Advanced Training Techniques

Prajit Ramachandran
Outline

- Optimization
- Regularization
- Initialization
Optimization
Optimization Outline

- Gradient Descent
- Momentum
- RMSProp
- Adam
- Distributed SGD
- Gradient Noise
Optimization Outline

- Gradient Descent
- Momentum
- RMSProp
- Adam
- Distributed SGD
- Gradient Noise
Gradient Descent

- Goal: optimize parameters to minimize loss
- Step along the direction of steepest descent (negative gradient)

\[ \theta_t = \theta_{t-1} - \alpha \nabla_{\theta_{t-1}} f(\theta_{t-1}) \]
Gradient Descent
\[ \min_y f(y) \]
\[ \min_y f(y) \]
\[ \approx \min_y f(x) + \nabla_x f(x)^T (y - x) + (y - x)^T \mathbf{H} (y - x) \]

2\textsuperscript{nd} order Taylor series approximation around x
\[ \min_y f(y) \approx \min_y f(x) + \nabla_x f(x)^T (y - x) + (y - x)^T \mathbf{H} (y - x) \]

\[ \mathbf{H} = \begin{bmatrix}
    \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\
    \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\
    \vdots & \vdots & \ddots & \vdots \\
    \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2}
\end{bmatrix} \]

Hessian measures curvature
\[
\min_y f(y) \\
\approx \min_y f(x) + \nabla_x f(x)^T (y - x) + (y - x)^T \mathbf{H} (y - x)
\]
\[
\begin{align*}
\min_y f(y) & \\
\approx & \min_y f(x) + \nabla_x f(x)^T (y - x) + (y - x)^T \mathbf{H} (y - x) \\
\approx & \min_y f(x) + \nabla_x f(x)^T (y - x) + (y - x)^T \left( \frac{1}{t} \mathbf{I} \right) (y - x)
\end{align*}
\]

Approximate Hessian with scaled identity
\[
\min_y f(x) + \nabla_x f(x)^T (y - x) + (y - x)^T \left( \frac{1}{t} \mathbf{I} \right) (y - x)
\]
\[
\begin{align*}
\min_y f(x) + \nabla_x f(x)^T (y - x) + (y - x)^T \left( \frac{1}{t} \mathbf{I} \right) (y - x) &= 0 \\
0 &= \nabla_y (f(x) + \nabla_x f(x)^T (y - x) + (y - x)^T \left( \frac{1}{t} \mathbf{I} \right) (y - x))
\end{align*}
\]

Set gradient of function to 0 to get minimum
\[
\min_y f(x) + \nabla_x f(x)^T (y - x) + (y - x)^T \left( \frac{1}{t} \mathbf{I} \right) (y - x)
\]

\[
0 = \nabla_y \underbrace{f(x)}_{0} + \nabla_x f(x)^T (y - x) + (y - x)^T \left( \frac{1}{t} \mathbf{I} \right) (y - x)
\]

\[
= \nabla_x f(x)^T + \frac{1}{t} (y - x)
\]

Take the gradient
\[
\min_y f(x) + \nabla_x f(x)^T (y - x) + (y - x)^T \left( \frac{1}{t} I \right) (y - x)
\]

\[
0 = \nabla_y (f(x)) + \nabla_x f(x)^T (y - x) + (y - x)^T \left( \frac{1}{t} I \right) (y - x)
\]

\[
= \nabla_x f(x)^T + \frac{1}{t} (y - x)
\]

\[
y = x - t \nabla_x f(x)
\]

Solve for \( y \)
\[
\min_y f(x) + \nabla_x f(x)^T (y - x) + (y - x)^T \left( \frac{1}{t} \mathbf{I} \right) (y - x)
\]

\[
0 = \nabla_y f(x) + \nabla_x f(x)^T (y - x) + (y - x)^T \left( \frac{1}{t} \mathbf{I} \right) (y - x)
\]

\[
= \nabla_x f(x)^T + \frac{1}{t} (y - x)
\]

**Same equation!**

\[
y = x - t \nabla_x f(x)
\]

\[
\theta_t = \theta_{t-1} - \alpha \nabla_{\theta_{t-1}} f(\theta_{t-1})
\]
Computing the gradient

- Use backpropagation to compute gradients efficiently
- Need a differentiable function
  - Can’t use functions like argmax or hard binary
  - Unless using a different way to compute gradients
Stochastic Gradient Descent

- Gradient over entire dataset is impractical
- Better to take quick, noisy steps
- Estimate gradient over a mini-batch of examples
Mini-batch tips

- Use as large of a batch as possible
- Increasing batch size on GPU is essentially free up to a point
- Crank up learning rate when increasing batch size
- Trick: use small batches for small datasets
How to pick the learning rate?
Too big learning rate
Too small learning rate
How to pick the learning rate?

- Too big = diverge, too small = slow convergence
- No “one learning rate to rule them all”
- Start from a high value and keep cutting by half if model diverges
- Learning rate schedule: decay learning rate over time
The graph illustrates the relationship between the loss and the number of epochs for different learning rates.

