Josue Casco-Rodriguez
ELEC 576

Assignment 1

1 Backpropagation in a Simple Neural Network

---

a) Dataset

The first dataset for section 1 consists of 200 points from the scikit-learn Make-Moons Dataset, generated using a random seed of 0 and a noise parameter of 0.20, as shown in Figure 1.

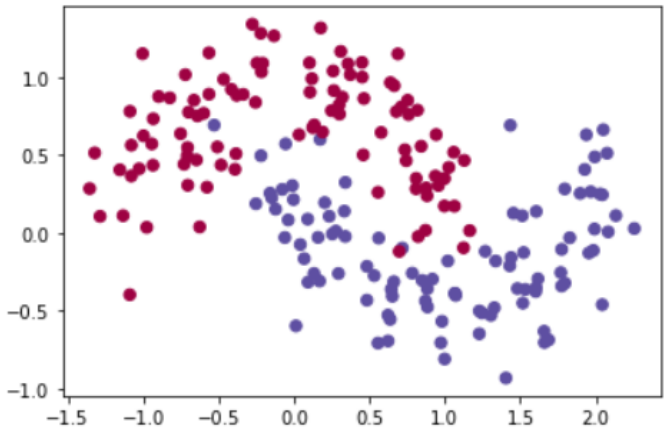
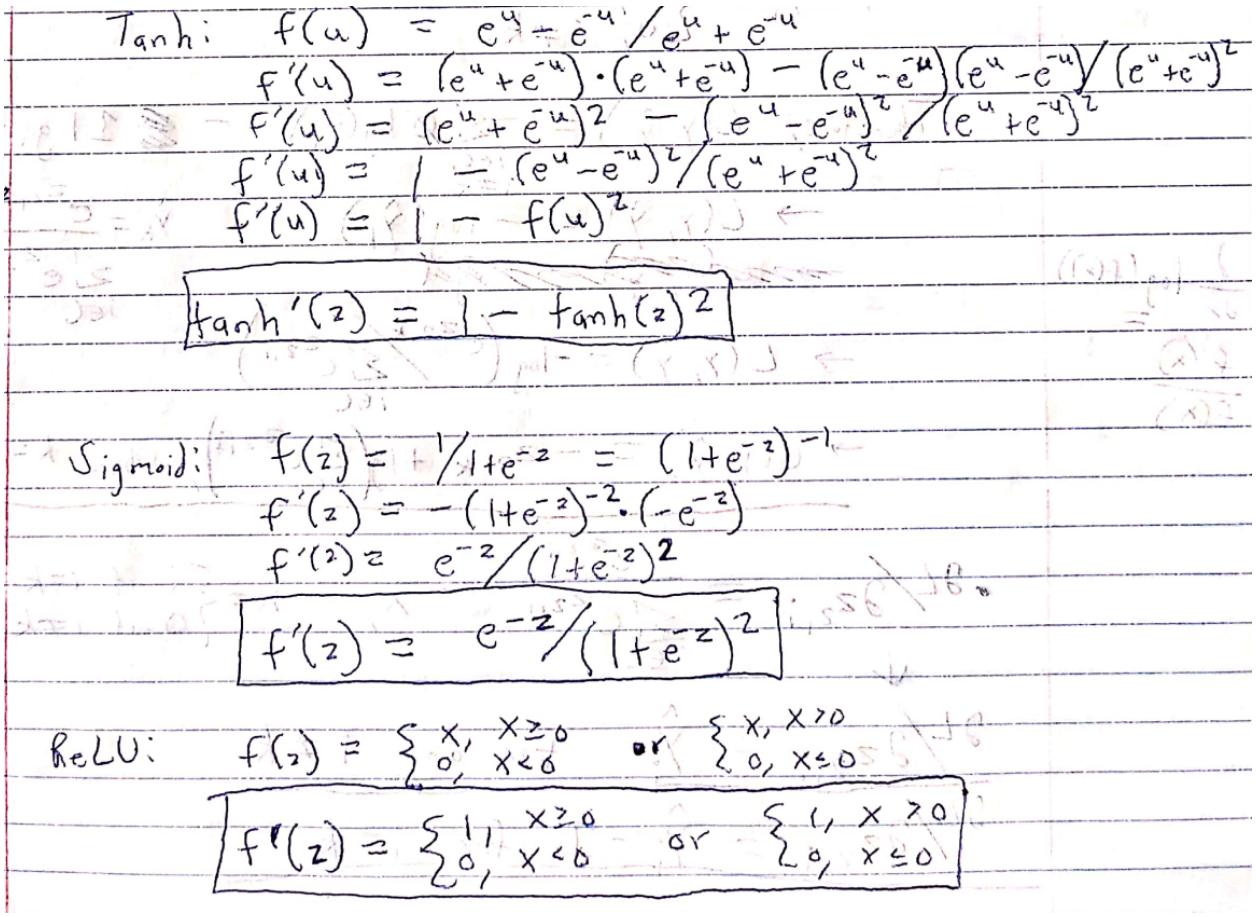


Figure 1: Make-Moons Dataset

b) Activation Function

Derivations of the derivatives:



Tanh: $f(u) = e^u - e^{-u} / e^u + e^{-u}$

$f'(u) = (e^u + e^{-u}) \cdot (e^u + e^{-u}) - (e^u - e^{-u})(e^u - e^{-u}) / (e^u + e^{-u})^2$

$f'(u) = (e^u + e^{-u})^2 - (e^u - e^{-u})^2 / (e^u + e^{-u})^2$

$f'(u) = 1 - (e^u - e^{-u})^2 / (e^u + e^{-u})^2$

$f'(u) = 1 - f(u)^2$

$$\tanh'(z) = 1 - \tanh(z)^2$$

Sigmoid: $f(z) = 1 / 1 + e^{-z} = (1 + e^{-z})^{-1}$

$f'(z) = -(1 + e^{-z})^{-2} \cdot (-e^{-z})$

$f'(z) = e^{-z} / (1 + e^{-z})^2$

$$f'(z) = e^{-z} / (1 + e^{-z})^2$$

ReLU: $f(z) = \begin{cases} x, & x \geq 0 \\ 0, & x < 0 \end{cases}$ or $\begin{cases} x, & x > 0 \\ 0, & x \leq 0 \end{cases}$

$$f'(z) = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases} \text{ or } \begin{cases} 1, & x > 0 \\ 0, & x \leq 0 \end{cases}$$

actFun and diff_actFun implementations:

```python
def actFun(self, z, actFun_type):
    '''
    actFun computes the activation functions
    :param z: net input
    :param type: Tanh, Sigmoid, or ReLU
    :return: activations
    '''

    # YOU IMPLEMENT YOUR actFun HERE
    if actFun_type.lower() == 'tanh':
        return np.tanh(z)

    if actFun_type.lower() == 'sigmoid':
        return 1 / (1 + np.exp(-z))

    if actFun_type.lower() == 'relu':
        return (z > 0) * z

    if actFun_type.lower() == 'softmax':
        exp_scores = np.exp(z)
        return exp_scores / exp_scores.sum(axis=1, keepdims=True)

    return None
```

```python
def diff_actFun(self, z, actFun_type):
    '''
    diff_actFun computes the derivatives of the activation functions wrt the net input
    :param z: net input
    :param type: Tanh, Sigmoid, or ReLU
    :return: the derivatives of the activation functions wrt the net input
    '''

    # YOU IMPLEMENT YOUR diff_actFun HERE

    if actFun_type.lower() == 'tanh':
        return 1 - np.tanh(z)**2

    if actFun_type.lower() == 'sigmoid':
        s = 1 / (1 + np.exp(-z))
        return s * (1 - s)

    if actFun_type.lower() == 'relu':
        return z > 0

    if actFun_type.lower() == 'softmax':
        return z

    return None
```

c) Build the Neural Network

```
def feedforward(self, X, actFun):
    '''
    feedforward builds a 3-layer neural network and computes the two probabilities,
    one for class 0 and one for class 1
    :param X: input data
    :param actFun: activation function
    :return:
    '''

    # YOU IMPLEMENT YOUR feedforward HERE

    self.z1 = X.dot(self.W1) + self.b1 # np.dot(self.W1, X) + self.b1
    self.a1 = actFun(self.z1)
    self.z2 = self.a1.dot(self.W2) + self.b2 # np.dot(self.W2, a1) + self.b2
    exp_scores = np.exp(self.z2)
    self.probs = exp_scores / np.sum(exp_scores, axis=1, keepdims=True)
    return None

def calculate_loss(self, X, y):
    '''
    calculate_loss computes the loss for prediction
    :param X: input data
    :param y: given labels
    :return: the loss for prediction
    '''
    num_examples = len(X)
    self.feedforward(X, lambda x: self.actFun(x, actFun_type=self.actFun_type))
    # Calculating the loss

    # YOU IMPLEMENT YOUR CALCULATION OF THE LOSS HERE

    data_loss = -np.log(self.probs.max(axis=1)).sum()

    # Add regulatization term to loss (optional)
    data_loss += self.reg_lambda / 2 * (np.sum(np.square(self.W1)) + np.sum(np.square(self.W2)))
    return (1. / num_examples) * data_loss
```

d) Backward Pass - Backpropagation

[see back end of this PDF for the derivations of the loss terms]

Backpropagation code:

```python
def backprop(self, X, y):
    '''
    backprop implements backpropagation to compute the gradients used to update the parameters in the backward step
    :param X: input data
    :param y: given labels
    :return: dL/dW1, dL/b1, dL/dW2, dL/db2
    '''

    # IMPLEMENT YOUR BACKPROP HERE
    num_examples = len(X)
    delta3 = self.probs.copy()
    delta3[range(num_examples), y] -= 1

    self.dW2 = self.a1.T.dot(delta3)
    self.db2 = delta3.sum(axis=0, keepdims=True)

    self.dz1 = self.diff_actFun(self.z1, self.actFun_type) * delta3.dot(self.W2.T)

    self.dW1 = X.T.dot(self.dz1)
    self.db1 = self.dz1.sum(axis=0, keepdims=True)

    return self.dW1, self.dW2, self.db1, self.db2
```

e) Time to Have Fun - Training!

In three_layer_neural_network.py, training the simple network for 20,000 epochs with three different activation functions (Tanh, Sigmoid, and ReLU) yielded three distinct decision boundaries, as shown in Figure 2. In accordance with modern understandings of neural networks as high-dimensional spline-fitting methods, the network fits a decision boundary spline using the training method. Interestingly, usage of the rectified linear unit yields a decision boundary that consists of a piecewise series of lines, while using the two smooth and continuously-differentiable activation functions yields a much smoother decision boundary that better captures the true pattern underlying the data.

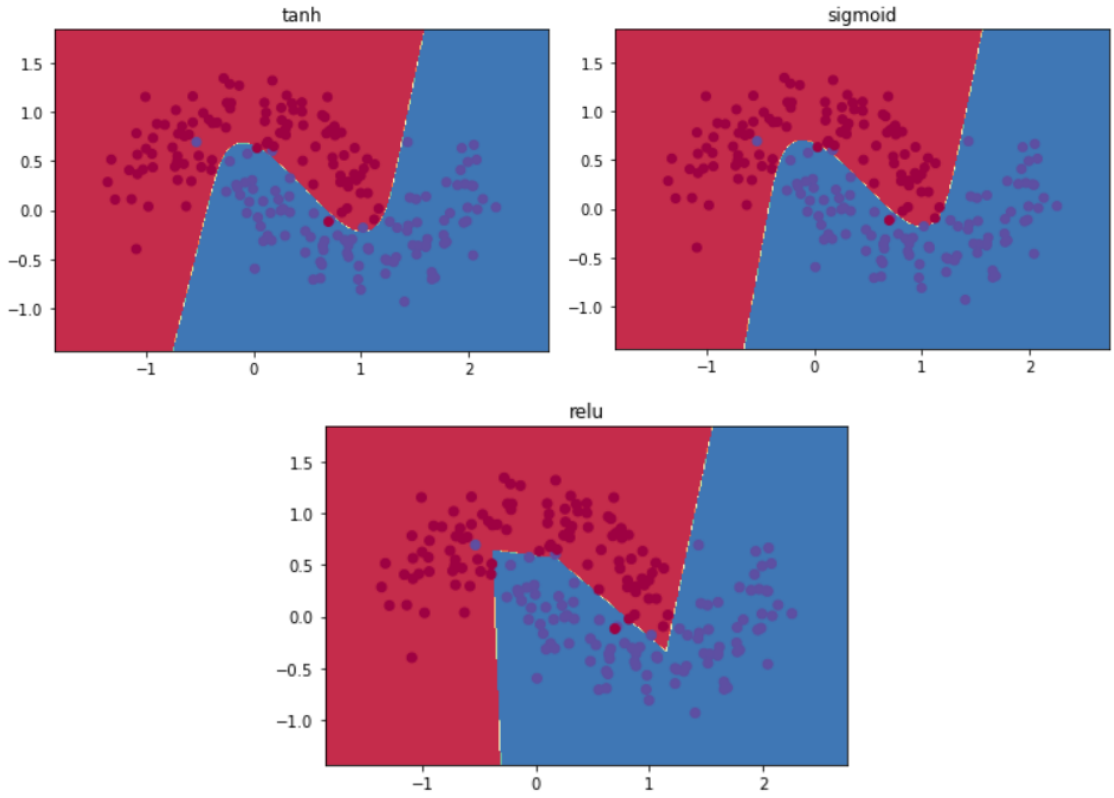


Figure 2: Decision Boundaries Produced by Three Activation Functions

Increasing the number of hidden dimensions (i.e., the hidden layer size) using a tanh activation function, the network produced decision boundaries that were able to better fit the specific set of training data given to it, at the cost of producing a more generalizable and intuitive shape resembling the Make-Moons Dataset. Decision boundaries produced by 4, 10, and 20 hidden units are shown in Figure 3.

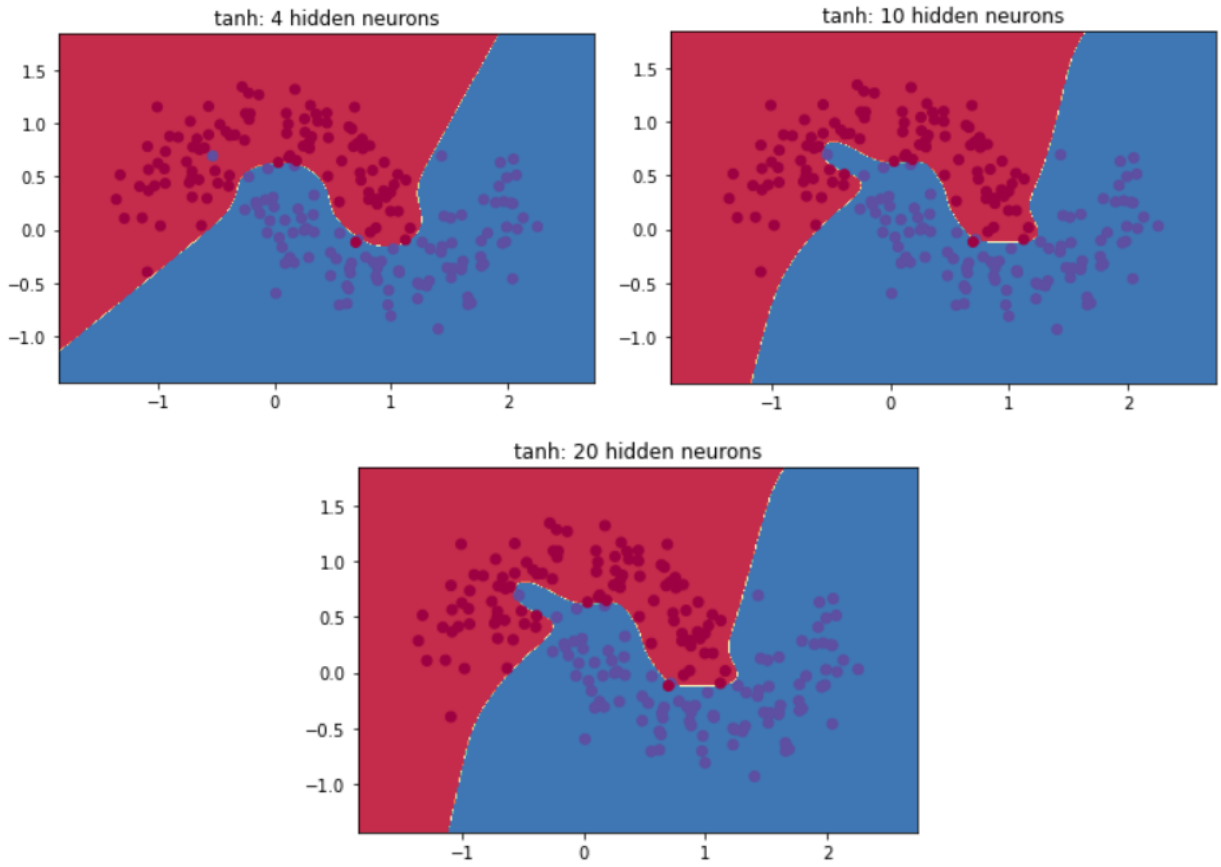


Figure 3: Decision Boundary Produced By Varying Hidden Dimension Sizes

f) Even More Fun - Training a Deeper Network!!!

For speed and ease of training, the number of epochs have been reduced to 10,000, since no significant improvements in accuracy occur after roughly 10,000 epochs for the architectures shown below. The resulting decision boundaries of various architectures are in Figure 4. One observation is that rectified linear activation functions train faster than tanh or sigmoid activation functions, since they are computationally simpler. Another key observation is that If the layer size (i.e., the hidden dimension) is not varied, then decision boundaries are often not affected by the number of layers. Instead, layer size impacts dictionary boundary variations more than the number of layers. Additionally, as shown in Figure 4, the tanh activation function is better able to produce discontinuous decision boundaries than the sigmoid or rectified linear activations.

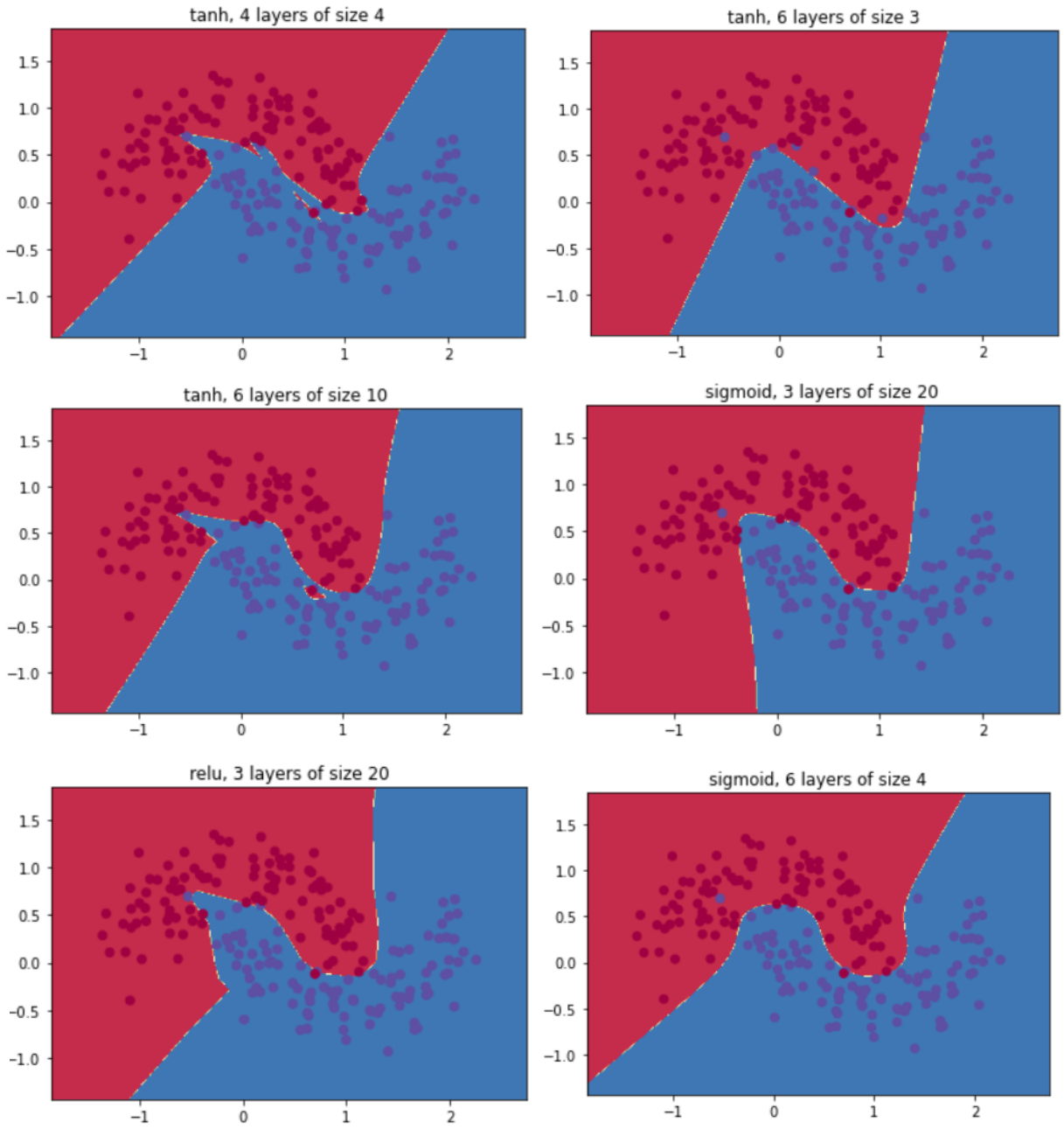


Figure 4: Decision Boundaries Produced By Various Neural Architectures

Again using 10,000 epochs, the existing neural network framework is applied to a new dataset: the scikit-learn make_gaussian_quantiles dataset. Specifically, the dataset consists of 4 two-dimensional concentric circular regions of random variables, as shown in Figure 5. This dataset is of notable interest because any decision boundary that accurately captures its dynamics should be discontinuous and nonlinear.

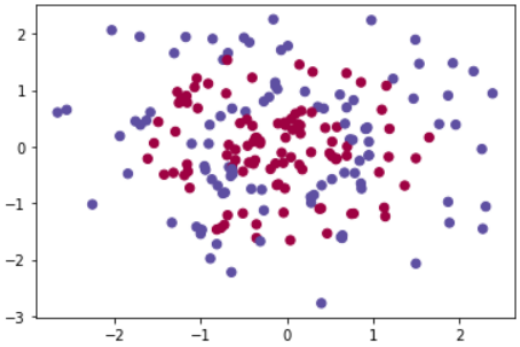
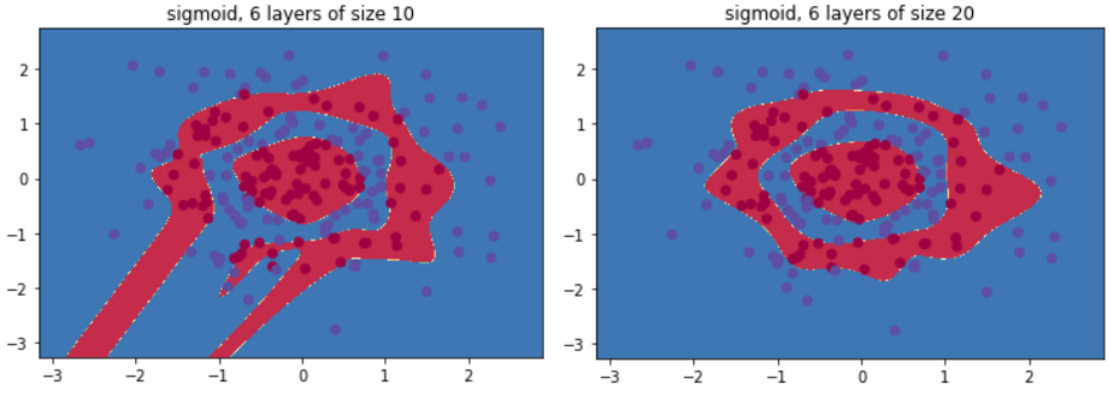


Figure 5: Gaussian Quantile dataset.

