ZPR PWr
Zintegrowany Program Rozwoju Politechniki Wrocławskiej

SIECI NEURONOWE – laboratorium

Ćwiczenie 3 – Sieć wielowarstwowa uczona metodą SGD Techniki poprawy szybkości uczenia (realizacja ćwiczenia na laboratorium6, laboratorium7)

Liczba punktów: 30

Ćwiczenie ma na celu praktyczne poznanie technik poprawiających szybkość uczenia sieci. W ramach ćwiczenia należy samodzielnie zaimplementować podane metody optymalizacji współczynnika uczenia i 2 metody inicjalizacji wag.

Fundusze Europejskie Wiedza Edukacja Rozwój
Politechnika Wrocławska
Unia Europejska Europejski Fundusz Społeczny

Laboratorium 6 : Optymalizacja współczynnika uczenia

Cel: Celem tego laboratorium jest praktyczne zapoznanie się metodami optymalizacji współczynnika uczenia

Wprowadzenie teoretyczne

W tym ćwiczeniu przyjmiemy, że mamy zdefiniowane zadanie do rozwiązania, przygotowany zbiór uczący oraz przyjętą funkcję kosztu. Jak pamiętamy znalezienie właściwych parametrów (wag i biasów) sieci polega na minimalizacji funkcji kosztu. Minimalizację kosztu przeprowadzamy stosując iteracyjnie metodę GD (gradient descent), ponieważ funkcja kosztu zależy od zbyt wielu zmiennych, aby proces poszukiwania minimum mógł być możliwy do obliczenia w sposób analityczny. W metodzie GD na początku inicjujemy wartości parametrów a następnie a następnie zmieniamy wartości parametrów iteracyjnie redukując wartość funkcji kosztu *C*. W każdej iteracji wartości parametrów zmieniane są przeciwnie do gradientu funkcji kosztu, co formalnie możemy zapisać jako:

$$W = W - \alpha \frac{\partial C}{\partial W}$$

W tym wzorze:

- W to macierz parametrów
- $\frac{\partial C}{\partial W}$ gradient wskazujący kierunek zmian W aby zmniejszyć koszt C
- α jest współczynnikiem uczenia decydującym o tym jak mocno chcemy korygować wagi w czasie jednej iteracji

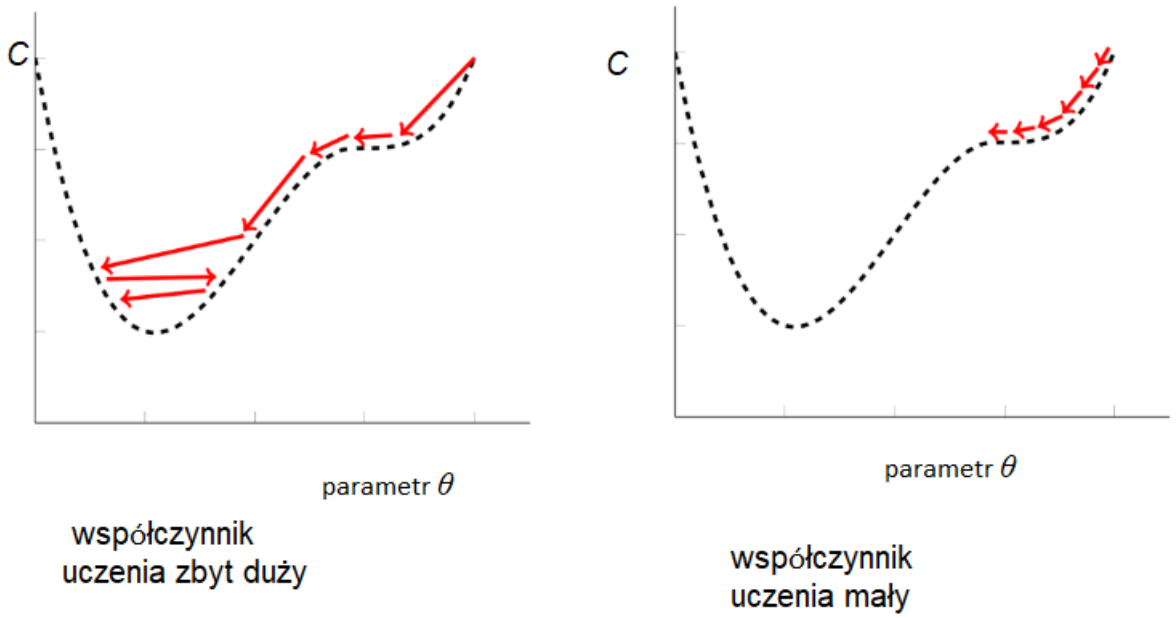
Metoda GD do obliczania gradientu bierze cały zbiór uczący, więc jego obliczenie jest bardzo wolne. Dlatego minimalizujemy średni koszt obliczony na podstawie paczki wzorców. Jej rozmiar oznaczymy przez *m*. Jest to jeden z kluczowych hiperparametrów modelu, którego wpływ badaliśmy w poprzednim ćwiczeniu. Innym hiperparametrem mającym wpływ mającym istotny wpływ na znalezienie wag ma współczynnik uczenia.

