

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



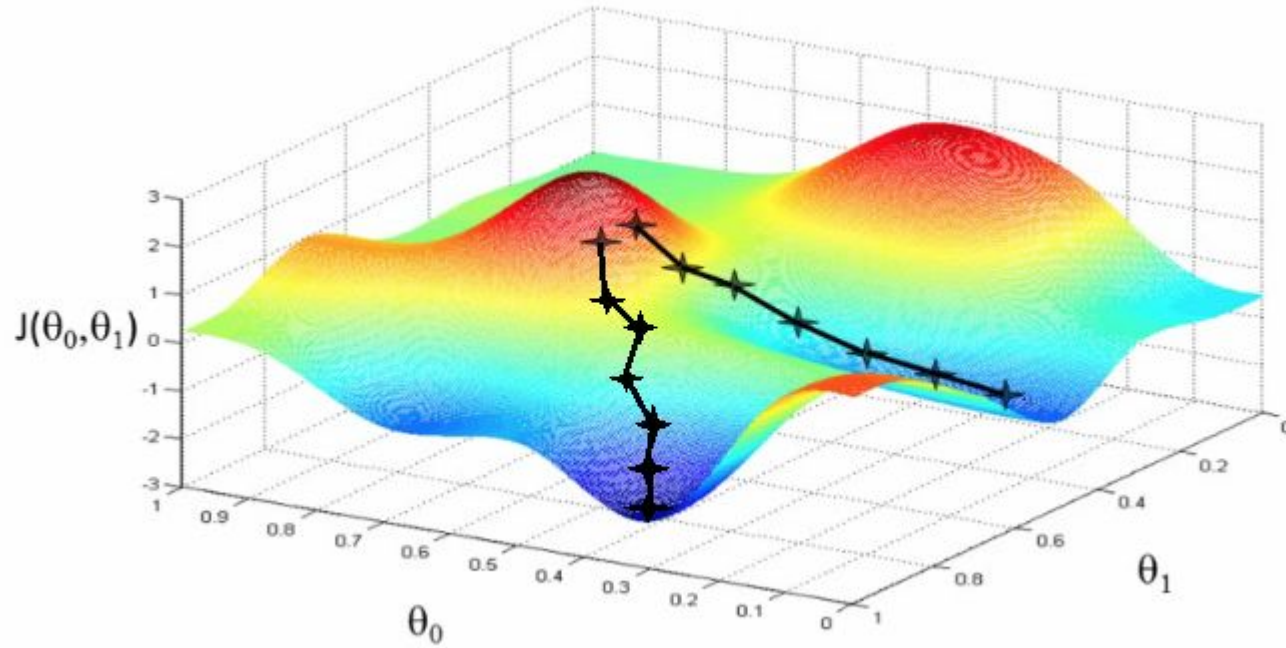
<http://weandthecolor.com/wp-content/uploads/2014/11/Hills-in-the-fog-Landscape-photography-by-Kilian-Sch%C3%B6nberger-a-photographer-from-Cologne-Germany.jpg>

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



$$\min_y f(y)$$

$$\min_y f(y)$$

$$\approx \min_y f(x) + \nabla_x f(x)^T (y - x) + (y - x)^T \mathbf{H}(y - x)$$

2nd 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

$$\min_y f(y)$$

$$\approx \min_y f(x) + \nabla_x f(x)^T (y - x) + (y - x)^T \mathbf{H} (y - x)$$

$$\begin{aligned} & \min_y f(y) \\ & \approx \min_y f(x) + \nabla_x f(x)^T (y - x) + (y - x)^T \mathbf{H} (y - x) \\ & \approx \min_y f(x) + \nabla_x f(x)^T (y - x) + (y - x)^T \left(\frac{1}{t} \mathbf{I} \right) (y - x) \end{aligned}$$


Approximate Hessian with scaled identity

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

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

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

Set gradient of function to 0 to get minimum

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

$$0 = \nabla_y \underbrace{f(x)}_0 + \underbrace{\nabla_x f(x)^T (y - x)}_{\text{red}} + \underbrace{(y - x)^T \left(\frac{1}{t} \mathbf{I} \right) (y - x)}_{\text{blue}}$$

$$= \underbrace{\nabla_x f(x)^T}_{\text{red}} + \underbrace{\frac{1}{t} (y - x)}_{\text{blue}}$$

Take the gradient

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

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

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

$$y = x - t \nabla_x f(x)$$

Solve for y

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

$$\begin{aligned} 0 &= \nabla_y (f(x) + \nabla_x f(x)^T (y - x) + (y - x)^T \left(\frac{1}{t} \mathbf{I} \right) (y - x)) \\ &= \nabla_x f(x)^T + \frac{1}{t} (y - x) \end{aligned}$$

Same
equation!

$$\boxed{y} = \boxed{x} - \boxed{t} \boxed{\nabla_x f(x)}$$

$$\boxed{\theta_t} = \boxed{\theta_{t-1}} - \boxed{\alpha} \boxed{\nabla_{\theta_{t-1}} f(\theta_{t-1})}$$

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

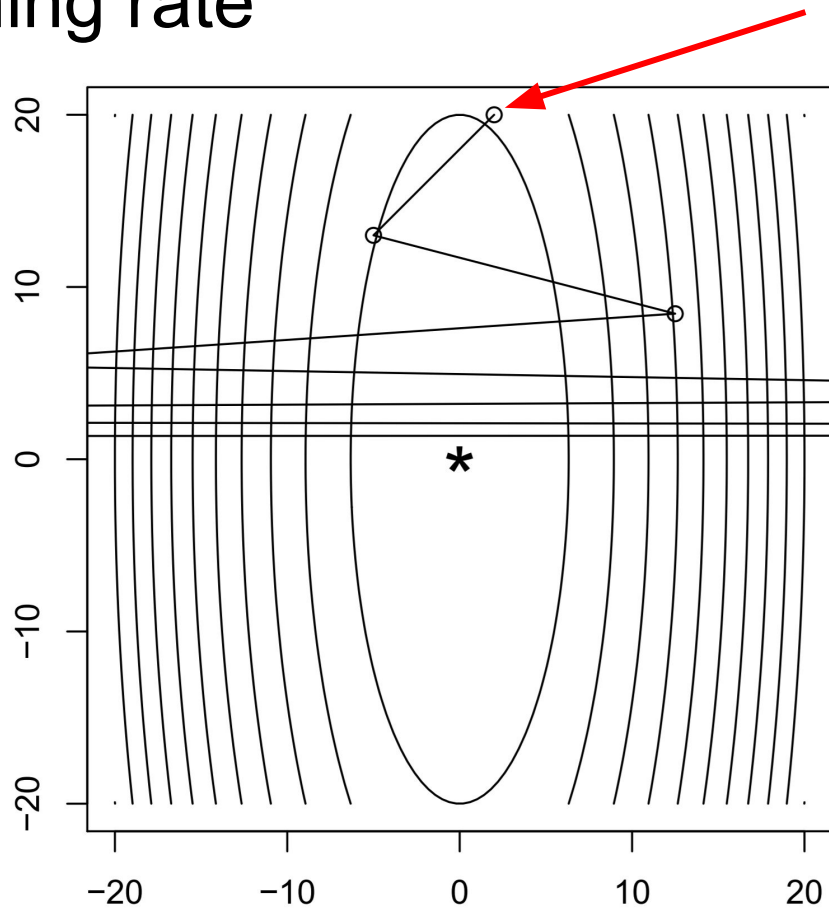
- Gradient over entire dataset is impractical
- Better to take quick, noisy steps
- Estimate gradient over a mini-batch of examples

Mini-batch tips

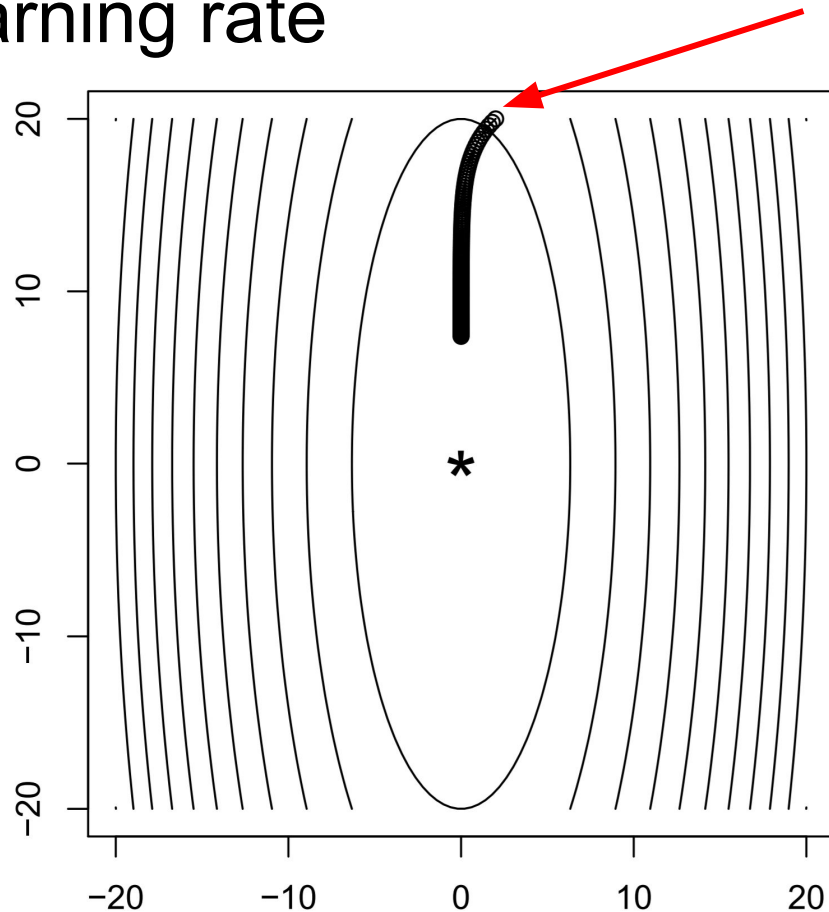
- Use as large of a batch as possible
- Increasing batch size on GPU is essentially free up to a point
- Crank up learning rate when increasing batch size
- Trick: use small batches for small datasets

How to pick the learning rate?

