File Name: T-ICML-O_1_I1_introduction

Format: Presenter in Studio

Presenter: Evan Jones

Linear and DNN Models for Image Classification

Evan Jones

Agenda

Introduction

Linear Models

Deep Neural Network Models

DNN Dropout

Learn how to...

Understand how image data is represented as floating point numbers that can be flattened

Compare functions for model confidence in image classification (Softmax)

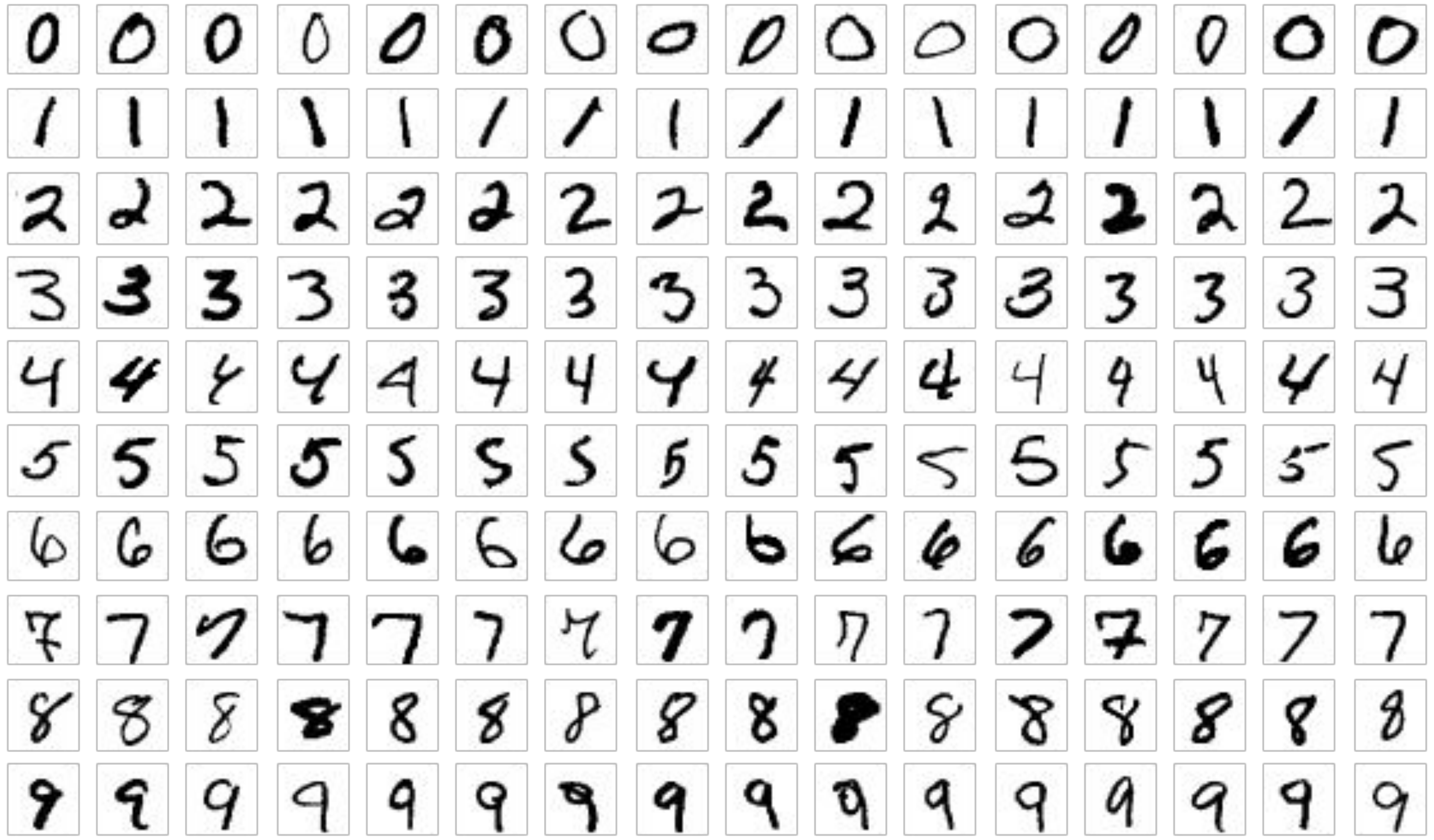
Train and evaluate a Linear model for image classification using TensorFlow

Train and evaluate a Deep Neural Network (DNN) model for image classification using TensorFlow

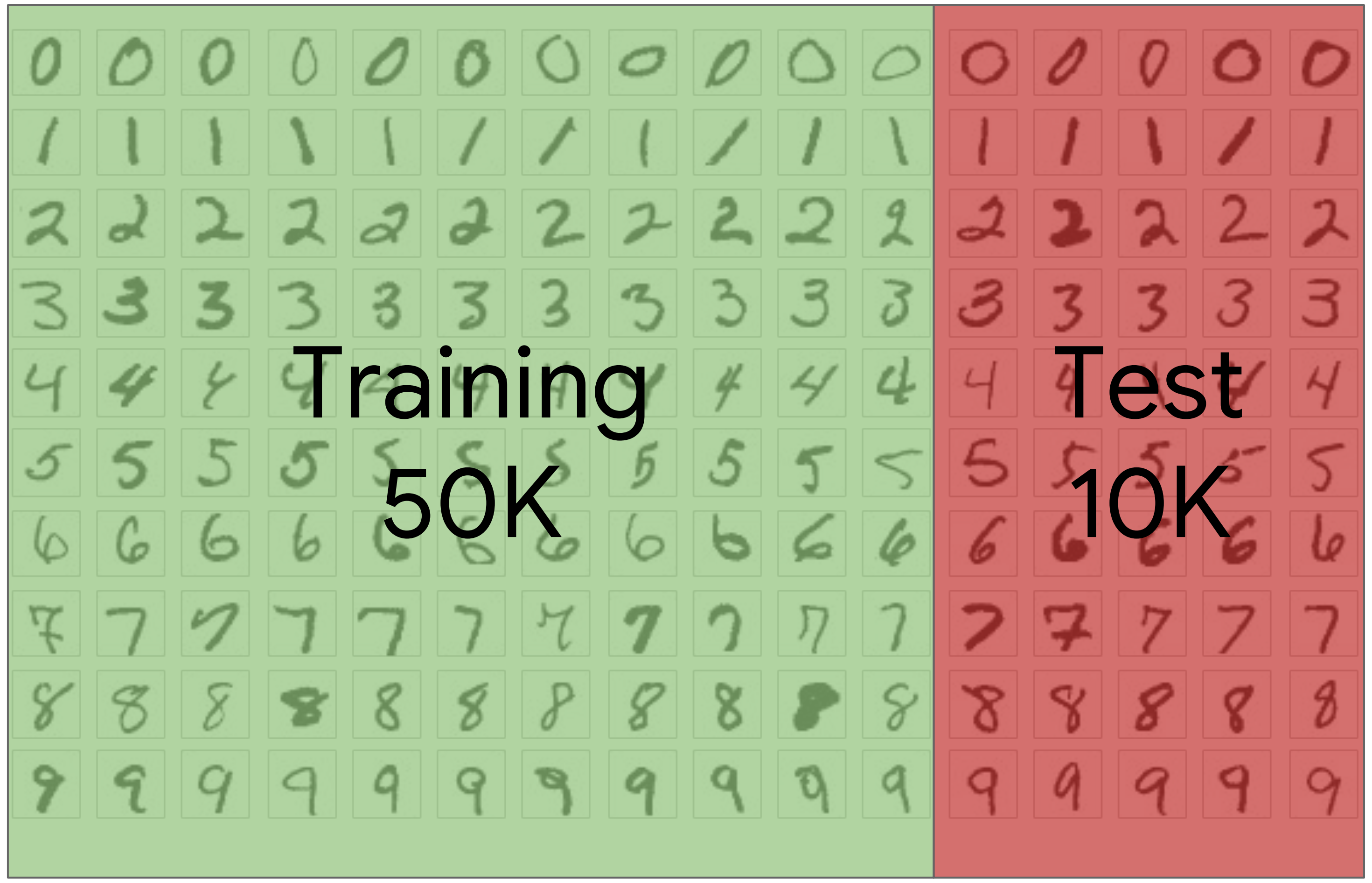
Understand how to apply dropout as a regularization technique for DNNs

Problem: Recognizing Handwritten Digits

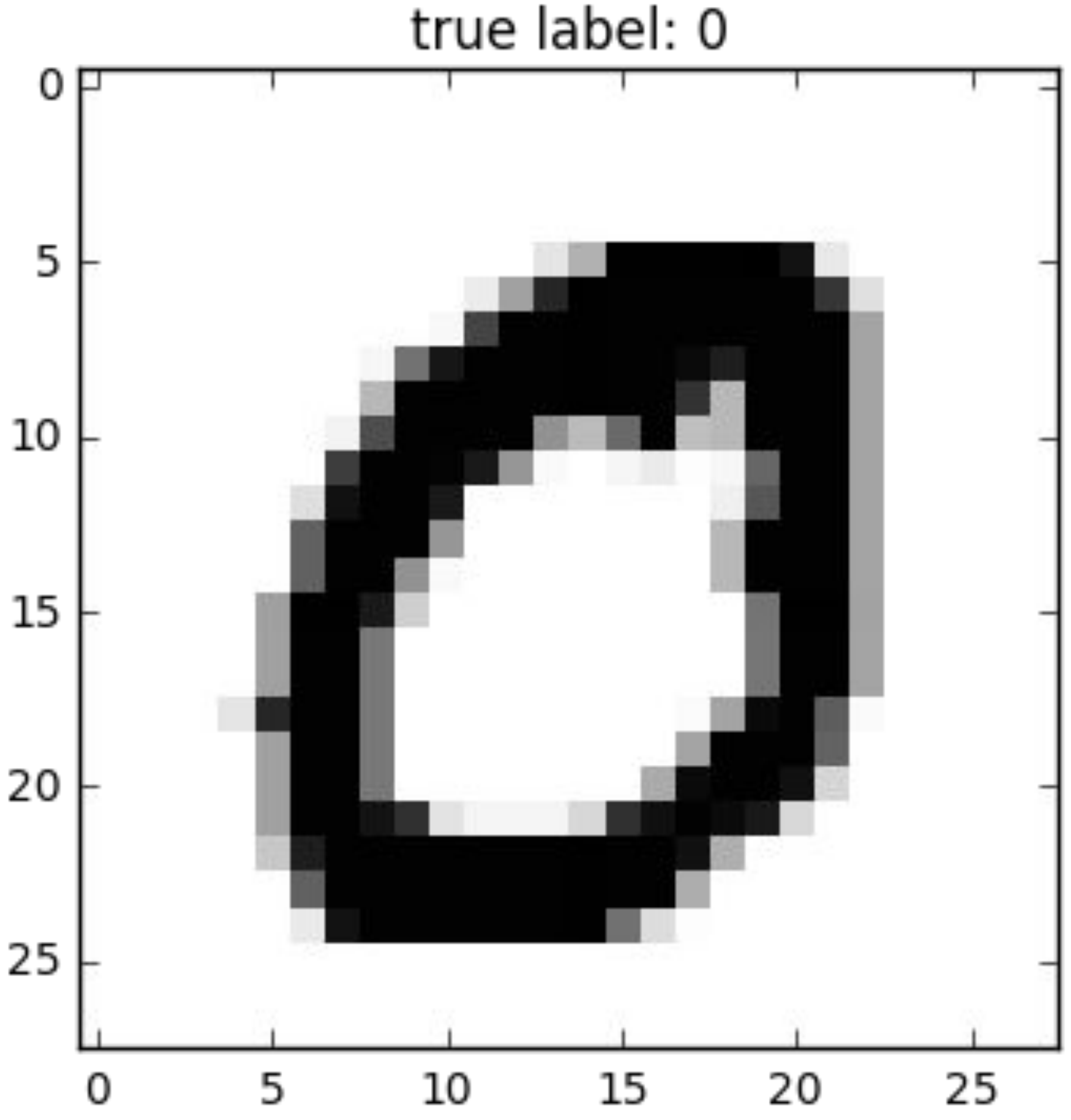
Introducing the MNIST dataset of labeled images



