FAKULTA
APLIKOVANÝCH VĚD
ZÁPADOČESKÉ
UNIVERZITY
V PLZNI

Semestrální práce z KIV/PPR

Predikce glykémie na zvolený počet minut dopředu

Zdeněk Častorál
A19N0026P
zcastora@students.zcu.cz

24. 11. 2020

Obsah

1 Zadání

V semestrální práci budete řešit zjednodušenou verzi problému predikce glykémie na zvolený počet minut dopředu (Blood Glucose Level Prediction Challenge). Budete trénovat feedforward neuronovou síť, v propojení každý s každým, která bude provádět multiclass klasifikaci. Síť bude mít následující strukturu:

1. Vstupní vrstva - 8 prvků, kterým bude dávat hodnoty IG v časech t-0, $t-5$, $t-10$, $t-15$, $t-20$, $t-25$, $t-30$, $t-35$ minut. IG je koncentrace glukózy v intersticiální tekutině. Buď použijte výpočet běžícího průměru, a průběžnou aktualizaci minimálních a maximálních hodnot IG, abyste vstup "vycentrovali" v mezích -1 a 1, anebo použijte risk funkci. Zde nebudou žádné váhy ani bias.

2. První skrytá vrstva - 16 prvků. Každý prvek bude mít svůj bias a váhy. Doporučená fce TanH.

3. Druhá skrytá vrstva - 26 prvků. Každý prvek bude mít svůj bias a váhy. Doporučená fce TanH.

4. Výstupní vrstva - 32 prvků. První prvek, index 0, bude reprezentovat koncentraci menší nebo rovnu 3 mmol/L. Poslední prvek bude reprezentovat koncentrace větší nebo rovnu 13 mmol/L. Ostatní koncentrace budou odpovídat poskytnutému kódu. Každý prvek bude mít svůj bias a váhy. Zde použijte funkci SoftMax.

Pokud byste chtěli použít sigmoid a multilabel klasifikaci, pak použijte vrstvy $8-16-16-5$. Výstup pak interpretujte jako binární číslo, přičemž první výstup je LSB. Tento přístup by byl sice efektivnější, ale má menší šanci, že bude fungovat.

Vstupní hodnoty najdete v SQLite databázi, v sekci ke stažení. Hodnoty IG jsou v tabulce measuredvalue, sloupeček ist. Sloupeček measuredat udává čas měření. Použijte pouze ty sekvence, pro které budete mít nepřerušované IG hodnoty s rozestupem 5 minut. IG hodnoty jsou totiž dělené do segmentů, které na sebe bezprostředně nenavazují.

Trénování neuronové sítě optimalizujte za pomoci relativní chyby - abs(vypočítaná hodnota - naměřená hodnota)/naměřená hodnota. Pro jednu sadu parametrů neuronové sítě tedy získáte mnoho relativních chyb. Z nich vypočtěte průměrnou chybu, kterou sečtete se standardní odchylkou těchto chyb. Výsledné číslo udává, jak dobře je síť natrénovaná.

Na začátku vygenerujte náhodné hodnoty, které pak optimalizujte pomocí backpropagation (můžete zkombinovat s SGD). Protože cílem je paralelní výpočet, trénujte několik instancí sítě najednou a nejúspěšnější parametry průměrovat či jinak kombinovat. Dle uvážení můžete zkusit i jinou metodu, nicméně ji předem konzultujte.

Vytvořený program bude mít tři argumenty - první z nich bude počet minut, na který se bude provádět predikce (např. 30, nebo 120). Druhý parametr bude jméno sqlite databáze. Třetí argument je volitelný, a bude to soubor s uloženými parametry sítě. V takovém případě se síť nebude trénovat. Ještě možné doplnit čtvrtý a pátý parametr, který zvolí multilabel vs multiclass, a běžící průměr vs risk.

Výstupem csv soubor, který bude obsahovat průměrnou relativní chybu, standardní odchylku relativních chyb a po jednom procentu navzorkovanou empirickou kumulativní distribuční funkce relativní chyby (setřídíte všechny relativní chyby od nejmenší po největší a pak je vypíšete od první do 100% všech chyb s krokem 1%). Výstupem bude také soubor nerual.ini, kde budou uloženy parametry sítě v zadaném formátu.

Program také vypíše, zda použil běžící průměr nebo funkci risk, multilabel nebo multiclass klasifikaci a jaký multiprocesor používá.

