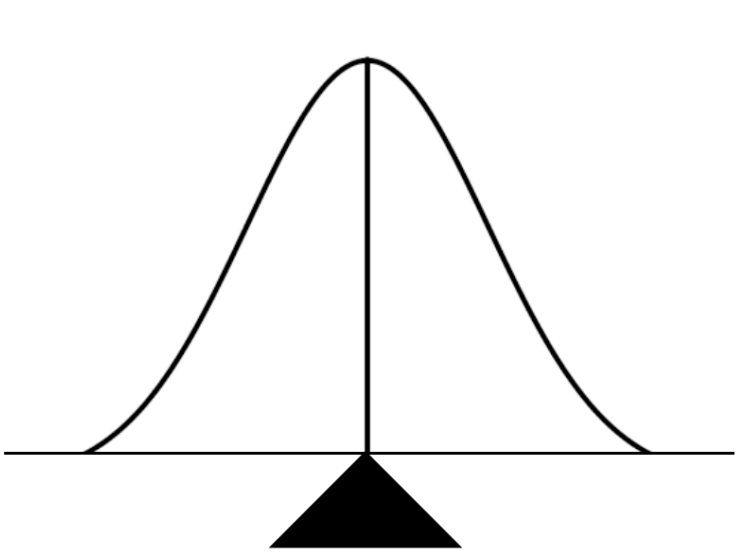
Initialization

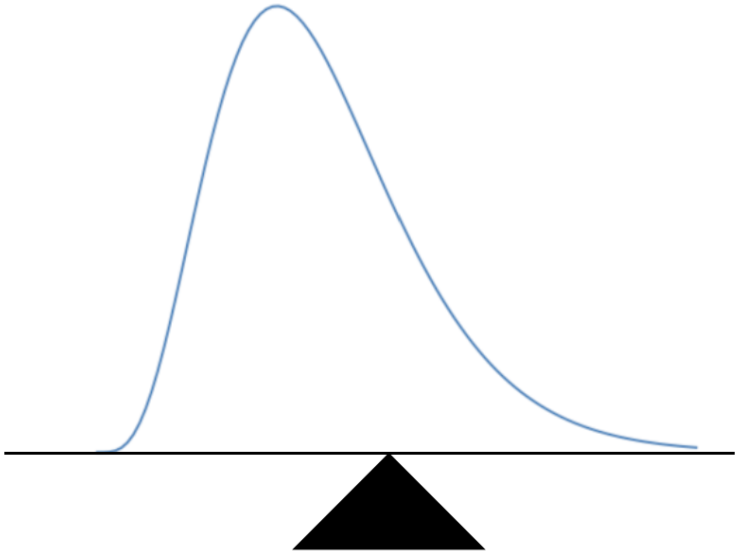
Gabriel Hope

Review of expectation and variance

Expectation

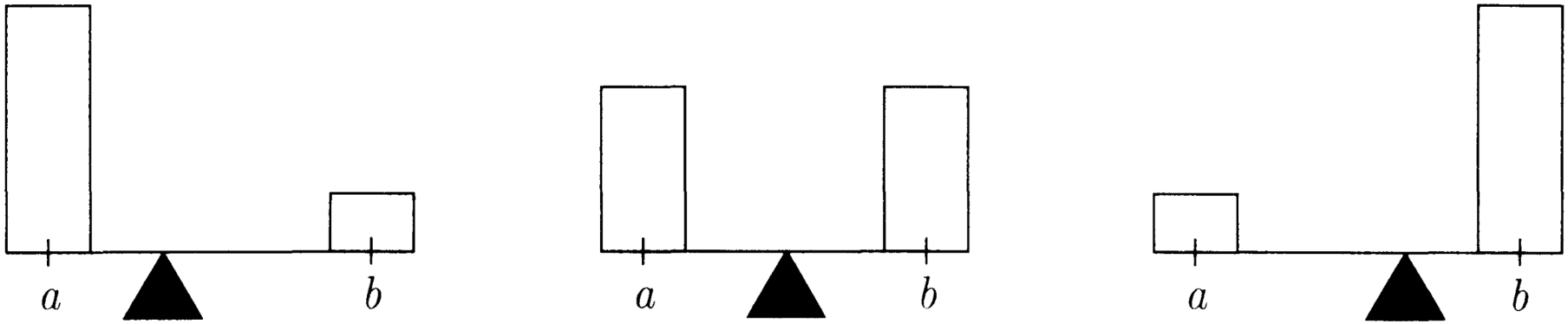
$$\mathbb{E}[X] = \int_x xp(x)dx \quad \text{(Continuous random variables)}$$





Expectation

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



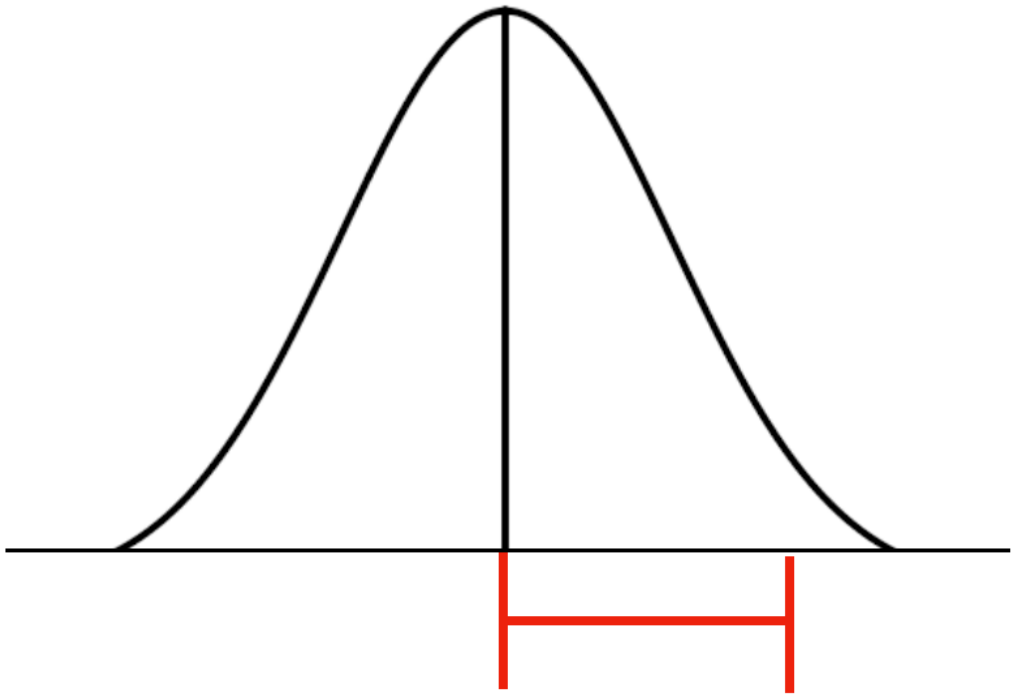
Linearity of expectation

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

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

Variance

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



Properties of variance

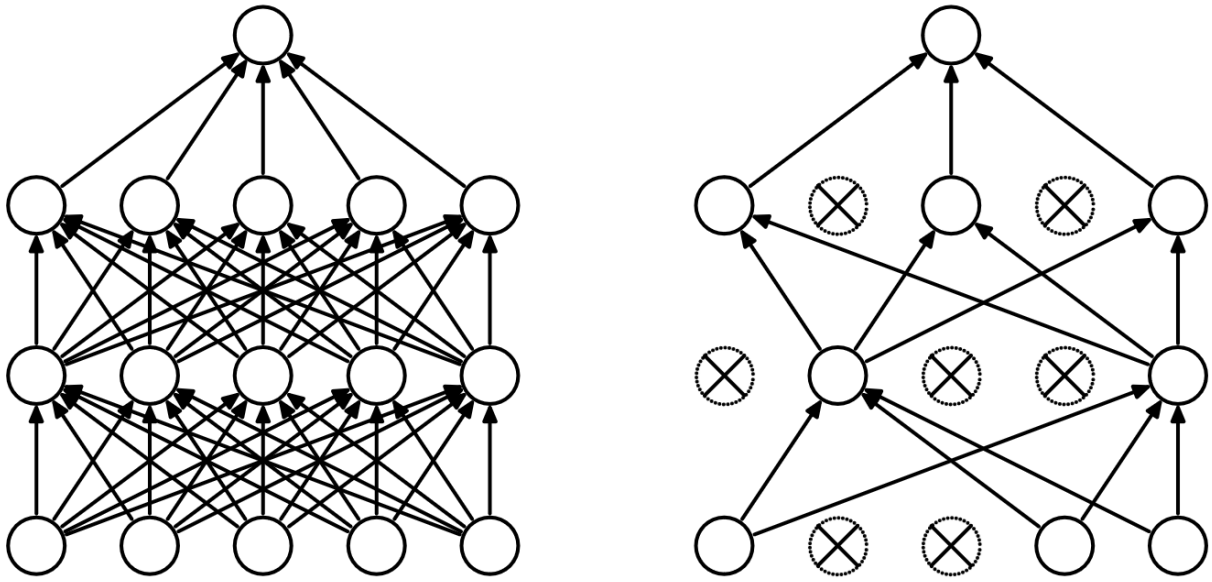
$$Var[X] = \mathbb{E}[X^2] - \mathbb{E}[X]^2$$

