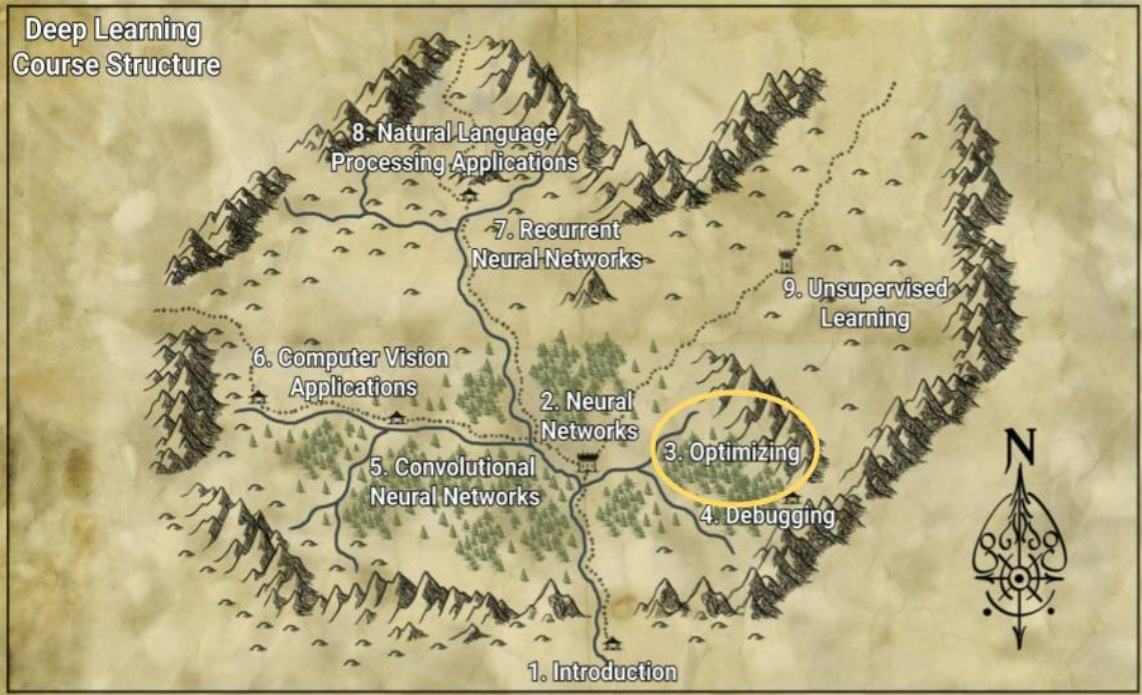
Optimization

Florin Gogianu,
Bitdefender ML Team


Deep Learning
Course Structure
8. Natural Language
Processing Applications
7. Recurrent
Neural Networks
9. Unsupervised
Learning
6. Computer Vision
Applications
2. Neural
Networks
3. Optimizing
5. Convolutional
Neural Networks
4. Debugging
N
1. Introduction

Bibliography

Textbooks:

- Deisenroth, Mathematics for Machine Learning, [Ch. 5 - Vector Calculus]
- Goodfellow, Deep Learning
- Prince, Understanding Deep Learning

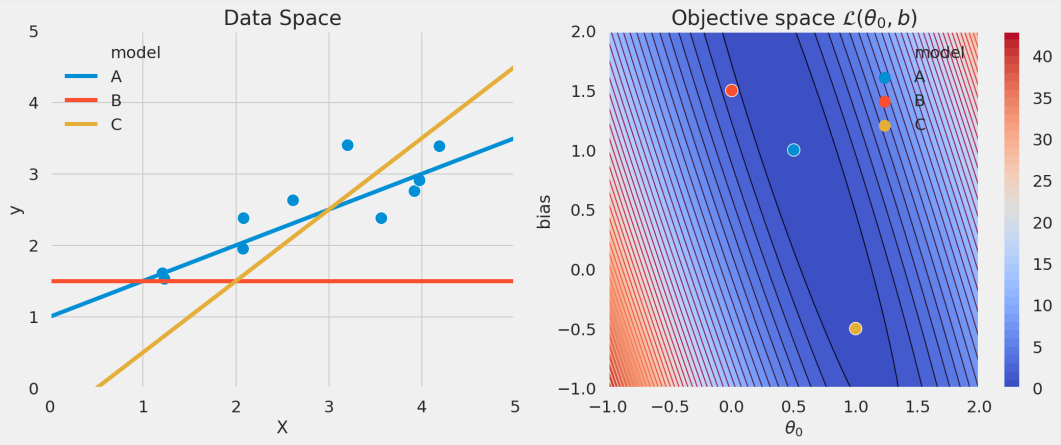
Courses:

- Stanford, CS231n
- NYU Deep Learning

Recap++

Recap. Linear Models

$$\mathbf{y} = \mathbf{W}\mathbf{x} + \mathbf{b}$$



Recap. Linear Models

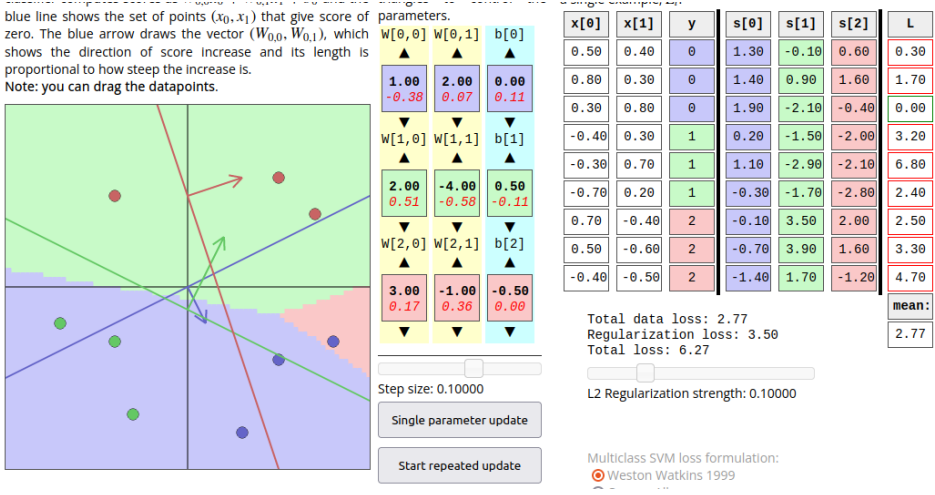
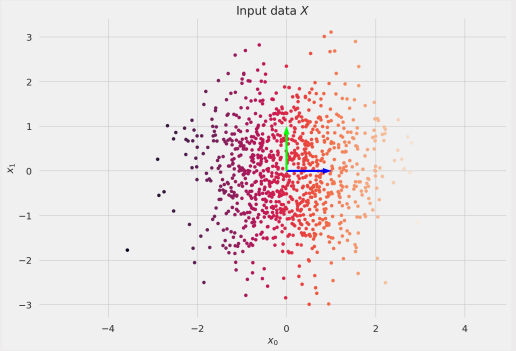
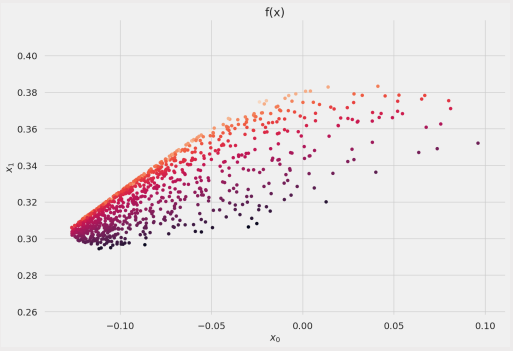


Figure: Linear classification visualization

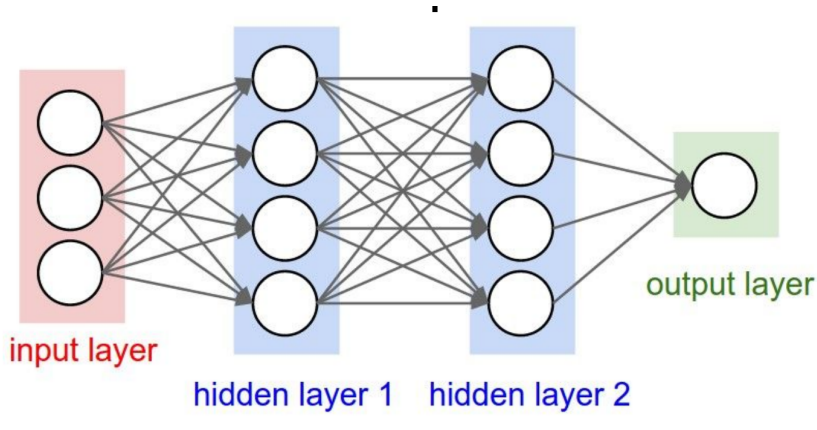
Recap. NN computations

- **y** = **Wx** computes: rotation, scale reflection transformations
- σ(**Wx**) adds non-linear transformations



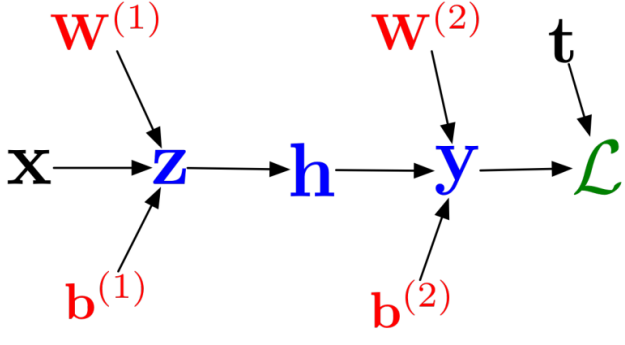


Recap. Fully Connected NN



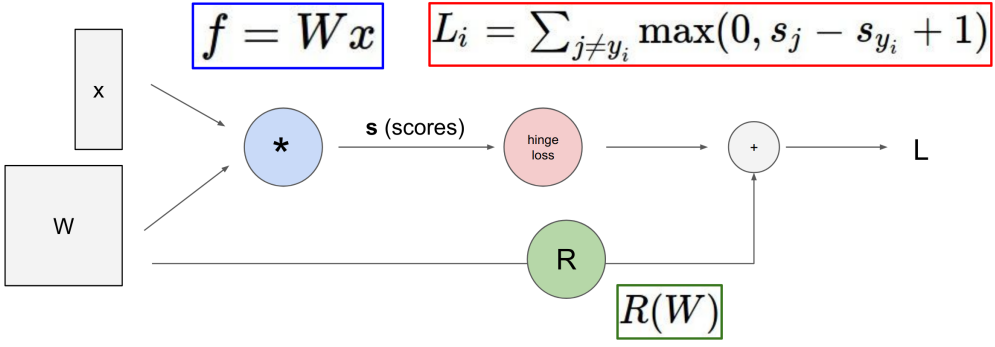
- Linear case: $f(\mathbf{x}) = \mathbf{W}\mathbf{x}$
- Neural net: $f(\mathbf{x}) = \mathbf{W}_3\sigma(\mathbf{W}_2\sigma(\mathbf{W}_1\mathbf{x}))$

