More Learning Methods

#### Topics

1. Multiclass classification
2. One-class classification
3. Multi-Label Classification
4. Ensemble Learning

* Boosting
  + Gradient Boosting
* Bagging
  + Random Forest

1. Sequence-to-Sequence Learning
2. Active Learning
3. Semi-Supervised Learning
4. One-Shot Learning
5. Zero-Shot Learning

# 1. Multiclass Classification

For multiclass classification problems, we extend the logistic regression to **softmax regression algorithm.** This is done by replacing sigmoid function in logistic regression with softmax function.

The loss function in softmax regression is typically the cross-entropy loss (also known as log-loss). Given a set of training examples, the cross-entropy loss measures the dissimilarity between the predicted probabilities and the true class labels.

- log a1, if y == 1

- log a2, if y == 2

L = :

:

- log an, if y == n

Where aj = ezj / ΣNk=1 = P(y = j | x) , where N is the number of classes

# 2. One class classification

One-class classification, also known as unary classification or class modeling, tries to identify objects of a specific class among all objects, by learning from a training set containing only the objects of that class. The most widely used in practice are one-class Gaussian, one-class k-means, one-class kNN, and one-class SVM.

# 3. Multi-Label Classification

In some situations, more than one label is appropriate to describe an example from the dataset. In this case, we talk about the multi-label classification. The only difference with the usual multiclass problem is that now we have a new hyperparameter: **threshold**. If the prediction score for some label is above the threshold, this label is predicted for the input feature vector. In this scenario, multiple labels can be predicted for one feature vector.

# 4. Ensemble learning

Ensemble learning is a learning paradigm that, instead of trying to learn one super-accurate model, focuses on training a large number of low-accuracy models and then combining the predictions given by those weak models to obtain a high-accuracy **meta-model.**

Low-accuracy models are usually learned by weak learners, that is, learning algorithms that cannot learn complex models, and thus are typically fast at the training and at the prediction time. The most frequently used weak learner is a decision tree learning algorithm in which we often stop splitting the training set after just a few iterations. The obtained trees are shallow and not particularly accurate, but the idea behind ensemble learning is that if the trees are not identical and each tree is at least slightly better than random guessing, then we can obtain high accuracy by combining a large number of such trees. To obtain the prediction for input x, the predictions of each weak model are combined using some sort of **weighted voting**. Two principal ensemble learning methods are boosting and bagging.

#### Boosting and Bagging

**Boosting** consists of using the original training data and iteratively creating multiple models by using a weak learner. Each new model would be different from the previous ones in the sense that the weak learner, by building each new model tries to “fix” the errors which previous models make. The final ensemble model is a certain combination of those multiple weak models built iteratively.

**Bagging** consists of creating many “copies” of the training data (each copy is slightly different from another) and then apply the weak learner to each copy to obtain multiple weak models and then combine them. A widely used and effective machine learning algorithm based on the idea of bagging is random forest.

#### Random Forest

Random Forest is a **specific implementation of bagging that is tailored for decision trees** and **incorporates feature randomness** to further enhance its performance and robustness. In a Random Forest, multiple decision trees are trained on different subsets of the data using sampling with replacement and with random feature subsets. The predictions from these individual trees are then combined to make a final prediction, typically through a majority vote for classification tasks or averaging for regression tasks.

Random forest is one of the most widely used ensemble learning algorithms. The reason is that by using multiple samples of the original dataset, we reduce the variance of the final model, lowering overfitting.

#### Gradient Boosting

Gradient boosting is a modified boosting algorithm that implements a gradient descent (in a non traditional way) to calculate the error caused by the ensemble trees. This error is used to update the trees, training the model and reducing the error. However, instead of getting the gradient directly, we use its proxy in the form of residuals: they show us how the model has to be adjusted so that the error (the residual) is reduced. It reduces underfitting, but can lead to overfitting. So careful selection of hyperparameters is adviced.

