# Deep Learning & Neural Networks
# Lecture 4

Kevin Duh

Graduate School of Information Science
Nara Institute of Science and Technology

Jan 23, 2014

# Advanced Topics in Optimization

- Today we'll briefly survey an assortment of exciting new tricks for optimizing deep architectures
- Although there are *many* exciting new things out of NIPS/ICML every year, I'll pick the four that I think is most helpful for practitioners:
  - SGD alternative
  - Better regularization
  - Scaling to large data
  - Hyper-parameter search

# Today's Topics

1. Hessian-free optimization [Martens, 2010]
2. Dropout regularization [Hinton et al., 2012]
3. Large-scale distributed training [Dean et al., 2012]
4. Hyper-parameter search [Bergstra et al., 2011]

# Difficulty of optimizing highly non-convex loss functions

- "Pathological curvature" is tough to navigate for SGD
- 2nd-order (Newton) methods may be needed to avoid underfitting

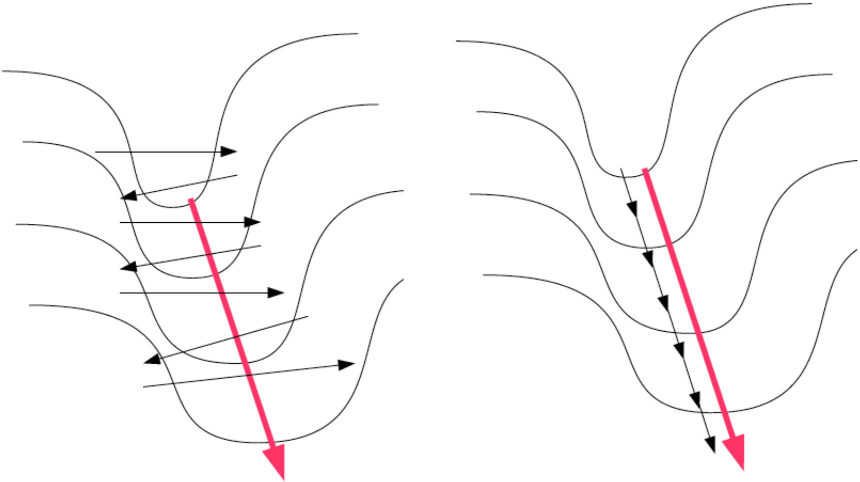


Figure from [Martens, 2010]

# 2nd-order (Newton) methods

- Idea: approximate the loss function locally with a quadratic

  $$L(w+z) \approx q_w(z) \equiv L(w) + \nabla L(w)^T z + \frac{1}{2} z^T H z$$

  where $H$ is the Hessian (curvature matrix) at $w$

# 2nd-order (Newton) methods

- Idea: approximate the loss function locally with a quadratic

  $$L(w+z) \approx q_w(z) \equiv L(w) + \nabla L(w)^T z + \frac{1}{2} z^T H z$$

  where $H$ is the Hessian (curvature matrix) at $w$
- Minimizing this gives the search direction: $z^* = -H^{-1} \nabla L(w)$
  - Intuitively, $H^{-1}$ fixes any pathological curvature for $\nabla L(w)$
  - In practice, don't want to store nor invert $H$

# 2nd-order (Newton) methods

- Idea: approximate the loss function locally with a quadratic

  $$L(w+z) \approx q_w(z) \equiv L(w) + \nabla L(w)^T z + \frac{1}{2} z^T H z$$

  where $H$ is the Hessian (curvature matrix) at $w$
- Minimizing this gives the search direction: $z^* = -H^{-1} \nabla L(w)$
  - Intuitively, $H^{-1}$ fixes any pathological curvature for $\nabla L(w)$
  - In practice, don't want to store nor invert $H$
- Quasi-Newton methods
  - L-BFGS: uses low-rank approximation of $H^{-1}$