Too big learning rate

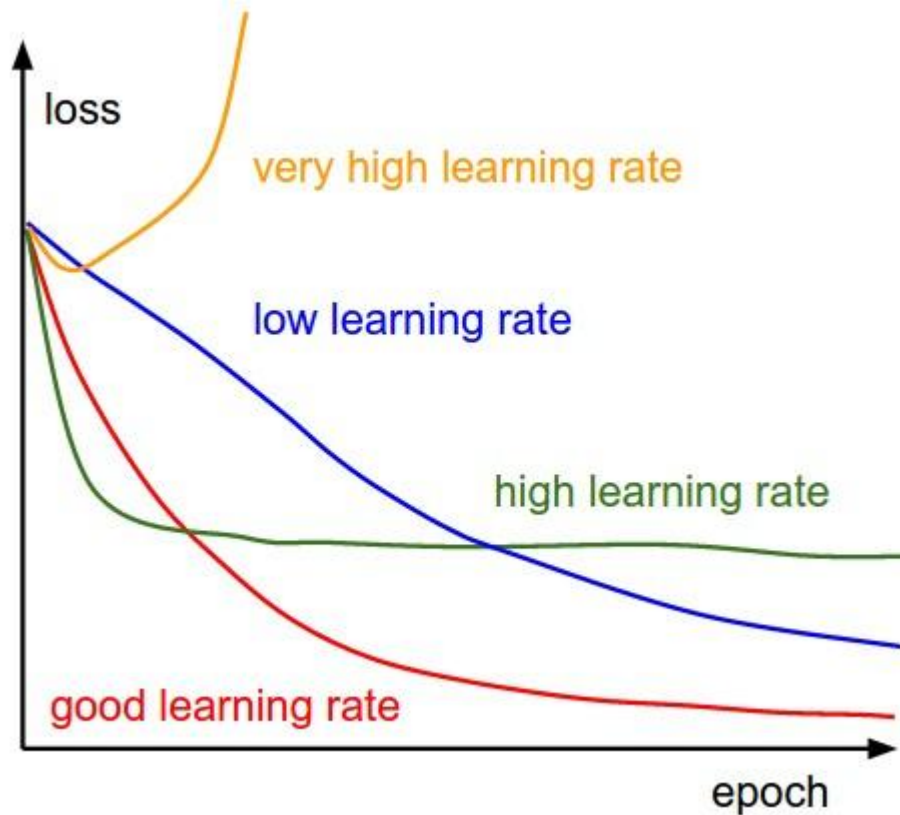


Too small learning rate



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

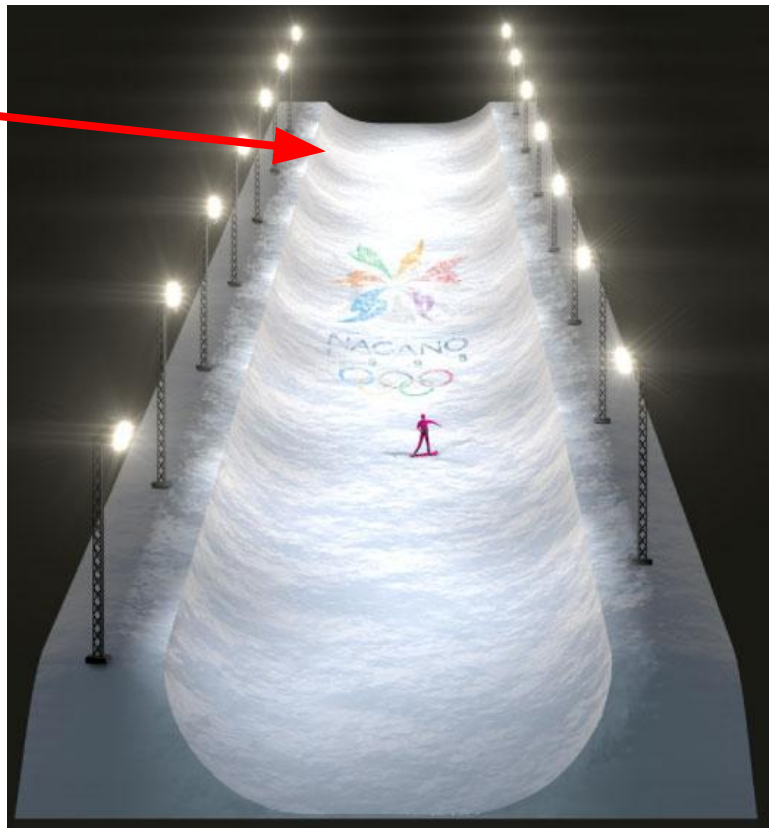


Optimization Outline

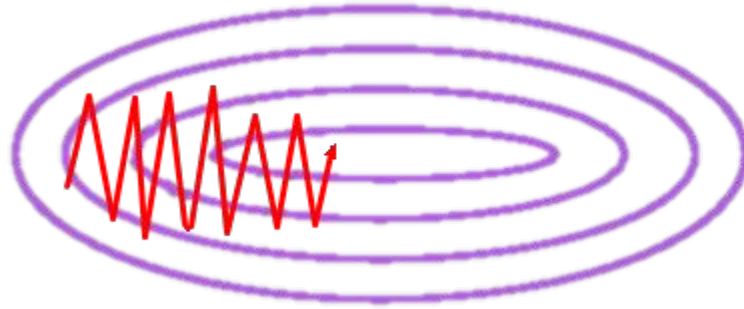
- Gradient Descent
- **Momentum**
- **RMSProp**
- **Adam**
- Distributed SGD
- Gradient Noise

What will SGD do?

Start

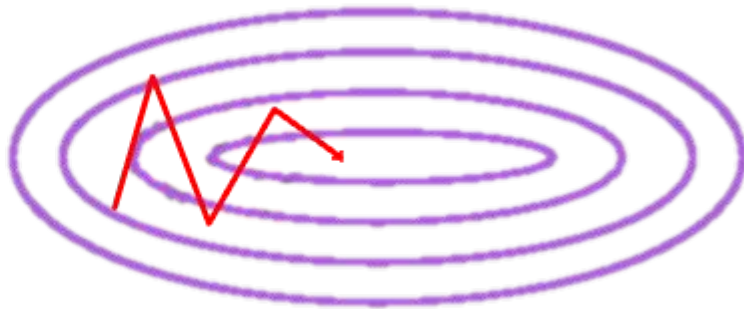


Zig-zagging



What we would like

- Avoid sliding back and forth along high curvature
- Go fast in along the consistent direction



$$\boxed{\theta_t} = \boxed{\theta_{t-1}} - \boxed{\alpha} \boxed{\nabla_{\theta_{t-1}} f(\theta_{t-1})}$$

$$\boxed{v_t} = \mu v_{t-1} - \boxed{\alpha} \boxed{\nabla f(\theta_{t-1})}$$

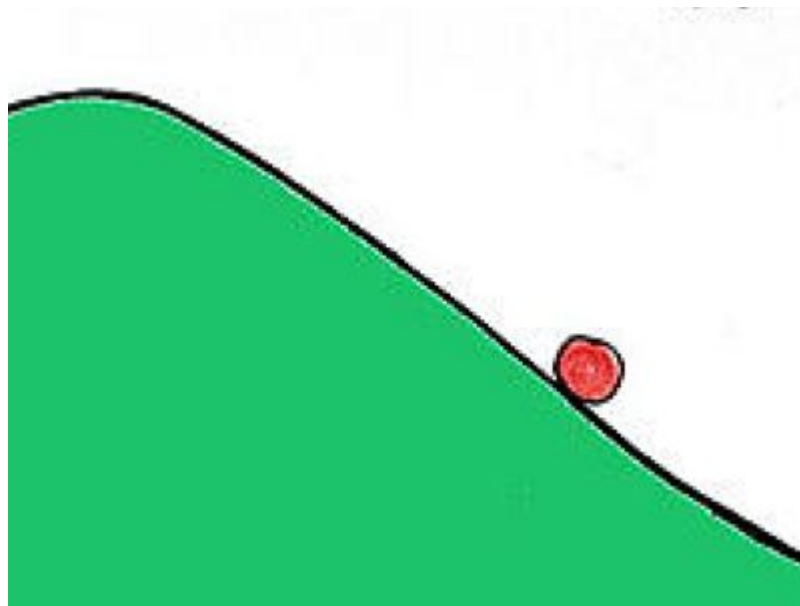
$$\boxed{\theta_t} = \boxed{\theta_{t-1}} + \boxed{v_t}$$

$$v_t = \mu v_{t-1} - \alpha \nabla f(\theta_{t-1})$$

$$\theta_t = \theta_{t-1} + v_t$$

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

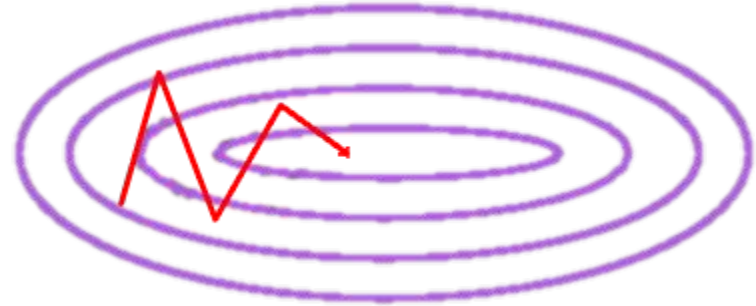
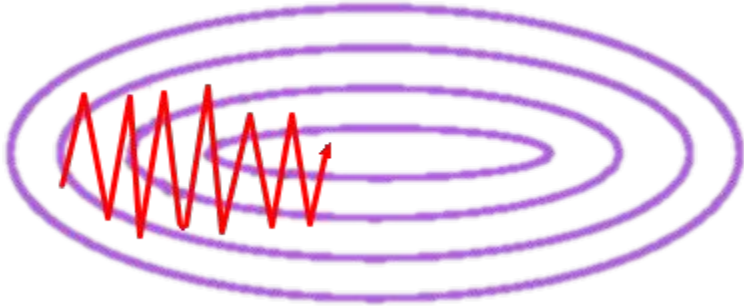
Same as vanilla gradient descent



SGD with Momentum

- Move faster in directions with consistent gradient
- Damps oscillating gradients in directions of high curvature
- Friction / momentum hyperparameter μ 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?

$$\begin{aligned} g_t &= \sum_{\tau=1}^{t-1} (\nabla f(\theta_{\tau}))^2 \\ &= g_{t-1} + (\nabla f(\theta_{t-1}))^2 \end{aligned}$$

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

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

$$\begin{aligned}
 g_t &= \sum_{\tau=1}^{t-1} (\nabla f(\theta_\tau))^2 \\
 &= g_{t-1} + (\nabla f(\theta_{t-1}))^2
 \end{aligned}$$

$$\begin{aligned}
 g_t &= (1 - \gamma) \sum_{\tau=1}^{t-1} \gamma^{t-1-\tau} (\nabla f(\theta_\tau))^2 \\
 &= \gamma g_{t-1} + (1 - \gamma) \nabla (f(\theta_{t-1}))^2
 \end{aligned}$$

$$\begin{aligned}
g_t &= (1 - \gamma) \sum_{\tau=1}^{t-1} \gamma^{t-1-\tau} (\nabla f(\theta_\tau))^2 \\
&= \gamma g_{t-1} + (1 - \gamma) \nabla (f(\theta_{t-1}))^2
\end{aligned}$$

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

RMSProp

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

$$\boxed{\mu} v_{t-1} - \boxed{\alpha} \nabla f(\theta_{t-1}) \quad \text{Momentum}$$

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

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

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

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

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

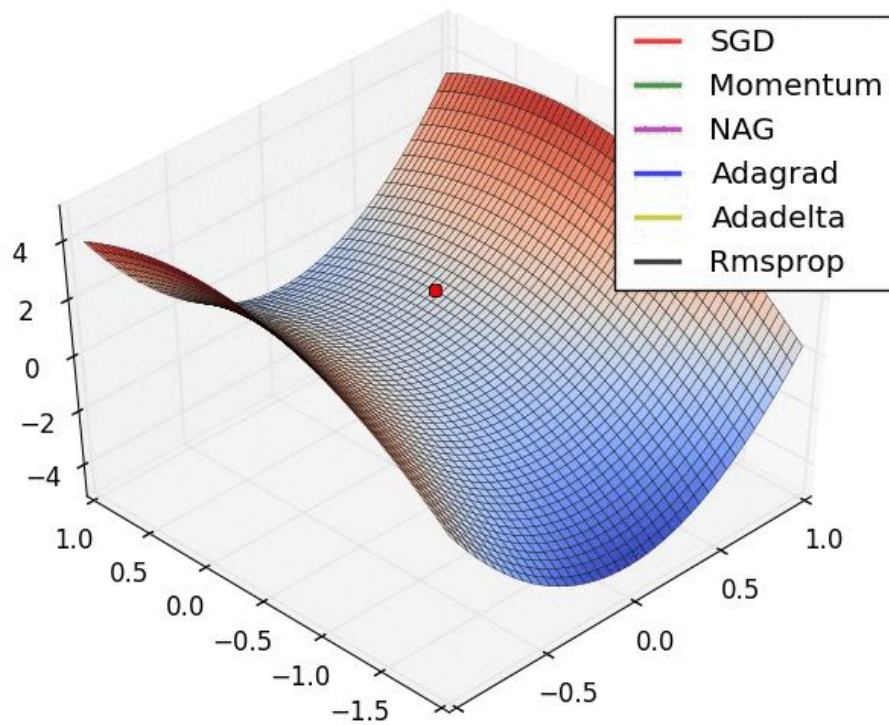
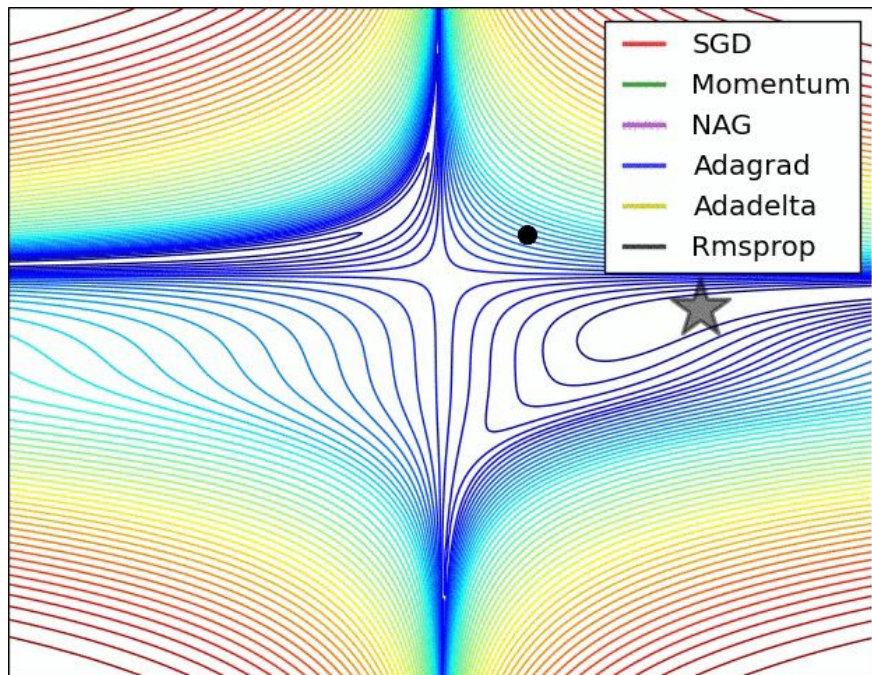
$$\theta_t = \theta_{t-1} - \frac{\alpha}{\sqrt{v_t} + \epsilon} \odot m_t$$

