Model interpretability, generative models, and uncertainty

Recitation 5

MIT - 6.802 / 6.874 / 20.390 / 20.490 / HST.506 - Spring 2019

Sachit Saksena

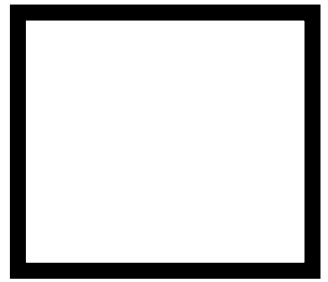
Model interpretability: model interpretation overview

- Adoption of deep learning has led to:
  - Large increase in predictive capabilities
  - Complex and poorly-understood black-box models

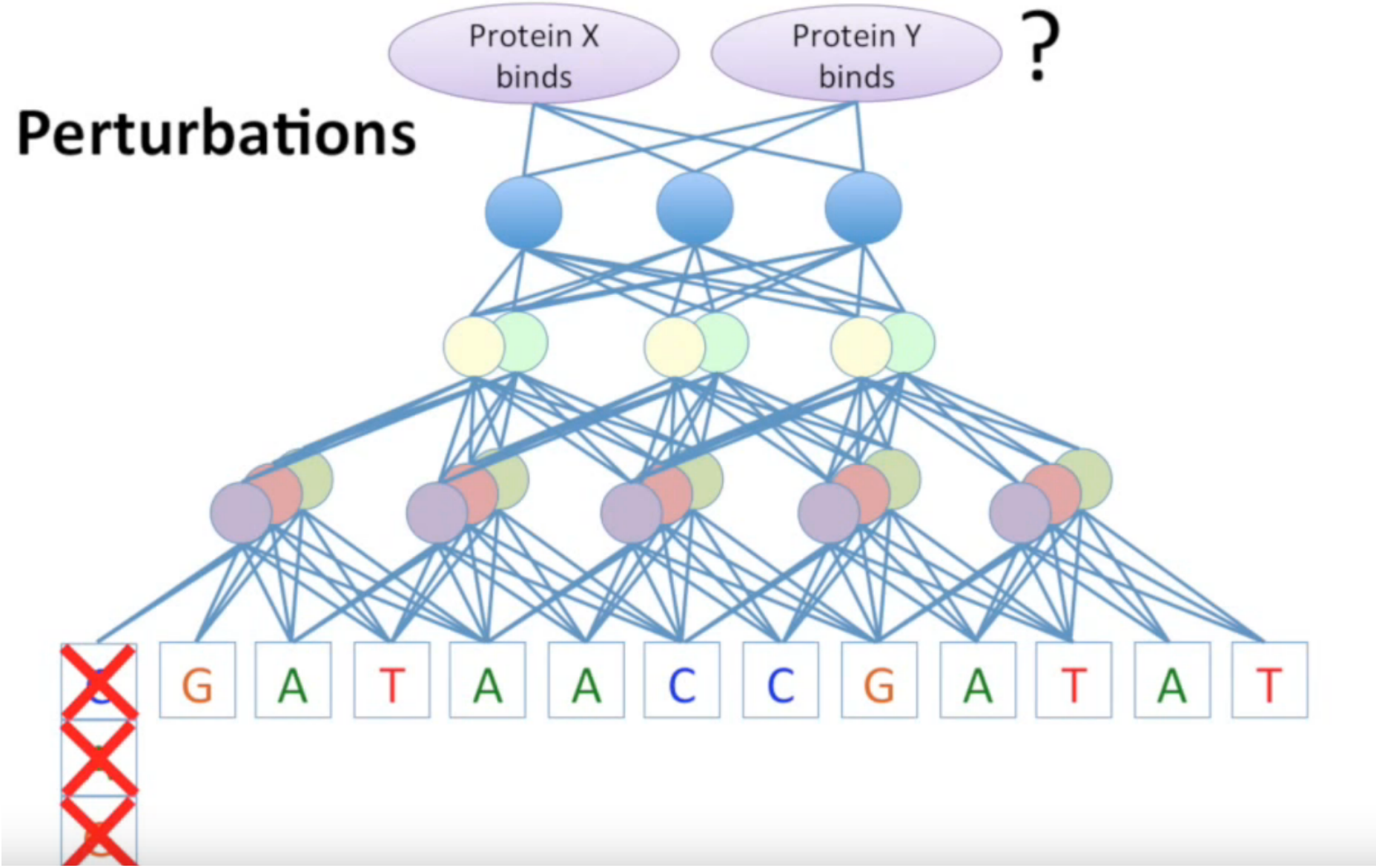
- Imperative that certain model decisions can be interpretably rationalized
  - Ex: loan-application screening, recidivism prediction, medical diagnoses, autonomous vehicles

- Explain model failures and improve architectures

- Interpretability is also crucial in scientific applications, where goal is to identify general underlying principles from accurate predictive models



Model interpretability: black-box models



Model interpretability: sufficient input subsets

SIScollection$(f, \mathbf{x}, \tau)$

$S = [p]$
for $k = 1, 2, \ldots$ **do**
  $R =$ **BackSelect**$(f, \mathbf{x}, S)$
  $S_k =$ **FindSIS**$(f, \mathbf{x}, \tau, R)$
  $S \leftarrow S \backslash S_k$
if $f(\mathbf{x}_S) < \tau$: **return** $S_1, \ldots, S_{k-1}$

