Introduction to Neural Networks and Deep Learning

Activation and Loss Functions in Deep Learning

Andres Mendez-Vazquez

July 6, 2025

Outline

More advanced activation function

Piecewise-Linear Function

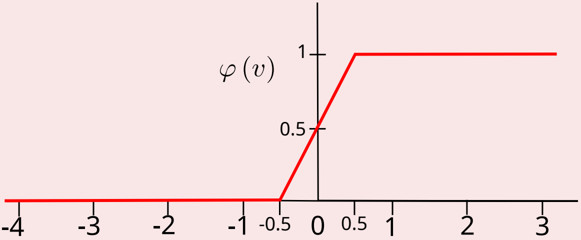
$$\varphi(v) = \begin{cases} 1 & \text{if } v_k \geq \frac{1}{2} \\ v & \text{if } -\frac{1}{2} < v_k < \frac{1}{2} \\ 0 & \text{if } v \leq -\frac{1}{2} \end{cases} \tag{1}$$

More advanced activation function

Piecewise-Linear Function

$$\varphi(v) = \begin{cases} 1 & \text{if } v_k \geq \frac{1}{2} \\ v & \text{if } -\frac{1}{2} < v_k < \frac{1}{2} \\ 0 & \text{if } v \leq -\frac{1}{2} \end{cases} \quad (1)$$

Example



Remarks

Notes about Piecewise-Linear function

The amplification factor inside the linear region of operation is assumed to be unity.

Remarks

Notes about Piecewise-Linear function

The amplification factor inside the linear region of operation is assumed to be unity.

Special Cases

- A linear combiner arises if the linear region of operation is maintained without running into saturation.

Remarks

Notes about Piecewise-Linear function

The amplification factor inside the linear region of operation is assumed to be unity.

Special Cases

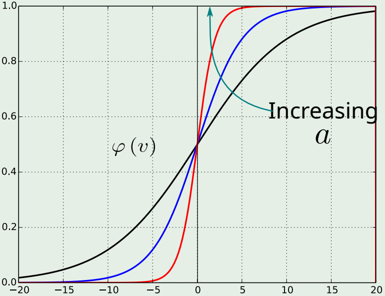
- A linear combiner arises if the linear region of operation is maintained without running into saturation.
- The piecewise-linear function reduces to a threshold function if the amplification factor of the linear region is made infinitely large.

A better choice!!!

Sigmoid/Logistic function

$$\varphi(v) = \frac{1}{1 + \exp\{-av\}} \tag{2}$$

Where a is a slope parameter.



Outline

The Problem of the Vanishing Gradient

When using a non-linearity

- However, there is a drawback when using Back-Propagation (As we saw in Machine Learning) under a sigmoid function

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

The Problem of the Vanishing Gradient

When using a non-linearity

- However, there is a drawback when using Back-Propagation (As we saw in Machine Learning) under a sigmoid function

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

Because if we imagine a Deep Neural Network as a series of layer functions f_i

$$y(A) = f_t \circ f_{t-1} \circ \cdots \circ f_2 \circ f_1 (A)$$

- With f_t is the last layer.

Then, using the Chain Rule

Example a two layer network

$$f\left(\boldsymbol{x}\right) = \sigma \circ B \circ \sigma \circ A\left(\boldsymbol{x}\right)$$

Then, using the Chain Rule

Example a two layer network

$$f(\boldsymbol{x}) = \sigma \circ B \circ \sigma \circ A(\boldsymbol{x})$$

Using the Chain Rule on Derivatives

$$\frac{\partial f(x)}{\partial x} = \frac{\partial \sigma(y_3)}{\partial y_3} \times \frac{\partial B(y_2)}{\partial y_2} \times \frac{\partial \sigma(y_1)}{\partial y_1} \times \frac{\partial A(\boldsymbol{x})}{\partial \boldsymbol{x}}$$

- where $y_3 = B \circ \sigma \circ A(\boldsymbol{x})$, $y_2 = \sigma \circ A(\boldsymbol{x})$ and $y_1 = A(\boldsymbol{x})$

Therefore

Given the commutativity of the product

- You could put together the derivative of the sigmoid's

$$f\left(x\right)=\frac{ds\left(x\right)}{dx}=\frac{e^{-x}}{\left(1+e^{-x}\right)^{2}}$$

Therefore

Given the commutativity of the product

- You could put together the derivative of the sigmoid's

$$f\left(x\right)=\frac{ds\left(x\right)}{dx}=\frac{e^{-x}}{\left(1+e^{-x}\right)^{2}}$$

Therefore, deriving again

$$\frac{df\left(x\right)}{dx}=-\frac{e^{-x}}{\left(1+e^{-x}\right)^{2}}+\frac{2\left(e^{-x}\right)^{2}}{\left(1+e^{-x}\right)^{3}}$$

Therefore

Given the commutativity of the product

- You could put together the derivative of the sigmoid's

$$f(x) = \frac{ds(x)}{dx} = \frac{e^{-x}}{(1+e^{-x})^2}$$

Therefore, deriving again

$$\frac{df(x)}{dx} = -\frac{e^{-x}}{(1+e^{-x})^2} + \frac{2(e^{-x})^2}{(1+e^{-x})^3}$$

After making $\frac{df(x)}{dx} = 0$

- We have the maximum is at $x = 0$

Therefore

The maximum for the derivative of the sigmoid

- $f(0) = 0.25$

Therefore

The maximum for the derivative of the sigmoid

- $f(0) = 0.25$

Therefore, Given a **Deep** Convolutional Network

- We could finish with

$$\lim_{k \to \infty} \left(\frac{ds(x)}{dx} \right)^k = \lim_{k \to \infty} (0.25)^k \to 0$$

Therefore

The maximum for the derivative of the sigmoid

- $f(0) = 0.25$

Therefore, Given a **Deep** Convolutional Network

- We could finish with

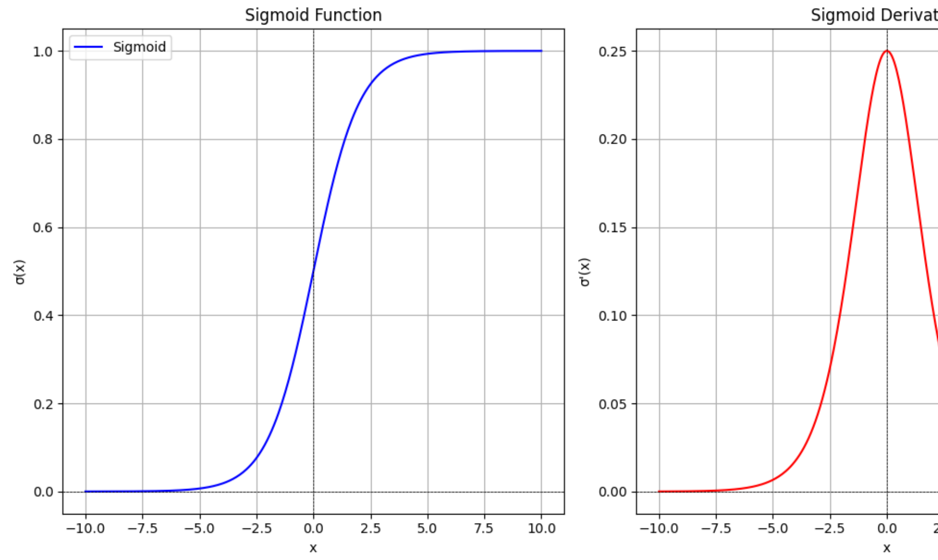
$$\lim_{k\to\infty}\left(\frac{ds(x)}{dx}\right)^k = \lim_{k\to\infty}(0.25)^k \to 0$$

A Vanishing Derivative or Vanishing Gradient

- Making quite difficult to do train a deeper network using this activation function for Deep Learning and even in Shallow Learning

Example

We have the following



Outline

Thus

The need to introduce a new function

$$f(x) = x^{+} = \max(0, x)$$

Thus

The need to introduce a new function

$$f\left(x\right) = x^{+} = \max\left(0, x\right)$$

It is called ReLU or Rectifier

With a smooth approximation (Softplus function)

$$f\left(x\right) = \frac{\ln\left(1 + e^{kx}\right)}{k}$$

Outline

As we can see

Pluses

- A clear benefit of ReLU is that both the function itself and its derivatives are easy to implement and computationally inexpensive.
- ReLU is infinitely many times differentiable at $x \in \mathbb{R} - \{0\}$

As we can see

Pluses

- A clear benefit of ReLU is that both the function itself and its derivatives are easy to implement and computationally inexpensive.
- ReLU is infinitely many times differentiable at $x \in \mathbb{R} - \{0\}$

Actually, we have that at the first and second derivative

$$\frac{dReLu\left(x\right)}{dx} = \begin{cases} 1 & x \in (0, \infty) \\ 0 & x \in (-\infty, 0) \end{cases} \qquad \frac{d^2 ReLu\left(x\right)}{dx^2} = 0 \ x \in \mathbb{R} - \{0\}$$

Additionally

At $x > 0$ we have basically the identity

- Therefore, the gradient pass through it with its full force when positive.... thus forget about controlling the

Additionally

At $x > 0$ we have basically the identity

- Therefore, the gradient pass through it with its full force when positive.... thus forget about controlling the

Another problem, The Dying ReLU

- Neurons with negative inputs (e.g., due to poor initialization or large gradients) may output 0 and stop learning.

Additionally

At $x > 0$ we have basically the identity

- Therefore, the gradient pass through it with its full force when positive.... thus forget about controlling the

Another problem, The Dying ReLU

- Neurons with negative inputs (e.g., due to poor initialization or large gradients) may output 0 and stop learning.

Therefore, we need something better

- Leaky ReLU
- PReLU

Outline

Leaky ReLU

Purpose

- To address the "dying ReLU" problem by allowing small gradients for negative inputs.

We have the following definition for a small α between $(0, 1)$

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{if } x \leq 0 \end{cases}$$

Leaky ReLU

Purpose

- To address the "dying ReLU" problem by allowing small gradients for negative inputs.

We have the following definition for a small α between $(0, 1)$

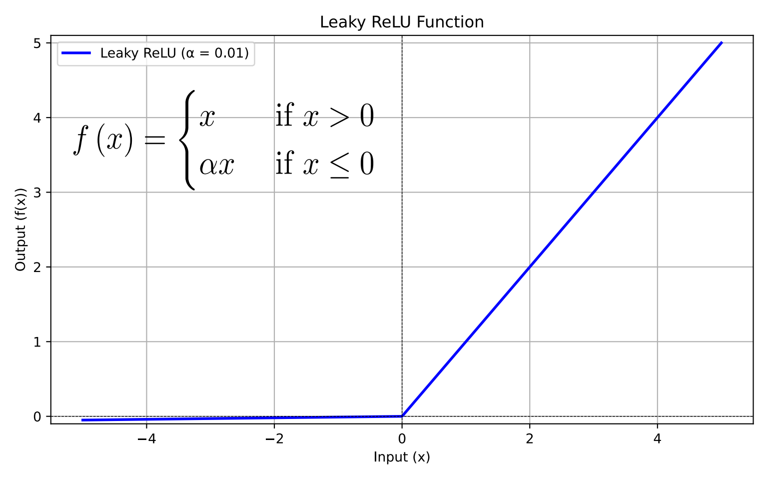
$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{if } x \leq 0 \end{cases}$$

Another Notation

$$LReLU(x) = \max(0, x) + \alpha * \min(0, x)$$

Example

The Leaky ReLU



Remarks

Advantages

- Prevents neurons from "dying" by allowing non-zero gradients for negative inputs.

