10) Metaparameters

Metaparam vs. Param

- parameter: feature learned by the model algorithm (ex. weight, bias)
- meta-parameter: feature u set for the model (ex. architecture, loss function)

Data Normalization

- Purpose: Ensure weights remain numerically stable: no gradient issues
  (data samples are in the same range w/o changing their relationship)

① Z-scoring: $z_i = \frac{x_i - \bar{x}}{\sigma_x}$ ← mean, ← stdev such that $\bar{z}=0, \sigma_z=1$ used for gaussian data

② Min-max scaling: $\tilde{x} = \frac{x - \min x}{\max x - \min x}$ ⇒ $x^* = a + \tilde{x}(b-a)$ used for bounded data

scale to $[0,1]$ scale to $[a,b]$ (optional)

Batch Normalization

- Not normalize inputs x_i but inputs to each layer this time
- Process: $y = \sigma(\hat{x}^T w)$, where $\hat{x} = \gamma x + \beta$ (γ, β: model param that scale stdev & shift mean)

y: input to next layer; $\hat{x}$: normalized input

- should be turned off during testing since batch_size differs

Activation Functions

- Add nonlinearities into layers: they won't collapse into 1 layer
- Common ones: ① Sigmoid (0, 0.5, 1) ② tanh (-1, 0, 1) ③ ReLU (0, x) Good with normalized inputs
- ReLU variants: ① Leaky ReLU (usually 0.01 → αx) ② ReLU-N (usually b) $\min(\max(0,x), N)$

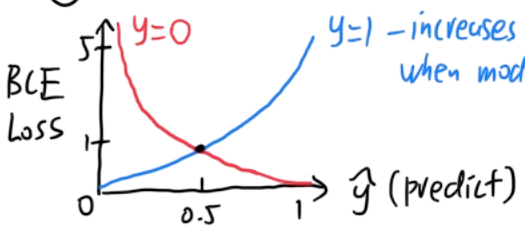
Loss Functions

- Metric to backpropgate (adjust weights)
- Common ones: ① BCE $L = -(y \log(\hat{y}) + (1-y)\log(1-\hat{y}))$

more general (BCE is when $N=1, C=2$)

② CCE $L = -\sum_{i=1}^{N}\sum_{k=1}^{C} y_i^{(k)} \log(\hat{y}_i^{(k)})$ (N: # data sample, C: # category)

③ MLE $L = \frac{1}{2}(\hat{y}-y)^2$



- Softmax variant: log-softmax $\sigma(x) = \log(e^z / \sum e^{z_i})$ is good for CCE
  penalizes errors more sensitively

Optimizers

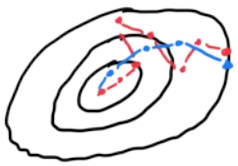
- algorithms that adjust weights during backprop

① Vanilla SGD: change weights after each sample (used when all samples are similar) } outliers point leads to volatile changes

② Mini-batch SGD: change weights after N samples, avg losses across them

③ Momentum SGD: change weights by a weighted avg of previous & current costs

- Formula: $V_t = (1-\beta)\, dJ + \beta V_{t-1}$, $W := W - \alpha V_t$
  $\beta \in [0,1]$ where $\beta = 0$ is vanilla SGD



Vanilla SGD
Momentum SGD

④ RMSprop: vanilla SGD that updates lr by total energy / rms(gradient)

- Formula: $rms = \sqrt{\frac{1}{m}\sum_{i=1}^{m} x_i^2}$, $S_t = (1-\beta)(dJ)^2 + \beta S_{t-1}$, $W := W - \frac{\alpha}{\sqrt{S_t}+\varepsilon} dJ$ ← $1E^{-8}$ (ensure den ≠ 0)

- Interpretation: Large gradients = volatile changes → slows down training
  Small gradients = vanishing changes → takes larger step

⑤ Adam (adaptive momentum): RMSprop + Momentum

- Formula: $V = (1-\beta_1)\, dJ + \beta_1 V_{t-1}$, $S = (1-\beta_2)(dJ)^2 + \beta_2 S_{t-1}$, $W := W - \frac{\alpha}{\sqrt{\hat{S}}+\varepsilon} \hat{V}$

  usually: $\beta_1 = 0.9$, $\beta_2 = 0.999$

  $\hat{S} = \frac{S}{1-\beta_2^t}$, $\hat{V} = \frac{V}{1-\beta_1^t}$, $\lim_{t\to\infty} \hat{V} = 0$

Schedulers

- Algorithms that decay lr over time

Ex) torch.optim.lr_scheduler.StepLR(optimizer, step_size, gamma)
step_size: interval to decay; gamma: applied factor when decay