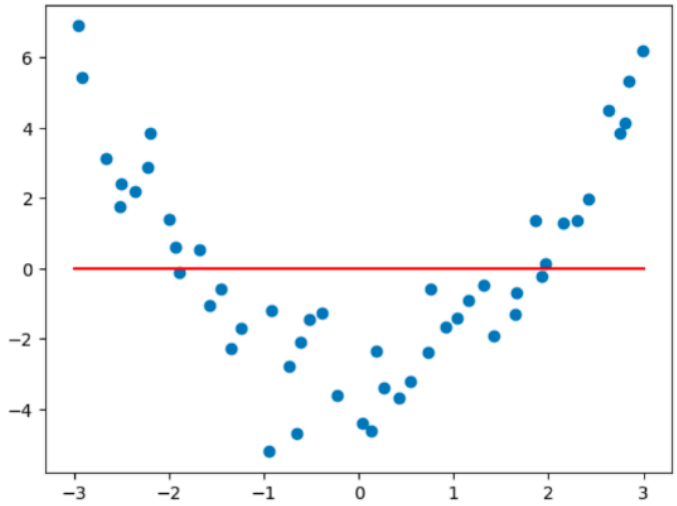
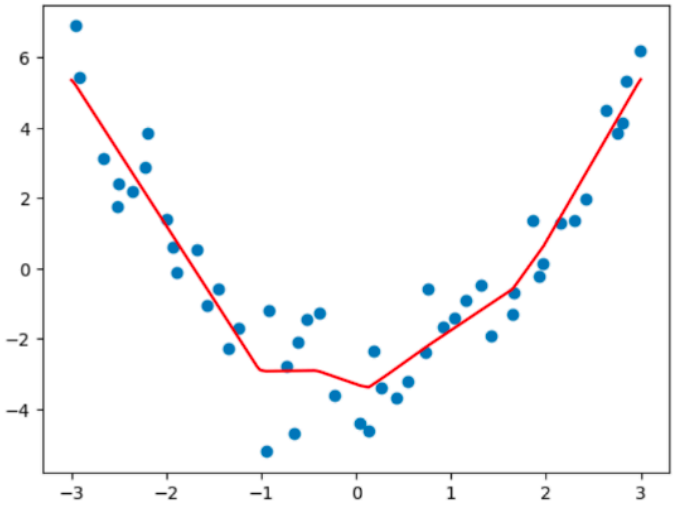
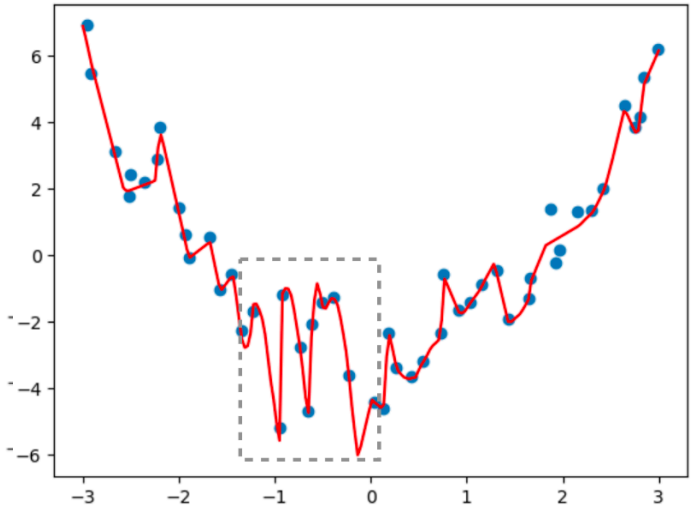
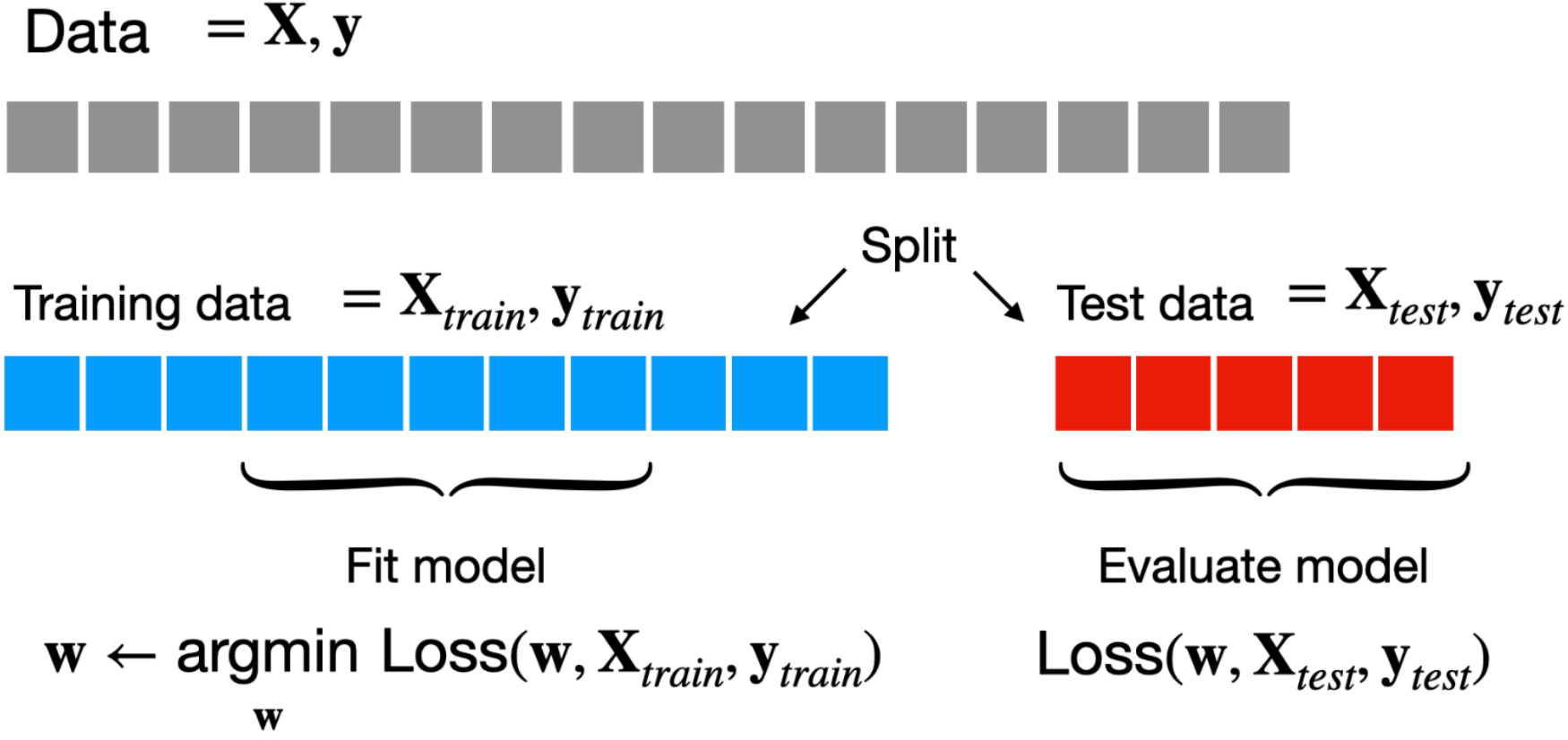
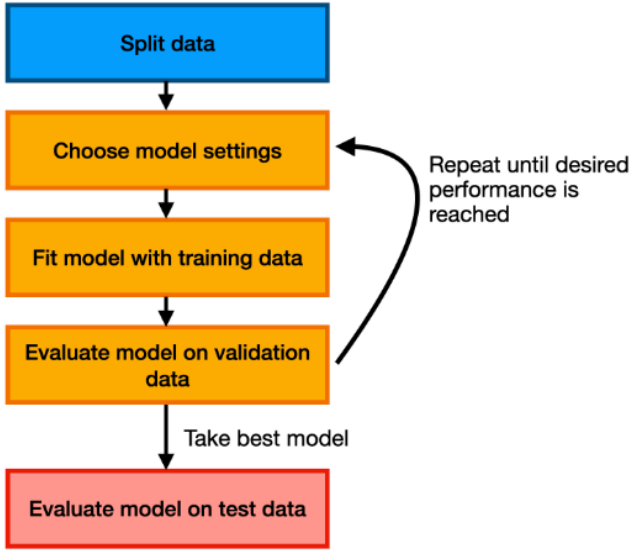


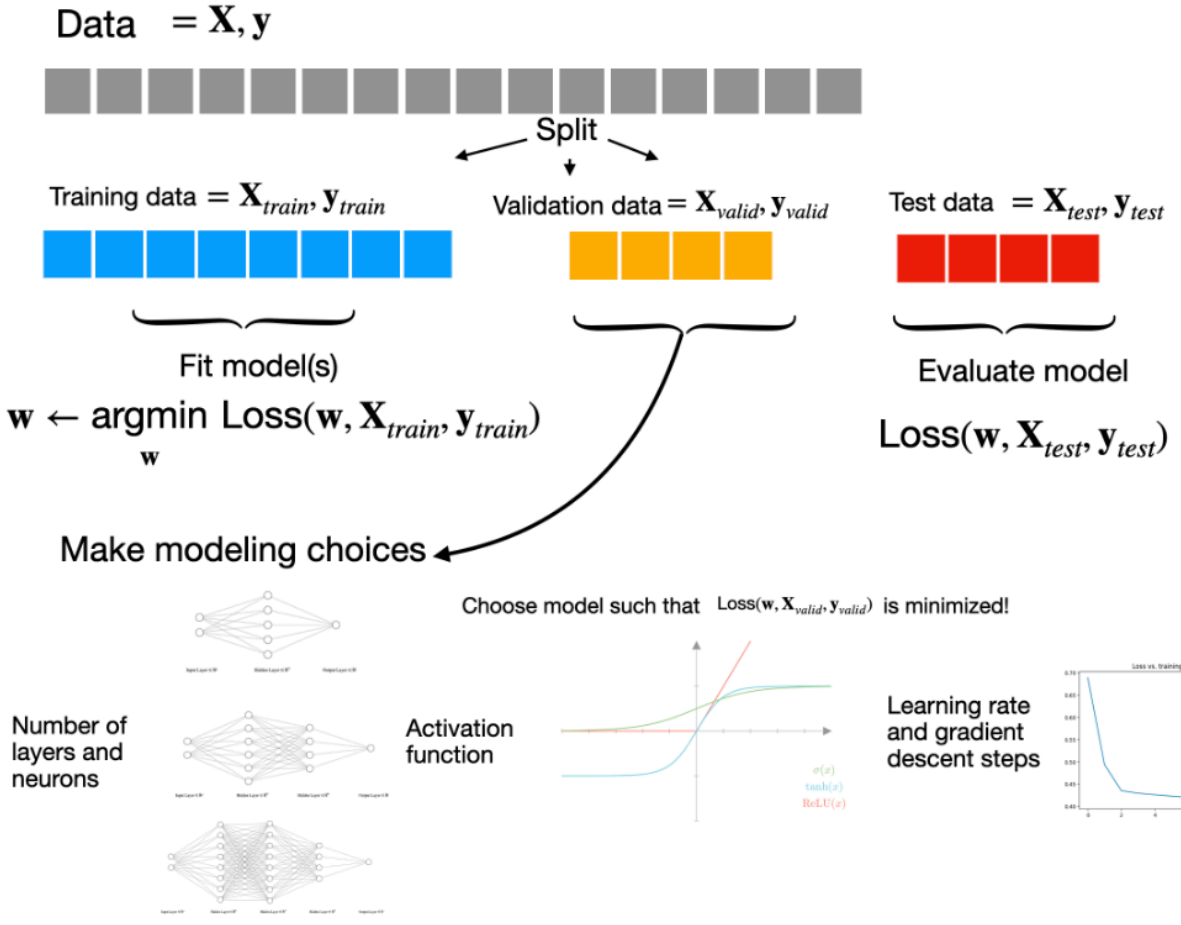


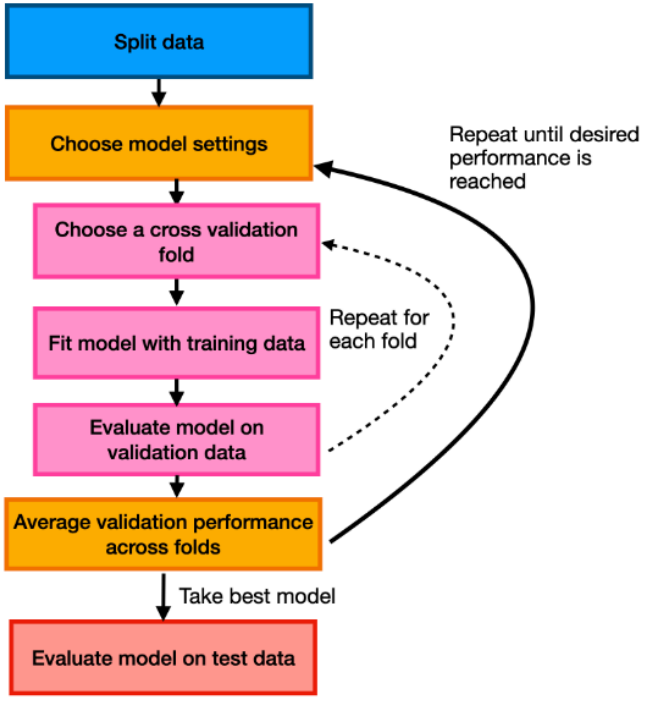


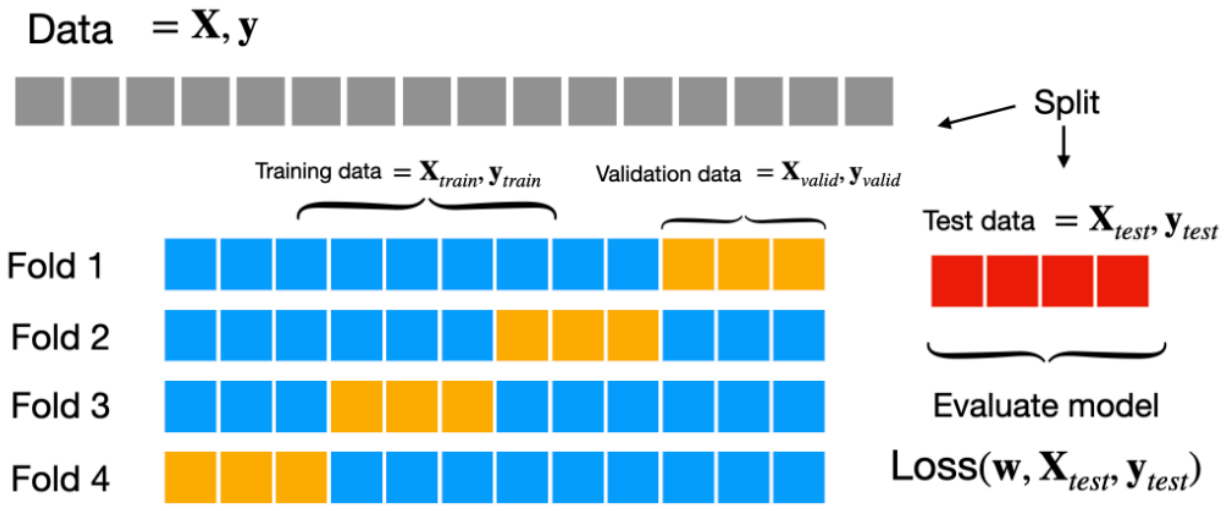
- Which function best fits this data? Why?
- What are some of the choices that we can make when designing a neural network?
- What might we infer about the differences in the neural networks used for each function?
- What can we conclude about the neural network weights in the third figure from looking at the highlighted region?