Warto również zauważyć, że im zbiór uczący jest większy, tym bliżej będą znalezione parametry modelu do parametrów użytych do wygenerowania danych (rzeczywistych). Nawet jeśli użyjemy najlepszych hiperparametrów, wytrenowany model nie musi w 100% odpowiadać prawdziwym wartościom, ponieważ zbiór danych uczących jest pewną próbą przybliżającą prawdziwy rozkład danych. Hiperparametry mają różny wpływ na zbieżność uczenia. W tym laboratorium zajmiemy się zajmiemy się sposobami ustawiania współczynnika uczenia i innymi technikami mającymi wpływ na wielkość i kierunek zmian wag. Precyzyjniej rzecz ujmując będziemy zajmować się metodami optymalizacji współczynnika uczenia oraz poznamy od strony praktycznej technikę momentum. Na następnych zajęciach poznamy praktycznie różne metody inicjalizacji wag i ich wpływ na uczenie sieci.

Współczynnik uczenia wpływa na zbieżność procesu uczenia. Również równoważy wpływ kształtu krzywizn funkcji kosztu. Wiemy już z dotychczasowych ćwiczeń, że współczynnik uczenia ma wpływ na zbieżność procesu uczenia:

- jeśli jest zbyt duży, proces uczenia może nie być zbieżny,
- jeśli za mały, sieć uczy się bardzo wolno.

To jak działa współczynnik uczenia zależy od kształtu funkcji kosztu. Metoda GD wykonuje liniową aproksymację funkcji kosztu w danym punkcie. Następnie przesuwa się w dół wzdłuż aproksymowanej wartości funkcji kosztu. Jeśli funkcja kosztu jest bardzo wykrzywiona - wówczas im większy jest współczynnik uczenia, tym mamy większą szansę, że algorytm „przestrzeli"i znajdziemy się na przeciwnej stronie krzywej kosztu zamiast schodzić w dół. Mały współczynnik uczenia redukuje ten problem, ale bardzo spowalnia uczenie Rys.1.



Rys.1. Wpływ wartości współczynnika uczenia na proces optymalizacji funkcji kosztu (na podstawie: https://www.math.purdue.edu/~nwinovic/deep_learning_optimization.html)

Innym wyzwaniem w znalezieniu optymalnej wartości szczególnie dla mało wypukłych funkcji kosztu jest unikanie wpadnięcia w liczne minima lokalne. Jeszcze większy problem pojawia się w punktach siodłowych (w punktach gdzie funkcja w jednym wymiarze idzie w górę a w innym w dół. Te punkty są zazwyczaj otoczone przez płaskowyż, gdzie funkcja kosztu jest stała. Powoduje to, że metoda SGD nie może wyskoczyć z tego miejsca, ponieważ gradient jest bliski 0 w każdym wymiarze.

