- GPUs are good with Tensor Multiplication and lots of cores (Like Matrix Multiplication)
- Regularization: Simple is better (Avoid overfitting)

- **Model complexity**: the ability to fit various functions
- **Data complexity**: the richness of information
- **Model selection**: match model and data complexities

DTs are one of the most popular classification methods for practical applications

- Reason #1: The learned representation is **easy to explain** a non-ML person
- Reason #2: They are **efficient** in both computation and memory

DTs can be applied to a wide variety of problems including **classification, regression, density estimation,** etc.

What you can do now

- Define a decision tree classifier
- Interpret the output of decision trees
- Learn a decision tree classifier using greedy algorithm
- Traverse a decision tree to make predictions
  - Majority class predictions
- Tackle continuous and discrete features
- Understand how to control overfitting and underfitting in DT

Question: What is a decision tree and how does it work in machine learning?

Answer: A decision tree is a **flowchart-like tree structure** where an internal node represents a feature(or attribute), the branch represents a decision rule, and each leaf node represents the outcome.

In the context of machine learning, it is used for both **classification** and **regression** tasks. The decision of making strategic splits heavily affects a tree's accuracy.

Question: Discuss how decision trees can be used for both classification and regression.

Answer: For classification, they predict discrete classes and splits are made in such a way that they best separate the classes.

For regression, decision trees predict continuous values. The tree is constructed similarly, but instead of voting for classes in the leaves, it predicts a numerical value, often **the mean of the target variable** of the instances in the leaf.

Question: Can you explain overfitting in decision trees and how to avoid it?

Answer: Overfitting occurs when the decision tree model **fits too closely** to the training data, including the **noise or random fluctuations** in the data. As a result, it performs poorly on unseen data.

To avoid overfitting, techniques such as **pruning** (removing parts of the tree that provide little power in classifying instances), **setting a minimum number of samples per leaf**, or maximum depth of the tree, and using random forests (an ensemble of decision trees) can be employed.

Question: What are the main advantages and disadvantages of using decision trees?

Answer: The advantages of decision trees are that they are easy to understand, **interpret**, and visualize. They can handle both numerical an categorical data and can model complex, non-linear relationships.

The disadvantages include a **tendency to overfit**, **sensitivity to small changes in the data**, and the problem of creating biased trees if some classes dominate.

KNN

What is the effect of k? **control the model capacity/complexity**

Will a large k make the decision boundary coarse or smooth? **Smoother**

How will that affect the overfitting and underfitting? **Small k leads to overfitting and large k leads to underfitting**

- Pros:
  - Intuitive / explainable
  - No training/retraining
  - Provably near-optimal in terms of true error rate
- Cons:
  - Computationally expensive
    - Always needs to store all data: $O(ND)$
    - Finding the k closest points in D dimensions: $O(ND + N\log(k))$
      - Can be sped up through clever use of data structures (trades off training and test costs)
      - Can be approximated using stochastic methods
  - Affected by feature scale Intuitive/explainable

Question: What are the advantages and disadvantages of KNN?

Answer: Advantages of KNN include its **simplicity**, effectiveness, and the fact that it requires **no training** phase. It's versatile, being used for both classification and regression.

However, KNN has several disadvantages like being **computationally expensive**, particularly with large datasets, as it requires storing all the training data. It's also **sensitive to the scale of the data and irrelevant features**, which can significantly degrade its performance.

Question: How does the choice of distance metric affect the performance of KNN?

Answer: The choice of distance metric in KNN can significantly affect its performance. Common distance metrics include Euclidean, Manhattan, and Minkowski distances. Euclidean distance works well in many cases, especially when the features are in the same units. Manhattan distance is more suitable for high-dimensional data. The choice of distance metric should match the data's structure and the problem's nature.

Treat it as a hyperparameter as K.

Question: Can KNN be used for regression? How?

Answer: Yes, KNN can be adapted for regression. In KNN regression, instead of voting for a class, the algorithm calculates the **average (or sometimes the median) of the values of its k nearest neighbors**. This average is then used as the predicted value for the input instance.

Clustering

No labels provided
…uncover cluster structure from input alone

Input: docs as vectors x_i
Output: cluster labels z_i

Question: What are the main challenges in clustering analysis?

Answer: The main challenges include:

- **Determining the Number of Clusters**: It's often not clear how many clusters should be chosen, especially in K-Means.
- **Sensitivity to Initial Conditions**: Some algorithms, like K-Means, are sensitive to the initial choice of centroids.
- **Noise and Outlier Sensitivity**: Some clustering algorithms are sensitive to noise and outliers in the data.
- **Scalability**: Large datasets pose computational challenges.