Data $= \mathbf{X}, \mathbf{y}$
Split
Training data $= \mathbf{X}_{train}, \mathbf{y}_{train}$
Test data $= \mathbf{X}_{test}, \mathbf{y}_{test}$
Fit model
Evaluate model
$\mathbf{w} \leftarrow \underset{\mathbf{w}}{\text{argmin}}\ \text{Loss}(\mathbf{w}, \mathbf{X}_{train}, \mathbf{y}_{train})$
$\text{Loss}(\mathbf{w}, \mathbf{X}_{test}, \mathbf{y}_{test})$


Split data
Choose model settings
Fit model with training data
Evaluate model on validation data
Take best model
Evaluate model on test data
Repeat until desired performance is reached


Data $= \mathbf{X}, \mathbf{y}$
Split
Training data $= \mathbf{X}_{train}, \mathbf{y}_{train}$
Validation data $= \mathbf{X}_{valid}, \mathbf{y}_{valid}$
Test data $= \mathbf{X}_{test}, \mathbf{y}_{test}$
Fit model(s)
$\mathbf{w} \leftarrow \underset{\mathbf{w}}{\text{argmin}}\ \text{Loss}(\mathbf{w}, \mathbf{X}_{train}, \mathbf{y}_{train})$
Evaluate model
$\text{Loss}(\mathbf{w}, \mathbf{X}_{test}, \mathbf{y}_{test})$
Make modeling choices
Choose model such that $\text{Loss}(\mathbf{w}, \mathbf{X}_{valid}, \mathbf{y}_{valid})$ is minimized!
Number of layers and neurons
Activation function
Learning rate and gradient descent steps





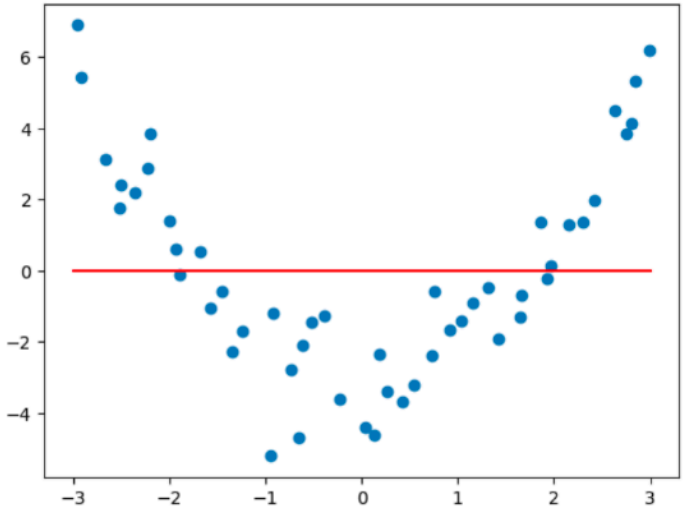
For *each* fold:

Fit model(s)

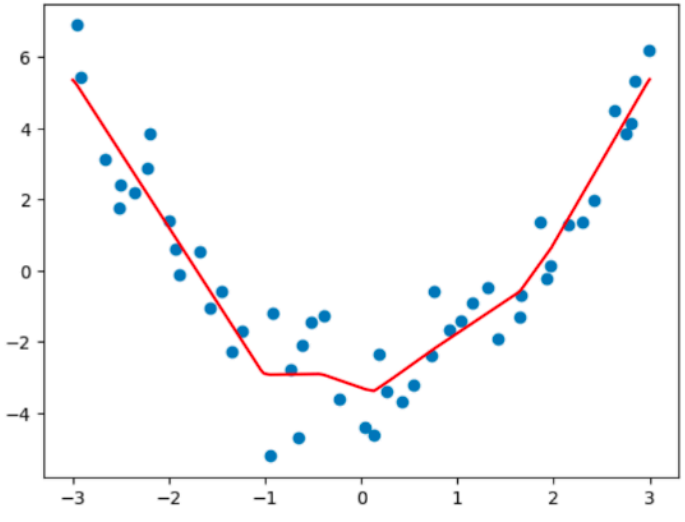
$$\mathbf{w} \leftarrow \underset{\mathbf{w}}{\text{argmin}}\ \text{Loss}(\mathbf{w}, \mathbf{X}_{train}, \mathbf{y}_{train})$$

Evaluate model

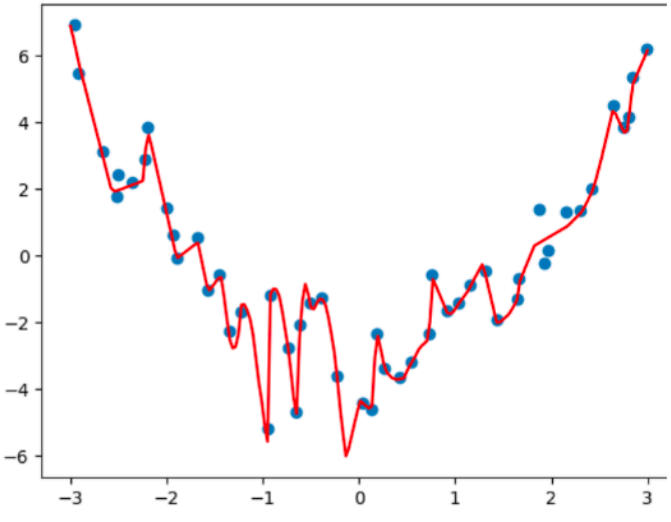
$$\text{Loss}(\mathbf{w}, \mathbf{X}_{valid}, \mathbf{y}_{valid})$$



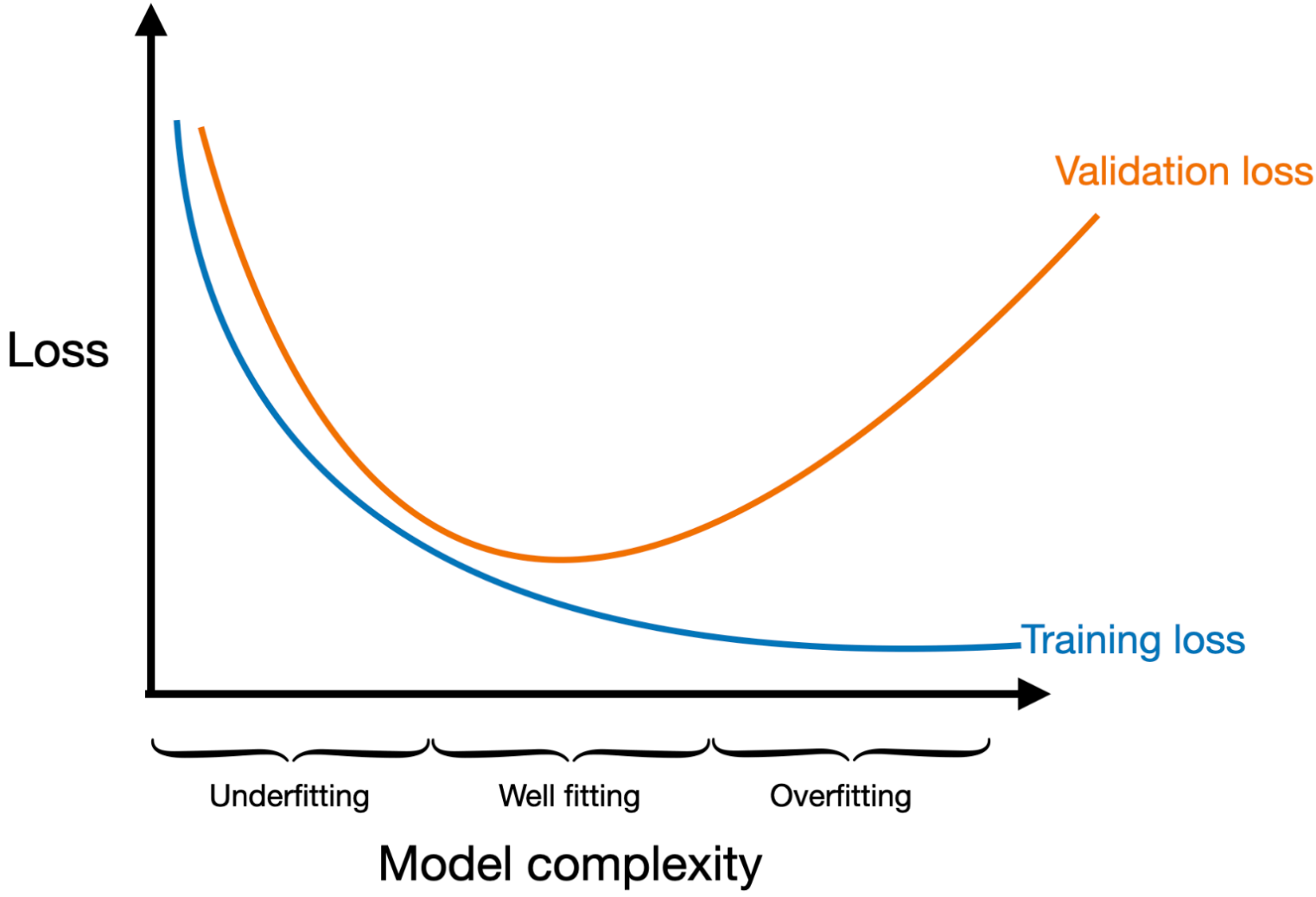
Underfitting

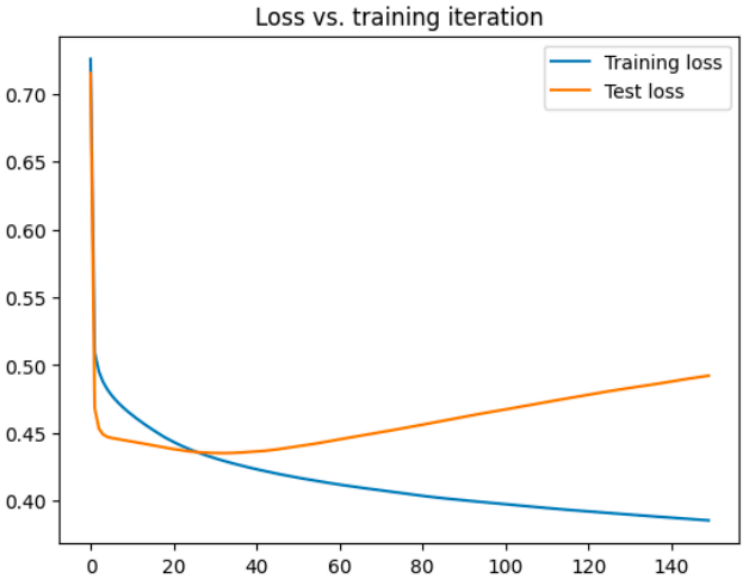


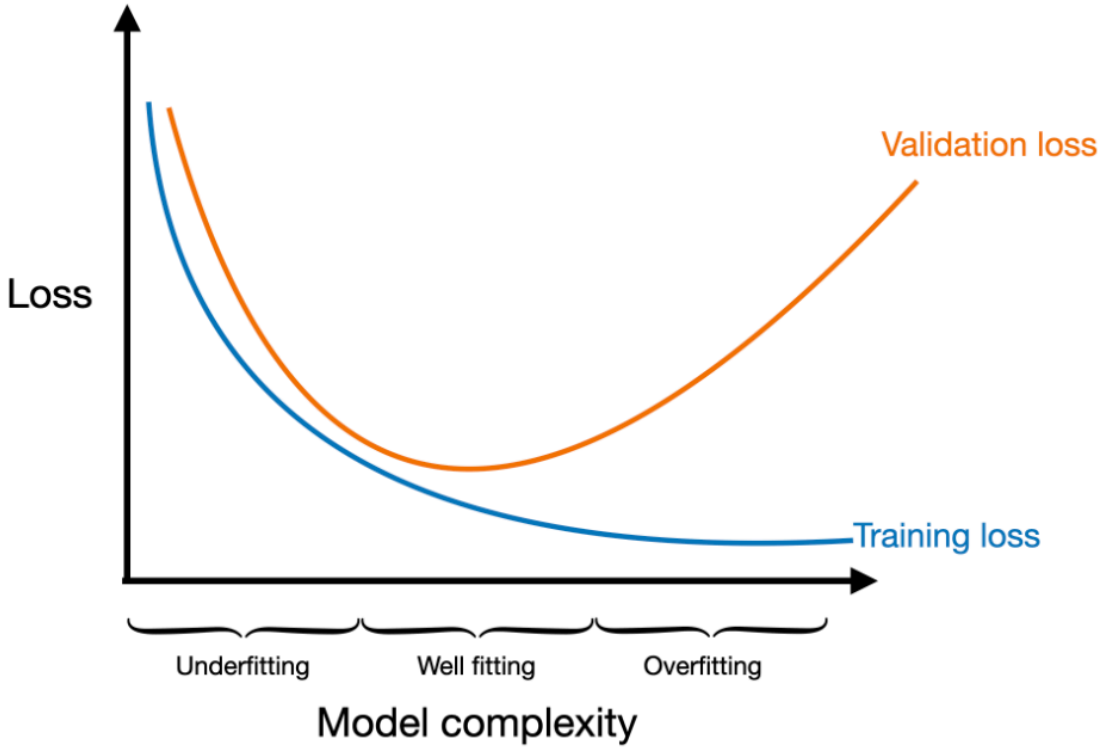
Well-fit

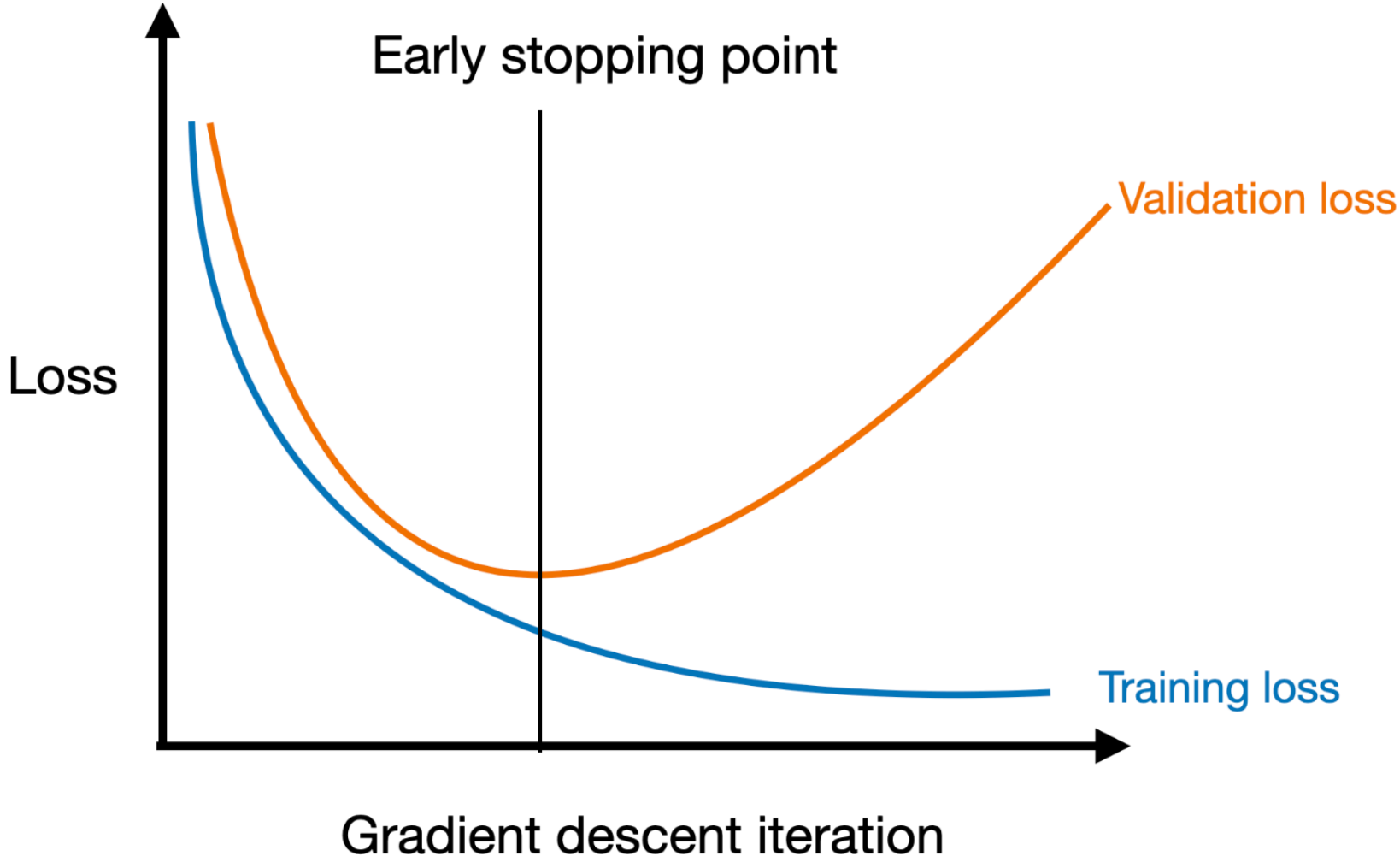


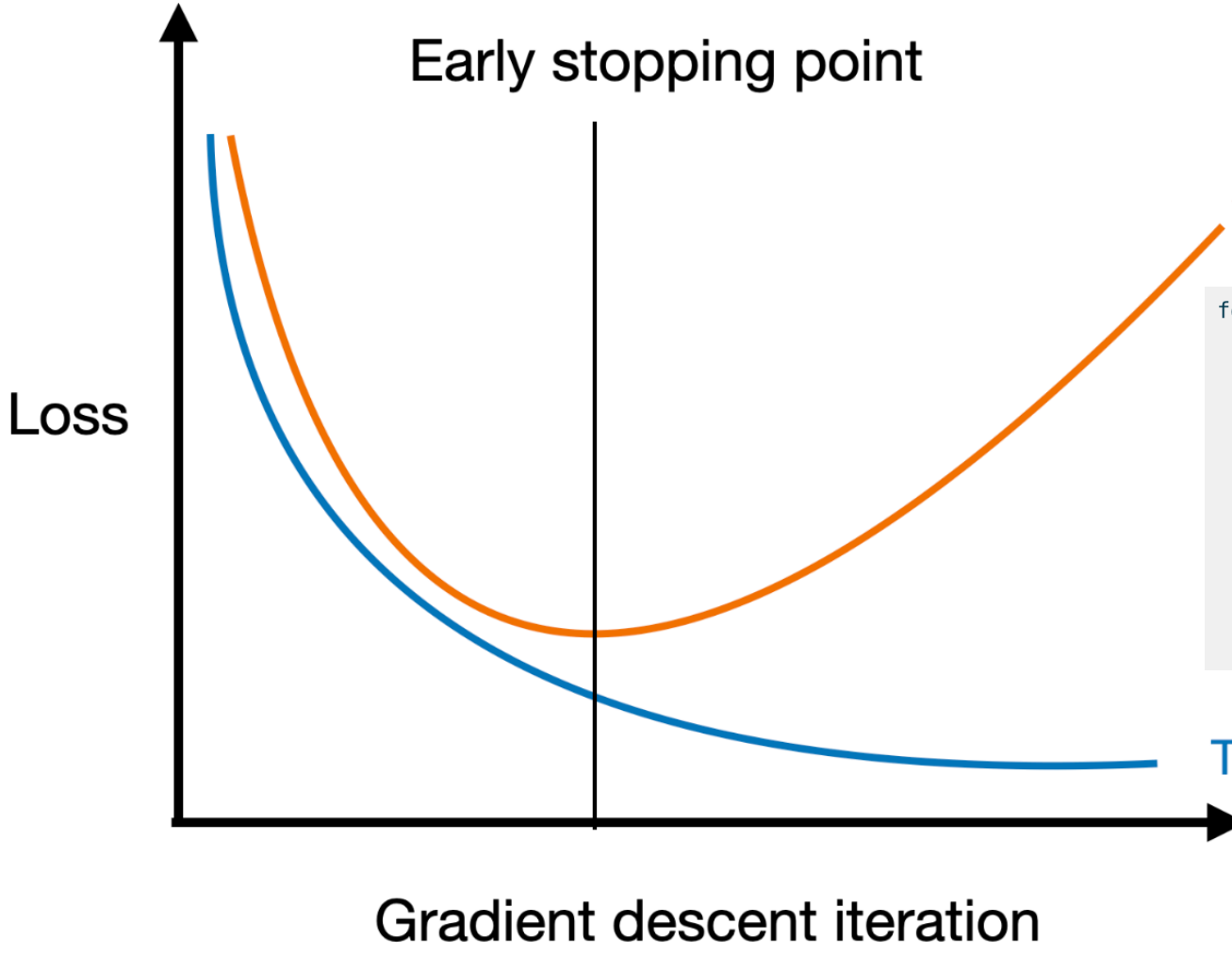
Overfit


Validation loss
Loss
Training loss
Underfitting
Well fitting
Overfitting
Model complexity

