Speeding Up and Reducing Memory for Scientific Machine Learning via Mixed Precision

Eric Wang
Mentor: Prof. Lu Lu

West Windsor-Plainsboro High School North

October 14, 2023
MIT PRIMES Conference

Scientific machine learning for partial differential equations

Scientific machine learning combines machine learning models with physics knowledge in a variety of applications, such as predicting fluid flow. Many problems can be described well with partial differential equations (PDEs)

Example (Kovasznay Flow)

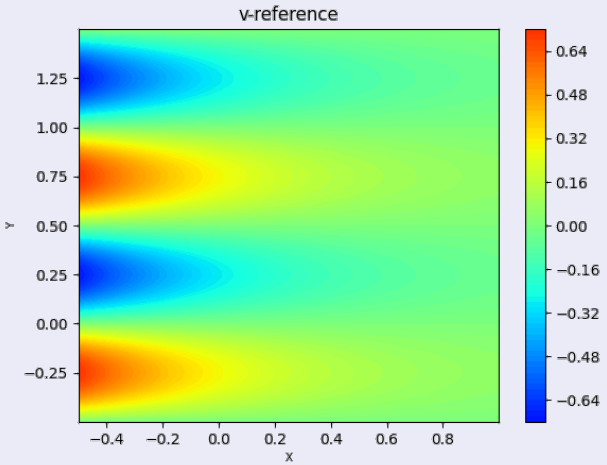
Navier-Stokes Equations

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0$$

$$u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} = -\frac{\partial p}{\partial x} + \frac{1}{Re}\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right)$$

$$u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} = -\frac{\partial p}{\partial y} + \frac{1}{Re}\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right)$$

Possible solution for v



PDE Problems can be very complicated and expensive to solve in general.

Deep neural networks

A deep neural network is a specific type of function with a set of parameters θ called *weights*. Formally, we can define a deep neural network to be of the form

$$\mathcal{N}(x;\theta) = T^L \circ (\sigma \circ T^{L-1}) \circ \cdots \circ (\sigma \circ T^1)(x)$$

