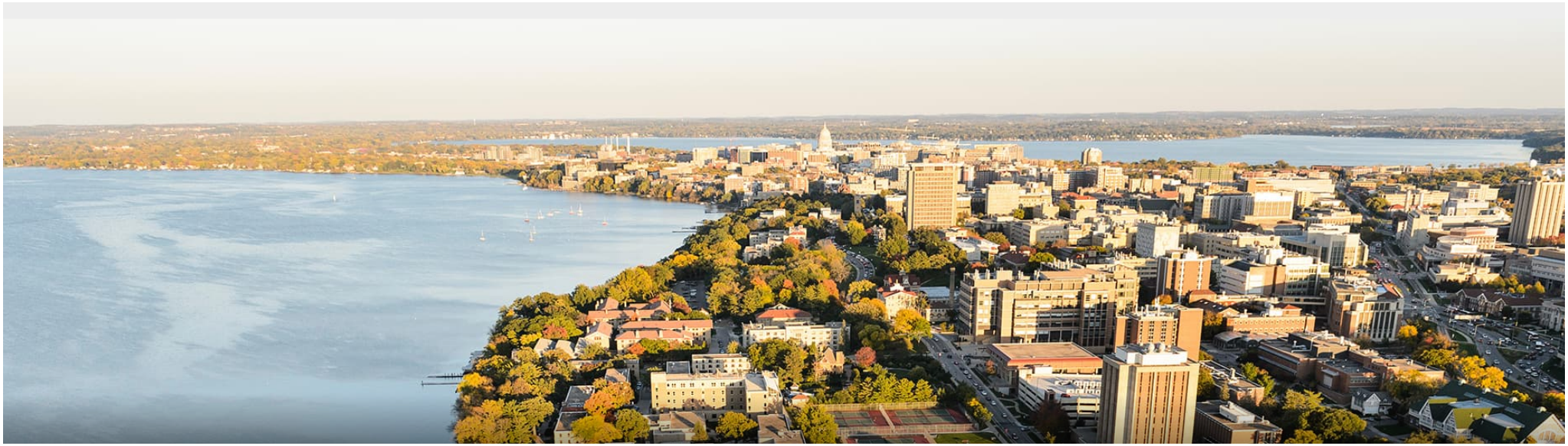


CS 760: Machine Learning
Practical Tips & Class Review

Ilias Diakonikolas

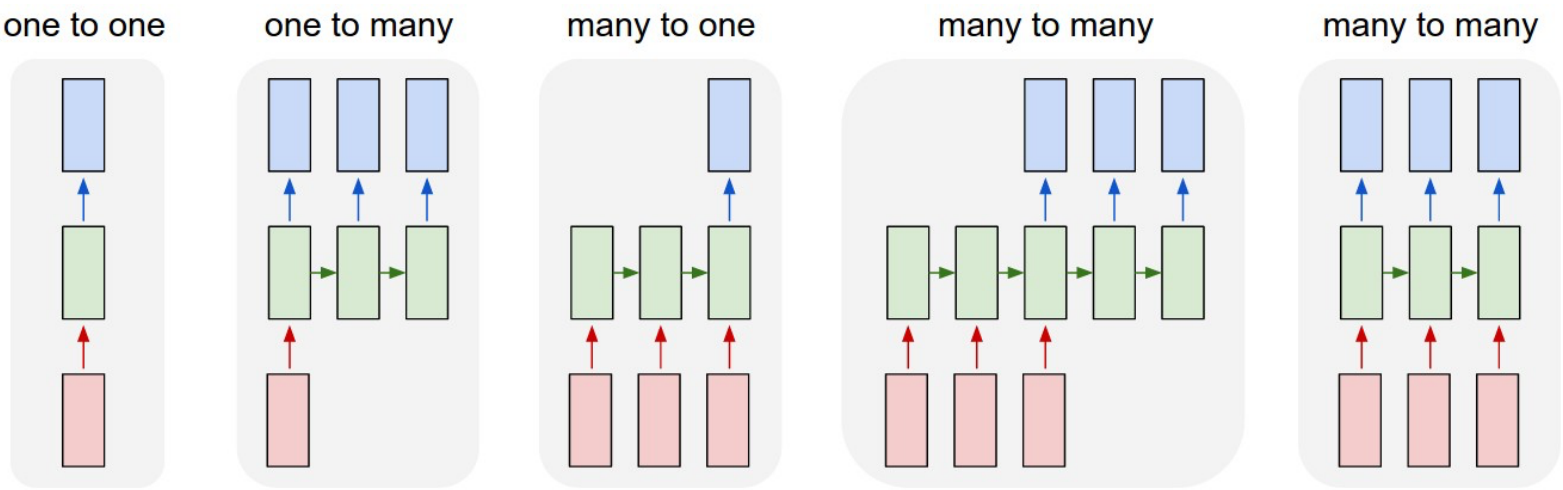
University of Wisconsin-Madison

Nov 1, 2022

Outline

- **RNN Variants + LSTMs**
  - LSTM cells, gates, variations
- **Practical Training Tips & Tricks**
  - Data pipelines, initialization, hyperparameter tuning
- **Review: Class so Far**
  - Cross-validation, optimization, models

Tasks We Can Handle?

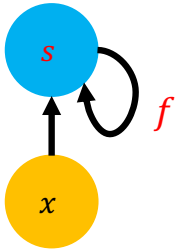


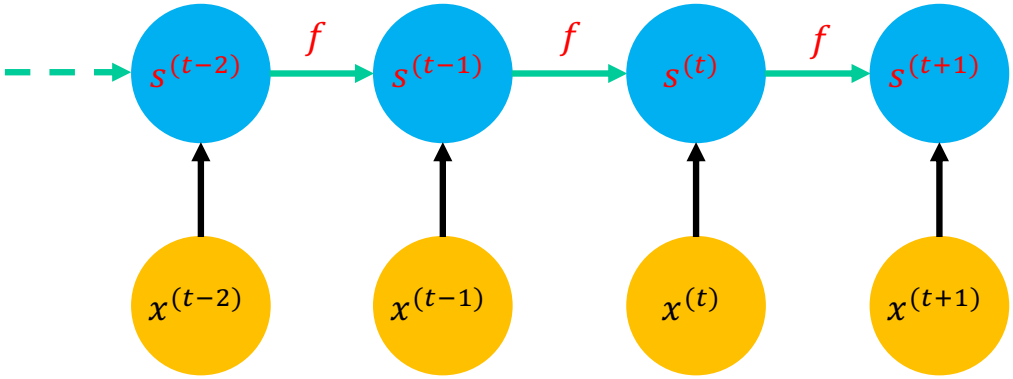
- Don’t have the ability to do anything except (1) so far…
  - Need a new kind of model

