Rozpoznawanie i klasyfikacja obrazów

Seminarium algorytmy numeryczne i graficzne

Mateusz Markiewicz

22 maja 2020

Spis treści

1 Czym jest rozpoznawanie i klasyfikacja obrazów

1.1 Klasyfikacja (Image Classification)

Klasyfikacja jest procesem polegającym na przypisaniu elementowi otrzymanemu jako wejście jednej klasy. Wszystkie klasy, które mogą zostać zwrócone musimy wcześniej znać.

1.2 Rozpoznawanie (Image Recognition)

Rozpoznawanie obrazów to proces polegający na rozpoznaniu na zdjęciu określonych osób, zwierząt, obiektów lub przedmiotów. Przykładem jest proces rozpoznania twarzy. Przykładem rozpoznawania obrazów jest rozpoznawanie twarzy, algorytm musi poradzić sobie z sytuacją, w której twarz nie znajduje się idealnie po środku, musi więc ją wykryć, a następnie sklasyfikować, czy należy do właściciela telefonu.

1.3 Wykrywanie (Image Detection)

Rozpoznawanie to proces polegający na oznaczeniu na obrazie osób / zwierząt / przedmiotów. Często algorytmy poza oznaczeniem gdzie znajdują się poszczególne obiekty klasyfikuje je.

1.4 Co nam będzie potrzebne?

Wszystkie wyżej wymienione problemy są częścią uczenia nadzorowanego (supervised learning). Oznacza to, że w procesie uczenia dysponujemy danymi, które są poprawnie zaklasyfikowane, przez co sam proces uczenia może być nadzorowany. Mając dany model wiemy, jak dobrze radzi sobie on na danych, którymi dysponujemy i w oparciu o to, możemy go poprawić.

1.5 Data sets

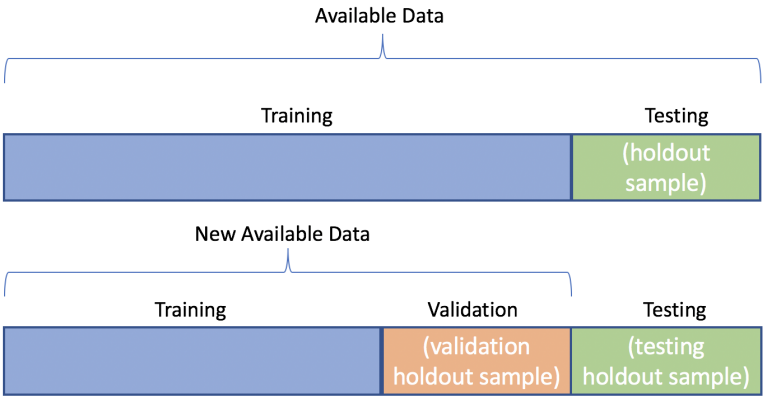
Będziemy więc potrzebowali zbioru danych z poprawnie przypisanymi klasami. Chcąc stworzyć model do rozpoznawania ręcznie zapisanych cyfr potrzebujemy przykładowych danych. Wiele takich zbiorów danych jest dostępnych w internecie, przykładowo najpopularniejszym zbiorem do opisanego wyżej zadania jest Mnist.

1.6 Train-test split

Nie możemy jednak używać całych danych, którymi dysponujemy do nauki.

Naszym celem jest, by nasz model radził sobie dobrze z danymi, których nigdy wcześniej nie widział.

Jeśli w procesie uczenia będziemy oceniać skuteczność modelu na podstawie tych samych danych, na których ten model się uczył, to pomiar ten nie będzie wiarygodny. W skrajnych przypadkach model może zapamiętać klasy wszystkich danych, na których go uczymy, uzyskując 100% skuteczności, pomimo że najprawdopodobniej nie poradzi sobie z danymi, których nigdy nie widział. Jak sobie z tym poradzić?



Rysunek 1: `https://datascience.stackexchange.com/questions/61467/clarification-on-train-test-and-val-and-how-to-use-implement-it`

2 KNN (K-nearest neighbors)

Pierwszym podejściem do problemu klasyfikacji będzie algorytm KNN. Idea algorytmu jest bardzo prosta. Dla obrazka, który dostajemy do sklasyfikowania znajdujemy K najbliższych mu obrazków z danych treningowych oraz zwracamy najczęstszą spośród ich klas.

