

Lecture 4:

CNN: Optimization Algorithms

boris.ginzburg@intel.com

Agenda

- Gradient-based learning for Convolutional NN
- Caffe: SGD parameters
 - Learning rate adaptation
 - Momentum and weight decay
- SGD with line search
- Newton methods;
 - Conjugate Gradient
 - Limited memory BFGS (L-BFGS)
- Imagenet training

Links

1. LeCun et al "Efficient Backpropagation "
<http://cseweb.ucsd.edu/classes/wi08/cse253/Handouts/lecun-98b.pdf>
 2. Le, Ng et al "On Optimization Methods for Deep Learning"
<http://cs.stanford.edu/people/ang/?portfolio=on-optimization-methods-for-deep-learning>
 3. Hinton Lecture
<https://www.cs.toronto.edu/~hinton/csc2515/notes/lec6tutorial.pdf>
- ++
1. <http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf>
 2. http://www.iro.umontreal.ca/~pift6266/H10/notes/gradient.html#flow_graph
 3. Bottou Stochastic Gradient Descent Tricks
<http://research.microsoft.com/pubs/192769/tricks-2012.pdf>

Stochastic Gradient Descent

We want to minimize an error for convolutional NN with weights W , over N samples (x_n, y_n) :

$$E(w) = \frac{1}{N} \sum_{n=1}^N l(f(x_n, w), y_n)$$

Option 1: batch training

- We compute gradient using back-propagation:

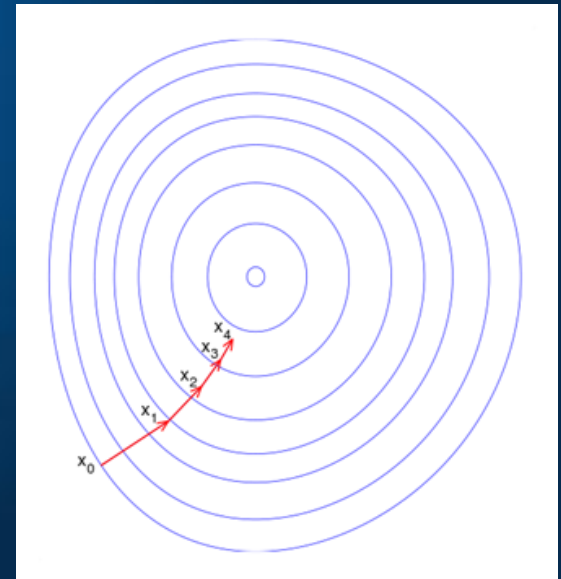
$$\frac{\partial E}{\partial y_{l-1}} = \frac{\partial E}{\partial y_l} \times \frac{\partial y_l(w, y_{l-1})}{\partial y_{l-1}}; \frac{\partial E}{\partial w_l} = \frac{\partial E}{\partial y_l} \times \frac{\partial y_l(w, y_{l-1})}{\partial w_l}$$

- average gradient over data set:

$$W(t+1) = W(t) - \lambda * \frac{1}{N} \sum_{n=1}^N \frac{\partial l}{\partial w}((x_n, w), y_n)$$

Option 2: stochastic gradient descent:

1. Randomly choose sample (x_k, y_k) :
2. $W(t+1) = W(t) - \lambda * \frac{\partial l}{\partial w}((x_k, w), y_k)$

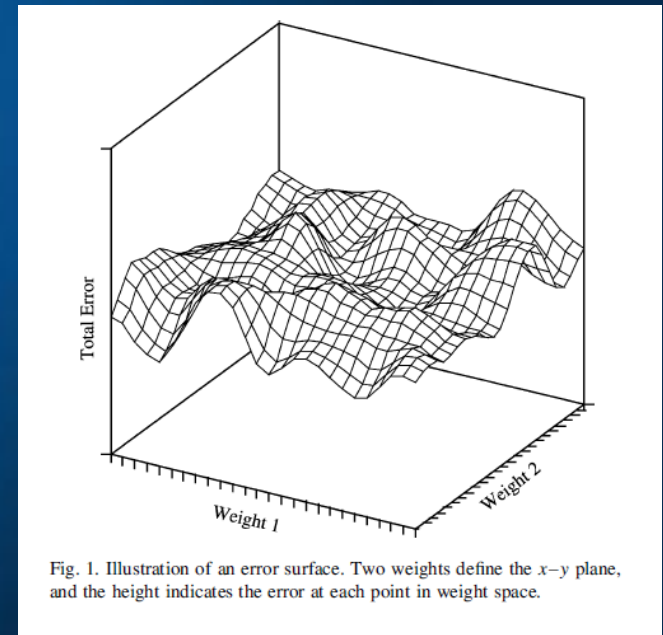


Stochastic Gradient Descent

Issues:

1. $E(.)$ is not convex and not smooth, with many local minima/flat regions. There is no guarantee of convergence.
2. Computation of gradient for batch is expensive.

Wilson, 'On The general inefficiency of batch training for gradient descent learning



Stochastic Gradient Descent

Stochastic Gradient Descent:

1. divide the dataset into small batches of examples, choose samples from different classes
2. compute the gradient using a single batch, make an update,
3. move to the next batch of examples...

Key parameters :

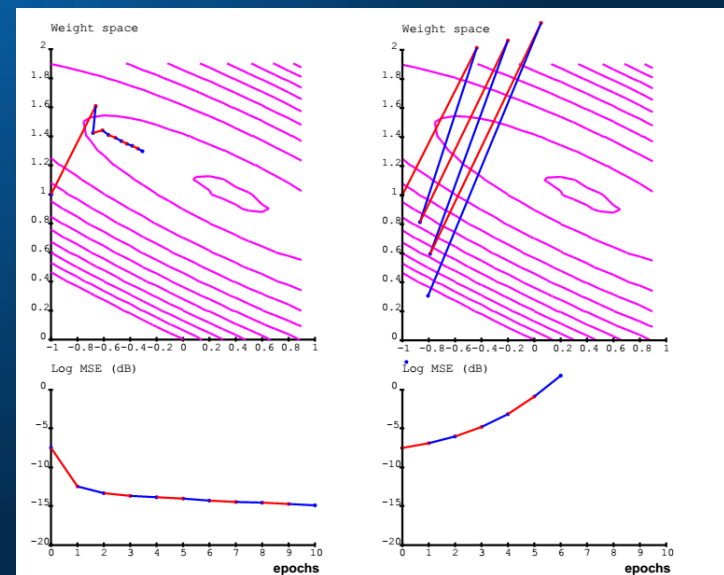
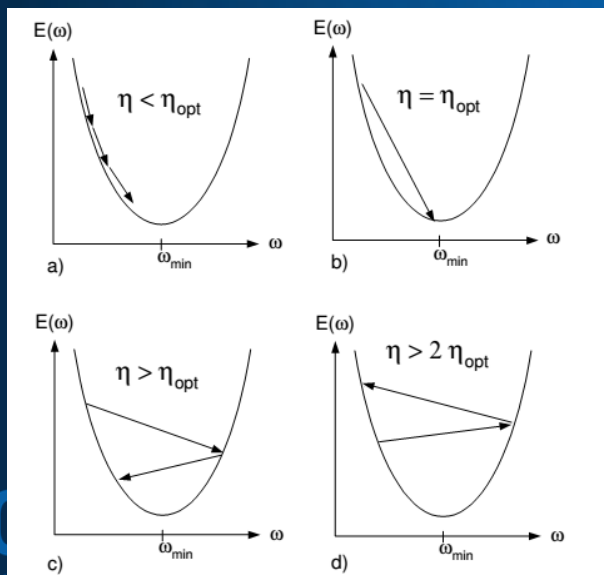
- size of mini-batch
- Learning rate (depends on batch size)
- number of iteration per mini-batch

Wilson " The general inefficiency of batch training for gradient descent learning "

