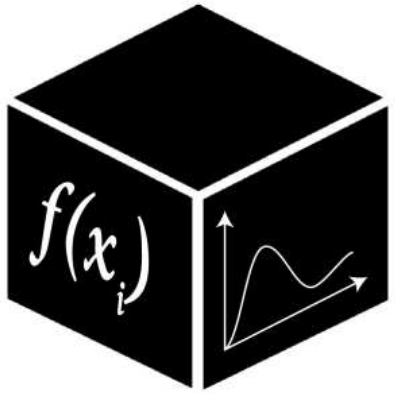
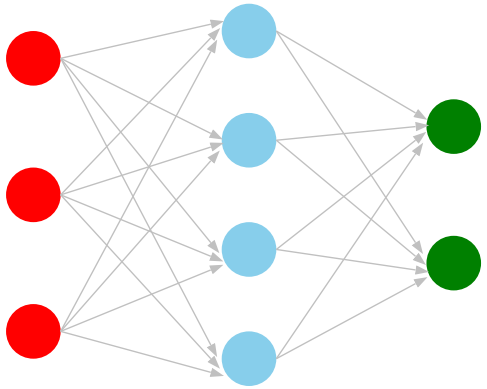


# COM SCI M148, Lecture 17:
# Interpreting Neural Networks

Prof. Nathan Szymanski
nszymanski@seas.ucla.edu
Department of Materials Science

UCLA

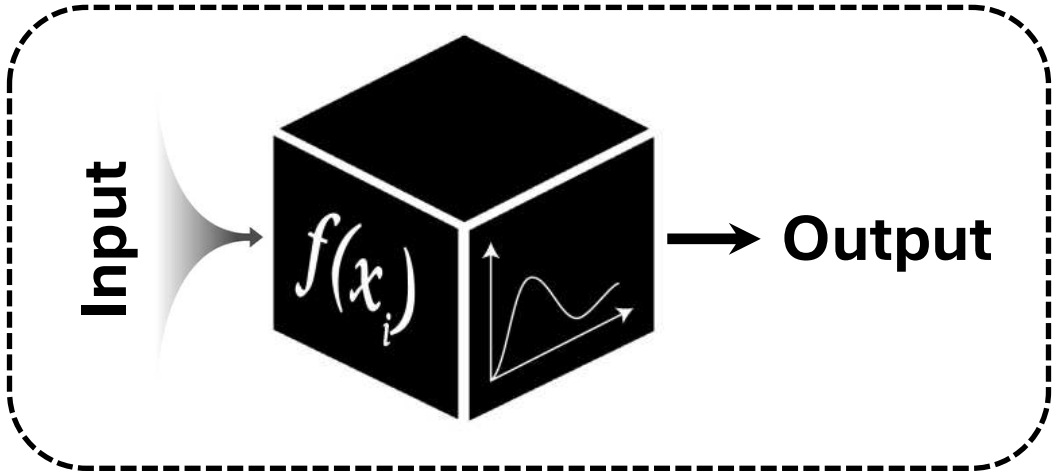
Samueli
School of Engineering

# From last time

| | Model-specific | Model-agnostic |
|---|---|---|
| **Global** | Regression coefficients ($\beta_i$)<br>Odds ratios ($e^{\beta_i}/e^{\beta_j}$)<br>Feature importance based on information gain in trees | Partial dependence, permutation importance, and ICE curves |
| **Local** | One calculation using $\beta_i$<br>One "path" down a tree<br>**Today: saliency and class activation maps for NNs** | LIME and SHAP |

# Black-box *vs.* NN-specific explanations

With LIME and SHAP: **perturb the input and see how output changes**

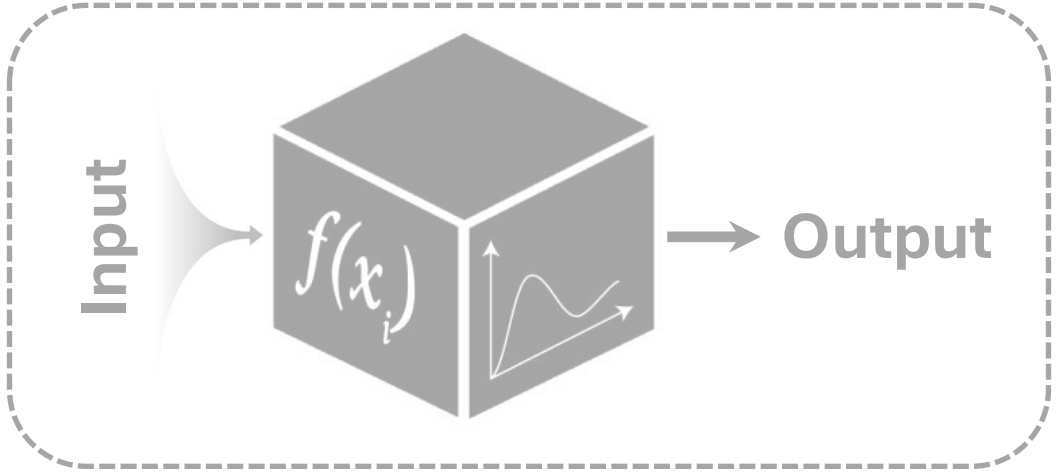



**This is essentially a derivative: $\partial f / \partial x_i$**

These methods apply to ***any* model**

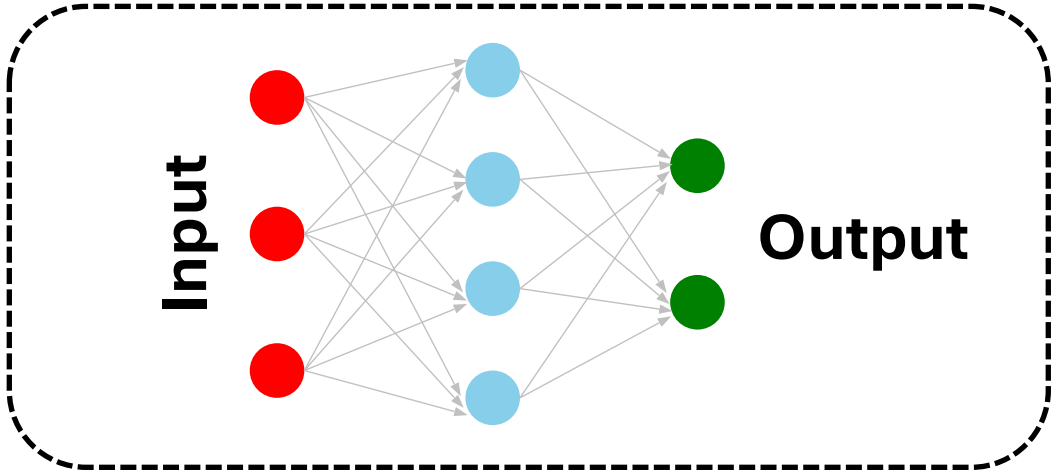
# Black-box *vs.* NN-specific explanations

With LIME and SHAP: perturb the input and see how output changes




These methods apply to ***any*** **model**

But if we *know* the model is a NN, we **can directly compute derivatives**




These methods apply only to ***differentiable*** **models** (including but not limited *to* neural networks)

# Black-box *vs.* NN-specific explanations

Think about why the derivatives are useful for interpretation by considering **linear regression:**

$$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2$$

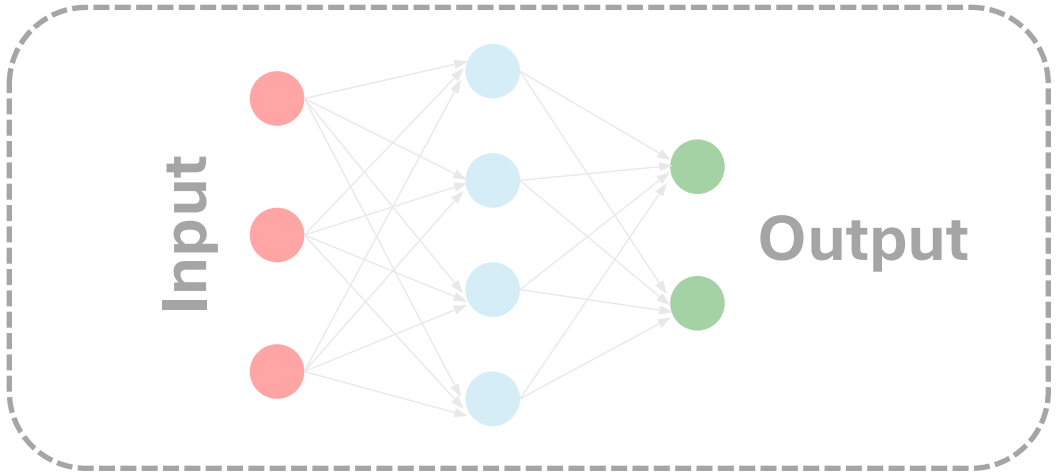
$$\frac{\partial \hat{y}}{\partial x_2} = \beta_2$$

If we differentiate the output with respect to one feature, we retrieve its coefficient

**Larger derivative → larger $\beta$**
**→ feature has more influence**

Can we obtain something similar with NNs?

But if we *know* the model is a NN, we can directly compute derivatives




These methods apply only to ***differentiable*** **models** (including *but not limited to* neural networks)

# Saliency Maps

Use backprop to determine which features most strongly affect the output

# Saliency in the context of NNs

For a trained neural network, we want to know: **Which input features most strongly influence the model's prediction?**

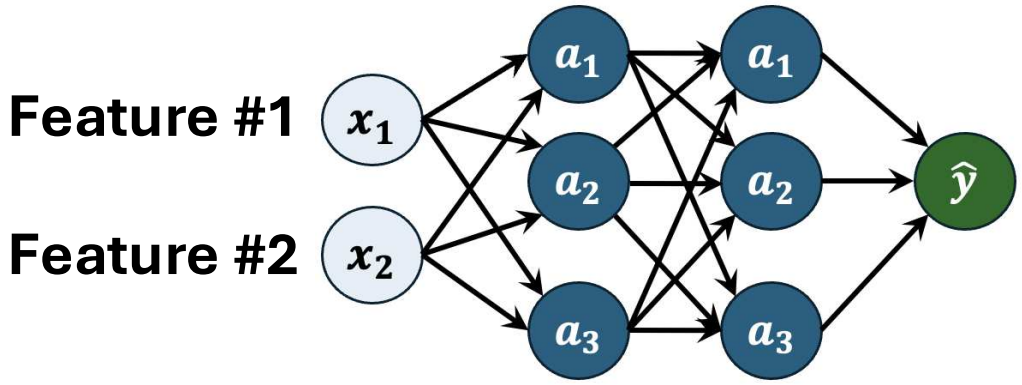
**salient** adjective

formal : very important or noticeable

- the *salient* facts
- a *salient* characteristic/feature

The **saliency** of feature $i$ in model $f$ for sample $x_0$ is:

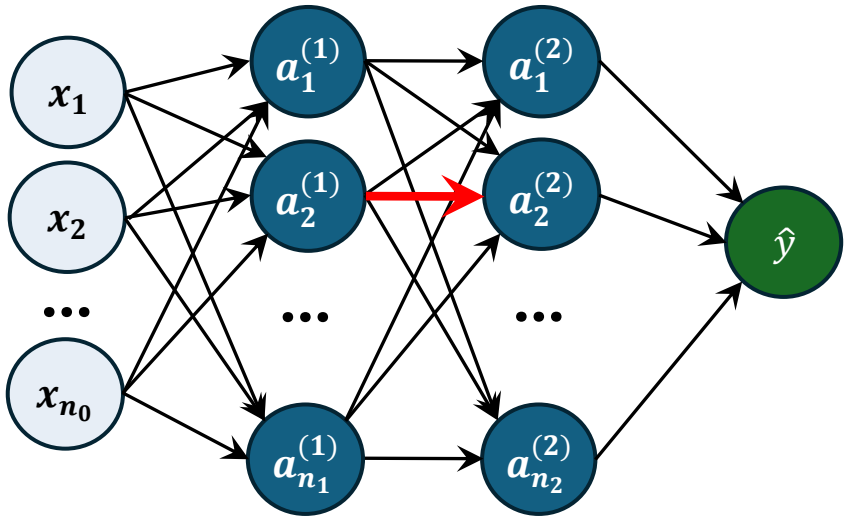
$$S_i = \left| \frac{\partial f}{\partial x_i} \right|_{x=x_0}$$




Suppose Feature #1 has large saliency for a given sample.* Then small changes to $x_1$ will lead to large changes in the output.

*Saliency is local, not global!

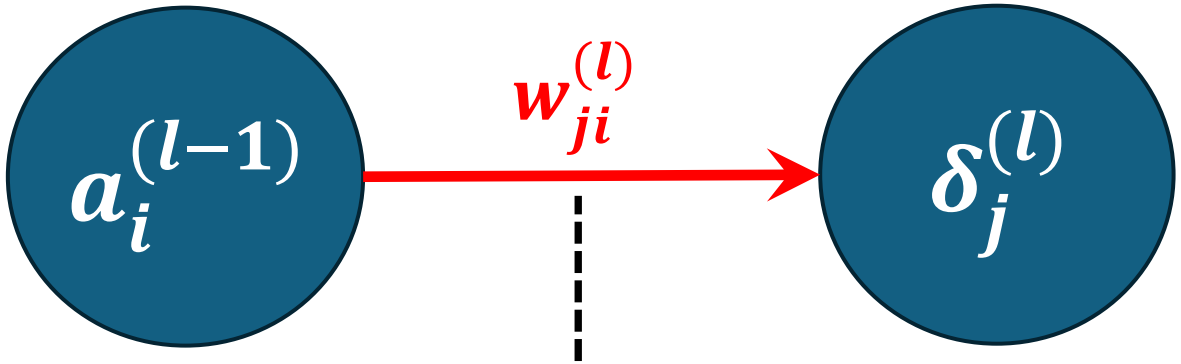
# Recall: Backpropagation




**Use error signal to calculate parameter gradients:**

$$\frac{\partial \mathcal{L}}{\partial w_{ji}^{(l)}} = \delta_j^{(l)} a_i^{(l-1)}$$

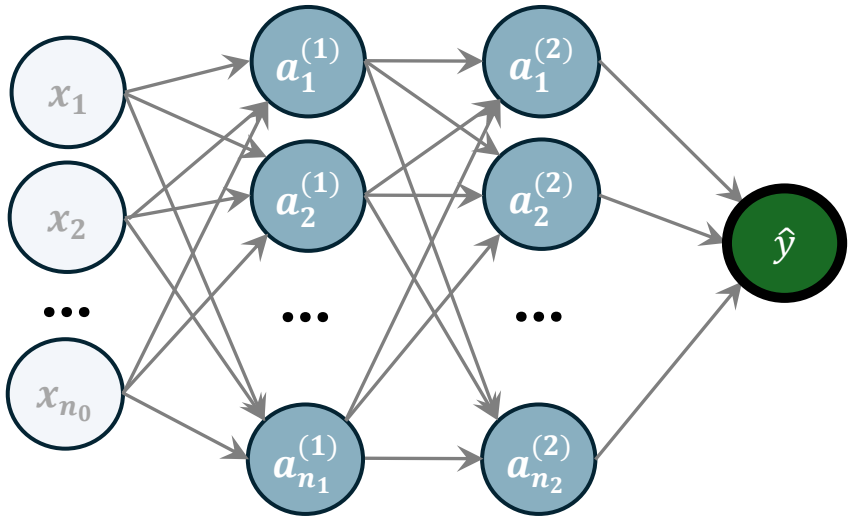
$$\frac{\partial \mathcal{L}}{\partial b_j^{(l)}} = \delta_j^{(l)}$$




What is the size of the input that feeds into the next neuron

How much this neuron contributes to the final prediction error

# Recall: Backpropagation




***Backward pass***

**Use error signal to calculate parameter gradients:**

$$\frac{\partial \mathcal{L}}{\partial w_{ji}^{(l)}} = \delta_j^{(l)} a_i^{(l-1)}$$

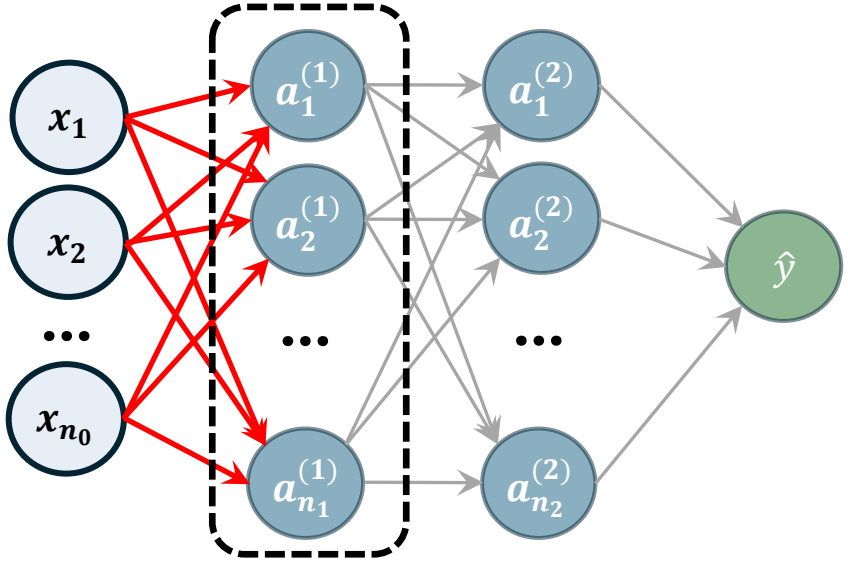
$$\frac{\partial \mathcal{L}}{\partial b_j^{(l)}} = \delta_j^{(l)}$$

Calculate error signal at the output, then **propagate the error backward:**

$$\delta_j^{(l)} = g'\left(z_j^{(l)}\right) \sum_k w_{kj}^{(l+1)} \delta_k^{(l+1)}$$

**Do this over and over until we reach the inputs, computing error signals along the way**

# Now: Backpropagation one step further



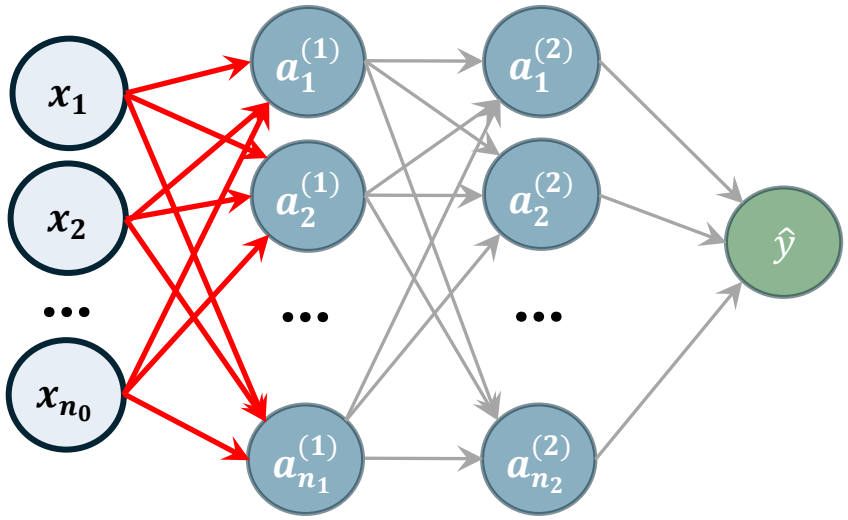

Usually, we calculate error signals back to the 1st hidden layer, then stop there

$$x_i \rightarrow f(x) \rightarrow \mathcal{L}$$

$$\frac{\partial \mathcal{L}}{\partial w_{ji}^{(1)}} = \delta_j^{(1)} x_i \qquad \frac{\partial \mathcal{L}}{\partial b_j^{(1)}} = \delta_j^{(1)}$$

When training neural networks, we go from loss **to the first layer of weights**

# Now: Backpropagation one step further




$$x_i \rightarrow f(x) \rightarrow \mathcal{L}$$

$$\frac{\partial \mathcal{L}}{\partial w_{ji}^{(1)}} = \delta_j^{(1)} \, x_i$$

$$\frac{\partial \mathcal{L}}{\partial b_j^{(1)}} = \delta_j^{(1)}$$

When training neural networks, we go from loss **to the first layer of weights**

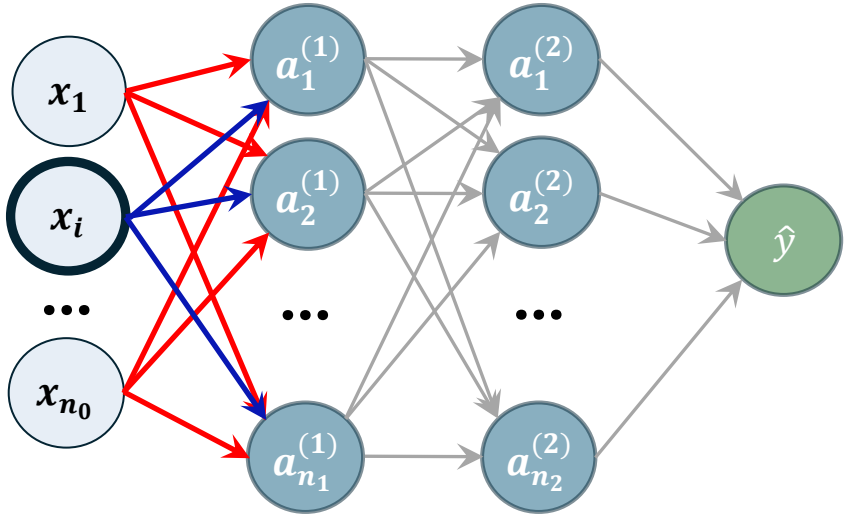
The **saliency** of feature $i$ in model $f$ for sample $x_0$ is:

$$S_i = \left| \frac{\partial f}{\partial x_i} \right|_{x=x_0}$$

**To get saliency, we go back one step further: from $f(x)$ to $x_i$**

$$x_i \rightarrow f(x)$$

# Saliency calculation




$$x_i \rightarrow z_j^{(1)} \rightarrow a_j^{(1)} \rightarrow \cdots \rightarrow f(x)$$

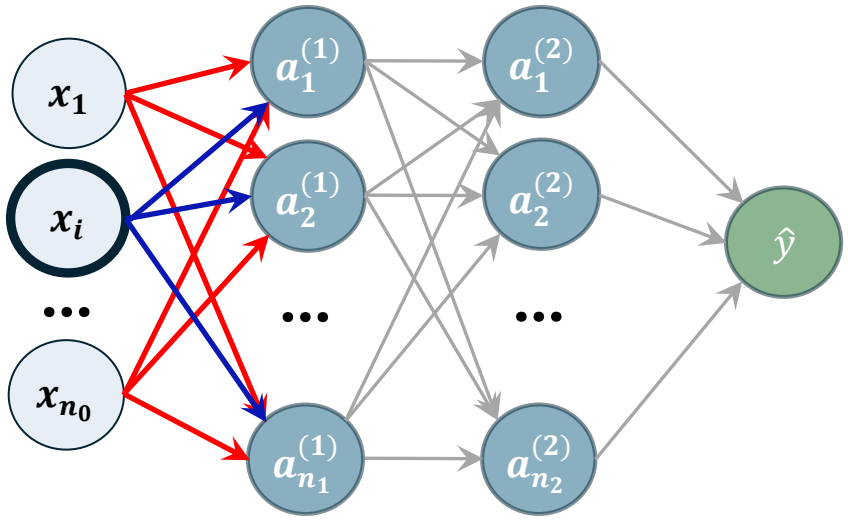
$$z_j^{(1)} = \sum_i w_{ji}^{(1)} x_i + b_j^{(1)} \qquad a_j^{(1)} = g\left(z_j^{(1)}\right)$$

