Politechnika Warszawska
Wydział Elektroniki i Technik Informacyjnych

Pracownia Dyplomowa Magisterska (PDMGR)

Metody głębokiego uczenia w wybranych problemach klasyfikacji

Streszczenie W artykule przedstawiono rozwój metod głębokiego uczenia. Następnie zaprezentowano zastosowania w jakich wykorzystywane są opisywane metody. W ostatniej części artykułu przedstawiono schemat działania tych mechanizmów oraz zaprezentowano sposób uczenia sieci wykorzystujący Głęboką Sieć Splotową (DCNN).

Autor:
Jacek Witkowski

Promotor:
mgr inż. Rajmund Kożuszek

1 lutego 2017

Spis treści

1. Wstęp

1.1. Cel pracy

dd

1.2. Historia uczenia maszynowego

W znakomitej większości przypadków tworzenie programu komputerowego polega na podaniu ciągu instrukcji, które mają zostać wykonane, aby rozwiązać zadany problem. Co jednak, jeśli nie wiadomo, w jaki sposób osiągnąć zamierzony cel?

W 1956 roku Arthur Samuel postanowił napisać program komputerowy potrafiący grać w warcaby lepiej niż jego autor. Stworzył więc aplikację, która grała sama ze sobą, ucząc się na własnych błędach i sukcesach. Z czasem oprogramowanie stawało się coraz to lepsze. W 1962 roku program był na tyle dobry, że pokonał mistrza stanu Connecticut: Roberta Nealey'ego (obecnie istnieje algorytm, z którym nie da się wygrać w warcaby).

Z czasem uczenie maszynowe powoli zyskiwało na zainteresowaniu. Wiele tworzonych mechanizmów było inspirowanych naturą (np. algorytmy ewolucyjne czy sztuczne sieci neuronowe). Jednak dużym ograniczeniem w większości problemów wciąż pozostawała konieczność nadzorowania procesu uczenia, tzn. dla każdego zbioru wejściowego należało podać spodziewany wynik (np. przy rozpoznawaniu cyfr podawano rzeczywistą cyfrę odpowiadającą podanemu obrazkowi).

W 1986 roku opracowano teoretyczny model mechanizmu potrafiącego stwierdzać podobieństwo różnych danych i uczyć się bez nadzoru. Była to tzw. Maszyna Boltzmanna. Potrafiła ona zauważać cechy charakterystyczne dla różnego typu danych (np. w problemie rozpoznawania cyfr zauważała charakterystyczne łuki, które odróżniały jedne znaki od drugich). Nie istniał jednak żaden efektywny sposób uczenia mechanizmu. Dopiero w połowie pierwszej dekady XXI wieku zaczęły powstawać pierwsze szybkie algorytmy uczące. Wówczas rozwój dziedziny znacznie przyspieszył. Szczególne zainteresowanie zyskało podejście zwane głębokim uczeniem.

1.3. Zastosowania

Głębokie uczenie (*ang. deep learning*), zwane również uczeniem o głębokich strukturach (*ang. deep structured learning*) lub uczeniem hierarchicznym (*ang. hierarchical learning*),

to koncepcja uczenia polegająca na tworzeniu modelu danych o wielu warstwach, gdzie każda kolejna warstwa reprezentuje wyższy poziom abstrakcji niż poprzednia (warstwa wejściowa ma poziom najniższy). Przykładowo dla warstwowej sieci neuronowej rozpoznającej twarze, w pierwszej warstwie ukrytej zauważane są najprostsze cechy (np. łuki, krawędzie), w kolejnych warstwach rozpoznawane są coraz to większe fragmenty obrazka, by w warstwie wyjściowej uzyskać rozpoznanie całej twarzy.

Obecnie podejście to jest z powodzeniem wykorzystywane w takich usługach jak:

- Google Search by Image,
- Google Voice Search,
- Google Now,
- Siri,
- S Voice,
- Amazon Recommendations,
- Google Self-Driving Car.

Szczególnie dużo projektów wykorzystujących głębokie uczenie prowadzi firma Google (w ramach projektu Google Brain). Między innymi jednym z problemów, którym się zajmowała było oznaczanie na mapie adresów budynków na podstawie obrazów z systemu Google Street View. Choć najpierw rozpoznawaniem zajmowali się ludzie, to na początku 2014 roku stworzono system, który samodzielnie potrafił rozpoznawać adresy. Robił to znacznie szybciej niż ludzie (adresy wszystkich budynków we Francji rozpoznał w niespełna godzinę), a przy tym popełniał mniej błędów.

W ramach badań nad głębokim uczeniem, w obrębie projektu Microsoft Research, powstał tłumacz potrafiący w czasie rzeczywistym tłumaczyć przemówienia wygłaszane w języku angielskim na język chiński, korzystając z głosu osoby mówiącej. Innym znacznym osiągnięciem w dziedzinie przetwarzania języka naturalnego (*ang. Natural Language Processing, NLP*) był system stworzony przez IBM: Watson. Głównym celem systemu było odpowiadanie na pytania zadawane w języku naturalnym. W 2011 roku Watson wziął udział w amerykańskim teleturnieju *Jeopardy!* (polskim odpowiednikiem był teleturniej *Va Banque*). Gra polegała na odpowiadaniu na pytania z różnych dziedzin, a za każdą prawidłową odpowiedź gracz otrzymywał punkty. Watson był tak dobry, że pokonał najlepszych graczy i wygrał milion dolarów.

