Self-Normalizing Neural Networks

Hurdles when training a neural network

Christoph Lange

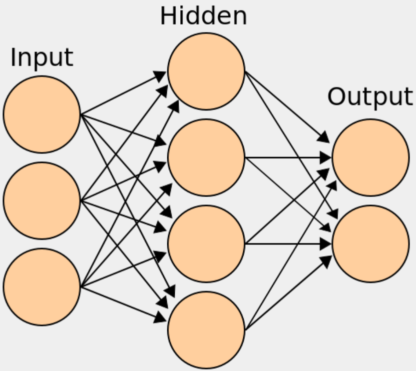
January 12, 2021

Backpropagation

The paper Self-Normalizing Neural Networks focuses on Fully Connected Neural Networks.
one layer:

- $y(x) = f(Wx + b)$ represents one layer
  - $x \in \mathbb{R}^n$ activations current layer
  - $y \in \mathbb{R}^m$ activations next layer
  - $W \in \mathbb{R}^{m \times n}$
  - $b \in \mathbb{R}^m$ the bias vector, assume $b = 0$ for simplicity
  - $f \in C^0$ continous non-linear function
- want to learn W for each layer

Figure: A fully connected neural network (picture from here)



For one layer we have:

$$y(x) = f(z(x)) \qquad z(x) = Wx \tag{1}$$

Derivatives with respect to the parameters W and the activations x:

$$\frac{dy}{dW} = \frac{dy}{dz} \cdot \frac{dz}{dW} = f'(z(x)) \cdot \frac{dg}{dW} = f'(Wx) \cdot x^T \tag{2}$$

$$\frac{dy}{dx} = \frac{dy}{dz} \cdot \frac{dz}{dx} = f'(z(x)) \cdot \frac{dz}{dx} = f'(Wx) \cdot W \tag{3}$$

For two layers we assume they have the same activation function f:

$$y^{(1)}(x) = f(z^{(1)}(x)) \qquad z^{(1)}(x) = W^{(1)}x \tag{4}$$

$$y^{(2)}(x) = f(z^{(2)}(x)) \qquad z^{(2)}(x) = W^{(2)}x^{(2)} = W^{(2)}y^{(1)} \tag{5}$$

The derivatives within one layer are the same:

$$\frac{dy^{(i)}}{dW^{(i)}} = f'(W^{(i)}x^{(i)}) \cdot x^{(i)T} \qquad \frac{dy^{(i)}}{dx^{(i)}} = f'(W^{(i)}x^{(i)}) \cdot W^{(i)} \tag{6}$$

Across two layers we get:

$$\frac{dy^{(2)}}{dW^{(1)}} = \frac{dy^{(2)}}{dy^{(1)}} \cdot \frac{dy^{(1)}}{dW^{(1)}} = \frac{dy^{(2)}}{dx^{(2)}} \cdot \frac{dy^{(1)}}{dW^{(1)}} \tag{7}$$

$$\frac{dy^{(2)}}{dx^{(1)}} = \frac{dy^{(2)}}{dy^{(1)}} \cdot \frac{dy^{(1)}}{dx^{(1)}} = \frac{dy^{(2)}}{dx^{(2)}} \cdot \frac{dy^{(1)}}{dx^{(1)}} \tag{8}$$

Derivative across N layers is :

$$\frac{dy^{(N)}}{dW^{(1)}} = \frac{dy^{(N)}}{dx^{(N)}} \cdot \ldots \cdot \frac{dy^{(2)}}{dx^{(2)}} \cdot \frac{dy^{(1)}}{dW^{(1)}} \tag{9}$$

Why is it called Backpropagation?

$$\frac{dy^{(N)}}{dW^{(2)}} = \underbrace{\frac{dy^{(N)}}{dx^{(N)}} \cdot \ldots \cdot \frac{dy^{(3)}}{dx^{(3)}}}_{=:u^{(2)}} \cdot \frac{dy^{(2)}}{dW^{(2)}} = u^{(2)} \cdot \frac{dy^{(2)}}{dW^{(2)}} \tag{10}$$

From the perspective of layer 2, $u^{(2)}$ is the derivative that comes from up front. And the components are travelling backwards.

Issues during the Training Process

This is the Fashion MNIST dataset

- contains of 70000 cloth items
- one picture and one label per item, i.e. "Dress", "Sneaker", "Coat"
- each picture is gray-scaled and 28 x 28 pixels
- example use case for now
- similar to classifying bacteria
- complex problem needs a more complex deeper network

Example

Let's oversimplify:

- each layer consists of one neuron
- the derivative of each layer is $\frac{dy^{(i)}}{dx^{(i)}} = c$ constant
- we have $N = 10$ layers

What happens to the derivative of the first layer parameters $W^{(1)}$?

$$\frac{dy^{(10)}}{dW^{(1)}} = \frac{dy^{(10)}}{dx^{(10)}} \cdot \ldots \cdot \frac{dy^{(2)}}{dx^{(2)}} \cdot \frac{dy^{(1)}}{dW^{(1)}}$$
$$= c \cdot \ldots \cdot c \cdot \frac{dy^{(1)}}{dW^{(1)}}$$

$$\frac{dy^{(N)}}{dW^{(1)}} = c^{N-1} \cdot \frac{dy^{(1)}}{dW^{(1)}}$$

For a huge number of layers N:

- $\frac{dy^{(N)}}{dW^{(1)}}$ get really big for $c > 1$
- for $c \approx 1$ the derivative $\frac{dy^{(N)}}{dW^{(1)}} \approx \frac{dy^{(1)}}{dW^{(1)}}$
- when $c < 1$ the gradient vanishes $\frac{dy^{(N)}}{dW^{(1)}} \approx 0$

For more general cases of multiplying the Jacobians

$$\frac{dy^{(N)}}{dW^{(1)}} = \frac{dy^{(N)}}{dx^{(N)}} \cdot \ldots \cdot \frac{dy^{(2)}}{dx^{(2)}} \cdot \frac{dy^{(1)}}{dW^{(1)}}$$

the largest eigenvalues determine the convergence behavior.

