Machine Learning on Graphs

PyTorch London Meetup

Dan Saattrup Nielsen
Danish Business Authority

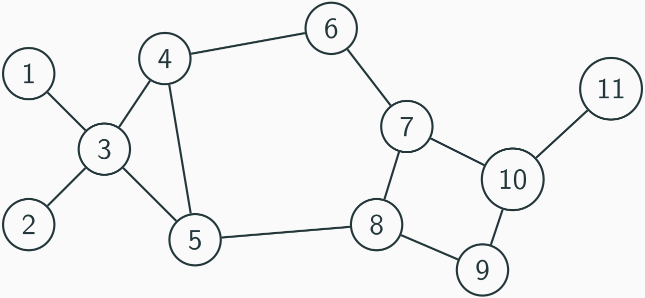
November 3, 2020

An overview of the talk

An overview of the talk

1. What is a graph?
2. Which machine learning tasks can we do on graphs?
   2.1 Node classification
   2.2 Graph classification
   2.3 Link prediction
3. A zoo of algorithms
   3.1 Transductive classification: DeepWalk
   3.2 Inductive classification: Graph convolutions
4. PyTorch implementation
5. Application: Fraud detection

What is a graph?



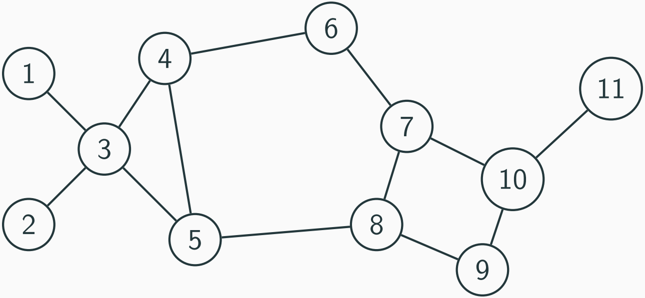
An overview of the talk

1. What is a graph?
2. Which machine learning tasks can we do on graphs?
   2.1 Node classification
   2.2 Graph classification
   2.3 Link prediction
3. A zoo of algorithms
   3.1 Transductive classification: DeepWalk
   3.2 Inductive classification: Graph convolutions
4. PyTorch implementation
5. Application: Fraud detection

An overview of the talk

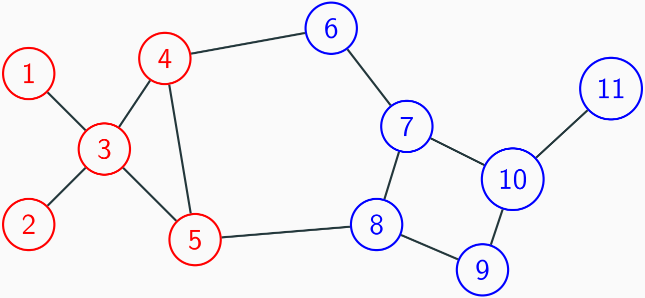
1. What is a graph?
2. Which machine learning tasks can we do on graphs?
   2.1 Node classification
   2.2 Graph classification
   2.3 Link prediction
3. A zoo of algorithms
   3.1 Transductive classification: DeepWalk
   3.2 Inductive classification: Graph convolutions
4. PyTorch implementation
5. Application: Fraud detection

Node classification



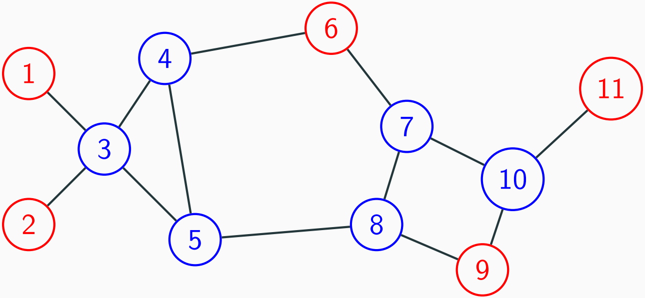
Node classification

Proximity classification:



Node classification

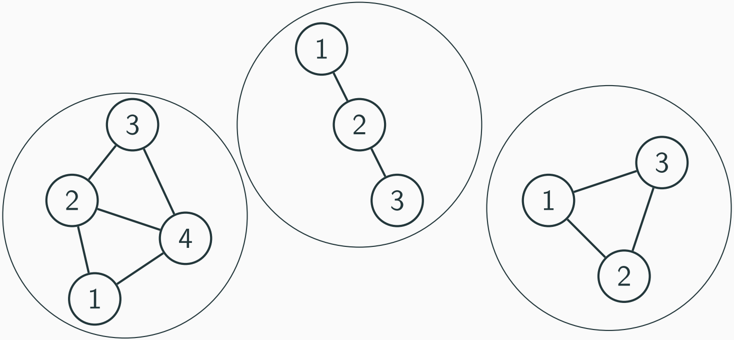
Structural classification:



