Semi-supervised Grouping of Spoken Words Using an Autoencoder

Andrew Vaughn

University of Texas at Dallas

ACN 6389.501

Introduction

Creating an automatic speech recognition (ASR) is still a fairly difficult task, requiring a large amount of labeled training data. Even so, there are many speech recognition systems that we interact with daily through smart devices, mainly virtual assistants. These ASR systems are computationally intense and must send speech to a server to process the audio accurately (Wu, Yilmaz, Zhang, Li, & Tan, 2020). The only onboard speech processing they implement is recognition of key words or phrases to know when the user is addressing them. Because of this, most ASR devices require an internet connection to function. ASR can also be limited by vocabulary, as Artificial Neural Networks (NNs) have a predetermined number of output nodes, each representing a single word. This limitation makes it difficult to expand a network's vocabulary without retraining.

The goal of this project is to explore using an autoencoder to generate word embeddings from audio, using input-output pairs that correspond to different recordings of the same word. Ideally, the embedded vectors would be grouped together, multiple recordings of the same word would produce similar vectors. This would allow the ANN some degree flexibility in its vocabulary, which would be represented by clusters in the embedded space rather than by a set number of output nodes. The utility of this would not necessarily be to identify the word being spoken, but as a language representation model. Once trained, the model could be used as a preprocessing step for other networks.

Background

In machine learning, there are two main types of learning, supervised and unsupervised. Supervised learning requires labeled data, an input and a desired output which the ANN learns to produce from a given input. This is useful in categorization tasks. By contrast, unsupervised learning does not require labels. Unsupervised learning is useful for identifying patterns in the training data, such as clustering, latent factor analysis, and word embeddings. This can be incredibly useful when dealing with large amounts of unlabeled training data, as it can reveal patterns in the data. Autoencoders are just one example of an unsupervised learning algorithm.

Artificial Neural Networks, and especially deep neural networks (DNNs), prove to be incredibly useful for language and speech processing (Deng, Hinton, & Kingsbury, 2013). An autoencoder (*Figure 1.*) is an ANN which uses the same feature vector as both the input and output. Autoencoders are not restricted in the types or number of hidden layers they contain and can be constructed the same as any other ANN, but where the goal of other networks is to learn to produce some output from an input, the

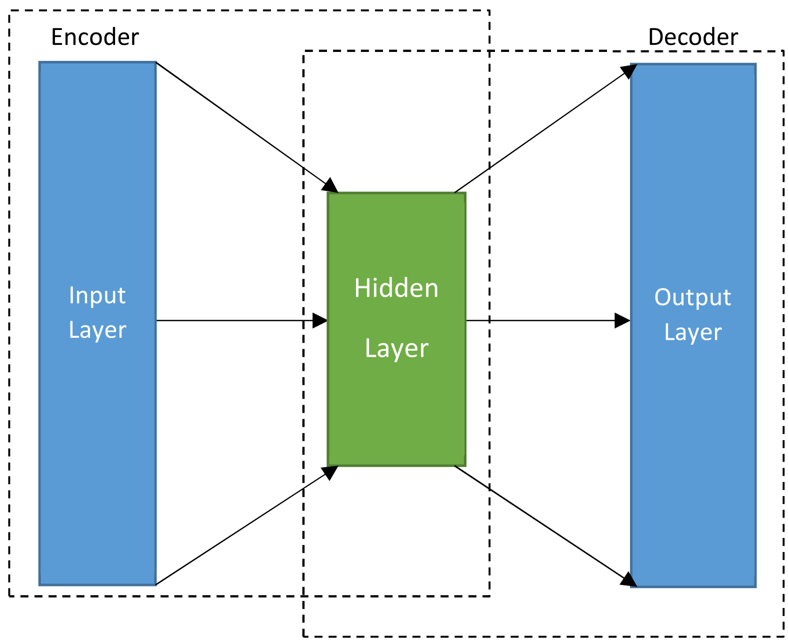

Figure 1: Autoencoder design

The encoder transforms a feature vector into an embedded vector(activation of the hidden layer) while the decoder converts the embedded vector back into a feature vector.

goal of an autoencoder is to learn the subspace spanned by the inputs. They are often used for data compression and noise reduction but also show promise for clustering (Xie, Girshick, & Farhadi, 2016). Autoencoders consist of two main parts, the Encoder and Decoder. The Encoder consists of the input layer, the embedded layer, and all layers in between. The Decoder consists of the embedded layer, the output layer, and all layers in between. The network simultaneously learns to encode and decode the input to and from the embedded layer, respectively. The embedded layer should be smaller than the input and output layer.