Modeling Sequential Data: External Input

- External inputs can also influence transitions
  - $s^{(t)}$ state at time t. Transition function f
  - $x^{(t)}$: input at time t

$$s^{(t+1)} = f(s^{(t)}, x^{(t+1)}; \theta)$$

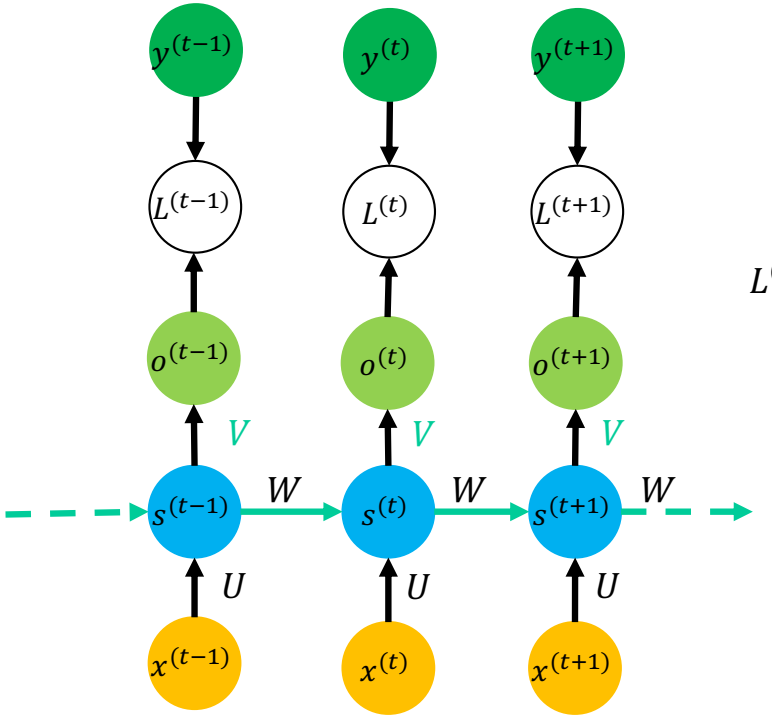




Important: the same f and θ for all time steps

Simple RNNs

- Classical RNN variant:



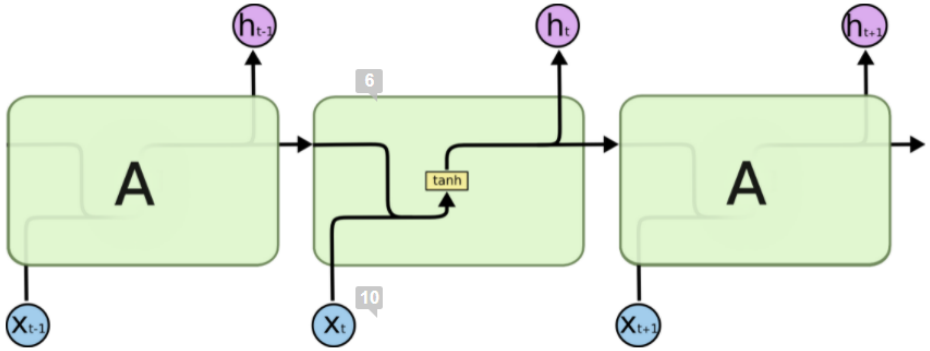
$$a^{(t)} = b + Ws^{(t-1)} + Ux^{(t)}$$
$$s^{(t)} = \tanh\left(a^{(t)}\right)$$
$$o^{(t)} = c + Vs^{(t)}$$
$$\hat{y}^{(t)} = \text{softmax}\left(o^{(t)}\right)$$
$$L^{(t)} = \text{CrossEntropy}(y^{(t)}, \hat{y}^{(t)})$$

RNN Problems

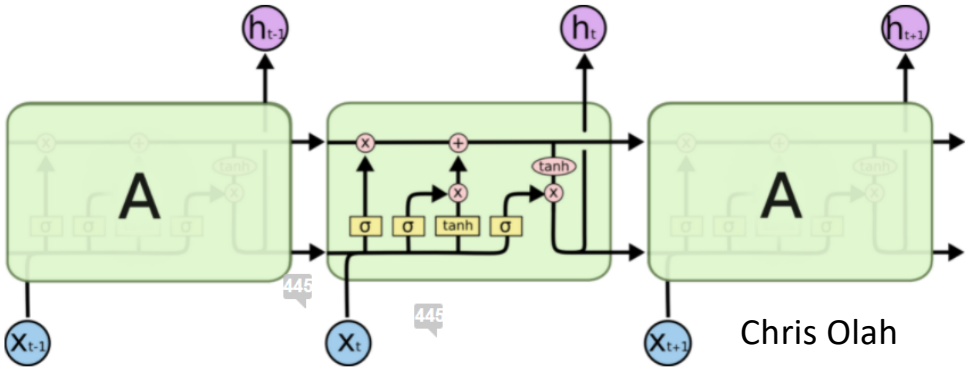
- What happens to gradients in backprop w. many layers?
  - In an RNN trained on long sequences (*e.g.* 100 time steps) the gradients can easily explode or vanish.
  - We can avoid this by initializing the weights very carefully.
- Even with good initial weights, very hard to detect that current target output **depends** on an input from long ago.
  - RNNs have difficulty dealing with long-range dependencies.

LSTM Architecture

- RNN: can write structure as:

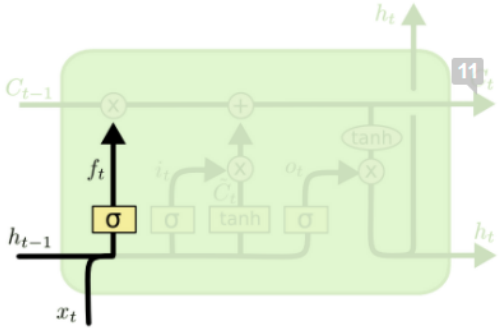


- Long Short-Term Memory: deals with problem. Cell:



Understanding the LSTM Cell

- Step-by-step
  - Good reference: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

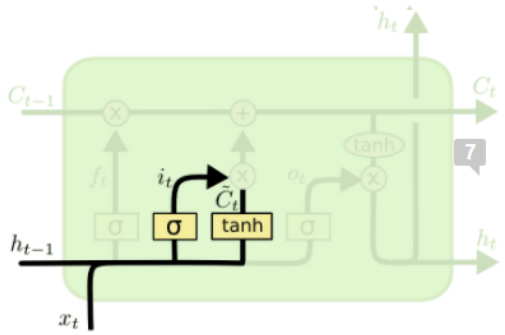


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

- "**Forget**" gate.
  - Can remove all or part of any entry in cell state C
  - Note the sigmoid activation

