Course matters

• Please attend tutorials!
• Solutions to tutorial 1 will be up soon

• Coursework will be out by end of day on Friday.
Neural networks

• Perceptron activation functions
• Each perceptron defines a half plane
• Together they can form complex boundaries using arrangements of half spaces
• More perceptrons, more options for regions available in the arrangement of half spaces
Challenges with perceptron and 0-1 values

• Gradients are not always useful
  • Eg. If a small change does not change the classification of any point
  • Hard to apply SGD type methods

• Sometimes it is useful to have real values
Other activations

• Sigmoid
  • \( f(x) = \frac{1}{1+e^x} \)

• ReLU
  • \( f(x) = \max(0, x) \)
Neural network structure

• Use ReLU or similar activation functions
  • More compatible with gradients
  • Easy to compute

• The middle layers produce a vector $\mathbf{y}$ of ”scores” for each class, called logit values

• Final layer: apply “softmax” to logits:
  • $softmax(\mathbf{y}_i) = \frac{e^{y_i}}{\sum e^{y_j}}$ (improved the notation from the lecture)
Question: Why softmax?
Hard max or exact max

• Take a vector of values eg. [2,3,5,2,6,4,9,2,2,4]
• Make one indicating the position of the max eg. [0,0,0,0,0,0,1,0,0,0]
Softmax

• Substitute for hard-max, but differentiable
• Normalized, can be treated as probability $p_i$ for each class
Cross entropy loss

• Neural networks are usually trained on the cross entropy loss of their output $p$

• Given:
  • Data point $x$
  • Probability estimate vector $p$
  • Truth label vector $t$: indicator vector or one-hot encoding where only the true class has value 1.

• Cross entropy loss: $\ell_{CE} = -\sum t_i \ln p_i$
  • Measures difference between the two probability distributions

$p=[0.1, 0.5, 0.2, 0.2]$
$t= [0.0, 1.0, 0.0, 0.0]$
Generalisation gap for neural networks: How does it grow?

• What do curves look like for training loss and test loss?
Generalisation gap for neural networks

What we might expect

What we find
Overfitting in neural networks

• The role of cross entropy loss
• Consider probability outputs for this data and this data space
  • One curve for each class
• What should the curves look like?
Probability curves for classification

• A reasonable model sacrifices the outlier for better generalization
• But what is the cross entropy loss at the outlier?
Overfitting

- Optimiser tries to modify probability curves
- Such that large CE losses become smaller
What prevents the NN from overfitting too much to every point?

- The NN architecture restricts the possible arrangements of hyperplanes
- The architecture and activation functions restrict
Overparameterised neural networks

• Idea:
  • More neurons/weights/parameters: more unknown variables
  • More data points: More information (similar to more equations)

• Recap of statistical ML: data requirements grow with parameters/complexity

• Modern neural networks:
  • Many more parameters than data points
  • High complexity and therefore high estimation error
  • We expect heavy overfitting and high test/generalization loss/error
Double descent

• With very large number of parameters (more than number of data points) testing performs well again!

• Out of many possible models with low training loss, SGD is finding ones that have low test loss!

• See also (optional): Neural Tangent kernels
Distribution of weights on trained NNs

• A large fraction of weights are close to zero
• Small fraction is far from zero
• Observation:
  • Zero weight edges have no effect – do not conduct information
  • Almost zero weights: Little effect

• Conclusion: While NNs have large number of parameters, after training, many of them have little to no effect!
Pruning

• Idea: take all the edges that are tiny weights, and remove them!

• Observations
  • Can sometimes remove 80% - 90% of edges
  • Retains comparable performance and sometimes better generalization
Lottery ticket hypothesis

• Hypothesis: A randomly initialised dense NN already contains a subnetwork (a winning ticket) that can give good performance.

