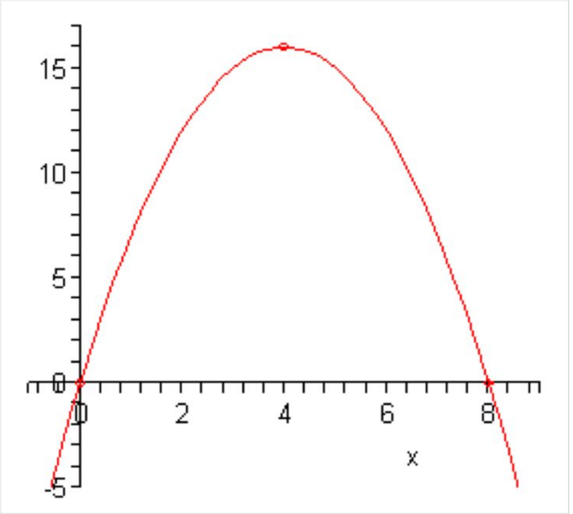
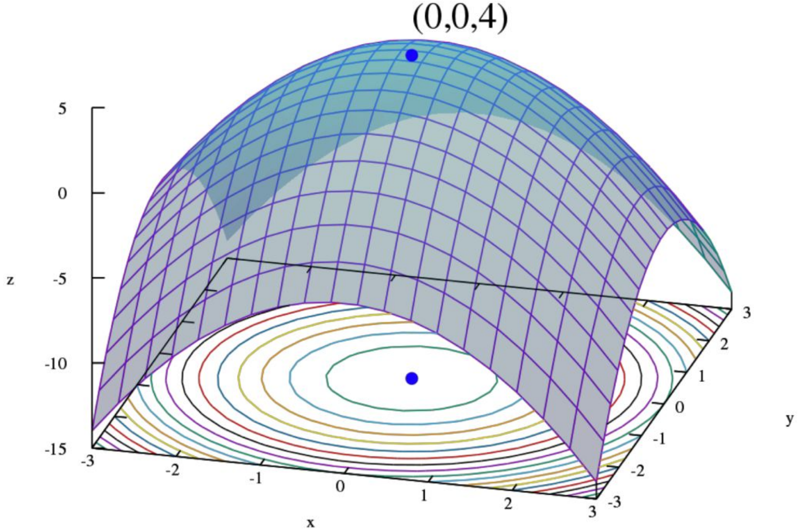
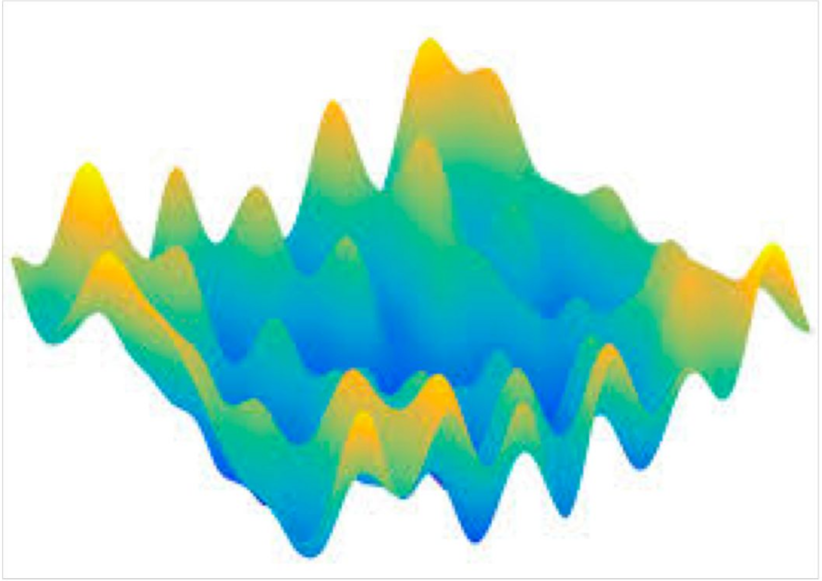
Section 3

Niranjan Balachandar, Richard Martinez

Why Learning?