Bardzo popularnym podejściem jest ustawienie początkowego współczynnika uczenia na dość dużą wartość np. 0.1 i jej zmniejszanie w trakcie uczenia. Decyzja jak często zmniejszać i o ile nie jest wcale trywialna. Dlatego często stosuje się (w sieciach głębokich to norma) adaptacyjnie zmieniane współczynniki uczenia, w których aktualna wartość zależy najczęściej od aktualnej iteracji i od wartości funkcji kosztu.

Po tym wprowadzeniu możemy przejść do omówienia poszczególnych metod, które wspomagają minimalizację funkcji kosztu.

Momentum SGD ma duże problemy z poruszaniem się po krzywej kosztu, w miejscach gdzie występują wąwozy. W takim przypadku SGD oscyluje, nie mogąc wyskoczyć z wąwozu. Takie możliwości daje technika momentum. Polega ona na tym, że dodaje się część poprzedniej aktualizacji wag do bieżącej zmiany.

$$v_t = \gamma v_{t-1} + \alpha \frac{\partial C(\theta)}{\partial \theta}$$

$$\theta = \theta - v_t$$

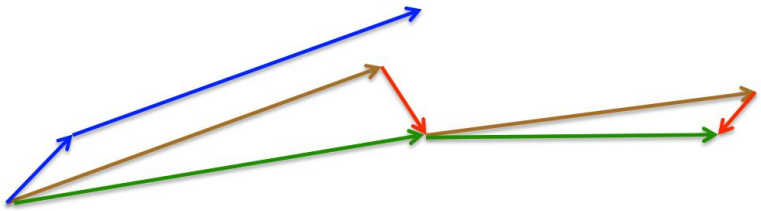
W tych wzorach v_t jest przyrostem parametru θ w aktualnym kroku t, α jest współczynnikiem uczenia, γv_{t-1} to składnik momentum, czyli część zmiany wag z poprzedniego kroku. γ to współczynnik momentum. Jego wartość jest wysoka w porównaniu do współczynnika uczenia i zazwyczaj mieści się od 0.7 do 0.9. Jak widać jest to niewielka zmiana w implementacji. Mometum zazwyczaj używa więcej pamięci dla paczki wzorców niż SGD.

Momentum Nesterova (NAG- Nesterov accelerated gradient). W tym przypadku człon momentum uwzględnia przewidywanie następnej pozycji. Dokładniej, obliczając $\theta - \gamma v_{t-1}$ mamy aproksymację następnej pozycji parametrów, czyli obliczamy gradient dla przewidywanych na przyszłość wartości parametrów

$$v_t = \gamma v_{t-1} + \alpha \frac{\partial C(\theta - \gamma v_{t-1})}{\partial \theta}$$

$$\theta = \theta - v_t$$

Wartości współczynnika momentum ustawiane są podobnie jak dla zwykłego momentum.



Rys. 2. Wizualizacja zmiany wag przy użyciu momentum i momentum Nesterowa (źródło: Hinton wykład 6c)

Rys. 2 pokazuje różnicę ze sposobu aktualizacji wag w obu przypadkach. Dla momentum najpierw obliczany jest gradient (mały wektor w kolorze niebieskim a następnie wykonywany jest skok w kierunku uaktualnionego gradientu – długi niebieski wektor)
Dla NAG najpierw wykonywany jest skok do poprzedniego zakumulowanego gradientu (wektor brązowy), mierzony jest gradient I obliczana jest korekcja (czerwony wektor), co powoduje, że w efekcie (wektor oznaczony na zielono) znajdujemy się w zupełnie innym miejscu.