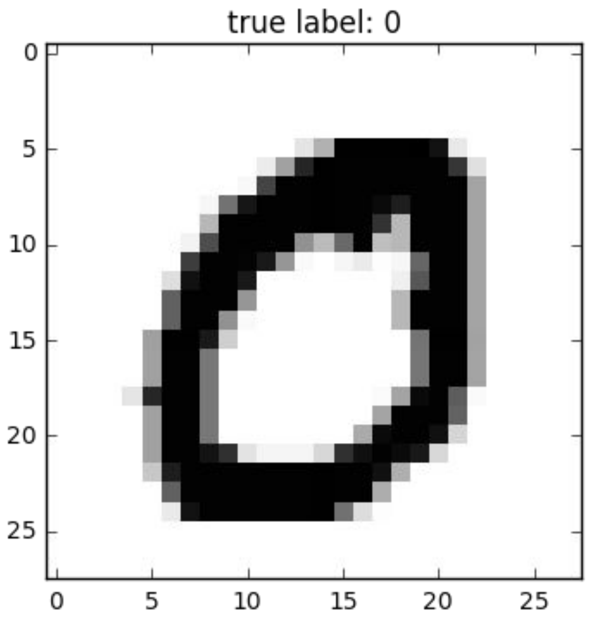
60,000 total images



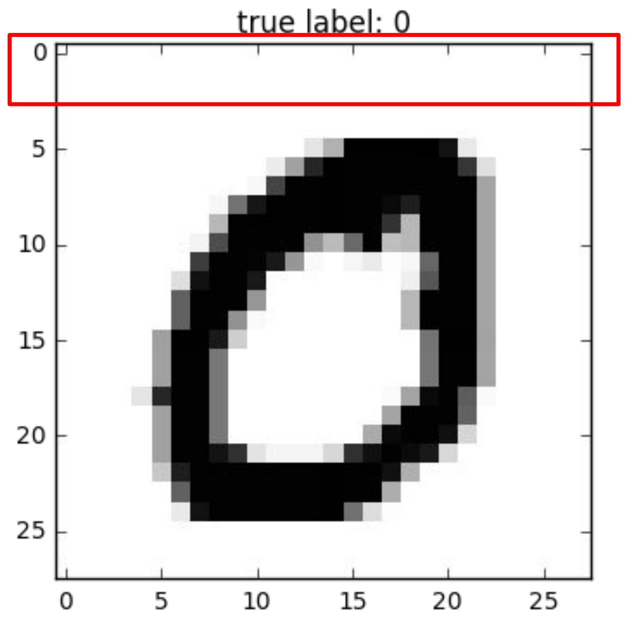
Each greyscale image is 28 x 28 pixels



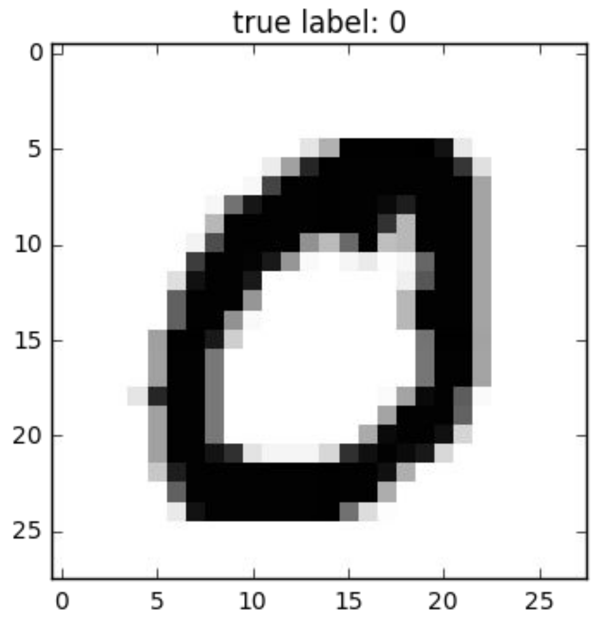
Each greyscale image is 28 x 28 pixels



Unstack the pixels into one long array

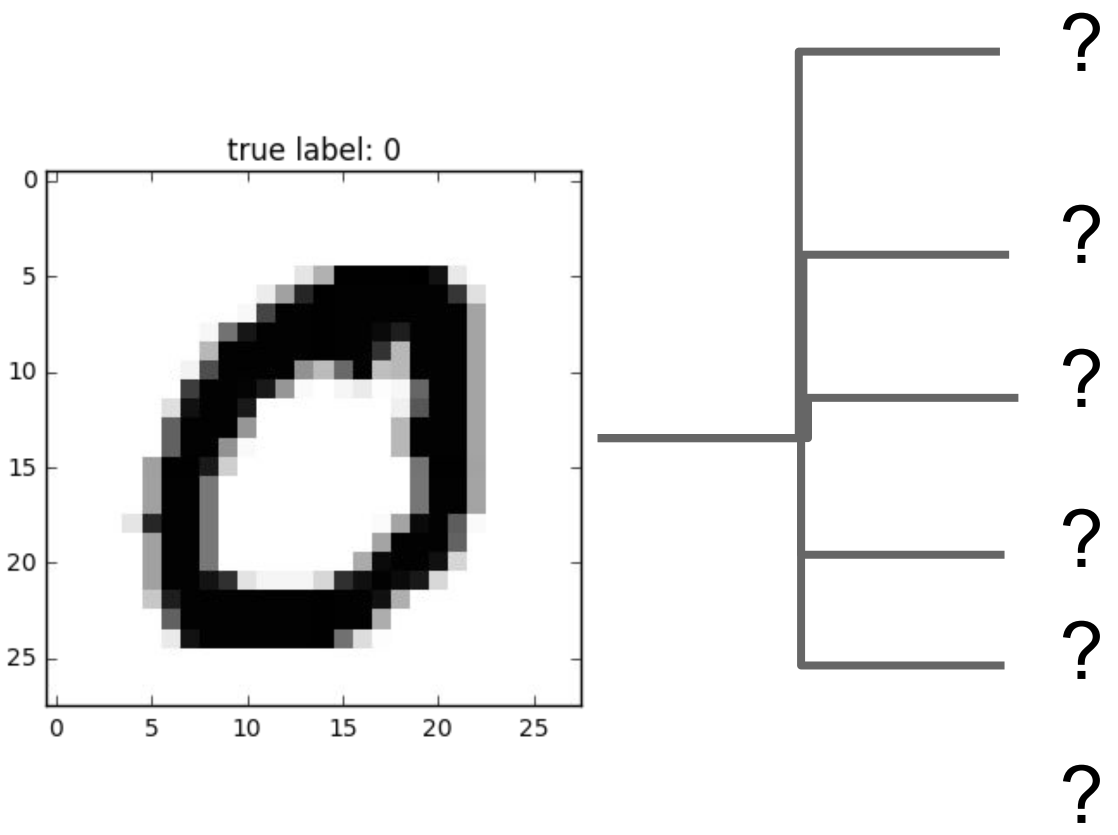


The flattened image is represented as an array

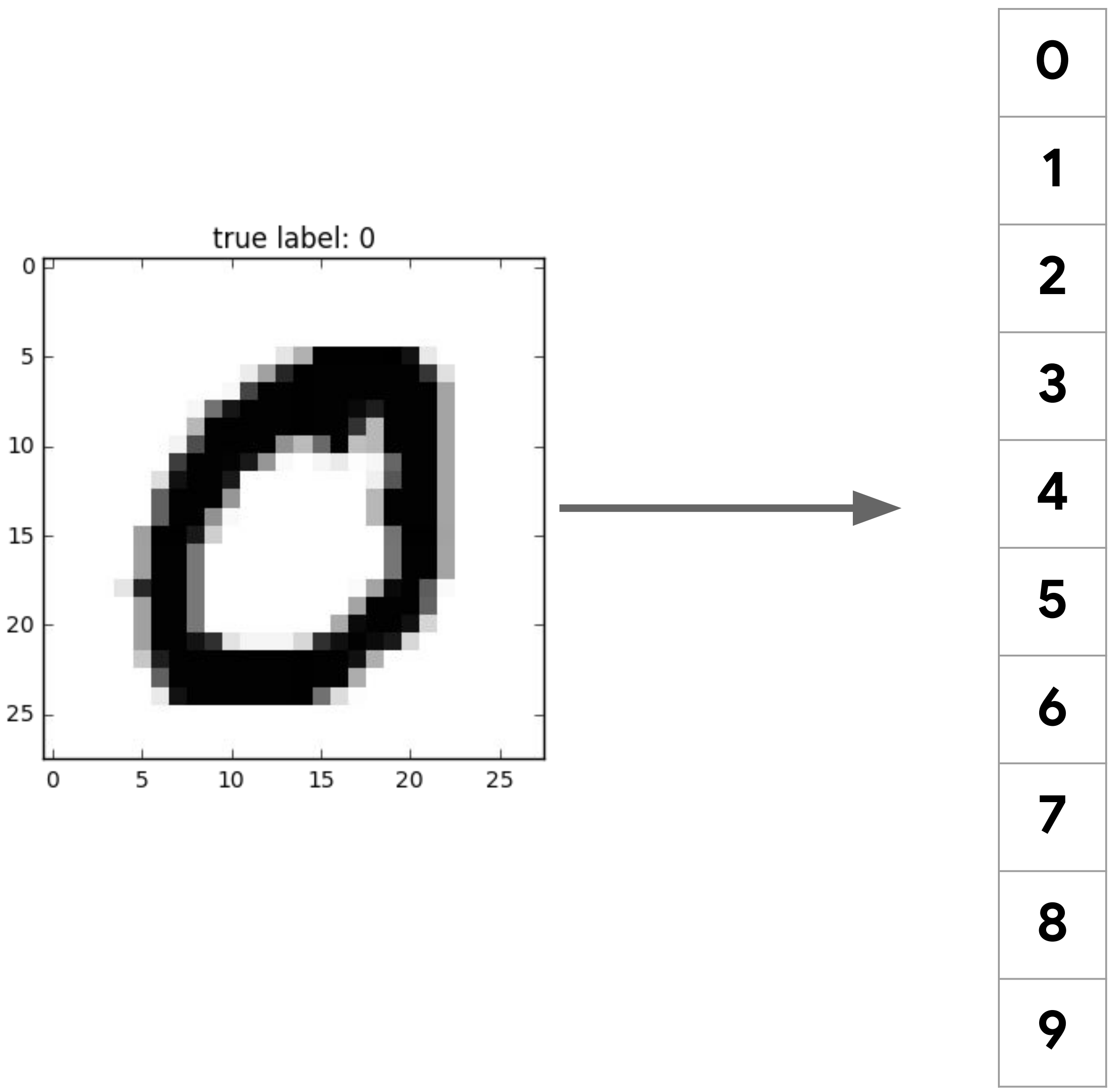


0	0.2	0.4	0.9	0.8	...

How many output classes do we have?

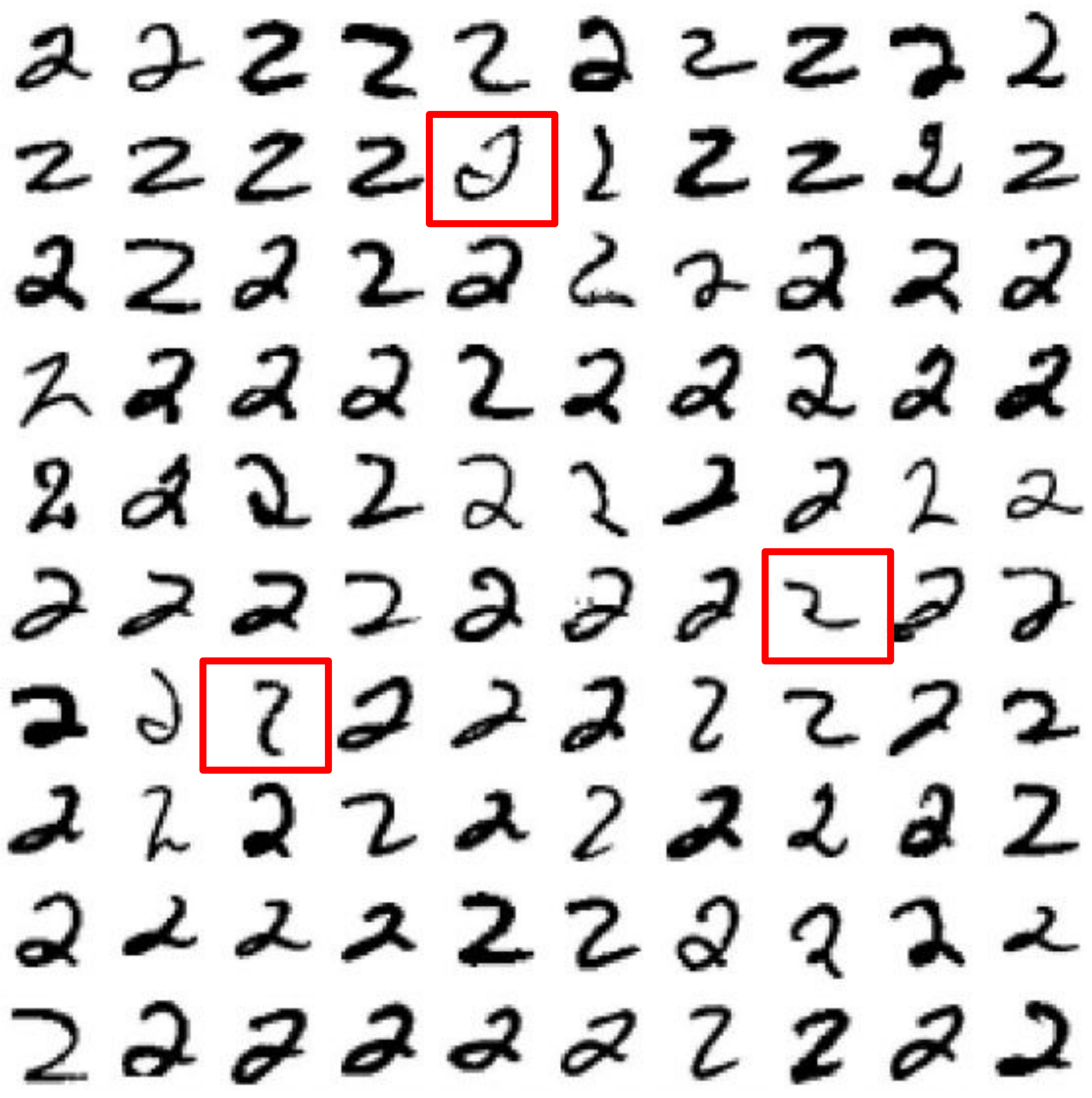


How many output classes do we have?

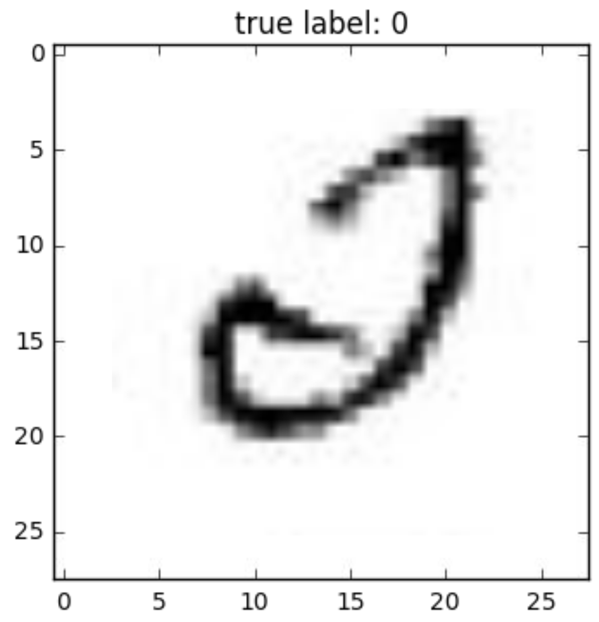


What if the model is unsure?

What if the model is unsure?

