Projekt

Głębokie Sieci Neuronowe

Przegląd literatury i przykładów obliczeniowych o sieciach typu Long Short-Term Memory (LSTM) oraz algorytmach Stochastic Gradient Descent with Momentum (SGDM) i Adaptive Moment Estimation (ADAM)

Martyna Żukowska, 252877

Jeremi Jarosz, 250426

Prowadzący:
dr hab. inż. Andrzej Rusiecki

Wydział Informatyki i Telekomunikacji

Politechniki Wrocławskiej

19 listopada 2023

Spis treści

1 Rekurencyjne Sieci Neuronowe i ich wady

Sieci Long Short-Term Memory (LSTM) z rodziny Rekurencyjnych Sieci Neuronowych (RNN) stanowią zaawansowane rozwiązanie problemów, które dotykają klasyczne RNN. Kluczowymi wyzwaniami, na które odpowiadają, są długotrwałe zależności i zanikający/eksplodujący gradient.
Problem zachowania długotrwałych zależności pojawia się, gdy niezbędne jest zachowanie kontekstu przez sieć przez dłuższy czas, jak na przykład w przypadku języka naturalnego, gdzie znaczenie słowa zależy od jego pozycji względem innych, czasami odległych słów w zdaniu. Proste RNN nie radzą sobie w takich przypadkach ze względu na swoją tendencję do "zapominania" wcześniejszych informacji w miarę przechodzenia do nowych elementów sekwencji.
Problem zanikającego/eksplodującego gradientu pojawia się podczas procesu uczenia. Kiedy sieć RNN jest trenowana na długich sekwencjach danych, gradienty, które są używane do aktualizacji wag sieci, mogą stać się bardzo małe (zanikające gradienty) lub bardzo duże (eksplodujące gradienty). To prowadzi do trudności w uczeniu sieci, ponieważ wag nie można skutecznie dostosować w procesie uczenia.

2 Long Short-Term Memory (LSTM)

2.1 Budowa LSTM

Architektura sieci typu LSTM jest bardziej złożona niż tradycyjne sieci rekurencyjne, takie jak proste rekurencyjne sieci jednokierunkowe. Składa się ona z kilku kluczowych komponentów, które umożliwiają efektywne uczenie na długich sekwencjach danych.

- Komórki pamięci: Głównym elementem sieci LSTM jest komórka pamięci. Komórka pamięci jest rodzajem kontenera, który może przechowywać i aktualizować informacje na przestrzeni wielu kroków czasowych. To właśnie dzięki komórkom pamięci sieć LSTM jest w stanie zachowywać istotne informacje na dłuższe okresy czasu.

- Bramki: Sieci LSTM posiadają trzy rodzaje bramek, które kontrolują przepływ informacji wewnątrz komórki pamięci. Są to:

  - Bramka zapominania f_t (Forget Gate): Odpowiada za decyzję, które informacje należy usunąć z komórki pamięci.

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

    * σ: funkcja sigmoidalna
    * W_f: macierz wag bramki zapomnienia
    * h_{t-1}: stan ukryty z poprzedniego kroku czasowego
    * x_t: wejście w obecnym kroku czasowym
    * b_f: bias bramki zapomnienia

– Bramka wejścia i_t (Input Gate): Określa, które nowe informacje należy dodać do komórki pamięci.

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \text{tgh}(W_C \cdot [h_{t-1}, x_t] + b_C)$$

  * σ: funkcja sigmoidalna
  * W_i: macierz wag bramki wejścia
  * W_C: macierz wag stanu kandydata
  * b_i: bias bramki wejścia
  * $\tilde{C}_t$: stan kandydata
  * b_C: bias stanu kandydata
  * tgh: tangens hiperboliczny

