BLG 561 E FALL 2021

Deep Learning

26.10.2021

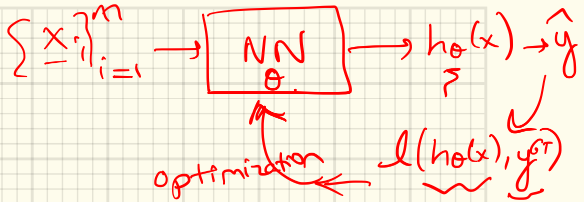
Gözde ÜNAL

All ML algorithms : 3 components

1) Hypothesis Class : ✓ NN ←

2) Loss function : ✓

3) Optimization ← ←

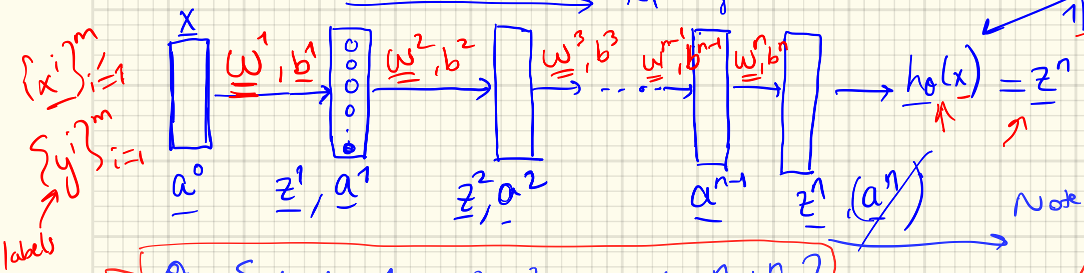


Neural Networks (Fully-Connected,
NNs ANNs, Multi-Layered Perceptron (MLP))

Ex: Multi-Class Classification : n-layer ANN. → ① ✓

depth of the NN

1) Hypothesis function ✓



② multi-class Hinge Loss ✓

$L_{sum}(\theta)$

Note: Typically, the last layer is used w/o activation fn.

$$\theta = \{W^1, b^1, W^2, b^2, \ldots, W^n, b^n\}$$

$$L_{sum} = \sum_{i=1}^{m} \ell(h_\theta(x^i), y^{(i)}) = \sum_{i=1}^{m} \sum \max\left(0, \underbrace{h_\theta(x^i)_j}_{S_j} - h_\theta(x^i)_{y^i} + 1\right)$$

jth argument of vector fn h

③ Optimization/Training the NN: estimating parameters θ !

Today

★ ③ How do we optimize the loss? $\underline{\theta}^* = \arg\min_{\theta} \underbrace{\sum_{i=1}^{m} \ell(h_\theta(x^i), y^i)}_{\mathcal{L}(\theta)}$.

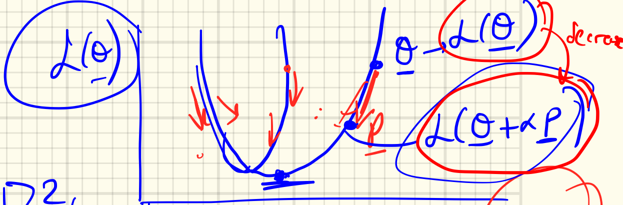
$\underline{Q}$: Is this loss fn. convex? Not convex

$$h_\theta(x^i) = \underline{W}^n(\dots\dots \underline{W}^3 f_2(\underline{W}^2 f_1(\underline{W}^1 \underline{x}^i + \underline{b}^1) + \underline{b}^2) + b^3 + \dots) + b^n$$

Before $h_\theta(x) = \theta^T x$ linear hypothesis → - - .

