pp_trigram_dict-diff-170308

March 8, 2017

1 Hypervector algorithms for simultaneous one-shot learning and generalization

Inspired by the work of Rumelhart and McCleland, we developed a neural network system for solving the challenge of learning the rules that govern the transformation of present-tense verbs to past-tense.

We utilize hypercomputing to formalize the algorithm that they proposed, including how to represent "Wickelfeatures" and use a neural network to learn the transformation from the present-tense Wickelfeature representation to the past-tense.

With advances in hypercomputing theory, we are able to go beyond the previous work and achieve three notable goals: 1. Perform one-shot learning on the training set. 2. Create a system that can also generalize to words in the test set. 3. Infer the letter sequence of the word from the Wickelfeature representation.

Goal 3 presents a fundamental challenge of performing a combinatoric search, and was a desired aspect of the original work but never realized. Because it is such a challenge to infer the word from the Wickelfeature representation, validation of the learning in goals 1/2 is difficult. To validate the performance of their original network, Rumelhart and McCleland examined the similarity of the abstract high-level Wickelfeature representations of the network's output to the correct words representation. Although this is indirect, comparison in the abstract feature space is still useful to understand the network's performance.

In this section, we compare several variations of a hypercomputing algorithm that can perform one-shot learning of the present to past transformation. We use the abstract representation of Wickelfeatures to represent each word in this generalizable abstract feature space, and use the binding mechanism to create a content-addressable dictionary of present-tense to past-tense words. By representing the words in a generalizable feature space, we are really creating a dictionary of word parts and mapping common present-tense word parts to common past-tense word parts. This potentially allows the dictionary to highlight common mappings between word parts, and can enable generalization of the transformation to novel verbs.

```
In [1]: from __future__ import division

        from pylab import *
        import scipy
        import time

        import sklearn
        from sklearn.decomposition import PCA, FastICA, TruncatedSVD, NMF
```

```
import utils
import hrr_utils

import string

from scipy import spatial
from matplotlib.collections import LineCollection
%matplotlib inline

plt.rcParams.update({'axes.titlesize': 'xx-large'})
plt.rcParams.update({'axes.labelsize': 'xx-large'})
plt.rcParams.update({'xtick.labelsize': 'x-large', 'ytick.labelsize': 'x-la
plt.rcParams.update({'legend.fontsize': 'x-large'})
plt.rcParams.update({'text.usetex': True})
```

In [52]:
```
N=30000
```

In [53]:
```
alphabet = string.ascii_lowercase + "#."

def ngram_encode_cl(ngram_str, letter_vecs, window=3):
    vec = np.zeros(letter_vecs.shape[1])
    full_str = '#' + ngram_str + '.'
    for il, l in enumerate(full_str[:-(window-1)]):
        trivec = letter_vecs[alphabet.find(full_str[il]), :]
        for c3 in range(1, window):
            trivec = trivec * np.roll(letter_vecs[alphabet.find(full_str[i
        vec += trivec
    return 2* (vec + 0.1*(np.random.rand(letter_vecs.shape[1])-0.5) < 0) -



def genX(verbs):
    X = np.zeros((len(verbs), N)) # Exclusively difference PAST1-PRES1
    PRES1 = np.zeros((len(verbs), N))
    PRES2 = np.zeros((len(verbs), N))
    PAST1 = np.zeros((len(verbs), N))
    PAST2 = np.zeros((len(verbs), N))

    for m, pair in enumerate(verbs):
        past1 = ngram_encode_cl(pair[1], dic1, 3)
        past2 = ngram_encode_cl(pair[1], dic2, 3)
        pres1 = ngram_encode_cl(pair[0], dic1, 3)
        pres2 = ngram_encode_cl(pair[0], dic2, 3)
        PRES1[m] = pres1
        PRES2[m] = pres2
        PAST1[m] = past1
        PAST2[m] = past2
```

```
        #X = np.where(PAST1-PRES1 > 0, 1, -1)
        X = PAST1-PRES1
        return X, PRES1, PRES2, PAST1, PAST2

    def train(tv, past, present):
        tv += np.multiply(past, present)
        return tv

    def reg_train(tv, past, present):
        pred = np.multiply(tv, present)
        #pred = np.where(pred>0, 1, -1)
        #print (sim(pred, past)),
        tv += ((N-sim(pred, past))/float(N)) * np.multiply(past, present)
        return tv

    def train_diff(tv, past2, present1, present2):
        tv += np.multiply(present1, past2-present2)
        return tv

    def reg_train_diff(tv, past2, present1, present2):
        pred = np.multiply(tv, present1) + present2
        #pred = np.where(pred>0, 1, -1)
        #print (sim(pred, past)),
        tv += ((N-sim(pred, past2))/float(N)) * np.multiply(past2-present2, pr
        return tv

    def sim(x, y):
        if len(x.shape) == 1 or len(y.shape)==1:
            return np.dot(x, y)
        return np.sum(np.multiply(x, y), axis=1)

    def graph(x, y1, y2, y3, title=None):

        fig = plt.figure()
        plt.plot(x, y1, label="train")
        plt.plot(x, y2, label="test")
        plt.plot(x, y3, label="random")
        fig.suptitle(title, fontsize=20)
        plt.xlabel('number of words', fontsize=18)
        plt.ylabel('average dot product', fontsize=16)
        plt.legend(loc='lower right')
        plt.show()
        #fig.savefig('test.jpg')
```

