# NLP ASSIGNMENT 1 PID: A69034019

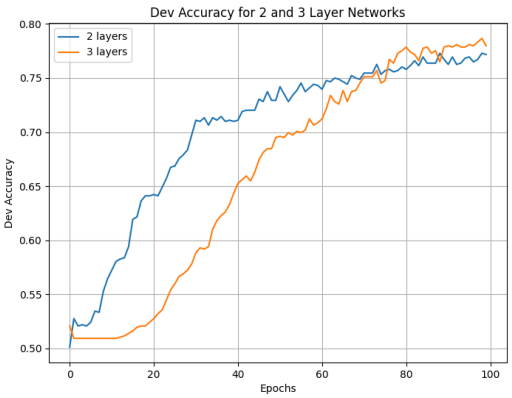
## QUESTION 1

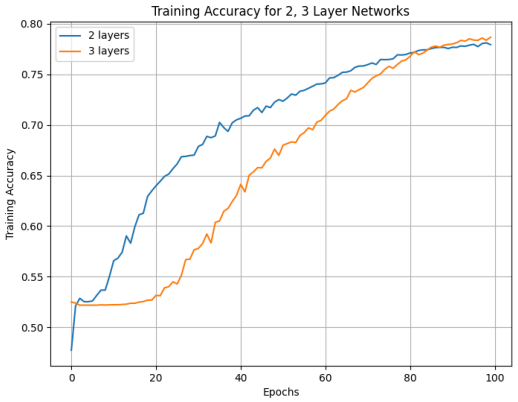
LOGIC: Got word embeddings from glove using get_word_embedding and took mean for getting the embed_vec.
EXPERIMENTS:

1. No of layers: 2 layers have more accuracy than 3 at the end because of overfitting.
2. Hidden layer size: 100 is the below image output. default size. by increasing to 200:
   3 LAYERS
   a. Epoch #80: train accuracy 1.000, dev accuracy 0.784 train loss 0.001, test losss 1.511
   b. Epoch #90: train accuracy 1.000, dev accuracy 0.786 train loss 0.001, test losss 1.571
   c. Epoch #100: train accuracy 1.000, dev accuracy 0.790 train loss 0.001, test losss 1.679

   2 LAYERS:
   Epoch #80: train accuracy 0.886, dev accuracy 0.783 train loss 0.282, test losss 0.480
   Epoch #90: train accuracy 0.899, dev accuracy 0.788 train loss 0.260, test losss 0.483
   Epoch #100: train accuracy 0.911, dev accuracy 0.795 train loss 0.241, test losss0.483
   Increasing hidden layers increases accuracy.
3. source of embedding: 50 vs 300d - 300 has more accuracy than 50. 50 doesn't reach 77 percent accuracy for a simple 2 layer neural network but 300 reaches. Time taken for 77 percent accuracy < 2min.
4. Dropout layers: made less difference in DAN training compared to RANDAN. In DAN accuracy improved marginally.
5. Optimizer: SGD:
   2 layers:
   i. Epoch #10: train accuracy 0.550, dev accuracy 0.564 train loss 0.683, test losss 0.683
   ii. Epoch #20: train accuracy 0.635, dev accuracy 0.641 train loss 0.668, test losss 0.666
   iii. Epoch #70: train accuracy 0.758, dev accuracy 0.749 train loss 0.529, test losss 0.523
   iv. Epoch #100: train accuracy 0.779, dev accuracy 0.772 train loss 0.476, test losss 0.476
   v. 3 layers:
   vi. Epoch #10: train accuracy 0.522, dev accuracy 0.509 train loss 0.690, test losss 0.691
   vii. Epoch #20: train accuracy 0.527, dev accuracy 0.524 train loss 0.686, test losss 0.687
   viii. Epoch #70: train accuracy 0.737, dev accuracy 0.745 train loss 0.552, test losss 0.543
   ix. Epoch #100: train accuracy 0.787, dev accuracy 0.780 train loss 0.453, test losss 0.465

Adam vs SGD: More accuracy with SGD as it works well with large datasets, The way it converges is different from ADAM.

