STUDIA INFORMATICA 2012
Volume 33 Number 2A (105)

Dariusz R. AUGUSTYN, Kamil BADURA
Politechnika Śląska, Instytut Informatyki

REALIZACJA PRZETWARZANIA W CHMURZE OBLICZENIOWEJ NA PRZYKŁADZIE SYSTEMU UCZENIA SIECI NEURONOWEJ OPARTEGO NA TECHNOLOGII MICROSOFT WINDOWS AZURE

Streszczenie. W artykule przedstawiono budowę systemu uczenia sieci neuronowej, opartego na koncepcji przetwarzania w chmurze obliczeniowej. Implementacja systemu bazuje na technologii Microsoft Windows Azure. W rozwiązaniu zastosowano znany algorytm uczenia – metodę wstecznej propagacji błędu – dostosowany do rozproszonego sposobu realizacji. Zaproponowano architekturę systemu, w której wykorzystano współpracujące procesy (instancje) typu *WorkerRole*. W opracowaniu przedstawiono sposób wykorzystania różnych metod magazynowania danych, dostępnych przez mechanizmy *Windows Azure Table*, *Queue*, *Blob Storage*. Równoległe przetwarzanie systemu zostało zapewnione nie tylko dzięki zastosowaniu wielu procesów *WorkerRole*, ale również dzięki wykorzystaniu modułu *Parallel Extension for .NET* przy implementacji kodu *WorkerRole*.

Słowa kluczowe: przetwarzanie w chmurze obliczeniowej, przetwarzanie równoległe, Microsoft Windows Azure, uczenie sieci neuronowej

THE MICROSOFT WINDOWS AZURE-BASED SYSTEM FOR NEURAL NETWORK LEARNING AS AN EXAMPLE OF CLOUD PROCESSING APPLICATION

Summary. The paper presents the system for neural network learning based on the idea of Cloud computing. System implementation uses Microsoft Windows Azure technology. The well-known learning algorithm i.e. back propagation method was adopted for parallel and distributed execution. The architecture of cooperative worker role processes was proposed. The paper describes applying of methods of data storage like Windows Azure Table, Queue, Blob. The advantages of parallelization result from either applying multiple processes (instances) of WorkerRoles or applying Parallel Extension for .NET module in WorkeRole's implementation.

Keywords: cloud computing, parallel processing, Microsoft Windows Azure, neural network learning

1. Wstęp

Koncepcja przetwarzania danych w tzw. chmurze obliczeniowej (ang. *cloud comuping*) to kolejny, istotny etap rozwoju zastosowań Internetu. Technologia obliczeń w chmurze stwarza nowe możliwości w wielu zakresach, np. wydajności (skalowalność na potencjalnie olbrzymią liczbę węzłów przetwarzających) czy bezpieczeństwa (automatyczne tworzenie replik danych). Nowy model przetwarzania pozwala na dostosowanie infrastruktury do potrzeb efektywnościowych przez dostosowanie mocy obliczeniowej wirtualnej infrastruktury do aktualnego zapotrzebowania wynikającego z obciążenia.

Technologie przetwarzania dostarczają nowych metod magazynowania danych (np. takich jak systemy NoSQL). Dodatkowo niektóre z technologii umożliwiają tworzenie zrównoleglonych systemów aplikacyjnych, działających w środowisku rozproszonym (nowy model programowania).

Systemem spełniającym opisane wymagania są technologia i infrastruktura Microsoft Windows Azure. Wykorzystanie własności Windows Azure umożliwiło utworzenie wydajnego i skalowalnego systemu uczenia sztucznej sieci neuronowej. Oczywiście wymagało to przystosowania algorytmu uczenia sieci (powszechnie znanej metody wstecznej propagacji błędu) do równoległego trybu obliczeń.

Celem niniejszego artykułu jest omówienie zaproponowanej architektury zrównoleglonego systemu uczenia sieci neuronowej, opartego na technologii Windows Azure. W szczególności chodzi o prezentację sposobu obsługi wielozadaniowości oraz doboru metod magazynowania danych.

Chociaż system ma bardzo konkretne zastosowanie (skalowalna usługa uczenia sieci), to sama architektura (w szczególności obsługa szeregowania podzadań) może być wykorzystana do budowy innych aplikacji, przetwarzających dane w sposób rozproszony z użyciem Windows Azure.

2. Wprowadzenie do technologii Windows Azure

Microsoft Windows Azure [1, 2] jest technologią i infrastrukturą programowo-sprzętową, umożliwiającą tworzenie efektywnych aplikacji, działających w ramach tzw. chmury oblicze-

niowej. Firma Microsoft udostępnia zarówno docelowe środowisko uruchomieniowe, dostępne przez Internet, jak i środowisko lokalne – *Windows Azure Emulator*.

Microsoft Corporation „dostarcza" płatną infrastrukturę *Windows Azure*, zapewniającą skalowalność i bezpieczeństwo przetwarzanych danych (system automatycznego tworzenia kopii zapasowych). Jednym z jej elementów są usługi przechowywania danych – *Windows Azure Storage* [3]. W ramach tych usług wyróżnia się:

- *Azure Blob* – dającą możliwość przechowywania plików binarnych, zorganizowanych w ramach tzw. kontenerów,
- *Azure Queue* – umożliwiającą kolejkowanie danych (z zachowaniem pewności dokończenia wykonania zadania pobierającego dane z kolejki),
- *Azure Table* – pozwalającą na przechowywanie prostych danych strukturalnych, zorganizowanych wg koncepcji klucz-wartość,
- *SQL Azure* – udostępniającą usługi relacyjnego systemu zarządzania bazą danych, w znacznym zakresie kompatybilną z SZBD SQL Server.

Azure Blob jest wydajnym rozwiązaniem przeznaczonym do obsługi danych niestrukturalnych. W *Azure Table* [4] obsługiwane są tzw. tablice, co powala na odzwierciedlenie bardzo prostego relacyjnego modelu danych. W tablicach przechowywane są tzw. encje (w uproszczeniu, odpowiedniki wierszy w podejściu relacyjnym). Struktura encji może, ale nie musi, być jednakowa w ramach tablicy. Struktura encji określona jest przez własności (pola) (w uproszczeniu, odpowiedniki kolumn w podejściu relacyjnym). Klucz tablicy jest dwuelementowy i składa się z własności: *PartitionKey* i *RowKey*. Wyszukiwanie i sortowanie wg klucza jest szybkie, wyszukiwanie wg pozostałych własności może być niewydajne. Mechanizmy skalowania usług przechowywania *Azure Table* wykorzystują własność *PartitionKey* (dane o tej samej wartości *PartitionKey*, w przypadku wystąpienia spadku wydajności, mogą być obsługiwane przez specjalnie dla nich wydzielony węzeł przetwarzający). Z przedstawionych faktów wynika, że ustalenie, jakie (w sensie merytorycznym) dane będą przechowywane w ramach klucza, jest szczególnie istotne z punktu widzenia całego rozwiązania. Właśnie z takich powodów identyfikator konkretnego zadania uczącego sieć neuronową jest umieszczany w *PartitionKey* we wszystkich *Azure Table* wykorzystanych w omawianym systemie uczenia sieci.

Model kosztowy usług *Windows Azure Storage* jest skonstruowany tak, że użycie *Azure Table* i *Blob* jest tańsze niż wykorzystanie *SQL Azure*. Mechanizmy dostarczane w ramach usług *Azure Table* i *Blob* są proste, ale za to szybkie (w porównaniu do *SQL Azure*). Oczywiście w przypadku tworzenia aplikacji o istotnie złożonym modelu danych użycie *SQL Azure* może być łatwiejsze dla projektantów i programistów tworzących chmurowe systemy informatyczne. W omawianym systemie uczenia sieci neuronowych taka konieczność nie wystąpiła – funkcjonalność *Azure Table* i *Blob* wystarczyła.

