ZPR PWr
Zintegrowany Program Rozwoju Politechniki Wrocławskiej

SIECI NEURONOWE – laboratorium

Ćwiczenie 2 – sieć wielowarstwowa uczona metodą propagacji wstecznej

Celem ćwiczenia jest zapoznanie się z siecią wielowarstwową, uczeniem sieci za pomocą algorytmu propagacji wstecznej w wersji klasycznej oraz wpływem parametrów odgrywających istotną rolę w uczeniu sieci z propagacją wsteczną.
Na realizację opisanego niżej ćwiczenia przeznaczone są trzy laboratoria, czyli 3 tygodnie czasu. Pierwsze zajęcia przeznaczone są na implementację architektury sieci, drugie implementację metody uczenia, trzecie przeprowadzenie wstępnych badań.

Uwaga: Raport z ćwiczenia nie powinien zawierać części teoretycznej a jedynie dokumentować przeprowadzone eksperymenty. Każdy eksperyment powinien być powtórzony 10- krotnie

UWAGA: w tym ćwiczeniu można używać pakietu do mnożenia macierzy (numpy) jednak zbudowanie architektury sieci jak również metody uczenia musi być napisane samodzielnie.

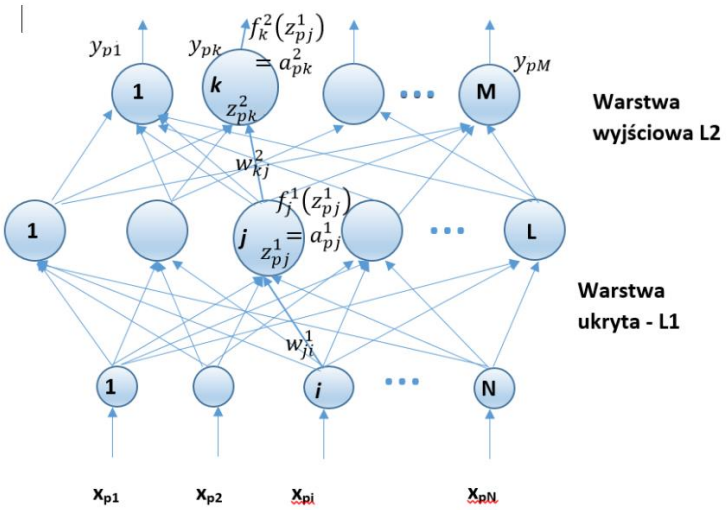
Fundusze Europejskie Wiedza Edukacja Rozwój
Politechnika Wrocławska
Unia Europejska Europejski Fundusz Społeczny

Laboratorium 3: Implementacja architektury płytkiej sieci jednokierunkowej

Cel: Należy zaimplementować architekturę wielowarstwowego perceptronu do rozpoznawania cyfr ze zbioru MNIST i fazę przesłania wzorca w przód.

Wprowadzenie teoretyczne

Wielowarstwowy perceptron (MLP – MultiLayer Perceptron) jest uogólnieniem sieci, którymi zajmowaliśmy się w poprzednim ćwiczeniu. BPN jest siecią warstwową z przesyłaniem sygnału do przodu, w pełni połączoną między sąsiednimi warstwami. A więc nie istnieją połączenia ze sprzężeniami zwrotnymi i połączeniami, które omijając jakąś warstwę przechodzą do innej. Dozwolonych jest więcej niż jedna warstwa ukryta i wówczas taka sieć jest przykładem sieci głębokiej. W ćwiczeniu będziemy budować sieć z jedną warstwą ukrytą jednak w części teoretycznej będziemy odwoływać się do uogólnionego przypadku z większą liczbą warstw ukrytych.



Rys. 1. Wielowarstwowy perceptron z jedną warstwą ukrytą – wersja omawiana na wykładzie .

Odnosząc się do Rys. 1. przypomnijmy, że wektor **x** oznacza wzorce podawane do sieci neuronowej (dolny indeks p – oznacza p-ty wzorzec) **z** oznacza całkowite pobudzenie neuronu, f zaś oznacza funkcję aktywacji, której wynikiem działania jest aktywacja **a** neuronu. Zawsze górny indeks w oznaczeniach będzie się odnosił do numeru warstwy. Sieć uczona jest w trybie nadzorowanym, a to oznacza, że
mamy dany zbiór uczący D w postaci podanej poniżej:

$$D = \{\langle \mathbf{x}_1, \mathbf{y}_1 \rangle, \langle \mathbf{x}_2, \mathbf{y}_2 \rangle \dots \langle \mathbf{x}_P, \mathbf{y}_P \rangle\}$$

Dana jest sieć neuronowa z parametrami θ, która realizuje funkcję $f_\theta(\mathbf{x})$.

Naszym zadaniem jest wyuczenie parametrów θ (wag i biasów), takich, że $\forall i \in [1, N]: f_\theta(\mathbf{x}_i) = \mathbf{y}_i$
W tym ćwiczeniu nasze wektory **x** to obrazy cyfr ze zbioru MNIST a odpowiadające im wektory **y** na wyjściu, to etykiety cyfr.

Obliczenie $f_\theta(\mathbf{x})$ odbywa się w fazie przesłania p-tego wektora wejsciowego w przód aż do obliczenia wyjścia (faza forward), jak na Rys 1., i polega na obliczeniu dla każdego neuronu w danej warstwie najpierw jego całkowitego pobudzenia z, czyli dla warstwy pierwszej mielibyśmy:

$$z_{pj}^1 = \sum_{i=1}^{N} w_{ji}^1 x_{pi}$$

Natomiast wyjście z tej warstwy byłoby równe:

$$a_{pj}^1 = f_j^1(z_{pj}^1)$$

W warstwie drugiej całkowite pobudzenie liczy się w identyczny sposób tylko tym razem wejściem jest wyjście z poprzedniej warstwy

$$z_{pk}^2 = \sum_{j=0}^{L} w_{kj}^2 \, a_{pj}^1$$

A wyjściem z tej warstwy jest wyjście z sieci y

$$y_{pk} = f_k^2(z_{pk}^2)$$

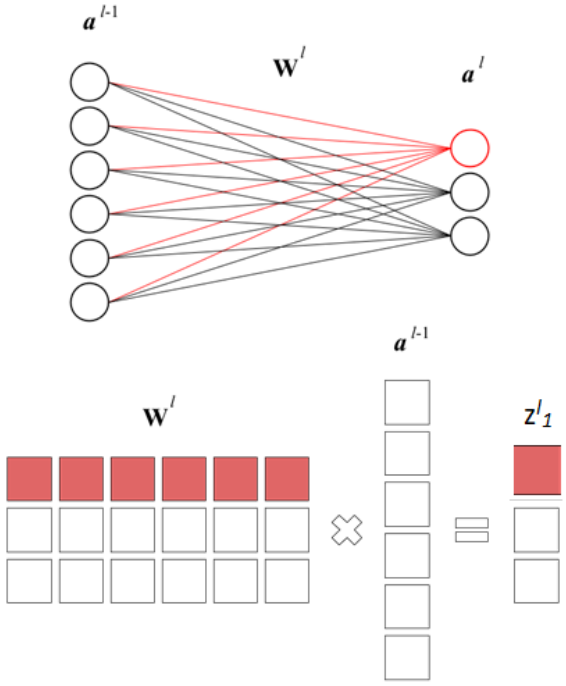
Jednak znacznie lepiej (i szybciej jeśli chodzi o obliczenia) jest używać notacji wektorowej.

