statinfer

Neural Networks - Part2

Venkat Reddy

Statinfer.com

Data Science Training and R&D

Training on

Data Science

Bigdata Analytics

Machine Learning

Predictive Modelling

Deep Learning

Corporate Training

Classroom Training

Online Training

Contact us

info@statinfer.com

venkat@statinfer.com

Note

- This presentation is just my class notes. The course notes for data science training is written by me, as an aid for myself.
- The best way to treat this is as a high-level summary; the actual session went more in depth and contained detailed information and examples
- Most of this material was written as informal notes, not intended for publication
- Please send questions/comments/corrections to info@statinfer.com
- Please check our website statinfer.com for latest version of this document

-Venkata Reddy Konasani

(Cofounder statinfer.com)

Contents

- Overfitting
- Regularization
- Activation Functions
- Learning Rate

The problem of Overfitting

The problem of Overfitting

- Neural networks are very powerful. They have capacity to learn any type of pattern.
- With high number of hidden layers, we can fit to training data with any level of non-linearity
- Too many hidden layers might be fitting the model for random pattern or noise in the data
- Throughout the neural network algorithm we were trying optimise weights to make the error zero. This might lead to overfitting

Cost function regularization

$$Actual\ Cost\ function\ or\ Eror = \sum_{i=1}^{n}\left[y_i - g\left(\sum_{k=1}^{m} w_k h_{ki}\right)\right]^2$$

$$New\ Regularized\ error = \sum_{i=1}^{n}\left[y_i - g\left(\sum_{k=1}^{m} w_k h_{ki}\right)\right]^2 + \frac{1}{2}\lambda\sum_{i=1}^{n} {w_i}^2$$

- Apart from minimising error sum of squares, we are minimising error + weights sum of squares also
- The second term is imposing a penalty on weights.
- This is known as Regularization – What is **Regularization**

What is Regularization?

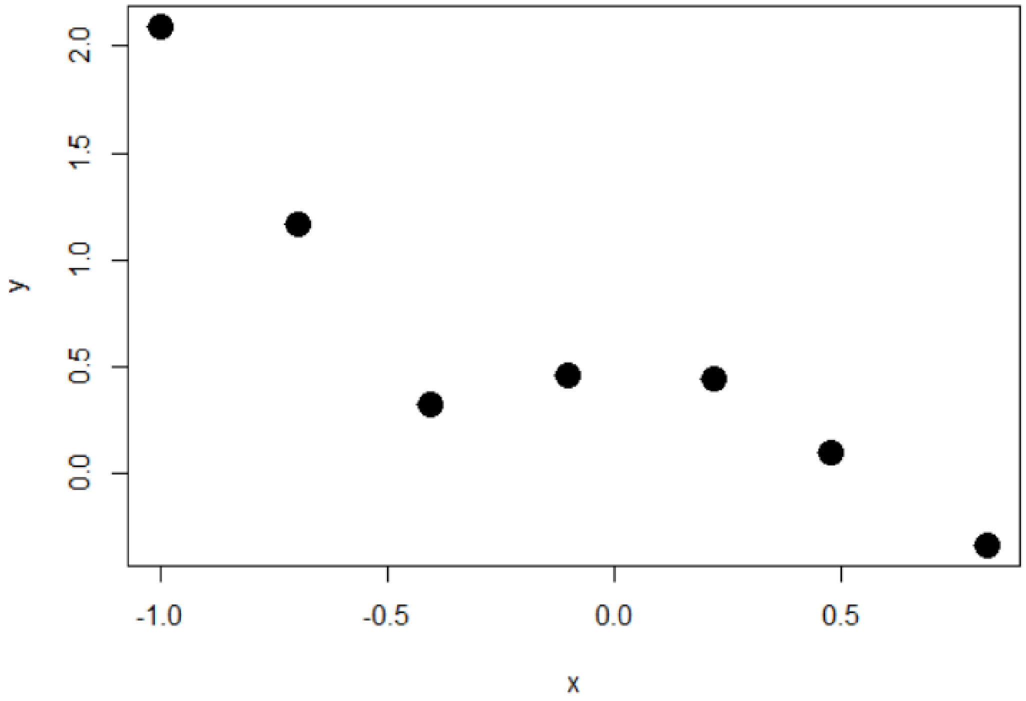
- In any model building we try to find weights by minimising the cost function.
- For example in regression we try to find minimum squared error –Cost function
- Cost function $=\sum(y_i - \sum \beta_k x_{ki})^2$
- Always trying to minimise the overall cost function might not be a good idea.
- A really high degree polynomial function will make this cost function zero. But that will lead to overfitting

LAB: Higher order polynomial model

- Data: Regular/Reg_Sim_Data1.csv
- Plot the points X vs Y
- Build three regression models m1, m2 and m3 and calculate SSE
  - m1: simple linear model – Calculate SSE
  - m2: Second order polynomial model – Calculate SSE
  - m3: Fifth order polynomial model – Calculate SSE
- Which model is the best based on SSE.

Code: Higher order polynomial model

```
plot(Reg_Sim_Data1$x, Reg_Sim_Data1$y,lwd=10)
```



Code: Higher order polynomial model

```
> #Simple Linear regression model
> m1<-lm(y~x, data=Reg_Sim_Data1)
> SSE1=sum((Reg_Sim_Data1$y-predict(m1))^2)
> SSE1
[1] 0.7107401
>
> #Second order polynomial regression
> m2<-lm(y~x+I(x^2), data=Reg_Sim_Data1)
> SSE2=sum((Reg_Sim_Data1$y-predict(m2))^2)
> SSE2
[1] 0.4572317
>
> #Fifth order digree polynomial regression
> m3<-lm(y~x+I(x^2)+I(x^3)+I(x^4)+I(x^5), data=Reg_Sim_Data1)
> #summary(m3)
> SSE3=sum((Reg_Sim_Data1$y-predict(m3))^2)
> SSE3
[1] 0.01056289
```

Reduce Overfitting

- We have two options to reduce overfitting
  - Build a simple model. Drop all polynomial terms. – But this might lead to underfitting
  - Or keep the complex terms but give them less weightage. This will take care of overfitting
- Instead of minimising SSE alone, minimise both SSE and weights
- The regularization term Imposes some penalty on weights. It impacts the overall weights and reduces the overfitting

$$New\ Cost\ function = \sum\left(y_i - \sum \beta_k x_{ki}\right)^2 + \frac{1}{2}\lambda \sum {\beta_k}^2$$

Regularization comments

- By adding the regularization term we are avoiding the risk of over fitting
- Regularization also allows us to have a complex model.
- Regularization is added in final cost function but its final impact is on weights
- A high value of λ avoids the overfitting

$$New\ Cost\ function = \sum \left(y_i - \sum \beta_k x_{ki} \right)^2 + \frac{1}{2} \lambda \sum {\beta_k}^2$$

LAB: Regularization

- Build a fifth order polynomial function.
- consider new regularized cost function.
- Recalculate the weights. Build three models
  - λ =0
  - λ=1
  - λ=10
- Plot the models. Which model is more generalised (less over fitted)
- If you have to choose one model with high degree polynomials what will be your λ value?

Code: Regularization

Create independent variable matrix X

```
> mydata<-Reg_Sim_Data1
>
> m = length(mydata$x) # samples
> x = matrix(c(rep(1,m), mydata$x, mydata$x^2, mydata$x^3, mydata$x^4, mydata$x^5), ncol=6)
> x
     [,1]     [,2]       [,3]         [,4]         [,5]          [,6]
[1,]    1 -0.99768 0.99536538 -0.993056135 0.9907522445 -9.884537e-01
[2,]    1 -0.69574 0.48405415 -0.336775833 0.2343084178 -1.630177e-01
[3,]    1 -0.40373 0.16299791 -0.065807147 0.0265683196 -1.072643e-02
[4,]    1 -0.10236 0.01047757 -0.001072484 0.0001097795 -1.123703e-05
[5,]    1  0.22024 0.04850566  0.010682886 0.0023527988  5.181804e-04
[6,]    1  0.47742 0.22792986  0.108818272 0.0519520194  2.480293e-02
[7,]    1  0.82229 0.67616084  0.556000300 0.4571934871  3.759456e-01
```

Create Dependent variable matrix Y

```
> n = ncol(x) # features
> y = matrix(mydata$y, ncol=1)
> y
         [,1]
[1,]  2.08850
[2,]  1.16460
[3,]  0.32870
[4,]  0.46013
[5,]  0.44808
[6,]  0.10013
[7,] -0.32952
>
```

Code: Regularization

```
> #Prepare lambda and weights
> lambda = c(0,1,10)
> th = array(0,c(n,length(lambda)))
>
> #Below matrix will be used in normal equations later
> d = diag(1,n,n)
> d[1,1] = 0
```

```
> #Below matrix will be used in normal equations later
> d = diag(1,n,n)
> d[1,1] = 0
> # apply normal equations for each of the lambda's
> for (i in 1:length(lambda)) {
+   th[,i] = solve(t(x) %*% x + (lambda[i] * d),tol = 1e-30) %*% (t(x) %*% y)
+ }
>
> #Print new weights
> th
          [,1]       [,2]        [,3]
[1,]  0.4725288  0.3975953  0.52047074
[2,]  0.6813529 -0.4206664 -0.18250706
[3,] -1.3801284  0.1295921  0.06064258
[4,] -5.9776875 -0.3974739 -0.14817721
[5,]  2.4417327  0.1752555  0.07433006
[6,]  4.7371143 -0.3393877 -0.12795737
```

- $(X^tX)^{-1} (X^tY)$ is the general.
- But we are solving $(X^tX+ \lambda)^{-1} (X^tY)$
- This is the result of adding regularization in the cost function