In [54]:
```
reg_pres, reg_past, reg_freq = utils.GetRegularVerbs(frequency=1)
irreg_pres, irreg_past, irreg_freq = utils.GetIrregularVerbs(frequency=1)
```

```
regular = zip(reg_pres, reg_past, reg_freq)
irregular = zip(irreg_pres, irreg_past, irreg_freq)

train_frac = 2./3.

train_reg = regular[0:int(train_frac*len(regular))]
train_irreg = irregular[0:int(train_frac*len(irregular))]


test_reg = regular[int(train_frac*len(regular)):]
test_irreg = irregular[int(train_frac*len(irregular)):]

#dic1 = hrr_utils.GenerateDefaultDictionary(N)
#dic2 = hrr_utils.GenerateDefaultDictionary(N)

D = len(alphabet)

dic1 = 2 * (np.random.randn(D, N) < 0) - 1
dic2 = 2 * (np.random.randn(D, N) < 0) - 1

regular.extend(irregular)
verbs = regular
```

```
In [55]: trainX, trainpres1, trainpres2, trainpast1, trainpast2 = genX(train_reg +
         testX, testpres1, testpres2, testpast1, testpast2 = genX(test_reg + test_i
```

1.1 Trigram dictionary, unregularized

In this section, we begin with the simplest algorithm that forms a content-addressable dictionary by binding present-tense verbs to past-tense verbs and holding the superposition of all verbs in the training set in memory.

```
In [56]: tst = time.time()

         psi = np.zeros(N)
         psi = train(psi, trainpast2[0], trainpres1[0])

         random_vecs = np.random.randn(testpast2.shape[0], N)

         train_hist = np.nan* np.zeros((trainpres1.shape[0], trainpres1.shape[0]))
         test_hist = np.zeros((trainpres1.shape[0], testpast2.shape[0]))
         random_hist = np.zeros((trainpres1.shape[0], testpast2.shape[0]))


         y1 = np.zeros(trainpres1.shape[0])
         y2 = np.zeros(trainpres1.shape[0])
         y3 = np.zeros(trainpres1.shape[0])
```

```
        for k in range(1,len(trainpres1)):#trainpres1.shape[0]):
            train_pred = np.multiply(psi, trainpres1[:k])
            #train_pred = np.where(train_pred>0, 1, -1)
            test_pred = np.multiply(psi, testpres1)
            #test_pred = np.where(test_pred>0, 1, -1)

            train_hist[k, :k] = sim(train_pred, trainpast2[:k])
            test_hist[k, :] = sim(test_pred, testpast2)
            random_hist[k, :] = sim(random_vecs, testpast2)

            psi = train(psi, trainpast2[k], trainpres1[k])


        print 'Elapsed: ', time.time() - tst

        fname = ('data/trigram_dict-unreg-N=' + str(N) + '-W=' + str(len(trainpres
                + '-' + time.strftime('%y%m%d') +'.npz')

        print fname
        np.savez(fname,N=N, train_hist=train_hist, test_hist=test_hist, random_his
```

```
Elapsed:  866.606426954
data/trigram_dict-unreg-N=30000-W=1758-170307.npz
```

In [57]:
```
figure(figsize=(4,3))

train_mean = np.nanmean(train_hist, axis=1)
test_mean = test_hist.mean(axis=1)
random_mean = random_hist.mean(axis=1)

train_std = np.nanstd(train_hist, axis=1) #/ (arange(len(train_mean))+1) *
test_std = test_hist.std(axis=1) #/ testpast2.shape[0]**0.5
random_std = random_hist.std(axis=1) #/ testpast2.shape[0]**0.5

ntrain_x = arange(len(train_mean))+1

plot([0, 2000], [0, 0], 'k')

plot(ntrain_x, train_mean, c='b', lw=2, label='Train')
plot(ntrain_x, test_mean, c='g', lw=2, label='Test')
#plot(ntrain_x, random_mean, c='r', lw=2, label='Random')

fill_between(ntrain_x, train_mean-train_std, train_mean+train_std, facecol
fill_between(ntrain_x, test_mean-test_std, test_mean+test_std, facecolor='
#fill_between(ntrain_x, random_mean-random_std, random_mean+random_std, fa

legend(loc='upper left')
```

```
        xlabel('Number Training Examples')
        ylabel('Feature Similarity')
        title('Dictionary, Unregularized')

        xlim([0, 2000])

        plt.tight_layout()

        fname =  ('figures/trigram_dict-unreg-N=' + str(N) + '-W=' + str(len(train
                  + '-' + time.strftime('%y%m%d'))
        print fname

        plt.savefig(fname + '.png', transparent=True)
        plt.savefig(fname + '.eps', transparent=True)
```

```
figures/trigram_dict-unreg-N=30000-W=1758-170307
```

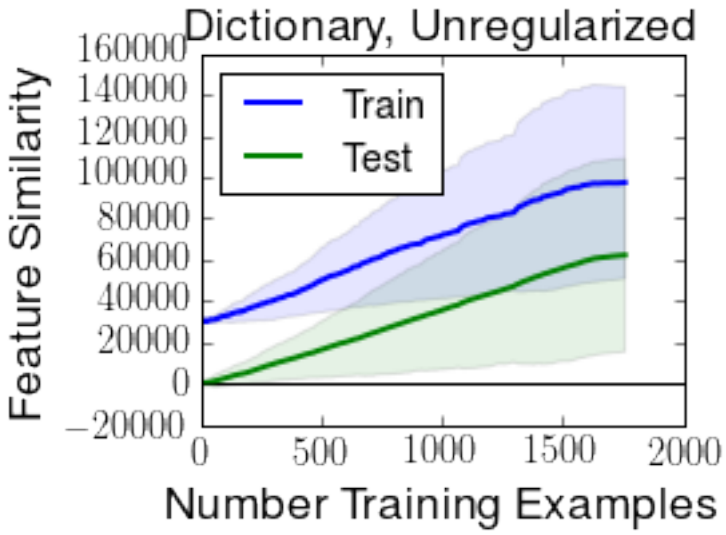


Figure 1. Wickelfeature dictionary. We evaluated the performance of the network by comparing the network's output to the Wickelfeature representation of the correct word. We see that all words in the training set have high similarity (blue line), but also that the similarity increases beyond the normal level. We also see that as more verb pairs are encountered that words in the test set also increase in similarity to the correct answer in the feature space.

