Semestrální práce

Vícevrstvý perceptron pro automatizaci domácnosti

Vypracoval : Lískovský David

Předmět : 2VINS + 2NESI

Datum : 25.11.2024

Obsah

Úvod

V rámci předmětu Vývoj inteligentních systémů se zaměřuji na projekt, který demonstruje praktickou aplikaci umělé inteligence pomocí mikropočítače Micro:bit. Cílem projektu je simulovat fungování inteligentní domácnosti, kde neuronová síť řídí větrání a stínění na základě aktuálních podmínek, konkrétně teploty a intenzity světla. Tento projekt je ukázkou jednoduchého systému, který lze implementovat pro domácí nebo kancelářské využití.

Neuronová síť v projektu slouží jako alternativa k tradičnímu programování. Namísto pevných pravidel používá systém datově orientovaný přístup, kdy se model učí na historických datech, aby byl schopen předpovídat vhodné reakce na nové vstupy.

Popis využitých komponentů

K provedení projektu, budeme potřebovat hardwarové prostředky systému Micro:Bit. Pro potřeby tohoto projektu postačí startovní sada:

BBC micro:bit Starter Kit

Softwarových nástrojů k programování mikropočítače je hned několik, ale v tomto projektu bude využit software od společnosti Microsoft:

MAKEKODE.MICROBIT.ORG

a vývojového prostředí IntelliJ s knihovnami podporující implementaci neuronových modelů.

Hardware

Je využito následujících komponentů:

- 1 x Deska Micro:bit V2
- 1 x Rozšiřující modul pro kontaktní pole
- 2 x Kontaktní pole 83 x 55 mm
- 1 x micro:servo 180° pro micro:bit
- 1 x Rezistor 323 Ω
- 1 x 5 V motor s větráčkem
- 1 x Výkonový tranzistor npn
- 1 x Dioda
- 1 x Kondenzátor 100 µF
- sada propojovacích vodičů

Rozšiřující modul

Využití rozšiřujícího modulu pro kontaktní pole, které jednoduše zasuneme do nepájivého pole a desku Micro:bit V2 do něj. Díky tomu získáme snadný a stabilní přístup ke všem pinům na desce.

Obrázek 1 - Rozšiřující modul

Servomotor

Servomotor použitý v projektu bude simulovat pohyb žaluzií, tedy dle slunečního svitu nám zatáhne či otevře žaluzie.

Obrázek 2 - Servomotor

Větrák

Motorek s větráčkem, v projektu jej uvidíme jako komponent reagující na změnu teploty, při zvýšené teplotě prostředí se spustí a bude tak simulovat funkčnost klimatizace nebo reálného větráku.

Obrázek 3 - Motorek větráku

Shrnutí

Z desky Micro:bit využijeme integrovaný snímač teploty. Zjištěné hodnoty se budou ukládat pro pozdější trénování neuronové sítě a zjišťování aktuální teploty okolí. Další veličinou, kterou hlídáme, je sluneční svit. Ten registrujeme pomocí snímače osvětlení, který je taktéž integrovaný v desce Micro:bit. Zjištěné hodnoty se budou opět ukládat pro pozdější trénování neuronové sítě a zjišťování aktuální intenzity osvětlení okolí.

Softwarové řešení

1. ***Sběr dat*** - Systém shromažďuje data ze senzorů Micro:Bitu, jako je teplota a světlo. Tato data jsou následně použita k trénování neuronové sítě.