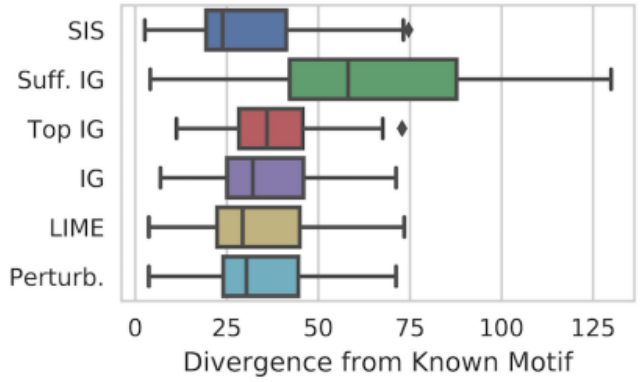
BackSelect$(f, \mathbf{x}, S)$

$R =$ empty stack
while $S \neq \varnothing$ **do**
  $i^* = \text{argmax}_{i \in S} f(\mathbf{x}_{S \backslash \{i\}})$
  Update $S \leftarrow S \backslash \{i^*\}$
  Push i^* onto top of R
return R

FindSIS$(f, \mathbf{x}, \tau, R)$

$S = \varnothing$
while $f(\mathbf{x}_S) < \tau$ **do**
  Pop i from top of R
  Update $S \leftarrow S \cup \{i\}$
if $f(\mathbf{x}_S) \geqslant \tau$: **return** S
else: **return** *None*

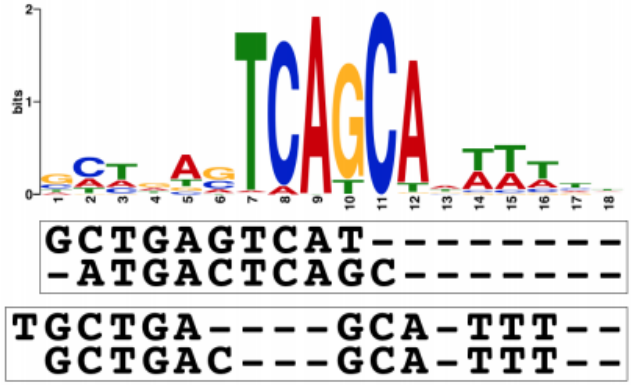
(a)



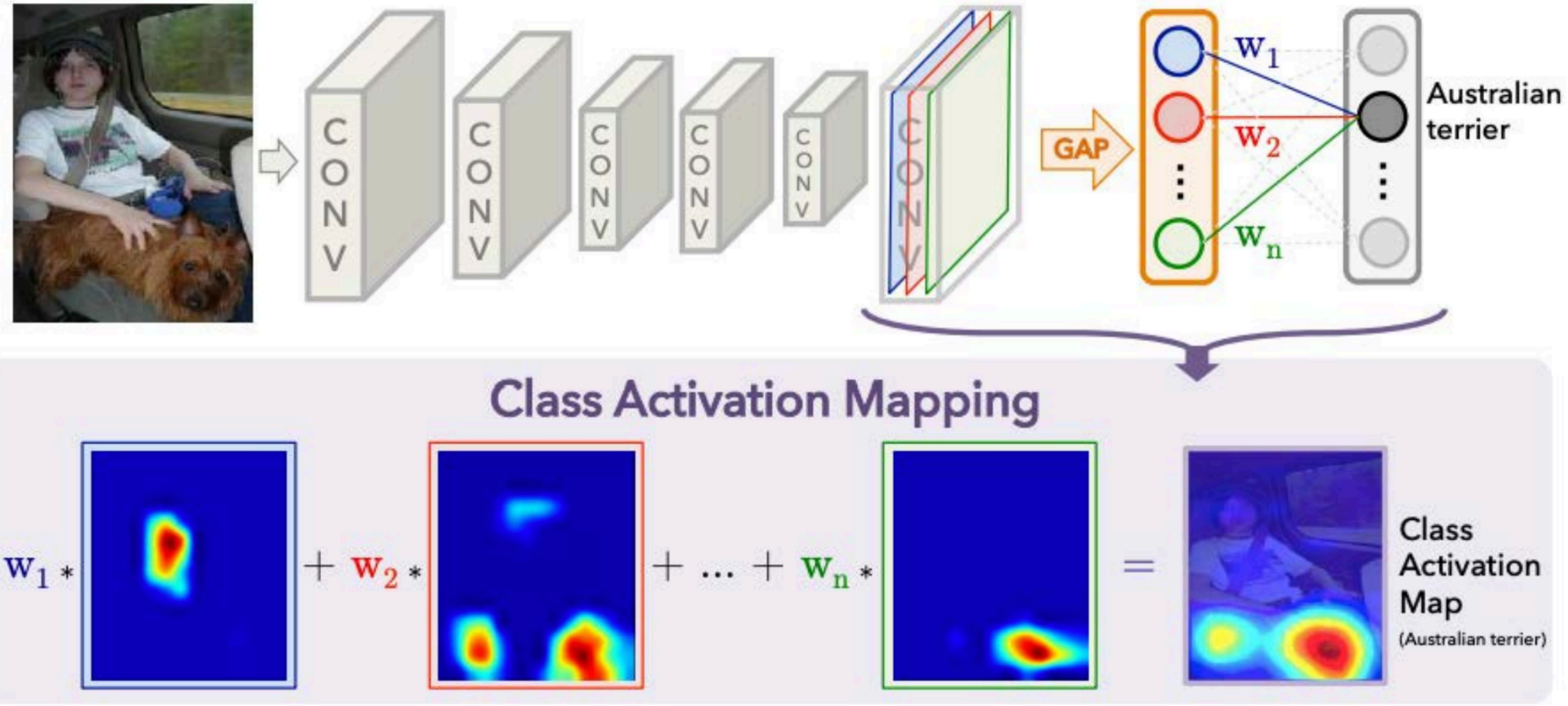
(b)

