Sustainable Machine Learning

•••

28.04.2020

About us

Dr. Thorsten Papenbrock

Senior Researcher at HPI
Information Systems Group

Phillip Wenig

PhD Student at HPI,
Information Systems Group

Description

In this project seminar, we develop a prototype system that runs **end-to-end machine learning pipelines** on **clusters of edge devices** with limited compute power to **encourage sustainable hardware usage**.

Background

Artificial intelligence plays an increasingly important role in industry. Many successful modern companies draw their strength from data analytics and machine learning to optimize their business processes, risk management, and decision making. Meanwhile, the topic is so relevant that it became a political concern and the subject of various national AI strategies. Large companies already leverage AI or they have the resources to do so. **Small and medium-sized companies, on the other hand, cannot afford powerful servers or expensive GPU clusters for data analytics and are often reluctant to use cloud services for data protection and privacy reasons.** However, in our experience, they often have spare, low-spec hardware to their disposal, such as disused computers, laptops or phones. In this project, we develop a distributed system that runs an entire data analytics pipeline, including data preparation, feature extraction and model learning, on such devices. **Edge devices in the context of our project are low-spec, potentially aged commodity computers and smartphones as well as resource-saving low-energy devices.** The resulting prototype should enable small companies to use their spare hardware for state-of-the-art machine learning and data analytics. **The project should also encourage a more sustainable hardware management, where functioning hardware does not needlessly get replaced with more powerful hardware just for the sake of analytical experimenting.**

Machine Learning

A machine learning model is the product of a multi-step data engineering and training process. Each step usually requires a significant amount of resources, i.e., CPU power and memory. To execute this process without high-performance hardware, we need to find resource-saving solutions for each of the following steps:

1. **Data Preparation & Cleaning**
2. **Data Profiling & Analysis**
3. **Data Classification & Prediction**

Machine Learning

A machine learning model is the product of a multi-step data engineering and training process. Each step usually requires a significant amount of resources, i.e., CPU power and memory. To execute this process without high-performance hardware, we need to find resource-saving solutions for each of the following steps:

1. **Data Preparation & Cleaning**
   Automatically fix the structural integrity, i.e., table shape, data types, and value formats, (data preparation) and solve data quality issues, such as duplicates and missing values (data cleaning). Some of these operations, are complex tasks (in O(n^2)) and, therefore, a challenge for edge device hardware.

2. **Data Profiling & Analysis**

3. **Data Classification & Prediction**

Machine Learning

A machine learning model is the product of a multi-step data engineering and training process. Each step usually requires a significant amount of resources, i.e., CPU power and memory. To execute this process without high-performance hardware, we need to find resource-saving solutions for each of the following steps:

1. **Data Preparation & Cleaning**

2. **Data Profiling & Analysis**
   Discover implicit metadata, such as data dependencies and constraints, (data profiling) and supplementary statistics, such as aggregates and histograms, (data analysis) that may serve as features. Many data profiling tasks are in O(2^n) and thus particularly challenging for edge device hardware.

3. **Data Classification & Prediction**

Machine Learning

A machine learning model is the product of a multi-step data engineering and training process. Each step usually requires a significant amount of resources, i.e., CPU power and memory. To execute this process without high-performance hardware, we need to find resource-saving solutions for each of the following steps:

1. **Data Preparation & Cleaning**

2. **Data Profiling & Analysis**

3. **Data Classification & Prediction**
   Train a machine learning model. In this seminar, **we focus on deep learning models, i.e., neural networks** and different variants of such. They can be trained for different purposes, such as classification or prediction. The training processes is both data- and compute-intensive and, therefore, has high resource requirements. Fitting them on low-spec hardware devices is a challenge and might require alternative training approaches.

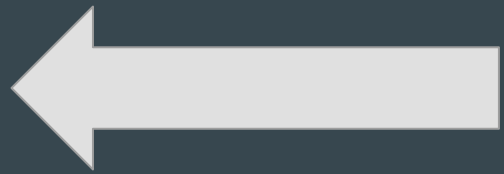
Hardware

For the development and evaluation of the planned machine learning prototype, we use three systems:

1. **Server cluster**
   12 nodes á 10 physical cores and 32 GB RAM. This cluster will produce baseline measurements, with which we can compare the results produced on the low-spec clusters.

