Tensorflow vs. PyTorch Prediction Comparisons

Update Date: 15 June, 2023

NYU

Optimizer Definitions

```
# Optimizers
self.optimizer = tf.contrib.opt.ScipyOptimizerInterface(self.loss,
                                                        method = 'L-BFGS-B',
                                                        options = {'maxiter': 50000,
                                                                   'maxfun': 50000,
                                                                   'maxcor': 50,
                                                                   'maxls': 50,
                                                                   'ftol' : 0.000001 * np.finfo(float).eps})

self.optimizer_Adam = tf.train.AdamOptimizer(learning_rate = learning_rate)
self.train_op_Adam = self.optimizer_Adam.minimize(self.loss)
```

Tensorflow

```
# Define optimizers
self.optimizer_LBFGS = torch.optim.LBFGS(
    self.dnn.parameters(),
    lr                  = 0.001,
    max_iter            = LBFGS_epochs,
    max_eval            = LBFGS_epochs * 1.25,
    history_size        = 5000,
    tolerance_grad      = 1e-7,
    tolerance_change    = 0.000001 * np.finfo(float).eps,
    line_search_fn      = "strong_wolfe")

self.optimizer_Adam = torch.optim.Adam(self.dnn.parameters(), lr = Adam_lr)
```

PyTorch

Tensorflow: Adam Initialization

- `learning_rate` : A Tensor or a floating point value. The learning rate.
- `beta1` : A float value or a constant float tensor. The exponential decay rate for the 1st moment estimates.
- `beta2` : A float value or a constant float tensor. The exponential decay rate for the 2nd moment estimates.
- `epsilon` : A small constant for numerical stability. This epsilon is "epsilon hat" in the Kingma and Ba paper (in the formula just before Section 2.1), not the epsilon in Algorithm 1 of the paper.
- `use_locking` : If True use locks for update operations.
- `name` : Optional name for the operations created when applying gradients. Defaults to "Adam".

```
__init__(
    learning_rate=0.001,
    beta1=0.9,
    beta2=0.999,
    epsilon=1e-08,
    use_locking=False,
    name='Adam'
)
```

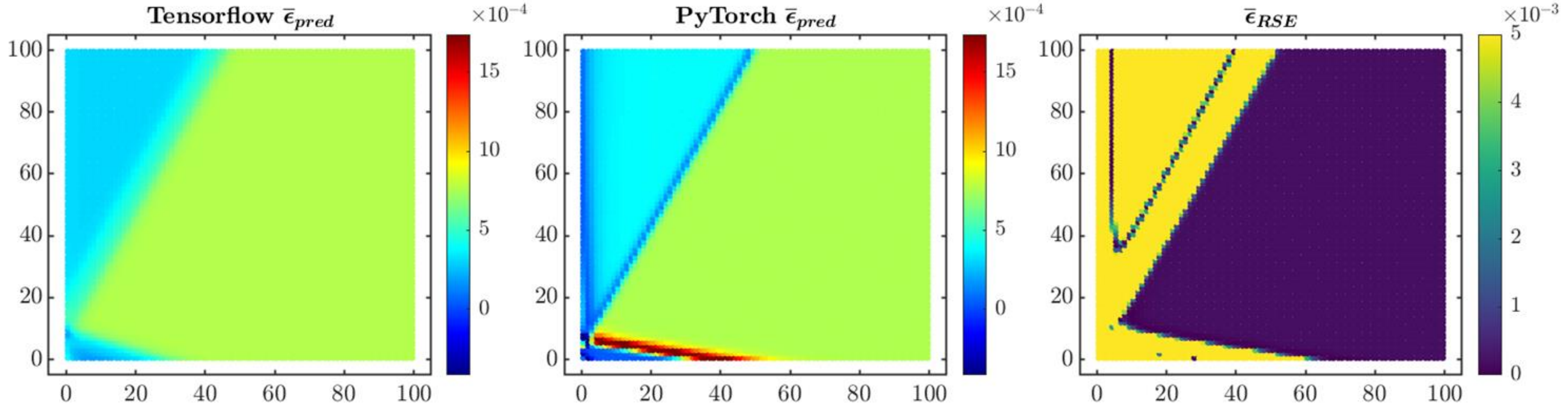
http://man.hubwiz.com/docset/TensorFlow.docset/Contents/Resources/Documents/api_docs/python/tf/train/AdamOptimizer.html

NYU

- Completed the training and derived prediction contours using both Tensorflow and PyTorch and also plotted them side by side.
- In calculating the error in non-local equivalent strain contour, I used the formula below due to the fact that errors in the Pytorch predictions are the reason for this investigation:

$$\frac{\left(\tilde{\epsilon}_{pred}, Tensorflow - \tilde{\epsilon}_{pred}, Pytorch\right)^2}{\left(\tilde{\epsilon}_{pred}, Tensorflow\right)^2}$$

NYU



Prediction Contour Generated Using Tensorflow

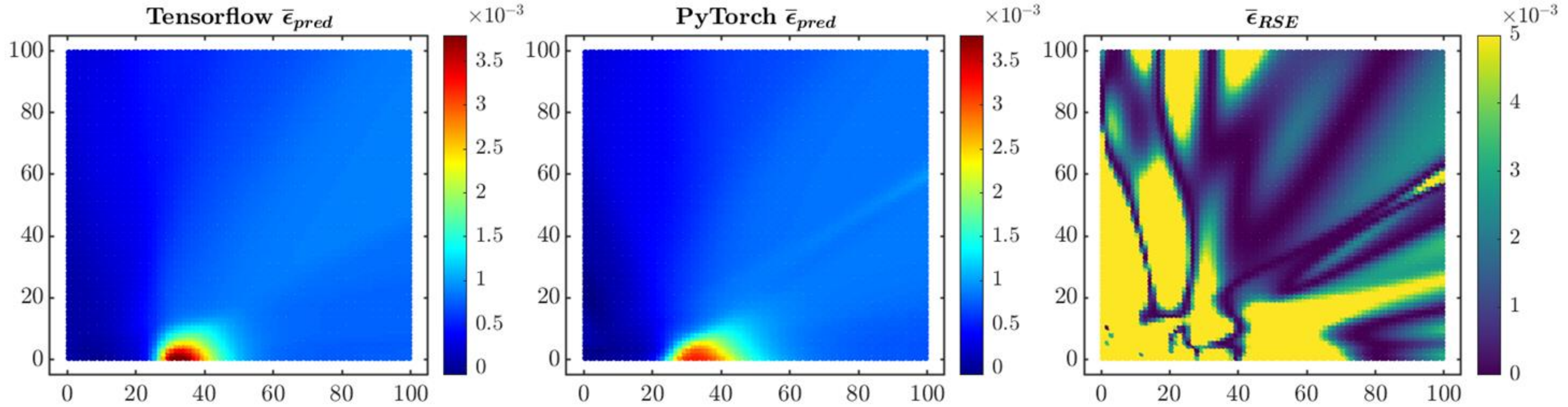
Prediction Contour Generated Using PyTorch

Error in Non-Local Equivalent Strain Contour

- Initialized with Weights and Biases generated with Tensorflow approach.
- The above are the predictions for a trained 8 x 8 Neural Network after 5000 ADAM + L-BFGS

NYU

Folder with generated matlab figure (.fig) and .png file: Prediction Contours



Prediction Contour Generated Using Tensorflow

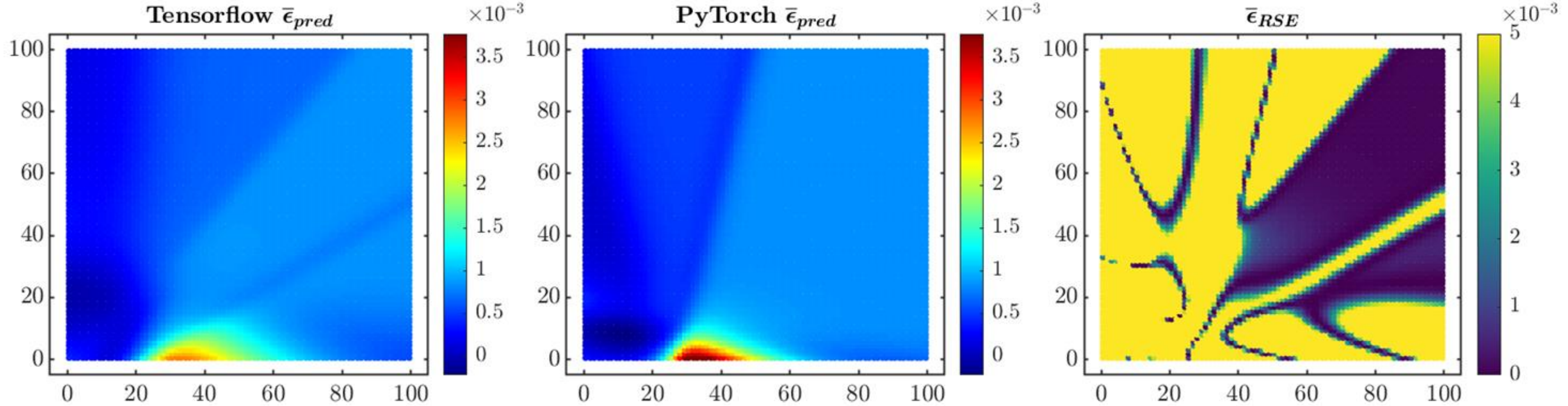
Prediction Contour Generated Using PyTorch

