ZPR PWr
Zintegrowany Program Rozwoju Politechniki Wrocławskiej

SIECI NEURONOWE – laboratorium

Ćwiczenie 4 – prosta sieć konwolucyjna (realizacja ćwiczenia na laboratorium8, laboratorium9)

Liczba punktów: 40

Celem ćwiczenia jest poznanie podstawowych operacji realizowanych w sieciach konwolucyjnych, sposobu przetwarzania informacji wejściowej i sposobu uczenia takich sieci.
Na to ćwiczenie przeznaczone są dwa laboratoria. Pierwsze z nich obejmuje zaprojektowanie i implementację architektury sieci, laboratorium drugie to implementacja metody oraz przeprowadzenie eksperymentów z jej uczeniem przy różnych wartościach hiperparametrów.

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

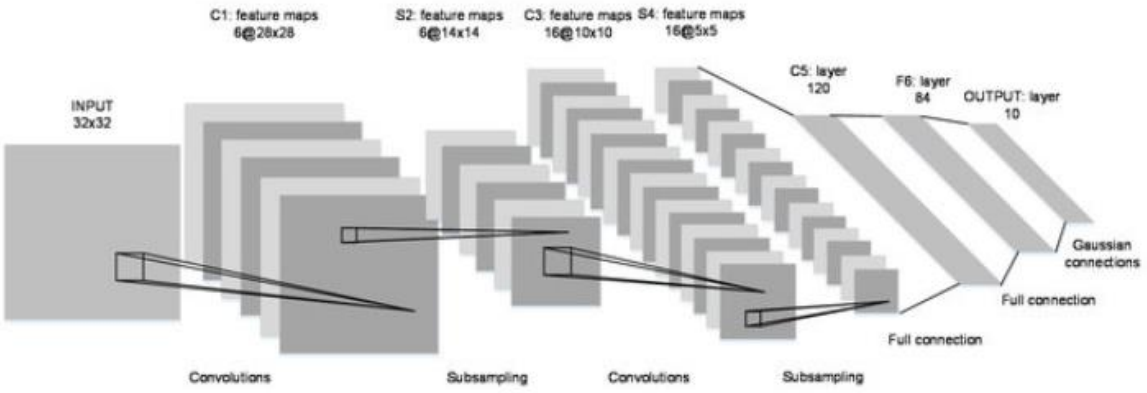
Fundusze Europejskie Wiedza Edukacja Rozwój
Politechnika Wrocławska
Unia Europejska Europejski Fundusz Społeczny

Laboratorium 8: Architektura prostej sieci konwolucyjnej

Cel: projekt i implementacja prostej architektury sieci konwolucyjnej do rozpoznawania cyfr ze zbioru MNIST.

Wprowadzenie teoretyczne

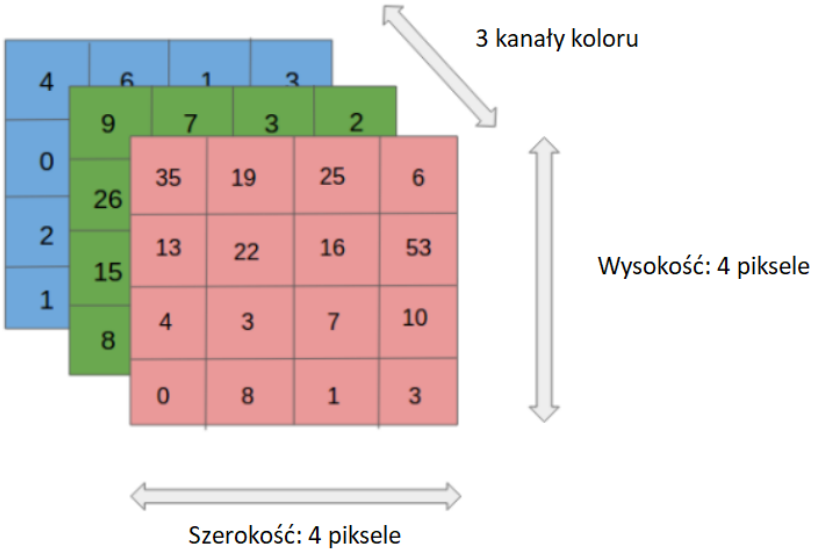
Pierwsza prosta sieć konwolucyjna nazwana przez jej twórcę LeNet pokazana jest na Rys. 1.