Datapoints

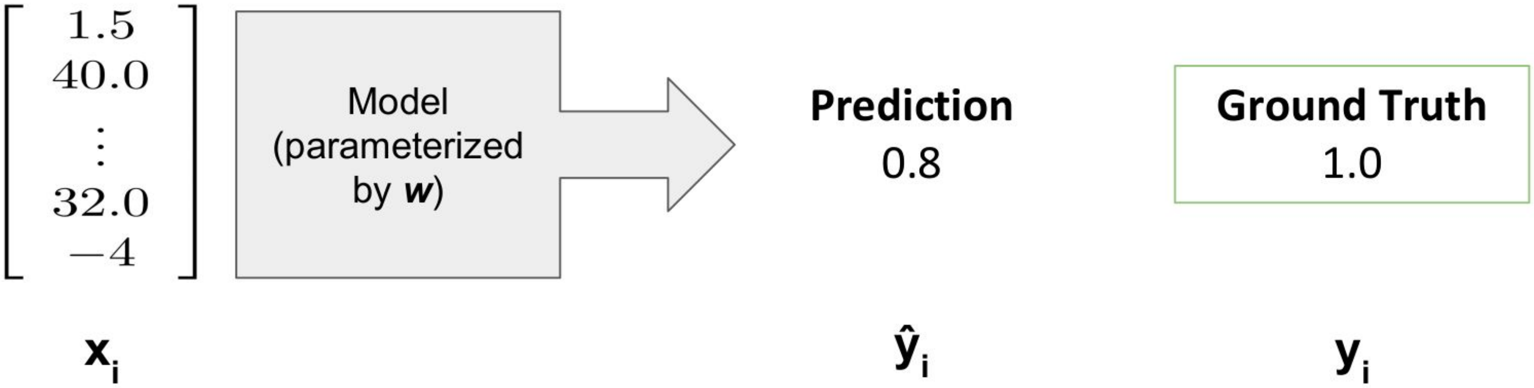
$$\begin{bmatrix} 1.5 \\ 40.0 \\ \vdots \\ 32.0 \\ -4 \end{bmatrix}$$

$\mathbf{x}_i$

Ground Truth
1.0

$\mathbf{y}_i$

Model



$\begin{bmatrix} 1.5 \\ 40.0 \\ \vdots \\ 32.0 \\ -4 \end{bmatrix}$
Model (parameterized by ***w***)
Prediction
0.8
Ground Truth
1.0
$\mathbf{x}_i$
$\hat{\mathbf{y}}_i$
$\mathbf{y}_i$

Binary Classification

Let's review binary classification

$$x \longrightarrow \boxed{f_{\mathbf{w}}} \longrightarrow y \in \{-1, +1\}$$

Score:

$$\text{score}_{+1}(x, \mathbf{w}) = \mathbf{w} \cdot \phi(x)$$
$$\text{score}_{-1}(x, \mathbf{w}) = (-\mathbf{w}) \cdot \phi(x)$$

Prediction:

$$f_{\mathbf{w}}(x) = \begin{cases} +1 & \text{if } \text{score}_{+1}(x, \mathbf{w}) > \text{score}_{-1}(x, \mathbf{w}) \\ -1 & \text{otherwise} \end{cases}$$

$$f_{\mathbf{w}}(x) = \arg \max_{y \in \{-1,+1\}} \text{score}_y(x, \mathbf{w})$$

