A list of cost functions used in neural networks, alongside applications

What are common cost functions used in evaluating the performance of neural networks?

Details

(feel free to skip the rest of this question, my intent here is simply to provide clarification on notation that answers may use to help them be more understandable to the general reader)

I think it would be useful to have a list of common cost functions, alongside a few ways that they have been used in practice. So if others are interested in this I think a community wiki is probably the best approach, or we can take it down if it's off topic.

Notation

So to start, I'd like to define a notation that we all use when describing these, so the answers fit well with each other.

This notation is from Neilsen's book.

A Feedforward Neural Network is a many layers of neurons connected together. Then it takes in an input, that input "trickles" through the network and then the neural network returns an output vector.

More formally, call a^i_j the activation (aka output) of the j^{th} neuron in the i^{th} layer, where a^1_j is the j^{th} element in the input vector.

Then we can relate the next layer's input to it's previous via the following relation:

$$a^i_j = \sigma(\sum_k (w^i_{jk} \cdot a^{i-1}_k) + b^i_j)$$

where

σ is the activation function,

w^i_{jk} is the weight from the k^{th} neuron in the $(i-1)^{th}$ layer to the j^{th} neuron in the i^{th} layer,

b^i_j is the bias of the j^{th} neuron in the i^{th} layer, and

a^i_j represents the activation value of the j^{th} neuron in the i^th layer.

Sometimes we write z^i_j to represent $\sum_k (w^i_{jk} \cdot a^{i-1}_k) + b^i_j$, in other words, the activation value of a neuron before applying the activation function.

enter image description here

For more concise notation we can write

$$a^i = \sigma(w^i \times a^{i-1} + b^i)$$

To use this formula to compute the output of a feedforward network for some input $I \in \mathbb{R}^n$, set $a^1 = I$, then compute a^2, a^3, ...,a^m, where m is the number of layers.

Introduction

A cost function is a measure of "how good" a neural network did with respect to it's given training sample and the expected output. It also may depend on variables such as weights and biases.

A cost function is a single value, not a vector, because it rates how good the neural network did as a whole.

Specifically, a cost function is of the form

$$C(W, B, S^r, E^r)$$

where W is our neural network's weights, B is our neural network's biases, S^r is the input of a single training sample, and E^r is the desired output of that training sample. Note this function can also potentially be dependent on y^i_j and z^i_j for any neuron j in layer i, because those values are dependent on W, B, and S^r.

In backpropagation, the cost function is used to compute the error of our output layer, δ^L, via

$$\delta^L_j = \frac{\partial C}{\partial a^L_j} \sigma'(z^i_j)$$

.

Which can also be written as a vector via

$$\delta^L = \nabla_a C \odot \sigma'(z^i)$$

.

We will provide the gradient of the cost functions in terms of the second equation, but if one wants to prove these results themselves, using the first equation is recommended because it's easier to work with.

Cost function requirements

To be used in backpropagation, a cost function must satisfy two properties:

1: *The cost function C must be able to be written as an average*

$$C = \frac{1}{n} \sum_x C_x$$

over cost functions C_x for individual training examples, x.

This is so it allows us to compute the gradient (with respect to weights and biases) for a single training example, and run Gradient Descent.

2: *The cost function C must not be dependent on any activation values of a neural network besides the output values a^L.*

Technically a cost function can be dependent on any a^i_j or z^i_j. We just make this restriction so we can backpropagte, because the equation for finding the gradient of the last layer is the only one that is dependent on the cost function (the rest are dependent on the next layer). If the cost function is dependent on other activation layers besides the output one, backpropagation will be invalid because the idea of "trickling backwards" no longer works.

Also, activation functions are required to have an output $0 \leq a^L_j \leq 1$ for all j. Thus these cost functions need to only be defined within that range (for example, $\sqrt{a^L_j}$ is valid since we are guaranteed $a^L_j \geq 0$).

machine-learning neural-networks

edited May 31 '15 at 19:57 — community wiki Phylliida

1 This is a Q&A site, and the format of this post doesn't really fit that. You should probably put the majority of the content in an answer, and leave just the question (e.g. What is a list of cost functions used in NNs?). – Roger Fan May 31 '15 at 19:47

OK, I will do that, thanks. – Phylliida May 31 '15 at 19:47

Okay, is that better? I think the definitions are important otherwise the answers become vague for those that aren't familiar with the terminology the writer uses. – Phylliida May 31 '15 at 19:49

But what if a different answer uses different notation or terminology? – Roger Fan May 31 '15 at 19:52

2 The idea is that everyone uses the same terminology here, and that if it's different we convert it to this, so the answers "fit" with each other. But I suppose I could remove that piece if you don't think it's helpful. – Phylliida May 31 '15 at 19:54

2 Answers

Here are those I understand so far. Most of these work best when given values between 0 and 1.