Rys. 1. LeNet – pierwsza sieć konwolucyjna ([LeCun et al., Gradient based learning applied to document recognition, 1998]

Istotą przetwarzania w takiej sieci jest operacja konwolucji, wykonywana w warstwach konwolucyjnych. Jak można zauważyć, na wejściu pierwsza warstwa konwolucyjna dostaje obraz, który na Rys.1 ma rozmiar 32x32. W całej sieci występuje wiele warstw konwolucyjnych. Warstwa ta składa się z wielu map cech ang. *feature maps*, inaczej (kanałów). Po warstwach konwolucyjnych występują warstwy, których zadaniem jest zmniejszenie wymiarowości. Wykonywana w nich jest operacja wyboru wartości maksymalnej z okna (ang. max pooling) albo uśrednianie (ang. *average pooling*). Rozmiar map cech w kolejnych warstwach zmniejsza się, ale zazwyczaj zwiększa się ich liczba w kolejnych warstwach konwolucyjnych. Sieć zakończona jest warstwami w pełni połączonymi, w której ostatnią warstwą jest (dla zadania klasyfikacji) warstwa softmax. Innymi słowy, końcowe w pełni połączone warstwy, to poznana na poprzednich ćwiczeniach sieć MLP. Od czasu sieci LeNet powstało wiele innych jej wersji, np. AlexNet, VGGNet, GoogLeNet, ResNet.

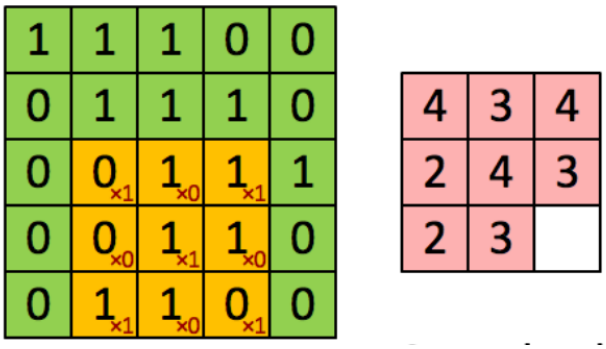
Architektura sieci konwolucyjnej została zaprojektowana w celu łatwiejszego przetwarzania dwuwymiarowej struktury obrazu (zachowuje zależności przestrzenne między obrazami). Osiąga się to dzięki lokalnym połączeniom i współdzielonym wagom. Zaletą CNN jest mniejsza liczba parametrów (wag i biasów) w porównaniu do MLP z tą samą liczbą warstw
Po przedstawieniu ogólnej architektury sieci konwolucyjnej zajmiemy się jej poszczególnymi elementami. Zaczniemy od obrazu wejściowego.
Obraz wejściowy w skali RGB jest dekomponowany na 3 kanały odpowiadające trzem kolorom składowym. Obraz ma swoje wymiary (wysokość i szerokość podane w pikselach)



Rys. 2. Obraz RGB w tym przykładzie o wymiarach 4x4x3 (źródło: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53)

W ten sposób na obraz patrzymy przez trzy wymiary (wysokość x szerokość x liczba kanałów). **Warstwa konwolucyjna** Zachowuje strukturę przestrzenną wzorca wejściowego. Każdy neuron w warstwie konwolucyjnej widzi tylko niewielką część obrazu (albo poprzedzającej go warstwy dla dalszych warstw konwolucyjnych). Ten obszar, który obejmuje neuron nazywany jest oknem. Może mieć rozmiar 3x3 lub 5x5 lub większy. Połączenia wchodzące do danego neuronu od poszczególnych elementów znajdujących się w oknie mają przypisane wagi, które nazywane są jądrem lub filtrem. Odpowiadają one wykrywanym cechom na obrazie, na przykład we wczesnych warstwach mogą to być segmenty lub plamy. Im dalej w kierunku do wyjścia sieci, tym cechy są coraz bardziej złożone. Wszystkie neurony należące do jednej mapy cech w warstwie współdzielą wagi. To oznacza, że mapa cech jest odpowiedzialna za wykrycie jednej cechy na obrazie a kolejne neurony są poszukują jej w innym miejscu. Załóżmy, że nasz filtr ma wymiar 3x3.
Jądro przesuwa się po obrazie i wykonywana jest operacja konwolucji. Na załączonym rysunku przedstawiono wykonywaną operację konwolucji. Zielonym kwadratem zaznaczony jest obraz z wartościami pikseli dla ułatwienia równymi 0 lub1. Na żółto oznaczone jest jądro (filtr) a na różowo przedstawiono wynik operacji konwolucji. Jądro odpowiada macierzy o wartościach:

