CSCI 566: Deep Learning and Its Applications

Jesse Thomason

Lecture 2: DL and Machine Learning

Welcome to CSCI 566!

- This course is still evolving and we are trying out some new things this time.
- The syllabus, course policy, and grading details may change over the semester.
- It will be fun but challenging!
- Our main organizing tool will be **Piazza**
- **Website:** https://csci566-spring2023.github.io/
- **Piazza:** https://piazza.com/usc/spring2023/csci566/info

CSCI 566 Resources

- **Piazza:** https://piazza.com/usc/spring2023/csci566/info
- **Website:** https://csci566-spring2023.github.io/
- **GDrive:** https://drive.google.com/drive/folders/11YcbSZcJUdRSbrNBWiOs_HM2VkxdNKWS?usp=share_link
- **Zoom:** https://usc.zoom.us/j/95538924150?pwd=TTRBM283VCt4WDFkN1hLMmtrKzdGUT09
- Where to find these links?
  - Piazza → Resources → Course Information

CSCI 566 Resources

- **GDrive:** https://drive.google.com/drive/folders/11YcbSZcJUdRSbrNBWiOs_HM2VkxdNKWS?usp=share_link
  - Living syllabus
  - Lecture slides
  - Links for Zoom recordings of past lectures

CSCI 566 Resources

- **Piazza:** https://piazza.com/usc/spring2023/csci566/info
- Please use Piazza for any course related communication
- You will be added to CARC/Piazza/etc. en masse when D-clearances through Blackboard are done propagating
  - If you're new to the course, please fill Quiz 0 in the Resources tab of Piazza
- Personal matters and emergency communication only:
  - csci566.spring2023@gmail.com
  - Any email that should have been a Piazza post will be ignored without acknowledgement

Course Office Hours

Jesse Thomason

- Instructor Office Hours
  - Mon 10am-11am
  - SAL 244
- NOT for homework related questions.

- TA Office Hours
- HW questions; project questions & feedback
- Gautam + Deqing
  - Wednesdays 3-4pm
- Tejas + Bingjie
  - Thursdays 10am-11am
- Location:
  - RTH 4th Floor
  - See website/Piazza for most recent

Deqing Fu

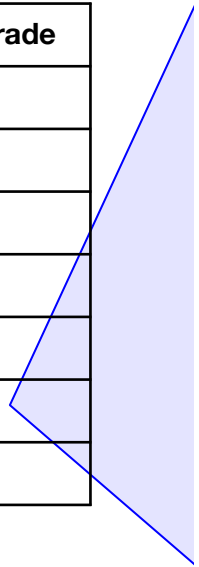
Gautam Salhotra

Tejas Srinivasan

Bingjie Tang

Course Deliverables

Deliverable	Points of the total grade
Pop-up Quizzes	5
Assignment #1	10
Assignment #2	10
Paper Roleplaying Breakout	10
Midterm	20
Course Project	45
TOTAL	**100**



Course Project - Quick Recap

- Project teaming starts now; look for people with common interests and complementary skills!
  - Inform the TAs via filling up the online form (one submission for each team); this form will be posted to Piazza.
  - For project grades, whole team receives the same grade for each of these entries.
- **Teams of 4** (strictly enforced)
  - You may *not* work with students outside this course.
  - You may *not* share code with other teams in the course.
  - You *must* enumerate individual contributions in reports

Map to the Midterm

January 2023

Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

www.a-printable-calendar.com

February 2023

Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28				

www.a-printable-calendar.com

Module 1: Neural Network Basics

CSCI 566 Roadmap

- **Module 1:** Neural Network Basics
  - Jan 13 - Feb 24 [5 lecture days, 1 exam day, 1 free week]
  - Some lectures will involve tutorials from TAs
  - Coding Assignment 1
- **Module 2:** Deep Learning Applications
  - Mar 3 - Mar 24 [3 lecture days, 1 free week]
  - Project lightning pitches in class
  - Project survey due
  - Coding Assignment 2

CSCI 566 Roadmap

- **Module 3:** Advanced Topics in Deep Learning
  - Mar 31 - Apr 28 [4 lecture days, 2 presentation days]
  - Paper Roleplaying Breakout sessions
  - Project mid-term report due
  - Final project presentations
- **Final Report**
  - Due on the final exam day, **Wednesday May 3**
  - There will be no in-class final exam
  - You **may** use Late Day Tokens for the final report

Welcome to CSCI 566!

- This course is still
  things this time.
- The syllabus, cours
  over the semester.
- It will be fun but challenging!
- Q
  will be **Piazza**
  -spring2023.github.io/
  m/usc/spring2023/csci566/info

For example, our enrollment just increased by 100 students.
Welcome, 100 new students!
Please look over Lecture 1 slides online!

That many new faces and Project Teams means we have some adjustments to our planned assignments.

Changes To Course Project Deliverables

- Project pitches → 1 minute lightning pitches given by one representative per team
  - Previously these would have been longer
- Final presentations → 5 minutes per team, spread over two weeks' meeting times (i.e., two of our classes)
  - Previously, these would have been longer and maybe crammed into a single week's meeting time
- Paper Roleplaying Breakout Sessions [Module 3]
  - Replaces "Paper Presentations" given by team members

Due Dates and Late Day Tokens

- Material is always due at 11:59pm PT on the day of the deadline; the late penalty is 5% off the total score per day after
- Each student will have **5 Late Day Tokens** to expend as they see fit on any *non-presentation-based* assignments
- When you turn something in late, you must leave a note with the assignment specifying how many Late Day Tokens you wish to expend
- Tokens are individual, so when using for project-related assignments they must be expended by all team members; team members who do not expend will receive the late penalty

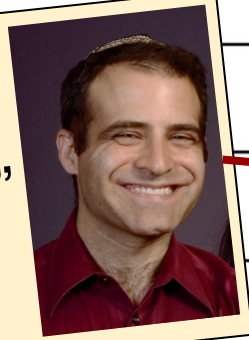
Course Supplemental Readings

- In most weeks there are *suggested* readings that will help you deep dive into lecture topics.
- Textbooks:
  - Deep Learning (MIT Press)
    - Ian Goodfellow, Yoshua Bengio, and Aaron Courville
    - Free online version is available at http://www.deeplearningbook.org/
  - Neural Network Methods for Natural Language Processing
    - Yoav Goldberg

[Jan 20] A Look at January

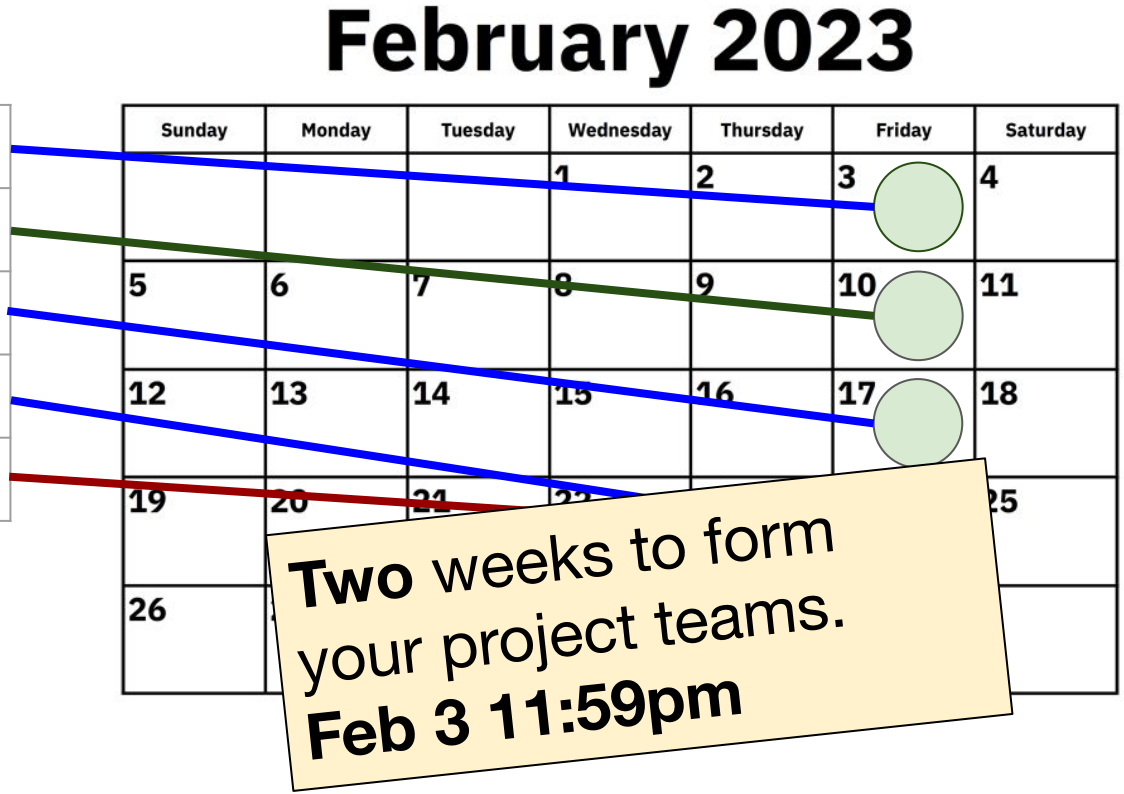
Week 1	Jan 13
Week 2	Jan 20
No Class Next Week!	Jan 27

CS Invited Talk: David Held
"Robots Perceiving And Doing"
Jan 27 11am-12pm RTH 115



[Jan 20] A Look at February Deliverables

Project Teams Formed	Feb 3
Assignment 1 Out	Feb 10
Project Proposal Due	Feb 17
Assignment 1 Due	Feb 24
Midterm Exam	Feb 24



Course Project

- Form 4-person teams
- Select from provided project ideas
  - https://docs.google.com/document/d/1bQusP0uS8KoVk7ZmxpGs209gBD3N9p-aSzMTTM63k1k/
    - This is a *living document*; you can expect more projects to be added, especially when we get more TAs!! 😮
    - We will add this doc to Piazza resources
- *Or* design your own project, using the project ideas doc as an outline for what you should include in your proposal slide deck

Course Compute

- We have applied for both CARC and Google Cloud Platform credits for the course
  - CARC is approved!
- We will have TA-led tutorials on using these resources during Module 1

Overview of Today's Plan

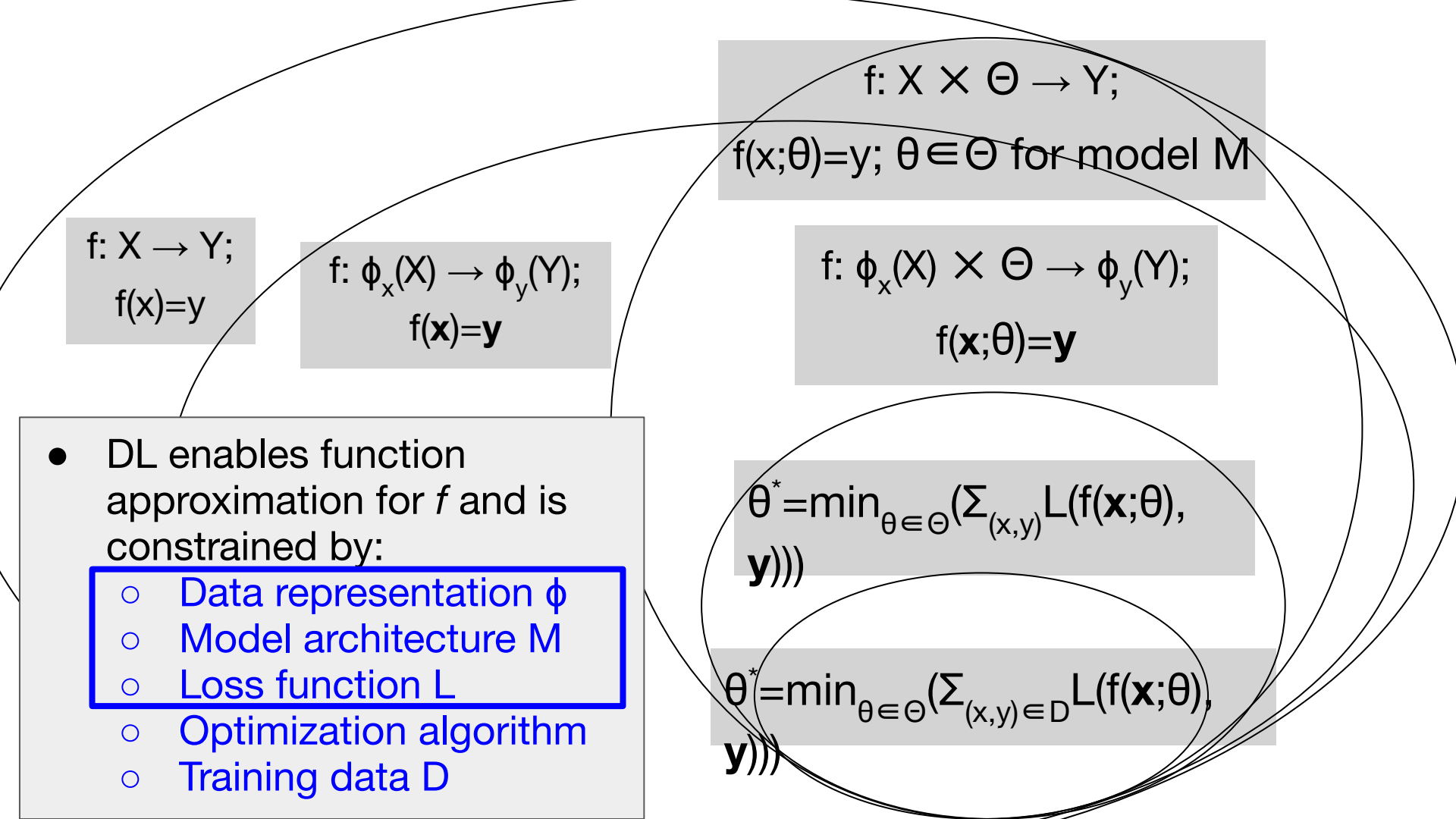
- ~~Course organization and deliverables recap~~
  - Any questions before we move on?
- Framing Problems for Machine Learning
- Loss Functions and Optimization
- Representations and Models