Softwarová část na straně Micro:Bitu (zkráceno):

```
let temperature = 0
let Light = 0
// Hlavní smyčka pro měření a odesílání dat (bez zobrazení na displeji)
basic.forever(function () {
    Light = input.lightLevel()
    temperature = input.temperature()
    let data = {
        "Teplota": temperature,
        "Svetlo": Light
    }
    // Odeslání jako JSON řetězec s delimitery
    serial.writeLine("$" + JSON.stringify(data) + "$")
    basic.pause(10000)
})
```

Softwarová část na straně IntelliJ (zkráceno):

```
try {
byte[] readBuffer = new byte[1024];
int bytesRead = comPort.readBytes(readBuffer, readBuffer.length);
if (bytesRead > 0) {
String receivedData = new String(readBuffer, 0, bytesRead).trim();
dataBuffer.append(receivedData);
String dataString = dataBuffer.toString();
int start = dataString.indexOf('$');
int end = dataString.indexOf('$', start + 1);
if (start != -1 && end != -1) {
String jsonString = dataString.substring(start + 1, end);
try {
JsonObject jsonData = JsonParser.parseString(jsonString).getAsJsonObject();
writer.write(jsonData.toString() + ",\n");
writer.flush();
dataBuffer.delete(0, end + 1);
} catch (Exception e) {
System.err.println("Chyba při parsování JSON: " + e.getMessage());
}}}} catch (Exception e) {
System.err.println("Chyba při zpracování dat: " + e.getMessage());
}}});
```

Shrnutí:

Microbit: Měří teplotu a světlo, odesílá data jako JSON řetězec každých 10 sekund.

IntelliJ:

- Inicializuje a otevírá sériový port.
- Nastavuje listener pro čtení dat.
- Čte a parsuje data ze sériového portu.
- Ukládá JSON data do souboru.

Výsledný formát přijímaných dat:

```
[{"Teplota": 14, "Svetlo": 81},
{"Teplota": 27, "Svetlo": 185},
{"Teplota": 26, "Svetlo": 178}]
```

2. ***Normalizace a úprava dat*** – Shromážděná data je potřeba doplnit o informace očekávaných výstupů. Nejjednodušší formou doplnění datové sady je použitím Excelu, kde dodám potřebné sloupce a jednoduchou podmínkou:

   Ventilátor: $= KDYŽ(teplota < 25; 0; KDYŽ(teplota <= 33; 1; 2))$

   Žaluzie: $= KDYŽ(světlo < 150; 0; KDYŽ(světlo <= 190; 1; 2))$

Výsledný formát trénovacích dat:

$$[\{"Teplota": 14, "Svetlo": 81, "Rychlost": 0, "Natoceni": 0\},$$
$$\{"Teplota": 27, "Svetlo": 195, "Rychlost": 1, "Natoceni": 2\},$$
$$\{"Teplota": 28, "Svetlo": 244, "Rychlost": 1, "Natoceni": 2\},$$
$$\{"Teplota": 31, "Svetlo": 12, "Rychlost": 1, "Natoceni": 0\},$$
$$\{"Teplota": 26, "Svetlo": 178, "Rychlost": 1, "Natoceni": 1\}]$$

Celkový objem trénovacích dat aktuálně činí 1000 řádků naměřených hodnot a očekávaných reakcí. Při trénování neuronové sítě jsem se často potýkal s jednotvárnou reakcí modelu, protože některé očekávané reakce byly zastoupeny více a jiné zase méně, což způsobovalo, že model některé predikce "přehlížel". Musel jsem proto standardizovat data do rovnoměrného zastoupení očekávaných hodnot, kdy je teď každá zastoupená cca 330krát a mělo by tak být zajištěno vyváženosti trénovací sady.

Co je neuronová síť a proč ji používáme?

Neuronové sítě jsou matematické modely, které se učí rozpoznávat vzory v datech. Fungují podobně jako mozek, kde neurony přijímají vstupy, kombinují je a na základě těchto vstupů rozhodují o výstupech. V tomto případě neuronová síť přijímá informace o teplotě, intenzitě světla a očekávaných parametrech a na základě těchto informací rozhoduje, jaký by měl být výstup - jak rychle má běžet ventilátor nebo jak natočené mají být žaluzie. Výhody neuronových sítí zahrnují flexibilitu a schopnost se učit z dat, což je činí ideálními pro komplexní úkoly, jako je automatizace domácnosti.

