**Terms -**

CVPR - Computer Vision and Pattern Recognition

FCN - Fully Convolutional Network

FPN - Feature Pyramid Network

**Precision:**

Precision (P) is defined as the number of true positives (Tp) over the number of true positives plus the number of false positives (Fp). Out of the predicted positives, it shows the percent of true positives.

**Recall** (R) is defined as the number of true positives (Tp) over the number of true positives plus the number of false negatives (Fn). Out of the actual positives, it shows the percent of true positives.

Recall is also called Sensitivity or True Positive Rate.

These quantities are also related to the (F1) score, which is defined as the harmonic mean of precision and recall.

Specificity is the total number of true negatives divided by the sum of the number of true negatives and false positives.

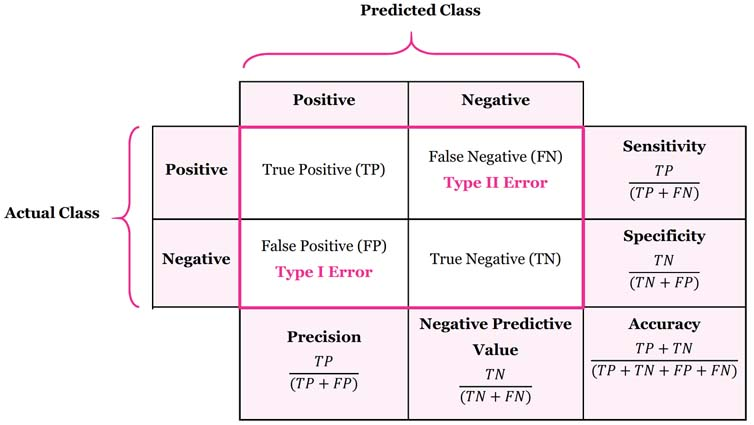
Specificity = TN/(TN+FP)

FPR = 1 – Specificity = FP/(TN+FP)

ROC curve is useful when the outcome is binary, it is a plot of FPR(x-axis) vs TPR(y-axis)

**K-fold validation:**

Cross validation is a resampling procedure to evaluate models on a limited dataset. We choose a suitable k such as 10, and divide the dataset into k groups. For each group, take the group as the validation set, and train on the remaining data. Evaluate the model on the training set and retain the evaluation score.



**Loss functions:**

Cross-Entropy loss -

Cross-entropy loss, or log loss, measures the performance of a classification model whose output is a probability value between 0 and 1.

Formula: −(ylog(p)+(1−y)log(1−p))

where y is the true label and p is the predicted probability,

when yt = 1, Entropy loss = -log(p) ⇒ 0 if p == 1 and inf. if p == 0,

when yt = 0, Entropy loss = -log(1-p) ⇒ 0 if p == 0 and inf. if p == 1,

When there are more than 2 classes like multi-class classification, we calculate loss for each label separately and take the summation.

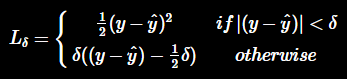
Entropy loss = Sum over all labels ( -y\_label\*log(p\_label))

Hinge loss:

**def** Hinge(yHat, y):

**return** np.max(0, y - (1-2\*y)\*yHat)

Huber loss: It is used for regression, and it is less sensitive to outliers as it gives more importance to losses inside the interval.



**def** Huber(yHat, y, delta=1.):

**return** np.where(np.abs(y-yHat) < delta,.5\*(y-yHat)\*\*2 , delta\*(np.abs(y-yHat)-0.5\*delta))

Kullback Leibler:

Minimizing KL divergence corresponds to minimizing the cross entropy between the distributions.

**def** KLDivergence(yHat, y):

*"""*

*:param yHat:*

*:param y:*

*:return: KLDiv(yHat || y)*

*"""*

**return** np.sum(yHat \* np.log((yHat / y)))

Mean Absolute Error (MAE) or L1 loss:

**def** L1(yHat, y):

**return** np.sum(np.absolute(yHat - y)) / y.size

Mean Squared Error (MSE) or L2 loss:

**def** MSE(yHat, y):

**return** np.sum((yHat - y)\*\*2) / y.size

Sigmoid = 1/(1 + exp(-x))

Log-sigmoid = log(1/(1+e^(-x)) to some base b,

= – ln(1+e^(-x)) / ln(b)

Unlike sigmoid, log of sigmoid produces outputs in scale of (-∞, 0]

Negative Log Likelihood Loss = -log(p) for all classes where y\_true = 1

p for each class is obtained from softmax.

Jaccard loss = IOU loss.

Dice loss = 2\*(AB)/(A+B), It is always greater than Jaccard loss.

Log likelihood loss

Xavier Initialization

**Bootstrapping:**

**Oversampling and Undersampling:**

The bootstrap method is a resampling technique used to estimate statistics on a population by sampling a dataset with replacement. The training of the model on the sample and evaluation is done on those samples not included in the sample. These samples not included in a given sample are called the out-of-bag samples, or OOB for short.

**SMOTE (Synthetic Minority Oversampling Technique**):

One approach to addressing imbalanced datasets is to oversample the minority class. The simplest approach involves duplicating examples in the minority class, although these examples don’t add any new information to the model. Instead, new examples can be synthesized from the existing examples. This is a type of [data augmentation](https://machinelearningmastery.com/how-to-configure-image-data-augmentation-when-training-deep-learning-neural-networks/) for the minority class and is referred to as the **Synthetic Minority Oversampling Technique**, or **SMOTE** for short.

SMOTE works by selecting examples that are close in the feature space, drawing a line between the examples in the feature space and drawing a new sample at a point along that line.

Specifically, a random example from the minority class is first chosen. Then *k* of the nearest neighbors for that example are found (typically *k=5*). A randomly selected neighbor is chosen and a synthetic example is created at a randomly selected point between the two examples in feature space.

**Gradient Boosting**

Boosting is a technique to convert weak learners into strong learners. Each new tree is fit on a modified version of the original dataset. In AdaBoost, the decision tree starts training with equal weight for each observation. Then we increase the weights of the incorrectly classified observations and lower the weights of the correctly classified samples. The second tree is trained on the weighted data. We can then compute the classification error on the ensemble and train a new tree.

Predictions of the final ensemble is the weighted sum of predictions made by the previous tree models.

Gradient Boosting trains many models in a gradual, additive and sequential manner. As compared to AdaBoost, Gradient boosting uses gradients of the loss function to boost the weak learner. The loss fn. is a measure of how good the model is fitting the data.

For classification ,we can use logarithmic loss function and for regression, we can minimize the squared error.

**XGBoost**

XGBoost is an optimized distributed gradient boosting library designed to be highly efficient, flexible and portable. XGBoost library can use parallelization, distributed computing and cache optimization. The execution and model performance is high.

**AdaBoost:**

**Fuzzy C means (FCM) or soft clustering**

**Fuzzy clustering** (also referred to as **soft clustering** or **soft *k*-means**) is a form of clustering in which each [data point](https://en.wikipedia.org/wiki/Data_point) can belong to more than one cluster.

**Singular Value Decomposition (SVD)**

SVD can be used in Matrix completion, dimension reduction of big and sparse dataset etc. It is a generalization of the Fourier Transform (FFT). It is the basis for Linear regression or PCA. It is simple and scalable, so it can be performed in large datasets.

SVD decomposes a matrix into 3 matrices,

A = U\*Sigma\*V\_transpose

U and V are orthogonal matrices, so U\*U\_transpose = 1,

U\_transpose = U\_inverse

Each row has a length 1, and the rows are mutually perpendicular.