Dále pro každý prvek výstupní vrstvy do vlastního souboru nakreslíte graf reprezentující neuronovou síť. Tu zakreslíte světle šedou barvou. Zelenou barvou (s intenzitou $0-255$) zakreslíte propojení jednotlivých neuronů. Intenzitu každého propojení určíte tak, že každé propojení budete asociovat s čítačem. Ten bude na začátku nula. S každou predikcí mu přičtete číslo, které předal připojenému neuronu. Nakonec ze všech propojení určíte maximální hodnotu a naškálujete čítače všech propojení tak, aby byly v intervalu $0 - 255$. Tj. minimální hodnotu neurčujete - ta je nula. Tím získáte intenzitu zelené barvy. Ke každému propojení (tj. různě zelené čáry) připíšete i intenzitu jako číslo. Analogicky nakreslíte ještě jeden graf. Ten bude ale používat modrou barvu a počítadla se budou zvětšovat jenom tehdy, bude-li mít aktuální predikce relativní chybu menší nebo rovnu 15%. Jedná se o krok směrem k XAI (vysvětlitelná umělá inteligence), která vám umožní vysvětlit, co jste vlastně vaší neuronovou síť naučili.

Semestrální práce využije alespoň dvě z celkem tří možných technologií:

1. paralelní program pro systém se sdílenou pamětí - `C++` vč. `PSTL C++17`, popř. `WinAPI`,
2. program využívající asymetrický multiprocesor - konkrétně `x86 CPU` a `OpenCL` kompatibilní `GPGPU`. Po domluvě lze použít `SYCL`,
3. paralelní program pro systém s distribuovanou pamětí - `C++ MPI`.

2 Analýza

V této kapitole je popsána analýza neuronové sítě a zpracování trénovacích dat. Dále jsou uvedeny technologie použité k realizaci semestrální práce.

2.1 Neuronová síť

V rámci této semestrální práce byla realizována *feedforward* neuronová síť, kde každý prvek vrstvy je propojen s každým prvkem následující vrstvy. Síť provádí *multiclass* klasifikaci do 32 tříd. Síť má následující strukturu:

- *vstupní vrstva* – 8 neuronů (tzn. 8 hodnot trénovací množiny),
- *první skrytá vrstva* – 16 neuronů, každý prvek má svoje váhy a bias, použítá funkce *tanh*,
- *druhá skrytá vrstva* – 26 neuronů, každý prvek má svoje váhy a bias, použítá funkce *tanh*,
- *výstupní vrstva* – 32 neuronů, každý prvek má svoje váhy a bias, použítá funkce *softmax*.

Každý prvek výstupní vrstvy reprezentuje koncentraci glykemie v $mmol/L$. První prvek (index 0), reprezentuje koncentrace menší nebo rovny 3 $mmol/L$. Poslední prvek reprezentuje koncentrace větší nebo rovny 13 $mmol/L$. Výpočet ostatních koncentrací zajišťuje funkce `Band_Index_To_Level`, která převede index nejvíce aktivovaného neuronu výstupní vrstvy na hodnotu glykemie v $mmol/L$ (viz ukázka kódu 2.1).

Ukázka kódu 2.1: Výpočet hodnot glykemie

```
1  //mmol/L below which a medical attention is needed
2  static constexpr double Low_Threshold = 3.0;
3
4  //dtto above
5  static constexpr double High_Threshold = 13.0;
6
7  //number of bounds inside the thresholds
8  static constexpr size_t Internal_Bound_Count = 30;
9
10 //must imply relative error <= 10%
11 static constexpr double Band_Size = (High_Threshold -
       Low_Threshold) / static_cast<double>(Internal_Bound_Count);
12 static constexpr double Inv_Band_Size = 1.0 / Band_Size;
13 static constexpr double Half_Band_Size = 0.5 / Inv_Band_Size;
14 static constexpr size_t Band_Count = Internal_Bound_Count + 2;
15
16 double Band_Index_To_Level(const size_t index) {
17   if (index == 0) return Low_Threshold - Half_Band_Size;
18   if (index >= Band_Count - 1) return High_Threshold +
         Half_Band_Size;
19
20   return Low_Threshold + static_cast<double>(index - 1)*
         Band_Size + Half_Band_Size;
21 }
```

2.1.1 Softmax funkce

Aktivační funkcí poslední vrstvy je funkce *softmax*. Jejím použitím je získána pravděpodobnost každého neuronu výstupní vrstvy. Neuron, který má nejvyšší pravděpodobnost (je nejvíce aktivovaný), je výsledkem predikce. Funkce *softmax* je definována následovně:

$$f(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}},$$

kde:

- $\vec{z}$ je vektor vstupních hodnot $(z_1, z_2, ..., z_K)$,
- z_i je vstupní hodnota pro i-tý neuron výstupní vrstvy,
- K je počet tříd klasifikace (tzn. počet neuronů výstupní vrstvy).

