

DETERMINISTIC
DREW



It will
be ...

MON

TUE

WED

THU

FRI

SAT

dry ☀️

dry ☀️

rain 💧

dry ☀️

rain 💧

dry ☀️

10% ☀️

20% ☁️

80% ☁️

30% ☁️

60% ☁️

10% ☀️

The
chance of
rain is ...

PROBABILISTIC
PARKER



How the day actually turned out:

dry ☀️

rain 💧

rain 💧

dry ☀️

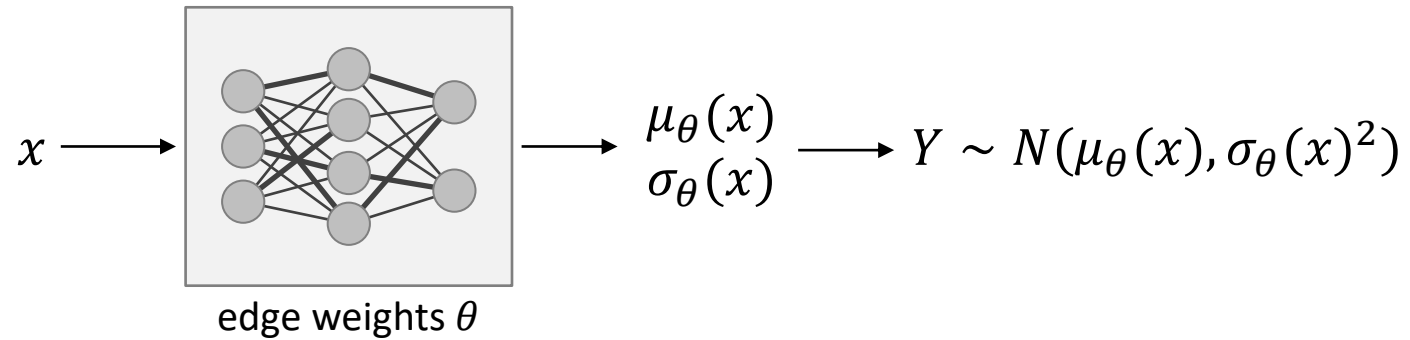
dry ☀️

dry ☀️

QUESTION. Who is the better forecaster?
(N.b. they both made their forecasts at the same time.)

EXAMPLE

We have a neural network that makes probabilistic predictions about a response Y given predictor variable x .