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

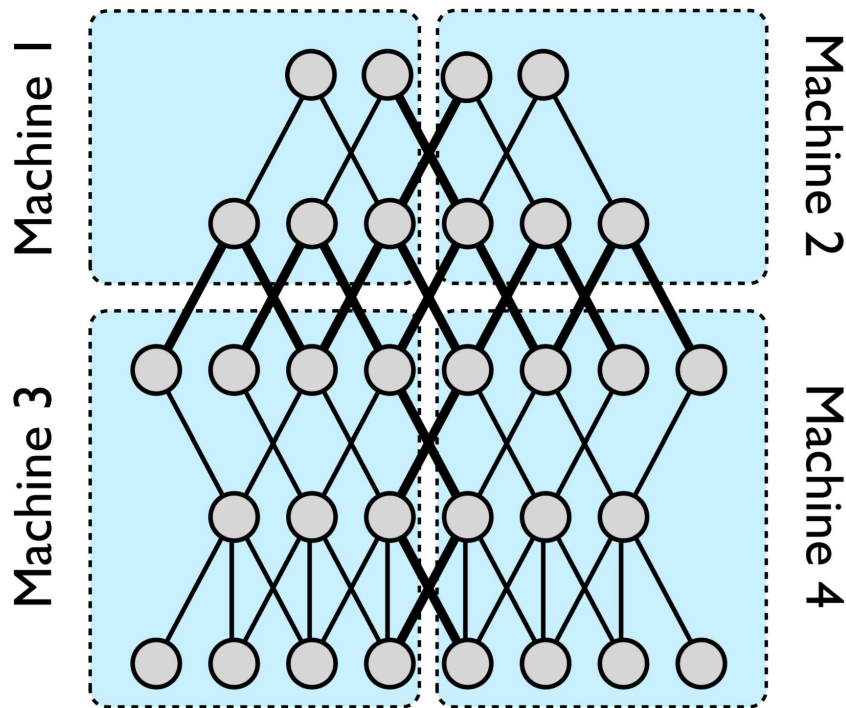


Optimization Outline

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

How to scale beyond 1 GPU:

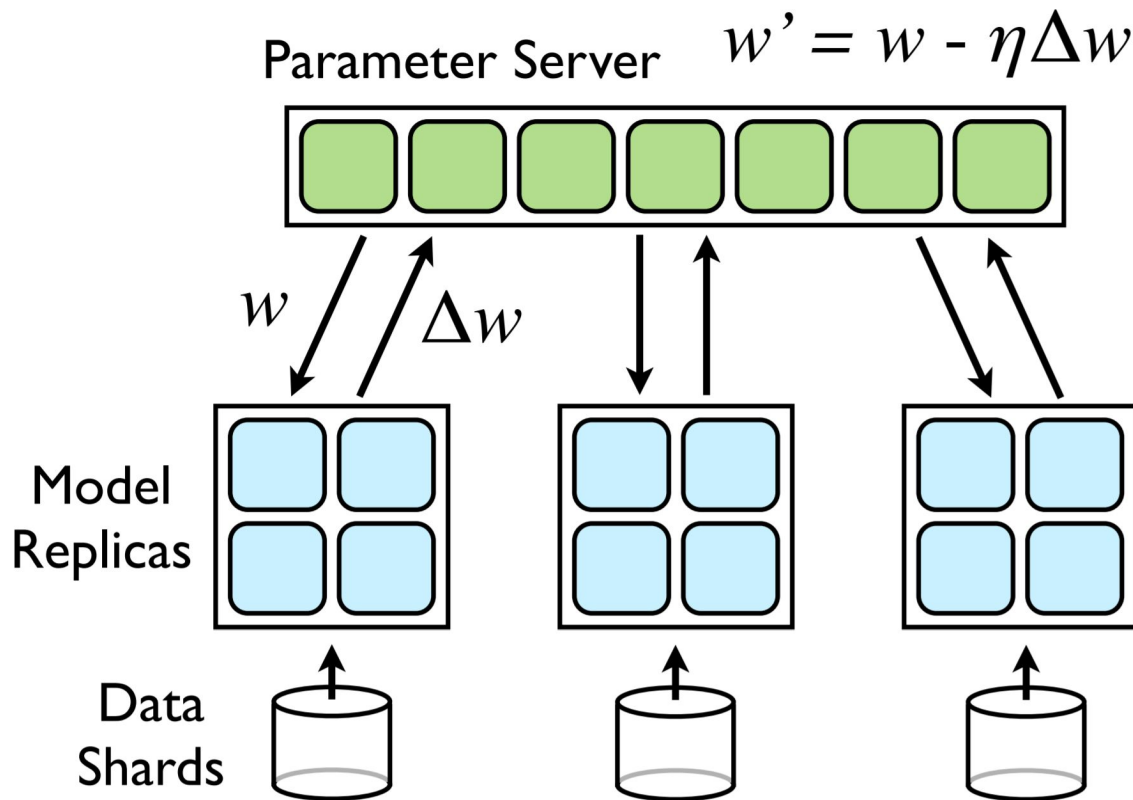
- Model parallelism: partition model across multiple GPUs



Hogwild!

