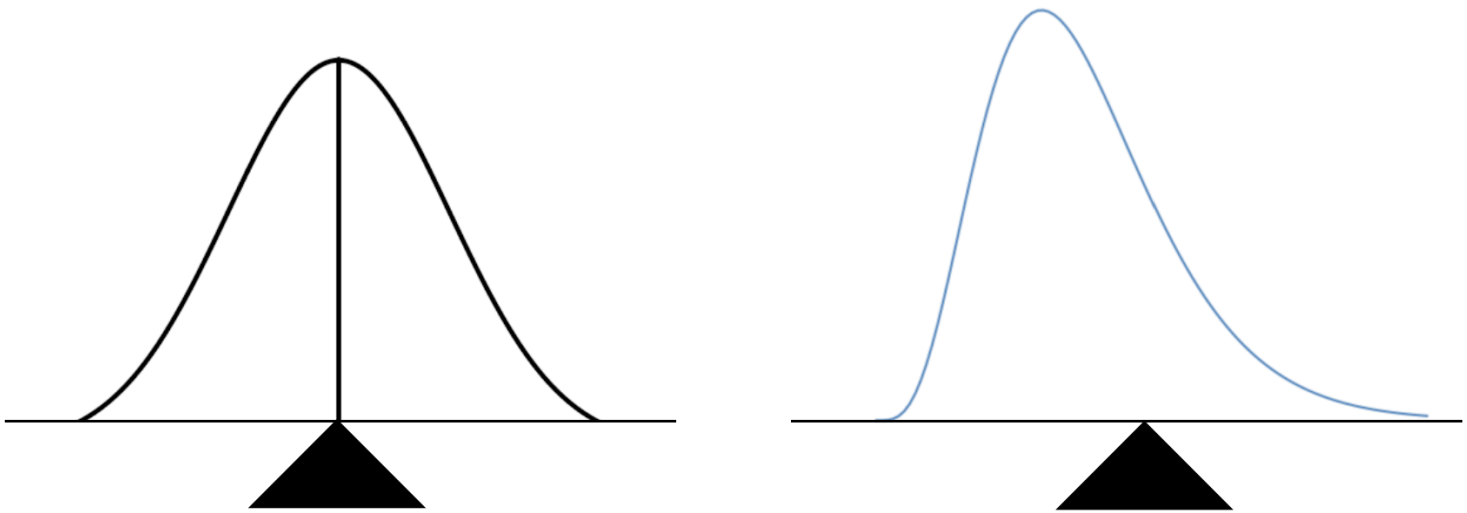
Initialization

Gabriel Hope

Review of expectation and variance

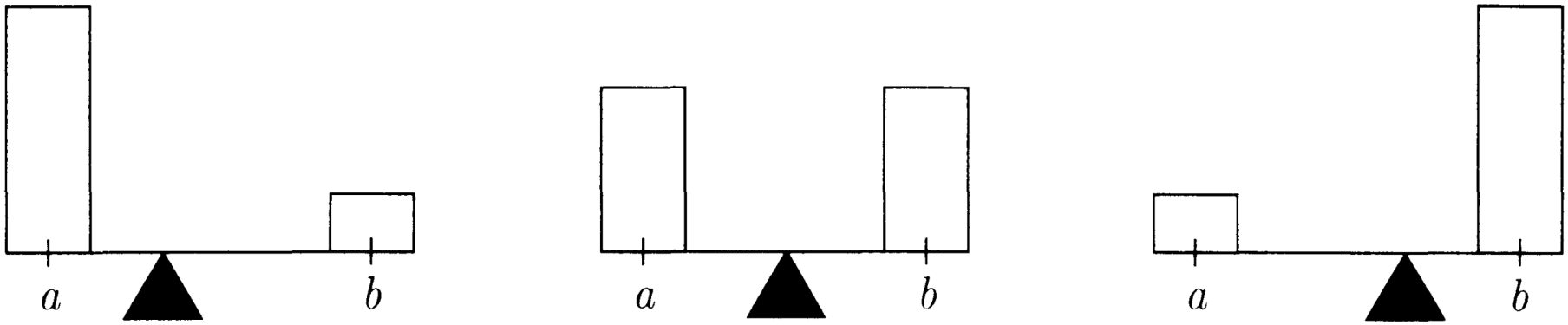
Expectation

$$\mathbb{E}[X] = \int_x xp(x)dx \quad \text{(Continuous random variables)}$$



Expectation

$$\mathbb{E}[X] = \sum_x xp(x) \quad \text{(Discrete random variables)}$$



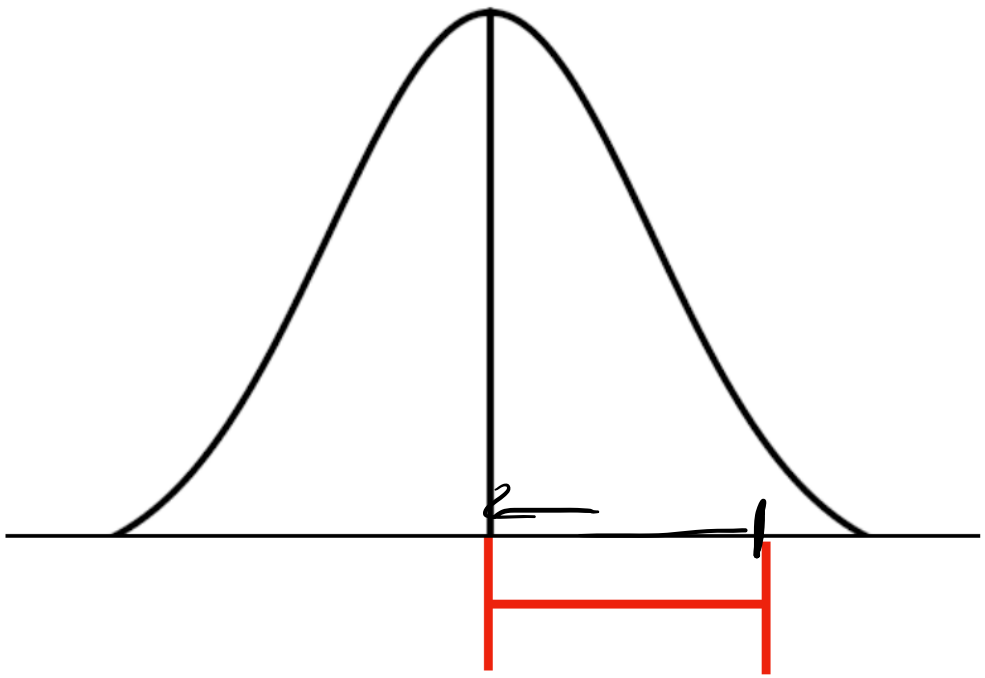
Linearity of expectation

$$\mathbb{E}\left[aX\right] = a\mathbb{E}\left[X\right]$$

$$\mathbb{E}\left[\sum_{i=1}^{n} X_i\right] = \sum_{i=1}^{n} \mathbb{E}\left[X_i\right]$$

