Introduction to Neural Networks

Mark Johnson

Dept of Computing
Macquarie University
Sydney
Australia

September 2015

MACQUARIE University

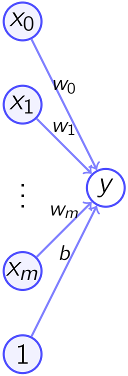
Outline

What are (deep) neural networks?

- *Neural networks* are functions mapping *inputs* to *outputs*
  - the inputs are usually *feature vectors*, encoding pictures, documents, etc.
  - the outputs are usually a single value (e.g., a label for the input), but can be structured (e.g., a sentence)
- A *neural network* is a network of models typically organised into *a sequence of layers*
  - the first layer is the input feature vector
  - the output of one layer serves as the input to the next layer
  - the output is the last layer
- *Deep neural networks* have a relatively large number of layers (e.g., 5 or more)

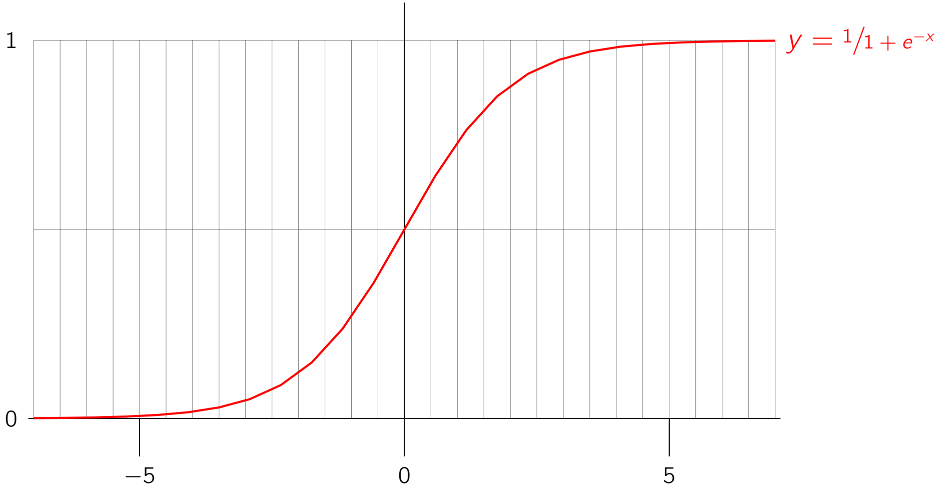
Logistic regression as a 2-layer neural network

$$
\begin{aligned}
\mathrm{P}(Y{=}1 \mid \mathbf{x}) &= \sigma\Big(\sum_j w_j x_j + b\Big) \\
&= \sigma(\mathbf{w} \cdot \mathbf{x} + b) \\
\sigma(v) &= \frac{1}{1 + \exp(-v)}
\end{aligned}
$$



- $\mathbf{x} = (x_1, \ldots, x_m)$ is the *input vector* (feature values)
- $\mathbf{w} = (w_1, \ldots, w_m)$ is a vector of *connection weights* (feature weights)
- b is a *bias weight*
  - can be viewed as the weight associated with an "always on" input

The logistic sigmoid function



Weaknesses of logistic regression

- Logistic regression is a *log-linear model*
  i.e., the *log odds* are a *linear function of the input*

$$\mathrm{P}(Y{=}1 \mid \mathbf{x}) = \frac{1}{1+\exp(-\mathbf{w}\cdot\mathbf{x})} \quad \Leftrightarrow \quad \log\left(\frac{\mathrm{P}(Y{=}1 \mid \mathbf{x})}{\mathrm{P}(Y{=}0 \mid \mathbf{x})}\right) = \mathbf{w}\cdot\mathbf{x}$$

- The presence of feature x_j increases the log odds by w_j

$\Rightarrow$ Unable to capture *non-linear interactions* between the features
  - **XOR** is the classic example of a function that a linear model cannot express
  - can a linear function of pixel intensities recognise a cat?
- Neural networks with hidden layers can learn non-linear feature interactions
- Note: a linear model can capture non-linear interactions if the inputs are *non-linearly transformed* first
  - **XOR** = **OR** − **AND**
  - *Extreme learning* consists of linear regression applied to *random non-linear transformations* of the input

