[page=-]assignment.pdf

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Bachelor's thesis

Gesture detector with Leap Motion sensor

Anh Tran Viet

Department of Theoretical Computer Science
Supervisor: Tomáš Nováček

March 31, 2024

Acknowledgements

THANKS (remove entirely in case you do not with to thank anyone)

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as a school work under the provisions of Article 60 (1) of the Act.

In Prague on March 31, 2024 . . . . . .

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Tran, Anh Viet. *Gesture detector with Leap Motion sensor.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2024.

Abstrakt

V několika větách shrňte obsah a přínos této práce v českém jazyce.

Klíčová slova Replace with comma-separated list of keywords in Czech.

Abstract

Summarize the contents and contribution of your work in a few sentences in English language.

Keywords Replace with comma-separated list of keywords in English.

Contents

List of Figures

Introduction

Computer graphics and 3D modeling is a fundamental field in our constantly increasing digital world. Its application in engineering, science, art, product visualization, and most notably, entertainment has pushed this field with a constant demand for better visual quality and performance optimization.

3D models create the very core of any 3D visualization, bringing objects to computer-generated 3D space. The 3D model defines the shape of an object and can be further improved to closely represent real-world objects by applying textures, lighting, and many more techniques. Models are typically created in 3D modeling software by an artist or by 3D scanning. 3D scanning can often bring too many details in terms of polygon numbers. A shape that could be perfectly recognizable by ten polygons would now be described by thousands. The side effect of things is big hardware requirements.

Lowering the number of polygons is a usual practice done either manually or by many developed tools over the years. These tools do not consider any edge flow, a critical feature particularly important for character modeling and animation. Edge flow allows smooth deformation. In terms of animation, we can look at muscle movement, optimal edge avoids wrinkled defects on the character model, and the skin would stretch and contract in a way as we are used to seeing in the real world.

This thesis aims to introduce the final step in polygon reduction workflow when using tools. We present a neural network taking a 3D model and outputting an improved model in terms of edge flow quality, approximating similar polygon quantity.

In Chapter Neural Networks, we explore basics of neural networks, introduce its terminology and several commonly used network architectures.

In Chapter **??**, we describe used methods and key implementation points of our work.

In Chapter **??**

In **??**, we evaluate the results of the work and suggest possible improvements for future works.

Introduction

Computer graphics and 3D modeling is a fundamental field in our constantly increasing digital world. Its application in engineering, science, art, product visualization, and most notably, entertainment has pushed this field with a constant demand for better visual quality and performance optimization.

3D models create the very core of any 3D visualization, bringing objects to computer-generated 3D space. The 3D model defines the shape of an object and can be further improved to closely represent real-world objects by applying textures, lighting, and many more techniques. Models are typically created in 3D modeling software by an artist or by 3D scanning. 3D scanning can often bring too many details in terms of polygon numbers. A shape that could be perfectly recognizable by ten polygons would now be described by thousands. The side effect of things is big hardware requirements.

Lowering the number of polygons is a usual practice done either manually or by many developed tools over the years. These tools do not consider any edge flow, a critical feature particularly important for character modeling and animation. Edge flow allows smooth deformation. In terms of animation, we can look at muscle movement, optimal edge avoids wrinkled defects on the character model, and the skin would stretch and contract in a way as we are used to seeing in the real world.

This thesis aims to introduce the final step in polygon reduction workflow when using tools. We present a neural network taking a 3D model and outputting an improved model in terms of edge flow quality, approximating similar polygon quantity.

In Chapter Neural Networks, we explore basics of neural networks, introduce its terminology and several commonly used network architectures.

In Chapter **??**, we describe used methods and key implementation points of our work.

In Chapter **??**

In **??**, we evaluate the results of the work and suggest possible improvements for future works.

Chapter 1

Neural Networks

An artificial neural network (ANN), first introduced by Warren McCulloch and Walter Pitts in "A logical calculus of the ideas immanent in nervous activity" published in 1943 [**?**], is a mathematical model based on biological neural networks. It carries the ability to learn and correct errors from previous experience [**?**], [**?**].

The ANN has gained popularity in recent years with still increasing advancements in technology and availability of training data. ANN now becomes a default solutions for complex tasks previously thought to be unsolvable by computers [**?**].

This chapter will briefly introduce different types of neural units and their activation functions, along with some commonly used network architectures.

