1 Discussion 1: Bias-variance tradeoff

Overfit/underfit data. Model complexity. (See DATA 100.)

Notation

- Model parameters: θ
- Supervised dataset: X, Y

Sometimes, the data is denoted by $\mathcal{D}$.

1.1 MLE

Maximum Likelihood Estimation: find the model parameters that maximize the likelihood of observing the data.

$$\theta_{MLE} = \arg\max_\theta \prod_{i=1}^{N} P_\theta(y_i|x_i) \equiv \arg\max_\theta \sum_{i=1}^{N} P_\theta(y_i|x_i) \tag{1}$$

1.2 Maximum A Posteriori (MAP) estimation

MAP: find the model parameters that maximize the posterior probability of the parameters given the data (so, incorporates a prior distribution over the model parameters).

$$\begin{aligned} \theta_{MAP} &= \arg\max_\theta P(\theta|Y, X) \\ &= \arg\max_\theta P(Y|\theta, X)P(\theta) \\ &= \arg\max_\theta \log P(Y|\theta, X) + \log P(\theta) \end{aligned} \tag{2}$$

The term $\log P(\theta)$ is the log-prior, and serves as a regularizer.
Regularization is when we give a range to θ.

1.3 Gaussian prior and ℓ_2 regularization

A common choice for the prior is a Gaussian distribution: $P(\theta) = \mathcal{N}(\theta; 0, \sigma^2 I)$, where mean is 0, standard deviation is σ, I is an identity matrix.

The log-prior is then:

$$\begin{aligned} \log P(\theta) &= \log \left[\frac{1}{(2\pi)^{d/2}|\sigma^2 I|^{1/2}} \exp\left(-\frac{1}{2}\theta^T(\sigma^2 I)^{-1}\theta\right)\right] \\ &= -\frac{d}{2}\log(2\pi) - \frac{1}{2}\log(|\sigma^2 I|) - \frac{1}{2\sigma^2}\theta^T\theta \\ &= -\frac{1}{2\sigma^2}\|\theta\|_2^2 + \text{const.} \end{aligned} \tag{3}$$

where d is the dimensionality of the model parameters θ.

Thus, $\lambda\|\theta\|_2^2$ is the regularizer, where λ is the ℓ_2 regularization term.

2 Discussion 2: Optimization methods

Define α as learning rate (step size).

- Gradient descent: $\theta^{t+1} = \theta^t - \frac{\alpha}{|\mathcal{D}|} \sum_{x_i, y_i \in \mathcal{D}} \nabla_\theta L(x_i, y_i, \theta^t)$
- Stochastic GD with a random minimatch B^t for each iteration t:
  $\theta^{t+1} = \theta^t - \frac{\alpha}{|B^t|} \sum_{x_i, y_i \in \mathcal{D}} \nabla_\theta L(x_i, y_i, \theta^t)$ for some $B^t \subseteq \mathcal{D}$

Denote GD/SGD as: $\theta^{t+1} = \theta^t - \alpha \nabla L(\theta^t)$.

Since gradient descent performs poorly for non-convex problems and can get stuck in local minima, we introduce two techniques:

1. **SGD with momentum:** quicker progress towards the optimum along the horizontal axis, while not diverging along y-axis. Heavy ball method:

$$\begin{aligned} v^t &= mv^{t-1} + \nabla L(\theta^t) \\ \theta^{t+1} &= \theta^t - \alpha v^t \end{aligned} \tag{4}$$

   where m controls how much we remember the past gradients, v^t is our accumulated gradient vector.

2. **Adaptive learning:** rescaling different components of the gradient to get better direction.

$$\begin{aligned} \theta_k^{t+1} &= \theta_k^t - \frac{\alpha}{\sqrt{s_k^t + \epsilon}} \nabla_{\theta_k} L(\theta^t) \quad (\text{ update } k\text{th coordinate of parameters } \theta^t) \\ s_k^t &= \beta s_k^{t-1} + (1-\beta)(\nabla_{theta_k} L(\theta^t)^2) \quad \text{RMSProp: keep running average} \\ s_k^t &= s_k^{t-1} + \beta(\nabla_{theta_k} L(\theta^t)^2) \quad \text{Adagrad: keep sum } (s^t \text{ increases over time => stop learning}) \end{aligned} \tag{5}$$

   where a vector s^t tracks the "size" of the past gradients in each dimension.

3 Discussion 3: NN Building blocks

3.1 Affine layer forward/backward pass

$$\begin{aligned} \mathbf{z} &= \mathbf{x}W + b \\ \nabla_{\mathbf{x}} L &= \frac{\partial L}{\partial \mathbf{z}} \cdot \frac{\partial \mathbf{z}}{\partial \mathbf{x}} = \frac{\partial L}{\partial \mathbf{z}} W^T \\ \nabla_W L &= \frac{\partial L}{\partial \mathbf{z}} \cdot \frac{\partial \mathbf{z}}{\partial W} = X^T \frac{\partial L}{\partial \mathbf{z}} \\ \nabla_b L &= \sum_{i=1}^N \frac{\partial L}{\partial \mathbf{z}_i} \end{aligned} \tag{6}$$

where $\mathbf{z} \in \mathbb{R}^{N \times M}$, $\mathbf{x} \in \mathbb{R}^{N \times D}$ for number of samples N and dimensionality (number of features) D.
Note that weights $W \in \mathbb{R}^{D \times M}$, bias $b \in \mathbb{R}^M$.