SIS	Freq.
GCTGAGTCAT	197
ATGACTCAGC	185
GCTGAGTCA-C	83
GCTGAGTCAC	53
GCTGACTCAGCA	42
SIS	**Freq.**
TGCTGA--GCA-TTT	12
GCTGAC--GCA-TTT	8
TGCTGAC--GCA-TT	6
TGCTGAC--GCA-AA	5
TGCTGAC--GCA-AT	4

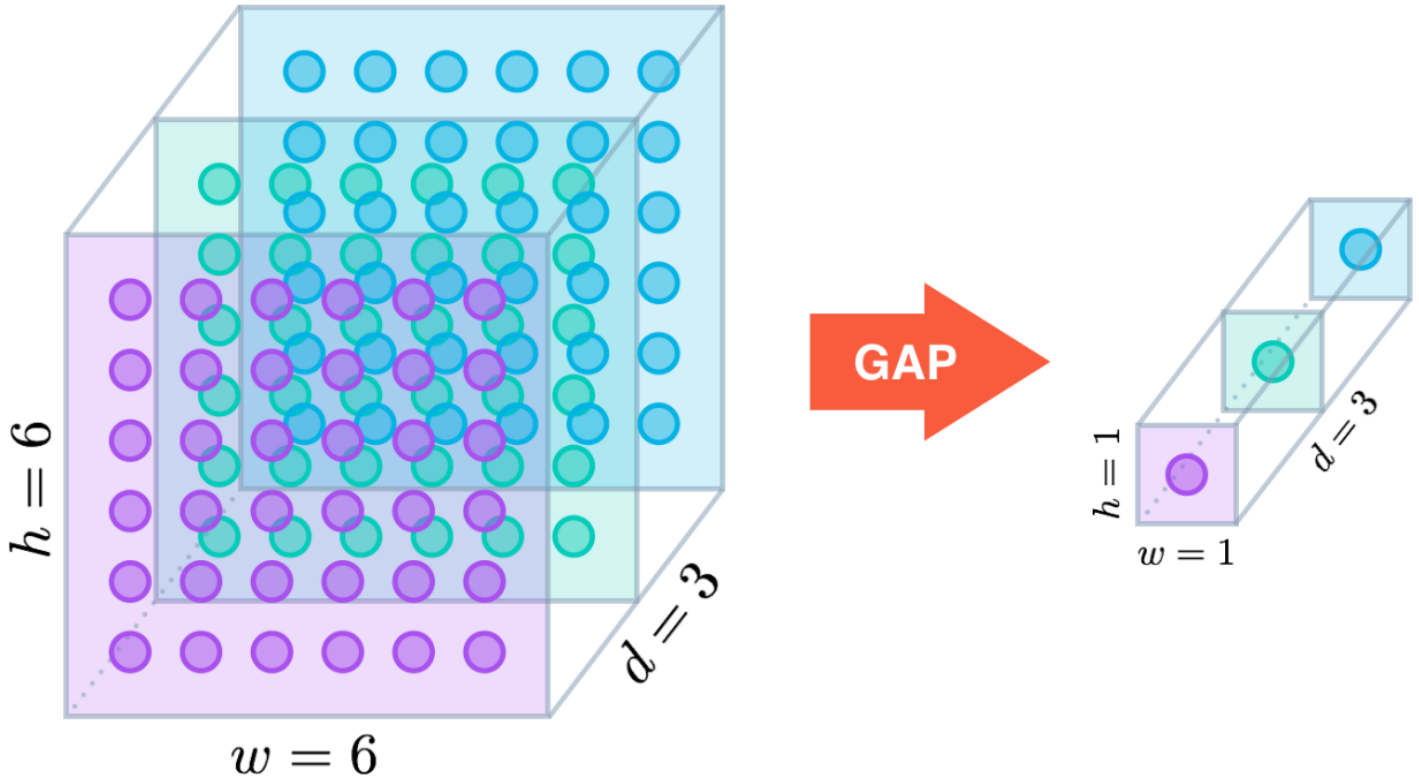
(c)



