→ Makemore - 3

Problems in the net:

① large initial loss:

fix: $W_2 * 0.01$, $b_2 = 0$ } be

↑ last layer.

→ the logits that come out of the last layer

eg: [0.1, 2.0, 7.2, 3.2]

softmax: [$10^{0.1}$, $10^{2.0}$, $10^{7.2}$, $10^{3.2}$]

prob: [0, 0, 1, 0]

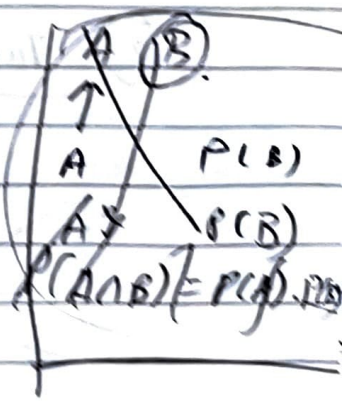
∴ we need logits closer to 0.

∴ $W_2 \rightarrow$ near 0

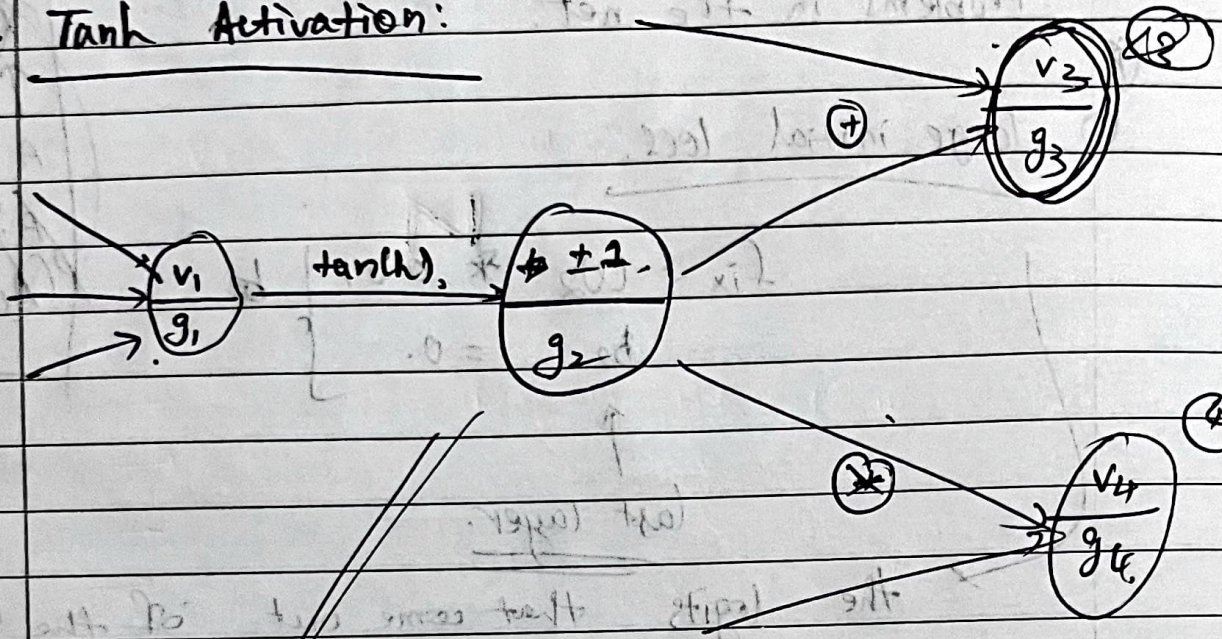
$b_2 \rightarrow 0$

∴ prob → better and more neutral.

W_2 cannot be 0 → symmetry problem



(2) Tanh Activation:



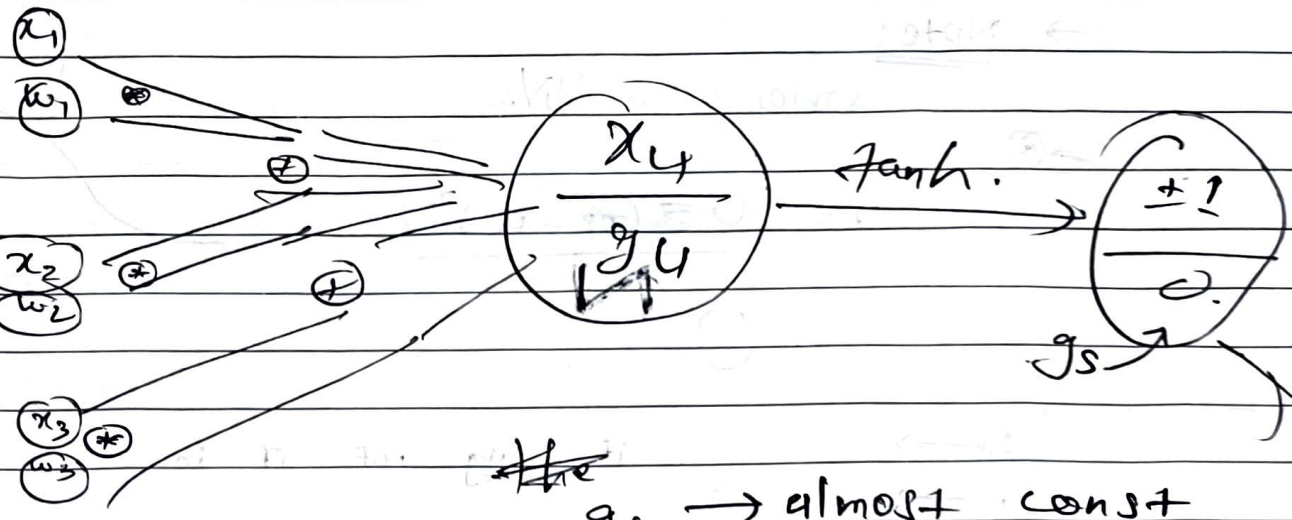
∴ Backprop thru (3) :- $\frac{d \tanh(x)}{dx}$

$g_2 \mathrel{+}= g_3 \cdot \frac{dL}{d(\tanh)} = \frac{dL}{d(\tanh x)} \cdot \frac{d(\tanh x)}{dx}$

this is not a weight,
∴ the data only depends on prev node
∴ we need

Whenever the gradient goes back thru this node, it only adds g_3.

∴ the only way this ever changes if x changes via other activations then $\tanh(x)$ changes ...



~~the~~ $g_4 \rightarrow$ almost const as longs as $g_5 \sim 0$

∴ the only way for it to come out of saturation is for x_i, w_i ($i = 0, 1, 2, 3$) to change greatly via other backprop thus changing $x_4 \Longrightarrow$ bringing $\tanh(x_4)$ closer to 0 $\rightarrow$ gradient $g_5 \neq 0$ (post activation) and the neuron stops being dead / saturation.

Fix:

~~We scale~~ scale down w_1 (hidden layer) and b_1,

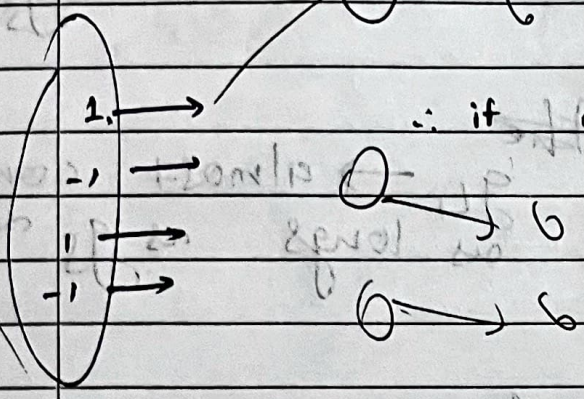
$w_2 * = 0.2$

$b_1 \sim$ also tone it down.