Error in Non-Local Equivalent Strain Contour

- Initialized with Weights and Biases generated with Tensorflow approach.
- The above are the predictions for a trained 16 x 16 Neural Network after 5000 ADAM + L-BFGS

NYU

Folder with generated matlab figure (.fig) and .png file: Prediction Contours



Prediction Contour Generated Using Tensorflow

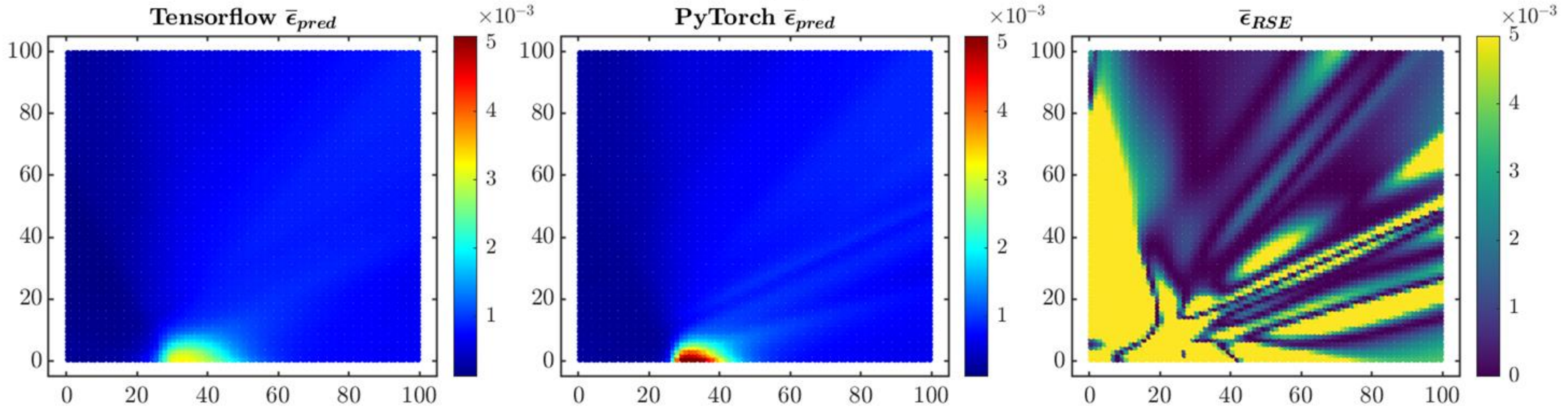
Prediction Contour Generated Using PyTorch

Error in Non-Local Equivalent Strain Contour

- Initialized with Weights and Biases generated with PyTorch approach.
- The above are the predictions for a trained 8 x 8 Neural Network after 5000 ADAM + L-BFGS

NYU

Folder with generated matlab figure (.fig) and .png file: Prediction Contours



Prediction Contour Generated Using Tensorflow

Prediction Contour Generated Using PyTorch

Error in Non-Local Equivalent Strain Contour

- Initialized with Weights and Biases generated with PyTorch approach.
- The above are the predictions for a trained 16 x 16 Neural Network after 5000 ADAM + L-BFGS

NYU

Folder with generated matlab figure (.fig) and .png file: Prediction Contours

Tensorflow vs. PyTorch Prediction Comparisons Update 2

Update Date: 22 June, 2023

NYU

Adam Optimizer Parameters

Mathematical Aspect of Adam Optimizer

Taking the formulas used in the above two methods, we get

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\left[\frac{\delta L}{\delta w_t}\right] v_t = \beta_2 v_{t-1} + (1-\beta_2)\left[\frac{\delta L}{\delta w_t}\right]^2$$

Parameters Used :

1. ϵ = a small +ve constant to avoid 'division by 0' error when ($v_t \to 0$). (10^{-8})
2. β_1 & β_2 = decay rates of average of gradients in the above two methods. (β_1 = 0.9 & β_2 = 0.999)
3. α – Step size parameter / learning rate (0.001)

Cite: https://www.geeksforgeeks.org/intuition-of-adam-optimizer/

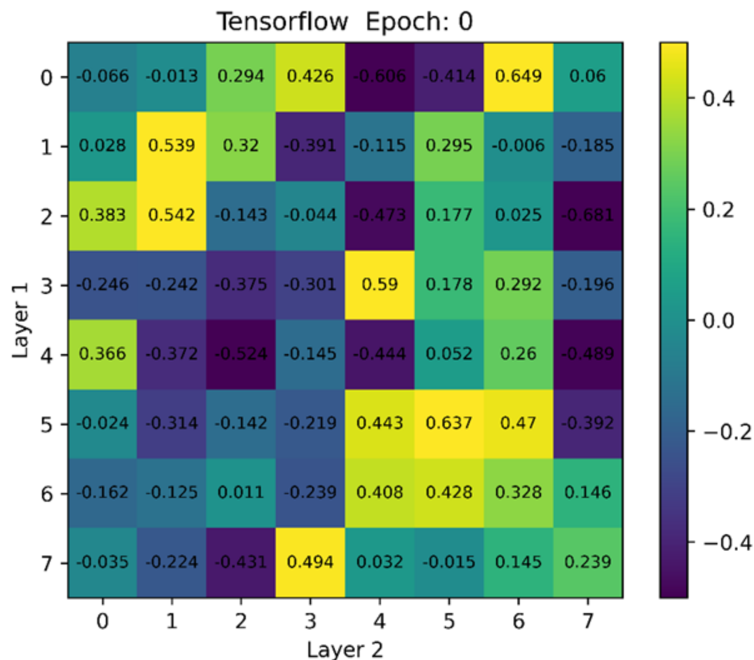
Adam Optimizer Parameters for Tensorflow

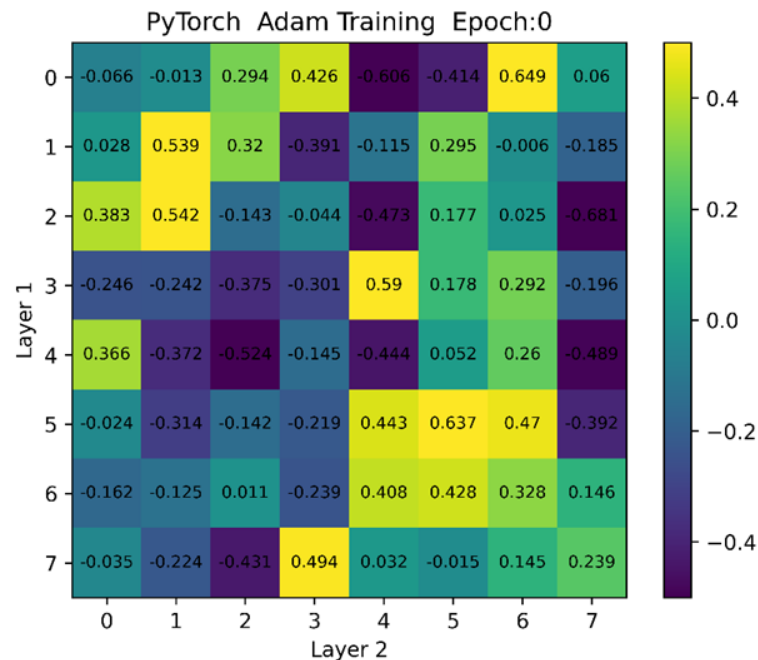
```
self.optimizer_Adam = tf.train.AdamOptimizer(learning_rate = learning_rate,
                                             beta1=0.9,
                                             beta2=0.999,
                                             epsilon=1e-08,
                                             use_locking=True,
                                             name='Adam')
```

Adam Optimizer Parameters for Pytorch

```
self.optimizer_Adam = torch.optim.Adam(self.dnn.parameters(),
                                       lr = Adam_lr,
                                       betas=(0.9, 0.999),
                                       eps=1e-08,
                                       amsgrad=False)
```