Choć rozpoznawanie obrazów było niegdyś dziedziną, w której komputery wypadały znacznie gorzej niż ludzie, to wraz z rozwojem głębokiego uczenia sytuacja uległa zmianie. Już w 2011 roku podczas konferencji IJCNN (International Joint Conference on Neural Networks) komputery poradziły sobie z problemem rozpoznawania znaków drogowych lepiej od ludzi (dwukrotnie lepiej).

Kolejnym przełomem w rozpoznawaniu obrazów był system stworzony w ramach Microsoft Research w lutym 2015 roku, którego zadaniem było stwierdzenie, co znajduje się na obrazku, a następnie przypisanie mu odpowiedniej kategorii spośród ponad 100000

dostępnych. Okazało się, że ludzie średnio popełniali 5.1% błędów, a algorytm Microsoftu: 4.94%.

Inne zastosowania głębokiego uczenia jakie są badane to m.in:

- rozpoznawanie niechcianych wiadomości (tzw. spamu),
- automatyczne generowanie opisów dla obrazków w języku naturalnym,
- rozpoznawanie zmian nowotworowych,
- opracowywanie nowych leków,
- rozpoznawanie emocji (*ang. affective computing*).

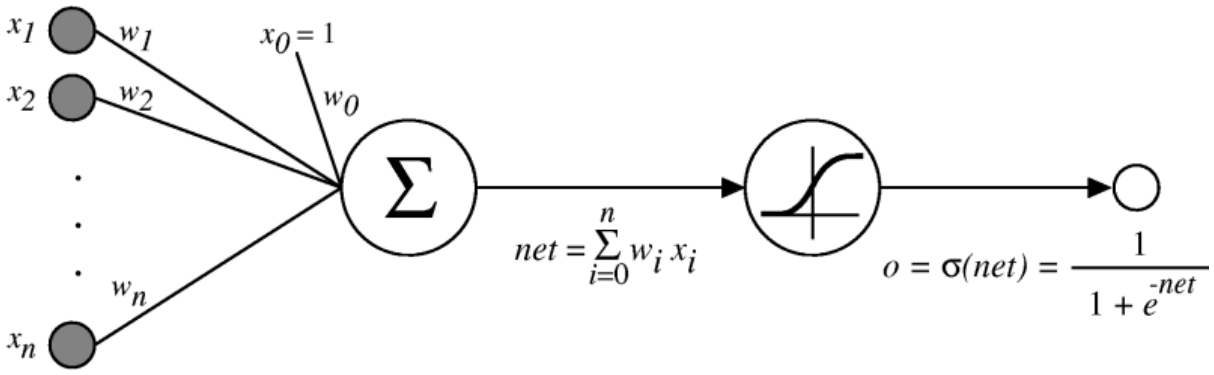
2. Sztuczne sieci neuronowe

W uczeniu maszynowym poprzez **sztuczne sieci neuronowe** rozumiana jest rodzina statystycznych modeli uczenia się zainspirowanych biologicznymi sieciami neuronowymi. Mechanizmy te są wykorzystywane do estymacji lub aproksymacji funkcji, które zależą od wielu wartości wejściowych i są co do zasady nieznane.

Sieć neuronowa składa się z wielu połączonych ze sobą jednostek (zwanych neuronami), które wymieniają między sobą informacje. Połączenia między neuronami posiadają przypisane wagi, które w trakcie uczenia sieci są modyfikowane. Typowo każdy neuron na wejściu otrzymuje zbiór wartości (pochodzący z wyjść innych neuronów lub będący danymi wejściowymi sieci). Następnie wartość każdego z wejść jest mnożona przez wagę przypisaną do odpowiedniego wejścia. Tak otrzymana suma jest poddawana działaniu funkcji aktywacji, która jest charakterystyczna dla danego typu neuronu i nie ulega zmianie wraz z działaniem sieci.

2.1. Model neuronu

Przykładowym neuronem używanym powszechnie w sieciach neuronowych jest neuron z sigmoidalną funkcją aktywacji (nazywany również neuronem sigmoidalnym). Schemat takiej jednostki przedstawiono na poniższym rysunku.



W pierwszej fazie działania neuronu sumowana są wartości na wejściach neuronu pomnożone przez odpowiadające im wagi. W drugiej fazie dla otrzymanej sumy obliczana jest wartość funkcji sigmoidalnej $f(x) = \frac{1}{1+e^{-x}}$, która stanowi wartość wyjściową neuronu.

Innym często stosowanym neuronem jest neuron o progowej funkcji aktywacji, tzn. dla wartości powyżej pewnego progu przyjmuje wartość 1, a dla pozostałych wartości przyj-

muje 0 (lub -1). Wyjścia jednostek tego typu zazwyczaj są jednocześnie wyjściami całej sieci neuronowej i nie są przesyłane na wejścia innych neuronów.