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

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

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:

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

Dropout

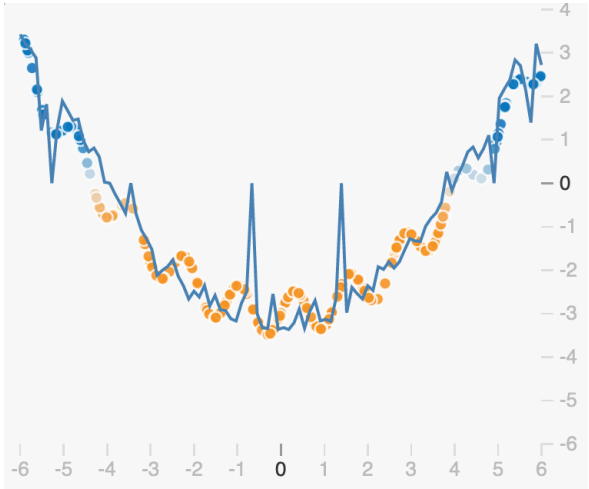
A network with several dropout layers:

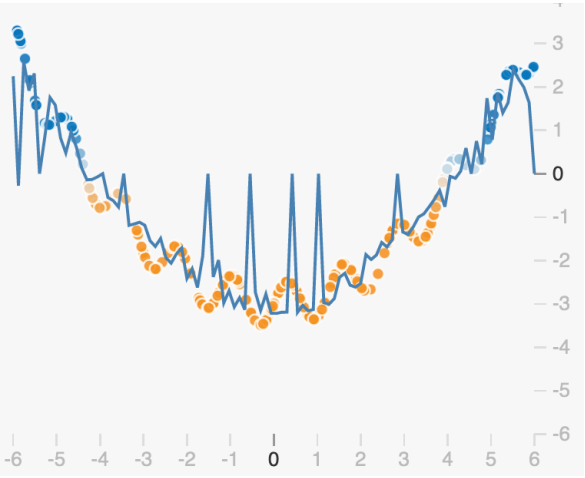
$$\mathbf{a} = \sigma(\mathrm{DO}_r(\mathbf{x})^T\mathbf{W}_2 + \mathbf{b}_2)$$

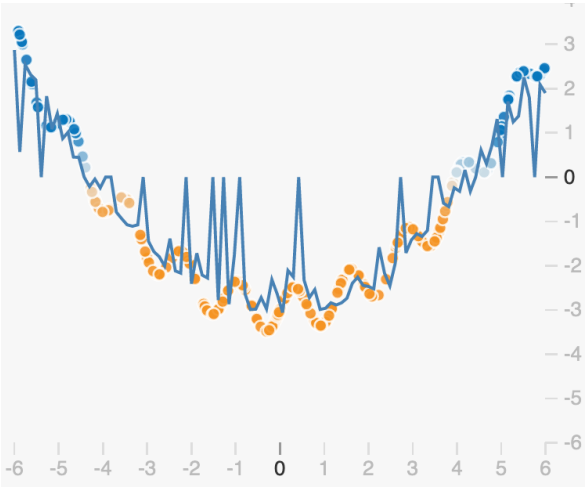
$$\mathbf{b} = \sigma(\mathrm{DO}_r(\mathbf{a})^T\mathbf{W}_1 + \mathbf{b}_1)$$

