Electronic Supplementary Materials

For https://doi.org/10.1631/jzus.A2000403

Physics-informed neural networks for estimating stress transfer mechanics in single lap joints

Shivam SHARMA, Rajneesh AWASTHI, Yedlabala Sudhir SASTRY,
Pattabhi Ramaiah BUDARAPU

Abstract

The electronic supplementary materials consist of snippets of the python script indicating the key implementation steps and the histograms denoting the weights and biases during the network training.

1. Snippets of Python script

Listing 1: Architecture and initialization of the network

```
def neural_net(self, X, weights, biases):
        num_layers = len(weights)+1
        A = 1.5*(X-self.lb)/(self.ub-self.lb)
        for l in range(0, num_layers-2):
            W = weights[l]
            b = biases[l]
            A = tf.tanh(tf.add(tf.matmul(A, W, name='matmul'+str(l)), b, \
            name='add'+str(l)), name='tanh'+str(l))
        W = weights[-1]
        b = biases[-1]
        Y = tf.add(tf.matmul(A, W, name='matmul'+str(num_layers-1)), b, \
            name='add'+str(num_layers-1))
        return Y
```

Listing 2: Implementation of the developed PINN

```
def net_uv(self, x):
        uv = self.neural_net(x, self.weights, self.biases)
        u = uv[:,0:1]
        v = uv[:,1:2]
        u_x = tf.gradients(u, x, name='u_x')[0]
        v_x = tf.gradients(v, x, name='v_x')[0]
        return u, v, u_x, v_x

    def net_f_uv(self, x):
        u, v, u_x, v_x = self.net_uv(x)

        u_xx = tf.gradients(u_x, x, name='u_xx')[0]
        u_xxx = tf.gradients(u_xx, x, name='u_xxx')[0]
        u_xxxx = tf.gradients(u_xxx, x, name='u_xxxx')[0]

        v_xx = tf.gradients(v_x, x, name='v_xx')[0]
        v_xxx = tf.gradients(v_xx, x, name='v_xxx')[0]
        v_xxxx = tf.gradients(v_xxx, x, name='v_xxxx')[0]

        #functions f and g
        f_u = (5.4968*(10**-9))*u_xxxx + (1.3410*(10**-9))*v_xxxx + \
              (1.4805*(10**-6))*u_xx + (1.4400*(10**-6))*v_xx + \
              (0.00038407)*u + (0.00038400)*v - (0.00038400)
        f_v = (1.3410*(10**-9))*u_xxxx + (2.0119*(10**-9))*v_xxxx + \
              (1.4400*(10**-6))*u_xx + (5.7390*(10**-7))*v_xx + \
              (0.00038400)*u + (0.00038407)*v - (0.00038400)

        return f_u, f_v, u_xx, v_xx
```

Listing 3: Training data

```
    X_f = lb + (ub-lb)*lhs(1, N_f)
    X_f = np.array(sorted(X_f))
    X_f = X_f.reshape((len(X_f),1))
    with open("X_f.txt","wb") as f1:
      np.savetxt(f1, X_f, delimiter=",")
```

Listing 4: Xavier initialization steps

```
def xavier_init(self, size, index):
        in_dim = size[0]
        out_dim = size[1]
        xavier_stddev = np.sqrt(2/(in_dim+out_dim))
        name = 'weight'+ str(index)
        return tf.Variable(tf.truncated_normal([in_dim, out_dim], stddev=xavier_stddev),\
        dtype=tf.float32, name=name)
```

Listing 5: Loss function

```
self.loss = tf.reduce_mean(tf.square(self.u0_tf[0:1,0]-self.u0_pred)) + \
                    tf.reduce_mean(tf.square(self.v0_tf[0:1,0]-self.v0_pred)) + \
                    tf.reduce_mean(tf.square(self.u0_tf[1:2,0]-self.u_ub_pred)) + \
                    tf.reduce_mean(tf.square(self.v0_tf[1:2,0]-self.v_ub_pred)) + \
                    tf.reduce_mean(tf.square(self.u0_x_pred)) + \
                    tf.reduce_mean(tf.square(self.v0_x_pred)) + \
                    tf.reduce_mean(tf.square(self.u_x_ub_pred)) + \
                    tf.reduce_mean(tf.square(self.v_x_ub_pred)) + \
                    tf.reduce_mean(tf.square(self.f_u_pred)) + \
                    tf.reduce_mean(tf.square(self.f_v_pred))
```

2. Histograms during network training

Histograms representing the variation of the weights during the training of physics informed neural networks are plotted in Figs. S1a–d, where the number of histogram represents the number of epochs in the training. Variation of the weights between input layer and hidden layers 1, 2, 3 and 4, are indicated by Figs. S1a–d, respectively. In the similar lines, variation in weight distribution with the number of iterations during the training of physics informed neural network are plotted in Figs. S1e–h. Weight distribution between input layer and hidden layers 1, 2, 3, and 4 are shown in Figs. S1e–h, respectively.

