FAKULTA
APLIKOVANÝCH VĚD
ZÁPADOČESKÉ
UNIVERZITY
V PLZNI

Blood Glucose Level Prediction Challenge

KIV/PPR - Paralelní programování

Hoang Ngoc Hung - hoanghun@students.zcu.cz
22. listopadu 2020

Obsah

1 Zadání

Práce se zabývá zjednodušenou verzí problému predikce glykémie, tzv. blood glucose level prediction challenge, na zvolený počet minut dopředu. Provádí se trénování feed forward neuronové sítě, 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-40 minut. IG je koncentrace glukózy v intersticiální tekutině. Buď bude použit výpočet běžícího průměru, a průběžnou aktualizaci minimálních a maximálních hodnot IG, anebo bude použita funkce risk.

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 posktytnutému kódu. Každý prvek bude mít svůj bias a váhy. Zde použijte funkci SoftMax.

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 bezpresotředně 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. Učení sítě není povinné, můžou se nechat jednotlivé váhy.

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 zvoli 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 (setřídítě 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.

Samostatná 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.

2 Analýza

V této kapitole popíšu princip neuronové sítě, trénovací data a technologie.

2.1 Neuronová síť

Umělá neuronová síť je matematickým modelem biologických struktur neuronů u živých organismů. Skládá se z umělých neuronů, které jsou propojeny pomocí synapsí a uspořádany do vrstev. Každý umělý neuron má pouze jeden výstup a libovolný počet vstupů.

Každý neuron je propojen s neurony v předchozí vrstvě, které mu předávají signál. Spojení mezi neurony mají váhy (tzv. synaptické váhy), které se během

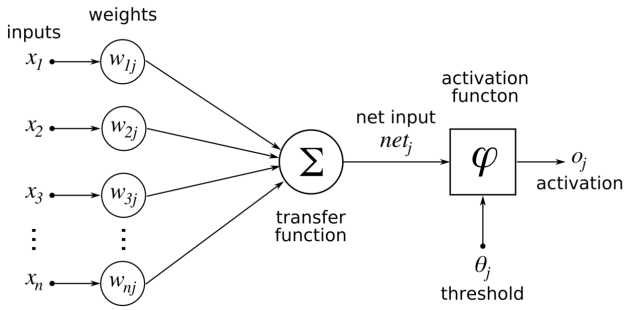
učení sítě upravují, aby síť dosahovala co nejlepších výsledků.

Umělá neuronová síť je vhodná na úlohy z oblasti klasifikace, aproximace a predikace.

Existuje celá řada modelů, které umělý neuron popisují. Jedním z nejpoužívanějších je McCulloch-Pitsův model neuronu, viz obr. 1. Jednotlivé neurony jsou popsány uspořádanou trojicí

$$(w, \varphi, \Theta),$$

kde $w = (w_1, .., w_n) \in \mathbb{R}$ je vektor synaptických vah, $\varphi : \mathbb{R} \longrightarrow \mathbb{R}$ je aktivační funkce neuronu a $\Theta \in \mathbb{R}$ je práh.



Obrázek 1: Model umělého neuronu

Vstup je reprezentován vektorem $x = (x_1, .., x_n) \in \mathbb{R}$, každá složka vektoru má vlastní váhu, která ji přisuzuje významnost. Vážená suma složek vektoru společně s prahem tvoří *vnitřní potenciál* neuronu ξ, vypočítán podle (1).

$$\xi = \sum_{i=1}^{n} w_i x_i + \theta \tag{1}$$

Výstup neuronu je získán použitím aktivační funkce, v našem případě `tanh`, na vnitřní potenciál neuronu dle rovnice (2).

$$o = \varphi(\sum_{i=1}^{n} w_i x_i + \theta) \tag{2}$$

Na výstupu naší neuronové sítě je 32 neuronů. Zjistíme index neuronu, který má největší výstup, a tento index použijeme jako argument funkce `Band_Index_To_Level`, viz fragment kódu (1). Toto je hodnota, kterou predikuje naše neuronová síť.