Co oznacza najbliższych w kontekście obrazów? Obraz jest macierzą, której każdym element reprezentuje pojedynczy piksel. W przypadku MNIST piksele są w skali szarości, więc każdy reprezentowany jest jako pojedyncza wartość. Możemy rozwinąć macierz w wektor, a następnie mając dwa takie wektory obliczamy drugą normę ich różnicy, a więc wartość:

$$dist(X^i, Y^j) = \sqrt{\sum_{k=0}^{K}(X_k^i - Y_k^j)^2}$$

gdzie X^i to i-ty wektor spośród danych treningowych, a Y^j to j-ty wektor spośród danych testowych.

Dla każdego wektora, któremu chcemy przyporządkować klasę potrzebujemy jego odległość do wszystkich danych treningowych, czyli macierzy D, takiej że

$$D_{i,j} = Dist(X^i, Y^j)$$

Potrzebujemy metody na szybkie obliczanie takiej macierzy. Pierwszą obserwacją jest to, że nie potrzebujemy obliczać pierwiastka, ponieważ interesuje nas jedynie wyznaczenie K najbliższych wektorów, nie potrzebujemy znać dokładnej odległości do tych wektorów. Następnie możemy zauważyć, że:

$$\sum_{k=0}^{K}(X_k^i - Y_k^j)^2 = (X^i - Y^j)^T(X^i - Y^j)$$

następnie rozwijamy to do postaci:

$$(X^i - Y^j)^T(X^i - Y^j) = \sum_{k=0}^{K}(X_k^i)^2 + \sum_{k=0}^{K}(Y_k^j)^2 - 2 * (Y^j)^T * X^i$$

mając taką postać możemy pominąć wyrażenie $\sum_{k=0}^{K}(Y_k^j)^2$, ponieważ jest ono takie same dla odległości do dowolnego wektora z danych treningowych. Taka postać wyrażenie wykorzystujemy do obliczenia całej macierzy D w tym samym czasie za pomocą wyrażenia:

$$\sum_{k=0}^{K} X_k^2 - 2YX^T$$

2.1 Redukcja wymiarowości

Aktualnie nasze wektory mają 784 (28*28) wymiarów. W niektórych sytuacjach będziemy chcieli przedstawić te wektory w przestrzeni mniej wymiarowej. Przykładem może być konieczność wizualizacji tych danych, ale takie działanie może również poprawić skuteczność oraz prędkość działania klasyfikatora

2.1.1 Random projection

Wybieramy n losowych wymiarów.

2.1.2 PCA

Principal component analysis - analiza głównych składowych. Wyznacza nowe, nieskorelowane współrzędne tak, by maksymalizowały wariancję danych. Nowe współrzędne są kombinacją liniową starych współrzędnych.
Szczegóły - `http://cs229.stanford.edu/notes/cs229-notes10.pdf`

2.1.3 K-PCA

Kernel principal component analysis. Od PCA różni się tym, że najpierw zwiększamy wymiarowość danych poprzez zastosowanie kernelizacji, a następnie stosujemy analizę składowych głównych.

2.1.4 Isomap

Isomap dla każdego punktu wyznacza jego k najbliższych sąsiadów, następnie tworzy ich graf oraz zapamiętuje w nim najkrótsze ścieżki pomiędzy każdymi dwoma wierzchołkami. Mając takie dane próbujemy znaleźć takie punkty, które jak najlepiej odwzorowują te informacje w mniej wymiarowej przestrzeni.

