Stanford University

CS 329P : Practical Machine Learning (2021 Fall)

Deep Network Tuning

Qingqing Huang, Mu Li, Alex Smola

https://c.d2l.ai/stanford-cs329p

Deep Network Tuning

- DL is a programming language to extract information from data
  - Some values will be filled by data later
  - Differentiable
- Various design patterns, from layers to network architecture
- Here we talk about some of them

Stanford University

CS 329P : Practical Machine Learning (2021 Fall)

Batch and Layer Normalizations

Qingqing Huang, Mu Li, Alex Smola

https://c.d2l.ai/stanford-cs329p

Batch Normalization

- Standardizing data makes the loss smother for linear methods
  - Smooth: $||\nabla f(\mathbf{x}) - \nabla f(\mathbf{y})||^2 \leq \beta ||\mathbf{x} - \mathbf{y}||^2$
  - A smaller β allows a larger learning rate
  - Does not help deep NN
- Batch Normalization (BN) standards inputs for internal layers
  - Improves the smoothness to make training easier
  - (Still controversial why BN works)

Batch Normalization

- Reshape input $\mathbf{X}$ into 2D (no change for 2D input $\mathbf{X} \in \mathbb{R}^{n \times p}$)
  - $\mathbf{X} \in \mathbb{R}^{n \times c \times w \times h} \rightarrow \mathbf{X}' \in \mathbb{R}^{nwh \times c}$ (batch n, channel c, width w, height h)
- Normalize by standardization each column $\mathbf{x}'_j, j = 1,\ldots,p$
  - $\hat{\mathbf{x}}'_j \leftarrow (\mathbf{x}'_j - \text{mean}(\mathbf{x}'_j))/\text{std}(\mathbf{x}'_j)$
- Recovery $\mathbf{Y}'$ with $\mathbf{y}'_j = \gamma_j \hat{\mathbf{x}}_j + \beta_j$ as the j-th column, γ_j, β_j are parameters
- Output $\mathbf{Y}$ by reshaping $\mathbf{Y}'$ to the same shape as $\mathbf{X}$

Batch Normalization Code

```
def batch_norm(X, gamma, beta, moving_mean, moving_var, eps, momentum):
    if not torch.is_grad_enabled(): # In prediction mode
        X_hat = (X - moving_mean) / torch.sqrt(moving_var + eps)
    else:
        assert len(X.shape) in (2, 4)
        if len(X.shape) == 2:
            mean = X.mean(dim=0)
            var = ((X - mean)**2).mean(dim=0)
        else:
            mean = X.mean(dim=(0, 2, 3), keepdim=True)
            var = ((X - mean)**2).mean(dim=(0, 2, 3), keepdim=True)
        X_hat = (X - mean) / torch.sqrt(var + eps)
        moving_mean = momentum * moving_mean + (1.0 - momentum) * mean
        moving_var = momentum * moving_var + (1.0 - momentum) * var
    Y = gamma * X_hat + beta
    return Y, moving_mean, moving_var
```

Full code: http://d2l.ai/chapter_convolutional-modern/batch-norm.html

Layer Normalization

- If apply to RNN, BN needs maintain separated moving statistics for each time step
  - Problematic for very long sequences during inference
- Layer normalization reshapes input $\mathbf{X} \in \mathbb{R}^{n \times p} \to \mathbf{X}' \in \mathbb{R}^{p \times n}$ or $\mathbf{X} \in \mathbb{R}^{n \times c \times w \times h} \to \mathbf{X}' \in \mathbb{R}^{cwh \times n}$, rest is same with BN
  - Normalizing within each example, up to current time step
  - Consistent between training and inference
  - Popularized by Transformers

More Normalizations

- Modify "reshape", e.g.
  - InstanceNorm: $n \times c \times w \times h \rightarrow wh \times cn$
  - GroupNorm: $n \times c \times w \times h \rightarrow swh \times gn$ with $c = sg$
  - CrossNorm: swap mean/std between a pair of features
- Modify "normalize": e.g. whitening
- Modify "recovery": e.g. replace γ, β with a dense layer
- Apply to weights or gradients

Summary

- Normalizing inputs of internal layers makes deep NNs easier to train
- A normalization layer performs three steps: reshape input, normalize data, recovery with learnable parameters
    - Notable examples include Batch Normalization for CNNs, Layer Normalization for Transformers