BLG561E FALL 2021

Deep Learning

02.11.2021

Gözde ÜNAL

$\frac{\partial \ell}{\partial a_1} \cdot \frac{\partial a_1}{\partial w^1} + 2\lambda w^1 = \nabla_{\underline{w^1}} \mathcal{L}$

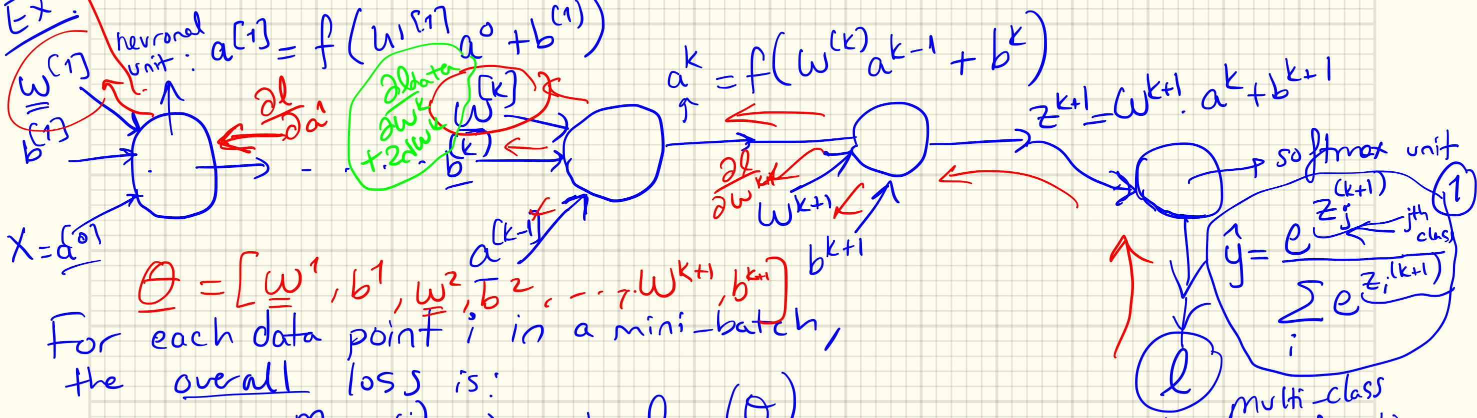
$w^{(i)(t+1)} \leftarrow w^{(i)(t)} - LR \cdot \nabla_{w^i} \mathcal{L}_{overall}$

Recap: We covered Backprop last time;

Today, we've seen computation graphs & how to backprop over the computation graph.

We've learned how to backpropagate the derivative of the loss fn.

Ex:



$\theta = [w^1, b^1, w^2, b^2, \ldots, w^{k+1}, b^{k+1}]$

For each data point i in a mini-batch, the overall loss is:

multi-class classification loss ②

$$\mathcal{L}_{overall}(\theta) = \frac{1}{M} \sum_{i=1}^{m} \ell^{(i)}_{data}(\theta) + \lambda \cdot \ell_{Regularizer}(\theta)$$

$\lambda \cdot \|\underline{w}^1\|_2^2$

$$\ell^{(i)}_{data} = \sum_{j=1}^{C} y_j^{(i)} \log \hat{y}_j^{(i)}$$

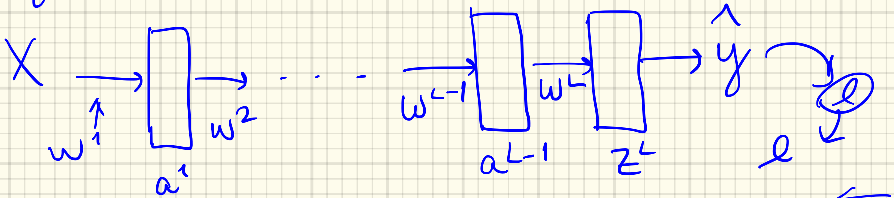
eg.
$$\frac{\partial \mathcal{L}_{overall}}{\partial \underline{w}^1} = \frac{\partial \ell_{data}}{\partial w^{(1)}} + 2\lambda \underline{w}^{(1)} \quad ★$$

During backprop, Only this gradient propagates back over the computation graph due to its dependence on the intermediate layer activations. ∴ We add the regularizer derivative ★ at the update step.

Vanishing / Exploding Gradient Problem: in $\boxed{\text{Deep}}$ Networks

Say L Layers in a FCN

Say activation $f(z) = z$ linear.



Recall

← Forward pass uses $\underline{\underline{W}}$ multip.