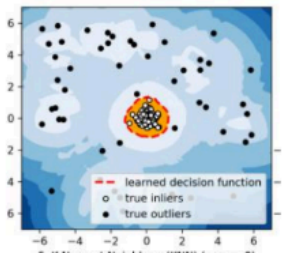
Characteristics & challenges of outlier detection:

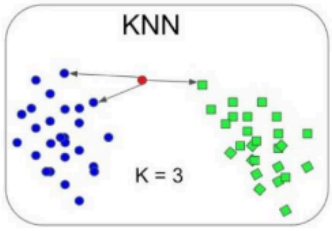
1. Outliers are conducted in an **unsupervised** fashion, no ground truth labels for evaluation
2. Outlier detection often faces **data imbalance**
3. It is often challenging to provide **interpretability** of outlier detection results
4. Outlier detection algorithms are often **costly**—scalable algorithms are needed
5. Data quality can often be a huge problem in OD, with noise and corruption

These challenges tell us that we should consider outlier detection problems by :
(1) availability of labels (2) computational cost and (3) interpretability (4) data quality

OD Algorithm 1: k Neatest Neighbors (kNN)

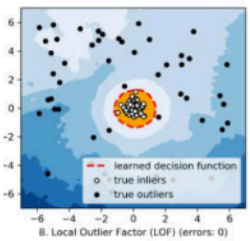
Idea: calculate the distance to the k-th nearest neighbor. The larger, the more anomalous.
Usage: from pyod.models.knn import KNN
Advantages: easy to understand and implement
Disadvantages: high computational cost; not for high dimensional datasets
Decision boundary



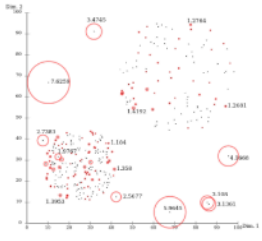


OD Algorithm 2: Local Outlier Factor (LOF)

Idea: calculate sample density on the local region and the density of its neighbors
Usage: from pyod.models.lof import LOF
Advantages: easy to understand (slightly more complex than kNN)
Disadvantages: high computational cost; not for high dimensional datasets
Decision boundary



8. Local Outlier Factor (LOF) (errors: 0)



OD Algorithm 3: Histogram-based OD (HBOS)

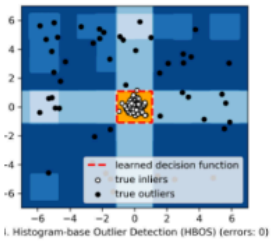
Idea: assume each feature is independent; estimate the histograms separately and combine

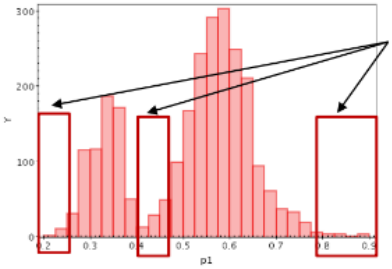
Usage: from pyod.models.hbos import HBOS

Advantages: simple to use; easy to be distributed; suited for large-scale problem

Disadvantages: cannot capture complex feature dependency, while it works well in general

Decision boundary





Low density region

OD Algorithm 4: Isolation Forests (iForest)

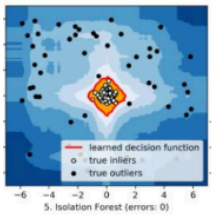
Idea: build multiple decision trees to split the feature space, and observe the difficulty of "isolating" a sample. An outlier is easier to be isolated with a small tree depth.

Usage: from pyod.models.iforest import IForest

Advantages: fast computation; easy to be distributed; and empirically robust

Disadvantages: many hypereparameters to tune with some built-in randomness

Decision boundary



OD Summary

1. Most of the algorithms are unsupervised
2. Detection depends on the data similarity – measured by distance and/or density
3. Data faces extreme imbalance – need to choose the evaluation carefully

Question: What challenges are faced in anomaly detection?

Answer:

- **High False Positive Rate**: Distinguishing between a true anomaly and a legitimate data point that happens to be unusual.
- **Dynamic Data**: In many real-world scenarios, the data is constantly evolving, making it difficult to establish a static model of normal behavior.
- **Imbalanced Data**: Typically, in anomaly detection, the number of normal instances significantly outweighs the number of anomalies.
- **Adversarial setting**: patterns change frequently

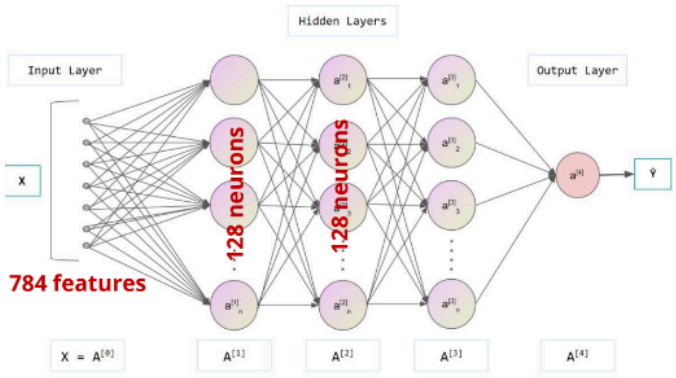
Question: Explain the difference between supervised and unsupervised anomaly detection.