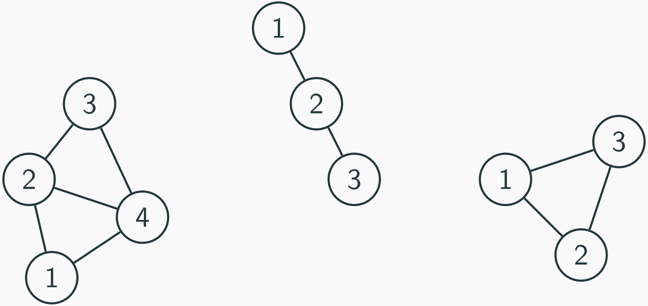
An overview of the talk

1. What is a graph?
2. Which machine learning tasks can we do on graphs?
   2.1 Node classification
   2.2 Graph classification
   2.3 Link prediction
3. A zoo of algorithms
   3.1 Transductive classification: DeepWalk
   3.2 Inductive classification: Graph convolutions
4. PyTorch implementation
5. Application: Fraud detection

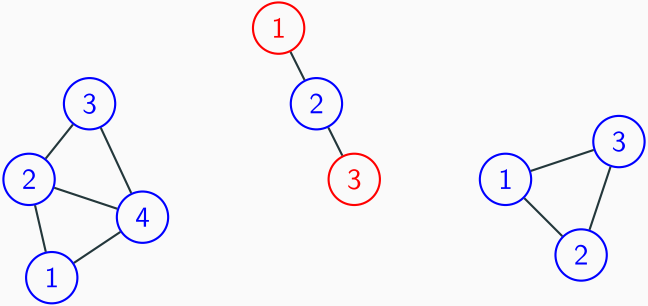
Graph classification



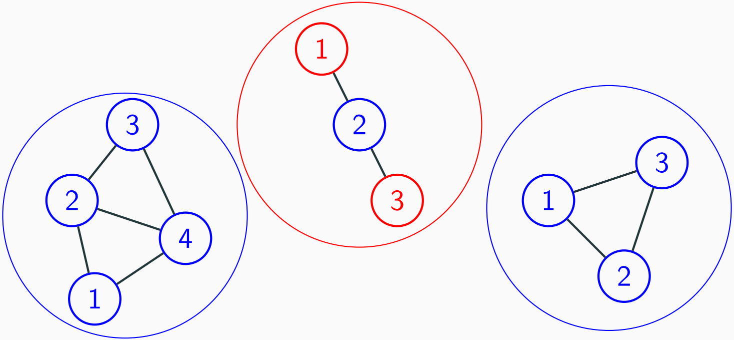
Graph classification



Graph classification



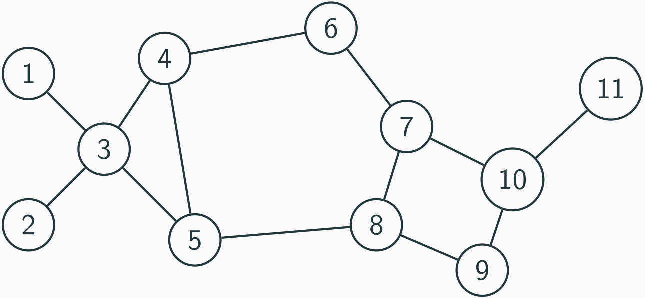
Graph classification



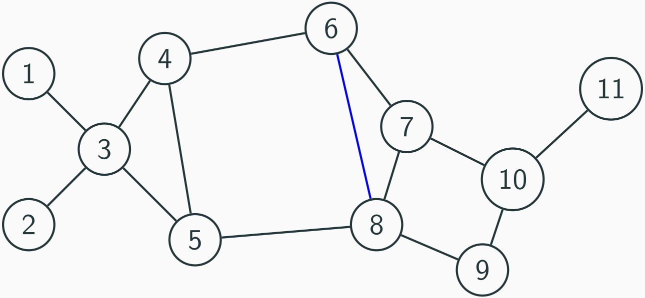
An overview of the talk

1. What is a graph?
2. Which machine learning tasks can we do on graphs?
    2.1 Node classification
    2.2 Graph classification
    2.3 Link prediction
3. A zoo of algorithms
    3.1 Transductive classification: DeepWalk
    3.2 Inductive classification: Graph convolutions
4. PyTorch implementation
5. Application: Fraud detection

Link prediction



Link prediction

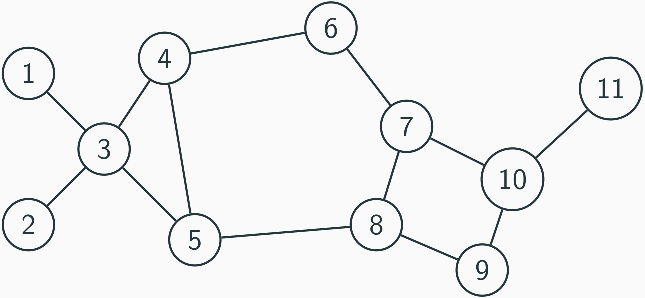


An overview of the talk

An overview of the talk

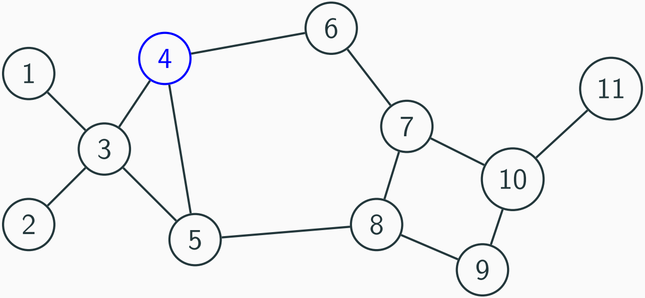
1. What is a graph?
2. Which machine learning tasks can we do on graphs?
   2.1 Node classification
   2.2 Graph classification
   2.3 Link prediction
3. A zoo of algorithms
   3.1 Transductive classification: DeepWalk
   3.2 Inductive classification: Graph convolutions
4. PyTorch implementation
5. Application: Fraud detection

