Approximate Functions

VLC = $\iiint$ Vision Learning Control $dV dL dC$

Going Deep

Jonathon Hare

Vision, Learning and Control
University of Southampton

Overview

- No free lunch and universal approximation
- Why go deep?
- Problems of going deep
- Some fixes:
  - Improving gradient flow with skip connections
  - Regularising with Dropout

No Free Lunch

- Statistical learning theory claims that a machine can generalise well from a finite training set.

[1] or perhaps more generally rules which are not certain

No Free Lunch

- Statistical learning theory claims that a machine can generalise well from a finite training set.
- This contradicts basic inductive reasoning which says to derive a rule describing every member of a set one must have information about every member.

[1] or perhaps more generally rules which are not certain

No Free Lunch

- Statistical learning theory claims that a machine can generalise well from a finite training set.
- This contradicts basic inductive reasoning which says to derive a rule describing every member of a set one must have information about every member.
- Machine learning avoids this problem by learning probabilistic[1] rules which are *probably* correct about *most* members of the set they concern.

[1] or perhaps more generally rules which are not certain

No Free Lunch

- Statistical learning theory claims that a machine can generalise well from a finite training set.
- This contradicts basic inductive reasoning which says to derive a rule describing every member of a set one must have information about every member.
- Machine learning avoids this problem by learning probabilistic[1] rules which are *probably* correct about *most* members of the set they concern.
- But, **no free lunch theorem** states that every possible classification machine has the *same error* when averaged over *all possible* data-generating distributions.

[1] or perhaps more generally rules which are not certain

No Free Lunch

- Statistical learning theory claims that a machine can generalise well from a finite training set.
- This contradicts basic inductive reasoning which says to derive a rule describing every member of a set one must have information about every member.
- Machine learning avoids this problem by learning probabilistic[1] rules which are *probably* correct about *most* members of the set they concern.
- But, **no free lunch theorem** states that every possible classification machine has the *same error* when averaged over *all possible* data-generating distributions.
  - **No machine learning algorithm is universally better than any other!**

[1] or perhaps more generally rules which are not certain

No Free Lunch

- Statistical learning theory claims that a machine can generalise well from a finite training set.
- This contradicts basic inductive reasoning which says to derive a rule describing every member of a set one must have information about every member.
- Machine learning avoids this problem by learning probabilistic[1] rules which are *probably* correct about *most* members of the set they concern.
- But, **no free lunch theorem** states that every possible classification machine has the *same error* when averaged over *all possible* data-generating distributions.
  - **No machine learning algorithm is universally better than any other!**
  - Fortunately, in the real world, data is generated by a small subset of generating distributions...

[1] or perhaps more generally rules which are not certain

The Universal Approximation Theorem

Let $\psi : \mathbb{R} \rightarrow \mathbb{R}$ be a nonconstant, bounded, and continuous function.

The Universal Approximation Theorem

Let $\psi : \mathbb{R} \rightarrow \mathbb{R}$ be a nonconstant, bounded, and continuous function. Let I_m denote the m-dimensional unit hypercube $[0, 1]^m$.

The Universal Approximation Theorem

Let $\psi : \mathbb{R} \rightarrow \mathbb{R}$ be a nonconstant, bounded, and continuous function. Let I_m denote the m-dimensional unit hypercube $[0, 1]^m$. The space of real-valued continuous functions on I_m is denoted by $C(I_m)$.

The Universal Approximation Theorem

Let $\psi : \mathbb{R} \to \mathbb{R}$ be a nonconstant, bounded, and continuous function. Let I_m denote the m-dimensional unit hypercube $[0, 1]^m$. The space of real-valued continuous functions on I_m is denoted by $C(I_m)$. Then, given any $\varepsilon > 0$ and any function $f \in C(I_m)$, there exist an integer N, real constants $v_i, b_i \in \mathbb{R}$ and real vectors $w_i \in \mathbb{R}^m$ for $i = 1, \ldots, N$, such that we may define:

$F(x) = \sum_{i=1}^{N} v_i \psi(w_i^T x + b_i)$ as an approximate realization of the function f ; that is,

The Universal Approximation Theorem

Let $\psi : \mathbb{R} \to \mathbb{R}$ be a nonconstant, bounded, and continuous function. Let I_m denote the m-dimensional unit hypercube $[0, 1]^m$. The space of real-valued continuous functions on I_m is denoted by $C(I_m)$. Then, given any $\varepsilon > 0$ and any function $f \in C(I_m)$, there exist an integer N, real constants $v_i, b_i \in \mathbb{R}$ and real vectors $w_i \in \mathbb{R}^m$ for $i = 1, \dots, N$, such that we may define:

$F(x) = \sum_{i=1}^{N} v_i \psi(w_i^T x + b_i)$ as an approximate realization of the function f ; that is,

$|F(x) - f(x)| < \varepsilon \, \forall \; \in I_m$.

