Deep Learning for Computer Vision

Lecture 7: Universal Approximation Theorem, More Hidden Units, Multi-Class Classifiers, Softmax, and Regularization

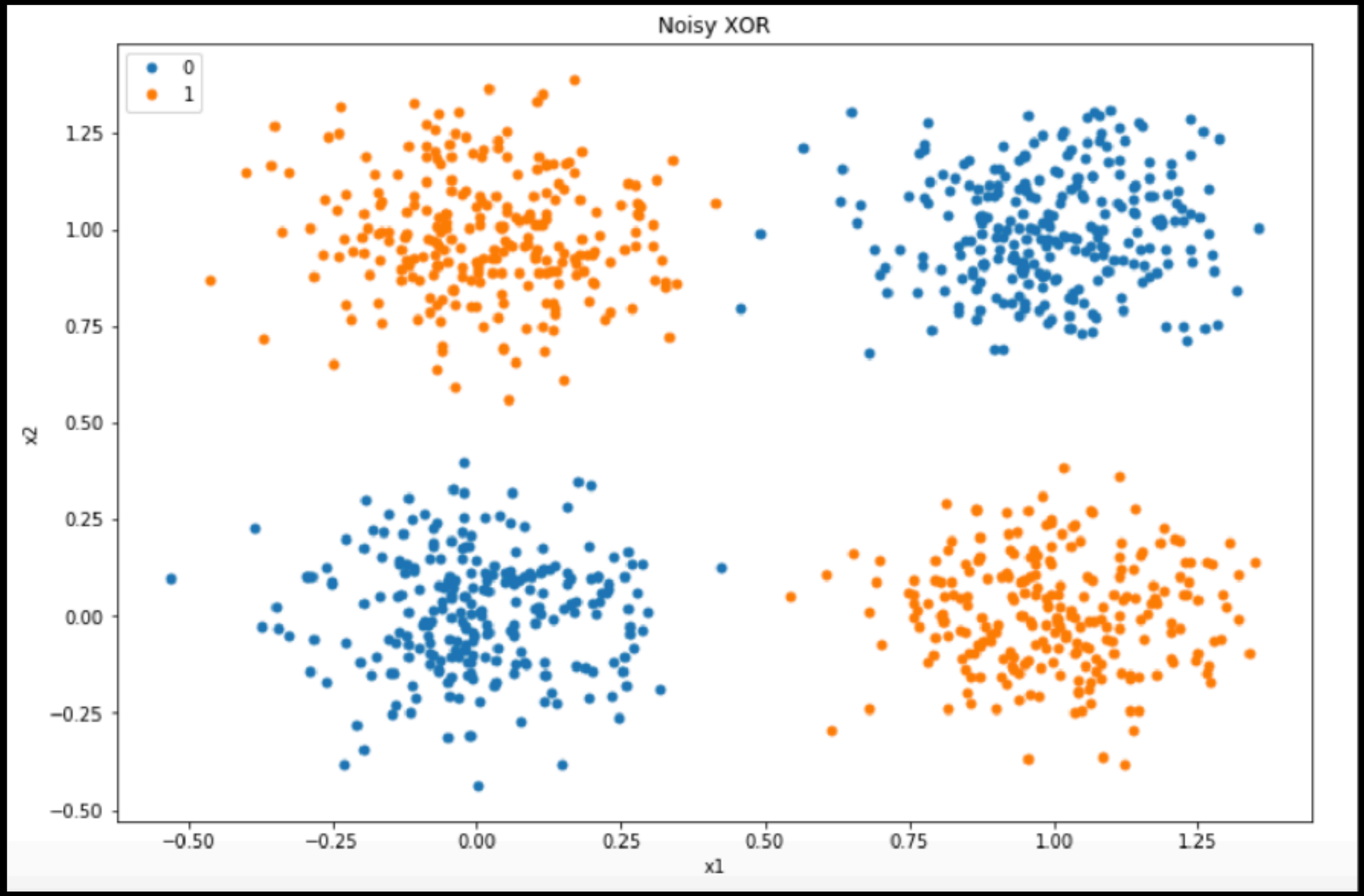
Peter Belhumeur

Computer Science
Columbia University

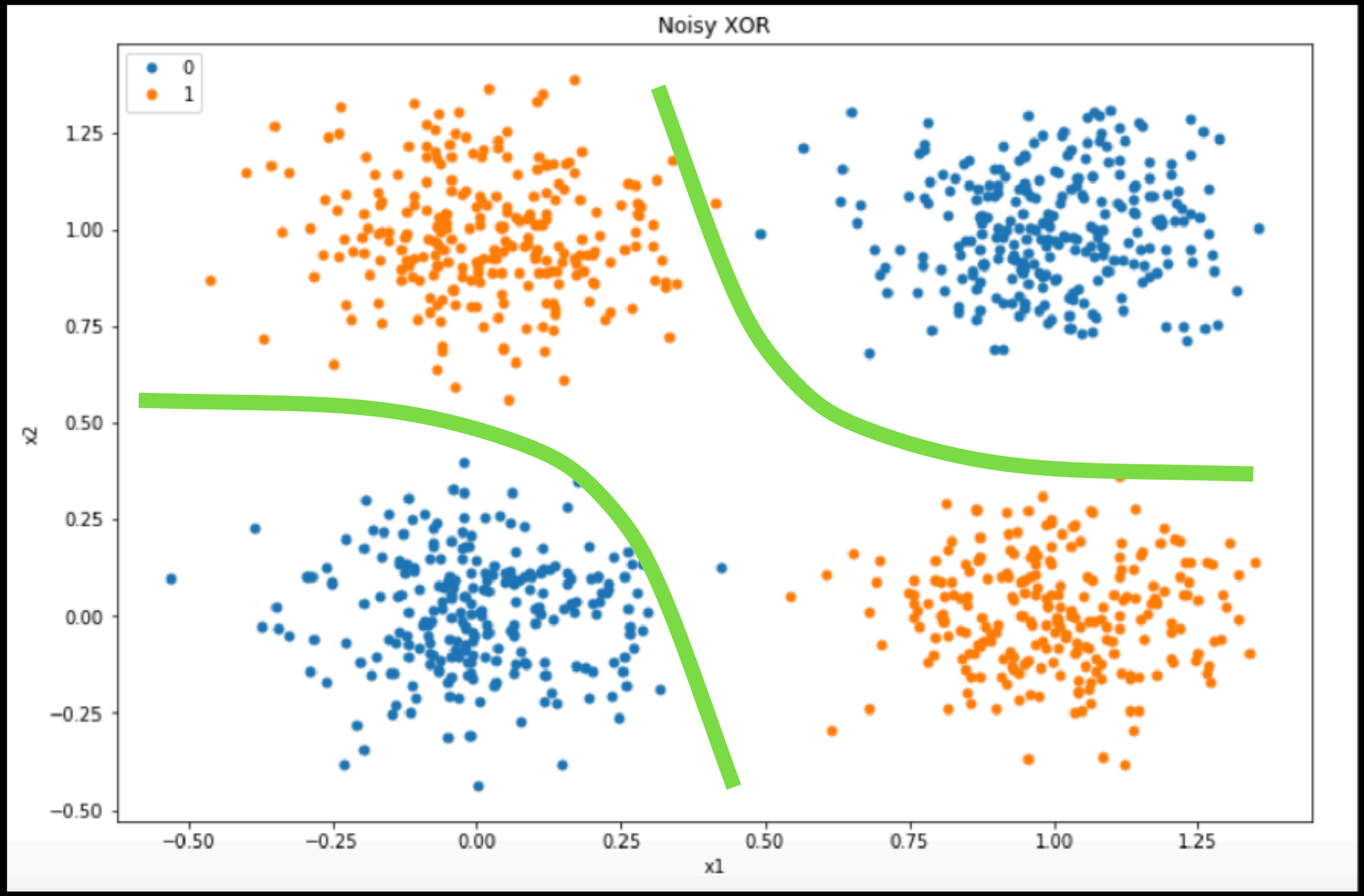
We saw last time that we were able to approximate a noisy XOR function using a MLP with one hidden layer.

But our network struggled to converge and was not great at carving up the crossing hourglass shape of the data.