- **Very high learning rate**: The loss increases drastically after a certain point, indicating instability and divergence.
- **Low learning rate**: The loss decreases slowly over epochs, converging gradually.
- **High learning rate**: The loss decreases rapidly initially but may converge slowly or not at all.
- **Good learning rate**: The loss decreases steadily and converges efficiently to a minimum.

The graph shows that a good learning rate balances rapid convergence with stability, whereas very high and low rates lead to poor performance.
Optimization Outline

- Gradient Descent
- Momentum
- RMSProp
- Adam
- Distributed SGD
- Gradient Noise
What will SGD do?
Zig-zagging

http://dsdeepdive.blogspot.com/2016/03/optimizations-of-gradient-descent.html
What we would like

- Avoid sliding back and forth along high curvature
- Go fast in along the consistent direction
\[ \theta_t = \theta_{t-1} - \alpha \nabla \theta_{t-1} f(\theta_{t-1}) \]

\[ \nu_t = \mu \nu_{t-1} - \alpha \nabla f(\theta_{t-1}) \]

\[ \theta_t = \theta_{t-1} + \nu_t \]
\[ v_t = \mu v_{t-1} - \alpha \nabla f(\theta_{t-1}) \]

\[ \theta_t = \theta_{t-1} + v_t \]

\[ \theta_t = \theta_{t-1} + (0 \cdot v_{t-1} - \alpha \nabla f(\theta_{t-1})) \]

Same as vanilla gradient descent
SGD with Momentum

- Move faster in directions with consistent gradient
- Damps oscillating gradients in directions of high curvature
- Friction / momentum hyperparameter $\mu$ typically set to $\{0.50, 0.90, 0.99\}$
- Nesterov’s Accelerated Gradient is a variant
Momentum

- Cancels out oscillation
- Gathers speed in direction that matters
Per parameter learning rate

- Gradients of different layers have different magnitudes
- Different units have different firing rates
- Want different learning rates for different parameters
- Infeasible to set all of them by hand
\[ \theta_t = \theta_{t-1} - \alpha \nabla \theta_{t-1} f(\theta_{t-1}) \]

\[ g_t = \sum_{\tau=1}^{t-1} (\nabla f(\theta_{\tau}))^2 \]

\[ = g_{t-1} + (\nabla f(\theta_{t-1}))^2 \]

\[ \theta_t = \theta_{t-1} - \frac{\alpha}{\sqrt{g_t} + \epsilon} \odot \nabla f(\theta_{t-1}) \]
Adagrad

- Gradient update depends on history of magnitude of gradients
- Parameters with small / sparse updates have larger learning rates
- Square root important for good performance
- More tolerance for learning rate

What happens as \( t \) increases?

\[
g_t = \sum_{\tau=1}^{t-1} (\nabla f(\theta_\tau))^2
\]

\[
= g_{t-1} + (\nabla f(\theta_{t-1}))^2
\]

\[
\theta_t = \theta_{t-1} - \alpha \frac{\nabla f(\theta_{t-1})}{\sqrt{g_t} + \epsilon}
\]
Adagrad learning rate goes to 0

- Maintain entire history of gradients
- Sum of magnitude of gradients always increasing
- Forces learning rate to 0 over time
- Hard to compensate for in advance
Don’t maintain all history

- Monotonically increasing because we hold all the history
- Instead, forget gradients far in the past
- In practice, downweight previous gradients exponentially
\[ g_t = \sum_{\tau=1}^{t-1} (\nabla f(\theta_\tau))^2 \]
\[ = g_{t-1} + (\nabla f(\theta_{t-1}))^2 \]

\[ g_t = (1 - \gamma) \sum_{\tau=1}^{t-1} \gamma^{t-1-\tau} (\nabla f(\theta_\tau))^2 \]
\[ = \gamma g_{t-1} + (1 - \gamma) \nabla (f(\theta_{t-1}))^2 \]
\[ g_t = (1 - \gamma) \sum_{\tau=1}^{t-1} \gamma^{t-1-\tau} (\nabla f(\theta_\tau))^2 \]

\[ = \gamma g_{t-1} + (1 - \gamma) \nabla(f(\theta_{t-1}))^2 \]

\[ \theta_t = \theta_{t-1} - \frac{\alpha}{\sqrt{g_t + \epsilon}} \odot \nabla f(\theta_{t-1}) \]
RMSProp

- Only cares about recent gradients
- Good property because optimization landscape changes
- Otherwise like Adagrad
- Standard gamma is 0.9

Hinton et al. 2012,
\[ \mu v_{t-1} - \alpha \nabla f(\theta_{t-1}) \quad \text{Momentum} \]

\[ \gamma g_{t-1} + (1 - \gamma) \nabla (f(\theta_{t-1}))^2 \quad \text{RMSProp} \]

\[ m_t = \beta_1 m_{t-1} + (1 - \beta_1) \nabla f(\theta_{t-1}) \]

\[ v_t = \beta_2 v_{t-1} + (1 - \beta_2) (\nabla f(\theta_{t-1}))^2 \]
\[ m_t = \beta_1 m_{t-1} + (1 - \beta_1) \nabla f(\theta_{t-1}) \]

\[ v_t = \beta_2 v_{t-1} + (1 - \beta_2) (\nabla f(\theta_{t-1}))^2 \]

\[ \theta_t = \theta_{t-1} - \frac{\alpha}{\sqrt{v_t + \epsilon}} \odot m_t \]
Adam