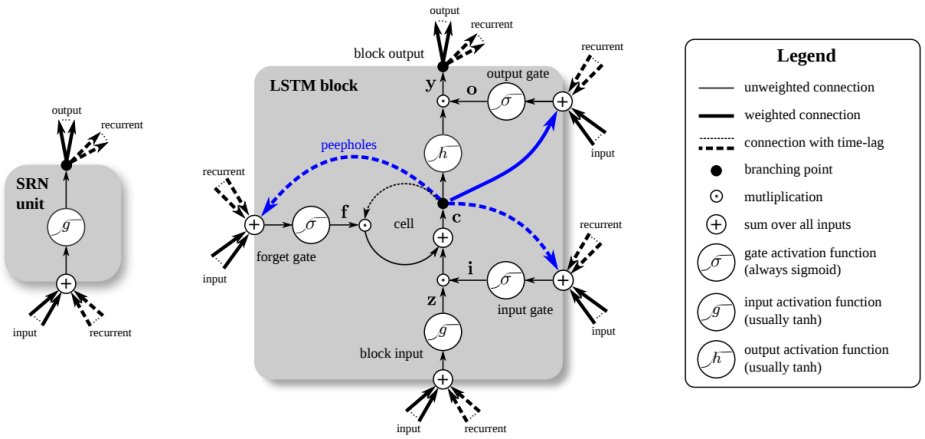
– Bramka wyjścia o_t (Output Gate): Określa, jakie informacje zostaną wydobyte z komórki pamięci i przekazane na wyjście.

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * \text{tgh}(C_t)$$

  * σ: funkcja sigmoidalna
  * W_o: macierz wag bramki wyjścia
  * b_o: bias bramki wyjścia
  * tgh: tangens hiperboliczny
  * h_t: stan ukryty
  * C_t: aktualny stan komórki

- Warstwa ukryta (Hidden State): W każdym kroku czasowym, komórka pamięci generuje ukryty stan, który jest przekazywany do następnego kroku czasowego. Ukryty stan jest rodzajem skondensowanej reprezentacji informacji zawartych w komórce pamięci.



Rysunek 1: Schemat pojedynczej komórki prostej sieci rekurencyjnej (SRN) (po lewej) i pojedynczego bloku sieci rekurencyjnej typu LSTM (po prawej)[7].

2.2 Zasada działania sieci LSTM

1. Inicjalizacja: Na początku działania sieci LSTM komórka pamięci i ukryty stan są zazwyczaj inicjowane na wartości zerowe.

2. Kroki czasowe: W każdym kroku czasowym, sieć LSTM przyjmuje nowe dane wejściowe oraz ukryty stan i komórkę pamięci z poprzedniego kroku.

3. Operacje bramek: Sieć LSTM przeprowadza operacje bramek, aby zdecydować, które informacje przechować w komórce pamięci, które informacje usunąć i jakie informacje wydobyć z komórki pamięci.

4. Aktualizacja stanu: Komórka pamięci jest aktualizowana na podstawie wyników bramek i nowych danych wejściowych.

5. Generowanie wyjścia: Na podstawie stanu komórki pamięci generowany jest ukryty stan, który może być używany do przewidywania, klasyfikacji lub generowania sekwencji.

2.3 Przykład działania

Nauczenie sieci generowania nowych zdań na podstawie tych podanych wcześniej w zbiorze treningowym.

1. Inicjalizacja: Na początku, komórka pamięci i ukryty stan są inicjowane na wartości zerowe.

2. Kroki czasowe: Dla każdego kroku czasowego, sieć LSTM przyjmuje jako wejście poprzednie słowo w zdaniu i ukryty stan z poprzedniego kroku.

3. Operacje bramek: Sieć LSTM używa bramek, aby zdecydować, które słowo dodać do komórki pamięci, które słowo usunąć i jakie słowo wydobyć jako wyjście.

4. Aktualizacja stanu: Komórka pamięci jest aktualizowana na podstawie wyników bramek i nowego słowa wejściowego.

5. Generowanie wyjścia: Na podstawie stanu komórki pamięci generowany jest ukryty stan, który jest używany do przewidywania następnego słowa w zdaniu.

W ten sposób sieć LSTM jest w stanie generować kolejne słowa, tworząc spójne zdania.