Deep Learning models: we work w/ non-convex optimization problem

$\mathcal{L}(\underline{\theta})$: overall loss function $\underline{p}$: search direction



T.S. expansion:

$$\mathcal{L}(\underline{\theta}+\alpha \underline{p}) \approx \mathcal{L}(\underline{\theta}) + \underline{p} \cdot \underbrace{\nabla_\theta \mathcal{L}(\theta)} + \underbrace{\frac{1}{2}\underline{p}^T \nabla^2_\theta \mathcal{L}(\theta)\underline{p}}_{HOT}$$

Let $\boxed{\underline{P} = -\alpha \nabla_\theta \mathcal{L}(\theta)}$

$$\mathcal{L}(\underline{\theta}+\alpha\underline{p}) \cancel{\approx} \mathcal{L}(\underline{\theta}) - \alpha \underbrace{\|\nabla_\theta \mathcal{L}(\theta)\|^2}$$

$$\leq \mathcal{L}(\theta) - \alpha \underbrace{\quad}_{\geq 0}$$

$\underline{\theta}$ in millions billions

Recall: Newton method:

Search dir: $\propto \underbrace{\nabla_\theta^2 \mathcal{L}}$: Hessian

GRADIENT DESCENT Algorithm (GD): We can use it whether $\mathcal{L}(\theta)$ is convex or not!

GD Algorithm: θ_0 : "initial" parameters ←

Iterate until "convergence" ←

$$\underline{\theta}^{k+1} \leftarrow \underline{\theta}^{k} - \underbrace{\alpha}_{} \cdot \nabla_\theta \mathcal{L}(\underline{\theta})$$

Learning Rate (LR) : step size } one of the important hyperparameters

e.g. Newton method ;
uses other search dir.

$$\underline{\theta} \leftarrow \underline{\theta} - \underbrace{\left(\nabla^2_\theta \mathcal{L}(\theta)\right)^{-1} \cdot \nabla_\theta \mathcal{L}(\theta)}_{\underline{p}}$$

$\underline{p}$: search direction.

Note: Deep NNs (Millions / Billions of parameters) $\underline{\theta}_{100m \times 1}$ $\xrightarrow{\text{imagine computing}}$ $\nabla^2_\theta \mathcal{L} = \text{Hessian}$!!

↳ + inverting !!

$\exists$ Quasi-Newton methods ;

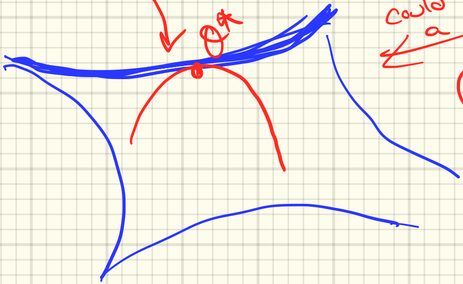
We approximate the Hessian thru gradients ; e.g. BFGS.

Q: You have your $\mathcal{L}(\theta)$; is it enough to set $\nabla_\theta \mathcal{L}(\theta) = 0$ to obtain your minimizer of the loss function?

Thm: Necessary 1st order condition: If θ^* is a local minimizer & $\mathcal{L}$ is continuously differentiable then $\nabla_\theta \mathcal{L}(\theta^*) = 0$

Any local minimizer θ^* $\Longrightarrow$ $\nabla_\theta \mathcal{L}(\theta^*) = 0$ (Necessary Cond.)

θ^* is a stationary point $\Longleftarrow$ $\nabla_\theta \mathcal{L}(\theta^*) = 0$ (Sufficient Cond)



could be a saddle point

(A saddle point is a stationary point.

Hessian $\nabla^2_\theta \mathcal{L}$ is not +ve definite

↳ its eigen values both > 0 < 0

Thm 2nd order Necessary Condition: Add $\nabla^2_\theta \mathcal{L}$ exists, then $\nabla \mathcal{L}(\theta^*) = 0$

~~Given~~ If θ^* is a local minimizer

$\nabla^2 \mathcal{L}(\theta^*) \geq 0$

Thm: 2^{nd} order Sufficient Conditions:

If $\nabla^2_\theta \mathcal{L}$ is continuous & positive definite & $\nabla_\theta \mathcal{L}(\theta^*) = 0$

then θ^* is a local minimizer of $\mathcal{L}$.

$\therefore$ If $\nabla_\theta \mathcal{L} = 0$ & $\underline{\nabla^2 \mathcal{L} \text{ p.d.}} \rightarrow \theta^*$ is a minimizer.

In Summary: In deep learning: We want $\nabla_\theta \mathcal{L}(\theta)$ to vanish.