Remarks

Advantages

- Prevents neurons from "dying" by allowing non-zero gradients for negative inputs.

Disadvantages

- The slope α is fixed, which may not be optimal for all tasks.

Outline

Parametric ReLU

They exposed the parameter α to the backpropagation training

- Basically the same function

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{if } x \leq 0 \end{cases}$$

Parametric ReLU

They exposed the parameter α to the backpropagation training

- Basically the same function

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{if } x \leq 0 \end{cases}$$

Therefore

- More flexible than the Leaky ReLU

Outline

Exponential Linear Units (ELUs)

We have the following paper

- Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs) [1]

Exponential Linear Units (ELUs)

We have the following paper

- Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs) [1]

They proved two theorems to talk about the Natural Gradient

- They observed that when neurons have a non-zero weight they correlate between layer units slow down the learning

Exponential Linear Units (ELUs)

We have the following paper

- Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs) [1]

They proved two theorems to talk about the Natural Gradient

- They observed that when neurons have a non-zero weight they correlate between layer units slow down the learning

They proved that the use of the Natural Gradient avoids this shifting

- Then the normal gradient will get near to the natural gradient speeding up the training

Outline

As always the KL Divergence

Consider a second-order Taylor approximation to the KL divergence around θ_t

- Assuming $KL\left(\theta, \theta_t\right) = KL\left(p_\theta\left(x\right) | p_{\theta_t}\left(x\right)\right)$

As always the KL Divergence

Consider a second-order Taylor approximation to the KL divergence around θ_t

- Assuming $KL\left(\theta,\theta_t\right) = KL\left(p_{\theta}\left(x\right)|p_{\theta_t}\left(x\right)\right)$

We have that with H_t is the Hessian of the $KL\left(\theta,\theta_t\right)$ at θ_t

$$
\begin{aligned}
KL\left(\theta,\theta_t\right) =& KL\left(\theta_t,\theta_t\right) + \left(\nabla_{\theta} KL\left(\theta,\theta_t\right)|_{\theta=\theta_t}\right)^T\left(\theta-\theta_t\right) + \cdots \\
& \frac{1}{2}\left(\theta-\theta_t\right)^T H_t\left(\theta-\theta_t\right)
\end{aligned}
$$

Therefore

As you can imagine $KL\left(\theta_t, \theta_t\right) = 0$

- The second term

$$
\begin{aligned}
\nabla_\theta KL\left(\theta, \theta_t\right) =& E_{p(x|\theta)}\left[\log \frac{p\left(x|\theta\right)}{p\left(x|\theta_t\right)}\right] \\
=& E_{p(x|\theta)}\left[\nabla_\theta \log \frac{p\left(x|\theta\right)}{p\left(x|\theta_t\right)}\right] \\
=& E_{p(x|\theta)}\left[\nabla_\theta \log p\left(x|\theta\right)\right] = E_{p(x|\theta)}\left[\frac{1}{p\left(x|\theta\right)} \nabla_\theta p\left(x|\theta\right)\right]
\end{aligned}
$$

In this way, we have that

We have that

$$E_{p(x|\theta)}\left[\frac{1}{p\left(x|\theta\right)}\nabla_{\theta}p\left(x|\theta\right)\right]=\nabla_{\theta}\int p\left(x|\theta\right)=\nabla_{\theta}1=0$$

In this way, we have that

We have that

$$E_{p(x|\theta)}\left[\frac{1}{p\left(x|\theta\right)}\nabla_{\theta}p\left(x|\theta\right)\right]=\nabla_{\theta}\int p\left(x|\theta\right)=\nabla_{\theta}1=0$$

Now, What is the second derivative of the $KL\left(\theta,\theta_t\right)$

$$\begin{aligned}\nabla^2_{\theta_t}KL\left(\theta,\theta_t\right)&=-\nabla_{\theta_t}\int p\left(x|\theta\right)\nabla_{\theta_t}\log p\left(x|\theta_t\right)dx\\&=-\int p\left(x|\theta\right)\nabla^2_{\theta_t}\log p\left(x|\theta_t\right)dx\\&=-E\left[\nabla^2_{\theta_t}\log p\left(x|\theta_t\right)\right]\\&=-E\left[H_{\log p\left(x|\theta_t\right)}\right]=F\end{aligned}$$

In this way, we have

We have using the previous equations

$$KL\left(\theta,\theta_t\right)=\frac{1}{2}\left(\theta-\theta_t\right)^T H_t\left(\theta-\theta_t\right)$$

In this way, we have

We have using the previous equations

$$KL(\theta, \theta_t) = \frac{1}{2}(\theta - \theta_t)^T H_t (\theta - \theta_t)$$

Now, using the following notation $\theta - \theta_t = \delta$

- We can define the following gradient descent

$$\delta^* = \arg_{\delta \text{ s.t. }} \min_{KL(\theta,\theta_t)=c} \mathcal{L}(\theta + \delta)$$

In this way, we have

We have using the previous equations

$$KL\left(\theta,\theta_t\right) = \frac{1}{2}\left(\theta-\theta_t\right)^T H_t\left(\theta-\theta_t\right)$$

Now, using the following notation $\theta-\theta_t=\delta$

- We can define the following gradient descent

$$\delta^* = \arg\min_{\delta \text{ s.t. } KL(\theta,\theta_t)=c} \mathcal{L}\left(\theta+\delta\right)$$

With update rule for the new gradient descent

$$\theta_{k+1} = \theta_k + \delta^*$$

Lagrangian

the Lagrangian would be

$$\delta^* = \arg\min_{\delta} \mathcal{L}(\theta + \delta) + \lambda \left[KL(\theta, \theta_t) - c\right]$$

Lagrangian

the Lagrangian would be

$$\delta^{*} = \arg\min_{\delta} \mathcal{L}\left(\theta + \delta\right) + \lambda\left[KL\left(\theta, \theta_{t}\right) - c\right]$$

Using the first and second Taylor approximation, we get

$$\delta^{*} = \arg\min_{\delta} \mathcal{L}\left(\theta\right) + \nabla_{\theta}^{T}\mathcal{L}\left(\theta\right)\delta + \lambda\left[\frac{1}{2}\delta^{T}H_{t}\delta - c\right]$$

Lagrangian

the Lagrangian would be

$$\delta^{*}=\arg\min_{\delta}\mathcal{L}\left(\theta+\delta\right)+\lambda\left[KL\left(\theta,\theta_{t}\right)-c\right]$$

Using the first and second Taylor approximation, we get

$$\delta^{*}=\arg\min_{\delta}\mathcal{L}\left(\theta\right)+\nabla_{\theta}^{T}\mathcal{L}\left(\theta\right)\delta+\lambda\left[\frac{1}{2}\delta^{T}H_{t}\delta-c\right]$$

Derive against δ and make it equal to zero

$$\frac{\partial}{\partial\delta}\left[\mathcal{L}\left(\theta\right)+\nabla_{\theta}^{T}\mathcal{L}\left(\theta\right)\delta+\lambda\left[\frac{1}{2}\delta^{T}H_{t}\delta-c\right]\right]$$

We have then

Something Notable

$$\nabla_\theta \mathcal{L}\left(\theta\right) + \lambda F \delta = 0$$

We have then

Something Notable

$$\nabla_{\theta}\mathcal{L}\left(\theta\right) + \lambda F\delta = 0$$

In this way, we have that

$$\delta^{*} \propto F^{-1}\nabla_{\theta}\mathcal{L}\left(\theta\right)$$

Outline

Here, we know that every neurons has a bias

Here, we have training data $X = (\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_n)$ with $\boldsymbol{x}_i = \left(\boldsymbol{z}_i^T, y_i\right)^T \in \mathbb{R}^{d+1}$

- We define the natural gradient on the Loss function as previously

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_{t+1} - \eta F^{-1} \nabla_\theta \mathcal{L}(\theta)$$

  - Where the $\mathcal{L}(p(\boldsymbol{z}|\boldsymbol{w}))$ is the loss function for a model $p(\boldsymbol{z}|\boldsymbol{w})$

Here, we know that every neurons has a bias

Here, we have training data $X = (\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_n)$ with $\boldsymbol{x}_i = \left(\boldsymbol{z}_i^T, y_i\right)^T \in \mathbb{R}^{d+1}$

- We define the natural gradient on the Loss function as previously

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_{t+1} - \eta F^{-1} \nabla_\theta \mathcal{L}(\theta)$$

  - Where the $\mathcal{L}(p(\boldsymbol{z}|\boldsymbol{w}))$ is the loss function for a model $p(\boldsymbol{z}|\boldsymbol{w})$

Thus, we have that the gradient to going into the neuron i

$$a_i = f\left(\underbrace{\sum w_{ij} a_j}_{net}\right)$$

Therefore

If we want to compute the Fisher information matrix

$$\frac{\partial}{\partial w_j} \ln p\left(\boldsymbol{z}|\boldsymbol{w}\right)$$

Therefore

If we want to compute the Fisher information matrix

$$\frac{\partial}{\partial w_j} \ln p\left(\boldsymbol{z}|\boldsymbol{w}\right)$$

We can use the backpropagation to obtain a version of it

$$\delta = \frac{\partial}{\partial net} \ln p\left(\boldsymbol{z}|\boldsymbol{w}\right)$$

Therefore

If we want to compute the Fisher information matrix

$$\frac{\partial}{\partial w_j}\ln p\left(\boldsymbol{z}|\boldsymbol{w}\right)$$

We can use the backpropagation to obtain a version of it

$$\delta=\frac{\partial}{\partial net}\ln p\left(\boldsymbol{z}|\boldsymbol{w}\right)$$

Thus the classic chain rule

$$\frac{\partial}{\partial w_j}\ln p\left(\boldsymbol{z}|\boldsymbol{w}\right)=\frac{\partial\ln p\left(\boldsymbol{z}|\boldsymbol{w}\right)}{\partial net}\times\frac{\partial net}{\partial w_j}$$

Therefore

The Unit Fisher information matrix looks like

$$[F(\boldsymbol{w})]_{kj} = E_{p(\boldsymbol{z}|\boldsymbol{w})}\left[\frac{\partial}{\partial w_k}\ln p(\boldsymbol{z}|\boldsymbol{w}) \times \frac{\partial}{\partial w_j}\ln p(\boldsymbol{z}|\boldsymbol{w})\right] = E_{p(\boldsymbol{z}|\boldsymbol{w})}\left(\delta^2 a_k a_j\right)$$

Therefore

The Unit Fisher information matrix looks like

$$[F(\boldsymbol{w})]_{kj} = E_{p(\boldsymbol{z}|\boldsymbol{w})}\left[\frac{\partial}{\partial w_k}\ln p(\boldsymbol{z}|\boldsymbol{w}) \times \frac{\partial}{\partial w_j}\ln p(\boldsymbol{z}|\boldsymbol{w})\right] = E_{p(\boldsymbol{z}|\boldsymbol{w})}\left(\delta^2 a_k a_j\right)$$