→ Note:

variance in NNs

$$V = \frac{\sum (x_i - u)^2}{N}$$



∴ if avg not 0 of in

Variance in a Dataset:

We use standardized inputs with variance of 1 to make all types of inputs of same importance

$$\text{input} = \frac{\text{input} - \bar{x}}{\sqrt{\frac{\sum_i (x_i - \bar{x})^2}{N}}}$$

solves the issue where one type of input has values of much greater magnitude than others, hence dominating the loss fn

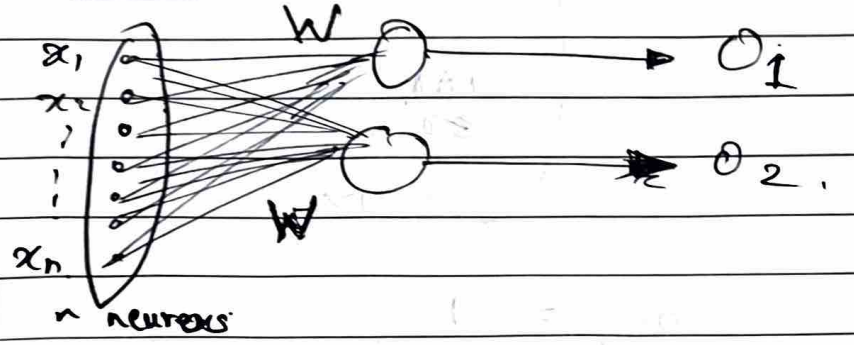
~~The issue is for~~

Now we come to conservation of variance:

variance is also a good measure of signal strength.

for eg:

layer 1.



var = 'k'.

grad. $O_i^0 = \sum x_i w_i^0$.

$\therefore \text{var}(O_i) = \sum_1^M \left(\sum x_i w_i - \frac{\sum\sum x_i w_i}{n} \right)^2$

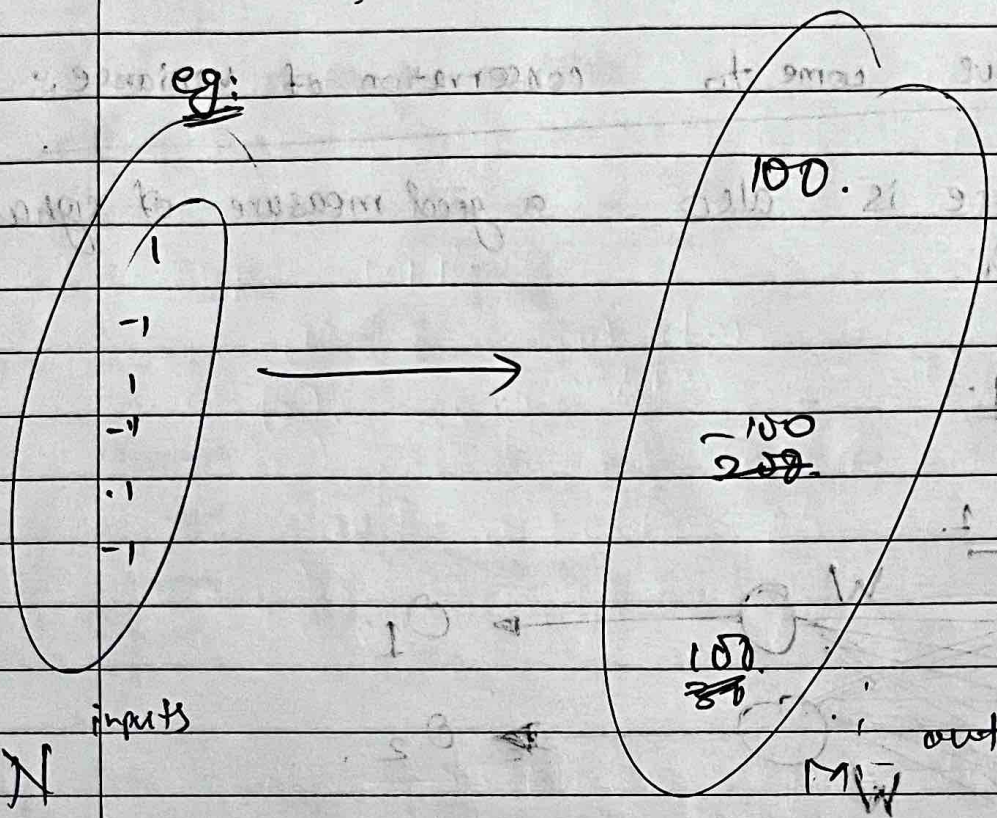
$\approx \frac{\sum (x_i - \frac{\sum x_i}{n})^2}{N}$.

Basically it got scaled up.

$E(w^2)$

$\hookrightarrow \frac{\sum x^2}{n} - \left(\frac{x}{n}\right)^2 = \text{var} \sum x^2$

eg:



$\therefore \text{var}_{og} = 1$

$\text{var}_{new} = 100^2 \times M$

M

$\sim 10000.$

∴ It gets scaled up by M^2

This all depends on the weights w.

We want w to be such that variance is maintained so that after multiple layers the strength neither explodes nor vanishes.

Thm

$$Var(y) = \left(\sum w_i^2\right) Var(x)$$

assuming all inputs have var 1.

$$\therefore \frac{var(y)}{var(x)} = \sum w_i^2$$

$$= \text{var}$$

if i mul all value by k. $\therefore \sigma \Longrightarrow k$ times.

$$\sigma = \sqrt{var} = \sqrt{s^2_{new}}$$

$$= \sqrt{\frac{k^2 x^2}{n} - k^2 \bar{x}^2} = k\sqrt{s^2}$$

__/__/____

Makemore lec-3 notes:

① Use torch.no_grad() when u don't expect to do a backprop. or @torch.no_grad() decorator.

② Setup: MLP.

1. 3 input char
2. $3 \times 10^{\text{dim/char}} = 30$. ~~vector~~ dim vector.
3. passed into hidden layer 200 neurons and activation f^n [$W_1, b_1, \tanh$]
4. final output layers — 27 layers. produces raw scores / 27 logits.
5. ⟶ softmax : prob distr.
6. loss = nll of prob.

③ Poor initial loss.

exp: $\rightarrow -\log\left(\frac{1}{27}\right) \sim 3.29$.

∴ Our logits were very overconfident at start:
↳ make these close to 0.

$\therefore$ $b_2 = 0$.
$W_2 * = 0.01$

$W_2 \neq 0$ ⟶ Symmetry prob

∴ we no longer spend a lot of time squashing weights.

④ Activation saturation:

tanh is linear and resp b/w (-2.5, 2.5) otherwise loses off info.

$$H = C(@W_1 + b_1$$

*0.1 ~0

done. thes down.

⑤ torch.randn() → creates a distribution with

mean = 0.
$\sigma = 1$.
$v = 0.1$.

$v = \sigma^2 = \frac{\Sigma(x - \bar{x})^2}{n}$.

