Statistical Machine Learning

Christian Walder

Machine Learning Research Group
CSIRO Data61

and

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")

Outlines

Part VIII

Neural Networks 2

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 backpropagation of gradients algorithm.
- Regularisation/model selection.
- Incorporating invariances/domain knowledge.
- Bayesian neural network (Laplace's method).

Good News

We study back propagation for pedagogical reasons: in 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 g' respectively
- Chain rule

$$\frac{d}{dx} f(g(x)) = f'(g(x))g'(x)$$

- 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. Propagate the errors backwards through the network to efficiently calculate the gradient.
  2. Update the weights using the calculated gradient.

- Sequential procedure : Calculate gradient and update weights for each data/target pair.
- Batch procedure : Collect gradient information for all data/target pairs for the same weight setting. Then adjust the weights.

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

Error Backpropagation

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

$$E(\mathbf{w}) = \sum_{n=1}^{N} E_n(\mathbf{w}).$$

- After applying input $\mathbf{x}_n$ to the network, we get the output $\mathbf{y}_n$ and calculate the error $E_n(\mathbf{w})$.
- What is the gradient for one such term $E_n(\mathbf{w})$?
- Note : In the following, we will drop the subscript n in order 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 error after applying input $\mathbf{x}_n$

$$E_n(\mathbf{w}) = \frac{1}{2}\sum_k (y_k - t_k)^2.$$

- 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:

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

- Error after applying input training pair $(\mathbf{x}, \mathbf{t})$

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

- 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 $(\mathbf{x}, \mathbf{t})$

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

- Define $\nabla_{\boldsymbol{d}} f(\boldsymbol{x}) = \frac{\mathrm{d}}{\mathrm{d}h} f(\boldsymbol{x} + h\boldsymbol{d})$.
- Relate this to the gradient by $\nabla_{\boldsymbol{d}} f(\boldsymbol{x}) = \langle \nabla f(\boldsymbol{x}), \boldsymbol{d} \rangle$.
- The directional derivative with respect to $\mathbf{W}$ is now

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

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

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

Error Backpropagation

- The gradient
$$\nabla_{\mathbf{W}} E_n(\mathbf{W}) = (\mathbf{y}_n - \mathbf{t})\mathbf{x}^\top$$
or in components
$$\frac{\partial E_n(\mathbf{w})}{\partial w_{ji}} = (y_j - t_j)\, x_i.$$
looks like the product of the output error $(y_j - t_j)$ with the input x_i associated with an edge for w_{ji} in the network 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}

$$a_j = \sum_i w_{ji} \, z_i$$

$$z_j = h(a_j).$$

- Use the chain rule to calculate the gradient

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

  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 δ 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:

$$\delta_k = y_k - t_k.$$

- For the hidden units we use the total derivative, *e.g.*

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

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 δ_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 differentiate

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

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

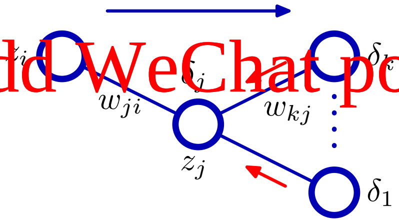
$$\delta_j = h'(a_j) \sum_k w_{kj}\, \delta_k.$$

Error Backpropagation

- The backfpropagation formula

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

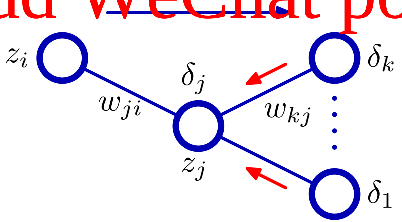
- Form of $h'(\cdot)$ is available, because we choose the activation function $h(\cdot)$ to be a fixed function such as a $\tanh$ *etc.*.



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 network using the backpropagation formula.
4. Calculate all components of ∇E_n by

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

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



- For batch processing, we repeat backpropagation for each pattern in the training set and then sum over all patterns

$$\frac{\partial E(\mathbf{w})}{\partial w_{ji}} = \sum_{n=1}^{N} \frac{\partial E_n(\mathbf{w})}{\partial w_{ji}}$$

- Backpropagation can be generalised by assuming that each node has a different activation function $h(\cdot)$.

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 gradients of E w.r.t. the parameters are (total derivative)

$$
\frac{\partial E}{\partial W^{(l)}} = \frac{\partial E}{\partial a^{(l)}} \cdot \frac{\partial a^{(l)}}{\partial W^{(l)}} = \delta^{(l)} {z^{(l-1)}}^{\top},
$$

where (neglecting transposes — assume conformant shapes)

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

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

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

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.
- Compare this to numerical differentiation using *e.g.*

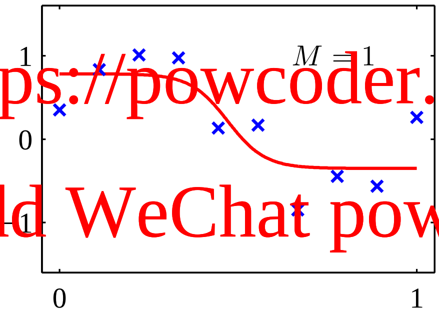
$$\frac{\partial E_n(\mathbf{w})}{\partial w_{ji}} = \frac{E_n(w_{ji} + \epsilon) - E_n(w_{ji} - \epsilon)}{2\epsilon} + O(\epsilon^2)$$

which needs $O(W^2)$ operations, and is less accurate.

