Adaptive Activation Functions for Deep Networks

Michael Dushkoff, Raymond Ptucha
Rochester Institute of Technology

IS&T International Symposium on Electronic Imaging 2016
Computational Imaging
Feb 16, 2016

Convolutional Neural Networks have Revolutionized Computer Vision and Pattern Recognition

Taigman et al., 2014
Simonyan et al., 2014
Szegedy et al., 2014
Karpathy et al., 2014
Deep Learning - Surpassing The Visual Cortex’s Object Detection and Recognition Capability Top-5 error on ImageNet

Outline

- Introduction
- Background
- Methodology
- Datasets
- Results
- Conclusions
The Human Brain

• We’ve learned more about the brain in the last 5 years than we have learned in the last 5000 years!
• It controls every aspect of our lives, but we still don’t understand exactly how it works.

Neurons in Brain vs. Computer

• The brain has billions of cells called neurons.
• Each is connected to up to 10K others, forming a network of 100T connections.
• If the sum of inputs > threshold, the neuron will fire.

• Artificial neurons, inspired by biology, compute a weighted sum of inputs, then pass through a non-linear activation function.
• Artificial neural networks are formed by connecting thousands to millions of these artificial neurons together.
Three Most Common Activation Functions

**Sigmoid**

\[ h_θ(x) = \frac{1}{1 + e^{-θx}} \]

- Constrains \( 0 \leq \text{out} \leq 1 \)
- Gradient saturates to 0
- Inputs centered on 0, but output centered on 0.5
- Gradient easy to calculate.

**Tanh**

\[ h_θ(x) = \frac{e^{θx} - e^{-θx}}{e^{θx} + e^{-θx}} \]

- Constrains \(-1 \leq \text{out} \leq 1\)
- Gradient saturates to 0
- Input and output centered on 0
- Gradient easy to calculate

**Rectified Linear Units (ReLU)**

\[ h_θ(x) = \max(0, x) \]

- Unbounded upper range with no gradient saturation
- Empirical faster and better result
- Neurons can “die” if allowed to grow unconstrained

---

Tanh vs. ReLU on CIFAR-10 dataset

[Krizhevsky’12]

ReLU reaches 25% error 6x faster!

Note: Learning rates optimized for each, no regularization, four layer CNN.
Lots of other Activation Functions

- Non-monotonic functions [Dawson’92]
- Adaptive cubic spline [Vecci’98]
- Adaptive parameters [Nakayama’98]
- Monotonic and non-monotonic mixtures [Wong’02]
- Gated adaptive functions [Scheler’04]
- Periodic functions [Kang’05]
- Maxout & Leaky ReLU’s [Goodfellow ‘13]
- Adaptive Leaky ReLU’s [He’15]

Contributions

- Prior work either was constrained to small networks, or forced all nodes in a layer to have the same activation function.
- This work learns functions on a node by node basis (for images, every pixel can have own activation function), and experiments on larger datasets.
- This work finds that allowing nodes to adaptively learn their own activation functions results in faster convergence and higher accuracies.
Traditional Artificial Neuron

Note, \( x_0 \) is the bias unit, \( x_0 = 1 \)

\[
x = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix}, \quad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}
\]

Activation function

\[
h_\theta(x) = g(x_0 \theta_0 + x_1 \theta_1 + \ldots + x_n \theta_n) = g\left(\sum_{i=0}^{n} x_i \theta_i\right)
\]

• Where:
  – \( x \) is the input
  – \( h_\theta(x) \) is the output
  – \( g \) is a activation function

Proposed Method

• Adaptive activation functions are defined by:

\[
v = \sum_{i=1}^{N} f_i(u)l(g_i)
\]

• Where:
  – \( u \) is the input
  – \( v \) is the output
  – \( f_i \) is a unique activation function
  – \( l \) is a convex (sigmoid) limiting function
  – \( g_i \) is the “gating factor” which is learned
Proposed Method

• Adaptive activation functions are defined by:

\[ v = \sum_{i=1}^{N} f_i(u) l(g_i) \]

• Where:
  – \( u \) is the input
  – \( v \) is the output
  – \( f_i \) is a unique activation function
  – \( l \) is a convex (sigmoid) limiting function
  – \( g_i \) is the “gating factor” which is learned

Architecture

• VGG-like network structure [1]
  – Modified forward and back propagation to handle adaptive activation functions
  – Batch normalization after each convolution [2]

(64×64 input to 100 class example)

Technical Approach

• Adaptive functions were used only on certain layers
  — First $n$ layers vs. last $n$ layers

Datasets

CIFAR100:
  — 100 classes
  — 32×32×3 pixels/image
  — 500 images training, 100 testing/class

CalTech256:
  — 257 classes
  — 300×200×3 pixels/image*
  — 80 to 827 images/class

*resampled to 64×64×3
Results – CIFAR 100

• ReLU Baseline Results (57.4%)

Results – CIFAR 100

• Adaptive case – First 7 Adaptive (51.5%)
Results – CIFAR 100

• Adaptive case – Last 7 Adaptive (59.8%)

![Graph showing training and testing accuracies over epochs.]

Results – CIFAR 100

• Comparison (Baseline vs. Adaptive)

![Graph showing CIFAR 100 training and testing accuracies.]

3/22/2016 Dushkoff, Ptucha Ef’16
Additional CIFAR 100 Results

• Usage Statistics

Additional CIFAR 100 Results

• Randomly selected adaptive functions
Results – Caltech 256

- Baseline Results (32.5%)

- Adaptive Results (32.6%)

Last 5 layers
Results – Caltech 256

• Comparison (Baseline vs. Adaptive)

Conclusions

• Adaptive accuracies are improved over ReLU in CIFAR 100, but not in Caltech 256.
• For both datasets, training time is faster using adaptive activation functions.
• Additional training strategies can be implemented in order to combat the problem of the adaptive function parameters taking over the optimization problem.
Next Steps

• Implement new training method (ON/OFF training with gradient scaling)

• Apply non-monotonic functions to the adaptive definition to allow for more complex non-linear behavior

Thank you!!

rwpeec@rit.edu