The Universal Approximation Theorem

Let $\psi : \mathbb{R} \to \mathbb{R}$ be a nonconstant, bounded, and continuous function. Let I_m denote the m-dimensional unit hypercube $[0, 1]^m$. The space of real-valued continuous functions on I_m is denoted by $C(I_m)$. Then, given any $\varepsilon > 0$ and any function $f \in C(I_m)$, there exist an integer N, real constants $v_i, b_i \in \mathbb{R}$ and real vectors $w_i \in \mathbb{R}^m$ for $i = 1, \dots, N$, such that we may define:

$F(x) = \sum_{i=1}^{N} v_i \psi(w_i^T x + b_i)$ as an approximate realization of the function f ; that is,

$|F(x) - f(x)| < \varepsilon \; \forall \; \in I_m.$

$\implies$ simple neural networks can represent a wide variety of interesting functions when given appropriate parameters.

So a single hidden layer network can approximate most functions?

- Yes!

[2] note that it has been shown that the gradients of the function are approximated by the network to an arbitrary precision

So a single hidden layer network can approximate most functions?

- Yes!
- But, ...
  - worse-case analysis shows it might be exponential (possibly one hidden unit for *every* input configuration)

[2] note that it has been shown that the gradients of the function are approximated by the network to an arbitrary precision

So a single hidden layer network can approximate most functions?

- Yes!
- But, ...
  - to get the precision you require (small ε), you might need a really large number of hidden units (very large N).
  - worse-case analysis shows it might be exponential (possibly one hidden unit for *every* input configuration)

[2] note that it has been shown that the gradients of the function are approximated by the network to an arbitrary precision

So a single hidden layer network can approximate most functions?

- Yes!
- But, ...
  - to get the precision you require (small ε), you might need a really large number of hidden units (very large N).
  - worse-case analysis shows it might be exponential (possibly one hidden unit for *every* input configuration)
  - We've not said anything about learnability...

[2] note that it has been shown that the gradients of the function are approximated by the network to an arbitrary precision

So a single hidden layer network can approximate most functions?

- Yes!
- But, ...
  - to get the precision you require (small ε), you might need a really large number of hidden units (very large N).
  - worse-case analysis shows it might be exponential (possibly one hidden unit for *every* input configuration)
  - We've not said anything about learnability...
    - The optimiser might not find a good solution[2].

[2] note that it has been shown that the gradients of the function are approximated by the network to an arbitrary precision

So a single hidden layer network can approximate most functions?

- Yes!
- But, ...
  - to get the precision you require (small ε), you might need a really large number of hidden units (very large N).
  - worse-case analysis shows it might be exponential (possibly one hidden unit for *every* input configuration)
  - We've not said anything about learnability...
    - The optimiser might not find a good solution[2].
    - The training algorithm might just choose the wrong solution as a result of overfitting.

[2] note that it has been shown that the gradients of the function are approximated by the network to an arbitrary precision

So a single hidden layer network can approximate most functions?

- Yes!
- But, ...
  - to get the precision you require (small ε), you might need a really large number of hidden units (very large N).
  - worse-case analysis shows it might be exponential (possibly one hidden unit for *every* input configuration)
  - We've not said anything about learnability...
    - The optimiser might not find a good solution[2].
    - The training algorithm might just choose the wrong solution as a result of overfitting.
    - *There is no known universal proceedure for examining a set of examples and choosing a function that will generalise to points out of the training set.*

[2] note that it has been shown that the gradients of the function are approximated by the network to an arbitrary precision

Then Why Go Deep?

- There are functions you can compute with a deep neural network that shallow networks require exponentially more hidden units to compute.
- The following function is more efficient to implement using a deep neural network: $y = x_1 \oplus x_2 \oplus x_3 \oplus \cdots \oplus x_n$

Vanishing and Exploding Gradients

- The vanishing and exploding gradient problem is a difficulty found in training NN with gradient-based learning methods and backpropagation.