Understanding the LSTM Cell

- Step-by-step



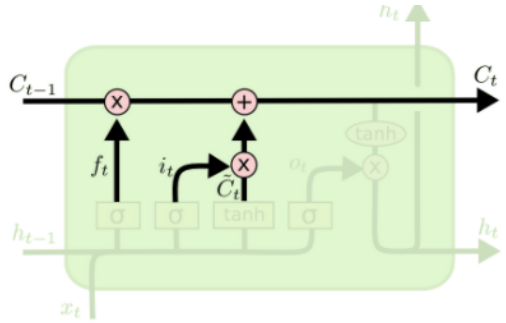
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

- **Input** gate. Combine:
  - What entries in C_{t-1} we'll update
  - Candidates for updating: $\acute{C}_t$
  - Add information to cell state C_{t-1} (post-forgetting)

Understanding the LSTM Cell

- Step-by-step

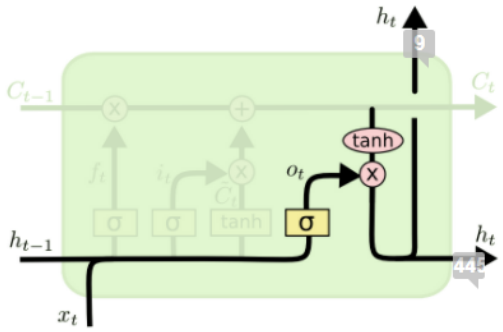


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

- Updating C_{t-1} to C_t
  - Forget, then
  - Add new information

Understanding the LSTM Cell

- Step-by-step

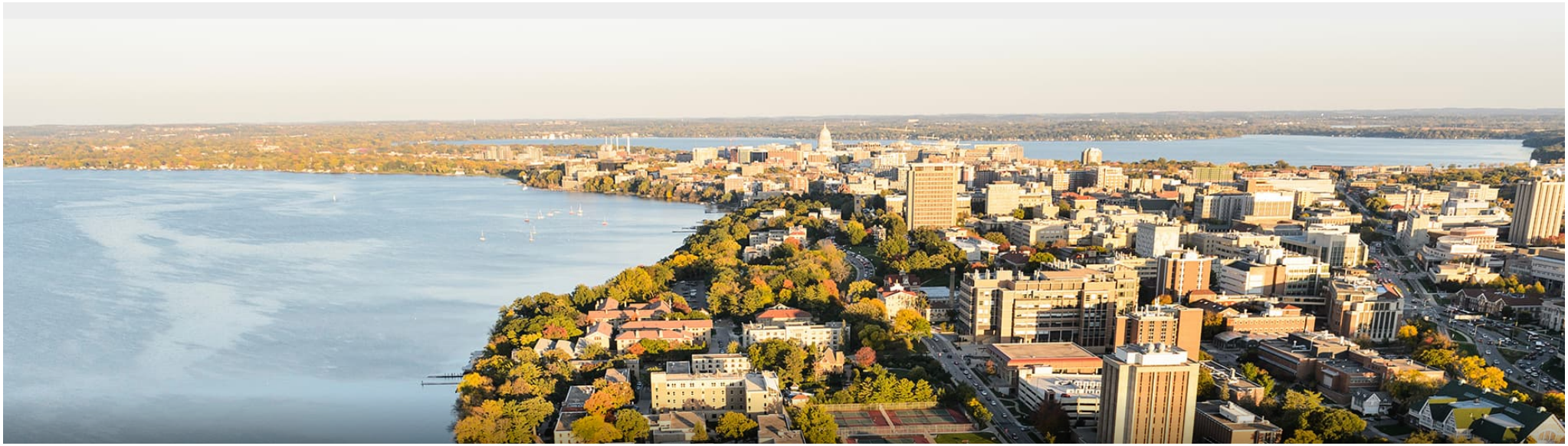


$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

- **Output** gate
  - Combine hidden state, input as before, but also
  - Modify according to cell state C_t

Outline

- **RNN Variants + LSTMs**
  - LSTM cells, gates, variations
- **Practical Training Tips & Tricks**
  - Data pipelines, initialization, hyperparameter tuning



Break & Quiz

Tips & Tricks: Initial Pipeline

First step: building a simple pipeline

- Set up data, model training, evaluation loop

- Use a fixed seed
  - Don’t want to get different values each time

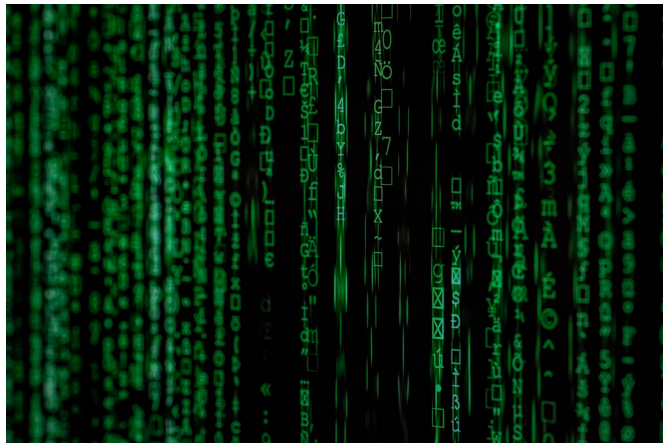
- Overfit on one batch
  - Goal: see that we can get zero loss, catch any bugs

- Check training loss: goes down?

Tips & Tricks: Data

- Shuffle the training data
  - In training ,usually don't select random examples, but rather go through the dataset for each epoch
  - Shuffle to avoid relationships between consecutive points

- Pay attention to your data
  - Properties?



Tips & Tricks: Initialization

Usually want to pick small random values

- Final layer, could use knowledge of problem. **Ex:** if mean is u, initialize to u
- Don't want the same value: symmetry means every weights has same gradient, hard to break out of

- Multiple methods: various rules of thumb
  - Sample from a normal distribution
  - Note that #inputs affects the variance… grows as d^2 for d inputs. Can correct by dividing by $1/\sqrt{n}$

Tips & Tricks: Learning Rate Schedule

- Simple ways:
  - Constant
  - Divide by a factor ever certain number of epochs (annealing)
  - Look at validation loss and reduce on plateau

- Also simple: use an optimizer like Adam that internally tracks learning rates
  - In fact, per parameter

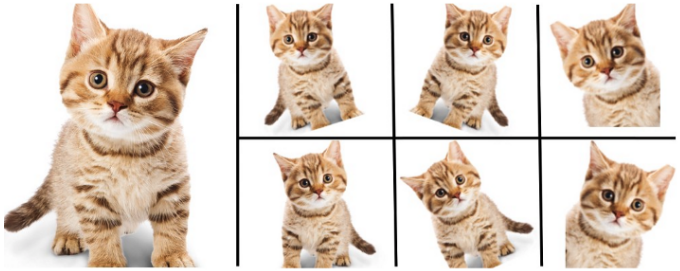
- Lots of variations available

Tips & Tricks: Regularizing

- Best thing to do: get more data!
  - Not always possible or cheap, but start here.