Adagrad. Ten algorytm zmienia wartość współczynnika uczenia w zależności od parametrów. Mniejsza wartość współczynnika uczenia jest dla parametrów odpowiadających za często

pojawiające się cechy, natomiast wysoka wartość współczynnika uczenia dla parametrów skojarzonych z rzadkimi cechami. Jest to odpowiedni algorytm optymalizacji współczynnika uczenia dla rzadkich danych. Adagrad ma różne współczynniki uczenia dla każdego parametru θ_i w danym kroku *t*. Oznaczmy przez g_t gradient w korku *t*; a przez $g_{t,i}$ gradient w kroku t dla parametru θ_i

$$g_{t,i} = \frac{\partial C(\theta_{t,i})}{\partial \theta}$$

Wówczas uaktualnienie parametru θ_i w każdym kroku t jest obliczane następująco:

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\alpha}{\sqrt{G_{t,ii} + \varepsilon}} g_{t,i}$$

G_t jest macierzą diagonalną, w której każdy element na diagonali jest sumą kwadratów poprzednich gradientów θ_iaż do kroku *t*. Element zapobiega dzieleniu przez 0 i zwykle jest równy 1e-8.

Możemy ten wzór zapisać w postaci wektorowej

$$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{G_t + \varepsilon}} \boldsymbol{g}_t$$

gdzie mamy iloczyn macierzy G_t i wektora gradientu $\boldsymbol{g}_t$.

Adagrad eliminuje manualne ustawianie współczynnika uczenia. Większość implementacji przyjmuje $\alpha = 0.01$. Jej słabością jest sumowanie gradientów podniesionych do kwadratu w mianowniku może powodować tak małe wartości współczynnika uczenia, ze uczenie nie będzie dalej możliwe (wszystkie wartości kwadratów gradientów są dodatnie).

Adadelta Jest rozszerzeniem metody Adagrad. Zamiast sumować wszystkiekwadraty gradientów Adadelta ogranicza sumowanie do rozmiaru w. Jednak w celu poprawy efektywności, w tym algorytmie suma gradientów jest rekursywnie zdefiniowana jako zmniejszajaca się średnia poprzednich gradientów.Bieżaca średnia $E[g^2]_t$ w chwili *t* zależy od poprzedniej średniej i bieżącego gradientu:

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1 - \gamma) g_t^2$$

Współczynnik γ przyjmuje wartość ok. 0.9. W klasycznej metodzie SGD parametry θ wyznaczane są jako:

$$\theta_{t+1} = \theta_t + \Delta\theta_t$$

$$\Delta\theta_t = -\alpha g_{t,i}$$

Wektor uaktualnień dla Adagrad był równy:

$$\Delta\theta_t = -\frac{\alpha}{\sqrt{G_t + \varepsilon}}\boldsymbol{g}_t$$

W mianowniku mamy pierwiastek z błędu średniokwadratowego (RMS root mean squared error), wiec tę równość możemy zapisać jako:

$$\Delta\theta_t = -\frac{\alpha}{\sqrt{RMS[g]_t + \varepsilon}}\boldsymbol{g}_t$$

Autorzy tej metody zauważyli, ze ten przyrost nie ma takich samych (hipotetetycznych) jednostek jak parametry uaktualniane. Aby to zapewnić wprowadzili eksponencjalnie malejącą średnią podniesionych do kwadratu przyrostów parametrów.

$$E[\Delta\theta^2]_t = \gamma E[\Delta\theta^2]_{t-1} + (1 - \gamma)\Delta\theta_t^2$$

RMS jest wówczas równy:

$$RMS[\Delta\theta^2]_t = \sqrt{E[\Delta\theta^2]_t + \varepsilon}$$

Ponieważ $RMS[\Delta\theta^2]_t$ jest nieznany , więc aproksymujemy go przez RMS z poprzedniego kroku czyli $RMS[\Delta\theta^2]_{t-1}$. Ostatecznie

$$\Delta\theta_t = -\frac{RMS[\Delta\theta]_{t-1}}{RMS[g]_t}\boldsymbol{g}_t$$

$$\theta_{t+1} = \theta_t + \Delta\theta_t$$

Adam to również metoda optymalizacji współczynnika uczenia, w której jest on aktualizowany dla każdego parametru oddzielnie. Metoda dąży do płaskich minimów. Używa się poprzednich gradientów m_t i przeszłych kwadratów gradientów v_t