MACQUARIE University

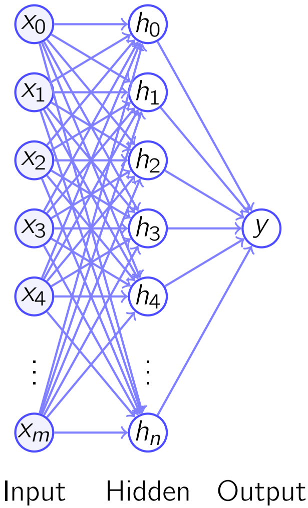
Hidden nodes

- *Hidden nodes* are nodes that are neither input nor output nodes
- Each hidden layer provides extra expressive power
- Three layer neural network (ignoring bias nodes):

$$
\begin{aligned}
h_i &= \sigma(\sum_j A_{ij} x_j) \\
y &= \sigma(\sum_j b_j h_j)
\end{aligned}
$$

or in matrix terms:

$$
\begin{aligned}
\mathbf{h} &= \sigma(\mathbf{A}\,\mathbf{x}) \\
y &= \sigma(\mathbf{b} \cdot \mathbf{h})
\end{aligned}
$$



One-hot encoding for categorical inputs

- A *categorical variable* ranges over a fixed set of categories (e.g., words in a fixed vocabulary)
- A *one-hot encoding* of a categorical variable associates a binary node for each possible value of the categorical variable
  - the one-hot encoding of $c \in 1, \ldots, m$ is $\mathbf{x} = (x_1, \ldots, x_m)$, where $x_c = 1$ and $x_{c'} = 0$ for $c' \neq c$

- A sequence of categorical variables can be represented by a sequence of their 1-hot encodings
- If $\mathbf{x}_c$ is the one-hot encoding of category c, then $\mathbf{A}\mathbf{x}_c = \mathbf{A}_{\cdot,c}$ so a 1-hot encoding of c corresponds to a table-lookup of column c

Why is the non-linearity important?

- General form of a neural network:

$$\mathbf{x}^{(i)} = g^{(i)}(\mathbf{W}^{(i)}\mathbf{x}^{(i-1)})$$

  where:
  - $\mathbf{x}^{(i)}$ are the activations at level i,
  - $\mathbf{W}^{(i)}$ is a weight matrix mapping activations at level $i-1$ to level i
  - $g^{(i)}$ is a *non-linear function* (e.g., the sigmoid function)
- E.g., a general 3-layer network has the following form:

$$y = g^{(2)}(\mathbf{W}^{(2)}\mathbf{x}^{(1)}) = g^{(2)}(\mathbf{W}^{(2)}g^{(1)}(\mathbf{W}^{(1)}\mathbf{x}^{(0)}))$$

- If $g^{(1)} = g^{(2)} = I$ (the identity function), then:

$$y = \mathbf{W}^{(2)}\mathbf{W}^{(1)}\mathbf{x}^{(0)}$$

- But the product of two matrices is another matrix!

$\Rightarrow$ *A three-layer network is equivalent to a two-layer network if $g^{(1)}$ is a linear function*

MACQUARIE University

Outline

MACQUARIE University

Learning weights via Stochastic Gradient Descent

- Given labelled training data $D = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n))$, we want to find weights $\widehat{\mathbf{W}}$ with *minimum loss*:

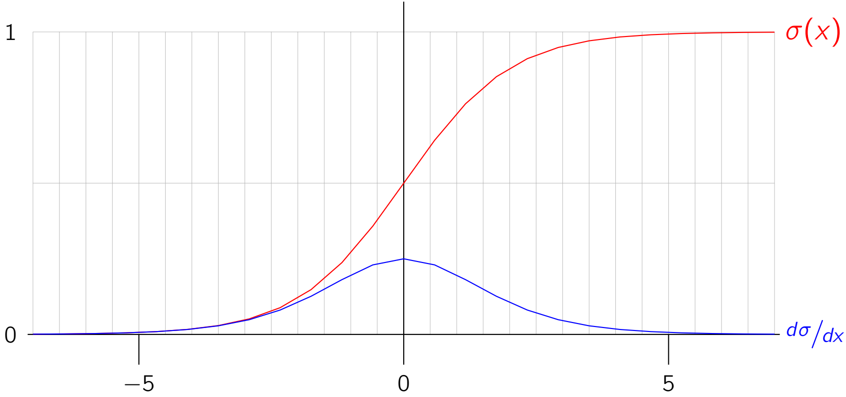
$$\begin{aligned} \widehat{\mathbf{W}} &= \underset{\mathbf{W}}{\operatorname{argmin}} \sum_{i=1}^{n} \ell(\mathbf{W}; \mathbf{x}_i, y_i), \text{ where:} \\ \ell(\mathbf{W}; \mathbf{x}, y) &= -\log \mathrm{P}(y \mid \mathbf{x}) \end{aligned}$$

- $\ell(\mathbf{W}; \mathbf{x}_i, y_i)$ is *loss* incurred when predicting y from $\mathbf{x}$ using weights $\mathbf{W}$
  - probabilities are not greater than 1 $\Rightarrow$ loss is never negative
  - $\log(1)=0 \Rightarrow$ perfect prediction incurrs zero loss
- Minimising log-loss is *maximum likelihood estimation*
- *Stochastic gradient descent:*
  - choose a random training example $(\mathbf{x}, y) \in D$
  - update $\mathbf{W}$ by adding $-\varepsilon \frac{\partial \ell(\mathbf{W}; \mathbf{x}, y)}{\partial \mathbf{W}}$

Derivatives of the sigmoid function

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

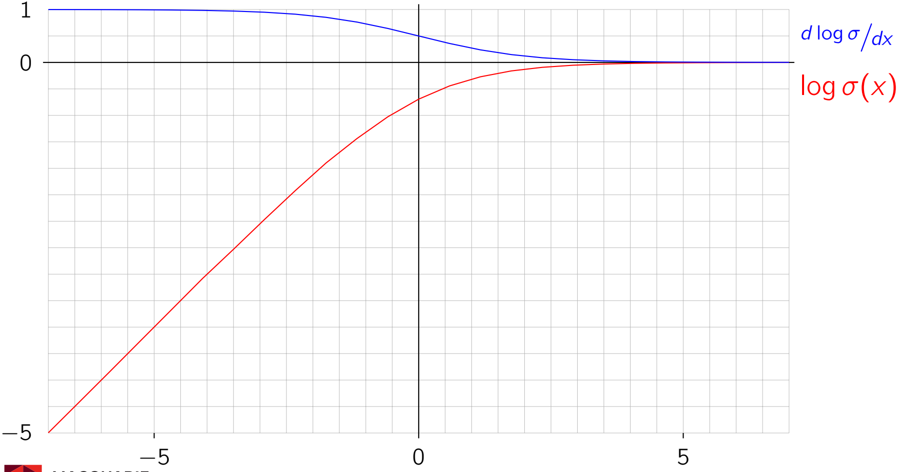
$$\frac{d\sigma(x)}{dx} = \frac{1}{2+e^{-x}+e^{x}} = \sigma(x)\sigma(-x)$$



MACQUARIE University

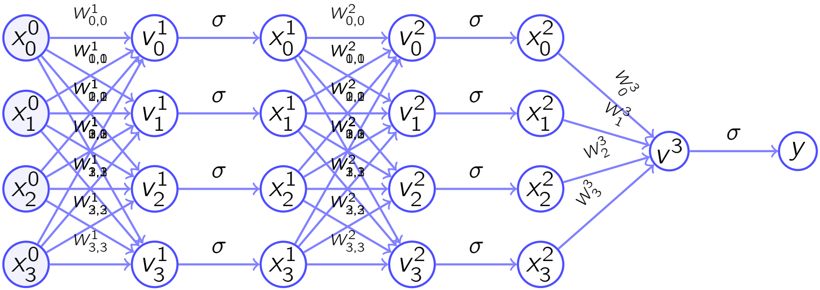
Derivatives of the log sigmoid function

$$\frac{d \, \log \sigma(x)}{dx} = \frac{1 + e^{-x}}{2 + e^{-x} + e^{x}} = \sigma(-x) = 1 - \sigma(x)$$