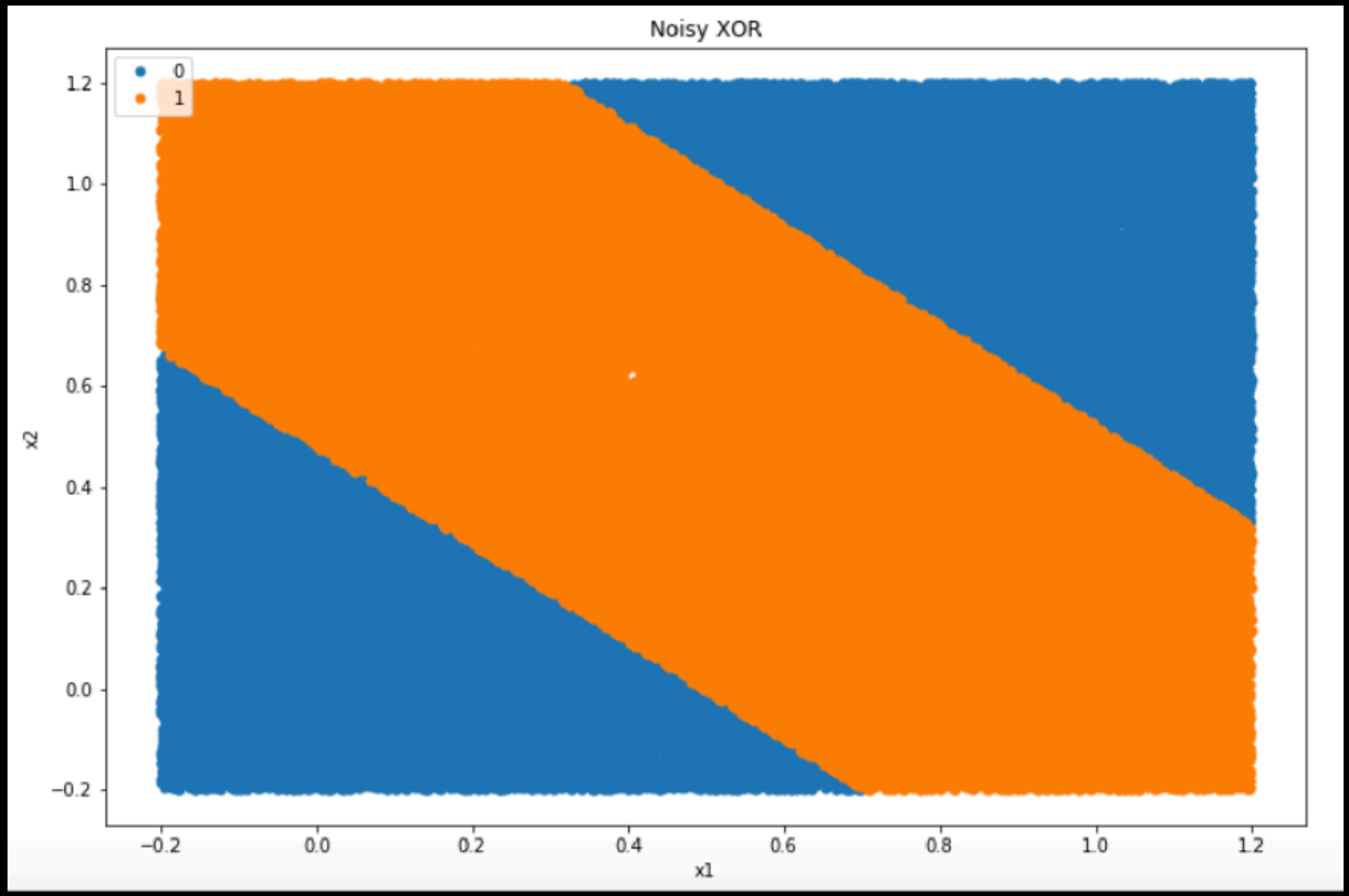
Noisy XOR Data



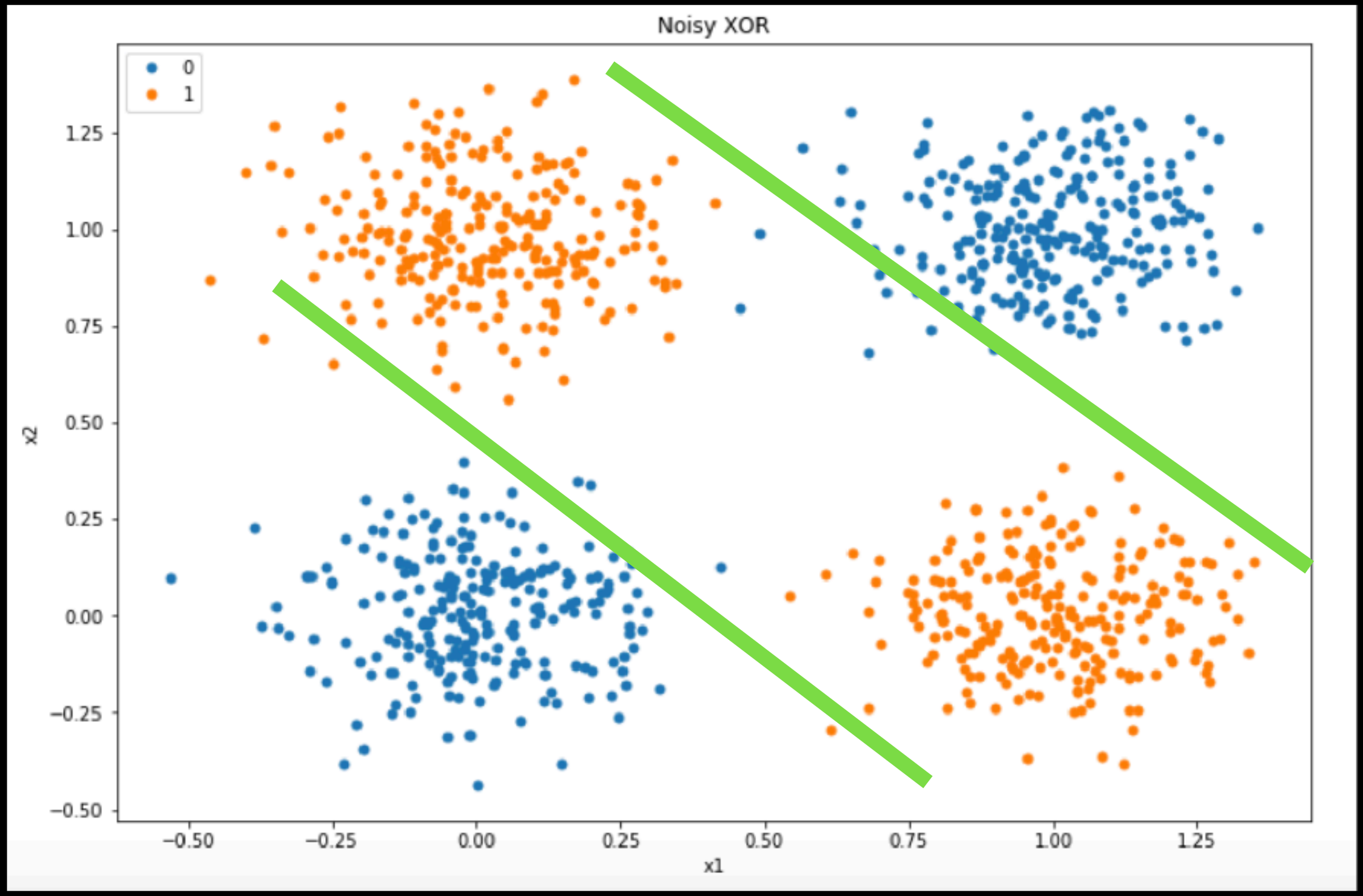
Nice fit might look like…



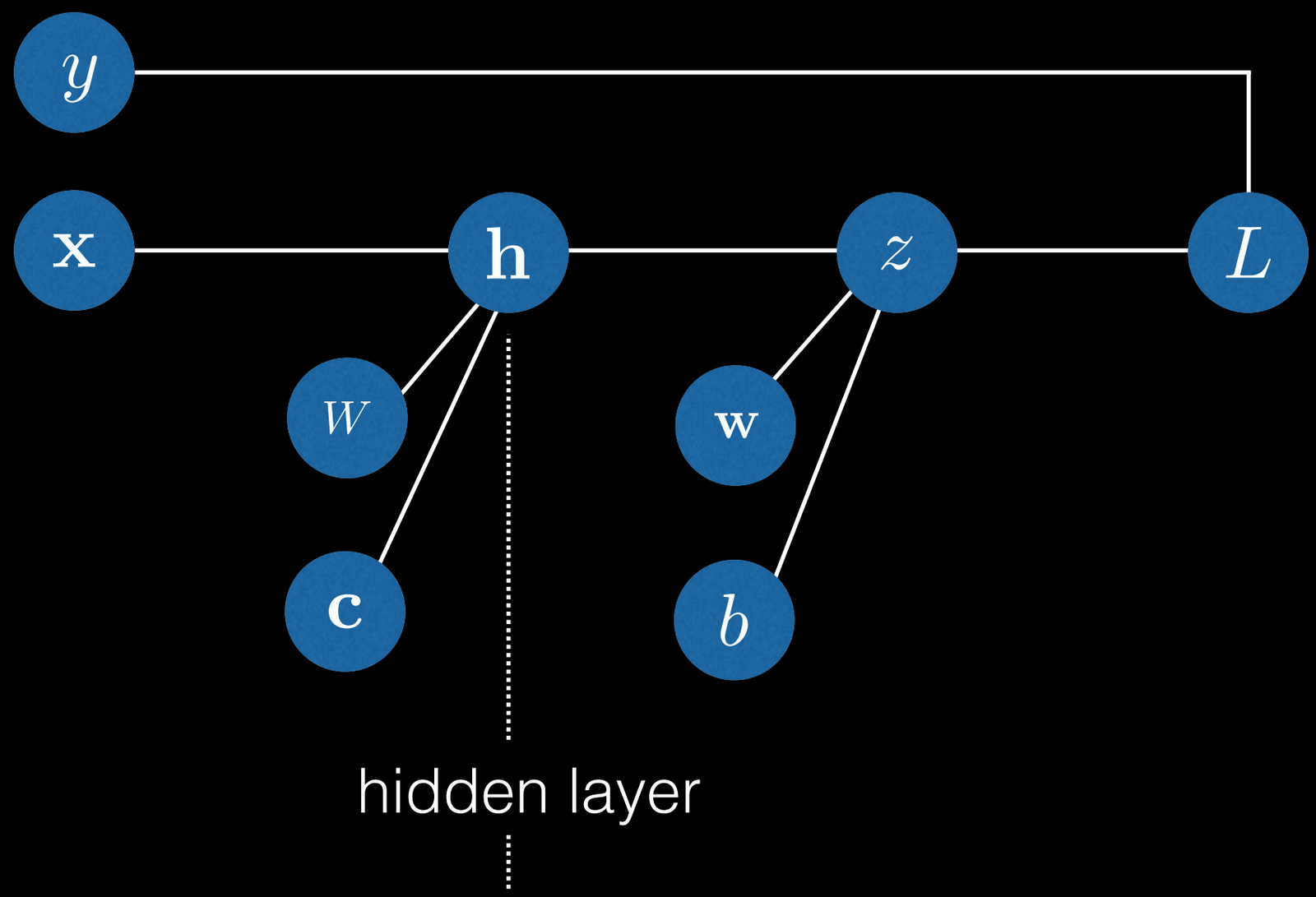
Here is what we did…



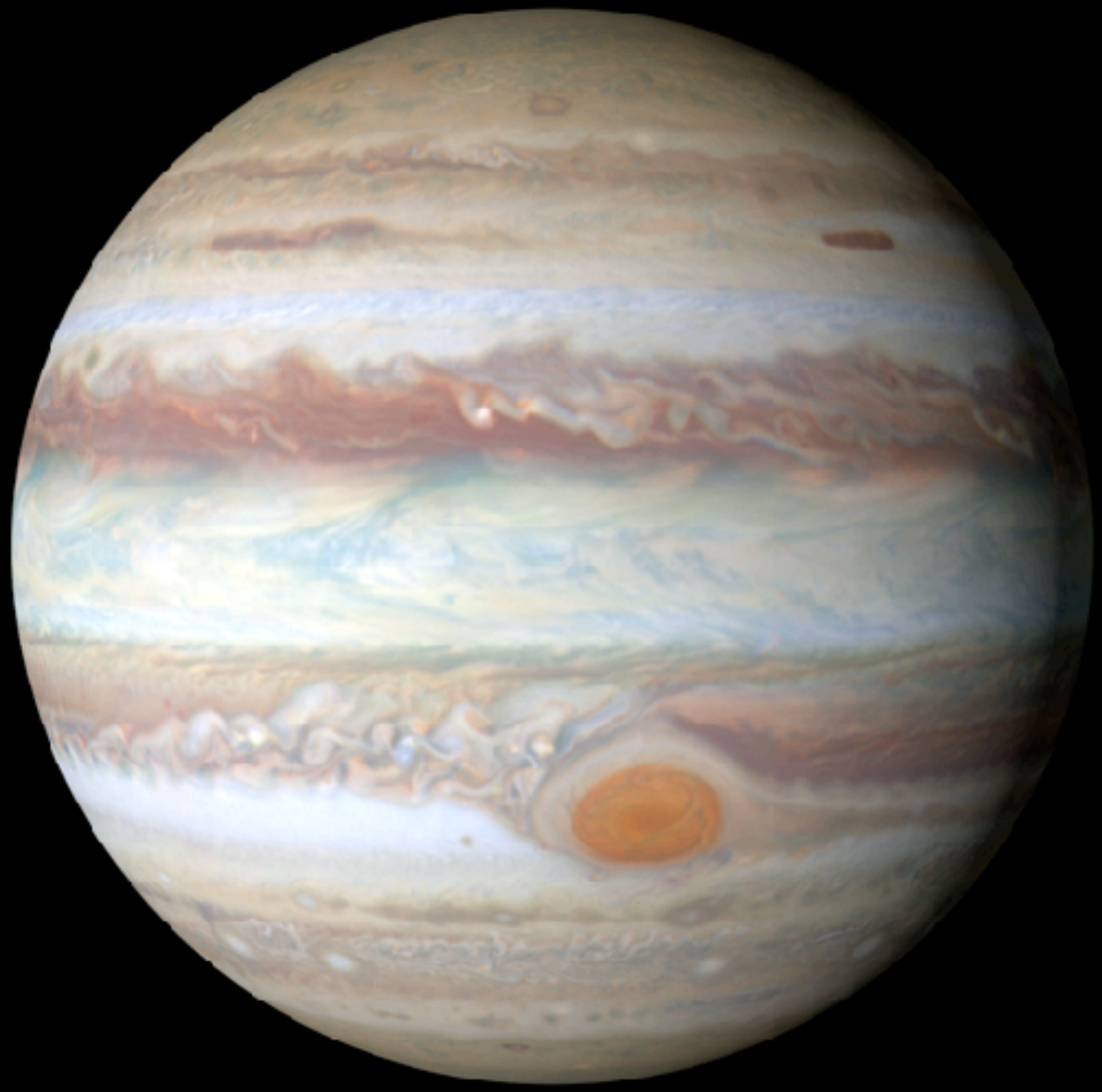
Eh…Could be better...



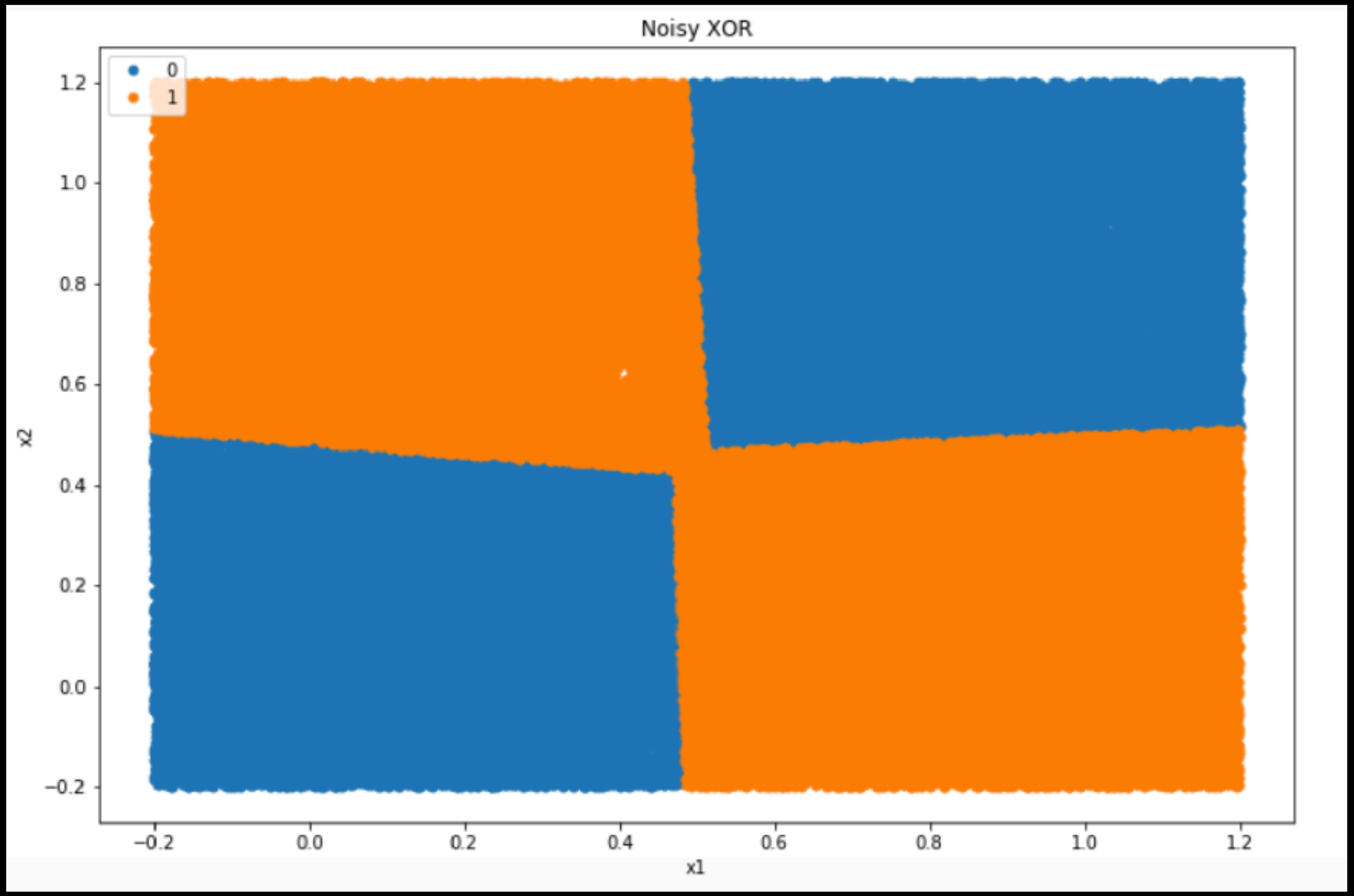
Simple MLP



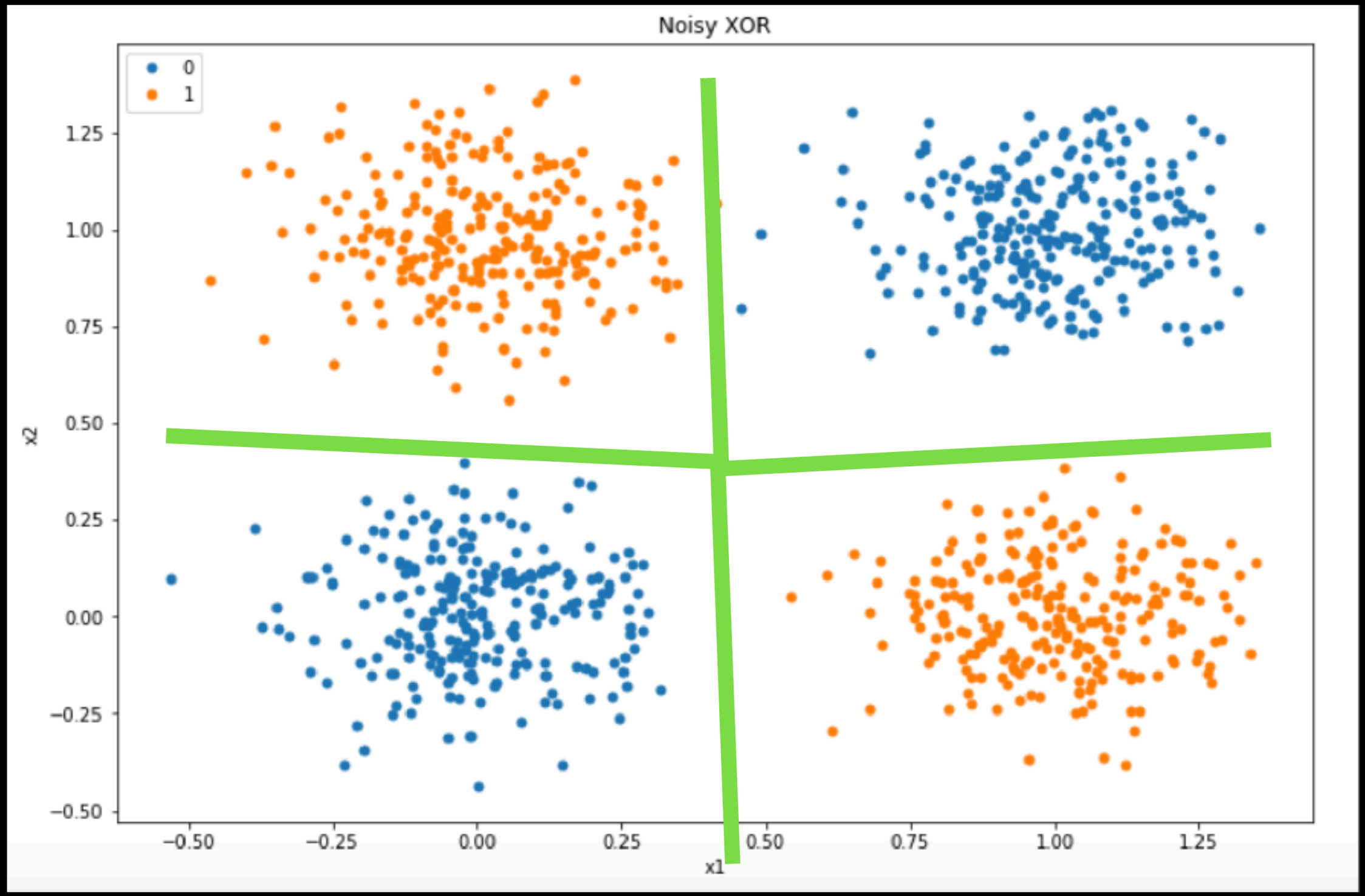
Let's **increase** the number of **hidden units** in this layer!



If we add more hidden units…



Done!


Noisy XOR
0
1
1.25
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
x2
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
1.25
x1

So what just happened?!

Universal Approximation Theorem

- tldr: if we have enough hidden units we can approximate "any" function! … but we may not be able to train it.
- **Universal Approximation Theorem**: A feedforward neural network with a linear output layer and one or more hidden layers with ReLU **[Leshno et al. '93]**, or sigmoid or some other "squashing" activation function **[Hornik et al. '89, Cybenko '89]** can approximate any continuous function on a closed and bounded subset of $\mathbb{R}^n$. This holds for functions mapping finite dimensional discrete spaces as well.

Universal Approximation Theorem: Caveats

- Optimization may fail to find the parameters needed represent the desired function.
- Training might choose the wrong function due to overfitting.
- The network required to approximate this function might be so large as to be infeasible.

