

Loss Function and Optimization

CS5330, Huaizu Jiang

Fall 2021, Northeastern University

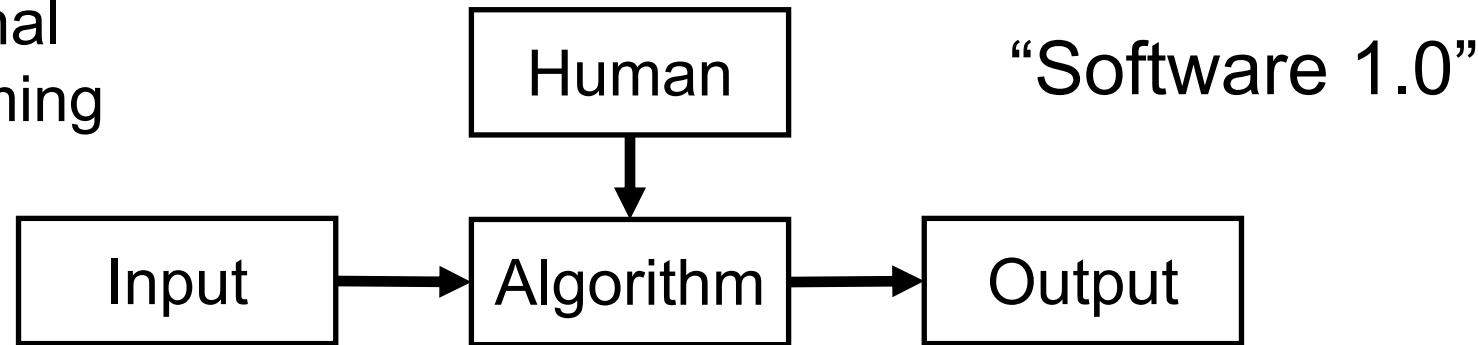
Announcement

- GPU resources for the final project
 - Discovery (**your account should be ready to use**)
 - try ssh <nu username>@login.discovery.neu.edu
 - Documentation of Discovery: <https://rc-docs.northeastern.edu/en/latest/>
 - AWS
 - Google could
- Office hour of Prajanan (**only this week**)
 - Friday, 2-3pm -> Sunday, 10-11pm
- Make-up Office Hour by me (**only this week**)
 - Friday, 2-3pm
- Project proposal abstract due on Nov 1 (**next Monday**)

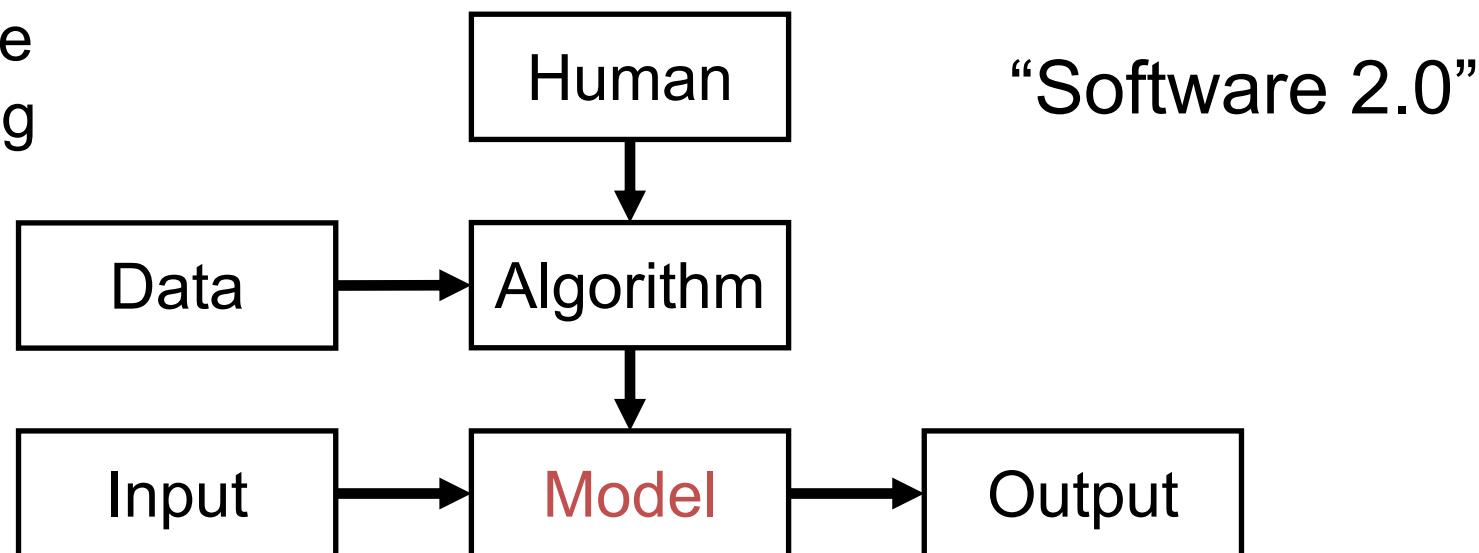
Recap

Machine Learning vs Programming

Traditional
Programming



Machine
Learning



<https://medium.com/@karpathy/software-2-0-a64152b37c35>

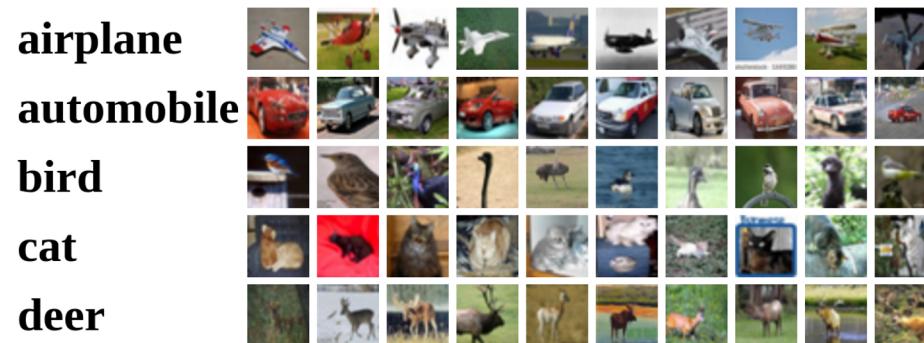
Machine Learning: Data-Driven Approach

1. Collect a large set of data
2. Use Machine Learning to train a model
3. Evaluate the model on new data

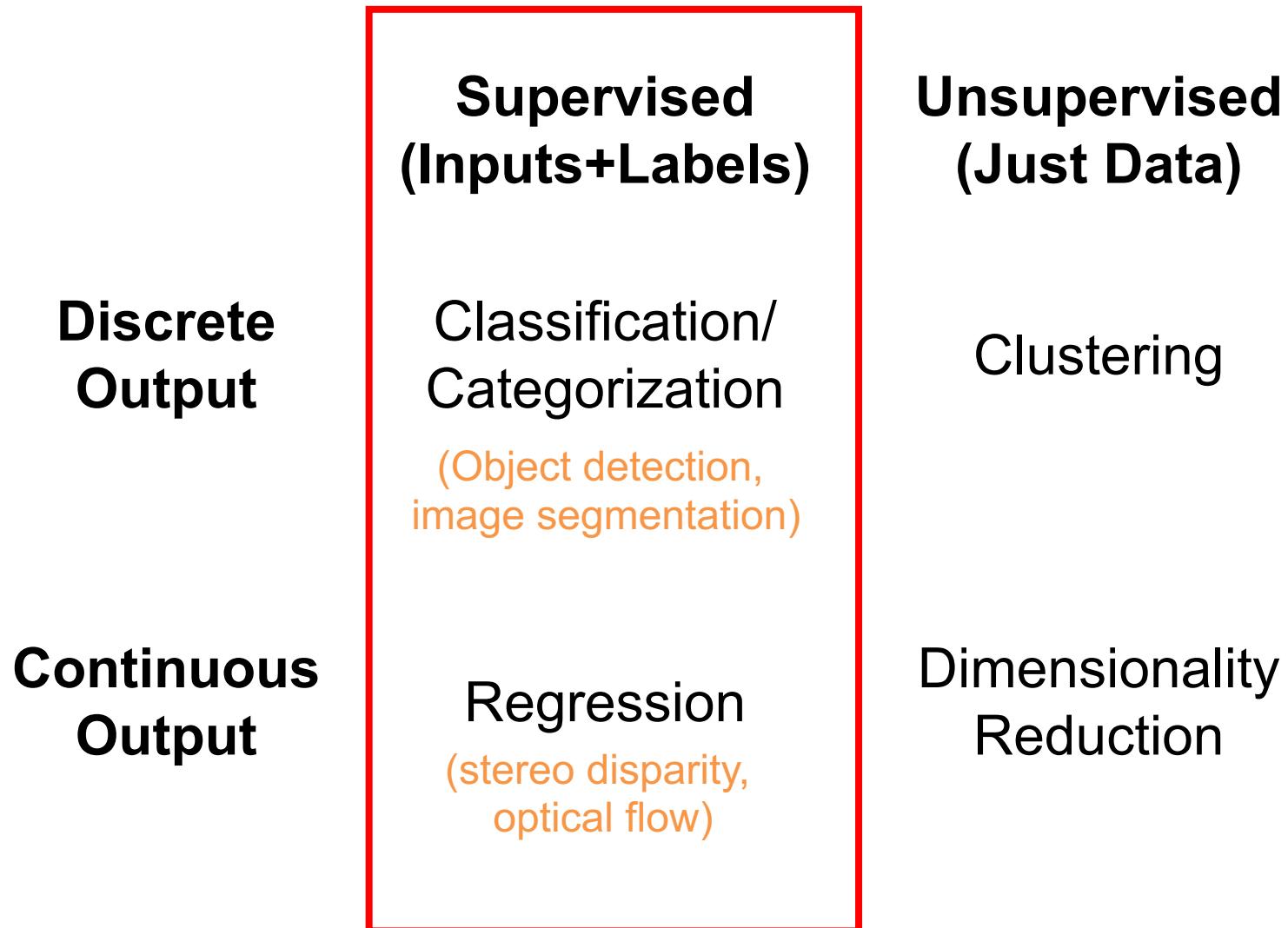
```
def train(images, labels):  
    # Machine learning!  
    return model
```

```
def predict(model, test_images):  
    # Use model to predict labels  
    return test_labels
```

Example training set

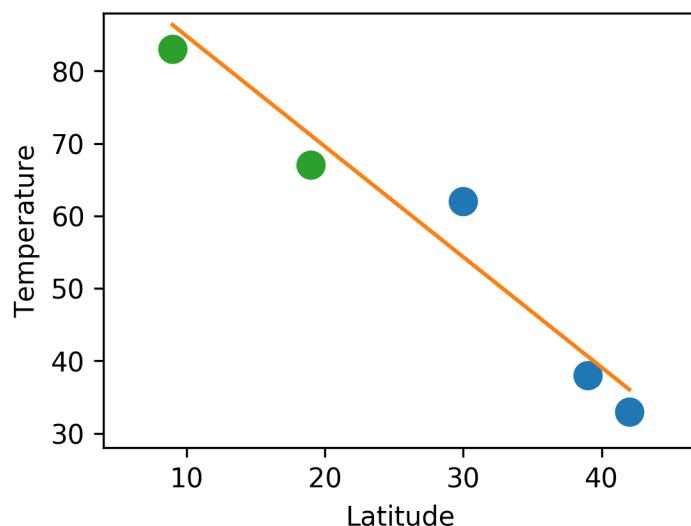


ML Problems in Vision



Regularization

	<u>City</u>	<u>Latitude (°)</u>	<u>Temp (F)</u>	No regularization $(\lambda = 0)$		Regularized $(\lambda = 0.02)$	
				<u>Prediction</u>	<u>Error</u>	<u>Prediction</u>	<u>Error</u>
Train	Ann Arbor	42	33	31.9	1.0	36.0	3.0
	Washington, DC	39	38	39.4	1.4	40.6	2.6
	Austin, TX	30	62	61.7	0.3	54.3	7.7
Test	Mexico City	19	67	88.9	21.9	71.1	4.1
	Panama City	9	83	113.6	30.6	86.4	3.4



L2-Regularized Least Squares
$$\arg \min_w \|y - Xw\|^2 + \lambda \|w\|^2$$

Fit training data Regularization Strength Penalize complexity

Parameters and Hyperparameters

L2-Regularized Least Squares

$$\arg \min_{\mathbf{w}} \|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2 + \lambda \|\mathbf{w}\|^2$$

Parameter (w): **(Automatically)** selected during training by fitting to training data

Hyperparameter (λ): **(Manually)** chosen before training, does not depend on training data

Question: How to choose hyperparameters?

Choosing Hyperparameters

Idea #1: Choose hyperparameters that work best on the data

BAD: $\lambda = 0$ always works best on training data

Your Dataset

Idea #2: Split data into **train** and **test**, choose hyperparameters that work best on test data

BAD: No idea how we will perform on new data

train

test

Idea #3: Split data into **train**, **val**, and **test**; choose hyperparameters on val and evaluate on test

Better
!

train

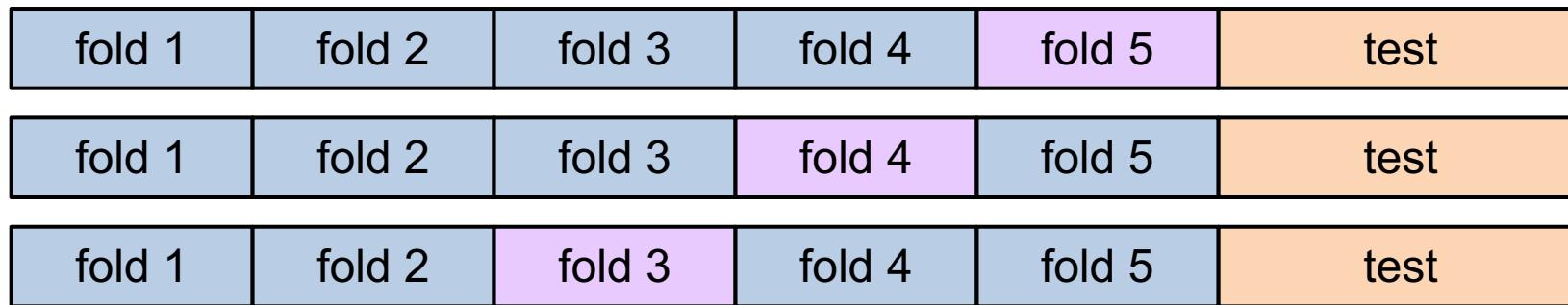
validation

test

Choosing Hyperparameters

Your Dataset

Idea #4: Cross-Validation: Split data into **folds**,
try each fold as validation and average the results



Useful for small datasets, but (unfortunately) not
used too frequently in deep learning

Image Classification: Core Vision Task

Input: image



This image by [Nikita](#) is
licensed under [CC-BY 2.0](#)

Output: Assign image to
one of a fixed set of
categories

cat

bird

deer

dog

truck



Linear Classifiers

$$y_i = \mathbf{w}x_i + b, \quad x_i \in \mathbb{R}^D \text{ is image feature}$$



Model – one weight per class: $\mathbf{w}_0, \mathbf{w}_1, \mathbf{w}_2$

$\mathbf{w}_0^T \mathbf{x}$ big if cat

$\mathbf{w}_1^T \mathbf{x}$ big if dog

$\mathbf{w}_2^T \mathbf{x}$ big if hippo

Stack together: W_{3xD}

Choosing W: Loss Function

A **loss function** tells how good our current classifier is

Low loss = good classifier
High loss = bad classifier

(Also called: **objective function**; **cost function**)