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)g_t$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)g_t^2$$

Następnie używa się skorygowanych form podanych niżej:

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

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

Są one dalej używane do korekcji parametrów:

$$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{\hat{v}_t} + \varepsilon}\hat{m}_t$$

Metoda jest często wykorzystywana jako domyślny optymalizator w wielu środowiskach. Spośród metod optymalizacji współczynnika uczenia, przedstawianych w tym wprowadzeniu, potrzebuje najwięcej pamięci dla danego rozmiaru paczki. Warto w tym miejscu wspomnieć o innych optymalizatorach, gdyż badania nad nowymi metodami wciąż trwają, np. ADAMAX, NADAM czy AMSgrad.

Realizacja ćwiczenia

Realizacja ćwiczenia wymaga implementacji metody momentum, momentum Nesterowa oraz Adagrad , Adadelta i Adam a następnie przeprowadzenia badań umożliwiających porównanie tych metod pod względem szybkości uczenia i skuteczności sieci.

W ćwiczeniu należy wykorzystać sieć zbudowaną w ćwiczeniu 2 do rozpoznawania cyfr ze zbioru MNIST. Wagi inicjować w ten sam sposób jak w ćwiczeniu 2. Przyjąć najlepszą architekturę sieci, uzyskaną na podstawie badań przeprowadzonych w ćwiczeniu 2.

Zbadać wpływ:

- momentum,
- momentum Nesterova,
- optymalizacji współczynnika uczenia metodą Adagrad,
- optymalizacji współczynnika uczenia metodą Adadelta,
- optymalizacji współczynnika uczenia metodą Adam,

na szybkość uczenia. Badania przeprowadzić dla 2 różnych funkcji aktywacji: Funkcji sigmoidalnej i dla ReLU.

Proszę porównać uśrednione z 10 badań przebiegi funkcji straty dla każdego optymalizatora oraz porównać z wynikami uzyskanymi w ćwiczeniu 2.

Sposób oceny:

20% - Implementacja metod optymalizacji współczynnika uczenia na zajęciach

20% - Przebadanie wpływu momentum, Momentum Nesterowa i AdaGrad

20% - Przebadanie Adadelta i Adam

Użyteczne linki

https://ruder.io/optimizing-gradient-descent/

wizualizacje: http://louistiao.me/notes/visualizing-and-animating-optimization-algorithms-with-matplotlib/

https://www.deeplearning-academy.com/p/ai-wiki-optimization-algorithms

https://www.deeplearning.ai/ai-notes/optimization/

Laboratorium 7 : Inicjalizacja wag sieci

Cel : Praktyczne poznanie metod inicjalizacji wag

Wprowadzenie teoretyczne

Inicjalizacja wag (sposób wyznaczenia wag początkowych) ma istotne znaczenie dla całego procesu uczenia. Wyznacza nam możliwości szybkiego wyuczenia sieci lub wprost przeciwnie nie jesteśmy w stanie wyuczyć sieci lub proces uczenia kończy się w płytkim optimum lokalnym. Poniżej przedstawionych jest kilka metod inicjalizacji wag.

Zacznijmy może od tego, że **nie powinniśmy inicjować wag od ich ustawienia na 0**. Jeśli wszystkie wagi będą ustawione na 0, wszystkie będą miały to samo pobudzenie równe 0, niezależnie od płynących sygnałów. Dalej po przekształceniu przez funkcję aktywacji neurony mają te same wartości wyjść. Dlatego będą miały te same gradienty a co za tym idzie te same uaktualnienia parametrów w warstwie i dalej będą miały te same wartości. Ponieważ ustawiliśmy wagi na 0 będą miały te same wartości wag. Dlatego wartości wag nigdy nie powinny być inicjalizowane od tej samej wartości. W sieci nie możemy mieć takiej symetrii wag.