Transductive classification: DeepWalk



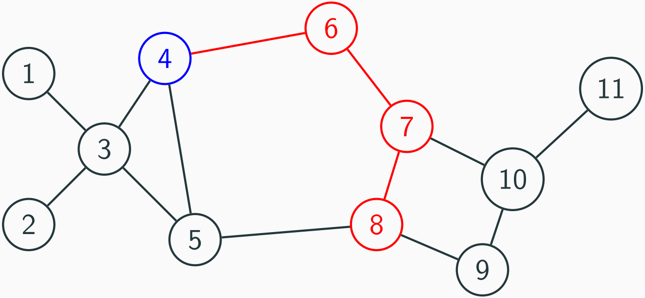
Transductive classification: DeepWalk

For every node...



Transductive classification: DeepWalk

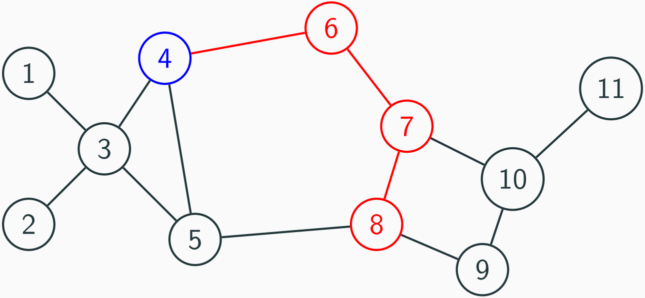
Go for a random walk...



Transductive classification: DeepWalk

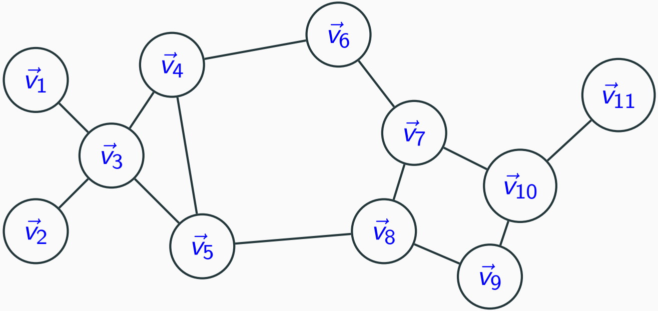
And teach a model to predict the node's neighbours:

$$\texttt{model}(4) \in \{6, 7, 8\}$$



Transductive classification: DeepWalk

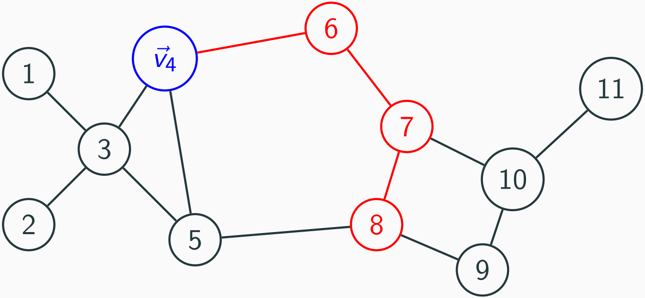
This is done by assigning a learnable vector $\vec{v}_i \in \mathbb{R}^d$ to every node i



Transductive classification: DeepWalk

And training a neural network to predict node indices on the random walk:

$$\mathtt{net}(\vec{v}_4) = (0, 0, 0, 0, 0, \tfrac{1}{3}, \tfrac{1}{3}, \tfrac{1}{3}, 0, 0, 0)$$

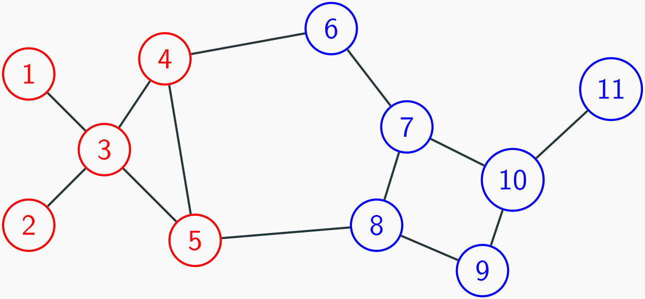


Transductive classification: DeepWalk

From these embeddings we can then train any classification model (say, logistic regression) to get our node labels:

$$\texttt{classificationModel}(\vec{v}_4) \sim 0$$

$$\texttt{classificationModel}(\vec{v}_7) \sim 1$$



Transductive classification: DeepWalk

This algorithm is transductive, meaning that we learn a static embedding (and thus, classification) for every node.