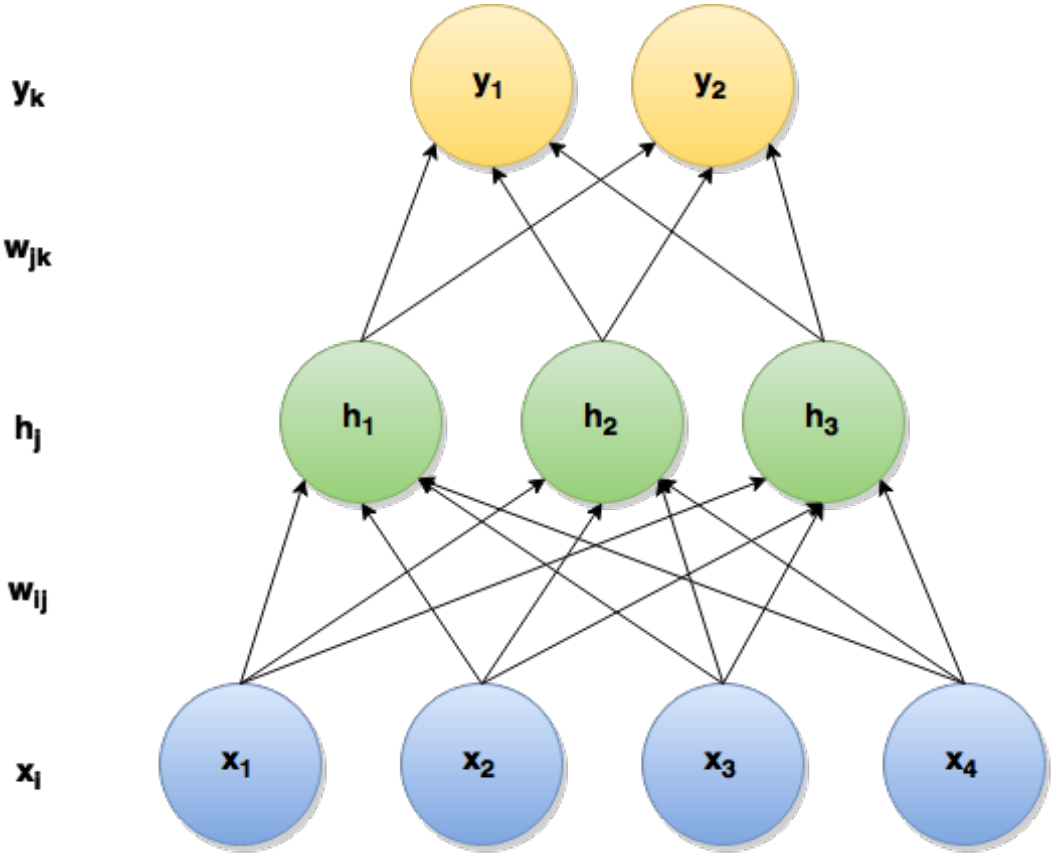
2.2. Warstwowa sieć neuronowa

Przy tworzeniu sztucznej sieci neuronowej ważnym czynnikiem mającym istotny wpływ na sposób rozwiązania zadanego problemu, jest dobór odpowiedniego typu sieci. Obecnie najczęściej wykorzystywane są sieci:

- jednokierunkowe,
- rekurencyjne,
- komórkowe.

2.2.1. Sieć jednokierunkowa

Jednym z najczęściej wykorzystywanych typów sieci jest sieć jednokierunkowa. Charakterystyczną cechą takiej sieci jest brak sprzężeń zwrotnych, tzn. sygnały przesyłane są od warstwy wejściowej poprzez warstwy ukryte aż do warstwy wyjściowej. Model przykładowej sieci warstwowej przedstawiono poniżej.



2.2.2. Wsteczna propagacja błędów

Wsteczna propagacja błędów jest podstawową metodą uczenia nadzorowanego wielowarstwowych jednokierunkowych sieci neuronowych. Poprzez uczenie nadzorowane należy rozumieć proces uczenia, w którym sieć neuronowa otrzymuje dane wraz z ich etykietami (spodziewanym wyjściem sieci).

W pierwszym kroku uczenia należy zdefiniować funkcję straty $Loss(w)$ przyjmującą jako argument wektor wag sieci neuronowej. Funkcją tą może być średni błąd kwadratowy:

$$Loss(w) = \frac{1}{2}\sum_{m}(\sigma(w^T x^{(m)}) - y^{(m)})^2$$

gdzie:

- w - wagi sieci neuronowej,
- m - numer próbki uczącej,
- x - wartości wejściowe,
- y - spodziewane wartości wyjściowe.

Celem uczenia sieci jest minimalizacja funkcji $Loss(w)$. Na początku należy określić gradient funkcji straty. Następnie można przystąpić do optymalizacji tej funkcji wykorzystując metodę gradientu prostego. Kroki tego algorytmu przedstawiono poniżej:

1. zainicjuj losowo wektor wag w,
2. oblicz gradient funkcji straty,
3. $w := w - \alpha \nabla Loss(w)$, α - współczynnik określający długość kroku,
4. wróć do kroku 2. jeśli nie został spełniony warunek stopu.

