Deep Learning

Optimizers

Edgar Roman-Rangel.
edgar.roman@itam.mx

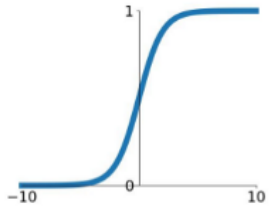
Department of Computer Science.
Instituto Tecnológico Autónomo de México, ITAM.

Outline

Comparison

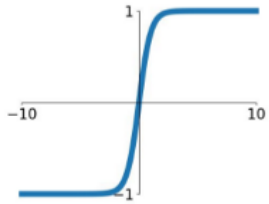
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



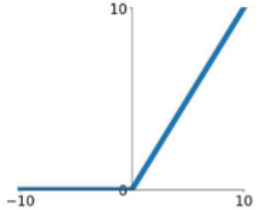
tanh

$$\tanh(x)$$



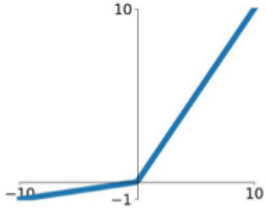
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

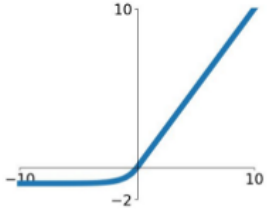


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Linear

$$a = \sum_{n=0}^{N} x_n \omega_n.$$

Derivative:

$$\frac{\partial a}{\partial \omega_i} = x_i.$$

Sigmoid

$$a = \sigma(s) = \frac{1}{1 + e^{-s}}.$$

Derivative:

$$\sigma'(s) = \sigma(s)(1 - \sigma(s)).$$

Tanh

$$a = \frac{e^s - e^{-s}}{e^s + e^{-s}}.$$

Derivative:

$$a' = 1 - a^2.$$

ReLU

Logistic functions suffer of the so-called *vanishing gradient* issue. Rectified Linear Units (ReLU) are an alternative:

$$a = \begin{cases} 0, & s < 0, \\ s, & s \geq 0. \end{cases}$$

Derivative:

$$a' = \begin{cases} 0, & s < 0, \\ 1, & s \geq 0. \end{cases}$$

They are also more efficient computationally speaking.

Leaky ReLU

$$a = \begin{cases} \alpha s, & s < 0, \\ s, & s \geq 0, \end{cases}$$

with $0 < \alpha < 1$.

Derivative:

$$a' = \begin{cases} \alpha, & s < 0, \\ 1, & s \geq 0. \end{cases}$$

ELU: Exponential Linear Unit

$$a = \begin{cases} \alpha(e^s - 1), & s < 0, \\ s, & s \geq 0, \end{cases}$$

with $0 < \alpha < 1$.

Derivative:

$$a' = \begin{cases} a + \alpha, & s < 0, \\ 1, & s \geq 0. \end{cases}$$

Softmax

All previous functions are element-wise operations. Softmax is a vector normalizer.

$$a_i = \frac{e^{s_i}}{\sum_j e^{s_j}}.$$

Derivative:

$$a_i' = a_i(1 - a_i).$$

Used to exaggerate the most probable of the elements of the vector. Useful in output layers for multi-class classification problems.

