Głębokie sieci neuronowe.

Sprawozdanie z pracowni problemowej magisterskiej (PPMGR(A)).

autor: Szymon Bugaj
promotor: prof. Andrzej Pacut

13 lipca 2015

Zestaw stosowy auto-enkoderów odszumiających (ang. stacked denoising autoencoders) uczony do problemu klasyfikacjia na zbiorze cyfr pisanych odręcznie (MNIST [4]). Wpływ składu zbioru uczącego fazy pretreningu na rezultaty klasyfikacji.

Spis treści

1 Głęboki autoenkoder odszumiający

Stworzyłem implementację zestawu stosowego auto-enkoderów odszumiających (ang. stacked denoising autoencoder) zgodnie z dokumentem [2] korzystając z biblioteki Theano [6] (pythonowa biblioteka, której najważniejszą cechą jest możliwość transparentnego użycia na karcie graficznej - w trybie GPU wykorzystywać może bibliotekę cuDNN [7]).

Pod terminem głębokiego autoenkodera kryje się sieć o topologii perceptronu wielowarstwowego, którego wagi inicjowane są z wykorzystaniem auto-enkoderów (w fazie tej niewykorzystywana jest informacja o przynależności obiektów do klas - jest to uczenie nienadzorowane). Faza ta nazywana jest fazą pretreningu (ang. pretrening). Po tej fazie następuje standardowe uczenie perceptronu wielowarstwowego i nazywane jest w tym kontekście fazą strojenia (ang. fine-tuning).

Autoenkoder to również sieć o topologii perceptronu wielowarstwowego. Uczony jest w trybie nienadzorowanym - niewykorzystywana jest informacja o przynależności obiektów do klas, jeżeli takowa jest dostępna. Liczba elementów w warstwie wejściowej równa jest liczbie neuronów w warstwie wyjściowej oraz funkcja kosztu dobrana jest w ten sposób, by sieć uczyła się replikować na wyjściu zbiór wejściowy. Celem tego jest ekstrakcja cech w zbiorze wejściowym.

Do uczenia sieci zarówno w fazie pretreningu (ang. pretrening) jak i w fazie dostrajania (ang. fine-tuning) wykorzystałem metodę stochastycznego lub zbiorczego spadku gradientu (ang. stochastic gradient descent).

1.1 Metoda stochastycznego spadku gradientu

Metodę stochastycznego spadku gradientu opisana jest równianiami:

- 1 i 3 - uczenie nadzorowane (faza dostrajania),
- 2 i 3 - uczenie nienadzorowane dla autoenkodera (faza pretreningu).

Przyjęte zostały oznaczenia:

- x^t - pojedynczy element ze zbioru wejściowego,
- y^t - etykieta reprezentująca klasę elementu ze zbioru wejściowego w przypadku uczenia nadzorowanego,
- θ - zbiór wszystkich parametrów sieci (wagi połączeń między elementami w sieci),
- l - funkcja kosztu, porównująca wyjście sieci ($f(x^t)$) z oczekiwaną wartością, malejąca wraz z zmniejszaniem się rozbieżności między tymi dwoma wartościami,
- Ω - funkcja, której zadaniem jest wymuszenie zadanego rozkładu parametrów sieci (np. równomierny rozkład wartości parametrów), nazywana regularyzatorem,
- λ - stała, waga przypisana regularyzatorowi,
- α - stała, krok uczenia.

$$\Delta = -\nabla_\theta l(f(x^{(t)}); \theta), y^{(t)}) - \nabla_\theta \lambda \Omega(\theta) \tag{1}$$
$$\Delta = -\nabla_\theta l(f(x^{(t)}); \theta), x^{(t)}) - \nabla_\theta \lambda \Omega(\theta) \tag{2}$$
$$\theta \leftarrow \theta + \alpha\Delta \tag{3}$$

1.2 Wejściowe zbiory danych

Wyodrębnić należy 4 zbiory danych, wykorzystane w procesie uczenia i testowania sieci:

- zbiór uczący fazy pretreningu,
- zbiór uczący fazy dostrajania (musi być etykietowany),
- zbiór walidacyjny fazy dostrajania (musi być etykietowany),
- zbiór testowy (musi być etykietowany).