Basically δ^2 works as at increasing or decreasing the probability of $\boldsymbol{z}$

- Why not to use a probability for $\boldsymbol{z}$

$$q(\boldsymbol{z}) = \delta^2(\boldsymbol{z})\, p(\boldsymbol{z})\, E^{-1}_{p(\boldsymbol{z})}\left(\delta^2\right)$$

Therefore

The Unit Fisher information matrix looks like

$$[F(\boldsymbol{w})]_{kj} = E_{p(\boldsymbol{z}|\boldsymbol{w})}\left[\frac{\partial}{\partial w_k}\ln p(\boldsymbol{z}|\boldsymbol{w}) \times \frac{\partial}{\partial w_j}\ln p(\boldsymbol{z}|\boldsymbol{w})\right] = E_{p(\boldsymbol{z}|\boldsymbol{w})}\left(\delta^2 a_k a_j\right)$$

Basically δ^2 works as at increasing or decreasing the probability of $\boldsymbol{z}$

- Why not to use a probability for $\boldsymbol{z}$

$$q(\boldsymbol{z}) = \delta^2(\boldsymbol{z})\, p(\boldsymbol{z})\, E^{-1}_{p(\boldsymbol{z})}\left(\delta^2\right)$$

Thus, we can express the Fisher Matrix as second central moments

$$[F(\boldsymbol{w})]_{kj} = E_{p(\boldsymbol{z})}\left(\delta^2\right) E_{q(\boldsymbol{z})}\left(a_k a_j\right)$$

Therefore, it is possible to prove that

The Unit Gradient Descent

$$\begin{pmatrix} \Delta \boldsymbol{w} \\ \Delta w_0 \end{pmatrix} = \begin{pmatrix} A^{-1}\left(\mathbf{g} - \Delta w_0 \mathbf{b}\right) \\ s\left(\mathbf{g}_0 - \mathbf{b}^T A^{-1} \mathbf{g}\right) \end{pmatrix}$$

Therefore, it is possible to prove that

The Unit Gradient Descent

$$\begin{pmatrix} \Delta \boldsymbol{w} \\ \Delta w_0 \end{pmatrix} = \begin{pmatrix} A^{-1}\left(\mathbf{g} - \Delta w_0 \mathbf{b}\right) \\ s\left(\mathbf{g}_0 - \mathbf{b}^T A^{-1} \mathbf{g}\right) \end{pmatrix}$$

Here we have that

- $\mathbf{b} = \left[F\left(\boldsymbol{w}\right)\right]_0$ the bias weight and not only that
  $\mathbf{b} = E_{p(\boldsymbol{z})}\left(\delta^2 \boldsymbol{a}\right) = Cov_{p(\boldsymbol{z})}\left(\delta^2, \boldsymbol{a}\right) + E_{p(\boldsymbol{z})}\left(\boldsymbol{a}\right) E_{p(\boldsymbol{z})}\left(\delta^2\right)$
- $A = \left[F\left(\boldsymbol{w}\right)\right]_{-0,-0} = E_{p(\boldsymbol{z})}\left(\delta^2\right) E_{a(\boldsymbol{z})}\left(\boldsymbol{a}\boldsymbol{a}^T\right)$ Fisher without row and column 0
- $s = E^{-1}_{p(\boldsymbol{z})}\left(\delta^2\right)\left[1 + E^T_{p(\boldsymbol{z})}\left(\boldsymbol{a}\right) Var^{-1}_{q(\boldsymbol{z})} E_{q(\boldsymbol{z})}\left(\boldsymbol{a}\right)\right]$

Finally

The bias shift correction by the unit natural gradient is equivalent to

- An additive correction of the incoming mean by $-kE_{q(\boldsymbol{z})}\left(\boldsymbol{a}\right)$
- An a multiplicative correction of the bias unit by

$$k = 1 + \left[E_{q(\boldsymbol{z})}\left(\boldsymbol{a}\right) - E_{p(\boldsymbol{z})}\left(\boldsymbol{a}\right)\right]^{T} Var_{q(\boldsymbol{z})}^{-1} E_{q(\boldsymbol{z})}\left(\boldsymbol{a}\right)$$

Finally

The bias shift correction by the unit natural gradient is equivalent to

- An additive correction of the incoming mean by $-kE_{q(\boldsymbol{z})}(\boldsymbol{a})$
- An a multiplicative correction of the bias unit by

$$k = 1 + \left[E_{q(\boldsymbol{z})}(\boldsymbol{a}) - E_{p(\boldsymbol{z})}(\boldsymbol{a})\right]^T Var_{q(\boldsymbol{z})}^{-1} E_{q(\boldsymbol{z})}(\boldsymbol{a})$$

Therefore k increases with the length $E_{q(\boldsymbol{z})}(\boldsymbol{a})$

- Not only that

$$E_{q(\boldsymbol{z})}(\boldsymbol{a}) = E_{p(\boldsymbol{z})}(\boldsymbol{a}) + E_{p(\boldsymbol{z})}^{-1}\left(\delta^2\right) Cov_{p(\boldsymbol{z})}\left(\delta^2, \boldsymbol{a}\right)$$

Finally

The bias shift correction by the unit natural gradient is equivalent to

- An additive correction of the incoming mean by $-kE_{q(\boldsymbol{z})}\left(\boldsymbol{a}\right)$
- An a multiplicative correction of the bias unit by

$$k = 1 + \left[E_{q(\boldsymbol{z})}\left(\boldsymbol{a}\right) - E_{p(\boldsymbol{z})}\left(\boldsymbol{a}\right)\right]^{T} Var_{q(\boldsymbol{z})}^{-1} E_{q(\boldsymbol{z})}\left(\boldsymbol{a}\right)$$

Therefore k increases with the length $E_{q(\boldsymbol{z})}\left(\boldsymbol{a}\right)$

- Not only that

$$E_{q(\boldsymbol{z})}\left(\boldsymbol{a}\right) = E_{p(\boldsymbol{z})}\left(\boldsymbol{a}\right) + E_{p(\boldsymbol{z})}^{-1}\left(\delta^{2}\right) Cov_{p(\boldsymbol{z})}\left(\delta^{2}, \boldsymbol{a}\right)$$

In general

- In general, smaller positive $E_{p(\boldsymbol{z})}\left(\boldsymbol{a}\right)$ lead to smaller positive $E_{q(\boldsymbol{z})}\left(\boldsymbol{a}\right)$, therefore to smaller corrections.

Therefore

Something Notable

- The unit natural gradient corrects the bias shift of unit i via the interactions of incoming units with the bias unit to ensure efficient learning

Therefore

Something Notable

- The unit natural gradient corrects the bias shift of unit i via the interactions of incoming units with the bias unit to ensure efficient learning

Meaning

- This correction is equivalent to shifting the mean activation's of the incoming units toward zero and scaling up the bias unit.

However Fisher is quite expensive to calculate

Therefore, two proposal are done

- Activation of incoming units can be centered at zero or
- Activation functions with negative values can be used.

However Fisher is quite expensive to calculate

Therefore, two proposal are done

- Activation of incoming units can be centered at zero or
- Activation functions with negative values can be used.

Exponential Linear Units (ELU's)

- The exponential linear unit (ELU) with $0 < \alpha$ is

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha \left(exp(x) - 1 \right) & \text{if } x \leq 0 \end{cases}$$

We have then the derivative is

We have for $x > 0$

$$f'(x) = 1$$

We have then the derivative is

We have for $x > 0$

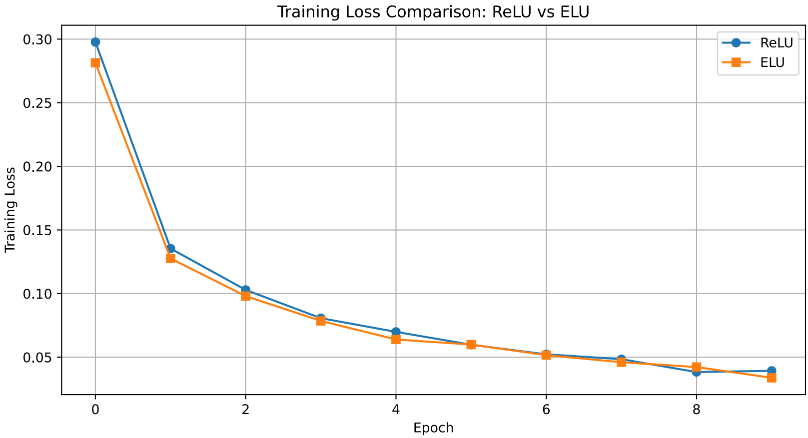
$$f'(x) = 1$$

For $x \leq 0$

$$f'(x) = \alpha exp(x)$$

Example

With a MLP using ELU vs ReLU



However

With a CNN using ELU vs ReLU

Outline

Scaled Exponential Linear Unit

The SELU activation function is given by

$$selu\left(x\right) = \lambda \begin{cases} x & \text{if } x > 0 \\ \alpha\left(exp\left(x\right) - 1\right) & \text{if } x \leq 0 \end{cases}$$

Scaled Exponential Linear Unit

The SELU activation function is given by

$$selu(x) = \lambda \begin{cases} x & \text{if } x > 0 \\ \alpha \left(exp(x) - 1\right) & \text{if } x \leq 0 \end{cases}$$

It is an expansion on the ELU

- Advantages:
  - Ensures stable training without explicit normalization (e.g., batch normalization).
  - Theoretical guarantees for convergence in deep networks.

Scaled Exponential Linear Unit

The SELU activation function is given by

$$selu\left(x\right) = \lambda \begin{cases} x & \text{if } x > 0 \\ \alpha\left(exp\left(x\right) - 1\right) & \text{if } x \leq 0 \end{cases}$$

It is an expansion on the ELU

- Advantages:
  - Ensures stable training without explicit normalization (e.g., batch normalization).
  - Theoretical guarantees for convergence in deep networks.

Problems

- Requires specific initialization and network architectures to work effectively.

