Feedforward Neural Networks

Michael Collins, Columbia University

Recap: Log-linear Models

A log-linear model takes the following form:

$$p(y|x;v) = \frac{\exp\left(v \cdot f(x,y)\right)}{\sum_{y' \in \mathcal{Y}} \exp\left(v \cdot f(x,y')\right)}$$

- $f(x,y)$ is the representation of (x,y)
- Advantage: $f(x,y)$ is highly flexible in terms of the features that can be included
- Disadvantage: can be hard to design features by hand
- Neural networks allow **the representation itself to be learned**. Recent empirical results across a broad set of domains have shown that learned representations in neural networks can give very significant improvements in accuracy over hand-engineered features.

Example 1: The Language Modeling Problem

- w_i is the i'th word in a document
- Estimate a distribution $p(w_i|w_1, w_2, \ldots w_{i-1})$ given previous "history" $w_1, \ldots, w_{i-1}$.
- E.g., $w_1, \ldots, w_{i-1} =$

Third, the notion "grammatical in English" cannot be identified in any way with the notion "high order of statistical approximation to English". It is fair to assume that neither sentence (1) nor (2) (nor indeed any part of these sentences) has ever occurred in an English discourse. Hence, in any statistical

Example 2: Part-of-Speech Tagging

Hispaniola/NNP quickly/RB became/VB an/DT important/JJ base/?? from which Spain expanded its empire into the rest of the Western Hemisphere .

- There are many possible tags in the position ??
{NN, NNS, Vt, Vi, IN, DT, . . . }

- The task: model the distribution

$$p(t_i|t_1, \ldots, t_{i-1}, w_1 \ldots w_n)$$

where t_i is the i'th tag in the sequence, w_i is the i'th word

Overview

- Basic definitions
- Stochastic gradient descent
- Defining the input to a neural network
- A single neuron
- A single-layer feedforward network
- Motivation: the XOR problem

An Alternative Form for Log-Linear Models

Old form:

$$p(y|x;v) = \frac{\exp\left(v \cdot f(x,y)\right)}{\sum_{y' \in \mathcal{Y}} \exp\left(v \cdot f(x,y')\right)} \tag{1}$$

New form:

$$p(y|x;v) = \frac{\exp\left(v(y) \cdot f(x) + \gamma_y\right)}{\sum_{y' \in \mathcal{Y}} \exp\left(v(y') \cdot f(x) + \gamma_{y'}\right)} \tag{2}$$

- Feature vector $f(x)$ maps input x to $f(x) \in \mathbb{R}^D$.
- Parameters: $v(y) \in \mathbb{R}^D$, $\gamma_y \in \mathbb{R}$ for each $y \in \mathcal{Y}$.
- The score $v \cdot f(x,y)$ in Eq. 1 has essentially been replaced by $v(y) \cdot f(x) + \gamma_y$ in Eq. 2.
- We will use v to refer to the set of all parameter vectors and bias values: that is, $v = \{(v(y), \gamma_y) : y \in \mathcal{Y}\}$

Introducing Learned Representations

$$p(y|x;\theta,v) = \frac{\exp\left(v(y)\cdot\phi(x;\theta)+\gamma_y\right)}{\sum_{y'\in\mathcal{Y}}\exp\left(v(y')\cdot\phi(x;\theta)+\gamma_{y'}\right)} \tag{3}$$

- Replaced $f(x)$ by $\phi(x;\theta)$ where θ are some additional parameters of the model
- The parameter values θ will be estimated from training examples: the representation of x is then "learned"
- In this lecture we'll show how feedforward neural networks can be used to define $\phi(x;\theta)$.

Definition (Multi-Class Feedforward Models)

A multi-class feedforward model consists of:

- A set $\mathcal{X}$ of possible inputs. A finite set $\mathcal{Y}$ of possible labels. A positive integer D specifying the number of features in the feedforward representation.
- A parameter vector θ defining the *feedforward parameters* of the network. We use Ω to refer to the set of possible values for θ.
- A function $\phi : \mathcal{X} \times \Omega \to \mathbb{R}^D$ that maps any (x, θ) pair to a "feedforward representation" $\phi(x; \theta)$.
- For each label $y \in \mathcal{Y}$, a parameter vector $v(y) \in \mathbb{R}^D$, and a bias value $\gamma_y \in \mathbb{R}$.

