6.5. Anchors

Notation and first definitions

- **Back to text:** ξ = document to explain = ordered sequence of tokens $(\xi_1, \ldots, \xi_T)$, f = classifier

Definition: we define an *anchor* A as an ordered subset of the words of ξ. We let $\mathcal{A}$ be the set of all possible *non-empty* anchors.

- two key definitions:
  1. *precision* = probability of same classification knowing that the document contains A
  2. *coverage* = how many documents in the dataset contain A
- one-sentence summary: find anchor with prescribed precision and maximal coverage

```
The selection on the menu is great, and so is the food! The service is not bad, prices are fine.
```

$\Longrightarrow$

$$\mathrm{Prec}(A) = 0.97$$
$$\mathrm{Cov}(A) = 0.12$$

How precision is computed

- **Formal definition:**

$$\mathrm{Prec}(A) := \mathbb{P}_A\left(f(X) = f(\xi)\right) ,$$

where X is a random perturbation of ξ containing all words in A

- **Question:** what is the distribution of "X given A" in this definition?
  - **default implementation:** i.i.d. Bernoulli for each word not in A to decide removal, replace by `UNK` token if removed (more on that later)
  - **generative model:** for instance, using BERT[49] to generate the missing words,...
  - **deterministic replacements:** get word embedding and replace by word having similar embeddings,[50]...

---

[49] Devlin, Chang, Lee, Toutanova, *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*, Proc. ACL, 2019

[50] Ribeiro, Singh, Guestrin, *"Why should I trust you?" Explaining the prediction of any classifier*, ACM SIGKDD, 2016

Sampling mechanism

The selection on the menu is great, and so is the food! The service is not bad, prices are fine.

the selection on the menu is great and so is the food the service is not bad prices are fine

the selection on the menu is great and so is the food the service is not bad prices are fine

the selection on the menu is great and so is the food the service is not bad prices are fine

the selection UNKthe menu is great and so is the food the UNK is not bad prices UNK fine

Sampling mechanism

The selection on the menu is great, and so is the food! The service is not bad, prices are fine.

the selection on the menu is great and so is the food the service is not bad prices are fine

the selection on the menu is great and so is the food the service is not bad prices are fine

the selection on the menu is great and so is the food the service is not bad prices are fine

the selection on the menu is great and so is UNK food the UNK is not bad prices are fine

Sampling mechanism

The selection on the menu is great, and so is the food! The service is not bad, prices are fine.

the selection on the menu is great and so is the food the service is not bad prices are fine

the selection on the menu is great and so is the food the service is not bad prices are fine

the selection on the menu is great and so is the food the service is not bad prices are fine

UNK selection on the menu UNKgreat and UNKUNKthe UNK the UNK UNKnot bad UNK are fine

Estimating $\mathrm{Prec}(A)$

- wlog, one can assume that $f(\xi) = 1$
- thus

$$\mathrm{Prec}(A) := \mathbb{P}_A\left(f(X) = 1\right) .$$

- **Remark:** of course, impossible to compute in practice (too costly with `UNK` replacement, worse with BERT)
- **Solution:** Monte-Carlo estimate:

$$\widehat{\mathrm{Prec}}_n(A) := \frac{1}{n}\sum_{i=1}^{n} \mathbb{1}_{f(X_i)=1} ,$$

  where X_i i.i.d. draw from X
- in practice, $n = 10$

Coverage

- **Formal definition:** let $\mathcal{C}$ be a given set of documents. For any anchor A, we define

$$\mathrm{Cov}(A) := \left|\{\delta \in \mathcal{C} \quad \text{s.t.} \quad \forall w \in A, w \in \delta\}\right| \, .$$

- **Remark:** in practice, shorter anchors have higher coverage
- **Why?** think one common word: contain in many documents
- in the other direction, whole sentence $\to$ only contained in one document
- since $\mathrm{Cov}(A)$ costly to compute, Anchors minimizes $|A|$ instead of maximizing $\mathrm{Cov}(A)$

Summary

- let $\varepsilon > 0$ be some tolerance threshold (by default, $\varepsilon = 0.05$)
- **What is described originally:**

$$\operatorname*{Maximize}_{A \in \mathcal{A}} \mathrm{Cov}(A) \quad \text{subject to} \quad \mathrm{Prec}(A) \geq 1 - \varepsilon\,.$$

- **What the actual goal is:**

$$\operatorname*{Minimize}_{A \in \mathcal{A}} |A| \quad \text{subject to} \quad \widehat{\mathrm{Prec}}_n(A) \geq 1 - \varepsilon\,. \qquad (\star)$$

- **Additional caveat:** if ξ has length b, $|\mathcal{A}| = 2^b$...
- **What is done in practice:** use KL-UCB[51] to approximately solve $(\star)$

[51] Kaufmann and Kalyanakrishnan, *Information complexity in bandit subset selection*, COLT, 2013

Visualizing (⋆)

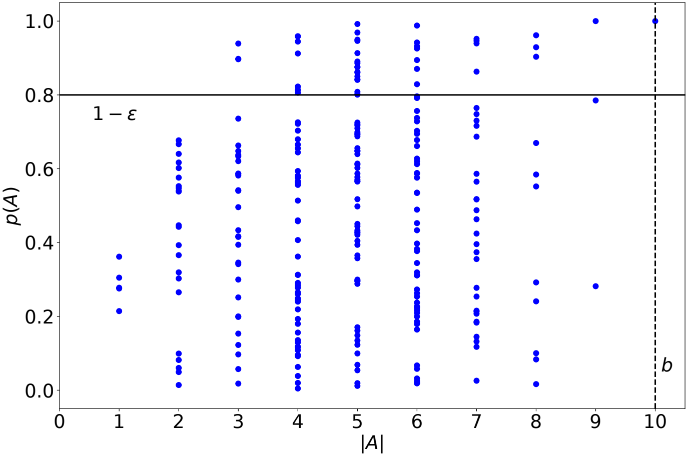


Figure: all anchors for a given example / classifier represented in the $|A|$ / $p(A) = \mathrm{Prec}(A)$ space

Visualizing $(\star)$

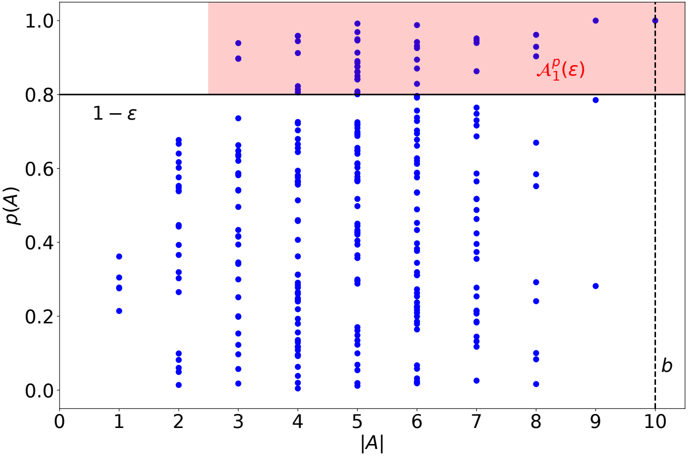


Figure: selecting $\mathcal{A}_1^p(\varepsilon)$, set of all anchors with evaluation higher than $1 - \varepsilon$

Visualizing $(\star)$

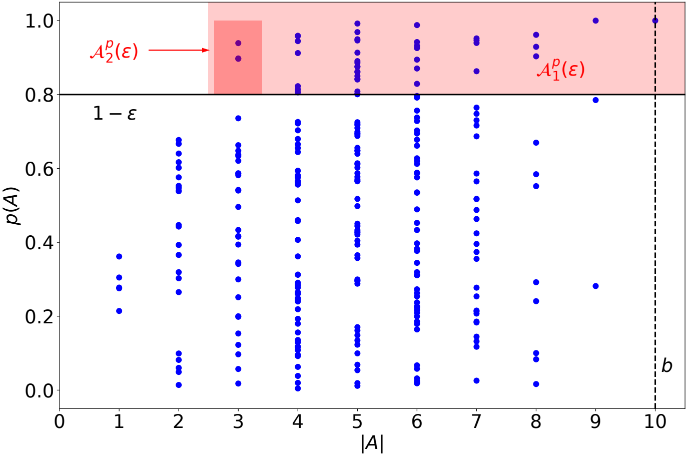


Figure: selecting $\mathcal{A}_2^p(\varepsilon)$, anchors with $p(A) \geq 1-\varepsilon$ and minimal length