Recap. Fully Connected NN



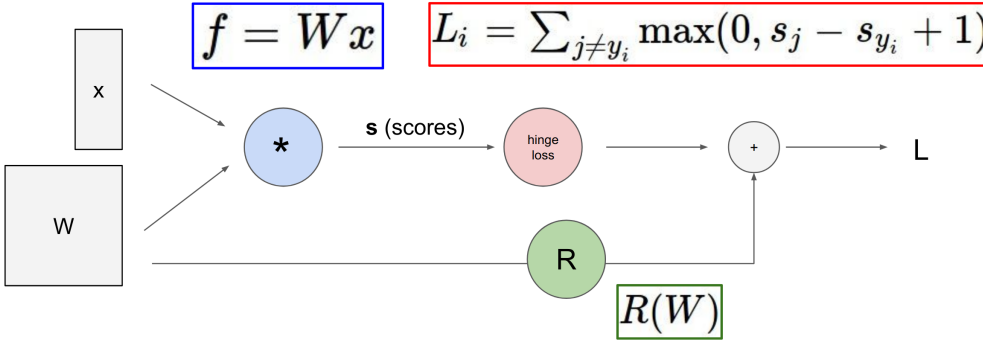
- Neural net: $f(\mathbf{x}) = \mathbf{y} = \mathbf{W}_2\sigma(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2$
- Loss function: $L = \frac{\mathbf{y}\cdot\mathbf{t}}{\|\mathbf{y}\|\|\mathbf{t}\|}$

Computation graphs + automatic differentation



[1] Grosse, Ba, CS421

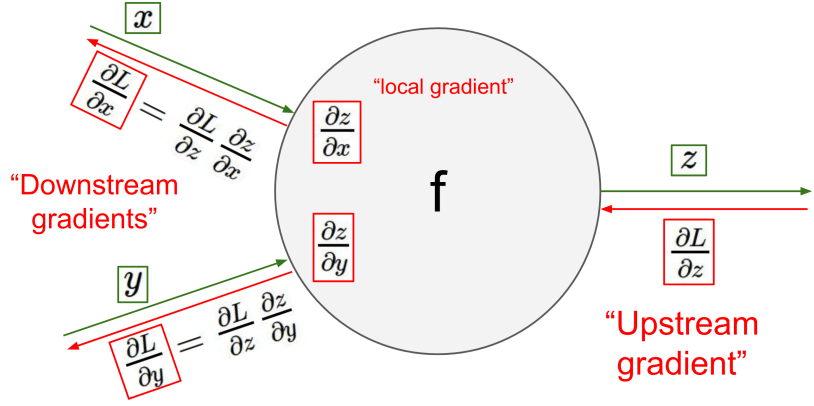
Computation graphs + automatic differentation



Automatic differentiation: takes a program which computes a value, and automatically constructs a procedure for computing derivatives of that value[1].

[1] Grosse, Ba, CS421

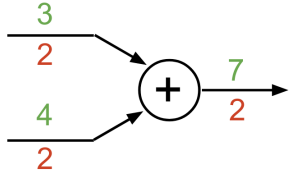
Backprop



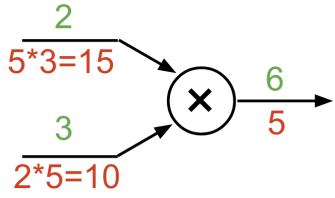
[1] from CS231n, lecture 4, 2019

Recap. Backprop patterns

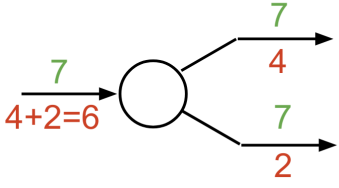
add gate: gradient distributor



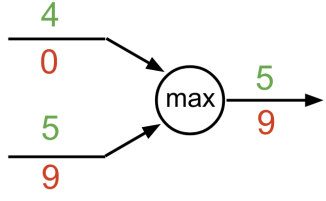
mul gate: “swap multiplier”



copy gate: gradient adder



max gate: gradient router

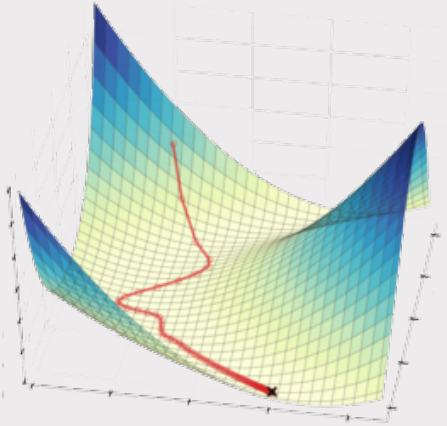


Optimization concepts

Gradient Descent

1. Measure the error of our model using $L(D, \boldsymbol{\theta})$,
2. Compute the gradients w.r.t. $\boldsymbol{\theta}$
3. Find incremental solutions that better explain the data using:

$$\boldsymbol{\theta}_{j+1} \leftarrow \boldsymbol{\theta}_j - \eta \nabla_{\boldsymbol{\theta}_j} L$$



B

Gradient Descent

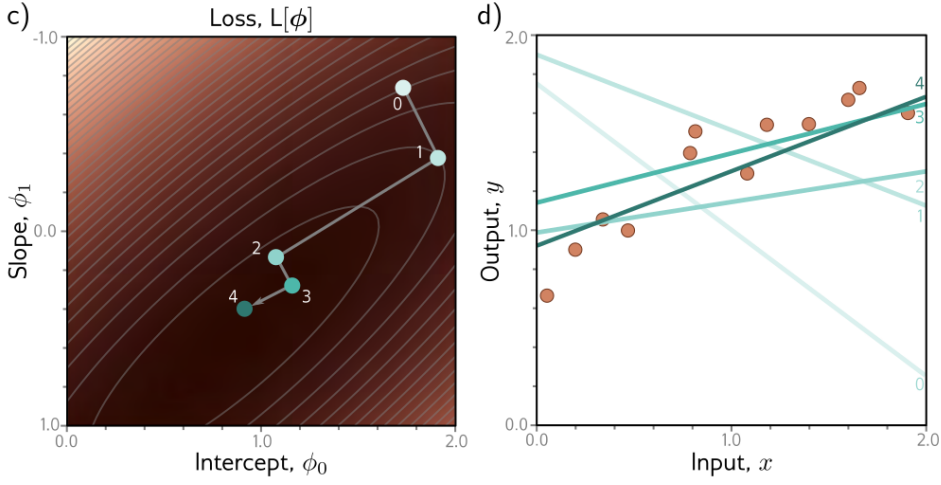


Figure: Gradient descent for the linear model.