- Essentially, combine RMSProp and Momentum
- Includes bias correction term from initializing m and v to 0
- Default parameters are surprisingly good
- Trick: learning rate decay still helps
- Trick: Adam first then SGD
What to use

- SGD + momentum and Adam are good first steps
- Just use default parameters for Adam
- Learning rate decay always good
Optimization Outline

- Gradient Descent
- Momentum
- RMSProp
- Adam
- Distributed SGD
- Gradient Noise
How to scale beyond 1 GPU:

- Model parallelism: partition model across multiple GPUs

Dean et al. 2012. “Large scale distributed deep learning”.
Hogwild!

- Lock-free update of parameters across multiple threads
- Fast for sparse updates
- Surprisingly can work for dense updates

Data Parallelism and Async SGD

Parameter Server \[ w' = w - \eta \Delta w \]

Dean et al. 2012. “Large scale distributed deep learning”.
Async SGD

- Trivial to scale up
- Robust to individual worker failures
- Equally partition variables across parameter server
- Trick: at the start of training, slowly add more workers
Stale Gradients

$$w' = w - \eta \Delta w$$

These are gradients for $w$, not $w'$.
Stale Gradients

- Each worker has a different copy of parameters
- Using old gradients to update new parameters
- Staleness grows as more workers are added
- Hack: reject gradients that are from too far ago
Sync SGD

Parameter Server \( w' = w - \eta \Delta w \)

Model Replicas

Data Shards

Wait for all gradients before update

Chen et al. 2016. “Revisiting Distributed Synchronous SGD”
Sync SGD

- Equivalent to increasing up the batch size N times, but faster
- Crank up learning rate
- Problem: have to wait for slowest worker
- Solution: add extra backup workers, and update when N gradients received

Chen et al. 2016. “Revisiting Distributed Synchronous SGD”
Gradient Noise

- Add Gaussian noise to each gradient
- Can be a savior for exotic models

Neelakantan et al. 2016 “Adding gradient noise improves learning for very deep networks”
Anandkumar and Ge, 2016 “Efficient approaches for escaping higher order saddle points in non-convex optimization”
http://www.offconvex.org/2016/03/22/saddlepoints/
Regularization
Regularization Outline

- Early stopping
- L1 / L2
- Auxiliary classifiers
- Penalizing confident output distributions
- Dropout
- Batch normalization + variants
Regularization Outline

- Early stopping
- L1 / L2
- Auxiliary classifiers
- Penalizing confident output distributions
- Dropout
- Batch normalization + variants
Early Stopping

Error

Validation

Training

Time to stop training

Number of epochs
L1 / L2 regularization

\[ ||\theta||_1 = \sum_i |\theta_i| \]

\[ ||\theta||_2 = \sqrt{\sum_i \theta_i^2} \]
L1 / L2 regularization

- L1 encourages sparsity
- L2 discourages large weights
  - Gaussian prior on weight

http://www.efunda.com/math/hyperbolic/images/tanh_plot.gif
Auxiliary Classifiers

$Z^{(M)}$: Feature maps

$Q^{(M)}$: Responses

$W^{(M)}$: Filters

$Z^{(m)}$: Feature maps

$W^{(m)}$: Filters

Input

Output
Regularization Outline

- Early stopping
- L1 / L2
- Auxiliary classifiers
- Penalizing confident output distributions
- Dropout
- Batch normalization + variants
Penalizing confident distributions

- Do not want overconfident model
- Prefer smoother output distribution
- Invariant to model parameterization
- (1) Train towards smoother distribution
- (2) Penalize entropy

Pereyra et al. 2017 “Regularizing neural networks by penalizing confident output distributions”
True Label
\[ y_i = [0, 0, 1, 0] \]

Baseline
\[ u = [0.25, 0.25, 0.25, 0.25] \]

\[ \hat{y}_i = (1 - \epsilon) y_i + \epsilon u \]

Mixed Target
\[ \hat{y}_i = [0.01, 0.01, 0.97, 0.01] \]
When is uniform a good choice? Bad?

\[ y_i = [0, 0, 1, 0] \]

\[ u = [0.25, 0.25, 0.25, 0.25] \]

\[ \hat{y}_i = (1 - \epsilon)y_i + \epsilon u \]

\[ \hat{y}_i = [0.01, 0.01, 0.97, 0.01] \]

Szegedy et al. 2015 “Rethinking the Inception architecture for computer vision”
\[ H(p_\theta(y|x)) = - \sum p_\theta(y|x) \log p_\theta(y|x) \]
Enforce entropy to be above some threshold
Regularization Outline

- Early stopping
- L1 / L2
- Auxiliary classifiers
- Penalizing confident output distributions
- Dropout
- Batch normalization + variants
Dropout

(a) Standard Neural Net

(b) After applying dropout.