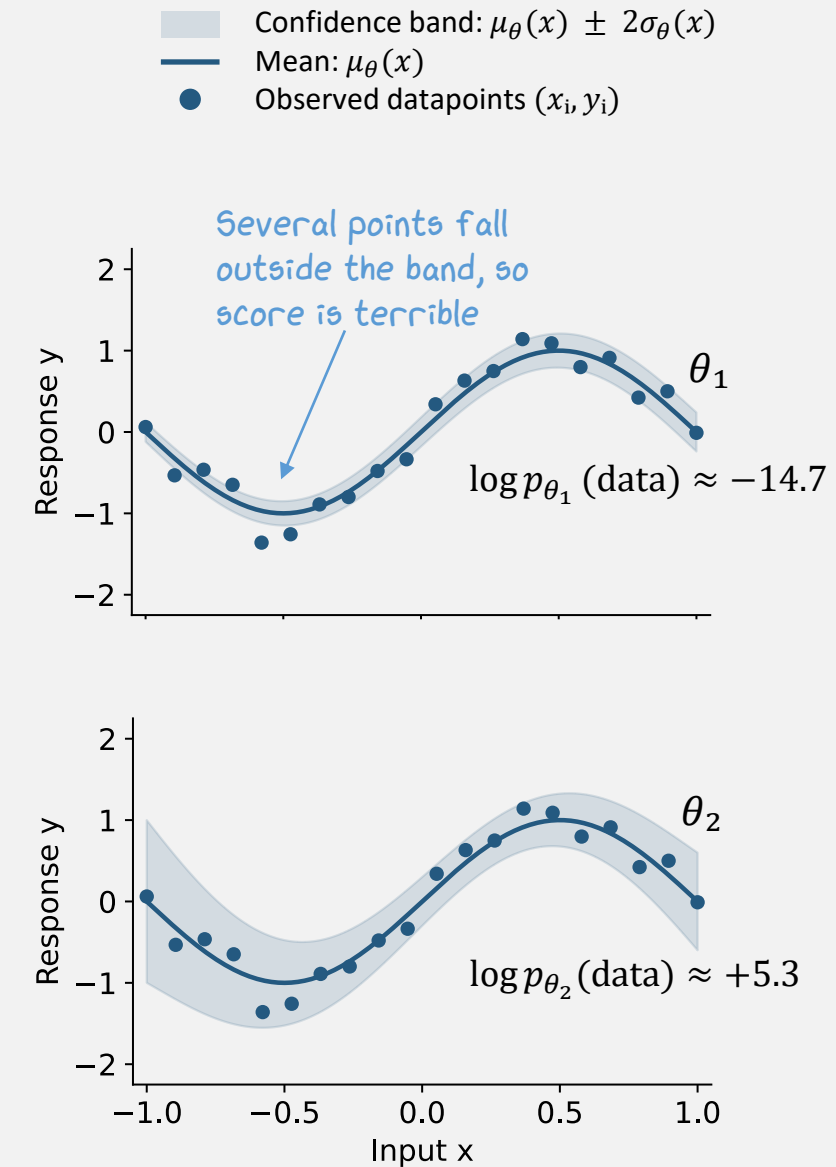


We observe 20 datapoints (x_i, y_i) . Two sets of weights, θ_1 and θ_2 , give the predictions plotted on the right.

How would you decide which is better, θ_1 or θ_2 ?

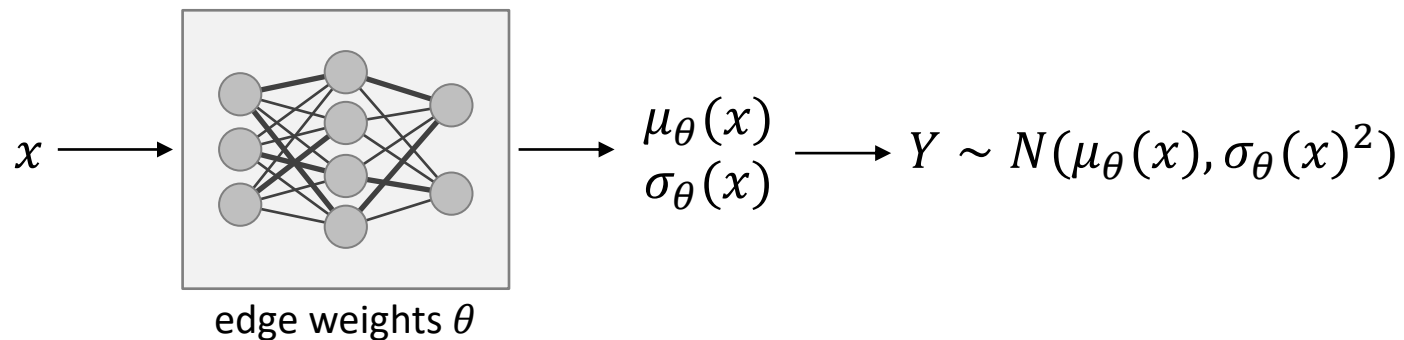
This is just a probabilistic model!

We know how to score probabilistic models: $\log p_{\theta}(\text{data})$, or $\text{KL}(\hat{q} \parallel p_{\theta})$.