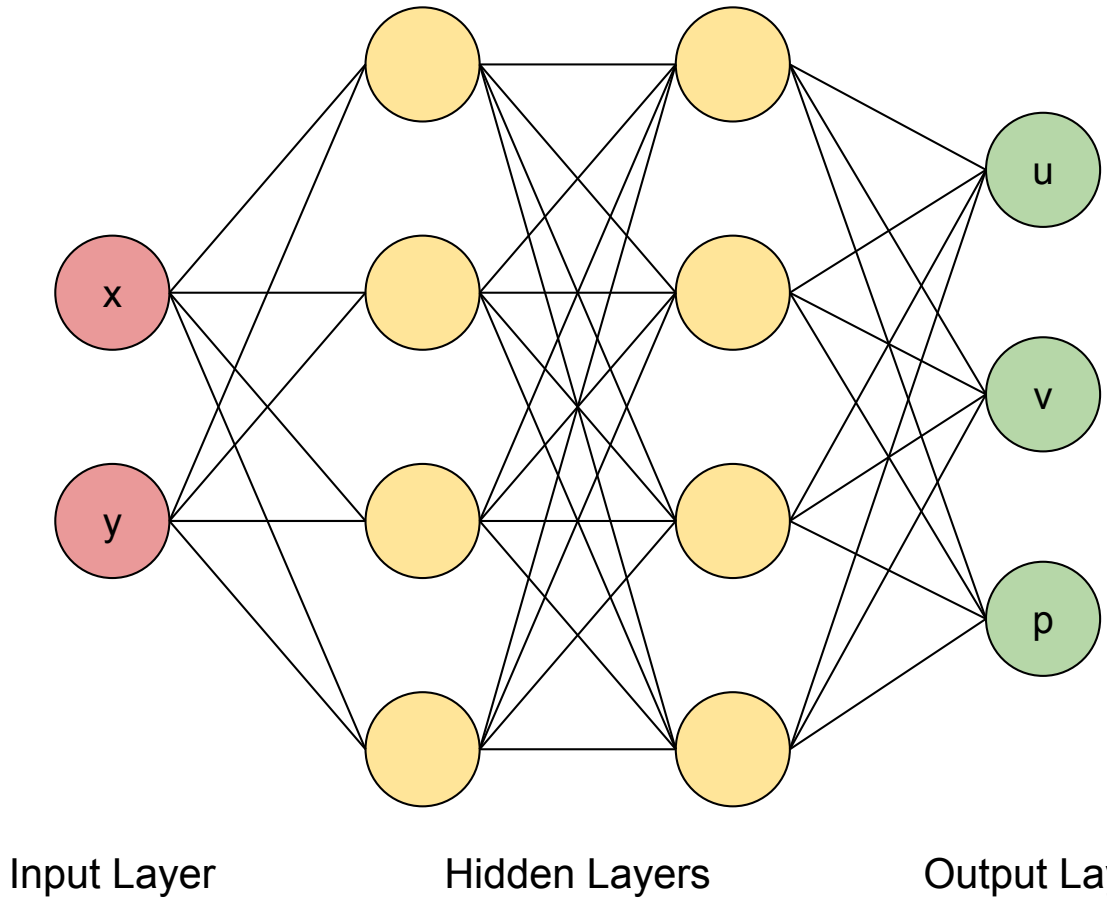
composed of layers

$$(\sigma \circ T^\ell)(x) = \sigma(\mathbf{W}^\ell x + \mathbf{b}^\ell)$$

the nonlinear function σ is called the *activation function*.

Deep neural networks for PDEs

A deep neural network can be trained to learn almost any function, including the solution to a PDE. Here is an example of a network that might be used to learn a fluid flow problem.

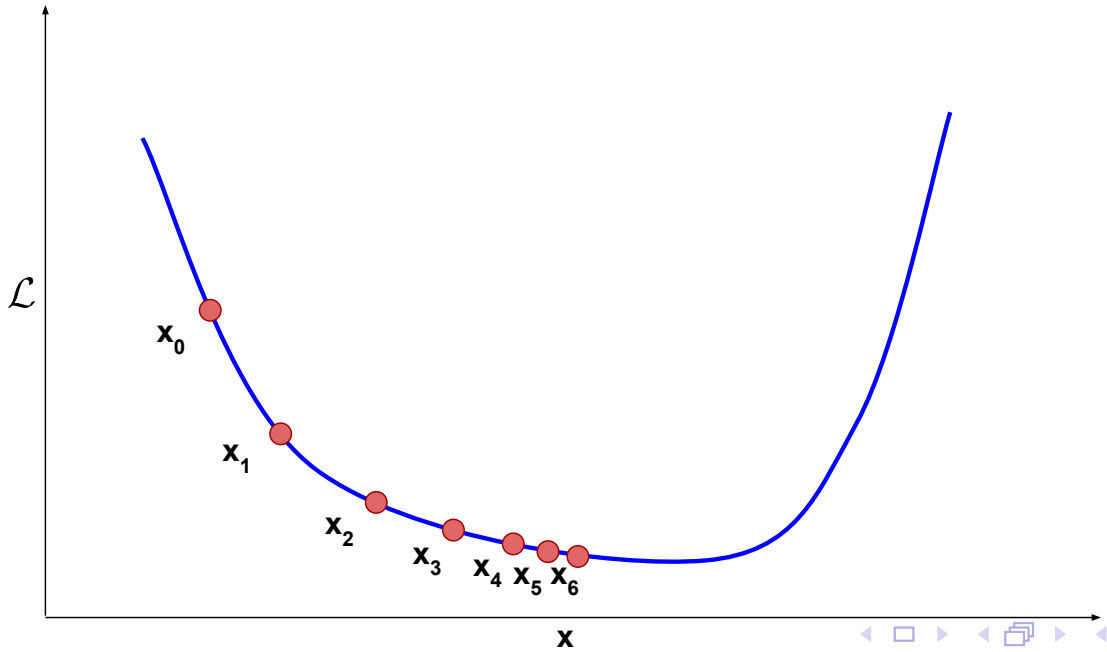


Loss function and gradient descent

All neural networks are trained with a **loss function**. The loss function $\mathcal{L}$ aggregates the total error of the neural network into a real number. Algorithms for minimizing the loss function are based on **gradient descent**.

Gradient descent formula

$$x_{t+1} = x_t - \eta \frac{\partial \mathcal{L}}{\partial x}$$



PDE loss

Navier-Stokes equations in 2 dimensions

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0$$

$$u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} + \frac{\partial p}{\partial x} - \frac{1}{Re}\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) = 0$$

$$u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} + \frac{\partial p}{\partial y} - \frac{1}{Re}\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right) = 0$$

Recall that we want the output of the neural network to satisfy these differential equations. How can we encode this information into the loss function?