Dropout

- Complex co-adaptations probably do not generalize
- Forces hidden units to derive useful features on own
- Sampling from $2^n$ possible related networks


Present with probability $p$

(a) At training time

Always present

(b) At test time
Bayesian interpretation of dropout

- Variational inference for Gaussian processes
- Monte Carlo integration over GP posterior
- [http://mlg.eng.cam.ac.uk/yarin/blog_3d801aa532c1ce.html](http://mlg.eng.cam.ac.uk/yarin/blog_3d801aa532c1ce.html)
Dropout for RNNs

- Can dropout layer-wise connections as normal
- Recurrent connections use same dropout mask over time
- Or dropout specific portion of recurrent cell

Zaremba et al. 2014. “Recurrent neural network regularization”
Semenuita et al. 2016. “Recurrent dropout without memory loss”
Regularization Outline

● Early stopping
● L1 / L2
● Auxiliary classifiers
● Penalizing confident output distributions
● Dropout
● Batch normalization + variants
Internal Covariate Shift

- Distribution of inputs to a layer is changing during training
- Harder to train: smaller learning rate, careful initialization
- Easier if distribution of inputs stayed same
- How to enforce same distribution?

Ioffe and Szegedy, 2015. “Batch normalization: accelerating deep network training by reducing internal covariate shift”
Fighting internal covariate shift

- Whitening would be a good first step
- Would remove nasty correlations
- Problems with whitening?

Ioffe and Szegedy, 2015. “Batch normalization: accelerating deep network training by reducing internal covariate shift”
Problems with whitening

- Slow (have to do PCA for every layer)
- Cannot backprop through whitening
- Next best alternative?
Normalization

- Make mean = 0 and standard deviation = 1
- Doesn’t eliminate correlations
- Fast and can backprop through it
- How to compute the statistics?

Ioffe and Szegedy, 2015. “Batch normalization: accelerating deep network training by reducing internal covariate shift”
How to compute the statistics

- Going over the entire dataset is too slow
- Idea: the batch is an approximation of the dataset
- Compute statistics over the batch

Ioffe and Szegedy, 2015. “Batch normalization: accelerating deep network training by reducing internal covariate shift”
\[
\begin{align*}
\text{Mean} & \quad \mu_B & \leftarrow & \frac{1}{m} \sum_{i=1}^{m} x_i \\
\text{Variance} & \quad \sigma_B^2 & \leftarrow & \frac{1}{m} \sum_{i=1}^{m} (x_i - \mu_B)^2 \\
\text{Normalize} & \quad \hat{x}_i & \leftarrow & \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}
\end{align*}
\]
Distribution of an activation before normalization
Distribution of an activation after normalization
Not all distributions should be normalized

- A rare feature should not be forced to fire 50% of the time
- Let the model decide how the distribution should look
- Even undo the normalization if needed
$\hat{x}_i \leftarrow \frac{x_i - \mu B}{\sqrt{\sigma_B^2 + \epsilon}}$

$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i)$

Ioffe and Szegedy, 2015. “Batch normalization: accelerating deep network training by reducing internal covariate shift”
Test time batch normalization

- Want deterministic inference
- Different test batches will give different results
- Solution: precompute mean and variance on training set and use for inference
- Practically: maintain running average of statistics during training

Ioffe and Szegedy, 2015. “Batch normalization: accelerating deep network training by reducing internal covariate shift”
Advantages

- Enables higher learning rate by stabilizing gradients
- More resilient to parameter scale
- Regularizes model, making dropout unnecessary
- Most SOTA CNN models use BN

Ioffe and Szegedy, 2015. “Batch normalization: accelerating deep network training by reducing internal covariate shift”
Batch Norm for RNNs?

- Naive application doesn’t work
- Compute different statistics for different time steps?
- Ideally should be able to reuse existing architectures like LSTM

Laurent et al. 2015 “Batch normalized recurrent neural networks”
\[
\begin{pmatrix}
\tilde{f}_t \\
\tilde{i}_t \\
\tilde{o}_t \\
\tilde{g}_t
\end{pmatrix} = \text{BN}(W_h h_{t-1}; \gamma_h, \beta_h) + \text{BN}(W_x x_t; \gamma_x, \beta_x) + b
\]

\[
c_t = \sigma(\tilde{f}_t) \odot c_{t-1} + \sigma(\tilde{i}_t) \odot \tanh(\tilde{g}_t)
\]

\[
h_t = \sigma(\tilde{o}_t) \odot \tanh(\text{BN}(c_t; \gamma_c, \beta_c))
\]
Recurrent Batch Normalization

- Maintain independent statistics for the first T steps
- \( t > T \) uses the statistics from time T
- Have to initialize \( \gamma \) to \( \sim 0.1 \)

