Optimization, Methodology and Application Challenges in Deep Learning

Deep Learning Decal

Hosted by Machine Learning at Berkeley

Agenda

Background

The Optimization Perspective

- We are optimizing performance measure $\mathcal{P}$ assumed to be indirectly improved when minimizing loss function $J(\theta)$.

The Optimization Perspective

- We are optimizing performance measure $\mathcal{P}$ assumed to be indirectly improved when minimizing loss function $J(\theta)$.
- Ideally: solve optimization problem below, but true distribution of $J(\theta)$ is unknown:

$$\min_{\theta} \mathbb{E}_{\theta} J(h(x, \theta), y)$$

The Optimization Perspective

- We are optimizing performance measure $\mathcal{P}$ assumed to be indirectly improved when minimizing loss function $J(\theta)$.
- Ideally: solve optimization problem below, but true distribution of $J(\theta)$ is unknown:

$$\min_{\theta} \mathbb{E}_{\theta} J(h(x, \theta), y)$$

- Instead minimize empirical risk given by test set:

$$\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} J(h(x^{(i)}, \theta), y^{(i)})$$

Batch Size Justifications

Standard error from mean given n samples is proportional to $\frac{\sigma}{\sqrt{n}}$

Batch Size Justifications

Standard error from mean given n samples is proportional to $\frac{\sigma}{\sqrt{n}}$

- Improvement in error scales sublinearly with batch size while calculation cost scales linearly.

Batch Size Justifications

Standard error from mean given n samples is proportional to $\frac{\sigma}{\sqrt{n}}$

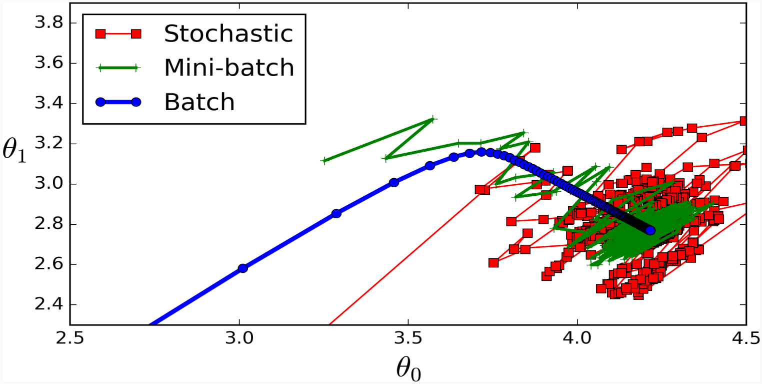
- Improvement in error scales sublinearly with batch size while calculation cost scales linearly.
- Small batch sizes underuse multicore architectures and GPU processing optimized for batches that are powers of 2

Batch Size Justifications

Standard error from mean given n samples is proportional to $\frac{\sigma}{\sqrt{n}}$

- Improvement in error scales sublinearly with batch size while calculation cost scales linearly.
- Small batch sizes underuse multicore architectures and GPU processing optimized for batches that are powers of 2
- Typical batch sizes: powers of 2 between 32 and 256. Small batch $\rightarrow$ regularizing effect since more noise. Best generalization comes from batch size being 1.

Batch Size Justifications



Challenges in Optimization

$J(\theta)$ must be differentiable to use gradient descent. Even then, issue of **ill-conditioned Hessian Matrix: H**

Challenges in Optimization

$J(\theta)$ must be differentiable to use gradient descent. Even then, issue of **ill-conditioned Hessian Matrix: H**

- Taylor series expansion of loss estimates the extra cost from gradient descent step $-\epsilon\mathbf{g}$ makes error of:

$$\frac{1}{2}\epsilon^2\mathbf{g}^T\mathbf{H}\mathbf{g} - \epsilon\mathbf{g}^T\mathbf{g}$$

$J(\theta)$ must be differentiable to use gradient descent. Even then, issue of **ill-conditioned Hessian Matrix: H**

- Taylor series expansion of loss estimates the extra cost from gradient descent step $-\epsilon\mathbf{g}$ makes error of:

$$\frac{1}{2}\epsilon^2\mathbf{g}^T\mathbf{H}\mathbf{g} - \epsilon\mathbf{g}^T\mathbf{g}$$

- If $\mathbf{H}$ becomes ill-conditioned, learning slows because ϵ shrinks accordingly.

Challenges Continued

Loss function topology can be quite pathological. When gradient is product of many terms, you may encounter

Challenges Continued

Loss function topology can be quite pathological. When gradient is product of many terms, you may encounter

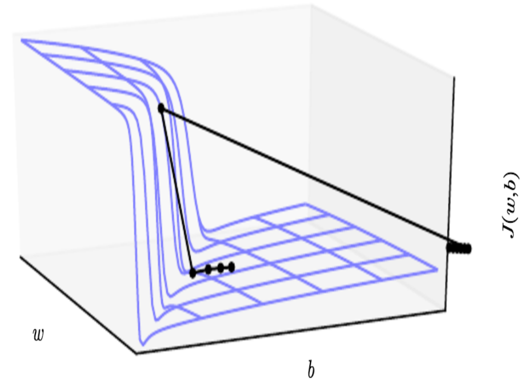
- **cliffs and exploding gradients**: extremely steep gradients that may "catapult" values so far it undoes previous learning.

Challenges Continued

Loss function topology can be quite pathological. When gradient is product of many terms, you may encounter

- **cliffs and exploding gradients**: extremely steep gradients that may "catapult" values so far it undoes previous learning.
- **vanishing gradients**: gradient calculated w.r.t. parameters "far away" (either temporally in RNN's or spatially in deep networks) vanishes.

Challenges Continued



Main Algorithms

SGD Review

SGD algorithm has been modified to make it robust against pathological loss functions. Recall the parameter update step:

SGD algorithm has been modified to make it robust against pathological loss functions. Recall the parameter update step:

- Compute gradient: $\mathbf{g} \leftarrow \frac{1}{n} \nabla_\theta \sum_i J(h(x^{(i)}, \theta), y^{(i)})$

SGD Review

SGD algorithm has been modified to make it robust against pathological loss functions. Recall the parameter update step:

- Compute gradient: $\mathbf{g} \leftarrow \frac{1}{n}\nabla_\theta \sum_i J(h(x^{(i)}, \theta), y^{(i)})$
- Update parameters: $\theta \leftarrow \theta - \epsilon \mathbf{g}$

Best Learning Rate Practices

The overall most important hyperparameter is the SGD learning rate: ϵ. Commonly, we lower ϵ as iteration number k increases.