PDE loss

Navier-Stokes equations in 2 dimensions (continuity equation)

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0$$

The most straightforward way to enforce this equation is to add the term

$$\sum_{i=1}^{N} \left[\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} \right]^2_{(x,y)=(x_i,y_i)}$$

for some observation points (x_i, y_i) scattered throughout the domain. Minimizing this loss effectively enforces the differential equation.

Physics-informed neural network

Formally, a *Physics-informed neural network* is a type of deep neural network with a special loss function:

$$\mathcal{L}(\theta; T) = \mathcal{L}_{\text{ic}}(\theta; T_{\text{ic}}) + \mathcal{L}_{\text{bc}}(\theta; T_{\text{bc}}) + \mathcal{L}_{\text{f}}(\theta; T_{\text{f}})$$

where

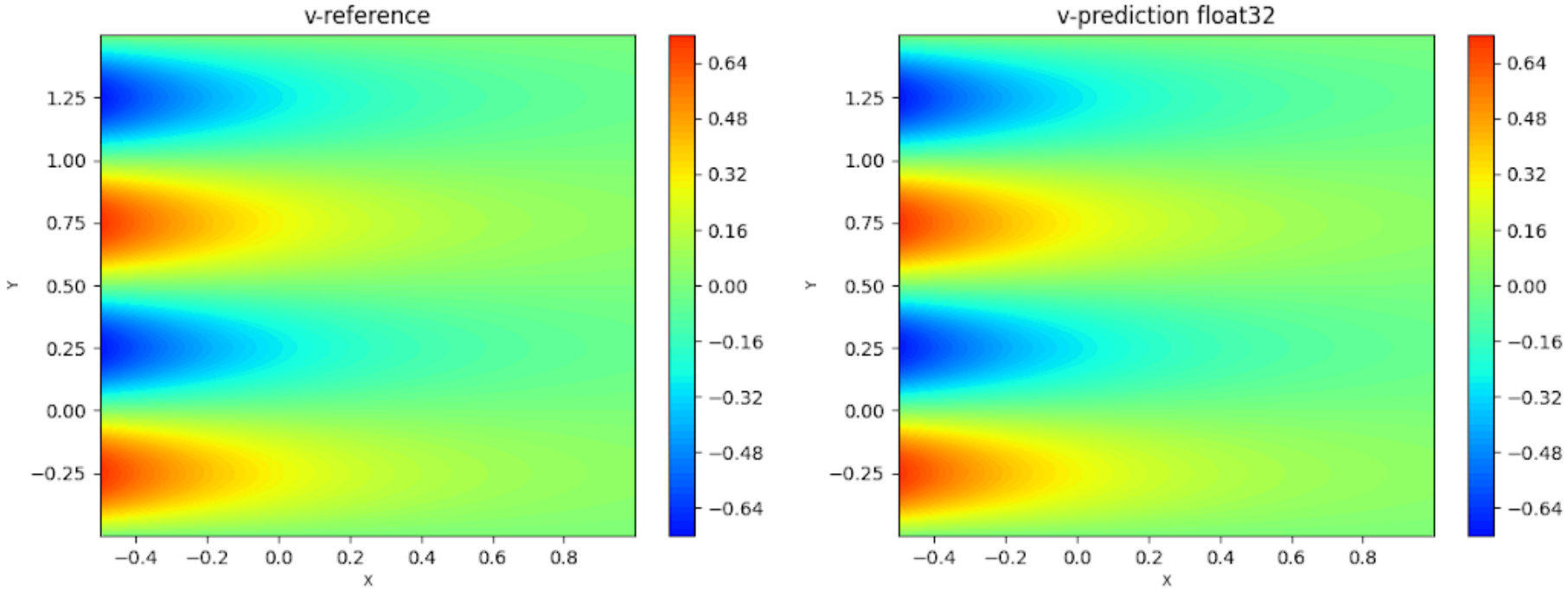
$$\mathcal{L}_{\text{ic}}(\theta; T_{\text{ic}}) = \frac{1}{|T_{\text{ic}}|} \sum_{\mathbf{x} \in T_{\text{ic}}} \|\hat{u}(\theta, \mathbf{x}) - u(\mathbf{x})\|_2^2 \qquad \text{(initial conditions)}$$

