CIS 522: Lecture 5

Regularization

Lyle Ungar

Penn Engineering

Today

(1) Admin
(2) Review last week
(3) Regularization = shrinkage
(4) Dropout
(5) Meta-learning

Course Goals

- Lectures
- Worksheets
- Pods
- Final project

Time Expectations

1.5 lecture

1.0 pod

3.0 worksheets (median:1.5+1.5 hours)

2.0 homework

2.5 other: extra reading

---

10 hours/week

Last week: Gradient descent

Loss function of geometry leads to poorly conditioned gradients

- Momentum
- Rate schedule
    - Shrink learning weights over time
- Adjust learning rates for each weight
- Minibatch size and normalization
- Fairness

Many methods;

Few concepts

Method	Update equation
SGD	$g_t = \nabla_{\theta_t} J(\theta_t)$ $\Delta\theta_t = -\eta \cdot g_t$ $\theta_t = \theta_t + \Delta\theta_t$
Momentum	$\Delta\theta_t = -\gamma\, v_{t-1} - \eta g_t$
NAG	$\Delta\theta_t = -\gamma\, v_{t-1} - \eta \nabla_\theta J(\theta - \gamma v_{t-1})$
Adagrad	$\Delta\theta_t = -\dfrac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t$
Adadelta	$\Delta\theta_t = -\dfrac{RMS[\Delta\theta]_{t-1}}{RMS[g]_t} g_t$
RMSprop	$\Delta\theta_t = -\dfrac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$
Adam	$\Delta\theta_t = -\dfrac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$

Momentum

- Keep moving the the same direction you were going
  - Less time bouncing up and down steep directions
    - So more doing down the slower directions
  - Gives an approximation to the Hessian

Reduce learning rates over time

- Take big steps initially, when you are far from the optimum
- Slow down as you get closer

Adagrad

- Good: automatically picks learning rate for each weight
- Bad: learning can slow too fast

$$w_j := w_j - \frac{\eta}{\sqrt{G_{j,j}}} g_j. \qquad G_{j,j} = \sum_{\tau=1}^{t} g_{\tau,j}^2.$$

RMSprop

- Exponentially forgets the previous gradients

$$w_j := w_j - \frac{\eta}{\sqrt{G_{j,j}}} g_j.$$

Intuitively:

$$G_{j,j} = \sum_{\tau=1}^{t} g_{\tau,j}^2 \cdot \gamma^{T\text{-}t}$$

Adam

- Behaves like a heavy ball with friction
- Lots of adjustments to make things work better

Minibatch size

- Bigger is faster per observation
  - As long as minibatch fits in GPU DRAM
- Smaller usually gives better models
  - Finds shallower minima
  - May converge faster
- Common: 32, 64, or 128 points
  - Smaller often gives better generalization
  - Depending on other hyperparameters

Minibatch size - theory

- What can you fit on a GPU DRAM?
- **Model**: 4 bytes * 4 * #weights
  - 4 bytes = 32 bit floating point
  - 4 = |weight, gradient, sum of gradients, previous change|
  - Alexnet: 62,378,344 weights -> **1GB**
- **Inputs**: 4 bytes * #features * #training_points
  - Alexnet: 4 * 227x227x3 * 32 = **5 MB**
- **Gradients**: 4 bytes * #weights * #training_points
  - Alexnet * 32 pts -> **8 GB**

Minibatch size - worksheet example

- **Model**
  - # of parameters = [(784 * 128) + 128] + [(128 * 10) + 10]
- **Gradients** (stored for every observation)
  - # of gradients = # of observations * # of parameters
- **Inputs**
  - # of inputs = # of observations * 28 * 28
- **Storage**: 4 * (# of parameters + # of gradients + # of inputs) (bytes)
  - Vanilla GD: # of observations = 60000
    - Cost ~ 25 GB
  - Mini-batch GD: # of observations = 50
    - Cost ~ 0.02 GB

Minibatch size - practice

- GPUs often use 32 minibatch samples of 32-bit floating-point data to create 32*32= 1024-bit-wide data vectors.
  - Gives local storage requirement of over 2 GB.
- In practice, ResNet-50 training with a mini-batch of 32 on a typical GPU needs over 7.5 GB of local DRAM.

https://www.graphcore.ai/posts/why-is-so-much-memory-needed-for-deep-neural-networks

Batch Normalization

- Standardize every input to every neuron within each minibatch
- Can increase learning rate by 10x
  - "reduces internal covariate shift"
- Provides regularization