Chybu vypočítáme pomocí následující rovnice

$$E = \frac{|x - y|}{y}, \tag{3}$$

kde x je hodnota predikovaná sítí, y je hodnota, která měla vyjít a E je relativní chyba.

```
static constexpr double Low_Threshold = 3.0;       //mmol/L below which
    a medical attention is needed
static constexpr double High_Threshold = 13.0;      //dtto above
static constexpr size_t Internal_Bound_Count = 30; //number of bounds
    inside the thresholds

static constexpr double Band_Size = (High_Threshold - Low_Threshold) /
    static_cast<double>(Internal_Bound_Count);          //must imply
    relative error <= 10%
static constexpr double Inv_Band_Size = 1.0 / Band_Size;
    //abs(Low_Threshold-Band_Size)/Low_Threshold
static constexpr double Half_Band_Size = 0.5 / Inv_Band_Size;
static constexpr size_t Band_Count = Internal_Bound_Count + 2;

double Band_Index_To_Level(const size_t index) {
    if (index == 0) return Low_Threshold - Half_Band_Size;
    if (index >= Band_Count - 1) return High_Threshold + Half_Band_Size;

    return Low_Threshold + static_cast<double>(index - 1)*Band_Size +
        Half_Band_Size;
}
```

Fragment kódu 1: Převedení indexu do hodnoty ist

2.2 Zpracování chyb

Neuronová síť pro každý vstup musí uložit relativní chybu. Z uložených chyb se potom vypočítá průměr a střední odchylka dle (4) a (5).

$$\overline{x} = \frac{1}{n} \cdot \sum_{i=1}^{n} \tag{4}$$

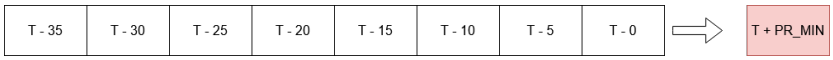
$$\sigma = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (x_i - \overline{x})^2} \tag{5}$$

Součet průměru a střední odchylky určuje natrénovanost neuronové sítě.

2.3 Trénovací data

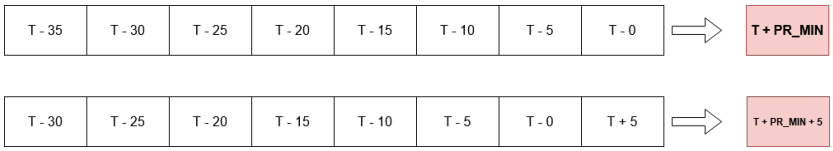
Data z databáze jsou dvojice čas měření a hodnota IG. Vstupem neuronové sítě musí být 8 hodnot, které mají mezi sebou rozestupy pěti minut, viz obrázek 2,

vytvořená osmice musí mít validní predikovanou hodnotu, to je hodnota, která má čas `T` + počet minut na kterou se dělá predikce. Predikovaná hodnota se použije k výpočtu relativní chyby. Nemá-li osmice k sobě validní predikovanou hodnotu, nelze ji použít k učení sítě a výpočtu relativní chyby.



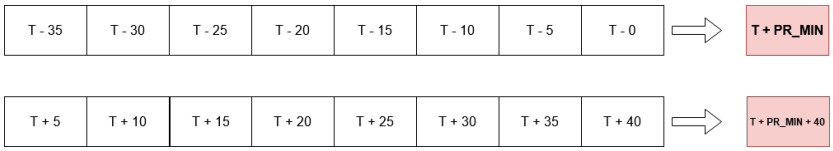
Obrázek 2: Příklad vzorku z trénovacích dat s predikovanou hodnotou.

Vytvoření trénvacích dat můžeme dělat pomocí posuvného okénka, které se bude posouvat o jednu hodnotu viz obrázek 3.



Obrázek 3: Vytváření dat pomocí posuvného okénka.

Druhou možností je posouváním celého okna o celou osmici viz obrázek 4.



Obrázek 4: Vytváření dat posouváním po celé velikosti vstupu.

Použita bude možnost s posunutím o jedna, díky tomu bude vytvořena větší trénovací sada než kdyby byla použita alternativa.

2.4 Technologie

Pro symetrický multiprocesor bude použita knihovna `Parallel STL`, implementace `STL` knihovny s podporou tzv. execution policy pro standard `C++17` a výše. `Parallel STL` nabízí efektivní podporu pro paralelizaci a vektorizaci provedení algoritmů.

K implementaci programu využívající asymetrický multiprocesor bude použit průmyslový standard `OpenCL`, který se používá k paralelizaci heterogenních

počítačových systémů, jako jsou například osobní počítače vybavené CPU a GPU[1].

3 Řešení

V této kapitole bude popsáno řešení programu pro jednotlivé technologie.

3.1 Sériová verze

Neuronová síť je definována dle kódu ve fragmentu 2, kde jsou ukázany pouze data bez jednotlivých metod.

Základním prvkem je synapse, která je reprezentována strukturou `Connection`, v ní se uchovávají váhy a čítače intenzit. Jsou dva druhy čítačů, první akumuluje absolutní hodnotu výstupu dané synapse, druhý akumuluje pouze v případě, že neuronová síť vyprodukovala predikci s chybou menší než 0.15.

