# npfl067 assignment 1

Jaromir Salamon

Tuesday, March 03, 2015

## 1 Source data

Source data are available here:

- TEXTEN1.TXT
- TEXTCZ1.TXT

## 2 Tasks to do

Complete description here: Assignment #1: PFL067 Statistical NLP

### 2.1 Entropy of a Text

In this experiment, you will determine the conditional entropy of the word distribution in a text given the previous word. Compute this conditional entropy and perplexity for source data.

10 times mess up the text and measure how this alters the conditional entropy. First for every character in the text, mess it up with a likelihood of 10%, 5%, 1%, .1%, .01%, and .001%. Then every word in the text, mess it up with the same likelihood as in previous case.

### 2.2 Cross-Entropy and Language Modeling

This task will show you the importance of smoothing for language modeling, and in certain detail it lets you feel its effects.

Prepare 3 datasets out of each:

1. strip off the last 20,000 words and call them the Test Data,
2. then take off the last 40,000 words from what remains, and call them the Held out Data,
3. and call the remaining data the Training Data.

Extract word counts from the training data so that you are ready to compute unigram-, bigram- and trigram-based probabilities from them; compute also the uniform probability based on the vocabulary size.

Compute the four smoothing parameters "interpolation parameters" or whatever, for the trigram, bigram, unigram and uniform distributions) from the held out data using the EM algorithm. Then do the same using the training data again.

And finally, compute the cross-entropy of the test data using your newly built, smoothed language model. Now tweak the smoothing parameters in the following way: add 10%, 20%, 30%, ..., 90%, 95% and 99% of the difference between the trigram smoothing parameter and 1.0 to its value, discounting at the same the remaining three parameters proportionally (remember, they have to sum up to 1.0!!). Then set the trigram smoothing parameter to 90%, 80%, 70%, ... 10%, 0% of its value, boosting proportionally the other three parameters, again to sum up to one.

# 3 Common

## 3.1 N-Gram method

For both tasks I need to calculate n-grams, for Conditional Entropy I need uni- and bi-grams, for Cross Entropy and Language Modeling I need uni-, bi- and tri-grams. So, following method allows preparing dictionaries for such structures where value is count of n-grams:

```
def nGram(d, N = 1):
    nGram = {}
    nGram.clear()
    key = ""
    prevWord1 = "<S>"
    prevWord2 = "<S>"

    data = ['<S>','<S>']
    data += d
    for word in data:
        if N == 1:
            key = word #unigram key
        elif N == 2:
            key = word + "|" + prevWord1 #bigram key
        elif N == 3:
            key = word + "|" + prevWord2 + " " + prevWord1 #trigram key

        if nGram.has_key(key): # calculate counts of n-gram
            nGram[key] = nGram[key] + 1
        else:
            nGram[key] = 1
        prevWord2 = prevWord1
        prevWord1 = word
    if N == 1:
        nGram['<S>'] = 1 #fix count of start key for unigram
    elif N == 2:
        nGram['<S>|<S>'] = 1 #fix count of start key for bigram

    return nGram;
```

# 4 Entropy of a Text

## 4.1 Conditional Entropy method

From a unigram and a bigram number of words appearing in text is calculated a joined probability and a conditional probability and then according to the equation a conditional probability:

```
def condEntropy(uniGram, biGram):
    H = 0
    total = sum(uniGram.values())

    for key in biGram:

        pJoined = biGram[key] / (1.0 * total) #joined prob
        pCond = biGram[key] / (1.0 * uniGram[getKeys(key)[1]]) #conditional prob

        H -= pJoined * math.log(pCond,2)

    return H
```

## 4.2 Mess up words and characters methods

Both methods are similar. In the both cases are calculated samples of the counts of words or characters which needs to be messed up according to likelihood and then words are picked until the full sample count isn't satisfied. In case of words each sampled is replaced by randomly chosen from the unique list of words. In case of the character replacement, in each sampled word character is randomly replaced from the unique character set taken from corpus source.

```
def replaceWords(d, p):
    random.seed(1)

    data = d
    words = list(set(data)) #make words from list unique
    sample = int(round(len(data) * (p / 100.0))) #how many replace based on
probability

    while sample:
        sample = sample - 1
        i = random.randint(1, len(data)-1)
        j = random.randint(1, len(words)-1)
        data[i] = words[j]

    return data
```

```
def replaceChars(d, p):
    random.seed(1)

    data = d
```

```
    ch = [] #unige characters of all words
    length = 0 #lenght of characters of all words
    for word in data:
        ch += list(set(word))
        length += len(word)

    ch = list(set(ch))
    sample = int(round(length * (p / 100.0)))
    w = []

    while sample:
        sample = sample - 1
        i = random.randint(1, len(data)-1)
        chars = list(data[i])
        w[:] = []
        for char in chars:
            j = random.randint(1, len(ch)-1) #random word index
            char = ch[j]
            w.append(char)
        data[i] = ''.join(w)

    return data
```

## 4.3 Final calculation of Conditional Entropy

Following snippet of code reveals how this is done, 10 times calculated messed up words (could be another method for characters), then calculated uni- and bi- grams from which conditional entropy is calculated). Finally min, average and max values of entropy are printed together with perplexity from average entropy value:

```
print 'probability\tmin(H)\tavg(H)\tmax(H)\tperplexity'
for px in p:
    H[:] = []
    for i in range(10):
        lines = npfl067.replaceWords(lines_ref,px)
        uni_gram = npfl067.nGram(lines,1)
        bi_gram = npfl067.nGram(lines,2)

        H.append(npfl067.condEntropy(uni_gram, bi_gram))


    print px,'\t','{0:.4f}'.format(min(H)),'\t','{0:.4f}'.format(sum(H) /
len(H)),'\t','{0:.4f}'.format(max(H)),'\t','{0:.4f}'.format(2**(sum(H) / len(H)))
```

## 4.4 Results

### 4.4.1 Conditional Entropy and Perplexity of EN text

Replace words

| probability | min(H) | avg(H) | max(H) | perplexity |
|---|---|---|---|---|
| **0** | 5.2874 | 5.2874 | 5.2874 | 39.0546 |
| **0.001** | 5.2874 | 5.2874 | 5.2874 | 39.0552 |
| **0.01** | 5.2877 | 5.2877 | 5.2877 | 39.0616 |
| **0.1** | 5.2887 | 5.2887 | 5.2888 | 39.0901 |
| **1** | 5.3064 | 5.3067 | 5.3074 | 39.581 |
| **5** | 5.3795 | 5.3801 | 5.3808 | 41.6458 |
| **10** | 5.4572 | 5.4593 | 5.4608 | 43.9951 |
| **20** | 5.564 | 5.5661 | 5.5679 | 47.3762 |

Replace characters

| probability | min(H) | avg(H) | max(H) | perplexity |
|---|---|---|---|---|
| **0** | 5.2874 | 5.2874 | 5.2874 | 39.0546 |
| **0.001** | 5.2873 | 5.2873 | 5.2873 | 39.0514 |
| **0.01** | 5.287 | 5.287 | 5.287 | 39.0433 |
| **0.1** | 5.2846 | 5.2846 | 5.2846 | 38.9776 |
| **1** | 5.2561 | 5.2561 | 5.2561 | 38.2162 |
| **5** | 5.0141 | 5.0141 | 5.0141 | 32.3132 |
| **10** | 4.5651 | 4.5651 | 4.5651 | 23.6719 |
| **20** | 3.5869 | 3.5869 | 3.5869 | 12.0165 |