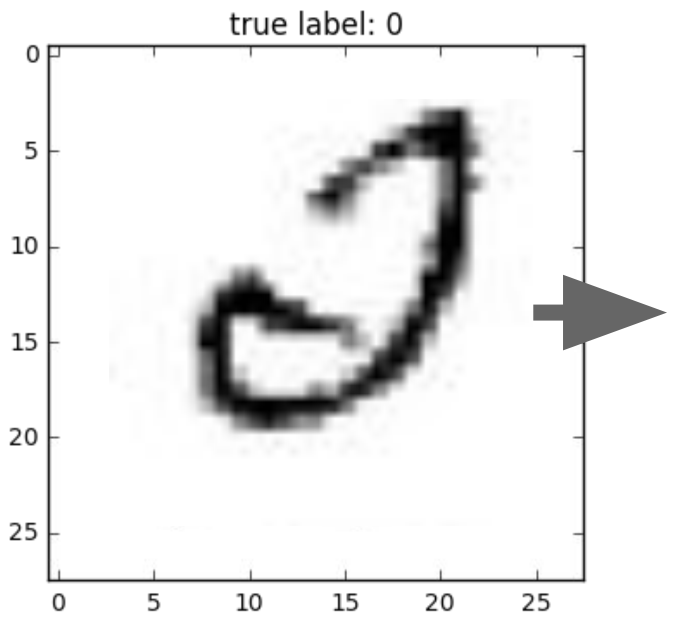


Assessing the model's confidence with a function



0	20%
1	70%
2	
3	
4	
5	
6	
7	
8	10%
9	

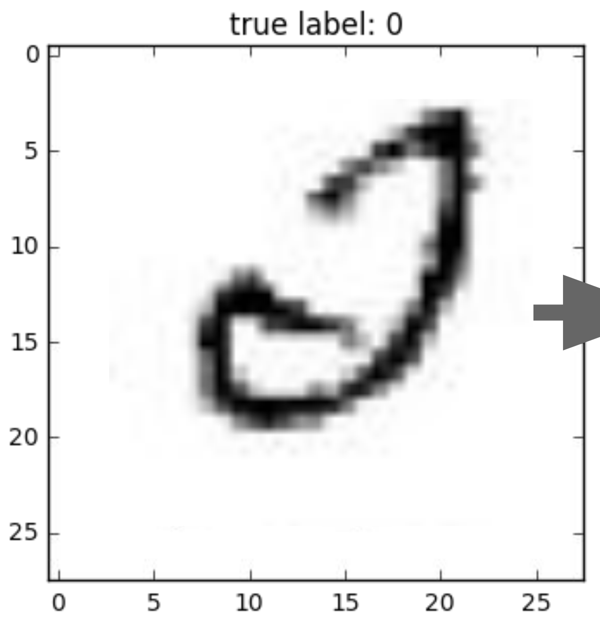
Assessing the model’s confidence with a function

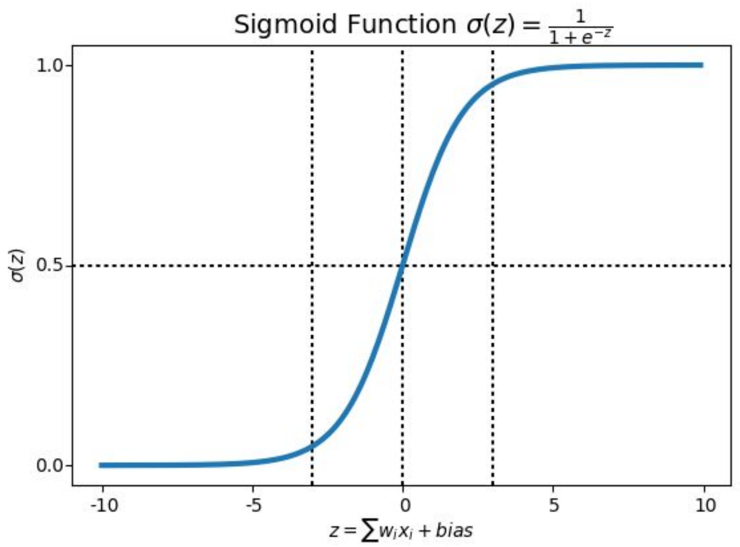


??

Assessing the model's confidence with a function

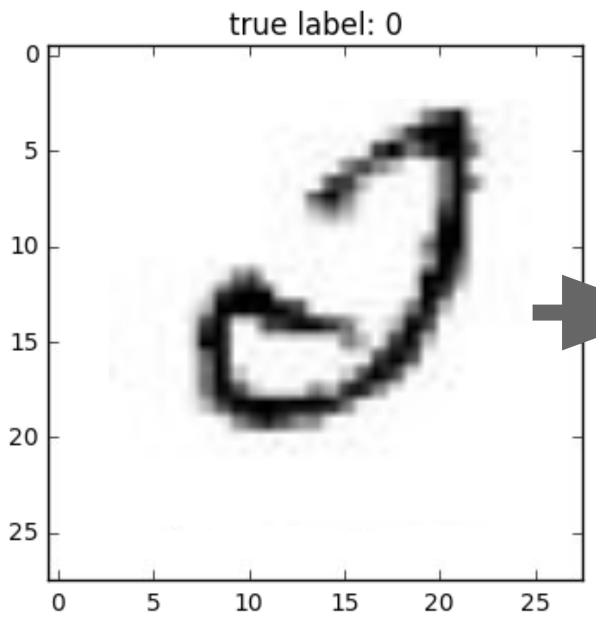
Sigmoid?

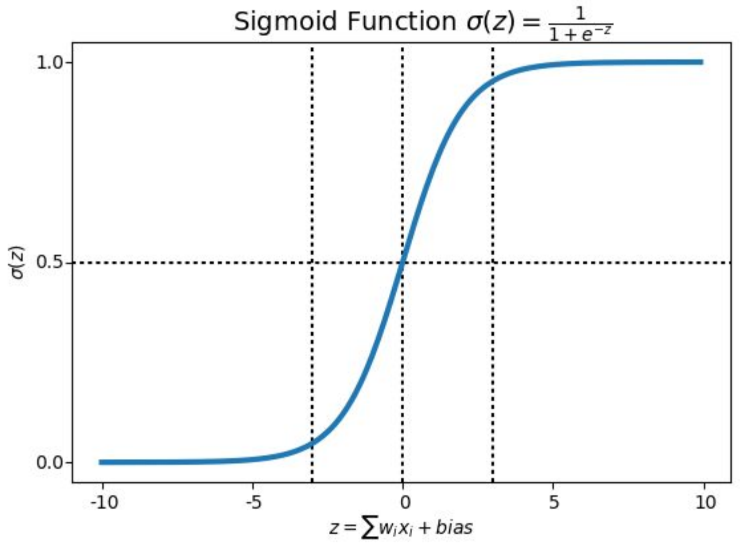




Assessing the model's confidence with a function

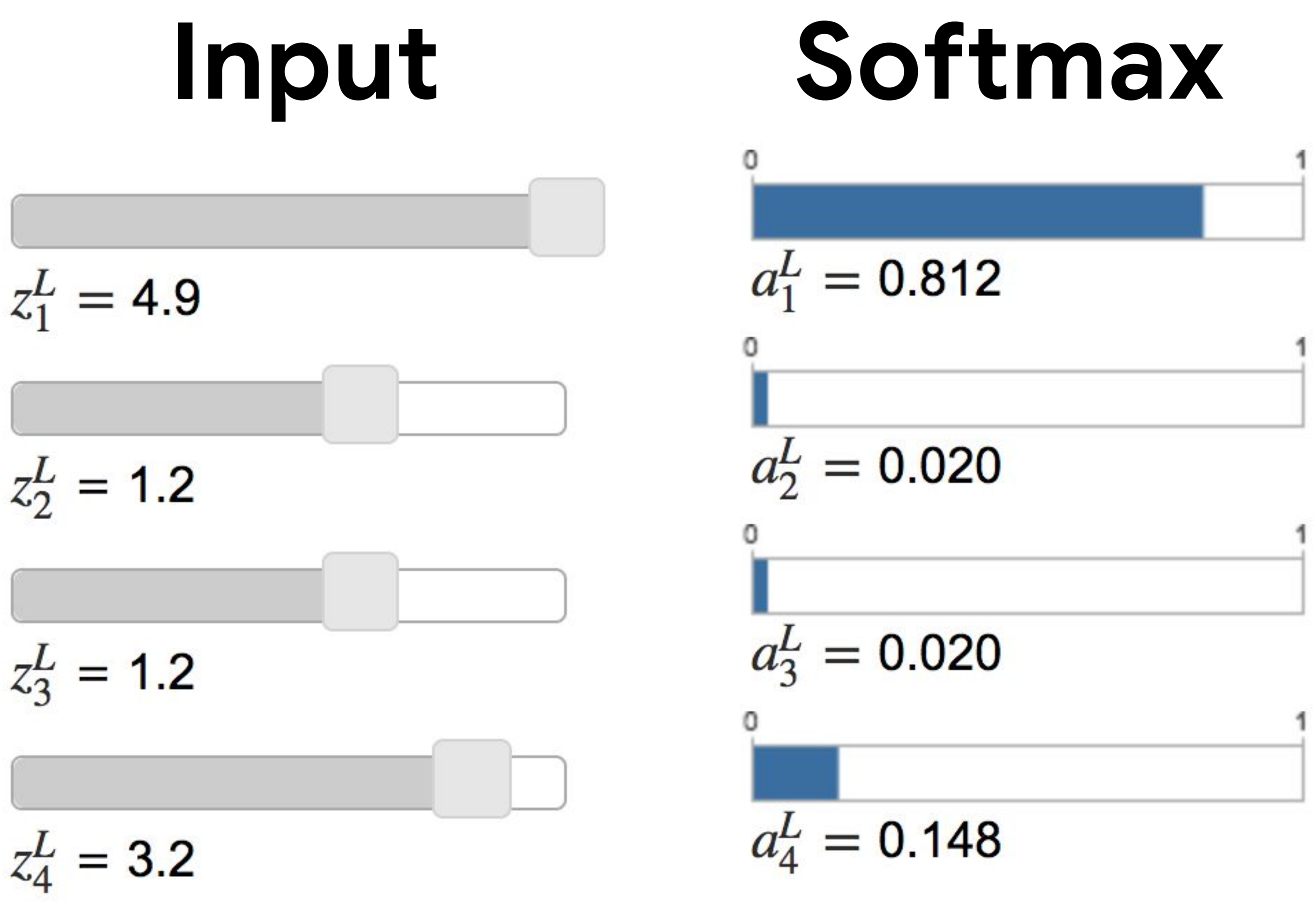
~~Sigmoid?~~ Nope.





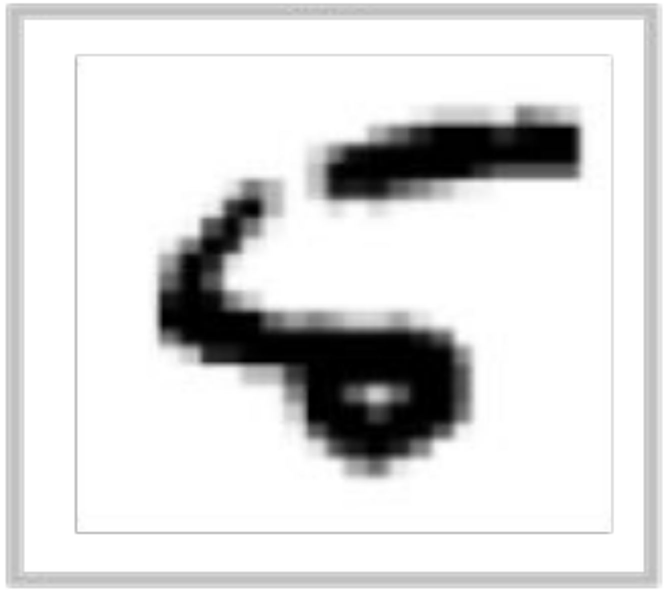
Softmax exponentiating its inputs and then normalizing them

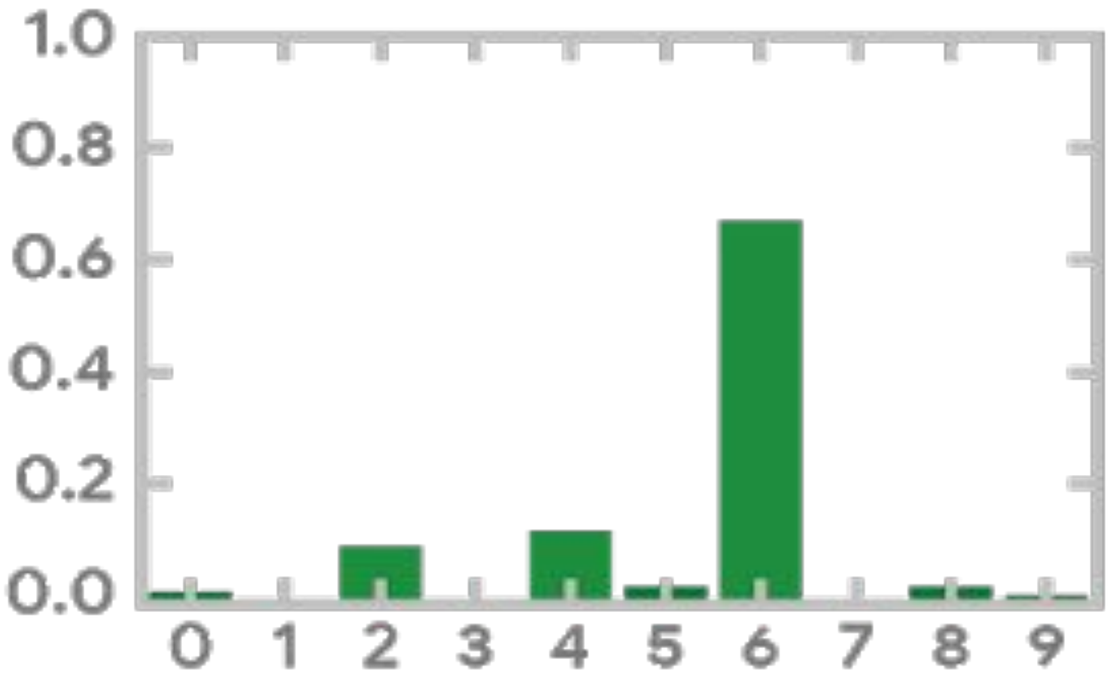
$$\mathrm{softmax}(x)_i = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$