• Algorithm to find the winning ticket
  • Initialise a network to random weights
  • Train for some iterations
  • Prune p% of edges with small weights
  • Reset the remaining edge to their original random weights

• Works surprisingly well on MNIST, CIFAR with test performance comparable to a well trained network [Frankle and Carbin, 2019]
Standard pruning methods

• One shot:
  • Train
  • Remove small weights
  • Return to initialization weights and retrain
  • Stop

• Iterative
  • Set random weights
  • Train
  • Remove edges with small weights
  • Start over
Other results

• Theoretical proofs (special cases, few layers etc)
  • [Malach et al. 2020, Bartoldson et al. 2020]

• Pruning and finding winning tickets without data
  • [Wang et al. 2020, Tanaka et al. 2020]
Pruning and dimension

- The dimension of $\mathcal{H}$ is determined by the number of parameters.
- The pruning and lottery tickets papers suggest that there are low-dimensional subspaces of $\mathcal{H}$ that contain good solutions.
Question

• If a small network is good enough, why are we using a large one?
Shape of minima

• Why it is good
• Hessians and eps
Flat and sharp minima

• A minimum of the loss function can be flat or sharp

• Which is better?
Flat and sharp minima

• Flat minima generalize better
• Sharper minima likely to represent overfitting
Flat minima are also more likely to be stable
Curvature as a sharpness measure

• For the min of a real valued function in 1-D we can measure curvature as the second derivative
  • \[
  \frac{d^2 y}{dx^2}
  \]

• For loss over models
  • \[
  \frac{d^2 L}{dw^2}
  \]

• Larger second derivative => sharper min
\( \varepsilon \)-Sharpness

- At min \( \theta \) take ball \( B(\theta, \varepsilon) \) of radius \( \varepsilon \)
- Set of all points within a distance \( \varepsilon \) of \( \theta \)
- Sharpness is:

\[
\max_{\theta' \in B_2(\varepsilon, \theta)} \frac{L(\theta') - L(\theta)}{1 + L(\theta)}
\]
Model spaces are high dimensional

- $\epsilon$ – Sharpness definition applies directly

- Curvature requires considering the Hessian – high dim representation of $2^{nd}$ derivative
Partial derivatives

• Suppose $f$ is a function of many variables $x, y, z, ...$

• We can ask how $f$ changes with $x$. This is written as $\frac{\partial f}{\partial x}$
  • Same as $\frac{df}{dx}$, but implying that there are other variables to potentially consider
  • And we can write the curvature along $x$ as $\frac{\partial^2 f}{\partial x^2}$: how $\frac{\partial f}{\partial x}$ changes with $x$
Partial derivatives

• Now we can also ask how $\frac{\partial f}{\partial x}$ changes with $y$
• This is written as $\frac{\partial^2 f}{\partial y \partial x}$
• Hessian is just a collection of all these written as a matrix
• With two variable models:

$$
\left( \begin{array}{cc}
\frac{\partial^2 f}{\partial w_1^2} & \frac{\partial^2 f}{\partial w_1 \partial w_2} \\
\frac{\partial^2 f}{\partial w_2 \partial w_1} & \frac{\partial^2 f}{\partial w_2^2}
\end{array} \right)
$$

$$
H_f =
\begin{bmatrix}
\frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\
\frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\
\vdots & \vdots & \ddots & \vdots \\
\frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2}
\end{bmatrix}
$$
Curvature directions

- The problem is that strongest directions of curvature may not align exactly with $w_1, w_2$ etc.
- So, we need to take eigen values and eigen vectors of the hessian.
- The eigen values represent the principal curvatures. Corresponding eigen vectors represent the directions of these curvatures.
- Larger eigen values of hessian imply sharper minima.

(Think Principal components of curvature matrix)
So, the method is

- Take the hessian
- Compute its eigen values
- Look at their distributions
- If there are more of large values, that implies a sharper min
Algorithms

• Shapness aware minimization
  • Use $\epsilon$ sharpness
  • Minimize $L(\theta) + [L(\theta + \epsilon') - L(\theta)]$

• Entropy SGD
  • Optimise a different function
  • Computationally very expensive

• Stochastic weight averaging
  • Average the weights of the last $c$ models
  • Shown to produce flat minima
Flat minima

• Current topic of research
• While flat minima are generally agreed to be good, the full picture is not clear
• There are works showing that sometimes sharp minima can work well
• Neural nets are highly redundant (e.g. symmetric) and many possible weight assignments achieve the same effective function
  • It is possible to reconfigure weights such that the effective prediction function is same, therefore loss is same, but the curvature is different
SGD and Flat minima

• SGD is known to have a bias toward flat and well generalizable min
• Large batch sizes and small learning rate approximates a smooth gradient
  • And more likely to find a sharp min
• Small batch sizes and larger learning rate makes a more random, jumpy trajectory that can skip over sharp min.
  • Also easy to jump away from sharp min neighborhood since that is likely a small region
• However, a flat min means that even after step away from it, SGD is likely in the same basin