Answer: In supervised anomaly detection, the algorithm is trained on a labeled dataset that contains both normal and anomalous samples. It's essentially a classification problem. However, in **unsupervised anomaly detection**, the algorithm is not trained on labeled data, and it tries to identify anomalies based on the inherent characteristics of the data, often by building a profile of what's normal and flagging data points that deviate significantly from this profile.

Question: How do you evaluate the performance of an anomaly detection system?

Answer: human verification (human-in-the-loop, sampling, etc.)

The performance of anomaly detection systems is typically evaluated using metrics such as precision, recall, F1-score, and the area under the ROC curve (AUC-ROC). Due to the imbalanced nature of anomaly detection problems (where anomalies are rare), precision-recall curves are often more informative than ROC curves.



After the first layer, it will have [1x784] · [784x128] = **1x128** outputs

The second layer also gets 128 neurons, and the weight matrix **W2** for the second layer will be 128x128

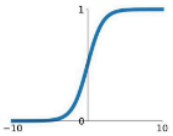
(**Before**) Linear score function: $f = Wx$

(**Now**) 2-layer Neural Network $f = W_2 \max(0, W_1 x)$

or 3-layer Neural Network $f = W_3 \max(0, W_2 \max(0, W_1 x))$

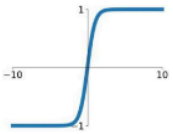
Sigmoid

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



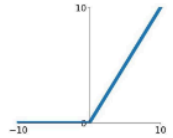
tanh

$\tanh(x)$



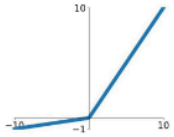
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

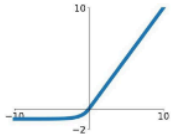


Maxout

$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Sigmoid Function

- **Characteristics**: Sigmoid squashes th input values into a range between 0 an 1. It's historically been used for binary classification.
- **When to use**: In the output layer for binary classification tasks. When you need probabilities as outputs, since its output can be interpreted as a probability.
- **Limitations**: Prone to **vanishing gradient problem (discussed later)**. Outputs are not zero-centered.

ReLU:

Characteristics: ReLU is a piece-wise linear function that outputs the input directly if positive, otherwise, it will output zero.

When to use: In most hidden layers, as it helps with the vanishing gradient problem and speeds up training. Good default choice for deep neural networks.

Limitations:

Can suffer from "dying ReLU" problem, where neurons can become inactive and only output zero for all inputs.

Question: What is an activation function in neural networks and why are they used?

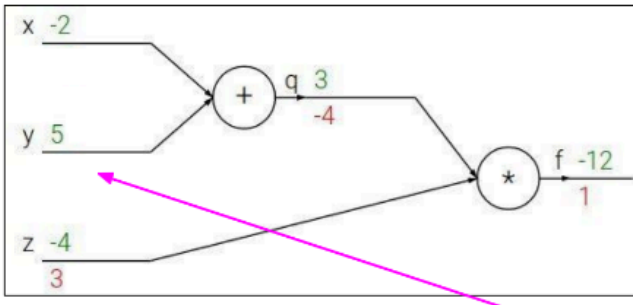
Answer: An activation function in a neural network is a mathematical function applied to the output of a neuron or layer of neurons, transforming the input signal into an output signal.

It's used for introducing **non-linear** properties to the network. Without non-linearity, the neural network would essentially become a linear regression model, incapable of handling complex data patterns.

Backpropagation: a simple example

$$f(x,y,z) = (x+y)z$$

e.g. x = -2, y = 5, z = -4



$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

$\frac{\partial f}{\partial y}$

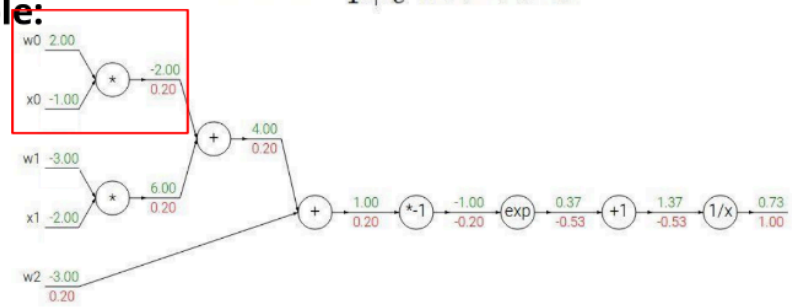
Chain rule:

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial y}$$

Upstream gradient — Local gradient

Another example:

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$

$f_a(x) = ax \rightarrow \frac{df}{dx} = a$

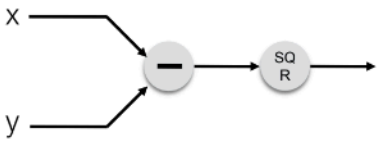
$f(x) = \frac{1}{x} \rightarrow \frac{df}{dx} = -1/x^2$

$f_c(x) = c + x \rightarrow \frac{df}{dx} = 1$

Computational Graphs

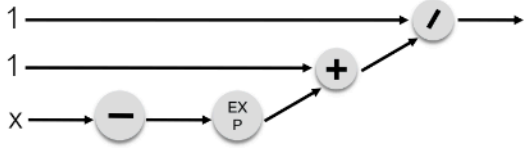
L2-loss

$L_2(x,y) = (x-y)^2$



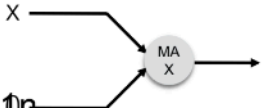
Sigmoid

$\text{sigmoid}(x) = \frac{1}{1+e^{-x}}$



ReLU

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



If the gradient becomes too big, then the SGD update step becomes too big:

$$\theta^{new} = \theta^{old} - \overbrace{\alpha}^{\text{learning rate}} \underbrace{\nabla_\theta J(\theta)}_{\text{gradient}}$$

This can cause bad updates: we take too large a step and reach a bad parameter configuration (with large loss)

In the worst case, this will result in Inf or NaN in your network (then you have to restart training from an earlier checkpoint)

Gradient clipping: if the norm of the gradient is greater than some threshold, scale it down before applying SGD update

Algorithm 1 Pseudo-code for norm clipping

$\hat{\mathbf{g}} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$
if $\|\hat{\mathbf{g}}\| \geq threshold$ **then**
  $\hat{\mathbf{g}} \leftarrow \frac{threshold}{\|\hat{\mathbf{g}}\|}\hat{\mathbf{g}}$
end if

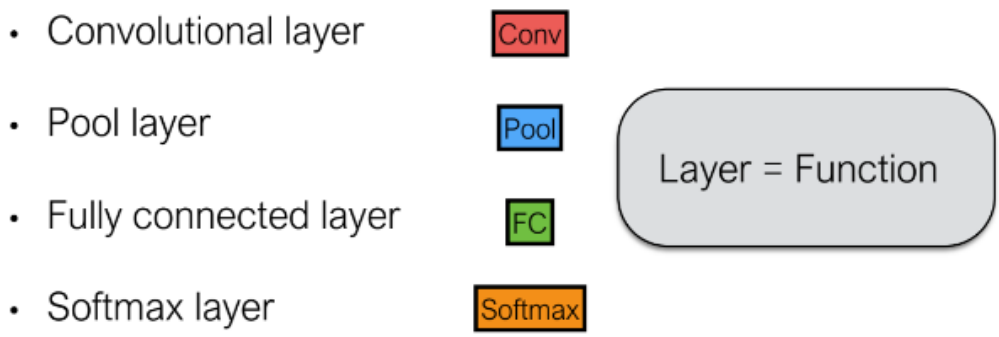
Intuition: take a step in the same direction, but a smaller step

Question: What are the limitations of backpropagation?

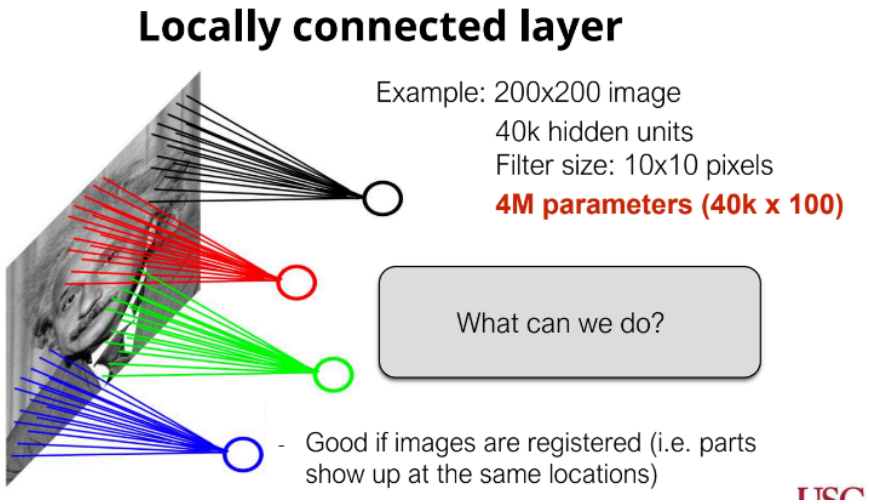
Answer:

- **Vanishing Gradient Problem**: In deep networks, gradients can become very small, effectively preventing the weights from being updated during training.
- **Exploding Gradient Problem**: Conversely, gradients can become excessively large, leading to divergent behavior.
- **Dependency on Data and Initial Weights**: The efficiency of backpropagation heavily relies on the quality of the data and the initial weight settings.
- **Local Minima**: Backpropagation can lead the network to converge to local minima rather than the global minimum.
- **Computational Intensity**: For large networks, backpropagation can be computationally expensive.