2.1.5 LLE

Locally Linear Embedding. Algorytm polega na wyznaczeniu k najbliższych sąsiadów, następnie znalezieniu wag pozwalających jak najlepiej przybliżyć dany punkt jako liniową kombinację jego najbliższych sąsiadów wraz z tymi wagami, tj.

$$x_i \approx \sum_{k=0}^{K} x_k * w_{ik}$$

Następnie staramy się znaleźć takie punkty w przestrzeni mniej wymiarowej, że każdy punkt jest jak najlepiej opisany za pomocą wcześniej uzyskanych wag, tj.

$$y_i \approx \sum_{k=0}^{K} y_k * w_{ik}$$

2.1.6 MDS

Multidimensional scaling. Algorytm polega na wyznaczeniu macierzy odległości pomiędzy każdą parą punktów, a następnie stara się znaleźć takie punkty w przestrzeni mniej wymiarowej, że ich macierz odległości jest jak najbardziej zbliżona do tej oryginalnej.

2.1.7 t-SNE

T-distributed Stochastic Neighbor Embedding. Znając podobieństwa par punktów możemy wyznaczyć prawdopodobieństwo znalezienia się blisko siebie dla każdej pary punktów. Następnie staramy się znaleźć punkty w przestrzeni mniej wymiarowej, które jak najlepiej realizują takie prawdopodobieństwa.

2.1.8 Uwaga

MDS oraz t-SNE służą głównie do wizualizacji. W przeciwieństwie do pozostałych metod nie pozwalają one na wyznaczeniu odpowiedniego przekształcenia tylko za pomocą danych treningowych, a następnie zastosowaniu go również do danych testowych. Uniemożliwia to użycie tych metod do klasyfikacji, ponieważ chcąc wyznaczyć klasę nowego wektora musielibyśmy wyznaczać nową redukcję wymiarowości i szkolić cały model na podstawie tak zredukowanych danych treningowych.

3 SVM

SVM tutorial - `https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/svmtutorial.pdf`

Niech X^i będzie i-tym wektorem danych (np. rozwiniętą macierzą reprezentującą obraz), a Y^i będzie klasą tego wektora, gdzie $Y \in \{-1, 1\}$, mamy więc 2 możliwe klasy.

Celem SVM jest wyznaczenie klasy dla X^i na podstawie signum$(w^T X^i + b)$.

Mając zbiór danych testowych X chcemy wyznaczyć odpowiednie wagi w oraz przesunięcie b. Dopuszczamy pojedyncze błędy w klasyfikacji danych treningowych, chcemy więc, by:

$$Y^i(w^T X^i + b) \geq 1 - \xi_i$$

Realizujemy to minimalizując następujące równanie:

$$\begin{aligned} &\min_{w,b,\xi} \frac{1}{2} w^T w + C \sum_{i=1}^{m} \xi_i \\ &\text{t.że } y^{(i)}(w^T x^{(i)} + b) \geq 1 - \xi_i \ \ \forall_i \\ &\quad \xi_i \geq 0 \ \ \forall_i. \end{aligned} \tag{1}$$

3.1 Kernel trick

Kernel trick - `http://cs229.stanford.edu/notes2019fall/cs229-notes3.pdf`

Powyżej sformułowane równanie możemy rozwiązać za pomocą mnożników Lagrangea. w można przedstawić jako liniową kombinację danych treningowych, czyli:

$$w = \sum_{i=0}^{I} \alpha_i * X^i$$

Gdzie α_i to mnożniki Langrangea.

Następnie możemy zaobserwować, że chcąc wyznaczyć klasę wektora X^k musimy obliczyć

$$f(\mathbf{X}) = w^T\mathbf{X} + b = \sum_{i=0}^{I} \alpha_i * (X^i)^T * \mathbf{X} + b$$

Wiemy, że nie wszystkie dane możemy rozdzielić linią. Jeśli przedstawimy je jednak w przestrzeni wyżej wymiarowej może się okazać, że będzie się dało znaleźć linie (w tej wyżej-wymiarowej przestrzeni), która je rozdziela. Chcemy wykorzystać nieliniowe przekształcenie $x \to \phi(x)$. Powyższe równanie sprowadzają się do:

$$w = \sum_{i=0}^{I} \alpha_i * \phi(X^i)$$

$$f(\mathbf{X}) = w^T\phi(\mathbf{X}) + b = \sum_{i=0}^{I} \alpha_i * (\phi(X^i))^T * \phi(\mathbf{X}) + b$$

Potrzebujemy więc wartość iloczynu skalarnego obliczonego w przestrzeni $\phi(*)$. Jeśli będziemy znali sposób na szybkie obliczanie iloczynu skalarnego nie będziemy musieli obliczać $\phi(\mathbf{X})$. Czy to jest jednak możliwe?