### 4.4.2 Conditional Entropy and Perplexity of CZ text

Replace words

| probability | min(H) | avg(H) | max(H) | perplexity |
|---|---|---|---|---|
| **0** | 4.7478 | 4.7478 | 4.7478 | 26.8683 |
| **0.001** | 4.7478 | 4.7478 | 4.7478 | 26.8677 |
| **0.01** | 4.7475 | 4.7475 | 4.7475 | 26.8618 |
| **0.1** | 4.7469 | 4.747 | 4.7471 | 26.8526 |
| **1** | 4.739 | 4.7393 | 4.7396 | 26.7106 |
| **5** | 4.7013 | 4.7044 | 4.7055 | 26.0719 |
| **10** | 4.655 | 4.6602 | 4.6618 | 25.2851 |
| **20** | 4.5413 | 4.5603 | 4.5667 | 23.5933 |

Replace characters

| probability | min(H) | avg(H) | max(H) | perplexity |
|---|---|---|---|---|
| **0** | 4.7478 | 4.7478 | 4.7478 | 26.8683 |
| **0.001** | 4.7476 | 4.7476 | 4.7476 | 26.8647 |
| **0.01** | 4.747 | 4.747 | 4.747 | 26.8521 |
| **0.1** | 4.7389 | 4.7389 | 4.7389 | 26.7028 |
| **1** | 4.6649 | 4.6649 | 4.6649 | 25.3676 |
| **5** | 4.347 | 4.347 | 4.347 | 20.3505 |
| **10** | 3.9677 | 3.9677 | 3.9677 | 15.6454 |
| **20** | 3.3544 | 3.3544 | 3.3544 | 10.2274 |

### 4.4.3 Conditional Entropy and Perplexity of EN + CZ text

Replace words

| probability | min(H) | avg(H) | max(H) | perplexity |
|---|---|---|---|---|
| **0** | 5.1658 | 5.1658 | 5.1658 | 35.898 |
| **0.001** | 5.1658 | 5.1658 | 5.1658 | 35.8978 |
| **0.01** | 5.1659 | 5.1659 | 5.1659 | 35.899 |
| **0.1** | 5.1658 | 5.1658 | 5.1659 | 35.8982 |
| **1** | 5.1677 | 5.1681 | 5.1685 | 35.9539 |
| **5** | 5.1727 | 5.1749 | 5.1759 | 36.1232 |
| **10** | 5.1671 | 5.1724 | 5.1744 | 36.0616 |
| **20** | 5.1265 | 5.1376 | 5.145 | 35.2018 |

Replace characters

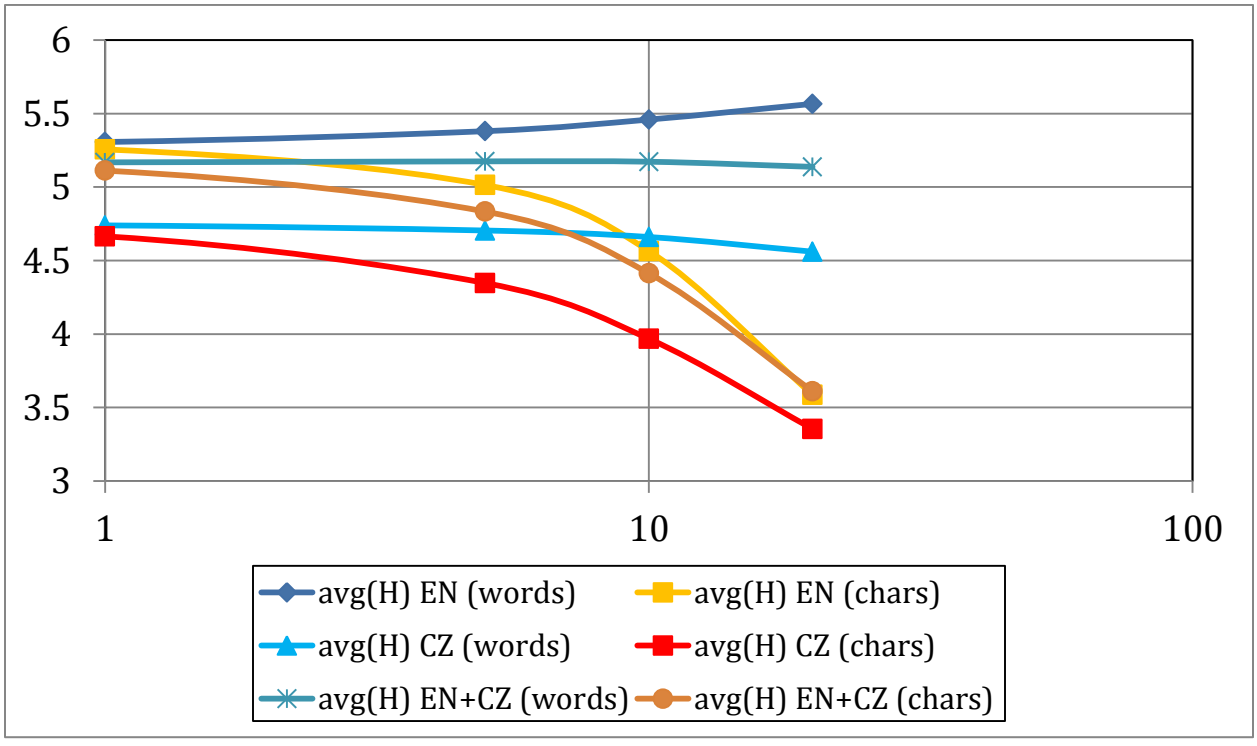
| probability | min(H) | avg(H) | max(H) | perplexity |
|---|---|---|---|---|
| **0** | 5.1658 | 5.1658 | 5.1658 | 35.898 |
| **0.001** | 5.1659 | 5.1659 | 5.1659 | 35.899 |
| **0.01** | 5.165 | 5.165 | 5.165 | 35.878 |
| **0.1** | 5.1594 | 5.1594 | 5.1594 | 35.7382 |
| **1** | 5.1118 | 5.1118 | 5.1118 | 34.5789 |
| **5** | 4.8333 | 4.8333 | 4.8333 | 28.508 |
| **10** | 4.413 | 4.413 | 4.413 | 21.3037 |
| **20** | 3.6109 | 3.6109 | 3.6109 | 12.2179 |

## 4.5 Charts and conclusions

Comparing in graph average Conditional Entropy for those 6 different cases ([EN, CZ, EN + CZ] * mess up [words, characters]) shows following:

- When mess up with words (blue lines) for specific group [EN, CZ, EN + CZ] entropy is higher and in log scale almost linear with small increase or decrease compare to entropy of test with mess up characters (orange/red lines). So, the uncertainty is increasing when I am messing with words not with characters.
- Conditional entropy up to 1% of messing with words or chars is similar no matter if EN or CZ text. Then Conditional Entropy is dropping in CZ text radically, so messing up words and characters leads to lower uncertainty of prediction bigrams in CZ text.
- According to the following statistics:
    - EN unigram counts: 9608
    - EN bigram counts: 73248
    - CZ unigram counts: 42827
    - CZ bigram counts: 147139
    - EN + CZ unigram counts: 52127
    - EN + CZ bigram counts: 220195

  With increasing of unique unigrams (words) and bigrams (word + history) is decreasing its entropy. So, the uncertainty is decreasing with increasing of number of processed words.
- And for combination of text EN + CZ is entropy between original languages EN and CZ no matter if I am messing up with words or counts (except 20% mess up characters). So, I can hardly say that text EN + CZ has little bit more than average entropy of text EN and text CZ.



