Deep Learning

Deep learning is a subfield of machine learning that is concerned with algorithms inspired by the structure and function of the brain called artificial **neural networks**.

Recap

- Linear Algebra;
- Probability and Statistics;
- Optimization;
- Machine Learning.

Syllabus

1. Linear Models - linear regression, perceptron, logistic regression, regularization;
2. Neural Networks;
3. Representation Learning;
4. Convolutional Neural Networks;
5. Recurrent Neural Networks
6. Sequence-to-Sequence Models - introduction to Attention Mechanisms;
7. Attention Mechanisms and Transformers;
8. Self-Supervised Learning and Large Pretrained Models;
9. Deep Generative Models;
10. Interpretability and Fairness.

The following is a summary of the course's contents.

Linear Models

Linear models are a class of **regression** and **classification** models in which the prediction is a linear function of the input variables.

Linear Regression: $y = w^T x + b$

- w is a d-dimensional vector of **weights**;
- b is a **bias** - usually included in w as a constant feature $x_0 = 1$;
- Given **training data** $D = \{(x_n, y_n)\}_{n=1}^N$, we want to find the **best** w and b, so we use want to fit the model, i.e. find the best w and b, **minimizing the loss function** - usually, the **square loss**: $L(w,b) = \sum_{n=1}^N (y_n - (w^T x_n + b))^2$;
- **Closed-form solution**: $w = (X^T X)^{-1} X^T y$, where $X = \begin{bmatrix} x_1^T \\ x_2^T \\ \vdots \\ x_N^T \end{bmatrix}$ and $y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$;
- **Regularization** is a technique used to **reduce overfitting** by **constraining** the **weights** of the model;
  - **Ridge regression** is a **linear regression** model with **regularization**;
    * $L(w,b) = \sum_{n=1}^N (y_n - (w^T x_n + b))^2 + \lambda ||w||_2^2$;
    * **Closed-form solution**: $w_{\hat{ridge}} = (X^T X + \lambda I)^{-1} X^T y$;

Maximum A Posteriori (MAP) Estimation

- MAP estimation is a **Bayesian** approach to **estimation**, used to **estimate** the **parameters** of a **distribution** - used in **regularization**;
- **Bayes' rule**: $P(\theta|x) = \frac{P(x|\theta)P(\theta)}{P(x)}$;
  - $\hat{w}_{MAP} = argmax_w p(w|y) = argmax_w p(y|w)p(w)/p(y) = argmin_w \lambda ||w||_2^2 + \sum_{n=1}^N (y_n - w^T \phi(x_n))^2$;
  - The **prior** $||w||_2^2$ is the **regularization term** - regularizer;
  - The **likelihood** $\sum_{n=1}^N (y_n - w^T \phi(x_n))^2$ is the **loss function**;
  - The **regularization constant** is $\lambda = \frac{\sigma^2}{\tau^2}$;

- **Maximum likelihood estimation (MLE)** is a **Bayesian** approach to **estimation**;
  - $\hat{w}_{MLE} = argmax_w p(y|w) = argmin_w \sum_{n=1}^{N}(y_n - w^T\phi(x_n))^2$;
  - The **likelihood** $\sum_{n=1}^{N}(y_n - w^T\phi(x_n))^2$ is the **loss function**;
  - **MLE** is a **special case** of **MAP** estimation, where $\lambda = 0$;

Binary Classification -> Perceptron: $y = sign(w^Tx + b)$

- **Perceptron** is a **linear model** for **binary classification**;
- Usually, the **bias** b is included in w as a constant feature $x_0 = 1$, and the w vector is **augmented** with b;
- The x vector can be represented as $\phi(x)$, where ϕ is a **feature map**;
- Algorithm:

```
1. Initialize $w$ to zero: $w_0 = 0$;
2. While not converged, for each $(x_n, y_n)$ in $D$:
   (a) Predict: $\hat{y} = sign(w^Tx_n)$;
   (b) If $\hat{y} \neq y_n$:
       i. Update: $w_{t+1} = w_t + y_nx_n$;
       ii. Go to step 2;
```