Zakładając, że wartość błędu dla danego przykładu trenującego wyraża się wzorem

$$Error^{(m)} = \sigma(w^T x^{(m)}) - y^{(m)}$$

gradient funkcji straty ma wzór:

$$\nabla_w Loss = \sum_{m} Error^{(m)} \sigma'(w^T x^{(m)}) x^{(m)}$$

Ponieważ $\sigma(x) = \frac{1}{1+e^{-x}}$, pochodna funkcji sigmoidalnej wyraża się wzorem:

$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$

Dla uproszczenia załóżmy, że w sieci występuje tylko jedna warstwa ukryta. Wówczas gradient funkcji straty względem wag połączeń pomiędzy warstwą wyjściową, a ukrytą:

$$\frac{\partial Loss}{\partial w_{jk}} = \frac{\partial Loss}{\partial in_k}\frac{\partial in_k}{\partial w_{jk}} = \delta_k \frac{\partial(\sum_j w_{jk} h_j)}{\partial w_{jk}} = \delta_k h_j$$

Poprzez in_k oznaczono wektor wejść trafiających do funkcji aktywacji poszczególnych neuronów warstwy wyjściowej.

Gradient funkcjii straty względem wag połączeń pomiędzy warstwą wejściową a ukrytą:

$$\frac{\partial Loss}{\partial w_{ij}} = \frac{\partial Loss}{\partial in_j}\frac{\partial in_j}{\partial w_{ij}} = \delta_j \frac{\partial(\sum_j w_{ij}x_i)}{\partial w_{ij}} = \delta_j x_i$$

$$\delta_k = \frac{\partial}{\partial in_k}(\sum_k \frac{1}{2}[\sigma(in_k) - y_k]^2) = [\sigma(\in_k) - y_k]\sigma'(in_k)$$

$$\delta_j = \sum_k \frac{\partial Loss}{\partial in_k}\frac{\partial in_k}{\partial in_j} = \sum_k \delta_k \cdot \frac{\partial}{\partial in_j}(\sum_j w_{jk}\sigma(in_j)) = [\sum_k \delta_k w_{jk}]\sigma'(in_j)$$

Jak widać w ostatnim wzorze: wykorzystujemy gradient obliczony dla warstwy wyjściowej do obliczenia gradientu dla poprzedniej warstwy. Przy większej liczbie warstw w sieci gradienty dla kolejnych warstw są obliczane analogicznie (na podstawie wartości obliczanych w następnych warstwach). Stąd właśnie metoda ta nosi nazwę wstecznej propagacji błędów.

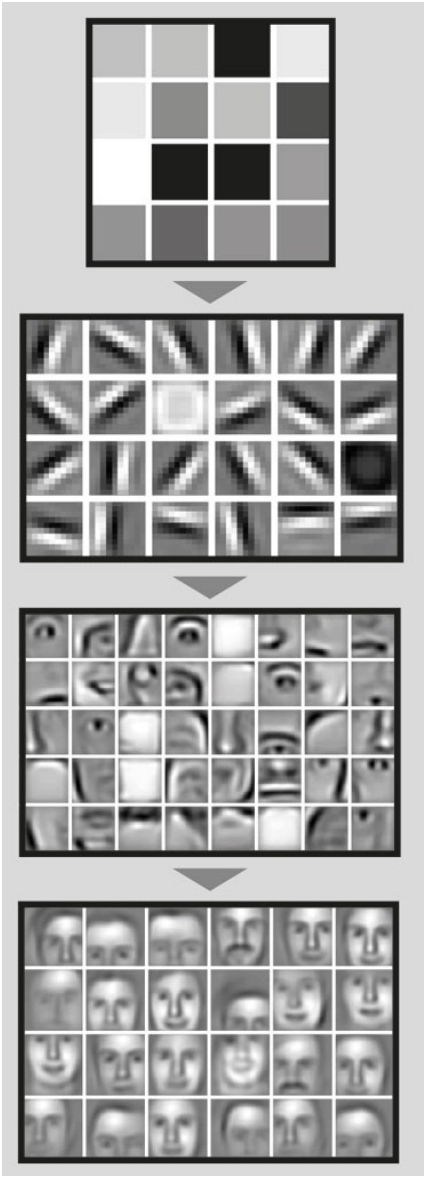
2.3. Ogólna koncepcja głębokiego uczenia

Sama koncepcja głębokiego uczenia nie jest szczególnie nowa. Sieci wielowarstwowe uczone za pomocą wstecznej propagacji błędów istnieją już od ponad 40 lat. Sieciom tym towarzyszy jednak poważna wada: przy wielu warstwach w sieci, wagi wyjść początkowych warstw są aktualizowane w bardzo nieznacznym stopniu (na ogół: im większa odległość warstwy od wyjścia, tym mniejsze korekcje wag). W literaturze problem ten nazywany jest problemem zanikającego gradientu (*ang. vanishing gradient problem*). Z tego powodu do efektywnego uczenia metodą wstecznej propagacji błędów konieczne jest stosowanie niewielkiej liczby warstw (w praktyce stosowano przeważnie do czterech warstw). Choć teoretycznie zastosowanie dwóch warstw wystarcza do aproksymacji dowolnej funkcji, to jednak może wymagać bardzo dużej liczby neuronów w warstwie ukrytej (w niektórych problemach liczba neuronów w warstwie ukrytej może rosnąć wykładniczo względem rozmiaru danych wejściowych).