As with the previous dataset, the key to improving successfully fitting the neural architecture to the data is hidden dimension size (i.e., layer size) instead of layer count. Decision boundaries are shown in Figure 6. Perfect accuracy is achieved using a sigmoid activation of 6 size-20 layers, a tanh activation of 4 size-25 layers, and a tanh activation of 6 size-15 layers. Interestingly, some decision boundaries have inner (red) sections that extend well beyond the true range of the second outermost circular cluster, while others have inner sections that stay within the outermost (blue) circular cluster. Additionally, in order to investigate the impact of layer size, neural networks of 3 size-100 layers are used with sigmoid, tanh, and rectified linear activations. Perhaps surprisingly, the shallow rectified linear network achieves near-perfect accuracy outperforms the shallow tanh and shallow sigmoid networks by at least 10% and 20% accuracy, respectively.



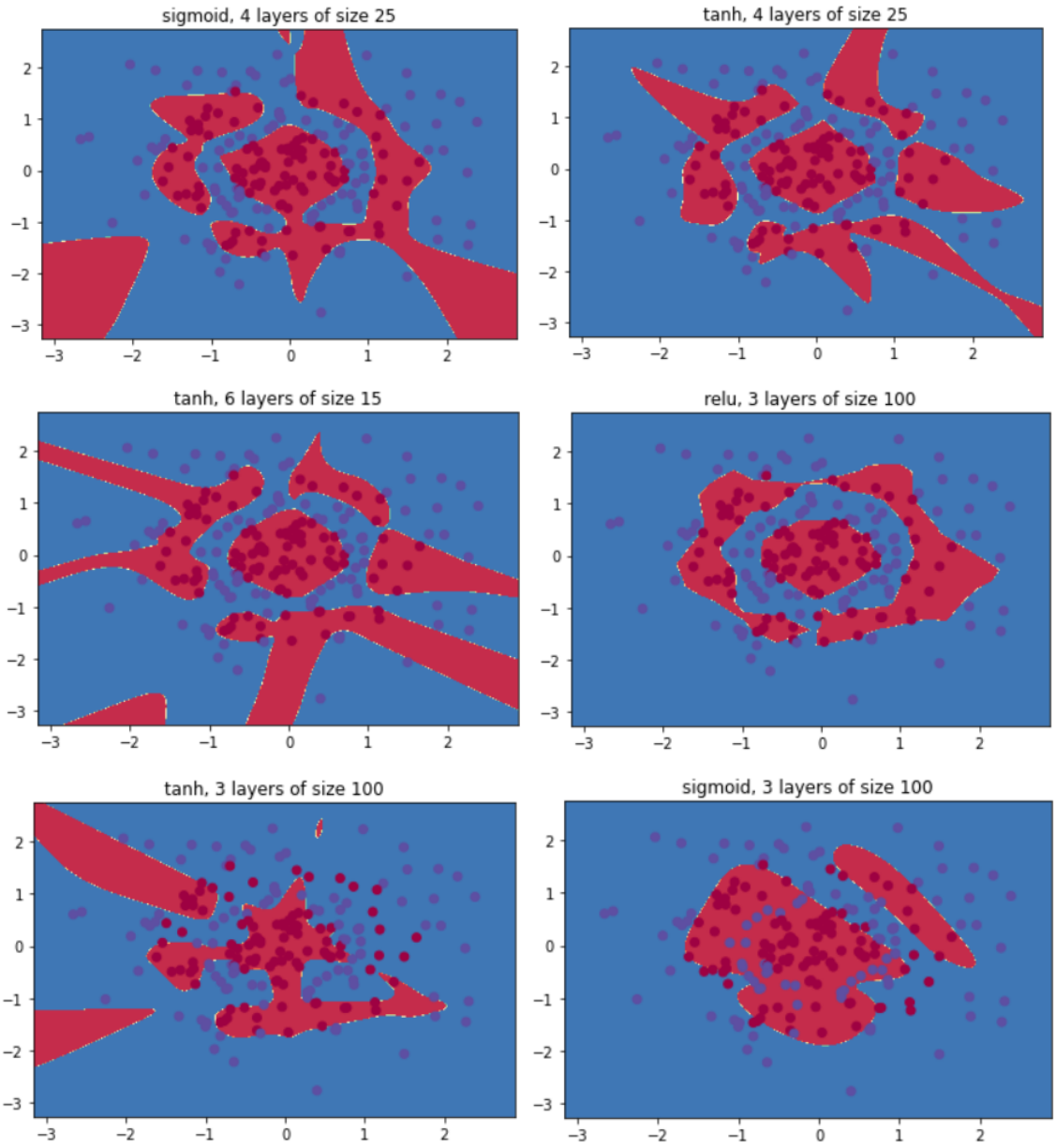


Figure 6: Decision Boundaries on Gaussian Quantile Dataset

2 Training a Simple Deep Convolutional Network on MNIST

---

a) Build and Train a 4-layer DCN

The MNIST image recognition dataset is fitted using a LeNet5 architecture, using rectified linear (ReLU) activation, Truncated Normal weight initialization, and Adam optimization. The dataset consists of 60,000 training images and 10,000 test images, with each image being of 28 x 28 pixels in size. For the purposes of rigorous evaluation, the 60,000 training images are split into a 55,000-image training set and a 5,000-image validation set. The results of training are shown in Figure 7.

b) More on Visualizing Your Training

The histograms, evaluated after every 100 batches, associated with the inputs, outputs, weights (if applicable), and biases (if applicable) of each layer of the aforementioned LeNet5 architecture are shown in Figure 8. Interestingly, the distributions of weights maintain their shape, while the distributions of biases go from a highly clustered shape to a more uniform, distributed shape. As for the statistics (minimum, maximum, mean, and standard deviation) of the inputs, outputs, weights, and biases of each layer, they are in Figure 9. Specifically, they are plotted such that the central black trendline is the mean, the thick and short black lines are the standard deviation from the mean, and the remaining thin lines are the range of the data (maximum - minimum).

c) Time for More Fun!!

Parts (a) and (b) are repeated for the LeNet5 architecture using ReLU, tanh, sigmoid, leaky ReLU, and exponential linear unit (ELU) activations; Truncated Normal, Xavier Uniform, and Xavier Normal weight initializations; and Adam, Stochastic Gradient Descent, Nadam, and Adagrad optimization methods. While they all perform fairly similarly thanks to the well-crafted design of the LeNet5 architecture, usage of the Nadam optimization method seems to yield the best performance, showing a test set accuracy of almost 99.2%. The figures depicting the training and accuracies of the aforementioned LeNet5 architectural variants are shown in Figures 7 - 36.

ReLU:

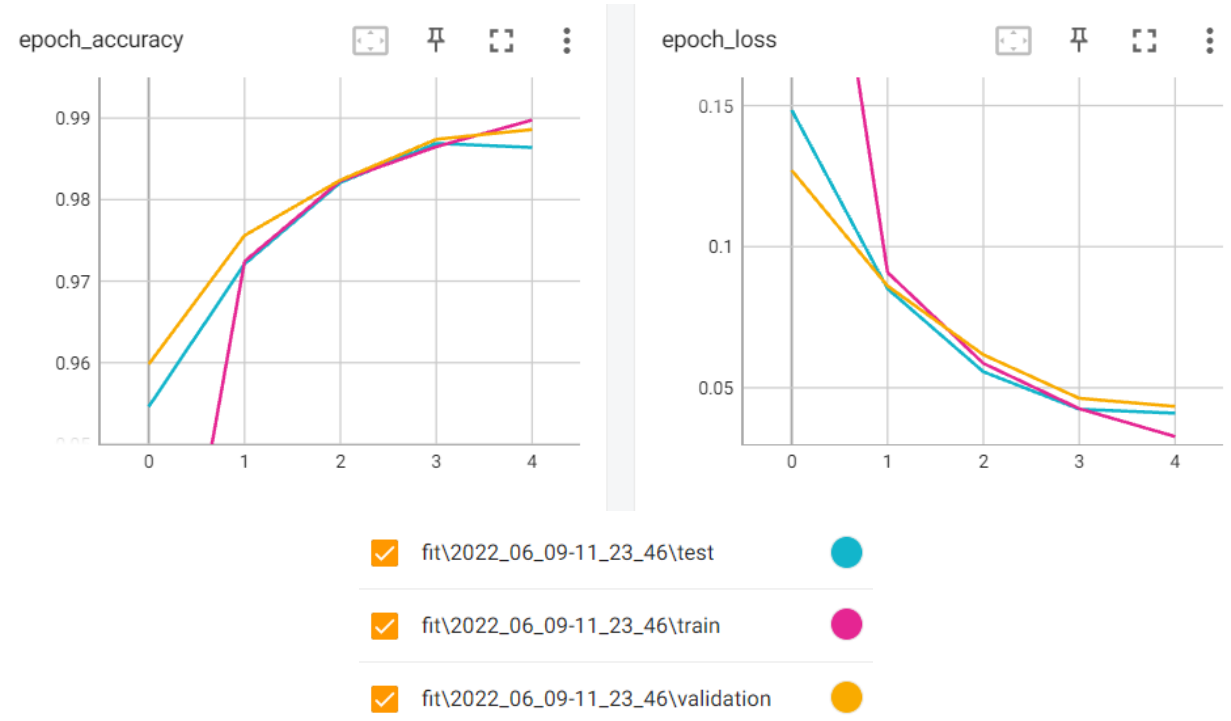
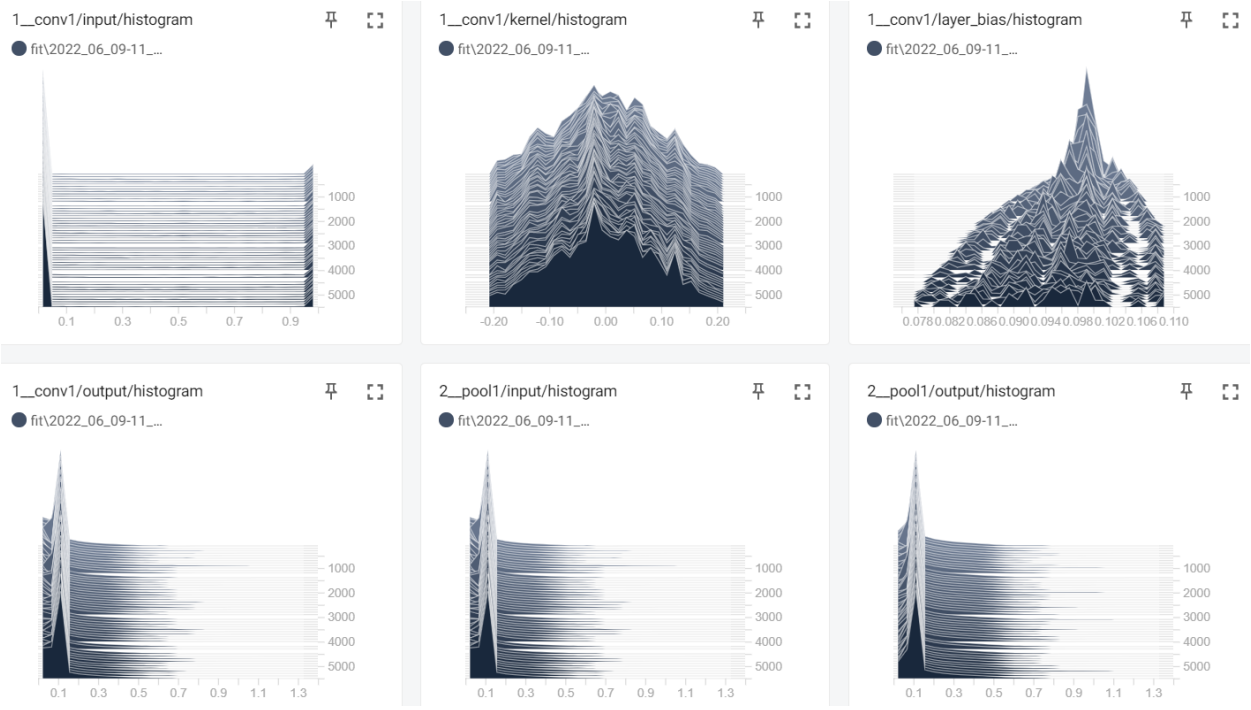
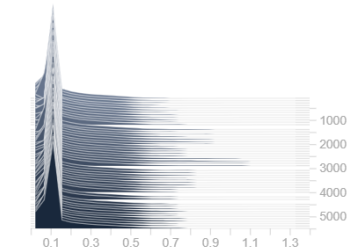


Figure 7: Accuracy and Loss from ReLU Activation



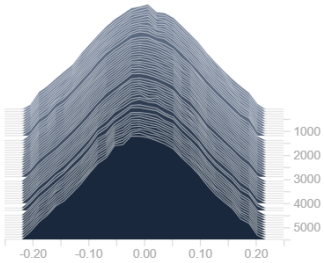
3__conv2/input/histogram

fit\2022_06_09-11_...



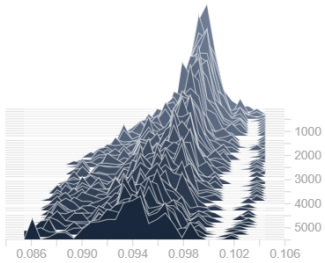
3__conv2/kernel/histogram

fit\2022_06_09-11_...



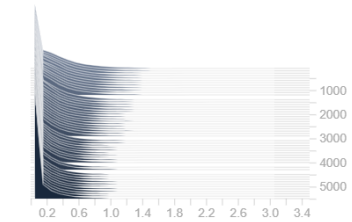
3__conv2/layer_bias/histogram

fit\2022_06_09-11_...



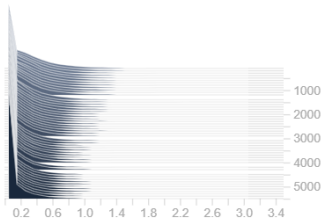
3__conv2/output/histogram

fit\2022_06_09-11_...



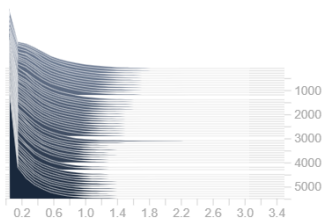
4__pool2/input/histogram

fit\2022_06_09-11_...



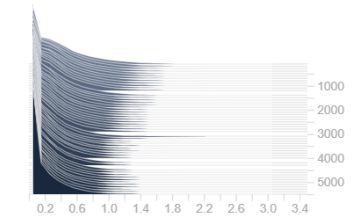
4__pool2/output/histogram

fit\2022_06_09-11_...



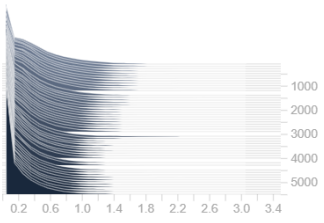
5__reshape/input/histogram

fit\2022_06_09-11_...



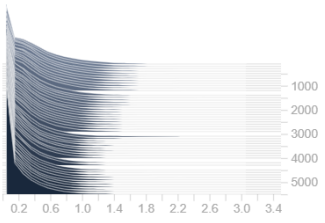
5__reshape/output/histogram

fit\2022_06_09-11_...



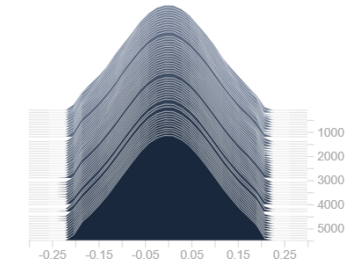
6__dense1/input/histogram

fit\2022_06_09-11_...



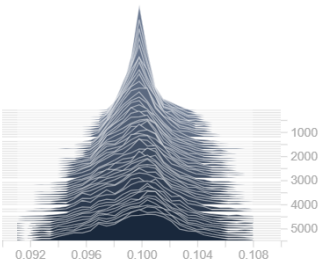
6__dense1/kernel/histogram

fit\2022_06_09-11_...



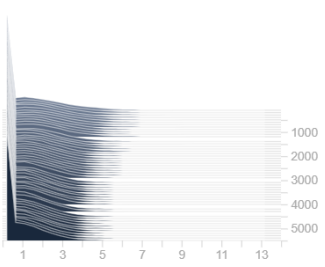
6__dense1/layer_bias/histogram

fit\2022_06_09-11_...



6__dense1/output/histogram

fit\2022_06_09-11_...



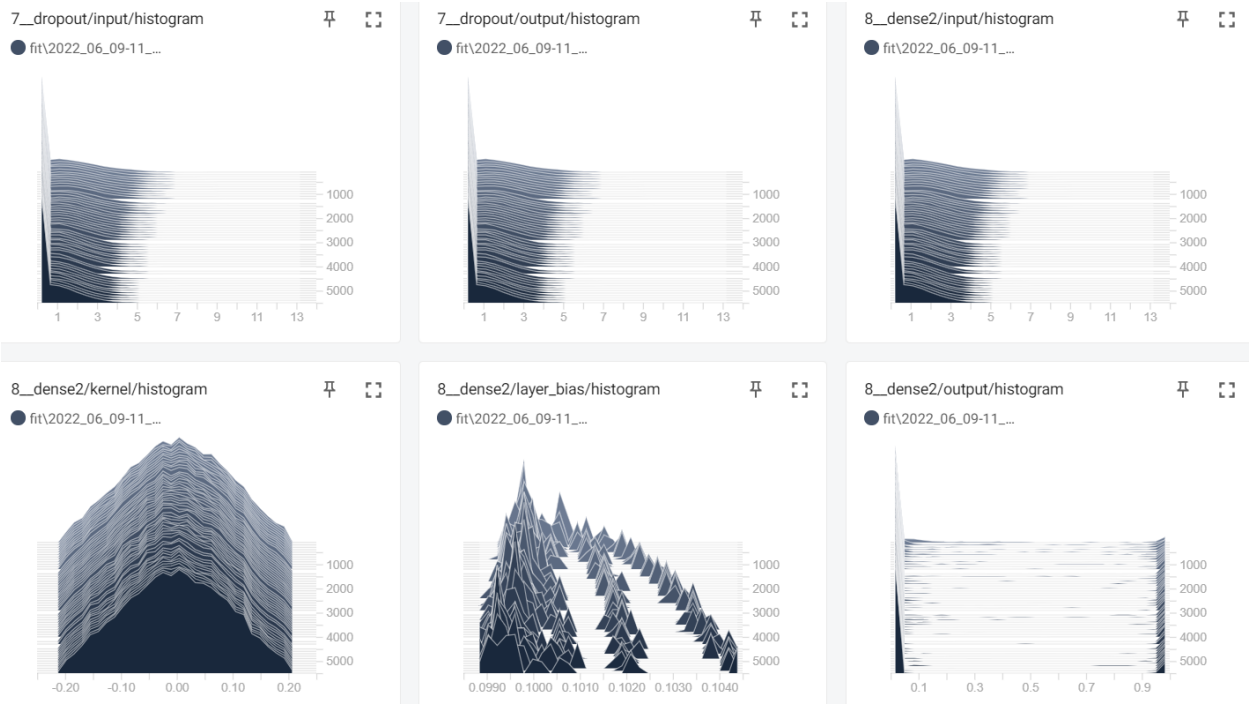


Figure 8: Layer Input, Output, Weight and Bias Histograms from ReLU Activation

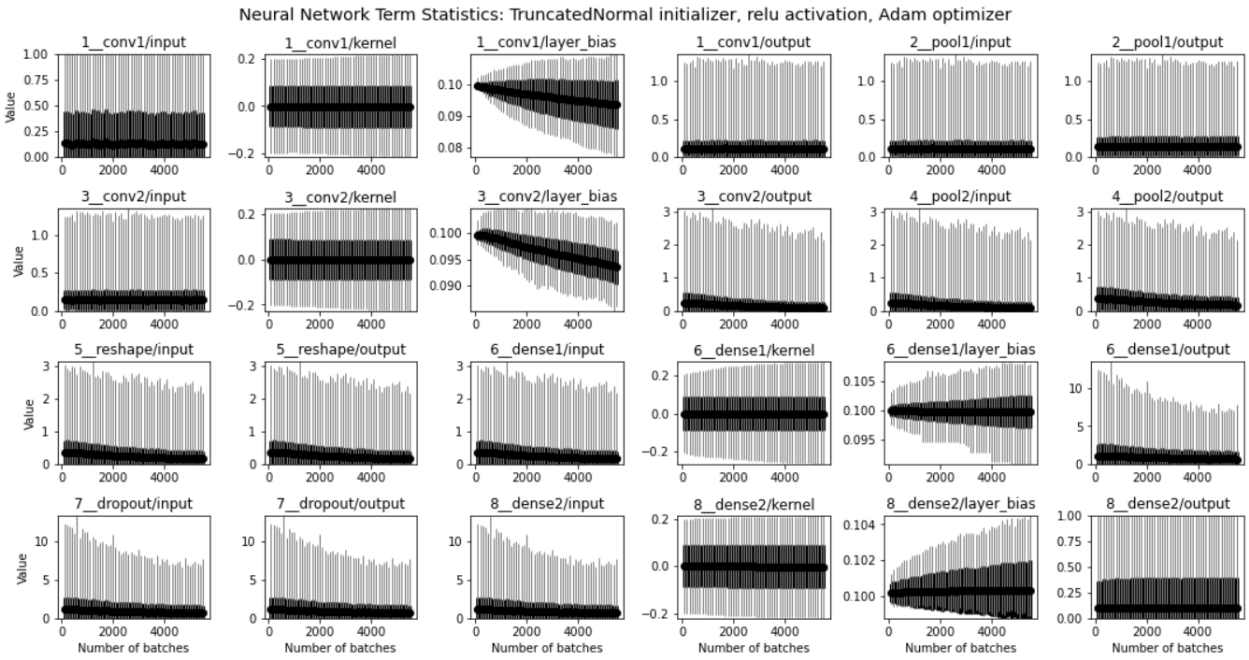


Figure 9: Layer Input, Output, Weight and Bias Statistics from ReLU Activation

Tanh:

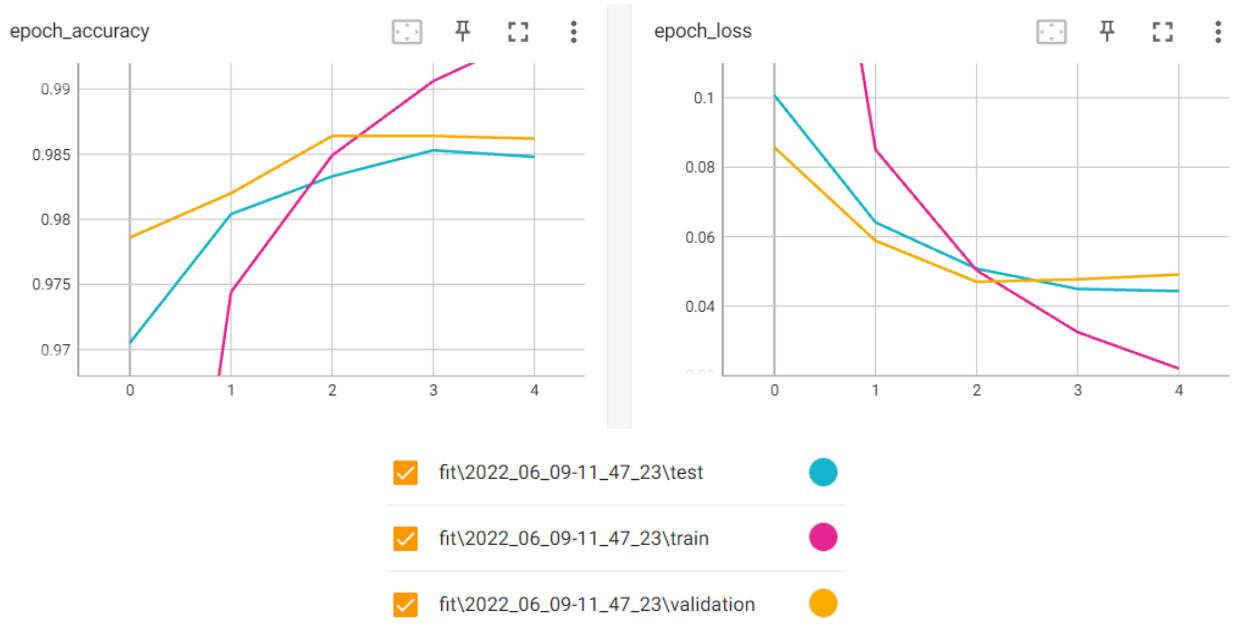
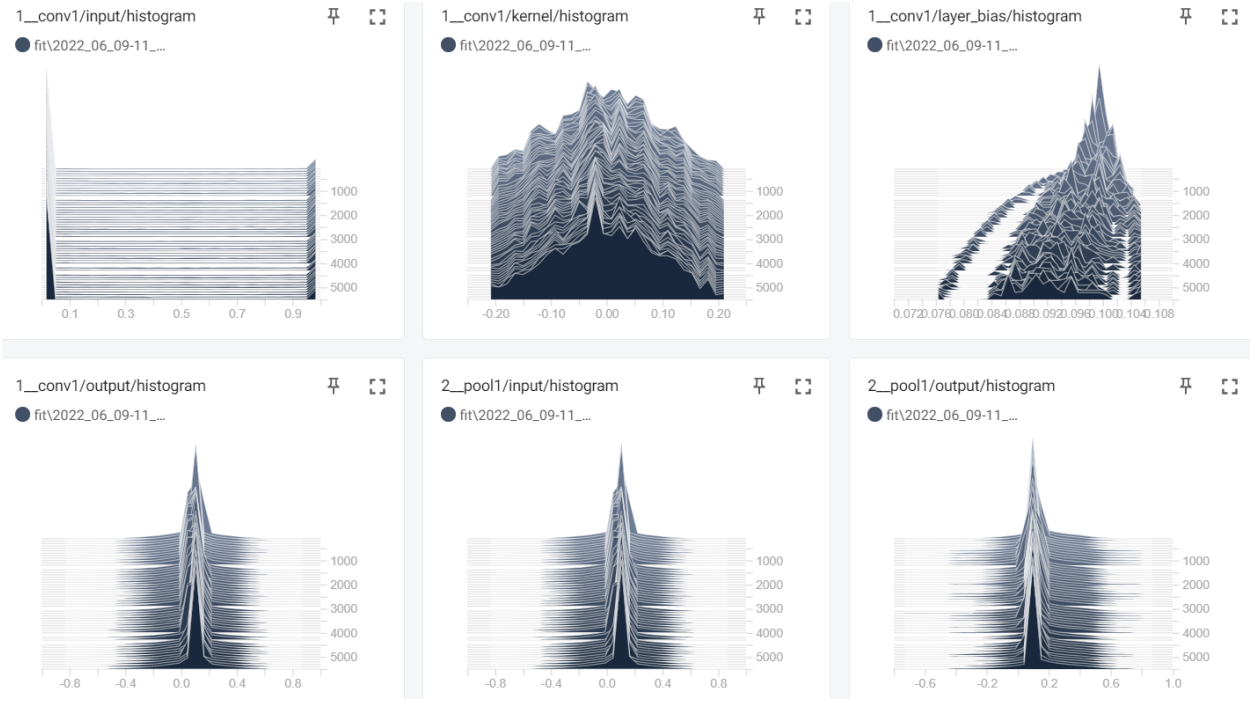
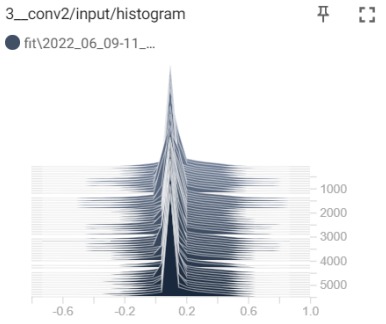


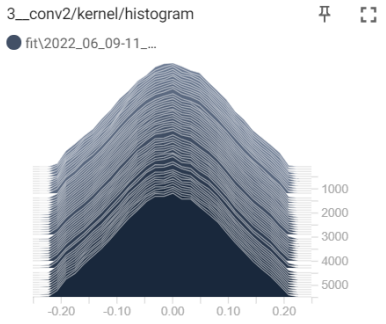
Figure 10: Accuracy and Loss from Tanh Activation Function



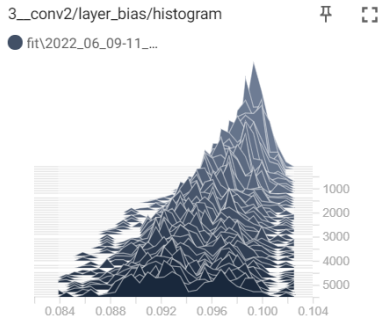
3__conv2/input/histogram



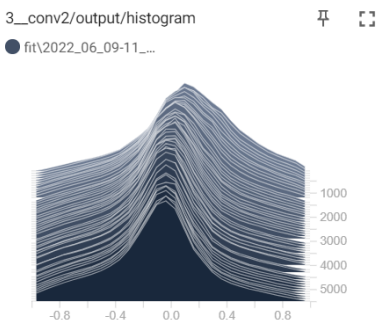
3__conv2/kernel/histogram



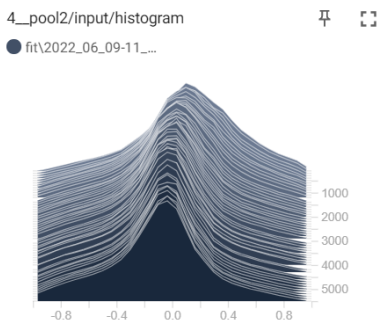
3__conv2/layer_bias/histogram



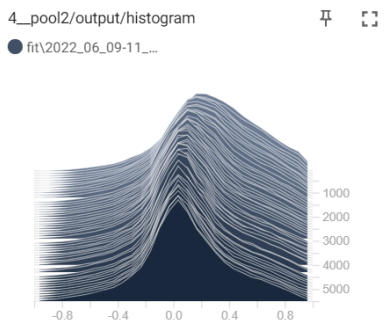
3__conv2/output/histogram



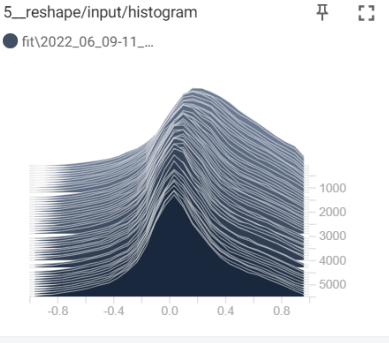
4__pool2/input/histogram



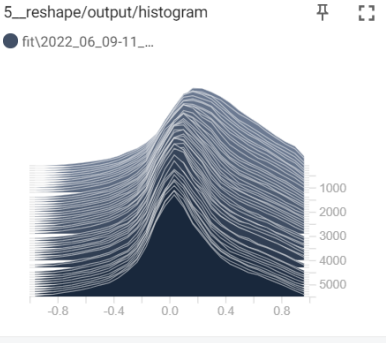
4__pool2/output/histogram



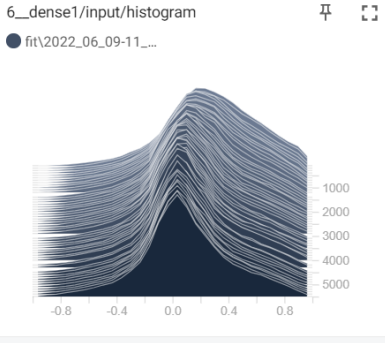
5__reshape/input/histogram



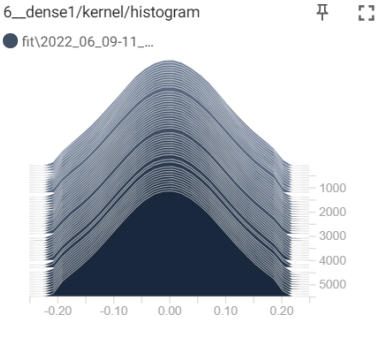
5__reshape/output/histogram



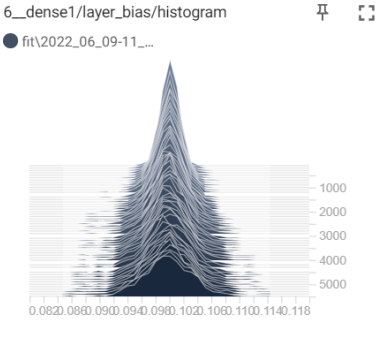
6__dense1/input/histogram



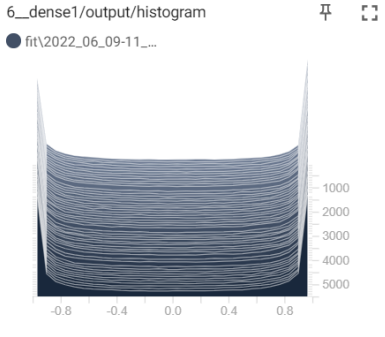
6__dense1/kernel/histogram



6__dense1/layer_bias/histogram



6__dense1/output/histogram



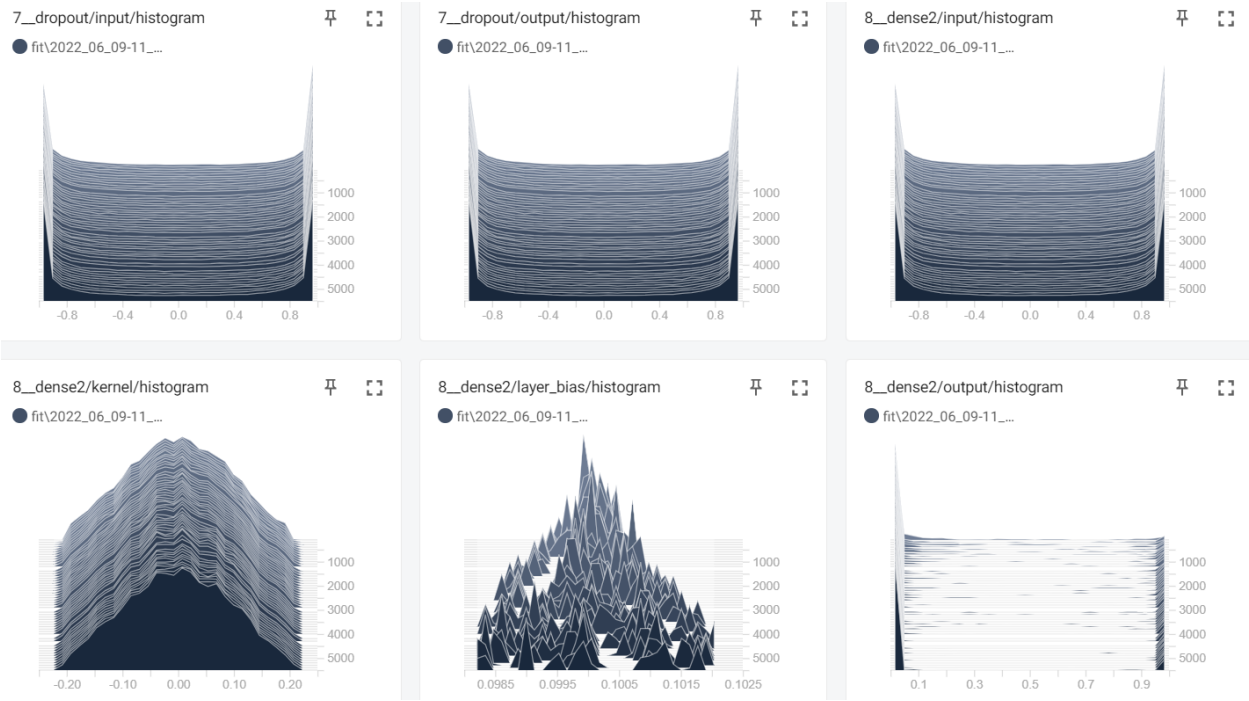


Figure 11: Layer Input, Output, Weight and Bias Histograms from Tanh Activation

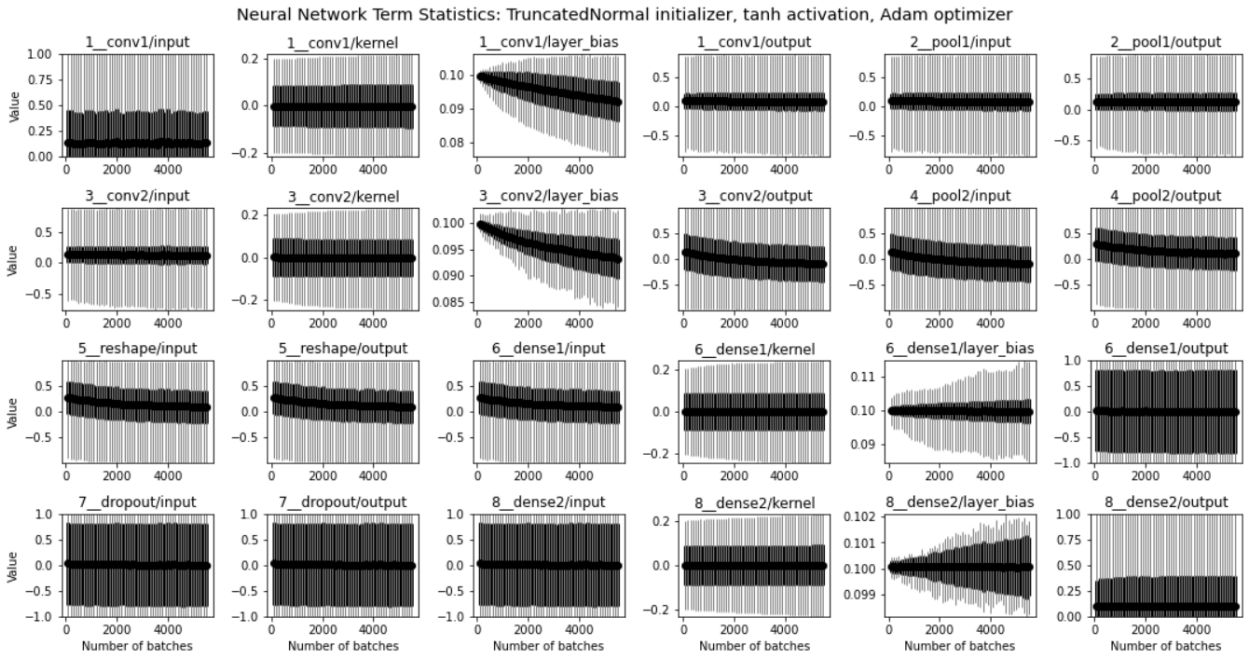


Figure 12: Layer Input, Output, Weight and Bias Statistics from Tanh Activation

Sigmoid:

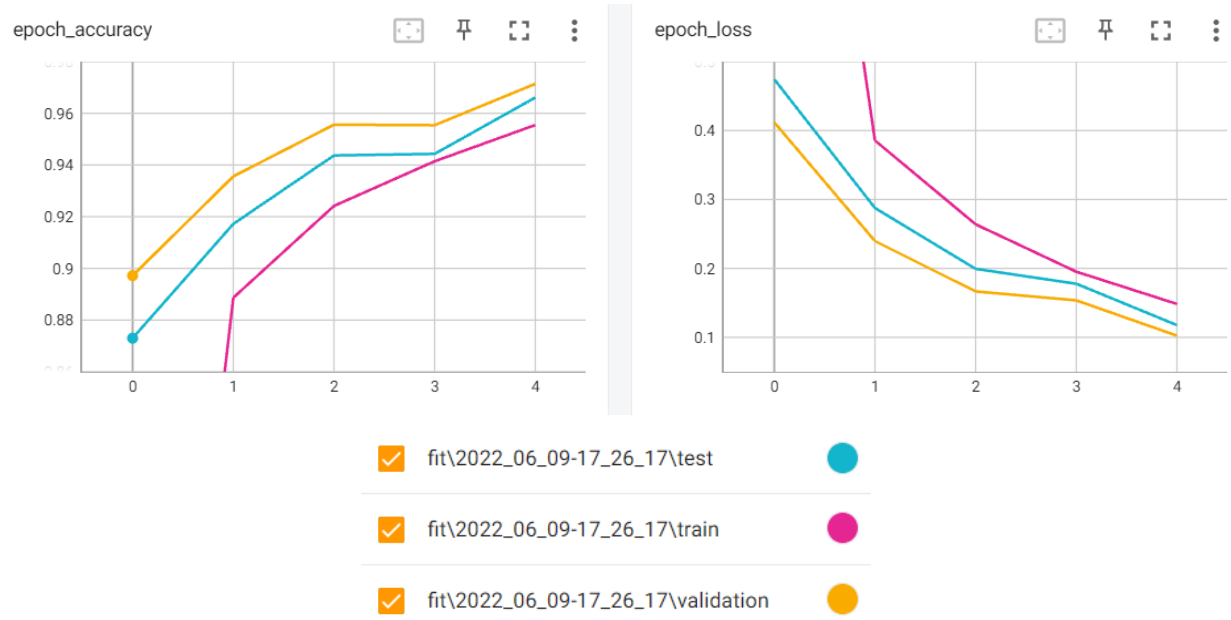
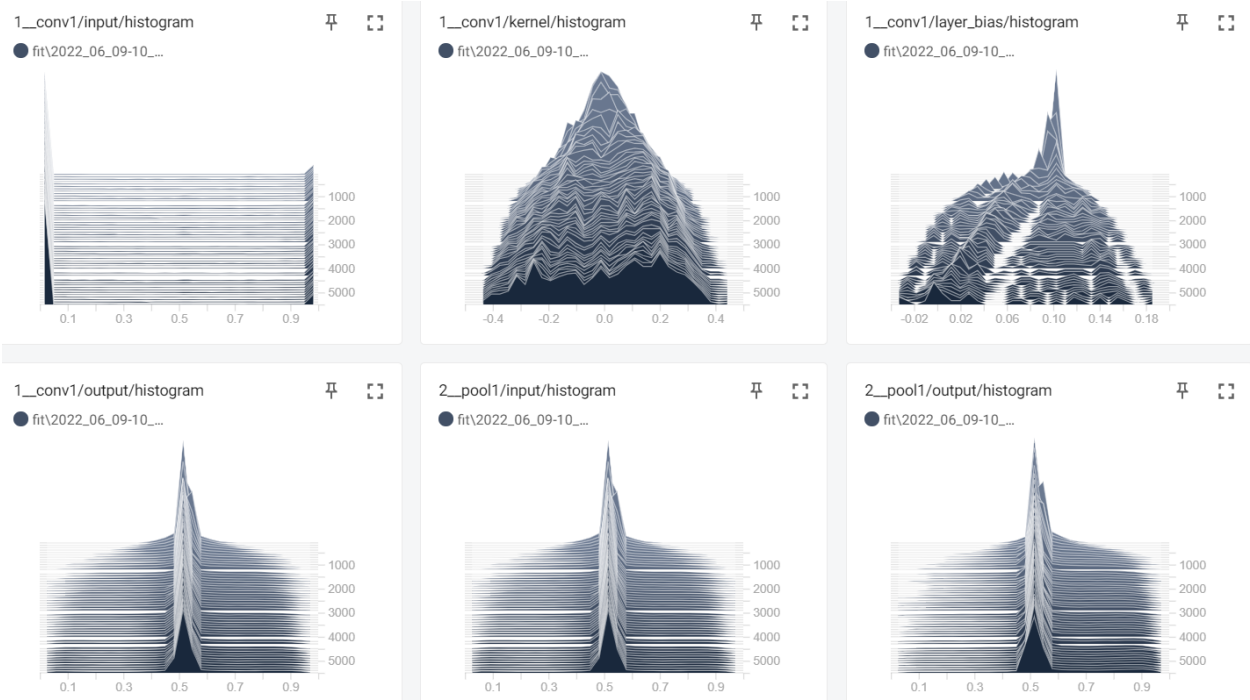
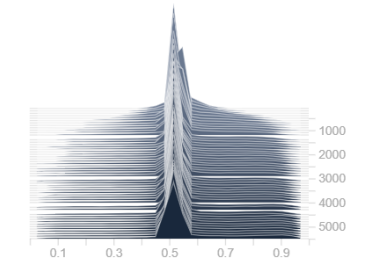


Figure 13: Accuracy and Loss from Sigmoid Activation



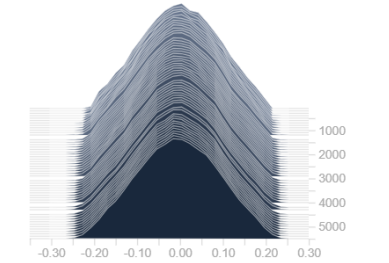
3__conv2/input/histogram

fit\2022_06_09-10_...



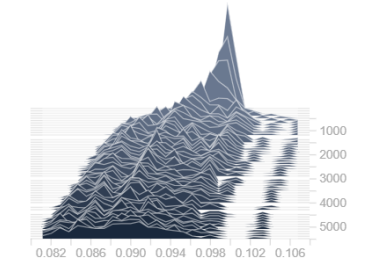
3__conv2/kernel/histogram

fit\2022_06_09-10_...