If a new node appears, we thus have to train all the embeddings from scratch.

Transductive classification: DeepWalk

This algorithm is transductive, meaning that we learn a static embedding (and thus, classification) for every node.

If a new node appears, we thus have to train all the embeddings from scratch.

Note also that this is, by construction, a proximity classification: nearby nodes will get classified similarly.

An overview of the talk

Inductive classification: Graph convolutions

Let's start with a quick recap of what convolutional neural networks are doing.

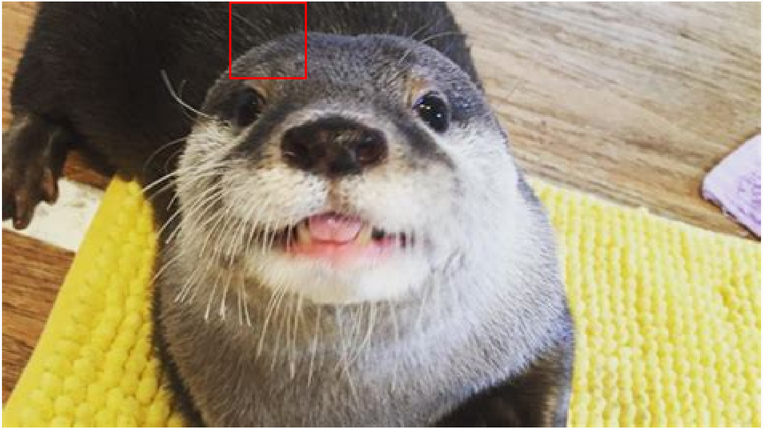
Inductive classification: Graph convolutions

Inductive classification: Graph convolutions

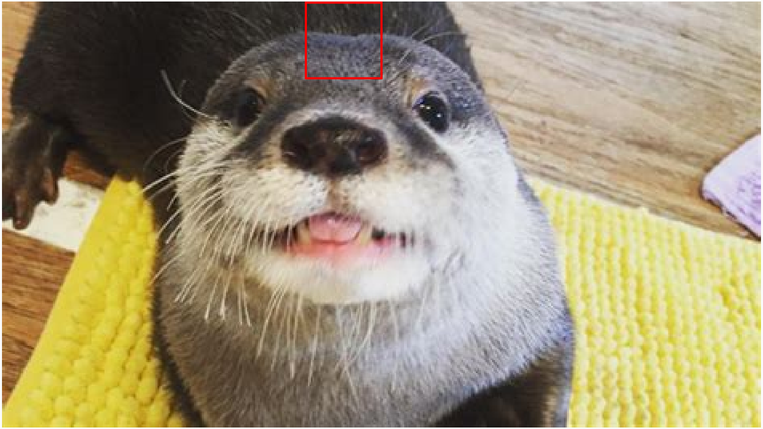
Inductive classification: Graph convolutions

Inductive classification: Graph convolutions

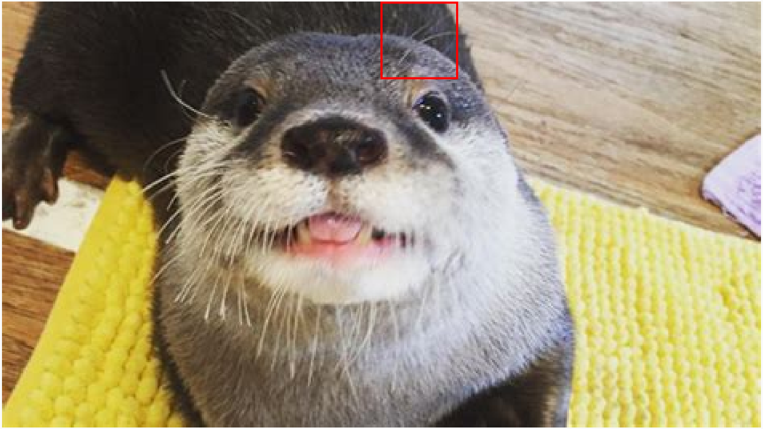
Inductive classification: Graph convolutions



Inductive classification: Graph convolutions



Inductive classification: Graph convolutions



Inductive classification: Graph convolutions

Inductive classification: Graph convolutions

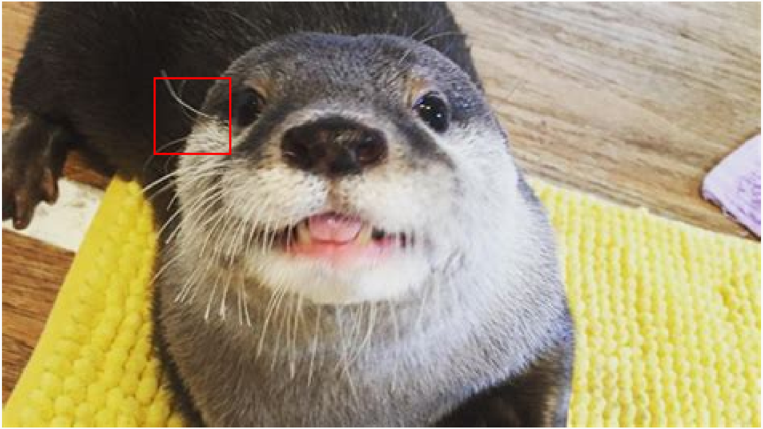
Inductive classification: Graph convolutions