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

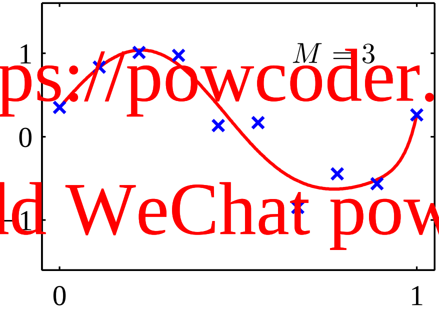
Regularisation in Neural Networks

- 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.

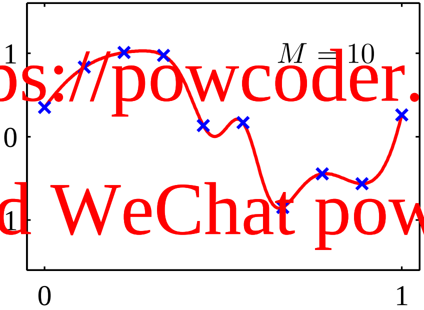
- Number of input and output nodes determined by the application.
- Number of hidden nodes is a free parameter.



Training a two-layer network with 3 hidden nodes.

Regularisation in Neural Networks

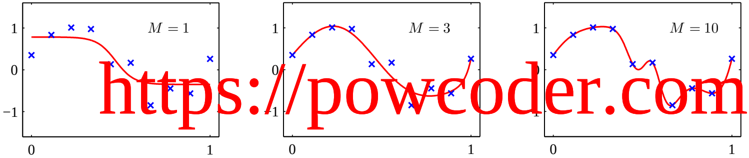
- Number of input and output nodes determined by the application.
- Number of hidden nodes is a free parameter.



Training a two-layer network with 10 hidden nodes.

Regularisation in Neural Networks

- Model complexity matters again.



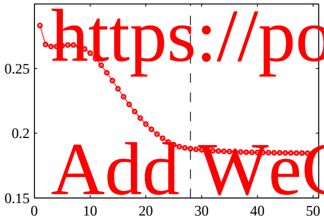
$M = 1$ $M = 3$ $M = 10$

- As before, we can use the regularised error

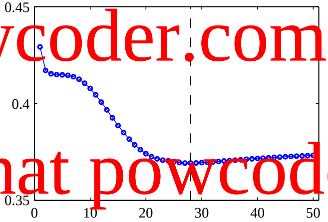
$$\widetilde{E}(\mathbf{w}) = E(\mathbf{w}) + \frac{\lambda}{2}\mathbf{w}^\top\mathbf{w}$$

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. handwritten digits translated in the input space).
- Or create extra artifical input data by applying several transformations to the original input data.
- Alternatively, preprocess the input data to remove the transformation.
- Or use convolutional neural networks (e.g. in image processing where close pixels are more correlated than far 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 β

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

- Prior distribution over weights $\mathbf{w}$ is also assumed to be Gaussian

$$p(\mathbf{w} \mid \alpha) = \mathcal{N}(\mathbf{w} \mid \mathbf{0}, \alpha^{-1}\mathbf{I})$$

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

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

- Posterior distribution

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

Bayesian Neural Networks

- 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. Evaluate the matrix of second derivatives of the negative log posterior distribution.
- Find a (local) maximum using the log-posterior

$$\ln p(\mathbf{w} \,|\, \mathcal{D}, \alpha, \beta) = -\frac{\alpha}{2} \mathbf{w}^\top \mathbf{w} - \frac{\beta}{2} \sum_{n=1}^{N} \left(y(\mathbf{x}_n, \mathbf{w}) - t_n \right)^2 + \text{const}$$

- Find the matrix of second derivatives of the negative log 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 $\mathbf{w}$.

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 the predictive distribution further linearly approximate

$$y(\mathbf{x}, \mathbf{w}) \approx y(\mathbf{x}, \mathbf{w}_{MAP}) + \mathbf{g}^{\top}(\mathbf{w} - \mathbf{w}_{MAP})$$

$$\text{where} \quad \mathbf{g} = \nabla_{\mathbf{w}} y(\mathbf{x}, \mathbf{w})|_{\mathbf{w}=\mathbf{w}_{MAP}}.$$

Then

$$p(t \mid \mathbf{x}, \mathcal{D}, \alpha, \beta) = \mathcal{N}(t \mid y(\mathbf{x}, \mathbf{w}_{MAP}), \sigma^2(\mathbf{x}))$$

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: β^{-1}
- variance due to the model parameter $\mathbf{w}$: $\mathbf{g}^{\top}\mathbf{A}^{-1}\mathbf{g}$