Machine learning for signal processing [5LSL0]

Ruud van Sloun, Rik Vullings

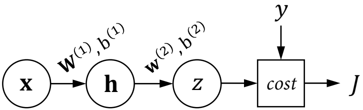
May 2019

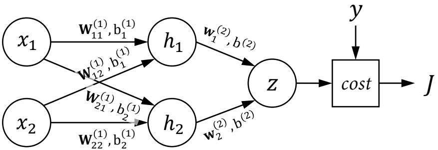
Technische Universiteit Eindhoven University of Technology

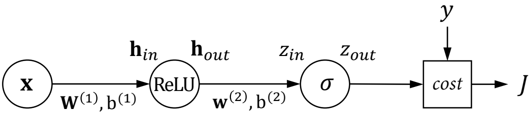
Where innovation starts

Recap nonlinear classification models

The full binary classification model becomes:







binary cross-entropy:

$$J = \frac{1}{m}\sum_{i=0}^{m-1} y_i \log z_{out} + (1 - y_i)\log(1 - z_{out})$$

Parameter optimization

Finding optimal parameter values given the cost

$\Rightarrow$ Gradient-based learning (e.g. Gradient-Descent algorithm):

- $\mathbf{W}_{n+1}^{(1)} = \mathbf{W}_{n}^{(1)} - \mu \partial_{\mathbf{W}^{(1)}} J(\theta_n)$
- $\mathbf{w}_{n+1}^{(2)} = \mathbf{w}_{n}^{(2)} - \mu \partial_{\mathbf{w}^{(2)}} J(\theta_n)$
- $\mathbf{b}_{n+1}^{(1)} = \mathbf{b}_{n}^{(1)} - \mu \partial_{\mathbf{b}^{(1)}} J(\theta_n)$
- $b_{n+1}^{(2)} = b_{n}^{(2)} - \mu \partial_{b^{(2)}} J(\theta)$

With learning rate μ.

Parameter optimization

Finding optimal parameter values given the cost

$\Rightarrow$ Gradient-based learning (e.g. Gradient-Descent algorithm):

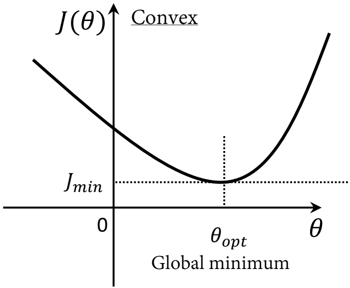
- Linear models; Convex $\rightarrow$ guaranteed global convergence.
- Nonlinear models; Often non-convex $\rightarrow$ no global convergence guarantees.

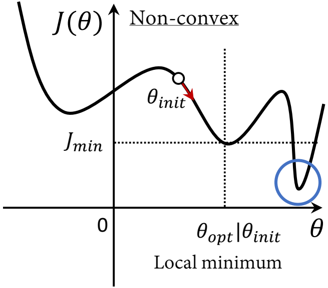
Parameter optimization

Finding optimal parameter values given the cost

⇒ Gradient-based learning (e.g. Gradient-Descent algorithm):

- Linear models; Convex → guaranteed global convergence.
- Nonlinear models; Often non-convex → no global convergence guarantees.



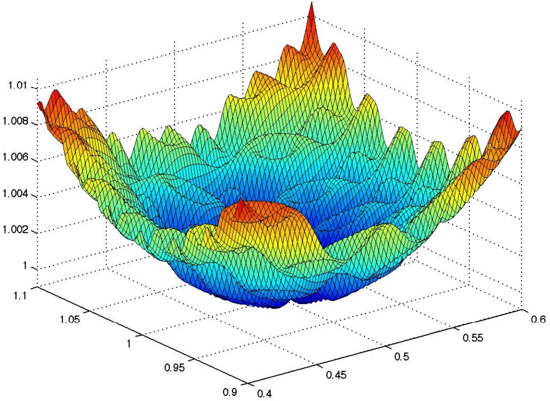


Optimization

Challenge: learning the optimal coefficients of a highly flexible (high-capacity) model that minimize the generalization error.

Optimization

Challenge: learning the optimal coefficients of a highly flexible (high-capacity) model that minimize the generalization error.



(Martin Takac, 2016)

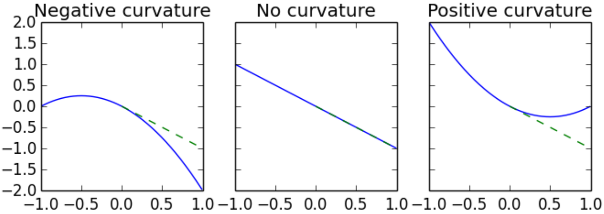
Optimization

Challenge: learning the optimal coefficients of a highly flexible (high-capacity) model that minimize the generalization error.

