SIECI NEURONOWE: KRÓTKI* TUTORIAL

Autor: Radosław Łazarz

Sieci neuronowe to struktury algebraiczne luźno inspirowane strukturą ludzkiego mózgu. Odpowiednio zaprojektowane i poddane procedurze uczenia stają się potężnym narzędziem pozwalającym na modelowanie złożonych procesów i predykcję ich rezultatów.

ROZDZIAŁ PIERWSZY (I OSTATNI…): THROUGH THE LOOKING GLASS...

PRZYGOTOWANIE ŚRODOWISKA DO DALSZEJ PRACY

UWAGA! Rozpocznij przygotowywanie środowiska przed przeczytaniem reszty dokumentu (będą „przestoje”, np. na pobieranie/indeksowanie bibliotek).

- Będziesz potrzebował(a) interpretera języka Python (polecana wersja 3.8) oraz bibliotek wymienionych w pliku **requirements.txt** (warto zainstalować je w tzw. Środowisku wirtualnym).
- Przydatne może być skorzystanie z IDE PyCharm w wersji Professional (w trybie naukowym zapewnia bardzo wygodny podczas debugowania podgląd zawartości macierzy).

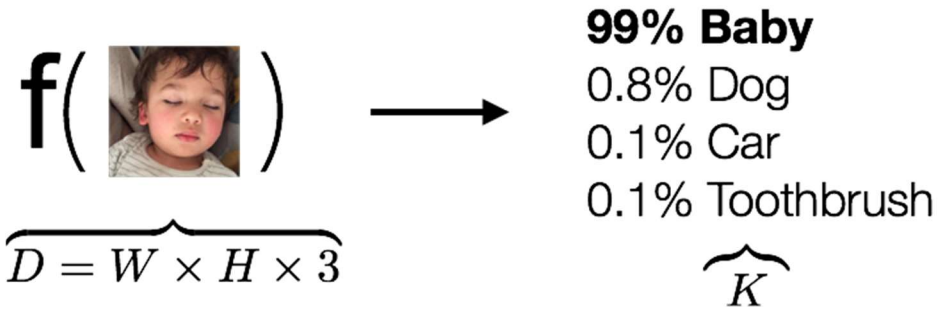
PROBLEM KLASYFIKACJI OBRAZÓW

Wprowadzenie zaczniemy od problemu klasyfikacji obrazów. Pozornie jest on bardzo prosty: przyporządkuj wejściową fotografię do jednej z kilku zadanych kategorii („pies”, „małpa”, „samolot”, itp.).

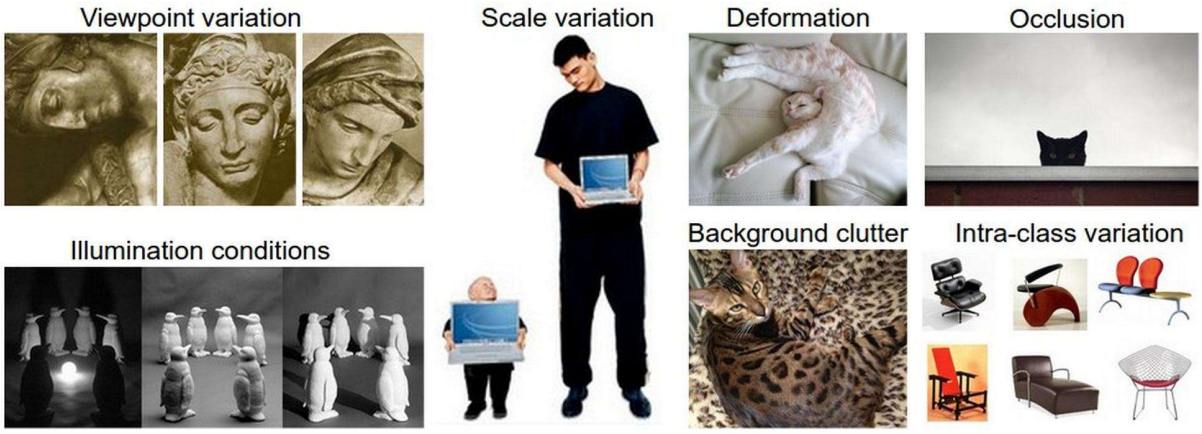
Jego bardziej precyzyjna definicja wygląda następująco. Kolorowe zdjęcie (RGB), szerokie na **W** pikseli i wysokie na **H** pikseli może być reprezentowane w pamięci komputera jako tablica **D = W * H * 3** liczb z zakresu od 0 (minimalna jasność) do 1 (maksymalna jasność. Załóżmy również, że mamy **K** kategorii do których chcemy zaklasyfikować nasz obraz. Naszym zadaniem jest więc znalezienie funkcji, która przyjmie tą tablicę jako wejście i zwróci etykietę właściwej kategorii.