Methods

Training Data

The audio data was obtained from Mozilla's Common Voice project (Mozilla, 2020), which consisted of transcripts and recordings of 61,528 voices. The audio was then processed using Gentle forced aligner (Lowerquality, 2017) to retrieve the timestamps of individual words. All timing data was then imported into a MySQL database for efficient sorting and retrieval. All clips of the top ten most occurring words (*Appendix B.)* were imported from the database into Matlab, resulting in 30,295 clips. Each clip had the first three formant frequencies measured on fifteen windows with 80% overlap before being normalized to create the feature vectors. Each feature vector was then randomly paired with 200 other feature vectors of the same word. This resulted in a set of 6,059,000 pairs of feature vectors, which were sampled from during training.

Network Design

The autoencoder is a shallow network, with on layer of ten sigmoidal hidden units and forty-five input and output units. The network was trained in batches of ten training pairs, and a grouping analysis was performed every 10 batches. A means-squared error was used as the loss function of the network. During training, the network was exposed to a total of 500,000 pairs.

Analyzing Network Performance

Performance of the network was measured by the grouping factor (GF), calculated by dividing the average distance between embeddings of the same word by the average distance of embeddings of different words. Because this was computationally intense for large data sets, the calculation during training was performed on a randomized sample of the full data set. As the network trains, the GF was expected to increase, since tight clusters would have a low average distance within-groups and a high average distance between-groups. The loss of the network, usually used to determine how accurately the output is being predicted, was expected to decrease throughout training, but not very much, as each input corresponds to multiple different outputs. A t-Distributed Stochastic Neighborhood Embedding was also performed on the feature vectors as well as the embeddings before and after training to try and visualize any clustering that arose.

Results

The loss for the network rapidly dropped after the first several batches but did not decrease much throughout the rest of training. This is probably because all feature vectors have small values, so the network learns to produce outputs with small values. The grouping factor starts off greater than

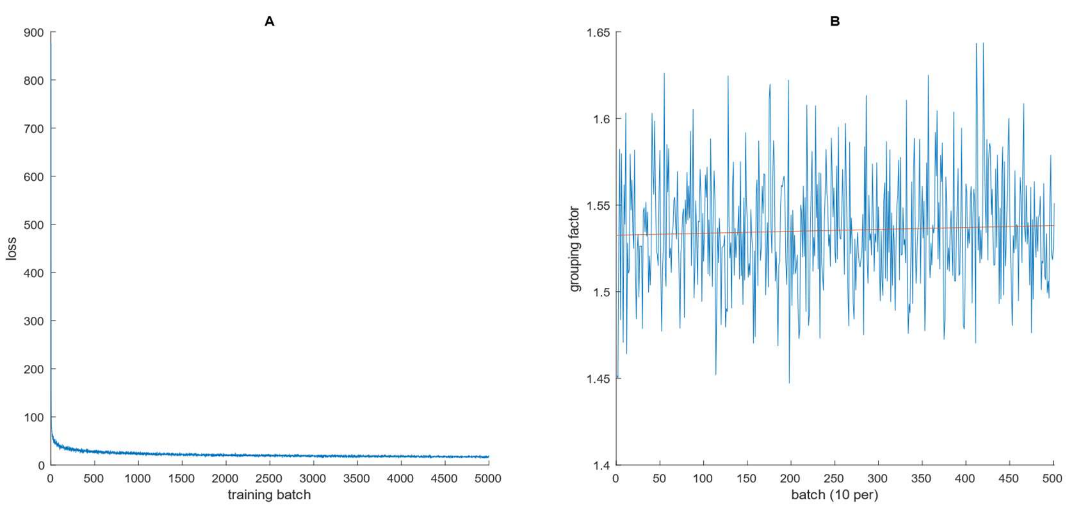

Figure 2: Loss and Grouping Factor

The network's loss (A) and the grouping factor (B). The grouping factor is plotted with a 1^{st} degree polynomial line of best fit (slope < .001, intercept = 1.5326) to more clearly show the overall trend as the network learns.