MACQUARIE University

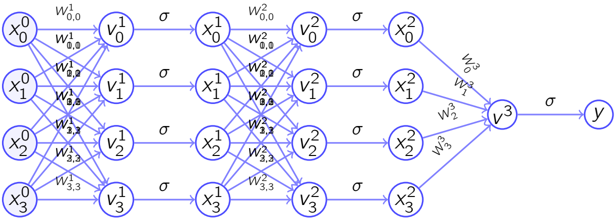
Calculating derivatives with Backpropagation (1)



$$
\begin{aligned}
y &= \sigma(v^{(3)}) \\
\mathbf{v}^{(k)} &= \mathbf{W}^{(k)}\,\mathbf{x}^{(k-1)}, k = 1, \ldots, 3 \\
\mathbf{x}^{(k)} &= \sigma(\mathbf{v}^{(k)}), k = 1, 2
\end{aligned}
$$

MACQUARIE University

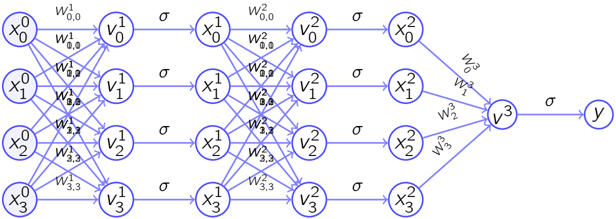
Calculating derivatives with Backpropagation (2)



$$
\begin{aligned}
y &= \sigma(v^{(3)}) \\
v^{(3)} &= \mathbf{W}^{(3)}\,\mathbf{x}^{(2)} \\
\frac{\partial \log y}{\partial \mathbf{W}^{(3)}} &= \frac{\partial \log y}{\partial v^{(3)}}\,\frac{\partial v^{(3)}}{\partial \mathbf{W}^{(3)}} = (1-\sigma(v^{(3)}))\,\mathbf{x}^{(2)} \\
\frac{\partial \log y}{\partial \mathbf{x}^{(2)}} &= \frac{\partial y}{\partial v^{(3)}}\,\frac{\partial v^{(3)}}{\partial \mathbf{x}^{(2)}} = (1-\sigma(v^{(3)}))\,\mathbf{W}^{(3)} \\
\frac{\partial \log y}{\partial \mathbf{v}^{(2)}} &= \frac{\partial \log y}{\partial \mathbf{x}^{(2)}}\,\frac{\partial \mathbf{x}^{(2)}}{\partial \mathbf{v}^{(2)}} = (1-\sigma(v^{(3)}))\,\mathbf{W}^{(3)}\sigma(\mathbf{v}^{(2)})\,\sigma(-\mathbf{v}^{(2)})
\end{aligned}
$$

MACQUARIE University

Calculating derivatives with Backpropagation (3)



$$\mathbf{x}^{(2)} = \sigma(\mathbf{v}^{(2)})$$

$$\mathbf{v}^{(2)} = \mathbf{W}^{(2)}\mathbf{x}^{(1)}$$

$$\frac{\partial \log y}{\partial \mathbf{v}^{(2)}} = \frac{\partial \log y}{\partial \mathbf{x}^{(2)}} \frac{\partial \mathbf{x}^{(2)}}{\partial \mathbf{v}^{(2)}} = \frac{\partial \log y}{\partial \mathbf{x}^{(2)}} \left(\sigma(\mathbf{v}^{(2)})\, \sigma(-\mathbf{v}^{(2)})\right)$$

$$\frac{\partial \log y}{\partial \mathbf{W}^{(2)}} = \frac{\partial \log y}{\partial \mathbf{v}^{(2)}} \frac{\partial \mathbf{v}^{(2)}}{\partial \mathbf{W}^{(2)}} = \frac{\partial \log y}{\partial \mathbf{v}^{(2)}}\, \mathbf{x}^{(1)}$$

$$\frac{\partial \log y}{\partial \mathbf{x}^{(1)}} = \frac{\partial \log y}{\partial \mathbf{v}^{(2)}} \frac{\partial \mathbf{v}^{(2)}}{\partial \mathbf{x}^{(1)}} = \frac{\partial \log y}{\partial \mathbf{v}^{(2)}}\, \mathbf{W}^{(2)}$$

