Optimization Techniques

Reading: C.M.Bishop *NNPR* §7

15-496/782: Artificial Neural Networks

Kornel Laskowski

Spring 2004

What is Parameter Optimization?

A fancy name for training: the selection of parameter **values**, which are optimal in some desired sense (eg. minimize an objective function you choose over a dataset you choose). The parameters are the weights and biases of the network.

In this lecture, we will not address learning of network structure. We assume a fixed number of layers and a fixed number of hidden units.

In neural networks, training is typically iterative and time-consuming. It is in our interests to reduce the training time as much as possible.

Lecture Outline

In detail:

1. Gradient Descent (& some extensions)
2. Line Search
3. Conjugate Gradient Search

In passing:

4. Newton's method
5. Quasi-Newton methods

We will not cover Model Trust Region methods (Scaled Conjugate Gradients, Levenberg-Marquart).

Linear Optimization

Applicable to networks with exclusively linear units (and therefore can be reduced to single layer networks).

In one step:

$$\begin{bmatrix} w^*_{1,0} & w^*_{1,1} & w^*_{1,2} & \cdots & w^*_{1,I} \\ w^*_{2,0} & w^*_{2,1} & w^*_{2,2} & \cdots & w^*_{2,I} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w^*_{K,0} & w^*_{K,1} & w^*_{K,2} & \cdots & w^*_{K,I} \end{bmatrix} \begin{bmatrix} 1 & 1 & \cdots & 1 \\ x_1^{(1)} & x_1^{(2)} & \cdots & x_1^{(N)} \\ x_2^{(1)} & x_2^{(2)} & \cdots & x_2^{(N)} \\ \vdots & \vdots & \ddots & \vdots \\ x_I^{(1)} & x_I^{(2)} & \cdots & x_I^{(N)} \end{bmatrix} \asymp \begin{bmatrix} y_1^{(1)} & y_1^{(2)} & \cdots & y_1^{(N)} \\ y_2^{(1)} & y_2^{(2)} & \cdots & y_2^{(N)} \\ \vdots & \vdots & \ddots & \vdots \\ y_K^{(1)} & y_K^{(2)} & \cdots & y_K^{(N)} \end{bmatrix}$$

$$\begin{aligned} \mathbf{W}^* \cdot \mathbf{X} &\asymp \mathbf{Y} \\ \mathbf{W}^* \cdot \mathbf{X} \cdot \mathbf{X}^T &= \mathbf{Y} \cdot \mathbf{X}^T \\ \mathbf{W}^* &= \left(\mathbf{Y} \cdot \mathbf{X}^T\right)\left(\mathbf{X} \cdot \mathbf{X}^T\right)^{-1} \end{aligned}$$

This is linear regression. A good idea to always try first – maybe you don't need non-linearities.

Non-linear Optimization

Given a fixed neural network architecture with non-linearities, we seek iterative algorithms which implement a search in parameter space:

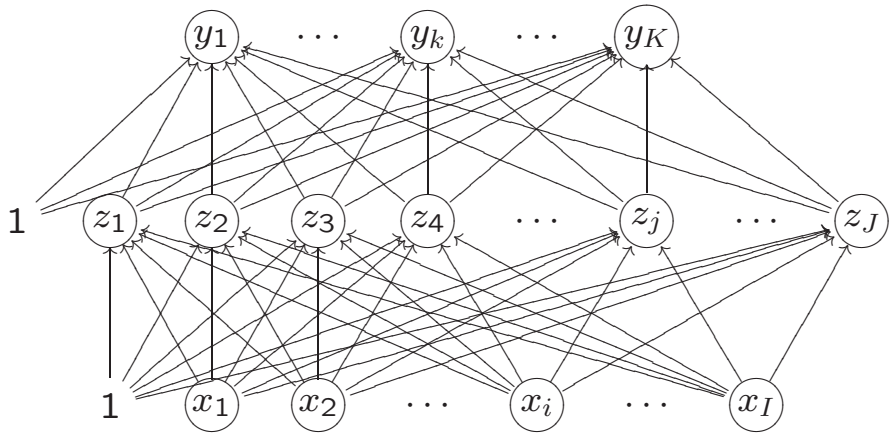
$$\begin{aligned} \mathbf{w}^{(\tau+1)} &= \mathbf{w}^{(\tau)} + \boldsymbol{\Delta}\mathbf{w}^{(\tau)} \\ \mathbf{w} &\in \Re^W \end{aligned}$$

At each timestep τ, $\boldsymbol{\Delta}\mathbf{w}^{(\tau)}$ is chosen to reduce an objective (error) function $E(\{\mathbf{x}, \mathbf{t}\}; \mathbf{w})$. For example, for a network with K linear output units, the appropriate choice is the sum-of-squares error:

$$E = \frac{1}{2} \sum_{n=1}^{N} \sum_{k=1}^{K} [y_k(\mathbf{x}^n; \mathbf{w}) - t_k^n]^2$$

where N is the number of patterns.

The Parameter Space



$$\mathbf{W}_2 = \begin{bmatrix} w_{0,1}^{(2)} & w_{1,1}^{(2)} & \cdots & w_{J,1}^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{0,K}^{(2)} & w_{1,K}^{(2)} & \cdots & w_{J,K}^{(2)} \end{bmatrix} \in \Re^{(K,J+1)}$$

$$\mathbf{W}_1 = \begin{bmatrix} w_{0,1}^{(1)} & w_{1,1}^{(1)} & \cdots & w_{I,1}^{(1)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{0,J}^{(1)} & w_{1,J}^{(1)} & \cdots & w_{I,J}^{(1)} \end{bmatrix} \in \Re^{(J,I+1)}$$

Want to think of (and operate on) weight matrices as a single vector:

$$\mathbf{w} = \textit{mapping}\,(\mathbf{W_1}, \mathbf{W_2}) \in \Re^W, \quad W = J\,(I+1) + K\,(J+1)$$

Doesn't matter what *mapping* is, as long as we can reverse it when necessary.

Approximating Error Surface Behaviour

Holding the dataset $\{\mathbf{x}, \mathbf{t}\}$ fixed, consider a second order Taylor series expansion of $E(\mathbf{w})$ about a point $\mathbf{w}_0$:

$$\overline{E}(\mathbf{w}) = E(\mathbf{w}_0) + (\mathbf{w} - \mathbf{w}_0)^T \mathbf{b} + \frac{1}{2}(\mathbf{w} - \mathbf{w}_0)^T \mathbf{H}(\mathbf{w} - \mathbf{w}_0) \quad (1)$$

where $\mathbf{b}$ is the *gradient* of $E|_{\mathbf{w}_0}$ and $\mathbf{H}$ is the *Hessian* of $E|_{\mathbf{w}_0}$:

$$\mathbf{b} \equiv \nabla E|_{\mathbf{w}_0} = \begin{bmatrix} \frac{\partial E}{\partial w_1} \\ \vdots \\ \frac{\partial E}{\partial w_W} \end{bmatrix}_{\mathbf{w}_0}$$

$$\mathbf{H} \equiv \nabla^2 E|_{\mathbf{w}_0} = \begin{bmatrix} \frac{\partial^2 E}{\partial w_1^2} & \cdots & \frac{\partial^2 E}{\partial w_1 \partial w_W} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 E}{\partial w_W \partial w_1} & \cdots & \frac{\partial^2 E}{\partial w_W^2} \end{bmatrix}_{\mathbf{w}_0}$$

In a similar way, we can define a first order approximation to the gradient:

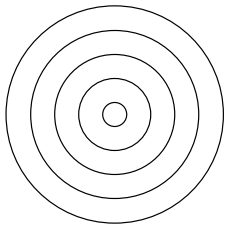
$$\nabla \overline{E}|_{\mathbf{w}} = \mathbf{b} + \mathbf{H}(\mathbf{w} - \mathbf{w}_0) \quad (2)$$

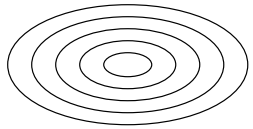
Near a Minimum

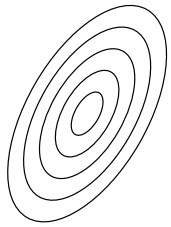
How does the error surface behave near a minimum $\mathbf{w}^*$?

