Przegląd literatury - uczenie maszynowe

Rafał Smolak (252973)
Hubert Gabory (255743)

Kwiecień 2023

1 Sieć neuronowa

Pierwszy model neuronu był zaprojektowany przez McCulloch-Pitts'a. Model ten polega na sumowniu i wejść w postaci x_i przemnożonych przez wagi w postaci w_i. Oprócz wejść dodawany jest jeszcze tzw. *bias*, który jest specjalnym rodzajem wejścia wynoszącym zawsze 1, ale jest również opatrzony swoją wagą, która również może ulegać zmianie. Traktując bias jako wejście dostajemy:

$$\sum_{i=1}^{n} w_i x_i + b = \sum_{i=0}^{n} w_i x_i = \mathbf{w}^T \mathbf{x}, \tag{1}$$

gdzie $\mathbf{w}^T$ oznacza wektor wag transponowany, $\mathbf{x}$ to wektor wejściowy, b to bias, a n to liczba wejść. Następnie wartość wynikowa przekazywana jest do funkcji aktywacji ϕ. Oryginalnie McCulloch-Pitts użył funkcji skokowej *(b)*, ale w dzisiejszych czasach bardziej popularna jest funkcja sigmoidalna *(d)* lub ReLU *(c)*.

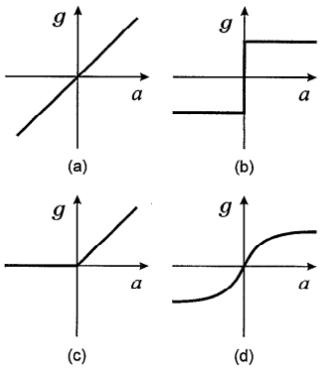

Figure 1: Przykładowe funkcje aktywacji. [1]

W procesie nauczania neuronu, ważne jest obliczenie błędu, który posłuży nam do obliczenia korekty wag, tak aby odpowiedź neuronu była bliższa wartości oczekiwanej. Zakładając, że sieć uczy się na wszystkich q punktach danych, otrzymujemy następujący wzór:

$$E(w) = \frac{1}{2} \sum_{q=1}^{n} (t_q - y(x_q; w))^2 \tag{2}$$

gdzie n to liczba punktów danych w zbiorze treningowym, t_q to wartość docelowa dla k-tej jednostki wyjściowej dla punktu danych o indeksie q, a $y(x_q; w)$ to wartość przewidywana przez k-tą jednostkę wyjściową dla wektora wejściowego x_q. Celem treningu sieci neuronowej jest minimalizacja wartości funkcji błędu $E(w)$ poprzez optymalizację wag sieci w.

Zazwyczaj sieć neuronowa składa się z więcej niż jednej warstwy. Wyjście sieci jednowarstwowej można przedstawić w następujący sposób:

$$z_j = g\left(\sum_{i=0}^{m} w_{ji} x_i\right) \tag{3}$$

W sieci wielowarstwowej, wyjścia jednej warstwy są wejściami drugiej warstwy, Dla sieci dwuwarstwowej mamy:

$$y_k = g\left(\sum_{j=0}^{n} w_{kj}^{(2)} z_j\right) = g\left(\sum_{j=0}^{n} w_{kj}^{(2)} g\left(\sum_{i=0}^{m} w_{ji}^{(1)} x_i\right)\right) \tag{4}$$

Warstwy sieci znajdujące się pomiędzy warstwą wejściową i warstwą wyjściową nazywają się warstwami ukrytymi. Tworzenie wielowarstwowych sieci neuronowych ma sens jedynie w przypadku, gdy zastosujemy funkcje nieliniowe jako funkcje aktywacji g. W przeciwnym wypadku cała sieć będzie dała się redukować do jednej warstwy, ponieważ będzie mogła być reprezentowana jako macierz.

Jako, że w procesie uczenia sieci neuronowej będziemy korzystać z różniczkowania, ważne aby wybrana przez nas funkcja aktywacji była łatwo różniczkowalna. Dlatego najczęściej wybierana jest sigmoida:

$$g(a) = \frac{1}{1 + e^{-a}} \tag{5}$$

Której pochodna to:

$$g'(a) = g(a)(1 - g(a)) \tag{6}$$

Proces uczenia się sieci neuronowej polega na minimalizacji funkcji błędu, takiej jak przytoczona w równaniu 2. Funkcja błędu określa jak bardzo wyjście sieci różni się od jego wejścia. Wiele algorytmów uczenia stosuje pochodne funkcji błędu po wagach sieci. Wektor takich pochodnych nazywamy gradientem. Takie podejście pozwala nam określić, w którą stronę powinniśmy zmieniać wagi sieci aby zbliżyć się do minimum (niekoniecznie globalnego). Aby wyznaczyć błędy wszystkich warstw sieci będziemy musieli je przekazywać wstecz od wyjścia sieci w stronę wejścia. Taki proces nazywa się *backpropagation*.

Wzór na pochodną złożoną dla błędu względem wag sieci dla warstwy końcwej, wygląda następująco:

$$\frac{\partial E}{\partial w_{jk}} = \frac{\partial E}{\partial y_k} \cdot \frac{\partial y_k}{\partial z_k} \cdot \frac{\partial z_k}{\partial w_{jk}} \tag{7}$$

Pochodne cząstkowe można wyliczyć ze wzorów:
$\frac{\partial E}{\partial y_k} = (y_k - t_k)$ - pochodna funkcji kosztu po przewidywanej wartości wyjściowej k-tego neuronu w warstwie wyjściowej.
$\frac{\partial y_k}{\partial z_k} = g'(z_k)$ - pochodna funkcji aktywacji $f(z)$, stosowanej na sumie ważonej z_k dla k-tego neuronu w warstwie wyjściowej.
$\frac{\partial z_k}{\partial w_{jk}} = y_j$ - wejście do k-tego neuronu w warstwie wyjściowej, które jest wagą w_{jk} pomnożoną przez wyjście j-tego neuronu w warstwie ukrytej.