MACQUARIE University

Calculating derivatives with Backpropagation (4)

- Backpropagation is essentially just repeated application of the "chain rule" for differentiation
- *High-level overview of backpropagation:*
  - *Forward pass:* propagate activations from input layer to output layer
  - *Backward pass:* propagate error signal from output layer back to input layer
  - *Compute derivatives:* derivatives involve both "forward" and "backward" activations

MACQUARIE University

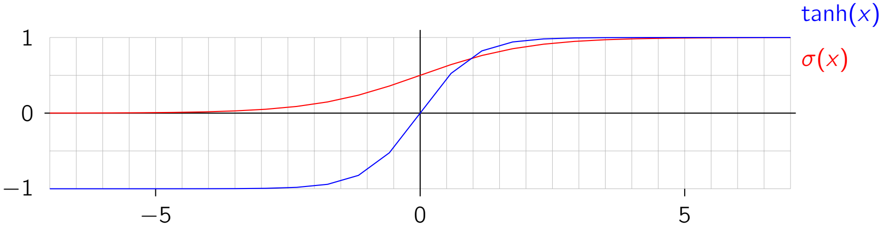
Outline

MACQUARIE University

tanh activation function (a scaled sigmoid)

- No reason for hidden unit activations to be probabilities
  - output unit is often a predicted value, e.g., a (log) probability
- Idea: allow negative as well as positive hidden unit activations

$$\begin{aligned}\tanh(v) &= \frac{\exp(v) - \exp(-v)}{\exp(v) + \exp(-v)} = 2\sigma(2v) - 1 \\ \sigma(v) &= \frac{1}{1 + \exp(-v)}\end{aligned}$$

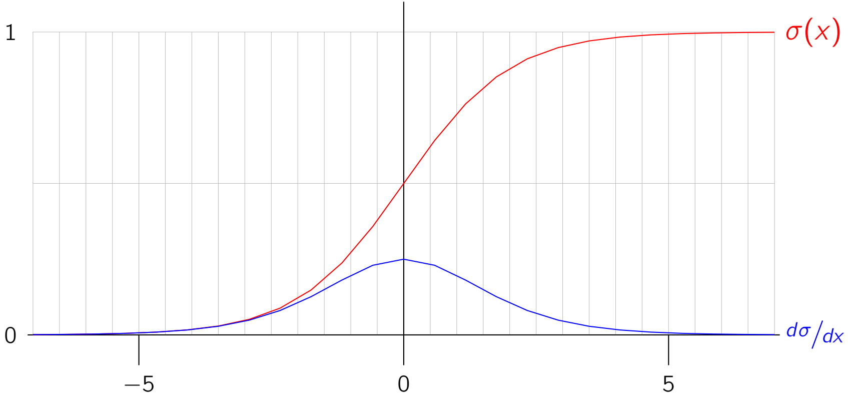


The vanishing gradient problem

- The sigmoid and tanh functions *saturate* on very large or very small inputs

$\Rightarrow$ *Vanishing gradient problem:* in deep networks, gradient is often close to zero for weights close to input layer

$\Rightarrow$ That part of the network learns very slowly, or not at all

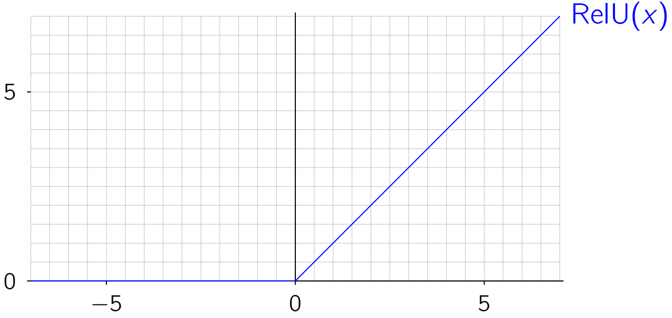


Rectlinear activation function

- The *Rectlinear* (rectified linear) activation function doesn't saturate: (in positive direction)

$$\mathrm{RelU}(v) \quad = \quad \max(v, 0)$$

- It's also very fast to calculate!