# 5 Cross-Entropy and Language Modeling

## 5.1 Cross-Entropy method

According to definition is counted Cross Entropy as sum of logarithms with base 2 of interpolated probabilities over all word in corpus:

```
def crossEntropy(dataTest, dataTrain, l):
    sumProb = 0

    data = dataTest
    uniGram = nGram(dataTrain, 1)
    biGram = nGram(dataTrain, 2)
    triGram = nGram(dataTrain, 3)
    nGrams = [uniGram, biGram, triGram]

    prevWord1 = "<S>"
    prevWord2 = "<S>"
    for word in data:
        keys = [word,prevWord1,prevWord2]
        iProb = probInt(keys, nGrams, l)

        prevWord2 = prevWord1
        prevWord1 = word

        sumProb += math.log(iProb,2)

    H = -1.0 * sumProb / len(data)
    return H
```

## 5.2 Interpolated (smoothed) probability method

Interpolated probability is then calculated as sum of uniform probability and unigram-, bigram- and trigram-based probabilities by following method:

```
def probInt(keys, nGrams, l):
    iProb = l[0] * probAll(keys, nGrams, 0) + \
            l[1] * probAll(keys, nGrams, 1) + \
            l[2] * probAll(keys, nGrams, 2) + \
            l[3] * probAll(keys, nGrams, 3)
    return iProb
```

## 5.3 Overall (n-gram) probability method

Separated calculation of uniform and specific n-gram probabilities is done in this method:

```
def probAll(keys, nGrams, n):
    p = 0
    # nGrams = [uniGram, biGram, triGram]
    uniGram = nGrams[0]
    biGram = nGrams[1]
    triGram = nGrams[2]
```

```
    # keys = [word,prevWord1,prevWord2]
    word = keys[0]
    prevWord1 = keys[1]
    prevWord2 = keys[2]

    biKey = word + "/" + prevWord1
    triKey = word + "/" + prevWord2 + " " + prevWord1

    if n > 0 and prevWord1 in uniGram and (prevWord1 + "/" + prevWord2) in biGram:
        if   n == 1:
            p = 1.0 * uniGram[word]/(sum(uniGram.values())-1) if
uniGram.has_key(word) else 0
        elif n == 2:
            p = 1.0 * biGram[biKey] / uniGram[prevWord1] if biGram.has_key(biKey)
else 0
        elif n == 3:
            p = 1.0 * triGram[triKey] / biGram[prevWord1 + "/" + prevWord2] if
triGram.has_key(triKey) else 0
    else:
        p = 1.0 / (len(uniGram)-1)

    return p
```

## 5.4 Learning lambda parameters method

Method for learning lambda is calculating expected counts and follows up formula for “next lambda”:

```
def trainLambda(dataTest, dataTrain, l):
    newl = [0, 0, 0, 0]
    data = dataTest
    uniGram = nGram(dataTrain, 1)
    biGram = nGram(dataTrain, 2)
    triGram = nGram(dataTrain, 3)
    nGrams = [uniGram, biGram, triGram]

    prevWord1 = "<S>"
    prevWord2 = "<S>"
    # expected counts calculations
    for word in data:
        keys = [word,prevWord1,prevWord2]
        newl[0] += 1.0 * probAll(keys, nGrams, 0) / probInt(keys, nGrams, l)
        newl[1] += 1.0 * probAll(keys, nGrams, 1) / probInt(keys, nGrams, l)
        newl[2] += 1.0 * probAll(keys, nGrams, 2) / probInt(keys, nGrams, l)
        newl[3] += 1.0 * probAll(keys, nGrams, 3) / probInt(keys, nGrams, l)
        prevWord2 = prevWord1
        prevWord1 = word
    # next lambda calculations
    for i in range(len(newl)):
        newl[i] *= l[i]
    sumNewl = sum(newl)
    for i in range(len(newl)):
        newl[i] /= sumNewl
    return newl
```

## 5.5 Final calculation of Cross-Entropy

Process about calculation:

1. Lambdas are calculated from training data.
2. Then is calculated cross entropy from test data with usage of lambdas calculated from training data.
3. Then, because lambdas calculated from training data are converging lambda 3 to 1 and rest to 0, is in next step calculated lambda from held out data (input is lambda from step 1).
4. Again is calculated cross entropy from test data with usage of lambdas calculated from held out data.
5. Finally is done tweaking lambda 3 and calculation of cross entropy from test data.

## 5.6 Results

### 5.6.1 Cross Entropy of EN text

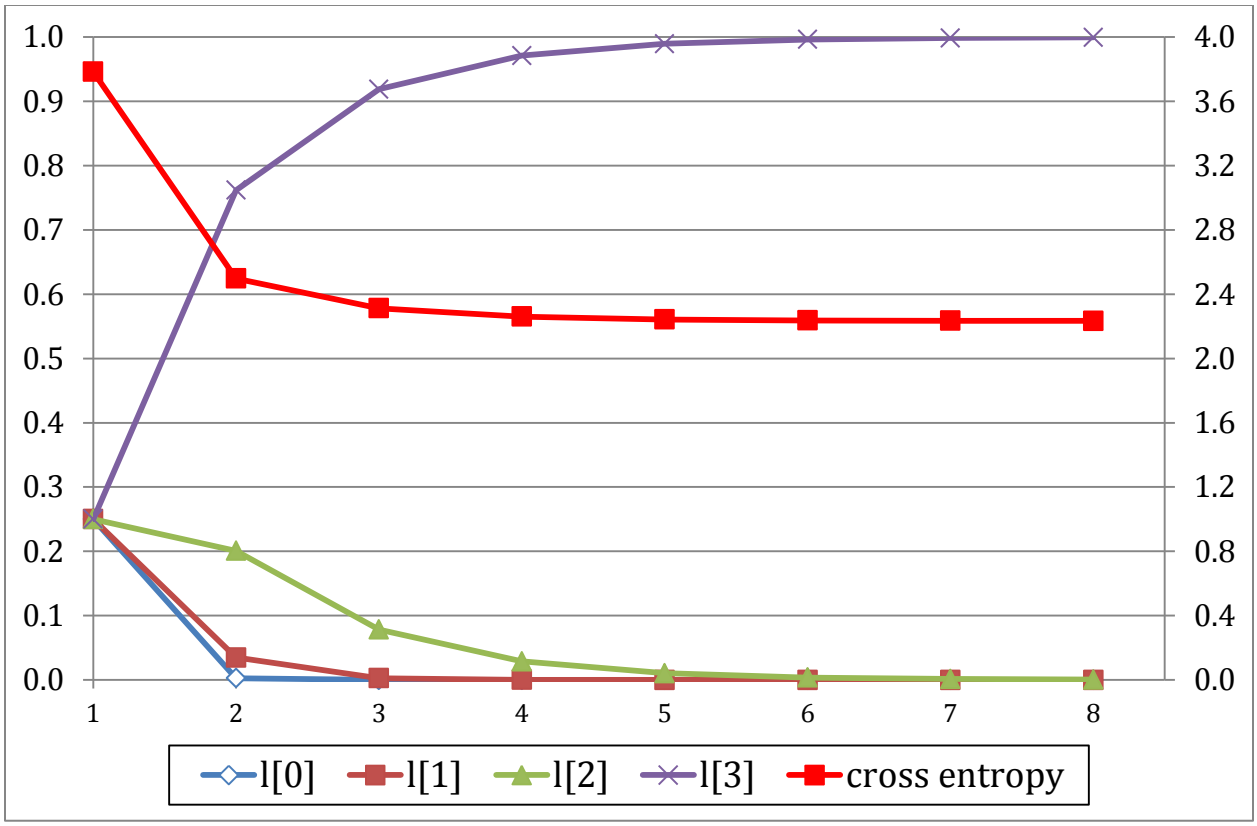
Initial values based on split data for EN text:

- training data length 161098
- held-out data length 40000
- testing data length 20000
- vocabulary size (V): 8076

#### 5.6.1.1 Training smoothing parameters

Cross entropy and learning lambda on training data, just 7 iterations were enough for commit condition that difference of all lambda between two iterations are < 0.001. Anyway it is obvious that lambda with index 3 is converging to 1

| iteration | l[0] | l[1] | l[2] | l[3] | cross entropy |
|---|---|---|---|---|---|
| **0** | 0.250000 | 0.250000 | 0.250000 | 0.250000 | 3.78379 |
| **1** | 0.002678 | 0.034811 | 0.200696 | 0.761814 | 2.49747 |
| **2** | 0.000013 | 0.002675 | 0.078392 | 0.918920 | 2.31210 |
| **3** | 0.000000 | 0.000196 | 0.028754 | 0.971050 | 2.26045 |
| **4** | 0.000000 | 0.000014 | 0.010438 | 0.989548 | 2.24302 |
| **5** | 0.000000 | 0.000001 | 0.003780 | 0.996219 | 2.23683 |
| **6** | 0.000000 | 0.000000 | 0.001368 | 0.998632 | 2.23460 |
| **7** | 0.000000 | 0.000000 | 0.000495 | 0.999505 | 2.23380 |