Framing Problems for Machine Learning

- What is the relationship between Deep Learning and Machine Learning?
  - Deep Learning $\subset$ Machine Learning
- What is a reasonable working definition of Artificial Intelligence?
  - Computer *systems* whose *applications* are those that are normally done by humans (e.g., translation, perception)
- What is the relationship between Deep Learning and AI?
  - AI *systems* utilize DL as a method to approach *applications*

Framing Problems for Machine Learning

- How do we frame an AI application as a deep learning problem?
  - We can start by framing it as a machine learning problem, since Deep Learning $\subset$ Machine Learning
- What is the most general formulation of the goal of Machine Learning we can express?
  - Given an input $x \in X$, produce a desired output $y \in Y$, using a *learned function* $f(x) \approx y$.


f: X → Y;
f(x)=y
f: $\phi_x(X) \rightarrow \phi_y(Y)$;
f(**x**)=**y**
f: X × Θ → Y;
f(x;θ)=y; θ∈Θ for model M
f: $\phi_x(X) \times \Theta \rightarrow \phi_y(Y)$;
f(**x**;θ)=**y**
$\theta^* = \min_{\theta \in \Theta}(\Sigma_{(x,y)} L(f(\mathbf{x};\theta), \mathbf{y}))$
$\theta^* = \min_{\theta \in \Theta}(\Sigma_{(x,y) \in D} L(f(\mathbf{x};\theta), \mathbf{y}))$
• DL enables function approximation for *f* and is constrained by:
○ Data representation ϕ
○ Model architecture M
○ Loss function L
○ Optimization algorithm
○ Training data D

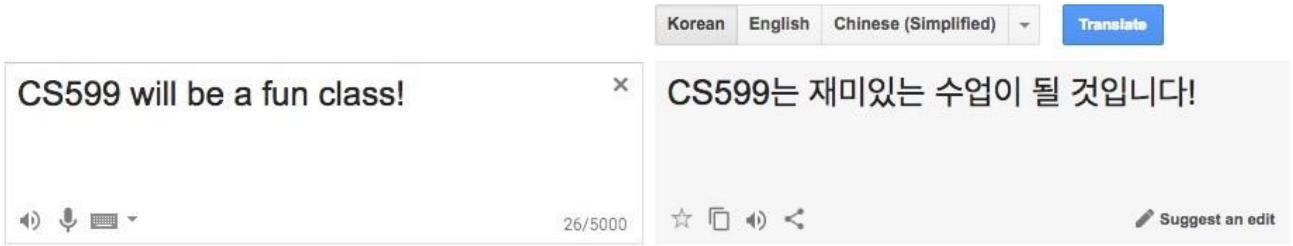
Framing Problems for Machine Learning

- If you can:
  - Squeeze your input and output into numerical representations (e.g., vectors)
  - Compose neural layers that eat something shaped like the input and spit out something shaped like the output
  - Define a loss function and apply an optimizer that are valid in the underlying problem and model space
- Then:
  - Your DL model will train and loss will go down, *even if you aren't actually solving the problem you wanted to solve*

Framing Problems for Machine Learning

- Different problem classes based on input and output spaces can be tackled by varying our function *f(x)*
- **Regression:**
  - Output space *y* is *continuous* and *ordered*
  - X={Input Texts}, Y={Reading Grade Level Difficulty}
- **Classification:**
  - Output space *y* is *discrete* and *unordered*
  - X={ImageNet Images}, Y={1000 ImageNet Classes}

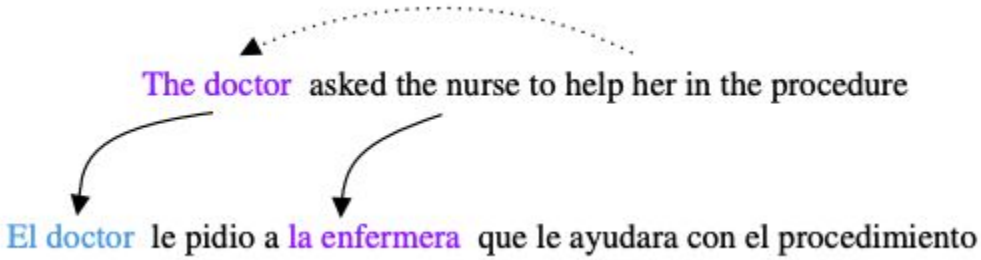
Framing Problems for Machine Learning

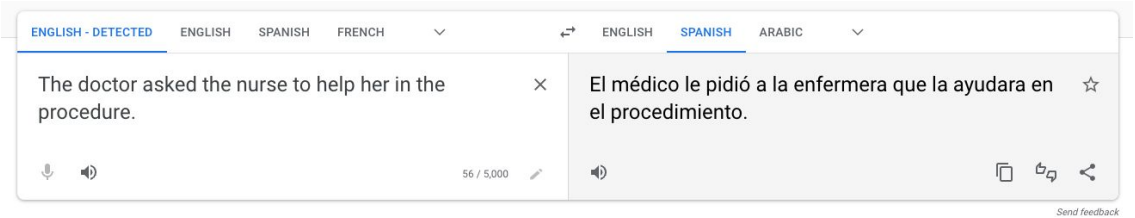


- Input space *X*?
  - Source language *tokens S*
  - $X=S^N$ for maximum translation source length *N*
- Output space *Y*?
  - Target language *tokens T*
  - $Y=T^M$ for maximum translation target length *M*

- What are tokens? Why might *N* and *M* differ?
- What might our function *f(x)* look like? Classification or regression?

Framing Problems for Machine Learning

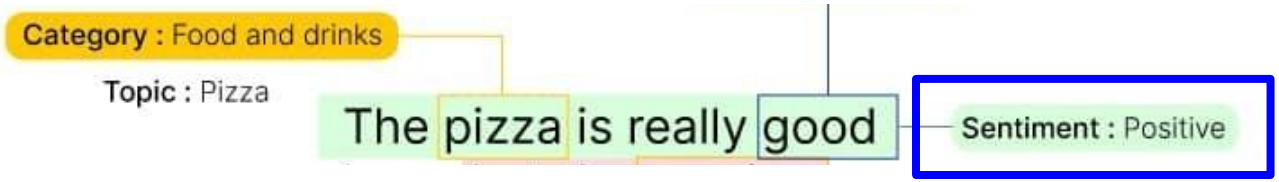




- What might our function *f(x)* look like?
- What went wrong when learning *f(x)* to enable this mistake?

Evaluating Gender Bias in Machine Translation (2019)

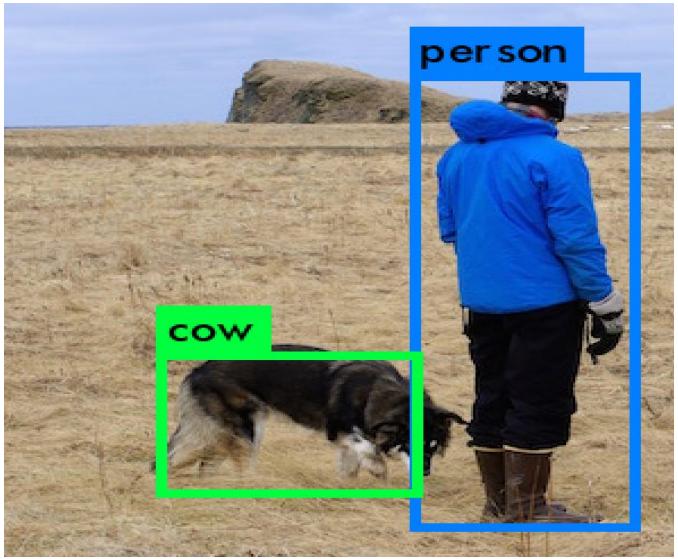
Framing Problems for Machine Learning



- Input space *X*?
  - Review *tokens T*
  - $X=T^N$ for maximum review length *N*
- Output space *Y*?
  - Possible sentiments *S*
  - *Y=S*

- What labels might you put in the set *S*?
- What might our function *f(x)* look like? Classification or regression?

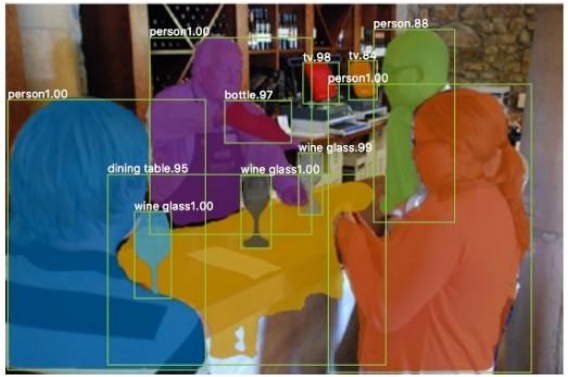
Framing Problems for Machine Learning



- Input space *X*?
  - Pixels in the image
  - $X=\boldsymbol{R}^{W,H,3}$
- Output space *Y*?
  - Object classes *C*
  - Object bounding boxes
  - $Y=C \times \boldsymbol{R}^{x,y,w,h} \times N$

- What's *N* in the output space?
- What might our function *f(x)* look like? Classification or regression?

Deep Learning for Vision: Object Segmentation



- Input space *X*?
  - Pixels in the image
  - $X=\boldsymbol{R}^{W,H,3}$
- Output space *Y*?
  - Object classes |C| per pixel in the image
  - $Y=\boldsymbol{R}^{W,H,|C|}$

- How else might you represent *Y*?
- What might our function *f(x)* look like? Classification or regression?

Framing Problems for Machine Learning

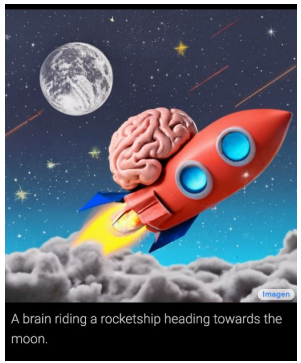
Who is wearing glasses?

man

- Input space *X*?
  - Pixels in the image
  - Tokens in the question
  - $X=\boldsymbol{R}^{W,H,3} \times T^{N}$
- Output space *Y*?
  - Possible answers *A*
  - *Y=A*

- How else might you represent *Y*?
- What might our function *f(x)* look like? Classification or regression?

Framing Problems for Machine Learning



- Input space X?
  - Tokens in the caption
  - $X=T^N$
- Output space Y?
  - Pixels of an image
  - $Y=\boldsymbol{R}^{W,H,3}$

- What might our function *f(x)* look like? Classification or regression?
- Why does this example *feel different* from the others?

ML FUNDAMENTALS: Discriminative v. Generative

Bayes' Rule

$$P(X,Y) = P(X|Y)P(Y) = P(Y|X)P(X)$$

Likelihood of Y

Prior of Y

Posterior of Y

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)}$$

Discriminative

Estimate P(Y|X) from data.
E.g., {f(x)=y} ↔ {p(y|x)}

Generative

Estimate P(X|Y) and P(Y) from data.
E.g., {f(x)=y} ↔ {p(y) occurs and p(x|y) that y *explains* observed x}

Framing Problems for Machine Learning

Prompt: ceo;
Date: April 6, 2022

- Should we estimate *f(x)* with a discriminative or generative approach?
- What went wrong in the estimation of *f(x)* above?

