First Principles of Deep Learning

Steve Avsec

Illinois Institute of Technology

March 4, 2024

Overview

Entropy

Entropy is a measure of disorder or uncertainty in a system.

Entropy

Entropy is a measure of disorder or uncertainty in a system.

For a probability distribution, p:

$$H(p) = -E_p[log(X)] = -\sum_{j=1}^{n} p_j \log(p_j)$$

Entropy

Entropy is a measure of disorder or uncertainty in a system.

For a probability distribution, p:

$$H(p) = -E_p[log(X)] = -\sum_{j=1}^{n} p_j \log(p_j)$$

Maximized when p is the uniform distribution.

KL Divergence and Cross-Entropy

Let p and q be two probability distributions. Then

$$D_{KL}(p||q) = E_p[\log(\frac{p(X)}{q(X)})] = \sum_{j=1}^{n} p_j \log(p_j) - \sum_{j=1}^{n} p_j \log(q_j)$$

KL Divergence and Cross-Entropy

Let p and q be two probability distributions. Then

$$D_{KL}(p||q) = E_p[\log(\frac{p(X)}{q(X)})] = \sum_{j=1}^{n} p_j \log(p_j) - \sum_{j=1}^{n} p_j \log(q_j)$$

KL Divergence is a measure of "surprise": If q is the assumed distribution and p is the actual, $D_{KL}(p||q)$ can be interpreted as a difference between these realities.

KL Divergence and Cross-Entropy

Let p and q be two probability distributions. Then

$$D_{KL}(p||q) = E_p[\log(\frac{p(X)}{q(X)})] = \sum_{j=1}^{n} p_j \log(p_j) - \sum_{j=1}^{n} p_j \log(q_j)$$

KL Divergence is a measure of "surprise": If q is the assumed distribution and p is the actual, $D_{KL}(p||q)$ can be interpreted as a difference between these realities.
The *cross-entropy* of two distributions p and q is defined by

$$H(p, q) = -E_p[\log(q(X))] = -\sum_{j=1}^{n} p_j \log(q_j)$$

KL Divergence and Cross-Entropy

Let p and q be two probability distributions. Then

$$D_{KL}(p||q) = E_p[\log(\frac{p(X)}{q(X)})] = \sum_{j=1}^{n} p_j \log(p_j) - \sum_{j=1}^{n} p_j \log(q_j)$$

KL Divergence is a measure of "surprise": If q is the assumed distribution and p is the actual, $D_{KL}(p||q)$ can be interpreted as a difference between these realities.
The *cross-entropy* of two distributions p and q is defined by

$$H(p, q) = -E_p[\log(q(X))] = -\sum_{j=1}^{n} p_j \log(q_j)$$

Often, we want to minimize $D_{KL}(p||q)$, but because

$$H(p, q) = H(p) + D_{KL}(p||q)$$

we often minimize $H(p, q)$.

Maximum Likelihood Estimation

Let **X** be a data set drawn from some unknown probability distribution p_{data}

Maximum Likelihood Estimation

Let $\mathbf{X}$ be a data set drawn from some unknown probability distribution p_{data}

Let $p_{model}(\mathbf{x}, \theta)$ be a parameterized family of models.

Maximum Likelihood Estimation

Let $\mathbf{X}$ be a data set drawn from some unknown probability distribution p_{data}

Let $p_{model}(\mathbf{x}, \theta)$ be a parameterized family of models.

$p_{model}(\mathbf{x}, \theta)$ is an estimate to $p_{data}(\mathbf{x})$ for a fixed θ.

MLE Continued

$$\begin{aligned}
\theta_{ML} &= \text{argmax}_{\theta} p_{model}(\mathbf{X}, \theta) \\
&= \text{argmax}_{\theta} \prod_{j=1}^{N} p_{model}(\mathbf{x}_j, \theta) \\
&= \text{argmax} \max_{\theta} \sum_{j=1}^{N} \log(p_{model}(\mathbf{x}, \theta)) \\
&= \text{argmax} \max_{\theta} E_{\hat{p}_{data}} \log(p_{model}(\mathbf{x}))
\end{aligned}$$

MLE Continued

$$\begin{aligned}
\theta_{ML} &= \text{argmax}_\theta p_{model}(\mathbf{X}, \theta) \\
&= \text{argmax}_\theta \prod_{j=1}^{N} p_{model}(\mathbf{x}_j, \theta) \\
&= \text{argmax} \max_\theta \sum_{j=1}^{N} \log(p_{model}(\mathbf{x}, \theta)) \\
&= \text{argmax} \max_\theta E_{\hat{p}_{data}} \log(p_{model}(\mathbf{x}))
\end{aligned}$$

and we can conclude that the maximum likelihood estimator θ_{ML} coincides with the minimizer of $H(p_{data}, p_{model})$

Linear Models vs. Deep Learning

Goal: Optimize

$$f : \mathbb{R}^d \to \mathbb{R}$$

according to some loss function given a data set $(\mathbf{X}, \mathbf{y})$.