Initialization for SELU

For Convolutional Layers (Conv2d)

$$w \sim N\left(0, \left[\frac{1}{\sqrt{in\_channels \times kernel\_size^2}}\right]^2\right)$$

Initialization for SELU

For Convolutional Layers (Conv2d)

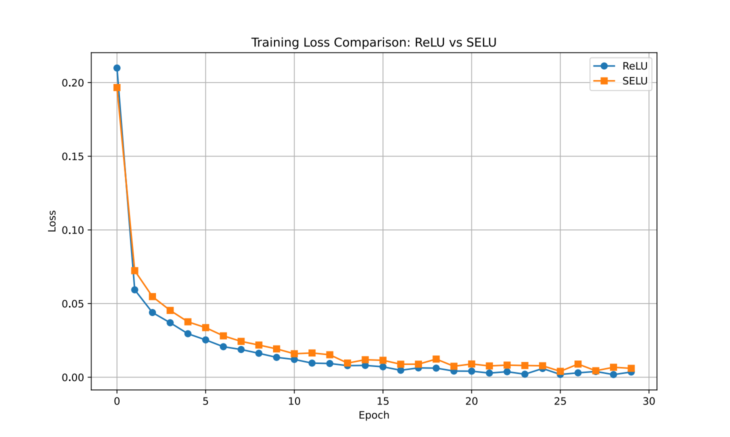
$$w \sim N\left(0, \left[\frac{1}{\sqrt{in\_channels \times kernel\_size^2}}\right]^2\right)$$

For Linear init

$$w \sim N\left(0, \left[\frac{1}{\sqrt{in\_features}}\right]^2\right)$$

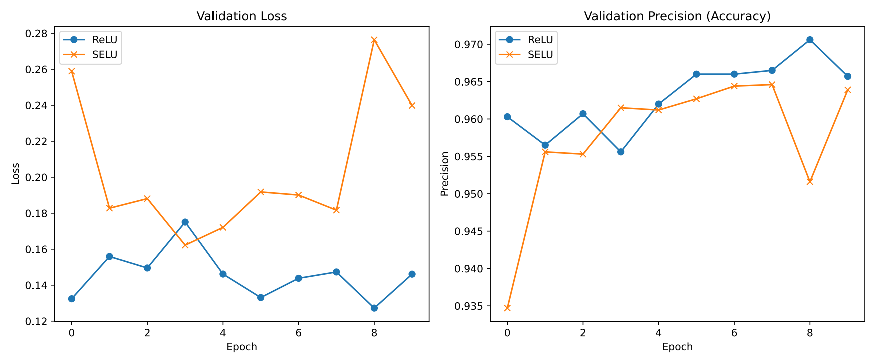
Example

With a CNN



Example

With a Dense



Outline

This is interesting

Combine the benefits of ReLU and sigmoid-like functions for smoother gradients.

$$swish_{\beta}(x) = x \times sigmoid(\beta x) = \frac{x}{1 + \exp\{\beta x\}}$$

This is interesting

Combine the benefits of ReLU and sigmoid-like functions for smoother gradients.

$$swish_{\beta}(x) = x \times sigmoid(\beta x) = \frac{x}{1 + \exp\{\beta x\}}$$

Advantages

- Smooth and non-monotonic, which can improve performance in deep networks.
- Used in Google's models (e.g., MobileNetV3).

This is interesting

Combine the benefits of ReLU and sigmoid-like functions for smoother gradients.

$$swish_{\beta}(x) = x \times sigmoid(\beta x) = \frac{x}{1 + \exp\{\beta x\}}$$

Advantages

- Smooth and non-monotonic, which can improve performance in deep networks.
- Used in Google's models (e.g., MobileNetV3).

Disadvantages

- Computationally more expensive than ReLU.

Outline

Gaussian Error Linear Units

Approximate the Gaussian error function for better performance in NLP tasks.

- $f(x) = x \times \Phi(x)$ where $\Phi(x)$ is the cumulative distribution function of the standard normal distribution

Gaussian Error Linear Units

Approximate the Gaussian error function for better performance in NLP tasks.

- $f(x) = x \times \Phi(x)$ where $\Phi(x)$ is the cumulative distribution function of the standard normal distribution

Advantages

- Used in transformer models (e.g., BERT, GPT-3) for improved performance.
- Smooth and differentiable.

Gaussian Error Linear Units

Approximate the Gaussian error function for better performance in NLP tasks.

- $f(x) = x \times \Phi(x)$ where $\Phi(x)$ is the cumulative distribution function of the standard normal distribution

Advantages

- Used in transformer models (e.g., BERT, GPT-3) for improved performance.
- Smooth and differentiable.

Disadvantages

- Computationally heavy due to the normal distribution approximation.

Outline

Adaptive ReLU

We want to

- Make the ReLU slope adaptive to input data.

$$f(x) = \max\{\alpha(x) \times x, x\}$$

Adaptive ReLU

We want to

- Make the ReLU slope adaptive to input data.

$$f(x) = \max\{\alpha(x) \times x, x\}$$

Advantages

- Highly flexible and data-dependent.
- Can outperform fixed-slope ReLU variants in some cases.

Adaptive ReLU

We want to

- Make the ReLU slope adaptive to input data.

$$f(x) = \max\{\alpha(x) \times x, x\}$$

Advantages

- Highly flexible and data-dependent.
- Can outperform fixed-slope ReLU variants in some cases.

Disadvantages

- Increases model complexity and training time.

Outline

Why Loss Functions?

Long ago the Perceptron showed many shortcomings

- The XOR problem could not be solved by the Perceptron
- The loss function was quite simple

$$y\left(i\right)=\sum_{i=1}^{m}w_{k}\left(i\right)x_{k}\left(i\right)$$

Why Loss Functions?

Long ago the Perceptron showed many shortcomings

- The XOR problem could not be solved by the Perceptron
- The loss function was quite simple

$$y\left(i\right)=\sum_{i=1}^{m}w_k\left(i\right)x_k\left(i\right)$$

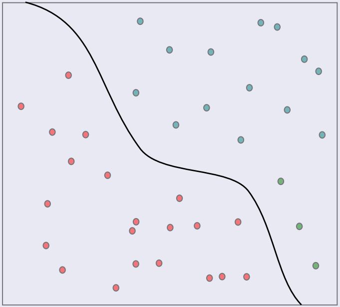
We want a better function for classification

- The classification case is harder because it is not obvious what loss function to use!!!

As we have found

Classification task started tweaking the Regression Method, $\sum_{i=1}^{N} L^2(x_i, y_i)$

- Which has serious disadvantages given that you are approximating a function where points could not exist...



Serious Disadvantages

You need to have dense classes with similar number of elements

- Basically, you are required to collect data under those two characteristics.

Serious Disadvantages

You need to have dense classes with similar number of elements

- Basically, you are required to collect data under those two characteristics.

Thus, we have a need to find better loss functions

- That reflect better the task of classification

Serious Disadvantages

You need to have dense classes with similar number of elements

- Basically, you are required to collect data under those two characteristics.

Thus, we have a need to find better loss functions

- That reflect better the task of classification

Way more explainable and adaptive

- Given the structures at the Deep Learners

Outline

At [2]

Several Loss Functions for Neural Networks have been studied

- Here, o is the output of the last layer in the deep learner and σ is the probability estimate

Name	Equation
L_1 Loss	$\mathcal{L}_1 = \lVert y - o \rVert_1$
L_2 Loss	$\mathcal{L}_2 = \lVert y - o \rVert_2^2$
Expectation Loss	$\lVert y - \sigma(o) \rVert_1$
Regularized expectation Loss	$\lVert y - \sigma(o) \rVert_1$
Chebyshev Loss	$\max_j \left\lvert \sigma(o)^{(j)} - y^{(j)} \right\rvert$
Hinge Loss	$\sum_j \max \left\{ 0, \frac{1}{2} - \hat{y}^{(j)} o^{(j)} \right\}$
Log Loss (Cross Entropy)	$-\sum_j y^{(j)} \log \sigma(o)^{(j)}$
Squared Log Loss	$-\sum_j \left[y^{(j)} \log \sigma(o)^{(j)} \right]^2$

For example, we have the following property

We have that for $\boldsymbol{y}_i \in \{0,1\}^K$ with $L_j(y_i) = 1$ if $i \neq j$ else 0, and $p_i = \widehat{p}(y_i|x_i)$

$$
\begin{aligned}
&= \frac{1}{N}\sum_i \sum_j \left| p_i^{(j)} + y_i^{(j)} p_i^{(j)} - y_i^{(j)} p_i^{(j)} - y_i^{(j)} \right| \\
&= \frac{1}{N}\sum_i \sum_j \left| y_i^{(j)} \left(p_i^{(j)} - 1\right) + p_i^{(j)} \left(1 - y_i^{(j)}\right) \right| \\
&= \frac{1}{N}\sum_i \sum_j \left[y_i^{(j)} \left(1 - p_i^{(j)}\right) + p_i^{(j)} \left(1 - y_i^{(j)}\right) \right] \\
&= \frac{1}{N}\sum_i \left[\sum_j y_i^{(j)} - 2\sum_j y_i^{(j)} p_i^{(j)} + \sum_j p_i^{(j)} \right] \\
&= \frac{1}{N}\sum_i \sum_j y_i^{(j)} - 2\frac{1}{N}\sum_i \sum_j y_i^{(j)} p_i^{(j)} + \frac{1}{N}\sum_i \sum_j p_i^{(j)} \\
&= 2 - 2\frac{1}{N}\sum_i \sum_j y_i^{(j)} p_i^{(j)} \approx -2E_{P(x,y)}\left[P\left(\widehat{l} = l | \widehat{l} \sim p_i, l \sim y_i\right) \right]
\end{aligned}
$$

For example, we have the following property

We have that for $\boldsymbol{y}_i \in \{0,1\}^K$ with $L_j(y_i) = 1$ if $i \neq j$ else 0, and $p_i = \widehat{p}(y_i|x_i)$

$$
\begin{aligned}
&= \frac{1}{N}\sum_i \sum_j \left| y_i^{(j)} \left(p_i^{(j)} - 1\right) + p_i^{(j)} \left(1 - y_i^{(j)}\right) \right| \\
&= \frac{1}{N}\sum_i \sum_j \left[y_i^{(j)} \left(1 - p_i^{(j)}\right) + p_i^{(j)} \left(1 - y_i^{(j)}\right) \right] \\
&= \frac{1}{N}\sum_i \left[\sum_j y_i^{(j)} - 2\sum_j y_i^{(j)} p_i^{(j)} + \sum_j p_i^{(j)} \right] \\
&= \frac{1}{N}\sum_i \sum_j y_i^{(j)} - 2\frac{1}{N}\sum_i \sum_j y_i^{(j)} p_i^{(j)} + \frac{1}{N}\sum_i \sum_j p_i^{(j)} \\
&= 2 - 2\frac{1}{N}\sum_i \sum_j y_i^{(j)} p_i^{(j)} \approx -2E_{P(x,y)}\left[P\left(\widehat{l} = l | \widehat{l} \sim p_i, l \sim y_i\right) \right]
\end{aligned}
$$

For example, we have the following property

We have that for $\boldsymbol{y}_i \in \{0,1\}^K$ with $L_j(y_i) = 1$ if $i \neq j$ else 0, and $p_i = \widehat{p}(y_i|x_i)$

$$
\begin{aligned}
&= \frac{1}{N}\sum_i \sum_j \left[y_i^{(j)}\left(1 - p_i^{(j)}\right) + p_i^{(j)}\left(1 - y_i^{(j)}\right)\right] \\
&= \frac{1}{N}\sum_i \left[\sum_j y_i^{(j)} - 2\sum_j y_i^{(j)} p_i^{(j)} + \sum_j p_i^{(j)} \right] \\
&= \frac{1}{N}\sum_i \sum_j y_i^{(j)} - 2\frac{1}{N}\sum_i \sum_j y_i^{(j)} p_i^{(j)} + \frac{1}{N}\sum_i \sum_j p_i^{(j)} \\
&= 2 - 2\frac{1}{N}\sum_i \sum_j y_i^{(j)} p_i^{(j)} \approx -2E_{P(x,y)}\left[P\left(\widehat{l} = l | \widehat{l} \sim p_i, l \sim y_i\right)\right]
\end{aligned}
$$