Search θ^* from $\nabla_\theta \mathcal{L}(\theta) = 0$.

GD: (Loss: uses all samples) $\boxed{\mathcal{L} = \frac{1}{m} \sum_{i=1}^{m} \ell(h_\theta(x^i), y^i)}$

```
Repeat
    For i=1:m
        $grad^{(i)} \leftarrow \nabla_\theta \ell(h_\theta(x^i), y^i)$
    Update the parameters
        $\theta \leftarrow \theta - \underset{LR}{\alpha} \underbrace{\frac{1}{m} \sum_{i=1}^{m} grad^{(i)}}_{\nabla_\theta \mathcal{L}}$
until "convergence"
```

SGD: $\boxed{\text{Stochastic}}$ GD ↑ introduces randomness.

```
For i=1:m
    $\theta \leftarrow \theta - \alpha \underbrace{\nabla_\theta \ell(h_\theta(x^i), y^i)}_{grad^{(i)} = \text{loss for 1 sample}}$
```

Noisy gradient update.

In practice: we use $\underbrace{\text{MiniBatch GD}}_{\Leftrightarrow \text{SGD}}$: a compromise btw GD & SGD.

Note: Vanilla SGD uses $B = 1$
not used in practice

Let B : batchsize
another hyperparameter you have to fit.

e.g. $B = 8$
$= 16$
$= 32$
$= 64$
$= 128$

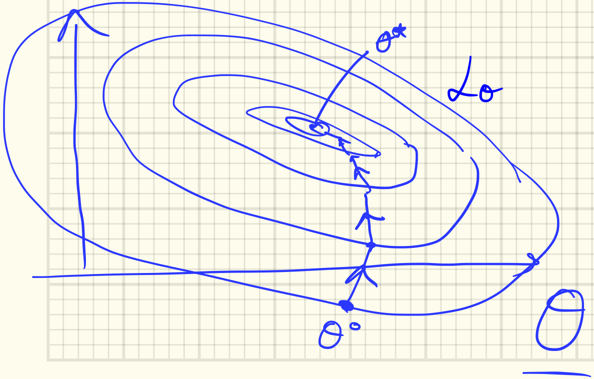
$$\mathcal{L}_{mB} = \frac{1}{B} \underbrace{\sum_{i=1}^{B} \ell(h_\theta(x^i), y^i)}_{\nabla_\theta \ell^i}$$

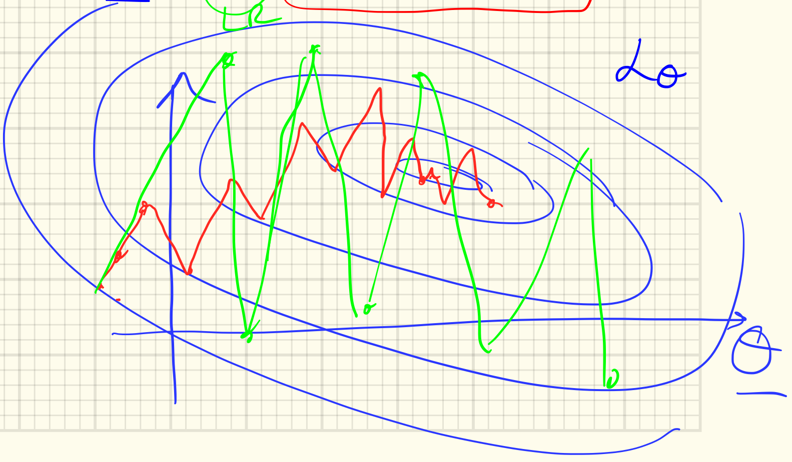
SGD (minibatch): For $k = 1 \ldots\ldots \left\lfloor \frac{m}{B} \right\rfloor$ (# minibatches)

$$\underline{\theta} \leftarrow \underline{\theta} - \underset{LR}{\alpha} \nabla_\theta \mathcal{L}_{mB}(\underline{\theta})$$