Code: Regularization

```
> #Print new weights
> th
           [,1]       [,2]        [,3]
[1,]  0.4725288  0.3975953  0.52047074
[2,]  0.6813529 -0.4206664 -0.18250706
[3,] -1.3801284  0.1295921  0.06064258
[4,] -5.9776875 -0.3974739 -0.14817721
[5,]  2.4417327  0.1752555  0.07433006
[6,]  4.7371143 -0.3393877 -0.12795737
```

The new weights will be adjusted based on λ

```
> th[,1]
[1]  0.4725288  0.6813529 -1.3801284 -5.9776875  2.4417327  4.7371143
>
> #Fifth order digree polynomial regression
> m3<-lm(y~x+I(x^2)+I(x^3)+I(x^4)+I(x^5), data=Reg_Sim_Data1)
> m3$coefficients
(Intercept)           x      I(x^2)      I(x^3)      I(x^4)      I(x^5) 
  0.4725288   0.6813529  -1.3801284  -5.9776875   2.4417327   4.7371143 
```

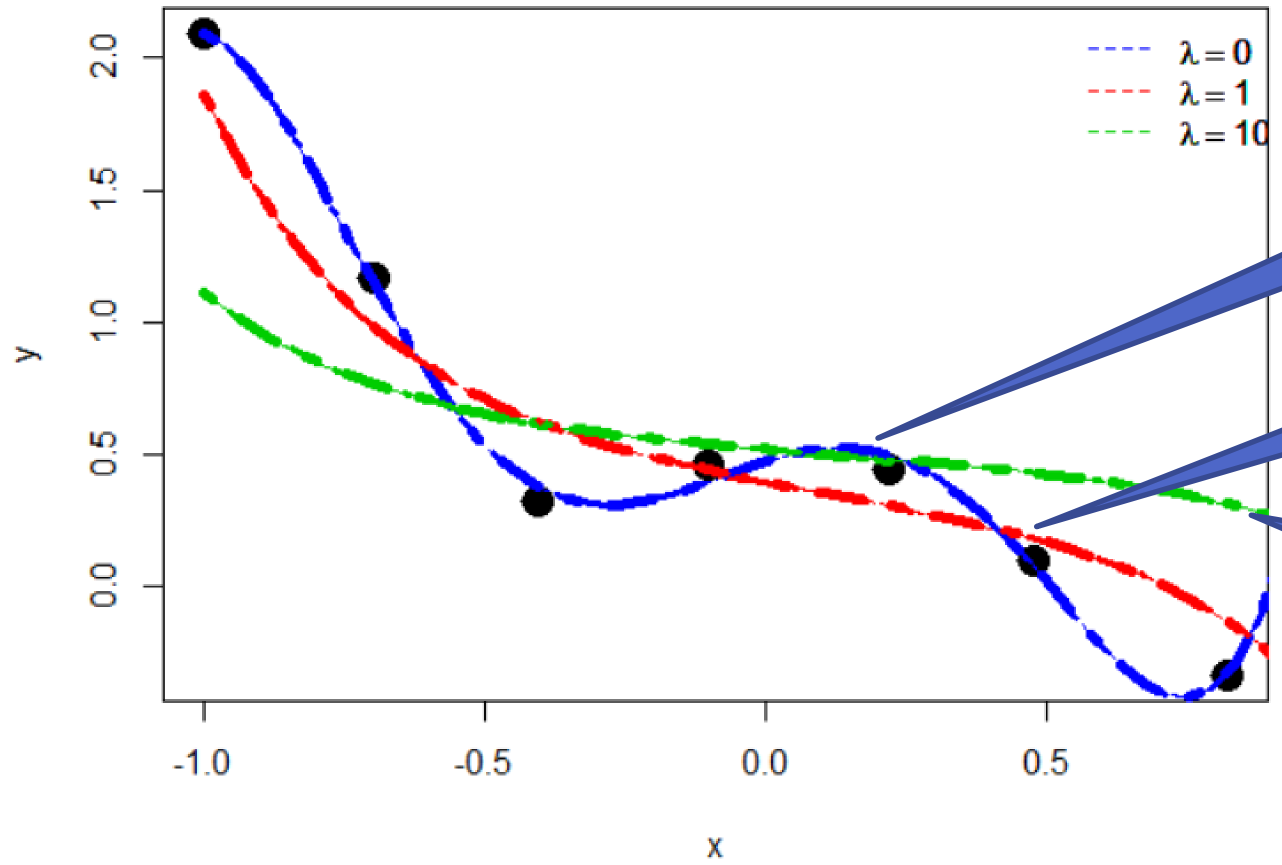
Observe that when λ = 0 the weights are same as usual regression model

Code : Plotting All three models

```
# plot
plot(mydata,lwd=10)

# lets create many points
nwx = seq(-1, 1, len=50);
x = matrix(c(rep(1,length(nwx)), nwx, nwx^2, nwx^3, nwx^4, nwx^5), 
ncol=6)
lines(nwx, x %*% th[,1], col="blue", lty=2)
lines(nwx, x %*% th[,2], col="red", lty=2)
lines(nwx, x %*% th[,3], col="green3", lty=2)
legend("topright", c(expression(lambda==0), 
expression(lambda==1),expression(lambda==10)), lty=2,col=c("blue", 
"red", "green3"), bty="n")
```

Code : Plotting All three models



Blue line λ=0
Overfitted model when there is no regularization

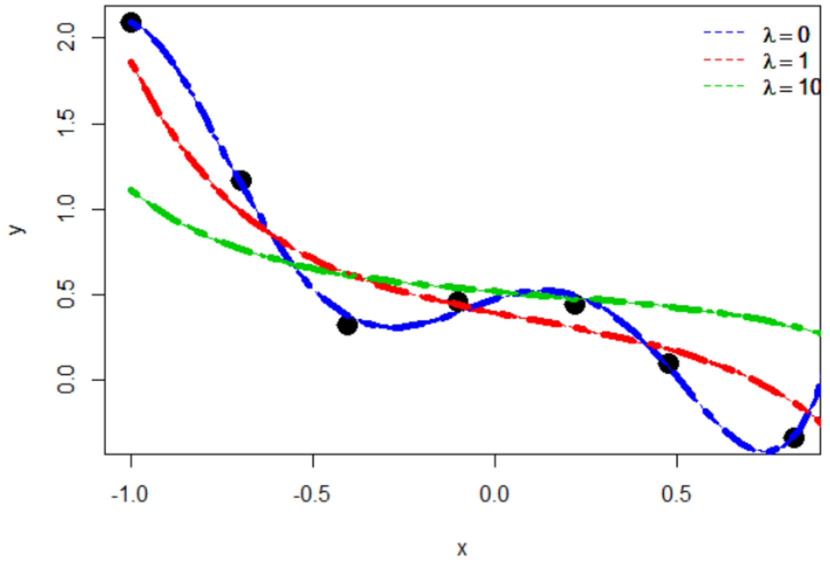
Red line λ=1
Slightly smooth model

Green line λ=10
Very smooth model, almost a straight line.

Code :Choosing the lambda

- If you have to choose one model with high degree polynomials what will be your λ value?

```
> #Lets consider model with lamda=1
> x = matrix(c(rep(1,m), mydata$x, mydata$x^2, mydata$x^3, mydata$x^4, mydata$x^5), nc
>
> #Weigths
> th[,2]
[1]  0.3975953 -0.4206664  0.1295921 -0.3974739  0.1752555 -0.3393877
>
> #Predictions
> pred<-x %*% th[,2]
>
> #SSE
> SSE_Final=sum((mydata$y-pred)^2)
> SSE_Final
[1] 0.243632
```



How Regularization works in Neural Nets

- In linear regression having higher order polynomial terms lead to overfitting.
- Did we reduce the polynomial terms?
- We added regularization term in the cost function recalculated the weights.

- In neural networks having too many hidden layers might lead to overfitting.
- Shall we reduce the number of hidden layers?

How Regularization works in Neural Nets

- In neural network having too many hidden layers will lead to too many weights, which might lead to over fitting
- Having very less hidden layers might lead to underfitting
- How do we keep several hidden layers with less weightage?
- We can use regularization and have an optimal value for λ to avoid over fitting

$$Actual\ Cost\ function\ or\ Eror = \sum_{i=1}^{n}\left[y_i - g\left(\sum_{k=1}^{m} w_k h_{ki}\right)\right]^2$$

$$New\ Regularized\ error = \sum_{i=1}^{n}\left[y_i - g\left(\sum_{k=1}^{m} w_k h_{ki}\right)\right]^2 + \frac{1}{2}\lambda\sum_{i=1}^{n} w_i^2$$