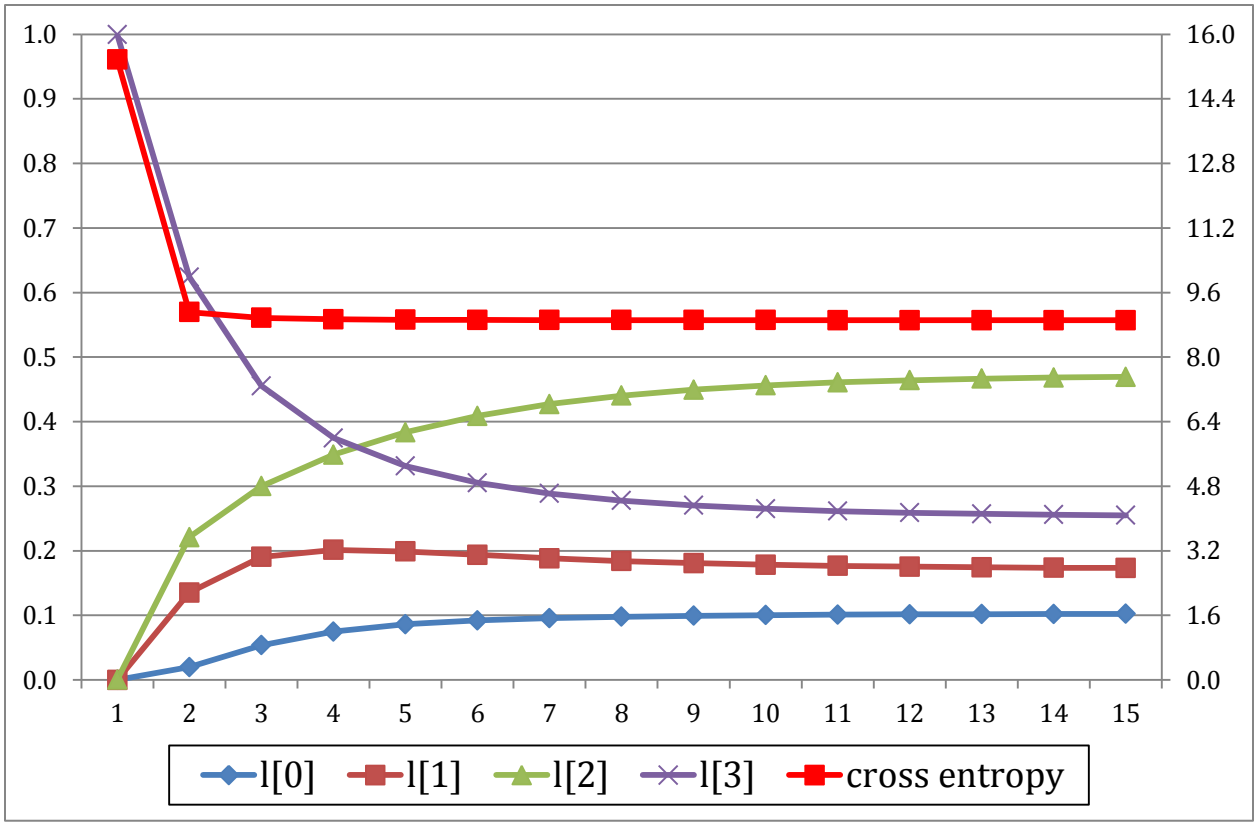

Cross entropy on test data based on trained lambda from training data:

| iteration | l[0] | l[1] | l[2] | l[3] | cross entropy |
|---|---|---|---|---|---|
| **TEST** | 0.000000 | 0.000000 | 0.000495 | 0.999505 | 15.09935 |

Cross entropy and learning lambda on held-out data, 14 iterations were enough to satisfy difference condition of all lambda < 0.001. Currently lambdas are spread more equally.

| iteration | l[0] | l[1] | l[2] | l[3] | cross entropy |
|---|---|---|---|---|---|
| **0** | 0.000000 | 0.000000 | 0.000495 | 0.999505 | 15.372563 |
| **1** | 0.019875 | 0.135301 | 0.220920 | 0.623904 | 9.114110 |
| **2** | 0.053821 | 0.190645 | 0.300047 | 0.455487 | 8.971630 |
| **3** | 0.074906 | 0.201467 | 0.348838 | 0.374789 | 8.937077 |
| **4** | 0.086158 | 0.199075 | 0.383572 | 0.331195 | 8.924975 |
| **5** | 0.092230 | 0.193646 | 0.408867 | 0.305258 | 8.919793 |
| **6** | 0.095722 | 0.188362 | 0.427161 | 0.288755 | 8.917360 |
| **7** | 0.097888 | 0.184051 | 0.440268 | 0.277792 | 8.916171 |
| **8** | 0.099320 | 0.180768 | 0.449598 | 0.270313 | 8.915581 |
| **9** | 0.100307 | 0.178347 | 0.456218 | 0.265128 | 8.915286 |
| **10** | 0.101006 | 0.176589 | 0.460908 | 0.261497 | 8.915138 |
| **11** | 0.101505 | 0.175323 | 0.464231 | 0.258940 | 8.915063 |
| **12** | 0.101865 | 0.174416 | 0.466587 | 0.257132 | 8.915026 |

| | | | | | |
|---|---|---|---|---|---|
| **13** | 0.102124 | 0.173767 | 0.468258 | 0.255851 | 8.915007 |
| **14** | 0.102310 | 0.173304 | 0.469444 | 0.254942 | 8.914997 |




Cross entropy on test data based on trained lambda:

| iteration | l[0] | l[1] | l[2] | l[3] | cross entropy |
|---|---|---|---|---|---|
| **TEST** | 0.102310 | 0.173304 | 0.469444 | 0.254942 | 8.934893 |

#### 5.6.1.2 Tweaking smoothing parameters

Tweaking lambda l[3] and calculation of cross entropy is then in next few tables.