This week

Regularization

- L1, L2 penalties
- Early stopping
- Data augmentation
- Gradient descent (implicitly)
- Dropout

Hyperparameter tuning + AutoML

Distillation

Adversarial attacks

Regularization is Shrinkage

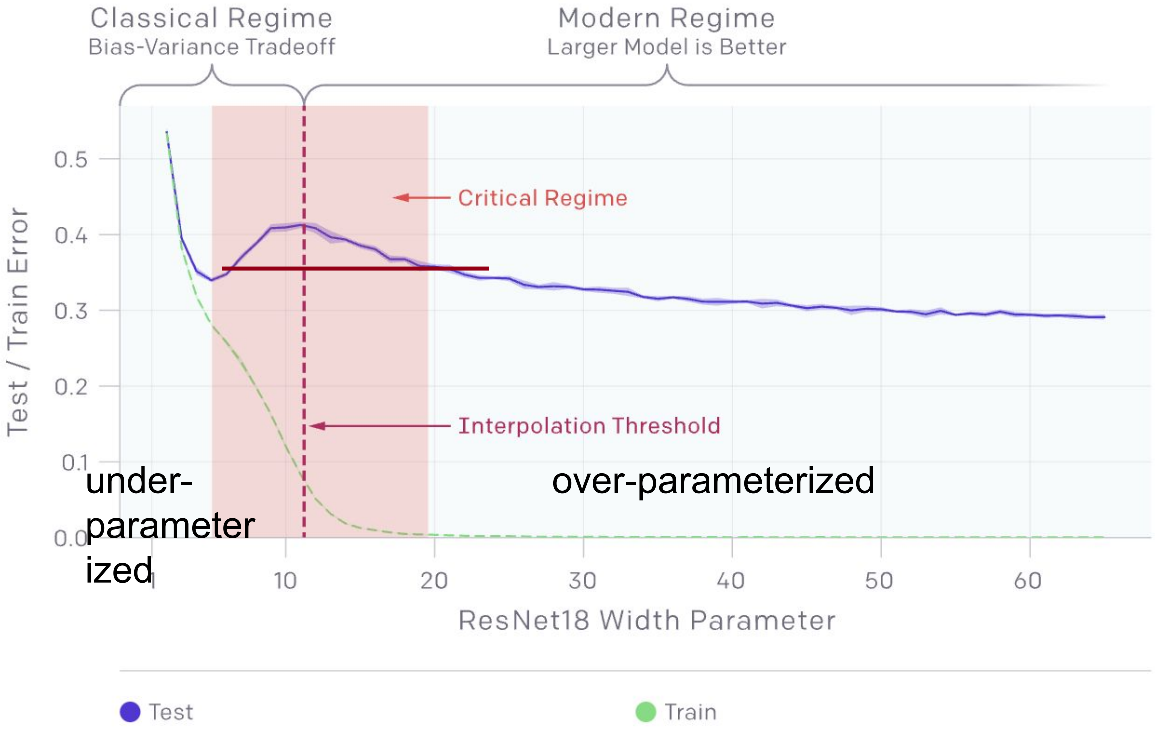
L1, L2 penalties

Early stopping

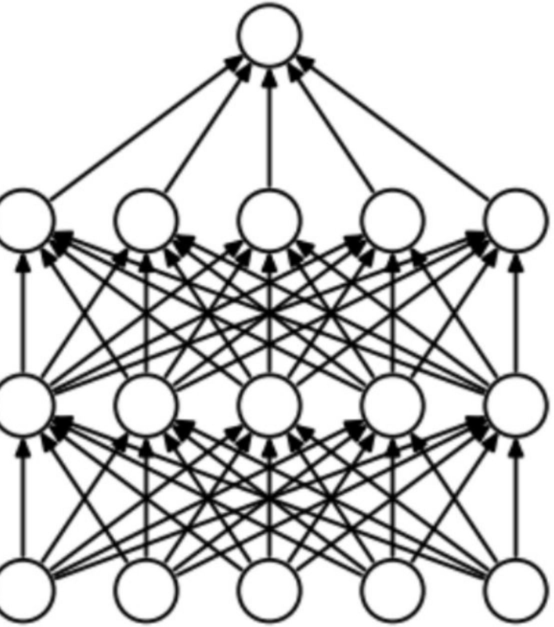
Gradient descent (implicitly)