Sigma is a diagonal matrix, so all the entries outside the diagonal are zero.

In SVD, we assume that the features are aligned for all rows/data points.

**Robust PCA (RPCA, 2011)**

It is robust to outliers compared to normal PCA. To do this, we split the original matrix into a true matrix (low rank matrix) and a sparse matrix containing outliers.

X = L + S

t-SNE:

**Hierarchical clustering**

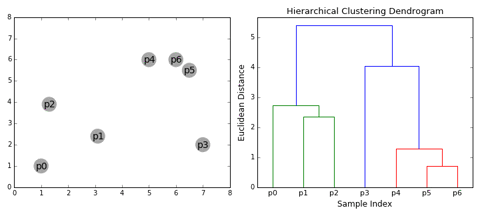
When we have a set of data points and distances between them, the points are grouped into clusters. Initially, each point is treated as a separate cluster, and we merge the points/clusters together based on the following criteria,

We identify the data points that are closest to each other and merge them into a cluster. We merge two similar clusters.

The distance between clusters can be defined in the following ways,

1. Complete Linkage - It is the longest distance between two points in each cluster
2. Single Linkage - It is the shortest distance between two points in each cluster
3. Average Linkage - It is the average distance between points in each cluster
4. Centroid Linkage - It is the distance between the centroids in each cluster

The process is iterated until we merge all points to a single cluster. The output of Hierarchical clustering is a Dendrogram which shows the relationship between clusters/points.



We can set a distance threshold for drawing a cutting line and prevent further agglomeration.

Code:

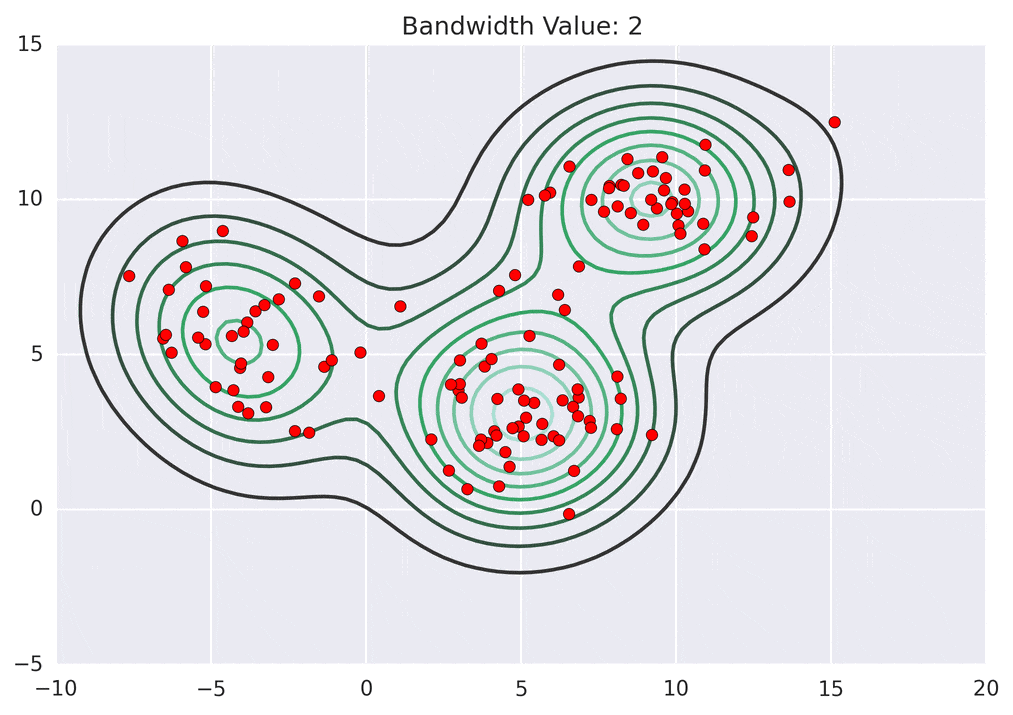
import scipy.cluster.hierarchy as sch

dendrogram = sch.dendrogram(sch.linkage(X, method = 'ward'))

This type of merging the clusters is known as Agglomerative Hierarchical Clustering, we can also start with a single cluster of all points and divide them. This is known as Divisive Hierarchical Clustering.

Mean Shift Clustering

It is based on Kernel Density Estimation (KDE). We place a kernel in each point (the most commonly used cluster is the gaussian kernel), the points are made to climb uphill to the nearest peak on the KDE surface. The points which end up in the same hill are placed in the same cluster.



The KDE plot will be different depending on the bandwidth used. Smaller bandwidths will lead to sharp kernels and large number of clusters. Larger bandwidth will lead to a widely spread KDE surface with shorter peaks.

**Density-Based Spatial Clustering of Applications with Noise (DBSCAN)**

We choose core points having many other data points in their vicinity (within a distance e) and form a cluster. We then grow the cluster if the points are reachable from the core points. These points are called edge points, and we don’t extend the cluster beyond that.

The points which are neither core nor reachable are called noise points, and they don’t belong to any cluster.

Expectation Maximization using Gaussian Mixture Models (GMM):

**Factorization Machines (FM, 2010)**

It is a generalized supervised learning algorithm which can be used for classification and regression. It is an extension of the Linear model that is designed to capture interactions between features in high dimensional sparse datasets such as click prediction and item recommendation.



The global bias and the linear term is same as the linear model. The pairwise feature interactions are modeled by the 3rd term. Vi and Vj are vectors with k dimensions where k is a hyperparameter, usually between 50 to 100.

If a pair of features tend to occur more often in positive samples then the inner product term would be large. The embedding vectors would be close to each other in cosine similarity.

The dataset is built based on users, content of the recommended ad/item, context of the item. Not all user behavior is important, and it’s best to focus on frequently logged-in users. Context of the content wrt the user can be if the user liked the content, commented on it or shared it, the keywords the content has etc.

**Field aware Factorization Machines (FFM, 2012):**

In FM, we have a single latent vector for each feature, but in FFM, we have multiple latent vectors for each feature. So, the latent vectors are field dependent.

DeepFM:

<https://arxiv.org/pdf/1703.04247.pdf>

**Collaborative Filtering (Matrix Factorization)**

It is used for recommendation systems, where users are recommended a set of products based on a score.

The data is organized into a matrix where users are the rows and the items are in columns, and the values of the matrix are ratings of the user. There will be a lot of missing values which has to be predicted based on the available values.

In content filtering, we use the information about people and items to predict if a user will like an item. This is also known as Matrix completion problem. We define a set of features (like comedy or action for movie items) and map it to users and items. This will lead to 2 matrices, one for users and features, and the other for features and items. We can multiply the matrices and get the scores for each user and movie. The drawback is that it is very simplistic and not accurate. We also need to collect values for the features manually.

In collaborative filtering, we generate a set of features from the incomplete user-movie data based on the patterns in the data. It is a reverse approach to content filtering.

The user-item matrix is split into 2 matrices which can then be multiplied to output the missing values.

Mean normalization: When we have a new user who has not rated any items, we can assign the mean values of each item wrt. to all users.

**Topic Modelling**

Topic models are commonly used to produce topics from corpuses that (1) coherently encapsulate semantic meaning and (2) describe documents well. As such, topic models aim to minimize perplexity and maximize topic coherence.

