

1. The Memory Cost of an Optimizer

Training a network with N parameters requires the GPU to hold more than the weights $\theta \in \mathbb{R}^N$. At every step t , backpropagation produces a gradient $g_t = \nabla f_t(\theta_{t-1}) \in \mathbb{R}^N$, and the optimizer may need to remember extra vectors from one step to the next. Those vectors are the *optimizer state*, and each of them is as large as θ itself. In this problem we count these vectors and turn the count into gigabytes. We ignore activations, which scale with batch size and sequence length rather than directly with N (and can be traded for compute by recomputing them), and count only what scales with N . Every number is stored in fp32, which takes 4 bytes, and $1 \text{ GB} = 10^9$ bytes.

- (a) **Fill in the table below.** For each optimizer, write the update rule, list the vectors in $\mathbb{R}^N$ that must be carried in memory from step $t-1$ to step t (the optimizer state), and count the numbers per parameter that the GPU holds during a step, including θ and g_t but not temporaries such as g_t^2 . Use η for the learning rate and β_1, β_2 for the averaging constants.

(Hint: momentum keeps an exponential running average m of the gradient and steps along it; RMSPprop keeps a running average v of the squared gradient and divides the step by $\sqrt{v}$; Adam does both.)

Optimizer	Update rule	Optimizer state	Numbers per parameter
Gradient descent			
SGD			
SGD with momentum			
RMSProp	$v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$, $\theta_t \leftarrow \theta_{t-1} - \eta g_t / \sqrt{v_t}$	v	3
Adam			

- (b) A 7-billion-parameter model ($N = 7 \times 10^9$) is trained with every vector stored in fp32. **How many GB does each optimizer need for θ , g_t , and its state? Which of them fit on a single GPU with an 80 GB memory budget?** Remember that the activations still have to fit on top of this.

2. Weight Decay and Adam

Because Adam can use different learning rates for different weights, it can have different implicit regularization behavior than traditional SGD or SGD with momentum. Consequently, it can interact differently when combined with other approaches for explicit regularization — even if those other approaches are equivalent with traditional SGD.

Two such regularization approaches are putting an L2 penalty on weights in the loss function itself (in the style of explicit ridge regularization) and alternatively, adding a weight-decay term during learning updates. For a simplified variant of Adam (without momentum), we have:

- **L2 regularization** can be expressed as: $\theta_{t+1} \leftarrow \theta_t - \alpha \mathbf{M}_t (\nabla f_t(\theta_t) + \lambda_w \theta_t)$
- **Weight decay** can be expressed as: $\theta_{t+1} \leftarrow (1 - \lambda_r) \theta_t - \alpha \mathbf{M}_t \nabla f_t(\theta_t)$

Here $\mathbf{M}_t$ denotes the adaptive term in the Adam update step — think of this as a diagonal matrix.

- (a) **When will the updates for L2 regularization and weight decay be identical?**

(Hint: your answer should involve things like α , λ_w , λ_r , and $\mathbf{M}_t$ in some way.)

- (b) **Where should you add $\lambda_r \theta_{t-1}$ to the Adam code below to get explicit weight decay?**

Algorithm 1 Adam Optimizer (without bias correction)

```

1: Given  $\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999$ 
2: Initialize time step  $t \leftarrow 0$ , parameter  $\theta_{t=0} \in \mathbb{R}^n$ ,  $m_{t=0} \leftarrow \mathbf{0}$ ,  $v_{t=0} \leftarrow \mathbf{0}$ 
3: Repeat
4:    $t \leftarrow t + 1$ 
5:    $\nabla f_t(\theta_{t-1}) \leftarrow \text{SelectBatch}(\theta_{t-1})$ 
6:    $g_t \leftarrow \nabla f_t(\theta_{t-1}) + \text{---(A)---}$ 
7:    $m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1)(g_t + \text{---(B)---})$ 
8:    $v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2)g_t^2 + \text{---(C)---}$ 
9:    $\theta_t \leftarrow \theta_{t-1} - (\alpha \cdot \frac{m_t}{\sqrt{v_t}} + \text{---(D)---})$ 
10: Until the stopping condition is met

```

- ☐ **A** on line 6 above
☐ **B** on line 7 above
☐ **C** on line 8 above
☐ **D** on line 9 above

3. Coding: ReLU Networks with Different Optimizers

This notebook trains the same one-hidden-layer ReLU network as in the elbows notebook, but with three optimizers: SGD, SGD with momentum, and Adam, across several widths and random seeds. The choice of optimizer changes how well the fit goes, and it matters most when the network is small.

Work through `q_relu_optim.ipynb`. Run all of the active cells. Leave the optional training cells commented out; the notebook loads pretrained checkpoints, so you do not have to retrain anything. Read the “Bug Hunt” cell without executing it. Then answer the questions below.

- (a) **How do the hidden layer width and the choice of optimizer affect the learned function and the test error?** Say something about the error bars as well as the medians.

- (b) **Compare the first and last saved checkpoints for the example run. How do the elbow locations change relative to the target’s kinks under each optimizer?**

- (c) The notebook shows a training loop with a bug in it (the “Bug Hunt” cell). **Identify the bug and explain why it can make training unstable. If the first two batch gradients are g_1 and g_2 , what is in each parameter’s `.grad` field after the second `loss.backward()` call? Does `optimizer.step()` clear `.grad`? Write the corrected loop.**

Contributors:

- Matteo Guarrera.
- Mert Cemri.
- Sizhe Chen.
- Gireeja Ranade.

- Kevin Li.
- Anant Sahai.
- Kumar Krishna Agrawal.
- Naman Jain.