CS395T - DEEP LEARNING SEMINAR

Philipp Krähenbühl

OVERVIEW

Philipp Krähenbühl
office GDC 4.824
office hours by
appointment (send email)

TA Huihuang Zheng
office GDC 6.802
office hours We 10-11am

Try piazza first!

OVERVIEW

https://philkr.github.io/CS395T/

OVERVIEW

before class

Read and review
two papers

is covariate shift
important?

how about this baseline?

in class

10 min intro (Philipp)

10-15 min paper 1
(one of you)

25 min paper 1
(all of you)

why not …?

10-15 min paper 2
(one of you)

25 min paper 2
(all of you)

Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift

Sergey Ioffe
Google Inc., sioffe@google.com

Christian Szegedy
Google Inc., szegedy@google.com

Abstract

Training Deep Neural Networks is complicated by the fact that the distribution of each layer's inputs changes during training, as the parameters of the previous layers change. This slows down the training by requiring lower learning rates and careful parameter initialization, and makes it notoriously hard to train models with saturating nonlinearities. We refer to this phenomenon as *internal covariate shift*, and address the problem by normalizing layer inputs. Our method draws its strength from making normalization a part of the model architecture and performing the normalization *for each training mini-batch*. Batch Normalization allows us to use much higher learning rates and be less careful about initialization. It also acts as a regularizer, in some cases eliminating the need for Dropout. Applied to a state-of-the-art image classification model, Batch Normalization achieves the same accuracy with 14 times fewer training steps, and beats the original model by a significant margin. Using an ensemble of batch-normalized networks, we improve upon the best published result on ImageNet classification: reaching 4.9% top-5 validation error (and 4.8% test error), exceeding the accuracy of human raters.

1 Introduction

Deep learning has dramatically advanced the state of the art in vision, speech, and many other areas. Stochastic gradient descent (SGD) has proved to be an effective way of training deep networks, and SGD variants such as momentum (Sutskever et al., 2013) and Adagrad (Duchi et al., 2011) have been used to achieve state of the art performance. SGD optimizes the parameters Θ of the network, so as to minimize the loss

$$\Theta = \arg\min_{\Theta} \frac{1}{N} \sum_{i=1}^{N} \ell(x_i, \Theta)$$

where $x_{1\ldots N}$ is the training data set. With SGD, the training proceeds in steps, and at each step we consider a *mini-batch* $x_{1\ldots m}$ of size m. The mini-batch is used to approximate the gradient of the loss function with respect to the parameters, by computing

$$\frac{1}{m} \frac{\partial \ell(x_i, \Theta)}{\partial \Theta}.$$

Using mini-batches of examples, as opposed to one example at a time, is helpful in several ways. First, the gradient of the loss over a mini-batch is an estimate of the gradient over the training set, whose quality improves as the batch size increases. Second, computation over a batch can be much more efficient than m computations for individual examples, due to the parallelism afforded by the modern computing platforms.

While stochastic gradient is simple and effective, it requires careful tuning of the model hyper-parameters, specifically the learning rate used in optimization, as well as the initial values for the model parameters. The training is complicated by the fact that the inputs to each layer are affected by the parameters of all preceding layers – so that small changes to the network parameters amplify as the network becomes deeper.

The change in the distributions of layers' inputs presents a problem because the layers need to continuously adapt to the new distribution. When the input distribution to a learning system changes, it is said to experience *covariate shift* (Shimodaira, 2000). This is typically handled via domain adaptation (Jiang, 2008). However, the notion of covariate shift can be extended beyond the learning system as a whole, to apply to its parts, such as a sub-network or a layer. Consider a network computing

$$\ell = F_2(F_1(u, \Theta_1), \Theta_2)$$

where F_1 and F_2 are arbitrary transformations, and the parameters Θ_1, Θ_2 are to be learned so as to minimize the loss ℓ. Learning Θ_2 can be viewed as if the inputs $x = F_1(u, \Theta_1)$ are fed into the sub-network

$$\ell = F_2(x, \Theta_2).$$

For example, a gradient descent step

$$\Theta_2 \leftarrow \Theta_2 - \frac{\alpha}{m} \sum_{i=1}^{m} \frac{\partial F_2(x_i, \Theta_2)}{\partial \Theta_2}$$

(for batch size m and learning rate α) is exactly equivalent to that for a stand-alone network F_2 with input x. Therefore, the input distribution properties that make training more efficient – such as having the same distribution between the training and test data – apply to training the sub-network as well. As such it is advantageous for the distribution of x to remain fixed over time. Then, Θ_2 does

Journal of Machine Learning Research 15 (2014) 1929-1958 Submitted 11/13; Published 6/14

Dropout: A Simple Way to Prevent Neural Networks from Overfitting

Nitish Srivastava NITISH@CS.TORONTO.EDU
Geoffrey Hinton HINTON@CS.TORONTO.EDU
Alex Krizhevsky KRIZ@CS.TORONTO.EDU
Ilya Sutskever ILYA@CS.TORONTO.EDU
Ruslan Salakhutdinov RSALAKHU@CS.TORONTO.EDU
Department of Computer Science
University of Toronto
10 Kings College Road, Rm 3302
Toronto, Ontario, M5S 3G4, Canada.

