 0} f(x_{t+1}) < f(x_t)$$

Dowód.

$$\Delta f(x_t) = f(x_{t+1}) - f(x_t) = f(x_t - \alpha \nabla f(x_t)) - f(x_t)$$

$$D_{-\nabla f(x_t)} f(x_t) = \lim_{\alpha \to 0} \frac{\Delta_\alpha f(x_t)}{\alpha \|\nabla f(x_t)\|} = \nabla f(x_t) \cdot \left(-\frac{\nabla f(x_t)}{\|\nabla f(x_t)\|} \right) = -\|\nabla f(x_t)\| < 0$$

Z definicji granicy Cauchy'ego:

$$\forall_{\epsilon \in \mathbb{R}^+} \exists_{\delta \in \mathbb{R}^+} \forall_{\alpha < \delta} \left| \frac{\Delta_\alpha f(x_t)}{\alpha \|\nabla f(x_t)\|} + \|\nabla f(x_t)\| \right| < \epsilon$$

Niech $\epsilon = \|\nabla f(x_t)\|$, wtedy z powyższego wynika:

$$\exists_\delta \in \mathbb{R}^+ - \|\nabla f(x_t)\| < \frac{\Delta_\alpha f(x_t)}{\alpha \|\nabla f(x_t)\|} + \|\nabla f(x_t)\| < \|\nabla f(x_t)\|$$

$$-2\|\nabla f(x_t)\| < \frac{\Delta_\alpha f(x_t)}{\alpha \|\nabla f(x_t)\|} < 0$$

Ponieważ $\alpha > 0$ mamy:

$$\alpha \|\nabla f(x_t)\| > 0$$

Stąd dostajemy:

$$\forall_{\alpha<\delta} \Delta_\alpha f(x_t) < 0 \implies f(x_{t+1}) < f(x_t)$$

□

Powyższe twierdzenie w intuicyjny sposób ilustruje, że dla szerokiej gamy funkcji zejście gradientowe generuje ciąg, w którego punktach wartości funkcji maleją. Teraz przy silniejszych (przedstawionych na początku) założeniach pokażemy, że zejście gradientowe jest zbieżne. Zaczniemy od kilku lematów, pierwszy jest dość ogólny i będzie także potrzebny w innych dowodach.

Lemat 1.1. Niech $\mathcal{C}^1 \ni f : \mathbb{R}^N \to \mathbb{R}$ i $x, y \in \mathbb{R}^N$ wtedy:

$$f(y) \leq f(x) + \nabla f(x) \cdot (y - x) + \max_{0 \leq t \leq 1} \|\nabla f(x + t(y - x)) - \nabla f(x)\| \|y - x\|$$

Dowód. Niech $\phi(t) = x + t(y - x)$ dla $t \in [0, 1]$ wtedy:

$$f(y) = f(x) + \int_0^1 f(\phi(t))' \, dt = f(x) + \int_0^1 \nabla f(\phi(t)) \cdot \phi'(t) \, dt$$

$$= f(x) + \nabla f(x) \cdot (y - x) + \int_0^1 (\nabla f(x + t(y - x)) - \nabla f(x)) \cdot (y - x) \, dt$$

$$\leq f(x) + \nabla f(x) \cdot (y - x) + \max_{0 \leq t \leq 1} \|\nabla f(x + t(y - x)) - \nabla f(x)\| \|y - x\| \cos \theta_t$$

$$\leq f(x) + \nabla f(x) \cdot (y - x) + \max_{0 \leq t \leq 1} \|\nabla f(x + t(y - x)) - \nabla f(x)\| \|y - x\|$$

□

Lemat 1.2. Dla $\mathcal{C}^1 \ni f : \mathbb{R}^N \to \mathbb{R}$ wypukłej i $x, y \in \mathbb{R}^N$ mamy:

$$f(x) - f(y) \leq \nabla f(x) \cdot (x - y)$$

Dowód. Dzieląc stronami nierówność z definicji wypukłości przez $t \neq 0$ otrzymujemy:

$$\frac{f(x + t(y - x)) - f(x)}{t} \leq f(y) - f(x)$$

Przechodząc do granicy przy $t \to 0$ dostajemy po lewej stronie definicję pochodnej złożenia i dalej:

$$\nabla f(x) \cdot (y - x) \leq f(y) - f(x)$$

Mnożąc stronami przez -1 dostajemy tezę. □

Lemat 1.3. Dla $\mathcal{C}^1 \ni f : \mathbb{R}^N \to \mathbb{R}$ L-gładkiej i $x, y \in \mathbb{R}^N$ mamy:

$$f(y) \leq f(x) + \nabla f(x) \cdot (y - x) + \frac{L}{2} \|y - x\|^2$$

Dowód. Początek dowodu przebiega analogicznie do dowodu lematu 1.1, aż do otrzymania:

$$f(y) \leq f(x) + \nabla f(x) \cdot (y-x) + \int_0^1 (\nabla f(x + t(y-x)) - \nabla f(x)) \cdot (y-x)\, dt$$

$$\leq f(x) + \nabla f(x) \cdot (y-x) + \int_0^1 \|\nabla f(x + t(y-x)\| - \nabla f(x)\| \|y-x\|\, dt$$

Korzystając z L-gładkości dostajemy:

$$f(y) \leq f(x) + \nabla f(x) \cdot (y-x) + \int_0^1 Lt\|y-x\|^2\, dt$$

$$\leq f(x) + \nabla f(x) \cdot (y-x) + \frac{L}{2}\|y-x\|^2$$

□

Lemat 1.4. Dla $\mathcal{C}^1 \ni f : \mathbb{R}^N \to \mathbb{R}$ wypukłej i L-gładkiej i $x, y \in \mathbb{R}^N$ mamy:

$$\frac{1}{L}\|\nabla f(x) - \nabla f(y)\|^2 \leq (\nabla f(y) - \nabla f(x)) \cdot (y-x)$$

Dowód.

$$f(x) - f(y) = f(x) - f(z) + f(z) - f(y)$$

Z lematów 1.2 i 1.3 mamy:

$$f(x) - f(y) \leq (\nabla f(x) \cdot (x-z)) + (\nabla f(y) \cdot (z-y)) + \frac{L}{2}\|z-y\|^2$$

Podstawiając $z = y - \frac{1}{L}(\nabla f(y) - \nabla f(x))$ otrzymujemy:

$$f(x) - f(y) \leq \nabla f(x) \cdot (x - y + \frac{1}{L}\Delta) + \nabla f(y) \cdot (-\frac{1}{L}\Delta) + \frac{1}{2L}\|\Delta\|^2$$

Dla skrócenia zapisu oznaczyliśmy $\Delta = \nabla f(y) - \nabla f(x)$. Po pogrupowaniu otrzymujemy:

$$f(x) - f(y) \leq \nabla f(x) \cdot (x-y) - \frac{1}{2L}\|\nabla f(y) - \nabla f(x)\|^2$$

