Loss functions, optimizers and tricks to train a NN

Komal Teru
Mila, McGill University

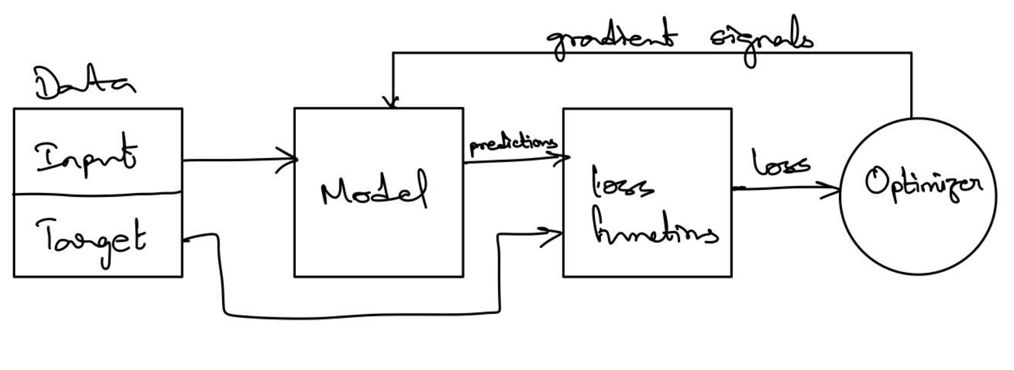
Topics to cover today

- Review of MLP
- More types of optimizers and loss functions
- Tricks to train an MLP

Review of MLP

Key ingredients of ML pipeline

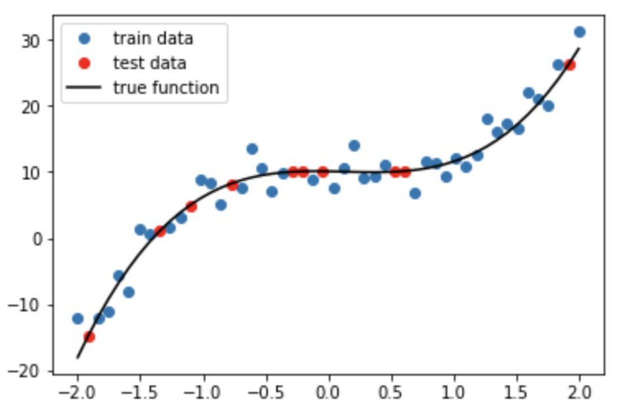
- <u>Data</u> : image, text, speech, graph etc.
- <u>Model</u> : MLP, CNN, RNN, etc.
- <u>Loss function</u> : MSE, cross-entropy, etc.
- <u>Optimization algorithm (optimizer)</u> : SGD, Adam, Adagrad, etc.



Review of MLP

Multi layer perceptron for regression

Data



$X \in \mathbb{R}$

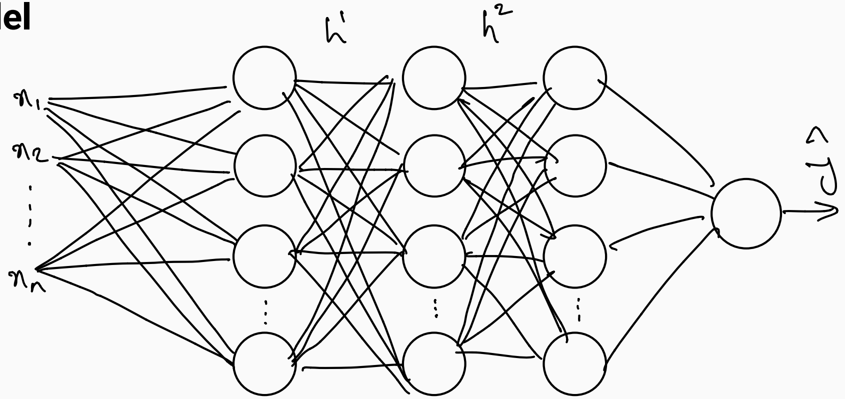
$Y \in \mathbb{R}$

$D = (X, Y)$

Review of MLP

Multi layer perceptron for regression

Model



$$h^1 = f(x_i W^1 + b^1)$$

$$h^2 = f(h^1 W^2 + b^2)$$

$$\hat{y}_i = \underline{\underline{h^2 W^3 + b^3}}$$

$$\boxed{mse = \frac{1}{m} \sum_{i=1}^{m} (\hat{y}_i - y_i)^2}$$

Loss function : mean squared error **Optimizer:** SGD

Review of MLP

Multi layer perceptron for regression

Key points:

- The final layer doesn’t have activation function. (why?)
- Bias is necessary to adjust any offset in the data.
- Activation function is necessary to fit to non-linear data.

Demo link: https://colab.research.google.com/drive/1sB1bGE2Q76fBv1NtcWT_2zbuhJMFVsr-

Optimizers

SGD algorithm

Algorithm 1 Stochastic gradient descent (SGD) update at training iteration k

Require: Learning rate ϵ_k.
Require: Initial parameter $\boldsymbol{\theta}$
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient estimate: $\hat{\boldsymbol{h}} \leftarrow +\frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon\hat{\boldsymbol{h}}$
  end while

Optimizers

Momentum method

Algorithm 1 Stochastic gradient descent (SGD) with momentum

Require: Learning rate ϵ, momentum parameter α.

Require: Initial parameter $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$.

  while stopping criterion not met **do**

    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.

    Compute gradient estimate: $\boldsymbol{h} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$

    Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \boldsymbol{h}$

    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$

  end while

Optimizers

Adagrad algorithm

Algorithm 1 The AdaGrad algorithm

Require: Global learning rate ϵ
Require: Initial parameter $\boldsymbol{\theta}$
Require: Small constant δ, perhaps 10^{-7}, for numerical stability
Initialize gradient accumulation variable $\boldsymbol{r} = \mathbf{0}$
while stopping criterion not met **do**
Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
Compute gradient: $\boldsymbol{h} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
Accumulate squared gradient: $\boldsymbol{r} \leftarrow \boldsymbol{r} + \boldsymbol{h} \odot \boldsymbol{h}$
Compute update: $\Delta\boldsymbol{\theta} \leftarrow -\frac{\epsilon}{\delta + \sqrt{\boldsymbol{r}}} \odot \boldsymbol{h}$. (Division and square root applied element-wise)
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
end while

Optimizers

RMSProp method

Algorithm 1 The RMSProp algorithm

Require: Global learning rate ϵ, decay rate ρ.
Require: Initial parameter $\boldsymbol{\theta}$
Require: Small constant δ, usually 10^{-6}, used to stabilize division by small numbers.

Initialize accumulation variables $\boldsymbol{r} = 0$

while stopping criterion not met **do**

Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.

Compute gradient: $\boldsymbol{h} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$

Accumulate squared gradient: $\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1 - \rho) \boldsymbol{h} \odot \boldsymbol{h}$

Compute parameter update: $\Delta \boldsymbol{\theta} = -\frac{\epsilon}{\sqrt{\delta + \boldsymbol{r}}} \odot \boldsymbol{h}$. ($\frac{1}{\sqrt{\delta + \boldsymbol{r}}}$ applied elem-wise)

Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta \boldsymbol{\theta}$

end while

Optimizers

Adam method

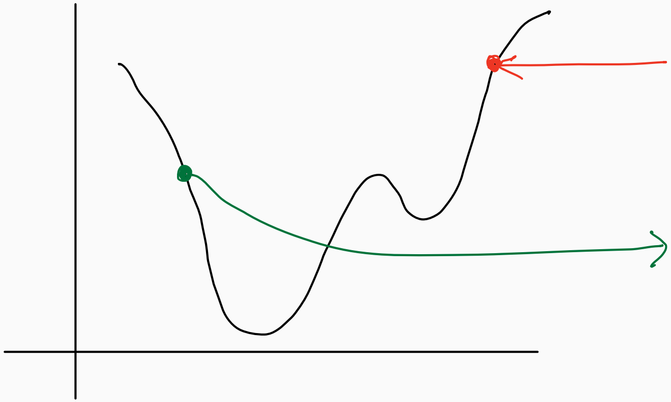
Algorithm 1 The Adam algorithm

Require: Step size ϵ (Suggested default: 0.001)
Require: Exponential decay rates for moment estimates, ρ_1 and ρ_2 in $[0, 1)$. (Suggested defaults: 0.9 and 0.999 respectively)
Require: Small constant δ used for numerical stabilization. (Suggestion: 10^{-8})
Require: Initial parameters $\boldsymbol{\theta}$

Initialize 1st and 2nd moment variables $\boldsymbol{s} = \boldsymbol{0}$, $\boldsymbol{r} = \boldsymbol{0}$
Initialize time step $t = 0$
while stopping criterion not met **do**
  Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
  Compute gradient: $\boldsymbol{h} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
  $t \leftarrow t + 1$
  Update biased first moment estimate: $\boldsymbol{s} \leftarrow \rho_1 \boldsymbol{s} + (1 - \rho_1)\boldsymbol{h}$
  Update biased second moment estimate: $\boldsymbol{r} \leftarrow \rho_2 \boldsymbol{r} + (1 - \rho_2)\boldsymbol{h} \odot \boldsymbol{h}$
  Correct bias in first moment: $\hat{\boldsymbol{s}} \leftarrow \frac{\boldsymbol{s}}{1 - \rho_1^t}$
  Correct bias in second moment: $\hat{\boldsymbol{r}} \leftarrow \frac{\boldsymbol{r}}{1 - \rho_2^t}$
  Compute update: $\Delta \boldsymbol{\theta} = -\epsilon \frac{\hat{\boldsymbol{s}}}{\sqrt{\hat{\boldsymbol{r}}} + \delta}$ (operations applied element-wise)
  Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta \boldsymbol{\theta}$
end while

Xavier/Glorot Initialization

Initialization of weights is key to the training dynamics.



If you start at this point, you can reach the global minimum by using various tricks like momentum, RMSProp, etc.

Starting here will make the learning process easy and fast.

Xavier/Glorot Initialization

These initializations put you in a better position to start your optimization

Xavier uniform initialization

$$\theta \sim U(-a, a)$$

$$a = \sqrt{\frac{6}{f_{in} + f_{out}}} \approx \sqrt{\frac{3}{f_{in}}}$$

Xavier normal initialization

$$\theta = N(0, \sigma^2)$$

$$\sigma = \sqrt{\frac{2}{f_{in} + f_{out}}} \approx \sqrt{\frac{1}{f_{in}}}$$

In class exercise

We feed a 4 dimensional data to a 1 hidden layer MLP with 30 nodes/perceptrons. The output has a dimension of 4. How many trainable parameters does the model have?

L2 Regularization

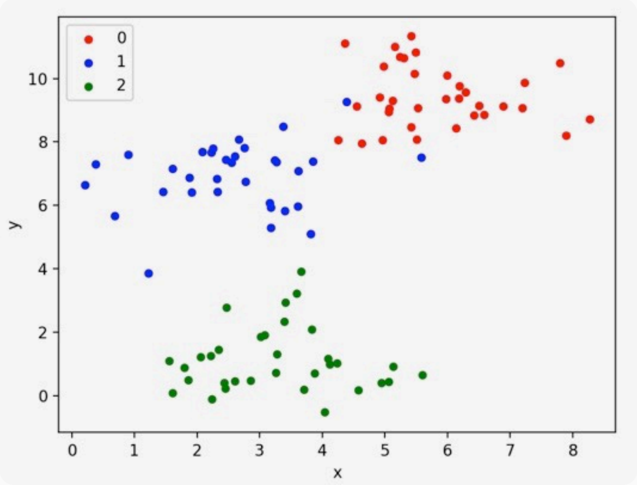
$$L(x,y) \equiv \sum_{i=1}^{n} (y_i - h_\theta(x_i))^2 + \boxed{\lambda \sum_{i=1}^{n} \theta_i^2}$$

weight penalty.