Co jeżeli takie jasne przyporządkowanie nie jest jednak wykonalne? Prostym rozwiązaniem jest rozszerzenie naszej funkcji w ten sposób, by dla każdej kategorii zwracała % prawdopodobieństwa takiego rozwiązania – pozwoli nam to podjąć decyzję jak również zweryfikować jak bardzo możemy na niej polegać. Co do zasady potrzebujemy więc znaleźć funkcję $\mathbf{f} : \mathbb{R}^D \mapsto \mathbb{R}^K$.

Jak się za to zabrać? Naiwnym podejściem byłaby próba ręcznej specyfikacji pewnych cech (niemowlęta mają duże głowy, szczoteczki są długie, etc.). Szybko jednak stwierdzilibyśmy, że nawet dla niewielkiego zbioru kategorii jest to tytaniczna praca bez gwarancji sukcesu. Co więcej, istnieje wiele czynników zniekształcających zawartość naszych zdjęć. Obiekty mogą być przedstawiane z różnych ujęć, w różnych warunkach oświetleniowych, w różnej skali, częściowo niewidoczne, ukryte w tle…



* To niestety kłamstwo. Mam nadzieję, że jedyne zawarte w tym dokumencie.



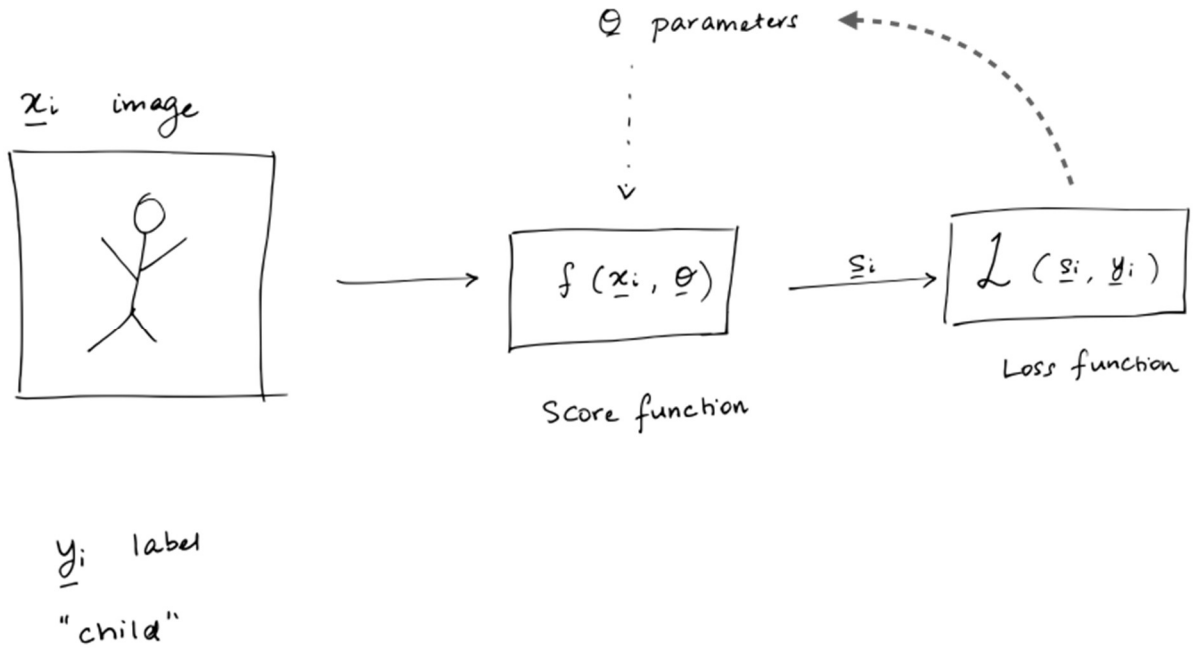
Wszystkie wymienione problemy są skutkiem istnienia semantycznej przepaści między tym jak reprezentowane są nasze dane wejściowe (tablica liczb), a tym czego w nich szukamy (kategorii i cech: zwierząt, nosów, głów, itp.). Zamiast więc próbować samodzielnie napisać funkcję **f** spróbujemy skorzystać z dobrodziejstw uczenia maszynowego by automatycznie skonstruować reprezentację wejścia właściwą dla postawionego sobie zadania (a przynajmniej lepszą od pierwotnej).