Vysvětlení neuronové sítě a jejího fungování

Neuronová síť je navržena tak, aby optimalizovala automatizaci domácnosti na základě senzorických dat z prostředí. Hlavním cílem je:

- Aktivace ventilátoru: Rozhodnout, kdy zapnout ventilátor a na jakou rychlost (3 úrovně).
- Ovládání žaluzií: Určit, kdy a jak moc pootočit žaluzie (3 polohy).

Síť přijímá údaje o aktuální teplotě a intenzitě světla a poskytuje rozhodnutí o rychlosti ventilátoru a natočení žaluzií.

Struktura Neuronové Sítě

Neuronová síť je vícevrstvý perceptron s následujícími vlastnostmi:

1. ***Vstupní vrstva přijímá dva vstupy:***
   - Teplota (°C)
   - Světlo (relativní intenzita 0-1)

2. ***Skryté vrstvy:***
   - 1.vrstva: 64 neuronů s aktivační funkcí ReLU* a Batch Normalization**.
   - 2.vrstva: 128 neuronů s aktivační funkcí ReLU, dropoutem (50 %)***, a Batch Normalization.
   - 3.vrstva: 256 neuronů s aktivační funkcí ReLU, dropoutem (50 %), a Batch Normalization.

***ReLU (Rectified Linear Unit)** je matematická funkce, která definuje výstup neuronu jako:

$$f(x) = max(0, x)$$

To znamená:

- Pokud je vstup $x > 0$, výstup je stejný jako vstup (x).
- Pokud je vstup $x \leq 0$, výstup je 0.

Proč se používá?

- Řešení problému mizení gradientu: Na rozdíl od dříve používaných aktivačních funkcí, jako je sigmoid nebo tanh, ReLU nezpůsobuje, že gradient (derivace) zmizí při zpětném šíření - backpropagation. To zlepšuje rychlost a efektivitu učení v hlubokých sítích.
- Jednoduchost a výpočetní efektivita: Výpočet je velmi jednoduchý
- Sparzita: ReLU vede k aktivaci pouze části neuronů v dané vrstvě, což snižuje výpočetní náročnost a přispívá ke generalizaci.

****Batch Normalization** je technika, která normalizuje výstupy neuronů v každé vrstvě sítě tak, aby měly nulovou střední hodnotu a jednotkovou směrodatnou odchylku v rámci jedné dávky.

$$\hat{x} = \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

kde:

x: vstupní hodnoty.
μ: střední hodnota vstupů v dávce.
σ^2: rozptyl vstupů v dávce.
ϵ: malá konstanta pro stabilizaci výpočtu (aby se předešlo dělení nulou).

Proč se používá?

- Rychlejší trénink: Normalizace zrychluje trénink neuronových sítí tím, že snižuje závislost na počátečních hodnotách vah.
- Stabilizace učení: BN pomáhá vyhnout se problémům s nestabilním tréninkem způsobeným velkými nebo malými vstupy.
- Regularizace: BN má lehce regularizační efekt, což znamená, že může snížit přeučení.

****Dropout** je technika regularizace, která během trénování náhodně „vypne“ určité procento neuronů ve vrstvě. V praxi to znamená, že během každé iterace tréninku se určitá část neuronů ignoruje. Při testování se dropout nepoužívá, ale váhy neuronů se škálují podle použitého dropout poměru.

Proč se používá?

- Prevence přeučení: Síť se učí „nezávislé“ reprezentace, protože jednotlivé neurony nemohou spoléhat na specifické jiné neurony.
- Robustnost modelu: Náhodné vypínání neuronů zajišťuje, že model lépe generalizuje na nová data.

Hodnota 50 % je obvykle volena ve skrytých vrstvách, protože poskytuje dobrý kompromis mezi regularizací a zachováním dostatečného množství informací pro učení.