Universal Approximation Theorem: Caveats

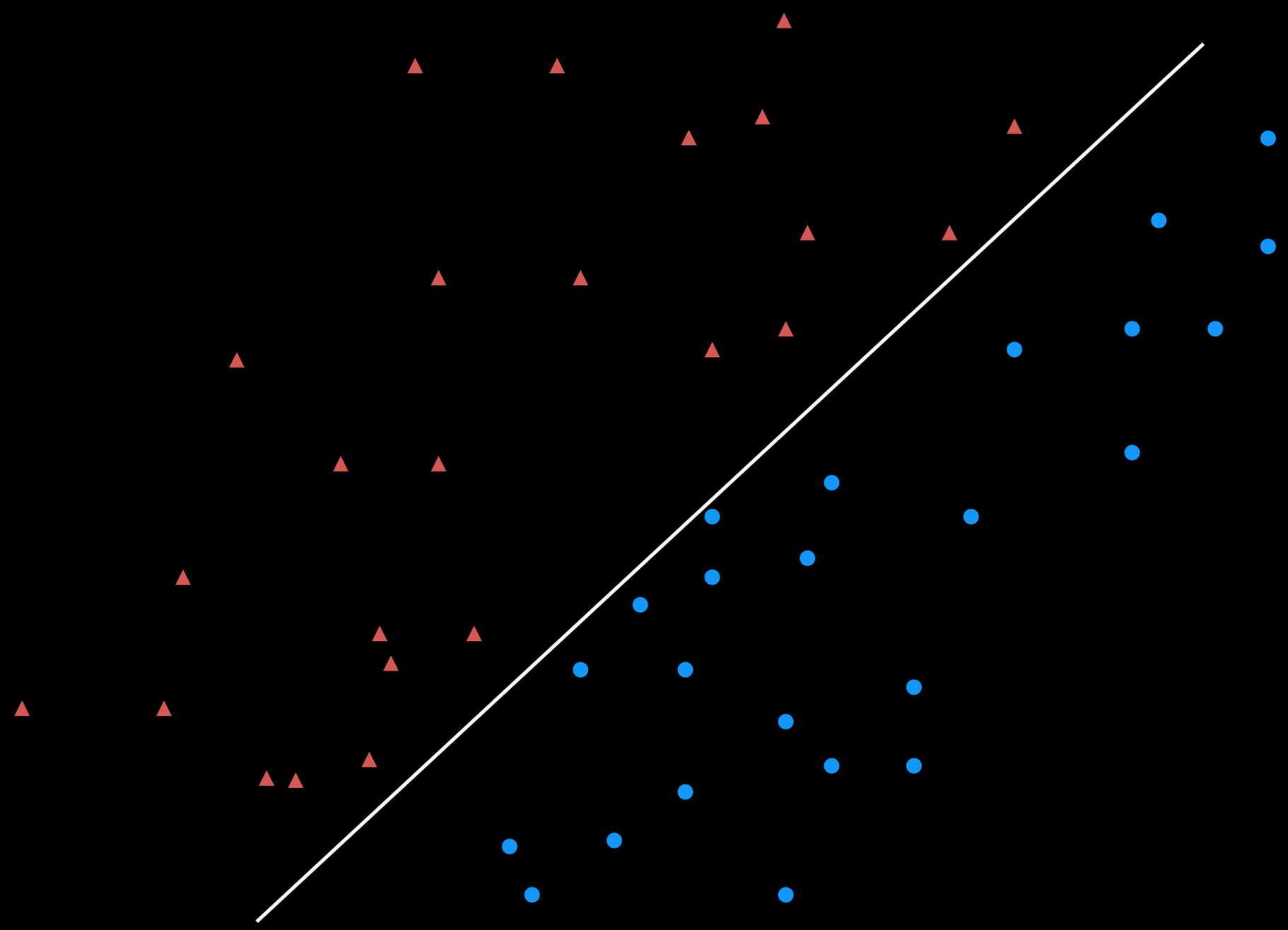
- So even though “any” function can be approximated with a network as described with single hidden layer, the network may fail to train, fail to generalize, or require so many hidden units as to be infeasible.
- This is both encouraging and discouraging!
- However, **[Montufar et al. 2014]** showed that **deeper networks are more efficient** in that a deep rectified net can represent functions that would require an exponential number of hidden units in a shallow one hidden layer network.
- Deep networks composed of many rectified hidden layers are good at approximating functions that can be composed from simpler functions. And lots of tasks such as image classification may fit nicely into this space.

Multi-Class Classification

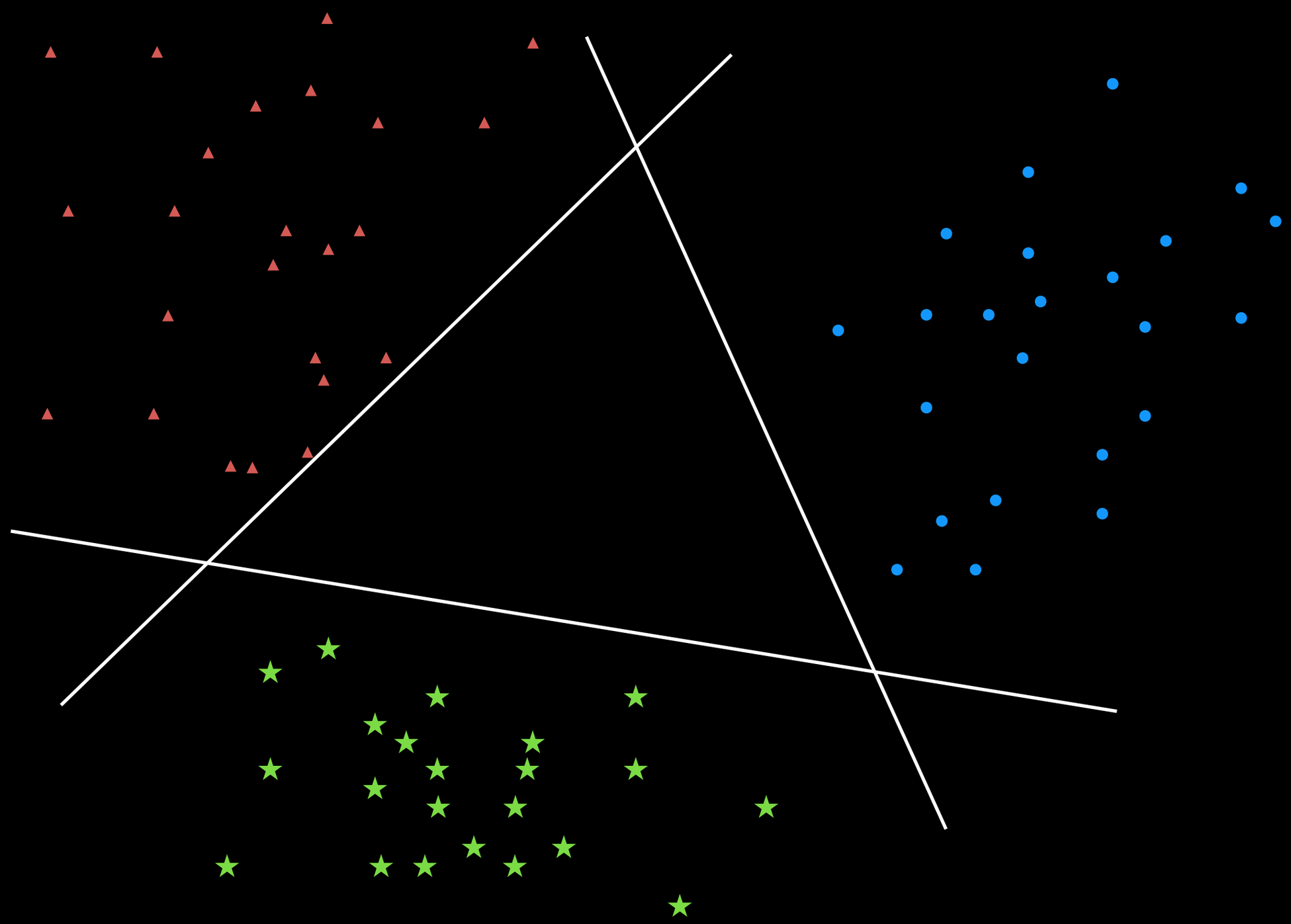
- So far we have only looked at binary classification problems for which we need to find a single separating surface or single discriminating function to predict the two classes from input data.

- What happens when we have *n* classes? In general you only need n - *1* discriminate functions. But in practice, it is simpler to use *n* and train *n* **one-vs-all** classifiers.

Binary Classification



One vs. All Classification



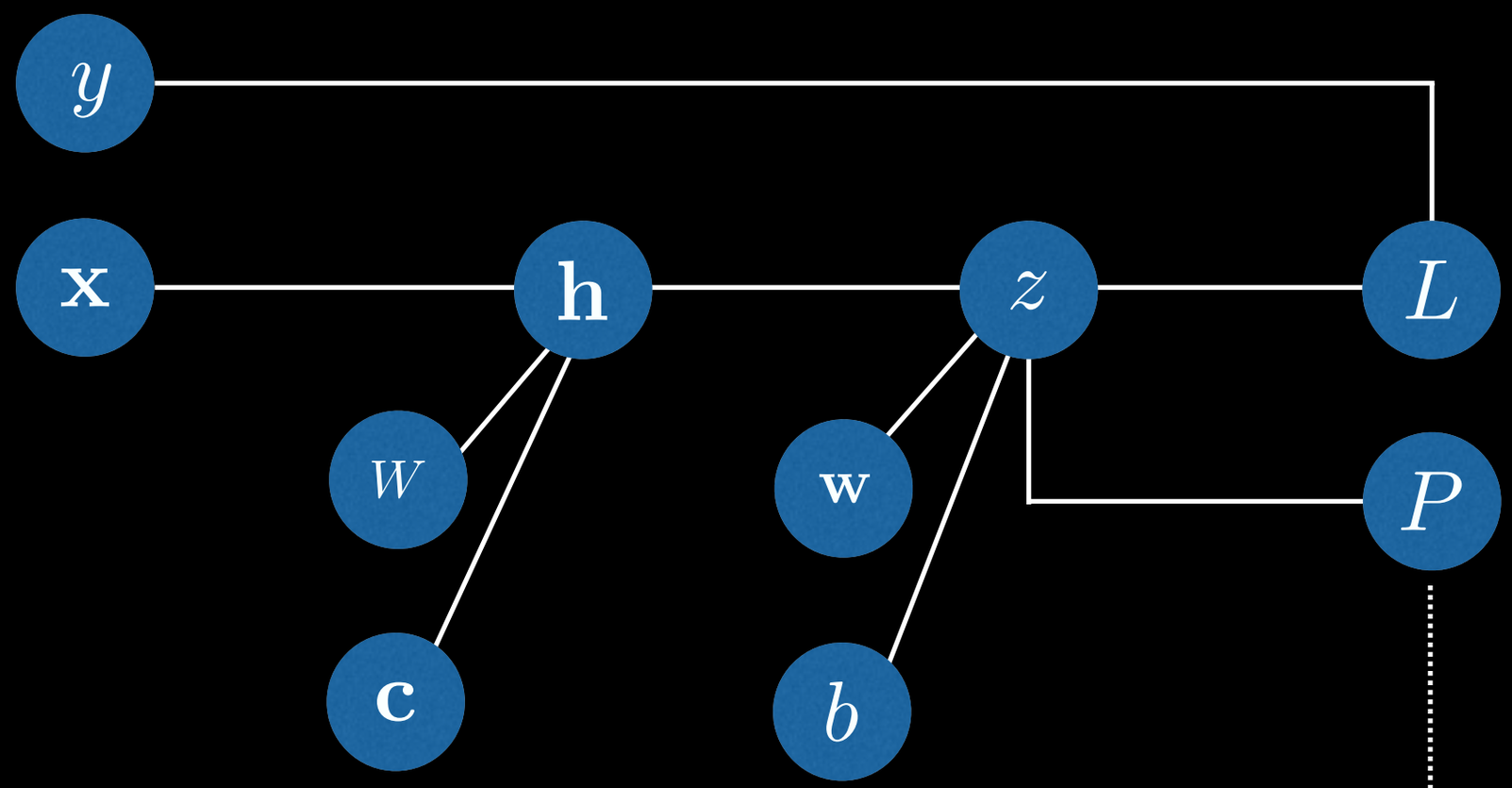
Multi-Class Classification

- In the binary case, we used a logistic sigmoid function to assign probabilities to the predictions: $P(y = 1|\mathbf{x})$

- In the multi-class case, we can generalize this using the **softmax** function to estimate $P(y = i|\mathbf{x})$