UCZENIE Z NADZOREM

Skorzystamy w tym celu z poddziedziny uczenia maszynowego zwanej uczeniem z nadzorem – obecnie najpopularniejszego podejścia do uczenia się na podstawie zbiorów danych. Można je opisać w następujących krokach.

- Startujemy posiadając zbiór obrazów o znanych wcześniej klasach (np. przyporządkowanych przez ludzkiego eksperta, tytułowego nadzorcę). Zbiór ten będziemy nazywać zbiorem treningowym (*training data*), stanowić on będzie bazę wiedzy do przyszłego uczenia systemu.
- Funkcja **f**, którą próbujemy znaleźć jest zazwyczaj zwana funkcją oceny (*score function*). Zakładamy z góry, że na jej działanie będzie miał wpływ szereg parametrów **θ**.
- Wprowadzamy również dodatkową funkcję **L**, tak zwaną funkcję straty (*loss function*). Będzie ona opisać jak bardzo wyniki zwracane przez funkcję **f** są zgodne z naszymi założeniami i oczekiwaniami. Duże wartości funkcji **L** oznaczają, iż funkcja **f** nie działa zbyt dobrze.
- Na koniec uruchamiamy algorytm optymalizujący, często nazywany po prostu algorytmem uczącym. Jego zadaniem jest iteracyjne poprawianie wartości **θ** na bazie przykładów uczących tak, by zapewnić jak najniższe **L**.
- Kiedy proces się zakończy, mamy nadzieję że uzyskana funkcja **f** ma wysoką zdolność do generalizacji – to jest, działa porównywalnie dobrze dla danych wejściowych z których nie korzystaliśmy uprzednio (tzw. test data).

Większość wyzwań współczesnego uczenia maszynowego sprowadza się do wyboru jak najlepszych narzędzi realizujących powyższe kroki. W tym tutorialu będziemy sukcesywnie przechodzić od tych najbardziej trywialnych do nieco bardziej skomplikowanych, po drodze poprawiając uzyskiwane rezultaty.



Warto jeszcze zwrócić uwagę na to, jaką formę może mieć opis właściwej, wyznaczonej przez eksperta klasy ($\mathbf{y_i}$). Planujemy porównywać go z oceną (score) $\mathbf{s_i}$, będącą wektorem % szans na przynależenie do danej klasy. W tej sytuacji najprościej by $\mathbf{y_i}$ był wektorem zawierającym zera na wszystkich pozycjach poza jedną, gdzie pojawi się jedynka (oznaczająca 100%). Taką reprezentację nazywa się często „*one hot encoding*".

KLASYFIKATOR LINIOWY

Niech nasze wejściowe zdjęcie będzie reprezentowane przez tablicę (wektor) **x**. Najprostszym przykładem szukanej funkcji **f** jest funkcja liniowa

$$\mathbf{f}(\mathbf{x}; \mathbf{W}, \mathbf{b}) = \mathbf{W}\mathbf{x} + \mathbf{b},$$

w przypadku której parametry **θ** to wagi **W** i bias **b**. **W** jest macierzą, **b** zaś – wektorem. W praktyce jest to równoznaczne jednoczesnemu wykonaniu wielu funkcji $\mathbf{w_j x} + \mathbf{b_j}$ ($\mathbf{w_j}$ będące wektorem, $\mathbf{b_j}$ skalarem) – po jednej dla każdej z **j** możliwych klas wynikowych. W zależności od tego jak zdefiniowane jest **W** i **x** (czy wektory są pionowe czy poziome) operacja przyjmuje opisaną wyżej formę, lub alternatywną **xW** + **b.**

Pojawia się tu jednak pewien drobny problem. Produkowane przez nią wyniki nie zawsze mieszczą się w zakresie od 0 do 1 (oraz niekoniecznie sumują do 100%) i trudno byłoby je interpretować jako prawdopodobieństwa. By poradzić sobie z tym mankamentem przepuścimy je przez dodatkową funkcję **σ**, zwaną funkcją softmax (miękkie maksimum: zamiast wybierać jeden wynik oblicza % prawdopodobieństwa), która sprowadzi je do oczekiwanego zakresu (zakładamy, że funkcja **f** to tak naprawdę zbiór podfunkcji $\mathbf{f_j}$ – po jednej dla każdej z rozważanych klas).