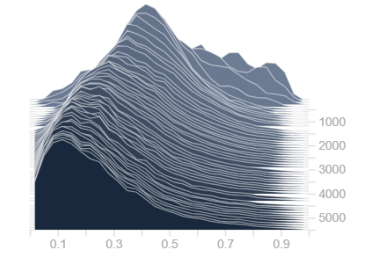
3__conv2/layer_bias/histogram

fit\2022_06_09-10_...



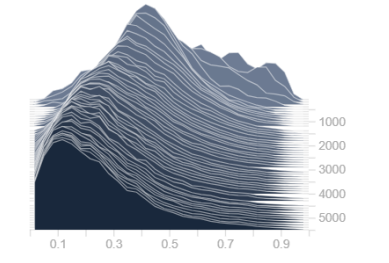
3__conv2/output/histogram

fit\2022_06_09-10_...



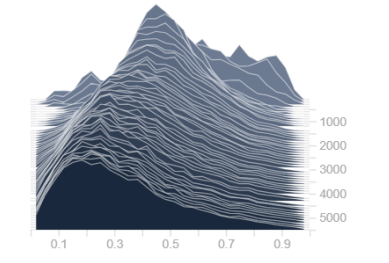
4__pool2/input/histogram

fit\2022_06_09-10_...



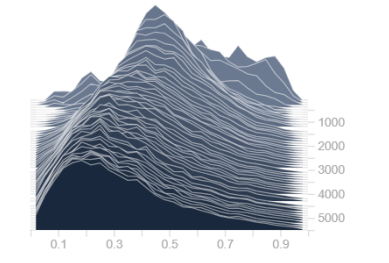
4__pool2/output/histogram

fit\2022_06_09-10_...



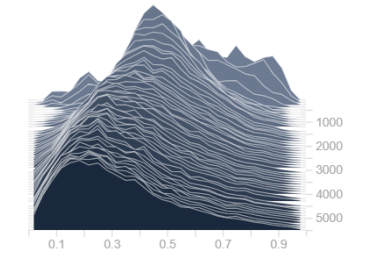
5__reshape/input/histogram

fit\2022_06_09-10_...



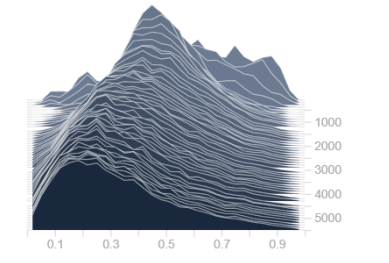
5__reshape/output/histogram

fit\2022_06_09-10_...



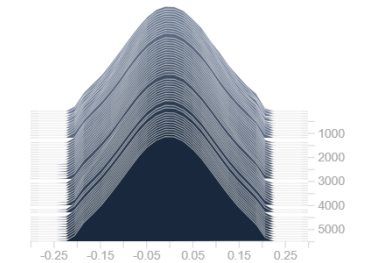
6__dense1/input/histogram

fit\2022_06_09-10_...



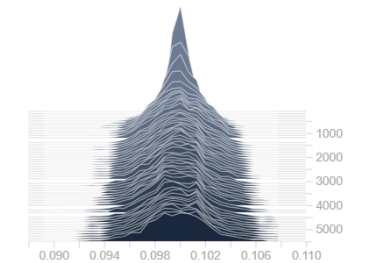
6__dense1/kernel/histogram

fit\2022_06_09-10_...



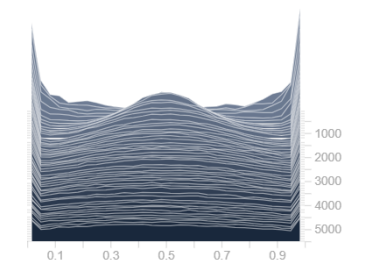
6__dense1/layer_bias/histogram

fit\2022_06_09-10_...



6__dense1/output/histogram

fit\2022_06_09-10_...



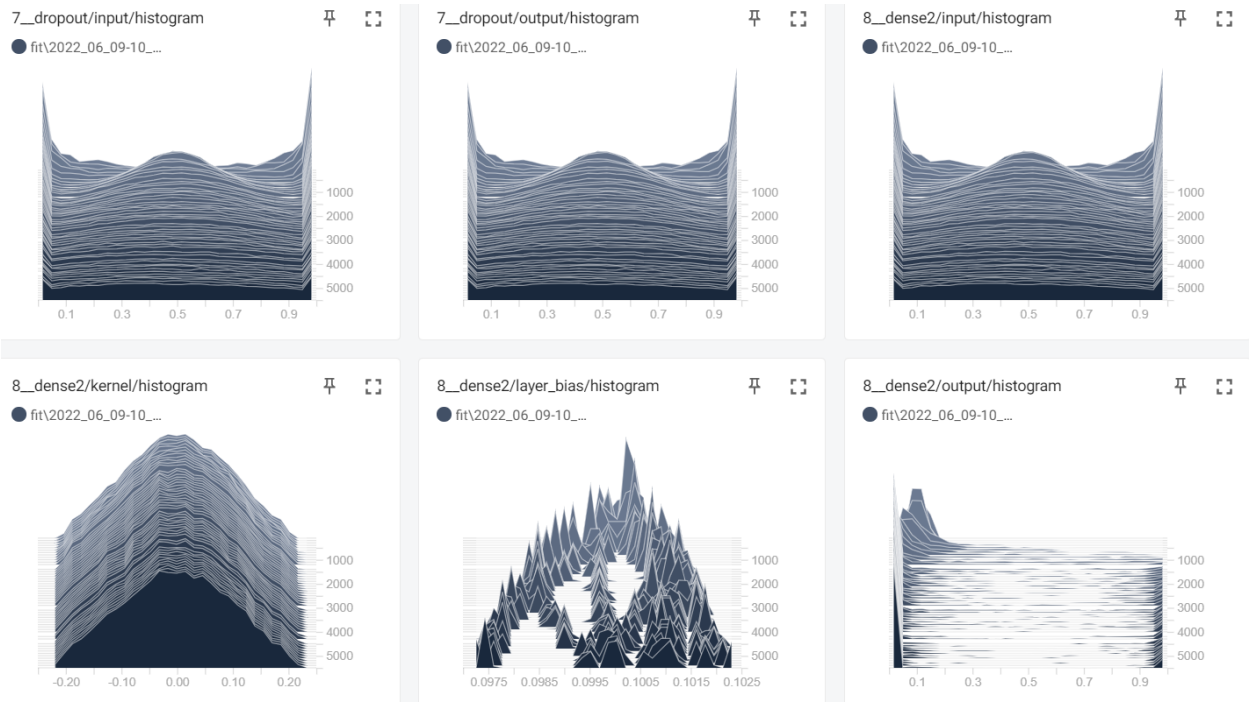


Figure 14: Layer Input, Output, Weight and Bias Histograms from Sigmoid Activation

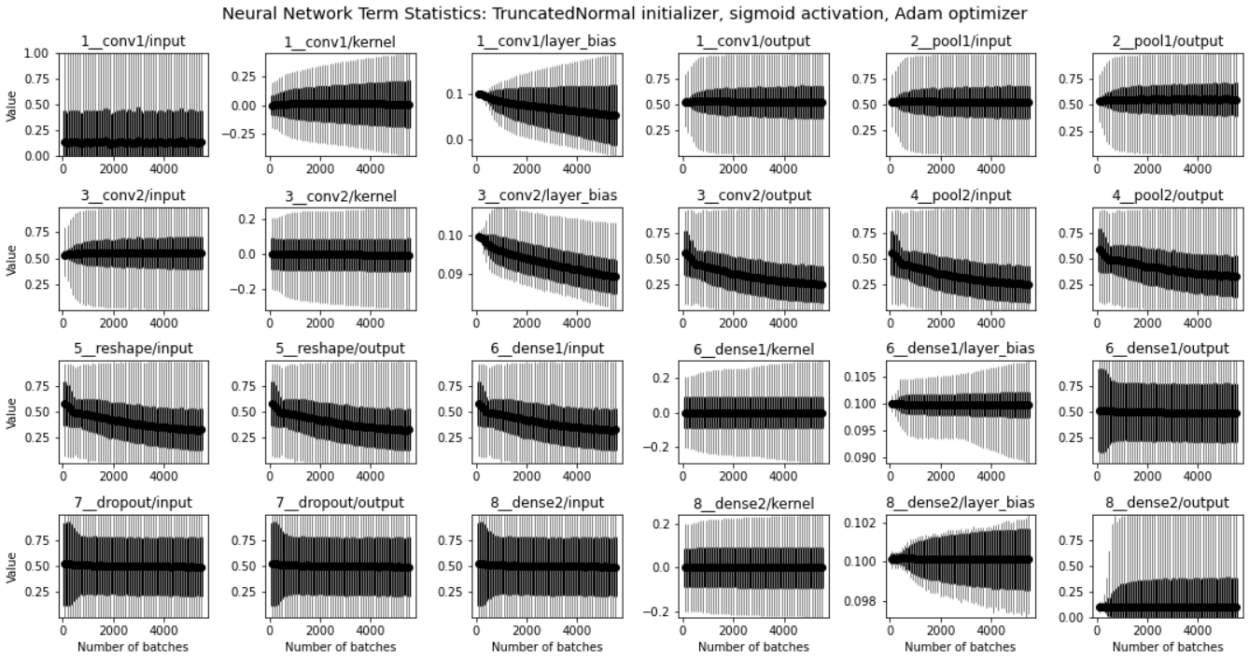
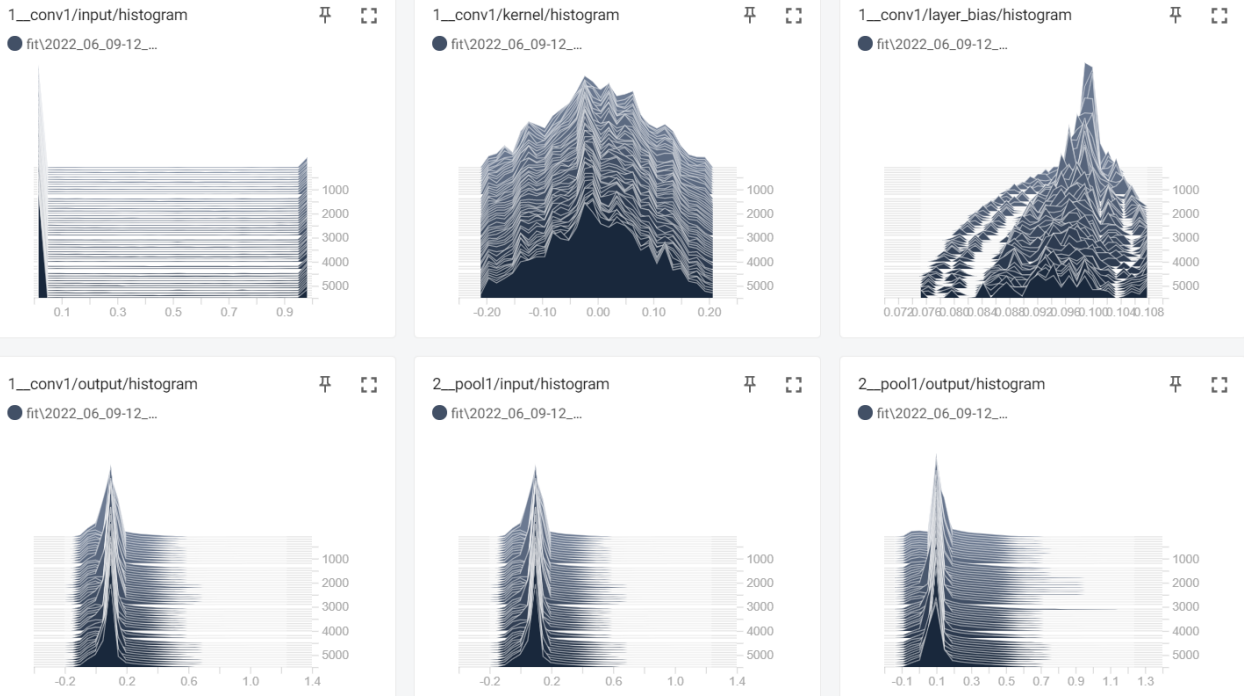


Figure 15: Layer Input, Output, Weight and Bias Statistics from Sigmoid Activation

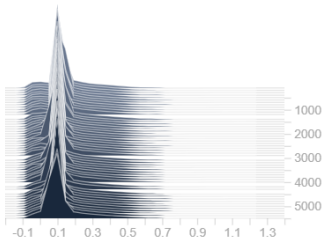
Leaky Relu (alpha = 0.2)

Figure 16: Accuracy and Loss from Leaky ReLU Activation



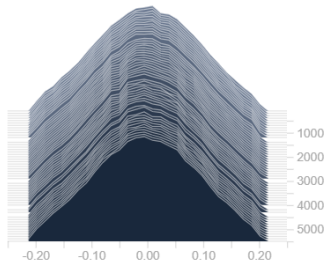
3__conv2/input/histogram

fit\2022_06_09-12_...



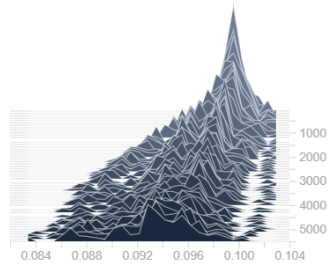
3__conv2/kernel/histogram

fit\2022_06_09-12_...



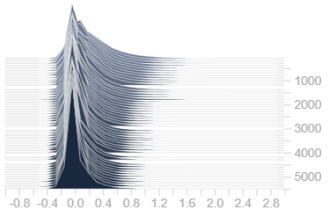
3__conv2/layer_bias/histogram

fit\2022_06_09-12_...



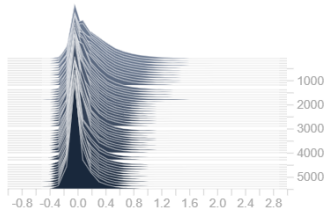
3__conv2/output/histogram

fit\2022_06_09-12_...



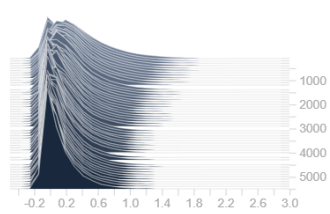
4__pool2/input/histogram

fit\2022_06_09-12_...



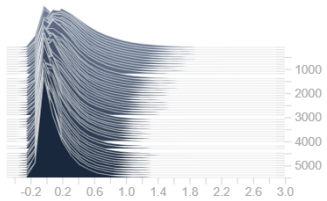
4__pool2/output/histogram

fit\2022_06_09-12_...



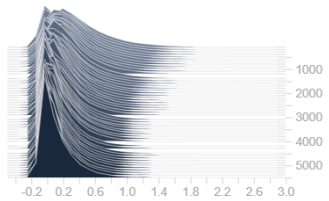
5__reshape/input/histogram

fit\2022_06_09-12_...



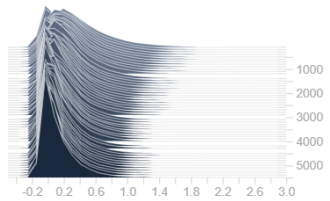
5__reshape/output/histogram

fit\2022_06_09-12_...



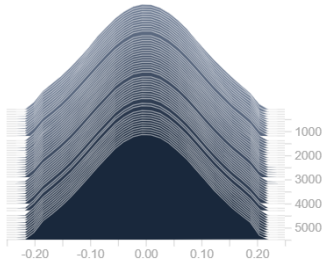
6__dense1/input/histogram

fit\2022_06_09-12_...



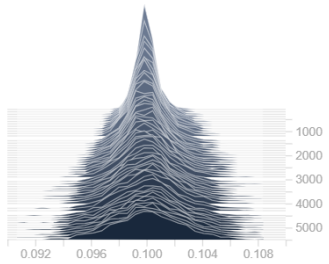
6__dense1/kernel/histogram

fit\2022_06_09-12_...



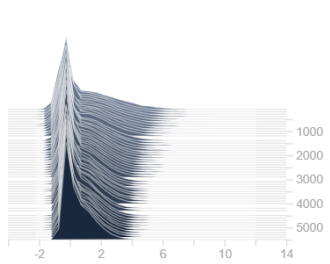
6__dense1/layer_bias/histogram

fit\2022_06_09-12_...



6__dense1/output/histogram

fit\2022_06_09-12_...



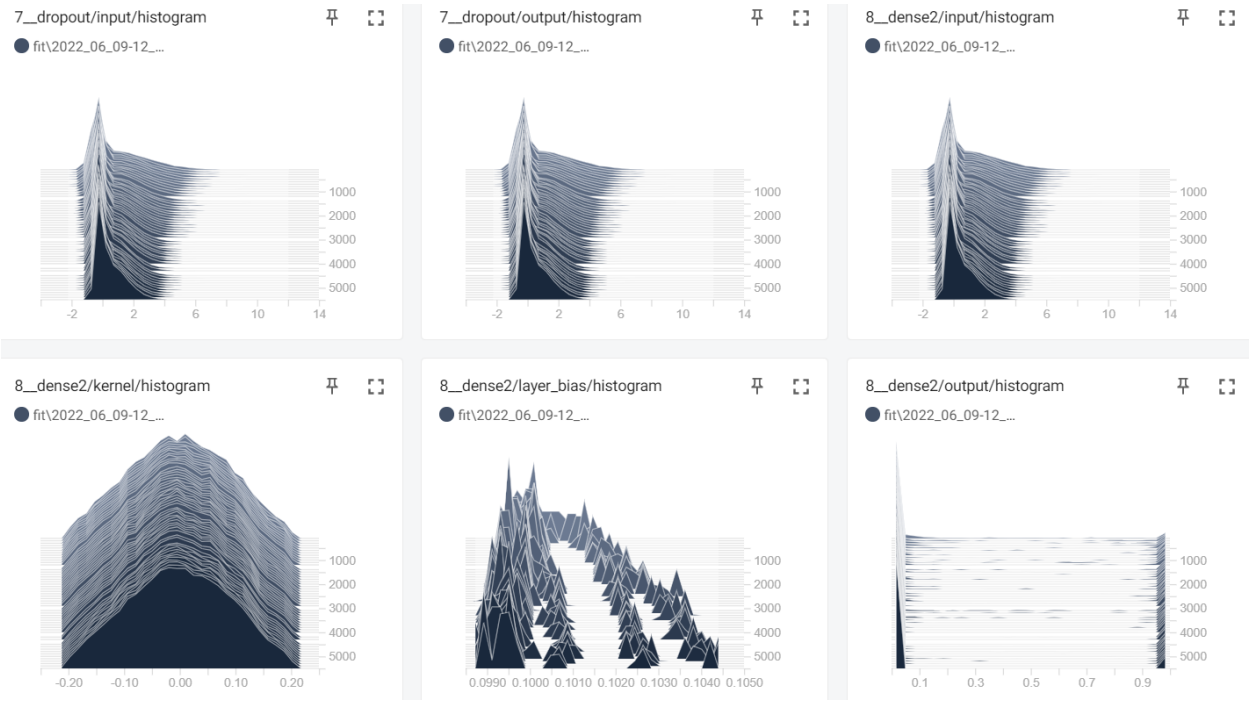


Figure 17: Layer Input, Output, Weight and Bias Histograms from Leaky ReLU Activation

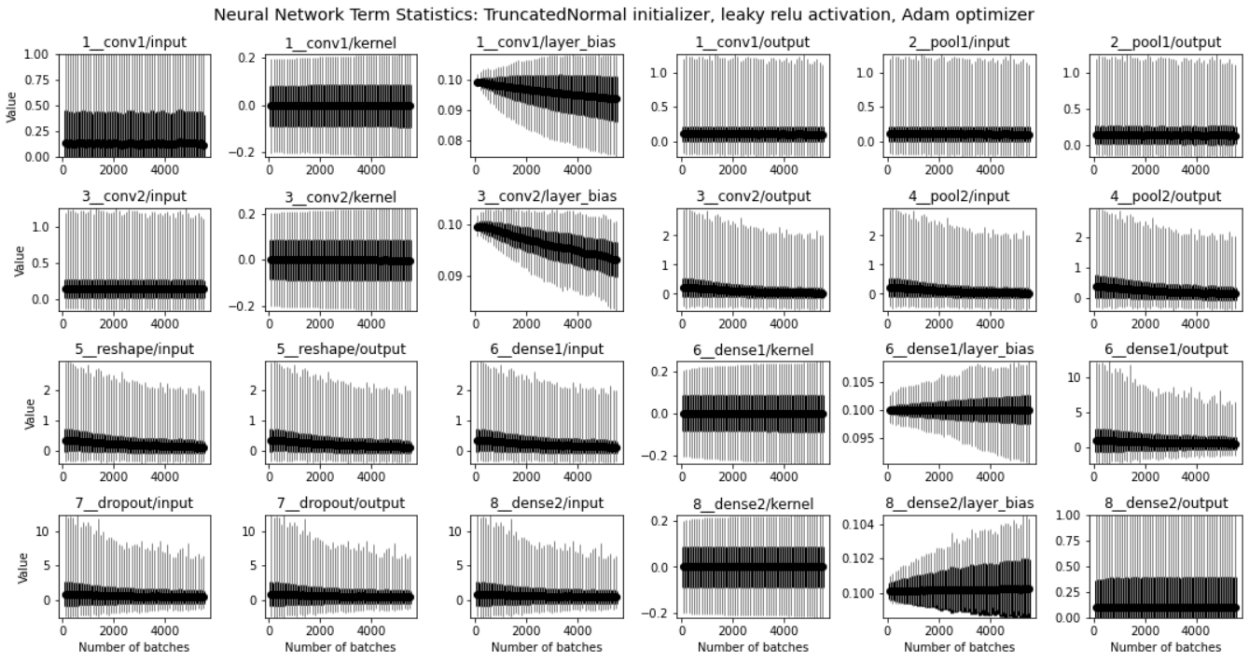
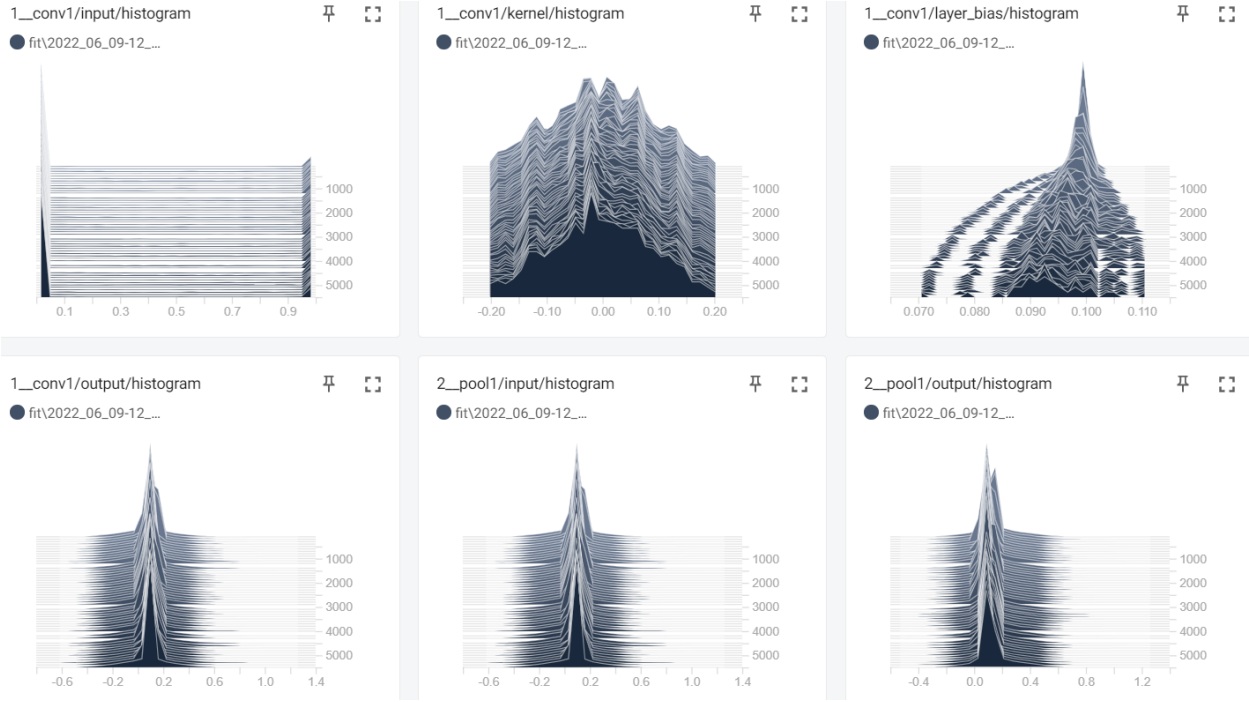


Figure 18: Layer Input, Output, Weight and Bias Statistics from Leaky ReLU Activation

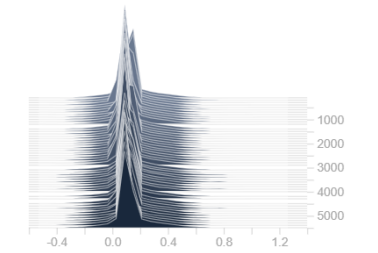
Exponential Linear Unit

Figure 19: Accuracy and Loss from ELU Activation



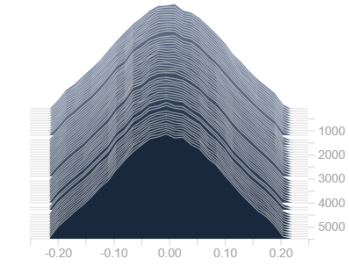
3__conv2/input/histogram

fit\2022_06_09-12_...