Okazuje się, że są takie nieliniowe przekształcenia $\phi(*)$, dla których jest to możliwe.

Niech $X = [X_0, X_1]$ oraz $Y = [Y_0, Y_1]$.

Niech $\phi(X) = [X_0^2, \sqrt{2} * X_0 * X_1, X_1^2]$.

Oznaczmy $K(X, Y) = \phi(X)^T\phi(Y)$ Pokażemy, że jesteśmy w stanie obliczyć $K(X, Y)$ wykonując jedynie obliczenia w oryginalnej przestrzeni.

Niech $K(X, Y) = (X^TY)^2$, wówczas:

$$
\begin{aligned}
K(X,Y) &= (X^T Y)^2 \\
&= (X_0 Y_0 + X_1 Y_1)^2 \\
&= (X_0^2 Y_0^2 + 2 X_0 Y_0 X_1 Y_1 + X_1^2 Y_1^2) \\
&= (\phi(X))^T (\phi(Y))
\end{aligned}
\tag{2}
$$

Okazuje się, że takich funkcji jest więcej, najczęściej używane:

Polynomial Kernel: $K(X,Y) = (X^T Y + \Theta)^d$

RBF (Radial Basis Function / Gaussian Kernel): $K(X,Y) = e^{-\frac{||X-Y||^2}{2\sigma^2}}$

Korzystając z tych obserwacji możemy wykorzystać Kernel trick do SVM. Wówczas równanie, które będziemy chcieli optymalizować w celu odnalezienia najlepszych parametrów przyjmie postać:

$$
\begin{aligned}
\min_{\alpha} \frac{1}{2} \alpha^T \mathbf{H} \alpha - 1^T \alpha \\
s.t. - \alpha_i \leq 0 \\
\alpha_i \leq C \\
y^T \alpha = 0 \\
H = y^T K y \\
K_{i,j} = K(X^i, X^j)
\end{aligned}
\tag{3}
$$

Równanie w tej postaci może zostać rozwiązane, np. za pomocą QP-solvera.

3.2 Multi-class SVM

Są dwa główne podejścia, by przystosować wyżej opisany algorytm do radzenia sobie z danymi mającymi więcej, niż dwie klasy.

3.2.1 One against all

Mając dane składające się z K klas trenujemy K klasyfikatorów. i-ty z nich decyduje, czy klasyfikowany wektor należy do i-tej klasy, czy do reszty klas. Po skończony treningu otrzymujemy K par (w_k, b_k). Chcąc sklasyfikować nowy wektor X obliczamy wartości $f^k(X) = w_k^T X + b_k$, a następnie zwracamy k dla którego $f^k(X)$ osiąga maksymalną wartość.

3.2.2 One against one

Trenujemy $K(K-1)$ klasyfikatorów, po jednym dla każdej pary klas. Klasyfikując nowy wektor tworzymy K liczników, po jednym dla każdej klasy Mając k-ty klasyfikator dla pary klas (i,j), jeśli $f^k(X)$ będzie równe 1 inkrementujemy licznik i-tej klasy, w przeciwnym przypadku inkrementujemy licznik j-tej klasy. Na koniec zwracamy klasę o największej wartości licznika.

4 Regresja logistyczna

Regresja liniowa i logistyczna - `http://cs229.stanford.edu/notes/cs229-notes1.pdf`

Regresja logistyczna jest modelem, który dla każdej wartości wejściowej zwraca wartość wyjściową z przedziału $[0, 1]$. Bardzo często wykorzystuje się ten fakt i stosuje się regresje logistyczną do klasyfikacji. Mając zbiór danych X oraz ich klasy Y, gdzie $Y^i \in \{0, 1\}$ możemy klasyfikować nowy wektor do klasy 1, jeśli model dla tego wektora zwróci wartość bliską 1.