2. **Commodity cluster**
   A heterogeneous cluster of (currently 8) de-commissioned desktop computers. The machines are about 5-15 years old and cover dual core to quad core CPUs as well as 2-6 GB of RAM.

3. **Raspberry Pi cluster**
   12 Raspberry Pi 4 model B. The Pi's have dual cores and 4 GB RAM.



Hardware

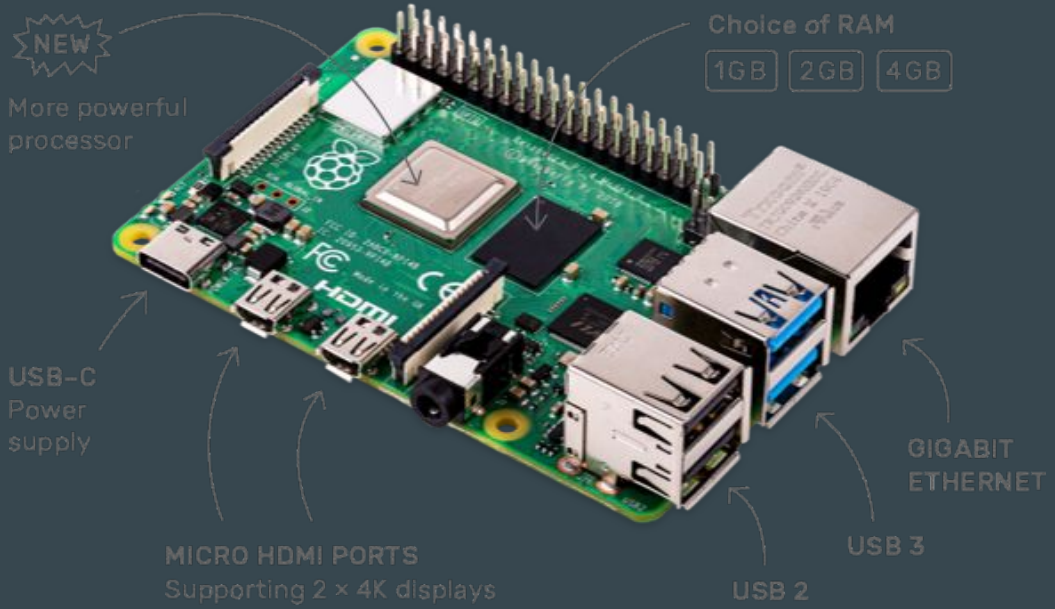
Raspberry Pi cluster

12 Raspberry Pi 4 model B. The Pi's have dual cores and 4 GB RAM.

Hardware

Raspberry Pi cluster

12 Raspberry Pi 4 model B. The Pi's have dual cores and 4 GB RAM.



Project Description

For most of the tasks in the multi-step training process, **parallel and distributed algorithms exist**. These algorithms are, however, usually optimized for powerful systems and struggle with low-spec hardware, i.e., they **cannot cope with heterogeneity, starve on slow processors and quickly exhaust available main memory**. We therefore aim to find solutions for these tasks that are more resource-aware and, hence, run on heterogeneous edge device clusters. For this, we exploit **reactive programming approaches that dynamically adapt to resource bottlenecks and special data characteristics**. Reactive strategies can dynamically change the data engineering and training processes based on intermediate results and they can be used to perform hyperparameter-tuning at runtime. We also look into data compression and summarization techniques to fit our processes on weaker systems and we use instance selection and approximation techniques to cope with very large input datasets. The result will be a working end-to-end model training prototype that runs distributedly on edge device clusters. We measure its execution time, evaluate how far we can reduce the resource consumption, and assess the quality of the trained models.