# 2nd-order (Newton) methods

- Idea: approximate the loss function locally with a quadratic

  $L(w+z) \approx q_w(z) \equiv L(w) + \nabla L(w)^T z + \frac{1}{2} z^T H z$

  where $H$ is the Hessian (curvature matrix) at $w$
- Minimizing this gives the search direction: $z^* = -H^{-1} \nabla L(w)$
  - Intuitively, $H^{-1}$ fixes any pathological curvature for $\nabla L(w)$
  - In practice, don't want to store nor invert $H$
- Quasi-Newton methods
  - L-BFGS: uses low-rank approximation of $H^{-1}$
  - Hessian-free (i.e. truncated Newton): (1) minimize $q_w(z)$ with conjugate gradient method; (2) computes $Hz$ directly using finite-difference: $Hz = \lim_{\epsilon \to 0} \frac{\nabla L(w+\epsilon z) - \nabla L(w)}{\epsilon}$

# Hessian-free optimization applied to Deep Learning

- [Martens, 2010] describes some important modifications/settings to make Hessian-free methods work for Deep Learning
- Experiments:

  (Random initialization + 2nd-order Hessian-free optimizer)

  gives lower training error than

  (Pre-training initialization + 1-order optimizer).

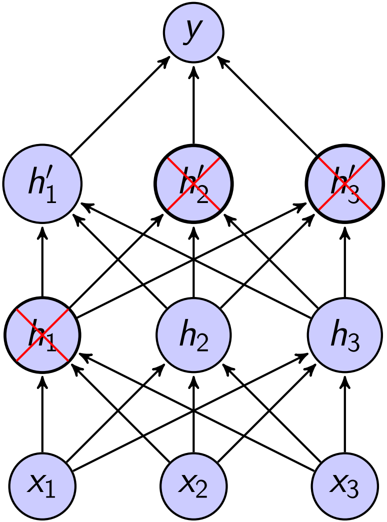
- Nice results in Recurrent Nets too [Martens and Sutskever, 2011]

# Today's Topics

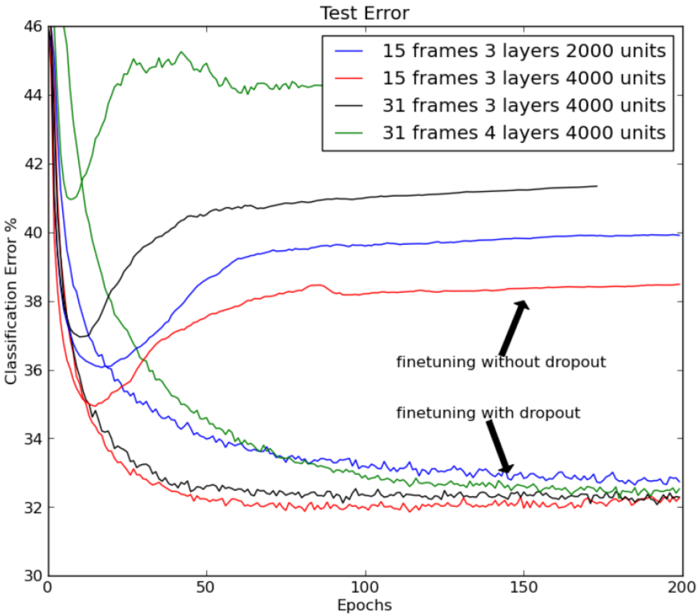
1. Hessian-free optimization [Martens, 2010]
2. Dropout regularization [Hinton et al., 2012]
3. Large-scale distributed training [Dean et al., 2012]
4. Hyper-parameter search [Bergstra et al., 2011]

# Dropout [Hinton et al., 2012]

- Each time we present $x^{(m)}$, randomly delete each hidden node with 0.5 probability
- This is like sampling from $2^{|h|}$ different architectures
- At test time, use all nodes but halve the weights
- Effect: Reduce overfitting by preventing "co-adaptation"; ensemble model averaging