Gradient boosting is one of the most powerful machine learning algorithms—not just because it creates very accurate models, but also because it is capable of handling huge datasets with millions of examples and features. It usually outperforms random forest in accuracy but, because of its sequential nature, can be significantly slower in training.

# 5. Sequence-to-Sequence Learning

**Sequence Labeling** is the problem of labelling a sequence of data. For eg, classifying each word in a string to {‘noun’, ‘verb’, ‘adjective’}.

**Sequence-to-Sequence learning (seq2seq learning)** is a generalisation of the sequence labeling problem. The crux of the problem is to take in a sequence and output another sequence (of different length) according to the situation. Eg: text generation, machine translation, etc.

**Word embeddings**

They are a replacemetn to the typical one hot vector representation of words. Word embeddings are a type of representation of words as numerical values in a way that allows machines to understand and process the meaning of words in a meaningful and context-aware manner. This can be done by training a neural network on a lot of data, and asking the nn to generate an n dimensional vector to represent a word. This also leads to similar words having similar embeddings. Popular word embedding techniques include Word2Vec, GloVe, and FastText.

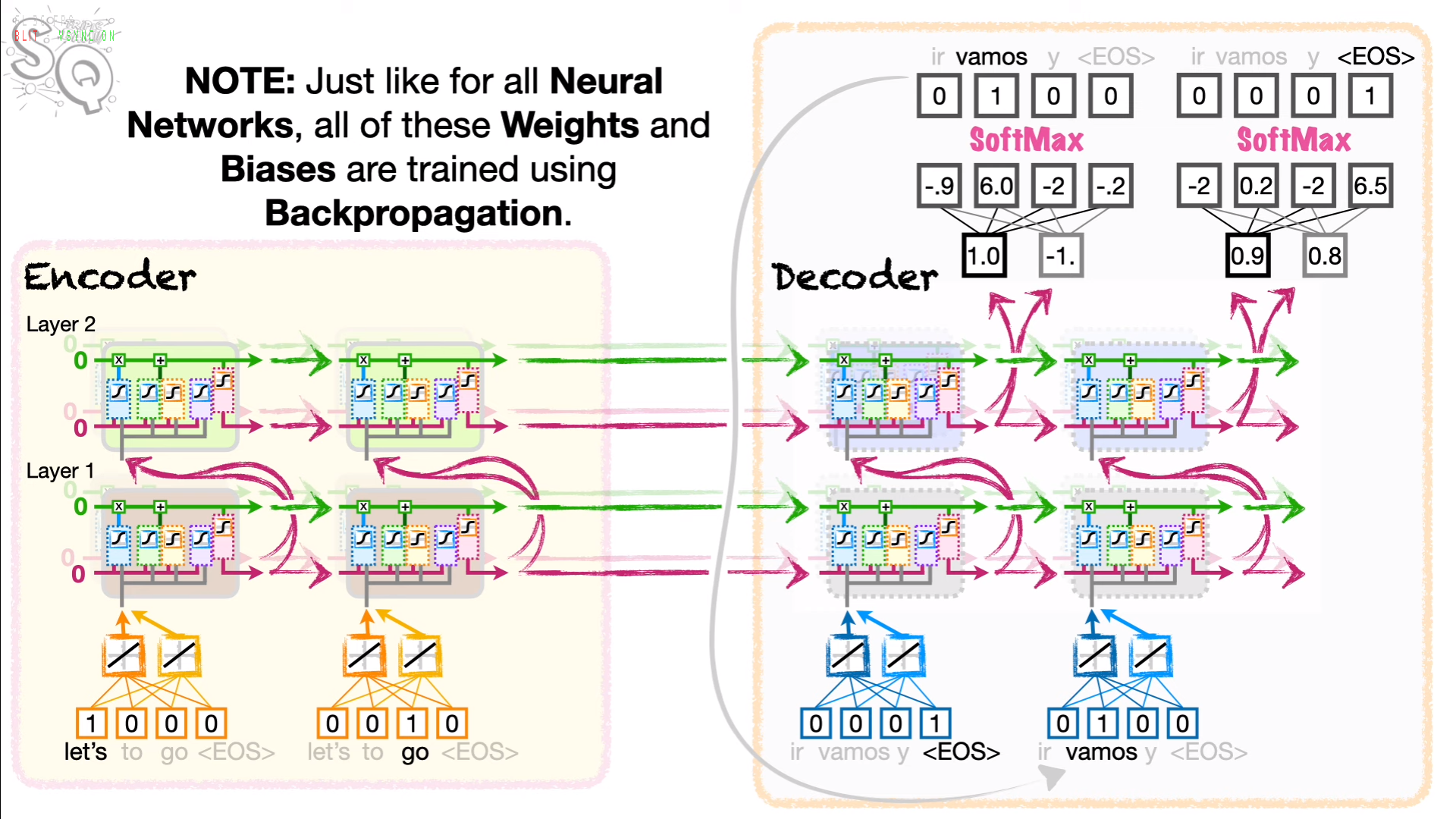
**Word2vec** does word embeddings, but also does two more things