1.1 Artificial Neuron

Artificial neurons are units of ANN, mimicking behavior of biological neurons. Just as biological neurons, it can receive as well as pass information to other neurons.

1.1.1 Perceptron

Perceptron, developed by Frank Rosenblatt in 1958 [**?**], is the simplest class of artificial neurons.

Perceptron takes several binary inputs in form of a vector $\vec{x} = (x_1, x_2, ..., x_n)$, and outputs a single binary number. Perceptron uses real numbers called *weights*, assigned to each edge, vector $\vec{w} = (w_1, w_2, ..., w_n)$, to express the importance of respected input edges,

A *step function* calculates the perceptron's output. The function output is either 0 or 1 determined by whether its weighted sum $\alpha = \sum_i x_i w_i$ is less

or greater than its *threshold* value, a real number, usually represented as an incoming edge with a negative weight -1 [**?**].

$$output = \begin{cases} 1, & \text{if } \alpha \geq threshold \\ 0, & \text{if } \alpha < threshold \end{cases} \tag{1.1}$$

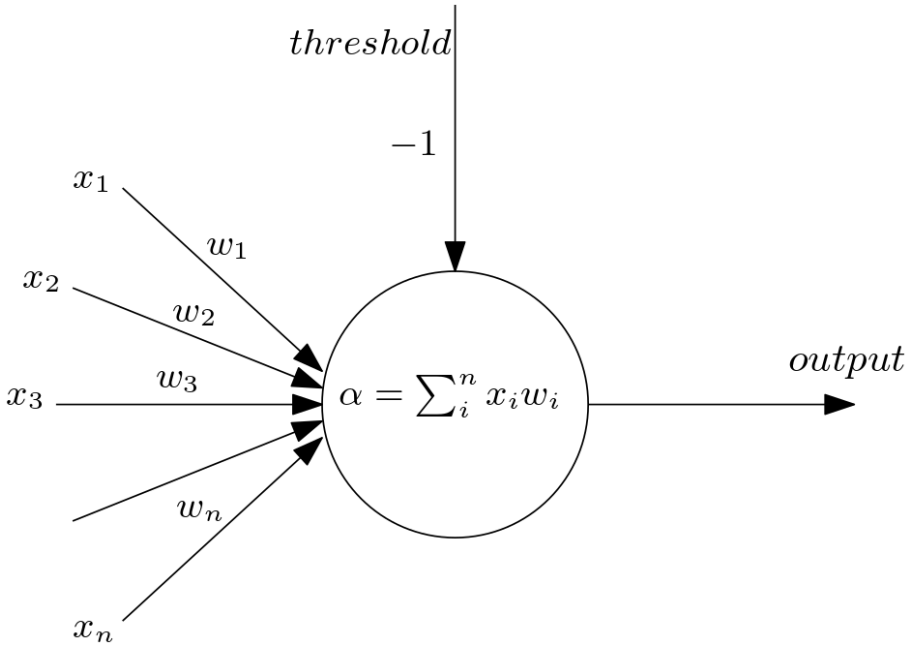


Figure 1.1: Perceptron [**?**]

1.1.2 Sigmoid Neuron

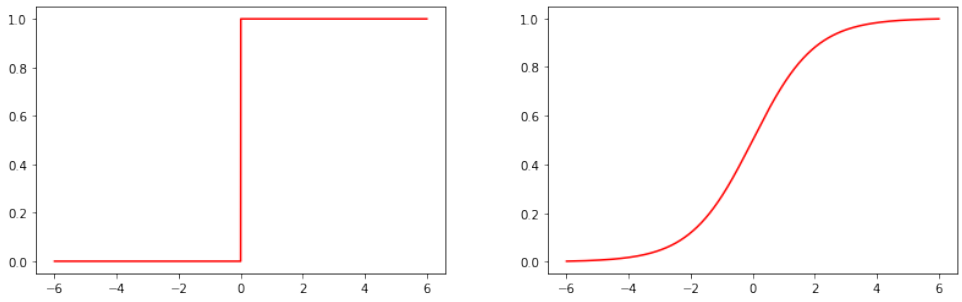
Sigmoid neuron, similarly to perceptron, has vector inputs $\vec{x}$ and weights. The difference is in the output value and its calculation. Instead of perceptron's binary output 0 or 1, a sigmoid neuron outputs a real number between 0 and 1 using a *sigmoid function* [**?**], [**?**], [**?**].

$$\sigma(\alpha) = \frac{1}{1 + e^{-\alpha}} \tag{1.2}$$

As shown in Figure 1.2, the sigmoid function (1.2b) is a smoothed-out step function (1.2a).