$\therefore \frac{\Sigma x^2}{n} = 1$. for these.

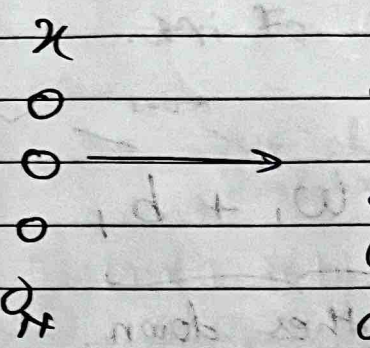
Now, as we do the forward pass the signal strength changes due to the weights and number of input to each neuron layer.

If the strength inc/dec too much ⇒ activation gets fucked up

We use variance as a measure of strength.

initial data : var = 1.

Now,



$$var(y) = \frac{\Sigma (y - \bar{y})^2}{n}$$

if we see a single neuron.

Single neuron:

$$var(out) = var\left(\sum w_i x_i + b\right)$$

$$= \sum var(w_i x_i)$$

$$= \sum w_i^2 \times var(x_i)$$

$$= \sum w_i^2 \quad (var(x_i) = 1)$$

if we look at all out neurons

$$var(y) = E(y^2) - (E(y))^2$$

$$= E(y^2) \quad \left(\Sigma E(w_i) E(x_i)\right)$$

$$= E\left(\left(\sum_{1}^{fan\text{-}in} w_i x_i\right)^2\right)$$

$$E\left(\sum w_i x_i\right) = \sum E(w_i x_i)$$

__/__/____

$\therefore$ ~~E(~~ var(y)

$= E(y^2)$

$= E\left(\sum_{i=0}^{n} \sum_{j=0}^{n} E(w_i)^2 E(x_i)^2 E(w_j) E(x_j) \right)$

$= \sum_{i=0}^{n} \sum_{j=0}^{n} E(w_i^2 x_i^2)$

$= \sum_{i=0}^{n} \sum_{j=0}^{n} E(w_i^2) E(x_i^2)$

$= E\left(\left(\sum_{i=0}^{n} E(w_i) E(x_i) \right)^2 \right)$

$= E\left(\sum_{i=1}^{n} (w_i x_i)^2 + \sum_{\substack{i,j=1 \\ i \neq j}}^{n} w_i x_i w_j x_j \right)$

$= \sum_{i=1}^{n} E(w_i x_i)^2 + \sum_{\substack{i,j=1 \\ i \neq j}}^{n} E(w_i) E(w_j) E(x_i) E(x_j)$ → 0.

$= \sum_{i=1}^{n} E(w_i^2) E(x_i^2)$

$\Rightarrow$ var(y) = $E(y^2) = n_{in} \, var(w) \, var(x$

__/__/____

$\therefore \text{var}(y) = \text{var}(x)$ iff.

$$\sigma_w^2 = \text{var}(w) = \frac{1}{\text{no. of neurons (input)}}$$

$$\therefore \sigma = \frac{1}{\sqrt{\text{fan\_in}}}$$

This accounts only for linear ops.

∴ for activation fn's aswell,

$$\sigma_w = \frac{\text{gain}}{\sqrt{\text{fan\_in}}}$$

→ 5/3 for tanh

→ $\sqrt{2}$ for Relu

__/__/__

⑥ Batch Normalization:

Internal Coreriate Shift:

to normalize a layer.

$$\mu = \frac{\Sigma x_i}{m} \quad \frac{1}{m} \Sigma x_i$$

$$\sigma^2 = \frac{1}{m} \Sigma (x_i - \mu)^2$$

$$\therefore \hat{x_i} = \frac{x_i - \mu}{\sqrt{\sigma^2 + \varepsilon}}$$

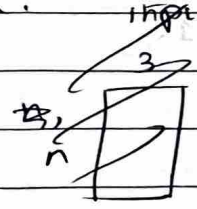
$$\rightarrow \sigma^2_{x_i} = \frac{1}{m} \sum \frac{x_i^2 + \mu^2 - 2x_i\mu}{\sigma^2} = \frac{(x_i - \mu)^2}{\sigma^2} = 1.$$

$$y_i = \gamma \hat{x_i} + \beta$$

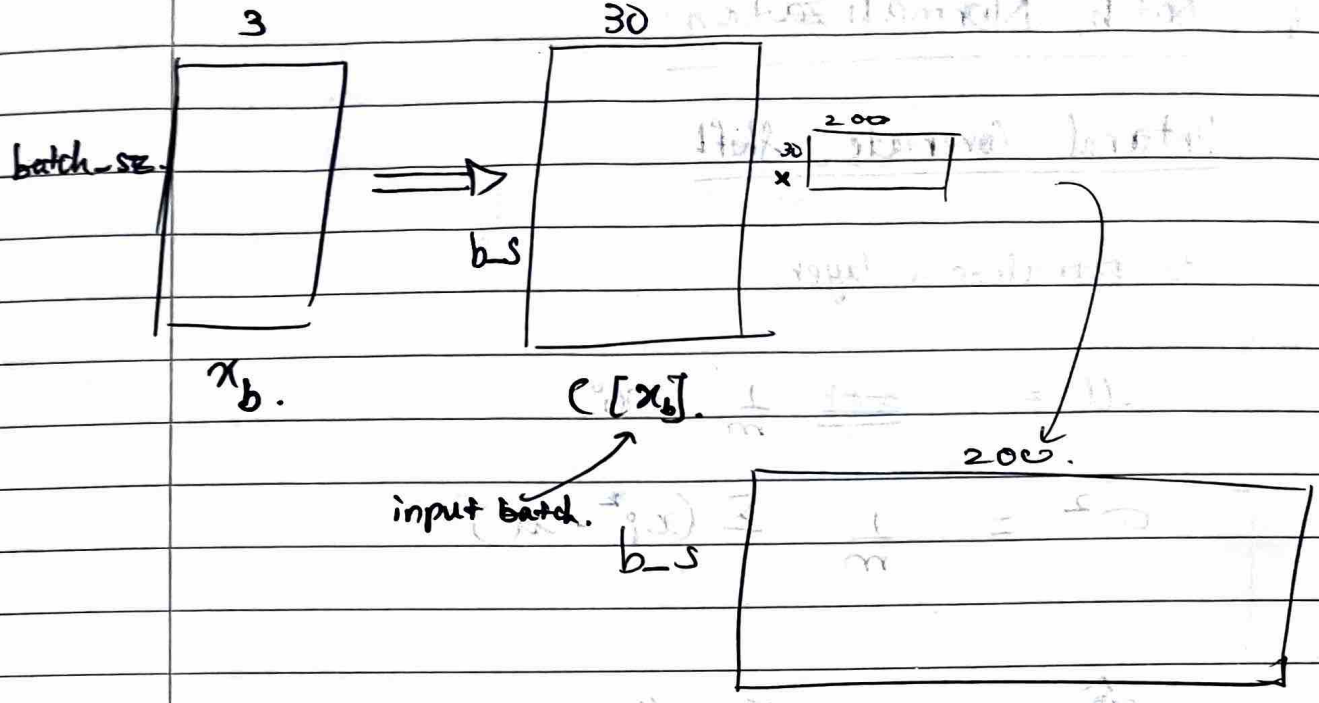
Batch normalization works on the whole batch.

Basically if we have n inputs ⟹ in 1 batch.