2.1.2 Výpočet chyby sítě

Po získání indexu nejvíce aktivovaného neuronu výstupní vrstvy a převedení tohoto indexu na hodnotu glykemie v *mmol/L* následuje výpočet *relativní chyby*. Ta se vypočítá podle vzorce:

$$x_i = \frac{|r_i - e_i|}{e_i},$$

kde:

- i je index trénovacího vzorku, $i \in \{1, 2, ..., n\}$,
- n je počet trénovacích vzorků,
- r_i je predikovaná (vypočítaná) hodnota pro i-tý vzorek,
- e_i je očekávaná (naměřená) hodnota pro i-tý vzorek.

Relativní chyba se získává pro každý průchod neuronovou sítí. Počet chyb tedy odpovídá počtu trénovacích vzorků. Z takto získaných chyb se dále vypočítá *průměrná relativní chyba* a *standardní odchylka* těchto chyb. Tyto dvě hodnoty se sečtou, výsledné číslo udává, jak dobře je síť natrénována.

2.1.3 Trénování neuronové sítě

Inicializační hodnoty vah synapsí jsou generovány náhodně prostřednictvím pseudonáhodného generátoru čísel dle *uniformního rozdělení*. Hodnoty vah jsou po každém *feedforward* průchodu neuronovou sítí optimalizovány pomocí *backpropagation*. Tímto postupem by se měly zmenšovat relativní chyby predikce neuronové sítě.

2.2 Trénovací data

Trénovací dana jsou získávána z přiložené *SQLite* databáze. V této databázi se nacházejí hodnoty glykémie v *mmol/L* (tabulka *measuredvalue*, sloupec *ist*). Naměřené hodnoty glykémie jsou děleny do segmentů, které na sebe bezprostředně nenavazují. Hodnoty v rámci jednoho segmentu jsou měřeny po 5 minutách.

2.2.1 Trénovací vzorek

Vstupem neuronové sítě je 8 naměřených hodnot glykémie, které mají mezi sebou rozestupy 5 minut (tj. hodnoty v časech t, $t-5$, $t-10$, $t-15$, $t-20$, $t-25$, $t-30$, $t-35$). Pro těchto 8 hodnot musí existovat očekávaná (naměřená) hodnota v čase $t+x$, tj. od osmé hodnoty ve vstupu je přičten zvolený počet minut dopředu, na který se dělá predikce. Očekávaná (naměřená) hodnota se používá k výpočtu relativní chyby (viz výše). Pokud existuje očekávaná hodnota pro danou osmici vstupních hodnot, tvoří společně s těmito osmi hodnotami jeden *trénovací vzorek*. Pokud k osmici vstupních hodnot očekávaná hodnota neexistuje, nelze tyto hodnoty použít (tzn. tyto hodnoty nejsou trénovacím vzorkem).

Získávání trénovacích vzorků z databáze funguje na principu posuvného okénka o osmi hodnotácch, kdy v každém následujícím běhu získávání dat je okénko posunuto o 1. Příklad získávání trénovacích vzorků s predickí na 60 minut dopředu je zobrazeno na obrázku 2.1, červenou barvou je vyznačeno okénko osmi vstupních hodnot, zelenou barvu je vyznačena očekávaná hodnota.

čas	ist
t - 35	11.05
t - 30	10.88
t - 25	10.88
t - 20	10.88
t - 15	10.77
t - 10	10.66
t - 5	10.49
t	10.32
t + 5	10.21
t + 10	10.1
t + 15	10.21
t + 20	10.27
t + 25	10.38
t + 30	10.38
t + 35	10.32
t + 40	10.38
t + 45	10.32
t + 50	10.21
t + 55	10.16
t + 60	10.27
t + 65	10.16
t + 70	9.99
t + 75	9.77
t + 80	9.71

čas	ist
t - 40	11.05
t - 35	10.88
t - 30	10.88
t - 25	10.88
t - 20	10.77
t - 15	10.66
t - 10	10.49
t - 5	10.32
t	10.21
t + 5	10.1
t + 10	10.21
t + 15	10.27
t + 20	10.38
t + 25	10.38
t + 30	10.32
t + 35	10.38
t + 40	10.32
t + 45	10.21
t + 50	10.16
t + 55	10.27
t + 60	10.16
t + 65	9.99
t + 70	9.77
t + 75	9.71

čas	ist
t - 45	11.05
t - 40	10.88
t - 35	10.88
t - 30	10.88
t - 25	10.77
t - 20	10.66
t - 15	10.49
t - 10	10.32
t - 5	10.21
t	10.1
t + 5	10.21
t + 10	10.27
t + 15	10.38
t + 20	10.38
t + 25	10.32
t + 30	10.38
t + 35	10.32
t + 40	10.21
t + 45	10.16
t + 50	10.27
t + 55	10.16
t + 60	9.99
t + 65	9.77
t + 70	9.71

Obrázek 2.1: Získávání trénovacích vzorků (pro tři běhy)

Po načtení všech trénovacích vzorků jsou vstupní hodnoty naškálovány tak, aby byly z intervalu $\langle -1, 1 \rangle$. Ke škálování hodnot je použita dodaná *risk funkce* (viz ukázka kódu 2.2).

