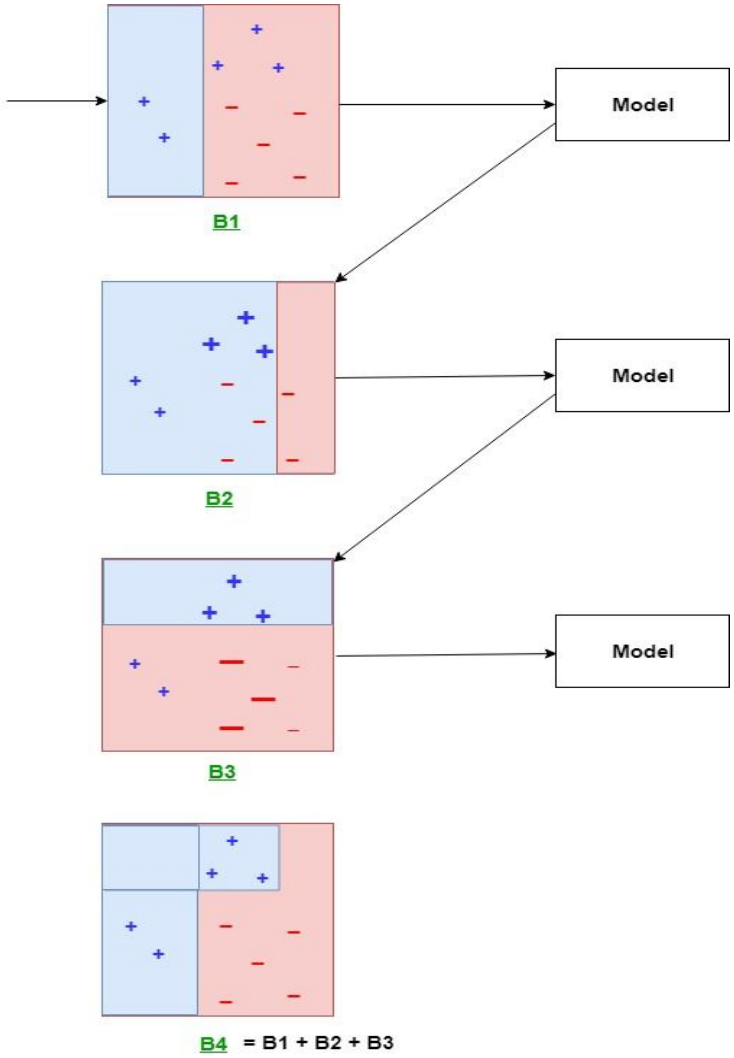
Algorithms in Sagemaker

XGBoost:

- XGboost stands for Extreme Gradient Boosting.
- This is another side of the ensembling methods for machine learning where we use a number of models to solve a problem.
- This algorithm comes under the boosting type of ensemble where a model is trained to reduce the error from the previous model.



- Uses decision trees and regression trees as models for classification and regression respectively.
- Recently, the algorithm is popularized in winning Kaggle competitions at very high margin accuracies and is really fast.