Microsoft Windows Azure to również nowy model programowania systemów informatycznych przeznaczonych do przetwarzania w chmurze obliczeniowej. Wzrost wydajności systemów działających na podstawie Windows Azure (w porównaniu do klasycznych systemów) może wynikać z możliwości zrównoleglenia i rozpraszania przetwarzania/obliczeń. W *Windows Azure* wprowadza się pojęcie instancji, która oznacza maszynę wirtualną (z systemem operacyjnym Windows) z uruchomionym procesem przetwarzającym. Rozróżniane są dwa rodzaje instancji: *WebRole* – instancja, w której procesem przetwarzającym jest uruchomiona aplikacja lub usługa Web (utworzona w technologii ASP.NET), oraz *WorkerRole* – instancja z uruchomionym procesem przetwarzającym (bez obsługi interakcji z użytkownikiem, funkcjonalność „niewizualna", realizowana na podstawie technologii .NET, ale również Tomcat/Java czy PHP).

Przyspieszenie obliczeń można osiągnąć przez odpowiednie przystosowanie (zrównoleglenie) algorytmu przetwarzającego (np. przez podział na niezależne podzadania) i uruchomienie go w konfiguracji z kilkoma instancjami/procesami typu *WorkerRole*. Taki sposób został wykorzystany w omawianym systemie uczenia sieci neuronowej.

Z kolei zwielokrotnienie instancji/procesów *WebRole* (odpowiedzialnych za kontakt z użytkownikiem systemu) może pozwolić na zwiększenie dostępności systemu dla użytkowników końcowych (w sytuacji, w której jednocześnie występuje wielu użytkowników systemu informatycznego). Takie działanie, które z braku potrzeb polegałoby jedynie na zmianach konfiguracyjnych (tzn. zwiększaniu liczby instancji/procesów typu *WebRole*), nie byłoby przedmiotem badań przy uruchamianiu systemu uczenia sieci neuronowej.

3. Algorytm uczenia sieci neuronowej – metoda wstecznej propagacji błędu dostosowana do przetwarzania współbieżnego

Standardowa metoda wstecznej propagacji błędu uczenia [5] zakłada aktualizacje wag sieci po uwzględnieniu każdego przykładu uczącego. To utrudnia jej bezpośrednie użycie przy tworzeniu systemu, w którym zakłada się zrównoleglenie obliczeń. Jest to spowodowane faktem, że dla każdego przykładu uczącego obliczenia zależą od wyników obliczeń przeprowadzonych dla poprzedniego przykładu.

Takiego ograniczenia nie ma zmodyfikowany algorytm uczenia o nazwie *batch learning* [6], który polega na kumulacji zmiany wag (kumulacja delt wag). W algorytmie tym zakłada się podział zbioru uczącego na rozłączne podzbiory (bloki). Następnie dla każdego podzbioru stosuje się podejście *back propagation*, ale obliczane są jedynie delty wag (nie zachodzi modyfikacja samych wag w sieci neuronowej). Dopiero po zakończeniu przetwarzania wszyst-

kich bloków następuje aktualizacja wag, co jest równoważne z zakończeniem przetwarzania w ramach jednej epoki uczenia.

Dalej zostanie opisana wersja zastosowanego algorytmu uczenia sieci neuronowej (z N-1 warstwami ukrytymi). Metoda ta zostanie przedstawiona dokładniej w zakresie czynności realizowanych w ramach pojedynczej epoki uczenia.

W ramach procesu propagacji w przód dla każdego i-tego neuronu w r-tej warstwie obliczana jest suma wag pomnożonych przez wejścia do neuronu:

$$s_i^{<r>} = \sum_{j=1}^{J^{<r-1>}} w_{ij}^{<r>} y_j^{<r-1>} - t_i^{<r>}, \tag{1}$$

gdzie: $i = 1..\ J^{<r>}$, $r = 0..\ N$; $J^{<r\text{-}1>}$ – liczba neuronów w warstwie poprzedniej; $w_{ij}^{<r>}$ – waga na połączeniu neuronu z j-tym neuronem warstwy poprzedniej; $t_i^{<r>}$ – tzw. bias, próg przełączenia; $y_j^{<r-1>}$ – wyjście j-tego neuronu z warstwy poprzedniej ($y_j^{<0>}$ oznacza j-te wejście sieci neuronowej).

Na podstawie (1) wyznaczane jest wyjście i-tego neuronu w r-tej warstwie:

$$y_i^{<r>} = \frac{1}{1 - \exp(-\beta s_i^{<r>})}, \tag{2}$$

gdzie $\beta = 1$ jest stałym współczynnikiem sigmoidalnej funkcji aktywacji neuronu.

W wyniku propagacji w przód, realizowanej dla każdego przykładu uczącego z bloku wektorów uczących, następuje porównanie wyznaczonych wartości wyjść sieci neuronowej $y_i^{<N>}$ z wartościami oczekiwanymi na wyjściu d_i, czyli wyznaczenie błędu w każdym z neuronów wyjściowych:

$$\delta_i^{<N>} = (y_i^{<N>} - d_i)\frac{dy_i^{<N>}}{ds_i^{<N>}}, \tag{3}$$

gdzie $\frac{dy_i^{<N>}}{ds_i^{<N>}}$ jest pochodną funkcji aktywacji, która dla funkcji sigmoidalnej wynosi:

$$\frac{dy_i^{<N>}}{ds_i^{<N>}} = \beta(1 - y_i^{<N>})y_i^{<N>}. \tag{4}$$

W ramach propagacji wstecznej wartość błędu wyjściowego jest dystrybuowana na pozostałe neurony:

$$\delta_i^{<r>} = \frac{dy_i^{<r>}}{dn_i^{<r>}} \sum_{k=1}^{J^{<r+1>}} \delta_k^{<r+1>} w_{ki}^{<r+1>}. \tag{5}$$

Propagowanie błędu przebiega od warstwy przedostatniej do pierwszej, tzn. $r = N\text{-}1..\ 1$.

Jednocześnie dla każdego połączenia w sieci wyznaczona jest zmiana wag:

$$\Delta w_{ij}^{<r>} = \eta \delta_i^{<r>} y_j^{<r-1>}, \tag{6}$$

gdzie $r = 1..\ N$ i $\eta = \text{const}$. (Oczywiście metoda pozwala również na wyznaczanie zmian współczynnika progu przełączenia w każdym z neuronów).

W ramach bloku uzyskane delty wag dla każdego połączenia są sumowane:

$$\Delta w_{ij<nr\ bloku>}^{<r>} = \sum_{\text{po\_wszystkich\_przykładach\_uczących\_w\_bloku\_o\_zadanym\_numerze}} \Delta w_{ij}^{<r>}. \tag{7}$$

Na tym etapie sumowany jest też błąd na wyjściu sieci neuronowej dla każdego przykładu uczącego, należącego do bloku. Czynności realizowane w ramach przetwarzania bloku są względem siebie niezależne i mogą być realizowane w sposób równoległy.