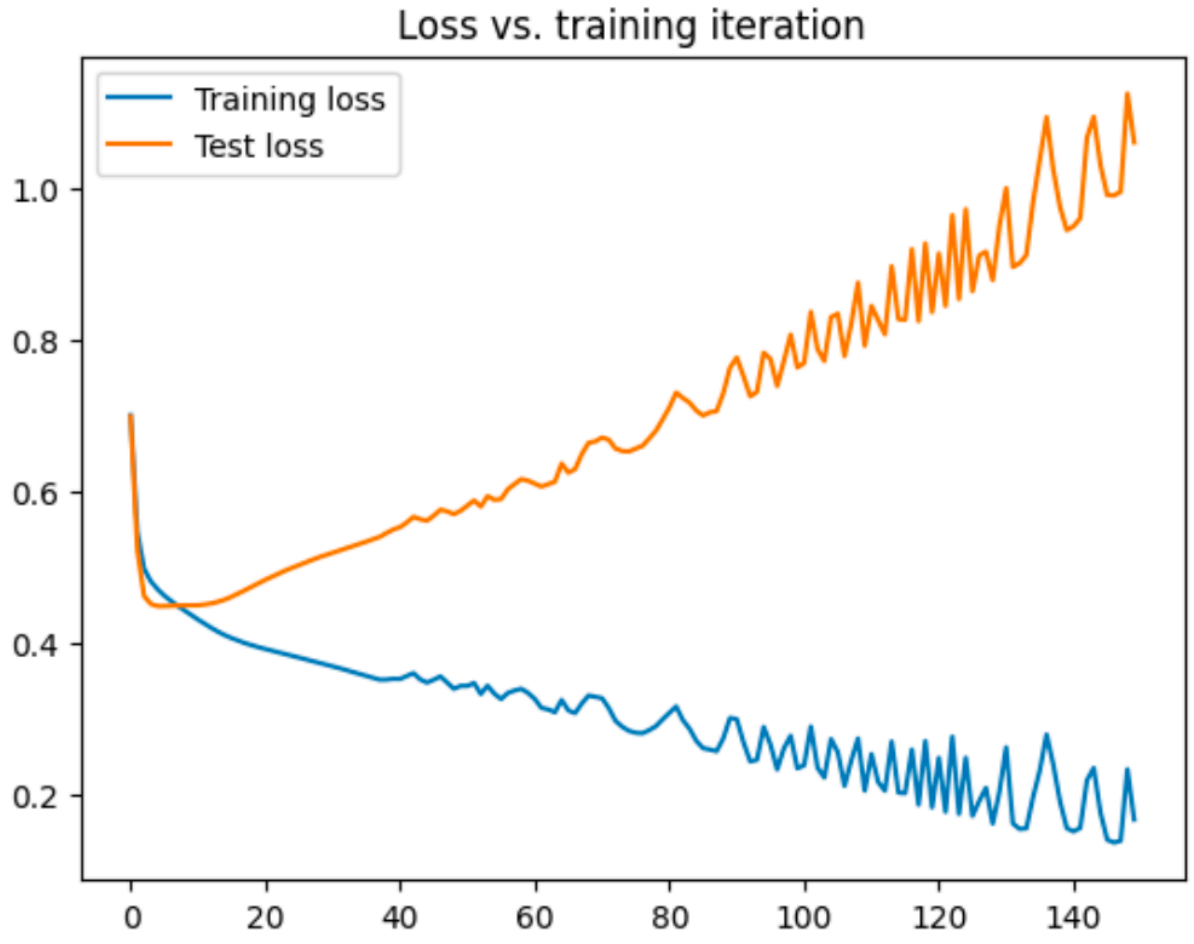

Loss vs. training iteration
Training loss
Test loss
0.70
0.65
0.60
0.55
0.50
0.45
0.40
0
20
40
60
80
100
120
140

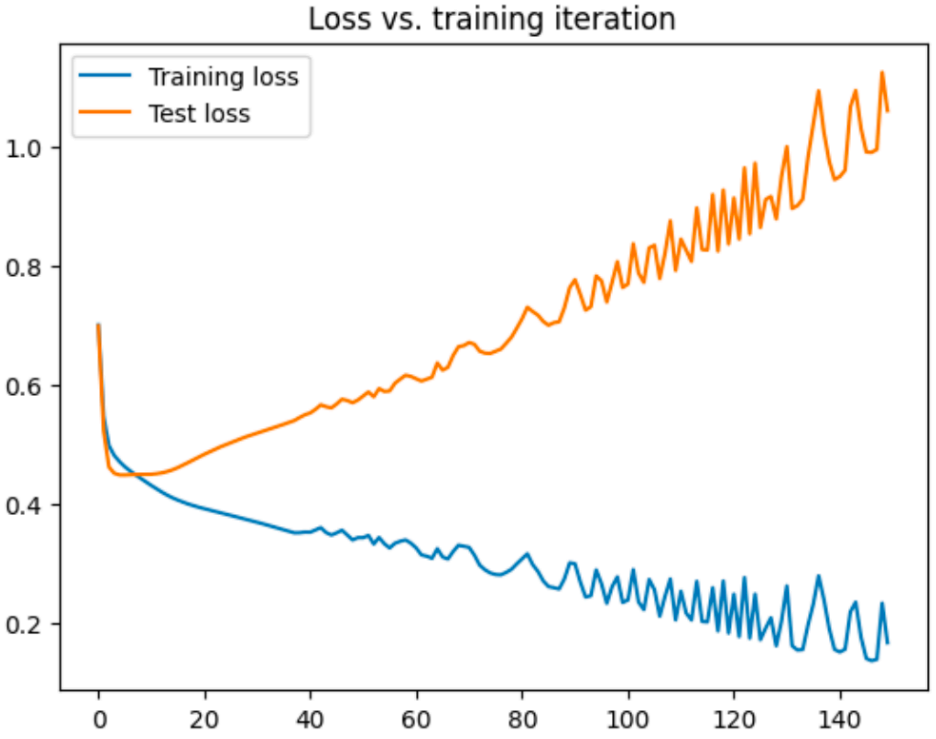

Loss
Validation loss
Training loss
Underfitting
Well fitting
Overfitting
Model complexity


Early stopping point
Validation loss
Loss
Training loss
Gradient descent iteration

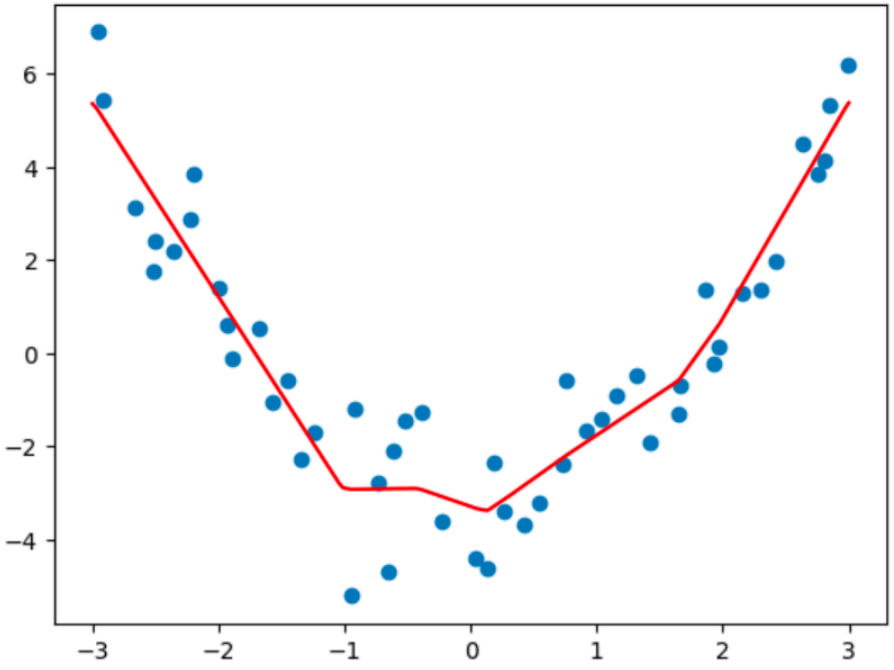



```
for i in range(steps):
    loss = compute_loss(model, training_data)
    loss.backward()
    optimizer.step()
    optimizer.zero_grad()

    valid_loss = compute_loss(model, training_data)
    if valid_loss > valid_losses[-1]:
        break

    valid_losses.append(valid_loss)
```

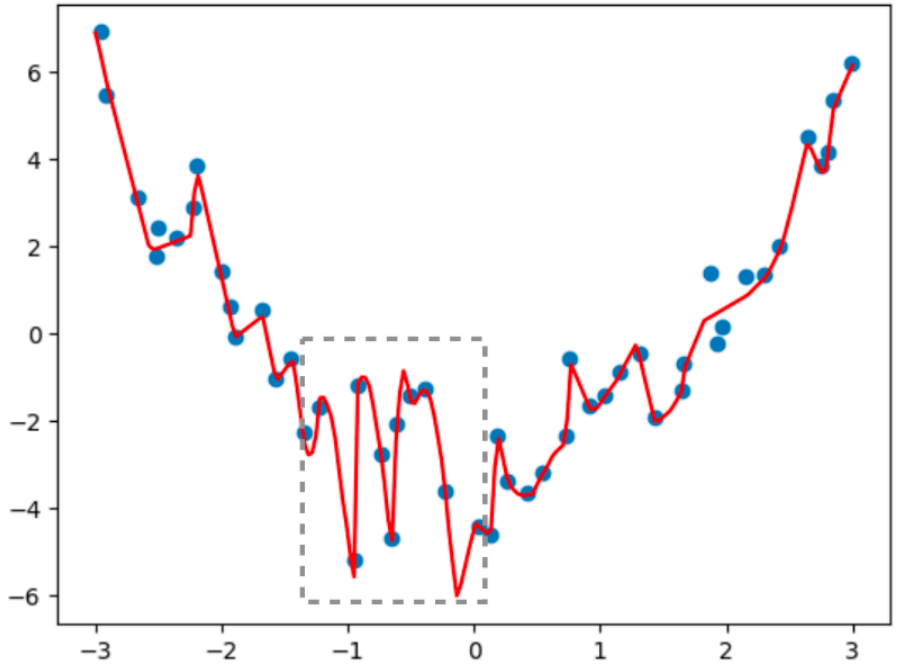

Loss vs. training iteration
Training loss
Test loss
1.0
0.8
0.6
0.4
0.2
0
20
40
60
80
100
120
140




```
patience = 5                    # Number of steps to wait before stopping
steps_since_improvement = 0     # Steps since validation loss improved
min_loss = 1e8                  # Minimum loss seen so far (start large)

for i in range(steps):
    ...

    valid_loss = compute_loss(model, training_data)

    # If the validation loss improves reset the counter
    if valid_loss < min_loss:
        steps_since_improvement = 0
        min_loss = valid_loss

    # Otherwise increment the counter
    else:
        steps_since_improvement += 1

    # If its been patience steps since the last improvement, stop
    if steps_since_improvement == patience:
        break
```

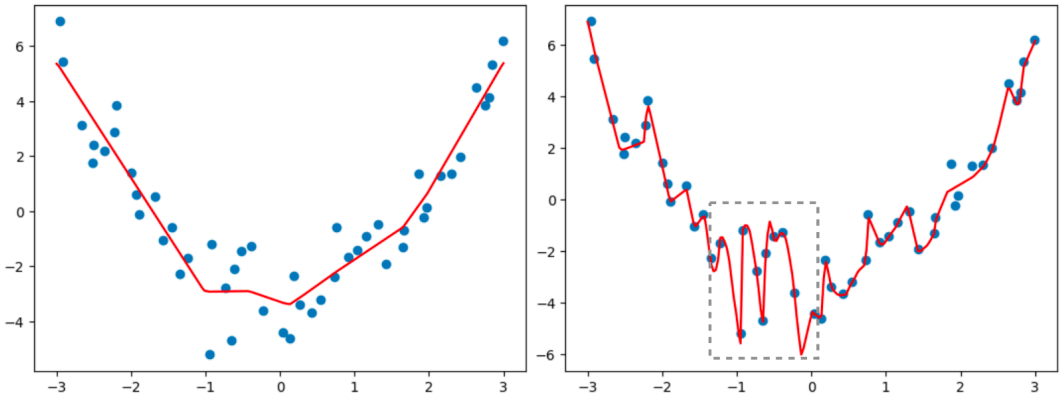


Well-fit

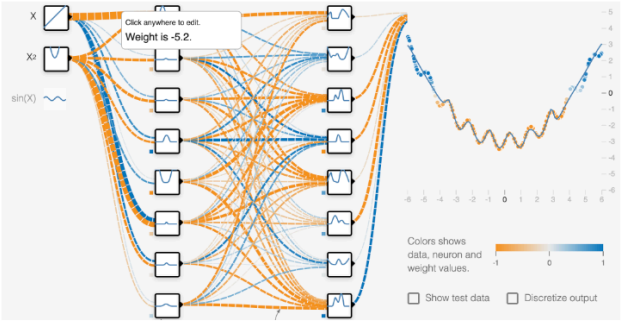


Overfit

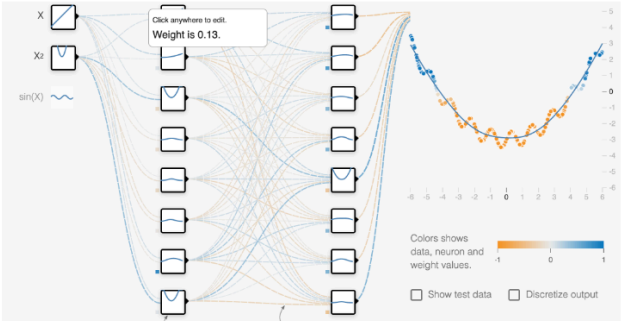
$$\mathbf{MSE}(\mathbf{X}, \mathbf{y}, \mathbf{w}) = \frac{1}{N} \sum_{i=1}^{N} ((f(\mathbf{x}_i, \mathbf{w}) - y_i)^2)$$



Well-fit Overfit



An overfit network will have large weights to encode large slopes.