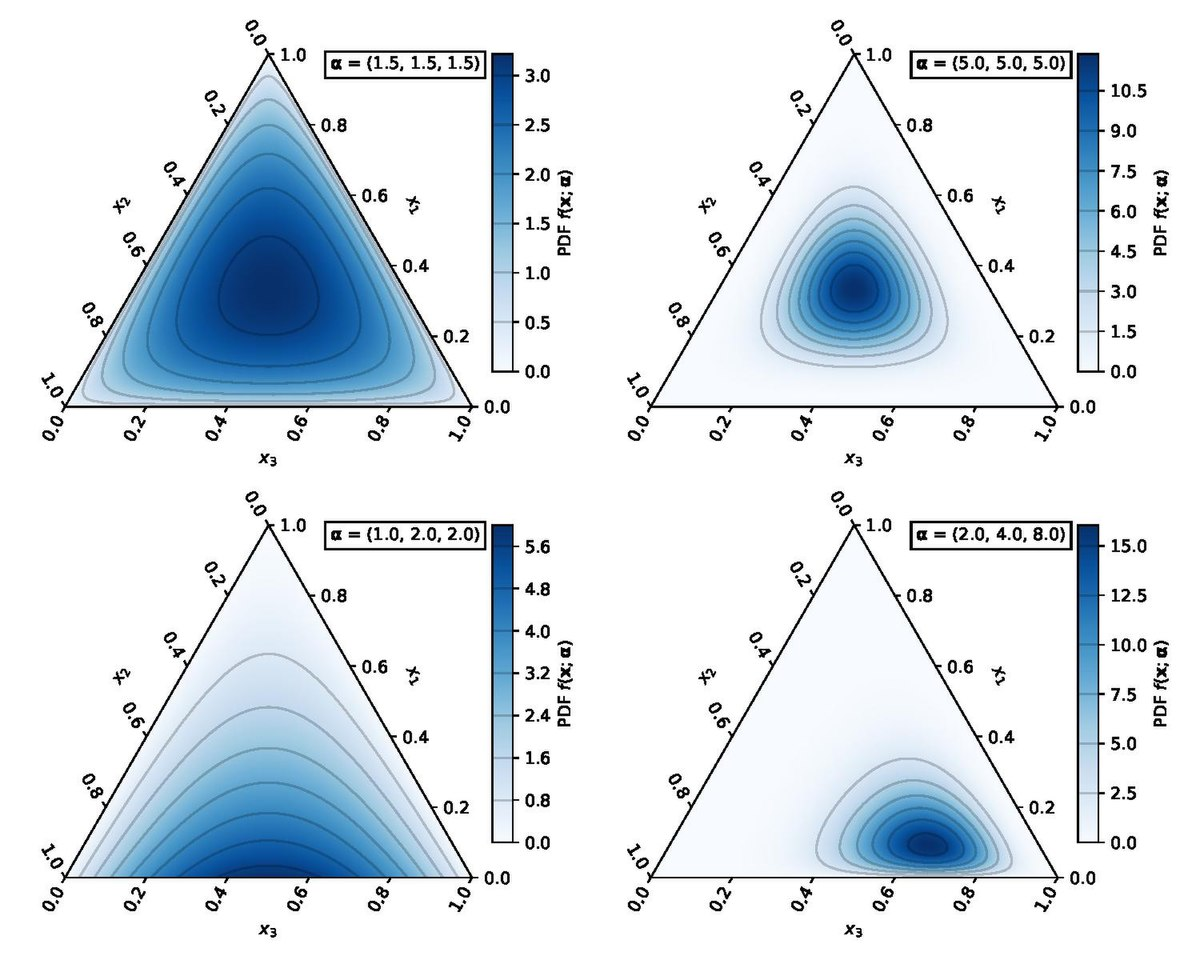
**Perplexity** is an intrinsic language modeling evaluation metric that measures the inverse of the geometric mean per-word likelihood in your test data. A lower perplexity score indicates better generalization performance.

**Topic coherence** is often defined as the average or median of the pairwise word-similarity scores of the words in that topic. It is computed to the top N words for that particular topic from your model.

There is a tradeoff between perplexity and topic coherence. LDA trained with collapsed Gibbs sampling achieves better perplexity. While NTM is promising for achieving high topic coherence.

**Latent Dirichlet Allocation (LDA)**: It is an unsupervised algorithm to group documents into categories. It is used to sort the documents into topics based on the content. It is a "bag-of-words" model, which means that the order of words does not matter.

LDA generates a fake document using Dirichlet and multinomial distributions. Dirichlet distribution is a multivariate generalization of Beta distribution, and it is parameterized by a positive vector alpha. For LDA, alpha values would be less than one.



We have to specify the number of topics beforehand, and LDA creates 2 distributions, one where the topics are present on the vertices and the words are placed inside the triangle (n-d tetrahedron) based on the content, and the other has words on the vertices and the topics inside. This indicates the probability that a word belongs to a topic, and the probability that a topic contains the word.

To create this distribution we use Gibbs sampling. Initially we assign a random topic to each word in all documents and we iterate through each word and change its topic to maximize the probability of a topic representing the document and the topic representing the word.

This is done by calculating the probability of the topic in that document (words containing that topic/ all words) and multiplying it with the probability of the topic for that word across all documents, for all topics and choosing the topic which maximizes this probability.

After several iterations we can assign the probability score of all topics for each document.

**Term Frequency-Inverse Document Frequency (TF-IDF)**

It is used to quantify a word among a set of documents, by assigning it a weight which describes the importance of that word in that document. It is a widely used technique in Information retrieval and Text mining.

Term frequency measures the frequency of the word in that document.

TF(t,d) = count of t in d / number of words in d

Document frequency measures the frequency of the word in all documents.

DF(t) = occurrence of t in all documents

It will be more for frequently occuring words such as “is”, “and” etc.

Inverse document frequency measures the informativeness of the word in documents.

IDF(t) = log(N/(DF + 1)) for document set N

TF-IDF is the product of term frequency and inverse document frequency.

TF-IDF(t, d) = TF(t, d) \* log(N/(DF + 1))

**Skip gram and Continuous Bag of Words (CBOW)**

Word2Vec is a technique where we use a neural network to find embeddings of words in the form of vectors. Words with similar contexts will have similar embeddings.

There are two unsupervised algorithms for generating Word2Vec, Skip gram and CBOW.

In CBOW, the model is made to predict the target word from context.

“Ask for \_\_\_\_ when in need”

In Skip gram, we predict the context words for a given target word. So we input a word, So we create fake data of the neighbouring words within a window size from the sentences containing the input word and train the model. All neighbouring words are treated equally during the training. The window size is usually 5-10.

**Subword embedding (FastText model)**

In English, words such as “helps”, “helped”, and “helping” are inflected forms of the same word “help”. In both the skip-gram model and the continuous bag-of-words model, different inflected forms of the same word are directly represented by different vectors without shared parameters.

Instead of learning word-level vector representations, fastText can be considered as the subword-level skip-gram, where each center word is represented by the sum of its subword vectors. For a given word, we add ‘<’ and ‘>’ to the start and end of the word and create subwords for a given size n.

For a word ‘Where’ and when n=3 , we obtain all subwords of length 3: “<wh”, “whe”, “her”, “ere”, “re>”, and the special subword “<where>”.

In fastText, all the extracted subwords have to be of the specified lengths, such as 3 to 6. The rest of fastText is the same as the skip-gram model. Compared with the skip-gram model, the vocabulary in fastText is larger, resulting in more model parameters.

