Explainable AI - Lecture 8

Concept based explanations

norwegian open ai lab NTNU

Before we start...

What does feature importance tell us for images?

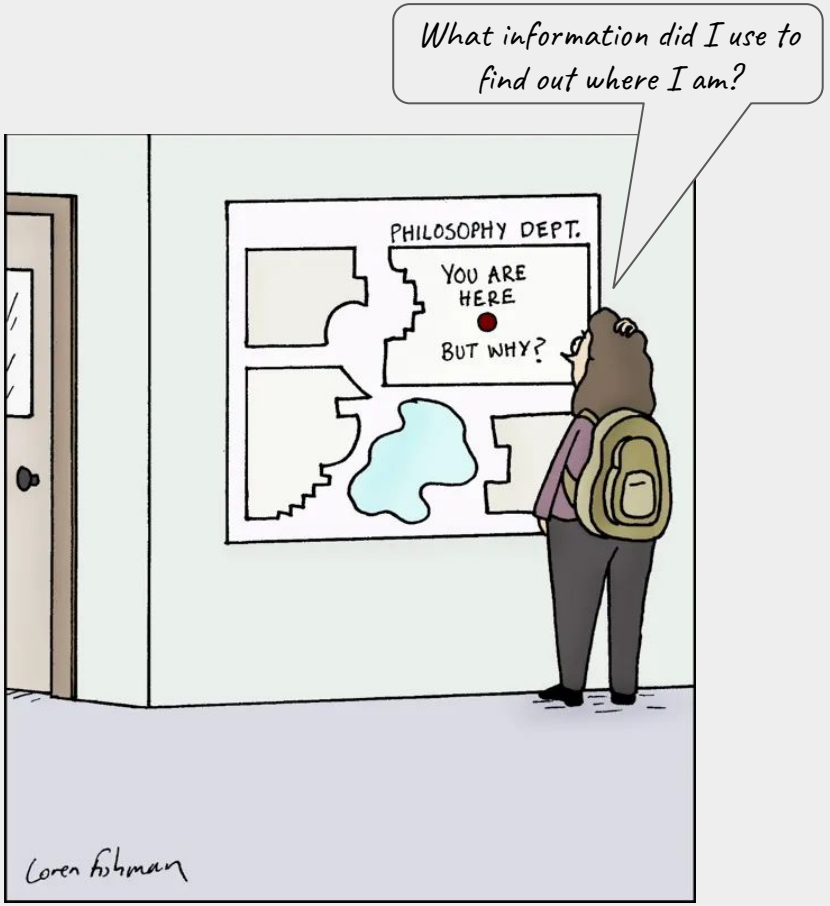


Image SHAP tells us...

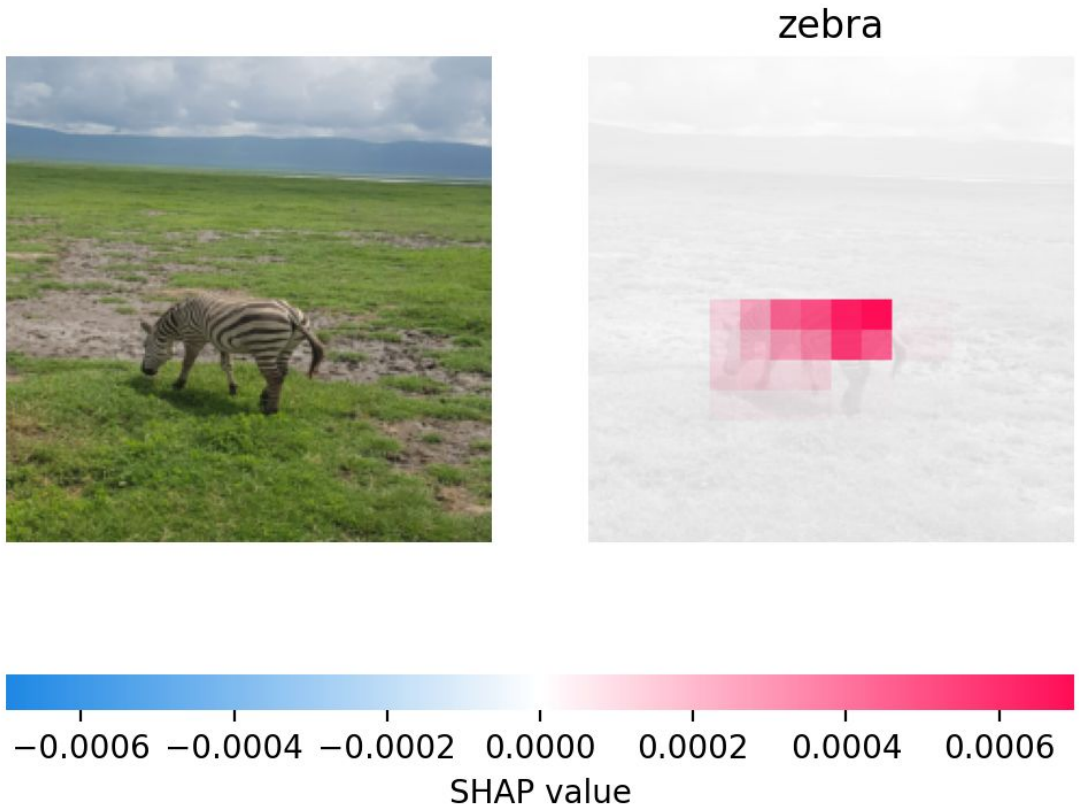
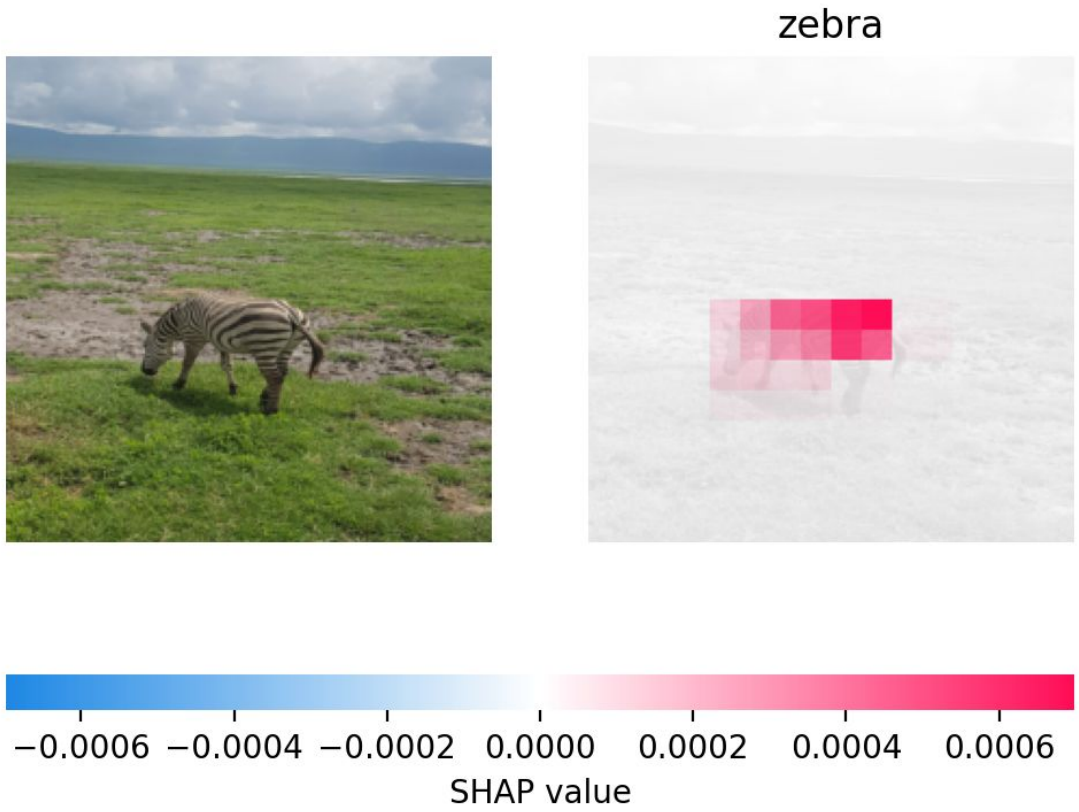
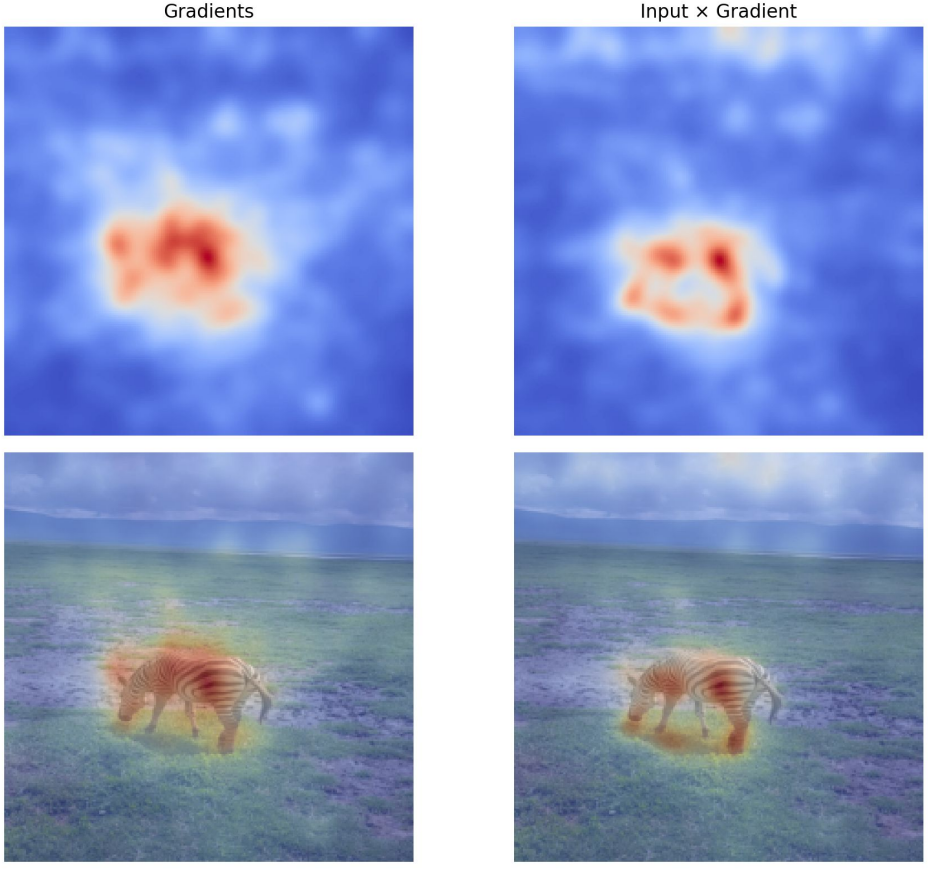


Image SHAP tells us...

How the different features (superpixels) drive the prediction away from the base value, towards the model prediction.

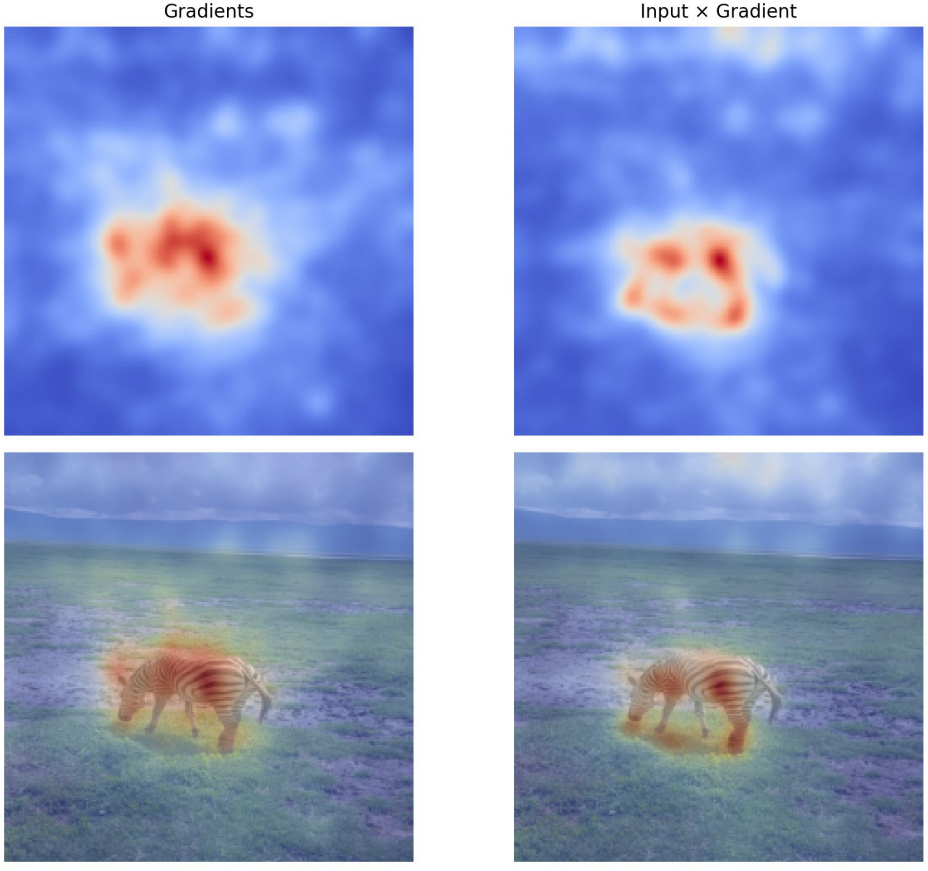


Gradient based saliency maps tell us...



Gradient based saliency maps tell us...

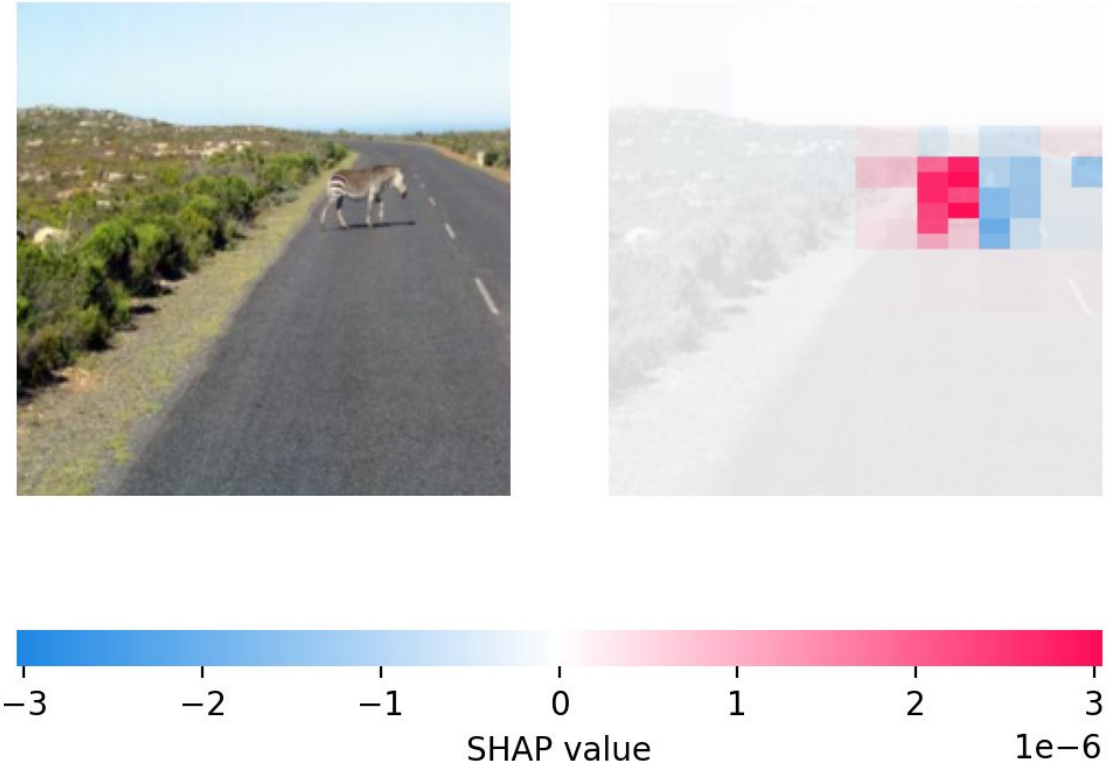
How much the prediction would change for a change in the input pixels.