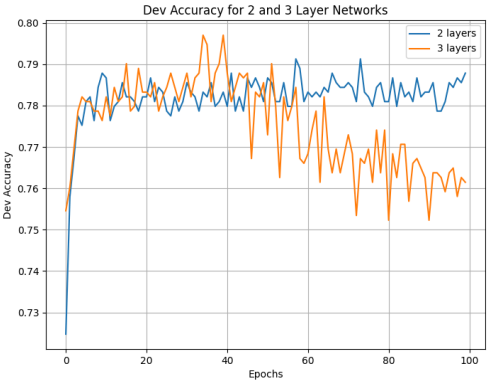


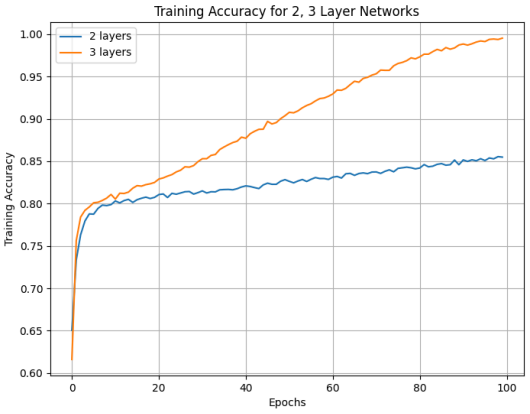




Adam:

```
2 layers:
Epoch #10: train accuracy 0.799, dev accuracy 0.788 train loss 0.431, test losss 0.454
Epoch #20: train accuracy 0.807, dev accuracy 0.782 train loss 0.414, test losss 0.459
Epoch #30: train accuracy 0.813, dev accuracy 0.781 train loss 0.402, test losss 0.458
Epoch #40: train accuracy 0.820, dev accuracy 0.783 train loss 0.392, test losss 0.456
Epoch #50: train accuracy 0.828, dev accuracy 0.781 train loss 0.382, test losss 0.456
Epoch #60: train accuracy 0.829, dev accuracy 0.781 train loss 0.371, test losss 0.457
Epoch #70: train accuracy 0.837, dev accuracy 0.784 train loss 0.361, test losss 0.469
Epoch #80: train accuracy 0.841, dev accuracy 0.781 train loss 0.351, test losss 0.460
Epoch #90: train accuracy 0.846, dev accuracy 0.783 train loss 0.341, test losss 0.466
Epoch #100: train accuracy 0.855, dev accuracy 0.788 train loss 0.330, test losss 0.466

3 layers:
Epoch #10: train accuracy 0.811, dev accuracy 0.776 train loss 0.413, test losss 0.447
Epoch #20: train accuracy 0.825, dev accuracy 0.783 train loss 0.383, test losss 0.455
Epoch #30: train accuracy 0.850, dev accuracy 0.784 train loss 0.342, test losss 0.464
Epoch #40: train accuracy 0.878, dev accuracy 0.797 train loss 0.294, test losss 0.481
Epoch #50: train accuracy 0.904, dev accuracy 0.786 train loss 0.246, test losss 0.531
Epoch #60: train accuracy 0.927, dev accuracy 0.766 train loss 0.193, test losss 0.589
Epoch #70: train accuracy 0.952, dev accuracy 0.768 train loss 0.143, test losss 0.651
Epoch #80: train accuracy 0.971, dev accuracy 0.774 train loss 0.100, test losss 0.770
Epoch #90: train accuracy 0.987, dev accuracy 0.763 train loss 0.061, test losss 0.904
Epoch #100: train accuracy 0.995, dev accuracy 0.761 train loss 0.032, test losss 1.086
```



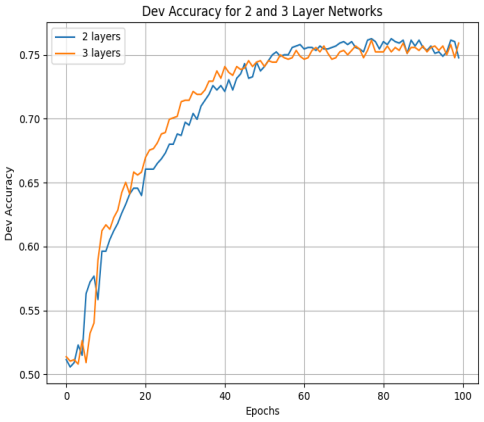

1b) How does this compare to using GloVe? RANDAN has a bad accuracy in the beginning but increases as it learns.