Conditions to guarantee convergence of SGD:

Best Learning Rate Practices

The overall most important hyperparameter is the SGD learning rate: ϵ. Commonly, we lower ϵ as iteration number k increases. Conditions to guarantee convergence of SGD:

$$\sum_{k=1}^{\infty} \epsilon_k = \infty$$

$$\sum_{k=1}^{\infty} \epsilon_k^2 < \infty$$

Best Learning Rate Practices

The overall most important hyperparameter is the SGD learning rate: ϵ. Commonly, we lower ϵ as iteration number k increases. Conditions to guarantee convergence of SGD:

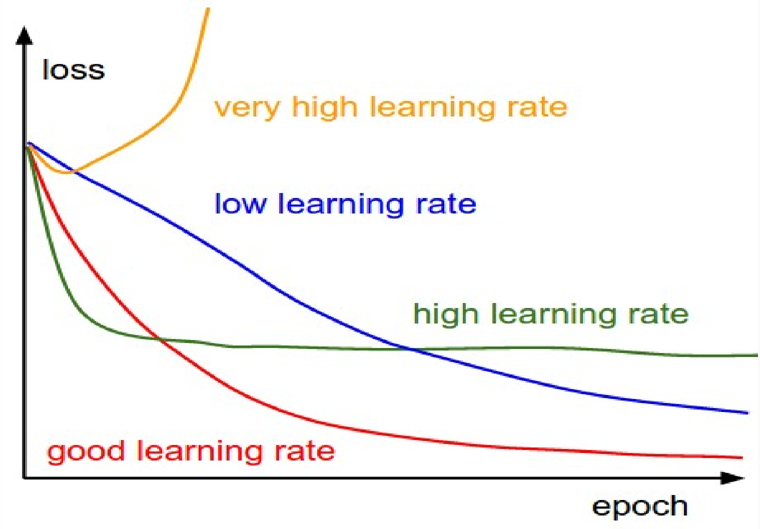
$$\sum_{k=1}^{\infty} \epsilon_k = \infty$$

$$\sum_{k=1}^{\infty} \epsilon_k^2 < \infty$$

Commonly we have ϵ decay linearly until iteration τ like so, with $\alpha = \frac{k}{\tau}$

$$\epsilon_k = (1-\alpha)\epsilon_0 + \alpha\epsilon_\tau$$

Best Learning Rate Practices



Learning Rate Best Practices Continued

- τ selected to match # of iterations needed to pass through training   100 times.*

Learning Rate Best Practices Continued

- τ selected to match # of iterations needed to pass through training   100 times.*
- ϵ_τ should be $\approx 1\%$ of ϵ_0.*

Learning Rate Best Practices Continued

- τ selected to match # of iterations needed to pass through training 100 times.*
- ϵ_τ should be $\approx 1\%$ of ϵ_0.*
- Find ϵ that does best in about 100 iterations, $\rightarrow \epsilon_0$ should be slightly higher.*

Learning Rate Best Practices Continued

- τ selected to match # of iterations needed to pass through training 100 times.*
- ϵ_τ should be $\approx 1\%$ of ϵ_0.*
- Find ϵ that does best in about 100 iterations, $\rightarrow \epsilon_0$ should be slightly higher.*
- ** Usually*

Improving on SGD

Even with best learning rate, SGD can still fail.

Improving on SGD

Even with best learning rate, SGD can still fail.

- **Idea**: Keep exponentially decaying memory of last-step gradient called **velocity**. Keeps updates "on-track" for quicker convergence.

Improving on SGD

Even with best learning rate, SGD can still fail.

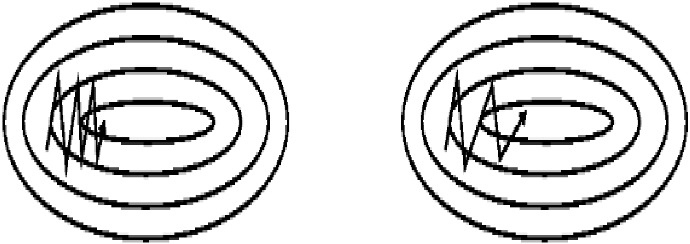
- **Idea**: Keep exponentially decaying memory of last-step gradient called **velocity**. Keeps updates "on-track" for quicker convergence.
- Compute velocity: $\mathbf{v} \leftarrow \alpha \mathbf{v} - \epsilon \nabla_\theta \frac{1}{n} \sum_i J(h(x^{(i)}, \theta), y^{(i)})$
  Same as $\mathbf{v} \leftarrow \alpha \mathbf{v} - \epsilon \mathbf{g}$

Improving on SGD

Even with best learning rate, SGD can still fail.

- **Idea**: Keep exponentially decaying memory of last-step gradient called **velocity**. Keeps updates "on-track" for quicker convergence.
- Compute velocity: $\mathbf{v} \leftarrow \alpha\mathbf{v} - \epsilon\nabla_\theta \frac{1}{n}\sum_i J(h(x^{(i)}, \theta), y^{(i)})$
  Same as $\mathbf{v} \leftarrow \alpha\mathbf{v} - \epsilon\mathbf{g}$
- Update parameters: $\theta \leftarrow \theta + \mathbf{v}$ Common values of α include 0.5, 0.9 and 0.99. Good to adapt α by increasing it (ϵ still more important).

Improving on SGD



(a) SGD without momentum

(b) SGD with momentum

Nesterov Momentum

Momentum can also be "correction step": compute gradient after taking velocity into account.

Nesterov Momentum

Momentum can also be "correction step": compute gradient after taking velocity into account.

- Compute velocity: $\mathbf{v} \leftarrow \alpha \mathbf{v} - \epsilon \nabla_\theta \frac{1}{n} \sum_i J(h(x^{(i)}, \theta + \alpha \mathbf{v}), y^{(i)})$

Nesterov Momentum

Momentum can also be "correction step": compute gradient after taking velocity into account.

- Compute velocity: $\mathbf{v} \leftarrow \alpha \mathbf{v} - \epsilon \nabla_\theta \frac{1}{n} \sum_i J(h(x^{(i)}, \theta + \alpha \mathbf{v}), y^{(i)})$
- Update parameters: $\theta \leftarrow \theta + \mathbf{v}$

Nesterov Momentum

Momentum can also be "correction step": compute gradient after taking velocity into account.

- Compute velocity: $\mathbf{v} \leftarrow \alpha \mathbf{v} - \epsilon \nabla_\theta \frac{1}{n} \sum_i J(h(x^{(i)}, \theta + \alpha \mathbf{v}), y^{(i)})$
- Update parameters: $\theta \leftarrow \theta + \mathbf{v}$

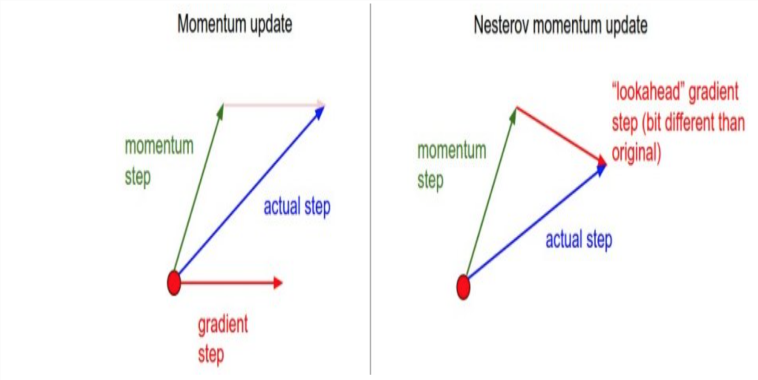
Nesterov Momentum

Momentum can also be "correction step": compute gradient after taking velocity into account.

- Compute velocity: $\mathbf{v} \leftarrow \alpha \mathbf{v} - \epsilon \nabla_\theta \frac{1}{n} \sum_i J(h(x^{(i)}, \theta + \alpha \mathbf{v}), y^{(i)})$
- Update parameters: $\theta \leftarrow \theta + \mathbf{v}$

Still, having only one ϵ for all parameters is not nuanced.