L2 regularization limits the models capacity by limiting the range of values it’s parameters can take.

Classification model

Data



$X_i \in \mathbb{R}$, $O_i \in \{0, 1, 2\}$

and $Y_{1\text{-hot}} \in \mathbb{B}^3$ where $\mathbb{B} = \{0, 1\}$

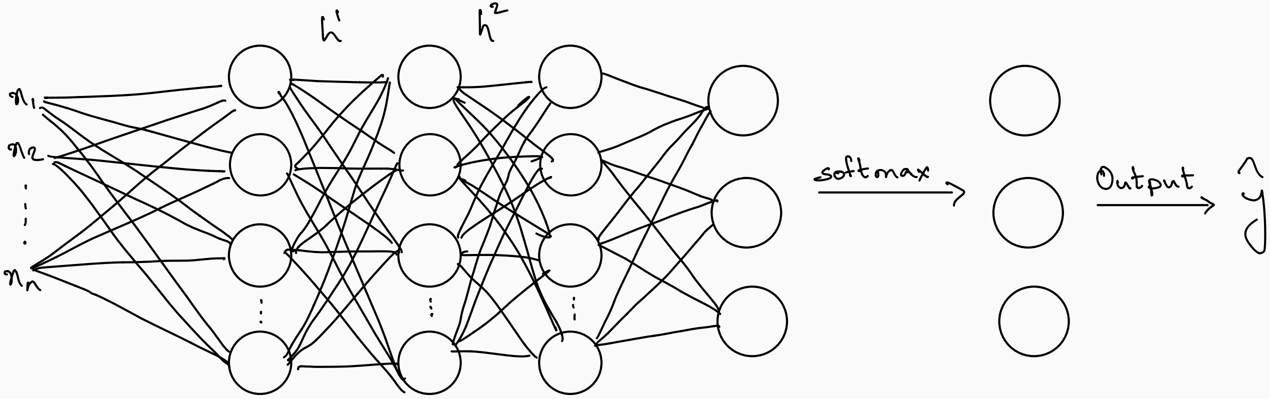
Eg: If $O_i = 0 \Rightarrow y_i^{1\text{-hot}} = [1, 0, 0]$

$O_i = 1 \Rightarrow y_i^{1\text{-hot}} = [0, 1, 0]$

$O_i = 2 \Rightarrow y_i^{1\text{-hot}} = [0, 0, 1]$

Classification model

Model



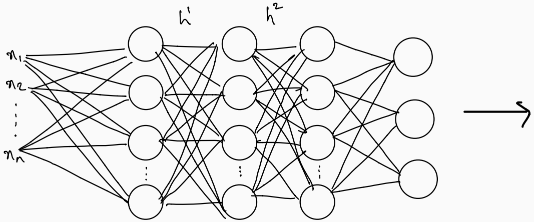
Loss function : cross-entropy **Optimizer:** SGD

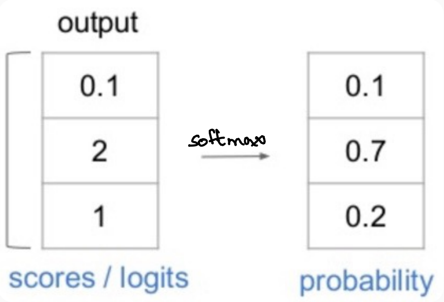
Classification model

Softmax function

scores (logits)

$$S(l_i) = \frac{e^{l_i}}{\sum_k e^{l_k}}$$





Ref: http://neuralnetworksanddeeplearning.com/chap3.html#softmax

Classification model

Cross-entropy

$$\mathcal{L} = \frac{1}{m} \sum_{i=1}^{m} -\log \hat{y}_i^{(o_i)}$$

Demo link: https://colab.research.google.com/drive/1fwnparxF9tEaV5--MBkqg0jOpatIU4lj