Parameters

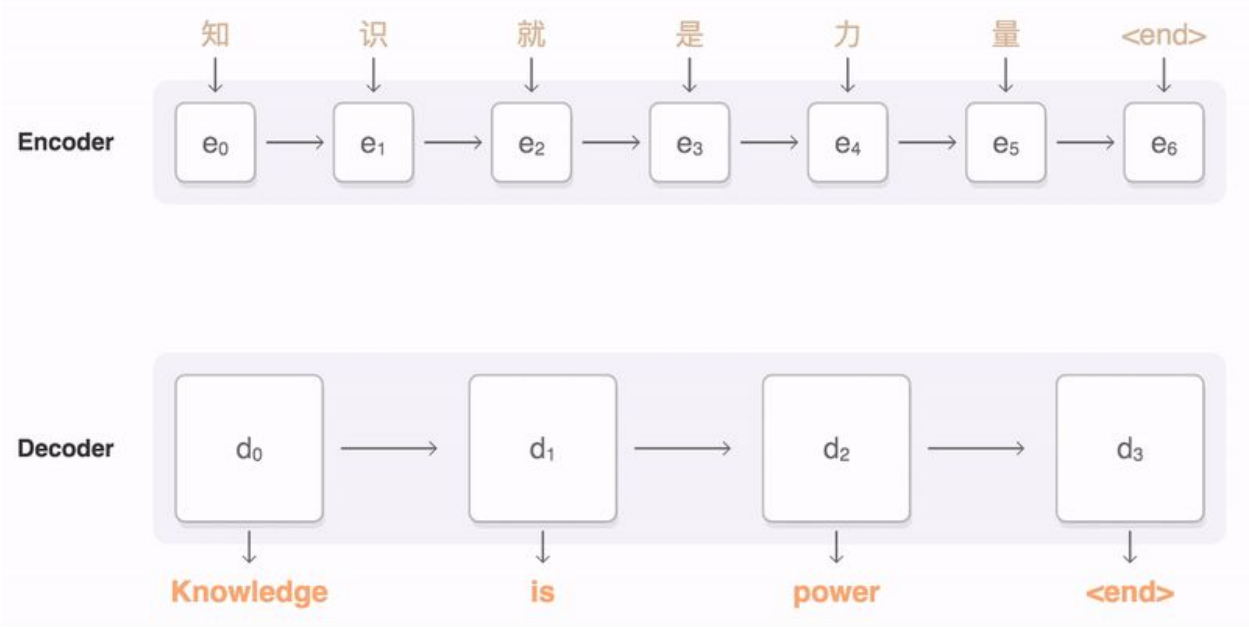
- Parameters of the algorithm include
    1) eta [default=0.3]
        a) Analogous to learning rate.
        b) Makes the model more robust by shrinking the weights on each step
    2) Subsample [default=1]
        a) Denotes the fraction of observations to be randomly samples for each tree
        b) Lower values make the algorithm more conservative and prevents overfitting but too small values might lead to under-fitting
    3) max_depth [default=6]
        a) The maximum depth of a tree
    4) booster [default=gbtree]
        a) Select the type of model to run at each iteration. It has 2 options:
        - gbtree: tree-based models
        - gblinear: linear model

XGBoost in Sagemaker

- Takes csv and libsvm as input formats.(Does not support AWS input mechanics)
- It is a CPU and memory greedy algorithm. it does not utilize GPU. So, M4 is a good selection of instance type for creating or deploying these models.

Seq2Seq:

- Generally used when the input of the problem involves a sequence of inputs or/and we require the outputs to be a sequence.
- Examples include, Machine Translation, Speech to Text, Transcription, Video Frames prediction and so on.
- This is built using CNN's and RNN's with LSTM's and attention mechanisms.



Seq2Seq in SageMaker

Input

- The input is expected in the form of some tokens or integers. For instance, the vocabulary needs to be vectorized using some techniques like one-hot, tf-idf, word2vec etc before passing into the neural network.
- The RecordIO format is used as the input format.

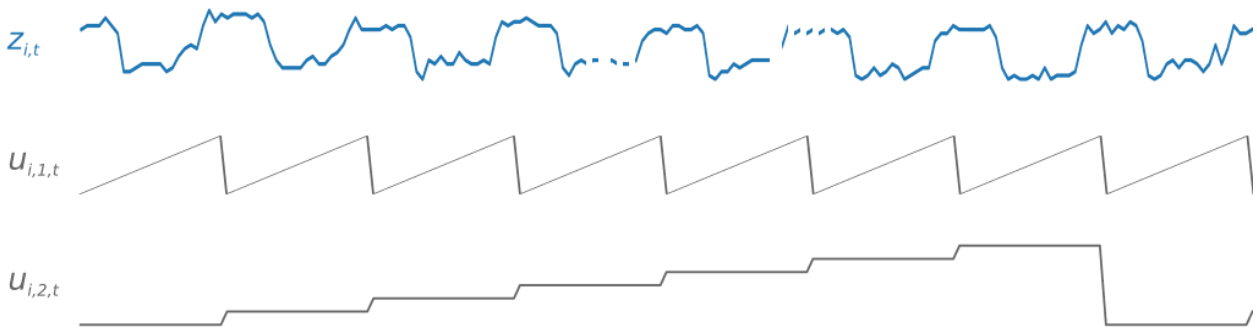
Parameters

a) Optimizer - The optimization algorithm (SGD,Adam,RMSprop)
b) No of encoding and decoding layers.
c) Accuracy Metrics - Accuracy, BLEU score or Perplexity(inverse of accuracy)

DeepAR:

- The Amazon SageMaker DeepAR forecasting algorithm is a supervised learning algorithm for forecasting scalar (one-dimensional) time series using recurrent neural networks (RNN).
- The DeepAR is specialized in cases where you have many similar time series across a set of cross-sectional units.