Project Description

In this project, we work in teams of á 2 students. The tasks for each team are the following:

- Define your scope by choosing ...
  a. **one type of neural network** (artificial, convolutional, recurrent, graph, ...);
  b. **one type of data** (image, graph, relational, text, ...);
  c. **at least one specific use case** (classification, prediction, clustering, ...).
- Analyse state-of-the-art solutions for your sub-project and identify bottlenecks, weaknesses and disadvantages when running them on low-spec hardware.
- Design and implement a more efficient/robust/applicable solution (using e.g. Akka) that circumvents these issues.
  (we do not aim to devise a better learning technique, but to make some technique work on edge-device clusters)
- Write a short summary of your insights and improved training pipeline.

Goal

The final artefact of this seminar will be a scientific paper “Sustainable Machine Learning on Edge Device Clusters” where we investigate how deep learning models can be trained on low-spec clusters.

Story:

- We investigate in 6 case studies what existing deep learning solutions are capable of when executed on low spec-hardware, demonstrate their limitations and propose practical solutions for these limitations.

Contributions for each sub-project:

1. A state-of-the-art evaluation on low-spec hardware clusters
2. A prototype that solves the training on low-spec hardware
   (which is more or less similar to an existing solution)
3. Solutions for different problems that are caused by the hardware limitations and distributed training
   (model and data partitioning, memory management, robustness, workload balancing, ...)
4. A careful evaluation

Goal

The final artefact of this seminar will be a scientific paper “Sustainable Machine Learning on Edge Device Clusters” where we investigate how deep learning models can be trained on low-spec clusters.

We write the introduction and conclusion for our common paper. Each team contributes ...

1. **a scientific, 2 page paper section** about their NN type, data, scenario, state-of-the-art analysis and own solution
2. **a prototype that runs end-to-end machine learning pipelines on edge devices.**

Written Summary

- **Introduction**
  a. Example: “Convolutional Neural Networks”
  b. Content: Short explanation of the chosen NN type; what makes this NN different from others, how does it conceptually work, and what are the challenges when trained in low-spec hardware.
- **Use case**
  a. Example: Image classification
  b. Content: Description of the focus use case, the learning objective and the input data.
- **State-of-the-art**
  a. Example: (distributed) PyTorch, Tensorflow or SparkML implementations.
  b. Content: What existing solutions for distributed training systems exist for your NN type, how do they work, and how do they perform on low-spec hardware, i.e., where are the problems.
- **Own approach**
  a. Example: A pipeline that builds on Akka and PyTorch.
  b. Content: Demonstration on how the problems with existing solutions can be overcome; technical description of the pipeline and careful, scientific evaluation.

Teaching Goals

1. Study an interesting and relevant deep learning technique
2. Collect first experiences with PyTorch and Tensorflow
3. Practice in distributed computing
4. Develop a complex project in Akka/PyTorch/Tensorflow/...
5. Work with real-life system limitations

Project Outline

1. Find a topic (NN type + data type + use case)
2. Study your topic (paper, documentation, blogs)
3. Construct your use case (e.g. search kaggle for datasets and classification tasks)
4. Setup existing solutions on your laptop and on one PI (using solutions from e.g. Tensorflow and PyTorch)
5. Evaluate the existing solutions (based on your specific use case)
6. Distribute an existing solution (bia e.g. PyTorch Distributed)
7. Analyze: Does it work? How fast is it? How robust is it? When does it crash and why?

→ **Decision on how to proceed: Optimize existing pipelines vs. rebuild them using e.g. Akka?**

8. Read up on Java/Python memory management and application profiling; read up on Akka
9. Enter optimization phase:
    a. Rebuild the model training pipeline in Akka
    b. Optimize the model training pipeline: e.g. model distribution; disk spilling, data partitioning, ...
    c. Test, improve, test, improve, test, ...
10. Write your 2-2,5 pages case study report

Next Steps

1. Find a topic (NN type + data type + use case)
    a. We provide some first material in the remainder of this session
    b. Then find two or three projects that you would like to work on
    c. Afterwards we together assign the topics
2. Study your topic (paper, documentation, blogs, ...)

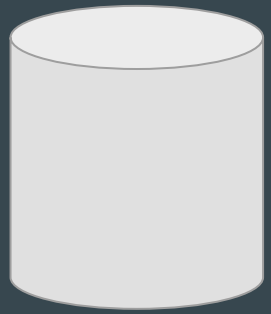
- We will provide additional material for **good practices in writing java code** and on **scientific paper writing** in the next session.