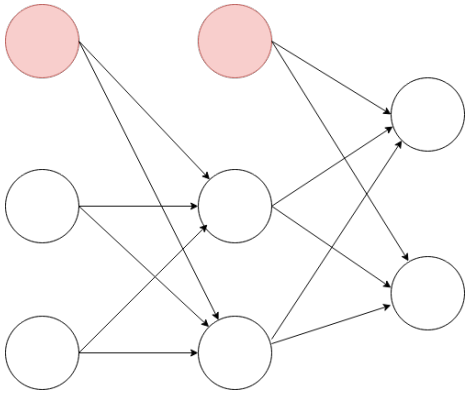
Neuron je reprezentován třídou `Neuron`. Každá instance neuronu má pole synapsí o velikosti vrstvy, která následuje vrstvu v níž je daný neuron. Jednotlivé vrstvy jsou pole těchto neuronů. Neuronová síť je opět pouze pole vrstev.

```
typedef std::vector<Neuron> Layer;
struct Connection {
   double weight;
   double intensity_counter;
   double xai_intensity_counter;
   double current_intensity;
};

class Neuron {
private:
    double output_signal; // vystup neuronu
    size_t index;
    std::vector<Connection> weights;
}

class Neural_Network {
public:
    void feed_forward(const std::vector<double>& input_vals);
private:
    std::vector<Layer> layers;
}
```

Fragment kódu 2: Definice neuronu

[1] https://cs.wikipedia.org/wiki/OpenCL

Každá vrstva obsahuje o neuron navíc, jedná se o falešný neuron, který slouží jako bias pro další vrstvy. Na obrázku 5 jsou falešné neurony zobrazeny růžovou barvou. Žádné synapse do nich nevedou, pouze z nich.



Obrázek 5: Vytváření dat posouváním po celé velikosti vstupu.

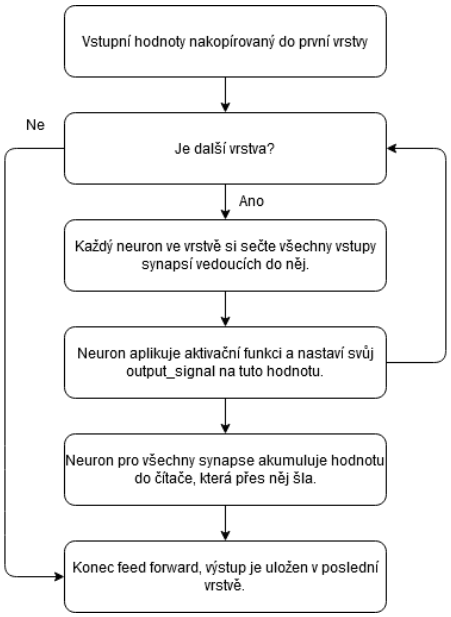
Jednotlivé váhy jsou inicializovány náhodně dle rovnoměrného rozdělení v rozsahu $\langle -1, 1)$ pomocí generátoru ze standardní knihovny jazyka.

```
std::random_device rd;
std::mt19937 gen(rd());
std::uniform_real_distribution<> distr(-1.0, 1.0);
// kus kodu vynechan
weights.push_back(Connection());
weights.back().weight = distr(gen);
```

Fragment kódu 3: Inicializace vah

3.1.1 Průchod sítí

Hodnoty vstupu do sítě jsou nakopírováný do první vrstvy sítě. Tyto hodnoty jsou nastaveny jako výstupy jednotlivých neuronů. Pokračuje se na následující vrstvě. Jednotlivé neurony sečtou vstupy synapsí pomocí reference na předchozí vrstvu, poté aplikují aktivační funkci. Tento postup je ukázan na obrázku 6.



Obrázek 6: Diagram feed forward.

Výstupní vrstva neprovádí žádnou operaci, jako je např. softmax. Ten je implementován, ale z počátku nebyl využíván kvůli zhoršení chyby, v případě učení sítě. V případě, kdy se síť neučí, softmax nepřidává žádnou hodnotu. Tento proces se opakuje pro každý vzorek z trénovací sady.