- For example, you might have time series groupings for demand for different products, server loads, and requests for webpages. For this type of application, you can benefit from training a single model jointly over all of the time series



Input

- The input is given in JSON format and should contain:
    1) `start`—A string with the format `YYYY-MM-DD HH:MM:SS`.
    2) `target`—An array of floating-point values or integers that represent the time series

Instance

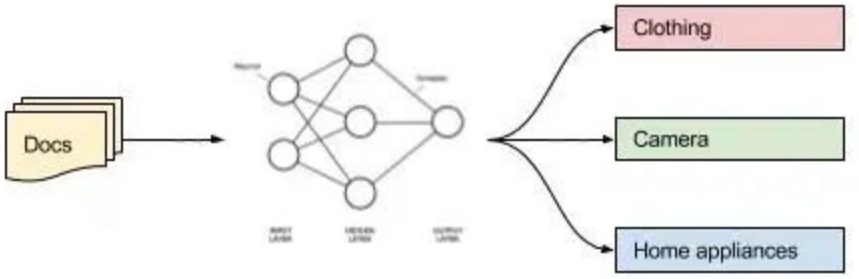
- It can be trained on both GPU and CPU instances and in both single and multi-machine settings.

Best Practices

- Except for when splitting your dataset for training and testing, always provide the entire time series for training, testing, and when calling the model for inference
- Avoid using very large values (>400) for the `prediction_length` because it makes the model slow and less accurate.
- It is recommended to train a DeepAR model on as many time series as are available to utilize it completely.

BlazingText:

- The Amazon SageMaker BlazingText algorithm provides highly optimized implementations of the Word2vec and text classification algorithms.
- The Word2vec algorithm is useful for many downstream natural language processing (NLP) tasks, such as sentiment analysis, named entity recognition, machine translation, etc.
- Text classification is an important task for applications that perform web searches, information retrieval, ranking, and document classification.



Input

- For Word2Vec training, upload the file under the *train* channel. No other channels are supported. The file should contain a training sentence per line.
- For supervised mode, you can train with file mode or with the augmented manifest text format.

  {"source":"linux ready for prime time , intel says , despite all the linux hype", "label":1}

  {"source":"bowled by the slower one again , kolkata , november 14 the past caught up with sourav ganguly", "label":2}

Parameters

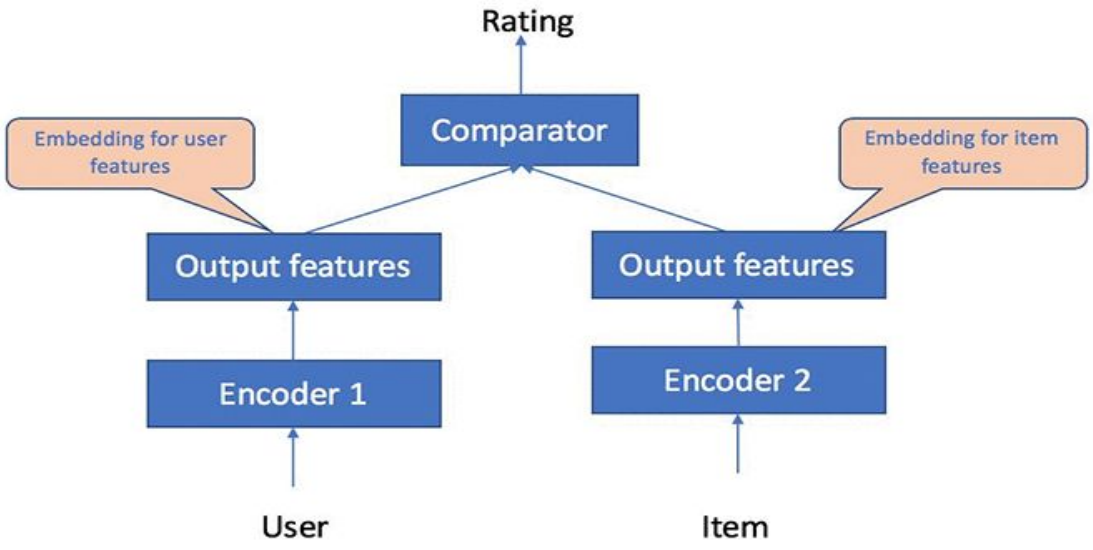
- For word2vec, we can choose between CBOW,Skip-Gram or Batch-Gram(for distributed training)
- We have common parameters like learning rate, window size and embedding dimensions, etc.

Instance Type

- For the supervised text classification mode, a C5 instance is recommended if the training dataset is less than 2 GB. For larger datasets, we use an instance with a single GPU (ml.p2.xlarge or ml.p3.2xlarge) and for Word2Vec it is best to use ml.p3.2xlarge instance since it requires only one GPU and CPU except Batch skip gram mode which can use single node with multiple CPU's because it is distributed.

Object2Vec

- Object2Vec is the generalized case of Word2Vec in BlazingText, where we can create an embedding not only for words, but also with other objects like sentences etc.