$$\mathcal{L}_{\text{bc}}(\theta; T_{\text{bc}}) = \frac{1}{|T_{\text{bc}}|} \sum_{\mathbf{x} \in T_{\text{bc}}} \|\mathcal{B}[\hat{u}(\mathbf{x}; \theta)]\|_2^2 \qquad \text{(boundary conditions)}$$

$$\mathcal{L}_{\text{f}}(\theta; T_{\text{f}}) = \frac{1}{|T_{\text{f}}|} \sum_{\mathbf{x} \in T_{\text{f}}} \|f[\hat{u}(\mathbf{x}; \theta); \lambda]\|_2^2 \qquad \text{(PDE loss)}$$

High accuracy of solutions

Implementing this problem in the DeepXDE library, we can see that the PINN is able to learn the PDE.



Low precision training

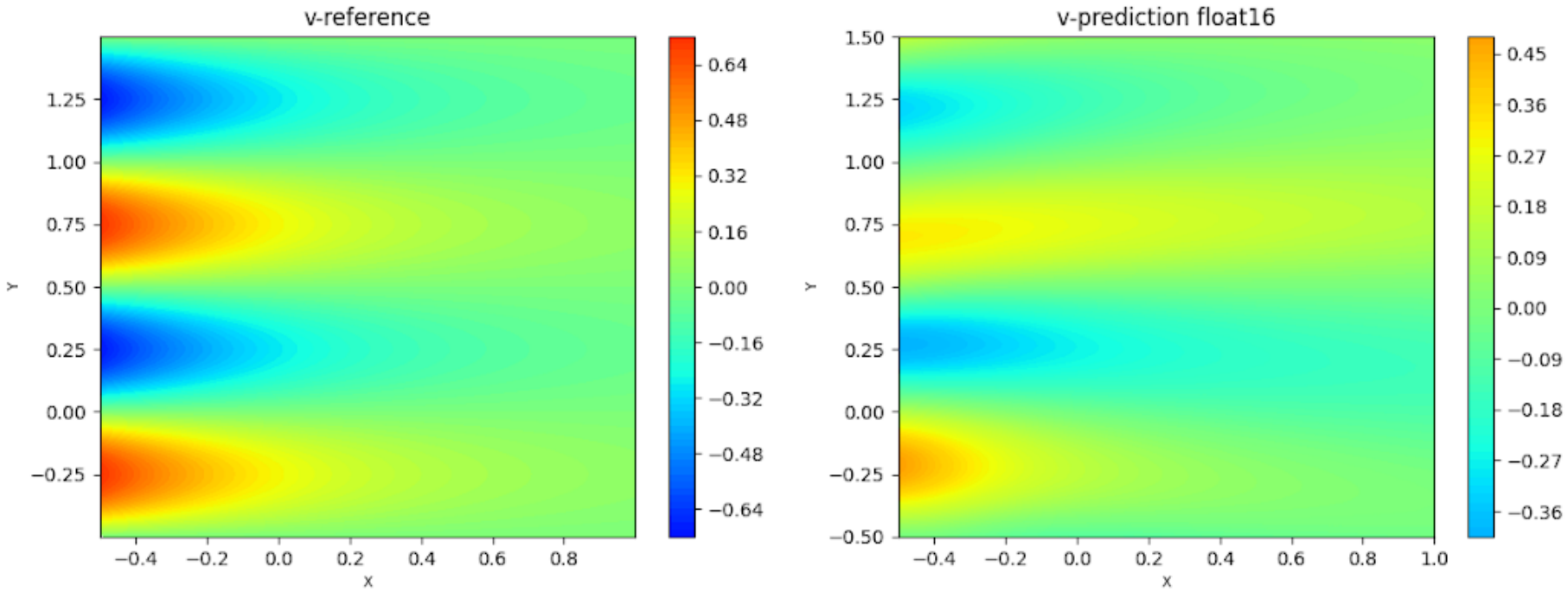
Even with deep learning, solving PDE problems can be slow and expensive. Modern GPUs support half precision (float16) data types. In the context of scientific machine learning, these data types

- are faster (good)
- take up less memory (good)
- are less accurate (not good)