4.1 Przypomnienie regresji liniowej

Mając dany wektor danych $x = [x_1, x_2, ..., x_n]$ regresję liniową możemy opisać za pomocą wyrażenie

$$h_\Theta(x) = \sum_{n=1}^{N} \Theta_n * x_n + \Theta_0$$

Możemy więc przyjąć, że wektor danych x przekształcamy do postaci $x = [1, x_1, x_2, ..., x_n]$, dzięki czemu powyższe równanie sprowadza się do:

$$h_\Theta(x) = \Theta^T x$$

4.2 Sigmoid

Łatwo zauważyć, że wartości $h_\Theta(x)$ mogą przyjmować wartości z przedziału $(-\infty, \infty)$. Możemy jednak nałożyć na tą wartość sigmoid, czyli funkcję $\sigma(z) = \frac{1}{1+e^{-z}}$. Stąd:

$$h_\Theta(x) = \sigma(\Theta^T x) = \frac{1}{1 + e^{-\Theta^T x}}$$

4.3 Funkcja straty

W pierwszym kroku możemy zastanowić się, jak wygląda pochodna dla sigmoidu.

$$\begin{aligned} \sigma'(x) &= \frac{d}{dx} \frac{1}{1 + e^{-x}} \\ &= \frac{1}{(1 + e^{-x})^2} e^{-x} \\ &= \frac{1}{1 + e^{-x}} (1 - \frac{1}{1 + e^{-x}}) \\ &= \sigma(z)(1 - \sigma(z)) \end{aligned} \tag{4}$$

Następnie możemy przyjąć, że wartość $h_\Theta(x)$ jest równa prawdopodobieństwu, że x ma klasę 1. Możemy to zapisać w postaci:

$$P(y=1|x;\Theta) = h_\Theta(x)$$
$$P(y=0|x;\Theta) = 1 - h_\Theta(x)$$

Łącząc te 2 wyrażenia i pamiętając, że $y \in \{0,1\}$ możemy wyprowadzić następującą zależność:

$$P(y|x;\Theta) = (h_\Theta(x))^y(1-h_\Theta(x))^{1-y}$$

Następnie będziemy chcieli skorzystać z MLE (estymator największej wiarygodności). W tym celu potrzebujemy wyrażenia funkcji wiarygodności $L(\Theta)$, a dokładniej jej ujemną, zlogarytmowaną wartość nazywaną $nll(\Theta)$.

$$\begin{aligned} nll(\Theta) &= -\sum_{i=1}^{N} y^{(i)} \log \sigma(\Theta^T x^{(i)}) + (1-y^{(i)}) \log(1-\sigma(\Theta^T x^{(i)})) = \\ &= -\sum_{i=1}^{N} y^{(i)} \log p(y=1|x^{(i)};\Theta) + (1-y^{(i)}) \log p(y=0|x^{(i)};\Theta) \end{aligned} \tag{5}$$

Jest to często używana funkcja kosztu nazywana cross-entropy loss function. Czasem chcemy, by nasze parametry Θ nie przyjmowały zbyt dużej wartości, wtedy używamy regularyzacji i funkcja kosztu przyjmuje wartość:

$$nll(\Theta) = -\sum_{i=1}^{N} y^{(i)} \log p(y=1|x^{(i)};\Theta) + (1-y^{(i)}) \log p(y=0|x^{(i)};\Theta) + \frac{\lambda}{2}\sum_i \theta_i^2 \tag{6}$$

4.4 Gradient

Żeby znaleźć (wytrenować) odpowiednie parametry Θ musimy policzyć gradient funkcji straty ($nll(\Theta)$)

Policzmy początkowo gradient dla funkcji straty pojedynczego wektora, czyli $nll^{(i)}$. Niech $z^{(i)} = \Theta^T x^{(i)}$. Korzystając z reguły łańcuchowej (chain rule) wiemy, że:

$$\frac{\partial nll^{(i)}(\Theta)}{\partial \Theta} = \frac{\partial nll^{(i)}}{\partial z^{(i)}} \frac{\partial z^{(i)}}{\partial \Theta}$$

Możemy więc wykonać te obliczenia w dwóch krokach.