Model interpretability: class activation mapping

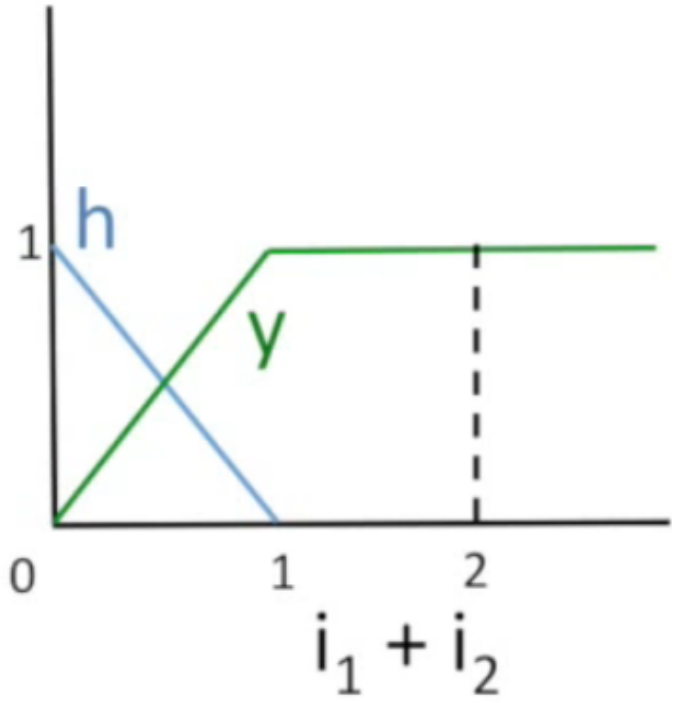


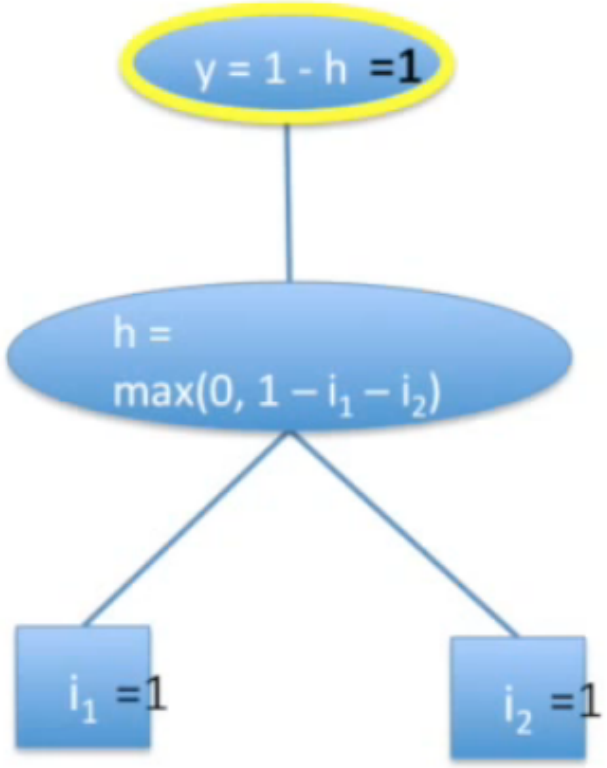
Model interpretability: global activation pooling