For example, we have the following property

We have that for $\boldsymbol{y}_i \in \{0,1\}^K$ with $L_j\left(y_i\right) = 1$ if $i \neq j$ else 0, and $p_i = \widehat{p}\left(y_i|x_i\right)$

$$= \frac{1}{N}\sum_i \left[\sum_j y_i^{(j)} - 2\sum_j y_i^{(j)} p_i^{(j)} + \sum_j p_i^{(j)}\right]$$

$$= \frac{1}{N}\sum_i \sum_j y_i^{(j)} - 2\frac{1}{N}\sum_i \sum_j y_i^{(j)} p_i^{(j)} + \frac{1}{N}\sum_i \sum_j p_i^{(j)}$$

$$= 2 - 2\frac{1}{N}\sum_i \sum_j y_i^{(j)} p_i^{(j)} \approx -2E_{P(x,y)}\left[P\left(\widehat{l} = l | \widehat{l} \sim p_i, l \sim y_i\right)\right]$$

For example, we have the following property

We have that for $\boldsymbol{y}_i \in \{0,1\}^K$ with $L_j(y_i) = 1$ if $i \neq j$ else 0, and $p_i = \widehat{p}(y_i|x_i)$

$$= \frac{1}{N}\sum_i \sum_j y_i^{(j)} - 2\frac{1}{N}\sum_i \sum_j y_i^{(j)} p_i^{(j)} + \frac{1}{N}\sum_i \sum_j p_i^{(j)}$$

$$= 2 - 2\frac{1}{N}\sum_i \sum_j y_i^{(j)} p_i^{(j)} \approx -2E_{P(x,y)}\left[P\left(\widehat{l} = l | \widehat{l} \sim p_i, l \sim y_i\right)\right]$$

For example, we have the following property

We have that for $\boldsymbol{y}_i \in \{0,1\}^K$ with $L_j(y_i) = 1$ if $i \neq j$ else 0, and $p_i = \widehat{p}(y_i|x_i)$

$$
\begin{aligned}
\mathcal{L}_1 &= \frac{1}{N}\sum_i \sum_j \left| p_i^{(j)} - y_i^{(j)} \right| \\
&= \frac{1}{N}\sum_i \sum_j \left| p_i^{(j)} + y_i^{(j)} p_i^{(j)} - y_i^{(j)} p_i^{(j)} - y_i^{(j)} \right| \\
&= \frac{1}{N}\sum_i \sum_j \left| y_i^{(j)} \left(p_i^{(j)} - 1 \right) + p_i^{(j)} \left(1 - y_i^{(j)} \right) \right| \\
&= \frac{1}{N}\sum_i \sum_j \left[y_i^{(j)} \left(1 - p_i^{(j)} \right) + p_i^{(j)} \left(1 - y_i^{(j)} \right) \right] \\
&= \frac{1}{N}\sum_i \left[\sum_j y_i^{(j)} - 2\sum_j y_i^{(j)} p_i^{(j)} + \sum_j p_i^{(j)} \right] \\
&= \frac{1}{N}\sum_i \sum_j y_i^{(j)} - 2\frac{1}{N}\sum_i \sum_j y_i^{(j)} p_i^{(j)} + \frac{1}{N}\sum_i \sum_j p_i^{(j)} \\
&= 2 - 2\frac{1}{N}\sum_i \sum_j y_i^{(j)} p_i^{(j)} \approx -2E_{P(x,y)}\left[P\left(\widehat{l} = l | \widehat{l} \sim p_i, l \sim y_i \right) \right]
\end{aligned}
$$

Therefore

We have

- For this reason we refer to this loss as expectation loss

Therefore

We have

- For this reason we refer to this loss as expectation loss

However, Why is this loss not being used?

- Maybe the following proposition will answer the question

We have

Proposition

- $\mathcal{L}_1$ and $\mathcal{L}_2$ losses applied to probabilities estimates coming from sigmoid (or softmax) have non-monotonic partial derivatives w.r.t. to the output of the final layer (and the loss is not convex nor concave w.r.t. to last layer weights). Furthermore, they vanish in both infinities, which slows down learning of heavily misclassified examples.

We have

Proposition

- $\mathcal{L}_1$ and $\mathcal{L}_2$ losses applied to probabilities estimates coming from sigmoid (or softmax) have non-monotonic partial derivatives w.r.t. to the output of the final layer (and the loss is not convex nor concave w.r.t. to last layer weights). Furthermore, they vanish in both infinities, which slows down learning of heavily misclassified examples.

Proof

- Let us denote sigmoid activation as

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

Thus, we have

Using Chain Rule

$$\frac{\partial \mathcal{L}_1 \circ \sigma}{\partial o}(o_p) = \frac{\partial \left[\left| 1 - \frac{1}{1+\exp\{-o\}} \right| \right] o_p}{\partial o}$$
$$= -\frac{\exp\{-o_p\}}{1+\exp\{-o_p\}}$$

Thus, we have

Using Chain Rule

$$\frac{\partial \mathcal{L}_1 \circ \sigma}{\partial o}(o_p) = \frac{\partial \left[\left\| 1 - \frac{1}{1+\exp\{-o\}} \right\| \right] o_p}{\partial o}$$

$$= -\frac{\exp\{-o_p\}}{1+\exp\{-o_p\}}$$

In addition, we have that

$$\lim_{o_p \to \infty} -\frac{\exp\{-o_p\}}{1+\exp\{-o_p\}} = \lim_{o_p \to -\infty} -\frac{\exp\{-o_p\}}{1+\exp\{-o_p\}} = 0$$

Additionally

We have that

$$\frac{\partial \mathcal{L}_1 \circ \sigma}{\partial o}(0) - \frac{\exp\{0\}}{1+\exp\{0\}} = -\frac{1}{4} < 0$$

Additionally

We have that

$$\frac{\partial \mathcal{L}_1 \circ \sigma}{\partial o}(0) - \frac{\exp\{0\}}{1+\exp\{0\}} = -\frac{1}{4} < 0$$

Additionally

- Lack of convexity comes from the same argument since second derivative w.r.t. to any weight in the final layer of the model changes sign.

Outline

Therefore

We have a problem with the use of these functions

- For the use on Neural Networks...

Therefore

We have a problem with the use of these functions

- For the use on Neural Networks...

We need something different

- Because even with the kernelized versions of them of the output at $\mathcal{L}_2$

$$\frac{\partial \mathcal{L}_2^{ker} \circ \sigma}{\partial o}(o_p) = \frac{\partial \left\| y - \sum_{i=1}^{m} \alpha_i k\left(\sigma\left(o\right), x_i\right)\right\|_2^2 (o_p)}{\partial o}$$

Therefore

We have a problem with the use of these functions

- For the use on Neural Networks...

We need something different

- Because even with the kernelized versions of them of the output at $\mathcal{L}_2$

$$\frac{\partial \mathcal{L}_2^{ker} \circ \sigma}{\partial o}\left(o_p\right) = \frac{\partial \left\| y - \sum_{i=1}^{m} \alpha_i k\left(\sigma\left(o\right), x_i\right)\right\|_2^2 \left(o_p\right)}{\partial o}$$

A small problem

- $k\left(\sigma\left(o\right), x_i\right)$ needs to be derivable by o

Not only that

This is applied to the exit of the neural network

- Actually, there is a layer that acts a kernel, the convolutional layer

$$Y_i^{(l)} = B_i^{(l)} + \sum_{j=1}^{m_1^{(l-1)}} K_{ij}^{(l)} * Y_j^{(l-1)}$$

Not only that

This is applied to the exit of the neural network

- Actually, there is a layer that acts a kernel, the convolutional layer

$$Y_i^{(l)} = B_i^{(l)} + \sum_{j=1}^{m_1^{(l-1)}} K_{ij}^{(l)} * Y_j^{(l-1)}$$

Thus, we can generalize this to a kernel layer

$$Y_i^{(l)} = B_i^{(l)} + \sum_{j=1}^{m_1^{(l-1)}} k\left(K_{ij}^{(l)}, Y_j^{(l-1)}\right)$$

Not only that

This is applied to the exit of the neural network

- Actually, there is a layer that acts a kernel, the convolutional layer

$$Y_i^{(l)} = B_i^{(l)} + \sum_{j=1}^{m_1^{(l-1)}} K_{ij}^{(l)} * Y_j^{(l-1)}$$

Thus, we can generalize this to a kernel layer

$$Y_i^{(l)} = B_i^{(l)} + \sum_{j=1}^{m_1^{(l-1)}} k\left(K_{ij}^{(l)}, Y_j^{(l-1)}\right)$$

And the problem

- Which One? A Research Topic...

Outline

Going back to our original cost function

Recall the binary linear classifiers with targets $y \in \{0, 1\}$

$$z = \boldsymbol{w}^T \boldsymbol{x} + b$$

$$y = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$

Going back to our original cost function

Recall the binary linear classifiers with targets $y \in \{0, 1\}$

$$z = \boldsymbol{w}^T \boldsymbol{x} + b$$

$$y = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$

The Goal is to correctly classify every training example

- this might be impossible if the dataset is not linearly separable.

Going back to our original cost function

Recall the binary linear classifiers with targets $y \in \{0, 1\}$

$$z = \boldsymbol{w}^T \boldsymbol{x} + b$$

$$y = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$

The Goal is to correctly classify every training example

- this might be impossible if the dataset is not linearly separable.

We want to avoid

- To do overfitting...

How to deal with this?

One natural criterion is to minimize the number of misclassified training examples

- We can try to solve by the using 0-1 loss:

$$\mathcal{L}_{0-1}(y,t) = \begin{cases} 0 & \text{if } y = t \\ 1 & otherwise \end{cases}$$

How to deal with this?

One natural criterion is to minimize the number of misclassified training examples

- We can try to solve by the using 0-1 loss:

$$\mathcal{L}_{0-1}(y,t) = \begin{cases} 0 & \text{if } y = t \\ 1 & otherwise \end{cases}$$

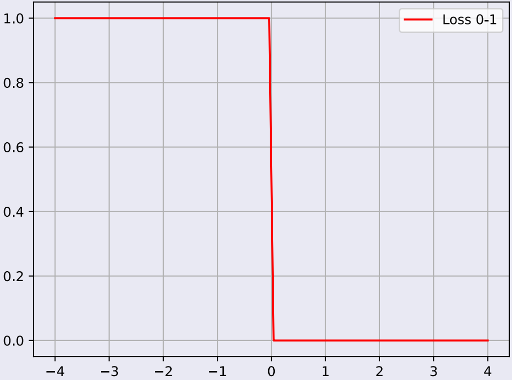
The cost function is just the loss averaged over the training examples

- We try to make it small

Outline

Attempt 0-1 Loss

We have something like this



Attempt 0-1 Loss

First Problem

- We need to compute the partial derivatives $\frac{\partial \mathcal{L}_{0-1}}{\partial w_j}$

Attempt 0-1 Loss

First Problem

- We need to compute the partial derivatives $\frac{\partial \mathcal{L}_{0-1}}{\partial w_j}$

Basically, we need to obtain

- How much does $\mathcal{L}_{0-1}$ change if you make a change to w_j?

Attempt 0-1 Loss

First Problem

- We need to compute the partial derivatives $\frac{\partial \mathcal{L}_{0-1}}{\partial w_j}$

Basically, we need to obtain

- How much does $\mathcal{L}_{0-1}$ change if you make a change to w_j?

