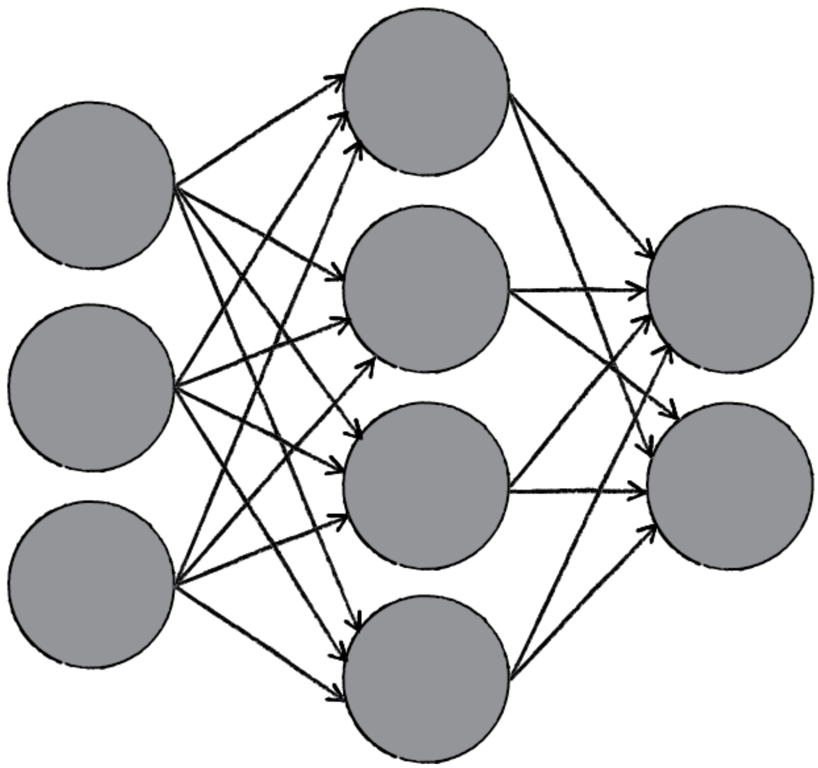
ELEC 576: Assignment 1

Wilfredo J. Molina

October 3, 2018



Contents

1 Big Picture

Suppose that we have a set $\{x_i\}_{i=1}^N \subset \mathbb{R}^{n_1}$ of $N \in \mathbb{N}$ *data points*, each with $n_1 \in \mathbb{N}$ *dimensions*, and that we have *classified* each such point as belonging to one of $n_L \in \mathbb{N}$ *classes*.

Let $\{y_i\}_{i=1}^N \subset \mathscr{C}^{n_L}$ and note that the information that x_i belongs to a particular class is captured by the ordered pair (x_i, y_i).

Our goal is to find a function

$$\begin{aligned}\Psi : \mathbb{R}^{n_1} \to (0,1)^{n_L} \text{ such that } -\frac{1}{N}\sum_{i=1}^{N} \langle y_i, \ln\left(\Psi\left(x_i\right)\right)\rangle \text{ is \textit{close enough} to 0 and that}\\ \langle \Psi\left(x\right), (\underbrace{1,1,\ldots,1}_{n_L \text{ times}})\rangle = 1 \text{ for every } x \in \mathbb{R}^{n_1}.\end{aligned} \tag{1.1}$$

In doing so, we hope that Ψ can help us classify points outside of $\{x_i\}_{i=1}^N$ in a manner that is consistent with our expectations, vague as they may be.

Note that $\mathscr{C}^{n_L}$ and the co-domain of Ψ are chosen in this way to represent probabilities and that we implicitly extended $\ln : (0,\infty) \to \mathbb{R}$ above to $\ln : (0,\infty)^{n_L} \to \mathbb{R}^{n_L}$ point-wise, i.e., $\ln\left(x_1, x_2, \ldots, x_{n_L}\right) = \left(\ln\left(x_1\right), \ln\left(x_2\right), \ldots, \ln\left(x_{n_L}\right)\right)$. We will continue to do this as it should be unambiguous in this context.

2 Neural Network of 3 Layers

$$
\begin{aligned}
\Psi : \mathbb{R}^{n_1} &\to (0,1)^{n_3}\,, \text{ defined by} \\
x &\mapsto \sigma_{n_3}\left(W_2\left(\varphi\left(W_1\left(x\right)+b_1\right)\right)+b_2\right),
\end{aligned}
\tag{2.1}
$$

is a *neural network of* 3 *layers*, where $W_i \in \operatorname{Hom}\left(\mathbb{R}^{n_i}, \mathbb{R}^{n_{i+1}}\right)$, $b_i \in \mathbb{R}^{n_{i+1}}$, $\varphi \in \mathscr{D}$, and $n_i \in \mathbb{N}$.

Given data points $\{x_i\}_{i=1}^{N} \subset \mathbb{R}^{n_1}$ and classes $\{y_i\}_{i=1}^{N} \subset \mathscr{C}^{n_3}$, we expect Ψ to be a *good enough* candidate for (1.1) for a suitable choice of W_i, b_i, and φ.

We justify our requirement that φ be in $\mathscr{D}$ by stating that it is *unlikely* that we will evaluate it at the points where it is not differentiable if we initialize W_i and b_i *well enough*. However, note that $\varphi : \mathbb{R} \to \mathbb{R}$, defined by $x \mapsto \sum_{n\in\mathbb{N}} \left|\sin\left(n\pi x\right)\right| / n^3$, is differentiable only on $\mathbb{R} \setminus \mathbb{Q}$, i.e., $\varphi \in \mathscr{D}$, but no machine can represent the points where φ is differentiable. Therefore, in practice, our requirement translates to φ being a function such that $|D \cap R|$ is *small enough*, where D is the set of points where φ is not differentiable and $R \subset \mathbb{R}$ is the set of numbers that a particular machine can represent (after all, R has Lebesgue measure 0 for every machine that has ever been and will be made).

2.1 Back-Propagation

Given data points $\{x_i\}_{i=1}^{N} \subset \mathbb{R}^{n_1}$ and classes $\{y_i\}_{i=1}^{N} \subset \mathscr{C}^{n_3}$, we fix $\varphi \in \mathscr{D}$, initialize W_i and b_i *well enough*, and attempt to find a suitable, *a posteriori* choice of W_i and b_i for (2.1) via an algorithm called *back-propagation*. To this end, we define the *loss function*

$$\begin{aligned}E : \mathbb{R}^{n_1} \times \mathscr{C}^{n_3} \times \operatorname{Hom}\left(\mathbb{R}^{n_1}, \mathbb{R}^{n_2}\right) \times \mathbb{R}^{n_2} \times \operatorname{Hom}\left(\mathbb{R}^{n_2}, \mathbb{R}^{n_3}\right) \times \mathbb{R}^{n_3} \to \mathbb{R} \text{ by}& \\ \left(x, y, W_1, b_1, W_2, b_2\right) \mapsto - \left\langle y, \ln\left(\sigma_{n_3}\left(W_2\left(\varphi\left(W_1\left(x\right) + b_1\right)\right) + b_2\right)\right)\right\rangle&\end{aligned} \tag{2.2}$$

and, in the interest of implementation, fix $j \in \{1, 2, \ldots, N\}$ and let

$$\begin{aligned}a_0 &\coloneqq x_j,\\ z_1 &\coloneqq W_1\left(a_0\right) + b_1,\\ a_1 &\coloneqq \varphi\left(z_1\right),\\ z_2 &\coloneqq W_2\left(a_1\right) + b_2,\\ a_2 &\coloneqq \sigma_{n_3}\left(z_2\right), \text{ and}\\ \delta_i &\coloneqq \nabla_{z_i} E.\end{aligned}$$

Then

$$\begin{aligned}\delta_2 &\coloneqq a_2 - y \text{ and}\\ \delta_1 &\coloneqq W_2^*\left(\delta_2\right) \odot \varphi'\left(z_1\right).\end{aligned}$$

Letting $\eta > 0$ be *close enough* to 0 (we call η the *learning rate*), we hope that repeating the assignments

$$\begin{aligned}W_i &\leftarrow W_i - \eta \overline{a_{i-1}} \otimes \delta_i \text{ and}\\ b_i &\leftarrow b_i - \eta \delta_i\end{aligned}$$

enough times for every $j \in \{1, 2, \ldots, N\}$ will *eventually* yield a *good enough* candidate for (1.1), which we consider to be *trained*.

2.2 Experiments

Each picture in the next few pages is a visual representation of 1000 distinct elements of some square $[a, b] \times [c, d] \subset \mathbb{R}^2$, each belonging to one of two classes (red or blue), where 0.000, 0.025, and 0.075 represent different levels of *noise* (0.000 represents no noise), and of the *decision boundary* of a trained neural network of 3 layers with $n_1 = 2$, $n_2 \in \{10, 20, 30\}$, $n_3 = 2$,

$$\varphi(x) = \begin{cases} 0 & \text{if } x < 0 \text{ and} \\ x & \text{if } x \geq 0, \end{cases} \tag{ReLU}$$

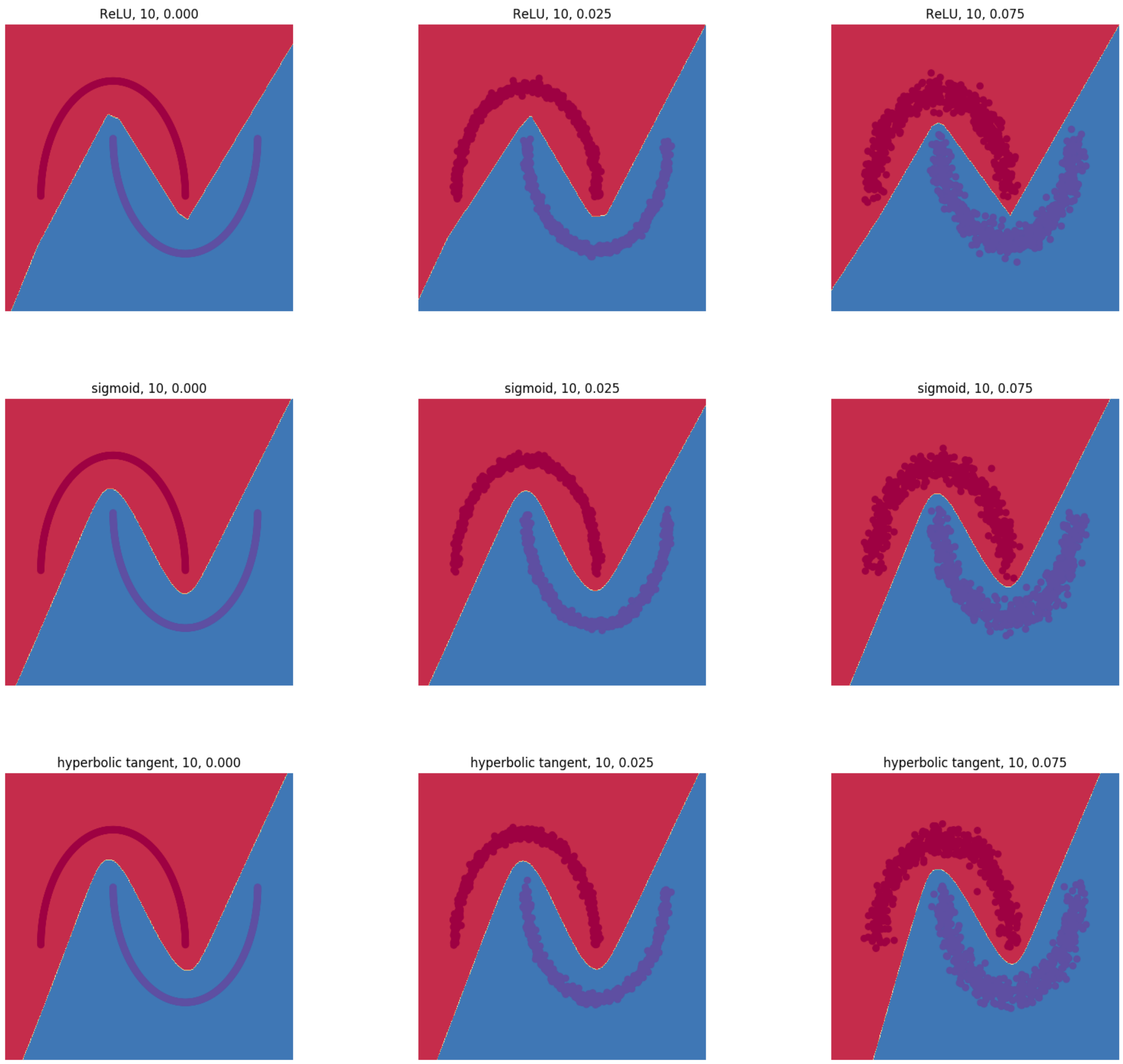
$$\varphi(x) = \frac{1}{1 + e^{-x}}, \text{ and} \tag{sigmoid}$$