$$\begin{aligned}\frac{\partial nll^{(i)}}{\partial z^{(i)}} &= \frac{\partial - \left(y^{(i)} \log \sigma(z^{(i)}) + (1 - y^{(i)}) \log(1 - \sigma(z^{(i)}))\right)}{\partial z^{(i)}} \\ &= -y^{(i)} \frac{\sigma(z^{(i)})(1 - \sigma(z^{(i)}))}{\sigma(z^{(i)})} + (1 - y^{(i)}) \frac{\sigma(z^{(i)})(1 - \sigma(z^{(i)}))}{1 - \sigma(z^{(i)})} \\ &= -y^{(i)}(1 - \sigma(z^{(i)})) + (1 - y^{(i)})\sigma(z^{(i)}) \\ &= \sigma(z^{(i)}) - y^{(i)} \end{aligned} \tag{7}$$

$$\begin{aligned}\frac{\partial z^{(i)}}{\partial \Theta} &= \frac{\partial \Theta^T x^{(i)}}{\partial \Theta} \\ &= x^{(i)}\end{aligned}$$

Stąd:

$$\frac{\partial nll(\Theta)}{\partial \Theta} = \sum_{i=1}^{N} \frac{\partial nll^{(i)}}{\partial z^{(i)}} \frac{\partial z^{(i)}}{\partial \Theta} = \sum_{i=1}^{N} \left(\sigma(z^{(i)}) - y^{(i)}\right) x^{(i)}$$

Powyższe wyrażenie możemy przedstawić w postaci:

$$\frac{\partial nll(\Theta)}{\partial \Theta} = X \left(\sigma(Z) - Y\right)^T$$

4.5 Gradient descent (metoda gradientu prostego)

Gradient nie jest liniowo zależny od Θ, więc znalezienie rozwiązania nie będzie takie proste, jak w przypadki regresji liniowej. Zamiast tego będziemy robić niewielkie kroki w kierunku gradientu tak długo, aż nie znajdziemy wystarczająco dobrego rozwiązanie. Nasz algorytm uczenia będzie miał więc postać:

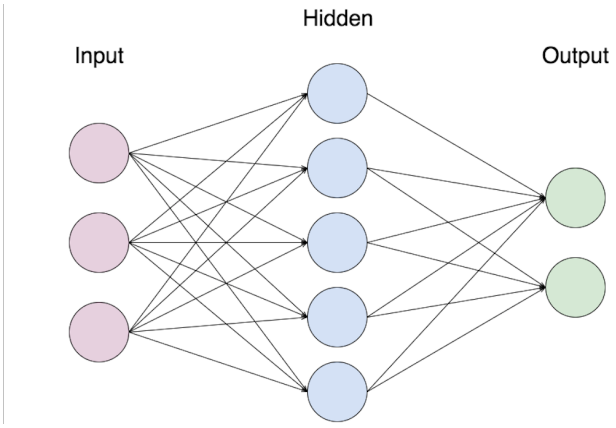
$\Theta \leftarrow$ wartość początkowa
while not bored: **do**
  $\Theta \leftarrow \Theta - \alpha \frac{\partial L(\Theta)}{\partial \Theta}$
end while

Gdzie α odpowiada za wielkość kroku, który robimy (zazwyczaj jest nazywana learning rate)

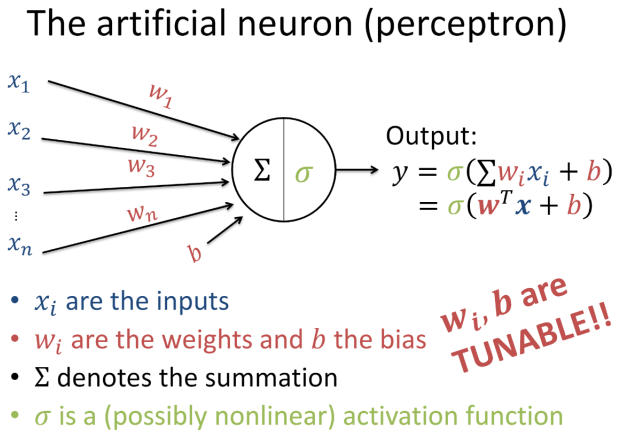
5 Sieci neuronowe

Sieć neuronowa jest zbudowana z wielu, połączonych ze sobą neuronów.