Variance

$$Var\left[X\right] = \mathbb{E}\left[(X - \mathbb{E}[X])^2\right]$$



Properties of variance

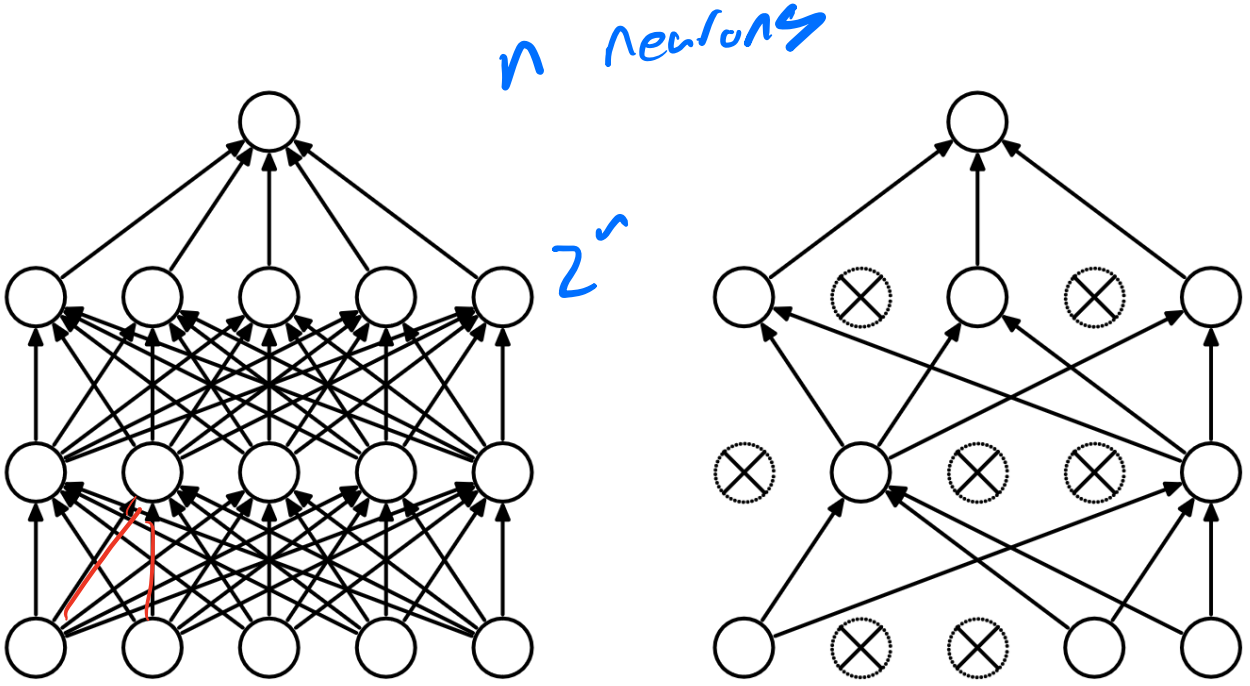
$$Var[X] = \mathbb{E}[X^2] - \mathbb{E}[X]^2$$

$$Var[aX] = a^2 Var[X]$$

If X and Y are independent:

$$Var[X + Y] = Var[X] + Var[Y]$$

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

Dropout rate: r

$$\mathrm{DO}(\mathbf{X}, r) = \mathbf{D} \odot \mathbf{X}, \quad \mathbf{D} = \begin{bmatrix} d_{11} & d_{12} & \dots & d_{1n} \\ d_{21} & d_{22} & \dots & d_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ d_{m1} & d_{m2} & \dots & d_{mn} \end{bmatrix}$$

$$d_{ij} \sim \mathrm{Bernoulli}(1 - r)$$

Dropout

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

Within a NN layer:

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

Dropout

A network with several dropout layers:

$$f(\mathbf{x}, \mathbf{w}_0, \dots) = \mathrm{DO}_r(\sigma(\mathrm{DO}_r(\sigma(\mathrm{DO}_r(\sigma(\mathrm{DO}_r(\mathbf{x})^T\mathbf{W}_2 + \mathbf{b}_2))

Dropout

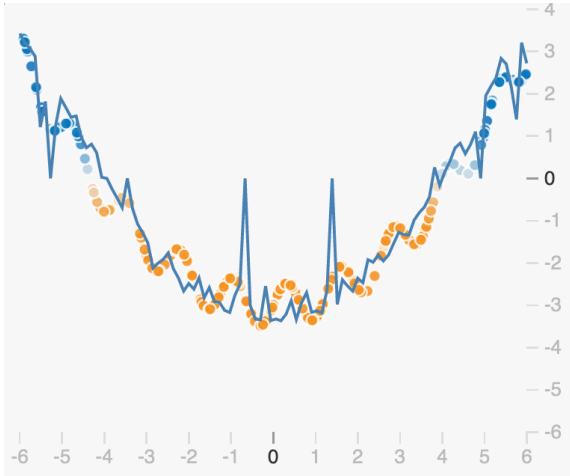
A network with several dropout layers:

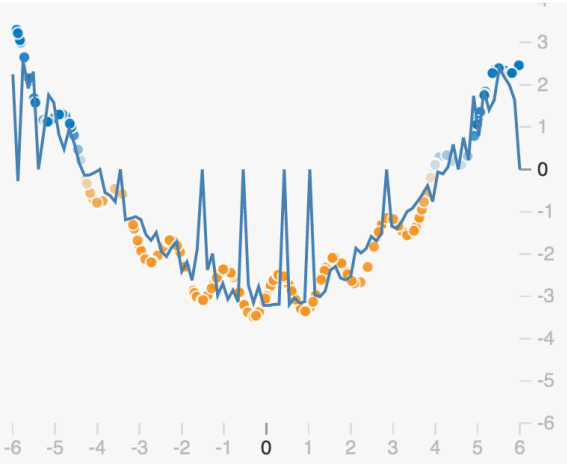
$$\mathbf{a} = \sigma(\mathrm{DO}_r(\mathbf{x})^T\mathbf{W}_2 + \mathbf{b}_2)$$

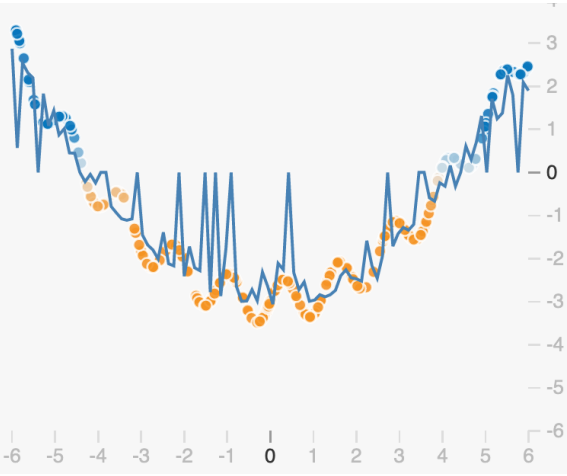
$$\mathbf{b} = \sigma(\mathrm{DO}_r(\mathbf{a})^T\mathbf{W}_1 + \mathbf{b}_1)$$

