COMS30035, Machine learning: Training Neural Networks

James Cussens

School of Computer Science
University of Bristol

24th September 2024

Problems and solutions when training neural networks

- Once we have computed the gradient (at the current value of the weights) . . .
- . . . we can just send that gradient to an off-the-shelf gradient based optimiser to reduce error (on the training set).
- That can work fine, but not always.
- In this lecture we look at various problems that may arise and how they are dealt with.

Lecture content

- This lecture is nothing more than a summary of (some of) the material that can be found in Chapter 13 of Kevin Murphy's book [Mur22]
- We'll be switching from these slides to look at some of the figures in that Chapter occasionally.

Vanishing and exploding gradients

- When training deep models (ones with many layers) the gradient can become very small or very large.
- These are the *vanishing gradient problem* and *exploding gradient problem* respectively.

Gradient clipping

- *Gradient clipping* can be used to cap the magnitude of the gradient.
- The original gradient $\mathbf{g}$ and the clipped gradient $\mathbf{g}'$ are related as follows:

$$\mathbf{g}' = \min\left(1, \frac{c}{||\mathbf{g}||}\right)\mathbf{g}$$

- Note that $\mathbf{g}'$ points in the same direction as $\mathbf{g}$.

Non-saturating activation functions for deep learning

- The sigmoid activation function, for example, *saturates* at large negative or large positive values.
- So its gradient there will be 0 preventing gradient descent from reaching good weights.
- A solution to this is simply to choose activation functions that don't saturate.
- Let's look at [Mur22, Fig 13.4].
- ReLU and its variants are popular choices. But not that plain ReLU has zero gradient for negative values so we might end up with the 'dead ReLU' problem.
- Note also that ReLU is not a differentiable function at 0, but we can just impose a derivative value of 0 there.

Using residual connections

- In a *residual network* each layer has the form of a *residual layer* defined as:

$$\mathcal{F}'_l(\mathbf{x}) = \mathcal{F}_l(\mathbf{x}) + \mathbf{x}$$

- where $\mathcal{F}_l(\mathbf{x})$ is the standard linear-activation-linear mapping.
- The weights defining $\mathcal{F}_l(\mathbf{x})$ learn what needs to be **added to** the input $\mathbf{x}$.
- This provides a 'short cut' for gradients to flow directly from the output layer to any previous layer.
- Let's look at [Mur22, Fig 13.15].

Parameter initialisation

- Neural network training is non-convex, . . .
- . . . i.e. the error 'landscape' has many 'valleys' and standard gradient descent just ends up at the bottom of whichever valley it happens to start in.
- So the choice of initial weights—*parameter initialisation*—matters.
- We'll see the same issue arise when we later look at another iterative algorithm: the EM algorithm.
- I've decided not to examine the various methods used for parameter initialisation in neural networks: it's enough that you know that it matters.

Dealing with overfitting

- Neural networks often have many parameters (ChatGPT has 175 billion!) so overfitting is an issue.
- If we have a very large training dataset the problem is, at least, greatly reduced. (GPT-3.5 used 570GB of data, about 300 billion words).
- But we are not always swimming in a vast sea of data!
- Let's look at some common ways of reducing overfitting in neural networks.

Early stopping

- Create a validation set.
- Stop training when error on the validation set starts to increase.
- (Of course, using a validation set to prevent overfitting is not unique to neural networks.)

Weight decay

- We can alter the loss function to penalise large weights.
- *Weight decay*, also known as ℓ_2 *regularisation*, adds a term $\lambda \mathbf{w}^\top \mathbf{w}$ to the loss function, where $\mathbf{w}$ is the weight vector and the user chooses the value of $\lambda > 0$.
- Murphy writes $\mathbf{w}^\top \mathbf{w}$ as $||\mathbf{w}||_2^2$ which makes it explicit that we are calculating (the square of) the 2-norm (aka Euclidean norm). See Murphy §7.1.3 for more on norms if you're interested.
- In scikit-learn a parameter called `alpha` sets λ to `alpha`$/N$, where N is the size of the training data. The default value of `alpha` is 0.0001, so we get a little ℓ_2 *regularisation*.
- Since the loss we are trying to minimise is now not just about fitting the data, overfitting is reduced.

Dropout

- *Dropout* is where, during training, we randomly turn of all outgoing connections from a unit with some given probability p.
- Do this on a per-example basis : "For each presentation of each training case, a new thinned network is sampled and trained." [SHK+14]
- Let's look at [Mur22, Fig 13.18].
- As Murphy puts it: " . . . each unit must learn to perform well even if some of the other units are missing at random. This prevents the units from learning complex, but fragile, dependencies on each other."
- Dropout can drastically reduce overfitting and is used a lot.
- Interesting comment from the inventors of dropout: ". . . we would like to approach the performance of the Bayesian gold standard using considerably less computation." [SHK+14]

We're missing out a lot, as ever

- There is of course much more to learn about neural networks.
- Most notably we have not looked into how the choice of *architecture* (neural network structure) is made ...
- ... for example, convolutional NNs for image data.
- Also there has recently been a lot of attention on *attention*: where the weights can depend on the input.
- We'll finish off our look at neural nets by going through some examples with PyTorch ...

Reading

- Murphy The relevant sections of Chapter 13.

Problems and quizzes

- No problems
- Quizzes:
  - Week 2: Training Neural Networks


Kevin P. Murphy.
Probabilistic Machine Learning: An introduction.
MIT Press, 2022.


Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov.
Dropout: A simple way to prevent neural networks from overfitting.
Journal of Machine Learning Research, 15(56):1929–1958, 2014.