Ammar Raşid.

Machine Learning Term Project Report

Contents

Introduction & Project overview:..... 2
Performance Metrics:..... 2
Data:..... 3
Approaches..... 4
Selective Waves..... 4
1- Converting the data frame to a dictionary..... 4
2- Training..... 4
3- Prediction..... 5
Quadro-bridge..... 7
1- Converting the data frame to dictionary..... 7
2- Training & Prediction..... 7
Results and Discussion..... 8
Appendix..... 9

Introduction & Project overview:

The project is about using multiple labelled textual corpora from different languages to leverage the representation power of resources-rich "source" languages (e.g. English) into enhancing the models of "target" languages that are relatively less available (e.g. Bengali, Swahili, …etc.). This project focuses on sentiment classification of movie reviews collected from IMBD for English as the source language, and Beyazperde for Turkish as the target language. The aim of the project is not to learn a new word embeddings for the target language, but to rather use the existing ones (e.g. Fasttext and Google's word2vec) to generalize the sentiment knowledge from the source language to the target language. One caveat is that, the motivation of this project is to mitigate the effect of data scarcity in many languages (e.g. Swahili, Bengali …etc.) and not to obtain an optimal representation. That being said, given enough data, existing algorithms (e.g. Logistic Regression, RandomForest …etc. using only doc2vec or BOW for sentence representation) will suffice in learning reasonably powerful sentiment classification models. Turkish, though does not have as abundant textual resources as English for example, has arguably big-enough corpus to train powerful word embeddings that are per se capable of achieving satisfactory results in sentiment classification tasks. However, my aim in this project is trying new approaches and architectures that have not been tried before, improving them to the degree that they perform at least as good as monolingual classifiers. In this project, I show that using languages that are rich with labelled corpora can make up for the scarcity of their relatively poor peer languages in labelled corpora, and, thus, eliminating the need for tedious tasks of creating and collecting large labelled corpora.

Performance Metrics:

Scores given to the movie reviews range from 1 to 10. For ratings with floating numbers and ratings that range from 1 to 5, I scale the ratings floor them so that all ratings range from 1 to 10. For the classification accuracy, I take the mean of the absolute difference between predicted scores and actual scores of testing movie reviews. Code 1. To insure the robustness of validation, I run 10-fold test for 10 trials. I use different random state for each trial. However, I noticed that all trials gave exactly similar results. Therefore, I will report only the scores of each of the 10-folds for just one trial.

```
def distance_accuracy(y_true, y_predict):
    res = 0
    for i in range(len(y_true)):
        res += abs(y_true[i]-y_predict[i])
    return 1-res/(len(y_true)*len(set(y_true)))
```

Code 1 Measuring Model's Accuracy

Data:

I used my own crawler that I have written last summer for my internship to crawl movie reviews from IMDB and Beyazperde and match them as shown in Figure 1. The approaches proposed in this project are not affected by the bias in the ratio of Score. There are 1000 reviews (500 English and 500 Turkish). I used TextBlob library to clean the corpus, tokenize reviews, stem the words (for Turkish I used TurkishStemmer: okullarından > okul) and handle special characters (e.g. Ö, Ü > O, U).