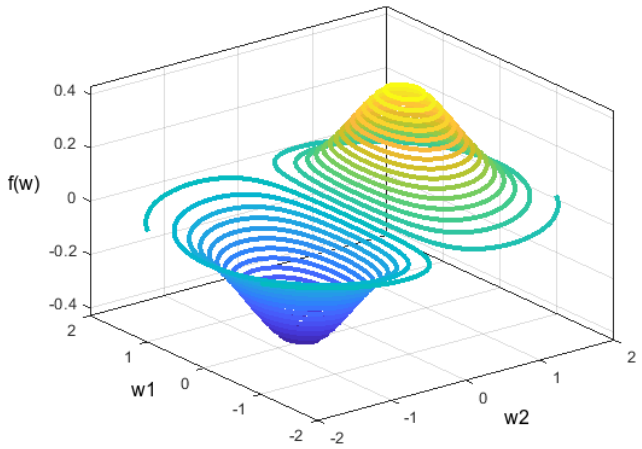
Common use scenarios

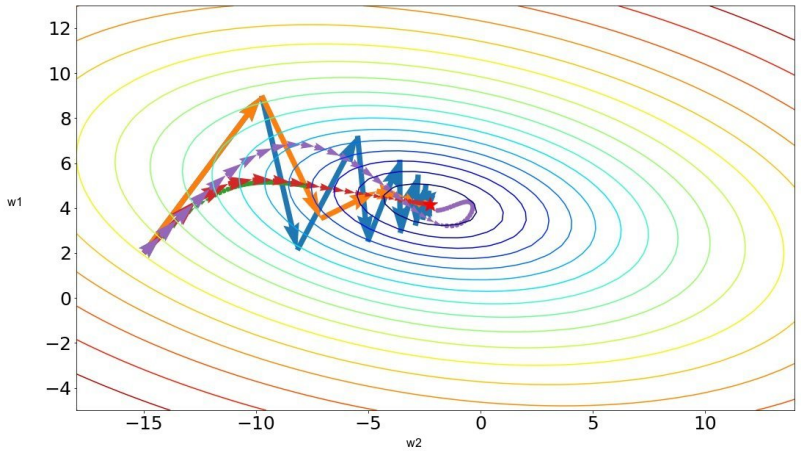
Activation	Use
ReLU (or variants)	All hidden layers in all scenarios.
Sigmoid (tanh)	Output layer for binary classification.
Sigmoid	Output layer for regression with $0 \leq y \leq 1$.
Tanh	Output layer for regression with $-1 \leq y \leq 1$.
Linear	Output layer for unbounded regression.
Softmax	Output layer for multi-class classification.

Outline

Parameter space

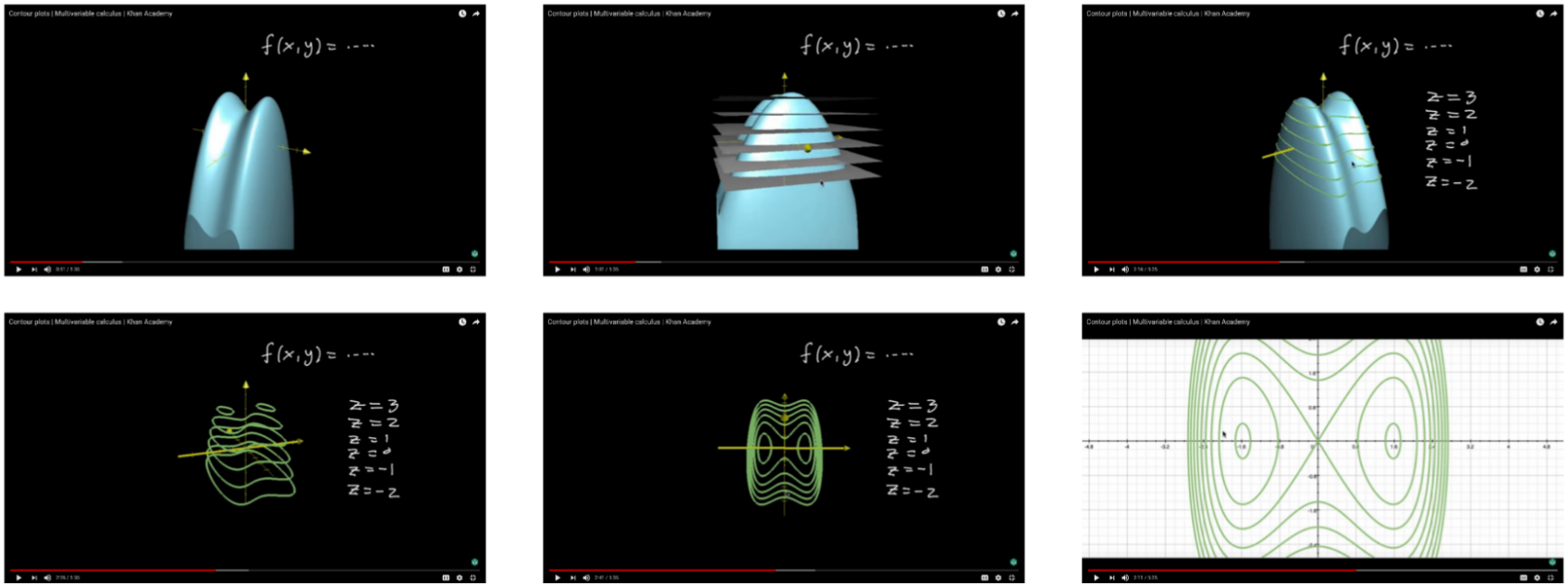
Example of different loss functions with 2 parameters.
Seen in 3D, and from above.