Batch Normalization

Layer Normalization

Ba et al. 2016. “Layer Normalization"
Advantages of LayerNorm

- Don’t have to worry about normalizing across time
- Don’t have to worry about batch size
Practical tips for regularization

- Batch normalization for feedforward structures
- Dropout still gives good performance for RNNs
- Entropy regularization good for reinforcement learning
- Don’t go crazy with regularization
Initialization
Initialization Outline

- Basic initialization
- Smarter initialization schemes
- Pretraining
Initialization Outline

- **Basic initialization**
- Smarter initialization schemes
- Pretraining
Baseline Initialization

- Weights cannot be initialized to the same value because all the gradients will be the same.
- Instead, draw from some distribution.
- Uniform from \([-0.1, 0.1]\) is a reasonable starting spot.
- Biases may need special constant initialization.
Initialization Outline

● Basic initialization
● Smarter initialization schemes
● Pretraining
He initialization for ReLU networks

- Call variance of input $\text{Var}[y_0]$ and of last layer activations $\text{Var}[y_L]$
- If $\text{Var}[y_L] >> \text{Var}[y_0]$?
- If $\text{Var}[y_L] << \text{Var}[y_0]$?
He initialization for ReLU networks

- Call variance of input $\text{Var}[y_0]$ and of last layer activations $\text{Var}[y_L]$
- If $\text{Var}[y_L] \gg \text{Var}[y_0]$, exploding activations $\rightarrow$ diverge
- If $\text{Var}[y_L] \ll \text{Var}[y_0]$, diminishing activations $\rightarrow$ vanishing gradient
- Key idea: $\text{Var}[y_L] = \text{Var}[y_0]$

He et al. 2015. “Delving deep into rectifiers: surpassing human level performance on ImageNet classification”
He initialization

$$w_l \sim \mathcal{N}\left(0, \frac{2}{n_l}\right)$$

Number of inputs to neuron

He et al. 2015. “Delving deep into rectifiers: surpassing human level performance on ImageNet classification”
\[ \frac{1}{2} \hat{\mu}_i \text{Var}[w_i] = 1 \quad \text{ours} \]

\[ \hat{\mu}_i \text{Var}[w_i] = 1 \quad \text{Xavier} \]
Identity RNN

- Basic RNN with ReLU as nonlinearity (instead of tanh)
- Initialize hidden-to-hidden matrix to identity matrix

<table>
<thead>
<tr>
<th>Methods</th>
<th>Frame error rates (dev / test)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RNN (500 neurons, 2 layers)</td>
<td>35.0 / 36.2</td>
</tr>
<tr>
<td>LSTM (250 cells, 2 layers)</td>
<td>34.5 / 35.4</td>
</tr>
<tr>
<td>iRNN (500 neurons, 2 layers)</td>
<td>34.3 / 35.5</td>
</tr>
<tr>
<td>RNN (500 neurons, 5 layers)</td>
<td>35.6 / 37.0</td>
</tr>
<tr>
<td>LSTM (250 cells, 5 layers)</td>
<td>35.0 / 36.2</td>
</tr>
<tr>
<td>iRNN (500 neurons, 5 layers)</td>
<td>33.0 / 33.8</td>
</tr>
<tr>
<td>Bidirectional RNN (500 neurons, 2 layers)</td>
<td>31.5 / 32.4</td>
</tr>
<tr>
<td>Bidirectional LSTM (250 cells, 2 layers)</td>
<td>29.6 / 30.6</td>
</tr>
<tr>
<td>Bidirectional iRNN (500 neurons, 2 layers)</td>
<td>31.9 / 33.2</td>
</tr>
<tr>
<td>Bidirectional RNN (500 neurons, 5 layers)</td>
<td>33.9 / 34.8</td>
</tr>
<tr>
<td>Bidirectional LSTM (250 cells, 5 layers)</td>
<td>28.5 / 29.1</td>
</tr>
<tr>
<td>Bidirectional iRNN (500 neurons, 5 layers)</td>
<td>28.9 / 29.7</td>
</tr>
</tbody>
</table>

Le et al. 2015. “A simple way to initialize recurrent neural networks of rectified linear units”
Initialization Outline

- Basic initialization
- Smarter initialization schemes
- Pretraining
Pretraining

- Initialize with weights from a network trained for another task / dataset
- Much faster convergence and better generalization
- Can either freeze or finetune the pretrained weights
## Pretraining for CNNs in vision

<table>
<thead>
<tr>
<th>Pretrained dataset is similar to new dataset</th>
<th>New dataset is small</th>
<th>New dataset is large</th>
</tr>
</thead>
<tbody>
<tr>
<td>Freeze weights and train linear classifier from top level features</td>
<td>Fine-tune all layers (pretrain for faster convergence and better generalization)</td>
<td></td>
</tr>
</tbody>
</table>

| Pretrained dataset is different from new dataset | Freeze weights and train linear classifier from non-top level features | Fine-tune all the layers (pretrain for improved convergence speed) |