The gradient $\mathbf{b} \equiv \nabla E|_{\mathbf{w}_0} \rightarrow 0$, so **the shape of the error surface is uniquely determined** (to second order) **by the Hessian**:

$$\overline{E}(\mathbf{w}) = E(\mathbf{w}^*) + \frac{1}{2}(\mathbf{w} - \mathbf{w}^*)^T \mathbf{H}(\mathbf{w} - \mathbf{w}^*)$$







$$\mathbf{H} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad \mathbf{H} = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix} \qquad \mathbf{H} = \begin{bmatrix} 2 & 1 \\ 1 & 3 \end{bmatrix}$$

General Algorithm Structure

STEP 1. $\tau = 0$. Initialize $\mathbf{w}^{(0)}$. Covered in §7.4.

while (TRUE) {

STEP 2. Compute $\boldsymbol{\Delta}\mathbf{w}^{(\tau)}$.
Update $\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} + \boldsymbol{\Delta}\mathbf{w}^{(\tau)}$.
Update $\tau = \tau + 1$.

STEP 3. If termination reached (covered in next lecture),
exit and return $\mathbf{w}^* = \mathbf{w}^{(\tau)}$.

}

Gradient Descent Search

$$\Delta \mathbf{w}^{(\tau)} = \eta \cdot \underbrace{-\nabla E|_{\mathbf{w}^{(\tau)}}}_{\text{(search) direction}} \tag{3}$$

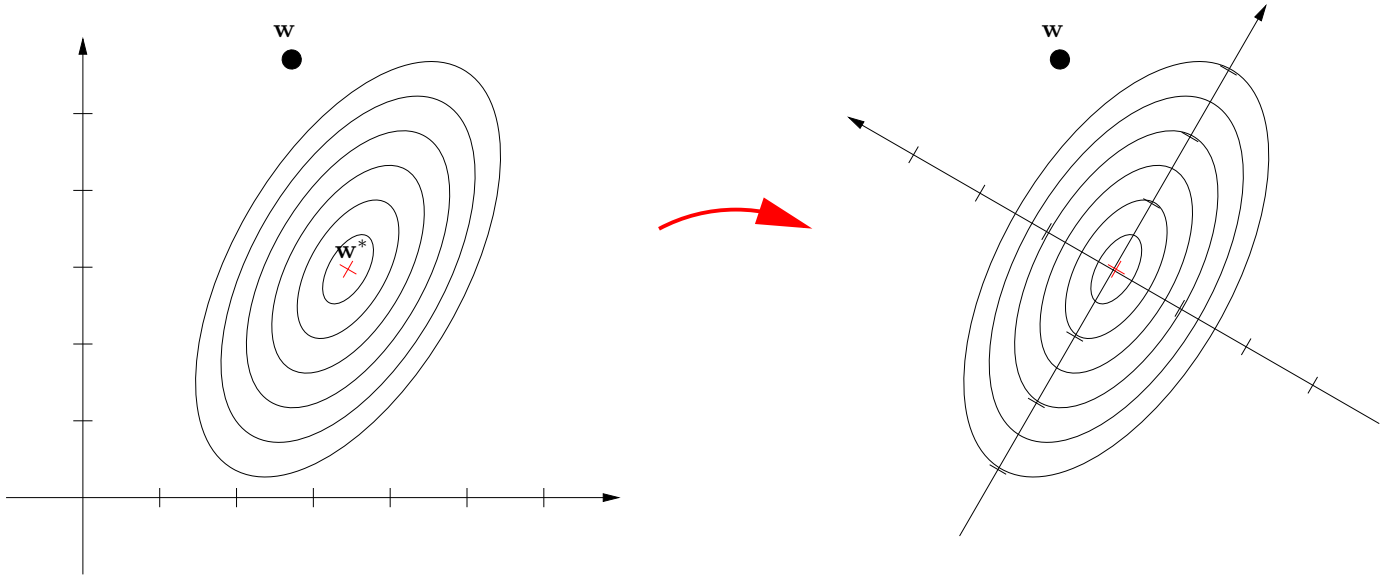
As we've seen, the **step size** η is not a function of $\mathbf{w}^{(\tau)}$ and is known in as the **learning rate**. In the most basic form of Gradient Descent Search, it's just a constant, and you have to set it by hand.

At each timestep τ, $\Delta \mathbf{w}$ is a function of the **current true gradient** only. The search history is ignored. As we will see, later algorithms will exploit the search history.

Conditions for Convergence

Under what conditions will we converge to a minimum?

Hard to say. But if we **assume a quadratic error surface**, we can answer this if we rotate into the eigenspace of $\mathbf{H}$ and move such that $\mathbf{w}^* \mapsto \mathbf{0}$ (assume we are doing a post-mortem and know both $\mathbf{H}$ and $\mathbf{w}^*$).



The Eigenspace of $\mathbf{H}$

Since $\mathbf{H} = \mathbf{H}^T$, its eigenvectors

$$\mathbf{H}\mathbf{u}_i = \lambda_i \mathbf{u}_i \quad i \in [1, W] \tag{4}$$

form a complete orthonormal basis of $\Re^W$. Why?

Because for any pair, $\mathbf{u}_j$ and $\mathbf{u}_k$, $j \neq k$,

$$\begin{aligned} \mathbf{u}_j^T \mathbf{H} \mathbf{u}_k &= \lambda_k \mathbf{u}_j^T \mathbf{u}_k \\ \left(\mathbf{u}_k^T \mathbf{H} \mathbf{u}_j\right)^T &= \left(\lambda_j \mathbf{u}_k^T \mathbf{u}_j\right)^T \\ \Rightarrow \; 0 &= \left(\lambda_k - \lambda_j\right) \mathbf{u}_k^T \mathbf{u}_j \end{aligned}$$

So for $\lambda_k \neq \lambda_j$, the eigenvectors are orthogonal. Otherwise, they define a plane in which all vectors are eigenvectors corresponding to $\lambda_k = \lambda_j$, so we can choose any two orthogonal vectors which span this plane. Then normalize all $\mathbf{u}_i$ to have unit length.