Na podstawie powyższego dla naprzemiennych podstawień otrzymujemy:

$$\begin{cases} \frac{1}{2L}\|\nabla f(y) - \nabla f(x)\|^2 \leq f(y) - f(x) - \nabla f(x) \cdot (y-x) \\ \frac{1}{2L}\|\nabla f(x) - \nabla f(y)\|^2 \leq f(x) - f(y) + \nabla f(y) \cdot (y-x) \end{cases}$$

Sumując stronami otrzymujemy tezę. □

Twierdzenie 1.2. (Zbieżność zejścia gradientowego)
Niech $\mathcal{C}^1 \ni f : \mathbb{R}^N \to \mathbb{R}$ wypukła, L-gładka i $\alpha \leq \frac{1}{L}$ wtedy dla każdego $x^* \in \arg\min f$ i $t \in \mathbb{N}$ mamy:

$$f(x_t) - \inf f \leq \frac{\|x_0 - x^*\|^2}{2\alpha t}$$

Dowód.

$$\|x_{t+1} - x^*\|^2 = \|x_t - x^* - \frac{1}{L}\nabla f(x_t)\|^2$$

$$= \|x_t - x^*\|^2 - \frac{2}{L}(\nabla f(x_t) \cdot (x_t - x^*)) + \frac{1}{L^2}\|\nabla f(x_t)\|^2$$

Z lematu 1.4 oraz warunku koniecznego minimum $\nabla f(x^*) = 0$ dostajemy:

$$\|x_{t+1} - x^*\|^2 \leq \|x_t - x^*\|^2 - \frac{1}{L^2}\|\nabla f(x_t)\|^2$$

Widać, że $\|x_t - x^*\|^2$ jest malejącym ciągiem, a więc:

$$\|x_t - x^*\| \leq \|x_0 - x^*\| \tag{*}$$

Z lematu 1.3 mamy:

$$f(x_{t+1}) \leq f(x_t) - \frac{1}{L}\|\nabla f(x_t)\|^2 + \frac{1}{2L}\|\nabla f(x_t)\|^2$$

Odejmując $f(x^*)$ stronami:

$$f(x_{t+1}) - f(x^*) \leq f(x_t) - f(x^*) - \frac{1}{2L}\|\nabla f(x_t)\|^2 \tag{**}$$

Z wypukłości i lematu 1.2 otrzymujemy:

$$f(x_t) - f(x^*) \leq \nabla f(x_t) \cdot (x_t - x^*) \leq \|\nabla f(x_t)\|\|(x_t - x^*)\| \overset{(*)}{\leq} \|\nabla f(x_t)\|\|(x_0 - x^*)\|$$

Przekształcając powyższe równanie aby ograniczyć $\|\nabla f(x_t)\|^2$ z góry i wstawiając do $(**)$ otrzymujemy:

$$f(x_{t+1}) - f(x^*) \leq f(x_t) - f(x^*) - \frac{1}{2L\|x_0 - x^*\|^2}(f(x_t) - f(x^*))$$

Teraz oznaczmy $\delta_t = f(x_t) - f(x^*)$ i $\beta = \frac{1}{2L\|x_0 - x^*\|^2}$:

$$\delta_{t+1} \leq \delta_t - \beta\delta_t^2 \iff \beta\frac{\delta_t}{\delta_{t+1}} \leq \frac{1}{\delta_{t+1}} - \frac{1}{\delta t} \overset{\delta_{t+1} \leq \delta_t}{\iff} \beta \leq \frac{1}{\delta_{t+1}} - \frac{1}{\delta_t}$$

Sumując po $t = 0, \ldots, T-1$ dostajemy:

$$T\beta \leq \frac{1}{\delta_T} - \frac{1}{\delta_0} \leq \frac{1}{\delta_T}$$

A po przestawieniu wyrażeń:

$$f(x_T) - f(x^*) \leq \frac{1}{\beta T} = \frac{2L\|x_0 - x^*\|^2}{T}$$

□

Powyższe twierdzenie nie tylko pokazuje, że zejście gradientowe jest zbieżne, ale również pozwala oszacować tempo jego zbieżności.

Wniosek 1.1. (Tempo zbieżności zejścia gradientowego)
Tempo zbieżności zejścia gradientowego to $\mathcal{O}(1/t)$.

Dowód. Weźmy $\alpha = \frac{1}{L}$ i dowolnie małe $\epsilon > 0$. Wtedy z powyższego twierdzenia:

$$t \geq \frac{2L\|x_0 - x^*\|^2}{\epsilon} \implies f(x_t) - f(x^*) \leq \epsilon$$

□

1.3 Proces uczenia nadzorowanego

Sieć neuronową uczymy analogicznie do wielu innych modeli: poprzez dopasowywanie parametrów aby minimalizować zadaną funkcję kosztu. Realizacja tego procesu jest oparta o wyżej opisane zejście gradientowe.

Niech $\mathcal{M}(w_0, \ldots, w_n)$ będzie naszą siecią, a w_i oznacza jej i-tą wagę. Mamy zadany zbiór $\mathcal{D} = \{(X_0, y_0), \ldots, (X_m, y_m)\}$ nazywany *zbiorem uczącym* gdzie X_i to i-te wejście modelu, a y_i i-ta oczekiwana odpowiedź. Dodatkowo obieramy *funkcje kosztu*: $\mathcal{L} : \mathbb{R}^N \ni ((y_0, \ldots, y_m), (\hat{y_0}, \ldots, \hat{y_m})) \to \mathbb{R}$,

gdzie $\hat{y_i} = \mathcal{M}(w_0, \ldots, x_m)(X_i)$. Minimalizujemy ją modyfikując wagi za pomocą zejścia gradientowego:

$$w_{i,t+1} = w_{i,t} - \alpha \frac{d\mathcal{L}(\ldots)}{dw_{i,t}}$$

W praktyce nie operuje się na pojedynczych wagach tylko wektorach wag dla warstw. Ich aktualizacja dzieje się miedzy kolejnymi krokami automatycznego różniczkowania wstecz, które zapewnia odpowiedni gradient:

$$W_{j,t+1} = W_{j,t} - \alpha \nabla_{W_{j,t}} \mathcal{L}(\ldots)$$

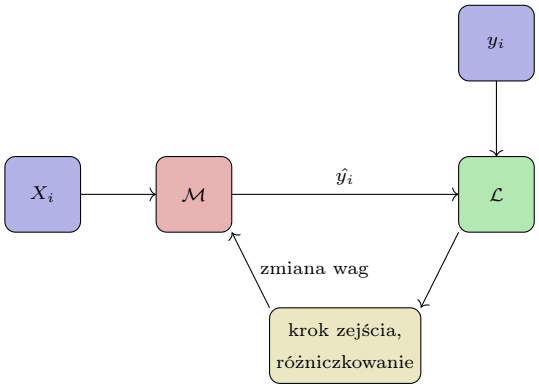
gdzie $W_{j,\cdot}$ to wektor wag j-tej warstwy.