Inductive classification: Graph convolutions

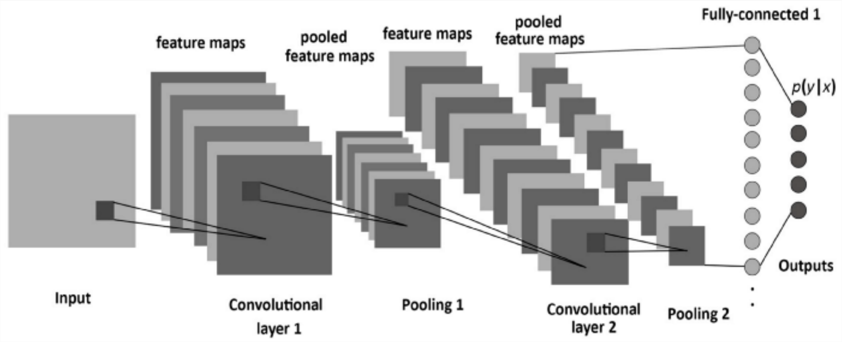
Inductive classification: Graph convolutions

Inductive classification: Graph convolutions

Inductive classification: Graph convolutions



Inductive classification: Graph convolutions

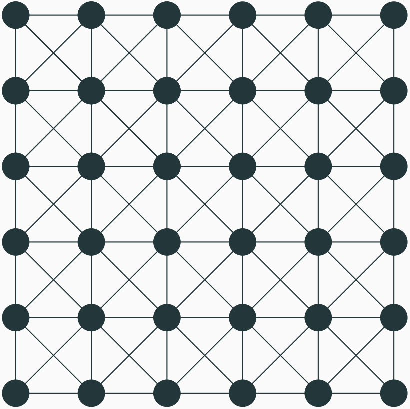


[www.mdpi.com/entropy/entropy-19-00242/article_deploy/html/images/entropy-19-00242-g001.png]

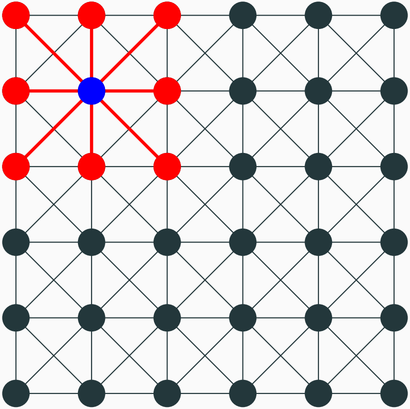
Inductive classification: Graph convolutions

Let's view the cute otter with our graph hat on.

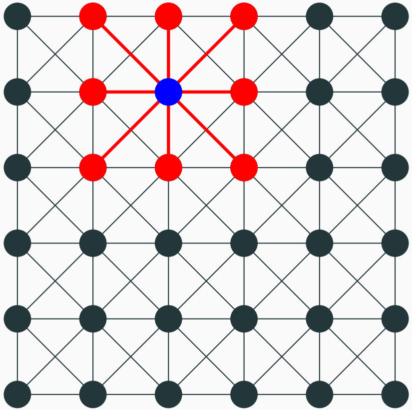
Inductive classification: Graph convolutions



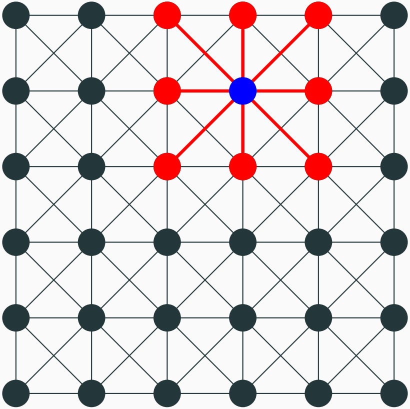
Inductive classification: Graph convolutions



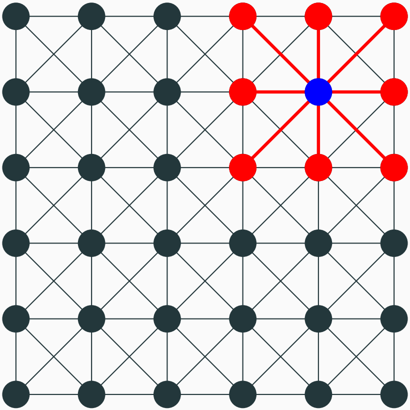
Inductive classification: Graph convolutions



Inductive classification: Graph convolutions



Inductive classification: Graph convolutions



Inductive classification: Graph convolutions

Note that every node has the same number of neighbours

Inductive classification: Graph convolutions

Note that every node has the same number of neighbours (if we included padding).

Inductive classification: Graph convolutions

Note that every node has the same number of neighbours (if we included padding).