one, indicating that embeddings of the same word are already closer to each other than they are to embeddings of other words. However, the GF does not increase much throughout training either, indicating that the network is not learning achieving better clustering of words by training.

Additionally, the TSNE does not show much differentiation between groups after training the network either (Appendix A: fig 4 and 5).

Discussion

Limitations

While the network is small and retrieving the embedding of an input is efficient, there was a lot of overhead due to the amount of data and computing the GF, slowing down analysis of the network's performance. The network's size was also a limiting factor, with only one layer of hidden units. Furthermore, most of the values used (number and types of features, learning rate, number of hidden units, etc.) were chosen based on empirical observations of small-scale tests involving the ANN. The network also only used ten words for training and was not tested on a different set of words. The words are all short, which could have made it more difficult for the network to separate them.

Conclusion and Future Directions

While the autoencoder did not achieve a high level of clustering, there were many limiting factors that may have played a part in that. This was a relatively small exploration of using an autoencoder for clustering. Only one shallow ANN was fully tested but performance may be improved by adding more layers, especially convolutional layers. The network also used sigmoidal activation units, but rectified linear units (RELU) have been found to work well for language processing (Deng et al., 2013). The number and types of features could also be further explored, as well as using longer words which may be more distinct. The grouping factor did show a very slight improvement in the final network and alterations to the training data and network architecture may achieve a more drastic change.

References

Deng, L., Hinton, G., & Kingsbury, B. (2013). *New types of deep neural network learning for speech recognition and related applications: an overview*. Paper presented at the 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, Vancouver, BC, Canada.

Lowerquality. (2017). Gentle [GitHub repository]. Github. Retrieved from https://github.com/lowerquality/gentle

Mozilla. (2020). cv-corpus-5.1-2020-06-22 (Audio files and transcripts). from Mozilla https://commonvoice.mozilla.org/en/datasets

Wu, J., Yilmaz, E., Zhang, M., Li, H., & Tan, K. C. (2020). Deep Spiking Neural Networks for Large Vocabulary Automatic Speech Recognition. *Front Neurosci, 14*, 199. doi:10.3389/fnins.2020.00199

Xie, J., Girshick, R., & Farhadi, A. (2016). *Unsupervised Deep Embedding for Clustering Analysis*. arXiv.org. Retrieved from https://arxiv.org/abs/1511.06335v2