3 Stochastic Gradient Descent (SGD)

SGD jest iteracyjną metodą często stosowaną w uczeniu maszynowym, która modyfikuje algorytm gradientu wsadowego poprzez obliczanie gradientu tylko dla jednego przykładu treningowego na każdą iterację. Proces ten obejmuje losowe przetasowanie zestawu danych, wybór szybkości uczenia się α, początkowe wartości parametrów θ i aktualizację wszystkich parametrów z gradientu pojedynczego przykładu treningowego $\nabla J(\theta; x^j; y^j)$, gdzie J to funkcja kosztu, a x^j, y^j to przykład treningowy i odpowiadająca mu etykieta. Wzór aktualizacji parametrów wygląda następująco:

$$\theta_{i+1} = \theta_i - \alpha \times \nabla_\theta J(\theta; x^j; y^j)$$

Następnie proces jest powtarzany aż do osiągnięcia minimum.

3.1 Przykład obliczeniowy

Załóżmy prostą funkcję kosztu $J(\theta) = \theta^2$, startujemy z $\theta_0 = 4$ i $\alpha = 0.1$. W pierwszej iteracji, gradient funkcji kosztu wynosi $\nabla J(\theta) = 2\theta = 8$, więc aktualizacja parametru to $\theta_1 = \theta_0 - 0.1 \times 8 = 3.2$. Proces jest powtarzany, aż do osiągnięcia minimum funkcji kosztu.

3.2 Wady SGD

Mimo swojej efektywności, SGD posiada kilka wad. Przede wszystkim, algorytm może być nieefektywny ze względu na wysoką wariancję aktualizacji, co może prowadzić do wolnej konwergencji, szczególnie w obliczu nieregularnych lub rzadkich danych. Dodatkowo, SGD nie uwzględnia historii gradientów, co może skutkować zbyt gwałtownymi zmianami kierunków i utrudniać osiągnięcie optymalnego minimum. Te ograniczenia skłaniają do poszukiwania ulepszonych wariantów algorytmu, takich jak Stochastic Gradient Descent with Momentum (SGDM) i Adaptive Moment Estimation (ADAM), które są szczegółowo opisane w dalszej części dokumentu.

4 Algorytm SGDM (Stochastic Gradient Descent with Momentum)

SGDM to zaawansowany algorytm optymalizacyjny, który ulepsza prosty stochastyczny spadek gradientu (SGD) poprzez dodanie mechanizmu zwanego „momentum". Ten mechanizm jest inspirowany zasadami fizyki, gdzie obiekt poruszający się pod wpływem siły grawitacji nabiera prędkości w miarę toczącego się w dół zbocza. W analogiczny sposób, momentum pomaga algorytmowi utrzymać stały kierunek podczas poruszania się po powierzchni funkcji błędu, co pozwala na skuteczniejsze pokonywanie niewielkich wzniesień, reprezentujących lokalne minima, oraz zapewnia płynniejsze przemieszczanie się w przestrzeni parametrów.

4.1 Matematyczne tło SGDM

Załóżmy, że istnieje funkcja kosztu $J(\theta)$, którą należy zminimalizować, gdzie θ reprezentuje parametry przyjętego modelu, na przykład wagi w sieci neuronowej. W SGDM wektor prędkości v jest wykorzystywany do akumulowania gradientów z poprzednich kroków w celu wygładzenia aktualizacji wag i wprowadzenie 'pędu' do procesu optymalizacji. Proces aktualizacji parametrów w SGDM składa się z dwóch kroków:

1. Aktualizacja wektora prędkości v:

$$v_{t+1} = \mu \cdot v_t + \eta \cdot \Delta J(\theta_t)$$

   gdzie:

   - v_t to wektor prędkości w kroku t,
   - μ to współczynnik momentum, który stabilizuje aktualizacje (często ustawiany jest na wartości takie jak 0.9),
   - η to szybkość uczenia się (learning rate),
   - $\Delta J(\theta_t)$ to gradient funkcji kosztu J obliczony względem parametrów θ w kroku t

2. Aktualizacja parametrów θ:

$$\theta_{t+1} = \theta_t - v_{t+1}$$

   gdzie:

   - θ_{t+1} to wartości parametrów modelu po aktualizacji.