This makes it possible to learn a 3×3 matrix (the kernel), which we can convolve over:

$$(\texttt{pixels} \star \texttt{kernel})_{m,n} := \sum_{i=-1}^{1} \sum_{j=-1}^{1} \texttt{kernel}_{i,j} \texttt{pixels}_{m-i,n-j}$$

Inductive classification: Graph convolutions

Note that every node has the same number of neighbours (if we included padding).

This makes it possible to learn a 3×3 matrix (the kernel), which we can convolve over:

$$(\texttt{pixels} \star \texttt{kernel})_{m,n} := \sum_{i=-1}^{1} \sum_{j=-1}^{1} \texttt{kernel}_{i,j} \texttt{pixels}_{m-i,n-j}$$

Crucially, for this to work, we are able shift in the x- and/or y-direction.

Inductive classification: Graph convolutions

Note that every node has the same number of neighbours (if we included padding).

This makes it possible to learn a 3×3 matrix (the kernel), which we can convolve over:

$$(\texttt{pixels} \star \texttt{kernel})_{m,n} := \sum_{i=-1}^{1} \sum_{j=-1}^{1} \texttt{kernel}_{i,j} \texttt{pixels}_{m-i,n-j}$$

Crucially, for this to work, we are able shift in the x- and/or y-direction.

We lose this feature for general graphs, so what do we do?

Inductive classification: Graph convolutions

MATHS

Inductive classification: Graph convolutions

The Convolution Theorem
Convolutions between two functions $f, g\colon \mathbb{R}^n \to \mathbb{R}$ are equivalent to element-wise multiplication in the Fourier domain:

$$\texttt{fourier}(f \star g) = \texttt{fourier}(f) \cdot \texttt{fourier}(g)$$

Inductive classification: Graph convolutions

The Convolution Theorem
Convolutions between two functions $f, g \colon \mathbb{R}^n \to \mathbb{R}$ are equivalent to element-wise multiplication in the Fourier domain:

$$\texttt{fourier}(f \star g) = \texttt{fourier}(f) \cdot \texttt{fourier}(g)$$

One can define a version of the Fourier transform applied to functions on nodes of a graph, call it `graphFourier`, and then define the graph convolution as

$$f \star g := \texttt{graphFourier}^{-1}(\texttt{graphFourier}(f) \cdot \texttt{graphFourier}(g))$$

Inductive classification: Graph convolutions

The Convolution Theorem
Convolutions between two functions $f, g \colon \mathbb{R}^n \to \mathbb{R}$ are equivalent to element-wise multiplication in the Fourier domain:

$$\mathtt{fourier}(f \star g) = \mathtt{fourier}(f) \cdot \mathtt{fourier}(g)$$

One can define a version of the Fourier transform applied to functions on nodes of a graph, call it `graphFourier`, and then define the graph convolution as

$$f \star g := \mathtt{graphFourier}^{-1}(\mathtt{graphFourier}(f) \cdot \mathtt{graphFourier}(g))$$

This is called a spectral graph convolution.

Inductive classification: Graph convolutions

The problem is that spectral graph convolutions are computationally expensive ($\mathcal{O}(\texttt{numNodes}^2)$).

Inductive classification: Graph convolutions

The problem is that spectral graph convolutions are computationally expensive ($\mathcal{O}(\texttt{numNodes}^2)$).

However, Hammond et al. (2011) from EPFL in Lausanne came up with an approximation of the graph Fourier transform.

Inductive classification: Graph convolutions

The problem is that spectral graph convolutions are computationally expensive ($\mathcal{O}(\texttt{numNodes}^2)$).

However, Hammond et al. (2011) from EPFL in Lausanne came up with an approximation of the graph Fourier transform.

Kipf and Welling (2017) from the University of Amsterdam used this to construct an approximation to the spectral graph convolution which is $\mathcal{O}(\texttt{numEdges})$.

Inductive classification: Graph convolutions

The problem is that spectral graph convolutions are computationally expensive ($\mathcal{O}(\texttt{numNodes}^2)$).

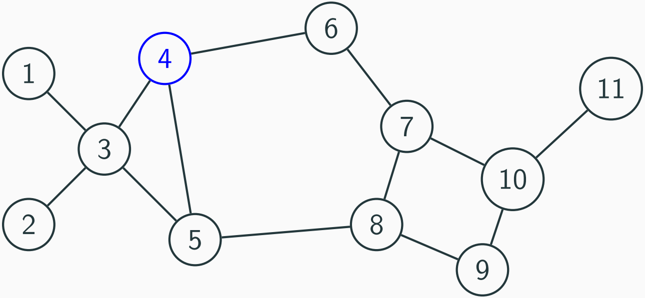
However, Hammond et al. (2011) from EPFL in Lausanne came up with an approximation of the graph Fourier transform.

Kipf and Welling (2017) from the University of Amsterdam used this to construct an approximation to the spectral graph convolution which is $\mathcal{O}(\texttt{numEdges})$.

This approximation is also quite simple. Let's have a look.