Framing Problems for Machine Learning

- What is the most general formulation of the goal of Machine Learning we can express?
  - Given an input $x \in X$, produce a desired output $y \in Y$, using a *learned function* $f(x) \approx y$.
- How do we start deep learning?
  - Well, first let's think about whether we oughta, and then see what might slow us down along the way.

DL FUNDAMENTALS: Start Without DL

- Always look at your data! What will the model see?
- Always implement a strong baseline!
  - Strong does not always mean “clever”
  - Strong *never* means “fancy” or “complicated”
    - https://twitter.com/seanjtaylor/status/1550326602105466880?s=20&t=0ZjH63z3S22vwAdG9l5D0g

Framing Problems for Machine Learning

- What is the most general formulation of the goal of Machine Learning we can express?
  - Given an input $x \in X$, produce a desired output $y \in Y$, using a *learned function* $f(x) \approx y$.
- What's a simple, non-degenerate learned function?
  - First assume we can transform input x into a *vector representation,* $\phi(x) = \boldsymbol{x}$
  - Then we can study a simple linear model: $f(\phi(x)) = W\phi(x) + b$
  - *Weight matrix* W, *bias vector* b

Linear Classification

- $f(\phi(x)) = W\phi(x) + b$
  - $f(\mathbf{x}) = W\mathbf{x} + b$
  - This is a familiar, linear function
  - Going back to Image Classification, the target application of the famous AlexNet, let's see how our linear function fits into the framework we're describing

Linear Image Classification

- Input space *X*?
  - Pixels in the image
  - $X=\boldsymbol{R}^{50,50,3}$
- Output space *Y*?
  - Object classes *C* = *{airplane, car, cat}*
  - *Y=C*

- f(**x**)=W**x**+b; size of *W* and *b*? How does *W**x**+b* give us a class *C*?
- *W* maps every pixel value (50*50*3) to a posterior probability of each C

Linear Image Classification

- *W* maps every pixel value (50*50*3) to a posterior probability of each C
- Then our output space should be dimension |*C*|*;* we can *encode Y* as a *one-hot* vector <0, 1, 0>
- To learn *f(x)=Wx+b* directly:
  - $W^{<|C| \text{ x } 50*50*3>}\mathbf{x}^{<50*50*3>} + b^{<|C|>}$

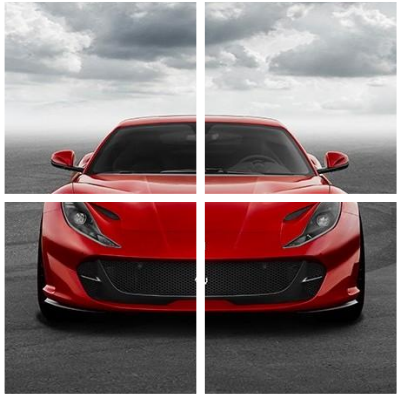
- Input space *X*?
  - Pixels in the image
  - $X=\boldsymbol{R}^{50,50,3}$

- Output space *Y*?
  - Object classes *C* = *{airplane, car, cat}*
  - *Y=C*

Linear Image Classification

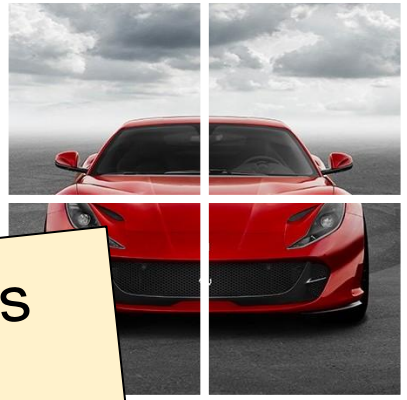
- *W* maps every pixel value (50*50*3) to a posterior probability of each C
- Then our output space should be dimension |*C*|*;* we can *encode Y* as a *one-hot* vector <0, 1, 0>
- To learn *f(x)=Wx+b* directly:
  - $W^{<|C| \times 50*50*3>}\mathbf{x}^{<50*50*3>} + b^{<|C|>}$
- That representation is really big for our slides demonstration
- Let's use our *encoding function ϕ(x)* to get a vector **x** from image input x that's smaller
- For simplicity, we'll divide the image into quadrants and create one *superpixel* per quadrant that's just the average intensity of all the pixels in it (e.g., flatten over x,y,RGB all together)

Linear Image Classification



- Now *W* maps every superpixel intensity value (2*2) to a posterior probability of each C
  - *Note*: yes, this is a dumb representation; bear with it for the purposes of the linear model demonstration.
- Remember our output space is now a *one-hot* vector <0, 0, 1, 0, 0> of length |C|
- Now To learn $f(\boldsymbol{x})=W\boldsymbol{x}+b$ directly:
  - $f(\mathbf{x})=W^{<2*2 \times |C|>}\mathbf{x}^{<2*2>} + b^{<|C|>}$
  - $f(\mathbf{x})=W^{<3x4>}\mathbf{x}^{<4>} + b^{<3>}$
  - In other words, we have 4 **input features** that we will learn to map to 3 **output classes**

Linear Image Classification



2x2 numbers

10	150

W: how much each input feature indicates each class (e.g., p(y|x))

ach class is

Great, so “Car” is the highest number. How do we get **W** and **b** though?

0.2	-0.3	0	0.6

150
7
232

+

0.2
-1
-0.9

=

96.4	Cat
212.8	Car
181.1	Airplane

W x b f(x; W,b)

Overview of Today's Plan

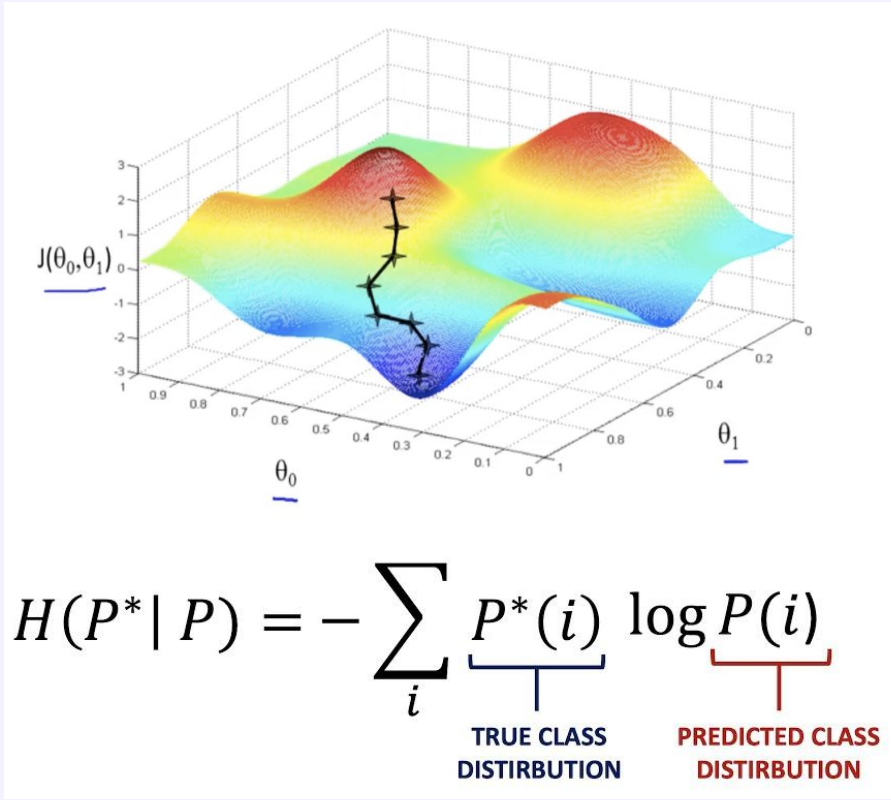
- ~~Course organization and deliverables recap~~
- ~~Framing Problems for Machine Learning~~
  - Any questions before we move on?
- Loss Functions and Optimization
- Representations and Models

ML FUNDAMENTALS: Neural Network Training

- A neural network is, commonly, a composition of matrix multiplications and non-linear transformations
- A neural network can perform *non-linear function approximation* through repeated linear layers and non-linear transformations
- Optimization: $M(\phi(x), \theta) = y$ for pairs (x, y) in data by making adjustments to θ
- Optimizers in modern NN packages are variations on the basics: **Stochastic Gradient Descent**

ML FUNDAMENTALS: Stochastic Gradient Descent

- Gradient Descent
- "Step" $\varepsilon \nabla L[M(\phi(x), \theta)]$
  - Go a little bit in the direction of the gradient (derivative) towards a local minima in the *loss*
- "Loss": measure of how poorly the model did, e.g., the cross-entropy between predicted and true classes



ML FUNDAMENTALS: Stochastic Gradient Descent

- "Step" $\varepsilon \nabla L[M(\phi(x), \theta)]$
- What's ε?
  - Learning rate
- Can we actually get ∇ *over all of D*?
  - Sometimes! Usually "no", and so we use *stochastic* GD
- Could estimate ∇_x with a single sample *(x,y)*. Why not?
- Could estimate $\nabla_{xi \ldots xj}$ with, e.g., as many examples as we can fit in RAM at a time $x_i \ldots x_j$. What do we call these?
- Each subset of *D* used to estimate gradient is a **mini-batch**

ML FUNDAMENTALS: Stochastic Gradient Descent

- "Step" $\varepsilon \nabla_{xi \ldots xj} L[M(\phi(x), \theta)]$
- **Learning rate**: ε
- **Mini-batch**: $(x,y)_i \ldots (x,y)_j$
- **Iteration**: $\theta := \theta - \varepsilon \nabla_{xi \ldots xj} L[M(\phi(x), \theta)]$
  - Move parameters in direction of gradient estimated for a mini-batch
- **Epoch**: Completed each time we've seen all of $(x,y) \in D$ again

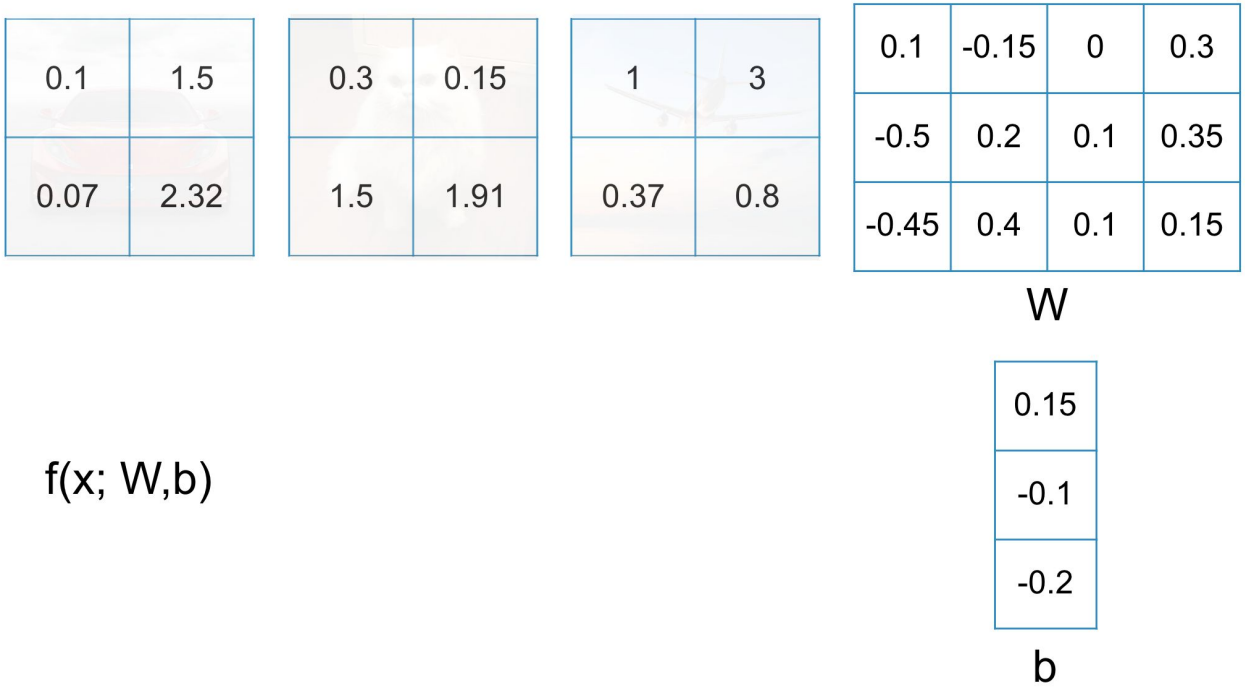
ML FUNDAMENTALS: Neural Network Training

- Optimizers in modern NN packages are variations on the basics: **Stochastic Gradient Descent**
- What do we need?
  - Pass over data for multiple **epochs**
  - Sample **mini-batches** of our data
  - Step the optimization algorithm to complete an **iteration**
- Let's step through a motivating example to see how we arrive at training deep neural networks for function approximation

Linear Image Classification

Linear Image Classification

- What does a good *f(x; W,b)* look like?