Weight Contour with Same Initialized Optimizer Parameters It: 0

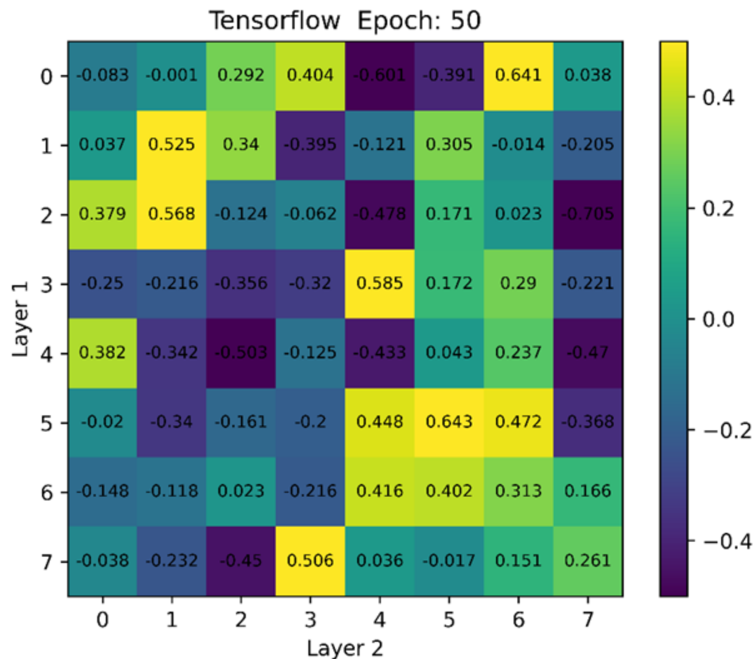


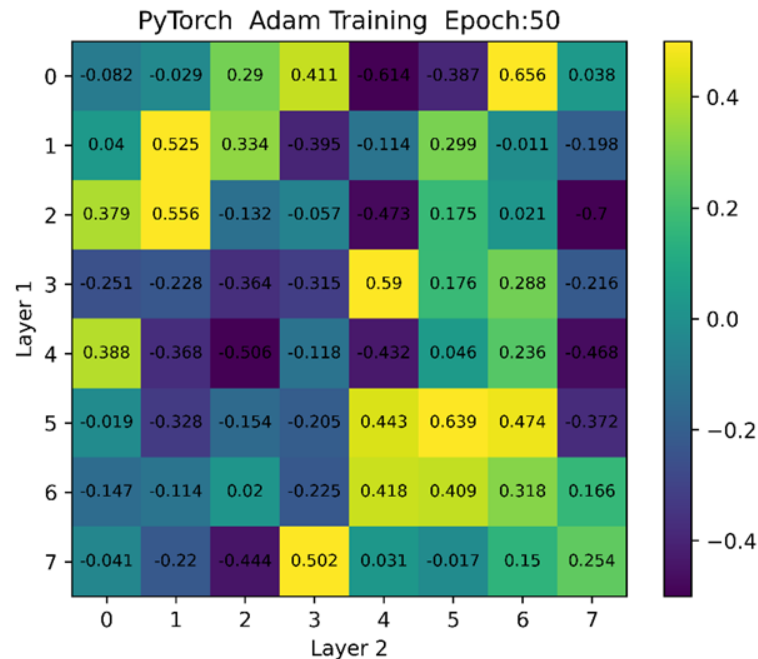


Weights between Hidden Layer 1 and Layer 2 for an 8x8 Network using Initial Weights and Biases Generated from Tensorflow

NYU

Weight Contour with Same Initialized Optimizer Parameters It: 50

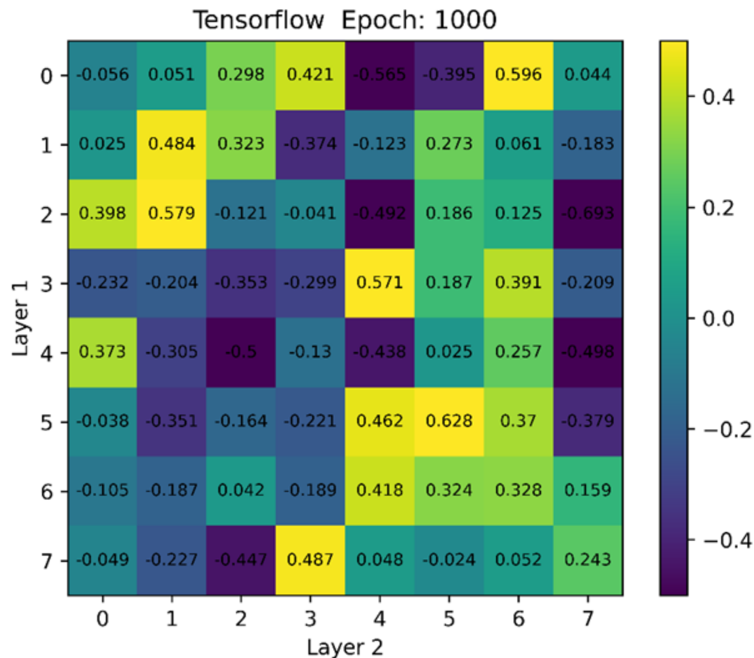


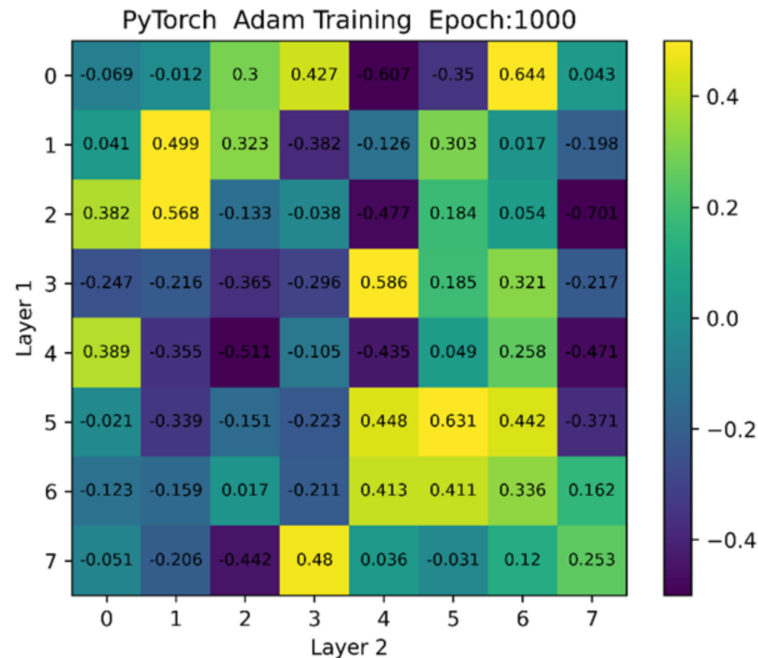


Weights between Hidden Layer 1 and Layer 2 for an 8x8 Network using Initial Weights and Biases Generated from Tensorflow

NYU

Weight Contour with Same Initialized Optimizer Parameters It: 1000





Weights between Hidden Layer 1 and Layer 2 for an 8x8 Network using Initial Weights and Biases Generated from Tensorflow

Tensorflow vs. PyTorch
Scaling Factor Error

Update Date: 4 July, 2023

NYU

Problem Definition: Trivial Solutions

In Pytorch, a scaling factor was introduced to the inputs of the neural network and later removed when storing the predictions as seen in the code below.

Scaling inputs up:

```
All_TrainingData_StrB100_GP_np[:,:,10]       = All_TrainingData_StrB100_GP_np[:,:,10] * elocalfactor
All_TrainingData_StrB100_nodeslrb_np[:,:,10] = All_TrainingData_StrB100_nodeslrb_np[:,:,10] * elocalfactor
All_TrainingData_StrB100_nodesbtb_np[:,:,10] = All_TrainingData_StrB100_nodesbtb_np[:,:,10] * elocalfactor
```

Scaling saved predictions back down:

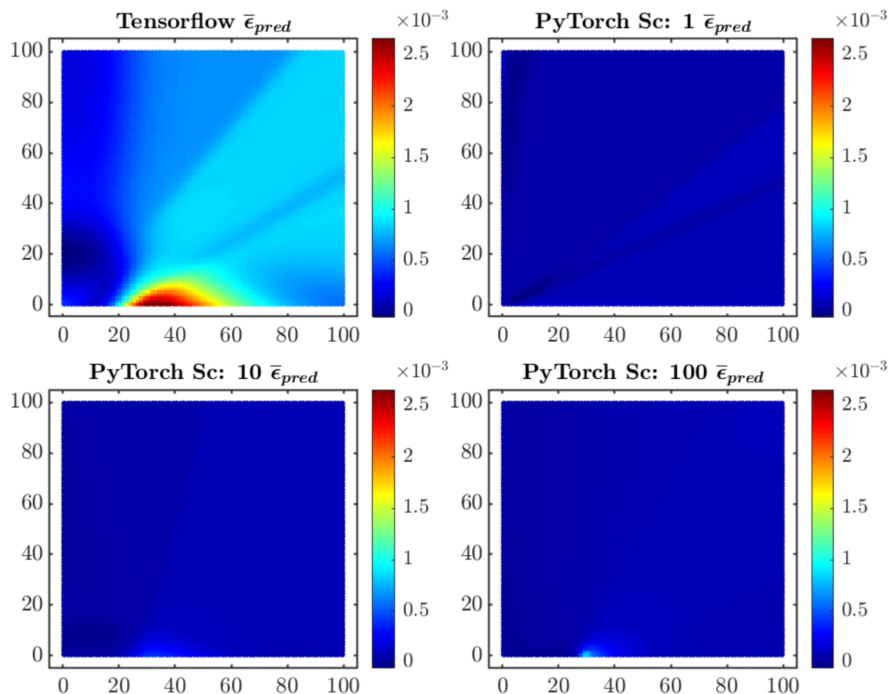
```
x = predictions.detach().cpu().numpy()
with open(prediction_file, 'w') as f:
    for i in range(6400):
        f.write(str(float(x[i])/elocalfactor))
        f.write('\n')
```

NYU

Problem Definition: Trivial Solutions

This resulted in most trivial solutions in Pytorch as observed below for different scaling factors used.

- I tried a scaling factor of 1, 10, 100.
- All 8x8 networks were run for 5000 Adam Epochs + L-BFGS.
- All networks were initialized with the same weights and biases.
- ***Problem:*** Prediction Contours were trivial as seen in the right Figure



NYU

Problem Definition: Trivial Solutions

On closer inspection, two errors were noticed:

1) It was seen that only the e_local inputs were scaled up by the e_local_factor. The x, y and g inputs were not scaled.

- In this code, only the element with index [:, : ,10] (the e_local variable) was scaled.
- The x, y and g variables were indexed [:, : ,0], [:, : ,1] and [:, : ,2] respectively and not scaled.
- As such, the code was rewritten to apply to the **scale factor to all input variables.**

```
All_TrainingData_StrB100_GP_np[:,:,10]       = All_TrainingData_StrB100_GP_np[:,:,10] * elocalfactor
All_TrainingData_StrB100_nodeslrb_np[:,:,10] = All_TrainingData_StrB100_nodeslrb_np[:,:,10] * elocalfactor
All_TrainingData_StrB100_nodesbtb_np[:,:,10] = All_TrainingData_StrB100_nodesbtb_np[:,:,10] * elocalfactor
```

NYU

Problem Definition: Trivial Solutions

On closer inspection, two errors were noticed:

2) The elocal input values in the .mat file for Pytorch were 0.1 times less than than the elocal input values in the .csv file for Tensorflow.

- The code printed out the raw data values from the input files of Tensorflow and Pytorch. The difference is highlighted below.
- As such, **before any scale factor is applied**, the values of the elocal in the .mat file should be **multiplied by 10**.

```
u3) C:\Users\kee301\Documents\CSM\Pytorch_vs_Te
         xcoord      ycoord      g          elocal
 0.528312    0.528312   8.0   1.417744e-06
 1.971688    0.528312   8.0   8.182386e-07
 0.528312    1.971688   8.0   8.158908e-07
```

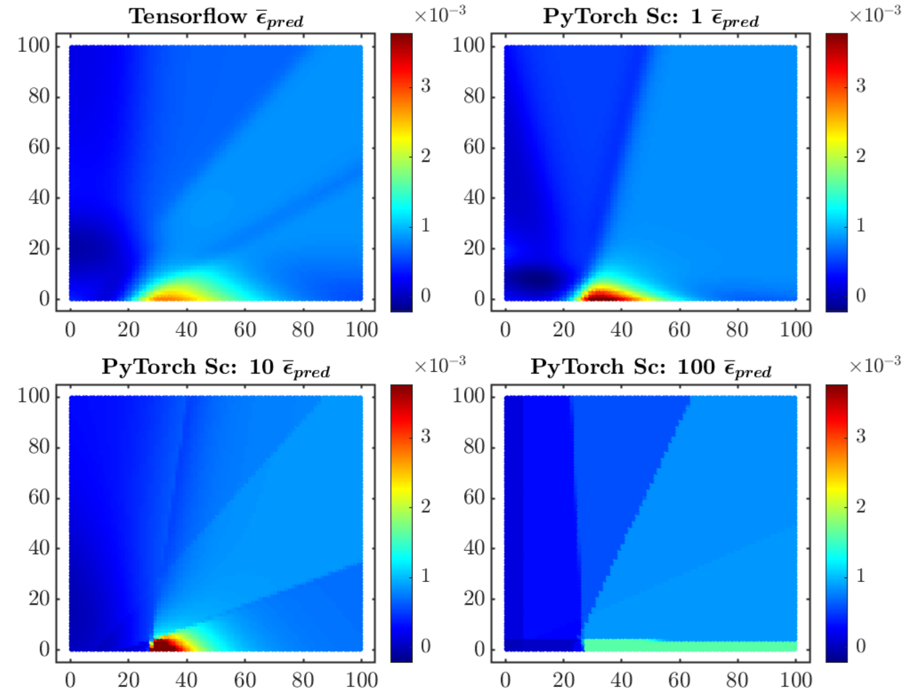
```
[[[       x                y                g              elocal    ]]
[[[5.28312163e-01 5.28312163e-01 8.00000000e+00 1.41774367e-07]]

 [[1.97168784e+00 5.28312163e-01 8.00000000e+00 8.18238599e-08]]

 [[5.28312163e-01 1.97168784e+00 8.00000000e+00 8.15890766e-08]]
```

NYU

Problem Definition: Trivial Solutions

The code was fixed and the same prediction contours were created as referenced in slide 16.

- ***Result:***
- Prediction Contours for Pytorch resulted in non-trivial solutions.
- Addition of a scale factor compromises accuracy, but makes solution faster.
- Error Plot btw Tensorflow and Pytorch (Scale 1) can be observed in **Slide 7**



NYU

Tensorflow vs. PyTorch

Investigation on Gradients

Update Date: 27 June, 2023

NYU

Tensorflow: Adam Initialization

Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while θ_t not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return θ_t (Resulting parameters)

(Kingma et. al) https://arxiv.org/pdf/1412.6980.pdf

Parameters (theta) refers to the elements in the weight and biases matrices.

g_t refers to the partial derivative of the loss function w.r.t. To the weights (w)/biases (b) at time t.

$$\frac{\delta L}{\delta w_t} \text{ or } \frac{\delta L}{\delta b_t}$$

NYU

First Study: Observation btw Tensorflow and Pytorch

Both platforms initialize the moment estimates as:

$$m_t = [0], v_t = [0]$$

A [4,1,1] Neural Network was created:

- Both Pytorch and Tensorflow were initialized with the same weights and biases
- After one epoch (t=1), the weights between the input layer and the first hidden layer (one neuron) were extracted for comparison.
- After one epoch, the **gradients ($\frac{\delta L}{\delta w_t}$)** computed to derive these weights were also extracted for comparison.
- Using this gradient, the weights at t=1 were calculated on MATLAB using the mathematical formula from the paper (Kingma et. al). Maximum error of each platform's weights at t=1 was -0.003%.
- **Hypothesis:** The result show that the differences in the weights (or biases) after one iteration can be attributed to different gradients being computed in Tensorflow and Pytorch.

TENSORFLOW				
Weights(t=0)	**Tensorflow Gradients**	**Weights(t=1) Calculated from MATLAB**	**Weights(t=1) Calculated from Tensorflow**	**Error(%)**
0.88782865	1.1028266E+00	0.88682865	0.88682866	0.000
0.06018033	-6.1464891E+00	0.06118033	0.06118033	0.000
-0.46538195	-6.9716954E-01	-0.46438195	-0.46438196	0.000
-0.49303034	-2.0500580E-05	-0.49203083	-0.49204552	-0.003

PYTORCH				
Weights(t=0)	**Pytorch Gradients**	**Weights(t=1) Calculated from MATLAB**	**Weights(t=1) Calculated from Pytorch**	**Error(%)**
0.88782865	1.5916174E-02	0.88682865	0.88682866	0.000
0.06018033	2.2873601E-01	0.05918033	0.05918033	0.000
-0.46538195	2.9203158E-02	-0.46638195	-0.46638194	0.000
-0.49303034	7.0351894E-07	-0.49401632	-0.49401632	0.000

Second Study: Observation btw Tensorflow and Pytorch

Next Study:

- **For a [4,4,1] Neural Network with same initial weights & biases**
- **50 epochs**

Method

- For each iteration, the gradients of weights and biases computed in Tensorflow for each iteration were saved.

