# Statistical Machine Learning



Christian Walder



College of Engineering and Computer Science
The Australian National University



Canberra
Semester One, 2020.

(Many figures from C. M. Bishop, "Pattern Recognition and Machine Learning")

# Part VIII

# Neural Networks 2: Overview



- Recall: we would like gradients w.r.t. parameters so that we can optimise.
- Today: gradients of neural network parameters via the bac
- Re
- Inc
- Ba



*Good News*



We study back propagation for pedagogical reaso practice one uses automatic differentiation which is far more general and efficient (see *e.g.* the especially easy to use PyTorch).

# Chain Rule / Total Derivative



- The composition of two functions is given by

$$f \circ g(x) = f(g(x))$$

- Let $f$ and $g$ be differentiable functions with derivatives $f'$ and
- Ch



- If we write $u = g(x)$ and $y = f(u)$,

$$\frac{dy}{dx} = \frac{dy}{du}\frac{du}{dx}$$

- Multivariate case we also need is the total derivative, *e.g.*

$$\frac{\mathrm{d}}{\mathrm{d}t} f(x(t), y(t)) = \frac{\partial f}{\partial x}\frac{\mathrm{d}x}{\mathrm{d}t} + \frac{\partial f}{\partial y}\frac{\mathrm{d}y}{\mathrm{d}t},$$

# Error Backpropagation



- Goal: Efficiently update the weights in order to find a local minimum of some error function $E(\mathbf{w})$ utilizing the gradient of the error function.
- Core ideas :
  1. 
  2. 



- Sequential procedure : Calculate gradient and update weights for each data/target pair.
- Batch procedure : Collect gradient informatio data/target pairs for the same weight setting. T the weights.



- Main question in both cases: How to calculate the gradient of $E(\mathbf{w})$ given one data/target pair?

# *Error Backpropagation*





*Ong & Walder & Webers*
*Data61 | CSIRO*
*The Australian National University*





- Assume the error is a sum over errors for each data/target pair

  $N$

- Aft
  and

  $n$

- What is the gradient for one such term
- Note : In the following, we will drop the subscript to unclutter the equations.
- Notation: Input pattern is $\mathbf{x}$.
  Scalar $x_i$ is the $i^{th}$ component of the input pattern $\mathbf{x}$.

# Backprop - One Layer - Scalar View

- Simple linear model without hidden layers
- One layer only, identity function as activation function!



$$y_k = \sum_l w_{kl}\, x_l,$$

and



- The gradient with respect to $w_{ji}$ is now



$$\begin{aligned}
\frac{\partial E_n(\mathbf{w})}{\partial w_{ji}} &= \sum_k (y_k - t_k) \frac{\partial}{\partial w_{ji}} y_k = \sum_k (y_k - t_k) \frac{\partial}{\partial w_{ji}} \sum_l w_{kl}\, x_l \\
&= \sum_k (y_k - t_k) \sum_l x_l\, \delta_{jk} \delta_{il} \\
&= (y_j - t_j)\, x_i.
\end{aligned}$$

# Backprop - One Layer - Vector Calculus

- Vector setup:

$$\mathbf{y} = \mathbf{W}\,\mathbf{x}$$
$$\mathbf{W} \in \mathbb{R}^{D_2 \times D_1}$$
$$\mathbf{x} \in \mathbb{R}^{D_1}$$

- Err

$$E_n(\mathbf{W}) = \frac{1}{2}\left\|\mathbf{y} - \mathbf{t}\right\|^2$$

- Using the vector calculus rules gives

$$\begin{aligned}\nabla_{\mathbf{W}} E_n(\mathbf{W}) &= \nabla_{\mathbf{W}} \frac{1}{2}\left\|\mathbf{y} - \mathbf{t}\right\|^2 \\ &= (\mathbf{y} - \mathbf{t})\nabla_{\mathbf{W}}\mathbf{y} \\ &= (\mathbf{y} - \mathbf{t})\mathbf{x}^\top.\end{aligned}$$

# *FYI Only: Backprop - One Layer - Directional Derivative*

- Do the same using the directional derivative:

$$\mathbf{y} = \mathbf{W}\mathbf{x} \qquad \mathbf{W} \in \mathbb{R}^{D_2 \times D_1},$$

and error after applying input training pair $(\mathbf{x}, \mathbf{t})$

- De
- Rel $\nabla_{\mathbf{d}} \; ( \; ) = \quad ( \; ),$
- The directional derivative with respect to

$$\nabla_{\xi} E_n(\mathbf{W}) = \frac{1}{2} \left( (\xi \mathbf{x})^{\top} (\mathbf{y} - \mathbf{t}) + (\mathbf{y} - \mathbf{t}) \right.$$