$$\begin{matrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{matrix}$$



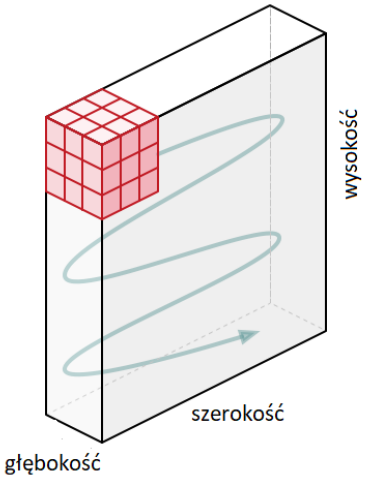
Rys. 3. Filtr przesuwający się po płaskiej powierzchni (jednej mapie cech lub jednym kanale obrazu) i wynik działania operacji konwolucji

W przykładzie pokazanym na Rys. 3. w wyniku konwolucji otrzymujemy wartość =3 pokazaną w mapie cech oznaczonej na różowo. Po przesunięciu filtra w prawo i wykonaniu operacji konwolucji wyliczona zostanie ostatnia brakująca wartość w tej macierzy.

Formalnie pobudzenie neuronu o pozycji (x,y) na mapie cech w (l+1) warstwie konwolucyjnej możemy zapisać:

$$z_{x.y}^{l+1} = W^{l+1} * f\left(z_{x,y}^{l}\right) + b_{x,y}^{l+1} = \sum_{a}\sum_{b} w_{a,b}^{l+1} f\left(z_{x-a,y-b}^{l}\right) + b_{x,y}^{l+1}$$

Obliczenie całkowitego pobudzenia neuronu w warstwie l+1 wyraża się podobnie do poznanej formuły dla sieci MLP z ćwiczenia 2, jednak iloczyn wag i aktywacji z poprzedniej warstwy ograniczony jest do sygnałów aktywacji z okna (filtra) z poprzedniej warstwy (obliczana jest operacja konwolucji oznaczana jako * pomiędzy wagami filtra i wartościami aktywacji z warstwy l). Dla pierwszej warstwy konwolucyjnej aktywacja poprzedniej warstwy to wartości pikseli na obrazie. Funkcja *f* jest funkcją aktywacji (zazwyczaj ReLU), W^{l+1} to wagi filtra. Dla całej mapy cech występuje jeden filtr. Nazywa się to współdzieleniem wag dla danej mapy cech.
Dla wielu map cech w warstwie poprzedzającej rozpatrywany neuron przemieszczanie się filtra ilustruje Rys. 4. Jeśli wyobrazimy sobie wiele kanałów (map cech) w warstwie poprzedzającej, to filtr przyjmuje formę tensora (uogólnienie macierzy do wielu wymiarów). W tym konkretnym przypadku na Rys. 4 pokazany jest w postaci sześcianu oznaczonego na różowo.



Rys. 4. Przesuwanie się filtra po poprzedzającej go warstwie (źródło: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53)

Warto dodać, że filtr może się przesuwać po przedzającej warstwie (obrazie) z krokiem (ang. stride) większym lub równym 1 pikselowi i jeśli rozmiar filtra nie mieści się całkowitą liczbę razy w mapie cech, musi być ona uzupełniona ramką (ang. padding).