$$\mathbf{f} = \mathrm{DO}_r(\mathbf{b})^T\mathbf{w}_0 + b_0$$

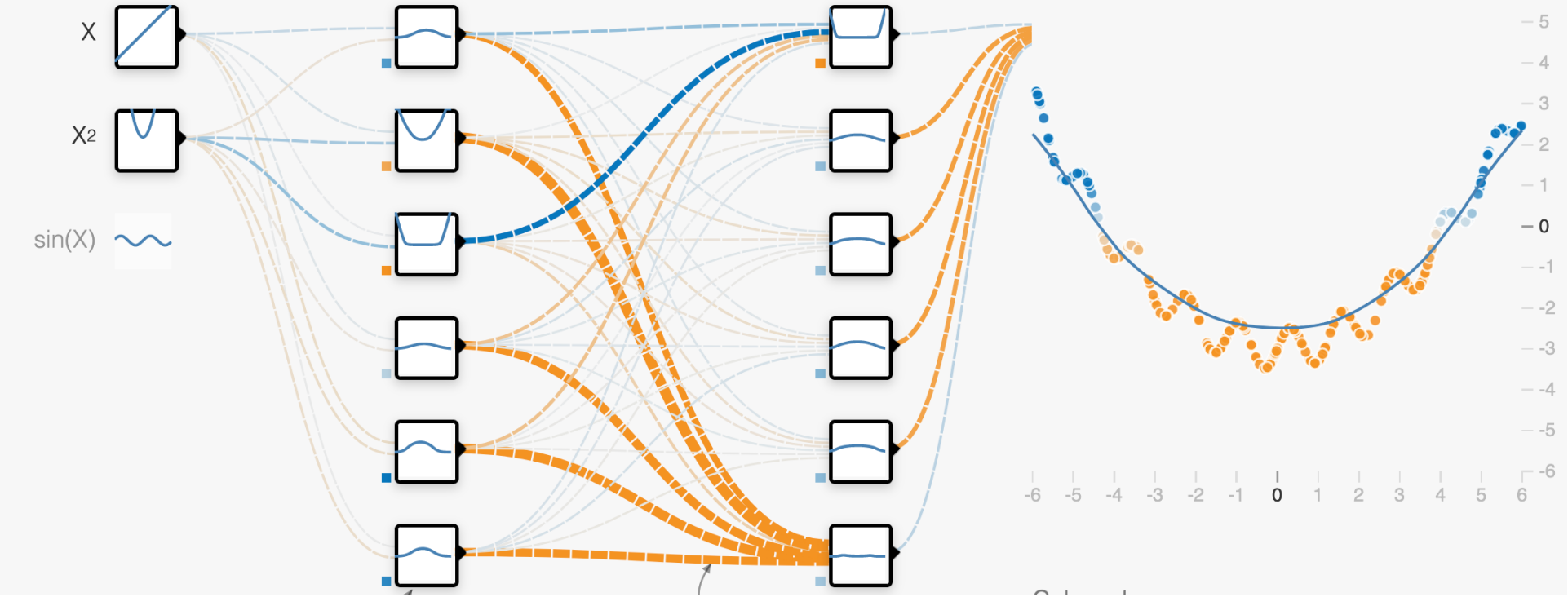
Dropout


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


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


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

Dropout



Dropout at evaluation time

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

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

This has a problem!

Dropout at evaluation time

$r = 0.5$

Consider the **expected value** of a linear function with dropout:

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

$d \begin{cases} 0 & \text{w/ prob. } r \\ 1 & \text{w/ prob } 1-r \end{cases}$

$E[d_i] = 1 - r$

$$E\left[\sum_i d_i x_i w_i\right] = \sum_i E[d_i x_i w_i] = \sum_i x_i w_i E[d_i] = (1-r)\sum_i x_i w_i$$

$$= (1-r) x^T w$$

Dropout at evaluation time

Consider the **expected value** of a linear function with dropout:

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

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

If $r = 0.5$ then on average the output of our function with dropout will only be half as large as the function without dropout!

Dropout at evaluation time

Correct solution

$f(x)$ take M samples of dropout network

$$E[f(x)] \approx \frac{1}{M} \sum_{i=}^{M} f_i(x)$$

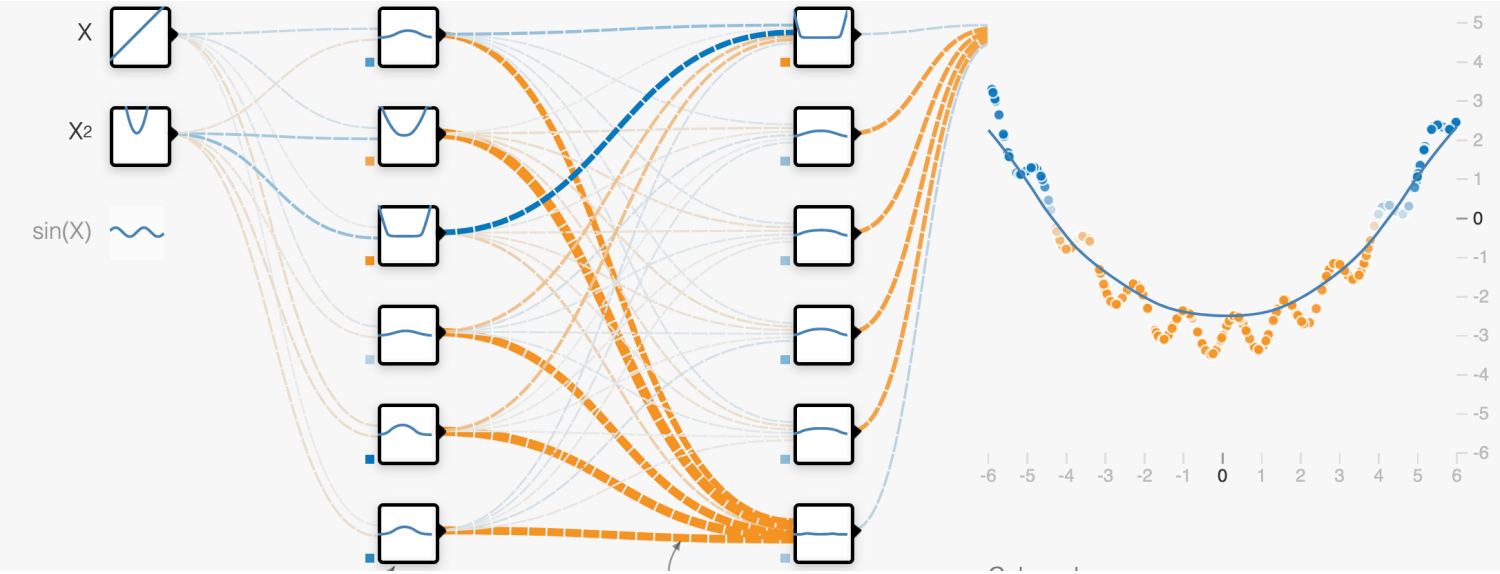
f_i = one randomly sampled network

Dropout at evaluation time

Simple (but not quite correct) solution

$$\text{Dropout}_{eval}(\mathbf{X}, r) = (1 - r)\mathbf{X}$$

This gives use the smooth prediction function we're looking for:



Dropout in PyTorch

```
1  # 2 Hidden-layer network with dropout
2  model = nn.Sequential(nn.Dropout(0.5), nn.Linear(2, 10), nn.ReLU(),
3                        nn.Dropout(0.5), nn.Linear(10, 10), nn.ReLU(),
4                        nn.Dropout(0.5), nn.Linear(10, 1)
5                       )
```

At train time

model.train()

At evaluation time

model.eval() → Set to use our eval version of dropout

Optimization

Initialization

Gradient descent:

$$\mathbf{w}^{(k+1)} \longleftarrow \mathbf{w}^{(k)} - \alpha \nabla_{\mathbf{w}} \mathbf{Loss}(\mathbf{w}^{(k)}, \mathbf{X}, \mathbf{y})$$

What do we choose for $\mathbf{w}^{(0)}$?