- With canonical inner product $\langle A, B \rangle = \operatorname{tr}\left\{ A^{\top} B \right\}$ the gradient of $E_n(\mathbf{W})(\xi)$ is

$$\mathcal{D}E_n(\mathbf{W})(\xi) = \operatorname{tr}\left\{ \underbrace{\mathbf{x}^{\top} \xi^{\top} (\mathbf{y} - \mathbf{t})}_{\text{scalar}} \right\} = \operatorname{tr}\left\{ \xi^{\top} \underbrace{(\mathbf{y} - \mathbf{t}) \mathbf{x}^{\top}}_{\text{gradient}} \right\}$$

- The gradient

$$\nabla_{\mathbf{W}} E_n(\mathbf{W}) = (\mathbf{y} - \mathbf{t})\mathbf{x}^\top$$

  or in c



  looks like the product of the output error
  input $x_i$ associated with an edge for $w_{ji}$
  diagram.



- Can we generalise this idea to nonlinear activation functions?

# Error Backpropagation



- Now consider a network with nonlinear activation functions $h(\cdot)$ composed with the sum over the inputs $z_i$ in one layer and $z_j$ in the next layer connected by edges with weights $w_{ji}$



$$a_j = \quad w_{ji} z_i$$



- Us

$$\frac{\partial E_n(\mathbf{w})}{\partial w_{ji}} = \frac{\partial E_n(\mathbf{w})}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}}$$



where we defined the error (a slight misnomer hailing from the derivative of the squared error) $\delta_j = \frac{\partial E_n(\mathbf{w})}{\partial a_j}$

- Same intuition as before: gradient is output error times the input associated with the edge for $w_{ji}$.

# *Error Backpropagation*



- Need to calculate the errors $\delta$ in every layer.

$$\frac{\partial E_n(\mathbf{w})}{\partial w_{ji}} = \delta_j z_i \qquad \delta_j = \frac{\partial E_n(\mathbf{w})}{\partial a_j}$$



- Start the recursion; for output units with squared error:



- For t

$$\frac{\mathrm{d}}{\mathrm{d}t} f(x(t), y(t)) = \frac{\partial f}{\partial x}\frac{\mathrm{d}x}{\mathrm{d}t} +$$



to calculate

$$\delta_j = \frac{\partial E_n(\mathbf{w})}{\partial a_j} = \sum_k \frac{\partial E_n(\mathbf{w})}{\partial a_k}\frac{\partial a_k}{\partial a_j} = \sum_k \delta_k \frac{\partial a_k}{\partial a_j},$$

using the definition of $\delta_k$.

# Error Backpropagation





- Express $a_k$ as a function of the incoming $a_j$



$$a_k = \sum_j w_{kj} z_j = \sum_j w_{kj} h(a_j),$$

- and



$$\frac{\partial a_k}{\partial a_j} = w_{kj} \frac{\partial h(a_j)}{\partial a_j} = w_{kj} \frac{\partial h(s)}{\partial s}\bigg|_{s=a_j} = w_{kj}\, h'(a_j).$$



- Finally, we get for the error in the previous layer

$$\delta_j = h'(a_j) \sum_k w_{kj}\, \delta_k.$$

# Error Backpropagation

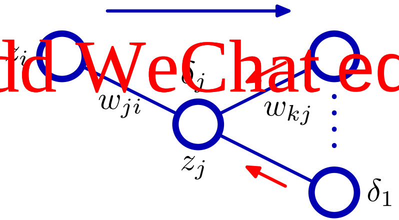
- The backfpropagation formula



$$\delta_j = h'(a_j) \sum_k w_{kj} \delta_k.$$

- For
acti
tan





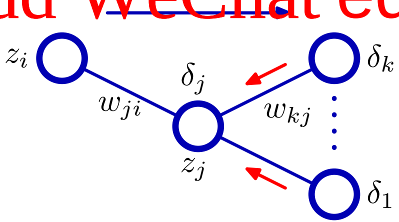
# Error Backpropagation Algorithms

1. Apply the input vector $\mathbf{x}$ to the network and forward propagate through the network to calculate all activations and outputs of each unit.
2. Compute the gradients of the error at the output.
3. Backpropagate the gradients backwards through the net
4. Cal

$$\overline{\phantom{xxxxx}}_{ji}$$

5. Update the weights $\mathbf{w}$ using $\frac{\partial E_n(\mathbf{w})}{\partial w_{ji}}$.





- For batch processing, we repeat backpropagation for each patt



$$n=1$$

- Backpropagation can be generalised by ass each node has a different activation function