Vanishing and Exploding Gradients

- The vanishing and exploding gradient problem is a difficulty found in training NN with gradient-based learning methods and backpropagation.
- In training, the gradient may become vanishingly small (or large), effectively preventing the weight from changing its value (or exploding in value).

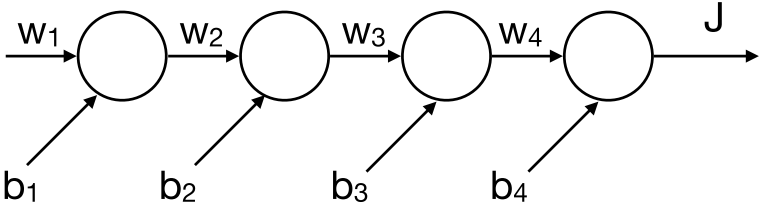
Vanishing and Exploding Gradients

- The vanishing and exploding gradient problem is a difficulty found in training NN with gradient-based learning methods and backpropagation.
- In training, the gradient may become vanishingly small (or large), effectively preventing the weight from changing its value (or exploding in value).
- This leads to the neural network not being able to train.

Vanishing and Exploding Gradients

- The vanishing and exploding gradient problem is a difficulty found in training NN with gradient-based learning methods and backpropagation.
- In training, the gradient may become vanishingly small (or large), effectively preventing the weight from changing its value (or exploding in value).
- This leads to the neural network not being able to train.
- This issue affects many-layered networks (feed-forward), as well as recurrent networks.

Issues with Going Deep



Residual Connections

- One of the most effective ways to resolve the vanishing gradient problem is with residual neural networks (ResNets)[3].

[3] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," CVPR, Las Vegas, NV, 2016, pp. 770-778.

Residual Connections

- One of the most effective ways to resolve the vanishing gradient problem is with residual neural networks (ResNets)[3].
- ResNets are artificial neural networks that use *skip connections* to jump over layers.

[3] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," CVPR, Las Vegas, NV, 2016, pp. 770-778.

Residual Connections

- One of the most effective ways to resolve the vanishing gradient problem is with residual neural networks (ResNets)[3].
- ResNets are artificial neural networks that use *skip connections* to jump over layers.
- The vanishing gradient problem is mitigated in ResNets by reusing activations from a previous layer.

[3] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," CVPR, Las Vegas, NV, 2016, pp. 770-778.

Residual Connections

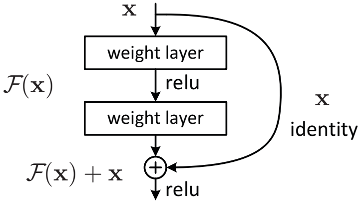


Figure 2. Residual learning: a building block.[4]

[4] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," CVPR, Las Vegas, NV, 2016, pp. 770-778.

Residual Connections

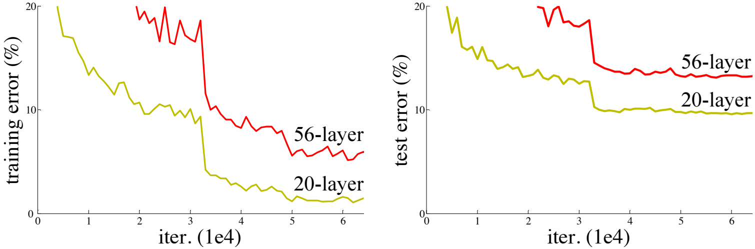


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4. [5]

[5] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," CVPR, Las Vegas, NV, 2016, pp. 770-778.

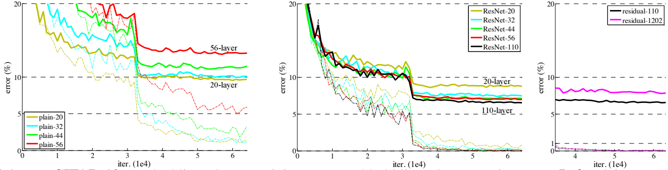


Figure 6. Training on **CIFAR-10**. Dashed lines denote training error, and bold lines denote testing error. **Left**: plain networks. The error of plain-110 is higher than 60% and not displayed. **Middle**: ResNets. **Right**: ResNets with 110 and 1202 layers. [6]

[6] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," CVPR, Las Vegas, NV, 2016, pp. 770-778.

Dropout

- Neural networks with a large number of parameters (and hidden layers) are powerful, however, overfitting is a serious problem in such systems.