Jeśli oznaczymy wymiar obrazu (lub poprzedzającego kanału) przez W, rozmiar filtra przez F, rozmiar kroku przez S natomiast szerokość ramki (padding) przez P, to formuła opisująca rozmiar mapy cech w kolejnej warstwie jest następująca:

$$\frac{W - F + 2P}{S} + 1$$

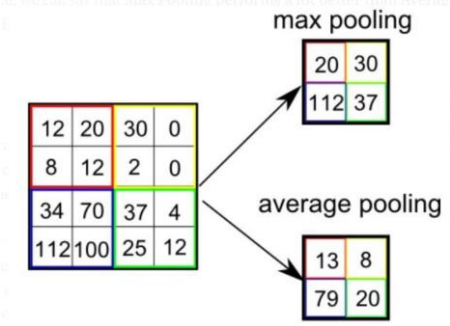
Dla przykładu, jeśli przyjmiemy, że rozmiar mapy cech jest 38x38 (W=38), filtr ma wymiary 3x3 (F=3), przy kroku równym 1 (S=1) i zerowej ramce (P=0) rozmiar powstałej mapy cech będzie wynosił 36. Jednak przy kroku równym 2 (S=2) nie jest możliwe wykonanie operacji przy zerowej ramce i podanej wielkości filtra, bo filtr nie mieściłby się całkowitą liczbę razy.

Warstwa konwolucyjna charakteryzuje się następującymi hiperparametrami:

- Jej rozmiar zdefiniowany jest przez W1×H1×D1
- Wymaga 4 hiperparametrów:
  - Liczba filtrów K,
  - Rozmiar filtra F,
  - Krok **S** ,
  - Szerokość ramki P.
- Produkuje rozmiar W2×H2×D2 where:
  - W2=(W1−F+2P)/S+1
  - H2=(H1−F+2P)/S+1(tzn szerokość I wysokość są określane w taki sam sposób)
  - D2=K
- Uwzględniają współdzielenie wag mamy F·F· D1 wag na filtr,
- Całkowita liczba wag to (F·F·D1)·K i K biasów.

Najbardziej popularne ustawienie hiperparametrów to F=3, S=1, P=1.

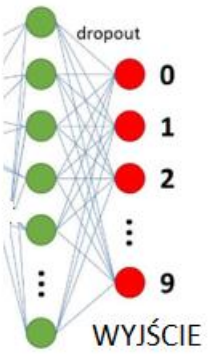
Warstwa pooling. Zazwyczaj po warstwie konwolucyjnej występuje warstwa poolingu, która ma 2 wersje. Może być realizowana jako operacja uśredniania (ang. average pooling) lub wyszukiwania wartości maksymalnej (ang. max pooling) w oknie.



Rys.5 Wersje operacji wykonywane w warstwach pooling

W przykładzie pokazanym na Rys.5. okno wynosi 2x2. W warstwach *poolingu* nie ma wyuczalnych wag. Możemy zatem powiedzieć, że na wszystkich połączeniach w tej warstwie wagi są równe jeden.

Warstwa w pełni połączona. Zanim przejdziemy do warstwy w pełni połączonej, która pokazana jest na Rys. 6. Musimy dokonać operacji spłaszczenia, gdyż poprzedzająca warstwa zawiera wiele map cech.

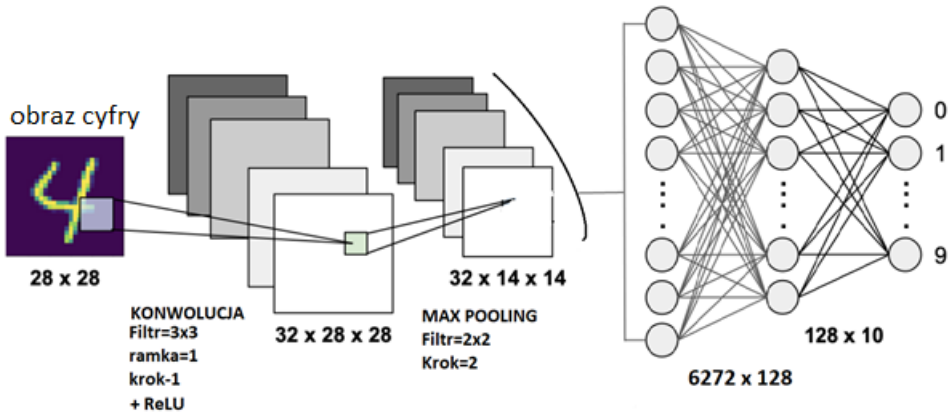


Rys. 6. Warstwy w pełni połączone

Zazwyczaj osiągamy to przez zastosowanie konwolucji 1x1 do poprzedniej warstwy. W ostatniej warstwie dokonującej klasyfikacji stosujemy funkcję softmax, poznaną w ćwiczeniu 2. Pozostałe warstwy w pełni połączone (jeśli jest ich więcej) mają strukturę, którą poznaliśmy w poprzednich ćwiczeniach.