3.2 ReLU

$$\text{ReLU}(x) = \max(0, x) \tag{7}$$

3.3 Batch normalization forward/backward pass

$$
\begin{aligned}
&\text{Var}(x) = \frac{1}{N}\sum_{i=1}^{N}(x_i-\mu)^2 \\
&x_{norm} = \frac{x-\mu}{\sqrt{\text{Var}(x)+\epsilon}} \quad \text{where std} = \sqrt{\text{Var}(x)+\epsilon} \\
&Y = \gamma \cdot x_{norm} + \beta \quad \text{scaled by } \gamma,\ \text{shifted by } \beta \\
&\nabla_\gamma L = \frac{\partial y_i}{\partial \gamma}\cdot\frac{\partial L}{\partial y_i} = \sum_{i=1}^{N}\frac{\partial L}{\partial y_i}\cdot x_{norm,i} \\
&\nabla_\beta L = \frac{\partial y_i}{\partial \beta}\cdot\frac{\partial L}{\partial y_i} = \sum_{i=1}^{N}\frac{\partial L}{\partial y_i} = \sum_{i=1}^{N}\delta_i \\
&\nabla_{x_{norm,i}} L = \frac{\partial y_i}{\partial x_{norm,i}}\cdot\frac{\partial L}{\partial y_i} = \frac{\partial L}{\partial y_i}\cdot\gamma \quad \text{and} \quad \nabla_{x_{norm}} L = \frac{\partial L}{\partial y}\cdot\gamma = \text{dx}_{norm} \\
&\nabla_x L = \frac{1}{N * \text{std}}\left(N\text{dx}_{norm} - \sum_{i=1}^{N}\text{dx}_{norm} - x_{norm}\cdot\sum_{i=1}^{N}(\text{dx}_{norm}\cdot x_{norm})\right)
\end{aligned}
\tag{8}
$$

4 Discussion 4: CNNs

Note: We denote $\star$ to be convolution operator with the filter on the right.

$$
\begin{aligned}
&Y = X \star w \quad \text{for input } X,\ \text{kernel } w \\
&\frac{\partial L}{\partial X} = \frac{\partial Y}{\partial X}\cdot\frac{\partial L}{\partial Y} = \left(\text{padded } \frac{\partial L}{\partial Y}\right)\star\overline{w} \\
&\frac{\partial L}{\partial w} = \frac{\partial Y}{\partial w}\cdot\frac{\partial L}{\partial Y} = X \star \frac{\partial L}{\partial Y} \\
&\delta_i = \frac{\partial L}{\partial y_i},\ \overline{w} = w \text{ rotated by 180 degrees}
\end{aligned}
\tag{9}
$$

Example: Assume $Y = X \star w \in \mathbb{R}^2$ for input $X \in \mathbb{R}^4$, kernel $w \in \mathbb{R}^3$. Alternatively, $y_{i,j,c'} = \sum_{h,w,c} x_{i-h,j-w,c}w_{h,w,c,c'}$.
Consider $\nabla_Y L = \begin{bmatrix}\frac{\partial L}{\partial y_1}\\ \frac{\partial L}{\partial y_2}\end{bmatrix}$.
Note that $Y = \begin{bmatrix}x_1w_1 + x_2w_2 + x_3w_3\\ x_2w_1 + x_3w_2 + x_4w_3\end{bmatrix}$.
Then gradient of loss w.r.t. X is:

$$
\nabla_X L = \begin{bmatrix}\frac{\partial y}{\partial x_1}\cdot\frac{\partial L}{\partial y}\\ \frac{\partial y}{\partial x_2}\cdot\frac{\partial L}{\partial y}\\ \frac{\partial y}{\partial x_3}\cdot\frac{\partial L}{\partial y}\\ \frac{\partial y}{\partial x_4}\cdot\frac{\partial L}{\partial y}\end{bmatrix} = \begin{bmatrix}w_1\cdot\frac{\partial L}{\partial y_1}\\ w_2\cdot\frac{\partial L}{\partial y_1} + w_1\frac{\partial L}{\partial y_2}\\ w_3\cdot\frac{\partial L}{\partial y_1} + w_2\frac{\partial L}{\partial y_2}\\ w_3\cdot\frac{\partial L}{\partial y_2}\end{bmatrix} = \begin{bmatrix}\frac{\partial L}{\partial x_1}\\ \frac{\partial L}{\partial x_2}\\ \frac{\partial L}{\partial x_3}\\ \frac{\partial L}{\partial x_4}\end{bmatrix} = \begin{bmatrix}0\\0\\ \nabla_Y L\\0\\0\end{bmatrix}\star\overline{w},
$$