We want the derivative of the output with respect to $x_i$ - let's use our old friend: **chain rule!**

$$\frac{\partial f}{\partial x_i} = \sum_j \frac{\partial f}{\partial z_j^{(1)}} \frac{\partial z_j^{(1)}}{\partial x_i}$$

We sum over all neurons that $x_i$ feeds into

# Saliency calculation




$$x_i \rightarrow z_j^{(1)} \rightarrow a_j^{(1)} \rightarrow \cdots \rightarrow f(x)$$

$$z_j^{(1)} = \sum_i w_{ji}^{(1)} x_i + b_j^{(1)} \qquad a_j^{(1)} = g\left(z_j^{(1)}\right)$$

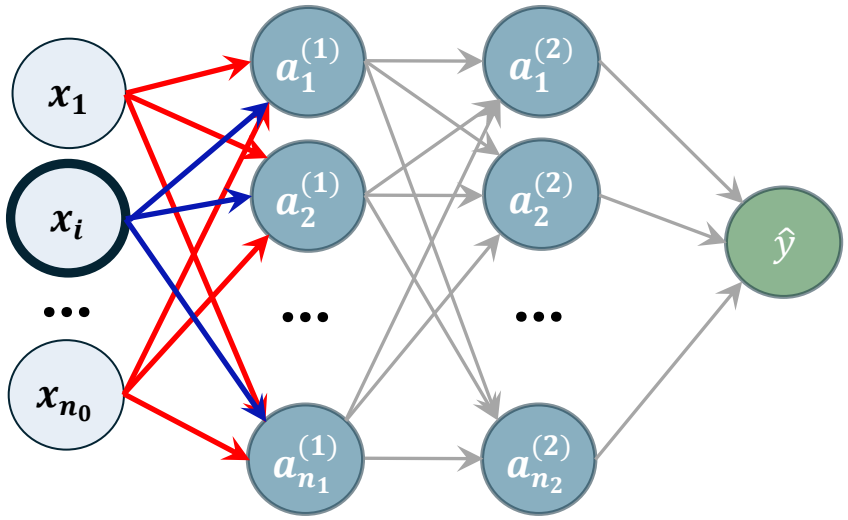
We want the derivative of the output with respect to $x_i$ - let's use our old friend: **chain rule!**

$$\frac{\partial f}{\partial x_i} = \sum_j \frac{\partial f}{\partial z_j^{(1)}} \frac{\partial z_j^{(1)}}{\partial x_i}$$

We sum over all neurons that $x_i$ feeds into

**How the model output is affected by changes to neurons in the first hidden layer**

# Saliency calculation




$$x_i \rightarrow z_j^{(1)} \rightarrow a_j^{(1)} \rightarrow \cdots \rightarrow f(x)$$

$$z_j^{(1)} = \sum_i w_{ji}^{(1)} x_i + b_j^{(1)} \qquad a_j^{(1)} = g\left(z_j^{(1)}\right)$$

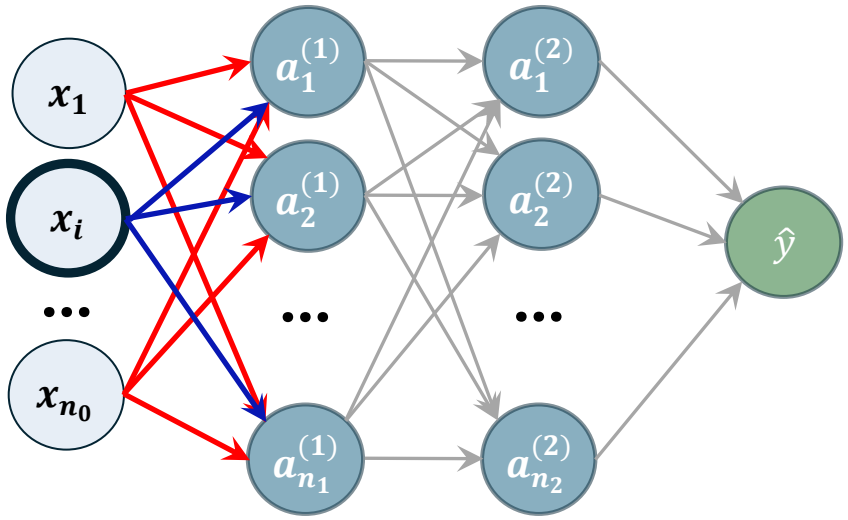
We want the derivative of the output with respect to $x_i$ - let's use our old friend: **chain rule!**

$$\frac{\partial f}{\partial x_i} = \sum_j \frac{\partial f}{\partial z_j^{(1)}} \frac{\partial z_j^{(1)}}{\partial x_i}$$

We sum over all neurons that $x_i$ feeds into

**How those neurons are affected by changes to the input feature**

# Saliency calculation




$$x_i \rightarrow z_j^{(1)} \rightarrow a_j^{(1)} \rightarrow \cdots \rightarrow f(x)$$

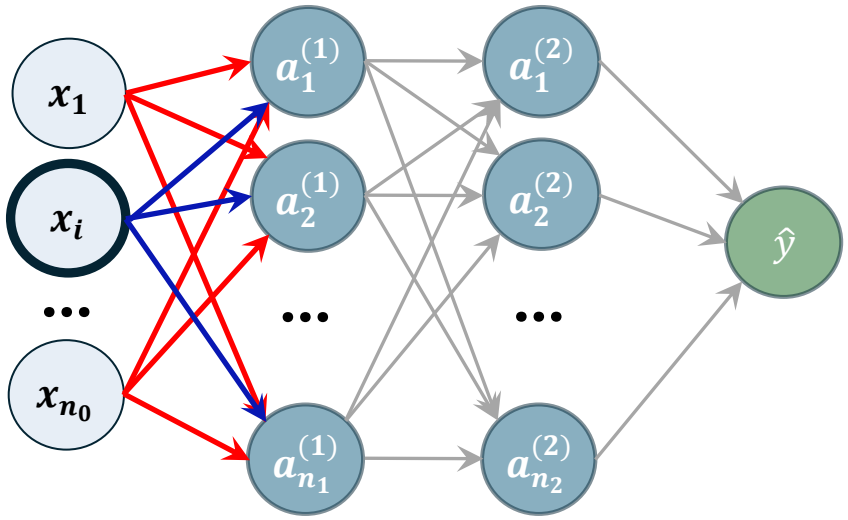
$$z_j^{(1)} = \sum_i w_{ji}^{(1)} x_i + b_j^{(1)} \qquad a_j^{(1)} = g\left(z_j^{(1)}\right)$$

We want the derivative of the output with respect to $x_i$ - let's use our old friend: **chain rule!**

$$\frac{\partial f}{\partial x_i} = \sum_j \frac{\partial f}{\partial z_j^{(1)}} \frac{\partial z_j^{(1)}}{\partial x_i} \rightarrow \frac{\partial z_j^{(1)}}{\partial x_i} = w_{ji}^{(1)}$$

**How those neurons are affected by changes to the input feature**

# Saliency calculation



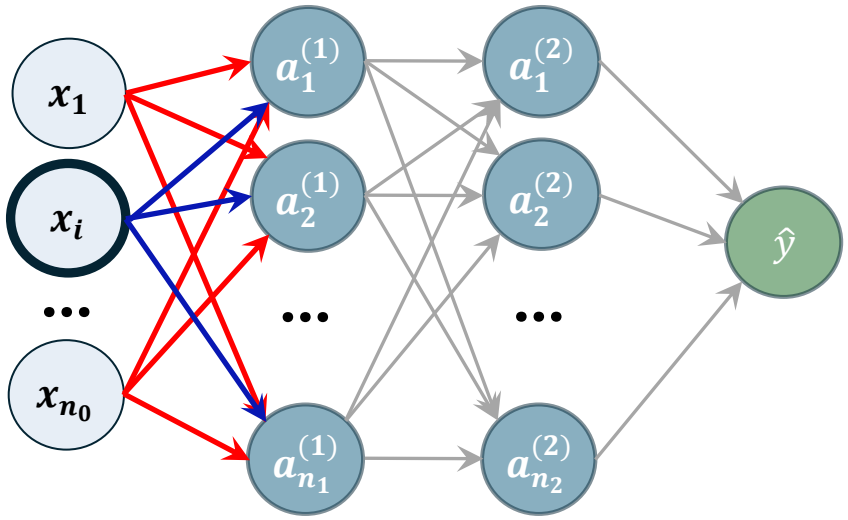

$$\frac{\partial f}{\partial x_i} = \sum_j \frac{\partial f}{\partial z_j^{(1)}} w_{ji}^{(1)}$$

Any way to simplify the green term?
Does it look familiar?

$$\frac{\partial f}{\partial x_i} = \sum_j \frac{\partial f}{\partial z_j^{(1)}} \frac{\partial z_j^{(1)}}{\partial x_i}$$

$$\frac{\partial z_j^{(1)}}{\partial x_i} = w_{ji}^{(1)}$$

# Saliency calculation




$$\frac{\partial f}{\partial x_i} = \sum_j \frac{\partial f}{\partial z_j^{(1)}} w_{ji}^{(1)}$$

**Recall the error signal:**
How much each neuron contributes to the loss

$$\delta_j^{(1)} = \frac{\partial \mathcal{L}}{\partial z_j^{(1)}}$$

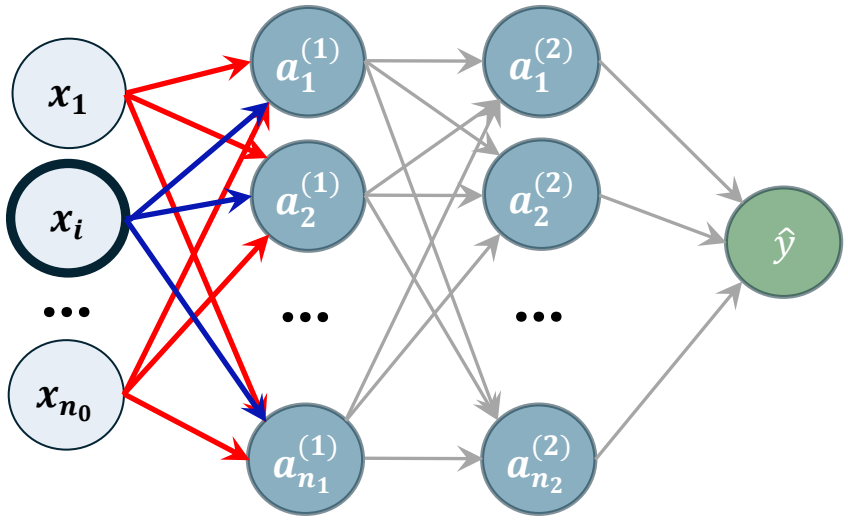
**We start from the output:**

$$\delta_j^{(L)} = g'\left(z_j^{(L)}\right) \frac{\partial \mathcal{L}}{\partial \hat{y}}$$

**And then backpropagate:**

$$\delta_j^{(l)} = g'\left(z_j^{(l)}\right) \sum_k w_{kj}^{(l+1)} \delta_k^{(l+1)}$$

# Saliency calculation




$$\frac{\partial f}{\partial x_i} = \sum_j \frac{\partial f}{\partial z_j^{(1)}} w_{ji}^{(1)}$$

**Recall the error signal:**
How much each neuron contributes to the loss

$$\delta_j^{(1)} = \frac{\partial \mathcal{L}}{\partial z_j^{(1)}}$$

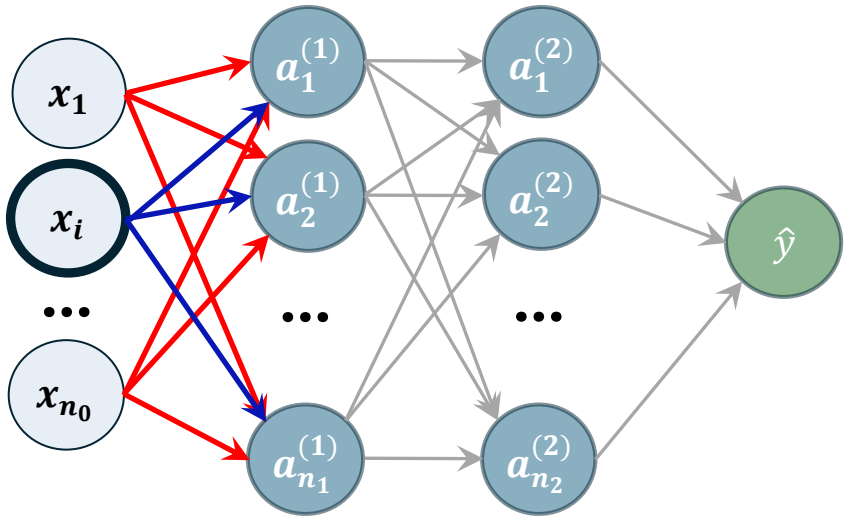
**We start from the output:**

$$\delta_j^{(L)} = g'\left(z_j^{(L)}\right) \frac{\partial f}{\partial \hat{y}}$$

Swap out $\mathcal{L}$ for $f$ to get the **sensitivity gradient**, which is similar to the error signal:

$$\delta_j^{(1)} = \frac{\partial f}{\partial z_j^{(1)}}$$

# Saliency calculation




$$\frac{\partial f}{\partial x_i} = \sum_j \frac{\partial f}{\partial z_j^{(1)}} w_{ji}^{(1)}$$

**Recall the error signal:**
How much each neuron contributes to the loss

$$\delta_j^{(1)} = \frac{\partial \mathcal{L}}{\partial z_j^{(1)}}$$

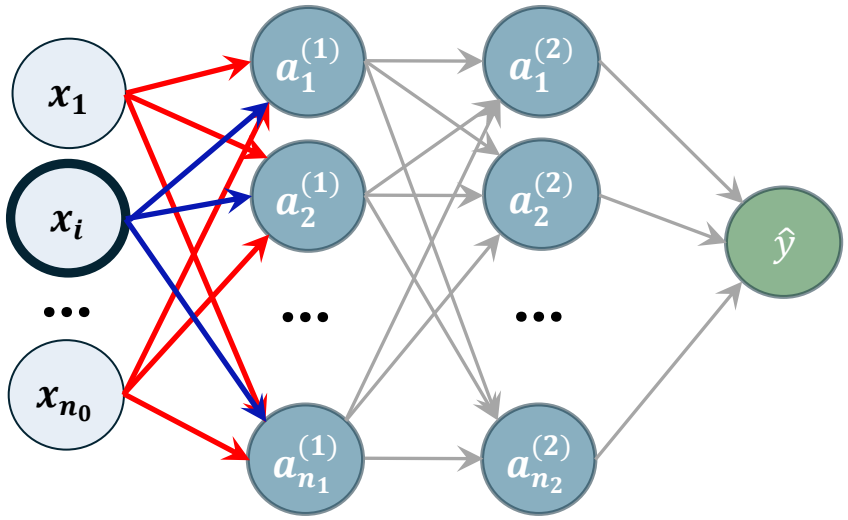
**We start from the output:**

$$\delta_j^{(L)} = g'\left(z_j^{(L)}\right) \frac{\partial f}{\partial \hat{y}}$$

→ **Remember that $\hat{y} = f(x)$ so this derivative is just one!**

$$\delta_j^{(L)} = g'\left(z_j^{(L)}\right)$$

# Saliency calculation




$$\frac{\partial f}{\partial x_i} = \sum_j \delta_j^{(1)} w_{ji}^{(1)}$$

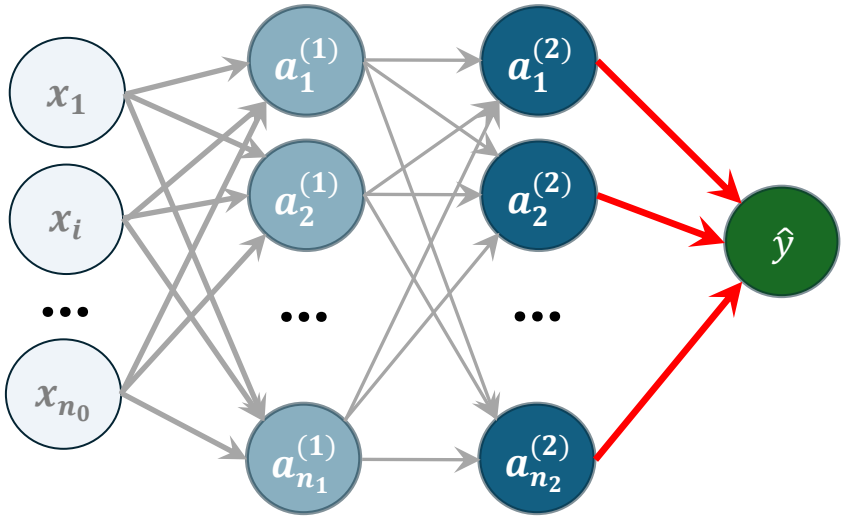
We get this by starting from the output (layer $L$)

$$\delta_j^{(L)} = g'\left(z_j^{(L)}\right)$$

And then backpropagating as usual

$$\delta_j^{(l)} = g'\left(z_j^{(l)}\right) \sum_k w_{kj}^{(l+1)} \delta_k^{(l+1)}$$

# Saliency calculation




How is the output affected by changes to the last hidden layer

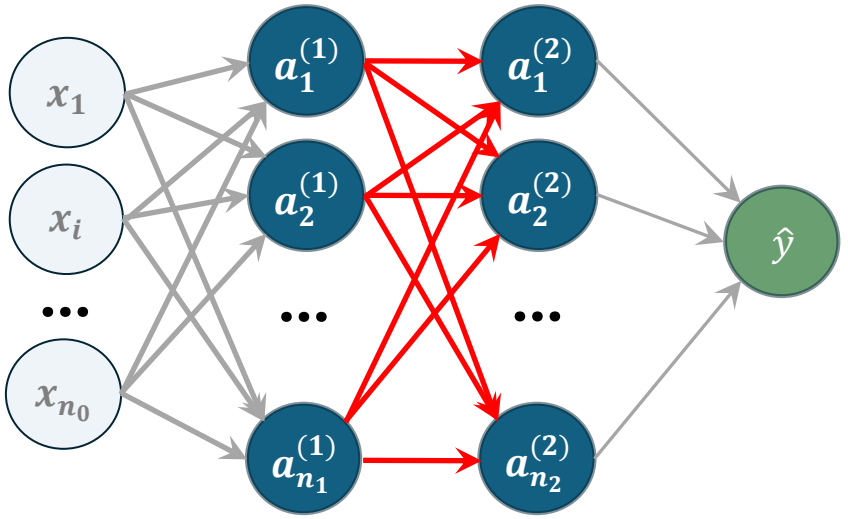
$$\delta_j^{(l)} = g'\left(z_j^{(l)}\right) \sum_k w_{kj}^{(l+1)} \delta_k^{(l+1)}$$

$$\delta_j^{(L)} = g'\left(z_j^{(L)}\right)$$

# Saliency calculation

How is last hidden layer affected by changes to the layer before it

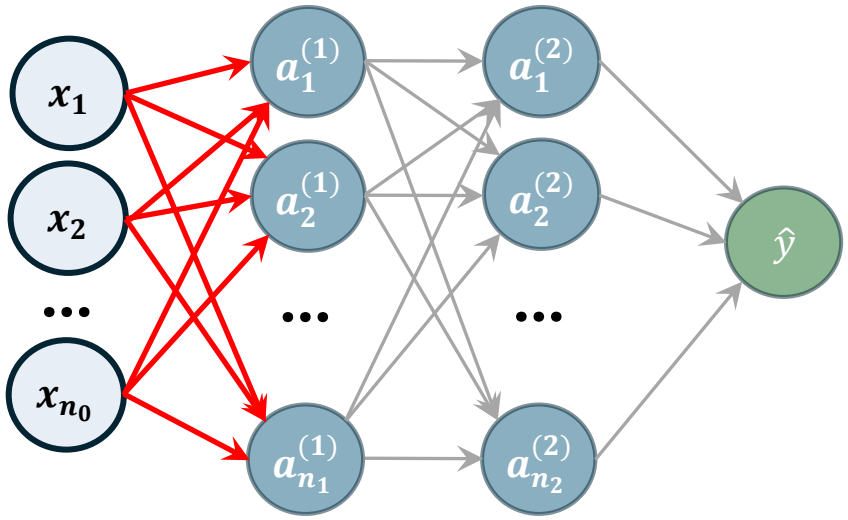
And so on…
Until we reach the input




$$\delta_j^{(l)} = g'\left(z_j^{(l)}\right) \sum_k w_{kj}^{(l+1)} \delta_k^{(l+1)}$$

$$\delta_j^{(L)} = g'\left(z_j^{(L)}\right)$$

# Saliency calculation




$$\frac{\partial f}{\partial x_i} = \sum_j \delta_j^{(1)} w_{ji}^{(1)}$$

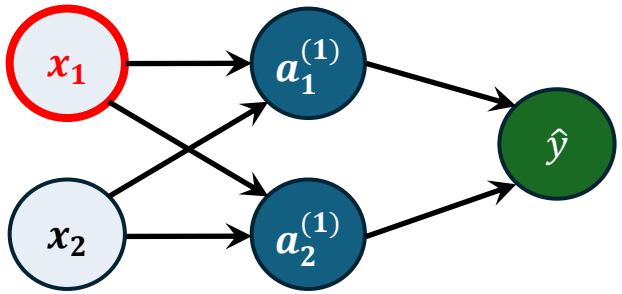
This gives you the gradient for any sample ($x$)

But remember that saliency is a **local** explanation, and we typically care about magnitude not sign (how strongly each input affects the output)

We plug in the values $x = x_0$ for our sample and take the absolute value of the result

$$S_i = \left| \frac{\partial f}{\partial x_i} \right|_{x=x_0} = \left| \sum_j \delta_j^{(1)} (x = x_0) w_{ji}^{(1)} \right|$$

# A simple example




Use linear activation on all neurons

$$z_1^{(1)} = w_{11}^{(1)} x_1 + w_{21}^{(1)} x_2 + b_1^{(1)}$$

$$z_2^{(1)} = w_{12}^{(1)} x_1 + w_{22}^{(1)} x_2 + b_2^{(1)}$$

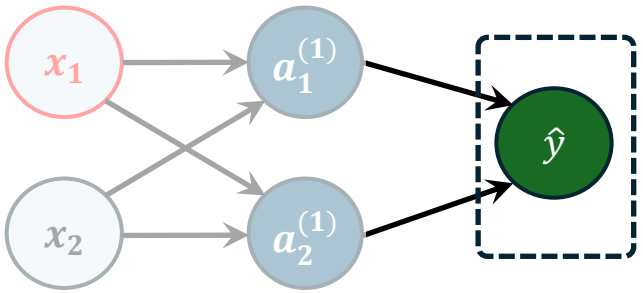
$$a_1^{(1)} = z_1^{(1)} \text{ (linear)}$$

$$a_2^{(1)} = z_2^{(1)} \text{ (linear)}$$

$$\hat{y} = z_1^{(2)} = w_{11}^{(2)} a_1^{(1)} + w_{12}^{(2)} a_2^{(1)} + b^{(2)}$$

What is the saliency of the first feature: $x_1$?