Optimization problems in machine learning:

- Plateaus, saddle Points, flat regions
- "Cliffs"
- Vanishing and exploding gradients
- Inexact gradients
- Local vs global structure

Beyond the gradient: curvature (Hessian)

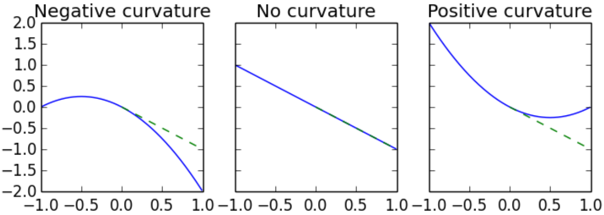


(Goodfellow, 2015)

Taylor series expansion of cost function:

$$J(\theta) = J(\theta_0) + (\theta - \theta_0)^T \mathbf{g} + \frac{1}{2}(\theta - \theta_0)^T \mathbf{H}(\theta - \theta_0) \qquad (1)$$

Beyond the gradient: curvature (Hessian)



(Goodfellow, 2015)

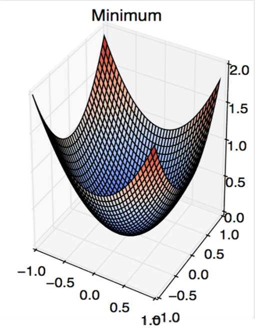
Taylor series expansion of cost function:

$$J(\theta) = J(\theta_0) + (\theta - \theta_0)^T \mathbf{g} + \frac{1}{2}(\theta - \theta_0)^T \mathbf{H}(\theta - \theta_0) \tag{1}$$

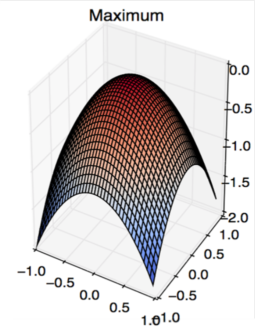
Second order Taylor series prediction of gradient step:

$$J(\theta - \mu \mathbf{g}) \approx J(\theta) - \mu \mathbf{g}^T \mathbf{g} + \frac{1}{2}\mu^2 \mathbf{g}^T \mathbf{H} \mathbf{g} \tag{2}$$

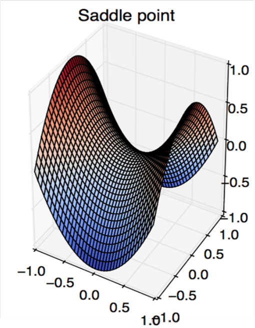
Critical points in cost functions



All positive eigenvalues



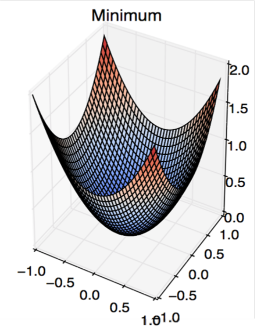
All negative eigenvalues



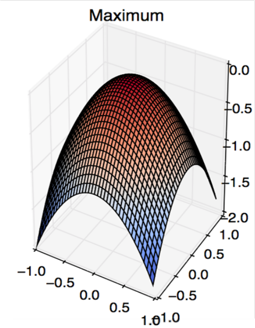
Some positive and some negative eigenvalues

(Goodfellow, 2015)

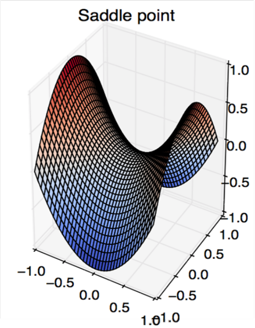
Critical points in cost functions



All positive eigenvalues



All negative eigenvalues

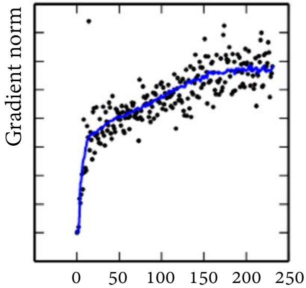


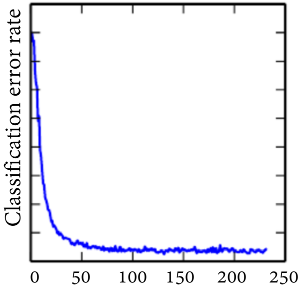
Some positive and some negative eigenvalues

(Goodfellow, 2015)

Critical points: local minima, saddle points, very small gradients..