Pojedynczy neuron zachowuje się tak samo, jak regresja logistyczna, co więcej, tak samo się go uczy.



Rysunek 2: `https://towardsdatascience.com/classical-neural-network-what-really-are-nodes-and-layers-ec51c6122e09`



Rysunek 3: `https://github.com/janchorowski/dl_uwr/blob/summer2020/Lectures/01-intro.pdf`

5.1 Forward pass

Każda kropka po lewej stronie odpowiada za jeden element wejścia, np. jeden piksel obrazu. Następnie każdy neuron z pierwszej warstwy oblicza swoją wartość w sposób pokazany na obrazku wyżej. Trzeba zwrócić uwagę, że każdy neuron ma osobną wagę dla każdego wejścia. Druga warstwa powtarza te same obliczenia, jedynie zamiast patrzeć na wejście (np. obraz) patrzy na wynik pierwszej warstwy. Ostatni warstwa ma dokładnie tyle neuronów, ile mamy możliwych klas. Ostatecznie wejście dostaje klasę, której odpowiadający neuron z ostatniej warstwy miał największą wartość. Takie przejście nazywa się forward passem.

5.2 Backward pass

By być w stanie uczyć sieć musimy potrafić policzyć gradienty. Możemy to zrobić przechodząc ponownie przez sieć, ale w przeciwnym kierunku. Jeśli skorzystamy kilkukrotnie z reguły łańcuchowej oraz wiemy, jakie obliczenia wykonują kolejny elementy sieci możemy z łatwością wyznaczyć, jaki wpływ na ostateczny wyniki (a dokładniej na funkcje straty) mają poszczególne wagi. Mając takie informacji wiemy jak zmienić każdą z wag w taki sposób, by nasza funkcja była bliższa poprawnego wyniku.

Większość bibliotek do sieci neuronowych wspiera takie obliczenia, zapamiętując kolejne etapy forward passu, dzięki czemu możemy otrzymać odpowiedni gradient nie wykonując samemu żadnych obliczeń.

5.3 Uczenie sieci

Skoro pojedyncze neurony wykonują takie same obliczenia, jak regresja logistyczna, to ich uczenie również powinno być takie samo. I w praktyce w dużej części tak właśnie jest.

5.3.1 Gradient descent

Musimy zdefiniować sobie funkcję straty, może to być opisywana wcześniej cross-entropy loss, ale dla prostoty możemy zastosować sumę kwadratów. Stąd:

$$L(\Theta) = \frac{1}{2}\sum_{i=0}^{I}(Net(x^{(i)}, \Theta) - y^{(i)})^2$$

Czyli funkcja straty zależy od błędów popełnianych na wszystkich elementach zbioru treningowego.

Wówczas trening sieci wygląda tak samo, jak trening regresji logistycznej, czyli:

$\Theta \leftarrow$ wartość początkowa
while not bored: **do**
  $\Theta \leftarrow \Theta - \alpha\frac{\partial L(\Theta)}{\partial \Theta}$
end while

Gdzie α odpowiada za wielkość kroku, który robimy (zazwyczaj jest nazywana learning rate)

5.3.2 Stochastic gradient descent

W praktyce często stosuje się inną, chociaż podobną metodę. Zamiast obliczać błąd dla wszystkich elementów zbioru treningowego, a następnie jednokrotnie aktualizować parametry możemy aktualizować parametry po każdym elemencie, na podstawie błędu na nim popełnionego. Taką metodę nazywamy stochastic gradient descent (metoda gradientu stochastycznego).

$\Theta \leftarrow$ wartość początkowa
while not bored: **do**
    $\Theta \leftarrow \Theta - \alpha \frac{\partial L^{(i)}(\Theta)}{\partial \Theta}$
end while

Zalety takiej metody to szybkość oraz łatwiejsza implementacji dla bardzo dużych danych, które nie mieszczą się w RAM'ie.