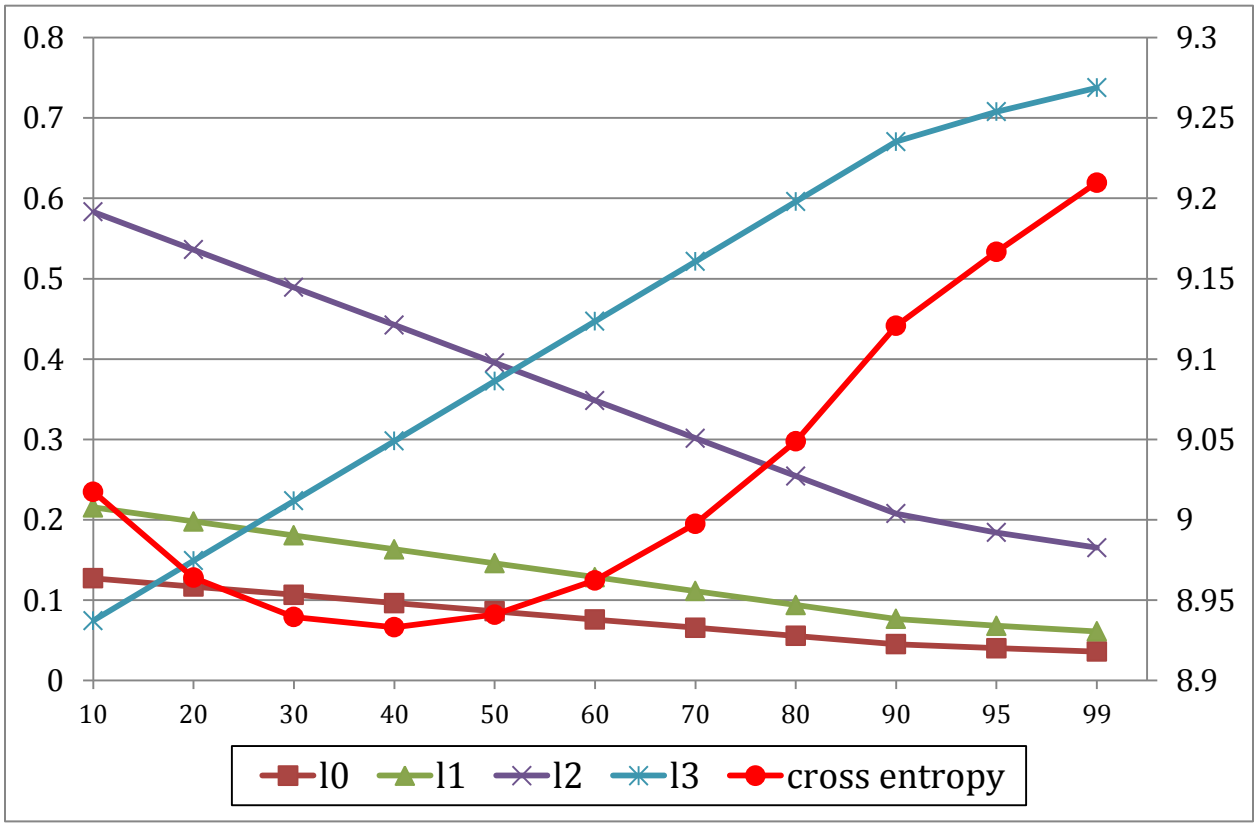
First 100% proportion which means without modification of lambda l[3]:

| proportion [%] | l[0] | l[1] | l[2] | l[3] | cross entropy |
|---|---|---|---|---|---|
| **100** | 0.1023 | 0.1733 | 0.4694 | 0.2549 | 8.934893 |

Then tweak the smoothing parameters in the following way: add 10%, 20%, 30%, ..., 90%, 95% and 99% of the difference between the trigram smoothing parameter and 1.0 to its value, discounting at the same the remaining three parameters proportionally:

| proportion [%] | l[0] | l[1] | l[2] | l[3] | cross entropy |
|---|---|---|---|---|---|
| **10** | 0.1271 | 0.2153 | 0.5831 | 0.0745 | 9.017483 |
| **20** | 0.1169 | 0.1979 | 0.5362 | 0.1490 | 8.963974 |

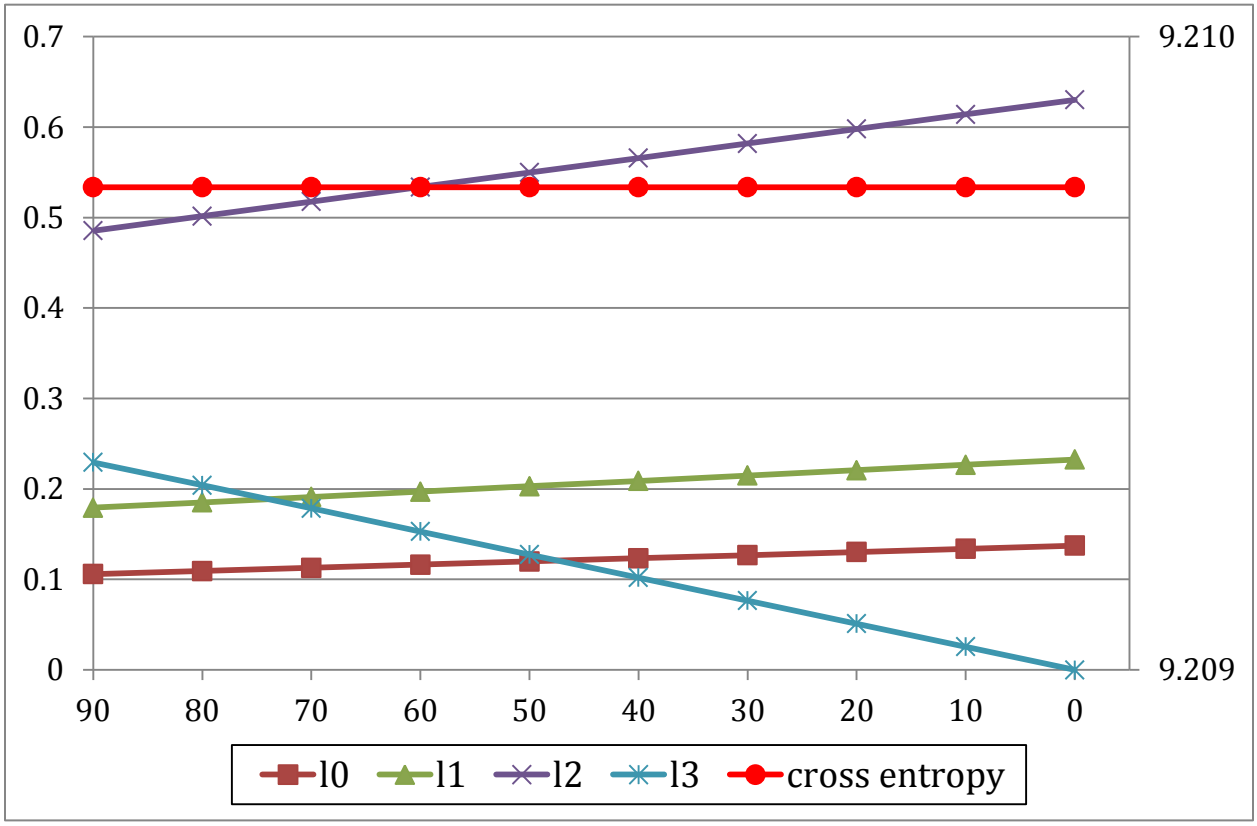
| | | | | | |
|---|---|---|---|---|---|
| **30** | 0.1066 | 0.1806 | 0.4892 | 0.2235 | 8.939504 |
| **40** | 0.0964 | 0.1633 | 0.4423 | 0.2980 | 8.933086 |
| **50** | 0.0862 | 0.1460 | 0.3954 | 0.3725 | 8.941035 |
| **60** | 0.0759 | 0.1286 | 0.3484 | 0.4470 | 8.962321 |
| **70** | 0.0657 | 0.1113 | 0.3015 | 0.5215 | 8.997537 |
| **80** | 0.0555 | 0.0940 | 0.2545 | 0.5960 | 9.048852 |
| **90** | 0.0452 | 0.0766 | 0.2076 | 0.6706 | 9.120675 |
| **95** | 0.0401 | 0.0680 | 0.1841 | 0.7078 | 9.166682 |
| **99** | 0.0360 | 0.0610 | 0.1653 | 0.7376 | 9.209762 |




Then set the trigram smoothing parameter to 90%, 80%, 70%, ... 10%, 0% of its value, boosting proportionally the other three parameters, again to sum up to one:

| proportion [%] | l[0] | l[1] | l[2] | l[3] | cross entropy |
|---|---|---|---|---|---|
| **90** | 0.1058 | 0.1792 | 0.4855 | 0.2294 | 9.209762 |
| **80** | 0.1093 | 0.1852 | 0.5016 | 0.2040 | 9.209762 |
| **70** | 0.1128 | 0.1911 | 0.5176 | 0.1785 | 9.209762 |
| **60** | 0.1163 | 0.1970 | 0.5337 | 0.1530 | 9.209762 |
| **50** | 0.1198 | 0.2030 | 0.5498 | 0.1275 | 9.209762 |
| **40** | 0.1233 | 0.2089 | 0.5658 | 0.1020 | 9.209762 |
| **30** | 0.1268 | 0.2148 | 0.5819 | 0.0765 | 9.209762 |

| | | | | | |
|---|---|---|---|---|---|
| **20** | 0.1303 | 0.2207 | 0.5980 | 0.0510 | 9.209762 |
| **10** | 0.1338 | 0.2267 | 0.6140 | 0.0255 | 9.209762 |
| **0** | 0.1373 | 0.2326 | 0.6301 | 0 | 9.209762 |




