

Sampling:

$N[0] \rightarrow$ raw counts of names starting w' char e.

$p = N[0].float.$

$p = p / p.sum(). \rightarrow \therefore p = \text{prob of a word.name}$
starting w' c.

g. Generator(). manual-seed(). \rightarrow seed gen.

~~g~~
 $p = \text{torch.rand}(3, \text{generator}=g) \rightarrow$ Tensor with
 $p = p / p.sum()$ \rightarrow normalize. 3 elements b/w 0,1.

Now, $\text{torch.multinomial}(p, \text{num-samples}=20, \text{generator}=g,$
 $\uparrow \quad \uparrow \quad \uparrow$
tensor no. of replacement
with prop. samples = True)

returns a sample tensor of dim: $(1 \times \text{num-samples})$

loop to print a name based on bigram probability

$ix = 0$
 $\text{word} = []$

while True:

$p = N[ix].float(); p = p / p.sum()$

$ix = \text{torch.multinomial}(p, \text{rep}=\text{True})$

$\text{word.append}[\text{itos}[ix]]$

if (~~if~~ $ix == 0$):

break

shorter
name

• then
of coz its L

Now to make a matrix P of prob distr:

$$P_0 = N.\text{float}().$$

↪ $P.\text{sum}() = ([\dots]) \rightarrow$ single item = sum of all in N. X

Broadcasting rules:

$P.\text{sum}(\text{dim}, \text{keepdim} = \text{True})$

↑
dim across which we sum. false by default.

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \xrightarrow{P.\text{sum}(0, \text{keepdim} = \text{true}) \quad \text{across rows}} \begin{bmatrix} 9, 12 \end{bmatrix} \quad 1 \times 2.$$

3×2

$\left\{ \begin{array}{l} P.\text{sum}(1, \text{keepdim} = \text{True}) \\ \text{across coln} \end{array} \right.$

$$\begin{bmatrix} [3] \\ [7] \\ [11] \end{bmatrix}$$

3×1

$$\begin{bmatrix} 1, 3 \\ 2, 7 \\ 5, 1 \end{bmatrix}$$

$P.\text{sum}(0)$

$[9, 12]$.
single dim 2 elem

$$\begin{bmatrix} 3 \\ 7 \\ 11 \end{bmatrix} = [3, 7, 11]$$

1-d tensor. = 3 elem

$$\{2^0, \}$$



we can verify the above by using .shape

Broadcasting Semantics :

to have tensor operations:

- 1) both must have min 1 d.
- 2) Align from RIGHT to LEFT.

then iterating over each \rightarrow both same
the dimension should be
~~either 1 or DNE.~~

BUT.

if $P = N.\text{float}()$ $\rightarrow [27 \times 27]$.

$$P = P / P.\text{sum}(1) \rightarrow \frac{[27 \times 27]}{[27]} \rightarrow \frac{27 \cdot 27}{1 \cdot 27}$$

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

$$P.\text{Sum}() = [3, 7, 11]$$

$$\therefore P/P.\text{sum} = \frac{3 \times 2}{1 \times 3}$$

not allowed.

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 5 \\ 7 & 8 & 9 \end{bmatrix} \rightarrow \text{P.sum(1)} [6, 15, 24].$$

P / P.sum

$$= \frac{3 \times 3}{- \times 3} = \frac{3 \times 3}{1 \times 3} = \frac{3 \times 3}{3 \times 3} \rightarrow \text{squeezing}$$

~~copying~~

$$[6, 15, 24] \rightarrow \begin{bmatrix} 6 & 15 & 24 \\ 6 & 15 & 24 \\ 6 & 15 & 24 \end{bmatrix}$$

to stretch vector

now division occurs and gives wrong ans.