Linear Image Classification

- What does a good *f(x; W,b)* look like?

0.1	1.5
0.07	2.32

0.1	-0.15	0	0.3
-0.5	0.2	0.1	0.35
-0.45	0.4	0.1	0.15

W

0.1
1.5
0.07
2.32

x

+

0.15
-0.1
-0.2

b

=

0.631	Cat
0.969	Car
0.71	Airplane

f(x; W,b)

Linear Image Classification

- What does a good *f(x; W,b)* look like?

0.3	0.15
1.5	1.91

0.1	-0.15	0	0.3
-0.5	0.2	0.1	0.35
-0.45	0.4	0.1	0.15

W

0.3
0.15
1.5
1.91

x

+

0.15
-0.1
-0.2

b

=

0.73	Cat
0.598	Car
0.161	Airplane

f(x; W,b)

Linear Image Classification

- What does a good *f(x; W,b)* look like?

1	3
0.37	0.8

0.1	-0.15	0	0.3
-0.5	0.2	0.1	0.35
-0.45	0.4	0.1	0.15

W

1
3
0.37
0.8

x

+

0.15
-0.1
-0.2

b

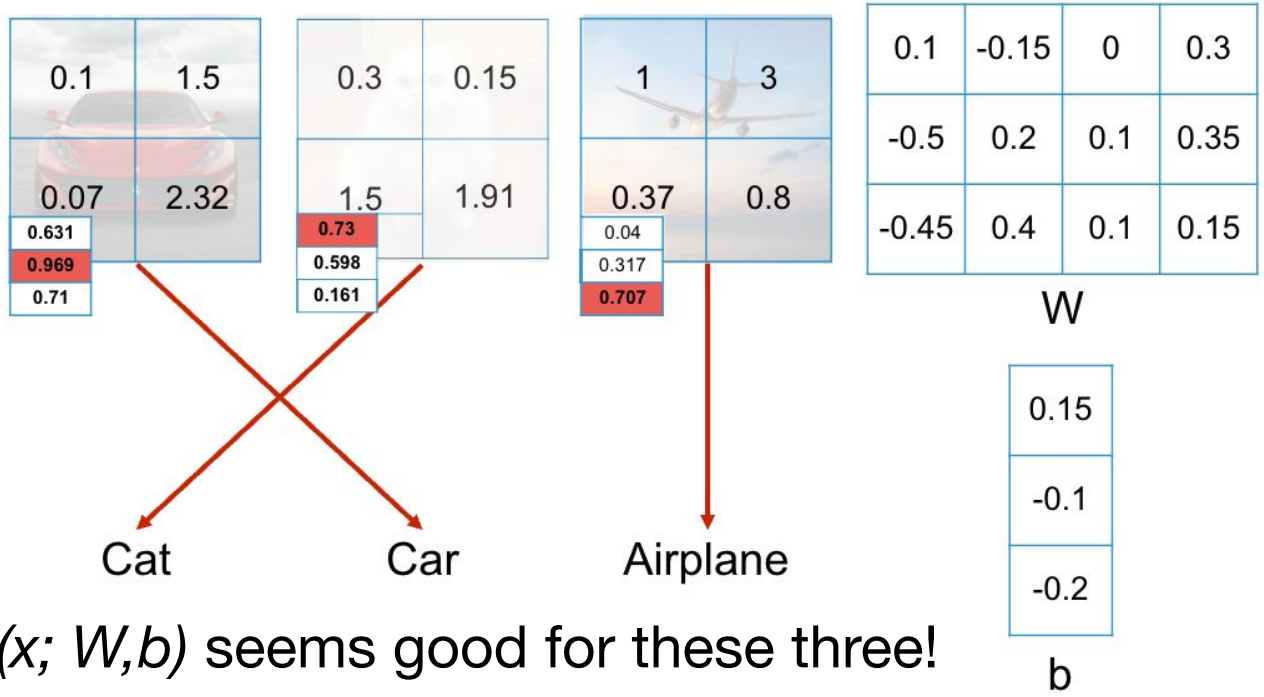
=

0.04	Cat
0.317	Car
0.707	Airplane

f(x; W,b)

Linear Image Classification

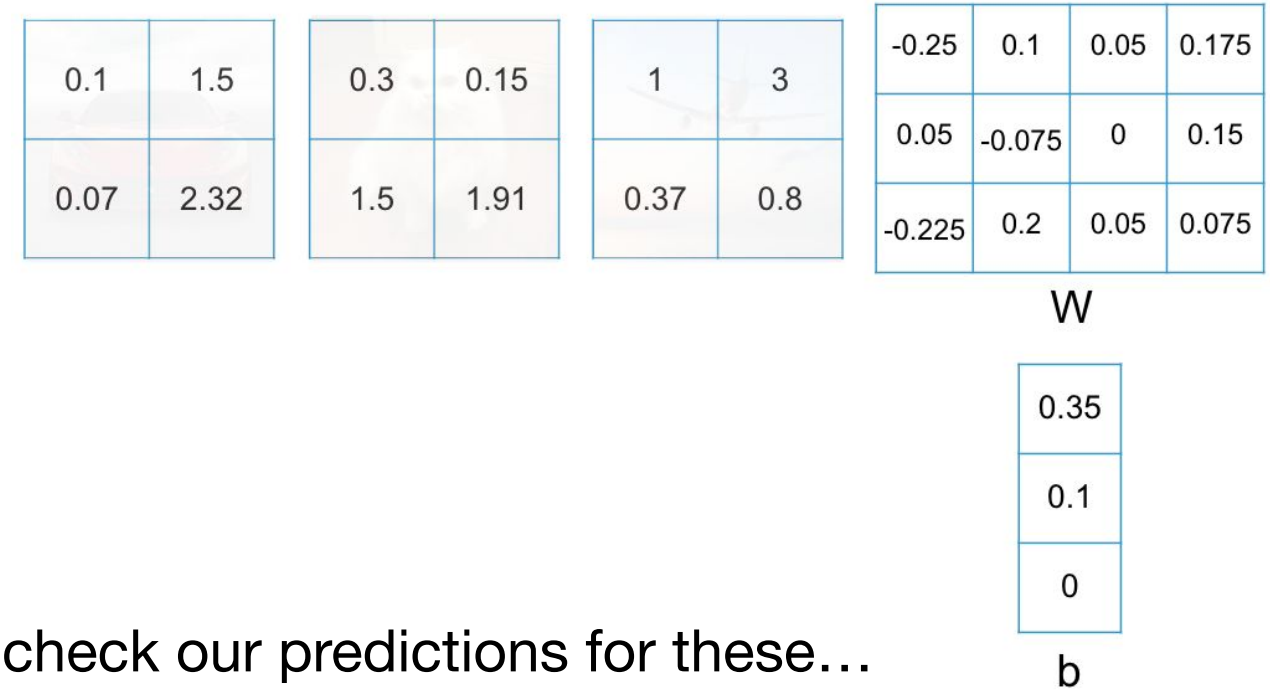
- What does a good *f(x; W,b)* look like?



- This *f(x; W,b)* seems good for these three!

Linear Image Classification

- Same images, new candidate weights *W* and bias vector *b*



- Let’s check our predictions for these…

Linear Image Classification

- Same images, new candidate weights *W* and bias vector *b*

0.1	1.5
0.07	2.32

-0.25	0.1	0.05	0.175
0.05	-0.075	0	0.15
-0.225	0.2	0.05	0.075

W

0.1
1.5
0.07
2.32

x

+

0.35
0.1
0

b

=

0.884	Cat
0.34	Car
0.455	Airplane

f(x; W,b)

Linear Image Classification

- Same images, new candidate weights *W* and bias vector *b*

0.3	0.15
1.5	1.91

-0.25	0.1	0.05	0.175
0.05	-0.075	0	0.15
-0.225	0.2	0.05	0.075

W

0.3
0.15
1.5
1.91

x

+

0.35
0.1
0

b

=

0.699	Cat
0.39	Car
0.181	Airplane

f(x; W,b)

Linear Image Classification

- Same images, new candidate weights *W* and bias vector *b*

1	3
0.37	0.8

-0.25	0.1	0.05	0.175
0.05	-0.075	0	0.15
-0.225	0.2	0.05	0.075

W

1
3
0.37
0.8

x

+

0.35
0.1
0

b

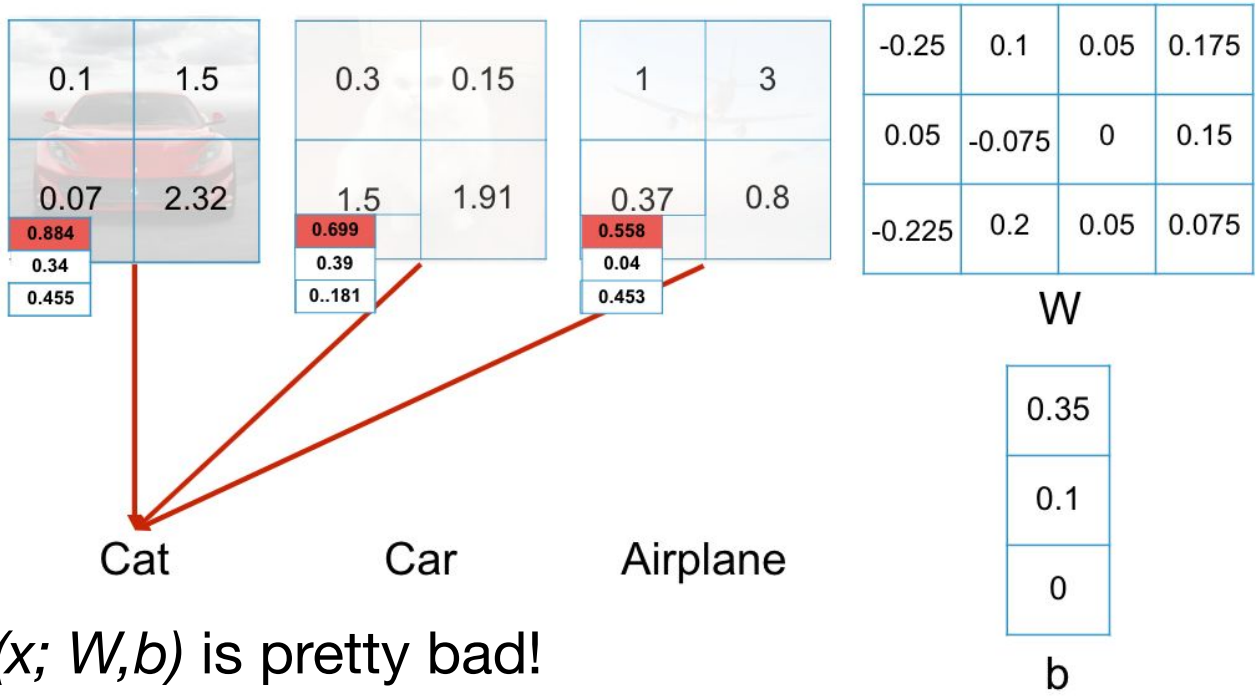
=

0.558	Cat
0.04	Car
0.453	Airplane

f(x; W,b)