We notice something

- As long we are not at the boundary, changes on w_j will not have no effect

$$\frac{\partial \mathcal{L}_{0-1}}{\partial w_j} = 0$$

As in the original 0-1 Cortez and Vapnik problem

Yes... at the original problem you have a 0-1 problem (0-1 SVM with Soft Margins)

- Which falls into a combinatorial problem... forget also on using Gradient to optimize it...

As in the original 0-1 Cortez and Vapnik problem

Yes... at the original problem you have a 0-1 problem (0-1 SVM with Soft Margins)

- Which falls into a combinatorial problem... forget also on using Gradient to optimize it...

Therefore, we need something different

- Ok... we need to look to another place...

Attempt Linear Regression

We have the following situation

$$y = \boldsymbol{w}^T \boldsymbol{x} + b$$
$$\mathcal{L}_2 = \frac{1}{2}(y - t)^2$$

Attempt Linear Regression

We have the following situation

$$y = \boldsymbol{w}^T \boldsymbol{x} + b$$

$$\mathcal{L}_2 = \frac{1}{2} (y - t)^2$$

We have two solutions (Look at our slides on Machine Learning)

- Closed form
- Gradient Descent form

Attempt Linear Regression

We have the following situation

$$y = \boldsymbol{w}^T \boldsymbol{x} + b$$

$$\mathcal{L}_2 = \frac{1}{2} (y - t)^2$$

We have two solutions (Look at our slides on Machine Learning)

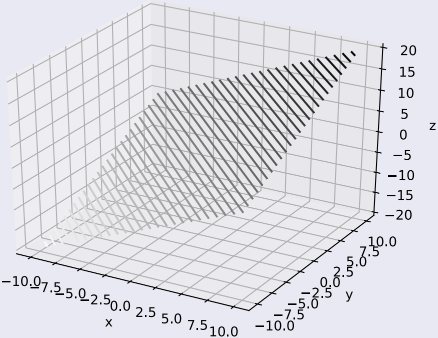
- Closed form
- Gradient Descent form

Does it make sense for classification?

- One obvious problem is that the predictions are real-valued rather than binary.

Example

WE have the loss function $y = \boldsymbol{w}^T \boldsymbol{x} + b$ with $t = 1.0$



It is possible to binarize this

By using a thrheshold

- At $y = \frac{1}{2}$

It is possible to binarize this

By using a thrheshold

- At $y = \frac{1}{2}$

This type of relaxation

- It is called **surrogate loss function**.

There is still a problem

Suppose we have a positive example, $t = 1$

- If we predict $y = 1$, we get a cost of 0, whereas if we make the wrong prediction $y = 0$, we get a cost of $\frac{1}{2}$,

$$\mathcal{L}_2 = \frac{1}{2}(y - t)^2$$

There is still a problem

Suppose we have a positive example, $t = 1$

- If we predict $y = 1$, we get a cost of 0, whereas if we make the wrong prediction $y = 0$, we get a cost of $\frac{1}{2}$,

$$\mathcal{L}_2 = \frac{1}{2}(y - t)^2$$

However, we can trick our output

- We really confident you have a positive example and we predict $y = 9$,

$$\mathcal{L}_2 = \frac{1}{2}(9 - 1)^2 = 32$$

There is still a problem

Suppose we have a positive example, $t = 1$

- If we predict $y = 1$, we get a cost of 0, whereas if we make the wrong prediction $y = 0$, we get a cost of $\frac{1}{2}$,

$$\mathcal{L}_2 = \frac{1}{2}(y - t)^2$$

However, we can trick our output

- We really confident you have a positive example and we predict $y = 9$,

$$\mathcal{L}_2 = \frac{1}{2}(9 - 1)^2 = 32$$

This is far higher than the cost for $y = 0$

- Therefore, the quadratic loss function sacrifices somethign when using it...

Outline

Attempt Logistic Nonlinearity

We can then filter the previous attempt by using a σ

$$z = \boldsymbol{w}^T \boldsymbol{x} + b$$

$$y = \sigma\left(z\right)$$

$$\mathcal{L}_2 = \frac{1}{2}\left(y - t\right)^2$$

$$\sigma\left(z\right) = \frac{1}{1 + \exp\left\{-z\right\}}$$

Attempt Logistic Nonlinearity

We can then filter the previous attempt by using a σ

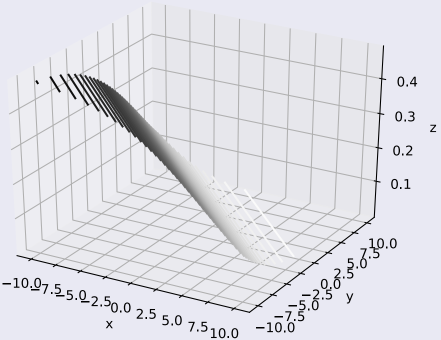
$$
\begin{aligned}
z &= \boldsymbol{w}^T \boldsymbol{x} + b \\
y &= \sigma\left(z\right) \\
\mathcal{L}_2 &= \frac{1}{2}\left(y - t\right)^2 \\
\sigma\left(z\right) &= \frac{1}{1 + \exp\left\{-z\right\}}
\end{aligned}
$$

Something Notable

- Notice that this model solves the problem we observed with linear regression.
  - As the predictions get more and more confident on the correct answer, the loss continues to decrease.

Example of this

We have the loss function $\mathcal{L}_2 = \frac{1}{2}\left(\sigma\left(z\right) - t\right)^2$



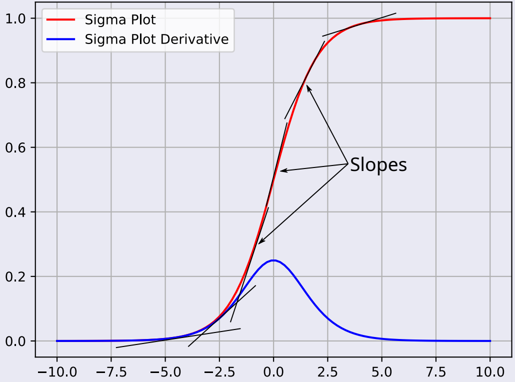
Therefore

The derivative is equal to

$$\frac{\partial \sigma\left(z\right)}{\partial z} = \frac{\exp\left\{-z\right\}}{\left[1+\exp\left\{-z\right\}\right]^{2}} = \sigma\left(z\right)\left[1-\sigma\left(z\right)\right]$$

Example

We have the following situation



The nice part of this function

Something Notable

- If your target is $t = 1$ and you are learning

The nice part of this function

Something Notable

- If your target is $t = 1$ and you are learning

You accelerate fast by the use of the Gradient Descent

- Once you get near to it you decelerate... in your learning

How does this learning looks like?

By Chain Rule

$$\frac{d\mathcal{L}_2}{dz} = \frac{d\mathcal{L}_2}{dy} \times \frac{dy}{dz} = (y - t)\, y\, (1 - y)$$

How does this learning looks like?

By Chain Rule

$$\frac{d\mathcal{L}_2}{dz} = \frac{d\mathcal{L}_2}{dy} \times \frac{dy}{dz} = (y - t)\, y\, (1 - y)$$

Therefore, we have that

$$\frac{\partial \mathcal{L}_2}{\partial w_j} = \frac{d\mathcal{L}_2}{dz} \times \frac{\partial z}{\partial w_j} = \frac{d\mathcal{L}_2}{dz} \times x_j$$

Outline

Relation with Automatic Differentiation

This formula can be used re-used

- Actually there is a way to reuse the previous formula for the bias

Relation with Automatic Differentiation

This formula can be used re-used

- Actually there is a way to reuse the previous formula for the bias

We have that

$$\frac{\partial \mathcal{L}_2}{\partial b} = \frac{d\mathcal{L}_2}{dz} \times \frac{\partial z}{\partial b} = \frac{d\mathcal{L}_2}{dz}$$

Relation with Automatic Differentiation

This formula can be used re-used

- Actually there is a way to reuse the previous formula for the bias

We have that

$$\frac{\partial \mathcal{L}_2}{\partial b} = \frac{d\mathcal{L}_2}{dz} \times \frac{\partial z}{\partial b} = \frac{d\mathcal{L}_2}{dz}$$

This re-usability

- It is at the center of the Automatic Differentiation

However there is a glitch!!!

If you have an incorrect classification of a sample

- You can predict a negative label with $z = -5$ thus $y \approx 0.0067$ for a positive one.

However there is a glitch!!!

If you have an incorrect classification of a sample

- You can predict a negative label with $z = -5$ thus $y \approx 0.0067$ for a positive one.

We find that

$$\frac{d\mathcal{L}_2}{dz} = -0.0066$$

However there is a glitch!!!

If you have an incorrect classification of a sample

- You can predict a negative label with $z = -5$ thus $y \approx 0.0067$ for a positive one.

We find that

$$\frac{d\mathcal{L}_2}{dz} = -0.0066$$

This is a pretty small value, considering how big the mistake was

- Therefore, we have that this gradient will not help this sample to get out of the error

The Problem

We have that

- The problem with squared error loss in the classification setting is that it does not distinguish bad predictions from extremely bad predictions.

The Problem

We have that

- The problem with squared error loss in the classification setting is that it does not distinguish bad predictions from extremely bad predictions.

We need something better for classification

- Question What?

Outline

Problem with Squared Error

It treats small values of different magnitudes equally

- $y = 0.01$ and $y = 0.00001$ as nearly equivalent (for a positive example)

Problem with Squared Error

It treats small values of different magnitudes equally

- $y = 0.01$ and $y = 0.00001$ as nearly equivalent (for a positive example)

What we want

- We want a loss function which makes these very different!!!

Cross-Entropy(CE)

Defined as follow

$$\mathcal{L}_{\mathcal{CE}}\left(y,t\right)=\begin{cases}-\log y & \text{if } t=1\\ -\log\left(1-y\right) & \text{if } t=0\end{cases}$$

Cross-Entropy(CE)

Defined as follow

$$\mathcal{L}_{\mathcal{CE}}\left(y,t\right)=\begin{cases}-\log y & \text{if } t=1\\ -\log\left(1-y\right) & \text{if } t=0\end{cases}$$

In our previous example

- $\mathcal{L}_{\mathcal{CE}}\left(0.01,1\right)=4.6$
- $\mathcal{L}_{\mathcal{CE}}\left(0.00001,1\right)=11.5$

Cross-Entropy(CE)

Defined as follow

$$\mathcal{L}_{\mathcal{CE}}\left(y,t\right) = \begin{cases} -\log y & \text{if } t=1 \\ -\log\left(1-y\right) & \text{if } t=0 \end{cases}$$

In our previous example

- $\mathcal{L}_{\mathcal{CE}}\left(0.01,1\right) = 4.6$
- $\mathcal{L}_{\mathcal{CE}}\left(0.00001,1\right) = 11.5$

Therefore

- cross-entropy treats the latter as much worse than the former.