Zbiór testowy i walidacyjny fazy dostrajania powinny być rozłączne z innymi zbiorami danych. Zbiór uczący fazy pretreningu jako, że wykorzystane jest w tej fazie uczenie nienadzorowane, może być zbiorem nieetykietowanym.

1.3 Losowanie początkowych parametrów sieci

Przed fazą losowane są parametry początkowe sieci. Przyjąłem następującą metodę wyboru początkowych wartości:

- wyrazy wolne (ang. bias) są inicjowane 0,
- wagi połączeń między neuronami losowane były z rozkładem jednostajnym z przedziału

$$(-\sqrt{6/(N_{h^{k-1}} + N_{h^k})}, \sqrt{6/(N_{h^{k-1}} + N_{h^k})})$$

1.4 Pretrening

Autoenkodery odszumiające z jedną warstwą ukrytą zostały użyte do inicjacji parametrów warstw ukrytych sieci. Tj. pojedynczy autoenkoder inicjował jedną warstwę ukrytę. Uczenie odbywało się w sposób zachłanny, niezależnie dla kolejnych sąsiadujących warstw poczynając od tej sąsiadującej z warstwą wejściową. Wejściem dla każdego autoenkodera było wyjście z poprzedniej warstwy uprzednio zainicjowanej. Wejściem dla pierwszego autoenkodera stanowił zbiór uczący fazy pretreningu.

Rozpatrzmy przykładową sieć: 784-500-500-500-10 (liczność kolejnych warstw sieci począwszy od wejściowej). Na początku losowane są parametry początkowe sieci. Następnie odbywa się faza pretreningu.

Uczona jest pierwsza warstwa ukryta. Tworzony jest autoenkoder 784-500-784 i jego wejściem jest zbiór wejściowy fazy pretreningu, a macierz wag warstwy ukrytej to macierz z głębokiej sieci, na której odbywa się pretrening.

Po zakończeniu uczenia tworzony jest autoenkoder mający uczyć drugą warstwę ukrytą. Liczność warstw wynosi 500-500-500. Wejściem autoenkodera stanowi wyjście z pierwszej warstwy ukrytej policzone już po uczenie tej warstwy w fazie pretreningu.

Tak samo się dzieje dla trzeciej warstwy ukrytej. Warstwa wyjściowa nie jest uczona w fazie pretreningu.

1.4.1 Autoenkoder odszumiający

Funkcja kosztu dla autoenkodera jest zdefiniowany, by wyjście autoenkodera stanowiło replikę wejścia. Ostatecznym problemem do rozwiązania jest klasyfikacja nowych elementów, jaki jest więc sens wykorzystania autoenkodera? Celem jest ekstrakcja charakterystycznych cech w zbiorze uczącym, tak by parametry sieci ograniczyć do rozpatrywanej podprzestrzeni problemu.

W dalszej części podrozdziału przedstawiam rozważania dla pojedynczego autoenkodera użytego do uczenia pojedynczej warstwy ukrytej głębokiej sieci neuronowej.

Autoenkodery zawierały jedną warstwę ukrytą.

Przyjąłem oznaczenia:

- x - zbiór wejściowy autoenkodera i N_x jego liczność,
- N_D - liczba cech dla elementów zbioru wejściowego,
- $\widetilde{x}$ - zaszumiony zbiór wejściowy,
- h - wyjście z warstwy ukrytej (funkcje przekształcające zaszumiony zbiór wejściowy w wyjście warstwy ukryteh),
- $\hat{x}$ - wyjście z wartswy wyjściowej, zrekonstruowane wejście (funkcja przekształcająca zaszumiony zbiór wejściowy w wyjście warstwy wyjściowej),
- W, W^* - zbiór połączeń odpowiednio dla warstwy ukrytej i wyjściowej.
- l - funkcja kosztu.

Zaszumienie zbioru wejściowego autoenkodera polepsza rezultaty autoenkodera w wyżej wymienionym zadaniu i polegało na jego losowej modyfikacji - dla każdego wektora wejściowego i dla każdego elementu tego wektora (cechy) losowano z zadanym stałym prawdopodobieństwem czy wartość ma zostać zmodyfikowana na 0.

Dla pierwszej warstwy ukrytej autoenkodera głębokiego oznaczało to modyfikację obrazów wejściowych poprzez zmianę losowych pikseli na czarne, dla drugiej modyfikację wyjścia z warstwy pierwszej poprzez zmianę wyjść losowo wybranych neuronów na 0.