# A simple example




$$z_1^{(1)} = w_{11}^{(1)} x_1 + w_{21}^{(1)} x_2 + b_1^{(1)}$$

$$z_2^{(1)} = w_{12}^{(1)} x_1 + w_{22}^{(1)} x_2 + b_2^{(1)}$$

$$a_1^{(1)} = z_1^{(1)} \text{ (linear)}$$

$$a_2^{(1)} = z_2^{(1)} \text{ (linear)}$$

Use linear activation on all neurons

$$\hat{y} = z_1^{(2)} = w_{11}^{(2)} a_1^{(1)} + w_{12}^{(2)} a_2^{(1)} + b^{(2)}$$

What is the saliency of the first feature: $x_1$?

**Start from the output:**

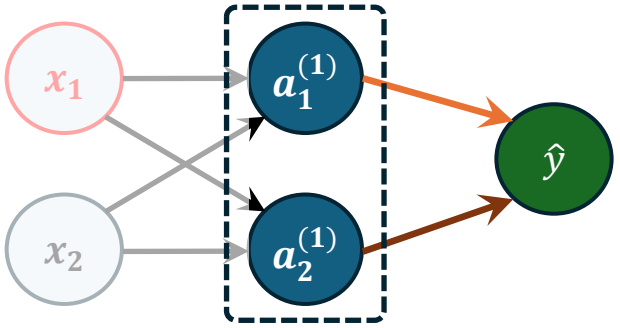
$$\delta_1^{(2)} = g'\left(z_1^{(2)}\right) = \frac{\partial}{\partial z_1}(z) = 1$$

Because we used linear activation

$$\frac{\partial f}{\partial x_1} = \sum_j \delta_j^{(1)} w_{j1}^{(1)}$$

Error signals on the neurons $x_1$ feeds into

# A simple example




Use linear activation on all neurons

$$z_1^{(1)} = w_{11}^{(1)} x_1 + w_{21}^{(1)} x_2 + b_1^{(1)}$$

$$z_2^{(1)} = w_{12}^{(1)} x_1 + w_{22}^{(1)} x_2 + b_2^{(1)}$$

$$a_1^{(1)} = z_1^{(1)} \text{ (linear)}$$

$$a_2^{(1)} = z_2^{(1)} \text{ (linear)}$$

$$\hat{y} = z_1^{(2)} = w_{11}^{(2)} a_1^{(1)} + w_{12}^{(2)} a_2^{(1)} + b^{(2)}$$

What is the saliency of the first feature: $x_1$?

**Start from the output:**

$$\delta_1^{(2)} = g'\left(z_1^{(2)}\right) = \frac{\partial}{\partial z_1}(z) = 1$$

Because we used linear activation

$$\frac{\partial f}{\partial x_1} = \sum_j \delta_j^{(1)} w_{j1}^{(1)}$$

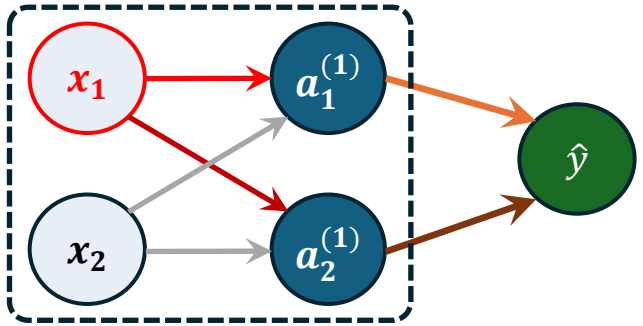
**Then backpropagate:**

$$\delta_1^{(1)} = g'\left(z_1^{(l)}\right) \sum_k w_{k1}^{(2)} \delta_k^{(2)} = w_{11}^{(2)}$$

...and...

$$\delta_2^{(1)} = w_{12}^{(2)}$$

# A simple example




Use linear activation on all neurons

$$\frac{\partial f}{\partial x_1} = \delta_1^{(1)} w_{11}^{(1)} + \delta_2^{(1)} w_{21}^{(1)}$$

$$\frac{\partial f}{\partial x_1} = w_{11}^{(2)} w_{11}^{(1)} + w_{12}^{(2)} w_{21}^{(1)}$$

What is the saliency of the first feature: $x_1$?

$$\frac{\partial f}{\partial x_1} = \sum_j \delta_j^{(1)} w_{j1}^{(1)}$$

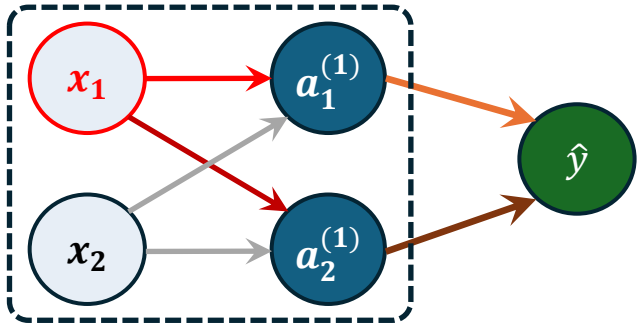
**Then backpropagate:**

$$\delta_1^{(1)} = g'\left(z_1^{(l)}\right) \sum_k w_{k1}^{(2)} \delta_k^{(2)} = w_{11}^{(2)}$$

...and...

$$\delta_2^{(1)} = w_{12}^{(2)}$$

# A simple example



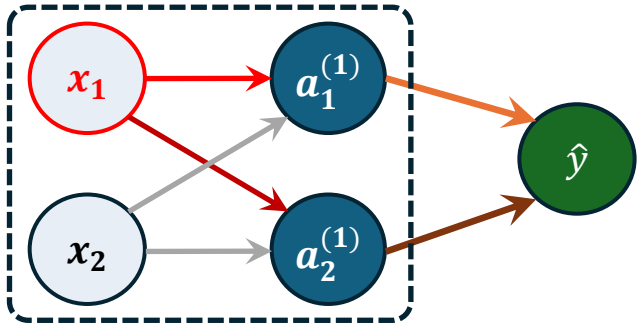

Use linear activation on all neurons

$$\frac{\partial f}{\partial x_1} = \delta_1^{(1)} w_{11}^{(1)} + \delta_2^{(1)} w_{21}^{(1)}$$

$$\frac{\partial f}{\partial x_1} = w_{11}^{(2)} w_{11}^{(1)} + w_{12}^{(2)} w_{21}^{(1)}$$

Notice that **larger weights** connected (directly or indirectly) to $x_1$ lead to **greater saliency** – large effect on output.
This makes sense!

# A simple example




Use linear activation on all neurons

$$\frac{\partial f}{\partial x_1} = \delta_1^{(1)} w_{11}^{(1)} + \delta_2^{(1)} w_{21}^{(1)}$$

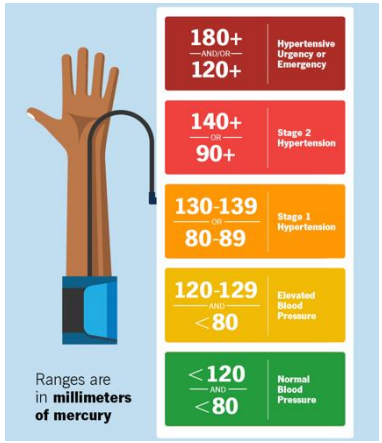
$$\frac{\partial f}{\partial x_1} = w_{11}^{(2)} w_{11}^{(1)} + w_{12}^{(2)} w_{21}^{(1)}$$

...Interestingly, it's a constant (same for every sample)...

**This would *not* be the case if non-linear activation was used.**

**Remember, saliency is local, not global!**

# Visualizing saliency results on tabular data

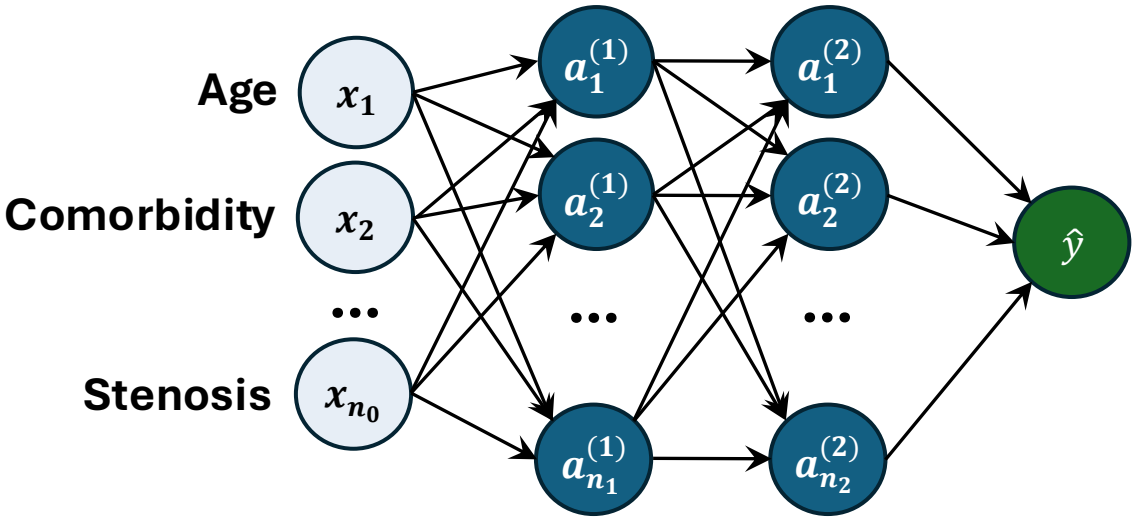




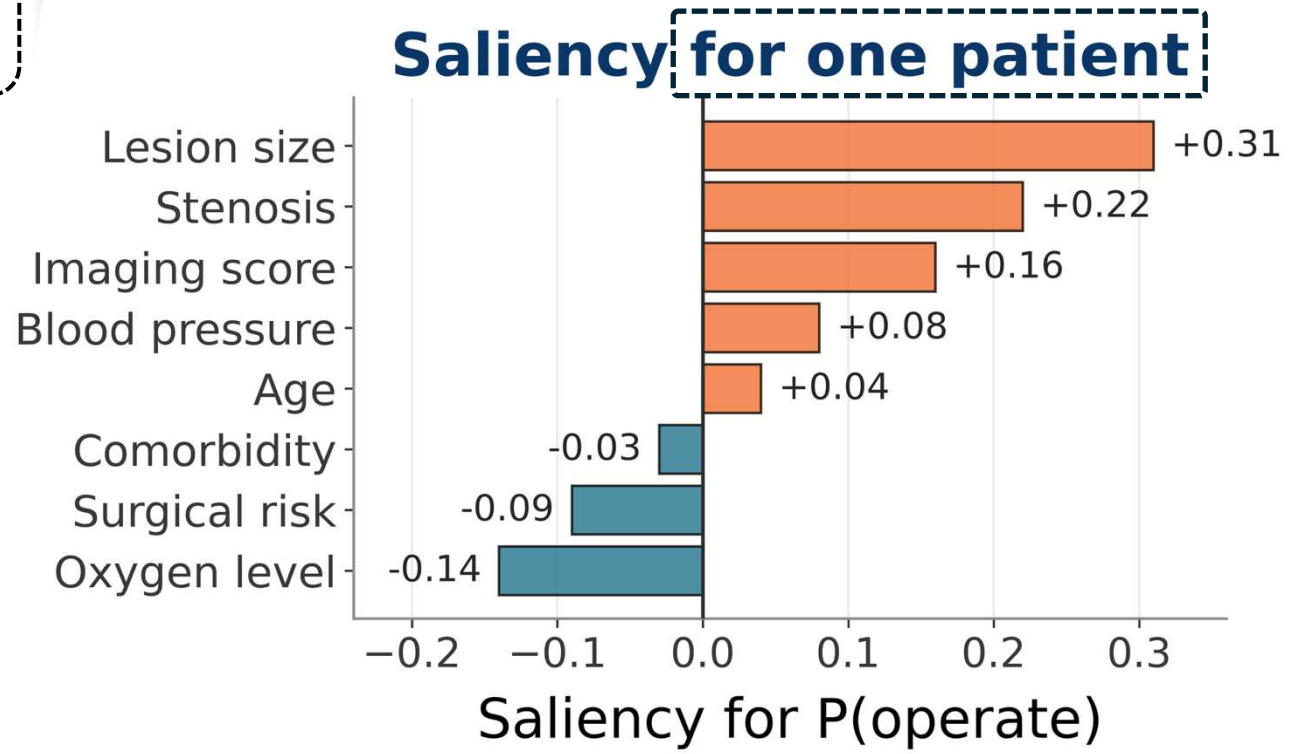



$P(\text{operate}|x)$

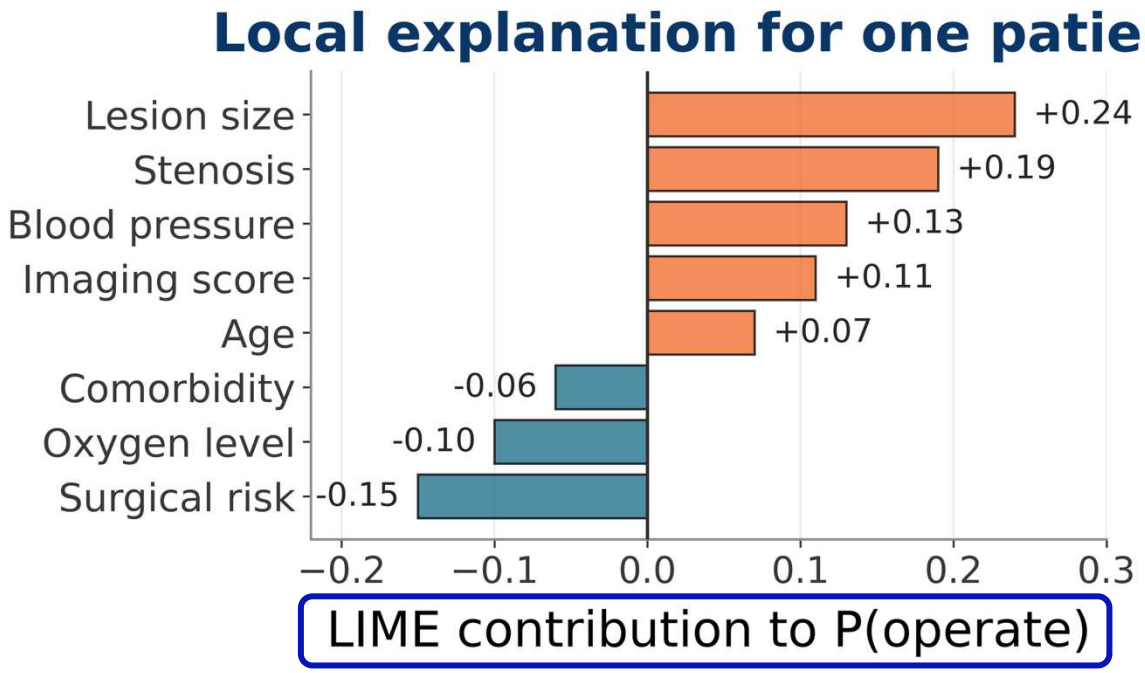
Here I care about sign, so I will ***not*** take absolute value




**Local!**

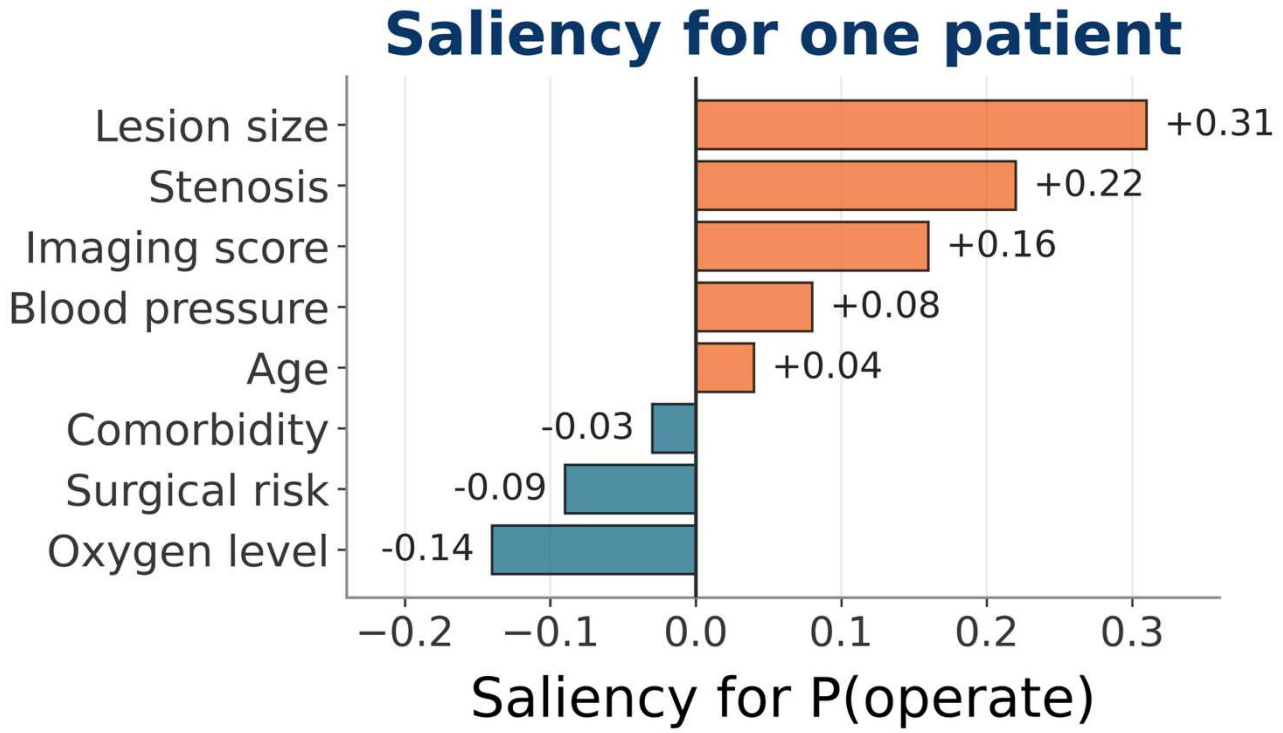


# Visualizing saliency results on tabular data

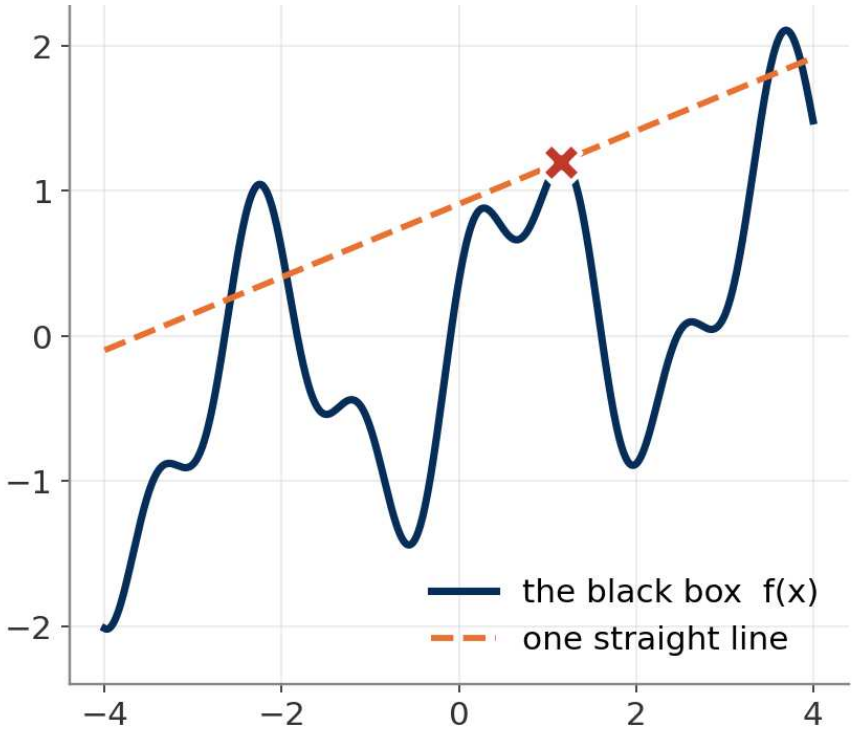


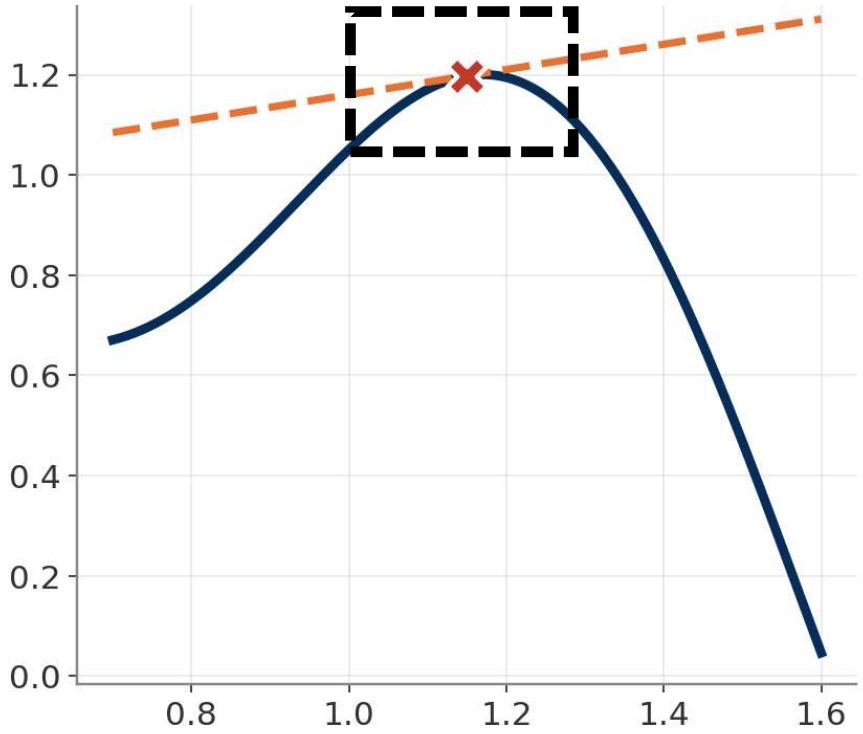

**Recall the LIME plot from our previous lecture. The values appear very similar to saliency…So, what's the difference?**

