10 Interpreting output of Deep Nets: Credit Attribution

This chapter considers methods that try to understand: *Why did the model give the answer it did?* The basic notions are quite old, but proper and efficient application to deep learning settings is fairly recent. Mathematically, what is needed is to do *credit attribution* for the final decision to various components of the system, including training data.

We consider two types of explanations. *Influence functions* try to understand how individual data points affect the model's answers on test data points. *Saliency methods* try to understand the model's answer on a test data point in terms of the contents of that same data point, typically in the form of a heat map (also called *saliency maps*) depicting the importance of individual coordinates. An elegant idea that often arises in these settings is *Shapley values*.

10.1 Influence Functions

For a fixed training dataset S, the *influence function* captures how adding or removing a datapoint x from the training set S affects the answer (or the loss) on a test datapoint z. Cook and Weisberg's text [1] is the standard reference on the topic. The naive way to compute the influence function is *leave-one-out retraining*: for every x, recompute the model on $S \setminus \{x\}$. But it is also possible to do a more direct computation using the model θ^* trained on S, which uses a more continuous notion of influence.

[1] R D Cook and S Weisberg. Residuals and influence in regression. 1982

Let $\ell()$ be a twice-differentiable loss function with $\ell(x, \theta)$ denoting the () loss of model parameters θ on datapoint x. For succinctness we let $\ell(S, \theta)$ denote the average loss on a set S of data points.

We assume the trained model is θ^* and the classical theory of influence functions assumes this is a *local minimum,* [2] meaning that the following conditions hold: (i) $\nabla_\theta(\ell(S, \theta^*)) = 0$. (ii) The Hessian $\nabla^2_\theta(\ell(S, \theta^*))$ (also denoted by H_{θ^*}) is positive semi-definite. In practice neither holds exactly, but $\nabla^2_\theta(S, \theta^*)$ usually does not have

[2] In a deep learning setting one hopes to get to a *stationary point*, i.e., $\nabla_\theta(S, \theta^*)$ is zero (or more-realistically, near-zero), which in addition is a *local optimum*, i.e., $\nabla^2_\theta(S, \theta^*)$ is positive semi-definite.

large negative eigenvalues. So the influence function computation in practice uses $(H_{\theta^*} + \lambda I)$ for some small $\lambda > 0$ that makes the matrix positive semi-definite.

The formal[3] definition of influence $I(x,z)$ involves the thought experiment of modifying the weighting of a training point x from $1/|S|$ to $1/|S| + \epsilon$. With θ^* be defined as above, we denote by $\theta^*_{x,\epsilon}$ be the minimizer after the perturbation.

[3] Better notation might be $I_S(x,z)$, to clarify dependence on S.

Definition 10.1.1. $I(x,z) = \frac{\partial}{\partial \epsilon}\ell(z, \theta^*_{x,\epsilon})|_{\epsilon=0}$.

Influence functions were invented for convex models such as least-squares linear regression, and thus the theory assumes θ^* satisfies $\nabla_\theta(S,\theta^*) = 0$ and $\nabla^2_\theta(S,\theta^*)$ is positive semi-definite.

Theorem 10.1.2. $I(x,z) = -\nabla_\theta(\ell(z,\theta^*))^T H^{-1}_{\theta^*} \nabla_\theta \ell(x,\theta^*)$.

Proof. By optimality, $\ell(S,\theta^* + \Delta\theta) \approx \ell(S,\theta^*) + \frac{1}{2}(\Delta\theta)^T H_{\theta^*}(\Delta\theta)$ for small perturbations $\Delta\theta$. Changing the weight on datapoint x from $1/\{S\}$ to $1/\{S\} + \epsilon$ and re-optimizing gives $\theta^*_{x,\epsilon} = \theta^* + \Delta\theta$ where $\Delta\theta$ is a minimizer of

$$\epsilon\ell(x, \theta^* + \Delta\theta) + \frac{1}{2}(\Delta\theta)^T H_{\theta^*}(\Delta\theta).$$

Since $\ell(x,\theta^* + \Delta\theta) \approx \ell(x,\theta^*) + \nabla\ell(x,\theta^*)\cdot\Delta\theta$ what we have is a quadratic expression and it is minimized by $\Delta\theta = -\epsilon H^{-1}_{\theta^*}\nabla_\theta(\ell(x,\theta^*))$. Since a change in parameters from θ^* to $\theta^* + \Delta\theta$ causes the loss on a test point z to change by $(\Delta\theta) \times \nabla_\theta(\ell(z,\theta^*))$, the theorem now follows. □

10.1.1 More detailed explanation by Gemini

Let the training set be $S = x_1, \ldots, x_n$, and let $\ell(\cdot, \theta)$ be a twice-differentiable loss function.