**Batch size and learning rate**

Bigger batch size leads to good results and stable convergence. Bigger batch size should usually be combined with large learning rates for faster convergence. Reducing batch size leads to more noise and it results in a broader local minima, and bigger batch size in smooth convergence to a deep local minima.

Higher learning rates and lower batch sizes can prevent our models from getting stuck in deep, narrow minima. Decreasing the learning rate would prevent overshooting the global loss function minimum.

**Convolution**

2D Convolution:

Zero Padding:

テキスト

自動的に生成された説明

Step 1: Matrix inversion

Step 2: Slide the kernel over the image and perform MAC operation at each instant

• 2-D discrete convolutions (N = 2),

• square inputs (i1 = i2 = i),

• square kernel size (k1 = k2 = k),

• same strides along both axes (s1 = s2 = s),

• same zero padding along both axes (p1 = p2 = p)

時計と文字の加工写真

低い精度で自動的に生成された説明

2D convolution using a kernel size of 3, stride of 1 and padding

図形

自動的に生成された説明

2D convolution with no padding, stride of 2 and kernel of 3

黒い背景と白い文字のロゴ

低い精度で自動的に生成された説明

A (half) padded convolution will keep the spatial output dimensions equal to the input.

The needed parameters for such a layer can be calculated by I\*O\*K

**Transposed convolution:**

Transposed 2D convolution with no padding, stride of 2 and kernel of 3

図形 が含まれている画像

自動的に生成された説明

The need for transposed convolutions generally arises from the desire to use a transformation going in the opposite direction of a normal convolution, i.e., from something that has the shape of the output of some convolution to something that has the shape of its input while maintaining a connectivity pattern that is compatible with said convolution.

テキスト, 手紙

自動的に生成された説明

**Up Sampling:**

The Upsampling layer is a simple layer with no weights that will double the dimensions of input and can be used in a generative model when followed by a traditional convolutional layer.

テーブル が含まれている画像

自動的に生成された説明

**Dilated Convolution (a.k.a. atrous convolutions):**

2D convolution using a 3 kernel with a dilation rate of 2 and no padding

黒い背景と白い文字のロゴ

低い精度で自動的に生成された説明

Dilated convolutions “inflate” the kernel by inserting spaces between the kernel elements. The dilation “rate” is controlled by an additional hyperparameter

d. Implementations may vary, but there are usually d−1 spaces inserted between

kernel elements such that d = 1 corresponds to a regular convolution.

Dilated convolutions are used to cheaply increase the receptive field of output units without increasing the kernel size, which is especially effective when multiple dilated convolutions are stacked one after another.

A kernel of size k dilated by a factor d has an effective size k’ = k + (k − 1)(d − 1).

テーブル

中程度の精度で自動的に生成された説明

**1D Convolution:**

Conv1D is widely applied on sensory data, and accelerometer data is one of it.

テーブル

自動的に生成された説明

**3D Convolution:**

In Conv3D, the kernel slides in 3 dimensions,

ダイアグラム

自動的に生成された説明

Conv3D is mostly used with 3D image data. Such as Magnetic Resonance Imaging (MRI) data. MRI data is widely used for examining the brain, spinal cords, internal organs and many more. A Computerized Tomography (CT) Scan is also an example of 3D data, which is created by combining a series of X-rays image taken from different angles around the body. We can use Conv3D to classify this medical data or extract features from it.

**Resnet Architecture:**

Deep networks are hard to train because of the vanishing gradient problem — as the gradient is back-propagated to earlier layers, repeated multiplication may make the gradient infinitively small. As a result, as the network goes deeper, its performance gets saturated or even starts degrading rapidly.

ダイアグラム

自動的に生成された説明

ダイアグラム, 箱ひげ図

自動的に生成された説明

ダイアグラム

自動的に生成された説明

**ResNeXt Architecture:**

Xie et al. proposed a variant of ResNet that is codenamed ResNeXt

ダイアグラム

自動的に生成された説明

it is very similar to the Inception module, they both follow the split-transform-merge paradigm, except in this variant, the outputs of different paths are merged by adding them together, while in Inception-Net they are depth-concatenated.

**DenseNet Architecture:**

Huang et al. [9] proposed a novel architecture called DenseNet that further exploits the effects of shortcut connections — it connects all layers directly with each other. In this novel architecture, the input of each layer consists of the feature maps of all earlier layer, and its output is passed to each subsequent layer. The feature maps are aggregated with depth-concatenation.

ダイアグラム, 設計図

自動的に生成された説明

Other than tackling the vanishing gradients problem, the authors argue that this architecture also encourages feature reuse, making the network highly parameter-efficient.

**Inception Net Architecture:**

**Inceptionv1:**

It performs convolution on an input, with 3 different sizes of filters (1x1, 3x3, 5x5). Additionally, max pooling is also performed. The outputs are concatenated and sent to the next inception module.

ダイアグラム

自動的に生成された説明

Deep neural networks are computationally expensive. To make it cheaper, the authors limit the number of input channels by adding an extra 1x1 convolution before the 3x3 and 5x5 convolutions. Though adding an extra operation may seem counterintuitive, 1x1 convolutions are far cheaper than 5x5 convolutions, and the reduced number of input channels also help. Do note that however, the 1x1 convolution is introduced after the max pooling layer, rather than before.

ダイアグラム

自動的に生成された説明

Further, 5x5 convolutions can be reduced to two 3x3 convolutions thus reducing the number of parameters.

**Graph Convolution Network:**

The major difference between CNNs and GNNs is that CNNs are specially built to operate on regular (Euclidean) structured data, while GNNs are the generalized version of CNNs where the numbers of nodes connections vary and the nodes are unordered (irregular on non-Euclidean structured data).

GCNs themselves can be categorized into 2 major algorithms, **Spatial Graph Convolutional Networks** and **Spectral Graph Convolutional Network**s.

Review: <https://arxiv.org/pdf/1901.00596.pdf>

Graph G=(V,E)

For these models, the goal is then to learn a function of signals/features on a graph G=(V,E) which takes as input:

* A feature description xi for every node i; summarized in a N×D feature matrix X (N: number of nodes, D: number of input features)
* A representative description of the graph structure in matrix form; typically, in the form of an adjacency matrix A (or some function thereof)

and produces a node-level output Z (an N×F feature matrix, where F is the number of output features per node). Graph-level outputs can be modeled by introducing some form of pooling operation (see, e.g. Duvenaud et al., NIPS 2015).

Every neural network layer can then be written as a non-linear function

テキスト

自動的に生成された説明

with H(0)=X and H(L)=Z (or z for graph-level outputs), L being the number of layers. The specific models then differ only in how f(⋅,⋅) is chosen and parameterized.

GCNs Part II: A simple example

As an example, let's consider the following very simple form of a layer-wise propagation rule:

テキスト

中程度の精度で自動的に生成された説明

where W(l) is a weight matrix for the l-th neural network layer and σ(⋅) is a non-linear activation function like the ReLU. Despite its simplicity this model is already quite powerful (we'll come to that in a moment).

But first, let us address two limitations of this simple model: multiplication with A means that, for every node, we sum up all the feature vectors of all neighboring nodes but not the node itself (unless there are self-loops in the graph). We can "fix" this by enforcing self-loops in the graph: we simply add the identity matrix to A.