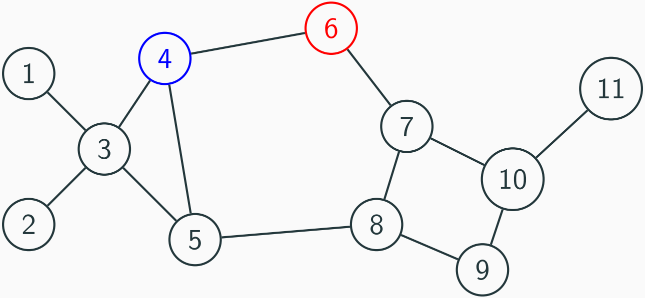
Inductive classification: Graph convolutions

Say we want to compute the convolution at node 4.



Inductive classification: Graph convolutions

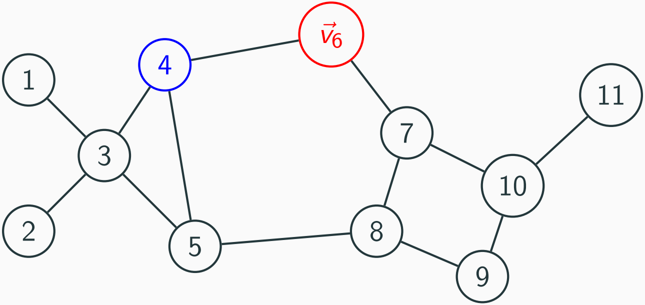
For every neighbour of 4...



Inductive classification: Graph convolutions

We take its feature vector and normalise it by both it's and 4's degrees...

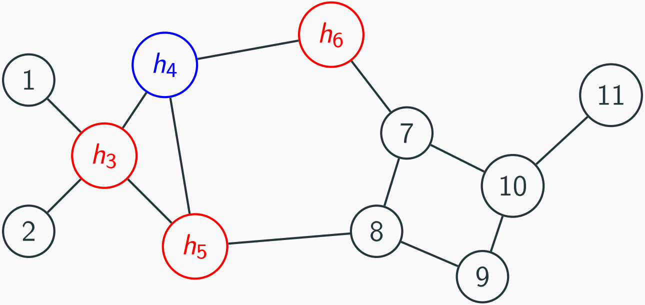
$$h_6 := \frac{\vec{v}_6}{\sqrt{\texttt{degree}(4)\texttt{degree}(6)}} = \frac{\vec{v}_6}{\sqrt{3 \cdot 2}}$$



Inductive classification: Graph convolutions

Node 4's new value is then the sum of the h_i's, multiplied with a learnable weight matrix W:

$$\vec{v}_4 := W(h_4 + h_3 + h_5 + h_6)$$



Inductive classification: Graph convolutions

This algorithm is inductive, meaning that if a new node appears in the graph, then we use the pre-trained model to do inference on it

Inductive classification: Graph convolutions

This algorithm is inductive, meaning that if a new node appears in the graph, then we use the pre-trained model to do inference on it

As opposed to DeepWalk, the GCNs are inherently supervised, but methods exist to train this in an unsupervised way (e.g. the Deep Graph Infomax algorithm from Veličković et al., 2018)

An overview of the talk

1. What is a graph?
2. Which machine learning tasks can we do on graphs?
   2.1 Node classification
   2.2 Graph classification
   2.3 Link prediction
3. A zoo of algorithms
   3.1 Transductive classification: DeepWalk
   3.2 Inductive classification: Graph convolutions
4. PyTorch implementation
5. Application: Fraud detection

PyTorch implementation

Deep Graph Library: `https://www.dgl.ai`

PyTorch implementation

Deep Graph Library: `https://www.dgl.ai`

```
import torch
import torch.nn as nn
import dgl
import dgl.nn.pytorch as dglnn

class GCN(nn.Module):
    def __init__(self, in_feats:int, hidden_size:int, num_classes:int):
        super().__init__()
        self.conv1 = dglnn.GraphConv(in_feats, hidden_size)
        self.conv2 = dglnn.GraphConv(hidden_size, num_classes)

    def forward(self, graph:dgl.DGLGraph, x:torch.tensor):
        x = self.conv1(graph, x)
        x = torch.relu(x)
        x = self.conv2(graph, x)
        return x
```

PyTorch implementation

Deep Graph Library: `https://www.dgl.ai`

```
import torch
import torch.nn as nn
import dgl
import dgl.nn.pytorch as dglnn

class GCN(nn.Module):
    def __init__(self, in_feats:int, hidden_size:int, num_classes:int):
        super().__init__()
        self.conv1 = dglnn.GraphConv(in_feats, hidden_size)
        self.conv2 = dglnn.GraphConv(hidden_size, num_classes)

    def forward(self, graph:dgl.DGLGraph, x:torch.tensor):
        x = self.conv1(graph, x)
        x = torch.relu(x)
        x = self.conv2(graph, x)
        return x
```

PyTorch implementation

Deep Graph Library: `https://www.dgl.ai`

```
import torch
import torch.nn as nn
import dgl
import dgl.nn.pytorch as dglnn

class GCN(nn.Module):
    def __init__(self, in_feats:int, hidden_size:int, num_classes:int):
        super().__init__()
        self.conv1 = dglnn.GraphConv(in_feats, hidden_size)
        self.conv2 = dglnn.GraphConv(hidden_size, num_classes)

    def forward(self, graph:dgl.DGLGraph, x:torch.tensor):
        x = self.conv1(graph, x)
        x = torch.relu(x)
        x = self.conv2(graph, x)
        return x
```