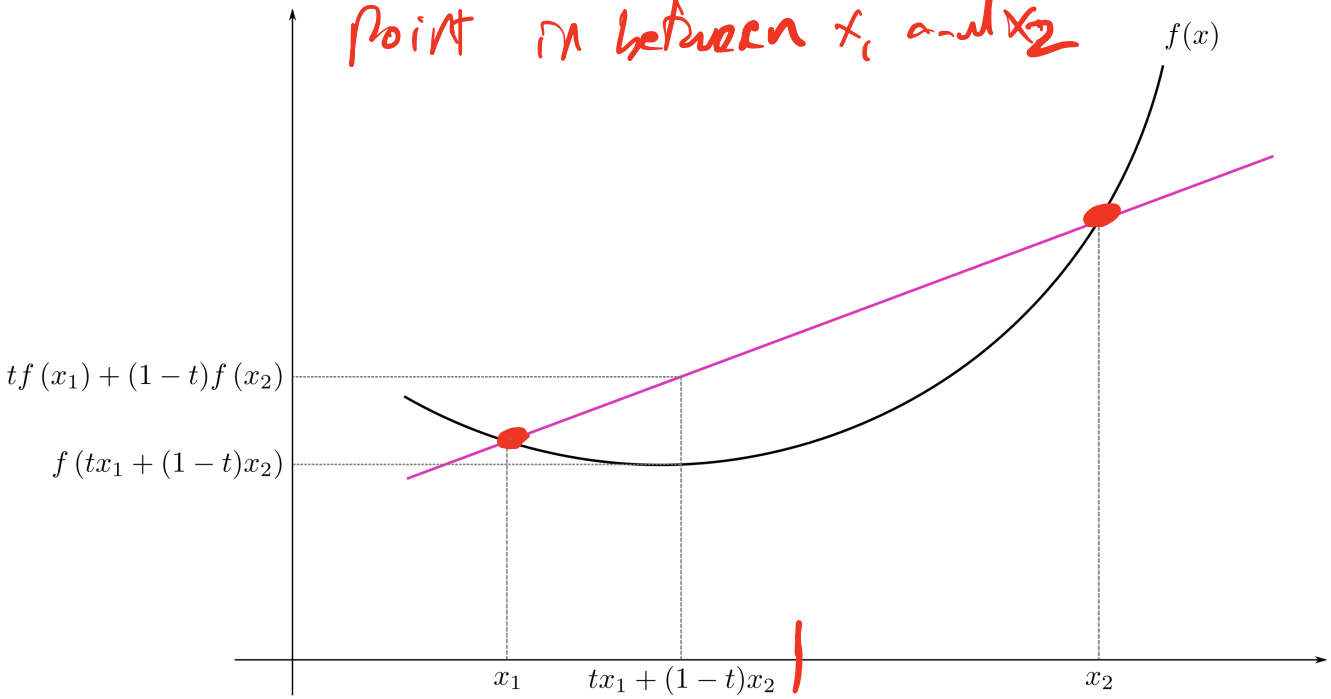
Convexity

Convex function f

$t \in [0, 1]$

$$f(tx_1 + (1-t)x_2) \leq tf(x_1) + (1-t)f(x_2)$$

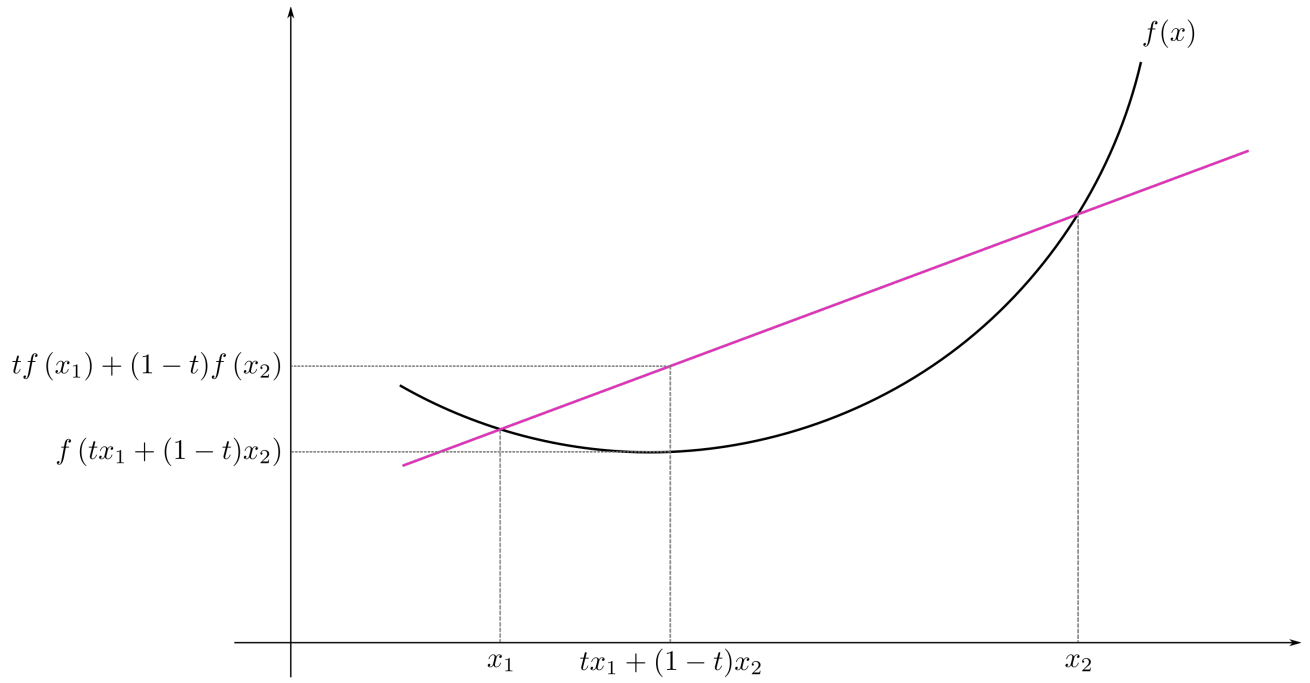
point in between x_1 and x_2



Convexity

Strictly convex function f

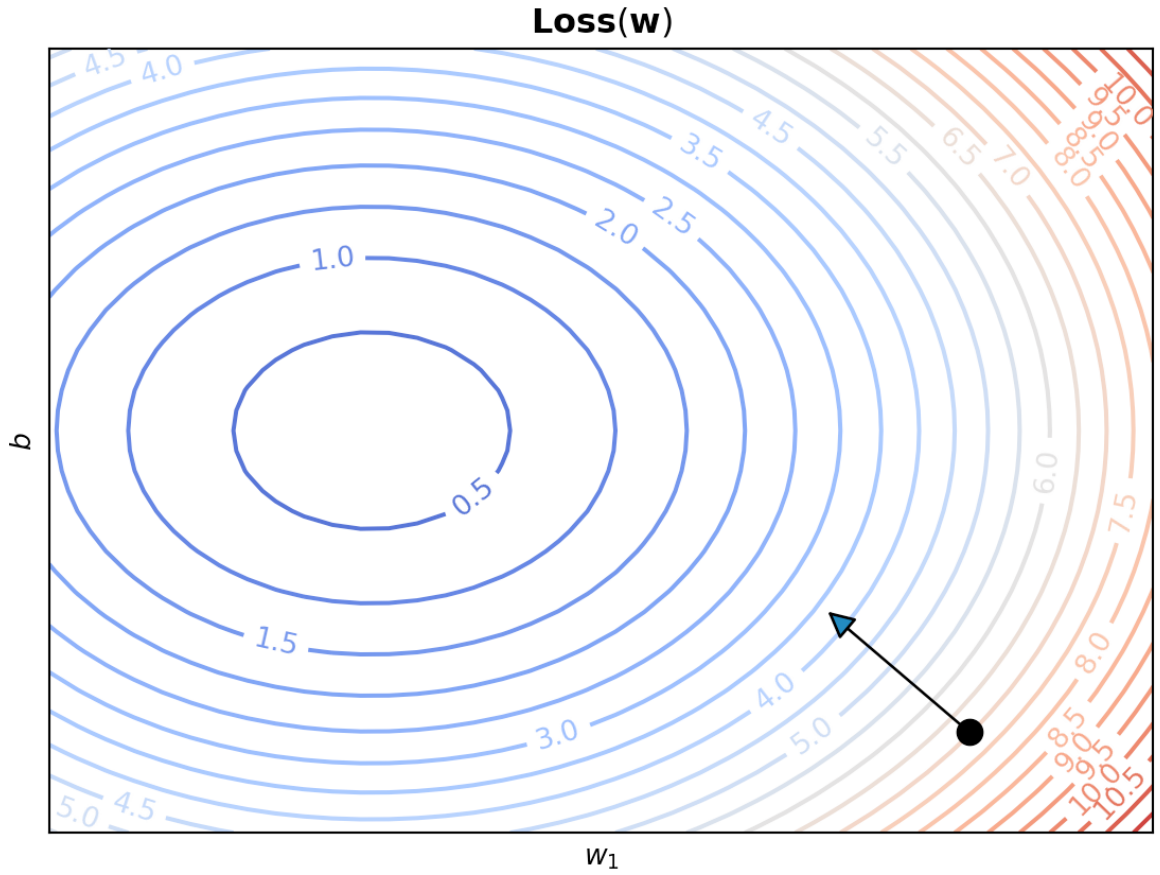
$$f(tx_1 + (1-t)x_2) < tf(x_1) + (1-t)f(x_2)$$