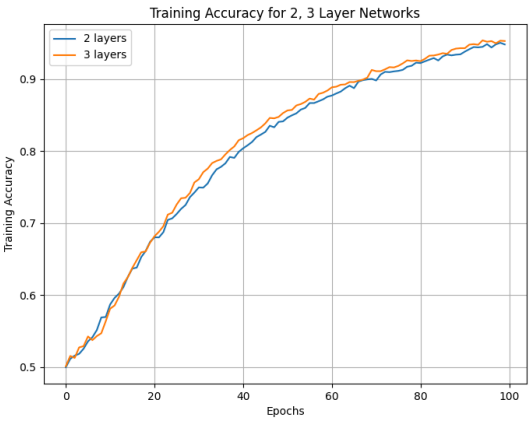
Discuss how training embeddings from scratch affect model convergence and performance:
RANDAN converges slower and less stable than DAN. Also, it has lesser accuracy(less generalization to unseen data) Have to give dropouts for it to train faster. else takes a lot of time.

RANDAN LOGIC: Got word index from the glove and initialised word embedding in Pytorch.

EXPERIMENTS

6. No of layers: 2 layers have more accuracy than 3 at the end because of overfitting.
7. Hidden layer size: increased to 200:
    a. 2 layers
        i. Epoch #70: train accuracy 0.919, dev accuracy 0.751 train loss 0.212, test losss 0.616
        ii. Epoch #80: train accuracy 0.936, dev accuracy 0.776 train loss 0.172, test losss 0.651
    b. 3 layers
        i. Epoch #70: train accuracy 0.918, dev accuracy 0.758 train loss 0.200, test losss 0.717
        ii. Epoch #80: train accuracy 0.938, dev accuracy 0.756 train loss 0.156, test losss 0.782
        iii. Epoch #90: train accuracy 0.954, dev accuracy 0.758 train loss 0.125, test losss 0.893
    c. greater size of hidden layer gives more accuracy.
8. Dropout layers: made less difference in DAN training compared to RANDAN. In DAN accuracy improved marginally. increasing the rate too much will underfit but too less results in overfitting.
    a. dropout rate: 0.5 - images below show accuracy
    b. dropout rate: 0.4: got better accuracy than 0.5
9. Optimizer: SGD gives bad accuracy since it can get stuck in local minima. Epoch #80: train accuracy 0.530, dev accuracy 0.518 train loss 0.690, test losss 0.690
10. Epoch #90: train accuracy 0.537, dev accuracy 0.521 train loss 0.689, test losss 0.690




## QUESTION 2

For BPE  2layers give more accuracy than in DAN as compared to three layers since the model overfits for 3 layers.
VOCAB :2000