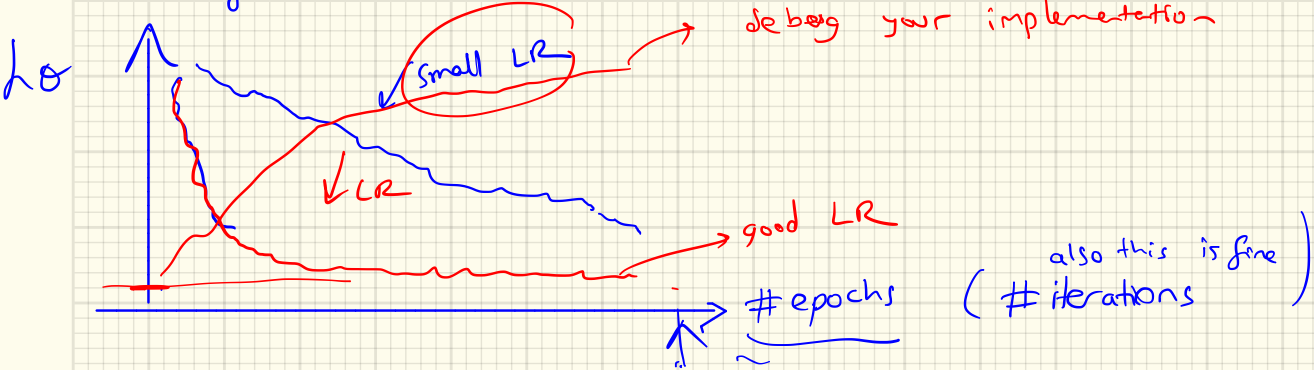
1 epoch: once you've seen all the samples in your training data

GD.

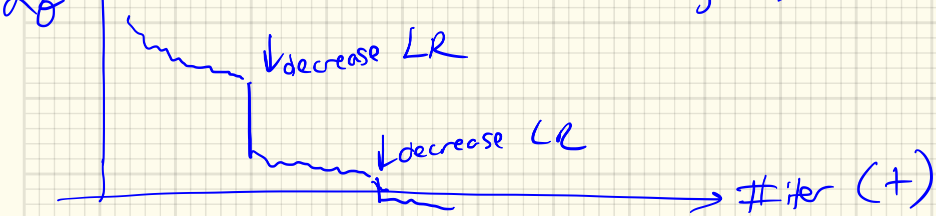




Checking for convergence in your optimization



LR regimes → ∃ different regimes : hyper setting.



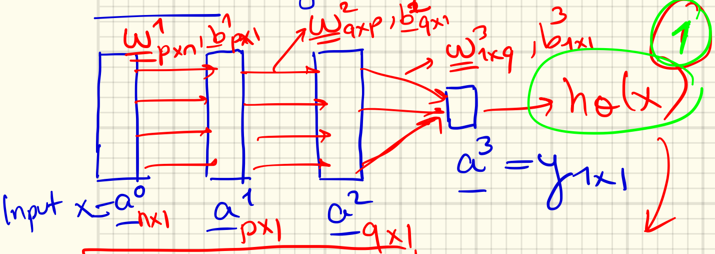
→ Always check your loss curves.

Optimization ? for NN

Q: How do we compute the gradient : $\nabla_{\theta} \ell(h_{\theta}(x^i, y^i))$? Backpropagation

$\underline{\theta}$: set of parameters of the NN : $\underline{\underline{W}}^1, \underline{b}^1, \underline{\underline{W}}^2, \underline{b}^2 \ldots, \underline{\underline{W}}^k, \underline{b}^k$.

K = 3 - layer NN (ANN = FCN = MLP)



eg. LS_loss

$$② \; \underbrace{\ell}_{\text{Loss}}(\underbrace{a^3}_{h_\theta(x)}, \overset{\text{GT}}{y}) = \frac{1}{2}(y - a^3)^2$$

$$\underline{\theta} = (\underline{\underline{W}}^1, \underline{b}^1, \underline{\underline{W}}^2, \underline{b}^2, \underline{\underline{W}}^3, \underline{b}^3)$$