For any $x \in \mathcal{X}$, $y \in \mathcal{Y}$, $\quad p(y|x; \theta, v) = \dfrac{\exp\left(v(y) \cdot \phi(x; \theta) + \gamma_y\right)}{\sum_{y' \in \mathcal{Y}} \exp\left(v(y') \cdot \phi(x; \theta) + \gamma_{y'}\right)}$

Two Questions

- How can we define the feedforward representation $\phi(x; \theta)$?
- Given training examples (x^i, y^i) for $i = 1 \ldots n$, how can we train the parameters θ and v?

Overview

- Basic definitions
- Stochastic gradient descent
- Defining the input to a neural network
- A single neuron
- A single-layer feedforward network
- Motivation: the XOR problem

A Simple Version of Stochastic Gradient Descent

Inputs: Training examples (x^i, y^i) for $i = 1 \ldots n$. A feedforward representation $\phi(x; \theta)$. An integer T specifying the number of updates. A sequence of learning rate values $\eta^1 \ldots \eta^T$ where each $\eta^t > 0$.

Initialization: Set v and θ to random parameter values.

A Simple Version of Stochastic Gradient Descent (Continued)

Algorithm:

- For $t = 1 \dots T$
  - Select an integer i uniformly at random from $\{1 \dots n\}$
  - Define $L(\theta, v) = -\log p(y_i | x_i; \theta, v)$
  - For each parameter θ_j, $\theta_j = \theta_j - \eta^t \times \frac{dL(\theta,v)}{d\theta_j}$
  - For each label y, for each parameter $v_k(y)$, $v_k(y) = v_k(y) - \eta^t \times \frac{dL(\theta,v)}{dv_k(y)}$
  - For each label y, $\gamma_y = \gamma_y - \eta^t \times \frac{dL(\theta,v)}{d\gamma_y}$

Output: parameters θ and v

Overview

- Basic definitions
- Stochastic gradient descent
- Defining the input to a neural network
- A single neuron
- A single-layer feedforward network
- Motivation: the XOR problem

Defining the Input to a Feedforward Network

- Given an input x, we need to define a function $f(x) \in \mathbb{R}^d$ that specifies the input to the network
- In general it is assumed that the representation $f(x)$ is "simple", not requiring careful hand-engineering.
- The neural network will take $f(x)$ as input, and will produce a representation $\phi(x; \theta)$ that depends on the input x and the parameters θ.

Linear Models

We could build a log-linear model using $f(x)$ as the representation:

$$p(y|x; v) = \frac{\exp\{v(y) \cdot f(x) + \gamma_y\}}{\sum_{y'} \exp\{v(y') \cdot f(x) + \gamma_{y'}\}} \tag{4}$$

This is a "linear" model, because the score $v(y) \cdot f(x)$ is linear in the input features $f(x)$. The general assumption is that a model of this form will perform poorly or at least non-optimally. Neural networks enable "non-linear" models that often perform at much higher levels of accuracy.

An Example: Digit Classification

- Task is to map an image x to a label y
- Each image contains a hand-written digit in the set $\{0, 1, 2, \ldots 9\}$
- The representation $f(x)$ simply represents pixel values in the image.
- For example if the image is 16×16 grey-scale pixels, where each pixel takes some value indicating how bright it is, we would have $d = 256$, with $f(x)$ just being the list of values for the 256 different pixels in the image.
- Linear models under this representation perform poorly, neural networks give much better performance

Simplifying Notation

- From now on assume that $x = f(x)$: that is, the input x is already defined as a vector
- This will simplify notation
- But remember that when using a neural network you will have to define a representation of the inputs

Overview

- Basic definitions
- Stochastic gradient descent
- Defining the input to a neural network
- A single neuron
- A single-layer feedforward network
- Motivation: the XOR problem

A Single Neuron

- A neuron is defined by a weight vector $w \in \mathbb{R}^d$, a bias $b \in \mathbb{R}$, and a transfer function $g : \mathbb{R} \to \mathbb{R}$.
- The neuron maps an input vector $x \in \mathbb{R}^d$ to an output h as follows:

$$h = g(w \cdot x + b)$$

- The vector $w \in \mathbb{R}^d$ and scalar $b \in \mathbb{R}$ are parameters of the model, which are learned from training examples.