1.3.1 Stochastyczne zejście gradientowe

W praktyce, ze względu na ograniczenia pamięciowe oraz chęć ominięcia lokalnych minimów, nie wykonujemy każdego kroku na całym zbiorze uczącym. Najpierw dzielimy go na rozłączne podzbiory $\mathcal{D}_i$ równej długości k nazywane *seriami*.

Następnie wykonujemy wyżej opisany krok zejścia gradientowego dla każdego $\mathcal{D}_i$ oddzielnie. Jedno przejście przez wszystkie serie określamy mianem *epoki*.

Dla $k > 1$ powyższy proces nazywamy **seryjnym zejściem gradientowym** [*ang. mini batch gradient descent*], a dla $k = 1$ nazywamy **stochastycznym zejściem gradientowym** [*ang. stochastic gradient descent* - SGD]. W obu przypadkach proces powtarzamy przez wiele epok, w każdej epoce zmieniając podział $\mathcal{D}$. Następnie sprawdzamy wynik uczenia na *zbiorze testowym* $\mathcal{T} = \{(X_0, y_0), \ldots, (X_l, y_l)\}$, który jest rozłączny z $\mathcal{D}$.



Rysunek 4: Wizualizacja pojedyńczej iteracji procesu stochastycznego zejścia gradientowego. Otrzymane dane są przekazujemy do modelu i obliczamy funkcję kosztu na podstawie wyjścia modelu i oczekiwanego wyjścia. Na jej podstawie warstwa po warstwie obliczamy gradient i aktualizujemy wagi.

1.3.2 Adam

Od jego wprowadzenia w 2015 roku *Adam* pozostaje najpopularniejszym algorytmem optymalizacji używanym w uczeniu głębokim. Opiera się o zasadę pędu inspirowaną fizyką. W wielu przypadkach zapewnia lepsze wyniki niż klasyczne stochastyczne zejście gradientowe.

Jego krok przebiega następująco:

$$m_{t+1} = \beta_1 m_t + (1 - \beta_1) \nabla W_{j,t} \mathcal{L}(\cdots)$$

$$v_{t+1} = \beta_2 v_t + (1 - \beta_2) (\nabla W_{j,t} \mathcal{L}(\cdots))^2$$

$$\hat{m_t} = \frac{m_t}{1 - \beta_1^t}$$

$$\hat{v_t} = \frac{v_t}{1 - \beta_2^t}$$

$$W_{j,t+1} = W_{j,t} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

gdzie $W_{j,t}$ reprezentuje macierz wag j-tej warstwy w momencie t, a $\alpha, \beta_1, \beta_2, \epsilon \in \mathbb{R}_+$ są parametrami algorytmu. m_t reprezentuje jak istotny jest poprzedni krok i jest określane mianem *pędu*. Jest obliczane jako wykładnicza średnia ruchoma. Jego odpowiednik dla kwadratu gradientu to v_t. Ponieważ $m_0 = v_0 = 0$ wprowadzamy $\hat{m}, \hat{v}$ aby trening nie był zbyt wolny dla początkowych iteracji. Całość jest normalizowana przez aproksymację średniej długości gradientu $\sqrt{\hat{v}_t}$.

1.4 Sieci konwolucyjne

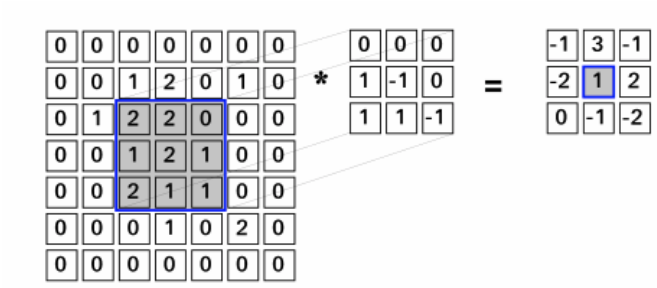
Sieci konwolucyjne to bardzo potężne narzędnie analizy obrazów. Dopiero od momentu ich powstawnia możemy mówić o *uczeniu głębokim*. Bazują na idei filtra konwolucyjnego będącej z klasyczną metodą przetwarzania obrazów. Wykonują ustalone operacje w otoczeniu każdego piksela przez co uczą się modelować zależności między sąsiadującymi pikselami, które mają reprezentować cechy obrazu jak tekstura czy krawędzie.

1.4.1 Filtr konwolucyjny

Definicja 1.6. Filtr konwolucyjny to tensor $W = [w_{ijk}]$, gdzie $w_{ijk} \in \mathbb{R}$. Pierwsze dwa wymiary są niewielkie w stosunku do rozmiaru obrazu, a trzeci taki sam jak obrazu. Filtr działa na każdym pikselu obrazu $H = [h_{ijk}]$, czego wynikiem jest tensor $G = [g_{ij}]$ obliczany następująco:

$$g_{xy} = \sum_{i,j,k} h_{i,j,k} w_{x+i,y+j,k}$$

Powyższą operację nazywamy *splotem* i oznaczamy przez $*$.



Rysunek 5: Wizualizacja operacji splotu. Źródło: [Mas]

Splot jest operacją liniową i niezmienniczą ze względu na translację. Filtry konwolucyjne są dobrze znane w klasycznej analizie obrazów i służą między innymi do wykrywania krawędzi w danym kierunku oraz realizacji rozmycia lub wyostrzenia.

1.4.2 Budowa sieci

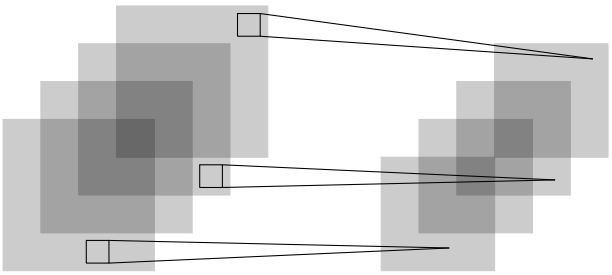
Konwolucyjna sieć neuronowa podobnie jak zwykła jest zbudowana z warstw. Jednak w przeciwieństwie do zwykłej każda z warstw definiuje zestaw filtrów, które są używane do przekształcenia danych wejściowych. Sieć uczy się poprzez dostosowywanie elementów tych filtrów - odpowiednik dostosowywania wag w zwykłych sieciach.

Definicja 1.7. Konwolucyjna sieć neuronowa składa się z n warstw $L_0, \ldots, L_n$. Każda z nich używa zestawu k_i filtrów $W_1, \ldots, W_{k_i}$. Biorąc odpowiednią funkcję aktywacji f, definiujemy $F(H)$ będącą aplikacją funkcji f do każdego elementu H, gdzie H to wejściowy tensor. Aktywację i-tej warstwy definiujemy następująco:

$$L_i(H) = [F(H * W_1), \ldots, F(H * W_{k_i})]$$

Sieć definiujemy jako złożenie warstw.