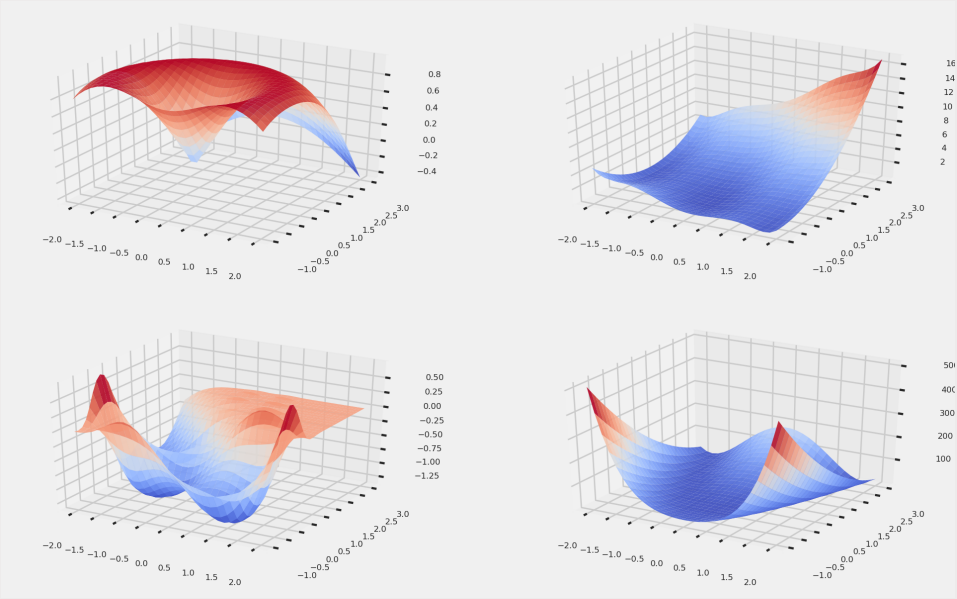
Stochastic Gradient Descent

Consider instead objective functions that are the sum of the losses – $L(\boldsymbol{\theta}) = \sum_{n=1}^{N} L_n(\boldsymbol{\theta})$, resulting in the update:

$$\boldsymbol{\theta}_{j+1} \leftarrow \boldsymbol{\theta}_j - \eta \sum_{n=1}^{N} \nabla_{\boldsymbol{\theta}_j} L_n(\boldsymbol{\theta})$$

- evaluating the sum of gradients can be expensive
- instead we can randomly choose a subset of L_n
- for gradient descent to converge we only require the samples to be unbiased

Optimization landscapes



Taylor Approximation

- Arbitrary non-linear functions are hard to globally analyze.
- Ideally, we would like to work with simple functions: polynomials.
- Solution: get a polynomial that approximates the function in a *neighbourhood* well enough

$$L(\boldsymbol{\theta}) \approx L\left(\boldsymbol{\theta}_0\right) + \nabla L\left(\boldsymbol{\theta}_0\right)^{\mathsf{T}} \left(\boldsymbol{\theta} - \boldsymbol{\theta}_0\right) + \frac{1}{2}\left(\boldsymbol{\theta} - \boldsymbol{\theta}_0\right)^{\mathsf{T}} \mathbf{H}\left(\boldsymbol{\theta}_0\right)\left(\boldsymbol{\theta} - \boldsymbol{\theta}_0\right)$$

Taylor Approximation

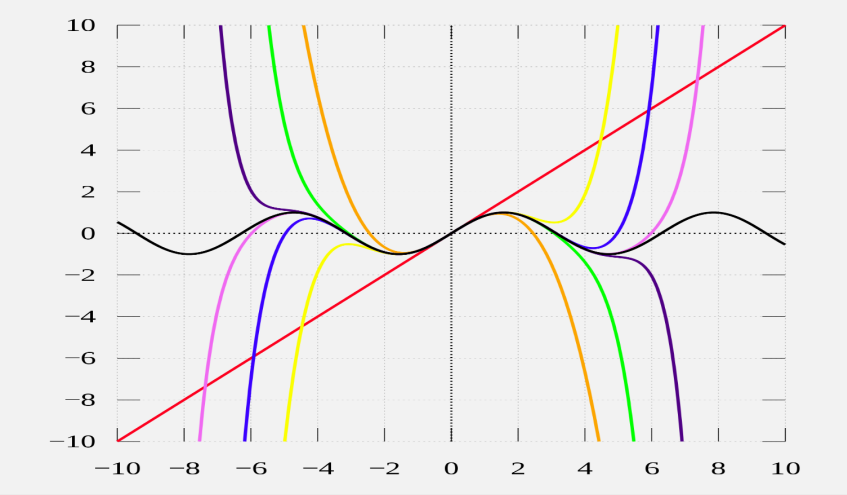
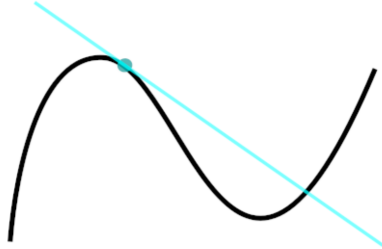
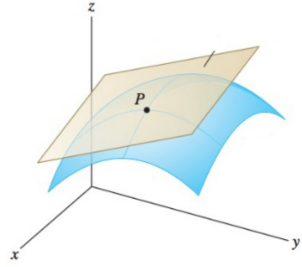


Figure: sin(x) and its Taylor approximations, polynomials of degree 1, 3, 5, 7, 9, 11, 13
By IkamusumeFan, CC BY-SA 3.0.