What do these explanations tell us?

What do you see on this image?

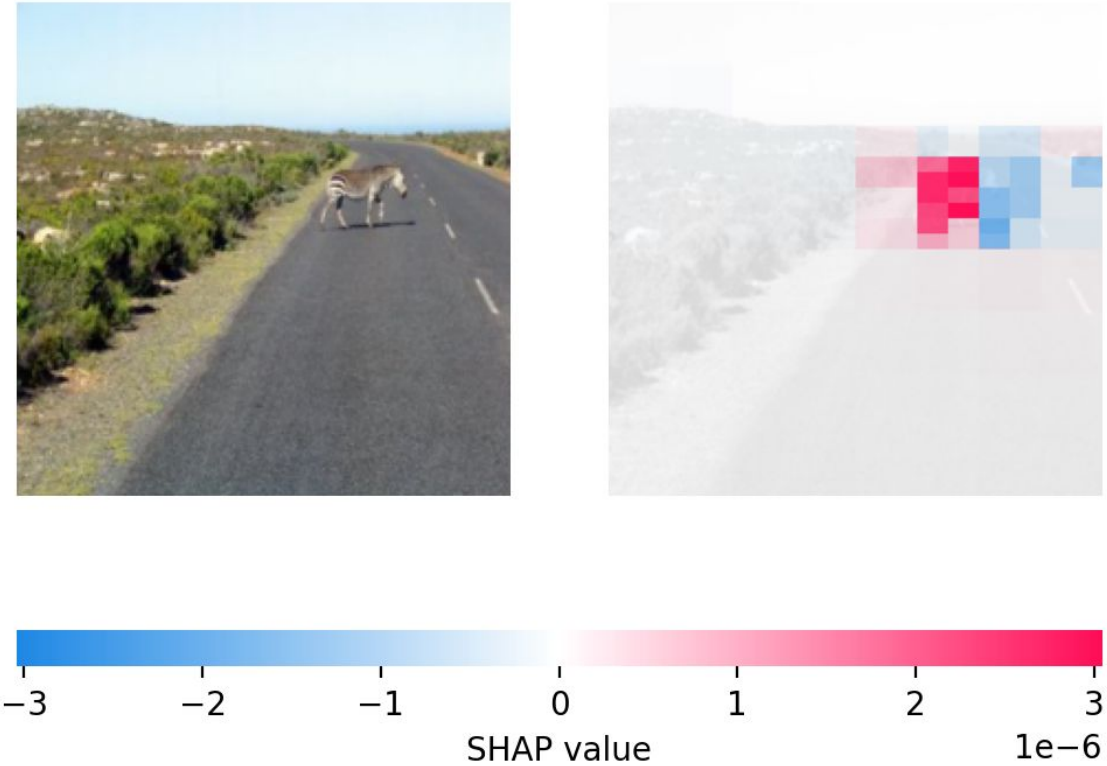
What was important for the classification?



What do these explanations tell us?

This image was classified as a volcano (with low confidence).

Why?

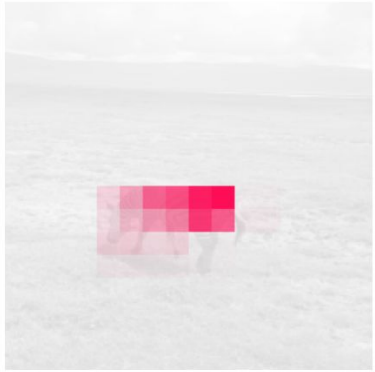


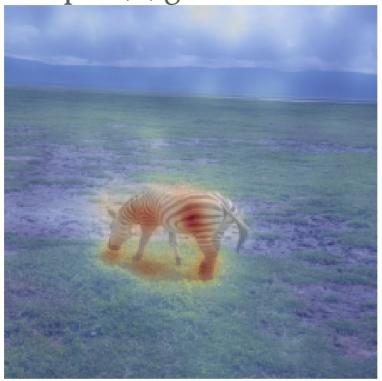
SHAP

Gradients

Input ⨯ gradients

prediction: zebra



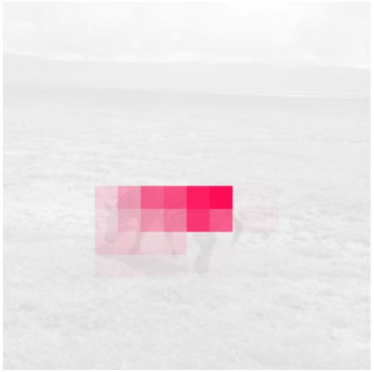


why correct?

prediction: volcano

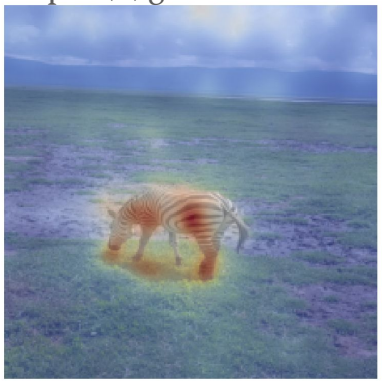
why incorrect?

SHAP



Gradients

Input ⨯ gradients



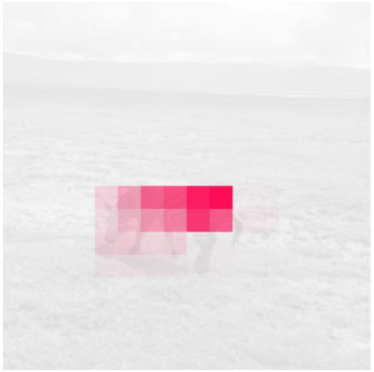
Is the model more sensitive to zebra related or background related concepts?

Did the zebra's texture matter?

Which concept mattered more?

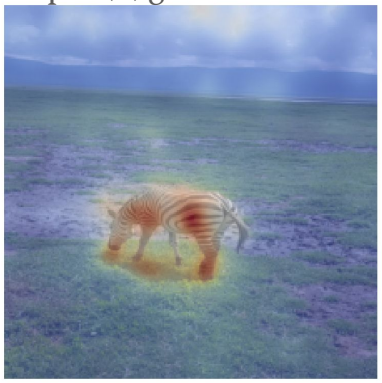
Is this true for all zebra predictions?

SHAP



Gradients

Input ⨯ gradients



Is the model more sensitive to zebra related or background related concepts?

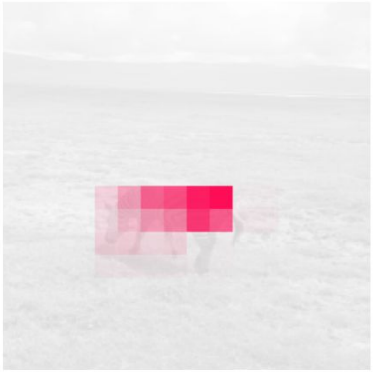
Did the zebra's texture matter?

Which concept mattered more?

Is this true for all zebra predictions?

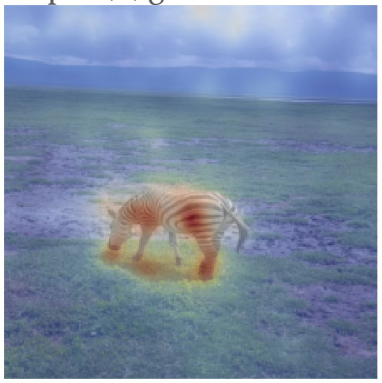
We can't represent these questions in terms of pixel(group)s!

SHAP



Gradients

Input ⨯ gradients



Is the model more sensitive to zebra related or background related concepts?

Did the zebra's texture matter?

Which concept mattered more?

Is this true for all zebra predictions?

However, we can measure how important any such concepts (defined by us!) are to the model.

We'd like to know what our model has *conceptualised*

What does our model do with the data from the input to the output layer?

What information does it extract? Does it add anything?

The data processing inequality

Assume three random variables X, Y, Z forming the Markov chain $X \rightarrow Y \rightarrow Z$

Meaning: *X is generated, then Y is sampled from X through some $p(y|x)$, then Z is sampled from a function of Y, $p(z|y)$. Any dependence of Z on X is mediated by Y.*

The data processing inequality

Assume three random variables X, Y, Z forming the Markov chain $X \rightarrow Y \rightarrow Z$

Meaning: *X is generated, then Y is sampled from X through some $p(y|x)$, then Z is sampled from a function of Y, $p(z|y)$. Any dependence of Z on X is mediated by Y.*

In this setting, no post-processing of Y can *increase* the information that Y contains about X. Using mutual information notation:

$I(X;Y) \geq I(X;Z)$

The mutual information quantifies how much information about X is obtained by observing Y (or Z), and it captures any dependence (linear or nonlinear).

The data processing inequality

Assume three random variables X, Y, Z forming the Markov chain $X \rightarrow Y \rightarrow Z$

We have $I(Y;X) \geq I(Z;X)$, i.e., Z cannot tell us anything about X that is not contained in Y.

In other terms, post-processing can only lose or discard information from the input; the process of generating Z cannot add reliable distinctions about X that weren't already in Y.

The data processing inequality

Assume three random variables X, Y, Z forming the Markov chain $X \rightarrow Y \rightarrow Z$

We have $I(Y;X) \geq I(Z;X)$, i.e., Z cannot tell us anything about X that is not contained in Y.

In other terms, post-processing can only lose or discard information from the input; the process of generating Z cannot add reliable distinctions about X that weren't already in Y.

The data processing inequality tells us that *post-processing cannot increase the information available*.

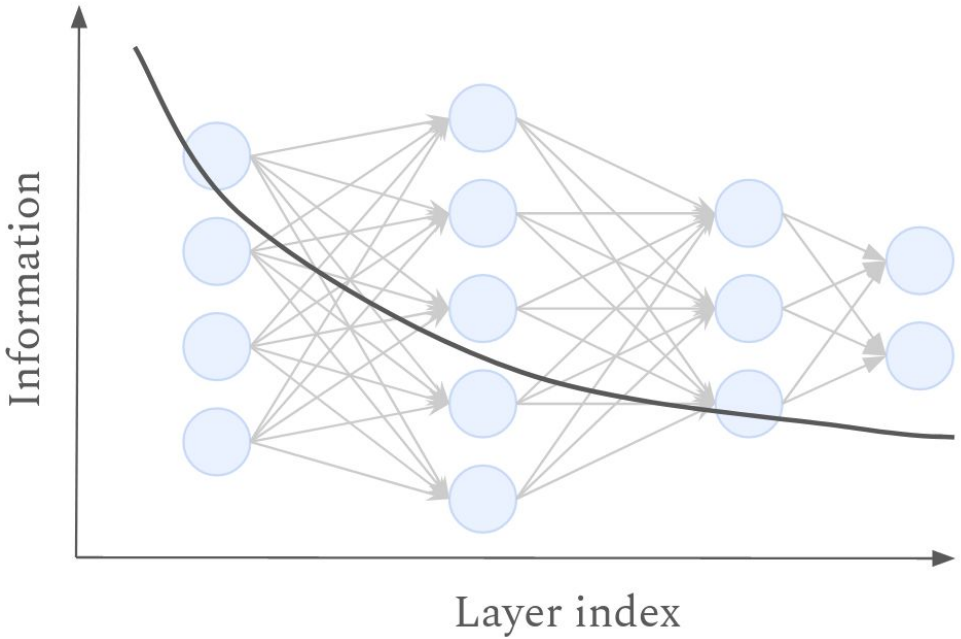
(you don't have to trust me, the proof is pretty simple. Check Wikipedia, for instance)

What does this imply for machine learning models that estimate some *y* based on training data *x*?

Q: What can we expect the model to know?

A: Nothing that's not contained in the training data.

Data processing inequality: post-processing cannot increase the information available.

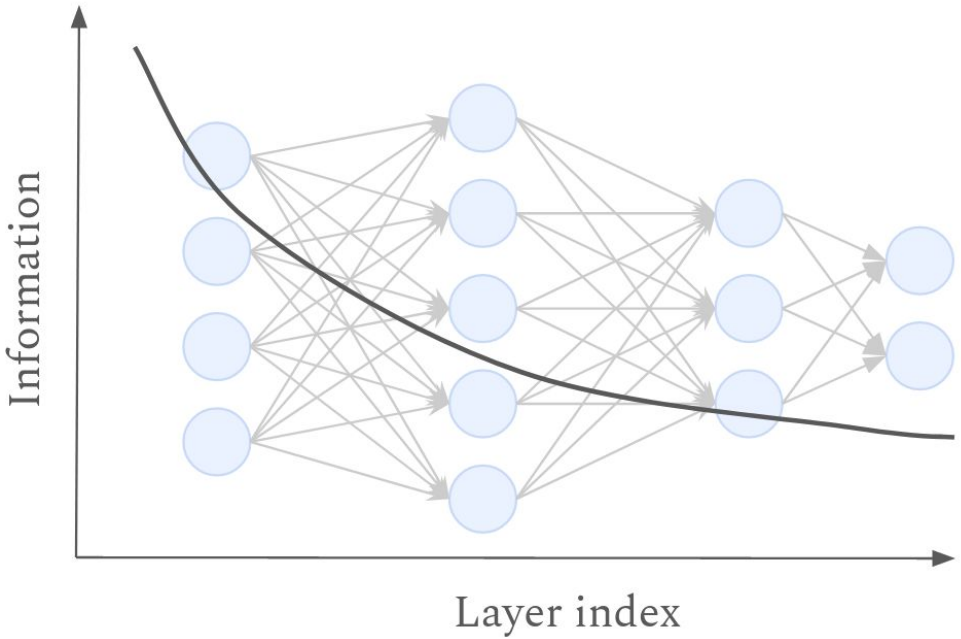


Based on the information perspective, what should we try to find out about our models?

Q: What can we expect the model to know?

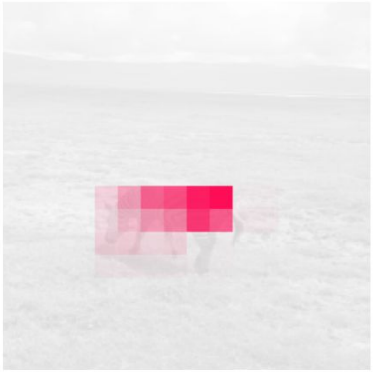
A: Nothing that's not contained in the training data.

Data processing inequality: post-processing cannot increase the information available.



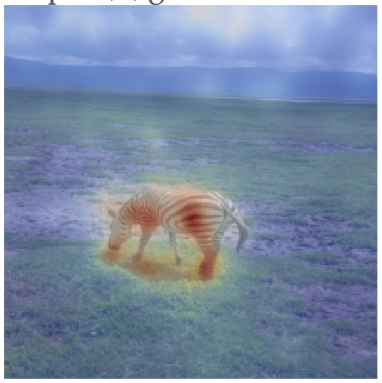
We should ask: "What part of the information present in the input data does the model extract as it processes the data from the input to the output space?"