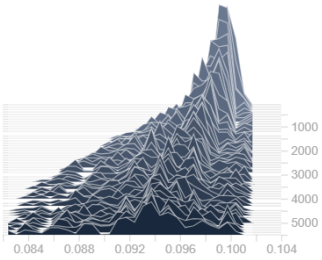
3__conv2/kernel/histogram

fit\2022_06_09-12_...



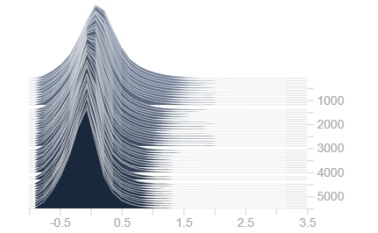
3__conv2/layer_bias/histogram

fit\2022_06_09-12_...



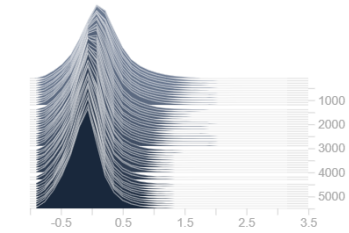
3__conv2/output/histogram

fit\2022_06_09-12_...



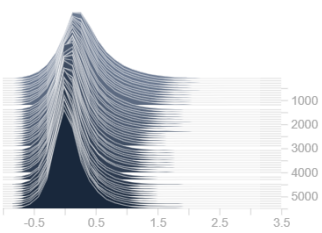
4__pool2/input/histogram

fit\2022_06_09-12_...



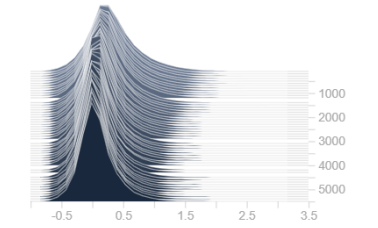
4__pool2/output/histogram

fit\2022_06_09-12_...



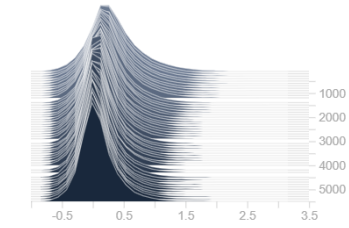
5__reshape/input/histogram

fit\2022_06_09-12_...



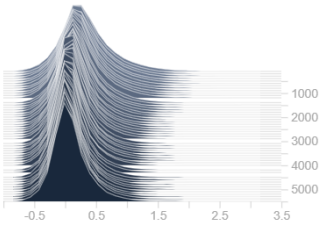
5__reshape/output/histogram

fit\2022_06_09-12_...



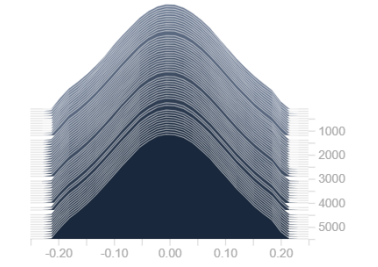
6__dense1/input/histogram

fit\2022_06_09-12_...



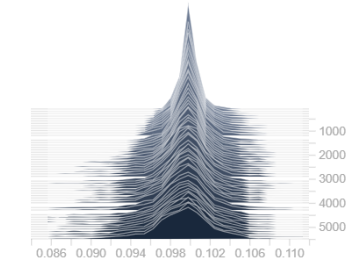
6__dense1/kernel/histogram

fit\2022_06_09-12_...



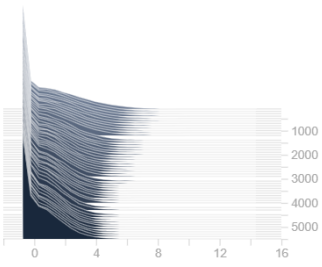
6__dense1/layer_bias/histogram

fit\2022_06_09-12_...



6__dense1/output/histogram

fit\2022_06_09-12_...



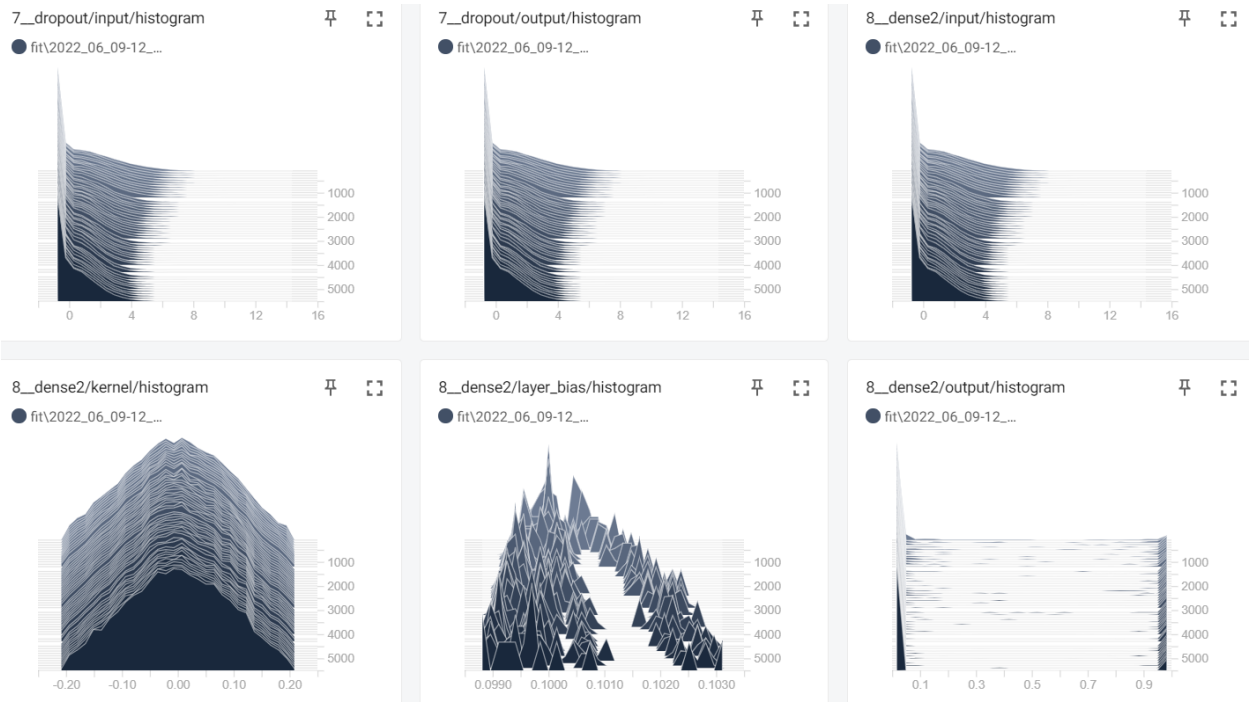


Figure 20: Layer Input, Output, Weight and Bias Histograms from ELU Activation

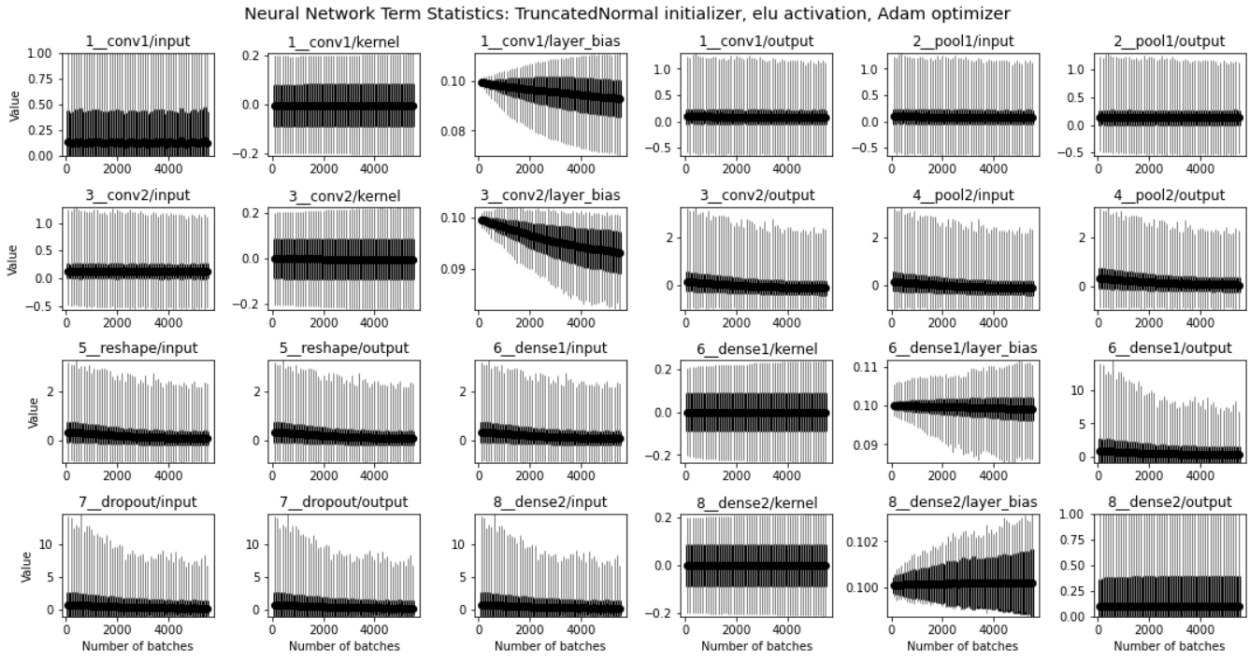
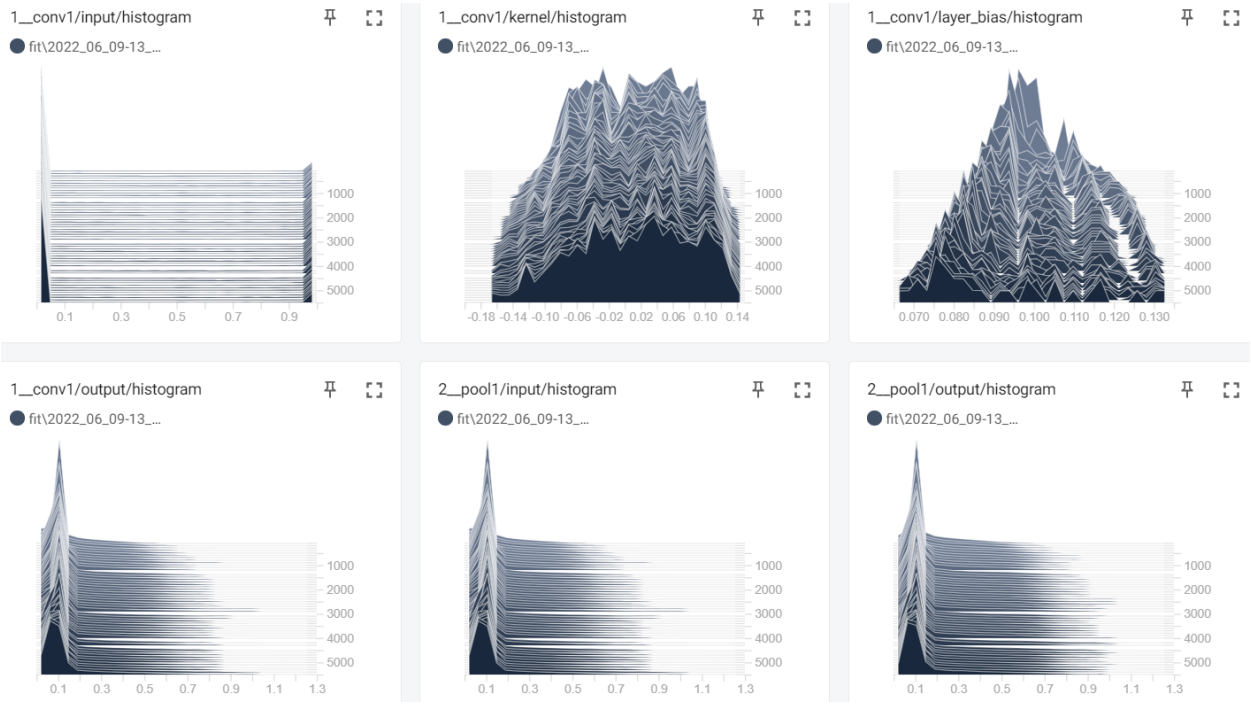


Figure 21: Layer Input, Output, Weight and Bias Statistics from ELU Activation

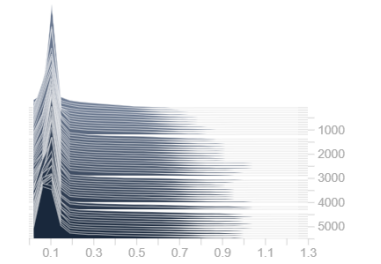
Xavier Uniform:

Figure 22: Accuracy and Loss from Xavier Uniform Initialization



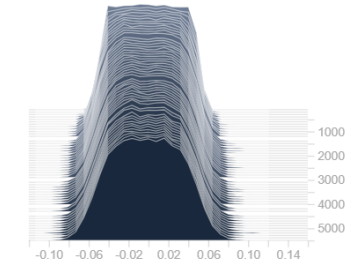
3__conv2/input/histogram

fit\2022_06_09-13_...



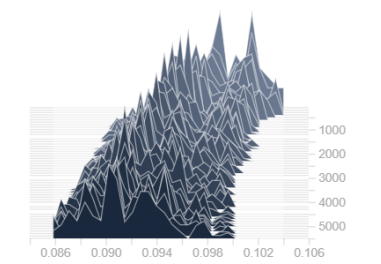
3__conv2/kernel/histogram

fit\2022_06_09-13_...



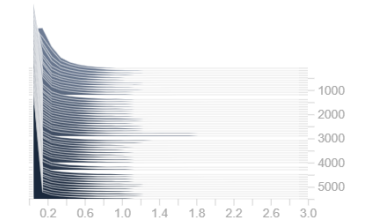
3__conv2/layer_bias/histogram

fit\2022_06_09-13_...



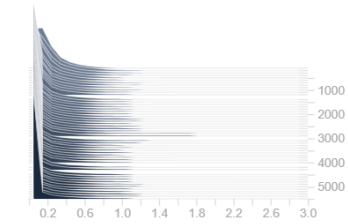
3__conv2/output/histogram

fit\2022_06_09-13_...



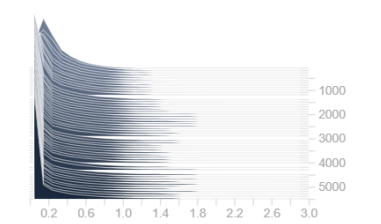
4__pool2/input/histogram

fit\2022_06_09-13_...



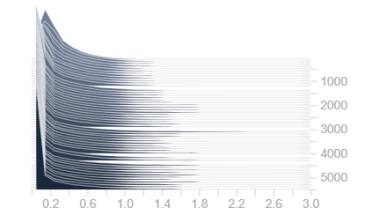
4__pool2/output/histogram

fit\2022_06_09-13_...



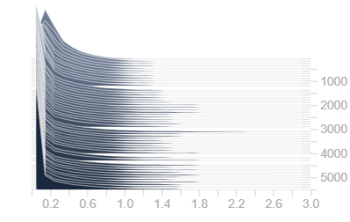
5__reshape/input/histogram

fit\2022_06_09-13_...



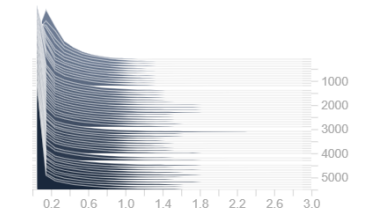
5__reshape/output/histogram

fit\2022_06_09-13_...



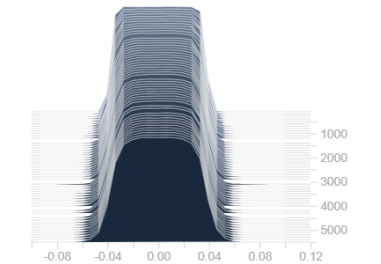
6__dense1/input/histogram

fit\2022_06_09-13_...



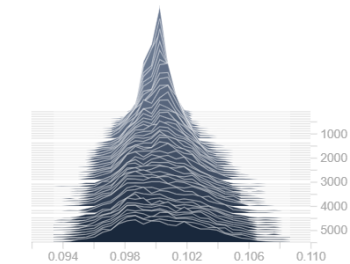
6__dense1/kernel/histogram

fit\2022_06_09-13_...



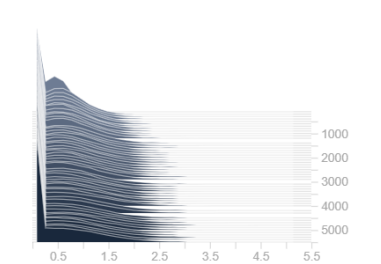
6__dense1/layer_bias/histogram

fit\2022_06_09-13_...



6__dense1/output/histogram

fit\2022_06_09-13_...



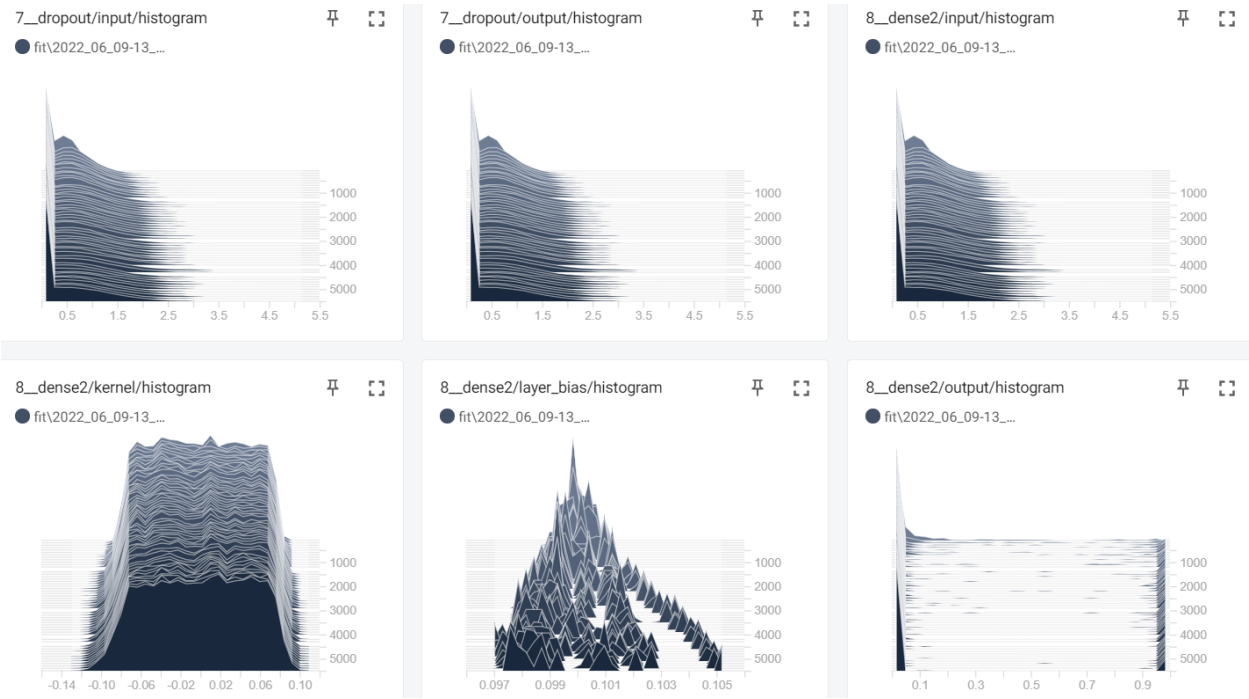


Figure 23: Layer Input, Output, Weight and Bias Histograms from Xavier Uniform Initialization