Setup The model parameters θ^* are found by minimizing the **average loss** over the training set, $L(S,\theta) = \frac{1}{n}\sum_{i\in S}\ell(x_i,\theta)$. The solution θ^* must satisfy the first-order optimality condition:

$$\nabla_\theta L(S,\theta) = \mathbf{0} \tag{10.1}$$

The **Hessian** of the average loss at this minimum is $\mathbf{H}_{\theta^*} = \nabla^2_\theta L(S,\theta^)$. We assume $\mathbf{H}_{\theta^*}$ is positive definite and thus invertible.

The Perturbation To measure the influence of a single training point x, we consider a new loss function where x is up-weighted by a small scalar ϵ:

$$L_\epsilon(\theta) = L(S,\theta) + \epsilon\ell(x,\theta) \tag{10.2}$$

Let θ^ϵ be the parameters that minimize this perturbed loss. The new optimality condition is:

$$\nabla_\theta L(S, \theta^\epsilon) + \epsilon \nabla_\theta \ell(x, \theta^*_\epsilon) = \mathbf{0} \qquad (10.3)$$

Deriving the Parameter Change We apply a first-order Taylor expansion to the gradient term in (10.3) around the original optimum θ:

$$\nabla_\theta L(S, \theta^\epsilon) \approx \nabla_\theta L(S, \theta^*) + \mathbf{H}_{\theta^*}(\theta^\epsilon - \theta^*) \qquad (10.4)$$

By the optimality condition in (10.1), the first term on the right, $\nabla_\theta L(S, \theta^*)$, is zero. This simplifies the approximation to:

$$\nabla_\theta L(S, \theta^\epsilon) \approx \mathbf{H}_{\theta^*}(\theta^\epsilon - \theta^*) \qquad (10.5)$$

Substituting this back into the perturbed optimality condition (10.3) gives:

$$\mathbf{H}_{\theta^*}(\theta^\epsilon - \theta^*) + \epsilon \nabla_\theta \ell(x, \theta^*_\epsilon) \approx \mathbf{0} \qquad (10.6)$$

Since $\theta^\epsilon \approx \theta^*$, we can assume that $\nabla_\theta \ell(x, \theta^\epsilon) \approx \nabla_\theta \ell(x, \theta^*)$. Rearranging the equation yields the response of the parameters to the perturbation:

$$\theta^*_\epsilon - \theta^* \approx -\epsilon \mathbf{H}^{-1}_{\theta^*} \nabla_\theta \ell(x, \theta^*) \qquad (10.7)$$

Influence on a Test Point The influence of the training point x on a test point z is the rate of change of the loss on z with respect to the perturbation ϵ, evaluated at $\epsilon = 0$.

$$I(x, z) = \left. \frac{d\ell(z, \theta^\epsilon)}{d\epsilon} \right|_{\epsilon=0} \qquad (10.8)$$

Applying the chain rule:

$$I(x, z) = \nabla_\theta \ell(z, \theta)^T \left. \frac{d\theta^\epsilon}{d\epsilon} \right|_{\epsilon=0} \qquad (10.9)$$

From our approximation in (10.7), we have $\left. \frac{d\theta^\epsilon}{d\epsilon} \right|_{\epsilon=0} = -\mathbf{H}^{-1}_{\theta^*} \nabla_\theta \ell(x, \theta)$. Substituting this in gives the final influence function formula:

$$I(x, z) = -\nabla_\theta \ell(z, \theta)^T \mathbf{H}^{-1}_{\theta^*} \nabla_\theta \ell(x, \theta^*) \qquad (10.10)$$

10.1.2 *Computing Influence Functions*

At first sight, computing influence functions appears difficult due to the inverse Hessian computation, which naively has cubic complexity in the number of parameters. Koh and Liang [4] designed much faster methods. They use an idea of Agarwal et al.[5] who noted how to use the matrix power series for fast but approximate computation of the form $H^{-1}v$. The key idea is a simple identify in the following question.

[4] P W Koh and P Liang. Understanding black-box predictions via influence functions. In *Proc. ICML*, 2017

[5] Agarwal N, Bullins B, and Hazan E. Second-order stochastic optimization for machine learning in linear time. 2017

Problem 10.1.3. *If A is any positive definite matrix with full rank and has max eigenvalue less than 2 then show that*[6] $A^{-1} = \sum_{i=0}^{\infty}(I - A)^i$.

[6] Hint: Note that A is assumed to be diagonalizable. How do eigenvectors and eigenvalues of A^i relate to those of A?