Linear Image Classification

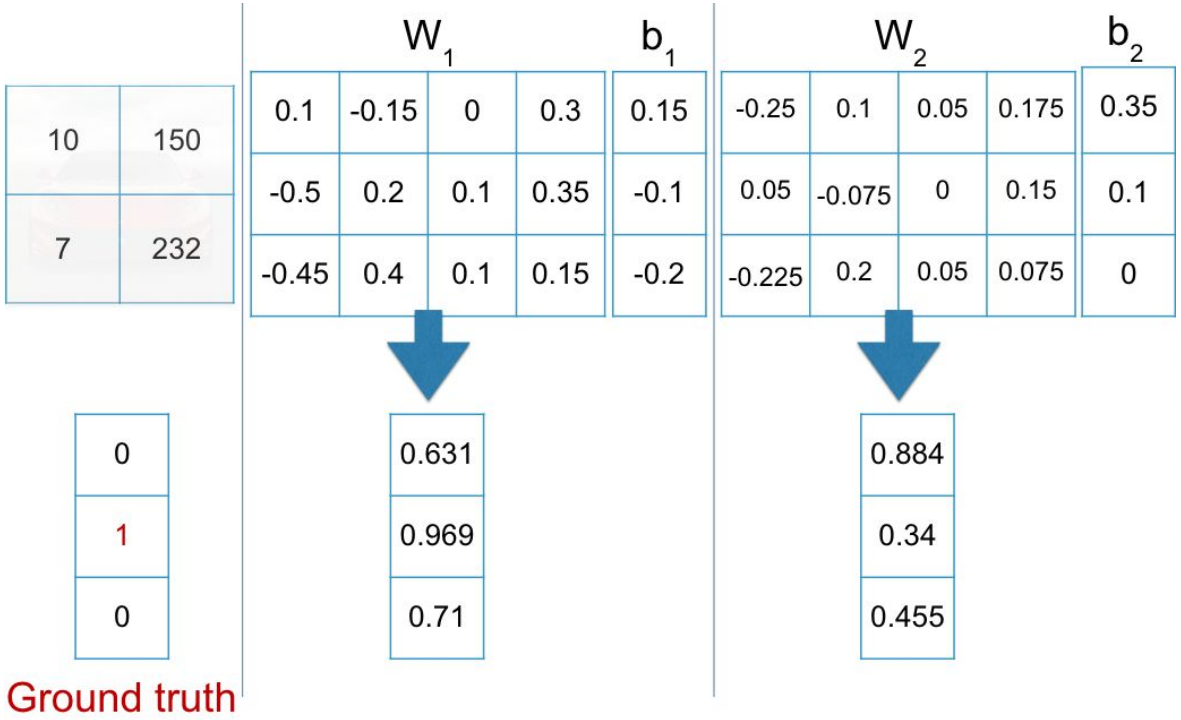
- Same images, new candidate weights *W* and bias vector *b*



- This *f(x; W,b)* is pretty bad!

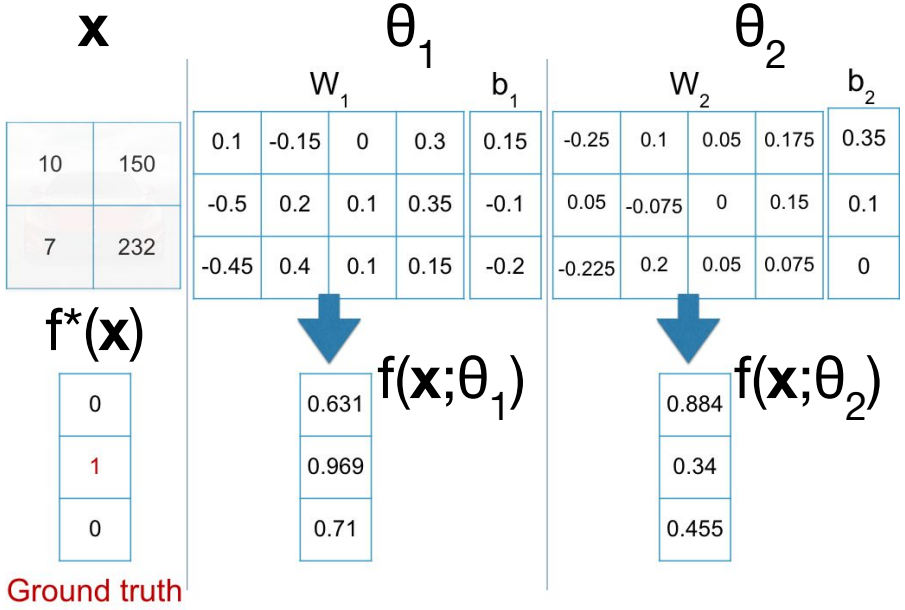
Linear Image Classification

- How do we measure the *fit* of candidate parameters?



How Do We Measure the Quality of $f(\mathbf{x};\theta)$?

- How do we measure the *fit* of candidate parameters?
- Target vector $f^*(\mathbf{x})$ if we knew the exact function, f^*:
  - <0.00, 1.00, 0.00>
- Prediction of $f(\mathbf{x};\theta_1)$:
  - <0.63, 0.97, 0.71>
- Prediction of $f(\mathbf{x};\theta_2)$:
  - <0.88, 0.34, 0.46>
- We want a way to penalize *distance* between $f^*(\mathbf{x})$ and predicted vectors



How Do We Measure the Quality of $f(\boldsymbol{x};\theta)$?

- How do we measure the *fit* of candidate parameters?
- Want to formulate an **optimization problem** to minimize the distance $\{f^*(\boldsymbol{x}) - f(\boldsymbol{x};\theta)\}$ across all $\boldsymbol{x}_i \in X$ in a dataset of (X, Y) pairs
- We call the function we want to minimize, subject to the available training data, the **loss function**
- What are some possible ways to measure such a distance penalty? We want “good” θ to have less loss than “bad” θ

How Do We Measure the Quality of $f(\boldsymbol{x};\theta)$?

- We could just sum up all the distances over the dataset!

$$Loss = \frac{1}{N}\sum_{i=1}^{N} \| \ y_i - f(x_i; W, b)\|_1$$

- This loss function is called **L_1 Loss**
- What could go wrong?
  - If many examples are right, some can be egregiously wrong and the loss will be linearly bounded by those wrong examples only; model could learn to ignore hard pairs

How Do We Measure the Quality of $f(\boldsymbol{x};\theta)$?

- We could add a quadratic penalty

$$Loss = \frac{1}{N}\sum_{i=1}^{N} \| \ y_i - f(x_i; W, b)\|_2^2$$

- This loss function is called **L_2 Loss**
- What could go wrong?
  - The model doesn't have a strong incentive to get *f* "exactly" right; for terms less than 1 the square makes them less severe

How Do We Measure the Quality of $f(\boldsymbol{x};\theta)$?

- We could individually discourage misclassifications

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

- This loss function is called **Hinge Loss**
- For every example $\boldsymbol{x_i}$, the loss is non-zero if $f(\boldsymbol{x_i};\theta)$ assigns a score to a wrong class that is within a *hinge* level of the score assigned to the true class y_i (here, the *hinge penalty* is 1)

How Do We Measure the Quality of $f(\boldsymbol{x};\theta)$?

- We could turn our scores into a probability distribution

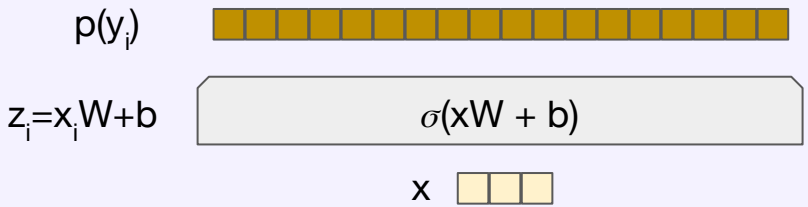
$$Loss = -\frac{1}{N}\sum_{i=1}^{N} \hat{y_i}\, log(P(y_i = j\,|x_i)) \qquad P(y_i = j\,|x_i) = \frac{e^{f(x_i;W,b)}}{\sum_{j=1}^{C} e^{f(x_j;W,b)}}$$

- This loss function is called **Cross-Entropy Loss**
- For every example x_i, the loss approaches zero as the score for true class y_i dominates the sum of all scores, and approaches infinity (slowly!) as score y_i is dominated by other class scores

ML FUNDAMENTALS: Softmax Function

- Sometimes we want a probability distribution over possible outputs from our model
- We can induce a probability distribution from a set of output logits (“activations” / “energies” / “scores”)
- For z_i the logit of class *i* in the prediction space, the softmax probability is given by:

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$



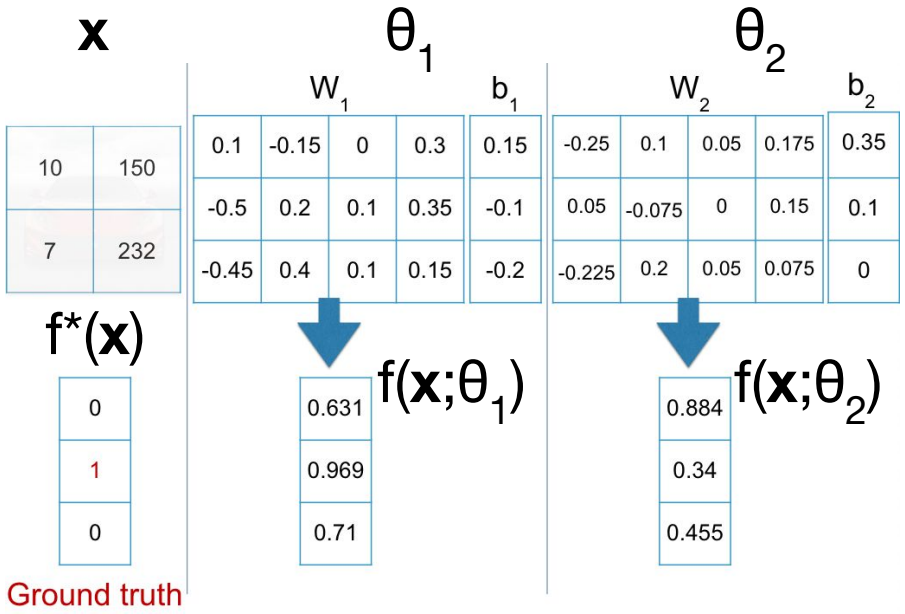
https://en.wikipedia.org/wiki/Softmax_function

How Do We Measure the Quality of $f(\boldsymbol{x};\theta)$?

- How do we measure the *fit* of candidate parameters?
- Want to formulate an **optimization problem** to minimize the distance $\{f^*(\boldsymbol{x}) - f(\boldsymbol{x};\theta)\}$ across all $\boldsymbol{x}_i \in X$ in a dataset of (X, Y) pairs
- We call the function we want to minimize, subject to the available training data, the **loss function**
- Which loss function should you use?
  - It depends!!
  - Loss is a **core component** of problem formulation
  - Input/output representation, model, loss function,

How Do We Measure the Quality of *f(**x**;θ)* ?

- How do we measure the *fit* of candidate parameters?
- We can calculate the **loss** of parameter options θ_1 versus θ_2 across a dataset (X, Y)
- Now that we can measure the *quality* of a set of parameters, we can consider how to *discover* those parameters



How Do We Find Good Parameters θ for $f(\boldsymbol{x};\theta)$?

- Backing up:
  - $f(\boldsymbol{x};\theta)$ is our notation for any general function we'd like to learn
  - $f(\boldsymbol{x};\theta)$ maps from $x \in X$ to $y \in Y$
  - Vector $\boldsymbol{x}$ is obtained by applying representation function $\phi(x)=\boldsymbol{x}$ to input
  - θ parameterizes f; f is a class of functions that can vary according to the selection of θ
  - For example, $\theta=\{W,b\}$ for $f(\boldsymbol{x};\theta)=W\boldsymbol{x}+b$

How Do We Find Good Parameters θ for $f(\mathbf{x};\theta)$?

- $f(\mathbf{x};\theta)=W\mathbf{x}+b$; $\theta=\{W,b\}$
- In our running image classification example, how do we *define* what are good parameters?
- We pick a loss function L!
  - $\theta^{*}=\min_{\theta\in\Theta}(L(f(x;\theta), y))$
- L is calculated on an (x, y) basis, so we really want to find good parameters for an entire *dataset*, not just one example!
  - $\theta^{*}=\min_{\theta\in\Theta}(\Sigma_{(x,y)\in D}L(f(x;\theta), y)))$
  - For $D=\{(x_1,y_1), (x_2,y_2), \ldots, (x_n,y_n)\}$ our *training dataset*

How Do We Find Good Parameters θ for $f(\mathbf{x};\theta)$?

- $f(\mathbf{x};\theta)=W\mathbf{x}+b$; $\theta=\{W,b\}$
- $\theta^{*}=min_{\theta \in \Theta}(\Sigma_{(x,y) \in D} L(f(x;\theta), y)))$
  - For $D=\{(x_1,y_1), (x_2,y_2), \ldots, (x_n,y_n)\}$ our *training dataset*
- Three basic classes of approach:
  - Analytical solution
    - Find where $^{d}/_{d\theta}(\Sigma_{(x,y) \in D} L(f(x;\theta), y))) = 0$
  - Heuristic search
    - Initialize random candidate θ's and check L value
  - Numerical approach
    - Estimate better θ's given current ones

How Do We Find Good Parameters θ for $f(\mathbf{x};\theta)$?