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

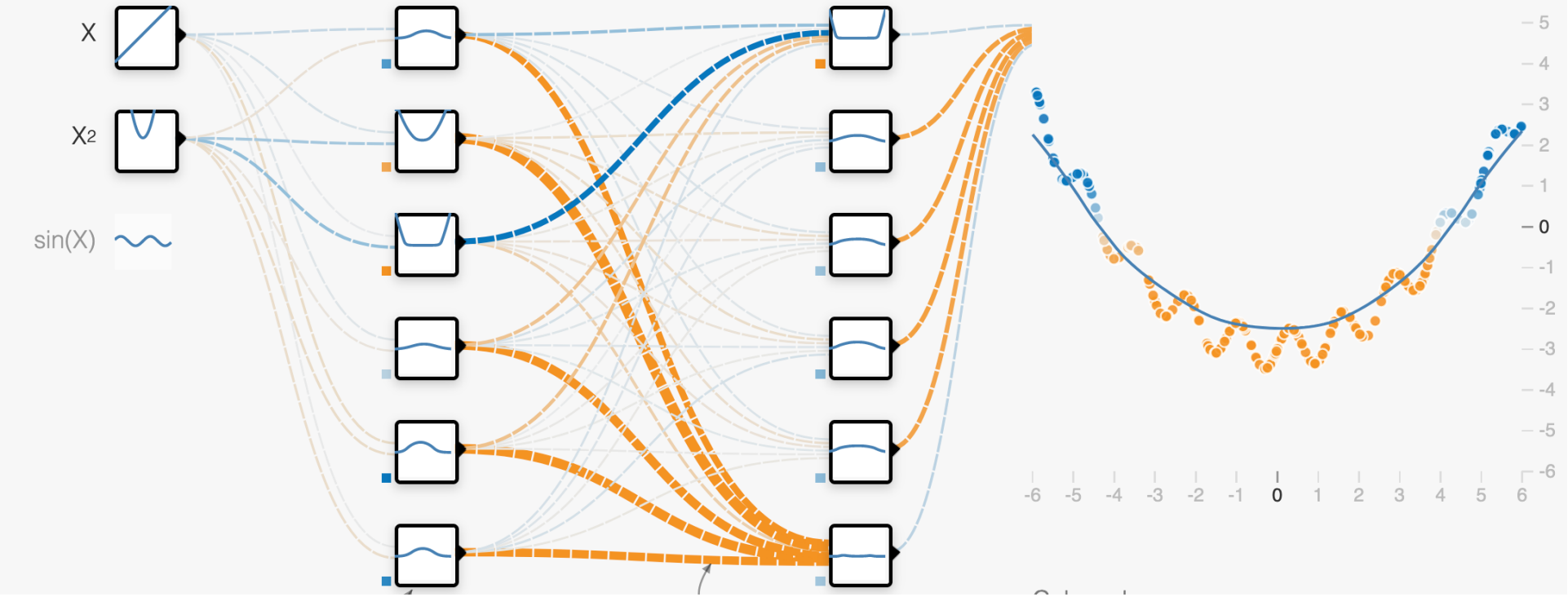
Dropout


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

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

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

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

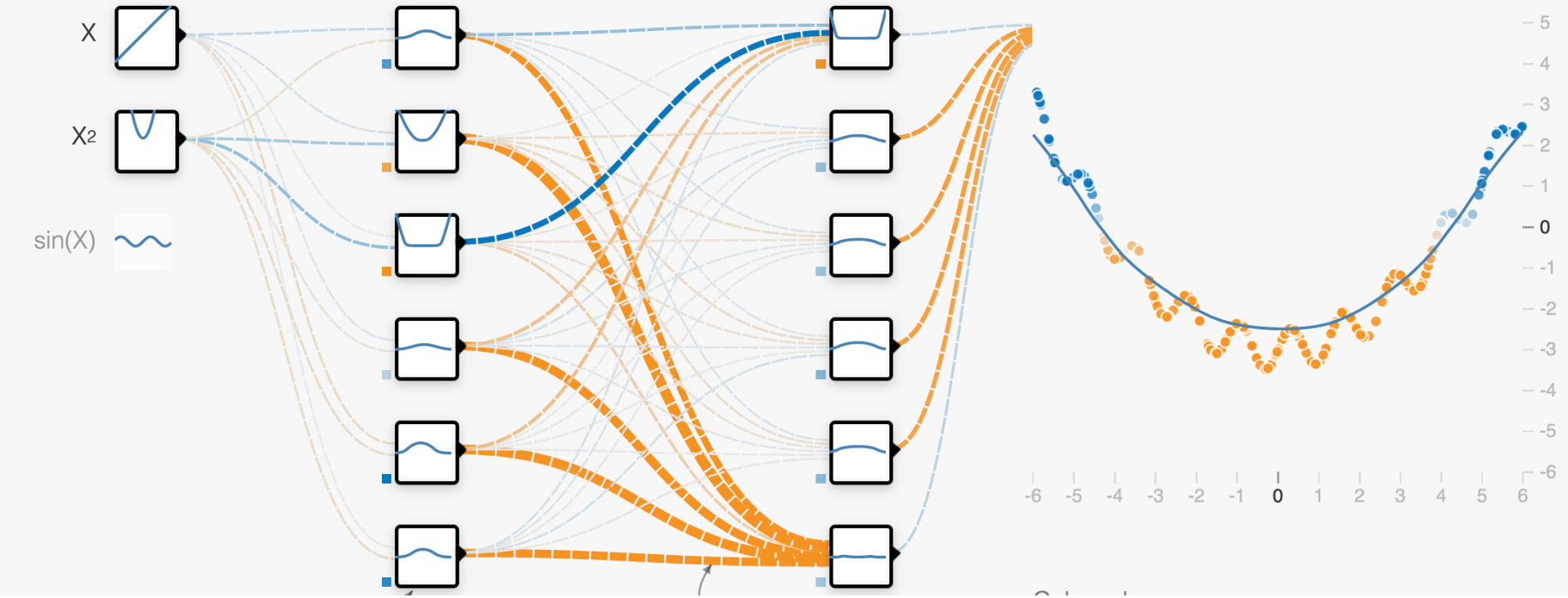
Correct solution

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

```
# 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)
                     )
```

Optimization

Initialization

Gradient descent:

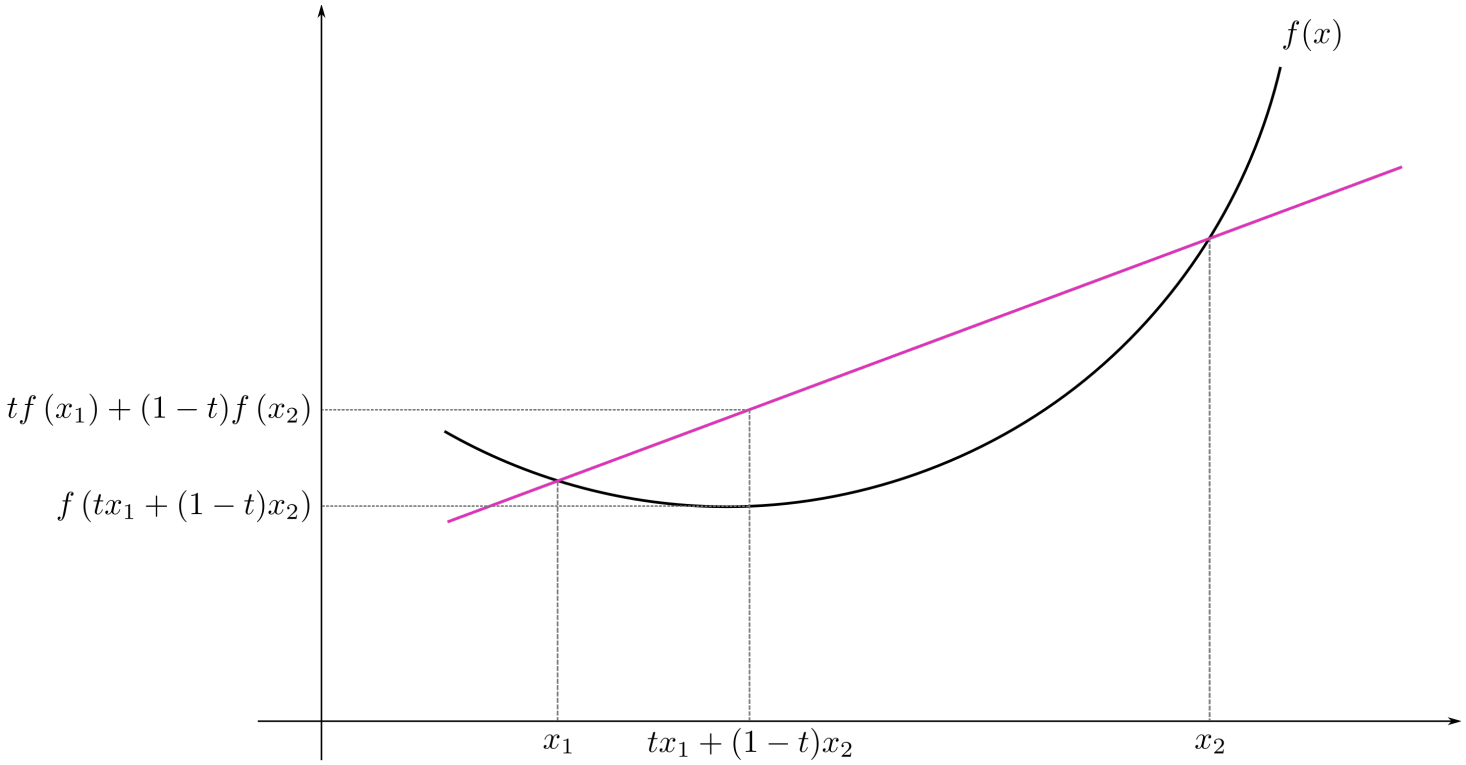
$$\mathbf{w}^{(k+1)} \longleftarrow \mathbf{w}^{(k)} - \alpha \nabla_{\mathbf{w}} \mathbf{Loss}(\mathbf{w}^{(k)}, \mathbf{X}, \mathbf{y})$$