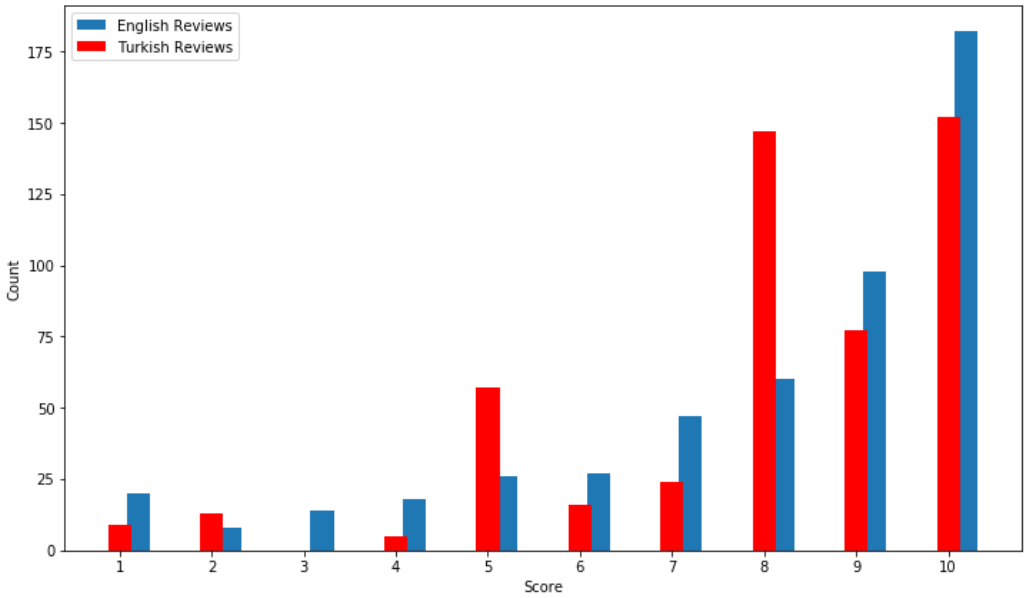

Figure 1 Data Statistics

I used Google's pre-trained word2vec word embeddings to vectorise the words in each English review. For Turkish review, I used Facebook's Fasttext. The vector representation of a review (rev_vec) is the mean of the matrix M representing that document. A Row R_i in a document's matrix M correspond to the vector representation (word2vec for English and Fasttext for Turkish) of the word at index i in that document. Figure 2

	Language	Movie_ID	Score	rev_vec
0	en	-800777728	9	[0.0476253, 0.0501914, 0.00908411, 0.0771895, ...
1	en	-800777728	10	[0.0624585, 0.0365933, 0.00594799, 0.088586, -...
2	en	-1018312192	8	[0.0634076, 0.0370061, 0.00736885, 0.0935822, ...
3	en	-1018312192	4	[0.0535533, 0.0397413, -0.0136467, 0.0953033, ...
4	en	-1018312192	7	[0.0395328, 0.0405981, 0.0048423, 0.0989682, -...

Figure 2 Representing Movie Reviews as vectors

Approaches

Selective Waves

"Selective Waves" is a neural network where there are a pool of weight matrices (hidden layers) between each two layers. The weight matrix between two layers is chosen based on the labels associated with the first layer, Figure 3. An overview of Selective Waves neural network is depicted in Figure 6.

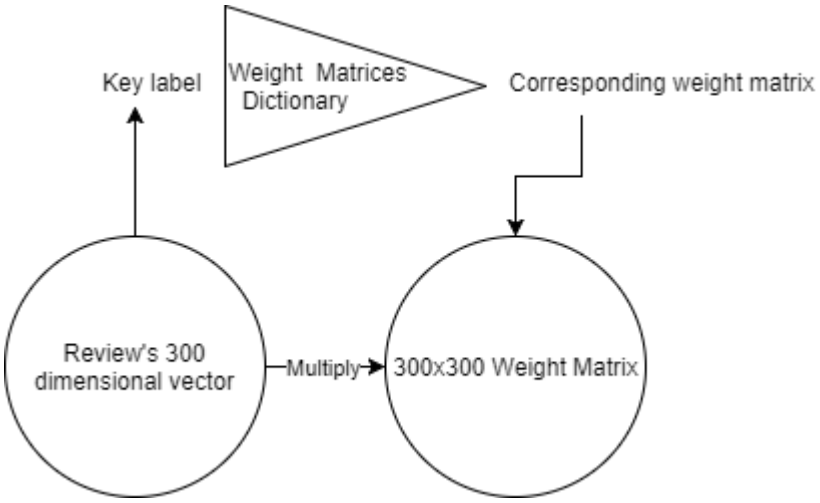

Figure 3 The building block of Selective Waves NN, key labels are "(language, movie_id, score), (language, score) and score" in that order. One caveat is that the last transition will have a Weight Matrix of size 300x10 as there are 10 classes (1-10).

I developed two variations of "Selective Waves" neural network. One compromises accuracy for speed (approximately 25 seconds per fold on 900 instances achieving 78% accuracy on 10-folds average). The other compromises speed for accuracy (approximately 250 seconds per fold achieving 81.8% accuracy on average). The working flow of each variation is delineated below.

