CIS 419/519 Recitation

Varun Jana, Chang Liu

Oct 13/14 2020

Penn Engineering

Content

- Perceptron & Activation Functions Overview
- Adagrad
- More about Regularization

Part 1: Perceptron & Act. Fns.

and the intuition

Intro to Perceptron

- Our lecture slides have some good content, but let's discuss some roadmaps & high level intuition stuff!
- https://www.seas.upenn.edu/~cis519/fall2020/assets/lectures/lecture-4/Lecture4-online.pdf (Page 50 …)

Intro to Perceptron

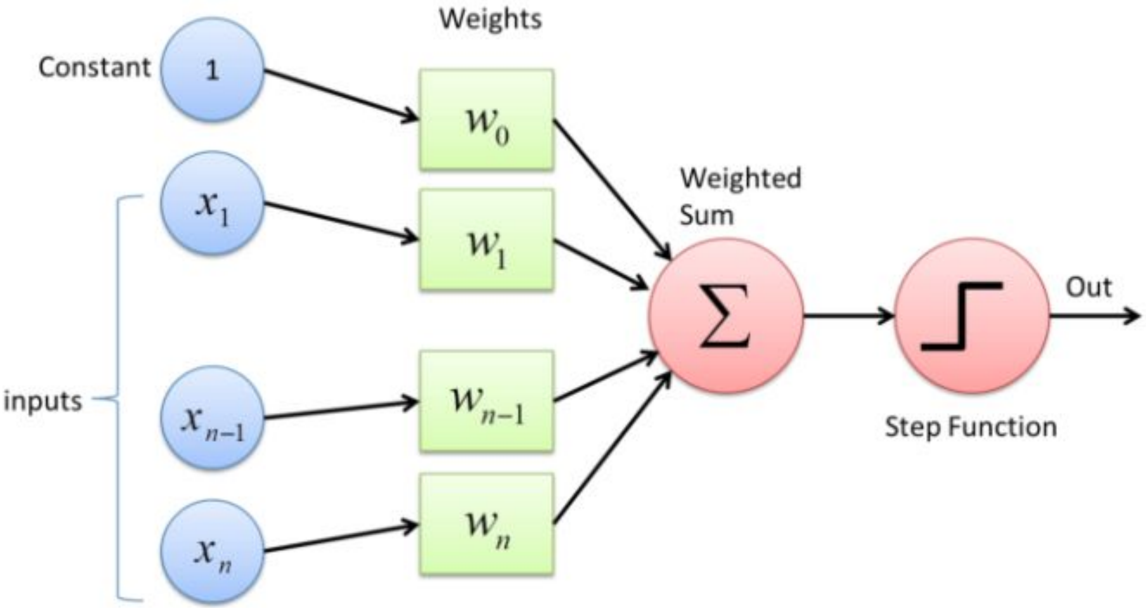


Fig : Perceptron

https://towardsdatascience.com/what-the-hell-is-perceptron-626217814f53

Activation Functions - Overview

- Most crucial part of every *neural network :*

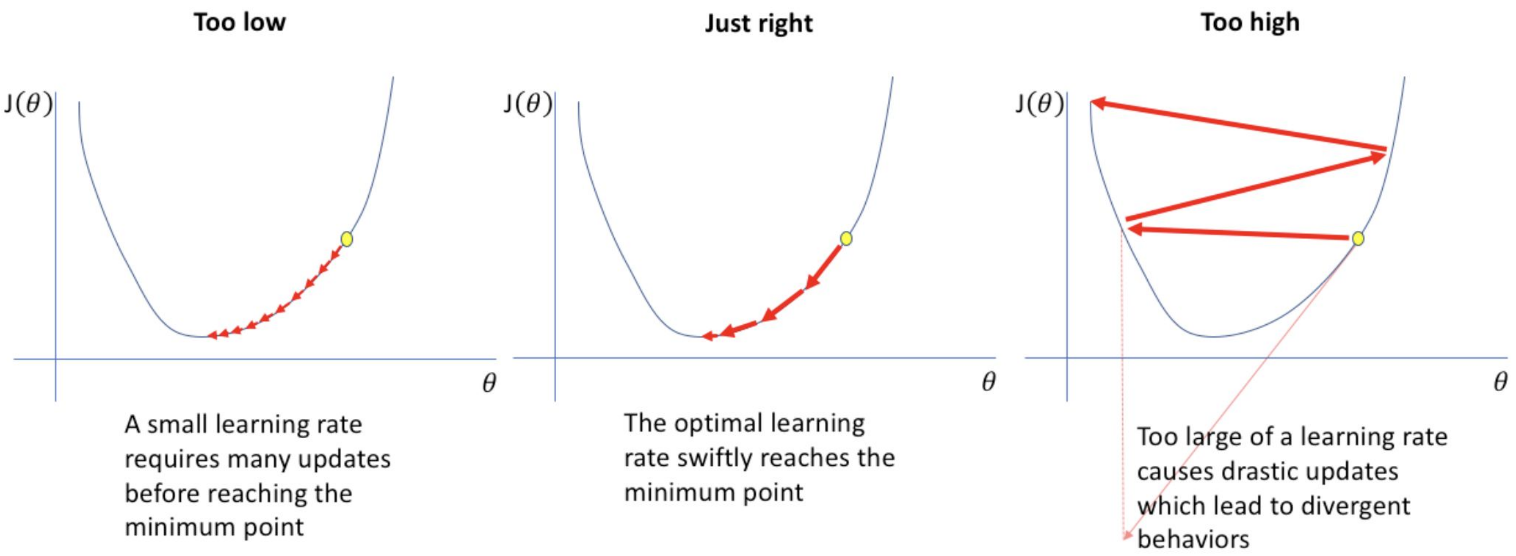
https://www.analyticssteps.com/blogs/7-types-activation-functions-neural-network