**Does this look familiar? It should!**



# LIME *vs.* saliency




With LIME, we ***approximated*** the slope by evaluating perturbed samples around the point of interest

**With saliency, we compute the slope *exactly*** without needing to evaluate many samples and choose hyperparameters (neighborhood)

# Saliency “maps” on images

Egyptian cat 54%



golden retriever 48%

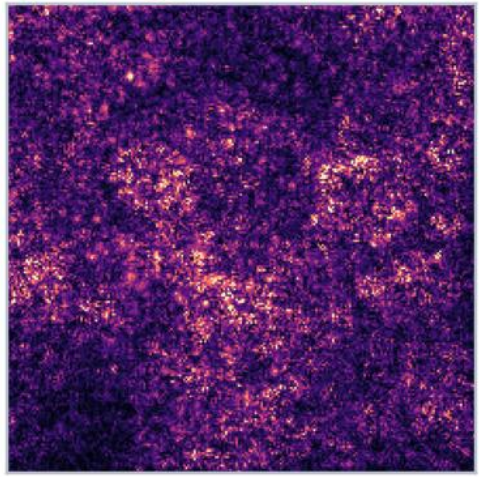


espresso 96%

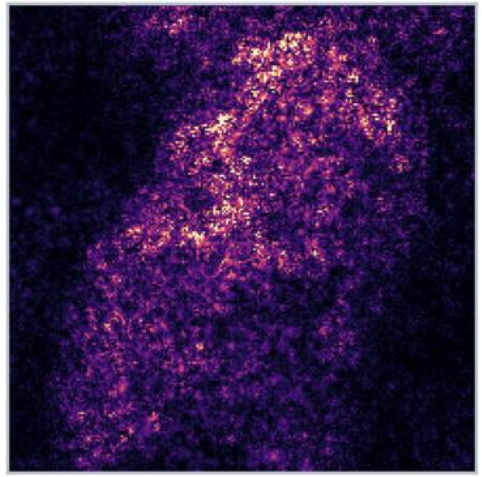


Same idea, except now **each pixel is a feature**

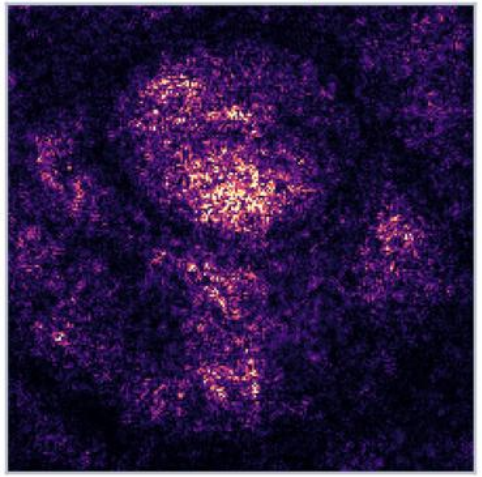
| ∂ logit / ∂ pixel |



| ∂ logit / ∂ pixel |



| ∂ logit / ∂ pixel |



And for classification **the output is a probability (logit)**

# Questions?

# Limitations of saliency maps for images




Saliency is often more interpretable on tabular data than on images, because tabular features are meaningful variables, whereas image saliency is computed on **individual pixels.**

**Challenges summarized:**

1. Gradients are often quite **noisy pixel-by-pixel.** But as humans, this is not really how we interpret images. Rather, we assess them at a *feature* level.

2. Saliency tells us which pixels the model's prediction is **most sensitive to**, but this does not necessarily mean they are the most important

# Recall: Convolutional neural networks (CNNs)

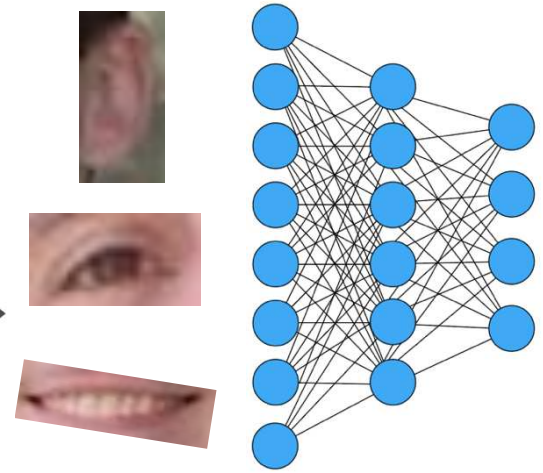


4032×3024 ~ **12 million pixels!**

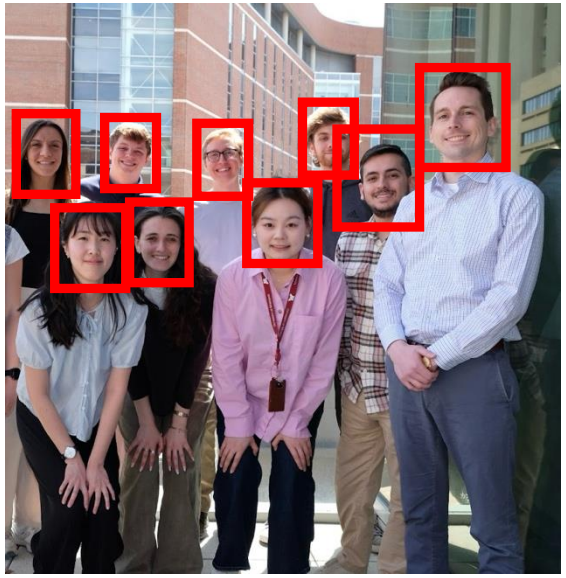
$$\begin{pmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{pmatrix}$$

***Convolution***

Edges,
corners, *etc.*



**Faces identified**



Convolutional filters extract high-level features from nearby groups of pixels. And **they are *translationally equivariant,*** meaning they can detect the same feature **anywhere in the image.**

# Conv filters produce *feature maps*





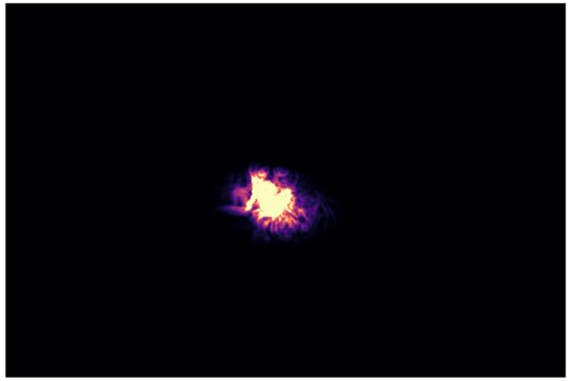
Conv filter #1 in the 1st layer detects edges



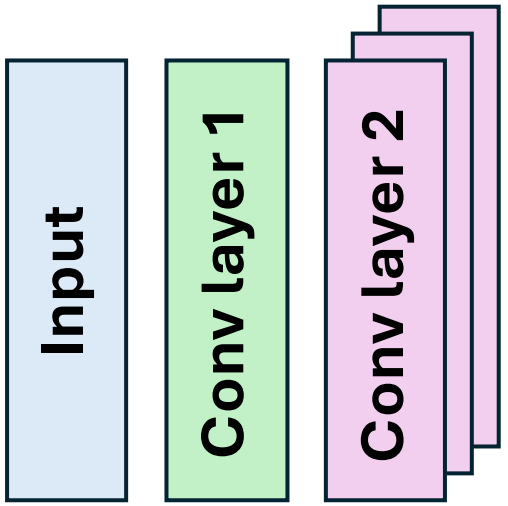
Filter #1 detects ears



Filter #2 detects eyes



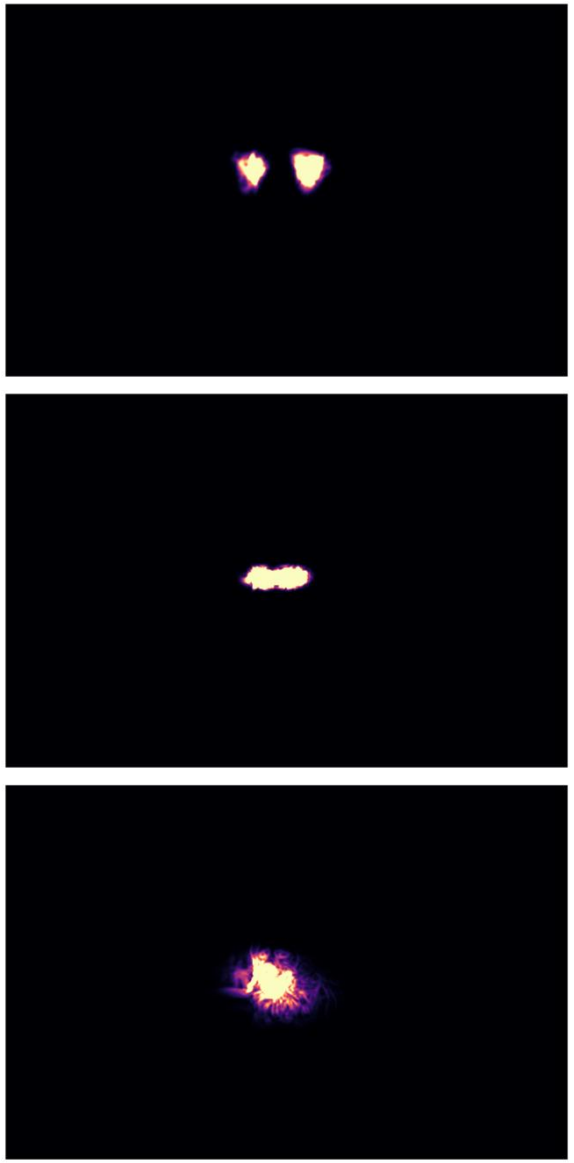
Filter #3 detects snouts




$P(\mathbf{dog}|x)$

Note: these are just schematic (usually not as nice)

# These feature maps inform the output



**Average** pixel value of Feature map #1

**Average** pixel value of Feature map #2

**Average** pixel value of Feature map #3

$w_1$

$w_2$

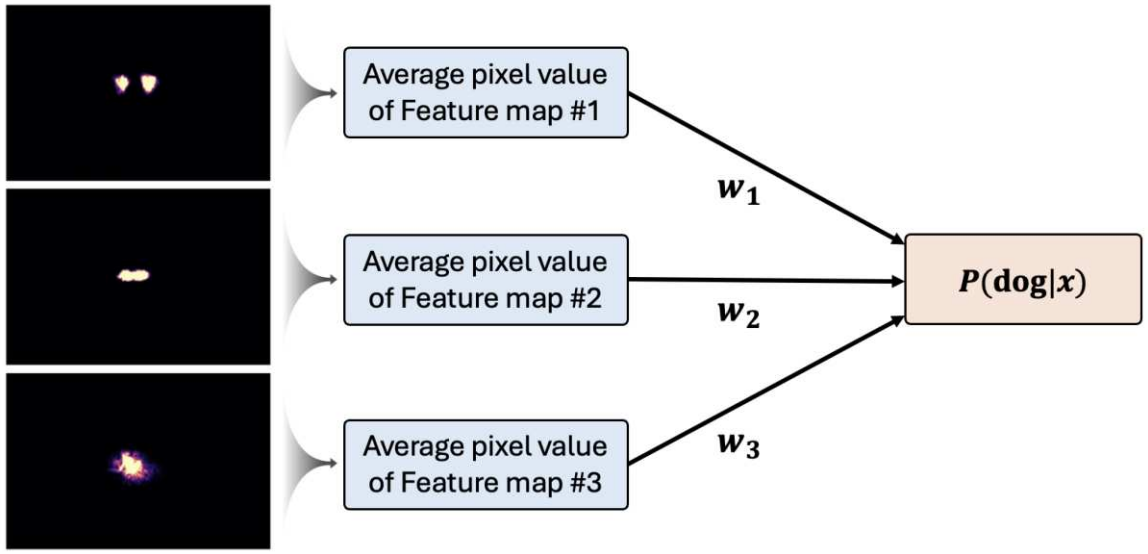
$w_3$

$P(\mathbf{dog}|x)$

This "averaging" step is called **global average pooling**

Weights describe how much each feature contributes to the output

# Class Activation Maps (CAMs) help visualize



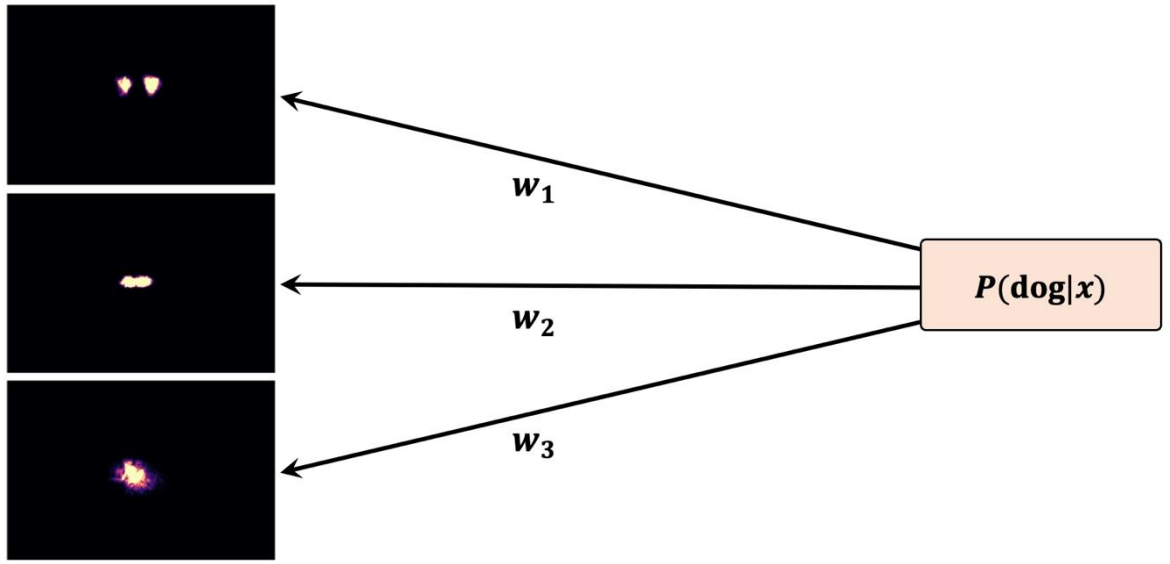

Mathematically: $\hat{y} = \sum_k w_k F_k + b$

$w_k$ is the weight connecting pooled feature $F_k$ to the output

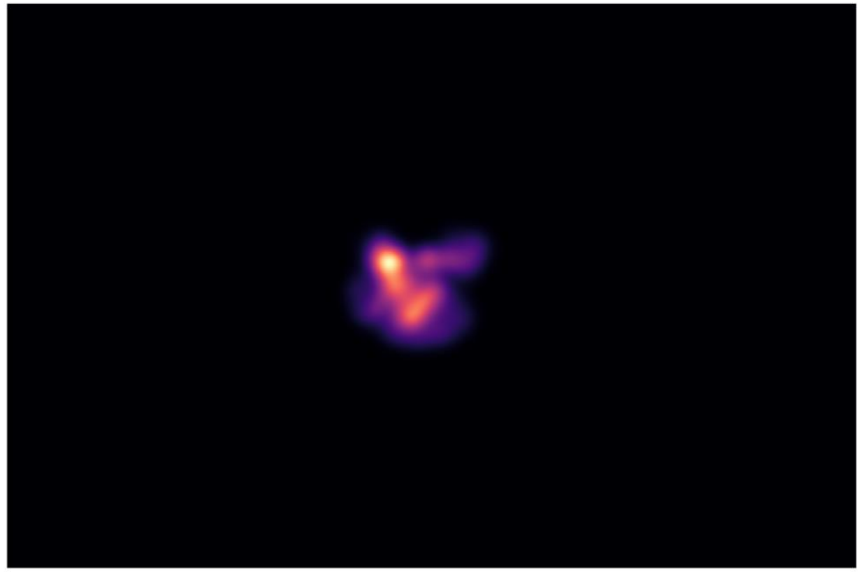
But what we *want* is a visual of **where these features are located** in the image.

With CAM, we feed weights back to the **raw feature maps (before pooling):**

$$M = \sum_k w_k A^k$$



# Class Activation Maps (CAMs) help visualize





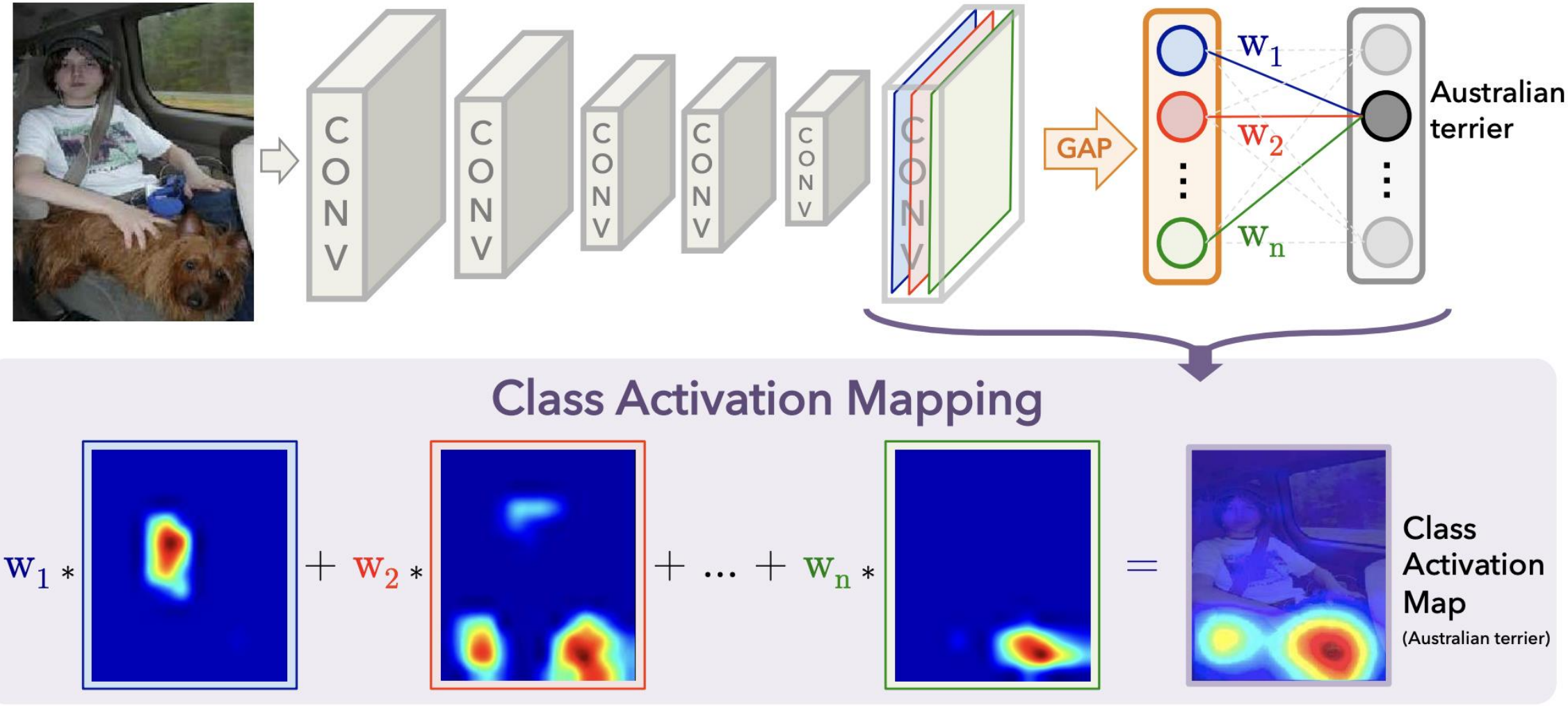
The **class activation map** shows us what parts of the image were used to make a prediction.