1- Converting the data frame to a dictionary

`get_data_dict` function takes a dataframe of the reviews as a parameter and returns two dictionaries; `data_dict` and `L1`. `data_dict` is a nested dictionary with a structure of: `{language:{score: {movie_id: [rev1, rev2, ..., revn]}}}`. `L1` on the other hand, L1 stands for first layer, looks like: `{(language, score, movie_id): [rev1, rev2, …, revn]}`. The other variation of Selective Waves returns two more dictionaries, L2 and L3 that are similar to L1 but with different keys. L2's keys are (language, score) and L3's keys are just scores. The values of L2 and L3 dictionaries are simply none as we just need the keys. Since the size of L2 and L3 are not clear while constructing the data dictionary, np arrays were not possible for assignment. I ought to use dictionaries instead as they are more efficient for inserting operations than append operation in Python lists.

2- Training

The first variation has three steps for training. The first step is obtaining a matrix Y of size MxF where M is the number of classes (10 in this case) and F is the number of features (300 in this case). The second step is training a repressor, with X as the training instances matrix of size NxF, where N is the number of instances, and the labels as the matrix Y obtained from the first step. Lastly, the third step is to train a classifier on the matrix Y obtained in the first step with labels y, a vector of the scores of the instances. The `fit` function returns the trained regressor and classifier. Code 2.

Score vectors are obtained with the Selective Wave neural network. Upon obtaining `data_dict` from above, I feed each layer with the corresponding weight matrix to a merging function that returns the vectors for the next layer. The output of the merging function is fed to another function to convert the output into a dictionary format to prepare it for merging. Knowing the size of each layer, I initialize a 3d array of weight matrices, where the first dimensions is just used as an indexer for the weight matrices that are initialized randomly with a zero mean. The role of the LSMR naming conventions arises here as after each merging function, one letter is shifted. So at first, we have language L, score S, movie M and a list of reviews of movie M, with score S and written in language L. By applying the first merging step, we get one vector representing all the reviews of language L, score S and movie M, thus, LSM_R. With the next merging step, we get one vector representing all movies with language L and score S, thus, LS_MR.

The `merge` function multiplies each vector in the input layer by the corresponding weight matrix in the input 3d array holding all weight matrices, adds nonlinearity to the multiplication output and finally takes the average of all resulted sub-items. For example, if we are merging all reviews for a key of (language L, movie M, score S), we would multiply each review vector of that movie to the weight matrix corresponding to (language L and movie M and score S), apply the sigmoid function on the multiplication output and take the average of all the resulting vectors for all reviews of that movie. Therefore the result would be a single vector representing a movie M for a language L and score S.

Having a prediction at the last layer, I implemented backpropagation algorithm to propagate the error back to the first layer and repeat for 100 epochs. I found that after 100 epochs the calculated error doesn't decrease much. The error calculation and backpropagation code is shown in Code 5.

The other variation differs in that it does not train a regressor on the score vectors, but rather maintains the weight matrices in a dictionaries and returns those dictionaries. This way, a user could make use of any available labels to improve the performance of the prediction. For example, if the model is given a document vector to classify and told that the language of that document is Turkish it performs better in predicting the rating associated with that document than if given only the document. The model, therefore, allows the utilization of any labelling of the data, such as region of the review, actors in the movie, year of release …etc. Another key advantage is the significant speed than having to train a neural network without weight matrices dictionaries which takes days to converge. Unlike the first variation which uses full-batch training, the second variation is trained instance by instance. I implemented this variation of Selective Waves using TensorFlow-GPU and trained it with a GTX 1070, which has a 6.1 computation capability. The two variations also differ in the prediction function as it will be explained in the next section. The code for the training function of the second variation is shown in Code 6.

3- Prediction

The predictor of the first variation of Selective Waves uses the regressor and classifier returned by the `fit` function (Code 2). First, the regressor is used to transform the testing instance 300-dimensional vector V_x to another 300-dimensional vector V_s that represents its sentiment. The sentiment vector V_s is then fed to the classifier to get a predicted score (Code 7). One advantage of this variation of Selective Waves is the freedom to use any regressor and any multi-label classifier. I have tested with various regressor and multi-label classifiers and best results were achieved with MLP Regressor and MLP Classifier. However, the nature of the model allows using a KNN classifier, which provides significantly similar accuracy in less time. Though I still ought to report the better accuracy regardless of the runtime as MLP Classifier is still fast with this small number of testing instances (i.e. 100 instances).