Model interpretability: saturation problem

$y = (i_1 + i_2)$ when $(i_1 + i_2) < 1$
$= 1$ when $(i_1 + i_2) >= 1$

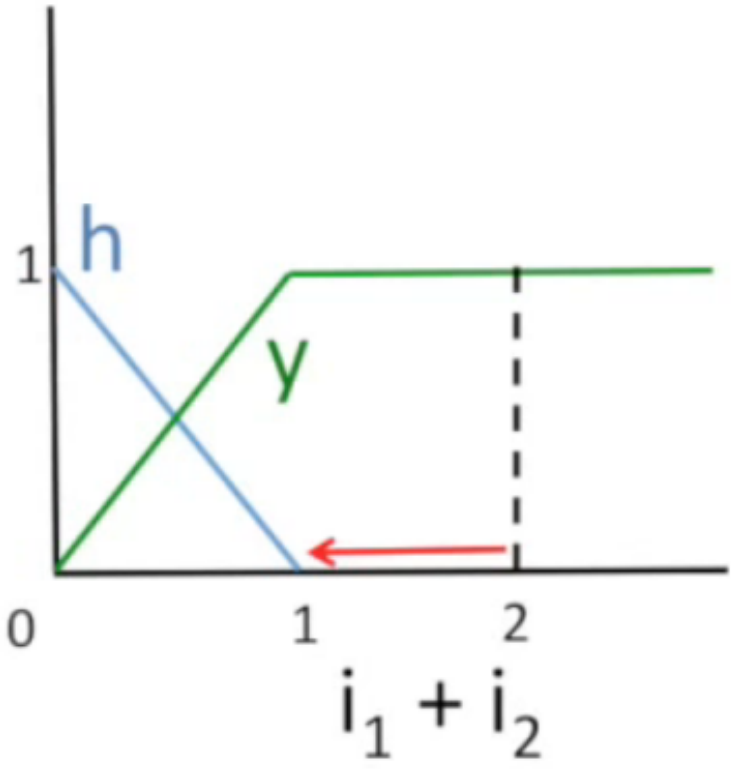


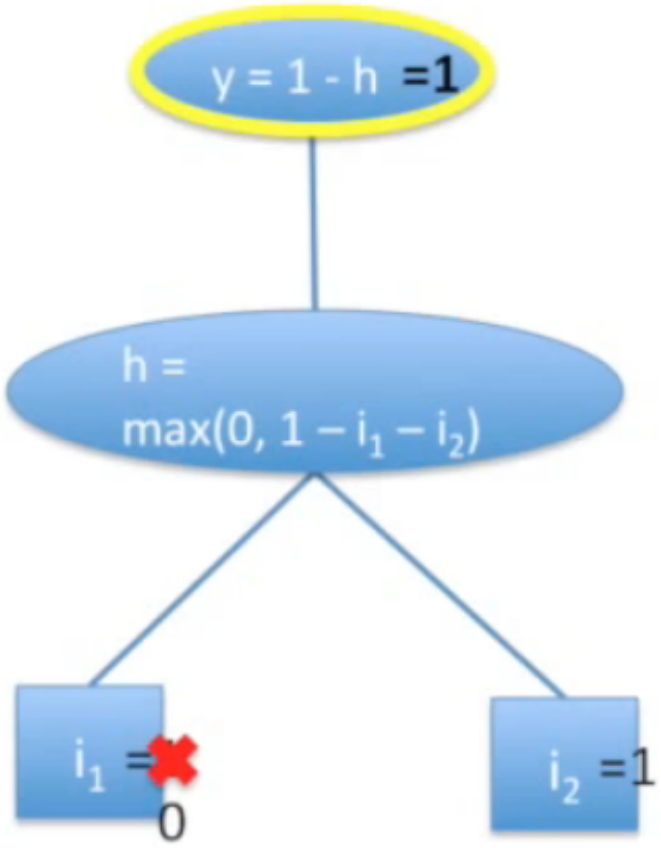


https://docs.google.com/presentation/d/0B15F_QN41VQXSXRFMzgtS01UOU0/edit#slide=id.p15

Model interpretability: saturation problem

$y = (i_1 + i_2)$ when $(i_1 + i_2) < 1$
$= 1$ when $(i_1 + i_2) >= 1$





https://docs.google.com/presentation/d/0B15F_QN41VQXSXRFMzgtS01UOU0/edit#slide=id.p15

Model interpretability: gradient-based methods

For the jth feature of ith datapoint in a dataset, the feature's contribution to the model's pre-activation output is,

$$W_j^{(i)} x_j^{(i)}$$

From our understanding of gradient learning and backpropogation, we can represent the weight above as,

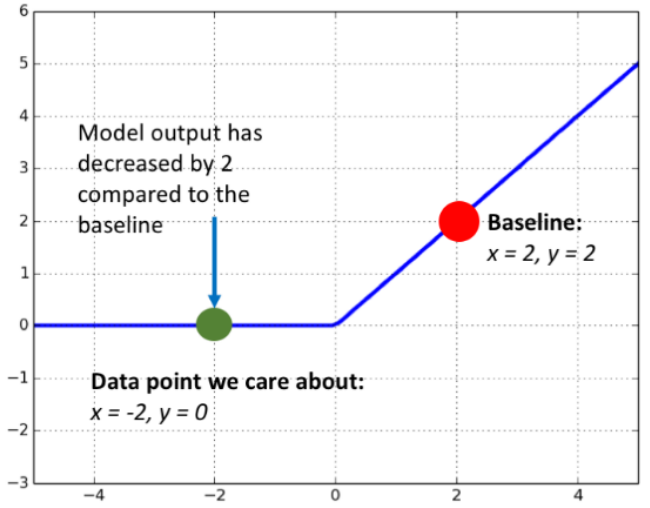
$$W_j^{(i)} = \frac{\partial Z_i}{\partial x_j^{(i)}}$$

In other words, the weight assigned to the j*th* feature tells us the gradient of that feature with respect to the pre-activation output (it is trivial to do this for a post-activation output as well). During backprop, we are computing these gradients, and we can now use various methods to leverage gradients for mapping feature contributions.

There is one problem: feature importance is *relative.* To solve this issue, we need a baseline to compare to. For MNIST, a black background is suitable.

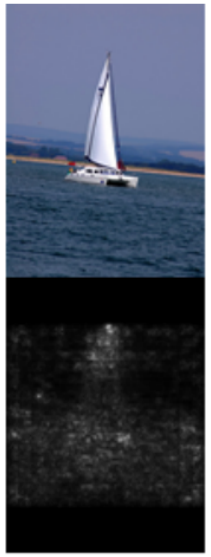
With other inputs, picking a baseline is not so intuitive. And in the context of a neural network, you can still have *saturation*. In a ReLU (shown below), this is clear, since the input and output are changing but the gradient taken at the point we care about (change from baseline) is locally zero (the problem here is that derivatives give change around an infinitesimal point).

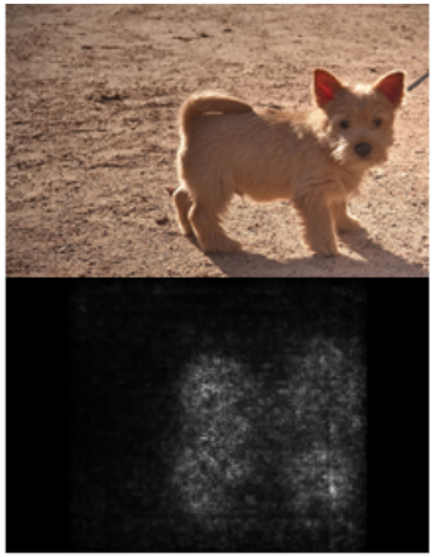
$$y = ReLU(x) = max(0, x)$$



Integrated gradients and DeepLIFT are two approaches to using gradient-based methods and mitigating saturation and numerical problems with computing infinitesimal differences instead of finite differences.

Model interpretability: saliency maps





Visualizing saliency

The procedure is related to back-propagation, but in this case the optimization is performed with respect to the input image, while the weights are fixed to those found during the training stage.