Editor: Yoshua Bengio

Abstract

Deep neural nets with a large number of parameters are very powerful machine learning systems. However, overfitting is a serious problem in such networks. Large networks are also slow to use, making it difficult to deal with overfitting by combining the predictions of many different large neural nets at test time. Dropout is a technique for addressing this problem. The key idea is to randomly drop units (along with their connections) from the neural network during training. This prevents units from co-adapting too much. During training, dropout samples from an exponential number of different "thinned" networks. At test time, it is easy to approximate the effect of averaging the predictions of all these thinned networks by simply using a single unthinned network that has smaller weights. This significantly reduces overfitting and gives major improvements over other regularization methods. We show that dropout improves the performance of neural networks on supervised learning tasks in vision, speech recognition, document classification and computational biology, obtaining state-of-the-art results on many benchmark data sets.

Keywords: neural networks, regularization, model combination, deep learning

1. Introduction

Deep neural networks contain multiple non-linear hidden layers and this makes them very expressive models that can learn very complicated relationships between their inputs and outputs. With limited training data, however, many of these complicated relationships will be the result of sampling noise, so they will exist in the training set but not in real test data even if it is drawn from the same distribution. This leads to overfitting and many methods have been developed for reducing it. These include stopping the training as soon as performance on a validation set starts to get worse, introducing weight penalties of various kinds such as L1 and L2 regularization and soft weight sharing (Nowlan and Hinton, 1992).

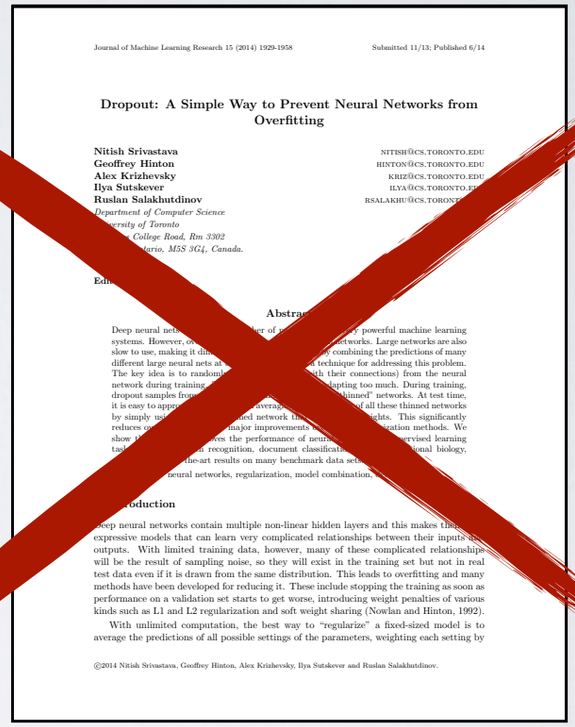
With unlimited computation, the best way to "regularize" a fixed-sized model is to average the predictions of all possible settings of the parameters, weighting each setting by

PRESENTATIONS

visual

no incredibly long walls of text or math than nobody can understand, follow or otherwise parse.

show presentation to Philipp 1 week ahead



no scrolling through paper

OVERVIEW

- two deep learning projects
- train a deep network
- write a latex report
- present your work