Single optimum, gradient is non-zero away from the optimum.

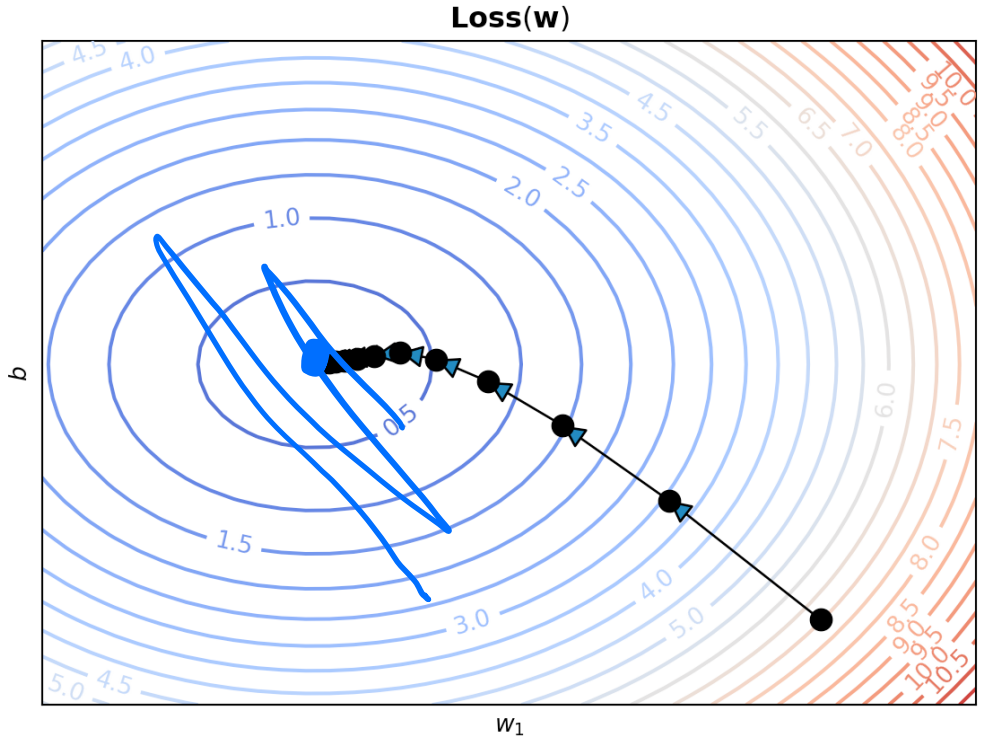
Convexity

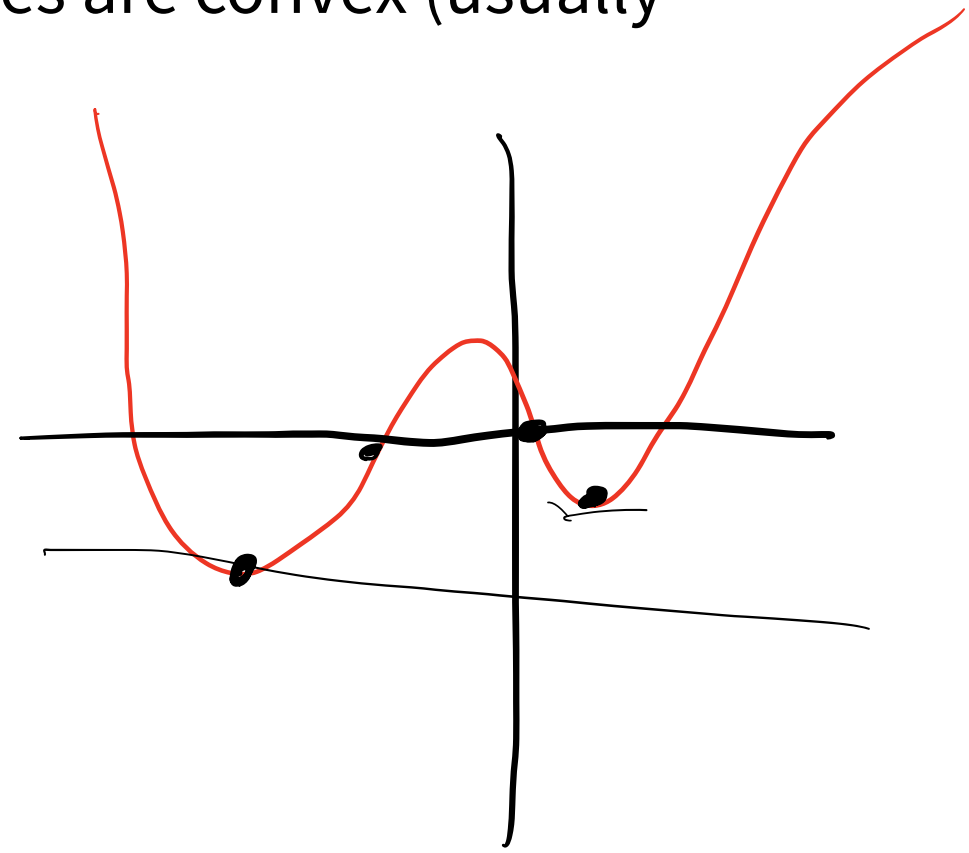
Linear and logistic regression losses are convex (usually strictly)



Convexity

Linear and logistic regression losses are convex (usually strictly)