PyTorch implementation

PyTorch Geometric: `https://github.com/rusty1s/pytorch_geometric`

PyTorch implementation

PyTorch Geometric: `https://github.com/rusty1s/pytorch_geometric`

```
import torch
import torch.nn as nn
import torch_geometric as tg
import torch_geometric.nn as tgnn

class GCN(nn.Module):
    def __init__(self, in_feats:int, hidden_size:int, num_classes:int):
        super().__init__()
        self.conv1 = tgnn.GCNConv(in_feats, hidden_size)
        self.conv2 = tgnn.GCNConv(hidden_size, num_classes)

    def forward(self, data:tg.data.Data):
        x, edge_index = data.x, data.edge_index
        x = self.conv1(x, edge_index)
        x = torch.relu(x)
        x = self.conv2(x, edge_index)
        return x
```

PyTorch implementation

PyTorch Geometric: `https://github.com/rusty1s/pytorch_geometric`

```
import torch
import torch.nn as nn
import torch_geometric as tg
import torch_geometric.nn as tgnn

class GCN(nn.Module):
    def __init__(self, in_feats:int, hidden_size:int, num_classes:int):
        super().__init__()
        self.conv1 = tgnn.GCNConv(in_feats, hidden_size)
        self.conv2 = tgnn.GCNConv(hidden_size, num_classes)

    def forward(self, data:tg.data.Data):
        x, edge_index = data.x, data.edge_index
        x = self.conv1(x, edge_index)
        x = torch.relu(x)
        x = self.conv2(x, edge_index)
        return x
```

PyTorch implementation

PyTorch Geometric: https://github.com/rusty1s/pytorch_geometric

```
import torch
import torch.nn as nn
import torch_geometric as tg
import torch_geometric.nn as tgnn

class GCN(nn.Module):
    def __init__(self, in_feats:int, hidden_size:int, num_classes:int):
        super().__init__()
        self.conv1 = tgnn.GCNConv(in_feats, hidden_size)
        self.conv2 = tgnn.GCNConv(hidden_size, num_classes)

    def forward(self, data:tg.data.Data):
        x, edge_index = data.x, data.edge_index
        x = self.conv1(x, edge_index)
        x = torch.relu(x)
        x = self.conv2(x, edge_index)
        return x
```

An overview of the talk

1. What is a graph?
2. Which machine learning tasks can we do on graphs?
   2.1 Node classification
   2.2 Graph classification
   2.3 Link prediction
3. A zoo of algorithms
   3.1 Transductive classification: DeepWalk
   3.2 Inductive classification: Graph convolutions
4. PyTorch implementation
5. Application: Fraud detection

Application: Fraud detection

The Danish Business Authority endeavours to create the best conditions for growth in Europe, and to make it easy and attractive to run a business in Denmark.

Application: Fraud detection

The Danish Business Authority endeavours to create the best conditions for growth in Europe, and to make it easy and attractive to run a business in Denmark.

All companies in Denmark have to register with the authority before they are officially recognised as a company.

Application: Fraud detection

The authority has a machine learning lab, who works with assisted tax fraud detection.

Application: Fraud detection

The authority has a machine learning lab, who works with assisted tax fraud detection.

No decisions are taken based on algorithms, however.

Application: Fraud detection

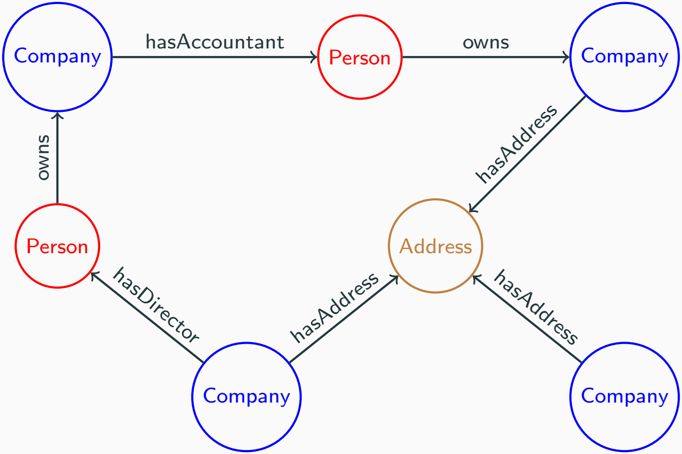
The authority has a machine learning lab, who works with assisted tax fraud detection.

No decisions are taken based on algorithms, however.

The lab has access to data from other authorities, like data about VAT, tax and income.

Application: Fraud detection

Most of our data is organised in a Neo4j graph database.



Application: Fraud detection

This database currently have more than 300 million nodes and 500 million relations.

Application: Fraud detection

This database currently have more than 300 million nodes and 500 million relations.

So far we have been using manually curated graph features in our machine learning models.

Application: Fraud detection

This database currently have more than 300 million nodes and 500 million relations.

So far we have been using manually curated graph features in our machine learning models.

We are currently in the process of implementing graph neural networks to automate this, and (hopefully!) improve the performance of the models as well.

The End

Thank you for your attention.