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

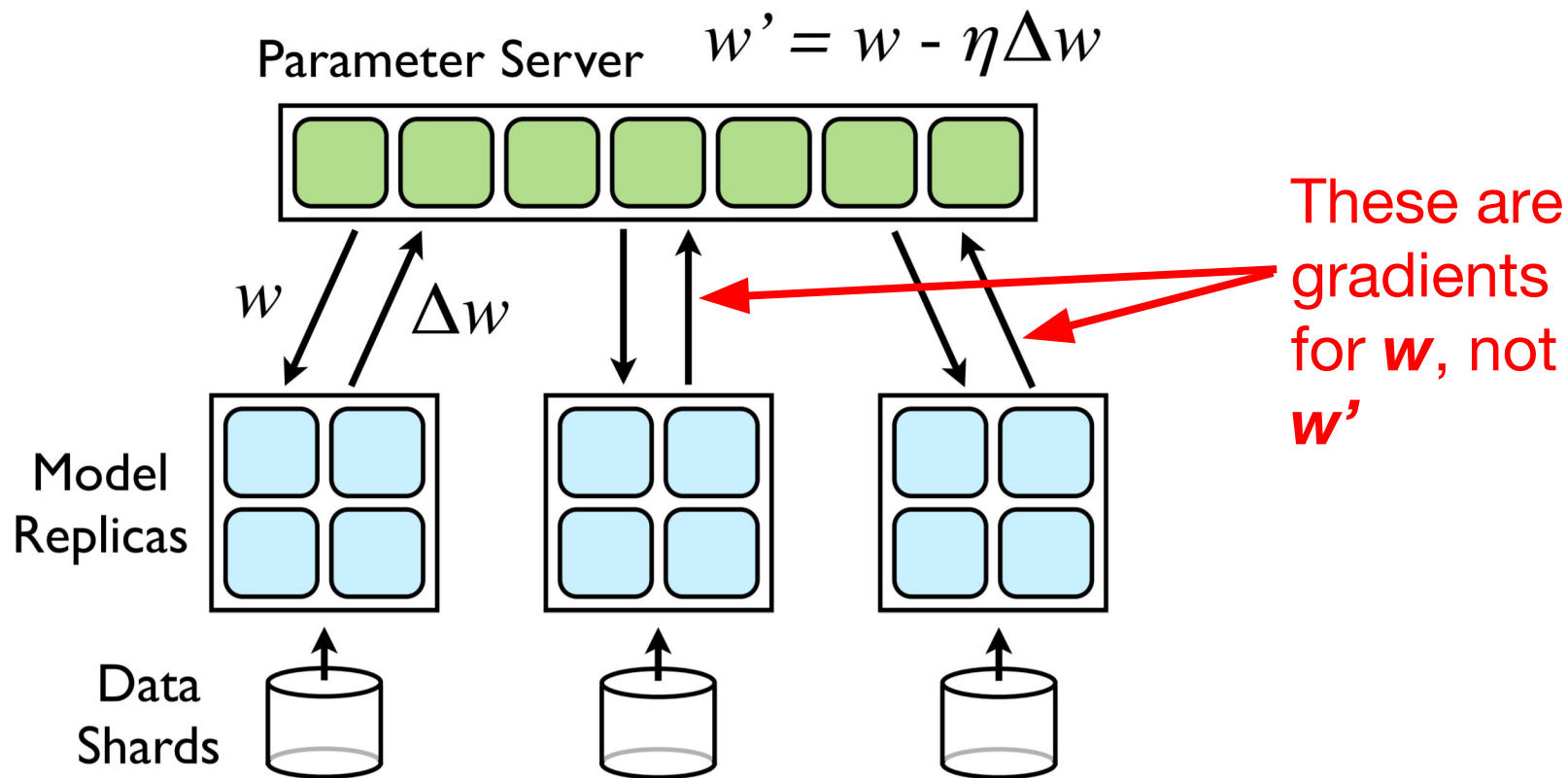
Data Parallelism and Async SGD



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

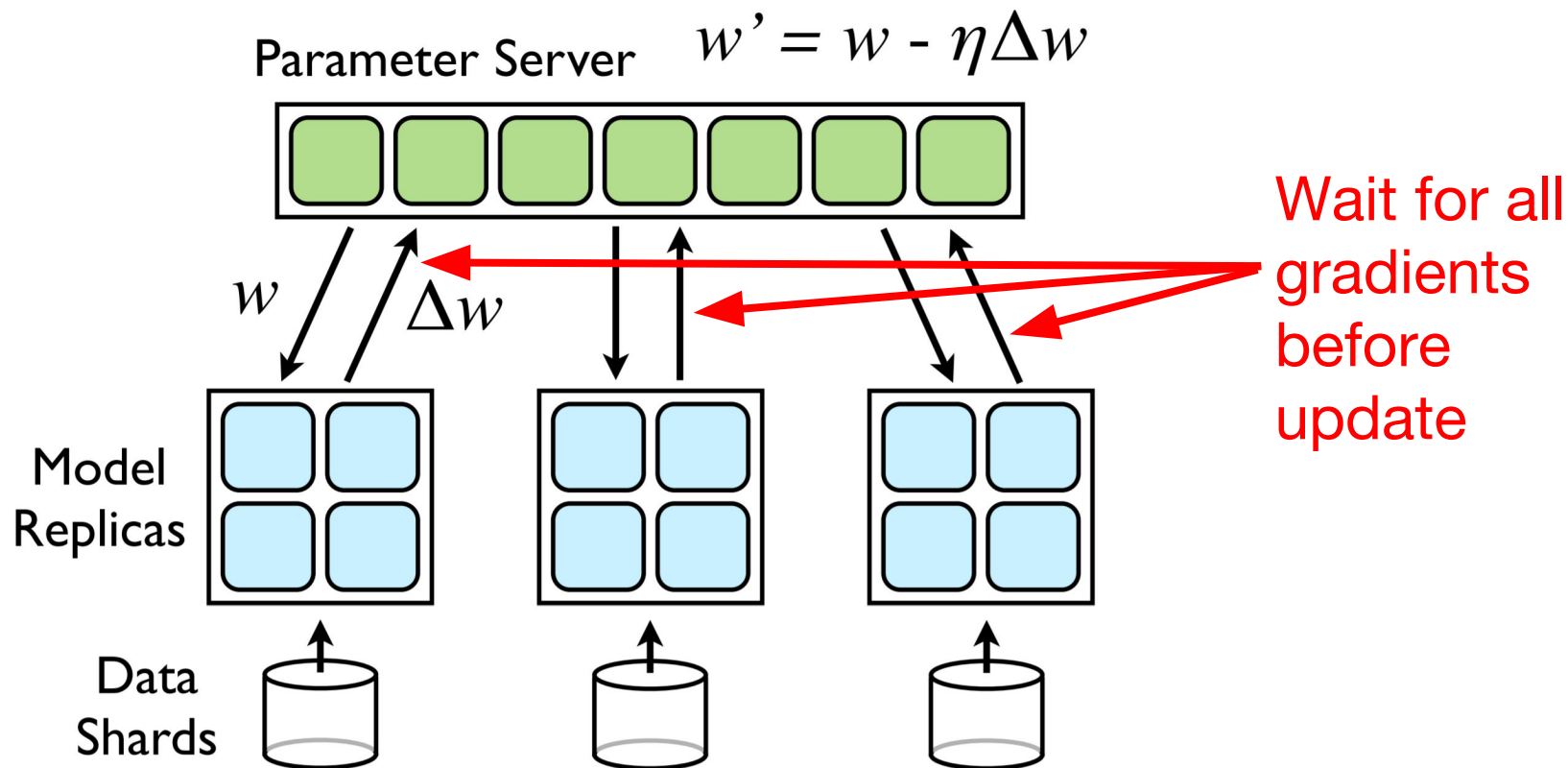
Stale Gradients

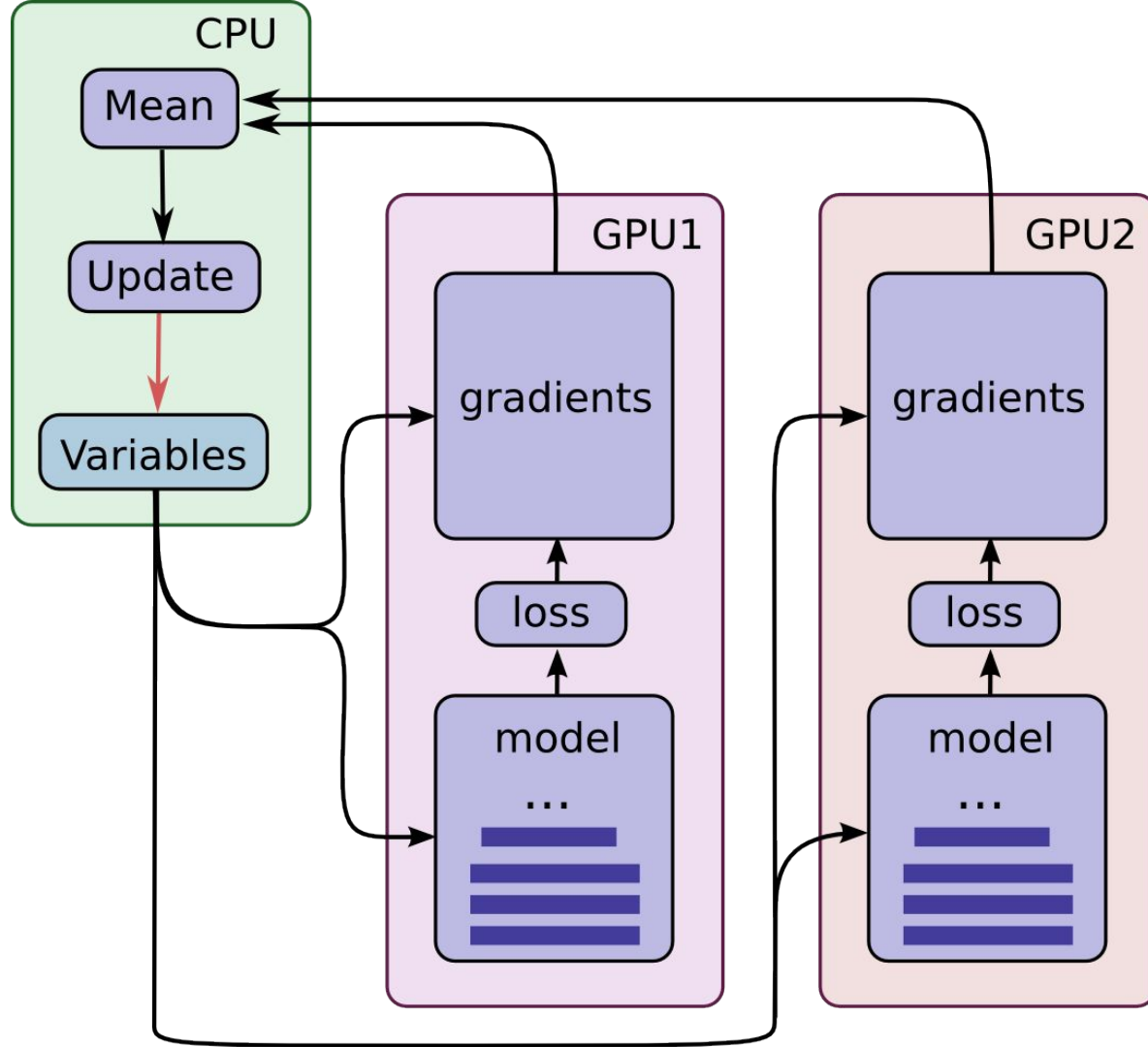


Stale Gradients

- 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

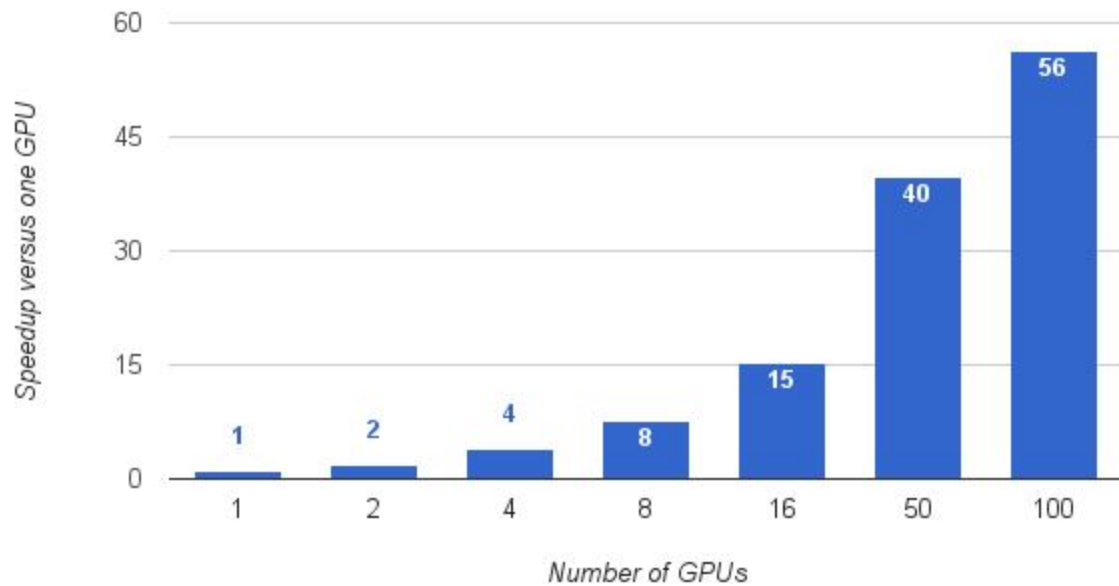




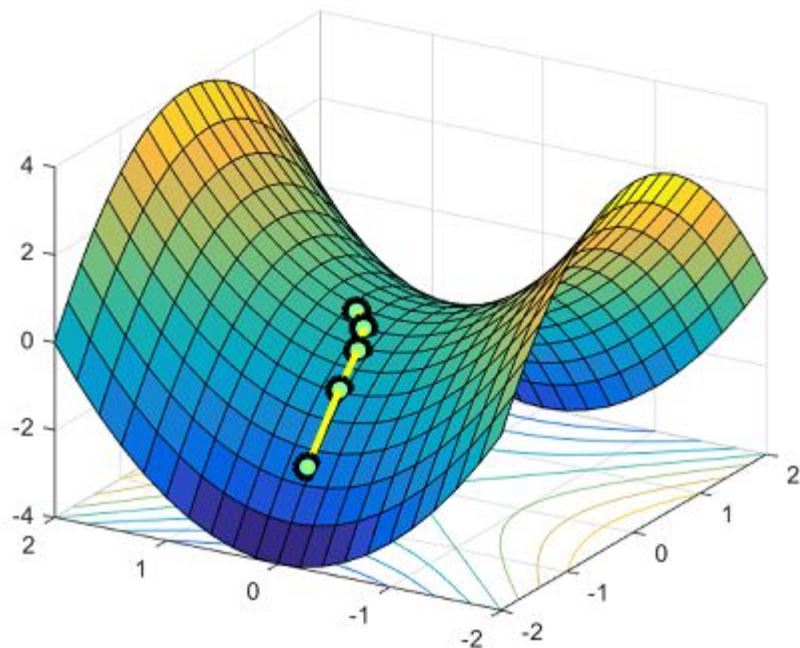
Sync SGD

- 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

Training Inception with Distributed TensorFlow



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

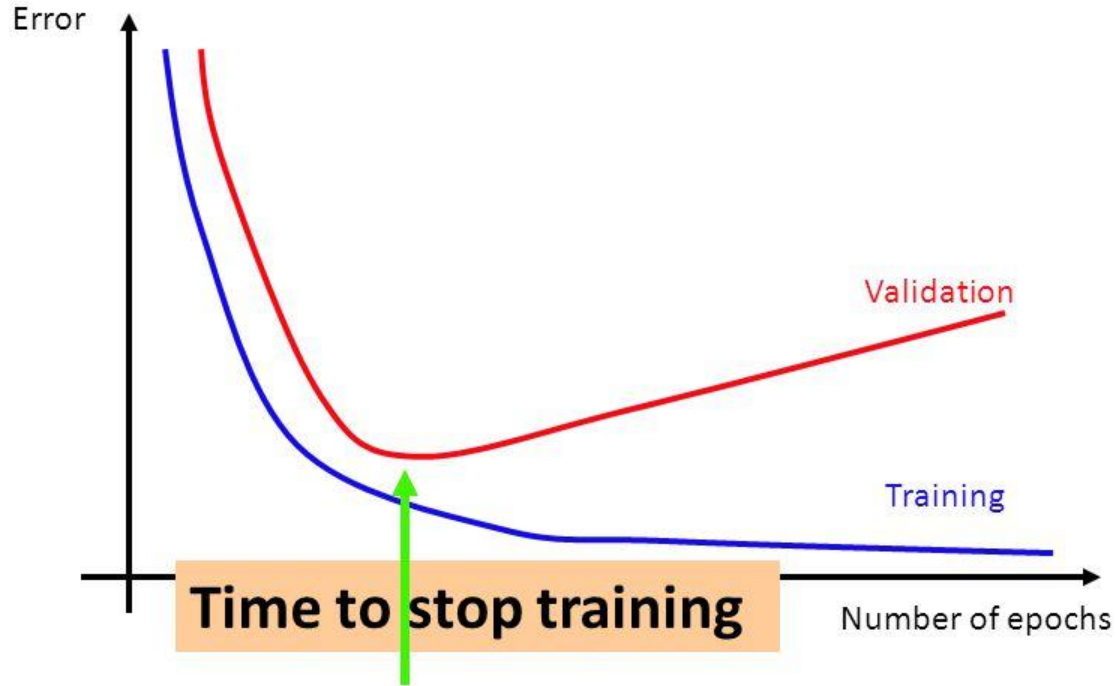
Regularization Outline

- 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



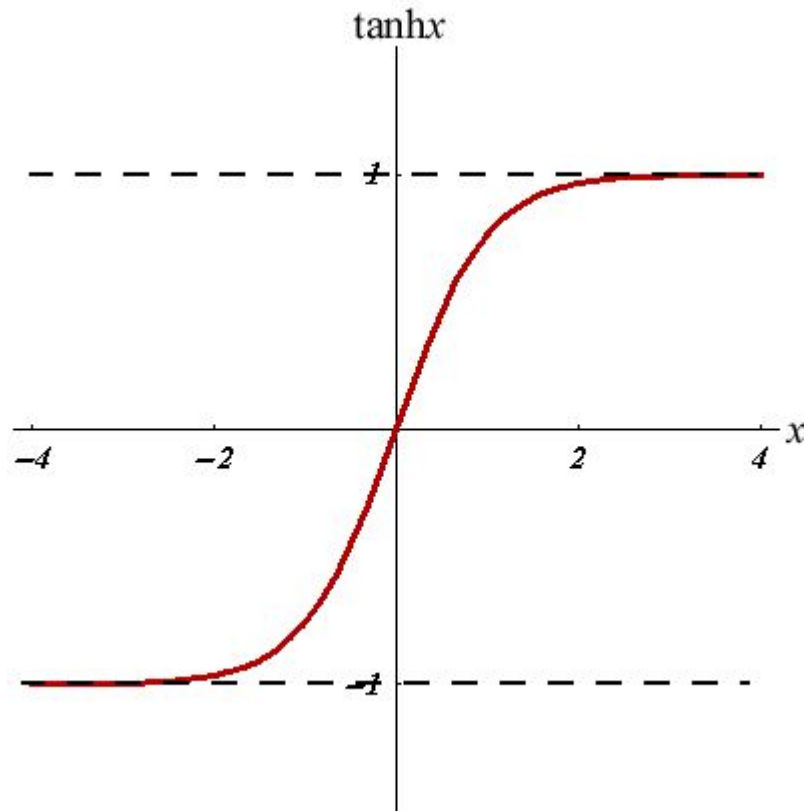
L1 / L2 regularization

$$||\theta||_1 = \sum_i |\theta_i|$$