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

Convexity

Convex function f

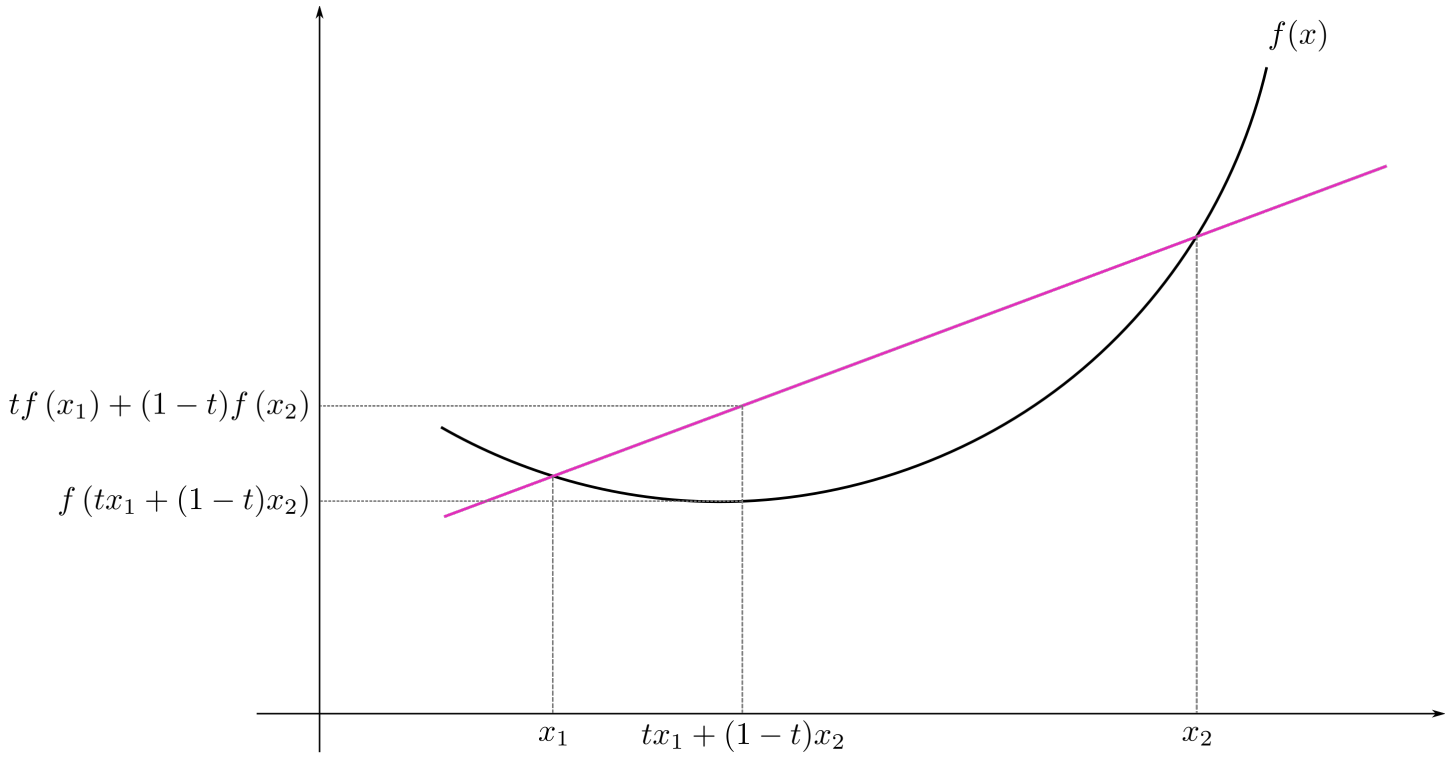
$$f(tx_1 + (1-t)x_2) \leq tf(x_1) + (1-t)f(x_2)$$



Convexity

Strictly convex function f

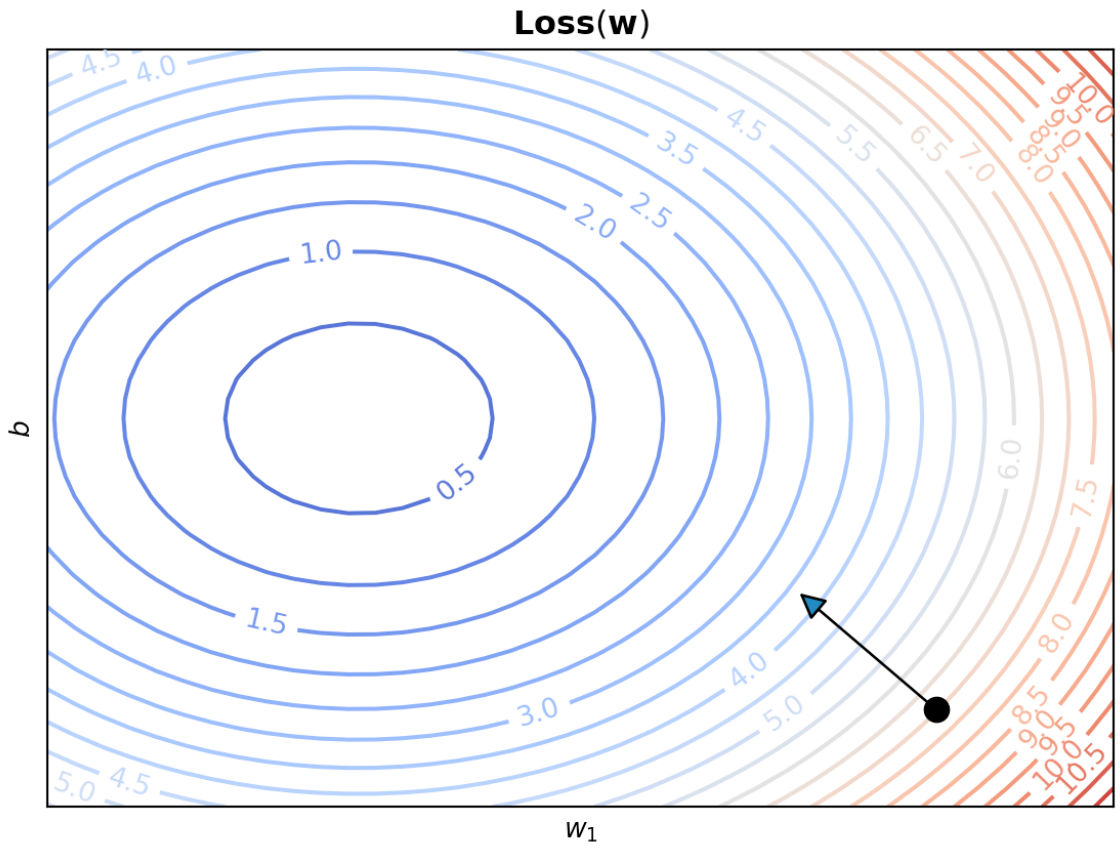
$$f(tx_1 + (1-t)x_2) < tf(x_1) + (1-t)f(x_2)$$



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

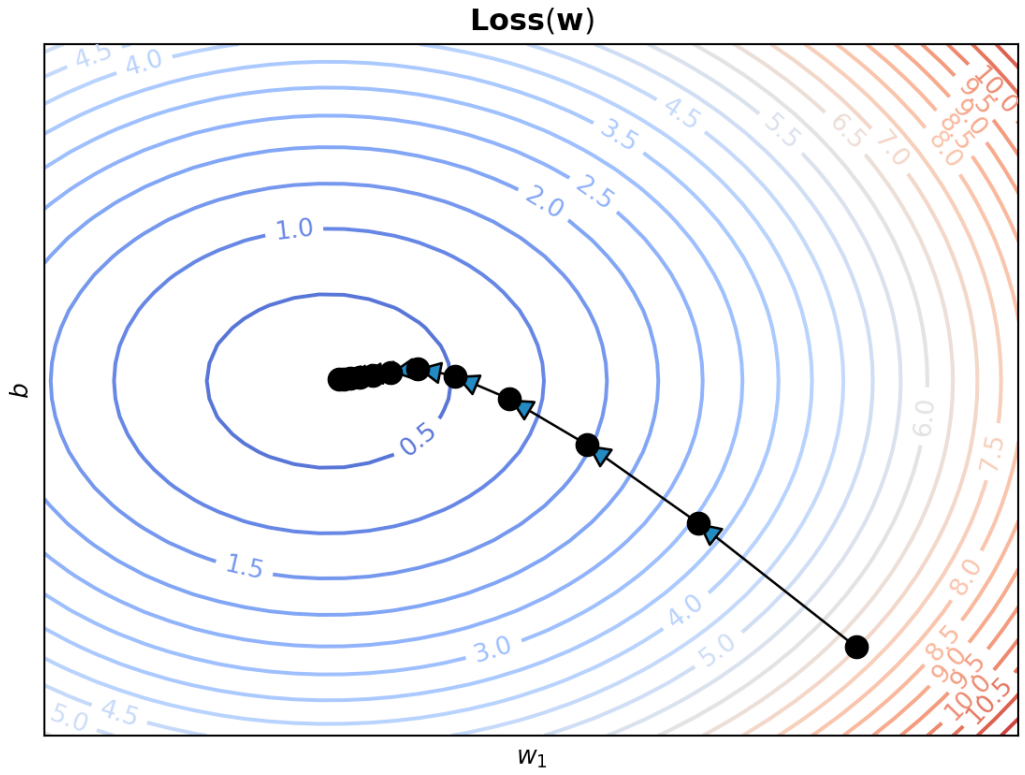
Convexity

Linear and logistic regression losses are convex (usually strictly)