Ukázka kódu 2.2: Risk funkce

```
1 double risk(const double bg) {
2
3   // DOI:  10.1080/10273660008833060
4    const double original_risk = 1.794 * (pow(log(bg), 1.026) -
        1.861);    //mmol/L
5
6    return original_risk / 3.5;
7 }
```

2.3 Zvolené technologie

Semestrální práce je rozdělena na dvě části dle zvolených technologií:

1. paralelní program pro symetrický multiprocesor – použit standard `C++17` a knihovna `Intel® TBB` (dále jen *program pro SMP*),
2. program využívající asymetrický multiprocesor – použit `x86 CPU` a `OpenCL` kompatibilní `GPGPU` (dále jen *program pro GPU*).

3 Implementace

Semestrální práce byla implementována v jazyce `C++` (standard `C++17`). K načítání dat z databáze byla použita knihovna `sqlite3.lib`, k paralelizaci na SMP byla použita knihovna `Intel® TBB 3.0`. K implementaci programu pro GPU bylo použito `OpenCL 2.0`.

3.1 Struktura zdrojových souborů

Zdrojové soubory jsou uloženy v adresáři `/src` a jsou strukturovány následovně:

- adresář `dao` – obsahuje zdrojové soubory pro vstupní a výstupní operace (např. načtení dat z databáze),
- adresář `gpu` – obsahuje zdrojové soubory programu pro GPU,
- adresář `smp` – obsahuje zdrojové soubory programu pro SMP,
- adresář `util` – obsahuje zdrojové soubory s pomocnými funkcemi programu,
- soubor `main.cpp` – hlavní zdrojový soubor programu, obsahuje spouštěcí bod.

3.2 Sériová verze

Neuronová síť je implementována v souladu s analýzou uvedenou v kapitole 2.1. Základní struktury neuronové sítě jsou:

- `TNetwork` – reprezentuje neuronovou síť, obsahuje vektor vrstev a vektor relativních chyb (viz ukázka kódu 3.1),
- `TLayer` – reprezentuje vrstvu sítě, obsahuje vektor neuronů (viz ukázka kódu 3.2),
- `TNeuron` – reprezentuje jeden neuron, který obsahuje výstupní hodnotu, výstupní synapse do dalších neuronů, index neuronu a gradient (viz ukázka kódu 3.3),
- `TSynapse` – spojení mezi dvěma neurony, obsahuje váhu, rozdíl vah a čítače pro vykreslování grafů (viz ukázka kódu 3.4).

Ukázka kódu 3.1: Struktra `TNetwork`

```
1 struct TNetwork {
2   std::vector<kiv_ppr_neuron::TLayer> layers;
3   std::vector<double> relative_errors_vector;
4 };
```

Ukázka kódu 3.2: Struktra `TLayer`

```
1 struct TLayer {
2   std::vector<kiv_ppr_neuron::TNeuron> neurons;
3 };
```

Ukázka kódu 3.3: Struktra `TNeuron`

```
struct TNeuron {
  double output_value = 0.0;
  std::vector<kiv_ppr_synapse::TSynapse> output_weights;
  unsigned neuron_index = 0;
  double gradient = 0.0;
};
```

Ukázka kódu 3.4: Struktra `TSynapse`

```
struct TSynapse {
  double weight = 0.0;
  double delta_weight = 0.0;
  double current_counter = 0.0;
  double counter_green_graph = 0.0;
  double counter_blue_graph = 0.0;
};
```

Bias je implementován jako neuron, který má výstupní hodnotu 1 a do kterého nevedou žádné synapse, pouze z něj.

Neuronová síť je trénována na celé trénovací množině (všechny validní trénovací vzorky). Výstupy trénování neuronové sítě jsou soubory:

- `neural.ini` – obsahuje uložené parametry sítě (váhy synapsí a jednotlivé biasy),
- `errors.csv` – obsahuje průměrnou relativní chybu, standardní odchylku relativních chyb a po jednom procentu navzorkovanou empirickou kumulativní distribuční funkci relativní chyby,
- `green_graph.svg` – graf neuronové sítě, který vykresluje synapse neuronů různou intenzitou zelené barvy podle toho, jaká je akumulovaná hodnota signálů synapse (viz obrázek 3.1),
- `blue_graph.svg` – graf neuronové sítě, vykresluje synapse neuronů různou intenzitou modré barvy podle akumulované hodnoty signálů synapse, ale pouze pro případy, kdy je chyba sítě menší nebo rovna 15%.