<table>
<thead>
<tr>
<th>Method</th>
<th>Part info</th>
<th>mean Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sift+Color+SVM[45]</td>
<td>✗</td>
<td>17.3</td>
</tr>
<tr>
<td>Pose pooling kernel[49]</td>
<td>✓</td>
<td>28.2</td>
</tr>
<tr>
<td>RF[47]</td>
<td>✓</td>
<td>19.2</td>
</tr>
<tr>
<td>DPD[50]</td>
<td>✓</td>
<td>51.0</td>
</tr>
<tr>
<td>Poof[5]</td>
<td>✓</td>
<td>56.8</td>
</tr>
<tr>
<td>CNN-SVM</td>
<td>✗</td>
<td>53.3</td>
</tr>
<tr>
<td>CNNaug-SVM</td>
<td>✗</td>
<td>61.8</td>
</tr>
<tr>
<td>DPD+CNN(DeCaF)+LogReg[10]</td>
<td>✓</td>
<td>65.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>mean Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>HSV[27]</td>
<td>43.0</td>
</tr>
<tr>
<td>SIFT internal[27]</td>
<td>55.1</td>
</tr>
<tr>
<td>SIFT boundary [27]</td>
<td>32.0</td>
</tr>
<tr>
<td>HOG[27]</td>
<td>49.6</td>
</tr>
<tr>
<td>HSV+SIFT+SIFTb+HOG(MKL)[27]</td>
<td>72.8</td>
</tr>
<tr>
<td>BOW(4000)[14]</td>
<td>65.5</td>
</tr>
<tr>
<td>SPM(4000)[14]</td>
<td>67.4</td>
</tr>
<tr>
<td>FLH100[14]</td>
<td>72.7</td>
</tr>
<tr>
<td>BiCos seg[7]</td>
<td>79.4</td>
</tr>
<tr>
<td>Dense HOG+Coding+Pooling[2] w/o seg</td>
<td>76.7</td>
</tr>
<tr>
<td>Seg+Dense HOG+Coding+Pooling[2]</td>
<td>80.7</td>
</tr>
<tr>
<td>CNN-SVM w/o seg</td>
<td>74.7</td>
</tr>
<tr>
<td>CNNaug-SVM w/o seg</td>
<td>86.8</td>
</tr>
</tbody>
</table>

Razavian et al. 2014. “CNN features off-the-shelf: an astounding baseline for recognition”

<table>
<thead>
<tr>
<th>Method</th>
<th>mean Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROI + Gist[36]</td>
<td>26.1</td>
</tr>
<tr>
<td>DPM[30]</td>
<td>30.4</td>
</tr>
<tr>
<td>Object Bank[24]</td>
<td>37.6</td>
</tr>
<tr>
<td>RBow[31]</td>
<td>37.9</td>
</tr>
<tr>
<td>BoP[21]</td>
<td>46.1</td>
</tr>
<tr>
<td>miSVM[25]</td>
<td>46.4</td>
</tr>
<tr>
<td>D-Parts[40]</td>
<td>51.4</td>
</tr>
<tr>
<td>IFV[21]</td>
<td>60.8</td>
</tr>
<tr>
<td>MLrep[9]</td>
<td>64.0</td>
</tr>
<tr>
<td>CNN-SVM</td>
<td>58.4</td>
</tr>
<tr>
<td>CNNaug-SVM</td>
<td>69.0</td>
</tr>
<tr>
<td>CNN(AlexConvNet)+multiscale pooling[16]</td>
<td>68.9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>aero</td>
<td>76.7</td>
<td>82.2</td>
<td>82.5</td>
<td>88.5</td>
<td>90.1</td>
</tr>
<tr>
<td>bike</td>
<td>74.7</td>
<td>83.0</td>
<td>79.6</td>
<td>81.0</td>
<td>84.4</td>
</tr>
<tr>
<td>bird</td>
<td>53.8</td>
<td>58.4</td>
<td>64.8</td>
<td>83.5</td>
<td>86.5</td>
</tr>
<tr>
<td>boat</td>
<td>72.1</td>
<td>76.1</td>
<td>73.4</td>
<td>82.0</td>
<td>84.1</td>
</tr>
<tr>
<td>bottle</td>
<td>40.4</td>
<td>56.4</td>
<td>54.2</td>
<td>42.0</td>
<td>48.4</td>
</tr>
<tr>
<td>bus</td>
<td>71.7</td>
<td>69.1</td>
<td>75.0</td>
<td>72.5</td>
<td>73.4</td>
</tr>
<tr>
<td>car</td>
<td>83.6</td>
<td>62.2</td>
<td>77.5</td>
<td>85.3</td>
<td>86.7</td>
</tr>
<tr>
<td>cat</td>
<td>66.5</td>
<td>51.3</td>
<td>54.2</td>
<td>58.5</td>
<td>85.4</td>
</tr>
<tr>
<td>chair</td>
<td>52.5</td>
<td>51.3</td>
<td>46.2</td>
<td>59.9</td>
<td>61.3</td>
</tr>
<tr>
<td>cow</td>
<td>57.5</td>
<td>61.8</td>
<td>41.4</td>
<td>58.5</td>
<td>67.6</td>
</tr>
<tr>
<td>table</td>
<td>62.8</td>
<td>64.2</td>
<td>74.6</td>
<td>66.5</td>
<td>69.6</td>
</tr>
<tr>
<td>dog</td>
<td>51.1</td>
<td>41.3</td>
<td>75.0</td>
<td>77.8</td>
<td>84.0</td>
</tr>
<tr>
<td>horse</td>
<td>81.4</td>
<td>74.6</td>
<td>62.7</td>
<td>78.8</td>
<td>84.0</td>
</tr>
<tr>
<td>mbike</td>
<td>71.5</td>
<td>85.0</td>
<td>80.0</td>
<td>88.0</td>
<td>85.4</td>
</tr>
<tr>
<td>person</td>
<td>86.5</td>
<td>91.1</td>
<td>91.1</td>
<td>90.2</td>
<td>92.0</td>
</tr>
<tr>
<td>plant</td>
<td>36.4</td>
<td>48.1</td>
<td>53.9</td>
<td>54.8</td>
<td>56.9</td>
</tr>
<tr>
<td>sheep</td>
<td>55.3</td>
<td>61.7</td>
<td>61.0</td>
<td>71.1</td>
<td>76.7</td>
</tr>
<tr>
<td>sofa</td>
<td>60.6</td>
<td>67.7</td>
<td>67.5</td>
<td>62.6</td>
<td>67.3</td>
</tr>
<tr>
<td>train</td>
<td>80.6</td>
<td>68.3</td>
<td>83.6</td>
<td>78.7</td>
<td>89.1</td>
</tr>
<tr>
<td>tv</td>
<td>57.8</td>
<td>70.9</td>
<td>70.6</td>
<td>71.8</td>
<td>74.9</td>
</tr>
<tr>
<td>mAP</td>
<td>64.7</td>
<td>71.1</td>
<td>70.5</td>
<td>73.9</td>
<td>77.2</td>
</tr>
</tbody>
</table>
Pretraining for Seq2Seq