Transfer Functions

- It is important that the transfer function $g(z)$ is **non-linear**
- A linear transfer function would be

$$g(z) = \alpha \times z + \beta$$

for some constants α and β

The Rectified Linear Unit (ReLU) Transfer Function

The ReLU transfer function is defined as

$$g(z) = \{z \text{ if } z \geq 0, \text{ or } 0 \text{ if } z < 0\}$$

Or equivalently, $g(z) = \max\{0, z\}$

It follows that the derivative is

$$\frac{dg(z)}{dz} = \{1 \text{ if } z > 0, \text{ or } 0 \text{ if } z < 0, \text{ or undefined if } z = 0\}$$

The tanh Transfer Function

The tanh transfer function is defined as

$$g(z) = \frac{e^{2z} - 1}{e^{2z} + 1}$$

It can be shown that the derivative is

$$\frac{dg(z)}{dz} = (1 - g(z))^2$$

Calculating Derivatives

Given

$$h = g(w \cdot x + b)$$

it will be useful to calculate derivatives

$$\frac{dh}{dw_j}$$

for the parameters $w_1, w_2, \ldots w_d$, and also

$$\frac{dh}{db}$$

for the bias parameter b

Calculating Derivatives (Continued)

We can use the *chain rule of differentiation*. First introduce an intermediate variable $z \in \mathbb{R}$:

$$z = w \cdot x + b, \quad h = g(z)$$

Then by the chain rule we have

$$\frac{dh}{dw_j} = \frac{dh}{dz} \times \frac{dz}{dw_j} = \frac{dg(z)}{dz} \times x_j$$

Here we have used $\frac{dh}{dz} = \frac{dg(z)}{dz}$, $\frac{dz}{dw_j} = x_j$.

Calculating Derivatives (Continued)

We can use the *chain rule of differentiation*. First introduce an intermediate variable $z \in \mathbb{R}$:

$$z = w \cdot x + b, \quad h = g(z)$$

Then by the chain rule we have

$$\frac{dh}{db} = \frac{dh}{dz} \times \frac{dz}{db} = \frac{dg(z)}{dz} \times 1$$

Here we have used $\frac{dh}{dz} = \frac{dg(z)}{dz}$, and $\frac{dz}{db} = 1$.

Definition (Single-Layer Feedforward Representation)

A single-layer feedforward representation consists of the following:

- An integer d specifying the input dimension. Each input to the network is a vector $x \in \mathbb{R}^d$.
- An integer m specifying the number of hidden units.
- A parameter matrix $W \in \mathbb{R}^{m \times d}$. We use the vector $W_k \in \mathbb{R}^d$ for each $k \in \{1, 2, \ldots m\}$ to refer to the k'th row of W.
- A vector $b \in \mathbb{R}^m$ of bias parameters.
- A transfer function $g : \mathbb{R} \to \mathbb{R}$. Common choices are $g(x) = \mathsf{ReLU}(x)$ or $g(x) = \tanh(x)$.

Definition (Single-Layer Feedforward Representation (Continued))

We then define the following:

- For $k = 1 \dots m$, the input to the k'th neuron is $z_k = W_k \cdot x + b_k$.
- For $k = 1 \dots m$, the output from the k'th neuron is $h_k = g(z_k)$.
- Finally, define the vector $\phi(x; \theta) \in \mathbb{R}^m$ as $\phi_k(x; \theta) = h_k$ for $k = 1 \dots m$. Here θ denotes the parameters $W \in \mathbb{R}^{m \times d}$ and $b \in \mathbb{R}^m$. Hence θ contains $m \times (d + 1)$ parameters in total.

Some Intuition

The neural network employs m units, each with their own parameters W_k and b_k, and these neurons are used to construct a "hidden" representation $h \in \mathbb{R}^m$.