Je možné spouštět program se vstupním souborem obsahujícím parametry sítě (`neural.ini`). V takovém případě není síť trénována, ale jsou načteny tyto parametry a je spuštěna predikce (tzn. není použit *backpropagation*). Jsou vygenerovány výstupní soubory `errors.csv`, `green_graph.svg`, `blue_graph.svg` a `results.txt`, který obsahuje informace o výsledních predikce.

3.3 Paralelizace na symetrickém multiprocesoru

Paralelní program pro symetrický multiprocesor vychází ze sériové verze programu. Paralelizováno je trénování více instancí neuronové sítě a pro každou instanci je paralelizována aktualizace vah čítačů pro grafy.

K paralelizaci je použita funkce `tbb::parallel_for`. Trénování více instancí sítě najednou je zobrazeno v ukázce kódu 3.5.

Ukázka kódu 3.5: Paralelní trénování více instancí neuronové sítě

```
1  tbb::parallel_for(size_t(0), neural_networks.size(), [&](size_t
       j) {
2
3      // --- Spusteni feed forward propagation ---
4      kiv_ppr_network::Feed_Forward_Prop(neural_networks[j],
           input_values_risk, i, topology[0]);
5
6      // --- Vypocitani relativni chyby a pridani do vektoru chyb
           v siti ---
7      kiv_ppr_network::Save_Relative_Error(neural_networks[j],
           expected_values[i]);
8
9      // --- Spusteni back propagation ---
10     kiv_ppr_network::Back_Prop(neural_networks[j],
           target_values, i, topology[topology.size() - 1]);
11
12 });
```

Po natrénování více instancí neuronové sítě je vybrána instance, která má nejmenší součet průměrné relativní chyby a standardní odchylky těchto chyb. Pro danou instanci jsou vygenerovány stejné výstupy, jako pro neuronovou síť u sériové verze uvedené v kapitole 3.2.

3.4 Program pro asymetrický multiprocesor

Narozdíl od paralelní verze pro symetrický multiprocesor popsané v kapitole 3.3, verze programu pro asymetrický multiprocesor nevychází ze struktur sériové verze. Trénuje se pouze jedna síť a není zde umožněna predikce z načtených parametrů neuronové sítě. Trénování je kompletně realizováno na GPU a je implementováno v `OpenCL`[1]. Data neuronové sítě jsou uložena v bufferech:

- `neural_net_data` – obsahuje hodnoty neuronů a jejich váhy,
- `delta_gradient_data` – slouží k uložení delt vah a gradientů neuronové sítě,
- `input_data` – buffer se vstupními daty (obsahuje vstupy všech trénovacích vzorků),
- `target_data` – buffer, který obsahuje pro každý trénovací vzorek 32 hodnot 0 a 1, kde 1 je na pozici indexu očekávané (naměřené) hodnoty (tzn. 31 hodnot 0 a jedna hodnota 1),
- `helper_data` – slouží pro pomocné výpočty na zařízení,
- `result_indexes` – pro každý průchod sítí je do tohoto bufferu uložen index neuronu výstupní vrstvy s nejvyšší hodnotou (nejvíce aktivovaný),
- `training_set_id` – čítač trénovacích vzorků.

Pro snazší a přehlednější přístup k prvkům jednotlivých bufferů jsou vytvořeny mapovací funkce. Příklad mapovacích funkcí pro přístup k vahám synapsí v bufferu `neural_net_data` je uveden v ukázce kódu 3.6.

[1] Inspirováno článkem: Bikker J, *Optimization & Vectorization / 0ptmzdSummary*

Ukázka kódu 3.6: Mapovací funkce pro přístup k vahám synapsí

```
int kiv_ppr_mapping_gpu::Weight_Input_Hidden1(int input, int
    hidden1) {
  return 100 + input * HIDDEN1_LAYER_NEURONS_COUNT + hidden1;
}

int kiv_ppr_mapping_gpu::Weight_Hidden1_Hidden2(int hidden1,
    int hidden2) {
  return 270 + hidden1 * HIDDEN2_LAYER_NEURONS_COUNT + hidden2;
}

int kiv_ppr_mapping_gpu::Weight_Hidden2_Output(int hidden2, int
     output) {
  return 750 + hidden2 * OUTPUT_LAYER_NEURONS_COUNT + output;
}
```