- Augmentation
  - But make sure you understand the transformations

- Use other strategies: dropout, weight decay, early stopping
  - Check each strategy one-at-a-time

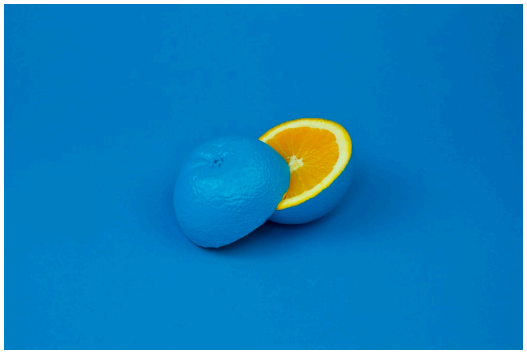


Tips & Tricks: Hyperparameter Tuning

Many solutions:

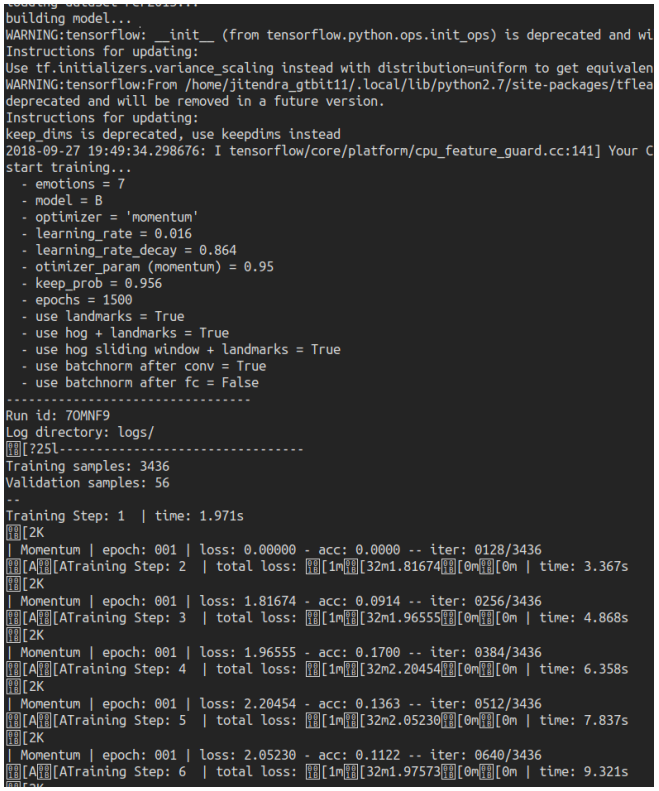
- **Grid Search**: pick candidate sets $S_1,\ldots,S_k$ for each hparam, search over every combination in $S_1 \times S_2 \times \ldots \times S_k$
- **Random Search**
- **Bayesian Approaches**
- **Hyperband**: use successive halfing

Li et al, "Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization", 2018



Tips & Tricks: Monitoring & Logging

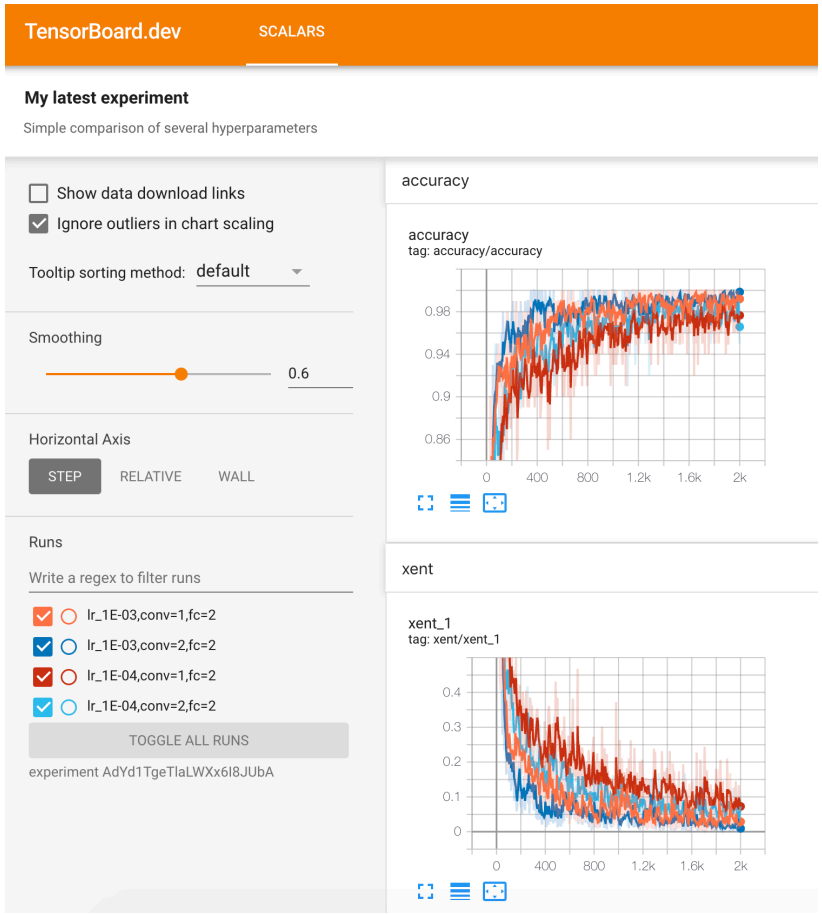
- Checkpoint your models
  - Save weights regularly
- Log information from training process
  - At least keep track of train / test losses, time elapsed, current training settings. Log regularly