Nesterov Momentum



Algorithms with Adaptive Learning Rates

Many algs set a separated learning rate for each parameter

$\sim \frac{1}{\text{gradient}}$

Algorithms with Adaptive Learning Rates

Many algs set a separated learning rate for each parameter $\sim \frac{1}{\text{gradient}}$

- **AdaGrad** *(Adaptive Gradient Algorithm)*: Gradient is inversely proportional to square root of average sum of all historical squared values.

Algorithms with Adaptive Learning Rates

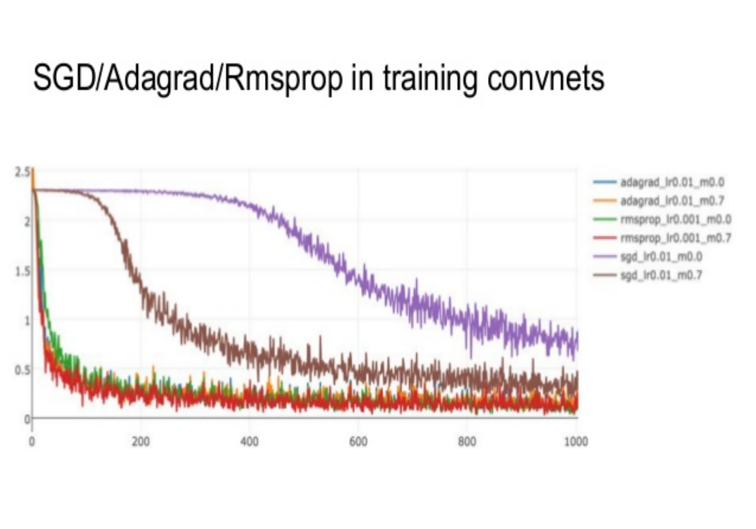
Many algs set a separated learning rate for each parameter $\sim \frac{1}{\text{gradient}}$

- **AdaGrad** *(Adaptive Gradient Algorithm)*: Gradient is inversely proportional to square root of average sum of all historical squared values.
  - Issue: First values of gradient are often bad and end up skewing the average.

Algorithms with Adaptive Learning Rates

Many algs set a separated learning rate for each parameter $\sim \frac{1}{\text{gradient}}$

- **AdaGrad** *(Adaptive Gradient Algorithm)*: Gradient is inversely proportional to square root of average sum of all historical squared values.
    - Issue: First values of gradient are often bad and end up skewing the average.
- **RMSProp** *(Root-Mean-Square Propagation)*: Gradient is inversely proportional to an exponentially weighted moving average $\rightarrow$ resolves issue with AdaGrad.


SGD/Adagrad/Rmsprop in training convnets
2.5
2
1.5
1
0.5
0
0
200
400
600
800
1000
adagrad_lr0.01_m0.0
adagrad_lr0.01_m0.7
rmsprop_lr0.001_m0.0
rmsprop_lr0.001_m0.7
sgd_lr0.01_m0.0
sgd_lr0.01_m0.7

Algorithms with Adaptive Learning Rates (Contd.)

RMSProp is very well adapted to handle non-convex problems and empirically performs better than AdaGrad.

Algorithms with Adaptive Learning Rates (Contd.)

RMSProp is very well adapted to handle non-convex problems and empirically performs better than AdaGrad.

- The most recent improvement to RMSProp is called **Adam**(*Adaptive Moments Algorithm)*.

Algorithms with Adaptive Learning Rates (Contd.)

RMSProp is very well adapted to handle non-convex problems and empirically performs better than AdaGrad.

- The most recent improvement to RMSProp is called **Adam**(*Adaptive Moments Algorithm)*.
- Adam combines RMSProp with momentum. Momentum is equivalent to the second **moment** of the gradient.

Algorithms with Adaptive Learning Rates (Contd.)

RMSProp is very well adapted to handle non-convex problems and empirically performs better than AdaGrad.

- The most recent improvement to RMSProp is called **Adam**(*Adaptive Moments Algorithm)*.
- Adam combines RMSProp with momentum. Momentum is equivalent to the second **moment** of the gradient.
- Adam also includes biases corrections to the momentum and gradient terms.

Algorithms with Adaptive Learning Rates (Contd.)

RMSProp is very well adapted to handle non-convex problems and empirically performs better than AdaGrad.

- The most recent improvement to RMSProp is called **Adam**(*Adaptive Moments Algorithm)*.
- Adam combines RMSProp with momentum. Momentum is equivalent to the second **moment** of the gradient.
- Adam also includes biases corrections to the momentum and gradient terms.

Algorithms with Adaptive Learning Rates (Contd.)

RMSProp is very well adapted to handle non-convex problems and empirically performs better than AdaGrad.

- The most recent improvement to RMSProp is called **Adam**(*Adaptive Moments Algorithm)*.
- Adam combines RMSProp with momentum. Momentum is equivalent to the second **moment** of the gradient.
- Adam also includes biases corrections to the momentum and gradient terms.

All of these are available to use as optimization algorithms in modern libraries e.g. Tensorflow.

Parameter Initialization Strategies

Learning can be doomed if we have bad parameter initialization.

- Definitely need random initialization of weight matrices (uniform or gaussian) to **break symmetry**.

Parameter Initialization Strategies

Learning can be doomed if we have bad parameter initialization.

- Definitely need random initialization of weight matrices (uniform or gaussian) to **break symmetry**.
- Need to balance between weights being too small (redundancy, undifferentiated) and too large (exploding gradients)

Parameter Initialization Strategies

Learning can be doomed if we have bad parameter initialization.

- Definitely need random initialization of weight matrices (uniform or gaussian) to **break symmetry**.
- Need to balance between weights being too small (redundancy, undifferentiated) and too large (exploding gradients)
- For a layer with m inputs, n outputs, Glorot and Bengio (2010) suggest using the **normalized initialization**:

$$W_{i,j} \sim U\left[-\sqrt{\frac{6}{m+n}}, \sqrt{\frac{6}{m+n}}\right]$$

Parameter Initialization Strategies

Learning can be doomed if we have bad parameter initialization.

- Definitely need random initialization of weight matrices (uniform or gaussian) to **break symmetry**.
- Need to balance between weights being too small (redundancy, undifferentiated) and too large (exploding gradients)
- For a layer with m inputs, n outputs, Glorot and Bengio (2010) suggest using the **normalized initialization**:

$$W_{i,j} \sim U\left[-\sqrt{\frac{6}{m+n}}, \sqrt{\frac{6}{m+n}}\right]$$

Parameter Initialization Strategies

Learning can be doomed if we have bad parameter initialization.