Softmax exponentiating its inputs and then normalizing them

$$\text{softmax}(x)_i = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$

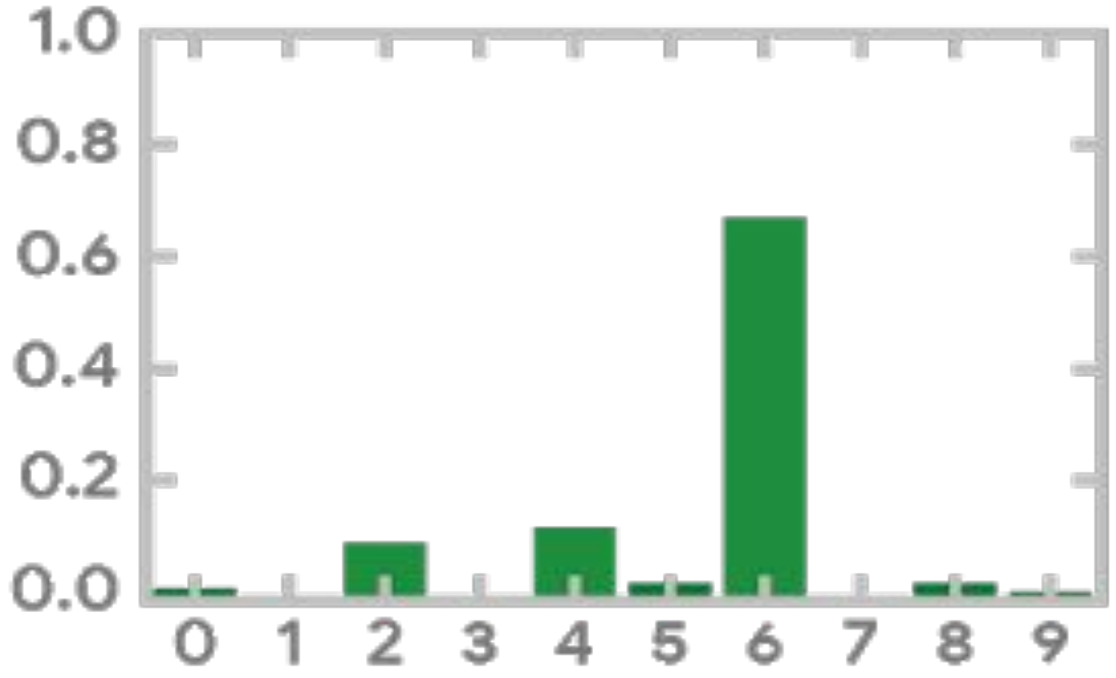




0	2%
1	
2	10%
3	
4	11%
5	3%
6	70%
7	
8	3%
9	1%

Softmax exponentiating its inputs and then normalizing them

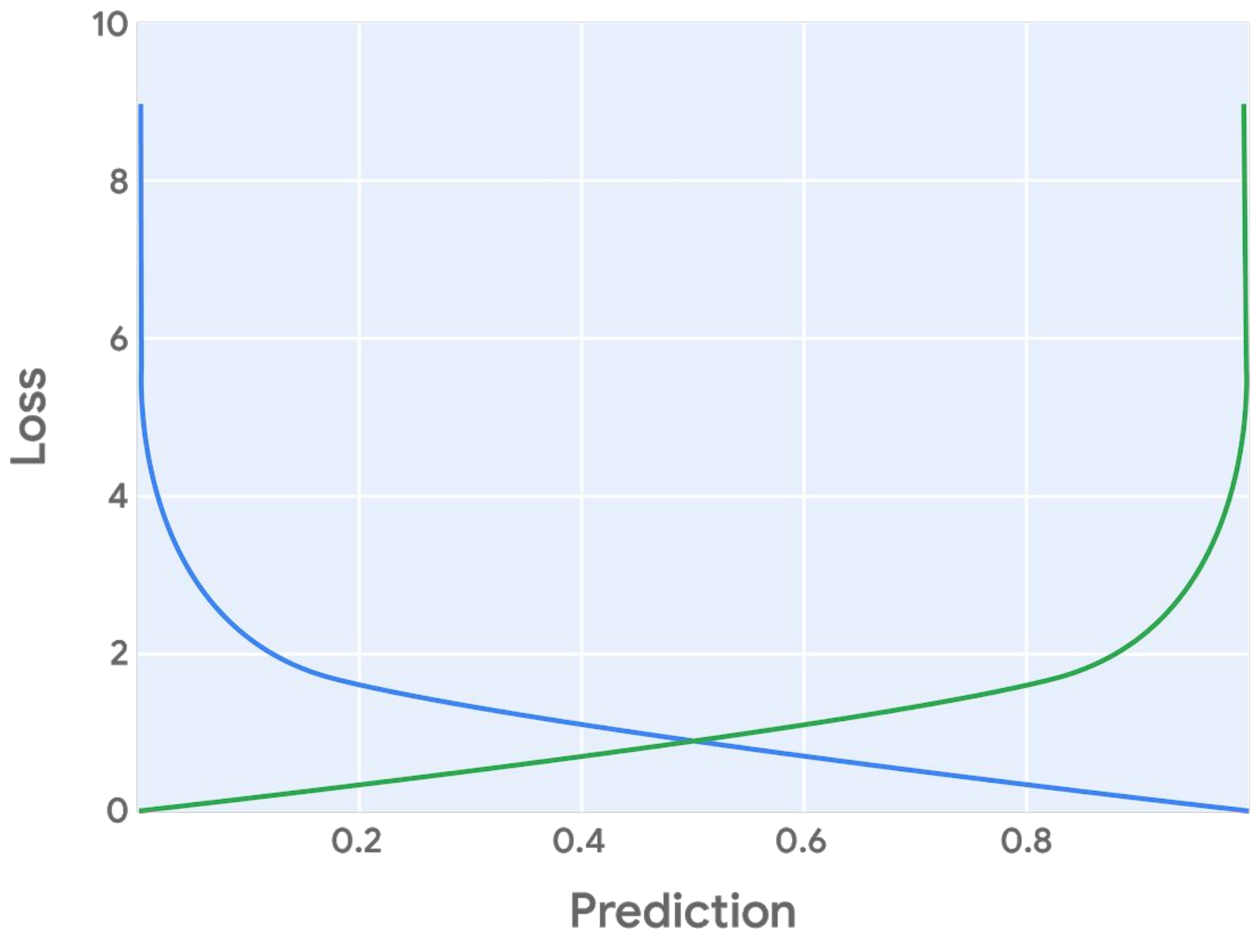
$$\text{softmax}(x)_i = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$



0	2%	
1		
2	10%	
3		
4	11%	
5	3%	
6	70%	X
7		
8	3%	
9	1%	

Training and evaluating our image classification model

Minimize cross-entropy error

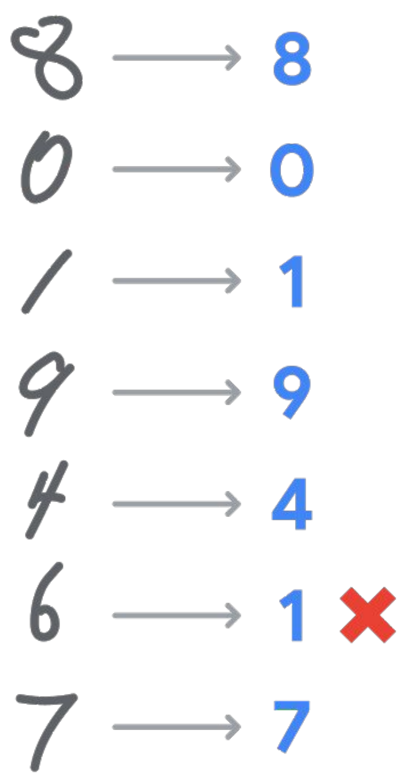


$$\frac{-1}{N} \times \sum_{1}^{N} y_i \times log(\hat{y}_i) + (1 - y_i) \times log(1 - \hat{y}_i)$$

Numerically more stable:
`softmax_cross_entropy_with_logits`

Accuracy for model performance

$$\frac{\text{\# correctly classified images}}{\text{\# number total images}}$$



$$\frac{7}{8} = 88\%$$

File Name: T-ICML-O_1_l2_linear_models

Format: Presenter in Studio

Presenter: Evan Jones

Agenda

Introduction

Linear Models

Deep Neural Network Models

DNN Dropout

Training and evaluating our image classification model

Learning

```
def my_model_fn(features, labels, mode):
    predictions, num_classes = my_model(features)
    loss = ...
    train_op = ...
    return tf.estimator.EstimatorSpec(
        mode=mode,
        predictions=predictions,
        loss=loss,
        train_op=train_op)

estimator = tf.estimator.Estimator(model_fn = my_model_fn, params)

train_input_fn = tf.estimator.inputs.numpy_input_fn(params)
eval_input_fn = tf.estimator.inputs.numpy_input_fn(params)
train_spec = tf.estimator.TrainSpec(input_fn = train_input_fn, params)
eval_spec = tf.estimator.EvalSpec(input_fn = eval_input_fn, params)
tf.estimator.train_and_evaluate(estimator, train_spec, eval_spec)`
```

Training and evaluating our image classification model

```
def my_model_fn(features, labels, mode):
    predictions, num_classes = my_model(features)
    loss = ...
    train_op = ...
    return tf.estimator.EstimatorSpec(
        mode=mode,
        predictions=predictions,
        loss=loss,
        train_op=train_op)

estimator = tf.estimator.Estimator(model_fn = my_model_fn, params)

train_input_fn = tf.estimator.inputs.numpy_input_fn(params)
eval_input_fn = tf.estimator.inputs.numpy_input_fn(params)
train_spec = tf.estimator.TrainSpec(input_fn = train_input_fn, params)
eval_spec = tf.estimator.EvalSpec(input_fn = eval_input_fn, params)
tf.estimator.train_and_evaluate(estimator, train_spec, eval_spec)`
```

Training and evaluating our image classification model

```
def my_model_fn(features, labels, mode):
    predictions, num_classes = my_model(features)
    loss = ...
    train_op = ...
    return tf.estimator.EstimatorSpec(
        mode=mode,
        predictions=predictions,
        loss=loss,
        train_op=train_op)

estimator = tf.estimator.Estimator(model_fn = my_model_fn, params)

train_input_fn = tf.estimator.inputs.numpy_input_fn(params)
eval_input_fn = tf.estimator.inputs.numpy_input_fn(params)
train_spec = tf.estimator.TrainSpec(input_fn = train_input_fn, params)
eval_spec = tf.estimator.EvalSpec(input_fn = eval_input_fn, params)
tf.estimator.train_and_evaluate(estimator, train_spec, eval_spec)
```

Training and evaluating our image classification model

```
def my_model_fn(features, labels, mode):
    predictions, num_classes = my_model(features)
    loss = ...
    train_op = ...
    return tf.estimator.EstimatorSpec(
        mode=mode,
        predictions=predictions,
        loss=loss,
        train_op=train_op)

estimator = tf.estimator.Estimator(model_fn = my_model_fn, params)

train_input_fn = tf.estimator.inputs.numpy_input_fn(params)
eval_input_fn = tf.estimator.inputs.numpy_input_fn(params)
train_spec = tf.estimator.TrainSpec(input_fn = train_input_fn, params)
eval_spec = tf.estimator.EvalSpec(input_fn = eval_input_fn, params)
tf.estimator.train_and_evaluate(estimator, train_spec, eval_spec)
```

Making predictions with our image classification model

```
HEIGHT=28
WIDTH=28
NCLASSES=10

def linear_model(img):
    """Uses a linear model to compute a vector representing relative confidence
    that img belongs to one of NCLASSES classes.

      Args:
        img [batchsize, HEIGHT, WIDTH]: A tensor of floats representing a batch
        of images.

      Returns:
          logits: the output of the model
         NCLASSES: the number of classes
    """
    X = tf.reshape(img, [-1, HEIGHT*WIDTH]) # [-1, HEIGHT * WIDTH]
    W = tf.Variable(tf.zeros([HEIGHT*WIDTH, NCLASSES]))  # [HEIGHT * WIDTH, CLASSES]
    b = tf.Variable(tf.zeros([NCLASSES])) # [NCLASSES]
    ylogits = tf.matmul(X, W) + b # [-1, NCLASSES]
    return ylogits, NCLASSES
```

Making predictions with our image classification model

```
HEIGHT=28
WIDTH=28
NCLASSES=10

def linear_model(img):
    """Uses a linear model to compute a vector representing relative confidence
    that img belongs to one of NCLASSES classes.

      Args:
        img [batchsize, HEIGHT, WIDTH]: A tensor of floats representing a batch
        of images.

      Returns:
          logits: the output of the model
         NCLASSES: the number of classes
    """
    X = tf.reshape(img, [-1, HEIGHT*WIDTH]) # [-1, HEIGHT * WIDTH]
    W = tf.Variable(tf.zeros([HEIGHT*WIDTH, NCLASSES]))  # [HEIGHT * WIDTH, CLASSES]
    b = tf.Variable(tf.zeros([NCLASSES])) # [NCLASSES]
    ylogits = tf.matmul(X, W) + b # [-1, NCLASSES]
    return ylogits, NCLASSES
```

Making predictions with our image classification model

```
HEIGHT=28
WIDTH=28
NCLASSES=10

def linear_model(img):
    """Uses a linear model to compute a vector representing relative confidence
    that img belongs to one of NCLASSES classes.

      Args:
        img [batchsize, HEIGHT, WIDTH]: A tensor of floats representing a batch
        of images.

      Returns:
          logits: the output of the model
         NCLASSES: the number of classes
    """
    X = tf.reshape(img, [-1, HEIGHT*WIDTH]) # [-1, HEIGHT * WIDTH]
    W = tf.Variable(tf.zeros([HEIGHT*WIDTH, NCLASSES]))  # [HEIGHT * WIDTH, CLASSES]
    b = tf.Variable(tf.zeros([NCLASSES])) # [NCLASSES]
    ylogits = tf.matmul(X, W) + b # [-1, NCLASSES]
    return ylogits, NCLASSES
```

Making predictions with our image classification model

```
HEIGHT=28
WIDTH=28
NCLASSES=10

def linear_model(img):
    """Uses a linear model to compute a vector representing relative confidence
    that img belongs to one of NCLASSES classes.

      Args:
        img [batchsize, HEIGHT, WIDTH]: A tensor of floats representing a batch
        of images.

      Returns:
          logits: the output of the model
         NCLASSES: the number of classes
    """
    X = tf.reshape(img, [-1, HEIGHT*WIDTH]) # [-1, HEIGHT * WIDTH]
    W = tf.Variable(tf.zeros([HEIGHT*WIDTH, NCLASSES]))  # [HEIGHT * WIDTH, CLASSES]
    b = tf.Variable(tf.zeros([NCLASSES])) # [NCLASSES]
    ylogits = tf.matmul(X, W) + b # [-1, NCLASSES]
    return ylogits, NCLASSES
```

Making predictions with our image classification model

```
HEIGHT=28
WIDTH=28
NCLASSES=10

def linear_model(img):
    """Uses a linear model to compute a vector representing relative confidence
    that img belongs to one of NCLASSES classes.

      Args:
        img [batchsize, HEIGHT, WIDTH]: A tensor of floats representing a batch
        of images.

      Returns:
          logits: the output of the model
          NCLASSES: the number of classes
    """
    X = tf.reshape(img, [-1, HEIGHT*WIDTH]) # [-1, HEIGHT * WIDTH]
    W = tf.Variable(tf.zeros([HEIGHT*WIDTH, NCLASSES]))  # [HEIGHT * WIDTH, CLASSES]
    b = tf.Variable(tf.zeros([NCLASSES])) # [NCLASSES]
    ylogits = tf.matmul(X, W) + b # [-1, NCLASSES]
    return ylogits, NCLASSES
```