1.1.3 Activation Function

An artificial neuron's activation function defines neuron's output value for given inputs, commonly being $f : \mathbb{R} \to \mathbb{R}$ [**?**]. An important trait of many activation functions is their differentiability, allowing them to be used for



(a) Step function

(b) Sigmoid function

Figure 1.2: Comparison between step function and sigmoid function

Backpropagation, ANN training algorithm. The activation function needs to have a derivative that does not saturate when headed towards 0 or explode when headed towards inf [**?**].

For such reasons, step function or any linear function are unsuitable for ANN.

1.1.3.1 Sigmoid Function

The sigmoid function is often used in ANN as an alternative to the step function. A popular choice of the sigmoid function is a *logistic sigmoid*, output value ranging between 0 and 1.

$$\sigma(\alpha) = \frac{1}{1 + e^{-\alpha}} = \frac{e^x}{1 + e^x} \tag{1.3}$$

One of the reasons for its popularity is the simplicity of its derivative calculation:

$$\frac{d}{dx}\sigma(\alpha) = \frac{e^x}{(1 + e^x)^2} = \sigma(x)(1 - \sigma(x)) \tag{1.4}$$

Its disadvantages is the *vanishing gradient*. A problem where if given a very high or very low input values, the prediction would stay almost the same. Possibly resulting in training complications or performance issues [**?**], [**?**].

1.1.3.2 Hyperbolic Tangent

Hyperbolic tangent is similar to logistic sigmoid function with a key difference in its output, ranging between -1 and 1.

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{1.5}$$

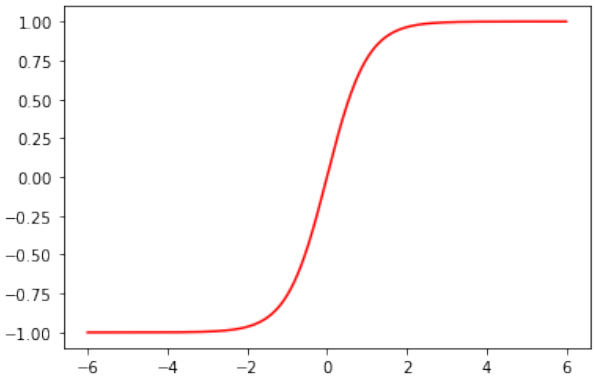


Figure 1.3: Hyperbolic tangent [?]

It shares the sigmoid's simple calculation of its derivative.

$$\frac{d}{dx}\tanh(x) = 1 - \frac{(e^x - e^{-x})^2}{(e^x + e^{-x})^2} = 1 - \tanh^2(x) \tag{1.6}$$

By being only moved and scaled version of the sigmoid function, hyperbolic tangent shares not only sigmoid's advantages but also its disadvantages [?], [?].

1.1.3.3 Rectified Linear Unit

The output of the Rectified Linear Unit (ReLU) is defined as:

$$f(x) = max(0, x) \begin{cases} x, & \text{if } x \geq 0 \\ 0, & \text{if } x < 0 \end{cases} \tag{1.7}$$

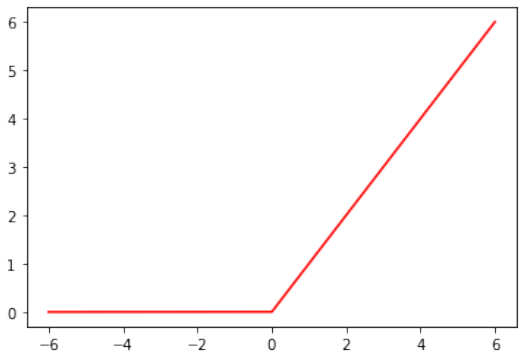


Figure 1.4: Rectified Linear Unit [?]

ReLU's popularity is mainly due to its computational efficiency [?]. Its disadvantages begin to show themselves once inputs approach zero or to a

negative number. Causing the so-called dying ReLu issue, where the network is unable to learn anymore. There are many variations of ReLu to this date, e.g., Leaky ReLU, Parametric ReLU, ELU, ...