Realizacja ćwiczenia

Należy zaprojektować i zaimplementować prostą architekturę sieci konwolucyjnej do rozpoznawania cyfr ze zbioru MNIST (Rys.7). W tym zadaniu nie nastawiamy się na poprawę wyników rozpoznawania w stosunku do sieci MLP, ale na poznanie podstawowych elementów sieci konwolucyjnej. Dlatego w ćwiczeniu ograniczymy się do architektury pokazanej na Rys. 7. Zawiera ona tylko jedną warstwę konwolucyjną, jedną warstwę poolingu i jedną warstwę w pełni połączoną.



Rys.7. Uproszczona architektura sieci konwolucyjnej do rozpoznawania zbioru MNIST do realizacji w ćwiczeniu 4.

Przyjmiemy, że na wejściu pojawia się obraz 28x28 pikseli. Warstwa konwolucyjna będzie zawierała 32 mapy cech, czyli będziemy mieć 32 różne filtry. Filtr w warstwie konwolucyjnej ma rozmiar 3x3 a krok P jest równy 1. Po obliczeniu konwolucji proszę zastosować funkcję aktywacji ReLU. Pooling proszę zrealizować używając operacji max. Hiperparametry dla

warstwy w pełni połączonej proszę dobrać na podstawie wyników otrzymanych w ćwiczeniu drugim i trzecim.

Realizacja tej części ćwiczenia wymaga implementacji fazy przesłania sygnału w przód.

1. Implementacja przetwarzania w warstwie konwolucyjnej

Zainicjować losowo wagi filtrów według metody He.
W pierwszej kolejności musimy obliczyć wynik operacji konwolucji dla każdego neuronu w warstwie konwolucyjnej

$$z^1_{x.y} = W^1 * \left(x_{x,y}\right) + b^1_{x,y} = \sum_a \sum_b w^1_{a,b} \left(x_{x-a,y-b}\right) + b^1_{x,y} = rot_{180^\circ}(w^1_{a,b}) * \left(x_{x,y}\right) + b^1_{x,y}$$

A następnie przekształcić go przez funkcję aktywacji *f*.

$$a^1_{x.y} = f(z^1_{x.y})$$

2. Implementacja przetwarzania w warstwie pooling

Wejściem do tej warstwy są wartości funkcji aktywacji z warstwy konwolucyjnej. Przechodzimy oknem 2x2 z krokiem równym 2 przejście po warstwie konwolucyjnej implementując operację max dla każdego okna.

Aby umożliwić uczenie sieci, podczas fazy przesłania sygnału w przód proszę zapamiętywać indeks neuronu, dla którego była znaleziona wartość maksymalna. W tej warstwie na połączeniach nie wag, które są uczone. Przyjmujemy, że wszystkie wartości wag są równe 1.

3. Implementacja warstw sieci w pełni połączonych z warstwą softmax jako wyjściem

Ta część sieci jest identyczna z siecią realizowaną w ćwiczeniu drugim i trzecim. Na podstawie przeprowadzonych w tych ćwiczeniach eksperymentów proszę wybrać najlepsze hiperparametry dostosowując liczbę neuronów w warstwach do wskazanej na Rys.7.

Sposób oceny:

Implementacja pełnej architektury sieci i fazy przesłania w przód -

30% całkowitej liczby punktów przydzielonej do ćwiczenia

Użyteczne linki:

http://ufldl.stanford.edu/tutorial/supervised/ConvolutionalNeuralNetwork/

https://victorzhou.com/blog/intro-to-cnns-part-1/

Laboratorium 9 – Metoda uczenia sieci konwolucyjnej

Cel: należy zaimplementować metodę uczenia sieci konwolucyjnej, zaprojektowanej do rozpoznawania cyfr ze zbioru MNIST. Sieć uczona jest metodą SGD, którą poznaliśmy realizując sieć MLP.

