Advanced Training Techniques for Neural Networks

Carlos Valle

Departamento de Informática
Universidad Técnica Federico Santa María

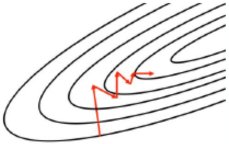
cvalle@inf.utfsm.cl

October 19, 2018

Overview

Gradient-Based Optimization

- The key idea of Backpropagation is iteratively modify the weights of the neuronal following the direction indicated by the gradient of the function of chosen error (for the task in question).
- This optimization strategy is called gradient descent (GD).
- Most deep learning algorithms involve optimization of some sort.
- η is the *learning rate*.



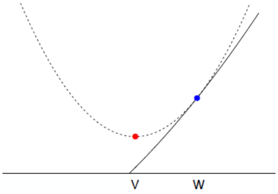
Problem $\min_{\boldsymbol{w}} E(\boldsymbol{w})$

Algorithm

```
1 for $t = 1, \dots, T$ do
2     $\boldsymbol{w}^{(t)} \leftarrow \boldsymbol{w}^{(t-1)} - \eta \frac{\delta E}{\delta \boldsymbol{w}}$;
3 end
```

Gradient-Based Optimization

- Locally, the gradient indicates the direction of greatest increase / decrease in objective function.
- Recall $E(v) \approx E(w) + \nabla_W E^T(v-w)$
- $\nabla_W E^T(v-w) < 0 \Rightarrow E(v) < E(w)$
- $v - w = -\nabla_W E^T \Rightarrow (v-w) = ||\nabla_W E^T(v-w)||^2$

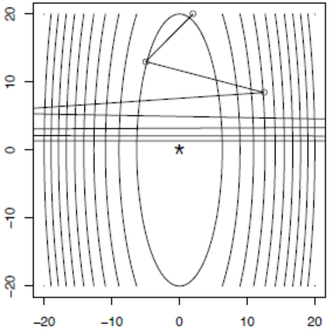


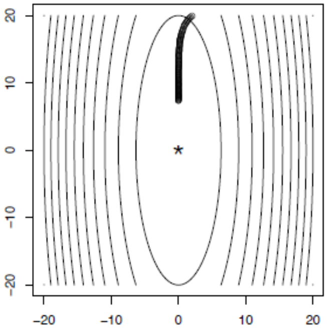
Important problems

- How to choose of the learning rate η?.
- Convergence (if it occurs) is at a stationary point $(f'(\mathbf{x}) = 0)$.

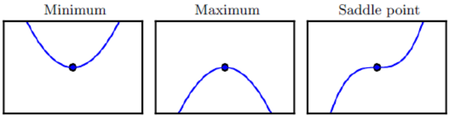
Learning Rate

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \nabla_{\mathbf{w}} E$$





Stationary Points

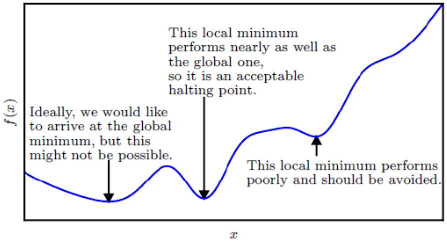


- When the Hessian is positive definite (all its eigenvalues are positive), the point is a local minimum.
- When the Hessian is negative definite (all its eigenvalues are negative), the point is a local maximum.
- When at least one eigenvalue is positive and at least one eigenvalue is negative, we know that this point is a local maximum on one cross section of E but a local minimum on another cross section.
- Global minimum:

$$\mathbf{w}^* = \text{argmin}_w E(\mathbf{w}) \Rightarrow E(\mathbf{w}^*) \leq E(\mathbf{w}) \forall \mathbf{w}$$

Convex versus non-convex

- When E is convex, the gradient always points in the direction of the optimum.
- In the context of deep learning, E is usually a non convex function.
- Thus, it may have many local minima that are not optimal, and many saddle points surrounded by very flat regions.
- In general, GD will converge (if indeed it does) to stationary points.

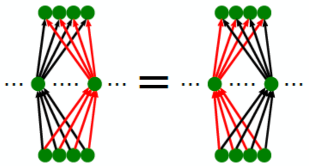


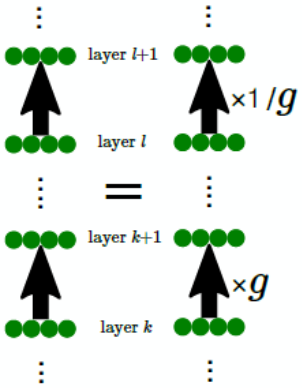
Why E is usually a non-convex function in deep learning?