Appendix A

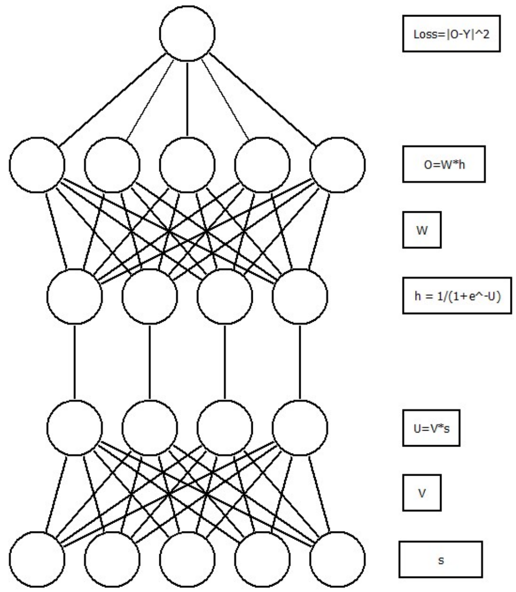

Figure 3: Network Diagram

The layout of the network and the mathematical formulas for each layer

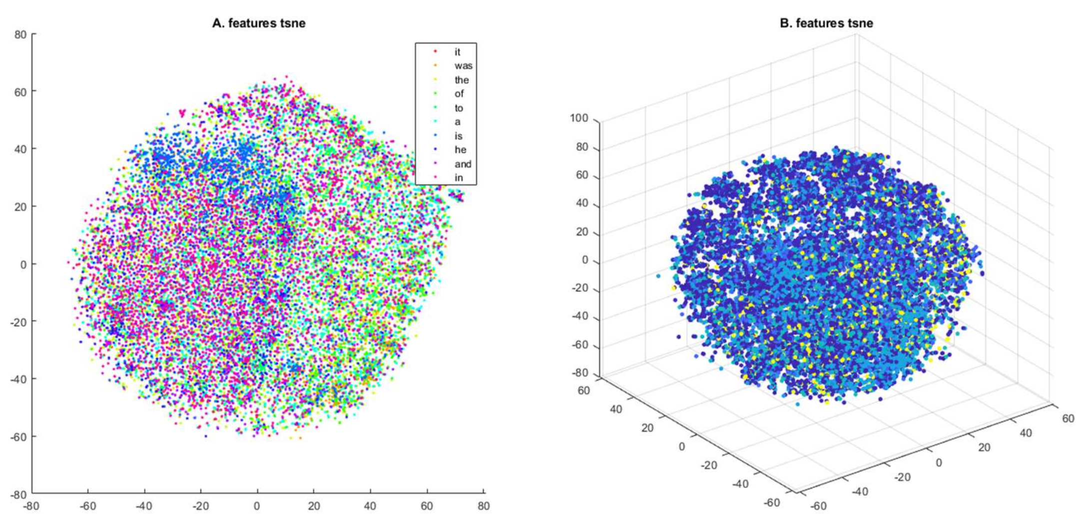

Figure 4: Feature TSNE

2-dimensional (left) and 3-dimensional (right) t-Distributed Stochastic Neighborhood Embedding performed on the feature vectors