Po przetworzeniu wszystkich bloków (uwzględnienie całego zbioru uczącego – koniec epoki) następuje agregacja delt wag z bloków uczących (zsumowanie):

$$\Delta w_{ij<ALL>}^{<r>} = \sum_{\text{po\_wszystkich\_blokach}} \Delta w_{ij<nr\ bloku>}^{<r>}. \tag{8}$$

Na tym etapie następuje również agregacja błędów z bloków uczących (zsumowanie) – wyznaczenie błędu dla bieżącej epoki.

W omawianym rozwiązaniu wykorzystano metodę *momentum* („zmiana wag z bezwładnością"), tzn. ostateczne wartości delt wag w aktualnej epoce wyznaczane są z uwzględnieniem zmian wag z epoki poprzedniej:

$$\Delta w_{ij}^{<r><e>} = \Delta w_{ij<ALL>}^{<r>} + \mu \Delta w_{ij}^{<r><e-1>}, \tag{9}$$

gdzie: e jest numerem bieżącej epoki, $\mu = \text{const}$.

Wyznaczony zagregowany błąd uczenia dla danej epoki służy do określenia, czy spełnione jest kryterium zakończenia całego procesu (tzn. czy osiągnięto zadowalającą dokładność uczenia).

4. Opis zaproponowanego rozwiązania – architektura systemu przetwarzającego w chmurze obliczeniowej

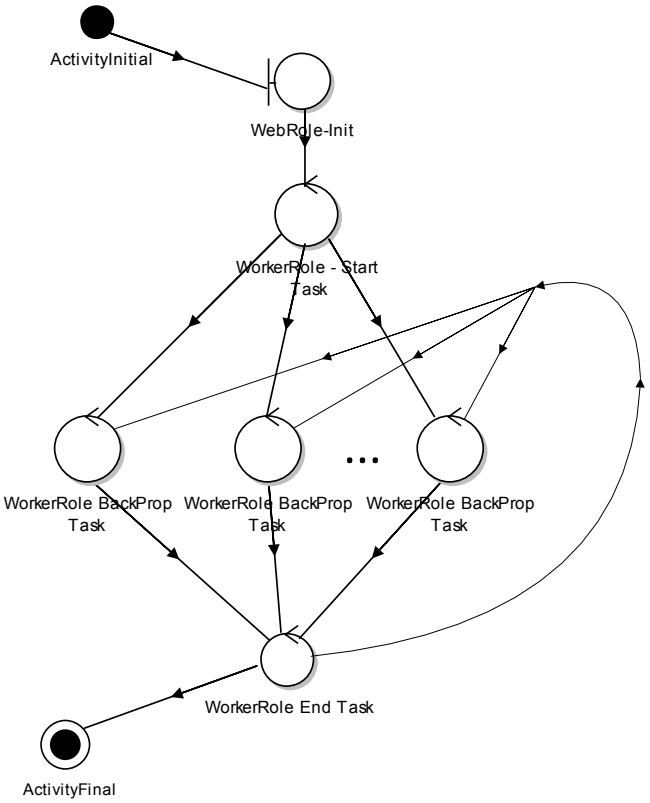
Architektura systemu uczenia zakłada możliwość pracy w trybie zrównoleglonym i środowisku rozproszonym. Korzyści efektywnościowe zastosowania takiej architektury wynikają z faktu, że podczas uczenia zbiór wejściowy, podzielony na bloki, przetwarzany jest przez niezależne podzadania typu *WorkerRole – BackProp Task*.

System zakłada współpracę kilku typów podzadań (rys. 1), tj.:

- *WebRole* – procesu zakładającego interakcję z użytkownikiem, służącego do inicjalizacji zadania uczenia (kontekst: *WebRole-Init* – rys. 4) albo do sprawdzania stanu zadania

i odebrania wyników uczenia (kontekst: *WebRole-Status* – rys. 8). Po inicjalizacji proces *WebRole* zgłasza do wykonania podzadanie *WorkerRole-Start Task*;

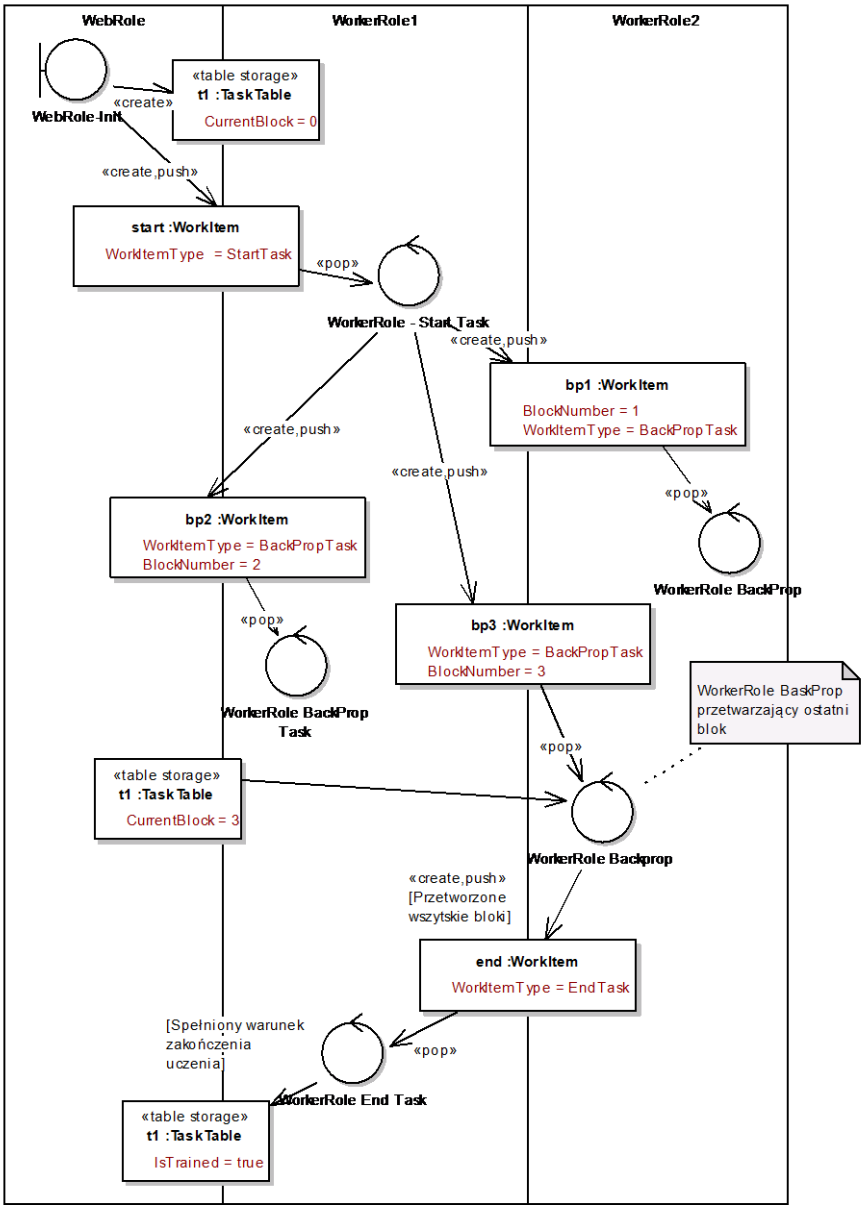
- *WorkerRole-Start Task* (rys. 5) – procesu/podzadania rozpoczynającego realizację zdania, tzn. tworzącego i inicjującego struktury danych oraz zgłaszającego do wykonania zbiór podzadań typu *Worker Role – BackProp Task*;
- *WorkerRole – BackProp Task* (rys. 6) – procesu/podzadania realizującego zmodyfikowany algorytm uczenia sieci neuronowej dla danego bloku (numer bloku jest parametrem podzadania); ostatni proces typu *Worker Role – BackProp Task* (tzn. ten proces przetwarzający, który wykryje, że przetworzył ostatni blok zbioru uczącego) zgłasza do wykonania podzadanie *WorkerRole – End Task*;
- *WorkerRole – End Task* (rys. 7) – procesu/podzadania kończącego iterację (epokę), wyznaczającego wagi na podstawie zmian wag wyznaczonych „w blokach" (przez podzadania *WorkerRole – BackProp Task*); jeśli zostały spełnione warunki zakończenia zadania, to podzadanie *WorkerRole – End Task* tworzy plik wyjściowy (z wynikowymi wagami na wejściach neuronów); w przeciwnym wypadku proces zgłasza do wykonania zbiór podzadań typu *WorkerRole – BackProp Task* i rozpoczyna się następna epoka.