In other words: **Where the dog is!**

Or if our model largely made classifications based on the background, CAM will help tell us – since the background feature map will get a large weight



# CAMs for multi-class prediction

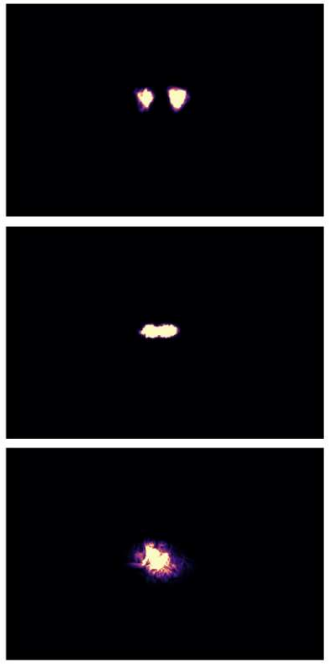



**From the original paper:** Learning Deep Features for Discriminative Localization (2015).

# CAMs for multi-class prediction

Now we get **one CAM per class:**

$$M^c = \sum_k w_k^c A^k$$



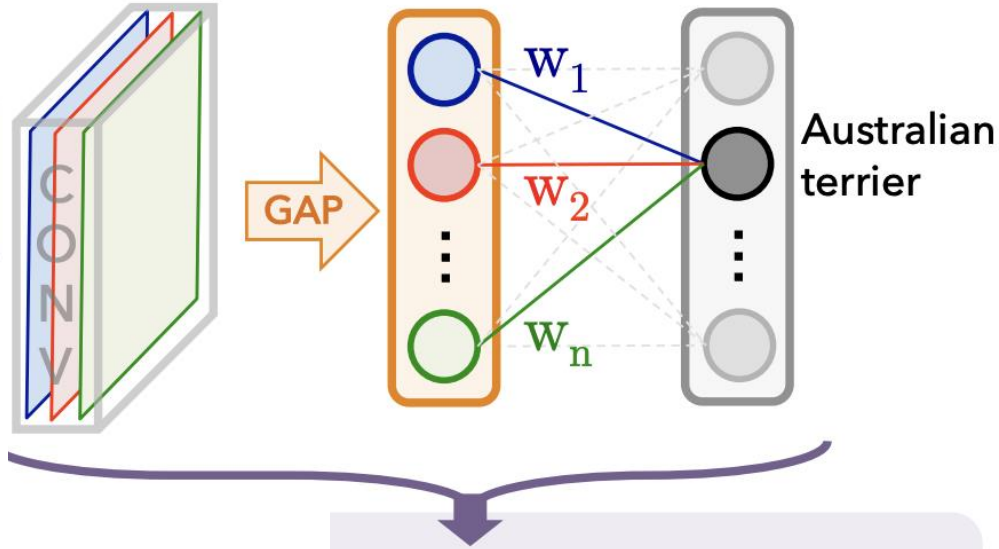
Average pixel value of Feature map #1

Average pixel value of Feature map #2

Average pixel value of Feature map #3

$P(\text{human}|x)$

$P(\text{dog}|x)$



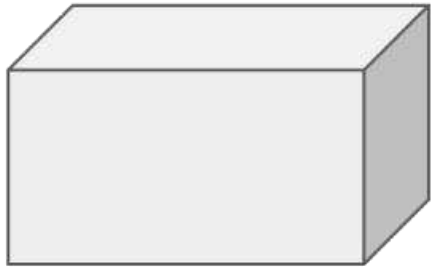




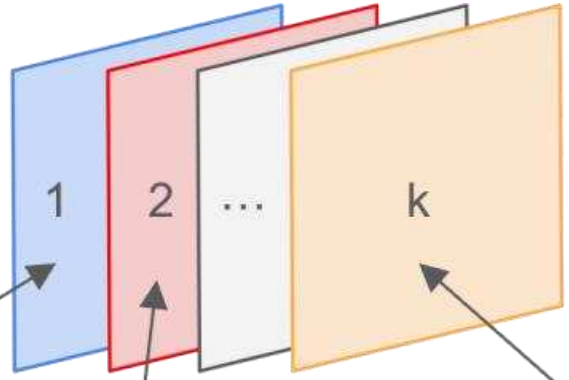
**Each CAM only cares about weights connected to a specific class ($c$)**

# CAMs for multi-class prediction



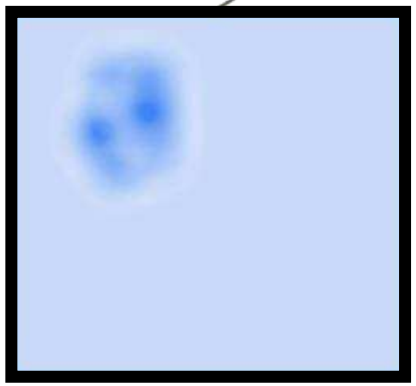


Conv Layer n-1




Conv Layer n

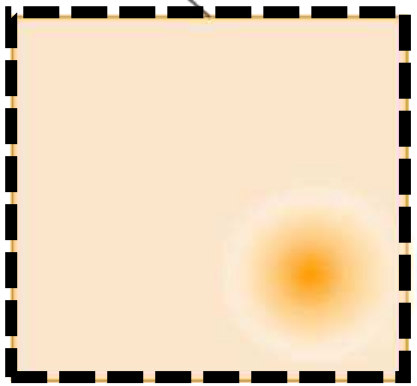
This should contribute to $P(\text{human}|x)$



human face



dog fur



dog ears

These to $P(\text{dog}|x)$

# CAMs for multi-class prediction

**CAM for “Cat”**





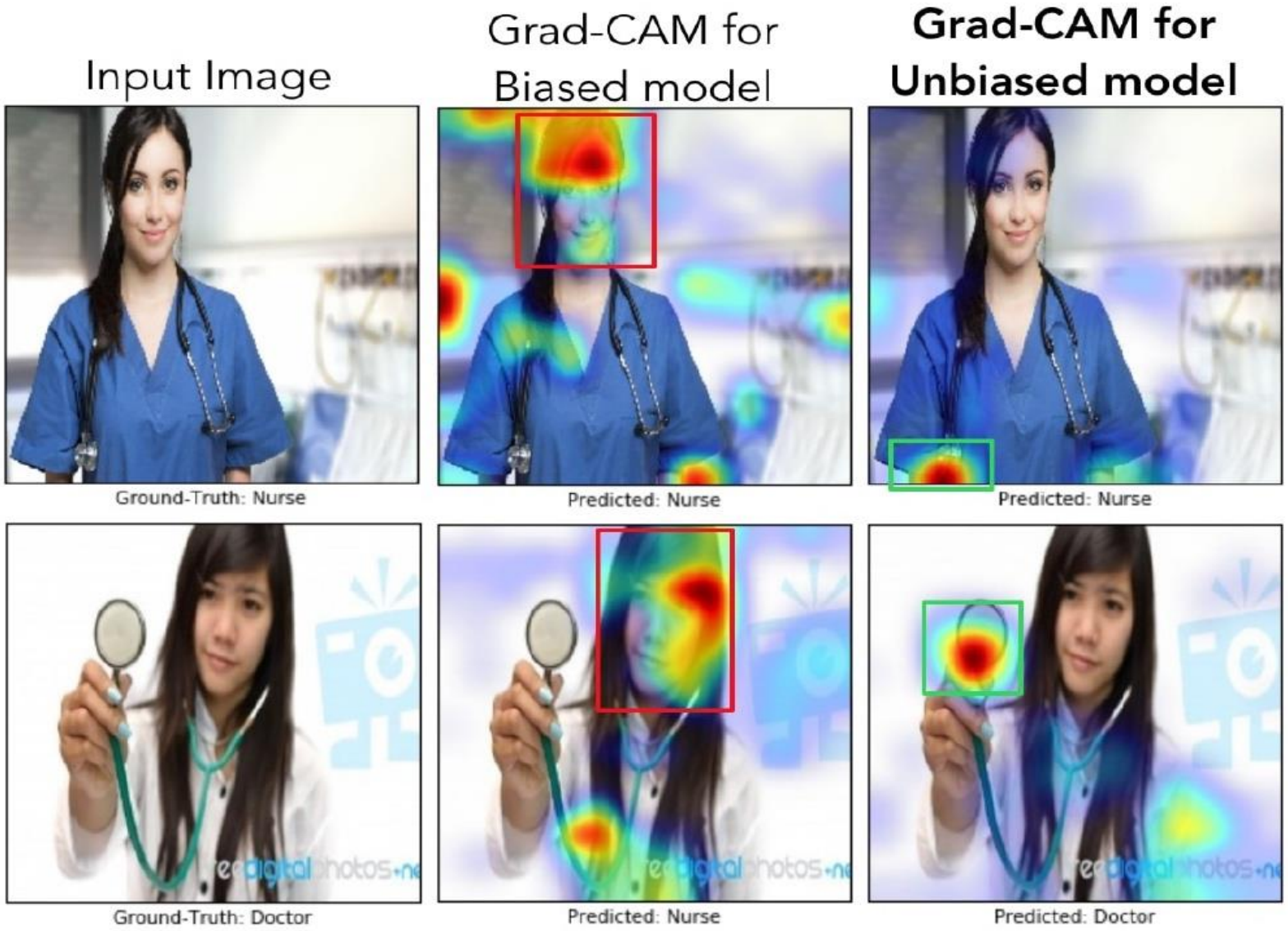
**CAM for “Dog”**



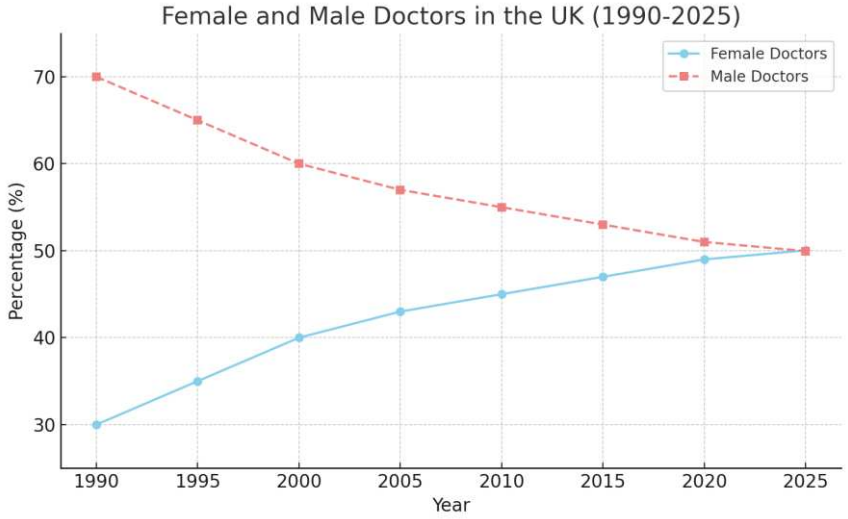
CAMs are sometimes used as a rough approach for **object detection**

(in practice, researchers use **Grad-CAM:** the saliency of each feature map)

# CAMs are useful for catching bias




In this example:
Does the model classify **doctor *vs.* nurse** based on clothing and stethoscope? Or based on the person's hair (long *vs.* short)?



# Questions?

Next: We'll cover **how sure** our models are in these predictions

# Measuring prediction (un)certainty

Grad-CAM for
Unbiased model



Predicted: Nurse



Predicted: Doctor

Our model gives:

$$P_{\text{nurse}} = 0.89$$

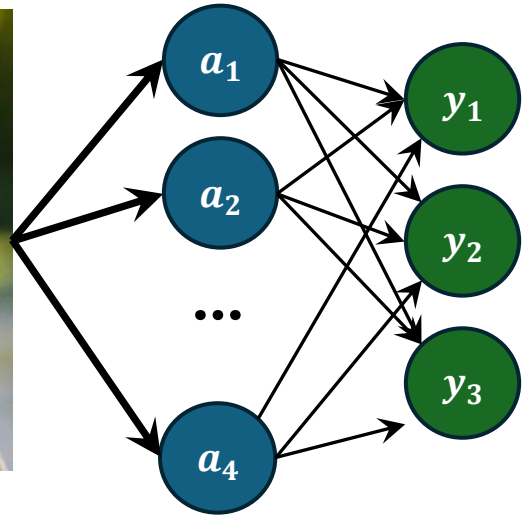
**Does this mean we are 89% and 42% *certain* in the predictions?**

$$P_{\text{doctor}} = 0.52$$

# With simogid & softmax, outputs sum to 1



Each pixel is an input neuron




**Cat** – raw output 1.2

**Dog** – raw output 5.7

**Fox** – raw output 4.9

***Softmax***

**Look like measures of confidence. Are they?**

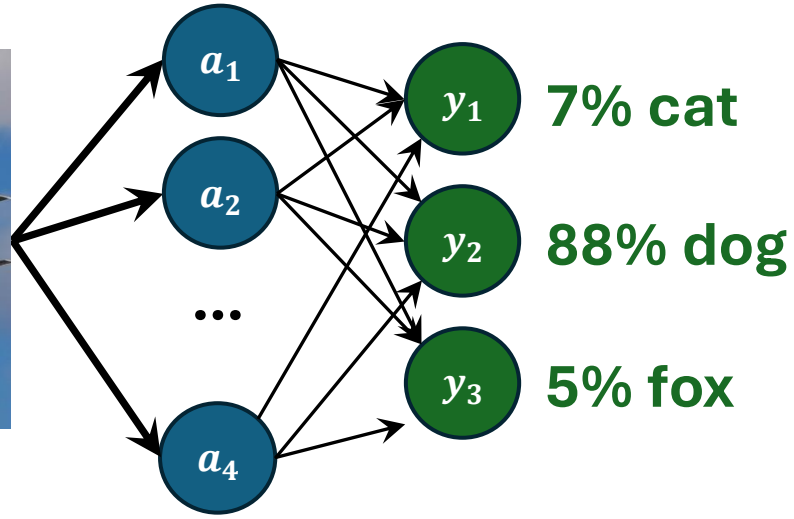
$$P(y = \text{dog}) = \frac{e^{5.7}}{e^{1.2} + e^{5.7} + e^{4.9}} \approx 0.685$$

$$P(y = \text{cat}) = \frac{e^{1.2}}{e^{1.2} + e^{5.7} + e^{4.9}} \approx 0.008$$

$$P(y = \text{cat}) = \frac{e^{4.9}}{e^{1.2} + e^{5.7} + e^{4.9}} \approx 0.308$$

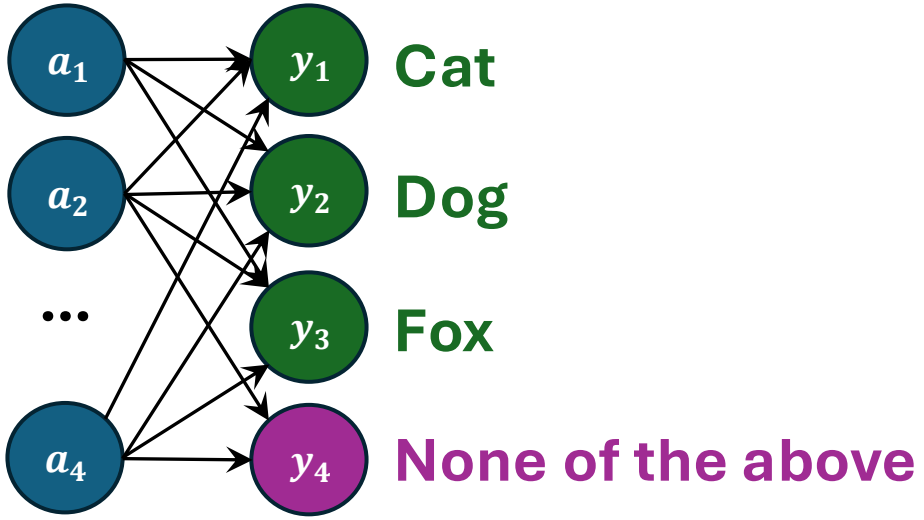
# Limitations of probability for model confidence






Suppose we gave our trained model an airplane. It should give **~0% probability** for the known classes (cat, dog, fox) but this is **not possible with our NN!**

We could introduce a 4th output that represents ***not*** a dog/fox/cat, but this is an unbounded set that would require many training samples.*
And: results will be **poorly calibrated**

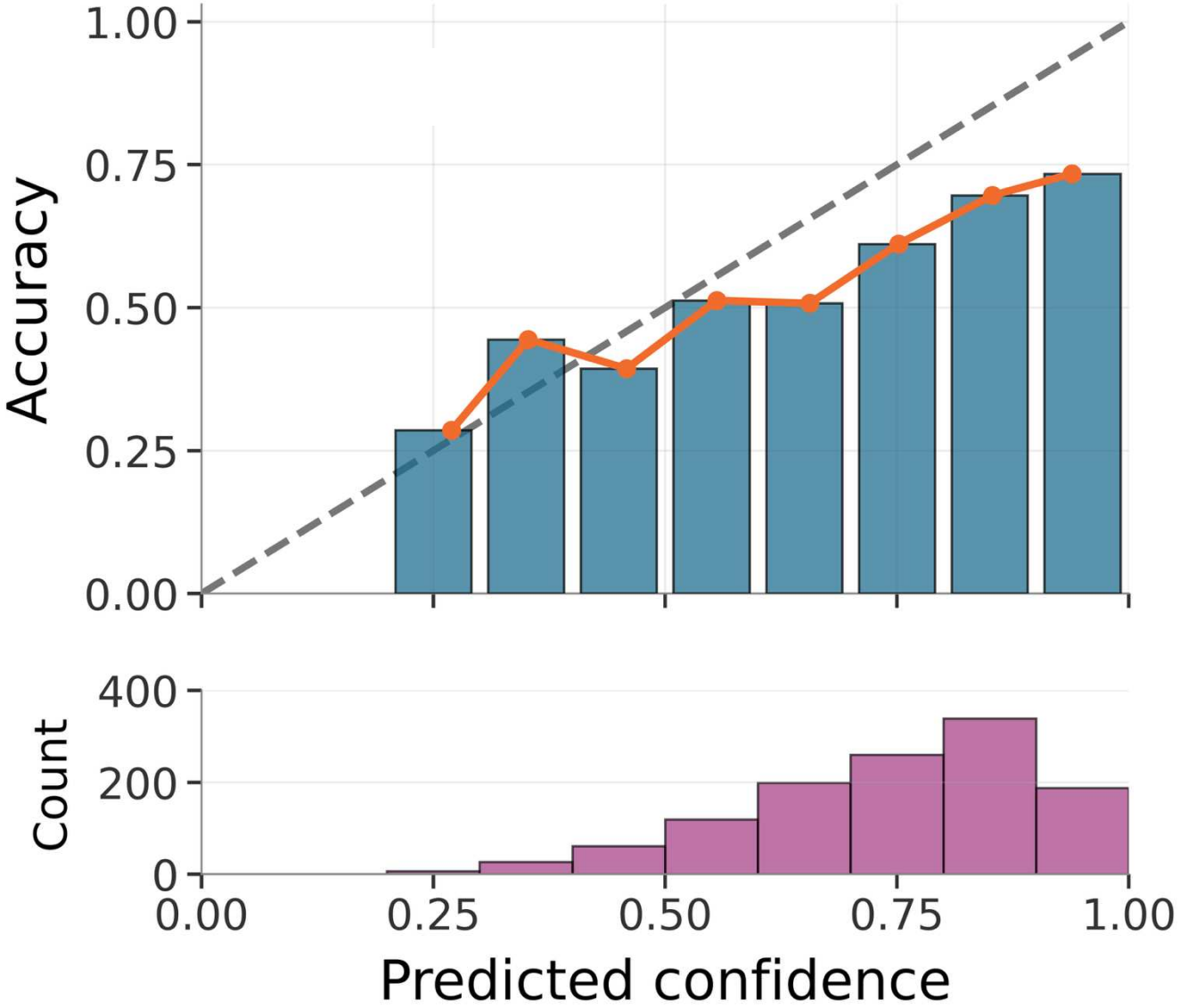



*There are many things that are not dogs, foxes, or cats

# Calibration: Reliability diagrams

- We have a set of 100 images
- Our model predicts each of them to be a cat with a **probability of 60%**
- If that model is **well calibrated**, it should be **correct 60%** of the time, meaning 60/100 images have cats





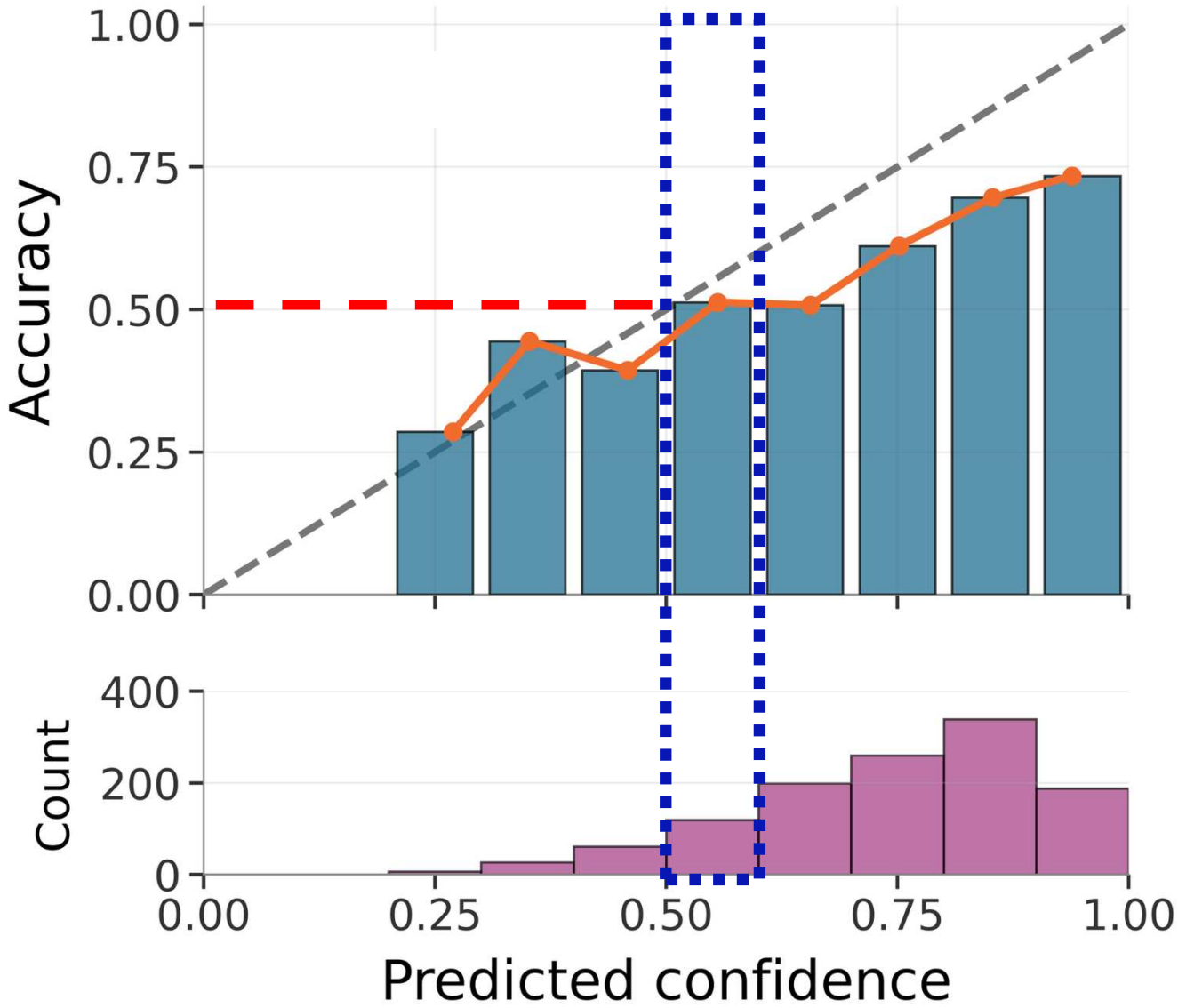
# Calibration: Reliability diagrams

**On the test set, we ask:**

Of the predictions where the output probability is between 50-60%, what percentage are correct? In other words, what is the accuracy?