Self-normalizing Neural Networks (SNNs)

Setup

Let us look at one layer:

- $y(x) = f(Wx)$
- $x \in \mathbb{R}^n$ activations current layer
- $y \in \mathbb{R}^m$ activations next layer
- $W \in \mathbb{R}^{m \times n}$
- $f \in C^0$ continous non-linear function

- mean of the activations $\mu = \mathbb{E}(x_i)$
- variance of the activations $\nu = \mathrm{Var}(x_i)$
- let w^i be the i-th row of W
- the mean of the i-th row weights is $\omega^i := \sum_j w_{ij}$
- accordingly the second moment $\tau^i := \sum_j w_{ij}^2$

How do mean μ and variance ν of the activations change to the next layer?

$$\begin{pmatrix}\mu\\\nu\end{pmatrix} \mapsto \begin{pmatrix}\tilde{\mu}\\\tilde{\nu}\end{pmatrix} = g\begin{pmatrix}\mu\\\nu\end{pmatrix}$$

Definition
A neural network is called **self-normalizing** if there is a domain $\Omega = \{(\mu, \nu) \in \mathbb{R} \mid \mu \in [\mu_{\text{min}}, \mu_{\text{max}}], \nu \in [\nu_{\text{min}}, \nu_{\text{max}}]\}$ and a mapping $g : \Omega \to \Omega$ such that

- $g(\Omega) \subset \Omega$ is a contraction
- g has a stable and attracting fixpoint $(\mu^*, \nu^*) \in \Omega$

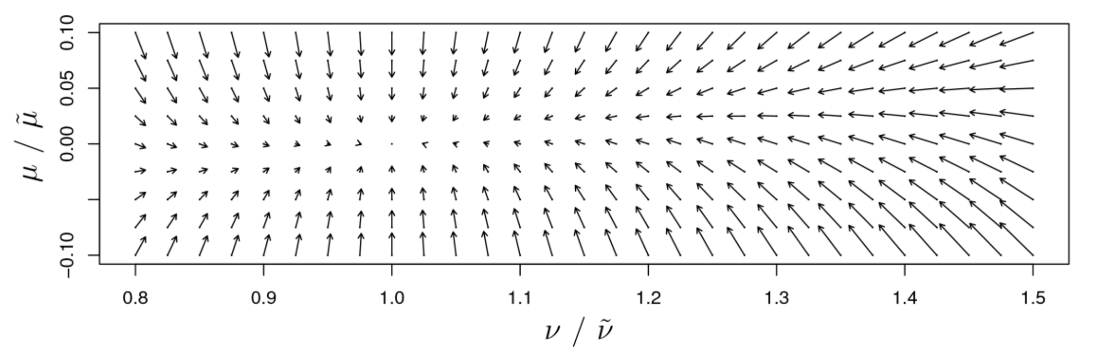


Figure: Assuming the weight of each row w^i to be normalized ($\omega = 0, \tau = 1$), we observe an attracting fixpoint ($\mu^* = 0, \nu^* = 1$). We need to pick the activation function accordingly.

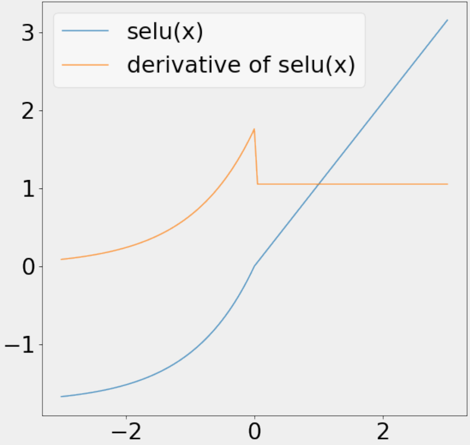
The key component to a SNN is to pick the activation function $f(x)$ to be

$$\mathrm{selu}(x) = \lambda \begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases} \tag{11}$$

a **scaled exponential linear unit**
Assuming

- normalized weights: $\omega = 0, \tau = 1$
- $z(x)$ to be normally distributed

specifies $\alpha \approx 1.6733$ and $\lambda \approx 1.0507$.



advantages

- filters noise $X - WH$
- basis vectors h_i capture joint phenomena
- basis vectors h_i can be interpreted to draw conclusions about the dataset
- imposing $WW^T = I$ makes NMF equivalent to K-means clustering

disadvantages

- Finding coordinates W requires solving $\min_W \|X - WH\|_F^2$.
- limited to nonnegative matricies

Thank you
Questions?

Link to Slides

References

Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter.
Self-normalizing neural networks, 2017.