$$\varphi(x) = \tanh(x). \tag{hyperbolic tangent}$$

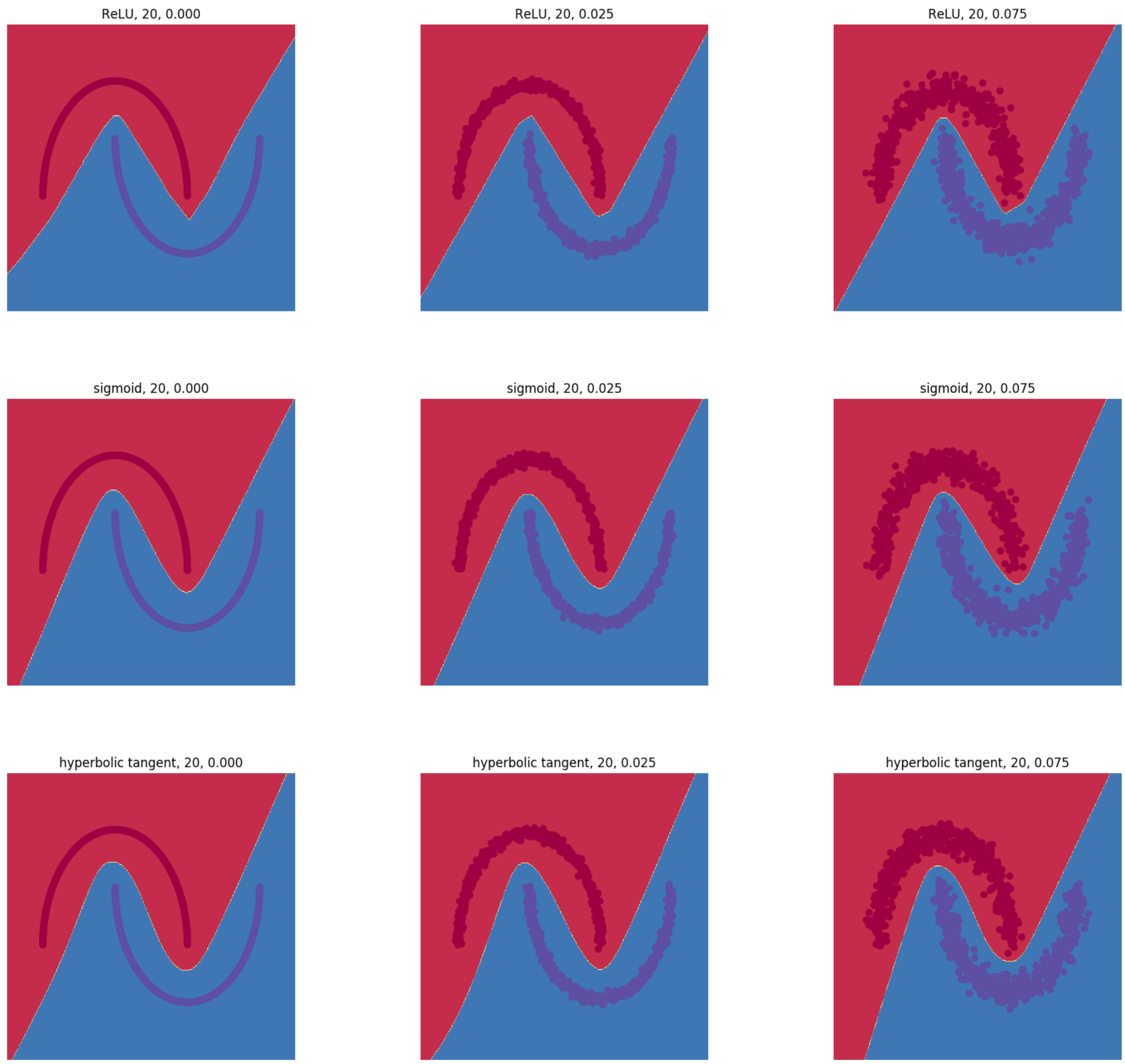
We implement Ψ in Python as a class that inherits all of its fields and methods from the class that we implemented for (3.1): the following is the entire code for this class.

```
class ThreeLayerNeuralNetwork(DeepNeuralNetwork):
    def __init__(self,
                 input_dim = 2,
                 hidden_dim = 10,
                 output_dim = 2,
                 activation_type = 'relu',
                 regularization = 0,
                 random_seed = None):
        super().__init__([input_dim,
                         hidden_dim,
                         output_dim],
                         activation_type,
                         regularization,
                         random_seed)
```