Specifically, we want to compute $H^{-1}v$ where H is the Hessian and v is a vector. However H may have eigenvalues outside $[0, 1]$ and in particular not be positive semidefinite. So instead of $I - H$ one uses a surrogate like $I - \alpha(H + \lambda I)$ for small α, λ, which amounts to $\gamma I - \delta H$ where γ is close to 1 and δ is small.

Problem 10.1.4. *If S_r denotes the truncation of the series to its first r terms, then show that $\lambda_{max}(A^{-1} - S_r) \leq \lambda_{max}(A)^{-1}(1 - \lambda_{max}(A))^{r+1}$. Compute a value of r for which the r-term approximation to the Hessian-vector product is within $(1 + \epsilon)$ multiplicative factor of the correct value.*

Theorem 10.1.2 shows that we need to compute $H^{-1}v$ for some vector v, but Problem 10.1.4 allows us to approximate it as $\sum_{i=0}^{r}(I - H)^i v$ for some reasonably small r. Since $(I - H)v = v - Hv$ we see that to compute $H^r v$ it suffices to do a sequence of r Hessian-vector computations, each of which takes computation time linear in the size of the deep net (with an additional multiplicative factor of r). See Section 3.4.3.

10.2 Shapley Values

Shapley Values [7] is a concept from cooperative game theory that assigns "credit" to members of a multi-party collaboration. The setting is as follows. There is a population of N players (for brevity the set of players is denoted $[N]$) who are willing to co-operate towards a certain goal. A utility function U stipulates the reward/utility for each subset of players: If a subset S of the players end up cooperating, they receive utility $U(S)$ receive as a group. If all N of them end up cooperating, what is the appropriate and fair way to split the utility $U([N])$ among them? Under some reasonable conditions, there turn out to be unique payments $s_1, s_2, \ldots, s_N$, called *Shapley values* such that $\sum_i s_i = U([N])$ (Theorem 10.2.5).

[7] Lloyd Shapley. *"Notes on the n-Person Game – II: The Value of an n-Person Game"*. RAND Corporation

Example 10.2.1 (Pricing of datapoints). *Suppose model training uses contributions from N human annotators. They can be viewed as "players"each holding data that could be useful for training an ML model. If we select data from a subset $\mathcal{S}$ of players, then $U(S)$ could be a measure of the quality of the model, and the payments represent fair reimbursement for use of their data.*

Another representative setting is when we try to understand the output of a deep net on a single (test) datapoint in terms of the contributions (aka *saliency*) of individual coordinates towards the deep net's decision. For example, if the datapoint is an image, then the coordinates are pixels. For example, *which pixels led the model to decide the picture contains a dog and not a cat*? Such questions are central to model interpretability.

Shapley value of the ith player is defined using the thought experiment of the players adding themselves to the coalition in a random order, and looking at the expected increase in utility when the ith player joins the coalition.

Definition 10.2.2 (Shapley Value)**.** *Shapley value of player i, denoted s_i, is defined as the following expected value, where π is a random permutation of $\{1, 2, \ldots, N\}$ and $\pi_{<i}$ is shorthand for the subset of players that appear before i in the permutation:*

$$s_i = E_\pi \left[U(\pi_{\leq i}) - U(\pi_{<i}) \right]. \tag{10.11}$$

The definition of s_i's is sort of natural, though one may quibble about the slightly artificial assumption of the players joining the coalition in a random order —which seems to have no meaning in context of datasets for deep learning.

Here is an equivalent definition that avoids the random ordering and feels more natural to students.

Definition 10.2.3 (Alternative Definition)**.** *Shapley value s_i as the expectation of the following random process:* randomly pick an integer k from $[0, N-1]$, then pick a random subset S of size k but not containing i. Then measure the change in utility upon adding i to S.

The equivalence follows easily from the following result, and the fact that $\binom{N-1}{k}$ is the number of subsets of size k in a set of size $N-1$.

Theorem 10.2.4. *Shapley value as defined in Definition 10.2.2 has the following formula:*

$$\begin{aligned} s_i &= \textstyle\sum_{S \subseteq [N] \backslash \{i\}} \frac{\{S\}!(N-\{S\}-1)!)}{N!} \left(U(S \cup \{i\}) - U(S) \right) & (10.12) \\ &= \frac{1}{N} \textstyle\sum_{S \subseteq [N] \backslash \{i\}} \frac{U(S \cup \{i\}) - U(S)}{\binom{N-1}{|S|}}. & (10.13) \end{aligned}$$

Proof. Follows from writing down an expression for the expectation in Definition 10.2.2, using the count for the number of subsets of size k. □

10.2.1 Are there alternatives to Shapley Values?

While the above all seems natural, students wonder about the seeming arbitrariness of the definition, and especially whether there exists some radically different method for distributing credit in such cooperative settings. To understand this issue, let's try to formalize natural properties ("axioms") that must be satisfied for any method of credit attribution, and then understand how many methods can satisfy those properties. The following axioms seem natural for any system of defining s_i's.