More concretely,

$$S_c(I) \approx W^T I + b$$

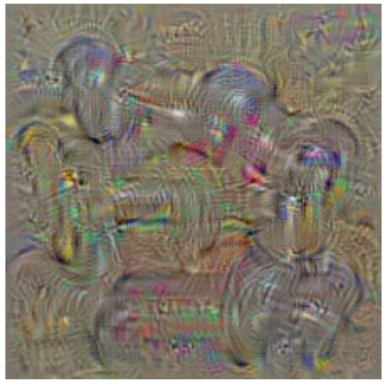
Where,

$$W = \frac{\partial S_c}{\partial I}\bigg|_{I_i}$$

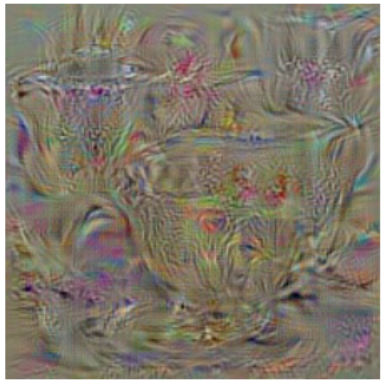
Optimization by gradient ascent

More formally, let Sc(I) be the score of the class c, computed by the classification layer of the ConvNet for an image I. We would like to find (via backprop) an L2-regularised image, such that the score Sc is high,

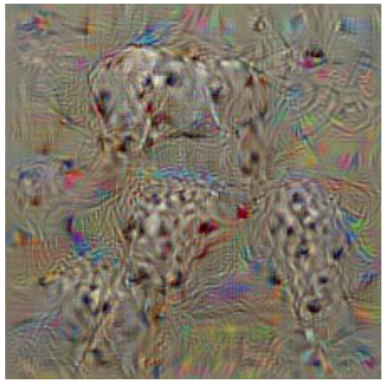
$$\arg\max_I S_c(I) - \lambda \|I\|_2^2$$