# Easy Backprop

Let
$$
\begin{aligned}
z^{(0)} &= x = \text{input} \\
a^{(l)} &= W^{(l)} z^{(l-1)} \\
z^{(l)} &= h(a^{(l)}) \\
E &= \mathcal{L}(a^{(L)}) = \mathcal{L}(y) \overset{\text{e.g.}}{=} \frac{1}{2}\|y - t\|^2.
\end{aligned}
$$


The gradi

where (neglecting transposes — assume confor

$$
\delta^{(l)} \equiv \frac{\partial E}{\partial a^{(l)}} = \frac{\partial a^{(l+1)}}{\partial a^{(l)}} \frac{\partial a^{(l+2)}}{\partial a^{(l+1)}} \cdots \frac{\partial a}{\partial a^{(}}
$$

has the recursion $\delta^{(L)} = \frac{\partial \mathcal{L}(a^{(L)})}{\partial a^{(L)}}$ along with

$$
\delta^{(l-1)} = \frac{\partial a^{(l)}}{\partial a^{(l-1)}} \delta^{(l)}
$$

$$
\frac{\partial a^{(l)}}{\partial a^{(l-1)}} = \frac{\partial W^{(l)} h(a^{(l-1)})}{\partial a^{(l-1)}} = \operatorname{diag}\{h'(a^{(l-1)})\} W^{(l)\top}.
$$

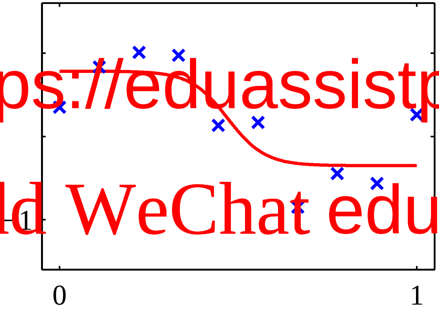
# Efficieny of Backpropagation

- For dense weight matrices, the complexity of calculating the gradient $\frac{\partial E_n(\mathbf{w})}{\partial w_{ji}}$ via backpropagation is of $O(W)$ where $W$ is the number of weights.
- Co

which needs $O(W^2)$ operations, and is les

FYI only — as in the previous lecture: In general we hav "cheap gradient principle". See (Griewank, A., 2000. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*, Section 5.1).

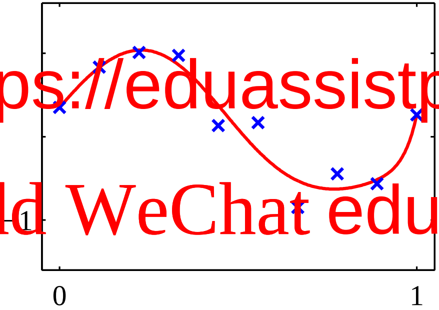
- Number of input and output nodes determined by the application.
- Number of hidden nodes is a free parameter.




Training a two-layer network with 1 hidden node.

- Number of input and output nodes determined by the application.
- Number of hidden nodes is a free parameter.

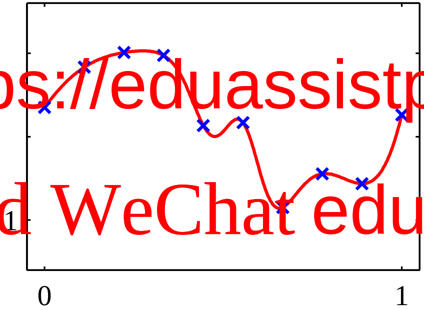





Training a two-layer network with 3 hidden nodes.

- Number of input and output nodes determined by the application.
- Number of hidden nodes is a free parameter.

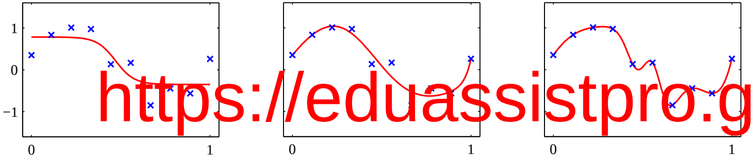





Training a two-layer network with 10 hidden nodes.