Histograms representing the variation of the biases during the training of physics informed neural networks are plotted in Figs. S2a–d, where the number of histogram represents the number of epochs in the training. Variation of the biases between input layer and hidden layers 1, 2, 3, and 4 are indicated by Figs. S2a–d, respectively. In the similar lines, variation in bias distribution with the number of iterations during the training of physics informed neural network are plotted in Figs. S2e–h. Bias distribution between input layer and hidden layers 1, 2, 3, and 4 are shown in Figs. S2e–h, respectively.

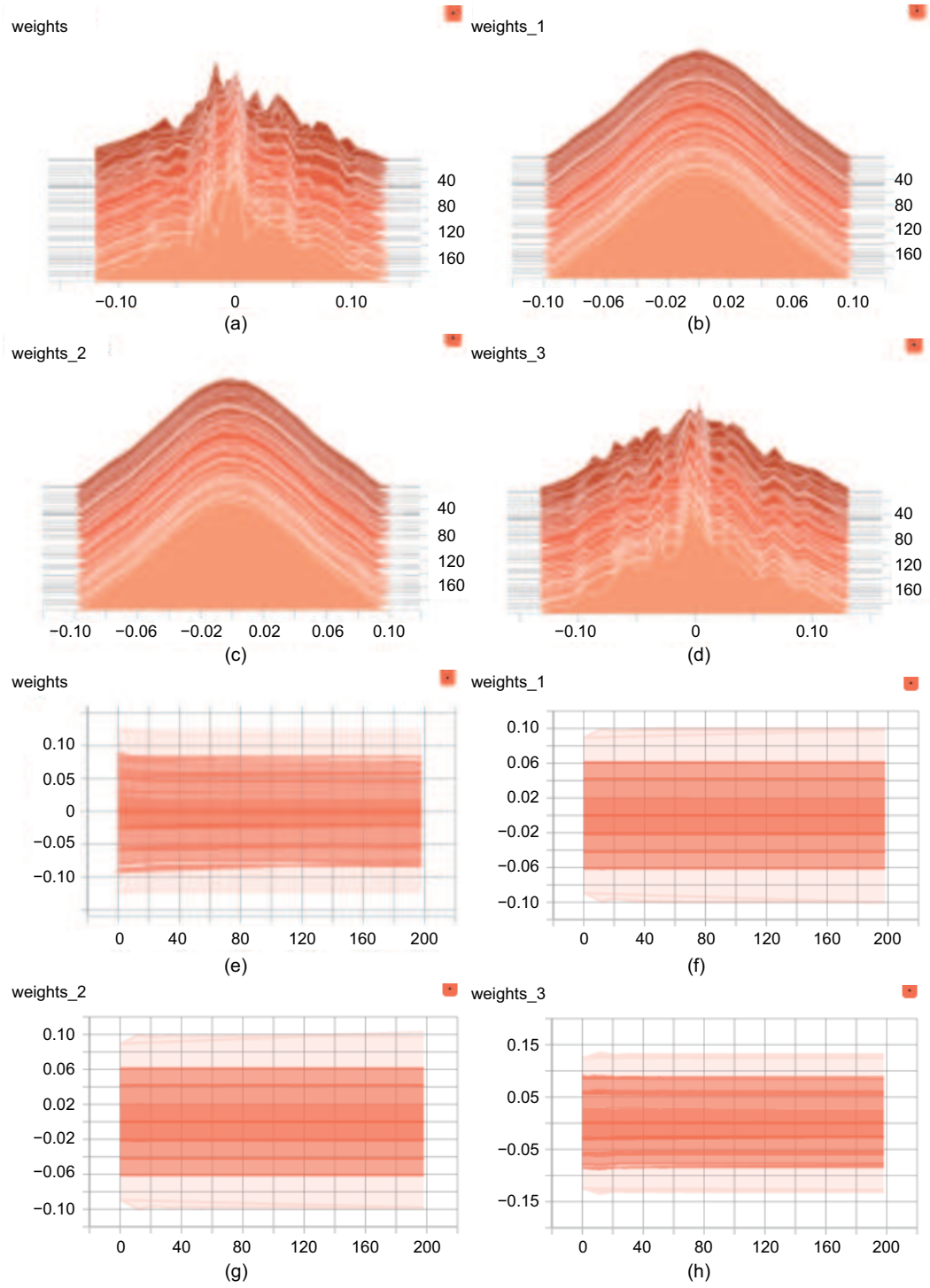

Fig. S1: (a–d) Histograms representing the variation of the weights during the training of PINNs, where the number of histogram represents the number of epochs in the training. Variation of the weights between input layer and hidden layers 1, 2, 3, and 4 are indicated by (a), (b), (c), and (d), respectively. (e–h) Variation in weight distribution with the number of iterations during the training of physics informed neural network. Weight distribution between input layer and hidden layers 1, 2, 3 and 4 are shown in (e), (f), (g), and (h), respectively