Given a dataset

$$\{(x_i, y_i)\}_{i=1}^N$$

of images x_i and labels y_i ,

Loss for a single example is:
 $L_i(f(x_i, W), y_i)$

Loss for the dataset is

$$L = \frac{1}{N} \sum_{i=1}^N L_i(f(x_i, W), y_i)$$

Cross-Entropy Loss

Want to interpret raw classifier scores as **probabilities**

Classifier scores

$$s = f(x_i, W)$$



cat **3.2**

car 5.1

frog -1.7

Cross-Entropy Loss

Want to interpret raw classifier scores as **probabilities**

Classifier scores
 $s = f(x_i, W)$



Softmax function

$$p_i = \frac{\exp(s_i)}{\sum_j \exp(s_j)}$$

cat **3.2**

car 5.1

frog -1.7

Cross-Entropy Loss

Want to interpret raw classifier scores as **probabilities**

Classifier scores
 $s = f(x_i, W)$



Softmax function

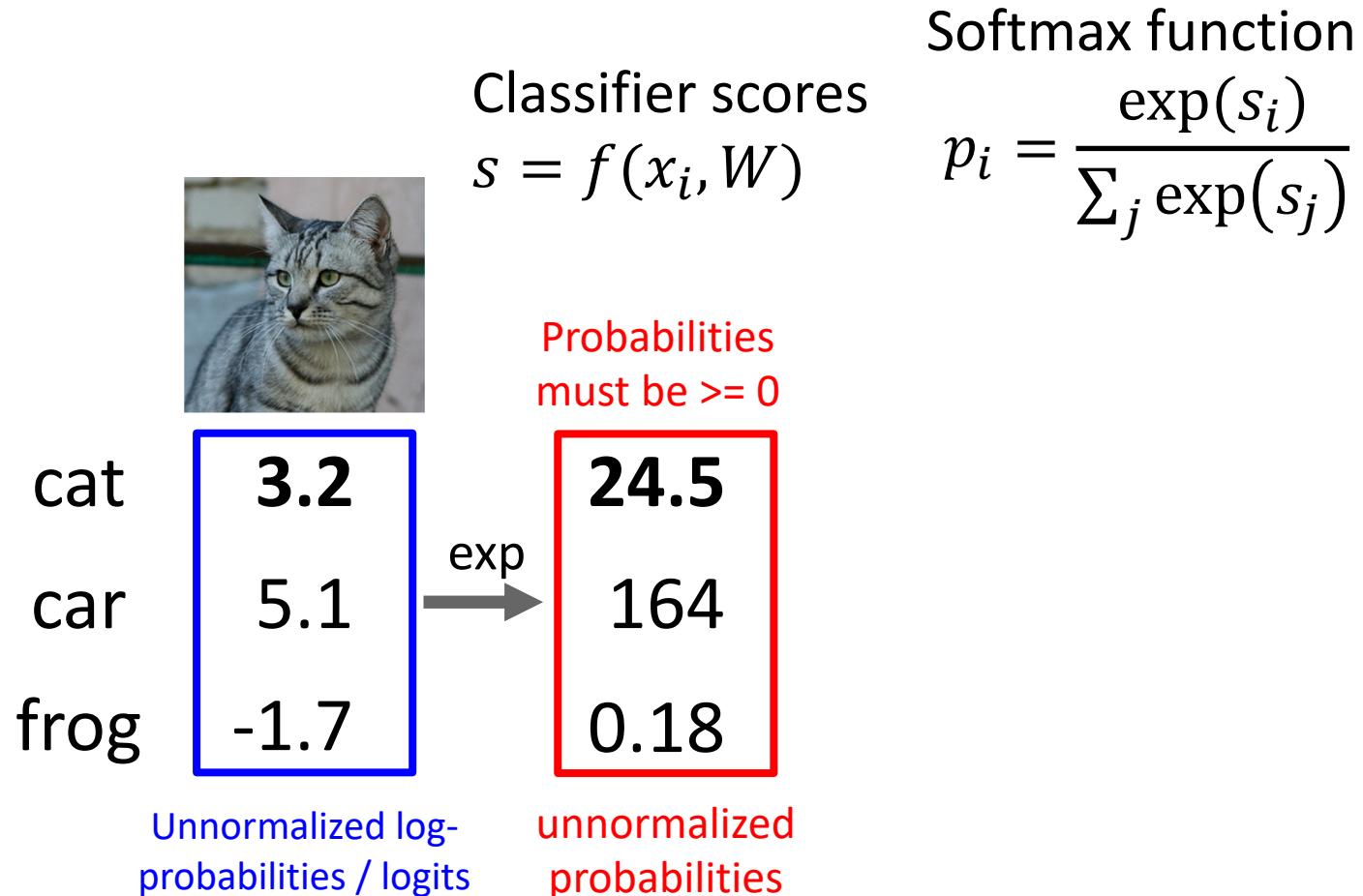
$$p_i = \frac{\exp(s_i)}{\sum_j \exp(s_j)}$$

cat	3.2
car	5.1
frog	-1.7

Unnormalized log-
probabilities / logits

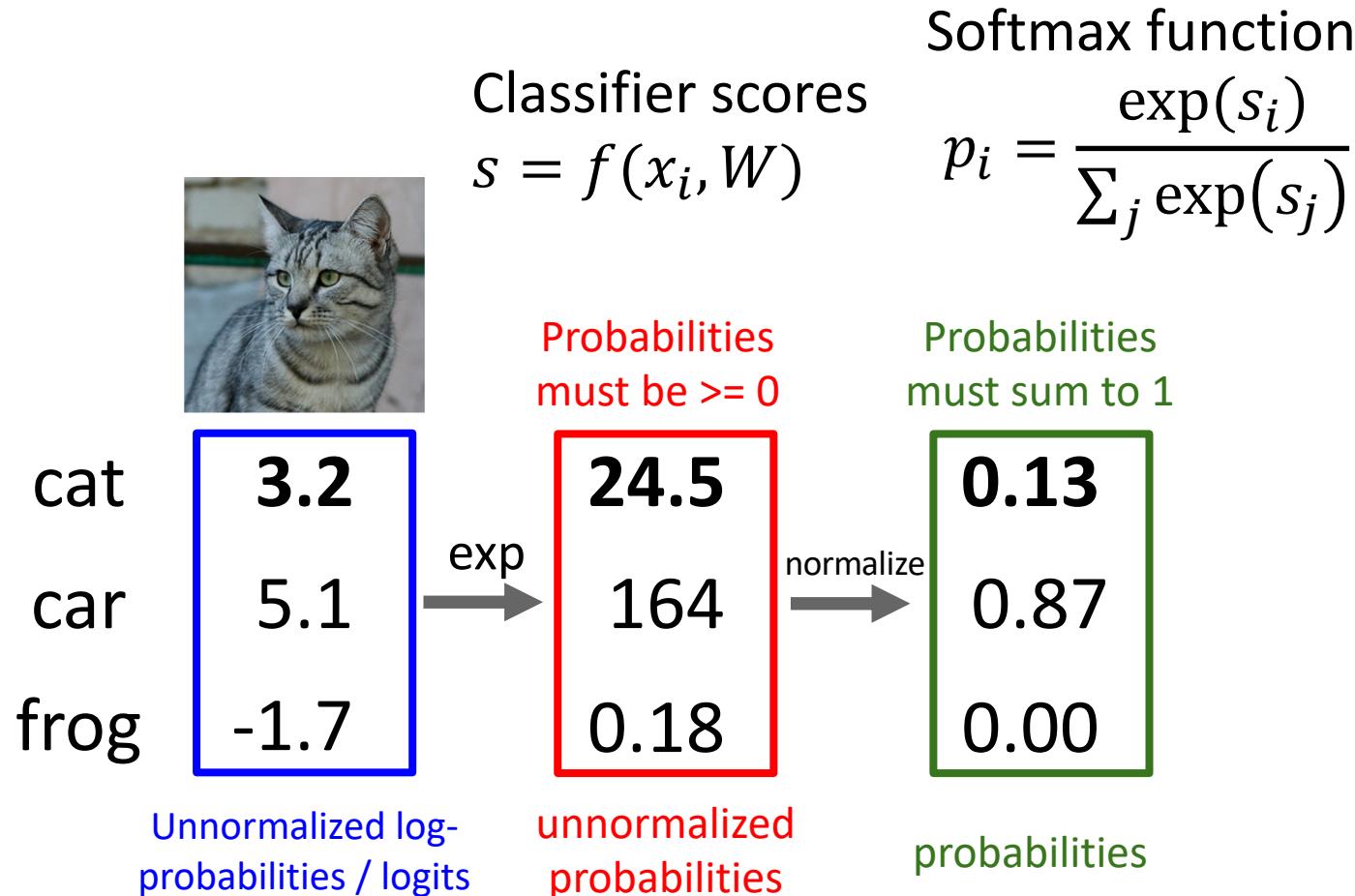
Cross-Entropy Loss

Want to interpret raw classifier scores as **probabilities**



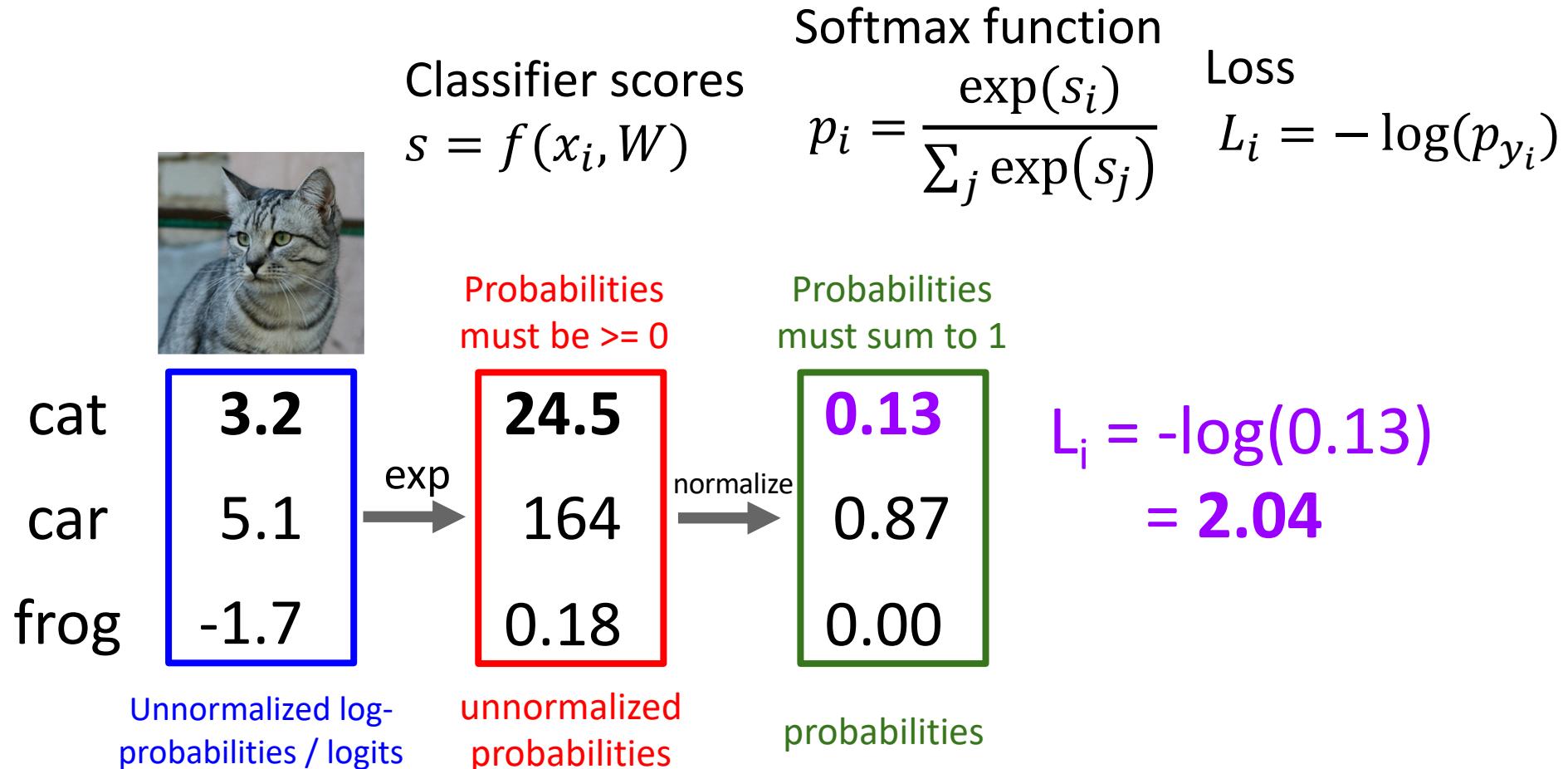
Cross-Entropy Loss

Want to interpret raw classifier scores as **probabilities**



Cross-Entropy Loss

Want to interpret raw classifier scores as **probabilities**



Cross-Entropy Loss

Want to interpret raw classifier scores as **probabilities**

	
cat	3.2
car	5.1
frog	-1.7

Unnormalized log-probabilities / logits

Classifier scores
 $s = f(x_i, W)$

Probabilities must be ≥ 0

24.5
164
0.18

\exp

unnormalized probabilities

Softmax function

$$p_i = \frac{\exp(s_i)}{\sum_j \exp(s_j)}$$

Probabilities must sum to 1

0.13
0.87
0.00

probabilities

Loss

$$L_i = -\log(p_{y_i})$$

$$\begin{aligned} L_i &= -\log(0.13) \\ &= 2.04 \end{aligned}$$

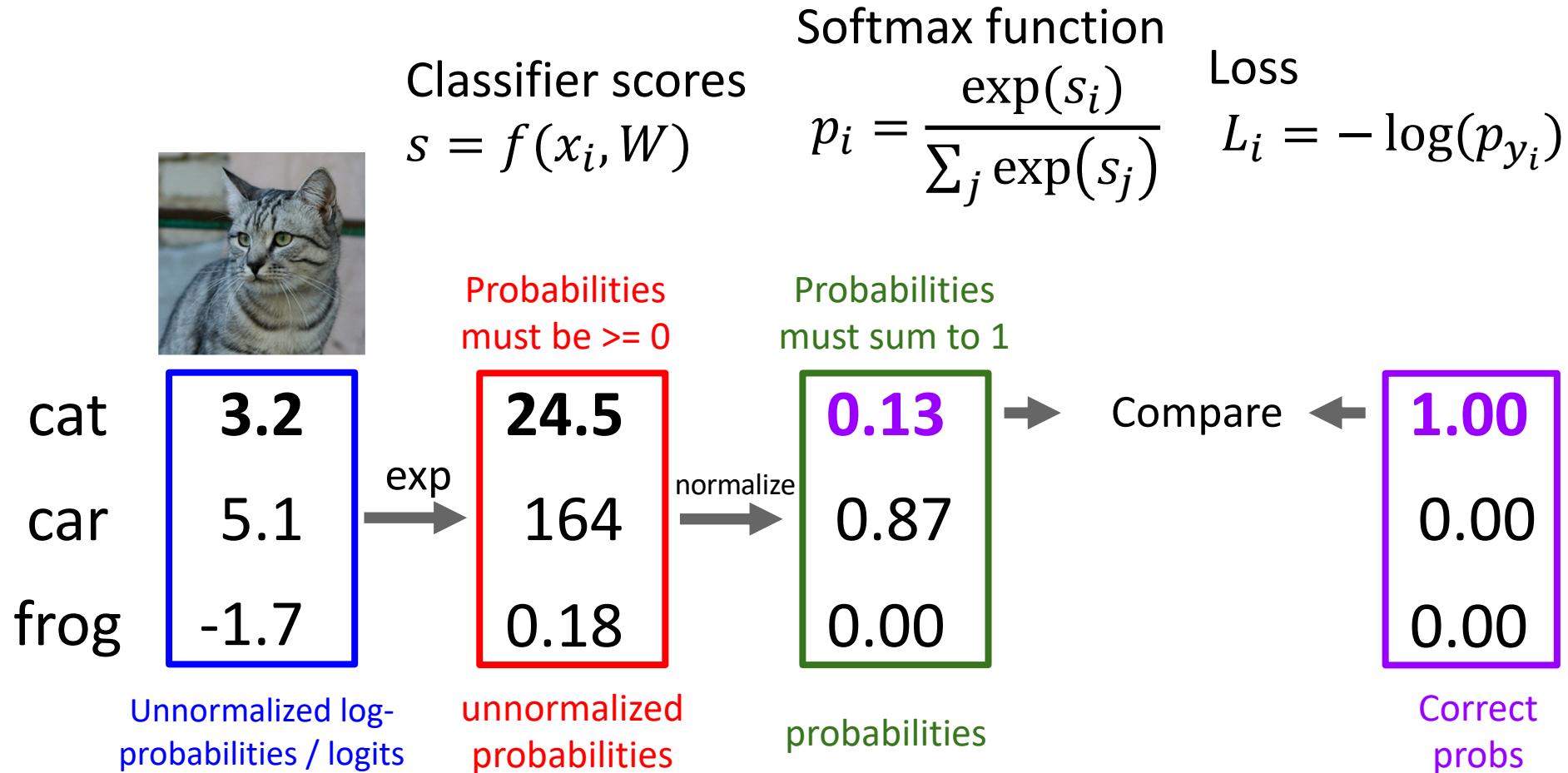
Maximum Likelihood Estimation
Choose weights to maximize the likelihood of the observed data