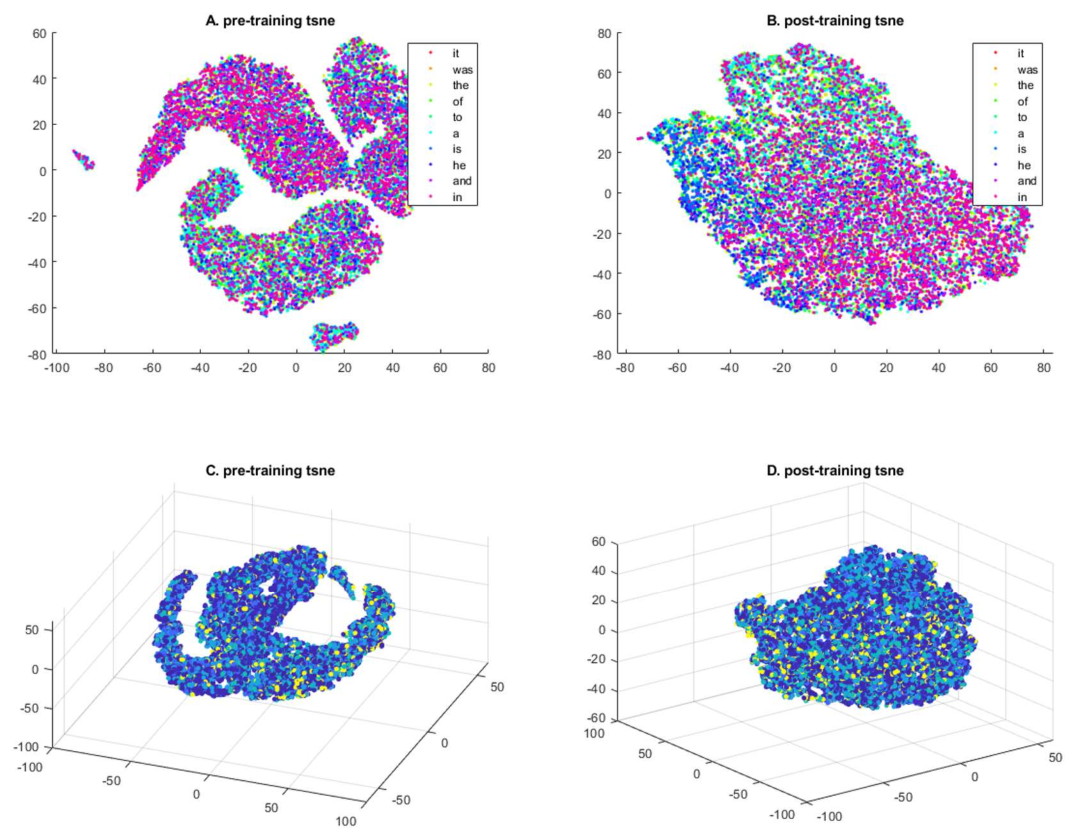

Figure 5: Embeddings TSNE

2-dimensional (top) and 3-dimensional (bottom) t-Distributed Stochastic Neighborhood Embedding performed on the embedding vectors before (left) and after (right) training the autoencoder

Appendix B

Training word list:

It
Was
The
Of
To
A
Is
He
And
In

Appendix C

load_data.m

```
num_pairs = 500;
resample_rate = 48000;
num_windows = 15;
window_overlap = 0.8;

% Prep the training data
inputs = {};
for i = 1:numel(data(:,1))
    f1 = [''A:\cv-corpus-5.1-2020-06-22\en\'clips\', char(data.record_id(i)),
'.mp3'];
    s1 = process(f1, data.start(i), data.xEnd(i), resample_rate, num_windows,
window_overlap);
    if ~isempty(s1)
        inputs{end+1} = [data.word_id(i); s1'];
    end
    if mod(i, 100) == 0
        disp([num2str(i), '/', num2str(numel(data(:,1)))]);
    end
end

num_clips = numel(inputs);

training_data = {};
for i = 1:num_clips
    c1 = cell2mat(inputs(i));
    n = 0;
    for j = 1:num_clips
        c2 = cell2mat(inputs(j));
        if c1(1) == c2(1) && n < num_pairs
            training_data{end+1} = [c1; c2(2:3*num_windows+1,1)];
            n = n + 1;
        end
    end
end
cd('C:\Users\andre\OneDrive\SCHOOL\2020 Fall\Speech Perception Lab');

td = cell2mat(training_data)';
td(:,2:6*num_windows+1) = td(:,2:6*num_windows+1)./max(max(td));

training_set_size = size(training_data);

inputs = cell2mat(inputs)';
inputs(:,2:3*num_windows+1) = inputs(:,2:3*num_windows+1)./max(max(inputs));
```

```
function input_data = process(file, start_time, end_time, resample_rate,
num_windows, window_overlap)
    try
        [y, fs] = audioread(file);
        y = resample(y, resample_rate, fs);
        % Get the sample numbers for the beginning and ending of the clip
        t1 = floor(start_time * resample_rate);
        t2 = floor(end_time * resample_rate);
        % Check that the provided clip fits within the audio file
        x = size(y(:,1));
        fit = t2 < x(1);
        clip_length = (t2-t1);
        if fit(1) && clip_length < 2 * resample_rate
            clip = y;
            window_size = floor(clip_length / num_windows);
            formants = [];
            % Compute formant frequencies for each window
            for t = 1:num_windows
                w1 = floor((t-1)*window_size) + t1;
                w2 = floor(min(w1 + window_size * (1+window_overlap), x));
                win = clip(w1:w2);
                formants(:,t) = formant_measure(win, resample_rate);
            end
            input_data = reshape(formants, [1, numel(formants)]);
        else
            input_data = [];
        end
    catch
        input_data = [];
    end
end

function formants = formant_measure(x, fs)
    preemph = [1 0.63];
    x = filter(1, preemph, x);
    A = lpc(x,8);
    rts = roots(A);
    rts = rts(imag(rts)>=0);
    angz = atan2(imag(rts),real(rts));
    [frqs,indices] = sort(angz.*(fs/(2*pi)));
    bw = -1/2*(fs/(2*pi))*log(abs(rts(indices)));
    nn = 1;
    for kk = 1:length(frqs)
        if (frqs(kk) > 90 && bw(kk) <400)
            formants(nn) = frqs(kk);
            nn = nn+1;
        end
    end
    formants = round(frqs(1:3));
end
```