Convexity

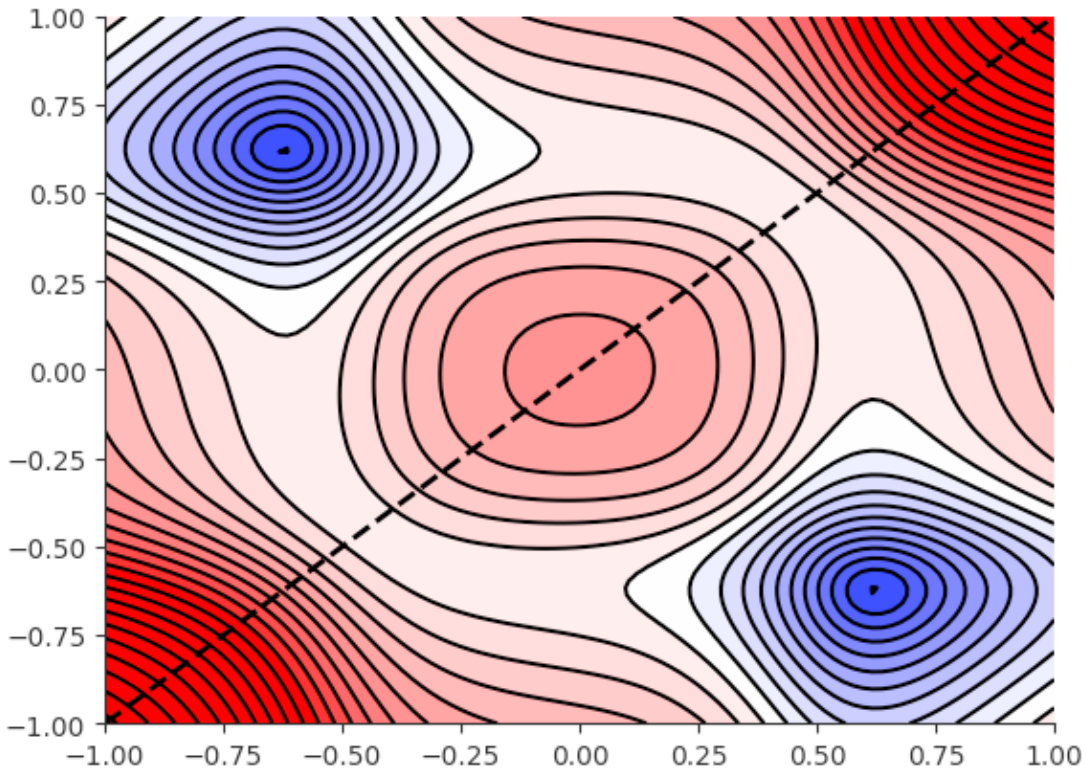
Linear and logistic regression losses are convex (usually strictly)



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

Convexity

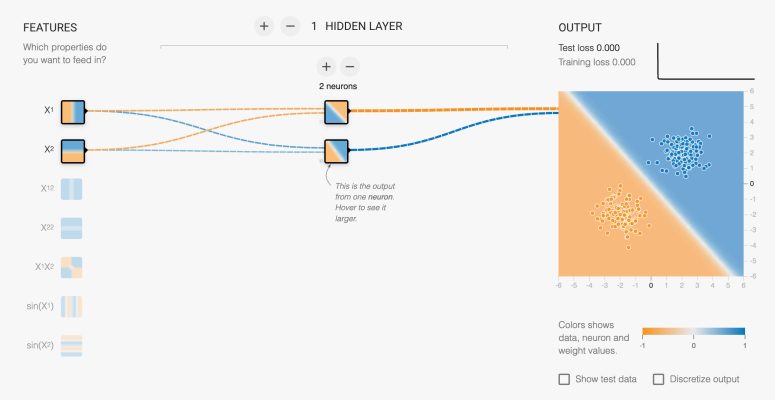
Neural network losses are (usually) not convex!

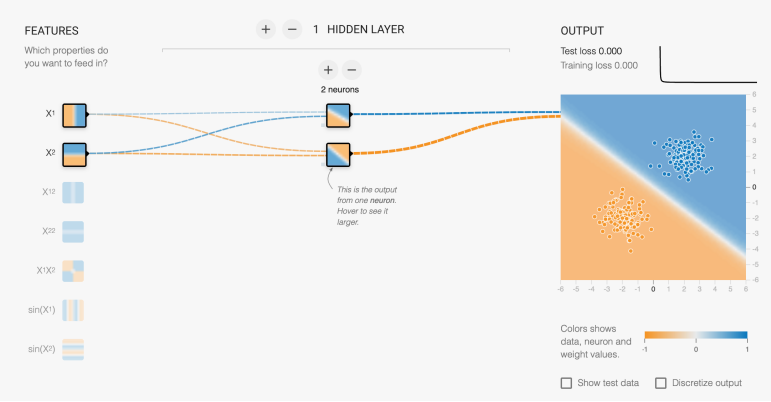


Why?

Convexity

Two equivalent solutions





Symmetry-breaking

A simple network

$$f(\mathbf{x}) = \sigma(\mathbf{x}^T\mathbf{W}_1)^T\mathbf{w}_0 = \sigma(x_1w_{11})w_{01} + \sigma(x_1w_{12})w_{02}$$

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

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

Symmetry-breaking

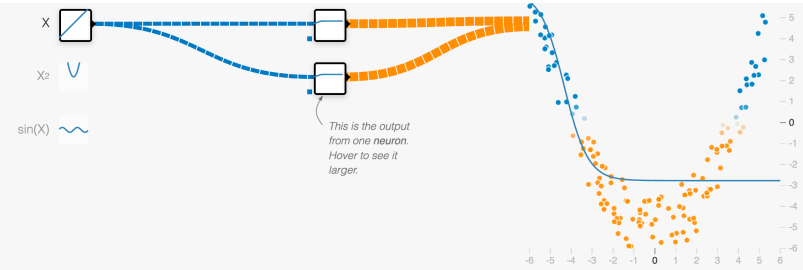
A simple network

$$f(\mathbf{x}) = \sigma(\mathbf{x}^T\mathbf{W}_1)^T\mathbf{w}_0 = \sigma(x_1w_{11})w_{01} + \sigma(x_1w_{12})w_{02}$$

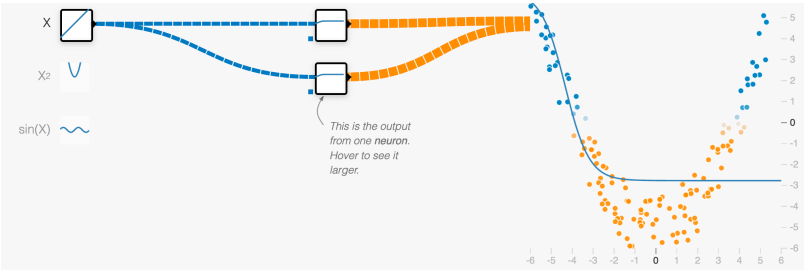
$$\frac{d}{dw_{01}}f(\mathbf{x}) = \sigma(x_1w_{11}) = \sigma(x_1a)$$

$$\frac{d}{dw_{02}}f(\mathbf{x}) = \sigma(x_1w_{12}) = \sigma(x_1a)$$

Symmetry-breaking

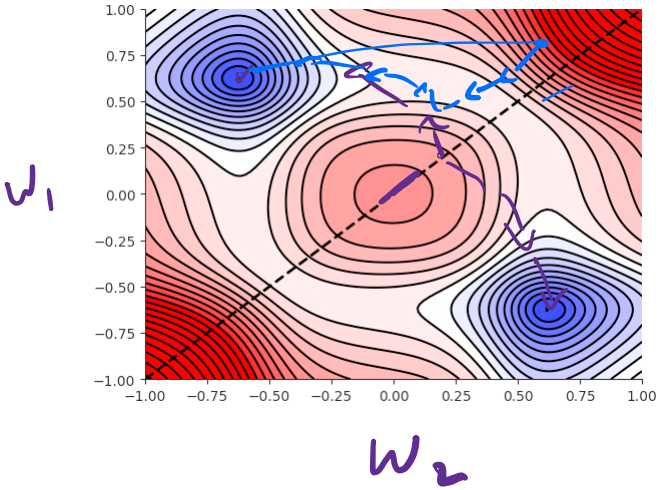


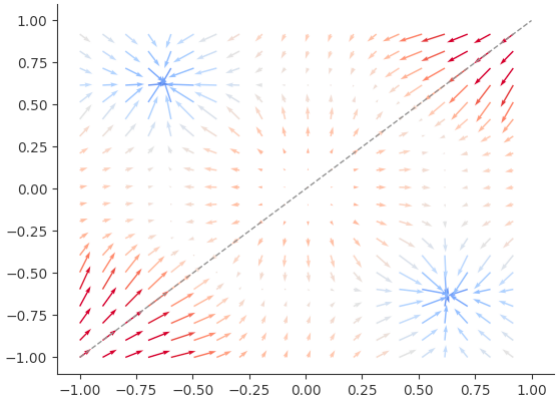
When the network is initialized with symmetry, the two neurons will always have the same output and our solution is poor.