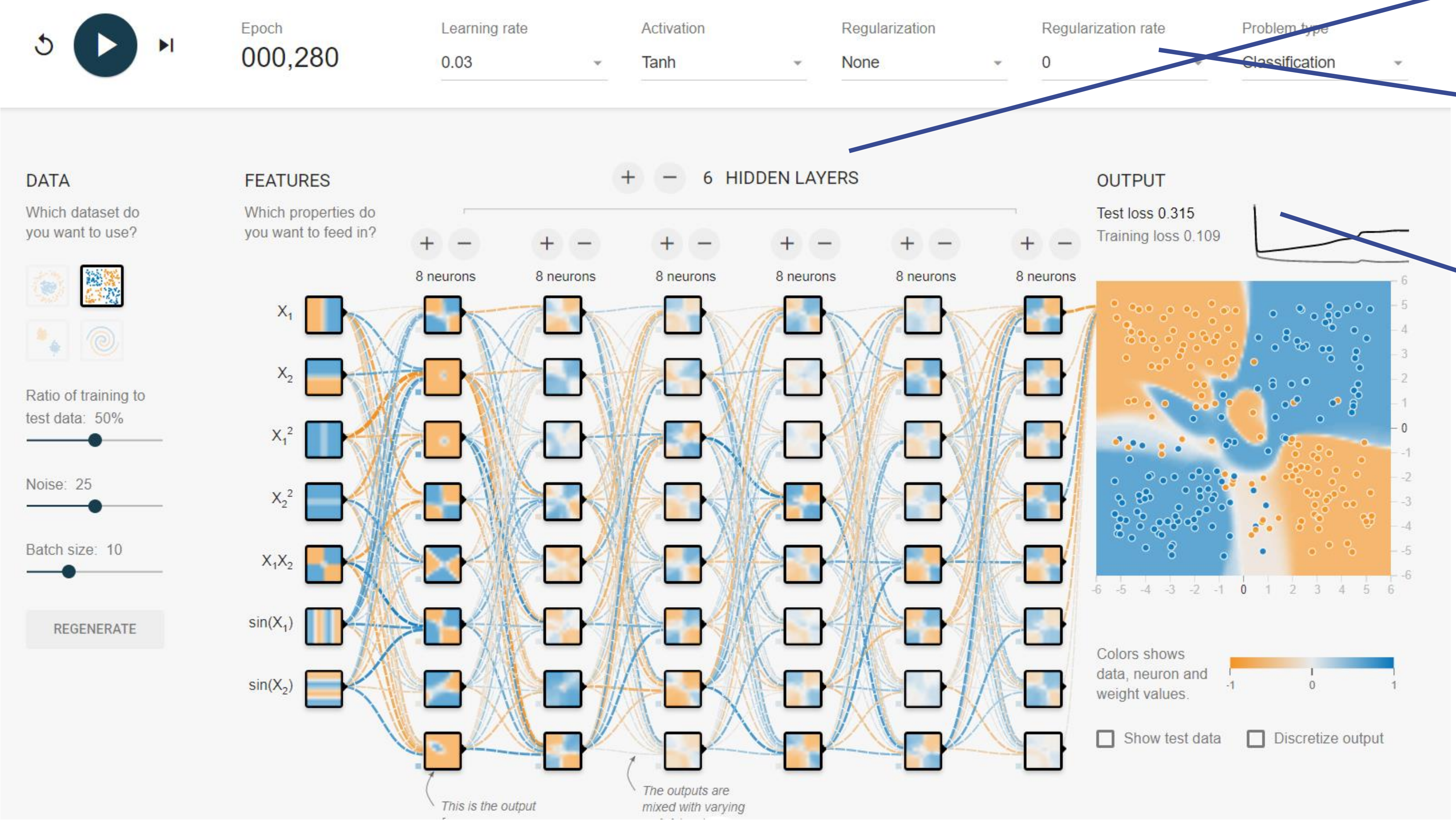
Important Note

- We directly work with weights in regularization.
- It is important to **standardize the data** before applying regularization in neural networks
- Since a single regularization parameter is applied on all weights, its very important to bring all weights on to same scale
- Regularization parameter might not have any impact if the data is not standardised.

$$New\ Regularized\ error = \sum_{i=1}^{n}\left[y_i - g\left(\sum_{k=1}^{m} w_k h_{ki}\right)\right]^2 + \frac{1}{2}\lambda\sum_{i=1}^{n} w_i^2$$

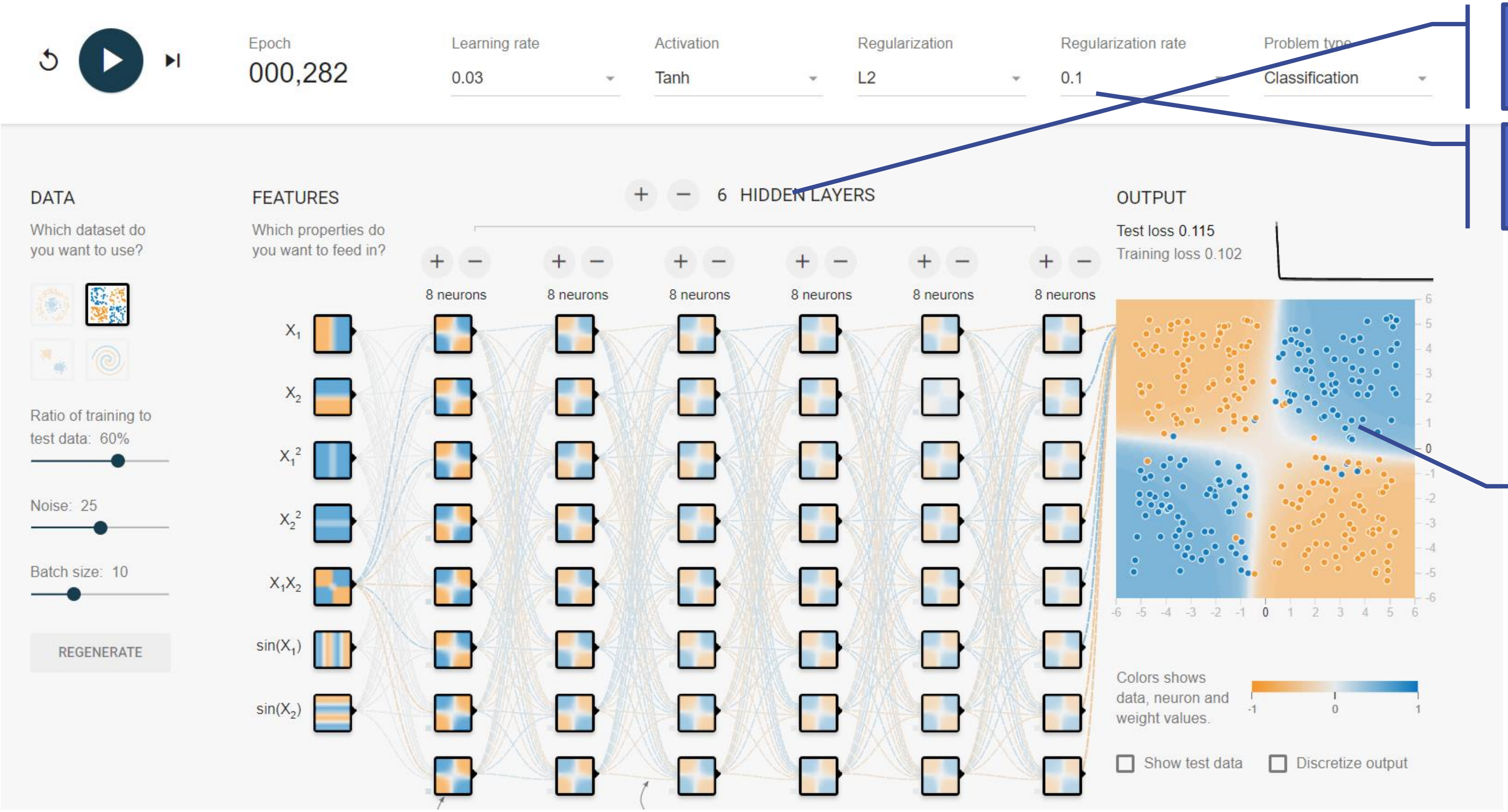
Demo: Regularization

playground.tensorflow.org



- Lot of hidden layers
- No Regularization
- Overfitted model

Demo: Regularization



- Lot of hidden layers
- With Regularization

Model is not overfitted

Lab: Regularization in Neural Nets

- Import Credit risk data
- The data has class-imbalance problem. Prepare balance sample for the model building
- Standardise the data
- Try to build a neural network with 15 hidden nodes
- Set the decay parameter as 0.5

Lab: Regularization in Neural Nets

```
library(clusterSim)
risk_train_strd<-data.Normalization (risk_train[,-1],type="n1",normalization="column")
head(risk_train_strd)
risk_train_strd$SeriousDlqin2yrs<-risk_train$SeriousDlqin2yrs

# x vector, matrix or dataset type ;type of normalization: n0 - without normalization
# n1 - standardization ((x-mean)/sd)
# n2 - positional standardization ((x-median)/mad)
# n3 - unitization ((x-mean)/range)
```

Normalize the data

Lab: Regularization in Neural Nets

```
library(nnet)
set.seed(35)
mod1<-nnet(as.factor(SeriousDlqin2yrs)~., data=risk_train,
           size=15,
           maxit=500)

####Results and Intime validation
actual_values<-risk_train$SeriousDlqin2yrs
Predicted<-predict(mod1, type="class")
cm<-table(actual_values,Predicted)
cm
acc<-(cm[1,1]+cm[2,2])/(cm[1,1]+cm[1,2]+cm[2,1]+cm[2,2])
acc

####Results on test data
actual_values_test<-risk_test$SeriousDlqin2yrs
Predicted_test<-predict(mod1, risk_test[,-1], type="class")
cm_test<-table(actual_values_test,Predicted_test)
cm_test
acc_test<-(cm_test[1,1]+cm_test[2,2])/(cm_test[1,1]+cm_test[1,2]+cm_test[2,1]+cm_test[2,2])
acc_test
```

Lab: Regularization in Neural Nets

```
library(nnet)
set.seed(35)
mod1<-nnet(as.factor(SeriousDlqin2yrs)~., data=risk_train,
           size=15,
           maxit=500,
           decay = 0.5)

####Results and Intime validation
actual_values<-risk_train$SeriousDlqin2yrs
Predicted<-predict(mod1, type="class")
cm<-table(actual_values,Predicted)
cm
acc<-(cm[1,1]+cm[2,2])/(cm[1,1]+cm[1,2]+cm[2,1]+cm[2,2])
acc

####Results on test data
actual_values_test<-risk_test$SeriousDlqin2yrs
Predicted_test<-predict(mod1, risk_test[,-1], type="class")
cm_test<-table(actual_values_test,Predicted_test)
cm_test
acc_test<-(cm_test[1,1]+cm_test[2,2])/(cm_test[1,1]+cm_test[1,2]+cm_test[2,1]+cm_test[2,2])
acc_test
```