Widzimy, że ostatnia pochodna cząstkowa jest zależna od wyjścia warstwy ukrytej, to wyjście z kolei jest zależne od wag warstwy ukrytej, itd. aż do samego początku sieci. Oznacza to, że aby wyliczyć pochodną dla każdej warstwy sieci, musimy zastosować regułę łańcuchową dla całej sieci.

Dla n-warstwowej sieci, wzór na pochodną złożoną dla błędu względem wag sieci dla warstwy końcowej można wyrazić jako iloczyn pochodnych cząstkowych dla każdej warstwy pośredniej i warstwy wyjściowej:

$$\frac{\partial E}{\partial w_{ij}^{(1)}} = \frac{\partial E}{\partial y_j^{(n)}} \cdot \frac{\partial y_j^{(n)}}{\partial z_j^{(n)}} \cdot \frac{\partial z_j^{(n)}}{\partial y_i^{(n-1)}} \cdot \frac{\partial y_i^{(n-1)}}{\partial z_i^{(n-1)}} \cdot \frac{\partial z_i^{(n-1)}}{\partial w_{ij}^{(n-1)}} \cdot \ldots \cdot \frac{\partial z_j^{(2)}}{\partial y_i^{(1)}} \cdot \frac{\partial y_i^{(1)}}{\partial z_i^{(1)}} \cdot \frac{\partial z_i^{(1)}}{\partial w_{ij}^{(1)}} \tag{8}$$

Aktualizacja wag sieci odbywa się za pomocą zejścia z gradientu i jest określona wzorem:

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}} \tag{9}$$

gdzie η jest współczynnikiem uczenia, czyli miarą o jaką zmienia się waga w każdym przejściu sieci.

Usprawnieniem jakie można wprowadzić do sieci jest, tzw. momentum, które modyfikuje stopień zmiany wagi w zależności od wielkości poprzedniej zmiany.

$$\Delta w_{ij}^{(\tau)} = -\eta \frac{\partial E}{\partial w_{ij}} + \alpha \Delta w_{ij}^{(\tau-1)} \tag{10}$$

gdzie τ to krok czasowy a α to współczynnik momentum.

Wzór na wyznaczenie nowej wagi prezentuje się następująco:

$$w_{ij}^{(\tau+1)} = w_{ij}^{(\tau)} + \Delta w_{ij}^{(\tau)} = w_{ij}^{(\tau)} - \eta \frac{\partial E}{\partial w_{ij}} + \alpha \Delta w_{ij}^{(\tau-1)} \tag{11}$$

W procesie uczenia sieci neuronowej, zbiór danych wejściowych dzielimy na dwie grupy - zbiór treningowy i zbiór testowy. Na zbiorze treningowym wyliczamy nasze wagi sieci, a zbiór testowy służy nam do sprawdzenia, czy nasza sieć jest dobrze wytrenowana. Ważnym aspektem w uczeniu sieci neuronowej jest uniknięcie przetrenowania, czyli stanu, w którym wagi sieci zbyt dobrze dopasują się do zbioru treningowego przez co stają się gorzej dopasowane do zbioru testowego. Do względnie obiektywnego sprawdzenia wydajności naszej sieci można zastosować podejście *cross-validation*, które dzieli zbiór na S równych części, z czego jedna jest zbiorem testowym a reszta treningowym. Po przetrenowaniu sieci bierzemy kolejną część jako zbiór testowy, a resztę traktujemy jako treningowy. Proces powtarzamy dla wszystkich S części, a na koniec wyciągamy średnią[1].

2 Kernel Density Estymation (KDE)

KDE to metoda na tworzenie przestrzeni gęstości występowania próbek w zbiorze. Charakteryzuje się ona tym, że na każdą próbkę w zbiorze jest nakładany kernel(jądro), które jest pewnego rodzaju funkcją, np. rozkładem Gaussa. Estymowany rozkład gęstościowy metodą KDE jest dany wzorem:

$$KDE(x) = \frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right) \tag{12}$$

gdzie:

- x to punkt, w którym szacowana jest gęstość
- n to liczba próbek
- h parametr wygładzania (smoothing parameter)
- K to funkcja jądra (kernel function)
- x_i reprezentuje każdą próbkę w zbiorze danych

Funkcja jądra może przyjmować wiele postaci takich jak rozkład gaussa, funkcja trójkątna, czy funkcja prostokątna. Parametr wygładzenia określa, jak bardzo funkcja KDE jest wygładzona. O ile sam wybór funkcji jądra nie ma bardzo dużego wpływu na skuteczność KDE, tak wybór parametru wygładzania jest kluczowy. Zbyt mała wartość parametru, może spowodować skupianie się na nieistotnych szczegółach, za to zbyt duża może spowodować utratę informacji.