- The presence of a large number of critical points in a neural network standard is because the model is invariant to a large number of "Transformations" of its parameters and architecture
- It is said that the model is non-identifiable
- See "Skip Connections as Effective Symmetry-Breaking" Emin Orhan, 2018.

Why E is usually a non-convex function in deep learning?

- For example,





Optimization versus Learning

- In learning, it is usually "enough to find a good local minimum".
- The objective function is usually an approximation to the real risk and therefore it does not always make sense to optimize it intensively.
- In fact, in large networks, the global optimum using the train examples often leads to overfitting.

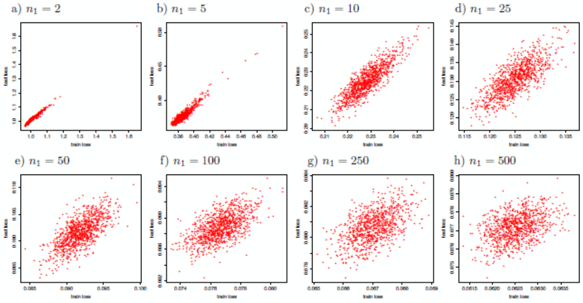
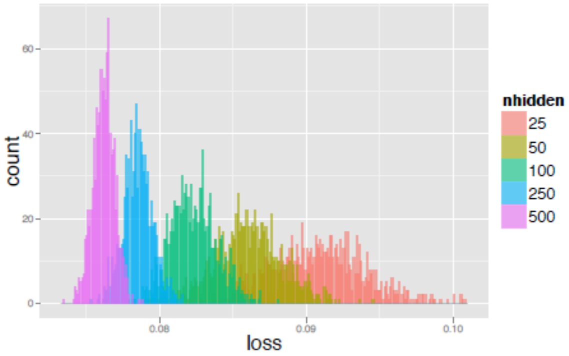


Figure 7: Test loss versus train loss for networks with different number of hidden units n_1.

Optimization versus Learning

- "The Loss Surfaces of Multilayer Networks", Choromanska et al. 2015.



Variants of BP

- Batch Gradient Descent (vanilla version)
- Stochastic Gradient Descent
- Mini-batch Gradient Descent

Batch gradient Descent

- Recall that the empirical error is the error computed over the train set:

$$R_{emp} = \min_{W} \frac{1}{M} \sum_{m=1}^{M} \ell(f(\mathbf{x}_m), y_m)$$

- If we use directly with this objective function, we obtain the classic version of GD.

Algorithm 1 Batch gradient Descent algorithm

```
1: Initialize $W^{(0)}$
2: for $t = 1$ to $T$ do
3:     $\nabla(E(W^{(t)}) = \frac{1}{M}\sum_{m=1}^{M} \nabla\ell(f(\mathbf{x}_m), y_m)$
4:     $W^{(t+1)} \longleftarrow W^{(t)} - \eta\nabla(E(W^{(t)})$
5: end for
```

Batch gradient Descent issues

- Commonly, batch gradient descent is implemented in such a way that it requires the entire training dataset in memory and available to the algorithm.
- Model updates, and in turn training speed, may become very slow for large datasets.

Stochastic gradient descent (SGD)

- Idea: train with 1 example at a time (original version of the BP).
- We do not work with the exact gradient but with an approximation stochastic with the aim of moving quickly towards a best solution.

Algorithm 2 SGD algorithm

```
1: Initialize $W^{(0)}$
2: **for** $t = 1$ to $T$ **do**
3:     Choose one example $(\mathbf{x}_m, y_m)$
4:     $\nabla(E(W^{(t)}) \approx \nabla\ell(f(\mathbf{x}_m), y_m)$
5:     $W^{(t+1)} \longleftarrow W^{(t)} - \eta\nabla(E(W^{(t)})$
6: **end for**
```

Stochastic gradient descent (SGD) (2)

- Each iteration is approximate (less accurate estimate of where the target descends) but at the same time of GD batch we can do n updates.
- For a fixed value of η, stochastic gradient descent may not converge to the optimum.
- Recall the real objective is to minimize the risk.
- Thus, working with a perfect estimate of the gradient is not necessary because this is an approximation of the true objective.

Online (Cyclic) gradient descent

- We orderly iterated the train set.
- In practice SGD performs better than Online gradient descent.

Algorithm 3 Online GD algorithm

```
1: Initialize $W^{(0)}$
2: for $t = 1$ to $T$ do
3:     for $m = 1$ to $M$ do
4:         $\nabla(E(W^{(t)}) \approx \nabla\ell(f(\mathbf{x}_m), y_m)$
5:         $W^{(t+1)} \longleftarrow W^{(t)} - \eta\nabla(E(W^{(t)})$
6:     end for
7: end for
```

Mini Batch gradient descent

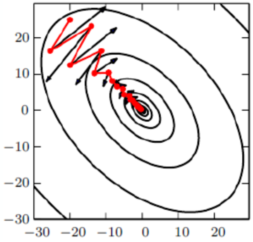
- Approximate the gradient using a subset (small) of examples with the aim of improving the approximation of the gradient.
- Currently, the most popular technique.