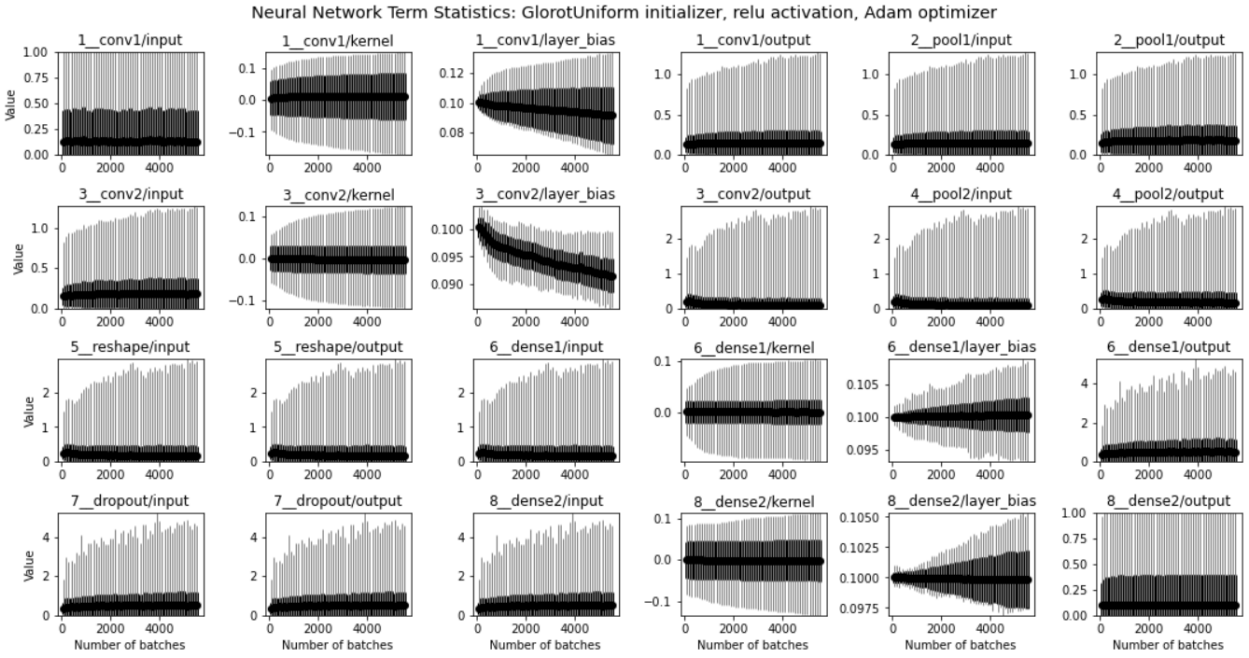
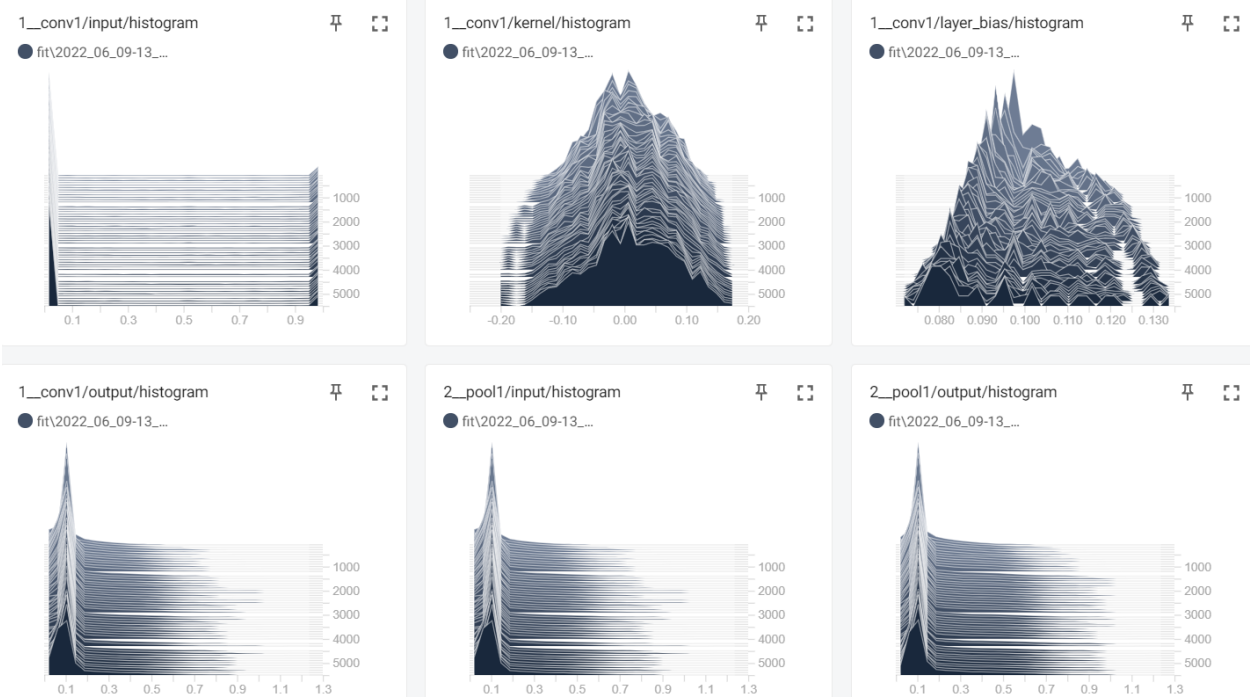


Figure 24: Layer Input, Output, Weight and Bias Statistics from Xavier Uniform Initialization

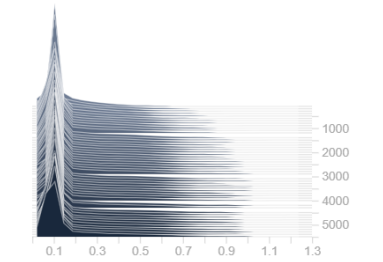
Xavier Normal:

Figure 25: Accuracy and Loss from Xavier Normal Initialization



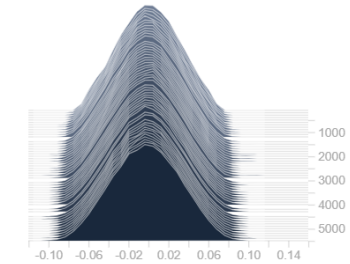
3__conv2/input/histogram

fit\2022_06_09-13_...



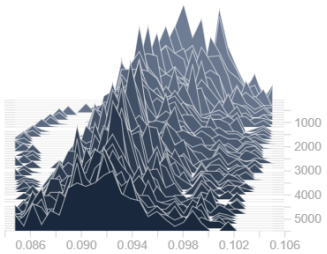
3__conv2/kernel/histogram

fit\2022_06_09-13_...



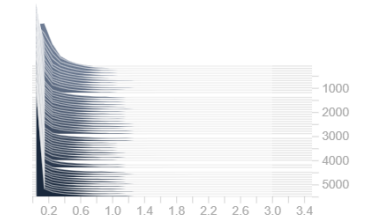
3__conv2/layer_bias/histogram

fit\2022_06_09-13_...



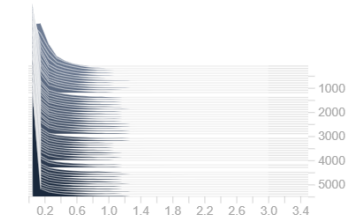
3__conv2/output/histogram

fit\2022_06_09-13_...



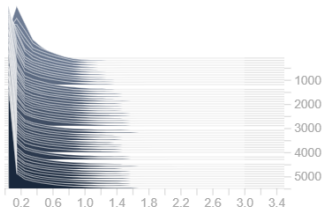
4__pool2/input/histogram

fit\2022_06_09-13_...



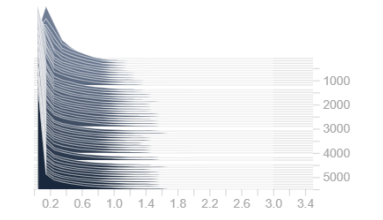
4__pool2/output/histogram

fit\2022_06_09-13_...



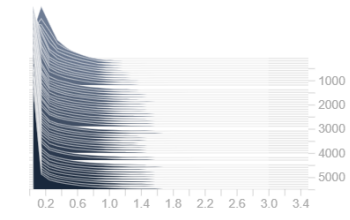
5__reshape/input/histogram

fit\2022_06_09-13_...



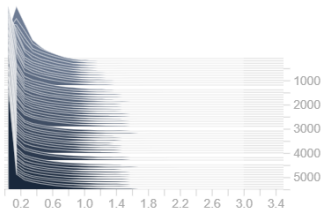
5__reshape/output/histogram

fit\2022_06_09-13_...



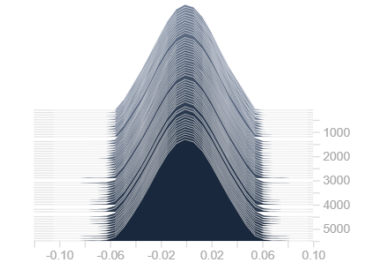
6__dense1/input/histogram

fit\2022_06_09-13_...



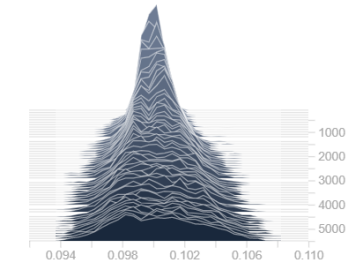
6__dense1/kernel/histogram

fit\2022_06_09-13_...



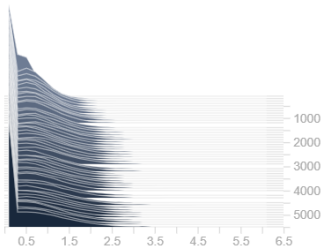
6__dense1/layer_bias/histogram

fit\2022_06_09-13_...



6__dense1/output/histogram

fit\2022_06_09-13_...



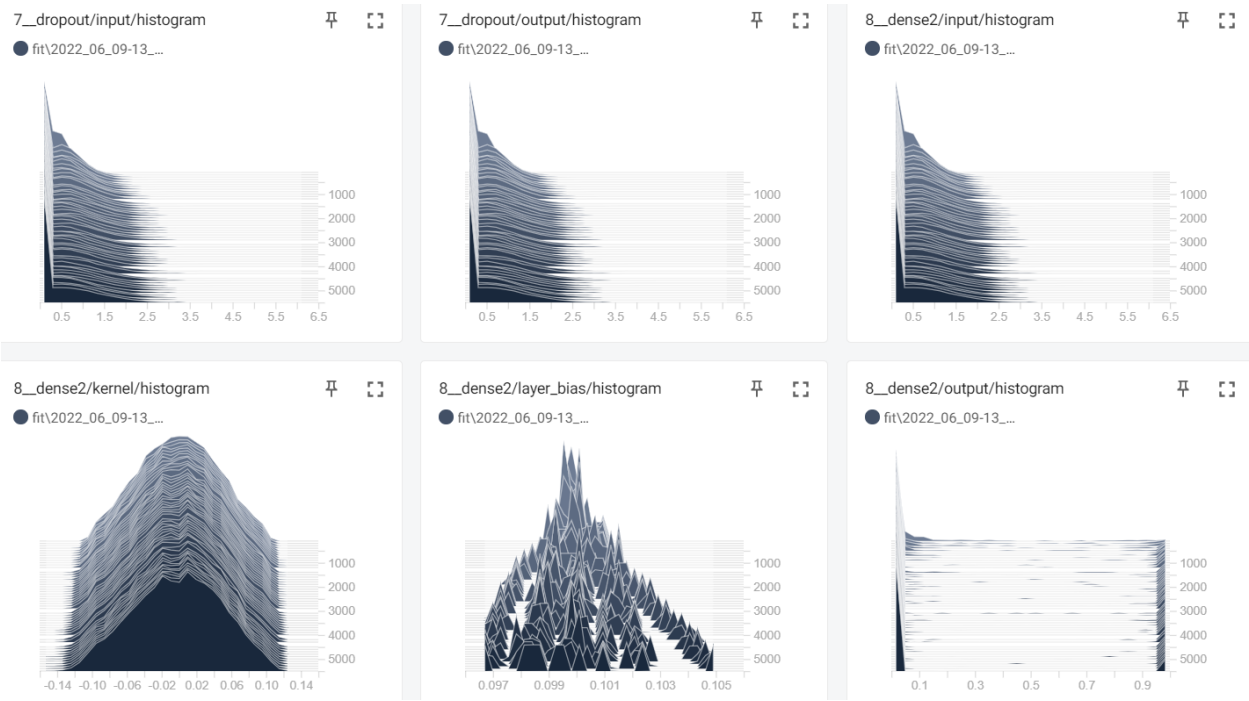


Figure 26: Layer Input, Output, Weight and Bias Histograms from Xavier Normal Initialization

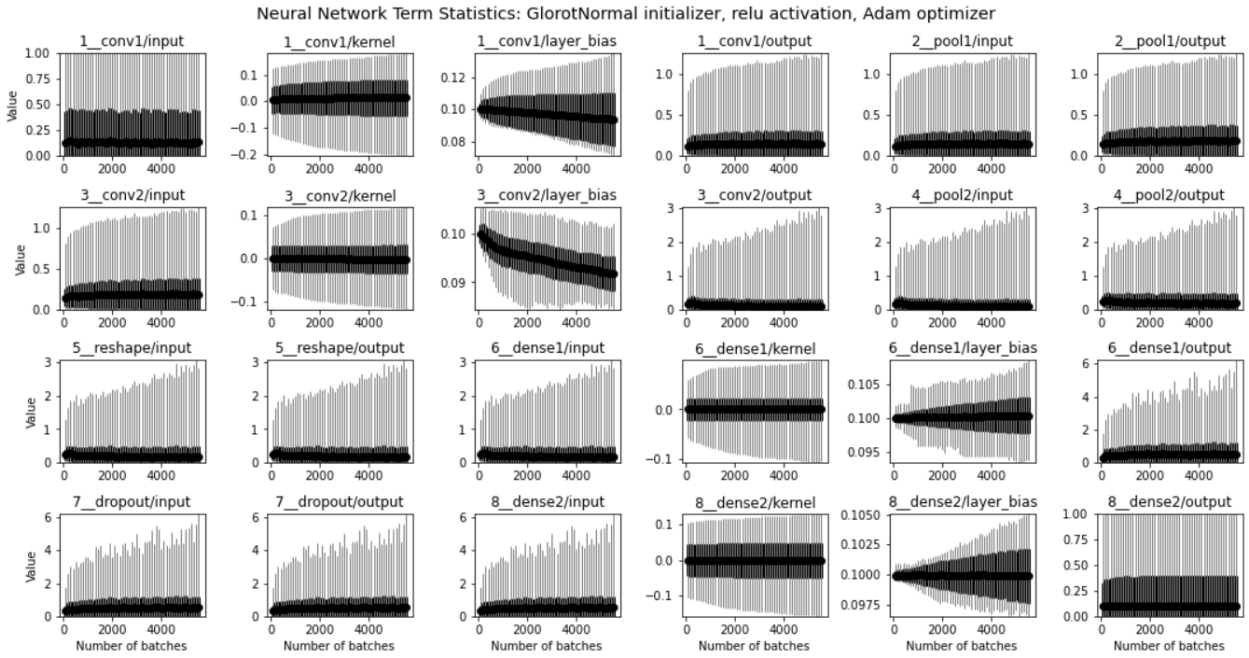
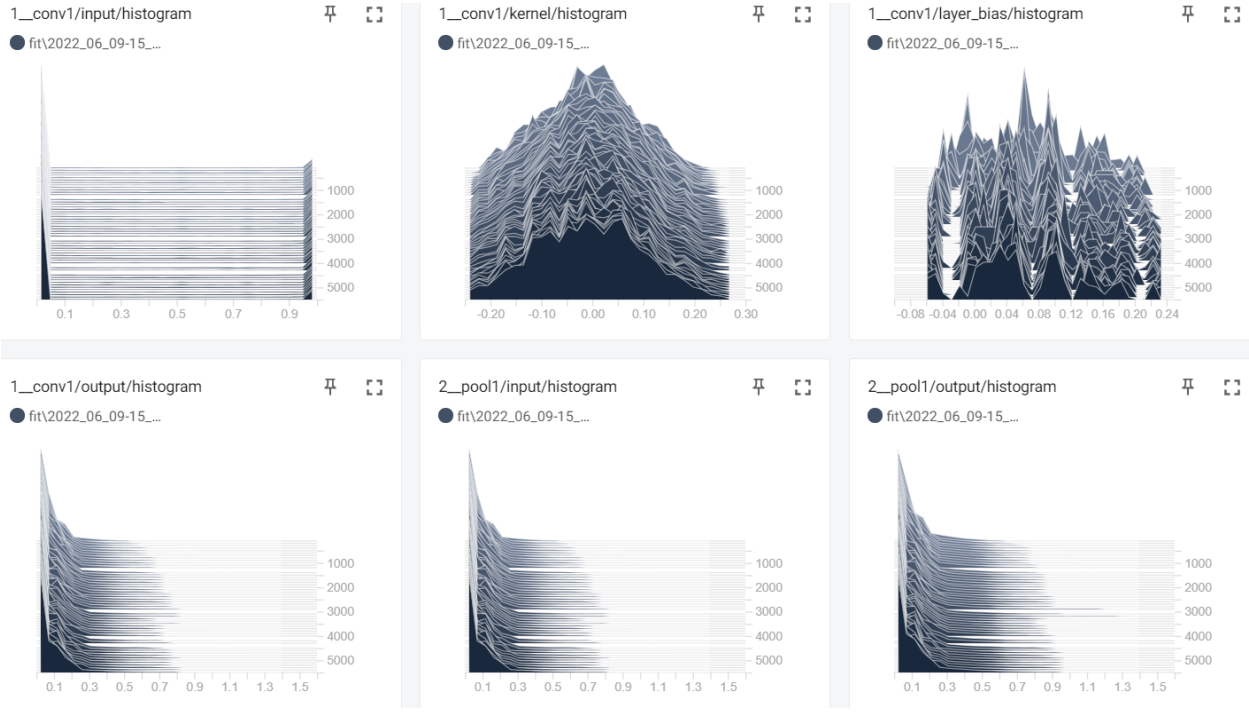


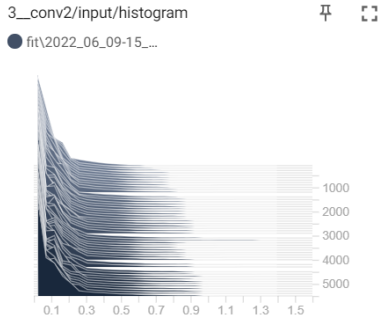
FIgure 27: Layer Input, Output, Weight and Bias Statistics from Xavier Normal Initialization

Stochastic Gradient Descent (SGD):

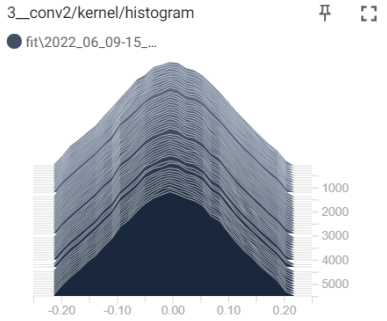
Figure 28: Accuracy and Loss from SGD Optimization