SHAP



Gradients

Input ⨯ gradients



Is the model more sensitive to zebra related or background related concepts?

Did the zebra's texture matter?

Which concept mattered more?

Is this true for all zebra predictions?

We can measure how important any concepts (defined by us!) are to the model

Concepts

Quickly: *What separates a zebra from a horse?*

Concepts

What separates a zebra from a horse?

Do we expect that a machine learning model capable of distinguishing a zebra from a horse has conceptualized "stripes"?

It would be nice if we could find out whether the stripes concept matters to the model, if another concept matters more, if this is the case for all other zebra predictions, …

Concepts

What separates a zebra from a horse?

Do we expect that a machine learning model capable of distinguishing a zebra from a horse has conceptualized “stripes”?

It would be nice if we could find out whether the stripes concept matters to the model, if another concept matters more, if this is the case for all other zebra predictions, …

The composite *concepts* in the images (stripes, dots, human, door, …)

1. cannot be represented in terms of pixels (or pixel groups), and
2. are not explicitly available in the input data.

Still, if we could quantify how important a user-chosen concept is, we could find out if the model has conceptualized and also utilizes this concept.

Concepts

What separates a zebra from a horse?

Do we expect that a machine learning model capable of distinguishing a zebra from a horse has conceptualized “stripes”?

It would be nice if we could find out whether the stripes concept matters to the model, if another concept matters more, if this is the case for all other zebra predictions, …

The composite *concepts* in the images (stripes, dots, human, door, …)

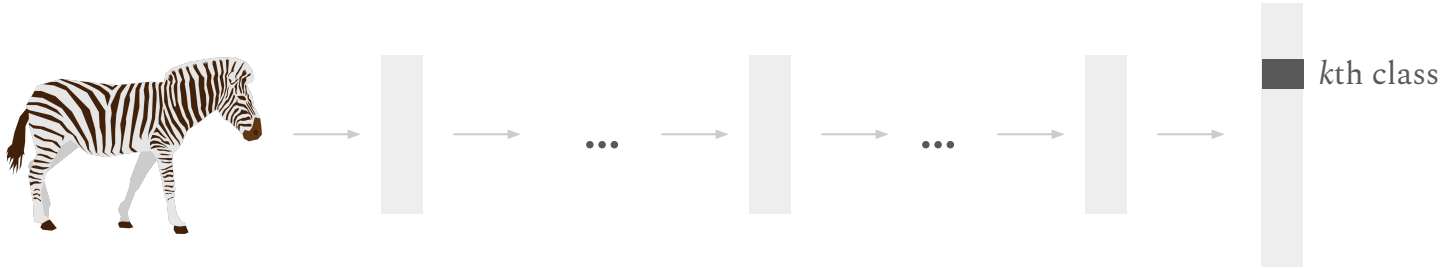
1. cannot be represented in terms of pixels (or pixel groups), and
2. are not explicitly available in the input data.

Still, if we could quantify how important a user-chosen concept is, we could find out if the model has conceptualized and also utilizes this concept.

Keep in mind: a concept is not localized in the image, and it’s not an input feature.

Concept based explanations, step by step

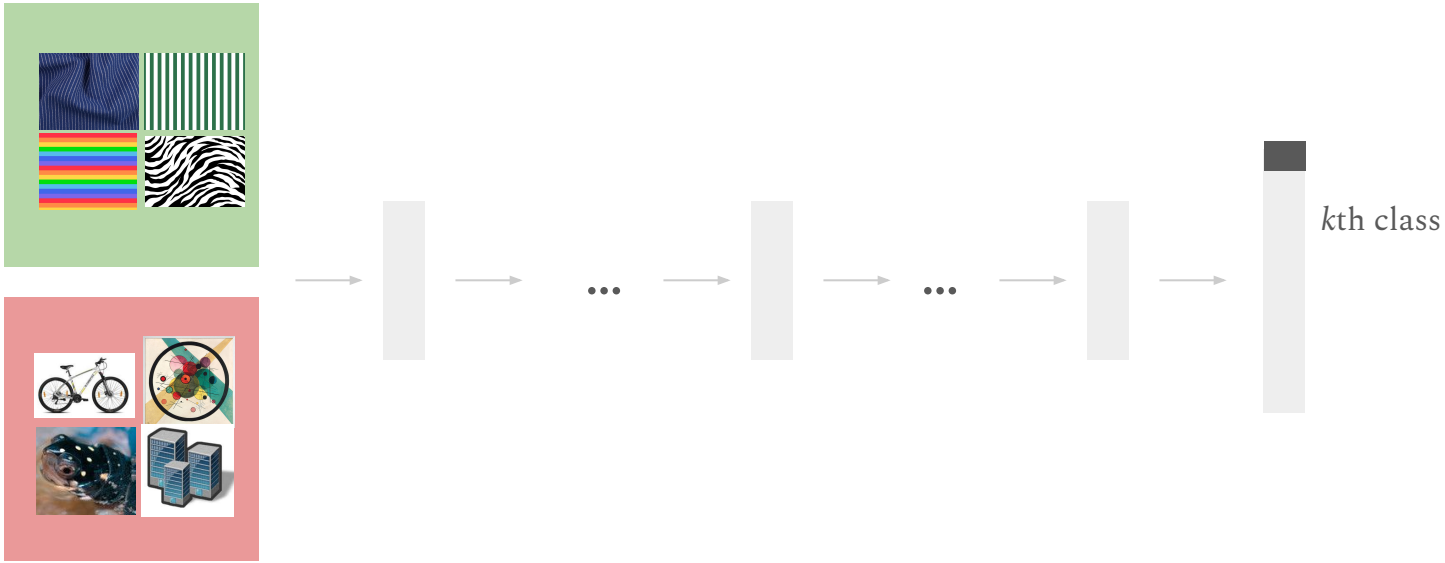
How much did the stripes of this zebra contribute to the prediction of the model?



How to represent a concept that is not explicitly labeled in the data?

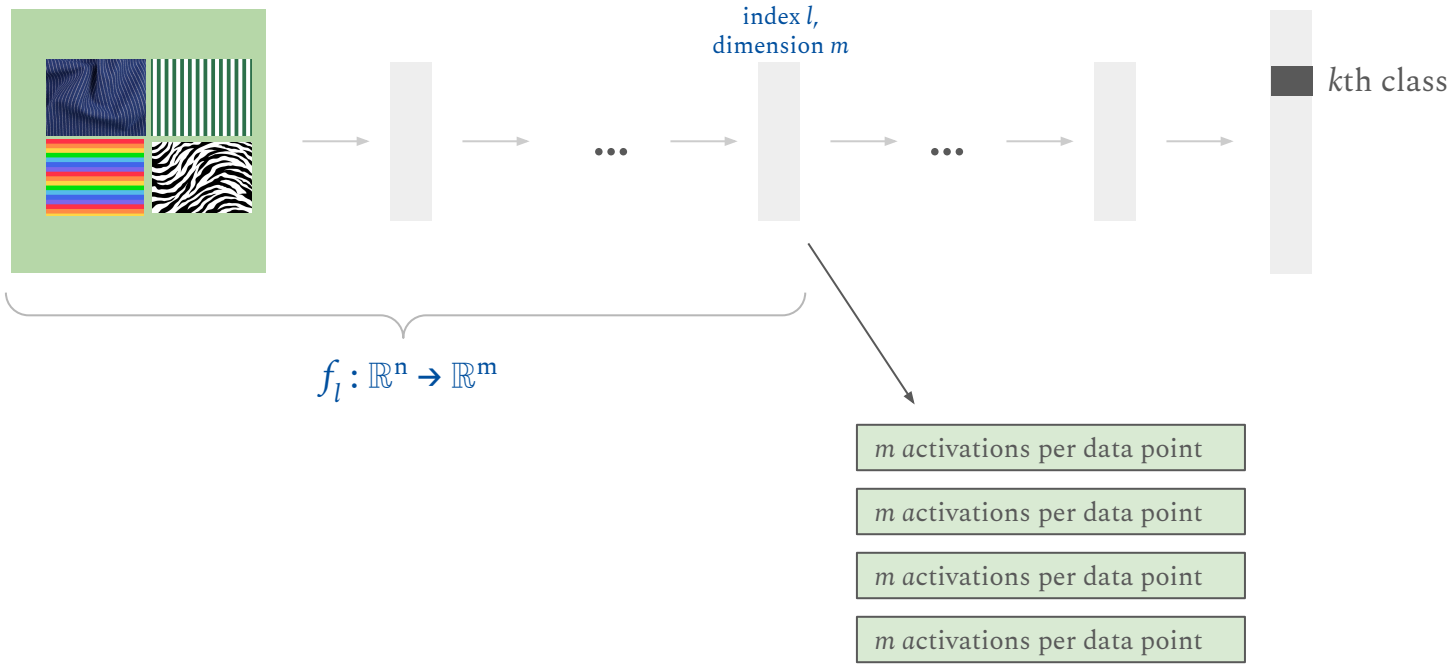
Concept based explanations, step by step

Step 1: Make a concept positive data set representing the human-defined concept, in this case *stripes*, and a concept negative data set representing the concept being absent.



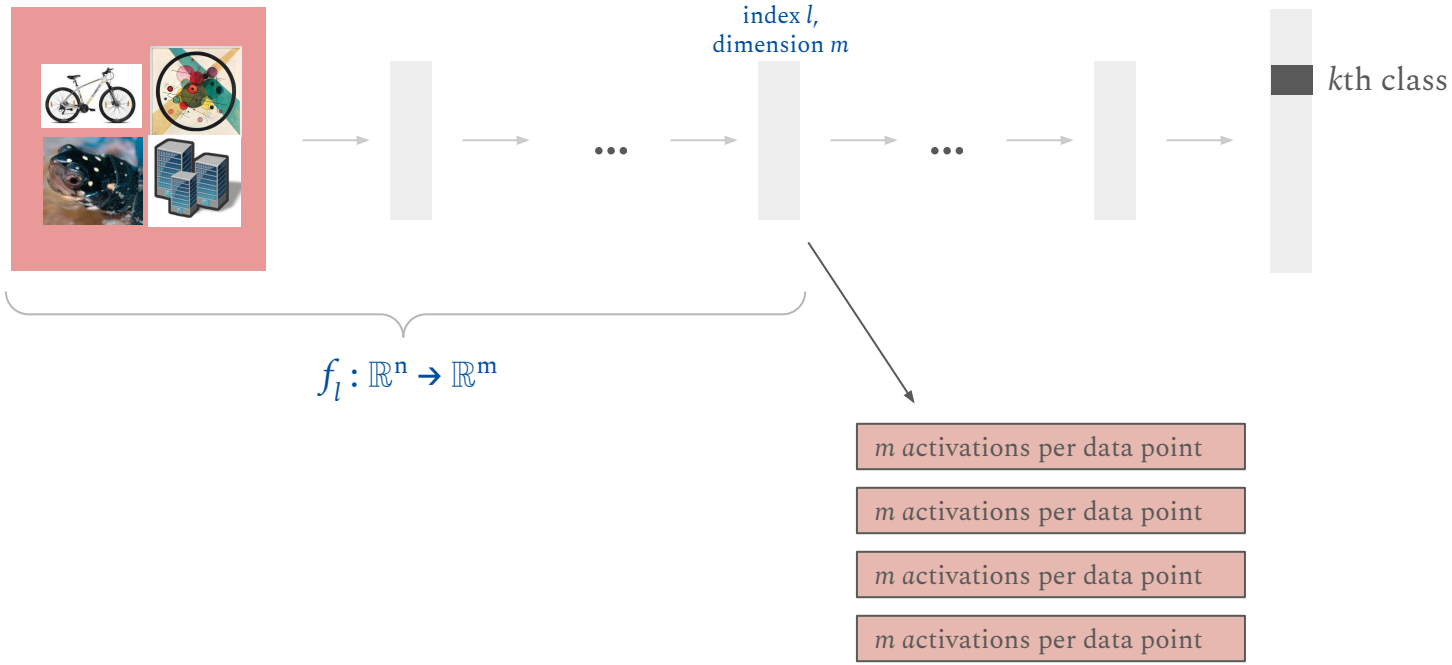
Concept based explanations, step by step

Step 2: Run forward passes on the two data sets, from the input space (dimension n), to some layer l of interest (dimension m). Collect the activations of that layer.



Concept based explanations, step by step

Step 2: Run forward passes on the two data sets, from the input space (dimension n), to some layer l of interest (dimension m). Collect the activations of that layer.

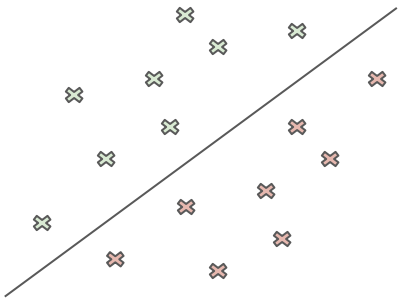


Concept based explanations, step by step

Step 3: Train a linear classifier on the activations of the concept positive and the concept negative data sets.

m activations for positive point 1	label 1
m activations for positive point 2	label 1
...	...
m activations for positive point *n*	label 1
m activations for negative point 1	label 0
m activations for negative point 2	label 0
...	...
m activations for negative point *n*	label 0

→ Logistic regression →

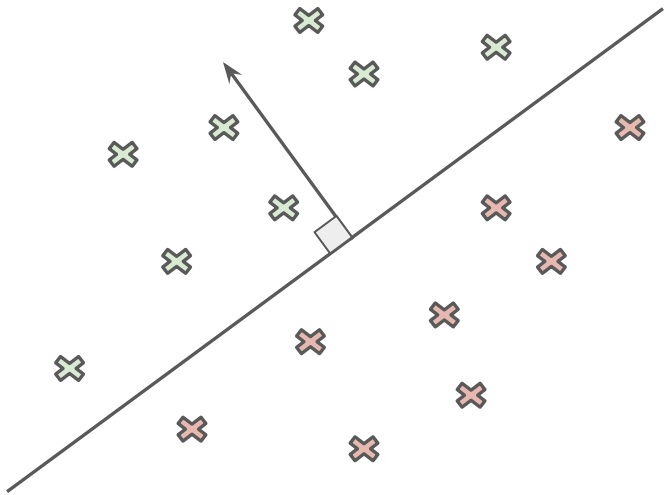


Logistic regression model separating the positive and negative concept data in the *m*-dimensional activation space of layer *l*. ***This is often referred to as a probe.***

Concept based explanations, step by step

Step 4: Find the normal vector to the decision surface of the logistic regression model (the probe).

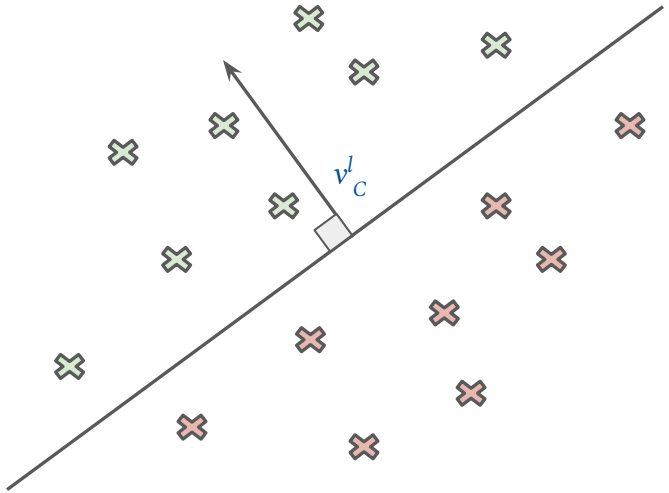
Which direction is this vector pointing in?