4.2 Przykład obliczeń

4.2.1 Założenia

Uczona jest sieć neuronowa z jednym parametrem θ, a funkcja kosztu $J(\theta)$ jest prosta i ma formę $J(\theta) = \theta^2$, co daje gradient $\Delta J(\theta) = 2 \cdot \theta$.
Przyjęte wartości potrzebnych parametrów:

- μ=0.9
- η=0.1
- v=0
- θ_0=5, więc $\Delta J(\theta_0) = 2 \cdot \theta_0 = 2 \cdot 5 = 10$

4.2.2 Zastosowanie SGDM:

1. Nowy wektor prędkości:

$$v_1 = \mu \cdot v_0 + \eta \cdot \Delta J(\theta_0) = 0.9 \cdot 0 + 0.1 \cdot 10 = 0 + 1 = 1$$

2. Aktualizacja θ:

$$\theta_1 = \theta_0 - v_1 = 5 - 1 = 4$$

3. Aktualizacja wektora prędkości:

$$v_2 = \mu \cdot v_1 + \eta \cdot \Delta J(\theta_1) = 0.9 \cdot 1 + 0.1 \cdot 8 = 1.7$$

4. Aktualizacja θ:

$$\theta_2 = \theta_1 - v_2 = 4 - 1.7 = 2.3$$

Dzięki wprowadzeniu momentum parametr θ szybciej zbliża się do minimum funkcji kosztu, które w tym przypadku wynosi 0. W kolejnych krokach wartość θ będzie kontynuować zmniejszanie się w kierunku tego minimum, ale z każdym krokiem aktualizacja będzie mniejsza, ponieważ gradient będzie maleć w miarę zbliżania się do minimum. Momentum pozwala jednak na dłuższe utrzymywanie kierunku aktualizacji, przez co proces optymalizacji jest szybszy i bardziej efektywny niż w przypadku zwykłego SGD.

4.3 Zastosowanie SGDM w sieciach LSTM

W przypadku sieci LSTM, które są specjalizowane w przetwarzaniu danych sekwencyjnych i często muszą radzić sobie z długotrwałymi zależnościami w danych, SGDM może być wyjątkowo wartościowy. Momentum pozwala szybciej przesuwać się po powierzchni funkcji kosztu, co jest korzystne, gdy sieć musi uczyć się na podstawie długich sekwencji danych, gdzie proste metody optymalizacyjne mogą nie być efektywne. Przyspiesza to uczenie się sieci LSTM, co jest kluczowe w złożonych zadaniach, takich jak przetwarzanie języka naturalnego czy analiza szeregów czasowych. Dzięki temu SGDM jest często wybierany jako algorytm optymalizacji podczas treningu sieci LSTM, aby zapewnić szybszą i bardziej stabilną konwergencję oraz lepsze ogólne wyniki.

5 Algorytm ADAM (Adaptive Moment Estimation)

Algorytm ADAM to popularna metoda optymalizacji używana w uczeniu głębokich sieci neuronowych, w tym w sieciach LSTM. Jest szczególnie ceniony za swoją efektywność w dużych zbiorach danych i w przypadkach, gdzie wymagana jest duża moc obliczeniowa. Jego główne założenia to :

- Adaptacyjne szybkości uczenia - ADAM dostosowuje szybkość uczenia się dla każdego parametru modelu indywidualnie, na podstawie obliczeń momentów pierwszego (średnia) i drugiego rzędu (niecentrowana wariancja) gradientów. Dzięki temu jest w stanie efektywnie radzić sobie z różnorodnością danych i różnymi topologiami funkcji kosztu.
- Kombinacja Momentum i RMSprop - ADAM łączy w sobie idee stojące za dwoma innymi popularnymi algorytmami optymalizacji - Momentum i RMSprop. Momentum pomaga w przyspieszaniu gradientów w odpowiednich kierunkach, natomiast RMSprop dostosowuje szybkość uczenia się dla każdego parametru.