Visualizing $(\star)$

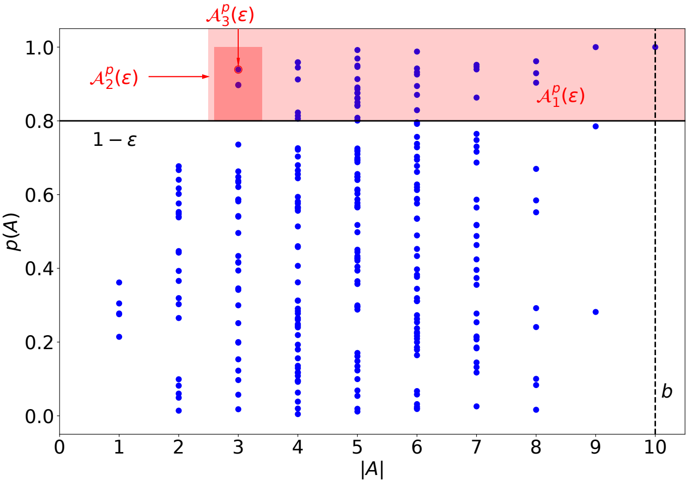


Figure: selecting $\mathcal{A}_3^p(\varepsilon)$, anchors with $p(A) \geq 1-\varepsilon$, minimal length, and maximal $p(A)$

Summary

- **Anchors** = rule selection via random perturbation
- interpretable features = subset of the words
- post-hoc, local method
- local rules have a global flavor
- very costly to run
- version for other data-types exist[52]
- some theoretical analysis (indicator and linear models)[53]

[52] `https://github.com/marcotcr/anchor`
[53] Lopardo, Precioso, Garreau, *A sea of words: an in-depth analysis of Anchors for text data*, AISTATS, 2023

6.6. A game-theoretical perspective

Shapley values

- **Setting:** D-player game[54]
- characteristic function $v : 2^D \to \mathbb{R}$, gives the *value* of a coalition S
- total sum of gains the members of S can obtain by cooperation
- **Idea:** distribute fairly the total gains to the players, assuming that they all contribute

Definition: Shapley value of player j:

$$\phi_j(v) = \sum_{S \subseteq [D] \setminus \{j\}} \frac{|S|!(D - |S| - 1)!}{D!} \left(v(S \cup \{j\}) - v(S) \right) .$$

- **Intuition:** if player j plays much better than the others, then $v(S \cup \{j\})$ consistently higher than $v(S)$, and $\phi_j(v) \gg 0$

[54] Shapley, *A value for n-person game*, Contributions to the theory of games, 1953

Shapley values: an example

- **Example:** $D = 2,\ v(1) = 1,\ v(2) = 2,\ v(12) = 4$
- fruitful collaboration $(4 > 3)$, how to split?
- since 2 brings more, $50 - 50$ split is arguably not "fair"
- let us compute the Shapley values:

$$\begin{aligned}\phi_1 &= \frac{0!(2-0-1)!}{2!}(v(1) - v(\emptyset)) + \frac{1!(2-1-1)!}{2!}(v(12) - v(2)) \\ &= \frac{1}{2} \cdot 1 + \frac{1}{2} \cdot 2 \\ &= 1.5\end{aligned}$$

- similarly, we obtain $\phi_2 = 2.5$

Properties

- Shapley values have nice theoretical properties:
  - *efficiency:* sum of Shapley values = gain of the whole coalition:

$$\sum_j \phi_j(v) = v([D])\,.$$

  - *symmetry:* players with the same skills are rewarded equally:

$$\forall S \subseteq [D],\ v(S \cup \{j\}) = v(S \cup \{k\}) \quad \Rightarrow \quad \phi_j(v) = \phi_k(v)\,.$$

  - *linearity:* v and w two characteristic functions, then

$$\forall j \in [D], \quad \phi_j(v + w) = \phi_j(v) + \phi_j(w)\,.$$

  - *null player:* a player that does not bring anything is not rewarded:

$$\forall j \in [D], \quad v(S \cup \{j\}) = v(S) \quad \Rightarrow \quad \phi_j(v) = 0\,.$$

Shapley values, ctd.

- other nice properties:
  - *anonymity*
  - *standalone test*
  - ...
- more interestingly:

Theorem:[55] Shapley values are the only payment rule satisfying efficiency, symmetry, linearity, and null player.

- **Question:** connection with interpretability?
- we can see f as the reward and a subset of features as the player

[55] *ibid*

Shapley regression values

- **Example:** linear model
- for each subset of features $S \subseteq [D]$, retrain a model f_S only using the features in S

Definition:[56] the *Shapley regression value* associated to feature j is given by

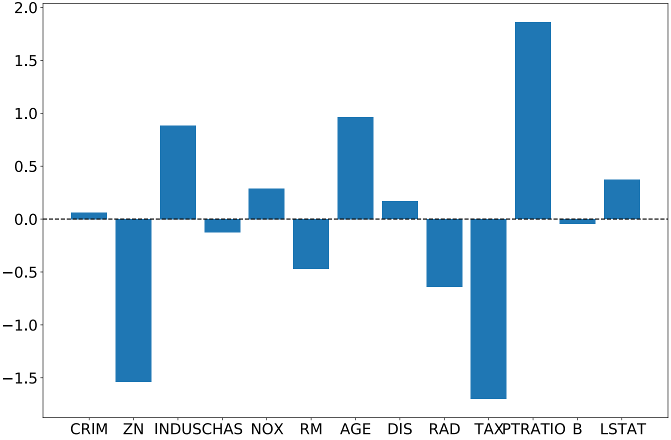
$$\phi_j := \sum_{S \subseteq [D] \setminus \{j\}} \frac{|S|!(D - |S| - 1)!}{D!} \left(f_{S \cup \{j\}}(\xi_{S \cup \{j\}}) - f_S(\xi_S) \right) ,$$

where ξ_S is the restriction of ξ to S features.

[56] Lipovetsky and Conklin, *Analysis of regression in game theory approach*, Applied Stochastic Models in business and industry, 2001

Shapley regression values

- **Example:** output for linear regressor on Boston housing dataset



Shapley sampling values

- several problems with this approach:
  - *computational cost* $= \mathcal{O}\left(2^D\right)$
  - *retraining* the model each time
- a first solution: *Shapley sampling values*[57]
  - subsample in the sum over all subsets
  - instead of retraining the model, mimic the removal a variables by randomly sampling over the training set
- in other words, replace $f_S(\xi_S)$ by

$$\mathbb{E}\left[f(x) \mid x_S = \xi_S\right] .$$

- f can now be any model, provided we can estimate this last quantity efficiently

[57] Štrumbelj and Kononenko, *Explaining models and individual predictions with feature contributions*, Knowledge and information systems, 2014

Kernel SHAP

- still very costly to test *all the coalitions*
- **Idea:** linear regression on the presence / absence of features
- as before, define interpretable features $z \in \{0,1\}^d$, with $d \leq D$
- $h_\xi : \{0,1\}^d \to \mathbb{R}^D$ mapping function such that $h_\xi(\mathbf{1}) = \xi$
- **Example:** $d = D$, $h_\xi(z) = \mathbb{E}\left[f(X) \mid X_S = \xi_S\right]$, where S encoded by z

Definition (kernel SHAP)[58]: define ϕ as the minimizer of

$$\sum_{z \in \{0,1\}^d} \frac{d-1}{\binom{d}{|z|} \cdot |z| \cdot (d - |z|)} \left(f(h_\xi(z)) - \phi^\top z\right)^2 .$$

[58] Lundberg and Lee, *A Unified Approach to Interpreting Model Predictions*, NeurIPS, 2017

Kernel SHAP

- **In practice:** $z_1, \ldots, z_n$ i.i.d. Bernoulli $\in \{0,1\}^d$ and minimize for $\phi \in \mathbb{R}^d$

$$\sum_{i=1}^{n} \pi_i \cdot \left(f(h_\xi^{-1}(z_i)) - \phi^\top z_i \right)^2 ,$$

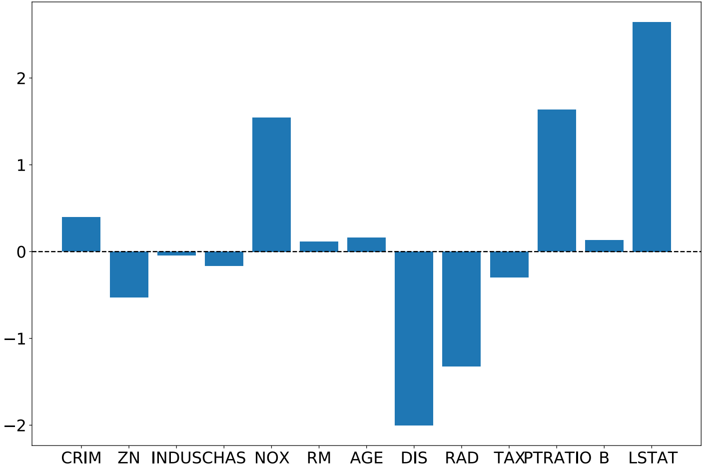
  with

$$\pi_i := \frac{d-1}{\binom{d}{|z_i|} \cdot |z_i| \cdot (d - |z_i|)} .$$

- weighted linear regression to approximate Shapley values
- computational cost: $\mathcal{O}\left(2^d + d^3\right)$
- **Remark:** very similar to LIME