We wszystkich współcześnie używanych platformach do uczenia sieci głębokich uczenie SGD w tym sieci sieci konwolucyjnych, metoda SGD jest zaimplementowana. Na tym laboratorium poznamy tę metodę implementując ją samodzielnie od podstaw przy użyciu pakietu *numpy*. Przedstawiona dalej realizacja algorytmu uczenia nie jest optymalna, ale pozwala na zapoznanie się ze sposobem uczenia w każdej warstwie.

Wprowadzenie teoretyczne

Uczenie każdej sieci metodą SGD wymaga przesłania sygnału wprzód (faza *forward*) i rzutowania błędów wstecz (faza *backward*), czyli po podaniu wzorca uczącego musimy policzyć wyjście sieci i na tej podstawie policzyć wartość funkcji straty a następnie obliczyć gradienty błędów w każdej warstwie.
Warto na początek zestawić architekturę sieci CNN w tym ćwiczeniu:

Warstwa konwolucyjna dokonuje przekształcenia 28x2x1 => 28x28x32
Następna warstwa to max pooling, dokonuje przekształcenia 28x28x32 => 14x14x32
Warstwy MLP w pełni połączone:
najpierw następuje transformacja 14x14x32 => 6772; 6272=>128;
warstwa softmax128=>10

A więc nasza sieć ma w sumie pięć warstw.

Jako funkcję kosztu *C* przyjmiemy, podobnie jak w ćwiczeniu drugim, ujemny gradient szansy (negative log likelihood), który wyraża się następująco:

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

W przypadku sieci konwolucyjnej mamy 3 różne elementy sieci, w których w różny sposób przebiega przetwarzanie i różnie obliczany jest gradient, konieczny do aktualizacji wag. Rozpoczynając od warstwy wyjściowej, musimy obliczyć zmiany wag od warstwy softmax.

1.W pierwszej kolejności obliczamy gradient funkcji straty dla warstwy wyjściowej (piątej).

$$\frac{\partial C(f(\boldsymbol{z}^5), \mathbf{y})}{\partial W^5} = \frac{\partial C\left((f(\boldsymbol{z}^5), \mathbf{y})\right)}{\partial\, \boldsymbol{z}^5} \frac{\partial\, \boldsymbol{z}^5}{\partial W^5}$$

Jak można zauważyć na rysunku przedstawiającym architekturę sieci ostatnia warstwa to warstwa softmax. Gradient z funkcji kosztu po funkcji softmax jest równy

$$\frac{\partial C\left((f(\boldsymbol{z}^5), \mathbf{y})\right)}{\partial\, \boldsymbol{z}^5} = -(\boldsymbol{y} - f(\boldsymbol{z}^5)) = (\hat{\mathbf{y}} - \mathbf{y}) \qquad \boldsymbol{\delta}^5$$

gdzie $\mathbf{y}$ jest kodowane jako „1 z n”, a $\hat{\mathbf{y}}$ jest aktualnym wektorem wyjściowym z sieci Przypomnijmy, że $\mathbf{z}^5 = \mathbf{W}^5\mathbf{a}^4 + \mathbf{b}^5$, więc gradient $\boldsymbol{z}^5$ względem ∂W^5wyraża się następująco:

$$\frac{\partial\, \boldsymbol{z}^5}{\partial W^5} = \left(\mathbf{a^4}\right)^T$$

Po połączeniu obu wyrażeń otrzymujemy:

$$\frac{\partial C(f(\boldsymbol{z}^5), \mathbf{y})}{\partial W^5} = -(\boldsymbol{y} - f(\boldsymbol{z}^5))\left(\mathbf{a^4}\right)^T = \boldsymbol{\delta}^5\left(\mathbf{a^4}\right)^T$$

2.Teraz przechodzimy do warstwy czwartej

$$\frac{\partial C(f(\boldsymbol{z}^5), \mathbf{y})}{\partial \mathbf{a}^4} = \frac{\partial C\left((f(\boldsymbol{z}^5), \mathbf{y})\right)}{\partial\, \boldsymbol{z}^5}\frac{\partial\, \boldsymbol{z}^5}{\partial \mathbf{a}^4}$$