1. Continuous bag of words: It looks at surrounding words to create the embedding.
2. Skip Gram: Uses words in middle to predict surrounding words.

The network architecture used to solve seq2seq problems have two parts: an encoder and a decoder.

**Encoder:** We input the **embedding** of the data into the encoder, which is converted into a **context vector** by the encoder layer. The embeddings provide a static representation of each word and are usually pre-trained on large collection of texts to capture word semantics but do not capture the context of the entire sequence. The encoder's role is to process these embeddings and produce a context vector that summarizes the sequence's context.

**Decoder:** The context vector from the encoder is used to initialise the decoder. The role of the whole decoder setup is to decode the context vector into the output sentence. It has a separate embeding layer that describes its vocabulory. The input into this layer (for inputing the output of the previous timestamp) is passed throught this embedding layer. The output from the decoder layer is passed to a fully connected layer, and into a softmax function, which gives out the probability of each word (in decoder vocabulory) as output.



Traditionally, sequence-to-sequence (seq2seq) models used recurrent neural networks (RNNs), including Long Short-Term Memory (LSTM) networks, for both encoders and decoders. LSTMs are capable of handling sequential data and have been widely used in NLP tasks. However, more modern seq2seq models, especially those based on the Transformer architecture, have become popular and do not rely on LSTMs.

Attention

**Learn later deeply**

# **6. Active Learning**

The idea is to start training a supervised model with relatively few labeled examples, and a large number of unlabeled ones, and then label only those examples that contribute the most to the model quality. It is usually applied when obtaining labeled examples is costly.

Data density and uncertainty based model: For each unlabeled example x, the following importance score is computed: density(x) \* uncertainty(x). Density for the example x can be obtained by taking the average of the distance from x to each of its k nearest neighbors. Once we know the importance score of each unlabeled example, we pick the one with the highest importance score and ask the expert to annotate it.

SVM based model: The support vector-based active learning strategy consists in building an SVM model using the labeled data. We then ask our expert to annotate the unlabeled example that lies the closest to the hyperplane that separates the two classes.

# 7.Semi-Supervised Learning

In semi-supervised learning (SSL) we also have labeled a small fraction of the dataset; most of the remaining examples are unlabeled. Our goal is to leverage a large number of unlabeled examples to improve the model performance without asking for additional labeled examples.

#### Self-learning

One frequently cited SSL method is called self-learning. In self-learning, we use a learning algorithm to build the initial model using the labeled examples. Then we apply the model to all unlabeled examples and label them using the model. If the confidence score of prediction for some unlabeled example x is higher than some threshold (chosen experimentally), then we add this labeled example to our training set, retrain the model and continue like this until a stopping criterion is satisfied. We could stop, for example, if the accuracy of the model has not been improved during the last m iterations.