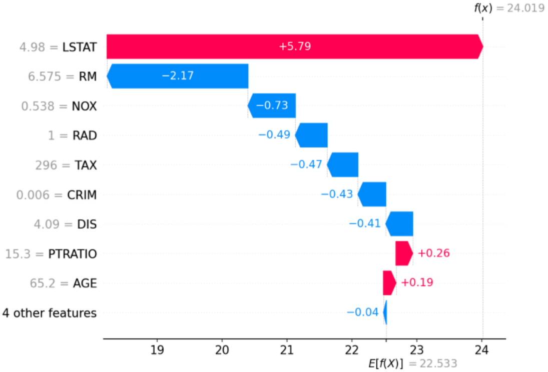
Kernel SHAP, tabular example

- **Example:** interpreting a linear model on the Boston dataset:



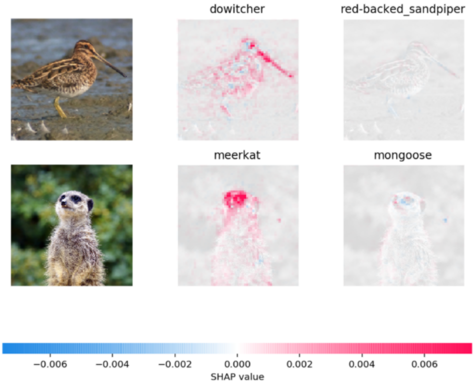
Kernel SHAP, tabular example

- we can also use the `shap` Python package
- really nice visualizations:



Extensions

- Kernel SHAP is not restricted to tabular data
- **Example:** explaining the predictions of VGG16 for two classes



Summary

- Kernel SHAP can be used on any model
- specialized versions for specific architectures:
  - *TreeSHAP*[59] (tree-based predictors)
  - *DeepSHAP* (DeepLIFT[60] + Shapley values)

Inconvenients:

- costly to run[61]
- not easy to read if many features

---

[59] Lundberg et al., *Consistent individualized feature attribution for tree ensembles*, arxiv, 2018
[60] Shrikumar et al., *Learning important features through propagating activation differences*, ICML, 2017
[61] improving the efficiency is work in progress, *e.g.*, Covert and Lee, *Improving KernelSHAP: Practical Shapley Value Estimation via Linear Regression*, AISTATS, 2021

7. Gradient-based approaches

7.1. Model agnostic methods

Introduction

- **General idea:** machine learning model = complicated function of the inputs
- approximate this function by a first order approximation

Theorem (Taylor, order one): let f be differentiable in the neighborhood of $\xi \in \mathbb{R}^D$. Then

$$f(x) = f(\xi) + \nabla f(\xi)^\top (x - \xi) + o\left(\|x - \xi\|\right) ,$$

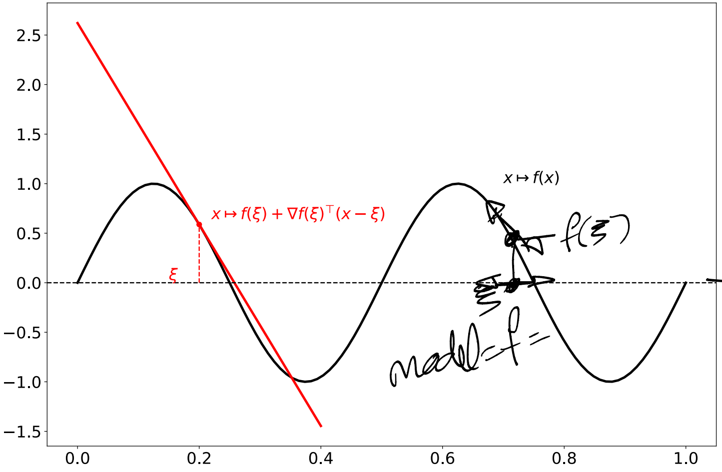
where $\nabla f(\xi)$ is the *gradient* of f at ξ.

- **Reminder:** gradient = vector of $\mathbb{R}^D$ defined by

$$\forall 1 \leq j \leq D, \qquad (\nabla f(\xi))_j = \left.\frac{\partial f(x)}{\partial x_j}\right|_{x=\xi} .$$

In low dimension

- **Example:** linear approximation in dimension 1:



Gradient explanation

- **Simple idea:** take the gradient of the function to explain at the point of interest:

$$\phi_j = (\nabla f(\xi))_j = \frac{\partial}{\partial x_j} f(x)\bigg|_{x=\xi} .$$

- generally referred to as *gradient explanation*[62], *sensivity map*, or *saliency map*
- **Intuition:** tells us how much a change in each input dimension would change the the prediction *in a small neighborhood around* ξ
- computational cost $= \mathcal{O}(1)$ if the model is "PyTorch-compatible"
- **Remark:** in this talk, one evaluation of the model costs $\mathcal{O}(1)$
- **Beware:** gradient with respect to the input, not the parameters!

[62] Baehrens et al., *How to Explain Individual Classification Decisions*, JMLR, 2010

Linear model

- **Usual question:** what happens for a linear model?
- that is, $f(x) := \sum_{k=1}^{d} \lambda_k x_k$
- in that case, simple answer:

$$
\begin{aligned}
\frac{\partial f(x)}{\partial x_j} &= \frac{\partial}{\partial x_j} \sum_{k=1}^{d} \lambda_k x_k \\
&= \sum_{k=1}^{d} \frac{\partial}{\partial x_j} \lambda_k x_k \\
\frac{\partial f(x)}{\partial x_j} &= \lambda_j \, .
\end{aligned}
$$

- we retrieve the coefficients of the model:

$$
\phi_j = \lambda_j \, .
$$

Gradient explanation for images

- in this context, particular instance of *saliency maps*[63]
- input variables = pixel values
- each pixel has typically 3 channels
- when taking gradient with respect to input, pixel $\xi_{(i,j)}$ is attributed

$$(\nabla f(\xi))_{(i,j)} = \left(\frac{\partial f}{\partial x_{(i,j),\text{red}}}(\xi), \frac{\partial f}{\partial x_{(i,j),\text{green}}}(\xi), \frac{\partial f}{\partial x_{(i,j),\text{blue}}}(\xi) \right)^\top \in \mathbb{R}^3 .$$

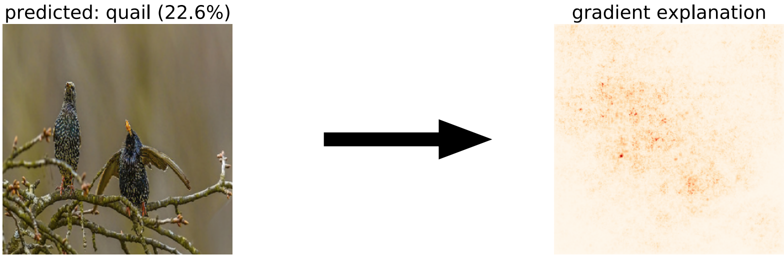
- **Question:** how to visualize this as a heatmap?
- typical to display the maximum

$$\max \left(\left| \frac{\partial f}{\partial x_{(i,j),\text{red}}}(\xi) \right| , \left| \frac{\partial f}{\partial x_{(i,j),\text{green}}}(\xi) \right| , \left| \frac{\partial f}{\partial x_{(i,j),\text{blue}}}(\xi) \right| \right) .$$

[63] Simonyan, Vedaldi, Zisserman, *Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps*, preprint, 2013

Gradient explanation for images

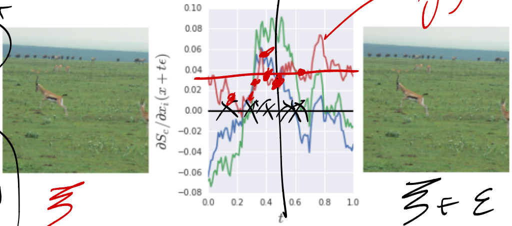
- **Example:** image classification on ILSVRC with InceptionV3:



- quite *noisy*, but we can see that the network is using the relevant part of the image
- also possible to look at *positive* and *negative* influence

Issues with the gradient

- **Main issue** with score functions of deep neural net: can be quite irregular at small scale
- **Example:** gradient fluctuations between similar images[64]



- ⇒ purely gradient-based explanations can be quite noisy

[64] Smilkov et al., *SmoothGrad: removing noise by adding noise*, arxiv, 2017

Gradient times input

- many works trying to address this issue
- **First idea:** gradient times input[65]
- namely, partial derivatives multiplied by feature values

$$\phi_j = (\xi \odot \nabla f(\xi))_j = \xi_j \cdot \frac{\partial f(\xi)}{\partial x_j} \,.$$