∴



context Len x emb_dim.

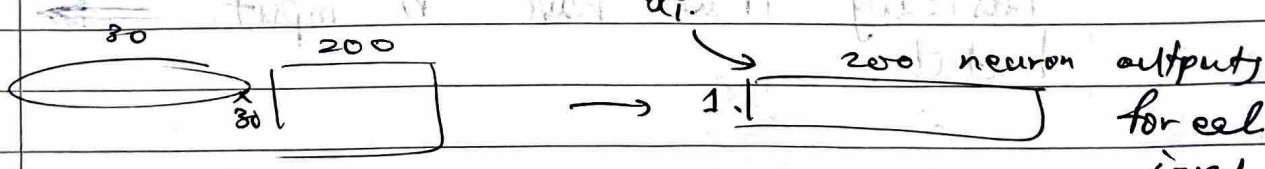


Now we got 200 neuron outputs.
~~wt~~ ~~bef~~ pre-activation.

now we apply normalization on these 32. (not 200). The "batch" is normalized.

$$\therefore \quad \bar{u} = \frac{\sum x_i}{32} \qquad \sigma^2 = \frac{\sum (x_i - u)^2}{32}$$

bs.



We adjust x_i's → unit variance, $\sigma^2 = 1$.

$$\hat{x_i} = \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon_0}}$$

ϵ_0 → small num in case $\sigma^2 = 0$

further scaling:

$$z = \gamma \hat{x} + \beta.$$

γ ← scale variance. β ← scale mean.

Note:

① γ, β are 1×200 vectors?!!

∴ ~~They will~~

$x\_hat = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \\ j & k & l \end{pmatrix}$ ∴ $\gamma = (\gamma_1 \ \gamma_2 \ \gamma_3)$

$\beta = (\beta_1 \ \beta_2 \ \beta_3)$

$$\therefore x = \begin{pmatrix} a\gamma_1 + \beta_1 & b\gamma_2 + \beta_2 & \cdots \\ d\gamma_1 + \beta_1 & e\gamma_2 + \beta_2 & \cdots \\ g\gamma_1 + \beta_1 & \vdots & \cdots \\ j\gamma_1 + \beta_1 & \vdots & \cdots \end{pmatrix}$$

② we don't use a bias anymore in the preactivation as is gets cancelled out anyways ~~eff~~ on $-\bar{u}$ on the $\hat{x_i}$ creation

③ So now for the linear part, the signal strength would maintain itself but tanh squishes ~~the~~ inputs so we gotta multiply the outs - by a factor $\beta_{gain} = 5/3$ to maintain strength

⟹ saturation ~ 5, -6 %.
std.
~~variance~~ ~ 0.65.

(in case of properly linear gain ⟹ 1.)

Batch norm takes care of most things.

Summary:

① Batch Norm

② Diagnostic tools:
- ① Activation Distr (past-act) per layer
- ② Gradient Distr
- ③ update to data (over time)
  - ↳ (upd. std / data.std())

_ / _ / _

Backprop Ninja :

$L = \text{loss} = -\frac{1}{n}\,[\,lp[0][Yb[0]] + lp[1][Yb[1]] + \cdots$

$= -\frac{1}{n}\sum_{i=0}^{n-1} \text{logprob}[i][Yb[i]]$.

$= -\frac{1}{n}\,[\,lp$

---

let's do Backdrop thru matrix mult.

1) consider : $Z = X @ W$

$(m \times n)$ ⟶ $(n \times p)$

m rows (1 batch) ⟶ n feaolters

let $G = \frac{\partial L}{\partial Z}$

$z(i,j) = \sum_{k=0}^{K} (X[i,k] * W[k,j])$ — each column is a weight vector for one output.

for a particular k.

$z(i,j) = X[i,k] * W[k,j]$

$\frac{\partial L}{\partial X}$

$\frac{\partial L}{\partial W}$

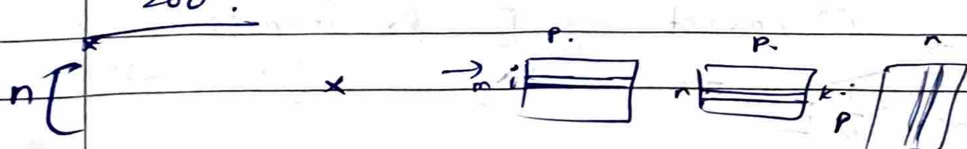
$\frac{\partial L}{\partial X_{ik}} = \sum_{j=1}^{p} \frac{\partial L}{\partial Z_{ij}} \frac{\partial Z_{ij}}{\partial X_{ik}}$.

$= \sum_{j=1}^{p} G_{ij} \times W_{kj}$

$= (G W^T)_{ik}$.

$\therefore \frac{\partial L}{\partial X} = G W^T$.

200.



__/__/____

∴ if $z = x @ w$.

$\therefore \frac{dz}{dx} =$

$\frac{dL}{dx} = \frac{dL}{dz} \cdot w^T$

$\frac{dL}{dw} = x^T \frac{dL}{dz}$.

$\frac{dL}{d\text{logprobs}}$

$L = -\frac{1}{n} \sum_{i=0}^{n-1} \text{logprobs}[i][Y_b[i]]$.

$= \frac{-1}{n} \sum_{i=0}^{n-1} S_i$.

$\therefore \frac{dL}{dS_i} = \frac{-1}{n}$ $\forall i \in$

matrix where

$\therefore \frac{dL}{dp} = \frac{dL}{dS_i} \cdot \frac{dS_i}{dp}$.

$E_{i,c} = 1$ if $c = Y_b[i]$ else 0.

$S_i = \text{logprobs} \cdot E \cdot Y$.

Y.

emb

$E =$

$\therefore S_i =$ the i^{th} row and Y_b^{th} column of logprobs

$\frac{dL}{d\text{logprobs}} = \frac{-1}{n} E$

i^{th} row of E $n \times bs$

logprobs $bs \times$ vocabsize

__/__/____

$\therefore$ ~~$\frac{\partial (x @ w)}{\partial x} = w^T$~~ $\quad \frac{\partial (x @ w)}{\partial w} = w \cdot x^T$

$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial (x @ w)} w^T \quad \frac{\partial L}{\partial w} = x^T \frac{\partial L}{\partial (x @ w)}$

~~$\frac{\partial L}{\partial \log}$~~

$\frac{\partial L}{\partial \text{logprob}} =$

$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial x}$ — $G = \frac{\partial L}{\partial z}$