"Stuck" in a critical point?



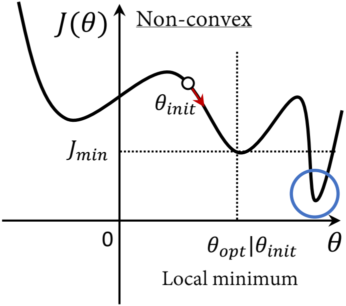


(Goodfellow, 2016)

⇒ Monitoring the norm of the gradient

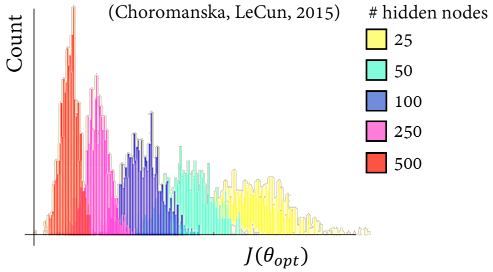
Converging to a “high cost” local minimum?

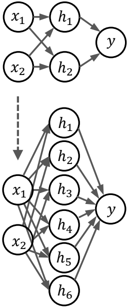
How likely is it that we end up converging to a local minimum with a large cost value $J(\theta)$?



Converging to a “high cost” local minimum?

How likely is it that we end up converging to a local minimum with a large cost value $J(\theta)$?

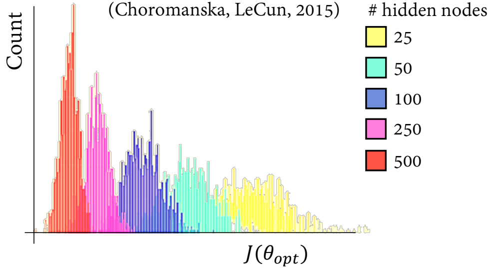


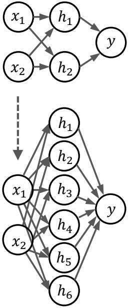


Remarkably, this is less likely to happen for big models with large latent dimensions (hidden nodes)!

Converging to a "high cost" local minimum?

How likely is it that we end up converging to a local minimum with a large cost value $J(\theta)$?



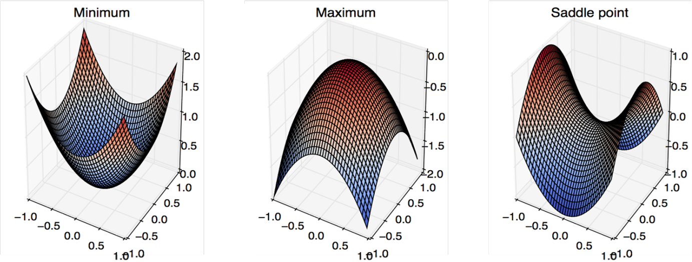


Remarkably, this is less likely to happen for big models with large latent dimensions (hidden nodes)!

$\Rightarrow$ Why?

Local minima vs saddle points?

How many local minima do we expect to encounter in our cost functions? Remember:



All positive eigenvalues

All negative eigenvalues

Some positive and some negative eigenvalues

(Goodfellow, 2015)

For large models: probability of having some positive and some negative eigenvalues in the local Hessian (rather than all positive or all negative) is very large. $\Rightarrow$ Many saddle points!

Local minima vs saddle points?

For many classes of random functions:

- Low dimensional space $\Rightarrow$ local minima common.
- High dimensional space $\Rightarrow$ local minima rare, and saddle points common.

Local minima vs saddle points?

For many classes of random functions:

- Low dimensional space $\Rightarrow$ local minima common.
- High dimensional space $\Rightarrow$ local minima rare, and saddle points common.
- The expected ratio of number of number of saddle points to local minima grows exponentially with n.
- Eigenvalues of the Hessian are more likely to be positive for regions with low cost; as such:
  - local minima are likely to have low cost
  - critical points with high costs are likely saddle points
  - critical points with very high costs are likely local maxima

$\Rightarrow$ There is hope that we can successfully train large models (e.g. deep neural networks)

Stochastic gradient descent

(deterministic) Gradient estimation on entire dataset is very expensive and requires evaluating the model on every example.

Stochastic gradient descent

(deterministic) Gradient estimation on entire dataset is very expensive and requires evaluating the model on every example.

Stochastic gradient descent

(deterministic) Gradient estimation on entire dataset is very expensive and requires evaluating the model on every example.