Przy przyjętych oznaczeniach metodę zaszumienia można zapisać następująco:

foreach *i in* N_x **do**
  foreach *d in* N_D **do**
    if *random_binomial (corrupt)* **then**
      $\widetilde{x}_d^i = 0$;
    else
      $\widetilde{x}_d^i = x_d^i$
    end
  end
end

Za funkcję aktywacji dla warstwy ukrytej jak i dla warstwy wyjściowej przyjęłem funkcję sigmoidalną (*sigmoid*).

Poniższe wzory przedstawiają rachunek macierzowy. Przykłady ułożone są w wierszach, a kolumny reprezentują cechy (odpowiednie piksele obrazka) w macierzy x.

$$h(x) = sigmoid(\widetilde{x} * W + b) \tag{4}$$
$$\hat{x}(x) = sigmoid(h(x) * W^* + c) \tag{5}$$

Dla warstwy wyjściowej macierz wag stanowi transpozycję macierzy wag warstwy ukrytej (ang. tied weights).

$$W^* = W^T \tag{6}$$

Użyłem funkcję kosztu postaci [**?**]:

$$l(f(x)) = -\sum_k (x_k log(\hat{x}_k) + (1 - x_k) log(1 - \hat{x}_k)) \tag{7}$$

Należy zaznaczyć, iż w powyższym równaniu porównywane są wartości:

- x - oryginalny zbiór wejściowy autoenkodera (bez szumu),
- $\hat{x}(x)$ - zrekonstruowany zbiór wejściowy (wyjście neuronów w warstwie ukrytej autoenkodera).

Uczenie odbywa się na zbiorze uczącym fazy pretreningu i kończone jest po zadanej liczbie epok.

1.5 Strojenie

Po etapie pretreningu, dla uzyskanych wartości połączeń dla warstw ukrytych odbywa się standardowe uczenie perceptronu wielowarstwowego.

Przyjąłem oznaczenia:

- g - funkcja aktywacji dla warstwy ukrytej,
- o - funkcja aktywacji dla warstwy wyjściowej,
- l - funkcja kary,
- h^i - funkcja transformująca zbiór wejściowy w wyjście dla i-tej warstwy ukrytej,
- $f(x) = h^{L+1}(x)$
- x, y - zbiór wejściowy i zbiór etykiet określających prawidłową klasę,
- L - liczba warstw ukrytych,
- W^i - macierz wag połączeń między neuronami między warstwą $i-1$ (dla $i = 0$ otrzymujemy warstwę wejściową) a warstwą i ($i = 1, 2...L+1$),
- b^i - wektor wag połączeń między neuronami z warstwy i a wyrazem wolnym z warstwy $i-1$

Poniższe wzory przedstawiają rachunek macierzowy. Przykłady ułożone są w wierszach, a kolumny reprezentują cechy (odpowiednie piksele obrazka) w macierzy x (by mnożenie macierzy miało sens, każda kolumna w macierzy połączeń W^i zawierać musi wagi połączeń wszystkich neuronów z warstwy $i-1$ z pojedynczym neuronem warstwy i)

$$
\begin{aligned}
&h^{k+1}(x) = g(h^k(x) * W^k + b^k) \\
&h^0(x) = x \\
&h^{L+1}(x) = o(h^k(x) * W^k + b^k)
\end{aligned}
$$

W modelu przyjęto funkcje:

$$
\begin{aligned}
&softmax(a_k) = \frac{e^{a_k}}{\sum_i e^{a_i}}, a = [a_1, a_2, ...] \\
&sigmoid(a) = \frac{1}{1+e^{-a}} \\
&g(a) = sigmoid(a) \\
&o(a) = softmax(a) \\
&l(f(x), y) = -logf(x)_y, f(x) = [f(x)_1, f(x)_2, ...]
\end{aligned} \tag{8}
$$

By zapobiec zjawisku nadmiernego dopasowania po każdej epoce zbioru uczącego fazy dostrajania model sprawdzany jest na zbiorze walidacyjnym fazy dostrajania.