- $f(\mathbf{x};\theta)=W\mathbf{x}+b$; $\theta=\{W,b\}$
- $\theta^{*}=min_{\theta \in \Theta}(\Sigma_{(x,y) \in D} L(f(x;\theta), y)))$
  - For $D=\{(x_1,y_1), (x_2,y_2), \ldots, (x_n,y_n)\}$ our *training dataset*
- Numerical approach
  - Estimate better θ's given current ones
- Best of both worlds: cheap (heuristic) + informed (analytical)
  - Finding where $^{d}/_{d\theta}(\Sigma_{(x,y) \in D} L(f(x;\theta), y))) = 0$ is hard
  - Finding good θ through random/population search is hard
- Calculating $^{d}/_{d\theta i}(\Sigma_{(x,y) \in D} L(f(x;\theta), y)))$ for dimension θ_i is easy…

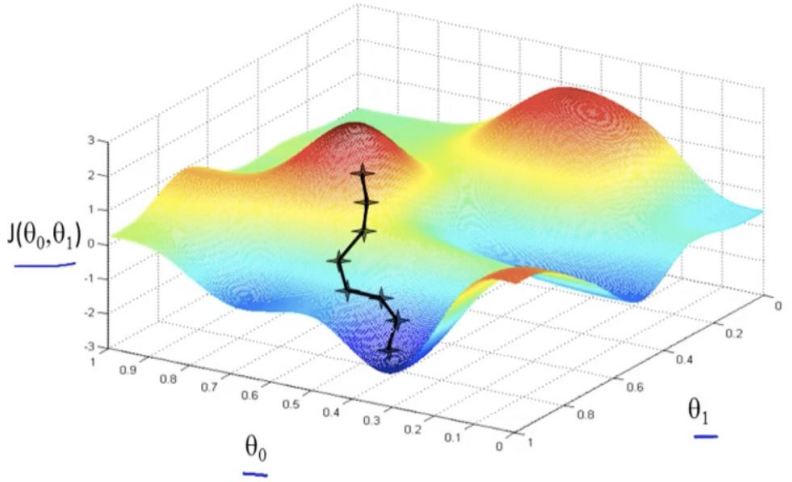
Gradient Descent

0.1	1.5
0.07	2.32

x

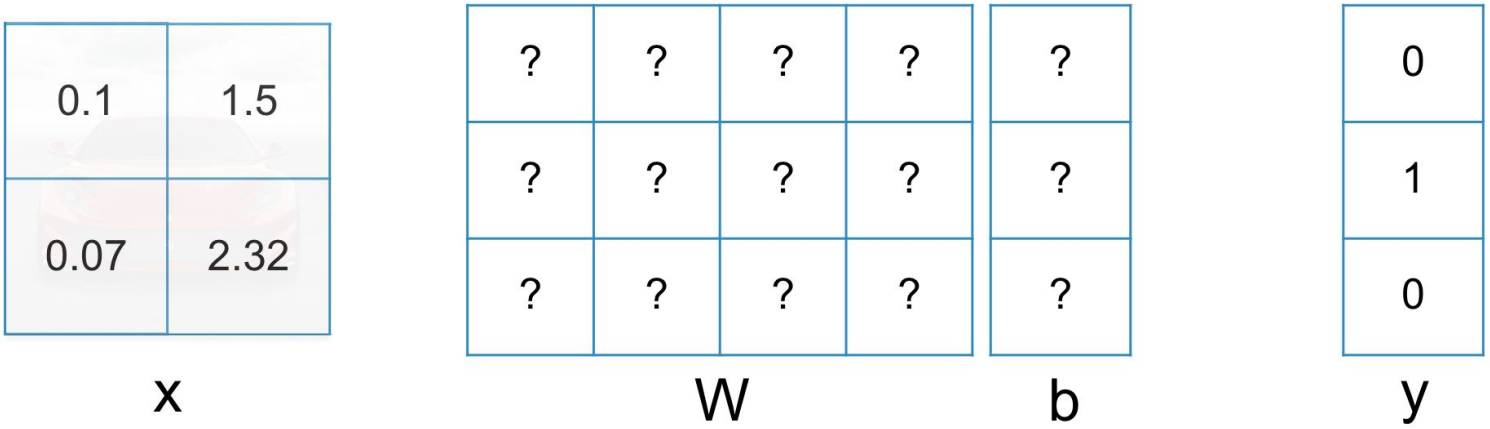
?	?
?	?
?	?

W



- We want θ *m*inimizing $\Sigma_{(x,y) \in D} L(f(x;\theta), y))$
- Calculating $^{d}/_{d\theta i} (\Sigma_{(x,y) \in D} L(f(x;\theta), y)))$ for dimension θ_i is easy
- For a candidate θ, we can calculate the *gradient* in each dimension of θ_i that could lead us towards the minimum

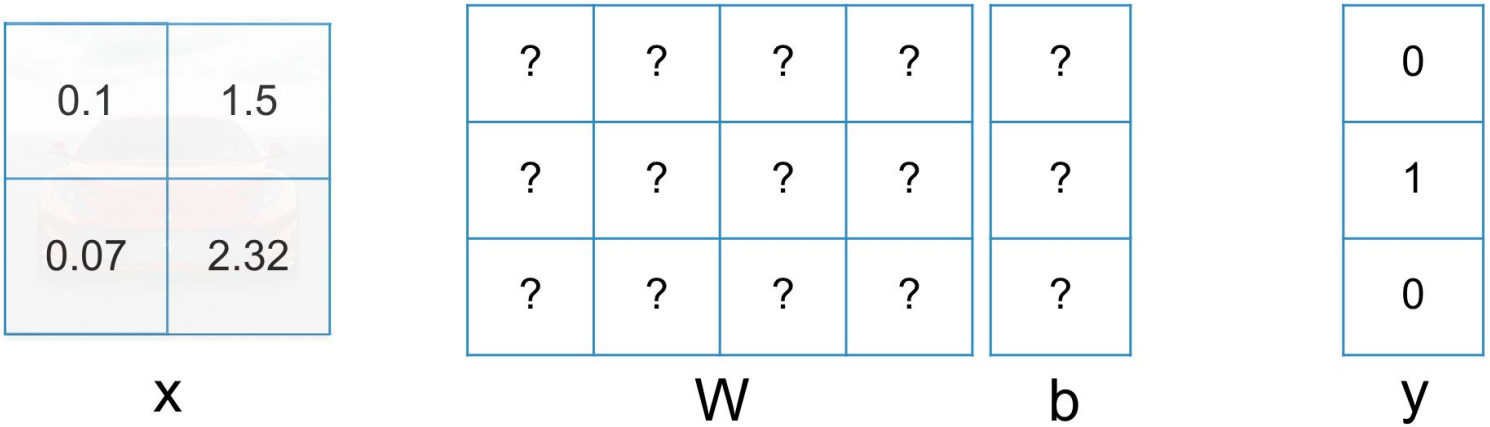
Gradient Descent



- Let's consider just $W \in \theta$ for the above (x,y) datum
- Recall the definition of a derivative for a generic function f

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Gradient Descent



$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

- We will take the derivative of *L(f(x;θ), y)* with respect to each individual dimension of *θ*
- That *vector* of partial derivatives is called the **gradient** wrt *(x,y)*

Gradient Descent

-0.1	0.2	0.15	0.05
0.25	0.05	0.2	-0.1
0.25	0.15	-0.15	0.4

W

-0.1+ 0.001	0.2	0.15	0.05
0.25	0.05	0.2	-0.1
0.25	0.15	-0.15	0.4

W+h

?	?	?	?
?	?	?	?
?	?	?	?

gradient (dW)

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

- For $L(f(x;\theta_1), y)$, we calculate L before and after adding a tiny perturbation h to θ_1, giving $f(x+h)$ verus original $f(x)$
- (Remember, here f is our loss L and derivative is against θ_1)

Gradient Descent

-0.1	0.2	0.15	0.05
0.25	0.05	0.2	-0.1
0.25	0.15	-0.15	0.4

W

-0.1+ 0.001	0.2	0.15	0.05
0.25	0.05	0.2	-0.1
0.25	0.15	-0.15	0.4

W+h

?	?	?	?
?	?	?	?
?	?	?	?

gradient (dW)

L(W) = 1.6688

L(W+h) = 1.6693

dW_1=(1.6693 - 1.6688) / 0.001

= 0.5

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Gradient Descent

-0.1	0.2	0.15	0.05
0.25	0.05	0.2	-0.1
0.25	0.15	-0.15	0.4

W

-0.1+ 0.001	0.2	0.15	0.05
0.25	0.05	0.2	-0.1
0.25	0.15	-0.15	0.4

W+h

0.5	?	?	?
?	?	?	?
?	?	?	?

gradient (dW)

L(W) = 1.6688

L(W+h) = 1.6693

dW_1=(1.6693 - 1.6688) / 0.001

= 0.5

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Gradient Descent

-0.1	0.2	0.15	0.05
0.25	0.05	0.2	-0.1
0.25	0.15	-0.15	0.4

W

L(W) = 1.6688

-0.1+ 0.001	**0.2+ 0.001**	0.15	0.05
0.25	0.05	0.2	-0.1
0.25	0.15	-0.15	0.4

W+h

L(W+h) = 1.6697

0.5	**0.9**	?	?
?	?	?	?
?	?	?	?

gradient (dW)

dW_2=(1.6697 - 1.6688) / 0.001

= 0.9

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

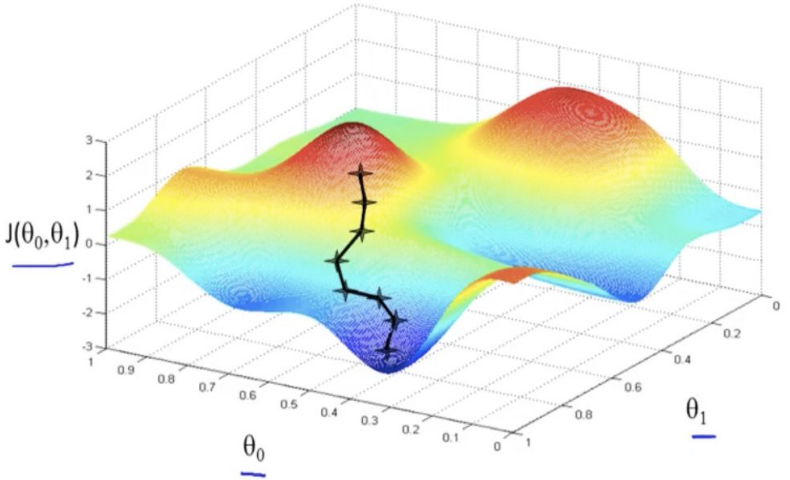
Gradient Descent

-0.1	0.2	0.15	0.05
0.25	0.05	0.2	-0.1
0.25	0.15	-0.15	0.4

W

0.5	0.9
-0.13	0.17
-0.55	.92

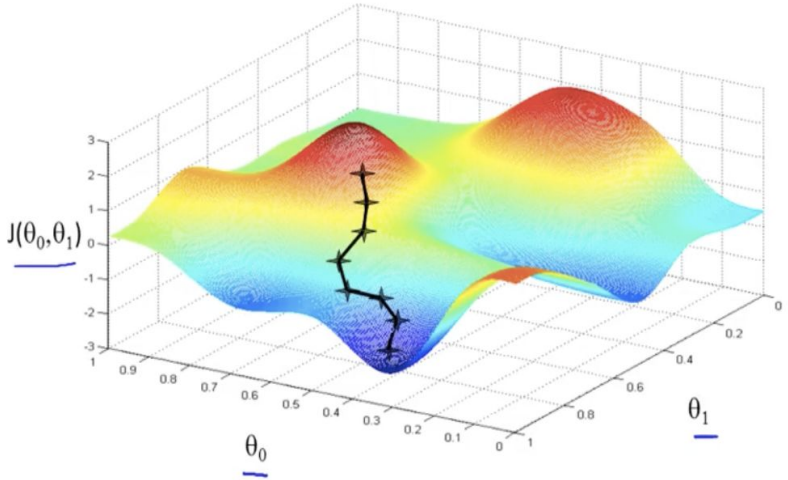
gradient (dW)



W + dW

- Gradient update step: add estimated gradient to existing parameters to take a “step” towards parameter values that are closer to minimizing the objective function
- What’s missing?