Efficiency: Sum of the players' values is $U([N])$.

Symmetry: If $U(S \cup \{i\}) = U(S \cup \{j\})$ for all S not containing i, j then the payments to i, j are the same. [8]

Linearity: If U_1, U_2 are any two utility functions then the payments for $U_1 + U_2$ are the sum of the payments for U_1 and the payments for U_2. [9]

Null Player: If $U(S \cup \{i\}) = U(S)$ for all S not containing i, then the payment for i is zero.

[8] Sometimes this is described as "payments should depend upon their contribution, not on their names."

[9] Linearity can be a problematic axiom when we try to apply Shapley values in AI and ML; see the discussion later in Section 10.5.1.

You can quickly convince yourself that Shapley values satisfy the axioms.

Theorem 10.2.5. *The payment scheme in Definition 10.2.2 is the only one that satisfies the previous axioms.*

Proof. 1. **Define Unanimity Games:** For any non-empty coalition $T \subseteq [N]$, define a simple utility function u_T called a unanimity game:

$$u_T(S) = \begin{cases} 1 & \text{if } T \subseteq S, \\ 0 & \text{otherwise.} \end{cases}$$

In this game, the coalition T gets a reward of 1 if all its members are present, and zero otherwise.

2. **Decomposition:** Show that any utility function U can be uniquely expressed as a linear combination of these unanimity games:

$$U(S) = \sum_{T \subseteq [N]} c_T u_T(S)$$

The coefficients c_T can be found via the principle of inclusion-exclusion.

3. **Apply Axioms to Unanimity Games:** Let $\phi_i(u_T)$ be the payment to player i for the game u_T. The axioms uniquely determine these payments:

- **Null Player:** If $i \notin T$, player i never contributes to the utility. Thus, $\phi_i(u_T) = 0$.

- **Symmetry:** All players $i \in T$ are symmetric. Their contributions are identical, so their payments must be equal: $\phi_i(u_T) = \phi_j(u_T)$ for all $i, j \in T$.

- **Efficiency:** The sum of payments must equal the total utility $u_T([N]) = 1$. Since only the $|T|$ players in T receive non-zero payments, and they all receive the same amount, it must be that $\phi_i(u_T) = 1/|T|$ for $i \in T$.

4. **Synthesize using Linearity:** Since the payments for each basis game u_T are uniquely determined, the **Linearity** axiom dictates that the payments for the general game U must also be unique:

$$\phi_i(U) = \phi_i\left(\sum_{T\subseteq[N]} c_T u_T\right) = \sum_{T\subseteq[N]} c_T \phi_i(u_T)$$

Since the axioms uniquely determine the payment scheme, and we have already established that Shapley values satisfy these axioms, the Shapley value scheme must be this unique solution. □

10.2.2 *Algorithms to approximate Shapley values*

Given a succinct description of the utility function U (e.g., as a circuit or formula or computer code) it is NP-hard in general to compute Shapley values [10]. This means (assuming $P \neq NP$) that the running time is going to increase faster than any polynomial of N and the description of U. [11] However, it is possible to compute them approximately if the utilities are bounded. Specifically, we assume an upper bound of R on the absolute value of $U(S \cup \{i\}) - U(S)$ for all S, i. (This means there is also an absolute upper bound of $2R$ on the difference $s_i - s_j$ for any i, j, which will be relevant later.)
Naive approximation: Pick $O(R^2 N \log N/\epsilon^2)$ random permutations and use them to estimate the expectation in (10.11) for every i. In other words, the total number of evaluations of $\mathcal{U}$ is $O(R^2N^2 \log N/\epsilon^2)$.

[10] X Deng and C Papadimitriou. On the complexity of cooperative solution concepts. *Math of Operations Research*, 1994

[11] Notice that computing the sum in (10.13) also requires evaluating $U(S)$ for all possible subsets of $[N]$, which is 2^N evaluations. The NP-hardness says such exponential running time may be unavoidable.

Lemma 10.2.6. *This method with high probability approximates the vector of all Shapley values within ℓ_2 norm ϵ, namely, $\|s - \widehat{s}\|_2 \leq \epsilon$.*

Proof. Recall that independent sampling causes the variance of the error from the true estimate to drop inversely with the number of samples.

Thus concentration bounds imply that for i the estimate $\widehat{s_i}$ of the expectation lies within $[s_i - \epsilon/\sqrt{N}, s_i + \epsilon/\sqrt{N}]$, Which implies that the vector of all Shapley values is estimated within ℓ_2 norm ϵ, namely, $\|s - \widehat{s}\|_2 \leq \epsilon$. □

Better Approximation method for Shapley Values: (1) Use naive approximation to approximate s_1 within additive error $\epsilon/2\sqrt{N}$. (2) Use the characterization in Theorem 10.2.8 to sample sets S suitably to estimate all differences of type $s_1 - s_j$ within error $\epsilon/2\sqrt{N}$.