The predictor of the second variation, on the other hand, uses the weight matrices returned by the Code 6 function predict the scores of the testing instances. Selective Waves neural network is run

with every weight matrix for each testing instances. The prediction made is the average of all predictions made by the network with each of the weight matrices. The worst case complexity of this prediction is O(n) where n is the number of training instances. However, this complexity could be reduced significantly if a relative information is known about the testing instance. For example, if there are n reviews in the training set, each has a different (language, movie, score) combination, the training function will return a dictionary W1 of n weight matrices, and two more dictionaries W2 and W3. W1's keys are tuples of (language, movie id, score). W2's keys are (language, score). W3's keys are (score). The values of each of the dictionaries are the weight matrices corresponding to the keys. If the predictor is given a movie id m, then only the weight matrices whose keys in W1 include m. Unlike the first variation where the regressor and classifier played a significant role in the predictor's performance, the second variation of Selective Waves depends only on the weight matrices. Therefore, the training of the second variation is directly related to the predictions, whereas the training of the first variation's score vectors is bounded by, and even might be refuted by, the performance of the classifier which is bounded by the performance of the regressor.

Quadro-bridge

In this approach the input is four layers. The first layer corresponds to the Turkish reviews. The second layer corresponds to the English translation of the Turkish reviews. The third layer corresponds to the Turkish translation of the English reviews. The fourth and last layer corresponds to the English reviews. The reviews are aligned based on the movies they were written for and the rating scores associated with them. The intuition of this approach is that, instead of bridging the gap between the vector spaces of the two languages only with sentiment alignment, we try to also capture semantic information by minimizing the distance between the review and its translation. Moreover, the sentiment alignment (by movie and score) is 'flattened' over three 'bridges' instead of one. The first bridge is the weight matrix between the first (Turkish reviews) and second (English translation of the Turkish reviews) layers. Likewise, the third bridge is the weight matrix between the third (Turkish translation of the English reviews) and the fourth (English reviews) layers. The second or middle bridge plays the role of 'sentiment alignment' by minimizing the distance between the second (English translation of the Turkish reviews) and the third (Turkish translation of the English reviews) layer.

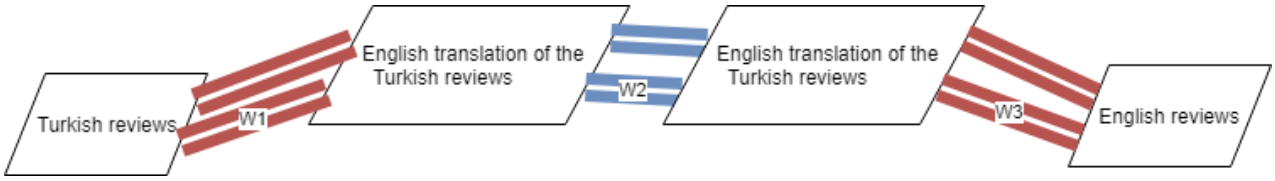

Figure 4 Quadro-bridge alignment overview

1- Converting the data frame to dictionary

Like the dictionaries made in Selective Waves, the dictionary of quadro-bridge is also nested. The key of the dictionary is a tuple of the labels used for alignment. I use the 'score' *s* associated with a review and 'movie id' *m* for which the review was written, for alignment. Therefore, the key of the dictionary is (score, movie id). The value is a dictionary in which the key is the language of the reviews, and the value is a list of reviews written in that language for the movie *m* and associated with which a score *s* (i.e. the key of that value). All scores are represented with 1-hot-encoding. The code for the constructing the data dictionary is shown in Code 9. Eventually the data dictionary looks like:

```
{(score, movie_id): {language: [reviews]}}
```

2- Training & Prediction

The cost function of this approach is the summation of the difference between the 'predicted layer' and the layer. The predicted layer is the vector produced by the multiplication of the previous layer and the corresponding weight matrix. Since we have three 'bridges' (i.e. weight matrices) as depicted in Figure 4, there are three cost functions C1, C2 and C3. Let X*i, W*i and y be the layer *i*, weight matrix *i* and the label (i.e. score). C*i* is the reduced summation of the squared difference between the sigmoid output of the dot product of X*i* with W*i* and X*i+1*. The last component of the cost function, C4, is cross entropy. The model's cost function is the summation of the four cost functions. I then use tensorflow for optimization. The Code 11 returns the optimized weight matrices to be used in prediction.