Learning Rate adaptation

$$W(t+1) = W(t) - \lambda(t) * \frac{\partial E}{\partial w}$$

1. Decrease learning rate ("annealing"), e.g. $\lambda(t) = \frac{C}{t}$;
e.g. $\sum_{t=1}^{\infty} \frac{1}{\lambda^2(t)} < \infty$ and $\sum_{t=1}^{\infty} \frac{1}{\lambda(t)} = \infty$
2. Choose different learning rate per layer:
– λ is \sim to square root of # of connections which share weights



Caffe: learning rate adaptation

Caffe supports 4 learning rate policy (see `solver.cpp`):

1. fixed: $\lambda = \text{const}$
2. exp: $\lambda_n = \lambda_0 * \gamma^n$
3. step: $\lambda_n = \lambda_0 * \gamma^{\lceil \frac{n}{step} \rceil}$
4. inverse: $\lambda_n = \lambda_0 * (1 + \gamma * n)^{-c}$

Caffe also supports SGD with momentum and weight decay:

- momentum:

$$\Delta W(t + 1) = \beta * \Delta W(t) + (1 - \beta) * (-\gamma * \frac{\partial E}{\partial w}),$$

- weight decay (regularization of weights) :

$$W(t + 1) = W(t) - \gamma * \frac{\partial E}{\partial w} - \gamma * \theta * W(t)$$

Exercise: Experiment with optimization parameters for CIFAR-10

Going beyond SGD

See Le et al "On Optimization Methods for Deep Learning"

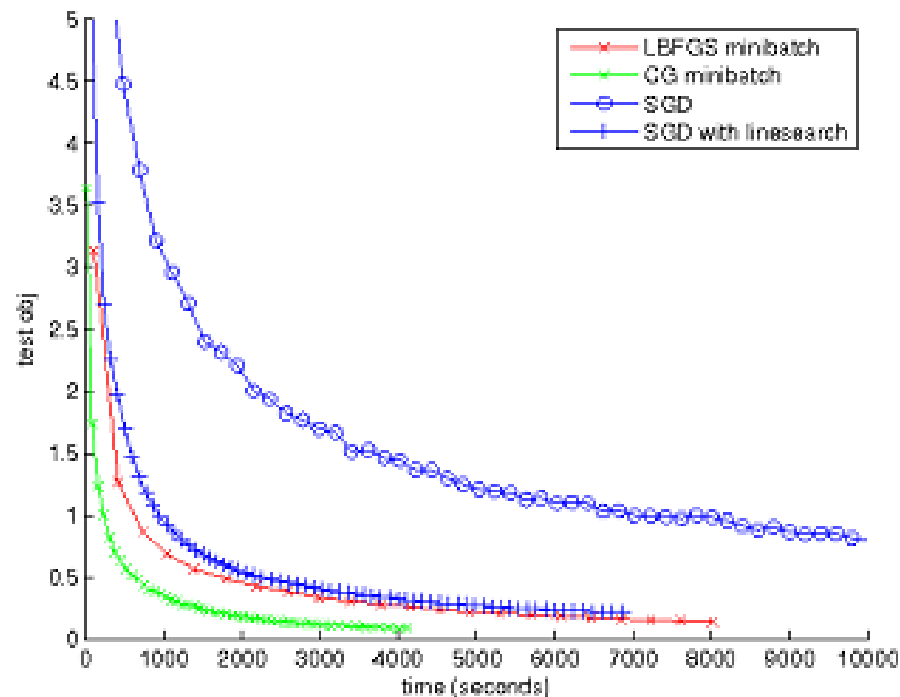


Figure 2. Control experiment with line search for SGDs.

AdaGrad/AdaDelta

Adagrad: adapt learning rate for each weight:

$$\Delta W_{ij}(t+1) = - \frac{\gamma}{\sqrt{\sum_1^{t+1} (\frac{\partial E}{\partial w_{ij}}(\tau))^2}} * \frac{\partial E}{\partial w_{ij}}(t+1)$$

// We can use the same method globally or per layer

AdaDelta: <http://arxiv.org/pdf/1212.5701v1.pdf>.