- Definitely need random initialization of weight matrices (uniform or gaussian) to **break symmetry**.
- Need to balance between weights being too small (redundancy, undifferentiated) and too large (exploding gradients)
- For a layer with m inputs, n outputs, Glorot and Bengio (2010) suggest using the **normalized initialization**:

$$W_{i,j} \sim U\left[-\sqrt{\frac{6}{m+n}}, \sqrt{\frac{6}{m+n}}\right]$$

Normalized initialization compromise: layers having same activation variance and having same gradient variance

Newton's Method

Newton's Method utilizes second-derivative information via Taylor Expansion of $J(\theta)$ at some point θ_0:

$$J(\theta) \approx J(\theta_0) + (\theta - \theta_0)^T \nabla_\theta J(\theta_0) + \frac{1}{2}(\theta - \theta_0)^T \mathbf{H}(\theta - \theta_0)$$

Newton's Method

Newton's Method utilizes second-derivative information via Taylor Expansion of $J(\theta)$ at some point θ_0:

$$J(\theta) \approx J(\theta_0) + (\theta - \theta_0)^T \nabla_\theta J(\theta_0) + \frac{1}{2}(\theta - \theta_0)^T \mathbf{H}(\theta - \theta_0)$$

with $\mathbf{H}$ is the Hessian of $J(\theta)$ at θ_0. The critical point can be found analytically as:

$$\theta^* = \theta_0 - \mathbf{H}^{-1} \nabla_\theta J(\theta_0)$$

- If convex (**H** is always positive definite), then this finds solution after finite iterations.

Newton's Method

Newton's Method utilizes second-derivative information via Taylor Expansion of $J(\theta)$ at some point θ_0:

$$J(\theta) \approx J(\theta_0) + (\theta - \theta_0)^T \nabla_\theta J(\theta_0) + \frac{1}{2}(\theta - \theta_0)^T \mathbf{H}(\theta - \theta_0)$$

with $\mathbf{H}$ is the Hessian of $J(\theta)$ at θ_0. The critical point can be found analytically as:

$$\theta^* = \theta_0 - \mathbf{H}^{-1} \nabla_\theta J(\theta_0)$$

- If convex ($\mathbf{H}$ is always positive definite), then this finds solution after finite iterations.
- If nonconvex, regularize $\mathbf{H}$ to be positive definite for approximate solution: $\mathbf{H} \leftarrow \mathbf{H} + \alpha \mathbf{I}$

Other Second Order Methods

Issues with Newton's Method:

Other Second Order Methods

Issues with Newton's Method:

- Regularizing with $\alpha\mathbf{I}$ is less effective with higher α.

Other Second Order Methods

Issues with Newton's Method:

- Regularizing with $\alpha\mathbf{I}$ is less effective with higher α.
- Computing $\mathbf{H}^{-1}$ has complexity $\mathcal{O}(k^3)$.

Other Second Order Methods

Issues with Newton's Method:

- Regularizing with $\alpha\mathbf{I}$ is less effective with higher α.
- Computing $\mathbf{H}^{-1}$ has complexity $\mathcal{O}(k^3)$.

Other Second Order Methods

Issues with Newton's Method:

- Regularizing with $\alpha\mathbf{I}$ is less effective with higher α.
- Computing $\mathbf{H}^{-1}$ has complexity $\mathcal{O}(k^3)$.

Other methods to address these issues:

Other Second Order Methods

Issues with Newton's Method:

- Regularizing with $\alpha\mathbf{I}$ is less effective with higher α.
- Computing $\mathbf{H}^{-1}$ has complexity $\mathcal{O}(k^3)$.

Other methods to address these issues:

- (Nonlinear) Conjugate Gradients: $\mathbf{d_t}^T\mathbf{H}\mathbf{d_{t-1}} = 0$, prevents zig-zagging, more computationally feasible

Other Second Order Methods

Issues with Newton's Method:

- Regularizing with $\alpha\mathbf{I}$ is less effective with higher α.
- Computing $\mathbf{H}^{-1}$ has complexity $\mathcal{O}(k^3)$.

Other methods to address these issues:

- (Nonlinear) Conjugate Gradients: $\mathbf{d_t}^T\mathbf{H}\mathbf{d_{t-1}} = 0$, prevents zig-zagging, more computationally feasible
- (L-)BFGS Algorithm: Approximates Hessian, takes $\mathcal{O}(k^2)$ memory, can be made $\mathcal{O}(k)$ with limited memory version.

Other Second Order Methods

Conjugate gradient

MIT 10.637 Lecture 3

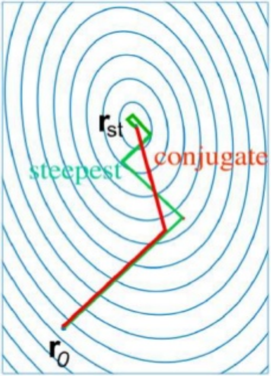
Improvement over SD method.
First step is same as SD:

$$\mathbf{q}_2 = \mathbf{q}_1 - \lambda_1 \nabla E_1$$

determined via line search.

Subsequent steps: linear combination of negative gradient and preceding search direction.

Pro: Using history, faster convergence near minimum.



Meta-Algorithms and Heuristics

Regularization - Weight Decay

Recall the L2 regularized optimization problem setup:

$$\min_{\theta} J(\theta) + ||\theta||_2^2$$

- With Gradient Descent, the regularization term will always dictate that we **shrink the magnitude** of the weights.

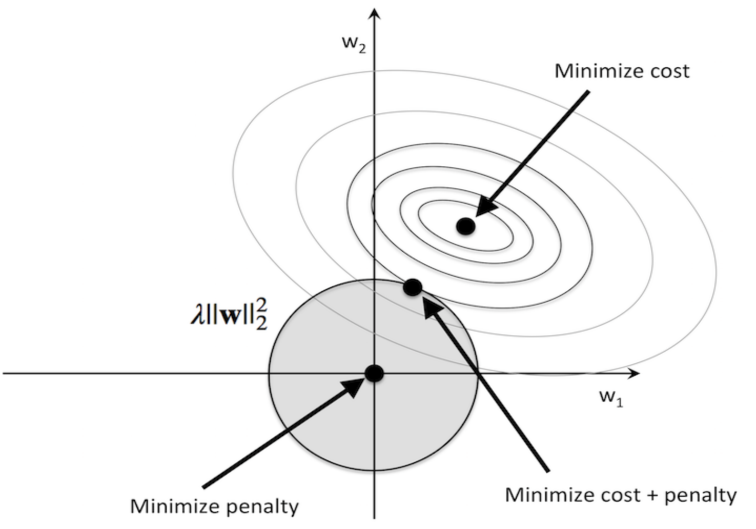
Regularization - Weight Decay

Recall the L2 regularized optimization problem setup:

$$\min_{\theta} J(\theta) + ||\theta||_2^2$$

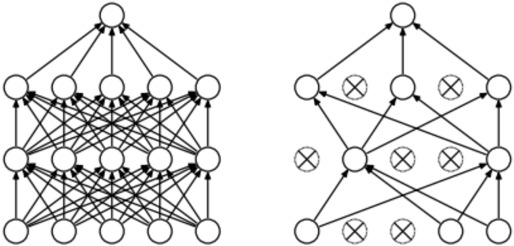
- With Gradient Descent, the regularization term will always dictate that we **shrink the magnitude** of the weights.
- Hence, a theoretically valid implementation of neural network regularization is that after each parameter update, multiply all weights by a number just below 1.

Regularization - Weight Decay



We can reduce overfitting by **preventing co-adaptations** on training data.