(black & white)

$\rightarrow \phi_{i,j} = \xi_{i,j} \cdot \frac{\partial f}{\partial x_{i,j}}(\xi)$

colour

$\phi^{\Omega}_{i,j} = \xi^{\Omega}_{i,j} \cdot \frac{\partial f}{\partial x^{\Omega}_{i,j}}(\xi)$

- **Intuition:** smoother explanations
- computational cost $= \mathcal{O}(D)$
- for a linear model:

$$f(x) = \sum_{j=1}^{D} \lambda_j x_j + b \quad \Rightarrow \quad \phi_j = \lambda_j \xi_j \,.$$

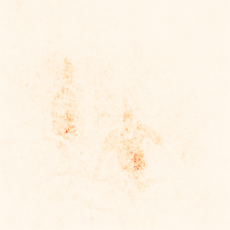
[65] Shrikumar et al., *Not just a black box: Learning important features through propagating activation differences*, ICML, 2016

Gradient times input for images

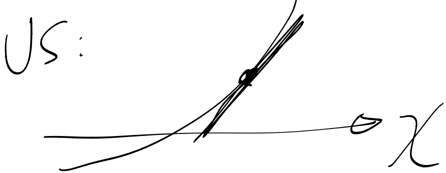
- **Example:** image classification on ILSVRC with InceptionV3

predicted: quail (22.6%)

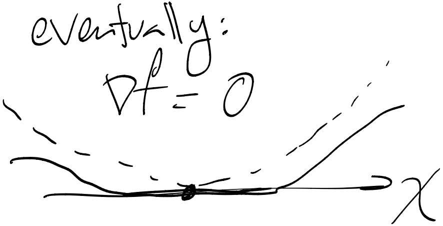
gradient times input



- much *smoother*, as promised
- still difficult to read in some cases



SmoothGrad



- another effort in this direction: *SmoothGrad*[66]
- **Idea:** average local fluctuations
- $\rightarrow$ sample many gradients in the surrounding and take the mean
- namely,

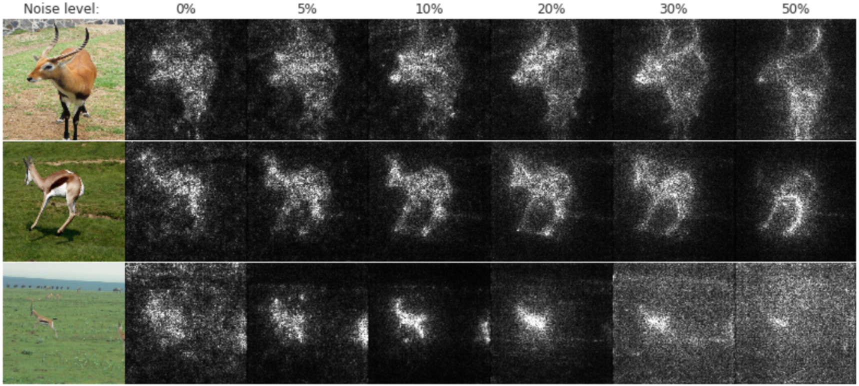
$$\forall j \in [D], \qquad \phi_j := \frac{1}{n} \sum_{i=1}^{n} (\nabla f(\xi + \varepsilon_i))_j$$

where ε_i i.i.d. $\mathcal{N}\left(0, \sigma^2 \, \mathrm{I}_d\right)$

- Smilkov recommends taking σ as $10 - 20\%$ of input range (see next slide)
- **Intuition:** adding noise is equivalent to regularization[67]

[66] Smilkov et al., *SmoothGrad: removing noise by adding noise*, arxiv, 2017
[67] Bishop, *Training with noise is equivalent to Tikhonov regularization*, Neural Computation, 1995

SmoothGrad, influence of σ



- **Figure:** SmoothGrad with σ as percentage of the range

SmoothGrad, behavior for large n

- when $n \to +\infty$, according to SLLN:

$$\phi_j \xrightarrow{\text{a.s.}} \mathbb{E}\left[\frac{\partial f}{\partial x_j}(\xi + \varepsilon)\right] ,$$

where $\varepsilon \sim \mathcal{N}\left(0, \sigma^2 \mathrm{I}\right)$

- set $g := \frac{\partial f}{\partial x_j}$
- let us assume that f is "nice" around ξ:

$$g(\xi + \varepsilon) \approx g(\xi) + \varepsilon^\top \nabla g(\xi)$$

- taking expectation in the previous display yields

$$\mathbb{E}\left[g(\xi + \varepsilon)\right] \approx g(\xi) + \mathbb{E}\left[\varepsilon^\top \nabla g(\xi)\right] \approx g(\xi) = \frac{\partial f}{\partial x_j}(\xi) .$$

- we recover (approximately) the gradient

Integrated gradients

- **Another idea:**[68] average gradients between given reference and point of interest
- formally, if ξ_0 is a reference image,

$$\phi = (\xi - \xi_0) \odot \int_0^1 \frac{\partial f}{\partial x}(\xi_0 + \alpha(\xi - \xi_0))\mathrm{d}\alpha\,.$$

- of course, we have no way to compute the previous integral
- Monte-Carlo approximation:

$$\int_0^1 \frac{\partial f}{\partial x}(\xi_0 + \alpha(\xi - \xi_0))\mathrm{d}\alpha \approx \frac{1}{m}\sum_{i=1}^{m} \frac{\partial f}{\partial x}\left(\xi_0 + \frac{i}{m}(\xi - \xi_0)\right)\,.$$

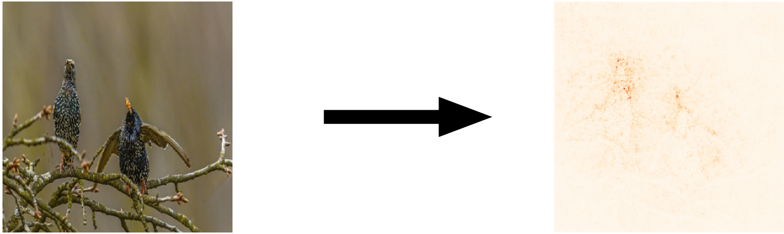
- computational cost $= \mathcal{O}(mD)$ ($m = 20$ gives good results)

[68] Sundararajan et al., *Axiomatic attribution for deep networks*, ICML 2017

Integrated gradients, ctd.

- **Example:** image classif. on ILSVRC with InceptionV3, reference image = 0

predicted: quail (22.6%) integrated gradients



- usually less “visual diffusion”
- main critic: similar to an *edge detector*[69]

[69] Adebayo et al., *Sanity checks for saliency maps*, NeurIPS 2018

Integrated gradients meets linear model

$$\frac{\partial f}{\partial x_j} = \frac{\partial}{\partial x_j}\left[\lambda_1 x_1 + \cdots + \lambda_{j-1} x_{j-1} + \lambda_j x_j + \lambda_{j+1} x_{j+1} + \cdots + \lambda_d x_d\right] = \lambda_j$$

- **Question:** what happens for a linear model?
- recall: $f(x) = \sum_{k=1}^{d} \lambda_k x_k$
- in that case:

$$\phi_j = (\xi_j - \xi_{0,j}) \cdot \int_0^1 \frac{\partial f}{\partial x_j}(\xi_0 + \alpha(\xi - \xi_0))\mathrm{d}\alpha$$

$$= (\xi_j - \xi_{0,j}) \cdot \int_0^1 \frac{\partial}{\partial x_j}(x \mapsto \lambda_j(\xi_{0,j} + \alpha(x_j - \xi_{0,j})))\mathrm{d}\alpha$$

$$= (\xi_j - \xi_{0,j}) \cdot \int_0^1 \lambda_j \alpha \mathrm{d}\alpha$$

$$\phi_j = \frac{1}{2}(\xi_j - \xi_{0,j})\lambda_j \,.$$

- up to constants, we recover gradient $\times$ input

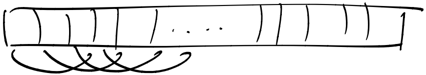
Summary

- if model is "smooth," natural idea = gradient with respect to the input
- gradient has somewhat irregular behavior
- to improve visualization, marginal modifications:
  - gradient times input
  - SmoothGrad
  - integrated gradient

7.2. Model-specific methods

Introduction

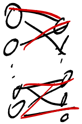
- so far, we have looked at generic black-box models
- in this section we look at gradient-based approaches taking advantage of model specificity
- focus on **convolutional neural networks**[70] (CNN)
- mostly for images, but used in other applications[71]