- In the source code of Pytorch Adam Optimizer, these Tensorflow gradients were used in place of the gradients computed.

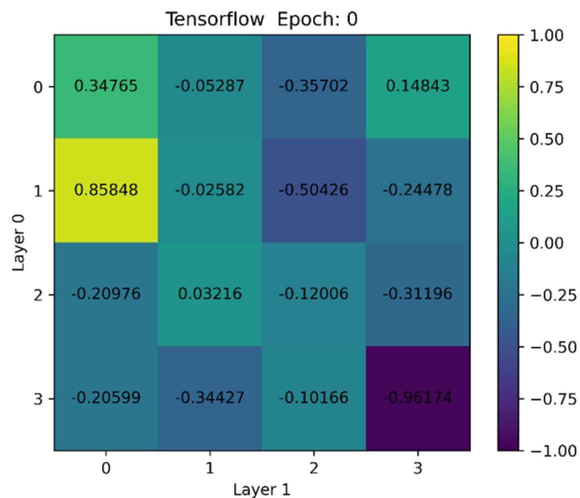
Results:

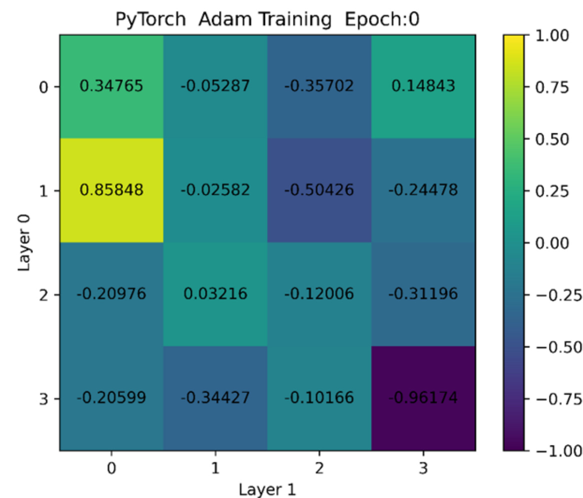
- When the gradients were set to be the same, we obtained the same weights for the first 3 out of 4 rows of weights with extremely close values in the fourth row.

- When the gradients were set to be the same, we obtained the same biases throughout.