Inicjowanie wag losowe. Możemy inicjować wagi używając standardowego rozkład normalnego lub z rozkładu jednostajnego. W tym przypadku mamy dwa możliwe scenariusze.

- Wagi są zbyt małe. W trakcie rzutowania wstecz błędów zgodnie z metodą propagacji wstecznej od warstwy wyjściowej do tej, w której aktualnie uaktualniamy wagi, kolejne gradienty mnoży się. Im dalej od wyjścia, tym wolniej adaptują się wagi. Może to powodować, że model nie osiągnie optimum w czasie, w jaki przeznaczyliśmy na uczenie.
- Wagi są zbyt duże. W trakcie przejścia sygnału w przód mnożymy kolejne sygnały i ostatecznie może się zdarzyć, że mogą one spowodować przekroczenie zakresu liczb w komputerze. Wynikiem jest pojawienie się NaN (Not a Number). W tym przypadku, nawet jeśli nie pojawi się błąd mamy do czynienia z zanikającym gradientem.

Warto dodać, że oprócz zanikającego gradientu problemem w uczeniu sieci może być również eksplodujący gradient, który jest wynikiem akumulowania gradientów dużych błędów, co może powodować duże przyrosty wag i niestabilność procesu uczenia. W ekstremalnym przypadku, może kończyć się przekroczeniem zakresu liczb (NaN). Dlaczego tak się dzieje?

Załóżmy, że do inicjalizacji wag użyliśmy rozkładu normalnego o średniej w 0 i jednostkowym odchyleniem standardowym. Prześledźmy jaki to ma wpływ. Dla uproszczenia obliczmy całkowite pobudzenie z jakiego neuronu w warstwie pierwszej:

$$z = x_1 w_1 + x_2 w_2 + x_3 w_3 + ..$$

Wariancja (kwadrat odchylenia standardowego) każdego elementu w tej sumie jest możemy zapisać jako wykorzystując fakt, że są to zmienne niezależne:

$$\mathrm{V}ar(x_i w_i) = [E(x_i)]2\mathrm{V}ar(w_i) + [E(w_i)]2\mathrm{V}ar(x_i) + \mathrm{V}ar(x_i)\mathrm{V}ar(w_i)$$

Jeśli przyjmiemy, że wejścia są wyskalowane ze średnią w 0 i jednostkową wariancją i inicjowaliśmy wagi ze średnią w zerze i jednostkową wariancją, otrzymujemy:

$$\mathrm{V}ar(x_i w_i) = 0x1 + 0x1 + 1x1 = 1$$

Oznacza to, że taki iloczyn ma wariancję równą 1. Co możemy powiedzieć o całkowitym pobudzeniu? Jeśli założymy, ze każdy iloczyn jest statystycznie niezależny otrzymujemy, że wariancja z jest równa

$$Var\ z = \sum_{i=0}^{n} Var(x_i w_i) = nx1 = n$$

gdzie n jest liczbą wejść. Przeanalizujmy ten wynik dla problemu MNIST. Odchylenie standardowe przy 784 wejściach będzie wynosiło 28. Zakładając użycie funkcji sigmoidalnej (tanh) spowoduje to, ze większość neuronów w pierwszej warstwie będzie pracować w zakresie nasycenia funkcji sigmoidalnej (większość wartości >>|2|).

Podsumowując, chociaż losowa inicjalizacja jest lepsza od ustawienia wszystkich początkowych wag na 0, jednak nie jest pozbawiona wad, dlatego podjęto próbę opracowania innych sposobów inicjalizacji wag. Musimy pamiętać, że odpowiednia inicjalizacja wag jest krytyczna dla wyuczenia się właściwego odwzorowania wejścia na wyjście sieci. Ponieważ przestrzeń poszukiwań zawierająca wiele wag jest ogromna, na dodatek zawiera ona wiele lokalnych minimów, w które może wpaść nasz proces poszukiwań, efektywne określenie wag początkowych zapewnia, że przestrzeń ta będzie odpowiednio eksplorowana.

