BatchNorm

A Preprint

1 Intro

SGD(2013) optimizes model parameters θ : $\theta = argmin\frac{1}{N}\sum_{i=1}^{N} l(x_i;\theta)$. The mini-batch is used to approximate the gradient of the loss function with respect to the parameters, by computing $\frac{1}{m} \cdot \frac{\delta l(x_i;\theta)}{\delta\theta}$.

2 main

Batch Normalization, that takes a step towards reducing internal covariate shift, and in doing so dramatically accelerates the training of deep neural nets. It accomplishes this via a normalization step that fixes the means and variances of layer inputs. Batch Normalization also has a beneficial effect on the gradient flow through the network, by reducing the dependence of gradients on the scale of the parameters or of their initial values. normalize each scalar feature independently, by making it have the mean of zero and the variance of 1. For d dimensional input: $x^{\hat{(k)}} = \frac{x^{(k)} - E[x^{(k)}]}{\sqrt{VAR[x^{(k)}]}}$

But if you take the inputs to a layer, say a sigmoid activation function and you normalize them to have a mean of 0 and a standard deviation of 1, most of these input values will be close to zero. The sigmoid function has a characteristic "S"shape. Around an input of 0, the sigmoid function is almost linear (it's not flat, nor sharply curved). If you always feed values in this narrow, linear range to the sigmoid, you are not utilizing its full non-linear capabilities. The sigmoid function's non-linearity is crucial for learning complex patterns, and this is most pronounced when inputs can take a wider range of values, pushing them into the "curved"or even "saturated"parts of the function. So, just normalizing could limit what the layer can learn by effectively constraining its inputs to the linear part of the sigmoid's activation. For solving this problem authors introduce for each activation $x^{(k)}$: $\gamma^{(k)}$: a scaling factor, $\beta^{(k)}$: a shifting factor(learnable parameters). Result: $y^{(k)} = \gamma^{(k)} x^{\hat{(k)}} + \beta^{(k)}$

Batch Normalization

Given a mini-batch of inputs:

$$B = \{x_1, x_2, \ldots, x_m\}$$

Let γ and β be the learnable parameters. The batch normalization procedure computes:

$$\mu_B \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \quad \text{// mini-batch mean}$$

$$\sigma_B^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_B)^2 \quad \text{// mini-batch variance}$$

$$\hat{x}_i \leftarrow \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \quad \text{// normalize} \quad y_i \qquad \leftarrow \gamma \hat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \quad \text{// scale and shift}$$

Learning γ and β

The parameters γ and β are learnable affine transformation parameters introduced after normalization. They allow the model to recover the representational power that might be lost due to normalization.

During training, γ and β are updated via backpropagation along with the rest of the model parameters, using gradients from the loss function $\mathcal{L}$:

$$\frac{\partial \mathcal{L}}{\partial \gamma}, \quad \frac{\partial \mathcal{L}}{\partial \beta}$$

These updates allow the network to optimally re-scale and re-shift the normalized outputs, adapting the batch normalization to the learning task.