3. ***Výstupní vrstva***
    - 6 neuronů pro regresní výstupy:
        - 3 hodnoty pro rychlosti ventilátoru (“0“, “1“, “2“)
        - 3 hodnoty pro natočení žaluzií (“0“, “1“, “2“)
    - Aktivační funkce: Identity pro regresní výstupy.
    - Ztrátová funkce: Mean Squared Error (MSE).

4. Optimalizace

Pro optimalizaci vah sítě byl použit adaptivní algoritmus Adam, který kombinuje výhody Adagrad a RMSProp. Regularizace je zajištěna:

- L2 regularizací s koeficientem 0.0001.
- Dropout vrstvami ve skrytých vrstvách, které pomáhají předcházet přeučení.

Softwarové řešení:

```
public class ModelUtils {
    private static final int NUM_INPUTS = 2; // Počet vstupů: Teplota a Světlo
    private static final int NUM_OUTPUTS
            = 6; // Počet výstupů: 3 rychlosti a 3 natočení
    public MultiLayerNetwork createModel() {
        return new MultiLayerNetwork(new NeuralNetConfiguration.Builder()
            .seed(123)
            .l2(0.0001) // L2 regularizace
            .updater(Updater.ADAM) // Použití Adam optimalizátoru
            .weightInit(WeightInit.XAVIER) // Inicializace vah pomocí Xavier metody
            .list()
            .layer(0, new DenseLayer.Builder().nIn(NUM_INPUTS).nOut(64)
                .activation(Activation.RELU) // Aktivace RELU
                .build())
            .layer(new BatchNormalization.Builder().build())
            .layer(1, new DenseLayer.Builder().nIn(64).nOut(128)
                .activation(Activation.RELU) // Aktivace RELU
                .dropOut(0.5)
                .build())
            .layer(new BatchNormalization.Builder().build())
            .layer(2, new DenseLayer.Builder().nIn(128).nOut(256)
                .activation(Activation.RELU) // Aktivace RELU
                .dropOut(0.5)
```

```
            .build())
        .layer(new BatchNormalization.Builder().build())
        .layer(3,new OutputLayer.Builder().nIn(256).nOut(NUM_OUTPUTS)
            .activation(Activation.IDENTITY) // pro regresní výstupy
            .lossFunction(LossFunctions.LossFunction.MSE) // Funkce ztráty
            .build())
        .build()); }
  public static void saveModel(MultiLayerNetwork model,String filePath)
throws IOException {
    model.save(new File(filePath)); }
  public static MultiLayerNetwork loadModel(String filePath) throws IOException {
            return MultiLayerNetwork.load(new File(filePath),false); }}
```

Trénování modelu

Trénování probíhá iterativně ve více epochách. Po každé epoše se vypočítá ztrátová funkce (loss), která vyjadřuje rozdíl mezi predikcemi modelu a skutečnými výstupy. Vývoj hodnoty ztráty je zaznamenán v konzoli, aby bylo možné sledovat průběh optimalizace.

Softwarové řešení:

```
private void trainModel(MultiLayerNetwork model, DataSet trainDataSet) {
    double bestLoss = Double.MAX_VALUE;
    int epochsWithoutImprovement = 0;
    for (int epoch = 0; epoch < NUM_EPOCHS; epoch++) {
      model.fit(trainDataSet);
      double currentLoss = model.score();
      System.out.println("Epoch: " + epoch + ", Loss: " + currentLoss);
      if (currentLoss < bestLoss) {
        bestLoss = currentLoss;
        epochsWithoutImprovement = 0;
      } else { epochsWithoutImprovement++; }
      if (epochsWithoutImprovement >= EARLY_STOPPING_PATIENCE) {
```

$System.out.println("Early\ stopping: Zastavení\ trénování\ po\ " + epoch + " epochách.");$

$break;\ \}\}\}$

Metoda Early Stopping

Aby se předešlo přetrénování modelu a šetřily se výpočetní zdroje, je implementována metoda early stopping. Pokud se ztrátová funkce nezlepší během definovaného počtu epoch (zde 10), trénink se automaticky zastaví.