The Gradient in the Eigenspace of $\mathbf{H}$

We've successfully constructed a new (more useful) orthonormal basis of our original space. We can express any original vector $\mathbf{w}$ in these new coordinates as:

$$\mathbf{w} = \mathbf{w}^* + \sum_{i=1}^{W} \alpha_i \mathbf{u}_i \quad \Leftrightarrow \quad \mathbf{w} - \mathbf{w}^* = \sum_{i=1}^{W} \alpha_i \mathbf{u}_i \tag{5}$$

Recalling Eq 2, we compute the gradient of our second order approximation at $\mathbf{w}$ using Eq 5 and simplify with Eq 4:

$$\begin{aligned} \nabla \overline{E}|_{\mathbf{w}} &= \mathbf{H}(\mathbf{w} - \mathbf{w}^*) \\ &= \mathbf{H} \sum_{i=1}^{W} \alpha_i \mathbf{u}_i \\ &= \sum_{i=1}^{W} \alpha_i \lambda_i \mathbf{u}_i \end{aligned} \tag{6}$$

Weight Update in the Eigenspace of H

Let's transform our weight increment from the original coordinate system to the new coordinates using Eq 5:

$$\begin{aligned}
\Delta\mathbf{w}^{(\tau)} &= \mathbf{w}^{(\tau+1)} - \mathbf{w}^{(\tau)} \\
&= \left(\mathbf{w}^* + \sum_{i=1}^{W} \alpha_i^{(\tau+1)}\mathbf{u}_i\right) - \left(\mathbf{w}^* + \sum_{i=1}^{W} \alpha_i^{(\tau)}\mathbf{u}_i\right) \\
&= \sum_{i=1}^{W} \Delta\alpha_i^{(\tau)}\mathbf{u}_i
\end{aligned}$$

Recall that for Gradient Descent Search,

$$\begin{aligned}
\Delta\mathbf{w}^{(\tau)} &= -\eta\nabla E|_{\mathbf{w}^{(\tau)}} \\
&= -\eta\nabla\overline{E}|_{\mathbf{w}^{(\tau)}} \qquad (7) \\
&= -\eta\sum_{i=1}^{W} \alpha_i^{(\tau)}\lambda_i\mathbf{u}_i \qquad (8)
\end{aligned}$$

where the equality in line 7 holds because we have been assuming a quadratic error surface, and Eq 8 follows from Eq 6.

Gradient Descent Search Convergence, cont'd...

By inspection from Eqs 7 & 8, $\Delta\alpha_i = -\eta\lambda_i\alpha_i$. So, in the new coordinate system,

$$\begin{aligned} \alpha_i^{(\tau+1)} &= \alpha_i^{(\tau)} + \Delta\alpha_i^{(\tau)} \\ &= (1 - \eta\lambda_i)\,\alpha_i^{(\tau)} \end{aligned} \tag{9}$$

Remember that we want to converge to $\mathbf{0}$ ($\mathbf{w}^*$ in the original basis). After R timesteps,

$$\lim_{R\to\infty} \alpha_i^{(R)} = \lim_{R\to\infty} (1 - \eta\lambda_i)^R \alpha_i^{(0)} \tag{10}$$

$$= 0 \tag{11}$$

$$\Rightarrow \lim_{R\to\infty} \mathbf{w}^{(R)} = \mathbf{w}^* \tag{12}$$

$$\text{iff } |1 - \eta\lambda_i| < 1 \tag{13}$$

Finally (it took 5 slides). We will converge to the minimum if and only if $\eta < \frac{2}{\lambda_{max}}$.

Speed of Convergence

Governed by how long it takes each of the $(1-\eta\lambda_i)^R$ to decay to 0, under the constraint that $\eta < 2/\lambda_{max}$.

The larger λ_i, the closer $(1-\eta\lambda_i)$ is to 0 and so the faster will $(1-\eta\lambda_i)^R$ decay. Therefore convergence is slowest for the smallest λ_i. Specifically, convergence will be governed by how close the following gets to zero:

$$\begin{aligned}\max_{\lambda_i}(1-\eta\lambda_i) &= \max_{\lambda_i}\left(1-\frac{2}{\lambda_{max}}\lambda_i\right) \\ &= 1-\frac{2\lambda_{min}}{\lambda_{max}}\end{aligned}$$

So the closer λ_{min} and λ_{max} are, the faster we will find the minimum.

The situation becomes even less optimistic when we drop our assumption of a quadratic error surface. Then $\mathbf{H}$ may not only be ill-conditioned, it is generally different for every point in weight space.

Gradient Descent Search Pros & Cons

Straightforward, iterative, tractable, locally optimal descent in error — see demo. But we have four main objections:

1. Cannot avoid local minima, and cannot escape them (but may occasionally overshoot them).

2. Cannot guarantee a scalable bound on time complexity. Rather, speed to convergence governed by the condition number of the local Hessian, which may be changing from point to point.

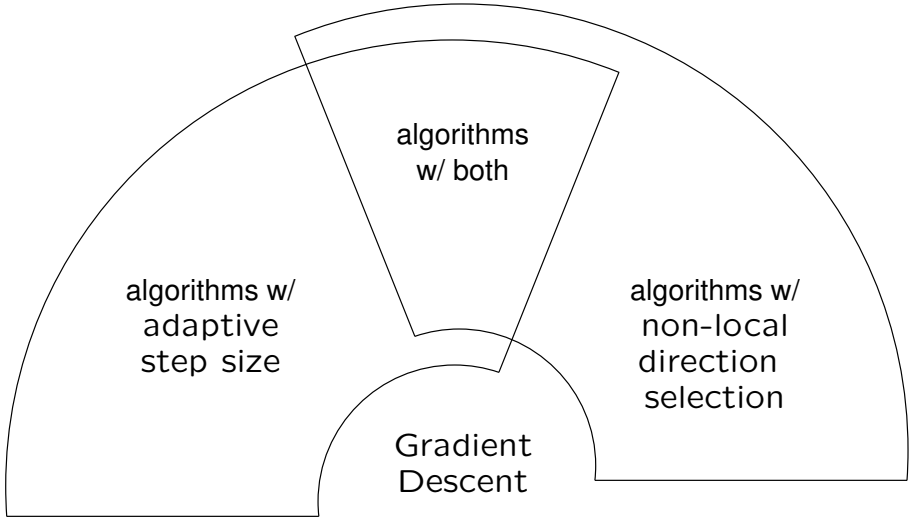
3. Step size constant, not sensitive to local topology. Furthermore, has to be carefully set by hand.

4. Search direction only locally optimal.

Addressing the Limitations

The rest of this lecture.

Want to find techniques for adapting the step size and for making direction decisions which are optimal in more than just a local sense.



Also want to try to limit time complexity, storage complexity, and the number of parameters which must be externally set by hand.

Sadly, can't do much about local minima.

Avoiding/Escaping Local Minima

The only way to avoid getting trapped in a local minimum is to accidentally step over it (with a step size or inertia which is locally too high). The likelihood of this occurring depends on the optimization technique.

Leaving local minima is possible by random perturbation. Simulated Annealing and Genetic Algorithms are examples, but we won't cover them in this lecture.

Stochastic Gradient Descent is a form of injecting randomness into Gradient Descent.

Stochastic Gradient Descent

Rather than computing the error gradient for all patterns, could go through the patterns sequentially, one pattern per iteration.

$$\Delta \mathbf{w}^{(\tau)} = \eta_s \cdot -\nabla E^n|_{\mathbf{w}^{(\tau)}}$$

In contrast to the batch version, this offers the possibility of escaping from local minima.