- Model complexity matters again.








$M = 1$ $M = 3$



- As before, we can use the regularised error

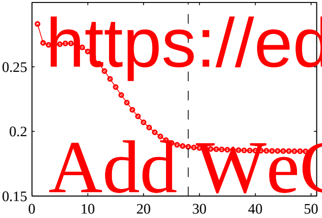
$$\widetilde{E}(\mathbf{w}) = E(\mathbf{w}) + \frac{\lambda}{2}\mathbf{w}^\top \mathbf{w}$$

# Regularisation via Early Stopping

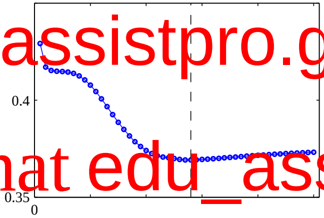




- Stop training at the minimum of the validation set error.




Training set error.




Validation set error.

# Invariances



- If input data should be invariant with respect to some transformations, we can utilise this for training.
- Use training patterns including these transformations (e.g. han
- Or c
  tran



- Alternatively, preprocess the input data to rem transformation.
- Or use convolutional neural networks (e.g. in im processing where close pixels are more correl away pixels; therefore extract local features first and later feed into a network extracting higher-order features).

# Synthetic Warping



- Create synthetic data by warping handwritten digits.



Left: Original digitised image. Right : Examples of warped images (above) and their corresponding displacement fields (below).

# Bayesian Neural Networks







- Predict a single target $t$ from a vector of inputs $\mathbf{x}$
- Assume conditional distribution to be Gaussian with precision $\beta$



$$p(t \,|\, \mathbf{x}, \mathbf{w}, \beta) = \mathcal{N}(t \,|\, y(\mathbf{x}, \mathbf{w}), \beta^{-1})$$

- Pri
  Ga



- For an i.i.d training data set $\{\mathbf{x}_n, t_n\}_{n=1}^{N}$, t
  targets $\mathcal{D} = \{t_1, \dots, t_N\}$ is



$$p(\mathcal{D} \,|\, \mathbf{w}, \beta) = \prod_{n=1}^{N} \mathcal{N}(t_n \,|\, y(\mathbf{x_n}, \mathbf{w}), \beta^{-1})$$

- Posterior distribution

$$p(\mathbf{w} \,|\, \mathcal{D}, \alpha, \beta) \propto p(\mathbf{w} \,|\, \alpha) p(\mathcal{D} \,|\, \mathbf{w}, \beta)$$

- But $y(\mathbf{x}, \mathbf{w})$ is nonlinear, and therefore we can no longer calculate the posterior in closed form.
- Use Laplace approximation
  1. Find a (local) maximum $\mathbf{w}_{MAP}$ of the posterior via numerical optimisation.
  2.
- Fin



$$\ln p(\mathbf{w} \,|\, \mathcal{D}, \alpha, \beta) = -\frac{\alpha}{2}\mathbf{w}^\top \mathbf{w} - \frac{\beta}{2}\sum_{n=1}^{N}(y($$

- Find the matrix of second derivatives of the nega posterior distribution

$$\mathbf{A} = -\nabla\nabla \ln p(\mathbf{w} \,|\, \mathcal{D}, \alpha, \beta) = \alpha\mathbf{I} + \beta\mathbf{H}$$

where $\mathbf{H}$ is the Hessian matrix of the sum-of-squares error function with respect to the components of $\mathbf{w}$.

Data61 | CSIRO
The Australian National University



# Bayesian Neural Networks

- Having $\mathbf{w}_{MAP}$, and $\mathbf{A}$, we can approximate the posterior by a Gaussian

$$q(\mathbf{w} \mid \mathcal{D}, \alpha, \beta) = \mathcal{N}(\mathbf{w} \mid \mathbf{w}_{MAP}, \mathbf{A}^{-1})$$

- For t

  Then

$$p(t \mid \mathbf{x}, \mathcal{D}, \alpha, \beta) = \mathcal{N}(t \mid y(\mathbf{x}, \mathbf{w}_M$$

  where

$$\sigma^2(\mathbf{x}) = \beta^{-1} + \mathbf{g}^\top \mathbf{A}^{-1} \mathbf{g}.$$

  (Recall the multivariate normal conditionals.)

- variance due to the intrinsic noise on the target: $\beta^{-1}$
- variance due to the model parameter $\mathbf{w}$ : $\mathbf{g}^\top \mathbf{A}^{-1} \mathbf{g}$