Outline

- SGD-typed algorithms for Deep Learning
- Parallel SGD for deep learning
Perceptron

- Prediction value for a training data:

\[
prediction = f(w^T x_i + w_0)
\]

- \(x_i\): data
- \(w\): parameters to be learned
- \(f(z) = \max(0, z)\): activation function

(Figure from Shlens and Toderici, ICIP 2016 slides)
Neural Network: A single hidden layer

- Prediction value for a training data:

\[
prediction = h_2(W_2 h_1(W_1 x_i)) := f(x_i; W)
\]

- \( W = \{W_i\} \): parameters to be learned
- \( h_2, h_1 \): activation function
Neural Network: Deep Network

- Minimize the following loss function:

  \[ \min_W \frac{1}{n} \sum_{i=1}^{n} \ell(f(x_i; W), y_i), \]

  \[ f(x_i; W) := W_k(h_{k-1}(W_{k-1} \ldots W_2 h_1(W_1 x_i))): k\text{-layers neural network} \]
  \[ \ell(\cdot): \text{loss function} \]

- \( W \) can be sparse and customized: corresponding to the network structure

- Famous example:
  - Convolutional network (extracting image features)
  - Recurrent network & LSTM (long-term short-term memory): learning from sequence data
Convolutional Network

- Convolution operator: an effective way to extract image features

(Figure from Shlens and Toderici, ICIP 2016 slides)

- Before deep learning: human design of the convolutional kernel
- Deep learning: learning features and classifiers together
Convolutional Network

- A convolution layer:

- Convolutional network (e.g., AlexNet)
Stochastic Gradient Descent

- Neural network objective function:

\[
\min_W \frac{1}{n} \sum_{i=1}^{n} \ell(f(x_i; W), y_i),
\]

\[
f(x_i; W) := W_k(h_{k-1}(W_{k-1} \ldots W_2 h_1(W_1 x_i)))
\]

- Stochastic gradient descent: for each iteration
  1. Randomly sample a small subset \( S \)
  2. Compute \( g = \frac{1}{|S|} \sum_{i \in S} \nabla_W \ell(f(x_i; W), y_i) \)
  3. Update \( W \leftarrow W + \alpha_t g \)

- Gradient computed by chain rule (back-propagation).
AdaGrad (2011): adaptive step size for each parameter

Update rule at the \(t\)-th iteration:

- Compute \( \mathbf{g} = \nabla \ell (\mathbf{x}_i) \)
- Estimate the second moment: \( G_{ii} = G_{ii} + g_i^2 \) for all \(i\)
- Parameter update: \( x_i \leftarrow x_i - \frac{\eta}{\sqrt{G_{ii}}} g_i \) for all \(i\)

Adam (Kingma and Ba, 2015): maintain both first moment and second moment

Update rule at the \(t\)-th iteration:

- Compute \( \mathbf{g} = \nabla \ell_i (\mathbf{x}) \)
- \( \mathbf{m} = \beta_1 \mathbf{m} + (1 - \beta_1) \mathbf{g} \)
- \( \hat{\mathbf{m}} = \mathbf{m}_t / (1 - \beta_1^t) \) (estimate of first moment)
- \( \mathbf{v} = \beta_2 \mathbf{v} + (1 - \beta_2) \mathbf{g}^2 \)
- \( \hat{\mathbf{v}} = \mathbf{v}_t / (1 - \beta_2^t) \) (estimate of second moment)
- \( \mathbf{x} \leftarrow \mathbf{x} - \eta \hat{\mathbf{m}} / (\hat{\mathbf{v}} + \epsilon) \)

(All the operations are element-wise)
Other Optimization Algorithms?

- Coordinate Descent:
  
  Can we update a subset of links at a time? (Yes)
  But the gradient has to be computed by chain rule (back propagation)
  
  ⇒ Time for updating one coordinate ≈ time for updating all coordinates (?)

- Second order method with Using exact Hessian?
  
  Hessian-vector product can be computed by chain rule efficiently!

  (Pearlmutter et al., “Fast Exact Multiplication by the Hessian”. 1993. )


  Better performance in auto-encoder

  (Kiros, “Training Neural Networks with Stochastic Hessian-Free Optimization”. 2013): better performance in conv-network
Other Optimization Algorithms?

- Second order method with approximate Hessian (Quasi-Newton)
  LBFGS (quasi-Newton): using gradients to approximate Hessian
  Shows better performance in auto-encoder
- Other second order methods?
  Subsampled gradient + subsampled Hessian?
Parallel SGD for Deep Learning
Synchronized Parallel: Minibatch SGD

- GPU parallelism:
  - Convolution operator: matrix-matrix product
  - Easy to parallel in GPU

- Each machine/core/GPU:
  
  Compute \( g_k = \frac{1}{|S_k|} \sum_{i \in S_k} \nabla W \ell(f(x_i; W), y_i) \)

- Averaging the results (synchronization):
  
  \[ g = \frac{1}{K} \sum_{k=1}^{K} g_k \]

- Update the parameters:
  
  \[ W \leftarrow W + \alpha_t g \]

- Slow in distributed system due to the synchronization step.
Asynchronous Parallel: Downpour-SGD

