Regularization and Optimization

Aziz Temirkhanov

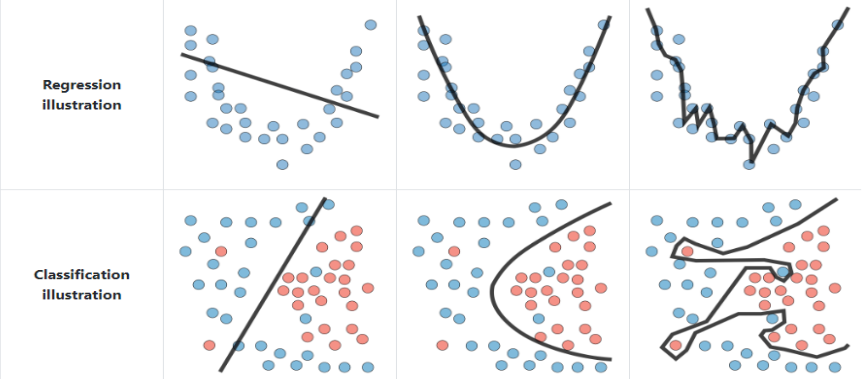
LAMBDA lab
Faculty of Computer Science
Higher School of Economics

March 4, 2024

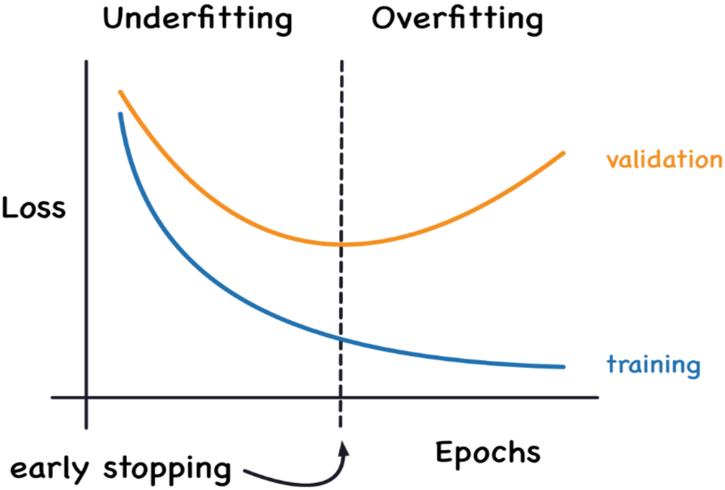
Outline

Overfitting

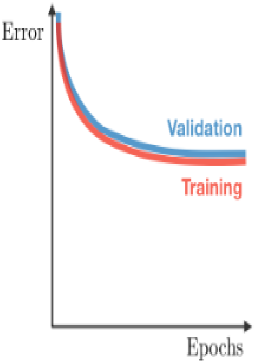
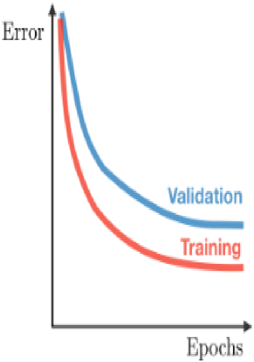
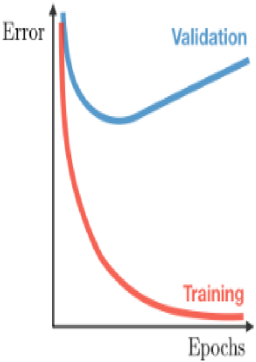
Overfitting



Overfitting



Overfitting

		
• Complexify model • Add more features • Train longer		• Perform regularization • Get more data

Overfitting

Possible remedies:

- Add regularizing term to the loss function, e.g. penalize for complex parameters
- Introduce random noise to the input data
- Introduce random noise to the labels
- Acquire more data samples
- Change the network topology
- ...

Dropot

Dropout

Let $z \in \mathbb{R}^{B \times N}$, B – batch size, N – hidden size be a batch of latent representations, i.e. logit of some intermediate layer.

Sample a binary mask $m \in \{0, 1\}^{B \times N}$ $m_{ij} \sim Bernoulli(1 - p)$, where p - dropout rate, typically $0.1 \leq p \leq 0.5$

Dropout

Train mode:

1. Sample new mask m
2. $y = m \odot z \times \frac{1}{1-p}$.

- Normalization by $\frac{1}{1-p}$ is required to preserve the expectation of neurons:

$$\mathbb{E}[y] = \mathbb{E}[m \odot z \cdot \frac{1}{1-p}] = \frac{1}{1-p} \cdot z \odot \mathbb{E}[m] = \frac{1}{1-p} \cdot z \cdot (1-p) = z \quad (1)$$

- The mask needs to be stored for backward pass

Dropout

Eval mode:

- As we normalized the output during training, it is an identical transform during testing:

$$y = z$$

- Dropout makes training slower but improves test quality

Batch Normalization

Batch Normalization

Main idea:

stabilize training by forcing zero mean and unit variance of features

Let:

$$X \in \mathbb{R}^{B \times N} = (x_1^{(N)}, x_2^{(N)}, ..., x_B^{(N)})^T$$

Train mode:

1. Compute batch statistics:

$$\mu = \frac{1}{B}\sum_{i=1}^{B} x_i, \qquad \sigma^2 = \frac{1}{B}\sum_{i=1}^{B}(x_i - \mu)^2 \tag{2}$$

$$\mu, \sigma^2 \in \mathbb{R}^N$$

2. Normalize x, recall that every operation is element-wise:

$$\hat{x}_i = \frac{x_i - \mu}{\sqrt{\sigma^2 \epsilon}} \tag{3}$$

Batch Normalization

All operations are differentiable, so we need to compute

$$\frac{d\ell}{d\hat{x}_i}, \frac{d\ell}{d\mu}, \frac{d\ell}{d\sigma^2}, \frac{d\ell}{dx_i}$$

during backward pass
Then, apply trainable element-wise affine transform:

$$y_i = \hat{x}_i \odot w + b; \quad w, b \in \mathbb{R}^N \tag{4}$$

We give the model freedom to set the desired mean and variance for x_i

Batch Normalization

During the test, we do not have access to a batch of data.
Thus, we will accumulate statistics from the training batches:

$$\begin{aligned} running\_mean &= running\_mean \cdot (1 - m) + \mu \cdot m \\ running\_var &= running\_var \cdot (1 - m) + \sigma^2 \cdot m \cdot \frac{B}{B-1} \end{aligned} \tag{5}$$

Here, m – is a momentum parameter (usually $m = 0.1$ to ensure slow updates).
In the beginning, *running_mean* is initialized with a zero-valued vector and $running_var$ as ones. These vectors are not considered trainable.

Batch Normalization

1. Normalizing using accumulated statistics:

$$\hat{x}_i = \frac{x_i - running\_mean}{\sqrt{running\_var + \epsilon}} \tag{6}$$

2. Apply affine transform:

$$y_i = \hat{x}_i \odot w + b \tag{7}$$

Optimization

SGD

- $\{x_i, y_i\}_{i=1}^{\ell}$ – training set
- B – batch size
- $\mathcal{L}(\cdot, \cdot)$ – loss function
- W – all trainable parameters of the model
- $f_W(\cdot)$ – neural network

At the step t we have:

$$\mathcal{L}^t(w) = \frac{1}{B} \sum_{i=1}^{B} \mathbf{L}(f_W(x_i), y_i), \quad t_i \sim \mathcal{U}(\{1, ..., \ell\}) \tag{8}$$

SGD

SGD algorithm:

1. $g_t = \nabla_w \mathcal{L}^t(w_{t-1})$
2. $m_t = \mu m_{t-1} + g_t$
3. $w_t = w_{t-1} - \eta_t m_t$

$m_0 = 0$, $0 \le \mu \le 1$, usually $\mu = 0.9$

SGD with Regularization

We can add L^2 **regularization**:

$$\mathcal{L}^t(w) \mapsto \mathcal{L}^t_\lambda(w) := \mathcal{L}^t(w) + \frac{\lambda}{2}\|w\|_2^2 \tag{9}$$

$$\nabla_w \|w\|_2^2 = 2w$$

1. $g_t = \nabla_w \mathcal{L}^t(w_{t-1}) + \lambda w_{t-1}$
2. $m_t = \mu m_{t-1} + g_t$
3. $w_t = w_{t-1} - \eta_t m_t$

It is essentially a weight decaying by a factor of less than 1:

$$w_t = w_{t-1}(1 - \eta_t \lambda) - \eta_t(\mu m_{t-1} + \nabla_w \mathcal{L}^t(w_{t-1}))$$

Adam

- Adaptive learning rate (AdaFrad, RMSProp)
- Momentum

1. $g_t = \nabla_w \mathcal{L}^t(w_{t-1}) + \lambda w_{t-1}$
2. $m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$
3. $v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$ – element-wise square
4. $\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1-\beta_2^t}$
5. $w_t = w_{t-1} - \eta_t \frac{\hat{m}_t}{\hat{v}_t + \epsilon}$

$$m_0 = 0, v_0 = 0, \beta_1, \beta_2, \epsilon \text{ – hyperparams}$$

$$\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}$$

AdamW

1. $g_t = \nabla_w \mathcal{L}^t(w_{t-1})$
2. $m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$
3. $v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$ – element-wise square
4. $\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1-\beta_2^t}$
5. $w_t = w_{t-1}(1 - \eta_t \lambda) - \eta_t \frac{\hat{m}_t}{\hat{v}_t + \epsilon}$

LR Schedulers

It is often a non-trivial task to choose a learning rate η_t
You can update it once per training epoch

- Constant. $\eta_t = \eta_0$. Often gives suboptimal quality
- StepRL. $\forall t \in \{t_0, ...t_n\} : \eta_{cur} = \eta_t, \eta_0 > \eta_1 > ...$
- ExponentialLR. $\eta_t = \gamma\eta_{t-1}, \gamma < 1$
- CosineLR. $\eta_t = \eta_0 \cdot \frac{1}{2}(1 + \cos\left(\frac{\pi T}{T}\right)),\ T$ – maximum number of epochs
- Linear warmup LR. Linearly increase lr first n-batches $\eta_0^{(0)} \mapsto \eta_0^{(n)}$. Then apply any scheduler.
- Reduce LR on a plateau. Reduce LR when validation loss stagnates, thus requiring have validation set


Thank
you