Jeżeli przez zadaną liczbę epok (oznaczany dalej symbolem $P_{fine-tuning}$) model nie poprawia najlepszego dotychczasowego rezultatu na zbiorze walidacyjnym optymalizacja jest zakończana, a rezultatem jest model, który uzyskał najlepszy wynik na tymże zbiorze. Następnie model ten jest sprawdzany na zbiorze testowym.

2 Testy

Możliwości klasyfikacyjne sieci testowane były na zbiorze cyfr pisanych MNIST [4] o wielkości 10000 elementów, który to był rozłączny z wszystkimi innymi zbiorami danych wykorzystanych w procesie uczenia.

Zbiór uczący fazy dostrajania miał 50000 elementów, a zbiór walidacyjny fazy dostrajania 10000 elementów. Wszystkie trzy wspomniane zbiory zawierały dane wyłącznie ze zbioru MNIST, czyli były to dane zawierające cyfry i były etykietowane.

W części uruchomień faza pretreningu była pominięta. Oczywiście w takim przypadku mamy do czynienia ze standardowym uczeniem perceptronu wielowarstwowego.

2.1 Warianty składu zbioru uczącego fazy pretreningu

W testach uwzględnione zostały następujące warianty składu zbioru uczącego fazy pretreningu:

- pretrening na zbiorze uczącym fazy dostrajania (50 000 cyfr z MNIST),
- pretrening na zbiorze uczącym fazy dostrajania rozszerzonym o zbiór liter (50 000 cyfr z MNIST + 120 000 liter z NIST [5], czyli 170 000 przykładów),
- pretrening wyłącznie na zbiorze liter (50 000 liter z NIST).

Zbiór liter został przetworzony w sposób odpowiadający opisowi sposobu przetworzenia zbioru cyfr użytych do konstrukcji zbioru MNIST, tj.

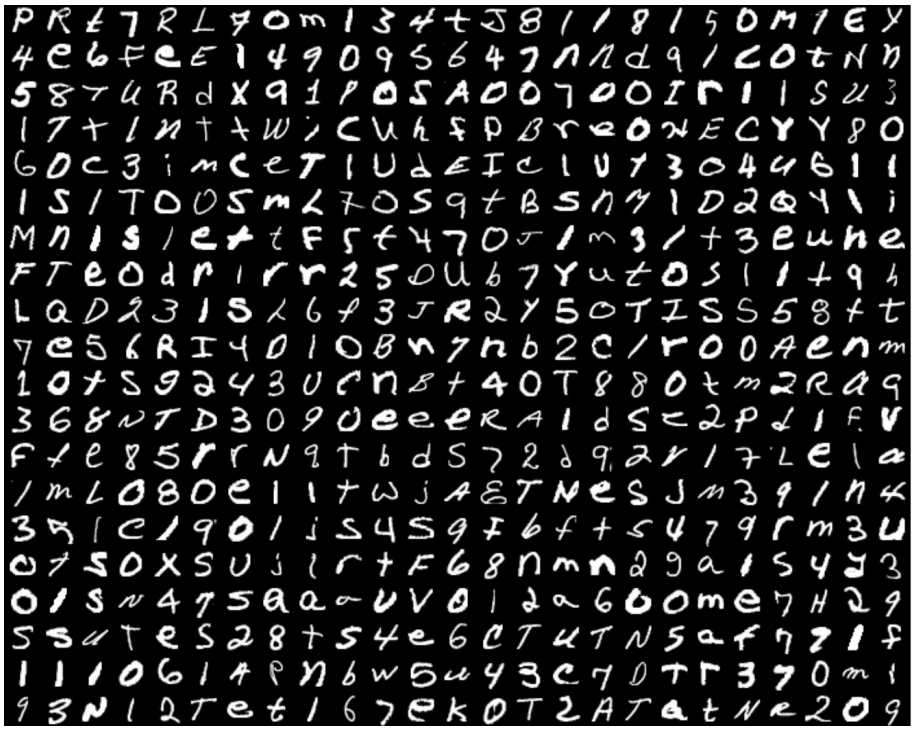
- odwrócenie kolorów,
- skadrowanie obrazka do najmniejszego prostokąta zawierającego znak,
- zmniejszenie obrazka, tak by dłuższy bok miał 20 pikseli,
- rozszerzenie wielkości obrazka do 28x28pikseli,

- normalizacja wartości jasności pikseli do zakresu 0..1.