5.1 Matematyczne tło ADAM

Załóżmy, że istnieje funkcja kosztu $J(\theta)$, którą należy zminimalizować, gdzie θ reprezentuje parametry modelu, na przykład wagi w sieci neuronowej. Algorytm ADAM wprowadza innowacyjne podejście do optymalizacji, wykorzystując obliczenia momentów pierwszego i drugiego rzędu gradientów w celu adaptacyjnego dostosowywania szybkości uczenia się dla każdego parametru modelu. W ADAM, moment pierwszego rzędu (średnia ruchoma gradientów) i moment drugiego rzędu (średnia ruchoma kwadratów gradientów) są wykorzystywane do obliczania aktualizacji wag. Proces aktualizacji parametrów w ADAM jest realizowany poprzez kilka kluczowych kroków, które skupiają się na skutecznym obliczaniu i wykorzystywaniu tych momentów w celu efektywnej i stabilnej konwergencji.

1. Oblicznie momentów:
   - Pierwszy moment (średnia ruchoma gradientów):
   $$m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$$
   - Drugi moment (średnia ruchoma kwadratów gradientów):
   $$v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$$

     gdzie:
     - g_t to gradient funkcji kosztu w czasie,
     - β_1 i β_2 to hiperparametry kontrolujące rozmiar okien dla średnich ruchomych (zwykle blisko 0.9 dla β_1 i 0.999 dla β_2).

2. Korekta odchylenia:
   Wczesne iteracje są skorygowane przez odchylenie, ponieważ m_t i v_t są inicjowane jako wektory zerowe:

   - Skorygowany pierwszy moment:

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

   - Skorygowany drugi moment:

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

3. Aktualizacja parametrów modelu:
   Parametry są aktualizowane z uwzględnieniem skorygowanych momentów:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \cdot \hat{m}_t$$

   gdzie:

   – ϵ to bardzo mala wartość dodana do mianownika w celu uniknięcia dzielenia przez zero np. 10^{-8}

   – η to szybkość uczenia się

5.2 Przykład obliczeń

5.2.1 Założenia

Sieć neuronowa ma jeden parametr θ. Jej zadaniem jest zminimalizowanie pewnej funkcji kosztu $J(\theta)$. Przyjmując, że w dwóch pierwszych iteracjach obliczono gradienty funkcji kosztu jako g_1 i g_2. Przyjęte wartości potrzebnych parametrów:

- m_0=0
- v_0=0
- β_1=0.9
- β_2=0.999
- $\eta = 0.001$
- $\epsilon = 10^{-8}$

5.2.2 Zastosowanie ADAM:

Iteracja 1

1. Obliczanie gradientu funkcji kosztu g_1 (jako przykład przyjęto $g_1 = 0.2$)
2. Aktualizacja momentów:

$$m_1 = \beta_1 \cdot m_0 + (1 - \beta_1) \cdot g_t = 0.9 \cdot 0 + 0.1 \cdot 0.2 = 0.02$$

$$v_1 = \beta_2 \cdot v_0 + (1 - \beta_2) \cdot g_t^2 = 0.999 \cdot 0 + 0.001 \cdot 0.2^2 = 0.00004$$

3. Korekta odchylenia momentów:

$$\hat{m_1} = \frac{m_1}{1 - \beta_1^1} = \frac{0.02}{1 - 0.9} = 0.2$$

$$\hat{v_1} = \frac{v_1}{1 - \beta_2^1} = \frac{0.00004}{1 - 0.999} = 0.04$$

4. Aktualizacja θ:

$$\theta_1 = \theta_0 - \frac{\eta}{\sqrt{\hat{v_1}} + \epsilon} \cdot \hat{m_1} = \theta_0 - \frac{0.001}{\sqrt{0.04} + 10^{-8}} \cdot 0.2$$

Iteracja 2

1. Obliczanie gradient funkcji kosztu nowej wartości θ g_2 (jako przyklad przyjęte $g_2 = 0.1$
2. Aktualizacja momentów:

$$m_2 = \beta_1 \cdot m_1 + (1 - \beta_1) \cdot g_t = 0.9 \cdot 0.02 + 0.1 \cdot 0.2 = 0.038$$

$$v_2 = \beta_2 \cdot v_1 + (1 - \beta_2) \cdot g_t^2 = 0.999 \cdot 0.00004 + 0.001 \cdot 0.2^2 = 0.00007996$$