A well-calibrated model has a **direct correlation between probability and accuracy**

If this is the case, we can use probability as a measure of confidence



# Calibration: Reliability diagrams

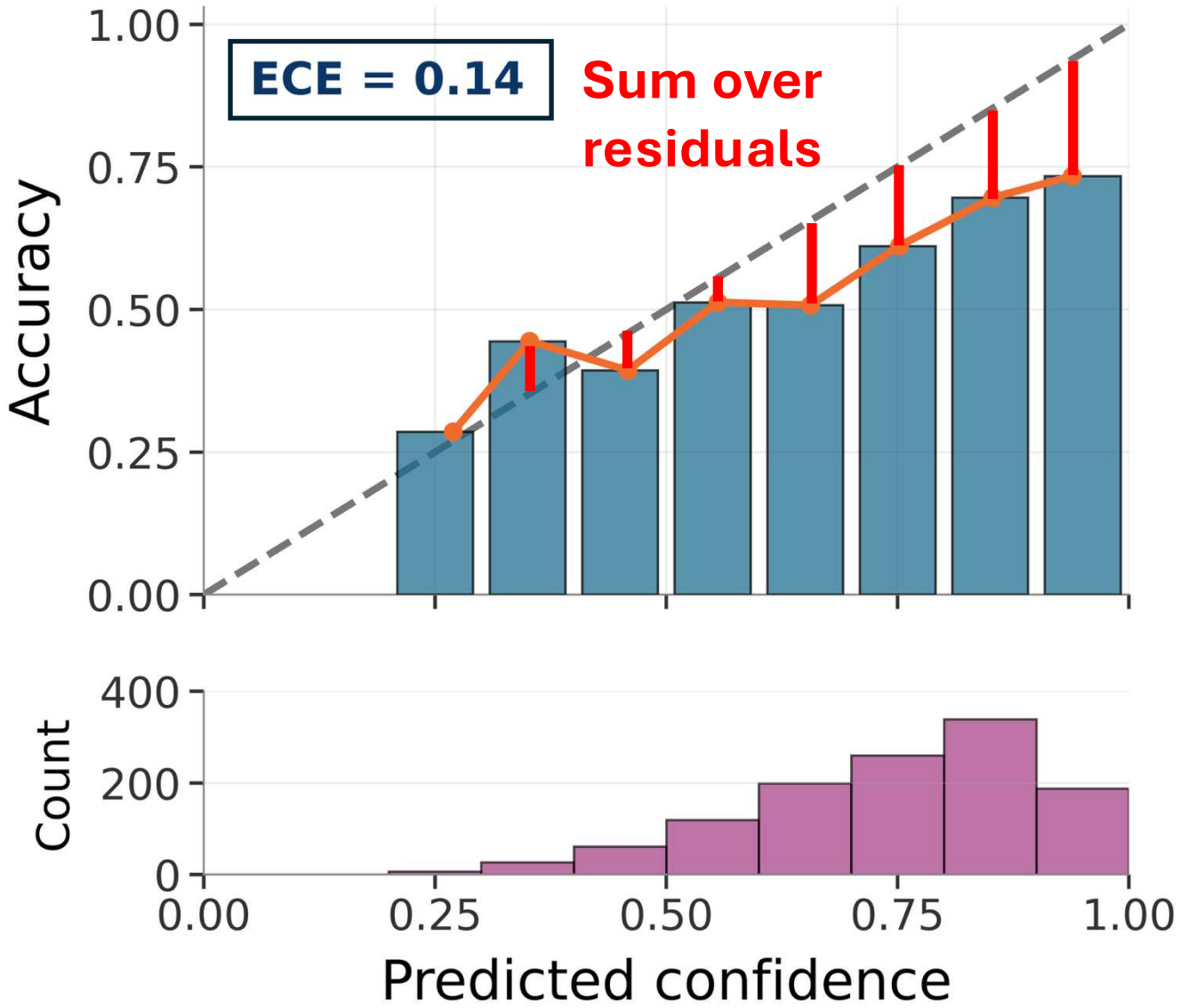
**We can quantify this using the expected calibration error (ECE):**

$$\sum_{m=1}^{M} \frac{|B_m|}{N} |\mathrm{acc}(B_m) - \mathrm{conf}(B_m)|$$

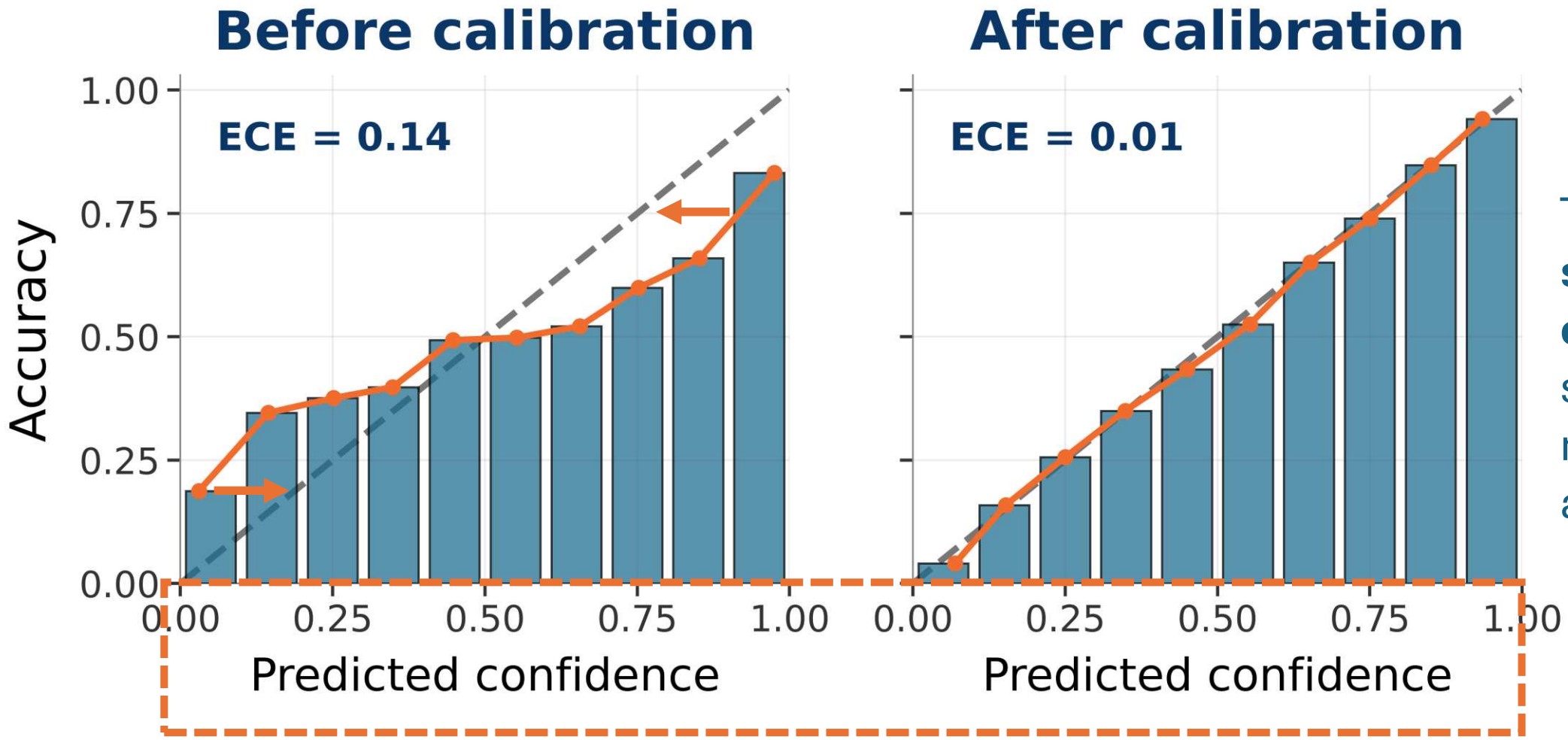
- $|B_m|$: size of each bin (*e.g.*, 50-60%)
- $N$: num samples across *all* bins

**Low ECE (close to zero) means the model is well calibrated**

**What if it is not calibrated?**



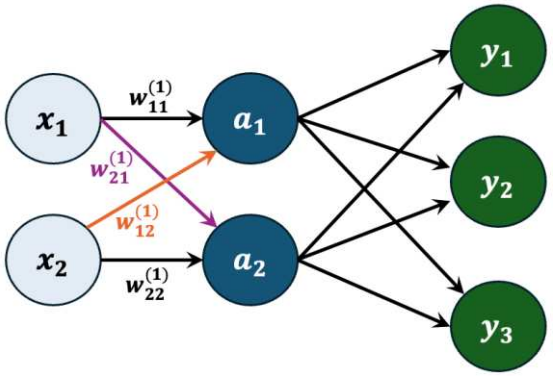
# Model calibration to lower ECE




The idea is to **scale the raw confidence** so it better matches accuracy

**The question then becomes: Scale by what factor?**

# Model calibration: Temperature scaling



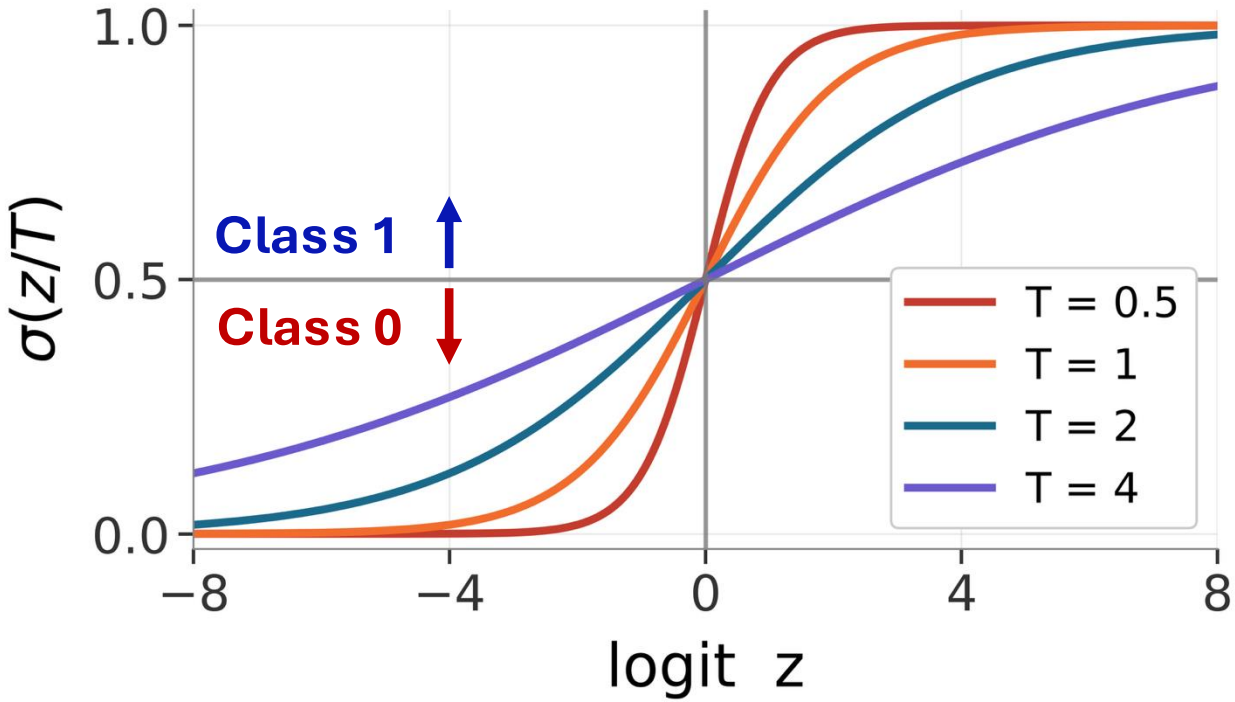

**Traditional softmax:**

$$P(y = k) = \frac{e^{y_k}}{\sum_{i=1}^{K} e^{y_i}}$$

**Temperature Scaling**

**Calibrated output:**

$$P(y = k) = \frac{e^{y_k/T}}{\sum_{i=1}^{K} e^{y_i/T}}$$

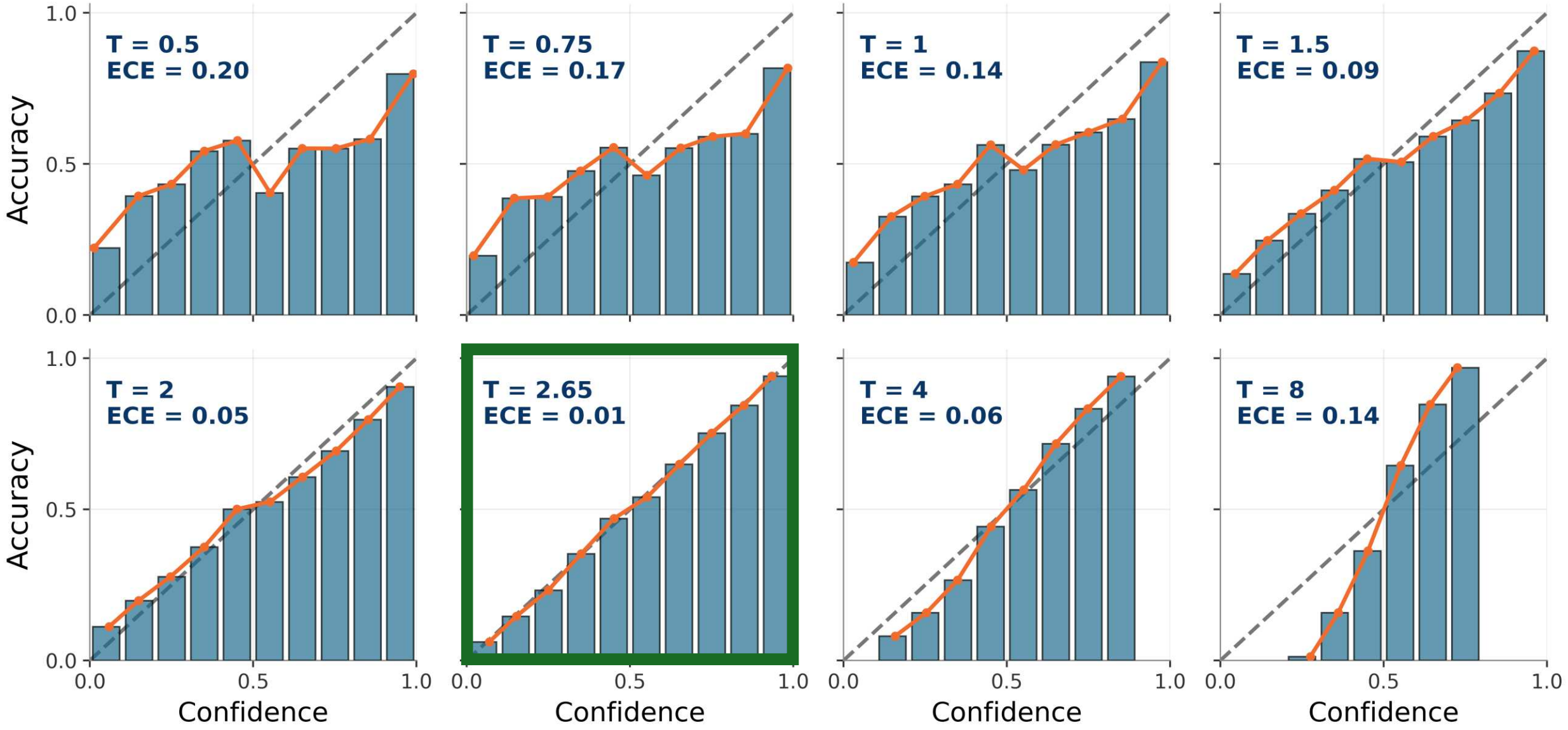



Temperature does ***not*** change ranking, just the probability!

**Higher *T* = smoother curve**

For the scientists in the room, this is called temperature given its similarity to the Boltzmann distribution: $e^{-E_i/k_B T}$

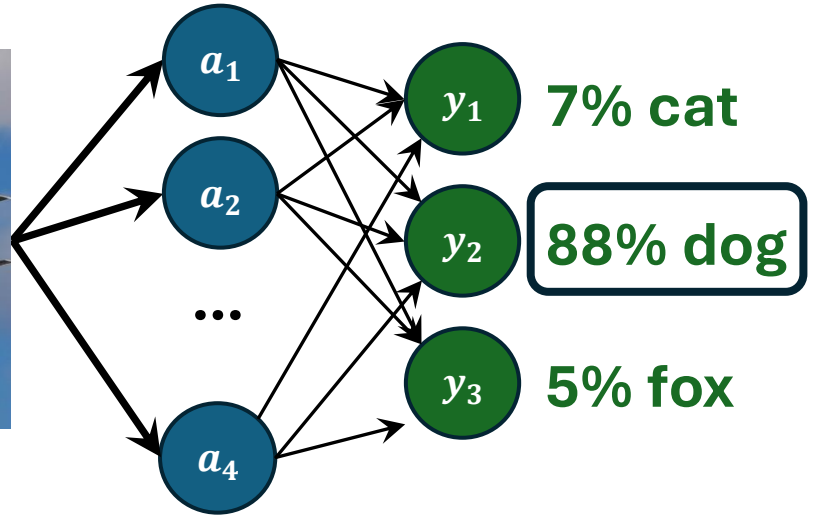
# Choose *T* that gives smallest ECE on a validation set



# Questions?

# Temperature scaling alone is often insufficient

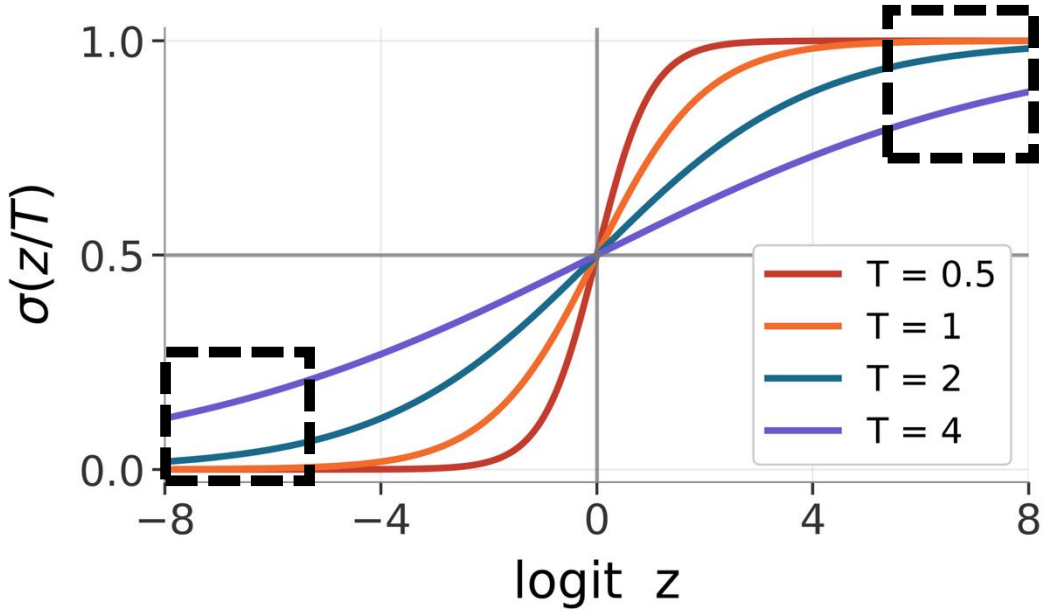





Even with temperature scaling, the model will still always **predict dog.** **Remember that temperature scaling does *not* change the predicted class**

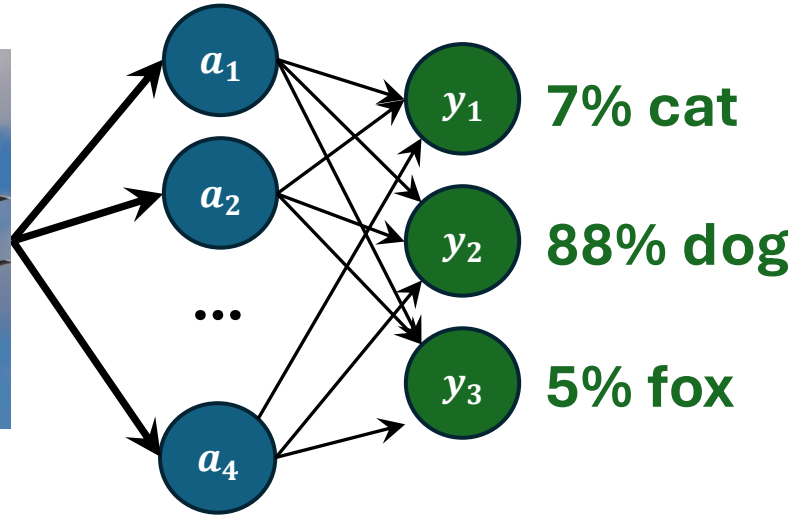
Moreover, when the output probability is very high, which can happen often on out-of-distribution samples, *T*-scaling does not change the output much.

**99.9% raw → still high after scaling**

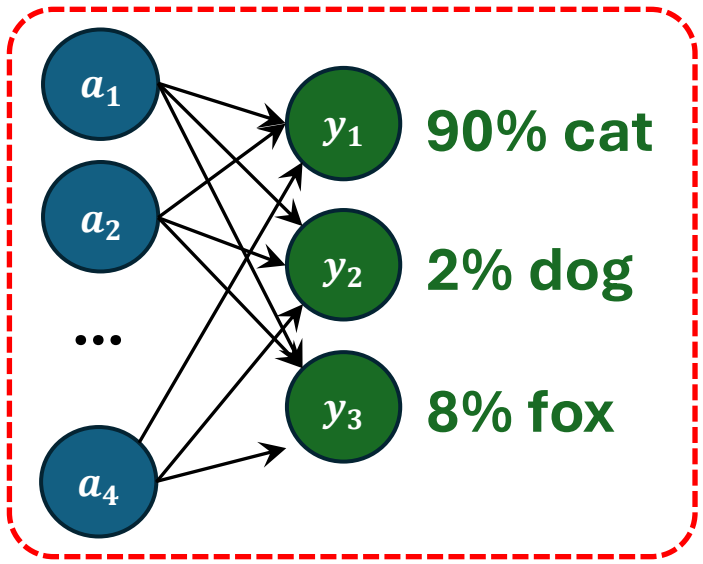


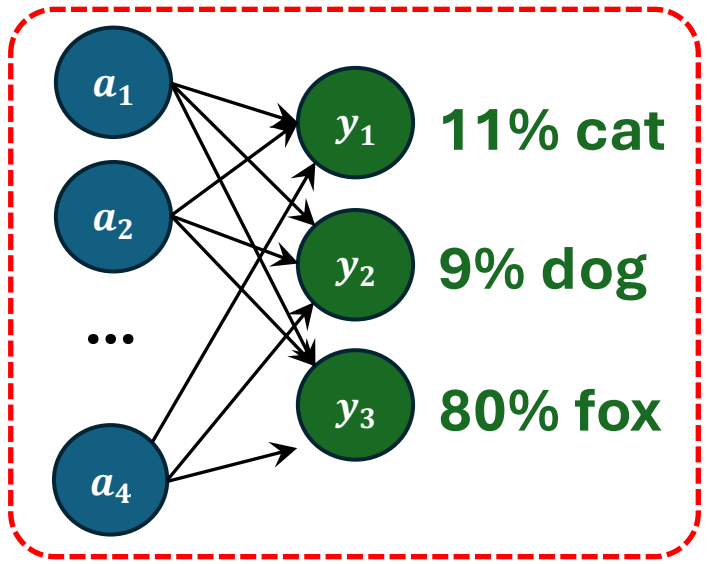
# Introducing *deep ensembles*

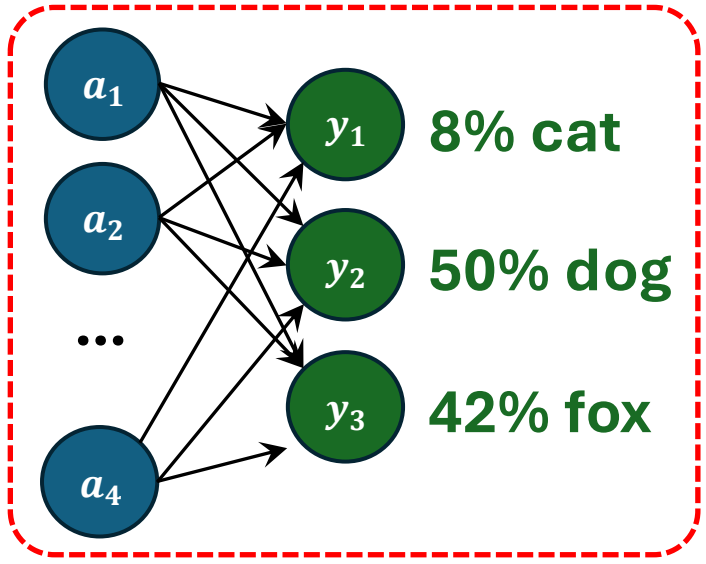





What if instead of training one model, we trained **many slightly different models**, then applied them all at inference?