Forward Pass : Compute $\underline{a}^1, \underline{a}^2, \underline{a}^3, \ldots, \underline{a}^k$; $\ell(a^k, y)$

$$\underline{a}^1 = f_1(\underline{\underline{W}}^1 a^0 + \underline{b}^1)$$

$$\underline{a}^2 = f_2(\underline{\underline{W}}^2 \underline{a}^1 + \underline{b}^2)$$

$$\underline{a}^3 = f_3(\underbrace{\underline{\underline{W}}^3 \underline{a}^2 + \underline{b}^3}_{z^3})$$

$$\underline{a}^i = f_i(\underline{\underline{W}}^i \underline{a}^{i-1} + \underline{b}^i)$$

Also $z^i = \underline{\underline{W}}^i \underline{a}^{i-1} + \underline{b}^i$

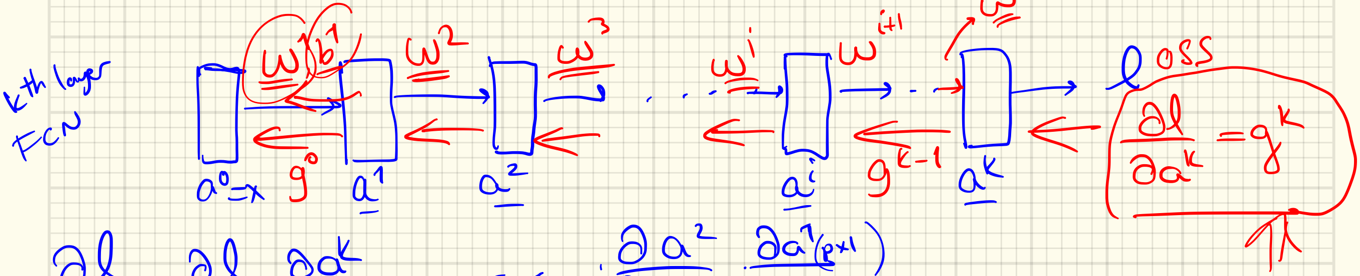
① Hypothesis function for each layer in ANN (FCN)

③ Optimization (BACKPROP)

$$\left.\frac{\partial l}{\partial w^i},\ \frac{\partial l}{\partial b^i}\right\} = \frac{\partial l}{\partial \theta} = \nabla_\theta l$$

Backward Pass: Based on the chain rule:

$$\frac{\partial l}{\partial w^1} = \frac{\partial l}{\partial a^k}\,\frac{\partial a^k}{\partial a^{k-1}} \cdots \frac{\partial a^3}{\partial a^2}\cdot\frac{\partial a^2}{\partial a^1}\cdot\frac{\partial a^1}{\partial w^1} \leftarrow$$



$$\frac{\partial l}{\partial b^1} = \frac{\partial l}{\partial a^k}\,\frac{\partial a^k}{\partial a^{k-1}} - - - \frac{\partial a^2}{\partial a^1}\,\frac{\partial a^1_{(p\times 1)}}{\partial b^1_{(p\times 1)}}$$

$$\boxed{\frac{\partial a^i}{\partial b^i}} = \frac{\partial f}{\partial z^i}\cdot\frac{\partial z^i}{\partial b^i} = \underbrace{\mathrm{diag}\, f'(z^i)}_{\text{diag matrix}}\cdot\mathbb{1} \longrightarrow \text{vector of 1's}$$

$$\boxed{\frac{\partial a^i}{\partial a^{i-1}}} = \frac{\partial f}{\partial z^i}\cdot\frac{\partial z^i}{\partial a^{i-1}} = \mathrm{diag}\, f'(z^i)\cdot w^i \Longleftarrow$$

$$\boxed{\frac{\partial a^i}{\partial w^i}} = \frac{\partial f}{\partial z^i}\cdot\boxed{\frac{\partial z^i}{\partial \mathrm{vec}(w^i)}} \longrightarrow$$

$$\rightarrow \underline{z}^i = \underline{\underline{I}}\,\underline{\underline{W}}^i\,\underline{a}^{i-1} + \underline{b}^i$$

$$z^i = (a^{i-1})^T \otimes \underline{\underline{I}}\;\; vec(\underline{\underline{W}}^i) + \underline{b}^i$$

$$\frac{\partial z^i}{\partial \underbrace{vec(\underline{\underline{W}}^i)}_{\text{vector}}} = (\underline{a}^{i-1})^T \otimes \underline{\underline{I}} \quad \leftarrow$$