$G = \frac{dL}{dz}$ $\quad z = x @ w$

$\partial L = G \partial z \quad z =$

$= G (\partial x @ w + x @ \partial w)$

$w + \partial w + x @ \partial w$

$\frac{\partial z}{\partial x} = w^T \quad \frac{\partial L}{\partial w} = x^T G$

$$Z = X @ W$$

$m \times n \quad m \times p \quad p \times n$

$$\left(\frac{\partial L}{\partial X}\right) = \left(\frac{\partial L}{\partial Z} \cdot \frac{\partial Z}{\partial X}\right)$$

$$= \sum_i \sum_j \left(\frac{\partial L}{\partial Z}\right)_{ij} \frac{\partial Z_{ij}}{\partial X}$$

$$\frac{\partial L}{\partial X} = \begin{bmatrix} a & b & c & d \\ e & f & g & h \end{bmatrix}$$

$$\frac{\partial L}{\partial [ae+bf+cg+hi]} \begin{bmatrix} \frac{\partial L}{\partial a} & \frac{\partial L}{\partial b} & \frac{\partial L}{\partial c} & \frac{\partial L}{\partial d} \\ \frac{\partial L}{\partial e} & \frac{\partial L}{\partial f} & \frac{\partial L}{\partial g} & \frac{\partial L}{\partial h} \end{bmatrix}$$

$$\frac{\partial L}{\partial (ae+bf+cg+hi)} \cdot \frac{\partial (ae+bf+cg+hi)}{\partial a} = \frac{\partial L}{\partial a}$$

$$\frac{\partial L}{\partial (ae+bf+cg+hi)} \cdot \frac{\partial (ae+bf+cg+hi)}{\partial ae} \cdot \frac{\partial ae}{\partial a}$$

$$\frac{\partial L}{\partial x} = G(W^T)$$

$$z = x @ w$$

$$x = \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \end{bmatrix} \quad w = \begin{bmatrix} \alpha & \beta & \gamma \\ \delta & \epsilon & \zeta \\ \eta & \theta & \iota \\ \kappa & \lambda & \mu \end{bmatrix}$$

$$z = \begin{bmatrix} a\alpha + b\delta + c\eta + d\kappa & \cdots \\ \cdots & \cdots \end{bmatrix}$$

$$\frac{\partial L}{\partial a} = \frac{\partial L}{\partial z} \cdot \frac{\partial z}{\partial a}$$

$$\frac{\partial a}{\partial L} = \frac{\partial (a\alpha + b\beta + c\gamma + d\zeta)}{\partial L \cdot \alpha}$$

$$= \frac{da}{dL} \frac{\partial L}{\partial z} + d(\alpha)$$

$$x @ w$$

$$\left(\frac{\partial L}{\partial x}\right)_{ij} = (G)_i @ w^T$$

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z} \cdot \frac{\partial z}{\partial x}$$

$$= G \cdot$$

__/__/____

$$Z_{ij} = \left[\sum_{k=1}^{n} x_{ik} y_{kj} \right]$$

$$\left(\frac{\partial L}{\partial Z}\right)_{ij} = \left(\frac{\partial L}{\partial \left(\sum_{k=1} x_{ik} y_{kj}\right)}\right)$$

$$G_{ij} = \frac{\partial L}{\partial (\Sigma x_{ik} y_{kj})}$$

$$\left(\frac{\partial L}{\partial x}\right)_{ij} = \left(\left(\frac{\partial L}{\partial z}\right) \cdot \left(\frac{\partial z}{\partial x}\right) \right)$$

$$\frac{\partial L}{\partial x} = G\, W^T$$

$$G = \frac{\partial L}{\partial z} \qquad z = x \circledast w$$

$$G = \begin{bmatrix} \frac{\partial L}{\partial (\alpha a + \beta b + c y + d \lambda)} \\ \vdots \end{bmatrix}_{4\times 1} \circledast \begin{bmatrix} a & b & c & d \\ \beta & y & \lambda & \cdots \end{bmatrix}_{1\times 4} \quad \begin{bmatrix} \alpha \\ \beta \\ \gamma \\ \vdots \end{bmatrix}_{4\times 1}$$

$$z = \begin{pmatrix} a\alpha + \beta b + c\gamma + d \\ \vdots \end{pmatrix}$$

$$= \left[\frac{\partial L_j}{\partial(a\alpha + \dots)} \alpha \right], \frac{\partial L}{} \; b$$

$$\frac{\partial L}{\partial a} = \frac{\partial L}{\partial(a\alpha + b\beta + \dots)} \alpha$$

$$\frac{\partial a}{\partial L} = \frac{\partial(a\alpha + b\beta + \dots)\alpha}{\partial L \; \alpha}$$

$$= \frac{\partial(a\alpha)\alpha}{\partial L} + \frac{\partial(b\beta)\alpha}{\partial L} + \dots$$

$$= \frac{\partial a}{\partial L} \frac{\partial(a\alpha)}{\partial L(\alpha)}$$

$$\frac{x^4}{x^2}$$

$$\frac{\partial(x^4)}{\partial(x^2)}$$

$$\frac{\partial L}{\partial a} = \frac{\partial L}{\partial a} \frac{\partial a}{\partial(a\alpha + b\beta + c\gamma + \dots)} \alpha$$

$$= \frac{\partial L}{\partial a} \cdot \frac{\partial(z - b\beta + c\gamma - \dots)\alpha}{\partial z}$$

$Z = x \cdot w$.

$G = \frac{dL}{dZ}$.

find: $\frac{dL}{dw}$.

$$\therefore \left(\frac{dL}{dw}\right)_{pq} = \sum_{ij} \left(\frac{dL}{dZ}\right)_{ij} \cdot \sum_{i,j} \frac{\partial Z_{ij}}{\partial w_{pq}}$$

$$= \sum G_{ij} \sum \frac{\partial (x_{pk} \cdot w_{kq})}{\partial w_{pq}}$$

$$= \sum G \cdot G_{kq} \cdot x_{kp}$$

$$= \sum x^T_{pk} \cdot G_{kq}$$

$$= x^T G_{pq}$$

$$\frac{dL}{dw}$$

Direct Calculation of Loss:

logits = tan(...)

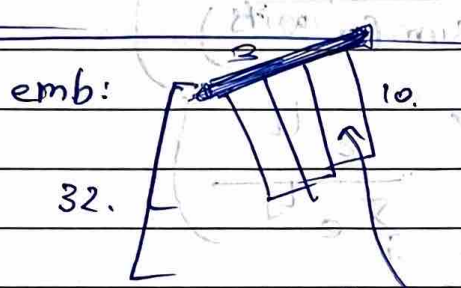
$$\therefore \log\left(\frac{\exp(\text{logits})}{\sum \exp(\text{logits})}\right)$$

$$\therefore \frac{\partial \text{loss}}{\partial \text{Logits}_i} = \frac{e^{l_j}}{e^{l_i}} \cdot \left(e^{l_i} \cdot \left(\frac{-1}{(\sum e^{l_j})^2}\right) \cdot e^{(i)}\right)$$

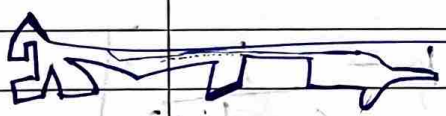
$= ($

logits = h2 @ W2 + b2
logit_maxes = logits.max(1, keepdim=True).
norm_logits = logits - logit_maxes.

$\therefore \frac{d(\text{norm})}{d\text{logits}} = 1$



we wanna add this vector to its resp grad emb row in C.



Direct loss Calculation:

We want derivative of loss wrt a particular row vector of the logits $\rightarrow L_i$

$$\therefore \frac{\partial Loss}{\partial L_i}$$

$$loss = -\log\left(\frac{e^{logits}}{sum(e^{logits})}\right)$$

$$loss_i = -\log\left(\frac{e^{L_i}}{\sum_j e^{L_j}}\right)$$

$$\therefore \frac{\partial Loss}{\partial L_i} = -\left(\frac{\sum_j e^{L_j}}{e^{L_i}}\right)\left[e^{L_i} + \frac{(-1)}{(\sum e^{L_i})^2}\cdot(0+0+\cdots+e^{L_i})\right]$$

$$= -\frac{\sum_j e^{L_j}}{e^{L_i}}\, e^{L_i}\left[1 - \frac{1}{(\sum e^{L_i})^2}\right]$$

$$= \frac{1}{\sum_j e^{L_j}} - \sum_j e^{L_i}$$

Wrong

✗

__/__/____

shape of logits = $n \times$ vocab_size

n → mini batch size.