Input

Object2Vec natively supports two types of input:

a) A discrete token, which is represented as a list of a single integer-id. For example, [10].

b) A sequence of discrete tokens, which is represented as a list of integer-ids. For example, [0,12,10,13].

Parameters

The algorithm supports the following as encoders:

a) Average-pooled embeddings

b) Hierarchical convolutional neural networks (CNNs),

c) Multi-layered bidirectional long short-term memory (BiLSTMs)

And there are some common neural network parameters.

Instance Type

When training a model using the Object2Vec algorithm on a CPU, it's best to start with an ml.m5.2xlarge instance. For training on a GPU, start with an ml.p2.xlarge instance.

For inference with a trained Object2Vec model that has a deep neural network, it is recommended using ml.p3.2xlarge GPU instance.

Object Detection in SageMaker

- The Amazon SageMaker Object Detection algorithm detects and classifies objects in images using a single deep neural network. It is a supervised learning algorithm that takes images as input and identifies all instances of objects within the image scene.
- The object is categorized into one of the classes in a specified collection with a confidence score that it belongs to the class. Its location and scale in the image are indicated by a rectangular bounding box.

Input

The recommended input format for the Amazon SageMaker object detection algorithms is Apache MXNet RecordIO. However, you can also use raw images in .jpg or .png format.

Parameters

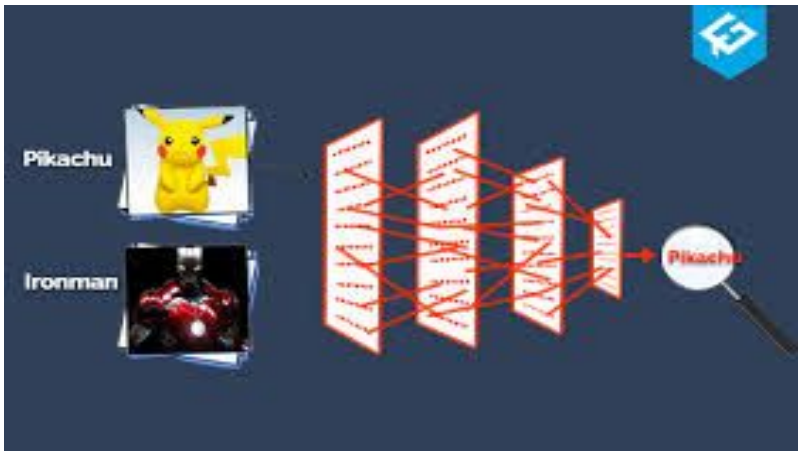
- Common Neural Network parameters and choice of classifier (Resnet or VGG19)

Instance Type

- For object detection, we support the following GPU instances for training: ml.p2.xlarge, ml.p2.8xlarge, ml.p2.16xlarge, ml.p3.2xlarge, ml.p3.8xlarge and ml.p3.16xlarge

Image Classification

- It uses a convolutional neural network (ResNet) that can be trained from scratch or trained using transfer learning when a large number of training images are not available.
- This is a simple version of detection where we predict only the class labels excluding the bounding boxes.



Input

- The SageMaker Image Classification algorithm supports both RecordIO (application/x-recordio) and image (image/png, image/jpeg, and application/x-image) content types for training in file mode, and supports the RecordIO (application/x-recordio) content type for training in pipe mode.

Parameters

- Common Neural Network Parameters.

Instance Type

- For image classification, we support the following GPU instances for training: ml.p2.xlarge, ml.p2.8xlarge, ml.p2.16xlarge, ml.p3.2xlarge, ml.p3.8xlargeand ml.p3.16xlarge. We recommend using GPU instances with more memory for training with large batch sizes.

Semantic Segmentation

- The more advanced version of object detection is segmentation where the detection takes place at pixel level.
- The segmentation output is represented as a grayscale image, called a *segmentation mask*. A segmentation mask is a grayscale image with the same shape as the input image.

Input

- It also supports the augmented manifest image format (application/x-image) for training in Pipe input mode straight from Amazon S3. For inference, an endpoint accepts images with an image/jpeg content type.

Parameters

- You also have a choice of backbones for the FCN, PSP, and DeepLabV3 algorithms: ResNet50 or ResNet101. These backbones include pretrained artifacts that were originally trained on the ImageNet classification task
- Other common Neural Network parameters.

Instance Type

- The SageMaker semantic segmentation algorithm only supports GPU instances for training, and we recommend using GPU instances with more memory for training with large batch sizes like `ml.p2.xlarge ml.p2.8xlarge ml.p2.16xlarge ml.p3.2xlarge`.