Today's Class

- Loss Function (quantify how good the model's output are compared to the labels)
 - A bit more about cross-entropy loss
 - SVM (hinge) loss
- Optimization (how to efficiently finding the parameters that minimize the loss function)
 - Gradient descent
 - SGD

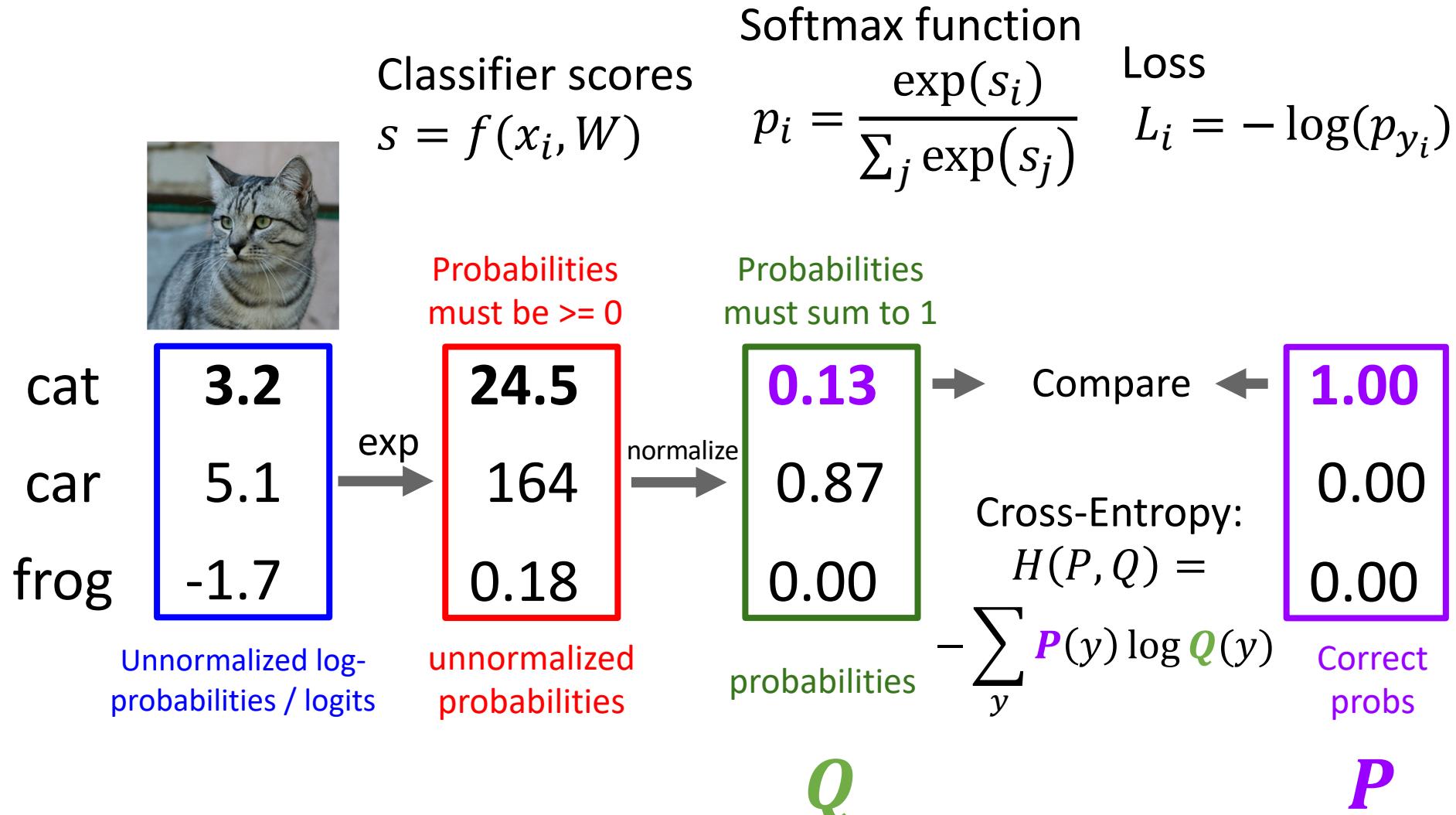
Cross-Entropy Loss

Want to interpret raw classifier scores as **probabilities**



Cross-Entropy Loss

Want to interpret raw classifier scores as **probabilities**



Cross-Entropy Loss

Want to interpret raw classifier scores as **probabilities**



cat	3.2
car	5.1
frog	-1.7

Classifier scores
 $s = f(x_i, W)$

Softmax function
 $p_i = \frac{\exp(s_i)}{\sum_j \exp(s_j)}$

Loss
 $L_i = -\log(p_{y_i})$

Putting it all together:

$$L_i = -\log \frac{\exp(s_{y_i})}{\sum_j \exp(s_j)}$$

Cross-Entropy Loss

Want to interpret raw classifier scores as **probabilities**



cat	3.2
car	5.1
frog	-1.7

Classifier scores
 $s = f(x_i, W)$

Softmax function
 $p_i = \frac{\exp(s_i)}{\sum_j \exp(s_j)}$

Loss
 $L_i = -\log(p_{y_i})$

Putting it all together:

$$L_i = -\log \frac{\exp(s_{y_i})}{\sum_j \exp(s_j)}$$

Q: What is the min /
max possible loss L_i ?

Cross-Entropy Loss

Want to interpret raw classifier scores as **probabilities**



cat	3.2
car	5.1
frog	-1.7

Classifier scores
 $s = f(x_i, W)$

Softmax function
 $p_i = \frac{\exp(s_i)}{\sum_j \exp(s_j)}$

Loss
 $L_i = -\log(p_{y_i})$

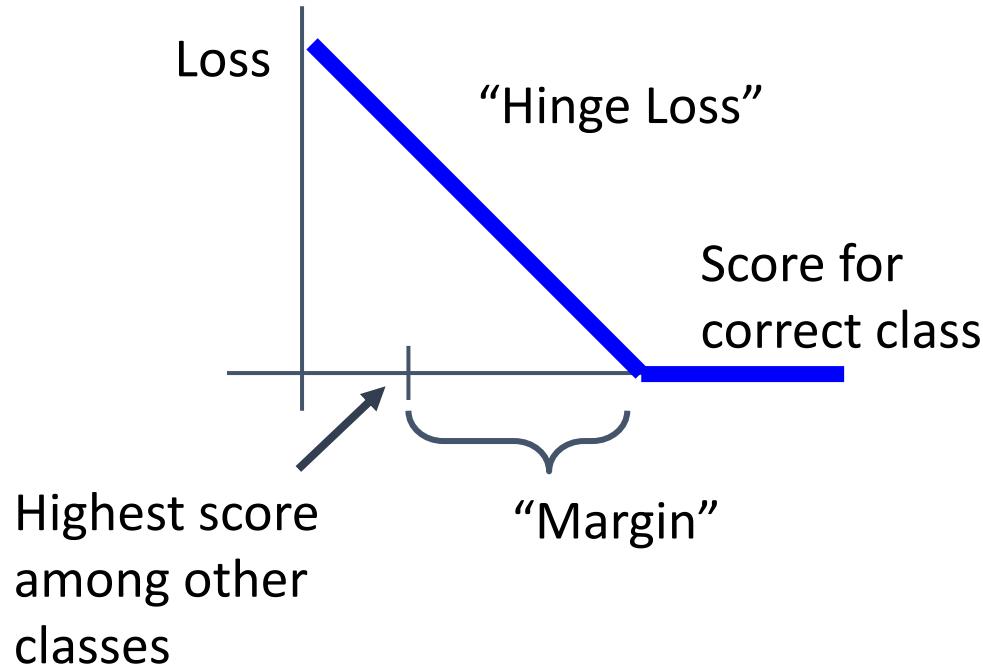
Putting it all together:

$$L_i = -\log \frac{\exp(s_{y_i})}{\sum_j \exp(s_j)}$$

Q: If all scores are small random values, what is the loss?

Multiclass SVM Loss

"The score of the correct class should be higher than all the other scores"



Given an example (x_i, y_i)
(x_i is image, y_i is label)

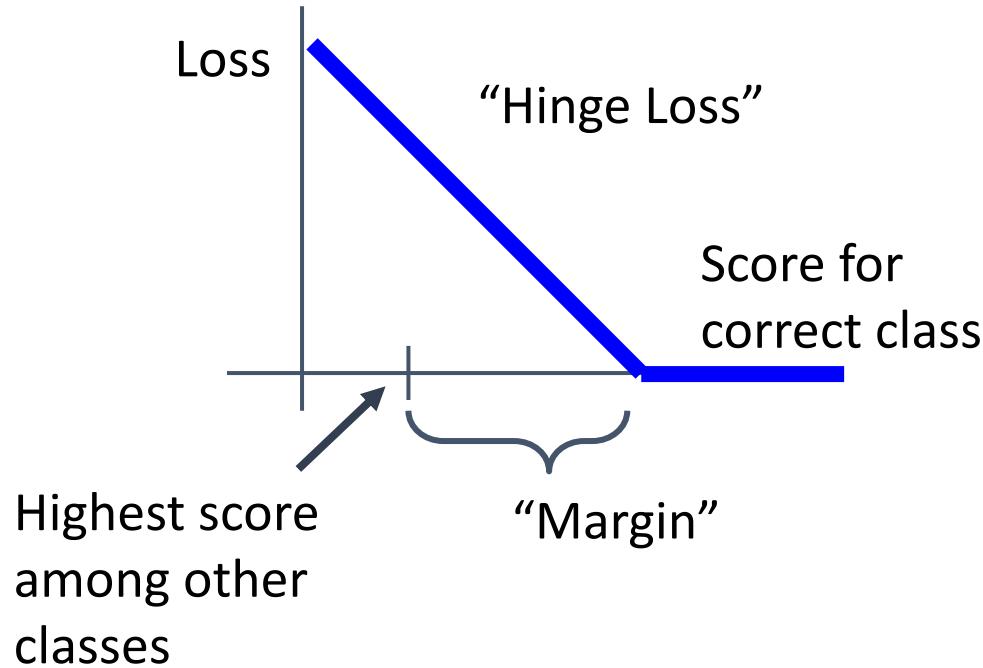
Let $s = f(x_i, W)$ be scores

Then the SVM loss has the form:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

Multiclass SVM Loss

"The score of the correct class should be higher than all the other scores"



Given an example (x_i, y_i)
(x_i is image, y_i is label)

Let $s = f(x_i, W)$ be scores

Then the SVM loss has the form:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

margin

Multiclass SVM Loss



cat	3.2	1.3	2.2
car	5.1	4.9	2.5
frog	-1.7	2.0	-3.1

Given an example (x_i, y_i)
 $(x_i$ is image, y_i is label)

Let $s = f(x_i, W)$ be scores

Then the SVM loss has the form:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

Multiclass SVM Loss



cat	3.2	1.3	2.2
car	5.1	4.9	2.5
frog	-1.7	2.0	-3.1

Given an example (x_i, y_i)
 $(x_i$ is image, y_i is label)

Let $s = f(x_i, W)$ be scores

Then the SVM loss has the form:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

Multiclass SVM Loss



cat	3.2	1.3	2.2
car	5.1	4.9	2.5
frog	-1.7	2.0	-3.1
Loss	2.9		

Given an example (x_i, y_i)
 $(x_i$ is image, y_i is label)

Let $s = f(x_i, W)$ be scores

Then the SVM loss has the form:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

$$= \max(0, 5.1 - 3.2 + 1)$$

$$+ \max(0, -1.7 - 3.2 + 1)$$

$$= \max(0, 2.9) + \max(0, -3.9)$$

$$= 2.9 + 0$$

$$= 2.9$$

Multiclass SVM Loss



cat	3.2	1.3	2.2
car	5.1	4.9	2.5
frog	-1.7	2.0	-3.1
Loss	2.9	0	

Given an example (x_i, y_i)
 $(x_i$ is image, y_i is label)

Let $s = f(x_i, W)$ be scores

Then the SVM loss has the form:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

$$\begin{aligned} &= \max(0, 1.3 - 4.9 + 1) \\ &\quad + \max(0, 2.0 - 4.9 + 1) \\ &= \max(0, -2.6) + \max(0, -1.9) \\ &= 0 + 0 \\ &= 0 \end{aligned}$$

Multiclass SVM Loss



cat	3.2	1.3	2.2
car	5.1	4.9	2.5
frog	-1.7	2.0	-3.1
Loss	2.9	0	12.9

Given an example (x_i, y_i)
 $(x_i$ is image, y_i is label)

Let $s = f(x_i, W)$ be scores

Then the SVM loss has the form:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

$$\begin{aligned} &= \max(0, 2.2 - (-3.1) + 1) \\ &\quad + \max(0, 2.5 - (-3.1) + 1) \\ &= \max(0, 6.3) + \max(0, 6.6) \\ &= 6.3 + 6.6 \\ &= 12.9 \end{aligned}$$

Multiclass SVM Loss



cat	3.2	1.3	2.2
car	5.1	4.9	2.5
frog	-1.7	2.0	-3.1
Loss	2.9	0	12.9

Given an example (x_i, y_i)
 $(x_i$ is image, y_i is label)

Let $s = f(x_i, W)$ be scores

Then the SVM loss has the form:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

Loss over the dataset is:

$$\begin{aligned} L &= (2.9 + 0.0 + 12.9) / 3 \\ &= 5.27 \end{aligned}$$

Multiclass SVM Loss



cat	3.2	1.3	2.2
car	5.1	4.9	2.5
frog	-1.7	2.0	-3.1
Loss	2.9	0	12.9

Given an example (x_i, y_i)
 $(x_i$ is image, y_i is label)

Let $s = f(x_i, W)$ be scores

Then the SVM loss has the form:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

Q: What happens to the loss if the scores for the car image change a bit?

Multiclass SVM Loss



cat	3.2	1.3	2.2
car	5.1	4.9	2.5
frog	-1.7	2.0	-3.1
Loss	2.9	0	12.9

Given an example (x_i, y_i)
 $(x_i$ is image, y_i is label)

Let $s = f(x_i, W)$ be scores

Then the SVM loss has the form:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

Q: What are the min
and max possible loss?

Multiclass SVM Loss



cat	3.2	1.3	2.2
car	5.1	4.9	2.5
frog	-1.7	2.0	-3.1
Loss	2.9	0	12.9

Given an example (x_i, y_i)
 $(x_i$ is image, y_i is label)

Let $s = f(x_i, W)$ be scores

Then the SVM loss has the form:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

Q: At initialization W is small so all $s \approx 0$. What is the loss?

Multiclass SVM Loss



cat	3.2	1.3	2.2
car	5.1	4.9	2.5
frog	-1.7	2.0	-3.1
Loss	2.9	0	12.9

Given an example (x_i, y_i)
 $(x_i$ is image, y_i is label)

Let $s = f(x_i, W)$ be scores

Then the SVM loss has the form:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

Q: What would happen if sum were over all classes?
(including $j = y_i$)

Multiclass SVM Loss



cat	3.2	1.3	2.2
car	5.1	4.9	2.5
frog	-1.7	2.0	-3.1
Loss	2.9	0	12.9

Given an example (x_i, y_i)
 $(x_i$ is image, y_i is label)

Let $s = f(x_i, W)$ be scores

Then the SVM loss has the form:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

Q: What if the loss used mean instead of sum?