First-order information



Tangent line [1]

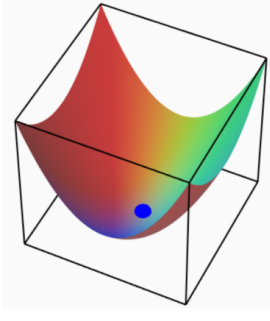


Tangent space [2]

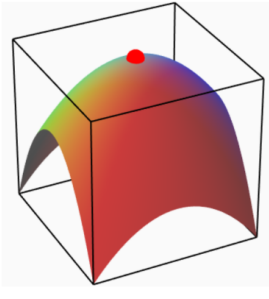
Growth of a function when a) changing θ and b) moving in the direction of $\theta - \theta_0$.

$$L(\boldsymbol{\theta}) \approx L\left(\boldsymbol{\theta}_0\right) + \nabla L\left(\boldsymbol{\theta}_0\right)^{\mathsf{T}}\left(\boldsymbol{\theta} - \boldsymbol{\theta}_0\right)$$

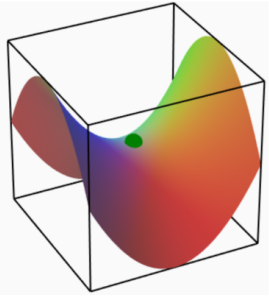
Second-order information



(a) Minima



(b) Maxima

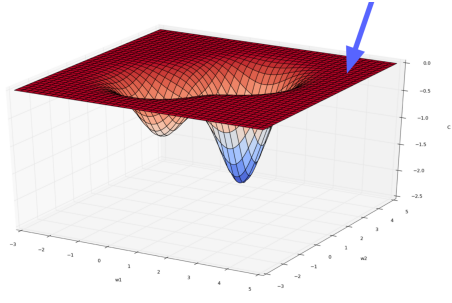


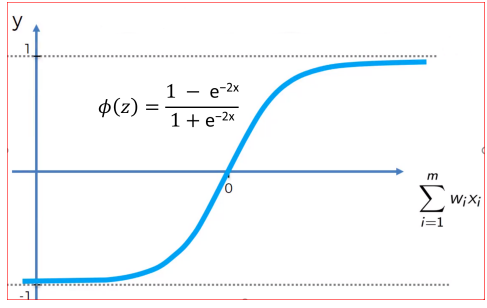
(c) Saddle

$$L(\boldsymbol{\theta}) \approx L(\boldsymbol{\theta}_0) + \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}_0)^{\mathsf{T}} \mathbf{H}(\boldsymbol{\theta}_0)(\boldsymbol{\theta} - \boldsymbol{\theta}_0)$$

How does moving in a direction affects the rate of change.

Patologies. Plateaux



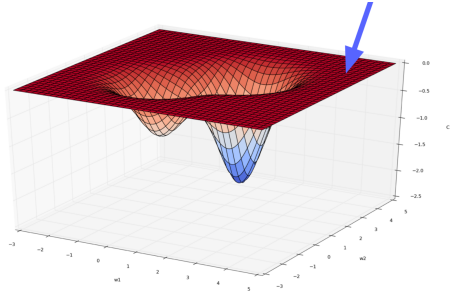


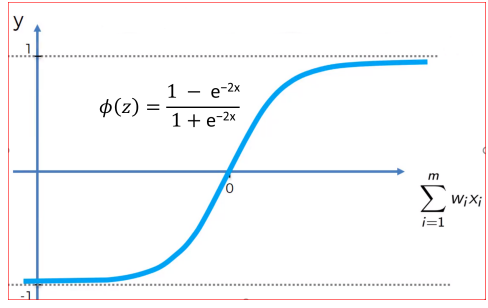
In cases where a neuron is saturated:

$$\delta_{\theta} = \delta_{z}\mathbf{x}$$

$$\delta_{z} = \delta_{h}\frac{\partial \mathbf{h}}{\partial \mathbf{z}} = \delta_{h}\phi'(\mathbf{z})$$

Patologies. Plateaux





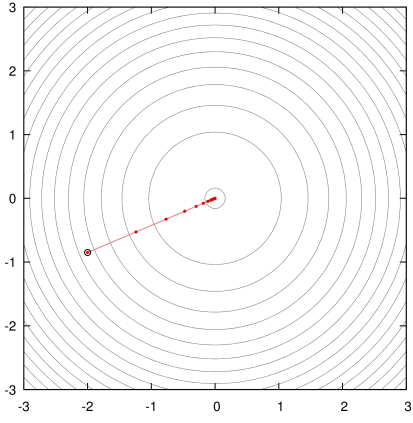
In cases where a neuron is saturated:

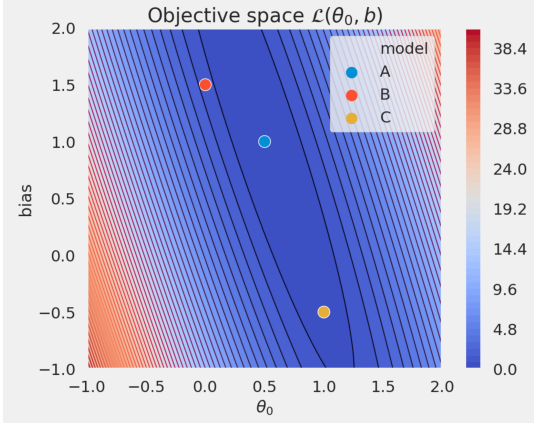
$$\delta_{\theta} = \delta_{z}\mathbf{x}$$

$$\delta_{z} = \delta_{h}\frac{\partial \mathbf{h}}{\partial \mathbf{z}} = \delta_{h}\phi'(\mathbf{z})$$

Use ReLU activations and good initialization schemes.