Rozwiązaniem problemu zanikającego gradientu jest zastosowanie dodatkowego etapu: tzw. uczenia wstępnego (*ang. pre-training*). W tym etapie optymalizowaną funkcją nie jest funkcja błędu (jak w przypadku metody wstecznej propagacji błędu), ale funkcja prawdopodobieństwa wystąpienia danych p(v), gdzie v stanowi dane wejściowe sieci. Celem, do którego dąży etap uczenia wstępnego jest takie uformowanie funkcji p(v), by dla danych podobnych do tych obserwowanych w zbiorze uczącym wartość funkcji prawdopodobieństwa była jak największa, natomiast dla pozostałych danych: jak najmniejsza.

Opisana metoda uczenia wstępnego zakłada, że po kolei uczona jest każda z warstw (nie wszystkie na raz), zaczynając od warstwy wejściowej, kończąc na wyjściowej. W jednej

iteracji algorytmu uczone są wagi tylko jednej z warstw. W pierwszym kroku algorytmu uczona jest pierwsza warstwa ukryta i efektem uczenia powinno być wyodrębnienie takich cech danych, które odróżniają je od danych losowych. W drugim etapie kolejna warstwa jest uczona w taki sposób, by była w stanie odróżnić kombinację cech występujących w danych od losowej kombinacji cech itd. Istotną różnicą tej metody w stosunku do metody wstecznej propagacji błędu jest uczenie tylko jednej warstwy na raz (po nauczeniu danej warstwy jej wagi nie są już modyfikowane w etapie uczenia wstępnego). Tak nauczona sieć, choć jeszcze nie nadaje się do przeprowadzania klasyfikacji, to dobrze modeluje charakter danych, tzn. właściwości danych, które często pojawiają się w zestawie uczącym.



Rysunek 2.1. cs.stanford.edu

Po etapie uczenia wstępnego następuje tzw. dostrajanie sieci (*ang. fine-tuning*). W tym etapie dodawana jest warstwa wyjściowa sieci. Następnie cała sieć jest uczona danymi

etykietowanymi (uczenie nadzorowane) metodą wstecznej propagacji błędu. Dzięki zastosowaniu uczenia wstępnego sztuczna sieć neuronowa znacznie lepiej generalizuje funkcję, którą ma modelować (tzn. jest znacznie mniej podatna na tzw. overfitting). Ponadto, do dostrajania jest potrzebna znacznie mniejsza ilość danych niż do pierwszego etapu. Zatem do uczenia sieci wystarczy, by tylko niewielka część danych uczących była etykietowana.

2.4. Modelowanie funkcji prawdopodobieństwa

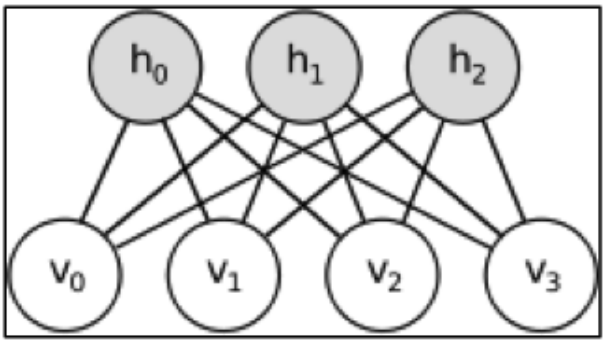
Cechą wspólną wszystkich mechanizmów głębokiego uczenia badanych w ramach tej pracy magisterskiej jest wykorzystanie etapu uczenia wstępnego, którego celem jest odpowiednie zamodelowanie funkcji rozkładu prawdopodobieństwa, tzn. osiągnięcie takiego rozkładu prawdopodobieństwa p(v), że danym wejściowym v, które są podobne do danych uczących, będą odpowiadały wysokie wartości funkcji prawdopodobieństwa, natomiast pozostałym danym (w szczególności danym losowym) będzie odpowiadało niskie prawdopodobieństwo.

Do modelowania funkcji prawdopodobieństwa można wykorzystać wiele mechanizmów. W tej pracy omówiona zostanie jedynie **Ograniczona Maszyna Boltzmanna**(*ang. Restricted Boltzmann Machine*).

2.4.1. RBM

Ograniczona Maszyna Boltzmanna jest sztuczną siecią neuronową składającą się z dwóch warstw neuronów:

- warstwy neuronów wejściowych,
- warstwy neuronow ukrytych.



Cechą charakterystyczną tego mechanizmu jest jego umiejętność do odszukiwania cech danych wejściowych (*ang. feature extraction*). Przykładowo Ograniczona Maszyna Boltzmanna, którą uczono obrazkami, będzie wykrywać charakterystyczne łuki czy krawędzie.

Dzięki tej właściwości mechanizm może być wykorzystywany m.in. do korekcji danych czy wręcz do ich generowania (np. do generowania pisma ręcznego).

Prawdopodobieństwo stanów sieci

W Ograniczonej Maszynie Boltzmanna każdy stan sieci (tzn. zestaw wartości wyjść poszczególnych neuronów) ma określone prawdopodobieństwo zadane wzorem:

$p(v,h) = \frac{1}{Z}e^{-E(v,h)}$, gdzie:

- v to macierz wyjść neuronów wejściowych,
- h to macierz wyjść neuronów ukrytych,
- Z to czynnik normalizujący (tzw. funkcja partycji),
- $E(v,h) = -a^T v - b^T h - v^T W h$,
- a,b oraz W to macierze wag sieci.