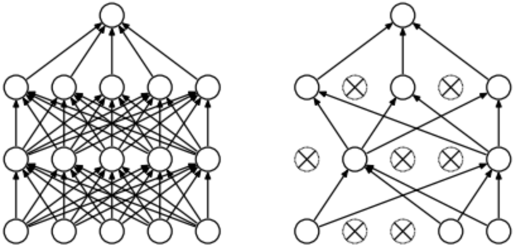
- Each unit is kept on for training with probability p. Thus, backpropagation affects only those units' weights.



(a) Standard Neural Net (b) After applying dropout.

We can reduce overfitting by **preventing co-adaptations** on training data.

- Each unit is kept on for training with probability p. Thus, backpropagation affects only those units' weights.
- Efficient form of model-averaging.



(a) Standard Neural Net (b) After applying dropout.

Batch Normalization

- Deep Neural Nets → **Many composed functions** via layers.

Batch Normalization

- Deep Neural Nets → **Many composed functions** via layers.
- Gradient descent updates parameters **assuming others stay same** → Unexpected results when all layers are updated simultaneously → difficulty choosing learning rate

Batch Normalization

- Deep Neural Nets → **Many composed functions** via layers.
- Gradient descent updates parameters **assuming others stay same** → Unexpected results when all layers are updated simultaneously → difficulty choosing learning rate

Batch Normalization

- Deep Neural Nets → **Many composed functions** via layers.
- Gradient descent updates parameters **assuming others stay same** → Unexpected results when all layers are updated simultaneously → difficulty choosing learning rate

Batch Normalization makes coordinating updates across many layers much easier.

- Automatically give a layer zero-mean, unit-variance so that next layer gets **normalized input**

Batch Normalization

- Deep Neural Nets → **Many composed functions** via layers.
- Gradient descent updates parameters **assuming others stay same** → Unexpected results when all layers are updated simultaneously → difficulty choosing learning rate

Batch Normalization makes coordinating updates across many layers much easier.

- Automatically give a layer zero-mean, unit-variance so that next layer gets **normalized input**
- Backpropagation works through this function so earlier layers are still treated correctly.

Batch Normalization (Contd.)

Suppose we have a matrix of activations, $\mathbf{M}$ at a layer. We normalize it by replacing it with $\mathbf{M}'$ like so:

$$\mathbf{M}' = \frac{\mathbf{M} - \mu}{\sigma}$$

Batch Normalization (Contd.)

Suppose we have a matrix of activations, $\mathbf{M}$ at a layer. We normalize it by replacing it with $\mathbf{M}'$ like so:

$$\mathbf{M}' = \frac{\mathbf{M} - \mu}{\sigma}$$

With μ, σ defined as:

$$\mu = \frac{1}{m} \sum_i \mathbf{M}_i$$

$$\sigma = \sqrt{\frac{1}{m} \sum_i (\mathbf{M} - \mu)_i^2}$$

- During training, compute μ, σ for each batch. During test, use averaged values.

Batch Normalization (Contd.)

Suppose we have a matrix of activations, $\mathbf{M}$ at a layer. We normalize it by replacing it with $\mathbf{M}'$ like so:

$$\mathbf{M}' = \frac{\mathbf{M} - \mu}{\sigma}$$

With μ, σ defined as:

$$\mu = \frac{1}{m} \sum_i \mathbf{M}_i$$

$$\sigma = \sqrt{\frac{1}{m} \sum_i (\mathbf{M} - \mu)_i^2}$$

- During training, compute μ, σ for each batch. During test, use averaged values.
- To maintain expressiveness, replace $\mathbf{M}$ with $\gamma \mathbf{M}' + \beta$. Mean and Stdv. come from single learned parameters rather than complex layer interactions.

Supervised Pre-Training

Usually, this is done with a **greedy** approach.

Supervised Pre-Training

Usually, this is done with a **greedy** approach.

- Start with input, hidden, output layer. Train with supervised learning signal.

Supervised Pre-Training

Usually, this is done with a **greedy** approach.

- Start with input, hidden, output layer. Train with supervised learning signal.
- Remove output layer, attach randomly initialized second hidden layer with an output layer. Supervised **training only on second layer**.

Supervised Pre-Training

Usually, this is done with a **greedy** approach.

- Start with input, hidden, output layer. Train with supervised learning signal.
- Remove output layer, attach randomly initialized second hidden layer with an output layer. Supervised **training only on second layer**.
- Not guaranteed to converge to optimal solution, but computationally much cheaper.

Supervised Pre-Training

Usually, this is done with a **greedy** approach.

- Start with input, hidden, output layer. Train with supervised learning signal.
- Remove output layer, attach randomly initialized second hidden layer with an output layer. Supervised **training only on second layer**.
- Not guaranteed to converge to optimal solution, but computationally much cheaper.

Supervised Pre-Training

Usually, this is done with a **greedy** approach.

- Start with input, hidden, output layer. Train with supervised learning signal.
- Remove output layer, attach randomly initialized second hidden layer with an output layer. Supervised **training only on second layer**.
- Not guaranteed to converge to optimal solution, but computationally much cheaper.

Can conclude process with a **fine-tuning** stage where the entire network is trained together.

Supervised Pre-Training (Contd.)

This technique has been shown to be effective in numerous contexts.

Supervised Pre-Training (Contd.)

This technique has been shown to be effective in numerous contexts.

- **Transfer Learning**: Train CNN with 8 layers on set of tasks. Initialize second network with $k < 8$ layers from first network, others random. Work on different tasks $\rightarrow$ need far less training examples.

Supervised Pre-Training (Contd.)

This technique has been shown to be effective in numerous contexts.

- **Transfer Learning**: Train CNN with 8 layers on set of tasks. Initialize second network with $k < 8$ layers from first network, others random. Work on different tasks $\rightarrow$ need far less training examples.
- **Teacher-Student Model**: Train a wide, shallow network first. Train thin, deep network second.

Supervised Pre-Training (Contd.)

This technique has been shown to be effective in numerous contexts.

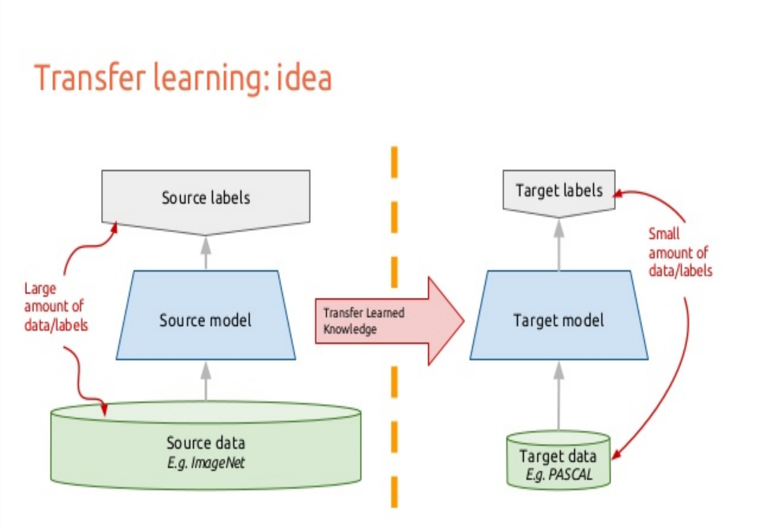
- **Transfer Learning**: Train CNN with 8 layers on set of tasks. Initialize second network with $k < 8$ layers from first network, others random. Work on different tasks $\rightarrow$ need far less training examples.
- **Teacher-Student Model**: Train a wide, shallow network first. Train thin, deep network second.
    - Student network hard to optimize using SGD since it's deep

Supervised Pre-Training (Contd.)