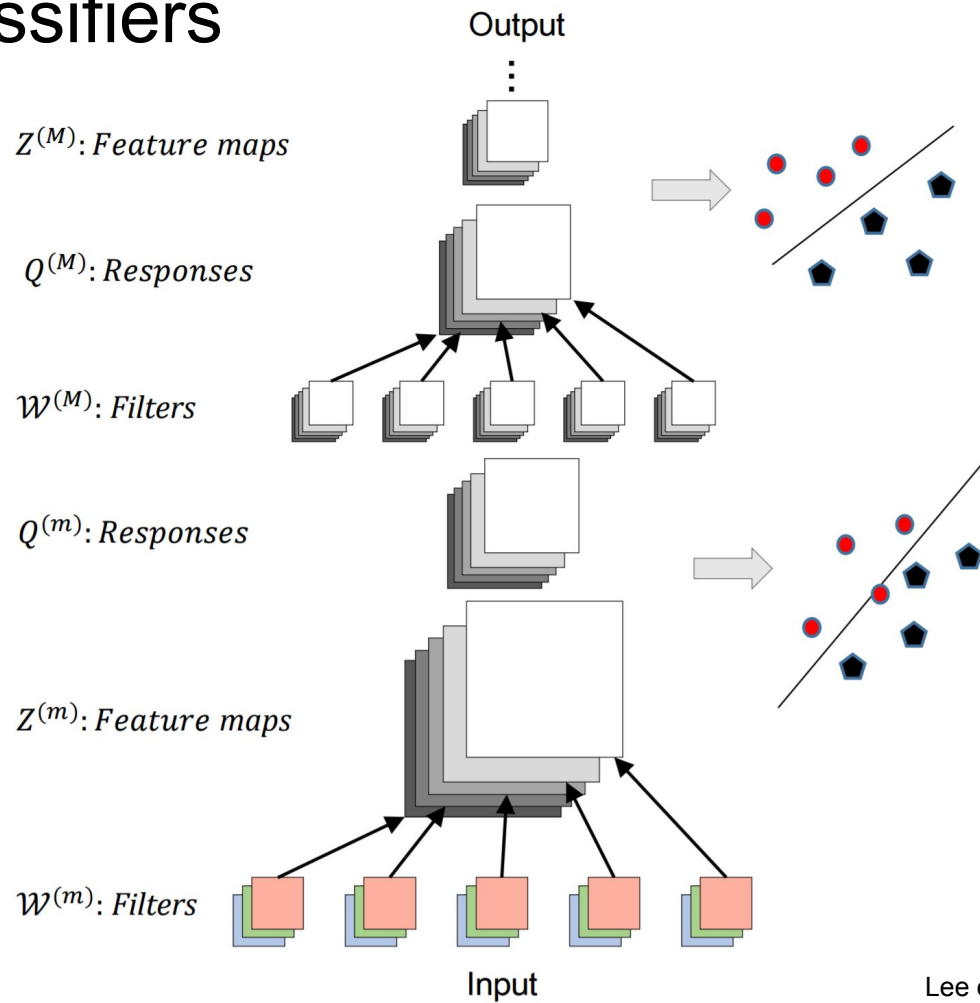
$$||\theta||_2 = \sqrt{\sum_i \theta_i^2}$$

L1 / L2 regularization

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



Auxiliary Classifiers



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

True Label $\boxed{y_i} = [0, 0, 1, 0]$

Baseline $\boxed{u} = [0.25, 0.25, 0.25, 0.25]$

$$\hat{y}_i = (1 - \epsilon)\boxed{y_i} + \epsilon\boxed{u}$$

Mixed Target

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

When is uniform
a good choice?
Bad?

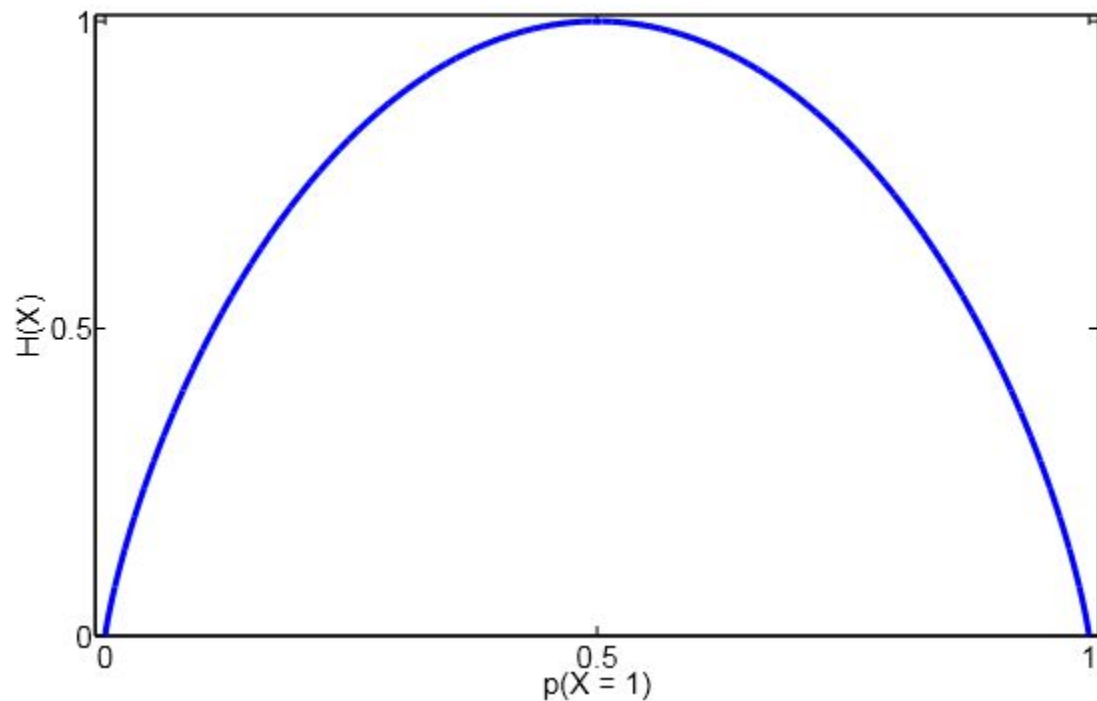
$$y_i = [0, 0, 1, 0]$$

$$u = [0.25, 0.25, 0.25, 0.25]$$

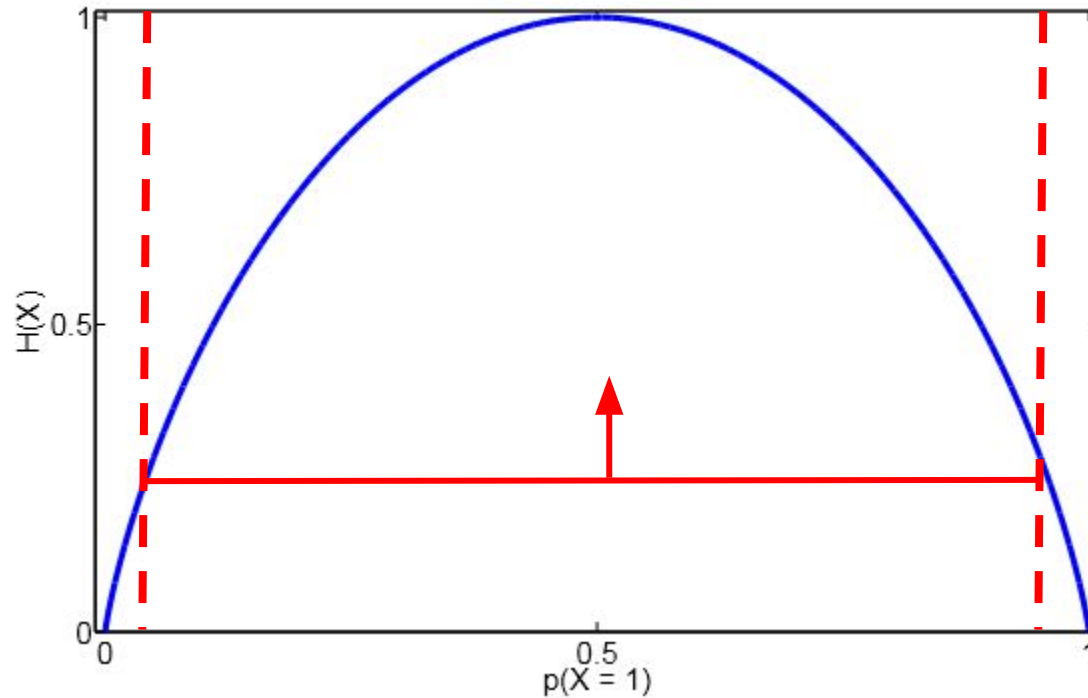
$$\hat{y}_i = (1 - \epsilon)y_i + \epsilon u$$

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

$$H(p_{\theta}(\mathbf{y}|x)) = - \sum p_{\theta}(\mathbf{y}|x) \log p_{\theta}(\mathbf{y}|x)$$