Biorąc pod uwagę, że $\mathbf{z}^5 = \mathbf{W}^5\mathbf{a}^4 + \mathbf{b}^5$ ostatni element będzie macierzą wag$\mathbf{W}^5$
A więc równanie możemy zapisać jako:

$$\frac{\partial C(f(\boldsymbol{z}^5), \mathbf{y})}{\partial \mathbf{a}^4} = (\mathbf{W}^5)^T\frac{\partial C\left((f(\boldsymbol{z}^5), \mathbf{y})\right)}{\partial\, \boldsymbol{z}^5} = (\mathbf{W}^5)^T\boldsymbol{\delta}^5$$

Gradient po wagach w warstwie czwartej

$$\frac{\partial C(f(\boldsymbol{z}^5), \mathbf{y})}{\partial W^4} = \frac{\partial C(f(\boldsymbol{z}^5), \mathbf{y})}{\partial \mathbf{a}^4}\frac{\partial\, \boldsymbol{a}^4}{\partial \boldsymbol{z}^4}\frac{\partial\, \boldsymbol{z}^4}{\partial \mathrm{W}^4}$$

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

Wiemy, że $\mathbf{z}^4 = \mathbf{W}^4\mathbf{a}^3 + \mathbf{b}^4$, więc ostatni element gradientu to będzie $(\mathbf{a}^3)^T$ Ostatecznie mamy:

$$\frac{\partial C(f(\boldsymbol{z}^5), \mathbf{y})}{\partial W^4} = ((\mathbf{W}^5)^T\boldsymbol{\delta}^5)\odot f'^4(\mathbf{a}^3)^T$$

Oznaczając jako błąd $\boldsymbol{\delta}^4$ w tej warstwie

$$\boldsymbol{\delta}^4 = (\mathbf{W}^5)^T\boldsymbol{\delta}^5\odot f'^4$$

Otrzymujemy gradient błędu po wagach w warstwie drugiej

$$\frac{\partial C(f(\boldsymbol{z}^5), \mathbf{y})}{\partial W^4} = \boldsymbol{\delta}^4(\mathbf{a}^3)^T$$

Gradient w odniesieniu do biasów:

$$\frac{\partial C(f(\boldsymbol{z}^5), \mathbf{y})}{\partial \mathbf{b}^4} = \boldsymbol{\delta}^4$$

3.Dalej powtarzamy ten sam schemat dla warstwy trzeciej.
Oznaczmy błąd dla warstwy pierwszej jako:

$$\boldsymbol{\delta}^3 = ((\mathbf{W}^4)^T \boldsymbol{\delta}^4) \odot f'^3$$

Wówczas gradient błędu po wagach w warstwie trzeciej wynosi:

$$\frac{\partial C(f(\boldsymbol{z}^5), \mathbf{y})}{\partial W^3} = \boldsymbol{\delta}^3 (\mathbf{a}^2)^T$$

Gradient w odniesieniu do biasów

$$\frac{\partial C(f(\boldsymbol{z}^5), \mathbf{y})}{\partial \mathbf{b}^3} = \boldsymbol{\delta}^3$$

4.Teraz przechodzimy do warstwy drugiej, która jest warstwą z operacją max pooling
W tej warstwie następuje przesłanie błędu z warstwy 3 do tych neuronów w warstwie konwolucyjnej, Rozpatrujemy każdy filtr oddzielnie i przesyłamy gradient do tych neuronów, które miały maksymalne wyjścia, ponieważ bardzo małe zmiany zmiany w wejściach mogą zaburzać wynik jedynie poprzez te jednostki.
Operacje rzutowania błędu dla danego filtra k zapiszemy jako:

$$\boldsymbol{\delta}^2 = \text{upsamle}(\boldsymbol{\delta}^3)$$

Operacja upsample oznacza, że wektor błędów $\boldsymbol{\delta}^2$ będzie miał wszędzie 0, oprócz $\boldsymbol{\delta}^2_{i*}$ gdzie $i^* = \underset{i}{\text{argmax}}\, a_i^1$