EXAMPLE

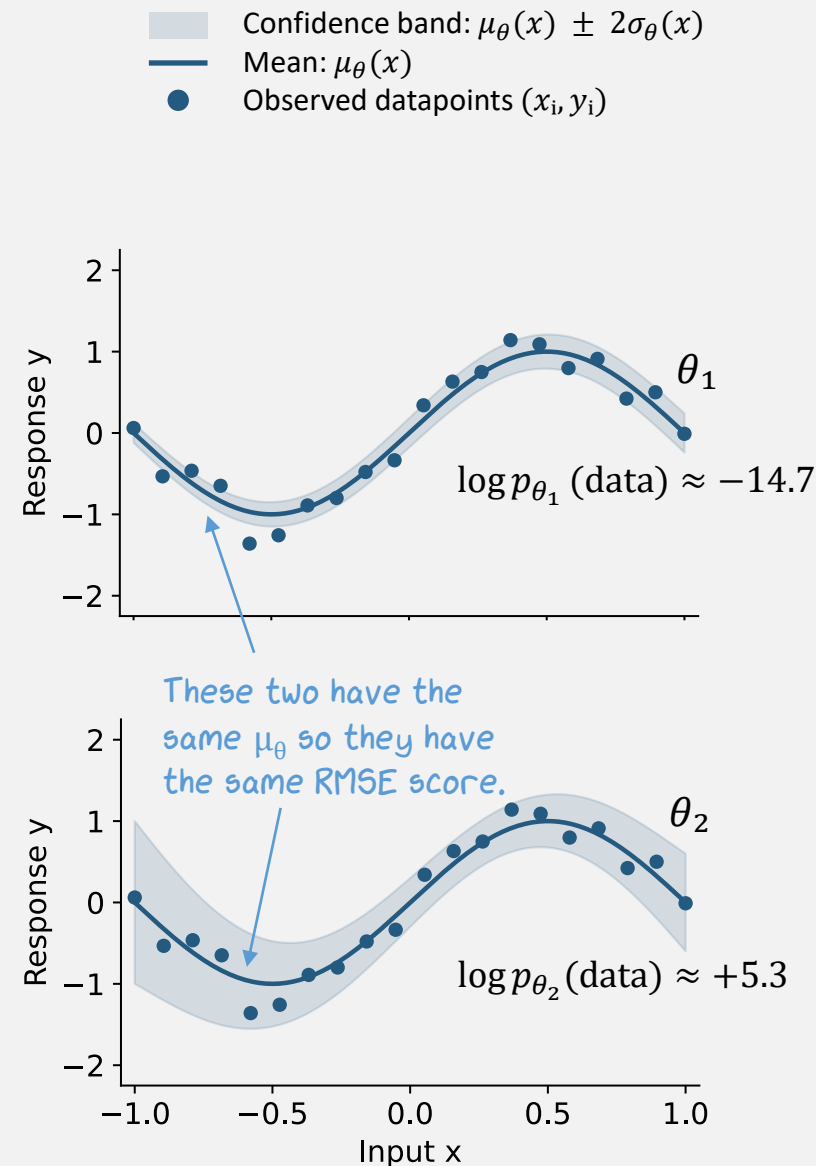
We have a neural network that makes probabilistic predictions about a response Y given predictor variable x .



We observe 20 datapoints (x_i, y_i) . Two sets of weights, θ_1 and θ_2 , give the predictions plotted on the right.

How would you decide which is better, θ_1 or θ_2 ?

$$\log p_{\theta}(\text{data}) = - \sum_{i=1}^n \left[\underbrace{\frac{1}{2} \log(2\pi\sigma_{\theta}(x_i)^2)}_{\text{tighter band = better}} + \underbrace{\frac{[y_i - \mu_{\theta}(x_i)]^2}{2\sigma_{\theta}(x_i)^2}}_{\substack{\text{more accurate = better} \\ \text{don't fuss about noisy regions}}} \right]$$



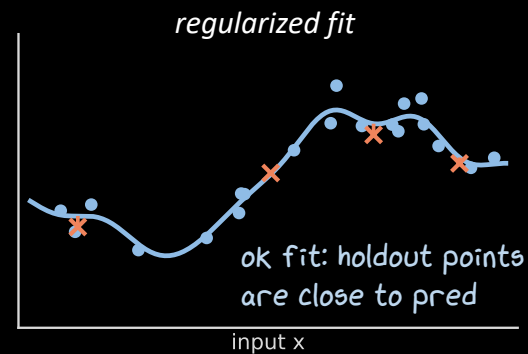
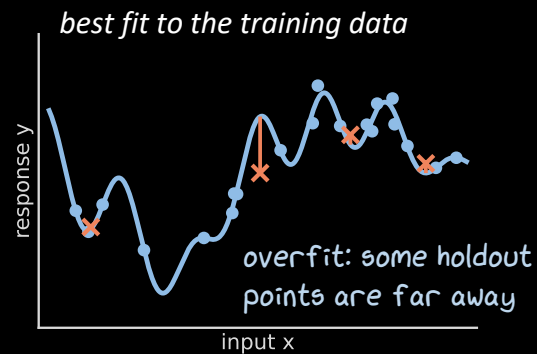
If we had chosen a model with a single shared σ_{θ} for all x , the accuracy term would look like RMSE.