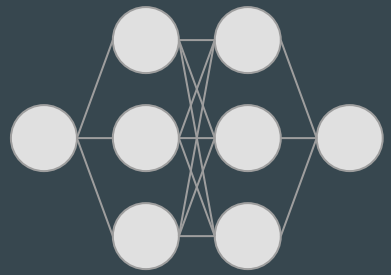
Akka

Overview on Neuronal Network Types

1. Artificial Neural Networks (Vanilla NNs)
2. Convolutional Neural Networks (CNNs)
3. Recurrent Neural Networks (RNNs)
4. Graph Convolution Networks (GCNs)

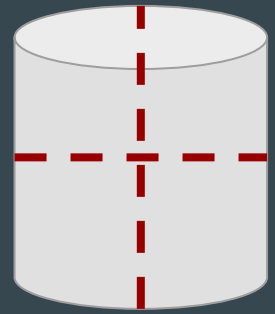
Distributing Machine Learning

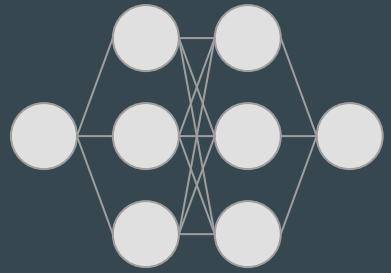




Data Parallel Distribution

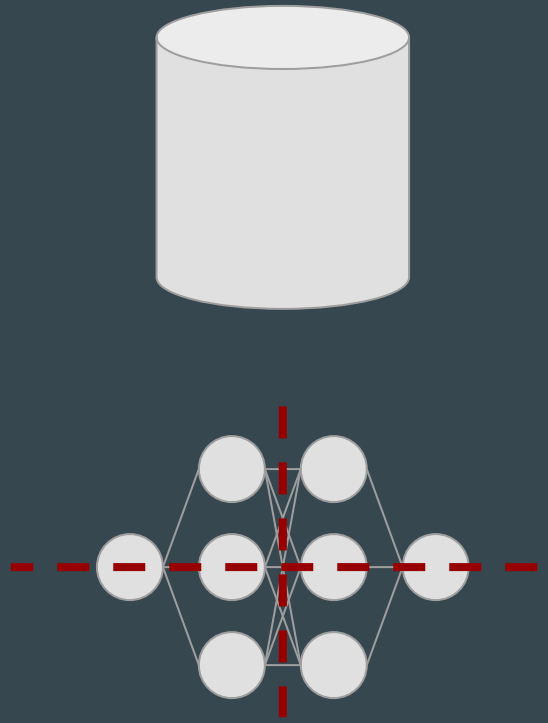
- Split dataset across multiple nodes in a cluster
- Often done with random partitioning
- Training a model on each node
- Communicating between nodes to update the models to a global model





Model Parallel Distribution

- Split the model across multiple nodes in a cluster
- Propagate data through the cluster for one training iteration
- Training one model within the cluster
- Communication between nodes to send calculations and gradients



Implementations Data Parallel

```
import torch
import torch.nn as nn
import torch.distributed as dist

# ...

dist.init_process_group(backend=GLOO, rank=0, size=12,
                        master_address=MA, master_port=MP)
model = nn.parallel.DistributedDataParallel(model)

# ...

dist.destroy_process_group()
```

Implementations Model Parallel

```
import torch
import torch.distributed as dist

# ...

dist.init_process_group(backend=GLOO, rank=rank, size=12,
                        master_address=MA, master_port=MP)
model = model_part

dist.recv(intermediate_input, src=rank-1)
intermediate_output = model(intermediate_input)
dist.send(intermediate_output, dst=rank+1)

# ...

dist.destroy_process_group()
```

Material

- https://pytorch.org/docs/stable/distributed.html?highlight=distributed#module-torch.distributed
- https://pytorch.org/docs/stable/notes/ddp.html#internal-design