Concept based explanations, step by step

Step 4: Find the normal vector to the decision surface of the logistic regression model (the probe).

This vector points in the direction of the positive concept activations. We call it the concept activation vector (CAV), denoted v^l_C for layer l and concept C.

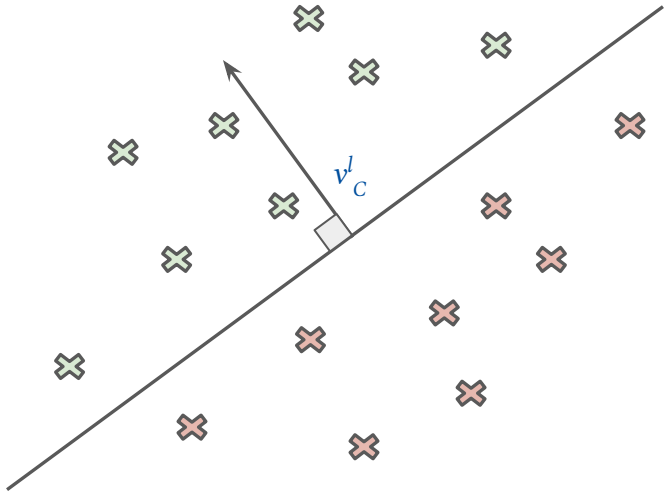


Been Kim et al, Proceedings of the 35 th International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018

Concept based explanations, step by step

What is the concept activation vector (CAV)? What does it represent?

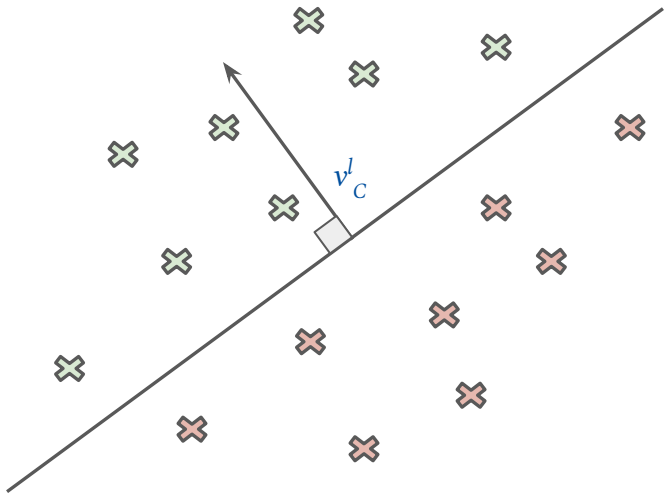
How many dimensions does the space of the CAV have?



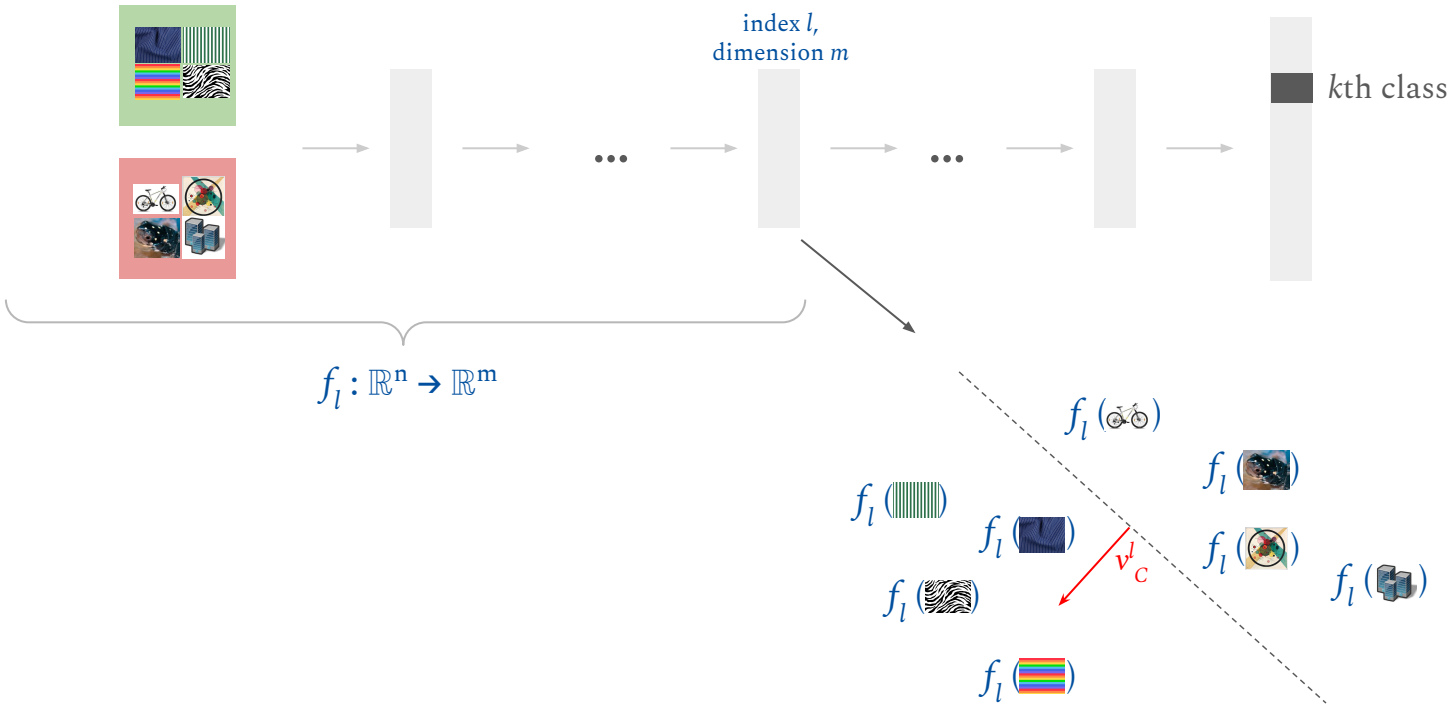
Concept based explanations, step by step

The CAV represents the direction in which the concept increases in the activation space.

The space of the CAV is m dimensional, m being the number of nodes in the layer we got the activations from.



CAVs: summary so far



Been Kim et al, Proceedings of the 35 th International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018

How to get activations from a pytorch model

We need to use hooks.

In short: A forward hook is a user-defined function in PyTorch that automatically runs during a module's forward pass, giving access to that layer's inputs and outputs without altering the model itself.

Hooks are callable objects that can be registered to any `nn.Module` object.

Hooks run automatically during a `forward()` pass.

Since intermediate layers of a model are of the type `nn.Module`, we can use forward hooks on them to serve as a lens to view their activations.

How to use hooks for getting activations

1. Create a container for activations (like a dict).
2. Define a hook function that saves layer outputs.
3. Register it on a specific layer of the model.
4. Run a forward pass — the hook fires automatically.
5. Remove the hook.
6. Use the activations container to inspect the activations.

1. Create a container for activations

```
activations = {}
```

Basic hook syntax

```
hook(model, input, output) -> None or modified output
```

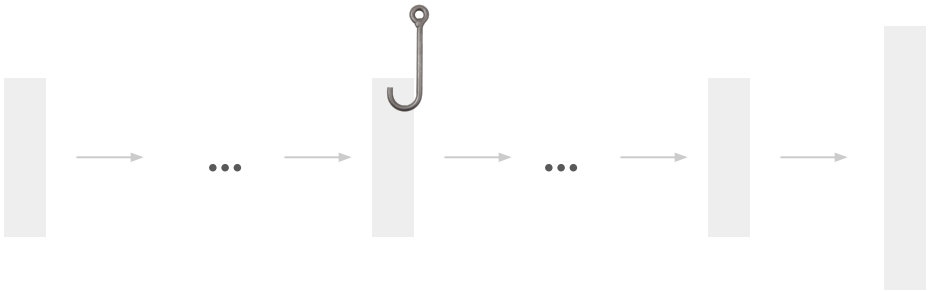
Arguments:

- Either the full model or the `nn.Module` instance to which the hook is attached, depending on whether we use `register_forward_hook` to run on the whole model or `register_module_forward_hook` to run only on one module.
- The input tensor(s) to the model's / module's `forward()` method.
- The output tensor(s) produced by the model's / module's `forward()` method.

2. Define a hook function that saves layer outputs

```
def get_activation(name):
     def hook(model, input, output):
          activations[name] = output.detach()
     return hook
```

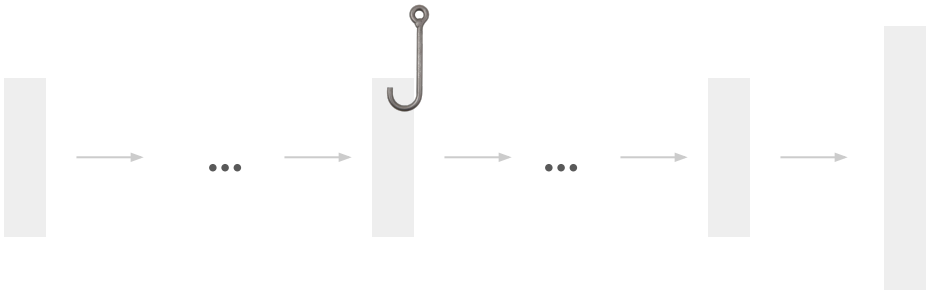
This is the actual hook function, which in our case takes in the name of the layer we're currently interested in, and modifies the activations dictionary with the output (activations) of that layer.



3. Register the forward hook

Registration: A forward hook is registered to an `nn.Module` instance using the `register_forward_hook()` method. This method takes a callable (the hook function) as an argument.

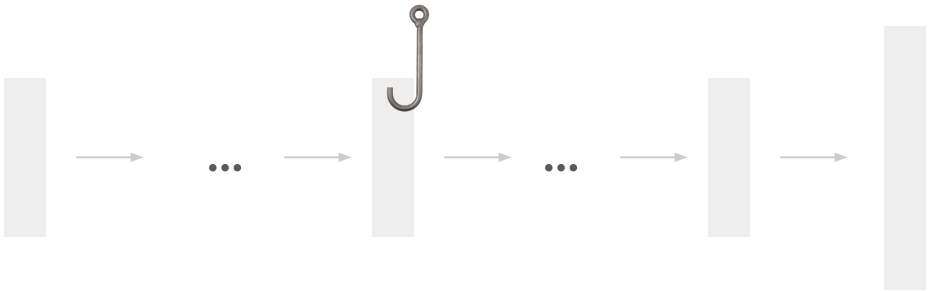
```
h = model.layername.register_forward_hook(get_activation(name))
```



4. Run a forward pass — the hook fires automatically.

When the `forward()` method of the module to which the hook is registered is called, the hook function is automatically executed after the module's forward computation is finished.

```
_ = model(x)
```

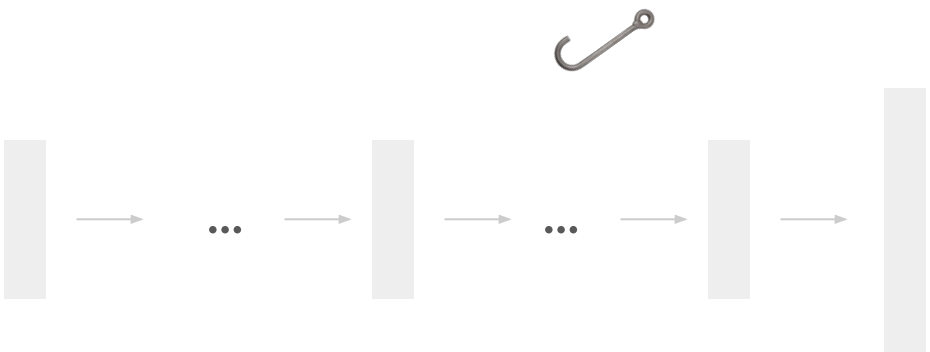


5. Remove the hook

```
h.remove()
```

After we register a hook, it remains registered, meaning it will be run **for every subsequent forward pass**. Hooks keep references to tensors in memory, and especially in Jupyter Notebooks, they can stack up silently.

If you're done with your inspections, you don't want the hook function to keep being called. Remove it.



Basic (forward) hook syntax for getting activations

```
activations = {} # a dict to store the activations
# keys = layer names (strings), values = activation tensors (detached from the computation graph).

def get_activation(name):
     # function factory for creating a hook function capturing the name of the layer it's attached to.
     def hook(model, input, output):
          activations[name] = output.detach()
     return hook

# Register forward hooks on the layers we want.
h = model.layername.register_forward_hook(get_activation(name))

_ = model(x) # run a forward pass

# Detach the hooks after the forward pass
h.remove()
```

Code :)

Let's get our resnet model and imagenet pics out again

```
batch_size = 16
num_concept_images = 100
num_target_images = 20
target_class = "zebra"
positive_concept = "striped" #"bubbly"
negative_concept = "dotted"
```

```
# Get the absolute path to concept and target images
base_dir = os.path.join(os.getcwd(), '..')
concept_folder = os.path.join(base_dir, 'data', 'broden', 'dtd')
target_folder = os.path.join(base_dir, 'data', 'imagenet', target_class)

p_concept_paths = select_concept_images(folder_path=concept_folder, concept=positive_concept, n=num_concept_images)
n_concept_paths = select_concept_images(folder_path=concept_folder, concept=negative_concept, n=num_concept_images)

target_paths = sample_images(folder_path=target_folder, n=num_target_images)
```

```
resnet = models.resnet18(weights=models.ResNet18_Weights.IMAGENET1K_V1).to(device)
resnet.eval();
```

```
class_idx = ResNet18_Weights.IMAGENET1K_V1.meta["categories"].index(target_class)
```

```
transform = transforms.Compose([transforms.ToTensor(), transforms.Resize((224, 224)),
                                transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])])

positive_loader = get_resnet_dataloader(p_concept_paths, transform, batch_size=batch_size)
negative_loader = get_resnet_dataloader(n_concept_paths, transform, batch_size=batch_size)
target_loader = get_resnet_dataloader(target_paths, transform, batch_size=batch_size)
```

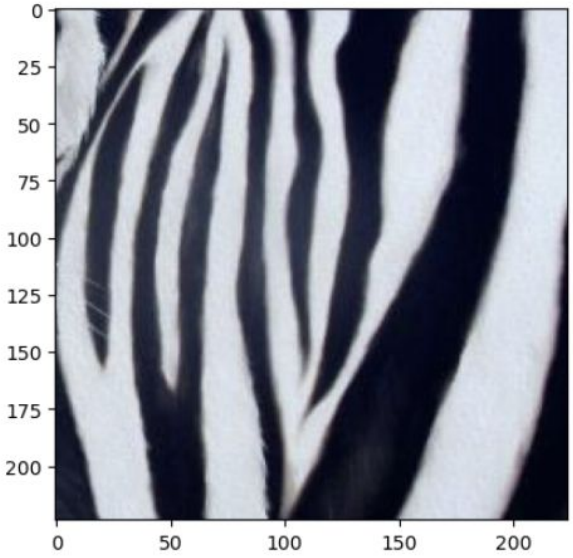
The Broden dataset combines several segmentation datasets, and it is insanely useful for concept detection in images.