3.1.2 Zpracování výsledků sítě

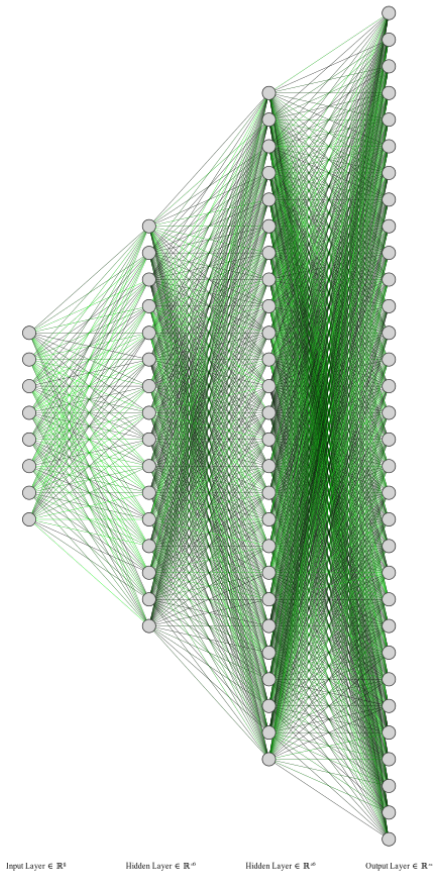
Relativní chybu získáme postupem popsaným na konci kapitoly 2.1. Chyba se přidá do pole, ze kterého se následně spočítá průměrná relativní chyba a směrodatná odchylka chyb. Zpracování chyb bylo popsáno v kapitole 2.2. Z chyb je vytvořena empirická kumulativní distribuční funkce navzorkována po 1%. Tyto hodnoty jsou uloženy do `csv` souboru.

Je-li chyba sítě menší nebo rovna 0.15, každá synapse akumuluje hodnotu do XAI (Explainable artificial intelligence) čítače.

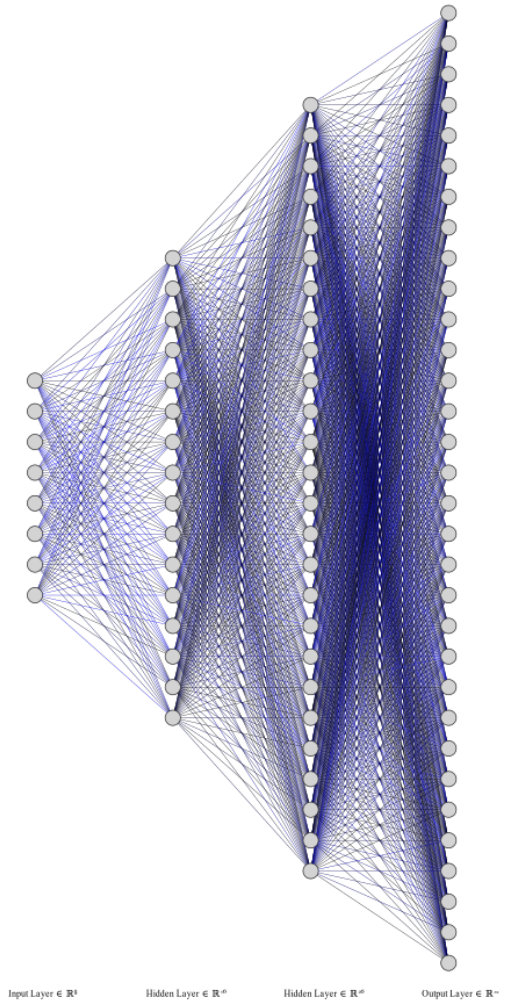
Parametry sítě se uloží do souboru `neural.ini`. Výstupní soubor se drobně liší od zadaného formátu. Neukládá se řádka `NeuronN_Bias=`, protože se používá falešný neuron. Místo toho je uložena váha pro příslušný falešný neuron.

3.1.3 Graf reprezentující síť

Neuronová síť umí vytvořit dva grafy, jeden pro klasický čítač a druhý pro XAI čítač. Grafy jsou ve formátu `svg`. Hodnota, která se akumuluje, je absolutní hodnota signálu synapse. Důvodem je zobrazení synapsí, které posílají velké množství záporné signály, jako neaktivní. Hodnoty jsou normalizovány přes vrstvy, díky tomu je lépe vidět aktivita synapsí v jednotlivých vrstvách. Bez této normalizace se synapse v první vrstvě jeví jako neaktivní, přestože se na významně podílejí na výsledku. Graf klasického čítače je na obrázku 7 a pro čítač XAI je na obrázku 8.



Obrázek 7: Obrázek grafu pro klasický čítač.



Obrázek 8: Obrázek grafu pro čítač XAI.

3.2 Paralelizace na symetrickém multiprocesoru

Samotná neuronová síť se neparalelizuje. Paralelizováno je učení několika neuronových sítí najednou.

K paralelizaci se používá standardní metoda `std::for_each`, které lze říci, jestli chceme paralelní běh algoritmu.