Multiclass SVM Loss



cat	3.2	1.3	2.2
car	5.1	4.9	2.5
frog	-1.7	2.0	-3.1
Loss	2.9	0	12.9

Given an example (x_i, y_i)
 $(x_i$ is image, y_i is label)

Let $s = f(x_i, W)$ be scores

Then the SVM loss has the form:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

Q: What if we used this loss instead?

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)^2$$

Cross-Entropy vs SVM Loss

$$L_i = -\log \frac{\exp(s_{y_i})}{\sum_j \exp(s_j)}$$

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

Assume scores:

[10, -2, 3]

[10, 9, 9]

[10, -100, -100]

and $y_i = 0$

Q: What is cross-entropy loss? What is SVM loss?

A: Cross-entropy loss > 0
SVM loss = 0

Cross-Entropy vs SVM Loss

$$L_i = -\log \frac{\exp(s_{y_i})}{\sum_j \exp(s_j)}$$

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

Assume scores:

[10, -2, 3]

[10, 9, 9]

[10, -100, -100]

and $y_i = 0$

Q: What happens to each loss if I slightly change the scores of the last datapoint?

A: Cross-entropy loss will change; SVM loss will stay the same

Cross-Entropy vs SVM Loss

$$L_i = -\log \frac{\exp(s_{y_i})}{\sum_j \exp(s_j)}$$

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

Assume scores:

[20, -2, 3]

[20, 9, 9]

[20, -100, -100]

and $y_i = 0$

Q: What happens to each loss if I double the score of the correct class from 10 to 20?

A: Cross-entropy loss will decrease, SVM loss still 0

There is a “bug” with the hinge loss:

$$f(x, W) = Wx$$

$$L = \frac{1}{N} \sum_{i=1}^N \sum_{j \neq y_i} \max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + 1)$$



E.g. Suppose that we found a W such that $L = 0$.
Is this W unique?

Suppose: 3 training examples, 3 classes.

With some W the scores $f(x, W) = Wx$ are:



cat	3.2	1.3	2.2
car	5.1	4.9	2.5
frog	-1.7	2.0	-3.1
Losses:	2.9	0	12.9

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

Before:

$$\begin{aligned} &= \max(0, 1.3 - 4.9 + 1) \\ &\quad + \max(0, 2.0 - 4.9 + 1) \\ &= \max(0, -2.6) + \max(0, -1.9) \\ &= 0 + 0 \\ &= 0 \end{aligned}$$

With W twice as large:

$$\begin{aligned} &= \max(0, 2.6 - 9.8 + 1) \\ &\quad + \max(0, 4.0 - 9.8 + 1) \\ &= \max(0, -6.2) + \max(0, -4.8) \\ &= 0 + 0 \\ &= 0 \end{aligned}$$

$$f(x, W) = Wx$$



An example:

What is the loss? (POLL)

cat 1.3

car 2.5

frog 2.0

Loss:

$$f(x, W) = Wx$$



An example:
What is the loss?

cat	1.3
car	2.5
frog	2.0
Loss:	0.5

$$f(x, W) = Wx$$



An example:

What is the loss?

How could we change W to eliminate
the loss? (POLL)

cat	1.3
car	2.5
frog	2.0
Loss:	0.5

$$f(x, W) = Wx$$



cat	1.3	2.6
car	2.5	5.0
frog	2.0	4.0
Loss:	0.5	0

An example:

What is the loss?

How could we change W to eliminate
the loss? (POLL)

Multiply W (and b) by 2!

$$f(x, W) = Wx$$



cat	1.3	2.6
car	2.5	5.0
frog	2.0	4.0
Loss:	0.5	0

An example:

What is the loss?

How could we change W to eliminate
the loss? (POLL)

Multiply W (and b) by 2!

Wait a minute! Have we done anything
useful???

$$f(x, W) = Wx$$

An example:

What is the loss?



cat	1.3	2.6
car	2.5	5.0
frog	2.0	4.0
Loss:	0.5	0

How could we change W to eliminate the loss? (POLL)

Multiply W (and b) by 2!

Wait a minute! Have we done anything useful???

No! Any example that used to be wrong is still wrong (on the wrong side of the boundary). Any example that is right is still right (on the correct side of the boundary).

Regularization

λ = regularization strength
(hyperparameter)

$$L(W) = \underbrace{\frac{1}{N} \sum_{i=1}^N L_i(f(x_i, W), y_i)}_{\text{Data loss}} + \lambda R(W)$$

Data loss: Model predictions should match training data

Regularization: Prevent the model from having too much flexibility.

Simple examples

L2 regularization: $R(W) = \sum_k \sum_l W_{k,l}^2$

L1 regularization: $R(W) = \sum_k \sum_l |W_{k,l}|$

Elastic net (L1 + L2): $R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$

Optimization

Goal: find the \mathbf{w} minimizing
some loss function L .

$$\arg \min_{\mathbf{w} \in R^D} L(\mathbf{w})$$

Works for lots of
different Ls:

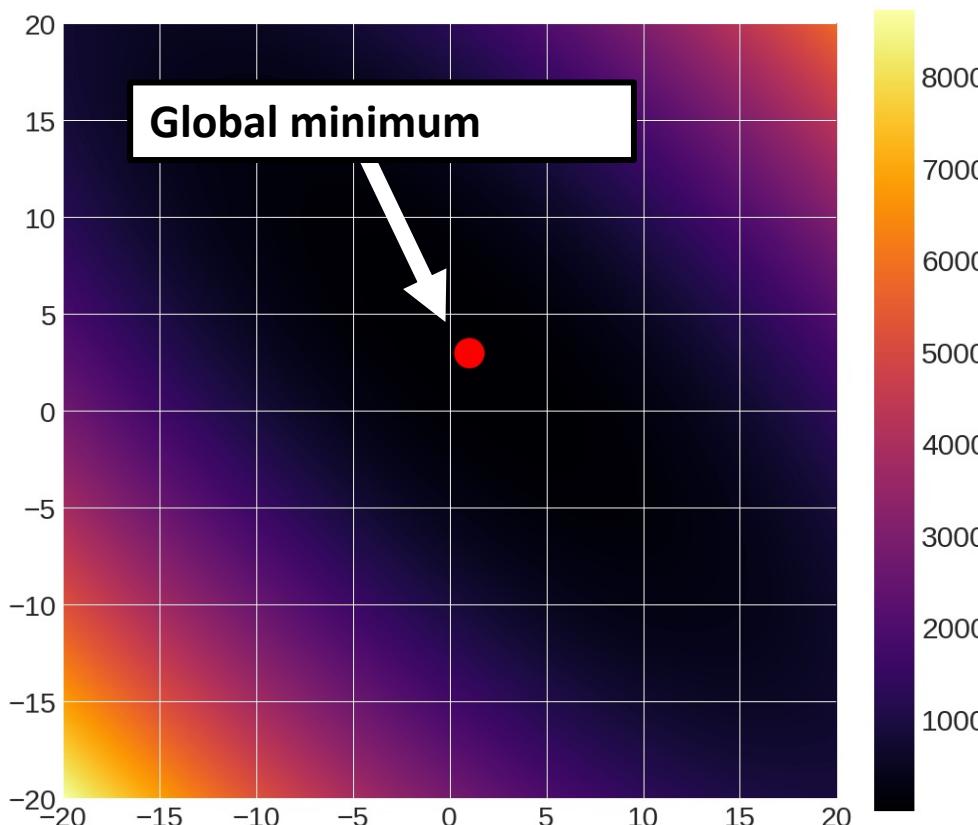
$$L(\mathbf{W}) = \lambda \|\mathbf{W}\|_2^2 + \sum_{i=1}^n -\log\left(\frac{\exp((\mathbf{W}\mathbf{x})_{y_i})}{\sum_k \exp((\mathbf{W}\mathbf{x})_k)}\right)$$

$$L(\mathbf{w}) = C \|\mathbf{w}\|_2^2 + \sum_{i=1}^n \max(0, 1 - y_i \mathbf{w}^T \mathbf{x}_i)$$

$$L(\mathbf{w}) = \lambda \|\mathbf{w}\|_2^2 + \sum_{i=1}^n (y_i - \mathbf{w}^T \mathbf{x}_i)^2$$

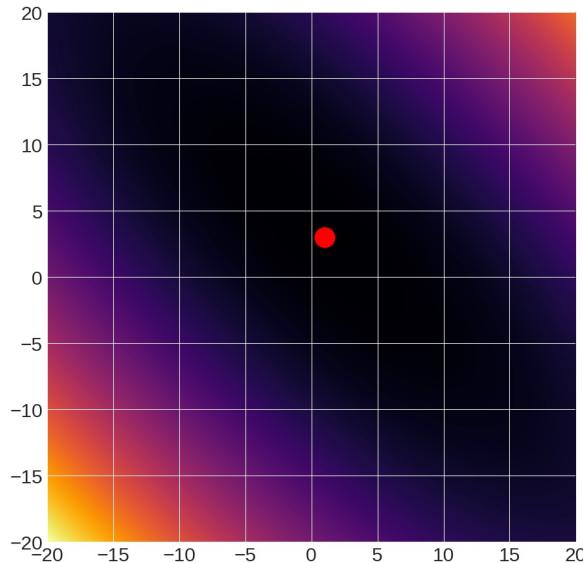
Sample Function to Optimize

$$f(x,y) = (x+2y-7)^2 + (2x+y-5)^2$$



Warning: This is
2D, intuition may
not generalize to
high dimension

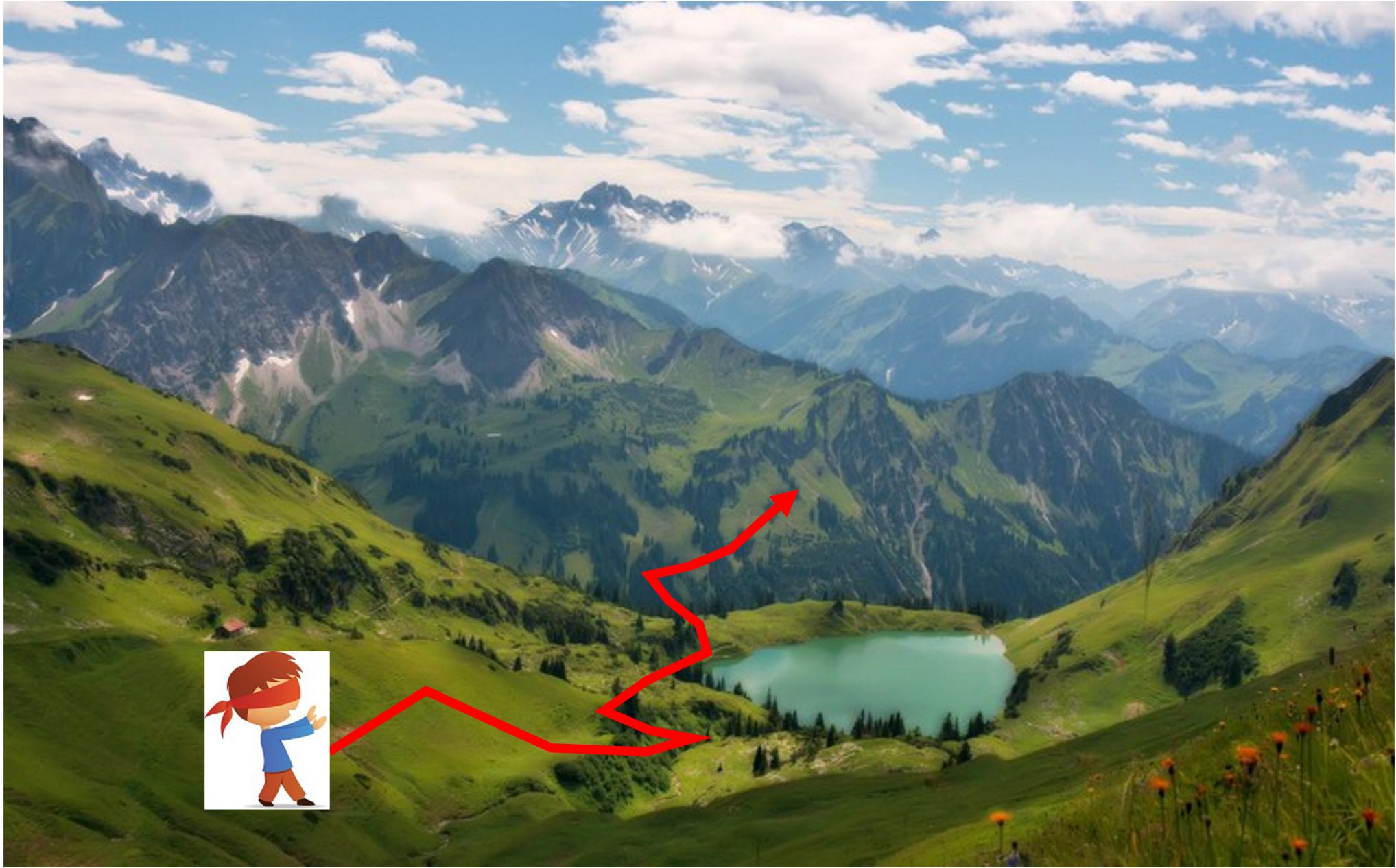
Optimization: A Caveat



- Each point in the picture is a function evaluation
- Here it takes microseconds – so we can easily see the answer
- Functions we want to optimize may take hours to evaluate



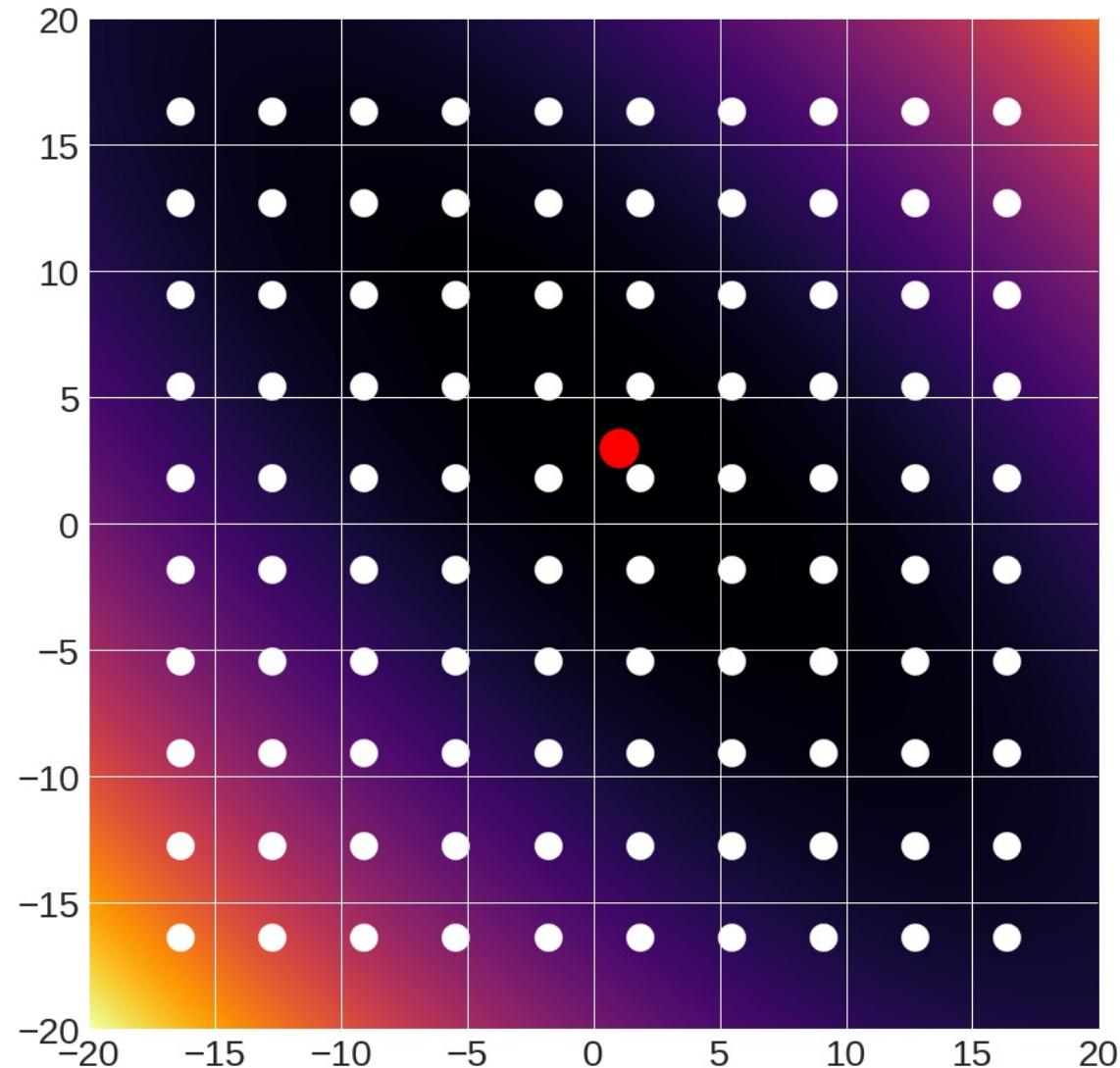
Slide credit: E Learned-Miller, originally developed by A Karpathy, J Johnson, Fei-Fei L



