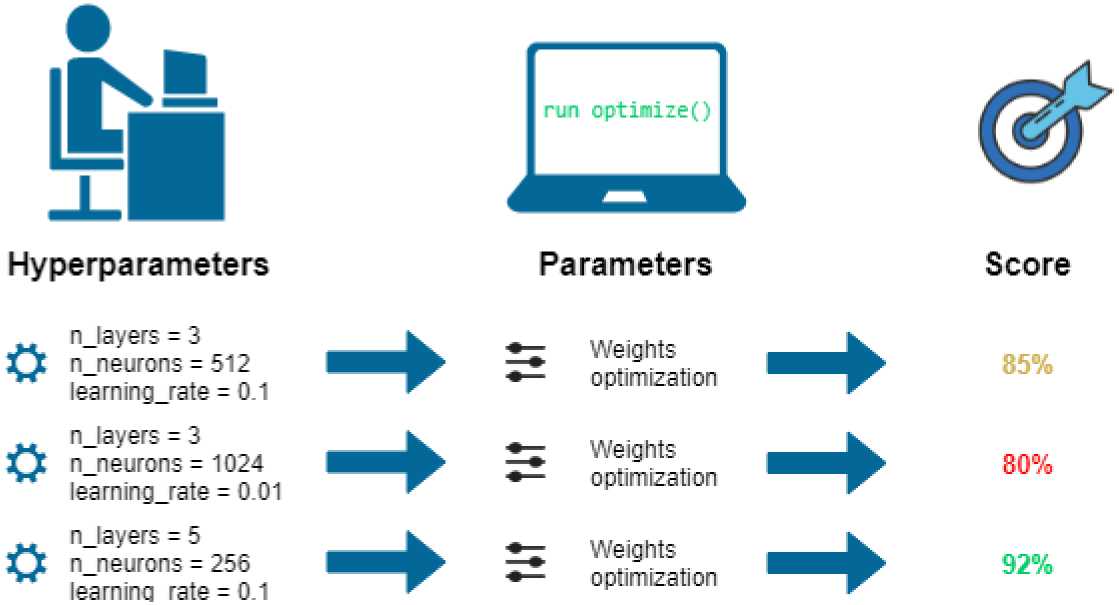
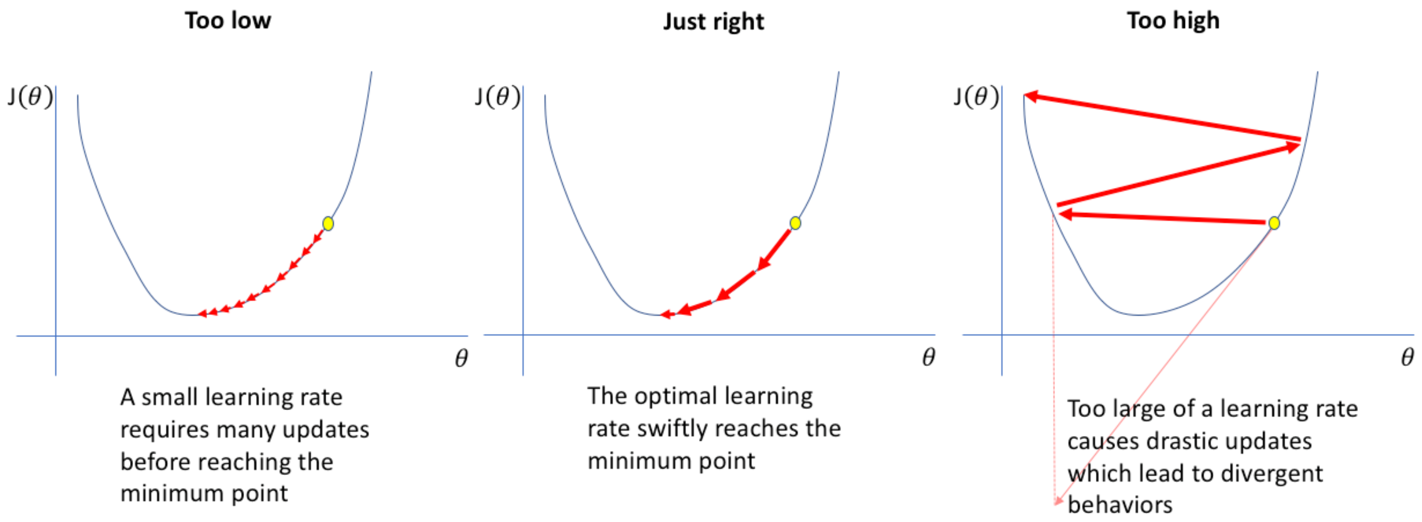
Hyperparameters



Learning rate

- The right learning rate η_t very important for fast convergence
  - Too strong → gradients overshoot and bounce
  - Too weak → slow training
- Learning rate per weight is often advantageous
  - Some weights are near convergence, others not



Convergence

- The step sizes **theoretically** should satisfy the following [Robbins–Monro]