This algorithm can be seen to generalize to the test set, but it is challenging to interpret its performance purely in the abstract feature space. We see that the training set similarity increases beyond the normal value, which indicates that the network is over-emphasizing some aspects of the transformation. Essentially, this increase beyond N in similarity is due to the same force that allows the network to generalize to novel verbs. As more verb pairs are added to the dictionary, then the common transformations start to stand out and get emphasized. The verbs in the training

set all contribute to the similarity of the output, and many of them are contributing the same thing – namely the regular rule of adding the trigram "ed." to create the past tense verbs. This explains the similarity going above N, as many verb-pairs are shouting the same correct feature.

1.2 Trigram dictionary, regularized

In the previous section, we see that a straight-forward dictionary created by hypercomputing binding results in over-emphasis of the common rule. Because we expect the representation of the dictionary elements to be generalizable, then there should be some common correlations that the features will emphasize. However, such a dictionary may drown-out other important aspects of the transformation to emphasize the common rule in the training data. The performance of the algorithm above can be explained by the system always emphasizing the most common transformation of adding "ed.".

Next, we use a regularization procedure akin to perceptron learning or gradient descent for which we evaluate the error of a new word in the training set prior to adding it to the dictionary. By checking first to see how much other words in the training set already explain the transformation described by the new verb-pair, we can create a system that will not simply be dominated by the most common transformation. Further, the verbs in the training set will not continuously increase beyond the natural limit for the similarity in the feature space, and thus the most common transformation will not be over-emphasized.

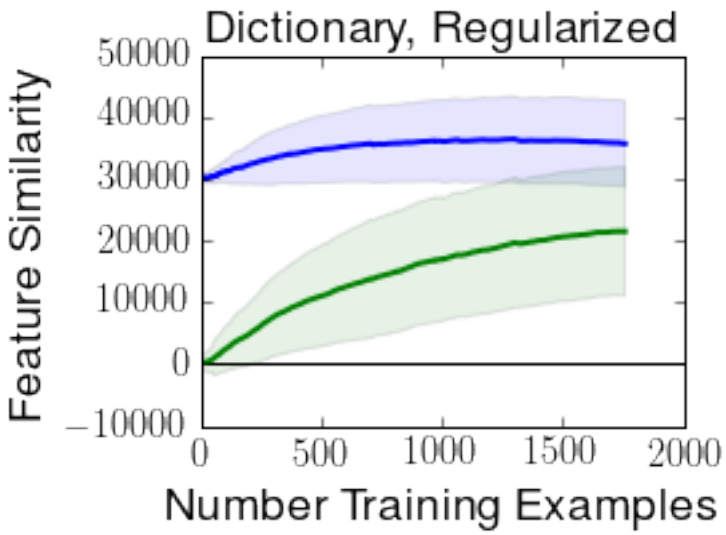
```
In [58]: tst = time.time()

         cycle=1
         psi = np.zeros(N)
         psi = reg_train(psi, trainpast2[0], trainpres1[0])

         random_vecs = np.random.randn(testpast2.shape[0], N)

         train_hist = np.nan* np.zeros((trainpres1.shape[0], trainpres1.shape[0]))
         test_hist = np.zeros((trainpres1.shape[0], testpast2.shape[0]))
         random_hist = np.zeros((trainpres1.shape[0], testpast2.shape[0]))


         for k in range(1,len(trainpres1)):#trainpres1.shape[0]):
             train_pred = np.multiply(psi, trainpres1[:k])
             #train_pred = np.where(train_pred>0, 1, -1)
             test_pred = np.multiply(psi, testpres1)
             #test_pred = np.where(test_pred>0, 1, -1)

             train_hist[k, :k] = sim(train_pred, trainpast2[:k])
             test_hist[k, :] = sim(test_pred, testpast2)
             random_hist[k, :] = sim(random_vecs, testpast2)

             psi = reg_train(psi, trainpast2[k], trainpres1[k])


         print 'Elapsed: ', time.time() - tst
```

```
        fname = ('data/trigram_dict-reg-N=' + str(N) + '-W=' + str(len(trainpres1)
                + '-' + time.strftime('%y%m%d') +'.npz')

        print fname
        np.savez(fname,N=N, train_hist=train_hist, test_hist=test_hist, random_his
```

```
Elapsed:  858.371195078
data/trigram_dict-reg-N=30000-W=1758-170307.npz
```

In [59]:
```
        figure(figsize=(4,3))

        train_mean = np.nanmean(train_hist, axis=1)
        test_mean = test_hist.mean(axis=1)
        random_mean = random_hist.mean(axis=1)

        train_std = np.nanstd(train_hist, axis=1) #/ (arange(len(train_mean))+1) *
        test_std = test_hist.std(axis=1) #/ testpast2.shape[0]**0.5
        random_std = random_hist.std(axis=1) #/ testpast2.shape[0]**0.5

        ntrain_x = arange(len(train_mean))+1

        plot([0, 2000], [0, 0], 'k')

        plot(ntrain_x, train_mean, c='b', lw=2, label='Train')
        plot(ntrain_x, test_mean, c='g', lw=2, label='Test')
        #plot(ntrain_x, random_mean, c='r', lw=2, label='Random')

        fill_between(ntrain_x, train_mean-train_std, train_mean+train_std, facecol
        fill_between(ntrain_x, test_mean-test_std, test_mean+test_std, facecolor='
        #fill_between(ntrain_x, random_mean-random_std, random_mean+random_std, fa


        xlabel('Number Training Examples')
        ylabel('Feature Similarity')
        title('Dictionary, Regularized')

        xlim([0, 2000])


        plt.tight_layout()

        fname =  ('figures/trigram_dict-reg-N=' + str(N) + '-W=' + str(len(trainpr
                 + '-' + time.strftime('%y%m%d'))
        print fname

        plt.savefig(fname + '.png', transparent=True)
        plt.savefig(fname + '.eps', transparent=True)
```

figures/trigram_dict-reg-N=30000-W=1758-170307



** Figure 2. Wickelfeature dictionary with regularization. ** We evaluate the network's performance by comparing the output to the correct answer and measuring the similarity in the feature space. We see that words in the training set start with high similarity and remain high (blue), indicating one-shot learning. Words in the test set increase as more words are added in the training set and common transformations are found and applied.