W praktyce między filtry konwolucyjne w sieciach wprowadza się operacje takie jak branie największej wartości z jakiegoś obszaru. Takie operacje nie są dostosowywane w trakcie treningu, ale pomagają przyspieszyć działanie i kontrolować wymiarowość danych w kolejnych przejściach.



Rysunek 6: Wizualizacja działania warstwy konwolucyjnej. Źródło: [Stu]

1.5 Sieci rekurencyjne

Do tej pory rozważaliśmy modele przetwarzające dane wektorowe o ustalonej długości. Teraz wprowadzimy sieci neuronowe operujące na danych sekwencyjnych to znaczy takich gdzie X_i, y_i są sekwencjami wektorów o zmiennej długości. Takie modele mogą przetwarzać dźwięk, dane tekstowe, czy nawet dane finansowe.

Definicja 1.8. Rekurencyjna sieć neuronowa $\mathcal{M}$ w każdym kroku oprócz parametrów wag θ przyjmuje również swój poprzedni stan h oraz kolejny wektor danych z sekwencji x, a zwraca wynik o:

$$o_{t+1}, h_{t+1} = \mathcal{M}(x_t, h_t, \theta)$$

Aby wyprodukować kolejne h oraz o sieć rekurencyjna przetwarza dane analogicznie do klasycznej przy użyciu warstw neuronów i funkcji aktywacji.

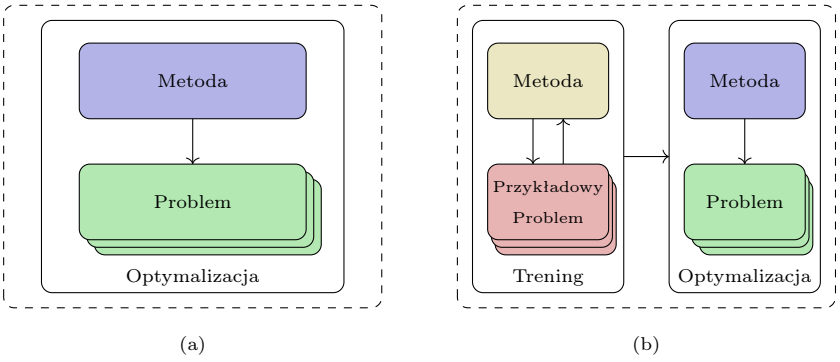
2 Uczenie optymalizacji

2.1 Wstęp

Klasyczne metody optymalizacji są budowane na solidnych teoretycznych fundamentach zapewniających ich zbieżność w odpowiednich warunkach. Są to ściśle zdefiniowane algorytmy, których działanie pozostaje takie samo.

Uczenie optymalizacji [*ang. learning to optimize*] [Che+21] jest alternatywnym paradygmatem, gdzie metoda optymalizacji jest wyłaniana w procesie treningu w oparciu o przykładowe problemy optymalizacyjne z danego rozkładu.

Różnice pomiędzy klasycznym *(a)*, a wyżej opisanym paradygmatem *(b)* reprezentuje poniższa ilustracja.



Rysunek 7: Porównanie paradygmatów optymalizacji. (a) - ten sam algorytm optymalizacji stosujemy na różnych problemach. (b) - mając dane problemy z jakiegoś rozkładu tworzymy metodę

optymalizacji w oparciu o trening na przykładach. Następnnie możemy ją aplikować do innych problemów z tego rozkładu.

Uczenie optymalizacji pozwala na wytrenowanie metody, która dla problemów z danego rozkładu zapewnia szybszą zbieżność. Nie ma jednak gwarantu jej zbieżności w ogólności.

W praktyce funkcje kosztu, które są optymalizowane przy uczeniu głębokich sieci neuronowych nie są *gładkie*, nie mają Lipschitzowskiego gradientu [Ber+21], ani innych potrzebnych własności by zapewnić zbieżność klasycznych metod [RKK19].

W takich wypadkach naturalnym jest użycie uczenia optymalizacji.

2.2 Formalizm w przypadku zejścia gradientowego

Sformalizujmy powyższe rozważania na podstawie zejścia gradientowego.

Rozważmy problem optymalizacyjny $min_x f(x)$, gdzie $x \in \mathbb{R}^d$. Używając metody uczenia optymalizacji w kolejnych krokach otrzymamy: $x_{t+1} = x_t - m(z_t, \phi)$, gdzie $m(z_t, \phi)$ to funkcja sparametryzowana ϕ generująca aktualizację x_t na podstawie wejściowego stanu procesu optymalizacji z_t. W szczególności może to być $\nabla f(x_t)$.

Proces uczenia optymalizacji to po prostu dobieranie parametrów ϕ modelu m, który stanowi kolejny problem optymalizacjny.

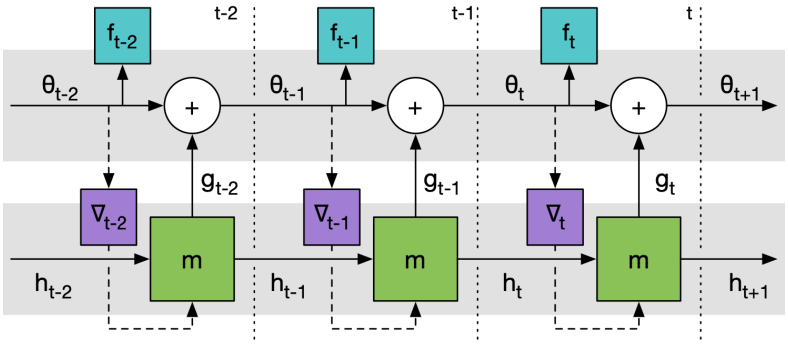
2.3 Realizacja

Oczywistą realizacją uczenia optymalizacji jest zastosowanie rekurencyjnej sieci neuronowej jako modelu m. Takie rozwiązanie zostało zastosowane w pracy [And+16], której krótkie podsumowanie przedstawiamy poniżej.

Rozważmy, w sposób analogiczny do powyższego, problem uczenia pewnej sieci neuronowej jako znalezienie: $argmin_{\theta \in \Theta} f(\theta)$, gdzie $f(\theta)$ to funkcja kosztu, sparametryzowana przez $\theta \in \Theta$. We wspomnianej pracy wprowadzony jest proces:

$$\theta_{t+1} = \theta_t + g_t(\nabla f(\theta_t), \phi)$$

Powyższy proces jest realizowany przez LSTM (rodzaj sieci rekurencyjnej) g (g_t reprezentuje g i jego stan w iteracji t) sparametryzowany przez ϕ w roli modelu m. Daje bardzo zadowalające wyniki w zaprezentowanych eksperymentach. Pokonuje typowe algorytmy optymalizacji używane w uczeniu maszynowym: Adam, SGD i inne.