Zaawansowane metody inicjalizacji wag redukujące problem zanikającego lub eksplodującego gradientu to: metoda Xaviera (czasem nazywana metodą Glorota) i metoda He. Ich idea działania jest podobna, ale mają też różnice, które zostaną przedstawione dalej. Obie metody normalizują początkowe wartości wag, w taki sposób aby nie pojawił się problem zanikającego, czy wybuchającego gradientu. Oznacza to, ze duże wartości inicjowanych wag będą zmniejszane, a bardzo małe będą zwiększane. Praktycznie, dla modeli głębokich stosuje się jedną tych dwóch metod inicjalizacji wag – He lub Xaviera.

Inicjalizacja wag Xaviera. Metoda powszechnie używana w modelach głębokich. Bierze pod uwagę opisane wyżej problemy i odnosi wariancję inicjowanych wag do liczby rozpatrywanych zmiennych.

Sposób inicjalizacji wag jest dostosowany do liczby wag. Autorom przyświecała idea, że jeśli można utrzymać wariancję stałą od warstwy do warstwy w obu kierunkach, tzn. przesłania w przód i rzutowania błędów wstecz- sieć będzie się uczyć optymalnie. Można to sobie wytłumaczyć w ten sposób, że jeśli wariancja wzrasta lub zmniejsza się jeśli przechodzimy przez warstwy wagi najprawdopodobniej znajda się w przedziale, gdzie funkcja aktywacji będzie w nasyceniu zarówno dla liczb dodatnich jak i ujemnych.

Jak ta idea jest zaimplementowana w praktyce? Jaka powinna być wariancja, aby jak najlepiej zainicjować wagi?
Zauważmy, że siec uczy się efektywnie, jeśli operuje na liniowych regionach funkcji tanh albo sigmoidalnej. Na tej liniowej części działanie funkcji możemy przybliżyć przez funkcję liniową, tzn.

$$y = w_1 x_1 + w_2 x_2 + w_3 x_3 + \cdots$$

Używając takiej liniowej aproksymacji funkcji sigmoidalnej albo tanh, możemy wykorzystać iloczyn niezależnych zmiennych i sumę nieskorelowanych niezależnych zmiennych, a mianowicie:

$$Var(y) = n_{in} Var(w_i) Var(x_i)$$

gdzie n_{in} jest liczbą wejść do neuronu. Jeśli chcemy, aby wariancja wejścia $Var(x_i)$ była taka sama jak wariancja wyjścia $Var(y)$, nasze równanie zredukuje się do:

$$Var(w_i) = \frac{1}{n_{in}}$$

Jest to wstępny wynik dla dobrej inicjalizacji wariancji wag w sieci. Ale takie podejscie umożliwia tylko utrzymania stałej wariancji w czasie fazy przesłania w przód. Jak utrzymać stałą wariancję w fazie przesłania wstecz? Żeby to osiągnąć musi być spełnione:

$$Var(w_i) = \frac{1}{n_{out}}$$

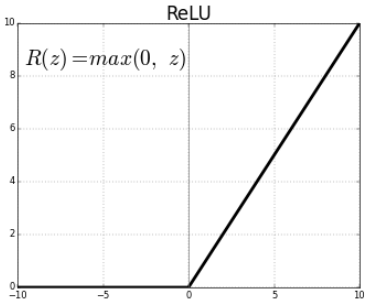
Teraz mamy dwa sposoby liczenia wariancji, jedna zależna od liczby wejść, druga zależna od liczby wyjść. Autorzy oryginalnej pracy [Xavier] uwzględnili średnia arytmetyczną obu:

$$n_{avg} = \frac{n_{in} + n_{out}}{2}$$

Czyli wariancja $Var(w_i) = \frac{2}{n_{in} + n_{out}}$

Jest to ostateczny wynik inicjalizacji wag wg Xaviera, odpowiedni dla funkcji sigmoidalnej lub tanh. Niestety badania pokazały, ze nie działa ona dobrze dla funkcji ReLU.