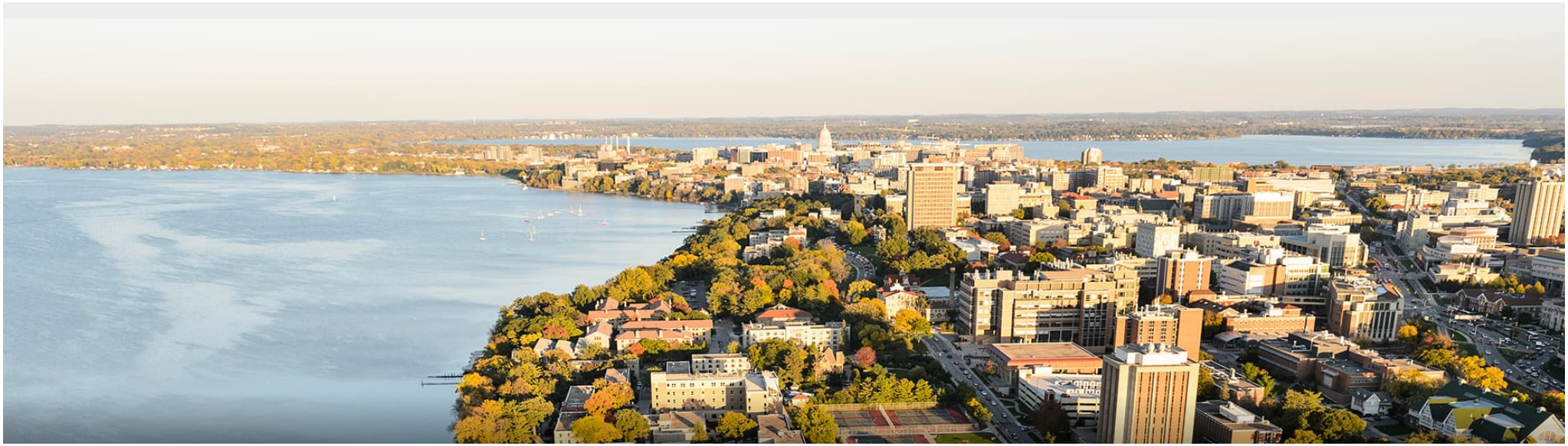
NeptuneAI

Tips & Tricks: Monitoring & Logging

- Log information from training process
  - Use software packages
  - Also have built-in visualization
  - Example: TensorBoard



pytorch.org



Break & Quiz

Outline

- **RNN Variants + LSTMs**
  - LSTM cells, gates, variations
- **Practical Training Tips & Tricks**
  - Data pipelines, initialization, hyperparameter tuning
- **Review: Class So Far**
  - Cross-validation, optimization, models

Supervised Learning: Formal Setup

Problem setting

- Set of possible instances $\mathcal{X}$
- Unknown *target function* $f : \mathcal{X} \rightarrow \mathcal{Y}$
- Set of *models* (a.k.a. *hypotheses*): $\mathcal{H} = \{h | h : \mathcal{X} \rightarrow \mathcal{Y}\}$

Get

- Training set of instances for unknown target function,

$$(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(n)}, y^{(n)})$$

safe

poisonous

safe

SL: Training & Generalization

Goal: model *h* that best approximates *f*

- One way: empirical risk minimization (ERM)

$$\hat{f} = \arg\min_{h \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^{n} \ell(h(x^{(i)}), y^{(i)}))$$

Hypothesis Class

Loss function (how far are we)?

Model prediction

- Generalization?

k-Nearest Neighbors: Classification

Training/learning: given

$$\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}$$

Prediction: for x , find ***k*** most similar training points

Return plurality class

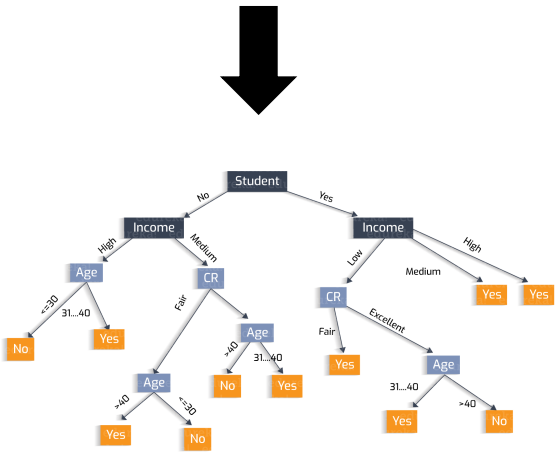
$$\hat{y} \leftarrow \arg\max_{v \in \mathcal{Y}} \sum_{i=1}^{k} \delta(v, y^{(i)})$$

- I.e., among the ***k*** points, output most popular class.

Decision Trees: Learning

- **Learning Algorithm**: MakeSubtree(set of training instances D)

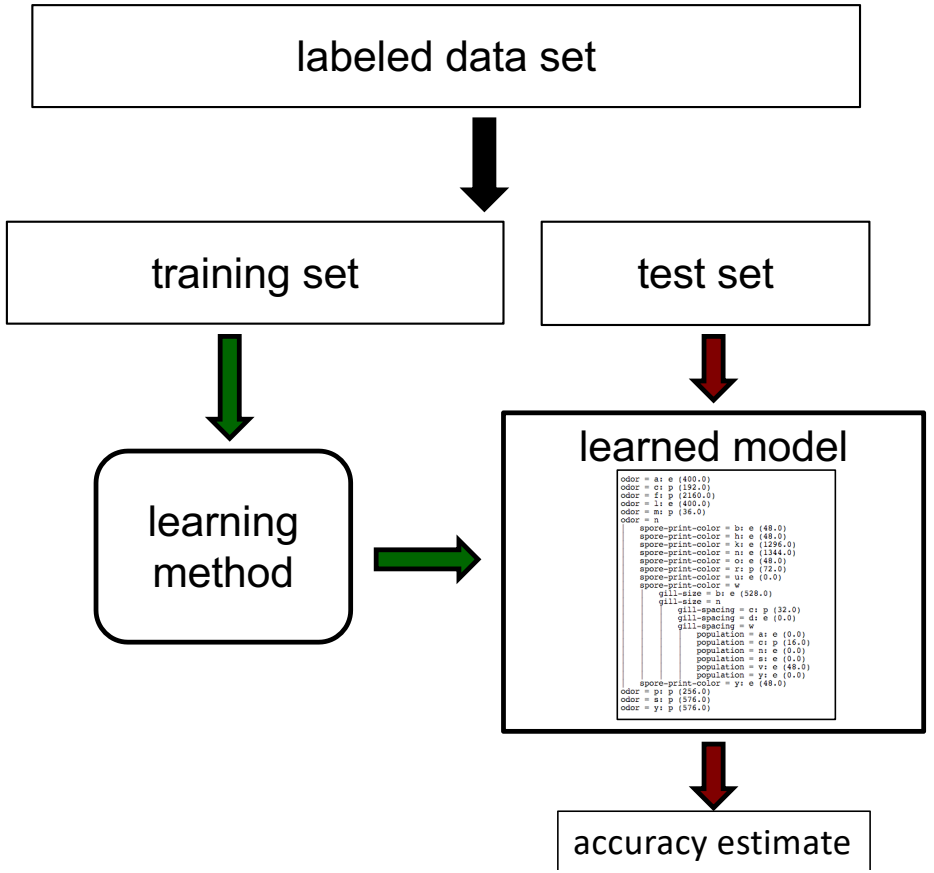
$\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}$



MakeSubtree(set of training instances D)
  C = DetermineCandidateSplits(D)
  if stopping criteria met
    make a leaf node N
    determine class label/probabilities for N
  else
    make an internal node N
    S = FindBestSplit(D, C)
    for each outcome k of S
      D_k = subset of instances that have outcome k
      k^{th} child of N = MakeSubtree(D_k)
  return subtree rooted at N

Bias: Accuracy of a Model

- How can we get an **unbiased** estimate of the accuracy of a learned model?