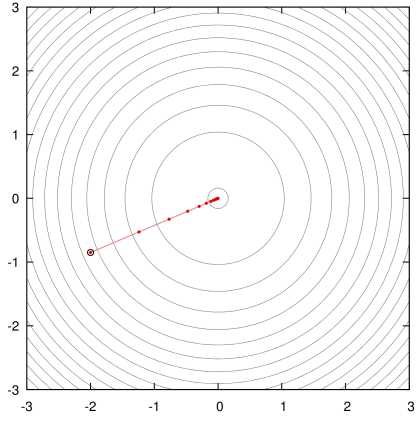
Patologies. High curvature

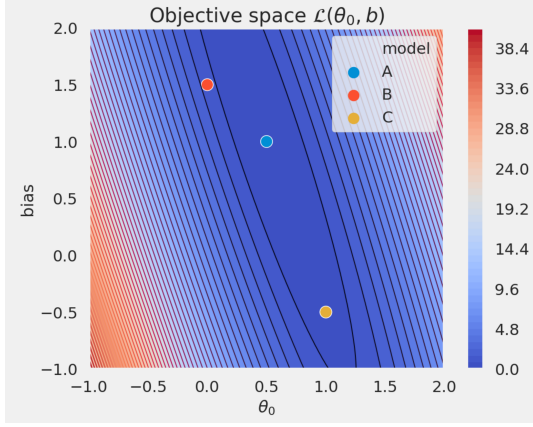




Remember: $\boldsymbol{\theta}_{j+1} \leftarrow \boldsymbol{\theta}_j - \alpha \nabla_{\boldsymbol{\theta}_j} L$. Badly conditioned curvatures arise even in simple cases, when the inputs have slightly different scales or are off-centered.

Patologies. High curvature





Remember: $\boldsymbol{\theta}_{j+1} \leftarrow \boldsymbol{\theta}_j - \alpha \nabla_{\boldsymbol{\theta}_j} L$. Badly conditioned curvatures arise even in simple cases, when the inputs have slightly different scales or are off-centered.

Solutions: i) standardize your input data, ii) normalize every pre-activation.

Demo time!

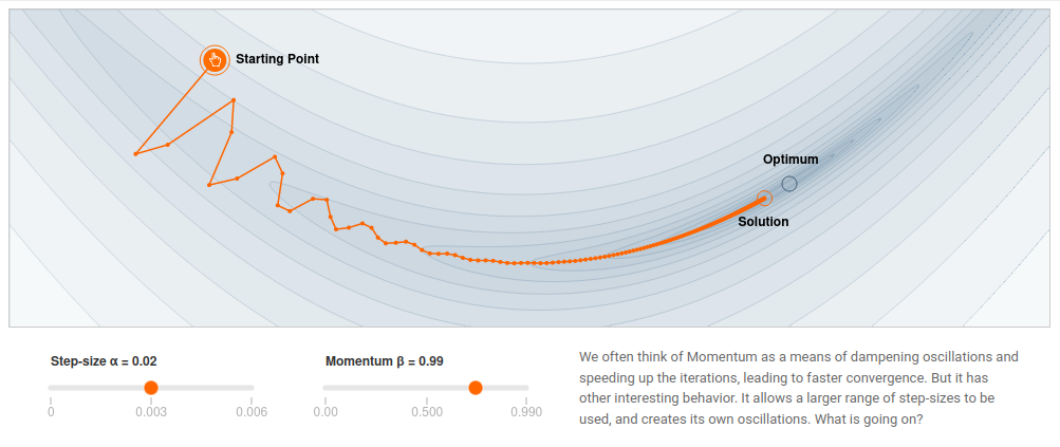


Figure: Example of high curvature inducing optimization problems.

Solution: Normalize pre-activations

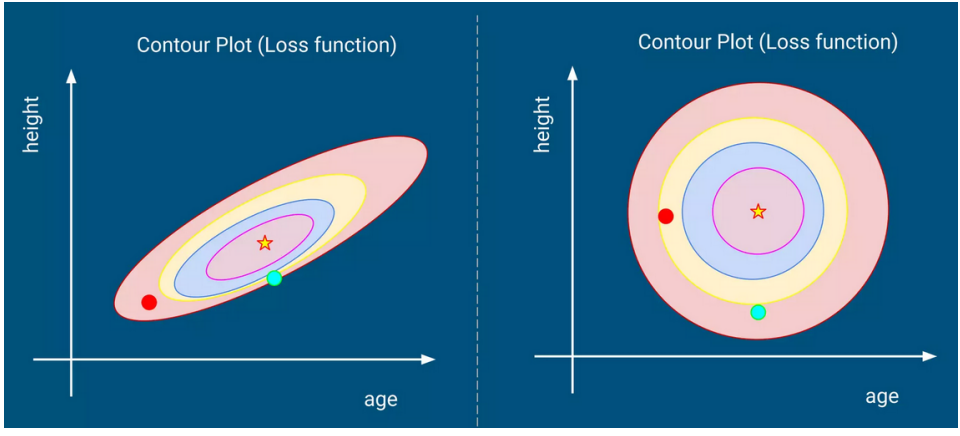


Figure: Feature normalization using BatchNormalization

$$y = \frac{x - \mathbb{E}[x]}{\sqrt{\text{Var}[x] + \varepsilon}} * \gamma + \beta$$

