Differentiate Almost Everywhere

VLC = ∭ Vision Learning Control dVdLdC

Differentiable Relaxations and Reparameterisations

Jonathon Hare

Vision, Learning and Control
University of Southampton

What are differentiable relaxations and reparameterisations?

- We've seen that we can build arbitrary computational graphs from a variety of building blocks

What are differentiable relaxations and reparameterisations?

- We've seen that we can build arbitrary computational graphs from a variety of building blocks
- But, those blocks need to be differentiable to work in our optimisation framework
  - More specifically they need to be continuous and *differentiable almost everywhere*.

What are differentiable relaxations and reparameterisations?

- We've seen that we can build arbitrary computational graphs from a variety of building blocks
- But, those blocks need to be differentiable to work in our optimisation framework
  - More specifically they need to be continuous and *differentiable almost everywhere*.
- That limits what we can do... Can we work around that?
  - Relaxations — make continuous (and differentiable everywhere) approximations.
  - Reparameterisations — rewrite functions to factor out stochastic variables from the parameters.

Aside: continuity and differentiable almost everywhere

- Consider the ReLU function $f(x) = max(0, x)$

Aside: continuity and differentiable almost everywhere

- Consider the ReLU function $f(x) = max(0, x)$
  - ReLU is *continuous*
    - it does not have any abrupt changes in value
    - small changes in x result in small changes to $f(x)$ everywhere in the domain of x

Aside: continuity and differentiable almost everywhere

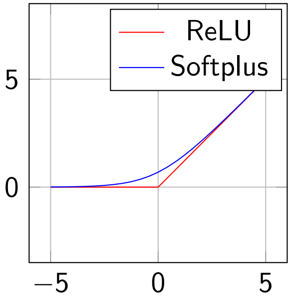
- Consider the ReLU function $f(x) = max(0, x)$
  - ReLU is *continuous*
    - it does not have any abrupt changes in value
    - small changes in x result in small changes to $f(x)$ everywhere in the domain of x
  - ReLU is *differentiable almost everywhere*
    - No gradient at $x = 0$; only *left* and *right* gradients at that point
    - There are *subgradients* at $x = 0$; implementations usually just arbitrarily pick $f'(0) = 0$

Aside: continuity and differentiable almost everywhere

- Consider the ReLU function $f(x) = max(0, x)$
  - ReLU is *continuous*
    - it does not have any abrupt changes in value
    - small changes in x result in small changes to $f(x)$ everywhere in the domain of x
  - ReLU is *differentiable almost everywhere*
    - No gradient at $x = 0$; only *left* and *right* gradients at that point
    - There are *subgradients* at $x = 0$; implementations usually just arbitrarily pick $f'(0) = 0$
- Functions that are differentiable almost everywhere or have subgradients tend to be compatible with gradient descent methods
  - We expect that the loss landscape is different for each batch & that we'll never actually reach a minima, and we only need to *mostly* take steps in the right direction.

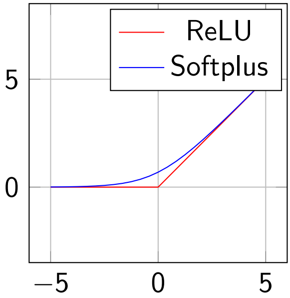
Relaxing ReLU

- Softplus ($\ln(1 + e^x)$) is a relaxation of ReLU that is *differentiable everywhere*.



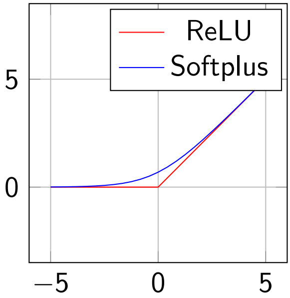
Relaxing ReLU

- Softplus ($\ln(1 + e^x)$) is a relaxation of ReLU that is *differentiable everywhere*.
- Its derivative is the Sigmoid function



Relaxing ReLU

- Softplus ($\ln(1 + e^x)$) is a relaxation of ReLU that is *differentiable everywhere*.
- Its derivative is the Sigmoid function
- Not widely used; counterintuitively, even though it neither saturates completely and is differentiable everywhere, empirically it has been shown that ReLU works better.