5.Obliczymy teraz błąd $\boldsymbol{\delta}^1$ w warstwie pierwszej (konwolucyjnej) korzystając z ogólnego wzoru, że jest to błąd w warstwie wyższej pomnożony przez wagi i pochodną funkcji aktywacji w warstwie pierwszej. W tym przypadku wagi na połączeniach między warstwą pooling i konwolucyjną są równe 1 zatem

$$\boldsymbol{\delta}^1 = \boldsymbol{\delta}^2 \odot f'(z^1)$$

Po obliczeniu błędu w warstwie konwolucyjnej jesteśmy gotowi określenia przyrostu wagi $w_{a,b}^1$ w filtrze o indeksach *a*,*b*

$$\frac{\partial C}{\partial w_{a,b}^1} = \sum_x \sum_y \frac{\partial C}{\partial z_{a,b}^1} \frac{\partial z_{a,b}^1}{\partial w_{a,b}^1} = \sum_x \sum_y \delta_{x,y}^1 \frac{\partial (\sum_{a'} \sum_{b'} w_{a',b'}^1 \; f\left(z_{x-a',y-b'}^1\right) + b_{x,y}^1)}{\partial w_{a,b}^1}$$

$$= \sum_x \sum_y \delta_{x,y}^1 f(z_{x-a',y-b'}^0) = \delta_{a,b}^1 * f(z_{x-a',y-b'}^0) = \delta_{a,b}^1 * ROT_{180^\circ}(x_{a,b})$$

Sumowanie po wszystkich gradientach w warstwie pierwszej

Obraz wejściowy X

$$\Delta w^1_{a,b} = \frac{\partial C}{\partial w^1_{a,b}}$$

Zmiana wag od bywa się więc zgodnie ze wzorem

$$w^1_{a,b} = w^1_{a,b} + \alpha \Delta w^1_{a,b}$$

Podsumowanie. Jak już wspomniano jest to można powiedzieć naiwne podejście do uczenia sieci konwolucyjnej. W każdej współczesnej platformie do uczenia sieci konwolucyjnych stosowane metody są zoptymalizowane pod kątem ich szybkości uczenia (przekształceń macierzowych). Jednak jak wspomniano we wstępie celem tego ćwiczenia było poznanie podstawowych operacji wykonywanych w sieciach konwolucyjnych, aby je rozumieć kiedy korzystamy z gotowych bibliotek.

Realizacja ćwiczenia

Na tym laboratorium należy zaimplementować przedstawioną w części teoretycznej metodę uczenia.

Dla warstw w pełni połączonych można wykorzystać implementację metody uczenia z ćwiczenia drugiego. Należy pamiętać o obliczeniu gradientu błędów w warstwie trzeciej.

Nowe elementy w procedurze uczenia to punkty 4. i 5. z algorytmu przedstawionego w części teoretycznej, które trzeba zaimplementować.

Proszę użyć inicjalizacji wag metodą He i optymalizacji współczynnika uczenia metodą Adam.

Zbudowana aplikacja powinna posłużyć do wykonania badań eksperymentalnych, w których zbadany zostanie:

1.Wpływ wielkości filtra w warstwie konwolucyjnej na dokładność rozpoznawania (accuracy)

2. Porównanie otrzymanych wyników z wynikami sieci NLP

Sposób oceny:

30% - Implementacja na zajęciach metody uczenia

20% - Wpływ wielkości filtra (kernela) w warstwie konwolucyjnej

20% - Porównanie otrzymanych wyników z wynikami sieci MLP

Raport do 9.12.2020

Użyteczne linki:

http://deeplearning.stanford.edu/tutorial/supervised/ConvolutionalNeuralNetwork/

https://www.jefkine.com/general/2016/09/05/backpropagation-in-convolutional-neural-networks/
https://victorzhou.com/blog/intro-to-cnns-part-2/
https://towardsdatascience.com/backpropagation-in-a-convolutional-layer-24c8d64d8509
http://cs231n.github.io/neural-networks-case-study/
http://cs231n.github.io/neural-networks-3/
https://towardsdatascience.com/gentle-dive-into-math-behind-convolutional-neural-networks-79a07dd44cf9
https://www.youtube.com/watch?v=BvrWiL2fd0M
https://medium.com/@2017csm1006/forward-and-backpropagation-in-convolutional-neural-network-4dfa96d7b37e