  - Great read! I definitely recommend trying these out in later HWs!

Part 2: Adagrad

What is Adagrad?

- tuning a fix learning rate is tricky
- adaptively scaled the learning rate for each dimension base on historical information

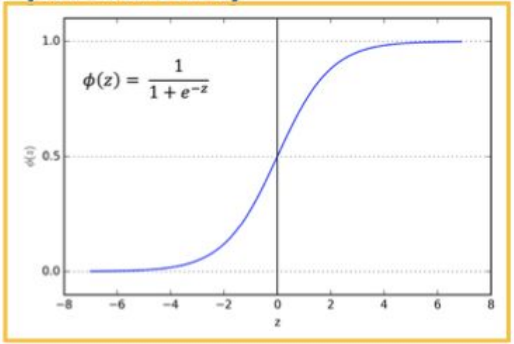


Conditional Probability

- Converting the output of a Perceptron to a conditional probability

$$P(y = +1|\boldsymbol{x}) = \frac{1}{1 + e^{-A\boldsymbol{w}^T\boldsymbol{x}}}$$

- The parameter A can be tuned on a development set

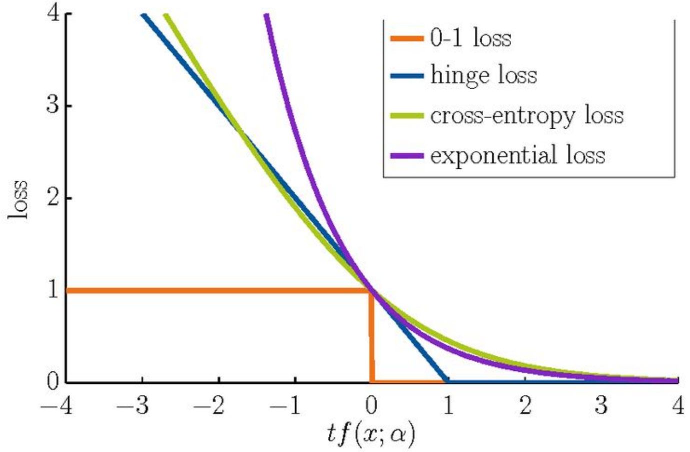


$$\frac{1}{1+e^{-w^T x}} > \frac{1}{2}$$

Recap on SGD/Gradient Descent

$$w_{t+1} = w_t - r_t\, g_w \boxed{\mathcal{L}(\mathbf{x}, y, \mathbf{w}, \theta)} = w_t - r_t\, g_t$$

$$\mathcal{L}(\mathbf{x}, y, \mathbf{w}, \theta) = \max\{0, 1 - y(\mathbf{w}^\intercal \mathbf{x} + \theta)\}$$