- **Perceptron convergence theorem**: if the **training data** is **linearly separable**, the **perceptron algorithm** will **converge** in a **finite number of steps**:
- It **cannot** be used for **non-linearly separable data** - XOR problem;

Binary Classification -> Logistic Regression: $y = \sigma(w^Tx + b)$

- **Logistic regression** is a **linear model** for **binary classification** - differs from perceptron, because it uses a **sigmoid function (continuous)** instead of a **sign function (discrete)**;
- $P_W(y|x) = \frac{exp(w_y^Tx+b_y)}{\sum_{y'\in Y} exp(w_{y'}^Tx+b_{y'})}$, where $W = \{w_y, b_y\}_{y\in Y}$;
  - Set weights to maximize conditional log-likelihood: $L(W) = \sum_{n=1}^{N} logP_W(y_n|x_n)$;
- No closed-form solution, so we use **stochastic gradient descent**;
- Update rule: $w_{y_n}^{k+1} = w_{y_n}^k + \eta x_n - \eta \sum_{y'\in Y} P_{W^k}(y'|x_n)x_n$;

Multi-class Classification

- **Multi-class classification** is a **classification** task with **more than two classes**, but there are several strategies to **reduce to binary classification**;
- Parametrized by a **weight matrix** $W \in \mathbb{R}^{d \times |Y|}$ and a **bias vector** $b \in \mathbb{R}^{|Y|}$;
- $\hat{y} = argmax_{y \in Y}(W\phi(x) + b)$;
- Multi-class perceptron algorithm:

```
1. Initialize $W$ to zero: $W_0 = 0$;
2. While not converged, for each $(x_n, y_n)$ in $D$:
   (a) Predict: $\hat{y} = argmax_{y \in Y}(W\phi(x_n))$;
   (b) If $\hat{y_n} \neq y_n$:
       i. Update: $W_{y_n}^{k+1} = W_{y_n}^{k} + \phi(x_n)$
       ii. Update: $W_{\hat{y_n}}^{k+1} = W_{\hat{y_n}}^{k} - \phi(x_n)$;
       iii. Go to step 2;
```

Neural Networks

- **Neural networks** are a class of **non-linear models** that are **inspired by the brain**;
- Consist of **neurons** that are **connected** to each other;
- **Pre-activation**: $z(x) = w^T x + b = \sum_{i=1}^{d} w_i x_i + b$;
- **Activation**: $h(x) = g(z(x))$; g can be:
  - **Linear**: $g(z) = z$ - linear regression;
  - **Sigmoid**: $g(z) = \frac{1}{1+e^{-z}}$ - logistic regression;
  - **Rectified Linear Unit (ReLU)**: $g(z) = max(0, z)$;
  - **Hyperbolic Tangent (tanh)**: $g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$;
  - **Softmax**: $g(z) = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}$;
  - Others;
- A **feed-forward neural network** is a **neural network** where the **neurons** are **organized in layers** - there are **hidden layers** between the **input layer** and the **output layer**;
  - **Input layer**: x - vector of features;

  - **Hidden layers**: $h^{(1)}, h^{(2)}, \ldots, h^{(L)}$;
  - **Output layer**: y - vector of predictions;
  - **Weights**: $W^{(1)}, W^{(2)}, \ldots, W^{(L)}$ - each weight matrix is between two layers;
  - **Biases**: $b^{(1)}, b^{(2)}, \ldots, b^{(L)}$ - each bias vector is between two layers;
  - **Hidden layer pre-activation**: $z^{(l)}(x) = W^{(l)} h^{(l-1)}(x) + b^{(l)}$;
  - **Hidden layer activation**: $h^{(l)}(x) = g(z^{(l)}(x))$;
  - **Output layer activation**: $f(x) = h^{(L)}(x)$;