This technique has been shown to be effective in numerous contexts.

- **Transfer Learning**: Train CNN with 8 layers on set of tasks. Initialize second network with $k < 8$ layers from first network, others random. Work on different tasks → need far less training examples.
- **Teacher-Student Model**: Train a wide, shallow network first. Train thin, deep network second.
    - Student network hard to optimize using SGD since it's deep
    - Student minimizes traditional loss, predicts the hidden layer state of Teacher. Student gets hints and improves overall generalization.

Supervised Pre-Training (Contd.)



Continuation Methods

Objective functions are high-dimensional and non-convex. We ease optimization by **blurring** $J(\theta)$.

Continuation Methods

Objective functions are high-dimensional and non-convex. We ease optimization by **blurring** $J(\theta)$.

- Create new cost functions $\{J^0, ..., J^n\}$ with $J^{(i+1)}$ harder than $J^{(i)}$, closer to true $J(\theta)$.

Continuation Methods

Objective functions are high-dimensional and non-convex. We ease optimization by **blurring** $J(\theta)$.

- Create new cost functions $\{J^0, ..., J^n\}$ with $J^{(i+1)}$ harder than $J^{(i)}$, closer to true $J(\theta)$.
- Easier function means better behaved, closer to convex over more θ space.

Continuation Methods

Objective functions are high-dimensional and non-convex. We ease optimization by **blurring** $J(\theta)$.

- Create new cost functions $\{J^0, ..., J^n\}$ with $J^{(i+1)}$ harder than $J^{(i)}$, closer to true $J(\theta)$.
- Easier function means better behaved, closer to convex over more θ space.
- Train on each $J^{(i)}$ until close to convergence, move to $J^{(i+1)}$.

Continuation Methods

Objective functions are high-dimensional and non-convex. We ease optimization by **blurring** $J(\theta)$.

- Create new cost functions $\{J^0, ..., J^n\}$ with $J^{(i+1)}$ harder than $J^{(i)}$, closer to true $J(\theta)$.
- Easier function means better behaved, closer to convex over more θ space.
- Train on each $J^{(i)}$ until close to convergence, move to $J^{(i+1)}$.

Continuation Methods

Objective functions are high-dimensional and non-convex. We ease optimization by **blurring** $J(\theta)$.

- Create new cost functions $\{J^0, ..., J^n\}$ with $J^{(i+1)}$ harder than $J^{(i)}$, closer to true $J(\theta)$.
- Easier function means better behaved, closer to convex over more θ space.
- Train on each $J^{(i)}$ until close to convergence, move to $J^{(i+1)}$.

Issues include computational cost to compute $J^{(i)}$ since it depends on sampling and if the function cannot be effectively be blurred.

Curriculum Learning

Curriculum Learning

Teach network with **simple concept** cost function that roughly captures objective, and then modify cost with more **complex concepts**.

- Concepts can be aspects of solution (validity, syntax).

Teach network with **simple concept** cost function that roughly captures objective, and then modify cost with more **complex concepts**.

- Concepts can be aspects of solution (validity, syntax).
- Weight concept costs with coefficients, change coefficients as simple concepts are learned.

Curriculum Learning

Teach network with **simple concept** cost function that roughly captures objective, and then modify cost with more **complex concepts**.

- Concepts can be aspects of solution (validity, syntax).
- Weight concept costs with coefficients, change coefficients as simple concepts are learned.
- Can have easier/harder examples in training data. Gradually increase proportion of hard examples.

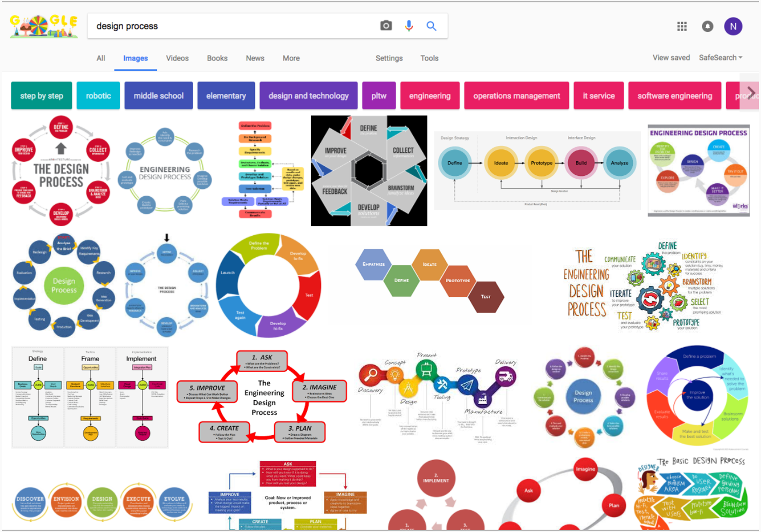
Curriculum Learning

Teach network with **simple concept** cost function that roughly captures objective, and then modify cost with more **complex concepts**.

- Concepts can be aspects of solution (validity, syntax).
- Weight concept costs with coefficients, change coefficients as simple concepts are learned.
- Can have easier/harder examples in training data. Gradually increase proportion of hard examples.
- Verified as **consistent with how humans teach**.

Methodology

Design Process Overview



Design Process Overview

Methodology is important. Having a design process will make engineering much more pleasant.

Design Process Overview

Methodology is important. Having a design process will make engineering much more pleasant.

- **Determine goals:** Error metrics and benchmarks. Should be motivated by the problem.

Design Process Overview

Methodology is important. Having a design process will make engineering much more pleasant.

- **Determine goals:** Error metrics and benchmarks. Should be motivated by the problem.
- **Establish end-to-end pipeline:** Data generation/cleaning, training, validation, saving, loading.

Design Process Overview

Methodology is important. Having a design process will make engineering much more pleasant.

- **Determine goals:** Error metrics and benchmarks. Should be motivated by the problem.
- **Establish end-to-end pipeline:** Data generation/cleaning, training, validation, saving, loading.
- **Build in units and modules:** To help debugging, diagnosing bottlenecks, under/overfitting, data defects, etc.

Design Process Overview

Methodology is important. Having a design process will make engineering much more pleasant.

- **Determine goals:** Error metrics and benchmarks. Should be motivated by the problem.
- **Establish end-to-end pipeline:** Data generation/cleaning, training, validation, saving, loading.
- **Build in units and modules:** To help debugging, diagnosing bottlenecks, under/overfitting, data defects, etc.
- **Iterate over incremental changes:** Gather more data, tune hyperparameters, change algorithms.

Performance Metrics

Choose carefully, since improving performance will motivate everything else.

Performance Metrics

Choose carefully, since improving performance will motivate everything else.

- For academic research, previous benchmarks. For industry research, thresholds of profitability/safety/human accuracy.

Performance Metrics

Choose carefully, since improving performance will motivate everything else.

- For academic research, previous benchmarks. For industry research, thresholds of profitability/safety/human accuracy.
- Weight false negatives/positives. Ex: blocking legit email much worse than letting spam in.