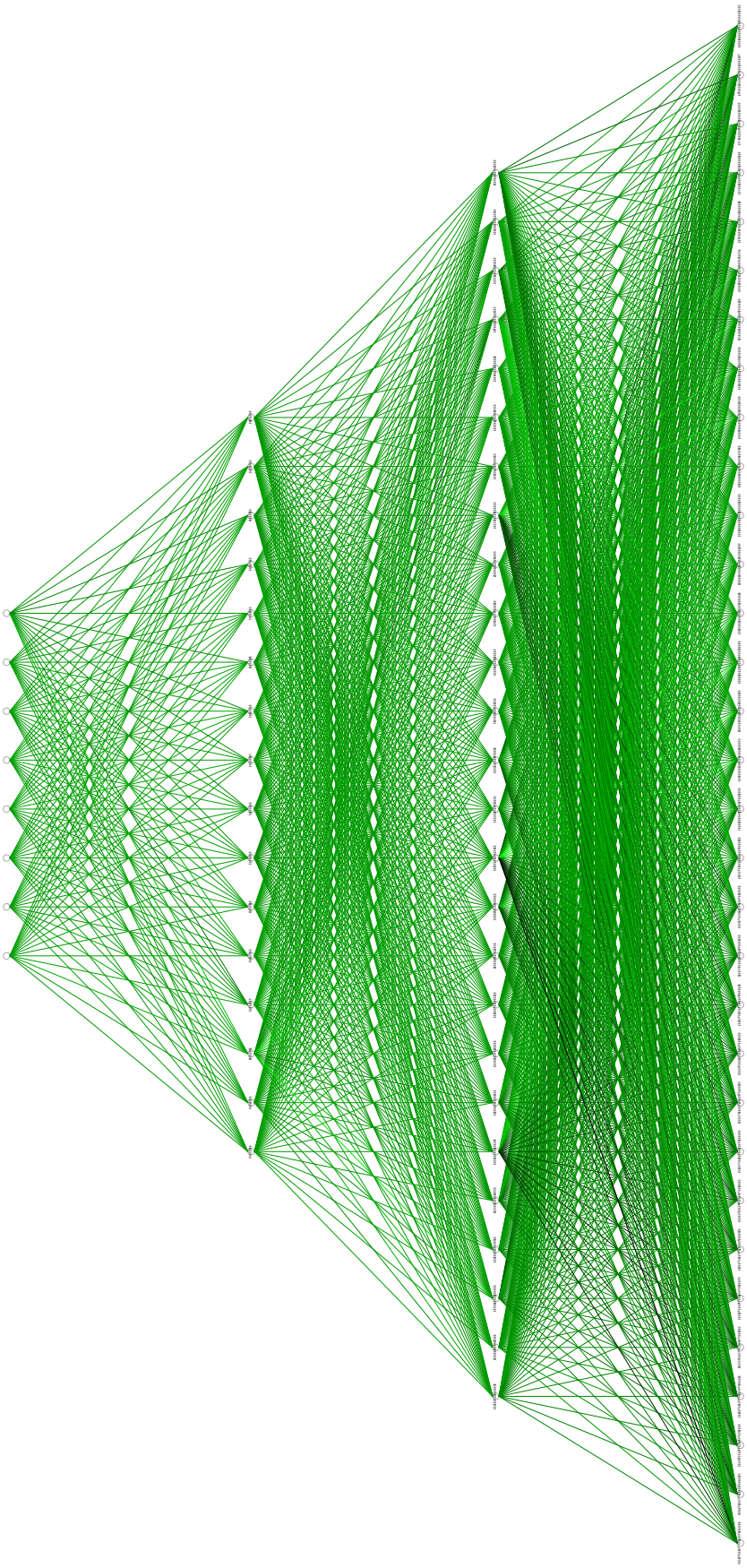
Kód v `OpenCL` obsahuje celkem 11 kernelů, které jsou postupně volány z CPU a starají se o všechny kroky trénování sítě. Příklad kernelu pro *feedforward* mezi vstupní a první skrytou vrstvou je uveden v ukázce kódu 3.7.

Ukázka kódu 3.7: Kernel pro feedforward mezi vstupní a první skrytou vrstvou

```
__kernel void feed_forward_input_hidden1(__global float*
    neural_net_data) {
    int id = get_global_id(0);

    if (id >= HIDDEN1_LAYER_NEURONS_COUNT)
    {
        return;
    }

    for (int i = 0; i <= INPUT_LAYER_NEURONS_COUNT; i++) {
        neural_net_data[hidden1_neuron_i(id)] +=
                neural_net_data[input_neuron_i(i)] *
                neural_net_data[weight_input_hidden1(i, id)];
    }

    neural_net_data[hidden1_neuron_i(id)] =
       transfer_function_hidden(neural_net_data[
       hidden1_neuron_i(id)]);
}
```

Na začátku trénování jsou inicializovány všechny buffery na jejich počáteční hodnoty. Do bufferů `input_data` a `target_data` jsou vloženy všechny trénovací vzorky. Kdyby byly vzorky do bufferů vkládány z CPU postupně před každým průchodem sítí, mnohonásobně by se zvýšila režie na komunikaci mezi CPU a GPU. Po dokončení trénování neuronové sítě na GPU jsou výsledky vyzvednuty z bufferu `result_indexes` (indexy nejvíce aktivovaných neuronů výstupní vrstvy). Indexy jsou poté převedeny na hodnoty glykemie v *mmol/L*, jsou vypočítány relativní chyby, průměrná relativní chyba a standardní odchylka těchto chyb.