Rysunek 8: Wizualne przedstawienie działania wyżej przytoczonego procesu. Źródło: [And+16]

3 Zaawansowane modyfikacje zejścia gradientowego

3.1 Metody hipergradientu

W klasycznych procedurach uczenia sztucznych sieci neuronowych przed treningiem należy ustalić wartości hiperparametrów, w szczególności wielkość kroku α. Oczywiście możemy ustalić pewien

ciąg $(\alpha_t)_{t\in\mathbb{N}}$, który dawałby inną wielkość kroku dla każdej iteracji i w praktyce korzysta się z takich rozwiązań, ale nadal jest to sztywno ustalony ciąg.

Co jeśli zastosujemy paradygmat uczenia optymalizacji? Moglibyśmy dostosowywać wartości hiperparametów w trakcie uczenia na podstawie przetwarzanych danych.

Taka metoda dla wielkości kroku została przedstawiona w pracy [Bay+18], a jej krok iteracyjny przedstawiamy poniżej.

$$\alpha_{t+1} = \alpha_t + \beta\nabla f(x_t)\cdot\nabla f(x_{t-1})$$
$$x_{t+1} = x_t - \alpha_t\nabla f(x_t)$$

Wzór na α_{t+1} został wyprowadzony w prosty sposób w oparciu o zejście gradientowe:

$$\alpha_{t+1} = \alpha_t - \beta\frac{\partial f(x_t)}{\partial\alpha}$$

Wystarczy zastosować wzór na x_t i regułę łańcuchową.

Eksperymenty z przytoczonej pracy wskazują na to, że taka metoda ograniczyła potrzebę ręcznego dopasowywania hiperparametrów (mimo posiadania własnego β) i dawała wyniki lepsze lub takie same jak klasyczne algorytmy uczenia.

3.2 Metody ważenia gradientu

W zejściu gradientowym wielkość kroku jest skalarem. W praktyce jednak część gradientu może wpływać na proces uczenia bardziej niż reszta. Wtedy chcielibyśmy mnożyć część gradientu przez większy skalar. Powstały różne metody aby rozwiązać ten problem, jedną z nich reprezentuje nawet Adam, ale my skupiamy się na *metodach ważenia gradientu*, które definiujemy poniżej:

Definicja 3.1. Mianem **metody ważenia gradientu** określamy proces optymalizacji oparty o zejście gradientowe gdzie α jest tensorem rozmiaru gradientu wyznaczanym na podstawie parametrów dostępnych w trakcie uczenia np. gradientu, a krok ma postać:

$$x_{t+1} = x_t - \alpha(\nabla f(x_t))\circ\nabla f(x_t)$$

Flagowym przykładem takiego procesu jest AdaGrad [MS10].

W metodach ważenia należy z góry określić funkcję α. Możemy oczywiście zastosować metodę hipergradientu do tensora ważeń. Rozwinięciem tego pomysłu jest stworzona przez nas i opisana poniżej metoda **GWNMO**.

4 GWNMO

4.1 Wstęp

Bezpośrednią motywacją do stworzenia opisanej w tym rozdziale metody była praca [GRD18]. Eksperymenty przedstawione w niej wskazują, że w trakcie uczenia głębokich sieci neuronowych szybko wyłania się kierunek V_{top} wyznaczany przez część gradientu, który w największym stopniu wpływa na uczenie.

Formalnie: oznaczmy przez $\nabla_{top}f(x_t)$ rzut ortogonalny gradientu w punkcie x_t na V_{top} i zdefiniujmy:

$$f_{top} = \frac{\|\nabla_{top}f(x_t)\|^2}{\|\nabla f(x_t)\|^2}$$

Z powyższej pracy wynika, że f_{top} szybko zmierza do 1 w trakcie treningu.

Przy założeniu, że schodzenie wzdłuż kierunku V_{top} poprawiłoby wyniki zejścia gradientowego, przedstawiamy poniżej realizację tego pomysłu. Nasza metoda czerpie naraz z wyżej wspomnianego uczenia optymalizacji, metod hipergradientowych oraz stanowi dalsze rozwinięcie metody ważenia gradientu.

4.2 Sformułowanie

Niech $\mathcal{C}^1 \ni f(x) : \mathbb{R}^N \to \mathbb{R}$ będzie funkcją wypukłą oraz L-gładką, $\alpha > 0$ wielkością kroku, a $S(x) = \frac{1}{1+e^{-x}}$. Rozwiązujemy problem przybliżenia wartości $x^* \in \arg\min f \neq \emptyset$.

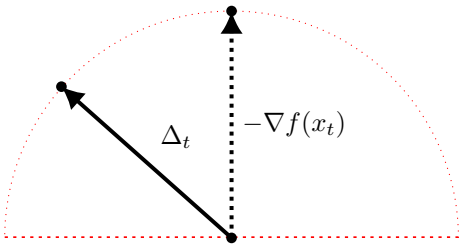
Definicja 4.1. Metoda ważenia gradientu w zejściu gradientowym przez sztuczną sieć neuronową [*ang. gradient weighting neural meta optimizer* - **GWNMO**] to metoda ważenia gradientu oparta o sieć neuronową $\mathcal{M}$, w którym kolejne przybliżenia $(x_t)_{t\in\mathbb{N}}$ są wyznaczane następująco:

$$h_t = S(\mathcal{M}(x_t, \nabla f(x_t)))$$

$$x_{t+1} = x_t - \alpha \frac{\|\nabla f(x_t)\|}{\|h_t \circ \nabla f(x_t)\|} h_t \circ \nabla f(x_t)$$

Dalej oznaczamy $\Delta_t = \frac{\|\nabla f(x_t)\|}{\|h_t \circ \nabla f(x_t)\|} h_t \circ \nabla f(x_t)$.

Korzystamy ze sztucznej sieci neuronowej $\mathcal{M}$, która ma modelować które elementy gradientu tworzą kierunek V_{top} i jakie *ważenie* jest w danym kroku optymalne. W każdym kroku wynik modelu obkładamy sigmoidą aby ograniczyć się do kombinacji barycentrycznych elementów gradientu - nie chcemy zmienić znaku iloczynu skalarnego. Złożenia gradientu i wyniku sieci dodatkowo normalizujemy aby zapewnić zbieżność. Intuicyjnie: nie chcemy żeby zważony gradient powodował bardzo duże kroki, które mogą zaburzyć proces zejścia gradientowego. W eksperymentach testujemy również wersję bez normalizacji i sigmoidy nazywaną **GWNMO'**.



Rysunek 9: Intuicja ważenia kroku z normalizacją i sigmoidą.

4.3 Działanie

Model $\mathcal{M}$ oceniamy funkcją kosztu którą pomaga optymalizować - f. Iteracyjnie dostosowujemy parametry $\mathcal{M}$ na podstawie poprzedniego kroku, a następnie modyfikujemy parametr x_t *ważąc* jego gradient siecią $\mathcal{M}$.