Gradient Update

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - r_t\, \boldsymbol{g}_w\, Q(\boldsymbol{z}_t, \boldsymbol{w}_t) = \boldsymbol{w}_t - r_t\, \boxed{\boldsymbol{g}_t}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = \begin{cases} \mathbf{0} & \text{if } y(\mathbf{w}^\intercal \mathbf{x} + \theta) > 1 \\ -y \cdot \mathbf{x} & \text{otherwise} \end{cases}$$

$$\frac{\partial \mathcal{L}}{\partial \theta} = \begin{cases} 0 & \text{if } y(\mathbf{w}^\intercal \mathbf{x} + \theta) > 1 \\ -y & \text{otherwise} \end{cases}$$

Now! Learning rate!

$$w_{t+1} = w_t - r_t\, g_w\, Q(z_t, w_t) = w_t - \boxed{r_t}\, g_t$$

$$G_j^t = \sum_{k=1}^{t} \left(\frac{\partial \mathcal{L}}{\partial w_j^k} \right)^2$$

$$H^t = \sum_{k=1}^{t} \left(\frac{\partial \mathcal{L}}{\partial \theta^k} \right)^2$$

$$\mathbf{w}^{t+1} \leftarrow \mathbf{w}^t + \eta \cdot \frac{y \cdot \mathbf{x}}{\sqrt{\mathbf{G}^t}}$$

$$\theta^{t+1} \leftarrow \theta^t + \eta \frac{y}{\sqrt{H^t}}$$

Why Adagrad?

Advantages of Using AdaGrad

- it eliminates the need to manually tune the learning rate
- convergence is faster and more reliable – than simple SGD when the scaling of the weights is unequal
- It is not very sensitive to the size of the master step

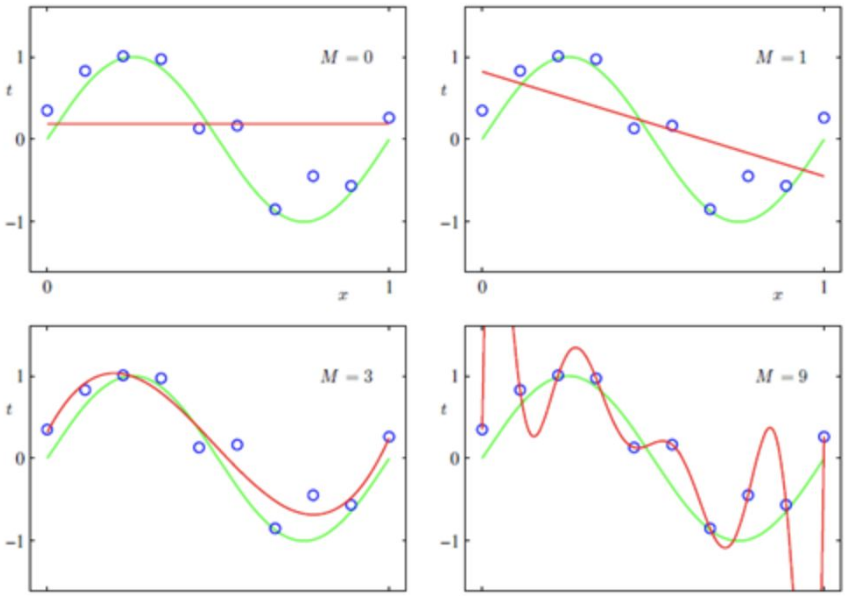
Overview of Gradient Descent Based Optimization Algorithm:
https://ruder.io/optimizing-gradient-descent/

Tips for hw

1. separately update w and theta
2. keep track of G for wi and H for theta
3. use sqrt sum of G / H to update learning rate

Part 3: Regularization

Why Regularization?