Optimization dynamics

Closed-form Gradient Descent update

Consider the objective:

$$L(\boldsymbol{\theta}) = \frac{1}{2}\boldsymbol{\theta}^\top \underbrace{(\mathbf{X}^\top\mathbf{X})}_{\mathbf{A}}\boldsymbol{\theta},$$

Write the gradient descent update as:

$$\begin{aligned}\boldsymbol{\theta}_{t+1} &\leftarrow \boldsymbol{\theta}_t - \alpha\nabla L\left(\boldsymbol{\theta}_t\right)\\ &= \boldsymbol{\theta}_t - \alpha\mathbf{A}\boldsymbol{\theta}_t\\ &= (\mathbf{I} - \alpha\mathbf{A})\boldsymbol{\theta}_t\end{aligned}$$

Breaking the recursion:

$$\boldsymbol{\theta}_t = (\mathbf{I} - \alpha\mathbf{A})^t\boldsymbol{\theta}_0.$$

B

Gradient Descent, component-wise

Spectral decomposition: $\mathbf{A} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^\top$ so that we can do a change of basis:

$$\begin{aligned}(\mathbf{I} - \alpha\mathbf{A})^t\boldsymbol{\theta}_0 &= \left(\mathbf{I} - \alpha\mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^\top\right)^t\boldsymbol{\theta}_0 \\ &= \left[\mathbf{Q}(\mathbf{I} - \alpha\mathbf{\Lambda})\mathbf{Q}^\top\right]^t\boldsymbol{\theta}_0 \\ &= \mathbf{Q}(\mathbf{I} - \alpha\mathbf{\Lambda})^t\mathbf{Q}^\top\boldsymbol{\theta}_0\end{aligned}$$

In the eigenspace of $\mathbf{Q}$, each coordinate is multiplied by $(1 - \alpha\lambda_i)^t$

Behaviours:

- $0 < \alpha\lambda_i \leq 1$: decays to 0 at a rate depending on λ_i
- $1 < \alpha\lambda_i \leq 2$: oscillations
- $\alpha\lambda_i > 2$: diverges

How does it look in practice?

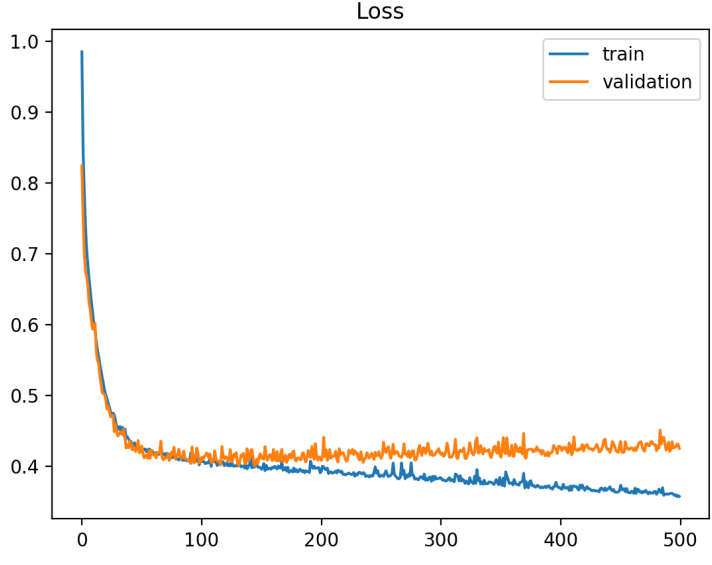


Figure: The optimizer is quickly descending along the direction of the largest eigenvalues.

Algorithms

Fix #1. Momentum

Plain gradient descent:

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \alpha \nabla_{\boldsymbol{\theta}_t} L$$

With momentum:

$$\mathbf{v}_{t+1} \leftarrow \mu \mathbf{v}_t + \nabla_{\boldsymbol{\theta}_t} L$$
$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \alpha \mathbf{v}_{t+1}.$$

- μ: damping factor and controls *how much momentum* we use. - Rolling ball on a slope analogy. - Keeps the *historic* direction in low curvature regions and *dampens* oscillations in high curvature directions.

Fix #1. Momentum

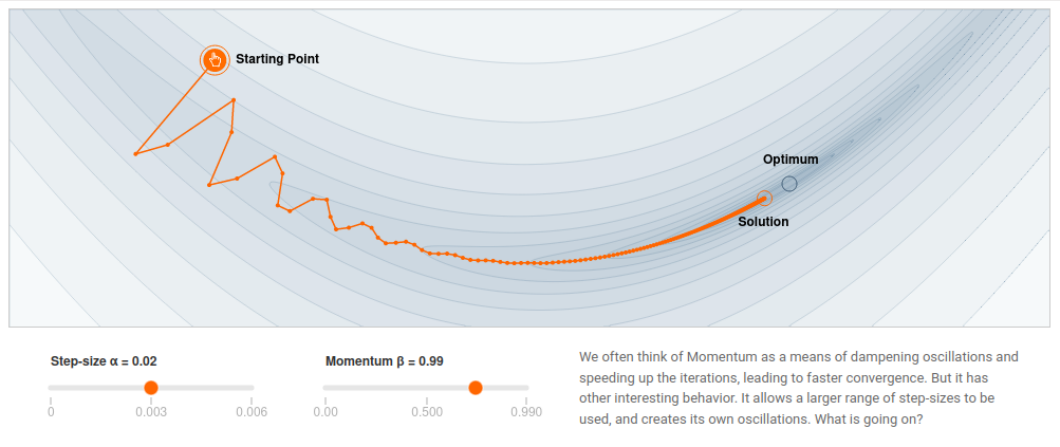


Figure: Why Momentum Really Works

Fix #1. Momentum

- no reason not to use it, it almost always helps...
- ...except when modelling distributions and small changes in θ means a large change in the distribution,
- low memory complexity,
- also check Nesterov momentum.