Na začátku je vytvořeno n instancí neuronové sítě, které se paralelně učí, respek-

tive paralelně provádějí pouze feed forward. Paralelizace je provedena pomocí tohoto kódu

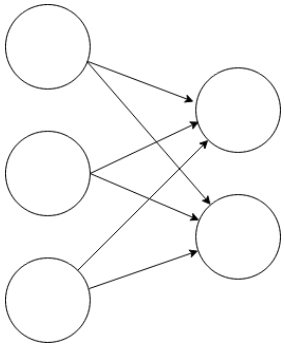
```
std::for_each(std::execution::par, training.begin(), training.end(),
   [&training_set](std::pair<Neural_Network, Results>& pair) {
      train_single_network(pair.first, pair.second, training_set,
          false);
   } // false == no back prop
);
```

Fragment kódu 4: PSTL paralelizace

Následně se vybere neuronová síť, která měla nejmenší průměrnou relativní chybu. Ta je považovavána jako nejúspěšnější a její výsledky se zpracují dle kapitoly 3.1.2, její chyby se uloží do `csv`, vytvoří se grafy a uloží váhy do souboru `neural.ini`.

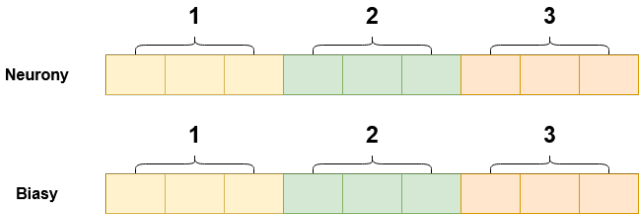
3.3 Asymetrický multiprocesor - OpenCL

Při implementaci algoritmu pro `GPGPU` byl kladen důraz na paralelní běh několika sítí najednou. Korespondující vrstvy jednotlivých neuronových sítí jsou spojeny dohromady do jednoho pole. Na obrázku 9 je jednoduchá neuronová síť se vstupní vrstvou o 3 neuronech a výstupní o 2 neuronech.



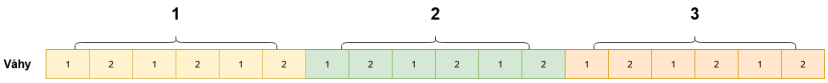
Obrázek 9: Neuronová síť s dvěma vrstvama.

Na obrázku 10 je zobrazena reprezentace neuronů, resp. hodnot jejich výstupů, a biasů pro první vrstvu a pro 3 instance neuronových sítí. Dohromady je tedy 9 neuronů a 9 biasů. Čísla jsou identifikátory neuronových sítí.



Obrázek 10: Neurony v první vrstvě pro 3 neuronové sítě.

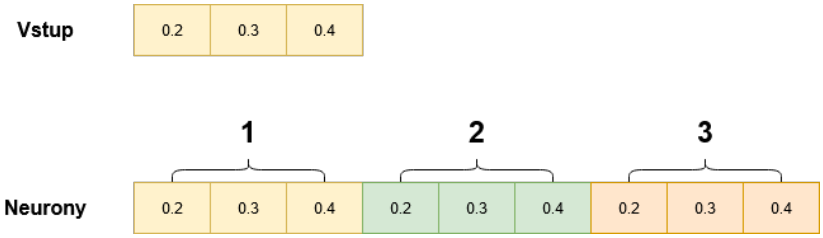
Pro každý neuron jsou dvě synapse vedoucí do další vrstvy, celkový počet vah pro všechny neurony a neuronové sítě je 18. Váhy jsou znázorněny na obrázku 11, kde čísla ve čtverci jsou identifikátory neuronu v následující vrstvě. Váhy jsou inicializovány stejným způsobem jako váhy v sériové verzi.



Obrázek 11: Pole váh pro všechny neurony ve všech neuronových sítích.

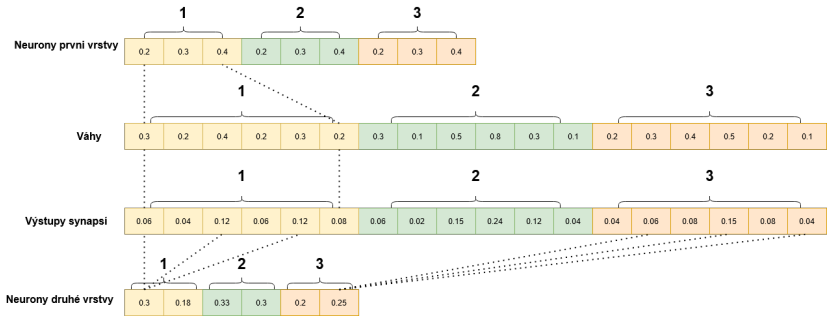
3.4 Průchod sítí

Vstup je nakopírován do první vrstvy neuronů. Vstup je tedy replikován tolikrát, kolik máme instancí neuronových sítí. Tento proces je ukázán na obrázku 12.