- Unbiased estimate of θ

$$\mathbb{E}[\hat{\theta}] = \theta$$

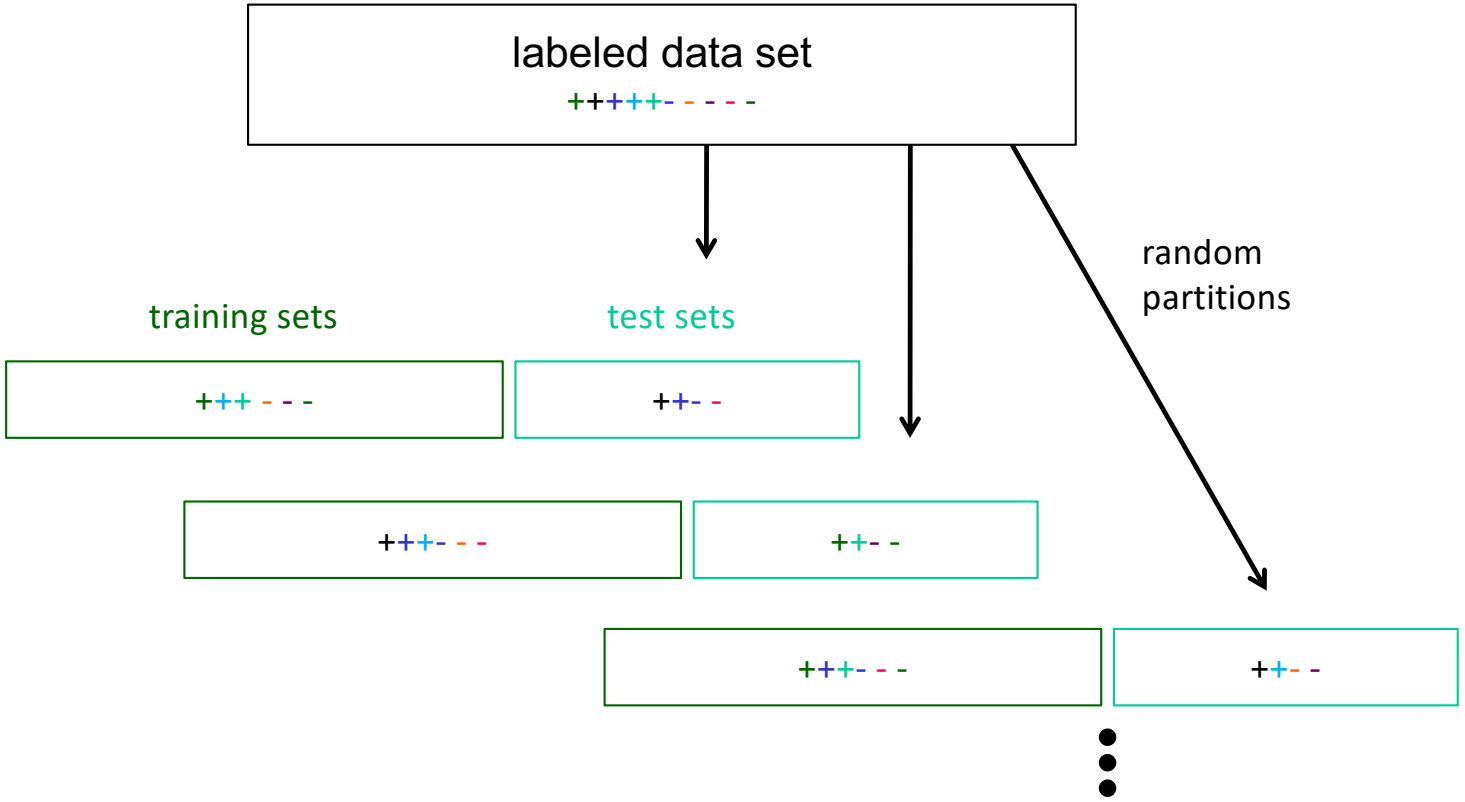
Bias: Using a Test Set

- How can we get an unbiased estimate of the accuracy of a learned model?
  - When learning a model, you should pretend that you don't have the test data yet (it is "in the mail")
  - If the test-set labels influence the learned model in any way, accuracy estimates will be **biased**

- **Don't train on the test set!**

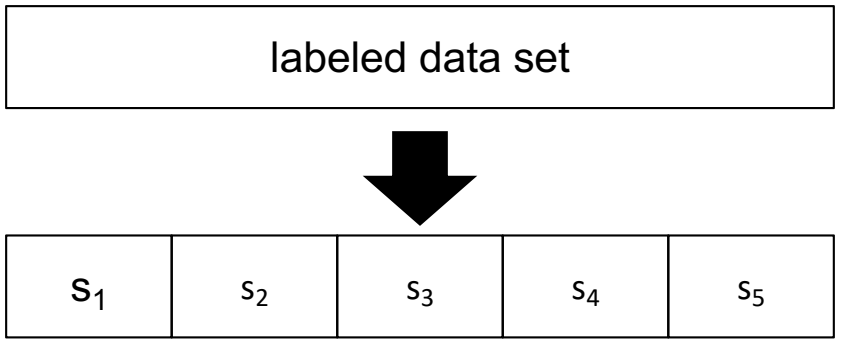
Strategy I: Random Resampling

- Address the second issue by repeatedly randomly partitioning the available data into training and test sets.