Jest kilka sposobów pomiaru błędu między rzeczywistym rozkładem prawdopodobieństwa opartym na gęstości, a estymowanym. Pierwszym jest mean squared error (MSE).

$$MSE_x(\hat{f}) = E\left[\left(\hat{f}(x) - f(x)\right)^2\right] = Var(\hat{f}(x)) + Bias^2(\hat{f}(x), f(x)) \tag{13}$$

, ale są również inne takie jak MISE i ISE.

$$MISE = \int MSE_x(\hat{f}) dx \tag{14}$$

$$ISE = \int \left[\hat{f}(x) - f(x)\right]^2 w(x)dx \tag{15}$$

Jest kilka metod na wyliczanie parametru wygładzania. Jedną z nich jest zasada kciuka (rule of thumb) spopularyzowana przez Silvermana. Metoda ta bazuje na liczeniu błędu AMISE, który rozszerza metodę MISE w szereg Taylora i obcina niepotrzebne fragmenty.

Według tej metody parametr wygładzania jest równy

$$h = 1.06 \cdot \hat{\sigma} \cdot n^{-1/5}$$

gdzie $\hat{\sigma}$ jest standardowym odchyleniem próbki. W celu zminimalizowania wpływu wartości skrajnych stosuje się IQR (inquartile range - zakres kwartylowy). Silverman proponuje kompromis pomiędzy dwoma podejściami w postaci[8]

$$h = 0.9 \cdot min\left(\hat{\sigma}, \frac{IQR}{1.34}\right) \cdot n^{-1/5}$$

3 Problem niezbalansowanych danych

Niezbalansowane dane to takie, w których jedna albo kilka klas posiada zdecydowanie mniej próbek danych niż pozostałe. Stanowi to pewien problem, ponieważ klasyfikatory przyporządkowujące etykiety do klasy w procesie uczenia mogą się nauczyć, błędnie przyporządkowywać etykiety dla mniej licznych klas. Innymi słowy, ich wpływ na uczenie klasyfikatora będzie taki mały, że nauczy się on pomijać takie próbki.

Jest dużo podejść, które próbują naprawiać problem niezbalansowanych danych. Do najprostszych koncepcyjnie można zaliczyć oversampling i undersampling. Oversampling polega na utworzeniu dodatkowych próbek dla klas o mniejszej ilości danych. Są różne podejścia do tworzenia próbek, takie jak random oversampling, czyli duplikowanie losowych próbek, czy SMOTE, czyli technika tworzenia nowych próbek na bazie dwóch innych z tej samej klasy. Z drugiej strony mamy technikę undersampling-u, która polega na zmniejszeniu ilości próek klas większościowych.

Inną metodą jest wpływanie na funkcję strat w taki sposób, aby poprawić wyniki uczenia. Wgłębiając się w szczegóły dowiemy się, że jednym z problemów, przez który klasyfikatory słabo sobie radzą z danymi niezbalansowanymi jest zła konstrukcja funkcji strat, tzn. nieuwzględniająca ilości danych w batch'u. Przez to mimo, że jedna z klas jest zawsze źle etykietowana funkcja strat i tak będzie niewielka, ponieważ bierze liczy ona średnią ze wszystkich błędów kwadratowych. Jednym z podejść jest tzw. *cost sensitive learning*, czyli funkcja, która określa koszty błędnych klasyfikacji w klasie. Oznacza to przesunięcie granicy decyzyjnej w jedną ze stron, mimo zmniejszenia ogólnej skuteczności sieci. Dobrym przykładem jest przesunięcie granicy tak by była bardziej wrażliwą na pacjentów chorych. Warto jest wykryć każdy przypadek choroby, nawet kosztem zakwalifikowania niektórych zdrowych pacjentów jako chorych.

Autorzy artykułu zdecydowali się na zbadanie autorskiej metody radzenia sobie z niezbalansowanymi danymi dla problemu klasyfikacji binarnej, poprzez zmianę funkcji kosztu. Nazwali swoją metodę MFE(Mean False Error). Metoda ta modyfikuje funkcję strat w taki sposób, aby była bardziej wrażliwa na mniejszościowe dane. MFE prezentuje się wzorem:

$$MFE = FPE + FNE = \frac{1}{N}\sum_{i=1}^{N}\sum_{n}\frac{1}{2}(d_n^{(i)} - y_n^{(i)})^2 + \frac{1}{P}\sum_{i=1}^{P}\sum_{n}\frac{1}{2}(d_n^{(i)} - y_n^{(i)})^2 \tag{16}$$

gdzie N i P to liczba próbek w klasach - negatywnej i pozytywnej. Powyższa metoda, według autorów, daje się ulepszyć poprzez jej kwadratowanie, na wskutek czego powstała ulepszona metoda MSFE, która defniuje się wzorem:

$$MSFE = FPE^2 + FNE^2 \tag{17}$$

W pracy wykazano, że taka modyfikacja funkcji celu znacząco poprawia wyniki sieci neuronowych dla danych niezbalansowanych dla metryk: f-measure i AUC.

Ponadto autorzy kilkukrotnie wspominają w pracy, że jest jeszcze wiele niezbadanych aspektów sieci neuronowych jeżeli chodzi o dane niezbalansowane, co rzuca zielone światło na problem, który będziemy badać[7].

4 Próbkowanie KDE dla niezbalansowanych zbiorów

W artykule autor prezentuje nową metodę oversamplingu opartą na SGD. Metody oversamplingu to takie, które radzą sobie z problemem danych niezbalansowanych poprzez dodawanie nowych próbek dla klas mniejszościowych. Głównym celem artykułu było porównanie tej metody oversamplingu do innych, już istniejących. Oprócz nowej metody oversamplingu opardej na KDE (kernel density estimation) autor opisuje też trzy inne metody, z którymi tę metodę porównywał. Oprócz tego metodę porównano do jednej z metod undersamplingu. Undersampling jest odwrotnością oversamplingu, co oznacza, że zamiast dodawać próbki do klas mniejszościowych to je zabieramy z klas większościowych.