∴ while creating the differentiation matrix, we gotta consider both.

$i \in \{0, 1, \ldots, n\}$

$c \in \{0, 1, \ldots, 26\}$

$\left(\frac{dLoss}{dLogits}\right)_{i,c} = ?$

Now let loss = $-\log(P_y)$ ← Prob of the expected char 'y'

and let us consider just one eg of the batch for now.

$loss_y = -\log\left(\frac{e^{l_y}}{\sum_j e^{l_j}}\right)$

$\therefore \left(\frac{dloss_y}{dlogits}\right)_c$ ← char.

$\therefore \frac{dloss_y}{dlogit_c} =$

$= -\left(\frac{\sum e^{l_j}}{e^{l_y}}\right) \cdot \left[\frac{e^{l_y} \cdot de^{l_y}}{dl_c} + \frac{(-1)}{(\sum e^{l_j})^2} \cdot \frac{d(\sum e^{l_j})}{dl_c}\right]$

if $y = c$. ↓ $= 0$.

$\therefore \left(\frac{dLoss_y}{dlogits}\right)_c = -\frac{\sum e^{l_j}}{e^{l_y}} \left[0 + \frac{(-1)}{(\sum e^{l_j})^2} \cdot e^{l_y} \cdot c\right]$

$\frac{u}{v} = \frac{v\,du/dx - u\,dv/dx}{v^2}$

$$\text{loss}_y = -\log\left(\frac{e^{\text{logit}_y}}{\sum_j e^{\text{logit}_j}}\right)$$

$$\therefore \left(\frac{\partial \text{loss}_y}{\partial \text{logit}_c}\right) = (-1)\left(\frac{\sum_j e^{l_j}}{e^{l_y}}\right)\left(\frac{\frac{\partial e^{l_y}}{\partial e^{l_c}} \cdot \sum_j e^{l_j} - e^{l_y} \cdot \frac{\partial \sum_j e^{l_j}}{\partial e^{l_c}}}{\left(\sum_j e^{l_j}\right)^2}\right)$$

if $y = c$. ← At the indices (columns) of logprobs that are the correct pred char.

$$\frac{\partial \text{loss}_y}{\partial \text{logit}_c} = (-1)\left(\frac{\sum_j e^{l_j}}{e^{l_y}}\right)\left(\frac{e^{l_y} \cdot \sum_j e^{l_j} - e^{l_y} e^{l_y}}{\left(\sum_j e^{l_j}\right)^2}\right)$$

$$= \frac{e^{l_y} - \left(\sum_j e^{l_j}\right)}{\left(\sum_j e^{l_j}\right)^2}$$

$$= P_c - 1 \quad \text{or} \quad P_y - 1$$

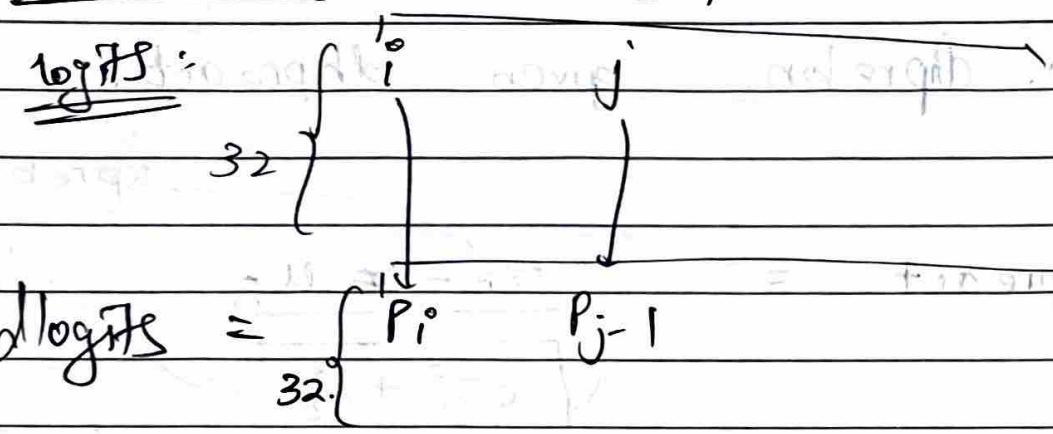
if $y \neq c$. $\frac{\partial \text{loss}_y}{\partial \text{logit}_c} = (-1)\left(\frac{\sum_j e^{l_j}}{e^{l_y}}\right)\left(\frac{0 - e^{l_y} \cdot e^{l_c}}{\left(\sum_j e^{l_j}\right)^2}\right)$

$$\frac{e^{l_c}}{\sum_j e^{l_j}} = P_c$$

where $P_c = \frac{e^{\text{logit}_c}}{\sum_j e^{\text{logit}_j}}$

__/__/__

Via code:



So once you think abt it,

P_c = some kind of prob b/w a 0 and 1.
$P_c - 1$ = b/w -1 and 0.

The grad of $c \neq y$ is +ve.
The grad of $c = y$ is -ve

We wanna pull down the probs of all the probs that are assigned to the incorrect characters and pull up all the correct ones.

Δ (data) = $-lr * grad$.

__/__/____

Same thing for batch normalization:

Calc. dhprebn. given dhpreact.

hprebn.

$$h.\text{preact} = \frac{x_i^\circ - \mu_B}{\sqrt{\sigma_B^2 + \epsilon_0}},$$

hprebn : 32

for a particular input,

$$\left(\frac{\partial h\text{preact}}{\partial h\text{prebn}}\right)_{ij} = \frac{\partial\left(\frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon_0}}\right)_j}{\partial x_i}$$

Good

$$\therefore \left(\frac{\partial h.\text{preact}}{\partial h\text{prebn}}\right)_{ij} = \frac{\left(\frac{x_i^\circ - \mu_B}{\sqrt{\sigma_B^2 + \epsilon_0}}\right)}{\partial x_i^\circ}$$

$$\sigma_B^2 = \frac{\sum (x_i - \mu)^2}{n-1}$$

$$\frac{\partial \sigma_B^2}{\partial x_i} = \frac{2x_i}{}$$

$$= \frac{\sqrt{\sigma_B^2 + \epsilon_0}\left[1 - \frac{1}{n}\right] - (x_i - \mu_B)\frac{1}{\sqrt{\sigma_B^2 + \epsilon_0}}\left(\frac{2x_i}{}\right)}{\sigma_B^2 + \epsilon_0}$$

$$\frac{\partial \sigma_B^2}{\partial x_i^\circ}$$

Contd. Makemore lec-5:

Building a Wavenet:

first we wanna make some improvements and revise our og code:

layers:

Linear:

```
def:
    fan_in, fan_out, bias = True)
    weight = randn(in, out) / √in
    bias   = randn(out) * 0.01   or torch.zeros(out)

call:
    out = x @ w.
    if bias:
        out += bias
    ret out.

param:
    All params → weight + bias if bias.
```

class BatchNorm:

def: (dim, ~~training = True~~, eps = 10^{-5}, momentum = 0.1)

~~Y, β = zeros randn (dim).~~

Y = ones(dim)

β = zeros(dim).

training = True.

run_mean = zeros (dim)

run_var. = ones (dim)

call: (x)

if training:

x_m = x.mean()

x_v = x.var().

else:

x_m = running_m

x_v = run_var

$$\hat{x} = \frac{x - x\_m}{\sqrt{x\_var + \varepsilon}}$$

out = $Y * \hat{x} + \beta$.

with torch.no_grad():

no backprop [running_mean = r_m * (1 - mom+m) + x_m * mom

run_var = r_v * (1 - mtm) + r_v * mtm

Params:

(Y, β)

Tanh:

call: (x)

return torch.tanh(x)

params:

[ ] x

Initial setup:

C = randn(vocab, emb) — 27, 10.

layers = [linear, BN, tanh, linear]

(cont* emb, n_hidden) (n_hidden) (n_hidden, vocab)

x = C[Xb]

for l in layers

x = l(x)

loss = F.cross_entropy(x)

for l in layers:

l.wt *= 0.1.