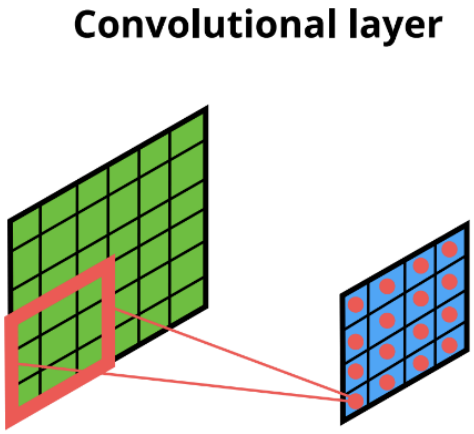
CNN



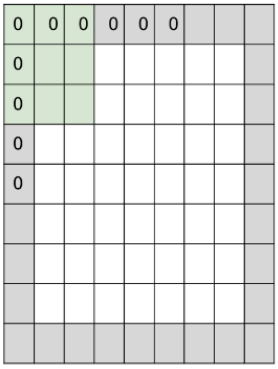
- Fully connected layer in a CNN combines the spatial information learned by earlier layers with global information to make a final decision or prediction about the input data, whether it's classifying objects in an image, detecting features in text, or any other task the CNN is designed for.



- Convolutional layer in a CNN extracts features from the input data by convolving it with a set of learnable filters. These filters detect patterns at different spatial locations in the input, enabling the network to learn hierarchical representations of the data.



In practice: Common to zero pad the border



e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output!
in general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with (F-1)/2. (will preserve size spatially)
e.g. F = 3 => zero pad with 1
F = 5 => zero pad with 2
F = 7 => zero pad with 3

USC

Let's assume input is $W_1 \times H_1 \times C$ Conv layer needs 4 hyperparameters:

- Number of filters **K**
- The filter size **F**
- The stride **S**
- The zero padding **P**

Common settings:
K = (powers of 2, e.g. 32, 64, 128, 512)
F = 3, S = 1, P = 1
- F = 5, S = 1, P = 2
- F = 5, S = 2, P = ? (whatever fits)
- F = 1, S = 1, P = 0

This will produce an output of $W_2 \times H_2 \times K$ where:

- $W_2 = (W_1 - F + 2P)/S + 1$
- $H_2 = (H_1 - F + 2P)/S + 1$

Number of parameters: F^2CK and K biases

Fully connected layer

Every neuron in a fully connected layer is connected to every neuron in the previous layer. This type of layer is often used toward **the end of neural network** architectures to make decisions based on the features extracted by previous layers.

It is **parameter-heavy** and does **NOT** account for the **spatial hierarchy** in the input data, making it less suitable for tasks like image recognition, where spatial relationships are key.

Convolutional layer

These layers apply a convolution operation to the input, effectively sliding a number of filters across the input data to produce feature maps.

This method **respects the spatial hierarchy of the data**, is **parameter-efficient** due to weight sharing, and is translation invariant. Convolutional layers are typically used in tasks involving images or time series data where such properties are advantageous.