**Then aggregate all the predictions in some way**

# How to generate *different* models

**1. Different initialization**

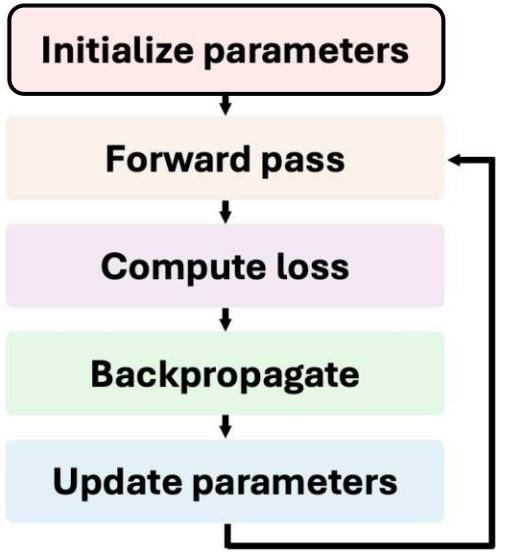
2. Different mini-batch ordering

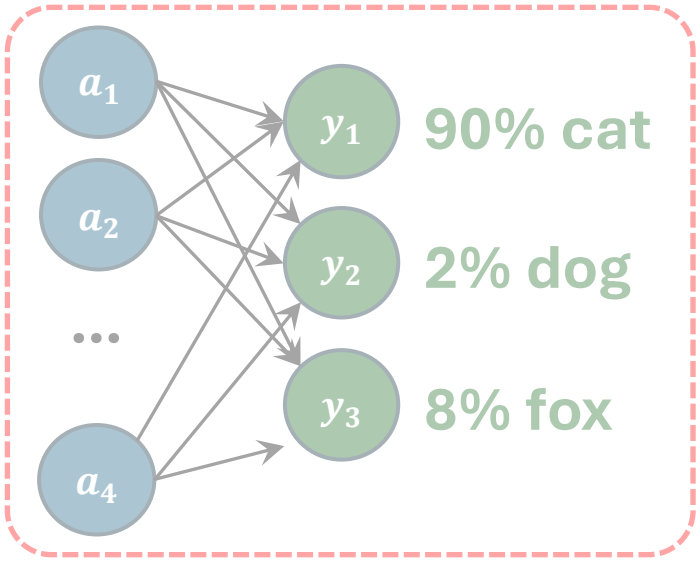
3. Different subset of training data

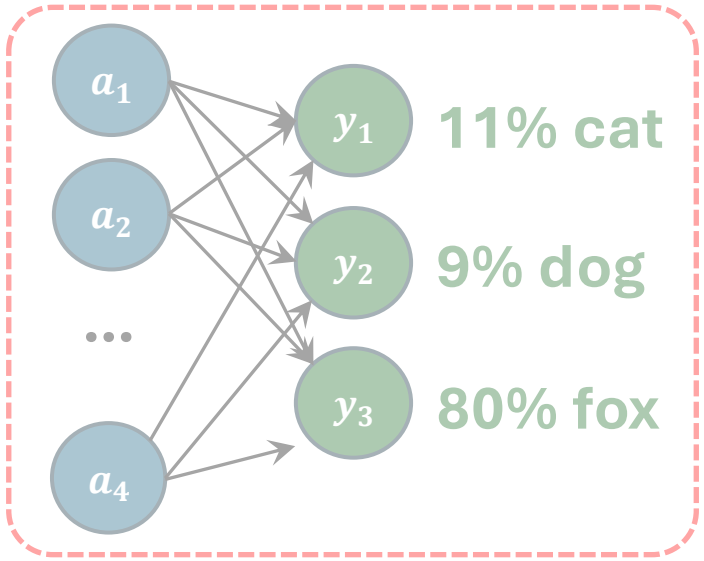
Remember these are **randomly initialized** as small values near zero

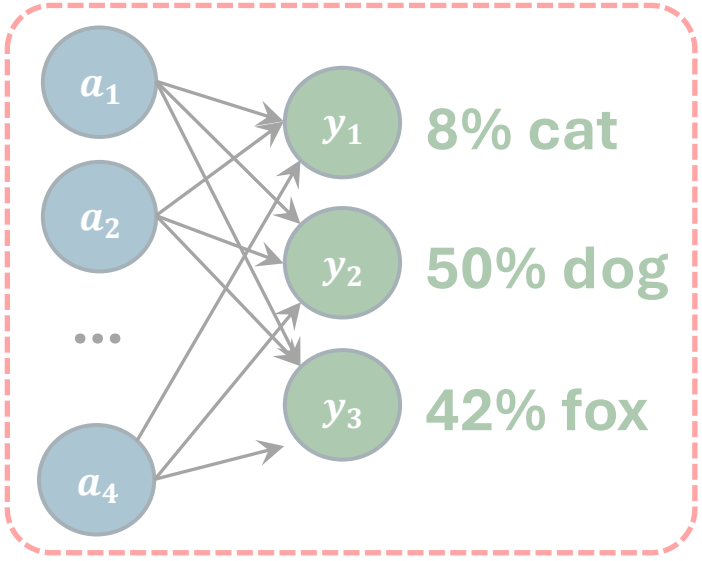
$(w_{11}, w_{12}, w_{13}, w_{21}, \dots)$

$(0.08, -0.02, 0.01, 0.05, \dots)$



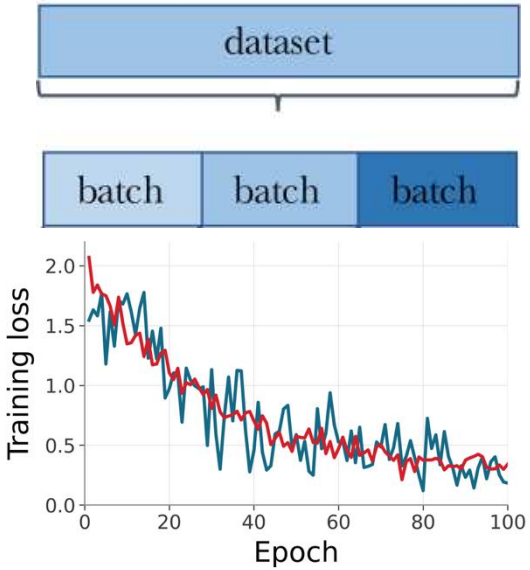

Then aggregate all the predictions in some way

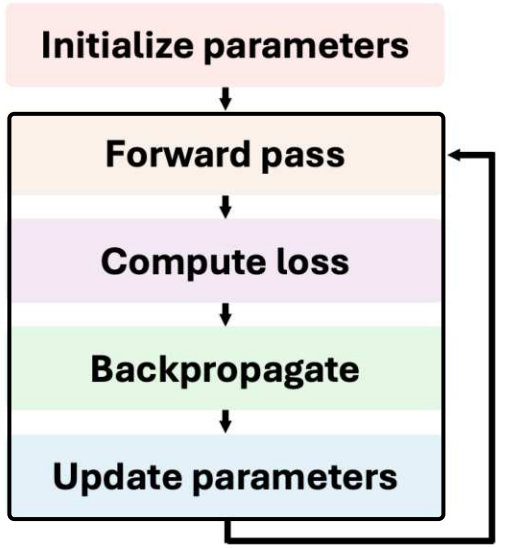
# How to generate *different* models

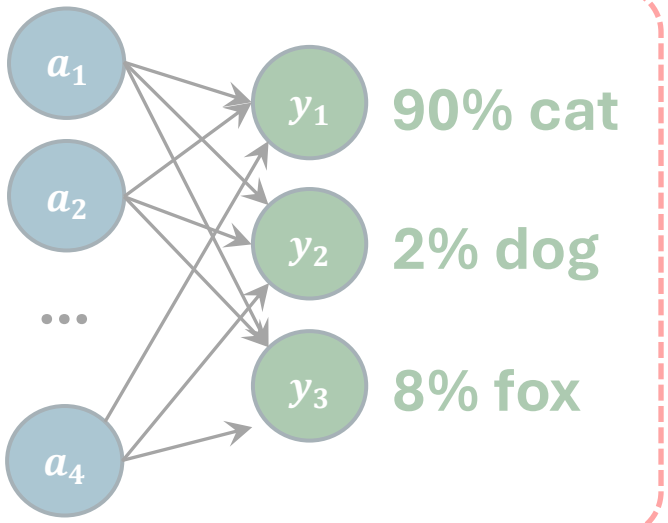
1. Different initialization

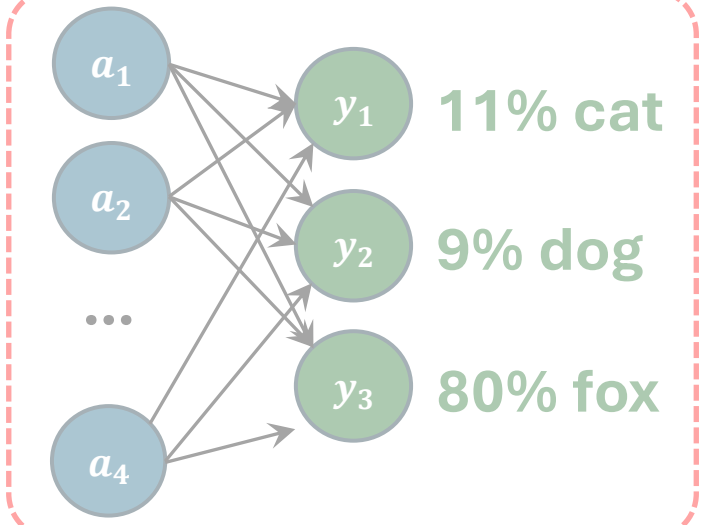
**2. Different mini-batch ordering**

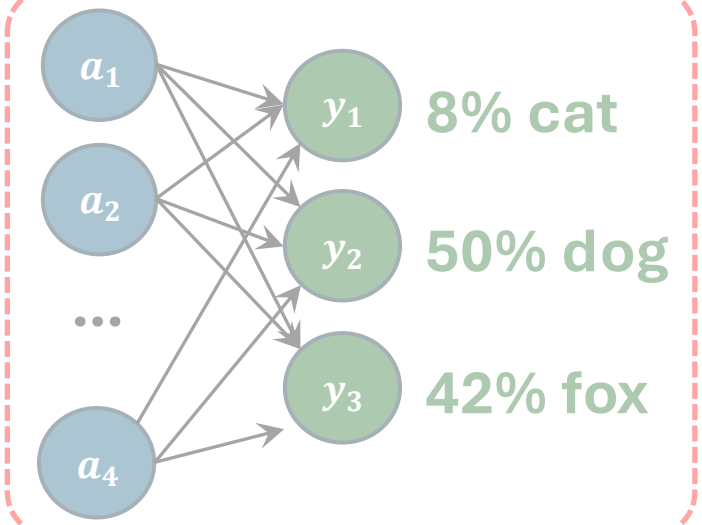
3. Different subset of training data



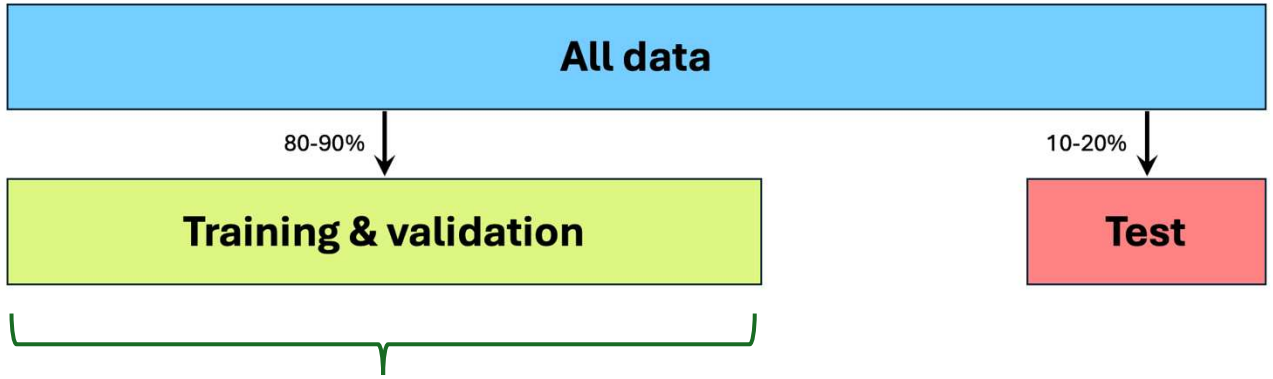

Then aggregate all the predictions in some way

# How to generate *different* models

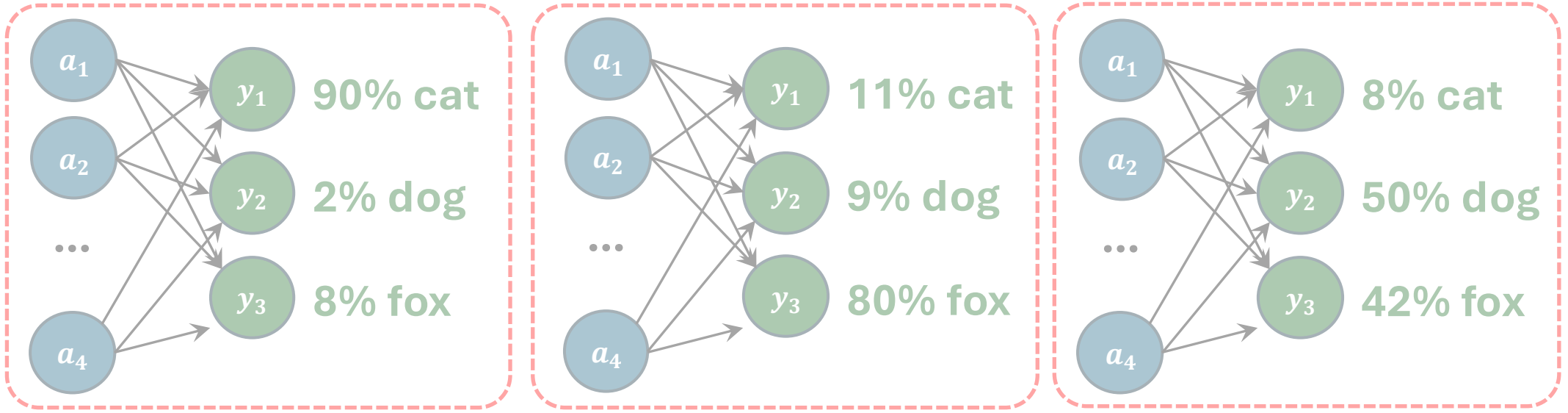
1. Different initialization

2. Different mini-batch ordering

**3. Different subset of training data**




Each model uses a **different part of the dataset for training *vs.* validation**




Then aggregate all the predictions in some way

# Aggregate predictions by averaging their outputs

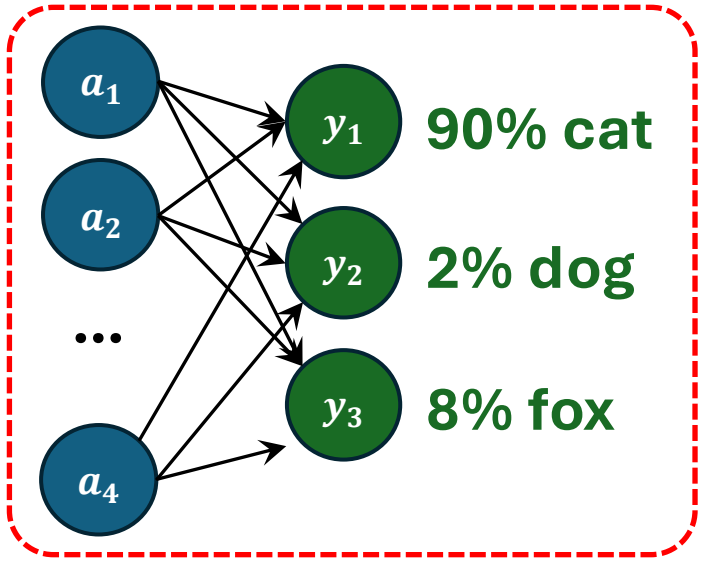
$$\bar{p}(x) = \frac{1}{M}\sum_{m=1}^{M} p_m(x)$$

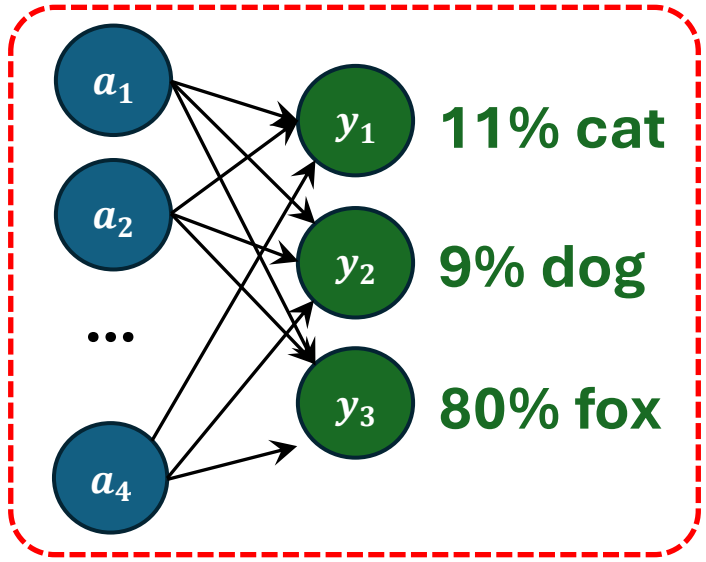
#1 #2 #3

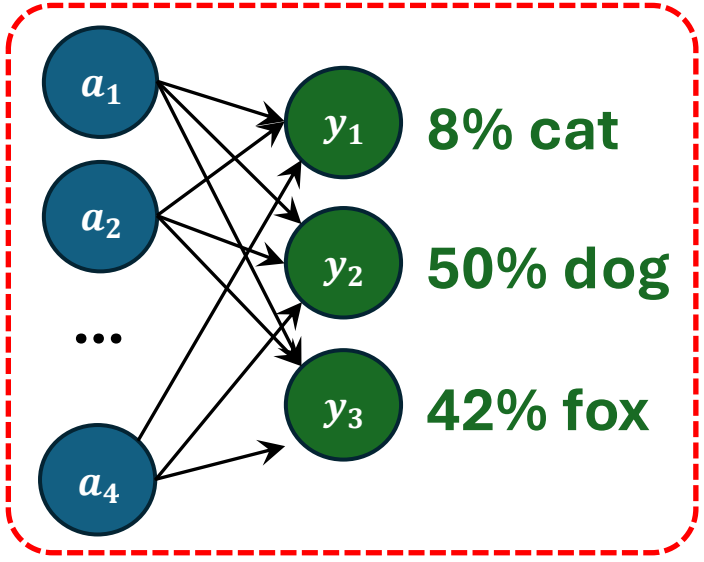
$$\bar{p}(\text{cat}|x) = \frac{1}{3}(0.90 + 0.11 + 0.08) \approx 0.363 \rightarrow 36.3\%$$

$$\bar{p}(\text{dog}|x) = \frac{1}{3}(0.02 + 0.09 + 0.50) \approx 0.203 \rightarrow 20.3\%$$

$$\bar{p}(\text{fox}|x) = \frac{1}{3}(0.08 + 0.80 + 0.42) \approx 0.433 \rightarrow 43.3\%$$




Then aggregate all the predictions in some way

# Aggregate predictions by averaging their outputs

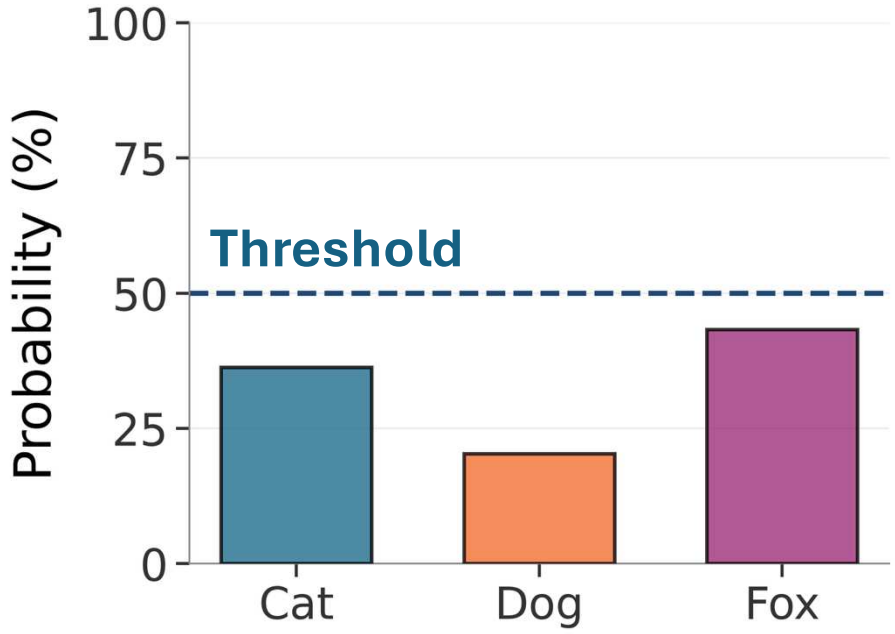
$$\bar{p}(x) = \frac{1}{M}\sum_{m=1}^{M} p_m(x)$$

$$\bar{p}(\text{cat}|x) = \frac{1}{3}\underbrace{(0.90}_{\#1} + \underbrace{0.11}_{\#2} + \underbrace{0.08}_{\#3}) \approx 0.363 \rightarrow 36.3\%$$

$$\bar{p}(\text{dog}|x) = \frac{1}{3}(0.02 + 0.09 + 0.50) \approx 0.203 \rightarrow 20.3\%$$

$$\bar{p}(\text{fox}|x) = \frac{1}{3}(0.08 + 0.80 + 0.42) \approx 0.433 \rightarrow 43.3\%$$

While there is still no "I don't know" output, you can set a **probability threshold**



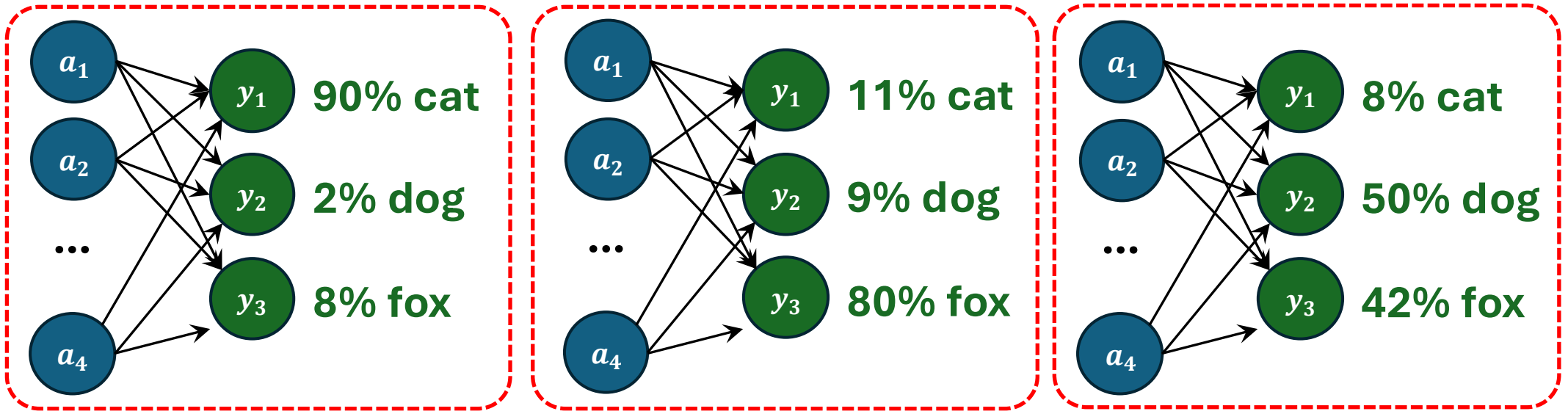

Even though each *individual* model was overly confident, **the *ensemble* was much more skeptical about any specific class!**

# Deep ensembles are nice but expensive

$$\bar{p}(x) = \frac{1}{M}\sum_{m=1}^{M} p_m(x)$$

We have to separately **train $M$ different models**. Around 5-10 is common…While more is better from a "diversity" perspective, the **training cost increases linearly with the number of models in the ensemble…**

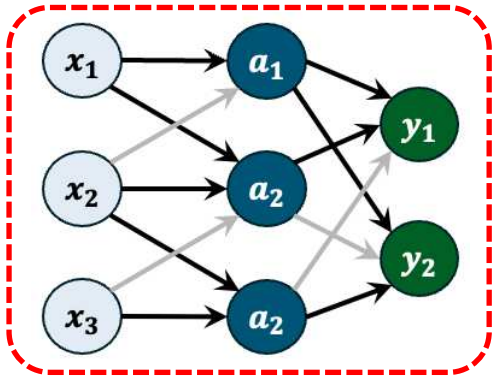
Any way around this? To get diverse predictions without having to train separate models?