1.1.3.4 Softmax

Softmax separates itself from all the previously mentioned functions by its ability to handle multiple input values in the form of a vector $\vec{x} = (x_1, x_2, ..., x_n)$ and output for each x_i defined as:

$$\sigma(x_i) = \frac{e_i^x}{\sum_{j=1}^{n} e_j^x} \tag{1.8}$$

Output is normalized probability distribution, ensuring $\sum_i \sigma(x_i) = 1$ [**?**]. It is being used as the last activation function of ANN, normalizing the network's output into n probability groups.

1.2 Types of Neural Networks

To this day, there are many types and variations of ANN, each with its structure and use cases. Here we will briefly introduce the most common ones, such as feed-forward networks, convolutional neural networks, or recurrent neural networks.

1.2.1 Feed-forward Networks

Feed-forward network (FFN), the first invented ANN and the simplest variation of an ANN. Its name comes from the way how information flows through the network. The data flows in one direction, oriented from the *input layer* to the *output layer*, without cycles. The input layer takes input data, vector $\vec{x}$, producing $\hat{y}$ at the output layer [**?**].

FFN contains several hidden layers of various widths but it can also have no hidden layers at all. By having no back-loops, FFN generally minimizes error, computed by *cost function*, in its prediction by using the *backpropagation* algorithm to update its weight values [**?**], [**?**].

1.2.1.1 Cost Function

Cost function $C(\vec{w})$ is used in ANN's training process. It takes all weights and biases of an ANN as its input, in the form of a vector $\vec{w}$ and calculates a single real number expressing ANN's error in prediction [**?**]. Higher number expressing poor prediction and as the number gets lower the ANN's output gets closer to the correct result. The main goal of training is to minimize the cost function.

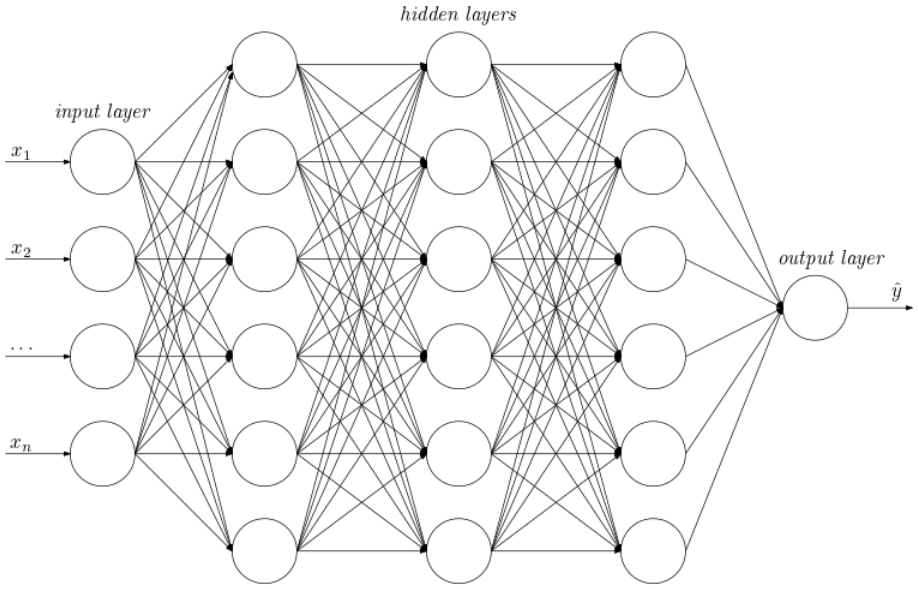


Figure 1.5: Fully connected Feed-forward Neural Network [**?**]

1.2.1.2 Backpropagation

Backpropagation, short of backward propagation of errors, is a widely used algorithm in training FFN using *gradient descent* to find a local minimum of a cost function and update ANN's weights [**?**].

The gradient of a multi-variable function provides us with the direction of the gradient ascent, where we should step to rapidly increase the output and find the local maximum. Conversely, the negative of the gradient points towards the local minimum.'

It is common practice to split training samples into small *batches* of size n. For each sample in the batch, we will calculate a gradient descent and use their average gradient descent to update the network's weights. This average gradient descent indicates the adjustments that need to be made to the weights so that the artificial neural network (ANN) moves closer to the correct results [**?**].

$$-\gamma \nabla C(\vec{w}_i) + \vec{w}_i \rightarrow \vec{w}_{i+1} \tag{1.9}$$

$\vec{w}_i$ is weights of the network at the current state (batch), $\vec{w_{i+1}}$ is updated weights, γ is the learning rate and $-\nabla C(\vec{w}_i)$ is the gradient descent.