Gradient Descent

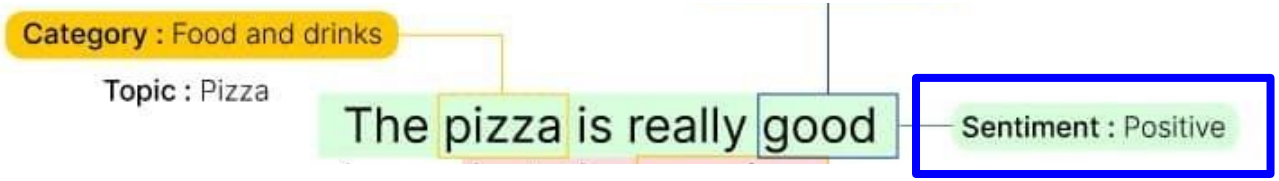


- $f(\mathbf{x};\theta)=W\mathbf{x}+b$; $\theta=\{W,b\}$
- $\theta^{*}=min_{\theta \in \Theta}(\Sigma_{(x,y) \in D} L(f(x;\theta), y)))$
- In our example, we calculated the gradient with respect to just one *(x,y)* pair; $\nabla_{(x,y)}$
- We actually want the gradient for $\Sigma_{(x,y) \in D} L(f(x;\theta), y))$, which requires repeating our process for *every* $(x,y) \in D$
- $\nabla_{D}=(1/|D|)\Sigma_{(x,y) \in D} \nabla_{(x,y)}$
- What's the problem?

Stochastic Gradient Descent

- Storing $\nabla_D = (1/|D|) \Sigma_{(x,y) \in D} \nabla_{(x,y)}$ is expensive
- Depending on the size of your dataset *D*, it's likely not possible to store the partial derivatives for every *(x,y)* for every parameter in your entire model in the GPU at once
- In *stochastic* gradient descent, we take just a slice (*batch*) of *D* at a time; in practice, as much as we can fit on the GPU
- What sounds best?
  - Randomly sample batch from D and repeat
  - Shuffle D, sample slices as batches, repeat
  - Shuffle D, sample slices as batches, reshuffle D and repeat

Where Does Deep Learning Come In?



- Input space *X*?
  - Review *tokens T*
  - $X=T^N$ for max length *N*
- Output space *Y*?
  - Possible sentiments *S*
  - *Y*=*S*
- Linear function: $f(\boldsymbol{x},\theta)=W\boldsymbol{x}+b$
- What loss function could we use for this *n*-ary classification problem?
  - Cross entropy
- Non-linear transformation:

$$Loss = -\frac{1}{N}\sum_{i=1}^{N} \hat{y_i}\, log(P(y_i = j\,|x_i)) \qquad P(y_i = j\,|x_i) = \frac{e^{f(x_i;W,b)}}{\sum_{j=1}^{C} e^{f(x_j;W,b)}}$$

Where Does Deep Learning Come In?

- Linear function: $f(\boldsymbol{x},\theta)=W\boldsymbol{x}+b$
- What loss function could we use for this *n*-ary classification problem?
  - Cross entropy
- Non-linear transformation:

$$Loss = -\frac{1}{N}\sum_{i=1}^{N} \hat{y_i}\, log(P(y_i = j\,|x_i)) \qquad P(y_i = j\,|x_i) = \frac{e^{f(x_i;W,b)}}{\sum_{j=1}^{C} e^{f(x_j;W,b)}}$$

- Our function can only capture the degree to which each dimension of the input, $\boldsymbol{x_i}$, contributes to the likelihood of each class $\boldsymbol{y_j}$
  - “... brought out the ice cold food.”
  - “... brought out the ice cold gazpacho.”
  - “... Amazing service all around. Yeah right! 🙃”

Where Does Deep Learning Come In?

- $f(\mathbf{x},\theta)=\sigma(W\mathbf{x}+b)$
  - Linear function with nor
- If we need to make decision
- $f(\mathbf{x},\theta)=\sigma(W_2(\sigma(W_1\mathbf{x}+b_1))+b_2)$
  - First layer *latent variables* $\sigma(W_1\mathbf{x}+b_1)$ are each activated by linear combination of values of $\boldsymbol{x}_i$ input variables
  - Now W_2, b_2 can mix signals from the first *layer*; the final outputs $\sigma(W_2(\sigma(W_1\mathbf{x}+b_1))+b_2)$ are each activated by linear combinations of the latent variables from layer one
  - So our class logits can now consider combinations of input words like "**cold gazpacho**" and "**Amazing … Yeah right**!"

Quick terminology note, these basic transformations are called *perceptron layers* / *fully-connected layers*

Where Does Deep Learning Come In?

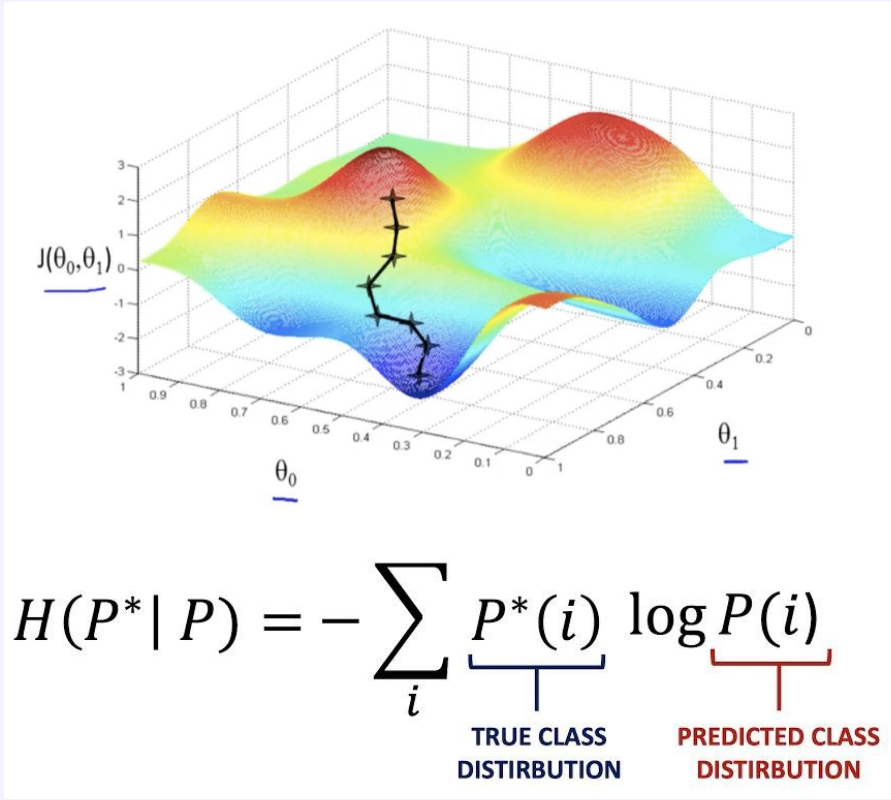
- $f(\mathbf{x},\theta)=\sigma(W_2(\sigma(W_1\mathbf{x}+b_1))+b_2)$
  - This derivative is more computationally intense to take, since we need it with respect to every parameter in both weight matrices and bias vectors
  - Modern DL software handles these computations automatically through *autograd* software for efficient derivative estimation
  - So what's stopping us from….
- $f(\mathbf{x},\theta)=\sigma(W_4(\sigma(W_3\sigma(W_2(\sigma(W_1\mathbf{x}+b_1))+b_2)+b_3))+b_4)$
- $f(\mathbf{x},\theta)=\sigma(W_8(\sigma(W_7\sigma(W_6(\sigma(W_5\sigma(W_4(\sigma(W_3\sigma(W_2(\sigma(W_1\mathbf{x}+b_1))+b_2)+b_3))+b_4)+b_5))+b_6)+b_7))+b_8)$
- Nothing!?
  - Well, realistically, our GPU memory and data constraints
  - For fixed *D*, quality of estimated parameters θ as $|\theta|$ grows?

ML FUNDAMENTALS: Neural Network Training

- A neural network is, commonly, a composition of matrix multiplications and non-linear transformations
- A neural network can perform *non-linear function approximation* through repeated linear layers and non-linear transformations
- Optimization: $M(\phi(x), \theta) = y$ for pairs (x, y) in data by making adjustments to θ
- Optimizers in modern NN packages are variations on the basics: **Stochastic Gradient Descent**

ML FUNDAMENTALS: Stochastic Gradient Descent

- Gradient Descent
- "Step" $\varepsilon \nabla L[M(\phi(x), \theta)]$
  - Go a little bit in the direction of the gradient (derivative) towards a local minima in the *loss*
- "Loss": measure of how poorly the model did, e.g., the cross-entropy between predicted and true classes



ML FUNDAMENTALS: Stochastic Gradient Descent

- "Step" $\varepsilon \nabla L[M(\phi(x), \theta)]$
- What's ε?
  - Learning rate
- Can we actually get ∇ *over all of D*?
  - Sometimes! Usually "no", and so we use *stochastic* GD
- Could estimate ∇_x with a single sample *(x,y)*. Why not?
- Could estimate $\nabla_{xi \ldots xj}$ with, e.g., as many examples as we can fit in RAM at a time $x_i \ldots x_j$. What do we call these?
- Each subset of *D* used to estimate gradient is a **mini-batch**

ML FUNDAMENTALS: Stochastic Gradient Descent

- "Step" $\varepsilon \nabla_{xi \ldots xj} L[M(\phi(x), \theta)]$
- **Learning rate**: ε
- **Mini-batch**: $(x,y)_i \ldots (x,y)_j$
- **Iteration**: $\theta := \theta - \varepsilon \nabla_{xi \ldots xj} L[M(\phi(x), \theta)]$
  - Move parameters in direction of gradient estimated for a mini-batch
- **Epoch**: Completed each time we've seen all of $(x,y) \in D$ again

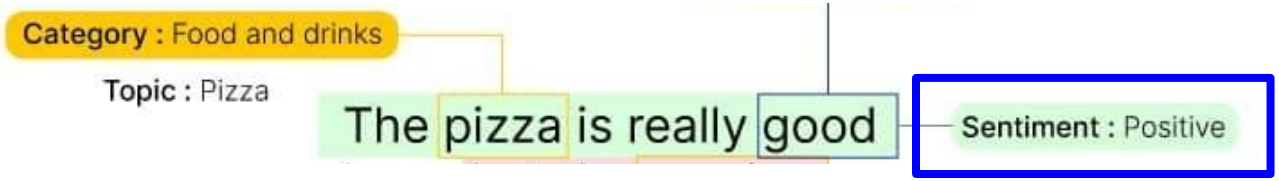
ML FUNDAMENTALS: Neural Network Training

- Optimizers in modern NN packages are variations on the basics: **Stochastic Gradient Descent**
- What do we need?
  - Pass over data for multiple **epochs**
  - Sample **mini-batches** of our data
  - Step the optimization algorithm to complete an **iteration**
- Let's step through a motivating example to see how we arrive at training deep neural networks for function approximation

Overview of Today's Plan

- ~~Course organization and deliverables recap~~
- ~~Framing Problems for Machine Learning~~
- ~~Loss Functions and Optimization~~
    - Any questions before we move on?
- Representations and Models