Ramachandran et al. 2016. “Unsupervised pretraining for sequence to sequence learning”
Progressive networks

Rusu et al 2016. “Progressive Neural Networks”
Key Takeaways

- Adam and SGD + momentum address key issues of SGD, and are good baseline optimization methods to use
- Batch norm, dropout, and entropy regularization should be used for improved performance
- Use smart initialization schemes when possible
Questions?
Appendix
Proof of He initialization
\[ y_l = W_l x_l \]

\( W \) is a \( d \)-by-\( n \) matrix
\[ y_l = W_l x_l \]

\( W \) is a \( d \)-by-\( n \) matrix

\[ \text{Var}[y_l] = \text{Var}[w_l^1 x_l^1 + \cdots + w_l^n x_l^n] \]
\[ y_l = W_l x_l \]

\( W \) is a \( d \)-by-\( n \) matrix

\[ Var[y_l] = Var[w_l^1 x_l^1 + \cdots + w_l^n x_l^n] \]

\( w, x \) are both i.i.d. and independent of each other.
\[ y_l = W_l x_l \]

\( W \) is a \( d \)-by-\( n \) matrix

\[ Var[y_l] = Var[w_l^1 x_l^1 + \cdots + w_l^n x_l^n] \]

\( w, x \) are both i.i.d. and independent of each other

\[ Var[y_l] = n_l Var[w_l x_l] \]
\[ \text{Var}[y_l] = n_l \text{Var}[w_l x_l] \]
\[ \text{Var}[y_l] = n_l \text{Var}[w_l x_l] \]

Let \( w_l \) have zero mean
\[ \text{Var}[y_l] = n_l \text{Var}[w_l x_l] \]

Let \( w_l \) have zero mean

\[ \text{Var}[ab] = E[a^2]E[b^2] + E[a]^2E[b]^2 \]
$$Var[y_l] = n_l Var[w_l x_l]$$

Let $w_l$ have zero mean


\[ \text{Var}[y_l] = n_l \text{Var}[w_l x_l] \]

Let \( w_l \) have zero mean

\[ \text{Var}[ab] = E[a^2] E[b^2] + E[a]^2 E[b]^2 \]

\[ \text{Var}[y_l] = n_l \left( E[w_l^2] E[x_l^2] + E[w_l]^2 E[x_l]^2 \right) \]

\[ \text{Var}[y_l] = n_l E[w_l^2] E[x_l^2] \]
\[ \text{Var}[y_i] = n_i E[w_i^2] E[x_i^2] \]

\[ \text{Var}[a] = E[(a - E[a])^2] \]
\[ \text{Var}[y_l] = n_l E[w_l^2] E[x_l^2] \]

\[ \text{Var}[a] = E[(a - E[a])^2] \]

\[ \text{Var}[y_l] = n_l \text{Var}[w_l] E[x_l^2] \]
\[ \text{Var}[y_l] = n_l \text{Var}[w_l] E[x_l^2] \]

\[ x_l = \max(0, y_{l-1}) \]

Let \( w_{l-1} \) have a zero-mean symmetric distribution.
\[
\text{Var}[y_l] = n_l \text{Var}[w_l] E[x_l^2]
\]
\[
x_l = \max(0, y_{l-1})
\]

Let \( w_{l-1} \) have a zero-mean symmetric distribution.