Inferencja

Ze względu na to, że neurony nie łączą się ze sobą w obrębie tej samej warstwy, wartość na wyjściu każdego neuronu w danej warstwie można uzyskać wyłącznie na podstawie wartości wyjść neuronów drugiej warstwy. Prawdopodobieństwa wzbudzeń neuronów można efektywnie obliczyć korzystając z wzorów:

- $p(h_j = 1|v) = \sigma(b_j + \sum_{i=1}^{m} w_{i,j}v_i)$,
- $p(v_i = 1|h) = \sigma(a_i + \sum_{j=1}^{n} w_{i,j}h_j)$,
- gdzie: σ to funkcja aktywacji, i oraz j to numery kolejnych neuronów (odpowiednio: wejściowych i ukrytych).

Uczenie

Celem uczenia sieci jest znalezienie takich parametrów a,b i W, że wartość funkcji prawdopodobieństwa p(v) dla danych obserwowanych będzie wysoka, a dla pozostałych danych niska. W praktyce zamiast maksymalizować funkcję prawdopodobieństwa p(v), minimalizuje się funkcję średniego ujemnego log-prawdopodobieństwa:

$$\frac{1}{T}\sum_{t} -\log p(v^{(t)})$$

Do minimalizacji funkcji metodami gradientowymi konieczne jest obliczenie pochodnej po parametrach sieci dla tej funkcji:

$$\frac{\partial -\log p(v^{(t)})}{\partial \theta} = \mathbb{E}_h[\frac{\partial E(v^{(t)},h)}{\partial \theta} \mid v^{(t)}] - \mathbb{E}_{v,h}[\frac{\partial E(v,h)}{\partial \theta}]$$

Pierwszy ze składników sumy występującej w powyższym wzorze to tzw. składnik pozytywny. Drugi składnik to tzw. składnik negatywny. Pierwszy z nich odpowiada zwiększaniu wartości funkcji prawdopodobieństwa dla obserwowanych danych, a drugi zmniejsza

prawdopodobieństwo danych nieobserwowanych. Drugi składnik jest skomplikowany obliczeniowo, gdyż występuje w nim iteracja po wszystkich możliwych stanach sieci, których liczba rośnie wykładniczo względem liczby neuronów w sieci. Z tego powodu dokonuje się estymacji. Jedną z wykorzystywanych metod jest tzw. próbkowanie Gibbsa.

Próbkowanie Gibbsa pozwala wyeliminować problem obliczania wartości oczekiwanej dla wszystkich możliwych stanów sieci. Zamiast tego wartość ta jest estymowana za pomocą algorytmu, którego kroki przedstawiono poniżej.

Próbkowanie Gibbsa (algorytm):

1. Oblicz wartości wyjściowe dla każdego neuronu z warstwy ukrytej na podstawie danych wejściowych (korzystając z wzorów podanych w sekcji 4.1.2).
2. Oblicz wartości wyjściowe dla każdego neuronu z warstwy wejściowej na podstawie obliczonych wartości wyjściowych neuronów ukrytych (korzystając z wzorów z sekcji 4.1.2).

Kroki 1 i 2 mogą być powtarzane wielokrotnie, choć w praktyce okazuje się, że wystarczy jeden przebieg algorytmu. Otrzymane w punkcie drugim „odtworzone" wartości wejścia oznaczane są jako $\tilde{v}$.

Aktualizacja parametrów sieci Parametry (wagi) sieci aktualizowane są według następujących wzorów:

- $W_t = W_{t-1} + \alpha(h(v^{(t)})v^{(t)T} - h(\tilde{v})\tilde{v}^T)$
- $b_t = b_{t-1} + \alpha(h(v^{(t)}) - h(\tilde{v}))$
- $a_t = a_{t-1} + \alpha(v^{(t)}) - \tilde{v})$

Przez $h(v)$ oznaczono wartości wyjściowe neuronów ukrytych obliczone na podstawie wartości wyjściowych neuronów wejściowych.

3. Splotowe sieci neuronowe

Splotowe sieci neuronowe są wielowarstwowymi sieciami jednokierunkowymi. Charakteryzuje je możliwość wykrywania wzorców występujących w różnych fragmentach przetwarzanego obrazu (np. rozpoznawanie oka w dowolnym fragmencie obrazka). Zasada ich działania była inspirowana neurobiologią, a mianowicie analizowano sposób przetwarzania informacji przez ośrodek wzrokowy kota.

3.1. Filtry splotowe

Splot dyskrentny jako pojęcie matematyczne jest zdefiniowany w następujący sposób:

$$f * g[n] \stackrel{\text{def}}{=} \sum_{m=-\infty}^{\infty} f[m]g[n-m] = \sum_{m=-\infty}^{\infty} f[n-m]g[m]$$

W cyfrowym przetwarzaniu obrazów filtry splotowe znajdują bardzo szerokie zastosowanie, gdyż w zależności od dobranej maski filtra, osiągane są różne rezultaty. Przykładowe filtry splotowe:

- filtr Gaussa (wygładzanie),
- filtr Laplace'a (wyostrzanie).

Maski dla wymienionych filtrów są z góry ustalone i nie ulegają zmianie w czasie działania algorytmu. Splotowa sieć neuronowa zamiast wykorzystywać z góry określone maski, „uczy się ich", a więc stara się tak dobrać ich wartości, aby jak najlepiej rozpoznawać wyznaczone obiekty na obrazach. Poszczególnym wartościom w masce filtra odpowiadają wagi splotowej sieci neuronowej.