The following is left to the reader.

Problem 10.2.7. *Figure out how to do step (2) and give an analysis for it. (Hint: You pick S with a certain probability $p(|S|)$, inspired by Theorem 10.2.8. The key insight is that a single randomly chosen subset S can be used to estimate the marginal contribution of a player i versus all other players $j \notin S \cup i$ simultaneously, leading to significant computational savings)*

The problem uses the following Theorem about Shapley values, which implicitly gives a way to probabilistically estimate $s_i - s_j$ for any desired pair i, j.

Theorem 10.2.8. *Differences of Shapley values satisfy the following:*

$$s_i - s_j = \frac{1}{N-1} \sum_{S \subseteq [N] \setminus \{i,j\}} \frac{U(S \cup \{i\}) - U(S \cup \{j\})}{\binom{N-2}{|S|}}.$$

Proof. We start with the definition of the Shapley value s_i from Theorem 10.2.4.

$$s_i = \sum_{T \subseteq [N] \setminus \{i\}} \frac{|T|!(N - |T| - 1)!}{N!} (U(T \cup \{i\}) - U(T))$$

We can split the sum over subsets T into two cases: those that do not contain player j, and those that do. A subset $T \subseteq [N] \setminus \{i\}$ either excludes j (in which case $T \subseteq [N] \setminus \{i, j\}$) or includes j (in which case $T = S \cup \{j\}$ for some $S \subseteq [N] \setminus \{i, j\}$).

Rewriting the expression for s_i by splitting the sum:

$$s_i = \sum_{S \subseteq [N] \setminus \{i,j\}} \frac{|S|!(N - |S| - 1)!}{N!} (U(S \cup \{i\}) - U(S)) + \\ \sum_{S \subseteq [N] \setminus \{i,j\}} \frac{|S \cup \{j\}|!(N - |S \cup \{j\}| - 1)!}{N!} (U(S \cup \{i, j\}) - U(S \cup \{j\}))$$

By symmetry, the expression for s_j is:

$$s_j = \sum_{S \subseteq [N] \setminus \{i,j\}} \frac{|S|!(N - |S| - 1)!}{N!} (U(S \cup \{j\}) - U(S)) \\ + \sum_{S \subseteq [N] \setminus \{i,j\}} \frac{|S \cup \{i\}|!(N - |S \cup \{i\}| - 1)!}{N!} (U(S \cup \{i, j\}) - U(S \cup \{i\}))$$

Let $k = |S|$. Then $|S \cup \{i\}| = |S \cup \{j\}| = k + 1$. Subtracting s_j from s_i:

$$s_i - s_j = \sum_{S \subseteq [N] \setminus \{i,j\}} \frac{k!(N - k - 1)!}{N!} (U(S \cup \{i\}) - U(S \cup \{j\})) \\ + \sum_{S \subseteq [N] \setminus \{i,j\}} \frac{(k+1)!(N - k - 2)!}{N!} (U(S \cup \{i\}) - U(S \cup \{j\}))$$

where the second line uses that $\Big((U(S \cup \{i,j\}) - U(S \cup \{j\})) - (U(S \cup \{i,j\}) - U(S \cup \{i\}))\Big)$ simplifies to $(U(S \cup \{i\}) - U(S \cup \{j\}))$.
We can now combine the sums:

$$s_i - s_j = \sum_{S \subseteq [N] \setminus \{i,j\}} (U(S \cup \{i\}) - U(S \cup \{j\})) \left[\frac{k!(N-k-1)!}{N!} + \frac{(k+1)!(N-k-2)!}{N!} \right]$$

Let's focus on the term in the brackets. Factoring out common terms:

$$\frac{k!(N-k-2)!}{N!} \Big((N-k-1) + (k+1) \Big) = \frac{k!(N-k-2)!}{N!} \cdot N = \frac{k!(N-k-2)!}{(N-1)!}$$

We can rewrite this term using the binomial coefficient $\binom{N-2}{k} = \frac{(N-2)!}{k!(N-k-2)!}$:

$$\frac{k!(N-k-2)!}{(N-1)!} = \frac{1}{N-1} \cdot \frac{k!(N-k-2)!}{(N-2)!} = \frac{1}{(N-1)\binom{N-2}{k}}$$