Linear Models vs. Deep Learning

Goal: Optimize

$$f : \mathbb{R}^d \to \mathbb{R}$$

according to some loss function given a data set $(\mathbf{X}, \mathbf{y})$.

Linear models:

$$f(\mathbf{x}) = \sum_{j=1}^{K} c_j f_j(\mathbf{x})$$

where $f_j \in \mathcal{F}$ and $\mathcal{F}$ is some appropriate class fo functions.

Linear Models vs. Deep Learning

Goal: Optimize

$$f : \mathbb{R}^d \to \mathbb{R}$$

according to some loss function given a data set $(\mathbf{X}, \mathbf{y})$.

Linear models:

$$f(\mathbf{x}) = \sum_{j=1}^{K} c_j f_j(\mathbf{x})$$

where $f_j \in \mathcal{F}$ and $\mathcal{F}$ is some appropriate class fo functions.

Deep learning:

$$f(\mathbf{x}) = f^{(K)} \circ \ldots \circ f^{(2)} \circ f^{(1)}(\mathbf{x})$$

where $f^{(k)} : \mathbb{R}^{d_k} \to \mathbb{R}^{d_{k+1}}$ and $d_{K+1} = 1$.

Some Jargon

- The model described here is a feedforward neural network.

Some Jargon

- The model described here is a feedforward neural network.
- The functions $f^{(k)}$ are often referred to as ”hidden” layers.

Some Jargon

- The model described here is a feedforward neural network.
- The functions $f^{(k)}$ are often referred to as "hidden" layers.
- The *depth* of a neural network is the number of layers (K).

Some Jargon

- The model described here is a feedforward neural network.
- The functions $f^{(k)}$ are often referred to as "hidden" layers.
- The *depth* of a neural network is the number of layers (K).
- The *width* of a neural network is the maximum dimension $\max_{1 \leq k \leq K} d_k$.

Some building blocks

- Loss functions (also commonly called cost functions in the DL literature).

Some building blocks

- Loss functions (also commonly called cost functions in the DL literature).
- Activation functions
  - ReLU (rectified linear unit)

$$g(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

Some building blocks

- Loss functions (also commonly called cost functions in the DL literature).
- Activation functions
  - ReLU (rectified linear unit)

$$g(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

  - Hyperbolic tangent

$$g(x) = \tanh(x)$$

Output Units

- Linear (Gaussian):

$$\mathbf{y} = \mathbf{W}^t\mathbf{h} + \mathbf{b}$$

Output Units

- Linear (Gaussian):

$$\mathbf{y} = \mathbf{W}^t\mathbf{h} + \mathbf{b}$$

- Sigmoid (Bernoulli):

$$y = \sigma(\mathbf{w}^t\mathbf{h} + b)$$

where $\sigma(x) = \frac{1}{1+e^{-x}}$.

Output Units

- Linear (Gaussian):

$$\mathbf{y} = \mathbf{W}^t\mathbf{h} + \mathbf{b}$$

- Sigmoid (Bernoulli):

$$y = \sigma(\mathbf{w}^t\mathbf{h} + b)$$

where $\sigma(x) = \frac{1}{1+e^{-x}}$.

- Softmax (Multinomial):

$$\text{softmax}(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{k=1}^{K} e^{z_k}}$$

where $z_i = \log(y = i|\mathbf{h}) = \mathbf{W}^t\mathbf{h} + \mathbf{b}$

Gradients

Feedforward neural networks lose *convexity* of their loss function with respect to model parameters.

Gradients

Feedforward neural networks lose *convexity* of their loss function with respect to model parameters.

This loss of convexity means that solutions are sensitive to initial parameters and convergence is not guaranteed.

Gradients

Feedforward neural networks lose *convexity* of their loss function with respect to model parameters.

This loss of convexity means that solutions are sensitive to initial parameters and convergence is not guaranteed.

It is called *saturation* when a function becomes very flat and its gradient becomes close to 0.