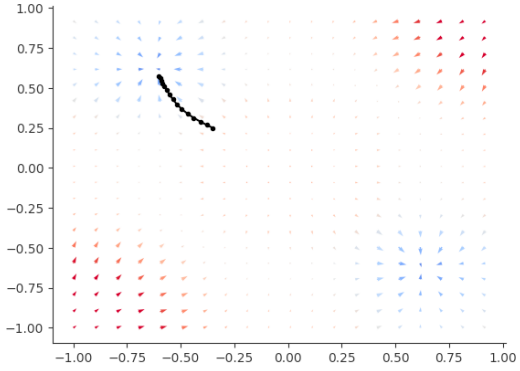
When initialized randomly, the two neurons can create different transforms and a much better solution is found.

Symmetry-breaking

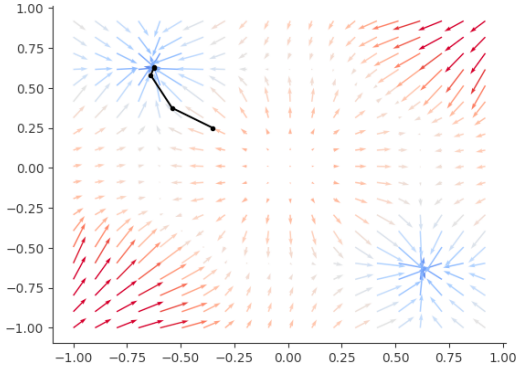




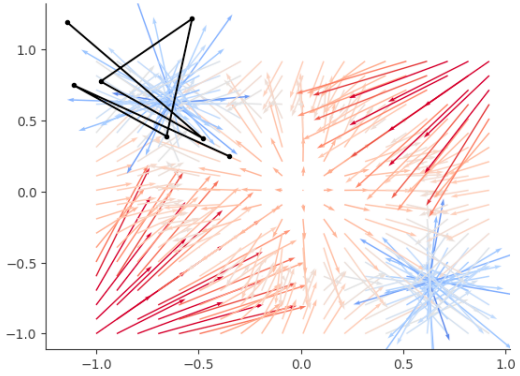
Visualizing learning rates



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?

$w_i \sim \text{Uniform}[0, 1]$

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 $\mathbf{w}$:

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

$$\mathrm{Var}\left[f(\mathbf{x})\right] =$$

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 $\mathbf{w}$:

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

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

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 $\mathbf{w}$:

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

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

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

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

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

$$= \frac{1}{d} \sum_{i=1}^{d} x_i^2$$

Scaled initialization

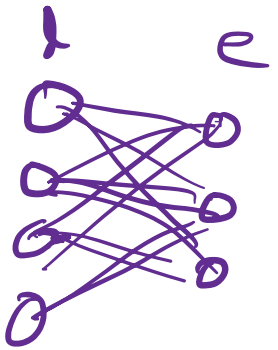
Computing the variance

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

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

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

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:

Kaiming init
the init.

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

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

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

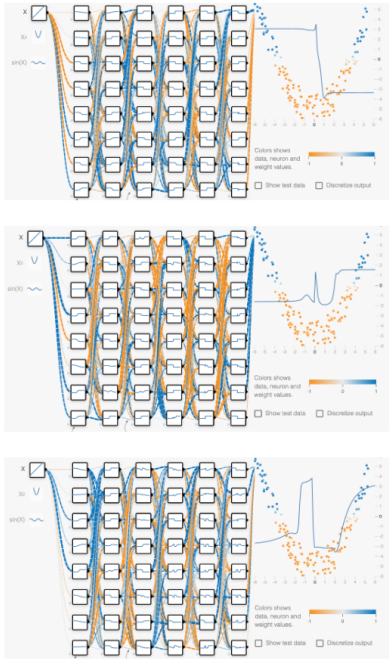
This is known as **Xavier initialization** (or **Glorot initialization** after the inventor Xavier Glorot).

$\alpha w_i \sim \mathcal{N}(0,1) \equiv w_i \sim \mathcal{N}(0,\alpha)$

Scaled initialization

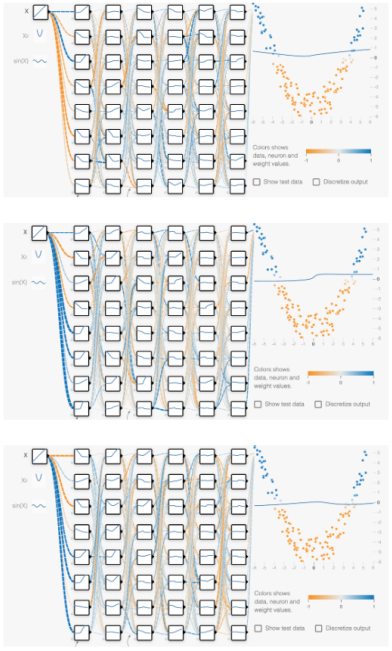
Standard normal

$$w_i \sim \mathcal{N}\left(0, 1\right)$$



Kaiming normal

$$w_i \sim \mathcal{N}\left(0, \frac{1}{\sqrt{d}}\right)$$

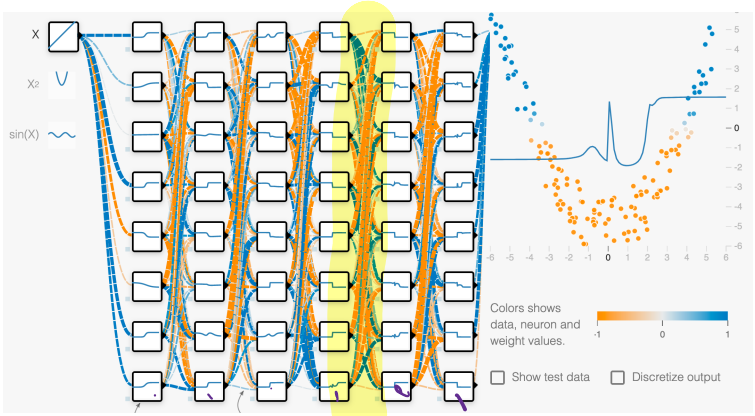


Stochastic Gradient Descent

Computational complexity

- **Dataset size:** N
- **Dimensionality :** d
- **Number of layers:** L
- **Number of steps:** S

We can consider one of the networks shown above as a specific example:



Computational complexity

Loss function

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

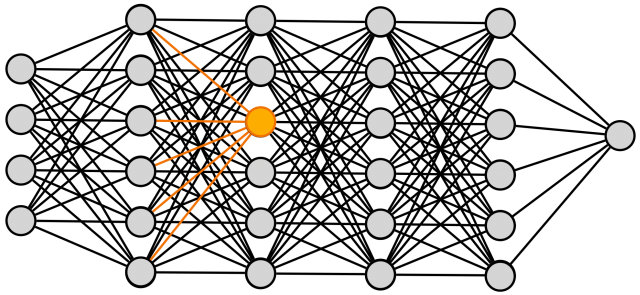
Gradient with respect to the parameters, $\mathbf{w}$.

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

Our analysis would be equivalent for classification problems.