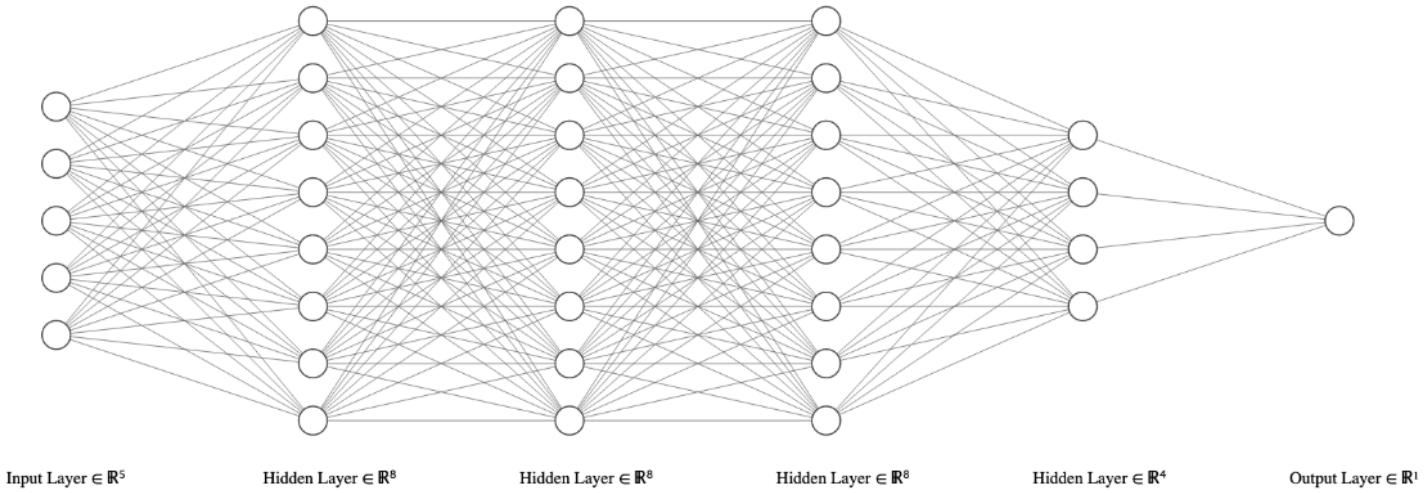
A regularized network will have smaller weights encoding a smooth function.

$$\mathbf{L}_2(\mathbf{w}) = \|\mathbf{w}\|_2^2 = \sum_{i=1}^{d} w_i^2$$

$$\mathbf{L}_2(\mathbf{W}) = \|\mathbf{W}\|_2^2 = \sum_{i=1}^{d} \sum_{j=1}^{e} w_{ij}^2$$

$$\mathbf{Loss}(\mathbf{X}, \mathbf{y}, \mathbf{w}) = \mathbf{MSE}(\mathbf{X}, \mathbf{y}, \mathbf{w}) + \lambda \mathbf{L}_2(\mathbf{w})$$

$$f(\mathbf{x}, \mathbf{w}_0, \dots) = \sigma(\sigma(\sigma(\sigma(\mathbf{x}^\mathbf{T}\mathbf{W}_4)^T\mathbf{W}_3)^T\mathbf{W}_2)^T\mathbf{W}_1)^T\mathbf{w}_0$$



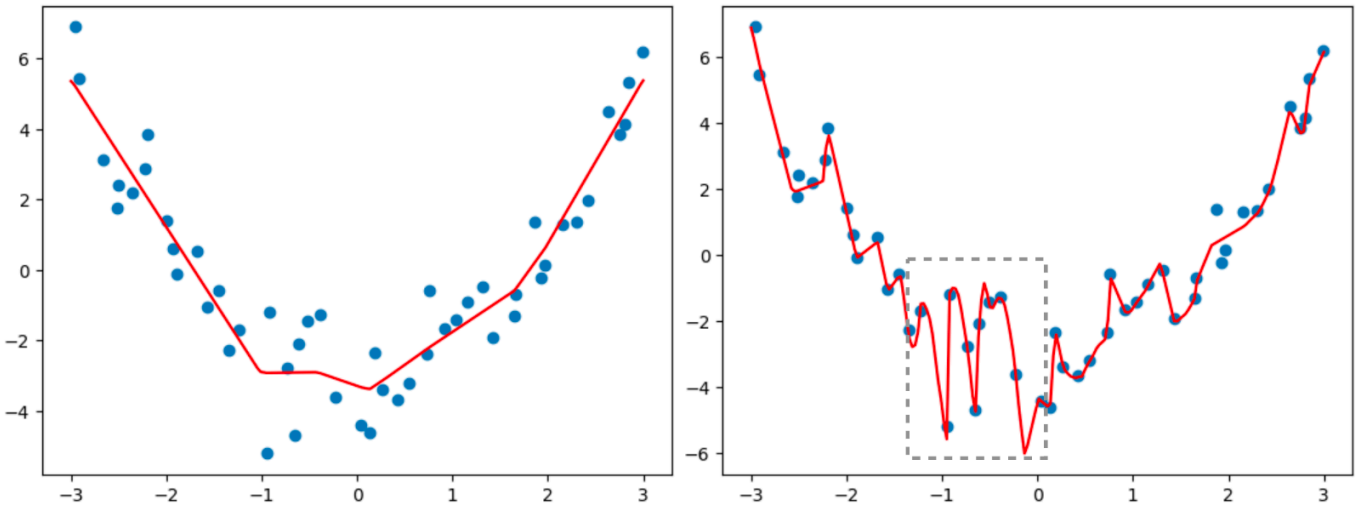
$$\mathbf{L}_2(\mathbf{w}_0, \mathbf{W}_1, \dots, \mathbf{W}_L) = \sum_{l=0}^{L} \|\mathbf{W}_l\|_2^2$$

In practice most networks also incorporate bias terms, so each linear function in our network can be written as:

$$\mathbf{x}^T\mathbf{W} + \mathbf{b}$$

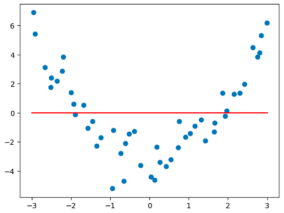
And the full prediction function for a sigmoid-activation network might be:

$$f(\mathbf{x}, \mathbf{w}_0, \dots) = \sigma(\sigma(\sigma(\mathbf{x}^\mathbf{T}\mathbf{W}_4 + \mathbf{b}_4)^T\mathbf{W}_3 + \mathbf{b}_3)^T\mathbf{W}_2 + \mathbf{b}_2)^T\mathbf{W}_1 + \mathbf{b}_1)^T\mathbf{w}_0 + \mathbf{b}_0$$

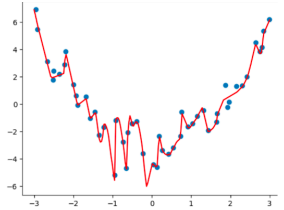


Well-fit Overfit

Does the bias contribute to overfitting?