$$s_j = \sigma(\mathbf{f})_j = \frac{e^{f_j}}{\sum_{k=1}^{K} e^{f_k}}$$

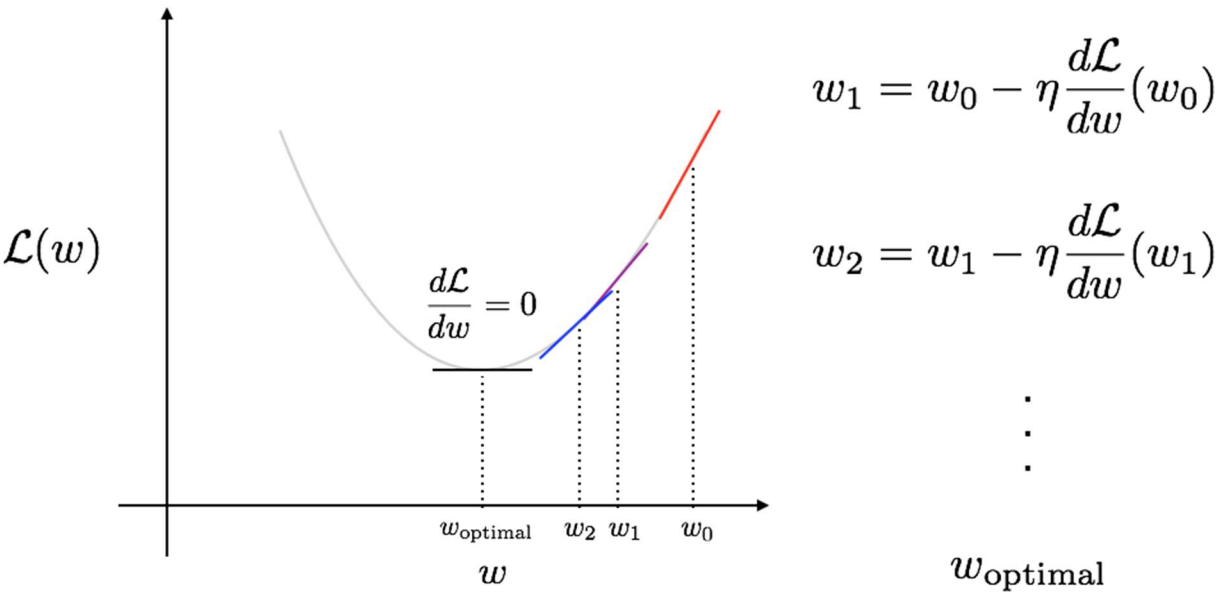
Jak teraz zdefiniować stratę (loss) **L**, która oceni jakość takiego **f**? Zakładając, że dla każdego z treningowych zdjęć $\mathbf{x_i}$ właściwą kategorią jest $\mathbf{y_i}$ (tu rozumiane jako indeks właściwej kategorii) jednym z rozwiązań jest policzenie tzw. entropii krzyżowej (cross entropy).

$$\mathcal{L}(\mathbf{s}) = -\sum_i log(s_{y_i})$$

Jeżeli dobrze przyjrzymy się powyższemu wzorowi, to zauważymy że im bliżej wynik odpowiedniego **f** jest do oczekiwanego, tym bliższy 0 jest jego przyczynek do funkcji straty (s_{y_i} powinno być jak najbliższe 1 – bo to właściwa klasa).

Pamiętaj, że nie sumujemy wszystkich składowych kolejnych wektorów **s**, a tylko te, które miały być zapalone. Postępowanie odwrotne to typowy błąd wstrzymujący proces uczenia (**L** nie zależałoby wtedy w żaden sposób od tego, jak podobne są uzyskane wyniki do oczekiwanych)!

Ostatnim krokiem jaki nam teraz pozostał jest optymalizacja parametrów funkcji **f**, tak by zminimalizować **L**. Istnieje wiele technik minimalizacji, my skorzystamy z jednej z najpopularniejszych: stochastycznego spadku po gradiencie. Nie zagłębiając się w szczegóły (choć warto to zrobić, jeżeli jest się osobą mocniej zainteresowaną tematem) sprowadza się on do poruszania w przestrzeni parametrów małymi krokami w kierunku największego spadku (gradientu), tak jak na tym filmie: https://www.youtube.com/watch?v=kJgx2RcJKZY. Gdyby **L(f(θ))** miała tylko jeden parametr dobrze ilustrowałby to również poniższy przykład.

