# TTIC 31230, Fundamentals of Deep Learning

David McAllester, Winter 2020

# Shannon's Source Coding Theorem

# Huffman Coding

# Perils of Continuous Information Theory

## Source Coding Theorem

Consider a probability distribution Pop on a finite set $S$.

Consider a code $C$ assigning a bit string code word $C(y_1, \ldots, y_B)$ to each possible batch of $B$ elements with $y_i \sim \mathrm{Pop}$.

Source coding theorem: As $B \to \infty$ the optimal coding uses exactly $H(\mathrm{Pop})$ bits per batch element.

## Prefix Free Codes

Let $S$ be a finite set.

Let $C$ be assignment of a bit string $C(y)$ to each $y \in S$.

$C$ is called *prefix-free* if for $x \neq y$ we have that $C(x)$ is not a prefix of $C(y)$.

A concatenation of sequence of prefix-free code words can be uniquely segmented (parsed) back into a sequence of code words.

## Prefix-Free Codes as Trees and as Probabilities

A prefix-free code defines a binary branching tree — branch on the first code bit, then the second, and so on.

The leaves of this tree are labeled with the elements of $S$.

The code defines a probability distribution on $S$ by randomly selecting branches.

We have $P_C(y) = 2^{-|C(y)|}$.

## The Source Coding Theorem

(1) There exists a prefix-free code $C$ such that

$$|C(y)| <= (-\log_2 \mathrm{Pop}(y)) + 1$$

and hence

$$E_{y\sim\mathrm{Pop}}|C(y)| \leq H(\mathrm{Pop}) + 1$$

(2) For any prefix-free code $C$

$$E_{y\sim\mathrm{Pop}}\,|C(y)| = E_{y\sim\mathrm{Pop}} - \ln\; P_C(y) = H(\mathrm{Pop}, P_C) \geq H(\mathrm{Pop})$$

## Code Construction

We construct a code by iterating over $y \in S$ in order of decreasing probability (most likely first).

For each $y$ select a code word $C(y)$ (a tree leaf) with length (depth)

$$|C(y)| = \lceil -\log_2 \mathrm{Pop}(y) \rceil$$

and where $C(y)$ is not an extension of (under) any previously selected code word.

## Code Existence Proof

At any point before coding all elements of $S$ we have

$$\sum_{y \in \text{Defined}} 2^{-|C(y)|} \leq \sum_{y \in \text{Defined}} \text{Pop}(y) < 1$$

Therefore there exists an infinite descent into the tree that misses all previous code words.

Hence there exists a code word $C(x)$ not under any previous code word with $|C(x)| = \lceil -\log_2 \text{Pop}(y) \rceil$.

Furthermore $C(x)$ is at least as long as all previous code words and hence $C(x)$ is not a prefix of any previously selected code word.

# Huffman Coding Produces an Optimal Code For Finite Distributions

Maintain a list of trees $T_1, \ldots, T_N$ where each leaf of each tree is labeled with a population element $y$.

We will write $y \in T_i$ to mean that some leaf of $T_i$ is labeled with $y$ and for $y \in T_i$ write $d(y, T_i)$ for the depth of the leaf labeled with $y$ in the tree $T_i$.

Initially each tree is just one root node labeled with a single $y$ value and every $y$ value labels some tree.

## Tree Weight

We define the weight of a free $T_i$ by

$$W(T_i) = \sum_{y \in T_i} \mathrm{Pop}(y) d(y, T_i)$$

The Huffman coding algorithm repeatedly merges the two trees of lowest weight into a single tree until all trees are merged.

When all trees are merged the weight of the final tree is the expected code length.

# Optimality of Huffman Coding

**Theorem**: The Huffman code $T$ for Pop gives an optimal code $C$ — for any other tree $T'$ defining code $C'$ we have

$$E_{y \sim \mathrm{Pop}}\ |C(y)| \leq E_{y \sim \mathrm{Pop}} |C'(y)|$$

**Invariant:** The merge operation preserves the invariant that there exists an optimal tree including all the subtrees on the list.

## The Merge Operation Preserves the Invariant

Assume there exists an optimal tree containing the given subtrees.