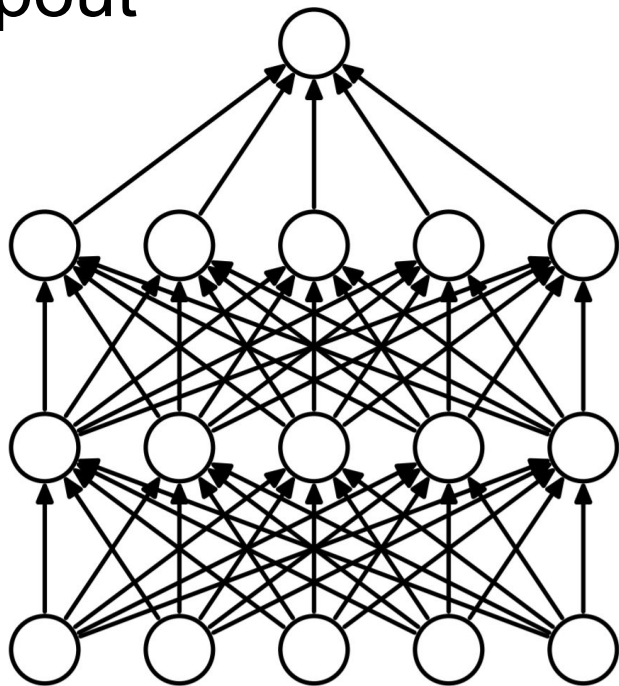
Enforce entropy to be
above some threshold



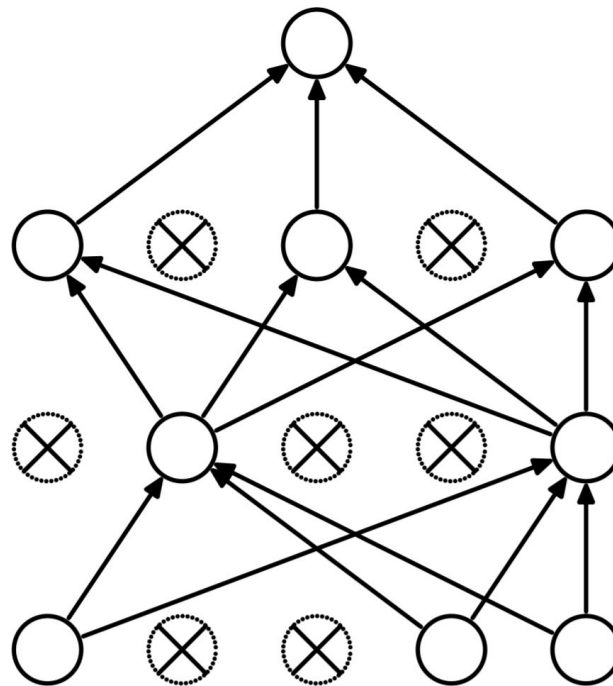
Regularization Outline

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

Dropout



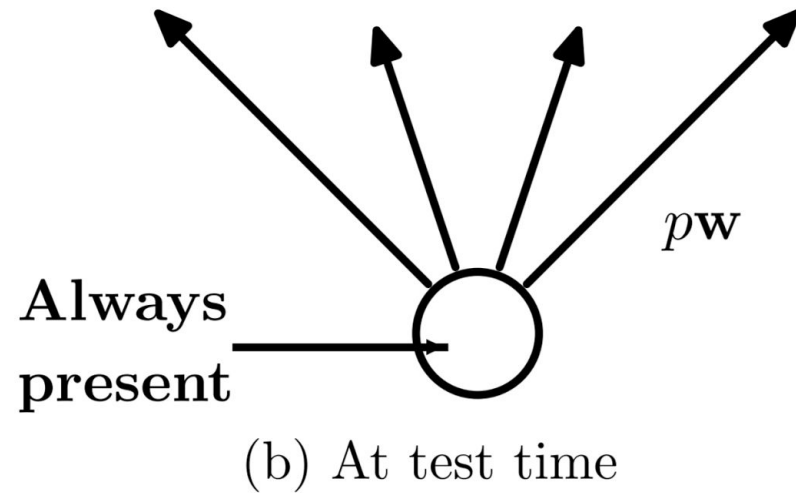
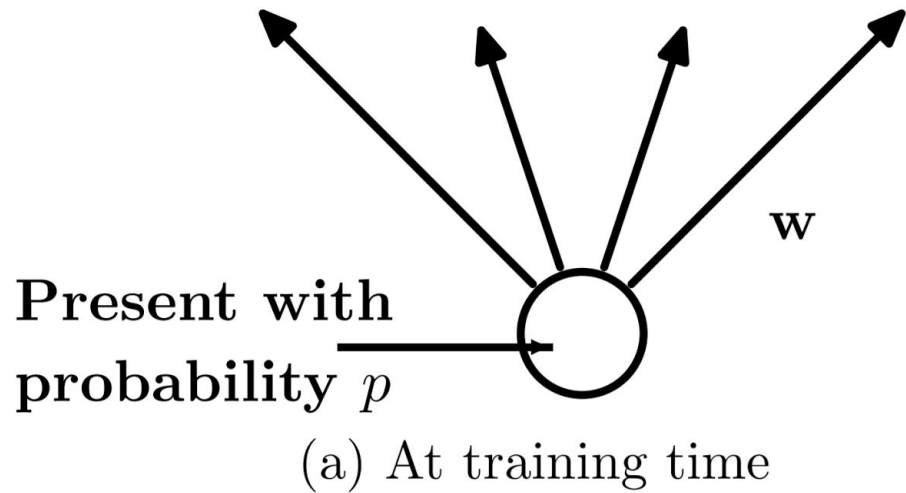
(a) Standard Neural Net



(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



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

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"

Gal 2015. "A theoretically grounded application of dropout in recurrent neural networks"

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?

Fighting internal covariate shift

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

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?

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

Mean

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i$$

Batch size

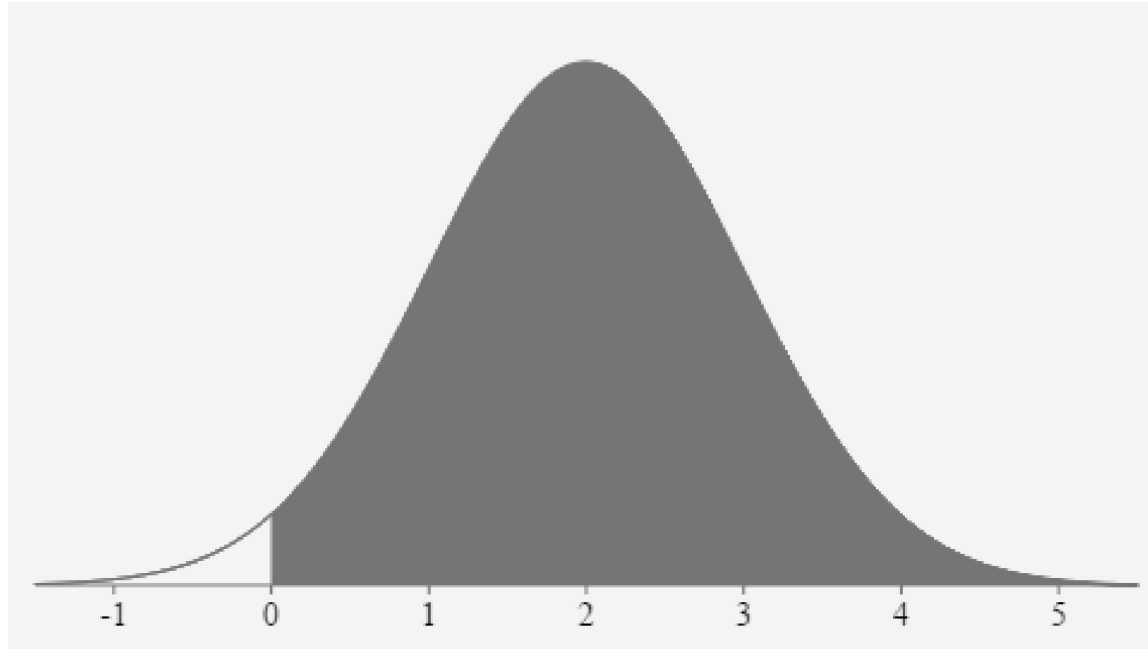
Variance

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2$$

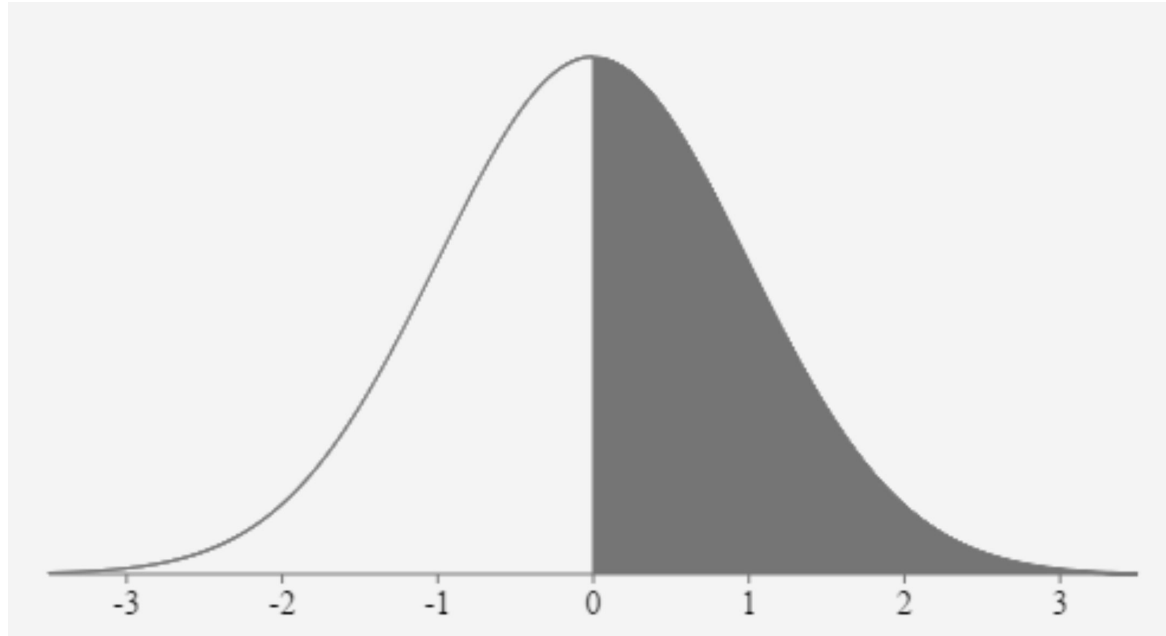
Normalize

$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$$

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

$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$$

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

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

Advantages

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

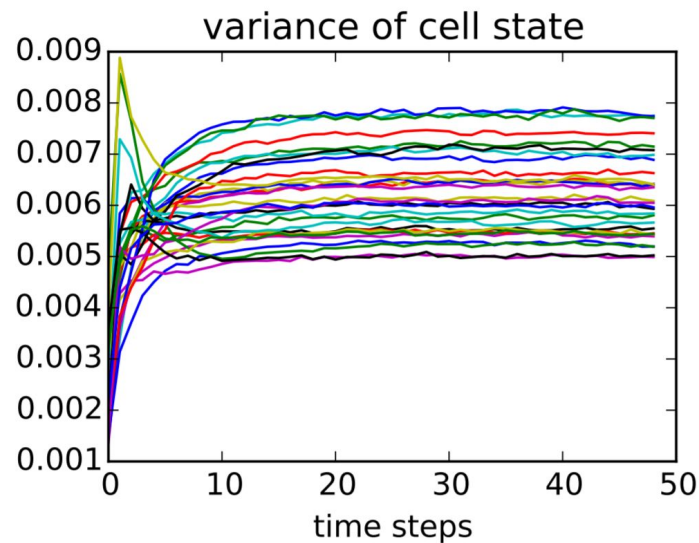
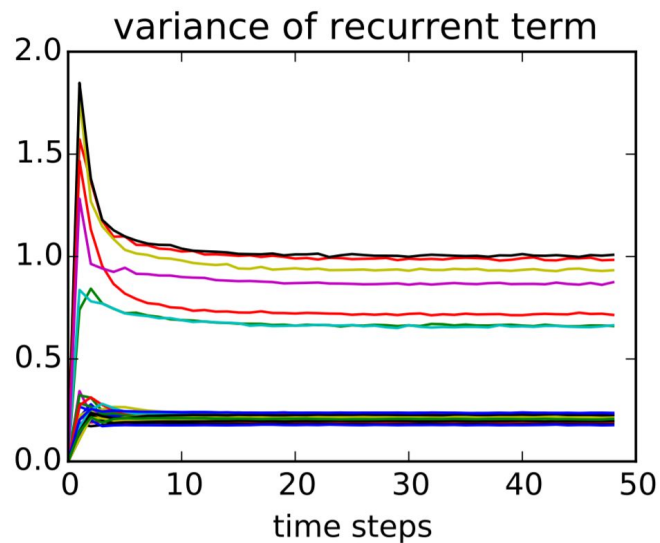
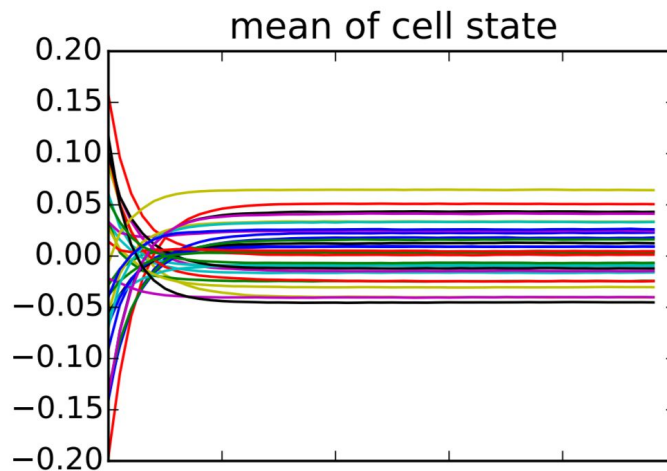
Batch Norm for RNNs?

