Deep learning: An Introduction for Applied Mathematicians

Mark Blyth

Background

Definition (Neural network)

A nonlinear model that's general enough to fit most data

Background

Definition (Neural network)

A nonlinear model that's general enough to fit most data

- Take input vectors $\mathbf{x}_i$

Background

Definition (Neural network)

A nonlinear model that's general enough to fit most data

- Take input vectors $\mathbf{x}_i$
- Take output vectors $\mathbf{y}_i$

Background

Definition (Neural network)

A nonlinear model that's general enough to fit most data

- Take input vectors $\mathbf{x}_i$
- Take output vectors $\mathbf{y}_i$
- Fit some nonlinear model $\mathbf{y} = f(\mathbf{x})$

Drilling site example

- Input data: $\mathbf{x}_i = (u_i, v_i)$, oil well location
- Output data: $y_i \in \{0, 1\}$, success or failure
- Learn some nonlinear model mapping locations to drill successes

Model form

How do we create a nice general model?

- A neural network is just a convenient representation of ‘stringed nonlinearities’

Model form

How do we create a nice general model?

- A neural network is just a convenient representation of ‘stringed nonlinearities’
- Take a simple nonlinearity, eg. sigmoid $\sigma(x)$

Model form

How do we create a nice general model?

- A neural network is just a convenient representation of ‘stringed nonlinearities’
- Take a simple nonlinearity, eg. sigmoid $\sigma(x)$
- String a load of them together

Model form

How do we create a nice general model?

- A neural network is just a convenient representation of ‘stringed nonlinearities’
- Take a simple nonlinearity, eg. sigmoid $\sigma(x)$
- String a load of them together
  - $f(x) = \sigma(\sigma(\ldots\sigma(x)\ldots))$

Model form

How do we create a nice general model?

- A neural network is just a convenient representation of 'stringed nonlinearities'
- Take a simple nonlinearity, eg. sigmoid $\sigma(x)$
- String a load of them together
  - $f(x) = \sigma(\sigma(\ldots\sigma(x)\ldots))$
- More useful: can shift mean, scale with a linear transform $\sigma(wx + b)$

Model form

How do we create a nice general model?

- A neural network is just a convenient representation of 'stringed nonlinearities'
- Take a simple nonlinearity, eg. sigmoid $\sigma(x)$
- String a load of them together
  - $f(x) = \sigma(\sigma(\ldots\sigma(x)\ldots))$
- More useful: can shift mean, scale with a linear transform $\sigma(wx+b)$
  - $f(x) = \sigma(b + w\sigma(b + w\sigma(\ldots\sigma(b + w\sigma(x))\ldots))$

Model form

How do we create a nice general model?

- A neural network is just a convenient representation of ‘stringed nonlinearities’
- Take a simple nonlinearity, eg. sigmoid $\sigma(x)$
- String a load of them together
  - $f(x) = \sigma(\sigma(\dots\sigma(x)\dots))$
- More useful: can shift mean, scale with a linear transform $\sigma(wx+b)$
  - $f(x) = \sigma(b + w\sigma(b + w\sigma(\dots\sigma(b + w\sigma(x))\dots))$
- Fit the model

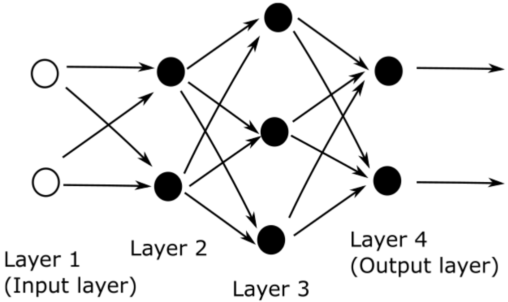
Model form

How do we create a nice general model?

- A neural network is just a convenient representation of 'stringed nonlinearities'
- Take a simple nonlinearity, eg. sigmoid $\sigma(x)$
- String a load of them together
  - $f(x) = \sigma(\sigma(\dots\sigma(x)\dots))$
- More useful: can shift mean, scale with a linear transform $\sigma(wx + b)$
  - $f(x) = \sigma(b + w\sigma(b + w\sigma(\dots\sigma(b + w\sigma(x))\dots))$
- Fit the model
  - Find appropriate values for each w, b

Why ‘neural network’?