With this regularization rule, we now can see a network that describes the training set words accurately and can perform one-shot learning and still there is generalization to the test set.

This regularized form of the algorithm can also iterate through the training data more than once. If we show it all of the training words a second time, we can improve its performance.

```
In [60]: tst = time.time()

         #psi = np.zeros(N)
         cycle+=1

         random_vecs = np.random.randn(testpast2.shape[0], N)

         train_hist_c = np.zeros((trainpres1.shape[0], trainpres1.shape[0]))
         test_hist_c = np.zeros((trainpres1.shape[0], testpast2.shape[0]))


         for k in range(len(trainpres1)):#trainpres1.shape[0]):
             psi = reg_train(psi, trainpast2[k], trainpres1[k])

             train_pred = np.multiply(psi, trainpres1)
             #train_pred = np.where(train_pred>0, 1, -1)
             test_pred = np.multiply(psi, testpres1)
```

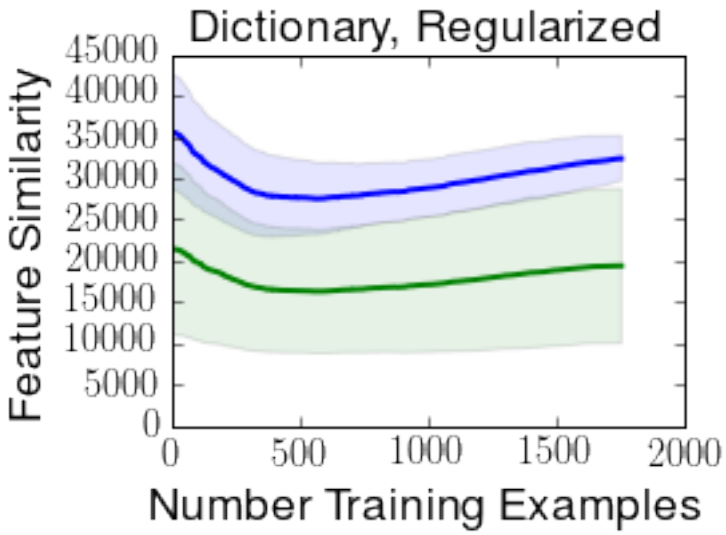
```
            #test_pred = np.where(test_pred>0, 1, -1)

            train_hist_c[k, :] = sim(train_pred, trainpast2)
            test_hist_c[k, :] = sim(test_pred, testpast2)



        print 'Elapsed: ', time.time() - tst

        fname = ('data/trigram_dict-reg-cycle=' + str(cycle) + '-N=' + str(N) + '-
                + '-' + time.strftime('%y%m%d') +'.npz')

        print fname
        np.savez(fname,N=N, train_hist_c=train_hist_c, test_hist_c=test_hist_c)
```

```
Elapsed:  998.880920172
data/trigram_dict-reg-cycle=2-N=30000-W=1758-170307.npz
```

```
In [61]: figure(figsize=(4,3))

        train_mean = np.mean(train_hist_c, axis=1)
        test_mean = test_hist_c.mean(axis=1)

        train_std = np.std(train_hist_c, axis=1) #/ (arange(len(train_mean))+1) **
        test_std = test_hist_c.std(axis=1) #/ testpast2.shape[0]**0.5

        ntrain_x = arange(len(train_mean))+1

        plot([0, 2000], [0,0], 'k')

        #plot(ntrain_x, train_hist, lw=0.2, label='Train')

        plot(ntrain_x, train_mean, c='b', lw=2, label='Train')
        plot(ntrain_x, test_mean, c='g', lw=2, label='Test')
        #plot(ntrain_x, random_mean, c='r', lw=2, label='Random')

        fill_between(ntrain_x, train_mean-train_std, train_mean+train_std, facecol
        fill_between(ntrain_x, test_mean-test_std, test_mean+test_std, facecolor='
        #fill_between(ntrain_x, random_mean-random_std, random_mean+random_std, fa

        xlabel('Number Training Examples')
        ylabel('Feature Similarity')
        title('Dictionary, Regularized')

        xlim([0, 2000])

        plt.tight_layout()
```

```
        fname =  ('figures/trigram_dict-reg-cycle=' + str(cycle) +'-N=' + str(N) +
                  + '-' + time.strftime('%y%m%d'))
        print fname

        plt.savefig(fname + '.png', transparent=True)
        plt.savefig(fname + '.eps', transparent=True)
```

```
figures/trigram_dict-reg-cycle=2-N=30000-W=1758-170307
```



** Figure 3. Regularized dictionary cycle 2. ** We see that another cycle of the learning algorith through the training set begins to decrease the similarity, but this is because it was overemphasized at the end of the first cycle. The error of the training set decreases after the second cycle. The test set performance slightly drops through the second cycle, but stays around the same level. The test set error is slightly lower, but it is hard to tell if this improved generalization performance.

Further cycles through the training data can solidify the verb pairs in the training set in the dictionary. This could also be helpful for generalization, but it is hard to tell if performance in the test set improved with the second cycle.

1.3 Trigram difference dictionary, unregularized

The algorithm we have described so far is a mapping of Wickelfeatures from the present tense verbs to the past tense verbs, however this mapping is not necessarily the best algorithm. One of the challenges with the mapping is that most present-past verb pairs have the same beginning letters. To get the correct output, we have to have the beginning of every present-tense word mapped to its same beginning for every past-tense word. This would require a lot more examples than are present in the training set.