Then \( y_{l-1} \) also has a zero-mean symmetric distribution.
\[ \text{Var}[y_l] = n_l \text{Var}[w_l] E[x_l^2] \]
\[ x_l = \max(0, y_{l-1}) \]

Let \( w_{l-1} \) have a zero-mean symmetric distribution. Then \( y_{l-1} \) also has a zero-mean symmetric distribution.

\[ \text{Var}[y_{l-1}] = \int_{-\infty}^{0} y_{l-1}^2 p(y_{l-1}) dy_{l-1} + \int_{0}^{\infty} y_{l-1}^2 p(y_{l-1}) dy_{l-1} \]
\[ V \text{ar}[y_l] = n_l V \text{ar}[w_l] E[x_l^2] \]
\[ x_l = \max(0, y_{l-1}) \]

Let \( w_{l-1} \) have a zero-mean symmetric distribution.

Then \( y_{l-1} \) also has a zero-mean symmetric distribution.

\[ V \text{ar}[y_{l-1}] = \int_{-\infty}^{0} y_{l-1}^2 p(y_{l-1}) \, dy_{l-1} + \int_{0}^{\infty} y_{l-1}^2 p(y_{l-1}) \, dy_{l-1} \]

\[ V \text{ar}[y_{l-1}] = 2 \int_{0}^{\infty} y_{l-1}^2 p(y_{l-1}) \, dy_{l-1} \]
\[ \text{Var}[y_{l-1}] = 2 \int_{0}^{\infty} y_{l-1}^2 p(y_{l-1}) dy_{l-1} \]
\[ \text{Var}[y_{l-1}] = 2 \int_0^\infty y_{l-1}^2 p(y_{l-1}) \, dy_{l-1} \]

\[ \text{Var}[y_{l-1}] = 2 \int_0^\infty x_{l}^2 p(x_{l}) \, dx_{l} \]
\[ Var[y_{l-1}] = 2 \int_0^\infty y_{l-1}^2 p(y_{l-1}) dy_{l-1} \]

\[ Var[y_{l-1}] = 2 \int_0^\infty x_l^2 p(x_l) dx_l \]

\[ \frac{1}{2} Var[y_{l-1}] = E[x_l^2] \]
\[
Var[y_{l-1}] = 2 \int_0^\infty y_{l-1}^2 p(y_{l-1}) dy_{l-1}
\]

\[
Var[y_{l-1}] = 2 \int_0^\infty x_l^2 p(x_l) dx_l
\]

\[
\frac{1}{2} Var[y_{l-1}] = E[x_l^2]
\]

\[
Var[y_l] = n_l Var[w_l] E[x_l^2]
\]
\[ \text{Var}[y_{l-1}] = 2 \int_{0}^{\infty} y_{l-1}^2 p(y_{l-1}) \, dy_{l-1} \]

\[ \text{Var}[y_{l-1}] = 2 \int_{0}^{\infty} x_l^2 p(x_l) \, dx_l \]

\[ \frac{1}{2} \text{Var}[y_{l-1}] = \text{E}[x_l^2] \]

\[ \text{Var}[y_l] = n_l \text{Var}[w_l] \text{E}[x_l^2] \]

\[ \text{Var}[y_l] = \frac{1}{2} n_l \text{Var}[w_l] \text{Var}[y_{l-1}] \]
\[ Var[y_l] = \frac{1}{2} n_l Var[w_l] Var[y_{l-1}] \]
\[ Var[y_l] = \frac{1}{2} n_l \text{Var}[w_l] \text{Var}[y_{l-1}] \]

\[ Var[y_L] = Var[y_1] \prod_{l=2}^{L} \left( \frac{1}{2} n_l \text{Var}[w_l] \right) \]
\[ Var[y_l] = \frac{1}{2} n_l Var[w_l] Var[y_{l-1}] \]

\[ Var[y_L] = Var[y_1] \prod_{l=2}^{L} \left( \frac{1}{2} n_l Var[w_l] \right) \]

\[ Var[y_L] = Var[y_1] \]
\[ \text{Var}[y_l] = \frac{1}{2} n_l \text{Var}[w_l] \text{Var}[y_{l-1}] \]

\[ \text{Var}[y_L] = \text{Var}[y_1] \prod_{l=2}^{L} \left( \frac{1}{2} n_l \text{Var}[w_l] \right) \]

\[ \text{Var}[y_L] = \text{Var}[y_1] \]

\[ \frac{1}{2} n_l \text{Var}[w_l] = 1 \]
\[
\frac{1}{2} n_l \text{Var}[w_l] = 1
\]
\frac{1}{2} n_l \text{Var}[w_l] = 1

w_l \sim \mathcal{N} \left( 0, \frac{2}{n_l} \right)