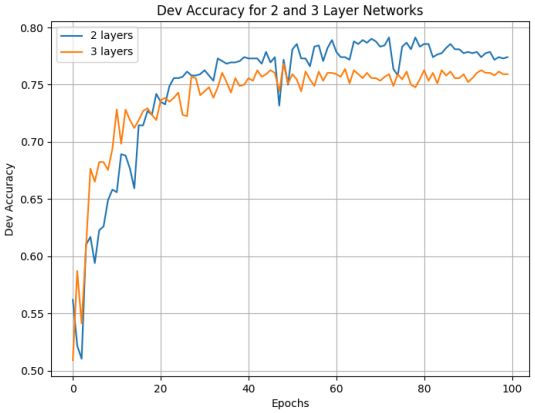
```
2 layers:
Read in 14923 vectors of size 300
Epoch #10: train accuracy 0.623, dev accuracy 0.673 train loss 0.653, test losss 0.639
Epoch #20: train accuracy 0.774, dev accuracy 0.737 train loss 0.488, test losss 0.533
Epoch #30: train accuracy 0.860, dev accuracy 0.774 train loss 0.333, test losss 0.501
Epoch #40: train accuracy 0.906, dev accuracy 0.787 train loss 0.236, test losss 0.537
Epoch #50: train accuracy 0.933, dev accuracy 0.764 train loss 0.171, test losss 0.644
Epoch #60: train accuracy 0.957, dev accuracy 0.779 train loss 0.117, test losss 0.710
Epoch #70: train accuracy 0.971, dev accuracy 0.780 train loss 0.083, test losss 0.829
Epoch #80: train accuracy 0.980, dev accuracy 0.782 train loss 0.062, test losss 0.942
Epoch #90: train accuracy 0.985, dev accuracy 0.771 train loss 0.047, test losss 1.026
Epoch #100: train accuracy 0.989, dev accuracy 0.786 train loss 0.034, test losss 1.136
```

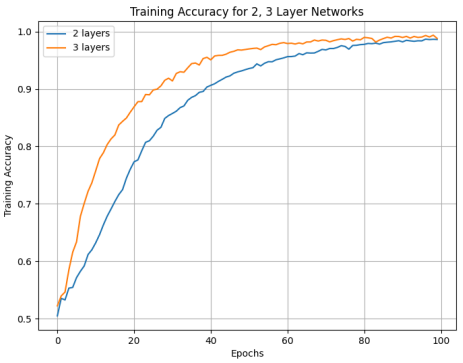
```
3 layers:
Read in 14923 vectors of size 300
Epoch #10: train accuracy 0.735, dev accuracy 0.703 train loss 0.527, test losss 0.572
Epoch #20: train accuracy 0.866, dev accuracy 0.740 train loss 0.312, test losss 0.594
Epoch #30: train accuracy 0.927, dev accuracy 0.743 train loss 0.189, test losss 0.707
Epoch #40: train accuracy 0.957, dev accuracy 0.747 train loss 0.118, test losss 0.879
Epoch #50: train accuracy 0.973, dev accuracy 0.751 train loss 0.075, test losss 1.022
Epoch #60: train accuracy 0.984, dev accuracy 0.757 train loss 0.049, test losss 1.235
Epoch #70: train accuracy 0.991, dev accuracy 0.763 train loss 0.032, test losss 1.320
Epoch #80: train accuracy 0.991, dev accuracy 0.752 train loss 0.028, test losss 1.595
Epoch #90: train accuracy 0.995, dev accuracy 0.758 train loss 0.017, test losss 1.734
Epoch #100: train accuracy 0.992, dev accuracy 0.758 train loss 0.021, test losss 1.703
```