Results and Discussion

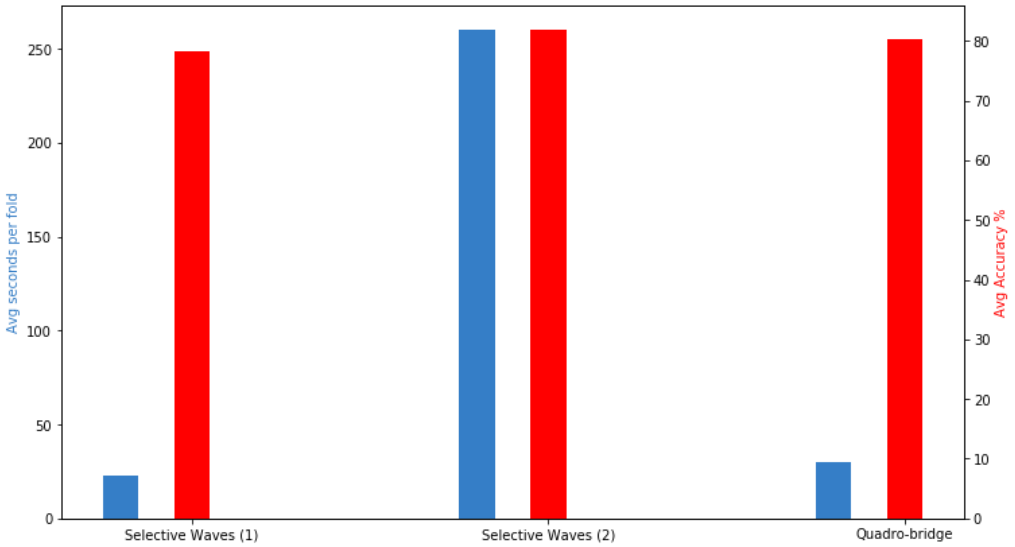

Figure 5 Models benchmarks

	Pros	Cons
Selective Waves (1) 23 sec/fold 78.2%	1-Freedom to use any regressor or classifier after obtaining the score vectors. 2-Freedom to use any combination of labels for merging. 3-Robust against biased training sets (as each combination of labels have their own weight matrices, so even if some weight matrices are trained more than others, it just means the predictor will perform better predicting those labels). 4-The fastest of the three models.	The optimization process returns score vectors only not predictor model. Chaining models (score vectors then regressor then classifier) leaks some information, thus reducing the prediction accuracy.
Selective Waves (2) 260 sec/fold 81.8%	1-Freedom to use any combination of labels for merging. 2-Robust against biased training sets. 3-Allows providing the predictor with partial information about the testing instance for improved performance. 4-Best accuracy of the three models.	Slowest of the three models.
Quadro-bridge 30 sec/fold 80.2%	1-Simpler. 2-Freedom to change the order of layers. 3-Reasonable balance between speed and performance.	Dependant on the translator's performance. Affected by biased training sets.

Appendix

The code is available at: https://github.com/AmmarRashed/CL-WaNeSC

```
def fit(LSMR, score_vect_dicts,random_state=42, regressor=MLPRegressor(), cl
assifier=MLPClassifier()):
    LSMR["score_vec"] = LSMR["Score"].apply(
        lambda x: score_vect_dicts[x] if x in score_vect_dicts else np.NaN)
    LSMR.dropna(inplace=True)

    X, Y, y = get_XYy(LSMR)

    regressor.random_state = random_state
    classifier.random_state = random_state

    regressor.fit(X, Y)
    classifier.fit(Y, y)
    return regressor, classifier
```

Code 2 `fit` *function of the first variation of Selective Waves. "LSMR" stands for Language Score Movie Review vectors, referring to the names of the columns of the input data frame.* `get_XYy` *takes a data frame as input and returns the reviews' vectors as numpy 2d array X, score vectors as numpy 2d array Y and scores as numpy 1d array y.*

```
if W1 is None:
    W1 = 2*np.random.random((len(L1), 300, 300))-1

LSM_R = merge(L1, W1)
L2 = get_L2(LSM_R, data_dict)
if W2 is None:
    W2 = 2*np.random.random((len(L2), 300, 300))-1

LS_MR = merge(L2, W2)
L3 = get_L3(LS_MR, data_dict)
if W3 is None:
    W3 = 2*np.random.random((len(L3), 300, 300))-1

score_vectors_dict = merge(L3, W3)
l4 = sigmoid(np.array([v for k, v in sorted(score_vectors_dict.items())]))
if W4 is None:
    W4 = 2*np.random.random((300, len(LSMR)))-1

l5 = softmax(l4.dot(W4))  # predicted scores
```