Multiclass Classification

Problem

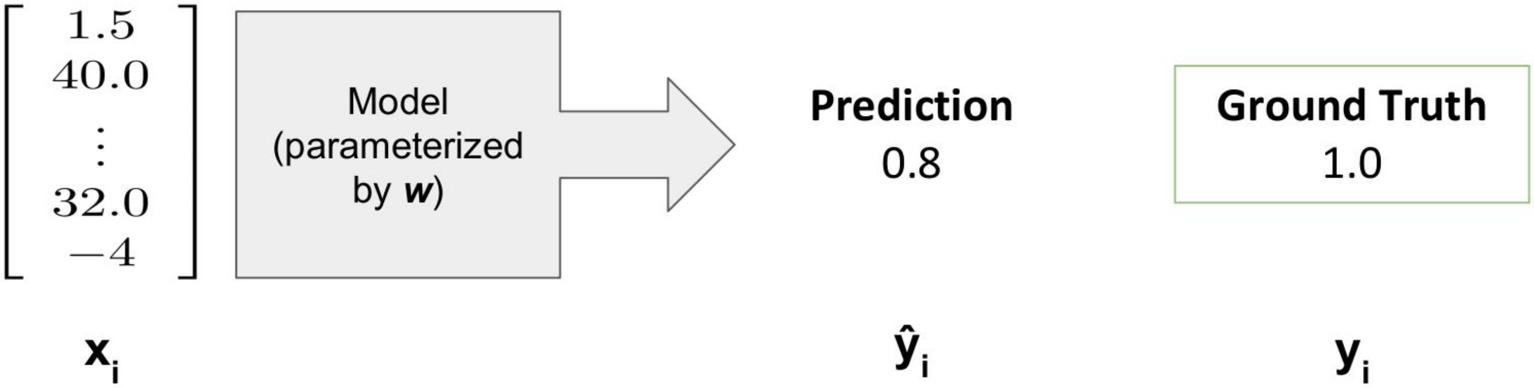
Suppose we have 3 possible labels $y \in \{\mathsf{R}, \mathsf{G}, \mathsf{B}\}$

Weight vectors: $\mathbf{w} = \{\mathbf{w}_{\mathsf{R}}, \mathbf{w}_{\mathsf{G}}, \mathbf{w}_{\mathsf{B}}\}$

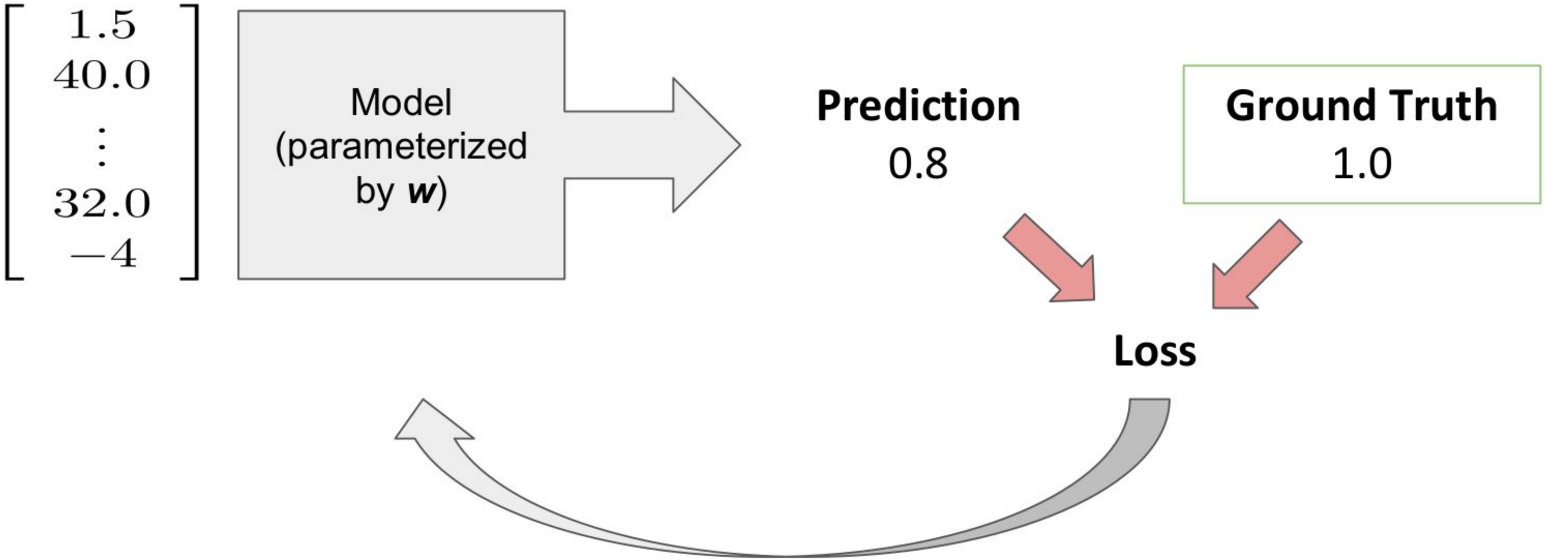
Scores: $[\mathbf{w}_{\mathsf{R}} \cdot \phi(x)], [\mathbf{w}_{\mathsf{G}} \cdot \phi(x)], [\mathbf{w}_{\mathsf{B}} \cdot \phi(x)]$