A Better Loss Function

We can collapse the previous definition to

$$\mathcal{L}_{\mathcal{CE}}(y, t) = -t \log y - (1 - t) \log (1 - y)$$

A Better Loss Function

We can collapse the previous definition to

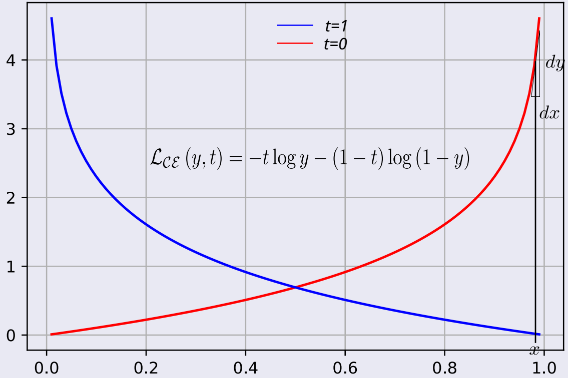
$$\mathcal{L}_{\mathcal{CE}}\left(y,t\right) = -t\log y - \left(1-t\right)\log\left(1-y\right)$$

We have the following example

- Split the real line in two classes positive side $t=1$ and negative side $t=0$

Example

We have the following



Therefore, we have

A small change on x, dx implies a large in y, dy

- This is what we wanted

Therefore, we have

A small change on x, dx implies a large in y, dy

- This is what we wanted

Now, the derivatives

- To analyze thew possible updates

Therefore, we have

The derivative of $\mathcal{L}_{\mathcal{CE}}$ with respect to y

$$\frac{d\mathcal{L}_{\mathcal{CE}}}{dy} = -\frac{t}{y} + \frac{1-t}{1-y}$$

Therefore, we have

The derivative of $\mathcal{L}_{\mathcal{CE}}$ with respect to y

$$\frac{d\mathcal{L}_{\mathcal{CE}}}{dy} = -\frac{t}{y} + \frac{1-t}{1-y}$$

The derivative of $\mathcal{L}_{\mathcal{CE}}$ with respect to z

$$\frac{d\mathcal{L}_{\mathcal{CE}}}{dz} = \frac{d\mathcal{L}_{\mathcal{CE}}}{dy} \times \frac{dy}{dz} = \frac{d\mathcal{L}_{\mathcal{CE}}}{dy} \times y\left(1-y\right)$$

Therefore, we have

The derivative of $\mathcal{L}_{\mathcal{CE}}$ with respect to y

$$\frac{d\mathcal{L}_{\mathcal{CE}}}{dy} = -\frac{t}{y} + \frac{1-t}{1-y}$$

The derivative of $\mathcal{L}_{\mathcal{CE}}$ with respect to z

$$\frac{d\mathcal{L}_{\mathcal{CE}}}{dz} = \frac{d\mathcal{L}_{\mathcal{CE}}}{dy} \times \frac{dy}{dz} = \frac{d\mathcal{L}_{\mathcal{CE}}}{dy} \times y\,(1-y)$$

The derivative of $\mathcal{L}_{\mathcal{CE}}$ with respect to w_j

$$\frac{d\mathcal{L}_{\mathcal{CE}}}{dw_j} = \frac{d\mathcal{L}_{\mathcal{CE}}}{dz} \times \frac{d\mathcal{L}_{\mathcal{CE}}}{dz} = \frac{d\mathcal{L}_{\mathcal{CE}}}{dz} \times x_j$$

Outline

The final touch up

There is a big problem

- What happens if we have a positive example $(t = 1)$
  - And you get $y \approx 0$

The final touch up

There is a big problem

- What happens if we have a positive example $(t = 1)$
  - And you get $y \approx 0$

This is likely to happen at the very beginning of training

- But if y is small enough, it could be smaller than the smallest floating point value
  - Basically 0 or near by to 0

The final touch up

There is a big problem

- What happens if we have a positive example $(t = 1)$
  - And you get $y \approx 0$

This is likely to happen at the very beginning of training

- But if y is small enough, it could be smaller than the smallest floating point value
  - Basically 0 or near by to 0

Then when we compute the cross-entropy

- We have that $\frac{d\mathcal{L}_{\mathcal{CE}}}{dy}$ becomes extremely large in magnitud

Better, we bound the output of the network

Through the use of the softmax for bounding the output of the network between 0 and 1

$$\sigma_i\left(z_i\right)=\frac{\exp\left\{z_i\right\}}{\sum_{k=1}^{C}\exp\left\{z_k\right\}}$$

Better, we bound the output of the network

Through the use of the softmax for bounding the output of the network between 0 and 1

$$\sigma_i\left(z_i\right) = \frac{\exp\left\{z_i\right\}}{\sum_{k=1}^{C} \exp\left\{z_k\right\}}$$

Or for the binary class

$$\sigma\left(z\right) = \begin{cases} \frac{\exp\{z\}}{1+\exp\{z\}} & t = 1 \\ \frac{1}{1+\exp\{z\}} & t = 0 \end{cases}$$

We finish with the Log Cross Entropy

Therefore, as we know $\mathcal{L}_{\mathcal{CE}}(y,t) = -t\log y - (1-t)\log(1-y)$, then

$$\mathcal{L}_{\mathcal{LCE}}(\sigma(z), t) = -t\log\left(\frac{exp\{z\}}{1+exp\{-z\}}\right) - (1-t)\log\left(\frac{1}{1+exp\{z\}}\right)$$

We finish with the Log Cross Entropy

Therefore, as we know $\mathcal{L}_{\mathcal{CE}}(y,t) = -t\log y - (1-t)\log(1-y)$, then

$$\mathcal{L}_{\mathcal{LCE}}(\sigma(z), t) = -t\log\left(\frac{exp\{z\}}{1+exp\{-z\}}\right) - (1-t)\log\left(\frac{1}{1+exp\{z\}}\right)$$

The interesting part is

What about the derivative?

We have

$$
\begin{aligned}
\frac{d\mathcal{L}_{\mathcal{LCE}}}{dz} &= \frac{d\mathcal{L}_{\mathcal{LCE}}}{d\sigma\left(z\right)} \times \frac{d\sigma\left(z\right)}{z} \\
&= \left\{-\frac{t}{\sigma\left(z\right)} + \frac{\left(1-t\right)}{1-\sigma\left(z\right)}\right\} \times \sigma\left(z\right)\left(1-\sigma\left(z\right)\right) \\
&= \sigma\left(z\right) - t
\end{aligned}
$$

What about the derivative?

We have

$$\begin{aligned}\frac{d\mathcal{L}_{\mathcal{LCE}}}{dz} &= \frac{d\mathcal{L}_{\mathcal{LCE}}}{d\sigma(z)} \times \frac{d\sigma(z)}{z}\\ &= \left\{-\frac{t}{\sigma(z)} + \frac{(1-t)}{1-\sigma(z)}\right\} \times \sigma(z)(1-\sigma(z))\\ &= \sigma(z) - t\end{aligned}$$

Wow... quite simple derivative

- Observe that this is exactly the same formula $\frac{d\mathcal{L}_2}{dy}$ as for in the case of linear regression.

Interpretation

if $y > t$, you made too positive a prediction

- You want to shift your prediction in the negative direction.

Interpretation

if $y > t$, you made too positive a prediction

- You want to shift your prediction in the negative direction.

if $y < t$

- You want to shift your prediction in the positive direction.

Outline

Now, we want to do multiclass problems

For this, we have the softmax

$$y_i = \sigma(\boldsymbol{z})_i = \frac{\exp\{z_i\}}{\sum_{d=1}^{C} \exp\{z_d\}} \text{ for } c = 1, ..., C$$

Derivative of the softmax function

We can do the following

$$\sum_{C} = \sum_{d=1}^{C} e^{z_d} \text{ for } c = 1, ..., C$$

- In this way $y_c = \frac{\exp\{z_c\}}{\sum_c}$

Derivative of the softmax function

We can do the following

$$\sum_{C} = \sum_{d=1}^{C} e^{z_d} \text{ for } c = 1, ..., C$$

- In this way $y_c = \frac{\exp\{z_c\}}{\sum_c}$

Then, we have the derivatives

1. if $i = j$:
   - $\frac{\partial y_i}{\partial z_i} = \frac{\partial \frac{\exp\{z_i\}}{\sum_c}}{\partial z_i} = \frac{\exp\{z_i\} \sum_C - \exp\{z_i\} \exp\{z_i\}}{\sum_C^2} = \frac{\exp\{z_i\}}{\sum_C} \times \frac{\sum_C - \exp\{z_i\}}{\sum_C} = \frac{\exp\{z_i\}}{\sum_C} \left(1 - \frac{\exp\{z_i\}}{\sum_C} \right) = y_i \left(1 - y_i\right)$
2. if $i \neq j$:
   - $\frac{\partial y_i}{\partial z_i} = \frac{dy_i}{\partial z_j} = \frac{\partial \frac{\exp\{z_i\}}{\sum_c}}{\partial z_j} = \frac{0 - \exp\{x_i\} \exp\left\{x_j\right\}}{\sum_C^2} = -\frac{\exp\{x_i\}}{\sum_C} \times \frac{\exp\left\{x_j\right\}}{\sum_C} = -y_i y_j$

Now

To derive the loss function for the softmax function we start out from the likelihood function

$$\arg\max \mathcal{L}\left(\theta|\boldsymbol{t},\boldsymbol{z}\right)$$

Now

To derive the loss function for the softmax function we start out from the likelihood function

$$\arg\max \mathcal{L}\left(\theta|\boldsymbol{t},\boldsymbol{z}\right)$$

Now, we can use the joint probability $P\left(\boldsymbol{t},\boldsymbol{z}|\theta\right)$

$$P\left(\boldsymbol{t},\boldsymbol{z}|\theta\right) = P\left(\boldsymbol{t}|\boldsymbol{z},\theta\right)P\left(\boldsymbol{z}|\theta\right)$$

Now

To derive the loss function for the softmax function we start out from the likelihood function

$$\arg\max \mathcal{L}\left(\theta|\boldsymbol{t},\boldsymbol{z}\right)$$

Now, we can use the joint probability $P\left(\boldsymbol{t},\boldsymbol{z}|\theta\right)$

$$P\left(\boldsymbol{t},\boldsymbol{z}|\theta\right) = P\left(\boldsymbol{t}|\boldsymbol{z},\theta\right)P\left(\boldsymbol{z}|\theta\right)$$

Since we are not interested in the probability of $\boldsymbol{z}$

$$\mathcal{L}\left(\theta|\boldsymbol{t},\boldsymbol{z}\right) = P\left(\boldsymbol{t}|\boldsymbol{z},\theta\right) = P\left(\boldsymbol{t}|\boldsymbol{z}\right)$$

Thus, we have that

Since each t_c is dependant on the full $\boldsymbol{z}$ and only one class can activated in the $\boldsymbol{t}$

$$P\left(\boldsymbol{t}|\boldsymbol{z}\right)=\prod_{i=1}^{C}P\left(\boldsymbol{t_c}|\boldsymbol{z}\right)^{t_c}=\prod_{i=1}^{C}\sigma\left(\boldsymbol{z}\right)^{t_c}=\prod_{i=1}^{C}{y_c}^{t_c}$$

Thus, we have that

Since each t_c is dependant on the full $\boldsymbol{z}$ and only one class can activated in the $\boldsymbol{t}$

$$P\left(\boldsymbol{t}|\boldsymbol{z}\right)=\prod_{i=1}^{C}P\left(\boldsymbol{t_c}|\boldsymbol{z}\right)^{t_c}=\prod_{i=1}^{C}\sigma\left(\boldsymbol{z}\right)^{t_c}=\prod_{i=1}^{C}{y_c}^{t_c}$$