Matrix Form

We can for example replace the operation

$$z_k = W_k \cdot x + b \qquad \text{for } k = 1 \dots m$$

with

$$z = Wx + b$$

where the dimensions are as follows (note that an m-dimensional column vector is equivalent to a matrix of dimension $m \times 1$):

$$\underbrace{z}_{m\times 1} = \underbrace{\underbrace{W}_{m\times d}\ \underbrace{x}_{d\times 1}}_{m\times 1} + \underbrace{b}_{m\times 1}$$

Definition (Single-Layer Feedforward Representation (Matrix Form))

A single-layer feedforward representation consists of the following:

- An integer d specifying the input dimension. Each input to the network is a vector $x \in \mathbb{R}^d$.
- An integer m specifying the number of hidden units.
- A matrix of parameters $W \in \mathbb{R}^{m \times d}$.
- A vector of bias parameters $b \in \mathbb{R}^m$
- A transfer function $g : \mathbb{R}^m \to \mathbb{R}^m$. Common choices would be to define $g(z)$ to be a vector with components $\mathsf{ReLU}(z_1), \mathsf{ReLU}(z_2), \ldots, \mathsf{ReLU}(z_m)$ or $\tanh(z_1), \tanh(z_2), \ldots, \tanh(z_m)$.

Definition (Single-Layer Feedforward Representation (Matrix Form) (Continued))

We then define the following:

- The vector of inputs to the hidden layer $z \in \mathbb{R}^m$ is defined as $z = Wx + b$.
- The vector of outputs from the hidden layer $h \in \mathbb{R}^m$ is defined as $h = g(z)$
- Finally, define $\phi(x; \theta) = h$. Here the parameters θ contain the matrix W and the vector b.
- It follows that

$$\phi(x; \theta) = g(Wx + b)$$

Overview

- Basic definitions
- Stochastic gradient descent
- Defining the input to a neural network
- A single neuron
- A single-layer feedforward network
- Motivation: the XOR problem

A Motivating Example: the XOR Problem (from Deep Learning, Ian Goodfellow and Yoshua Bengio and Aaron Courville)

We will assume a training set where each label is in the set $\mathcal{Y} = \{-1, +1\}$, and there are 4 training examples, as follows:

$$
\begin{aligned}
x^1 &= [0,0], \quad y^1 = -1 \\
x^2 &= [0,1], \quad y^2 = 1 \\
x^3 &= [1,0], \quad y^3 = 1 \\
x^4 &= [1,1], \quad y^4 = -1
\end{aligned}
$$

A Useful Lemma

Assume we have a model of the form

$$p(y|x;v) = \frac{\exp\{v(y)\cdot x + \gamma_y\}}{\sum_y \exp\{v(y)\cdot x + \gamma_y\ \}}$$

and the set of possible labels is $\mathcal{Y} = \{-1, +1\}$. Then for any x,

$$p(+1|x;v) > 0.5$$

if and only if

$$u \cdot x + \gamma > 0$$

where $u = v(+1) - v(-1)$ and $\gamma = \gamma_{+1} - \gamma_{-1}$. Similarly for any x,

$$p(-1|x;v) > 0.5$$

if and only if $u \cdot x + \gamma < 0$

Proof: We have

$$\begin{aligned} p(+1|x;v) &= \frac{\exp\{v(+1)\cdot x+\gamma_{+1}\}}{\exp\{v(+1)\cdot x+\gamma_{+1}\}+\exp\{v(-1)\cdot x+\gamma_{-1}\}} \\ &= \frac{1}{1+\exp\{-(u\cdot x+\gamma)\}} \end{aligned}$$

It follows that $p(+1|x;v) > 0.5$ if and only if $\exp\{-(u\cdot x+\gamma)\} < 1$ from which it follows that $u\cdot x+\gamma > 0$. A similar proof applies to the condition $p(-1|x;v) > 0.5$. □

Theorem

Assume we have examples (x^i, y^i) *for* $i = 1 \ldots 4$ *as defined above. Assume we have a model of the form*

$$p(y|x; v) = \frac{\exp\{v(y) \cdot x + \gamma_y\}}{\sum_y \exp\{v(y) \cdot x + \gamma_y\}}$$