Init_nn.m

```
rng default

nn = neuralNetwork(45, 10, 45);

% Obtain the reduced vector for each clip before training the network
reduced_vectors_pre_training = [];
for r = 1:num_clips
    reduced_vectors_pre_training = [reduced_vectors_pre_training;
nn.reduce(inputs(r,2:46))];
end

reduced_vectors_pre_training = [inputs(:,1) reduced_vectors_pre_training];

nn.add_grouping_factor(analyze_grouping_factor(reduced_vectors_pre_training))
;
```

get_reduced_vectors.m

```
function reduced = get_reduced_vectors(x, nn)
    % Obtain the reduced vector for each clip after training the network
    reduced = [];
    for r = 1:numel(x(:,1))
        reduced = [reduced; nn.reduce(x(r,2:46))];
    end

    reduced = [x(:,1) reduced];
end
```

train.m

```
rng default
warning('off','MATLAB:rankDeficientMatrix');

iterations = 50000; % Number of examples to train the model on
batch_size = 10; % Number of examples in each batch
analysis = 100; % How many batches to train before computing analysis

% Train the network
gf = nn.grouping_factor(end);
gamma = [0.1];
for i = 1:batch_size:iterations
    input = zeros(batch_size,nn.input_size);
    label = zeros(batch_size,nn.input_size);
    for j = 1:batch_size-1 % Randomly sample the training data
        row = datasample(td, 1);
        input(j,:) = row(2:46);
        label(j,:) = row(47:91);
    end

    nn.train(input, label, gamma(end), batch_size);

    if mod(i-1,analysis*batch_size) == 0 % Analyze the grouping
        disp(['batch: ' , num2str((i-1)/batch_size), '/',
num2str(floor(iterations/batch_size))]);
        reduced_vectors_post_training = get_reduced_vectors(inputs, nn);
        gf = nn.grouping_factor(end);

nn.add_grouping_factor(analyze_grouping_factor(reduced_vectors_post_training)
);
    end
    gamma = [gamma 0.1/i];
end

figure(1)
plot(nn.loss, 'blue');
hold on
plot(gamma, 'red');
hold off
title('loss (blue) and learning rate (red)');

figure(2);
plot(nn.grouping_factor);
title('grouping factor');
```