Backward pass uses $\underline{\underline{W}}^T$ multip.

$$\hat{y} = \underbrace{\underline{\underline{W}}^L \underline{\underline{W}}^{L-1} \cdots\cdots \underline{\underline{W}}^2 \underline{\underline{W}}^1}\, X$$

$$\frac{\partial \ell}{\partial W^1} = d\underline{\underline{W}}^1 = \underbrace{(W^1)^T (W^2)^T \cdots (W^L)^T}\, \underbrace{g_L}_{\frac{\partial \ell}{\partial a_L}} \cdot f'(.)\ (\text{others})$$

$$W^1 \leftarrow W^1 - \alpha\, dW^1$$

Say $\underline{\underline{W}} = 1.2\, \underline{\underline{I}}$ (say $L = 200$ # layers)

$$\hat{y} = \underbrace{(1.2)^{200}}_{\text{explosion}} \underline{\underline{I}} \rightarrow dW^1 = \underbrace{(1.2)^{200}}\, \underline{\underline{I}}\, (\text{others})$$

explosion ↑↑↑

or $\underline{\underline{W}} = 0.5\, \underline{\underline{I}}$

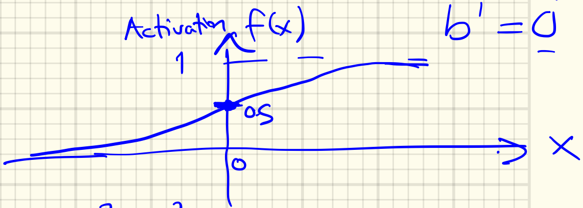
$$dW^1 = \underbrace{(0.5)^L}_{\underbrace{(0.5).(0.5) \cdots (0.5)}_{200}} \cdot \underline{\underline{I}}\ (\text{others}) \longrightarrow 0$$

vanishing gradient problem

To deal w/ the vanishing (exploding) gradients, we try to initialize our network weights carefully.

Q: How do we initialize Parameters?

A. What if we initialize all parameters to zero: $\forall i \quad W^i = 0, \quad b^i = 0$



eg. 3 layer network

wrong ✗

$$z^3 = W^3 a^2 + b^3$$

$$a^3 = f(z^3)$$

$$x \to z^1, a^1 \to z^2, a^2$$

Network outputs 0.5 no matter what the input is.

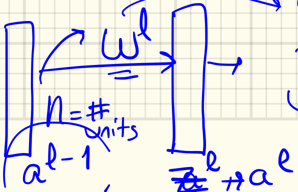
A. What if all parameters are set to some non-zero const. value.

✗

$$a^1 = f(z^1) = f(W^1 x + b^1) \to \begin{pmatrix} \vdots \end{pmatrix} \to \text{all the same const.}$$

(1) Random small weights w/ small variance:

weights at the l^{th} layer:

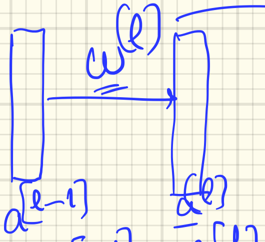


$$z^l = W_1^l a_1^{l-1} + W_2^l a_2^{l-1} + \dots + W_n^l a_n^{l-1}$$

$$Var(W^l) \propto \frac{1}{n} \leftarrow \text{\# units in } (l-1)^{st} \text{ layer.}$$

$\rightarrow \underline{\underline{W}}^{l} \sim \mathcal{N}\left(0, \frac{1}{n^{[l-1]}}\right)$ std $\propto \frac{1}{\sqrt{n}}$

② Xavier-He Initialization:



$\underline{\underline{W}}^{l} \sim \mathcal{N}\left(0, \sqrt{\frac{2}{n^{[l]}+n^{[l-1]}}}\right)$

#neurons: $n^{[l-1]}$, $n^{[l]}$

Widely used in practice.

③ There are others such as Glorot initialization.

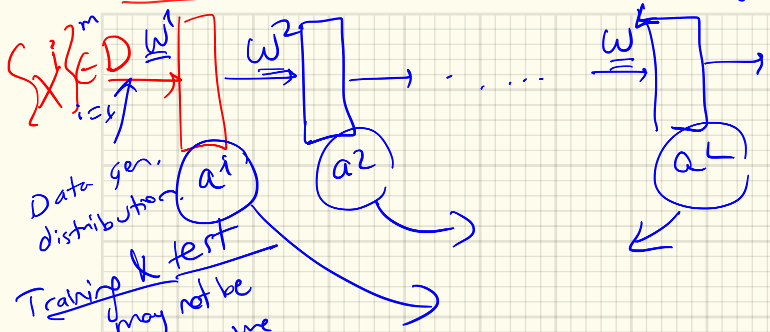
Check!

Exercise: Go & find out other network weight initialization schemes/suggestions

④ We will talk about Transfer Learning:

then we will initialize our networks w/ an already trained (pretrained) model $\rightarrow$ their weights are used as our initialization.