Correct way:

$$\text{P.sum(1, keepdim=True)} = [[6], [15], [24]].$$

stretch \rightarrow $\begin{bmatrix} [6, 6, 6], \\ [15, 15, 15], \\ [24, 24, 24] \end{bmatrix}$

now div

its v easy to add bugs coz.

we create dimensions as we go from R \rightarrow L.

$$P_1 = P_{\text{sum}}(1, R - \text{d.} = \text{True}) \Rightarrow P = P/P_{\dots}$$

↑
either

↑
creates new
object

Evaluate Quality of Model:

So to evaluate the quality of model, we multiply the prob of each bigram in the resp. names. take the -ve log of it and average of that.

$$\text{emana} = \begin{array}{l} \cdot e \rightarrow 0.04 \\ \cdot m \rightarrow 0.03 \\ \cdot m \rightarrow 0.02 \\ a. \rightarrow 0.38 \\ \cdot \rightarrow 0.19 \end{array}$$

\log -3.0 -3.2 -3.6 -0.9 -1.6	$-\log$ 3.0 3.2 3.6 0.9 1.6
\downarrow VV likely	

$$0.04 \rightarrow \frac{1}{27} = \text{avg.}$$

fog

avg =

Sum of Log s = log-likelihood.

we use

lower it is, better the model

avg neg log likelihood.

for w_k in "Hello world" string:

$chs = [':'] + \text{list}(w) + ['w']$

for ch_1, ch_2 :

$i \times 1, i \times 2 = \dots \text{sto};$

$\log\text{-prob} = \text{torch.log}(P(x[i_1], [i_2]))$

$\log\text{-lik} = \log\text{-prob}$

$n += 1$

$\text{print}(\text{prob})$

$nll = -\log\text{-lik}$

$nll/n = \text{avg}$

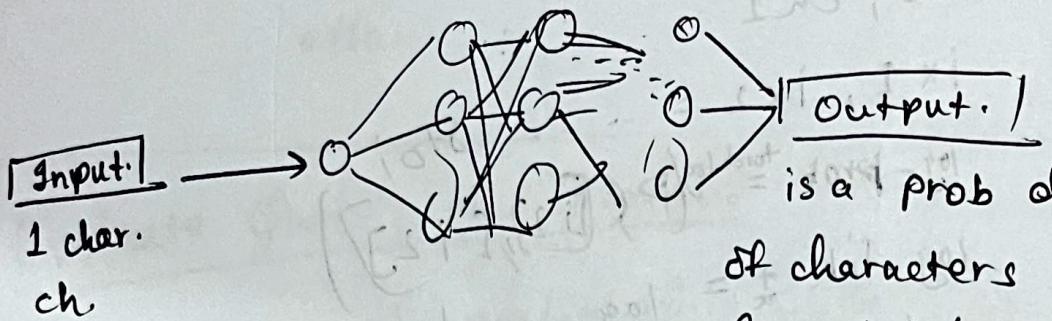
One flaw is that if the bigram never existed in training set, our model returns

$nll/n = +\infty$ for that, which we do not like

\therefore use $P = \cancel{N+1} (N+1).float()$

Now find total. nll/n for entire set.
quality

Doing the same with Neural Nets:



is a prob distribution of characters that can follow 'ch'. Now using info abt the actual bigram we calc loss func and minimise it

~~here we take in~~

e.g: emma

$b_1 = [\cdot, e, em, mm, ma, a.]$

$x_s = [\cdot, e, m, m, a]$

$y_s = [e, m, m, ma, \cdot]$

for \hat{o} in names:

$che = 2$

for ch1, ch2:

i_{x1}, i_{x2}

$x_s \cdot \text{appear}(i_{x1})$
 y_s

$x_s = \text{torch.tensor}(x_s)$

If it is best not to give single integer input to a neural net.

~~string~~

Note: `torch.tensor`

dt
 $dtpe$
= that
 dt list

`vs`
`torch.Tensor`

dt
 $dtpe = float32$

So we use one-hot funcs.

to convert integer index input to
27 - tensor zero with i th index = 1.

$$xs = [0, 5, 13, 13, 1] \rightarrow \begin{bmatrix} [1, 0, 0, \dots], \\ [0, 0, 0, 0, 1, 0, \dots], \\ [0, \dots, 1, 0, \dots] \end{bmatrix}$$

import torch.nn.functional as F

xenc = F.one_hot(xs, num_classes=27)

xenc.shape \rightarrow ([5, 27])

Since we had used .tensor, xs.dtype = int64

\therefore xenc \rightarrow int64 but in NN we want float

\therefore xenc = ".float()

$$5 \begin{bmatrix} \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \end{bmatrix}_{27 \times 1}$$

1 Neuron: takes word as input as xenc.

$W = \text{torch.rand}((27, 1)) \rightarrow$ uses normal distribution and returning (27×1) tensor of floats.

$$\begin{bmatrix} [1.232] \\ [3.148] \\ [-1.2] \\ \vdots \\ [0.6] \end{bmatrix}_n$$

input.