- We can represent the equation nicely as a graph



- Information flows forward through the graph much like it does through the brain

Model form

- Node inputs = weighted sum of previous layer's outputs

Model form

- Node inputs = weighted sum of previous layer's outputs
  - Vector form: $\mathbf{i}_k = W_k \mathbf{o}_{k-1} + \mathbf{b}_k$

Model form

- Node inputs = weighted sum of previous layer's outputs
  - Vector form: $\mathbf{i}_k = W_k \mathbf{o}_{k-1} + \mathbf{b}_k$
- Node output = sigmoid of inputs

Model form

- Node inputs = weighted sum of previous layer's outputs
  - Vector form: $\mathbf{i}_k = W_k \mathbf{o}_{k-1} + \mathbf{b}_k$
- Node output = sigmoid of inputs
  - $\mathbf{o}_k = \sigma(\mathbf{i}_k)$

Model form

- Node inputs = weighted sum of previous layer's outputs
  - Vector form: $\mathbf{i}_k = W_k \mathbf{o}_{k-1} + \mathbf{b}_k$
- Node output = sigmoid of inputs
  - $\mathbf{o}_k = \sigma(\mathbf{i}_k)$
- Inputs can be efficiently computed with some linear algebra

Model form

- Node inputs = weighted sum of previous layer's outputs
  - Vector form: $\mathbf{i}_k = W_k \mathbf{o}_{k-1} + \mathbf{b}_k$
- Node output = sigmoid of inputs
  - $\mathbf{o}_k = \sigma(\mathbf{i}_k)$
- Inputs can be efficiently computed with some linear algebra
- Much neater representation than 'stringed nonlinearities'

Why does this work?

Loosely speaking…

- Having lots of W_k, b_k gives us a very general model

Why does this work?

Loosely speaking...

- Having lots of W_k, b_k gives us a very general model
- We can fit the data by selecting W, b to minimise the error $\sum \|f(\mathbf{x}_i - \mathbf{y}_i\|^2$

Why does this work?

Loosely speaking…

- Having lots of W_k, b_k gives us a very general model
- We can fit the data by selecting W, b to minimise the error $\sum \|f(\mathbf{x}_i - \mathbf{y}_i\|^2$
- Universal approximator!

Network training

We now have a model; how do we fit it?

Definition (Gradient descent)

Optimisation method that iteratively updates parameters with a most-improving step

Network training

We now have a model; how do we fit it?

Definition (Gradient descent)

Optimisation method that iteratively updates parameters with a most-improving step

- Like a massless ball rolling down a hill

Network training

We now have a model; how do we fit it?

Definition (Gradient descent)

Optimisation method that iteratively updates parameters with a most-improving step

- Like a massless ball rolling down a hill
  - Travels in the direction of greatest slope

Network training

We now have a model; how do we fit it?

Definition (Gradient descent)

Optimisation method that iteratively updates parameters with a most-improving step

- Like a massless ball rolling down a hill
  - Travels in the direction of greatest slope
  - Reaches a flat bit eventually

Network training

We now have a model; how do we fit it?

Definition (Gradient descent)

Optimisation method that iteratively updates parameters with a most-improving step

- Like a massless ball rolling down a hill
  - Travels in the direction of greatest slope
  - Reaches a flat bit eventually
- Flat bit means cost stops changing

Network training

We now have a model; how do we fit it?

Definition (Gradient descent)

Optimisation method that iteratively updates parameters with a most-improving step

- Like a massless ball rolling down a hill
  - Travels in the direction of greatest slope
  - Reaches a flat bit eventually
- Flat bit means cost stops changing
  - Could be a good fit, could be a saddle or local minimum

Stochastic gradient descent

- Let $\mathrm{cost}_p = \sum_i \|f_p(\mathbf{x}_i) - \mathbf{y}_i\|^2$

Stochastic gradient descent

- Let $\mathrm{cost}_p = \sum_i \|f_p(\mathbf{x}_i) - \mathbf{y}_i\|^2$
- Iteratively let $p_{i+1} = p_i - \eta \frac{\partial \mathrm{cost}}{\partial p_i}$

Stochastic gradient descent

- Let $\text{cost}_p = \sum_i \|f_p(\mathbf{x}_i) - \mathbf{y}_i\|^2$
- Iteratively let $p_{i+1} = p_i - \eta \frac{\partial \text{cost}}{\partial p_i}$
  - Alternatively, calculate the cost over some 'minibatches' and perform iterations on these

Stochastic gradient descent

- Let $\mathrm{cost}_p = \sum_i \|f_p(\mathbf{x}_i) - \mathbf{y}_i\|^2$
- Iteratively let $p_{i+1} = p_i - \eta \frac{\partial \mathrm{cost}}{\partial p_i}$
  - Alternatively, calculate the cost over some ‘minibatches’ and perform iterations on these
- How do we find $\frac{\partial \mathrm{cost}}{\partial p_i}$?