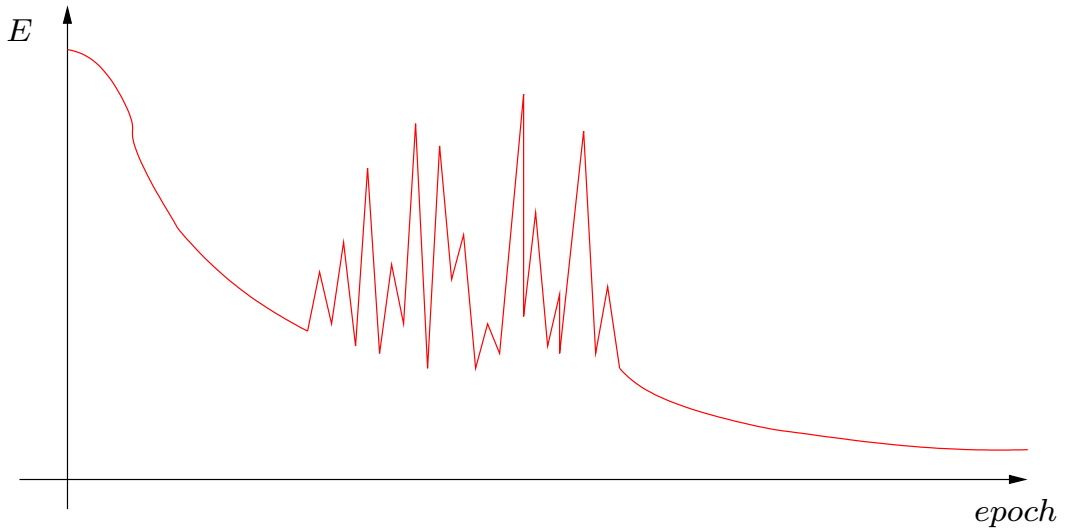
Likely to be more efficient for datasets with high redundancy.

Note that value of η_s may be different than for the η in the batch version.

Strictly speaking, it's stochastic only if you choose the patterns randomly.

Using Learning Rate Schedules

In practice, error vs epoch curves exhibit distinct regions where some definite statements about the suitability of the learning rate can be made.

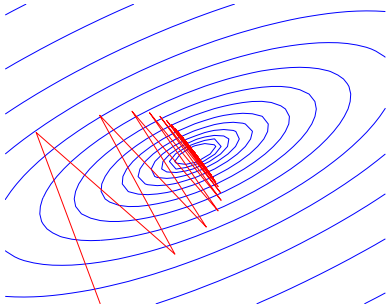


Once you are familiar with how your neural network is training you will be able to guess at several learning rates, which you will want to kick in at specific times.

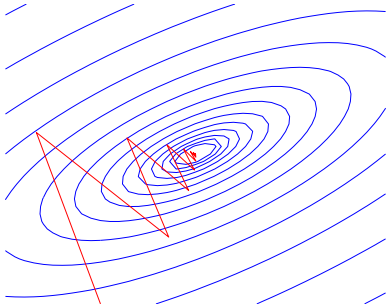
You'll see this in an upcoming homework assignment.

Adding Momentum

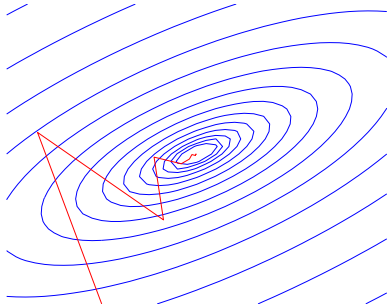
$$\Delta \mathbf{w}^{(\tau)} = -\eta \nabla E|_{\mathbf{w}^{(\tau)}} + \mu \Delta \mathbf{w}^{(\tau-1)} \tag{14}$$



$\mu = 0.0$



$\mu = 0.1$



$\mu = 0.2$

Generally leads to significant improvements in speed of convergence. See demo.

Escape from local minima possible.

Yet another parameter to set manually.

Line Search

Let's try to do something about the step size.

At every timestep τ, run a small subalgorithm:

1. Choose a search direction as in gradient descent:

$$\mathbf{d}^{(\tau)} \;=\; -\nabla E|_{\mathbf{w}^{(\tau)}}$$

2. Minimize the error along the search direction:

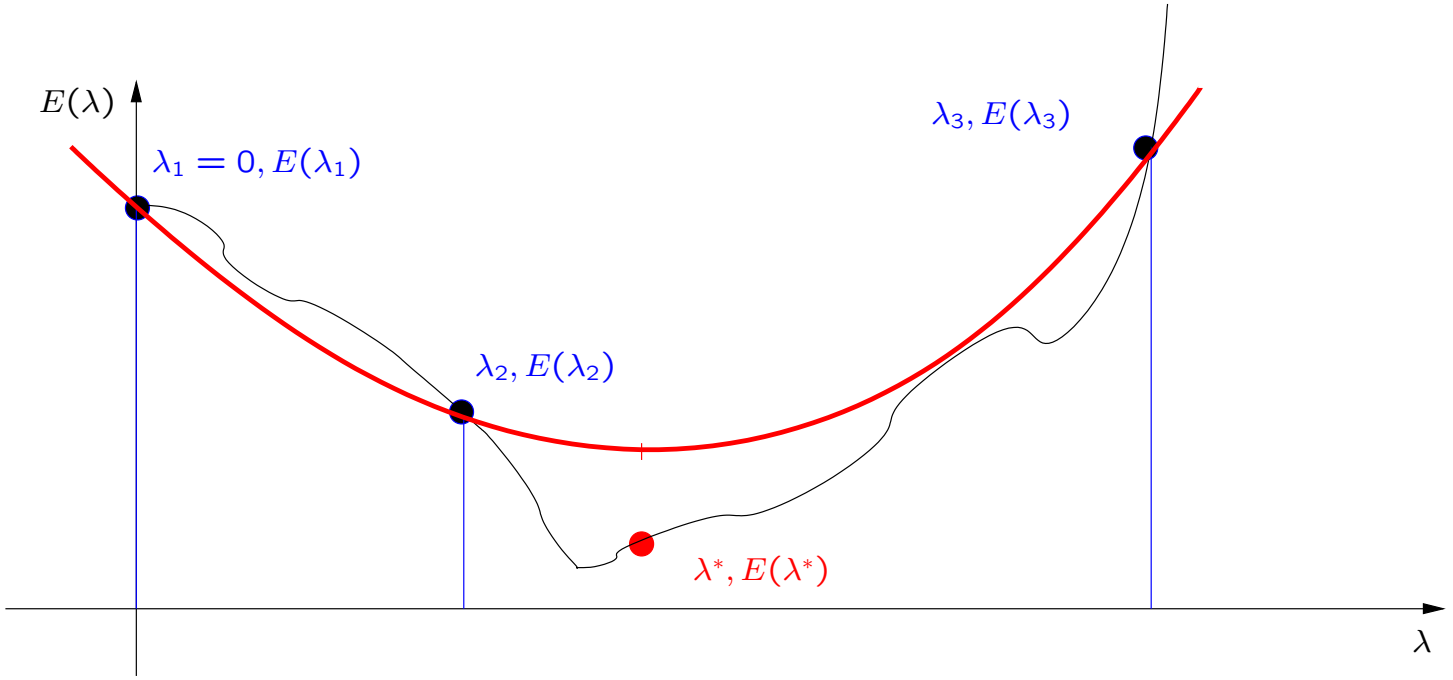
$$\lambda^{(\tau)} \;=\; \text{argmin}_{\lambda>0}\; E(\mathbf{w}^{(\tau)} + \lambda \mathbf{d}^{(\tau)})$$

   NOTE: This is a one-dimensional problem!