Prediction: $\hat{y} = f_{\mathbf{w}}(x) = \arg \max_{y \in \{\mathsf{R},\mathsf{G},\mathsf{B}\}} [\mathbf{w}_y \cdot \phi(x)]$

Model

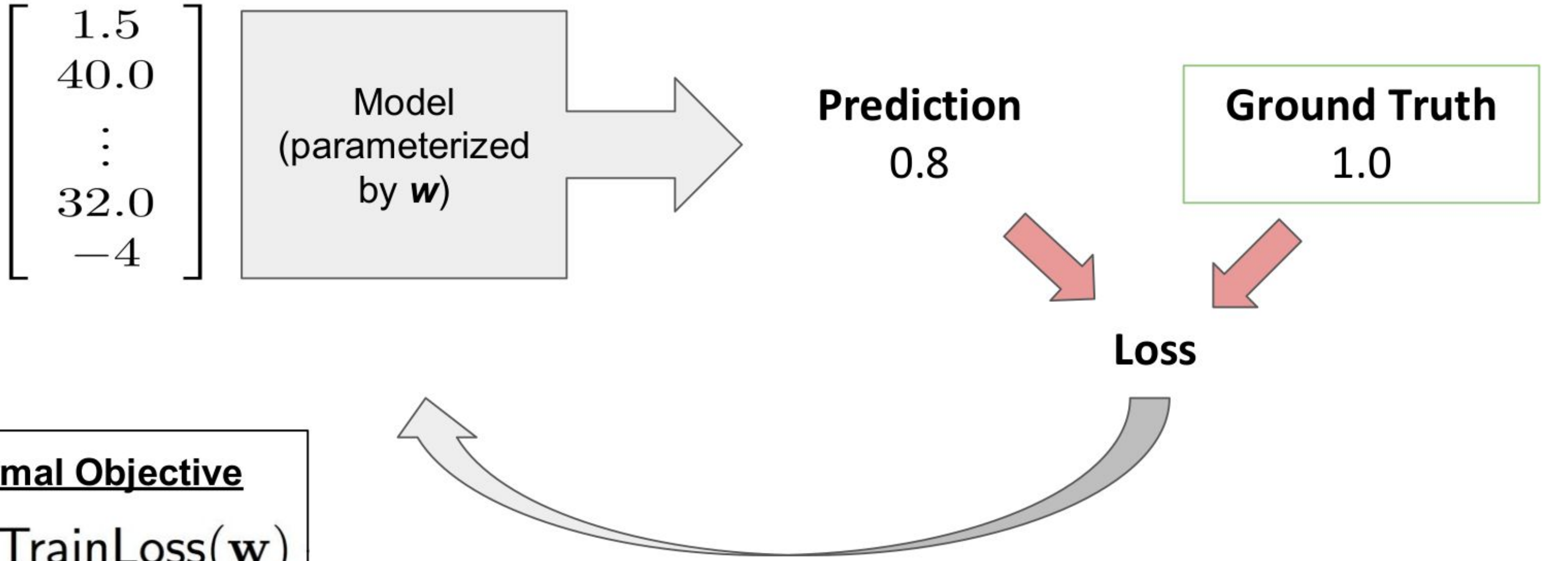

1.5
40.0
⋮
32.0
−4
Model
(parameterized
by *w*)
Prediction
0.8
Ground Truth
1.0
$\mathbf{x}_i$
$\hat{\mathbf{y}}_i$
$\mathbf{y}_i$

Loss



Key idea: Use loss to inform updates to weights.

Loss



Formal Objective

$$\min_{\mathbf{w}} \text{TrainLoss}(\mathbf{w})$$

Key idea: Use loss to inform updates to weights.

Loss Functions

How to learn $\mathbf{w}$?

How about **0-1 loss**:

$$\text{Loss}_{\text{0-1}}(x, y, \mathbf{w}) = \begin{cases} 1 & \text{if } \hat{y} \neq y \\ 0 & \text{otherwise} \end{cases}$$

Problem: Gradient is 0 almost everywhere

Hinge Loss

How to learn $\mathbf{w}$?