Odnosząc się do dowolnej warstwy, aktywacje neuronów w warstwie *l* pamiętane są w wektorze kolumnowym $\mathbf{a}^l$ natomiast wagi na połączeniach między warstwą l oraz l+1 są pamiętane w macierzy $\mathbf{W}^l$ a biasy w wektorze kolumnowym $\mathbf{b}^l$
Dla fazy przesłania sygnału do przodu dla warstwy l, w której funkcja aktywacji oznaczona jest jako f, aktywację możemy wyrazić jako:

$$\mathbf{a}^l = f(\mathbf{W}^l \mathbf{a}^{l-1} + \mathbf{b}^l)$$

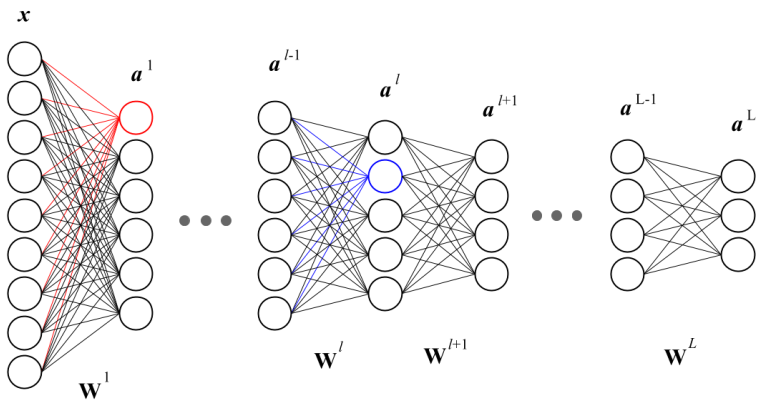
Powyższe mnożenie macierzy możemy zwizualizować jak na Rys.2., gdzie wprowadzony jest nowy wektor $\mathbf{z}^l$ oznaczający całkowite pobudzenie neuronu a następnie do tego wektora stosujemy funkcję aktywacji f, wykonując przekształcenie przez tę funkcję każdej składowej wektora.

$$\mathbf{a}^l = f(\mathbf{z}^l)$$



Rys. 2. Wizualizacja obliczania całkowitego pobudzenia jako operacji macierzowej; z^l_1 jest całkowtym pobudzeniem dla pierwszego neuronu w warstwie *l* (źródło: https://medium.com/@erikhallstrm/backpropagation-from-the-beginning-77356edf427d)

Cała sieć, o dowolnej liczbie warstw pokazana jest na Rys.3. Wejściem jest wektor **x**, a wyjściem z warstwy *l* jest wektor $\mathbf{a}^l$. Połączenia prowadzące do specyficznych neuronów wyróżnione są kolorami w dwóch warstwach.



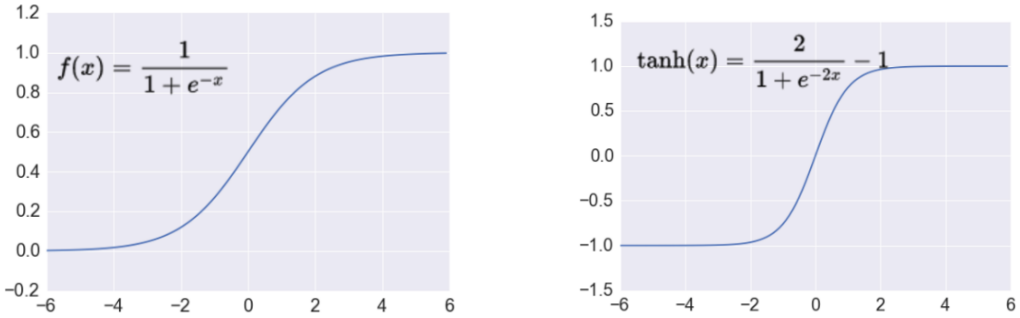
Rys.3. Głęboka sieć wielowarstwowa (źródło: https://medium.com/@erikhallstrm/backpropagation-from-the-beginning-77356edf427d)

Aktywacja n-tego elementu w warstwie wyjściowej może być opisana w postaci matematycznej formuły:

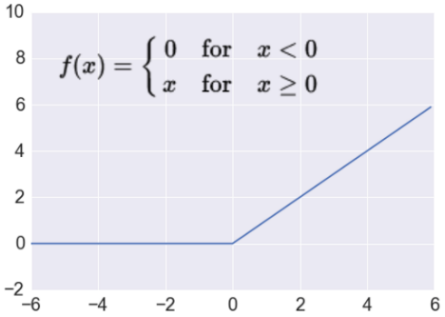
$$a_n^L = \left[f\left(\sum_m w_{nm}^L \left[\ldots \left[f\left(\sum_i w_{ji}^1 x_i + b_j^1 \right) + b_k^2 \right] \ldots \right] + b_n^L \right) \right]$$

Sieć neuronowa realizuje funkcję matematyczną.
Jak pamiętamy z wykładu funkcja aktywacji musi być funkcją różniczkowalną. Mamy do wyboru wiele funkcji, z których najbardziej popularne pokazane są na rysunkach poniżej. Jedną z pierwszych była funkcja sigmoidalna Rys.4. Funkcja sigmoidalna ma dwie asymptoty i w przedziałach gdzie jest prawie płaska gradient jest prawie równy 0. Podobnie wygląda kształt funkcji tangensa hiperbolicznego Rys.4b).

a) b)



Rys. 4. Klasyczne funkcje aktywacji a) sigmoidalna, b) tangensa hiperbolicznego (źródło: *http://adilmoujahid.com/images/activation.png*)



Rys.5 Funkcja aktywacji ReLU (źródło: *http://adilmoujahid.com/images/activation.png)*

Problem zanikającego gradientu powodował, że uczenie głębszych sieci było problematyczne. Dlatego w sieciach głębokich stosuje się najczęściej funkcję ReLU pokazaną na Rys. 5.

W warstwie wyjściowej, dla zadania klasyfikacji, które będziemy rozwiązywać w tym ćwiczeniu, informacja kodowana jest na zasadzie „1 z n". Oznacza to, że oczekujemy aby tylko 1 neuron miał wartość 1, ten który odpowiada rozpoznanej klasie, pozostałe neurony powinny mieć wartość równą 0. Aby umożliwić probabilistyczną interpretację wyniku otrzymanego na wyjściu, obecnie najczęściej stosuje się funkcję *softmax*.
Ogólnie możemy powiedzieć, że funkcja softmax bierze N wymiarowy wektor wartości rzeczywistych i produkuje inny N-wymiarowy wektor wartości rzeczywistych z przedziału (0,1), w taki sposób, że składowe wektora sumują się do 1.