3. Return the weight update:

$$\mathbf{\Delta w}^{(\tau)} \;=\; \lambda^{(\tau)} \cdot \mathbf{d}^{(\tau)}$$

Minimizing Error in 1 Dimension



2a. Pick 3 values $\lambda_1 = 0 < \lambda_2 < \lambda_3$ such that $E(\lambda_2)$ is smallest. Enter inner loop:

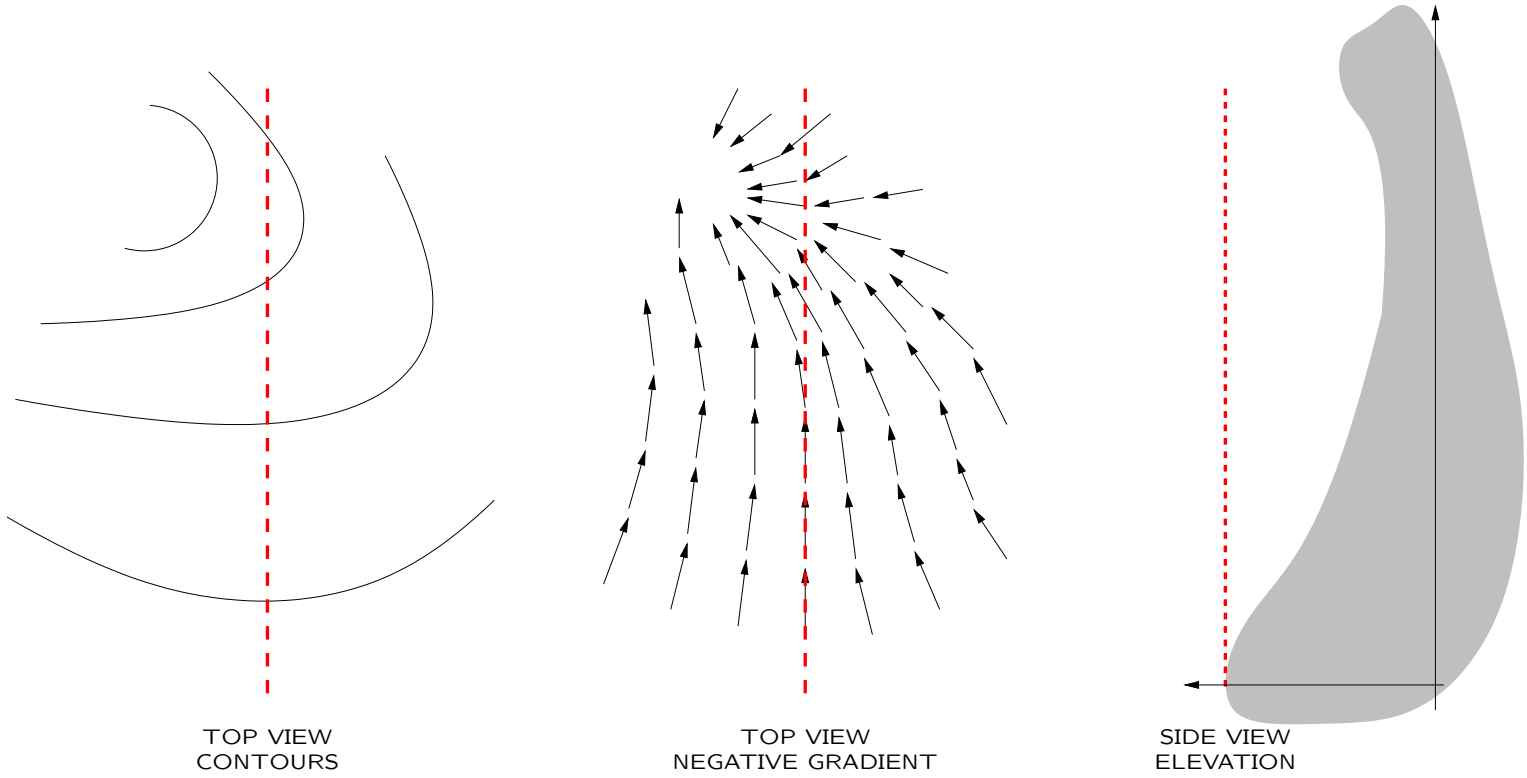
2b. Fit points to a parabola $E_i = a\lambda_i^2 + b\lambda_i + c$, $1 \leq i \leq 3$ (linear regression).

2c. Compute $\lambda^* = -\frac{b}{2a}$, the parabola's minimum; evaluate $E(\lambda^*)$.

2d. If $E(\lambda^*) \approx E(\lambda_2)$, return λ^* and exit.

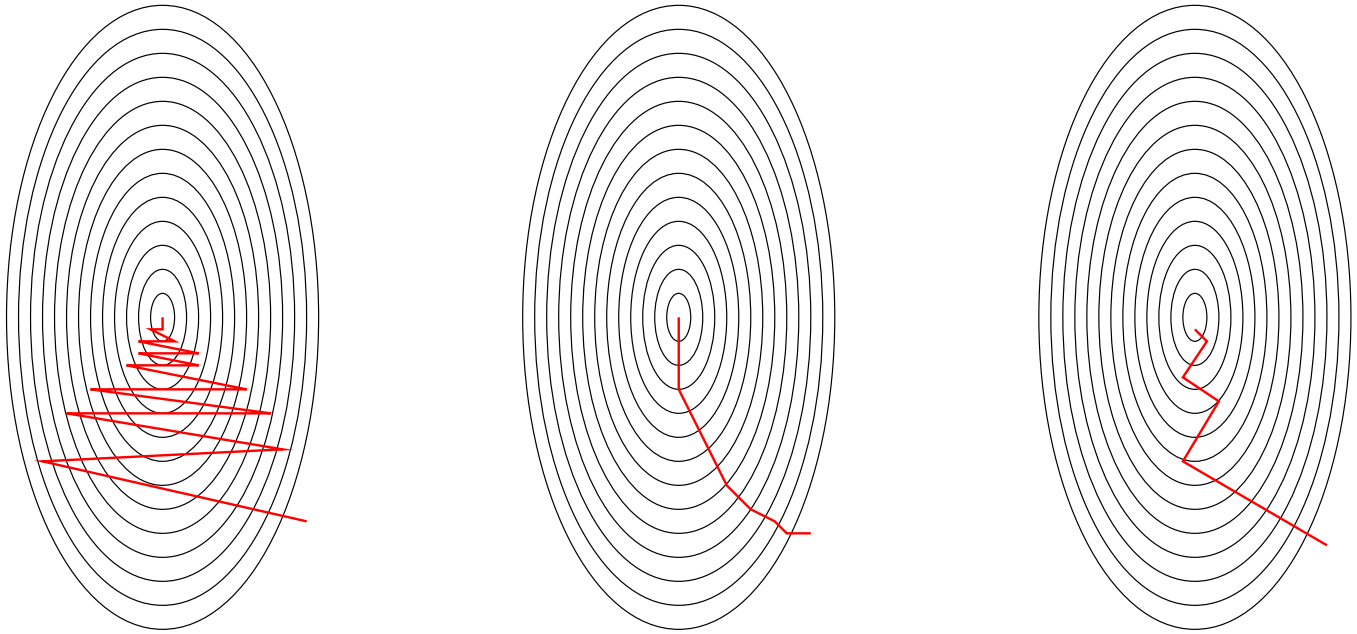
2e. Else replace $\{\lambda_2, E(\lambda_2)\}$ by $\{\lambda^*, E(\lambda^*)\}$ and goto 2b.

A Bird's Eye View of Line Search



As we move in a straight line, the gradient beneath our feet keeps changing (we're on a curved surface). Line search will stop us when the component of the gradient parallel to our direction of travel is zero. **Consequence:** successive directions in Line Search are perpendicular.

How good are $\perp$ Search Directions?



Gradient Descent, large η | Gradient Descent, small η | Line Search

In Gradient Descent Search, we may be oscillating back and forth such that the angle between successive directions is almost 180 degrees. But we have the opportunity, by keeping η small, to follow the optimal path to the minimum.

Orthogonal Search Directions, cont'd...

In Line Search, by contrast, we are stuck with successive directions that are at exactly 90 degrees to each other, always (see demo). This is an immutable fact, regardless of the topology.

This means that, even for a 2-dimensional quadratic error surface, we will take many steps to converge.