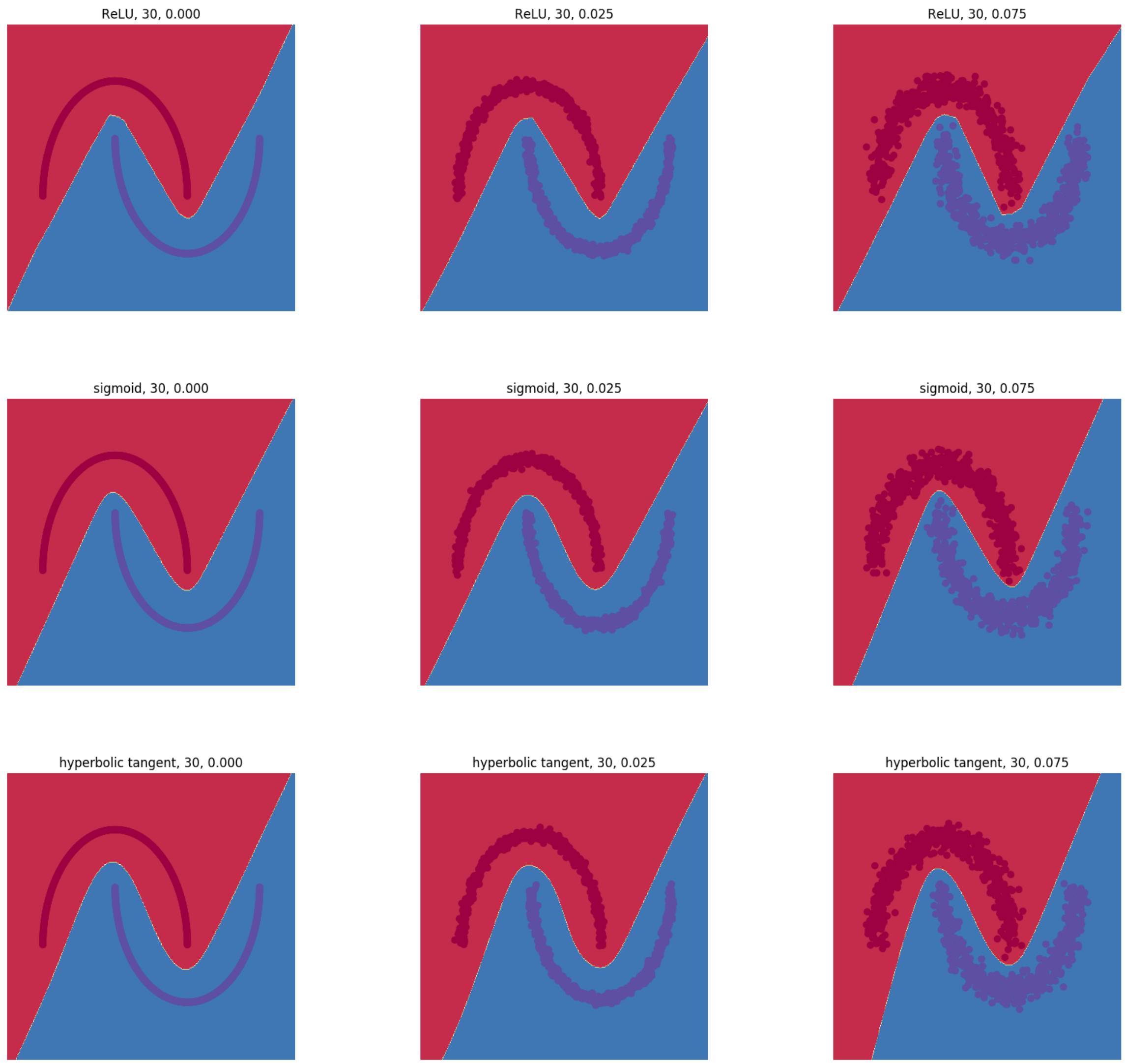
2.2.1 $n_2 = 10$, make_moons Data Set



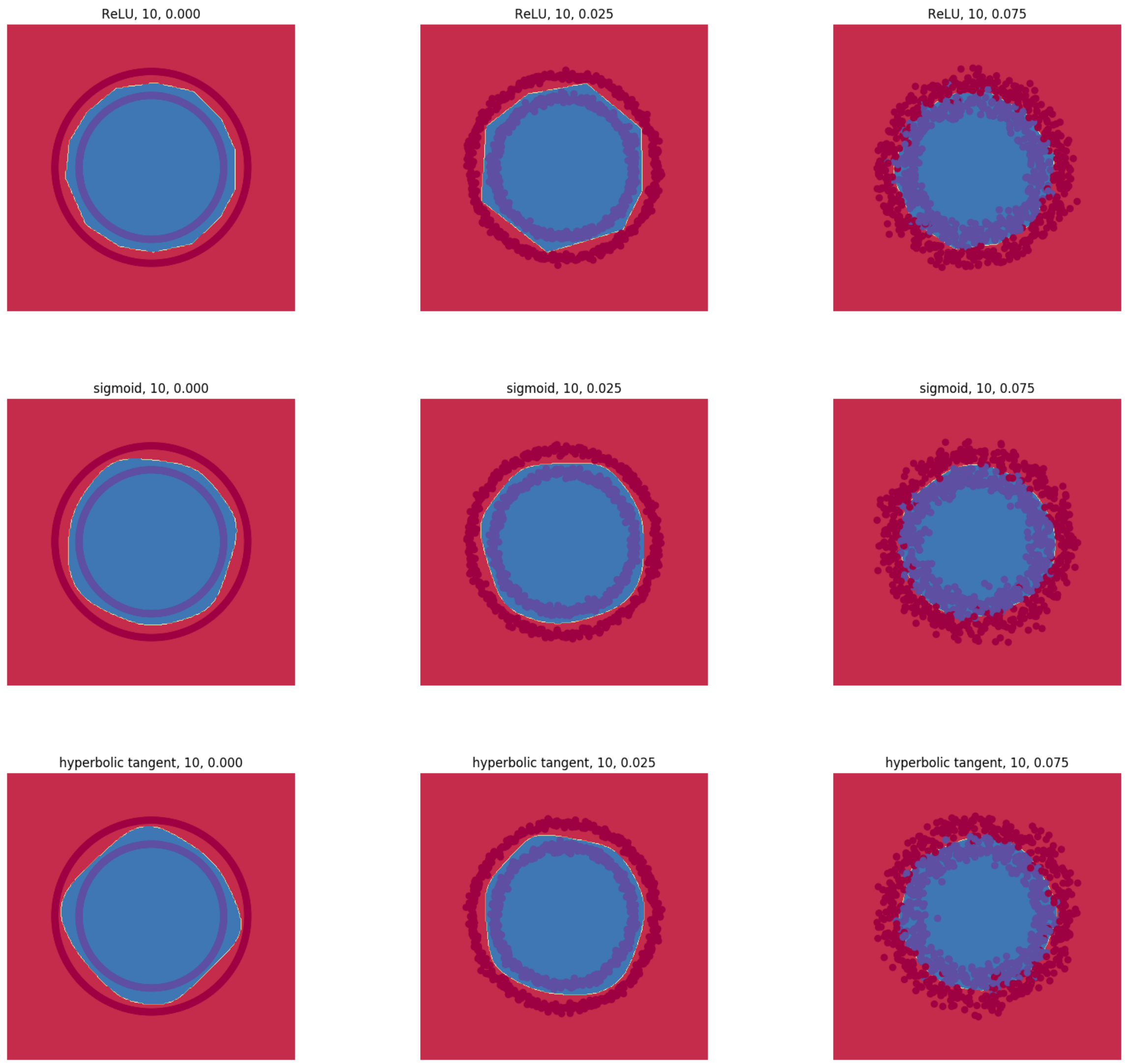
2.2.2 $n_2 = 20$, make_moons Data Set



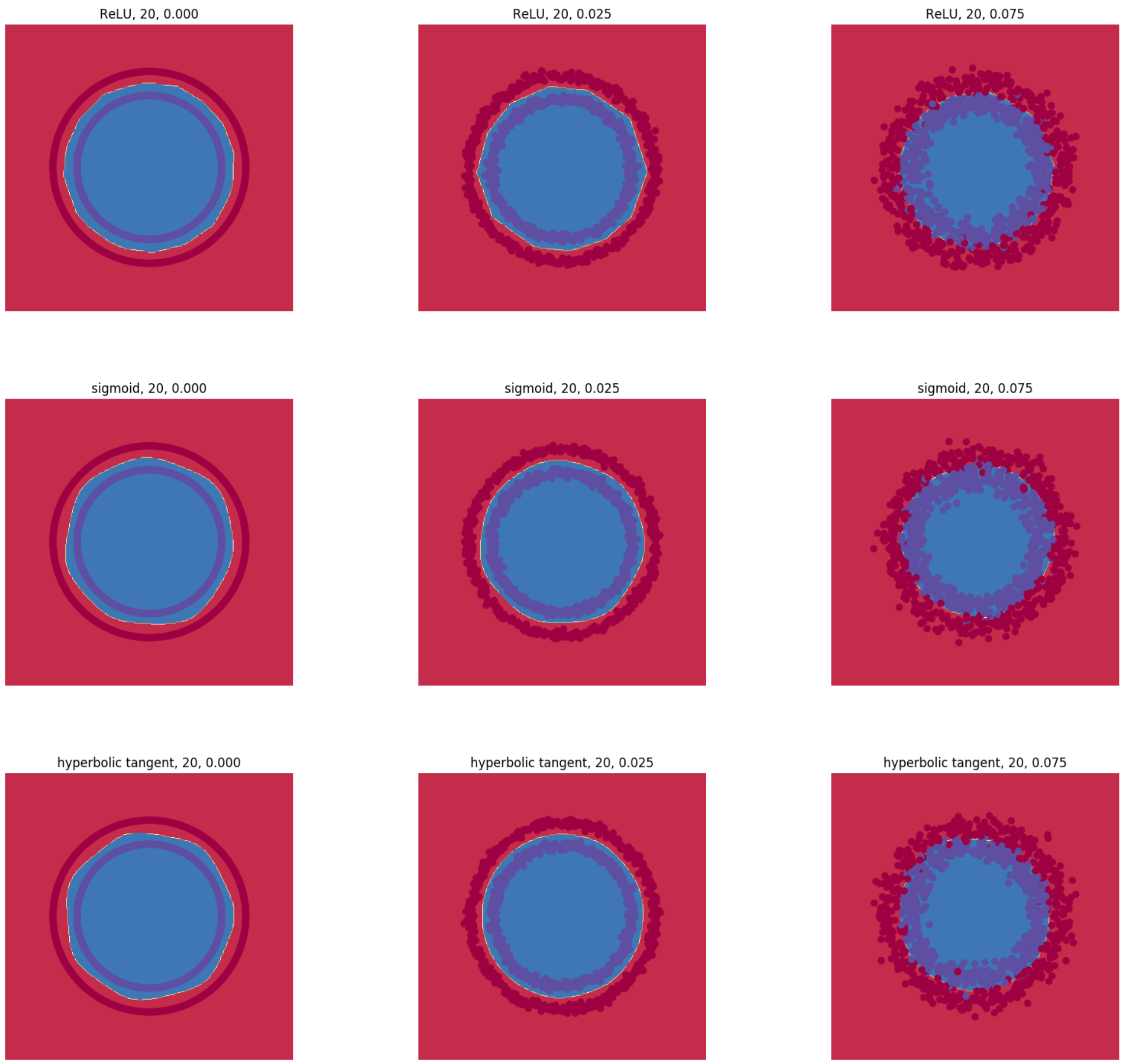
2.2.3 $n_2 = 30$, make_moons Data Set



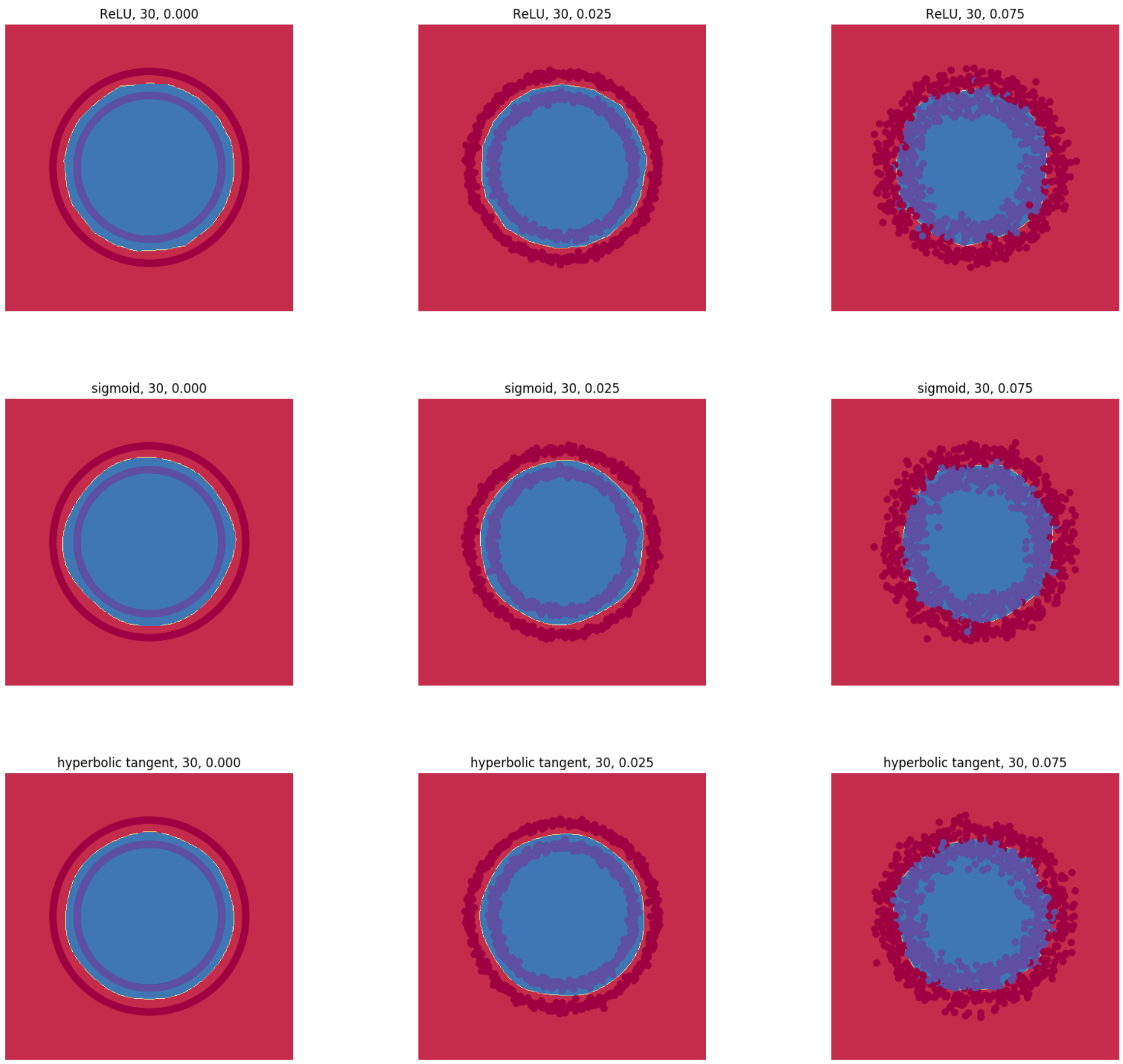
2.2.4 $n_2 = 10$, make_circles Data Set



2.2.5 $n_2 = 20$, make_circles Data Set



2.2.6 $n_2 = 30$, make_circles Data Set



3 Neural Network of n Layers

$$\begin{aligned}\Psi : \mathbb{R}^{n_1} &\to (0,1)^{n_L}\,, \text{ defined by} \\ x &\mapsto \sigma_{n_L}\left(W_{L-1}\left(\varphi\left(W_{L-2}\left(\cdots\varphi\left(W_1\left(x\right)+b_1\right)\cdots\right)+b_{L-2}\right)\right)+b_{L-1}\right),\end{aligned} \tag{3.1}$$

is a *neural network of* $L \in \mathbb{N}$ *layers*, where $W_i \in \operatorname{Hom}\left(\mathbb{R}^{n_i}, \mathbb{R}^{n_{i+1}}\right)$, $b_i \in \mathbb{R}^{n_{i+1}}$, $\varphi \in \mathscr{D}$, $n_i \in \mathbb{N}$, and $L > 1$.

Given data points $\{x_i\}_{i=1}^N \subset \mathbb{R}^{n_1}$ and classes $\{y_i\}_{i=1}^N \subset \mathscr{C}^{n_L}$, we expect Ψ to be a *better* candidate than (2.1) for (1.1) for a suitable choice of W_i, b_i, L, and φ.

3.1 Back-Propagation

Given data points $\{x_i\}_{i=1}^N \subset \mathbb{R}^{n_1}$ and classes $\{y_i\}_{i=1}^N \subset \mathscr{C}^{n_3}$, we fix $\varphi \in \mathscr{D}$, initialize W_i and b_i *well enough*, and attempt to find a suitable, *a posteriori* choice of W_i and b_i for (3.1).To this end, we fix $j \in \{1, 2, \ldots, N\}$ and let

$$\begin{aligned} a_0 &:= x_j, \\ z_i &:= W_i\left(a_{i-1}\right) + b_i, \\ a_i &:= \varphi\left(z_i\right), \\ a_{L-1} &:= \sigma_{n_L}\left(z_{L-1}\right), \text{ and} \\ \delta_i &:= \nabla_{z_i} E, \end{aligned}$$

where the definition of E is analogous to that of (2.2). Then

$$\begin{aligned} \delta_{L-1} &:= a_{L-1} - y \text{ and} \\ \delta_i &:= W_{i+1}^*\left(\delta_{i+1}\right) \odot \varphi'\left(z_i\right). \end{aligned}$$

Letting $\eta > 0$ be *close enough* to 0, we hope that repeating the assignments

$$\begin{aligned} W_i &\leftarrow W_i - \eta \overline{a_{i-1}} \otimes \delta_i \text{ and} \\ b_i &\leftarrow b_i - \eta \delta_i, \end{aligned}$$

enough times for every $j \in \{1, 2, \ldots, N\}$ will *eventually* yield a *good enough* candidate for (1.1).

