Feedforward and recurrent neural networks

Karl Stratos

Broadly speaking, a "neural network" simply refers to a composition of linear and nonlinear functions. We will review two most basic types of neural networks.

1 Feedforward neural networks

In feedfoward networks, messages are passed forward only. Cycles are forbidden.

1.1 Single-layer network

The parameter corresponding to the first (and the only) layer is $W \in \mathbb{R}^{d_1 \times d_0}$. Let $f : \mathbb{R}^{d_1} \to \mathbb{R}^{d_1}$ be a differentiable function. Given an input $x \in \mathbb{R}^{d_0}$, the network outputs

$$y := f(Wx)$$

To train the network, we minimize a loss function $l : \mathbb{R}^{d_1} \times \mathbb{R}^{d_1} \to \mathbb{R}$ over labeled examples (x, y).

1.1.1 Delta rule

The gradient of the squared loss $l(x, y) := (1/2) \left|\left|y - x\right|\right|^2$ with respect to W is

$$\frac{\partial}{\partial W} \left(\frac{1}{2} \left|\left|y - f(Wx)\right|\right|^2 \right) = \Big((f(Wx) - y) \odot f'(Wx) \Big) x^\top \tag{1}$$

where $\odot$ is the entry-wise multiplication operator. This gradient formula is sometimes called the "delta rule".

1.1.2 Linear regression, logistic regression

The network computes linear regression if $f(z) := z$ and $l(x, y) := (1/2) \left|\left|y - x\right|\right|^2$.

For output $y \in \{\pm 1\}$, the network computes logistic regression if $f(z) := 1/(1 + \exp(-z))$ and $l(x, y) := \log(1 + \exp(-yx))$.[1]

[1] The network is known as the perceptron if $f(z) := \text{sign}(z)$ and $l(x, y) := [[x = y]]$: it is usually trained with the so-called "perceptron-style" algorithm whose updates look very similar to stochastic gradient descent updates. It can be shown that the perceptron updates always achieve 100% accuracy on the training data (if possible) within a bounded number of misclassifications. But the perceptron updates cannot be derived directly from the delta rule, since f is not differentiable at 0.

1.2 Multi-layer network

A network with L layers has a parameter $W^{(l)} \in \mathbb{R}^{d_l \times d_{l-1}}$ and a differentiable function $f^{(l)} : \mathbb{R}^{d_l} \to \mathbb{R}^{d_l}$ corresponding to the l-th layer. Given an input $x \in \mathbb{R}^{d_0}$, the network outputs

$$y := a^{(L)}$$

where each $a^{(l)} \in \mathbb{R}^{d_l}$ is defined recursively from the base case $a^{(0)} := x$ as follows:

$$\begin{aligned} z^{(l)} &:= W^{(l)} a^{(l-1)} \\ a^{(l)} &:= f^{(l)}(z^{(l)}) \end{aligned}$$

Again, to train the network, we minimize a loss function $l : \mathbb{R}^{d_L} \times \mathbb{R}^{d_L} \to \mathbb{R}$ over labeled examples (x, y).

1.2.1 Gradients of the squared loss

The gradient of the squared loss on (x, y) with respect to $W^{(L)}$ is

$$\frac{\partial}{\partial W^{(L)}} \left(\frac{1}{2} \left|\left| y - a^{(L)} \right|\right|^2 \right) = \left(\left(a^{(L)} - y \right) \odot f^{(L)\prime} \left(z^{(L)} \right) \right) \left(a^{(L-1)} \right)^\top \tag{2}$$

Note that the form mirrors the delta rule (1) because $a^{(L)} = f^{(L)} \left(W^{(L)} a^{(L-1)} \right)$ where $a^{(L-1)}$ does not involve $W^{(L)}$. By defining the "error term"

$$\delta^{(L)} := \left(a^{(L)} - y \right) \odot f^{(L)\prime} \left(z^{(L)} \right)$$

we can simplify (2) as $\delta^{(L)} \left(a^{(L-1)} \right)^\top$. Similarly, the gradient with respect to $W^{(l)}$ for $l < L$ can be verified to be $\delta^{(l)} \left(a^{(l-1)} \right)^\top$ where

$$\delta^{(l)} := f^{(l)\prime}(z^{(l)}) \odot \left(W^{(l+1)\top} \delta^{(l+1)} \right)$$