The picture is far messier when we consider a W-dimensional parameter space with W large (ALVINN $W \approx 4000$). Throw away the quadratic assumption, and things get messier still.

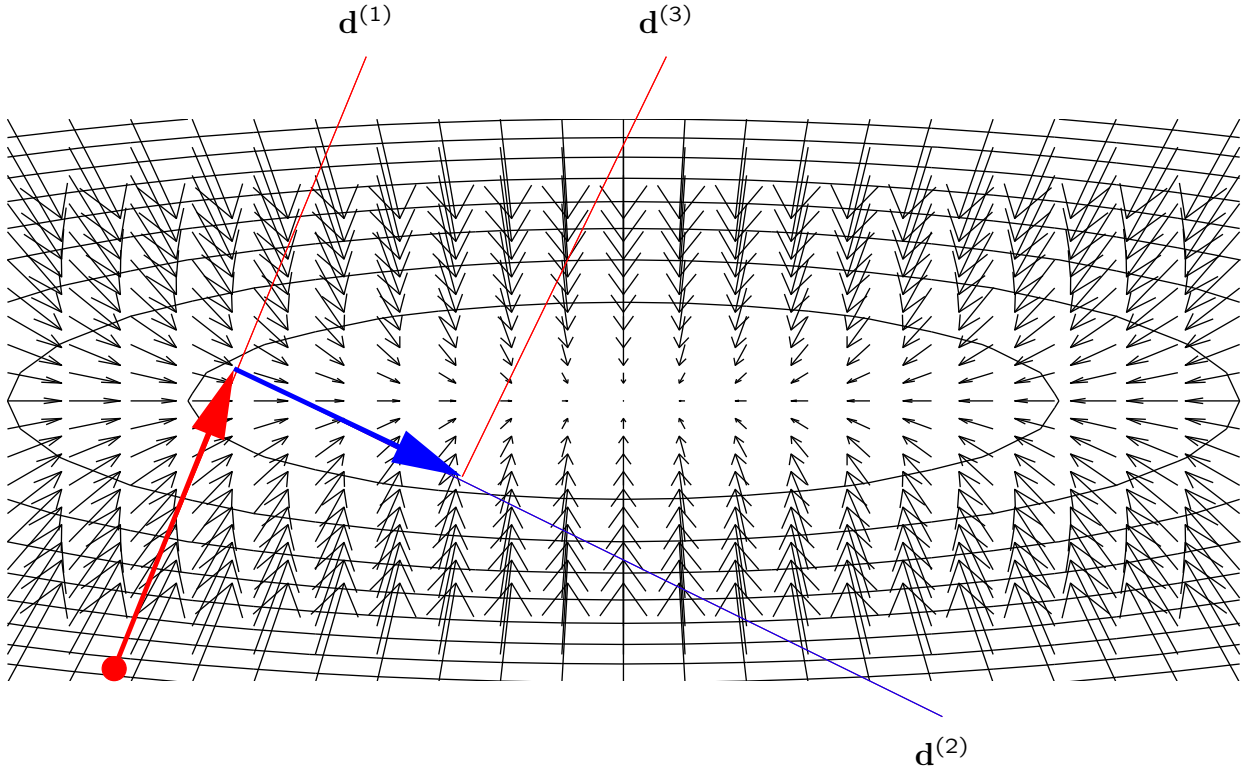
Line Search Pros & Cons

1. No parameters to set by hand, yay!

2. Successive search directions are orthogonal. Don't quite know whether this is a blessing or a curse. Seems better than Gradient Descent Search, but still somehow unsatisfactory. Could take a long time to converge.

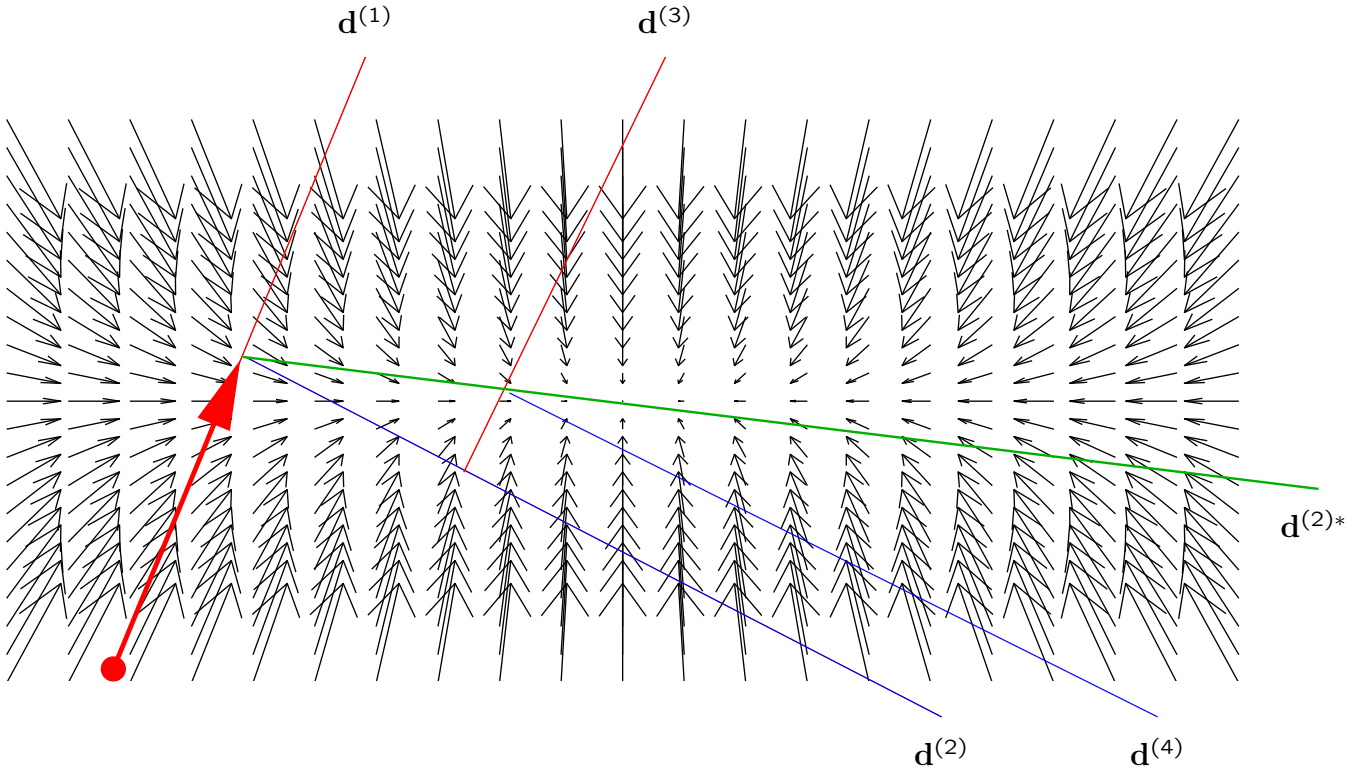
3. Line search requires that we evaluate $E(\lambda)$ many times per iteration. Each time we have to do a full forward propagation of the training set through our neural network.

What Can We Do About the $\perp$ Directions



Note that, at each step, we choose a direction which ends up undoing the progress we made on the previous step. We'll have to make it up again later.

Conjugate Gradient Search



At $\tau = 2$, we would really like to take direction $\mathbf{d}^{(2)*}$ instead. This would avoid us having to repeat progress already made during step 1 on the next step.