3.1.1. Padding

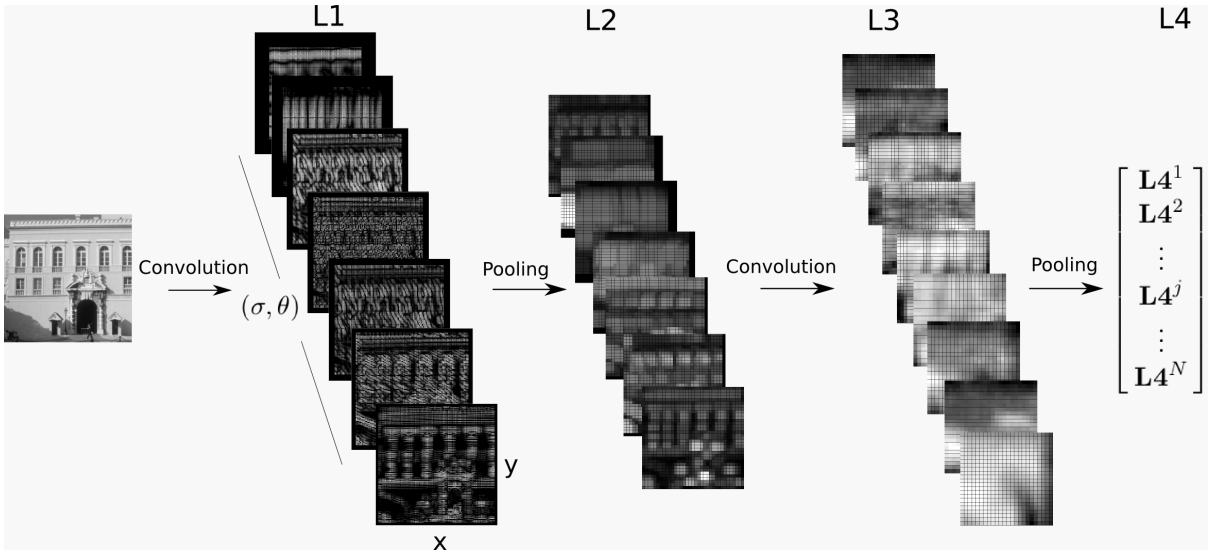
Przy stosowaniu filtrów splotowych pojawia się problem: jaką wartość powinny otrzymać piksele obrazka znajdujące się na jego krawędzi. Jest on rozwiązywany poprzez rozszerzanie obrazka o dodatkowe piksele na jego krawędziach. Można tego dokonać m.in. poprzez:

- rozszerzenie obrazka o czarne piksele (tzw. *zero-padding*),
- rozszerzenie obrazka odbicia lustrzane pikseli przy krawędziach.

3.2. Inferencja

W trakcie działania sieci na obrazie dokonywanie jest filtrowanie splotowe (przy wykorzystaniu różnych filtrów). Po takim filtrowaniu otrzymywane jest $n \cdot m$ obrazów, nazywanych mapami cech (*ang. features maps*), gdzie n - liczba początkowych obrazów (tzw. kanałów wejściowych), m - liczba użytych filtrów.

Po dokonaniu filtrowania splotowego obrazy są skalowane na mniejsze (tzw. faza pooling/subsampling) i znów stosowane są filtry splotowe. Oba kroki (filtrowanie i skalowanie) są powtarzane wielokrotnie (zależy od liczby warstw sieci), aż do osiągnięcia odpowiednio wyskokopoziomowych cech. Liczba warstw oraz liczba filtrów splotowych wykorzystanych w każdej z nich jest ustalana podczas fazy projektowania sieci.



Ostatnią warstwą sieci (zazwyczaj otrzymującą dużo bardzo małych obrazów) jest warstwa zawierająca neurony sigmoidalne. Każdy neuron warstwy wyjściowej jest połączony ze wszystkimi wartościami otrzymanymi na wyjściu poprzedniej warstwy.

3.2.1. Skalowanie

Po zastosowaniu n różnych filtrów splotowych w danej warstwie na m różnych mapach cech, powstaje $n \cdot m$ kolejnych map cech. Stąd ilość przetwarzanych danych szybko rośnie wraz z dokładaniem kolejnych warstw w sieci. Aby temu przeciwdziałać stosuje się skalowanie obrazów pomiędzy warstwami dokonującymi splotu. Najczęściej obraz skalowany jest poprzez:

1. podzielenie obrazka na nienachodzące na siebie kwadratowe obszary,
2. wybranie z każdego obszaru piksela o największej wartości (tzw. *max-pooling*).

Alternatywnie, w drugim kroku algorytmu można wybrać medianę wartości pikseli (*ang. mean-pooling*) lub wartość średnią (*average-pooling*).

3.2.2. Możliwe ulepszenia sieci

TODO rectification(prostowanie/rektyfikacja???), local contrast normalization(lokalna normalizacja kontrastu), dropout

3.3. Uczenie metodą wstecznej propagacji

Typową metodą uczenia sieci jest wsteczna propagacja błędów. W tym celu należy obliczyć gradient funkcji straty względem wag sieci (wartości masek filtrów splotowych) dla warstw:

- wyjściowych,
- skalujących,
- splotowych.