PROJECTS

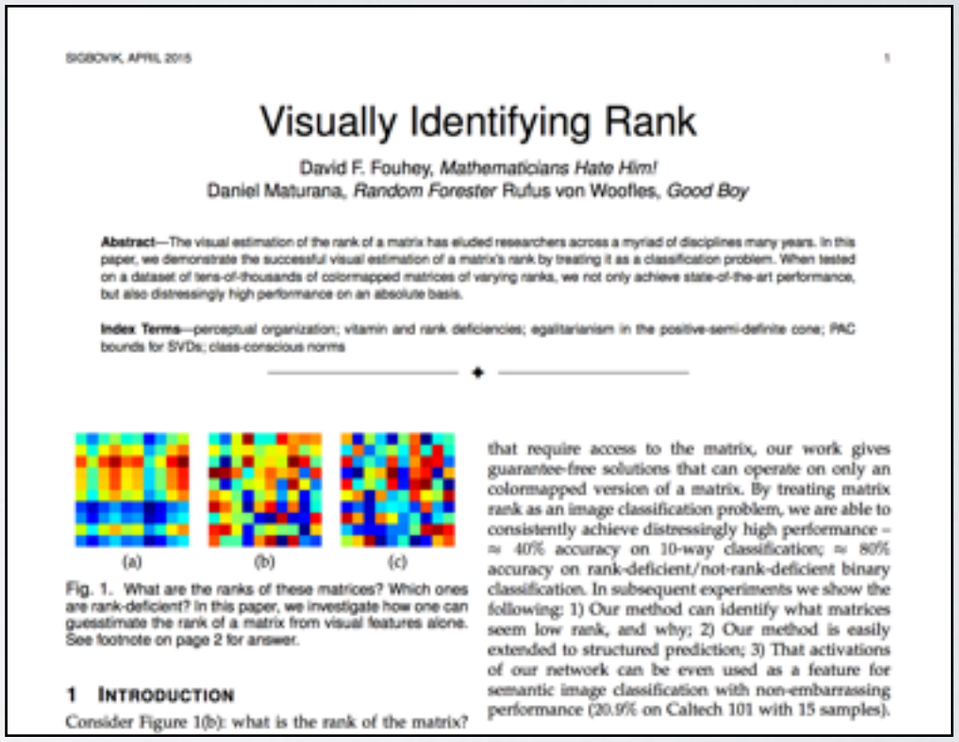
- two deep learning projects
  - Teams up to 3
- **GPU access**
- Python preferred
  - Caffe, TF or Theano

PROJECT 1

1960s

PROJECT 2

- Open ended
  - can be your research
  - **not** published by Dec 31
- level of a top tier workshop publication
  - CVPR, ICCV, ICML, NIPS, ACL, SIGGRAPH
  - SIGBOVIK



SIGBOVIK, APRIL 2015 1

Visually Identifying Rank

David F. Fouhey, *Mathematicians Hate Him!*
Daniel Maturana, *Random Forester* Rufus von Woofles, *Good Boy*

Abstract—The visual estimation of the rank of a matrix has eluded researchers across a myriad of disciplines many years. In this paper, we demonstrate the successful visual estimation of a matrix's rank by treating it as a classification problem. When tested on a dataset of tens-of-thousands of colormapped matrices of varying ranks, we not only achieve state-of-the-art performance, but also distressingly high performance on an absolute basis.

Index Terms—perceptual organization; vitamin and rank deficiencies; egalitarianism in the positive-semi-definite cone; PAC bounds for SVDs; class-conscious norms

(a) (b) (c)

Fig. 1. What are the ranks of these matrices? Which ones are rank-deficient? In this paper, we investigate how one can guesstimate the rank of a matrix from visual features alone. See footnote on page 2 for answer.

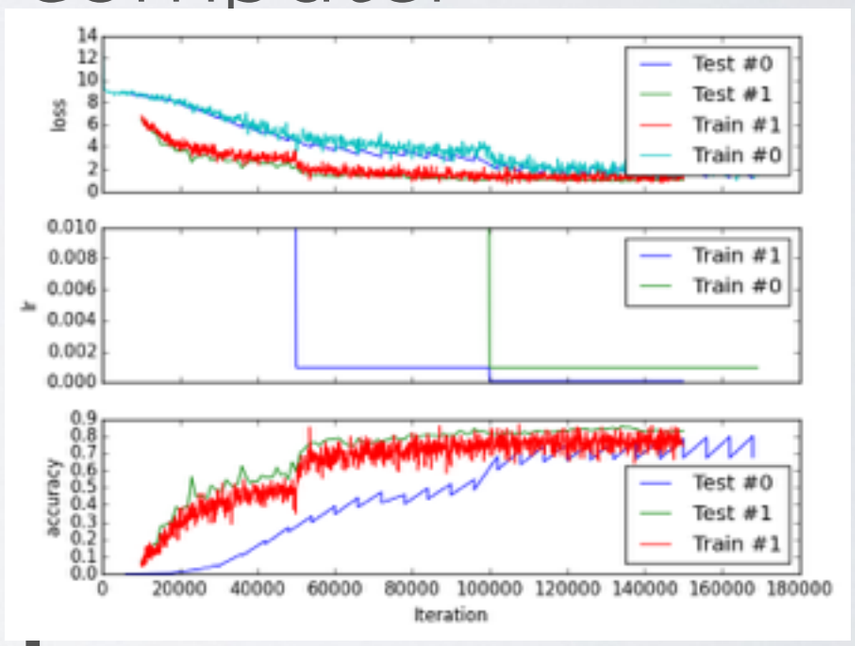
1 Introduction

Consider Figure 1(b): what is the rank of the matrix?