Examples from our positive and negative concept data sets

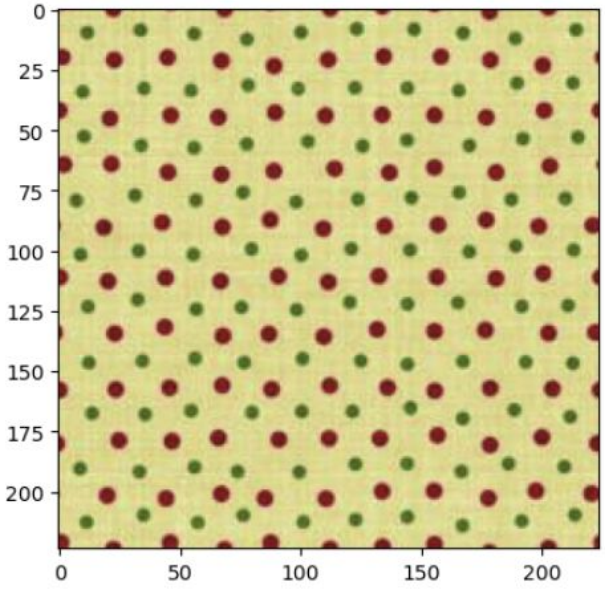
```
def look_at(image):
    img = (image - image.min()) / (image.max() - image.min())
    plt.imshow(img.permute(1, 2, 0))
    plt.show()
```

```
pos_image = next(iter(positive_loader))[7]
print(pos_image.shape)
look_at(pos_image)
```

```
torch.Size([3, 224, 224])
```




```
neg_image = next(iter(negative_loader))[7]
look_at(neg_image)
```



Pick some layer

```
sample_input = pos_image.unsqueeze(0).to(device)

with torch.no_grad():
    print(resnet(sample_input).shape)
```

```
torch.Size([1, 1000])
```

We remember that our resnet model does classification among 1000 different classes

```
layer = resnet.layer4[0]
layer._modules
```

```
{'conv1': Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False),
 'bn1': BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True),
 'relu': ReLU(inplace=True),
 'conv2': Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False),
 'bn2': BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True),
 'downsample': Sequential(
   (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
   (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
 )}
```

Pick one layer we want to inspect and look at it

Let's get the activations from this layer during one forward pass.

Pick some layer

```
sample_input = pos_image.unsqueeze(0).to(device)

with torch.no_grad():
    print(resnet(sample_input).shape)
```

```
torch.Size([1, 1000])
```

We remember that our resnet model does classification among 1000 different classes

```
layer = resnet.layer4[0]
layer._modules
```

```
{'conv1': Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False),
 'bn1': BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True),
 'relu': ReLU(inplace=True),
 'conv2': Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False),
 'bn2': BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True),
 'downsample': Sequential(
   (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
   (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
 )}
```

Pick one layer we want to inspect and look at it

Let's get the activations from this layer during one forward pass. If you're very familiar with convnets, you can see how many activations we'll get from this layer.

(*it's* 512 x 7 x 7 = 25088)

Use a hook to get the activations from layer 4

```
activations = {}
# keys = layer names (strings)
# values = activation tensors

def get_activation(name):
    # function factory for creating a hook function that captures the name of the layer it's attached to.
    def hook(model, input, output):
        activations[name] = output.detach()
    return hook

layer_name = 'layer4'
handle = dict(resnet.named_modules())[layer_name].register_forward_hook(get_activation(layer_name))

_ = resnet(pos_image.unsqueeze(0).to(device))

handle.remove()

print(activations["layer4"].shape)
```

These are our activations from layer4

```
torch.Size([1, 512, 7, 7])
```

We want to use these activations as data points for making a CAV, so we flatten them:

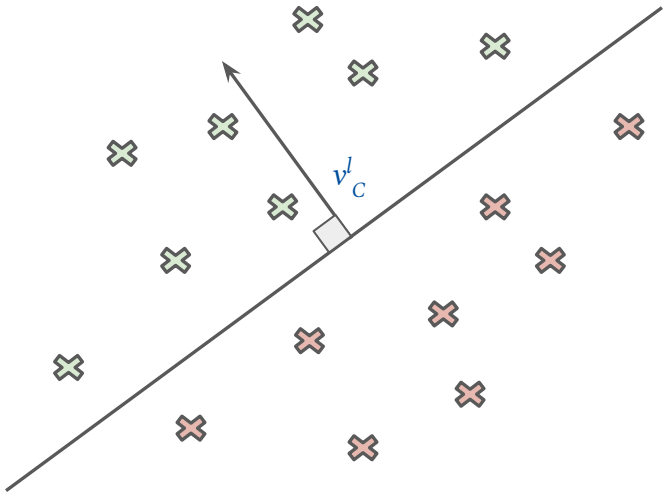
```
acts = activations["layer4"]
acts.view(acts.size(0), -1).numpy().shape
```

```
(1, 25088)
```

From the activations to a CAV

Great, we now know how to get activations for data points from our model, and can use these to make CAVs.

Remember, a CAV is the unit normal vector to a logistic regression model trained to separate concept positive and negative samples.



Code for making a CAV

```
def train_cav(positive_activations, negative_activations):
    # input: the layer activations on the positive and negative concept data

    # we use 1s and 0s as targets for the positive and negative activations, respectively
    X = np.concatenate([positive_activations, negative_activations], axis=0)
    y = np.concatenate([np.ones(len(positive_activations)),
np.zeros(len(negative_activations))])

    # use all the data for training, or make a train/test split
    X_train, y_train = shuffle(X, y, random_state=seed)

    # our classifier is a logistic regressor (straight line separating the classes)
    classifier = LogisticRegression(random_state=seed, max_iter=1000)
    classifier.fit(X_train, y_train)

    # the logistic regression function is f(x)=w^T x+b,
    # and the normal vector is the gradient, here ∇f(x)=w
    cav = classifier.coef_[0]

    # we want the normalised (unit length) normal vector
    unit_cav = cav / np.linalg.norm(cav)

    return unit_cav
```

Concept activation vectors

Knowing concept directions in a model's activation space isn't a **quantitative** explanation, and we have to keep working a bit on the CAVs before we're happy.

However, CAVs can be useful in themselves.

What does the accuracy of the logistic regression model producing the CAV tell us?

Concept activation vectors

Knowing concept directions in a model's activation space isn't a **quantitative** explanation, and we have to keep working a bit on the CAVs before we're happy.

However, CAVs can be useful in themselves.

The accuracy of the logistic regression model producing the CAV tells us how accessible the concept is in the internal representation of the model.

We can interpret the accuracy as an "importance of the concept, according to the model".

Making a CAV and the accuracy of the probe

```python
def train_cav(positive_activations, negative_activations, seed=1337):

    X = np.concatenate([positive_activations, negative_activations], axis=0)
    y = np.concatenate([np.ones(len(positive_activations)), np.zeros(len(negative_activations))])

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, shuffle=True, random_state=seed)
    #X_train, y_train = shuffle(X, y, random_state=seed)
    classifier = LogisticRegression(random_state=seed, max_iter=1000)
    classifier.fit(X_train, y_train)

    accuracy = classifier.score(X_test, y_test)
    print("Probe accuracy:", accuracy)

    cav = classifier.coef_[0]
    unit_cav = cav / np.linalg.norm(cav)

    return unit_cav
```

Fit the probe

If we want the accuracy, we have to do a train/test split

Normalise the CAV and return the unit CAV

```python
layer = resnet.layer4[0]
```

Choose a layer

```python
positive_activations = flatten(get_activations(resnet, positive_loader, device, layer))
negative_activations = flatten(get_activations(resnet, negative_loader, device, layer))
```

```python
stripes_cav = train_cav(positive_activations, negative_activations)
```

```
Probe accuracy: 1.0
```

Train the CAV and look at the accuracy

```python
print(stripes_cav.shape)
print(stripes_cav)
```

```
(25088,)
[-0.00176862 -0.01537377 -0.01394483 ...  0.00403293  0.00090289
  0.00399448]
```

Usefulness of probe accuracy

Knowing concept directions in a model's activation space isn't a **quantitative** explanation, and we have to keep working a bit on the CAVs before we're happy.

However, CAVs can be very useful in themselves.

The accuracy of the logistic regression model producing the CAV tells us how accessible the concept is in the internal representation of the model.

We can interpret the accuracy as an "importance of the concept, according to the model".

A very interesting application of this is to calculate the CAV logistic regression accuracy for different layers and training steps of a model.

Concept activation vectors in AlphaZero

In Acquisition of chess knowledge in AlphaZero, the authors monitor the evolution of different concepts throughout the training of the chess model AlphaZero.

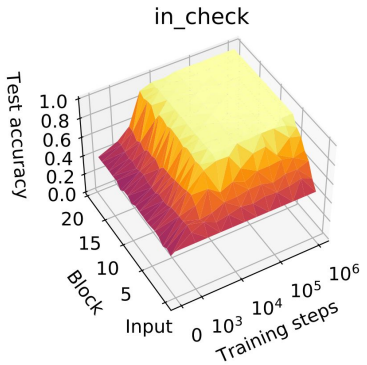
If you're interested, I recommend checking out the arXiv version (2111.09259)

Acquisition of chess knowledge in AlphaZero

Thomas McGrath, Andrei Kapishnikov, Nenad Tomašev, Adam Pearce, Martin Wattenberg, Demis Hassabis, Been Kim, Ulrich Paquet, and Vladimir Kramnik -5 Authors Info & Affiliations

Edited by David Donoho, Stanford University, Stanford, CA; received April 18, 2022; accepted September 19, 2022

November 14, 2022 | 119 (47) e2206625119 | https://doi.org/10.1073/pnas.2206625119



(c) Is the playing side in check?

In my opinion, this is one of the coolest XAI papers ever.

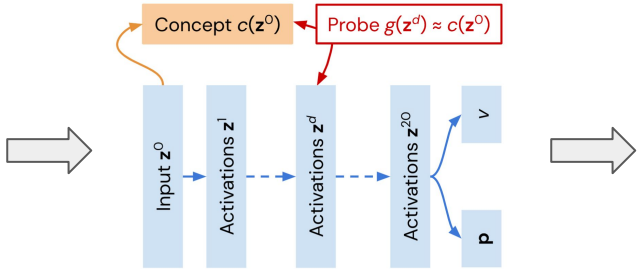
Also, do you recognize any of the authors? :)

Concepts in AlphaZero, step 1: define concepts

Concept names	Description
pawn_fork [m|o]	True if a pawn is attacking two pieces of higher value (knight, bishop, rook, queen, or king) and is not pinned.
knight_fork [m|o]	True if a knight is attacking two pieces of higher value (rook, queen, or king) and is not pinned.
bishop_fork [m|o]	True if a bishop is attacking two pieces of higher value (rook, queen, or king) and is not pinned.
rook_fork [m|o]	True if a rook is attacking two pieces of higher value (queen, or king) and is not pinned.
has_pinned_pawn [m|o]	True if the side has a pawn that is pinned to the king of the same colour.
has_pinned_knight [m|o]	True if the side has a knight that is pinned to the king of the same colour.
has_pinned_bishop [m|o]	True if the side has a bishop that is pinned to the king of the same colour.
has_pinned_rook [m|o]	True if the side has a rook that is pinned to the king of the same colour.
has_pinned_queen [m|o]	True if the side has a queen that is pinned to the king of the same colour.
material [m|o|diff]	Material calculated as (#♙) + 3 * (#♘) + 3 * (#♗) + 5 * (#♖) + 9 * (#♕)
num_pieces [m|o|diff]	Number of pieces that a side has.
in_check	True if the side that makes a turn is in check.
has_bishop_pair [m|o]	True if the side has a pair of bishops.
has_connected_rooks [m|o]	True if the side has connected rooks.
has_control_of_open_file [m|o]	True if the side controls an open file (with the rooks, queen)
has_mate_threat	True if the opponent could mate the current side in a single move if the turn was passed to the opponent.
has_check_move [m|o]	True if the side can check the opponent's King.
can_capture_queen [m|o]	True if the side can capture the opponent's queen.
num_king_attacked_squares [m|o|diff]	The number of squares around the opponent's king that the playing side attacks. Can include occupied squares.
has_contested_open_file	True if an open file is occupied simultaneously by a rook and/or queen of both colours.
has_right_bc_ha_promotion [m|o]	True if 1) the side has a passed pawn on a or h files and 2) the side has a bishop that is of the colour of the promotion square of that pawn.
num_scb_pawns_same_side [m|o|diff]	The number of own pawns that occupy squares of the same colour as the colour of own bishop. Applicable only when the side has a single bishop.
num_ocb_pawns_same_side [m|o|diff]	The number of own pawns that occupy squares of the opposite colour to that of own bishop. Applicable only when the side has a single bishop.
num_scb_pawns_other_side [m|o|diff]	The number of opponent's pawns that occupy the squares of the same colour as the colour of own bishop. Applicable only when the side has a single bishop.
num_ocb_pawns_other_side [m|o|diff]	The number of opponent's pawns that occupy the squares of the opposite colour to the colour of own bishop. Applicable only when the side has a single bishop.
capture_possible_on_{sq} [m|o] sq=[d1|d2|d3|e1|e2|e3|g5|b5]	True is the side can capture a piece on the given square. The squares are named as if the side were playing White.
capture_happens_next_move_... ...on_{sq} sq=[d1|d2|d3|e1|e2|e3|g5|b5]	True if the capture of a piece on the given square had happened according to the game data. The squares are named as if the side were playing White.

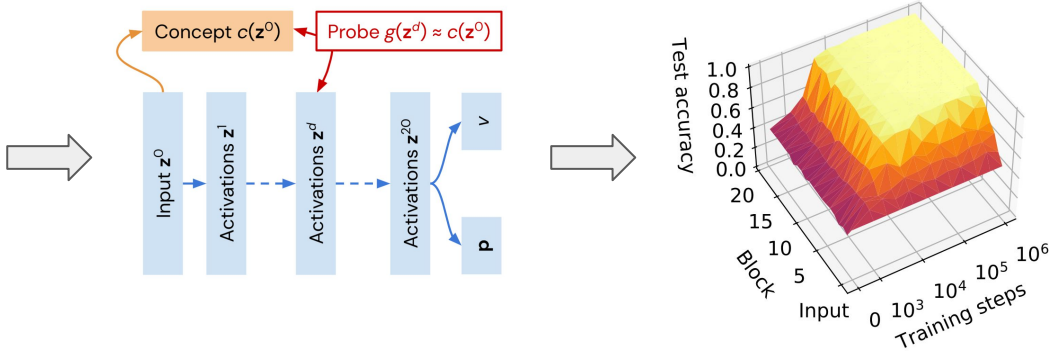
Concepts in AlphaZero, step 2: extract activations for concept data

Concept names	Description
pawn_fork [m|o]	True if a pawn is attacking two pieces of higher value (knight, bishop, rook, queen, or king) and is not pinned.
knight_fork [m|o]	True if a knight is attacking two pieces of higher value (rook, queen, or king) and is not pinned.
bishop_fork [m|o]	True if a bishop is attacking two pieces of higher value (rook, queen, or king) and is not pinned.
rook_fork [m|o]	True if a rook is attacking two pieces of higher value (queen, or king) and is not pinned.
has_pinned_pawn [m|o]	True if the side has a pawn that is pinned to the king of the same colour.
has_pinned_knight [m|o]	True if the side has a knight that is pinned to the king of the same colour.
has_pinned_bishop [m|o]	True if the side has a bishop that is pinned to the king of the same colour.
has_pinned_rook [m|o]	True if the side has a rook that is pinned to the king of the same colour.
has_pinned_queen [m|o]	True if the side has a queen that is pinned to the king of the same colour.
material [m|o|diff]	Material calculated as (#♙) + 3 * (#♘) + 3 * (#♗) + 5 * (#♖) + 9 * (#♕)
num_pieces [m|o|diff]	Number of pieces that a side has.
in_check	True if the side that makes a turn is in check.
has_bishop_pair [m|o]	True if the side has a pair of bishops.
has_connected_rooks [m|o]	True if the side has connected rooks.
has_control_of_open_file [m|o]	True if the side controls an open file (with the rooks, queen)
has_mate_threat	True if the opponent could mate the current side in a single move if the turn was passed to the opponent.
has_check_move [m|o]	True if the side can check the opponent's King.
can_capture_queen [m|o]	True if the side can capture the opponent's queen.
num_king_attacked_squares [m|o|diff]	The number of squares around the opponent's king that the playing side attacks. Can include occupied squares.
has_contested_open_file	True if an open file is occupied simultaneously by a rook and/or queen of both colours.
has_right_bc_ha_promotion [m|o]	True if 1) the side has a passed pawn on a or h files and 2) the side has a bishop that is of the colour of the promotion square of that pawn.
num_scb_pawns_same_side [m|o|diff]	The number of own pawns that occupy squares of the same colour as the colour of own bishop. Applicable only when the side has a single bishop.
num_ocb_pawns_same_side [m|o|diff]	The number of own pawns that occupy squares of the opposite colour to that of own bishop. Applicable only when the side has a single bishop.
num_scb_pawns_other_side [m|o|diff]	The number of opponent's pawns that occupy the squares of the same colour as the colour of own bishop. Applicable only when the side has a single bishop.
num_ocb_pawns_other_side [m|o|diff]	The number of opponent's pawns that occupy the squares of the opposite colour to the colour of own bishop. Applicable only when the side has a single bishop.
capture_possible_on_{sq} [m|o] sq=[d1|d2|d3|e1|e2|e3|g5|b5]	True is the side can capture a piece on the given square. The squares are named as if the side were playing White.
capture_happens_next_move_... ...on_{sq} sq=[d1|d2|d3|e1|e2|e3|g5|b5]	True if the capture of a piece on the given square had happened according to the game data. The squares are named as if the side were playing White.