Behind most loss functions there's a probability model. cross-entropy loss $\equiv$ Categorical random variable



What matters is holdout accuracy.
Regularize to achieve this.

$$x \rightarrow \boxed{f_\theta} \rightarrow \hat{y} = f_\theta(x) \text{ evaluated by accuracy}$$

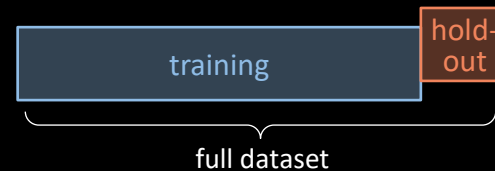


Naively maximizing accuracy on the training data usually gives poor holdout performance (overfitting).

We care about holdout performance, since it's a good proxy for in-the-wild performance.

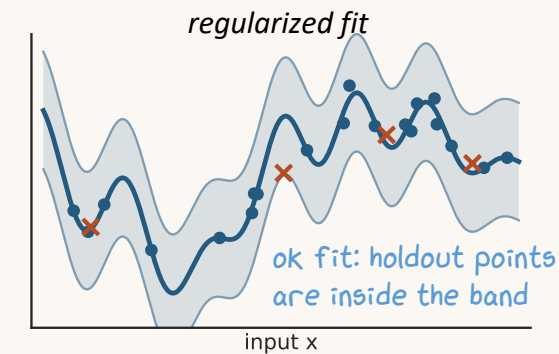
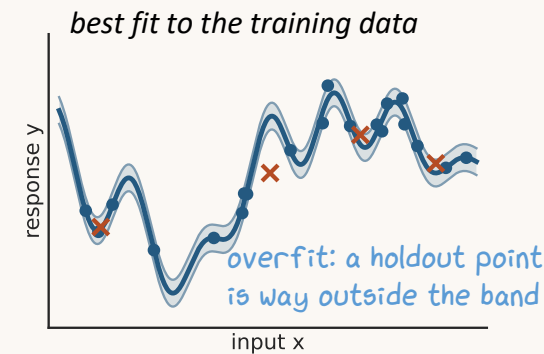
Pragmatic solution: regularize the fit.

e.g. minimize $\sum_i [y_i - f_\theta(x_i)]^2 + \lambda \|\theta\|^2$



What matters is holdout log likelihood.
Regularize to achieve this.

$$x \rightarrow \boxed{\mu_\theta, \sigma_\theta} \rightarrow Y \sim N(\mu_\theta(x), \sigma_\theta(x)^2) \text{ evaluated by log lik}$$



Naively maximizing log likelihood on the training data usually gives poor holdout performance (overfitting).

We care about holdout performance, since it's a good proxy for in-the-wild performance.

Pragmatic solution: regularize the fit.

e.g. maximize $\sum_i \log p_\theta(y_i | x_i) - \lambda \|\theta\|^2$



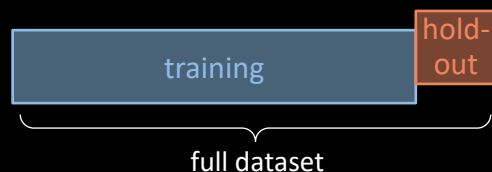


What matters is holdout accuracy.
Erm... wait, what?

Task: an image generator, for generating face pictures.



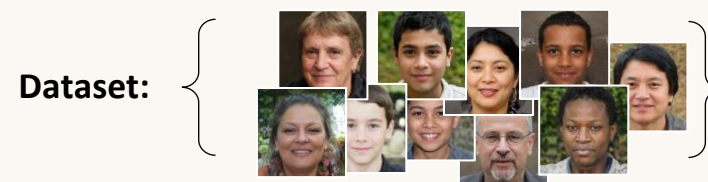
Generative model:



What matters is holdout log likelihood.
Regularize to achieve this.



Task: an image generator, for generating face pictures.



Generative model:

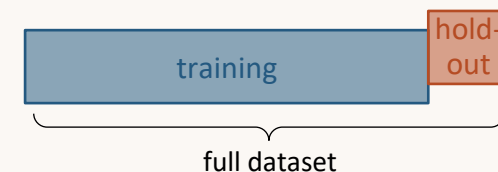


Naively maximizing log likelihood on the training data usually gives poor holdout performance (overfitting).

