

# Prezentácia - Deep learning v jazyku Rust

Elektronické spracovanie a prezentácia dokumentov

Filip Priečinský

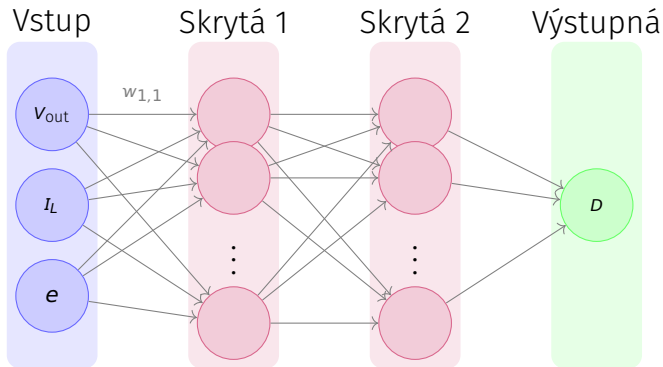
2025/2026      Žilinská univerzita v Žiline - FRI

# ČO JE NEURÓNOVÁ SIEŤ?

- Matematický model inšpirovaný ľudským mozgom
- Skladá sa z neurónov usporiadaných do vrstiev
- Vstupná vrstva → skryté vrstvy → výstupná vrstva
- Každé spojenie má váhu  $w$  — sieť sa učí úpravou týchto váh



# SCHÉMA NN



**Obr. 1:** Neurónová sieť o 3 vstupoch, dvoch skrytých vrstvách a jednej výstupnej vrstve

## AKO SA SIEŤ UČÍ?

1. **Forward pass** — dáta prechádzajú sieťou, dostaneme predikciu
2. **Loss** — zmeriame, ako veľmi sa sieť mýli
3. **Backpropagation** — spočítame gradienty (derivácie chyby podľa váh)
4. **SGD update** — upravíme váhy v smere najväčšieho poklesu chyby:

$$w \leftarrow w - \eta \cdot \frac{\partial L}{\partial w} \quad (1)$$

kde  $\eta$  je *learning rate*



## AKTIVAČNÉ FUNKCIE

Bez aktivačných funkcií by bola celá sieť len lineárna transformácia.

**ReLU** — najpoužívannejšia funkcia:

$$\text{ReLU}(x) = \max(0, x) \quad (2)$$

**Softmax** — na výstupe pre klasifikáciu:

$$\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}} \quad (3)$$

Výstupom je pravdepodobnostné rozdelenie cez triedy.



- Deep learning framework napísaný v Ruste
- Cieľ: byť pre Rust tým, čím je PyTorch pre Python
- Kľúčové vlastnosti:
  - Backend-agnostický dizajn
  - Typová bezpečnosť v compile-time
  - Automatická diferenciácia (autodiff)
  - Podpora pre CPU, CUDA, WebGPU, ...



Burn oddeluje logiku modelu od hardvéru:

`Tensor<B, 2>` — B je generický backend

- `NdArray` — Rust, CPU, žiadne závislosti
- `Wgpu` — GPU cez WebGPU (cross-platform)
- `LibTorch` — wrapper nad C++ LibTorch (CUDA)
- `Candle` — Hugging Face backend

Ten istý kód beží na CPU aj GPU — stačí zmeniť typ.



## BACKEND SA MENÍ JEDNÝM RIADKOM

```
// CPU, Rust, žiadne závislosti
```

```
type B = NdArray;
```

```
// GPU cez WebGPU, cross-platform
```

```
type B = Wgpu;
```

```
// NVIDIA GPU cez LibTorch
```

```
type B = LibTorch;
```

Zvyšok kódu sa nemení — model, tréning, inferencia, všetko zostáva rovnaké.





Tensor v Burn nesie informáciu o type počas kompilácie:

- `Tensor<B, 1>` – vektor
- `Tensor<B, 2>` – matica
- `Tensor<B, 3>` – 3D tenzor (napr. batch obrázkov)

Ak sa pokúsite násobiť maticu s vektorom nesprávnych rozmerov, kompilátor to zastaví ešte pred spustením.



```
#[derive(Module)]
```

```
pub struct MnistModel<B: Backend> {  
    linear1: Linear<B>,  
    linear2: Linear<B>,  
    activation: Relu,  
}
```

] Derive makro **Module** automaticky vygeneruje:

- Zber všetkých parametrov na tréning
- Serializáciu a deserializáciu modelu
- Presun medzi zariadeniami (CPU ↔ GPU)



## FORWARD PASS

```
fn forward(&self, x: Tensor<B, 2>) -> Tensor<B, 2>
    let x = self.linear1.forward(x);
    let x = self.activation.forward(x);
    self.linear2.forward(x)
}
```



## AKO FUNGUJE AUTODIFF V BURN

Burn zabalí ľubovoľný backend do autodiff vrstvy:

1. Počas forward passu sa buduje výpočtový graf
2. Každá operácia si pamätá, odkiaľ prišli dáta
3. `loss.backward()` prejde graf nazad.
4. Výsledok: gradient pre každý parameter

Typ rozlišuje fázy — **Autodiff<Wgpu>** na tréning a **Wgpu** na inferenciu.





Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani a Jonathan Taylor. *An Introduction to Statistical Learning with Applications in Python.*, 2023. Kap. 6.2.2, s. 241–250.