Computing a single neuron

$$\phi(\mathbf{x}) = \sigma\left(\mathbf{x}^T\mathbf{w}\right) = \sigma\left(\sum_{i=1}^{d} x_i w_i\right)$$

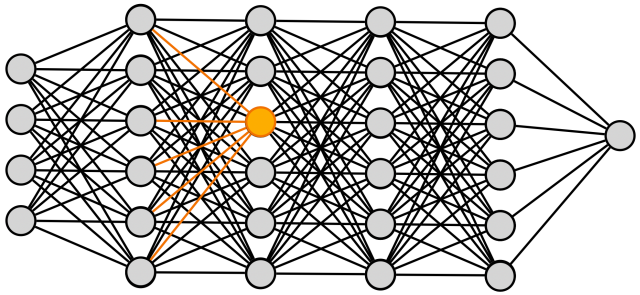


Activation function $\sigma(\cdot)$: Constant, $\mathcal{O}(1)$

Summation: $\sum_{i=1}^{d} x_i w_i$: Linear, $\mathcal{O}(d)$.

Computing a single neuron

$$\phi(\mathbf{x}) = \sigma\left(\mathbf{x}^T\mathbf{w}\right) = \sigma\left(\sum_{i=1}^{d} x_i w_i\right)$$



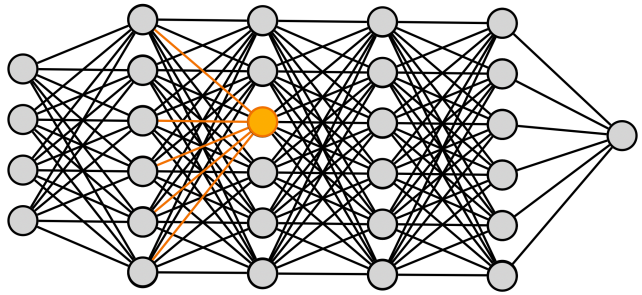
Backward pass?

- Given: $\frac{dLoss}{d\phi}$
- Update: $\frac{dLoss}{dx_i}$ $\frac{dLoss}{dw_i}$

Computing a single neuron

Backward pass?

- Given: $\frac{dLoss}{d\phi}$
- Update: $\frac{dLoss}{dx_i}$ $\frac{dLoss}{dw_i}$



Activation function:

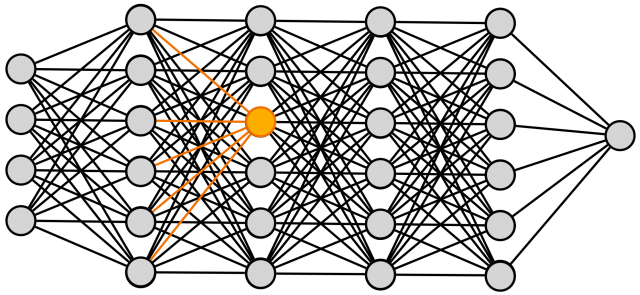
$$\frac{dLoss}{d(\mathbf{x}^T\mathbf{w})} = \frac{dLoss}{d\phi}\frac{d\phi}{d(\mathbf{x}^T\mathbf{w})}$$

All scalars! $\mathcal{O}(1)$

Computing a single neuron

Backward pass?

- Given: $\frac{dLoss}{d(\mathbf{x}^T\mathbf{w})}$
- Update: $\frac{dLoss}{dx_i}$ $\frac{dLoss}{dw_i}$



$$\mathbf{x}^T\mathbf{w} = \sum_{i=1}^{d} x_i w_i$$

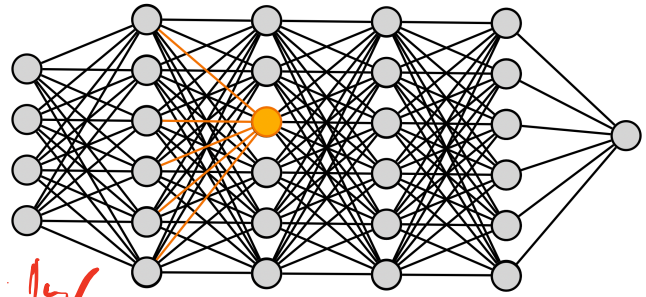
$\frac{dLoss}{dw_i} =$

Computing a single neuron

Backward pass?

- Given: $\frac{dLoss}{d(\mathbf{x}^T\mathbf{w})}$
- Update: $\frac{dLoss}{dx_i}$ $\frac{dLoss}{dw_i}$

$$w^T x = \sum_{i=1}^{d} x_i w_i$$



Scalar

Scalar

$$\frac{dLoss}{dx_i} = \frac{dLoss}{d(\mathbf{x}^T\mathbf{w})}\frac{d(\mathbf{x}^T\mathbf{w})}{dx_i} = \frac{dLoss}{d(\mathbf{x}^T\mathbf{w})}w_i$$

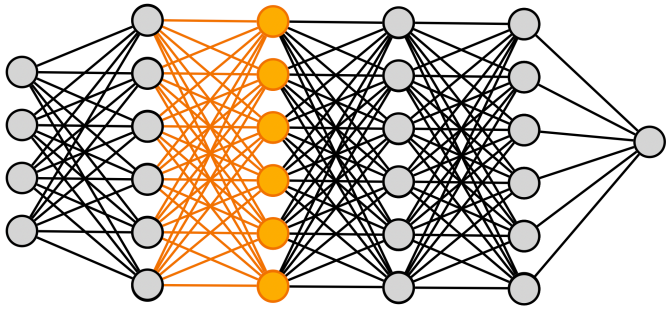
$$\frac{dLoss}{dw_i} = \frac{dLoss}{d(\mathbf{x}^T\mathbf{w})}\frac{d(\mathbf{x}^T\mathbf{w})}{dw_i} = \frac{dLoss}{d(\mathbf{x}^T\mathbf{w})}x_i$$

$\mathcal{O}(1)$ per entry. Total: $\mathcal{O}(d)$.

Cost per layer

$\mathcal{O}(d)$ Neurons per layer

Total cost: $\mathcal{O}(d^2)$



Full network: $\mathcal{O}(Ld^2)$

Computational complexity

We've bounded the time it takes to compute *one* of our predictions: $f(\mathbf{x}_i, \mathbf{w})$.

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

one pred.

N terms in summation, total cost of $\mathcal{O}(NLd^2)$ for a single gradient descent update:

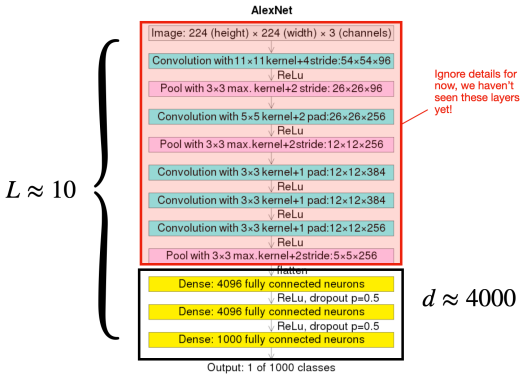
$$\mathbf{w}^{(k+1)} \longleftarrow \mathbf{w}^{(k)} - \alpha \nabla_{\mathbf{w}} \mathbf{Loss}(\mathbf{w}^{(k)}, \mathbf{X}, \mathbf{y})$$

Total cost of gradient descent: $\mathcal{O}(SNLd^2)$.