# VOCAB : 1500

```
2 layers:
Read in 14923 vectors of size 300
Epoch #10: train accuracy 0.620, dev accuracy 0.658 train loss 0.654, test losss 0.643
Epoch #20: train accuracy 0.760, dev accuracy 0.742 train loss 0.513, test losss 0.544
Epoch #30: train accuracy 0.854, dev accuracy 0.759 train loss 0.353, test losss 0.509
Epoch #40: train accuracy 0.903, dev accuracy 0.774 train loss 0.248, test losss 0.527
Epoch #50: train accuracy 0.934, dev accuracy 0.750 train loss 0.170, test losss 0.642
Epoch #60: train accuracy 0.954, dev accuracy 0.789 train loss 0.123, test losss 0.666
Epoch #70: train accuracy 0.969, dev accuracy 0.788 train loss 0.091, test losss 0.774
Epoch #80: train accuracy 0.977, dev accuracy 0.783 train loss 0.065, test losss 0.887
Epoch #90: train accuracy 0.984, dev accuracy 0.778 train loss 0.050, test losss 0.975
Epoch #100: train accuracy 0.986, dev accuracy 0.774 train loss 0.042, test losss 1.064

3 layers:
Read in 14923 vectors of size 300
Epoch #10: train accuracy 0.737, dev accuracy 0.694 train loss 0.524, test losss 0.567
Epoch #20: train accuracy 0.860, dev accuracy 0.719 train loss 0.331, test losss 0.596
Epoch #30: train accuracy 0.919, dev accuracy 0.741 train loss 0.203, test losss 0.699
Epoch #40: train accuracy 0.955, dev accuracy 0.750 train loss 0.128, test losss 0.842
Epoch #50: train accuracy 0.969, dev accuracy 0.751 train loss 0.086, test losss 0.906
Epoch #60: train accuracy 0.981, dev accuracy 0.760 train loss 0.059, test losss 1.078
Epoch #70: train accuracy 0.985, dev accuracy 0.756 train loss 0.042, test losss 1.236
Epoch #80: train accuracy 0.986, dev accuracy 0.755 train loss 0.039, test losss 1.264
Epoch #90: train accuracy 0.991, dev accuracy 0.759 train loss 0.024, test losss 1.562
Epoch #100: train accuracy 0.988, dev accuracy 0.759 train loss 0.036, test losss 1.537
```



We can observe that reducing vocab size gives less accuracy. Reasons:

1. Small vocab size means less words: so the more frequent pairs maybe missed.
2. if a word is not present, BPE breaks into subwords which may not be correct semantic meaning.
3. smaller size => captures less meaning from the text
4. multiple words can have the same set of subwords after the split so it can not generalize well.

Compare the performance of your subword-based DAN model to the word-level DAN model you implemented earlier: BPE/SUBWORD-based model converges slower than DAN and gives higher accuracy than DAN. (expected trend)

QUESTION 3: Attached is a handwritten pdf after references.

INSTRUCTIONS TO EXECUTE CODE: Readme.md

## REFERENCES:


1. https://www.youtube.com/watch?v=zduSFxRajkE&t=20s
2. Pytorch documentation
3. DAN: slides and sentiment.py understanding was enough to implement.
4. SKIPGRAM: using slides and online examples.
5. Stackoverflow, pytorch forums for errors: mostly debugging RANDAN.
    a. x = .xlong() error
    b. size issues: neural network
    c. embedding errors for Randan
6. Chatgpt - for errors ( though used checked forums and PyTorch subs for right error corrections, understand some parts of bpe, debugging RANDAN. to check some small mistakes like loop indices and syntax for IN bpe. (MY Loop was giving wrong answers for debugging. couldn't correct - debugged for a long time. then used this directly)

```
7.                      w = w.replace(''.join(p),p[0]+' '+p[1])
8.                  final_txt.extend(w.split())
```

ASSIGN 1
A69034019

PART ③

Q① (a):-

skip gram model:

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

$o$ - context word, $c$ - center word

$u_w$ - context word vector
$v_w$ - center word vector.

$K=1$ $d=2$

$u_{w\ dog,cat} = (0,1)$
$u_{w\ a,the} = (1,0)$

$$\text{likelihood} = \sum_{(x,y)} \log(P(y|x))$$

| WORD | CONTEXT WORD |
|---|---|
| the | dog, cat |
| dog | the, a |
| cat | the |
| a | dog |

ANS: $P(dog|the) = 1/2$ $P(cat/the) = 1/2$

$$\Rightarrow P(y|the) = \begin{cases} 1/2 & y = dog \\ 1/2 & y = cat \\ 0 & \text{otherwise} \end{cases}$$

EXPLANATION:
the above is case because both dog & cat have equal likelihood of appearing after the. (they appear exactly once).

If we have $P(dog|the) = 0.8$ $P(cat|the) = 0.2$ then it means dog has more probability of appearing after the than cat. which is false.