Consider the two subtrees $T_i$ and $T_j$ of minimal weight. Without loss of generality we can assume that $T_i$ is at least as deep as $T_j$.

Swapping the sibling of $T_i$ for $T_j$ brings $T_i$ and $T_j$ together. This can only improve the average depth (next slide).

## Why The Swap Can Only Improve the Tree

We can swap a heavier deeper tree for a shallower lighter tree.

For a subtree $T_i$ of an optimal tree $T$ let $d(T_i)$ be the depth of $T_i$ in $T$.

The contribution of $T_i$ to the average code length is $d(T_i)W(T_i)$.

For $d(T_1) \geq d(T_2)$ and $W(T_1) \geq W(T_2)$ we have

$$d(T_1)W(T_1) + d(T_2)W(T_2) \geq d(T_1)W(T_2) + d(T_2)W(T_1)$$

## Perils of Differential Entropy

Consider a continuous density $p(x)$. For example

$$p(x) = \frac{1}{\sqrt{2\pi}\ \sigma}\ e^{\frac{-x^2}{2\sigma^2}}$$

Differential entropy is defined as

$$H(p) \doteq \int \left( \ln \frac{1}{p(x)} \right) p(x) dx = E_{x \sim p}\ - \ln p(x)$$

# Perils of Differential Entropy

$$
\begin{aligned}
H(\mathcal{N}(0,\sigma)) &= +\int \left(\ln(\sqrt{2\pi}\sigma) + \frac{x^2}{2\sigma^2}\right) p(x)dx \\
&= \ln(\sigma) + \ln(\sqrt{2\pi}) + \frac{1}{2}
\end{aligned}
$$

$$
\lim_{\sigma \to 0} H(N(0,\sigma)) = -\infty
$$

.

Hence differential entropy then depends on the choice of units — a distributions on lengths will have a different entropy when measuring in inches than when measuring in feet.

## Differential Cross Entropy can Diverge to $-\infty$

Consider the unsupervised training object.

$$\Phi^* = \operatorname*{argmin}_{\Phi} E_{y\sim \text{train}} \; -\ln p_\Phi(y)$$

The training set is finite (discrete).

For each $y$ the density $p_\Phi(y)$ can go to infinity.

This will drive the cross entropy training loss to $-\infty$.

## Differential Cross Entropy can Diverge to $-\infty$

$$\Phi^* = \operatorname*{argmin}_\Phi E_{y\sim\mathrm{train}}\; -\ln p_\Phi(y)$$

To avoid divergence to $-\infty$ we can enforce an upper bound on the density $p_\Phi(y)$.

We will see how this is implicitly done in continuous variational auto-encoders (VAEs).

## Differential Entropy is Actually Infinite

An actual real number carries an infinite number of bits.

Consider quantizing the real numbers into bins.

A continuous probability densisty $p$ assigns a probability $p(B)$ to each bin.

As the bin size decreases toward zero the entropy of the bin distribution increases toward $\infty$.

A meaningful convention is that $H(p) = +\infty$ for any continuous density $p$.

## Differential KL-divergence is Meaningful

$$KL(p, q) = \int \left( \ln \frac{p(x)}{q(x)} \right) p(x) dx$$

This integral can be computed by dividing the real numbers into bins and computing the $KL$ divergence between the distributions on bins.

The KL divergence between the bin distribution typically approaches a finite limit as the bin size goes to zero.

Unlike entropy, differential KL divergence is always non-negative. But as in the discrete case, it can be infinite.

## Mutual Information

For two random variables $x$ and $y$ there is a distribution on pairs $(x, y)$ determined by the population distribution.

Mutual information is a KL divergence and hence differential mutual information is meaningful.

$$
\begin{aligned}
I(x, y) &\doteq KL(p(x, y), p(x)p(y)) \\
&= E_{x,y} \ln \frac{p(x, y)}{p(x)p(y)}
\end{aligned}
$$

## The Data Processing Inequality

For continuous $y$ and $z$ with $z = f(y)$ we get that $H(z)$ can be either larger or smaller than $H(y)$ (consider $z = ay$ for $a > 1$ vs. $a < 1$).

However, mutual information is a KL divergence and is more meaningful than entropy and for $z = f(y)$ we do have

$$I(x, z) \leq I(x, y)$$

**END**