Jest to proces analogiczny do tego opisanego w pracy [Cha+22].

Ponieważ pracujemy na sekwencjach danych, a ważenie gradientu powinno zależeć od etapu treningu zastosowanie rekurencyjnej sieci neuronowej tak jak w pracy [And+16] wydaje się oczywiste. My jednak w celu uproszczenia modelu zdecydowaliśmy się ograniczyć do zwykłej sieci. Liczymy, że kontekst dotyczący etapu uczenia może zostać wydedukowany przez sieć ważącą na podstawie danych wejściowych X_i, aktualnych wag sieci docelowej oraz gradientu funkcji kosztu względem nich.

4.3.1 Algorytm uczenia

Poniżej przedstawiamy formalny algorytm uczenia sieci neuronowych korzystający z wyżej wprowadzonego ważenia GWNMO.

W praktyce implementujemy i testujemy seryjny odpowiednik poniższego algorytmu, który może być wyprowadzony w prosty sposób na podstawie poniższego. Do uczenia $\mathcal{M}$ używamy algorytmów Adam lub SGD.

Algorytm 1 GWNMO w stochastycznym zejściu gradientowym

Require: zbiór uczący $\mathcal{D} = \{X_t, y_t\}$, funkcję kosztu f, model $\mathcal{M}$ sparametryzowany przez θ, model uczony $\mathcal{T}$ sparametryzowany przez ϕ, wielkość korku $\alpha > 0$ i *meta-wielkość kroku* $\beta > 0$, numer iteracji t

Inicjalizujemy θ, ϕ

while nie skończyliśmy **do**

$\mathcal{L} \leftarrow f(\mathcal{T}(\phi_t; X_t), y_t)$

$h \leftarrow S(\mathcal{M}(\theta_t; X_t, \phi_t, \nabla_{\phi_t}\mathcal{L}))$

$\phi_{t+1} \leftarrow \phi_t - \alpha \frac{\|\nabla_{\phi_t}\mathcal{L}\|}{\|h \circ \nabla_{\phi_t}\mathcal{L}\|} h \circ \nabla_{\phi_t}\mathcal{L}$

$\mathcal{L}' \leftarrow f(\mathcal{T}(\phi_{t+1}; X_{t+1}), y_{t+1})$

$\theta_{t+1} \leftarrow \theta_t - \beta \nabla_{\theta_t}\mathcal{L}'$

end while

4.4 Analiza

4.4.1 Złożoność czasowa

Rozważmy pojedyńczą iterację. Załóżmy, że aktualizacja wagi zajmuje $\mathcal{O}(U)$, a wyliczenie gradientu dla jednej warstwy zajmuje $\mathcal{O}(T)$. Wtedy dla sieci docelowej $\mathcal{T}$ o głębokości N i szerokości każdej warstwy W oraz sieci ważącej $\mathcal{M}$ głębokości M i szerokości każdej warstwy Z mamy złożoność $\mathcal{O}(N(T + WU) + M(T + ZU))$.

Bez użycia $\mathcal{M}$ otrzymamy złożoność $\mathcal{O}(N(T + WU))$. Jak widać nasza metoda nie powoduje dodatkowych komplikacji aktualizacji wag, jedyna różnica polega na konieczności aktualizacji wag $\mathcal{M}$.

4.4.2 Lokalne malenie

Poniższe twierdzenie obrazuje, że zejście gradientowe z naszym ważeniem faktycznie utrzymuje cechę malenia wartości funkcji w kolejnych punktach.

Twierdzenie 4.1. (Lokalne malenie funkcji kosztu w zejściu gradientowym z ważeniem GWNMO) Niech $\mathcal{C}^1 \ni f : \mathbb{R}^N \to \mathbb{R}$, $\forall_{x \in \mathbb{R}^N} \nabla f(x) \neq 0$ i $(x_t)_{t \in \mathbb{N}}$ jest generowany algorytmem 1 wtedy:

$$\exists_{\alpha > 0} f(x_{t+1}) < f(x_t)$$

Dowód. Zacznijmy od zastosowania lematu 1.1:

$$f(x_{t+1}) \leq f(x_t) + \nabla f(x_t) \cdot (x_{t+1} - x_t) + \max_{0 \leq t \leq 1} \|\nabla f(x_t + t(x_{t+1} - x_t)) - \nabla f(x_t)\| \|x_{t+1} - x_t\|$$

$$f(x_{t+1}) \leq f(x_t) + \nabla f(x_t) \cdot (-\alpha \Delta_t) + \max_{0 \leq t \leq 1} \|\nabla f(x_t - t\alpha\Delta_t) - \nabla f(x_t)\| \|-\alpha\Delta_t\|$$

Przez θ_t oznaczamy kąt między $-\Delta_t$, a $-\nabla f(x_t)$

$$f(x_{t+1}) - f(x_t) \leq -\alpha \|\nabla f(x_t)\| \|\Delta_t\| \|\cos(\theta_t)\| + \max_{0 \leq t \leq 1} \alpha \|\nabla f(x_t - t\alpha\Delta_t) - \nabla f(x_t)\| \|\Delta_t\|$$

$$f(x_{t+1}) - f(x_t) \leq -\alpha \|\nabla f(x_t)\| \|\Delta_t\| \left(\|\cos(\theta_t)\| - \max_{0 \leq t \leq 1} \frac{\|\nabla f(x_t - t\alpha\Delta_t) - \nabla f(x_t)\|}{\|\nabla f(x_t)\|} \right)$$

Widzimy że teza jest spełniona gdy prawa część nierówności będzie dodatnia.
Oznaczmy $A = \|\cos(\theta_t)\| - \max_{0 \leq t \leq 1} \frac{\|\nabla f(x_t - t\alpha\Delta_t) - \nabla f(x_t)\|}{\|\nabla f(x_t)\|}$

$$f(x_{t+1}) \leq f(x_t) \iff A \geq 0$$

Z ciągłości ∇f i postaci A wnioskujemy, że istnieje takie α, że nierówność jest spełniona.

□

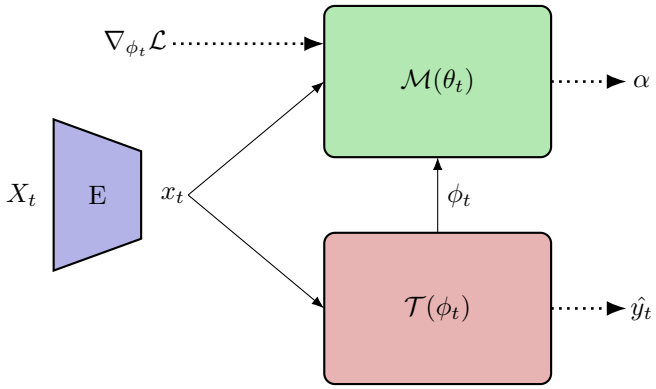
4.5 Eksperymenty

Eksperymenty były przeprowadzane na zadaniu klasyfikacyjnym (model ma dla danego obrazu zwrócić odpowiednią etykietę liczbową) na zbiorach danych: CIFAR10, MNIST i FMNIST. Naszą metodę porównywaliśmy do algorytmu Adam oraz metody hipergradientowej z macierzą wielkości kroku dla każdego elementu gradientu (oznaczanej HG). Co epokę sprawdzaliśmy sieć docelową na zbiorze testowym i zapisywaliśmy wyniki.