that require access to the matrix, our work gives guarantee-free solutions that can operate on only an colormapped version of a matrix. By treating matrix rank as an image classification problem, we are able to consistently achieve distressingly high performance – ≈ 40% accuracy on 10-way classification; ≈ 80% accuracy on rank-deficient/not-rank-deficient binary classification. In subsequent experiments we show the following: 1) Our method can identify what matrices seem low rank, and why; 2) Our method is easily extended to structured prediction; 3) That activations of our network can be even used as a feature for semantic image classification with non-embarrassing performance (20.9% on Caltech 101 with 15 samples).

PREREQUISITES

- 391L - Intro Machine learning (or equivalent)
- 311 or 311H - Discrete math for computer science (or equivalent)
- Proficiency in Python
- Basic **deep learning background**



GOALS

- Review a deep learning paper
- Give an interesting DL presentation
- Devise and execute a DL project

GRADES

- 30% paper presentation
- 30% project 1 (10% presentation, 20% project)
- 40% project 2 (10% presentation, 30% project)
- (optional) 12.5% volunteering for second presentation

GRADES

```
def grade(p):
  from math import floor
  if p < 50: return 'F'
  v = (100-p) * 4 / (50 + 1e-5)
  return chr(ord('A')+floor(v)) +
         ['+','','','-'][floor((v-floor(v))*4)]
```

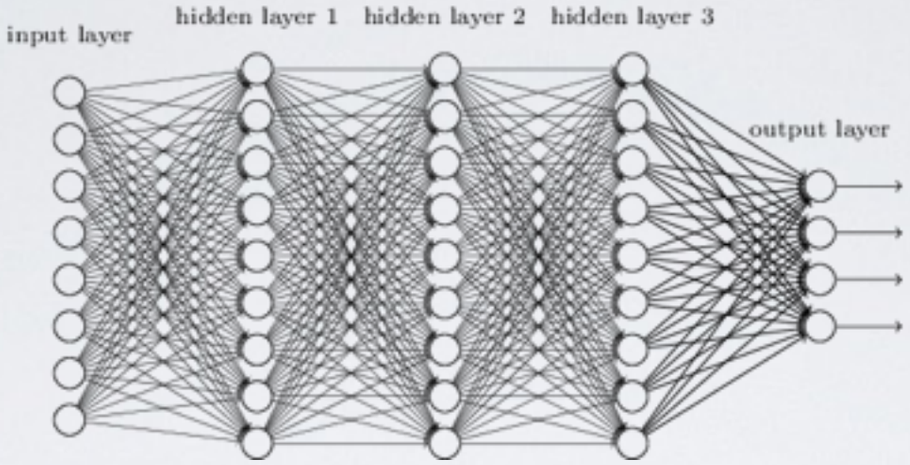
we might train a deep network to grade instead

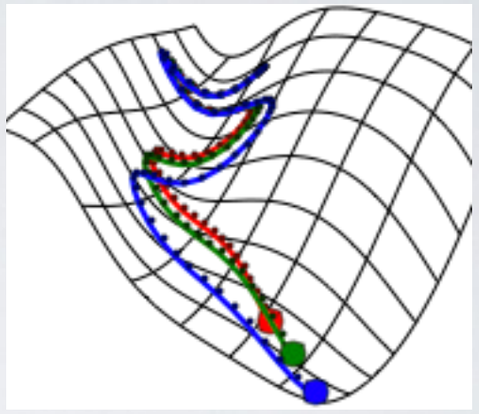
AUDITING

- Allowed
- No homework or presentation required
- Paper review and discussion required