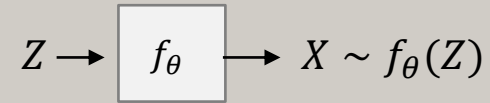
We care about holdout performance, since it's a good proxy for in-the-wild performance.

Pragmatic solution: regularize the fit.

e.g. maximize $\sum_i \log p_\theta(\cancel{y_i} | x_i) - \lambda \| \theta \|^2$



GENERATIVE MODELLING



Log likelihood of a datapoint: $\log p_\theta(x)$

KL optimization: $\min_{\theta} \text{KL}(\text{Pr}_X^{\text{true}} \parallel p_\theta) \approx \min_{\theta} \{\text{const} - n^{-1} \sum_i \log p_\theta(x_i)\}$

SUPERVISED LEARNING



Log likelihood of a datapoint: $\log p_\theta(y|x)$

ANOTHER WAY TO SEE SUPERVISED LEARNING

Generate $X \sim r$ then $(Y|x) \sim p_\theta(\cdot | x)$

Log likelihood of a datapoint: $\log r(x) + \log p_\theta(y|x)$

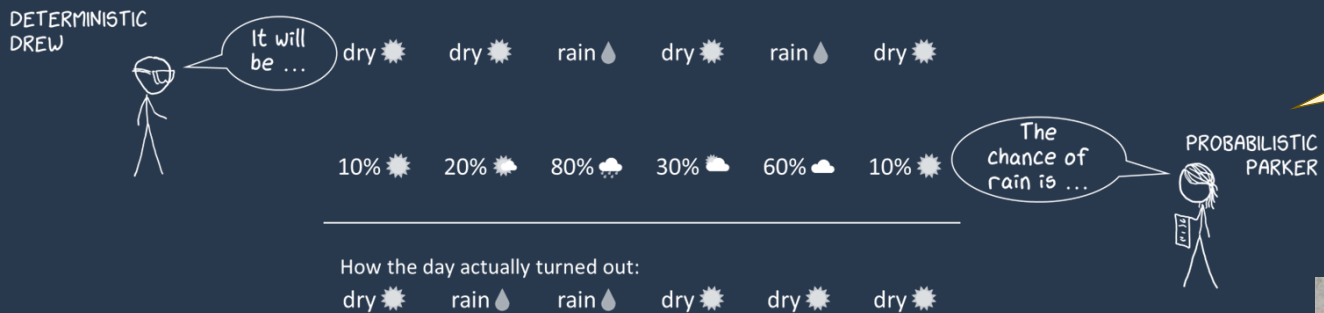
KL optimization: $\min_{r, \theta} \text{KL}(\text{Pr}_{X,Y}^{\text{true}} \parallel r p_\theta) \approx \min_{\theta} \{\text{const} - n^{-1} \sum_i \log p_\theta(y_i|x_i)\}$

The optimal r matches the true distribution of X , and drops out of the KL equation
[we did the algebra for this in the first half]

The goal is the same in both cases:
we want $\text{KL}(\text{Pr}^{\text{true}} \parallel \text{Pr}^{\text{model}})$ to be small.

Are the supervised and generative cases really that different, after all?

$$\sum_i \log p_\theta(y_i | x_i)$$



What is Probabilistic Parker's KL score?

$$\text{KL} \approx \text{const} - (\log 0.9 + \log 0.2 + \dots)/6$$

$$= \text{const} + 0.553$$

QUESTION. Who is the better forecaster?
(N.b. they both made their forecasts at the same time.)



QUESTION. Who paints ceilings better?

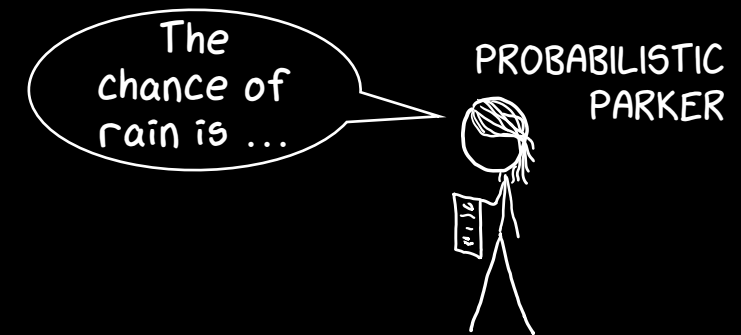
They're doing different things!
Comparing them is answering the wrong question.