Fix #2. RMSprop

Plain gradient descent:

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \alpha \nabla_{\boldsymbol{\theta}_t} L$$

With RMSprop:

$$\mathbf{s}_t \leftarrow (1-\gamma)\mathbf{s}_t + \gamma[\nabla_{\boldsymbol{\theta}_t} L]^2,$$
$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \frac{\alpha}{\sqrt{\mathbf{s}_t + \epsilon}} \nabla_{\boldsymbol{\theta}_t} L.$$

- adaptive learning rate!
- small components will use a higher step size
- alternate view: the gradient is rescaled to have norm 1

Fix #3. Adam

Bring back momentum!

$$\mathbf{v}_t \leftarrow \mu \mathbf{v}_{t-1} + (1 - \alpha)\nabla_{\boldsymbol{\theta}_t} L$$
$$\mathbf{s}_t \leftarrow \gamma \mathbf{s}_{t-1} + (1 - \gamma)[\nabla_{\boldsymbol{\theta}_t} L]^2,$$

The parameter update is then:

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \frac{\alpha}{\sqrt{\mathbf{s}_t + \epsilon}} \mathbf{v}_t.$$

Comparison

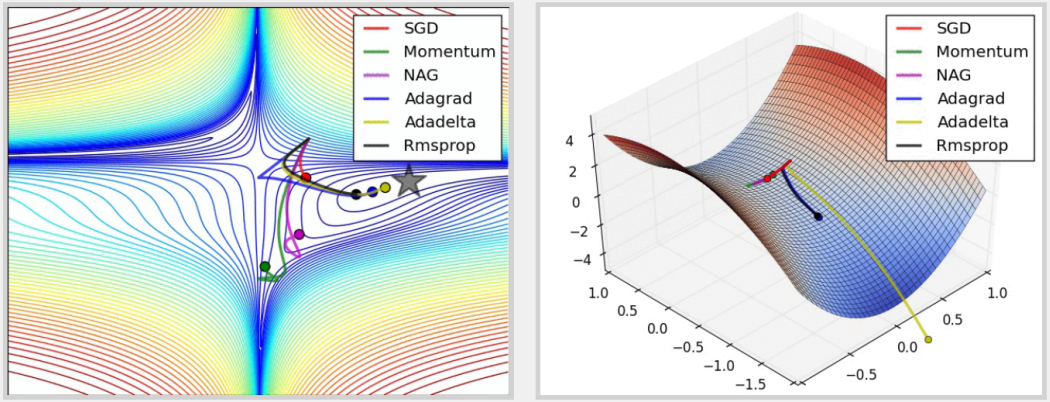


Figure: check animation

Regularization

Regularization. The modified objective

Remember the our purpose is to find $\boldsymbol{\Theta}$ that minimizez the objective:

$$\begin{aligned}\hat{\boldsymbol{\Theta}} &= \arg\min_{\boldsymbol{\Theta}} \left[L(\boldsymbol{\Theta})\right] \\ &= \arg\min_{\boldsymbol{\Theta}} \left[\sum_i l_i\left[\mathbf{x}_i, \mathbf{y}_i\right]\right]\end{aligned}$$

We can bias the optimization towards certain solutions:

$$\hat{\boldsymbol{\Theta}} = \arg\min_{\boldsymbol{\Theta}} \left[\sum_i l_i\left[\mathbf{x}_i, \mathbf{y}_i\right] + \lambda \cdot g(\boldsymbol{\Theta})\right],$$

where $g(\boldsymbol{\Theta})$ is a function that returns a (scalar) large value when the weights are large.

Regularization

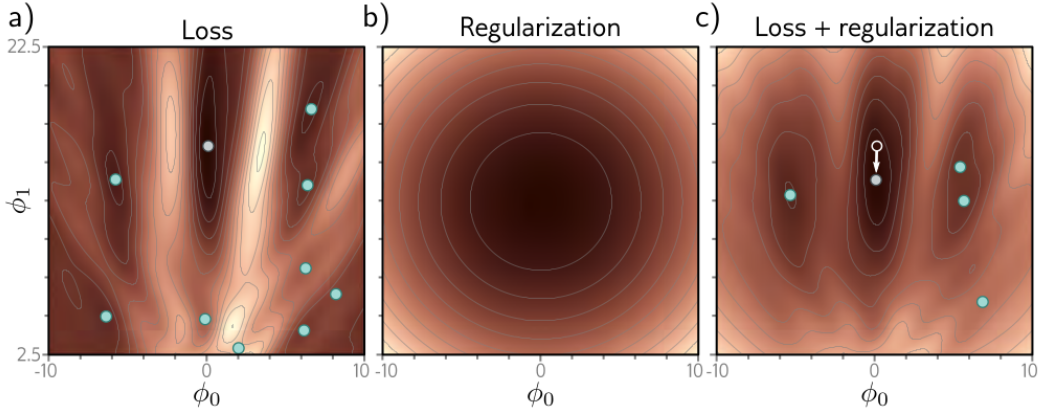


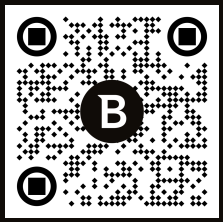
Figure: Effect of regularization objective.

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \left[\sum_i l_i \left[\mathbf{x}_i, \mathbf{y}_i \right] + \lambda \cdot g(\boldsymbol{\theta}) \right],$$

B

Thank you!

florin.gogianu@gmail.com



bit-ml.github.io