Idea - accumulate the denominator over last k gradients (sliding window):

$$\alpha(t+1) = \sum_{t-k+1}^{t+1} (\frac{\partial E}{\partial w}(\tau))^2 \text{ and}$$

$$\Delta W(t+1) = - \frac{\gamma}{\sqrt{\alpha(t+1)}} * \frac{\partial E}{\partial w}(t+1) .$$

This requires to keep k gradients. Instead we can use simpler formula:

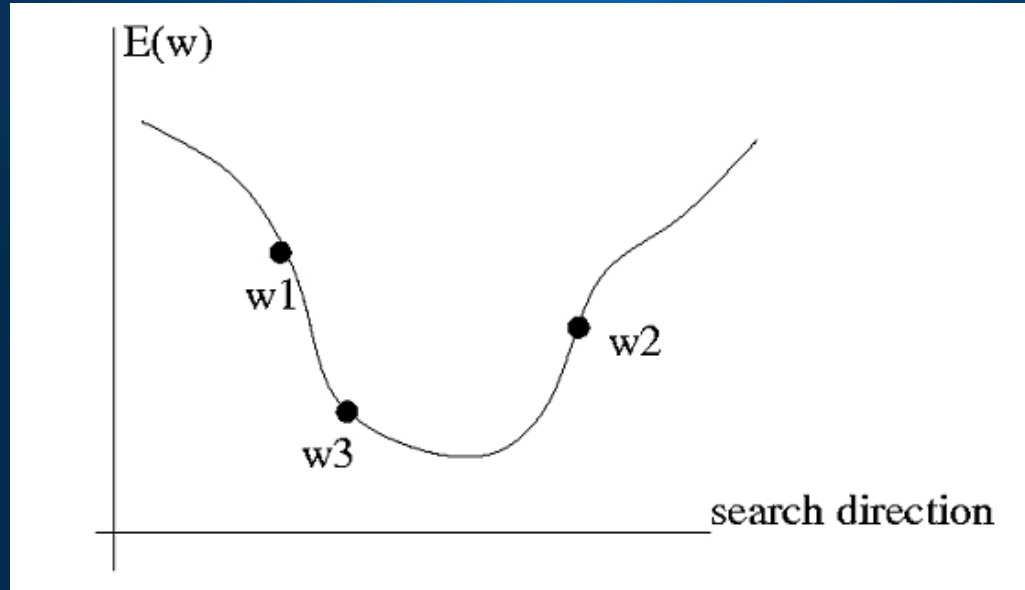
$$\beta(t+1) = \rho * \beta(t) + (1 - \rho) * (\frac{\partial E}{\partial w}(t+1))^2$$

$$\text{and } \Delta W(t+1) = - \frac{\gamma}{\sqrt{\beta(t+1) + \epsilon}} * \frac{\partial E}{\partial w}(t+1) .$$

SGD with Line search

Gradient computation is $\sim 3x$ time more expensive than Forward(.). Rather than take one fixed step in the direction of the negative gradient or the momentum-smoothed negative gradient, we do a search along that direction to find the minimum of the function $E(t)$:

$$E(t) = E(W_n - \lambda * t * \frac{\partial E}{\partial w}(W_n))$$

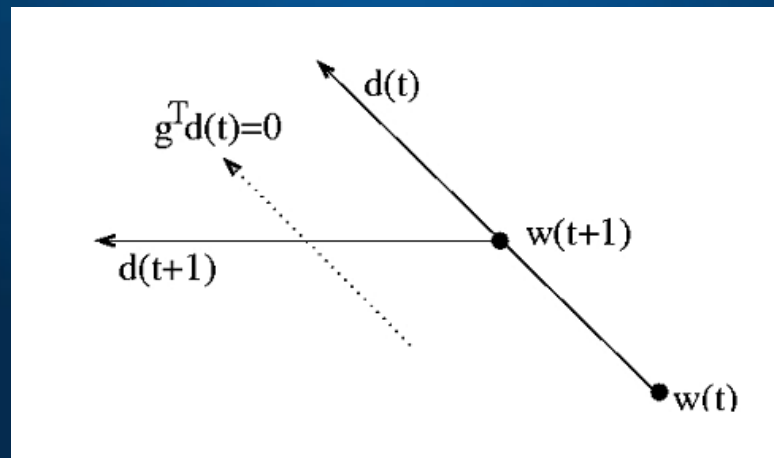


Conjugate Gradient

At the end of a line search, the new gradient is \sim orthogonal to the direction we just searched in. So if we choose the next search direction to be the new gradient, we will always be searching orthogonal directions and things will be slow ! Instead, let's select a new direction so that, to as we move in the new direction the gradient parallel to the old direction stays \sim zero.