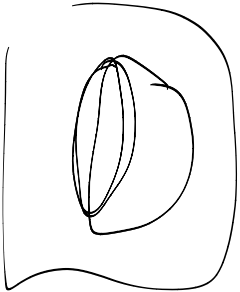
[70]Fukushima, *Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position*, Biological Cybernetics, 1980

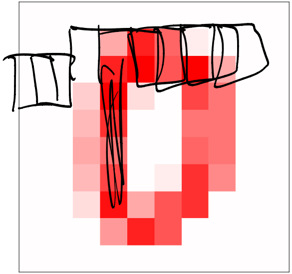
[71]Kim, *Convolutional Neural Networks for Sentence Classification*, EMNLP, 2014

Convolutional layers



- convolutional neural network[72] (CNN) = neural network with a convolutional layer
- **Question:** what is this?
- **Idea:** take the structure of the data (*i.e.*, images) into account
- in particular, invariant by (small) translations





0	0	0	0	0	0	0	0	0	0
0	0	0	5	13	9	1	0	0	0
0	0	0	13	15	10	15	5	0	0
0	0	3	15	2	0	11	8	0	0
0	0	4	12	0	0	8	8	0	0
0	0	5	8	0	0	9	8	0	0
0	0	4	11	0	1	12	7	0	0
0	0	2	14	5	10	12	0	0	0
0	0	0	6	13	10	0	0	0	0
0	0	0	0	0	0	0	0	0	0

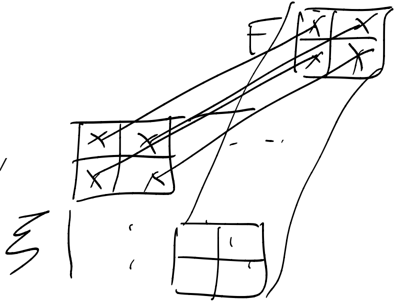
[72] Fukushima, *Neocognitron: A Self-organizing Neural Network Modelf or a Mechanism of Pattern Recognition Unaffected by Shift in Position*, Biological Cybernetics, 1980

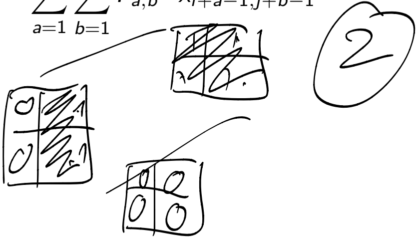
Filters

- **Idea:** apply a *filter* $F \in \mathbb{R}^{k \times k}$ on the image $x \in \mathbb{R}^{H \times W}$
- same idea than in other areas of mathematics:
  1. translate the filter
  2. multiply termwise
- in equation:

$$\forall i, j \in [H - k + 1] \times [W - k + 1], \qquad c_{i,j} = \sum_{a=1}^{k} \sum_{b=1}^{k} F_{a,b} \cdot x_{i+a-1, j+b-1}$$

- filters are square matrices with small size
- kernel size k = number of pixels being processed
- **Typical values:** 2×2, 16×16





Visualization

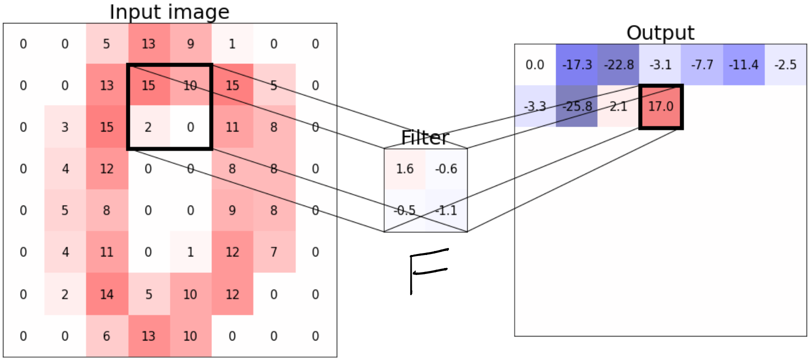
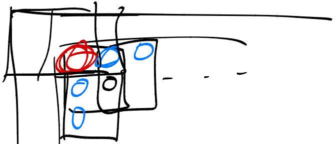


Figure: sliding the filter ($k = 2$ in this example) over the input image to compute the convolution

Padding



- **Remark:** pixels at the border of the image are counted only once
- to avoid this, one can add *padding*
- that is, p pixels with constant values added on the border
- in equation: define $\tilde{x} \in \mathbb{R}^{(H+2p)\times(W+2p)}$ such that

$$\forall i,j \in [(p+1):(H+p)] \times [(p+1):(W+p)], \qquad \tilde{x}_{i,j} = x_{i-p,j-p}$$

  and $x_{i,j} = 0$ otherwise
- then convolution on $\tilde{x}$ instead of x
- **Remark:** other colors can be chosen depending on the context
- canonical choice for $p =$ kernel dimension minus one

Visualization

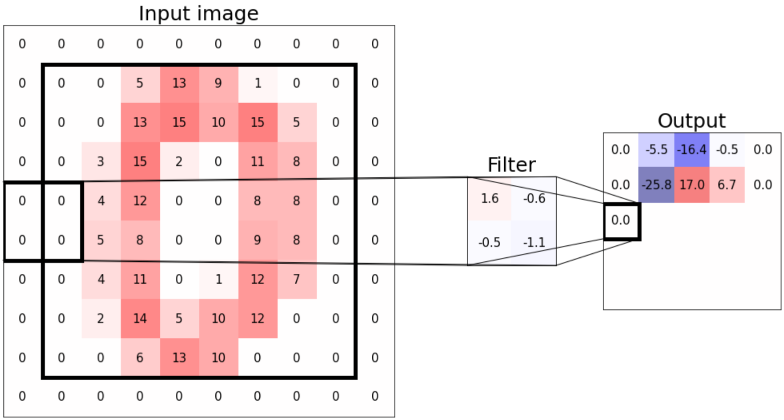
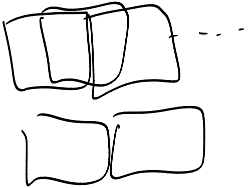
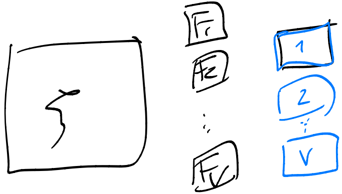


Figure: sliding the filter over the padded input image ($p = 1$ in this example) to compute the convolution

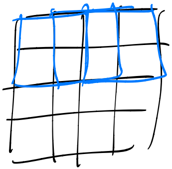


Stride

- **Additional idea:** redundant information, skip a few pixels each time
- instead of moving one pixel to the right / down, move from s pixels
- s is called the *stride*
- **Typical value:** $s = k$
- **Important:** the size of the output matrix changes
- see `https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html` for the exact shape used by Pytorch

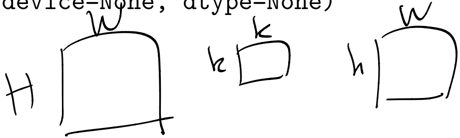


Other important details



- usually, images have *several channels* (RGB $\to c = 3$)
- each filter has one coefficient matrix per channel (real size is $k \times k \times c$)
- sum the outputs of each channel convolution, sometime add a *bias*
- also common to have several filters (say 64)
- this is still called a convolutional layer
- in Pytorch:

```
torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1,
padding=0, dilation=1, groups=1, bias=True, padding_mode='zeros',
device=None, dtype=None)
```

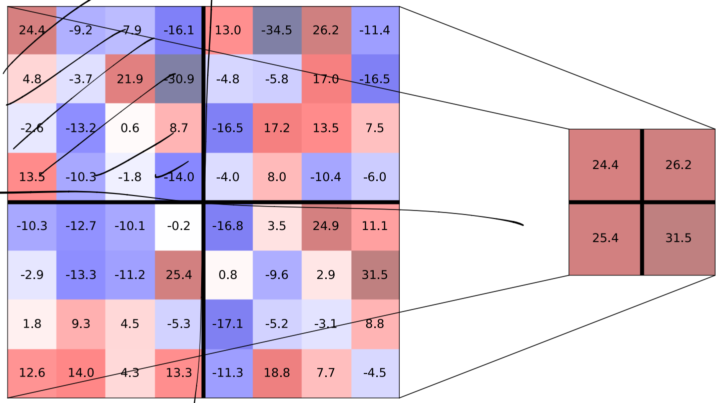


$$\begin{cases} h = H - k + 1 \\ w = W - k + 1 \end{cases}$$

$$3 = 4 - 2 + 1$$

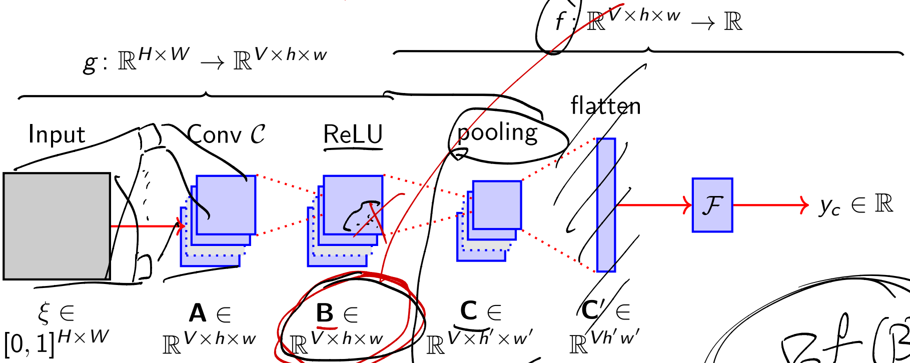
Pooling

- **Idea:** get rid of the noise
- → take the max / mean on distinct parts of the image
- usually used after a convolution