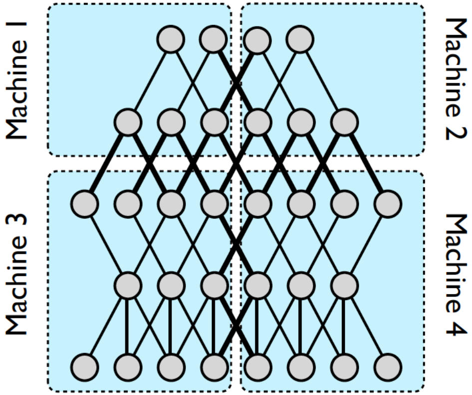
# Some Results: TIMIT phone recognition



# Today's Topics

1. Hessian-free optimization [Martens, 2010]
2. Dropout regularization [Hinton et al., 2012]
3. Large-scale distributed training [Dean et al., 2012]
4. Hyper-parameter search [Bergstra et al., 2011]

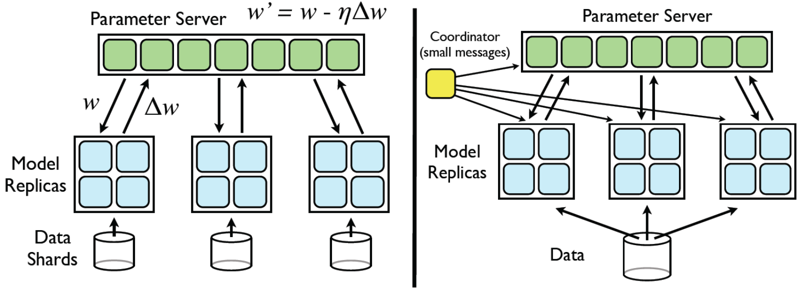
# Model Parallelism




- Deep net is stored and processed on multiple cores (multi-thread) or machines (message passing)
- Performance benefit depends on connectivity structure vs. computational demand

Figure from [Dean et al., 2012]

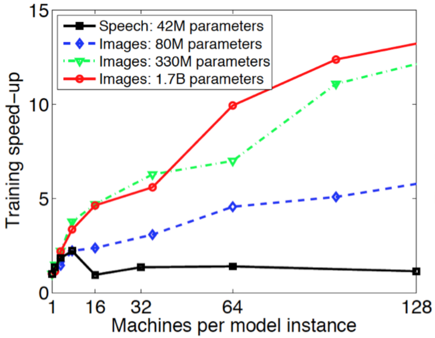
# Data Parallelism

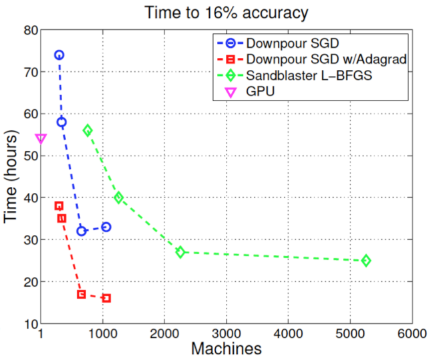



- Left: Asynchronous SGD
  - Training data is partitioned; different model per shard
  - Each model pushes and pulls gradient/weight info from parameter server asynchronously* $\rightarrow$ robust to machine failures
  - Note gradient updates may be out-of-order and weights may be out-of-date. But it works! (c.f. [Langford et al., 2009])
  - AdaGrad learning rate is beneficial in practice
- Right: Distributed L-BFGS

*More precisely, each machine within the model communicates with the relevant parameter server. (Figure from [Dean et al., 2012])

# Performance Analysis




- Left: Exploiting model parallelism on a single data shard, up to 12x measured for sparse nets (Images) but diminishing returns. For dense nets (Speech), max at 8 machines.
- Right: Exploiting data parallelism also, how much time and how many machines are needed to achieve 16% test accuracy (Speech)?