We are using this identity

$$vec(\underline{\underline{A}}\,\underline{\underline{B}}\,\underline{\underline{C}}) = (\underline{\underline{C}}^T \otimes \underline{\underline{A}}) . vec\,\underline{\underline{B}}$$

Recall $\otimes$ Kronecker product

$$C = (A_{m\times n} \otimes B_{p\times q}) \in \mathbb{R}^{mp\times nq}$$

$$= \begin{bmatrix} a_{11}\underline{\underline{B}} & a_{12}\underline{\underline{B}} & \cdots & a_{1n}\underline{\underline{B}} \\ a_{21}\underline{\underline{B}} & \cdots & & \\ \vdots & & & \\ a_{m1}\underline{\underline{B}} & \cdots & & a_{mn}\underline{\underline{B}} \end{bmatrix}$$

$$\Rightarrow \underline{g}^i \triangleq \frac{\partial l}{\partial \underline{a}^i} \rightarrow \text{upstream gradients}$$

$i = 1, \ldots, k$

start w/ this → $g^k \xrightarrow{\text{calculate}} g^{k-1}, \ldots, g^1$: want a recursive computation:

$$g^{i-1} \triangleq \frac{\partial l}{\partial a^{i-1}} = \overbrace{\frac{\partial l}{\partial a^k} \cdot \frac{\partial a^k}{\partial a^{k-1}} \cdots \frac{\partial a^{i+1}}{\partial a^i}}^{g^i} \cdot \underbrace{\frac{\partial a^i}{\partial a^{i-1}}}_{} = (\underline{\underline{W}}^i)^T diag f'_{(z^i)} \cdot g^i$$

→ Target derivatives: $\nabla_{\underline{\underline{W}}^i} l$, $\nabla_{\underline{b}} l$, $i = 1, \ldots, k$ $\Rightarrow$

$$\nabla_{\underline{\underline{w}}^i} \ell = \underbrace{g^{i-1}}_{\text{upstream gradient}} \cdot \underbrace{\operatorname{diag} f_i'(a^{i-1})^T \otimes \underline{\underline{I}}}_{\text{local gradient; } \frac{\partial a^i}{\partial \underline{\underline{w}}^i} \rightarrow vec(\underline{\underline{w}}^i)}$$

Summarize :

1) Forward Pass : $$\underline{a}^i = f_i(\underbrace{\underline{\underline{w}}^i \underline{a}^{i-1} + \underline{b}^i}_{\underline{z}^i})$$

$$\Longrightarrow$$

Backpropagation Algorithm for a k-layer NN:

Parameters we search for

$$\underline{\theta} = \{ \underline{\underline{W}}^1, b^1, \ldots, \underline{\underline{W}}^k, b^k \}$$

1) Forward Pass: Compute $\underline{a}^1, \ldots, \underline{a}^k, \ell(\underline{a}^k, y) \Leftarrow$

Also, compute $\boxed{\text{local gradients}}$: $\frac{\partial a^i}{\partial a^{i-1}}, \frac{\partial a^i}{\partial b^i}, \frac{\partial a^i}{\partial \underline{\underline{W}}^i}$

2) Backward Pass: Compute the upstream Gradients:

Compute recursively: $g^k, \ldots, g^1$ $\left(g^k = \frac{\partial \ell}{\partial a^k} \right) \left(g^{k-1} \right) \ldots g^{k-2} \ldots$

$$\boxed{\underline{g}^i = (\underline{\underline{W}}^i)^T \, \text{diag} f'(z^i) \cdot g^{i-1}}$$

Note:
Forward Pass uses multiply by $\underline{\underline{W}}^i$
Backward Pass uses multiply by $\underline{\underline{W}}^{iT}$

3) Compute the gradients of the loss fn. w.r.t. $\underline{\theta}$:

$$\nabla_{\underline{\underline{W}}^i_{n\times p}} \ell \neq \frac{\partial \ell}{\partial (\text{vec}\, \underline{\underline{W}}^i)_{1\times np}} = \left[\underbrace{g^i}_{\text{upstream grad}} \left(\underbrace{\text{diag} f'_i \, (\underline{a}^{i-1})^T \otimes \underline{\underline{I}}}_{\text{local grad.}} \right) \right]_{1\times np}$$