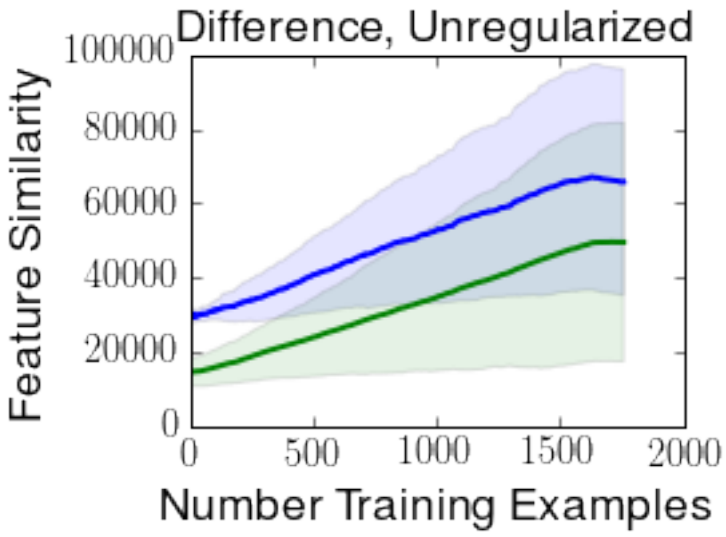
A more intuitive version of the algorithm is not to find a mapping, but rather a transformation. A transformation is described by taking the difference between the past-tense verb and the present-tense verb. Rather than storing a dictionary between all present-tense Wickelfeatures to all past-tense Wickelfeatures, we store a dictionary of all present-tense Wickelfeatures to all present-to-past transforms. We compute the transform simply as the difference between the past-tense word and the present-tense word in the feature space. Then to transform a present tense word to a past tense word, we look up the transformation from the dictionary and add it to the present tense word.

We first start by again making a naive dictionary that stores the transforms for each word in the training set with no regularization.

```
In [62]: tst = time.time()

         psi = np.zeros(N)
         psi = train_diff(psi, trainpast2[0], trainpres1[0], trainpres2[0])

         # shouldn't this be {+1, -1}?
         random_vecs = np.random.randn(testpast2.shape[0], N)

         train_hist = np.nan* np.zeros((trainpres1.shape[0], trainpres1.shape[0]))
         test_hist = np.zeros((trainpres1.shape[0], testpast2.shape[0]))
         random_hist = np.zeros((trainpres1.shape[0], testpast2.shape[0]))


         for k in range(1,len(trainpres1)):#trainpres1.shape[0]):
             train_pred = np.multiply(psi, trainpres1[:k]) + trainpres2[:k]
             #train_pred = np.where(train_pred>0, 1, -1)
             test_pred = np.multiply(psi, testpres1) + testpres2
             #test_pred = np.where(test_pred>0, 1, -1)

             train_hist[k, :k] = sim(train_pred, trainpast2[:k])
             test_hist[k, :] = sim(test_pred, testpast2)
             random_hist[k, :] = sim(test_pred, random_vecs)

             psi = train_diff(psi, trainpast2[k], trainpres1[k], trainpres2[k])


         print 'Elapsed: ', time.time() - tst

         fname = ('data/trigram_difference_dict-unreg-N=' + str(N) + '-W=' + str(le
                 + '-' + time.strftime('%y%m%d') +'.npz')

         print fname
         np.savez(fname,N=N, train_hist=train_hist, test_hist=test_hist, random_his

Elapsed:  1174.23419404
data/trigram_difference_dict-unreg-N=30000-W=1758-170307.npz
```

In [63]:
```
figure(figsize=(4,3))

train_mean = np.nanmean(train_hist, axis=1)
test_mean = test_hist.mean(axis=1)
random_mean = random_hist.mean(axis=1)

train_std = np.nanstd(train_hist, axis=1) #/ (arange(len(train_mean))+1) *
test_std = test_hist.std(axis=1) #/ testpast2.shape[0]**0.5
random_std = random_hist.std(axis=1) #/ testpast2.shape[0]**0.5

ntrain_x = arange(len(train_mean))+1
plot([0, 2000], [0,0], 'k')

plot(ntrain_x, train_mean, c='b', lw=2, label='Train')
plot(ntrain_x, test_mean, c='g', lw=2, label='Test')
#plot(ntrain_x, random_mean, c='r', lw=2, label='Random')

fill_between(ntrain_x, train_mean-train_std, train_mean+train_std, facecol
fill_between(ntrain_x, test_mean-test_std, test_mean+test_std, facecolor='
#fill_between(ntrain_x, random_mean-random_std, random_mean+random_std, fa

xlabel('Number Training Examples')
ylabel('Feature Similarity')
title('Difference, Unregularized')

xlim([0, 2000])

plt.tight_layout()

fname =  ('figures/trigram_diff_dict-unreg-N=' + str(N) + '-W=' + str(len
          + '-' + time.strftime('%y%m%d'))
print fname

plt.savefig(fname + '.png', transparent=True)
plt.savefig(fname + '.eps', transparent=True)
```

```
figures/trigram_diff_dict-unreg-N=30000-W=1758-170307
```



** Figure 4. Wickelfeature difference dictionary. ** The algorithm that stores the transform immediately has some similarity with words in the test set, because the present-tense word is usually similar to the past-tense word without any transformation. We again see the over-emphasis of certain transformation features, leading to similarities going above N.

We see that the difference algorithm immediately has similarity for verb-pairs in the test set, and this occurs because this algorithm by default repeats the present-tense verb as the output. Since the present-tense verb is often similar to the past-tense verb, the network outputs similar verbs without any training.