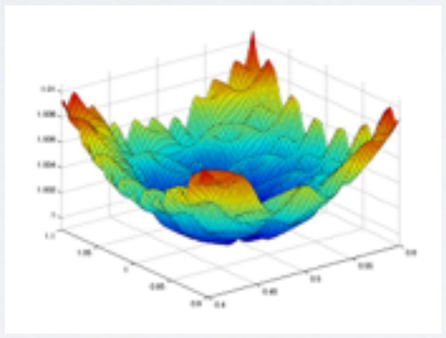
TOPICS

week 2

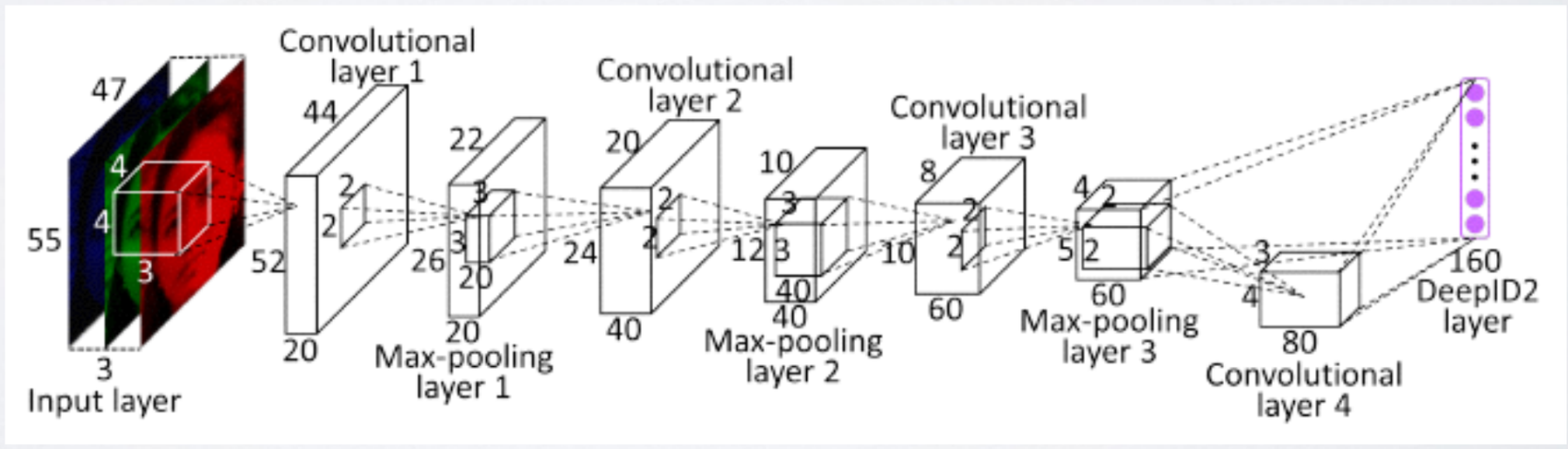




week 3

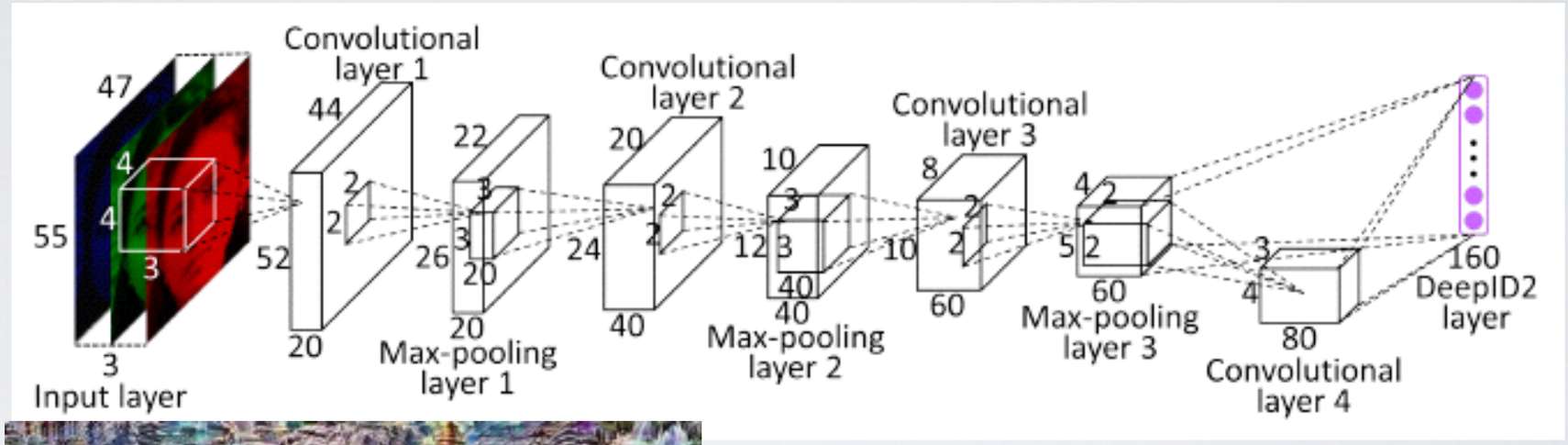


week 4



TOPICS

week 5

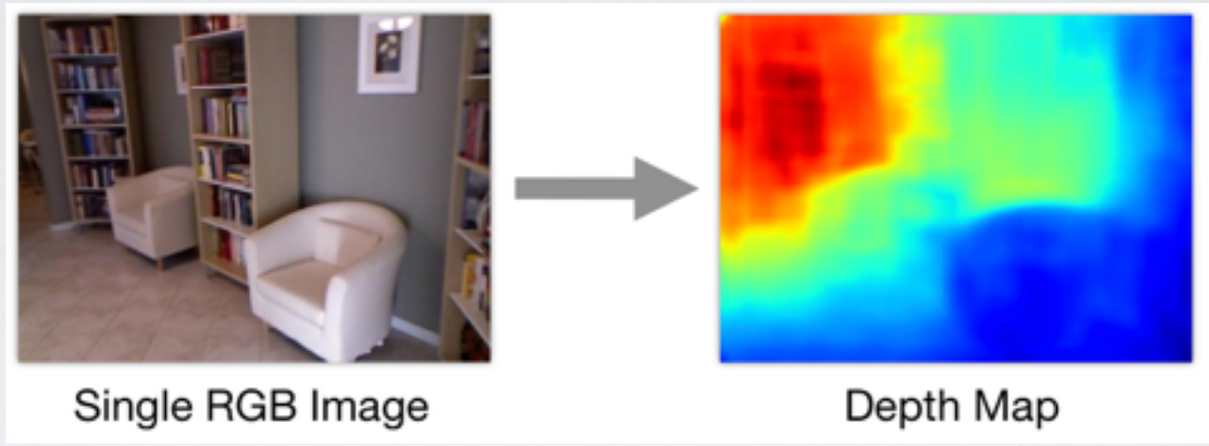


week 6

week 7 Project 1 presentations

TOPICS

week 8



week 9

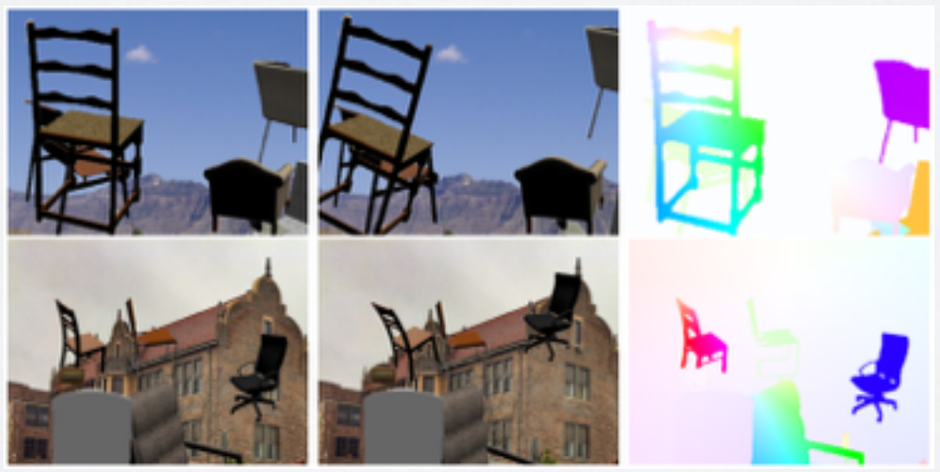