Then, using the negative log-likelihood

$$-\log\mathcal{L}\left(\theta|\boldsymbol{t},\boldsymbol{z}\right)=\phi\left(\boldsymbol{t},\boldsymbol{z}\right)=-\sum_{c=1}^{C}t_c\log\left(y_c\right)$$

- Which is the cross-entropy error function

Therefore

We have that under a batch of n samples

$$\phi\left(T,Y\right)=\sum_{i=1}^{n}\phi\left(\boldsymbol{t}_{i},\boldsymbol{y}_{i}\right)=-\sum_{i=1}^{n}\sum_{c=1}^{C}t_{ic}\log\left(y_{ic}\right)$$

Derivative of the cross-entropy loss function for the softmax function

We have that

$$
\begin{aligned}
\frac{\partial \phi\left(\boldsymbol{t}, \boldsymbol{z}\right)}{\partial z_i} &= -\sum_{j=1}^{C} \frac{\partial t_j \log\left(y_j\right)}{\partial z_i} = -\sum_{j=1}^{C} t_j \frac{\partial \log\left(y_j\right)}{\partial z_i} \\
&= -\sum_{j=1}^{C} t_j \frac{1}{y_j} \times \frac{\partial y_j}{\partial z_i} \\
&= -\frac{t_i}{y_i} \times \frac{\partial y_i}{\partial z_i} - \sum_{j \neq i}^{C} \frac{t_j}{y_j} \times \frac{\partial y_j}{\partial z_i} \\
&= -\frac{t_i}{y_i} y_i \left(1 - y_i\right) - \sum_{j \neq i}^{C} \frac{t_j}{y_j} \left(-y_j y_i\right) \\
&= -t_i + t_i y_i + \sum_{j \neq i}^{C} t_j y_j = -t_i + y_i \sum_{j=1}^{C} t_j = y_i - t_i
\end{aligned}
$$

Outline

Yann LeCunn “Who is afraid of non-convex loss functions?”[3]

Machine Learning theory has essentially never moved beyond convex models

- This is actually wrong

Yann LeCunn "Who is afraid of non-convex loss functions?"[3]

Machine Learning theory has essentially never moved beyond convex models

- This is actually wrong

Given the previous development

- Accepting non-convexity allows elegant models

Yann LeCunn "Who is afraid of non-convex loss functions?"[3]

Machine Learning theory has essentially never moved beyond convex models

- This is actually wrong

Given the previous development

- Accepting non-convexity allows elegant models

Not only that

- The price we pay for insisting on convexity is an unbearable increase in the size of the model
  - Actually fat shallow models vs something else...

Therefore

Based on this idea

- We need to look at different functions for loss

Therefore

Based on this idea

- We need to look at different functions for loss

For example in [4]

- They proposed a more general loss function based in a parameter $\alpha \in (0, \infty]$

Outline

We have

Definition [5, 4]

- Let $\mathcal{P}(\mathcal{Y})$ be the set of probability distributions over $\mathcal{Y}$. For $\alpha \in (0, \infty]$, we define α-loss for $\alpha \in (0,1) \cup (1, \infty)$, $l^{\alpha} : \mathcal{Y} \to \mathbb{R}^{+}$ as

$$l^{\alpha}(y, P_Y) = \frac{\alpha}{1-\alpha}\left[1 - P_Y(y)^{1-1/\alpha}\right]$$

and by continuous extension,

$$l^{1}(y, P_Y) = -\log P_Y(y) \text{ and}$$

$$l^{\infty}(y, P_Y) = 1 - \log P_Y(y)$$

Cases

For $\alpha = 1$

- Such a risk minimization involves minimizing the average log loss,
  - refining a posterior belief over all y for a given observation x.

Cases

For $\alpha = 1$

- Such a risk minimization involves minimizing the average log loss,
  - refining a posterior belief over all y for a given observation x.

Furthermore, as α increases from 1 to ∞

- The loss function increasingly limits the effect of the low probability outcomes

$$\lim_{\alpha \to \infty} l^{\alpha}(y, P_Y) = \lim_{\alpha \to \infty} \frac{\alpha}{1-\alpha} \times \lim_{\alpha \to \infty} \left[1 - P_Y(y)^{1 - 1/\alpha}\right] = P_Y(y) - 1$$

Not only that

As α decreases from 1 towards 0

- The loss function places increasingly higher weights on the low probability outcomes

Not only that

As α decreases from 1 towards 0

- The loss function places increasingly higher weights on the low probability outcomes

Until at $\alpha = 0$

$$\lim_{\alpha \to 0} \frac{\alpha}{1-\alpha} \left[1 - P_Y(y)^{1-1/\alpha} \right] = \lim_{\alpha \to 0} P_Y(y)^{1-1/\alpha} - 1 = \lim_{\alpha \to 0} \frac{P_Y(y)}{P_Y(y)^{1/\alpha}} - 1 = \infty$$

Therefore

We have that

- The loss function pays an infinite cost by ignoring the training data distribution completely.

Therefore

We have that

- The loss function pays an infinite cost by ignoring the training data distribution completely.

Note the following

- α quantifies the level of certainty placed on the posterior distribution

Therefore

We have that

- The loss function pays an infinite cost by ignoring the training data distribution completely.

Note the following

- α quantifies the level of certainty placed on the posterior distribution

Therefore

- Larger α indicate increasing certainty over a smaller set of Y.
- Smaller α distributes the uncertainty over more (and eventually, all) possibles values of Y .

Actually

For $\alpha = \infty$

- The distribution becomes the hard-decoding Maximum A Posteriori rule.

Risk Minimization under this loss

Proposition

- For each $\alpha \in (0, \infty]$, the minimal α-risk is

$$\min_{P_{\widehat{Y}|X}} \mathbb{E}_{X,Y}\left[l^{\alpha}\left(Y, P_{\widehat{Y}|X}\right)\right] = \frac{\alpha}{\alpha - 1}\left[1 - \exp\left\{\frac{1-\alpha}{\alpha} H_{\alpha}^{A}\left(Y|X\right)\right\}\right]$$

where $H_{\alpha}^{A}\left(Y|X\right) = \frac{\alpha}{1-\alpha}\log \sum_y \left(\sum_x P_{X,Y}\left(x,y\right)^{\alpha}\right)^{1/\alpha}$ is the Arimoto conditional entropy of order α. The resultin minimizer is the α-tilted true posterior

$$P^{*}_{\widehat{Y}|X}\left(y|x\right) = \frac{P_{Y|X}\left(y|x\right)^{\alpha}}{\sum_y P_{Y|X}\left(y|x\right)^{\alpha}}$$

Risk Minimization under this loss

Proposition

- For each $\alpha \in (0, \infty]$, the minimal α-risk is

$$\min_{P_{\widehat{Y}|X}} \mathbb{E}_{X,Y}\left[l^{\alpha}\left(Y, P_{\widehat{Y}|X}\right)\right] = \frac{\alpha}{\alpha-1}\left[1 - \exp\left\{\frac{1-\alpha}{\alpha} H_{\alpha}^{A}\left(Y|X\right)\right\}\right]$$

where $H_{\alpha}^{A}\left(Y|X\right) = \frac{\alpha}{1-\alpha}\log\sum_{y}\left(\sum_{x} P_{X,Y}\left(x,y\right)^{\alpha}\right)^{1/\alpha}$ is the Arimoto conditional entropy of order α. The resultin minimizer is the α-tilted true posterior

$$P^{*}_{\widehat{Y}|X}\left(y|x\right) = \frac{P_{Y|X}\left(y|x\right)^{\alpha}}{\sum_{y} P_{Y|X}\left(y|x\right)^{\alpha}}$$

Take a look at [5]

- For the proof

Outline

Examples

Differentially Private Empirical Risk Minimization with Smooth Non-Convex Loss Functions: A Non-Stationary View [6]

- Here, the Differentially Private Empirical Risk Minimization is studied

Examples

Differentially Private Empirical Risk Minimization with Smooth Non-Convex Loss Functions: A Non-Stationary View [6]

- Here, the Differentially Private Empirical Risk Minimization is studied

From Convex to Nonconvex: a Loss Function Analysis for Binary Classification [7]

- A new smoothed version of the loss 0-1 function is proposed
  - Although, it seems to be that sigmoid cross entropy is better...
- An new method to compare different loss functions

Examples

Differentially Private Empirical Risk Minimization with Smooth Non-Convex Loss Functions: A Non-Stationary View [6]

- Here, the Differentially Private Empirical Risk Minimization is studied

From Convex to Nonconvex: a Loss Function Analysis for Binary Classification [7]

- A new smoothed version of the loss 0-1 function is proposed
  - Although, it seems to be that sigmoid cross entropy is better...
- An new method to compare different loss functions

Deep Neural Networks with Multi-Branch Architectures Are Intrinsically Less Non-Convex [8]

- Architectures using subnetworks as the transformers are non-convex in nature

Outline

It is clear that many connections need to be done

From the Reproducing Kernels

- As Layers on the Neuronal Networks
  - Still a Deeper study needs to be done to finish the connections on this regard...

It is clear that many connections need to be done

From the Reproducing Kernels

- As Layers on the Neuronal Networks
  - Still a Deeper study needs to be done to finish the connections on this regard...

To the need to explore novel non-convex loss functions

- Making possible to improve upon the traditional loss functions for Neural Networks

It is clear that many connections need to be done

From the Reproducing Kernels

- As Layers on the Neuronal Networks
  - Still a Deeper study needs to be done to finish the connections on this regard...

To the need to explore novel non-convex loss functions

- Making possible to improve upon the traditional loss functions for Neural Networks

Therefore

- This is a new frontier in the study of neural networks...

D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (elus)," 2016.

K. Janocha and W. M. Czarnecki, "On loss functions for deep neural networks in classification," *arXiv preprint arXiv:1702.05659*, 2017.

Y. Lecun, "Who is afraid of nonconvex loss functions?."

T. Sypherd, M. Diaz, H. Laddha, L. Sankar, P. Kairouz, and G. Dasarathy, "A class of parameterized loss functions for classification: Optimization tradeoffs and robustness characteristics," *arXiv preprint arXiv:1906.02314*, 2019.

J. Liao, O. Kosut, L. Sankar, and F. P. Calmon, "A tunable measure for information leakage," in *2018 IEEE International Symposium on Information Theory (ISIT)*, pp. 701–705, IEEE, 2018.

D. Wang and J. Xu, "Differentially private empirical risk minimization with smooth non-convex loss functions: A non-stationary view," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, pp. 1182–1189, 2019.

L. Zhao, M. Mammadov, and J. Yearwood, "From convex to nonconvex: a loss function analysis for binary classification," in *2010 IEEE International Conference on Data Mining Workshops*, pp. 1281–1288, IEEE, 2010.

H. Zhang, J. Shao, and R. Salakhutdinov, "Deep neural networks with multi-branch architectures are intrinsically less non-convex," in *The 22nd International Conference on Artificial Intelligence and Statistics*, pp. 1099–1109, 2019.