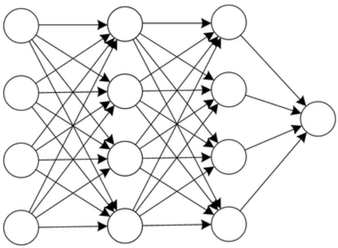
Dropout

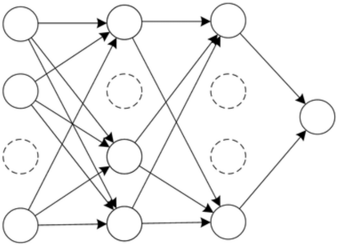
- Neural networks with a large number of parameters (and hidden layers) are powerful, however, overfitting is a serious problem in such systems.
- Dropout is a form of regularization

Dropout

- Neural networks with a large number of parameters (and hidden layers) are powerful, however, overfitting is a serious problem in such systems.
- Dropout is a form of regularization
- The key idea in dropout is to randomly drop neurons, including all of the connections, from the neural network during training.

Dropout


(a) Standard Neural Network


(b) Network after Dropout [7]

[7] Image from: https://www.researchgate.net/figure/Dropout-neural-network-model-a-is-a-standard-neural-network-b-is-the-same-fig3_309206911

How Does Dropout Work in Practice?

- In the learning phase, we stochastically remove hidden units by setting a dropout probability for each layer in the network. We then randomly decide wether or not a neuron in a given layer is removed stochastically.

How Does Dropout Work in Practice?

- We define a random binary mask $m^{(l)}$ which is used to remove neurons, and note, $m^{(l)}$ changes for each iteration of the backpropagation algorithm.

How Does Dropout Work in Practice?

- We define a random binary mask $m^{(l)}$ which is used to remove neurons, and note, $m^{(l)}$ changes for each iteration of the backpropagation algorithm.
- For layers, $l = 1$ to $L - 1$, for the forward pass of backpropagation, we then compute

$$a^{(l)} = \sigma(w^{(l)} a^{(l-1)} + b^{(l)}) \odot m^{(l)} \tag{1}$$

How Does Dropout Work in Practice?

- We define a random binary mask $m^{(l)}$ which is used to remove neurons, and note, $m^{(l)}$ changes for each iteration of the backpropagation algorithm.
- For layers, $l = 1$ to $L - 1$, for the forward pass of backpropagation, we then compute

$$a^{(l)} = \sigma(w^{(l)} a^{(l-1)} + b^{(l)}) \odot m^{(l)} \tag{1}$$

- For layer L,

$$a^{(L)} = \sigma(w^{(L)} a^{(L-1)} + b^{(l)}) \tag{2}$$

How Does Dropout Work in Practice?

- We define a random binary mask $m^{(l)}$ which is used to remove neurons, and note, $m^{(l)}$ changes for each iteration of the backpropagation algorithm.
- For layers, $l = 1$ to $L - 1$, for the forward pass of backpropagation, we then compute

$$a^{(l)} = \sigma(w^{(l)}a^{(l-1)} + b^{(l)}) \odot m^{(l)} \tag{1}$$

- For layer L,

$$a^{(L)} = \sigma(w^{(L)}a^{(L-1)} + b^{(l)}) \tag{2}$$

- For the backward pass of the backpropagation algorithm,

$$\delta^L = \Delta_a J \odot \sigma'(z^L) \tag{3}$$

How Does Dropout Work in Practice?

- We define a random binary mask $m^{(l)}$ which is used to remove neurons, and note, $m^{(l)}$ changes for each iteration of the backpropagation algorithm.
- For layers, $l = 1$ to $L - 1$, for the forward pass of backpropagation, we then compute

$$a^{(l)} = \sigma(w^{(l)} a^{(l-1)} + b^{(l)}) \odot m^{(l)} \tag{1}$$

- For layer L,

$$a^{(L)} = \sigma(w^{(L)} a^{(L-1)} + b^{(l)}) \tag{2}$$

- For the backward pass of the backpropagation algorithm,

$$\delta^L = \Delta_a J \odot \sigma'(z^L) \tag{3}$$

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l) \odot m^{(l)} \tag{4}$$

Why Does Dropout Work?

- Neurons cannot co-adapt to other units (they cannot assume that all of the other units will be present)

Why Does Dropout Work?

- Neurons cannot co-adapt to other units (they cannot assume that all of the other units will be present)
- By breaking co-adaptation, each unit will ultimately find more general features