Concepts in AlphaZero, step 3: train logistic regressor and check accuracy

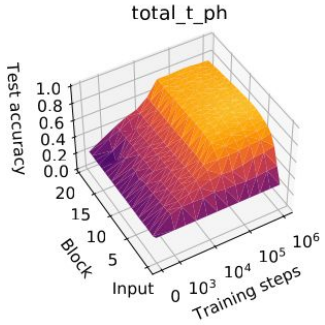
Concept names	Description
pawn_fork [m|o]	True if a pawn is attacking two pieces of higher value (knight, bishop, rook, queen, or king) and is not pinned.
knight_fork [m|o]	True if a knight is attacking two pieces of higher value (rook, queen, or king) and is not pinned.
bishop_fork [m|o]	True if a bishop is attacking two pieces of higher value (rook, queen, or king) and is not pinned.
rook_fork [m|o]	True if a rook is attacking two pieces of higher value (queen, or king) and is not pinned.
has_pinned_pawn [m|o]	True if the side has a pawn that is pinned to the king of the same colour.
has_pinned_knight [m|o]	True if the side has a knight that is pinned to the king of the same colour.
has_pinned_bishop [m|o]	True if the side has a bishop that is pinned to the king of the same colour.
has_pinned_rook [m|o]	True if the side has a rook that is pinned to the king of the same colour.
has_pinned_queen [m|o]	True if the side has a queen that is pinned to the king of the same colour.
material [m|o|diff]	Material calculated as (#♙) + 3 * (#♘) + 3 * (#♗) + 5 * (#♖) + 9 * (#♕)
num_pieces [m|o|diff]	Number of pieces that a side has.
in_check	True if the side that makes a turn is in check.
has_bishop_pair [m|o]	True if the side has a pair of bishops.
has_connected_rooks [m|o]	True if the side has connected rooks.
has_control_of_open_file [m|o]	True if the side controls an open file (with the rooks, queen)
has_mate_threat	True if the opponent could mate the current side in a single move if the turn was passed to the opponent.
has_check_move [m|o]	True if the side can check the opponent's King.
can_capture_queen [m|o]	True if the side can capture the opponent's queen.
num_king_attacked_squares [m|o|diff]	The number of squares around the opponent's king that the playing side attacks. Can include occupied squares.
has_contested_open_file	True if an open file is occupied simultaneously by a rook and/or queen of both colours.
has_right_bc_ha_promotion [m|o]	True if 1) the side has a passed pawn on a or h files and 2) the side has a bishop that is of the colour of the promotion square of that pawn.
num_scb_pawns_same_side [m|o|diff]	The number of own pawns that occupy squares of the same colour as the colour of own bishop. Applicable only when the side has a single bishop.
num_ocb_pawns_same_side [m|o|diff]	The number of own pawns that occupy squares of the opposite colour to that of own bishop. Applicable only when the side has a single bishop.
num_scb_pawns_other_side [m|o|diff]	The number of opponent's pawns that occupy the squares of the same colour as the colour of own bishop. Applicable only when the side has a single bishop.
num_ocb_pawns_other_side [m|o|diff]	The number of opponent's pawns that occupy the squares of the opposite colour to the colour of own bishop. Applicable only when the side has a single bishop.
capture_possible_on_{sq} [m|o] sq=[d1|d2|d3|e1|e2|e3|g5|b5]	True is the side can capture a piece on the given square. The squares are named as if the side were playing White.
capture_happens_next_move_... ...on_{sq} sq=[d1|d2|d3|e1|e2|e3|g5|b5]	True if the capture of a piece on the given square had happened according to the game data. The squares are named as if the side were playing White.



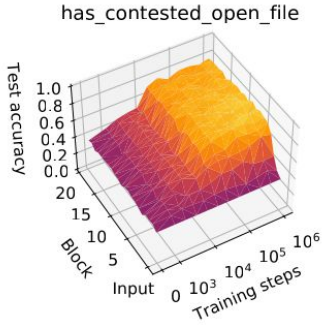
(c) Is the playing side in check?

norwegian open ai lab NTNU

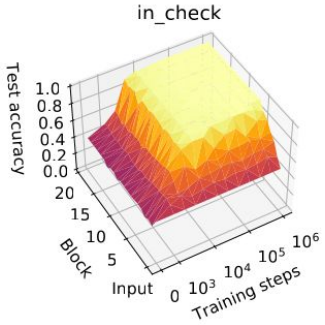
Concept activation vectors in AlphaZero



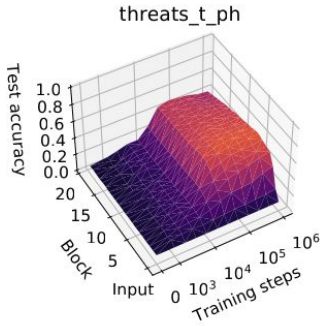
(a) Stockfish 8's total score



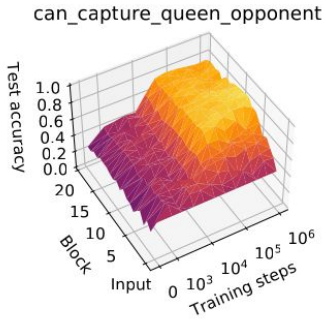
(b) A contested open file is occupied by rooks and/or queens of opposite colours



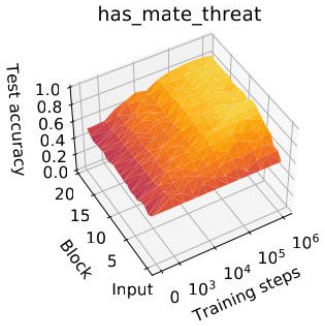
(c) Is the playing side in check?



(d) Stockfish 8's evaluation of threats.



(e) Can the playing side capture their opponent's queen?



(f) Could the opposing side checkmate the playing side in one move?

Usefulness of probe accuracy

Knowing concept directions in a model's activation space isn't a **quantitative** explanation, and we have to keep working a bit on the CAVs before we're happy.

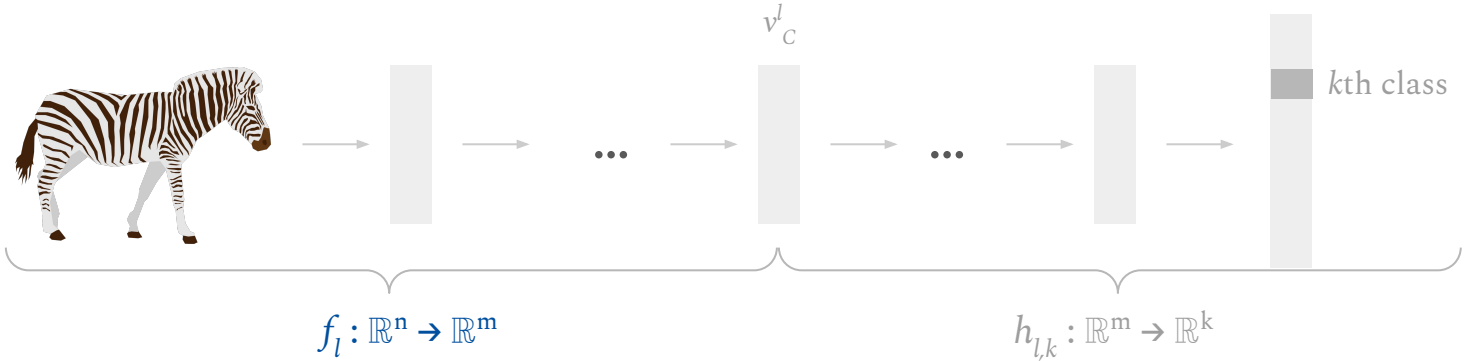
However, CAVs can be useful in themselves.

The accuracy of the logistic regression model producing the CAV tells us how accessible the concept is in the internal representation of the model.

We can interpret the accuracy as an "importance of the concept, according to the model".

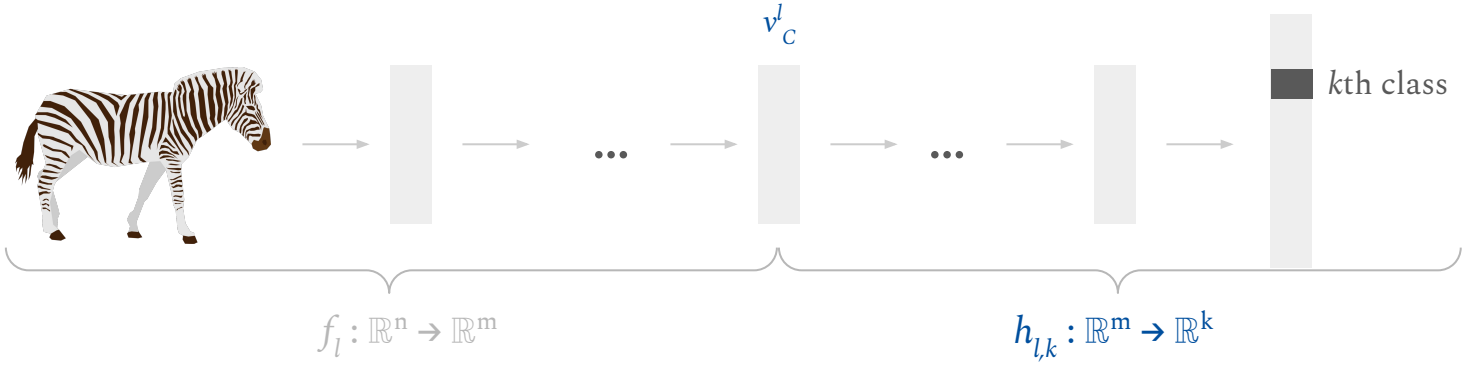
Concept based explanations, next steps

We have found the direction in our model's activation space representing the concept of interest (stripes, in this case).



Concept based explanations, next steps

We have found the direction in our model's activation space representing the concept of interest (stripes, in this case). Now, we want to use this knowledge, the CAV, to quantify how important the concept is for the model prediction (classification).



TODO Concept based explanations, next steps

For gradient based explanation methods, we used the gradients of the logits, $h_k(x)$ for data point x and class k, with respect to the input features, like a pixel value at some position (w,h):

$$\frac{\partial h_k(x)}{\partial x_{h,w}}$$

This answers the question "how sensitive is the activation of output class k to changes in the value of the pixel at (w,h) ?

With our concepts, we want to do something similar...

Sensitivity of prediction to concept

We want to know the sensitivity of the activation of output class k to changes in the input towards the direction of a concept C, in some layer l. We denote this sensitivity as $S_{C,k,l}(x)$.

Simplified, the sensitivity of class k to concept C represented in layer l, is the derivative of the model prediction with respect to the CAV:

$$S_{C,l,k}(x) = \frac{\partial p_k(x)}{\partial v_C^l}$$

This asks: "How sensitive is the prediction to the concept?"

Sensitivity of prediction to concept

We want to know the sensitivity of the activation of output class k to changes in the input towards the direction of a concept C, in some layer l. We denote this sensitivity as $S_{C,k,l}(x)$.

Since we have represented the concept direction in terms of a CAV in some layer l, this sensitivity can be calculated using the directional derivative

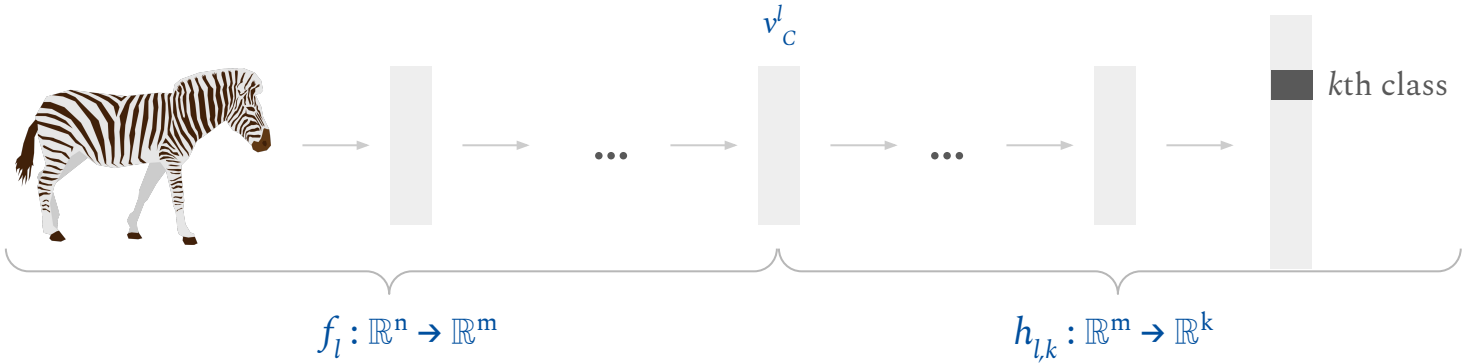
$$\nabla_v f(x) = \lim_{\epsilon \to 0} \frac{f(x + v\epsilon) - f(x)}{\epsilon ||v||}$$

(The directional derivative measures how a function changes as you move in a specific direction from a given point. We'll use this in a second. Don't stress if you don't see how it will be useful.)

Sensitivity of prediction to concept

Remember, the input data, the CAV and the model prediction happen in different layers. Therefore, we have to split up the forward pass into

- the forward pass of the data to layer l, this is $f_l(x)$
- the forward pass of the (modified) activations from layer l to the final layer, this is $h_{l,k}(f_l(x))$



$$\nabla_v f(x) = \lim_{\epsilon \to 0} \frac{f(x + v\epsilon) - f(x)}{\epsilon ||v||}$$

Sensitivity of prediction to concept

Putting it all together, and keeping in mind the expression for the directional derivative, we write the sensitivity $S_{C,k,l}(x)$ of the class prediction k to the direction of a concept C, as represented by a CAV in layer l, for some data point x as

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

= the directional derivative of the function from the CAV layer to the output, in the direction of the CAV.