- avoid overfitting
- more robust to noise

Regularization Via Averaged Perceptron

- Variables:
  - m: number of examples
  - k: number of mistakes
  - c_i: consistency count for hypothesis $\boldsymbol{v}_i$
  - T: number of epochs
- Input: a labeled training $\{\{\boldsymbol{x}_1, y_1\}, \{\boldsymbol{x}_2, y_2\}, \dots \{\boldsymbol{x}_m, y_m\}\}$
- Output: a list of weighted perceptrons $\{\{\boldsymbol{v}_1, c_1\}, \dots, \{\boldsymbol{v}_k, c_k\}\}$
- Initialize: $k = 0, \boldsymbol{v}_1 = 0, c_1 = 0$
- Repeat T times:
  - For $i = 1, \dots, m$;
  - Compute prediction $y' = \mathrm{sgn}(\boldsymbol{v}_k^T \cdot \boldsymbol{x}_i)$
  - If $y' = y$, then $c_k = c_k + 1$
    else: $\boldsymbol{v}_{k+1} = \boldsymbol{v}_k + y_i \boldsymbol{x}$; $c_{k+1} = 1$; $k = k + 1$
- Prediction:
  - Given: a list of weighted perceptrons $\{\{\boldsymbol{v}_1, c_1\}, \dots, \{\boldsymbol{v}_k, c_k\}\}$; a new example x
  - Predict the label (x) as follows: $y(\boldsymbol{x}) = \mathrm{sgn}[\sum_1^k c_i(\boldsymbol{v}_i^T \boldsymbol{x})]$

- This can be done on top of any online mistake driven algorithm.
- In HW 2 you will run it over three different algorithms.
- The implementation requires thinking.

Averaged version of Perceptron/Winnow is as good as any other linear learning algorithm, if not better.

Perceptron

Input set of examples and their labels

$$Z = ((\boldsymbol{x}_1, y_1), \dots (\boldsymbol{x}_m, y_m)) \in \boldsymbol{R}^n \times \{-1,1\}^m, \eta, \theta_{Init}$$

- Initialize $\boldsymbol{w} \leftarrow 0$ and $\theta \leftarrow \theta_{Init}$
- For every training epoch:
- for every $\boldsymbol{x}_j \in \boldsymbol{X}$:
  - $\hat{y} \leftarrow sign(<\boldsymbol{w}, \boldsymbol{x}_j> - \theta)$
  - If $(\hat{y} \neq y_j)$
    - $\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta y_j \boldsymbol{x}_j$
    - $\theta \leftarrow \theta + \eta y_j$

Just to make sure we understand that we learn both $\boldsymbol{w}$ and θ

The Key Ideas

- want to somehow regularize or shrink coefficient (w) towards zero

- In other words, this technique discourages learning a more complex or flexible model, so as to avoid the risk of overfitting

Mathematically

$$J(\mathbf{w}) = \boxed{\frac{1}{m}\sum_1^m Q(\mathbf{z}_i, \mathbf{w}_i)} + \boxed{\lambda\, R_i\,(\mathbf{w}_i)}$$

Loss:

Regularization:

LMS case: $Q((\boldsymbol{x}, y), \boldsymbol{w}) = (y - \boldsymbol{w}^T \boldsymbol{x})^2$

- $R(\boldsymbol{w}) = ||\boldsymbol{w}||_2^2$ gives the optimization problem called Ridge Regression.
- $R(\boldsymbol{w}) = ||\boldsymbol{w}||_1$ gives a problem called the LASSO problem

Hinge Loss case: $Q((\boldsymbol{x}, y), \boldsymbol{w}) = \max(0, 1 - y\, \boldsymbol{w}^T \boldsymbol{x})$

- $R(\boldsymbol{w}) = ||\boldsymbol{w}||_2^2$ gives the problem called Support Vector Machines

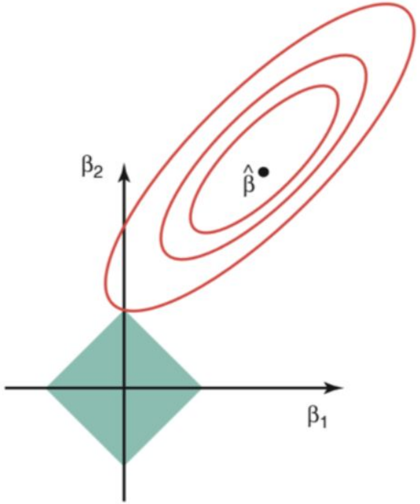
Logistics Loss case: $Q((\boldsymbol{x}, y), \boldsymbol{w}) = \log(1 + \exp\{-y\, \boldsymbol{w}^T \boldsymbol{x}\})$