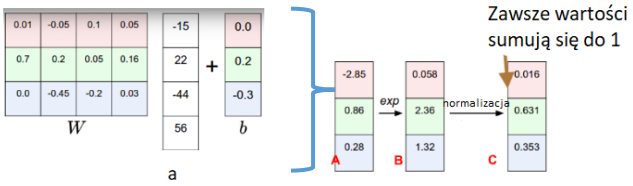
$$S(\mathbf{z}): \begin{bmatrix} z_1 \\ z_2 \\ \dots \\ z_N \end{bmatrix} \rightarrow \begin{bmatrix} S_1 \\ S_2 \\ \dots \\ S_N \end{bmatrix}$$

Formuła określająca pojedynczy element wektora jest następująca:

$$S_j = \frac{e^{z_j}}{\sum_{k=1}^{N} e^{z_k}}$$

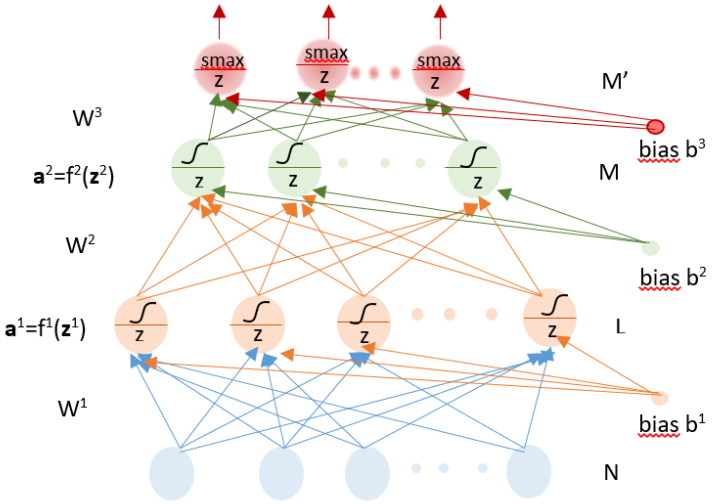
Łatwo zobaczyć, że wartości są zawsze dodatnie (z powodu exponent) a w mianowniku mamy sumowanie po liczbach dodatnich, dlatego zakres jest ograniczony do przedziału (0,1). Na przykład, 3 elementowy wektor [1.0, 2.0, 3.0] jest transformowany do wektora [0,09, 0.24, 0.67] a więc względne relacje jeśli chodzi o wielkość składowych są zachowane a składowe sumują się do 1. Jeśli rozważany wektor zmienimy [1.0, 2.0, 5.0] otrzymamy wektor [0.02, 0.05, 0.93]. Zauważmy, że ostatni element wektora jest znacznie większy niż pierwsze dwa.

Intuicyjnie softmax jest miękką wersją funkcji maksimum, ale zamiast wybierać jedynie wartość maksymalną softmax wyraża składowe wektora w stosunku do całości.



Rys.6. Wizualizacja operacji wykonywanych w warstwie softmax (na podstawie slajdu z kursu CS 231n Stanford University)

Realizacja ćwiczenia. Na zajęciach należy zaimplementować prostą jednokierunkową sieć neuronową, do rozpoznawania cyfr ze zbioru MNIST oraz fazę przesłania wzorca do przodu (forward), tak aby otrzymać wyjście. Architekturę sieci przedstawia Rys. 6.



Rys.7. Architektura sieci, którą należy zaimplementować w ćwiczeniu 2.

W sieci wyróżniamy warstwę wejściową, dwie warstwy ukryte oraz warstwę wyjściową, która realizuje funkcję softmax.
Aplikacja powinna być napisana na tyle ogólnie, aby była możliwość:

b) użycie od 2-4 warstw,
c) użycie różnych funkcji aktywacji w warstwach ukrytych (sigmoidalna, tanh, ReLU),
d) użycie warstwy softmax (na Rys. 6. smax) w warstwie wyjściowej,
e) zmiany sposobu inicjalizowania wag (w tym ćwiczeniu przyjmujemy, że wagi będą inicjalizowane z rozkładu normalnego ze zmiennym odchyleniem standardowym),
f) Zmiany liczby neuronów w warstwach ukrytych,
g) przerwania uczenia i ponownego rozpoczęcia nauki od poprzednich wartości wag.

Faza przesłania wzorca w przód dla sieci pokazanej na Rys. 7przedstawia się następująco.
Dla warstwy pierwszej:

Obliczenie całkowitego pobudzenia $\mathbf{z}^1 = \mathbf{W}^1\mathbf{x} \quad + \mathbf{b}^1$

Obliczenie aktywacji $\mathbf{a}^1 = f(\mathbf{z}^1)$

Dla warstwy drugiej:

Obliczenie całkowitego pobudzenia $\mathbf{z}^2 = \mathbf{W}^2\mathbf{a}^1 + \mathbf{b}^2$

Obliczenie aktywacji $\mathbf{a}^2 = f(\mathbf{z}^2)$

Dla warstwy trzeciej

Obliczenie całkowitego pobudzenia $\mathbf{z}^3 = \mathbf{W}^3\mathbf{a}^2 + \mathbf{b}^3$

Oblicz wyjście z sieci , poszczególne składowe wyjścia to:

$$\hat{y}_j = \frac{e^{z_j}}{\sum_{k=1}^{M'} e^{z_k}}$$

Za realizację zadania na zajęciach, można otrzymać 20% całkowitej liczby punktów.

Laboratorium 4: Implementacja metody uczenia dla płytkiej sieci jednokierunkowej

Cel: rozszerzenie aplikacji zbudowanej na poprzednim laboratorium o możliwość uczenia sieci

Wprowadzenie teoretyczne

Jak wspomniano przy wprowadzeniu do poprzedniego laboratorium, jednokierunkowe sieci wielowarstwowe uczone są metodą nadzorowaną. Uczenie (trenowanie sieci) sprowadza się do znalezienia parametrów sieci θ, czyli wag i biasów, takich że $\forall i \in [1, N]: \mathrm{f}_\theta\,(\mathbf{x}_i) = \mathbf{y}_i$

Możemy to osiągnąć minimalizując błąd (koszt, funkcję straty) na zbiorze uczącym D, co możemy zapisać jako:

$$\min_\theta \mathrm{L}(f_\theta, D) = \min_\theta \frac{1}{P}\sum_{i=1}^{P} L(f_\theta(\mathbf{x}_i), \mathbf{y}_i)$$

W tym wzorze L jest funkcją straty. Dla regresji używamy błędu średniokwadratowego:

$$L(f_\theta(\mathbf{x}_i), \mathbf{y}_i) = \frac{1}{2}\sum_{j=1}^{M}\left(f_{j,\theta}(\mathbf{x}) - \mathrm{y}_j\right)^2$$

Dla klasyfikacji do M klas funkcją straty jest ujemny logarytm szans (negative log likelihood)

$$L(f_\theta(\mathbf{x}_i), \mathbf{y}_i) = \sum_{j=1}^{M} -log\left(f_{j,\theta}(\mathbf{x})\right)\ \mathrm{y}_j$$

Najprostszą metodą uczenia jest metoda GD (Gradient Descent), kiedy uaktualnianie wag odbywa się po każdym wzorcu.