Making predictions with our image classification model

```
HEIGHT=28
WIDTH=28
NCLASSES=10

def linear_model(img):
    """Uses a linear model to compute a vector representing relative confidence
    that img belongs to one of NCLASSES classes.

      Args:
        img [batchsize, HEIGHT, WIDTH]: A tensor of floats representing a batch
        of images.

      Returns:
          logits: the output of the model
         NCLASSES: the number of classes
    """
    X = tf.reshape(img, [-1, HEIGHT*WIDTH]) # [-1, HEIGHT * WIDTH]
    W = tf.Variable(tf.zeros([HEIGHT*WIDTH, NCLASSES]))  # [HEIGHT * WIDTH, CLASSES]
    b = tf.Variable(tf.zeros([NCLASSES])) # [NCLASSES]
    ylogits = tf.matmul(X, W) + b # [-1, NCLASSES]
    return ylogits, NCLASSES
```

Training and evaluating our image classification model

```
def my_model_fn(features, labels, mode):
    logits, num_classes = linear_model(features['image'])
    probabilities = tf.nn.softmax(logits)
    loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(
                logits=logits, labels=labels))
    train_op = ...
    return tf.estimator.EstimatorSpec(
        mode=mode,
        predictions=probabilities,
        loss=loss,
        train_op=train_op)
```

Training and evaluating our image classification model

```
def my_model_fn(features, labels, mode):
    logits, num_classes = linear_model(features['image'])
    probabilities = tf.nn.softmax(logits)
    loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(
                logits=logits, labels=labels))
    train_op = ...
    return tf.estimator.EstimatorSpec(
        mode=mode,
        predictions=probabilities,
        loss=loss,
        train_op=train_op)
```

Training and evaluating our image classification model

```
def my_model_fn(features, labels, mode):
    logits, num_classes = linear_model(features['image'])
    probabilities = tf.nn.softmax(logits)
    loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(
                logits=logits, labels=labels))
    train_op = ...
    return tf.estimator.EstimatorSpec(
        mode=mode,
        predictions=probabilities,
        loss=loss,
        train_op=train_op)
```

Training and evaluating our image classification model

```
def my_model_fn(features, labels, mode):
    logits, num_classes = linear_model(features['image'])
    probabilities = tf.nn.softmax(logits)
    loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(
                logits=logits, labels=labels))
    train_op = ...
    return tf.estimator.EstimatorSpec(
        mode=mode,
        predictions=probabilities,
        loss=loss,
        train_op=train_op)
```

Training and evaluating our image classification model

```
def my_model_fn(features, labels, mode):
    logits, num_classes = linear_model(features['image'])
    probabilities = tf.nn.softmax(logits)
    loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(
               logits=logits, labels=labels))
    train_op = ...
    return tf.estimator.EstimatorSpec(
        mode=mode,
        predictions=probabilities,
        loss=loss,
        train_op=train_op)
```

Training and evaluating our image classification model

```
def my_model_fn(features, labels, mode):
    logits, num_classes = linear_model(features['image'])
    probabilities = tf.nn.softmax(logits)
    loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(
                logits=logits, labels=labels))
    train_op = ...
    return tf.estimator.EstimatorSpec(
        mode=mode,
        predictions=probabilities,
        loss=loss,
        train_op=train_op)
```

Training and evaluating our image classification model

```
def my_model_fn(features, labels, mode):
    logits, num_classes = linear_model(features['image'])
    probabilities = tf.nn.softmax(logits)
    loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(
               logits=logits, labels=labels))
    train_op = ...
    return tf.estimator.EstimatorSpec(
        mode=mode,
        predictions=probabilities,
        loss=loss,
        train_op=train_op)

estimator = tf.estimator.Estimator(model_fn = my_model_fn, params)
```

Training and evaluating our image classification model

```
def my_model_fn(features, labels, mode):
    logits, num_classes = linear_model(features['image'])
    probabilities = tf.nn.softmax(logits)
    loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(
                logits=logits, labels=labels))
    train_op = ...
    return tf.estimator.EstimatorSpec(
        mode=mode,
        predictions=probabilities,
        loss=loss,
        train_op=train_op)

estimator = tf.estimator.Estimator(model_fn = my_model_fn, params)
```

Training and evaluating our image classification model

```
def my_model_fn(features, labels, mode):
    logits, num_classes = linear_model(features[‘image’])
    probabilities = tf.nn.softmax(logits)
    loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(
                logits=logits, labels=labels))
    train_op = ...
    return tf.estimator.EstimatorSpec(
        mode=mode,
        predictions=probabilities,
        loss=loss,
        train_op=train_op)

estimator = tf.estimator.Estimator(model_fn = my_model_fn, params)
```

Training and evaluating our image classification model

```
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets('mnist/data', one_hot=True, reshape=False)

train_input_fn = tf.estimator.inputs.numpy_input_fn(
   x={'image':mnist.train.images},
   y=mnist.train.labels,
   batch_size=100,
   num_epochs=None,
   shuffle=True,
   queue_capacity=5000
 )
```

Training and evaluating our image classification model

```
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets('mnist/data', one_hot=True, reshape=False)

train_input_fn = tf.estimator.inputs.numpy_input_fn(
   x={'image':mnist.train.images},
   y=mnist.train.labels,
   batch_size=100,
   num_epochs=None,
   shuffle=True,
   queue_capacity=5000
 )
```

Training and evaluating our image classification model

```
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets('mnist/data', one_hot=True, reshape=False)

train_input_fn = tf.estimator.inputs.numpy_input_fn(
   x={'image':mnist.train.images},
   y=mnist.train.labels,
   batch_size=100,
   num_epochs=None,
   shuffle=True,
   queue_capacity=5000
 )
```

Training and evaluating our image classification model

```
train_spec = tf.estimator.TrainSpec(input_fn = train_input_fn,
                                    max_steps = hparams['train_steps'])

exporter = tf.estimator.LatestExporter('Servo', model.serving_input_fn)

eval_spec = tf.estimator.EvalSpec(input_fn = eval_input_fn,
                                  steps = None,
                                  exporters = exporter,
                                  throttle_secs = EVAL_INTERVAL)
```

Training and evaluating our image classification model

```
train_spec = tf.estimator.TrainSpec(input_fn = train_input_fn,
                                    max_steps = hparams['train_steps'])

exporter = tf.estimator.LatestExporter('Servo', model.serving_input_fn)

eval_spec = tf.estimator.EvalSpec(input_fn = eval_input_fn,
                                  steps = None,
                                  exporters = exporter,
                                  throttle_secs = EVAL_INTERVAL)
```

Training and evaluating our image classification model

```
train_spec = tf.estimator.TrainSpec(input_fn = train_input_fn,
                                    max_steps = hparams['train_steps'])

exporter = tf.estimator.LatestExporter('Servo', model.serving_input_fn)

eval_spec = tf.estimator.EvalSpec(input_fn = eval_input_fn,
                                  steps = None,
                                  exporters = exporter,
                                  throttle_secs = EVAL_INTERVAL)
```

Training and evaluating our image classification model

```
train_spec = tf.estimator.TrainSpec(input_fn = train_input_fn,
                                   max_steps = hparams['train_steps'])

exporter = tf.estimator.LatestExporter('Servo', model.serving_input_fn)

eval_spec = tf.estimator.EvalSpec(input_fn = eval_input_fn,
                                 steps = None,
                                 exporters = exporter,
                                 throttle_secs = EVAL_INTERVAL)
```

Training and evaluating our image classification model

```
train_spec = tf.estimator.TrainSpec(input_fn = train_input_fn,
                                    max_steps = hparams['train_steps'])

exporter = tf.estimator.LatestExporter('Servo', model.serving_input_fn)

eval_spec = tf.estimator.EvalSpec(input_fn = eval_input_fn,
                                  steps = None,
                                  exporters = exporter,
                                  throttle_secs = EVAL_INTERVAL)
```

Training and evaluating our image classification model

```
train_spec = tf.estimator.TrainSpec(input_fn = train_input_fn,
                                   max_steps = hparams['train_steps'])

exporter = tf.estimator.LatestExporter('Servo', model.serving_input_fn)

eval_spec = tf.estimator.EvalSpec(input_fn = eval_input_fn,
                                 steps = None,
                                 exporters = exporter,
                                 throttle_secs = EVAL_INTERVAL)
```

Training and evaluating our image classification model

```
train_spec = tf.estimator.TrainSpec(input_fn = train_input_fn,
                                   max_steps = hparams['train_steps'])

exporter = tf.estimator.LatestExporter('Servo', model.serving_input_fn)

eval_spec = tf.estimator.EvalSpec(input_fn = eval_input_fn,
                                 steps = None,
                                 exporters = exporter,
                                 throttle_secs = EVAL_INTERVAL)
```

Training and evaluating our image classification model

```
train_spec = tf.estimator.TrainSpec(input_fn = train_input_fn,
                                    max_steps = hparams['train_steps'])

exporter = tf.estimator.LatestExporter('Servo', model.serving_input_fn)

eval_spec = tf.estimator.EvalSpec(input_fn = eval_input_fn,
                                  steps = None,
                                  exporters = exporter,
                                  throttle_secs = EVAL_INTERVAL)

tf.estimator.train_and_evaluate(estimator, train_spec, eval_spec)
```