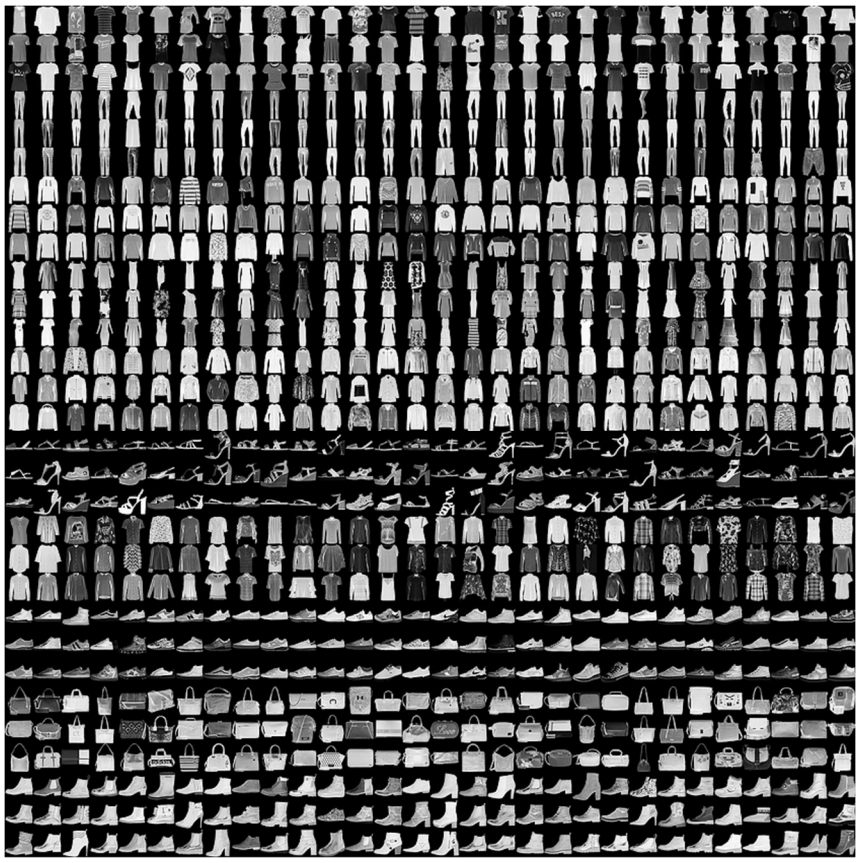


Ponieważ nie wiemy jaki zestaw parametrów jest tym właściwym, optymalizację zaczniemy od ich losowego lub pseudo-losowego wyboru (w praktycznych realizacjach jest to nieco bardziej złożone, niepotrzebnie komplikowałoby jednak przykład). Stała **η** używana na rysunku powyżej określa tzw. tempo uczenia się (learning rate) – długość kroków jakie wykonujemy szukając optimum.

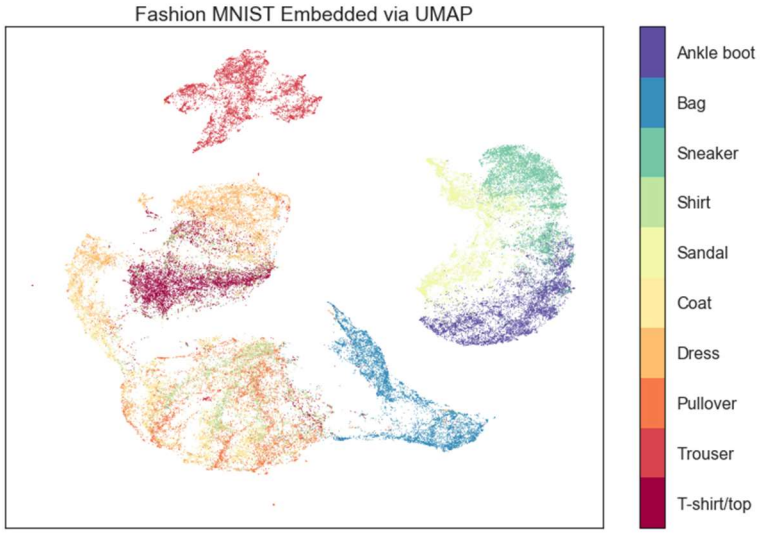
Dlaczego stochastyczny? Policzenie dokładnej wartości **L** wymagałaby wzięcia pod uwagę wszystkich przykładów ze zbioru treningowego – byłby to proces kosztowny obliczeniowo. Zamiast tego korzystamy z przybliżonej wartości **L** (i jej pochodnych) obliczonej na bazie pewnego ich podzbioru, zwanego *batch*. Takie przybliżanie może być obarczone pewnym losowym błędem – a więc metoda będzie zmierzać w tym samym kierunku, ale potencjalnie lekko błądząc.