Algorytm Gradient Descent (GD)

Zainicjuj losowo θ^0

Wykonaj

$$\theta^{t+1} = \theta^{t+1} - \gamma\frac{\partial L(f_\theta, D)}{\partial\theta}$$

dopóki

$$\left(\min_\theta \mathrm{L}(f_{\theta^{t+1}}, V) - \min_\theta \mathrm{L}(f_{\theta^t}, V)\right)^2 > \epsilon$$

gdzie V jest zbiorem walidacyjnym, γ jest współczynnikiem uczenia.
W tym przypadku obliczenie gradientu jest kosztowne obliczeniowo, jeśli zbiór uczący jest duży. Problemem jest także odpowiedni dobór współczynnika uczenia, ale to będzie tematem ćwiczenia 3., w którym użyjemy bardziej zaawansowanych technik do optymalizacji współczynnika uczenia, takich jak Rprop/Rmsprop, Adagrad czy Adam.

Powiedzieliśmy, że GD jest kosztowne obliczeniowo, dlatego próbuje się obliczać gradient na części danych i taki algorytm nazywany jest SGD od Stochastic Gradient Descent.

Algorytm Stochastic Gradient Descent SGD

Zainicjuj losowo θ^0

Wykonaj

Próbkuj przykłady$(\boldsymbol{x}', \mathbf{y}')$ ze zbioru D

$$\theta^{t+1} = \theta^{t+1} - \gamma^t \frac{\partial L(f_\theta(\mathbf{x}'), \mathbf{y}')}{\partial \theta}$$

dopóki

$$\left(\min_\theta \mathrm{L}(f_{\theta^{t+1}}, V) - \min_\theta \mathrm{L}(f_{\theta^t}, V)\right)^2 > \epsilon$$

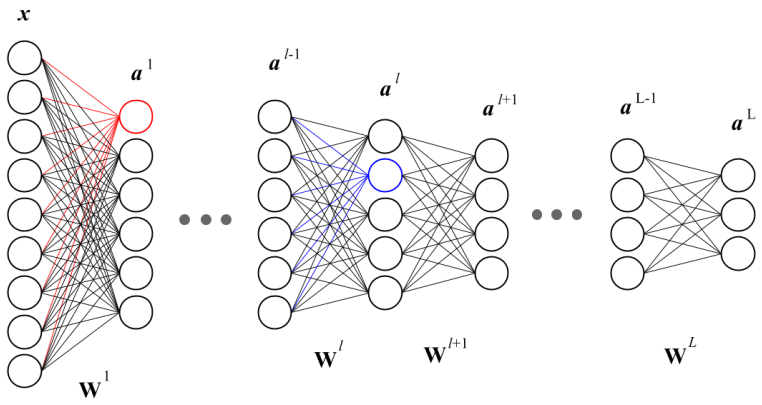
gdzie $\sum_{t=1} \gamma^t \to \infty$ oraz $\sum_{t=1} (\gamma^t)^2 < \infty$

SGD przyspiesza optymalizację dla dużych zbiorów, ale wprowadza szum do uaktualniania wag. W praktyce używa się tzw. minibatch - paczki wzorców. Paczka, w zależności od zbioru danych, liczy od kilku do kilkuset wzorców.

W obliczaniu gradientu stosujemy regułę łańcuchową. Przypomnijmy, że jeśli mamy do czynienia z funkcją złożoną *f*, która zależy of funkcji *g*, to pochodną obliczamy następująco:
-1

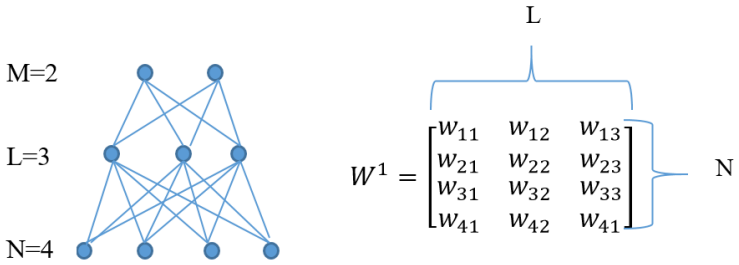
$$\frac{\partial f(g(x))}{\partial x} = \frac{\partial f(g(x))}{\partial g(x)} \frac{g(x)}{\partial x}$$

Przypomnijmy ogólny schemat architektury sieci o liczbie warstw L. W tym przypadku macierz wag W^l odpowiada wagom na połączeniach między neuronami z warstwy l-1 a neuronami warstwy l.



Rys. 8. Głęb0ka jednokierunkowa sieć neuronowa

W macierzy tej wiersze odpowiadają neuronom w warstwie wcześniejszej l a kolumny neuronom w warstwie l+1. Pokazuje to przykład na Rys. 9.



Rys. 9. Przykład zapisu wag w macierzy dla prostej architektury sieci

Jak pokazano na wykładzie będzie polegało na obliczeniu błędów w każdej warstwie począwszy od ostatniej do pierwszej, a po ich obliczeniu na adaptacji wag proporcjonalnie do błędu. Przyjmując L warstw w sieci i funkcję kosztu C (funkcja, którą będziemy minimalizować) błąd δ^L w ostatnie-tej warstwie sieci możemy zdefiniować począwszy od warstwy końcowej L jako:

$$\delta^L = \frac{\partial C}{\partial a^L} \odot f'(z^L)$$

gdzie $\mathbf{a}^L$ to wynik działania funkcji aktywacji w tej warstwie czyli $\mathbf{a}^L = f(\mathbf{z}^L)$ i ponieważ jest to ostatnia warstwa w sieci to jej wyjście $\hat{\mathbf{y}} = \mathbf{a}^L$. C Jest funkcją kosztu a znak $\odot$ odpowiada iloczynowi Hadamarda (mnożeniu odpowiadających sobie składowych)
Ogólnie dla l-tej warstwy błąd liczymy jako:
$\boldsymbol{\delta}^l = (W^{l+1})^T \boldsymbol{\delta}^{l+1} \odot f'(\boldsymbol{z}^l)$

Możemy więc powiedzieć, że dla dowolnej warstwy l ten błąd jest uzależniony od błędu warstwy wyższej, który pomnożony jest przez transponowaną macierz wag a następnie wykonywana jest operacja iloczynu Hadamarda przez pochodna funkcji aktywacji w tej warstwie. Obliczenie tych błędów stanowi podstawę do obliczania gradientu błędu po wagach w danej warstwie i dlatego należy tę operacje dobrze rozumieć. Załóżmy, że warstwa l ma K neuronów a warstwa l+1 ma M neuronów, wynik mnożenia pokazany jest poniżej. Zachęcam do zrobienia ćwiczenia na kartce papieru.