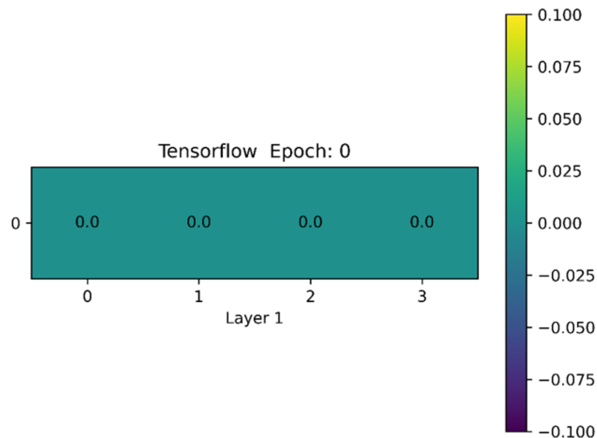
Initialization

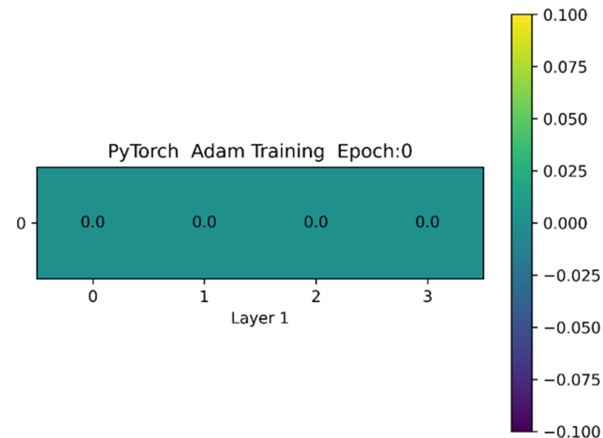
At It 0:
Same Weights and Biases





At It 0:
Initialized with Same Weights and Biases

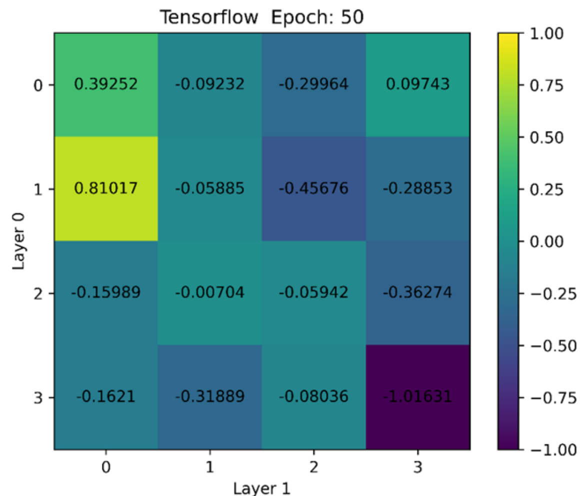


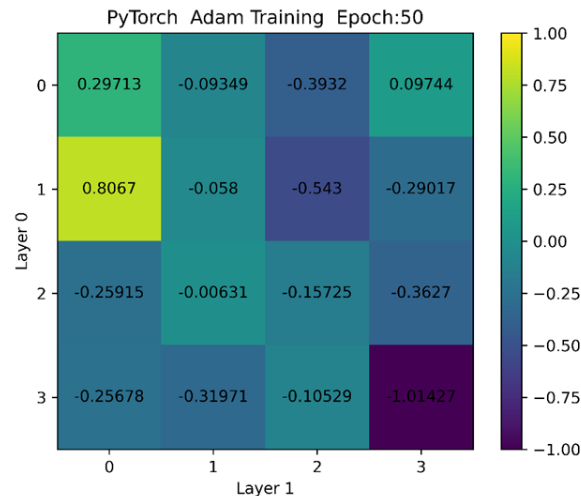


NYU

At It 50:
With Platform (Different) Computed Gradients

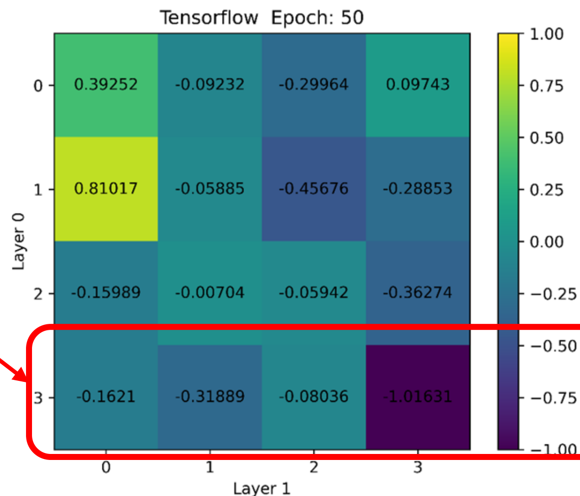
Slight deviation in all cells

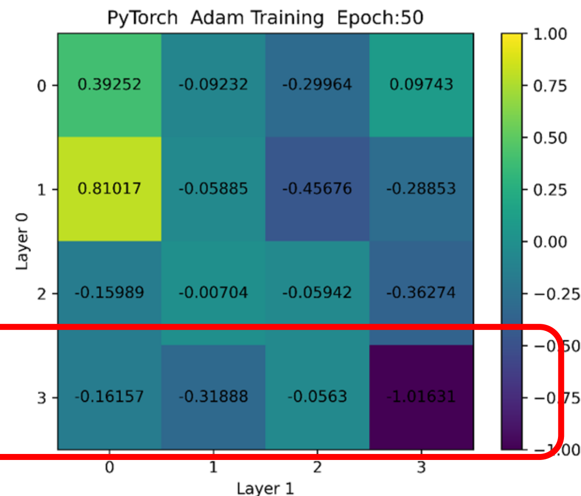




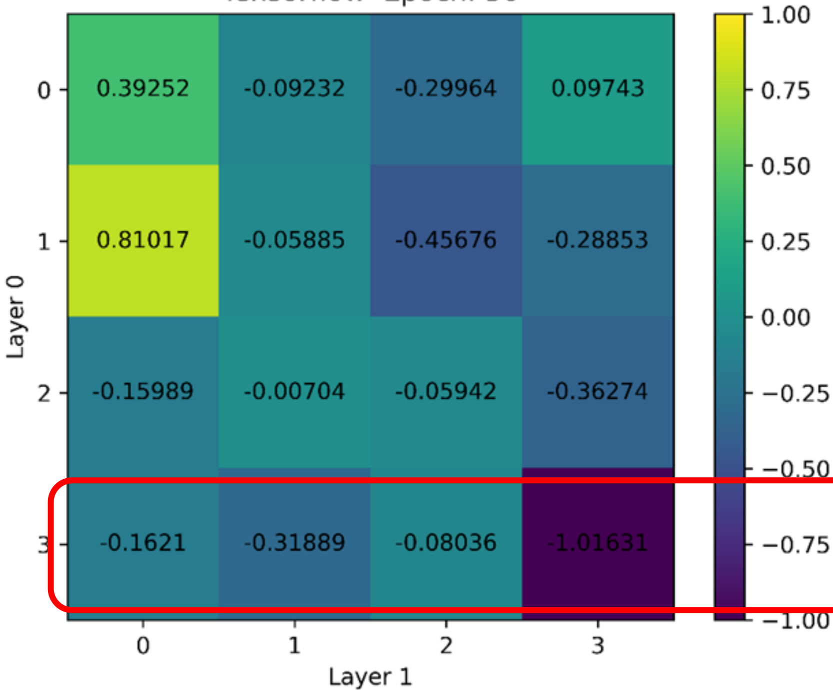
At It 50:
With Same Gradients

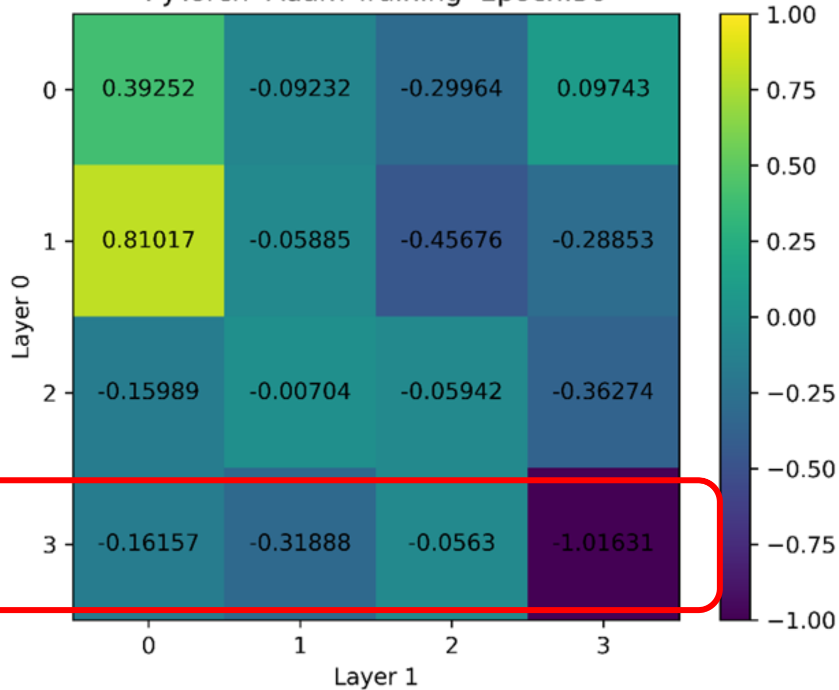
Only difference is in this row





NYU

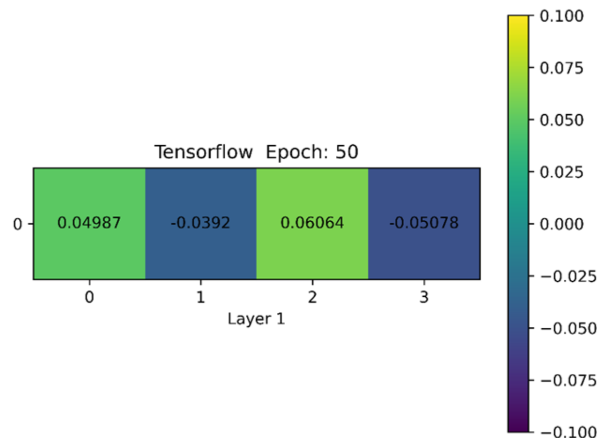


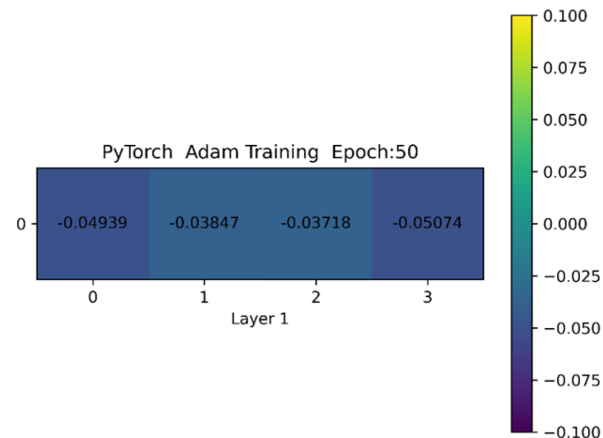


NYU

At It 50:
With Platform (Different) Computed Gradients

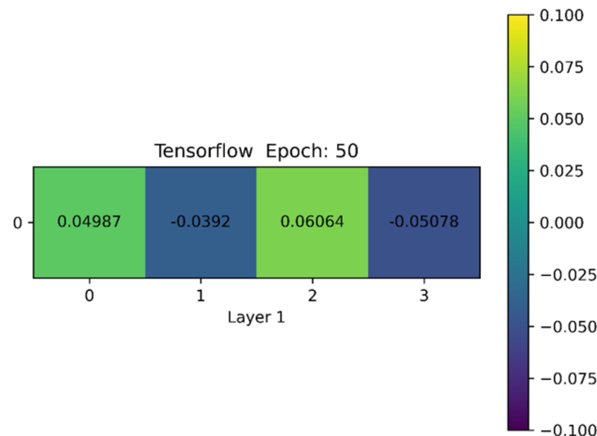
Slight deviation in all values

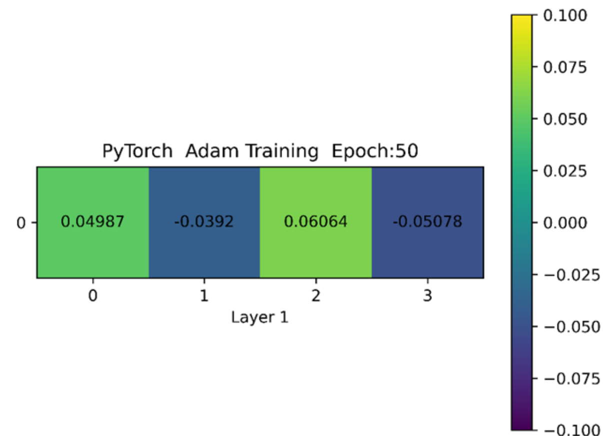




At It 50:
With Same Gradients

Similar to at least 5 sig fig





NYU

Tensorflow vs. PyTorch

Investigation on Pytorch Adam Gradients and Automatic Differentiation

Update Date: 9 July, 2023

NYU

Investigation of Gradient Computation in Tensorflow and Pytorch

Problem Description: The past update showed that when the tensorflow values for the adam optimizer gradients (g(t)) were set in pytorch, it produced almost similar results (last row for the weights showed some deviation at 5 sig. fig.)

Therefore, as of now, the following values are the same:

- Initial Weights and Biases
- Adam Optimizer Gradients at every iteration

However, the values derived in the loss function were observed to be different despite being mathematically similar due to differences in the derivatives calculated.

Current Loss Function:

$$Loss = \sqrt{\sum (Residual_{PDE})^2} + \sqrt{\sum \left(Residual_{PDEBCs_{lrb}} \right)^2} + \sqrt{\sum \left(Residual_{PDEBCs_{btb}} \right)^2}$$

$$Residual_{PDE} = \varepsilon_{nl} - g\left(\frac{\delta^2 \varepsilon_{nl}}{\delta x^2} + \frac{\delta^2 \varepsilon_{nl}}{\delta y^2} \right) - \varepsilon_l$$

$$Residual_{PDEBCs_{lrb}} = \frac{\delta \varepsilon_{nl}}{\delta x},\ Residual_{PDEBCs_{btb}} = \frac{\delta \varepsilon_{nl}}{\delta y}$$

NYU

Experiment 1

1. **A 4,1,1, Neural Network was created and initialized with the same weights and biases. Adam Training was run for 5000 epochs.**

1. **Only one set of inputs were used:**
   data: [x, y, g, elocal] = [2, 3, 8, 5]

1. **Loss function was simplified to not include differentiation.**
   enon_local_true = 11