(to make initial loss less overconfident)

Setup:

torch.manual_seed (—)

n_emd = 10

n_hidden = 200.

Xb

c = randn (~~vocab_size~~ vocab_size, nemd).

] gets replaced

~~C [Xb].~~

layers = [Linear (~~batch_size~~ * ~~block_size~~ context, n_hidden),

BN (n_hidden),

tanh(),

Linear (n_hidden, vocab_size)].

with no_grad():

for l in layers:

l.weight *= 0.1 # decrease initial overconfidence.

params = [p for l in layer for p in l.parameters()]

+ [C].

→ we need.

for p in params():

p.requires_grad

iter = 200,000.
batch_size = 32
loss_i = [].

loop(iter): i.

~~x = [[X~~

Xbix = ~~multi~~ randint (batch_size).

Xb, Yb = Xtr(ix), Ytr(ix)

x = [[Xb]]
x = x.view (---)
for l in layers
    x = L(x).

] repk.

loss = F. cross_entropy (x, Yb)

for p in params(): p.grad = 0.
loss.backward()

loss_i [i] = loss.

lr = 0.1 if i < 150,000 else 0.01.
p.data -= lr * p.grad.

testing:

for l in layers:

    l.trainings = false

    :

and sampling:

---

fixing lr plot.

loss_i.view(-1, 100) or loss_i.view(-1, 100).

---

putting lookup and embedding table into layers.

class Embedding:

    def (num_emb, emb_dim)

        wt = torch.randn((n_e, e_d))

    call (ix)

    return wt[ix].

    params.

        ret [wt].

class Flatten

    call (x)

    out = x.view (x.shape(0), -1))

        ret out.

    params

        ret ().

∴ now we don't need C,

→ params directly takes from layers.

→ Add:

setup { Embedding (n_emb, emb_dim),
Flatten
to layers.
now need for explicit C, = [C] in params,

Also

loop { x = C[Xb]
x = x.view] x → x = Xb
for l in layer:
x = l(x)

And now,

class Sequential:

def: (layers=list),
self.layers = layers.

call: (x)
for l in layers:
x = l(x)
return x

[- - - - for for]
return

---

now.
x = Xb.
for ...
x = ...

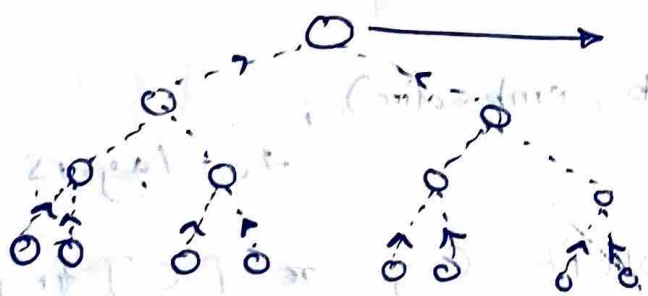
} → logits = model(Xb).

model = Seq([Emb, Flat, Lin, BN, tanh, Lin])

↳ Set layers to model.layers if to training in.

false for testing

Introducing the WaveNet:



currently, if we increase context_size to 8

then we get: emb[Xb]

(batch_size, 8, 10) —flatten→ (batch_siz., 80)

but we want

(b_s, 8, 10) ⟶ (b_s, 4, 20)

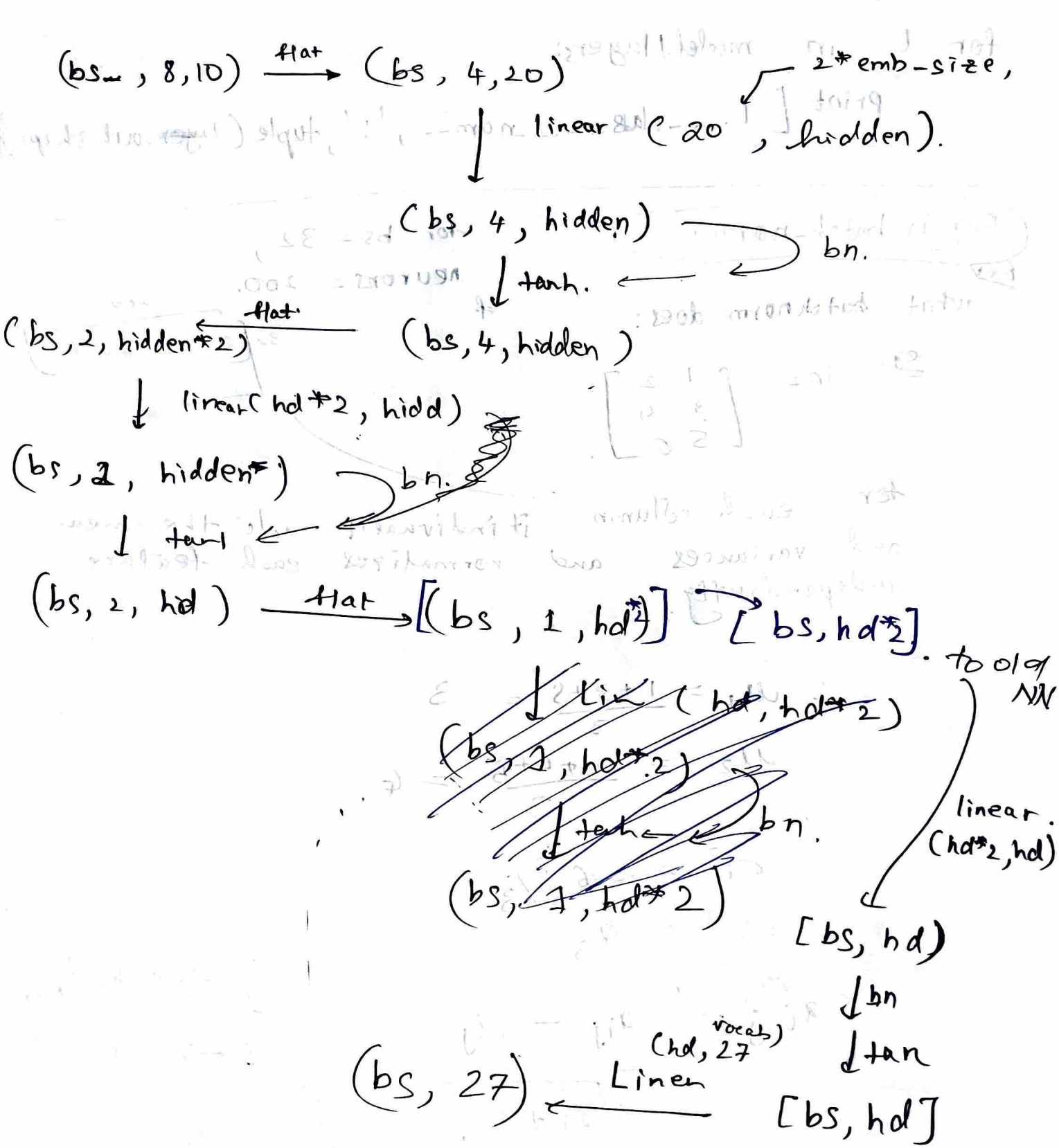
instead of (b_s, 80) @ (80, 200) + (200)

emb wt. bias.