The second major limitation is that A is typically not normalized and therefore the multiplication with A will completely change the scale of the feature vectors (we can understand that by looking at the eigenvalues of A). Normalizing A such that all rows sum to one, i.e. D−1A, where D is the diagonal node degree matrix, gets rid of this problem. Multiplying with D−1A now corresponds to taking the average of neighboring node features. In practice, dynamics get more interesting when we use a symmetric normalization, i.e. D−12AD−12 (as this no longer amounts to mere averaging of neighboring nodes). Combining these two tricks, we essentially arrive at the propagation rule introduced in Kipf & Welling (ICLR 2017):

テキスト が含まれている画像

自動的に生成された説明

テキスト が含まれている画像

自動的に生成された説明

where I is the identity matrix and D^ is the diagonal node degree matrix of A^.

Reference: <https://tkipf.github.io/graph-convolutional-networks/>

**Object Detection:**

**Yolo:**

Yolov3: <https://arxiv.org/pdf/1804.02767.pdf>

Yolov2: <https://arxiv.org/pdf/1612.08242.pdf>

Yolo: <https://arxiv.org/pdf/1506.02640.pdf>

In Yolo, A single convolutional network simultaneously predicts multiple bounding boxes and class probabilities for those boxes.

YOLO trains on full images and directly optimizes detection performance

YOLO reasons globally about the image 2016 making predictions. Unlike sliding window and region proposal-based techniques, YOLO sees the entire image during training and test-time so it implicitly encodes contextual information about classes as well as their appearance.

YOLO learns generalizable representations of objects.

The image is divided into SxS grid, and each grid predicts B bounding boxes,

For each box, the model predicts x, y, w, h and confidence.

x and y are relative to the grid, and w and h are relative to the image.

Confidence is measured as P(object)\*IOU

P(object) is the probability of the presence of object in the image. If there is no object then the probability and confidence becomes 0.

Each grid cell also predicts the conditional probability of each class given an object, P(Ci|object) for all classes.

The output of the model becomes SxSx(B\*5+C)

Architecture:

ダイアグラム, 設計図

自動的に生成された説明

A linear activation function is used for the final layer and all other layers use the following leaky rectified linear activation. Since many grids don’t have any object so to remedy the imbalance, we increase the loss from bounding box

coordinate predictions and decrease the loss from confidence predictions for boxes that don’t contain objects. We use two parameters, λcoord and λnoobj to accomplish this. We

set λcoord = 5 and λnoobj = .5.

テキスト, 手紙

自動的に生成された説明

**Yolov2:**

Batch Normalization is added to the architecture. Batch normalization leads to significant improvements in convergence while eliminating the need for other forms of regularization.

Convolutional with Anchor Boxes:

We remove the fully connected layers from YOLO and use anchor boxes to predict bounding boxes. First, we eliminate one pooling layer to make the output of the network’s convolutional layers higher resolution. We also shrink the network to operate on 416 input images instead of 448×448. We do this because we want an odd number of locations in our feature map so there is a single center cell. Using anchor boxes, we get a small decrease in accuracy, but the recall increases.

Bounding Box priors are chosen from k-means clustering, with distance metric,

d(box, centroid) = 1 − IOU(box, centroid)

テキスト, 手紙

自動的に生成された説明

ダイアグラム

自動的に生成された説明

YOLOv3 predicts an objectness score for each bounding box using logistic regression. This should be 1 if the bounding box prior overlaps a ground truth object by more than any other bounding box prior.

Each box predicts the classes the bounding box may contain using multilabel classification. We do not use a softmax as we have found it is unnecessary for good performance, instead we simply use independent logistic classifiers. During training we use binary cross-entropy loss for the class predictions.

YOLOv3 predicts boxes at 3 different scales. we predict 3 boxes at each scale so the tensor is N × N × [3 ∗ (4 + 1 + 80)] for the 4 bounding box offsets, 1 objectness prediction, and 80 class predictions.

We still use k-means clustering to determine our bounding box priors. We just sort of chose 9 clusters and 3 scales arbitrarily and then divide up the clusters evenly across scales. On the COCO dataset the 9 clusters were: (10×13),(16×30),(33×23),(30×61),(62×4).

**Faster RCNN:**

Fast RCNN Paper: <https://arxiv.org/pdf/1504.08083.pdf>

RCNN: Region based CNN

RPN: Region Proposal Networks

Faster R-CNN, is composed of two modules. The first module is a deep fully convolutional network that proposes regions, and the second module is the Fast R-CNN detector [2] that uses the proposed regions.

A Region Proposal Network (RPN) takes an image (of any size) as input and outputs a set of rectangular object proposals, each with an object score. RPN is fully convolutional and trains end-to-end to classify the proposal regions into object categories or background.

R-CNN mainly plays as a classifier, and it does not predict object bounds.

RPN is then unified with Fast RCNN to get the output boxes.

RPNs with Fast R-CNNs produce detection accuracy better than the strong baseline of Selective Search with Fast R-CNNs.

To generate region proposals, we slide a small network over the convolutional feature map output by the last shared convolutional layer. This small network takes as input an n × n spatial window of the input convolutional feature map. Each sliding window is mapped to a lower-dimensional feature (256-d for ZF and 512-d for VGG, with ReLU [33] following). This feature is fed into two sibling fully-connected layers—a box-regression layer (reg) and a box-classification layer (cls).

We use n = 3 in this paper, noting that the effective receptive field on the input image is large (171 and 228 pixels for ZF and VGG, respectively). This architecture is naturally implemented with an n×n convolutional layer followed by two sibling 1 × 1 convolutional layers (for reg and cls, respectively).

The fully-connected layers are shared across all spatial locations.

ダイアグラム

自動的に生成された説明

At each sliding-window location, we simultaneously predict multiple region proposals, where the number of maximum possible proposals for each location is denoted as k. So the reg layer has 4k outputs encoding the coordinates of k boxes, and the cls layer outputs 2k scores that estimate probability of object or not object for each proposal.

ダイアグラム

自動的に生成された説明

The k proposals are parameterized relative to k reference boxes, or anchors. An anchor is centered at the sliding window in question, and is associated with a scale and aspect ratio (Figure 3, left). By default we use 3 scales and 3 aspect ratios, yielding k = 9 anchors at each sliding position. For a convolutional feature map of a size W × H (typically ∼2,400), there are W\*H\*k anchors in total.

The model classifies and regresses bounding boxes with reference to anchor boxes of multiple scales and aspect ratios.

Training:

For training RPNs, we assign a binary class label (of being an object or not) to each anchor. We assign a positive label to two kinds of anchors: (i) the anchor/anchors with the highest Intersection-overUnion (IoU) overlap with a ground-truth box, or (ii) an anchor that has an IoU overlap higher than 0.7 with any ground-truth box. We assign a negative label to a non-positive anchor if its IoU ratio is lower than 0.3 for all ground-truth boxes.

Loss Function:

テキスト, 手紙

自動的に生成された説明

The two terms are normalized by Ncls and Nreg and weighted by a balancing parameter λ.

Fast RCNN:

A Fast R-CNN network takes as input an entire image and a set of object proposals. The network first processes the whole image with several convolutional (conv) and max pooling layers to produce a conv feature map. Then, for each object proposal a region of interest (RoI) pooling layer extracts a fixed-length feature vector from the feature map. Each feature vector is fed into a sequence of fully connected (fc) layers that finally branch into two sibling output layers, classification layer and bounding-box regression layer: one that produces softmax probability estimates over K object classes plus a catch-all “background” class and another layer that outputs four real-valued numbers for each of the K object classes. Each set of 4 values encodes refined bounding-box positions.