Code 3 Selective Waves, first variation, Forward pass.

```
def merge(L, W):
    merged = dict()  # {item: vector of merged subitems}
    for i, item in enumerate(sorted(L)):
        for subitem in L[item]:
            merged.setdefault(item, [np.zeros(VECTOR_SIZE),0])
            merged[item][0] += sigmoid(subitem.dot(W[i]))
            merged[item][1] += 1
    for item in merged:
        merged[item] = merged[item][0]/ merged[item][1]
    return merged
```

Code 4 Selective Waves `merge` function

```
def get_layer_error(delta, W):
    error = 0
    for i in range(len(delta)):
        error += delta[i].dot(W[i].T)
    return error/len(delta)

def get_layer_delta(error, layer, size):
    delta = np.zeros((size, VECTOR_SIZE))
    j = 0
    for i,k in enumerate(sorted(layer)):
        for l in layer[k]:
            delta[j] = error[i]*sigmoid(l, True)
            j += 1
    return delta

# Calculate the error
l5_error = np.mean(np.dot(np.log(l5), y))

# Back propagation
l5_delta = l5_error * sigmoid(l5, True)
W4 += l4.T.dot(l5_delta)*alpha

l4_error = l5_delta.dot(W4.T)
l4_delta = l4_error * sigmoid(l4, True)

W3 = update_weights(L3, l4_delta, W3, alpha)

l3_error = get_layer_error(l4_delta, W3)
l3_delta = get_layer_delta(l3_error, L3, len(L2))

W2 = update_weights(L2, l3_delta, W2, alpha)

l2_error = get_layer_error(l3_delta, W2)
l2_delta = get_layer_delta(l2_error, L2, len(LSMR))

W1 = update_weights(L1, l2_delta, W1, alpha)
```