ĆWICZENIE 1 – LINIOWY KLASYFIKATOR UBRAŃ

Wymienione wyżej składowe (funkcja liniowa opakowana w funkcję softmax, funkcja entropii krzyżowej i stochastyczny spadek po gradiencie) pozwolą nam na nauczenie naszego pierwszego klasyfikatora! Skocz za chwilę do załączonego pliku .py i wykonaj intro oraz pierwsze ćwiczenie. Nasza (na razie jednowarstwowa) sieć będzie się uczyć rozpoznawania kilku rodzajów ubrań ze zbioru FashionMNIST (opartego o asortyment z serwisu Zalando) – jego przykładową zawartość możemy zobaczyć poniżej. Większość jest łatwa od odróżnienia od pozostałych, ale niektóre mogą być zwodnicze (gdzie przebiega granica między długim topem, a krótką sukienką?).



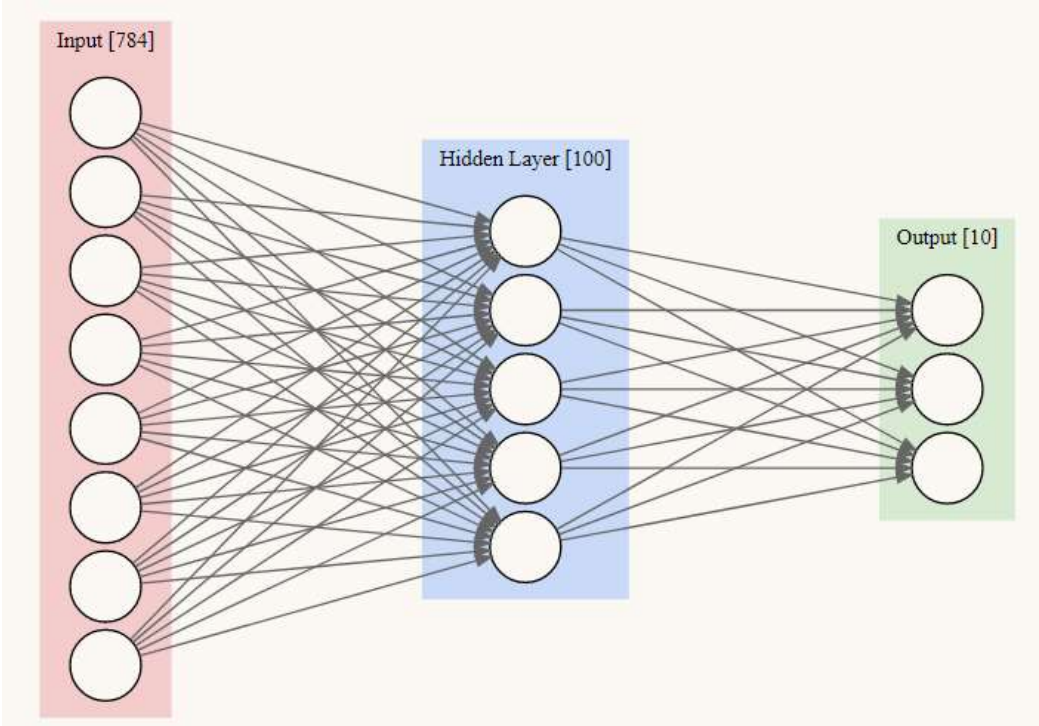
Możemy też obejrzeć jego wizualizację 2D z użyciem algorytmu UMAP. Widać tam kilka ciekawych zjawisk – 3 rodzaje butów tworzą osobną wyspę, torebki i spodnie zajmują swoje własne unikatowe regiony, zaś reszta jest już w znacznym stopniu przemieszana.



Jeżeli wszystkie kroki wykonano poprawnie, to taki prosty klasyfikator powinien uzyskać zupełnie przyzwoitą skuteczność rozpoznawania ubrań (a na pewno znacznie większą niż losowe zgadywanie jednej kategorii z dziesięciu).