Figure from [Dean et al., 2012]

# Today's Topics

1. Hessian-free optimization [Martens, 2010]
2. Dropout regularization [Hinton et al., 2012]
3. Large-scale distributed training [Dean et al., 2012]
4. Hyper-parameter search [Bergstra et al., 2011]

# Hyper-parameter search is important

- Lots of hyper-parameters!
  1. Number of layers
  2. Number of nodes per layer
  3. SGD learning rate
  4. Regularization constant
  5. Mini-batch size
  6. Type of non-linearity
  7. Type of distribution for random initialization

# Hyper-parameter search is important

- Lots of hyper-parameters!
  1. Number of layers
  2. Number of nodes per layer
  3. SGD learning rate
  4. Regularization constant
  5. Mini-batch size
  6. Type of non-linearity
  7. Type of distribution for random initialization

- It's important to invest in finding good settings for your data
  - could be the difference between a winning system vs. useless system

## Approaches to hyper-parameter search

1. Grid search

# Approaches to hyper-parameter search

1. Grid search
2. Random search

# Approaches to hyper-parameter search

1. Grid search
2. Random search
3. Manual search, a.k.a Graduate Student Descent (GSD)

# Approaches to hyper-parameter search

1. Grid search
2. Random search
3. Manual search, a.k.a Graduate Student Descent (GSD)
4. Treat search itself as a meta machine learning problem [Bergstra et al., 2011]
    - Input $x$ = space of hyper-parameters
    - Output $y$ = validation error after training with given hyper-parameters

# Hyper-parameter search as machine learning problem

- Input $x$ = space of hyper-parameters
- Output $y$ = validation error after training with given hyper-parameters

1. Computing $y$ is expensive, so we learn a function f(x) that can predict it based on past $(x, y)$ pairs
   - e.g. Linear regression
   - e.g. Gaussian Process, Parzen Estimator [Bergstra et al., 2011]

# Hyper-parameter search as machine learning problem

- Input $x$ = space of hyper-parameters
- Output $y$ = validation error after training with given hyper-parameters

1. Computing $y$ is expensive, so we learn a function f(x) that can predict it based on past $(x, y)$ pairs
   - e.g. Linear regression
   - e.g. Gaussian Process, Parzen Estimator [Bergstra et al., 2011]
2. Try the hyper-parameter setting $x^* = \arg\min_x f(x)$, or some variant

# Hyper-parameter search as machine learning problem

- Input $x$ = space of hyper-parameters
- Output $y$ = validation error after training with given hyper-parameters

1. Computing $y$ is expensive, so we learn a function f(x) that can predict it based on past $(x, y)$ pairs
   - e.g. Linear regression
   - e.g. Gaussian Process, Parzen Estimator [Bergstra et al., 2011]
2. Try the hyper-parameter setting $x^* = \arg\min_x f(x)$, or some variant
3. Repeat steps 1-2 until you solve AI!

# References I


Bergstra, J., Bardenet, R., Bengio, Y., and Kégel, B. (2011).
Algorithms for hyper-parameter optimization.
In *Proc. Neural Information Processing Systems 24 (NIPS2011)*.

Dean, J., Corrado, G. S., Monga, R., Chen, K., Devin, M., Le, Q. V., Mao, M. Z., Ranzato, M., Senior, A., Tucker, P., Yang, K., and Ng, A. Y. (2012).
Large scale distributed deep networks.
In *Neural Information Processing Systems (NIPS)*.

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2012).
Improving neural networks by preventing co-adaptation of feature detectors.
*CoRR*, abs/1207.0580.

# References II


Langford, J., Smola, A., and Zinkevich, M. (2009).
Slow learners are fast.
In *NIPS*.

Martens, J. (2010).
Deep learning via Hessian-free optimization.
In *Proceedings of the 27th International Conference on Machine Learning (ICML)*.

Martens, J. and Sutskever, I. (2011).
Learning recurrent neural networks with hessian-free optimization.
In *Proceedings of the 28th International Conference on Machine Learning (ICML)*.