Paper: <https://arxiv.org/pdf/1506.01497.pdf>

**PAFNet: Paddle Anchor Free Network**

Large amounts of anchors hamper the generalization ability of detector, and increase the amount of computation and memory significantly. Anchor-free detectors are proposed to address these issues by removing pre-defined anchors and regressing the locations directly, which can achieve higher efficiency.

ダイアグラム

自動的に生成された説明

Paper: <https://arxiv.org/pdf/2104.13534.pdf>

**Semantic Segmentation:**

**Mask RCNN:**

Mask R-CNN extends Faster R-CNN by adding a branch for predicting binary segmentation masks on each Region of Interest (RoI) in parallel with the existing branch for classification and bounding box regression.

Instance segmentation is challenging because it requires the correct detection of all objects in an image while also precisely segmenting each instance. It therefore combines elements from the classical computer vision tasks of object detection, where the goal is to classify individual objects and localize each using a bounding box, and semantic segmentation, where the goal is to classify each pixel into a fixed set of categories without differentiating object instances.

The mask branch is a small Fully Convolutional Network (FCN) applied to each RoI, predicting a segmentation mask in a pixel-to-pixel manner. Faster R-CNN has two outputs for each candidate object, a class label and a bounding-box offset; to this we add a third branch that outputs the object mask.

Loss:

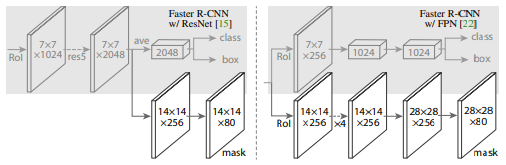
During training, we define a multi-task loss on each sampled RoI as L = Lcls + Lbox + Lmask. The mask branch has a Km2 - dimensional output for each RoI, which encodes K binary masks of resolution m × m, one for each of the K classes. To this we apply a per-pixel sigmoid, and define Lmask as the average binary cross-entropy loss. For an RoI associated with ground-truth class k, Lmask is only defined on the k-th mask (other mask outputs do not contribute to the loss).

Lmask allows the network to generate masks for every class without competition among classes; we rely on the dedicated classification branch to predict the class label used to select the output mask. This decouples mask and class prediction.

RoIAlign: RoIPool [9] is a standard operation for extracting a small feature map (e.g., 7×7) from each RoI. RoIAlign layer removes the harsh quantization of RoIPool, properly aligning the extracted features with the input. we avoid any quantization of the RoI boundaries or bins (i.e., we use x/16 instead of [x/16]). We use bilinear interpolation to compute the exact values of the input features at four regularly sampled locations in each RoI bin, and aggregate the result (using max or average).

Training:

As in Fast R-CNN, an RoI is considered positive if it has IoU with a ground-truth box of at least 0.5 and negative otherwise. The mask loss Lmask is defined only on positive RoIs. The mask target is the intersection between an RoI and its associated ground-truth mask.



Left/Right panels show the heads for the ResNet C4 and FPN backbones

Paper: <https://arxiv.org/pdf/1703.06870.pdf>

**DeepLabv3**

DeepLab Paper: <https://arxiv.org/pdf/1606.00915.pdf>

DeepLabv3: <https://arxiv.org/pdf/1706.05587.pdf>

**Generative Models:**

Generative modeling is an unsupervised learning task in machine learning that involves automatically discovering and learning the regularities or patterns in input data in such a way that the model can be used to generate or output new examples that plausibly could have been drawn from the original dataset.

In supervised learning, we may be interested in developing a model to predict a class label given an example of input variables.

This predictive modeling task is called classification. Classification is also traditionally referred to as discriminative modeling.

Alternatively, unsupervised models that summarize the distribution of input variables may be able to be used to create or generate new examples in the input distribution.

As such, these types of models are referred to as [generative models](https://en.wikipedia.org/wiki/Generative_model).

[Naive Bayes](https://machinelearningmastery.com/naive-bayes-for-machine-learning/) is an example of a generative model that is more often used as a discriminative model. Naive Bayes works by summarizing the probability distribution of each input variable and the output class. When a prediction is made, the probability for each possible outcome is calculated for each variable, the independent probabilities are combined, and the most likely outcome is predicted. Used in reverse, the probability distributions for each variable can be sampled to generate new plausible (independent) feature values.

Other examples of generative models include Latent Dirichlet Allocation, or LDA, and the Gaussian Mixture Model, or GMM.

**Generative Adversarial Network (GAN):**

Generative Adversarial Networks, or GANs for short, are an approach to generative modeling using deep learning methods, such as convolutional neural networks.

The generator model takes a fixed-length random vector as input and generates a sample in the domain.

The vector is drawn from randomly from a Gaussian distribution, and the vector is used to seed the generative process. After training, points in this multidimensional vector space will correspond to points in the problem domain, forming a compressed representation of the data distribution.

This vector space is referred to as a latent space, or a vector space comprised of [latent variables](https://en.wikipedia.org/wiki/Latent_variable). Latent variables, or hidden variables, are those variables that are important for a domain but are not directly observable.

The discriminator model takes an example from the domain as input (real or generated) and predicts a binary class label of real or fake (generated).

In this way, the two models are competing against each other, they are adversarial in the game theory sense, and are playing a [zero-sum game](https://en.wikipedia.org/wiki/Zero-sum_game). zero-sum means that when the discriminator successfully identifies real and fake samples, it is rewarded or no change is needed to the model parameters, whereas the generator is penalized with large updates to model parameters.

Alternately, when the generator fools the discriminator, it is rewarded, or no change is needed to the model parameters, but the discriminator is penalized and its model parameters are updated.

**Conditional GANs**

An important extension to the GAN is in their use for conditionally generating an output. The generative model can be trained to generate new examples from the input domain, where the input, the random vector from the latent space, is provided with (conditioned by) some additional input.

The additional input could be a class value, such as male or female in the generation of photographs of people, or a digit, in the case of generating images of handwritten digits.

The discriminator is also conditioned, meaning that it is provided both with an input image that is either real or fake and the additional input.

Conditional GANs can perform text-to-image translation, or image-to-image translation and can be used in style transfer, photo colorization, photo enhancement. Successful generative modeling also provides an alternative and potentially more domain-specific approach for data augmentation.

### Anime GAN: <https://arxiv.org/pdf/1708.05509.pdf>

Self-Attention GAN: <https://arxiv.org/pdf/1805.08318.pdf>

Attention is all you need: <https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf>

**Linear Factor Models:**

• A linear factor model is defined by the use of a stochastic, linear decoder function that generates x by adding noise to a linear transformation of h, where h are latent variables.

• A linear factor model describes the data generation process as follows. First, we sample the explanatory factors h from a distribution

ℎ~𝑝(ℎ)

p(h) is a factorial distribution with,



Next we sample the real-valued observable variables given the factors,

𝑥 = 𝑊ℎ + 𝑏 + 𝑛𝑜𝑖𝑠𝑒

where the noise is typically Gaussian and diagonal

**KL Divergence (Kullback–Leibler divergence):**

In mathematical statistics, the Kullback–Leibler divergence, is a measure of how one probability distribution is different from a second, reference probability distribution.

we may have a single random variable and two different probability distributions for the variable, such as a true distribution and an approximation of that distribution.