Setting

- many methods rely on the **last convolutional layer** (here **B**)

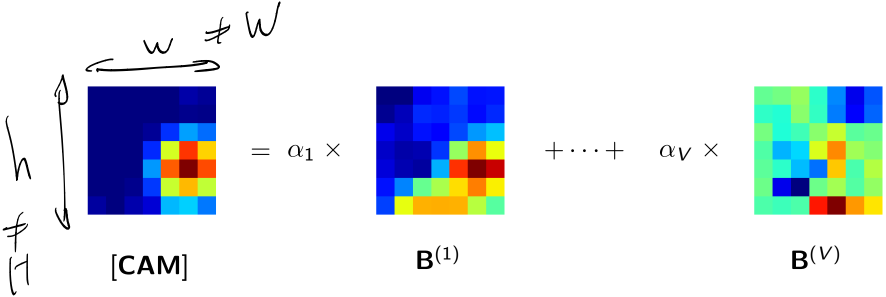


Class Activation Map (CAM)

- **Class Activation Map (CAM):**[73] specific architecture
- GAP of each convolutional feature map
- in our notation, $h' = w' = 1$
- for a specific class, coefficients of the last linear layer can be written $\alpha_1, \ldots, \alpha_V$
- **Idea:** weighted sum of the activations:

$$y_c = \sum_{v=1}^{V} \alpha_v \left(\mathrm{GAP}(\mathbf{B}^{(v)}) \right)$$

V = # filters of last



[73] Zhou et al., *Learning deep features for discriminative localization*, CVPR, 2016

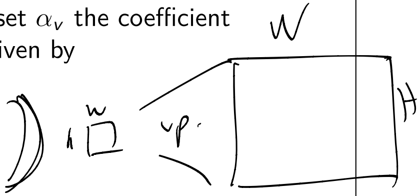
Class Activation Map (CAM)

- formally, if

$$y_c = \sum_{v=1}^{V} \alpha_v \mathrm{GAP}(\mathbf{B}^{(v)}),$$

Definition: Assume GAP-architecture for the last layer and set α_v the coefficient associated to each activation. Then CAM explanations are given by

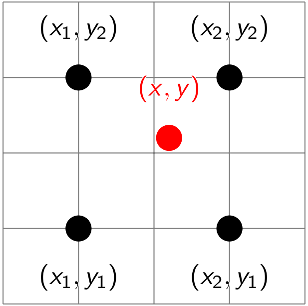
$$[\mathbf{CAM}] := \sum_{v=1}^{V} \alpha_v \mathbf{B}^{(v)} \in \mathbb{R}^{h \times w}.$$

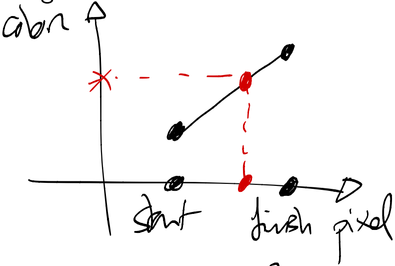


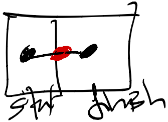
- **Remark (i):** often, $h, w \ll H, W$ $\rightarrow$ upscaling step sor visualization purposes
- **Remark (ii):** specific to this architecture choice

Upscaling

- **Additional step for most methods:** upscale to original image size
- **Example:** bilinear interpolation



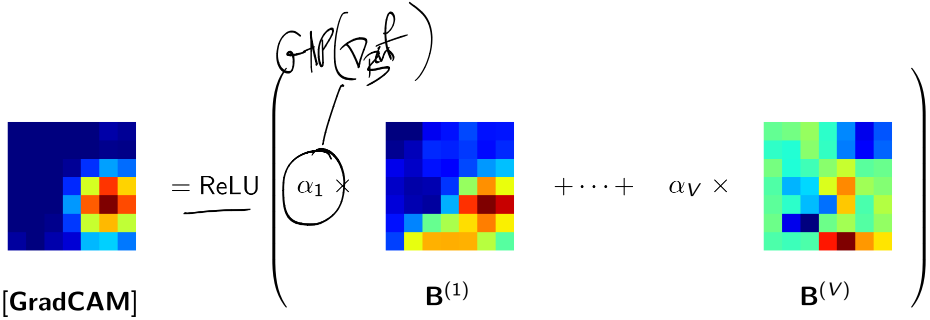




$$f(x, y) = \frac{1}{(x_2 - x_1)(y_2 - y_1)} [f(x_1, y_1)(x_2 - x)(y_2 - y) + f(x_2, y_1)(x - x_1)(y_2 - y)$$
$$f(x_1, y_2)(x_2 - x)(y - y_1) + f(x_2, y_2)(x - x_1)(y - y_1)] \ .$$

GradCAM

- improving on CAM idea, *GradCAM*[74]
- **Main change:** not model specific
- **Solution:** simply compute the gradient of y_c w.r.t. $\mathbf{B}^{(v)}$ and average (GAP)
- then weighted average as in CAM, followed by a ReLU:



[74]Selvaraju et al., *Grad-CAM: why did you say that?*, arxiv, 2016

GradCAM

- formally:

Definition: Assume convolutional architecture, let $\mathbf{B}^{(v)}$ for $v \in [V]$ be the activations of a convolutional layer. Define

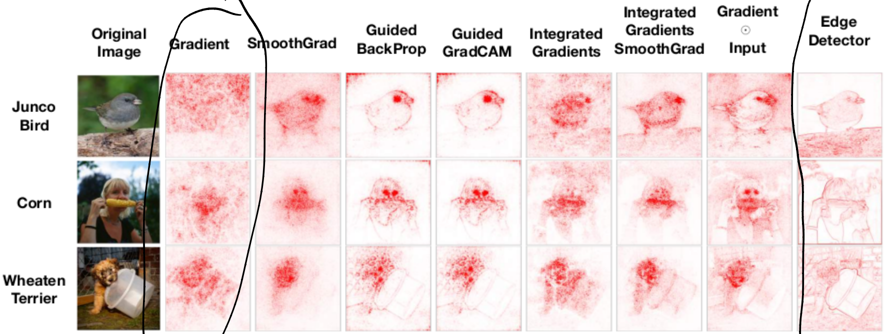
$$\forall v \in [V], \qquad \alpha_v := \mathrm{GAP}\left(\nabla_{\mathbf{B}^{(v)}} f(\mathbf{B})\right) \in \mathbb{R}\,.$$

Then GradCAM explanations are given by

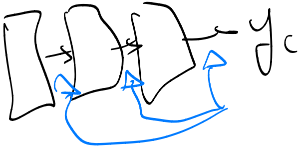
$$[\mathbf{GC}] := \mathrm{ReLU}\left(\sum_{v=1}^{V} \alpha_v \mathbf{B}^{(v)}\right) \in \mathbb{R}_{+}^{h \times w}\,,$$

Criticisms

- **Edge detector:** too much information from ξ can lead to misleading explanations[75]

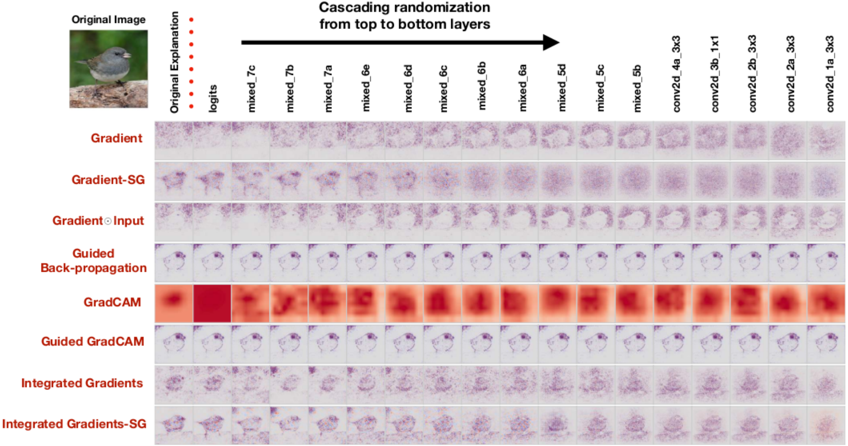


[75] Adebayo et al., *Sanity checks for saliency maps*, NeurIPS, 2018



Criticisms

- **Another way to see this:** increasingly randomize the layers



Summary

- dedicated methods for CNNS:
  - CAM (for specific architecture)
  - GradCAM
- many, many other extensions:
  - GradCAM++ [Chattopadhay et al., 2018]
  - XGradCAM [Fu et al., 2020]
  - ScoreCAM [Wang et al., 2020]
  - AblationCAM [Desai et al., 2020]
  - ~~EigenCAM [Muhammad et al., 2020]~~
  - HiResCAM [Draelos et al., 2020] → $\sum \alpha_v \odot B^{(v)}$
  - Opti-CAM [Zhang et al., 2024]
- **Beware:** these methods are also not perfect!