# 8. One shot learning

One-shot learning is a machine learning paradigm that focuses on training models to recognize and classify objects or patterns based on just a single or very few examples of each class. One way to effectively solve the problem is to train a siamese neural network (SNN). To train an SNN, we use the triplet loss function.

# 9. Zero-shot learning

Zero-shot learning (ZSL) is a machine learning paradigm where a model is trained to recognize and classify objects or concepts it has never seen during the training phase. For eg, it could be to predict a the class of the object, if that class name and object was never in the training data.

Handling Imbalanced Datasets

* We should use precision and recall for error measurement
* **Over-sampling:** It consists of increasing the importance of examples of some class by making multiple copies of the examples of that class.Eg: SMOTE and ADASYN.
* **Undersampling** is to randomly remove from the training set some examples of the majority class
* Some algorithms are less sensitive to the problem of an imbalanced dataset. Decision trees, as well as random forest and gradient boosting, often perform well on imbalanced datasets.

Combining Models

A modification to be done to ensemble method is to use various models (of different architectures) to create an ensemble. Three typical ways to combine models are 1) averaging, 2) majority vote and 3) stacking.

Tokenization

A tokenizer, in the context of natural language processing (NLP), is a fundamental component that breaks down a sequence of text, typically a sentence or a document, into individual units known as tokens. Tokens are usually words, subwords, or characters, depending on the level of granularity chosen for tokenization. Tokenization is a crucial preprocessing step in NLP that helps convert raw text into a format suitable for analysis and further processing.

Advanced Regularization

In neural networks, besides L1 and L2 regularization, you can use neural network specific regularizers: dropout, early stopping, and batch-normalization.

* **Dropout:** Each time you run a training example through the network, you temporarily exclude at random some units from the computation. The higher the percentage of units excluded the higher the regularization effect.
* **Early stopping** is the way to train a neural network by saving the preliminary model after every epoch and assessing the performance of the preliminary model on the validation set. Models saved after each epoch are called checkpoints.
* **Batch normalization** (which rather has to be called batch standardization) is a technique that consists of standardizing the outputs of each layer before the units of the subsequent layer receive them as input. In practice, batch normalization results in faster and more stable training, as well as some regularization effect.
* **Data augmentation:** It is a technique used to preprocess image dataset. Once you have your original labeled training set, you can create a synthetic example from an original example by applying various transformations to the original image: zooming it slightly, rotating, flipping, darkening, and so on.

**Standardization** scales data to have a mean (average) of 0 and a standard deviation of 1. It transforms the data to follow a standard normal distribution (zero mean and unit variance). **Normalization** scales data to a specific range, typically [0, 1], or any user-defined range [a, b]. It transforms the data proportionally to fit within this range.

#### Transfer Learning

In transfer learning, you pick an existing model trained on some dataset, and you adapt this model to predict examples from another dataset, different from the one the model was built on.

Neural networksexcel in transfer learning by enabling a process where you extract the benefits of pre-trained models. You start by discarding the final one or more layers of the original model, typically responsible for classification or regression. Then, you customize the model by introducing new layers tailored to your specific problem. To retain the valuable knowledge gained from the initial training, you "freeze" the parameters of the remaining layers from the original model.

AI Algorithm Efficiency

Whenever we are creating an algorithm, we have to make sure that it is efficient. It can be done by:

* Replace as many loops as possible by vectorisation.
* Use appropriate data structures.
* Unless you know exactly what you do, always prefer using popular libraries to writing your own scientific code.They have many methods implemented in the C programming language for maximum efficiency.
* If you need to iterate over a vast collection of elements, use generators that create a functionthat returns one element at a time rather than all the elements at once.
* Use the cProfile package in Python to find inefficiencies in your code.
* multiprocessing package to run computations in parallel
* Compilation of python code with PyPy, Numba, Pytorch 2.0, etc