### 5.6.2 Cross Entropy of CZ text

Initial values based on split data for EN text:

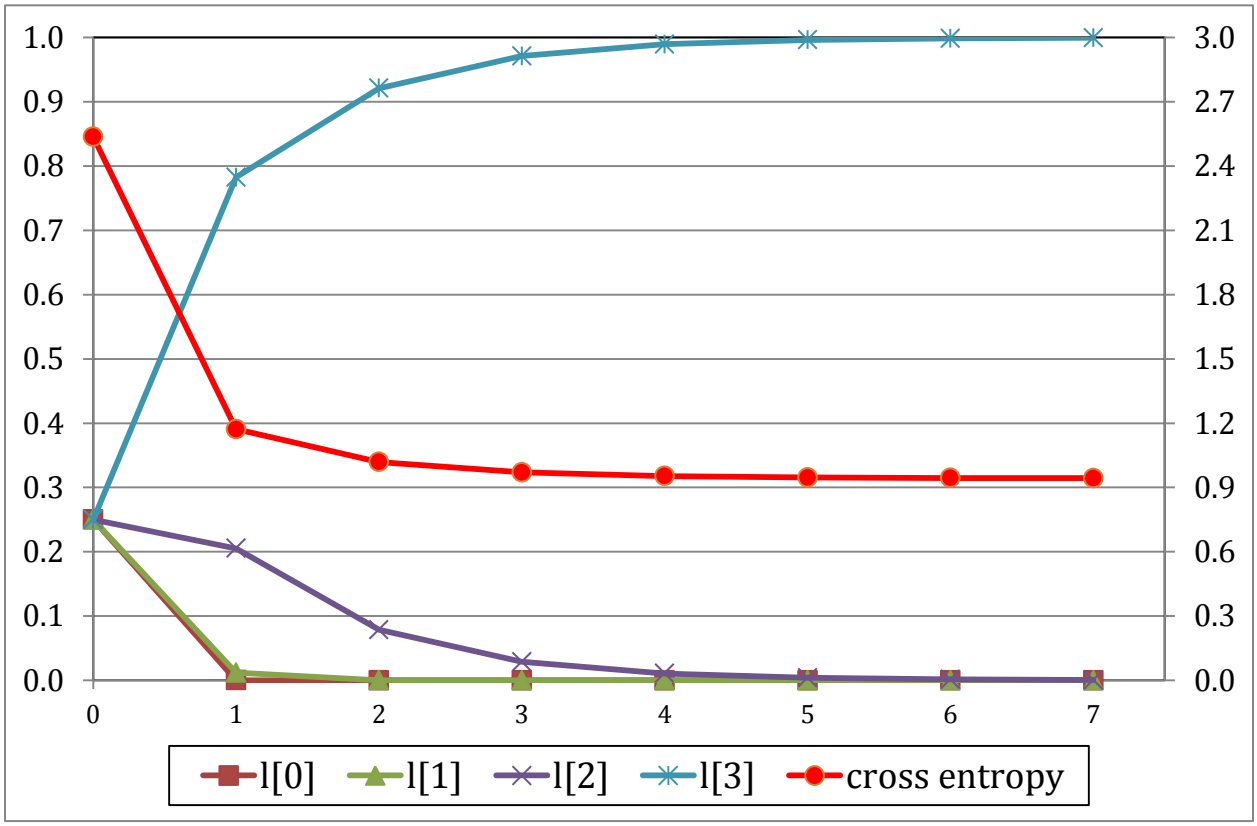
- training data length 162412
- held-out data length 40000
- testing data length 20000
- vocabulary size (V): 35303

#### 5.6.2.1 Training smoothing parameters

Cross entropy and learning lambda on training data, just 7 iterations were enough for commit condition that difference of all lambda between two iterations are < 0.001. Anyway it is obvious that lambda with index 3 is converging to 1

| iteration | l[0] | l[1] | l[2] | l[3] | cross entropy |
|---|---|---|---|---|---|
| **0** | 0.250000 | 0.250000 | 0.250000 | 0.250000 | 2.536295 |
| **1** | 0.000219 | 0.011867 | 0.205397 | 0.782517 | 1.172338 |
| **2** | 0.000000 | 0.000278 | 0.078683 | 0.921039 | 1.018883 |
| **3** | 0.000000 | 0.000006 | 0.028920 | 0.971074 | 0.970345 |

| | | | | | |
|---|---|---|---|---|---|
| **4** | 0.000000 | 0.000000 | 0.010533 | 0.989467 | 0.953118 |
| **5** | 0.000000 | 0.000000 | 0.003827 | 0.996173 | 0.946911 |
| **6** | 0.000000 | 0.000000 | 0.001389 | 0.998611 | 0.944666 |
| **7** | 0.000000 | 0.000000 | 0.000504 | 0.999496 | 0.943851 |



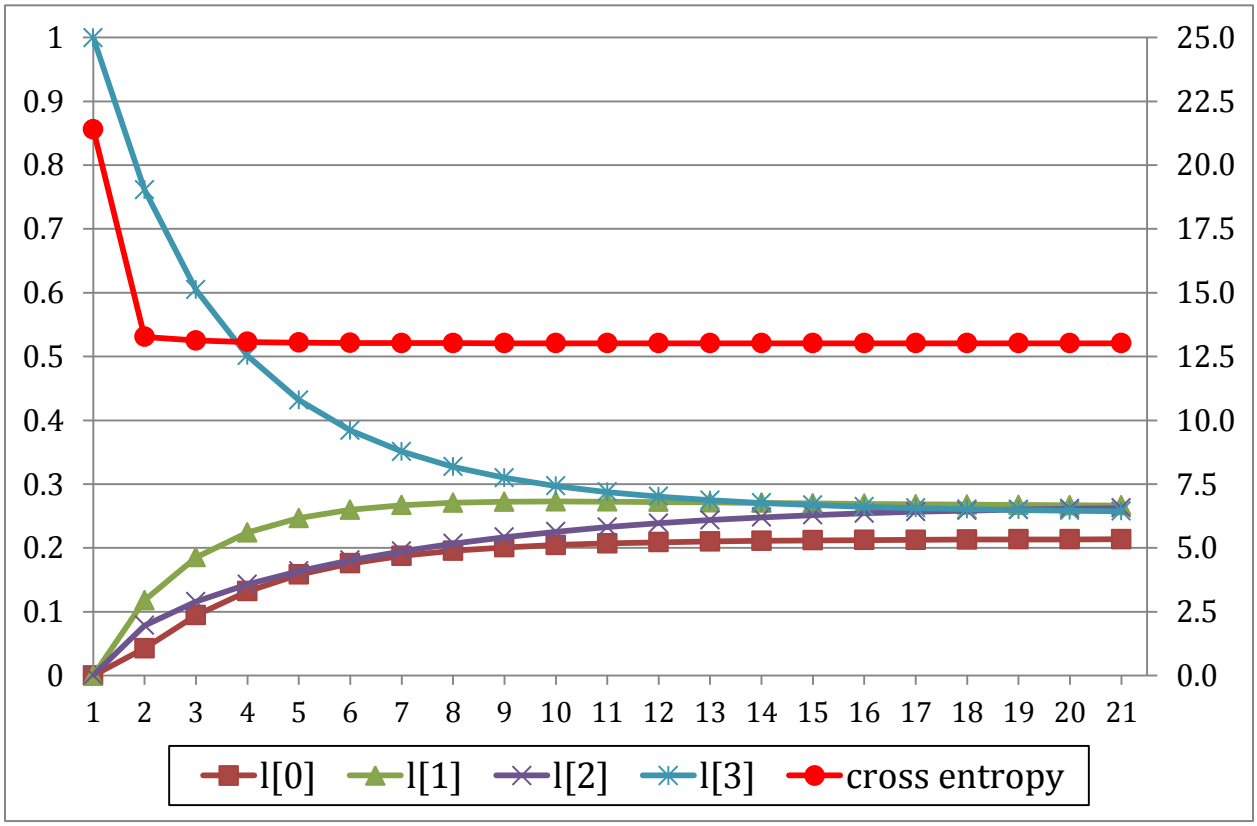

Cross entropy on test data based on trained lambda from training data:

| iteration | l[0] | l[1] | l[2] | l[3] | cross entropy |
|---|---|---|---|---|---|
| **TEST** | 0.000000 | 0.000000 | 0.000504 | 0.999496 | 20.898289 |