Pierwszą metodą opisaną przez autora jest metoda undersamplingu – NearMiss. W tej metodzie próbki są wybierane ze zbioru negatywnego(zakładając, że rozważamy problem klasyfikacji binarnej) w taki sposób aby średnia odległość do k najbliższych próbek zbioru pozytywnego była jak najmniejsza. Mimo, że takie podejście często daje dobre rezultaty to jednak undersampling zawsze prowadzi do utraty informacji, co jest niepożądane. Drugą metodą jest random sampling, która losowo wybiera punkty ze zbioru mniejszościowego i je duplikuje do momentu wyrównania liczności zbiorów. Takie podejście może prowadzić do overfitting'u czyli zbytniego dopasowania klasyfikatora do zbioru. Kolejną metodą opisaną przez autora jest SMOTE. Jest to metoda oversamplingu, która losuje punkt, będący nową próbką na linii wyznaczonej przez dwie inne próbki ze zbioru mniejszościowego. Ostatnim omówionym algorytmem oversamplingu jest modyfikacja SMOTE'a czyli ADASYN. Metoda ta tworzy nowe próbki w okolicy granicy między dwoma klasami.

Metoda tworzenia próbek opisana w artykule polega na oszacowaniu gęstości klasy mniejszościowej bazując na dostępnych próbkach. Gęstość klasy będzie szacowana na podstawie metody KDE. Samą metodę KDE opisaliśmy już bazując na jednym z innych artykułów, dlatego w tym artykule skupimy się jedynie na informacjach, które nie były tam zawarte oraz na metodach wybranych przez autora artykułu. W celu optymalizacji parametru wygładzania (h – smoothness parameter) należy zoptymalizować całkowity błąd kwadratowy MISE.

$$MISE_n(h) = \frac{1}{n}\sum_{i=1}^{n}(\hat{f(x_i} - f(x_i))^2 \tag{18}$$

Jako, że błąd ten nie może być wyliczony ze wzoru z powodu nieznajomości rzeczywistej gęstości klasy to zastosowano metody przybliżające. Pierwszą z nich jest reguła kciuka, którą opisaliśmy wcześniej, drugą jest cross-validation, czyli metoda próbująca minimalizować wartość błędu MISE poprzez testowanie go dla różnych wartości parametru h.

KDE dla wielu zmiennych jest bardzo podobne do KDE dla jednej zmiennej i jest dane następującym wzorem:

$$\hat{f}_H = \frac{1}{n}\sum_{i=1}^{n} K_H(x - x_i) \tag{19}$$

Gdzie H jest macierzą o rozmiarach d x d. Autorzy do wyznaczenia wartości H użyli reguły kciuka Scott'a dla problemu wielowymiarowego. Użyto następującego wzoru:

$$H = n^{-\frac{1}{d+4}} \cdot S \tag{20}$$

Gdzie S jest macierzą kowariancji danych. Dla wyznaczenia jądra użyto specjalnego wzoru rozkładu normalnego dla problemu wielowymiarowego:

$$K_H(x) = \frac{1}{\sqrt{(2\pi)^d |H|}} e^{-\frac{1}{2}x^T H^{-1} x} \tag{21}$$

Jako, że zwykły pomiar skuteczności(accuracy) nie ma za bardzo sensu dla danych niezbalansowanych, ponieważ klasy większościowe będą go sztucznie zawyżały, do porównania metod użyto miar G-mean i F1 – score. Prezentują się one następującymi wzorami:

$$G - mean = \sqrt{\frac{TP}{TP + FN} \cdot \frac{TN}{TN + FP}} \tag{22}$$

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{23}$$

Rezultaty eksperymentu były bardzo obiecujące. Dla dwóch z trzech klasyfikatorów nowa metoda okazała się dawać najlepsze rezultaty. Były to klasyfikatory KNN i SVM. Klasyfikatorem, który poradził sobie gorzej był MLP czyli perceptron wielowarstwowy, jednak warto zauważyć, że dla niektórych zbiorów danych przetestowanych w artykule poradził on sobie najlepiej. Z badania można wywnioskować, że metoda KDE może być bardzo przydatna do rozwiązywania problemu niezbalansowanych danych, co jest dobrą informacją zważywszy na temat, który realizujemy[4].

5 Paczki i Epoki (ang. Batches and Epochs)

5.1 SGD

Stochastic Gradient Descent (SGD) jest używany do trenowania algorytmów uczenia maszynowego, szczególnie sztucznych sieci neuronowych w głębokim uczeniu. Celem algorytmu jest znalezienie wewnętrznych parametrów modelu, które dają dobre wyniki dla pewnej miary wydajności. Algorytm jest iteracyjny i polega na wielu krokach, w których wykorzystuje się model z aktualnymi parametrami, aby dokonać przewidywań i porównać je z rzeczywistymi wynikami [3].

5.2 Paczki (ang. Batch)

Batch size to hiperparametr, który definiuje liczbę próbek, przez które algorytm np. SGD ma przejść przed zaktualizowaniem wewnętrznych parametrów modelu. Rozmiar jednej paczki (batch) jest określony przez liczbę próbek w jednym przejściu pętli. Po wykonaniu wszystkich iteracji w jednej paczce obliczane jest w tym miejscu spadek funkcji kosztu i używany jest on do zaktualizowania wewnętrznych parametrów modelu. Dzięki temu procesowi algorytm iteracyjnie zmniejsza wartość funkcji kosztu i poprawia jakość predykcji.