week 10

TOPICS

week 11

week 12

week 13



TOPICS

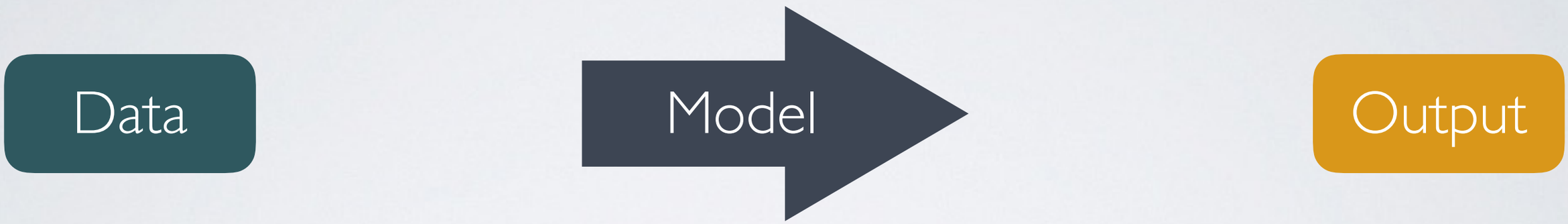
week 14 TPD

week 15 Project 2 presentations

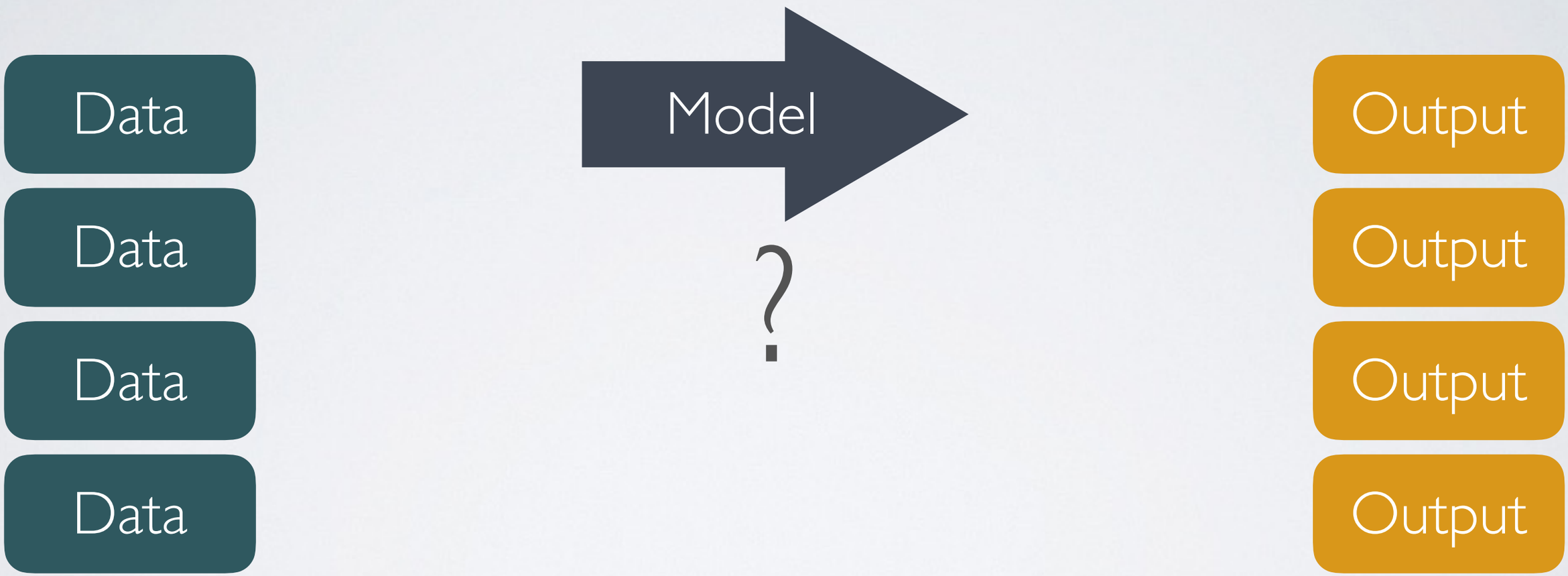
THE N-WORD

- Neural
  - try to keep Neuroscience out of this class
  - try to motivate through optimization and ML