Obliczanie gradientu funkcji straty dla warstw zawierających neurony sigmoidalne zostało omówione w sekcji (TODO), stąd wyjaśnienia wymaga jedynie obliczanie gradientów dla warstw skalujących i splotowych.

3.3.1. Obliczanie gradientu dla warstw skalujących

Znając gradient funkcji straty $\nabla_{y_{ijk}} l$ z warstwy kolejnej, można w prosty sposób policzyć gradient w stosunku do wejścia warstwy dla której jest on liczony. W przypadku, gdy warstwa wykorzystywała algorytm *max-pooling*, gradient względem wejść warstwy przyjme wartość:

- $\nabla_{x_{ijk}} l = \nabla_{y_{ijk}} l$, dla pikseli, które miały maksymalną wartość w obszarze,
- $\nabla_{x_{ijk}} l = 0$, dla pozostałych pikseli.

3.3.2. Obliczanie gradientu dla warstw splotowych

Przy wstecznej propagacji błędów z warstwy kolejnej otrzymujemy gradient błędu względem wyjścia aktualnej warstwy $\nabla_{y_j} l$. Do aktualizacji wag sieci konieczne jest policzenie gradientu funkcji straty wzlgędem wag tej warstwy.

$$\frac{\partial E}{\partial w_{ab}} = \sum_{i=0}^{N-m} \sum_{j=0}^{N-m} \frac{\partial E}{\partial x_{ij}^l} \frac{\partial x_{ij}^l}{\partial w_{ab}} = \sum_{i=0}^{N-m} \sum_{j=0}^{N-m} \frac{\partial E}{\partial x_{ij}^l} y_{(i+a)(j+b)}^{l-1}$$

gdzie:

- $N \times N$ - rozmiar mapy cech,
- $m \times m$ - rozmiar filtra,
- x_{ij}^l - wartość wejścia neuronu o wsp. (i,j) w warstwie l,

- w_{ab} - wartość w masce filtra znajdująca się na pozycji (a,b).

Następnie należy policzyć tzw. delty ($\frac{\partial E}{\partial x_{ij}^l}$):

$$\frac{\partial E}{\partial x_{ij}^l} = \frac{\partial E}{\partial y_{ij}^l}\frac{\partial y_{ij}^l}{\partial x_{ij}^l} = \frac{\partial E}{\partial y_{ij}^l}\frac{\partial}{\partial x_{ij}^l}(\sigma(x_{ij}^l)) = \frac{\partial E}{\partial y_{ij}^l}\sigma'(x_{ij}^l)$$

Na koniec należy policzyć gradient funkcji błędu względem wyjść warstwy poprzedniej (po to, by przekazać go do poprzedniej warstwy):

$$\frac{\partial E}{\partial y_{ij}^{l-1}} = \sum_{a=0}^{m-1}\sum_{b=0}^{m-1}\frac{\partial E}{\partial x_{(i-a)(j-b)}^l}\frac{\partial x_{(i-a)(j-b)}^l}{\partial y_{ij}^{l-1}} = \sum_{a=0}^{m-1}\sum_{b=0}^{m-1}\frac{\partial E}{\partial x_{(i-a)(j-b)}^l}w_{ab}$$

Należy zwrócić uwagę na dwie rzeczy: po piersze na to, że podczas obliczania gradientów również wykonywany jest splot jednak na obrazku o „odwróconych" wierszach i kolumnach. Dodatkowo, by móc dokonywać splotu na krawędziach map cech, należy zastosować *zero-padding*.

3.3.3. Uczenie wstępne

W celu lepszego zainicjowania wag sieci należy przeprowadzić uczenie wstępne. W tym etapie sieć zamiast uczyć się rozpoznawania obiektów, ma za zadanie nauczyć się charakteru danych wejściowych (ma zauważać charakterystyczne cechy, podobieństwa obiektów it.p.). Do uczenia wstępnego wykorzystuje się mechanizmy uczenia nienadzorowanego, takie jak:

- autoenkoder,
- RBM (Restricted Boltzmann Machine).

Uczenie wstępne przebiega w następujący sposób:

1. wyodrębnienie losowych fragmentów obrazka (tzw. łatki),
2. uczenie mechanizmu uczenia nienadzorowanego (np. RBM),
3. zainicjowanie wag uczonej warstwy wagami z tego mechanizmu,
4. policzenie map cech dla łatek,
5. wykonanie kroków 1-3 dla uzyskanych map cech (uczenie wstępne kolejnej warstwy).

4. Podsumowanie

Choć początkowo sieci neuronowe nie były zbyt szeroko stosowane, to dzięki opracowaniu metod głębokiego uczenia rozwój uczenia maszynowego znacznie przyspieszył. Obecnie metody te są wykorzystywane w wielu mechanizmach, z których dużej części używamy na co dzień. Celem mojej pracy magisterskiej będzie opisanie tych mechanizmów oraz sprawdzenie jak sprawdzają się one w różnych problemach klasyfikacji.

5. Bibliografia

- Hugo Larochelle, Neural networks class - Université de Sherbrooke: http://tinyurl.com/lpkvjm4,
- DeepLearning.net,
- Stanford University - Machine Learning:
  https://www.coursera.org/course/ml,
- Image Description Generator: http://deeplearning.cs.toronto.edu/i2t,
- Deep Learning - Przykłady:
  http://deeplearning.net/demos,