Zbiór treningowy może być podzielony na jedną lub wiele paczek. Gdy wszystkie próbki treningowe są używane do jednej paczki, algorytm uczenia nazywany jest metodą batch gradient descent. Gdy paczka składa się z jednej próbki, algorytm nazywamy metodą stochastycznego spadku gradientu (SGD). Gdy rozmiar paczki wynosi więcej niż jedna próbka i mniej niż rozmiar zestawu treningowego, algorytm nazywany jest metodą mini-batch gradient descent.

Dla popularnych rozmiarów paczek w metodzie mini-batch gradient descent stosowane są wartości 32, 64 i 128 próbek. Często zdarza się, że rozmiar zestawu treningowego nie dzieli się równomiernie przez rozmiar paczki. W takim przypadku ostatnia paczka zawiera mniej próbek niż pozostałe paczki. Można wtedy usunąć niektóre próbki ze zbioru danych lub zmienić rozmiar paczki tak, aby liczba próbek w zbiorze dzieliła się równomiernie przez rozmiar paczki [3].

5.3 Epoki (ang. Epoch)

Liczba epok określa, ile razy algorytm uczenia maszynowego będzie pracował na całym zbiorze treningowym. Jedna epoka oznacza, że każdy próbka w zbiorze treningowym ma okazję zaktualizować wewnętrzne parametry modelu. Epoka składa się z jednej lub więcej partii. Liczba partii określa liczbę próbek, na których algorytm uczenia maszynowego będzie działał przed aktualizacją wewnętrznych parametrów modelu. Wartość ta jest nazywana rozmiarem partii (batch size). Liczba epok jest tradycyjnie duża i często wynosi setki lub tysiące, pozwalając algorytmowi na pracę do momentu, w którym błąd modelu zostanie dostatecznie zminimalizowany. W celu zdiagnozowania, czy model jest zbyt dobrze dopasowany, zbyt słabo dopasowany czy właściwie dopasowany do zbioru treningowego, często stosuje się wykresy zwane krzywymi uczenia (learning curves), które pokazują zależność między epokami a błędem modelu lub jego skutecznością [3].

5.4 Podsumowanie

Rozmiar partii określa liczbę próbek, które przetwarzane są przed aktualizacją modelu, podczas gdy liczba epok to liczba przejść przez cały zestaw danych uczących. Oba te parametry są ustalane przez użytkownika

i wpływają na wyniki uczenia się modelu. Nie ma jednej "magicznej" wartości, która działa dobrze dla każdego problemu - należy eksperymentować z różnymi wartościami, aby znaleźć optymalne parametry [3].

6 Batch Normalization

6.1 Definicja

Batch Normalization (BN) to technika normalizacji danych wejściowych do warstw sieci neuronowych w celu przyspieszenia procesu uczenia oraz poprawy jakości predykcji modelu.

W przypadku sieci neuronowych, normalizacja polega na przeskalowaniu wartości wejściowych tak, aby miały one średnią wartość zero i stałą wariancję. Dzięki temu sieć jest w stanie efektywniej nauczyć się odpowiednich wag, co przekłada się na lepszą jakość predykcji.

Batch Normalization działa w sposób podobny, jednak zamiast normalizować tylko dane wejściowe do sieci, normalizuje także wartości pośrednie w poszczególnych warstwach sieci. W każdej iteracji trenowania modelu, wartości z każdej warstwy są normalizowane względem średniej i wariancji tego batcha. W ten sposób, sieć nie jest wrażliwa na zmiany wartości wejściowych, co pozwala na przyspieszenie procesu uczenia i zwiększenie jego stabilności [2].

6.2 Algorytm Batch Normalization

Wejściem i wyjściem warstwy BN są czterowymiarowe tensory $I_{b,c,x,y}$ i $O_{b,c,x,y}$, gdzie wymiary odpowiadają przykładom w partii b, kanałowi c oraz dwóm wymiarom przestrzennym x, y. Dla obrazów wejściowych, kanały odpowiadają kanałom RGB. BN stosuje tę samą normalizację dla wszystkich aktywacji w danym kanale. Wynik normalizacji $O_{b,c,x,y}$ obliczamy ze wzoru:

$$O_{b,c,x,y} \leftarrow \gamma_c \frac{I_{b,c,x,y} - \mu_c}{\sqrt{\sigma_2^2 + \epsilon}} + \beta_c \ \ \forall b, c, x, y.$$

BN odejmuje od wszystkich aktywacji wejściowych w kanale c średnią wartość $\mu_c = \frac{1}{|B|} \sum_{b,x,y} I_{b,c,x,y}$, gdzie B zawiera wszystkie aktywacje w kanale c dla wszystkich cech b w całej partii oraz wszystkich lokalizacjach przestrzennych x, y. Następnie BN dzieli skośność aktywacji przez odchylenie standardowe σ_c (dodając ϵ dla stabilności numerycznej), które jest obliczane analogicznie. Podczas testowania używane są bieżące średnie wartości i wariancje. Normalizacja jest następnie poddana przekształceniu afinicznemu kanałów parametryzowanemu za pomocą γ_c, β_c, które są uczone w trakcie procesu uczenia [2].