Inicjalizacja wag He. Przypomnijmy funkcję ReLU – dla wszystkich wartości mniejszych od 0 , wyjście z funkcji aktywacji jest równe 0.



Rys. 3. Funkcja aktywacji ReLu (źródło: blog Kanchan Sarkar's blog)

Dla wartości większych od 0 ReLU zwraca wejście . Innymi słowy, dla połowy zakresu wartości (tam gdzie funkcja jest liniowa) , wszystkie założenia omówione dla inicjalizacji Xaviera są spełnione. Dla drugiej części, tam gdzie wartości są mniejsze od 0 wartość funkcji jest równa zero. Jeśli przyjmiemy, że wartości wejściowe do ReLU są w przybliżeniu wycentrowane koło0, połowa wariancji, będzie taka jak w inicjalizacji Xaviera a połowa będzie równa 0. Odpowiadałoby to zmniejszeniu o połowę liczby wejść, a wiec odnosząc się do obliczeń Xaviera i zmniejszając liczbę wejść otrzymujemy:

$$Var(y) = \frac{n_{in}}{2} Var(w_i) Var(x_i)$$

Ponownie, jeśli chcemy aby wariancja wejść $Var(x_i)$ była równa wariancji wyjścia $Var(y)$, formuła redukuje się do:

$$Var(w_i) = \frac{2}{n_{in}}$$

He zbadał, że zaproponowana przez niego inicjalizacja wag jest lepsza dla funkcji aktywacji ReLU. Trzeba jednak pamiętać, że badania robione były dla bardzo głębokiej sieci.

Realizacja ćwiczenia

W tym ćwiczeniu do sieci neuronowej z ćwiczenia 2 zbudowanej do rozpoznawania zbioru MNIST, należy doimplementować dwie możliwości inicjalizacji wag w sieci, przedstawione w części teoretycznej, a mianowicie metodę Xaviera i metodę He oraz przeprowadzić badania eksperymentalne nad skutecznością procesu uczenia sieci przy każdej z nich. Podczas badań należy wybrać najlepszy optymalizator współczynnika uczenia.

W ćwiczeniu należy wykorzystać sieć zbudowaną w ćwiczeniu 2 do rozpoznawania cyfr ze zbioru MNIST. Przyjąć najlepszą architekturę sieci, uzyskaną na podstawie badań

przeprowadzonych w ćwiczeniu 2. Zastosować optymalizator współczynnika uczenia, który okazał się najlepszy w eksperymentach przeprowadzonych na laboratorium 6.

Zbadać wpływ sposobu inicjalizacji wag dla:

- Metodą Xaviera
- Metodą He

na szybkość uczenia. Badania przeprowadzić dla różnych funkcji aktywacji (funkcja sigmoidalna i ReLU). Proszę porównać uśrednione z 10 badań przebiegi funkcji straty dla każdego sposobu inicjalizacji oraz porównać z wynikami uzyskanymi w ćwiczeniu 2.

Sposób oceny

20% - Przebadanie 2 różnych metod inicjalizacji wag

20% - implementacja metod inicjalizacji wag na zajęciach

Przesłanie raportu do prowadzącego do dnia ……

Literatura i użyteczne linki

Xavier Glorot, Yoshua Bengio: Understanding the difficulty of training deep feedforward neural networks,

Kaiming He Xiangyu Zhang Shaoqing Ren Jian Sun: Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification

https://www.deeplearning.ai/ai-notes/initialization/

https://medium.com/usf-msds/deep-learning-best-practices-1-weight-initialization-14e5c0295b94

https://adventuresinmachinelearning.com/weight-initialization-tutorial-tensorflow/