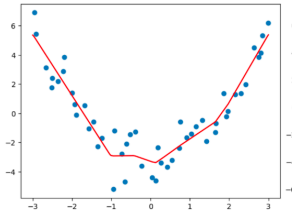


Underfitting



Overfit

Validation loss



Well-fit

Loss

Training loss

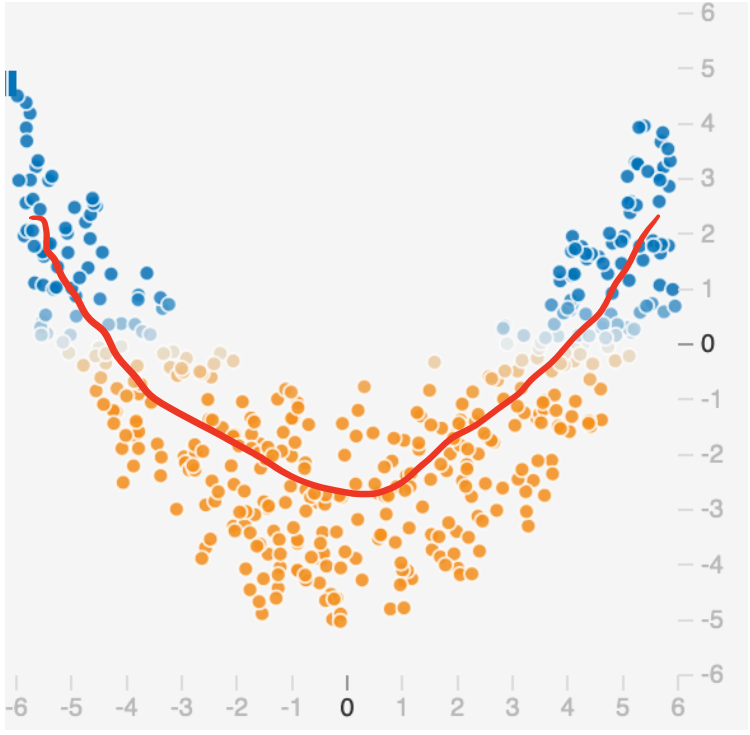
Underfitting

Well fitting

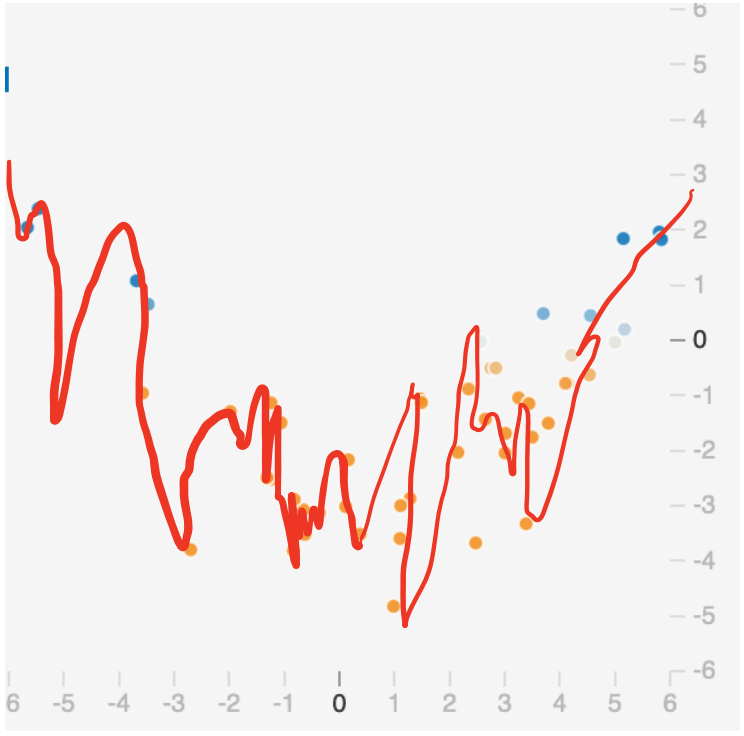
Overfitting

Model complexity

Sampling data

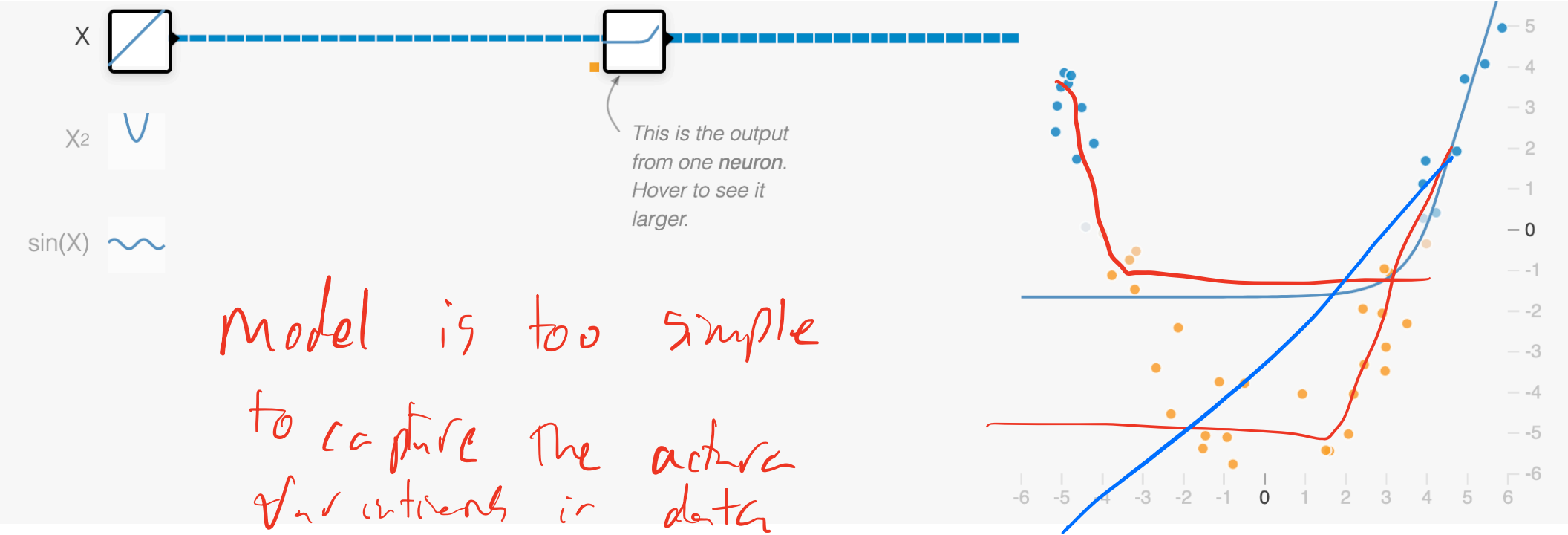


Data distribution

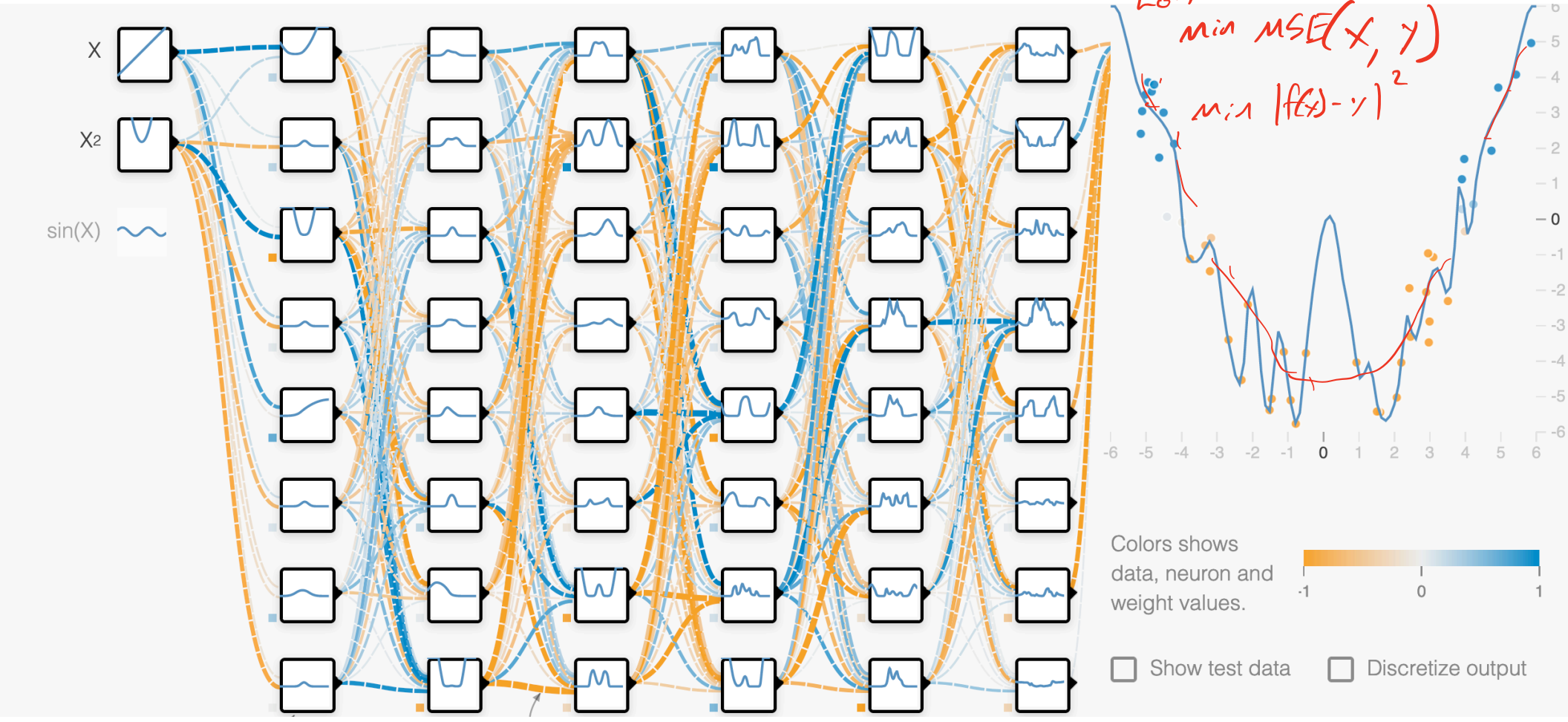


Training data

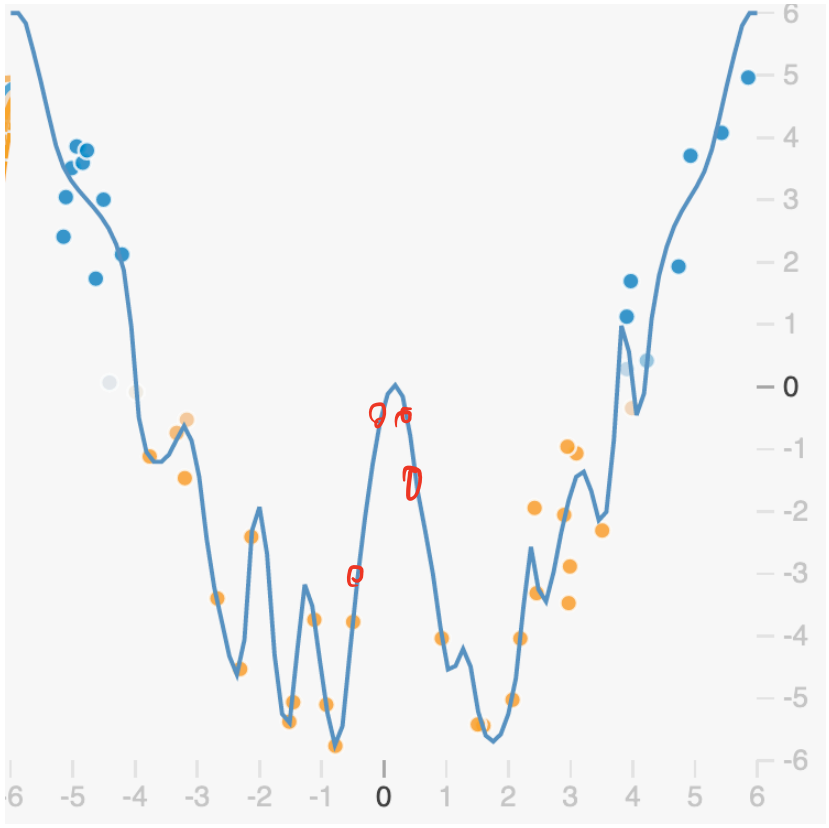
Underfitting

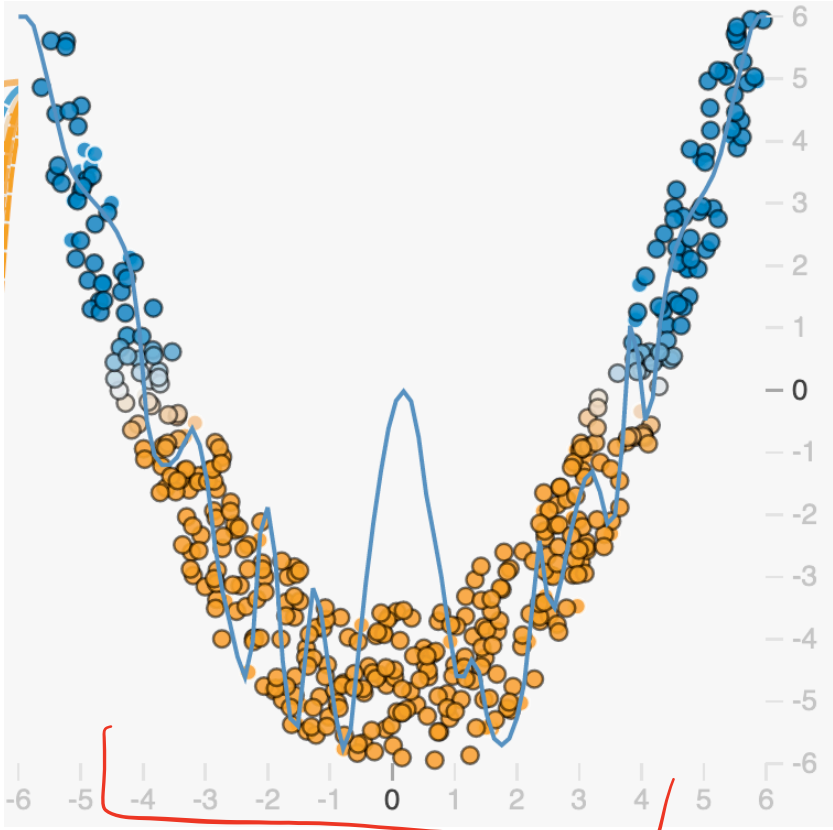


Overfitting

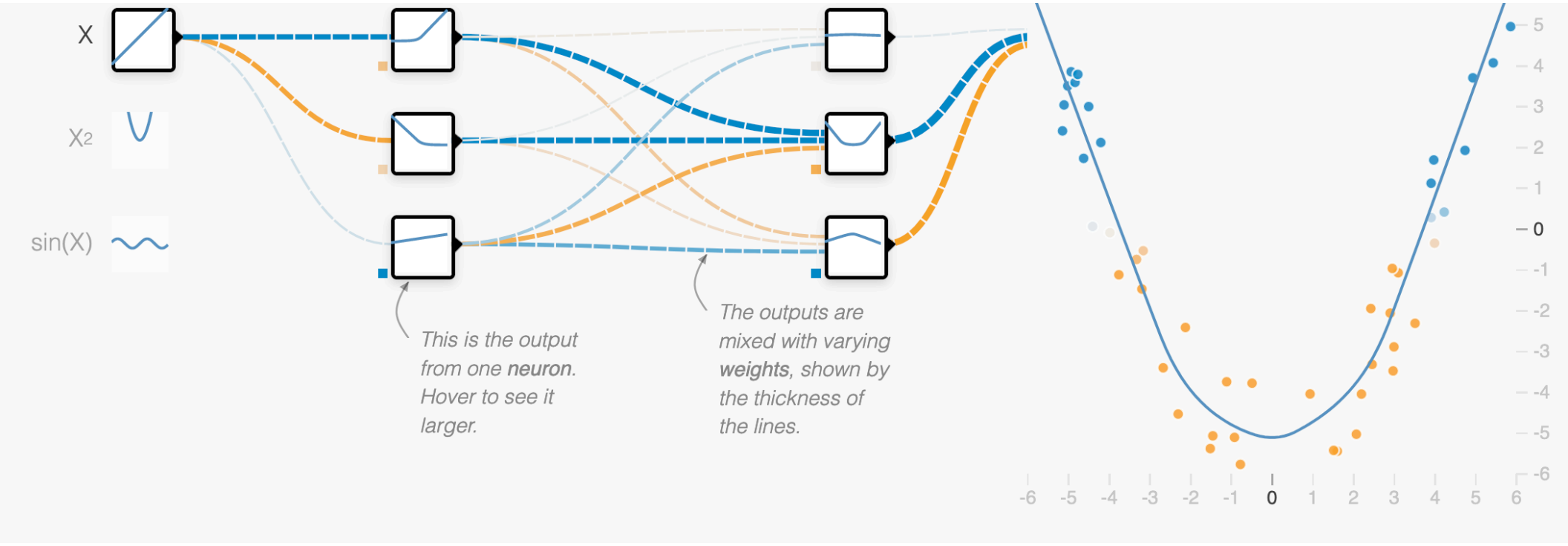


Overfitting

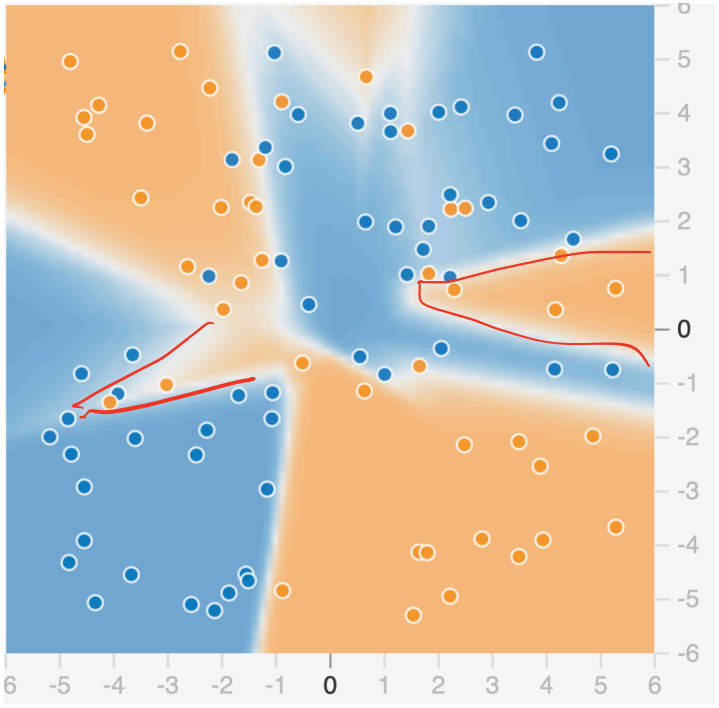


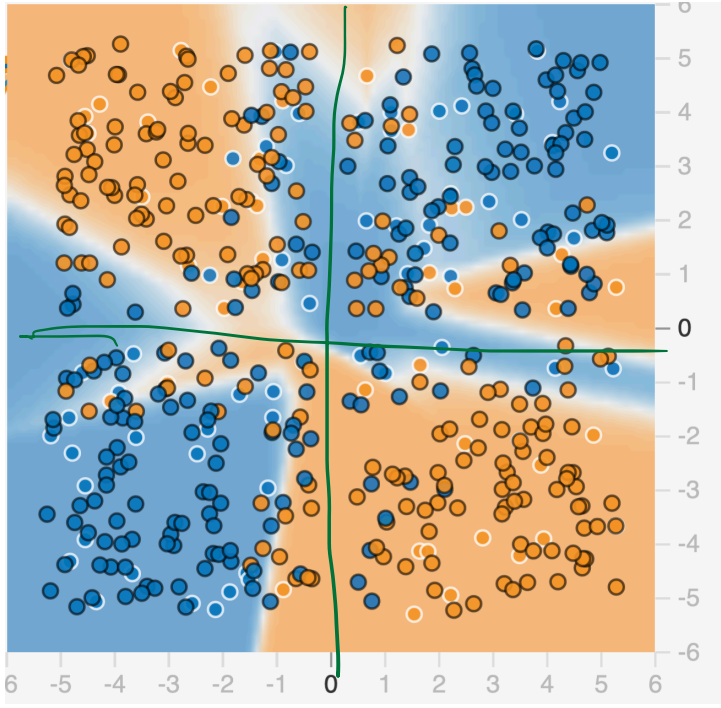


Good fit

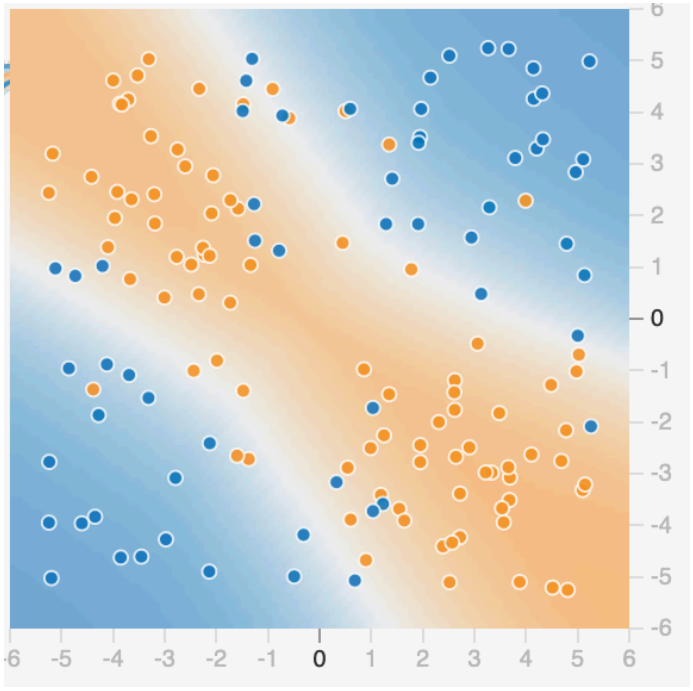

X
X2
sin(X)
This is the output from one neuron. Hover to see it larger.
The outputs are mixed with varying weights, shown by the thickness of the lines.
5
4
3
2
1
0
-1
-2
-3
-4
-5
-6
-6 -5 -4 -3 -2 -1 0 1 2 3 4 5 6