(5x27)

W
(27x1)

1 Neuron

]

.

.

]

$$\therefore \text{out} = \text{xenc} @ W \rightarrow [5 \times 1]$$

out gets 1 output for
each input character.

[[]
[]
[]
[]]

but for each input char, we want
a prob distribution of all pos characters
to follow it.

∴ We use 27 neurons.

∴ $W = \text{torch}.randn((27, 27))$

$\text{xenc} @ W$

(5x27)

$\times (27, 27) \rightarrow (5 \times 27) \rightarrow$

$(xenc @ W)[5][\frac{1}{27}]$
prob that 'm' is
followed by 'a' [$\frac{1}{27}$]

torch.exp

[[]
[]
[]
[]
[]]

but $x_{enc} @ W \rightarrow$ has +ve, -ve nos.

$$\therefore \text{logits} = x_{enc} @ W \quad (\sim \text{log}\mathcal{S})$$

$$\text{counts} \equiv \text{logits}. \exp() \quad (\sim N)$$

$$\text{probs} = \frac{\text{counts}}{\text{counts.sum(1, keepdims=True)}}$$

\therefore now we have a prob distribution for each input char.

$$\text{probs.shape} = (5 \times 27)$$

$$\text{probs[0].sum()} = 1.0$$

$$\text{nlls} = \text{torch.zeros}(5)$$

for i in (5):

$$x = xs[i]; y = ys[i];$$

"input cha" x

"prob dis" $\text{prob}[i][\cancel{x}]$

"expect for y" $\text{prob}[i,y].item() \rightarrow = p$.

$$\therefore \text{logp} = \text{torch.log}(p)$$

$$\text{nll} = -\text{logp}.$$

$$\text{nlls}[i] = \text{nll}.$$

$$\text{avg(logp)} = \text{nlls.mean().item().}$$

we added I to λ in the ~~statistical~~
model to smoothen things out. Adding larger
nos to N will smoothen it.

Similarly in NNs, W ke values, if we bring them
closer to zero. then it smoothen
cos $\text{logits} = \text{xenc} @ W \sim 0$
 $\therefore \cancel{\text{prob}} \sim 1$.
 $\therefore \text{prob} \sim \text{smoothen}$

↳ called regularization

$$\text{loss} = -\text{probs}[-].(\text{log.mean}) + 0.1 * (\text{w}^{**2})_{\text{mean}}$$

\therefore gradient descent tries to decrease
 w^{**2} i.e. bring value of
 w closer to 0.

Karpfthy lec-2

-/-/-

X-encoding:

$$\text{Sx2, T1, S1, 13, 5}$$

(reg)

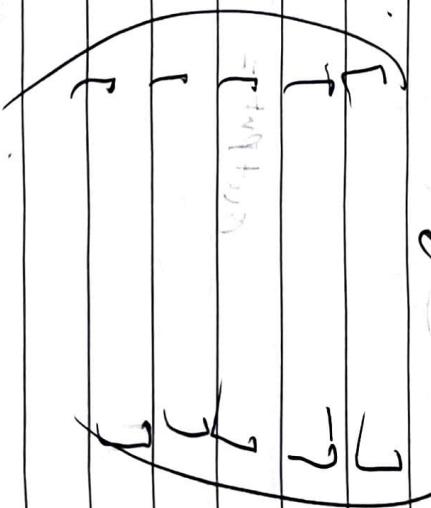
$$\begin{matrix} \text{emma} \\ = .e \\ \text{mem} \\ \text{m} \\ \text{ma} \\ \text{Fa} \end{matrix} \rightarrow \begin{matrix} \text{Sx2, T1, S1, 13, 5} \\ \text{Sx2, T1, S1, 13, 5} \end{matrix}$$

$$\text{Sx24} \rightarrow \text{27x1}$$

$$\text{Sx24} \rightarrow \text{27x27} \rightarrow \text{8x27}$$

$$\text{Sx24} \rightarrow \text{27x27} \rightarrow \text{8x27}$$

b) prob for 's'
next letter for each of the inputs.