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

$$\begin{pmatrix} \tilde{\mathbf{f}}_t \\ \tilde{\mathbf{i}}_t \\ \tilde{\mathbf{o}}_t \\ \tilde{\mathbf{g}}_t \end{pmatrix} = \boxed{\text{BN}}(\mathbf{W}_h \mathbf{h}_{t-1}; \gamma_h, \beta_h) + \boxed{\text{BN}}(\mathbf{W}_x \mathbf{x}_t; \gamma_x, \beta_x) + \mathbf{b}$$

$$\mathbf{c}_t = \sigma(\tilde{\mathbf{f}}_t) \odot \mathbf{c}_{t-1} + \sigma(\tilde{\mathbf{i}}_t) \odot \tanh(\tilde{\mathbf{g}}_t)$$

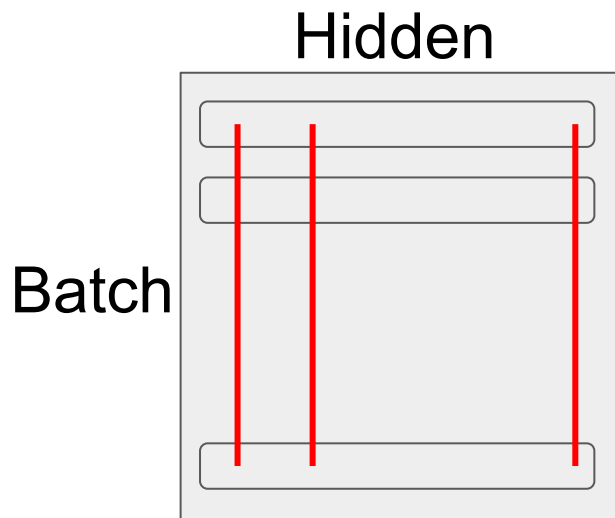
$$\mathbf{h}_t = \sigma(\tilde{\mathbf{o}}_t) \odot \tanh(\boxed{\text{BN}}(\mathbf{c}_t; \gamma_c, \beta_c))$$



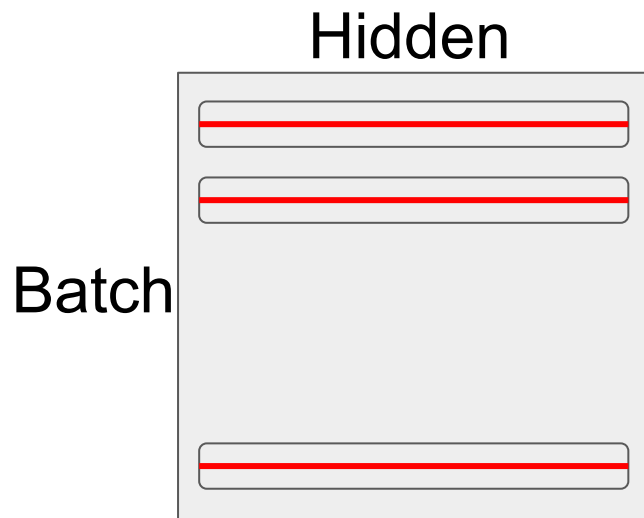
Recurrent Batch Normalization

- Maintain independent statistics for the first T steps
- $t > T$ uses the statistics from time T
- Have to initialize γ to ~ 0.1

Batch Normalization



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 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 $\text{Var}[\mathbf{y}_0]$ and of last layer activations $\text{Var}[\mathbf{y}_L]$
- If $\text{Var}[\mathbf{y}_L] \gg \text{Var}[\mathbf{y}_0]$?
- If $\text{Var}[\mathbf{y}_L] \ll \text{Var}[\mathbf{y}_0]$?

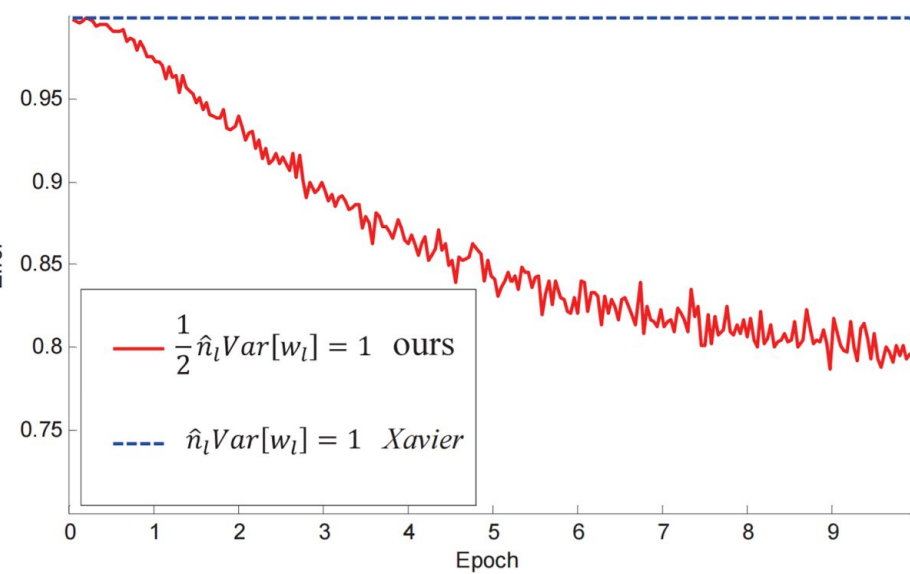
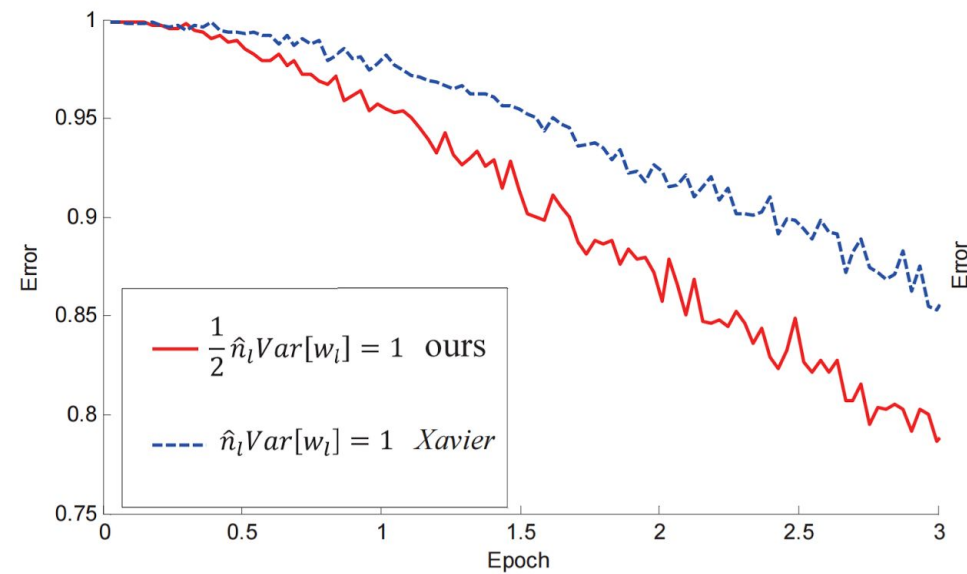
He initialization for ReLU networks

- Call variance of input $\mathbf{Var}[\mathbf{y}_0]$ and of last layer activations $\mathbf{Var}[\mathbf{y}_L]$
- If $\mathbf{Var}[\mathbf{y}_L] \gg \mathbf{Var}[\mathbf{y}_0]$, exploding activations \rightarrow diverge
- If $\mathbf{Var}[\mathbf{y}_L] \ll \mathbf{Var}[\mathbf{y}_0]$, diminishing activations \rightarrow vanishing gradient
- Key idea: $\mathbf{Var}[\mathbf{y}_L] = \mathbf{Var}[\mathbf{y}_0]$

He initialization

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


Number of inputs to neuron



Identity RNN

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

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

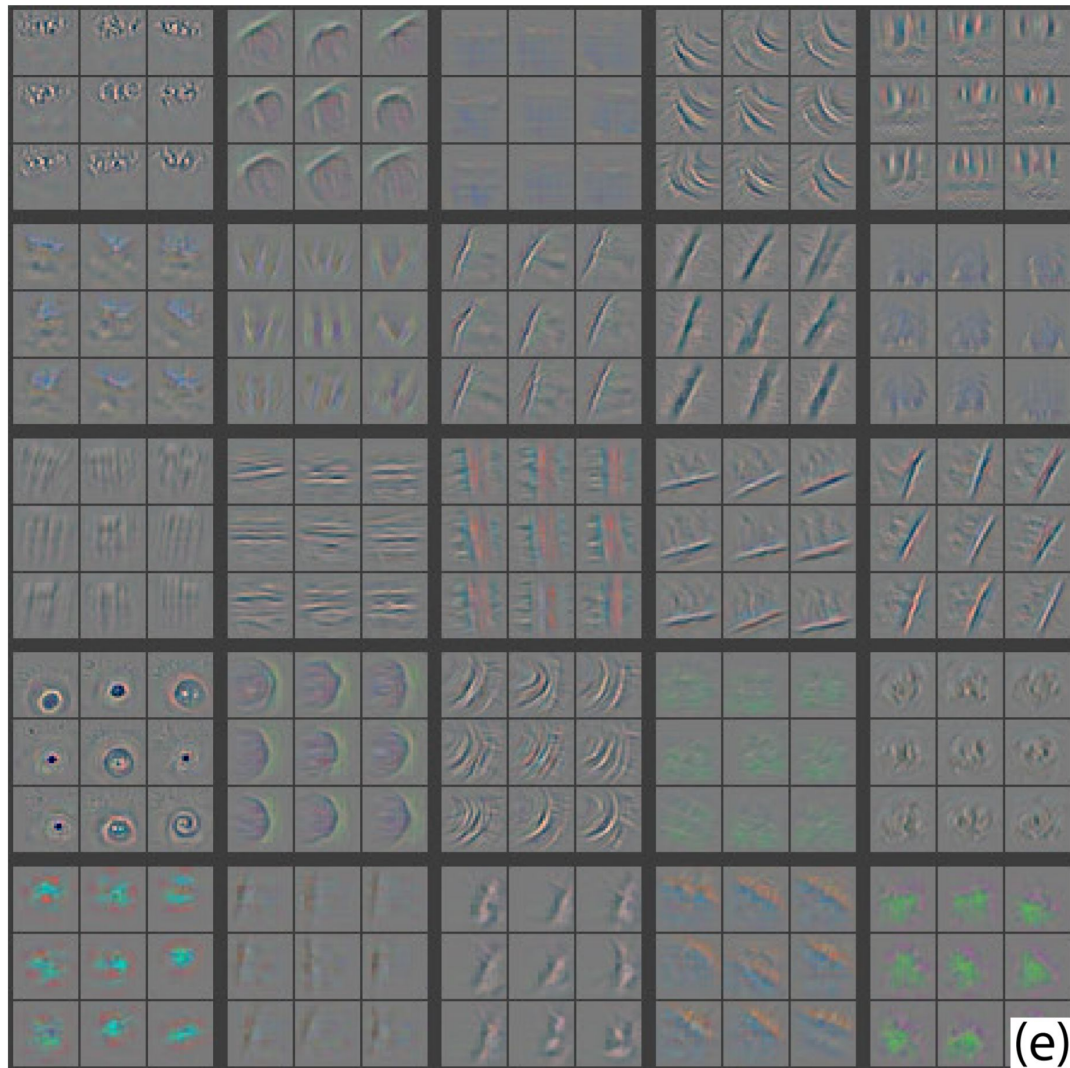
Le et al. 2015. “A simple way to initialize recurrent neural networks of rectified linear units”