Interpretations of softmax

- Up until now we've really considered softmax as a generalisation of sigmoid (which represents a probability distribution over a binary variable) to many output categories.
    - softmax transforms a vector of logits into a probability distribution over categories.

Interpretations of softmax

- Up until now we've really considered softmax as a generalisation of sigmoid (which represents a probability distribution over a binary variable) to many output categories.
    - softmax transforms a vector of logits into a probability distribution over categories.
- As you might guess from the name, softmax is a relaxation...

Interpretations of softmax

- Up until now we've really considered softmax as a generalisation of sigmoid (which represents a probability distribution over a binary variable) to many output categories.
  - softmax transforms a vector of logits into a probability distribution over categories.
- As you might guess from the name, softmax is a relaxation...
  - but not of the max function like the name would suggest!

Interpretations of softmax

- Up until now we've really considered softmax as a generalisation of sigmoid (which represents a probability distribution over a binary variable) to many output categories.
  - softmax transforms a vector of logits into a probability distribution over categories.
- As you might guess from the name, softmax is a relaxation...
  - but not of the max function like the name would suggest!
  - softmax can be viewed as a continuous and differentiable relaxation of the $\arg\max$ function with one-hot output encoding.

Interpretations of softmax

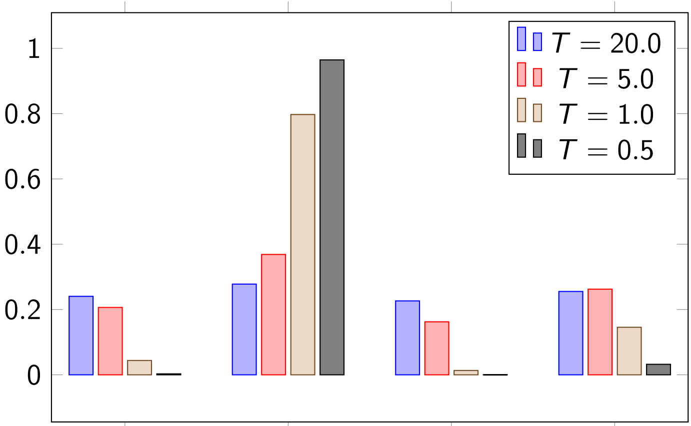
- Up until now we've really considered softmax as a generalisation of sigmoid (which represents a probability distribution over a binary variable) to many output categories.
  - softmax transforms a vector of logits into a probability distribution over categories.
- As you might guess from the name, softmax is a relaxation...
  - but not of the max function like the name would suggest!
  - softmax can be viewed as a continuous and differentiable relaxation of the arg max function with one-hot output encoding.
  - The arg max function is not continuous or differentiable; softmax provides an approximation:

$$
\begin{aligned}
\boldsymbol{x} &= [\quad 1.1 \quad 4.0 \quad -0.1 \quad 2.3 \quad] \\
\arg\max(\boldsymbol{x}) &= [\quad 0 \quad 1 \quad 0 \quad 0 \quad] \\
\operatorname{softmax}(\boldsymbol{x}) &= [\quad 0.044 \quad 0.797 \quad 0.013 \quad 0.146 \quad]
\end{aligned}
$$

The Softmax function with temperature

Consider what happens if you were to divide the input logits to a softmax by a scalar temperature parameter T.

$$\text{softmax}(\boldsymbol{x}/T)_i = \frac{e^{x_i/T}}{\sum_{j=1}^{K} e^{x_j/T}} \qquad \forall i = 1, 2, \ldots, K$$



arg max — softmax with temperature

$$
\begin{array}{rlllll}
\boldsymbol{x} = & [& 1.1 & 4.0 & \text{-0.1} & 2.3 &] \\
\text{softmax}(\boldsymbol{x}/1.0) = & [& 0.044 & 0.797 & 0.013 & 0.146 &] \\
\text{softmax}(\boldsymbol{x}/0.8) = & [& 0.023 & 0.868 & 0.005 & 0.104 &] \\
\text{softmax}(\boldsymbol{x}/0.6) = & [& 0.008 & 0.937 & 0.001 & 0.055 &] \\
\text{softmax}(\boldsymbol{x}/0.4) = & [& \text{6.997e-04} & \text{9.852e-01} & \text{3.484e-05} & \text{1.405e-02} &] \\
\text{softmax}(\boldsymbol{x}/0.2) = & [& \text{5.042e-07} & \text{9.998e-01} & \text{1.250e-09} & \text{2.034e-04} &]
\end{array}
$$