Cross entropy and learning lambda on held-out data, X iterations were enough to satisfy difference condition of all lambda < 0.001. Currently lambdas are spread more equally.

| iteration | l[0] | l[1] | l[2] | l[3] | cross entropy |
|---|---|---|---|---|---|
| **0** | 0.000000 | 0.000000 | 0.000504 | 0.999496 | 21.402806 |
| **1** | 0.042600 | 0.117650 | 0.078502 | 0.761248 | 13.272542 |
| **2** | 0.094396 | 0.185153 | 0.115870 | 0.604581 | 13.123375 |
| **3** | 0.132094 | 0.223984 | 0.142740 | 0.501182 | 13.068853 |
| **4** | 0.158028 | 0.246611 | 0.163629 | 0.431732 | 13.044456 |
| **5** | 0.175588 | 0.259637 | 0.180625 | 0.384150 | 13.032456 |
| **6** | 0.187426 | 0.266887 | 0.194809 | 0.350879 | 13.026163 |
| **7** | 0.195415 | 0.270653 | 0.206780 | 0.327152 | 13.022689 |

| | | | | | |
|---|---|---|---|---|---|
| **8** | 0.200832 | 0.272337 | 0.216912 | 0.309919 | 13.020687 |
| **9** | 0.204533 | 0.272800 | 0.225475 | 0.297192 | 13.019491 |
| **10** | 0.207089 | 0.272575 | 0.232682 | 0.287654 | 13.018757 |
| **11** | 0.208878 | 0.271988 | 0.238722 | 0.280412 | 13.018296 |
| **12** | 0.210149 | 0.271237 | 0.243761 | 0.274853 | 13.018001 |
| **13** | 0.211070 | 0.270439 | 0.247946 | 0.270545 | 13.017812 |
| **14** | 0.211749 | 0.269661 | 0.251411 | 0.267179 | 13.017688 |
| **15** | 0.212260 | 0.268939 | 0.254271 | 0.264531 | 13.017607 |
| **16** | 0.212651 | 0.268288 | 0.256625 | 0.262436 | 13.017554 |
| **17** | 0.212956 | 0.267714 | 0.258559 | 0.260771 | 13.017519 |
| **18** | 0.213198 | 0.267215 | 0.260144 | 0.259442 | 13.017496 |
| **19** | 0.213392 | 0.266787 | 0.261443 | 0.258379 | 13.017481 |
| **20** | 0.213549 | 0.266421 | 0.262505 | 0.257525 | 13.017470 |




Cross entropy on test data based on trained lambda:

| iteration | l[0] | l[1] | l[2] | l[3] | cross entropy |
|---|---|---|---|---|---|
| **TEST** | 0.213549 | 0.266421 | 0.262505 | 0.257525 | 12.488975 |

#### 5.6.2.2 Tweaking smoothing parameters

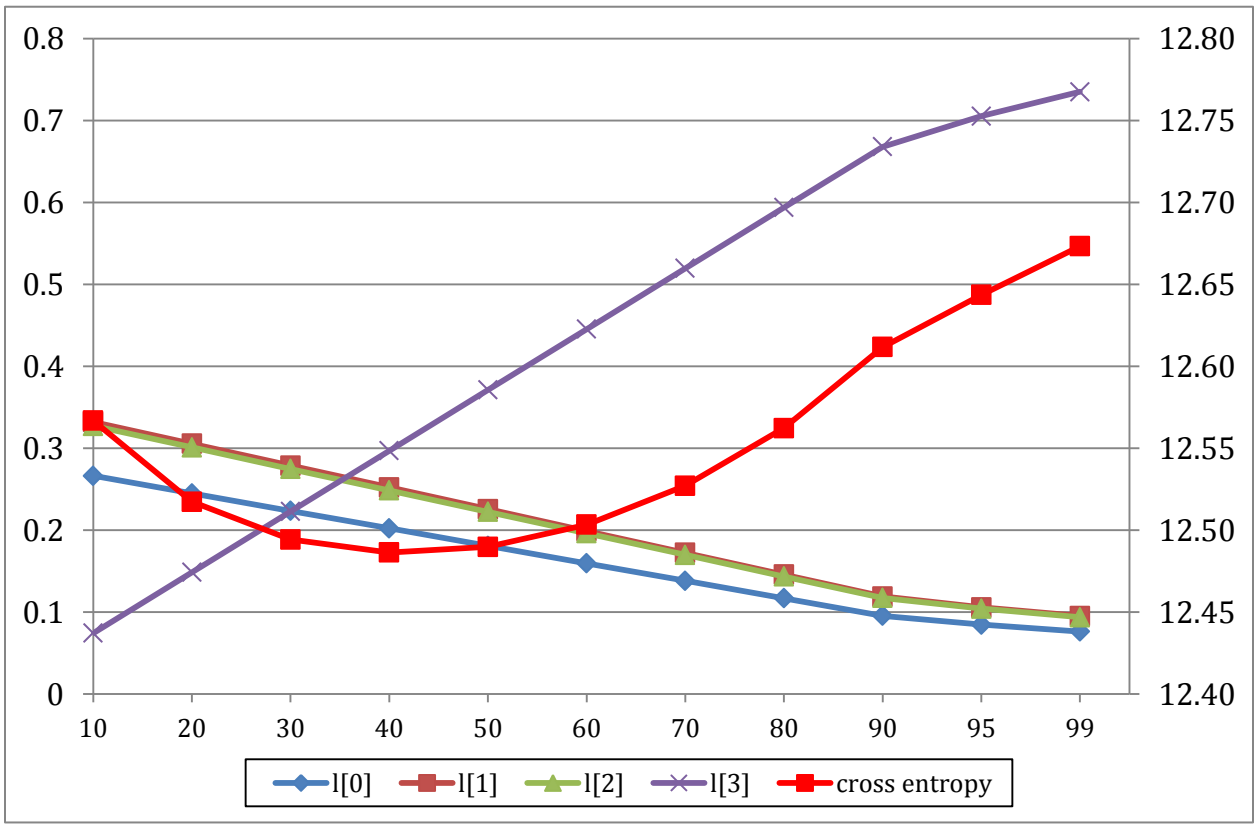
Tweaking lambda l[3] and calculation of cross entropy is then in next few tables.