4.5.1 Architektura

Sieć docelowa składa się z pretrenowanej na zbiorze ImageNet sieci konwolucyjnej ResNet18 (oznaczanej E), gdzie ostatnia warstwa klasyfikacyjna została zastąpiona przez dwie klasyczne warstwy. W procesie uczenia modyfikujemy jedynie wagi dwóch ostatnich warstw, pozostałe są zamrożone. Dlatego faktycznie naszą siecią docelową są te dwie ostatnie warstwy, oznaczamy je $\mathcal{T}(\phi_t)$, a ich wyjście jako $\hat{y_t}$. W metodzie GWNMO zastosowana sieć ważąca $\mathcal{M}(\theta_t)$ składa się z trzech klasycznych warstw. Ważenie generowane przez nią oznaczamy α. W celu ograniczenia liczby parametrów $\mathcal{M}$ przekazujemy do niej wynik działania sieci $E(X_t) = x_t$ zamiast X_t.

W przypadku użycia metody Adam, lub HG oczywiście pomijaliśmy sieć $\mathcal{M}$.



Rysunek 10: Wizualizacja architektury wykorzystanej w eksperymentach. Przypadek dla algorytmu 1 i ważenia GWNMO. E - pretrenowana sieć konwolucyjna ResNet18, $\mathcal{M}$ - trójwarstwowa sieć ucząca, $\mathcal{T}$ - dwuwarstwowa sieć uczona. $\mathcal{T}$ otrzymuje wyjście E - x_t i zwraca odpowiedź, $\mathcal{M}$ otrzymuje x_t, ϕ_t oraz odpowiedni gradient i zwraca ważenie.

Jak widać użyta przez nas architektura jest bardzo prosta, jednak eksperymenty pokazały, że nawet takie minimalne zastosowanie naszej metody osiąga obiecujące wyniki.

4.5.2 Wyniki

W poniższych tabelkach pierwsza liczba to ilość dobrych odpowiedzi w serii [*ang. accuracy*], a druga wartość funkcji kosztu [*ang. loss*]. Porównywane są najlepsze wartości osiągnięte na zbiorze testowym. Najlepsze wyniki są pogrubione.

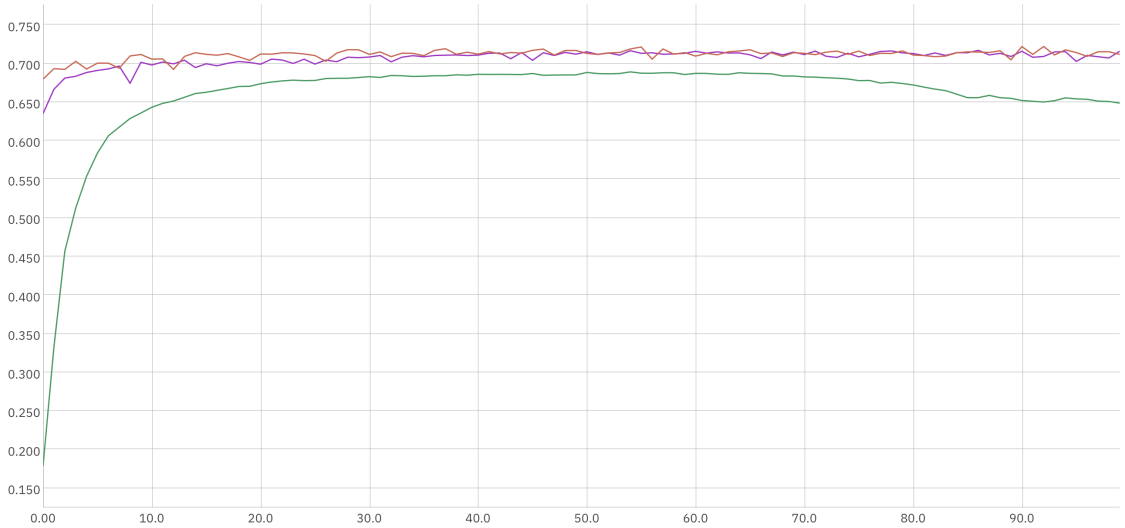
Zbiór danych	GWNMO(SGD)		GWNMO(Adam)		Adam		HG	
CIFAR10	71.7%	0.820	72.1%	**0.808**	**72.2%**	0.815	68.9%	0.898
MNIST	96.2%	0.126	**96.4%**	0.120	**96.4%**	**0.119**	94.8%	0.167
FMNIST	86.4%	0.383	**87.2%**	**0.375**	87.0%	0.377	83.1%	0.489

Tabela 1: Porównanie odsetku dobrych odpowiedzi w serii i wartości funkcji kosztu dla różnych algorytmów. GWNMO(SGD) - Algorytm 1 z ważeniem GWNMO, gdzie sieć ważącą dostosowujemy za pomocą SGD. GWNMO(Adam) - Analogicznie z użyciem algorytmu Adam.

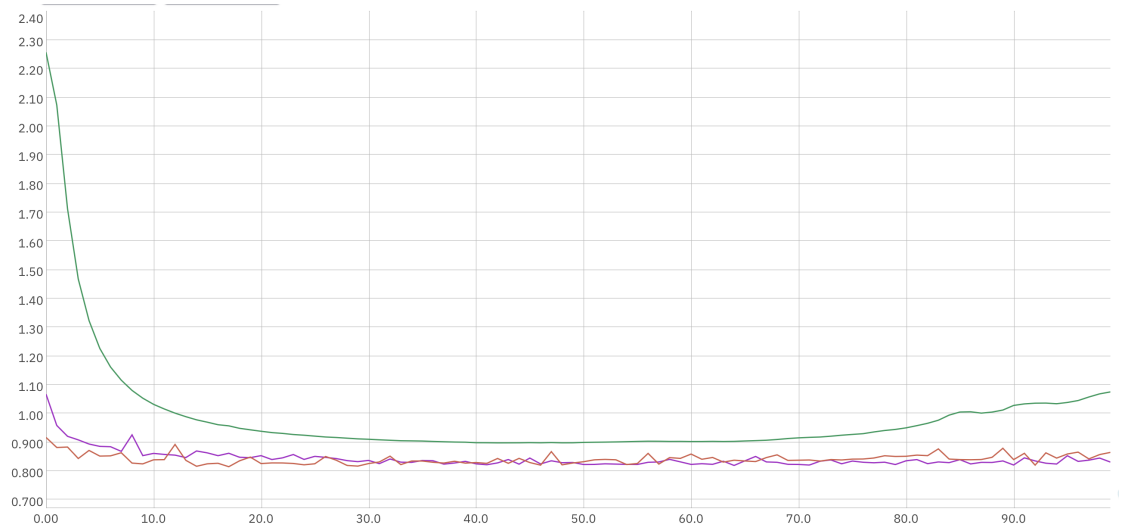
Z powyższej tabeli wynika, że GWNMO osiąga znacznie lepsze wyniki niż proste zaaplikowanie metody hipergradientu do ważenia gradientu. Przy zastosowaniu algorytmu Adam dla sieci ważącej