Video Name: T-ICML-O_1_l3_lab_intro:_linear_models

Format: Studio with Presenter

Presenter: Evan Jones

Lab

Image Classification with a Linear Model

Evan Jones

Video Name: T-ICML-O_1_l4_lab_solution:_linear_models

Format: Studio with Presenter

Presenter: Evan Jones

File Name: T-ICML-O_1_l5_dnn_models

Format: Presenter in Studio

Presenter: Evan Jones

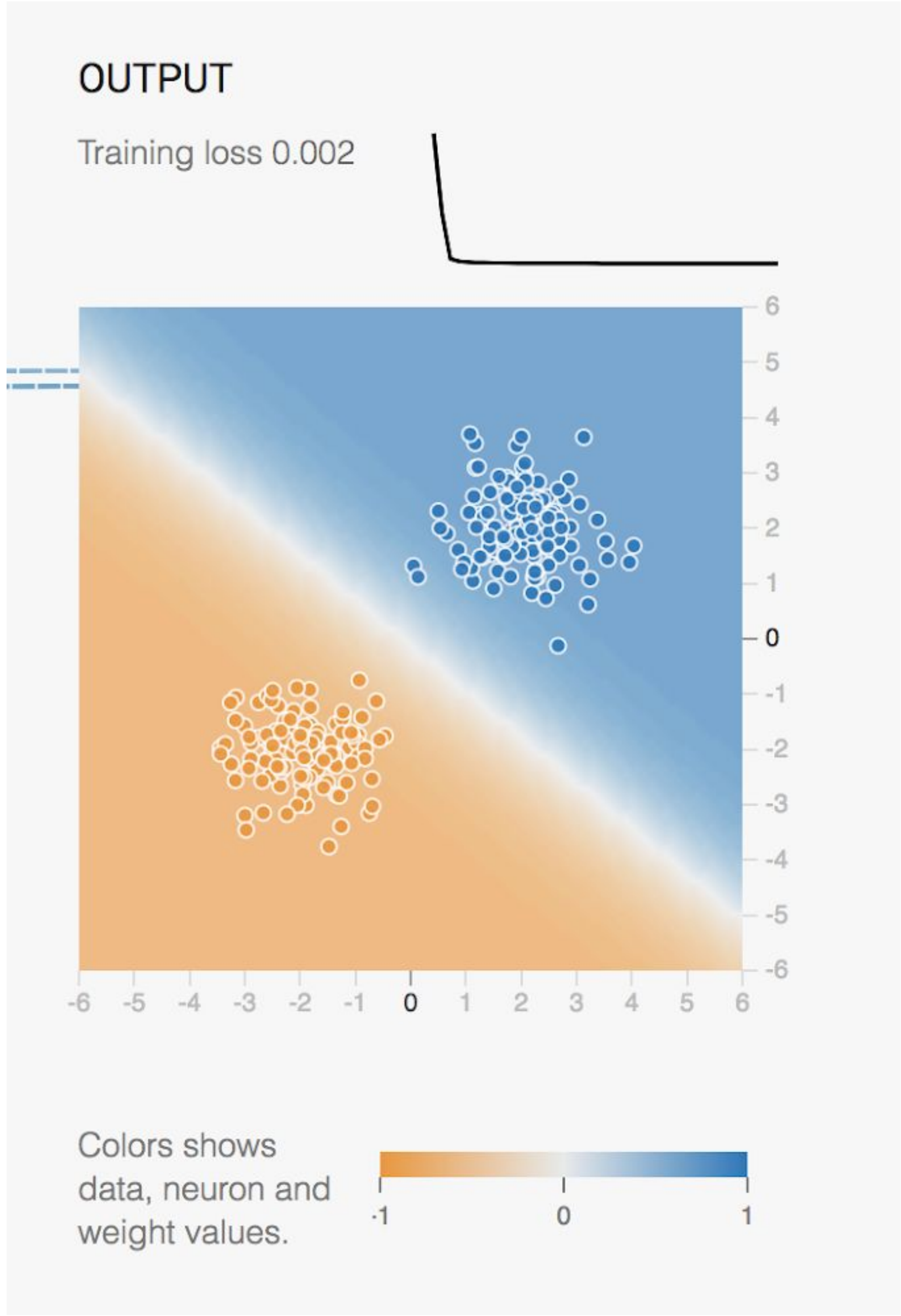
Agenda

Introduction

Linear Models

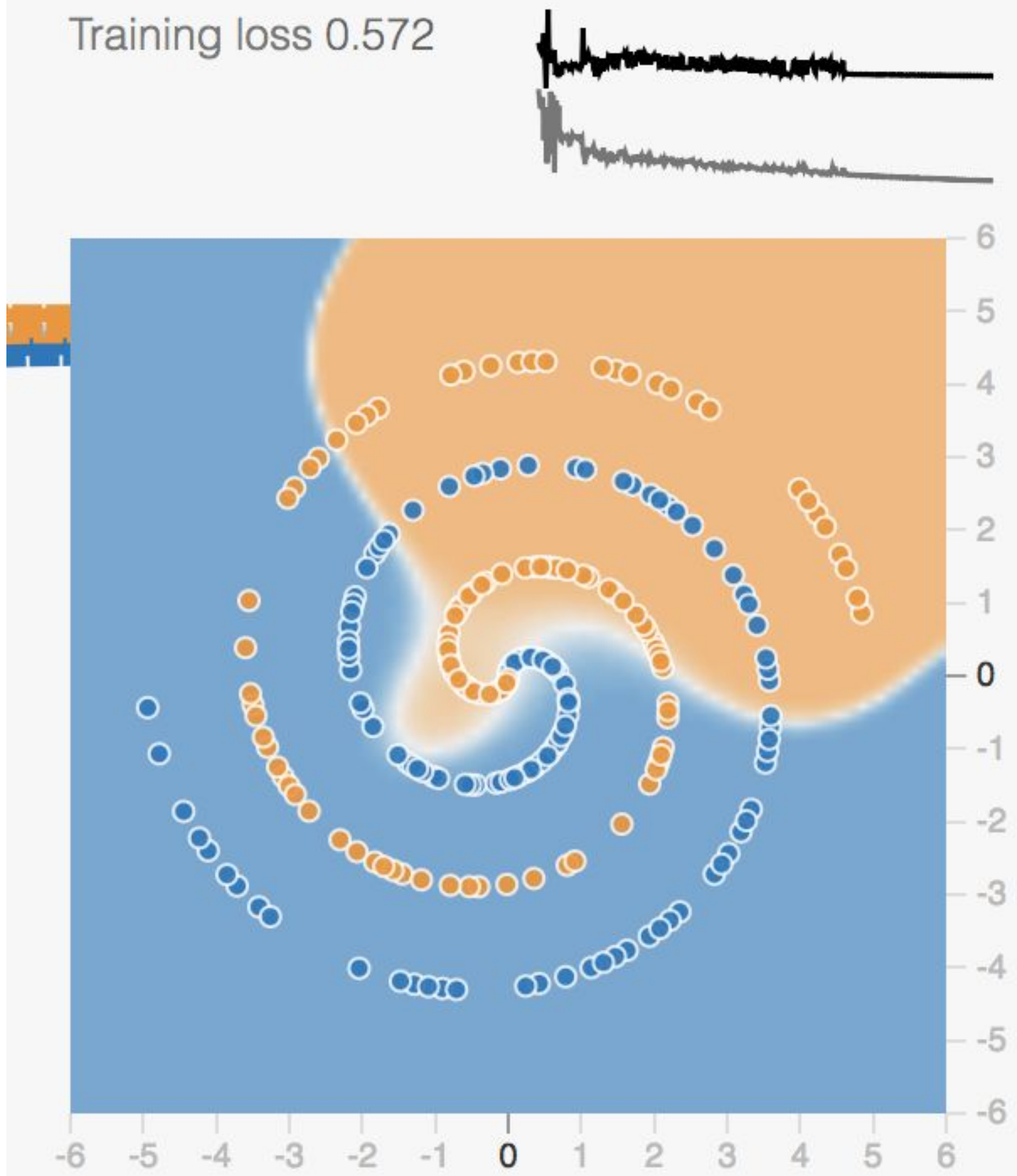
Deep Neural Network Models

DNN Dropout


OUTPUT
Training loss 0.002
6
5
4
3
2
1
0
-1
-2
-3
-4
-5
-6
-6 -5 -4 -3 -2 -1 0 1 2 3 4 5 6
Colors shows data, neuron and weight values.
-1
0
1