AlexNet (2012)

- **Dataset size:** $N =$
- **Dimensionality :** $d =$ 4000
- **Number of layers:** $L =$ 10
- **Number of steps:** $S =$

Network:



AlexNet (2012)

- **Dataset size:** $N = 10,000,000$
- **Dimensionality :** $d = 4000$
- **Number of layers:** $L = 10$
- **Number of steps:** $S =$

14,197,122 images

One step

$O(LNd^2)$

$(10)(10,000,000)(4000)(4000)$

Estimating loss

Neural network MSE loss:

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

Estimating loss

Neural network MSE loss:

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

Estimate by sampling:

$$\underset{\text{MSE}}{\mathbf{Loss}}(\mathbf{w}, \mathbf{X}, \mathbf{y}) \approx (f(\mathbf{x}_i, \mathbf{w}) - y_i)^2, \quad i \sim \text{Uniform}(1, N)$$

Estimating loss

$$\underset{\text{MSE}}{\mathbf{Loss}}(\mathbf{w}, \mathbf{X}, \mathbf{y}) \approx (f(\mathbf{x}_i, \mathbf{w}) - y_i)^2, \quad i \sim \text{Uniform}(1, N)$$

Expectation of sampled loss

$$\mathbb{E}_i[(f(\mathbf{x}_i, \mathbf{w}) - y_i)^2] = ?$$

$$E[X] = \sum_{i=1}^{N} P(X=i) X$$

$$E[f(X)] = \sum_{i=1}^{N} P(X=i) f(X)$$

$$= \sum_{i=1}^{N} \left(\frac{1}{N}\right) (f(x_i, w) - y_i)^2$$

$$= \frac{1}{N} \sum_{i=1}^{N} (f(x_i, w) - y_i)^2$$

$$= \text{Loss}$$

Estimating loss

$$\underset{\text{MSE}}{\mathbf{Loss}}(\mathbf{w}, \mathbf{X}, \mathbf{y}) \approx (f(\mathbf{x}_i, \mathbf{w}) - y_i)^2, \quad i \sim \text{Uniform}(1, N)$$

Expectation of sampled loss is the true loss!

$$\mathbb{E}_i[(f(\mathbf{x}_i, \mathbf{w}) - y_i)^2] = \sum_{i=1}^{N} p(i)(f(\mathbf{x}_i, \mathbf{w}) - y_i)^2$$

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

Estimating loss

In general any loss that can be written as a mean of individual losses can be estimated in this way:

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

$$\mathbf{Loss}(\mathbf{w}, \mathbf{X}, \mathbf{y}) = \mathbb{E}[\mathbf{Loss}(\mathbf{w}, \mathbf{x}_i, y_i)],\ i \sim \text{Uniform}(1, N)$$

Cost for full loss: $\mathcal{O}(N)$

Cost of *estimated* loss: $\mathcal{O}(1)$

Cost of GD step -> $O(LN^2)$

Estimating gradients

Gradient descent update:

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

Gradient can be composed into a sum of gradients and estimated the same way!

$$\nabla_{\mathbf{w}} \mathbf{Loss}(\mathbf{w}, \mathbf{X}, \mathbf{y}) = \nabla_{\mathbf{w}} \left(\frac{1}{N} \sum_{i=1}^{N} \mathbf{Loss}(\mathbf{w}, \mathbf{x}_i, y_i) \right)$$

$$= \frac{1}{N} \sum_{i=1}^{N} \nabla_{\mathbf{w}} \mathbf{Loss}(\mathbf{w}, \mathbf{x}_i, y_i)$$

Estimating gradients

$$\nabla_{\mathbf{w}} \mathbf{Loss}(\mathbf{w}, \mathbf{X}, \mathbf{y}) = \nabla_{\mathbf{w}} \left(\frac{1}{N} \sum_{i=1}^{N} \mathbf{Loss}(\mathbf{w}, \mathbf{x}_i, y_i) \right)$$

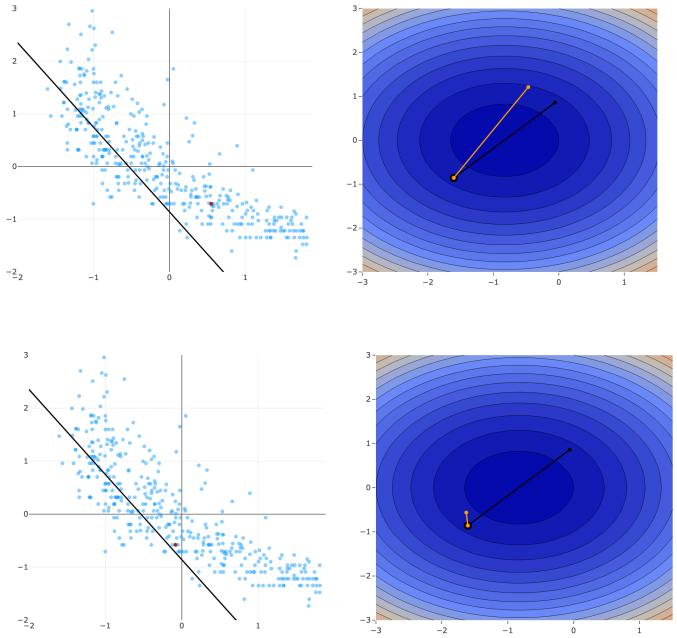
$$= \frac{1}{N} \sum_{i=1}^{N} \nabla_{\mathbf{w}} \mathbf{Loss}(\mathbf{w}, \mathbf{x}_i, y_i) = \mathbb{E}[\nabla_{\mathbf{w}} \mathbf{Loss}(\mathbf{w}, \mathbf{x}_i, y_i)])$$

Stochastic gradient descent update:

$$\mathbf{w}^{(k+1)} \longleftarrow \mathbf{w}^{(k)} - \alpha \nabla_{\mathbf{w}} \mathbf{Loss}(\mathbf{w}^{(k)}, \mathbf{x}_i, y_i)$$

$$i \sim \text{Uniform}(1, N)$$

Estimating gradients



Minibatch SGD

Can estimate gradients with a *minibatch* of B observations:

$$\text{Batch: } \{(\mathbf{x}_{b_1}, y_{b_1}), (\mathbf{x}_{b_2}, y_{b_2}), \ldots, (\mathbf{x}_{b_B}, y_{b_B})\}$$

$$\{b_1, b_2, \ldots, b_B\} \sim \text{Uniform}(1, N)$$

New estimate:

$$\nabla_{\mathbf{w}} \mathbf{Loss}(\mathbf{w}, \mathbf{X}, \mathbf{y}) \approx \frac{1}{B} \sum_{i=1}^{B} \nabla_{\mathbf{w}} \mathbf{Loss}(\mathbf{w}, \mathbf{x}_{b_i}, y_{b_i})$$

$$\{b_1, b_2, \ldots, b_B\} \sim \text{Uniform}(1, N)$$

Minibatch SGD

Does this still give the correct expectation?

$$\mathbb{E}\left[\frac{1}{B}\sum_{i=1}^{B}\nabla_{\mathbf{w}}\mathbf{Loss}(\mathbf{w},\mathbf{x}_{b_i},y_{b_i})\right]=?$$

Minibatch SGD

Does this still give the correct expectation?

$$\mathbb{E}\left[\frac{1}{B}\sum_{i=1}^{B}\nabla_{\mathbf{w}}\mathbf{Loss}(\mathbf{w},\mathbf{x}_{b_i},y_{b_i})\right]$$

$$=\left(\frac{1}{B}\right)\sum_{i=1}^{B}\mathbb{E}\left[\nabla_{\mathbf{w}}\mathbf{Loss}(\mathbf{w},\mathbf{x}_{b_i},y_{b_i})\right]$$

$$=\nabla_{\mathbf{w}}\mathbf{Loss}(\mathbf{w},\mathbf{X},\mathbf{y})$$

Minibatch SGD

What about variance?

$$\mathrm{Var}\left[\frac{1}{B}\sum_{i=1}^{B}\nabla_{\mathbf{w}}\mathbf{Loss}(\mathbf{w}, \mathbf{x}_{b_i}, y_{b_i})\right] = ?$$

Minibatch SGD

The variance decreases with the size of the batch!

$$\text{Var}\left[\frac{1}{B}\sum_{i=1}^{B}\nabla_{\mathbf{w}}\mathbf{Loss}(\mathbf{w},\mathbf{x}_{b_i},y_{b_i})\right]$$

$$=\left(\frac{1}{B^2}\right)\sum_{i=1}^{B}\text{Var}\left[\nabla_{\mathbf{w}}\mathbf{Loss}(\mathbf{w},\mathbf{x}_{b_i},y_{b_i})\right]$$

$$=\left(\frac{1}{B}\right)\text{Var}\left[\nabla_{\mathbf{w}}\mathbf{Loss}(\mathbf{w},\mathbf{x}_{b_i},y_{b_i})\right]$$

Minibatch SGD