otrzymujemy wyniki takie same, lub lepsze niż zastosowanie go bez GWNMO. Co więcej, w drugim przypadku, mimo bardzo małej architektury sieci $\mathcal{M}$ i zastosowania prostego stochastycznego zejścia gradientowego otrzymujemy niewiele gorsze wyniki.

Z poniższych wykresów widać, że pod koniec treningu GWNMO(SGD) radzi sobie lepiej z minimalizacją funkcji kosztu niż Adam. Bardzo prawdopodobne, że gdyby wydłużyć trening nasza metoda osiągnełaby lepsze rezultaty.



Rysunek 11: Odsetek dobrych odpowiedzi na zbiorze testowym na przestrzeni epok. Kolor zielony - HG, różowy - GWNMO(SGD), czerwony - Adam.



Rysunek 12: Wartość funkcji kosztu na zbiorze testowym na przestrzeni epok. Kolor zielony - HG, różowy - GWNMO(SGD), czerwony - Adam.

GWNMO		GWNMO'	
72.1%	**0.808**	69.0%	0.893

Tabela 2: Porównanie odsetku dobrych odpowiedzi w serii i wartości funkcji kosztu dla różnych wersji ważenia GWNMO(Adam) na zbiorze CIFAR10.

Jak widzimy z powyższej tabelki sieć $\mathcal{M}$ nie potrzebuje ani normalizacji ani sigmoidy aby dobrać ważenia odpowiednio - faktycznie dostosowuje się w procesie treningu. Jednak dzięki zastosowaniu tych usprawnień osiąga znacząco lepsze wyniki.

4.5.3 Implementacja

Implementacja przeprowadzonych eksperymentów jest dostępna pod adresem: `https://github.com/OneAndZero24/GWNMO`.

Przy implementacji została użyta biblioteka PyTorch oraz [Arn+20], całość jest napisana w języku programowania Python.

4.6 Dalsze prace

Jak widać zaproponowana metoda nie jest jeszcze do końca zbadana. To dopiero początek pracy nad nią zarówno od strony teoretycznej jak i praktycznej.

W ramach dalszych prac planujemy:

- przeprowadzić eksperymenty, w których przeprowadzamy serię treningów na różnych sieciach docelowych zachowując między treningami sieć ważącą
- zbadać formalnie jej zbieżność
- zbadać jej zależność od parametrów i stabilność uczenia
- zaaplikować ją do problemów meta-uczenia
- zaimplementować oraz przetestować bardziej złożone architektury sieci uczącej (np. zastosować mechanizm atencji)
- zastosować metody kompresji danych wejściowych do sieci, lub metodę sekwencyjnego ich przekazywania aby umożliwić działanie na modelach docelowych z większą ilością parametrów
- przeprowadzić eksperymenty na większych architekturach sieci docelowej

5 Podsumowanie

W powyższej pracy wprowadziliśmy nową, czerpiącą z wielu aktualnych badań metodę ważenia gradientu w zejściu gradientowym - GWNMO. Wyniki eksperymentów jasno pokazują, że jest to pomysł z dużym potencjałem i możliwościami dalszego rozwoju. Nawet w najprostszej formie dorównuje aktualnie najpowszechnieszemu algorytmowi Adam.

Przed sformułowaniem naszej metody omówiliśmy potrzebny do jej zrozumienia i oceny szeroki kontekst zarówno innych badań jak i dziedziny uczenia maszynowego.

Jest to pierwsza spisana praca prezentująca tą metodę. Planujemy dalej ją rozwijać zarówno od strony formalnej jak i zastosowań.

Bibliografia

[MS10] H. B. McMahan i Matthew J. Streeter. *Adaptive Bound Optimization for Online Convex Optimization*. 2010.

[And+16] Marcin Andrychowicz i in. *Learning to learn by gradient descent by gradient descent*. 2016. arXiv: 1606.04474 [cs.NE].

[Bay+18] Atilim Gunes Baydin i in. *Online Learning Rate Adaptation with Hypergradient Descent*. 2018. arXiv: 1703.04782 [cs.LG].

[GRD18] Guy Gur-Ari, Daniel A. Roberts i Ethan Dyer. *Gradient Descent Happens in a Tiny Subspace*. 2018. arXiv: 1812.04754 [cs.LG].

[RKK19] Sashank J. Reddi, Satyen Kale i Sanjiv Kumar. *On the Convergence of Adam and Beyond*. 2019. arXiv: 1904.09237 [cs.LG].

[Arn+20] Sébastien M R Arnold i in. "learn2learn: A Library for Meta-Learning Research". W: (sierp. 2020). arXiv: 2008.12284 [cs.LG]. URL: http://arxiv.org/abs/2008.12284.

[Ber+21] Jeremy Bernstein i in. *On the distance between two neural networks and the stability of learning*. 2021. arXiv: 2002.03432 [cs.LG].

[Che+21] Tianlong Chen i in. *Learning to Optimize: A Primer and A Benchmark*. 2021. arXiv: 2103.12828 [math.OC].

[Cha+22] Kartik Chandra i in. *Gradient Descent: The Ultimate Optimizer*. 2022. arXiv: 1909.13371 [cs.LG].

[Tab+22] Jacek Tabor i in. *Głębokie Uczenie Wprowadzenie*. Helion, 2022. ISBN: 978-83-283-8541-2.

[GG23] Guillaume Garrigos i Robert M. Gower. *Handbook of Convergence Theorems for (Stochastic) Gradient Methods*. 2023. arXiv: 2301.11235 [math.OC].

[Mas] ESA Grupa Uczenia Maszynowego. *Convolutional Neural Networks Introduction*. URL: https://www.cosmos.esa.int/web/machine-learning-group/convolutional-neural-networks-introduction.

[Neu] Izaak Neutelings. *Neural Networks*. URL: https://tikz.net/neural_networks/.

[Stu] David Stutz. *Illustrating (Convolutional) Neural Networks in LaTeX with TikZ*. URL: https://davidstutz.de/illustrating-convolutional-neural-networks-in-latex-with-tikz/.