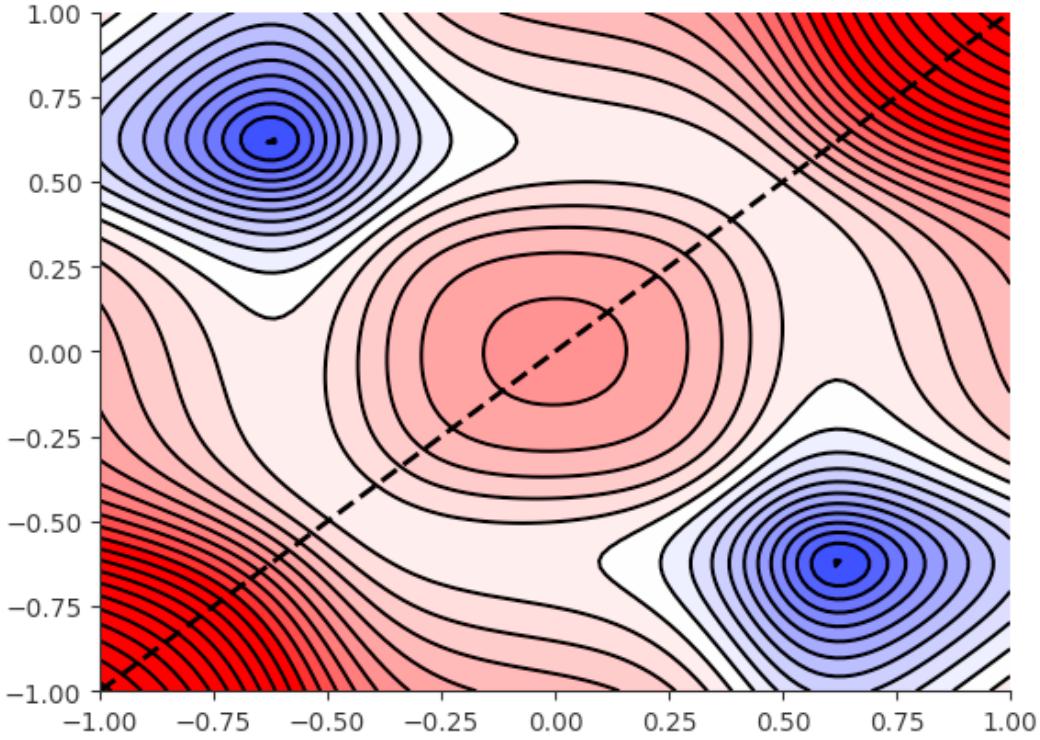




Gradient descent will always* get us to the right answer.

Convexity

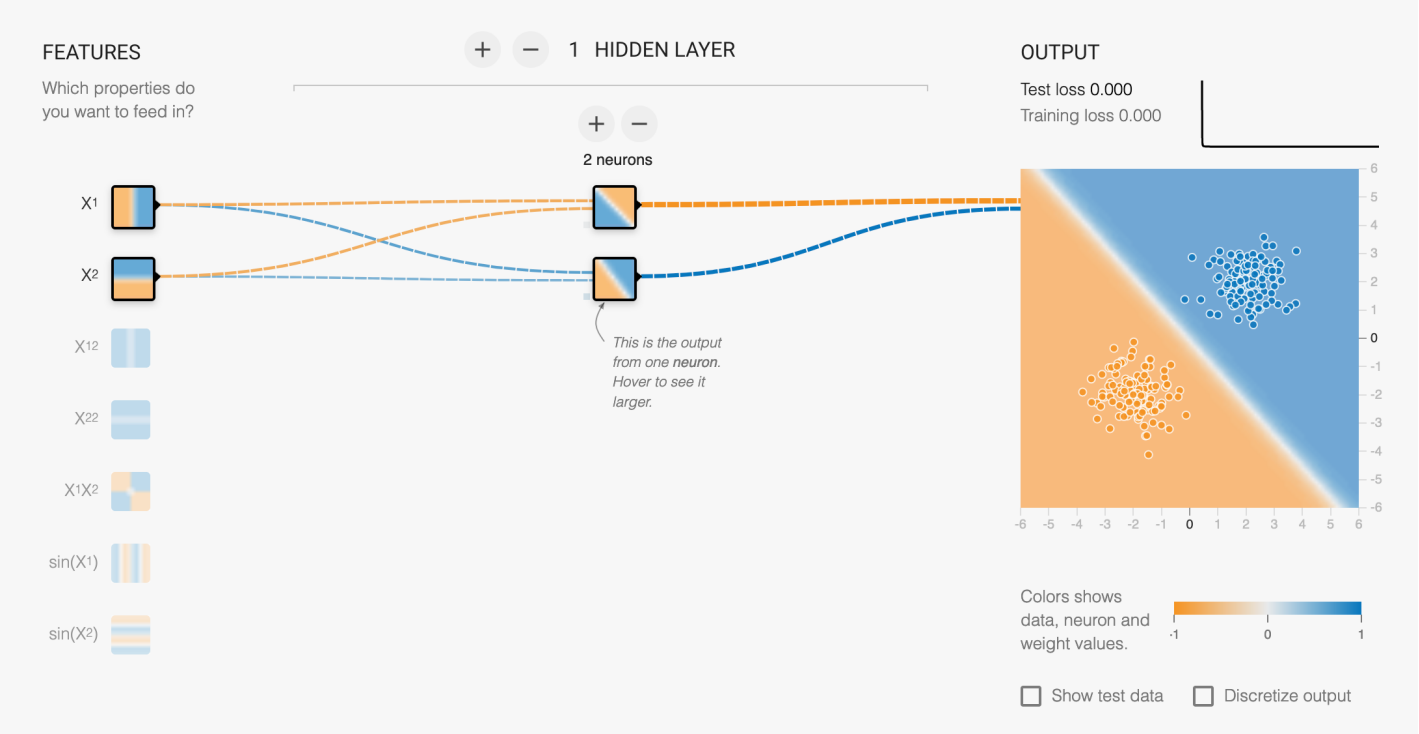
Neural network losses are (usually) not convex!