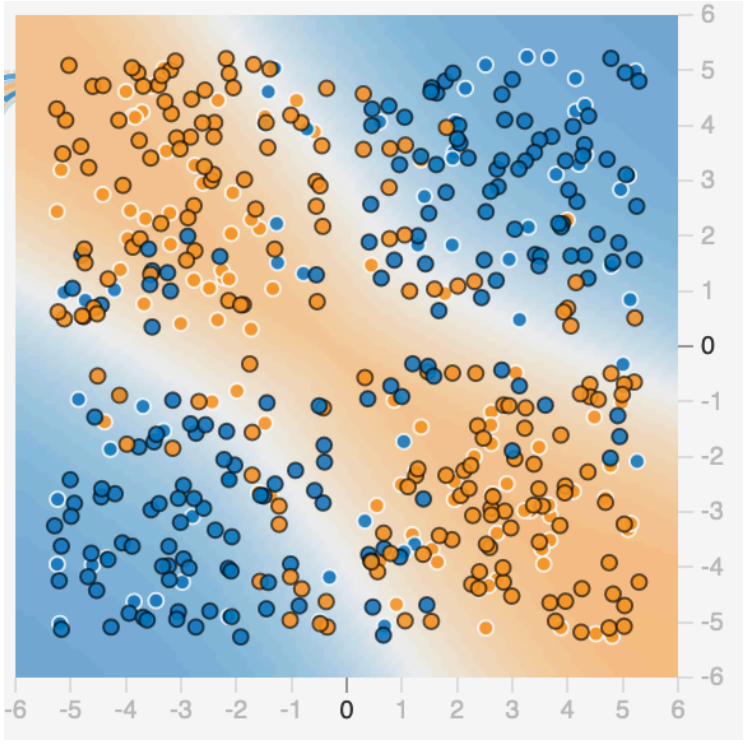
Overfitting



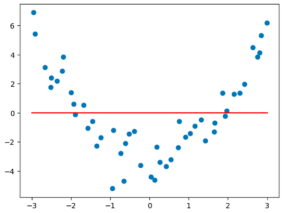


Better fit

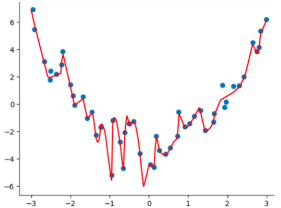

6
5
4
3
2
1
0
-1
-2
-3
-4
-5
-6
-6 -5 -4 -3 -2 -1 0 1 2 3 4 5 6


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

Model complexity curve



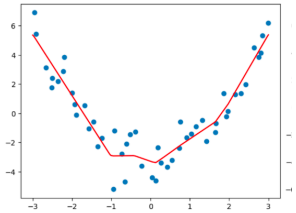
Underfitting



Overfit

Validation loss

Loss

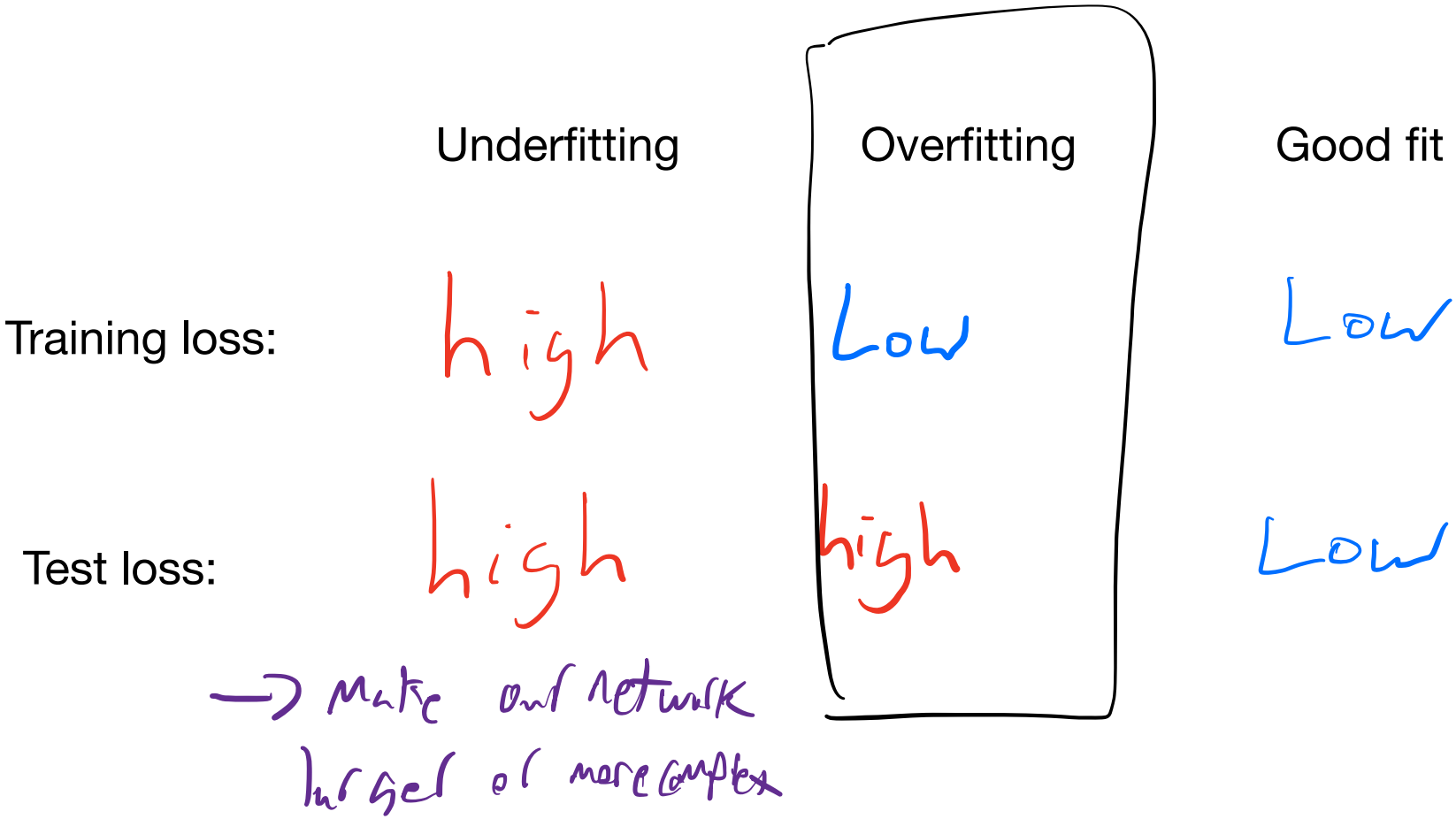


Well-fit

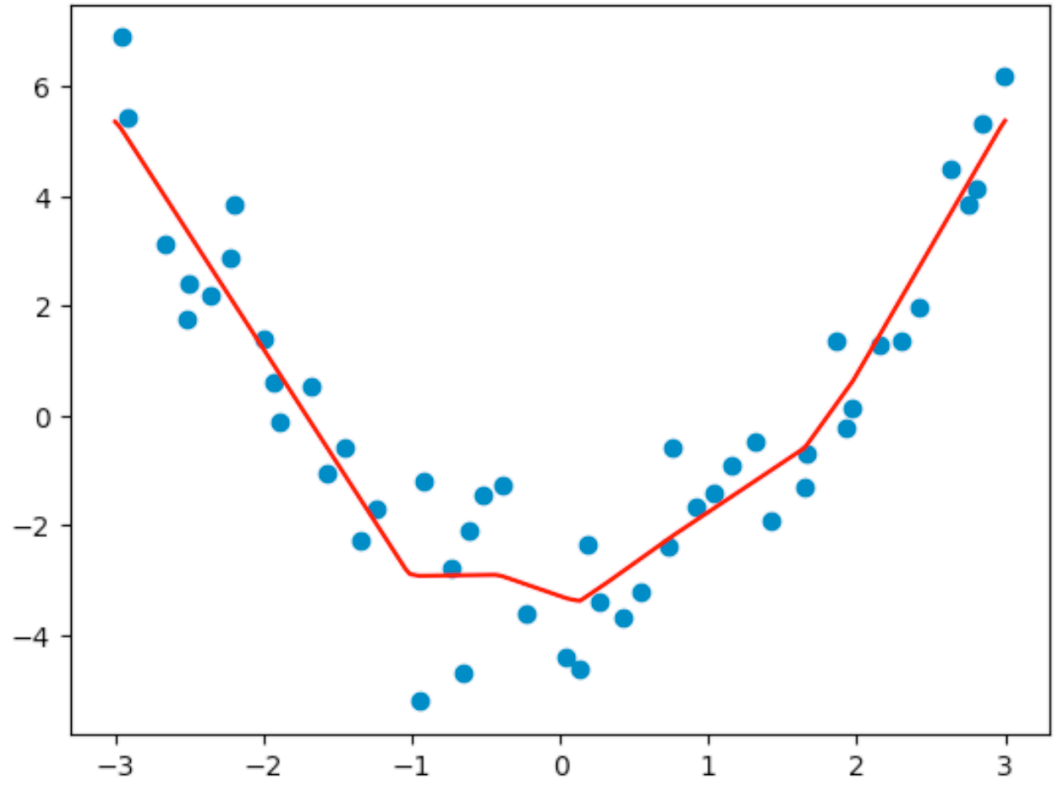
Training loss

Underfitting | Well fitting | Overfitting

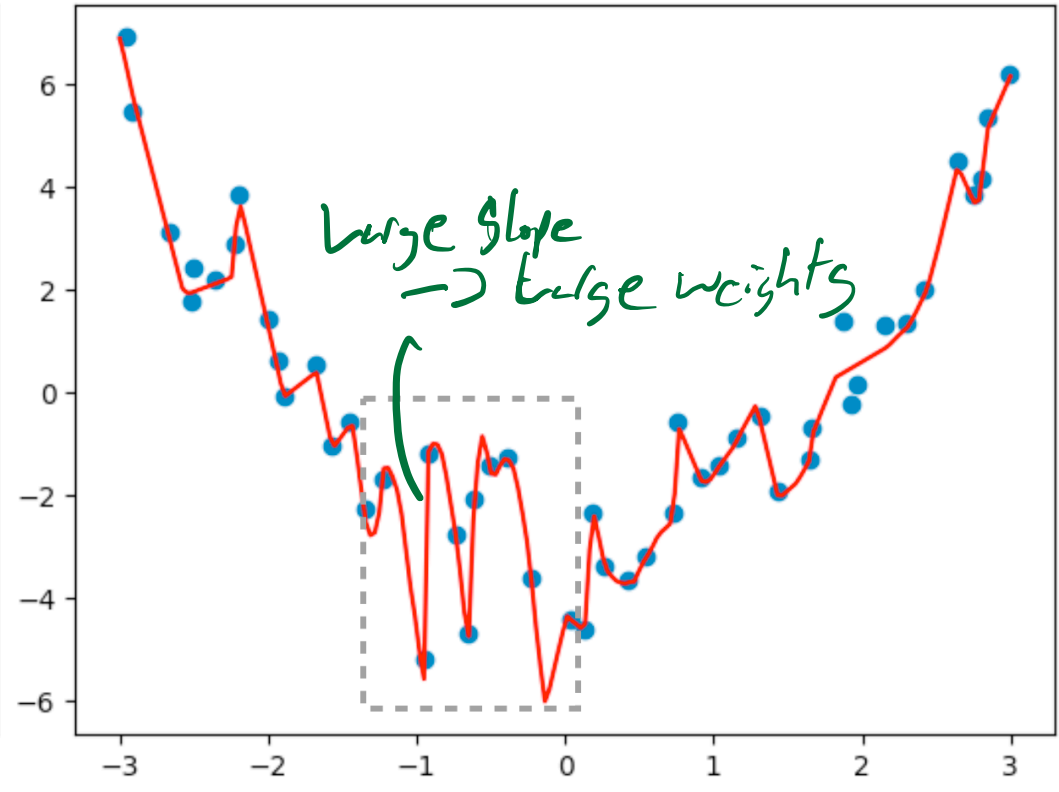
Model complexity


Underfitting
Overfitting
Good fit
Training loss:
high
Low
Low
Test loss:
high
high
Low
→ Make our network
larger or more complex

Overfitting with high weights

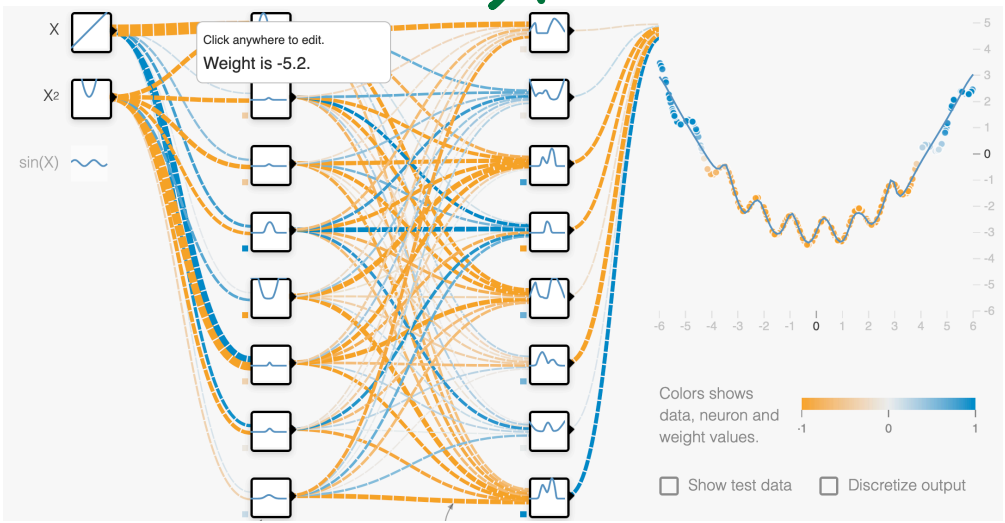


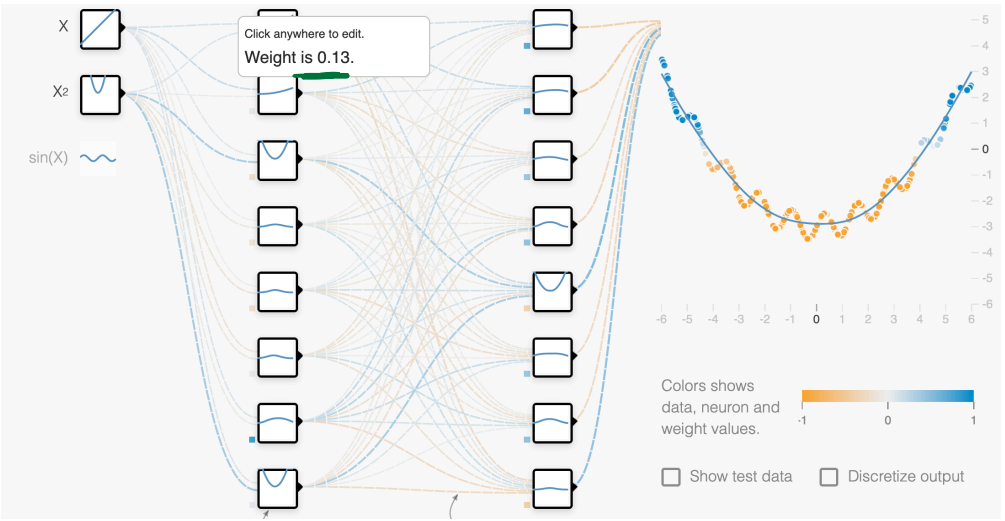
Well-fit



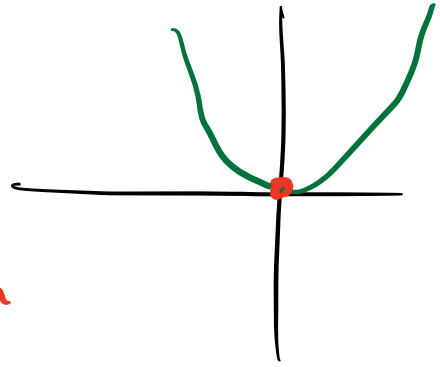
Overfit

Overfitting with high weights



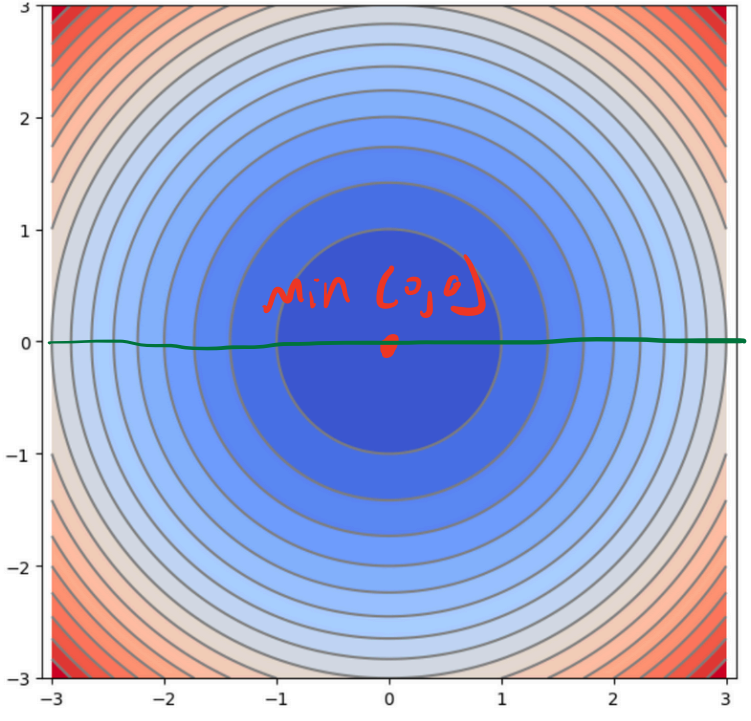


L2 Regularization



L2-Loss

$$\ell^2 = w_1^2 + w_2^2$$



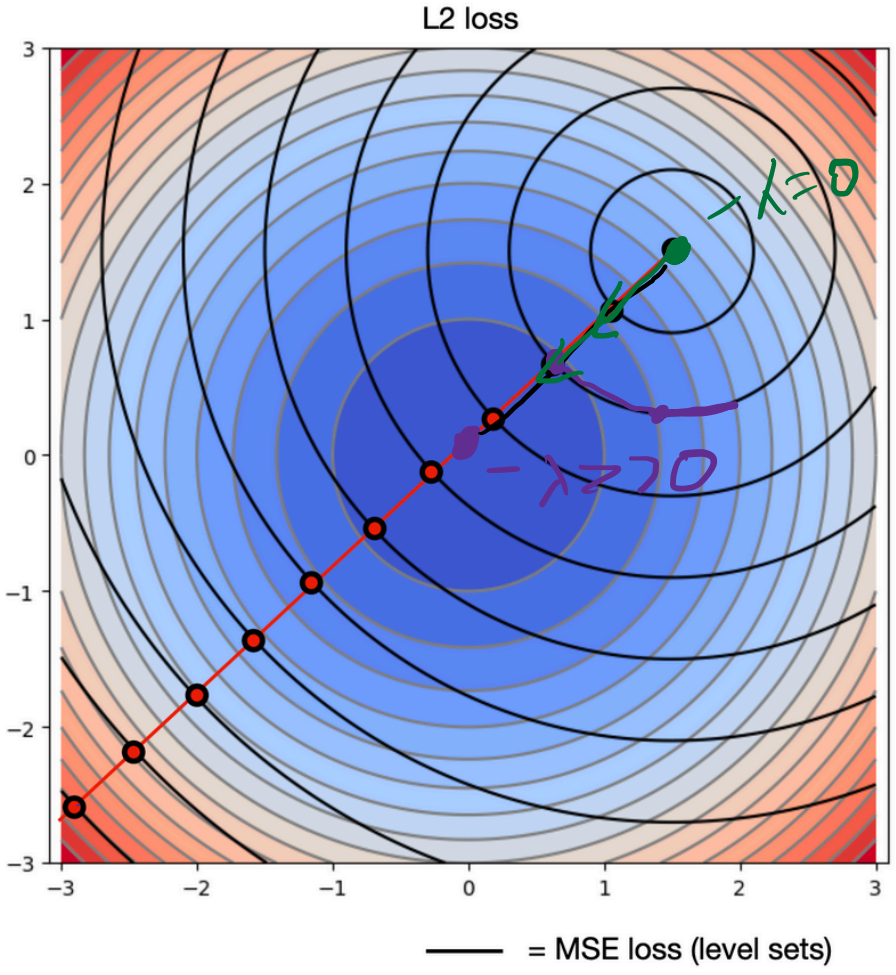
Squared L2 Norm

Min $\mathbf{L}_2(\mathbf{W}) = \|\mathbf{W}\|_2^2 = \sum_{i=1}^{d}\sum_{j=1}^{e} w_{ij}^2$

minimized where $w_{ij} = 0$

$\forall\, i, j$

L2 Regularization



$$\mathbf{L_2(W)} = \|\mathbf{W}\|_2^2 = \sum_{i=1}^{d}\sum_{j=1}^{e} w_{ij}^2$$

$$\mathbf{Loss(X, y, w)} = \mathbf{MSE(X, y, w)} + \lambda \mathbf{L_2(w)}$$

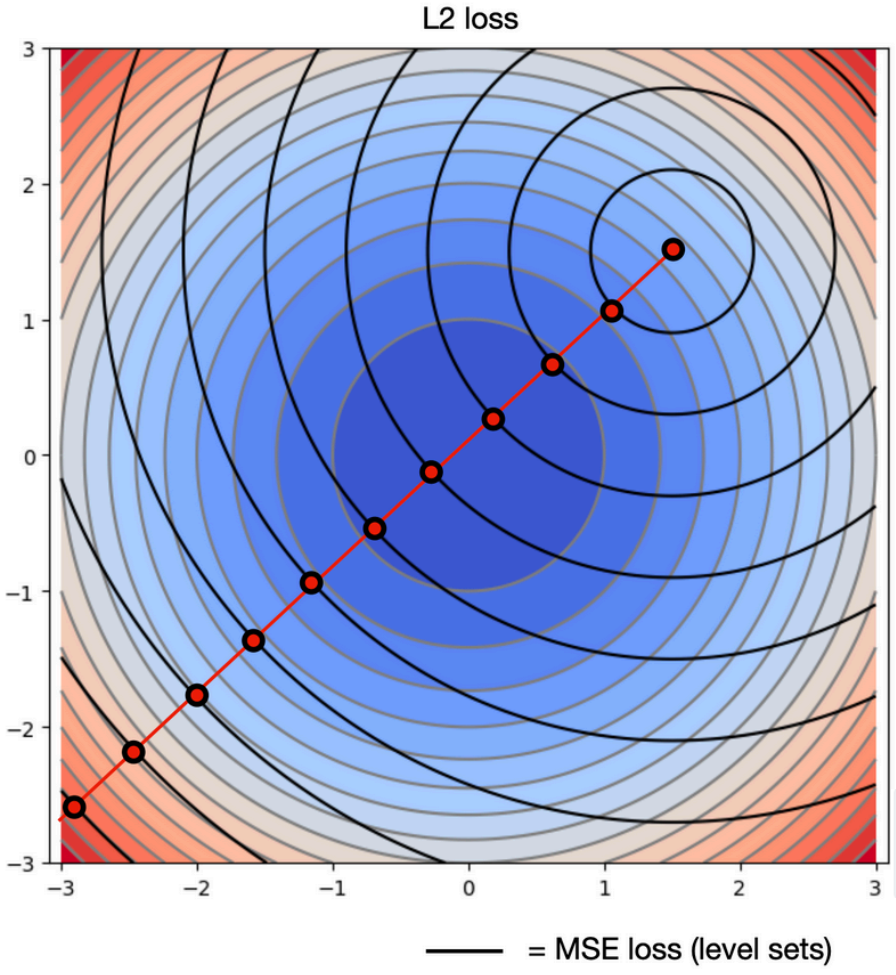
Reg.

