Neural Networks for Machine Learning

tensorflow playground
TensorFlow Playground

• Great javascript app demonstrating many basic neural network concepts (e.g., MLPs)
• Doesn’t use TensorFlow software, but a lightweight js library
• Runs in a Web browser
• See http://playground.tensorflow.org/
• Code also available on GitHub
• Try the playground exercises in Google’s machine learning crash course
Tinker With a **Neural Network** Right Here in Your Browser.
Don’t Worry, You Can’t Break It. We Promise.

**DATA**
Which dataset do you want to use?
- [Random](#)
- [Gauss](#)

Ratio of training to test data: 50%
Noise: 0
Batch size: 10

**FEATURES**
Which properties do you want to feed in?
- $X_1$
- $X_2$
- $X_1^2$
- $X_2^2$
- $X_1X_2$
- $\sin(X_1)$
- $\sin(X_2)$

**OUTPUT**
Test loss 0.435
Training loss 0.432

Colors shows data, neuron and weight values.

Show test data  Discretize output

---

**Additional Information**

[HTTP://PLAYGROUND.TENSORFLOW.ORG/](http://playground.tensorflow.org/)
Datasets

- Six datasets, each with 500 \((x,y)\) points on a plane where \(x\) and \(y\) between -5 and +5
- Points have \textit{labels} of positive (orange) or negative (blue)
- Two possible machine learning \textit{tasks}:
  - Classification: Predict class of test points
  - Regression: find function to separate classes
- \textit{Evaluation}: split dataset into training and test, e.g., 70% training, 30% test
Available Input features

\[ X_1 \] Point’s x value
\[ X_2 \] Point’s y value
\[ X_1^2 \] Point’s x value squared
\[ X_2^2 \] Point’s y value squared
\[ X_1X_2 \] Product of point’s x & y values
\[ \sin(X_1) \] Sine of point’s x value
\[ \sin(X_2) \] Sine of point’s y value
Designing a neural network

• Simple feed forward NNs have a few choices
  – What input features to use
  – How many hidden layers to have
    • How many neurons are in each layer
    • How each layer is connected to ones before & after

• Complex NNs have more choices
  – E.g., CNNs, RNNs, etc.

• High-level interfaces (Keras, TensorFlow, PyTorch, ...) try to make this easier
Tinker With a **Neural Network** Right Here in Your Browser.
Don’t Worry, You Can’t Break It. We Promise.

**DATA**
Which dataset do you want to use?

- [ ] Flowers
- [ ] Swirls
- [ ] Spiral

Ratio of training to test data: 50%
Noise: 0
Batch size: 10

**FEATURES**
Which properties do you want to feed in?

- $x_1$
- $x_2$
- $x_1^2$
- $x_2^2$
- $x_1 x_2$
- $\sin(x_1)$
- $\sin(x_2)$

**OUTPUT**
Test loss 0.435
Training loss 0.432

This is the output from one neuron. Hover to see it larger.

The outputs are mixed with varying weights, shown by the thickness of the lines.

Colors shows data, neuron and weight values.

[Show test data] [Discretize output]

[HTTP://PLAYGROUND.TENSORFLOW.ORG/]
Training a Neural Network

• Neural networks are used for supervised machine learning and need to be trained

• The training process is broken done in a series of *epochs*
  In each epoch, all of the training data is run through the system to adjust the NN parameters

• Process ends after a fixed # of epochs or when error rate flattens or starts increasing
• Divide training data into batches of instances (e.g., batch size = 10)

• For each epoch:
  – For each batch:
    • Instances run through network, noting difference between predicted and actual value
    • Backpropagation used to adjust connection weights
  – Stop when training loss flatten out

• If test loss is too high, then try
  – Adding additional hidden layers
  – Adding more features to inputs
  – Adjusting hyperparameters (e.g., learning rate)
  – Get more training data
**Hyperparameters**

- Parameters whose values are set before the learning process begins
- Basic neural network hyperparameters
  - Learning rate (e.g., 0.03)
  - Activation function (e.g., ReLU)
  - Regularization (e.g., L2)
  - Regularization rate (e.g., 0.1)
**Learning rate**

- **Gradient descent** used in backpropagation to adjust weights to minimize the loss function
- Learning rate determines how much weights are adjusted each time
- If too high, we may miss some or most minima
  - Result: erratic performance or never achieving a low loss
- If too low, learning will take longer than necessary
Gradient Descent

• Iterative process used in ML to find local minimum in our loss function measuring errors
• Moves in direction of steepest descent
• Step size decreases as steepness lessens to avoid missing minima
• Custom variants for NNs include adam optimization
Activation Function

• Determines a node’s output given its inputs
• The ReLu (rectified linear unit) is simple and a good choice for most networks
• Returns zero for negative values and its input for positive ones
  \[-f(x) = \max(0,x)\]
Regularization

• Parameter to control overfitting, i.e. when the model does well on training data but poorly on new, unseen data

• L2 regularization is the most common

• Using dropout is another common way of controlling overfitting in neural networks
  – At each training stage, some hidden nodes temporarily removed (dropped out)
Hyperparameter optimization

• How do we find the best settings for these hyperparameters?

• Experimentation
  – Experiment with a range of different settings (e.g., for learning rate) via multiple runs
  – Use a grid search tool, e.g., scikit learn’s

• Experience
  – Similar problems with similar data will probably benefit from similar settings
Tinker With a **Neural Network** Right Here in Your Browser.
Don’t Worry, You Can’t Break It. We Promise.

[HTTP://PLAYGROUND.TENSORFLOW.ORG/](http://playground.tensorflow.org/)

**DATA**
Which dataset do you want to use?
- ![Dataset Icons](image)

Ratio of training to test data: 50%
Noise: 0
Batch size: 10

**FEATURES**
Which properties do you want to feed in?

**OUTPUT**
Test loss 0.435
Training loss 0.432

The outputs are mixed with varying weights, shown by the thickness of the lines.

This is the output from one neuron. Hover to see it larger.

Colors shows data, neuron and weight values.

- Show test data
- Discretize output