Why don't we divide by the magnitude of the CAV?

$$\nabla_v f(x) = \lim_{\epsilon \to 0} \frac{f(x + v\epsilon) - f(x)}{\epsilon ||v||}$$

Sensitivity of prediction to concept

Putting it all together, and keeping in mind the expression for the directional derivative, we write the sensitivity $S_{C,k,l}(x)$ of the class prediction k to the direction of a concept C, as represented by a CAV in layer l, for some data point x as

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

We don't divide by the magnitude of the CAV because it is a normal vector (it has unit length).

There's more :)

Sensitivity of prediction to concept

For a differentiable function f, we have

$$\begin{aligned}\nabla_v f(x) &= \lim_{\epsilon \to 0} \frac{f(x + v\epsilon) - f(x)}{\epsilon ||v||} \\ &= \nabla f(x) \frac{v}{||v||}\end{aligned}$$

where ∇ denotes the gradient of f.

In our case, f represents different parts of a trained neural network (depending on which layer we want the CAV for). For these, we typically have the gradients :-D

Sensitivity of prediction to concept

$$\nabla_v f(x) = \lim_{\epsilon \to 0} \frac{f(x + v\epsilon) - f(x)}{\epsilon ||v||}$$
$$= \nabla f(x) \frac{v}{||v||}$$

Step 5: We can calculate the sensitivity $S_{C,k,l}(x)$ of the class prediction k to the direction of a concept C, as represented by a CAV in layer l, for some data point x as

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

$$\nabla_v f(x) = \lim_{\epsilon \to 0} \frac{f(x + v\epsilon) - f(x)}{\epsilon ||v||}$$
$$= \nabla f(x) \frac{v}{||v||}$$

Sensitivity of prediction to concept

Step 5: We can calculate the sensitivity $S_{C,k,l}(x)$ of the class prediction k to the direction of a concept C, as represented by a CAV in layer l, for some data point x as

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

... using the gradients pytorch already knows for the model, to calculate the expression exactly
(see the function `get_grads()` in utils.py)

The sensitivity $S_{C,k,l}(x)$ can take all positive and negative numerical values.

We mostly care about the sign of $S_{C,k,l}(x)$.

Summary: Concept data - layer activations - probe - CAV - TCAV score

Step 1: Make a concept positive data set representing the human-defined concept (in our case stripes) and a concept negative data set representing the concept being absent.

Step 2: Run forward passes on the two data sets, from the input space (dimension n), to some layer l of interest (dimension m). Collect the activations of that layer.

Step 3: Train a linear classifier on the activations of the concept positive and the concept negative data sets. This is the concept probe.

Step 4: Find the normal vector to the decision surface of the logistic regression model (the probe). This is the concept activation vector; the CAV.

Step 5: Calculate the sensitivity the class prediction to the direction of a concept, by multiplying the gradient and the CAV.

Testing with Concept Activation Vectors

We can now design a metric to help us interpret the concept sensitivities globally for some class label *k*.

Let X_k denote all model inputs with class label *k*.

The TCAV score tells us how large a fraction of *k*-class inputs whose sensitivity to concept *C* in layer *l* for predicting class *k* is positive:

$$\text{TCAV}_{Q_{C,k,l}} = \frac{|x \in X_k : S_{C,k,l}(x) > 0|}{|X_k|}$$

Simply put: TCAV quantifies how important a user-defined concept is to a classification result.

For example, how sensitive a prediction of zebra is to the presence of stripes.

Been Kim et al, Proceedings of the 35 th International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018

Code :)

Let's get the gradients for the zebra class from our layer

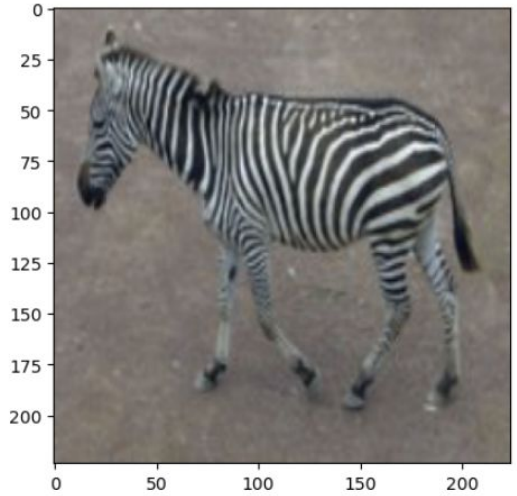
```
def make_gradient_hook(grads):
    def bwd_hook(module, grad_input, grad_output):
        grad = grad_output[0]
        grads.append(grad.squeeze(0).detach().cpu())
    return bwd_hook

resnet.eval()
sample_input = image.unsqueeze(0).to(device)

grads = []
hook = layer.register_full_backward_hook(make_gradient_hook(grads))

resnet.zero_grad()
output = resnet(sample_input)
logits = output[0]

logit = logits[340]
logit.backward()
hook.remove()

print("Number of gradients from layer:", len(grads[0]))
```

zebra is class 340 in imagenet

```
Model output shape: torch.Size([1, 1000])
Number of gradients from layer: 512
```

In imagenet, zebra is class 340

```
image = next(iter(target_loader))[5]
look_at(image)
```



A function for getting the gradients for any class

```
def get_grads(model, data_loader, device, layer, target_idx=None):
      model.eval() # put the model in evaluation mode
      all_grads = [] # initialize list for storing all gradients
      for batch_inputs in data_loader: # for every batch
            for i in range(batch_inputs.size(0)): # for every data point
                  sample_input = batch_inputs[i].unsqueeze(0).to(device) # get data point ready for the model
                  grads = [] # initialize list for storing the gradients of the current pass
                  hook = layer.register_full_backward_hook(make_gradient_hook(grads)) # make a backward hook
                  model.zero_grad() # turn off gradients before the forward pass
                  output = model(sample_input)  # get the model output from a forward pass
                  logits = output[0] # keep the logit (only element in output?)

                  if target_idx is None: # TODO: class index?
                        logit = logits[logits.argmax()] # use as output class the predicted class
                  else:
                        logit = logits[target_idx] # use as output class the provided class
            logit.backward() # do a backward pass from the logit representing the class we're interested in
            hook.remove() # remove the hook :)
            all_grads.append(grads[0]) # add the gradients to the list of all gradients

      return torch.stack(all_grads) # stack the gradients and return as a torch tensor
```

Calculating the TCAV score from the equations

Now that we have the gradients, calculating the sensitivity

$$S_{C,l,k}(x) = \nabla h_{l,k}(f_l(x))v_C^l$$

and the TCAV score

$$\text{TCAV}_{Q_{C,k,l}} = \frac{|x \in X_k : S_{C,k,l}(x) > 0|}{|X_k|}$$

is easy peasy :)

What does the TCAV score count/calculate?

Calculating the TCAV score from the equations

Now that we have the gradients, calculating the sensitivity

$$S_{C,l,k}(x) = \nabla h_{l,k}(f_l(x))v_C^l$$

and the TCAV score

$$\text{TCAV}_{Q_{C,k,l}} = \frac{|x \in X_k : S_{C,k,l}(x) > 0|}{|X_k|}$$

is easy peasy :)

TCAV = the number of (data points x in X_k such that $S_{C,k,l}(x)$ is greater than 0) divided by the size of X_k.

Calculating the TCAV score from the equations

```
def tcav(cav, grads):
     # input: the CAV and gradients of the model

     directional_derivatives = np.dot(grads, cav)

     positive_counts = np.sum(directional_derivatives > 0)

     tcav_score = positive_counts / directional_derivatives.shape[0]

     return tcav_score
```

$$S_{C,l,k}(x) = \nabla h_{l,k}(f_l(x))v_C^l$$

$$\text{TCAV}_{Q_{C,k,l}} = \frac{|x \in X_k : S_{C,k,l}(x) > 0|}{|X_k|}$$

TCAV scores for our stripes CAV

```python
def tcav(cav, grads):

    directional_derivatives = np.dot(grads, cav)
    positive_counts = np.sum(directional_derivatives > 0)
    tcav_score = positive_counts / directional_derivatives.shape[0]

    return tcav_score
```

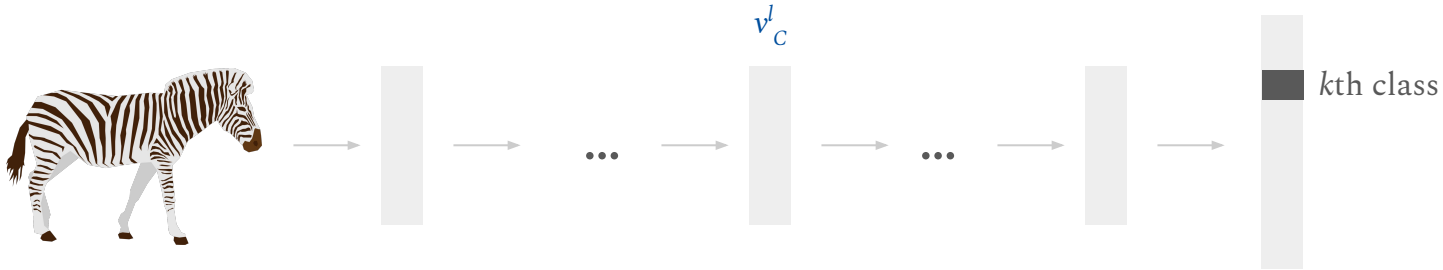
```python
grads = flatten(get_grads(resnet, target_loader, device, layer, target_idx=340))
tcav_score = tcav(stripes_cav, grads)
print(f"TCAV score: {tcav_score}")
```

```
TCAV score: 1.0
```

TCAV - interpretation

We have class k=zebra and concept C=stripes, and get a TCAV score of 1.0

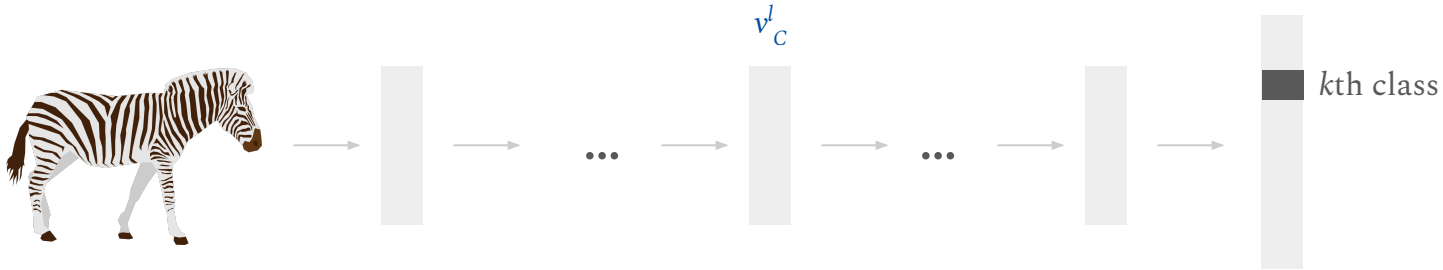
What does this indicate?



TCAV - interpretation

We have class k=zebra and concept C=stripes, and get a TCAV score of 1.0

This indicates that 100% of predictions for the “zebra” class are positively influenced by the presence of stripes in the images (change to more stripes ⇒ classify more zebra).



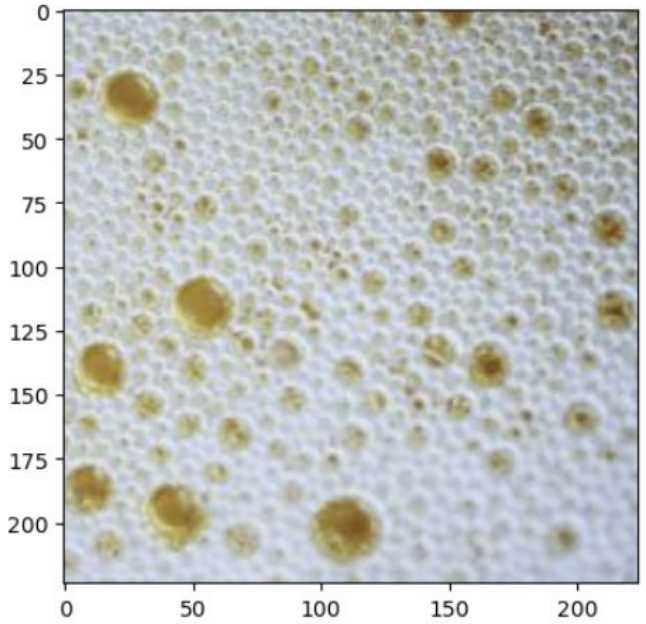
Small sanity check: choose a different concept

Change to concept “bubbly”...

```
batch_size = 16
num_concept_images = 100
num_target_images = 20
target_class = "zebra"
positive_concept = "bubbly" #"striped"
negative_concept = "dotted"
```

```
pos_image = next(iter(positive_loader))[6]
print(pos_image.shape)
look_at(pos_image)
```

```
torch.Size([3, 224, 224])
```



Change to concept “bubbly”...

```
#stripes_cav = train_cav(positive_activations, negative_activations)
bubbly_cav = train_cav(positive_activations, negative_activations)
```

```
Probe accuracy: 1.0
```

The model seems to have represented the concept of “bubbly”

```
grads = flatten(get_grads(resnet, target_loader, device, layer, target_idx=340))
#tcav_score = tcav(stripes_cav, grads)
tcav_score = tcav(bubbly_cav, grads)
print(f"TCAV score: {tcav_score}")
```

```
TCAV score: 0.2
```

What does a TCAV score of 0.2 for “bubbly” mean?

Change to concept “bubbly”...

```
#stripes_cav = train_cav(positive_activations, negative_activations)
bubbly_cav = train_cav(positive_activations, negative_activations)
```

```
Probe accuracy: 1.0
```

The model seems to have represented the concept of “bubbly”

```
grads = flatten(get_grads(resnet, target_loader, device, layer, target_idx=340))
#tcav_score = tcav(stripes_cav, grads)
tcav_score = tcav(bubbly_cav, grads)
print(f"TCAV score: {tcav_score}")
```

```
TCAV score: 0.2
```

but it’s *not* as useful for increasing the classification of zebras;

Moving the activations in the direction of the bubbly CAV in our layer (layer 4) increases the classification to zebras 20% of the time.

Summary: Concept data - layer activations - probe - CAV - TCAV score

Step 1: Make a concept positive data set representing the human-defined concept (in our case stripes) and a concept negative data set representing the concept being absent.

Step 2: Run forward passes on the two data sets, from the input space (dimension n), to some layer l of interest (dimension m). Collect the activations of that layer.

Step 3: Train a linear classifier on the activations of the concept positive and the concept negative data sets. This is the concept probe.

Step 4: Find the normal vector to the decision surface of the logistic regression model (the probe). This is the concept activation vector; the CAV.

Step 5: Calculate the sensitivity the class prediction to the direction of a concept, by multiplying the gradient and the CAV. The TCAV score is the positive frequency of this sensitivity across a data set.

(T)CAV - (Testing with) Concept Activation Vectors

Where do we place this in the taxonomy?

Post-hoc?

Model-agnostic or model-specific?

Local or global?

Post-hoc methods	Model-agnostic	Model-specific
Local		
Global		

(T)CAV - (Testing with) Concept Activation Vectors

Where do we place this in the taxonomy?

Post-hoc: *We got a model and want to explain its behaviour.*

Model-agnostic or model-specific?

Local or global?