$$P(y = i|\mathbf{x}) = \frac{e^{z_i}}{\sum_j e^{z_j}}$$ Softmax

Recall Loss Layer in MLP in Binary Case

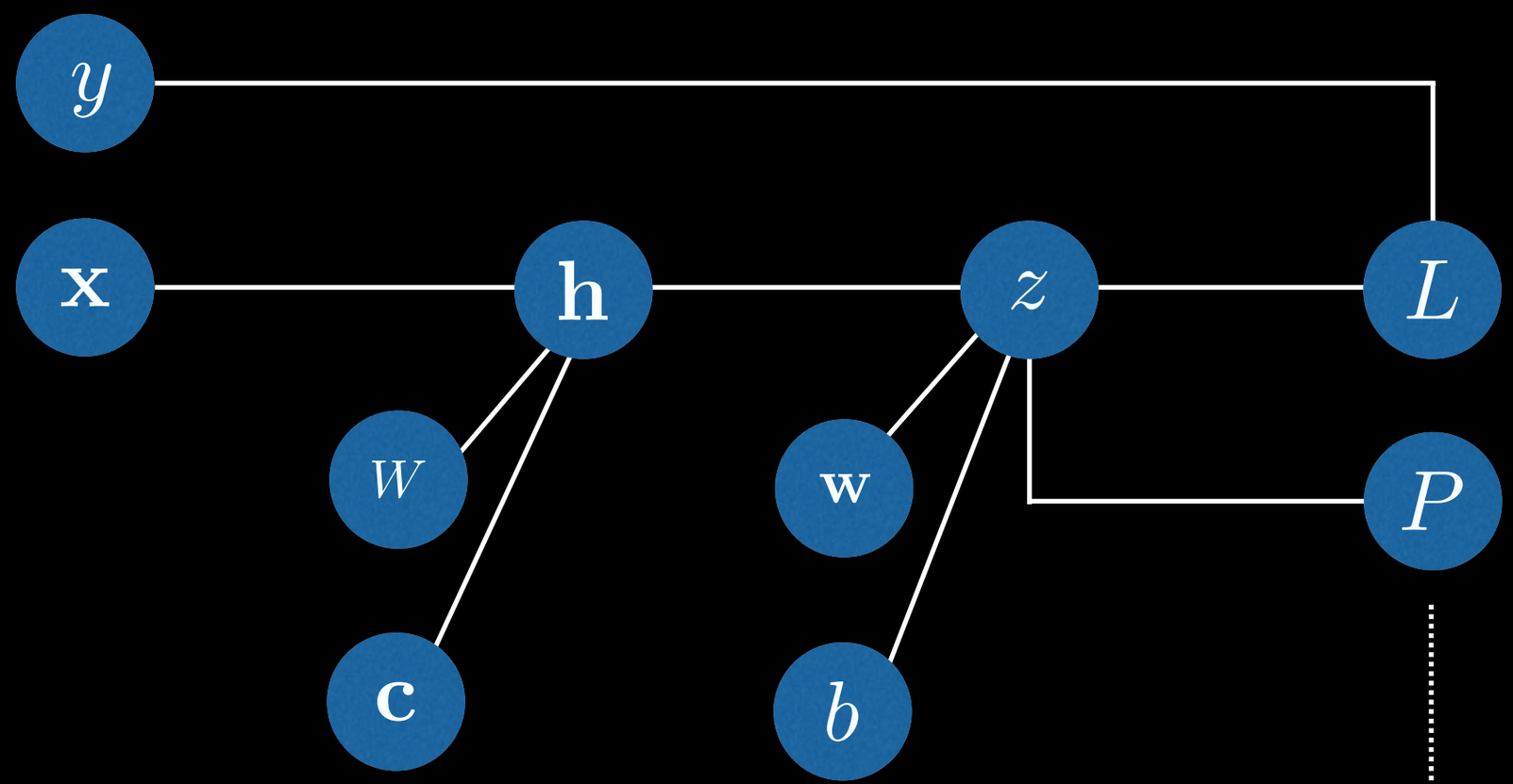


$$P = P(y = 1|\mathbf{x}) = \frac{1}{1 + e^{-z}}$$

$$L = -\log P(y|\mathbf{x}) = \xi((1 - 2y)z)$$

loss layer

Recall Loss Layer in MLP in Binary Case

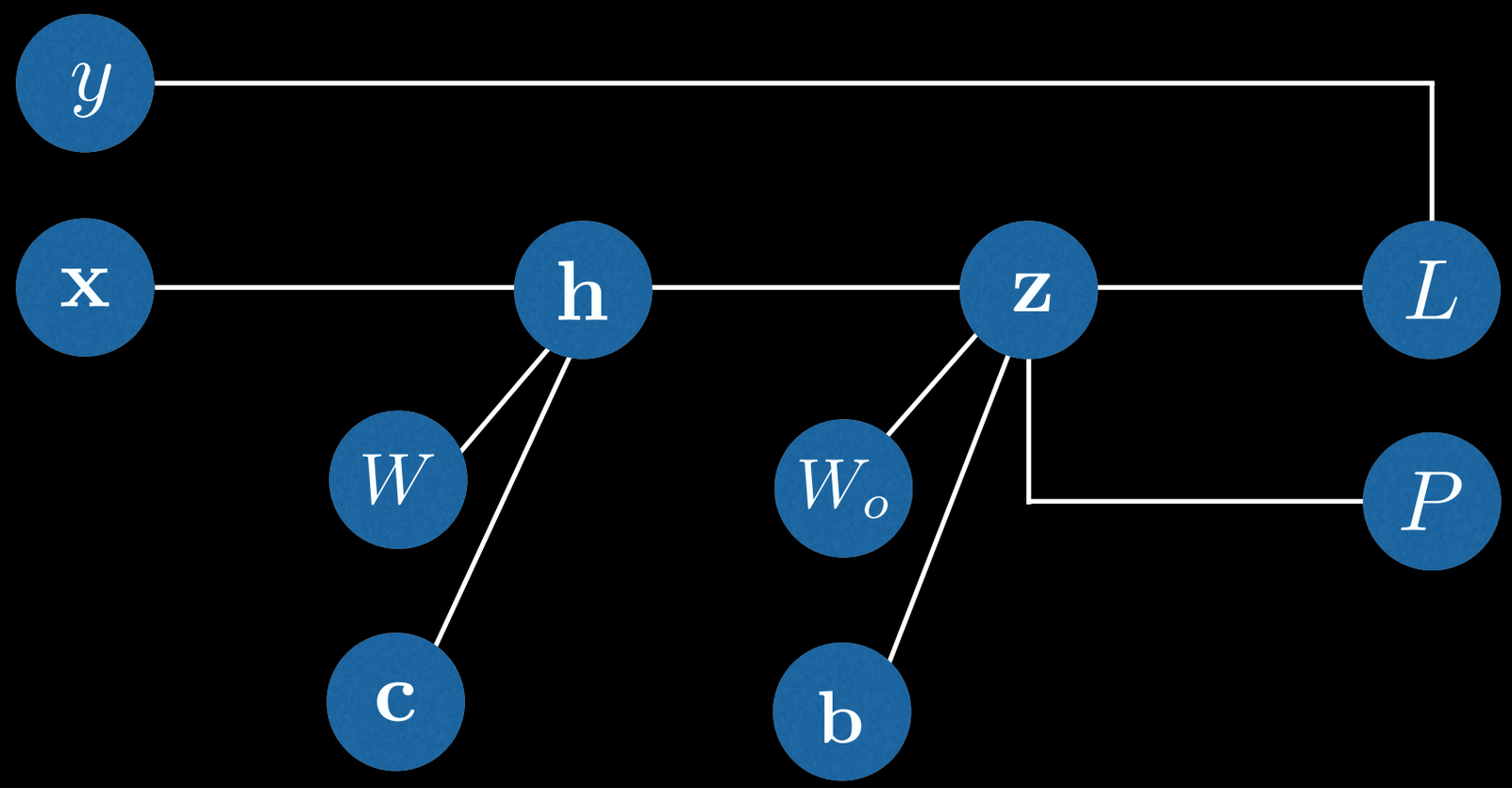


$$P = P(y|\mathbf{x}) = \frac{1}{1 + e^{(1-2y)z}}$$

$$L = -\log P(y|\mathbf{x}) = \xi((1-2y)z)$$

loss layer

Softmax Loss Layer

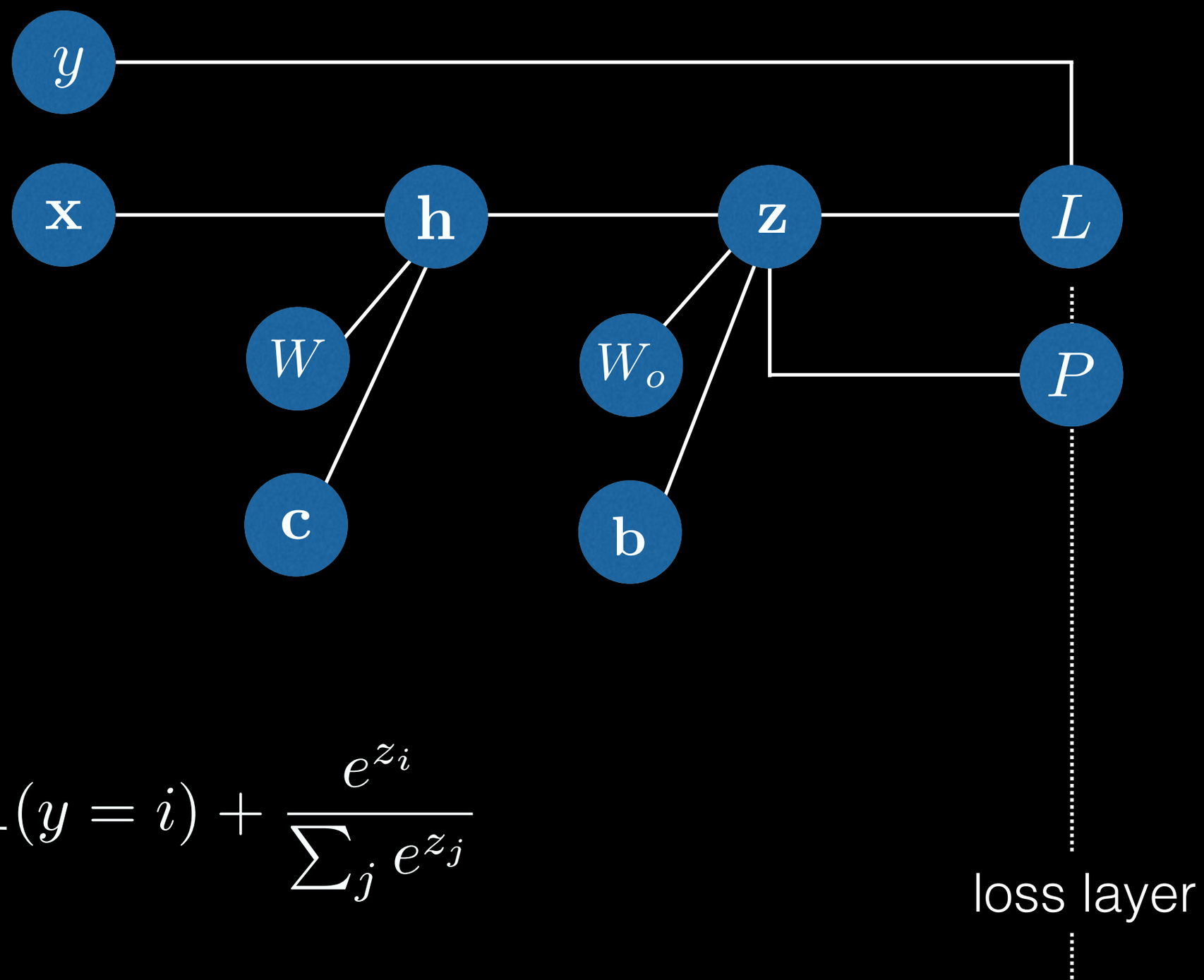


$$P = P(y = i|\mathbf{x}) = \frac{e^{z_i}}{\sum_j e^{z_j}}$$ Softmax

i = correct label

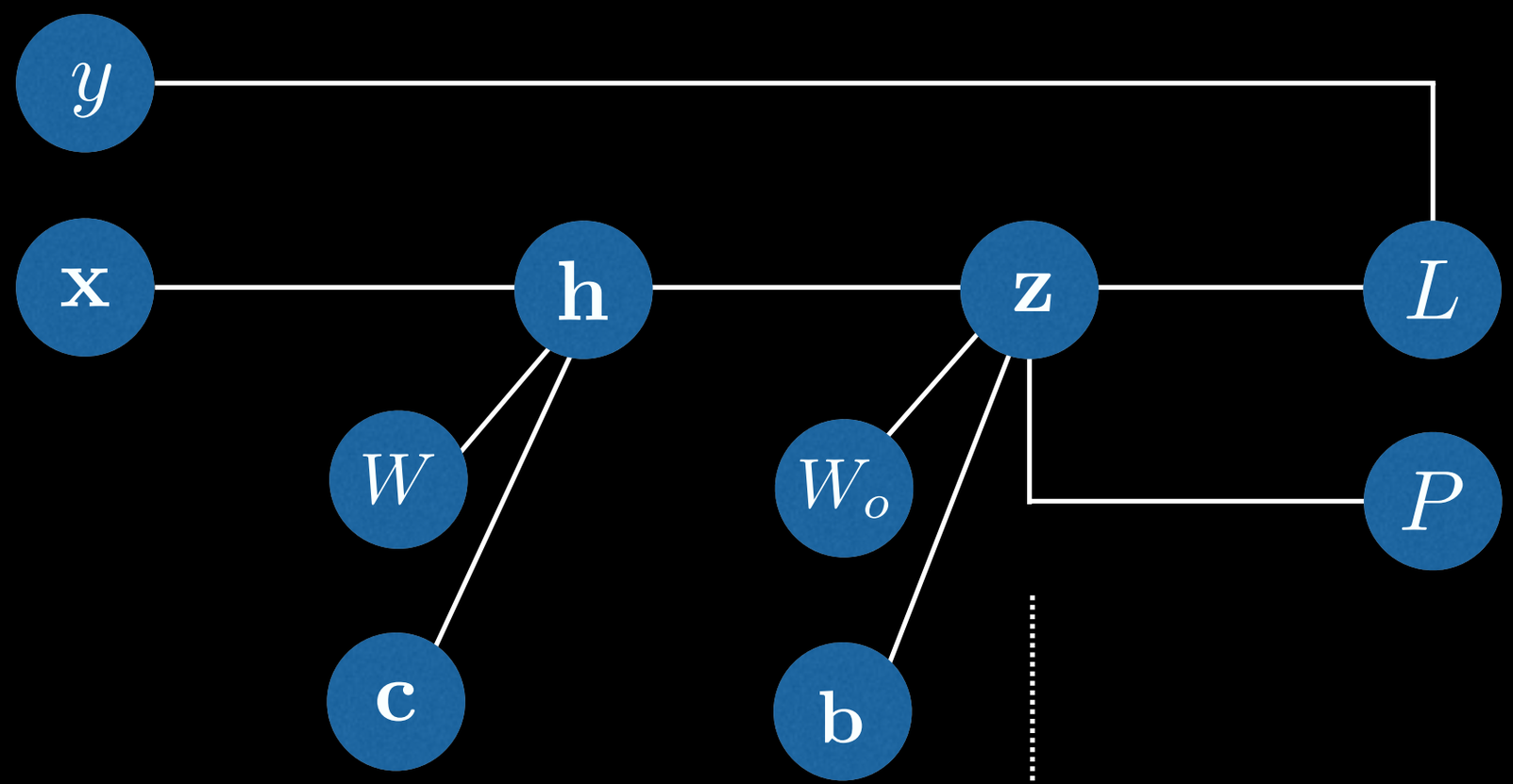
$$L = -\log P = -z_i + \log \sum_j e^{z_j}$$

Backprop with Softmax Output



$$\nabla_{\mathbf{z}} L = -\mathbf{1}(y = i) + \frac{e^{z_i}}{\sum_j e^{z_j}}$$

Backprop with Softmax Output



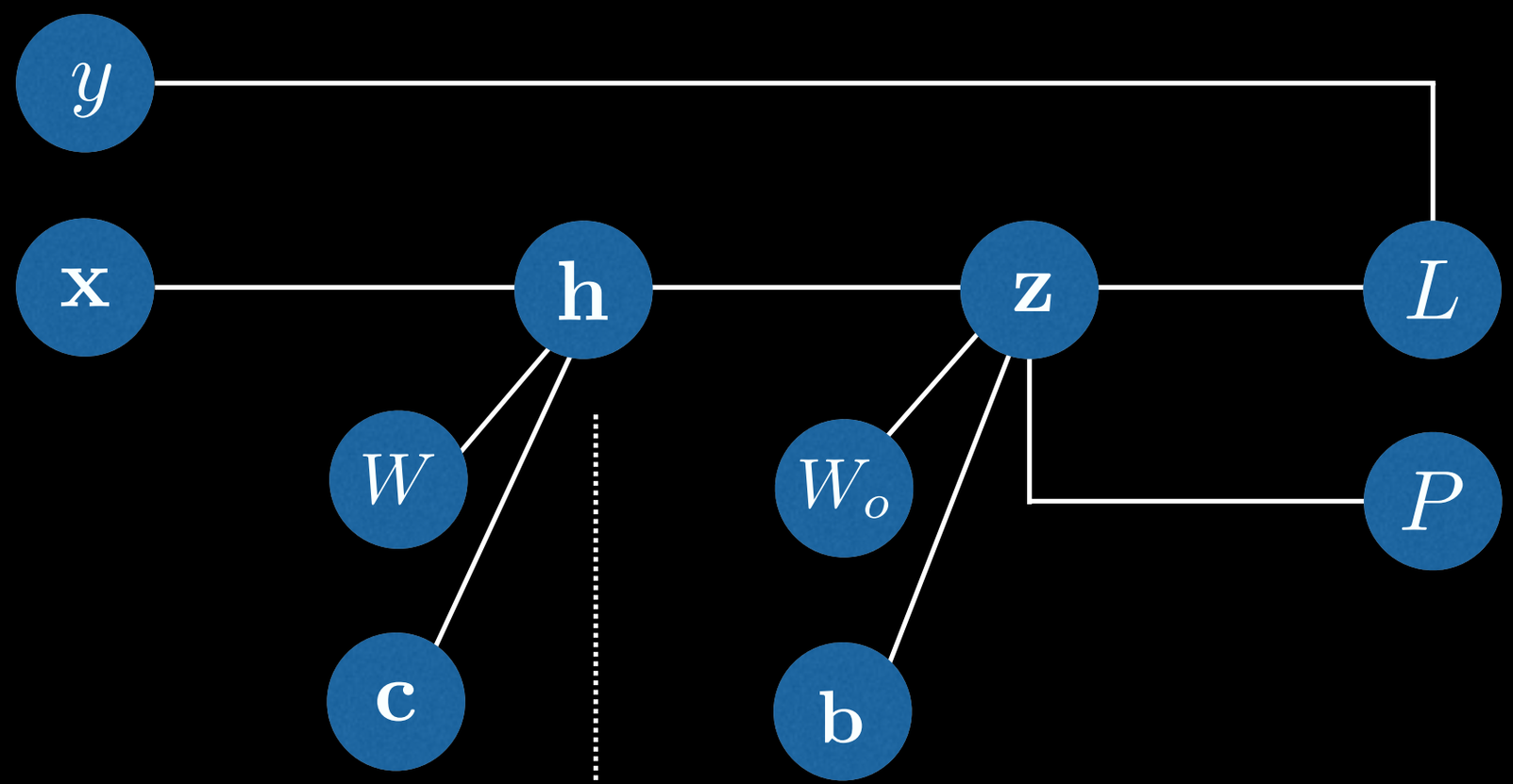
$$\nabla_{\mathbf{h}} L = W_o \nabla_{\mathbf{z}} L$$