Performance Metrics

Choose carefully, since improving performance will motivate everything else.

- For academic research, previous benchmarks. For industry research, thresholds of profitability/safety/human accuracy.
- Weight false negatives/positives. Ex: blocking legit email much worse than letting spam in.
- **Precision:** fraction of reported 1's that were true (True positive accuracy).

Performance Metrics

Choose carefully, since improving performance will motivate everything else.

- For academic research, previous benchmarks. For industry research, thresholds of profitability/safety/human accuracy.
- Weight false negatives/positives. Ex: blocking legit email much worse than letting spam in.
- **Precision:** fraction of reported 1's that were true (True positive accuracy).
- **Recall:** fraction of 1's that were reported (1 - False negative rate)

We can plot precision and recall on a **PR curve**. Trade precision for recall by changing the detection threshold.

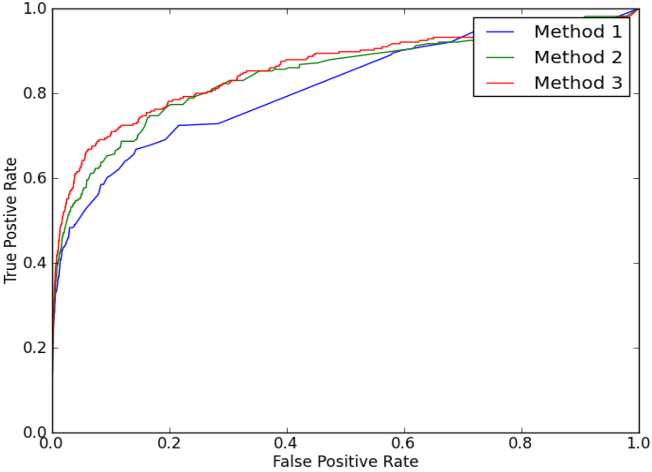
Performance Metrics (Contd.)

We can plot precision and recall on a **PR curve**. Trade precision for recall by changing the detection threshold.

- Scalar score for precision, recall is **F-score:** $\frac{2pr}{p+r}$.

We can also let the model refuse to give output if not confident. Fraction of examples for which model can give response is called **coverage.** Can trade accuracy for coverage.

Performance Metrics (Contd.)



Baseline Models, Gathering Data

Here are some tips to establish a good baseline:

- Use SGD + momentum or Adam. Batch normalization can have dramatic impact.

Baseline Models, Gathering Data

Here are some tips to establish a good baseline:

- Use SGD + momentum or Adam. Batch normalization can have dramatic impact.
- Unless millions of training examples, regularize (dropout, weight decay) from start since easy to implement.

Baseline Models, Gathering Data

Here are some tips to establish a good baseline:

- Use SGD + momentum or Adam. Batch normalization can have dramatic impact.
- Unless millions of training examples, regularize (dropout, weight decay) from start since easy to implement.
- Determine if unsupervised learning can help by making representations.

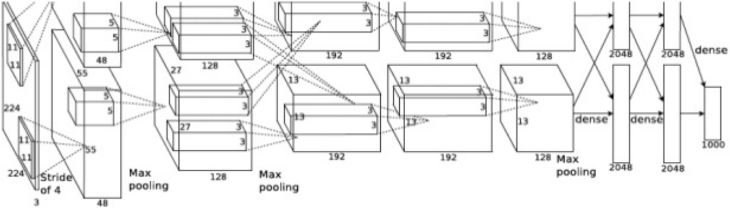
Baseline Models, Gathering Data

Here are some tips to establish a good baseline:

- Use SGD + momentum or Adam. Batch normalization can have dramatic impact.
- Unless millions of training examples, regularize (dropout, weight decay) from start since easy to implement.
- Determine if unsupervised learning can help by making representations.
- Try plotting performance for different amounts of data to extrapolate how much is needed to reach a performance level.

AlexNet

- Similar framework to LeCun'98 but:
  - Bigger model (7 hidden layers, 650,000 units, 60,000,000 params)
  - More data (10^6 vs. 10^3 images)
  - GPU implementation (50x speedup over CPU)
    - Trained on two GPUs for a week



A. Krizhevsky, I. Sutskever, and G. Hinton,
ImageNet Classification with Deep Convolutional Neural Networks, NIPS 2012

Manual Hyperparameter Tuning

Primary goal: Match **effective capacity** of model to **complexity** of task.

- **Representational capacity:** Bias-Variance, underfitting/overfitting. With all other hyperparams fixed, the error rate for one hyperparam setting follows a U-curve.

Manual Hyperparameter Tuning

Primary goal: Match **effective capacity** of model to **complexity** of task.

- **Representational capacity:** Bias-Variance, underfitting/overfitting. With all other hyperparams fixed, the error rate for one hyperparam setting follows a U-curve.
- **Learning alg/cost function match:** Learning algorithm may not effectively find a minimum of your cost function if their complexities are mismatched.

Manual Hyperparameter Tuning

Primary goal: Match **effective capacity** of model to **complexity** of task.

- **Representational capacity:** Bias-Variance, underfitting/overfitting. With all other hyperparams fixed, the error rate for one hyperparam setting follows a U-curve.
- **Learning alg/cost function match:** Learning algorithm may not effectively find a minimum of your cost function if their complexities are mismatched.
- Consider how # of hidden layer units, dropout probability, convolution kernel width, etc. **capacity**. Bias-Variance Tradeoff.

Algorithms exist to find good hyperparameter settings:

- **Grid Search**: nested for loops of hyperparameter values - search a grid of ordered tuples.

Automatic Hyperparameter Tuning

Algorithms exist to find good hyperparameter settings:

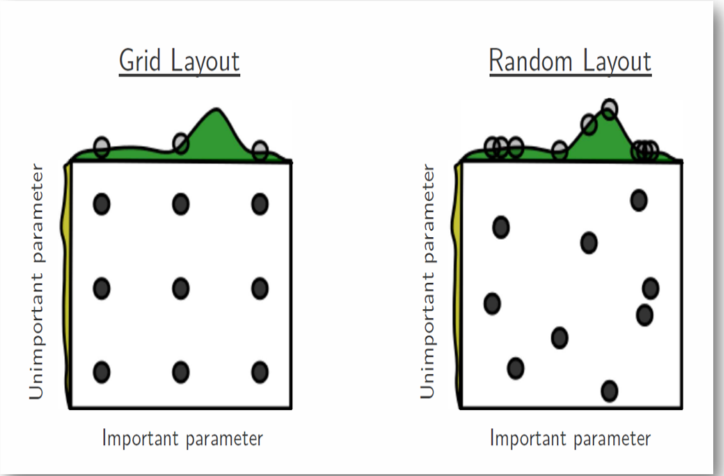
- **Grid Search**: nested for loops of hyperparameter values - search a grid of ordered tuples.
- **Random Search**: generate ordered tuples with each hyperparam value selected randomly, independently.

Automatic Hyperparameter Tuning

Algorithms exist to find good hyperparameter settings:

- **Grid Search**: nested for loops of hyperparameter values - search a grid of ordered tuples.
- **Random Search**: generate ordered tuples with each hyperparam value selected randomly, independently.
- **Bayesian Optimization**: Treat hyperparams as decision variables in meta-optimization problem. Try to maximize performance iteratively.