Finetuning neural network models

Major Parameters for finetuning NN

- Two major parameters
  - Number of hidden nodes/ layers
  - The decay parameter
- If number of hidden nodes are high they decay should be high to regularize.
- If the decay is too high then the model might be underfitted.
- We need to choose an optimal pair of hidden nodes and decay

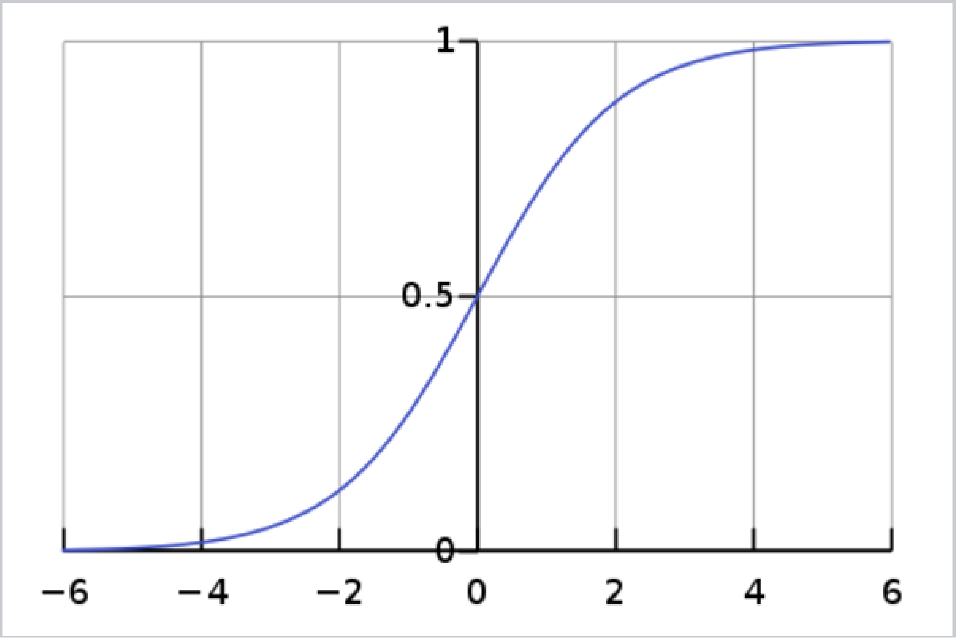
statinfer

Activation Functions

Activation Functions - Sigmoid

- Sigmoid function. Historically famous activation function
- Also known as Inverse logistic
- Works well for usual business related a data
- Works well for a binary or multi class output
- it takes a real-valued number and “squashes” and outputs number between 0 and 1.
- Large negative numbers become 0 and large positive numbers become 1.

$$S(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1}.$$



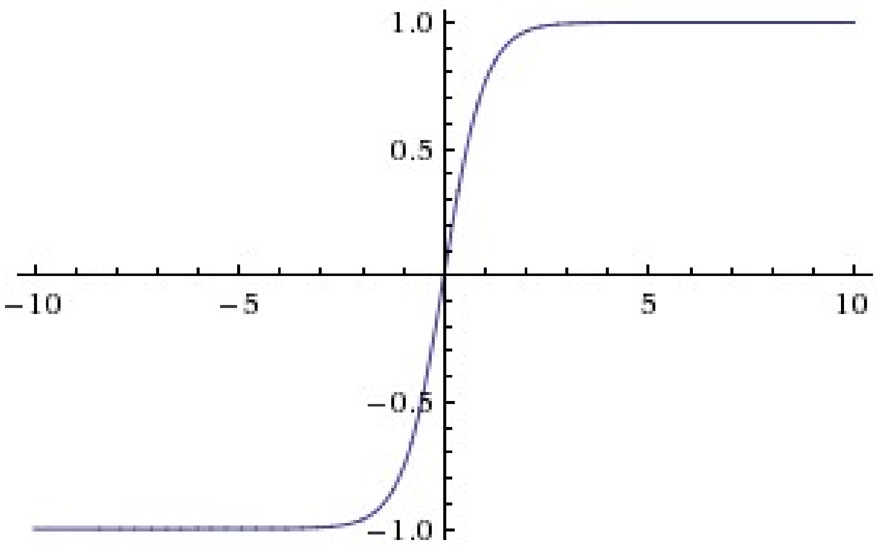
Sigmoid- Drawbacks

- Sigmoid output **not zero-centered.**
  - The middle value is 0.5
  - Most of the times we normalise the data(zero-centered) before building the neural network model.
- **Vanishing Gradients**
  - Computationally sigmoid values and gradient values and their multiplications are very small
  - For a deep network with many hidden layers sigmoid gradients vanish very quickly
- We need a different activation function which is zero-centered for better results.

New Activation Function- TanH

- The tanh similar to sigmoid
- Squashes the real values between -1 and +1
- The output is zero centered.
- Preferred activation function for zero centered (normalized) data.

$$g_{\tanh}(z) = \frac{\sinh(z)}{\cosh(z)} = \frac{e^z - e^{-z}}{e^z + e^{-z}}.$$