dumbbell



cup



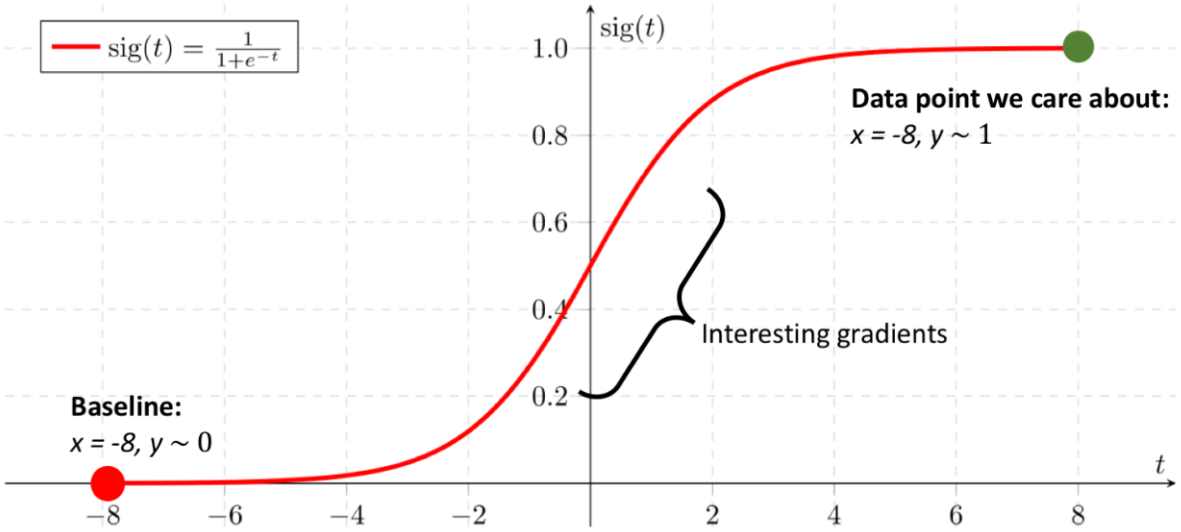
dalmatian

https://arxiv.org/pdf/1312.6034.pdf

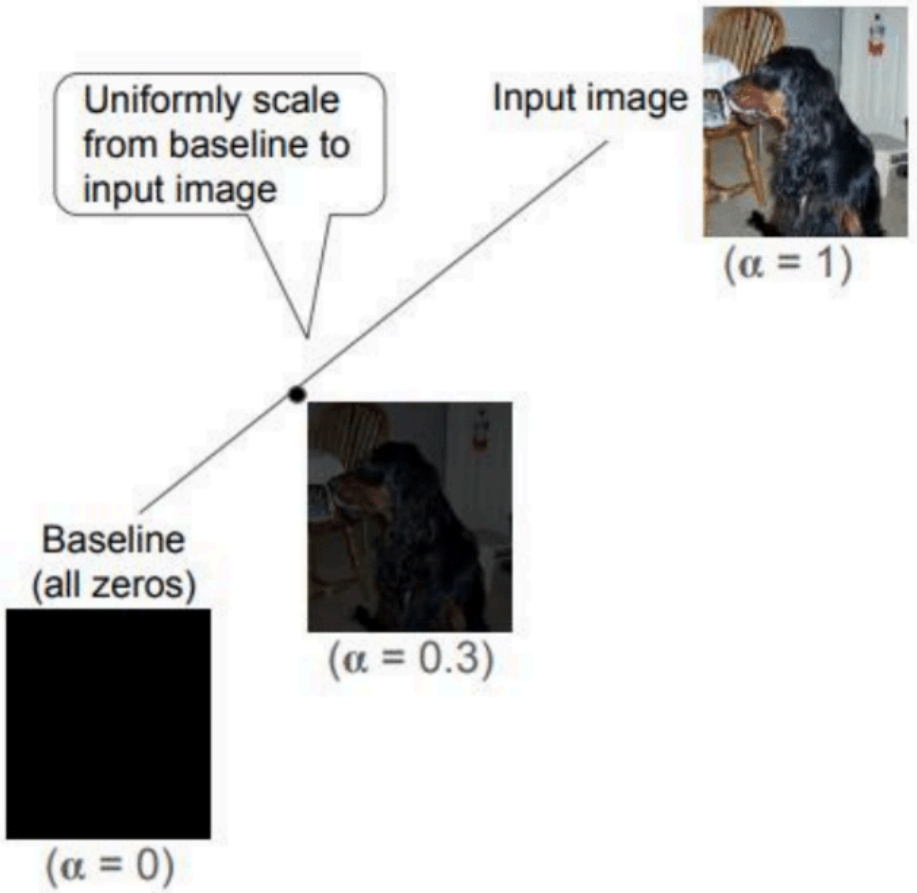
Model interpretability: integrated gradients

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

$$\text{Integrated Grads}_i^{\text{approx}}(x) ::= (x_i - x_i') \times \sum_{k=1}^{m} \frac{\partial F\left(x' + \frac{k}{m} \times (x - x')\right)}{\partial x_i} \times \frac{1}{m}$$



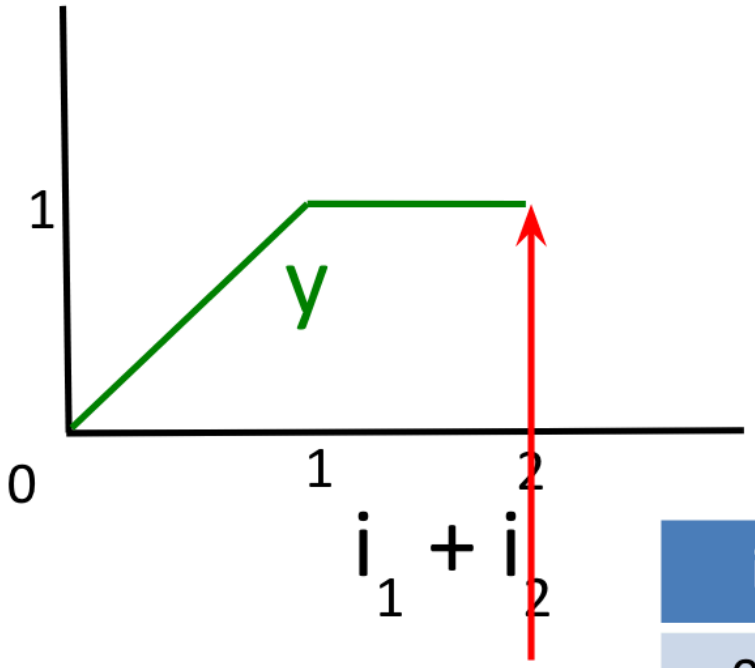
Model interpretability: integrated gradients

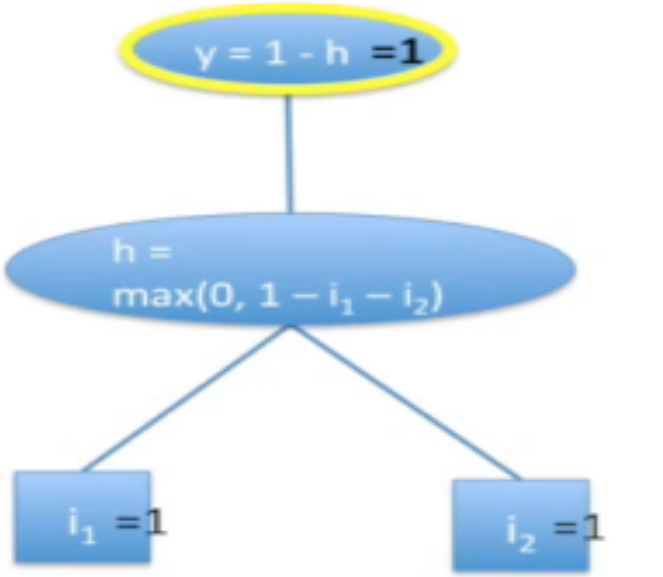


where:

- F is the prediction function for the label
- $image_i$ is the intensity of the i^{th} pixel
- $IG_i(image)$ is the integrated gradient w.r.t. the i^{th} pixel, i.e., attribution for i^{th} pixel

Model interpretability: integrated gradients (simple example)