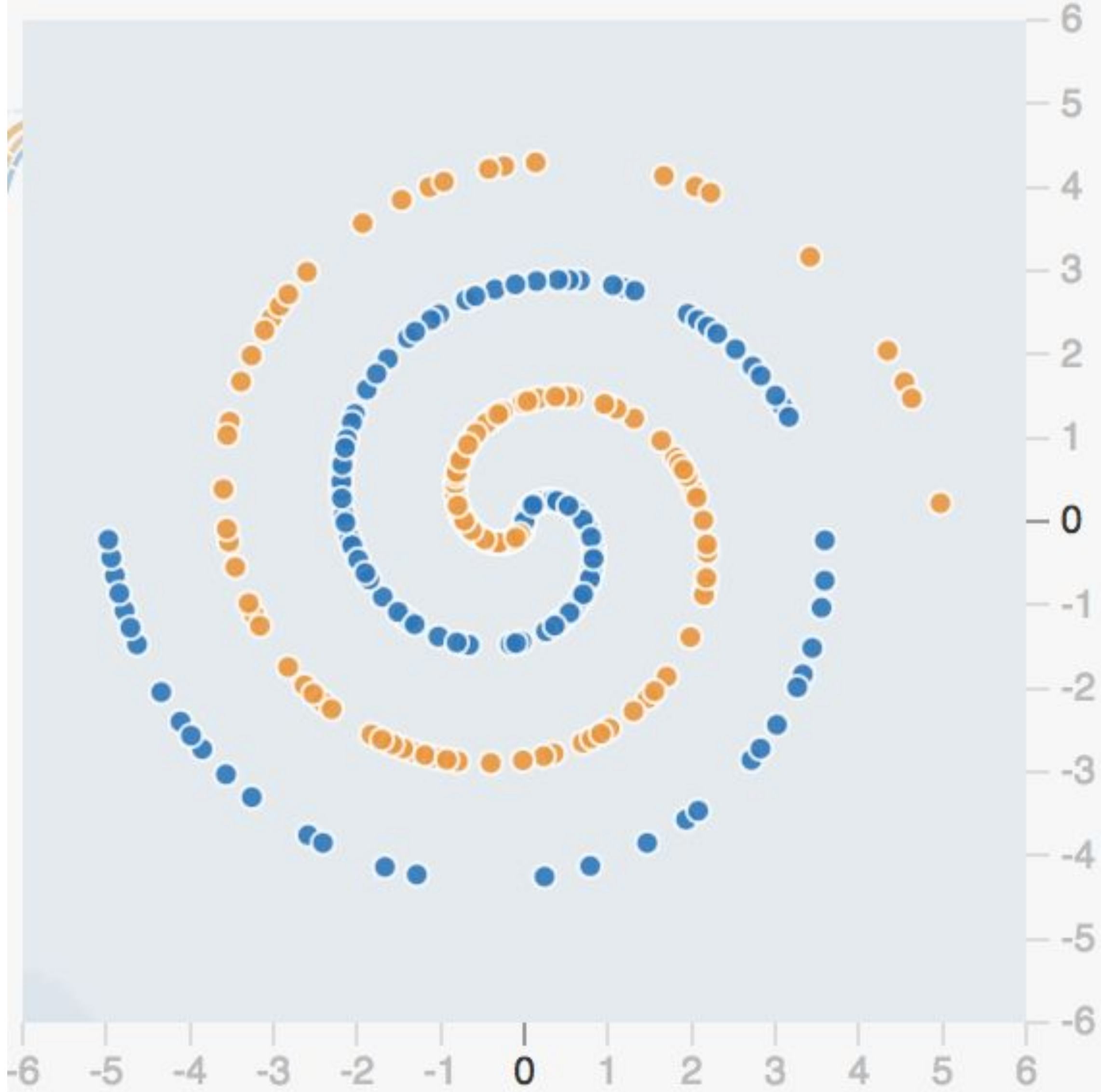
OUTPUT

Training loss 0.572



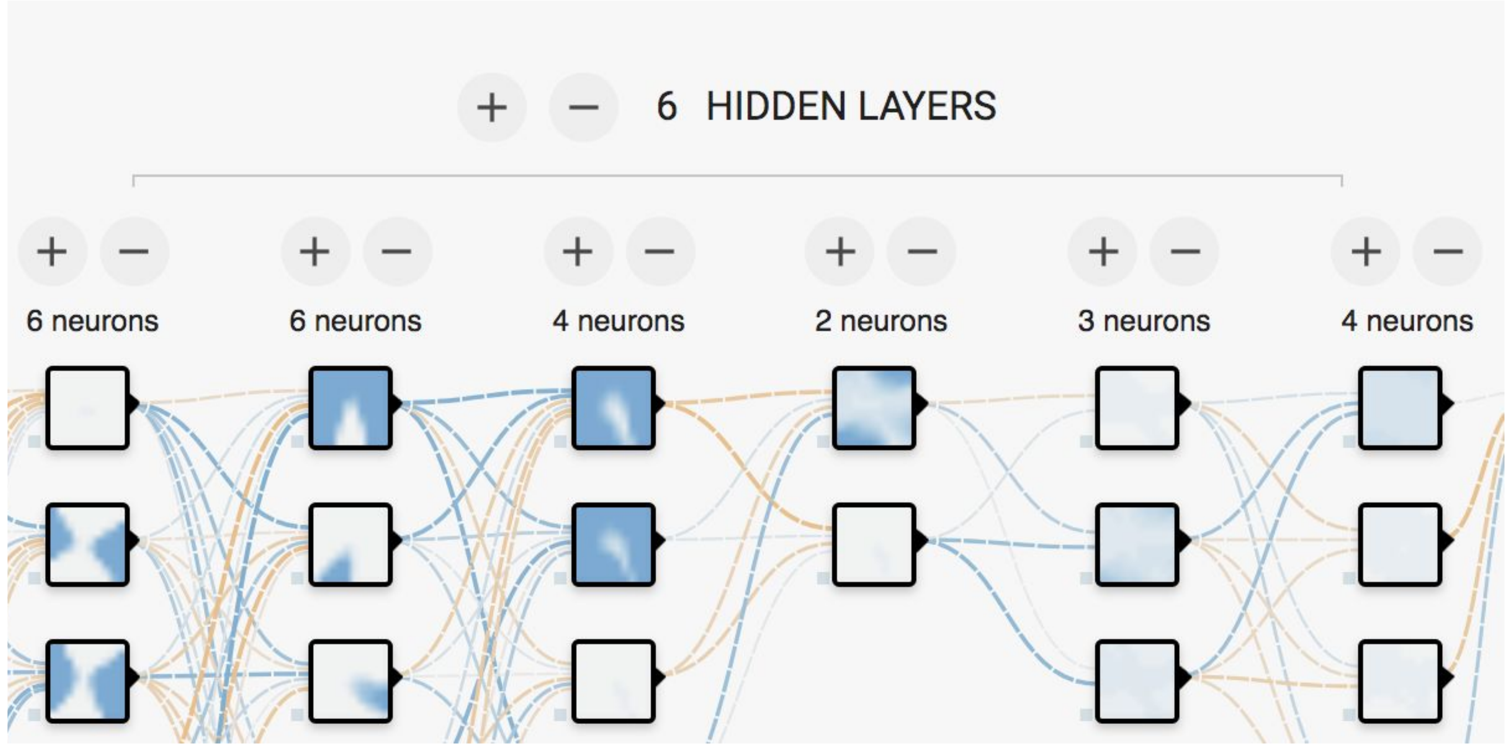
OUTPUT

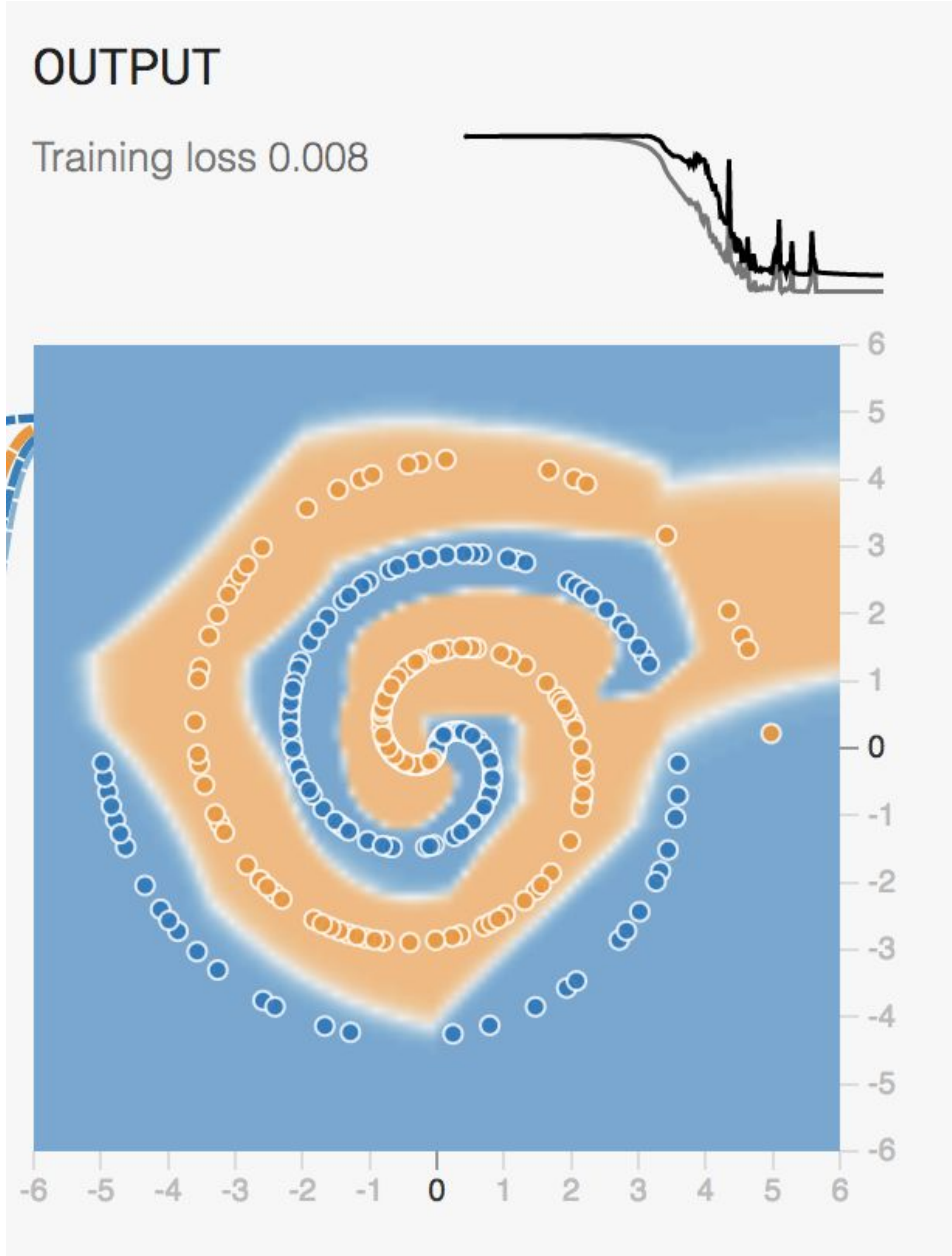
Training loss 0.504

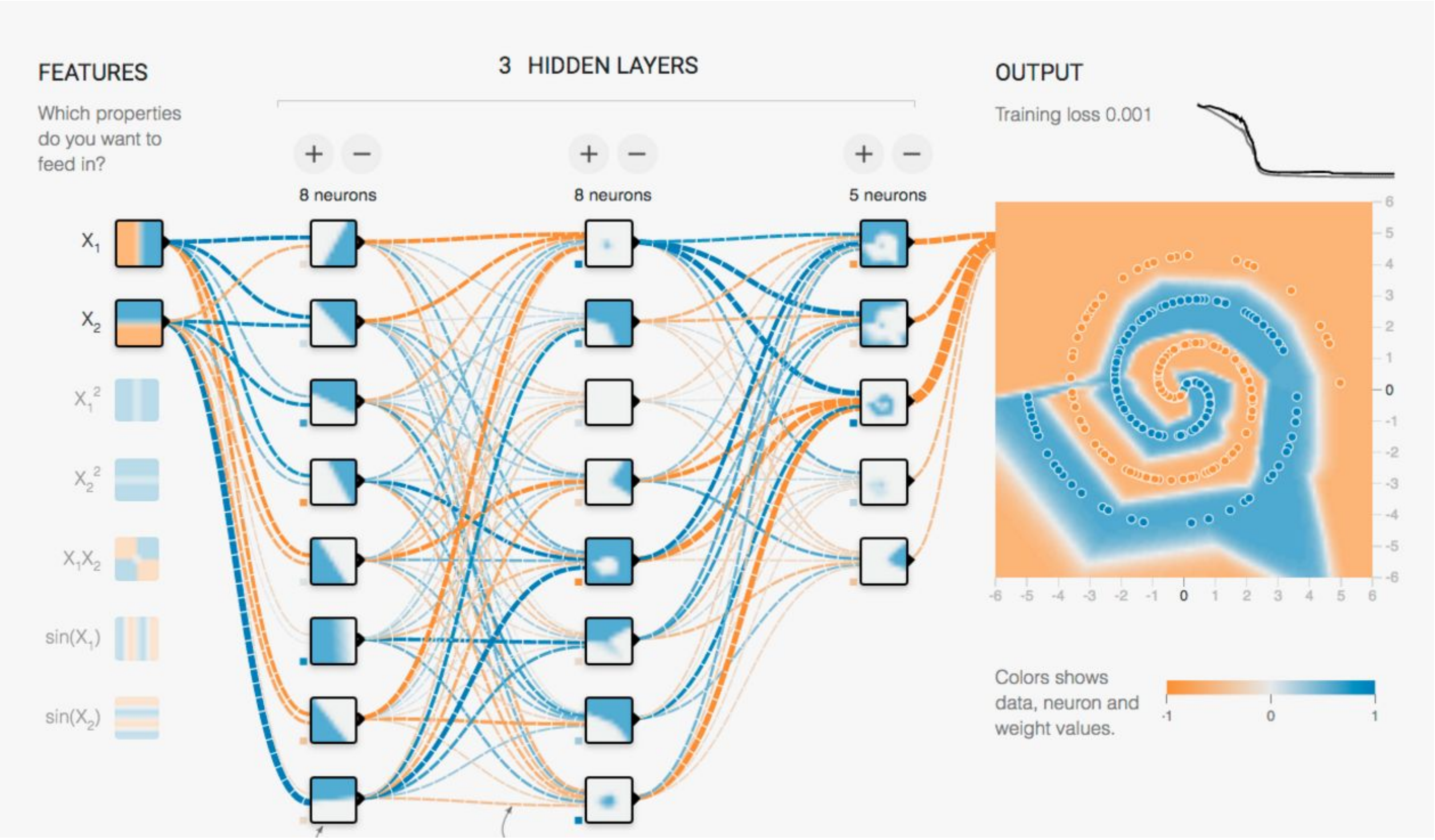


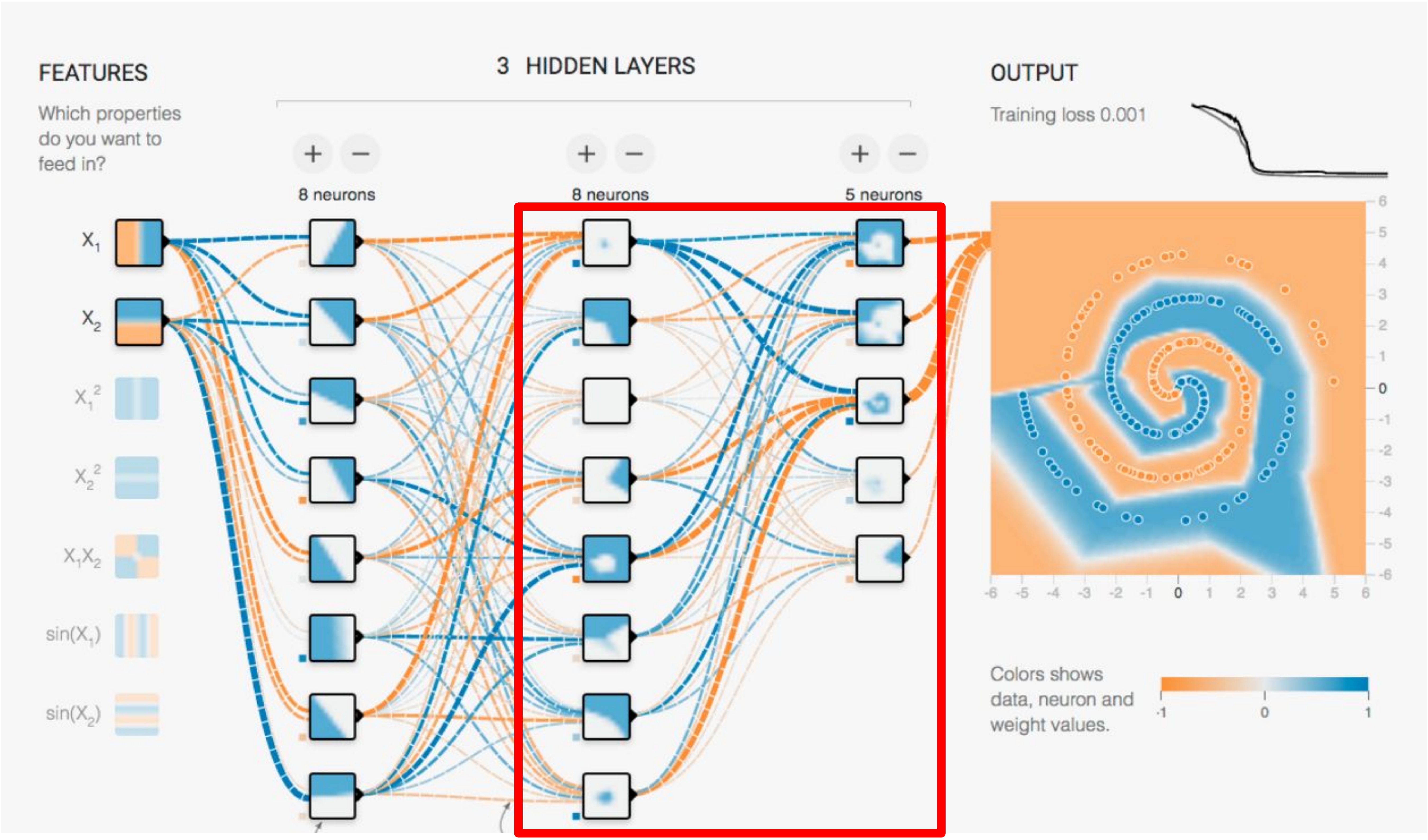
Activation

ReLU

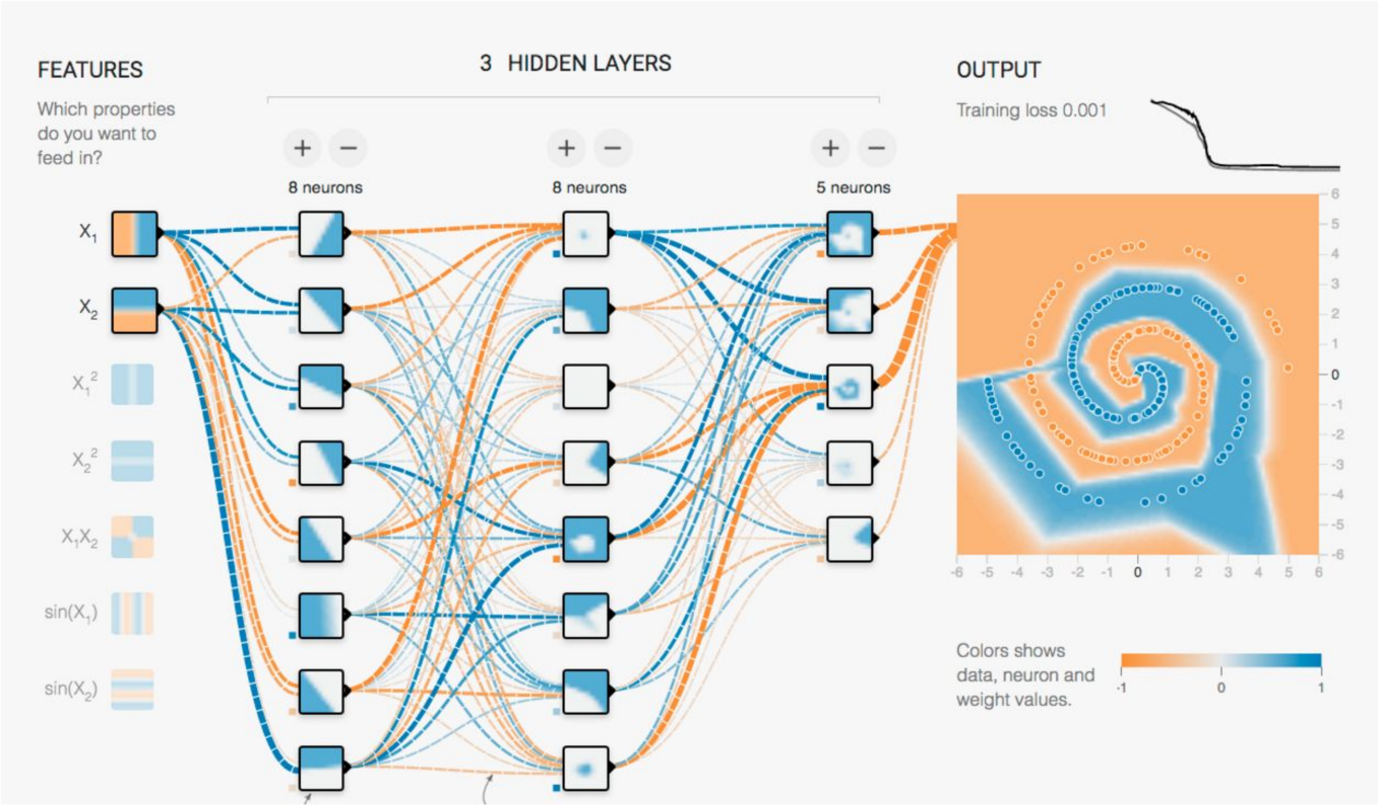

6 HIDDEN LAYERS
6 neurons
6 neurons
4 neurons
2 neurons
3 neurons
4 neurons


OUTPUT
Training loss 0.008
6
5
4
3
2
1
0
-1
-2
-3
-4
-5
-6
-6
-5
-4
-3
-2
-1
0
1
2
3
4
5
6


FEATURES
Which properties do you want to feed in?
X1
X2
X1^2
X2^2
X1X2
sin(X1)
sin(X2)
3 HIDDEN LAYERS
8 neurons
8 neurons
5 neurons
OUTPUT
Training loss 0.001
Colors shows data, neuron and weight values.
-1
0
1


FEATURES
Which properties do you want to feed in?
X1
X2
X1 2
X2 2
X1X2
sin(X1)
sin(X2)
3 HIDDEN LAYERS
8 neurons
8 neurons
5 neurons
OUTPUT
Training loss 0.001
Colors shows data, neuron and weight values.
-1
0
1

Models can learn which features to look at and which are most useful



Video Name: T-ICML-O_1_l6_lab_intro:_dnn_models

Format: Studio with Presenter

Presenter: Evan Jones

Lab

Image Classification with a Deep Neural Network Model

Evan Jones

Lab Steps

- Import the training dataset of MNIST handwritten images
- Reshape and preprocess the image data
- Setup your neural network model with 10 classes (one for each possible digit 0 through 9)
- Define and create your EstimatorSpec in tensorflow to create your custom estimator
- Define and run your train_and_evaluate function to train against the input dataset of 60,000 images and evaluate your model's performance