Representations and Models

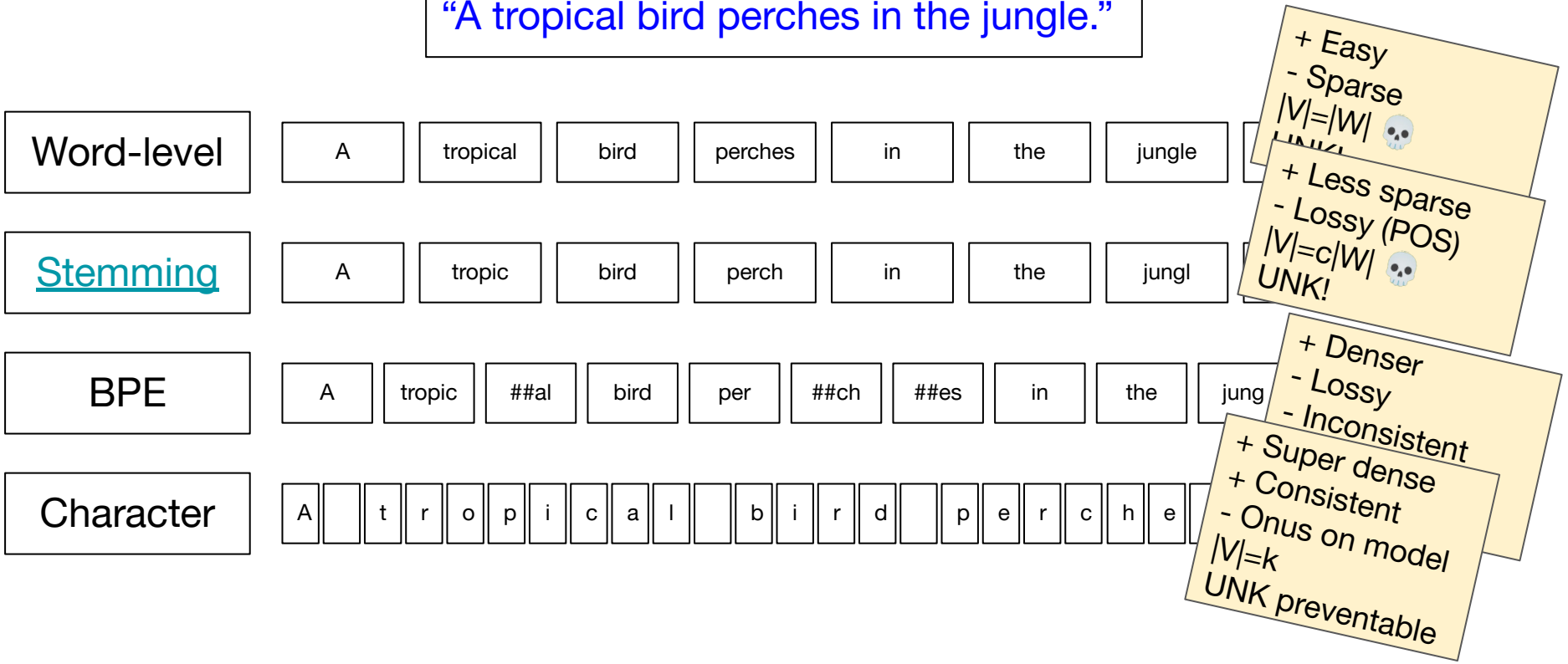


- Input space X?
  - Review *tokens* T
  - $X=T^N$ for max length N
- Output space Y?
  - Possible sentiments S
  - $Y=S$

- Each $x \in X$ is a sequence of *tokens* that together form the entire review
- For example:
  - x=<The, pizza, is, really, good>
- We can't feed that sequence/list into equation $Wx+b$
- We need a *vector representation* of x, $\phi(x)=\boldsymbol{x}$
- The final consideration to frame problems for DL is representation ϕ

Vector Representations for Language - Tokenization

"A tropical bird perches in the jungle."



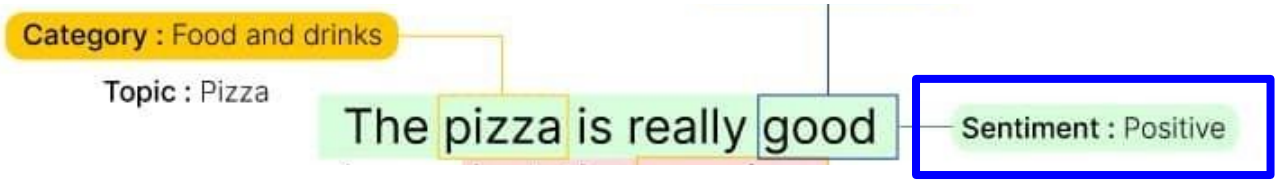
Vector Representations for Language

- But, again, we can't feed x=<The, pizza, is, really, good> into a model; we need to represent each individual token as a vector
- What are some options we have?
  - Bag of words vectors: take the top *K* most common words and count how frequently they happen near word *w*; the vector is <*K*> occurrence averages across a corpus
  - *Learned* word embedding: initialize d-dimensional vector for each word *w* randomly and use *backpropagation* during task training to fine-tune those layers as well.

Representations and Models

- The nature of the input, output, and intuitive relation between them gives us a lot of information as DL practitioners about what kind of *model* we should use
- What kind of model might you use for language inputs?
  - Perceptron? Why or why not?
  - Recurrent models (RNN, LSTM, GRU)? Why?
  - Transformers? Why?
- What kind of model might you use for vision inputs?
  - Perceptron? Why or why not?
  - Convolutional models (CNN, ResNets)? Why?

Representations and Models

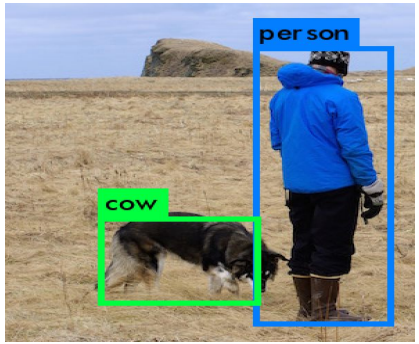


- Input space *X*?
  - Review *tokens T*
  - $X=T^N$
- Representation?
  - $\phi_x(X)$ word embeddings

- Output space *Y*?
  - Possible sentiments *S*
  - *Y=S*
- Representation?
  - $\phi_y(Y)$ = one-hot vectors of length $|S|$

- What model architecture might you use?
- What loss function would you want to optimize?

Representations and Models



- Input space *X*?
  - Pixels in the image
- Representation?
  - $\phi_x(X)=\boldsymbol{R}^{W,H,3}$ RGB or HSV

- Output space *Y*?
  - Object classes *C* + bboxes
- Representation?
  - $\phi_y(Y) = C \times \boldsymbol{R}^{x,y,w,h} \times N$

- What model architecture might you use?
- What loss function would you want to optimize?

Representations and Models

- Input space *X*?
  - Pixels in the image
- Representation?
  - $\phi_x(X)=\boldsymbol{R}^{W,H,3}$ RGB or HSV

- Output space *Y*?
  - Object classes |C| per pixel in the image
- Representation?
  - $\phi_y(Y) = \boldsymbol{R}^{W,H,|C|}$

- What model architecture might you use?
- What loss function would you want to optimize?

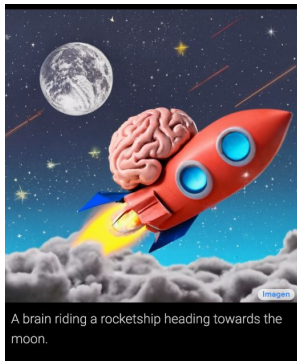
Representations and Models

Who is wearing glasses?

man

- Input space *X*?
  - Pixels in the image
  - Tokens in the question
- Representation?
  - $\phi_x(X)=\mathbf{R}^{W,H,3} \times V^N$
  - (V: word embedding space)
- Output space *Y*?
  - Possible answers Y=A
- Representation?
  - $\phi_y(Y)$ = one-hot vectors of size $|A|$
- What model architecture might you use?
- What loss function would you want to optimize?

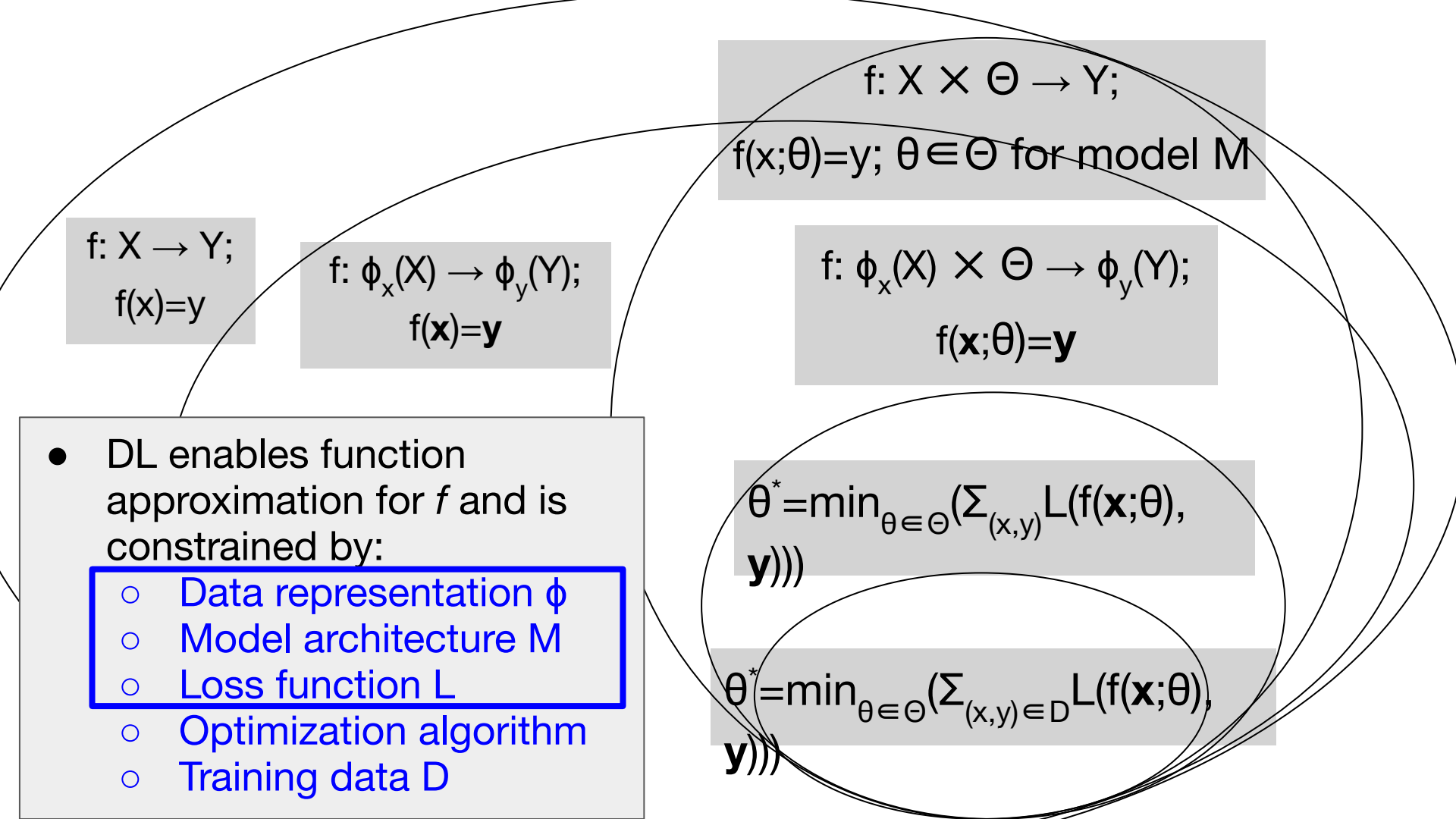
Representations and Models



- Input space *X*?
  - Tokens in the caption
- Representation?
  - $\phi_x(X)$ word embeddings

- Output space *Y*?
  - Pixels of an image
- Representation?
  - $\phi_y(Y) = \boldsymbol{R}^{W,H,3}$

- What model architecture might you use?
- What loss function would you want to optimize?


f: X → Y;
f(x)=y
f: $\phi_x(X) \rightarrow \phi_y(Y)$;
f(**x**)=**y**
f: X × Θ → Y;
f(x;θ)=y; θ∈Θ for model M
f: $\phi_x(X) \times \Theta \rightarrow \phi_y(Y)$;
f(**x**;θ)=**y**
$\theta^* = \min_{\theta \in \Theta}(\Sigma_{(x,y)} L(f(\mathbf{x};\theta), \mathbf{y}))$
$\theta^* = \min_{\theta \in \Theta}(\Sigma_{(x,y) \in D} L(f(\mathbf{x};\theta), \mathbf{y}))$
• DL enables function approximation for *f* and is constrained by:
○ Data representation ϕ
○ Model architecture M
○ Loss function L
○ Optimization algorithm
○ Training data D

Framing Problems for Machine Learning

- If you can:
  - Squeeze your input and output into numerical representations (e.g., vectors)
  - Compose neural layers that eat something shaped like the input and spit out something shaped like the output
  - Define a loss function and apply an optimizer that are valid in the underlying problem and model space
- Then:
  - Your DL model will train and loss will go down, *even if you aren't actually solving the problem you wanted to solve*

Overview of Today's Plan

- ~~Course organization and deliverables recap~~
- ~~Framing Problems for Machine Learning~~
- ~~Loss Functions and Optimization~~
- ~~Representations and Models~~

Action Items for You

- If you have gotten D-clearance but have not taken Quiz 0, please take it; it is just a survey for us to get a sense of the student body, but it is worth points!
- Form project teams; these must be finalized in two weeks
- Start brainstorming project topics; take a look at the deck of project ideas to get an idea what info you'll need if your team scopes a novel proposal
- Remember there is **no class next Friday**; you are encouraged to attend Dave Held's invited talk in RTH 115, 11am-12pm

CSCI 566: Deep Learning and Its Applications

Jesse Thomason

Lecture 2: DL and Machine Learning