    - instead of biology

REVIEW



TRAINING



TESTING / INFERENCE

Data

Model

?

DATA

Data

feature

$f_1 = \{a,b,c,\ldots\}$

$f_2 = \{d,e,f,\ldots\}$

…

EXAMPLE: GRADING

Data

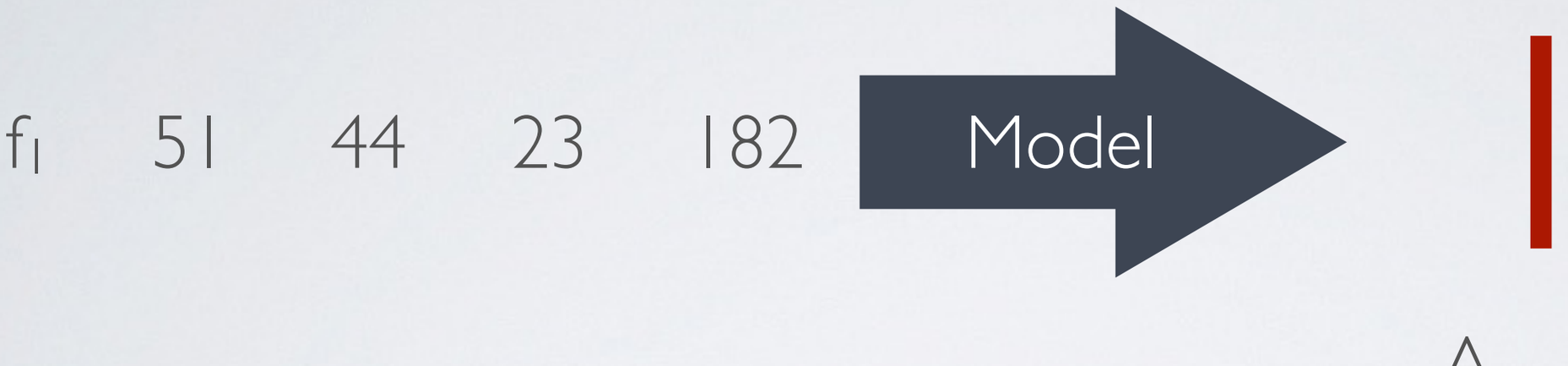
	feature			
	score Project 1	score Project 2	age	height
f_1	51	44	23	182
f_2	25	80	26	172