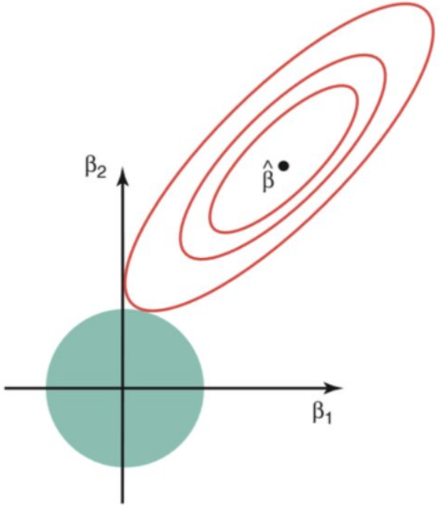
- $R(\boldsymbol{w}) = ||\boldsymbol{w}||_2^2$ gives the problem called Logistics Regression

L2: $\lambda \sum_{i=1}^{m} |w_i|^2$

L1: $\lambda \sum_{i=1}^{m} |w_i|$

Difference between L1 & L2 Regularization





L1:

$$\lambda \sum_{i=1}^{m} |w_i|$$

L2:

$$\lambda \sum_{i=1}^{m} |w_i|^2$$

Choose which?

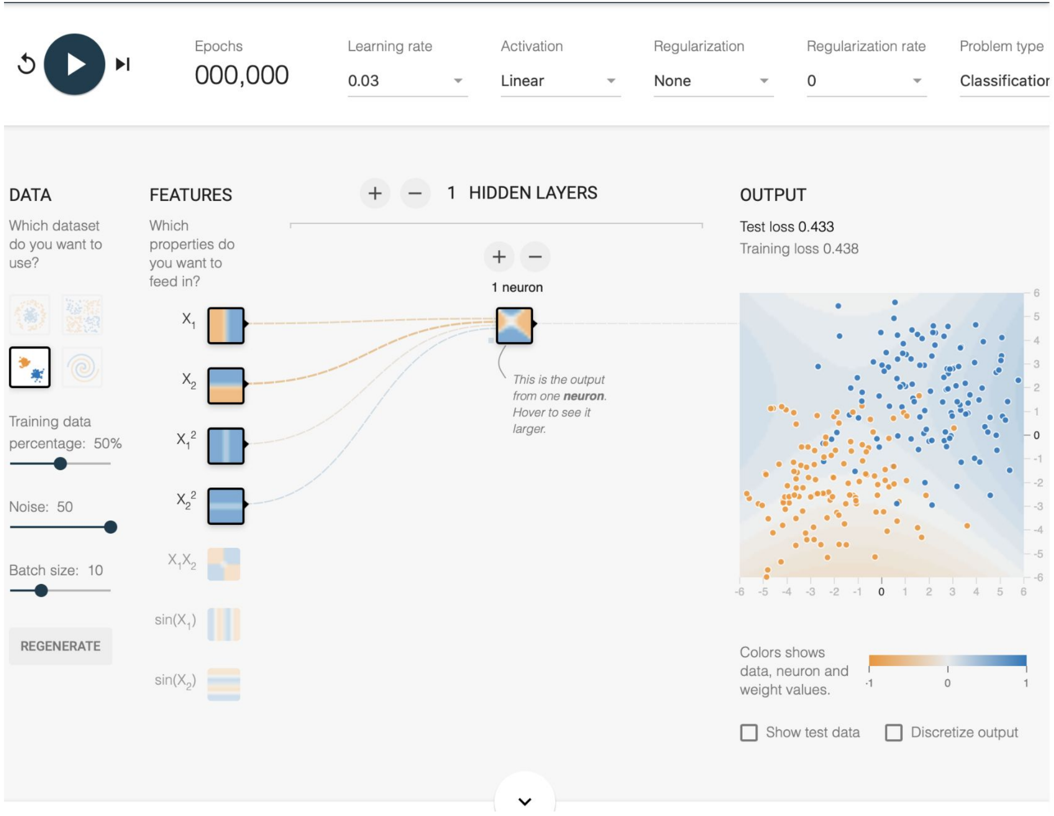
L1:

- penalizes sum of absolute value of weights.
- has a sparse solution
- multiple solutions
- has built in feature selection
- robust to outliers
- generates model that are simple and interpretable but cannot learn complex patterns

L2:

- penalizes sum of square weights.
- has a non sparse solution
- has one solution
- has no feature selection
- is not robust to outliers
- better prediction when output variable is a function of all input features
- is able to learn complex data patterns

Playground



https://developers.google.com/machine-learning/crash-course/regularization-for-simplicity/playground-exercise-examining-l2-regularization