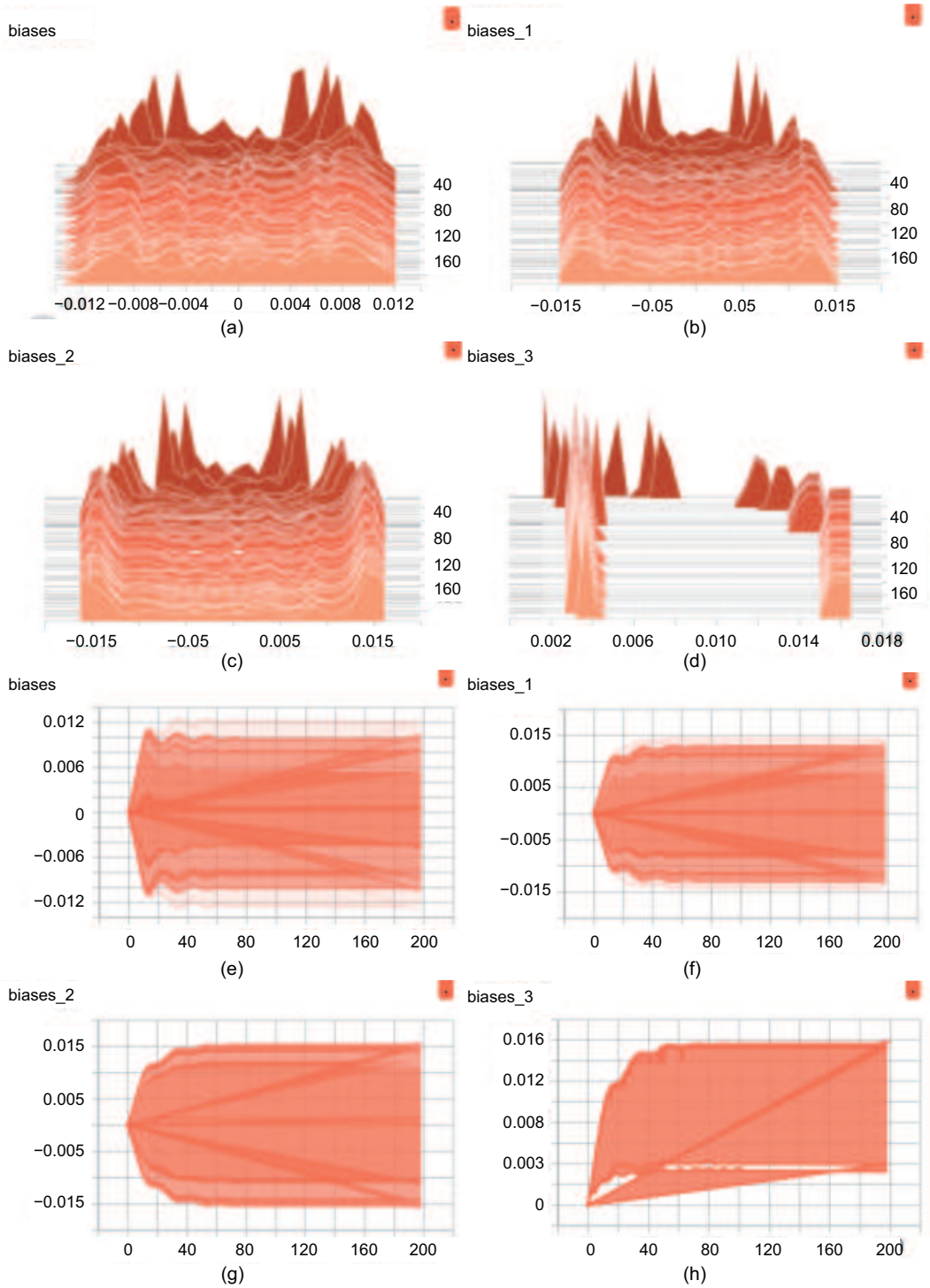

Fig. S2: (a–d) Histograms representing the variation of the biases during the training of PINNs, where the number of histogram represents the number of epochs in the training. Variation of the biases between input layer and hidden layers 1, 2, 3, and 4 are indicated by (a), (b), (c), and (d), respectively. (e–h) Variation in bias distribution with the number of iterations during the training of physics informed neural network. Bias distribution between input layer and hidden layers 1, 2, 3, and 4 are shown in (e), (f), (g), and (h), respectively