we want (b_s, 4, 20) @ (20, hid1) + (hid1)

So now in class Flatten, we want to basically each time concatenate 2 char embeddings at a time, pass them thru a linear and tanh layer and do the same thing twice more.

we wants:


(bs, 8, 10)
flat
(bs, 4, 20)
2*emb-size,
linear (20, hidden).
(bs, 4, hidden)
bn.
tanh.
flat
(bs, 2, hidden*2)
(bs, 4, hidden)
linear(hd*2, hidd)
(bs, 2, hidden*)
bn.
tanh
(bs, 2, hd)
flat
[(bs, 1, hd*2)]
[bs, hd*2].
to old NN
Lin (hd, hd*2)
(bs, 1, hd*2)
tanh
bn.
linear. (hd*2, hd)
(bs, 1, hd*2)
[bs, hd)
bn
tan
(hd, 27 vocab)
(bs, 27)
Linear
[bs, hd]

A very nice loop to see this in code:

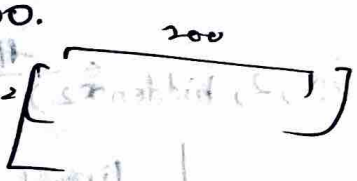
```
for L in model.layers:
    print [ L.__class__.__name__ , ':' , tuple( layer.out.shape) ]
```

Bug in batch_norm:

Rev

what batchnorm does:

for bs = 32, neurons = 200.



eg: $in = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$.

for each column it indivually calc the mean and variances and normalizes each feature independently.

$\therefore \mu_1 = \frac{1+3+5}{3} = 3$

$\mu_2 = \frac{2+4+6}{3} = 4.$

$\sigma_1^2 = 8/3$

$\sigma_2^2 = 8/3$

$\therefore \hat{x}_{ij} = \frac{x_{ij} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$

$j \rightarrow$ column/feature

$i \rightarrow$ row.

↳ batches

But now during varenet impl,

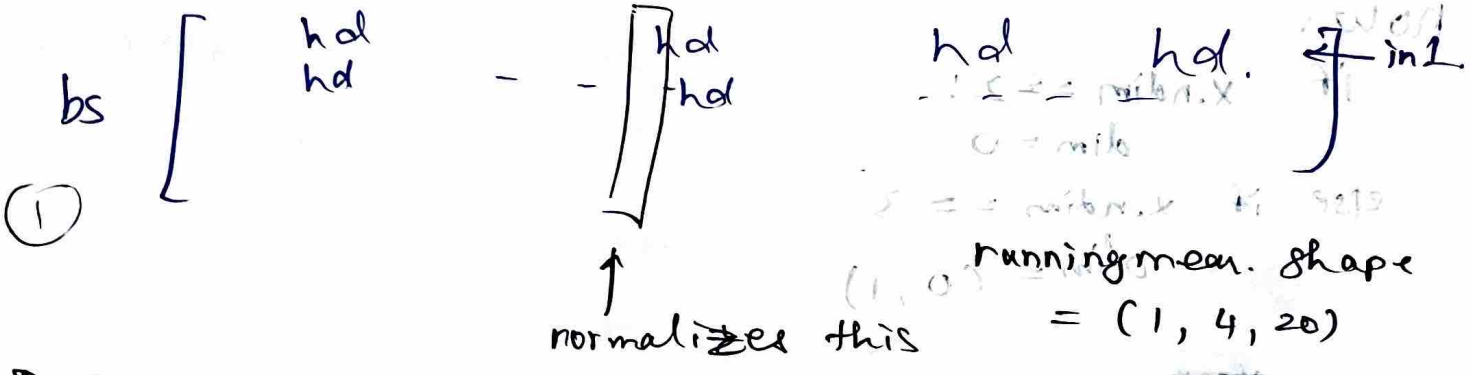
many ~~are~~ inputs are like (bs, 4, hd)

or (bs, 2, hd).

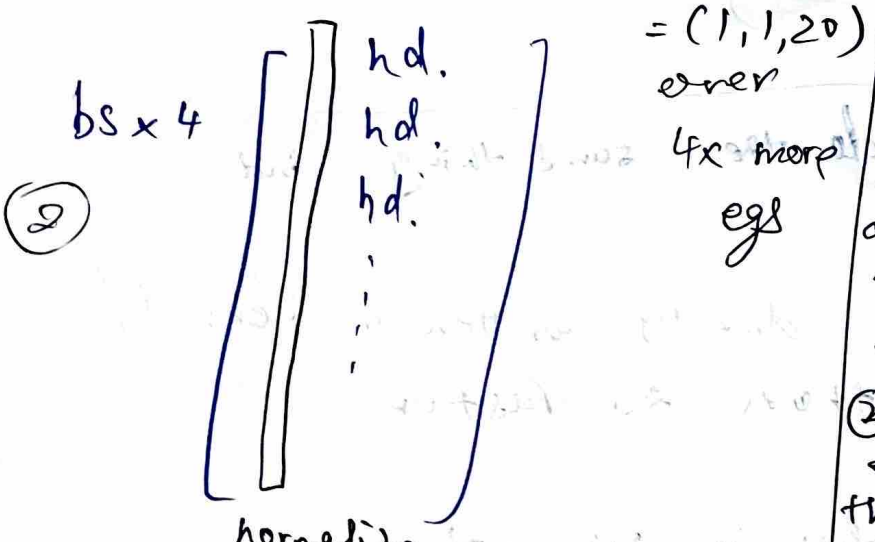
So now, it still ~~is~~ normalizes over bs;

So it treats each of the 4*hd numbers ~~as~~ as individual ~~features~~.

its like



But.
We want



= (1, 1, 20)
over
4x more
egs

Reasons for this:

① column 'j' of each vector repr the same feature so in meth-① we are treating them diff. They should be normalized together

② meth-① reduces the ~~num~~ num of examples ~~used~~ to calc the mean, variance stats.

③ final dimensions of γ, β, etc are gonna be emb_size only so better train them like that.

N*L, D ✓

N, L*D ✗

---

fix:

earlier:

```
xmean = x.mean(0, keep=T)
xvar = x.var (0, keep=T).
```

now:

```
if x.ndim == 2:
        dim = 0
else if x.ndim == 3
        dim = (0, 1)
xmean = x.mean(dim, k—)
xvar = x.var(dim, —)
```

Convolution: CNNs do the same thing, but more efficient

① for loop is somehow directly written in kernel/CUDA Shit, not in python so faster

② Reuse of nodes in Wavenet