$Epoch: 0, Loss: 0.34596888427553646$

$Epoch: 1, Loss: 0.32177607768868177$

$Epoch: 2, Loss: 0.3027749372860275$

$Epoch: 3, Loss: 0.28531628034075784$

...

$Epoch: 187, Loss: 0.07882641468595099$

$Epoch: 188, Loss: 0.07621106755814162$

$Epoch: 189, Loss: 0.07638210704616229$

$Epoch: 190, Loss: 0.07616921479585205$

Early stopping: Zastavení trénování po 190 epochách, když zlepšení přestalo být významné.

Vyhodnocení a uložení modelu

Po dokončení tréninku byl model vyhodnocen na testovacích datech. Předpovědi modelu byly porovnány s očekávanými výstupy za účelem stanovení přesnosti modelu a následně byl model uložen pro pozdější použití.

$Realita: Ventilátor = 1, Žaluzie = 1$

$Predikce: Ventilátor = 1, Žaluzie = 1$

...

$Predikce: Ventilátor = 1, Žaluzie = 1$

$Realita: Ventilátor = 1, Žaluzie = 1$

$Přesnost\ predikce\ rychlosti\ větráku: 1.0$

$Přesnost\ predikce\ pozice\ žaluzií: 0.95$

$Model\ byl$ úspěšně $uložen\ na: trained\_model.zip$

Takto máme uzavřen kompletní proces pro vývoj a trénink modelu neuronové sítě, který je uložen jako trained_model.zip pro následné použití na predikci nastavení ventilátoru a žaluzií.

Zpracování dat neuronovou sítí

Sběr dat z Micro:Bitu

Data jsou získávána pomocí Micro:Bitu, který měří teplotu a úroveň světla každých 10 sekund. Micro:bit odešle tato data ve formátu JSON přes sériový port. Využíváme standardní funkce pro čtení hodnot teploty a světla, přičemž tato data jsou odesílána v následujícím formátu:

$$\{\ "Teplota":\ <teplota>,\qquad "Svetlo":\ <světlo>\}$$

Tato data jsou pravidelně odesílána do hlavního systému, který je následně zpracuje.

Načtení modelu

Model neuronové sítě je načítán ze souboru trained_model.zip pomocí knihovny Deeplearning4j. Tento model je již předem vytrénován na historických datech a je připraven k použití pro predikci na základě aktuálně přijatých dat z Micro:Bitu.

Predikce na základě dat

Jakmile jsou data přijata, jsou použita k predikci výsledků. Případně obdržené hodnoty teploty a světla (vstupy pro neuronovou síť) jsou normalizovány a předány do modelu. Model na základě těchto vstupů provede predikci, která se skládá z následujících:

- Rychlost ventilátoru (0, 1, 2)
- Pozice žaluzií (0, 1, 2)

Výstupy modelu jsou poté zpracovány pomocí funkce softmax pro normalizaci hodnot do pravděpodobnostních tříd, což znamená, že model určí, jakou hodnotu pro každou kategorii by měl systém nastavit.

Odeslání predikce zpět do Micro:Bitu

Po provedení predikce je výsledek (rychlost ventilátoru a pozice žaluzií) odeslán zpět do Micro:Bitu. Tato data jsou zaslána v následujícím formátu přes sériový port:

$$<rychlost\ ventilátoru>,<pozice\ žaluzií>$$

$$1{,}2$$

Tato zpráva je přenesena zpět do Micro:Bitu, který následně ovládá příslušné zařízení. Pro ventilátor jsou použity funkce pro řízení rychlosti (pins.analogWritePin), a pro žaluzií jsou nastaveny různé úhly pomocí servomotoru (pins.servoWritePin).