• (Dean et al., “Large Scale Distributed Deep Networks”, in NIPS 2012)
• Parameter server: a centralized server to maintain/update latest parameters
• Update rule for each machine:
  For $t = 1, 2, \ldots$
  1. Read latest $W$ from parameter server
  2. Compute local SGD update:
     \[
     \Delta W = \alpha_t \frac{1}{|S|} \sum_{i \in S} \nabla W \ell(f(x_i; W), y_i)
     \]
  3. Send $\Delta W$ to parameter server
• Parameter server (usually multiple machines):
  • Maintain the unique global parameter $W$
  • Get $\Delta W$ from local workers and update $W \leftarrow W + \Delta W$
Downpour-SGD

\[ w' = w - \eta \Delta w \]
**Consensus ADMM**

- Summarized in (Boyd et al., 2011)
- Reformulate the ERM problem ($K$: number of partitions/machines)

\[
\min_{w_1,\ldots,w_K} \sum_{k=1}^{K} f_k(w_k) \quad \text{such that} \quad w_k = z, \; \forall k
\]

- Augmented Lagrangian:

\[
L(\{w_k, \lambda_k\}, z) = \sum_{k=1}^{K} (f_k(w_k) + \langle \lambda_k, w_k - z \rangle + \frac{\beta}{2} \| w_i - z \|^2)
\]

- Main idea: iteratively update $w_k$, $z$, and $\lambda_k$. 
For each iteration $t$:

1. Update local models (in parallel):
   \[ w_{k}^{t+1} = \arg \min_w f_k(w) + \langle \lambda^t_k, w \rangle + \frac{\beta}{2} \| w - z^t \|^2 \]

2. Update global model (synchronization):
   \[ w = \arg \min_w \sum_{k=1}^{K} -\langle \lambda^t_k, z \rangle + \frac{\beta}{2} \| w_{k}^{t+1} - z \|^2 \]

3. Update dual variables (in parallel):
   \[ \lambda^{t+1}_k = \lambda^t_k + \beta(w_{k}^{t+1} - z^{t+1}) \]

The global model update has a closed form solution:
\[ w = \frac{1}{K}(\sum_{k=1}^{K} w_{k}^{t+1} + \sum_{k=1}^{K} \lambda^t_k) \]

Usually slower convergence and need a synchronization step.
Elapsed SGD

- A reformulation (simpler than ADMM):

\[
\min_{w_1, \ldots, w_K, z} \sum_{k=1}^{K} f_k(w_k) + \frac{\rho}{2} \| w_k - z \|^2
\]

- Gradient descent:

\[
\begin{align*}
w_{k}^{t+1} &= w_{k}^{t} - \eta (\nabla f_k(w_k^t) + \rho (w_k^T - z^t)) \\
z^{t+1} &= z^t - \eta \sum_{k=1}^{K} \rho (z^t - w_k^t) \\
&= (1 - \eta \rho K)z^t + \eta \rho K \left( \frac{1}{K} \sum_{k=1}^{K} w_k^t \right)
\end{align*}
\]
Asynchronous version

Each worker repeat the following operation ($\alpha = \eta \rho$):

- $w \leftarrow w_k$
- If $t_k \% \tau = 0$:
  - $w_k \leftarrow w_k - \alpha (w - z)$
  - $z \leftarrow z + \alpha (w - z)$
  - $w_k \leftarrow w_k - \eta g_k$
- $t_k \leftarrow t_k + 1$

Convergence & Convergence rate (?)
Can add the momentum term.
Elapsed SGD: CIFAR

10 classes, totally 60,000 32x32 colour images.

(Figures from Zhang et al., “Deep Learning with Elastic Averaging SGD”)
1000 classes, > 1 million 221x221 colour images.

(Figures from Zhang et al., “Deep Learning with Elastic Averaging SGD”)

Elapsed SGD: ImageNet
ADMM for decoupling layers

- Proposed by (Taylor et al., “Training Neural Networks Without Gradients: A Scalable ADMM Approach”. ICML 2016)
- Another way to rewrite the neural network with $L$ layers:

$$\min_{\{W_i\}, \{A_i\}, \{Z_i\}} \ell(Z_L, Y)$$

s.t. $Z_i = W_iA_{i-1}$ for $i = 1, \ldots, L$

$A_i = h_i(Z_i)$ for $i = 1, \ldots, L - 1$

where $h_i(\cdot)$ is activation functions, and rows of $A_i, Z_i$ correspond to training samples

- ADMM-like formulation:

$$\min_{\{W_i\}, \{A_i\}, \{Z_i\}} \ell(Z_L, Y) + \sum_{i=1}^{L} \beta \|Z_i - W_iA_{i-1}\|_F^2 + \sum_{i=1}^{L-1} \beta \|A_i - h_i(Z_i)\|_F^2$$

$$+ \langle Z_L, \lambda \rangle$$
ADMM for decoupling layers

- Update of $Z_i$: element-wise problem, each one is a single variable optimization
- Update of $A_i$: least square problem decomposed into columns (each training samples)
- Update of $W_i$: least square problem:
  \[ W_i \leftarrow Z_i A_{i-1}^T (A_{i-1} A_{i-1}^T)^{-1} \]

Compute $(A_{i-1} A_{i-1}^T)$ distributedly
Proposal Discussions on Thursday (in my office)