Computing all gradients in a multi-layer network in this manner is commonly known as "backpropagation", which is just a special case of automatic differentiation. For concreteness, here is the backpropagation algorithm for an L-layer feedforward network with the squared loss:

BackpropationSquaredLoss
Input: labeled example $(x, y) \in \mathbb{R}^{d_L} \times \mathbb{R}^{d_L}$, parameters $\{W^{(l)}\}_{l=1}^{L}$
Output: $\overline{W}^{(l)} := \frac{\partial}{\partial W^{(l)}} (1/2) \left|\left| y - a^{(L)} \right|\right|^2$ for $l = 1 \ldots L$

(Feedforward phase)

- Set $a^{(0)} \leftarrow x$, and for $l = 1 \ldots L$ compute:

$$z^{(l)} \leftarrow W^{(l)} a^{(l-1)} \qquad\qquad a^{(l)} \leftarrow f^{(l)}(z^{(l)})$$

(Backpropagation phase)

- Set $\delta^{(L)} \leftarrow \left(a^{(L)} - y \right) \odot f^{(L)\prime} \left(z^{(L)} \right)$, and for $l = L - 1 \ldots 1$ compute:

$$\delta^{(l)} \leftarrow f^{(l)\prime}(z^{(l)}) \odot \left(W^{(l+1)\top} \delta^{(l+1)} \right)$$

- Set $\overline{W}^{(l)} \leftarrow \delta^{(l)} \left(a^{(l-1)} \right)^\top$ for $l = 1 \ldots L$.

1.3 Example: language models

Let V be the number of distinct word types. The goal of a language model is to estimate the probability of a word given its history/context.

1.3.1 Bengio et al. (2003)

Bengio et al. (2003) propose the following three-layer feedforward network. As input, it receives binary vectors $e_{x_1} \ldots e_{x_n} \in \{0,1\}^V$ indicating an ordered sequence of n words $x_1 \ldots x_n$. As output, it produces $u \in \mathbb{R}^V$ where u_y is the probability of the $(n+1)$-th word being y. The network is parametrized as follows:[2]

- Layer 1: matrix $W^{(1)} \in \mathbb{R}^{d_1 \times V}$, identity function
- Layer 2: matrix $W^{(2)} \in \mathbb{R}^{d_2 \times nd_1}$, entry-wise $\tanh_i(z) = \tanh(z)$
- Layer 3: matrix $W^{(3)} \in \mathbb{R}^{V \times d_2}$, $\mathrm{softmax}_i(z) = \exp(z_i)/\sum_j \exp(z_j)$

Then it defines the probability of word y following words $x_1 \ldots x_n$ as:

$$p(y|x_1 \ldots x_n) = \mathrm{softmax}_y \left(W^{(3)} \tanh \left(W^{(2)} \begin{bmatrix} W^{(1)} e_{x_1} \\ \vdots \\ W^{(1)} e_{x_n} \end{bmatrix} \right) \right) \tag{3}$$

The parameters of the network can be estimated from a sequence of words $x_1 \ldots x_N$ by maximizing the log likelihood:

$$\max_{W^{(1)}, W^{(2)}, W^{(3)}} \log \sum_{i=n+1}^{N} p(x_i | x_{i-n} \ldots x_{i-1})$$

1.3.2 Mikolov et al. (2013)

Mikolov et al. (2013) further simplify the model of Bengio et al. (2003) and propose the following continuous bag-of-words (CBOW) model. As input, it receives binary vectors $e_{x_{-m}} \ldots e_{x_{-1}}, e_{x_1} \ldots e_{x_m} \in \{0,1\}^V$ indicating m words to the left $x_{-m} \ldots x_{-1}$ and m words to the right $x_1 \ldots x_m$. As output, it produces $u \in \mathbb{R}^V$ where u_y is the probability of the current word being y. The network is parametrized as follows:

- Layer 1: matrix $W^{(1)} \in \mathbb{R}^{d_1 \times V}$, identity function
- Layer 2: matrix $W^{(2)} \in \mathbb{R}^{V \times d_1}$, $\mathrm{softmax}_i(z) = \exp(z_i)/\sum_j \exp(z_j)$

Let $\boldsymbol{x} \in \mathbb{R}^{d_1}$ be the x-th column of $W^{(1)}$. Let $\boldsymbol{y} \in \mathbb{R}^{d_1}$ be the y-th row of $W^{(2)}$. Then it defines the probability of word y given context words $x_{-m} \ldots x_{-1}$ and $x_1 \ldots x_m$ as:

$$\begin{aligned} p(y|x_{-m} \ldots x_{-1}, x_1 \ldots x_m) &= \mathrm{softmax}_y \left(W^{(2)} W^{(1)} (e_{x_{-m}} + \cdots + e_{x_{-1}} + e_{x_1} + \cdots + e_{x_m}) \right) \\ &= \frac{\exp(\boldsymbol{y}^\top (\boldsymbol{x_{-m}} + \cdots + \boldsymbol{x_{-1}} + \boldsymbol{x_1} + \cdots + \boldsymbol{x_m}))}{\sum_{y'} \exp(\boldsymbol{y'}^\top (\boldsymbol{x_{-m}} + \cdots + \boldsymbol{x_{-1}} + \boldsymbol{x_1} + \cdots + \boldsymbol{x_m}))} \end{aligned} \tag{4}$$

A variant of CBOW, known as the skip-gram model, has the same parameters but defines the probability of a context word y given a current word x:

$$p(y|x) = f_y^{(2)} \left(W^{(2)} W^{(1)} e_x \right) = \frac{\exp(\boldsymbol{y}^\top \boldsymbol{x})}{\sum_{y'} \exp(\boldsymbol{y'}^\top \boldsymbol{x}))} \tag{5}$$

[2] The original version has an additional paramater linking layer 1 to 3.

1.3.3 Hierarchical softmax trick

All the above language models (3, 4, 5) have a softmax layer at the top, which means that the complexity of computing gradients is $O(V)$ due to normalization. In the case of (4, 5), this complexity can be reduced to $O(\log V)$ with a trick known as the "hierarchical softmax". We will focus on the skip-gram model for illustration.

The trick assumes a binary tree over the vocabulary (thus the tree has V leaf nodes) as an additional input. This tree defines a path from the root for each word y. Let $L(y)$ be the length of this path. For $j = 1 \ldots L(y) - 1$, define:

$$\mathrm{dir}(y, j+1) := \begin{cases} +1 & \text{if the } (j+1)\text{-th node is the left child of the } j\text{-th node} \\ -1 & \text{if the } (j+1)\text{-th node is the right child of the } j\text{-th node} \end{cases}$$

Instead of having a vector $\mathbf{y} \in \mathbb{R}^{d_1}$ corresponding to each context word y as in (5), we will have a vector corresponding to each *internal node* in the tree. Specifically, for every word y in the vocabulary, for every $j = 1 \ldots L(y) - 1$, we will have a vector $\mathbf{y}(j) \in \mathbb{R}^{d_1}$ corresponding to the j-th node in the path from the root to y. Then we replace the definition of $p(y|x)$ from the original softmax (5) with the following:

$$p(y|x) = \prod_{j=1}^{L(y)-1} \sigma(\mathrm{dir}(y, j+1) \times \boldsymbol{y}(j)^\top \boldsymbol{x}) \tag{6}$$

where $\sigma(z) = 1/(1 + \exp(-z))$ is the sigmoid function. This particular construction makes (6) a proper distribution. Express the binary tree as a set of spans $S := \{(i,j)\}$ where each (i,j) has an associated vector $\boldsymbol{y}_{i,j}$ and a split point $k_{i,j}$. Then it can be seen that $\sum_y p(y|x) = \pi(1, V)$ where π is computed recursively as follows:

$$\pi(i,j) := \begin{cases} 1 & \text{if } i = j \\ \sigma(\boldsymbol{y}_{i,j}^\top x)\pi(i, k_{i,j}) + \sigma(-\boldsymbol{y}_{i,j}^\top x)\pi(k_{i,j}+1, j) & \text{otherwise} \end{cases}$$

Because $\sigma(z) + \sigma(-z) = 1$, each $\pi(i,j) = 1$.