$$\nabla_{W_o} L = \mathbf{h} (\nabla_{\mathbf{z}} L)^T$$

$$\nabla_{\mathbf{b}} L = \nabla_{\mathbf{z}} L$$

output layer

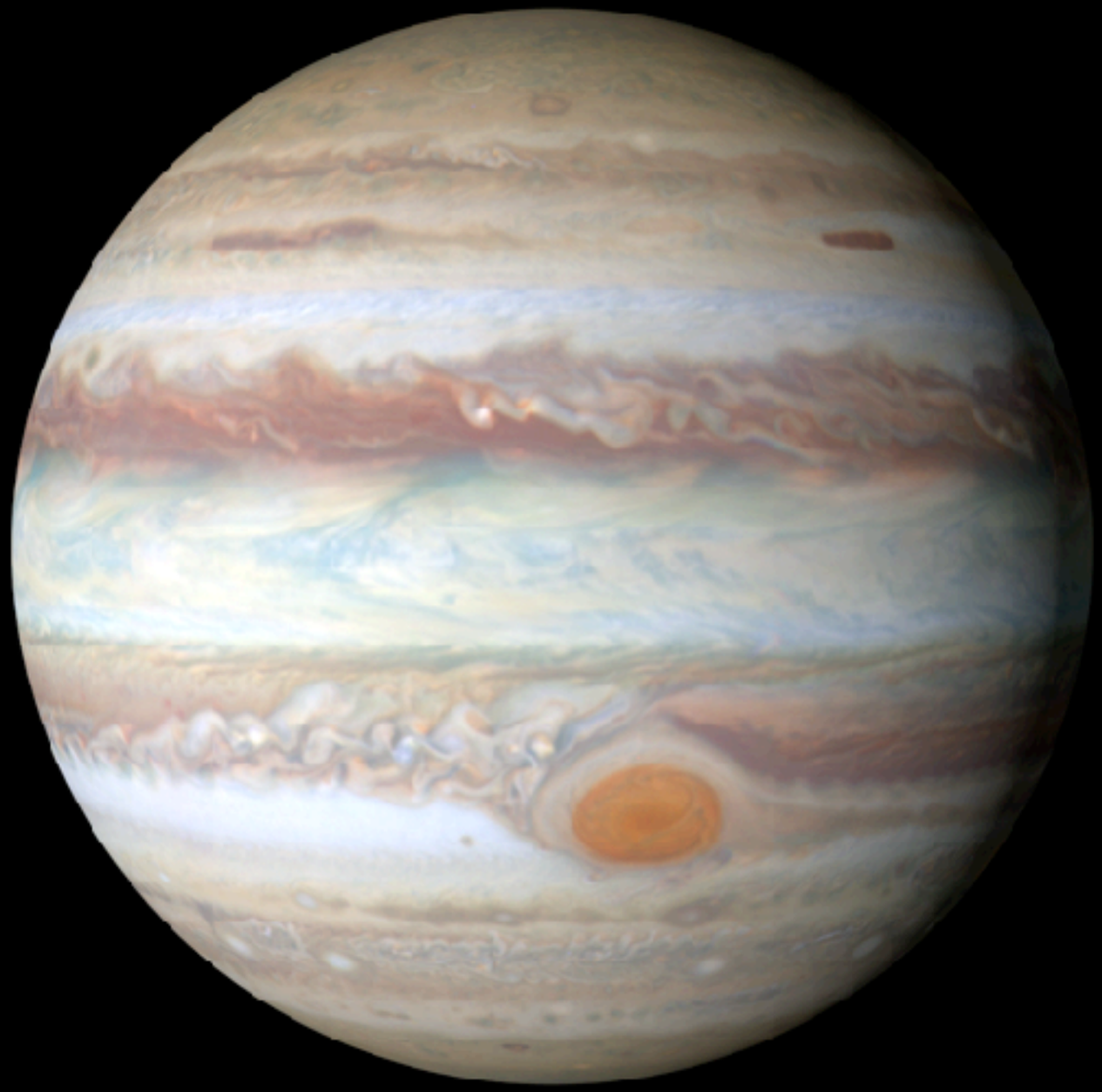
Backprop with Softmax Output



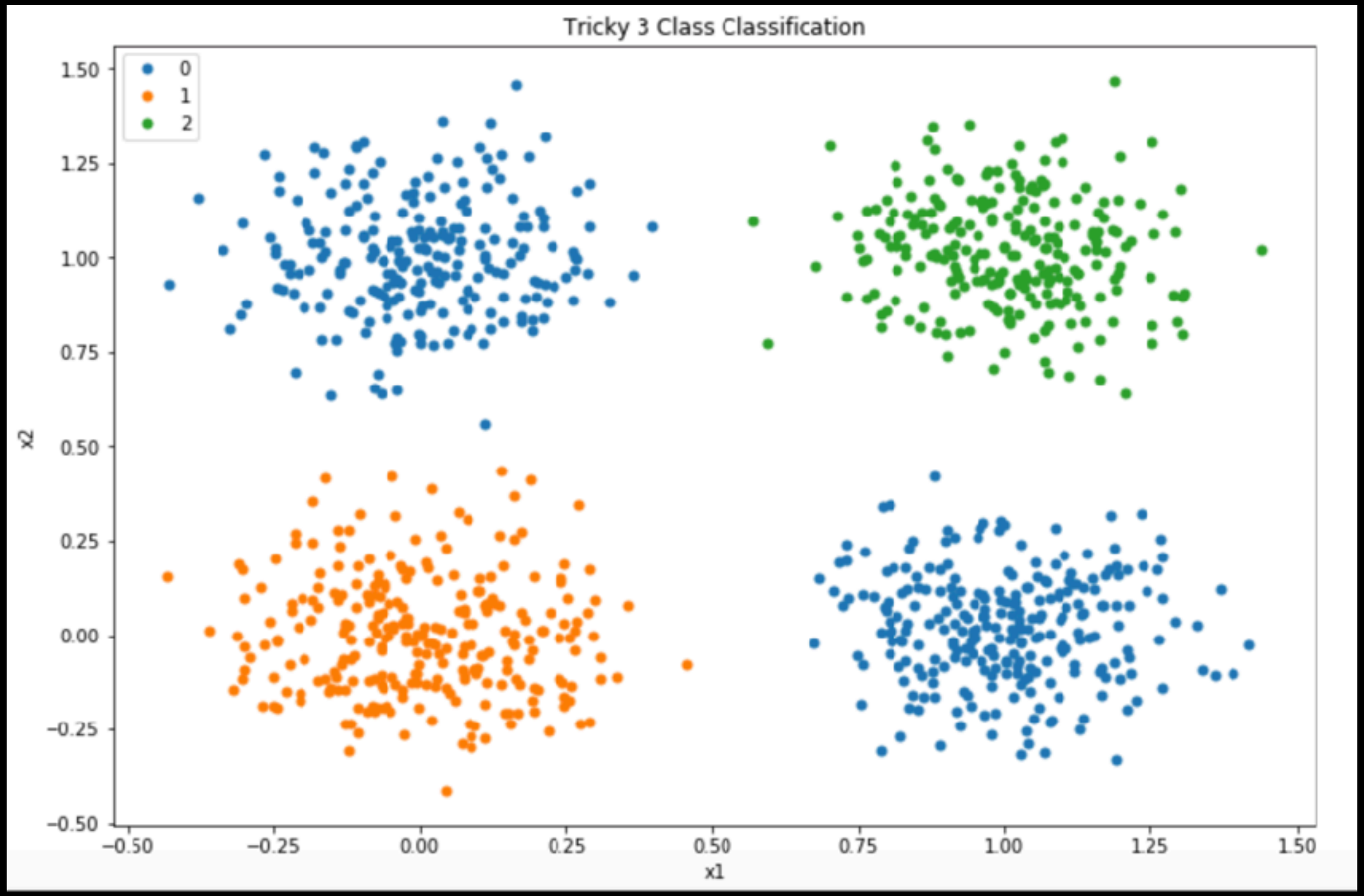
$$\nabla_W L = \mathbf{x}(f \odot \nabla_{\mathbf{h}} L)^T$$

$$\nabla_{\mathbf{c}} L = f \odot \nabla_{\mathbf{h}} L$$

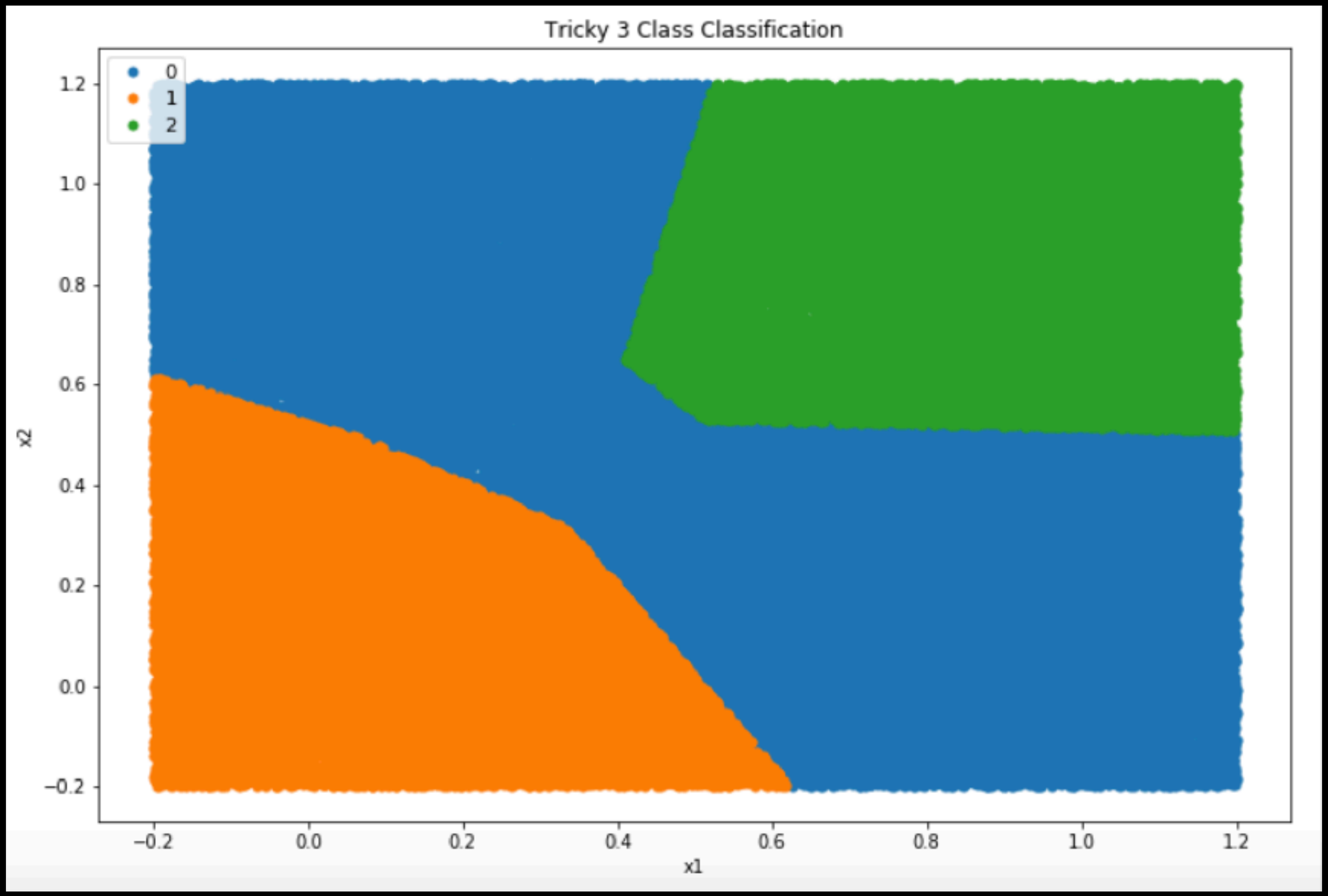
hidden layer



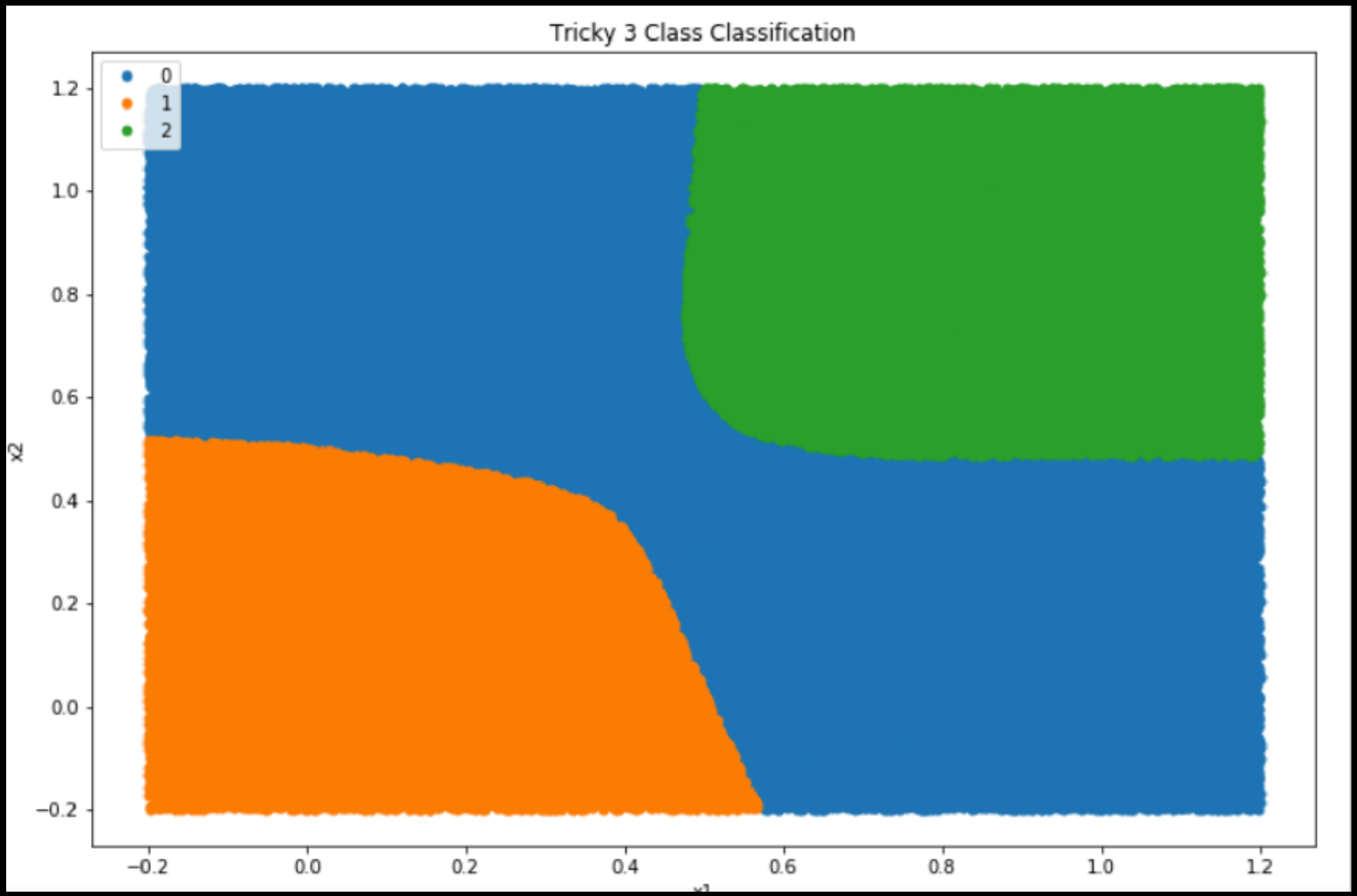
Training Samples for 3 Class Problem



Decision Regions: 16 Hidden Units



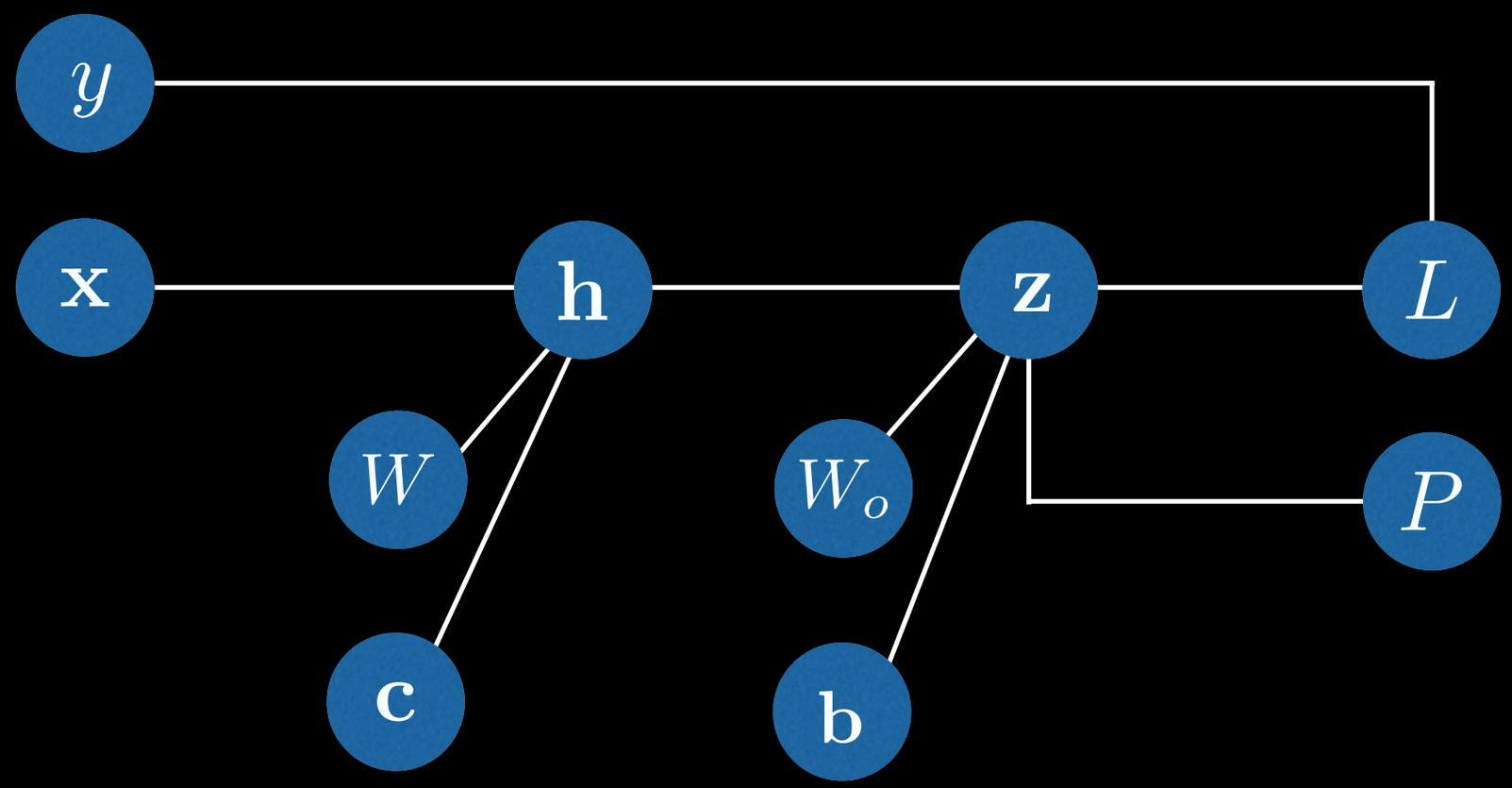
Decision Regions: 512 Hidden Units



Regularization

The goal of ***regularization*** is to prevent **overfitting** the training data with the hope that this improves **generalization**, *i.e.*, the ability to correctly handle data that the network has not trained on.