3. Aktualizacja θ: analogicznie jak w **Iteracji 1**.

W każdej iteracji algorytmu ADAM momenty m_t i v_t są aktualizowane na podstawie nowych gradientów, a następnie są wykorzystywane do dostosowania wartości parametrów θ. Ta metoda pozwala na bardziej stabilne i efektywne znajdowanie minimum funkcji kosztu, co jest szczególnie przydatne w złożonych modelach, takich jak sieci LSTM.

5.3 Zastosowanie ADAM w Sieciach LSTM

Dzięki adaptacyjnym szybkościom uczenia się, ADAM jest szczególnie skuteczny w pracy z sieciami LSTM, które często przetwarzają złożone i wysoce nieliniowe dane sekwencyjne. Pozwala również na szybsze osiąganie dobrej wydajności w porównaniu do tradycyjnych algorytmów optymalizacji, co jest ważne w modelach LSTM, gdzie proces uczenia może być czasochłonny.

6 Bibliografia

1. L. Rahman, N. Mohammed and A. K. Al Azad, "A new LSTM model by introducing biological cell state," 2016 3rd International Conference on Electrical Engineering and Information Communication Technology (ICEEICT), Dhaka, Bangladesh, 2016, pp. 1-6, doi: 10.1109/CEEICT.2016.7873164.

2. Yong Yu, Xiaosheng Si, Changhua Hu, Jianxun Zhang; A Review of Recurrent Neural Networks: LSTM Cells and Network Architectures. Neural Comput 2019; 31 (7): 1235–1270. doi: https://doi.org/10.1162/neco_a_01199.

3. Smagulova, K., James, A.P. A survey on LSTM memristive neural network architectures and applications. Eur. Phys. J. Spec. Top. 228, 2313–2324 (2019). https://doi.org/10.1140/epjst/e2019-900046-x.

4. S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," in Neural Computation, vol. 9, no. 8, pp. 1735-1780, 15 Nov. 1997, doi: 10.1162/neco.1997.9.8.1735.

5. A. Graves, A. -r. Mohamed and G. Hinton, "Speech recognition with deep recurrent neural networks," 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, Vancouver, BC, Canada, 2013, pp. 6645-6649, doi: 10.1109/ICASSP.2013.6638947.

6. K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink and J. Schmidhuber, "LSTM: A Search Space Odyssey," in IEEE Transactions on Neural Networks and Learning Systems, vol. 28, no. 10, pp. 2222-2232, Oct. 2017, doi: 10.1109/TNNLS.2016.2582924.

7. Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. "Deep Learning". MIT Press. http://www.deeplearningbook.org.

8. Yang, N., Tang, C., & Tu, Y. (2023). Stochastic Gradient Descent Introduces an Effective Landscape-Dependent Regularization Favoring Flat Solutions. Physical Review Letters, 130(237101)

9. Cornell University. (n.d.). Stochastic Gradient Descent. W Computational Optimization Open Textbook - Optimization Wiki

10. Liu, Yanli, Yuan Gao, and Wotao Yin. "An improved analysis of stochastic gradient descent with momentum." Advances in Neural Information Processing Systems 33 (2020): 18261-18271.

11. J. Fu, B. Wang, H. Zhang, Z. Zhang, W. Chen and N. Zheng "When and Why Momentum Accelerates SGD: An Empirical Study", arXiv:2306.09000

12. X. Li, M. Liu and F. Orabona, "On the Last Iterate Convergence of Momentum Methods", arXiv:2102.07002

13. Attrapadung, Nuttapong, et al. "Adam in private: Secure and fast training of deep neural networks with adaptive moment estimation." arXiv preprint arXiv:2106.02203 (2021).

14. S. Bock, J. Goppold and M. Weiß, "An improvement of the convergence proof of the ADAM-Optimizer", arXiv:1804.10587