6.3 Korzyści z BN

W artykule [2] autorzy badają wpływ Batch Normalization (BN) na szybkość uczenia i dokładność modelu. Bez BN nauka modelu wymaga znacznie mniejszej wartości współczynnika uczenia ($alpha = 0,0001$) oraz trwa dłużej (ok. 2400 epok) w porównaniu do modelu z BN, który może być nauczony z wyższym $\alpha = 0,1$ i w krótszym czasie. Aby zbadać, które korzyści są związane z zastosowaniem BN, autorzy trenują sieci z BN przy użyciu wartości α i liczby epok z modelu bez normalizacji. Okazuje się, że sieci te osiągają podobną dokładność jak nieznormalizowany model, a różnica między wynikami na zbiorze treningowym a testowym jest znacznie większa. Oznacza to, że to wyższa wartość współczynnika uczenia, którą umożliwia BN, decyduje o większości jej korzyści, takich jak poprawa regularyzacji, dokładności i szybsze uczenie [2].

6.4 Batch Normalization i rozbieżność

Sieci neuronowe bez BN często ulegają rozbieżności przy wysokich wartościach współczynnika uczenia. Porównując gradienty w sieciach z BN i bez niej, zauważono, że gradienty dla sieci bez BN są większe i bardziej skupione wokół jednej wartości, co prowadzi do szybszej rozbieżności. Odkryto, że BN reguluje aktywacje każdej warstwy w taki sposób, aby miały średnią wartość zero i odchylenie standardowe równe jeden, co zapobiega eksplozji aktywacji w głębszych warstwach sieci. W ten sposób BN umożliwia uczenie z wyższymi wartościami współczynnika uczenia, co prowadzi do szybszej zbieżności i wyższej dokładności [2].

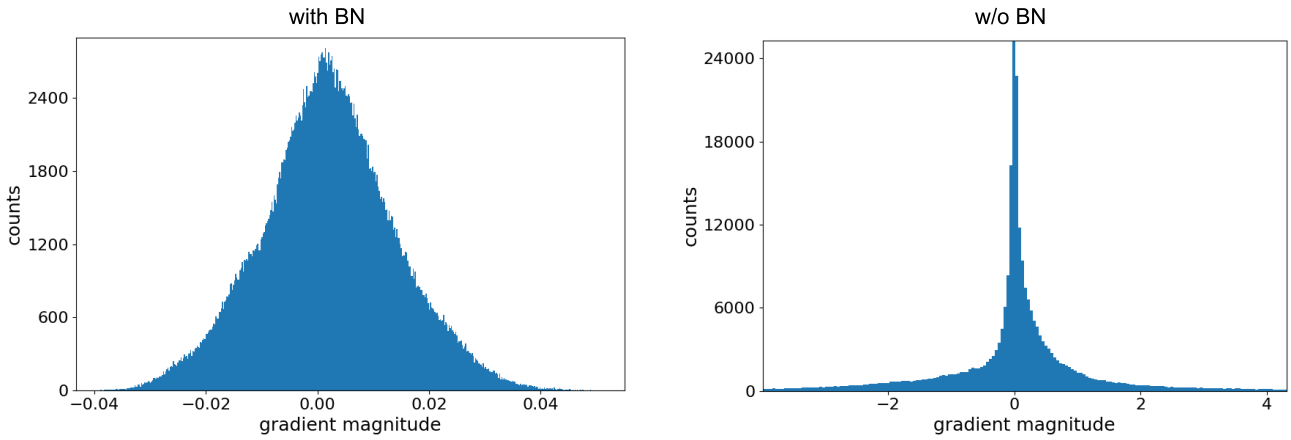

Figure 2: Histogramy dla gradientów na początku uczenia dla warstwy 55 sieci z Batch Normalization (lewa strona) i bez niej (prawa strona) [2].

6.5 Porównanie modeli stosujących BN

Autorzy artykułu proponują metodologię obejmującą rozwój różnych modeli uczenia maszynowego opartych na głębokich sieciach neuronowych z naciskiem na BN. Porównywana jest wydajność kilku modeli z i bez normalizacji wsadowej. We wszystkich przypadkach warstwy normalizacji wsadowej mają właściwości skalowania i przesuwania, które są trenowalne [6].

6.5.1 Sieć VGG

Porównano wyniki dwóch modeli VGG-19, jednego z warstwami konwolucyjnymi bez BN i drugiego, w którym warstwy BN zostały dodane po każdej warstwie konwolucyjnej i przed warstwą aktywacji. Oba modele zostały trenowane przy użyciu optymalizatora Adam przez taką samą liczbę epok. Dla obu modeli początkowa wartość współczynnika uczenia wynosiła 0,1, a została zredukowana do 0,01 po 50% epok i następnie do 0,001 po 75% całkowitej liczby epok [6].

6.5.2 Sieć Inception

Aby zobaczyć efekt BN na sieci Inception, skonstruowano dwa modele sieci Inception v3. Jedyną różnicą między nimi jest to, że dla jednego z modeli są dodane warstwy BN po warstwach konwolucyjnych i przed warstwami aktywacji. Zauważamy tutaj, że bez BN sieć nie może się nauczyć, nawet przy bardzo niskiej wartości współczynnika uczenia 0,0001. Z drugiej strony, sieć z BN uczy się łatwo i osiąga dokładność około 85% z optymalizatorem Adam z początkowym współczynnikiem uczenia 0,1. Współczynnik uczenia jest zmniejszany do 0,01 po 50% iteracji i do 0,001 po 75% iteracji [6].