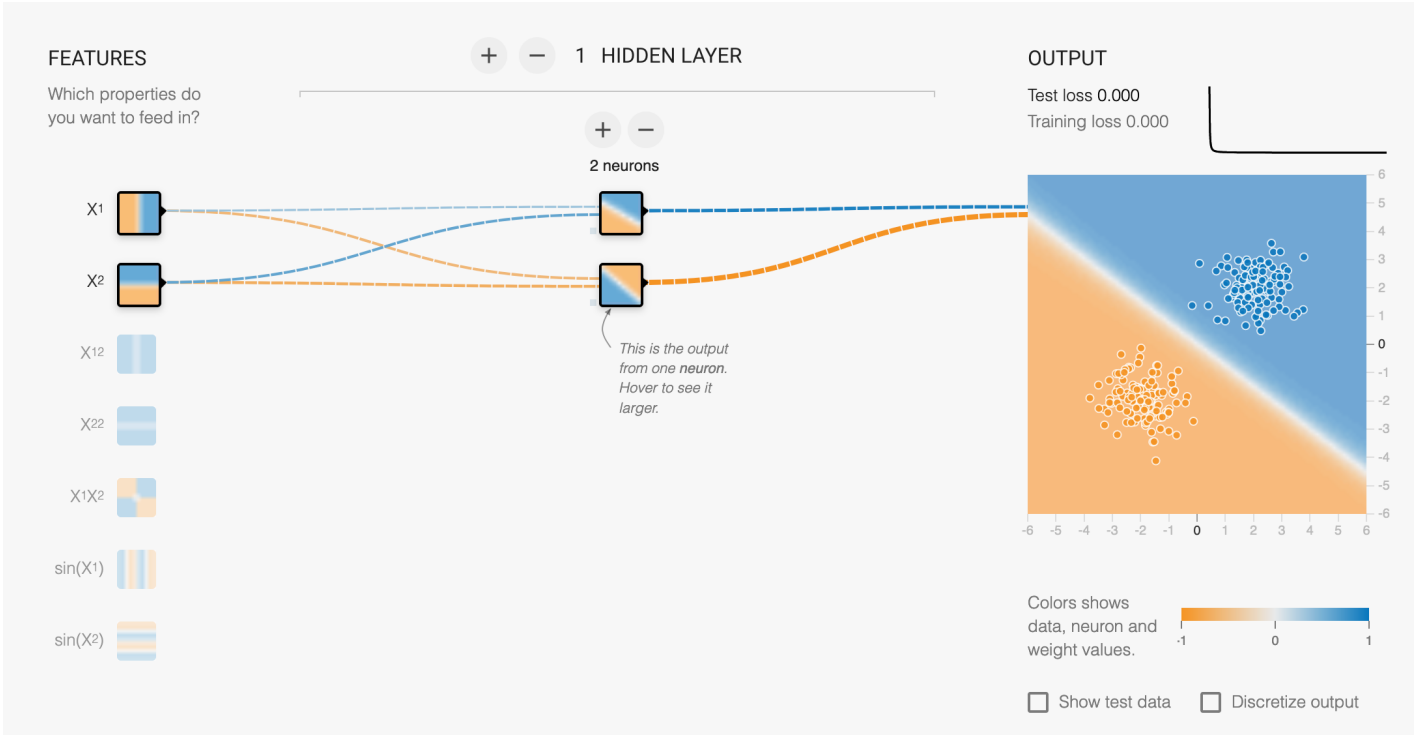


Why?

Convexity

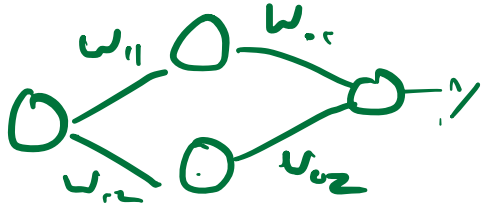
Two equivalent solutions




FEATURES
Which properties do you want to feed in?
1 HIDDEN LAYER
2 neurons
X1
X2
X12
X22
X1X2
sin(X1)
sin(X2)
This is the output from one neuron. Hover to see it larger.
OUTPUT
Test loss 0.000
Training loss 0.000
Colors shows data, neuron and weight values.
-1
0
1
Show test data
Discretize output

Symmetry-breaking

A simple network



$$f(\mathbf{x}) = \sigma(\mathbf{x}^T\mathbf{W}_1)^T\mathbf{w}_0 = \sigma(x_1 w_{11})w_{01} + \sigma(x_1 w_{12})w_{02}$$

$$\frac{d}{dw_{01}} f(\mathbf{x}) =$$

$$\frac{d}{dw_{02}} f(\mathbf{x}) =$$

Symmetry-breaking

A simple network

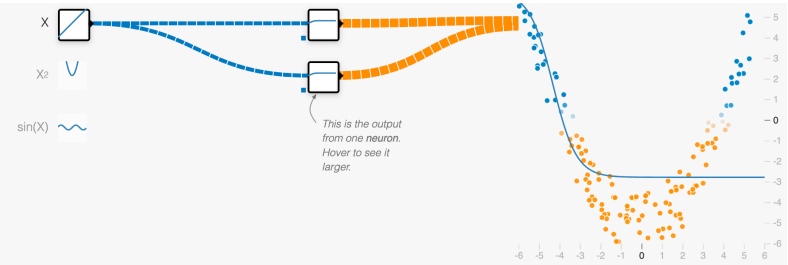
$$f(\mathbf{x}) = \sigma(\mathbf{x}^T \mathbf{W}_1)^T \mathbf{w}_0 = \sigma(x_1 w_{11}) w_{01} + \sigma(x_1 w_{12}) w_{02}$$

$$\frac{d}{dw_{01}} f(\mathbf{x}) = \sigma(x_1 w_{11}) = \sigma(x_1 a)$$

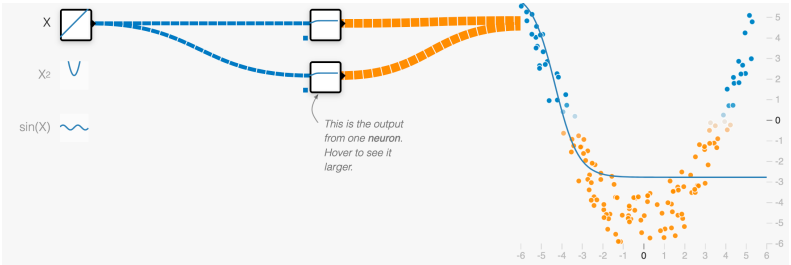
$$\frac{d}{dw_{02}} f(\mathbf{x}) = \sigma(x_1 w_{12}) = \sigma(x_1 a)$$

Set w_{11} and w_{12} to a?

Symmetry-breaking

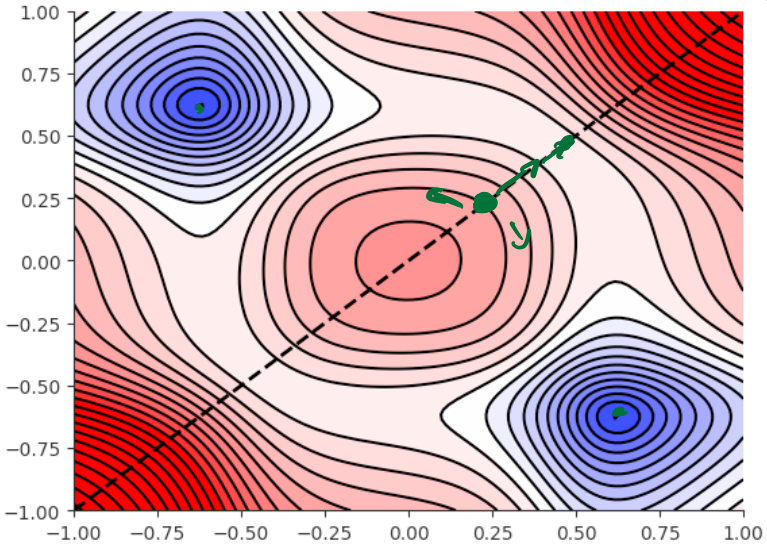


When the network is initialized with symmetry, the two neurons will always have the same output and our solution is poor.