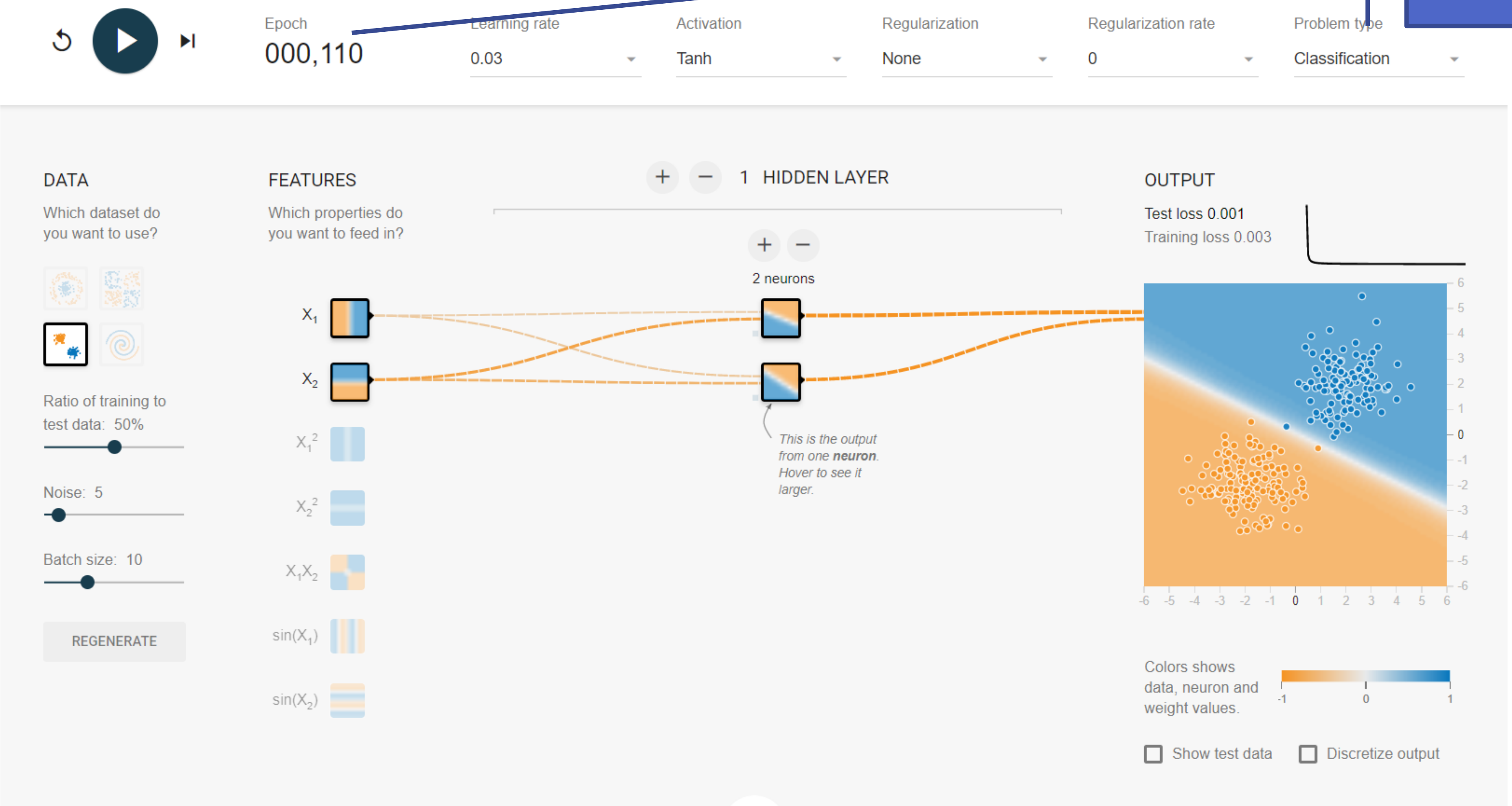


Demo: Sigmoid vs TanH

- Sigmoid takes 275 steps

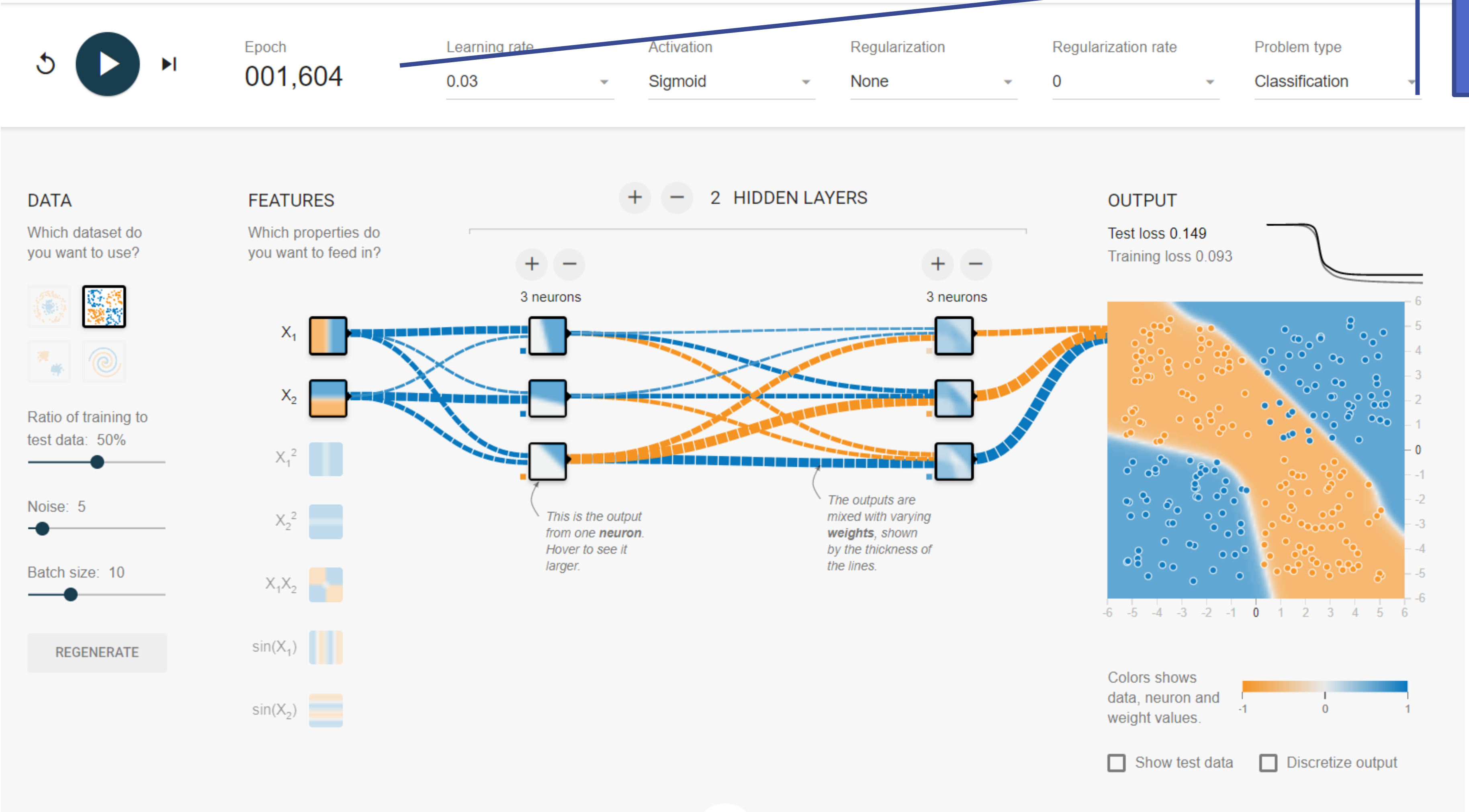
Demo: Sigmoid vs TanH

- TahH takes less steps

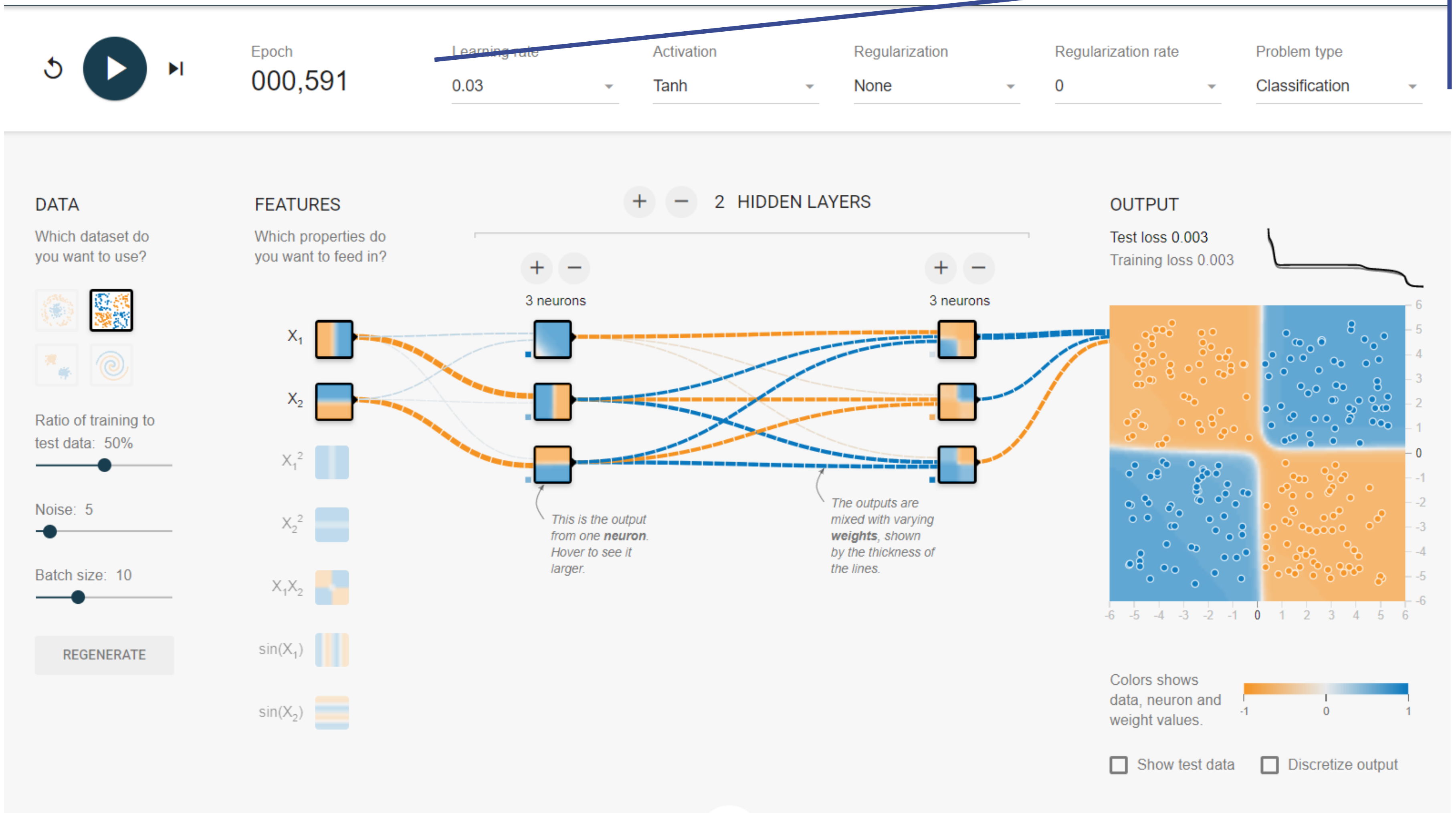


Demo: Sigmoid vs TanH

- Sigmoid takes 1,604 steps
- Error is still 15%



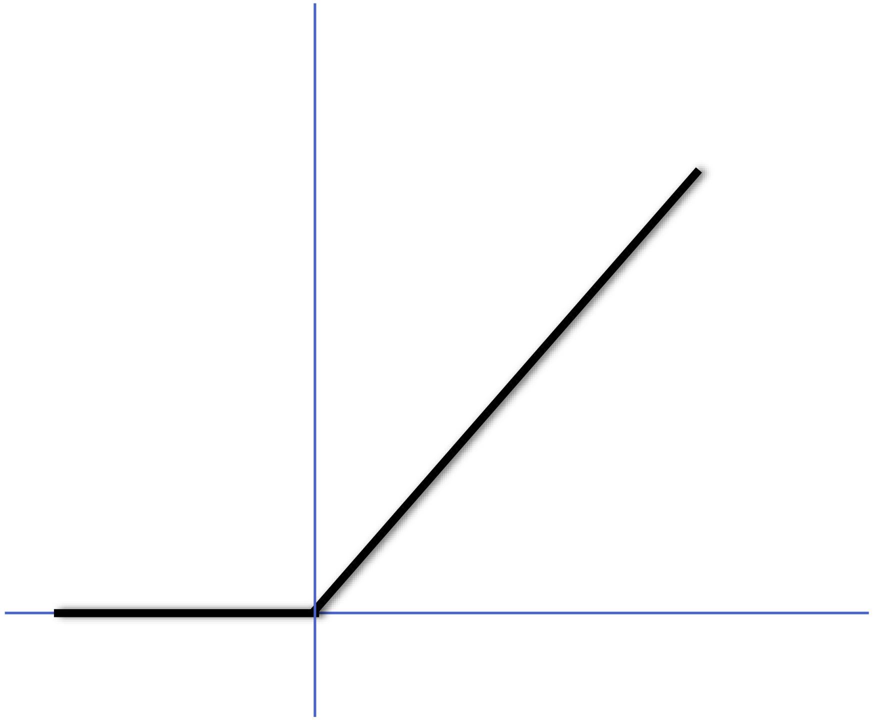
Demo: Sigmoid vs TanH



- TanH takes 600 steps
- Error is less than %

New Activation Function

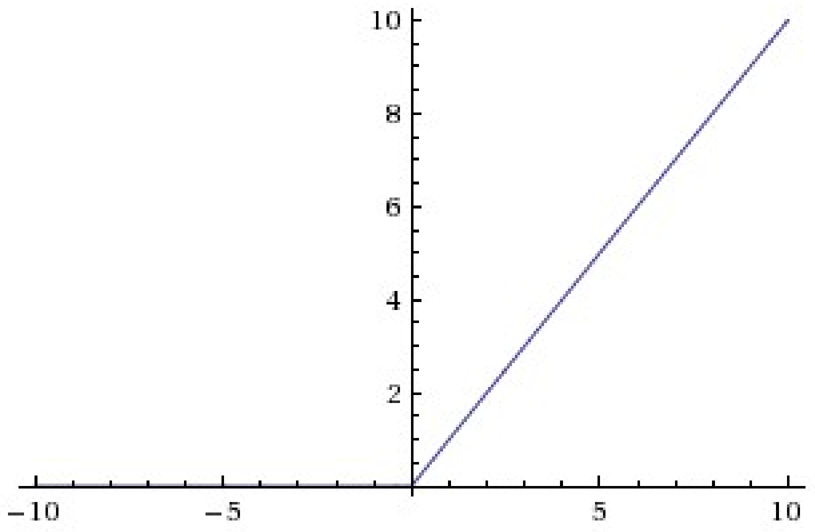
- What if the output is either zero or liner
  - Customer paid back the loan(zero loss) or loss of 1%,2%,…100%
  - Zero intensity(white/no-image) vs pixel intensity
- The above problem is neither classification nor regression.
  - We have a strong liner proton
  - We also have a considerable proportion of zeros
- Sigmoid and Tanh can work on this type of data but computationally very expensive.
  - For a deep network of image processing, computation and execution time is very critical
- We need a Rectified Linear unit type of activation function