2 Recurrent neural networks

In recurrent neural networks (RNNs), a notion of time is introduced. The input at time step t depends on an output from time step $t - 1$.

2.1 Simple RNN

A simple RNN (eponymously named the "simple RNN") has parameters $W^{(1)} \in \mathbb{R}^{d_1 \times d_0}$, $V^{(1)} \in \mathbb{R}^{d_1 \times m}$, and $W^{(2)} \in \mathbb{R}^{d_2 \times d_1}$. Let $f^{(1)} : \mathbb{R}^{d_1} \to \mathbb{R}^{d_1}$ and $f^{(2)} : \mathbb{R}^{d_2} \to \mathbb{R}^{d_2}$ be differentiable functions, and let $h_0 = 0 \in \mathbb{R}^m$. For time step $t \geq 1$, it receives input $x_t \in \mathbb{R}^{d_0}$ and produces output $y_t \in \mathbb{R}^{d_2}$ as

$$\begin{aligned} h_t &= f^{(1)}\left(W^{(1)}x_t + V^{(1)}h_{t-1}\right) \\ y_t &= f^{(2)}\left(W^{(2)}h_t\right) \end{aligned}$$

Observe that if we fix $V^{(1)} = 0$, we end up with a two-layer feedforward network at each time step.

2.1.1 Training a simple RNN

At each time step t, the "unrolled" simple RNN for the input sequence $(x_1 \dots x_t)$ and the output y_t is a giant feedforward network, namely:

$$y_n = f^{(2)}\left(W^{(2)} f^{(1)}\left(W^{(1)} x_T + \cdots + V^{(1)} f^{(1)}\left(W^{(1)} x_1 + V^{(1)} h_0\right)\right)\right)$$

Thus we can perform backpropagation on this unrolled network. Note that unlike the standard feedforward network, the gradient of $W^{(1)}$ will need to take into account multiple instances of $W^{(1)}$.

Given a labeled sequence $(x_1, y_1) \dots (x_n, y_n)$, we can control how much to unroll for training. This is known as "backpropagation through time".

BackpropationThroughTime
Input: $(x_1, y_1) \dots (x_n, y_n)$, how much to unroll $T \leq n$, RNN parameters

- Set $h_0 \leftarrow 0$.
- For $t = T \dots n$,
  - For $t' = t - T + 2 \dots t$, unroll the RNN on h_{t-T}, $(x_{t-T+1} \dots x_{t'})$, and $y_{t'}$ and perform backpropagation.
  - Store h_{t-T+1} (computed in the previous step).

If $T = 1$, backpropagation through time degenerates to standard backpropagation on:

$$y_t = f^{(2)}\left(W^{(2)} f^{(1)}\left(W^{(1)} x_t + V^{(1)} \hat{h}_t\right)\right)$$

where $\hat{h}_t$ is considered a constant input (so this network uses each parameter once).

A Derivative rules

Let $f, g : \mathbb{R} \to \mathbb{R}$ be differentiable functions with first derivatives $f', g' : \mathbb{R} \to \mathbb{R}$.

Product rule $\quad \frac{\partial}{\partial x}\left(f(x)g(x)\right) = f'(x)g(x) + f(x)g'(x)$

Quotient rule $\quad \frac{\partial}{\partial x}\left(\frac{f(x)}{g(x)}\right) = \frac{f'(x)g(x) - f(x)g'(x)}{g^2(x)}$

Chain rule $\quad \frac{\partial}{\partial x} f(g(x)) = f'(g(x))g'(x)$