Obrázek 12: Kopírování vstupu do první vrstvy

Je vytvořené jedno pomocné pole, které má stejnou velikost jako váhy. Slouží k výpočtu výstupu jednotlivých synapsí. Postup na další vrstvu je ukázán na obrázku 13. Výpočet je ukázán bez biasů a aktivační funkce.



Obrázek 13: Průchod přes jednu vrstvu bez biasů a aktivační funkce.

Tento výpočet by se v sériovém podání dal implementovat dle následujícího kódu.

```
size_t first_layer_size = 3;
size_t second_layer_size = 2;
size_t outputs_first_layer_size = first_layer_size * second_layer_size;
for (size_t nni = 0; nni < neural_networks_count; nni++) {
   for (size_t i = 0; i < first_layer_size; i++) {
      for (size_t j = 0; j < second_layer_size; j++) {
         size_t offset = i * second_layer_size + j;
         size_t nn_offset = nni * (outputs_first_layer_size);
         float weight = first_layer_weight[nn_offset + offset];
         float output = first_layer_neurons[nni * first_layer_size + i];
         first_layer_synapses_output[neural_networks_index * offset] =
             weight * output;
      }
   }
}

for (size_t nnid = 0; nnid < neural_networks_count; nnid++) {
   for (size_t i = 0; i < second_layer_size; i++) {
      size_t nn_offset = nnid * (outputs_first_layer_size);
      float sum = 0;
      for (size_t j = i; j < outputs_first_layer_size; j++) {
         sum += first_layer_synapses_output[nn_offset + j];
      }
      second_layer_neurons[nni * second_layer_size + i] = sum;
   }
}
```

Fragment kódu 5: Sériová implementace

Výpočet byl implementován pomocí dvou kernelů, které jsou ukázany ve fragmentu 6.

```
// dimenze pocet siti, velikost predchozi vrstvy, velikost dalsi vrstvy
__kernel void FeedForward(__global float* input, __global float*
    weights, __global float* layer_outputs, int previous_layer_size,
    int layer_size) {
  int previous_layer_id = get_global_id(0);
  int layer_id = get_global_id(1);
  int neural_network_id = get_global_id(2);

  int neural_network_offset = neural_network_id * (previous_layer_size
      * layer_size);
  int offset = previous_layer_id * layer_size + layer_id;

  float weight = weights[neural_network_offset + offset];
  float input_val = input[neural_network_id * previous_layer_size +
      previous_layer_id];

  layer_outputs[neural_network_offset + offset] = weight * input_val;
}

// dimenze pocet siti, velikost vrstvy
__kernel void FeedForwardSum(__global float* input, __global float*
    layer_outputs, __global float* layer_biases, int layer_size, int
    previous_layer_size) {
  int output_layer_id = get_global_id(0);
  int neural_network_id = get_global_id(1);

  int neural_network_offset = neural_network_id * (layer_size *
      previous_layer_size);
  float sum = 0;

  for (size_t i = output_layer_id; i < previous_layer_size *
      layer_size; i += layer_size) {
     sum += input[neural_network_offset + i];
  }

  layer_outputs[neural_network_id * layer_size + output_layer_id] =
      tanh(sum + layer_biases[output_layer_id]);
}
```

Fragment kódu 6: OpenCL kernely pro výpočet další vrstvy

Tyto operace se opakují pro každou vrstvu v neuronové síti, dokud nedostaneme výsledky ve výstupní vrstvě. Z té zjistíme chyby jednotlivých neuronových sítí a přidáme je do pole chyb neuronových sítí. Algoritmus učení nebyl implementován v `OpenCL`.

Stejně jako ve verzi symetrického multiprocesoru se najde neuronová síť s nejmenší průměrnou relativní chybou a ta je považována za nejlepší síť. Váhy, chyby ani

grafy nejsou vytvářeny.

4 Výsledky

V této kapitole budou popsány výsledky a jejich srovnání.

4.1 Konfigurace

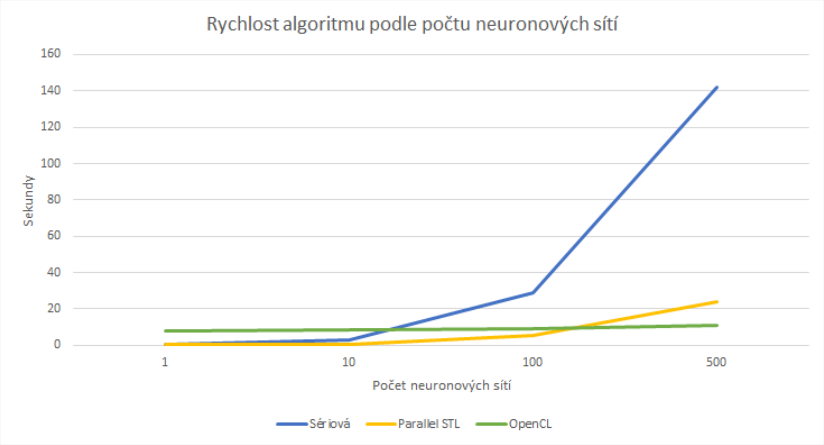
Všechny běhy byly prováděny na stroji s touto konfigurací:

- Operační system: Windows 10 Pro
- Procesor: AMD Ryzen 5 3600 6-Core Processor 3.60 GHz
- RAM: 32.0 GB
- Typ systému: 64-bit operační systém, x64-based processor
- GPU: GeForce RTX 2080 SUPER

Program byl přeložen v release módu.

4.2 Rychlost algoritmů v závislosti na počtu neuronových sítí

Probíhala predikce na 60 minut, trénovací sada obsahovala 64536 vzorků. Každý běh byl puštěn pětkrát a výsledek je jejich průměrem. Na obrázku 14 jsou výsledky sériové, PSTL a OpenCL verze. Z grafu a tabulky 1 lze vyčíst, že `PSTL` verze má větší urychlení pouze do počtu sítí kolem 150. Poté je `OpenCL` rychlejší.

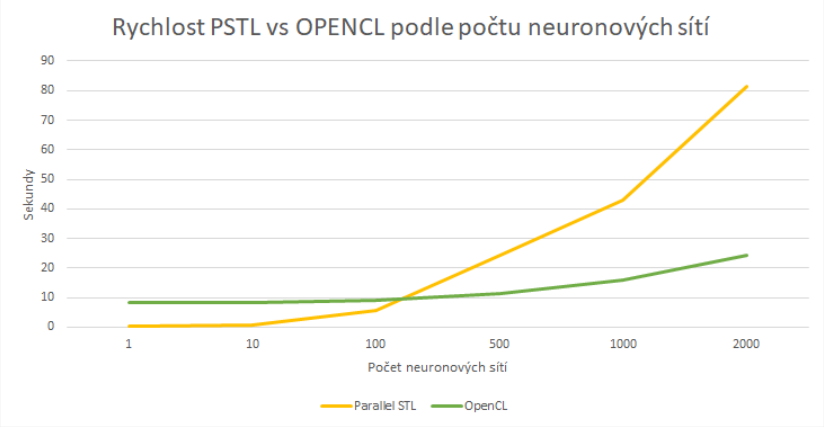


Obrázek 14:

Počet sítí	Sériově	PSTL	OpenCL	Urychl. PSTL	Urychl. OpenCL
1	284.6	293	8180	0.971x	0.035x
10	2864.4	749.4	8242.6	3.822x	0.0348x
100	28615.4	5587.8	8926.4	5.139x	3.217x
500	141828.8	24215	11214.8	5.857x	12.647x

Tabulka 1: Urychlení PSTL a OpenCL algoritmů vůči sériové verzi, čas v MS.

Na obrázku 15 je porovnání rychlostí implementace pomocí `PSTL` a `OpenCL`. Na křivkách jednotlivých implementací vidíme, že `OpenCL` je pomalejší v případě malého počtu neuronových sítí. Rychlost obou algoritmů je lineárně závislá na počtu neuronových sítí, v případě `OpenCL` je koeficient mnohem nižší. Urychlení `OpenCL` oproti `PSTL` je v tabulce 2.



Obrázek 15: Graf rychlosti algoritmu závislého na počtu neuronových sítí.

Počet neuronových sítí	PSTL	OpenCL	Urychlení OpenCL oproti PSTL
1	293	8180	0.035x
10	749.4	8242.6	0.091x
100	5587.8	8926.4	0.0626x
500	24215	11214.8	2.159x
1000	42938.4	15795.8	2.718x
2000	81526.8	24377	3.344x

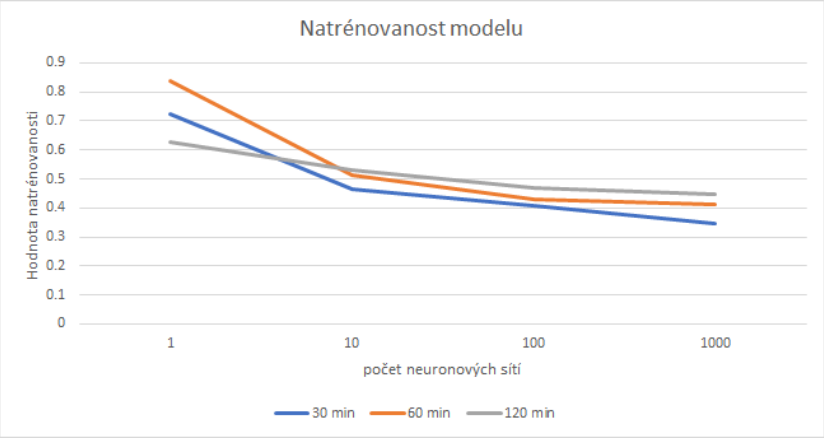
Tabulka 2: Urychlení OpenCL oproti PSTL, časy jsou v ms.