Outline

MACQUARIE University

Early stopping in stochastic gradient descent

- Neural networks (like all machine learning algorithms) *can over-fit the training data*
- You can detect over-fitting by measuring the difference in accuracy on training and *held-out development data* (don't use your test data!)
- *Early stopping:* stop training when accuracy on held-out development data starts decreasing
  - often over-fitting will have started before held-out accuracy starts decreasing

Regularisation via weight decay

- *Regularisation:* add a term to loss function that penalises large connection weights, such as the *L2 regulariser*

$$
\begin{aligned}
Q(\mathbf{w}) &= {}^1\!/_2 \sum_j w_j^2 \\
\frac{\partial Q}{\partial w_j} &= w_j
\end{aligned}
$$

- In SGD, the L2 penalty subtracts a fraction of each weight at each iteration
  - also known as a *shrinkage method*, as it "shrinks" the connection weights

MACQUARIE University

Model-averaging via Drop-out

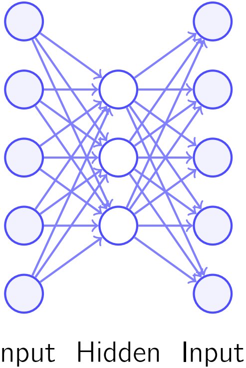
- *Drop-out* training:
  - at *training time*, during each SGD iteration *randomly select* half the connection weights, and ignore the others
    - i.e., calculate forward activations, derivatives and updates just using selected weights
  - at *test time*, use full network but divide all weights by 2
- This is *training a randomly-chosen subnetwork* at each iteration
  - it forces the network not to rely on any single connection
  - each time a data item is seen during training it has a different subnetwork $\Rightarrow$ encourages network to learn multiple generalisations about each data item
- Full network is an "average" of randomly chosen subnetworks
- Over-fitting is much less of a problem with drop-out

Outline

Auto-encoders learn low-dimensional representations

- An *auto-encoder* is a neural network that *predicts its own input* (i.e., learns an identity function)
- Typically auto-encoders have *fewer hidden units than visible input/output units*
  - can also impose a *sparsity penalty* (e.g., cross-entropy) on hidden units

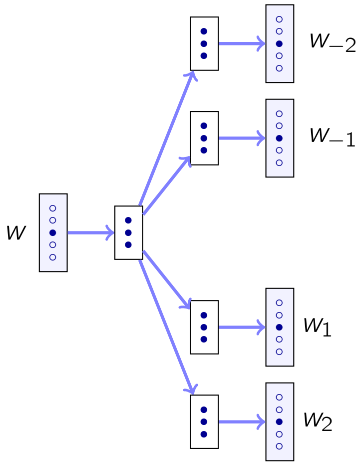
⇒ Auto-encoder has to learn generalisations about its inputs
  - hidden units represent these generalisations



MACQUARIE University

word2vec learns low-dimensional word representations

- word2vec learns 500-dimensional word representations by *predicting the neighbouring words* surrounding each word w in a 5-word window $(w_{-2}, w_{-1}, w, w_1, w_2)$
- Has surprising properties, e.g., [Paris] – [France] ≈ [Rome] – [Italy]
- Weight matrices mapping representation of word w to representations of neighbouring words are *shared* (i.e., position-independent)
  - would position-dependent weights be better?



Outline

Summary

- A neural network is a large combination of classifiers, arranged in a sequence of layers
- Neural networks are typically trained using stochastic gradient ascent
- The gradient can be efficiently calculated using the *backpropagation algorithm*
  - large number of "tricks" for learning
- The activation patterns of hidden units associated with each input can serve as low dimensional representations of that input

MACQUARIE University

Where to go from here

- Boltzmann Machines
- *Convolutional Neural Networks* (ConvNets) and weight tying:
  - CS231n: Convolutional neural networks for visual recognition, Stanford University
- *Recurrent Neural Networks* (RNNs) and sequence modelling:
  - CS224d: Deep learning for natural language processing, Stanford University

MACQUARIE University