Recall **hinge loss**:

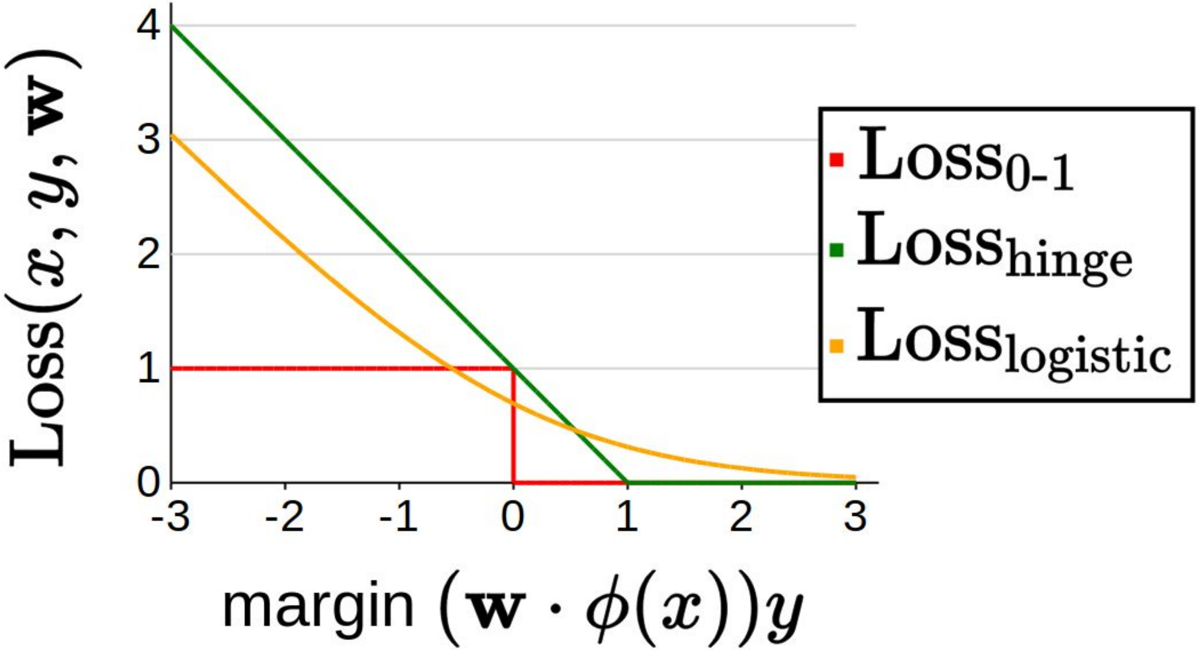
$$\text{margin} = \text{score}_y(x, \mathbf{w}) - \max_{y' \neq y} \text{score}_{y'}(x, \mathbf{w})$$

$$\text{Loss}_{\text{Hinge}}(x, y, \mathbf{w}) = \max\{1 - \text{margin}, 0\}$$

What is the gradient?

Logistic Regression

$$\text{Loss}_{\text{logistic}}(x, y, \mathbf{w}) = \log(1 + e^{-(\mathbf{w}\cdot\phi(x))y})$$

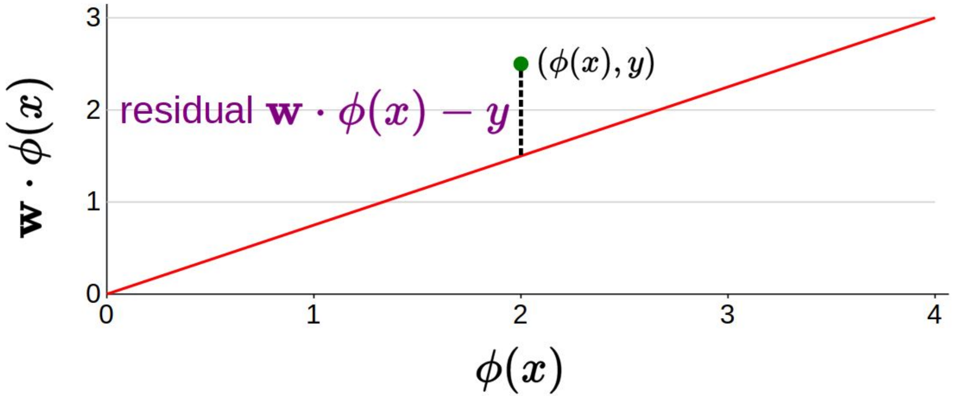


Cross-entropy loss

$$L_{\text{cross-entropy}}(\hat{\mathbf{y}}, \mathbf{y}) = -\sum_i y_i \log(\hat{y}_i)$$