PART (3)

Q (1) (B)

Lets say $u_{w,the} = (x, y)$, $u_{w,dog,cat} = (0,1)$ } take dot product. $\Rightarrow y$

$$\Rightarrow \begin{matrix} P(dog \mid the), \\ P(cat \mid the) \end{matrix} = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

$$\Rightarrow \cancel{\text{if}}$$

$$= \frac{e^y}{e^x + e^x + e^y + e^y}$$

$$= \frac{e^y}{2e^x + 2e^y}$$

For it to be optimal, ① $P(dog \mid the) \simeq P(cat \mid the)$ and

② $P(dog \mid the) \simeq 0.5$

$P(cat \mid the) \simeq 0.5$

since probability of them appearing is almost half using the given sentences.

Cases: (trying out values for (x,y)):-

$x=0, y=0$ $\quad P(dog \mid the) = \frac{e^0}{2e^0 + 2e^0} = \frac{1}{4} = 0.25$

$x=1, y=0$ $\quad P(dog \mid the) = \frac{e^0}{2e^0 + 2e^1} = \frac{1}{2+2e} \simeq 0.184$

$x=0, y=1$ $\quad P(dog \mid the) = \frac{e^1}{2e^0 + 2e^1} \simeq 0.365$

$x=0, y=5$ $\quad P(dog \mid the) = \frac{e^5}{2e^0 + 2e^5} \simeq 0.496.$

So x can't be large than y (since denominator will be very larger than Numerator so P↓)

$x=0$ $y=10$ $\quad P(dog \mid the) = \frac{e^{10}}{2e^0 + 2e^{10}} \simeq 0.5$

↓ so keep least x ($x=0$)

So the ans: ($x = 0$, $y$ = any large no.) the larger the no. the better $P \to 0.5$.

$$\lim_{y \to \infty} \frac{e^y}{2e^x + 2e^y} \simeq \frac{e^y}{2e^y} \simeq 1/2.$$

($x$ very small)

ANS:- ($x = 0$, $y$ = large_no.)

PART (3) - (3C)

u = context word  V = center word)

Training Set: (u = the, v = dog)
(u = dog, v = the)
(u = the, v = cat)
(u = a, v = dog)
(u = cat, v = the)
(u = dog, v = a)

PART (3) - (3D)

To maximize skip gram obj. $P(\text{dog}|\text{the}) \simeq P(\text{cat}|\text{the}) \simeq 0.5$.

Taking $v_c$ as (0,1) and $u_o$ as all (0,x) (simple case)

since from (3b) - 0, large no. gives $P(\text{cat}|\text{the}) \sim 0.5$ & $P(\text{dog}|\text{the})$

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

$v_{\text{the, dog, cat, a}} = (0,1)$

so $x$ can be.

(if $v$'s are $(0,1)$ then $u$ can't be $(x,0)$ since dot product $0$)

$P(0|c) \Rightarrow P(\text{dog}|\text{the}) \simeq P(\text{cat}|\text{the}) \simeq P(a|\text{the}) \simeq 0.25$

since its: $\frac{e^x}{e^x+e^x+e^x+e^x} \simeq \frac{e^x}{4e^x} \simeq 1/4$

But $P(\text{dog}|\text{the}) \simeq P(\text{cat}|\text{the}) \simeq 0.5$ & $P(a|\text{the}) \simeq 0.1$ or less.

so $u$ for: $(0,10)$ : can be 10 or large no°

dog, cat → should be ~~0~~. as $P(\text{the}|\text{the}) \simeq 0$.

$u_{\text{the}}, a \simeq 0$ small $(0,1)$

$\Rightarrow P(\text{dog}|\text{the}) = \frac{e^{10}}{e^1+e^1+2e^{10}} \simeq 1/2$

so greater the value of $(0,x)$ more $P \to 0.5$

ANS: $u_{\text{dog}} = (0, \text{large no.})$

$u_{\text{cat}} = (0, \text{large no.})$

$u_{\text{the}} \simeq (0, \text{small no.})$

$u_{a} \simeq (0, \text{small no.})$

$v_{\text{cat, dog, a, the}} = (0,1)$

(context vectors)