The direction of descent:
where β for example :

$$d(t+1) = -\nabla E(w_t) + \beta(t) * d(t) ,$$
$$\beta(k) = \frac{\nabla E(w_{t+1})^T \nabla E(w_{t+1})}{\nabla E(w_t)^T \nabla E(w_t)}$$

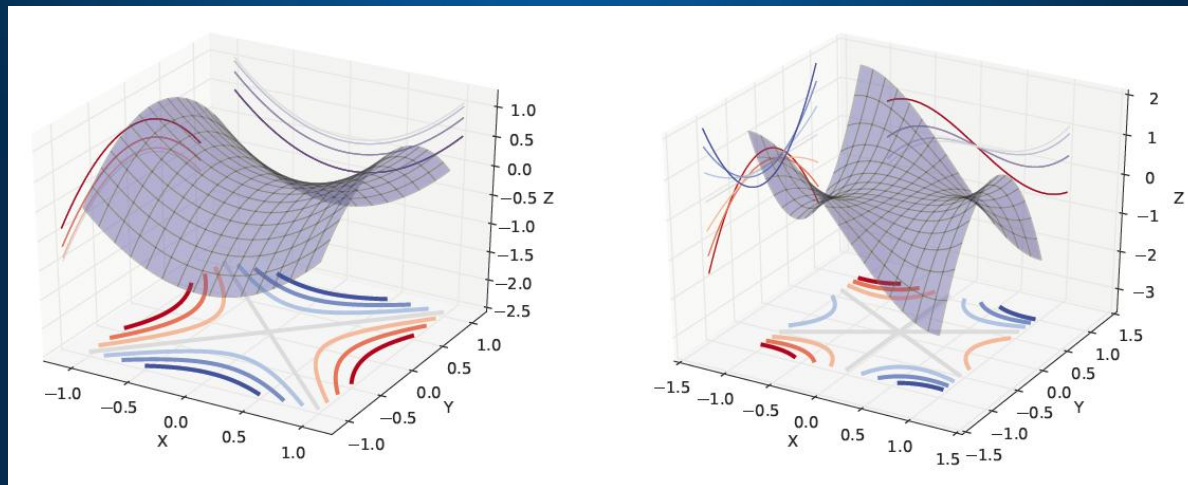


Newton methods near Saddle Points

"The problem with convnets cost functions is not local min, but local saddle points. How Newton methods behave near saddle point?"


R. Pascanu, "On the saddle point problem for non-convex Optimization",
<http://arxiv.org/abs/1405.4604>

Dauphin, "Identifying and attacking the saddle point problem in high-dimensional non-convex optimization",
<http://arxiv.org/pdf/1406.2572v1.pdf>