Code 5 Backpropagation

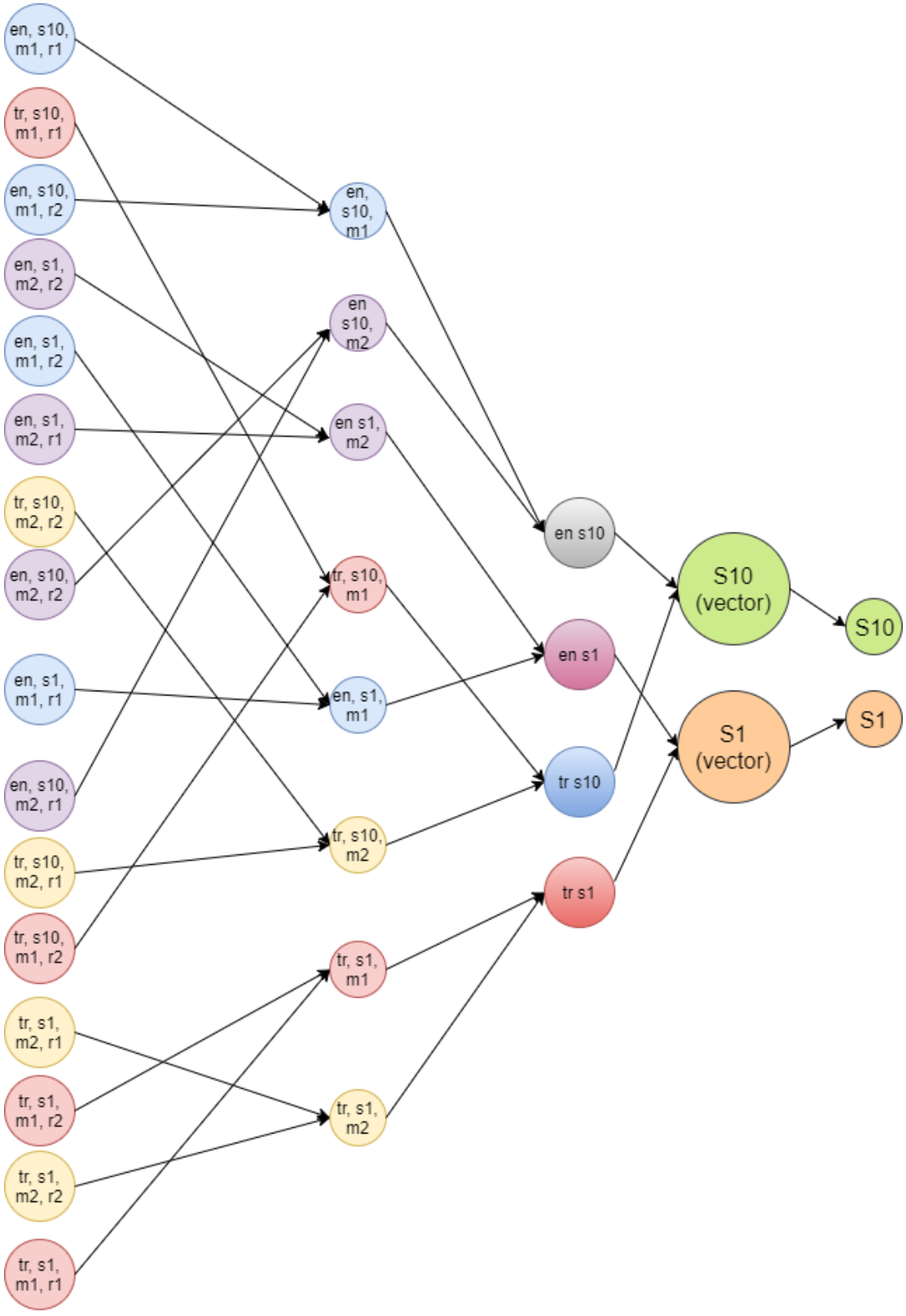

Figure 6 Training Selective Waves. S_i means a score of i. m_x means a movie with id x. en means English and tr means Turkish. r_y means the review at index y of movie x, with score I, and language l (i.e en or tr). Nodes whose outgoing edges meet in the same node in the next layer correspond to multiplying by the same weight matrix.

```
def train_selective(df_train,epochs=100, learning_rate = 0.1, random_state=42):

    LSMR_train = preprocess_data(df_train)
    np.random.seed(random_state)
    data_dict, L1, L2, L3 = get_data_dict(LSMR_train, get_L2and3=True)
    init_weights = lambda layer, i, o: {k:2*np.random.random((i, o))-1 for k in layer}
    W1 = init_weights(L1, 300, 300)  # (languge, score, movie_id)
    W2 = init_weights(L2, 300, 300)  # (languge, score):
    W3 = init_weights(L3, 300, 10)  # score:
    reset_graph()
    x = tf.placeholder(tf.float32, [None, 300])
    y = tf.placeholder(tf.float32, [None, 10]) # 1-10 => 10 classes
    w1 = tf.placeholder(tf.float32, [300, 300])
    w2 = tf.placeholder(tf.float32, [300, 300])
    w3 = tf.placeholder(tf.float32, [300, 10])

    b1 = tf.Variable(tf.zeros([300]))
    b2 = tf.Variable(tf.zeros([300]))
    b3 = tf.Variable(tf.zeros([10]))

    l2 = tf.nn.sigmoid(tf.matmul(x, w1) + b1)
    l3 = tf.nn.sigmoid(tf.matmul(l2, w2) + b2)
    pred = tf.nn.softmax(tf.matmul(l3, w3) + b3)

    cost = tf.reduce_mean(-tf.reduce_sum(y*tf.log(pred), reduction_indices=1))
    optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)
    with tf.device('/job:localhost/replica:0/task:0/device:GPU:0'):
        with tf.Session(config=tf.ConfigProto(log_device_placement=True)) as sess:
            sess.run(tf.global_variables_initializer())
            for e in range(epochs+1):
                avg_cost = 0.
                for _, row in LSMR_train.iterrows():
                    lang = row["Language"]
                    movie_id = row["Movie_ID"]
                    score = row["Score"]
                    y_ = np.zeros(10)
                    y_[score-1] = 1
                    y_ = np.atleast_2d(y_)
                    x_ = np.atleast_2d(row["rev_vec"])
                    w1_,w2_,w3_,_, c = sess.run([w1, w2, w3, optimizer, cost],
                                                feed_dict={x: x_,
                                                           y: y_,
                                                           w1:W1[(lang, score, movie_id)],
                                                           w2:W2[(lang, score)],
                                                           w3:W3[score]})
                    W1[(lang, score, movie_id)] = w1_
                    W2[(lang, score)] = w2_
                    W3[score] = w3_

                    avg_cost += c
```

Code 6 Training Selective Waves with TensorFlow

```
def predict(LSMR, score_vect_dicts, regressor, classifier):
    LSMR["score_vec"] = LSMR["Score"].apply(
lambda x: score_vect_dicts[x] if x in score_vect_dicts else np.NaN)
    LSMR.dropna(inplace=True)

    X, Y, y = get_XYy(LSMR)

    preds_score_vecs = regressor.predict(X)
    pred_scores = classifier.predict(preds_score_vecs)

    return pred_scores, y
```