3.2 Experiments

Each picture in the next few pages is a visual representation of 1000 distinct elements of some square $[a, b] \times [c, d] \subset \mathbb{R}^2$, each belonging to one of two classes (red or blue), where 0.000, 0.025, and 0.075 represent different levels of *noise* (0.000 represents no noise), and of the *decision boundary* of a trained neural network of 5 layers with $n_1 = 2$, $(n_2, n_3, n_4) \in \{(12, 12, 12), (16, 8, 16), (10, 16, 10)\}$, $n_5 = 2$,

$$\varphi(x) = \begin{cases} 0 & \text{if } x < 0 \text{ and} \\ x & \text{if } x \geq 0, \end{cases} \tag{ReLU}$$

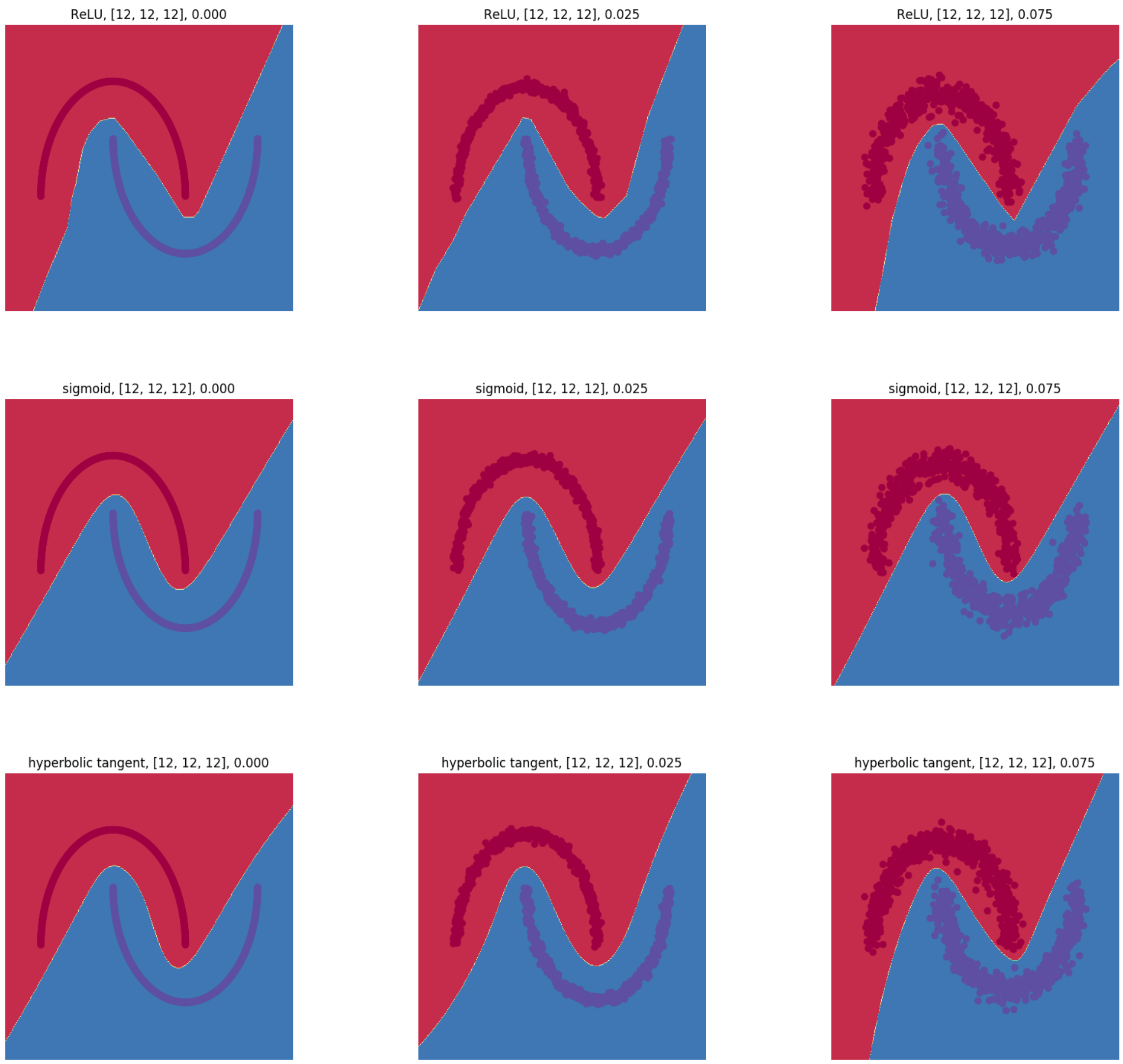
$$\varphi(x) = \frac{1}{1 + e^{-x}}, \text{ and} \tag{sigmoid}$$

$$\varphi(x) = \tanh(x). \tag{hyperbolic tangent}$$

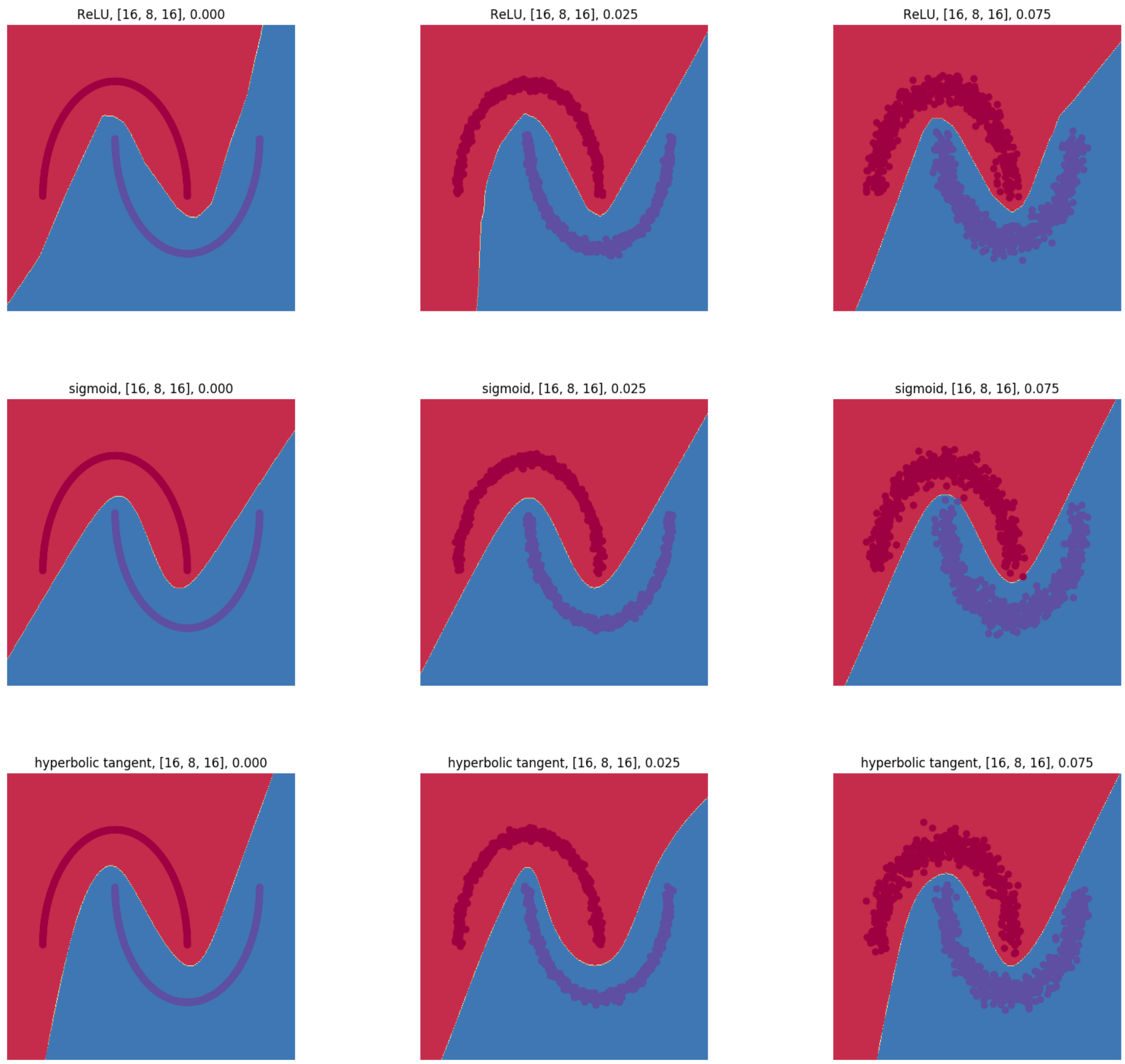
We implement Ψ in Python as a class with the following fields and methods:

```python
self.dimensions      # NumPy Array
self.activation_type # String
self.regularization  # Number
self.W               # Dictionary of NumPy Arrays
self.b               # Dictionary of NumPy Arrays
self.a               # Dictionary of NumPy Arrays
self.z               # Dictionary of NumPy Arrays

def __init__(self,
             dimensions = np.array([2, 10, 2]),
             activation_type = 'relu',
             regularization = 0,
             random_seed = None)
def feed_forward(self, a1)
def activation(self, x)
def activation_derivative(self, x)
def soft_max(x)
def back_propagation(self, a1, y)
def train(self, a1, y, train_rate, passes, print_loss, print_rate)
def calculate_loss(self, a1, y)
def predict(self, a1)
```

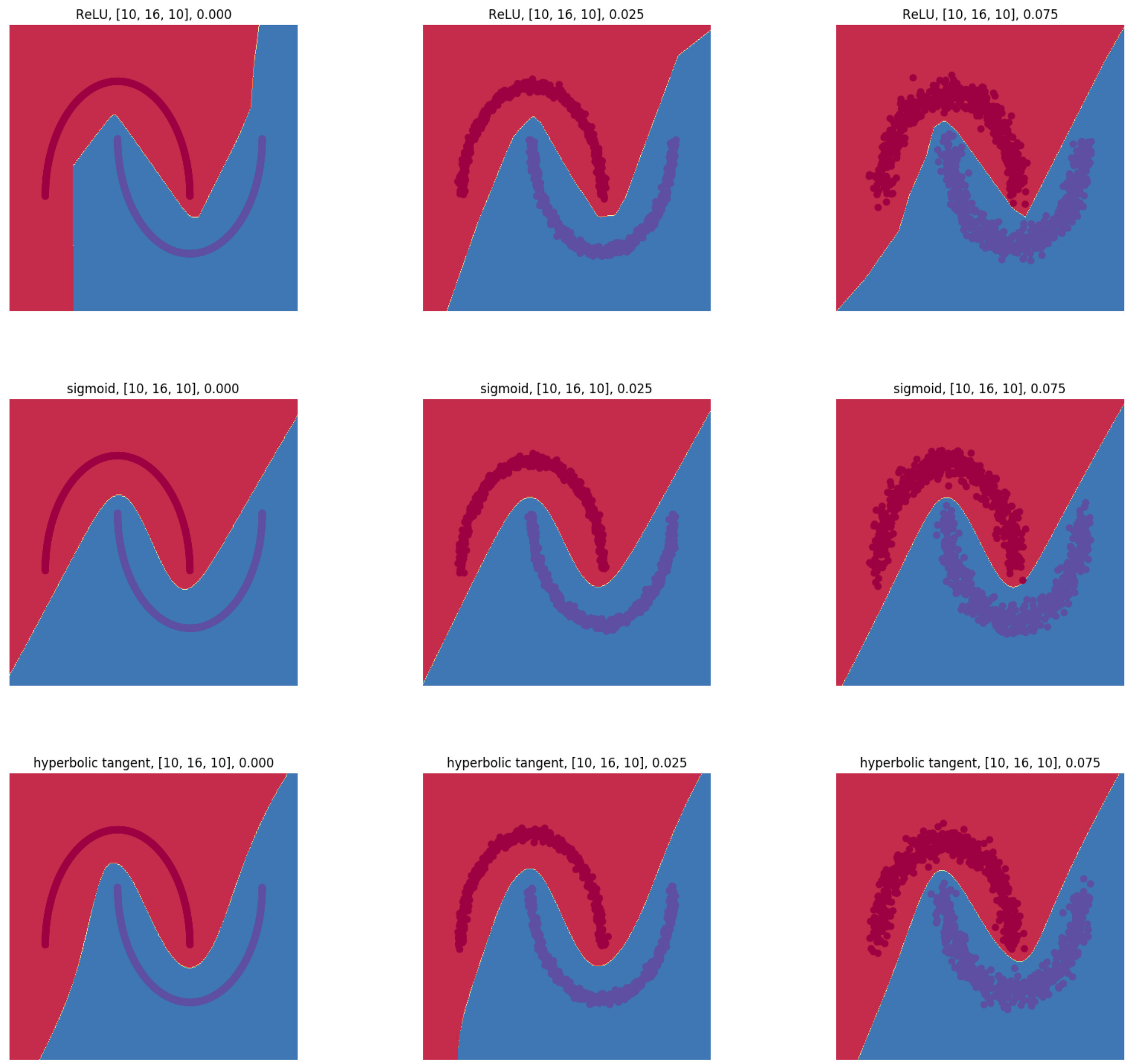
3.2.1 $(n_2, n_3, n_4) = (12, 12, 12)$, make_moons Data Set