Scaling parameter

makes good predic

has small weights

L2 Regularization



$$\mathbf{L}_2(\mathbf{W}) = \|\mathbf{W}\|_2^2 = \sum_{i=1}^{d}\sum_{j=1}^{e} w_{ij}^2$$

$$\mathbf{Loss}(\mathbf{X}, \mathbf{y}, \mathbf{w}) = \mathbf{MSE}(\mathbf{X}, \mathbf{y}, \mathbf{w}) + \lambda \mathbf{L}_2(\mathbf{w})$$

```
from torch import optim
optimizer = optim.SGD(model.parameters(), lr=0.1, weight_decay=0.01)
```

L1 Regularization

L2

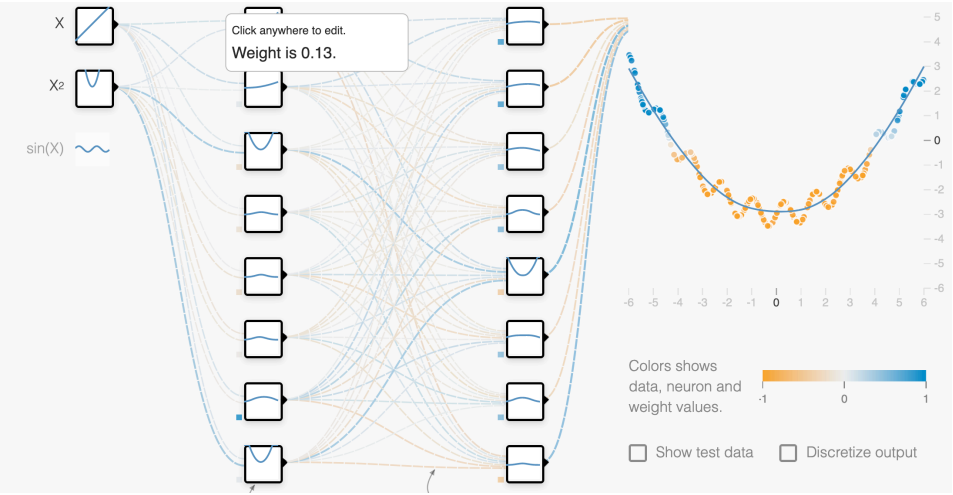
$$\text{Vector: } \mathbf{L}_2(\mathbf{w}) = \|\mathbf{w}\|_2^2 = \sum_{i=1}^{d} w_i^2 \qquad \text{Matrix: } \mathbf{L}_2(\mathbf{W}) = \|\mathbf{W}\|_2^2 = \sum_{i=1}^{d}\sum_{j=1}^{e} w_{ij}^2$$

L1

$$\text{Vector: } \mathbf{L}_1(\mathbf{w}) = \|\mathbf{w}\|_1 = \sum_{i=1}^{d} |w_i|, \qquad \text{Matrix: } \mathbf{L}_1(\mathbf{W}) = \|\mathbf{W}\|_1 = \sum_{i=1}^{d}\sum_{j=1}^{e} |w_{ij}|$$

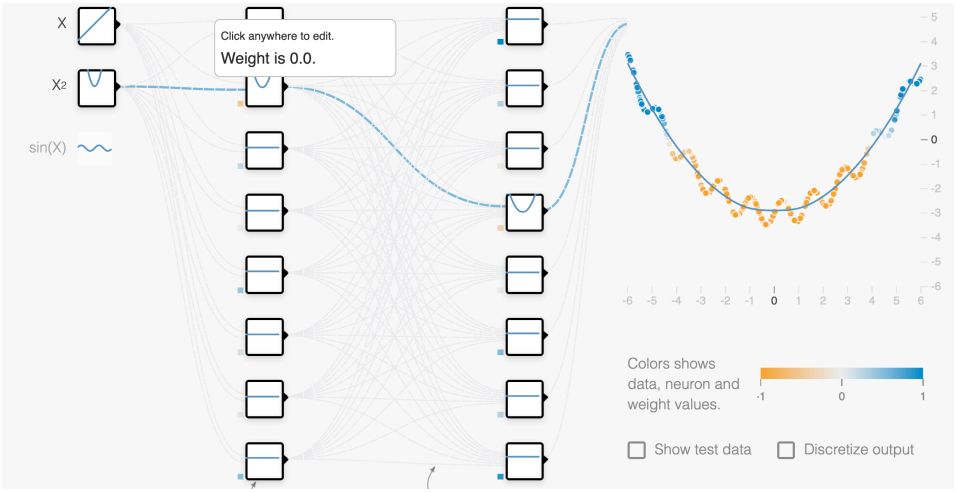
L1 Regularization

L2



All weights are small

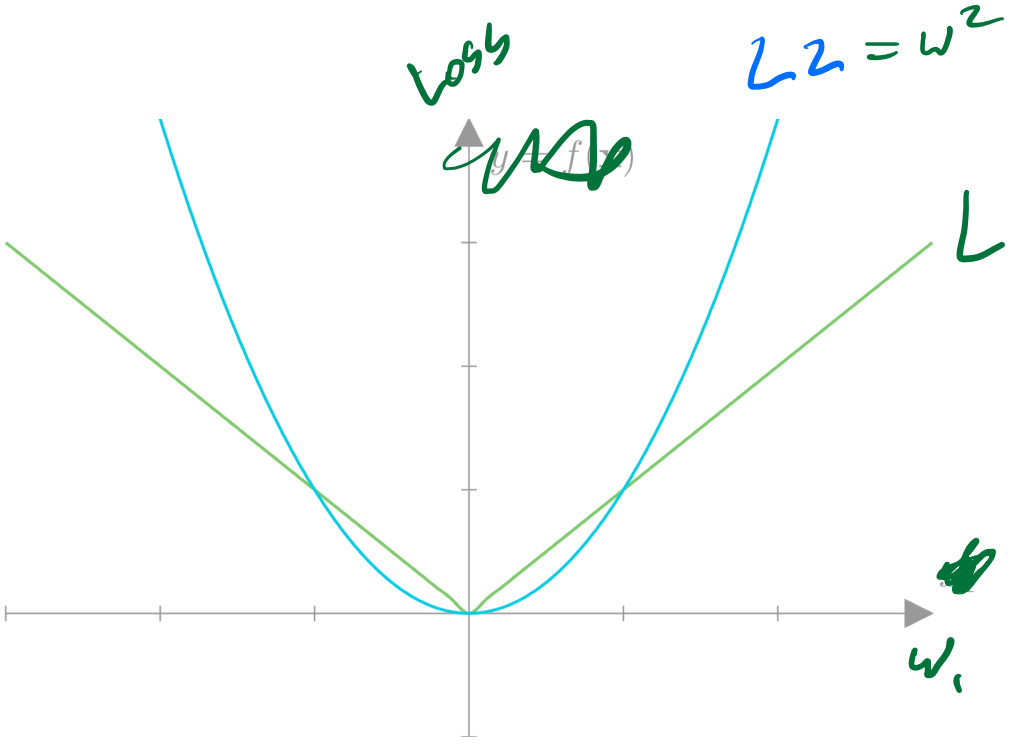
L1



Sparsity

most weights are 0

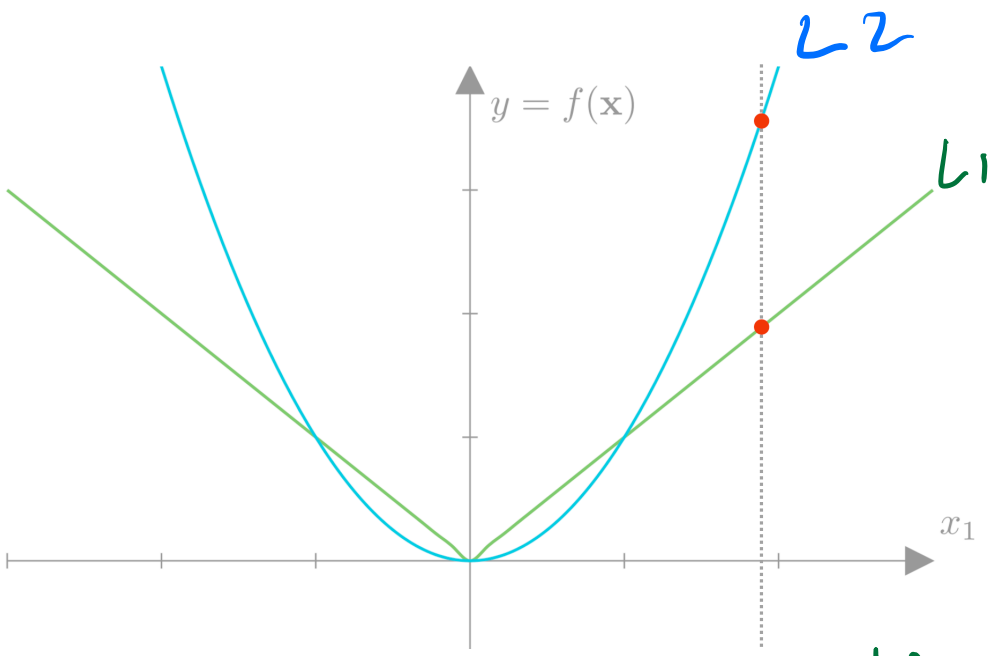
L1 Vs. L2 Regularization



$$\text{Vector: } \mathbf{L}_2(\mathbf{w}) = \|\mathbf{w}\|_2^2 = \sum_{i=1}^{d} w_i^2$$

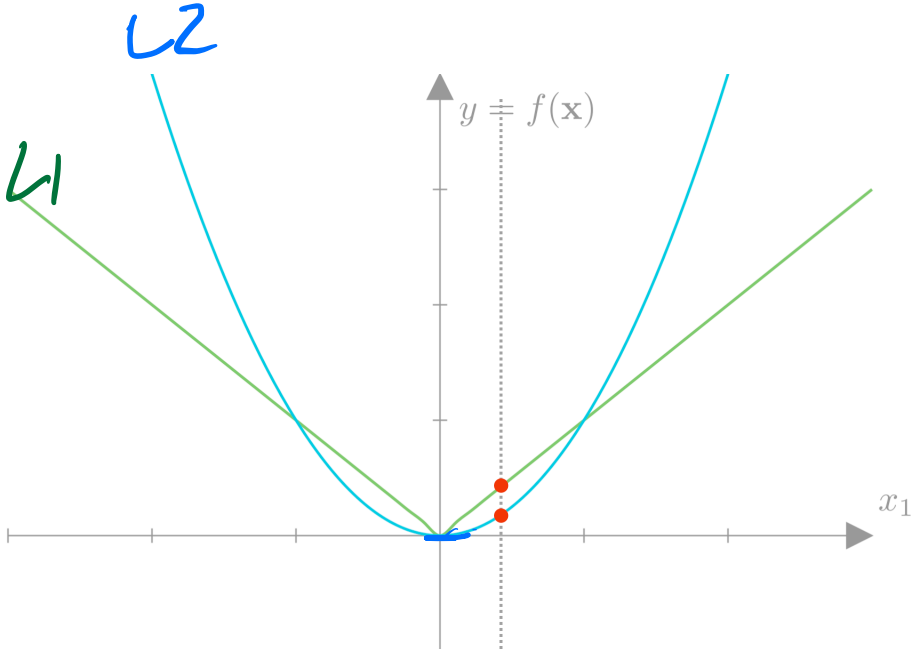
$$\text{Vector: } \mathbf{L}_1(\mathbf{w}) = \|\mathbf{w}\|_1 = \sum_{i=1}^{d} |w_i|,$$