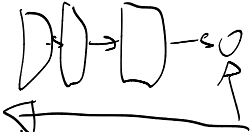
$A \odot B = (A_{ij} B_{ij})_{ij}$

$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \odot \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix} = \begin{pmatrix} 2 & 2 \\ 3 & 8 \end{pmatrix}$

termwise product

7.3. Backpropagation-based methods

Introduction



- **Intuition behind gradient-based methods:** propagate output back to input
- key tool = gradient
- for neural nets, computed by *backpropagation*[76]
- in this section, we review several methods based on modifications of backpropagation
- but first, what is backpropagation?
- **Our setting:** fully-connected neural network, no biases
- **Notation:** $f : \mathbb{R}^d \to \mathbb{R}$, L hidden layers
- $(L+1)$ layer sizes $d_0 = d, d_1, d_2, \ldots, d_L \in \mathbb{N}^\star$
- weight matrices $W^{(h)} \in \mathbb{R}^{d_h \times d_{h-1}}$ for $h \in [L]$
- $\psi : \mathbb{R} \to \mathbb{R}$ is an *activation function* (applied component-wise)
- **Example:** $\psi = \mathrm{ReLU}$

[76] Rumelhart, Hinton, *Learning representations by back-propagating errors*, Nature, 1986

Forward pass

$f^{(1)}(x) = W^{(1)} g^{(0)}(x) = W^{(1)} x$

- define recursively:

$$g^{(0)}(x) = x \in \mathbb{R}^d, \quad f^{(h)}(x) = W^{(h)} g^{(h-1)}(x) \in \mathbb{R}^{d_h} \text{ for } h \in [L],$$

and

$$g^{(h)}(x) = \psi(f^{(h)}(x)) \in \mathbb{R}^{d_h} \text{ for } h \in [L].$$

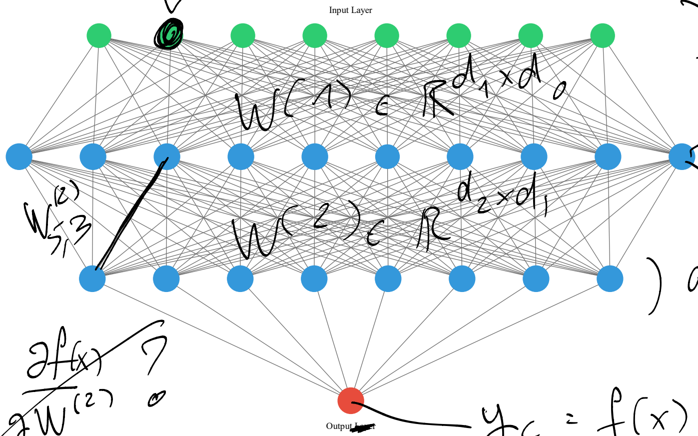
- **Vocabulary:** $f^{(h)}$ = pre-activations, $g^{(h)}$ = activations
- network defined by:

$$y_i = f(x) = f^{(L+1)}(x) = W^{(L+1)} g^{(L)}(x) \in \mathbb{R},$$

with $W^{(L+1)} \in \mathbb{R}^{1 \times d_L}$ $= \mathbb{R}^{d_{L+1} \times d_L}$

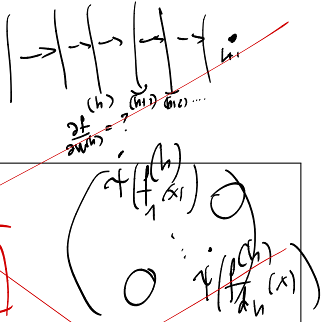
last layer

Forward pass



Backpropagation

- usually, gradient with respect to parameters $(W^{(h)})$



Proposition (backpropagation):[77] For all $h \in [L]$, define

$$D^{(h)}(x) := \operatorname{diag}\left(\dot{\psi}(f^{(h)}(x))\right) \in \mathbb{R}^{d_h \times d_h},$$

and, recursively,

$$b^{(h)}(x) := \begin{cases} 1 \in \mathbb{R} & \text{if } h = L+1 \\ D^{(h)}(x)\left(W^{(h+1)}\right)^\top b^{(h+1)}(x) \in \mathbb{R}^{d_h} & \text{if } h \in [L]. \end{cases}$$

Then

$$\frac{\partial f(x)}{\partial W^{(h)}} = b^{(h)}(x)\left(g^{(h-1)}(x)\right)^\top.$$

[77] adapted from Arora et al., *On exact computation with infinitely wide neural net*, NeurIPS, 2019

$F : \mathbb{R}^M \to \mathbb{R}^N$

$\nabla F \in \mathbb{R}^{M \times N}$

$(\nabla F)_{i,j} = \frac{\partial F_j}{\partial x_i}$

"normal" function:

$f : \mathbb{R}^d \to \mathbb{R}^1$

∇f is a vector

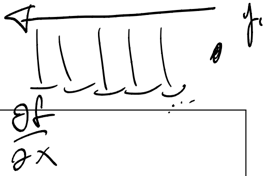
$((\nabla f)^T \ldots)$

$\nabla f \in \mathbb{R}^{d \times 1}$

∇ = "nabla"

Backpropagation

- **Remark:** hence the name: gradient is computed recursively starting from the *last layer*
- **Back to XAI:** we want the gradient with respect to the input
- there is a similar result:



Proposition: We have

$$\frac{\partial f(x)}{\partial x} = \left(W^{(1)}\right)^\top \frac{\partial f(x)}{\partial f^{(1)}} ,$$

and, for all $h \in [L+1]$,

$$\frac{\partial f(x)}{\partial f^{(h)}} = D^{(h)}(x) \left(W^{(h+1)}\right)^\top \frac{\partial f(x)}{\partial f^{(h+1)}} .$$

- $\Rightarrow$ starting from $\partial f / \partial f^{(L+1)} = 1$, we can compute $\partial f / \partial x$ by backpropagation

Proof

- **Reminder:** if $F : \mathbb{R}^a \to \mathbb{R}^b$, then $\partial F/\partial x \in \mathbb{R}^{a\times b}$, with $(\partial F/\partial x)_{i,j} = \partial F_j/\partial x_i$
- **First claim:** by the chain rule $(\partial(F \circ G) = (\partial G) \cdot (\partial F(G)))$,

$$\frac{\partial f(x)}{\partial x} = \frac{\partial f^{(1)}(x)}{\partial x} \frac{\partial f(x)}{\partial f^{(1)}} .$$

- recall that $f^{(1)}(x) = W^{(1)}x$, and notice that

(+ bias)

$$\frac{\partial}{\partial x}(Mx) = M^\top$$

for any fixed matrix

$A \in \mathbb{R}^{M \times N}$. $x \in \mathbb{R}^N$

$F : \mathbb{R}^N \to \mathbb{R}^M$

$x \mapsto Ax$

now we ask: ∇F? $\left(\frac{\partial F}{\partial x}\right)$?

$$\begin{pmatrix} A_{11} & \cdots & A_{1N} \\ \vdots & & \vdots \\ A_{M1} & \cdots & A_{MN} \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_N \end{pmatrix} \begin{bmatrix} A_{11}x_1 + A_{12}x_2 + \cdots + A_{1N}x_N \\ \vdots \\ A_{M1}x_1 + A_{M2}x_2 + \cdots + A_{MN}x_N \end{bmatrix}$$

$$\left(\frac{\partial F}{\partial x}\right)_{i,j} = \frac{\partial F_j}{\partial x_i}$$

$$= \frac{\partial}{\partial x_i}\left(A_{j1}x_1 + \cdots + A_{jN}x_N\right)$$

$$= \frac{\partial}{\partial x_i}\left(A_{ji}x_i\right)$$

$$= A_{ji}$$

$$\boxed{\frac{\partial F}{\partial x} = A^T}$$

Proof, ctd.