$$27 \times 27 \rightarrow 27 \times 27 \rightarrow 8 \times 27$$

$$\left[\begin{matrix} \text{t} \\ \text{t} \\ \text{t} \\ \text{t} \end{matrix} \right] \rightarrow \left[\begin{matrix} \text{t} \\ \text{t} \\ \text{t} \\ \text{t} \end{matrix} \right] \rightarrow \left[\begin{matrix} \text{t} \\ \text{t} \\ \text{t} \\ \text{t} \end{matrix} \right]$$

$$\left[\begin{matrix} \text{t} \\ \text{t} \\ \text{t} \\ \text{t} \end{matrix} \right] \rightarrow \left[\begin{matrix} \text{t} \\ \text{t} \\ \text{t} \\ \text{t} \end{matrix} \right] \rightarrow \left[\begin{matrix} \text{t} \\ \text{t} \\ \text{t} \\ \text{t} \end{matrix} \right]$$

each neuron gives out prob of its letter
coming next

emma.

mn \rightarrow I
ma \rightarrow I

[0.19] 0.04

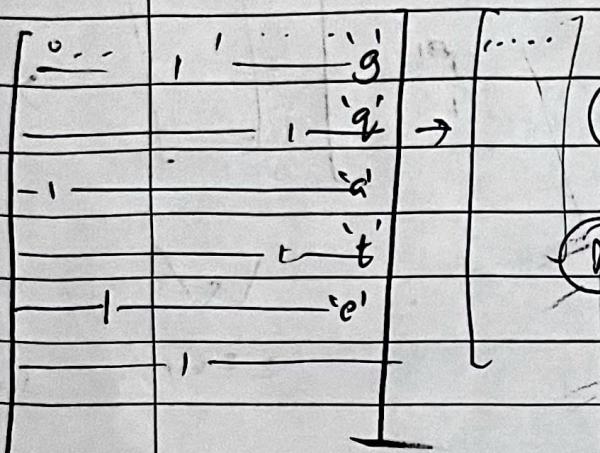
-0.5
0.451
prob of

each of those letters
being both

x. w. b.

$t \times 27$

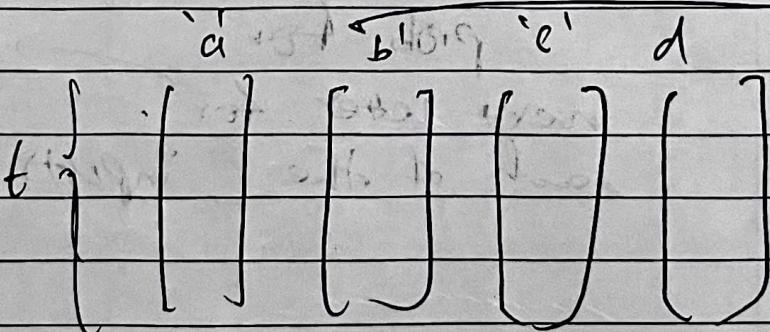
[] + [] \rightarrow prob of 'a' t []



prob of each of those
letters being
followed by an 'a'