Regression

$$f_{\mathbf{w}}(x) = \mathbf{w} \cdot \phi(x)$$



Definition: residual

The **residual** is $(\mathbf{w} \cdot \phi(x)) - y$, the amount by which prediction $f_{\mathbf{w}}(x) = \mathbf{w} \cdot \phi(x)$ overshoots the target y.

Regression losses

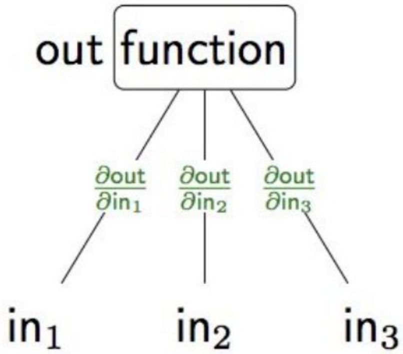
$$L1LossFunction = \sum_{i=1}^{n} |y_{true} - y_{predicted}|$$

$$L2LossFunction = \sum_{i=1}^{n} (y_{true} - y_{predicted})^2$$

When would you use L1 (absolute) vs L2 (squared) loss?

Partial Derivatives / Gradients

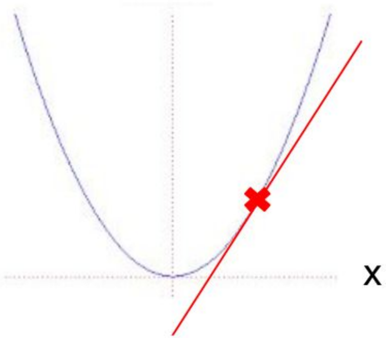
We want to know how each weight affects the training loss.
→ exactly what derivative (gradient in the vector case) tells us!



Partial derivatives (gradients): how much does the output change if an input changes?

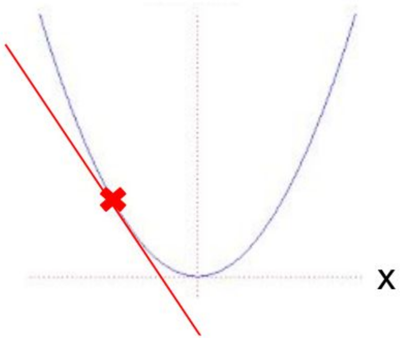
Gradient Descent

We want to know how each weight affects the training loss.
→ exactly what derivative (gradient in the vector case) tells us!



Positive gradient =
decrease x

$y=x^2$



Negative gradient =
increase x

Gradient Descent

Algorithm: gradient descent

Initialize $\mathbf{w} = [0, \ldots, 0]$

For $t = 1, \ldots, T$:

$$\mathbf{w} \leftarrow \mathbf{w} - \underbrace{\eta}_{\text{step size}} \underbrace{\nabla_{\mathbf{w}} \text{TrainLoss}(\mathbf{w})}_{\text{gradient}}$$

Gradient Descent

Algorithm: gradient descent

Initialize $\mathbf{w} = [0, \ldots, 0]$

For $t = 1, \ldots, T$:

$$\mathbf{w} \leftarrow \mathbf{w} - \underbrace{\eta}_{\text{step size}} \underbrace{\nabla_{\mathbf{w}} \text{TrainLoss}(\mathbf{w})}_{\text{gradient}}$$

$$\text{TrainLoss}(\mathbf{w}) = \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{(x,y) \in \mathcal{D}_{\text{train}}} \text{Loss}(x, y, \mathbf{w})$$