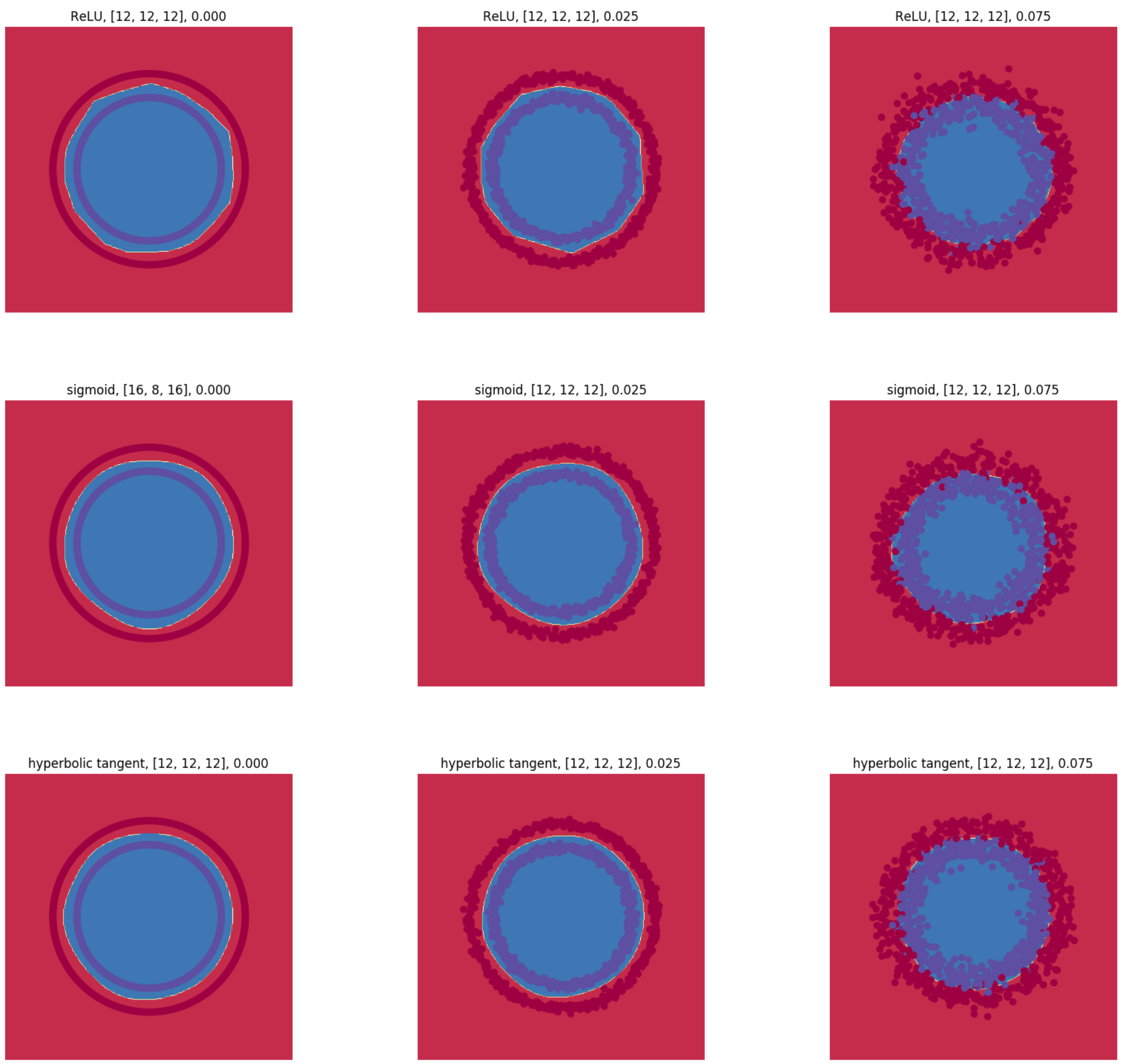
3.2.2 $(n_2, n_3, n_4) = (16, 8, 16)$, make_moons Data Set



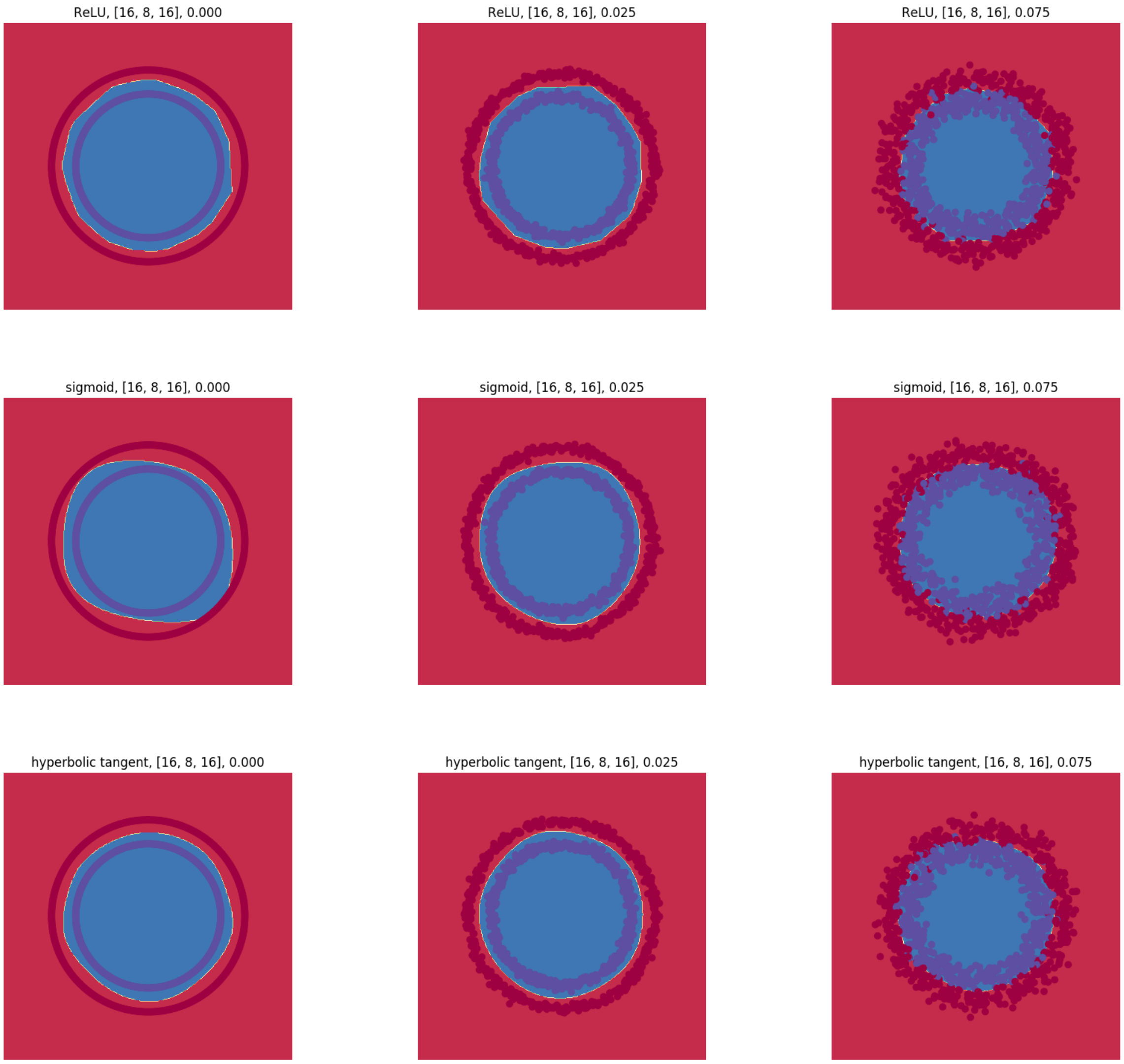
3.2.3 $(n_2, n_3, n_4) = (10, 16, 10)$, make_moons Data Set



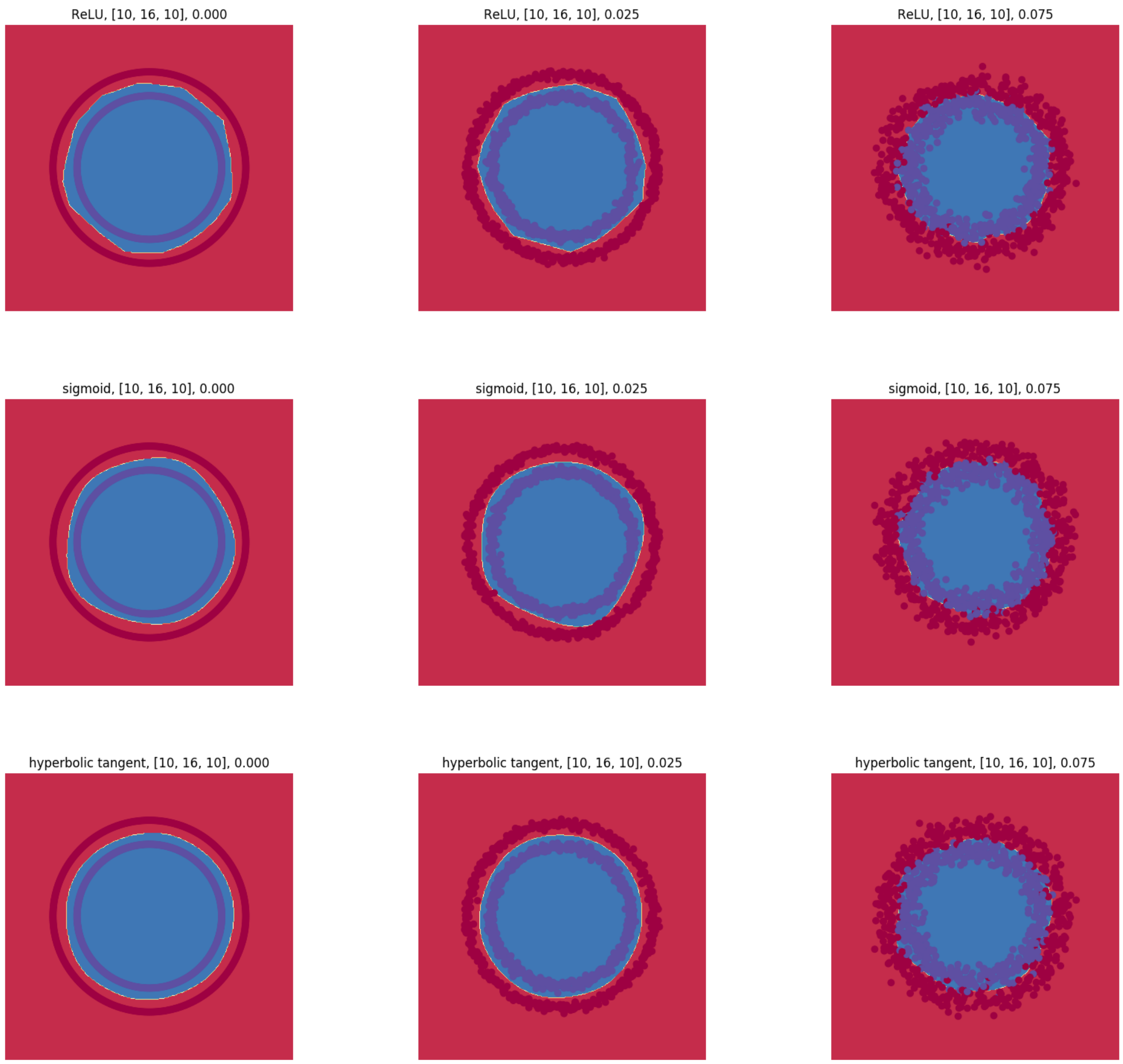
3.2.4 $(n_2, n_3, n_4) = (12, 12, 12)$, make_circles Data Set