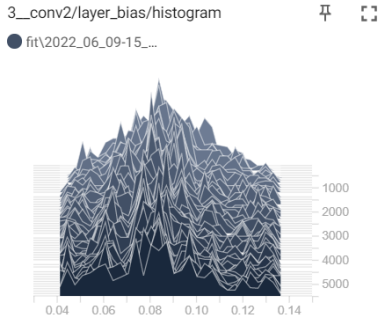
3__conv2/input/histogram



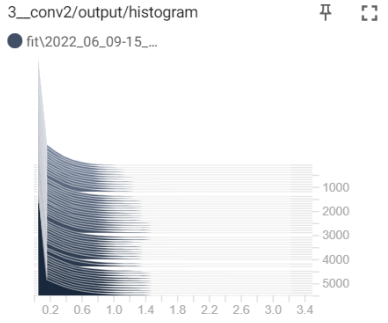
3__conv2/kernel/histogram



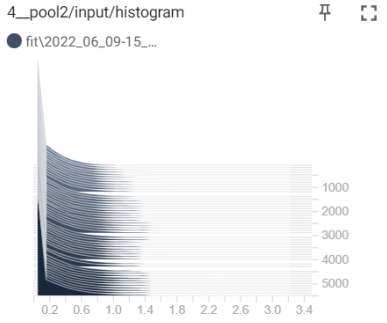
3__conv2/layer_bias/histogram



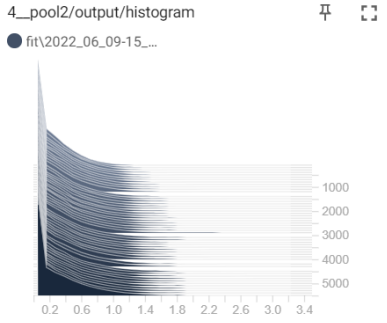
3__conv2/output/histogram



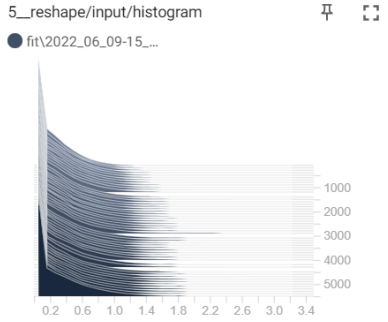
4__pool2/input/histogram



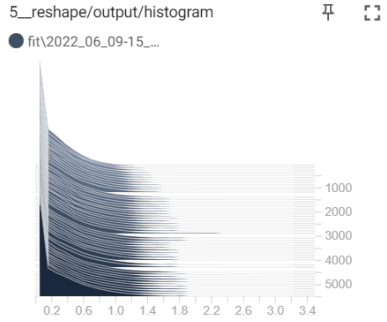
4__pool2/output/histogram



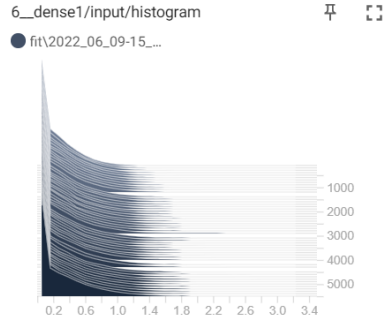
5__reshape/input/histogram



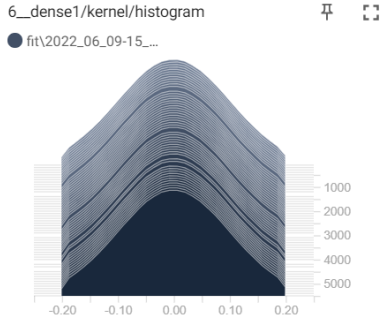
5__reshape/output/histogram



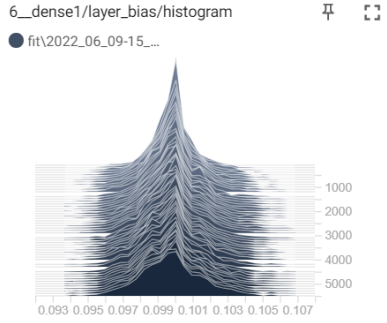
6__dense1/input/histogram



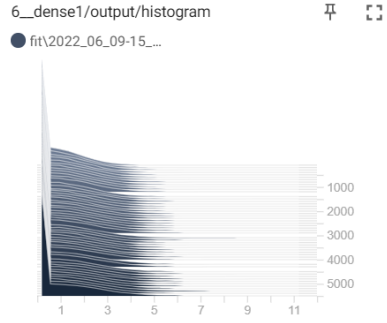
6__dense1/kernel/histogram



6__dense1/layer_bias/histogram



6__dense1/output/histogram



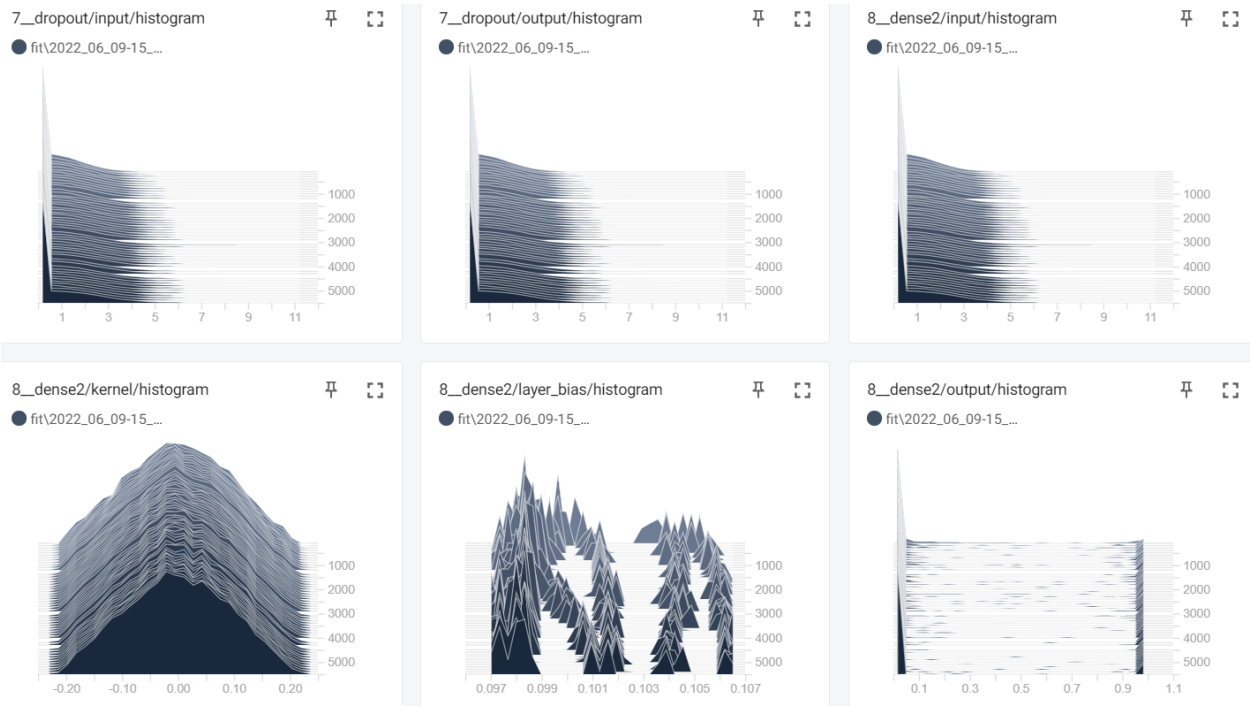


Figure 29: Layer Input, Output, Weight and Bias Histograms from SGD Optimization

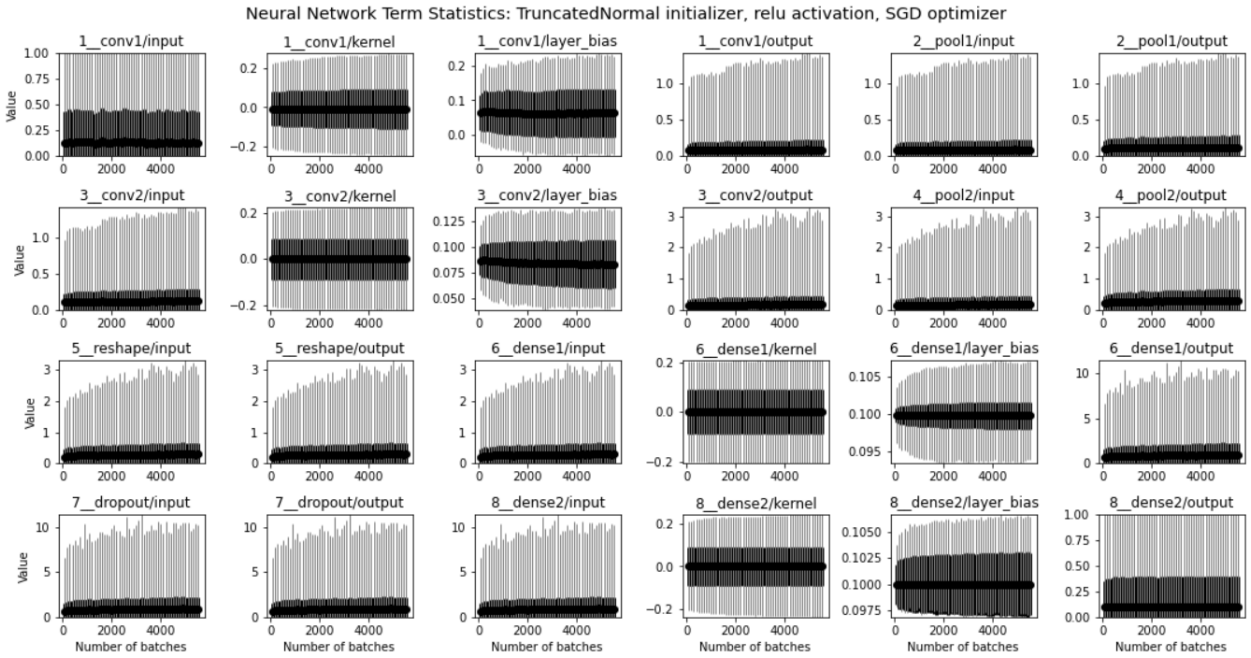
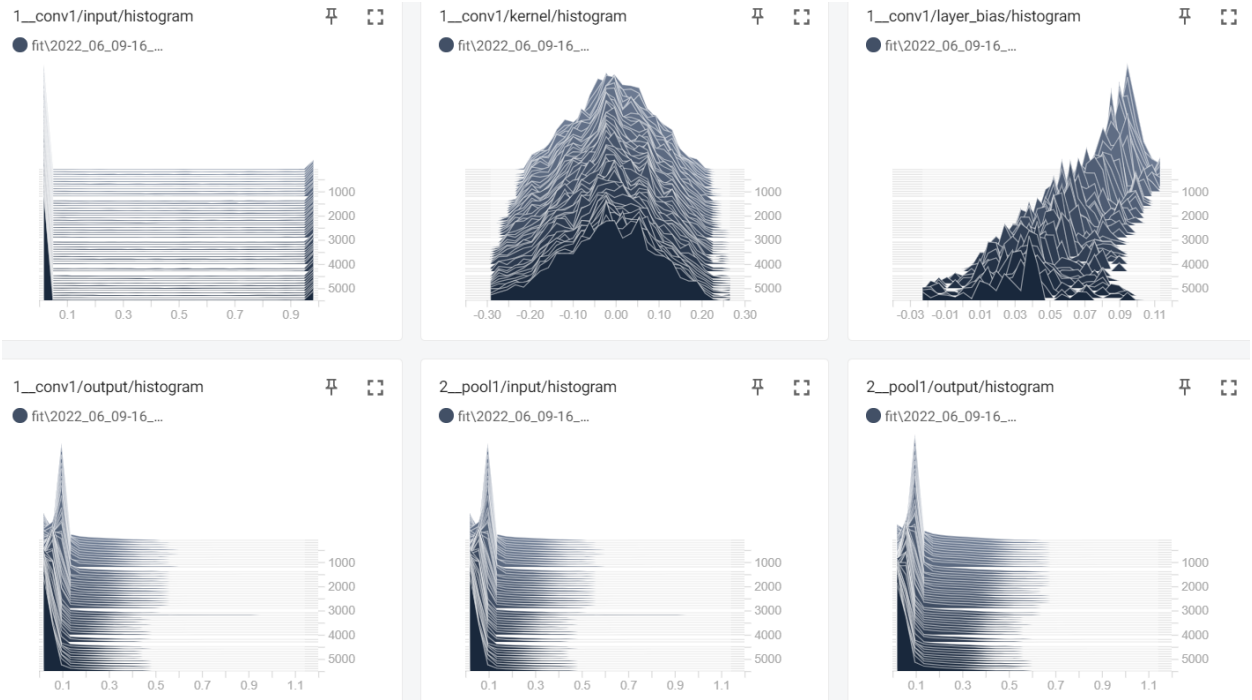


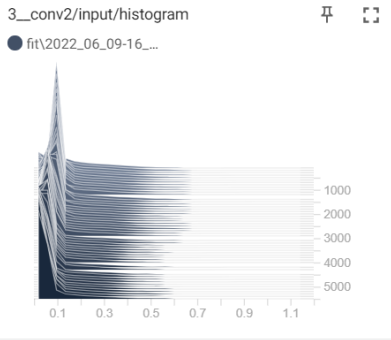
Figure 30: Layer Input, Output, Weight and Bias Statistics from SGD Optimization

Nadam:

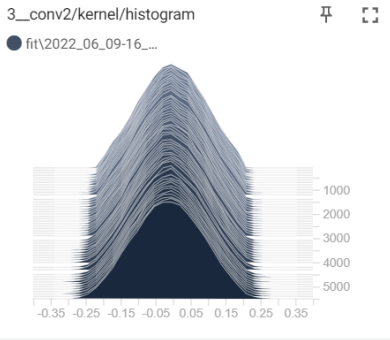
Figure 31: Accuracy and Loss from Nadam Optimization



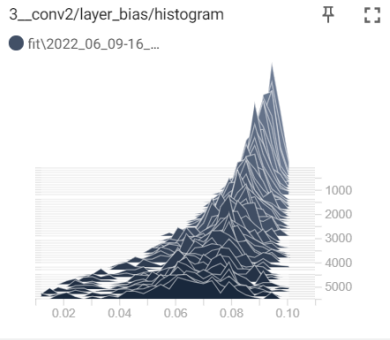
3__conv2/input/histogram



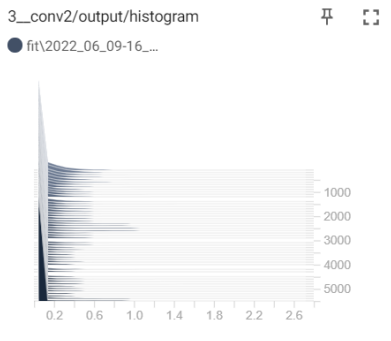
3__conv2/kernel/histogram



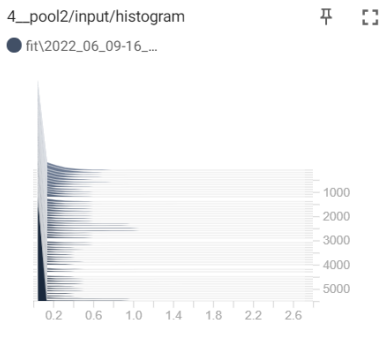
3__conv2/layer_bias/histogram



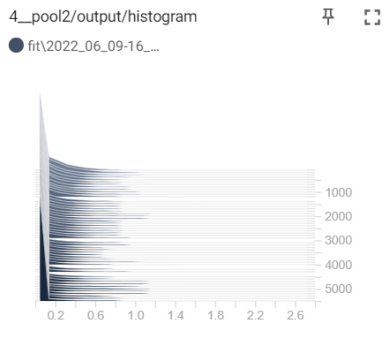
3__conv2/output/histogram



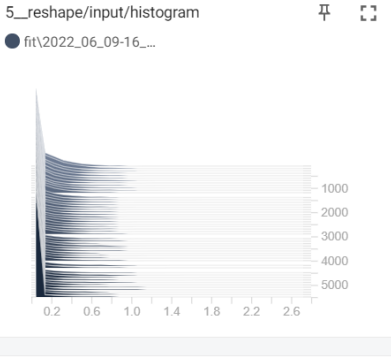
4__pool2/input/histogram



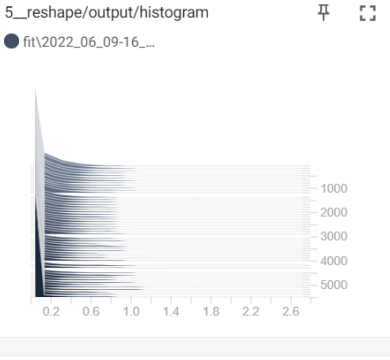
4__pool2/output/histogram



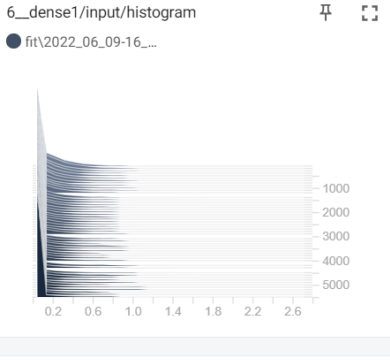
5__reshape/input/histogram



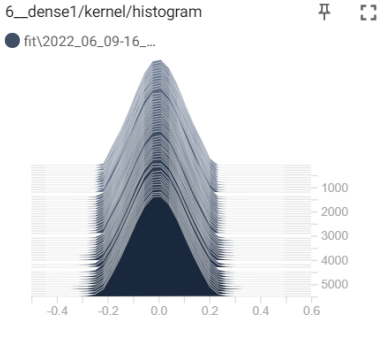
5__reshape/output/histogram



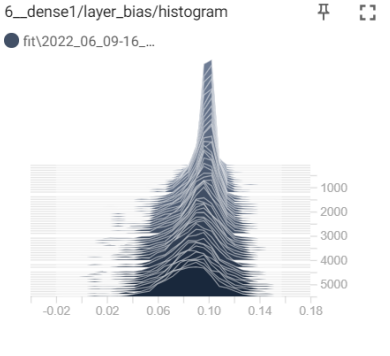
6__dense1/input/histogram



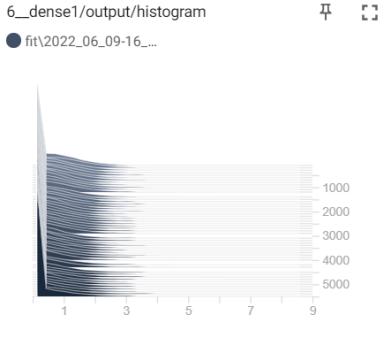
6__dense1/kernel/histogram



6__dense1/layer_bias/histogram



6__dense1/output/histogram



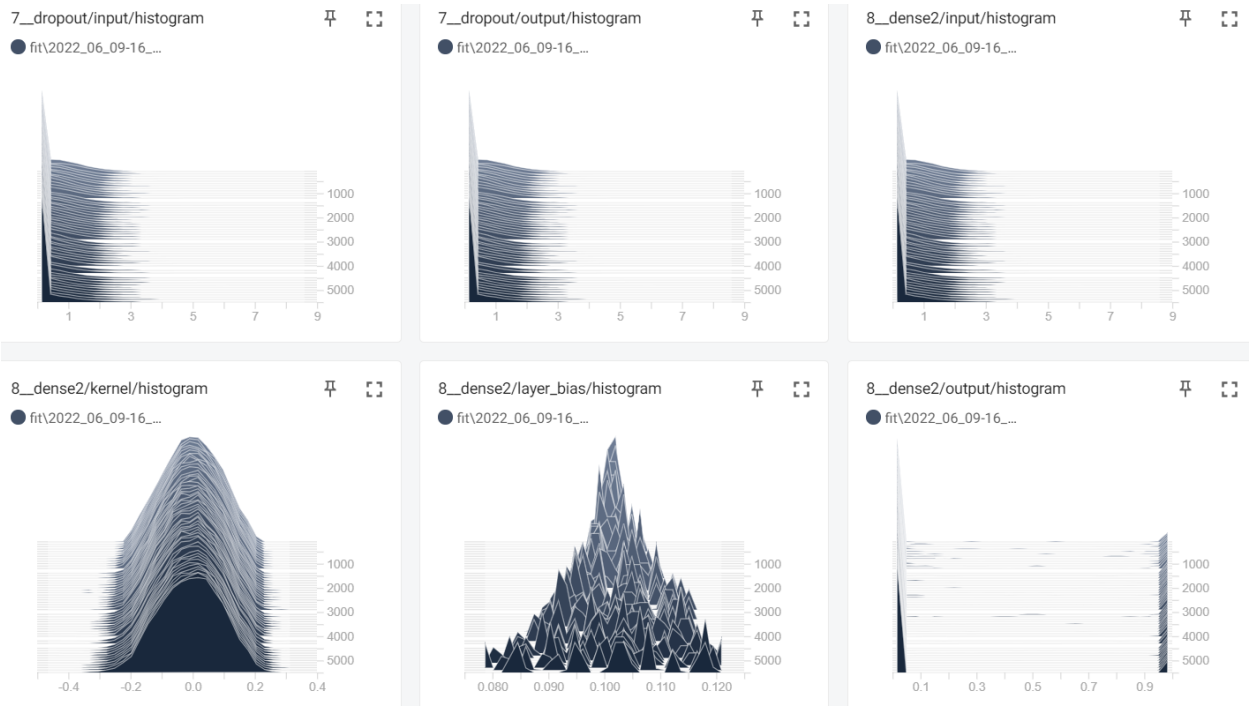


Figure 32: Layer Input, Output, Weight and Bias Histograms from Nadam Optimization

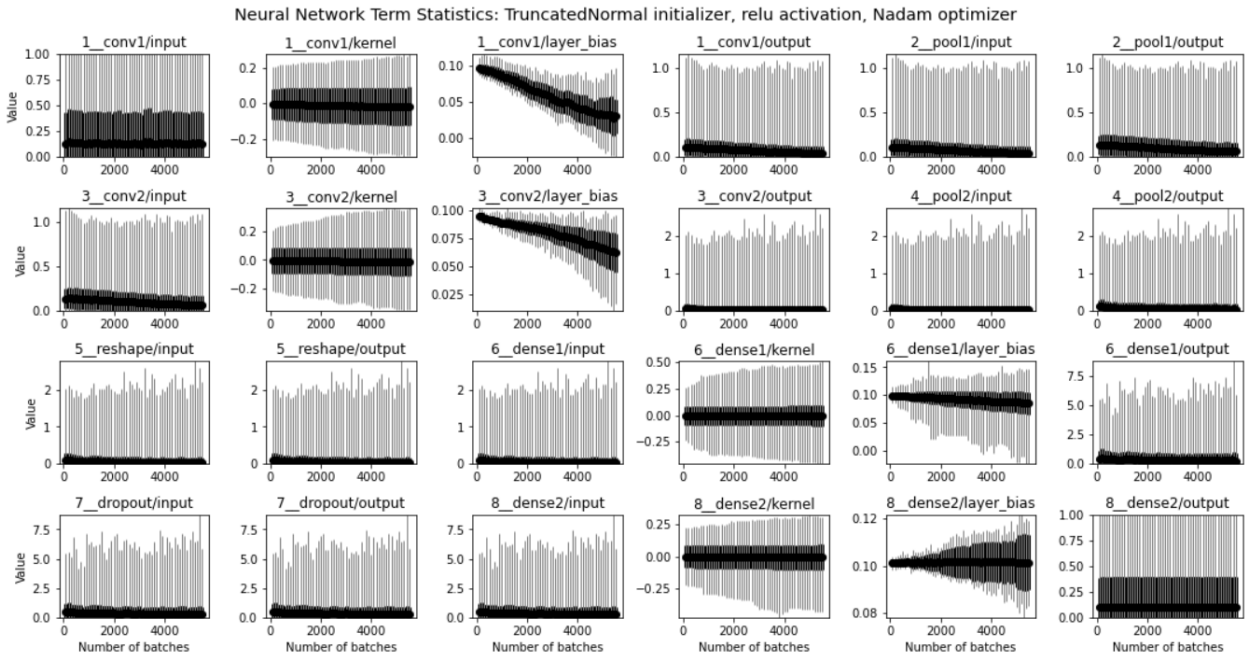


Figure 33: Layer Input, Output, Weight and Bias Statistics from Nadam Optimization

Adagrad:

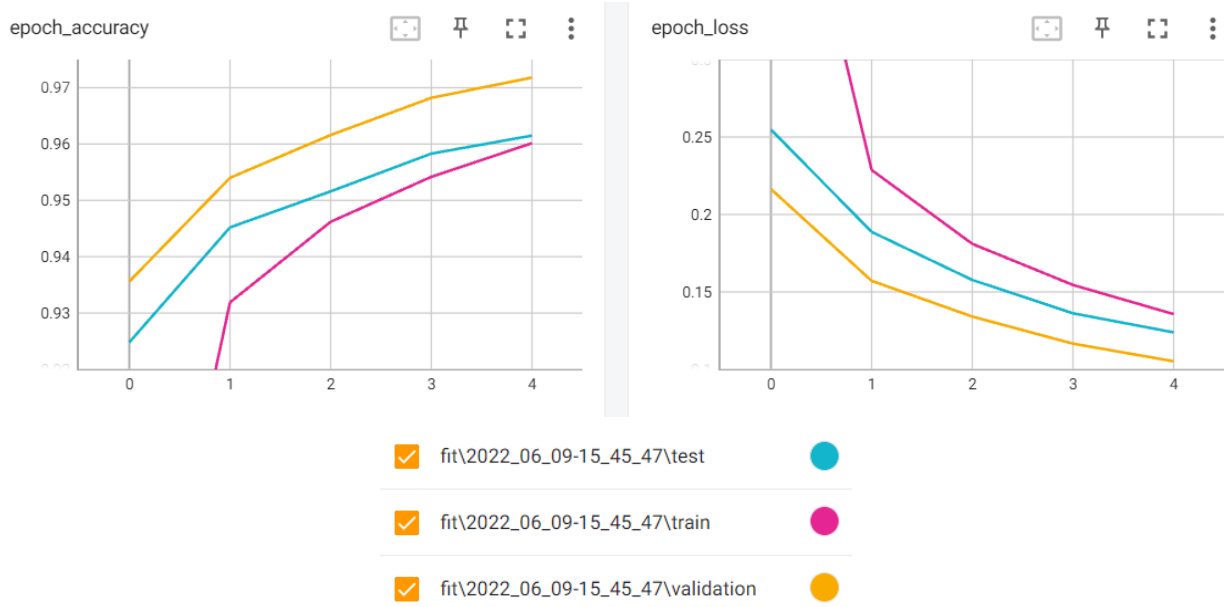
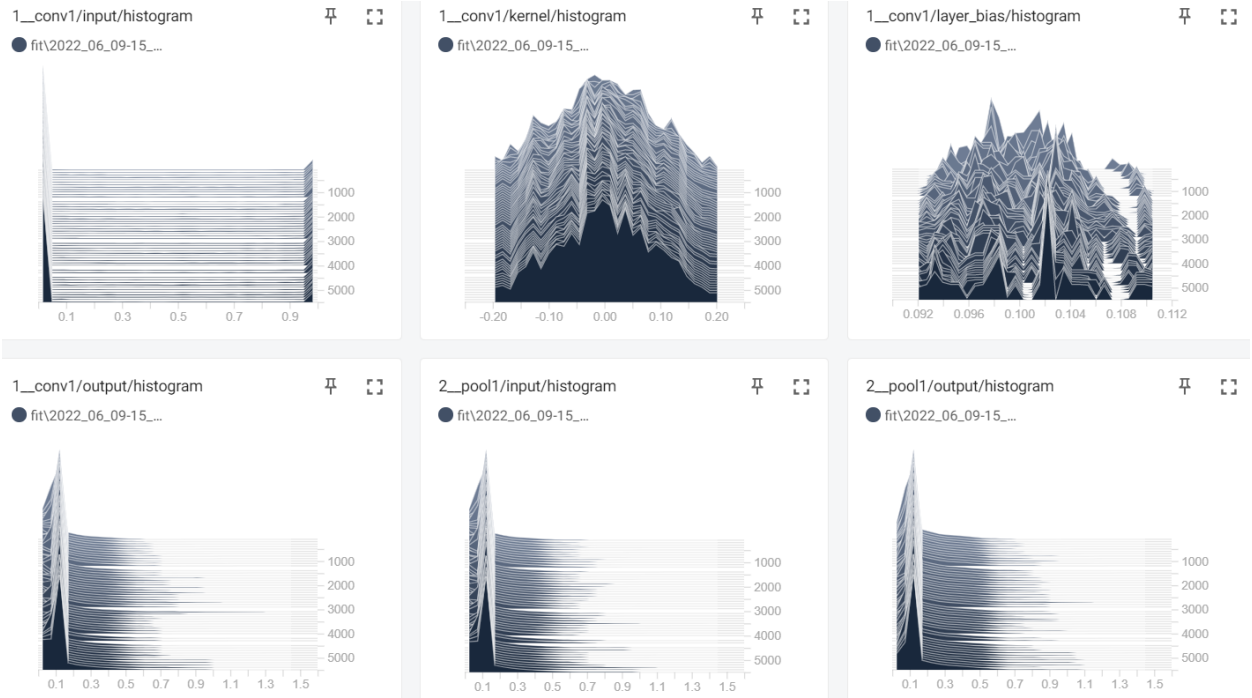
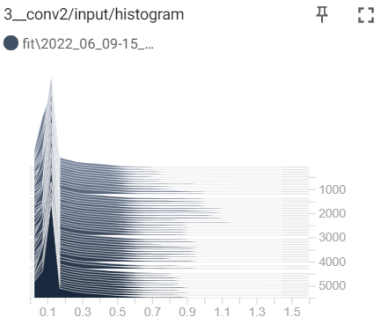