Rys. 1. Diagram współpracujących podzadań
Fig. 1. Cooperating subtasks diagram

Zgłaszanie podzadań do wykonania odbywa się za pośrednictwem kolejki (*WorkerOueue* z rys. 3). Do kolejki wstawiany jest obiekt klasy *WorkItem*, którego pole *WorkItemType* okre-

śla rodzaj podzadania, a jeśli podzadanie jest typu *BackProp Task*, przekazywany jest dodatkowo numer bloku do przetworzenia (*BlockNumber*).



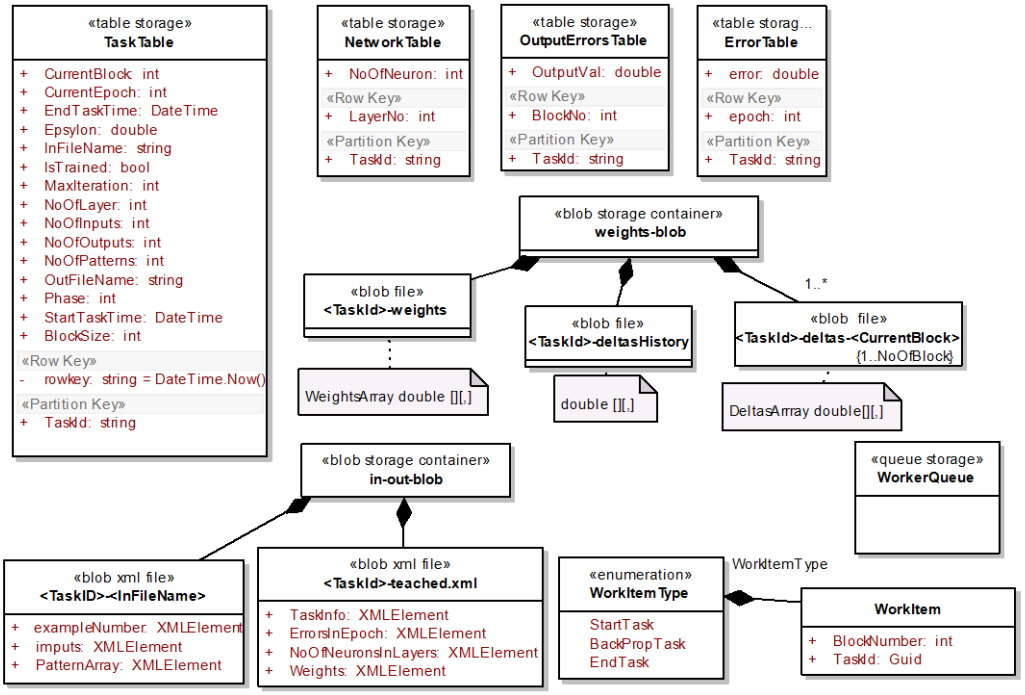
Rys. 2. Synchronizacja procesów przez dane: podzadania *Web Role-Init*, *Start Task*, *BackProp Task*, *End Task* synchronizowane przez obiekty *WorlItem* w kolejce, a podzadania *Back Prop Task* – przez współdzielony licznik *Current Block* w *TaskTable*

Fig. 2. Process synchronization by data: subtasks *Web Role-Init*, *Start Task*, *BackProp Task*, *End Task* synchronization using *WorkItem* objects placed in queue, and subtasks *Back Prop Task* synchronization using the shared counter – *Current Block* in *TaskTable*

W głównej części kodu programu implementującego przetwarzanie dowolnego procesu typu *WorkerRole* (w metodzie *Run*), w ramach nieskończonej pętli następuje pobieranie z kolejki zgłoszenia podzadania do wykonania i w zależności od jego typu zachodzi realizacja podzadania: *Start Task, BackProp Task* lub *End Task*. Liczba procesów (instancji) typu *Worker Role* jest parametrem konfiguracyjnym systemu. Liczba podzadań typu *BackProp Task*

wynika z parametrów zadania uczącego, tzn. wartości ceil [*NumberOfPatterns* / *BlockSize*]. W przykładzie z rys. 2 (gdzie pokazano sytuację, w której całe zadanie uczenia zostało zrealizowane w ramach jednej epoki) liczba procesów *WorkerRole* była równa 2, a liczba podzadań *BackProp Task* była równa 3. Synchronizacja wykonania podzadań odbywa się za pośrednictwem zawartości kolejki, natomiast wykrycie, czy podzadanie *BaskProp* jest ostatnie w ramach danej iteracji, odbywa się przez badanie stanu licznika przetworzonych bloków – *CurrentBlock* (rys. 2).

Przy implementacji systemu wykorzystano różne dostępne rodzaje magazynowania danych (rys. 3), tzn.: *Azure Tables* (do przechowywania danych strukturalnych – np. definiujących parametry zadania, wartości błędów w epokach czy definiujących architekturę sieci neuronowej), *Azure Queues* (do szeregowania wykonania podzadań), *Azure Blob* (do przechowywania zbiorów danych w postaci plików XML (dane wejściowe i wyjściowe) i plików binarnych (dane tymczasowe – wagi, zmiany wag)).



Rys. 3. Model danych – wykorzystanie metod przechowywania danych w technologii *Windows Azure*, tj. tablic, kolejki i pojemników na obiekty-pliki typu BLOB
Fig. 3. Data model – usage of *Windows Azure Table*, *Queue,* and *Blob Storage*

4.1. Opis współdziałających podzadań

Proces *WebRole-Init* (uruchomiona aplikacja ASP.NET), odpowiedzialny za komunikację z użytkownikiem za pośrednictwem strony WWW (rys. 4), pozwala na podanie parametrów zdania uczącego, takich jak:

- warunki zakończenia zadania, tj.: maksymalna liczba iteracji *MaxIteration* (maksymalna liczba epok) oraz dopuszczalny błąd uczenia *Epsylon*,
- architektura sieci neuronowej (liczba warstw ukrytych sieci neuronowej *NoOfLayers* i liczba neuronów w każdej warstw *NoOfNeurons* dla *LayNo* w *NetworkTable*),
- rozmiar bloku *BlockSize* (definiujący podział zbioru uczącego na bloki),
- nazwa pliku ze zbiorem uczącym *InFilename*.