Realizace na Micro:Bitu

Micro:bit přijímá zprávu, která obsahuje hodnoty pro rychlost ventilátoru a pozici žaluzií. Pokud jsou hodnoty správné, Micro:Bit použije funkce controlFan a controlBlinds, aby správně nastavila rychlost ventilátoru a pozici žaluzií. Pokud dojde k chybě v přijímaných datech (např. nesprávný formát), Micro:Bit zobrazí na displeji ikonu smutného obličeje.

Periodické zpracování

Celý proces (sběr dat, predikce, odeslání zpět) se opakuje každých 30 sekund. To je řízeno pomocí naplánovaného vykonávání úkolů v ScheduledExecutorService, který spouští funkci pro predikci a odesílání dat na pravidelných intervalech.

Softwarové řešení na straně Micro:Bitu:

```
serial.onDataReceived(serial.delimiters(Delimiters.NewLine), function () {
    try {
        let received
            = serial.readUntil(serial.delimiters(Delimiters.NewLine)).trim()
        let parts = received.split(",")
        if (parts.length == 2) {
            let fanSpeed = parseInt(parts[0])
            let blindPosition = parseInt(parts[1])
            controlFan(fanSpeed)
            controlBlinds(blindPosition)
            basic.showString(`F: ${fanSpeed} B: ${blindPosition}`)
        } else { basic.showIcon(IconNames.Sad) }
    } catch (error) {
        basic.showIcon(IconNames.Sad) }})
function controlBlinds(blindPosition: number) {
    if (blindPosition == 0) {
        pins.servoWritePin(AnalogPin.P1, 0)
    } else if (blindPosition == 1) {
        pins.servoWritePin(AnalogPin.P1, 45)
    } else if (blindPosition == 2) {
        pins.servoWritePin(AnalogPin.P1, 90) }}
```

```
function controlFan(fanSpeed: number) {
    if (fanSpeed == 0) {
        pins.analogWritePin(AnalogPin.P0, 0)
    } else if (fanSpeed == 1) {
        pins.analogWritePin(AnalogPin.P0, 512)
    } else if (fanSpeed == 2) {
        pins.analogWritePin(AnalogPin.P0, 1023)    }}
let temperature = 0
let Light = 0
// Hlavní smyčka pro měření a odesílání dat
basic.forever(function () {
    Light = input.lightLevel()
    temperature = input.temperature()
    let data = {
        "Teplota": temperature,
        "Svetlo": Light }
    // Odeslání jako JSON řetězec s delimitery
    serial.writeLine("$" + JSON.stringify(data) + "$")
    basic.pause(10000)})
```

Softwarové řešení na straně IntelliJ (zkráceno):

```
private void processSensorData() throws IOException {
        byte[] readBuffer = new byte[1024];
        int bytesRead = serialPort.readBytes(readBuffer, readBuffer.length);
        if (bytesRead > 0) {
            String receivedData = new String(readBuffer, 0, bytesRead).trim();
            int start = receivedData.indexOf("{");
            int end = receivedData.indexOf("}", start);
            if (start != -1 && end != -1) {
                String jsonString = receivedData.substring(start, end + 1);
```

```
        try {
            JsonObject jsonData
                = JsonParser.parseString(jsonString).getAsJsonObject();
            temp = jsonData.get("Teplota").getAsDouble();
            light = jsonData.get("Svetlo").getAsDouble();
            System.out.println("Přijatá data: Teplota: " + temp + ", Světlo: "
                + light);
            // Zobrazení přijatých dat
            System.out.println("Přijatá data: Teplota: " + temp + ", Světlo: "
                + light);
        } catch (Exception e) {
            System.err.println("Chyba při parsování JSON: "
                + e.getMessage()); } }}

private void predictAndDisplayCommands() {
    INDArray input = Nd4j.create(new double[][]{{temp, light}});
     System.out.println("Vstupní data pro model: Teplota: " + temp + ", Světlo: "
             + light);
    // Provádění predikce
    INDArray output = model.output(input);
    System.out.println("Výstup modelu (skóre): " + output);
    // Aplikace softmax funkce na výstupy pro ventilátor a žaluzie
    INDArray fanOutput
            = output.get(NDArrayIndex.all(), NDArrayIndex.interval(0, 3));
    INDArray blindsOutput
            = output.get(NDArrayIndex.all(), NDArrayIndex.interval(3, 6));
    INDArray fanProbabilities = softmax(fanOutput);
    INDArray blindsProbabilities = softmax(blindsOutput);
    // Extrakce predikovaných tříd
    int fanSpeedClass = fanProbabilities.argMax(1).getInt(0);
    int blindsPositionClass = blindsProbabilities.argMax(1).getInt(0);
```