Strategy II: Cross Validation

Partition data into *n* subsamples



Iteratively leave one subsample out for the test set, train on the rest

iteration	train on	test on
1	s_2 s_3 s_4 s_5	s_1
2	s_1 s_3 s_4 s_5	s_2
3	s_1 s_2 s_4 s_5	s_3
4	s_1 s_2 s_3 s_5	s_4
5	s_1 s_2 s_3 s_4	s_5

Strategy II: Cross Validation Example

- Suppose we have 100 instances, and we want to estimate accuracy with cross validation

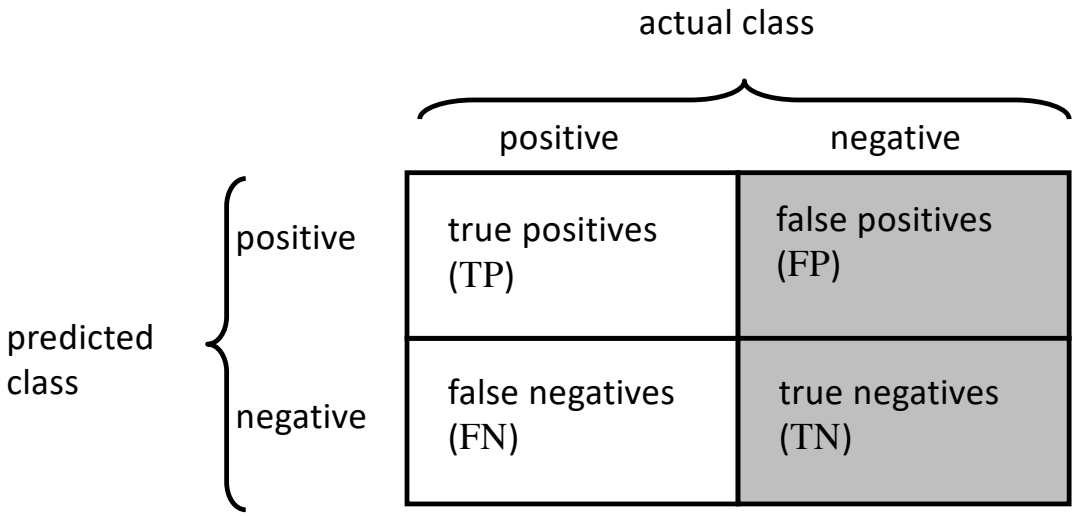
iteration	train on	test on	correct
1	s_2 s_3 s_4 s_5	s_1	11 / 20
2	s_1 s_3 s_4 s_5	s_2	17 / 20
3	s_1 s_2 s_4 s_5	s_3	16 / 20
4	s_1 s_2 s_3 s_5	s_4	13 / 20
5	s_1 s_2 s_3 s_4	s_5	16 / 20

accuracy = 73/100 = 73%

Strategy II: Cross Validation Tips

- 10-fold cross validation is common, but smaller values of *n* are often used when learning takes a lot of time
- in *leave-one-out* cross validation, *n* = # instances
- in *stratified* cross validation, stratified sampling is used when partitioning the data
- CV makes efficient use of the available data for testing
- note that whenever we use multiple training sets, as in CV and random resampling, we are evaluating a learning method as opposed to an individual learned hypothesis

Other Metrics

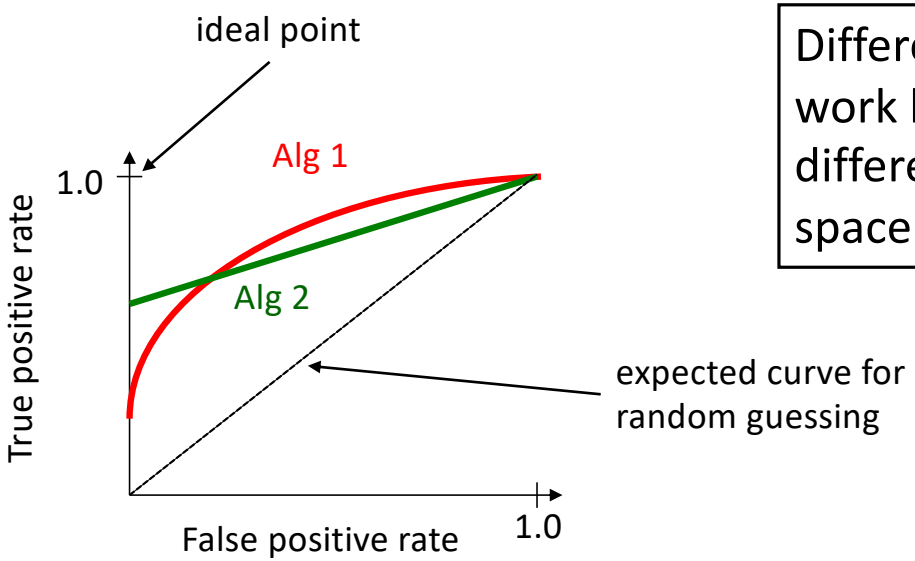


$$\text{true positive rate (recall)} = \frac{TP}{\text{actual pos}} = \frac{TP}{TP + FN}$$

$$\text{false positive rate} = \frac{FP}{\text{actual neg}} = \frac{FP}{TN + FP}$$

Other Metrics: ROC Curves

- A *Receiver Operating Characteristic* (*ROC*) curve plots the TP-rate vs. the FP-rate as a threshold on the confidence of an instance being positive is varied



Different methods can work better in different parts of ROC space.

Iterative Methods: Gradient Descent

- What if there's no closed-form solution?
- Use an iterative approach. Goal: get closer to solution.