6.5.3 Sieć Residualna

W przypadku sieci Residualnej obserwacje są takie same jak dla sieci Inception. Dla sieci bez BN, sieć nie jest w stanie niczego się nauczyć, nawet przy nauce z bardzo niską wartością współczynnika uczenia wynoszącym 0,0001. Dla sieci z warstwami normalizacji wsadowej, sieć uczy się nawet z wyższym współczynnikiem uczenia, bez żadnych przeszkód. Początkowo wartość współczynnika uczenia ustawiona jest na 0,1, a następnie zmniejszana do 0,01 po 50% epok i do 0,001 po 75% epok [6].

6.5.4 Wnioski

Bez Batch Normalization, głębokie sieci mogą mieć problemy z nauką lub mogą nauczyć się tylko częściowo, a następnie się psuć, co prowadzi do ograniczonej wydajności.

7 Balanced mini-batch training dla danych niezbalansowanych

7.1 Kowalencyjne uczenie mini-paczek

Konwencjonalnye uczenie z mini-paczkami wybiera niewielką liczbę próbek, które są losowo wybierane ze wszystkich próbek treningowych i przekazywane do sieci neuronowej jako mini-paczka. Podczas generowania mini-paczek określona liczba próbek jest losowo wybierana z pełnego zbioru próbek, a następnie wybrane próbki są usuwane z pełnego zbioru. Ta procedura jest uważana za jedną epokę, a próbki w pełnym zestawie są przywracane za każdym razem, gdy pojawia się nowa epoka. Określona liczba próbek w jednym mini-paczce nazywana jest rozmiarem paczki, więc liczba mini-paczek wygenerowanych w jednej epoce jest równa liczbie wszystkich próbek podzielonej przez rozmiar paczki [5].

7.2 Balanced mini-batch training

W przypadku konwencjonalnego treningu mini-paczek, wszystkie próbki są ekstrahowane tylko raz bez duplikacji w ramach jednej epoki, a mini-paczki dziedziczą niezrównoważenie próbek dla klas w zbiorze macierzystym, gdy zestaw danych treningowych jest niezrównoważony. Dlatego konieczne są metody równoważenia, np. over-sampling lub under-sampling, gdy korzysta się z konwencjonalnego treningu mini-paczek z niezrównoważonym zestawem danych. W przeciwieństwie do konwencjonalnego treningu mini-paczek, metoda równoważenia mini-paczek pozwala na nakładające się wybory próbek mniejszościowych w tej samej epoce i ogranicza liczbę próbek w każdej klasie w jednej mini-paczce do rozmiaru paczki podzielonej przez liczbę klas. Korzystając z tej metody, próbki są równoważone tylko wtedy, gdy tworzona jest mini-paczka, więc nie jest wymagane over-sampling ani under-sampling, a ryzyko przeuczenia się jest zmniejszone [5].

7.3 Zbiory danych

Autorzy metody [5] przerowadzili eksperymenty na dwóch zbiorach danych:

Imbalanced MNIST Jest to zbiór danych, który został celowo zniekształcony, aby potwierdzić skuteczność proponowanej metody w przypadku danych treningowych, które są niesymetryczne. Zbiór ten został utworzony przez wyodrębnienie zdjęć cyfr "5" i "6" z oryginalnego zbioru MNIST. Zmieniono stosunek klasy 6 do klasy 5 w próbkach treningowych na 100%, 10%, 1% i 0,1% i oceniono wpływ proponowanej metody na różne wariancje próbek treningowych. W celu testowania wyodrębniono wszystkie zdjęcia cyfr "5" i "6" z oryginalnego zbioru testowego [5].

Egg image dataset Wykorzystano oznakowane zdjęcia jaj kurzych jako przykład rzeczywistego zestawu danych przemysłowych. Zestaw ten obejmuje sześć klas jakości, a każde zdjęcie jest oznaczone jedną klasą. Z powodu faktycznego procesu produkcyjnego, w którym zostały wykonane zdjęcia, zestaw ten jest niezrównoważony, a klasa 0 stanowiąca jaja nie mające defektów jest klasą większościową. Zdjęcia mają rozmiar 224 x 224 pikseli [5].

7.4 Konfiguracje sieci neuronowych i metody treningu

Do klasyfikacji wykorzystano model sieci konwolucyjnej o nazwie GoogLeNet, który został zmodyfikowany poprzez zmianę wymiarów warstwy wyjściowej z 1000 na 2 (dla zbioru danych imbalanced MNIST) lub 6 (dla zbioru danych obrazów jaj). Parametry były optymalizowane za pomocą algorytmu Adam z początkowym współczynnikiem nauki 0,001, a rozmiar paczki wynosił 60. Wartości początkowe parametrów zostały ustalone na podstawie rozkładu normalnego o odchyleniu standardowym 0,1. Szkolenie zostało zakończone po zrealizowaniu 1000 mini-paczek dla imbalanced MNIST lub 10000 mini-paczek dla egg image dataset, co odpowiada łącznie 60000 i 600000 próbkom obrazów [5].

Porównano cztery różne metody [5]: zwykłe trenowanie przy użyciu danych niezbalansowanych, trenowanie przy użyciu danych z over-sampling, trenowanie przy użyciu danych z under-sampling oraz balanced mini-batch training. Do over-sampling zastosowano metodę opartą na mixup, a do under-sampling usuwano losowo próbki.

7.5 Ocena metod i rezultaty

Metody były oceniane pod kątem zdolności do generalizacji na danych testowych [5]. Do oceny wykorzystano miarę F-measure.