Architecture

Layers are typically broken down by

$$\mathbf{h}^{(1)} = g^{(1)}(\mathbf{W}^{(1)t}\mathbf{x} + \mathbf{b}^{(1)})$$
$$\mathbf{h}^{(2)} = g^{(2)}(\mathbf{W}^{(2)t}\mathbf{h}^{(1)} + \mathbf{b}^{(2)})$$

et cetera.

Architecture

Layers are typically broken down by

$$\mathbf{h}^{(1)} = g^{(1)}(\mathbf{W}^{(1)t}\mathbf{x} + \mathbf{b}^{(1)})$$
$$\mathbf{h}^{(2)} = g^{(2)}(\mathbf{W}^{(2)t}\mathbf{h}^{(1)} + \mathbf{b}^{(2)})$$

et cetera.

Theorem (The Universal Approximation Theorem)

A feedforward neural network with at least one hidden layer and a linear output layer and any "reasonable" activation function can approximate any "reasonable" function.

Architecture

Layers are typically broken down by

$$\mathbf{h}^{(1)} = g^{(1)}(\mathbf{W}^{(1)t}\mathbf{x} + \mathbf{b}^{(1)})$$
$$\mathbf{h}^{(2)} = g^{(2)}(\mathbf{W}^{(2)t}\mathbf{h}^{(1)} + \mathbf{b}^{(2)})$$

et cetera.

Theorem (The Universal Approximation Theorem)

A feedforward neural network with at least one hidden layer and a linear output layer and any "reasonable" activation function can approximate any "reasonable" function.

The downside: While such a neural network can *represent* any function, due to optimization issues with non-convex functions, there is no guarantee that such a representation can be learned.

The Chain Rule

Let

- $\mathbf{x} \in \mathbb{R}^d$,
- $g : \mathbb{R} \to \mathbb{R}$ be some activation function,
- $\mathbf{W}$ be an d-dim vector,
- and b be some offset.

The Chain Rule

Let

- $\mathbf{x} \in \mathbb{R}^d$,
- $g : \mathbb{R} \to \mathbb{R}$ be some activation function,
- $\mathbf{W}$ be an d-dim vector,
- and b be some offset.

If

$$f(\mathbf{x}) = g(\mathbf{w}^t\mathbf{x} + b)$$

then

The Chain Rule

Let

- $\mathbf{x} \in \mathbb{R}^d$,
- $g : \mathbb{R} \to \mathbb{R}$ be some activation function,
- $\mathbf{W}$ be an d-dim vector,
- and b be some offset.

If

$$f(\mathbf{x}) = g(\mathbf{w}^t\mathbf{x} + b)$$

then

$$\nabla f(\mathbf{x}) = g'(\mathbf{w}^t\mathbf{x} + b) \times \mathbf{w}^t$$

Backprop

Let l denote the depth of the feedforward neural network.

1. Let $\mathbf{g} = \nabla_{\hat{\mathbf{y}}} L(\mathbf{y}, \hat{\mathbf{y}})$.

Backprop

Let l denote the depth of the feedforward neural network.

1. Let $\mathbf{g} = \nabla_{\hat{\mathbf{y}}} L(\mathbf{y}, \hat{\mathbf{y}})$.

2. For $k = l, l-1, \ldots, 1$:

Backprop

Let l denote the depth of the feedforward neural network.

1. Let $\mathbf{g} = \nabla_{\hat{\mathbf{y}}} L(\mathbf{y}, \hat{\mathbf{y}})$.

2. For $k = l, l-1, \ldots, 1$:
   1. $\mathbf{g} = \nabla_{\mathbf{a}^{(k)}} L = \mathbf{g} \circ g'(\mathbf{a}^{(k)})$

Backprop

Let l denote the depth of the feedforward neural network.

1. Let $\mathbf{g} = \nabla_{\hat{\mathbf{y}}} L(\mathbf{y}, \hat{\mathbf{y}})$.
2. For $k = l, l-1, \ldots, 1$:
   1. $\mathbf{g} = \nabla_{\mathbf{a}^{(k)}} L = \mathbf{g} \circ g'(\mathbf{a}^{(k)})$
   2. Gradients wrt $\mathbf{b}^{(k)}$ and $\mathbf{W}^{(k)}$ can be expressed by
      - $\nabla_{\mathbf{b}^{(k)}} L = \mathbf{g} + \text{reg}.$
      - $\nabla_{\mathbf{W}^{(k)}} L = \mathbf{g} + \text{reg}.$
      - $\mathbf{g} = \mathbf{W}^{(k)t}\mathbf{g}$