Implementations

```
import tensorflow as tf
import tensorflow_federated as tff
# ...

@tff.federated_computation(tff.FederatedType(tf.float32, tff.CLIENTS))
def get_average_temperature(sensor_readings):
    return tff.federated_mean(sensor_readings)

# ...
get_average_temperature([68.5, 70.3, 69.8])  # > 69.53334
```

Implementations

```
# …

SERVER_FLOAT_TYPE = tff.FederatedType(tf.float32, tff.SERVER)


@tff.federated_computation(SERVER_MODEL_TYPE, SERVER_FLOAT_TYPE, CLIENT_DATA_TYPE)
def federated_train(model, learning_rate, data):
  return tff.federated_mean(
      tff.federated_map(local_train, [
          tff.federated_broadcast(model),
          tff.federated_broadcast(learning_rate), data
      ]))
```

Running on Multiple Machines

Multi-machine simulations on GCP/GKE, GPUs, TPUs, and beyond...

Coming very soon.

Material

- https://www.tensorflow.org/federated/tutorials/custom_federated_algorithms_1

Problem Settings

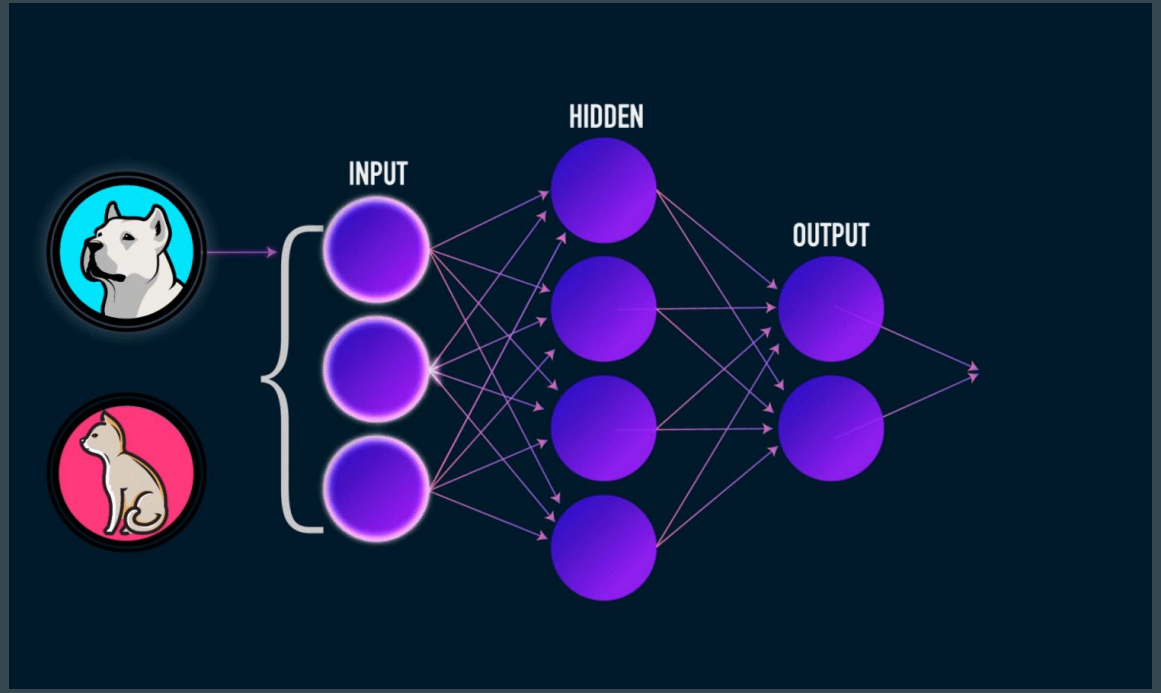
Artificial Neural Networks

Data:

Structured relational data

Distribution:

- Dataset is too big for one device
- Model is too big for one device
- both



https://cdn-images-1.medium.com/max/1600/1*bhFifratH9DjKqMBTeQG5A.gif

Material

Papers:

- LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton. "Deep learning." nature 521.7553 (2015): 436-444. https://www.nature.com/articles/nature14539.pdf

Blog Posts:

- http://neuralnetworksanddeeplearning.com/chap1.html
- https://towardsdatascience.com/lets-code-a-neural-network-in-plain-numpy-ae7e74410795

Implementations:

- https://pytorch.org
- https://tensorflow.org

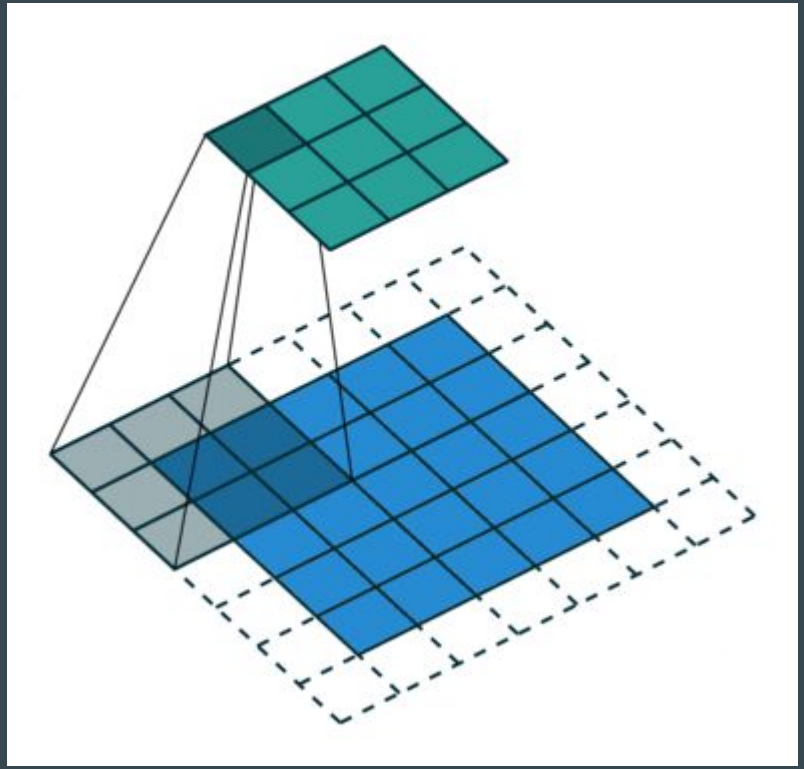
Convolutional Neural Networks

Data:

(Often) Image Data

Distribution:

- Dataset is too big for one device
- Model is too big for one device
- both



https://www.jeremyjordan.me/content/images/2018/04/padding_strides.gif

Material

Papers:

- Krizhevsky, Alex. "One weird trick for parallelizing convolutional neural networks." arXiv preprint arXiv:1404.5997 (2014). https://arxiv.org/abs/1404.5997

Blog Posts:

- https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

Implementations:

- https://pytorch.org/hub/pytorch_vision_alexnet/

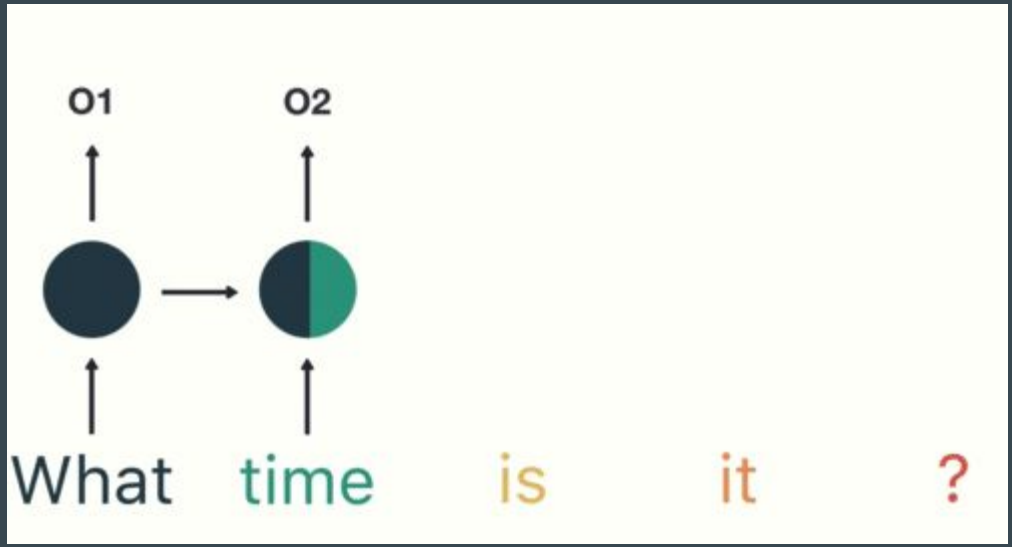
Recurrent Neural Networks

Data:

- Time series
- Natural language

Distribution:

- Dataset is too big for one device
- Model is too big for one device
- both
- Backpropagation is very expensive through time factor



https://cdn-images-1.medium.com/max/1600/1*d_POV7c8fzHbKuTgJzCxtA.gif

Material

Papers:

- Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." Neural computation 9.8 (1997): 1735-1780.

Blog Posts:

- https://towardsdatascience.com/recurrent-neural-networks-d4642c9bc7ce
- https://medium.com/mlreview/understanding-lstm-and-its-diagrams-37e2f46f1714

Implementations:

- https://pytorch.org/docs/stable/nn.html#rnn

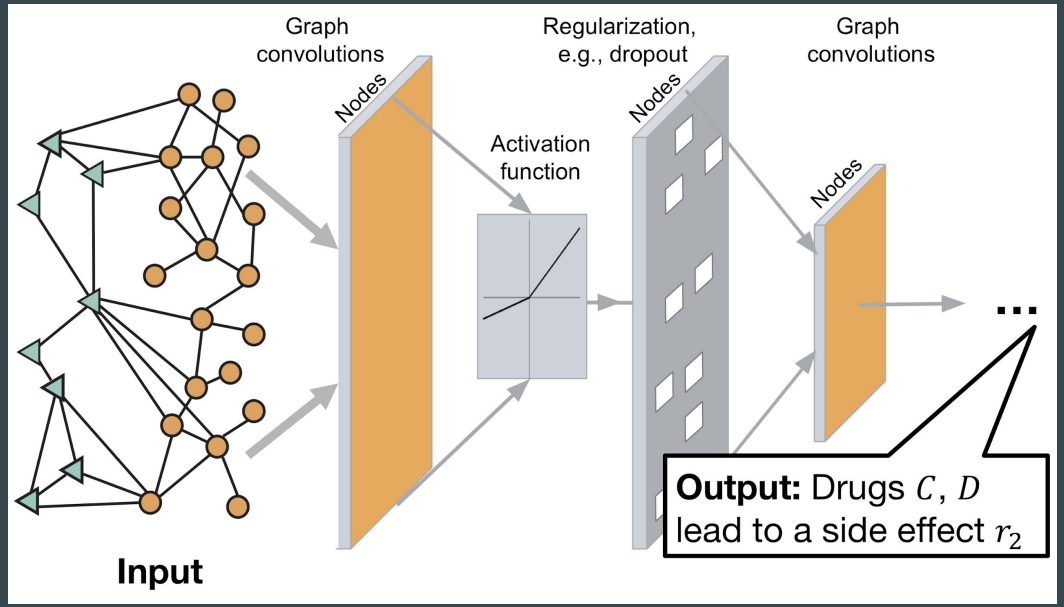
Graph Convolutional Networks

Data:

- Graphs
  - Molecules
  - Social Networks
  - (Text)

Distribution:

- Graph is too big for one device
- Model is too big for one device
- both



http://snap.stanford.edu/decagon/decagon-overview.png

Material

Papers:

- Kipf, Thomas N., and Max Welling. "Semi-supervised classification with graph convolutional networks." arXiv preprint arXiv:1609.02907 (2016). http://arxiv.org/abs/1609.02907

Blog Posts:

- https://towardsdatascience.com/how-to-do-deep-learning-on-graphs-with-graph-convolutional-networks-7d2250723780
- https://tkipf.github.io/graph-convolutional-networks/

Implementations:

- https://github.com/tkipf/gcn
- https://github.com/tkipf/gcn/issues/4