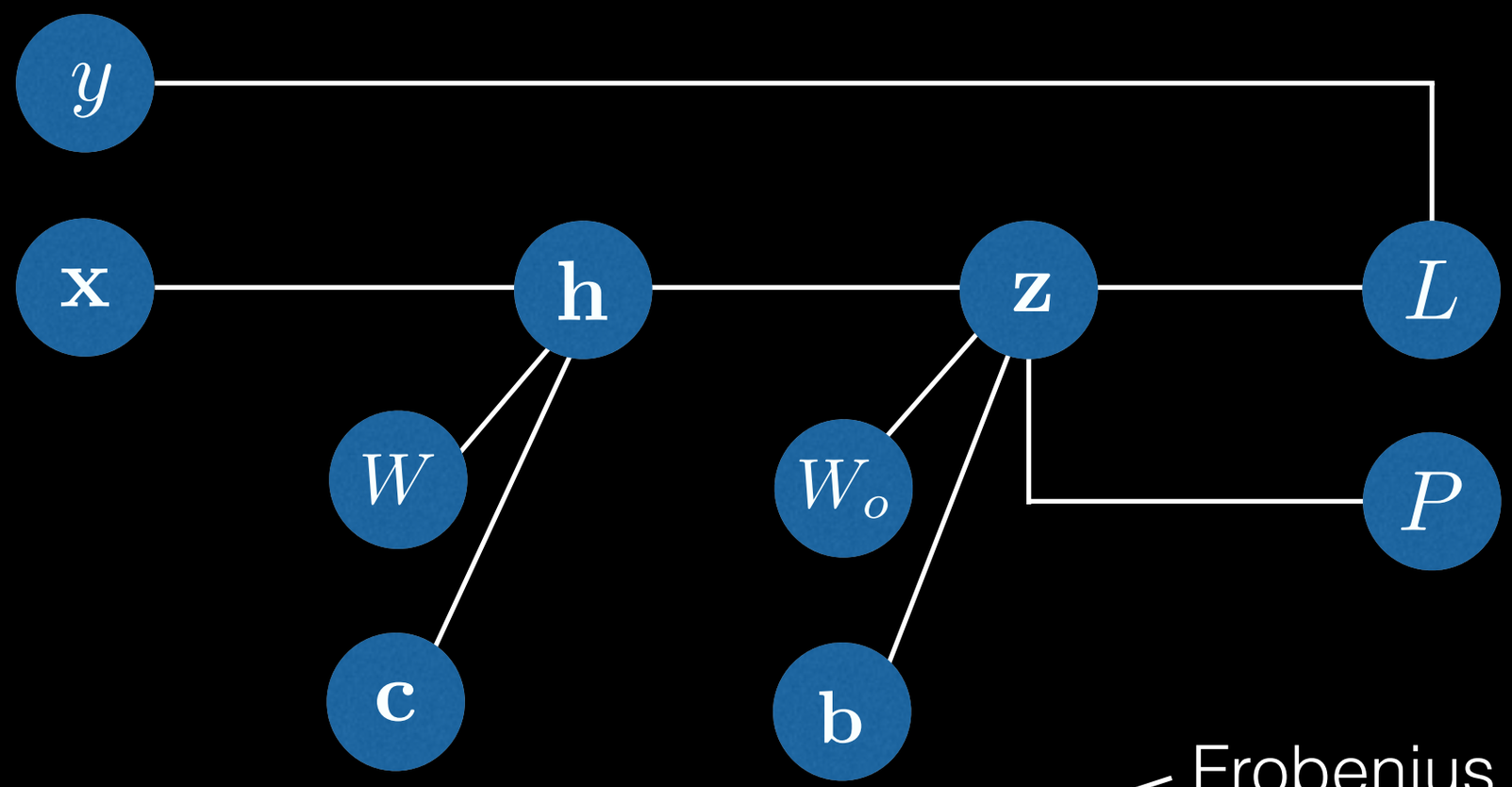
Regularization for MLP



$$L = -z_i + \log \sum_j e^{z_j} + \alpha\ \Omega(W, W_o)$$

loss regularization

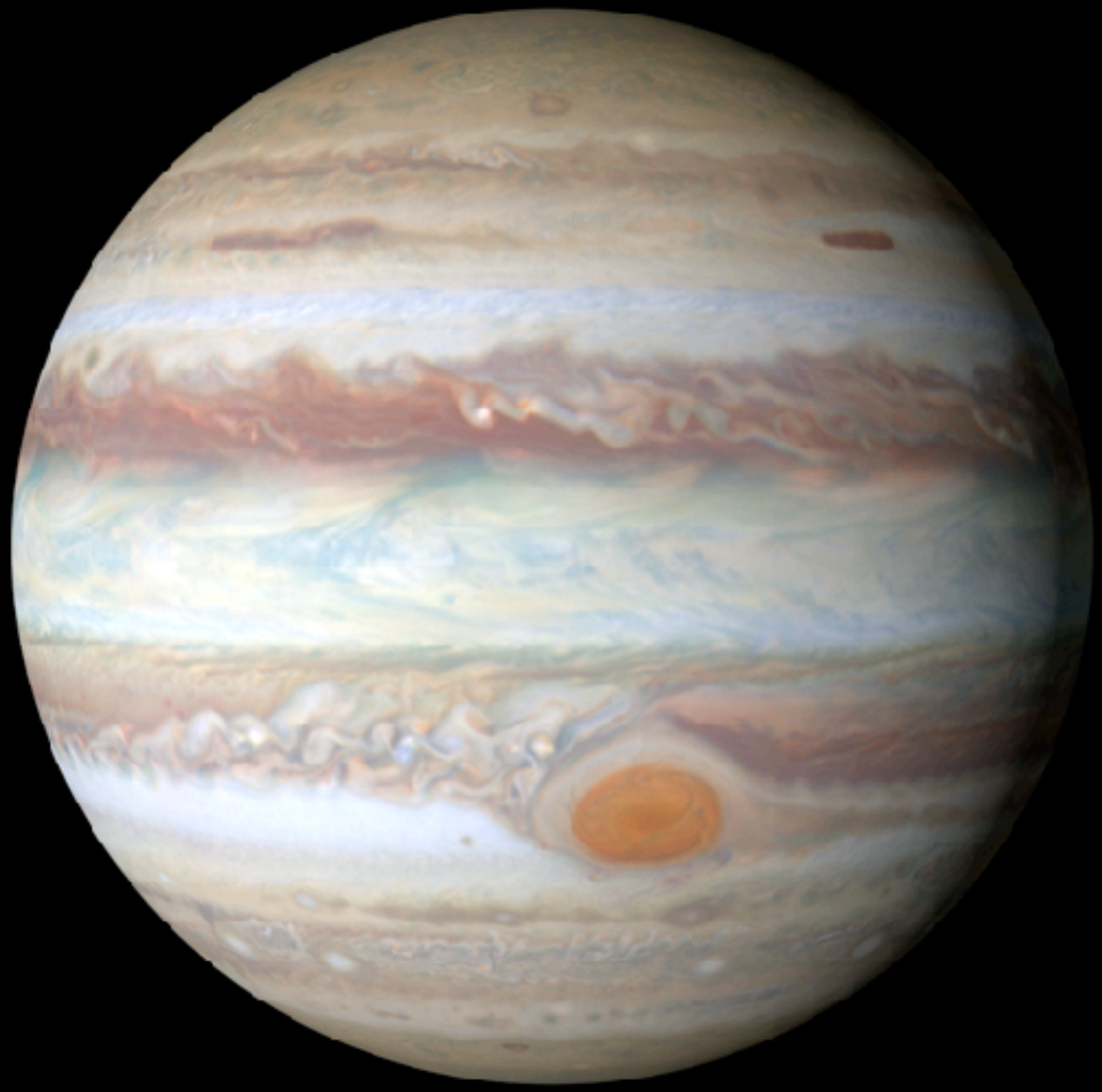
L^2 Regularization for MLP



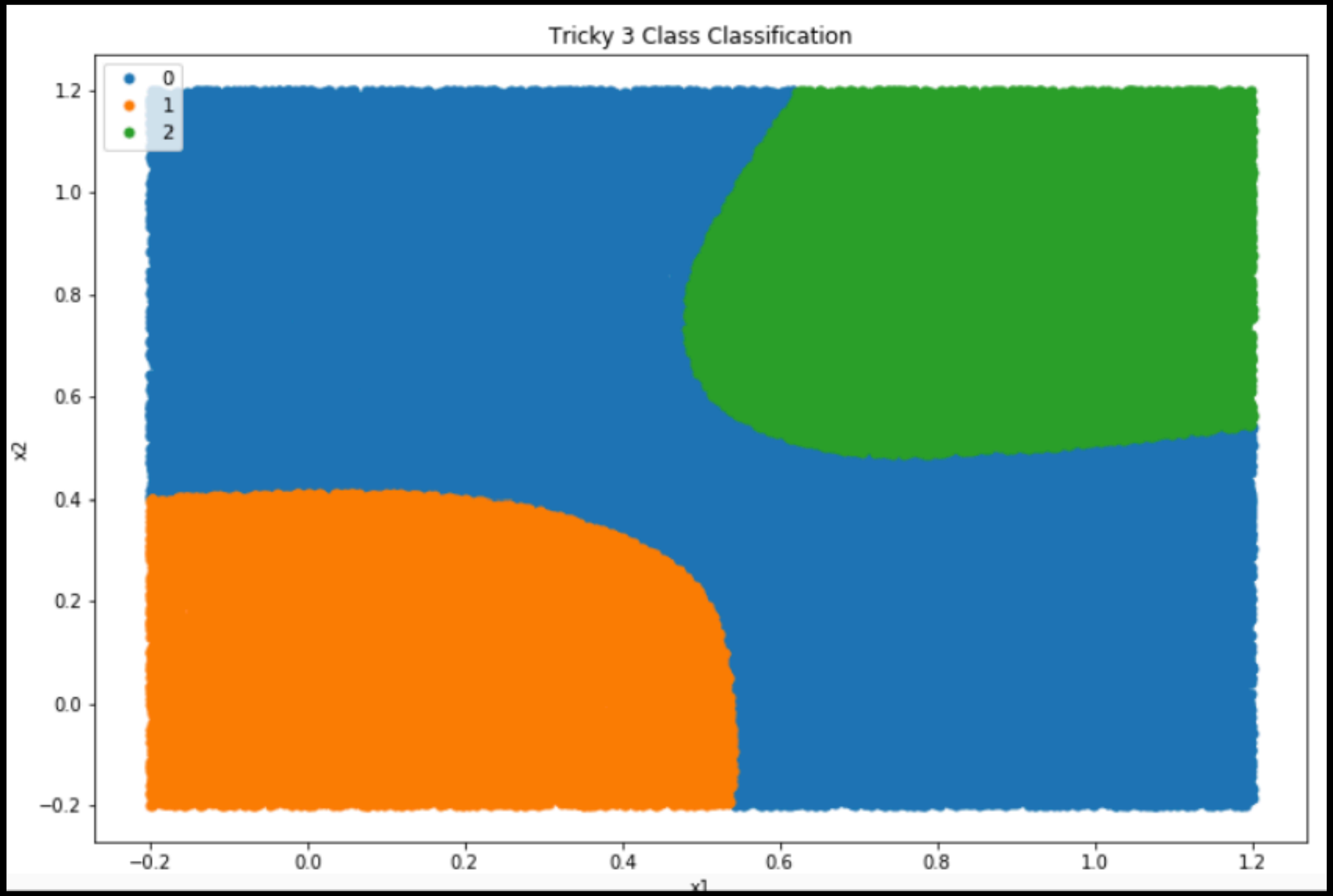
Frobenius norm

$$L = -z_i + \log \sum_j e^{z_j} + \frac{\lambda}{2}(||W||^2 + ||W_o||^2)$$

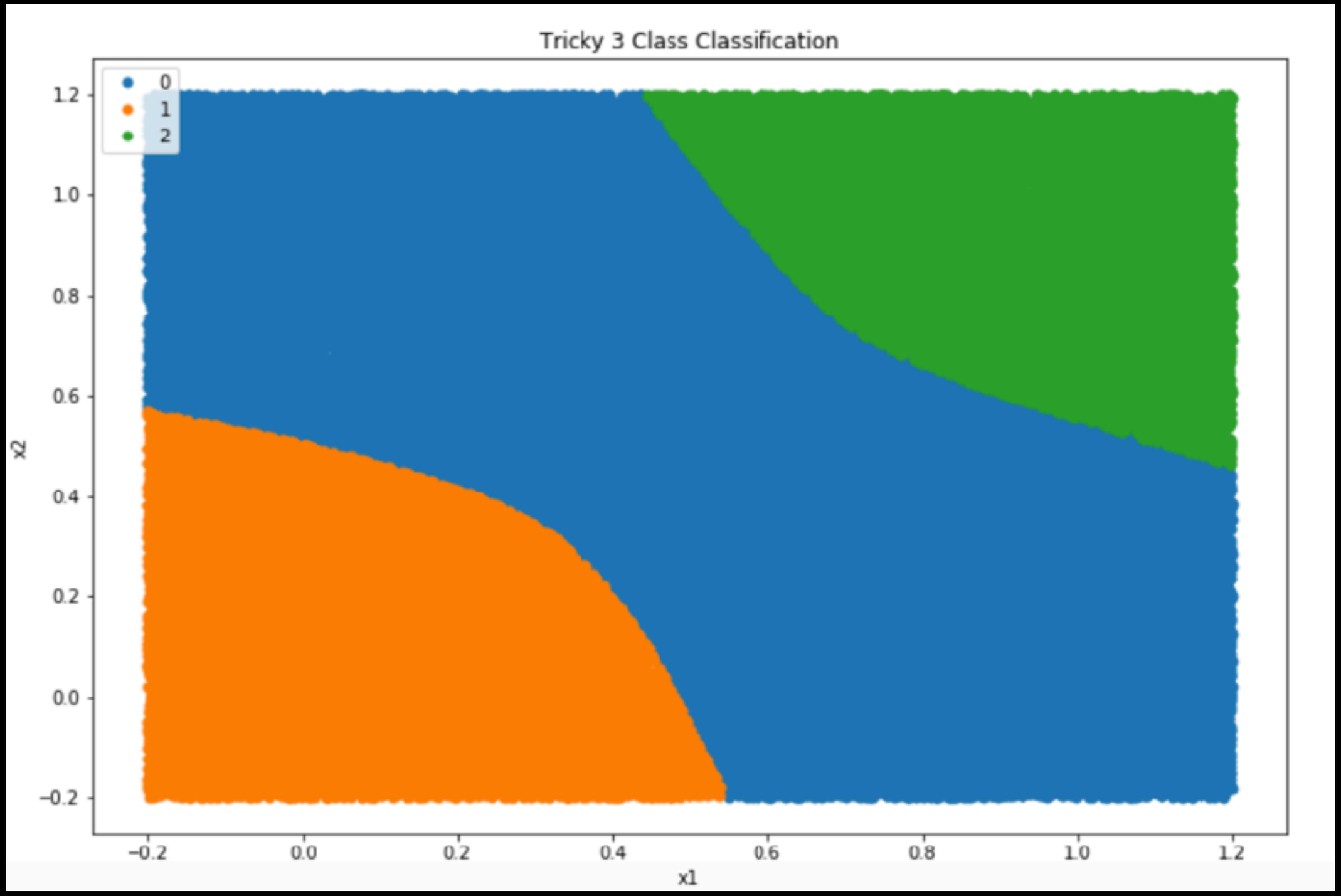
loss regularization



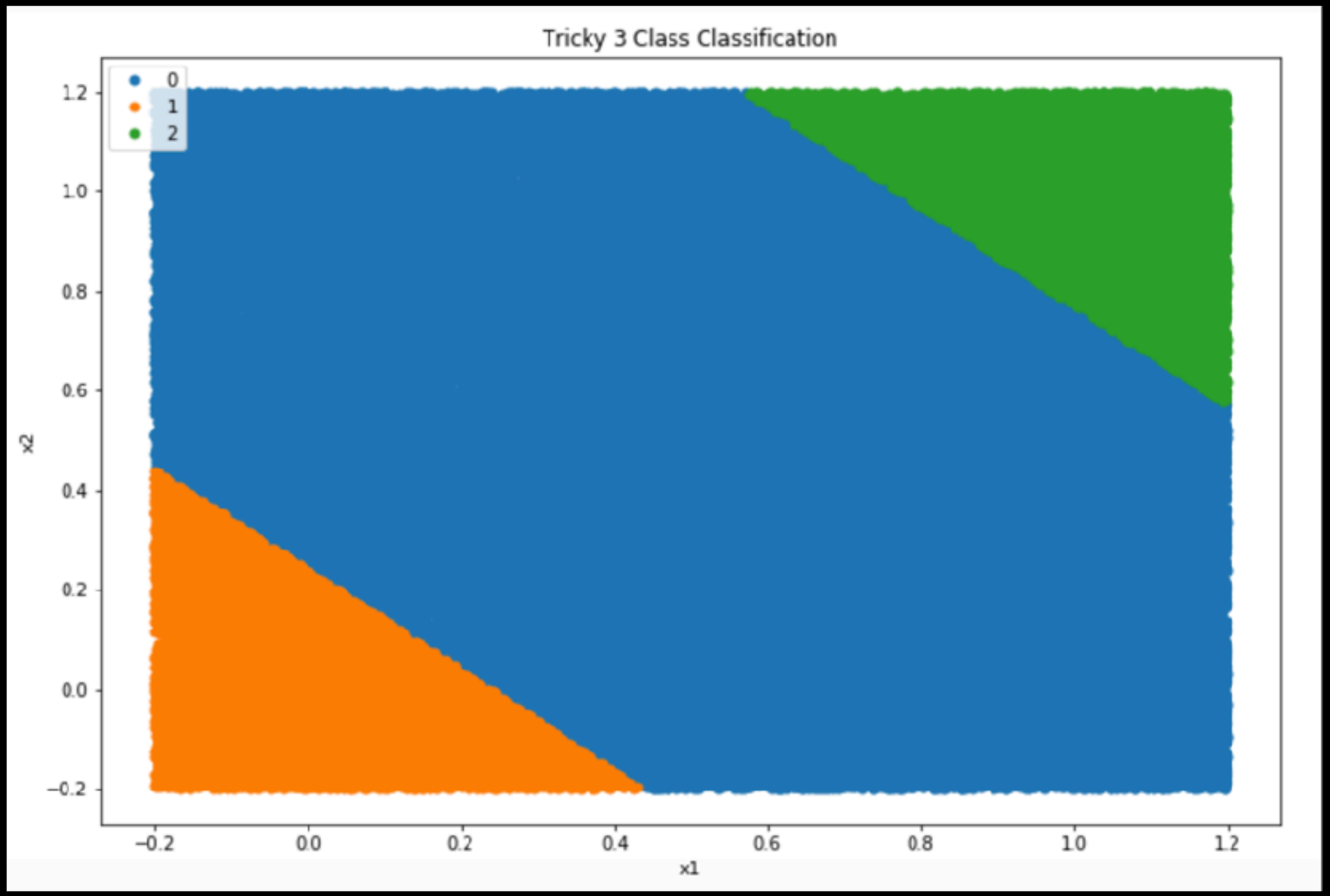