Conjugate Gradients, Theory Part 1

At $\mathbf{w}^{(\tau+1)}$,

$$\left(-\nabla E|_{\mathbf{w}^{(\tau+1)}}\right)^T \mathbf{d}^{(\tau)} = 0 \tag{15}$$

Once we choose the next direction $\mathbf{d}^{(\tau+1)}$ and begin a line search step, our position along that direction will be $\mathbf{w}^{(\tau+1)} + \lambda \mathbf{d}^{(\tau+1)}$ and our gradient at that position will be $\nabla E|_{\mathbf{w}^{(\tau+1)}+\lambda \mathbf{d}^{(\tau+1)}}$.

To a first order approximation,

$$\begin{aligned} \nabla \overline{E}|_{\mathbf{w}^{(\tau+1)}+\lambda \mathbf{d}^{(\tau+1)}} &= \nabla E|_{\mathbf{w}^{(\tau+1)}} \\ &\quad + \left(\mathbf{w}^{(\tau+1)} + \lambda \mathbf{d}^{(\tau+1)} - \mathbf{w}^{(\tau+1)}\right)^T \cdot \mathbf{H} \\ &= \nabla E|_{\mathbf{w}^{(\tau+1)}} + \lambda \mathbf{d}^{(\tau+1)\,T} \mathbf{H} \end{aligned} \tag{16}$$

Conjugate Gradients, Theory Part 2

We want to have chosen our next direction $\mathbf{d}^{(\tau+1)}$ such that, to a first order approximation, our gradient along this direction will remain orthogonal to the previous direction $\mathbf{d}^{(\tau)}$:

$$\left(-\nabla\overline{E}|_{\mathbf{w}^{(\tau+1)}+\lambda\mathbf{d}^{(\tau+1)}}\right)^T \mathbf{d}^{(\tau)} = 0 \tag{17}$$

We can substitute Eq 16 into Eq 17

$$\begin{aligned} -\left(\nabla E|_{\mathbf{w}^{(\tau+1)}} + \lambda\mathbf{d}^{(\tau+1)\,T}\mathbf{H}\right)^T \mathbf{d}^{(\tau)} &= 0 \\ \left(-\nabla E|_{\mathbf{w}^{(\tau+1)}}\right)^T \mathbf{d}^{(\tau)} - \lambda\mathbf{d}^{(\tau+1)\,T}\mathbf{H}\mathbf{d}^{(\tau)} &= 0 \\ \mathbf{d}^{(\tau+1)\,T}\mathbf{H}\mathbf{d}^{(\tau)} &= 0 \end{aligned} \tag{18}$$

where the first term on the second line cancels due to Eq 15.

Pairs of directions $\mathbf{d}^{(\tau+1)}$ and $\mathbf{d}^{(\tau)}$ for which Eq 18 holds are called mutually *conjugate*. They are orthogonal in the (rotated) space where $\mathbf{H}$ is the identity.

Constructing the Next Conjugate Direction

Note that the new direction is a linear combination of the current negative gradient and the previous search direction:

$$\mathbf{d}^{(\tau+1)} = -\nabla E|_{\mathbf{w}^{(\tau+1)}} + \beta^{(\tau)}\mathbf{d}^{(\tau)} \tag{19}$$

We can solve for $\beta^{(\tau)}$ by first taking the transpose of Eq 19 and then multiplying by $\mathbf{H}\mathbf{d}^{(\tau)}$ and imposing Eq 18:

$$\mathbf{d}^{(\tau+1)\,T}\mathbf{H}\mathbf{d}^{(\tau)} = -\left(\nabla E|_{\mathbf{w}^{(\tau+1)}}\right)^T \mathbf{H}\mathbf{d}^{(\tau)} + \beta^{(\tau)}\mathbf{d}^{(\tau)\,T}\mathbf{H}\mathbf{d}^{(\tau)}$$

This yields

$$\beta^{(\tau)} = \frac{\left(\nabla E|_{\mathbf{w}^{(\tau+1)}}\right)^T \mathbf{H}\mathbf{d}^{(\tau)}}{\mathbf{d}^{(\tau)\,T}\,\mathbf{H}\mathbf{d}^{(\tau)}} \tag{20}$$

Can Computation of $\mathbf{H}$ Be Avoided?

$\mathbf{H}$ is costly to compute, and so we would like to avoid its evaluation at every step. Under a quadratic error surface assumption, it can be shown that Eq 20 reduces to the *Polak-Ribiere* formula:

$$\beta^{(\tau)} = \frac{\left(\nabla E|_{\mathbf{w}(\tau+1)}\right)^T \left(\nabla E|_{\mathbf{w}(\tau+1)} - \nabla E|_{\mathbf{w}(\tau)}\right)}{\left(\nabla E|_{\mathbf{w}(\tau)}\right)^T \nabla E|_{\mathbf{w}(\tau)}} \tag{21}$$

There are several competing expressions; this one is believed to generalize better to non-quadratic error surfaces.

The Conjugate Gradient Search Algorithm

Just like the inner-loop subalgorithm for Line Search. Prior to incrementing τ:

1a. Compute $-\nabla E|_{\mathbf{w}^{(\tau+1)}}$ (one back propagation).

1b. For $\tau = 1$, set $\beta^{(1)} = 0$. Else evaluate $\beta^{(\tau)}$ using Eq 21.

1c. Evaluate $\mathbf{d}^{(\tau+1)}$ using Eq 19.

1d. Update $\tau = \tau + 1$.

Check out the demo.

How Does Conjugate Gradient Search Stack Up?

If we had a quadratic error surface, we would need to perform at most W weight updates before reaching the minimum. 2-D toy problems will be solved two steps.

Need to store previous search direction ($O(W)$ storage). But this isn't so bad.

Require multiple evaluations of the error (forward propagations through the neural network) during line error minimization.

We need an accurate line error minimization routine since we are using our position to set up a conjugate system. In Line Search this wasn't so important.

Still no chance of leaving poor local minima.

Newton's Method

For a quadratic error surface, our 1st order approximation to the gradient (Eq 2) is the true gradient. Expanding about $\mathbf{w}^*$ and noting that the gradient at $\mathbf{w}^*$ is 0:

$$\nabla E|_{\mathbf{w}} = \mathbf{H}(\mathbf{w} - \mathbf{w}^*) \tag{22}$$

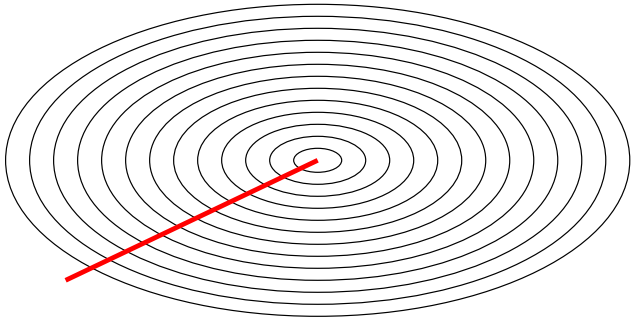
Eq 22 can be solved directly

$$\mathbf{w}^* = \mathbf{w} - \mathbf{H}^{-1}\nabla E|_{\mathbf{w}}$$

and the corresponding weight update,

$$\Delta\mathbf{w} = -\mathbf{H}^{-1}\nabla E|_{\mathbf{w}}$$

is called the Newton step.



Newton's Method: Caveats

1. The error surface isn't really quadratic; algorithm must be applied iteratively like everything else.

2. Computation of $\mathbf{H}^{-1}$ is $O(W^3)$.

3. Points 1 and 2 together should make you cringe. In ALVINN, with 4000 parameters, that's 6.4×10^{10} computations per iteration.