ĆWICZENIE 2 – PROSTA SIEĆ NEURONOWA

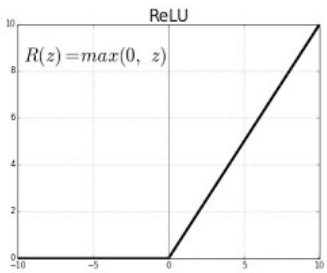
Wynik z poprzedniego ćwiczenia pozostawia wiele do życzenia… Jak go poprawić? Po pierwsze zamienimy naszą jednowarstwową liniową sieć w taką, która posiada ciekawszą, bardziej złożoną architekturę. Jaką? Taką jak na schemacie poniżej.



Zauważmy, że pojawiła się nowa warstwa – tzw. warstwa ukryta. Każdy z widocznych powyżej neuronów to nadal funkcja liniowa, wraz ze swoimi wagami i biasem. Różnica polega na tym, że neurony końcowe nie korzystają z danych wejściowych, a z tego co zwraca warstwa poprzednia.

Choć wydaje się, że taka sieć powinna mieć znacznie większą siłę wyrazu…to tak naprawdę nic jeszcze nie uzyskaliśmy! Gdyby korzystając z zasad algebry liniowej uprościć wykonywane operacje okazałoby się, że powyższa sieć jest dokładnie równoważna takiej zwykłej, jednowarstwowej (przemnożone przez siebie macierze wag dają macierz wynikową).

By skorzystać z dodatkowej warstwy sieci i zwiększyć jej siłę wyrazu musimy znów zmodyfikować wyniki funkcji liniowej i przetworzyć je przez inną, nieliniową funkcję. Funkcję tą zwiemy zwyczajowo funkcją aktywacji. Popularnym przykładem takiej funkcji jest funkcja ReLU, o przebiegu pokazanym poniżej.



Nasza sieć będzie się więc zachowywać następująco:

- bierze wektor wejściowy **x**
- przemnaża go przez funkcje liniowe pierwszej warstwy $\mathbf{y_1 = W_1 x + b_1}$

- wynik ten przetwarza nieliniową funkcją ReLU $\mathbf{h_1 = max(0, y_1)}$
- te wyniki przechodzą przez funkcje liniowe warstwy drugiej $\mathbf{y_2 = W_2x + b_2}$
- na końcu zaś są zamieniane w prawdopodobieństwa z użyciem funkcji softmax $\mathbf{s = \sigma(y_2)}$

Zmodyfikuj kod z ćwiczenia drugiego tak, aby spełniał powyższe założenia i sprawdź wyniki.

ĆWICZENIE 3 – ROZWIĄZUJEMY PROBLEM Z AKTYWACJĄ

Uzyskany wynik okazał się być niedorzecznie niski, bliski 10% (odpowiednik losowego zgadywania)? To normalne! Co właściwie poszło nie tak? Wszystkie wagi i biasy w naszej sieci były inicjalizowane wartością 0. W przypadku dwuwarstwowej sieci z funkcją ReLU takie wartości zostały również przestałe na kolejną warstwę (**max(0, 0) = 0**). Ta sytuacja uniemożliwiła sieci wykorzystywanie algorytmu stochastycznego spadku po gradiencie (druga warstwa otrzymywała zawsze takie samo zerowe wejście, niemożliwe było policzenie odpowiednich pochodnych), a więc efektywnie zablokowany został proces uczenia.

Aby rozwiązać ten problem należy zmienić startowe wagi i biasy tak, by zamiast 0 rozpoczynały od losowych niewielkich wartości (np. między -0.1 a 0.1). Udaj się do ćwiczenia 3 i dokonaj tej modyfikacji. Teraz wynik powinien być lepszy niż kiedykolwiek!

PS. Wszystkie uzyskane wyniki mogą być nieco rozczarowujące – w końcu daleko nam do pełnej poprawności. Po pierwsze – to nie jest bardzo łatwy zbiór. Powstał on w 2017 roku jako trudniejsza i bardziej różnorodna alternatywa dla MNISTa (zawierającego ręcznie pisane cyfry). Po drugie (i ważniejsze) – w naszym podejściu niszczymy jedną z zawartych w każdej fotografii kluczowych informacji: o tym jak piksele są położone względem siebie (w trakcie wstępnej obróbki zdjęcia zamieniane są w podłużny wektor). By brać ją pod uwagę musimy skorzystać z dedykowanego fotografiom rozwiązania, np. sieci z warstwami konwolucyjnymi – ale to już temat na osobny tutorial.

ĆWICZENIE 4 – CZY MUSI BYĆ TAK CIĘŻKO?