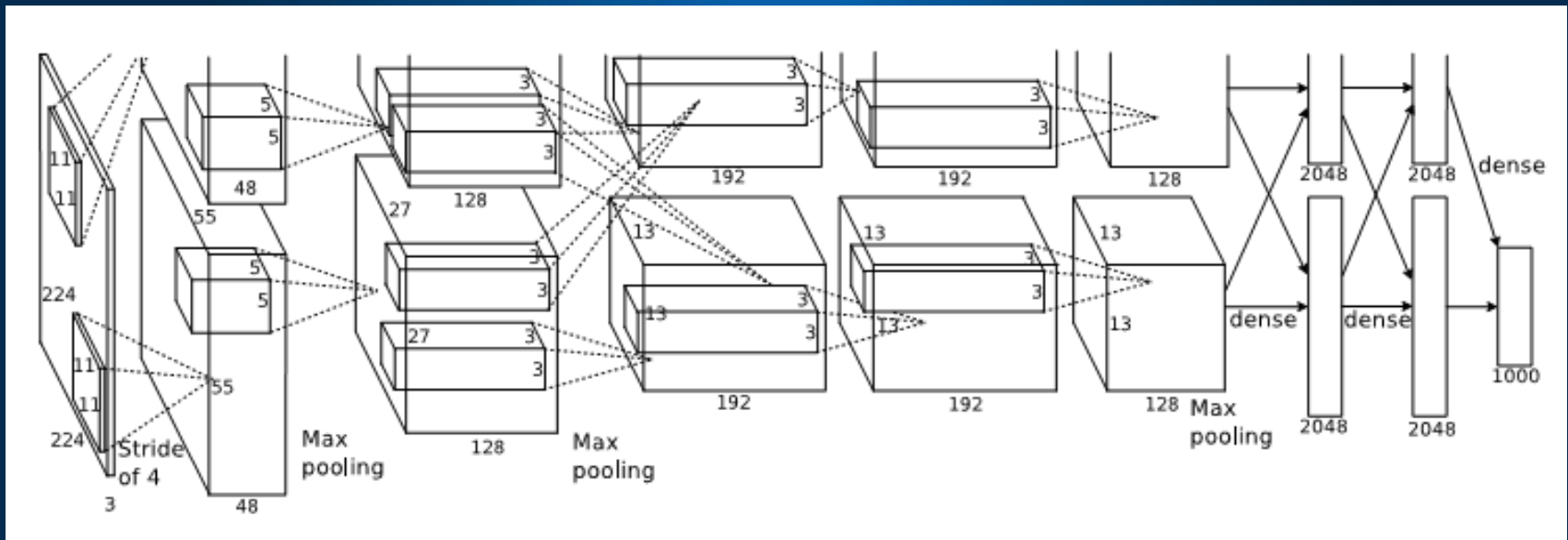
IMAGENET TRAINING

Imagenet Training

1. ILSVRC uses a subset of ImageNet DB with roughly 1 000 images in each of 1 000 categories. In all, there are ~ 1.2 million training images, 50,000 validation images, and 150,000 testing images.
2. LSVRC competition rules:
 <http://www.image-net.org/challenges/LSVRC/2014/>
<http://image-net.org/challenges/LSVRC/2012/>
3. do Imagenet tutorial following the tutorial:
<http://caffe.berkeleyvision.org/gathered/examples/imagenet.html>
The data is already pre-processed and stored in leveldb, so you can start with `./train_imagenet.sh`

AlexNet

Alex K. train his net using two GTX-580 with 3 GB memory.
Training took 6 days: www.cs.toronto.edu/~fritz/absps/imagenet.pdf
Can you train the net with the same performance in 1 day using one Titan Black?



AlexNet Parameters

- batch size of 128 examples,
- momentum of 0.9,
- weight decay of 0.0005.
- Weight initialization: a zero-mean Gaussian distribution with standard deviation 0.01.
- learning rate:
 - The same for all layers,
 - The learning rate was initialized at 0.01 and adjusted manually throughout training: The heuristic which we followed was to divide the learning rate by 10 when the validation error rate stopped improving with the current learning rate
- Dropout for fully connected layer (will discuss tomorrow)
- 90 epochs through whole image dataset.

$$\begin{aligned}v_{i+1} &:= 0.9 \cdot v_i - 0.0005 \cdot \epsilon \cdot w_i - \epsilon \cdot \left\langle \frac{\partial L}{\partial w} \Big|_{w_i} \right\rangle_{D_i} \\w_{i+1} &:= w_i + v_{i+1}\end{aligned}$$

Exercises & Projects

Exercises:

1. Experminet with CIFAR-10:
 - batch size, learning rate, momentum...
2. Read Alex K. paper: <http://www.cs.toronto.edu/~fritz/absps/imagenet.pdf>
3. Train Imagenet. How good you can get till tomorrow 9am?

Home-work:

1. Add to caffe:
 - line-search , CGD, Adagrad/AdaDelta (<http://arxiv.org/pdf/1212.5701v1.pdf>)

Stochastic Diagonal Levenberg-Marquardt