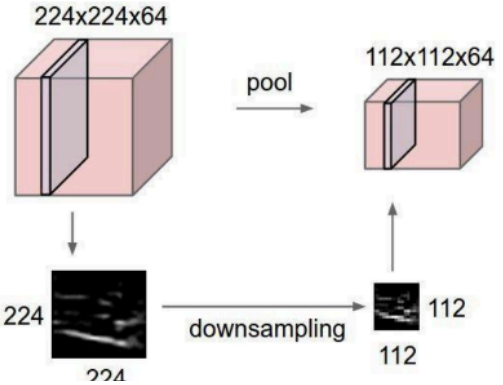
Locally connected layer

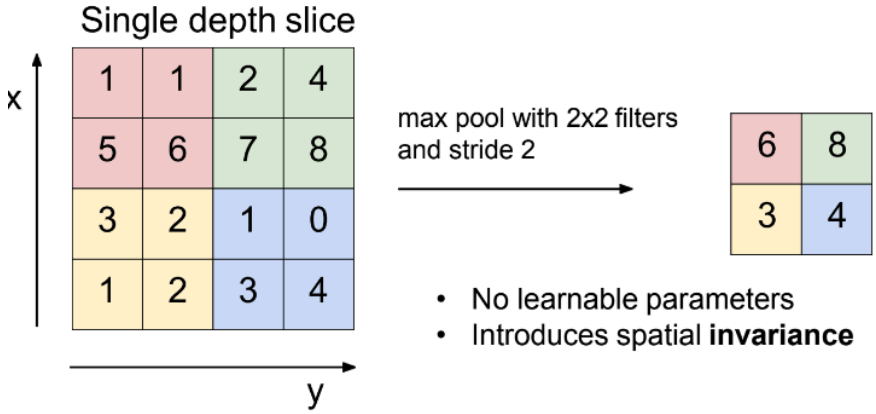
Each neuron is connected only to **a subset of the input data**, maintaining spatial relationships similar to convolutional layers.

However, unlike convolutional layers, **the weights are not shared across different spatial locations**. This allows the model to learn different features at different locations, but it can lead to **a large number of parameters**, making the network prone to overfitting and computationally expensive.

Pooling layer

- makes the representations smaller and more manageable
- operates over each activation map independently





- No learnable parameters
- Introduces spatial **invariance**

Slide credit: CS 231N

- Average Pooling
- L2 Norm Pooling

Let's assume input is W_1 x H_1 x C Pooling layer needs 2 hyperparameters:

- The spatial extent **F**
- The stride **S**

This will produce an output of W_2 x H_2 x C where:

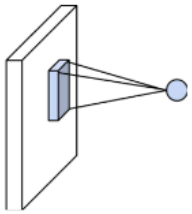
- $W_2 = (W_1 - F)/S + 1$
- $H_2 = (H_1 - F)/S + 1$

Number of parameters: 0 -> nothing to be learned

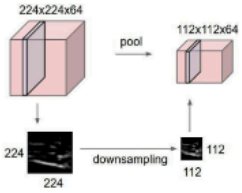
Softmax is implemented through a neural network layer just ***before*** the output layer. The Softmax layer must have the same number of nodes as the output layer.

The goal is to map the non-normalized output of a network to a **probability distribution** over predicted output classes.

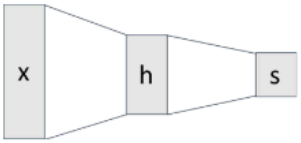
Convolution Layers



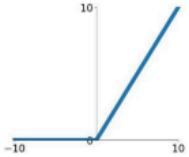
Pooling Layers



Fully-Connected Layers



Activation Function

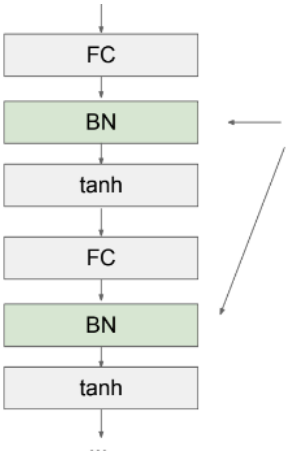


Normalization

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

USC

Batch Normalization normalizes the input data within each mini-batch, stabilizes training by reducing the likelihood of vanishing or exploding gradients, acts as a form of regularization, and can speed up the training process of neural networks.



Usually inserted after Fully Connected or Convolutional layers, and before nonlinearity.

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

- Makes deep networks **much** easier to train!
- Improves gradient flow
- Allows higher learning rates, faster convergence
- Networks become more robust to initialization
- Acts as regularization during training
- Zero overhead at test-time: can be fused with conv!
- Behaves differently during training and testing: this is a very common source of bugs!

Fact: Deep models have more representation power (more parameters) than shallower models.

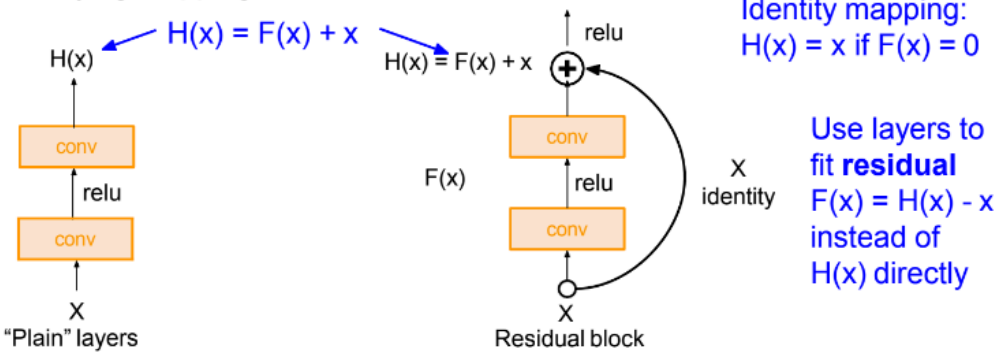
Hypothesis: the problem is an *optimization* problem, deeper models are harder to optimize

What should the deeper model learn to be at least as good as the shallower model?