Algorithm 4 Mini Batch GD algorithm

```
1: Initialize $W^{(0)}$
2: for $t = 1$ to $T$ do
3:     Randomly choose $n$ examples
4:     $\nabla(E(W^{(t)}) \approx \frac{1}{n}\sum_j \nabla\ell(f(\mathbf{x}_j), y_j)$
5:     $W^{(t+1)} \longleftarrow W^{(t)} - \eta\nabla(E(W^{(t)})$
6: end for
```

Momentum

- The method of momentum is designed to accelerate learning, especially in the face of high curvature, small but consistent gradients, or noisy gradients.
- Idea: keep an estimate of the recent address that the algorithm and modify this address only partially with the current gradient information.

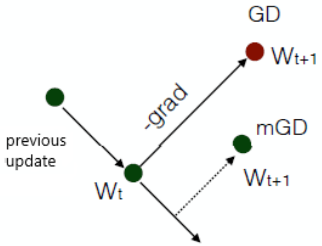


Momentum

- Formally:

$$\begin{aligned} g_t &\longleftarrow \nabla(E(W^{(t)}) \\ v_{t+1} &\longleftarrow \gamma v_t + \eta g_t \\ w_{t+1} &\longleftarrow \gamma w_t - v_{t+1} \end{aligned}$$

- v_{t+1} is called the speed
- Momentum parameter (gamma) usually is considerably larger than the learning rate (e.g. 0.9).



What is momentum doing?

- Improve the estimation of the gradient used in the iteration t, using a moving average over the previous gradients.

$$\begin{aligned}
v_0 &= 0 \\
v_1 &= \eta g_1 \\
v_2 &= \eta g_2 + \gamma \eta g_1 \\
v_3 &= \eta g_3 + \gamma \eta g_2 + \gamma \eta^2 g_1 \\
v_t &= \eta \sum_{j=1}^{t} \gamma^{j-t} g_j
\end{aligned}$$

Nesterov Momentum

- The difference between Nesterov momentum and standard momentum is where the gradient is evaluated.
- Nesterov momentum evaluates the gradient after the current velocity is applied.
- Thus we can view Nesterov momentum as attempting to add a correction factor to the standard method of momentum.

$$\begin{aligned} g_t &\longleftarrow \nabla(E(W^{(t)} - \gamma v_t) \\ v_{t+1} &\longleftarrow \gamma v_t + \eta g_t \\ w_{t+1} &\longleftarrow \gamma w_t - v_{t+1} \end{aligned}$$

Algorithms with Adaptive Learning Rates

- Idea: If we believe that the directions of sensitivity are somewhat axis-aligned
- It can make sense to use a separate learning rate for each parameter, and automatically adapt these learning rates throughout the course of learning.

Progressive decay

- As we discussed previously, for a fixed value of η, stochastic gradient descent may not converge to the optimum.
- progressive decay could be used
- Initialize $\eta = \eta_0$
- η_d is the learning rate decay. At each iteration $s : \eta^(s) = \eta_0/(1 + s\eta_d)$

Adagrad

- Adagrad individually adapts the learning rates of all model parameters by scaling them inversely proportional to the square root of the sum of all of their historical squared values

Algorithm 5 AdaGrad algorithm

```
1: Initialize $W^{(0)}$
2: $r \longleftarrow 0$
3: while stopping criterion not met do
4:     Randomly choose $n$ examples
5:     $g \longleftarrow \frac{1}{n}\sum_j \nabla\ell(f(\mathbf{x}_j), y_j)$
6:     $r \longleftarrow r + g \odot g$
7:     $W^{(t+1)} \longleftarrow W^{(t)} - \eta\frac{1}{\sqrt{r}+\epsilon} \odot g$
8: end while
```

Adagrad

- The idea of normalizing the geometry of the gradients results very useful in the context of neural networks because, in Generally, layers can have gradients on very high scales different
- However, Adagrad is proposed for convex functions.
- In practice, the accumulation of squared gradients from the beginning of training can result in a premature and excessive decrease in the effective learning rate.