Výstupy trénování neuronové sítě na GPU jsou soubory `neural.ini` a `errors.csv`. Grafy neuronové sítě nejsou v této verzi generovány.



Obrázek 3.1: Graf neuronové sítě dle akumulované hodnoty signálů synapsí

4 Uživatelská příručka

V kořenovém adresáři semestrální práce se nacházejí následující podadresáře:

- `data` – obsahuje databázi s trénovacími daty,
- `doc` – obsahuje dokumentaci semestrální práce,
- `msvc` – obsahuje projektové soubory pro *Visual Studio*,
- `out` – zde se nachází výstupní soubory aplikace (grafy, soubor s parametry sítě, CSV soubor relativních chyb a soubor s výsledky predikce),
- `src` – obsahuje zdojové soubory semestrální práce.

4.1 Sestavení programu

Projekt je možné otevřít a sestavit ve vývojovém prostředí *Microsoft Visual Studio* (otevřít soubor `/msvc/BloodGlucoseLevelPredictionChallenge.sln` a sestavit vybráním `Build` → `Build Solution`).

Pro překlad programu je nutné mít nainstalovanou knihovnu `Intel® TBB` a ovladače pro `OpenCL` kompatibilní zařízení. V konfiguračních soubrech projektu je nutné nastavit příslušné cesty ke knihovně `TBB` a `OpenCL`.

Pokud bylo sestavení úspěšné, vytvoří se spustilený `EXE` soubor.

4.2 Spuštění programu

Program je možné spustit z příkazové řádky nebo vývojového prostředí. Argumenty příkazového řádku pro spuštění jsou:

```
<prediction_minutes> <database> [device] [neural_params]
```

- `prediction_minutes` – počet minut dopředu, na které má být provedeno trénování či predikce (hodnota musí být kladné celé číslo dělitelné 5),
- `database` – cesta k databázi obsahující trénovací data,
- `device` (*nepovinný*) – zařízení, na kterém se má provést trénování, 0 = CPU (default), 1 = GPU,
- `neural_params` (*nepovinný*) – soubor s parametry neuronové sítě.

Pokud je zadán parametr `neural_params` obsahující parametry neuronové sítě, nemůže být zadán parametr `device`. V tomto případě bude spuštěna predikce na CPU.

5 Dosažené výsledky

V této kapitole jsou popsány dosažené výsledky semestrální práce. Program byl sestavován v `x86-Release` módu. Měření výsledků bylo realizováno na stroji s těmito parametry:

- Operační systém: `Microsoft Windows 10 Pro 64-bit`,
- CPU: `Intel® Core i5-5200U 2-Cores 4-Threads 2.20 GHz`,
- GPU: `Intel® HD Graphics 5500`,
- RAM: `12 GB`.

5.1 Porovnání rychlosti trénování

Rychlost trénování byla měřena na 65 030 trénovacích vzorcích s predikcí na 60 minut dopředu. Jelikož program pro *asymetrický multiprocesor* trénuje pouze jednu síť, měření rychlosti trénování je porovnáváno odděleně pro:

- *sériovou verzi* a paralelní verzi pro *symetrický multiprocesor*,
- *sériovou verzi* a verzi pro *asymetrický multiprocesor*.

V tabulce 5.1 je uvedeno porovnání rychlosti trénování pro sériovou verzi symetrický multiprocesor. Výsledky porovnání jsou také uvedeny na obrázku 5.1.

Rychlost trénování jedné instance neuronové sítě ve verzi pro asymetrický multiprocesor trvá průměrně 29 487 ms. Tato verze je tedy přibližně $8,137$x pomalejší, než sériová verze. To je způsobeno jak návrhem algoritmu, tak i malou velikostí neuronové sítě (malý počet neuronů v každé vrstvě).