Dropout

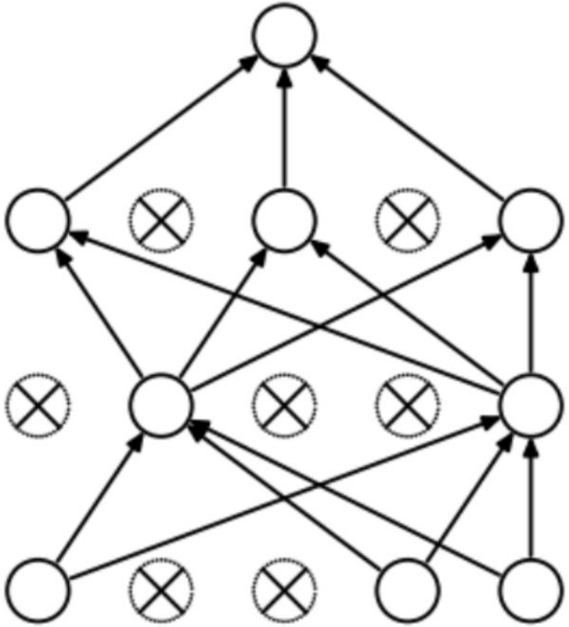
Double Descent (revisited)



Dropout - Review



(a) Standard Neural Net



(b) After applying dropout.

Each minibatch:

Remove a random fraction *p* of the nodes

At test time:

Multiply weights by (1-*p*)

Dropout - theory

- Approximates an ensemble
- Avoids local minima
- Avoids "dead neurons"
- Tends to make the norm of weight vectors of all the hidden nodes in a layer equal

Dropout - practice

- Often use dropout $p=0.5$
  - 0.5 gives maximum regularization
- Sometimes limit to certain layers
  - often use less (or no) dropout on input
  - And in convolutional layers
    - where dropout is weird because the same filter is used many times

AutoML - Gluon

- automatic hyperparameter tuning
- model selection/ensembling
- architecture search
- data processing
- Separate modules for
  - images, text, tabular data

https://auto.gluon.ai/stable/index.html

AutoML – Gluon pytorch

Two layer MLP where we optimize over:

- the number of units on the first layer
- the number of units on the second layer
- the dropout rate after each layer
- the learning rate
- the scaling

https://auto.gluon.ai/stable/index.html

AutoML – Gluon pytorch

Searchers:

- Random
- Bayesian Optimization
  - Gaussian Process-based
  - SkOpt
  - Fair

https://auto.gluon.ai/stable/index.html

Bayesian Optimization

- $\mathit{argmin}_{\mathbf{x}}\ f(\mathbf{x})$
  - $f(x)$ is the testing error for hyperparameters, $\mathbf{x}$
- Model $f(\mathbf{x})$ by a Gaussian Process
  - Gives uncertainties as well as values
- Use $f(\mathbf{x})$ to decide which $\mathbf{x}$ to evaluate next

Practical Bayesian Optimization of Machine Learning Algorithms

Gaussian Processes

- Training data Y
  - Specifies the prior - based on earlier experiments
- New data X
  - Accuracy at new points
- Predict $f(\mathbf{x})$ at $|\mathbf{X}|=N$ new points
  - Generates N-dimensional multivariate Gaussian distribution. (i.e. in the dual)

Bayesian Optimization

- *argmin*$_x$ $f(\mathbf{x})$
  - $f(\mathbf{x})$ is the testing error for hyperparameters, x
- Model $f(\mathbf{x})$ by a Gaussian Process
  - Gives mean and standard deviation at all points
- Use $f(\mathbf{x})$ to decide which **x** to next evaluate
  - E.g., maximize Expected Improvement (EI)
  - Or EI per second

Practical Bayesian Optimization of Machine Learning Algorithms

AutoML

- [Auto-Pytorch]()
  - From the auto-Sklearn team
  - Tabular only
- AutoML competitions: [automl.ai]()

Adversarial Deep Learning

- Pick an input to change the output
    - Object recognition
    - Credit scoring
    - Chatbots
- Modify the training data to change the model

For fun: what is google doing?

- More Capable, General-Purpose ML Models
- Continued Efficiency Improvements for ML
- ML Is Becoming More Personally and Communally Beneficial
- Growing Benefits of ML in Science, Health and Sustainability
- Deeper and Broader Understanding of ML

https://ai.googleblog.com/2022/01/google-research-themes-from-2021-and.html

Questions?

Have an awesome week!