- ❖ Neural networks can be seen as probability models
(and this describes all types of learning activity)
- ❖ The goal is to get a good holdout KL score
- ❖ For fitting, regularize the training KL score

You've tried out several different probabilistic models.
You've picked the one with the best holdout KL score.
It is $KL = \text{const} + 0.553$.

Is that low enough?



Holdout KL score can be used to *compare* models, but not to *rate* them.

We have a holdout dataset $\{x_1, \dots, x_n\}$, which we take to be sampled from some unknown distribution q .

The holdout score of our proposed model p is

$$\begin{aligned}\text{KL}(q \parallel p) &= \mathbb{E}_{X \sim q} \log \frac{q(X)}{p(X)} \\ &= \mathbb{E}_{X \sim q} \log q(X) - \mathbb{E}_{X \sim q} \log p(X) \\ &\approx \underbrace{\mathbb{E}_{X \sim q} \log q(X)}_{\text{const}} - \frac{1}{n} \sum_{i=1}^n \log p(x_i)\end{aligned}$$

The holdout score of a competitor model $\tilde{p}$ is

$$\text{KL}(q \parallel \tilde{p}) \approx \underbrace{\mathbb{E}_{X \sim q} \log q(X)}_{\text{same const}} - \frac{1}{n} \sum_{i=1}^n \log \tilde{p}(x_i)$$

We can compare models by holdout KL
(i.e. pick the better-scoring model)
even though we don't know the const.

But we can't measure how close the KL is to zero.

How else might we judge whether our
best-model-so-far is good enough?

Technical note. Why “holdout”?

The Monte Carlo approximation is only valid if p is a fixed function, not if it depends on the sampled x_i .

KL is a good general-purpose metric.
But it might not agree with the specific metric your application needs.

APPLICATION: MODEL CALIBRATION
Guo, Pleiss, Sun and Weinberger, ICML 2017

The task. We want a human-ML hybrid system for classifying images. We first use a model to estimate probabilities for each class. Then

- if our model gives probability ≥ 0.9 to a class, label it with that class
- otherwise, ask a human to label it

Starting point. Several image classifiers, trained and evaluated on CIFAR-100:

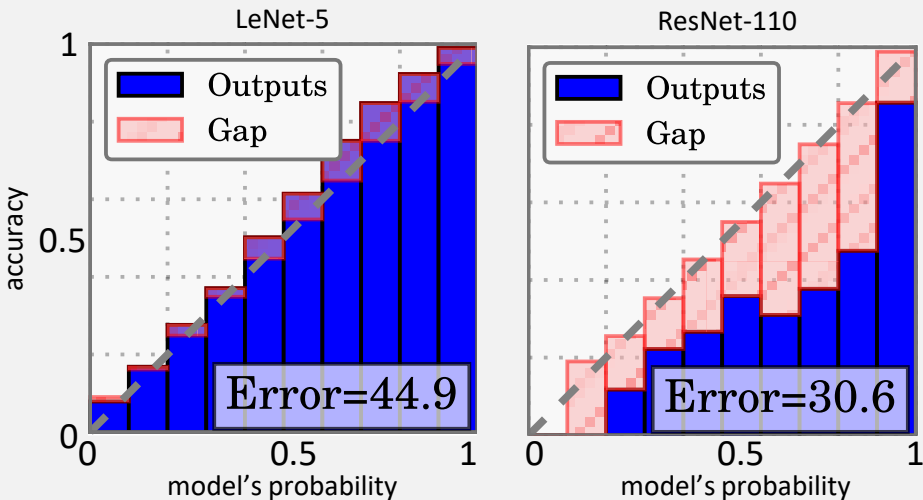
model	holdout NLL
ResNet-110 (2016), 110 layers	1.50
LeNet-5 (1998), 5 layers	1.66
⋮	⋮

Which of these models, if any, should we use?

QUESTION. Of the images that the model labels on its own, what fraction do we expect it will get right?