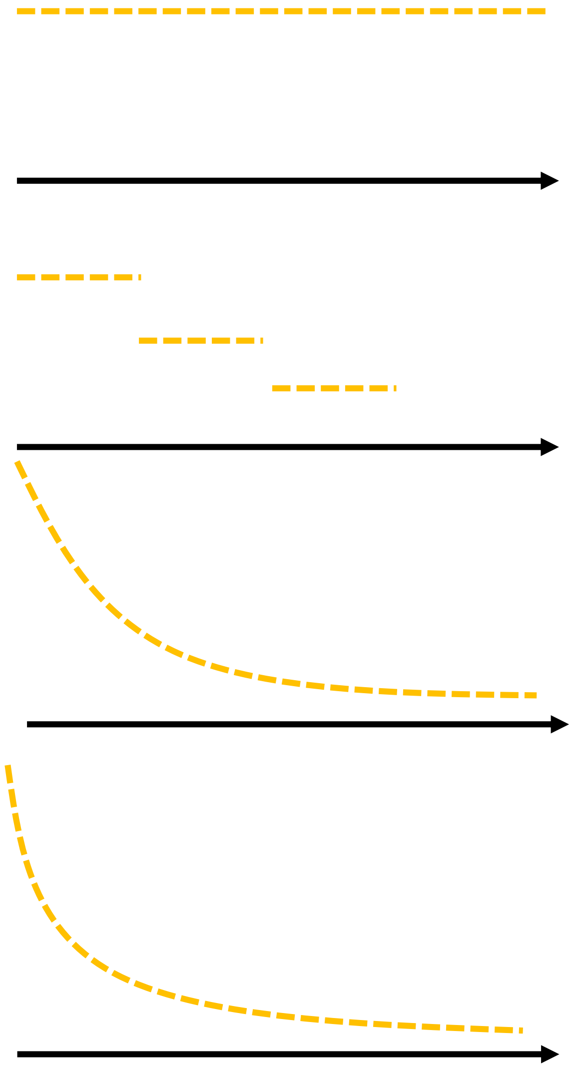
$$\sum_t^{\infty} \eta_t = \infty \quad \text{and} \quad \sum_t^{\infty} \eta_t^2 < \infty$$

- Intuitively,
  - The first term ensures that search will explore enough
  - The second term ensures convergence

Learning rate schedules

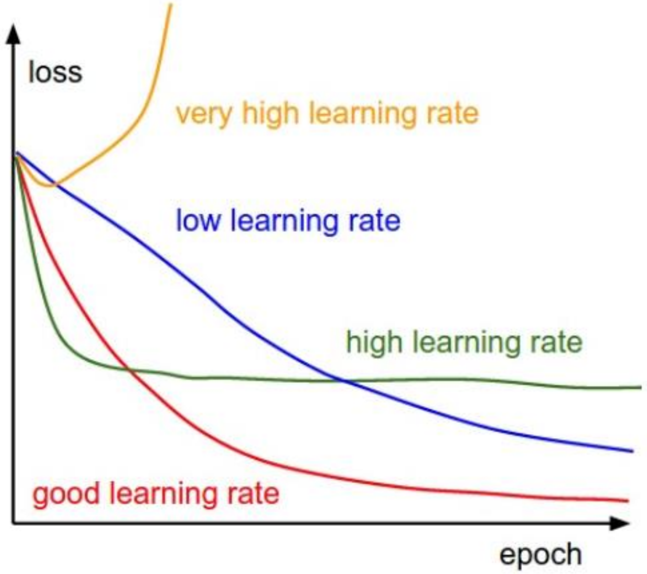
- Constant
  - Learning rate remains the same for all epochs
- Step decay
  - Decrease every T number of epochs or when validation loss stopped decreasing
- Inverse decay $\eta_t = \frac{\eta_0}{1+\varepsilon t}$
- Exponential decay $\eta_t = \eta_0 e^{-\varepsilon t}$

- Often step decay preferred
  - simple, intuitive, works well



In practice

- Try several log-spaced values $10^{-1}, 10^{-2}, 10^{-3}, \ldots$ on a smaller set
  - Then, you can narrow it down from there around where you get the lowest **validation** error
- You can decrease the learning rate every T (e.g., 10) training set epochs
  - Although this highly depends on your data



Picture credit: [Stanford Course]

Dropout rate

- Start with a relatively small rate, like 20-50%
  - If too high, your network will underfit
- With dropout you can also try larger neural networks

Batch size

- If possible, start with at least 32
- Generally, as big as your GPU memory fits

Number of layers and neurons

- For a new problem, generally start from moderate sizes
  - 3-5 layers
  - A few dozens neurons at most
  - When things check out, start increasing complexity
- For a known problem, *e.g.*, image classification, reuse hyperparameters
  - The one suggested by the model of choice are usually decent

Babysitting Deep Nets

- Always check your gradients if not computed automatically
- Check that in the first round you get loss that corresponds to random guess
- Check network with few samples
  - Turn off regularization. You should predictably overfit and get a loss of 0
  - Turn on regularization. The loss should be higher than before
- **Always** a separate validation set for hyper-parameter tuning
  - Compare the training and validation losses - there should be a gap, not too large
- Preprocess the data (at least to have 0 mean)
- Initialize weights based on activations functions Xavier or Kaiming initialization
- Use regularization (ℓ_2-regularization, dropout, ...)
- Use batch normalization
- Prefer residual connections, they make a difference

Reading material

- Deep Learning Book: Chapter 8, 11
- Efficient Backprop
- How Does Batch Normalization Help Optimization?
- https://medium.com/paperspace/intro-to-optimization-in-deep-learning-momentum-rmsprop-and-adam-8335f15fdee2
- http://ruder.io/optimizing-gradient-descent/
- https://github.com/Jaewan-Yun/optimizer-visualization
- https://blog.paperspace.com/intro-to-optimization-in-deep-learning-gradient-descent/

Summary

- Advanced optimizers
- Initialization
- Normalization
- Regularization
- Hyperparameters

Reading material

- Chapter 8, 11
- And the papers mentioned in the slide