Rmsprop

- Estimate the covariance matrix using a moving average.
- Adagrad: $r_t \longleftarrow \sum_k g_k \odot g_k = \frac{t-1}{t} r_{t-1} + \frac{1}{t} g_t \odot g_t$
- Rmsprop: $r_t \longleftarrow \gamma r_{t-1} + (1-\gamma) g_t \odot g_t$

Algorithm 6 Rmsprop algorithm

```
1: Initialize $W^{(0)}$
2: $r \longleftarrow 0$
3: while stopping criterion not met do
4:     Randomly choose $n$ examples
5:     $g \longleftarrow \frac{1}{n} \sum_j \nabla \ell(f(\mathbf{x}_j), y_j)$
6:     $r \longleftarrow \gamma r + (1-\gamma) g \odot g$
7:     $W^{(t+1)} \longleftarrow W^{(t)} - \eta \frac{1}{\sqrt{r}+\epsilon} \odot g$
8: end while
```

Rmsprop

- γ parameter controls the length scale of the moving average.
- Algorithm is quite robust to the choice of $\gamma = 0.9$ and $\eta = 0.01$

Adam

- This algorithm tries to correct the bias of the estimates of the first and second moment of the gradient inherent to RMS Prop (with momentum)
- See "Adam: A Method for Stochastic Optimization", Kingma et al. (2015).

Adam (2)

Algorithm 7 Adam algorithm

```
1: Initialize $W^{(0)}$
2: $r \longleftarrow 0$
3: $t \longleftarrow 0$
4: while stopping criterion not met do
5:     Randomly choose $n$ examples
6:     $g \longleftarrow \frac{1}{n}\sum_j \nabla \ell(f(\mathbf{x}_j), y_j)$
7:     $v \longleftarrow \gamma v + (1-\beta)g$
8:     $r \longleftarrow \gamma r + (1-\gamma)g \odot g$
9:     $\hat{v} \longleftarrow \frac{v}{1-\beta^t}$
10:    $\hat{r} \longleftarrow \frac{v}{1-\gamma^t}$
11:    $W^{(t+1)} \longleftarrow W^{(t)} - \eta \frac{1}{\sqrt{\hat{r}+\epsilon}} \odot \hat{s}$
12: end while
```

How to initialize Ws?

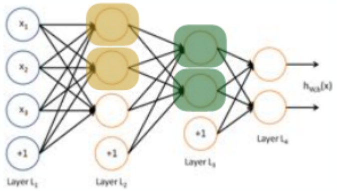
- We need to give initial values to the parameters of the network
- The starting point for GD strongly determines the stationary point to which it converges and influences also in the speed and stability of the convergence.
- The methods currently used are characterized by being simple, heuristic and generally motivated by some property that is desired Keep at least the first iterations of training.

Strategy

- Choose the weights of each neuron independently, sampling a very entropic probability distribution, with mean null and reasonably high variance.
- Goal: Encourage what is called "symmetry" breaking", that is, that two units capable of the same function should be initialized to a function different.

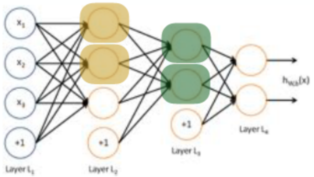
Simetry Breaking

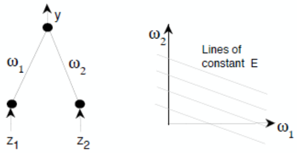
- We want to discover relevant attributes.
- If the two units in yellow are initialized with the same weights, will transmit exactly the same attribute to the next layer for each entry x.
- This redundancy is not necessary if the units of the next layer can "read" the attribute from either unit.
- In addition, if the units in green are initialized with the same weights, the Yellow units will receive the same adjustments during training, perpetuating redundancy.



Simetry Breaking (2)

- If the two units in yellow transmit the same attribute to the next layer for each entry x, there will be two input attributes perfectly linearly correlated.
- This scenario creates complete sub-spaces of the parameter space with the same value of the objective function.





$$w_1 z_1 + w_2 z_2 = w_1 z_1 + w_2 z_1 = (w_1 + w_2) z_1$$

Maximum entropy probability distribution

- From all distributions with finite support (that is, not null in a finite interval), the most entropic is the uniform distribution.
- From all distributions with support in $\mathbb{R}$, with finite average and finite variance, the most entropic distribution is the normal distribution.

Basic Strategy

- To choose weight with an entropic distribution centered at cero.
- In order to minimize to avoid that the input attributes be perfectly linearly correlated from beginning.

Two proposals

- LeCun et al. Efficient Backprop,1998:

$$w_{ij}\ U\left(\frac{-1}{\sqrt{m}},\frac{1}{\sqrt{m}}\right)$$

- where m is the dimensionality of the input to the neuron.
- Glorot and Bengio. Understanding the difficulty of training deep feedforward neural networks. Aistats. Vol. 9. 2010. 2010:

$$w_{ij}\ U\left(-\sqrt{\frac{6}{m+n}},\sqrt{\frac{6}{m+n}}\right)$$

- where n is the output dimensionality of the neuron.

Any questions?