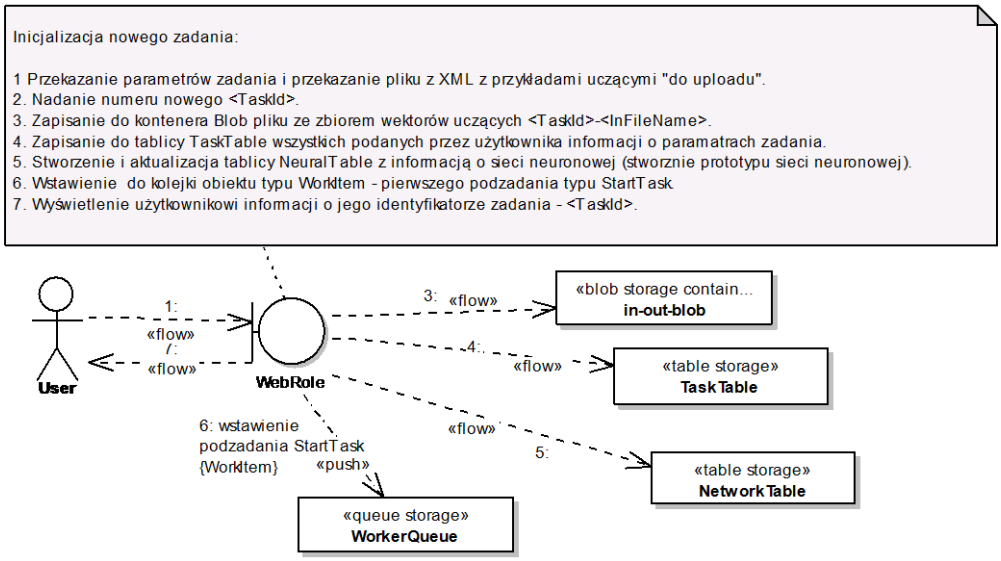
Zbiór uczący jest w postaci pliku XML i powinien zawierać następujące elementy: liczbę przykładów uczących (wartość *NoOfPatterns*), liczbę wejść (wartość *NoOfInputs*), liczbę wyjść sieci (wartość *NoOfOutputs*) oraz oczywiście sekwencję wejściowych wektorów uczących i odpowiadających im spodziewanych wartości wyjść sieci.

Rozpoczęcie realizacji zadania uczącego (naciśnięcie przycisku na formatce) powoduje utworzenie nowego identyfikatora zadania *TaskId* (GUID) i pokazanie go w przeglądarce WWW. Po stronie systemu Azure w *TaskTable* tworzona jest encja z opisem zadania.

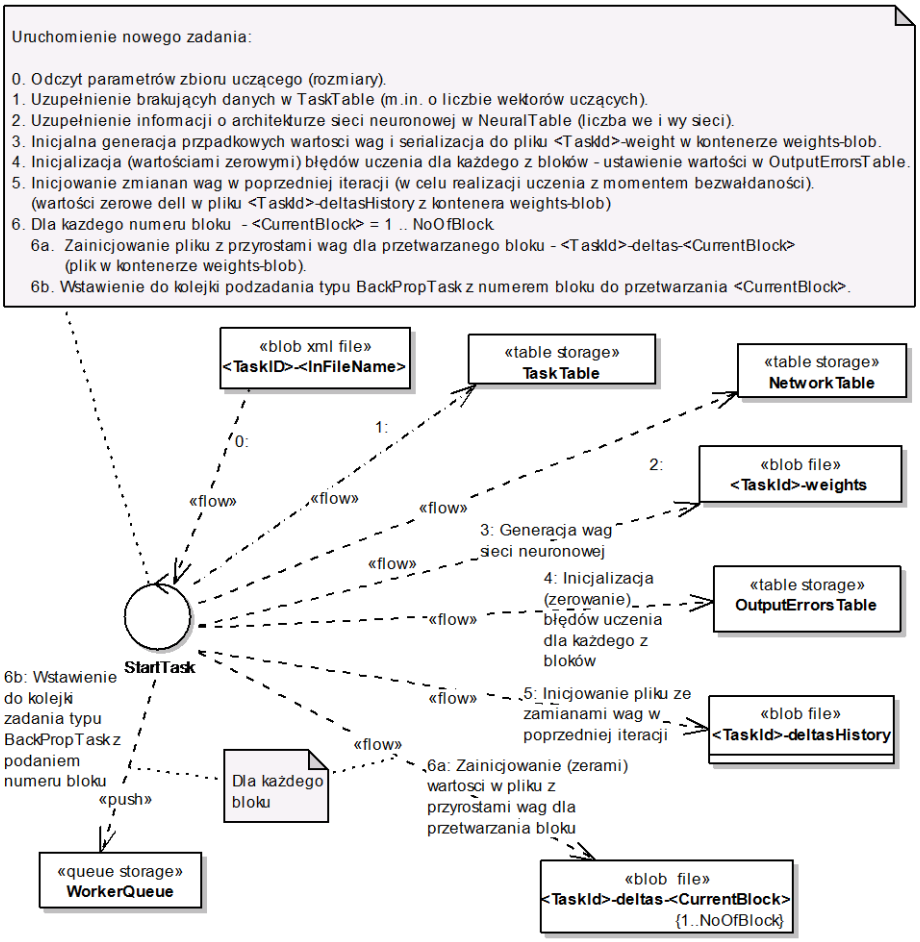
Użytkownik, który później będzie chciał sprawdzić stan zadania i ewentualnie pobrać wyniki (realizacja procesu *WebRole-Status*, rys. 8), będzie się domyślnie posługiwał tym identyfikatorem (chyba że zlecił jeszcze inne zadania uczenia, wtedy będzie musiał jawnie wybrać jeden z wielu identyfikatorów zadań).

W ramach inicjalizacji zadania do chmury przekazywany jest plik wejściowy do kontenera *in-out-storage* (plik o nazwie <*TaskID*>-<*InFileName*>), tworzony jest opis architektury sieci neuronowej (liczba neuronów w każdej z warstw ukrytych) oraz wstawiany jeden obiekt typu *WorkItem*, będący zgłoszeniem do wykonania podzadania *WorkerRole – Start*.

Podzadanie typu *WorkerRole – Start* (rys. 5) jest realizowane przez proces (instancję) typu *WorkerRole*. W ramach tego procesu następuje: uzupełnienie parametrów bieżącego zadania (np. *NoOfOutputs*, *NoOfInputs*, *NoOfPatterns* w *TaskTable*), wprowadzenie informacji o architekturze sieci (liczba wejść i wyjść sieci w *NeuralTable*), wygenerowanie inicjalnych, przypadkowych wartości wag sieci neuronowej i ich serializacja, tzn. zapis do pliku binarnego <*TaskId*>*-weigths*, inicjowanie zerami „poprzednich” zmian wag i serializacja do jednego pliku binarnego <*TaskID*>*-deltasHistory* (na potrzeby liczenia zmian wag metodą *momentum* (wzór (9)) oraz inicjowanie zerami zmian wag dla każdego z bloków, czyli utworzenie tylu plików <*TaskId*>*-deltas-*<*CurrentBlock*>, na ile bloków został podzielony cały zbiór uczący. Po zakończeniu opisanych czynności następuje wstawienie do kolejki odpowiedniej liczby obiektów *WorkItem* typu *BackPropTask* – każdy z podanym numerem bloku do przetworzenia. To działanie inicjuje uruchomienie wielu niezależnych podzadań *WorkerRole – Back-Prop Task*.