Pominięty został krok translacji obrazka by tak na środku obrazka znajdował się środek ciężkości obrazka otrzymanego po etapie zmniejszania rozmiaru.

Stworzony został zbiór 120 000 liter.

Rysunek 1: Zbiór uczący fazy pretreningu zawierający zarówno literki jak i cyfry.



2.2 Metaparametry

Wszystkie testy zostały przeprowadzone na sieci o liczności warstw $784-500-500-500-10$ (dobrane zostały w testach nie zamieszczonych w tym sprawozdaniu).

Niżej wymienione są wszystkie metaparametry przyjęte w testach:

- początkowe wagi neuronów dla k warstwy ukrytej losowane były z rozkładem jednostajnym z przedziału $(-\sqrt{6/(N_{h^{k-1}}+N_{h^k})}, \sqrt{6/(N_{h^{k-1}}+N_{h^k})})$, a wyrazy wolne inicjowane 0.
- autoenkoder, faza pretreningu:
  - $g(a) = sigmoid(a)$,
  - $o(a) = sigmoid(a)$,
  - $W^* = W^T$,
  - $l(f(x)) = -\sum_k, (x_k log(\hat{x}_k) + (1-x_k) log(1-\hat{x}_k))$,
  - $corrupt = [0.3, 0.3, 0.3]$ - prawdopodobieństwo zaszumienia dla autoenkoderów dla kolejnych warstw,
  - do uczenia zastosowano metodę stochastycznego spadku gradientu,
  - $\alpha_{pretrain} = 0.01$ - krok dla metody gradientowej,
  - uczenie kończone po 15 epokach.
- głęboki autoenkoder, faza strojenia
  - $g(a) = sigmoid(a)$,
  - $o(a) = softmax(a)$,
  - $l(f(x), y) = -log f(x)_y$,
  - do uczenia zastosowano metodę stochastycznego spadku gradientu,

Liczba cyfr	Liczba liter	avg	std	min	$t_{avg}[m]$
0	0	1.86%	0.06%	1.77%	47
$50k$	0	1.55%	0.05%	1.48%	66
0	$50k$	1.58%	0.08%	1.41%	65
$50k$	$120k$	1.59%	0.045%	1.56%	132

Tabela 1: Jakość klasyfikacji i czasy uczenia dla zestawu stosowego auto-enkoderów odszumiających z trzema warstwami ukrytymi $784-500-500-500-10$. Dla każdego wariantu wykonano 10 uruchomień. Implementację stworzona z wykorzystaniem biblioteki Theano i uruchomiono na czterordzeniowym CPU Intel Core i7.

- $\alpha_{finetuning} = 0.01$ - krok dla metody gradientowej,
- $P_{fine-tuning} = 10$ - uczenie kończone, gdy przez 10 iteracji nie ma poprawy najlepszego wyniku na zbiorze walidacyjnym.

2.3 Wpływ pretreningu na głęboką sieć neuronową

Dla każdego zestawienia metaparametrów wykonane zostało 10 uruchomień. Następnie policzony średni czas uczenia, średni błąd klasyfikacji, odchylenie średniokwadratowe dla błędów klasyfikacji, minimalny błąd klasyfikacji. Tabela 1 zawiera wyniki.

Literatura

[1] Bengio, P. Lamblin, D. Popovici and H. Larochelle, Greedy Layer-Wise Training of Deep Networks, in Advances in Neural Information Processing Systems 19 (NIPS'06), pages 153-160, MIT Press 2007.

[2] Vincent, H. Larochelle Y. Bengio and P.A. Manzagol, Extracting and Composing Robust Features with Denoising Autoencoders, Proceedings of the Twenty-fifth International Conference on Machine Learning (ICML'08), pages 1096 - 1103, ACM, 2008.

[3] Hugo Larochelle http://info.usherbrooke.ca/hlarochelle/

[4] The MNIST database of handwritten digits, Yann LeCun, Corinna Cortes, Christopher J.C. Burges, http://yann.lecun.com/exdb/mnist/

[5] NIST Special Database 19, http://www.nist.gov/srd/nistsd19.cfm

[6] Biblioteka Theano, http://deeplearning.net/software/theano/

[7] NVIDIA® cuDNN – GPU Accelerated Deep Learning library, https://developer.nvidia.com/cuDNN