(convert back to a matrix)

$$\nabla_{\underline{b}^i} \ell = \frac{\partial \ell}{\partial b^i} = g^i \, \text{diag} f' \, \mathbb{I}_{p\times p} \rightarrow \text{vector.}$$

4) Update the Parameters:

$$\left. \begin{aligned} \underline{\underline{W}}^{i(t)} &\Leftarrow \underline{\underline{W}}^{i(t-1)} - \alpha \nabla_{\underline{\underline{W}}^i} \ell^{(t-1)} \\ \underline{b}^{i(t)} &\leftarrow \underline{b}^{i(t-1)} - \alpha \nabla_{\underline{b}^i} \ell^{(t-1)} \end{aligned} \right\}$$

for each layer i's parameters

$\forall i = 1, \ldots, k$ layers.

for a k-layer NN.

(Variants of GD) SGD w/ Momentum:

Idea: Smooth out your gradients to speed up GD.

At iteration t: Compute $\nabla_\theta \mathcal{L} \rightarrow \underbrace{\nabla_{\underline{w}} \mathcal{L}}_{dw}$, $\underbrace{\nabla_b \mathcal{L}}_{db}$ on the current minibatch

Add Momentum: $dV_w = 0$
$dV_b = 0$

GD w/ momentum algo.:

$$dV_w^t = \beta_1 \cdot \underline{dV_w^{t-1}} + (1-\beta_1)\, \overbrace{dW^t}$$

$$dV_b^t = \beta_1 \cdot \underline{dV_b^{t-1}} + (1-\beta_1)\, db^t$$

↳ previous gradients

dw^t, db^t ⟸ current gradients at t

(β_1: a new hyperparameter)

eg. $\beta_1 = 0.9$

Now, update the parameters:

$$W^t = W^{t-1} - \alpha \cdot dV_w^t$$

$$b^t = b^{t-1} - \alpha\, dV_b^t$$

Note: A bias correction is added to affect initial iterations

divide by $\boxed{\dfrac{dV^t}{1-\beta_1^t}}$ ⟹ bias term

$t = 1, \rightarrow \dfrac{1}{1-(0.9)^1} \gg 1$

$\vdots$

$t = 10 \Rightarrow \dfrac{1}{1-(0.9)^{10}} \approx 1$

After a few iterations bias term has no effect

RMS_prop GD : At iteration t: Want to compute $\underline{\underline{dW}}$, $\underline{db}$:

Compute :

$$\underline{sdW} = \beta_2 \underline{SdW} + (1-\beta_2)(dW)^2 \leftarrow \text{current gradients}$$

$$sdb = \beta_2 \, sd\beta + (1-\beta_2)(db)^2$$

Update Rules:

$$\underline{\underline{W}} \leftarrow \underline{\underline{W}} - \alpha \frac{\underline{\underline{dW}}}{\sqrt{sdw+\varepsilon}} \leftarrow$$

like normalizing by std. dev. of the gradient.

$$b \leftarrow b - \alpha \frac{db}{\sqrt{sdb+\varepsilon}}$$

ADAM optimizer: Combination of Momentum & RMS prop:

Note: Use bias-corrected versions

$$dV_W^{corr} = \frac{dV_W}{(1-\beta_1^{\,t})}$$

Update

At iteration t:

$$W \leftarrow W - \alpha \frac{dV_W^{corr}}{\sqrt{S_{dW}^{corr}+\varepsilon}}$$

$$b \leftarrow b - \alpha \frac{dV_b^{corr}}{\sqrt{S_{db}^{corr}+\varepsilon}}$$

Initialize:

$$\left.\begin{aligned} dV_W &= 0 \\ dV_b &= 0 \\ dS_W &= 0 \\ dS_b &= 0 \end{aligned}\right\}$$

At each iteration t; Compute $\underbrace{dW, db}_{\text{current gradients}}$ w/ current minibatch

dV_W, dV_b, dS_W, dS_b