Automatic Hyperparameter Tuning



If things aren't working as expected,

- Look at actual outputs - image segmentations, generated audio, autoencoder reconstructions, etc.

If things aren't working as expected,

- Look at actual outputs - image segmentations, generated audio, autoencoder reconstructions, etc.
- Fit a tiny dataset - if not possible, probably bad implementation

Debugging Neural Networks

If things aren't working as expected,

- Look at actual outputs - image segmentations, generated audio, autoencoder reconstructions, etc.
- Fit a tiny dataset - if not possible, probably bad implementation
- Compare backprop derivatives to finite differences

Debugging Neural Networks

If things aren't working as expected,

- Look at actual outputs - image segmentations, generated audio, autoencoder reconstructions, etc.
- Fit a tiny dataset - if not possible, probably bad implementation
- Compare backprop derivatives to finite differences
- Measure activations of neuron layers

Application Strategies

Parallelization

Practical Deep Learning relies on large models.

- GPUs are ideal - graphics tasks rely on many simple calculations (3D $\rightarrow$ 2D)

Parallelization

Practical Deep Learning relies on large models.

- GPUs are ideal - graphics tasks rely on many simple calculations (3D $\rightarrow$ 2D)
- **Data parallelism** - put data and forward passes on different machines

Parallelization

Practical Deep Learning relies on large models.

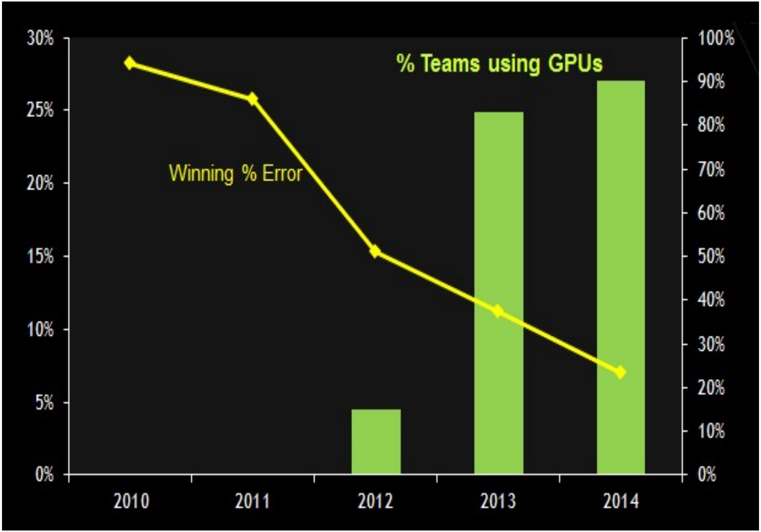
- GPUs are ideal - graphics tasks rely on many simple calculations (3D $\rightarrow$ 2D)
- **Data parallelism** - put data and forward passes on different machines
- **Model parallelism** - different machines handle different parts of model

Parallelization

Practical Deep Learning relies on large models.

- GPUs are ideal - graphics tasks rely on many simple calculations (3D $\rightarrow$ 2D)
- **Data parallelism** - put data and forward passes on different machines
- **Model parallelism** - different machines handle different parts of model
- **Asynchronous SGD** - noisier parameter updates, but faster overall progress

Application Strategies



Model Compression

- Train a model f on data (X, y) to maximize validation accuracy.

Model Compression

- Train a model f on data (X, y) to maximize validation accuracy.
- Produce new data $(X, f(X))$ which clearly has underlying function to learn.

Model Compression

- Train a model f on data (X, y) to maximize validation accuracy.
- Produce new data $(X, f(X))$ which clearly has underlying function to learn.
- Train simpler model g on $(X, f(X))$ to try and approximate f with fewer parameters.

Dynamic Structure

Can improve model performance by giving flexibility

Can improve model performance by giving flexibility

- **Cascade** of classifiers - Test for rare phenomenon efficiently by having low capacity, high recall first. Then give to high capacity, high precision afterward.

Dynamic Structure

Can improve model performance by giving flexibility

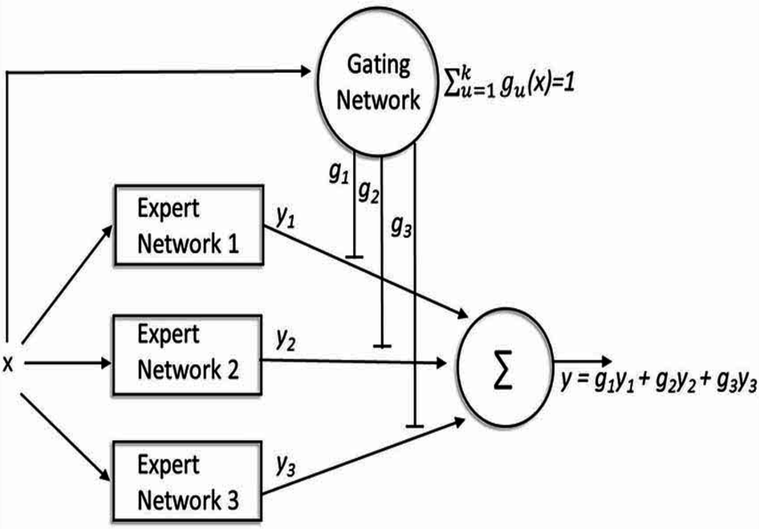
- **Cascade** of classifiers - Test for rare phenomenon efficiently by having low capacity, high recall first. Then give to high capacity, high precision afterward.
- **Mixture of Experts** - Have a *gater* neural network which selects distribution of *expert* networks to process input.

Dynamic Structure

Can improve model performance by giving flexibility

- **Cascade** of classifiers - Test for rare phenomenon efficiently by having low capacity, high recall first. Then give to high capacity, high precision afterward.
- **Mixture of Experts** - Have a *gater* neural network which selects distribution of *expert* networks to process input.
- Hard MOE (pick only one expert) will save lots of time compared to regular MOE.

Dynamic Structure



Challenges in Application

Especially in Computer Vision, NLP tasks, most pressing issue is runtime and storage constraints

Especially in Computer Vision, NLP tasks, most pressing issue is runtime and storage constraints

- Storing models on low-memory hardware (i.e. FPGA)

Challenges in Application

Especially in Computer Vision, NLP tasks, most pressing issue is runtime and storage constraints

- Storing models on low-memory hardware (i.e. FPGA)
- Having to do realtime analysis like for self-driving cars, conversational AI

Challenges in Application

Especially in Computer Vision, NLP tasks, most pressing issue is runtime and storage constraints

- Storing models on low-memory hardware (i.e. FPGA)
- Having to do realtime analysis like for self-driving cars, conversational AI
- If we want deep learning to be in IoT, important to develop efficient, fast, small implementations.

Conclusion

- Deep Learning is arguably more art than science. No hard rules, mostly heuristics.

Conclusion

- Deep Learning is arguably more art than science. No hard rules, mostly heuristics.
- Need intuition for practice, Get intuition from theory.

Conclusion

- Deep Learning is arguably more art than science. No hard rules, mostly heuristics.
- Need intuition for practice, Get intuition from theory.
- Can make huge impact by compressing, simplifying existing models.

Questions

Questions?