Substituting this back into the expression for $s_i - s_j$ gives the desired result:

$$s_i - s_j = \sum_{S \subseteq [N] \setminus \{i,j\}} \frac{U(S \cup \{i\}) - U(S \cup \{j\})}{(N-1)\binom{N-2}{|S|}}$$

$\square$

10.3 Data Models

This is a method that also assigns credit to individual training datapoints, but uses a linear regression approach [12]. By training many models on subsets of p fraction of datapoints in the training set, the authors show that some interesting measures of test error (defined using logit values) behave as follows: the measure $f(x)$ is well-approximable as a (sparse) linear expression $\theta_0 + \sum_i \theta_i x_i$, where x is a binary vector denoting a sample of p fraction of training datapoints, with $x_i = 1$ indicating presence of i-th training point and $x_i = -1$ denoting absence. The coefficients θ_i are estimated via lasso regression. The surprise here is that $f(x)$ —which is the result of deep learning on dataset x—is well-approximated by $\theta_0 + \sum_i \theta_i x_i$. The authors note that the θ_i's can be viewed as heuristic estimates for the discrete influence of the ith datapoint. Note that the estimated θ_i depends on the value of p in the above procedure.

[12] Andrew Ilyas, Sung Min Park, Logan Engstrom, Guillaume Leclerc, and Aleksander Madry. Datamodels: Predicting predictions from training data. *arXiv preprint arXiv:2202.00622*, 2022

Why do data models work? [13] The reason has to do with the fact that the models are being trained on a random subset of p fraction of the training set. One can understand it using interesting but elementary harmonic analysis.

[13] Mark Braverman Sanjeev Aror Nikunj Saunshi, Arushi Gupta. Understanding influence functions and data models via harmonic analysis. *ICLR*, 2023

TO BE EXPANDED

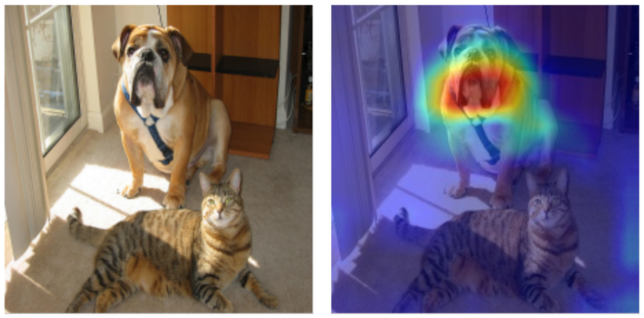

Figure 10.1: Saliency map (aka heatmap) highlighting why the image was labeled as containing a dog in it.

10.4 Interpreting Deep Net outputs on a particular input: Saliency Methods

Saliency methods try to understand the model's answer on a particular test data point in terms of the contents of that same data point. A popular way to represent importance is in the form of a heat map (also called *saliency maps*) depicting the importance of individual coordinates. For example, if the deep net is labeling images by their labels, then a saliency map for an image that has been labeled "dog" might involve highlighting the pixels that were determinative for assigning this label.

A common approach to creating a saliency map for an input x is to use the gradient of the output logit (or loss) with respect to the input pixels. The simplest method, *Vanilla Gradient*, uses the magnitude of the gradient: $S(x) = \left|\frac{\partial F(x)}{\partial x}\right|$, where $F(x)$ is the class score. However, this is just a linear approximation of the model's behavior at x and can be noisy and unreliable. More sophisticated methods attempt to go beyond this.

10.4.1 Integrated Gradients (An Axiomatic Approach)

Integrated Gradients (IG) is a popular method that satisfies two key axioms also found in Shapley values: **Sensitivity** and **Implementation Invariance**. It also satisfies **Completeness**, which states that the attributions must sum up to the difference between the model's output at the input x and its output at a baseline x' (e.g., a black image).

It is defined by integrating the gradients along a straight-line path from the baseline x' to the input x. The attribution for the i-th feature

x_i is:

$$\text{IntegratedGradients}_i(x) ::= (x_i - x'_i) \times \int_{\alpha=0}^{1} \frac{\partial F(x' + \alpha(x - x'))}{\partial x_i} d\alpha$$

IG is notable because it connects gradient-based methods back to the axiomatic foundation of Shapley values (specifically, the Aumann-Shapley value, a continuous version). However, it still fundamentally relies on a linear path integration, which may not capture complex, non-linear feature interactions that occur off this path.

10.4.2 Concept Activation Vectors (TCAV)

To move beyond pixel-level importance, Testing with Concept Activation Vectors (TCAV) asks a different question: *How important is a human-understandable concept for a given prediction?* For example, how important was the concept of "stripes" for a "zebra" classification?

The method works in three steps:

1. **Define a Concept:** A user provides a set of examples P_C for a concept C (e.g., images with stripes) and a set of random counter-examples N.