- **Universal approximation theorem**: a **feed-forward neural network** with a **single hidden layer** can **approximate any function** - given enough **neurons**;

- Training consists of **finding the best parameters** θ - weights and biases, that **minimize the loss function**: $L(\theta) := \lambda\omega(\theta) + \frac{1}{N}\sum_{n=1}^{N} l(f(x_n;\theta), y_n)$;
  - λ is the **regularization constant**;
  - $\omega(\theta)$ is the **regularization term**;
  - $l(f(x_n;\theta), y_n)$ is the **loss function**;
  - We use **stochastic gradient descent** to **minimize the loss function**: $\nabla_\theta L(\theta) = \lambda\nabla_\theta\omega(\theta) + \frac{1}{N}\sum_{n=1}^{N} \nabla_\theta l(f(x_n;\theta), y_n)$;
  - **Backpropagation** is a technique used to **compute the gradients** of the **loss function** with respect to the **parameters - chain rule**.

Representation Learning

- Neural networks can be used to learn **representations** of the data;
- Distributed representations are exponentially more compact than one-hot representations;
- Deeper networks exhibit hierarchical compositionality: **higher layers** represent **higher-level concepts**;
- **Auto-encoders** are a class of **neural networks** that are used to **learn representations** of the data;
- **Word embeddings** are a type of **representation** used to **represent words** as **vectors**.

Convolutional Neural Networks

- **Convolutional neural networks (CNNs)** are a class of **deep neural networks** that are **specialized** for **processing data** that has a **grid-like topology**, such as **images**.
- **Convolution layers** are alternated with **pooling layers** - **convolution** is a **linear operation** that **preserves the grid-like topology** of the input;
- **Activation maps** are the **output** of a **convolutional layer**;
- **Stride** is the **step size** of the **convolution** - S;
- **N of Channels** - K;
- **N of Filters** - M;
- **Padding** is the **number of zeros** added to the **input** - P;
  - A common padding size is $P = \frac{F-1}{2}$, which preserves the input size;
- Given an $N \times N \times D$ input, a $F \times F \times D$ filter, a stride S and padding P, the output will be a $M \times M \times K$ activation map, where $M = \frac{N-F+2P}{S} + 1$;
- **Number of units** in a **convolutional layer**: $M^2 \times K$;
- **Number of trainable parameters** in a **convolutional layer**: $M \times ((N^2 \times K) + bias)$;
- Properties of CNNs:
  - **Invariance** - the output is **invariant** to **small translations** of the input;
  - **Locality** - the output is **only affected** by a **small region** of the input;
  - **Sparse interactions** - each output value is the result of a **small number of interactions** with the input;
  - **Parameter sharing** - the **same parameters** are used for **different parts** of the input.

Recurrent Neural Networks

- RNNs allow to take advantage of **sequential data** - words in text, DNA sequences, sound waves, etc;
  - $h_t = g(Vx_t + Uh_{t-1} + c)$ - h_t is the **hidden state** at time t;
  - $\hat{y}_t = Wh_t + b$ - $\hat{y}_t$ is the **output** at time t;