1.2.2 Convolutional Neural Networks

The primary objective of a Convolutional Neural Network (CNN) is to enable a computer to identify images and objects, making it ideal for image classification and object recognition tasks.

CNNs are based on the biological processes in the human brain and its connectivity patterns resemble those of the human visual cortex. However, images are perceived differently by the human brain and a computer, with the latter interpreting images as arrays of numbers. As a result, CNNs are designed to work with two-dimensional image arrays, although they can also work with one-dimensional or three-dimensional arrays [**?**].

CNN is a variation of FNN [**?**]. It typically consists of an input layer, followed by multiple hidden layers, including several *convolutional layers* and *pooling layers*, and concluding with an output layer.

1.2.2.1 Convolutional Layer

The convolutional layer's goal is to extract key features from the input image by passing a matrix known as a *kernel* over the input image abstracted into a matrix [**?**].

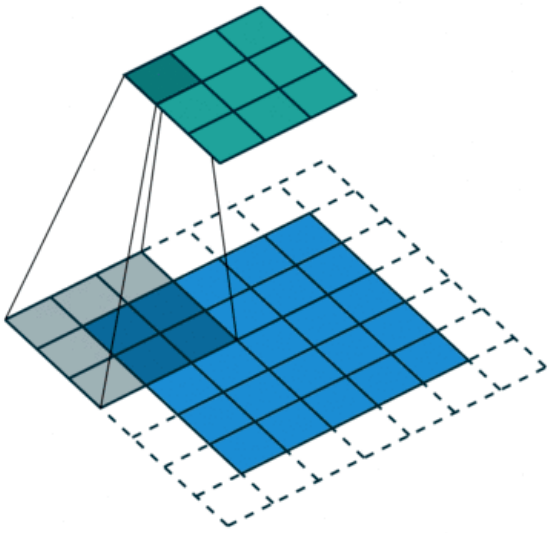


Figure 1.6: Convolution of an 5x5x1 image with 3x3x1 kernel [**?**]

The outcome of a convolution operation can be either reduced or increased in size. If the size is reduced, it is referred to as *valid padding*. For example, a convolution operation on an 8x8 input image would result in a 6x6 convoluted feature. On the other hand, if the size remains the same or is increased, it is referred to as *same padding* [**?**].

1.2.2.2 Pooling Layer

Like the convolutional layer, the pooling layer reduces the size of the convolved feature to reduce computational power needed for data processing. It also

extracts dominant features that are invariant to rotation and position, making it beneficial for training the model effectively [**?**].

There are two types of pooling: max pooling and average pooling. Max pooling returns the maximum value from the portion of the image covered by the kernel, acting as a noise suppressant by removing noisy activations and performing de-noising and dimensionality reduction. Average pooling returns the average of all values in the same portion, reducing dimensions as a noise suppression mechanism. It is worth noting that max pooling performs better [**?**].

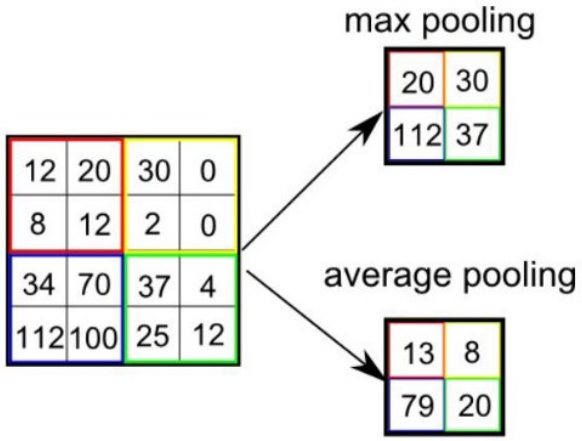


Figure 1.7: Types of pooling [**?**]

1.2.3 Graph Neural Networks

Graphs are commonly used to describe and analyze entities with relations or interactions. Still, today's deep learning modern toolbox is specialized for simple data types, e.g., grids for images, sequences for text or speech. These data structures have spatial locality, the grid size or sequence length is fixed and can be resized. We can also determine the starting position and ending position. Graph problems are, on the other hand, much more challenging to process as they have arbitrary size and complex topological structure (i.e., no spatial locality like grids). Graphs also have no fixed node ordering or reference point compared to grids or sequences. For such Franco Scarselli et al. introduced graph neural network (GNN) [**?**].