neuralNetwork.m

```
classdef neuralNetwork < handle
    properties
        input_size;
        hidden_units;
        output_size;
        eyeoutput;
        eyehidden;
        V;
        W;
        input_activation;
        hidden_activation;
        output_activation;
        loss = [];
        grouping_factor = [];
    end
    methods (Static)
        function self = neuralNetwork(input_size, hidden_units, output_size)
            self.input_size = input_size;
            self.hidden_units = hidden_units;
            self.output_size = output_size;
            self.V = rand(hidden_units, input_size);
            self.W = rand(output_size, hidden_units);
        end
        function[test] = test()
            test = 1;
        end
    end
    methods
        function s = sigmoid(self, x)
            s = (1./(1+exp(-x)))';
        end
        function ds = d_sigmoid(self, x)
            ds = (x.*(1-x));
        end
        function output = feedForward(self, input)
            self.input_activation = input;
            vs = self.V * input;
            self.hidden_activation = self.sigmoid(vs);
            self.output_activation = self.W * self.hidden_activation';
            output = self.output_activation;
        end
        function reduced = reduce(self, input)
            vs = self.V * input';
            reduced = self.sigmoid(vs);
        end
        function dcdy = dc_dy(self, label, output)
            dcdy = -2*(label'-output');
        end
        function dydw = dy_dw(self)
            for k = 1:self.output_size
                uk = zeros(self.output_size, 1);
                uk(k) = 1;
                dydw{k} = uk*self.hidden_activation;
            end
        end
```

```
        function grad_w = grad_W(self, label, output)
            grad_w = zeros(self.output_size, self.hidden_units);
            dcdy = self.dc_dy(label, output);
            dydw = self.dy_dw();
            for k = 1:self.output_size
                grad_w(k,:) = grad_w(k,:) + dcdy * dydw{k};
            end
        end
        function dydh = dy_dh(self)
            dydh = self.W;
        end
        function dhdv = dh_dv(self, input)
            for j = 1:self.hidden_units
                uj = zeros(self.hidden_units, 1);
                uj(j) = 1;
                hj = self.hidden_activation(j);
                dhdv{j} = uj*self.d_sigmoid(hj)*input';
            end
        end
        function grad_v = grad_V(self, input, label, output)
            grad_v = zeros(self.hidden_units, self.input_size);
            dcdy = self.dc_dy(label, output);
            dydh = self.dy_dh();
            dhdv = self.dh_dv(input);

            for j = 1:self.hidden_units
                grad_v(j,:) = grad_v(j,:) + dcdy*dydh*dhdv{j};
            end
        end
        function loss = train(self, inputs, labels, gamma, batch_size)
            loss = 0;
            grad_w = zeros(self.output_size, self.hidden_units);
            grad_v = zeros(self.hidden_units, self.input_size);

            for i = 1:batch_size
                input = inputs(i,:)';
                label = labels(i,:)';
                output = self.feedForward(input);
                loss = loss + (label-output)'*(label-output);

                grad_w = grad_w + self.grad_W(label, output);

                grad_v = grad_v + self.grad_V(input, label, output);
            end

            self.loss(end+1) = loss / batch_size;
            self.W = self.W - gamma * grad_w / batch_size;
            self.V = self.V - gamma * grad_v / batch_size;
        end
        function add_grouping_factor(self, gf)
           self.grouping_factor = [self.grouping_factor; gf];
        end
    end
end
```

analyze_grouping_factor.m

```
function grouping_factor = analyze_grouping_factor(v)
    % For each clip, find the average distance to other clips of the same
word
    avg_within_group_distances = [];
    avg_between_group_distances = [];
    for i = 1:numel(v(:,1))
        c1 = v(i,:);
        within_group = {};
        between_group = {};

        for j = 1:numel(v(:,1))
            c2 = v(j,:);

            d = sqrt(c1*c2');

            if c1(1) == c2(1)
                within_group = [within_group; d];
            else
                between_group = [between_group; d];
            end
        end
        avg_within_group_distances = [avg_within_group_distances;
mean(cell2mat(within_group))];
        avg_between_group_distances = [avg_between_group_distances;
mean(cell2mat(between_group))];
    end

    grouping_factor =
mean(avg_between_group_distances./avg_within_group_distances);
end
```

plot_data.m

```
rng default

features = tsne(inputs(:,2:46), 'NumDimensions',2);
pre = tsne(reduced_vectors_pre_training(:,2:end), 'NumDimensions',2);
post = tsne(reduced_vectors_post_training(:,2:end), 'NumDimensions',2);

figure(1)
plot(nn.loss)
title('Loss');
figure(2)
gscatter(features(:,1),features(:,2),inputs(:,1))
title('features tsne');
figure(3)
gscatter(pre(:,1),pre(:,2),inputs(:,1))
title('pre-training tsne');
figure(4)
gscatter(post(:,1),post(:,2),inputs(:,1))
title('post-training tsne');

features = tsne(inputs(:,2:31), 'NumDimensions',3);
pre = tsne(reduced_vectors_pre_training(:,2:end), 'NumDimensions',3);
post = tsne(reduced_vectors_post_training(:,2:end), 'NumDimensions',3);

figure(5)
scatter3(features(:,1),features(:,2),features(:,3),15,inputs(:,1),'filled')
title('features tsne');
figure(6)
scatter3(pre(:,1),pre(:,2),pre(:,3),15,inputs(:,1),'filled')
title('pre-training tsne');
figure(7)
scatter3(post(:,1),post(:,2),post(:,3),15,inputs(:,1),'filled')
title('post-training tsne');

grouping_factor_pre_train = 0;
grouping_factor_post_train = 0;

avg_distance_pre = [];
avg_distance_post = [];
```