where $\overline{w}$ is reversed filter w.
Similarly, the gradient of loss w.r.t. w is:

$$
\nabla_w L = \begin{bmatrix}x_1\cdot\frac{\partial L}{\partial y_1} + x_2\cdot\frac{\partial L}{\partial y_2}\\ x_2\cdot\frac{\partial L}{\partial y_1} + x_3\cdot\frac{\partial L}{\partial y_2}\\ x_3\cdot\frac{\partial L}{\partial y_1} + x_4\cdot\frac{\partial L}{\partial y_2}\end{bmatrix} = X \star \nabla_Y L.
$$

We use $\begin{bmatrix}\frac{\partial L}{\partial y_1}\\ \frac{\partial L}{\partial y_2}\end{bmatrix}$ as a 2×1 filter over 4×1 vector x.

5 Discussion 5: RNNs

5.1 Vanilla RNN

"Unroll" node at each time step.

$$\begin{aligned} h_t &= \tanh(W_{hh}h_{t-1} + W_{xh}x_t + b_h) \\ y_t &= W_{hy}h_t + b_y \end{aligned} \tag{10}$$

At different time steps, RNNs always use the same parameters. So, the parameterization cost does *not* grow as number of time steps increase.

5.2 Exploding/vanishing gradients

For RNNs, we effectively multiply h_t by W for t times, facing same problems with gradients as very deep feed forward networks. Gradients can explode (become infinitely large for large weight scales W) or vanish (reduce to 0 for too small weights W, providing no updates).

Techniques to address problem with gradients:

1. Gradient clipping: "clip" gradient vector by its magnitude to avoid vanishing gradients.
   $\nabla L \leftarrow \min(1, \frac{c}{\|\nabla L\|_2})\nabla L$
2. Truncation: terminate the sequence (fixed, random) to avoid exploding gradients.
   (Short-term dependencies matter more anyway.)
3. Dropout layers.
4. Residual connections. Consider Jacobian "through" the layer:
   $x_{i+1} = x_i + F(x_i)$, then $\frac{\partial x_{i+1}}{\partial x_i} = I + \frac{\partial F}{\partial x_i}$.
   Even if $\frac{\partial F}{\partial x_i}$ is close to 0, identity is closer to 1, so can stack and multiply!
5. Layer normalization.

5.3 Loss and backpropagation

Given h_t and o_t:

$$\begin{aligned} h_t &= f(x_t, h_{t-1}, w_h) \\ o_t &= g(h_t, w_o) \\ L(x_1, \dots, x_T, y_1, \dots, y_T, w_h, w_o) &= \frac{1}{T}\sum_{t=1}^{T} l(y_t, o_t) \end{aligned} \tag{11}$$

Backpropagation through time:

$$\frac{\partial L}{\partial w_h} = \frac{1}{T}\sum_{t=1}^{T} \frac{\partial l(y_t, o_t)}{\partial w_h} = \frac{1}{T}\sum_{t=1}^{T} \frac{\partial l(y_t, o_t)}{\partial o_t} \cdot \frac{\partial g(h_t, w_o)}{\partial h_t} \cdot \frac{\partial h_t}{\partial w_h}$$

$$\text{where} \quad \frac{\partial h_t}{\partial w_h} = \frac{\partial f(x_t, h_{t-1}, w_h)}{\partial w_h} + \frac{\partial f(x_t, h_{t-1}, w_h)}{\partial h_{t-1}} \cdot \frac{\partial h_{t-1}}{\partial w_h}$$

$$\text{specifically} \quad \frac{\partial h_t}{\partial w_h} = \frac{\partial f(x_t, h_{t-1}, w_h)}{\partial w_h} + \sum_{i=1}^{t-1} \left(\prod_{j=i+1}^{t} \frac{\partial f(x_j, h_{j-1}, w_h)}{\partial h_{j-1}} \right) \frac{\partial f(x_i, h_{i-1}, w_h)}{\partial w_h}$$

Do more reading! https://towardsdatascience.com/backpropagation-in-rnn-explained-bdf853b4e1c2

6 Discussion 6: Attention

6.1 Attention

$$\text{attention scores: } a_{ij} = \text{softmax}\left(\frac{q_i k_j^T}{\sqrt{d_k}}\right) = \frac{\exp(q_i k_j^T/\sqrt{s_k})}{\sum_r \exp(q_i k_r^T/\sqrt{s_k})}$$

$$a(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Given k input sequences of length M and ouput sequences of length N, the complexities are:

- encoder self-attention is $O(M^2k)$
- encoder-decoder attention is $O(MNk)$
- decoder self-attention is $O(N^2k)$

Multi-head attention
Loss