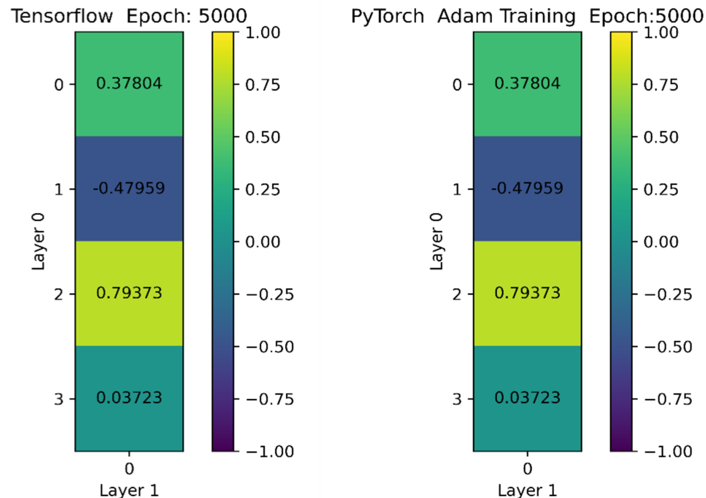
$$Loss = \left(\varepsilon_{non\ local\ pred} - \varepsilon_{non\ local\ true}\right)^2$$

1. **The maps of the weights and biases (between the input layer and hidden layer) were printed for every 1000 epochs following these 2 procedures:**
   a. **First Method:** Each program was allowed to generate maps and predictions using the Adam Optimizer Gradients of that program
   b. **Second Method:** The Tensorflow Adam Optimizer Gradients (g(t)) were set in Pytorch

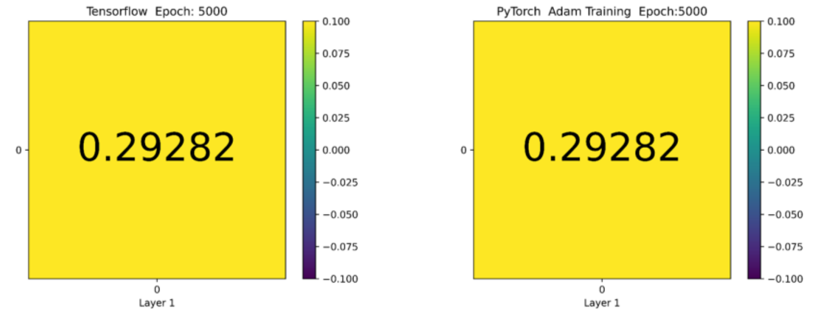
Experiment 1: Method 1

Method 1: Tensorflow and Pytorch were allowed to generate maps and predictions using the Adam Optimizer Gradients of that platform.

After 5000 epochs, the weight contour map was generated between the input layer and hidden layer:



Weights Contour Maps (Same values)

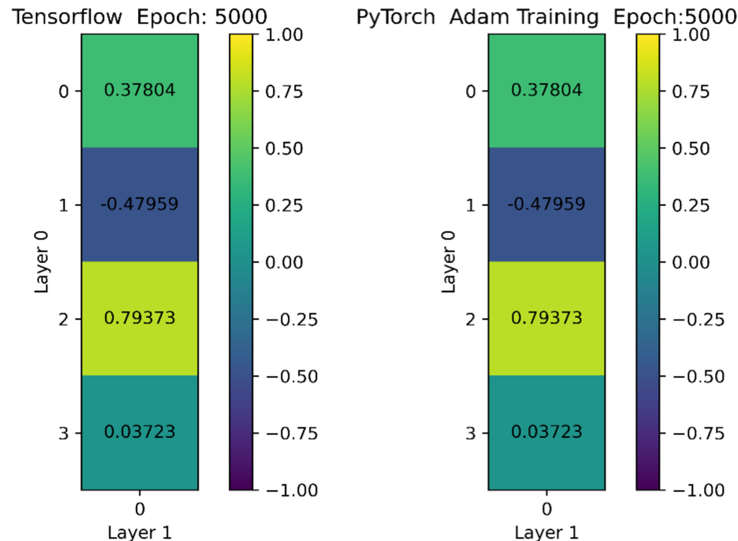


Biases Contour Maps (Same values)

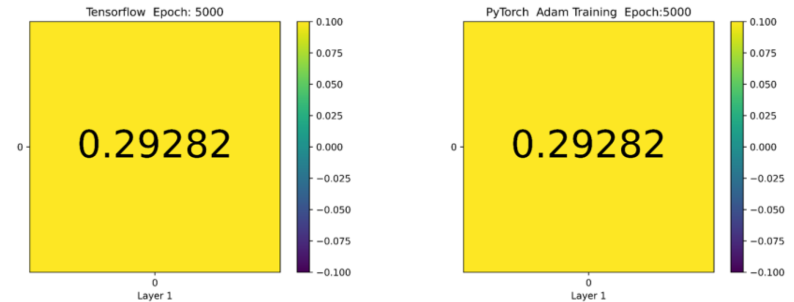
Experiment 1: Method 2

Method 2: The Tensorflow Adam Optimizer Gradients (g(t)) were set in Pytorch

After 5000 epochs, the weight contour map was generated between the input layer and hidden layer:



Weights Contour Maps (Same values)



Biases Contour Maps (Same values)

Experiment 1: Discussion

1. In this simple neural network, the same biases and weights were computed in Pytorch despite the method used in training the networks (Method 1 vs Method 2).

1. The Predictions derived were also similar as shown here:

	Tensorflow Predictions	Pytorch Predictions	Error (%)
Method 1	8.3652573	8.3652334	0.00029
Method 2	8.3652573	8.3652229	0.00041

1. For further analysis, the Adam Optimizer Gradients for each neural network was extracted at 5000 epochs and compared.

	Tensorflow Gradients	Pytorch Gradients	Error (%)
Weights	-8.28680E-04	-8.28680E-04	0.0000
	-1.24302E-03	-1.24302E-03	0.0000
	-3.31471E-03	-3.31473E-03	-0.0006
	-2.07170E-03	-2.07171E-03	-0.0005
Biases	-4.14340E-04	-4.14340E-04	0.0000

NYU

Experiment 2

1. **The results of experiment 1 hinted that the Adam Optimizer functions of both Tensorflow and Pytorch are correctly executing the same mathematical formula.**

1. **I returned to the loss function below and decided to print out the values of the variables that comprise the function to note differences.**

$$Loss = \sqrt{\sum (Residual_{PDE})^2} + \sqrt{\sum \left(Residual_{PDEBCs_{lrb}} \right)^2} + \sqrt{\sum \left(Residual_{PDEBCs_{btb}} \right)^2}$$

$$Residual_{PDE} = \varepsilon_{nl} - g\left(\frac{\delta^2 \varepsilon_{nl}}{\delta x^2} + \frac{\delta^2 \varepsilon_{nl}}{\delta y^2}\right) - \varepsilon_l$$

$$Residual_{PDEBCs_{lrb}} = \frac{\delta \varepsilon_{nl}}{\delta x}, \; Residual_{PDEBCs_{btb}} = \frac{\delta \varepsilon_{nl}}{\delta y}$$

1. **To obtain results, a 4,1,1, Neural Network was created and initialized with the same weights and biases. Adam Training was run for 5000 epochs.**

1. **Only one set of inputs for each file was used:**
   data: [x, y, g, elocal] = [0.5, 0.5, 8, 1.4 E-06]
   data_lrb: [x, y, g, elocal] = [0, 0, 8, 1.8 E-06]
   data_btb: [x, y, g, elocal] = [0, 0, 8, 1.8 E-06]

		At Adam Epoch = 1	
	Description of Variable	**Tensorflow Values**	**Pytorch Values**
Adam Predictions		6.690440E-01	6.690439E-01
dee/dxx	Second-order derivative of nonlocal strain prediction w.r.t. the x-coordinate	-2.019621E-06	0
dee/dyy	Second-order derivative of nonlocal strain prediction w.r.t. the y-coordinate	-2.992257E-06	0
de/dx	First-order derivative of nonlocal strain prediction (lrb nodes) w.r.t. the x-coordinate	2.078373E-06	0
de/dy	First-order derivative of nonlocal strain prediction (btb nodes) w.r.t. the y-coordinate	-2.529810E-06	0
Loss (Python)		6.690874E-01	6.690425E-01
Loss (MATLAB)		6.690873E-01	6.690425E-01
Error Check (%)		-0.00001	0