Algoritmus implementován pomocí `PSTL` je rychlejší do hranice 150 neuronových sítí. Poté je výrazně rychlejší implementace pomocí `OpenCL`.

4.3 Chyba sítě

Žádná z verzí síť neučí, všechny váhy jsou náhodně vygenerovány a zůstávají stejné po celý běh. Chyby sítě budou na základě výstupu algoritmu implementovaného pomocí `PSTL`, ostatní implementace dávají podobné výsledky.

Pokusy byly provedeny s predikcí na 30, 60 a 120 minut. Natrénovanost je součet průměrné relativní chyby a střední odchylky. Výsledky jsou na obrázku 16. Model podává nejlepší výsledky při predikci na 30 minut. Výsledky nejsou dobré, to se ale dá očekávat, jelikož se neuronová síť neučí.



Obrázek 16: Natrénovanost modelu v závislosti na počtu sítí a počtu minut predikce.

5 Uživatelská příručka

Práce byla napsána v jayzce `C++` ve vývojovém prostředí `Visual Studio 2019`[2].

V adresáři se nachází:

- doc - adresář s dokumentací a excelovým souborem, kde se nacházejí výsledky,
- msvc - adresář s projektovými soubory, jako je např. solution,
- opencl - adresář se zdrojovými soubory a knihovnami OpenCL,
- src - zdrojové kódy,
- asc2018.sqlite - databáze s daty.

[2]https://visualstudio.microsoft.com/cs/

Projekt má závislost na knihovně Thread Building Blocks, tu lze získat zde `https://software.seek.intel.com/performance-libraries`. Projekt lze otevřít ve vývojovém prosředí Visual Studio (otevření souboru `msvc/PPR.sln`), ve kterém ho je možné sestavit příkazem `Build -> Build Solution`.

Projekt má nastavené `Warning Level - Level4(/W4)`. Zdrojové kódy napsané mnou by neměly hlásit žádné warningy, přidané knihovny třetích stran jako je např. `sqlite3` však produkují velké množství warningů.

5.1 Spuštění

Ve zmíněném vývojovém prostředí ho i lze spustit `Debug -> Start Debugging` nebo klávesou `F5`. Na příkazové řádce `PPR.exe`.

Syntax spuštění je:

```
> PPR.exe MINUTES_PREDICTION SQLITE_DB_PATH [options]
```

První dva argumenty jsou povinné a to:

- `MINUTES_PREDICTION` - počet minut na kterou se bude provádět predikce,
- `SQLITE_DB_PATH` - cesta k souboru s daty.

Program nabízí několik volitelných argumentů (`[options]`):

- `-t WEIGHTS_FILE` - načtení souboru `WEIGHTS_FILE` ve kterém jsou váhy v požadovaném formátu, běží pouze jedna síť, pokud není použitý další `option` (`-all, -pstl, -opencl, -serial`),
- `-s NETWORKS_SIZE` - nastaví počet neuronových sítí, které budou provádět feed forward, defaultně je 100,
- `-serial` - pustí pouze algoritmus v sériové verzi,
- `-pstl` - pustí pouze algoritmus v PSTL verzi,
- `-opencl` - pustí pouze algoritmus v OpenCL verzi,
- `-all` - pustí všechny algoritmy.

Spuštění může např. vypadat takto:

```
> PPR.exe 30 ../asc2018.sqlite
```

V tomto případě pustíme všechny 3 verze algoritmu každý se 100 neuronovými sítěmi, a predikujeme na 30 minut.

```
> PPR.exe 30 ../asc2018.sqlite -s 20
```

Pustíme všechny algoritmy, počet neuronových sítí je 20.

```
> PPR.exe 30 ../asc2018.sqlite -s 20 -opencl
```

Pustíme pouze `OpenCL` verzi, počet neuronových sítí je 20.

```
> PPR.exe 30 ../asc2018.sqlite -s 20 -t neural.ini -all
```

Pustíme všechny algoritmy, počet neuronových sítí je 20. Navíc načteme váhy ze souboru `neural.ini` a spustíme feed forward nad jednou sítí.