arg max — scalar approximation

- What if you want to get a scalar approximation to the index of the arg max rather than a probability distribution approximating the one-hot form?
  - Caveat: we can't actually get a guaranteed integer representation as that would be non-differentiable; we'll have to live with a float that is an approximation.

$\arg\max$ — scalar approximation

- What if you want to get a scalar approximation to the index of the $\arg\max$ rather than a probability distribution approximating the one-hot form?
  - Caveat: we can't actually get a guaranteed integer representation as that would be non-differentiable; we'll have to live with a float that is an approximation.
- First, consider how to convert a one-hot vector to index representation in a differentiable manner: $[0, 0, 1, 0] \rightarrow 2$
  - Just dot product with a vector of indices: $[0, 1, 2, 3]$

arg max — scalar approximation

- What if you want to get a scalar approximation to the index of the $\arg\max$ rather than a probability distribution approximating the one-hot form?
    - Caveat: we can't actually get a guaranteed integer representation as that would be non-differentiable; we'll have to live with a float that is an approximation.
- First, consider how to convert a one-hot vector to index representation in a differentiable manner: $[0, 0, 1, 0] \to 2$
    - Just dot product with a vector of indices: $[0, 1, 2, 3]$
- The same process can be applied to the softmax distribution
    - As temperature $T \to 0$, $\text{softmax}(\boldsymbol{x}/T) \cdot [0, 1, \dots, N] \to \arg\max(\boldsymbol{x})$ for $\boldsymbol{x} \in \mathbb{R}^N$.

arg max — scalar approximation

$$\boldsymbol{x} = \begin{bmatrix} 1.1 & 4.0 & -0.1 & 2.3 \end{bmatrix}^\top$$

$$\boldsymbol{i} = \begin{bmatrix} 0.0 & 1.0 & 2.0 & 3.0 \end{bmatrix}^\top$$

$$\operatorname{softmax}(\boldsymbol{x}/1.0)^\top \boldsymbol{i} = 1.2606$$

$$\operatorname{softmax}(\boldsymbol{x}/0.8)^\top \boldsymbol{i} = 1.1894$$

$$\operatorname{softmax}(\boldsymbol{x}/0.6)^\top \boldsymbol{i} = 1.1037$$

$$\operatorname{softmax}(\boldsymbol{x}/0.4)^\top \boldsymbol{i} = 1.0274$$

$$\operatorname{softmax}(\boldsymbol{x}/0.2)^\top \boldsymbol{i} = 1.0004$$

max

- A similar trick applies to finding the maximum value of a vector:

max

- A similar trick applies to finding the maximum value of a vector:
  - Use $\text{softmax}(\boldsymbol{x})$ as an approximate one-hot $\arg\max$, and dot product with the vector $\boldsymbol{x}$.

max

- A similar trick applies to finding the maximum value of a vector:
  - Use $\text{softmax}(\boldsymbol{x})$ as an approximate one-hot $\arg\max$, and dot product with the vector $\boldsymbol{x}$.
  - As temperature $T \to 0$, $\text{softmax}(\boldsymbol{x}/T)^\top \boldsymbol{x} \to \max(\boldsymbol{x})$.