Ramay

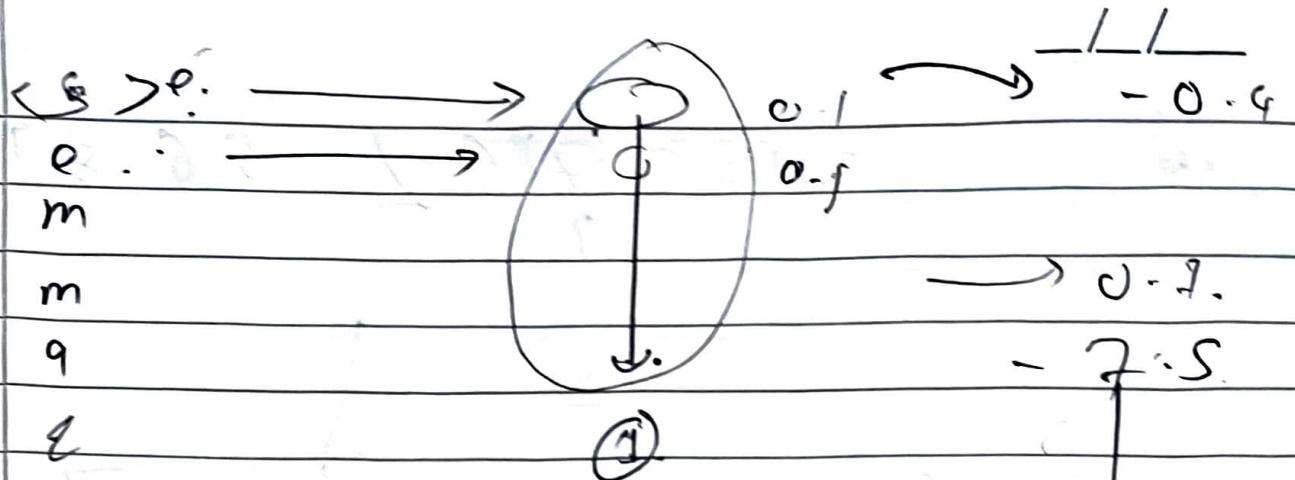
out put =



\therefore out [3, 13]

firing rate of 13th neuron by the 3 input

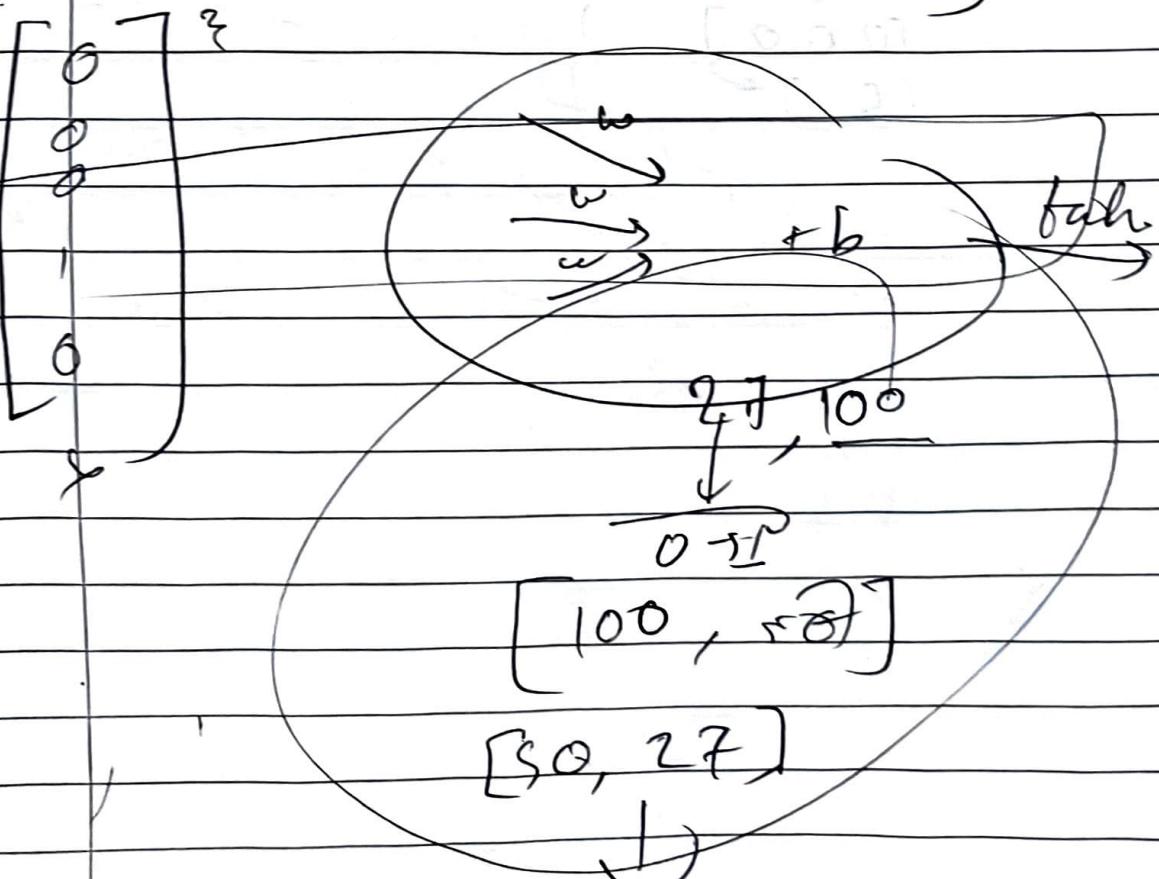
\rightarrow prob of 13th letter following the
3rd input.