Rys. 4. Opis działań podzadania *WebRole-Init* (czynności inicjalizacji zadania)
Fig. 4. Subtask *WebRole-Init* in details (task initialization actions)

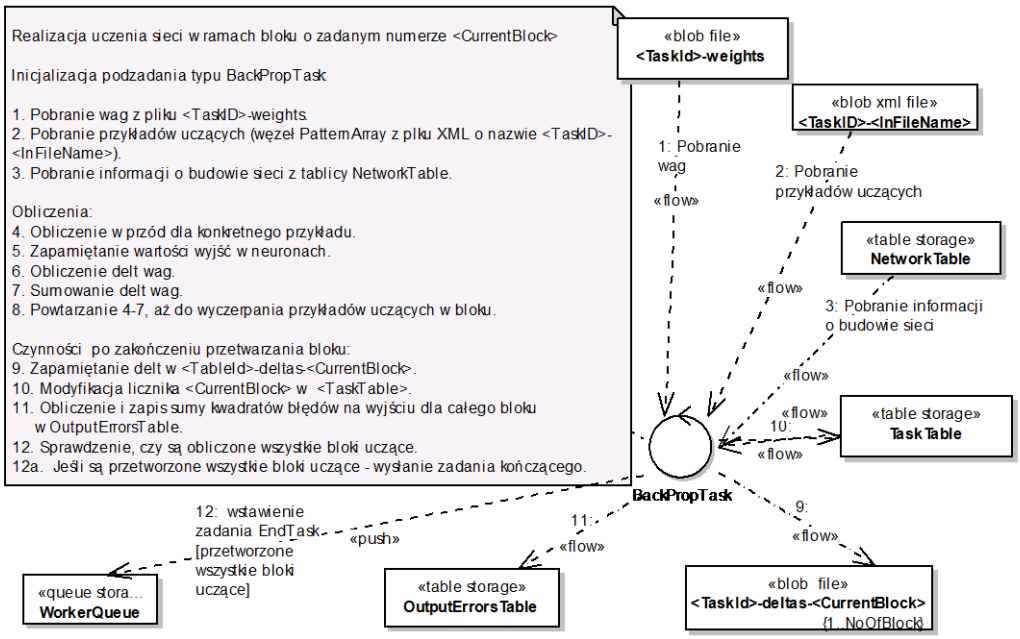


Rys. 5. Opis działań podzadania *WorkerRole-Start Task* (rozpoczęcie zadania)
Fig. 5. Subtask *WorkerRole-Start Task* in details (starting the task)

Podzadanie *WorkerRole – BackProp Task* (rys. 6) realizuje algorytm wstecznej propagacji błędu w procesie uczenia sieci na podstawie wektorów uczących z bloku o zadanym numerze.

Korzystając z części przykładów uczących (z <*TaskID*>-<*InFileName*>) i wag z poprzedniej (iteracji) (z <*TaskId*>-*weights*), następuje wyznaczenie **zmian** wag połączeń między neuronowych (i ich zapis do <*TaskId*>-*deltas*-<*CurrentBlock*>) (wzór (8)). Dodatkowo liczony jest sumaryczny błąd średniokwadratowy, wyznaczany na podstawie bloku, i zapisywany jest do odpowiedniej encji w *OutputErrorTable* (indeksowanej numerem przetwarzanego bloku). Dodatkowo, w *TaskTable*, inkrementowany jest licznik przetworzonych bloków (zerowany na początku każdej iteracji). Każde podzadanie typu *WorkerRole – Back Prop Task* wykonuje przedstawione czynności.

Jeśli podzadanie przetworzyło ostatni blok (stan licznika równy liczbie bloków), oznacza to koniec iteracji (zakończona epoka – przetworzone wszystkie przykłady uczące) i do kolejki jest wstawiany obiekt *WorkItem* typu *EndTask*.



Rys. 6. Opis działań podzadania *WorkerRole-BackProp Task* (podzadanie uczenia sieci metodą *Back Propagate* na podstawie przykładów z konkretnego bloku)

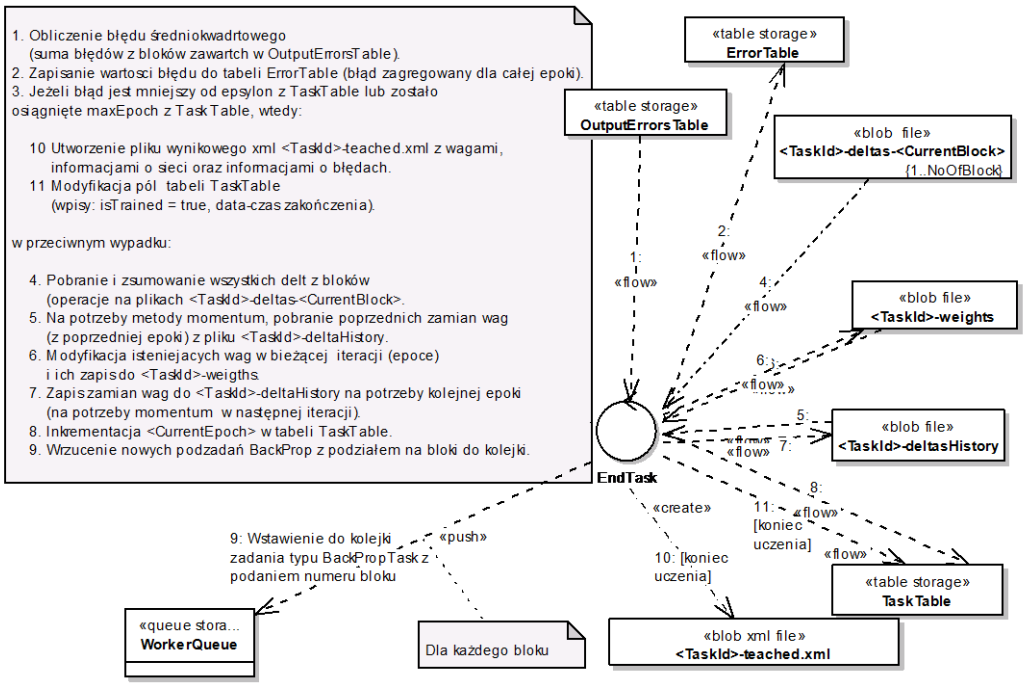
Fig. 6. Subtask *WorkerRole-BackProp Task* in details (subtask of network learning using *Back Propagate* method based on vectors form the concrete block)

W ramach podzadania *WorkerRole – End Task* (rys. 7) następuje wyznaczenie nowych wag dla całej sieci neuronowej – tzn. sumowane są zamiany wag z każdego <*TaskId*>-*deltas*-<*CurrentBlock*>. Następnie zmiany te są modyfikowane wartościami zmian wag z poprzedniej iteracji (z <*TaskId*>-*deltasHistory*) (wzór (9)) i zapisywane do <*TaskId*>-*weights*. Przy okazji, na potrzeby ewentualnej kolejnej iteracji, otrzymane zmiany wag nadpisują te zawarte w pliku <*TaskId*>-*deltasHistory*.

W podzadaniu *WorkerRole – End Task* następuje również agregacja błędów wyznaczonych w ramach każdego z bloków (z tablicy *OutputErrorTable*) i zapis zsumowanej wartości błędu do encji *ErrorTable* (indeksowanej numerem epoki, czyli numerem bieżącej iteracji).

Jeśli nie zostały spełnione kryteria zakończenia uczenia, to inkrementowany jest numer iteracji (<*CurrentEpoch*> w *TaskTable*) i wygenerowana jest odpowiednia liczba obiektów *WorkItem* typu *BackPropTask* (każdy z podanym unikalnym numerem bloku do przetworzenia), wstawianych do kolejki zgłoszeń. Tym samym rozpoczyna się kolejna iteracja (epoka ucząca).