Idea #1A: Grid Search

```
#systematically try things
best, bestScore = None, Inf
for dim1Value in dim1Values:
    ....
    for dimNValue in dimNValues:
        w = [dim1Value, ..., dimNValue]
        if L(w) < bestScore:
            best, bestScore = w, L(w)
return best
```

Idea #1A: Grid Search



Idea #1A: Grid Search

Pros:

1. Super simple
2. Only requires being able to evaluate model

Cons:

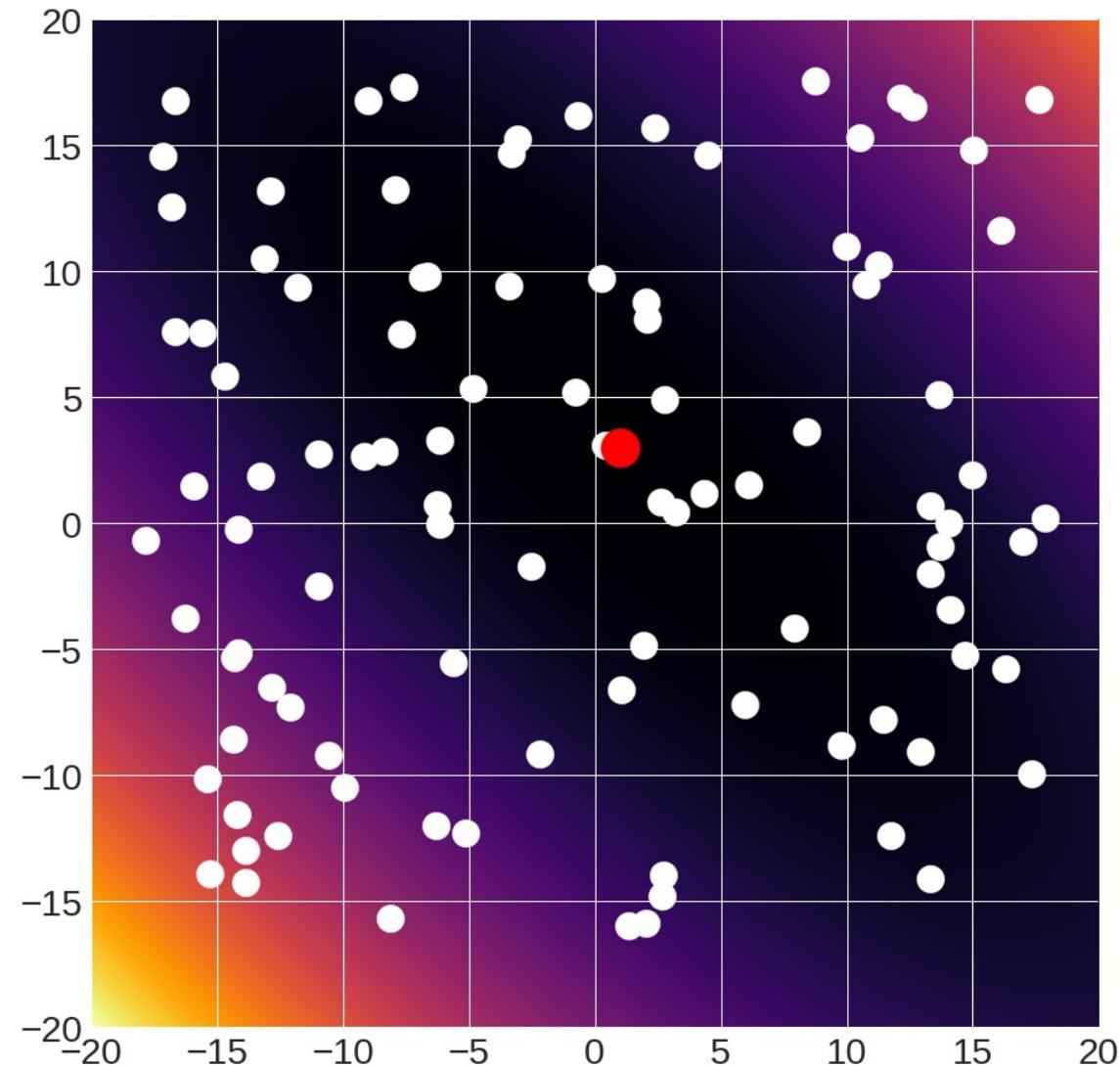
1. Scales horribly to high dimensional spaces

Complexity: $\text{samplesPerDim}^{\text{numberOfDims}}$

Option #1B: Random Search

```
#Do random stuff RANSAC Style
best, bestScore = None, Inf
for iter in range(numIters):
    w = random(N,1) #sample
    score = L(w) #evaluate
    if score < bestScore:
        best, bestScore = w, score
return best
```

Option #1B: Random Search



Option #1B: Random Search

Pros:

1. Super simple
2. Only requires being able to sample model and evaluate it

Cons:

1. Slow – throwing darts at high dimensional dart board
2. Might miss something

$$P(\text{all correct}) =$$

$$\varepsilon^N$$

Good parameters



$$\varepsilon$$

All parameters



$$0$$

$$1$$

When To Use Options 1A / 1B?

Use these when

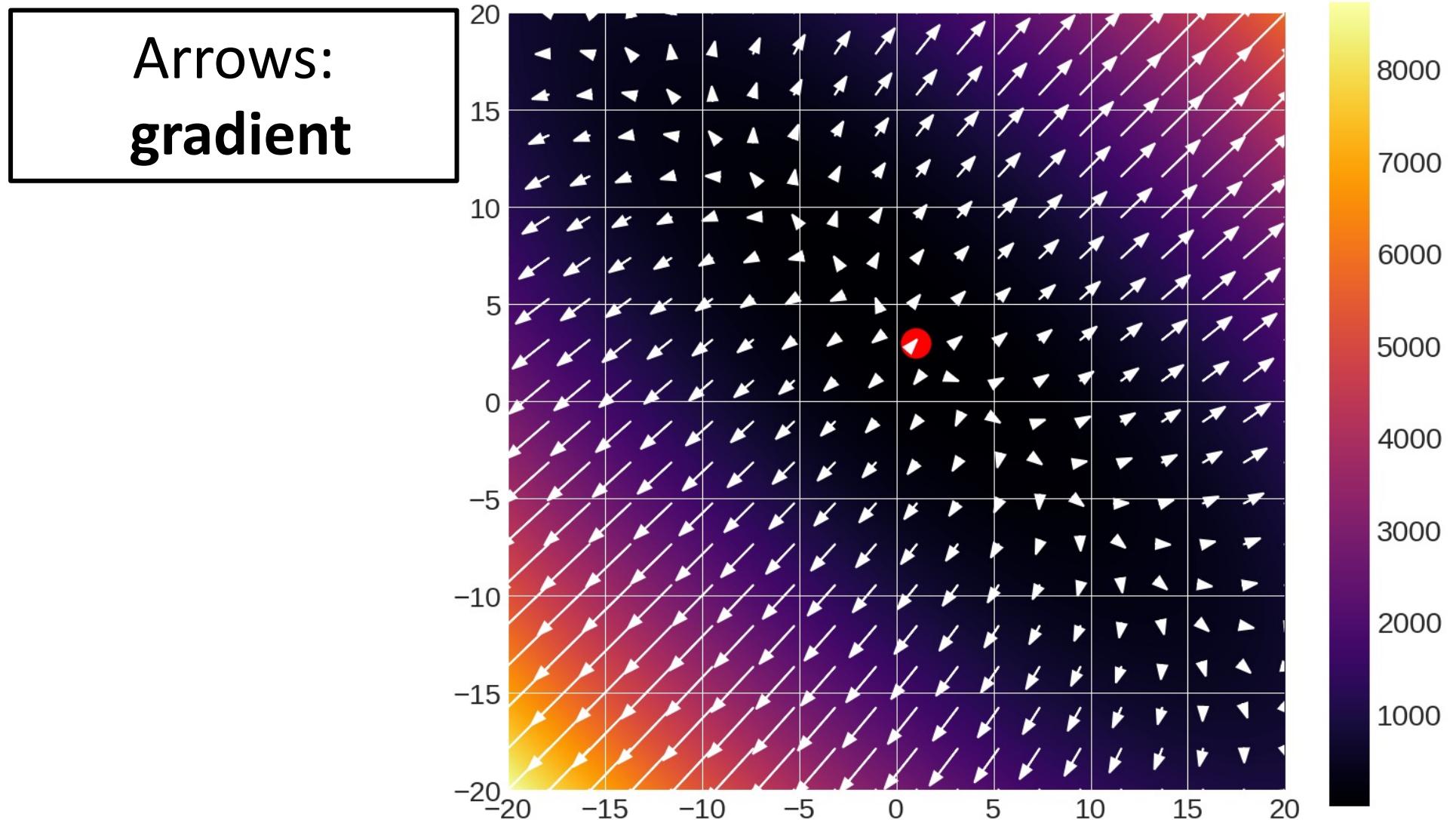
- Number of dimensions small, space bounded
- Objective is impossible to analyze

Random search is arguably more effective; grid search makes it easy to systematically test something (people love certainty)

Idea #2: Follow the slope

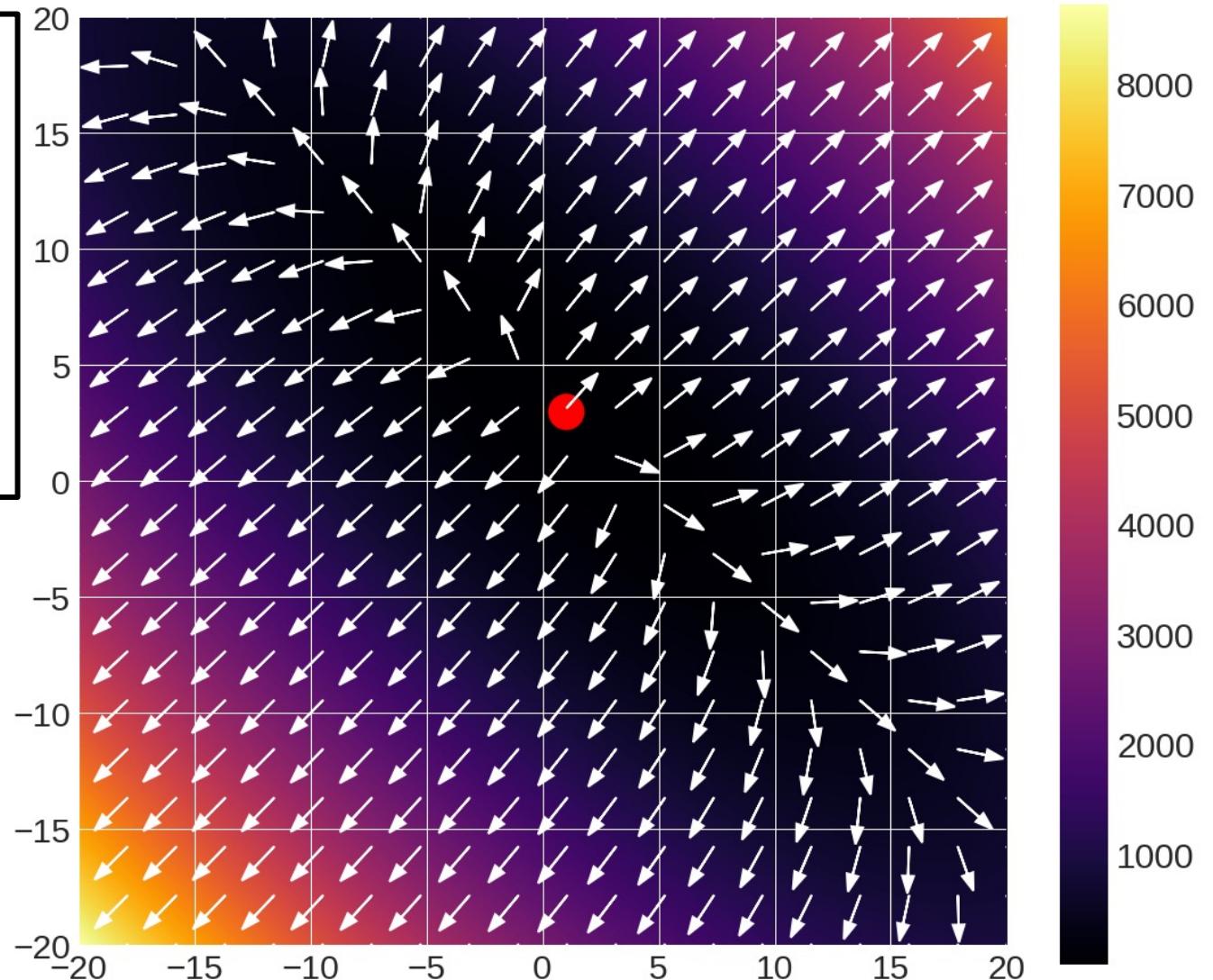


Idea #2: Follow the slope



Idea #2: Follow the slope

Arrows:
**gradient
direction**
(scaled to unit
length)



Idea #2: Follow the slope

Want: $\arg \min_{\mathbf{w}} L(\mathbf{w})$

What's the geometric interpretation of:

$$\nabla_{\mathbf{w}} L(\mathbf{w}) = \begin{bmatrix} \partial L / \partial x_1 \\ \vdots \\ \partial L / \partial x_N \end{bmatrix}$$

Which is bigger (for small α)?

$$L(\mathbf{w}) \stackrel{\leq ?}{>} L(\mathbf{w} + \alpha \nabla_{\mathbf{w}} L(\mathbf{w}))$$