Backprop

Definition (Backpropagation)

A method for finding cost-function gradient, given

- a cost function
- a nonlinearity
- a network topology

Backprop is the core of NN training!

How does backprop work?

For a single input…

- How does cost function change with last layer's outputs?

How does backprop work?

For a single input…

- How does cost function change with last layer's outputs?
  - $\frac{\partial \text{cost}}{\partial \text{output}} = 2\|f(\mathbf{x}_i) - \mathbf{y}_i\|$

How does backprop work?

For a single input...

- How does cost function change with last layer's outputs?
  - $\frac{\partial \text{cost}}{\partial \text{output}} = 2\|f(\mathbf{x}_i) - \mathbf{y}_i\|$
- How does ith layer output change with ith layer input?

How does backprop work?

For a single input…

- How does cost function change with last layer's outputs?
  - $\frac{\partial \text{cost}}{\partial \text{output}} = 2\|f(\mathbf{x}_i) - \mathbf{y}_i\|$
- How does ith layer output change with ith layer input?
  - $\frac{\partial \text{output}}{\partial \text{input}} = \sigma'(\text{input})$

How does backprop work?

For a single input…

- How does cost function change with last layer's outputs?
  - $\frac{\partial \text{cost}}{\partial \text{output}} = 2\|f(\mathbf{x}_i) - \mathbf{y}_i\|$
- How does ith layer output change with ith layer input?
  - $\frac{\partial \text{output}}{\partial \text{input}} = \sigma'(\text{input})$
- How does ith layer input change with ith layer weights, biases?

How does backprop work?

For a single input…

- How does cost function change with last layer's outputs?
  - $\frac{\partial \text{cost}}{\partial \text{output}} = 2\|f(\mathbf{x}_i) - \mathbf{y}_i\|$
- How does ith layer output change with ith layer input?
  - $\frac{\partial \text{output}}{\partial \text{input}} = \sigma'(\text{input})$
- How does ith layer input change with ith layer weights, biases?
  - $\frac{\partial \text{input}}{\partial \text{weights}} = (i-1)'\text{th layer output}$

How does backprop work?

For a single input...

- How does cost function change with last layer's outputs?
  - $\frac{\partial \text{cost}}{\partial \text{output}} = 2\|f(\mathbf{x}_i) - \mathbf{y}_i\|$
- How does ith layer output change with ith layer input?
  - $\frac{\partial \text{output}}{\partial \text{input}} = \sigma'(\text{input})$
- How does ith layer input change with ith layer weights, biases?
  - $\frac{\partial \text{input}}{\partial \text{weights}} = (i-1)'\text{th layer output}$
  - $\frac{\partial \text{input}}{\partial \text{biases}} = 1$

How does backprop work?

For a single input...

- How does cost function change with last layer's outputs?
  - $\frac{\partial \text{cost}}{\partial \text{output}} = 2\|f(\mathbf{x}_i) - \mathbf{y}_i\|$
- How does ith layer output change with ith layer input?
  - $\frac{\partial \text{output}}{\partial \text{input}} = \sigma'(\text{input})$
- How does ith layer input change with ith layer weights, biases?
  - $\frac{\partial \text{input}}{\partial \text{weights}} = (i-1)'\text{th layer output}$
  - $\frac{\partial \text{input}}{\partial \text{biases}} = 1$
- Can find cost function gradient by chain-ruling these all together

How does backprop work?