$$// Výpis\ predikovaných\ tříd\ na\ obrazovku$$

$$System.out.println("Predikovaná\ rychlost\ ventilátoru: " + fanSpeedClass +$$

$$", Pozice\ žaluzií: " + blindsPositionClass);$$

$$// Sestavení\ zprávy\ ve\ formátu\ "rychlost\ ventilátoru, pozice\ žaluzií"$$

$$String\ message = fanSpeedClass + "," + blindsPositionClass + "\n";$$

$$serialPort.writeBytes(message.getBytes(), message.length());$$

$$System.out.println("Odeslaná\ zpráva: " + message); \}$$

Ukázka výpisu z konzole:

$Přijatá\ data: Teplota: 22.0, Světlo: 239.0$

$Přijatá\ data: Teplota: 22.0, Světlo: 239.0$

$Přijatá\ data: Teplota: 22.0, Světlo: 241.0$

$Přijatá\ data: Teplota: 22.0, Světlo: 241.0$

$Vstupní\ data\ pro\ model: Teplota: 22.0, Světlo: 241.0$

$Přijatá\ data: Teplota: 22.0, Světlo: 241.0$

$Přijatá\ data: Teplota: 22.0, Světlo: 241.0$

$Výstup\ modelu\ (skóre): [[\ \ 66.247,\ \ \ 1.678,\ \ 64.478, -11.187, -346.553, 488.5513]]$

$Predikovaná\ rychlost\ ventilátoru: 0, Pozice\ žaluzií: 2$

$Odeslaná\ zpráva: 0,2$

Světlo vykazuje vysokou hodnotu, protože jsem osvětloval senzor uměle pro dosažení predikce.

Hardwarové řešení

Hardwarové řešení pro tento projekt je postaveno na platformě BBC Micro:Bit V2, což je malý a výkonný mikropočítač určený pro vzdělávací účely a prototypování. Tento mikropočítač je vybaven širokým spektrem integrovaných senzorů a možností rozšíření, což ho činí ideálním pro tento typ projektu zaměřený na sledování environmentálních podmínek a automatizaci.