5.2 Chyba sítě

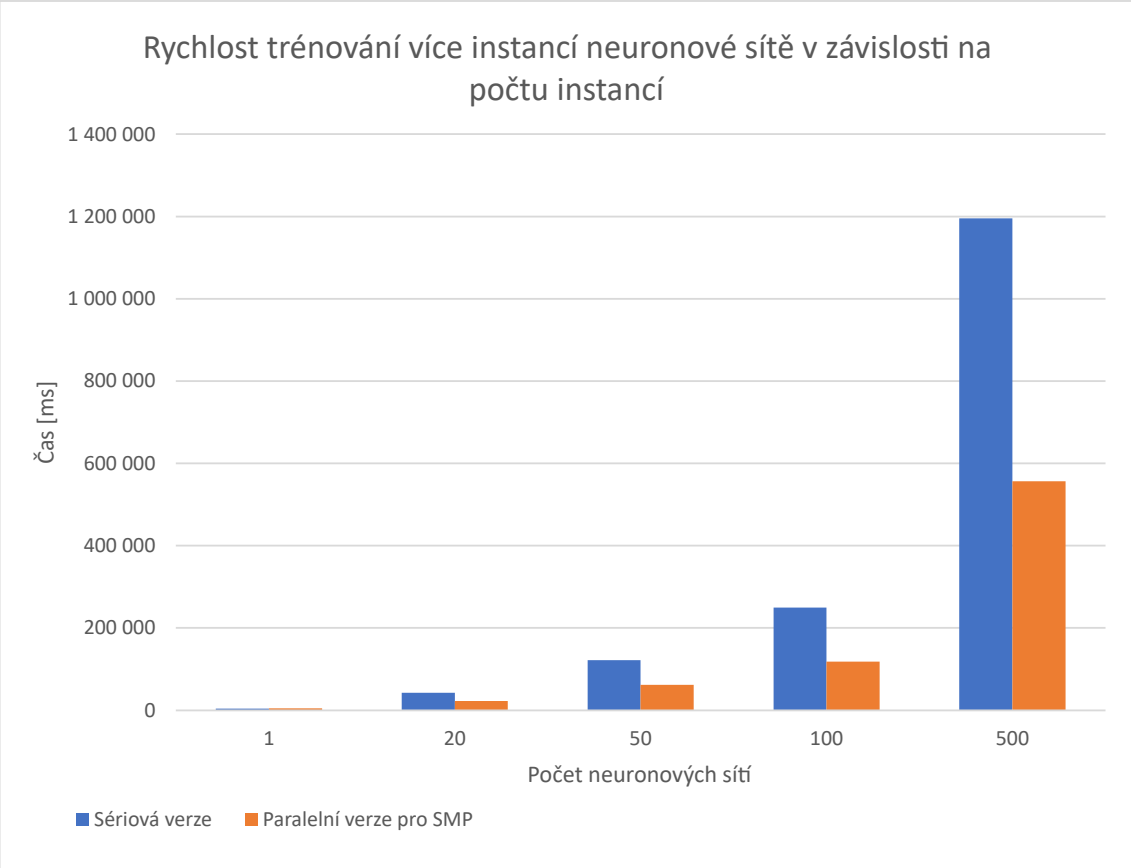
Měření chyby sítě probíhalo na trénování s 65 030 trénovacími vzorky predikcí na 60 minut dopředu. Parametry učení neuronové sítě byly nastaveny následovně:

- $\eta = 0,05$; rychlost učení sítě $\langle 0, 1\rangle$
- $\alpha = 0.1$; momentum (multiplikátor poslední změny váhy) $\langle 0, n\rangle$).

Chyba pro všechny verze programu konverguje okolo hodnoty 0.2.

Tabulka 5.1: Urychlení paralelní verze pro SMP vůči sériové verzi

Počet sítí	**Sériová verze**	**Paralelní verze pro SMP**	**Urychlení paralelní verze**
1	3 624 ms	4 893 ms	0,741x
20	42 491 ms	22 601 ms	1,880x
50	121 853 ms	62 012 ms	1,965x
100	249 534 ms	118 210 ms	2,111x
500	1 195 678 ms	556 213 ms	2,150x



Obrázek 5.1: Graf rychlosti sériové verze a paralelní verze pro SMP

6 Závěr

Cílem této semestrální práce bylo implementovat zjednodušenou verzi problému predikce glykémie na zvolený počet minut dopředu pomocí *feedforward* neuronové sítě s využitím alespoň dvou ze tří možných technologií uvedených v kapitole 1. Řešení semestrální práce implementuje paralelní program pro symetrický multiprocesor a program pro asymetrický multiprocesor (viz kapitola 2.3).

Rychlost trénování více instancí neuronové sítě je u paralelního programu pro symetrický multiprocesor přibližně 2x vyšší, než u sériové verze. Algoritmus trénování neuronové sítě pro asymetrický multiprocesor se ukázal jako ne příliš vhodný vzhledem k malému počtu neuronů v jednotlivých vrstvách sítě. Oproti sériové verzi zde dochází ke zpomalení trénování. K horšímu výkonu přispěla také nutnost převádět datový typ `double` na `float`, protože ne každé `OpenCL` kompatibilní zařízení podporuje *double precision*.

Semestrální práce z mého pohledu splňuje požadavky zadání.