$$(\mathrm{W}^{l+1})^T \delta^{l+1} = \begin{bmatrix} w_{11}^{l+1} & \dots & w_{m1}^{l+1} & \dots & w_{M1}^{l+1} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ w_{1k}^{l+1} & \dots & w_{mk}^{l+1} & \dots & w_{Mk}^{l+1} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ w_{1K}^{l+1} & \dots & w_{mK}^{l+1} & \dots & w_{MK}^{l+1} \end{bmatrix} \begin{bmatrix} \delta_1^{l+1} \\ \vdots \\ \delta_m^{l+1} \\ \vdots \\ \delta_M^{l+1} \end{bmatrix} = \begin{bmatrix} \sum_{m=1}^{M} \delta_m^{l+1} w_{m1}^{i+1} \\ \vdots \\ \sum_{m=1}^{M} \delta_m^{l+1} w_{mk}^{i+1} \\ \vdots \\ \sum_{m=1}^{M} \delta_m^{l+1} w_{mK}^{l-1} \end{bmatrix}$$

Schemat algorytmu uczenia, który poznaliśmy na wykładzie dla pojedynczego wzorca (metoda SGD dla pojedynczego wzorca) wygląda następująco:

1. **Propagacja sygnału do przodu** – obliczamy pobudzenia każdego neuronu przez wykonywane iteracyjnie mnożenia macierzy wag i wektorów aktywacji z warstwy poprzedzającej a następnie docierających a następnie wykonujemy operacje na odpowiadających sobie składowych pobudzenia i funkcji aktywacji w każdej warstwie. Zapamiętujemy wynik.

2. **Obliczenie błędu w ostatniej warstwie L –** ta operacja wymaga obliczenia gradientu kosztu funkcji. Wyrażenie zależy od bieżącego wzorca (x i y) a także od wybranej funkcji kosztu (straty, błędu).
3. **Wykonaj propagację sygnału wstecz** – oblicz błędy dla neuronów w każdej warstwie. W tych obliczeniach także będą potrzebne pobudzenia neuronów (dlatego trzeba je było zapamiętać w trakcie propagacji sygnałów w przód). Tutaj również wykonywane są iteracyjne obliczenia macierzowo wektorowe.
4. **Oblicz pochodne funkcji kosztu** ze względu na wagi. W tym przypadku konieczne są również aktywacje każdego neuronu. W wyniku otrzymamy macierz o tych samych wymiarach jak macierz wag.
5. **Oblicz pochodne funkcji kosztu ze względu na bias.** Wynikiem będzie wektor kolumna.
6. Zaktualizuj wagi zgodnie z regułą Generalized Delta RUle (GDR)

Wyjaśnienia powyższe odnoszą się do zmiany wag sieci po przetworzeniu jednego wzorca. Jak użyć paczki wzorców? Teraz nasze wejście będzie reprezentowane w postaci macierzy, w której kolumny będą odpowiadać wzorcom, natomiast wiersze, to odpowiednie składowe wzorca. Przez wykonanie propagacji w przód. Pobudzenia i aktywacje będą również pamiętane w macierzach, gdzie indeks kolumny odpowiada indeksowi wzorca a indeks wiersza indeksowi neuronu. W macierzach będą tez pamiętane błędy dla różnych warstw I wzorce . Pochodne cząstkowe funkcji kosztu w odniesieniu do wag będą w trójwymiarowym tensorze o wymiarach [nr próbki, nr neuronu, nr wagi]

Przyjmiemy, że w tym ćwiczeniu funkcją straty jest ujemny logarytm szansy (ang. negative log likelyhood), czyli:

$$C(f_\theta(\mathbf{x}_i), \mathbf{y}_i) = \sum_{j=1}^{M} -log\hat{y}_j y_j$$

W tym wzorze $\hat{y_j}$ jest rzeczywistym wyjściem z sieci natomiast y_j jest wartością wynikającą z wzorca. M jest liczbą klas (równą liczbie neuronów wyjściowych, bo kodowanie mamy 1 z n)
Wykorzystując metodę propagacji wstecznej najpierw musimy policzyć stratę C w warstwie wyjściowej a następnie rzutować błędy wstecz.

1.W pierwszej kolejności obliczamy gradient funkcji straty dla warstwy trzeciej.

$$\frac{\partial C(f(\boldsymbol{z}^3), \mathbf{y})}{\partial W^3} = \frac{\partial C\big((f(\boldsymbol{z}^3), \mathbf{y})\big)}{\partial\, \boldsymbol{z}^3} \frac{\partial\, \boldsymbol{z}^3}{\partial W^3}$$

Jak można zauważyć na rysunku przedstawiającym architekturę sieci ostatnia warstwa to warstwa softmax. Gradient z funkcji kosztu w odniesieniu do funkcji softmax rozważymy rozpatrując poszczególne składowe z_j

$$\frac{\partial C}{\partial z_i^3} = \sum_{j=1}^{M} \frac{-log\hat{y}_j y_j}{\partial z_i^3} = -\sum_{j=1}^{M} y_j \frac{log\hat{y}_j}{\partial z_i^3} = -\sum_{j=1}^{M} y_j \frac{1}{\hat{y}_j} \frac{\partial \hat{y}_j}{\partial z_i^3}$$

$$= -\frac{y_i}{\hat{y}_i} \frac{\partial \hat{y}_i}{\partial z_i^3} - \sum_{i \neq j}^{M} \frac{y_j}{\hat{y}_j} \frac{\partial \hat{y}_j}{\partial z_i^3} == -\frac{y_i}{\hat{y}_i} \hat{y}_i(1 - \hat{y}_i) - \sum_{i \neq j}^{M} \frac{y_j}{\hat{y}_j}(-\hat{y}_j \hat{y}_i)$$

Pochodna funkcii softmax

$$= -y_i + y_i \hat{y}_i + \sum_{i \neq j}^{M} y_i \hat{y}_i = -y_i + \sum_{j=1}^{M} y_j \hat{y}_i = -\,y_i + \hat{y}_i \sum_{j=1}^{M} y_j$$

Ta suma jest równa 1

$$= \hat{y}_i - y_i$$

W formie wektorowej ten wynik możemy zapisać jako:

$$\frac{\partial C\big((f(\boldsymbol{z}^3),\mathbf{y})\big)}{\partial\, \boldsymbol{z}^3} = -(\boldsymbol{y} - f(\boldsymbol{z}^3)) = (\hat{\mathbf{y}} - \mathbf{y}) \quad \leftarrow \boldsymbol{\delta}^3$$

gdzie **y** jest kodowane jako „1 z n".
Przypomnijmy, że $\mathbf{z}^3 = \mathbf{W}^3\mathbf{a}^2 + \mathbf{b}^3$, więc gradient $\boldsymbol{z}^3$ względem ∂W^3wyraża się następująco:

$$\frac{\partial\, \boldsymbol{z}^3}{\partial W^3} = \big(\mathbf{a^2}\big)^T$$

Po połączeniu obu wyrażeń otrzymujemy

$$\frac{\partial C(f(\boldsymbol{z}^3),\mathbf{y})}{\partial W^3} = -(\boldsymbol{y} - f(\boldsymbol{z}^3))\big(\mathbf{a^2}\big)^T = \boldsymbol{\delta}^3\big(\mathbf{a^2}\big)^T$$