Quadratic cost

Also known as *mean squared error*, *maximum likelihood*, and *sum squared error*, this is defined as:

$$C_{MST}(W, B, S^r, E^r) = 0.5 \sum_j (a^L_j - E^r_j)^2$$

The gradient of this cost function with respect to the output of a neural network and some sample r is:

$$\nabla_a C_{MST} = (a^L - E^r)$$

Cross-entropy cost

Also known as *Bernoulli negative log-likelihood* and *Binary Cross-Entropy*

$$C_{CE}(W, B, S^r, E^r) = -\sum_j [E_j^r \ln a_j^L + (1 - E_j^r) \ln (1 - a_j^L)]$$

The gradient of this cost function with respect to the output of a neural network and some sample r is:

$$\nabla_a C_{CE} = \frac{(a^L - E^r)}{(1 - a^L)(a^L)}$$

Exponentional cost

This requires choosing some parameter τ that you think will give you the behavior you want. Typically you'll just need to play with this until things work good.

$$C_{EXP}(W, B, S^r, E^r) = \tau \exp(\frac{1}{\tau} \sum_j (a_j^L - E_j^r)^2)$$

where $\exp(x)$ is simply shorthand for e^x.

The gradient of this cost function with respect to the output of a neural network and some sample r is:

$$\nabla_a C = \frac{2}{\tau}(a^L - E^r) C_{EXP}(W, B, S^r, E^r)$$

I could rewrite out C_{EXP}, but that seems redundant. Point is the gradient computes a vector and then multiplies it by C_{EXP}.

Hellinger distance

$$C_{HD}(W, B, S^r, E^r) = \frac{1}{\sqrt{2}} \sum_j (\sqrt{a_j^L} - \sqrt{E_j^r})^2$$

You can find more about this here. This needs to have positive values, and ideally values between 0 and 1. The same is true for the following divergences.

The gradient of this cost function with respect to the output of a neural network and some sample r is:

$$\nabla_a C = \frac{\sqrt{a^L} - \sqrt{E^r}}{\sqrt{2}\sqrt{a^L}}$$

Kullback–Leibler divergence

Also known as *Information Divergence*, *Information Gain*, *Relative entropy*, *KLIC*, or *KL Divergence* (See here).

Kullback–Leibler divergence is typically denoted

$$D_{\mathrm{KL}}(P \| Q) = \sum_i P(i) \ln \frac{P(i)}{Q(i)}$$

,

where $D_{\mathrm{KL}}(P \| Q)$ is a measure of the information lost when Q is used to approximate P. Thus we want to set $P = E^i$ and $Q = a^L$, because we want to measure how much information is lost when we use a_j^i to approximate E_j^i. This gives us

$$C_{KL}(W, B, S^r, E^r) = \sum_j E_j^r \log \frac{E_j^r}{a_j^L}$$

The other divergences here use this same idea of setting $P = E^i$ and $Q = a^L$.

The gradient of this cost function with respect to the output of a neural network and some sample r is:

$$\nabla_a C = \frac{E^r}{a^L}$$

Generalized Kullback–Leibler divergence

From here.

$$C_{GKL}(W, B, S^r, E^r) = \sum_j E_j^r \log \frac{E_j^r}{a_j^L} - \sum_j (E_j^r) + \sum_j (a_j^L)$$

The gradient of this cost function with respect to the output of a neural network and some sample r is:

$$\nabla_a C = \frac{E^r + a^L}{a^L}$$

Itakura–Saito distance

Also from here.

$$C_{GKL}(W, B, S^r, E^r) = \sum_j \left(\frac{E_j^r}{a_j^L} - \log \frac{E_j^r}{a_j^L} - 1 \right)$$

The gradient of this cost function with respect to the output of a neural network and some sample r is:

$$\nabla_a C = \frac{E^r + \left(a^L\right)^2}{\left(a^L\right)^2}$$

Where $\left(\left(a^L\right)^2 \right)_j = a_j^L \cdot a_j^L$. In other words, $\left(a^L\right)^2$ is simply equal to squaring each element of a^L.

edited Nov 8 '16 at 18:40

community wiki
4 revs
Phylliida

Thanks for sharing, you can also consider these: github.com/torch/nn/blob/master/doc/criterion.md – Yannis Assael May 31 '15 at 19:52

2 you have a small mistake in the denominator of the cross-entropy derivative, it should be `a*(1-a)` not `a*(1+a)` – Amro May 16 '16 at 17:51

It would also be cool to show the pinball loss function to minimize error quantiles rather than average error. Very used in decision support systems. – Ricardo Cruz Nov 10 '16 at 13:56

Don't have the reputation to comment, but there are sign errors in those last 3 gradients.

In the KL divergence,

$$\begin{aligned} C &= \sum_j E_j \log(E_j/a_j) \\ &= \sum_j E_j \log(E_j) - E_j \log(a_j) \end{aligned}$$

$$\begin{aligned} dC &= -\sum_j E_j \; d\log(a_j) \\ &= -\sum_j (E_j/a_j)\, da_j \end{aligned}$$

$$\nabla_a C = \frac{-E}{a}$$

This same sign error appears in the Generalized KL divergence.

In the Itakura-Saito distance ,

$$
\begin{aligned}
C &= \sum_j (E_j/a_j) - \log(E_j/a_j) - 1 \\
&= \sum_j (E_j/a_j) - \log(E_j) + \log(a_j) - 1
\end{aligned}
$$

$$
\begin{aligned}
dC &= \sum_j (-E_j/a_j^2)\, da_j + d\log(a_j) \\
&= \sum_j (1/a_j)\, da_j - (E_j/a_j^2)\, da_j \\
&= \sum_j (a_j - E_j)/a_j^2 \;\; da_j
\end{aligned}
$$

$$
\nabla_a C = \frac{a - E}{(a)^2}
$$

answered Nov 24 '16 at 18:31 community wiki
frank