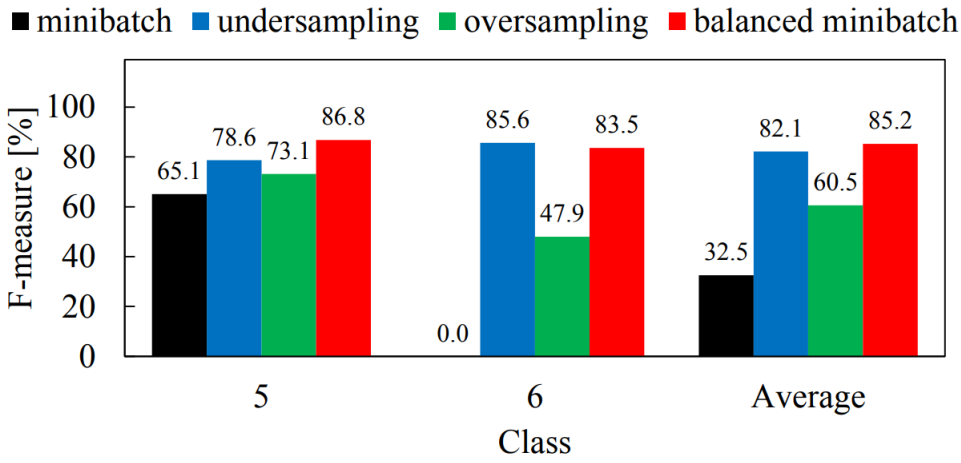

Figure 3: F-measure dla każdej klasy imbalanced MNIST [5].

Imbalanced MNIST Klasifikator trenowany z użyciem mini-batch nie był w stanie sklasyfikować próbek klasy 6, dlatego F-measure klasy 6 była dość niska. Ten problem został zniwelowany w pozostałych trzech metodach, a średnie F-measure klasy 5 i średnia F-measure były najwyższe, gdy użyto balanced mini-batch training [5].

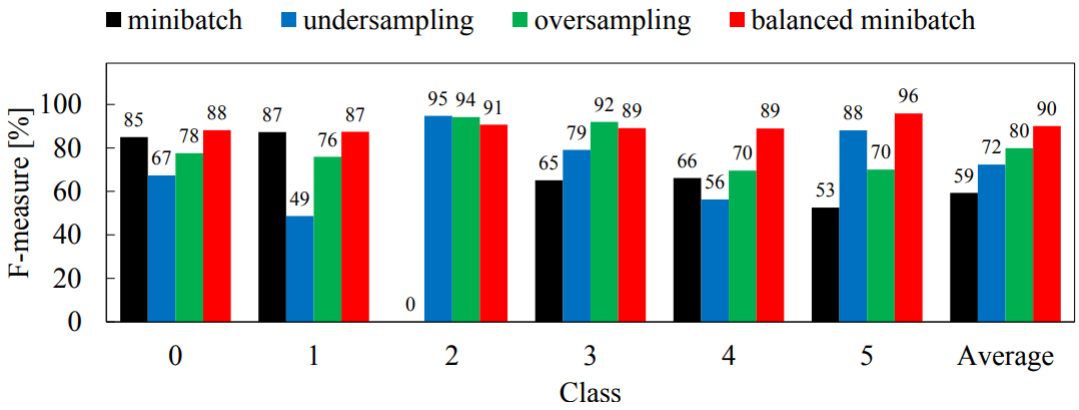

Figure 4: F-measure dla każdej klasy (egg image dataset) [5].

Egg image dataset W przypadku zastosowania oversamplingu lub undersamplingu F-measure dla klasy 2 poprawiła się powyżej 90%, ale pogorszyła się dla klasy 0. Jedynie mtoda balanced mini-batch training pozwoliła na poprawę F-measure dla wszystkich klas, a średnia wartość F-measure była o 10% lub więcej wyższa niż w przypadku over-sampling i under-sampling [5].

References

[1] C. M. Bishop. Neural networks and their applications. *Review of Scientific Instruments*, 65(6):1803, 1994.

[2] N. Bjorck, C. P. Gomes, B. Selman, K. Q. Weinberger. Understanding batch normalization. *Advances in neural information processing systems*, 31, 2018.

[3] J. Brownlee. What is the difference between a batch and an epoch in a neural network. *Machine Learning Mastery*, 20, 2018.

[4] F. Kamalov. Kernel density estimation based sampling for imbalanced class distribution. *Information Sciences*, 512:1192–1201, 2020.

[5] R. Shimizu, K. Asako, H. Ojima, S. Morinaga, M. Hamada, T. Kuroda. Balanced mini-batch training for imbalanced image data classification with neural network. *2018 First International Conference on Artificial Intelligence for Industries (AI4I)*, strony 27–30. IEEE, 2018.

[6] V. Thakkar, S. Tewary, C. Chakraborty. Batch normalization in convolutional neural networks—a comparative study with cifar-10 data. *2018 fifth international conference on emerging applications of information technology (EAIT)*, strony 1–5. IEEE, 2018.

[7] S. Wang, W. Liu, J. Wu, L. Cao, Q. Meng, P. J. Kennedy. Training deep neural networks on imbalanced data sets. *2016 international joint conference on neural networks (IJCNN)*, strony 4368–4374. IEEE, 2016.

[8] S. Węglarczyk. Kernel density estimation and its application. *ITM Web of Conferences*, 23, 2018.