What this application needs. We'd prefer a model that is accurate both about the class and about the class probability. We want its "90%" to mean "90%". (This is called "calibration".)

Experiment. Ask the model for its most likely class, and the class probability. Group holdout data into bins based on this class probability. Measure accuracy per bin.



Conclusion. Doing well on holdout KL / log likelihood does not mean doing well on calibration.

There's no reason it should! Calibration is a batch property (computed for a bin), but log likelihood is a per-datapoint property.

OPINION: When KL disagrees with our application's custom metric, revise our model but stick with KL-based training.

Shouldn't we just work with our custom metric from the start, rather than KL?

No.

- Often, we don't know what the downstream task is. Train a model once with KL, use for many purposes.
- Sometimes the custom metric is hard to train with. KL is a per-item score, excellent for back-propagation.
- KL is an aggregate measure, and it registers deviation from the truth *in any direction*. So it always pushes us towards the truth.
- If KL and our custom metric disagree, the custom metric is picking up one specific direction of deviation from the truth. Revise the *model* so that it has trainable parameters to fix this deviation, and KL will push to set those parameters right — and then KL will agree with our custom metric.

Guo et al. (ICML 2017) added a parameter at the end of the network, that tweaked the balance of probabilities. With this parameter, **the best-KL-scoring model was the best-calibrated model**.

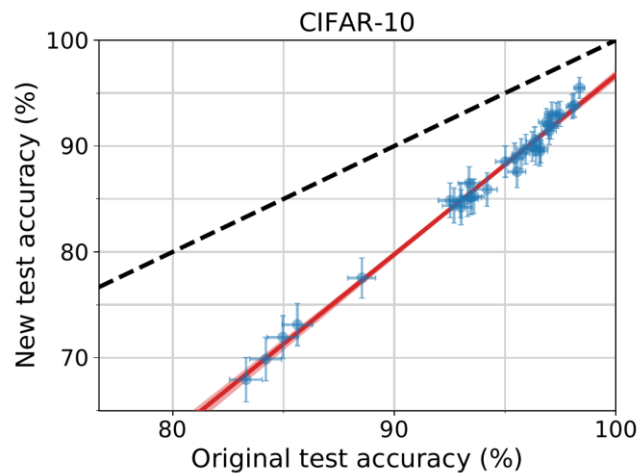
Holdout KL is only as good as your holdout set

REPRODUCIBILITY / GENERALIZATION

Recht, Roelofs, Schmidt and Shankar, ICML 2019

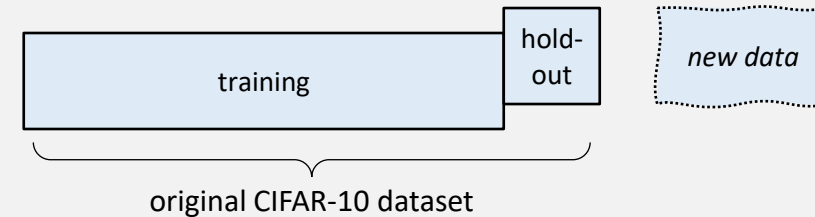
CIFAR-10 is a classic dataset of images from 2009.

The study. Collect 2000 new images from the same source and using the same protocol as CIFAR-10. Evaluate 30 published classifiers on them.



Their findings.

- Models that did better on CIFAR-10 did better on the new dataset, so it's probably not “evolutionary overfitting” to the CIFAR-10 holdout set.
- Performance on the *easy* subset of the new images was the same as performance on the original dataset, so it's probably because the new dataset's distribution has shifted from the original distribution.



The implicit claim of holdout evaluation is that **holdout performance is a proxy for deployment-time performance**. That's only true to the extent that your holdout set is a good match.

- Can we detect if the image distribution has changed? §2
- Can we build a classifier that's robust against distribution change? §8

- ❖ Neural networks can be seen as probability models
- ❖ KL is all you need (if you use it properly)

It applies equally well
to supervised and
generative modelling

and uncertainty,
and causality,
and ...

Use it (regularized)
for training

- it's a simple
per-item score
- it picks up all
sorts of model
weaknesses

Use holdout
KL to compare
models

the right sort
of holdout

If it doesn't agree with
metrics of interest,
then fix your model,
not your training
objective
(IMHO)