ReLU - The Rectified Linear Unit

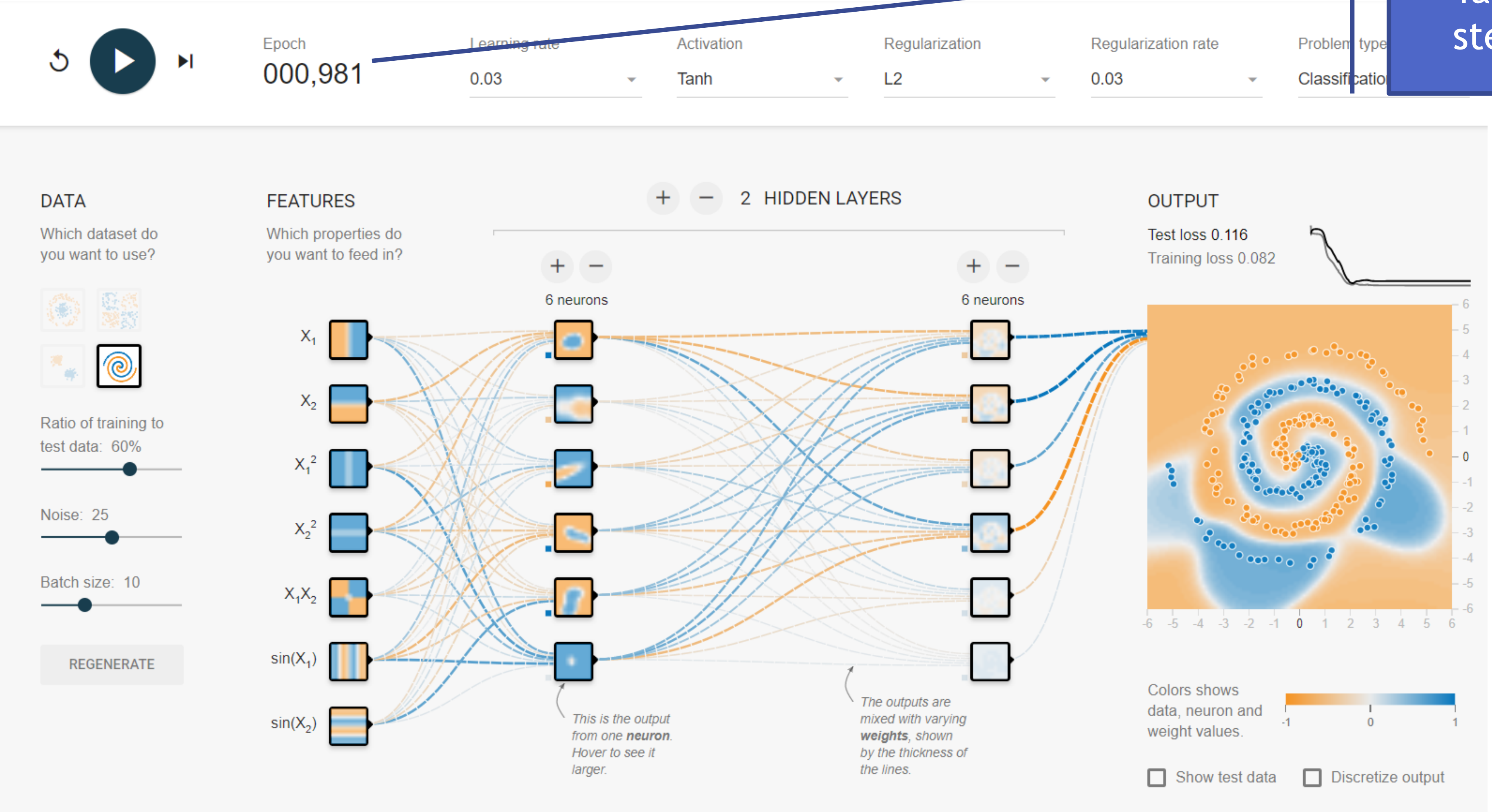
- Very popular activation function in recent times
- f(x)=max(0,x)
- In other words, the activation is simply thresholder at zero
- Very fast compared to sigmoid and tanH
- Works very well for a certain class of problems
- Doesn't have vanishing gradient problem. It can be used for modelling real values

f(x)=max(0,x)
f(x)=log(1+e^x)

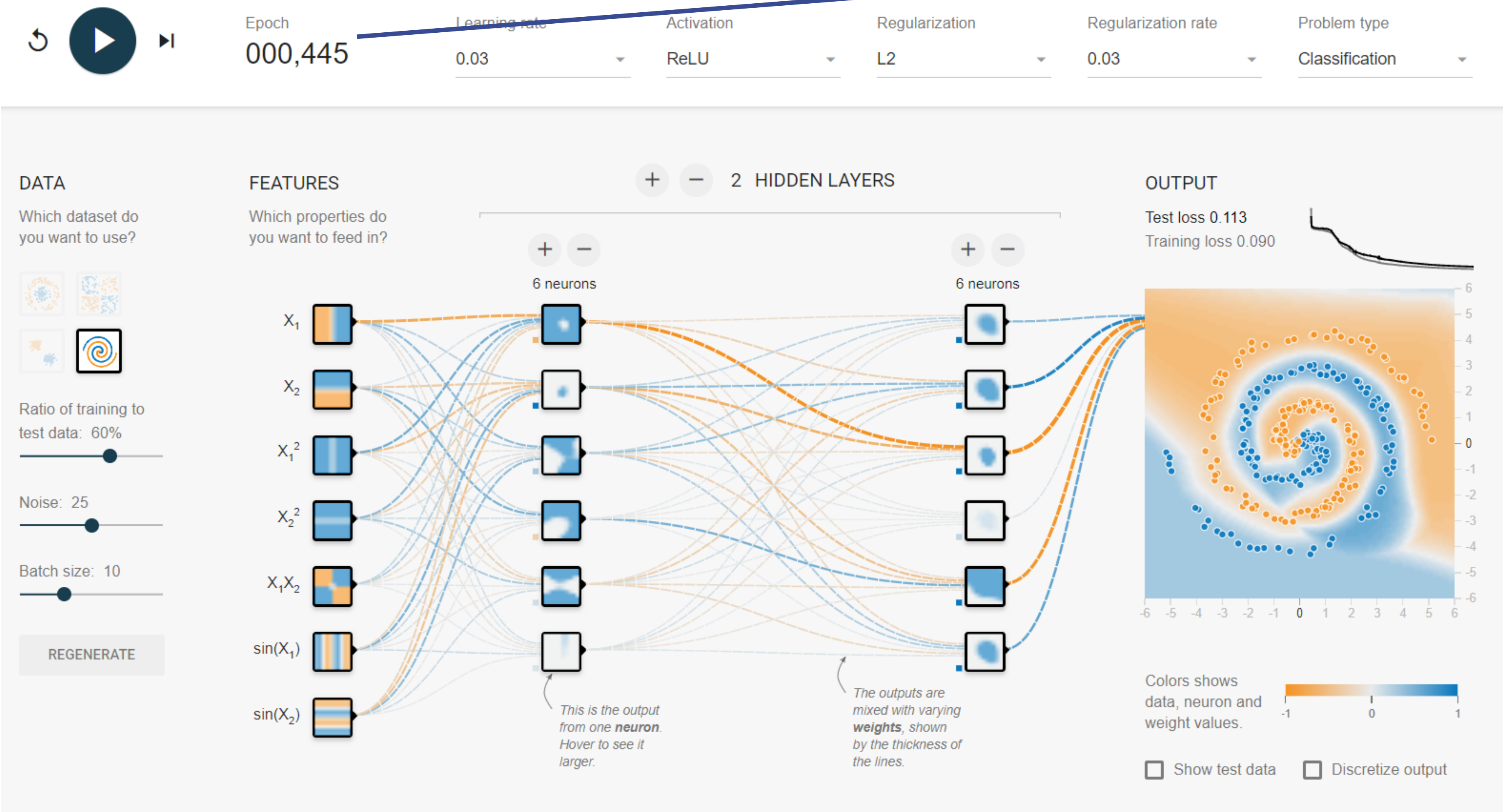


Demo: ThnH vs ReLu

- TanH takes 1,000 steps



Demo: ThnH vs ReLU



- ReLU takes less steps

Learning rate

Learning rate

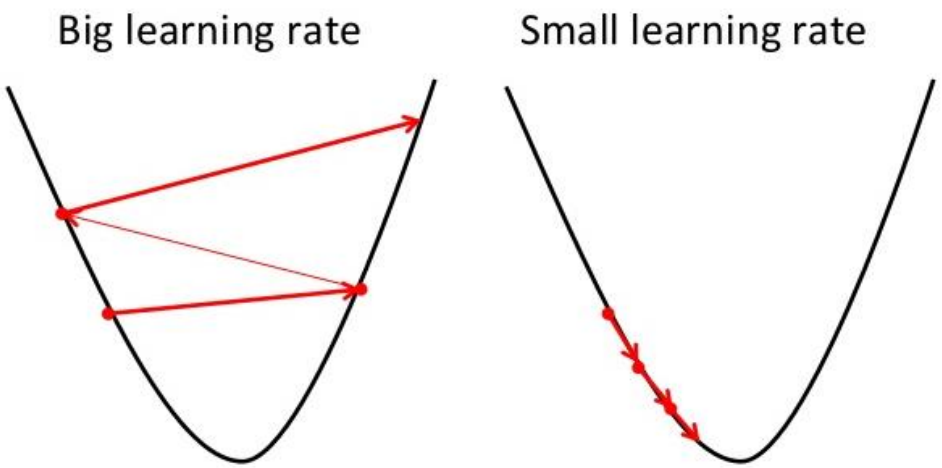
- The weight updating in neural networks