Stochastic Gradient Descent

Algorithm: stochastic gradient descent

Initialize $\mathbf{w} = [0, \ldots, 0]$

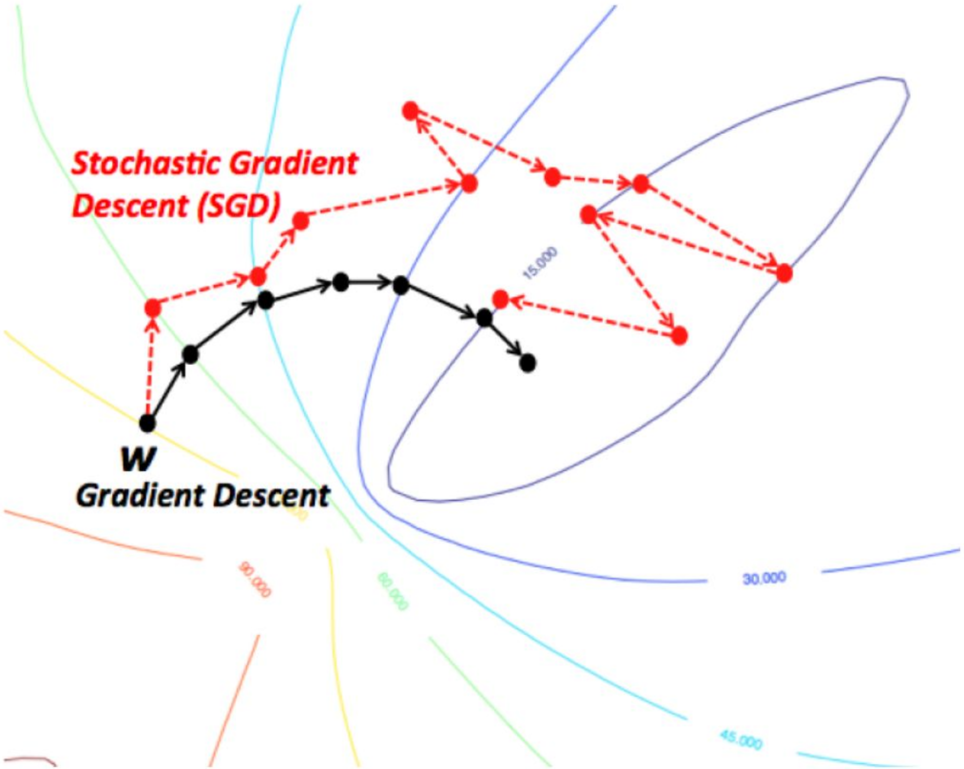
For $t = 1, \ldots, T$:

For $(x, y) \in \mathcal{D}_{\text{train}}$:

$\mathbf{w} \leftarrow \mathbf{w} - \eta_t \nabla_{\mathbf{w}} \text{Loss}(x, y, \mathbf{w})$

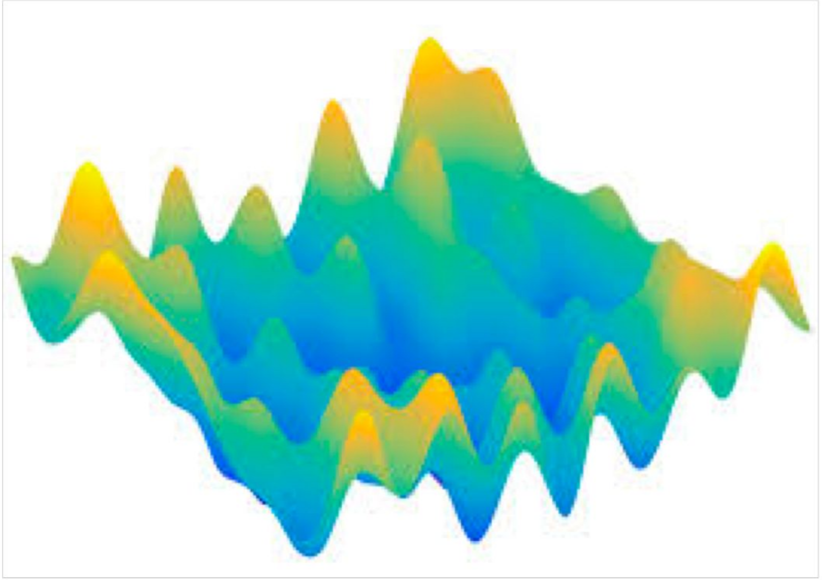
GD vs SGD?

GD vs SGD?