Idea #2: Follow the slope

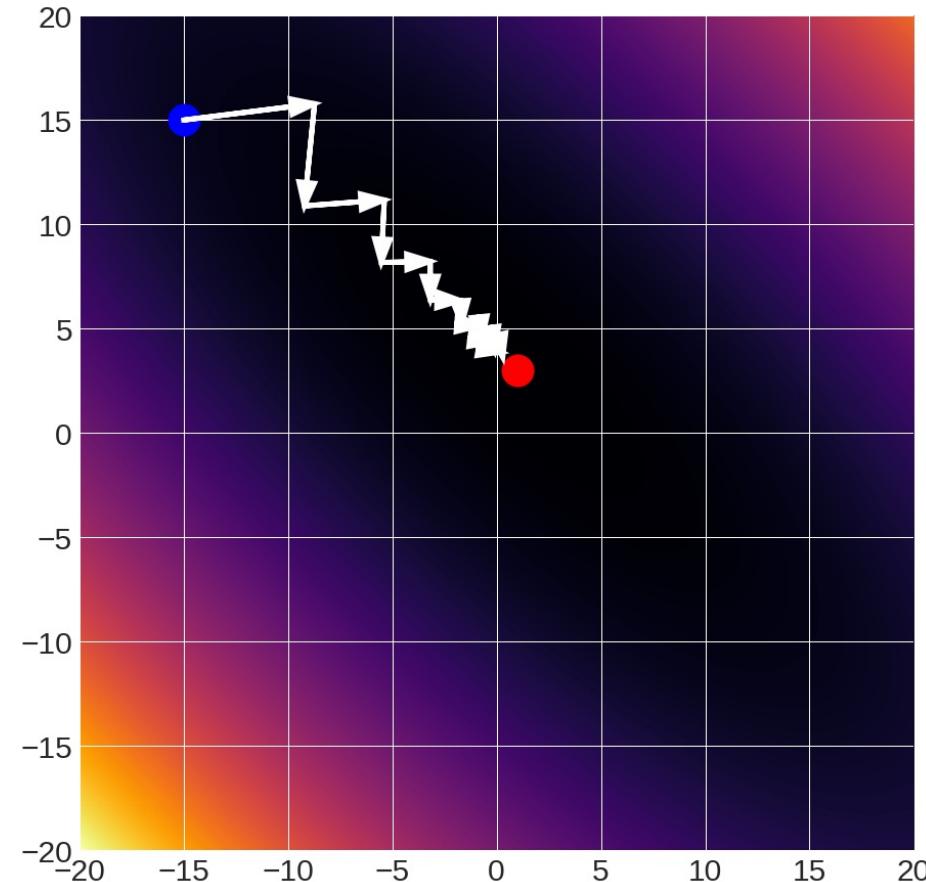
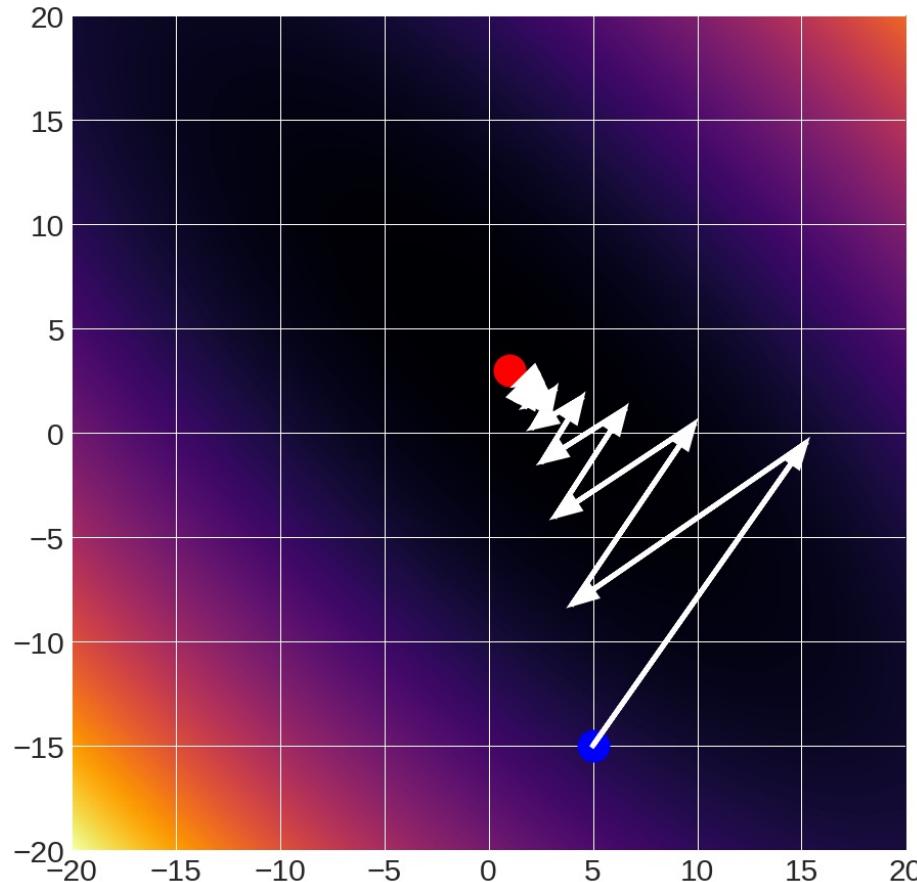
Method: at each step, move in direction of negative gradient

```
w0 = initialize() #initialize  
for iter in range(numIters):  
    g = ∇_w L(w)          # eval gradient  
    w = w + -stepsize*g   # update w  
return w
```

Gradient Descent

Given starting point (blue)

$$w_{i+1} = w_i + -9.8 \times 10^{-2} * \text{gradient}$$



Computing Gradients: Numeric

How Do You Compute The Gradient?
Numerical Method:

$$\nabla_w L(w) = \begin{bmatrix} \frac{\partial L(w)}{\partial w_1} \\ \vdots \\ \frac{\partial L(w)}{\partial w_n} \end{bmatrix}$$

How do you compute this?

$$\frac{\partial f(x)}{\partial x} = \lim_{\epsilon \rightarrow 0} \frac{f(x + \epsilon) - f(x)}{\epsilon}$$

In practice, use:

$$\frac{f(x + \epsilon) - f(x - \epsilon)}{2\epsilon}$$

Computing Gradients: Numeric

How Do You Compute The Gradient?
Numerical Method:

$$\nabla_w L(w) = \begin{bmatrix} \frac{\partial L(w)}{\partial x_1} \\ \vdots \\ \frac{\partial L(w)}{\partial x_n} \end{bmatrix}$$

Use: $\frac{f(x + \epsilon) - f(x - \epsilon)}{2\epsilon}$

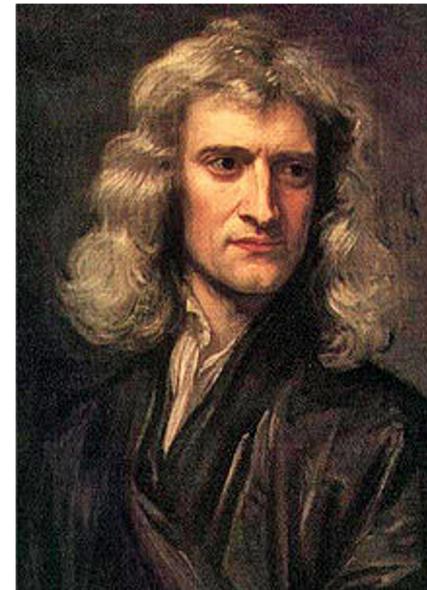
How many function evaluations per dimension?

Computing Gradients: Analytic

How Do You Compute The Gradient?

Better Idea: Use Calculus!

$$\nabla_w L(w) = \begin{bmatrix} \frac{\partial L(w)}{\partial x_1} \\ \vdots \\ \frac{\partial L(w)}{\partial x_n} \end{bmatrix}$$



[This image](#) is in the public domain



[This image](#) is in the public domain

During a pandemic, Isaac Newton had to work from home, too. He used the time wisely.



A later portrait of Sir Isaac Newton by Samuel Freeman. (British Library/National Endowment for the Humanities)

By **Gillian Brockell**

March 12, 2020 at 2:18 p.m. EDT

Isaac Newton was in his early 20s when the Great Plague of London hit. He wasn't a "Sir" yet, didn't

1. Developed calculus
2. Fundamentals of optics
3. Theory of gravity

Computing Gradients

- **Numeric gradient:** approximate, slow, easy to write
- **Analytic gradient:** exact, fast, error-prone

In practice: Always use analytic gradient, but check implementation with numerical gradient. This is called a **gradient check**.

```
torch.autograd.gradcheck(func, inputs, eps=1e-06, atol=1e-05, rtol=0.001,  
raise_exception=True, check_sparse_nnz=False, nondet_tol=0.0)
```

[SOURCE] ↗

Check gradients computed via small finite differences against analytical gradients w.r.t. tensors in `inputs` that are of floating point type and with `requires_grad=True`.

The check between numerical and analytical gradients uses `allclose()`.

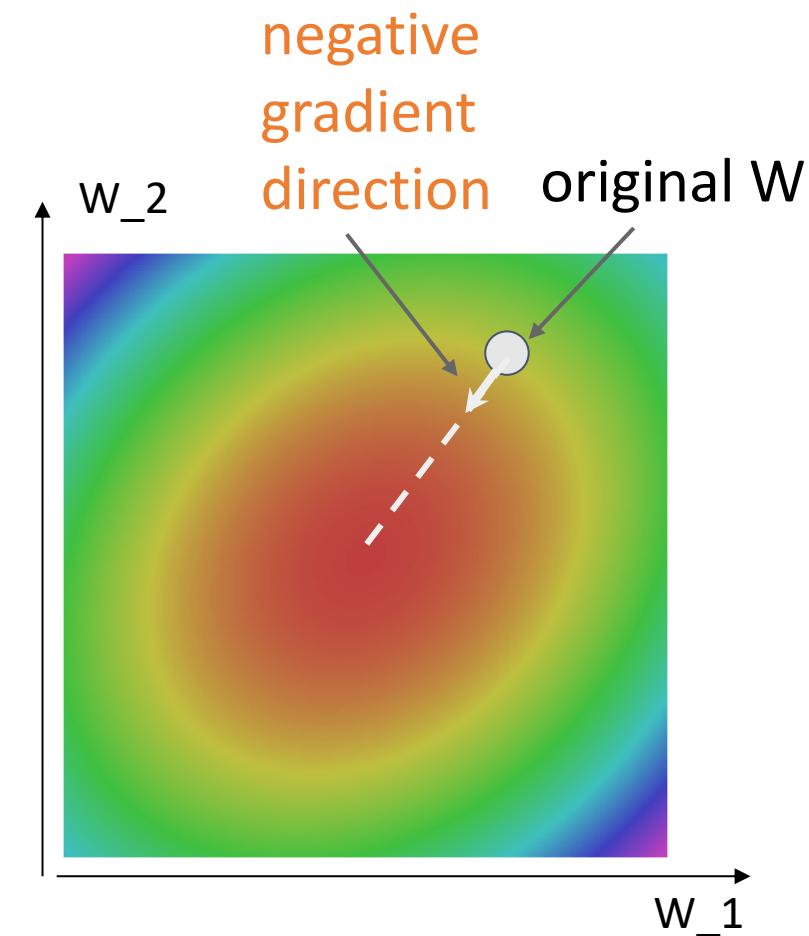
Gradient Descent

Iteratively step in the direction of the negative gradient
(direction of local steepest descent)

```
# Vanilla gradient descent
w = initialize_weights()
for t in range(num_steps):
    dw = compute_gradient(loss_fn, data, w)
    w -= learning_rate * dw
```

Hyperparameters:

- Weight initialization method
- Number of steps
- Learning rate



Batch Gradient Descent

$$L(W) = \frac{1}{N} \sum_{i=1}^N L_i(x_i, y_i, W) + \lambda R(W)$$

Problem: Full sum is expensive when N is large!

$$\nabla_W L(W) = \frac{1}{N} \sum_{i=1}^N \nabla_W L_i(x_i, y_i, W) + \lambda \nabla_W R(W)$$

Solution: Approximate sum using a minibatch of examples, e.g. 32

Stochastic Gradient Descent (SGD)

$$L(W) = \frac{1}{N} \sum_{i=1}^N L_i(x_i, y_i, W) + \lambda R(W)$$

Problem: Full sum is expensive when N is large!

$$\nabla_W L(W) = \frac{1}{N} \sum_{i=1}^N \nabla_W L_i(x_i, y_i, W) + \lambda \nabla_W R(W)$$

Solution: Approximate sum using a minibatch of examples, e.g. 32

```
# Stochastic gradient descent
w = initialize_weights()
for t in range(num_steps):
    minibatch = sample_data(data, batch_size)
    dw = compute_gradient(loss_fn, minibatch, w)
    w -= learning_rate * dw
```

Hyperparameters:

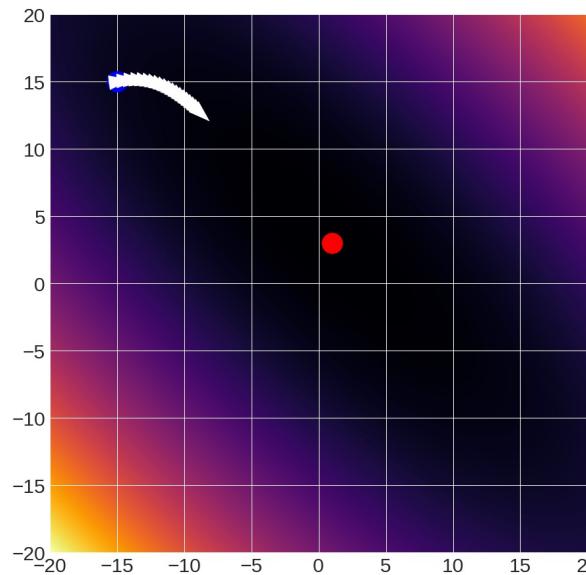
- Weight initialization
- Number of steps
- Learning rate
- Batch size
- Data sampling