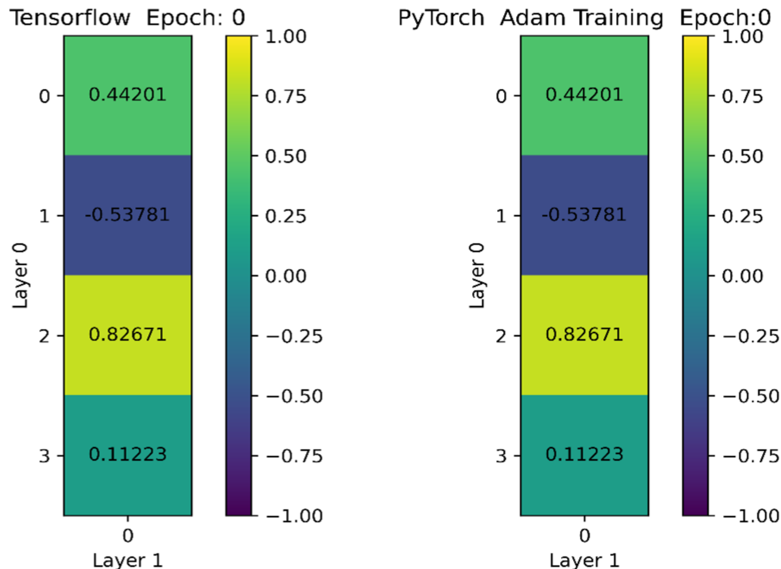
		At Adam Epoch = 5000	
	Description of Variable	**Tensorflow Values**	**Pytorch Values**
Adam Predictions		2.009869E-04	-3.733933E-04
dee/dxx	Second-order derivative of nonlocal strain prediction w.r.t. the x-coordinate	-1.084453E-09	0
dee/dyy	Second-order derivative of nonlocal strain prediction w.r.t. the y-coordinate	-9.466578E-09	0
de/dx	First-order derivative of nonlocal strain prediction (lrb nodes) w.r.t. the x-coordinate	4.657715E-09	0
de/dy	First-order derivative of nonlocal strain prediction (btb nodes) w.r.t. the y-coordinate	-1.376145E-08	0
Loss (Python)		1.996897E-04	3.747933E-04
Loss (MATLAB)		1.996897E-04	3.747933E-04
Error Check (%)		-0.00001	0

Experiment 2: Discussion

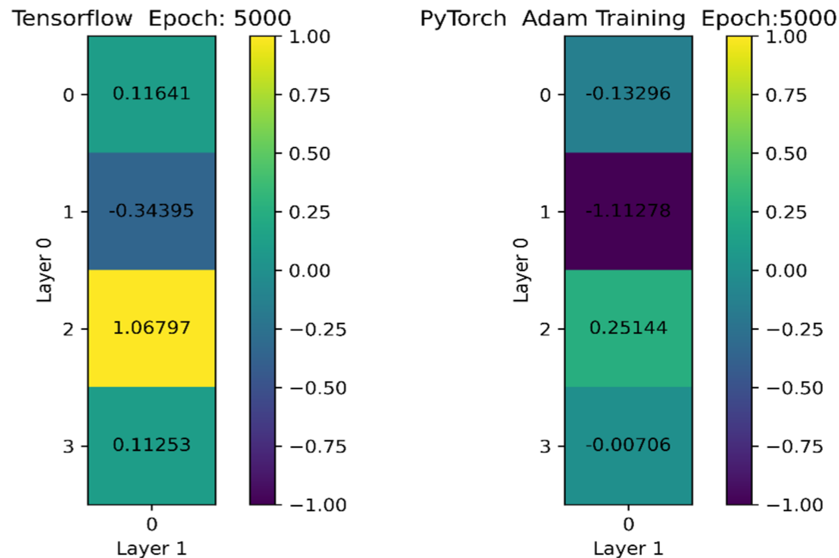
The automatic differentiation in Pytorch is yielding zero values.

Below shows that the weight and bias maps for Tensorflow and Pytorch deviated at 5000 epochs despite being initialized with the same values.

At Epoch 0



At Epoch 5000



Next Steps

Research Question:

- Could the Pytorch automatic differentiation be responsible for different weights and biases contour maps?

Possible Next Steps:

- While leaving Pytorch Adam Gradients untouched, I would input the derivatives from Tensorflow into Pytorch to see if it would now yield similar maps and predictions.
- Read online on errors encountered with pytorch automatic differentiation that yielded zero values.

```python
enonlocal_grad_xy_GP         = self.enonlocal_gradient(enonlocal_pred_GP, self.X_TrainingData_StrB100_GP)
enonlocal_grad_xy_nodeslrb  = self.enonlocal_gradient(enonlocal_pred_nodeslrb, self.X_TrainingData_StrB100_nodeslrb)
enonlocal_grad_xy_nodesbtb  = self.enonlocal_gradient(enonlocal_pred_nodesbtb, self.X_TrainingData_StrB100_nodesbtb)
```

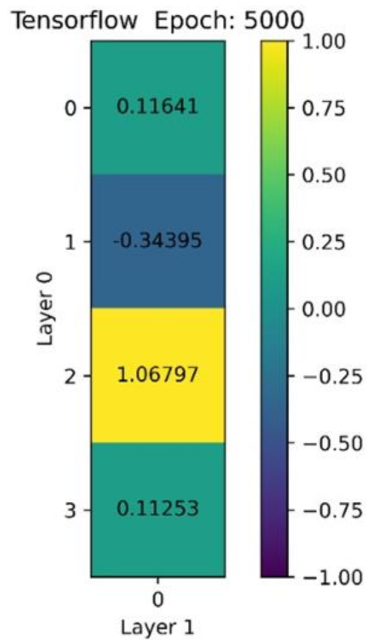
```python
def enonlocal_gradient(self, enonlocal, X):
    # input features: x-coord, y-coord, g, elocal
    # enonlocal_grad_xx, enonlocal_grad_yy
    denonlocal_dinput = torch.autograd.grad(enonlocal, X, grad_outputs=torch.ones_like(enonlocal), \
                                            create_graph=True, retain_graph=True, allow_unused = True)[0]
    enonlocal_grad_xy = torch.empty((len(X), self.Nincr, 2))
    enonlocal_grad_xy[:,:,0] = denonlocal_dinput[:, :, 0] # w.r.t. x_coord
    enonlocal_grad_xy[:,:,1] = denonlocal_dinput[:, :, 1] # w.r.t. y_coord

    print(denonlocal_dinput)
    print(enonlocal_grad_xy)
    print("\n")

    return enonlocal_grad_xy
```

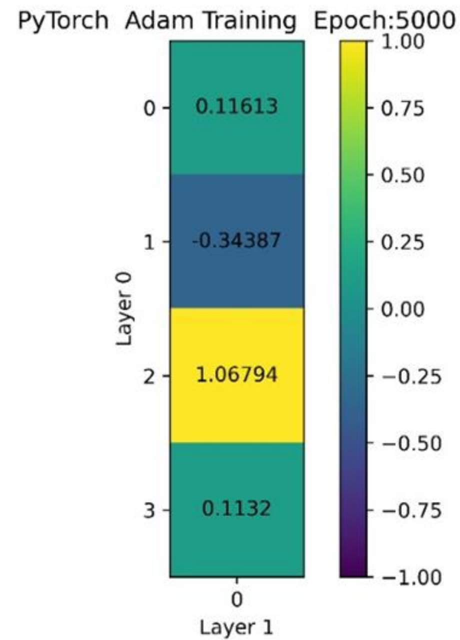
NYU

```
tensor([[[ 0.03503, -0.02032, -0.00082, -0.01739]]], device='cuda:0',
       grad_fn=<ReshapeAliasBackward0>)
tensor([], size=(1, 0, 2), grad_fn=<CopySlices>)


tensor([[[ 0.03432, -0.01991, -0.00080, -0.01704]]], device='cuda:0',
       grad_fn=<ReshapeAliasBackward0>)
tensor([], size=(1, 0, 2), grad_fn=<CopySlices>)


tensor([[[ 0.03432, -0.01991, -0.00080, -0.01704]]], device='cuda:0',
       grad_fn=<ReshapeAliasBackward0>)
tensor([], size=(1, 0, 2), grad_fn=<CopySlices>)
```

```python
def enonlocal_gradient(self, enonlocal, X):
    # input features: x-coord, y-coord, g, elocal
    # enonlocal_grad_xx, enonlocal_grad_yy
    denonlocal_dinput = torch.autograd.grad(enonlocal, X, grad_outputs=torch.ones_like(enonlocal), \
                                            create_graph=True, retain_graph=True, allow_unused = True)[0]
    # enonlocal_grad_xy = torch.empty((len(X), self.Nincr, 2))
    # enonlocal_grad_xy[:,:,0] = denonlocal_dinput[:, :, 0] # w.r.t. x_coord
    # enonlocal_grad_xy[:,:,1] = denonlocal_dinput[:, :, 1] # w.r.t. y_coord

    # print(denonlocal_dinput)
    # print(enonlocal_grad_xy)
    # print("\n")
    return denonlocal_dinput
```

NYU





NYU

1. ***Documentation for tf.gradients() function*** - https://github.com/tensorflow/tensorflow/blob/v2.13.0/tensorflow/python/ops/gradients_impl.py#L172-L315

1. ***Documentation for for torch.autograd.grad function -*** https://pytorch.org/docs/stable/_modules/torch/autograd.html#grad