Decision Regions: 4800 HUs and $\lambda = 0$



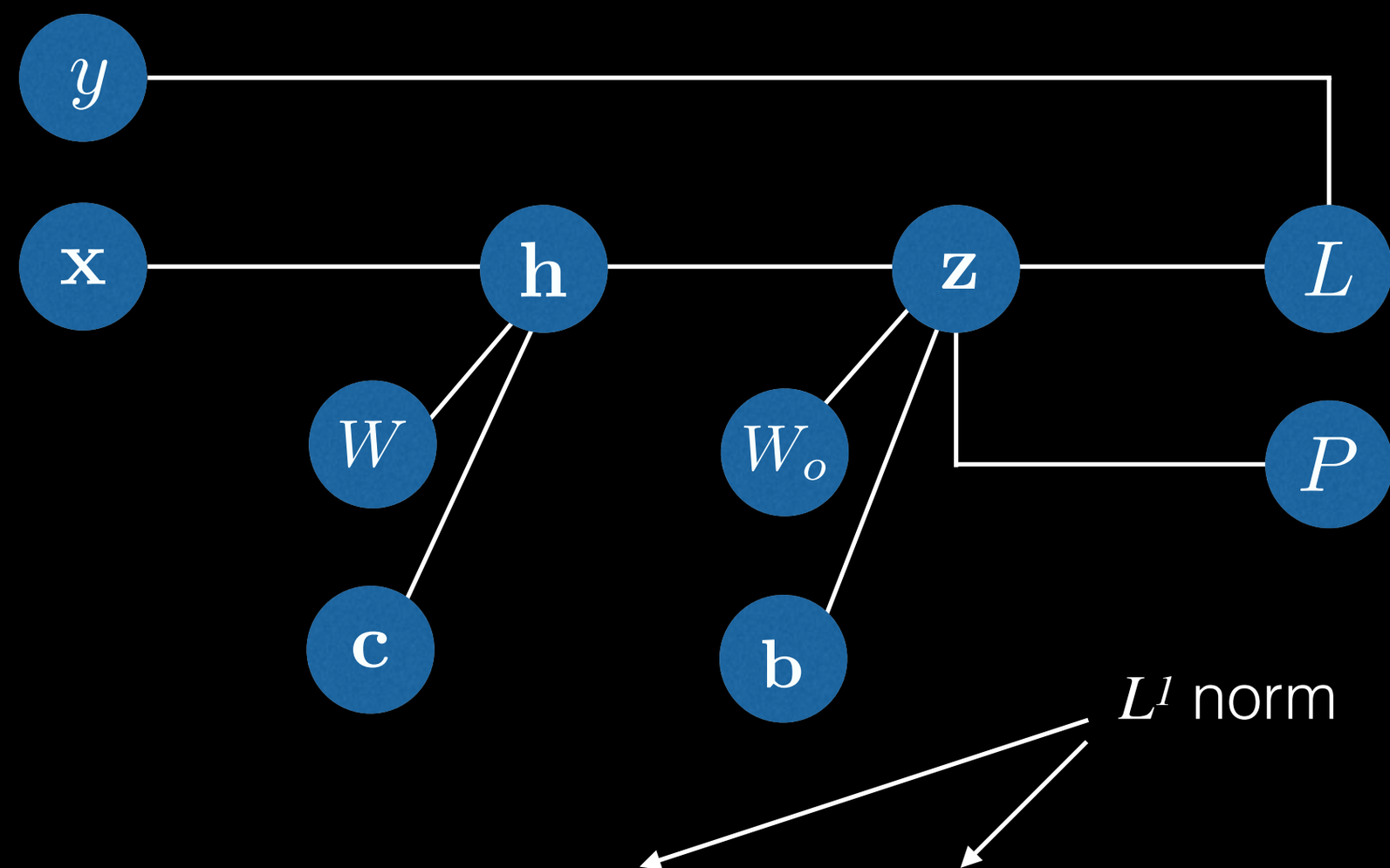
Decision Regions: 4800 HUs and $\lambda = 0.015$



Decision Regions: 4800 HUs and $\lambda = 0.1$



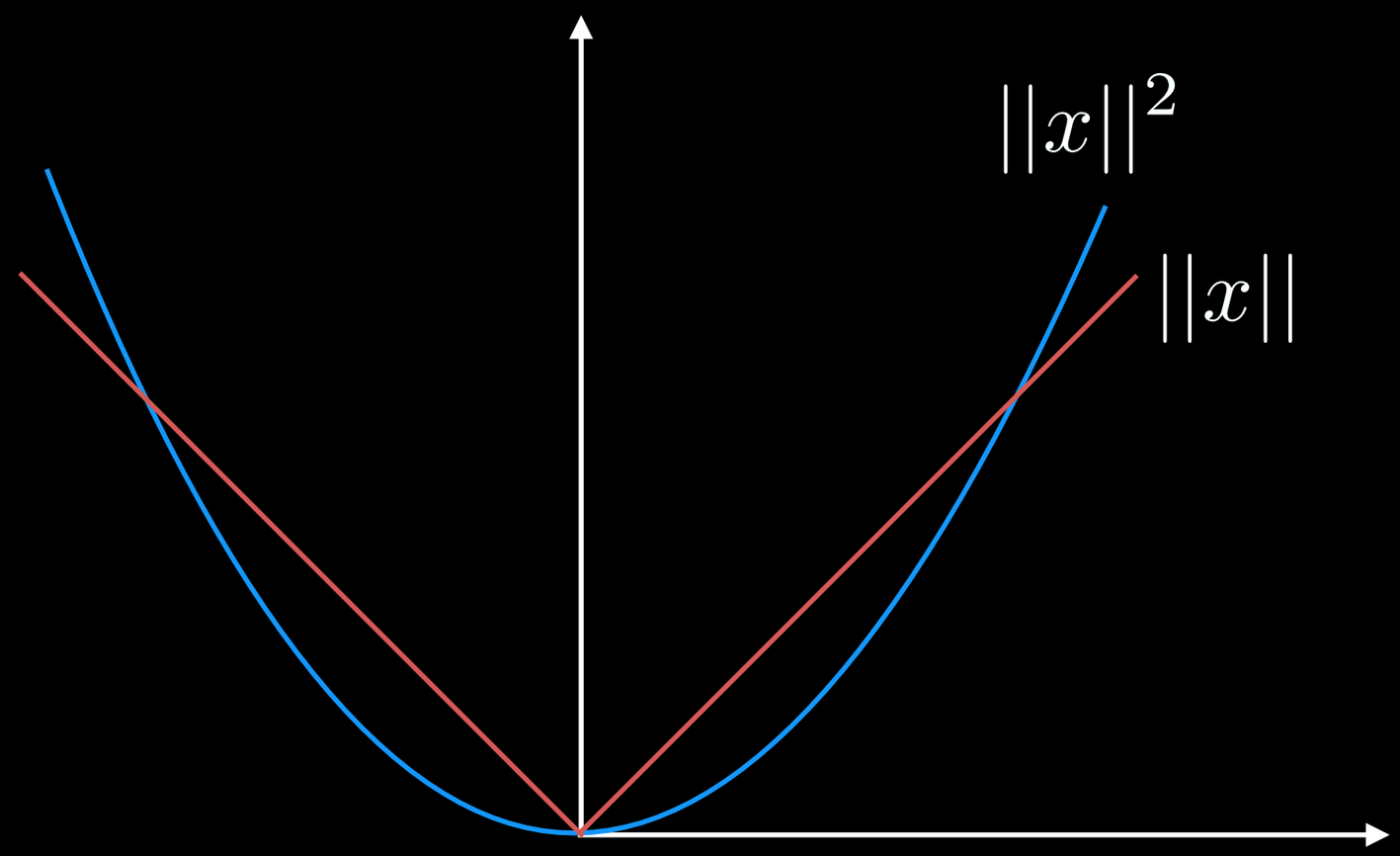
L^1 Regularization for MLP



$$L = -z_i + \log \sum_j e^{z_j} + \alpha(||W||_1 + ||W_o||_1)$$

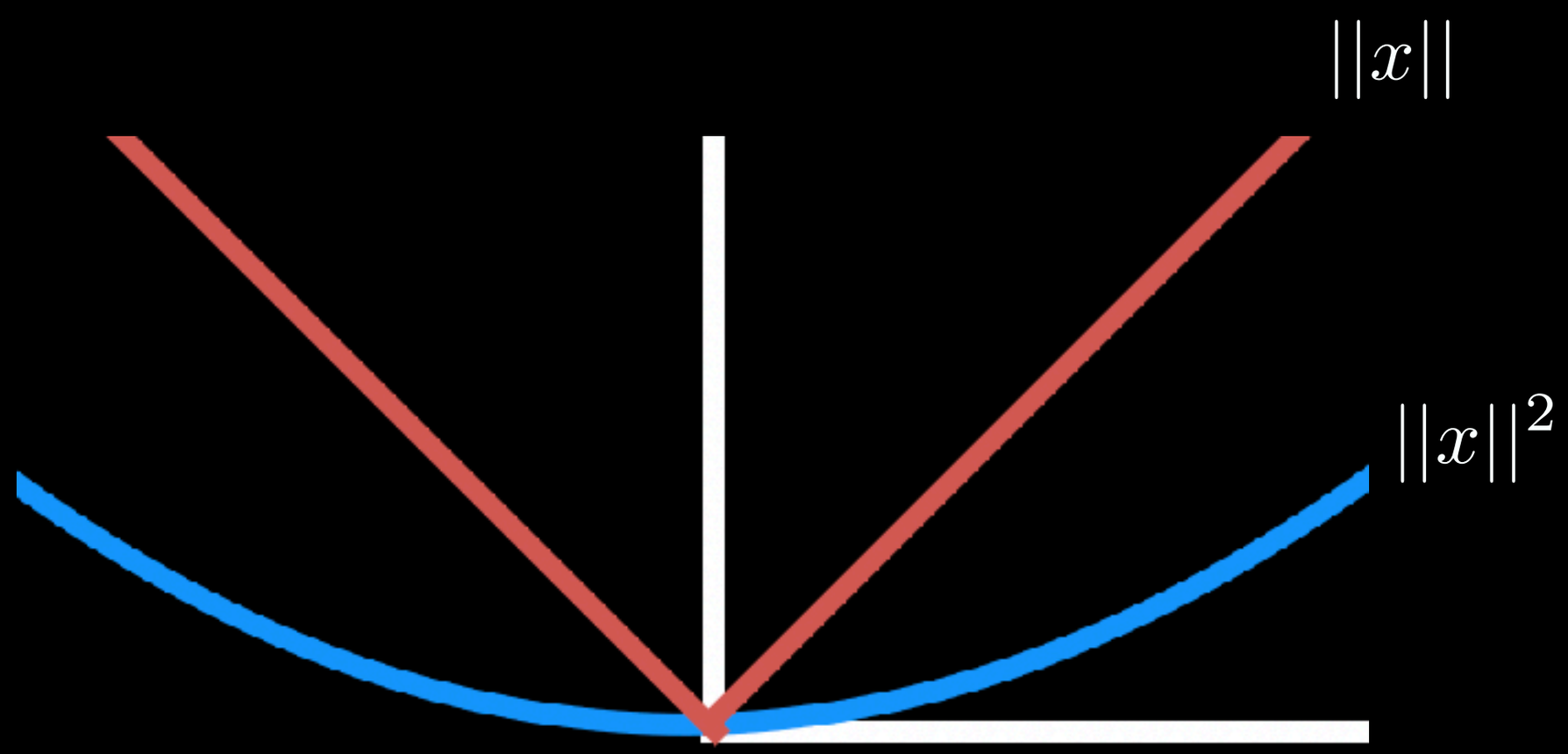
loss regularization

L^1 vs L^2 Regularization



Note gradient behavior close to 0!