Gradient Descent: Learning Rate

Step size (also called **learning rate / lr**)
critical parameter

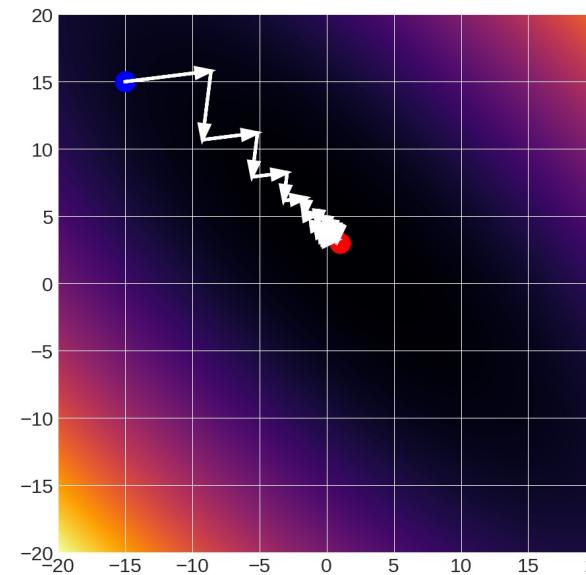
1×10^{-2}

falls short



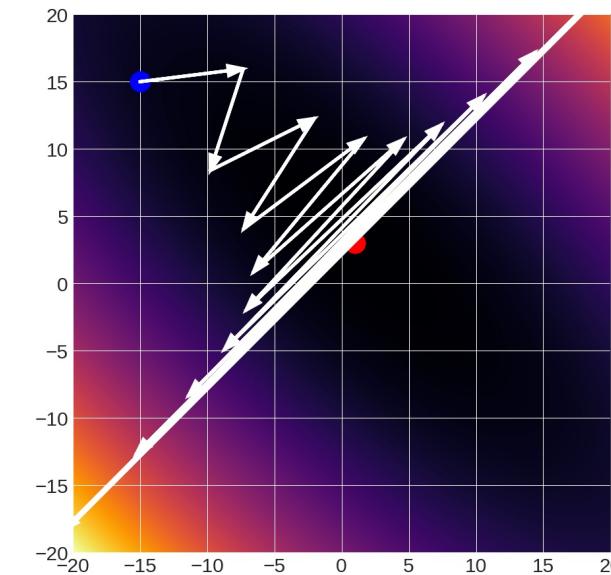
10×10^{-2}

converges



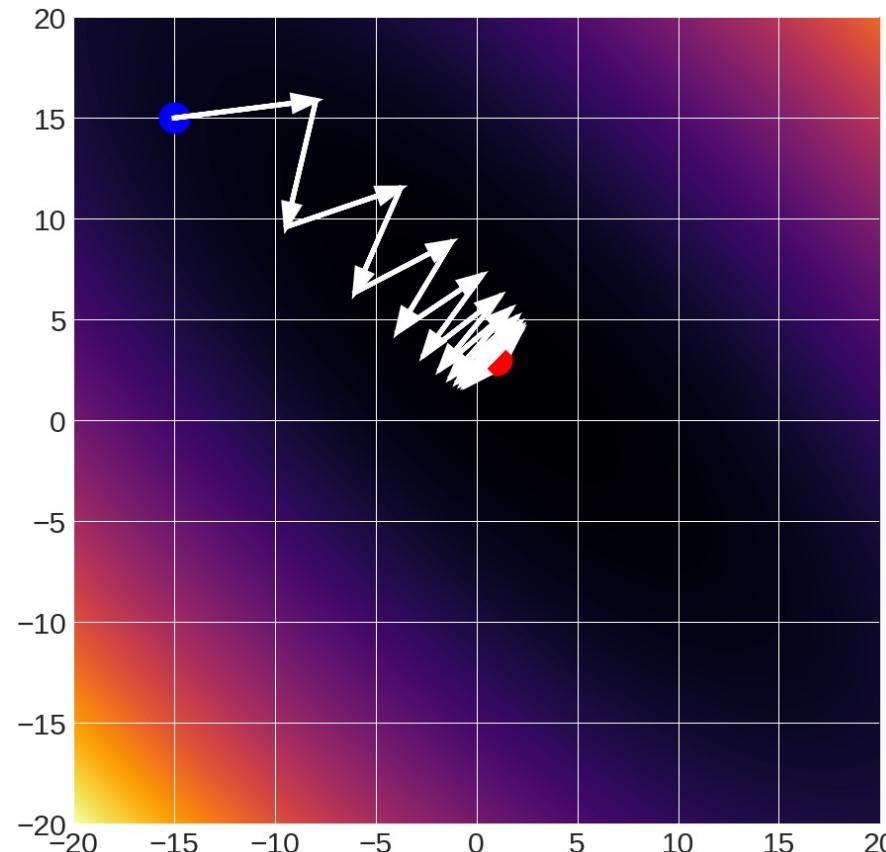
12×10^{-2}

diverges

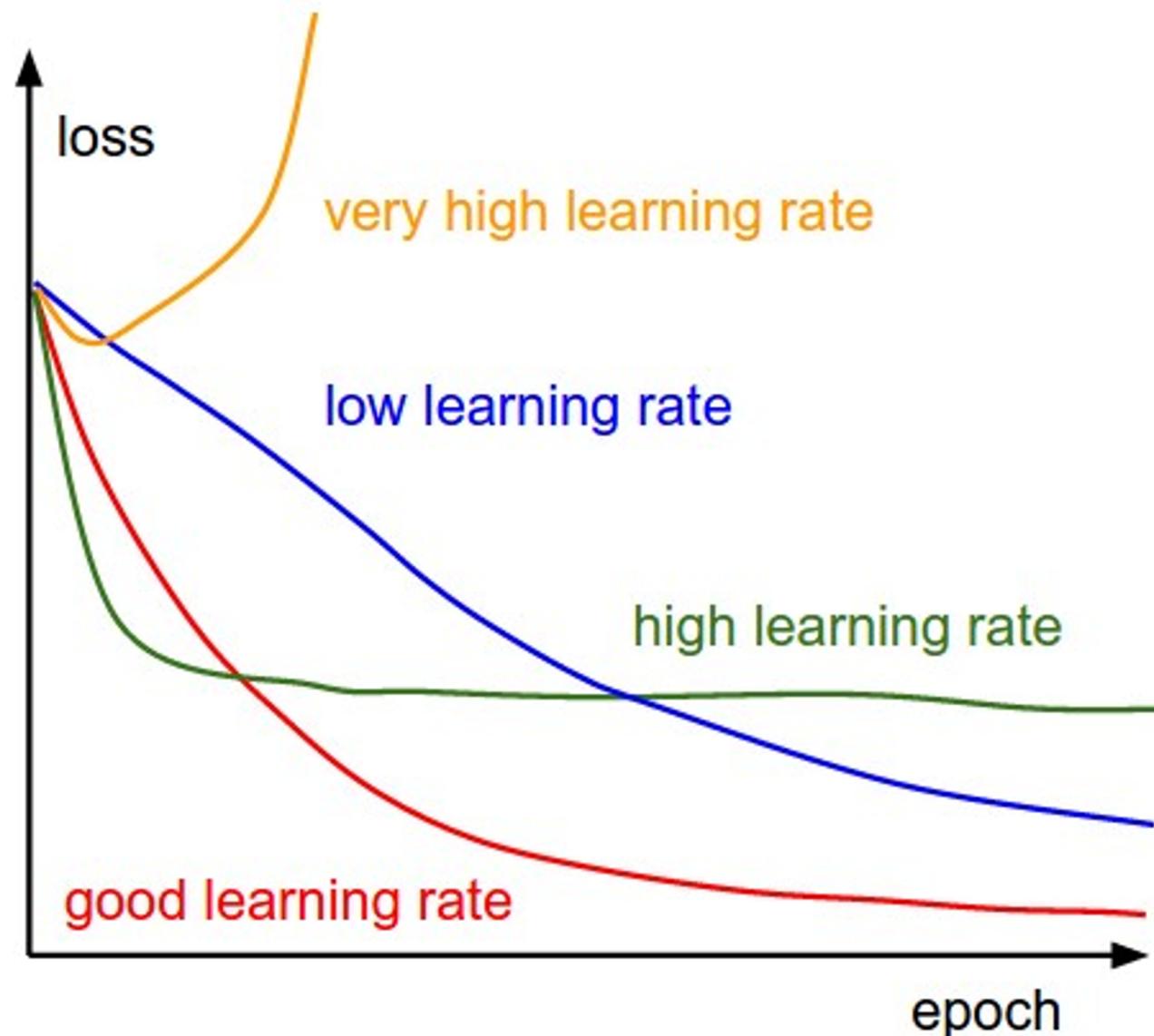


Gradient Descent: Learning Rate

11×10^{-2} : oscillates



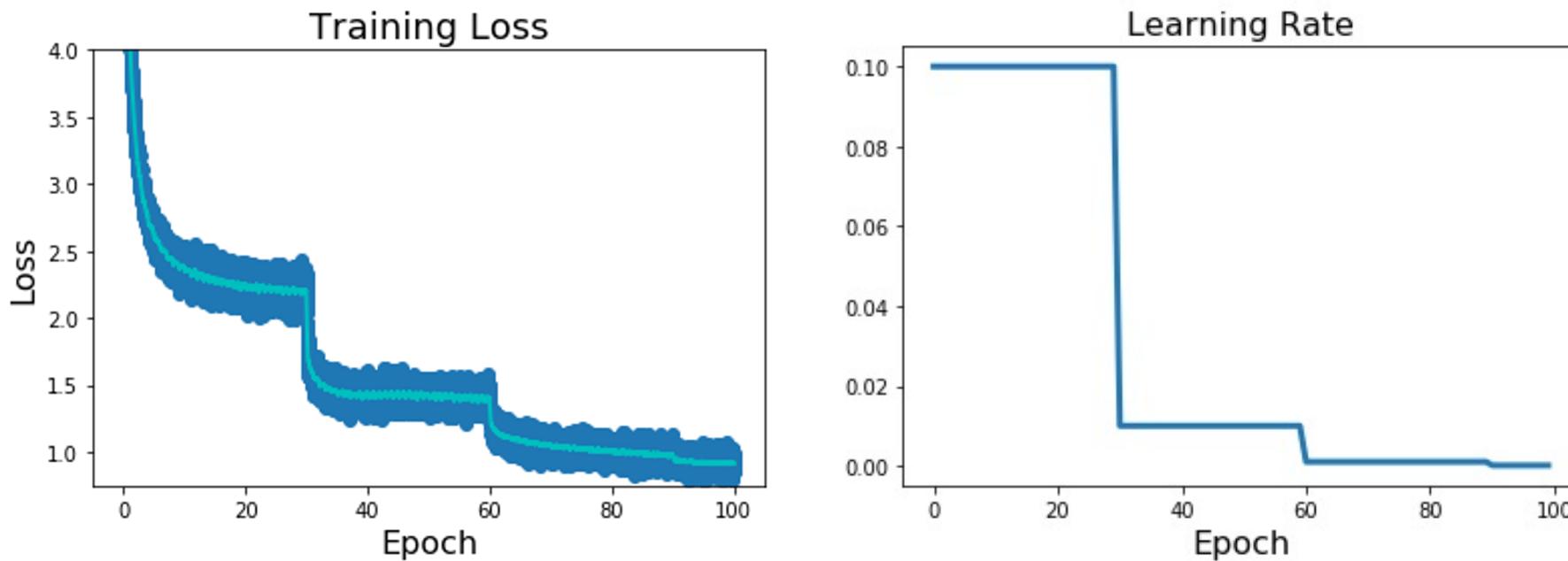
Effects of Learning Rate (Step Size)



Learning Rate Decay

Idea: Start with high learning rate, reduce it over time.

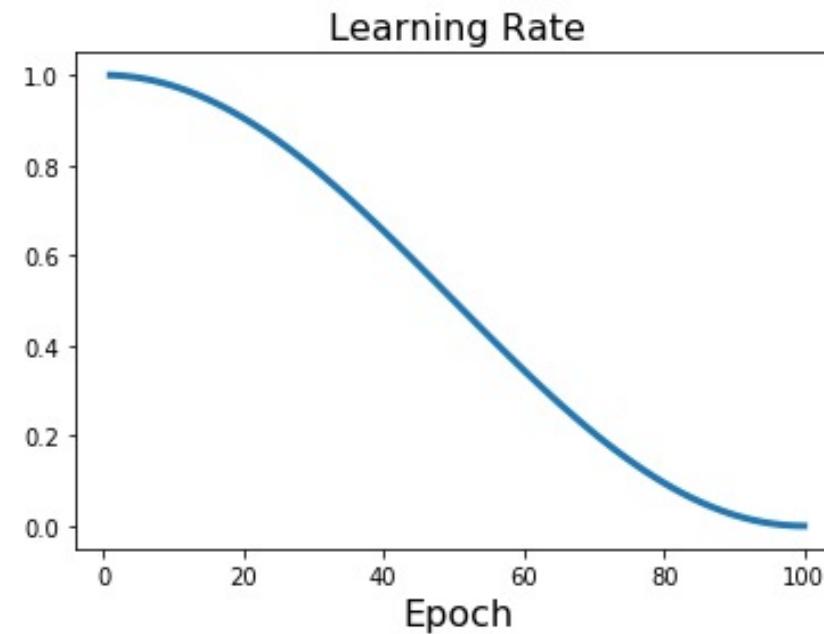
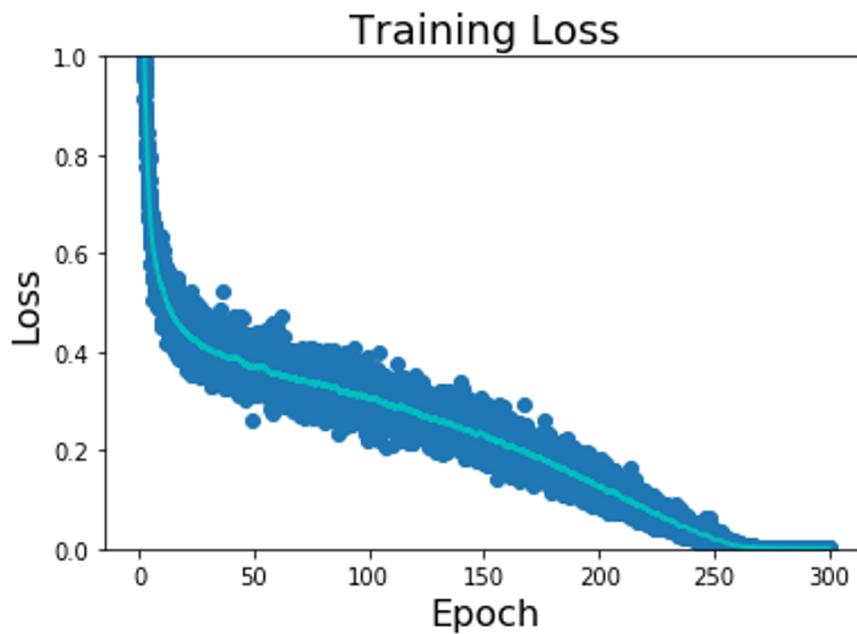
Step Decay: Reduce by some factor at fixed iterations



Learning Rate Decay

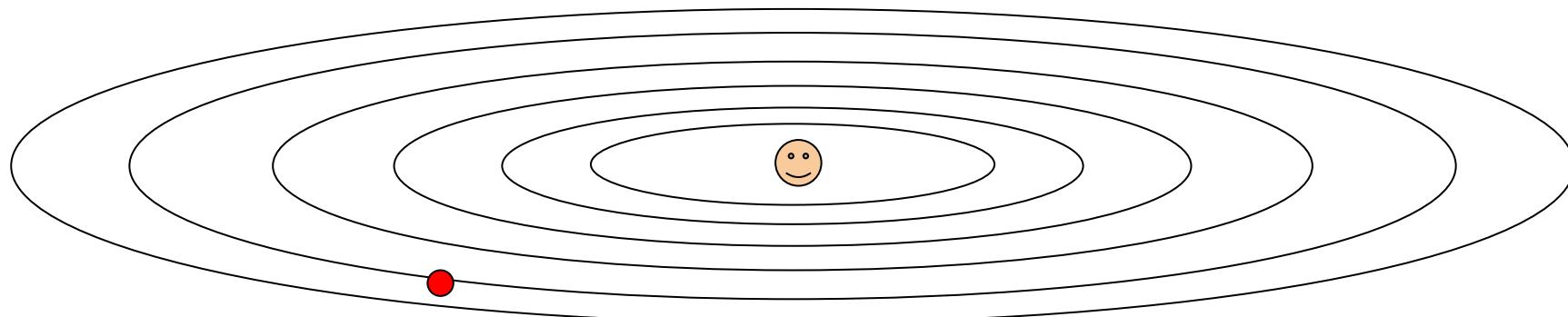
Idea: Start with high learning rate, reduce it over time.

$$\textbf{Cosine Decay: } \alpha_t = \frac{1}{2} \alpha_0 \left(1 + \cos \left(\frac{t\pi}{T} \right) \right)$$



Problems with SGD

What if loss changes quickly in one direction and slowly in another?

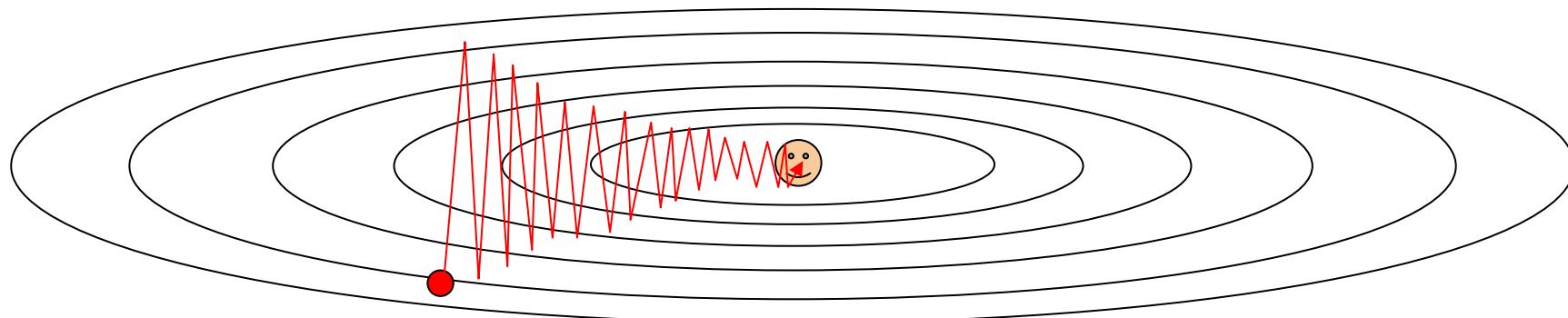


Loss function has high **condition number**: ratio of largest to smallest singular value of the Hessian matrix is large

Problems with SGD

What if loss changes quickly in one direction and slowly in another?