One approach is to calculate a distance measure between the two distributions. This can be challenging as it can be difficult to interpret the measure.

Instead, it is more common to calculate a [divergence](https://en.wikipedia.org/wiki/Divergence_(statistics)) between two probability distributions. A divergence is like a measure but is not symmetrical. This means that a divergence is a scoring of how one distribution differs from another, where calculating the divergence for distributions P and Q would give a different score from Q and P.

KL Divergence between two distributions Q and P is denoted by KL(P || Q).

|| synbol denotes the P’s divergence from Q

* KL(P || Q) = – sum x in X P(x) \* log(Q(x) / P(x))
* KL(P || Q) != KL(Q || P)

The [Jensen-Shannon divergence](https://en.wikipedia.org/wiki/Jensen%E2%80%93Shannon_divergence), or JS divergence, is another way to quantify the difference (or similarity) between two probability distributions.

It uses the KL divergence to calculate a normalized score that is symmetrical.

* JS(P || Q) == JS(Q || P)
* JS(P || Q) = 1/2 \* KL(P || M) + 1/2 \* KL(Q || M)
* where, M = 1/2 \* (P + Q)

**Auto-encoder:**

An autoencoder is a neural network that is trained to attempt to copy its input to its output. Internally, it has a hidden layer h that describes a code used to represent the input.

It has a Encoder + Decoder Structure.

Encoder h= f(x)

Decoder r=g(h)

• Traditionally, autoencoders were used for dimensionality reduction or feature learning

• Recently, theoretical connections between autoencoders and latent variable models have brought autoencoders to the forefront of generative modeling

• One way to obtain useful features from the autoencoder is to constrain h to have smaller dimension than x

• Learning an undercomplete representation forces the autoencoder to capture the most salient features of the training data

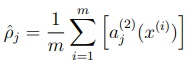
**Sparse autoencoders:**

A sparse representation uses more features where at any given time a significant number of the features will have a 0 value.

To implement, Use more hidden nodes in the encoder and Use regularization techniques which encourage sparseness, which can be Penalty in the learning function for non-zero nodes, Weight decay etc.

A sparse autoencoder is simply an autoencoder whose training criterion involves a sparsity penalty Ω(h) on the code layer h, in addition to the reconstruction error:

𝐿(𝑥, 𝑔(𝑓(𝑥))) + β\*Ω(ℎ)



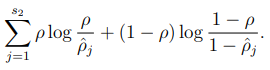
a (2) j (x) denotes the activation of hidden unit j in the autoencoder when the network is given a specific input x.

ρˆj is the average activation of hidden unit j

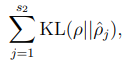
We would like to (approximately) enforce the constraint ρˆj = ρ,

where ρ is a sparsity parameter, typically a small value close to zero (say ρ = 0.05).

To achieve this we add a penalty term,



or,



**Stacked Autoencoders:**

Stack many (sparse) auto-encoders in succession and train them using greedy layer-wise training and Drop the decode layer each time.

Do supervised training on the last layer using final features

Then do supervised training on the entire network to fine- tune all weights.

**Denoising autoencoders:**

Just add noise to the input layer while training, so that the network can perform denoising operation.

**Variational Autoencoders (VAE):**

Autoencoders are not probabilistic, and VAE are generative models that attempt to describe data generation through a probabilistic distribution.

Paper: <https://arxiv.org/pdf/2003.05991.pdf>

**RNN, LSTM, GRU:**

**RNN:**

RNNs are a class of neural networks that allow previous outputs to be used as inputs while having hidden states.

RNNs can be,

1. Vector to sequence models which take a fixed size vector as the input and outputs a sequence. Eg. Image captioning
2. Sequence to vector models take a sequence as input and output a vector. Eg. Sentiment analysis
3. Sequence to sequence models take a sequence as input and output a sequence. Eg. Machine translation
4. Time series prediction models

The tanh activation is used in RNN to help regulate the values flowing through the network. The tanh function squishes values to always be between -1 and 1.

**LSTM:**

Recurrent Neural Networks suffer from short-term memory. If a sequence is long enough, they’ll have a hard time carrying information from earlier time steps to later ones.

During back propagation, recurrent neural networks suffer from the vanishing gradient problem. Gradients are values used to update a neural networks weights. The vanishing gradient problem is when the gradient shrinks as it back propagates through time. If a gradient value becomes extremely small, it doesn’t contribute too much learning.

So in recurrent neural networks, layers that get a small gradient update stops learning. Those are usually the earlier layers. So because these layers don’t learn, RNN’s can forget what it seen in longer sequences, thus having a short-term memory.

As the cell state passes through the network, information gets added or removed to the cell state via gates. The gates control the flow of information.

Gates contain sigmoid activation functions, they can forget the information by setting the value to 0, or remember by having value 1.

we have three different gates that regulate information flow in an LSTM cell. A forget gate, input gate, and output gate.

Forget gate: Forget gate decides what data should be kept or forgotten. It takes the previous cell's hidden state and the current input as its input passes it to sigmoid function.

Input gate: The hidden state of previous cell and the current cell is passed into a sigmoid function and tanh function independently, and the outputs are mutiplied together. The sigmoid decides the relevance of the inputs and updates the value obtained by tanh.

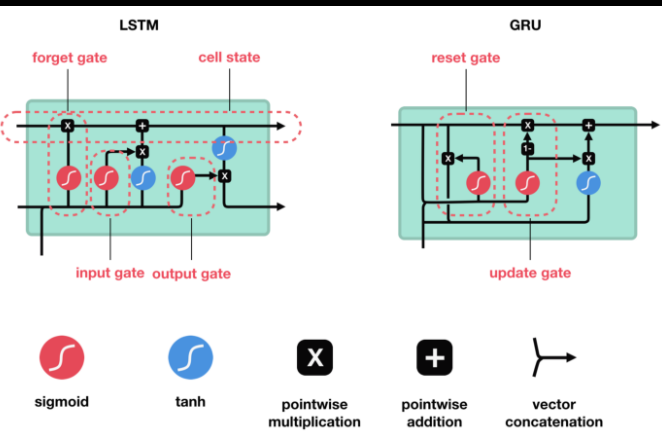
The cell state from the previous cell gets multiplied with the output of forget gate, and subsequently added to the output of the input gate.

Output gate: current cell input and the previous cell's hidden state is first passed to a sigmoid function and the cell state is passed to a tanh function and multiplied to obtain the new hidden state.

**GRU**:

GRU is a compact version of LSTM with few parameters. it doesn't have a cell state and uses the hidden state to transfer information. It has 2 gates, reset gate and update gate.

The update gate acts similar to the forget and input gate of an LSTM. It decides what information to throw away and what new information to add.



Pytorch Code: <https://www.kaggle.com/andradaolteanu/pytorch-rnns-and-lstms-explained-acc-0-99>

<https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21>

**Global Vectors for Word Representation (GloVe)**

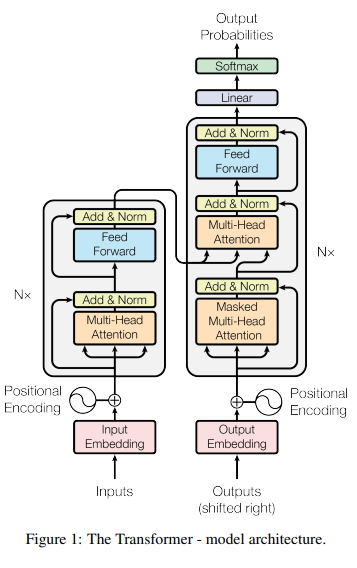
It is an unsupervised learning algorithm for obtaining embedding of words. Training is performed on word-word co-occurrence statistics from a corpus. Pretrained embeddings trained on Wikipedia, Twitter etc. are available.