$$\text{loss} = \frac{1}{n} \sum [y_i - \hat{y}_i]^2$$

$$w.\text{adata} = w.\text{grad}^{14} 0.01$$

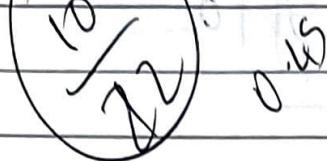
(0 00 . . . + 0 00 .)



$\underline{\underline{11}}$

$x \rightarrow 32 [e^-, e, -em, \dots]$

$32 [e, m, m, \dots]$



$SC[x] =$

$em + x$

$\frac{d}{dx}$

$\frac{d}{dx}$

$\frac{d}{dx}$

e^x

0^x

1^x

2^x

3^x

4^x

5^x

6^x

7^x

8^x

9^x

10^x

11^x

12^x

13^x

14^x

15^x

16^x

17^x

18^x

19^x

20^x

21^x

22^x

23^x

24^x

25^x

26^x

27^x

28^x

29^x

30^x

31^x

32^x

33^x

34^x

35^x

36^x

37^x

38^x

39^x

40^x

41^x

42^x

43^x

44^x

45^x

46^x

47^x

48^x

49^x

50^x

51^x

52^x

53^x

54^x

55^x

56^x

57^x

58^x

59^x

60^x

61^x

62^x

63^x

64^x

65^x

66^x

67^x

68^x

69^x

70^x

71^x

72^x

73^x

74^x

75^x

76^x

77^x

78^x

79^x

80^x

81^x

82^x

83^x

84^x

85^x

86^x

87^x

88^x

89^x

90^x

91^x

92^x

93^x

94^x

95^x

96^x

97^x

98^x

99^x

100^x

101^x

102^x

103^x

104^x

105^x

106^x

107^x

108^x

109^x

110^x

111^x

112^x

113^x

114^x

115^x

116^x

117^x

118^x

119^x

120^x

121^x

122^x

123^x

124^x

125^x

126^x

127^x

128^x

129^x

130^x

131^x

132^x

133^x

134^x

135^x

136^x

137^x

138^x

139^x

140^x

141^x

142^x

143^x

144^x

145^x

146^x

147^x

148^x

149^x

150^x

151^x

152^x

153^x

154^x

155^x

156^x

157^x

158^x

159^x

160^x

161^x

162^x

163^x

164^x

165^x

166^x

167^x

168^x

169^x

170^x

171^x

172^x

173^x

174^x

175^x

176^x

177^x

178^x

179^x

180^x

181^x

182^x

183^x

184^x

185^x

186^x

187^x

188^x

189^x

190^x

191^x

192^x

193^x

194^x

195^x

196^x

197^x

198^x

199^x

200^x

201^x

202^x

203^x

204^x

205^x

206^x

207^x

208^x

209^x

210^x

211^x

212^x

213^x

214^x

215^x

216^x

217^x

218^x

219^x

220^x

221^x

222^x

223^x

224^x

225^x

226^x

227^x

228^x

229^x

230^x

231^x

232^x

233^x

234^x

235^x

236^x

237^x

238^x

239^x

240^x

241^x

242^x

243^x

244^x

245^x

246^x

247^x

248^x

24