Slow progress along shallow dimension, jitter along steep direction

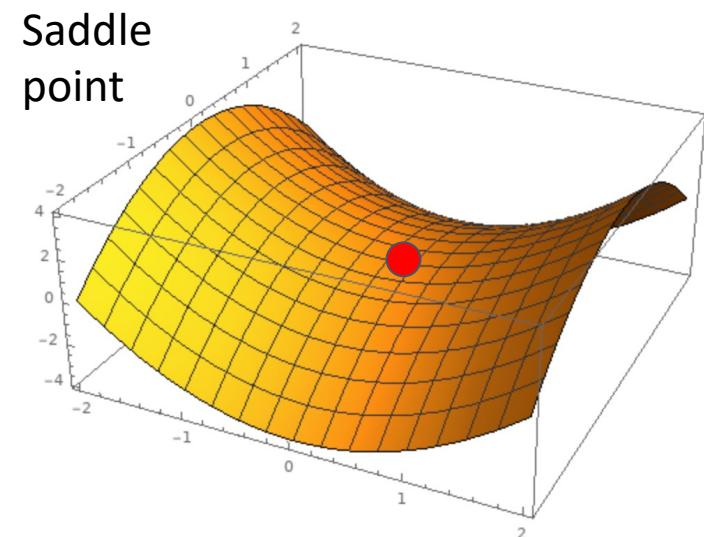


Loss function has high **condition number**: ratio of largest to smallest singular value of the Hessian matrix is large

Problems with SGD

What if the loss function
has a **local minimum** or
saddle point?

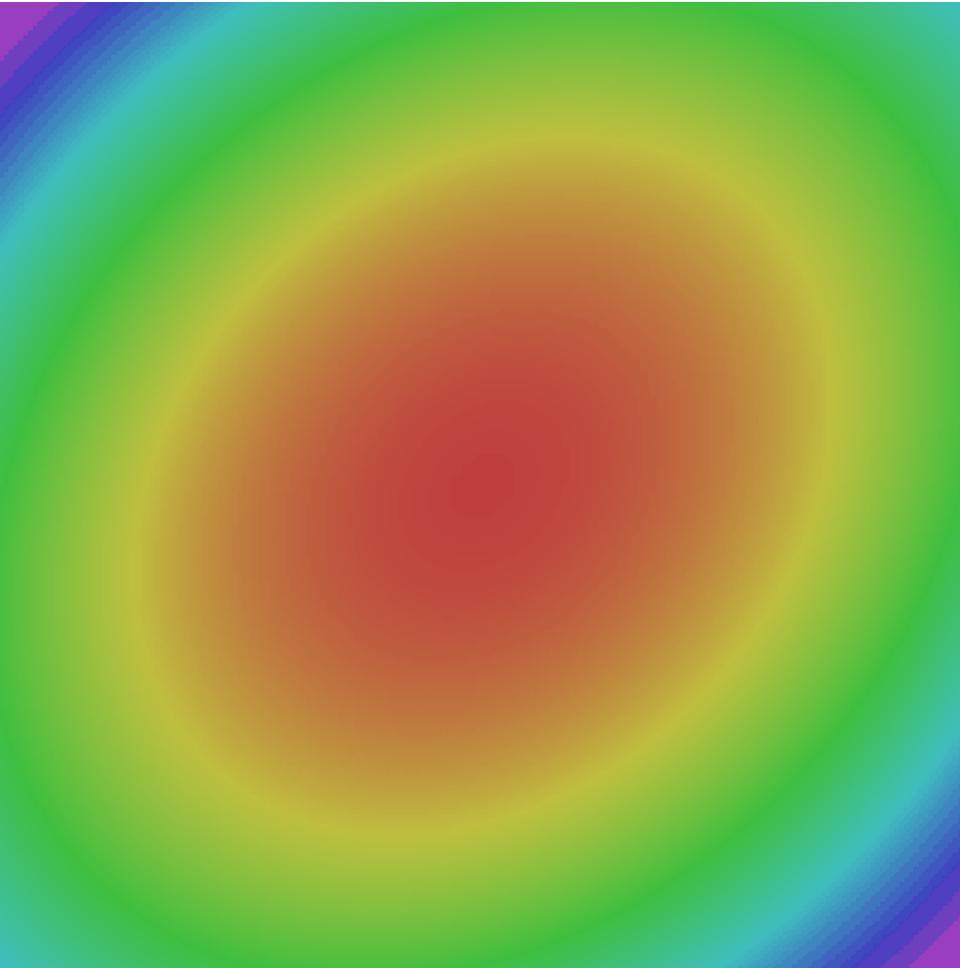
Gradient is zero,
SGD gets stuck



Problems with SGD

Our gradients come from minibatches so they can be noisy!

$$L(W) = \frac{1}{N} \sum_{i=1}^N L_i(x_i, y_i, W) + \lambda R(W)$$



SGD

SGD

$$x_{t+1} = x_t - \alpha \nabla f(x_t)$$

```
for t in range(num_steps):
    dw = compute_gradient(w)
    w -= learning_rate * dw
```

Sutskever et al, "On the importance of initialization and momentum in deep learning", ICML 2013

SGD + Momentum

SGD

$$x_{t+1} = x_t - \alpha \nabla f(x_t)$$

```
for t in range(num_steps):
    dw = compute_gradient(w)
    w -= learning_rate * dw
```

SGD + Momentum

$$\begin{aligned} v_{t+1} &= \rho v_t + \nabla f(x_t) \\ x_{t+1} &= x_t - \alpha v_{t+1} \end{aligned}$$

```
v = 0
for t in range(num_steps):
    dw = compute_gradient(w)
    v = rho * v + dw
    w -= learning_rate * v
```

- Build up “velocity” as a running mean of gradients
- Rho gives “friction”; typically $\rho = 0.9$ or 0.99

Sutskever et al, “On the importance of initialization and momentum in deep learning”, ICML 2013

SGD + Momentum

SGD + Momentum

$$v_{t+1} = \rho v_t - \alpha \nabla f(x_t)$$
$$x_{t+1} = x_t + v_{t+1}$$

```
v = 0
for t in range(num_steps):
    dw = compute_gradient(w)
    v = rho * v - learning_rate * dw
    w += v
```

SGD + Momentum

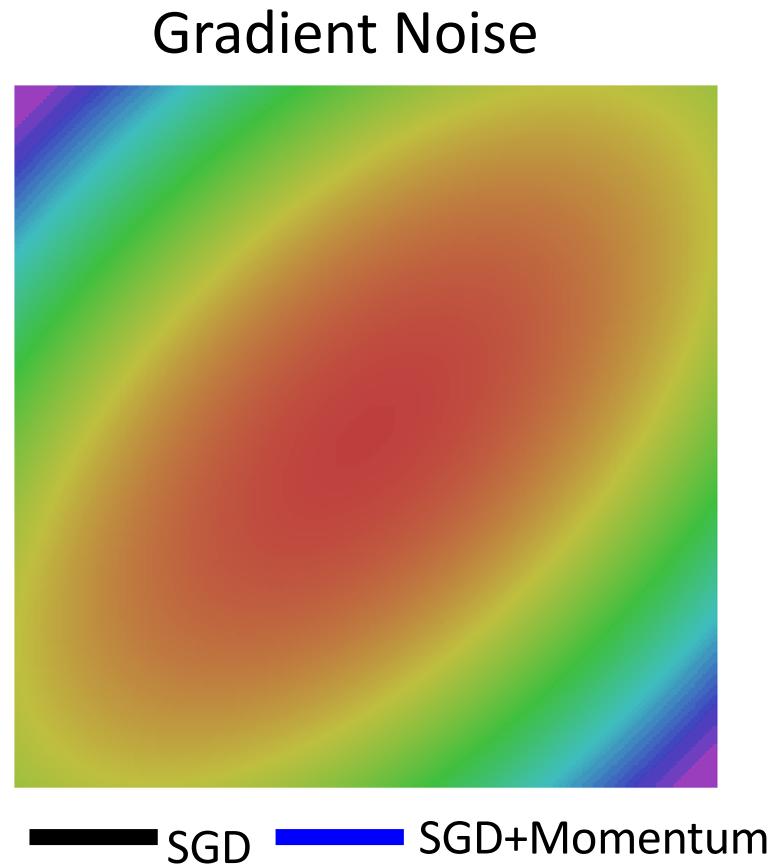
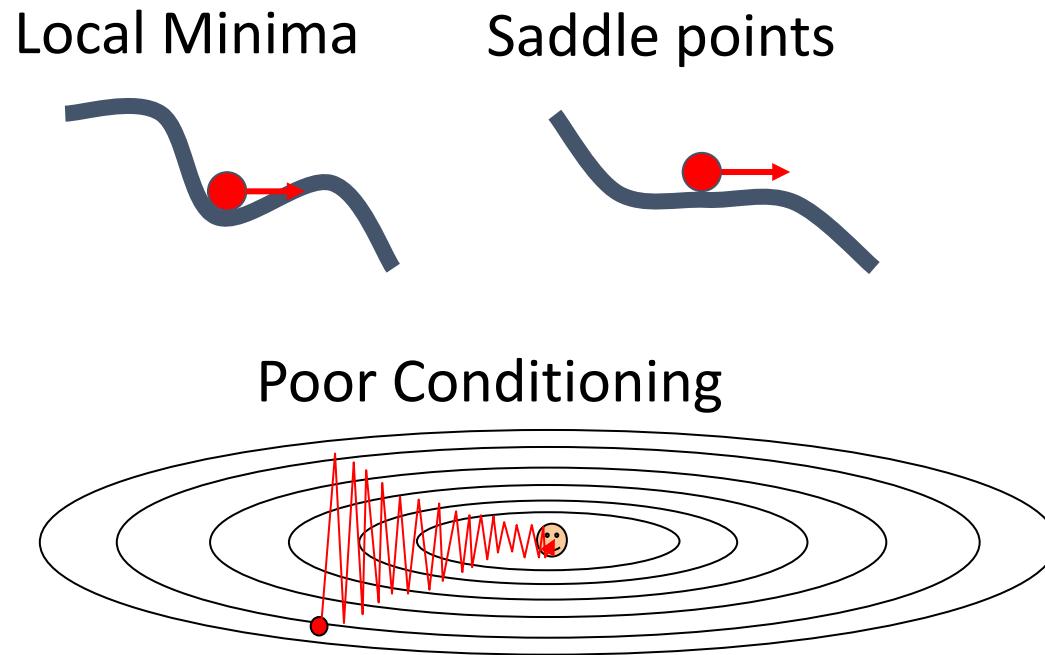
$$v_{t+1} = \rho v_t + \nabla f(x_t)$$
$$x_{t+1} = x_t - \alpha v_{t+1}$$

```
v = 0
for t in range(num_steps):
    dw = compute_gradient(w)
    v = rho * v + dw
    w -= learning_rate * v
```

You may see SGD+Momentum formulated different ways, but they are equivalent - give same sequence of x

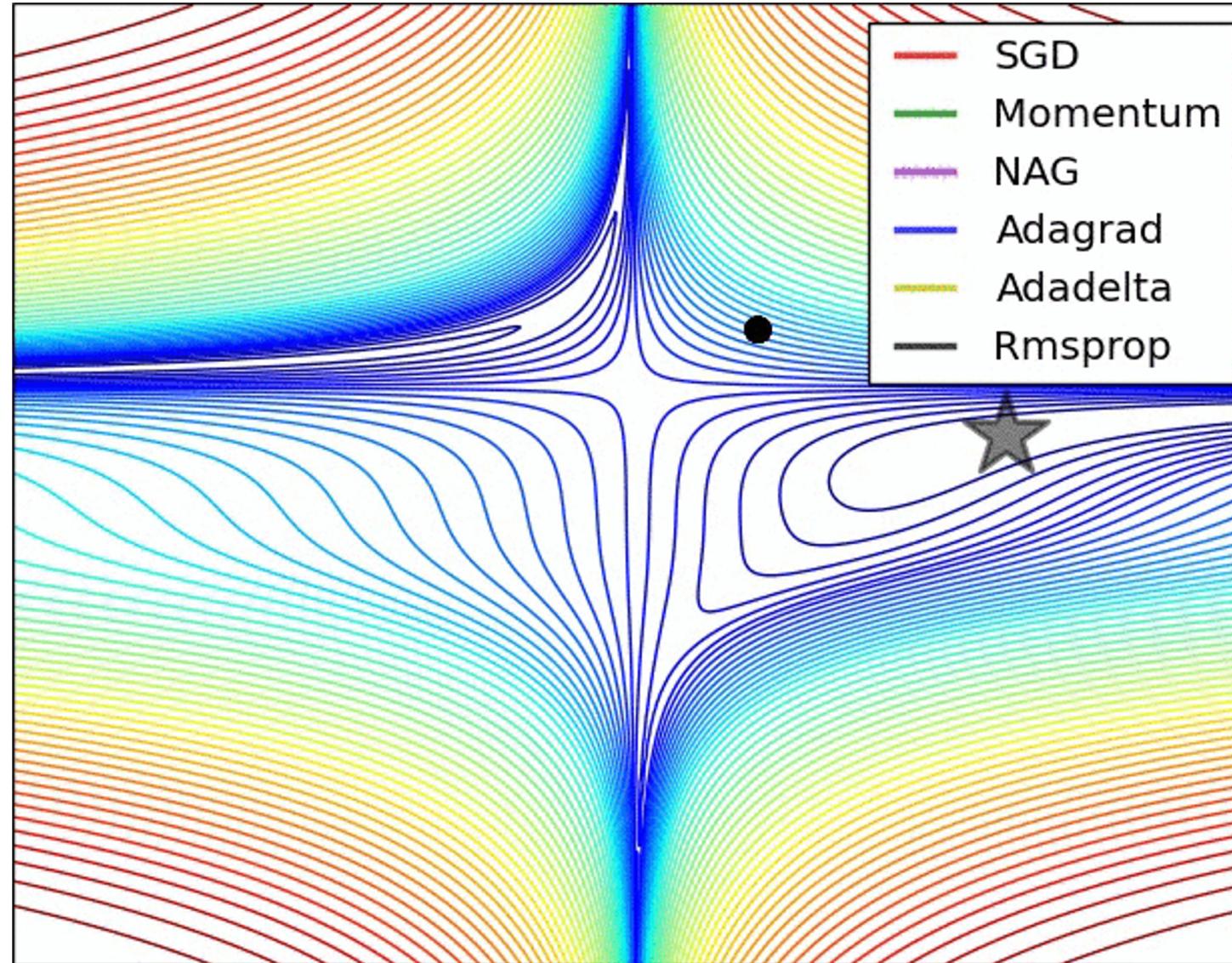
Sutskever et al, "On the importance of initialization and momentum in deep learning", ICML 2013

SGD + Momentum



Sutskever et al, "On the importance of initialization and momentum in deep learning", ICML 2013

The effects of different update form formulas



(image credits to Alec Radford)

Optimization in Practice

- **Conventional wisdom:** minibatch stochastic gradient descent (SGD) + momentum (package implements it for you) + some sensibly changing learning rate
- The above is typically what is meant by “SGD”
- Other update rules exist (Adam very common); sometimes better, sometimes worse than SGD

Optimizing Everything

$$L(\mathbf{W}) = \lambda \|\mathbf{W}\|_2^2 + \sum_{i=1}^n -\log \left(\frac{\exp((\mathbf{W}\mathbf{x})_{y_i})}{\sum_k \exp((\mathbf{W}\mathbf{x})_k)} \right)$$

$$L(\mathbf{w}) = \lambda \|\mathbf{w}\|_2^2 + \sum_{i=1}^n (y_i - \mathbf{w}^T \mathbf{x}_i)^2$$

- Optimize \mathbf{w} on training set with SGD to maximize training accuracy
- Optimize λ with random/grid search to maximize validation accuracy
- Note: Optimizing λ on training sets it to 0

Next Class

Neural Networks!