A solution by construction is copying the learned layers from the shallower model and setting additional layers to identity mapping.

Resnet:

Solution: Use network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping



Identity mapping:
H(x) = x if F(x) = 0

Use layers to fit **residual** F(x) = H(x) - x instead of H(x) directly

SENNEt (Improvement over Resnet)

Add a "feature recalibration" module that learns to adaptively reweight feature maps
Global information (global avg. pooling layer) + 2 FC layers used to determine feature map weights
ILSVRC'17 classification winner (using ResNeXt-152 as a base architecture)

Case Studies

- AlexNet
- VGG
- GoogLeNet
- ResNet

Also....

- SENet
- Wide ResNet
- ResNeXT
- DenseNet
- MobileNets
- NASNet

AlexNet showed that you can use CNNs to train Computer Vision models.
ZFNet, **VGG** shows that bigger networks work better
GoogLeNet is one of the first to focus on efficiency using 1x1 bottleneck convolutions and global avg pool instead of FC layers
ResNet showed us how to train extremely deep networks
- Limited only by GPU & memory!
- Showed diminishing returns as networks got bigger

After ResNet: CNNs were better than the human metric and focus shifted to **Efficient networks**:
- Lots of tiny networks aimed at mobile devices: **MobileNet**, **ShuffleNet**

Neural Architecture Search can now automate architecture design

Many popular architectures are available in model zoos.
ResNets are currently good defaults to use.
Networks have gotten increasingly deep over time.
Many other aspects of network architectures are also continuously being investigated and improved.

CNN applications

Image classification

Object detection

Object segmentation

Image resolution

Human pose detection

Action classification

Image captioning

A heterogeneous graph is defined as

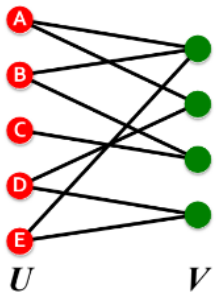
$$G = (V, E, R, T)$$

- Nodes with node types $v_i \in V$
- Edges with relation types $(v_i, r, v_m) \in E$
- Node type $T(v_i)$
- Relation type $r \in R$
- Nodes and edges have attributes/features

Bipartite graph is a graph whose nodes can be divided into two disjoint sets ***U*** and ***V*** such that every link connects a node in ***U*** to one in ***V***; that is, ***U*** and ***V*** are **independent sets**

Examples:

Authors-to-Papers (they authored)
Actors-to-Movies (they appeared in)
Users-to-Movies (they rated)
Recipes-to-Ingredients (they contain)



Node-Level Network Structure

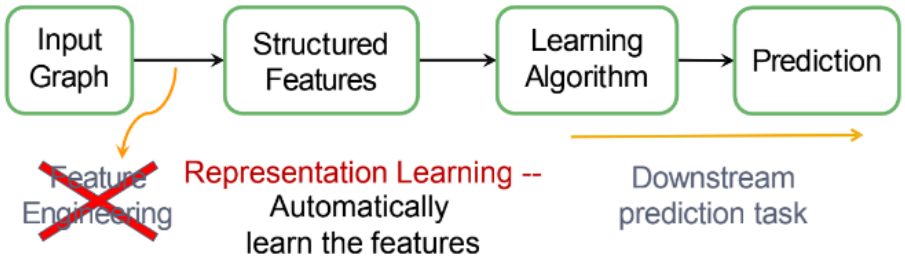
Goal: Characterize the structure and position of a node in the network:

- Node degree
- Node importance & position
  - E.g., Number of shortest paths passing through a node
  - E.g., Avg. shortest path length to other nodes
- Substructures around the node

Link-Level Prediction Task

- The task is to predict **new/missing/unknown links** based on the existing links.
- At test time, node pairs (with no existing links) are ranked, and top K node pairs are predicted.
- Task: Make a prediction for **a pair of nodes**.

Graph Representation Learning alleviates the need to do feature engineering every single time.



Goal is to encode nodes so that similarity in the embedding space (e.g., dot product) approximates similarity in the graph

Encoder + Decoder Framework

- Shallow encoder: Embedding lookup
- Parameters to optimize: $\mathbf{Z}$ which contains node embeddings $\mathbf{z}_u$ for all nodes $u \in V$
- We will cover deep encoders in the GNNs in 20 mins

- **Decoder:** based on node similarity.
- **Objective:** maximize $\mathbf{z}_v^{\mathrm{T}}\mathbf{z}_u$ for node pairs (u, v) that are **similar**

Loss function:

$$\min_{\Theta} \mathcal{L}(\boldsymbol{y}, f_{\Theta}(\boldsymbol{x}))$$

f can be a simple linear layer, an MLP, or other neural networks (e.g., a GNN later)

Sample a minibatch of input $\boldsymbol{x}$

Forward propagation: Compute $\mathcal{L}$ given $\boldsymbol{x}$

Back-propagation: Obtain gradient $\nabla_{\Theta}\mathcal{L}$ using a chain rule.