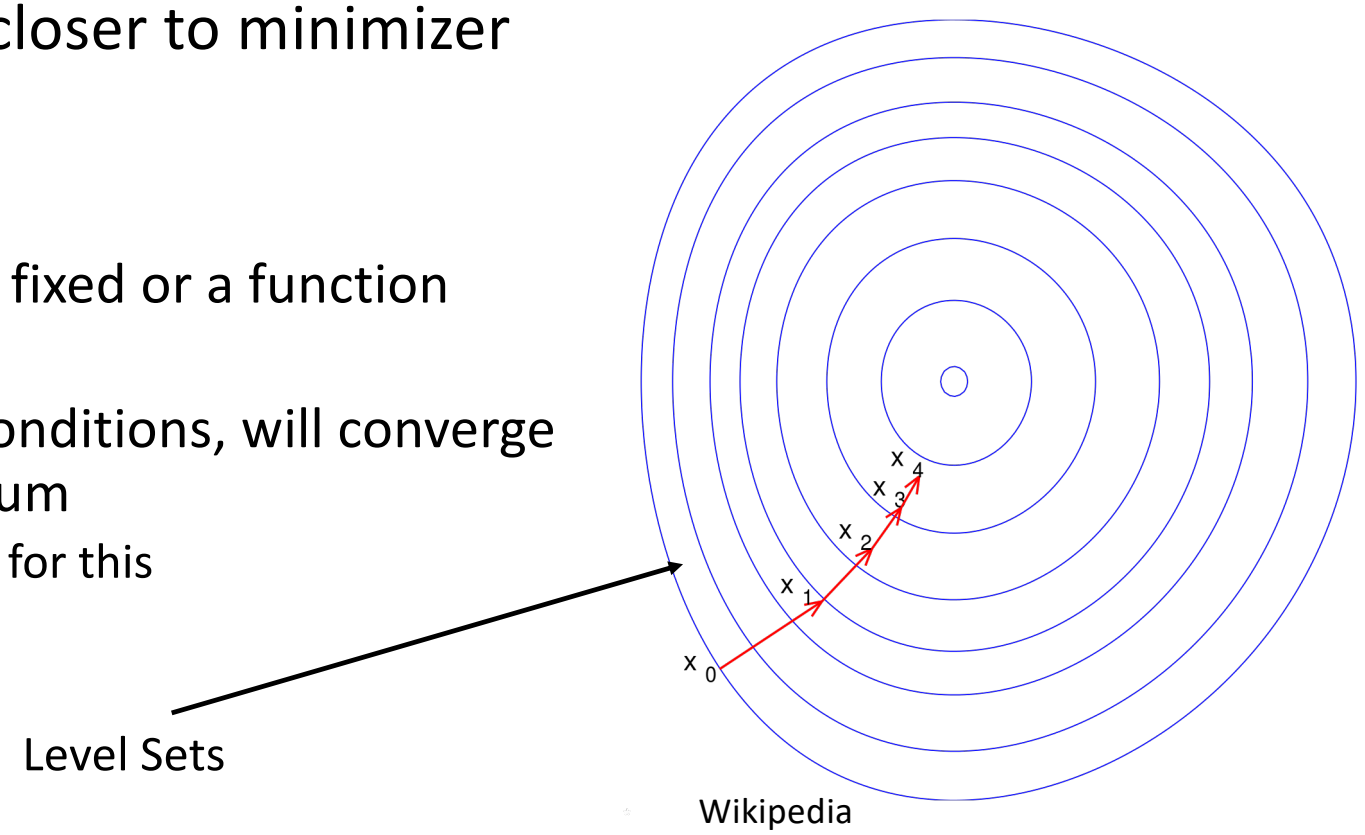
- Gradient descent.
  - Suppose we're computing $\min_{\theta} g(\theta)$
  - Start at some θ_0
  - Iteratively compute $\theta_{t+1} = \theta_t - \alpha \nabla g(\theta_t)$
  - Stop after some # of steps

Learning rate/step size

Gradient Descent: Illustration

- **Goal**: steps get closer to minimizer

- Some notes:
  - Step size can be fixed or a function
  - Under certain conditions, will converge to global minimum
    - Need **convexity** for this



Gradient Descent: Linear Regression

- Example for linear regression problem.
- Want to find $\min_\theta \ell(f_\theta) = \min_\theta \frac{1}{n}\|X\theta - y\|_2^2$
  - What's our gradient? $\nabla \ell(f_\theta) = \frac{1}{n}(2X^T X\theta - 2X^T y)$
  - So, plugging in , we get

$$\theta_{t+1} = \theta_t - \alpha\frac{1}{n}(2X^T X\theta_t - 2X^T y)$$

Iterative Methods: Gradient Descent

- Simple modification to GD.
- Let's use some notation: ERM:

$$\arg\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} \ell(f(\theta; x^{(i)}), y^{(i)})$$

Note: this is what we're optimizing over!
x's are fixed samples.

- GD: $$\theta_{t+1} = \theta_t - \frac{\alpha}{n} \sum_{i=1}^{n} \nabla \ell(f(\theta_t; x^{(i)}), y^{(i)})$$

Gradient Descent: Convergence

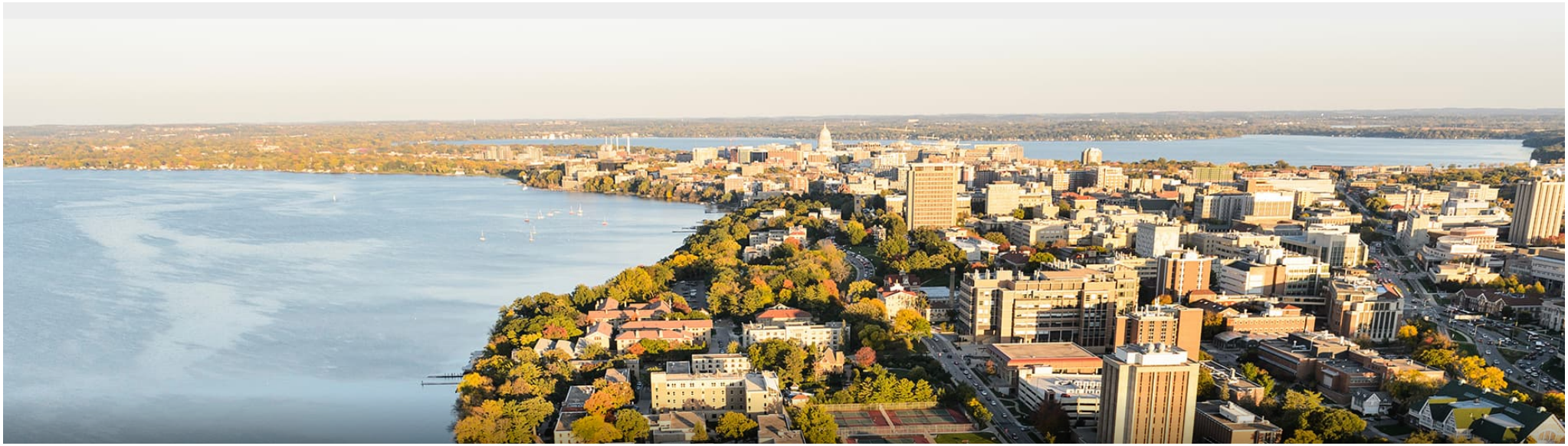
- Even if GD is cheaper, what does it give us?
- Let's analyze it. We'll need some ingredients
  - Convex function *g*
  - Differentiable (need this for gradients)
  - Lipschitz-continuous gradients

$$\|\nabla g(x_1) - \nabla g(x_2)\|_2 \leq L\|x_1 - x_2\|_2$$

  - If we run *t* steps with fixed step size, starting at x_0

$$g(x_t) - g(x^*) \leq \frac{\|x_0 - x^*\|_2^2}{2t\alpha}$$

Minimizer



Thanks Everyone!

Some of the slides in these lectures have been adapted/borrowed from materials developed by Mark Craven, David Page, Jude Shavlik, Tom Mitchell, Nina Balcan, Elad Hazan, Tom Dietterich, Pedro Domingos, Jerry Zhu, Yingyu Liang, Volodymyr Kuleshov , Sharon Li, Chris Olah, Fred Sala