1.4 Trigram difference dictionary, regularized

Finally, we use the difference algorithm with the regularization procedure as the algorithm.

```
In [64]: tst = time.time()
         cycle = 1
         psi = np.zeros(N)
         psi = reg_train_diff(psi, trainpast2[0], trainpres1[0], trainpres2[0])

         random_vecs = np.random.randn(testpast2.shape[0], N)

         train_hist = np.nan* np.zeros((trainpres1.shape[0], trainpres1.shape[0]))
         test_hist = np.zeros((trainpres1.shape[0], testpast2.shape[0]))
         random_hist = np.zeros((trainpres1.shape[0], testpast2.shape[0]))


         for k in range(1,len(trainpres1)):#trainpres1.shape[0]):
             train_pred = np.multiply(psi, trainpres1[:k]) + trainpres2[:k]
             #train_pred = np.where(train_pred>0, 1, -1)
             test_pred = np.multiply(psi, testpres1) + testpres2
             #test_pred = np.where(test_pred>0, 1, -1)
```

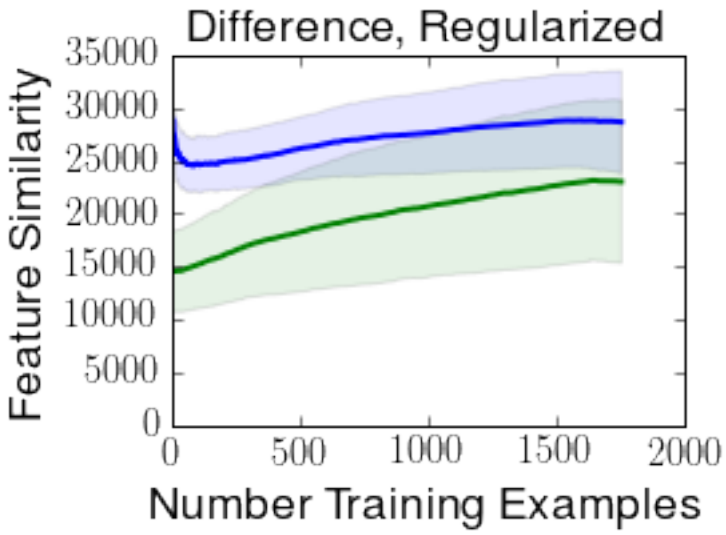
```
            train_hist[k, :k] = sim(train_pred, trainpast2[:k])
            test_hist[k, :] = sim(test_pred, testpast2)
            random_hist[k, :] = sim(random_vecs, testpast2)

            psi = reg_train_diff(psi, trainpast2[k], trainpres1[k], trainpres2[k])


        print 'Elapsed: ', time.time() - tst

        fname = ('data/trigram_difference_dict-reg-N=' + str(N) + '-W=' + str(len
                + '-' + time.strftime('%y%m%d') +'.npz')

        print fname
        np.savez(fname,N=N, train_hist=train_hist, test_hist=test_hist, random_his
```

```
Elapsed:  1176.95332694
data/trigram_difference_dict-reg-N=30000-W=1758-170307.npz
```

In [65]:
```
figure(figsize=(4,3))

train_mean = np.nanmean(train_hist, axis=1)
test_mean = test_hist.mean(axis=1)

train_std = np.nanstd(train_hist, axis=1) #/ (arange(len(train_mean))+1) *
test_std = test_hist.std(axis=1) #/ testpast2.shape[0]**0.5

ntrain_x = arange(len(train_mean))+1

plot([0, 2000], [0,0], 'k')

#plot(ntrain_x, train_hist, lw=0.2, label='Train')

plot(ntrain_x, train_mean, c='b', lw=2, label='Train')
plot(ntrain_x, test_mean, c='g', lw=2, label='Test')
#plot(ntrain_x, random_mean, c='r', lw=2, label='Random')

fill_between(ntrain_x, train_mean-train_std, train_mean+train_std, facecol
fill_between(ntrain_x, test_mean-test_std, test_mean+test_std, facecolor='
#fill_between(ntrain_x, random_mean-random_std, random_mean+random_std, fa

xlabel('Number Training Examples')
ylabel('Feature Similarity')
title('Difference, Regularized')

xlim([0, 2000])
```

```
plt.tight_layout()



fname =  ('figures/trigram_diff_dict-reg-N=' + str(N) + '-W=' + str(len(tr
          + '-' + time.strftime('%y%m%d'))
print fname

plt.savefig(fname + '.png', transparent=True)
plt.savefig(fname + '.eps', transparent=True)
```

```
figures/trigram_diff_dict-reg-N=30000-W=1758-170307
```

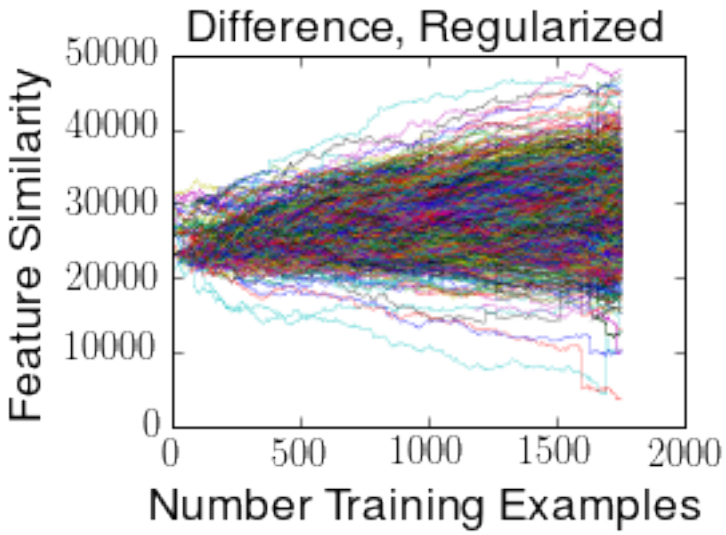


** Figure 5. Wickelfeature difference dictionary with regularization. ** We use the difference algorithm with regularization and see that words in the training set have high similarity throughout. Words in the test set start off with some similarity and increase.

```
In [66]: figure(figsize=(4,3))

         ntrain_x = arange(len(train_mean))+1

         plot([0, 2000], [0,0], 'k')

         plot(ntrain_x, train_hist, lw=0.2, label='Train')

         xlabel('Number Training Examples')
         ylabel('Feature Similarity')
         title('Difference, Regularized')
```

```
xlim([0, 2000])

plt.tight_layout()



fname =  ('figures/trigram_diff_dict-reg-N=' + str(N) + '-W=' + str(len(tr
          + '-' + time.strftime('%y%m%d'))
print fname

#plt.savefig(fname + '.png', transparent=True)
#plt.savefig(fname + '.eps', transparent=True)
```