3.2.5 $(n_2, n_3, n_4) = (16, 8, 16)$, make_circles Data Set



3.2.6 $(n_2, n_3, n_4) = (10, 16, 10)$, make_circles Data Set

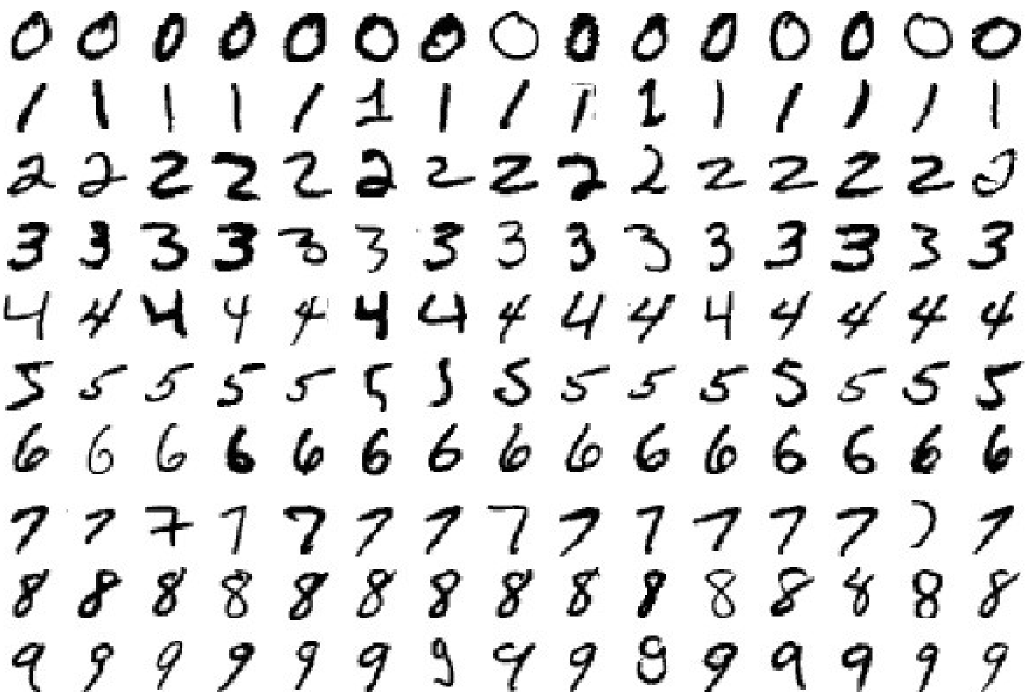


4 Convolutional Neural Network

We implement a convolutional neural network with the following architecture:

```
Convolution(5-5-1-32) - ReLU - MaxPool(2-2) - Convolution(5-5-1-64) - ReLU
                       - MaxPool(2-2) - Flatten(1024) - ReLU - DropOut(.5) - SoftMax(10)
```

and train it on MNIST, which is a database of 60000 $28 \times 28 \times 1$ images that look like this:



Using Keras, this can be done as follows:

```
model = Sequential()
model.add(Conv2D(32, kernel_size = (5, 5), activation = 'relu', input_shape = input_shape))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (5, 5), activation = 'relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
model.add(Dense(1024, activation = 'relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation = 'softmax'))
```

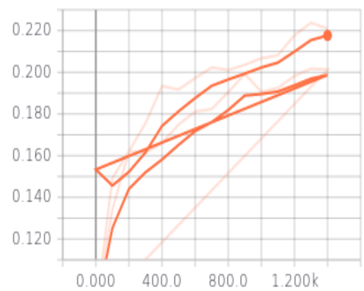
We train our convolutional neural network for 5 epochs with a batch size of 50. This can be done as follows:

```
(x_train, y_train), (x_test, y_test) = mnist.load_data()

model.compile(loss      = keras.losses.categorical_crossentropy,
              optimizer = keras.optimizers.Adadelta(),
              metrics   = ['accuracy'])

model.fit(x_train,
          y_train,
          batch_size      = 50,
          epochs          = 5,
          validation_data = (x_test, y_test))
```

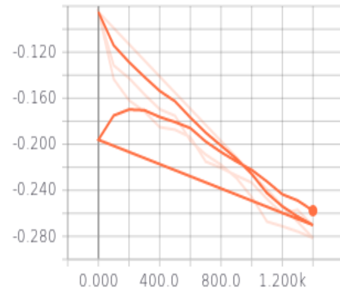
With this setup, we obtain a validation accuracy of 0.9882.

In the next few pages, we visualize our training using tensorboard.