2.Teraz przechodzimy do warstwy drugiej

$$\frac{\partial C(f(\boldsymbol{z}^3),\mathbf{y})}{\partial \mathbf{a}^2} = \frac{\partial C\big((f(\boldsymbol{z}^3),\mathbf{y})\big)}{\partial\, \boldsymbol{z}^3}\,\frac{\partial\, \boldsymbol{z}^3}{\partial \mathbf{a}^2}$$

Biorąc pod uwagę, że $\mathbf{z}^3 = \mathbf{W}^3\mathbf{a}^2 + \mathbf{b}^3$ ostatni element będzie macierzą wag$\mathbf{W}^3$
A więc równanie możemy zapisać jako:

$$\frac{\partial C(f(\boldsymbol{z}^3),\mathbf{y})}{\partial \mathbf{a}^2} = (\mathbf{W}^3)^T\frac{\partial C\big((f(\boldsymbol{z}^3),\mathbf{y})\big)}{\partial\, \boldsymbol{z}^3} = (\mathbf{W}^3)^T\boldsymbol{\delta}^3$$

Gradient po wagach w warstwie drugiej:

$$\frac{\partial C(f(\boldsymbol{z}^3),\mathbf{y})}{\partial W^2} = \frac{\partial C(f(\boldsymbol{z}^3),\mathbf{y})}{\partial \mathbf{a}^2}\frac{\partial\, \boldsymbol{a}^2}{\partial \boldsymbol{z}^2}\frac{\partial\, \boldsymbol{z}^2}{\partial W^2}$$

Pochodna funkcji aktywacji
Trzeba wstawić odpowiednie wartości w zależności od zaimplementowanej funkcji aktywacji

Wiemy, że $\mathbf{z}^2 = \mathbf{W}^2\mathbf{a}^1 + \mathbf{b}^2$, więc ostatni element gradientu to będzie $(\mathbf{a}^1)^T$ Ostatecznie mamy:

$$\frac{\partial C(f(\boldsymbol{z}^3),\mathbf{y})}{\partial W^2} = (\mathbf{W}^3)^T\boldsymbol{\delta}^3 \odot f'^2(\mathbf{a}^1)^T$$

Oznaczając jako błąd $\boldsymbol{\delta}^2$ w tej warstwie

$$\boldsymbol{\delta}^2 = (\mathbf{W}^3)^T\boldsymbol{\delta}^3 \odot f'^2$$

Otrzymujemy gradient błędu po wagach w warstwie drugiej

$$\frac{\partial C(f(\boldsymbol{z}^3), \mathbf{y})}{\partial W^2} = \boldsymbol{\delta}^2 (\mathbf{a}^1)^T$$

3.Dalej powtarzamy ten sam schemat dla warstwy pierwszej

Oznaczmy błąd dla warstwy pierwszej jako:

$$\boldsymbol{\delta}^1 = (\mathbf{W}^2)^T \boldsymbol{\delta}^2 \odot f'^1$$

Wówczas gradient błędu po wagach w warstwie pierwszej wynosi:

$$\frac{\partial C(f(\boldsymbol{z}^3), \mathbf{y})}{\partial W^1} = \boldsymbol{\delta}^1 (\mathbf{x})^T$$

4.Po policzeniu wszystkich błędów korygujemy wagi

Dla warstwy trzeciej:

$$\mathrm{W}^3 = \mathrm{W}^3 - \alpha \boldsymbol{\delta}^3 \left(\mathbf{a}^2\right)^T$$

Dla warstwy drugiej:

$$\mathrm{W}^2 = \mathrm{W}^2 - \alpha \boldsymbol{\delta}^2 \left(\mathbf{a}^1\right)^T$$

Dla warstwy pierwszej:

$$\mathrm{W}^1 = \mathrm{W}^1 - \alpha \boldsymbol{\delta}^1 (\mathbf{x})^T$$

5. Korekcja biasów

Dla warstwy trzeciej:

$$\mathbf{b}^3 = \mathbf{b}^3 - \alpha \boldsymbol{\delta}^3$$

Dla warstwy drugiej:

$$\mathbf{b}^2 = \mathbf{b}^2 - \alpha \boldsymbol{\delta}^2$$

Dla warstwy pierwszej:

$$\mathbf{b}^1 = \mathbf{b}^1 - \alpha \boldsymbol{\delta}^1$$

Przedstawiony algorytm odpowiada korekcji wag po każdym wzorcu, czyli SGD (stochastic gradient descent) z wielkością paczki m=1.
Rozmiar paczki może przyjmowac różne wartości. Typowe małe paczki zawierają 32, 64, 128, 256, 512a duże do kilku tysięcyWybór właściwej liczby wzorców w paczce ma wpływ na zbieżność funkcji kosztu i wartości parametrów oraz zdolności modelu do generalizacji. Istnieją badania jaka powinna być wielkość paczki, ale opinie nie są zgodne. W praktyce jest to jeszcze jeden hiperparametr, którego wartość określa się przez przeszukiwanie.

Dobre praktyki są następujące:

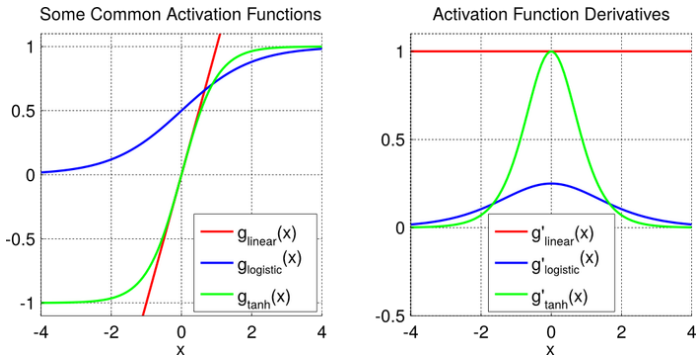
Paczka określa częstość uaktualnień parametrów (wag i biasów). Im mniejsza paczka, tym częściej uaktualniamy parametry

Im większa paczka, tym bardziej dokładny jest gradient funkcji kosztu w odniesieniu do parametrów, tzn. kierunek uaktualniania najprawdopodobniej zmierza w dół w kierunku lokalnego minimum optymalizowanej funkcji.

Większy rozmiar paczki wzorców, może poprawić możliwości zrównoleglenia, co jest szczególnie istotne dla uczenia głębokich modeli z GPU.

Wybór wielkości paczki powinien być równowagą pomiędzy dostępną mocą obliczeniową a zadaniem, które chcemy osiągnąć.