Video Name: T-ICML-O_1_l7_lab_solution:_dnn_models

Format: Studio with Presenter

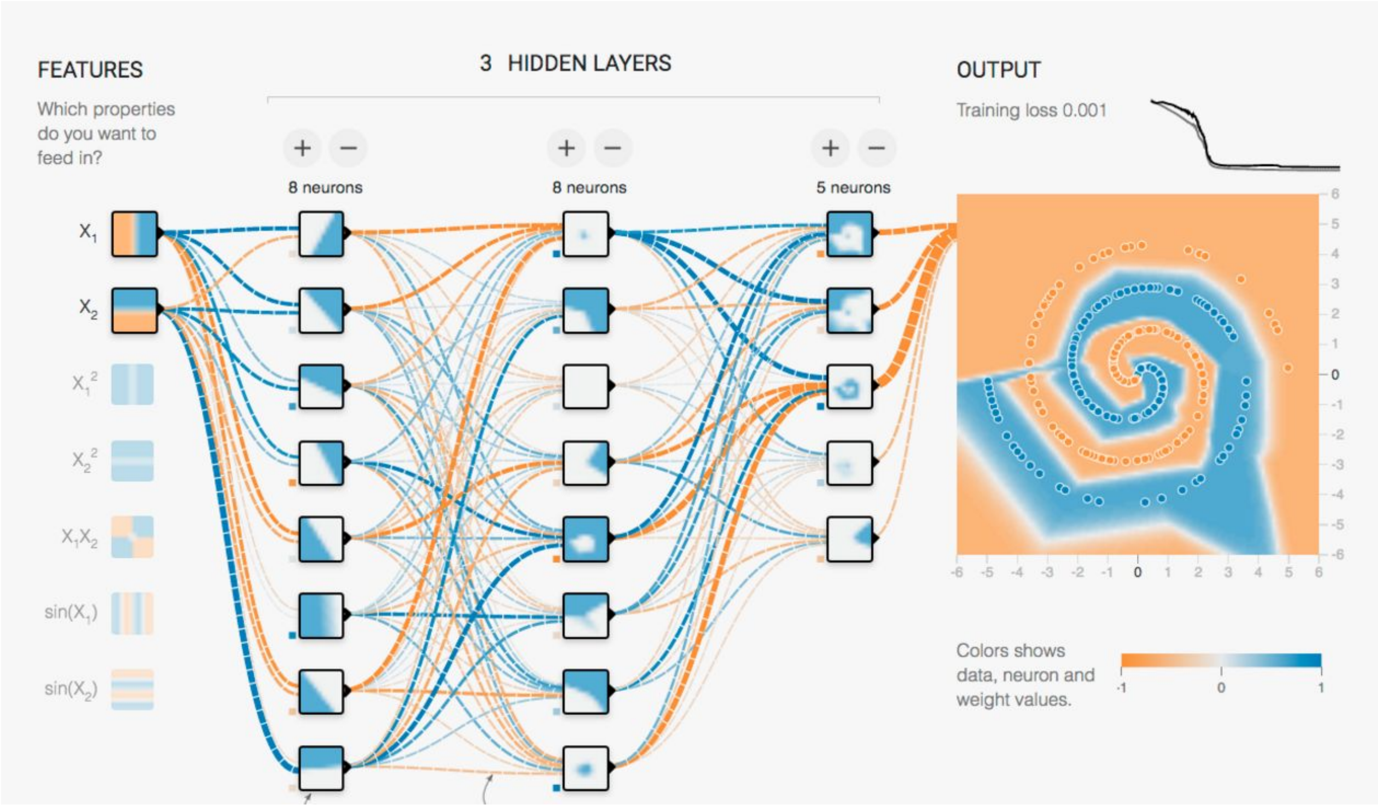
Presenter: Evan Jones

Video Name: T-ICML-O_1_l8_dropout

Format: Studio with Presenter

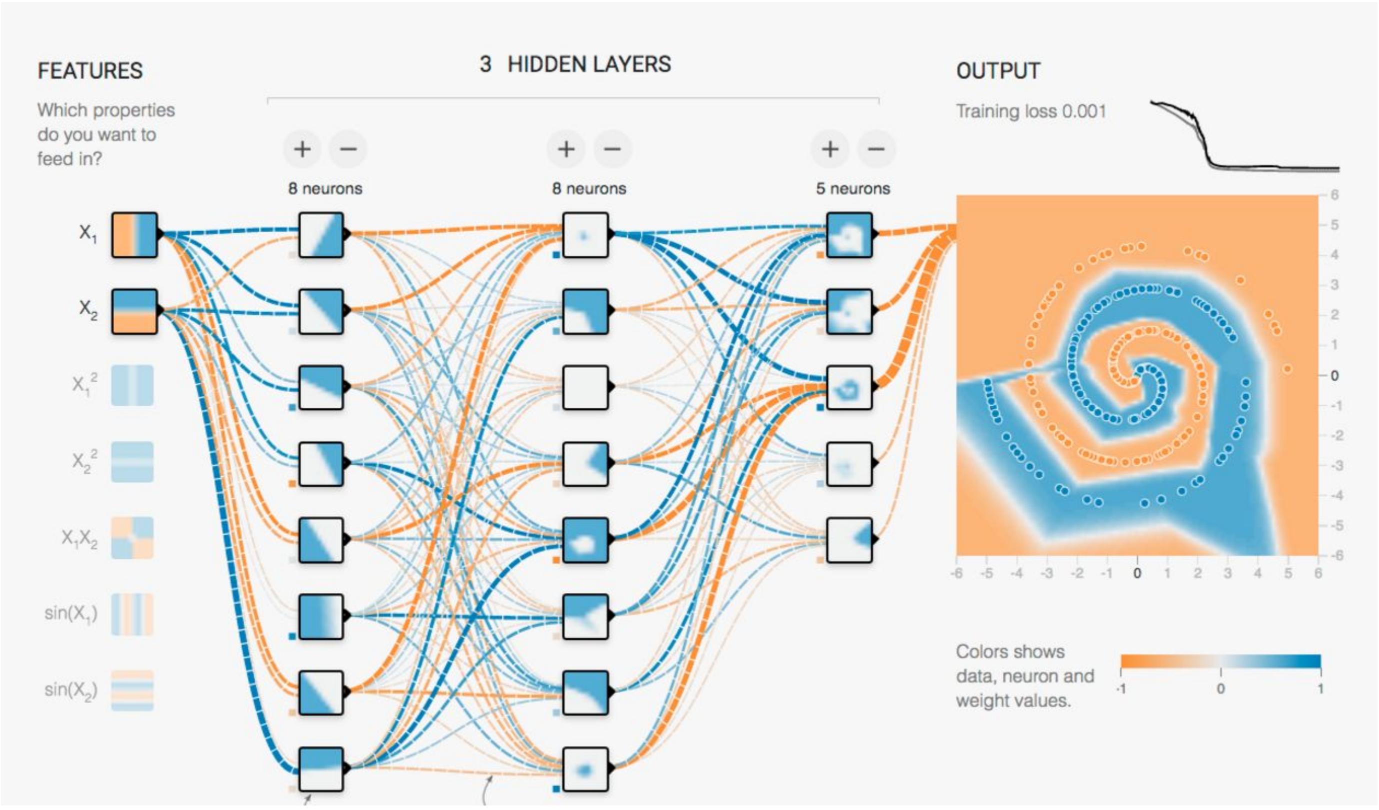
Presenter: Evan Jones

An infinitely large DNN could classify anything right?

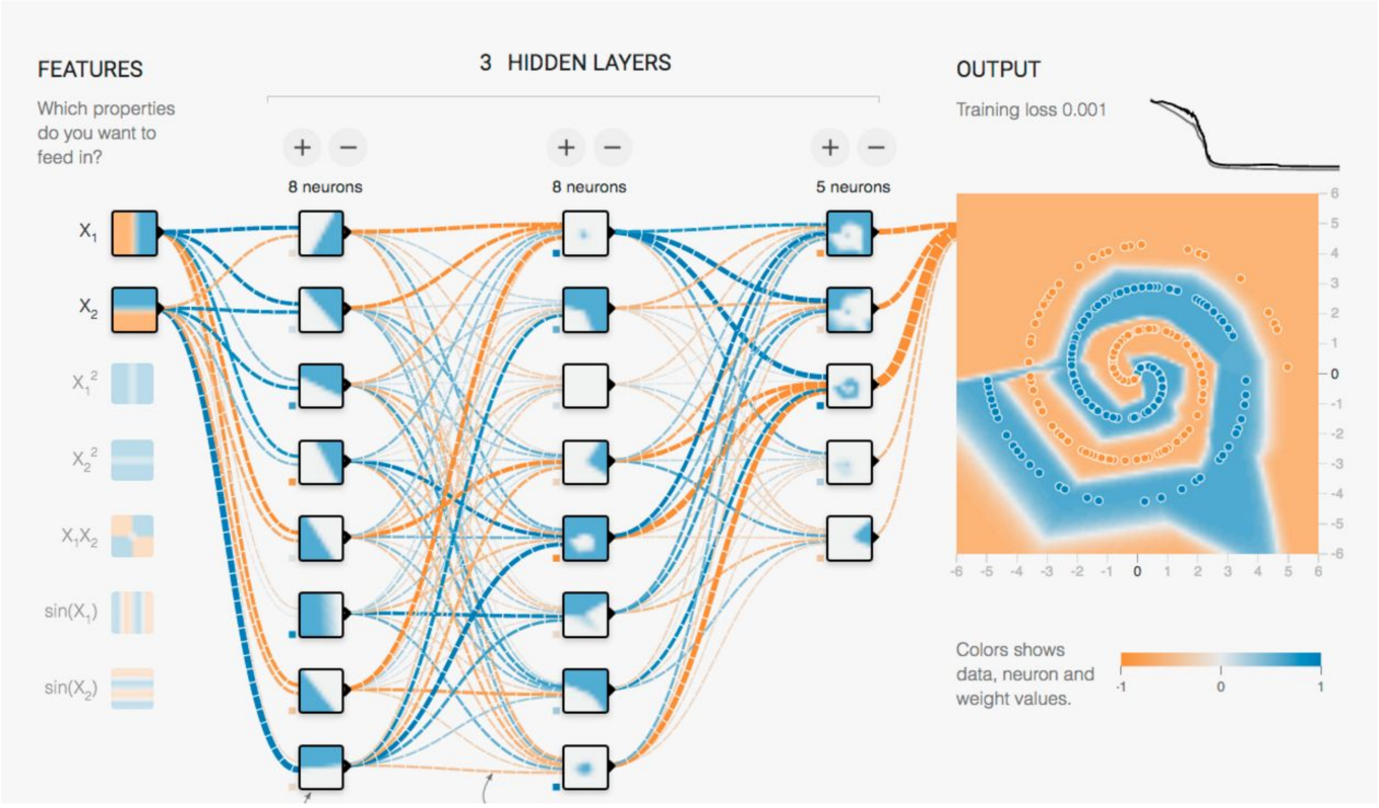


An infinitely large DNN
could ~~classify~~ *memorize* anything

Why not have a really large DNN?



Why not have a really large DNN?



1. Computational power
2. Training time
3. Likely to overfit

Recall: How can we combat overfitting in a DNN?

Combat overfitting with **regularization**

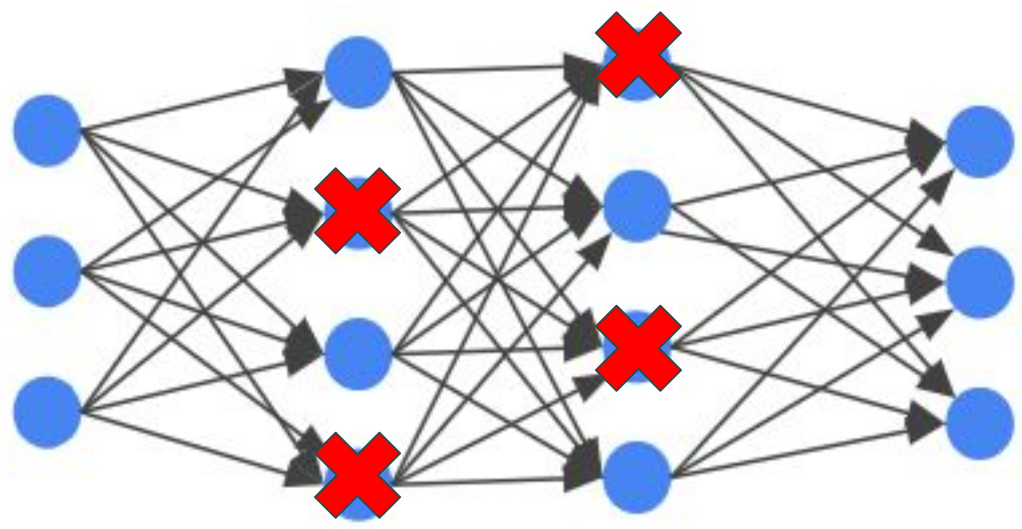
Which form of regularization is only used with neural networks?

1. Dropout
2. L1
3. L2

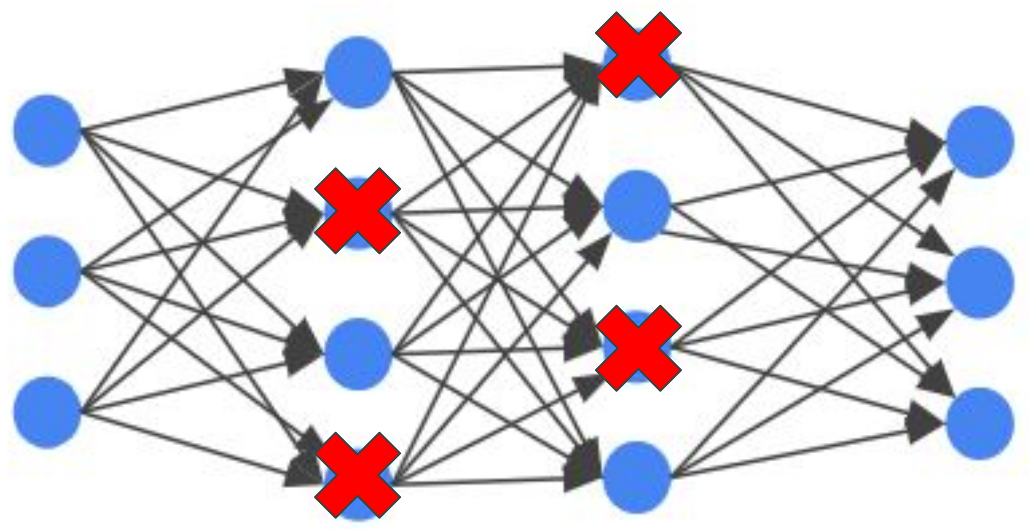
Which form of regularization is only used with neural networks?

1. **Dropout**
2. L1
3. L2

Use dropout to avoid overfitting in a DNN



Dropout can be thought of as an ensemble of NNs; useful in real-world problems



$$\lambda \sum_{i=1}^{k} \left| \omega_i \right|$$

L1 regularization term

$$\lambda \sum_{i=1}^{k} \omega_i^2$$

L2 regularization term

Adding a dropout layer to your DNN

```
layer_before = tf.layers.dense(previous_layer, 30)

dropout_layer = tf.layers.dropout(layer_before,
    rate=drop_probability,
    # only dropout when training
    training=(mode == tf.estimator.ModeKeys.TRAIN))

 layer_after = tf.layers.dense(dropout_layer, 15)
```

Video Name:
T-ICML-O_1_l9_lab_intro:_dnn_models_with_dropout

Format: Studio with Presenter

Presenter: Evan Jones

Lab

Image Classification with a DNN Model with Dropout

Evan Jones

Lab Steps

- Import the training dataset of MNIST handwritten images
- Reshape and preprocess the image data
- Setup your neural network model with 10 classes (one for each possible digit 0 through 9)
- Add a Dropout layer
- Define and create your EstimatorSpec in tensorflow to create your custom estimator
- Define and run your train_and_evaluate function to train against the input dataset of 60,000 images and evaluate your model's performance

Video Name:
T-ICML-O_1_l10_lab_solution:_dnn_models_with_dropout

Format: Studio with Presenter

Presenter: Evan Jones