Code 7 Predict function of the first variation of Selective Waves

```
def predict_selective(df, W1, W2, W3):
    LSMR = preprocess_data(df)
    reset_graph()
    x = tf.placeholder(tf.float32, [None, 300])

    w1 = tf.placeholder(tf.float32, [300, 300])
    w2 = tf.placeholder(tf.float32, [300, 300])
    w3 = tf.placeholder(tf.float32, [300, 10])

    b1 = tf.Variable(tf.zeros([300]))
    b2 = tf.Variable(tf.zeros([300]))
    b3 = tf.Variable(tf.zeros([10]))

    l2 = tf.nn.sigmoid(tf.matmul(x, w1) + b1)
    l3 = tf.nn.sigmoid(tf.matmul(l2, w2) + b2)
    pred = tf.nn.softmax(tf.matmul(l3, w3) + b3)

    prediction = tf.argmax(pred, 1)
    preds = np.zeros(len(LSMR))
    with tf.device('/job:localhost/replica:0/task:0/device:GPU:0'):
        with tf.Session(config=tf.ConfigProto(log_device_placement=True)) as sess:
            sess.run(tf.global_variables_initializer())
            j = 0
            for _, row in LSMR.iterrows():
                v = row["rev_vec"]
                predicted_scores = np.zeros(len(W1))
                for i, info in enumerate(W1):
                    language, score, movie_id = info
                    w_1 = W1[(language, score, movie_id)]
                    w_2 = W2[(language, score)]
                    w_3 = W3[score]

                    predicted_scores[i] = prediction.eval({x: np.atleast_2d(v),
                                                           w1:w_1,w2:w_2,w3:w_3})

                max_index, probability = get_max_index(softmax(predicted_scores))
                predicted_score = predicted_scores[max_index]

                preds[j] = predicted_score
                j+=1
    return preds, np.array(list(LSMR.Score))
```

Code 8 Predict function of the second variation of Selective Waves

```
def get_data_dict(df_train):
    data_dict = dict()  # {(score, movie_id): {language: [reviews]}}
    for _, row in df_train.iterrows():
        lang = row[0]
        movie_id = row[1]
        review = row[2]
        score = row[3]

        data_dict.setdefault((score, movie_id), dict())
        data_dict[(score, movie_id)].setdefault(lang, list())
        data_dict[(score, movie_id)][lang].append(review)
    return data_dict
```

Code 9 Quadro-bridge data dictionary construction

```
cost1 = tf.reduce_sum(tf.square(tf.nn.sigmoid(tf.matmul(X1, W1) + b1 - X2)))
cost2 = tf.reduce_sum(tf.square(tf.nn.sigmoid(tf.matmul(X2, W2) + b2 - X3)))
cost3 = tf.reduce_sum(tf.square(tf.sigmoid(tf.matmul(X3, W3) + b3 - X4)))
cost4 = tf.reduce_mean(-tf.reduce_sum(Y*tf.log(pred), reduction_indices=1))
cost = cost1 + cost2 + cost3 + cost4

optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)
```

Code 10 Quadro-bridge cost function

```
X1_, X2_, X3_, X4_, Y_ = get_training_batch(data_dict)
assert len(X1_) == len(X2_) == len(X3_) == len(X4_) == len(Y_)
with tf.device('/job:localhost/replica:0/task:0/device:GPU:0'):
    with tf.Session(config=tf.ConfigProto(log_device_placement=True)) as se
ss:
        sess.run(tf.global_variables_initializer())
            for e in range(epochs):
                avg_cost = 0.
                for i in range(len(X1_)):
                    _, c, w1, w2, w3, w4 = sess.run([optimizer, cost,
                                                     W1, W2, W3, W4],
                                feed_dict={X1: np.atleast_2d(X1_[i]),
                                           X2: np.atleast_2d(X2_[i]),
                                           X3: np.atleast_2d(X3_[i]),
                                           X4: np.atleast_2d(X4_[i]),
                                           Y: np.atleast_2d(Y_[i])})
                    avg_cost += c
                avg_cost /= len(df_train)
return w1, w2, w3, w4
```

Code 11 `fit` *function of Quadro-bridge*