Jak widać, aby określić błąd w każdej warstwie musimy obliczyć pochodną funkcji aktywacji . Oznacza to, że używane funkcje muszą być różniczkowalne, To ograniczenie spełniają wszystkie podane w poprzednim laboratorium funkcje: sigmoidalna, tangensa hiperbolicznego, liniowa i ReLu, ale musimy znać ich pochodne.



Rys.10. Popularne funkcje aktywacji i ich pochodne (źródło: https://theclevermachine.wordpress.com/2014/09/08/derivation-derivatives-for-common-neural-network-activation-functions/)

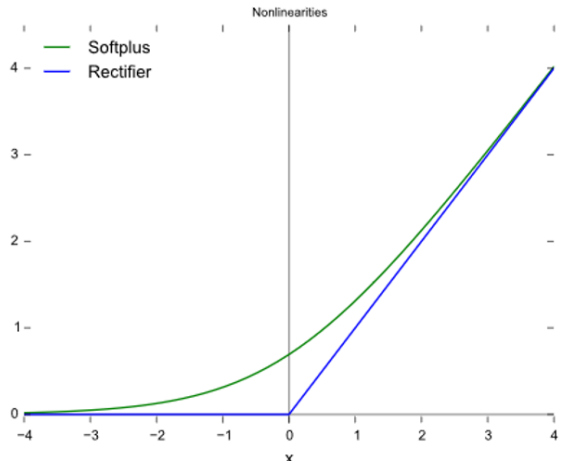
Funkcja liniowa: $f_{liniowa}(z) = z$ jej pochodna jest równa $\mathrm{f}'_{liniowa} = 1$
Funkcja sigmoidalna (ang. logistic function) $f_{sigmoidalna}(z) = \frac{1}{1+e^{-z}}$,
pochodna $\mathrm{f}'_{sigmoidalna} = f_{sigmoidalna}(1 - f_{sigmoidalna})$. Innymi słowy pochodna jest obliczana jako iloczyn wartości aktywacji neuronu pomnożonej przez wartość 1 minus wartość aktywacji.

Funkcja tangensa hiperbolicznego $f_{tanh}(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$

Pochodna funkcji tangensa hiperbolicznego $f'_{tanh}(z) = 1 - (f_{\tanh}\quad(z))^2$

Funkcja ReLU $f_{ReLU}(z) = \max(0, z)$. Funkcja ta nie jest różniczkowalna w 0. Istnieją dwa sposoby radzenia sobie z takim przypadkiem. Pierwszy to arbitralnie przypisać wartość pochodnej w 0. np. 0.

Alternatywą jest użycie pewnego przybliżenia tej funkcji ReLU na przykład funkcji *softplus*



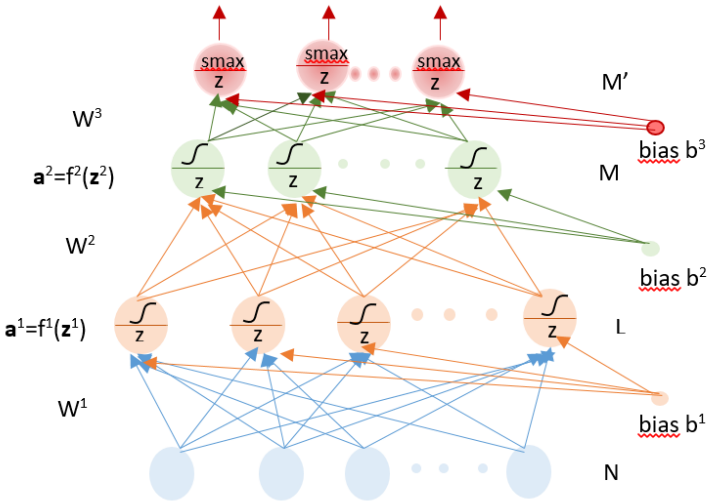
Rys. 11. Funkcja Relu (Rectifier) i Softmax (źródło: https://medium.com/@himanshuxd/activation-functions-sigmoid-relu-leaky-relu-and-softmax-basics-for-neural-networks-and-deep-8d9c70eed91e)

Funkcja Softpus $f_{softplus} = \ln(1 + e^z)$

Jej pochodna $f'_{softplus}(z) = \frac{1}{1+e^{-z}}$ jest funkcja logistyczną!

Realizacja ćwiczenia

Przypomnijmy oznaczenia i architekturę zbudowanej sieci jednokierunkowej



Architektura sieci zrealizowana na poprzednim laboratorium

Na poprzednim laboratorium zaimplementowana została sieć, której architektura pokazana jest na Rys.8 a przypomniana powyżej. Wykonanie poprzedniego ćwiczenia umożliwiło też przesłanie wzorca przez wszystkie warstwy, tak aby policzyć wyjście $\hat{\mathbf{y}}$ (z sieci (faza przesłania w przód).
Na tym laboratorium zadaniem jest implementacja możliwości uczenia sieci, czyli znajdowania jej parametrów θ (wag i biasów). Ich poszukiwanie odbywa się poprzez minimalizację funkcji straty L (inaczej nazywanej kosztem *C* lub błędem *E*).

Należy zaimplementować podany niżej algorytm uczenia parametrów sieci dla paczki *m* wzorców. Proszę pamiętać o tym by umożliwiać rozpoczęcie nauki od losowych wag z różnych zakresów. Należy umożliwić przerwanie uczenia i ponowne rozpoczęcie nauki od poprzednich wartości wag,

Optymalizowanym kosztem jest

$$C(f_\theta(\mathbf{x}_i), \mathbf{y}_i) = \sum_{j=1}^{M} -log\hat{y}_j y_j$$

Wejście x: dla warstwy wejściowej ustawiamy aktywację $\mathbf{a}^1$ równą wzorcowi wejściowemu
Przesłanie wejścia w przód: For each *l*=2,3,…,L oblicz

$\mathbf{z}^l = \mathrm{W}^l\mathbf{a}^{l-1} + \mathbf{b}^l$ oraz $\mathbf{a}^l=f(\mathbf{z}^l)$

Błąd na wyjściu: Obliczyć wektor

$$\boldsymbol{\delta}_{\mathbf{x}}^{\boldsymbol{L}} = \frac{\partial C_{\mathbf{x}}}{\partial a^L} \odot f'(\mathbf{z}_{\mathbf{x}}^L)$$

Dla przyjętej funkcji kosztu w warstwie L=3 mamy

$$\boldsymbol{\delta}_{\mathbf{x}}^{\boldsymbol{L}} = (\hat{\mathbf{y}} - \mathbf{y})$$

Rzutowanie błędu wstecz: For each l=L−1,L−2,…,2 oblicz

$$\boldsymbol{\delta}_{\mathbf{x}}^{l} = \left(\left(W^{l+1}\right)^T \boldsymbol{\delta}_{\mathbf{x}}^{l+1}\right) \odot \boldsymbol{f}'(\mathbf{z}_{\mathbf{x}}^l)$$

Uaktualnianie wag For each l=L,L−1,…,2
uaktualnianie wag zgodnie z regułą

$$\mathrm{W}^l = \mathrm{W}^l - \frac{\alpha}{m}\sum_{\mathbf{x}} \boldsymbol{\delta}_{\mathbf{x}}^{l} \left(\mathbf{a}_{\mathbf{x}}^{l-1}\right)^T$$

$$\mathbf{b}^l = \mathbf{b}^l - \frac{\alpha}{m}\sum_{\mathbf{x}} \boldsymbol{\delta}_{\mathbf{x}}^{l}$$

Faza przesłania w tył jest powtarzalną operacją stosowania reguły łańcuchowej, dlatego jest potencjalnym miejscem istnienia wielu błędów w aplikacji. Sprawdzenie gradientu jest jedną z możliwości ich znalezienia. Poniżej przedstawiony jest przydatny do tego celu algorytm Gradient checking.