2. **Learn a Concept Vector:** These examples are mapped to an internal activation layer l of the network. A linear classifier is trained to separate the activations of P_C from N. The vector orthogonal to the separating hyperplane is the **Concept Activation Vector (CAV)**, denoted v_C^l. This vector points in the direction of increasing presence of the concept in the activation space.

3. **Measure Conceptual Sensitivity:** For an input x being explained, the **TCAV score** is the directional derivative of the class logit $h_l(f_l(x))$ with respect to the concept vector v_C^l:

$$S_{C,k,l}(x) = \lim_{\epsilon \to 0} \frac{h_l(f_l(x) + \epsilon v_C^l) - h_l(f_l(x))}{\epsilon} = \nabla h_l(f_l(x)) \cdot v_C^l$$

   where $f_l(x)$ is the activation of layer l and h_l is the function from that layer to the output logit. A positive score means the presence of the concept C increases the probability of the class prediction. This method inherently deals with non-linearities by testing the model's response to directions in activation space that represent complex, distributed concepts rather than single input features. This feels a more promising starting point for the field, given the complexities of modern deep nets.

10.4.3 *Attention Maps in Transformers: A Critical View*

For vision and language transformers, it is tempting to interpret the attention weights as a direct measure of feature importance. An attention mechanism computes the output for a token (or image patch) i as a weighted sum over other tokens j: $y_i = \sum_j \alpha_{ij} v_j$. The weights α_{ij} are often visualized as a saliency map.

However, a significant body of research argues that **attention is not explanation**. The reasons are subtle but important for a critical understanding:

- **Non-linearity and Value-mixing:** The final output depends not just on the attention weights α_{ij} but also on the value vectors v_j and subsequent non-linear transformations. A token can have high attention but be transformed to have little impact on the final output.
- **Alternative Explanations:** Studies have shown that one can construct very different attention patterns that produce nearly identical model outputs, suggesting the attention map is not a unique or faithful explanation.

This shows interpretability in modern architectures is not straightforward and requires moving beyond simplistic readings of model internals. Possibly we will never have a complete explanation.

10.5 *Limitations of Shapley Values and related notions*

While notions such as Shapley values provide an important insight into assigning importance to contributions of individual actors, it is important to understand their limitations arising from the linearity axiom. While mathematically convenient (and central to Theorem 10.2.5), the axiom can be suspect when the value of a "player" (e.g., a training data point) is highly dependent on the context provided by other players.

10.5.1 *Limitations of Shapley Values for Pricing Data*

An obvious setting for Shapley values is for pricing data for model training. We describe how in scenarios of *data synergy*, Shapley values can look mathematically "fair" but end up with conceptually misleading valuation.

The Scenario: The "Matched Pair" Imagine a binary classification task on 2D data where the true decision boundary is the line $y = x$. We are training a simple linear classifier (e.g., a linear SVM). The utility function, $U(S)$, is the final test accuracy of the model trained on a dataset S.

Consider a training set that contains many "easy" points far from the boundary, but only two "critical" points that are very close to it:

- Datapoint $d_A = (1, 1.1)$, with label B (Above the line).
- Datapoint $d_B = (1.1, 1)$, with label A (Below the line).

Data Synergy and Non-Linear Value

1. *Low Individual Value:* If we train a model on a base set of easy points S and add *only* d_A, its presence might skew the learned decision boundary far into the Class A region to classify it correctly, leading to poor generalization. Its marginal contribution to accuracy could even be negative. The same is true for d_B alone.
$$U(S \cup \{d_A\}) - U(S) \approx \text{low or negative}$$
2. *High Synergy Value:* If we train a model on the set S plus *both* d_A and d_B, this matched pair provides a powerful constraint. They work together to pin the decision boundary precisely where it should be, leading to a large increase in accuracy. The marginal contribution of adding d_A to a set that already contains d_B is therefore very high.

$$U(S \cup \{d_B\} \cup \{d_A\}) - U(S \cup \{d_B\}) \approx \text{very high}$$

The Misleading Shapley Value The Shapley value for d_A is the average of its marginal contributions over all possible subsets of other data points. Roughly half of these subsets will contain its synergistic partner d_B, and half will not. The calculation will average the very high contribution (when d_B is present) with the very low contribution (when d_B is absent).

The result will be a **moderate Shapley value** for both d_A and d_B.

Discussion The moderate value is mathematically correct according to the axioms but provides a flawed narrative. A data curator might look at the value and conclude, "This is a reasonably useful data point." They might decide to acquire d_A but not d_B, believing they have acquired moderate value. In reality, they have acquired a low-value, perhaps even harmful, data point.