EXAMPLE: GRADING

f_1	51	44	23	182	Model	A+
f_2	25	80	26	172		A-

continuous and (differentiable)

EXAMPLE: GRADING

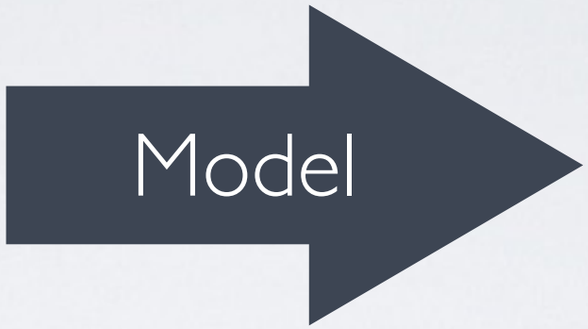


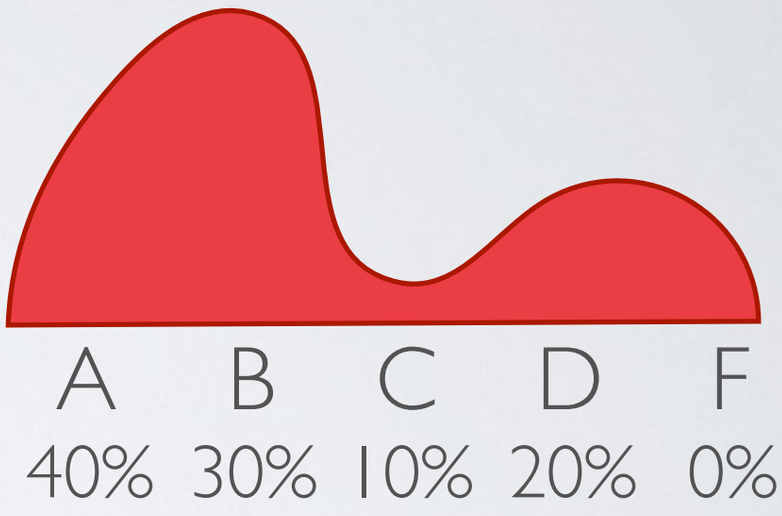
regression:
predict continuous value and round

linear regression
$g = A f + b$

EXAMPLE: GRADING

f_1 51 44 23 182





classification:
predict continuous distribution over grades

logistic regression
$P_g = \text{softmax}(A\ f + b)$

$\text{softmax}(x) = \exp(x) / (\Sigma \exp(x))$

TRAINING

f_1	51	44	23	182	A+
f_2	25	80	26	172	A-

Model

?

EXAMPLE: GRADING

regression

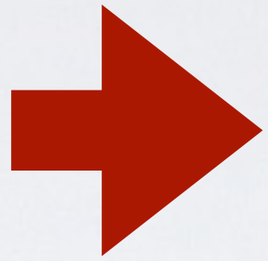
$$\min_{A,b} \Sigma_i (g_i - A f_i - b)^2$$

classification

$$\max_{A,b} \Sigma_i \log P(g_i)$$

FAIRNESS

$f_1 > f_2$



$g_1 \geq g_2$

EXAMPLE: GRADING

regression

$$\underset{A,b}{\text{minimize}} \; \Sigma_i (g_i - A\, f_i - b)^2 + \Sigma_{i,j:f_i<f_j} \max(Af_i - Af_j, 0)$$

classification

$$\underset{A,b}{\text{maximize}} \; \Sigma_i \log P(g_i) + \Sigma_{i,j:f_i<f_j} \max(Af_i - Af_j, 0)$$

EXAMPLE: GRADING

regression

$$\underset{A,b}{\text{minimize}} \; (g_i - A\, f_i - b)^2$$

$$A \geq 0$$

classification

$$\underset{A,b}{\text{maximize}} \; \log P(g_i)$$

$$A_0 \geq A_1$$

NEXT CLASS

- Look at list of papers
- Send Huihuang your top picks per email
  - instructions on Piazza