4. Additionally, if $\mathbf{H}$ isn't positive definite, the algorithm could fail to find a minimum.

Quasi-Newton Methods

AIM: Since exact computation of $\mathbf{H}^{-1}$ is so expensive, let's find an approximation $\mathbf{G}^{(\tau)}$ which is cheaper to compute and simultaneously ensure that it is positive definite.

At $\tau = 1$, initialize $\mathbf{G}^{(1)} = \mathbf{I}$.

At each timestep τ, generate a new $\mathbf{G}^{(\tau)}$. The $\mathbf{G}^{(\tau)}$ are a sequence of increasingly better approximations to $\mathbf{H}^{-1}$.

Then apply the Newton step weight update. Use line minimization just to make sure we're not taken outside of the range of validity of our quadratic approximation:

$$\Delta \mathbf{w}^{(\tau)} = -\lambda^{(\tau)} \mathbf{G}^{(\tau)} \nabla E|_{\mathbf{w}^{(\tau)}}$$

Computing $\mathbf{G}^{(\tau+1)}$ using BFGS

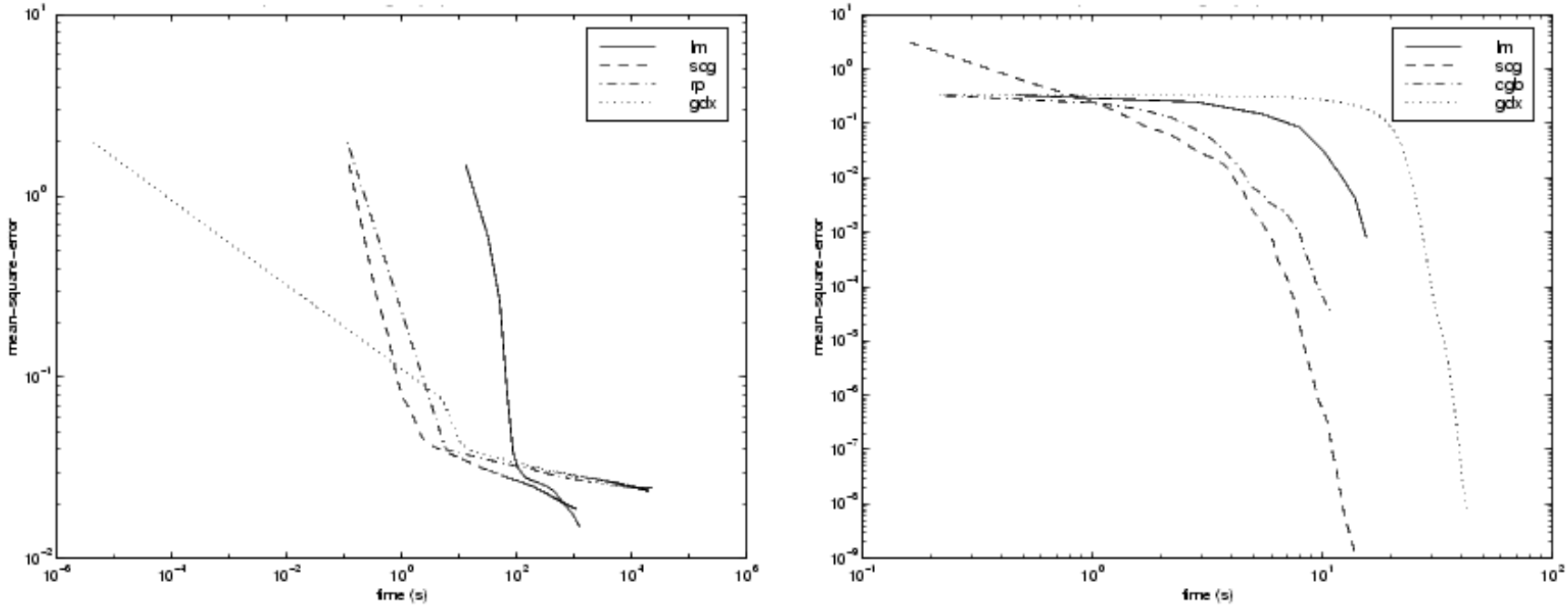
The Broyden-Fletcher-Goldfarb-Shanno procedure is the most well-known quasi-Newton method of $\mathbf{H}^{-1}$ approximation:

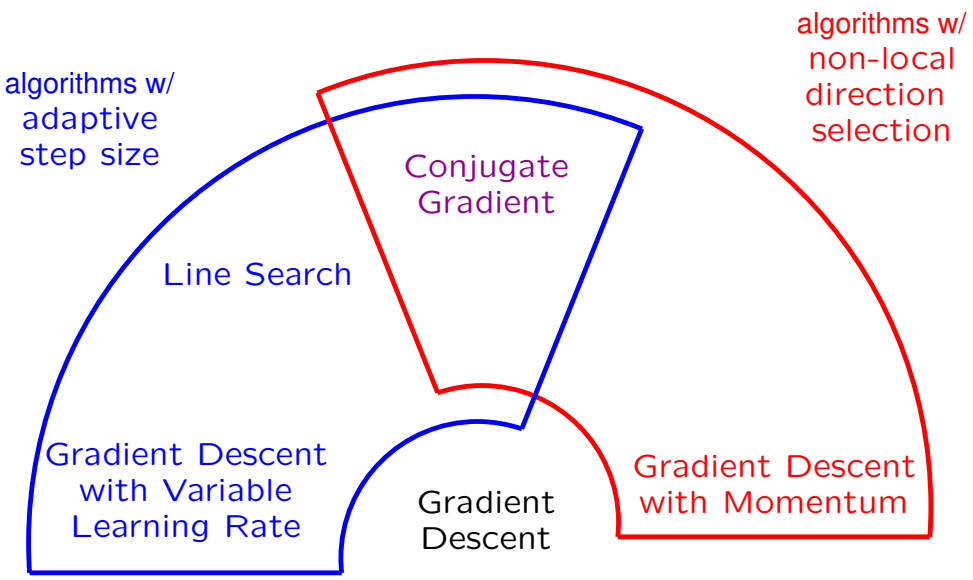
$$\begin{aligned}
\mathbf{p} &= \mathbf{w}^{(\tau+1)} - \mathbf{w}^{(\tau)} \\
\mathbf{v} &= \nabla E|_{\mathbf{w}^{(\tau+1)}} - \nabla E|_{\mathbf{w}^{(\tau)}} \\
\mathbf{u} &= \frac{\mathbf{p}}{\mathbf{p}^T\mathbf{v}} - \frac{\mathbf{G}^{(\tau)}\mathbf{v}}{\mathbf{v}^T\mathbf{G}^{(\tau)}\mathbf{v}}
\end{aligned}$$

$$\mathbf{G}^{(\tau+1)} = \mathbf{G}^{(\tau)} + \frac{\mathbf{p}\mathbf{p}^T}{\mathbf{p}^T\mathbf{v}} - \frac{\left(\mathbf{G}^{(\tau)}\mathbf{v}\right)\mathbf{v}^T\mathbf{G}^{(\tau)}}{\mathbf{v}^T\mathbf{G}^{(\tau)}\mathbf{v}} + \left(\mathbf{v}^T\mathbf{G}^{(\tau)}\mathbf{v}\right)\mathbf{u}\mathbf{u}^T$$

Technique Performance Comparison

Taken from `www.mathworks.com` Neural Networks Toolbox User Guide.



gdx = Variable Rate Gradient Descent, cgb = Conjugate Gradient Search with restarts, scg = Scaled Conjugate Gradient Search, rp = Resilient Backprop, lm = Levenberg-Marquart

What We've Covered