The true story is not in the data's average value, but in its *conditional value*: the value of d_A is extremely high *given the presence of* d_B, and low otherwise. This non-linear synergy is precisely what the Shapley value's implicit linearity assumption averages away, failing to capture the most important aspect of the data's contribution.

10.5.2 *Limitations of Shapley values for pricing datasets*

The reader may find the above example very toy, since it hinges on subtle relationships between two datapoints, which are only two out of billions (or trillions!) of datapoints that an AI model may be trained on.

However, these days LLM training involves assembling the dataset from multiple sources. A similar issue arises in pricing of datasets from different sources; the linearity axiom central to Shapley values does not hold since there can be synergies (positive and negative) between different types of data.

Problem 10.5.1 (Pricing Datasets with Synergies)**.** *You are a data strategist at an AI company building a next-generation Large Language Model. You have the option to purchase or license up to three large datasets:*

- ***Dataset A:*** *A massive corpus of General Web Text.*
- ***Dataset B:*** *A curated collection of University-level Mathematical Papers.*
- ***Dataset C:*** *A large repository of permissively licensed source Code.*

After extensive experimentation, your team has determined the final performance score (out of 100) of a model trained on any combination of these datasets. The utility function $U(S)$ is defined as follows, where S is a subset of $\{A, B, C\}$:

Dataset Combination S	***Performance U(S)***
$\emptyset$ *(baseline model)*	*0*
$\{A\}$	*50*
$\{B\}$	*20*
$\{C\}$	*25*
$\{A, B\}$	*75*
$\{A, C\}$	*80*
$\{B, C\}$	*90*
$\{A, B, C\}$	*100*

Your goal is to use concepts from cooperative game theory to value these datasets.

1. ***Contextual Value:*** *What is the marginal performance gain of adding Dataset C to a model already trained on A? What is its marginal gain when added to a model trained on B? What does this tell you about data synergy?*

2. ***Shapley Value Calculation:*** *Calculate the Shapley value (s_A, s_B, s_C) for each of the three datasets.*

3. ***Strategic Decision Making:***

- *If the company could only afford to license* ***one*** *dataset, which one provides the most value?*
- *Suppose the company has a budget for exactly* ***two*** *datasets. A junior analyst looks at the individual performance scores ($U(\{A\}), U(\{B\}), U(\{C\})$) and recommends purchasing A and C. What is the performance of this pair?*
- *What is the* ***optimal*** *pair of datasets to achieve the highest performance?*
- *Discuss the discrepancy. How might the Shapley values, which represent a "fair" average contribution, be a misleading guide for making this specific two-dataset purchase decision?*

10.5.3 Limitations of Shapley Value in interpreting deep net outputs

A vast field of interpretability research tries to understand why a deep net outputs a particular answer on a particular input. For example, why did the deep net label an image as "cat" and not "dog"? Section 10.4 discusses the obvious idea, which is to define "saliency values" for either individual image pixels, or for $k \times k$ collections of neighboring pixels. This can be problematic when the decision does not satisfy the linearity axiom, as is often the case in deep learning.

Problem 10.5.2 (Explaining a "Corner Detector" Model)**.** *This exercise uses a toy model to make the non-linearity issue of neural networks concrete and easy to analyze. The goal is to see how the Shapley axioms produce a mathematically "fair" but potentially misleading explanation for a model with strong feature interactions.*

Consider a simple 2x2 binary image with four pixels: Top-Left (P_{TL}), Top-Right (P_{TR}), Bottom-Left (P_{BL}), and Bottom-Right (P_{BR}).

A simple "corner detection" model is defined as follows. It outputs a high score (logit) if and only if three specific pixels are "on" (value 1): P_{TL}, P_{TR}, and P_{BL}.

The utility function $U(S)$ for a set of pixels S is:

$$U(S) = \begin{cases} 1 & \text{if } \{P_{TL}, P_{TR}, P_{BL}\} \subseteq S, \\ 0 & \text{otherwise.} \end{cases}$$

The image we want to explain is $\{P_{TL} = 1, P_{TR} = 1, P_{BL} = 1, P_{BR} = 0\}$. For this image, the total utility is $U(\{P_{TL}, P_{TR}, P_{BL}\}) = 1$. We assume the baseline (empty set) utility $U(\emptyset) = 0$.

1. *Calculate the Shapley values for all four pixels: s_{TL}, s_{TR}, s_{BL}, and s_{BR}.*
2. *Analyze the result:*

- *How does the model distribute the credit for the prediction?*

- *Does the Null Player axiom hold for the P_{BR} pixel?*
- *Do the Symmetry and Efficiency axioms hold? How do they help you find the solution?*
- *Discuss whether the resulting Shapley values provide a "true" explanation of how this AND-gate-like model works. What insight do the values provide, and what do they obscure about the model's logic?*