- let $h > 1$
- chain rule yields

$$\frac{\partial f(x)}{\partial f^{(h)}} = \frac{\partial f^{(h+1)}}{\partial f^{(h)}} \frac{\partial f(x)}{\partial f^{(h+1)}} \,.$$

- focus on the first term, recall that

$$f^{(h+1)}(x) = W^{(h+1)} g^{(h)}(x) = W^{(h+1)} \psi(f^{(h)}(x)) \,.$$

- chain rule again:

$$\frac{\partial f^{(h+1)}}{\partial f^{(h)}} = \left(\frac{\partial}{\partial f^{(h)}} \psi(f^{(h)}(x)) \right) \left(W^{(h+1)} \right)^{\top} \,.$$

- in this first term we recognize $D^{(h)}(x)$ (see next slide)

Proof, ctd.

- $f^{(h)}(x) \in \mathbb{R}^{d_h}$, and ψ is applied term-wise, thus $\psi(f^{(h)}(x)) \in \mathbb{R}^{d_h}$
- let us compute the gradient with respect to $f^{(h)}$:

$$\left(\frac{\partial \psi(f^{(h)}(x))}{\partial f^{(h)}}\right)_{i,j} = \frac{\partial \psi(z_j)}{\partial z_i} = \dot{\psi}(z_j)\mathbb{1}_{i=j} .$$

- in matrix form, this is

$$\operatorname{diag}\left(\dot{\psi}(f^{(h)}(x))\right) \in \mathbb{R}^{d_h \times d_h} ,$$

also known as $D^{(h)}(x)$

- **Remark:** in many applications, $\psi = \mathrm{ReLU}$
- then $\dot{\psi}(z) = \mathbb{1}_{z>0}$

$\square$

Convolutions

- **Question:** what happens for convolutional layers?
- as stated several times,

  convolutional layers can be seen as fully-connected layers with tied weights

- **Example:** $F \in \mathbb{R}^{2\times2}$, $X \in \mathbb{R}^{3\times3}$,

$$F \star X = \begin{pmatrix} F_{11} & F_{12} \\ F_{21} & F_{22} \end{pmatrix} \star \begin{pmatrix} X_{11} & X_{12} & X_{13} \\ X_{21} & X_{22} & X_{23} \\ X_{31} & X_{32} & X_{33} \end{pmatrix}$$

$$= \begin{pmatrix} F_{11}X_{11} + F_{12}X_{12} + F_{21}X_{21} + F_{22}X_{22} & F_{11}X_{12} + F_{12}X_{13} + F_{21}X_{22} + F_{22}X_{23} \\ F_{11}X_{21} + F_{12}X_{22} + F_{21}X_{31} + F_{22}X_{32} & F_{11}X_{22} + F_{12}X_{23} + F_{21}X_{32} + F_{22}X_{33} \end{pmatrix}$$

Convolutions, ctd.

- define

$$\mathrm{vec}(X) := (X_{11}, X_{12}, X_{13}, X_{21}, X_{22}, X_{23}, X_{31}, X_{32}, X_{33})^\top .$$

- **Remark:** this is the `X.flatten()` operation in Pytorch
- define

$$C(F) := \begin{pmatrix} F_{11} & F_{12} & 0 & F_{21} & F_{22} & 0 & 0 & 0 & 0 \\ 0 & F_{11} & F_{12} & 0 & F_{21} & F_{22} & 0 & 0 & 0 \\ 0 & 0 & 0 & F_{11} & F_{12} & 0 & F_{21} & F_{22} & 0 \\ 0 & 0 & 0 & 0 & F_{11} & F_{12} & 0 & F_{21} & F_{22} \end{pmatrix} .$$

- then one can easily check that

$$\mathrm{vec}(F \star X) = C(F) \cdot \mathrm{vec}(X) .$$

- of course, this is a more general phenomenon

Pooling

- **Average pooling:** can be implemented as a (*frozen*) fully-connected layer:
- **Example:** $X \in \mathbb{R}^{4\times 4}$, $\mathrm{vec}(X) \in \mathbb{R}^{16}$ as before
- set $A(X)$ the 2×2 average pool of X:

$$4A(X) = \begin{pmatrix} X_{11} + X_{12} + X_{21} + X_{22} & X_{13} + X_{14} + X_{23} + X_{24} \\ X_{31} + X_{32} + X_{41} + X_{42} & X_{33} + X_{34} + X_{43} + X_{44} \end{pmatrix}$$

- then, if we set

$$M_{2,2} := \begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & \cdots \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & \cdots \\ \vdots & & & & & & & & & \ddots \end{pmatrix} \in \mathbb{R}^{4\times 16},$$

we have

$$\mathrm{vec}(A(X)) = M_{2,2} \cdot \mathrm{vec}(X)\,.$$

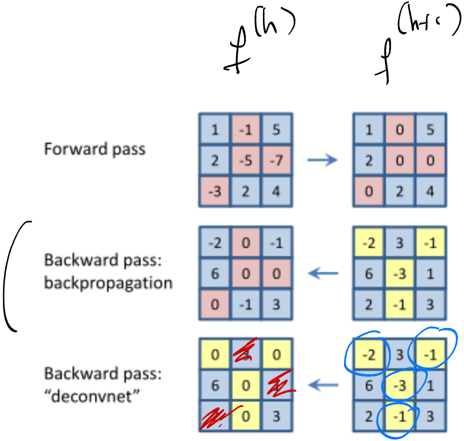
Deconvolution

- back to gradient with respect to input (= saliency maps), ReLU activation
- **Recall:** $f^{(h)}$ = pre-activations, $W^{(h)}$ = weights for layer h
- because of backpropagation, can be seen as *flowing back information* to the input
- the rule is:
$$S_B^{(h)} = \mathrm{diag}\left(\mathbb{1}_{f^{(h)}(x)>0}\right)\left(W^{(h+1)}\right)^\top S_B^{(h+1)},$$
with $S_B^{(L+1)}$ the output we want to explain
- **Deconvolution:**[78] take information from the subsequent layer:
$$\boxed{S_D^{(h)} = \left(W^{(h+1)}\right)^\top S_D^{(h+1)} \mathrm{diag}\left(\mathbb{1}_{f^{(h+1)}(x)>0}\right).}$$

[78] Springenberg et al., *Striving for simplicity: the all convolutional net*, ICLR workshop, 2015, inspired by Zeiler and Fergus, *Visualizing and understanding convolutional networks*, ECCV, 2014

Deconvolution, ctd.

- **Visual aid:**[79]



---

79 courtesy of Springenberg et al., *Striving for simplicity: the all convolutional net*, ICLR workshop, 2015

Guided-backpropagation

- qualitative results can still be fuzzy
- **Additional idea:** incorporate information from both layers
- in equation:

$$S_G^{(h)} = \operatorname{diag}\left(\mathbb{1}_{f^{(h)}(x)>0}\right) \left(W^{(h+1)}\right)^\top S_G^{(h+1)} \operatorname{diag}\left(\mathbb{1}_{f^{(h+1)}(x)>0}\right) .$$

- **Reminder:** backpropagation:

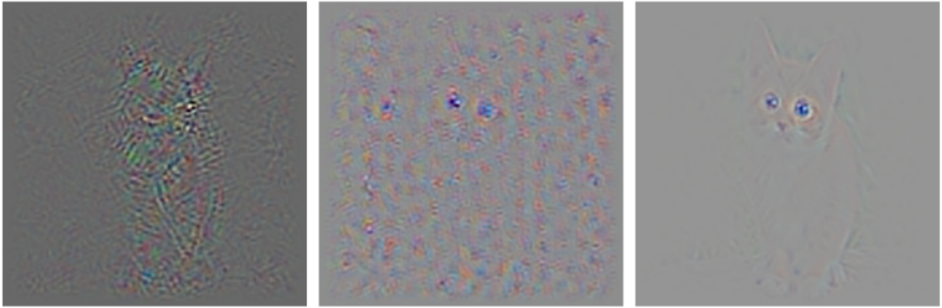
$$S_B^{(h)} = \operatorname{diag}\left(\mathbb{1}_{f^{(h)}(x)>0}\right) \left(W^{(h+1)}\right)^\top S_B^{(h+1)} .$$

- **Reminder:** deconvolution:

$$S_D^{(h)} = \left(W^{(h+1)}\right)^\top S_D^{(h+1)} \operatorname{diag}\left(\mathbb{1}_{f^{(h+1)}(x)>0}\right) .$$

Qualitative results

- **Example:** comparing the methods introduced in this section:



- **Figure:** backpropagation, deconvolution, and guided-backpropagation

Summary

- **Idea:** track signal from higher layers back to the input
- **How?** modify the backpropagation algorithm, “removing” negative gradients
- **Intuition:** correspond to neurons having negative influence on activation of layer we are trying to visualize
- advantage = can inspect all intermediary layers
- drawback = has to be adapted to different architectures