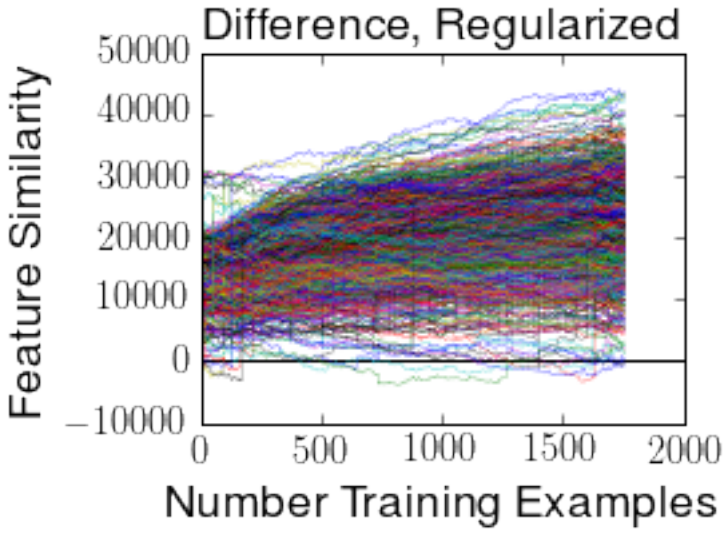
```
figures/trigram_diff_dict-reg-N=30000-W=1758-170307
```



In [67]:
```
figure(figsize=(4,3))

ntrain_x = arange(len(train_mean))+1

plot([0, 2000], [0,0], 'k')

plot(ntrain_x, test_hist, lw=0.2, label='Train')

xlabel('Number Training Examples')
ylabel('Feature Similarity')
title('Difference, Regularized')

xlim([0, 2000])
```

```
        plt.tight_layout()



        fname =  ('figures/trigram_diff_dict-reg-N=' + str(N) + '-W=' + str(len(tr
                  + '-' + time.strftime('%y%m%d'))
        print fname

        #plt.savefig(fname + '.png', transparent=True)
        #plt.savefig(fname + '.eps', transparent=True)
```

```
figures/trigram_diff_dict-reg-N=30000-W=1758-170307
```



In [68]:
```
tst = time.time()

#psi = np.zeros(N)
cycle+=1

random_vecs = np.random.randn(testpast2.shape[0], N)

train_hist_c = np.zeros((trainpres1.shape[0], trainpres1.shape[0]))
test_hist_c = np.zeros((trainpres1.shape[0], testpast2.shape[0]))


for k in range(len(trainpres1)):#trainpres1.shape[0]):
    psi = reg_train_diff(psi, trainpast2[k], trainpres1[k], trainpres2[k])
```

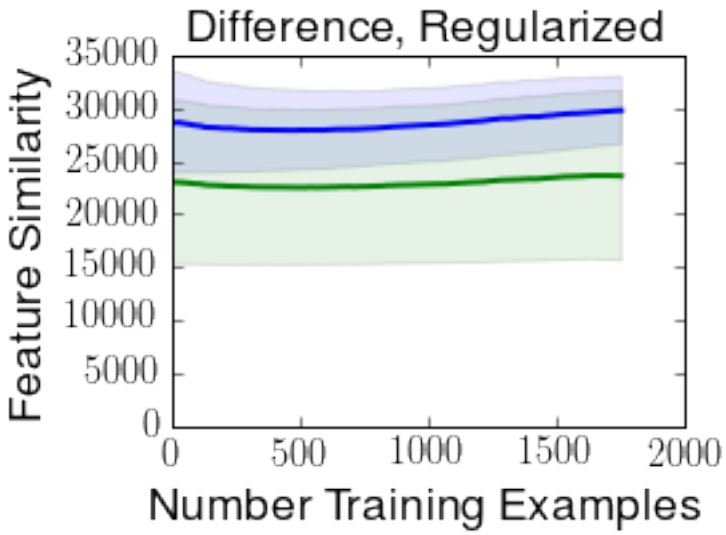
```
            train_pred = np.multiply(psi, trainpres1) + trainpres2
            #train_pred = np.where(train_pred>0, 1, -1)
            test_pred = np.multiply(psi, testpres1) + testpres2
            #test_pred = np.where(test_pred>0, 1, -1)

            train_hist_c[k, :] = sim(train_pred, trainpast2)
            test_hist_c[k, :] = sim(test_pred, testpast2)



        print 'Elapsed: ', time.time() - tst

        fname = ('data/trigram_difference_dict-reg-cycle=' + str(cycle) + '-N=' +
                + '-' + time.strftime('%y%m%d') +'.npz')

        print fname
        np.savez(fname,N=N, train_hist_c=train_hist_c, test_hist_c=test_hist_c)
```

```
Elapsed:  1476.75472689
data/trigram_difference_dict-reg-cycle=2-N=30000-W=1758-170307.npz
```

```
In [69]: figure(figsize=(4,3))

        train_mean = np.mean(train_hist_c, axis=1)
        test_mean = test_hist_c.mean(axis=1)

        train_std = np.std(train_hist_c, axis=1) #/ (arange(len(train_mean))+1) **
        test_std = test_hist_c.std(axis=1) #/ testpast2.shape[0]**0.5

        ntrain_x = arange(len(train_mean))+1

        plot([0, 2000], [0,0], 'k')

        #plot(ntrain_x, train_hist, lw=0.2, label='Train')

        plot(ntrain_x, train_mean, c='b', lw=2, label='Train')
        plot(ntrain_x, test_mean, c='g', lw=2, label='Test')
        #plot(ntrain_x, random_mean, c='r', lw=2, label='Random')

        fill_between(ntrain_x, train_mean-train_std, train_mean+train_std, facecol
        fill_between(ntrain_x, test_mean-test_std, test_mean+test_std, facecolor='
        #fill_between(ntrain_x, random_mean-random_std, random_mean+random_std, fa

        xlabel('Number Training Examples')
        ylabel('Feature Similarity')
        title('Difference, Regularized')
```

```
        xlim([0, 2000])

        plt.tight_layout()



        fname =  ('figures/trigram_diff_dict-reg-cycle=' + str(cycle) +'-N=' + str
                 + '-' + time.strftime('%y%m%d'))
        print fname

        plt.savefig(fname + '.png', transparent=True)
        plt.savefig(fname + '.eps', transparent=True)
```

```
figures/trigram_diff_dict-reg-cycle=2-N=30000-W=1758-170307
```