Algorytm Gradient Checking

Zainicjuj losowo wartości parametrów $\theta = (W^1, b^1, W^2, \dots)$
Losowo zainicjuj **x** oraz **y**
Oblicz analitycznie gradient używając propagacji wstecznej $g_{analytic} = \frac{\partial L(f_\theta(\mathbf{x}),\mathbf{y})}{\partial \theta}$
For *i* in # θ
$\hat{\theta} = \theta$
$\hat{\theta} = \hat{\theta} + \varepsilon$
$g_{numeric} = \frac{L(f_{\hat{\theta}}(\mathbf{x}),\mathbf{y}) - L(f_\theta(\mathbf{x}),\mathbf{y})}{\varepsilon}$
Musi być spełnione: $(\|g_{numeric} - g_{analytic}\| < \epsilon$

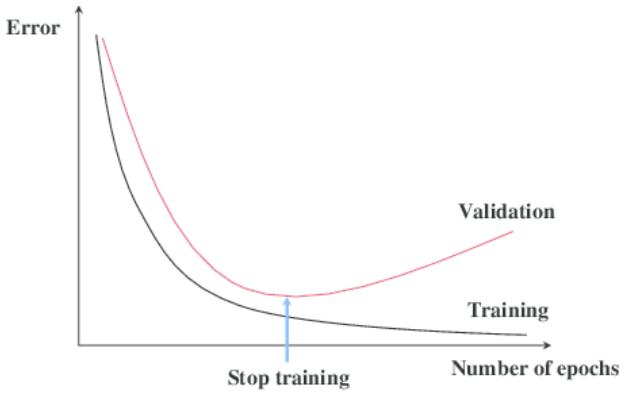
Sposób oceny

Za implementację zadania na zajęciach, można otrzymać 30% całkowitej liczby punktów.

Laboratorium 5: Przeprowadzenie badań eksperymentalnych dotyczących wpływu hiperparametrów na szybkość i skuteczność uczenia

Zbudowany model sieci pełni rolę klasyfikatora obrazów cyfr podawanych na wejście. Ocenę modelu przeprowadza się na zbiorze testowym, natomiast do oceny możliwości modelu w trakcie poszukiwania optymalnych wartości hiperparametrów wykorzystuje się zbiór walidujący. Uczenie przeprowadza się na zbiorze uczącym. Jak pamiętamy parametrami w sieci są wagi i biasy. Sieć ma wiele hiperparametrów, od których zależy szybkość i skuteczność uczenia. W tym ćwiczeniu zbadamy wrażliwość modelu na niektóre z nich. Należą do nich między innymi: liczba neuronów w warstwie ukrytej, zastosowana funkcja aktywacji, wartości początkowe wag, wartość współczynnika uczenia, liczność paczki wzorców, wartość współczynnika momentum itp

Ważną cechą każdego modelu jest jego zdolność do generalizacji, którą bada się przez obliczenie dokładności (accuracy) na zbiorze testowym. Jest procent przykładów testowych poprawnie zaklasyfikowanych przez model. W trakcie uczenia istotna jest minimalizacja funkcji kosztu, dlatego w trakcie uczenia dobrze jest obserwować ten proces, pokazując ten proces czyli przebieg funkcji kosztu w kolejnych epokach na zbiorze uczącym i zbiorze walidującym.
Wczesne zatrzymywanie uczenia. Hinton sugerował (NIPS 2015 Tutorial slajd 63), że zawsze powinniśmy monitorować postęp uczenia na zbiorze walidującym. Uczenie powinno być zatrzymywane, jeśli wartość kosztu na zbiorze walidującym rośnie powyżej określonej wartości (pozwalamy, aby błąd rósł, ale do określonej wartości - rzadko ma taki gładki przebieg jak na omawianym rysunku) wówczas przerywamy uczenie i wracamy do stanu, w którym błąd na zbiorze walidującym był najniższy .



Rys. 12. Wczesne zatrzymywanie uczenia (źródło: http://fouryears.eu/2017/12/06/the-mystery-of-early-stopping/)

Dlatego powinniśmy wizualizować przebieg funkcji kosztu zarówno na zbiorze uczącym, jak i na zbiorze walidującym.

Dropout

Realizacja ćwiczenia

Aby sobie ułatwić wykonanie ćwiczenia dobrze też byłoby zapewnić serializację badań i obserwację bieżącego błędu uczenia sieci.

Badania eksperymentalne obejmują:

1. Badanie wpływu liczności neuronów w warstwie ukrytej na przebieg procesu uczenia i jego wynik
2. Badanie wpływu wielkości paczki na proces uczenia
3. Badanie wpływu inicjalizacji wag na proces uczenia
4. Badanie wpływu wartości współczynnika uczenia na proces uczenia
5. Badanie wpływu funkcji aktywacji (Sigmoidalna/ReLU) na proces uczenia

Sposób oceny

Przebadanie wspływu rozmiaru warstwy ukrytej na proces uczenia - 10 % ogólnej liczby punktów przypisanej do zadania

Przebadanie wpływu wielkości paczki (batcha) - 10 % ogólnej liczby punktów przypisanej do zadania

Wpływ inicjalizacji wag (losowe wartości z różnych przedziałów) - 10 % ogólnej liczby punktów przypisanej do zadania

Wpływ współczynnika uczenia - 10 % ogólnej liczby punktów przypisanej do zadania

Przebadanie wpływu ReLU/sigmoida- - 10 % ogólnej liczby punktów przypisanej do zadania

UWAGA !!! Punkty bonusowe za implementację sprawdzania gradientu oraz dropout i regularyzację – 5% całkowitej liczby punktów za każdy wymieniony element

Przesłanie raportu do prowadzącego do dnia 4.11.2020

Interesujące linki:

https://medium.com/data-science-bootcamp/understand-the-softmax-function-in-minutes-f3a59641e86d
https://peterroelants.github.io/posts/cross-entropy-softmax/
http://bigstuffgoingon.com/blog/posts/softmax-loss-gradient/
https://web.stanford.edu/class/cs224n/readings/gradient-notes.pdf
https://www.deeplearning.ai/ai-notes/optimization/
https://www.deeplearning.ai/ai-notes/optimization/
https://eli.thegreenplace.net/2016/the-softmax-function-and-its-derivative/