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



(e)

Zeiler and Fergus, 2013. "Visualizing and Understanding Convolutional Networks"

Pretraining for CNNs in vision

	New dataset is small	New dataset is large
Pretrained dataset is similar to new dataset	Freeze weights and train linear classifier from top level features	Fine-tune all layers (pretrain for faster convergence and better generalization)
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)

Method	Part info	mean Accuracy
Sift+Color+SVM[45]	✗	17.3
Pose pooling kernel[49]	✓	28.2
RF[47]	✓	19.2
DPD[50]	✓	51.0
Poof[5]	✓	56.8
CNN-SVM	✗	53.3
CNNaug-SVM	✗	61.8
DPD+CNN(DeCaf)+LogReg[10]	✓	65.0

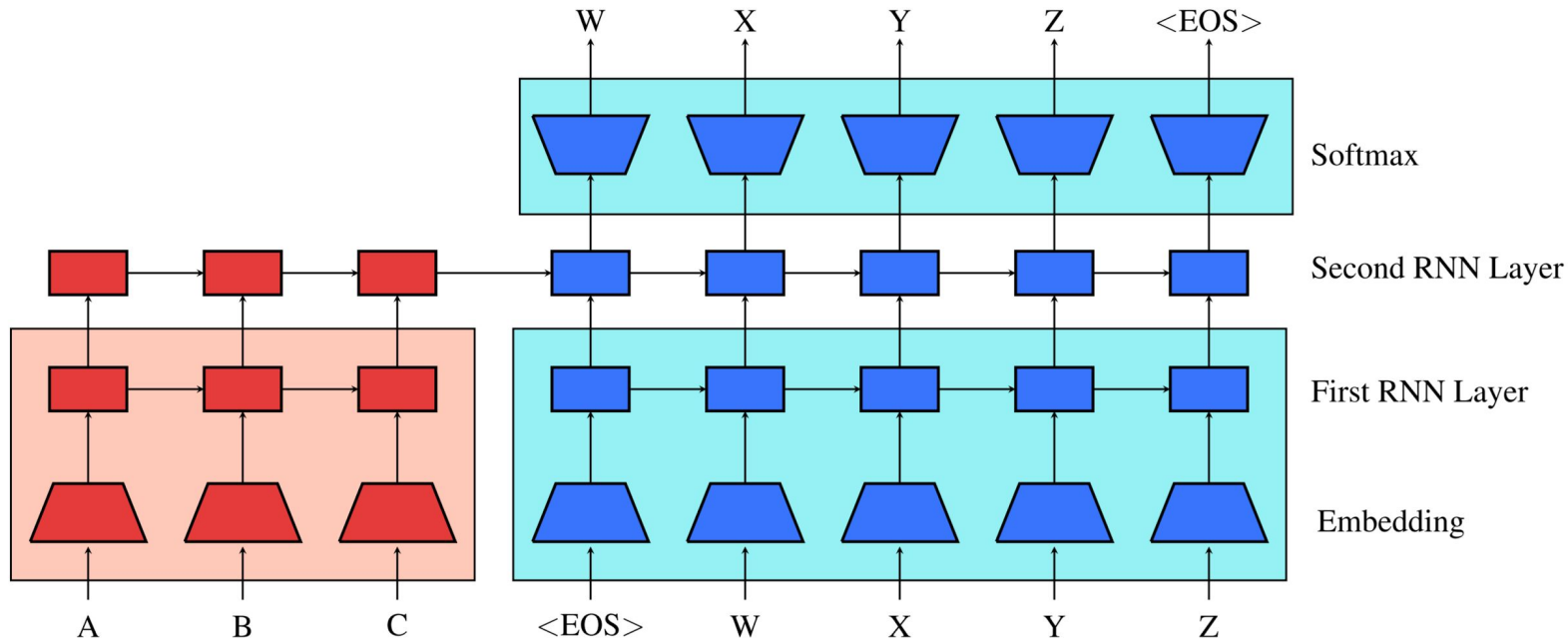
Method	mean Accuracy
HSV [27]	43.0
SIFT internal [27]	55.1
SIFT boundary [27]	32.0
HOG [27]	49.6
HSV+SIFTi+SIFTb+HOG(MKL) [27]	72.8
BOW(4000) [14]	65.5
SPM(4000) [14]	67.4
FLH(100) [14]	72.7
BiCos seg [7]	79.4
Dense HOG+Coding+Pooling[2] w/o seg	76.7
Seg+Dense HOG+Coding+Pooling[2]	80.7
CNN-SVM w/o seg	74.7
CNNaug-SVM w/o seg	86.8

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

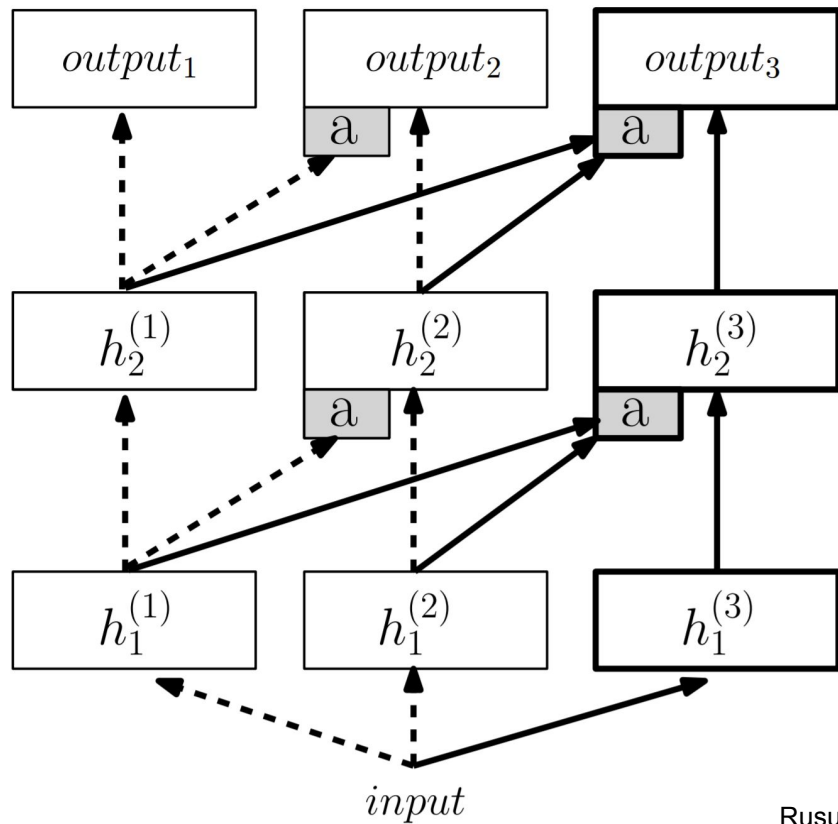
Razavian et al. 2014. “CNN features off-the-shelf: an astounding baseline for recognition”

	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv	mAP
GHM[8]	76.7	74.7	53.8	72.1	40.4	71.7	83.6	66.5	52.5	57.5	62.8	51.1	81.4	71.5	86.5	36.4	55.3	60.6	80.6	57.8	64.7
AGS[11]	82.2	83.0	58.4	76.1	56.4	77.5	88.8	69.1	62.2	61.8	64.2	51.3	85.4	80.2	91.1	48.1	61.7	67.7	86.3	70.9	71.1
NUS[39]	82.5	79.6	64.8	73.4	54.2	75.0	77.5	79.2	46.2	62.7	41.4	74.6	85.0	76.8	91.1	53.9	61.0	67.5	83.6	70.6	70.5
CNN-SVM	88.5	81.0	83.5	82.0	42.0	72.5	85.3	81.6	59.9	58.5	66.5	77.8	81.8	78.8	90.2	54.8	71.1	62.6	87.2	71.8	73.9
CNNaug-SVM	90.1	84.4	86.5	84.1	48.4	73.4	86.7	85.4	61.3	67.6	69.6	84.0	85.4	80.0	92.0	56.9	76.7	67.3	89.1	74.9	77.2

Pretraining for Seq2Seq



Progressive 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

$$\mathbf{y}_l = W_l \mathbf{x}_l$$

W is a d -by- n matrix

$$\mathbf{y}_l = W_l \mathbf{x}_l$$

W is a d -by- n matrix

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

$$\mathbf{y}_l = W_l \mathbf{x}_l$$

W is a d -by- n matrix

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

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

$$\mathbf{y}_l = W_l \mathbf{x}_l$$

W is a d -by- n matrix

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

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

$$Var[y_l] = n_l Var[w_l x_l]$$

$$Var[y_l] = n_l Var[w_l x_l]$$

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

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

$$Var[ab] = E[a^2]E[b^2] + E[a]^2 E[b]^2$$

$$Var[y_l] = n_l (E[w_l^2]E[x_l^2] + E[w_l]^2 E[x_l]^2)$$

$$Var[y_l] = n_l Var[w_l x_l]$$

Let w_l have zero mean

$$Var[ab] = E[a^2]E[b^2] + E[a]^2 E[b]^2$$

$$Var[y_l] = n_l (E[w_l^2]E[x_l^2] + E[w_l]^2 E[x_l]^2)$$

$$Var[y_l] = n_l E[w_l^2]E[x_l^2]$$

$$Var[y_l] = n_l E[w_l^2] E[x_l^2]$$

$$Var[a] = E[(a - E[a])^2]$$

$$Var[y_l] = n_l E[w_l^2] E[x_l^2]$$

$$Var[a] = E[(a - E[a])^2]$$

$$Var[y_l] = n_l Var[w_l] E[x_l^2]$$

$$Var[y_l] = n_l Var[w_l] E[x_l^2]$$

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

Let w_{l-1} have a zero-mean symmetric distribution

$$Var[y_l] = n_l Var[w_l] E[x_l^2]$$

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

Let w_{l-1} have a zero-mean symmetric distribution

Then y_{l-1} also has a zero-mean symmetric distribution

$$Var[y_l] = n_l Var[w_l] E[x_l^2]$$

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

Let w_{l-1} have a zero-mean symmetric distribution

Then y_{l-1} also has a zero-mean symmetric distribution

$$Var[y_{l-1}] = \int_{-\infty}^0 y_{l-1}^2 p(y_{l-1}) dy_{l-1} + \int_0^{\infty} y_{l-1}^2 p(y_{l-1}) dy_{l-1}$$

$$Var[y_l] = n_l Var[w_l] E[x_l^2]$$

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

Let w_{l-1} have a zero-mean symmetric distribution

Then y_{l-1} also has a zero-mean symmetric distribution

$$Var[y_{l-1}] = \int_{-\infty}^0 y_{l-1}^2 p(y_{l-1}) dy_{l-1} + \int_0^{\infty} y_{l-1}^2 p(y_{l-1}) dy_{l-1}$$

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

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

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

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

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

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

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

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

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

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

$$Var[y_l] = n_l Var[w_l] E[x_l^2]$$

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

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

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

$$Var[y_l] = n_l Var[w_l] E[x_l^2]$$

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

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

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

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

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

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

$$Var[y_L] = Var[y_1]$$

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

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

$$Var[y_L] = Var[y_1]$$

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

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

$$\frac{1}{2}n_lVar[w_l] = 1$$

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