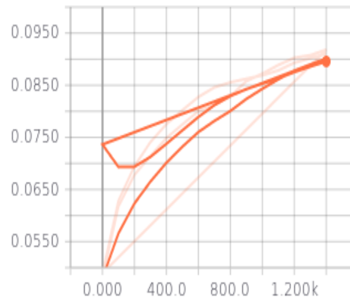
Layer1/biases/stddev/max_1



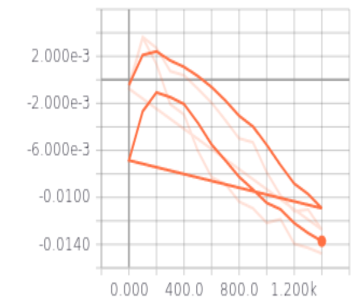
Layer1/biases/stddev/min_1



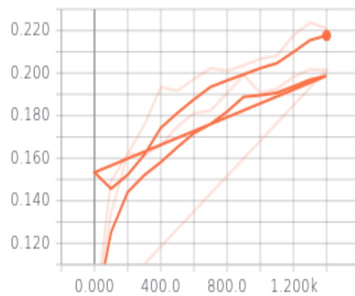
Layer1/biases/stddev/stddev



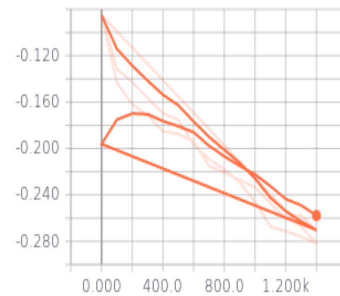
Layer1/biases/summaries/mean_1



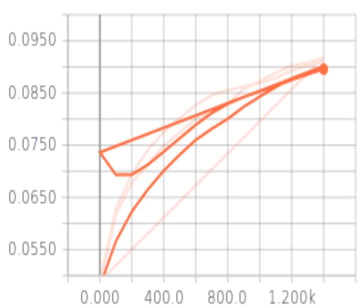
Layer1/weights/stddev/max_1



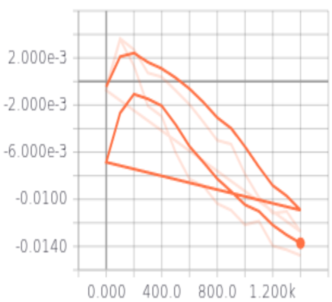
Layer1/weights/stddev/min_1



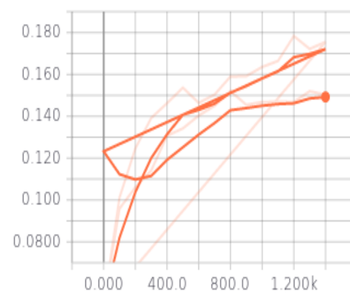
Layer1/weights/stddev/stddev



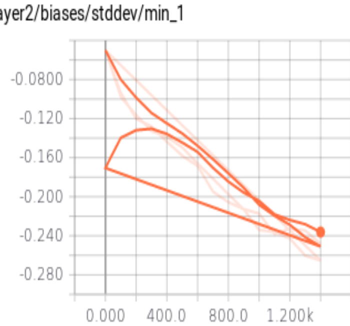
Layer1/weights/summaries/mean_1



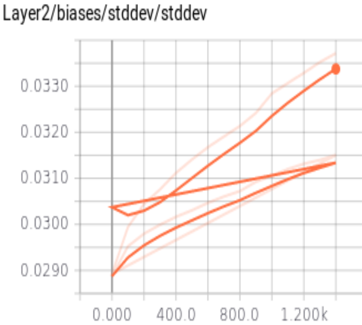
Layer2/biases/stddev/max_1



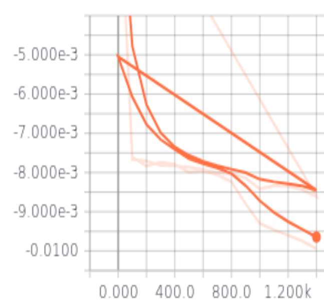
Layer2/biases/stddev/min_1



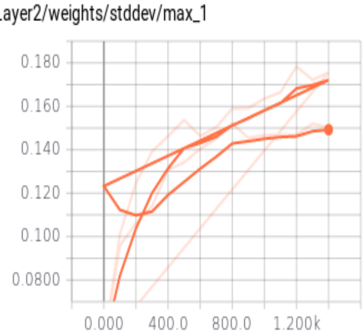
Layer2/biases/stddev/stddev



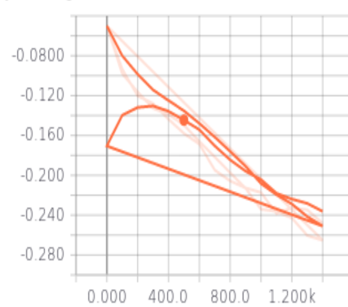
Layer2/biases/summaries/mean_1



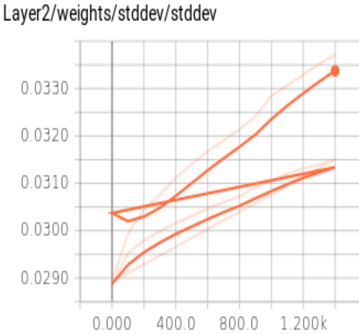
Layer2/weights/stddev/max_1

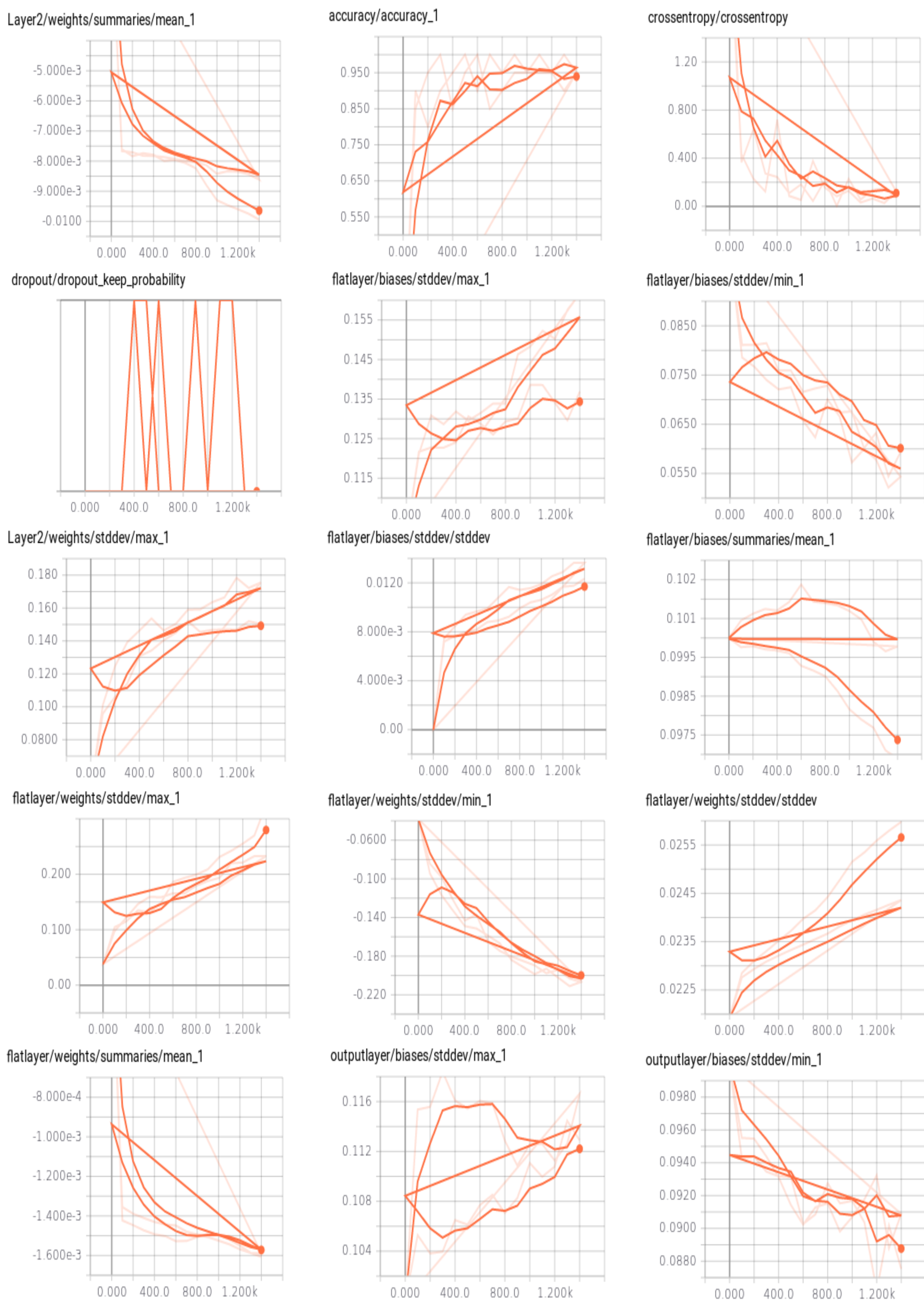


Layer2/weights/stddev/min_1

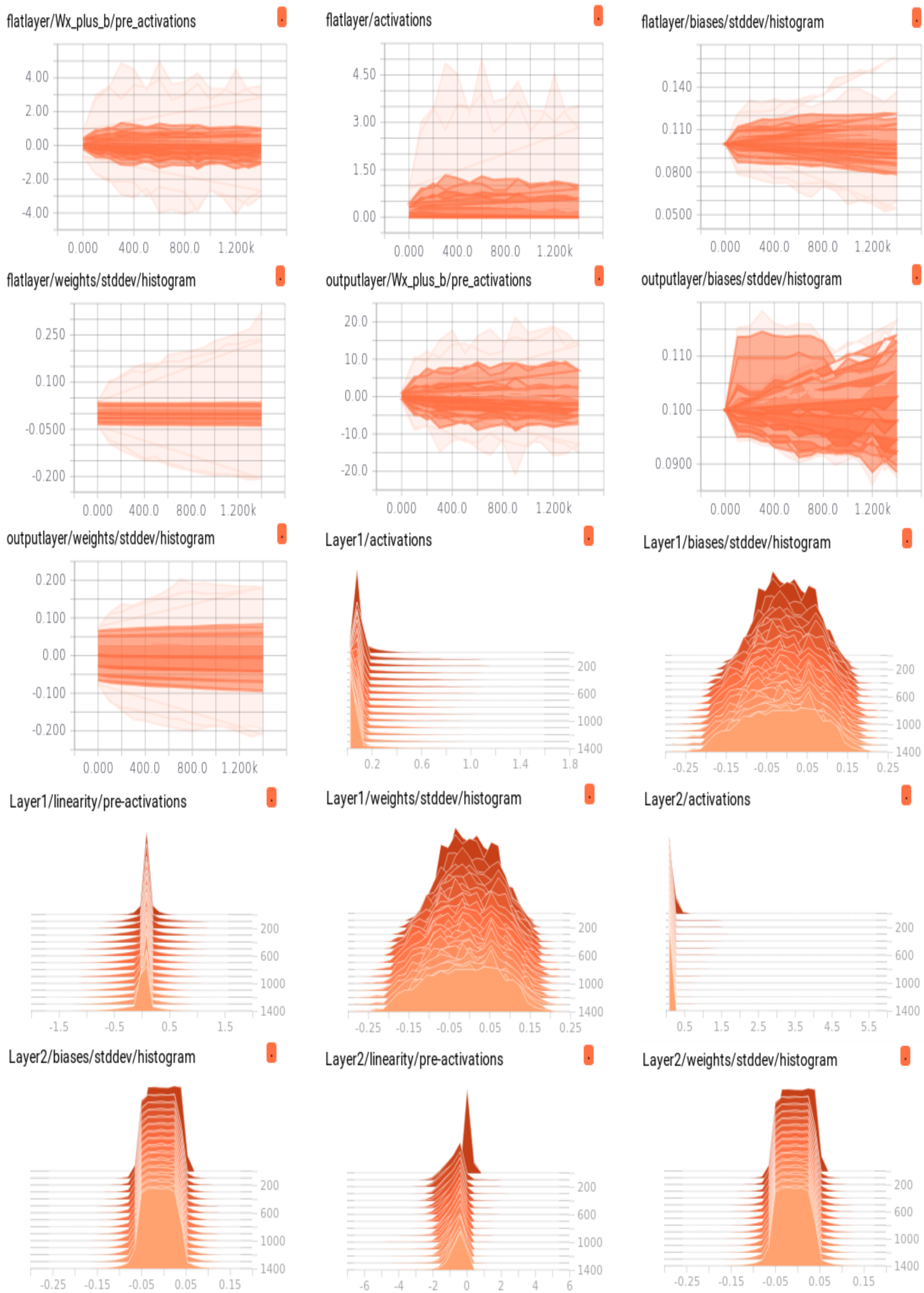


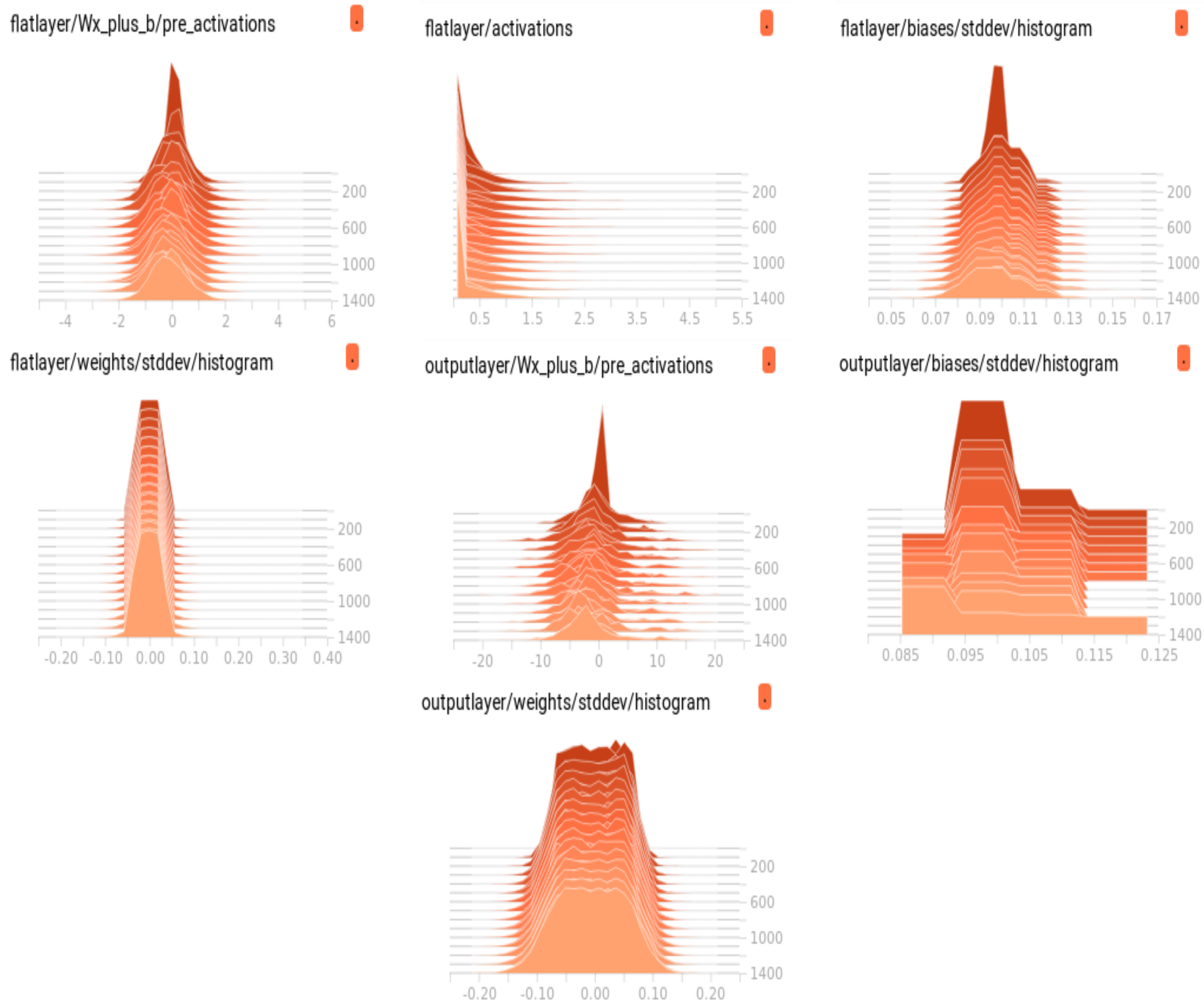
Layer2/weights/stddev/stddev

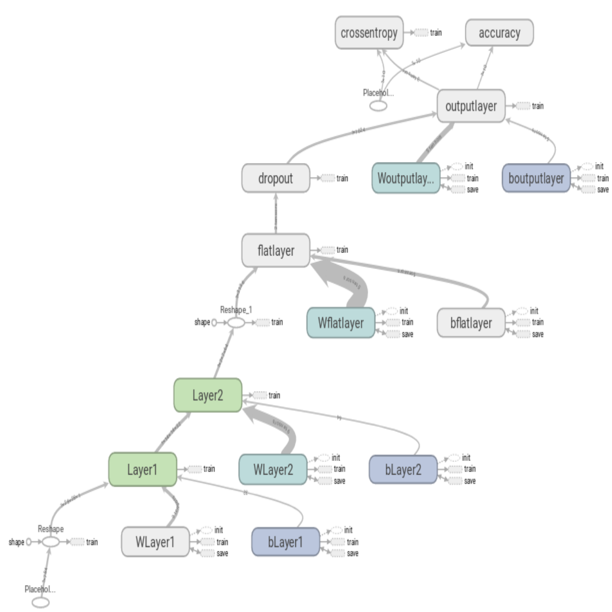


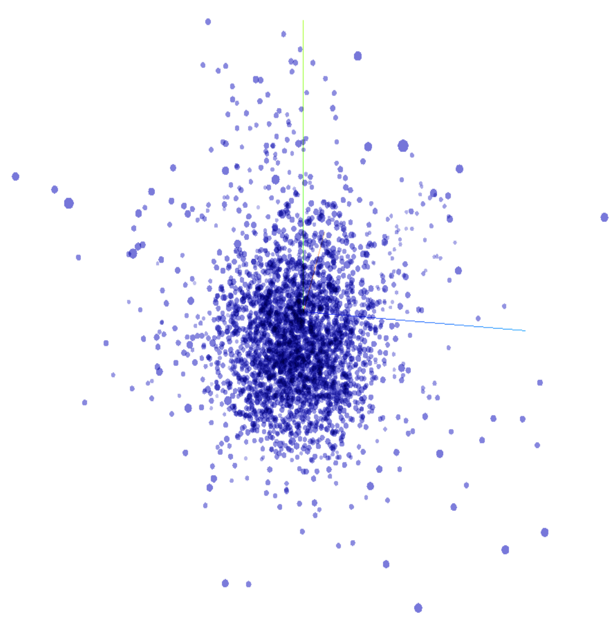

Layer2/weights/summaries/mean_1
-5.000e-3
-6.000e-3
-7.000e-3
-8.000e-3
-9.000e-3
-0.0100
0.000 400.0 800.0 1.200k
accuracy/accuracy_1
0.950
0.850
0.750
0.650
0.550
0.000 400.0 800.0 1.200k
crossentropy/crossentropy
1.20
0.800
0.400
0.00
0.000 400.0 800.0 1.200k
dropout/dropout_keep_probability
0.000 400.0 800.0 1.200k
flatlayer/biases/stddev/max_1
0.155
0.145
0.135
0.125
0.115
0.000 400.0 800.0 1.200k
flatlayer/biases/stddev/min_1
0.0850
0.0750
0.0650
0.0550
0.000 400.0 800.0 1.200k
Layer2/weights/stddev/max_1
0.180
0.160
0.140
0.120
0.100
0.0800
0.000 400.0 800.0 1.200k
flatlayer/biases/stddev/stddev
0.0120
8.000e-3
4.000e-3
0.00
0.000 400.0 800.0 1.200k
flatlayer/biases/summaries/mean_1
0.102
0.101
0.0995
0.0985
0.0975
0.000 400.0 800.0 1.200k
flatlayer/weights/stddev/max_1
0.200
0.100
0.00
0.000 400.0 800.0 1.200k
flatlayer/weights/stddev/min_1
-0.0600
-0.100
-0.140
-0.180
-0.220
0.000 400.0 800.0 1.200k
flatlayer/weights/stddev/stddev
0.0255
0.0245
0.0235
0.0225
0.000 400.0 800.0 1.200k
flatlayer/weights/summaries/mean_1
-8.000e-4
-1.000e-3
-1.200e-3
-1.400e-3
-1.600e-3
0.000 400.0 800.0 1.200k
outputlayer/biases/stddev/max_1
0.116
0.112
0.108
0.104
0.000 400.0 800.0 1.200k
outputlayer/biases/stddev/min_1
0.0980
0.0960
0.0940
0.0920
0.0900
0.0880
0.000 400.0 800.0 1.200k


outputlayer/biases/stddev/stddev
8.000e-3
6.000e-3
4.000e-3
2.000e-3
0.00
0.000 400.0 800.0 1.200k
outputlayer/biases/summaries/mean_1
0.101
0.100
0.0998
0.0992
0.0986
0.000 400.0 800.0 1.200k
outputlayer/weights/stddev/max_1
0.200
0.180
0.160
0.140
0.120
0.100
0.000 400.0 800.0 1.200k
outputlayer/weights/stddev/min_1
-0.110
-0.130
-0.150
-0.170
-0.190
-0.210
0.000 400.0 800.0 1.200k
outputlayer/weights/stddev/stddev
0.0580
0.0540
0.0500
0.0460
0.000 400.0 800.0 1.200k
outputlayer/weights/summaries/mean_1
-1.500e-3
-2.500e-3
-3.500e-3
-4.500e-3
-5.500e-3
-6.500e-3
0.000 400.0 800.0 1.200k
Layer1/activations
1.20
0.800
0.400
0.00
0.000 400.0 800.0 1.200k
Layer1/biases/stddev/histogram
0.150
0.0500
-0.0500
-0.150
-0.250
0.000 400.0 800.0 1.200k
Layer1/linearity/pre-activations
1.50
0.500
-0.500
-1.50
0.000 400.0 800.0 1.200k
Layer1/weights/stddev/histogram
0.150
0.0500
-0.0500
-0.150
-0.250
0.000 400.0 800.0 1.200k
Layer2/activations
4.50
3.00
1.50
0.00
0.000 400.0 800.0 1.200k
Layer2/biases/stddev/histogram
0.150
0.0500
-0.0500
-0.150
-0.250
0.000 400.0 800.0 1.200k
Layer2/linearity/pre-activations
5.00
2.00
-1.00
-4.00
-7.00
0.000 400.0 800.0 1.200k
Layer2/weights/stddev/histogram
0.150
0.0500
-0.0500
-0.150
-0.250
0.000 400.0 800.0 1.200k
Layer2/weights/stddev/histogram
200
600
1000
1400
-0.25 -0.15 -0.05 0.05 0.15


flatlayer/Wx_plus_b/pre_activations
4.00 2.00 0.00 -2.00 -4.00
0.000 400.0 800.0 1.200k
flatlayer/activations
4.50 3.00 1.50 0.00
0.000 400.0 800.0 1.200k