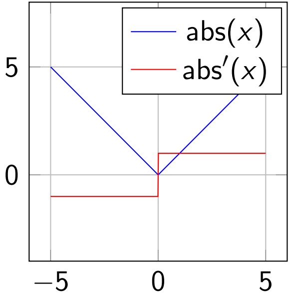
max

- A similar trick applies to finding the maximum value of a vector:
  - Use $\text{softmax}(\boldsymbol{x})$ as an approximate one-hot $\arg\max$, and dot product with the vector $\boldsymbol{x}$.
  - As temperature $T \to 0$, $\text{softmax}(\boldsymbol{x}/T)^\top \boldsymbol{x} \to \max(\boldsymbol{x})$.

$$
\begin{aligned}
\boldsymbol{x} &= [\ 1.1 \quad 4.0 \quad -0.1 \quad 2.3\ ]^\top \\
\text{softmax}(\boldsymbol{x}/1.0)^\top \boldsymbol{x} &= 3.571 \\
\text{softmax}(\boldsymbol{x}/0.8)^\top \boldsymbol{x} &= 3.736 \\
\text{softmax}(\boldsymbol{x}/0.6)^\top \boldsymbol{x} &= 3.881 \\
\text{softmax}(\boldsymbol{x}/0.4)^\top \boldsymbol{x} &= 3.974 \\
\text{softmax}(\boldsymbol{x}/0.2)^\top \boldsymbol{x} &= 3.999
\end{aligned}
$$

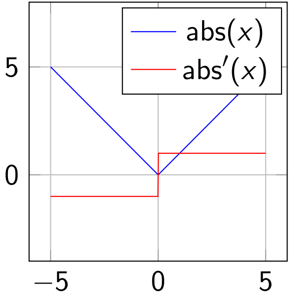
L1 norm

- L1 norm is the sum of absolute values of a vector



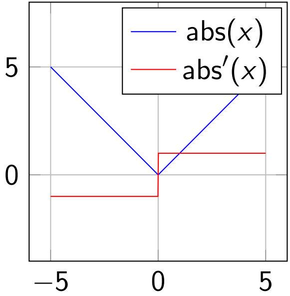
L1 norm

- L1 norm is the sum of absolute values of a vector
- We've seen that an L1 norm regulariser can induce sparsity in a model



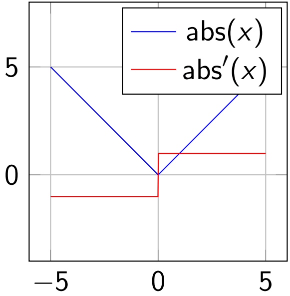
L1 norm

- L1 norm is the sum of absolute values of a vector
- We've seen that an L1 norm regulariser can induce sparsity in a model
- abs is continuous and differentiable almost everywhere, but...



L1 norm

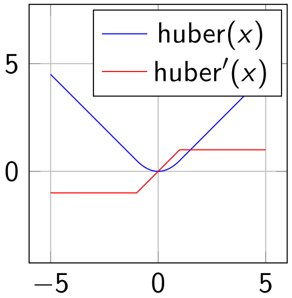
- L1 norm is the sum of absolute values of a vector
- We've seen that an L1 norm regulariser can induce sparsity in a model
- abs is continuous and differentiable almost everywhere, but...
- unlike ReLU, the gradients left and right of the discontinuity point in equal and opposite directions
    - This can cause oscillations that prevent or hamper learning



Relaxing the L1 norm

- Huber loss (aka Smooth L1 loss) relaxes L1 by mixing it with L2 near the origin:

$$z_i = \begin{cases} 0.5(x_i - y_i)^2, & \text{if } |x_i - y_i| < 1 \\ |x_i - y_i| - 0.5, & \text{otherwise} \end{cases}$$

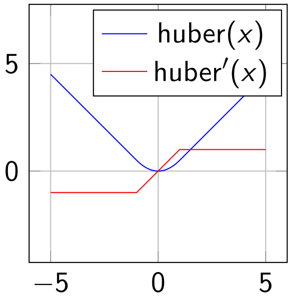


Relaxing the L1 norm

- Huber loss (aka Smooth L1 loss) relaxes L1 by mixing it with L2 near the origin:

$$z_i = \begin{cases} 0.5(x_i - y_i)^2, & \text{if } |x_i - y_i| < 1 \\ |x_i - y_i| - 0.5, & \text{otherwise} \end{cases}$$

- In both cases gradients reduce in magnitude and switch direction smoothly which can lead to much less oscillation.



Differentiable Sampling

The reparameterisation trick

Sampling from a diagonal-covariance Gaussian

Sampling from a categorical distribution: Gumbel Softmax

The Straight-Through operator

The Straight-Through operator: implementation