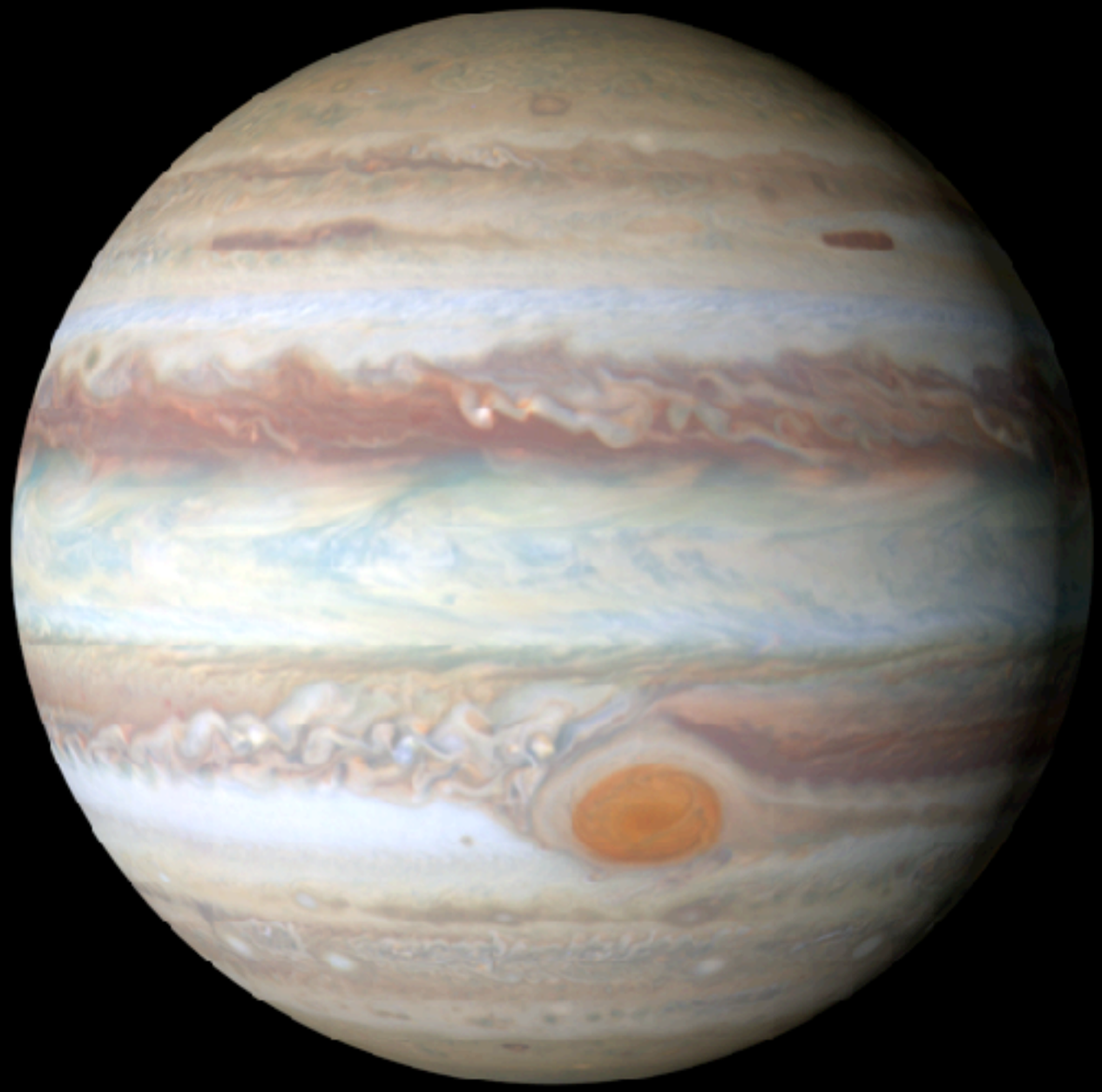
L^1 vs L^2 Regularization



Note gradient behavior close to 0!

L^1 Regularization

- L^1 produces solutions that are ***sparse*** compared with L^2.
- This means most weights are driven to zero. Why? The gradient of this regularizer does not flatten out as it approaches zero and pushes unneeded weights down.
- This can be used as a method of ***feature selection***.



Regularization: Bagging

- Train a model by random sampling with replacement (re-use training samples)
- Repeat this problem k times to produce k separate models
- Have all models vote on output for test samples

[Breiman, 1994]

Ensemble Methods

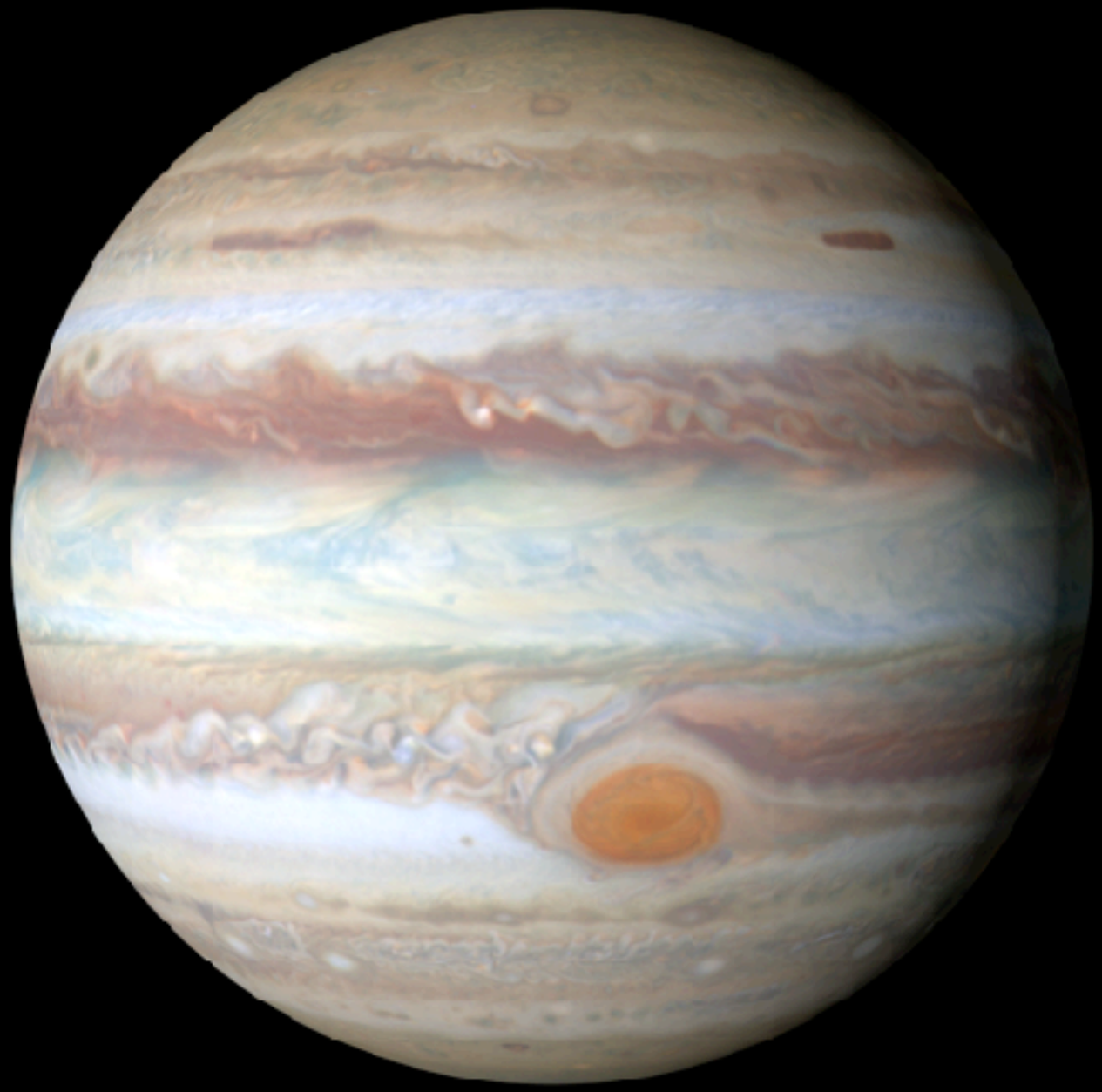
- Bagging involves model averaging which is a type of what are called ***ensemble methods.***
- Most model winners in ML competitions use an ensemble method.
- In practice ensemble methods have greater memory requirements and are slower as they require multiple models to be trained and evaluated.
- Great for getting your accuracy up a few % points but not necessarily best for deploying at scale.

Possible Ensemble Method

- What if you made a smaller network using fewer input and hidden units.
- We could do this by randomly removing some of the units, say with probability 0.2 for input units and probability 0.5 for hidden units.
- Let's keep all output units as these map to specific categories that we are trying to learn.
- Let's make LOTS of these smaller networks, half-size networks and then combine them at the end as one ensemble method.

Dropout [Srivastava et al., 2014]

- For every training batch through the network, ***dropout*** 0.5 of hidden units and 0.2 of input units. You can choose the probabilities as you like…
- Train as you normally would using SGD, but each time you impose a random ***dropout*** that essentially trains for that batch on a random sub-network.
- When you are done training, you use for your model the complete network with all its learned weights, except multiply the weight by the probability of including its parent unit.
- This is called the ***weight scaling inference rule***. [Hinton et al., 2012]



Early Stopping

- Typical deep neural networks have millions and millions of weights!
- With so many parameters at their disposal how to we prevent them from overfitting?
- Clearly we can use some of the other regularization techniques that have been mentioned…
- …but given enough training time, our network will eventually start to overfit the data.

DON’T EVER FORGET!

DON’T EVER FORGET!

DON’T EVER FORGET!

The goal of ***regularization*** is to prevent **overfitting** the training data with the hope that this improves **generalization**, *i.e.*, the ability to correctly handle data that the network has not trained on.

DON’T EVER FORGET!

DON’T EVER FORGET!

DON’T EVER FORGET!

Training and Validation Sets

Labeled Data

Training Data

Validation Data

Training and Validation Sets

Labeled Data

Training Data

Validation Data

NEVER TRAIN ON YOUR VALIDATION SET!

Training and Validation Sets

Labeled Data

Training Data

Validation Data

NEVER TRAIN ON YOUR VALIDATION SET!

Training and Validation Sets

Labeled Data

Training Data

Validation Data

NEVER TRAIN ON YOUR VALIDATION SET!

Training and Validation Sets

Labeled Data

Training Data

Validation Data

NEVER TRAIN ON YOUR VALIDATION SET!

Training, Testing, and Validation Sets

Labeled Data

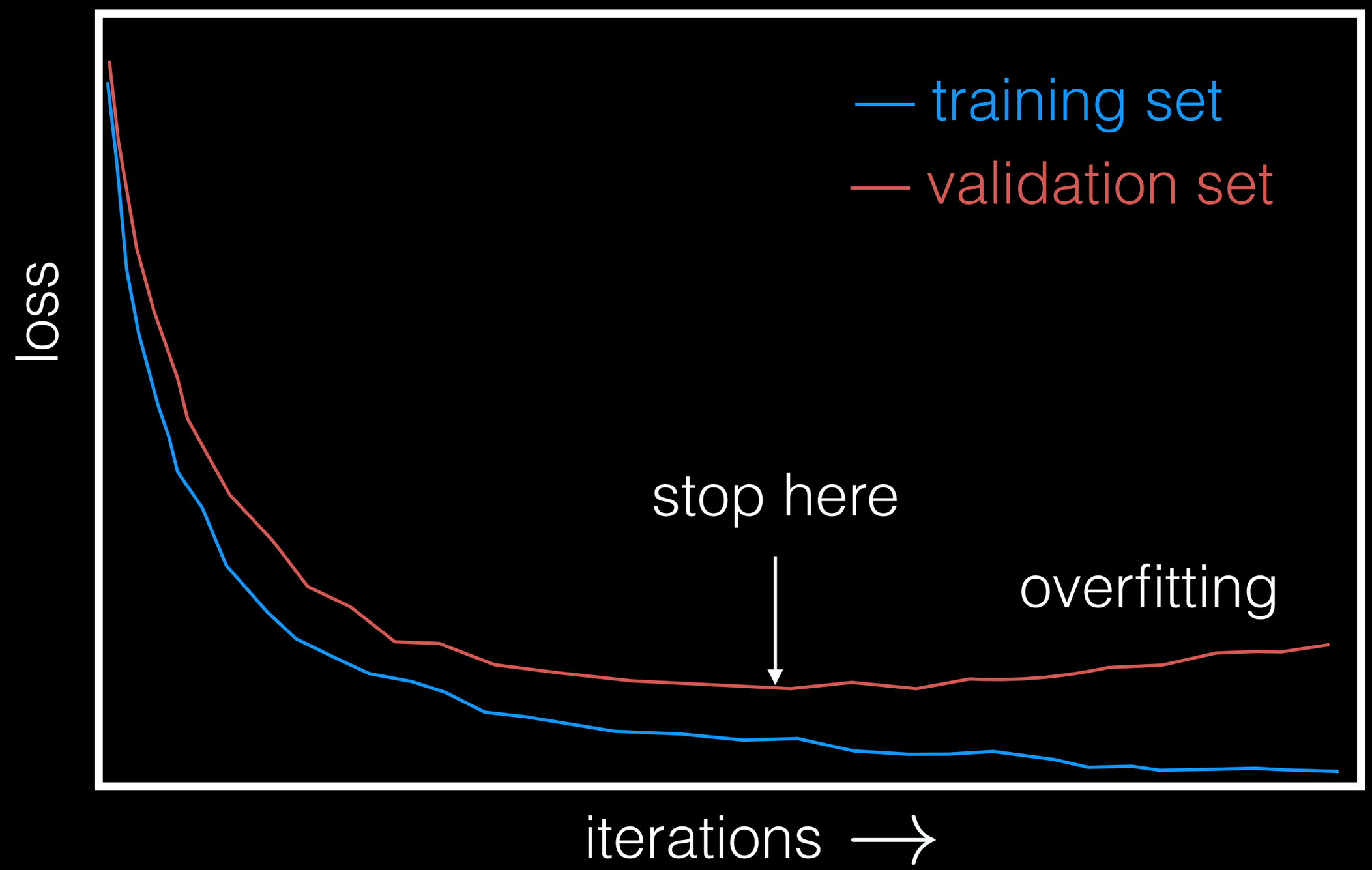
Test Data

Training Data

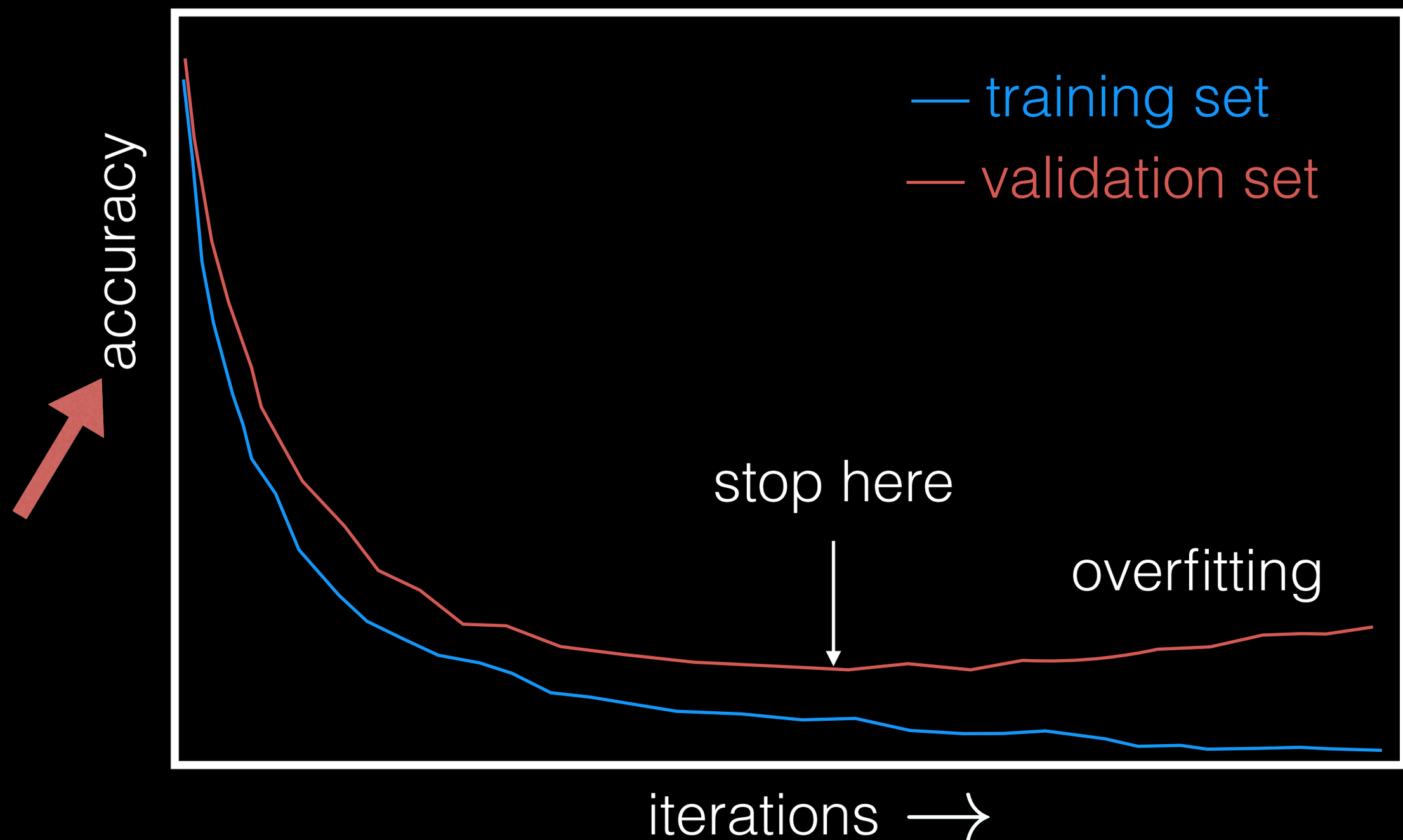
Validation Data

NEVER TRAIN ON YOUR VALIDATION SET!

Early Stopping



Early Stopping



More Data / Data Augmentation

- Possibly the best way to prevent overfitting is to get more training data!
- When this isn't possible, you can often perform ***data augmentation*** where training samples are randomly perturbed or ***jittered*** to produce more training samples.
- This is usually easy to do when the samples are images and one can crop, rotate, etc. the samples to produce new samples.
- And if the data can be generated synthetically using computer graphics, we can produce an endless supply.