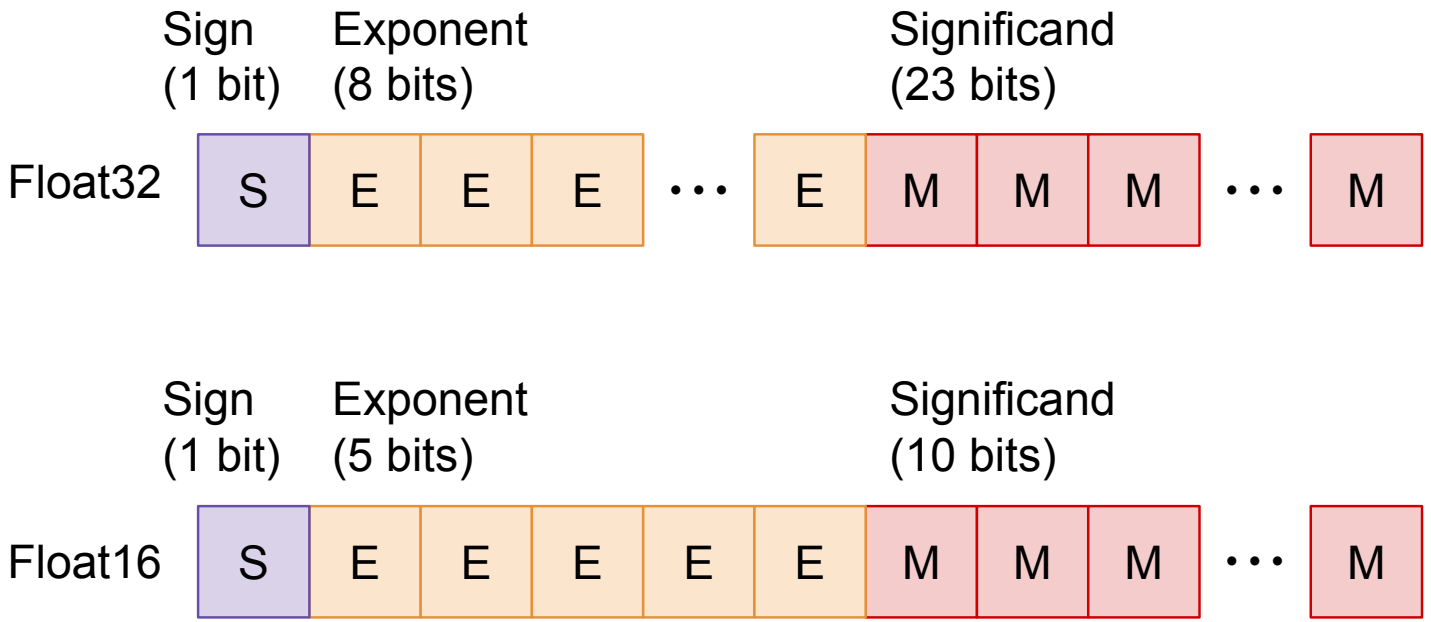
Are the benefits worth the accuracy loss?

Float16 failure

Implemented naively, float16 training fails.



Floating point theory



Floating point values are stored in binary scientific notation. As a rule of thumb, float32 values have around 7 significant (decimal) digits and float16 values have around 3 significant digits.

Float16 gradient descent

Gradient descent formula

$$x_{t+1} = x_t - \eta \frac{\partial \mathcal{L}}{\partial x}$$

What if we have

$$x_t = 10, \quad \eta = 10^{-3}, \quad \frac{\partial \mathcal{L}}{\partial x} = -10^{-1}?$$

The updated weight is

$$x_{t+1} = 1.000 \times 10^1 + 1.000 \times 10^{-4} = 1.000 \times 10^1 = x_t.$$

Float16 gradient descent

Gradient descent formula

$$x_{t+1} = x_t - \eta \frac{\partial \mathcal{L}}{\partial x}$$

What if we have

$$x_t = 10, \quad \eta = 10^{-3}, \quad \frac{\partial \mathcal{L}}{\partial x} = -10^{-1}?$$

The updated weight is

$$x_{t+1} = 1.000 \times 10^1 + 1.000 \times 10^{-4} = 1.000 \times 10^1 = x_t.$$

Even with perfect gradient computation, float16 weights are hard to fine tune!