flatlayer/biases/stddev/histogram
0.140 0.110 0.0800 0.0500
0.000 400.0 800.0 1.200k
flatlayer/weights/stddev/histogram
0.250 0.100 -0.0500 -0.200
0.000 400.0 800.0 1.200k
outputlayer/Wx_plus_b/pre_activations
20.0 10.0 0.00 -10.0 -20.0
0.000 400.0 800.0 1.200k
outputlayer/biases/stddev/histogram
0.110 0.100 0.0900
0.000 400.0 800.0 1.200k
outputlayer/weights/stddev/histogram
0.200 0.100 0.00 -0.100 -0.200
0.000 400.0 800.0 1.200k
Layer1/activations
200 600 1000 1400
0.2 0.6 1.0 1.4 1.8
Layer1/biases/stddev/histogram
200 600 1000 1400
-0.25 -0.15 -0.05 0.05 0.15 0.25
Layer1/linearity/pre-activations
200 600 1000 1400
-1.5 -0.5 0.5 1.5
Layer1/weights/stddev/histogram
200 600 1000 1400
-0.25 -0.15 -0.05 0.05 0.15 0.25
Layer2/activations
200 600 1000 1400
0.5 1.5 2.5 3.5 4.5 5.5
Layer2/biases/stddev/histogram
200 600 1000 1400
-0.25 -0.15 -0.05 0.05 0.15
Layer2/linearity/pre-activations
200 600 1000 1400
-6 -4 -2 0 2 4 6
Layer2/weights/stddev/histogram
200 600 1000 1400
-0.25 -0.15 -0.05 0.05 0.15


flatlayer/Wx_plus_b/pre_activations
flatlayer/activations
flatlayer/biases/stddev/histogram
flatlayer/weights/stddev/histogram
outputlayer/Wx_plus_b/pre_activations
outputlayer/biases/stddev/histogram
outputlayer/weights/stddev/histogram


crossentropy
accuracy
outputlayer
dropout
Woutputlay...
boutputlayer
flatlayer
Wflatlayer
bflatlayer
Layer2
Layer1
WLayer2
bLayer2
WLayer1
bLayer1



5 Definitions and Remarks

Definition 5.1. $\mathbb{N} := \{1, 2, \dots\}$ is the set of *natural numbers*.

Definition 5.2. $\mathbb{R}$ is the set of *real numbers*.

Definition 5.3. $\mathscr{C}^n := \left\{(c_1, c_2, \dots, c_n) \in \mathbb{R}^n : c_i = 1 \text{ and } c_j = 0 \text{ if } j \neq i, \text{ where } i \in \{1, 2, \dots, n\}\right\}$ is the set of *n classes*.

Definition 5.4. $\mathrm{Hom}\,(V, W)$ is the set of *linear maps* $V \to W$, i.e., $T \in \mathrm{Hom}\,(V, W)$ implies $T\,(\lambda v + w) = \lambda T\,(v) + T\,(w)$ for every $v, w \in V$ and every $\lambda \in \mathscr{F}$, where V and W are linear spaces over a field $\mathscr{F}$.

Remark 5.5. If V and W are finite-dimensional linear spaces (like $\mathbb{R}^n$), then $T \in \mathrm{Hom}\,(V, W)$ can be represented by a *matrix*.

Definition 5.6. $V^* := \mathrm{Hom}\,(V, \mathscr{F})$ is the *dual* of the linear space V over the field $\mathscr{F}$.

Definition 5.7. $T^* : W^* \to V^*$, defined by $f \mapsto f \circ T$, is the *adjoint* of $T \in \mathrm{Hom}\,(V, W)$, where V and W are linear spaces.

Remark 5.8. If V and W are finite-dimensional linear spaces and $T \in \mathrm{Hom}\,(V, W)$, then T^* is the *transpose* of T, when T is represented by a matrix, i.e., $T^* = T^{\mathrm{T}}$.

Definition 5.9. $v \odot w := (v_1 w_1, v_2 w_2, \dots, v_n w_n)$, where $v = (v_1, v_2, \dots, v_n)\,, w = (w_1, w_2, \dots, w_n) \in \mathbb{R}^n$, is a *Hadamard* product.

Definition 5.10. $\mathscr{D}$ is the set of functions $\mathbb{R} \to \mathbb{R}$ that are *differentiable almost everywhere* with respect to the Lebesgue measure on $\mathbb{R}$.

Definition 5.11. $f \otimes w : V \to W$, defined by $v \mapsto f\,(v)\,w$, is the *outer product* of f and w, where $f \in V^*$, $w \in W$, and V and W are linear spaces.

Definition 5.12. $\langle \cdot, \cdot \rangle : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$, defined by $((v_1, v_2, \dots, v_n)\,, (w_1, w_2, \dots, w_n)) \mapsto \sum_{i=1}^n v_i w_i$, is the *dot product* on $\mathbb{R}^n$.

Definition 5.13. $\sigma_n : \mathbb{R}^n \to \mathbb{R}^n$, defined by $(x_1, x_2, \dots, x_n) \mapsto (e^{x_1}, e^{x_2}, \dots, e^{x_n}) \,/ \sum_{i=1}^n e^{x_i}$, is the *softmax* function on $\mathbb{R}^n$.

Remark/Definition 5.14. If V is a finite-dimensional linear space over a field $\mathscr{F}$, then V and V^* are isomorphic, i.e., there is a bijection $\phi : V \to V^*$ such that $\phi\,(\lambda v + w) = \lambda \phi\,(v) + \phi\,(w)$ for every $v, w \in V$ and every $\lambda \in \mathscr{F}$. In this case, let $\overline{v} := \phi\,(v)$.