**Transformer:**

Attention is all you need

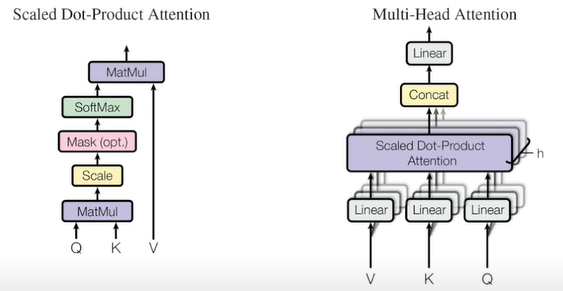
LSTMs perform well for longer sequences (up to 100s of words), but they are slow to train. The inputs are sequential as parallel processing is not possible.

Transformers was introduced in 2017, it has an encoder-decoder architecture but the input sequence can be passed in parallel.

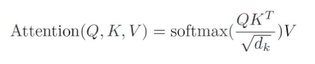


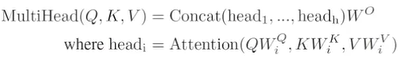
Encoder Components:

* Embedding layer - It converts words into a vector, and similar words are closely present in the embedding space.
* Positional Encoder - It gives context based on the position of the word in the sentence.
  + PE(pos, 2i) = sin (pos/10000^(2\*i/d\_model))
  + PE(pos, 2i+1) = cos (pos/10000^(2\*i/d\_model))
* Encoder block - It contains a Multi-head attention layer and a feed forward layer.
  + Attention is used for determining the weights/significance of each word to the other words in the sentence. Each word has attention weights which measure the contextual relationships. We can have multiple attention weights and take a weighted average of them. This is called as Multi-head attention.
  + Feed forward NNs are a series of linear/non-linear neural networks which transform input into output. The inputs need not be sequential and can be parallelized.



Q, K, V represent the Query, Key and Value. Value and Key come from the encoder and the Query from the output of the decoder/encoder.





Decoder Components:

* Embedding layer
* Positional Encoder
* Decoder block - It contains Masked multi-head attention, multi-head attention and feed forward layer.
  + The outputs from the decoder are passed to an attention block to get attention weights, they are combined with encoder attention weights and passed to another attention block. It represents the relationships between words in both the languages.

Decoder outputs one word at a time so the attention weights for the subsequent outputs are set to zero. This is known as Masked attention. When applying masks the right upper triangle becomes zero after softmax.

* + The outputs of the attention blocks are passed to feed forward neural networks and linear layers with softmax to get the output.

**Bidirectional Encoder Representation from Transformers (BERT) Model**

BERT is formed by stacking a number of encoders in sequence in a transformer. We can use pretrain BERT to encode the information about a language and fine-tune it for a specific task.

We can pretrain BERT by learning on two unsupervised tasks **simultaneously**. They are,

* Masked Language Model (MLM) - BERT takes a sentence with masked words (unknown words) and it is made to predict the missing words. It is similar to fill-in-the-blanks. We can also pretrain using question and answers where the model has to highlight the areas in the answers which corresponds to the question.
* Next Sentence Prediction (NSP) - It takes two sentences as input and predicts if the second sentence is a logical continuation of the first. It is a binary classification problem.

The embeddings are constructed by adding? concatenating? three vectors,

* Token embeddings pre-trained from a corpus
* Segment embeddings - It denotes the sentence number of the input
* Position embeddings - It denotes the position of the word in a sentence

We can use transfer learning and fine-tune the model for specific NLP tasks.

**Generative Pre-training Task (GPT) Model:**

It is a 12 layer 12 attention head transformer decoder developed by OpenAI. We can use pretrained models trained using semi-supervised learning (unlabelled dataset) and fine-tune it for specific tasks. It is trained on 12 supervised learning tasks.

It can be used for natural language inference, multiple choice questions, semantic similarity, classification (grammatically correct or incorrect).

Using more layers of pre-trained GPT can lead to a better performance.

GPT:[https://s3-us-west-2.amazonaws.com/op…](https://s3-us-west-2.amazonaws.com/op%E2%80%A6)

**Bilingual Evaluation Understudy (BLEU):**

BLEU is an algorithm for evaluating the quality of text which has been machine-translated from one natural language to another.

A perfect match results in a bleu score of 1.0, whereas a perfect mismatch results in a score of 0.0.

The approach works by counting matching n-grams in the candidate translation to n-grams in the reference text. The comparison is made regardless of word order.

NLTK provides the sentence\_bleu() function for evaluating a candidate sentence against one or more reference sentences, and a function called corpus\_bleu() for calculating the BLEU score for multiple sentences such as a paragraph or a document.

|  | from nltk.translate.bleu\_score import sentence\_bleu  reference = [['this', 'is', 'a', ‘small’, 'test'], ['this', 'is' 'test']]  candidate = ['this', 'is', 'a', 'test']  score = sentence\_bleu(reference, candidate)  print(score) |
| --- | --- |

**Few Shot Learning (FSL):**

**Triplet Loss:**

**Deep Affinity Network (DAN):**

**Deep Belief Network (DBN):**

**FlowNet2.0:**

論文: <https://arxiv.org/pdf/1612.01925.pdf>

動画: <https://www.youtube.com/watch?v=JSzUdVBmQP4>

Code: <https://github.com/NVIDIA/flownet2-pytorch>

**OpenPose:**

論文: <https://arxiv.org/pdf/1812.08008.pdf>

Code: <https://github.com/CMU-Perceptual-Computing-Lab/openpose>

Action Recognition:

**Temporal 3D Convnets:**

Review: <https://arxiv.org/pdf/1711.08200.pdf>

**I3D CNN:**

ダイアグラム

自動的に生成された説明

**SlowFast:**

論文：<https://arxiv.org/pdf/1812.03982.pdf>

動画：<https://www.youtube.com/watch?v=jt3axjinqIM>

<https://www.youtube.com/watch?v=Flm-kkCqACM>

Code: <https://github.com/facebookresearch/SlowFast>

Dataset: Kinetics, Charades and AVA

Kinetics: <https://deepmind.com/research/open-source/kinetics>

Charades: <http://vuchallenge.org/charades.html>

Keras implementation of Action Recognition

<https://www.youtube.com/watch?v=EliLhaAf-So>

**Multi Instance Learning (MIL):**

論文：<https://arxiv.org/pdf/1801.04264v3.pdf>

Code: <https://github.com/WaqasSultani/AnomalyDetectionCVPR2018>

**Chatbots:**

Satellite Image Analysis:

<https://www.frontiersin.org/articles/10.3389/frai.2020.534696/full>

<https://www.kaggle.com/kmader/segmenting-buildings-in-satellite-images>

<https://www.kaggle.com/kmader/synthetic-word-ocr?select=annotation.json>

<https://course19.fast.ai/videos/?lesson=3>

<https://github.com/robmarkcole/satellite-image-deep-learning#state-of-the-art>

<https://en.wikipedia.org/wiki/Conditional_random_field>

Explainable AI (XAI) - A field of AI, which tries to explain the results and reason for the AI algorithm predictions.

This can be used to understand automatic recommendations and AI decision making.