GNN is designed similarly to CNN. As previously noted, CNN operates on images. Given an image, a rectangular grid, the convolutional layer takes a subpart of the image, applies a function to it, and produces a new part, a new pixel. This is iterated for the whole image. What actually happened is the new pixel resulted in aggregated information from neighbors and itself. This cannot be easily applied to graphs as they have no spatial locality and no fixed

node ordering. As implied, a GNN design stands on passing and aggregating information from neighbors.

1.2.3.1 Node embedding

Graphs require a concept called node embedding. The general idea is to map nodes to a lower dimensional embedding space, where similar nodes in the embedding space approximate similarity in the graph network. For example, we can map a 3D vector to a 2D vector. Node embedding is useful for learning node representations and similarities and can be trained on graphs with arbitrary structures.

Nodes have embeddings at each layer. Taken node v, its layer-0 embedding would be its feature vector x_v. If we want layer-1 embedding, we will explore node v's neighbors. These neighbors are so-called one hop away from our original vector v. We take the feature vector of these nodes and aggregate the information into one single x_v feature vector. Layer-k would get information from nodes that are k hops away.

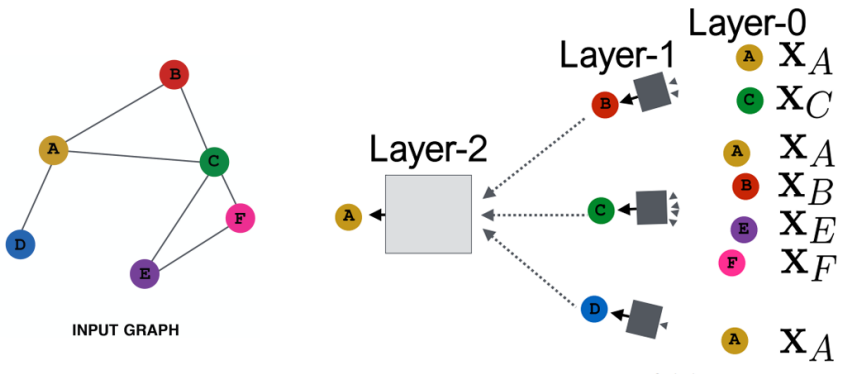


Figure 1.8: Layer-2 embedding applied on node A aggregating infromation from its neighborhood [**?**]

The described process is called neighborhood aggregation. If we want to predict node v, we need information from its neighborhood, meaning we need a way to propagate the message. Messages are passed and transformed through edges. All received messages are aggregated into a new message and then passed on. This is done systematically for every node in the graph.

The aggregation itself is done by a neural network. This implies the key difference from a typical ANN. Every node gets to define its own neural network and GNN is defined by multiple neural networks.

Using a neural network for each node in the graph, we generate a low-dimensional vector representation, embedding. The network optimizes its parameters to capture important information about the node graphs. The

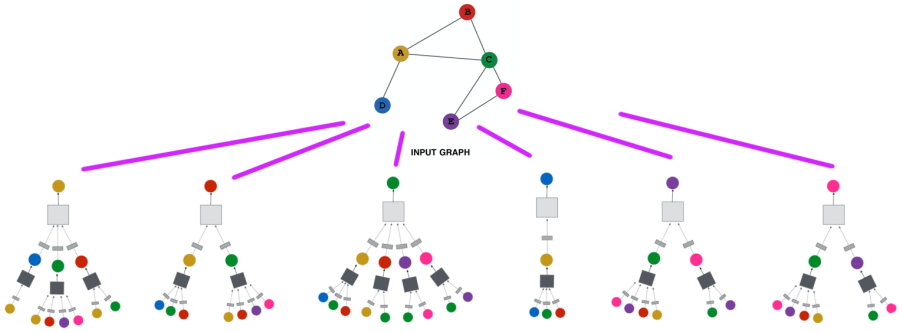


Figure 1.9: Neural network of each node for given input node graph [?]

optimization is typically done by minimizing a loss function expressing dissimilarity between predicted and targeted node embeddings. Similar nodes are close to each other, whereas dissimilar are embedded far apart.

1.2.3.2 Geometric graphs

TODO SECTION Geometric graph is a graph where its nodes represent coordinates in d-dimensional space. Invariant to translation, rotation and scale
... todo more mat...