Then there are no parameter settings for $v(+1), v(-1), \gamma_{+1}, \gamma_{-1}$ *such that*

$$p(y^i|x^i; v) > 0.5 \quad \text{for } i = 1 \ldots 4$$

Proof Sketch:

From the previous lemma, $p(y^i = 1|x^i; v) > 0.5$ if and only if

$$u \cdot x^i + \gamma > 0$$

where $u = v(+1) - v(-1)$ and $\gamma = \gamma_{+1} - \gamma_{-1}$.
Similarly $p(y^i = -1|x^i; v) > 0.5$ if and only if

$$u \cdot x^i + \gamma < 0$$

where $u = v(+1) - v(-1)$ and $\gamma = \gamma_{+1} - \gamma_{-1}$.
Hence to satisfy $p(y^i|x^i; v) > 0.5$ for $i = 1 \ldots 4$, there must exist parameters v and γ such that

$$u \cdot [0, 0] + \gamma < 0 \quad (5)$$
$$u \cdot [0, 1] + \gamma > 0 \quad (6)$$
$$u \cdot [1, 0] + \gamma > 0 \quad (7)$$
$$u \cdot [1, 1] + \gamma < 0 \quad (8)$$

The Constraints can not be Satisfied

$$
\begin{aligned}
u \cdot [0,0] + \gamma &< 0 \\
u \cdot [0,1] + \gamma &> 0 \\
u \cdot [1,0] + \gamma &> 0 \\
u \cdot [1,1] + \gamma &< 0
\end{aligned}
$$

The Constraints can not be Satisfied

$$
\begin{aligned}
u \cdot [0,0] + \gamma &< 0 \\
u \cdot [0,1] + \gamma &> 0 \\
u \cdot [1,0] + \gamma &> 0 \\
u \cdot [1,1] + \gamma &< 0
\end{aligned}
$$

Theorem

Assume we have examples (x^i, y^i) *for* $i = 1 \ldots 4$ *as defined above. Assume we have a model of the form*

$$p(y|x;\theta,v) = \frac{\exp\{v(y) \cdot \phi(x;\theta) + \gamma_y\}}{\sum_y \exp\{v(y) \cdot \phi(x;\theta) + \gamma_y\}}$$

where $\phi(x;\theta)$ *is defined by a single layer neural network with* $m = 2$ *hidden units, and the ReLU*(z) *activation function. Then there are parameter settings for* $v(0), v(1), \gamma_0, \gamma_1, \theta$ *such that*

$$p(y^i|x^i;v) > 0.5 \quad \text{for } i = 1 \ldots 4$$

Proof Sketch: Define $W_1 = [1, 1]$, $W_2 = [1, 1]$, $b_1 = 0$, $b_2 = -1$. Then for each input x we can calculate the value for the vectors z and h corresponding to the inputs and the outputs from the hidden layer:

$$
\begin{aligned}
x = [0, 0] &\Rightarrow z = [0, -1] \Rightarrow h = [0, 0] \\
x = [1, 0] &\Rightarrow z = [1, 0] \Rightarrow h = [1, 0] \\
x = [0, 1] &\Rightarrow z = [1, 0] \Rightarrow h = [1, 0] \\
x = [1, 1] &\Rightarrow z = [2, 1] \Rightarrow h = [2, 1]
\end{aligned}
$$

Proof Sketch (continued)

Hence to satisfy $p(y^i|x^i; v) > 0.5$ for $i = 1 \ldots 4$, there must exist parameters $u = v(+1) - v(-1)$ and $\gamma = \gamma_{+1} - \gamma_{-1}$ such that

$$
\begin{aligned}
u \cdot [0, 0] + \gamma &< 0 && (9)\\
u \cdot [1, 0] + \gamma &> 0 && (10)\\
u \cdot [1, 0] + \gamma &> 0 && (11)\\
u \cdot [2, 1] + \gamma &< 0 && (12)
\end{aligned}
$$

It can be verified that $u = [1, -2], \gamma = -0.5$ satisifies these contraints.