W przypadku SGD należy zwrócić szczególną uwagę na dobór parametru α. W praktyce po każdym kroku parametry będą chciały zmieniać się w taki sposób, by zminimalizować błąd dla tego jednego elementu. Learning rate nie może być więc zbyt duży, by nie "wyskoczyćź optymalnego obszaru oraz wystarczająco duży, by powoli poprawiać skuteczność sieci.

W praktyce używa się wielu metod doboru oraz zmiany parametru α (np. α schedule).

6 Sieci splotowe (Convolutional network)

6.1 Warstwa splotowa

ConvNets - `https://cs231n.github.io/convolutional-networks/`

W sieciach splotowych (konwolucyjnych) pojedyńczy neuron zamiast "patrzećńa każdy piksel patrzy jedynie na pewien wycinek obrazu wejściowego. Jeśli obraz ma 3 kanały (R-G-B) wówczas pojedynczy piksel bierze informacje ze wszystkich kanałów w tym obszarze. Co więcej wszystkie piksele w danym kanale współdzielą między sobą wagi, chociaż patrzą na inne obszary.

W danej warstwie tworzy się zazwyczaj więcej niż jeden kanał. Wtedy dla każdego obszaru mamy kilka pikseli, które na nie patrzą. Każdy z tych pikseli ma własne wagi, wagi te są współdzielone w danym kanale, ale nie między nimi.

6.2 Pooling

Pooling polega na zmniejszeniu wymiarowości w ten sposób, że patrzymy na obszar o pewnym rozmiarze i dla każdego rozpatrywany obszaru zapisujemy min/max/avg z wartości w tym obszarze.

W praktyce stosuje się wiele warstw conv + pool jedna po drugiej, przeplatając je funkcjami nieliniowymi, jak np. ReLu. Na końcu stosuje się warstwy w pełni połączone, czyli takie, jak w prostych sieciach opisywanych na początku.

7 Wykrywanie obiektów

W celu wykrycia, czy w danym miejscu zdjęcia znajduje się jakiś obiekt (i ewentualnie jaki to jest obiekt) moglibyśmy zastosować prosty, ale bardzo nieefektywny algorytm polegający na próbie klasyfikacji każdego fragmentu obrazu o

wszystkich możliwych rozmiarach. Powstało jednak wiele metod pozwalających rozwiązać ten problem w bardziej efektywny sposób.

7.1 Fast R-CNN

Fast R-CNN polega na zastosowaniu kilku warstw konwolucyjnych w celu uzyskania mapy obiektów / cech. Nie stosujemy jednak warstw liniowych, co zrobilibyśmy w celu uzyskania predykcji co znajduje się na obrazku. Zamiast tego przyglądamy się miejscom, w których feature-map ma dużą wartość, ponieważ podejrzewamy, że to właśnie tam mogą znajdować się jakieś obiekty. Następnie dla wszystkich tych miejscu staramy się (również za pomocą sieci) określić czy coś się tam faktycznie znajduje, jak duże to jest oraz czym jest ten obiekt.

7.2 YOLO (you only look once)

Inną, mniej skuteczną, ale dużo szybszą metodą jest YOLO. Ona również wykorzystuje sieci splotowe, ale zamiast wybierać obszary, które są dla nas potencjalne interesujące, a następnie ich dalszą analizę stawia ona na inne podejście. Po stworzeniu features map dla każdego elementu tej mapy sieć próbuje dokonać predykcji, czy w danym miejscu znajduje się (a dokładniej zaczyna się) jakiś obiekt oraz jak duży on jest. Dla każdego piksela można również dokonać predykcji klasy. Łącząc te informacje wybieramy kilka pikseli o najbardziej pewnych predykcjach i na tej podstawie tworzymy wynikową detekcję. Metoda ta jest bardzo szybka (ale nie jest najskuteczniejsza), dzięki czemu jest wykorzystywana tam, gdzie konieczna jest praca w czasie rzeczywistym.

8 Źródła

Materiały z których korzystałem zostały umieszczone jako odnośniki do danych tematów. Oprócz nich korzystałem z wykładów do Machine Learning oraz Deep Learning autorstwa Doktora Jana Chorowskiego.

Źródła wszystkich grafik znajdują się pod nimi.