Wszystkie dotychczasowe ćwiczenia wykonywaliśmy z użyciem biblioteki TensorFlow (wiodącego frameworku do wydajnego wykonywania operacji tensorowych). Pozwala on na pełną kontrolę wszystkich szczegółów danego procesu obliczeniowego – konsekwencją jest jednakże obszerny kod, w którym łatwo o błędy (konieczność dbania o odpowiednie wymiary tensorów i ich zawartość, efekt znany dopiero po zbudowaniu całego grafu obliczeń, niekiedy długie oczekiwanie na wyniki).

W wielu sytuacjach jest to klasyczne „wynajdywanie koła na nowo”. Na szczęście obecnie istnieją już także dobrze zaprojektowane biblioteki operujące na znacznie wyższym poziomie abstrakcji. Jedną z najpopularniejszych i uznawanych za najbardziej intuicyjne jest Keras – od niedawna będący również oficjalnym wysokopoziomowym API dla samego TensorFlow (choć wspierający również inne backendy).

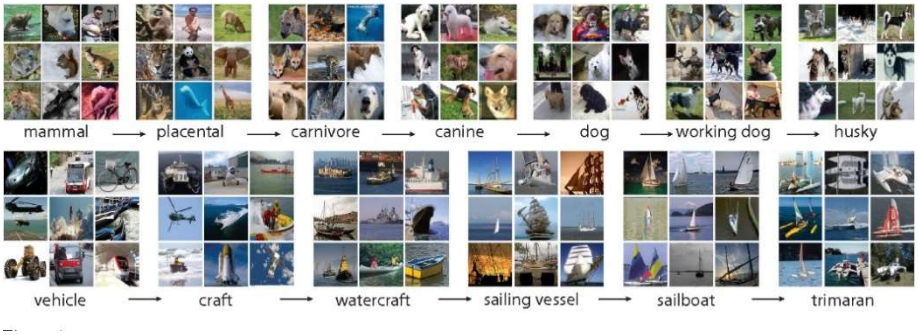
Naszym zadaniem jest skorzystać z zamieszczonych w komentarzach wskazówek i odtworzyć w Kerasie sieć zaimplementowaną wcześniej w ćwiczeniu 3. Tym razem powinno pójść o wiele łatwiej!

ĆWICZENIE 5 – WYKORZYSTYWANIE JUŻ WYTRENOWANYCH SIECI

Zakres tego laboratorium jest ograniczony – więc powyższe kroki były tylko powierzchownym wstępem do sieci neuronowych i ich architektury. Współczesne rozwiązania korzystają z dziesiątek warstw o różnych sposobach działania, są pretrenowane, odwołują się do siebie rekurencyjnie i wdrażają wiele innych pomysłów, które pojawiły się na przestrzeni ostatnich dekad.

Nasz (tymczasowy) brak wiedzy (i co niestety równie istotne - mocy obliczeniowej) nie oznacza jednak, że nie możemy w ogóle korzystać z ich dobrodziejstw. Ostatnie ćwiczenie pokazuje jak pobrać dużą, już wyuczoną sieć

rozpoznającą elementy zbioru ImageNet (przykłady na zdjęciu poniżej). Sprawdź jej skuteczność w praktyce! Wykonaj polecenia zawarte w komentarzach.



Kolejne etapy tego ćwiczenia pozwalają (orientacyjnie) zaznajomić się z jej budową i działaniem. Będą też świetnym wstępem przez kolejnym rozdziałem – ~~zadaniem domowym~~ dalszym pogłębianiem wiedzy na własną rękę. W tym roku laboratorium jest nieco okrojone, ale nic straconego – Internet jest pełen materiałów, a wysokopoziomowe biblioteki obniżają próg wejścia i czynią przygodę z sieciami neuronowymi dostępną dla każdego.

CHCESZ WIĘCEJ?

Zacznij przerabiać dostępne w sieci tutoriale i czytać artkuły (dobrym punktem wyjścia jest choćby https://keras.io/examples/)! Uczenie maszynowe oparte o sieci neuronowe jest obecnie niezwykle popularnym tematem – w źródłach wiedzy można wręcz przebierać. Jeżeli brakuje ci mocy obliczeniowej, to pomocny może być serwis Google Colab. A jeżeli chcemy być na bieżąco z najnowszymi odkryciami, to warto śledzić https://paperswithcode.com/.

CO DO DOMU?

Zrób wszystkie 5 etapów ćwiczenia (testy powinny przechodzić). W ostatnim odpowiedz na pytania w krótkim komentarzu w kodzie. Wrzuć uzupełniony plik na platformę UPEL – zadanie jest oceniane binarnie na zaliczenie/brak zaliczenia.