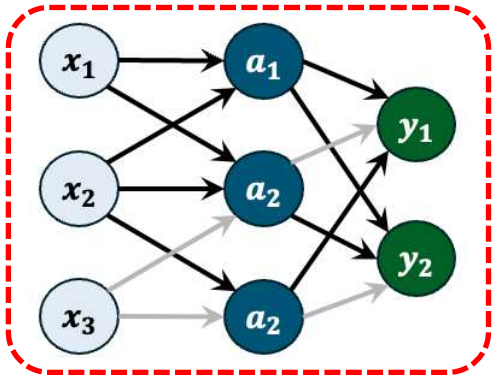

**Then aggregate all the predictions in some way**

# The solution: Monte Carlo (MC) dropout

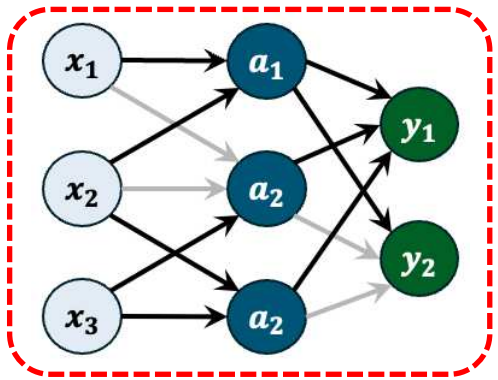
**Forward Pass #1**



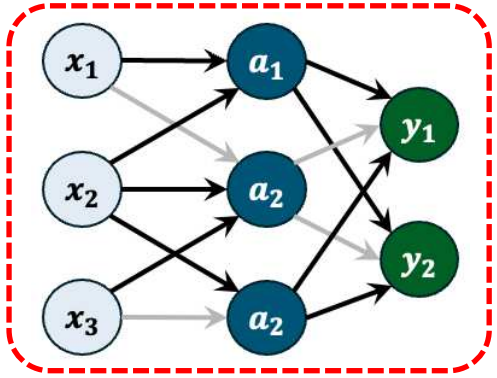

**Forward Pass #2**



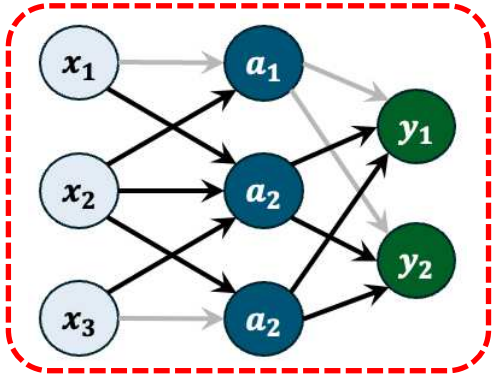

**Forward Pass #3**




**Forward Pass #4**




**Forward Pass #5**




With **dropout**, we randomly exclude some connections ***during training***

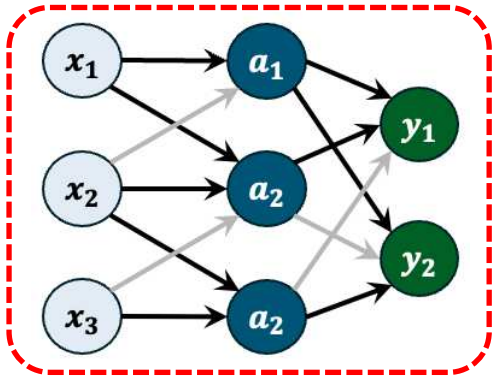
But then after training, ***at inference***, we use **all connections** to make a prediction

In MC dropout, we also **randomly set some % of weights $w_{ij}^{(l)} = 0$ during inference**, and then aggregate the outputs

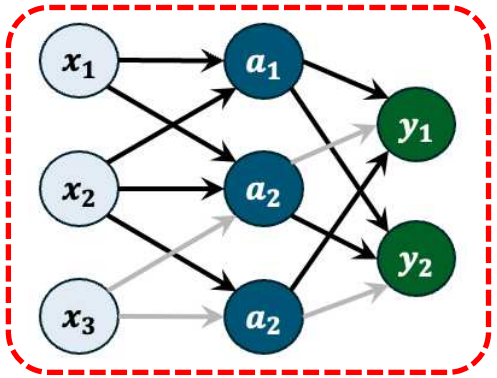
Each prediction should differ based on which weights are kept active, **yet only one model needed to be trained!**

# The solution: Monte Carlo (MC) dropout

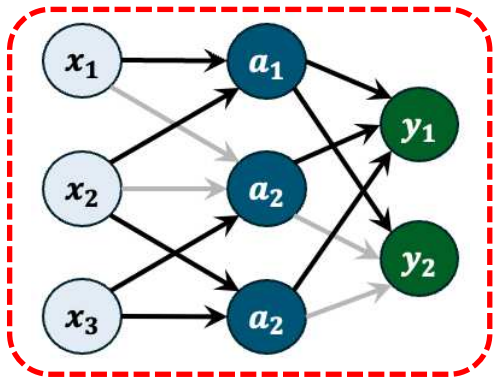
**Forward Pass #1**



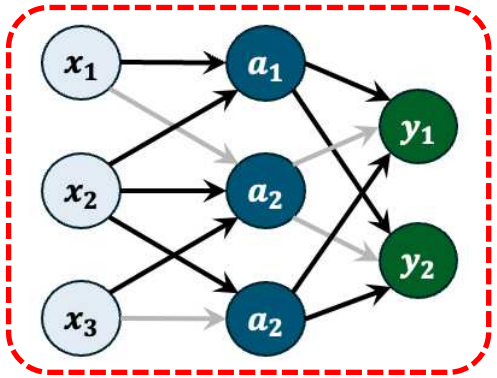

**Forward Pass #2**




**Forward Pass #3**




**Forward Pass #4**




...

We aggregate using the same approach. Simply average over the outputs:

$$\bar{p}(x) = \frac{1}{M}\sum_{m=1}^{M} p_m(x)$$

- **Now $M$ can be much larger (~100s)** because inference is (usually) cheap relative to training.
- Dropout rates of around 10-20% per layer are typical. You should generally **match the rates used during training.** Since the loss was minimized using these rates, each model should remain fairly reliable despite not leveraging all its connections.

# Confidence *vs.* uncertainty

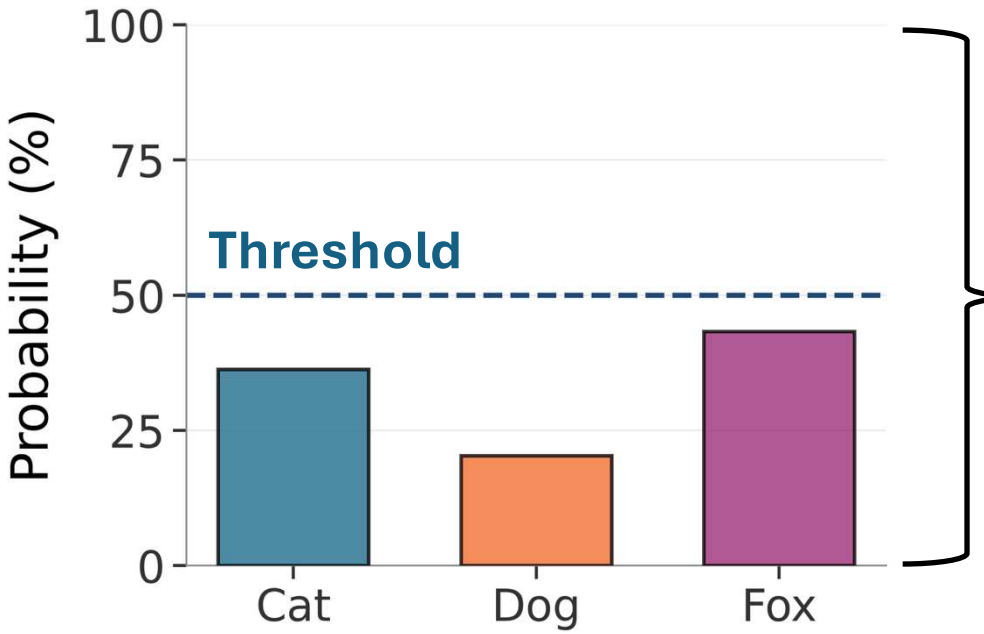
**Confidence alone may not tell the whole story** – consider two ensembles:

**Ensemble A:** $p_{\text{dog}} = 0.88, 0.90, 0.89, 0.91 \rightarrow$ **confidence for dog is 89.5%**

**Ensemble B:** $p_{\text{dog}} = 0.99, 0.95, 0.75, 0.89 \rightarrow$ **confidence for dog is 89.5%**

**Same confidence, but more disagreement in Ensemble B**

$$\bar{p}(x) = \frac{1}{M}\sum_{m=1}^{M} p_m(x)$$




If we measure **confidence** as the probability of the top class…Is that enough to gauge trustworthiness?

# Uncertainty as standard deviation

**Confidence alone may not tell the whole story** – consider two ensembles:

**Ensemble A:** $p_{\text{dog}} = 0.88, 0.90, 0.89, 0.91$

**Ensemble B:** $p_{\text{dog}} = 0.99, 0.95, 0.75, 0.89$

$$\bar{p}_{\text{dog}} = 0.895 \pm 0.011$$

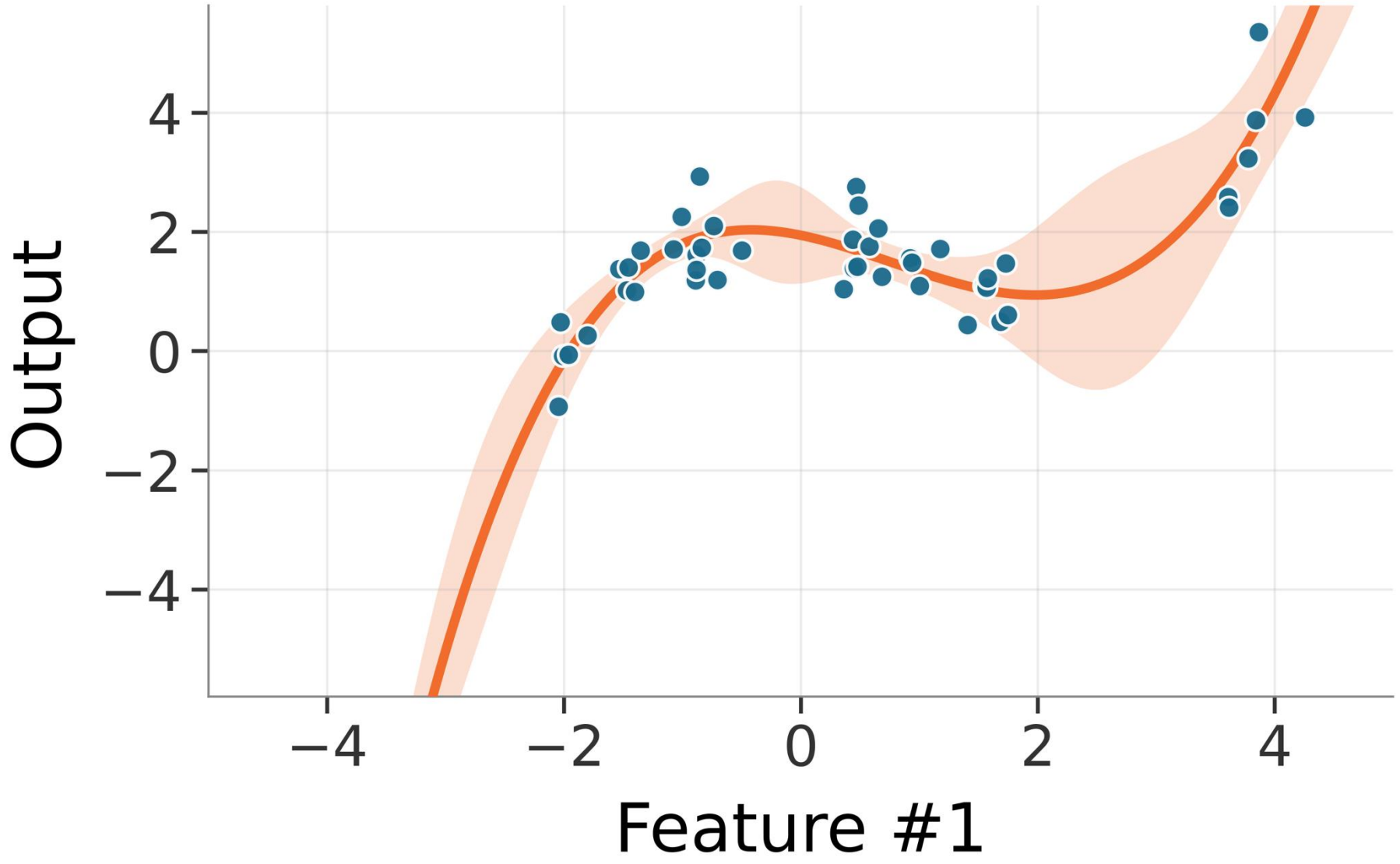
$$\bar{p}_{\text{dog}} = 0.895 \pm 0.091$$

**Report average and uncertainty!**

$$\sigma = \sqrt{\frac{1}{M}\sum_{m=1}^{M}\left(p_{m,\text{dog}} - \bar{p}_{\text{dog}}\right)^2}$$

- $p_{m,\text{dog}}$ is the prediction of each individual model
- $\bar{p}_{\text{dog}}$ is the average prediction of the entire ensemble

# Uncertainty is also very useful for regression

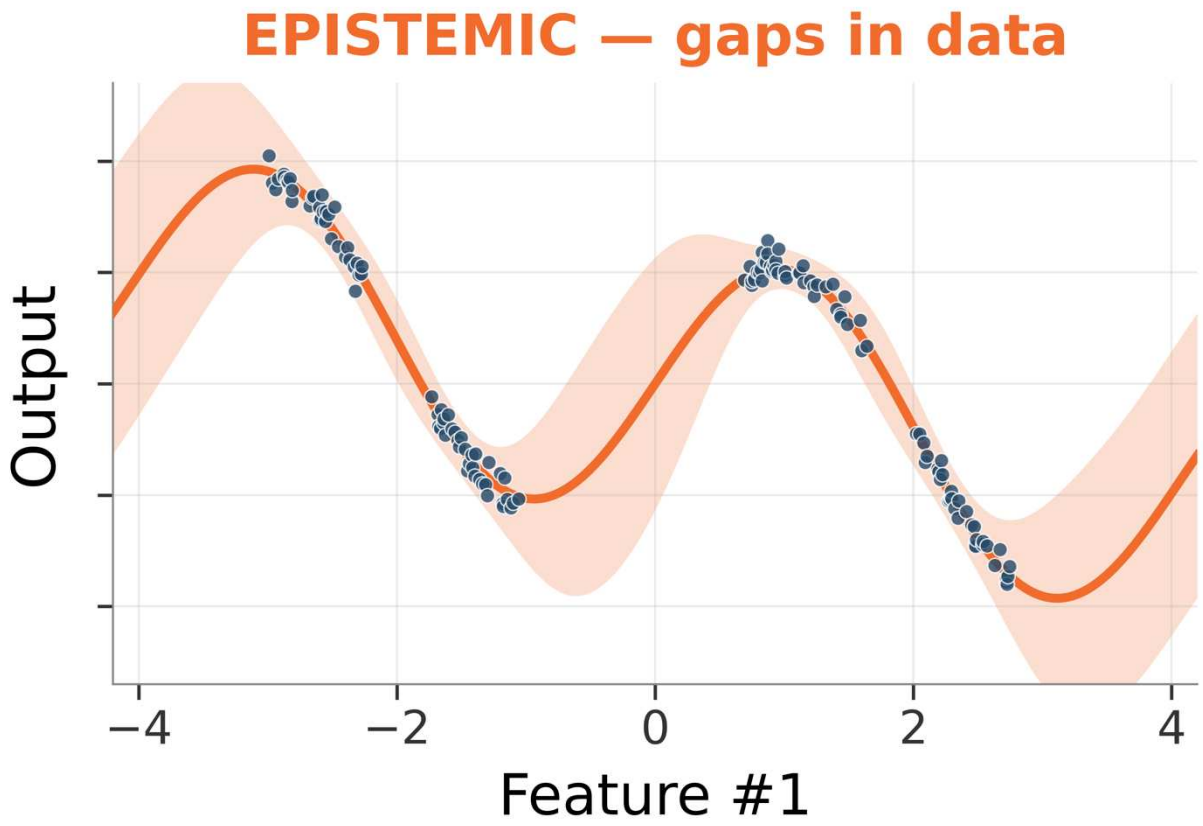


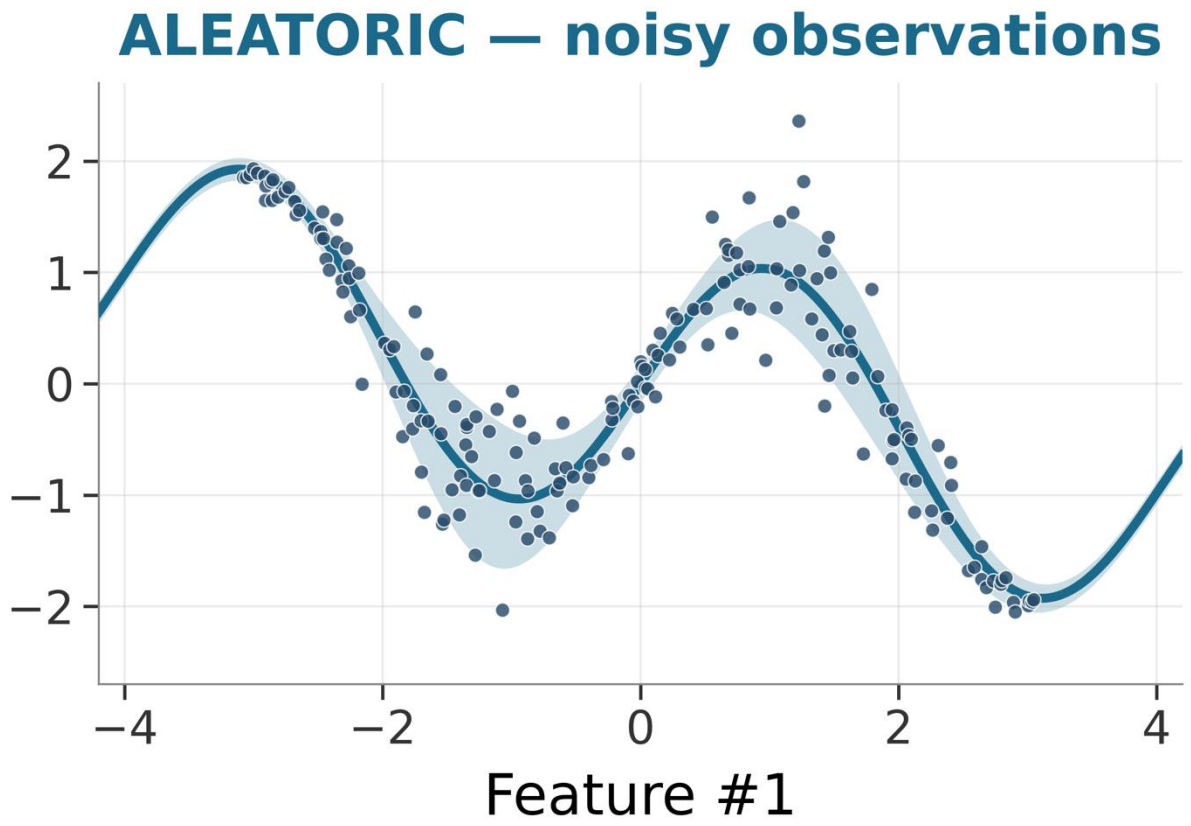

The solid curve measures the average ensemble prediction at each $x$-value

The shaded region measures the **uncertainty***

***What kind?**

# Epistemic *vs.* aleatoric uncertainty




- Deep ensembles and MC dropout capture **epistemic** uncertainty, not aleatoric
- To reduce epistemic uncertainty, **collect more data!**
- For aleatoric, not much we can do…Besides collecting better/cleaner data

# Questions?

# What to know for the final exam

- **Saliency:**
  - Understand it conceptually (what it tells you)
  - **You should be able to do a saliency calculation (backprop)**
- **Class Activation Maps:** Just know the concepts, no calculations needed
- The difference between output probability and confidence
- Interpret ECE plots and perform appropriate temperature scaling
- Concepts: Deep ensembles and Monte Carlo dropout
- **Epistemic *vs.* aleatoric uncertainty**