BATCH NORMALIZATION (BN) applies normalization to deep layers of your NN.



Training & test data may not be coming from the same distrib.

COVARIATE SHIFT:

Learning on a shifting input distrib. for each layer.

Typically: $\mu = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}$, $\sigma^2 = \frac{1}{m-1}\sum_{i=1}^{m}(x^i-\mu)^2$

we normalize the training & test inputs to the network.

Problem: Intermediate layers in our networks shifting distributions.

BN addresses this problem: Normalization is added before activation, on $z^{(l)}$'s $\forall l$,

∴ For an intermediate layer i, z^i are normalized as:

$$z^i_{norm} = \frac{z^i-\mu}{\sqrt{\sigma^2+\varepsilon}} \; ; \; \mu = \frac{1}{B}\sum_{i=1}^{B} z^i \; ; \; \sigma^2 = \frac{1}{B}\sum_{i}(z^i-\mu)^2 \Rightarrow$$

batch samples

→ Our hidden units z^i_{norm} have $(0, \sigma^2)$

→ Now → we make them (γ, β)

$$\tilde{z}^{(i)} = \gamma^{(i)} z^{(i)}_{norm} + \beta^{(i)}$$

Now, we have 2 more learnable parameters for each layer.

Parameters of our model becomes:

$\underline{W^1}, \beta^1, \gamma^1$
W^2, β^2, γ^2
$\vdots$
$\underline{W^L}, \beta^L, \gamma^L$

Training time

Add SGD / Optimizer:

$$\beta^i = \beta^i - \alpha\, d\beta^i$$
$$\gamma^i = \gamma^i - \alpha\, d\gamma^i$$

Note; w/ BN, bias is eliminated:

$$z^l = W^l a^{l-1} + \cancel{b^{l-1}}$$

eliminated

$$z^l_{norm} = \frac{z^l - \mu}{\sigma}$$

$$\tilde{z}^l = \gamma^l z^l_{norm} + \beta^l$$

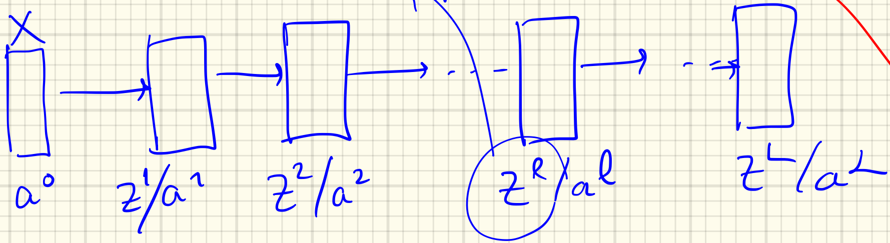
BN at Test Time: We need μ, σ^2 estimated through weighted averaging over mini-batches during training:

$$\mu^t_{av} \mathrel{+}= \alpha \mu^{t-1} + (1-\alpha)\mu^t$$

$$\sigma_{av} \mathrel{+}= \ldots$$

produce an overall μ_{av} & σ^2_{av}.

Test Time: $z^{(l)}_{norm} = \frac{z^l - \mu_{av}}{\sqrt{\sigma^2_{av}+\varepsilon}} \rightarrow \tilde{z}^l = \gamma^l \cdot z^l_{norm} + \beta^l$ ✓



γ^l, β^l are already learned.

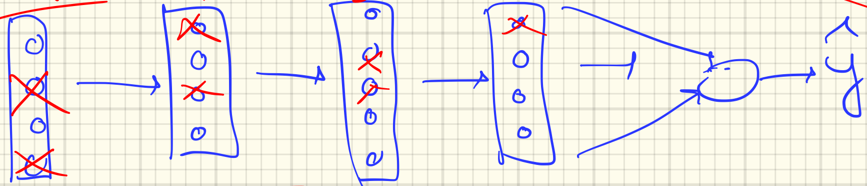
$a^l = f(\tilde{z}^l)$
↑ activation

pytorch → torch.nn.batchNorm ←

Dropout: → creates for us Ensemble of Networks. → come up w/ confidence measures

check → MCDrop-out : →

w/ prob. p : kill / randomly eliminate some nodes / hidden units w/ a dropout prob of p.



Helps w/ overfitting

Inverted Drop-out: In training: scale up; $1/p$ (to match the test time scales)

Test time: B/c all neurons are active we must scale the activations by p, so that

→ s.t. Output at test time = expected output at Training time

Note: In today's lecture, we also went thru slides from cs 231n (Stanford) & some slides from Francois Fleuret's Deep learning course at EPFL.

(You can study relevant material, if you want, from those websites.)

Today, we covered:

- Batch Norm, Computation graphs,
- Dropout, vanishing gradients,
- activation functions,
- initializing NN weights.