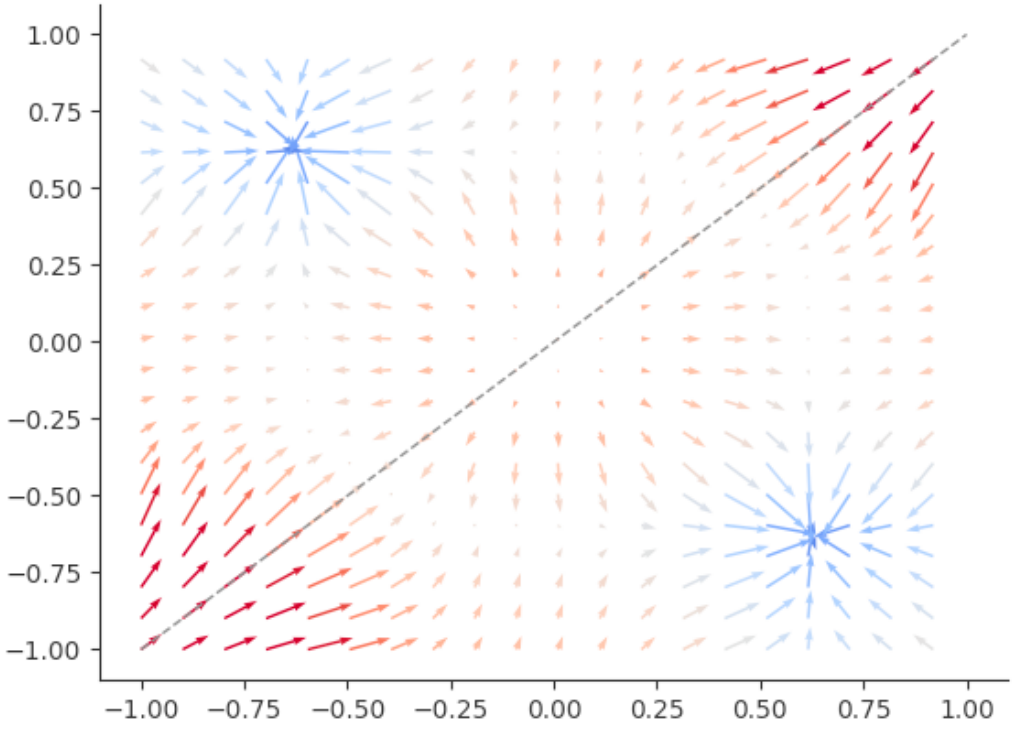


When initialized randomly, the two neurons can create different transforms and a much better solution is found.

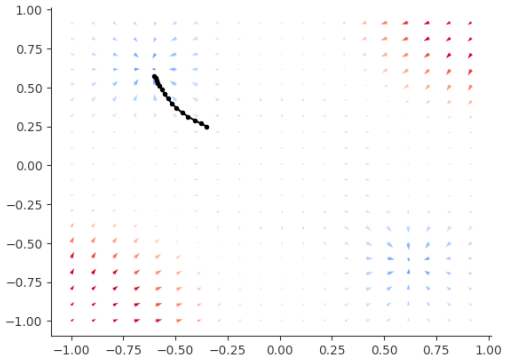
Symmetry-breaking



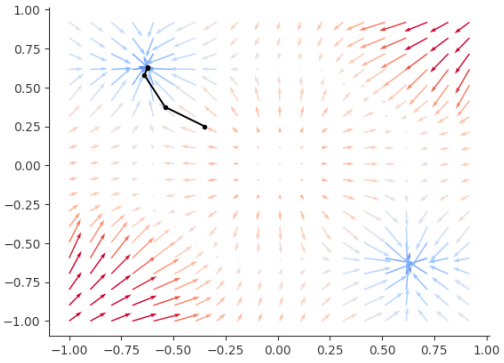
$w_1 = w_2$


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

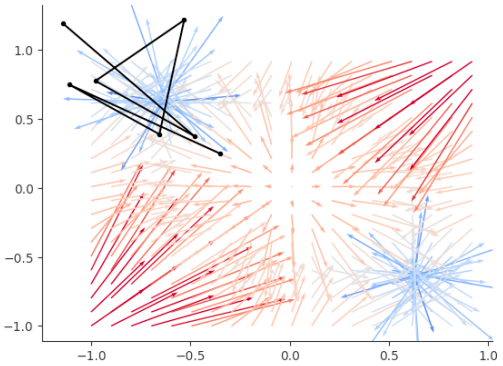
Visualizing learning rates



A small learning rate means we will move slowly, so It may take a long time to find the minimum.



A well-chosen learning rate lets us find a minimum quickly.



A too-large learning rate means that steps may take us flying past the minimum!

Scaled initialization

Initializing randomly:

$$w_i \sim \mathcal{N}(0, 1) \quad \forall\, w_i \in \mathbf{w}$$

This has a subtle issue though. Why?

Scaled initialization

To see why let's consider a linear function defined by randomly initialized weights:

$$f(\mathbf{x}) = \sum_{i=1}^{d} x_i w_i$$

Let's consider the mean and variance of this output with respect to $\mathbf{w}$:

$$\mathbb{E}\Big[f(\mathbf{x})\Big] =$$

$$\mathbf{Var}\Big[f(\mathbf{x})\Big] =$$

Scaled initialization

To see why let's consider a linear function defined by randomly initialized weights:

$$f(\mathbf{x}) = \sum_{i=1}^{d} x_i w_i$$

Let's consider the mean and variance of this output with respect to $\mathbf{w}$:

$$\mathbb{E}\big[f(\mathbf{x})\big] = \mathbb{E}\left[\sum_{i=1}^{d} x_i w_i\right]$$

$$= \sum_{i=1}^{d} x_i \mathbb{E}\big[w_i\big] = 0, \quad w_i \sim \mathcal{N}(0,1)$$

Scaled initialization

To see why let's consider a linear function defined by randomly initialized weights:

$$f(\mathbf{x}) = \sum_{i=1}^{d} x_i w_i$$

Let's consider the mean and variance of this output with respect to $\mathbf{w}$:

$$\mathbf{Var}\Big[f(\mathbf{x})\Big] = \mathbf{Var}\Bigg[\sum_{i=1}^{d} x_i w_i\Bigg]$$

$$= \sum_{i=1}^{d} \mathrm{Var}\left[x_i w_i\right] = \sum_{i=1}^{d} x_i^2 \mathrm{Var}[w_i] = \sum_{i=1}^{d} x_i^2$$

Scaled initialization

Scaled initialization

Scaling

$$f(\mathbf{x}) = \sum_{i=1}^{d} x_i w_i \left(\frac{1}{s} \right)$$

If we want the variance to be independent of d, then we want:

$$s =$$

Scaled initialization

Scaling

$$f(\mathbf{x}) = \sum_{i=1}^{d} x_i w_i \left(\frac{1}{s} \right)$$

If we want the variance to be independent of d, then we want:

$$s = \sqrt{d}$$

Scaled initialization

Computing the variance

$$\mathrm{Var}\left[\sum_{i=1}^{d} x_i w_i \left(\frac{1}{\sqrt{d}}\right)\right] =$$

$$\sum_{i=1}^{d} \mathrm{Var}\left[x_i w_i \left(\frac{1}{\sqrt{d}}\right)\right]$$

$$= \sum_{i=1}^{d} x_i^2 \left(\frac{1}{\sqrt{d}}\right)^2 \mathrm{Var}[w_i] = \frac{1}{d} \sum_{i=1}^{d} x_i^2$$

Scaled initialization

For neural network layers where the weights are a matrix $\mathbf{W} \in \mathbb{R}^{d \times e}$, this works the same way:

$$w_{ij} \sim \mathcal{N}\left(0, \frac{1}{\sqrt{d}}\right) \quad \forall\, w_{ij} \in \mathbf{W},\ \mathbf{w} \in \mathbb{R}^{d \times e}$$

A popular alternative scales the distribution according to both the number of inputs and outputs of the layer:

$$w_{ij} \sim \mathcal{N}\left(0, \sqrt{\frac{2}{d+e}}\right) \quad \forall\, w_{ij} \in \mathbf{W},\ \mathbf{w} \in \mathbb{R}^{d \times e}$$

This is known as **Xavier initialization** (or **Glorot initialization** after the inventor Xavier Glorot).

Scaled initialization