Alternative (unbiased) estimator for the exact gradient of the generalization error:
$\Rightarrow$ Stochastic gradient descent (SGD)

- Use a new "minibatch" of the full data to compute the gradient for each iteration

Tradeoff:

- Large batches $\Rightarrow$ more accurate estimate of the gradient
- Small batches $\Rightarrow$ regularizing effect (noisy gradient, more follows)

Stochastic gradient descent

At iteration k,

1. Gradient estimation:

$$\nabla_{\theta_k} = \frac{1}{m} \nabla_\theta \sum_i J(f(\mathbf{x}^{(i)}; \theta_k), \mathbf{y}^{(i)})$$

2. Parameter update rule:

$$\theta_{k+1} = \theta_k - \mu_k \nabla_{\theta_k}$$

Stochastic gradient descent with momentum

At iteration k,

1. Gradient estimation:

$$\nabla_{\theta_k} = \underbrace{\rho \nabla_{\theta_{k-1}}} + \tfrac{1}{m} \nabla_\theta \textstyle\sum_i J(f(\mathbf{x}^{(i)}; \theta_k), \mathbf{y}^{(i)})$$

2. Parameter update rule:

$$\theta_{k+1} = \theta_k - \mu_k \nabla_{\theta_k}$$

$\Rightarrow$ Exponentially decaying moving average of past gradients.
$\Rightarrow$ Accelerate learning for high curvatures, small (but consistent) gradients and noisy gradients.

AdaGrad: (naive) adaptive learning rate

At iteration k,

1. Gradient estimation:

$$\nabla_{\theta_k} = \frac{1}{m}\nabla_\theta \sum_i J(f(\mathbf{x}^{(i)};\theta_k),\mathbf{y}^{(i)})$$

2a. Accumulate squared gradient:

$$\mathbf{r}_k = \mathbf{r}_{k-1} + \nabla_{\theta_k} \odot \nabla_{\theta_k}$$

2b. Parameter update rule:

$$\theta_{k+1} = \theta_k - \frac{\mu_k}{\delta+\sqrt{\mathbf{r}_k}} \odot \nabla_{\theta_k} \text{ (element wise operations)}$$

$\Rightarrow$ Learning rate for individual parameters scaled based on ℓ_2 norm of previous k gradients w.r.t. those parameters (no decay / forgetting factor).

RMSprop: improved adaptive learning rate

At iteration k,

1. Gradient estimation:

$$\nabla_{\theta_k} = \frac{1}{m}\nabla_\theta \sum_i J(f(\mathbf{x}^{(i)};\theta_k),\mathbf{y}^{(i)})$$

2a. Accumulate squared gradient:

$$\mathbf{r}_k = \rho\mathbf{r}_{k-1} + (1-\rho)\nabla_{\theta_k} \odot \nabla_{\theta_k}$$

2b. Parameter update rule:

$$\theta_{k+1} = \theta_k - \frac{\mu_k}{\delta+\sqrt{\mathbf{r}_k}} \odot \nabla_{\theta_k} \text{ (element wise operations)}$$

$\Rightarrow$ Learning rate for individual parameters scaled based on exponentially weighted moving average of the previous gradients.
$\Rightarrow$ Often also used in conjunction with momentum on the gradients.

Adam: adaptive moments and bias correction

At iteration k,

1. Gradient estimation:

$$\nabla_{\theta_k} = \frac{1}{m}\nabla_\theta \sum_i J(f(\mathbf{x}^{(i)};\theta_k), \mathbf{y}^{(i)})$$

2a. Update (biased) first moment:

$$\mathbf{s}_k = \rho_1 \mathbf{s}_{k-1} + (1-\rho_1)\nabla_{\theta_k}$$

2b. Update (biased) second moment (accumulate squared gradient):

$$\mathbf{r}_k = \rho_2 \mathbf{r}_{k-1} + (1-\rho_2)\nabla_{\theta_k} \odot \nabla_{\theta_k}$$

2c. Bias correction moments:

$\hat{\mathbf{s}}_k = \frac{\mathbf{s}_k}{1-\rho_1^k}$ and $\hat{\mathbf{r}}_k = \frac{\mathbf{r}_k}{1-\rho_2^k}$

3. Parameter update rule:

$\theta_{k+1} = \theta_k - \frac{\mu_k}{\delta+\sqrt{\hat{\mathbf{r}}_k}} \odot \hat{\mathbf{s}}_k$ (element wise operations)

SGD algorithms visualized

SGD algorithms visualized

Vanishing gradients

Can occur when propagating gradients through many (nonlinear) layers. Which activation functions are most sensitive to this?