- Used to generate, tag and classify sequences, and are trained using **back-propagation through time**;
  - Parameters V, U, W, c and b are **shared** across **time steps - parameter sharing**;
- Applications:
  - **Sequence generation** - generate a sequence of words - **auto-regressive models**;
  - **Sequence tagging -** assign a label to each element in a sequence;
  - **Pooled classification -** classify a sequence as a whole;
- Standard RNNs suffer from vanishing and exploding gradients - alternative parameterizations like **LSTMs** and **GRUs** are used to avoid this problem;
- **Gated Recurrent Units (GRUs)** are a type of **recurrent neural network** that are **simpler** than **LSTMs** and **perform better** than **standard RNNs -** idea is to create some **shortcuts** in the **standard RNN**;
  - $u_t = \sigma(V_u x_t + U_u h_{t-1} + b_u)$ - **update gate**;
  - $r_t = \sigma(V_r x_t + U_r h_{t-1} + b_r)$ - **reset gate**;
  - $\tilde{h}_t = tanh(vx_t + U(r_t \odot h_{t-1}) + b)$ - **candidate hidden state**;
  - $h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t$ - **hidden state**;
- **Long Short-Term Memory (LSTM)** is a type of **recurrent neural network** that are **more complex** than **GRUs** and **perform better** than **standard RNNs** - idea is to use **memory cells** c_t to **store information**;
  - $i_t = \sigma(V_i x_t + U_i h_{t-1} + b_i)$ - **input gate**;
  - $f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f)$ - **forget gate**;
  - $o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o)$ - **output gate**;
  - $\tilde{c}_t = tanh(W_c x_t + U_c h_{t-1} + b)$ - **candidate cell state**;
  - $c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$ - **cell state**;
  - $h_t = o_t \odot tanh(c_t)$ - **hidden state**.

Sequence-to-Sequence Models

- **Sequence-to-sequence models** are a class of **neural networks** that are used to **map sequences** to **sequences - encoder-decoder** architecture;
  - Used for **machine translation**, **speech recognition**, **image captioning**, etc;

- A **Neural Machine Translation (NMT)** system is a **sequence-to-sequence model** that is used to **translate** a **sequence** in one **language** to a **sequence** in another **language - encoder-decoder** architecture;
  - **Encoder** RNN encodes source sentence into a **vector state -** $h_t = f(x_t, h_{t-1})$;
  - **Decoder** RNN decodes the **vector state** into a **target sentence -** $y_t = g(y_{t-1}, s_t)$;
- Representing the **input sequence** as a **single vector** is a **bottleneck - attention mechanisms** are used to **improve performance -** focus on different parts of the input.

Attention Mechanisms and Transformers

- Encoders/decoders can be RNNs, CNNs or **self-attention layers**;
- **Self-attention** is a **linear operation** that **maps** a **sequence** of **vectors** to a **sequence** of **vectors - encoder-decoder** architecture;
  - **Query** vector q_t;
  - **Key** vectors $k_1, k_2, \dots, k_n$;
  - **Value** vectors $v_1, v_2, \dots, v_n$;
  - **Attention weights** $\alpha_{t,i} = \frac{exp(q_t^T k_i)}{\sum_{j=1}^{n} exp(q_t^T k_j)}$;
  - **Output** vector $o_t = \sum_{i=1}^{n} \alpha_{t,i} v_i$;
- **Transformers**: **encoder-decoder** architecture with **self-attention** layers instead of **RNNs**;
  - **Encoder**: **self-attention** layers;
  - **Decoder**: **self-attention** layers + **encoder-decoder attention** layers;
- **Multi-head attention** is a **self-attention** layer with **multiple heads - parallel** self-attention layers;
  - **Query** vectors q_t;
  - **Key** vectors $k_1, k_2, \dots, k_n$;
  - **Value** vectors $v_1, v_2, \dots, v_n$;
  - **Attention weights** $\alpha_{t,i} = \frac{exp(q_t^T k_i)}{\sum_{j=1}^{n} exp(q_t^T k_j)}$;
  - **Output** vector $o_t = \sum_{i=1}^{n} \alpha_{t,i} v_i$.

Self-Supervised Learning and Large Pretrained Models

- Pretraining large models and fine-tuning them to a specific task is a common practice in deep learning:
  - **Pretraining** is a technique used to **initialize** the **parameters** of a **neural network - self-supervised learning**;
  - **Fine-tuning** is a technique used to **adapt** the **parameters** of a **neural network** to a **specific task**;
- Models: ELMo, BERT, GPT, etc;
- **Adapters** and **prompting** are other strategies more parameter-efficient than fine-tuning;
- Current models exhibit **few-shot learning** capabilities - can be trained with **few examples**.