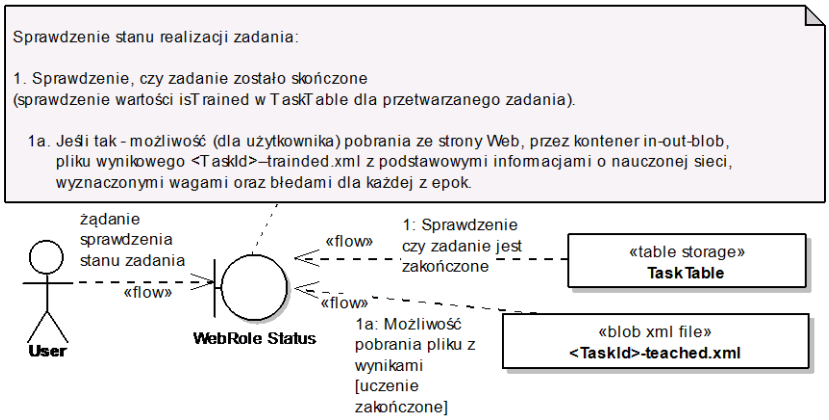
Jeśli zostały spełnione kryteria zakończenia uczenia (przekroczony w *TaskTable* licznik epok, tzn. *CurrentEpoch* >= *MaxIteration*, lub błąd uczenia jest odpowiednio mały, tzn. mniejszy niż *Epsylon*), to w kontenerze *in-out-blob* tworzony jest plik wynikowy <*TaskId*>*-teached.xml* z informacjami o: parametrach zadania, architekturze sieci neuronowej, błędach dla każdej z kolejnych epok uczenia i oczywiście wynikowych wartościach wag. Po wygenerowaniu pliku wynikowego ustawiany jest stan zadania jako: wykonane (*isTrained* = *true* w odpowiedniej encji w *TaskTable*).



Rys. 7. Opis działań podzadania *WorkerRole-End Task* (podzadanie zakończenia iteracji (epoki) i kontroli warunków zakończenia całego zadania uczącego)

Fig. 7. Subtask *WorkerRole-End Task* in details (subtask of iteration (epoch) ending and checking finish conditions of the whole learning task)

Proces *WebRole* (kontekst *WebRole-Status*, rys. 8) odpowiedzialny jest za odbiór wyników uczenia przez użytkownika. Za pomocą aplikacji WWW użytkownik może odpytać system Azure, czy zadanie o podanym <*TaskId*> jest zakończone (sprawdzenie *isTrained* w *TaskTable*). Jeśli zadanie jest zakończone, można pobrać plik wynikowy (<*TaskId*>*-teached.xml*).



Rys. 8. Opis działań podzadania *WebRole-Status* (czynności związane ze sprawdzeniem stanu zadania i pobraniem pliku wynikowego)

Fig. 8. Subtask *WebRole-Status* in details (activities: checking status of task execution and downloading the result file)

4.1.1. Zrównoleglenie czynności procesu BackProp Task przez wykorzystanie modułu Parallel Extension for .NET

W metodzie *teach*, implementującej funkcjonalność uczenia w ramach *BackProp Task*, wykorzystano rozszerzenie *Parallel Extension for .NET* [8] dla zwiększenia granulacji podziału podzadania na potrzeby dodatkowego zrównoleglenia obliczeń.

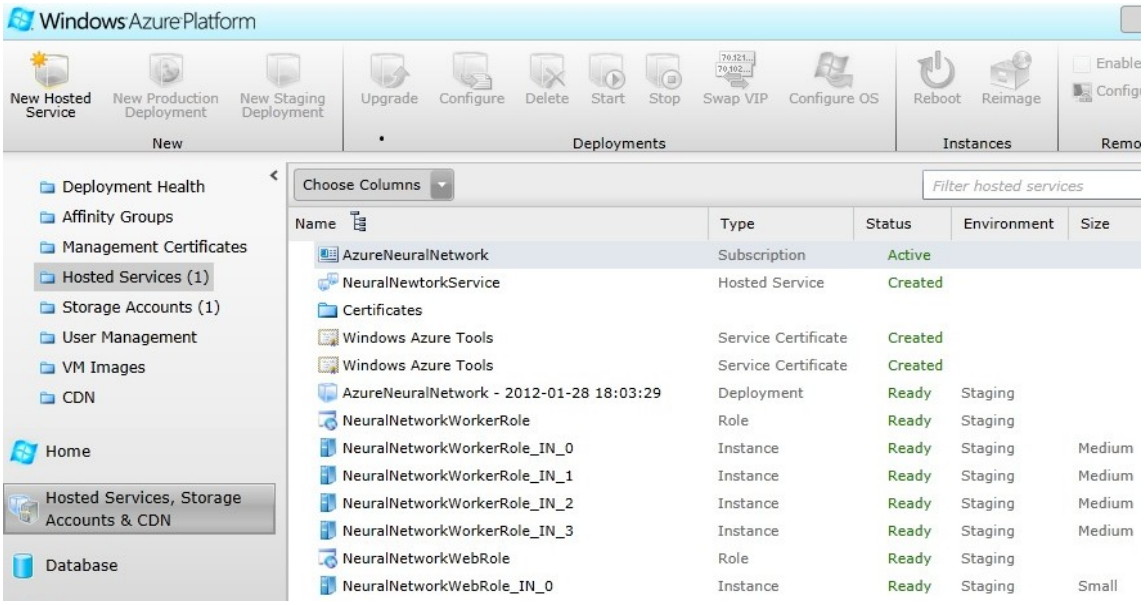
W metodzie *teach* uruchamianej dla bloku danych, dla każdego przykładu uczącego, tzn. wektora z bloku, następuje wyznaczenie zmiany wag w każdym neuronie i zsumowanie zmiany wag. Obliczenia dla pojedynczego przykładu przeprowadzane są niezależenie, tzn. zmiany wag, wyznaczane na podstawie przykładów uczących, są obliczane w pojedynczej iteracji zrównoleglonej pętli *Parallel.For* [9] (synchronizacja czynności w ramach pętli następuje tylko przy zsumowaniu zmian wag – wykorzystanie komendy *lock*). Takie podejście pozwala na zrównoleglenie działań nawet w ramach pojedynczego procesu *WorkerRole – Back Prop Task*, jeśli tylko proces ten będzie uruchamiany na maszynie z procesorem wielordzeniowym. Takie uruchomienie faktycznie miało miejsce, ponieważ wszystkie procesy typu *WorkerRole* były uruchamianie w ramach instancji typu *Medium* (patrz rys. 8). Taki typ instancji zakłada możliwość efektywnego wykorzystania mocy obliczeniowej maszyny z procesorami wielordzeniowymi dzięki użyciu *Parallel Extension for .NET*.

5. Opis przykładowego użycia systemu

5.1. Administracja systemem

Administrowanie systemem uczenia sieci neuronowej odbywa się za pośrednictwem standardowego portalu zarządzania chmurą *Microsoft Windows Azure*. Za jego pomocą system

można „wgrywać do chmury", uruchamiać, monitorować, rekonfigurować, zatrzymywać itp. Na rys. 9 przedstawiono widok panelu administracyjnego, pokazujący stan działania systemu: działające instancje – 1 × *WebRole* i 4 × *WorkeRole*; instancje *WorkeRole* są typu *Medium*.



Rys. 9. Widok standardowego panelu administracyjnego w przeglądarce internetowej; przykład uruchomienia systemu w chmurze obliczeniowej; konfiguracja: 1 proces WebRole i 4 procesy WorkerRole

Fig. 9. A view of standard administrative panel in Web browser; running system in cloud computing environment; the configuration: 1 WebRole instance and 4 WorkerRole instances

5.2. Wyniki prostych eksperymentów wydajnościowych

Testowanie odbywało się na emulatorze chmury, w środowisku lokalnym, na komputerze o następujących parametrach:

- procesor: AMD Athlon(tm) X2 Dual-Core QL-62 2,00 GHz,
- RAM: 2 GB.