Post-hoc methods	Model-agnostic	Model-specific
Local		
Global		

(T)CAV - (Testing with) Concept Activation Vectors

Where do we place this in the taxonomy?

Post-hoc: *We got a model and want to explain its behaviour.*

Model-specific: *We study the activations of a specific model's layer(s).*

Local or global?

Post-hoc methods	Model-agnostic	Model-specific
Local		
Global		

(T)CAV - (Testing with) Concept Activation Vectors

Where do we place this in the taxonomy?

Post-hoc: *We got a model and want to explain its behaviour.*

Model-specific: *We study the activations of a specific model's layer(s).*

Global: *We look for concepts represented by the model's layers using probing data sets.*

Post-hoc methods	Model-agnostic	Model-specific
Local		
Global		

(T)CAV - (Testing with) Concept Activation Vectors

Where do we place this in the taxonomy?

Post-hoc: *We got a model and want to explain its behaviour.*

Model-specific: *We study the activations of a specific model's layer(s).*

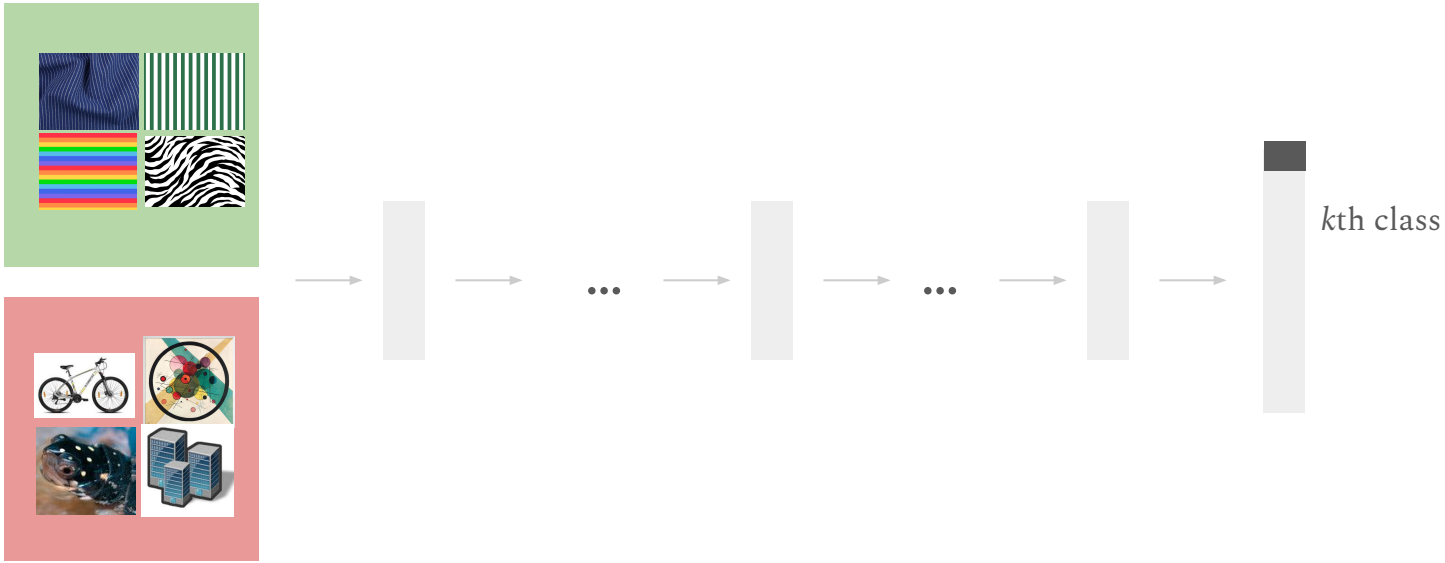
Global: *We look for concepts represented by the model's layers using probing data sets.*

Post-hoc methods	Model-agnostic	Model-specific
Local		
Global		CAV / TCAV

Confession time

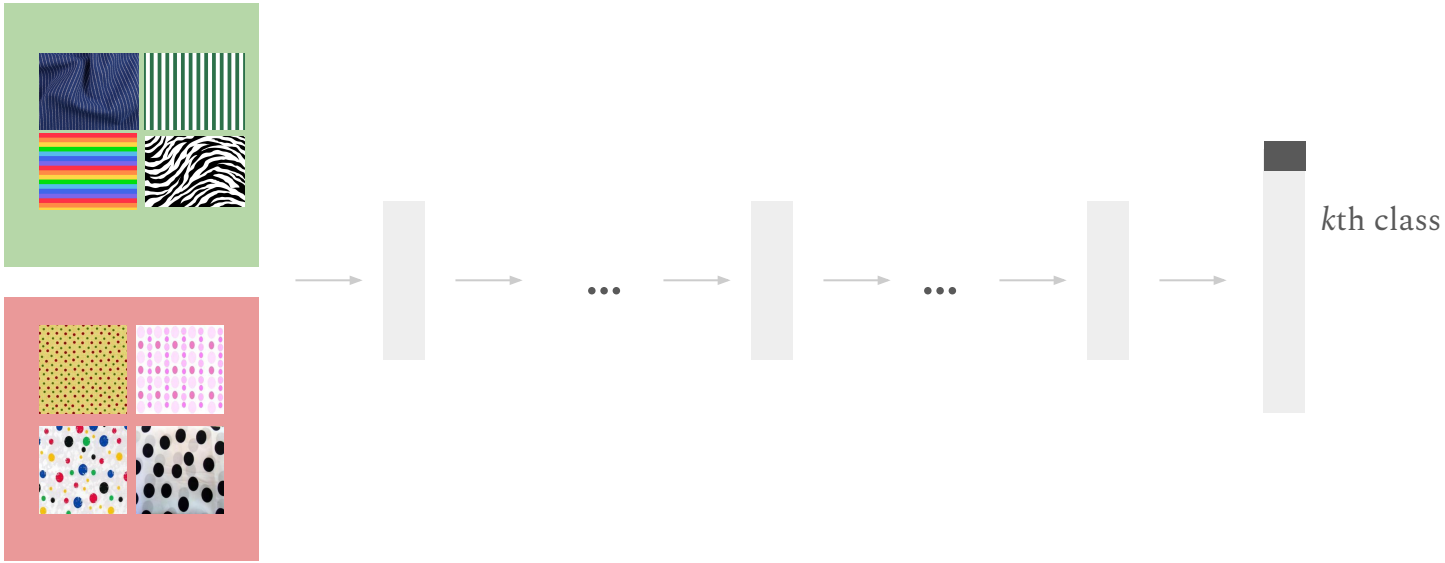
Remember that slide from earlier?

Step 1: Make a concept positive data set representing the human-defined concept, in this case stripes, and a concept negative data set representing the concept being absent.



That slide from earlier...

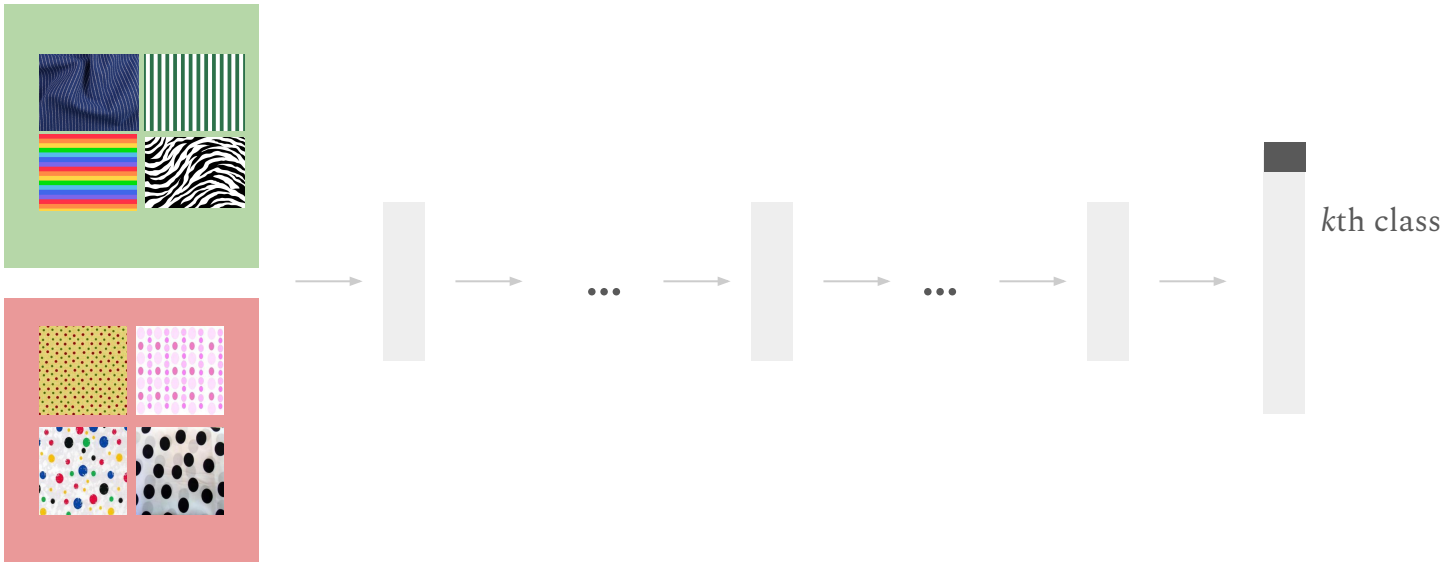
This is not what we've done. Instead of a concept negative dataset with random images, we used only **dotted images**. *What does this imply?*



Relative CAVs

Our positive concept dataset contained stripes, and our negative data dots. So our CAV pointed towards stripes and away from dots. This is a relative CAV.

⇒ Our TCAV score told us how much moving *towards stripes, away from dots*, increased the zebra class.



Relative CAVs

Relative CAVs involve datasets representing different concepts.

Introduced by Been Kim et al (2018) - and used in their experiments.

Relative CAVs are used to measure how relevant a concept (*stripes*) is, *relative to* another well-defined concept (*dotted*).

Especially for images, where we can expect the representation of textures to overlap in the model, one texture (*stripes*) and random images might not be orthogonal (likely even overlapping).

Relative CAVs strip away shared factors and focus the sensitivity measure on what actually distinguishes the concepts.

3.6. TCAV extensions: Relative TCAV

In practice, semantically related concepts (e.g., brown hair vs. black hair) often yield CAVs that are far from orthogonal. This natural, expected property may be beneficially used to make fine-grained distinctions since relative comparisons between related concepts are a good interpretative tool (Kim et al., 2015; Doshi-Velez et al., 2015; Tibshirani, 1994; Salvatore et al., 2014).

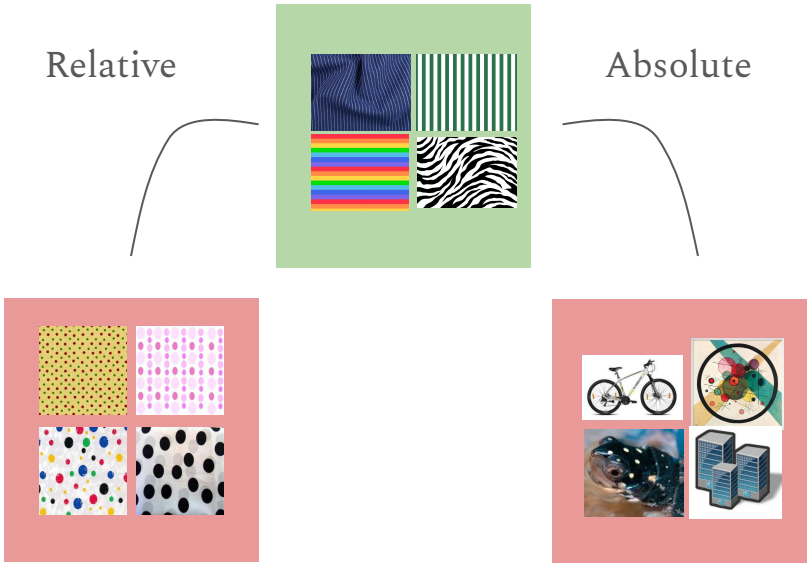
Relative CAVs allow making such fine-grained comparisons. Here the analyst selects two sets of inputs that represent two different concepts, C and D. Training a classifier on $f_l(P_C)$ and $f_l(P_D)$ yields a vector $\boldsymbol{v}^l_{C,D} \in \mathbb{R}^m$. The vector $\boldsymbol{v}^l_{C,D}$ intuitively defines a 1-d subspace in layer l where the projection of an embedding $f_l(\boldsymbol{x})$ along this subspace measures whether $\boldsymbol{x}$ is more relevant to concept C or D.

Relative CAVs may, for example, apply to image recognition, where we can hypothesize that concepts for 'dotted', 'striped', and 'meshed' textures are likely to exist as internal representations, and be correlated or overlapping. Given three positive example sets P_{dot}, P_{stripe}, and P_{mesh}, a relative CAV can be derived by constructing, for each, a negative input set by complement (e.g., $\{P_{\text{dot}} \cup P_{\text{mesh}}\}$ for the stripes).

Absolute vs relative CAVs

Absolute CAVs ask *"does any presence of concept C matter?"*

Relative CAVs ask *"does the presence of concept C matter more than the presence of a different concept C′?"*



Pro tip: Use relative CAVs
to have more control of the CAV direction
("from this to that" instead of "from whatever to that")

Benefits

What do you like?

Benefits

Interpretable: The TCAV score is easily interpretable (how often the concept increases the classification), is not prone to be misunderstood and doesn't require graphs to be conveyed.

Accessible: As long as the concept (data set) is well defined, the TCAV score can be used by domain experts to check whether the model uses concepts known to be relevant in the domain, without requiring ML or data science knowledge.

Diagnostic tool: Model developers can probe for CAVs after and even during training to verify that a model internalises concepts known to be relevant, and thus serve as a diagnostic tool.

Architecture agnostic: CAVs can be generated for any network representation, regardless of architecture, as long as the activations can be extracted from the model's layers.

Challenges

What's painful?

Challenges

Concept data: If the concept data doesn't exist / a function cannot be constructed to label concept data, the process requires additional annotations, which can be expensive, resource intensive and error-prone.

Data type: CAV/TCAV is widely used for image models, while applicability on text/tabular data is not as established.

Linearity assumption: CAV/TCAV assume linear separability of a concept in the activation space. Complex, nonlinear concept manifolds are approximated poorly by linear probes.

Concept definition: Abstract concepts ("sadness") can be hard to represent, and concepts might be ambiguous ("metal" vs "chrome").

Misalignment: CAVs can be spurious or unstable; a CAV trained for "grass" might really capture "green texture". CAV directions are *correlation-based*, and not guaranteed to reflect the intended concept - more on this in the next lecture :)

Tensorflow: The existing Python implementation requires tensorflow - which isn't a huge problem *for you* because we made Pytorch code today.

Homework :)

Using the provided Jupyter Notebooks and python scripts do the following.

Part 1: Make CAVs for all layers in the renset18 model, and plot them. This should be pretty straightforward :)

Part 2: For the misclassified mystery image, located in `data/imagenet/volcano_zebra.jpg`, use the stripes CAV to find out what went wrong.

Hint: we got a TCAV score of 1.0 for the stripes CAV, meaning that stripes contribute to the classification of zebra. You can use the code provided to get model activations for the mystery image.

Which direction should these activations point in, if the model finds the important stripes concept in the image?

Discussion

We would like to ensure that our neural network

1. Uses knowledge we know to be relevant
2. Is aligned with our values

How can concept based explanations be used to achieve this?

Have fun experimenting with your first model agnostic XAI method

… which is widely used, adaptable and offers a lot of research opportunities.