For a single input…

- How does cost function change with last layer's outputs?
  - $\frac{\partial \text{cost}}{\partial \text{output}} = 2\|f(\mathbf{x}_i) - \mathbf{y}_i\|$
- How does ith layer output change with ith layer input?
  - $\frac{\partial \text{output}}{\partial \text{input}} = \sigma'(\text{input})$
- How does ith layer input change with ith layer weights, biases?
  - $\frac{\partial \text{input}}{\partial \text{weights}} = (i-1)'\text{th layer output}$
  - $\frac{\partial \text{input}}{\partial \text{biases}} = 1$
- Can find cost function gradient by chain-ruling these all together
- Can sum the resulting gradient across the full minibatch

Backprop results

Backprop gives us an easy way to compute $\frac{\partial \text{cost}}{\partial \text{weights}}$ and $\frac{\partial \text{cost}}{\partial \text{biases}}$

- Forward pass: find each node's inputs and outputs

Backprop results

Backprop gives us an easy way to compute $\frac{\partial \text{cost}}{\partial \text{weights}}$ and $\frac{\partial \text{cost}}{\partial \text{biases}}$

- Forward pass: find each node's inputs and outputs
- Backward pass:

Backprop results

Backprop gives us an easy way to compute $\frac{\partial \text{cost}}{\partial \text{weights}}$ and $\frac{\partial \text{cost}}{\partial \text{biases}}$

- Forward pass: find each node's inputs and outputs
- Backward pass:
  - Relate last layer's output to cost function gradient

Backprop results

Backprop gives us an easy way to compute $\frac{\partial \text{cost}}{\partial \text{weights}}$ and $\frac{\partial \text{cost}}{\partial \text{biases}}$

- Forward pass: find each node's inputs and outputs
- Backward pass:
  - Relate last layer's output to cost function gradient
  - Relate each previous layer's outputs, weights, biases to next layer's error

Backprop results

Backprop gives us an easy way to compute $\frac{\partial \text{cost}}{\partial \text{weights}}$ and $\frac{\partial \text{cost}}{\partial \text{biases}}$

- Forward pass: find each node's inputs and outputs
- Backward pass:
  - Relate last layer's output to cost function gradient
  - Relate each previous layer's outputs, weights, biases to next layer's error
  - Relate next layer's error to cost function gradient

Backprop results

Backprop gives us an easy way to compute $\frac{\partial \text{cost}}{\partial \text{weights}}$ and $\frac{\partial \text{cost}}{\partial \text{biases}}$

- Forward pass: find each node's inputs and outputs
- Backward pass:
  - Relate last layer's output to cost function gradient
  - Relate each previous layer's outputs, weights, biases to next layer's error
  - Relate next layer's error to cost function gradient
- Propagates errors backward through the network

Convolutional neural networks

- Visual cortex has a ‘receptive field’
- CNN mirror this with local kernel transforms
- Convolutional layers automatically extract features
- Allows NNs to efficiently manipulate high-dimensional data

Practical aspects

Definition (Overfitting)

Representing the training data too closely, and losing the ability to generalise

Costs and activations

- We don’t have to use sigmoids

Costs and activations

- We don't have to use sigmoids
  - ReLU: linear activation with positive support

Costs and activations

- We don’t have to use sigmoids
  - ReLU: linear activation with positive support
  - Alternative: small gradient for negative numbers, large gradient for positive numbers

Costs and activations

- We don't have to use sigmoids
  - ReLU: linear activation with positive support
  - Alternative: small gradient for negative numbers, large gradient for positive numbers
- We don't have to use residuals

Costs and activations

- We don't have to use sigmoids
  - ReLU: linear activation with positive support
  - Alternative: small gradient for negative numbers, large gradient for positive numbers
- We don't have to use residuals
  - Softmax-log-loss

The essence of ML

- Machine learning sounds flashy and cool; it's just big statistics
- Large-scale model definitions and cost-function-fitting

Section 8 discussion points

- Why use NNs?
  - When do other methods generalise better?

Section 8 discussion points

- How robust are the results?
  - Do small input changes matter much? Should they?

Section 8 discussion points

- What's a sensible nonlinearity?
  - Is there any reason to choose ReLU over sigmoids?

Section 8 discussion points

- What topology do we need?
  - How many hidden layers? How big?

Section 8 discussion points

- Can we regularise?
  - Reduce overfitting by penalising model complexity

Section 8 discussion points

- Explainability
  - *Why* should NNs give good results? Why do they give the results they do?

Discussion

- Why use NNs vs. another method?
- What topology do we need?
- What's a sensible nonlinearity?
- How robust are the results, and how much do we care?
- Can we regularise?
- Explainability – how much do we trust the results?
- How much can we actually learn from a black box?
- How much data is enough data?
- What are the applications to nonlinear dynamics?

Next paper

Someone to lead?

Suggestion: Heinonen, Markus, et al. "Learning unknown ODE models with Gaussian processes." arXiv preprint arXiv:1803.04303 (2018).