Testy zrealizowano również w „prawdziwej" chmurze obliczeniowej, na instancjach maszyn wirtualnych o następujących parametrach:

- procesor: 2 x 1,6 GHz,
- RAM: 3,5 GB,
- lokalna pojemność dyskowa: 490 GB,
- prędkość sieci: ~200 Mbps.

Nie jest możliwe stwierdzenie, jakie rzeczywiście były użyte: komputer, procesor itd., ponieważ jest to ukryte „pod warstwą wirtualizacji".

Testy uczenia sieci neuronowej zrealizowano, opierając się na tzw. ogólnodostępnych danych benchmarkowych – zbiór „Thyroid" [7]. Zbiór ten został opracowany na podstawie da-

nych pacjentów oraz ich wyników badań. Zbiór dotyczy diagnoz chorób tarczycy. Dane te pozwalają określić, czy dany pacjent ma nadczynność, niedoczynność lub czy jego tarczyca pracuje normalnie. Są one określone przez 25 cech pacjenta, 3 klasy decyzyjne oraz 3600 przykładów uczących. Do tych danych testowych dobrano następującą architekturę sieci: 21 neuronów w warstwie wejściowej (tylko 21, gdyż pominięto te 4 cechy, w których występowały braki danych), 10 neuronów w warstwie ukrytej oraz 3 neurony wyjściowe. Przyjęto następujące wartości: dla współczynnika uczenia $\eta = 0{,}0001$ (wzór (6)) i dla współczynnika *momentum* $\mu = 0{,}9$ (wzór (9)). Dopuszczalny błąd uczenia sieci – *Epsylon* – przyjęto na poziomie 0,01, a maksymalna liczba epok – *MaxIteration* – była równa 100.

Rozmiar bloku uczącego *BlockSize* to 1800 przykładów. Został on dostosowany do konfiguracji testowej z 2 instancjami *WorkerRole*.

Tabela 1

Uśredniony czas uczenia sieci

Środowisko pracy	**Czas uczenia** [hh:mm:ss]
Emulator	00:09:04
Windows Azure	00:03:47

Jak pokazują wyniki w tabeli 1, użycie rozwiązania w realnej chmurze obliczeniowej, nawet w najprostszej konfiguracji (tylko 2 instancje *WorkeRole*), dało około 3-krotne przyspieszenie w stosunku do użycia systemu w środowisku emulatora.

6. Podsumowanie

Niniejszy artykuł opisuje system uczenia sieci neuronowej, wykonany na podstawie technologii Microsoft Windows Azure. Dzięki użyciu specyficznych metod przechowywania danych oraz wykorzystując model tworzenia systemu opartego na wielu współpracujących procesach/instancjach, zbudowano system skalowany pod względem wydajności.

System spełnił postawione mu wymagania (nawet przy użyciu tylko 2 instancji *WorkerRole* wydajność systemu działającego w chmurze była lepsza od jego odpowiednika uruchamianego w środowisku lokalnym). Pewnym rozczarowaniem dla autorów były wyniki efektywnościowe uzyskane w pierwszej wersji systemu, w której wykorzystano jedynie *Azure Table* (bez *Azue Blob*). Okazało się, że dodatkowe wykorzystanie *Azure Blob* w kolejnych wersjach sytemu (czyli „mieszane" podejście z *Azure Table* i *Blob*) było najefektywniejsze.

Oczywiście system może być rozwijany w warstwie merytorycznej (np. inne metody uczenia sieci). W warstwie technicznej jednym z kierunków rozwoju będzie wykorzystanie usług buforujących (ang. *cache*) – np. buforowanie zbioru przykładów uczących czy wartości

wag z poprzedniej epoki. To powinno dać pozytywne rezultaty zarówno w sensie efektywności, jak i kosztów eksploatacji (korzystanie z *cache* to inny, bardziej ekonomiczny model kosztowy).

Szczegółowy opis budowy tego systemu może posłużyć jako wskazówka dla osób tworzących podobne rozwiązania (niekoniecznie dotyczące dziedziny przedmiotowej, związanej z sieciami neuronowymi). Niektóre elementy architektury oraz proste komponenty systemu (np. moduł szeregowania zadań, struktury danych opisujące zadanie) można uczynić „reużywalnymi". Taki kierunek rozwoju prac może zaowocować powstaniem modułu szablonowego (ang. *framework*), wspierającego tworzenie systemów zrównoleglonych dla *Windows Azure*. Przy użyciu takiego szablonu projektant/programista będzie odpowiedzialny jedynie za dostarczenie implementacji algorytmu przetwarzającego (np. w postaci kodu źródłowego albo nawet gotowych podzespołów *DLL*) oraz utworzenie merytorycznych struktur danych. Może to ułatwić proces tworzenia rozproszonych systemów przetwarzających, opartych na *Windows Azure*.

BIBLIOGRAFIA

1. Windows Azure (2012), http://msdn.microsoft.com/en-us/library/dd179367.aspx.
2. Tour – Overview – Windows Azure (2012), http://www.windowsazure.com/en-us/home/tour/overview.
3. Cloud Storage – Windows Azure (2012), http://www.windowsazure.com/en-us/develop/net/fundamentals/cloud-storage.
4. Haridas J., Nilakantan N., Calder B.: Windows Azure Table, http://www.scribd.com/doc/63485303/Windows-Azure-Table-May-2009.
5. Tadeusiewicz R.: Sieci neuronowe. Akademicka Oficyna Wydawnicza, Warszawa 1993.
6. Wilson R., Martinez T.: The General Inefficiency of Batch Training for Gradient Descent Learning. Neural Networks, 2003.
7. Prechelt L.: A Set of Neural Network Benchmark Problems and Benchmarking Rules. Technical Report, Karlsruhe 1994.
8. Parallel Programming in the .NET Framework (2012), http://msdn.microsoft.com/en-us/library/dd460693.aspx.
9. Data Parallelism (Task Parallel Library) (2012), http://msdn.microsoft.com/en-us/library/dd537608.aspx.

Wpłynęło do Redakcji 31 stycznia 2012 r.

Abstract

The paper presents the system for artificial neural network learning based on the idea of cloud computing. System was implemented for Microsoft Windows Azure, so it is highly salable and available.

The well-known learning algorithm i.e. back propagation method with momentum was adopted for parallel and distributed execution.

The architecture of cooperative Worker Role instances was proposed for parallel execution of network learning. The paper describes applying of methods of data storage like Windows Azure Table, Queue, Blob. Thank to simple NoSQL-like storage services i.e. Azure Table and Blobs the data manipulation operations are efficient.

The advantages of parallelization result from either applying multiple processes (instances) of Worker Roles or applying Parallel Extension for .NET module in Worker Role's implementation.

Some elements of system architecture and component (ie. module for tasks management using queue) are reusable and may be useful in development of other Azure-based distributed system.

Adresy

Dariusz Rafał AUGUSTYN: Politechnika Śląska, Instytut Informatyki, ul. Akademicka 16, 44-100 Gliwice, Polska, draugustyn@polsl.pl.
Kamil BADURA: Politechnika Śląska, Instytut Informatyki, ul. Akademicka 16, 44-100 Gliwice, Polska, kamil.badura@hotmail.com.