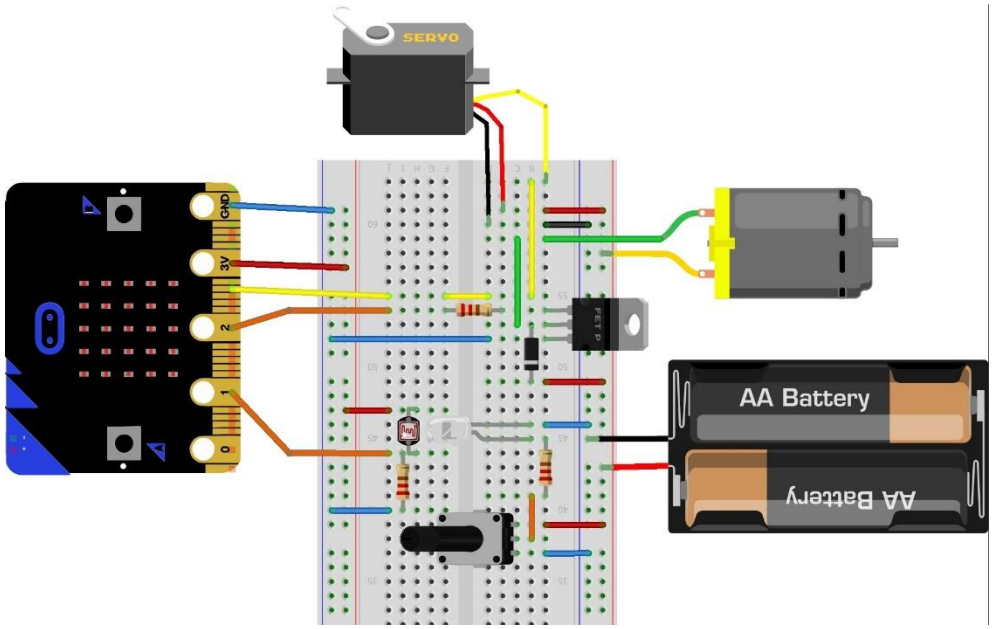
Propojení komponentů

- Servomotor a ventilátor jsou připojeny k výstupním pinům Micro:Bitu, a to prostřednictvím rozšiřujícího modulu, který poskytuje snadný přístup k těmto pinům.
- Teplotní a světelný senzor využívá integrované senzory Micro:Bitu pro měření aktuálních hodnot a jejich odesílání do softwarového systému.
- Komunikace mezi senzory a neuronovou sítí probíhá v reálném čase, kde hodnoty odesílané Micro:Bit připojeným USB kabelem do PC, které provádí vyhodnocení dat.

Výhody tohoto hardwarového řešení

- Modularita: Díky použití rozšiřujících modulů, kontaktních polí a propojovacích vodičů lze snadno připojit nebo změnit komponenty podle potřeby.
- Přenositelnost: Micro:bit je malý, lehký a energeticky efektivní, což z něj činí ideální volbu pro prototypování a testování v reálných podmínkách.
- Rozšiřitelnost: I když je výchozí hardware pro tento projekt dostatečný, systém je připraven pro rozšíření o další senzory nebo komponenty v případě potřeby.

Toto hardwarové řešení poskytuje flexibilní a efektivní základ pro vývoj inteligentního systému řízení teploty a osvětlení pomocí neuronové sítě.



Obrázek 4 - Schematické zapojení hardwaru

Závěr

Tento projekt se zaměřuje na vývoj systému pro automatizaci domácnosti pomocí mikrořadiče Micro:Bit a neuronové sítě. Cílem je využít data ze senzorů (teplota a světlo) pro predikci chování zařízení, jako je ventilátor a žaluzie, a automatické řízení jejich činnosti na základě těchto predikcí.

Data o teplotě a světle jsou získávána každých 10 sekund pomocí Micro:Bitu a odesílána ve formátu JSON. Tato data jsou následně použita pro trénování neuronové sítě, která se učí vztah mezi vstupními hodnotami a požadovanými výstupy, tj. rychlostí ventilátoru a polohou žaluzií.

Po natrénování modelu je tento uložen a používán pro predikce v reálném čase. Micro:bit každých 10 sekund přijímá aktuální hodnoty teploty a světla, které jsou odesílány do modelu pro predikci rychlosti ventilátoru a pozice žaluzií. Predikované hodnoty jsou následně každých 30 sekund vráceny zpět do Micro:Bitu, který je použije k ovládání zařízení.

Ventilátor je řízen třemi rychlostmi pomocí analogového výstupu, zatímco žaluzie jsou ovládány třemi pozicemi servomotorem. Micro:Bit pravidelně zobrazuje stav zařízení na displeji a zajišťuje jejich automatické řízení bez nutnosti manuálního zásahu.

Projekt kombinuje technologie jako Micro:Bit, model vícevrstvé neuronové sítě, sériovou komunikaci a různé softwarové nástroje pro trénování modelu, což umožňuje efektivní automatizaci domácnosti na základě reálných environmentálních podmínek.