Colored arrows represent different possible paths to reach the local minimum, as followed by different optimization strategies.

Several local minima



Images from Video: Contour plots - Multivariable calculus - Khan Academy
https://www.youtube.com/watch?v=WsZj5Rb6do8

Example for SGD

Batch gradient descent
Mini-batch gradient Descent
Stochastic gradient descent

SGD Limitation

$$\omega_i = \omega_i - \nabla_{\omega_i} \mathcal{L}(y, \hat{y}).$$

- Slows down around ravines.
- Oscillates across the slopes of the ravine.
- Limited progress towards the local minimum.
- Might never escape from saddle points.

Qian, 1999. "On the momentum term in gradient descent learning algorithms".

Outline

Notation

In this section, we will use subscripts to index time, e.g., ω_t refers to the value of parameter ω at time t.

Also, we omit the position index (i-th element) commonly seen as subscript, i.e, ω_i.

Momentum

$$\omega_t = \omega_{t-1} - v_t,$$

$$v_t = \gamma v_{t-1} + \eta \nabla_\omega \mathcal{L}(y, \hat{y}), \qquad v_0 = 0, \quad \gamma = 0.9, \eta \approx 0.001.$$

- Accelerates SGD in the relevant direction.
- Dampens oscillations.
- Includes a fraction of the historic direction.
- Momentum accelerates for gradients pointing in the same direction, and reduces for those in changing direction.

Sutskever et al., 2013. "On the importance of initialization and momentum in deep learning".

Nesterov Accelerated Gradient (NAG)

$$\omega_t = \omega_{t-1} - v_t,$$

$$v_t = \gamma v_{t-1} + \eta \nabla_{(\omega - \gamma v_{t-1})} \mathcal{L}(y, \hat{y}),$$

- $\nabla_{(\omega - \gamma v_{t-1})}$ approximates the next position of ω.
- Looks ahead by calculating the gradient w.r.t. future positions.
- Anticipates changes in the direction of the gradient.

Nesterov, 1983. "A method for unconstrained convex minimization problemwith the rate of convergence o(1/k2)".

Adaptive Gradient (AdaGrad)

$$\omega_t = \omega_{t-1} - \eta \frac{1}{\sqrt{G_t + \epsilon}} g_t, \qquad g_t = \nabla_\omega \mathcal{L}(y, \hat{y}),$$

$$G_t = \sum_{k=0}^{t} g_t^2, \qquad \epsilon \approx 1e^{-8}.$$

- G_t: sum of gradients^2 up to time t (heavy memory loads).
- Adapts η at each time step (always decreasing).
- Works well on sparse data and large models.

Duchi et al., 2011. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization".

Adadelta

$$\omega_t = \omega_{t-1} - \eta \frac{1}{\sqrt{\mathbb{E}[g^2]_t}} g_t, \qquad g_t = \nabla_\omega \mathcal{L}(y, \hat{y}),$$

$$\mathbb{E}[g^2]_t = \gamma \mathbb{E}[g^2]_{t-1} + (1-\gamma) g_t^2, \qquad \gamma = 0.9.$$

- Addresses the issue of monotonically decreasing η.
- Restricts the past to a moving window.
- Recursively computes the sum of past gradients using exponential smoothing.

Zeiler, 2012. "ADADELTA: An Adaptive Learning Rate Method".
RMSprop: a variant by Hinton (unpublished).

Adaptive Momentum (Adam)

Adadelta + Momentum.

$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$, first moment estimate (mean),
$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$, second moment estimate (stddv).

Correcting for bias towards zero:

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}; \qquad \hat{v}_t = \frac{v_t}{1-\beta_2^t}.$$

$$\omega_t = \omega_{t-1} - \eta \frac{1}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t,$$

Kingma & Ba, 2015. "Adam: a Method for Stochastic Optimization".
Other variants: AdaMax, Nadam, AMSGrad.

To know more

Ruder, 2016. "An overview of gradient descent optimization algorithms". https://arxiv.org/abs/1609.04747

Q&A

Thank you!

edgar.roman@itam.mx