L1 Vs. L2 Regularization



L2 Loss grows quickly away from 0

L1 Loss grows consistently



L2 Loss flat near 0

L1 Loss remains consistent

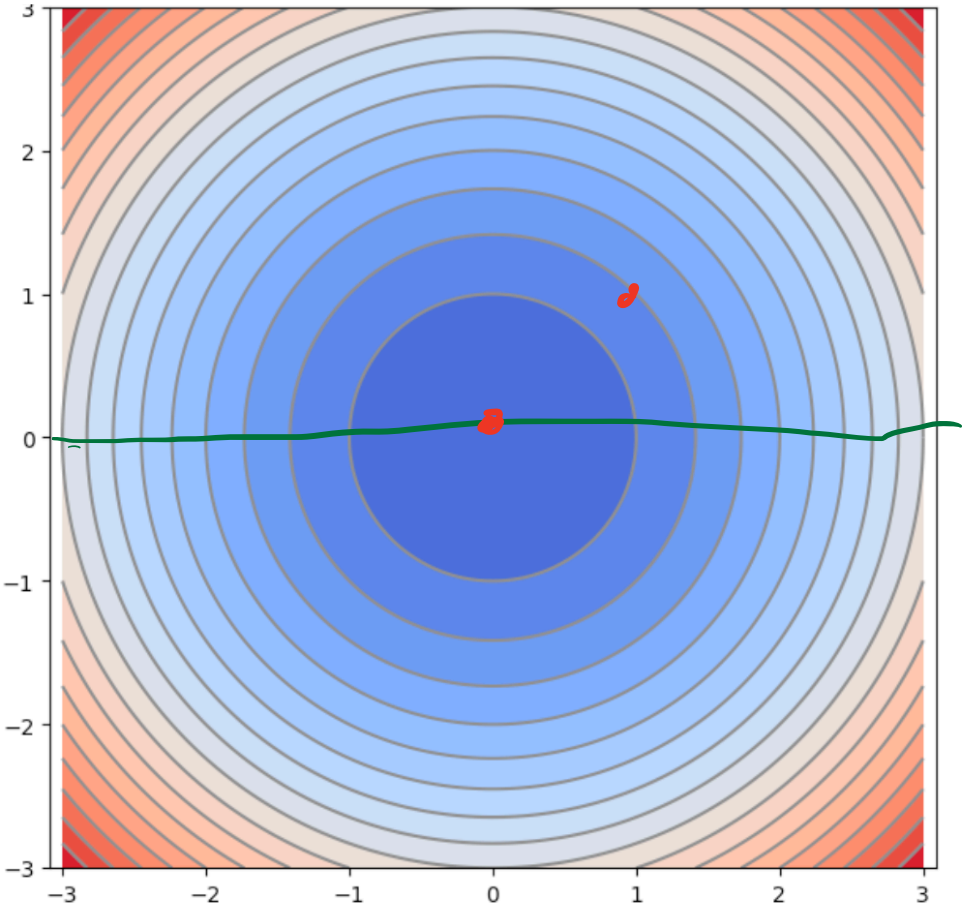
L1 Vs. L2 Regularization

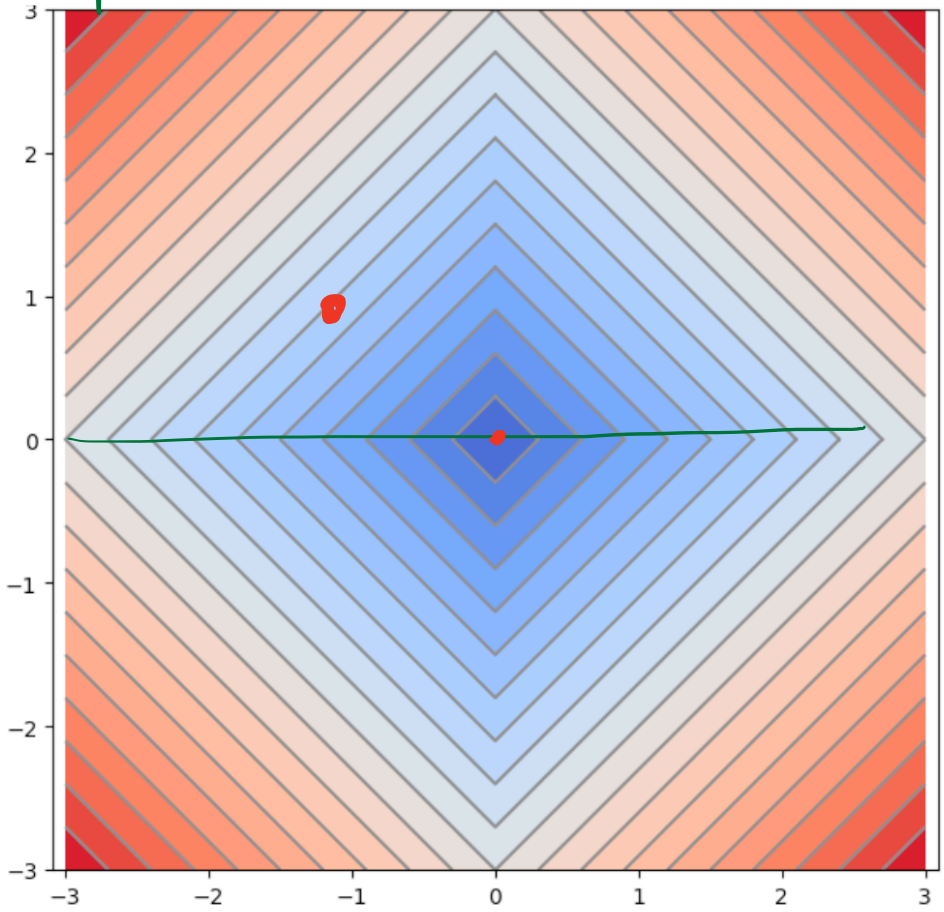
L2-Loss

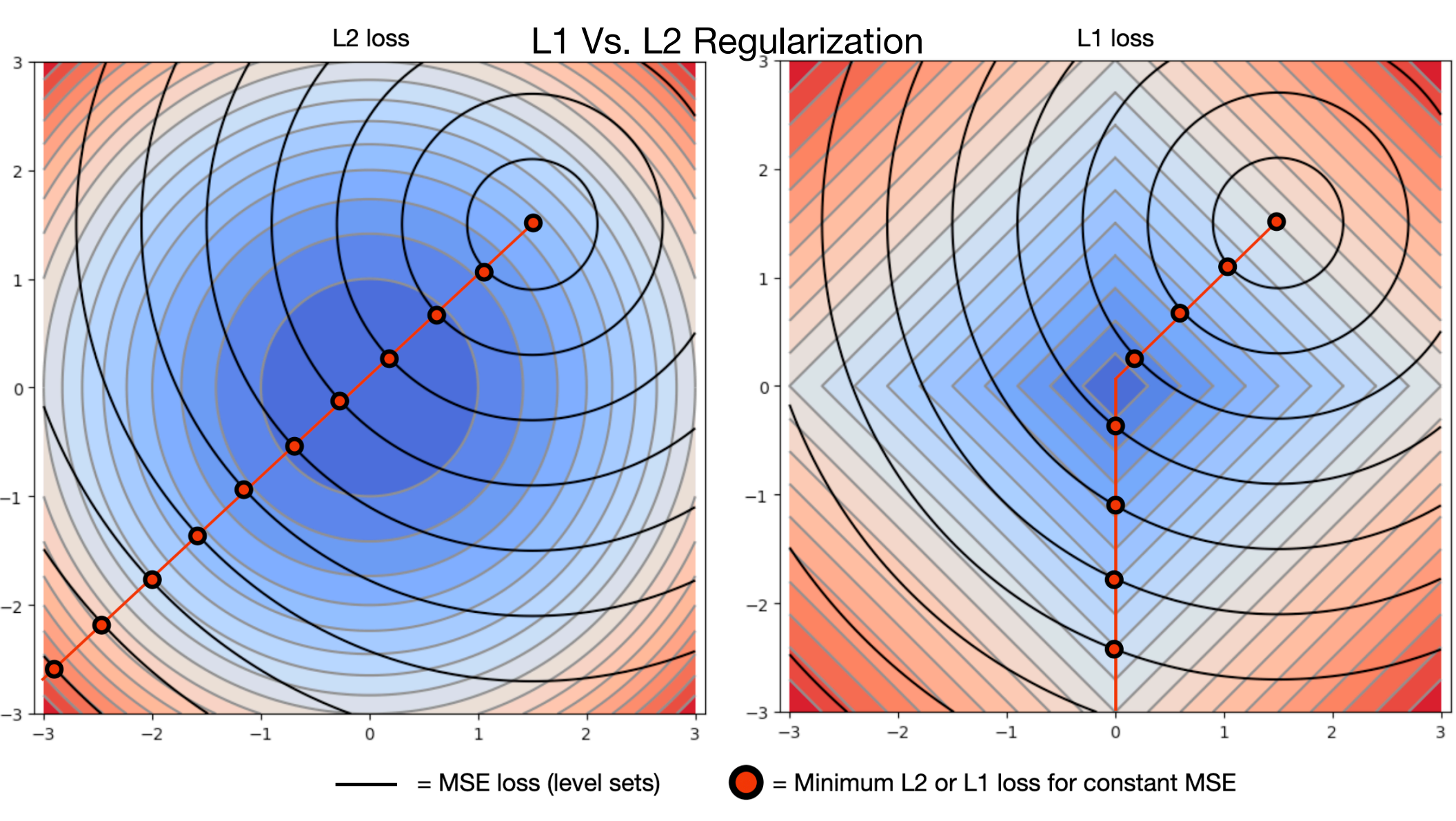
$$\ell^2 = w_1^2 + w_2^2$$

L1-Loss

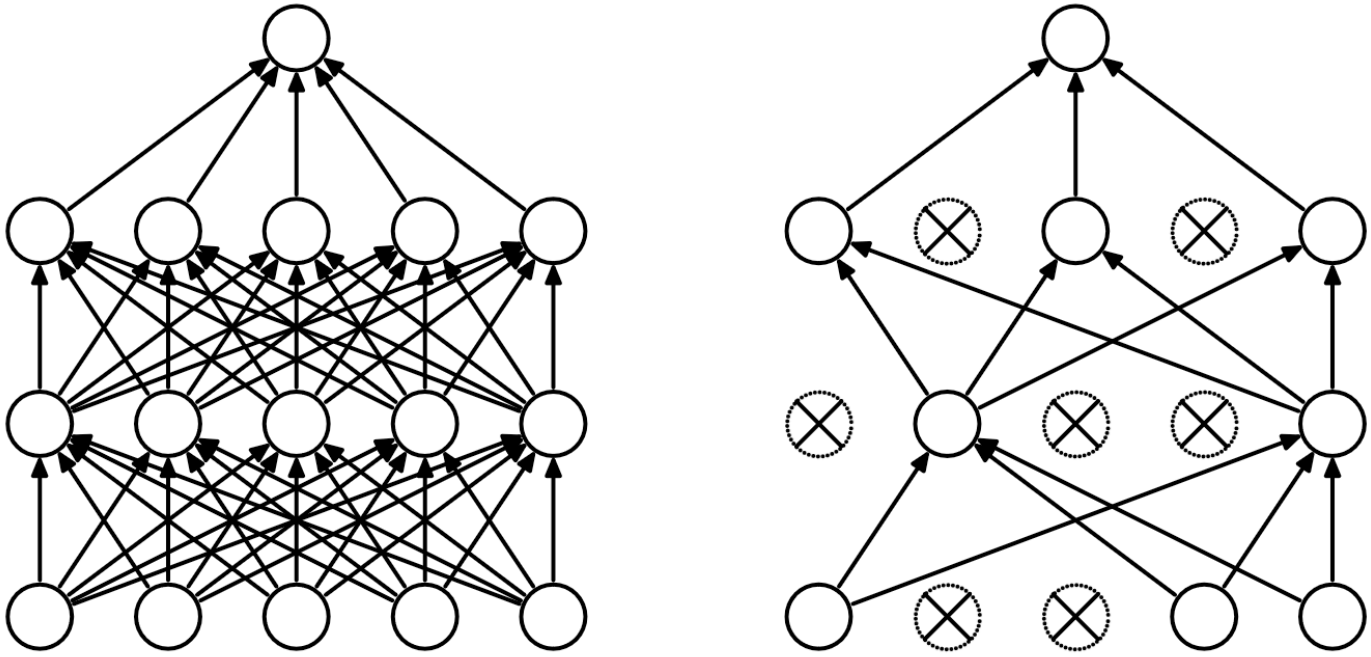
$$\ell^1 = |w_1| + |w_2|$$






L1 Vs. L2 Regularization
L2 loss
L1 loss
= MSE loss (level sets)
= Minimum L2 or L1 loss for constant MSE

Dropout



(a) Standard Neural Net

(b) After applying dropout.

Figure 1: Dropout Neural Net Model. **Left**: A standard neural net with 2 hidden layers. **Right**: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

Dropout

$$\text{Dropout}(\mathbf{X}, r) = \mathbf{D} \odot \mathbf{X}, \quad \mathbf{D} = \begin{bmatrix} d_{11} & d_{12} & \dots & d_{1n} \\ d_{21} & d_{22} & \dots & d_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ d_{m1} & d_{m2} & \dots & d_{mn} \end{bmatrix}, \; d_{ij} \sim \text{Bernoulli}(1 - r)$$

r = dropout rate or future

prob that we drop an input

In inn Layer

$$\phi(\mathbf{x}) = \sigma(\text{DO}_r(\mathbf{x})^T \mathbf{W} + \mathbf{b})$$

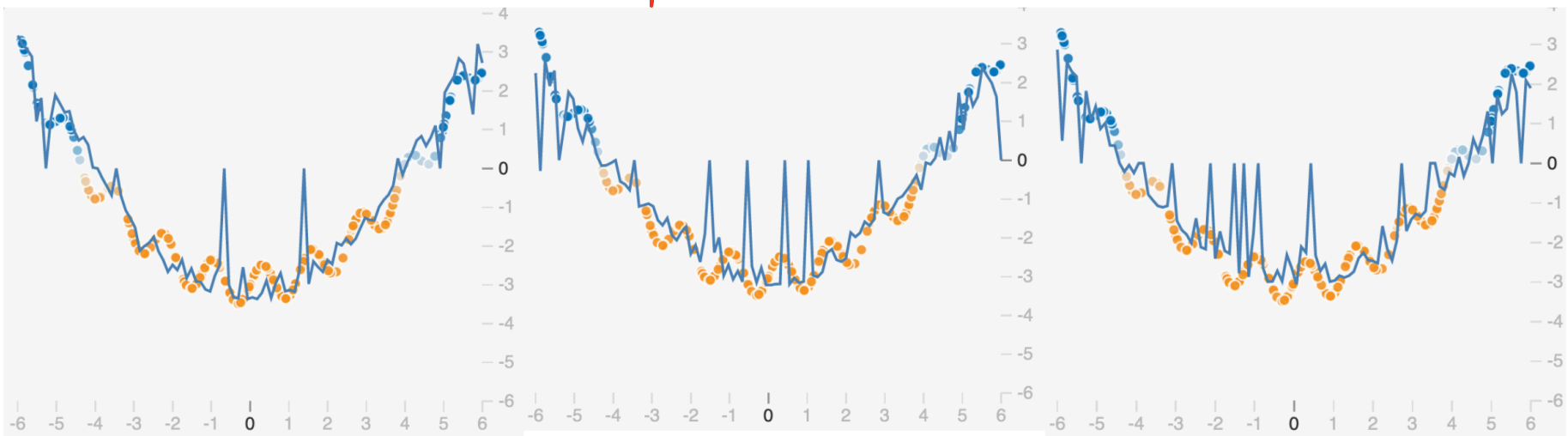
Or,

$$\phi(x) = \sigma(x^T W + b)$$

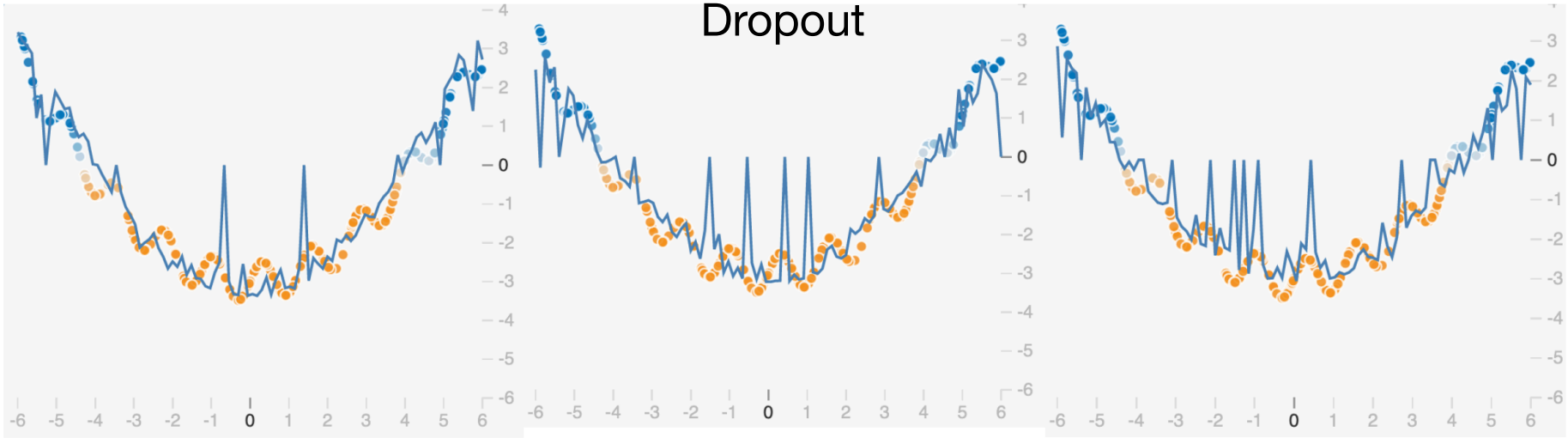
Dropout

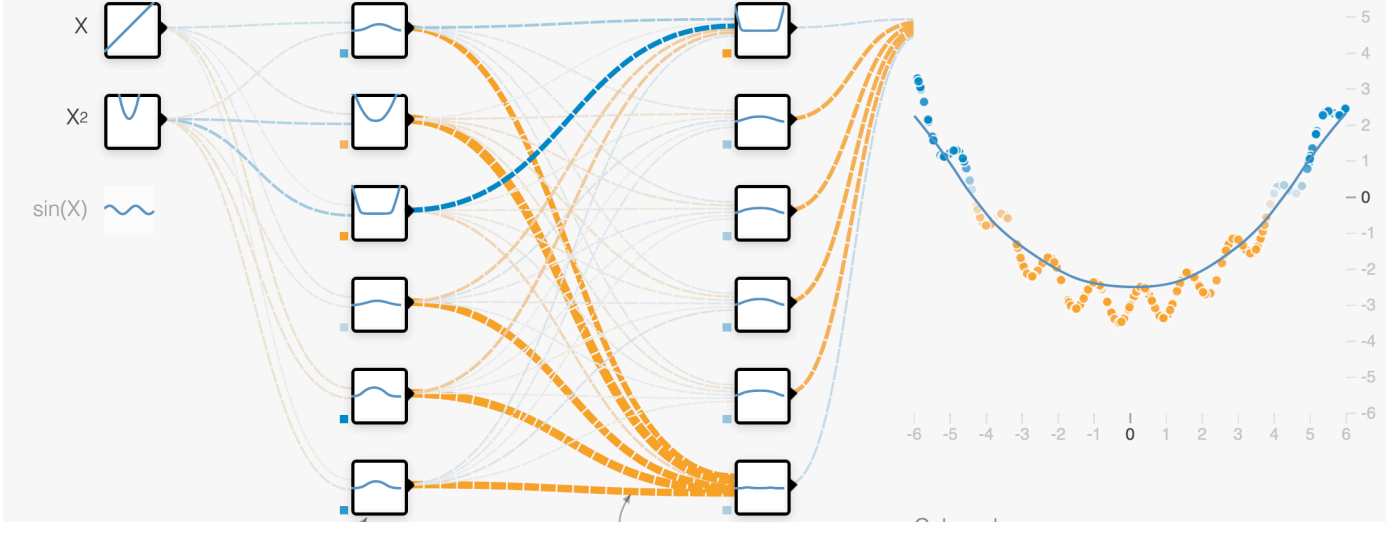
$$\text{Dropout}(\mathbf{X}, r) = \mathbf{D} \odot \mathbf{X}, \quad \mathbf{D} = \begin{bmatrix} d_{11} & d_{12} & \dots & d_{1n} \\ d_{21} & d_{22} & \dots & d_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ d_{m1} & d_{m2} & \dots & d_{mn} \end{bmatrix}, \; d_{ij} \sim \text{Bernoulli}(1 - r)$$

Three Random dropouts



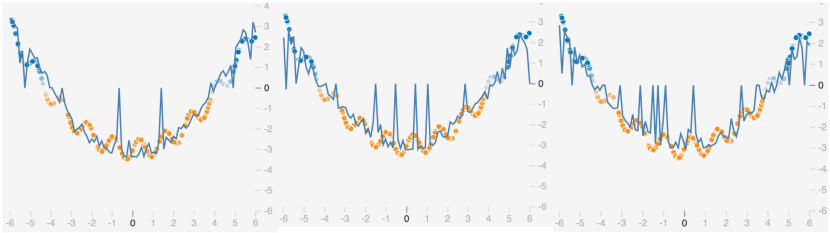
Dropout

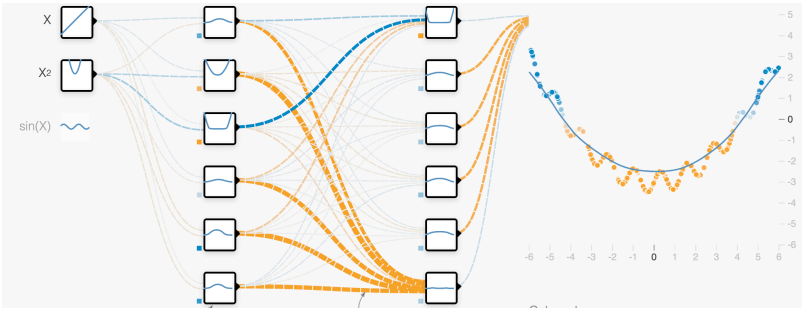




Dropout

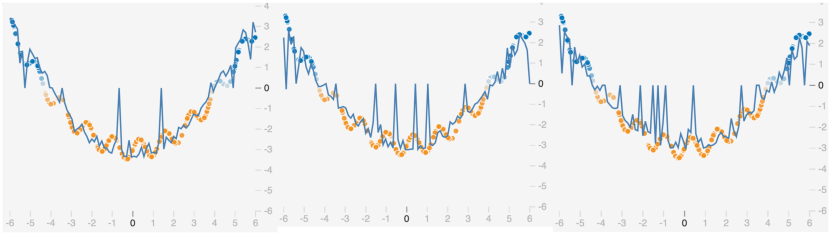
$$\phi(\mathbf{x})_{train} = \sigma(\mathrm{DO}_r(\mathbf{x})^T\mathbf{W} + \mathbf{b}) \quad \rightarrow \quad \phi(\mathbf{x})_{eval} = \sigma(\mathbf{x}^T\mathbf{W} + \mathbf{b})$$

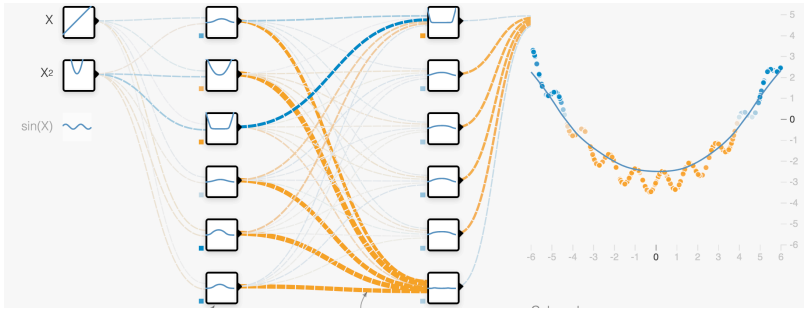




Dropout

$$\phi(\mathbf{x})_{train} = \sigma(\mathrm{DO}_r(\mathbf{x})^T\mathbf{W} + \mathbf{b}) \quad \rightarrow \quad \phi(\mathbf{x})_{eval} = \sigma(\mathbf{x}^T\mathbf{W} + \mathbf{b})$$





$$\mathbb{E}[\mathrm{DO}_r(\mathbf{x})^T\mathbf{w}] = \sum_i d_i x_i w_i, \quad d_i \sim \mathrm{Bernoulli}(1-r)$$

$$= \sum_i p(d_i = 1) x_i w_i = (1-r)\sum_i x_i w_i < \sum_i x_i w_i$$

Dropout

```
# 2 Hidden-layer network with dropout
model = nn.Sequential(nn.Dropout(0.5), nn.Linear(2, 10), nn.ReLU(),
                      nn.Dropout(0.5), nn.Linear(10, 10), nn.ReLU(),
                      nn.Dropout(0.5), nn.Linear(10, 1)
                     )
```