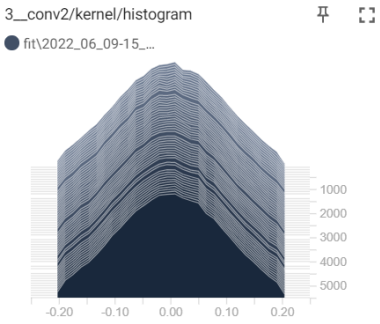
Figure 34: Accuracy and Loss from Adagrad Optimization



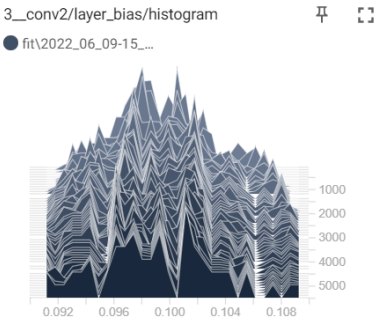
3__conv2/input/histogram



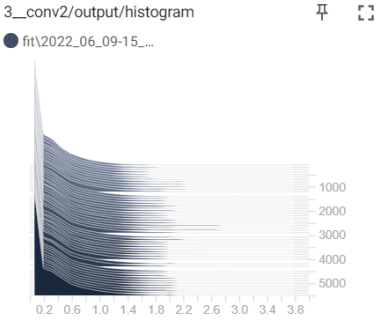
3__conv2/kernel/histogram



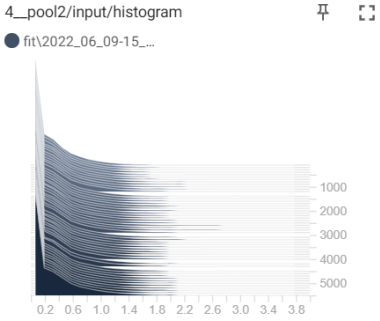
3__conv2/layer_bias/histogram



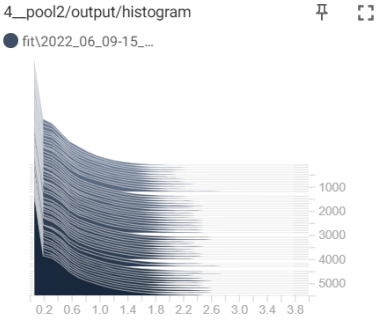
3__conv2/output/histogram



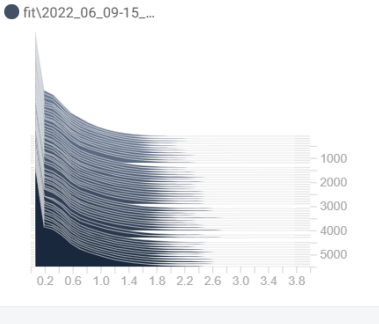
4__pool2/input/histogram



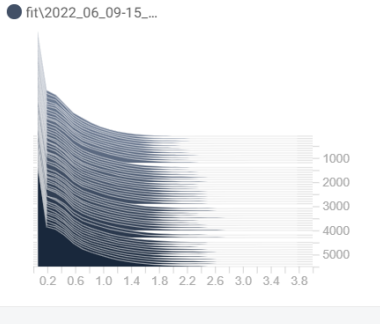
4__pool2/output/histogram



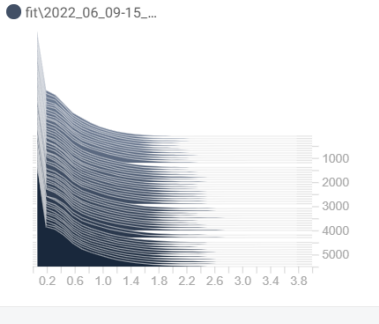
5__reshape/input/histogram



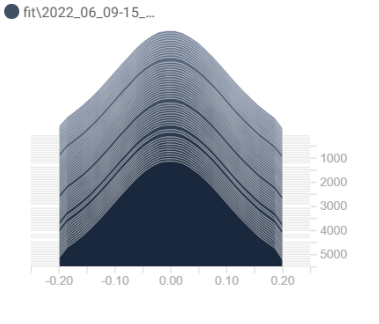
5__reshape/output/histogram



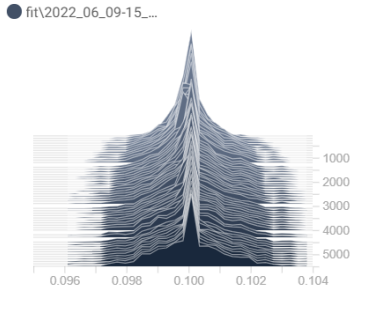
6__dense1/input/histogram



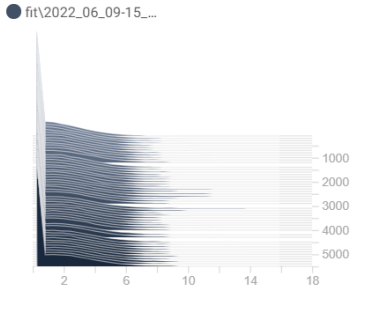
6__dense1/kernel/histogram



6__dense1/layer_bias/histogram



6__dense1/output/histogram



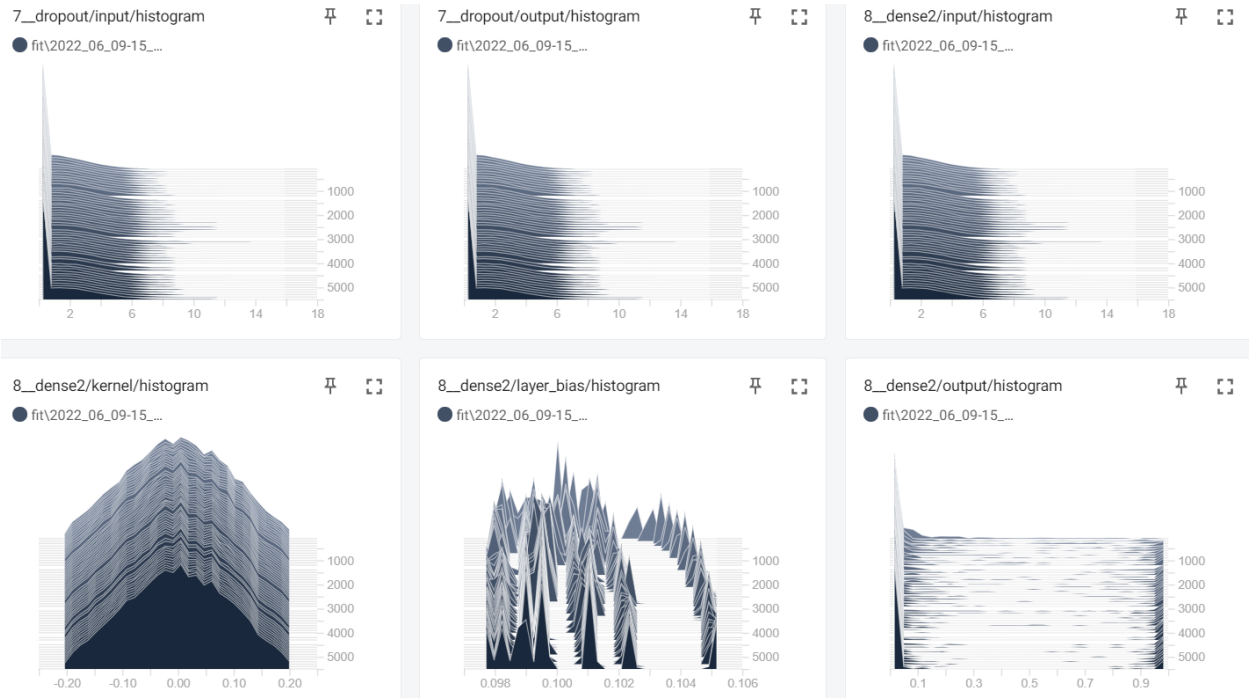


Figure 35: Layer Input, Output, Weight and Bias Histograms from Adagrad Optimization

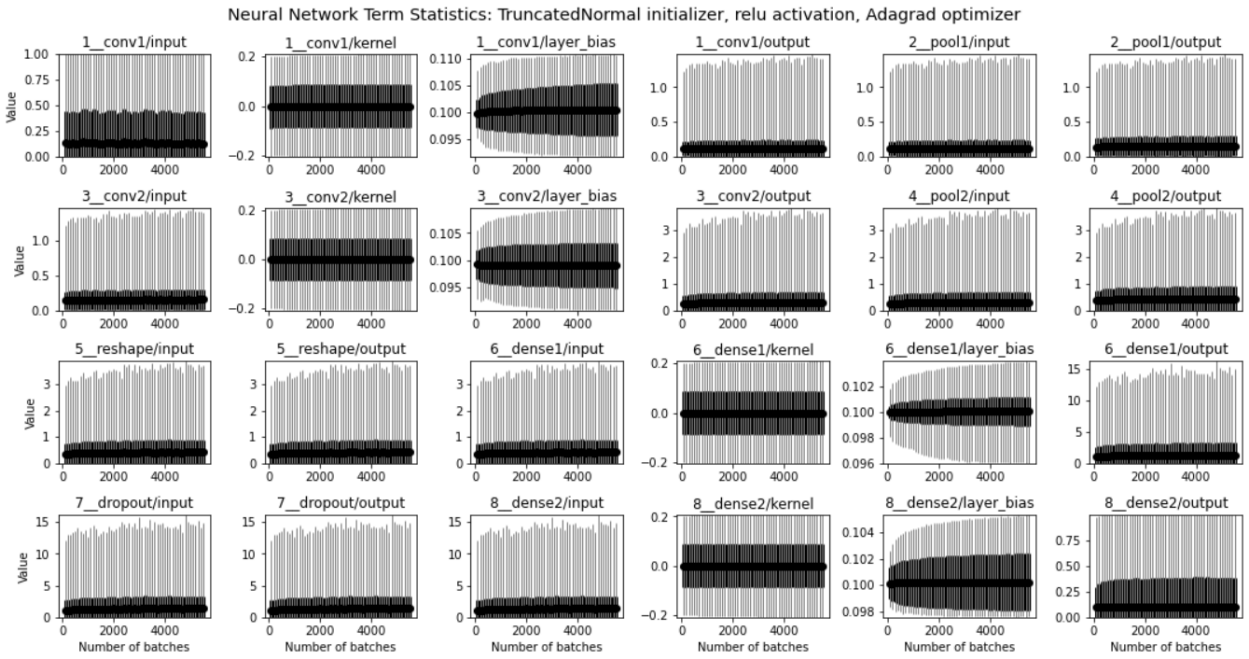


Figure 36: Layer Input, Output, Weight and Bias Statistics from Adagrad Optimization

Tanh: $f(u) = e^{u} - e^{-u} / e^{u} + e^{-u}$

$f'(u) = (e^{u}+e^{-u}) \cdot (e^{u}+e^{-u}) - (e^{u}-e^{-u})(e^{u}-e^{-u}) / (e^{u}+e^{-u})^2$

$f'(u) = (e^{u}+e^{-u})^2 - (e^{u}-e^{-u})^2 / (e^{u}+e^{-u})^2$

$f'(u) = 1 - (e^{u}-e^{-u})^2 / (e^{u}+e^{-u})^2$

$f'(u) = 1 - f(u)^2$

$$\boxed{\tanh'(z) = 1 - \tanh(z)^2}$$

Sigmoid: $f(z) = 1 / 1+e^{-z} = (1+e^{-z})^{-1}$

$f'(z) = -(1+e^{-z})^{-2} \cdot (-e^{-z})$

$f'(z) = e^{-z} / (1+e^{-z})^2$

$$\boxed{f'(z) = e^{-z} / (1+e^{-z})^2}$$

ReLU: $f(z) = \begin{cases} x, & x \geq 0 \\ 0, & x < 0 \end{cases}$ or $\begin{cases} x, & x > 0 \\ 0, & x \leq 0 \end{cases}$

$$\boxed{f'(z) = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases} \text{ or } \begin{cases} 1, & x > 0 \\ 0, & x \leq 0 \end{cases}}$$

S76 HW 1 pg. 2

d) 1. Loss = Cross-Entropy.

$L(Y, \hat{Y}) = -\frac{1}{N} \sum_{n \in N} \sum_{i \in C} Y_{n,i} \log(\hat{Y}_{n,i})$.

Let's first look at 1 training example ($N=1$):

- $L(y, \hat{y}) = -\sum_{i \in C} y_i \log(\hat{y}_i)$ $\qquad z_{2,i} = z_2[i]$
- $\hat{y} = \text{softmax}(z_2) \rightarrow \hat{y}_i = \text{softmax}(z_{2,i})$
- Since y is one-hot encoded it only has one non-zero element, which equals 1. Let's say that y belongs to class k.
  i.e., $y_i = \begin{cases} 1, & i=k \\ 0, & i \neq k. \end{cases}$

Therefore, $L(y, \hat{y}) = -\sum_{\substack{i \in C, \\ i \neq k}} 0 \log(\hat{y}_i) - 1 \log(\hat{y}_k)$

$\rightarrow L(y, \hat{y}) = -\log(\hat{y}_k)$. $\qquad \hat{y}_k = \frac{e^{z_{2,k}}}{\sum_{i \in C} e^{z_{2,i}}}$

$\rightarrow L(y, \hat{y}) = -\log\left(e^{z_{2,k}} / \sum_{i \in C} e^{z_{2,i}}\right)$

$\rightarrow L(y, \hat{y}) = -z_{2,k} + \log\left(\sum_{i \in C} e^{z_{2,i}}\right)$, where k = class of y.

- $\partial L / \partial z_{2,i} = \frac{e^{z_{2,i}}}{\sum_{i \in C} e^{z_{2,i}}} - x, \quad x = \begin{cases} 1 \text{ if } i=k \\ 0 \text{ if } i \neq k. \end{cases}$

$\partial L / \partial z_{2,i} = \hat{y}_i$ for any $i \neq k$

$\partial L / \partial z_{2,i} = \hat{y}_i - 1$ for $i = k$.

Therefore, for $N = 1$,

$\partial L / \partial z_2 = \hat{Y} - Y$, since $Y_i = 1$ iff $i = k$.

Now let's observe $N > 1$.

$$L(Y, \hat{Y}) = \frac{-1}{N} \sum_{n \in N} \sum_{i \in C} Y_{n,i} \log(\hat{Y}_{n,i})$$

$$= \frac{-1}{N} \sum_{n \in N} \log(\hat{Y}_{n,k_n}), \text{ where } k_n \text{ is the class of } Y_n.$$

$$L(Y, \hat{Y}) = \frac{-1}{N} \sum_{n \in N} \log\left(e^{z_{2,n,k_n}} / \sum_{i \in C} e^{z_{2,n,i}} \right),$$

where $z_{2,n,i} = z_2[i]$ of the n-th training example.

$$L(Y, \hat{Y}) = \frac{-1}{N} \sum_{n \in N} z_{2,n}[k_n] - \log\left(\sum_{i \in C} e^{z_{2,n}[i]} \right)$$

$$= \frac{1}{N} \sum_{n \in N} -z_{2,n}[k_n] + \log\left(\sum_{i \in C} e^{z_{2,n}[i]} \right)$$

$$\partial L / \partial z_{2,i} = \frac{1}{N} \sum_{n \in N} -X_n + e^{z_{2,n}[i]} / \sum_{i \in C} e^{z_{2,n}[i]}, \quad X = \begin{cases} 1, & i = k \\ 0, & i \neq k \end{cases}$$

$$\partial L / \partial z_2 = \frac{1}{N} \sum_{n \in N} -Y_n + \hat{Y}_n$$

$$\boxed{\partial L / \partial z_2 = \frac{1}{n} \sum_{n \in N} \hat{Y}_n - Y_n}$$

S76 HW I. pg. 4

$A : BC = AC^T : B$

$AC^T : B = tr(B^T AC) = tr(AC B^T)$

$A : BC = BC : A = tr(A^T BC)$

$\partial L / \partial W_2 = ???$

$\partial L / \partial z_2 = \frac{1}{N} \sum_{n \in N} \hat{y}_n - y_n.$

Source: "Partial derivative of matrix product in neural network"

$z_2 = W_2 a_1 + b_2.$

$\partial z_2 / \partial W_2 = ???$

- Frobenius product: $A : B = tr(A^T B) = tr(B^T A)^*$ ← complex conjugate
  $= \sum_{ij} A_{ij} B_{ij}$ = sum of entry-wise product of A and B.
  (A and B must have same dimensions)

- Property of Frobenius inner product: ↓

$A : BC = AC^T : B$ ↵ $A : BC = B^T A : C$

Proof: $A : BC = tr(C^T B^T A)$

→ $tr(C^T B^T A) = tr(AC^T B^T)$ (cyclic traces)

→ $= B : AC^T = tr(B^T AC^T)$ (cyclic traces) ✓

Proof: $A : BC = tr((BC)^T A)$

$= tr(C^T B^T A)$

$= tr(C^T (B^T A))$

$= B^T A : C$ ✓

~~$\partial z_2 = \partial W_2 : \partial W_2$~~

~~$\partial z_2 = \partial W_2 a_1 + W_2 \partial a_1 + \partial b_1$~~

- $dL = \partial L / \partial z_2 : dz_2$, $dz_2 = dW_2 a_1 + W_2 da_1 + db_2$

↳ $dL = \partial L / \partial z_2 : dW_2 a_1 + \frac{\partial L}{\partial z_2} : W_2 da_1 + \frac{\partial L}{\partial z_2} : db_2$

↓ Hold a_1 and b_1 constant to differentiate wrt W_2, meaning $da_1 = db_1 = \emptyset$

$dL = \partial L / \partial z_2 : dW_2 a_1$ (wrt W_2)

↓ Frobenius property $A : BC = B^T A : C = AC^T : B$

$dL = \partial L / \partial z_2 \cdot a_1^T : dW_2$

↓

$\partial L / \partial W_2 = \partial L / \partial z_2 \cdot a_1^T.$

↓

$$\boxed{\partial L / \partial W_2 = (\partial L / \partial z_2)\, a_1^T}$$

(outer product!)

The second layer has C neurons, $\partial L / \partial z_2$ is a C-length vector, and a_1 has as many elements as there are neurons in the first layer, so $\partial L / \partial W_2$ has the right dimension (same as W_2).

$\partial L/\partial b_2 = ??$

$z_2 = W_2 a_1 + b_2$

$\rightarrow \partial L/\partial b_2 = \partial L/\partial z_2 \cdot 1$

$\rightarrow \boxed{\partial L/\partial b_2 = \partial L/\partial z_2}$ has correct number (C) of elements. ✓

$\partial L/\partial W_1 = ?$

~~$\partial L/\partial z_1 = ?$~~ $z_1 = \text{actFun}(a_1)$

$\downarrow$

~~$\partial L/\partial z_1 = \text{actFun}'(a_1) \cdot \partial L/\partial a_1$~~

~~$\partial L/\partial a_1 = ?$~~

$\partial L/\partial z_2 = \frac{1}{N}\sum \hat{y}_n - y_n$ $\quad z_2 = W_2 a_1 + b_2$

$\partial L/\partial W_2 = \partial L/\partial z_2 \cdot a_1^T$

~~[illegible]~~

$\partial L/\partial a_1 = ?? \rightarrow dL = \partial L/\partial z_2 : dz_2$

$dL = \partial L/\partial z_2 : dW_2 a_1 + \frac{\partial L}{\partial z_2} : W_2 da_1 + \frac{\partial L}{\partial z_2} : d$

wrt $a_1 \rightarrow dL = \partial L/\partial z_2 : W_2 da_1$

$\rightarrow dL = W_2^T \partial L/\partial z_2 : da_1$ (A:BC = B^T

$\rightarrow \partial L/\partial a_1 = W_2^T \partial L/\partial z_2$

$a_1 = \text{act Fun}(z_1)$. $\partial L/\partial z_1 = ?$

$\partial L/\partial z_1 = \text{actFun}'(z_1) \cdot \partial L/\partial a_1$

$\partial L/\partial z_1 = \text{actFun}'(z_1) \cdot W_2^T \partial L/\partial z_2$

ELEC 576 HW 2 pg. 6

$$\partial L/\partial z_1 = actFun'(z_1) \cdot W_2^T \partial L/\partial z_2$$

$$z_1 = W_1 X + b_1$$

$$\downarrow$$

$$dL = \partial L/\partial z_1 : dz_1$$

$$dL = \partial L/\partial z_1 : dW_1 X + \partial L/\partial z_1 : W_1 dX + \partial L/\partial z_1 : db_1$$

$$\downarrow \text{ wrt } W_1$$

$$dL = \partial L/\partial z_1 : dW_1 X$$

$$\downarrow$$

$$dL = \partial L/\partial z_1 X^T : dW_1$$

$$\downarrow$$

$$\boxed{\partial L/\partial W_1 = \partial L/\partial z_1 X^T}$$

$$\boxed{\partial L/\partial b_1 = \partial L/\partial z_1}$$