First 100% proportion which means without modification of lambda l[3]:

| proportion | l[0] | l[1] | l[2] | l[3] | cross entropy |
|---|---|---|---|---|---|
| **100** | 0.2135 | 0.2664 | 0.2625 | 0.2575 | 12.48897 |

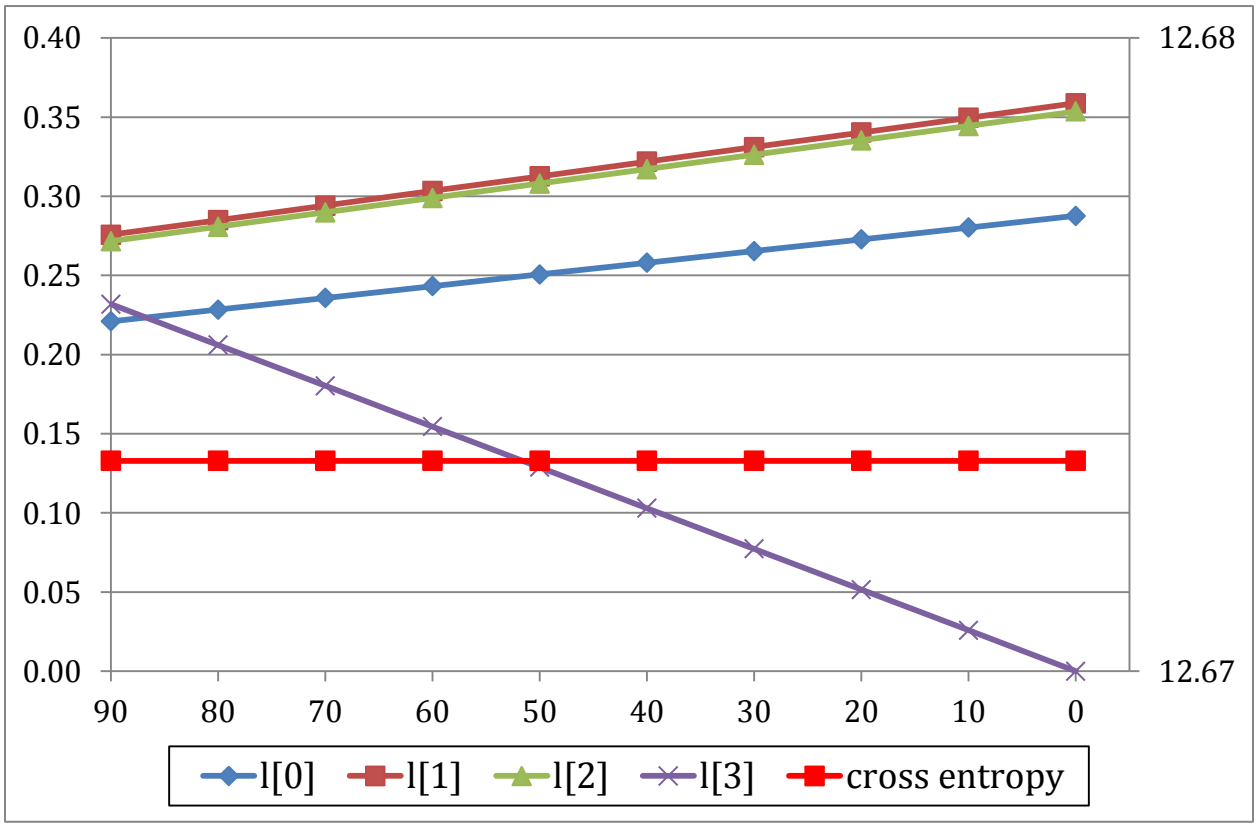
Then tweak the smoothing parameters in the following way: add 10%, 20%, 30%, ..., 90%, 95% and 99% of the difference between the trigram smoothing parameter and 1.0 to its value, discounting at the same the remaining three parameters proportionally:

| proportion | l[0] | l[1] | l[2] | l[3] | cross entropy |
|---|---|---|---|---|---|
| **10** | 0.2663 | 0.3322 | 0.3273 | 0.0742 | 12.56689 |
| **20** | 0.2449 | 0.3055 | 0.3011 | 0.1485 | 12.51733 |
| **30** | 0.2236 | 0.2789 | 0.2748 | 0.2227 | 12.49425 |
| **40** | 0.2022 | 0.2523 | 0.2486 | 0.2970 | 12.48634 |
| **50** | 0.1808 | 0.2256 | 0.2223 | 0.3712 | 12.48980 |
| **60** | 0.1595 | 0.1990 | 0.1961 | 0.4455 | 12.50334 |
| **70** | 0.1381 | 0.1723 | 0.1698 | 0.5197 | 12.52705 |
| **80** | 0.1168 | 0.1457 | 0.1435 | 0.5940 | 12.56224 |
| **90** | 0.0954 | 0.1190 | 0.1173 | 0.6682 | 12.61181 |
| **95** | 0.0847 | 0.1057 | 0.1042 | 0.7054 | 12.64358 |
| **99** | 0.0762 | 0.0951 | 0.0937 | 0.7351 | 12.67332 |



Then set the trigram smoothing parameter to 90%, 80%, 70%, ... 10%, 0% of its value, boosting proportionally the other three parameters, again to sum up to one:

| proportion | l[0] | l[1] | l[2] | l[3] | cross entropy |
|---|---|---|---|---|---|
| **90** | 0.2210 | 0.2757 | 0.2716 | 0.2318 | 12.67332 |
| **80** | 0.2284 | 0.2849 | 0.2807 | 0.2060 | 12.67332 |
| **70** | 0.2358 | 0.2941 | 0.2898 | 0.1803 | 12.67332 |
| **60** | 0.2432 | 0.3034 | 0.2989 | 0.1545 | 12.67332 |
| **50** | 0.2506 | 0.3126 | 0.3080 | 0.1288 | 12.67332 |
| **40** | 0.2580 | 0.3219 | 0.3171 | 0.1030 | 12.67332 |
| **30** | 0.2654 | 0.3311 | 0.3262 | 0.0773 | 12.67332 |
| **20** | 0.2728 | 0.3403 | 0.3353 | 0.0515 | 12.67332 |
| **10** | 0.2802 | 0.3496 | 0.3444 | 0.0258 | 12.67332 |
| **0** | 0.2876 | 0.3588 | 0.3536 | 0 | 12.67332 |



## 5.7 Conclusions

Expectations and observations for calculating lambda:

- Calculated lambda from training data always converge lambda l3 to 1 and rest to 0 which is expected.
- When lambda l3 is close to 1 when it is calculated from Training data (let's say > 0.9) then Cross Entropy is converge really fast to its final value.
- When there is more equality in calculation of lambda from Held-out data between lambda l0 – l3 (lambda l3 is about 0.5 after is calculated from training data) then Cross Entropy is starting to converge to its final value very quickly.
- Except Cross Entropy calculated from Training data is CZ text Cross Entropy higher than Cross Entropy of EN text (see table below).

| | **EN text** | **CZ text** | |
|---|---|---|---|
| | **Cross H** | **Cross H** | **Value notice** |
| **Training data** | 2.2 | 0.9 | converge to this value |
| **Test data (lambda from training data)** | 15.1 | 20.9 | 1 value calculated |
| **Held out data** | 8.9 | 13.0 | converge to this value |
| **Test data (lambda from held out data)** | 8.9 | 12.5 | 1 value calculated |

Expectations and observations from tweaking lambda:

- Boosting of lambda l3 is leading to convex function of Cross Entropy values based on proportion.
- By comparison of the 3rd grade polynomial function for EN text and CZ text which interpolate convex graph of Cross Entropy from 99.6 % is obvious that curves are similar except constant element:
  - EN text: $y = -0.0006\ x^3 + 0.0161\ x^2 - 0.0992\ x + 9.1$
  - CZ text: $y = -0.0005\ x^3 + 0.0131\ x^2 - 0.0838\ x + 12.6$
- Discounted lambda l3 is leading constant value of Cross Entropy values no matter on proportion. This constant corresponds to constant (zero) element of 3rd grade polynomial function.
- Cross Entropy of Czech text is higher than Cross Entropy of EN text (see table below).

| | EN text | CZ text | |
|---|---|---|---|
| | **Cross H** | **Cross H** | **Value notice** |
| Boosted l3 | 9.1 | 12.6 | Convex graph |
| Discounted l3 | 9.2 | 12.7 | Constant graph |