Basic mixed precision model

Gradient descent formula

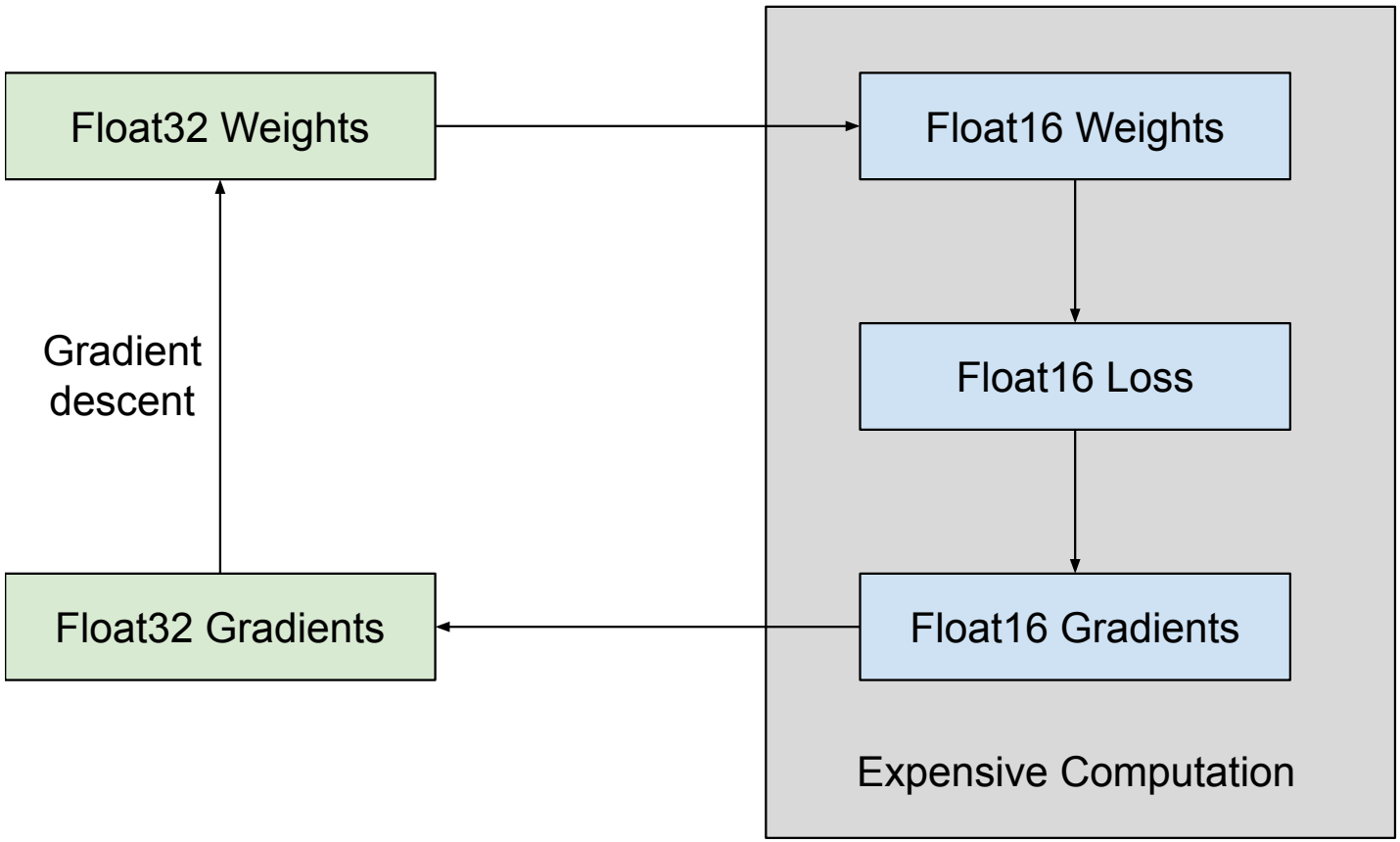
$$x_{t+1} = x_t - \eta \frac{\partial \mathcal{L}}{\partial x}$$

What we hold the **weights** in float32 and **gradients** in float16?

$$x_{t+1} = 1.0000000 \times 10^1 + 1.000 \times 10^{-4} = 1.0001000 \times 10^1$$