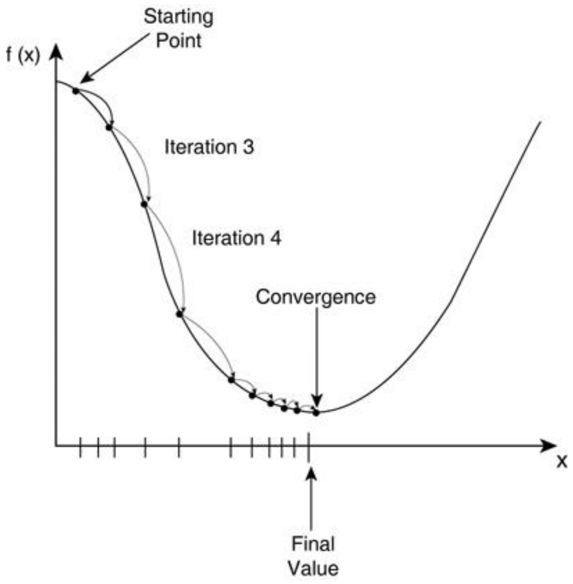
$$W_{jk} := W_{jk} + \Delta W_{jk}$$
$$where\ \Delta W_{jk} = \eta . y_j \delta_k$$
$$where\ \eta \text{ is the learning parameter}$$
$$W_{jk} := W_{jk} + \eta . y_j \delta_k$$

- Instead of just updating the weights based on actual calculations, we will manually setup weight update parameter η
- This will make the weights move by a factor η. How fast /slow you want to move the weights
- Read it as Step-size

Learning Rate tuning

- The speed at which the Neural Network arrives to minimum
- If the step-size is too high, the system will either oscillate about the true solution or it will diverge completely.
- If the step-size is too low, the system will take a long time to converge on the final solution.
- Generally start with a large value, reduce it gradually.

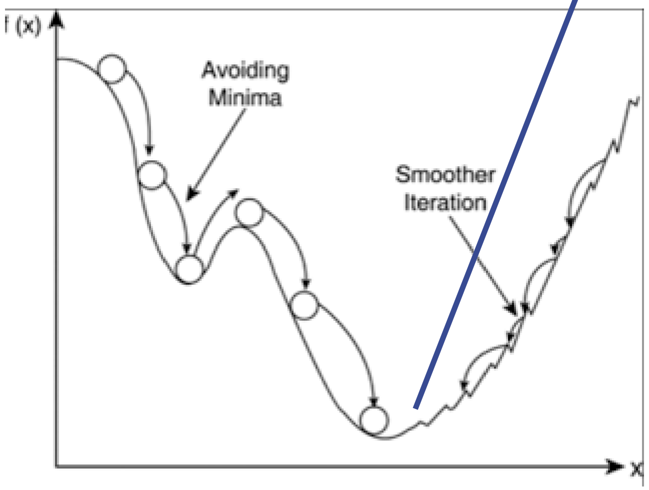




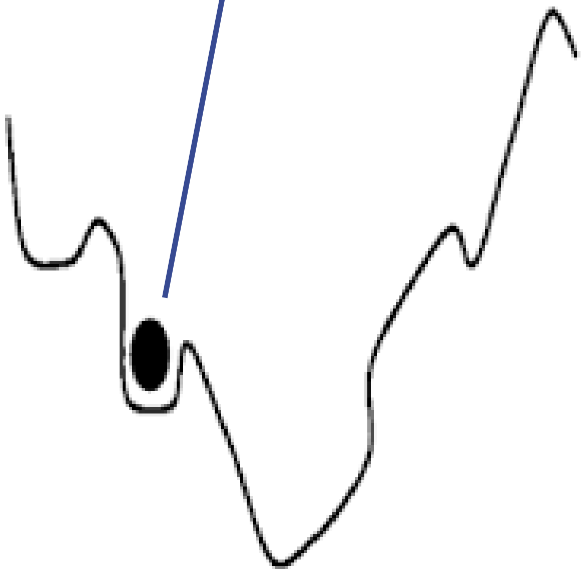
Learning Rate is Important

- For the actual practical problems the error surfaces are very complex
- Learning rate is very important to avoid local minima

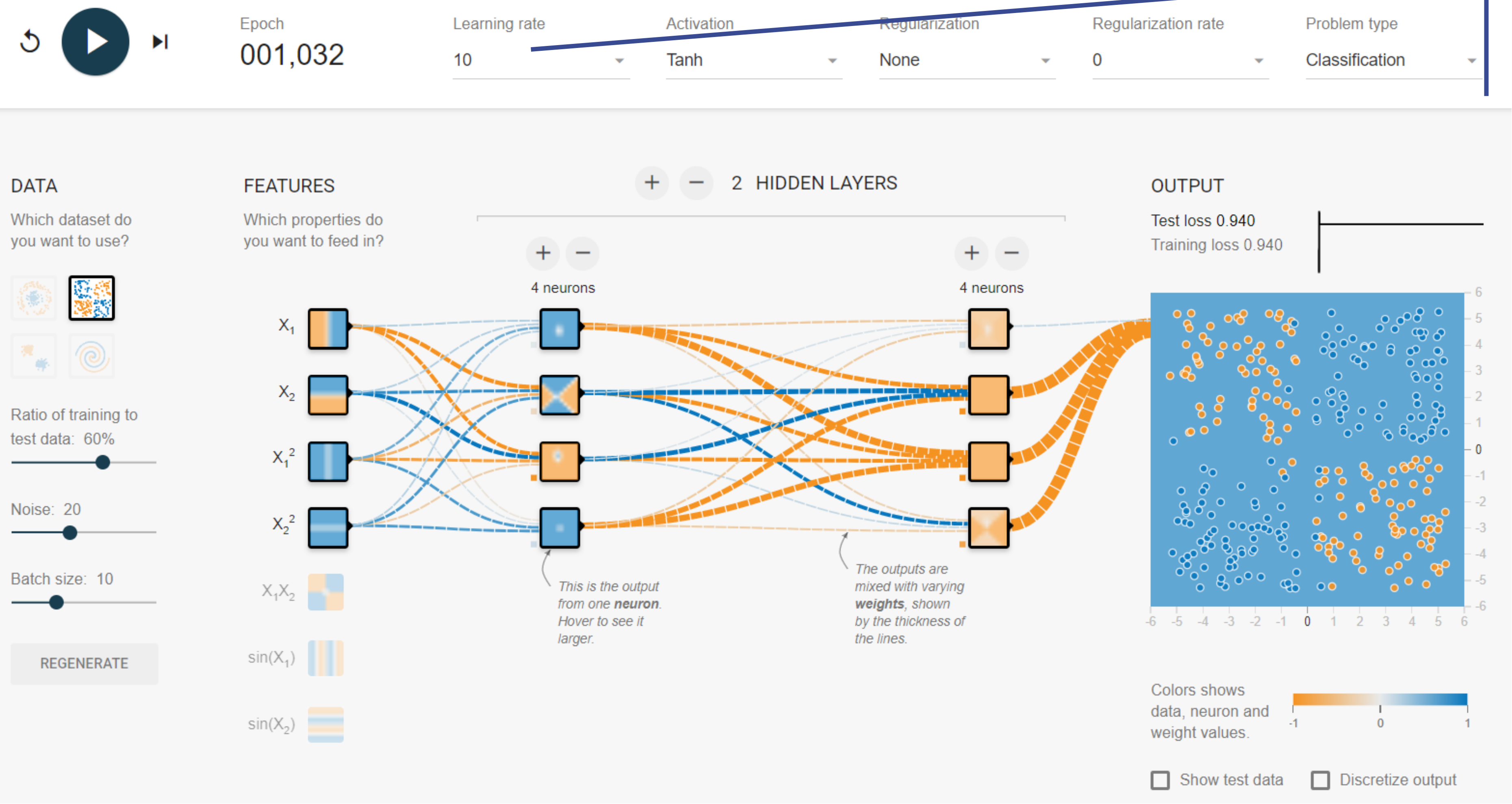
- Optimal learning rate
- Avoids local minima
- High chance of ending at global minima



- Small learning rate
- Small steps towards minima
- Has a risk of ending up in local minima