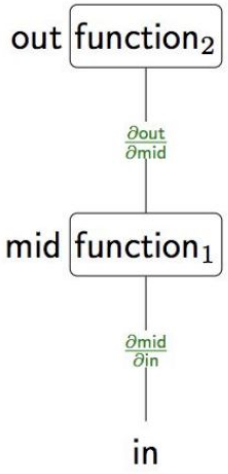

Stochastic Gradient
Descent (SGD)
w
Gradient Descent
15,000
30,000
45,000
60,000
90,000

Other optimizers

- Minibatch GD
- SGD with Momentum
- Adam



Backpropagation



Chain rule: $\frac{\partial \text{out}}{\partial \text{in}} = \frac{\partial \text{out}}{\partial \text{mid}} \frac{\partial \text{mid}}{\partial \text{in}}$

Why Backpropagate?

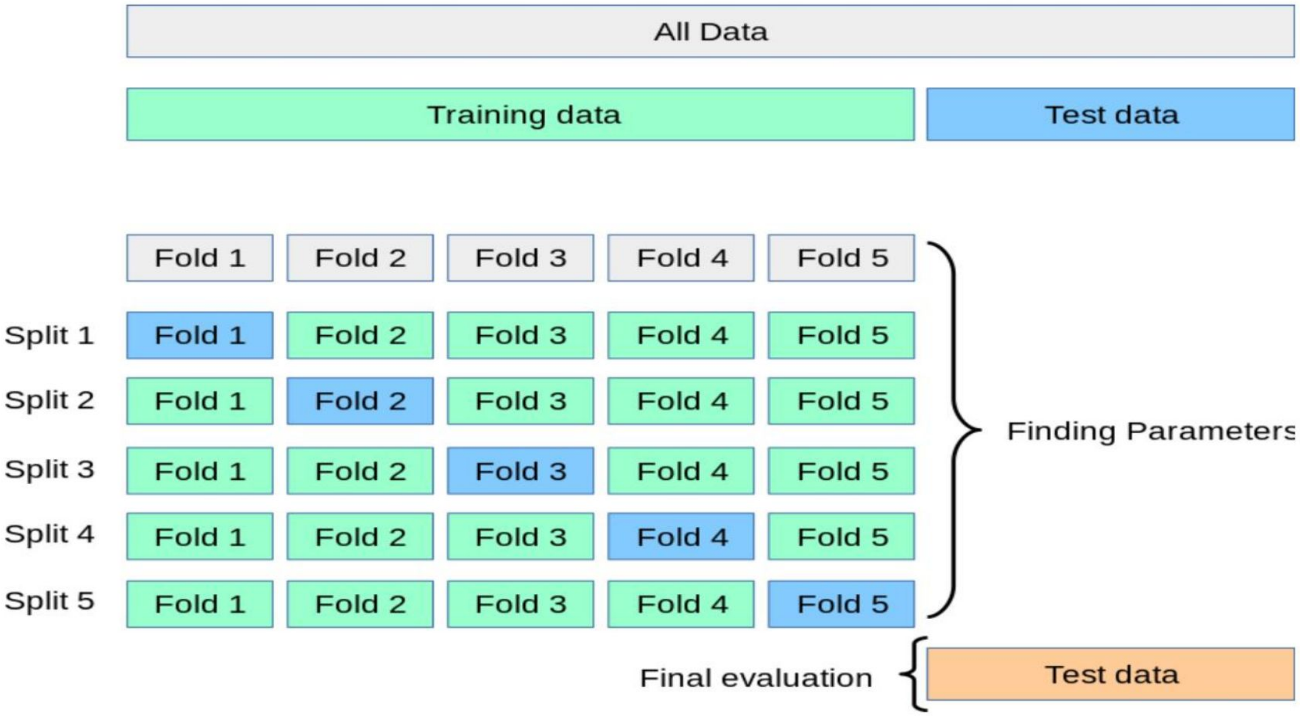
- Don’t have to deal with the nastiness of the chain rule with deep neural networks
- Performance optimizations (can hold onto intermediary values, don’t have to recompute)
- Translates into a modular framework, so packages like Tensorflow and PyTorch will auto-differentiate for you!

Backpropagation

Backprop: http://cs231n.github.io/optimization-2/

Vector, Matrix, and Tensor Derivatives: http://cs231n.stanford.edu/vecDerivs.pdf

k-fold cross-validation



https://scikit-learn.org/stable/modules/cross_validation.html sklearn