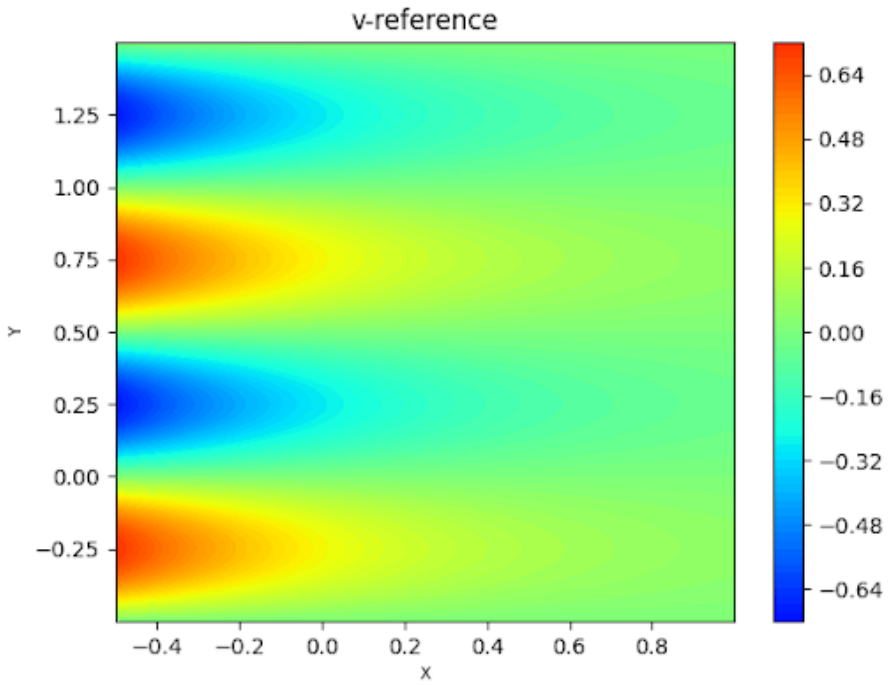
The weight updates correctly!

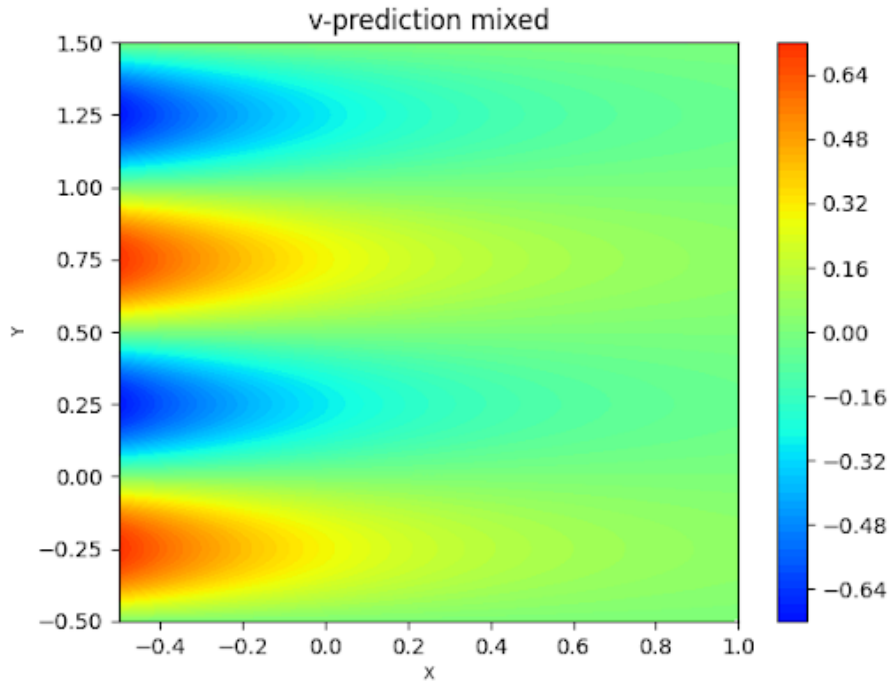
Basic mixed precision model



Results

The mixed precision model is as accurate as the float32 model.





Training time and memory improvements

Precision	Memory (GB)
Float32	1.51 ± 0.02
Float16	0.76 ± 0.01
Mixed precision	0.76 ± 0.01

There was also a 12% reduction in training time, which would be more drastic for a larger problem with more dimensions.

Acknowledgements

I would like to thank

- My mentor, Prof. Lu Lu
- Jacob Goldman-Wetzler, Joel Hayford
- MIT PRIMES-USA
- My family