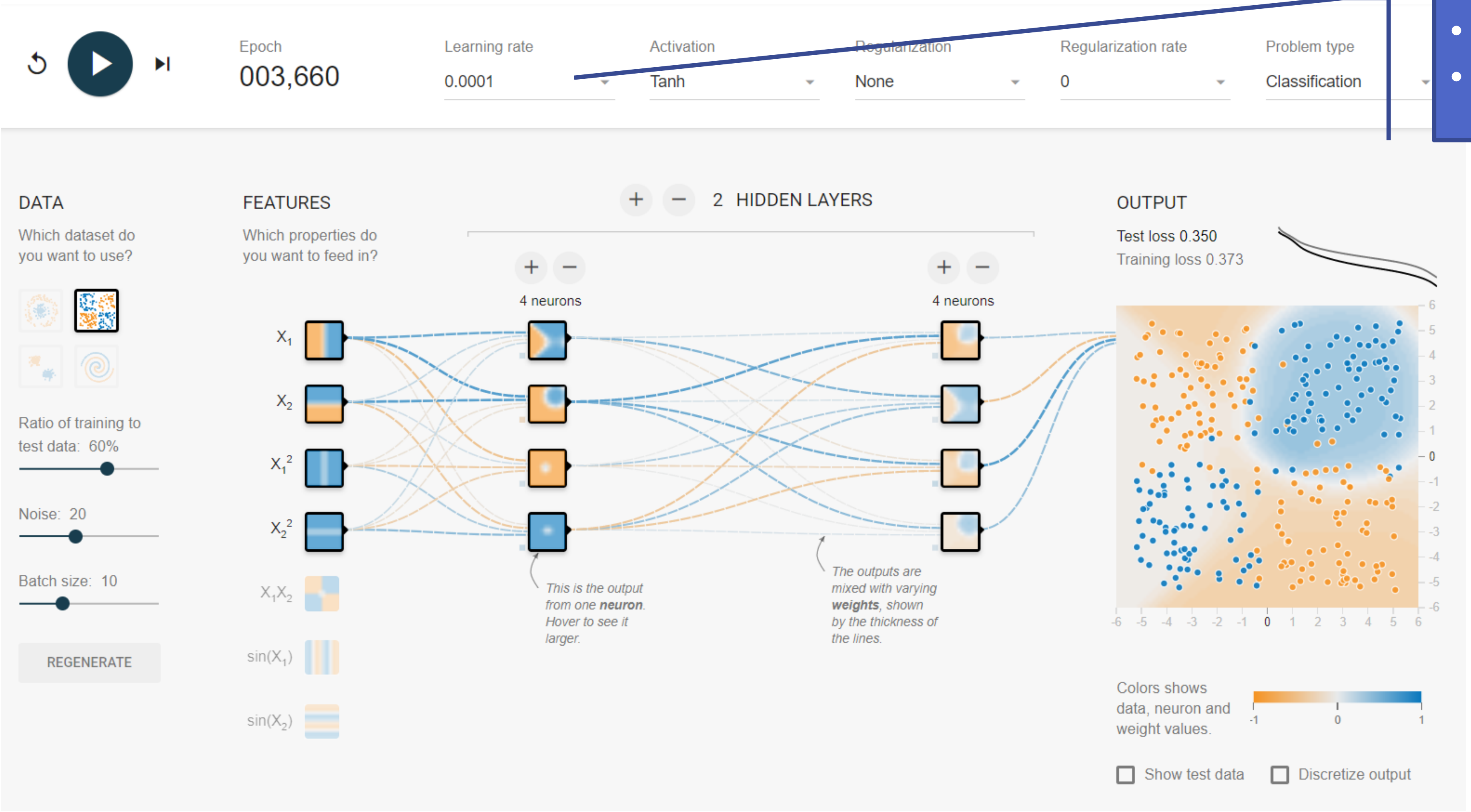
Demo: Learning rate



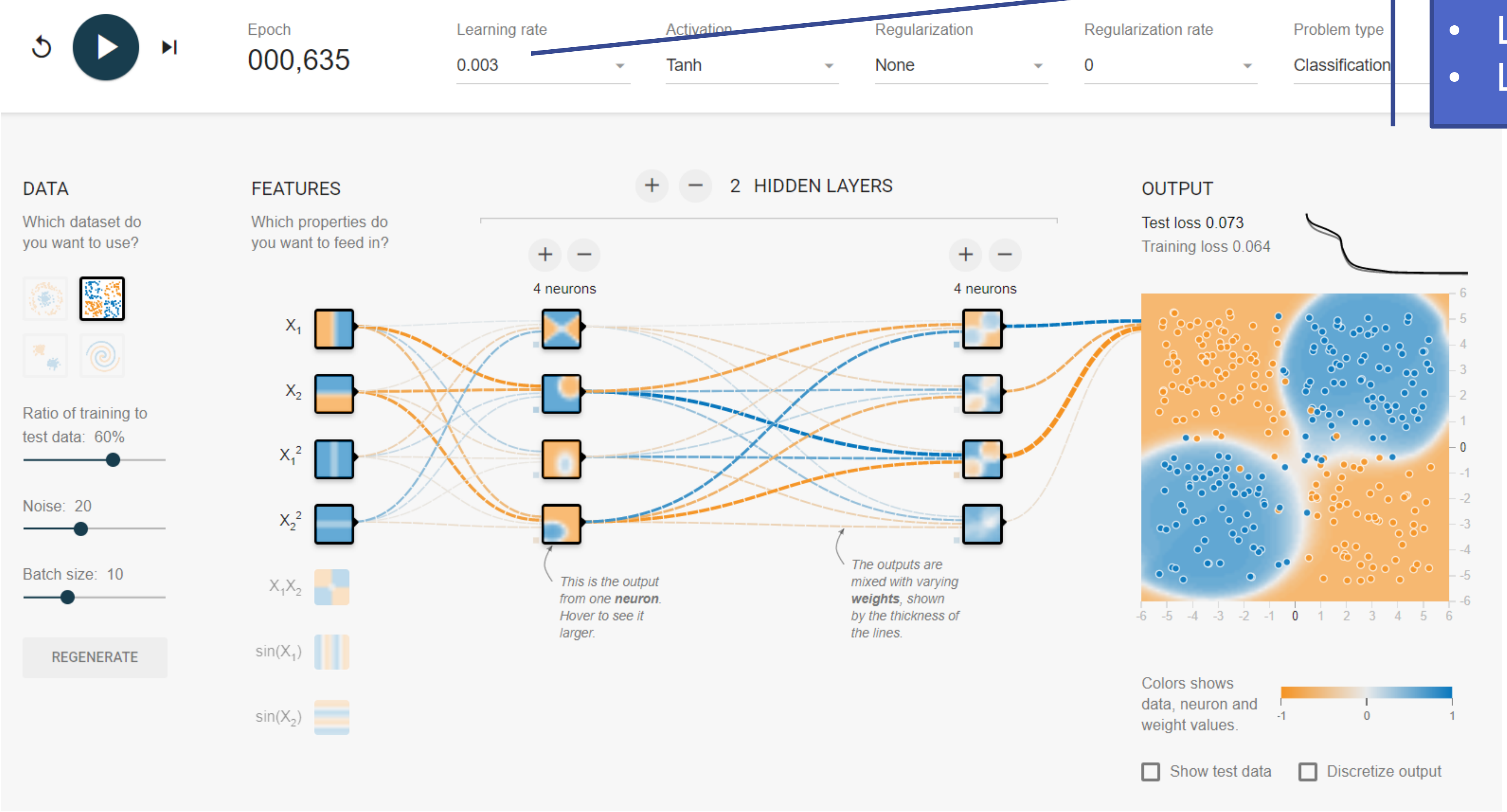
- High learning rate
- High error

Demo: Learning rate

- Very low learning rate
- Too many steps
- Takes a lot of time to converge



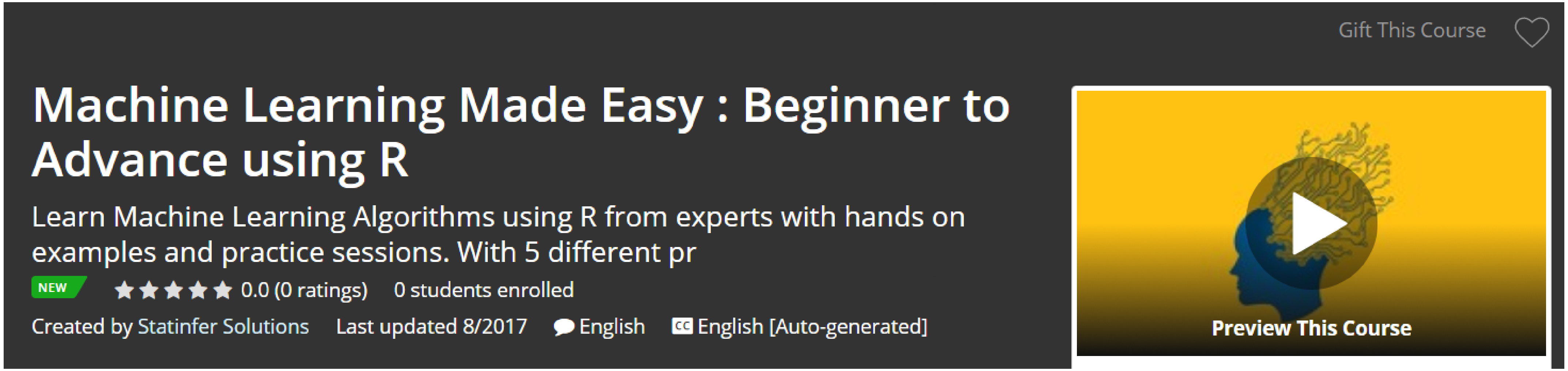
Demo: Learning rate



- Optimal learning rate
- Low error
- Less steps

Thank you

Our Course is Available on Udemy now



https://www.udemy.com/machine-learning-made-easy-beginner-to-advance-using-r/

Statinfer.com

Data Science Training and R&D

Training on

Data Science

Bigdata Analytics

Machine Learning

Predictive Modelling

Deep Learning

Corporate Training

Classroom Training

Online Training

Contact us

info@statinfer.com

venkat@statinfer.com