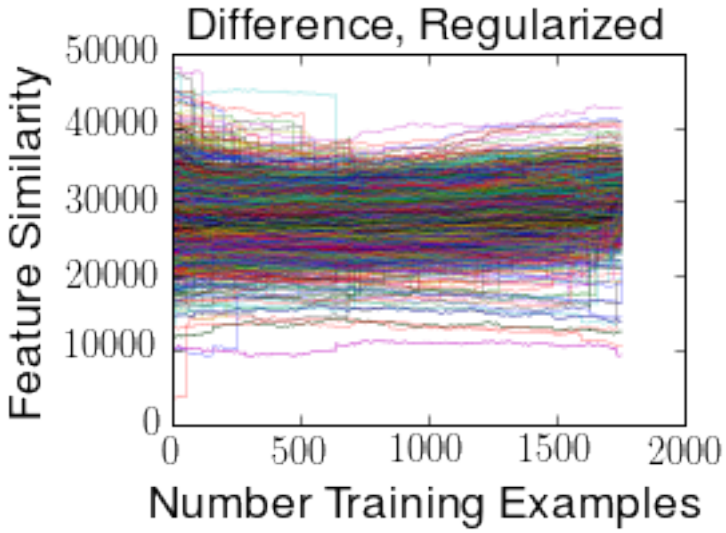
** Figure 6. Second cycle of Wickelfeature difference dictionary with regularization. ** We again see some improvement in the variance of the training set, but it is hard to tell if there is improvement with test set generalization.

```
In [70]: figure(figsize=(4,3))

         ntrain_x = arange(len(train_mean))+1

         plot([0, 2000], [0,0], 'k')

         plot(ntrain_x, train_hist_c, lw=0.2, label='Train')

         xlabel('Number Training Examples')
         ylabel('Feature Similarity')
```

```
title('Difference, Regularized')

xlim([0, 2000])

plt.tight_layout()
```

```
fname =  ('figures/trigram_diff_dict-reg-N=' + str(N) + '-W=' + str(len(tr
          + '-' + time.strftime('%y%m%d'))
print fname

#plt.savefig(fname + '.png', transparent=True)
#plt.savefig(fname + '.eps', transparent=True)
```

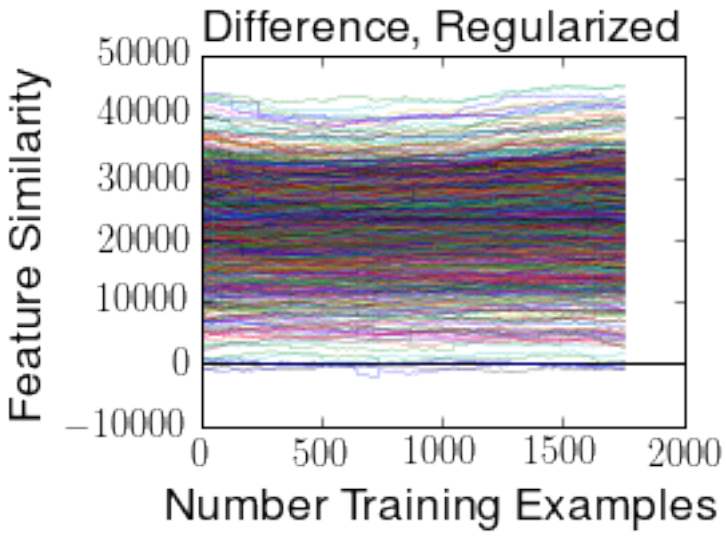
```
figures/trigram_diff_dict-reg-N=30000-W=1758-170307
```



In [71]:
```
figure(figsize=(4,3))

ntrain_x = arange(len(train_mean))+1

plot([0, 2000], [0,0], 'k')

plot(ntrain_x, test_hist_c, lw=0.2, label='Train')

xlabel('Number Training Examples')
ylabel('Feature Similarity')
title('Difference, Regularized')
```

```
        xlim([0, 2000])

        plt.tight_layout()



        fname =  ('figures/trigram_diff_dict-reg-N=' + str(N) + '-W=' + str(len(tr
                  + '-' + time.strftime('%y%m%d'))
        print fname

        #plt.savefig(fname + '.png', transparent=True)
        #plt.savefig(fname + '.eps', transparent=True)
```

```
figures/trigram_diff_dict-reg-N=30000-W=1758-170307
```




```
In [ ]:
```