Use **stochastic gradient descent (SGD)** to optimize $\mathcal{L}$ for Θ over many iterations.

Key idea: Generate node embeddings based on **local network neighborhoods**

Intuition: Nodes aggregate information from their neighbors using neural networks

Basic approach: Average neighbor messages and apply a neural network

Model Design:

1. Define neighborhood aggregation function
2. Define loss function on embeddings
3. Train on a set of nodes
4. Generate embeddings for nodes
    a. The same aggregation parameters for all nodes

GNN vs. CNN

Convolutional neural network (CNN) layer with 3x3 filter:

- CNN can be seen as a special GNN with fixed neighbor size and ordering:
    - The size of the filter is pre-defined for a CNN.
    - The advantage of GNN is it processes arbitrary graphs with different degrees for each node.
- CNN is not permutation invariant/equivariant.
    - Switching the order of pixels leads to different outputs.

Key difference: We can learn different $W[^1]$ for different "neighbor" u for pixel v on the image. The reason is we can pick an order for the 9 neighbors using **relative position** to the center pixel: {(-1,-1). (-1,0), (-1, 1), ..., (1, 1)}

Transformer

A general definition of attention:

Given a set of vector *values*, and a vector *query*, attention is a technique to compute a weighted sum of the values, dependent on the query.

Each token/word has a **value vector** and a **query vector**. The value vector can be seen as the representation of the token/word. We use the query vector to calculate the attention score (weights in the weighted sum).

In this lecture, we introduced

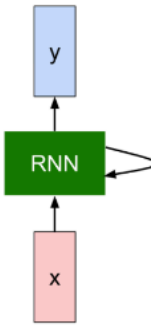
- Idea for Deep Learning for Graphs
  - Multiple layers of embedding transformation
  - At every layer, use the embedding at previous layer as the input
  - Aggregation of neighbors and self-embeddings
- Graph Convolutional Network
  - Mean aggregation (can be expressed in matrix form)
- GNN is a general architecture
  - CNN can be viewed as a special GNN

Recurrent Neural Network

We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:

$$h_t = f_W(h_{t-1}, x_t)$$

new state — some function with parameters W — old state — input vector at some time step



Note: the same function with the same parameters

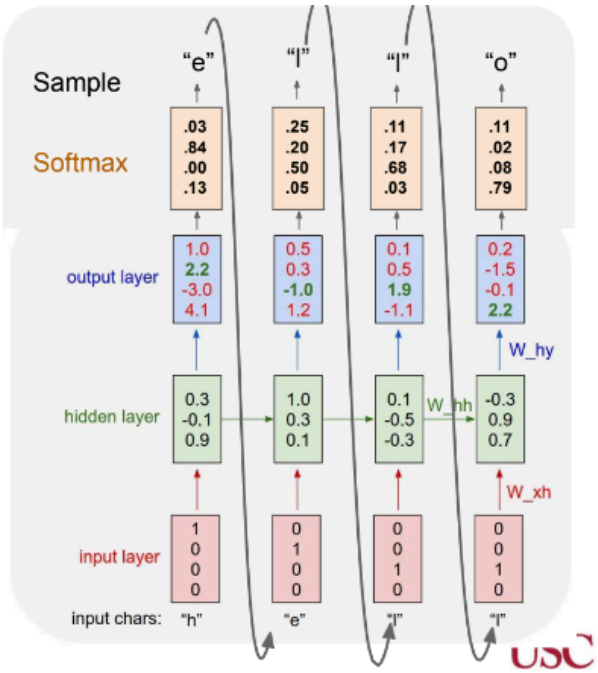
The Math: Deep Encoder

- **Basic approach:** Average neighbor messages and apply a neural network

Example: Character-level Language Model Sampling

Vocabulary:
[h,e,l,o]

At test-time sample characters one at a time, feed back to model



Learn **long-term** dependencies

- Remember information for long periods of time is practically their default behavior

Vanilla RNN

$$h_t = \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$

LSTM

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

e.g., $g = tanh(W_{gh} h_{t-1} + W_{gx} x_t)$

RNN applications

- Image Captioning
- Image Captioning with Attention
- Question Answering
- Visual Question Answering
- Speech Recognition
- Action Recognition in Videos
- Text Parsing
- Machine Translation

RNN & LSTM Summary

- RNNs deal with various sequential data or dependencies.
- LSTM is go-to models (rather than vanilla RNNs)
- RNNs are still under development (relative to CNNs)