Average $dy/di_x = 0.5$ $dy/di_x = 0$

(Average dy/di_1)*$\Delta i_1 = 0.5$

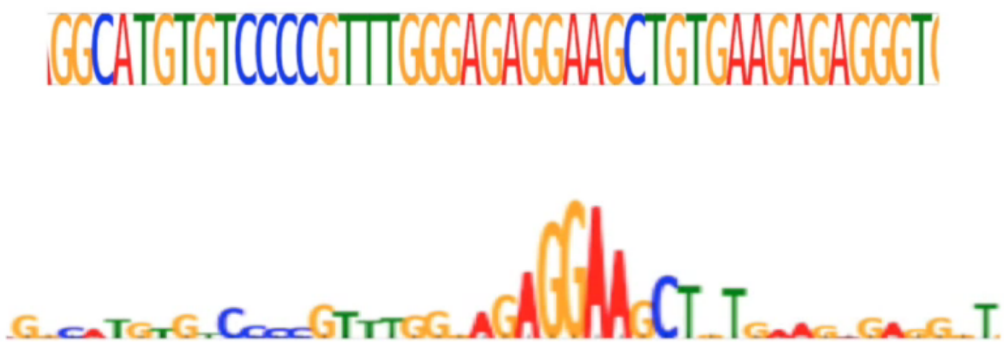
(Average dy/di_1)*$\Delta i_2 = 0.5$

i_1	i_2	dy/di_x
0.0	0.0	1
0.2	0.2	1
0.4	0.4	1

i_1	i_2	dy/di_x
0.6	0.6	0
0.8	0.8	0
1.0	1.0	0

https://docs.google.com/presentation/d/0B15F_QN41VQXSXRFMzgtS01UOU0/edit#slide=id.p15

Model interpretability: deeplift



Explains the difference from a reference value of what the output in terms of difference from a reference value of inputs

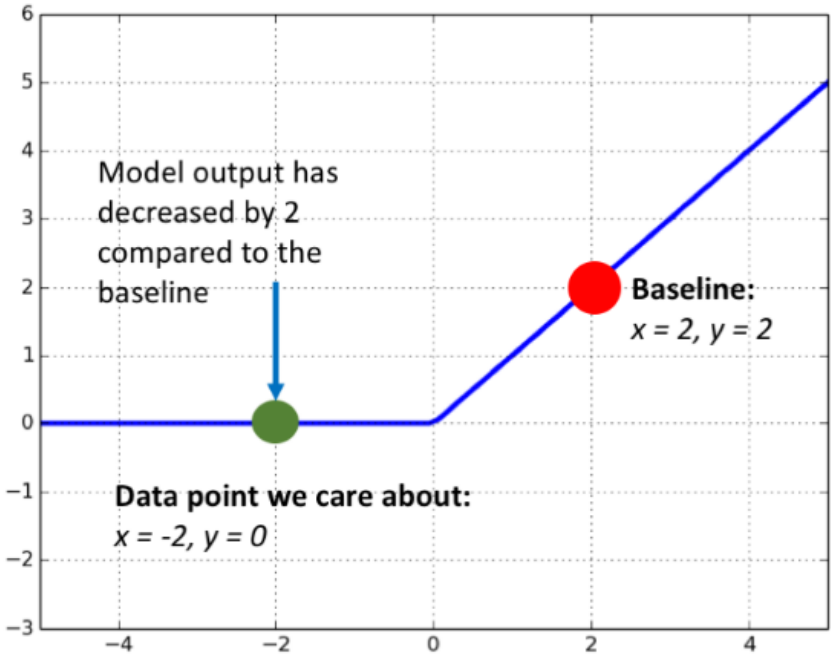
Suppose you have a target output neuron, *h,* with difference from reference Δt and input units $x_1, x_2, \ldots, x_n$ we want a contribution score $C_{\{\Delta x_i \Delta t\}}$ that satisfies the following condition:

$$\sum_{i=1}^{n} C_{\Delta x_i \Delta t} = \Delta t$$

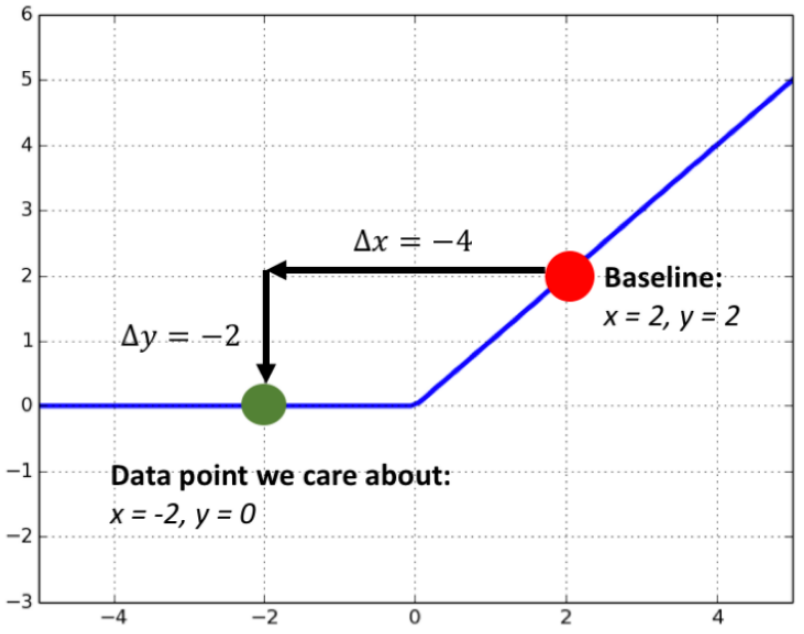
What is a good reference for MNIST? What about genomic sequences?

Model interpretability: deeplift

$$y = ReLU(x) = max(0, x)$$



$$y = ReLU(x) = max(0, x)$$



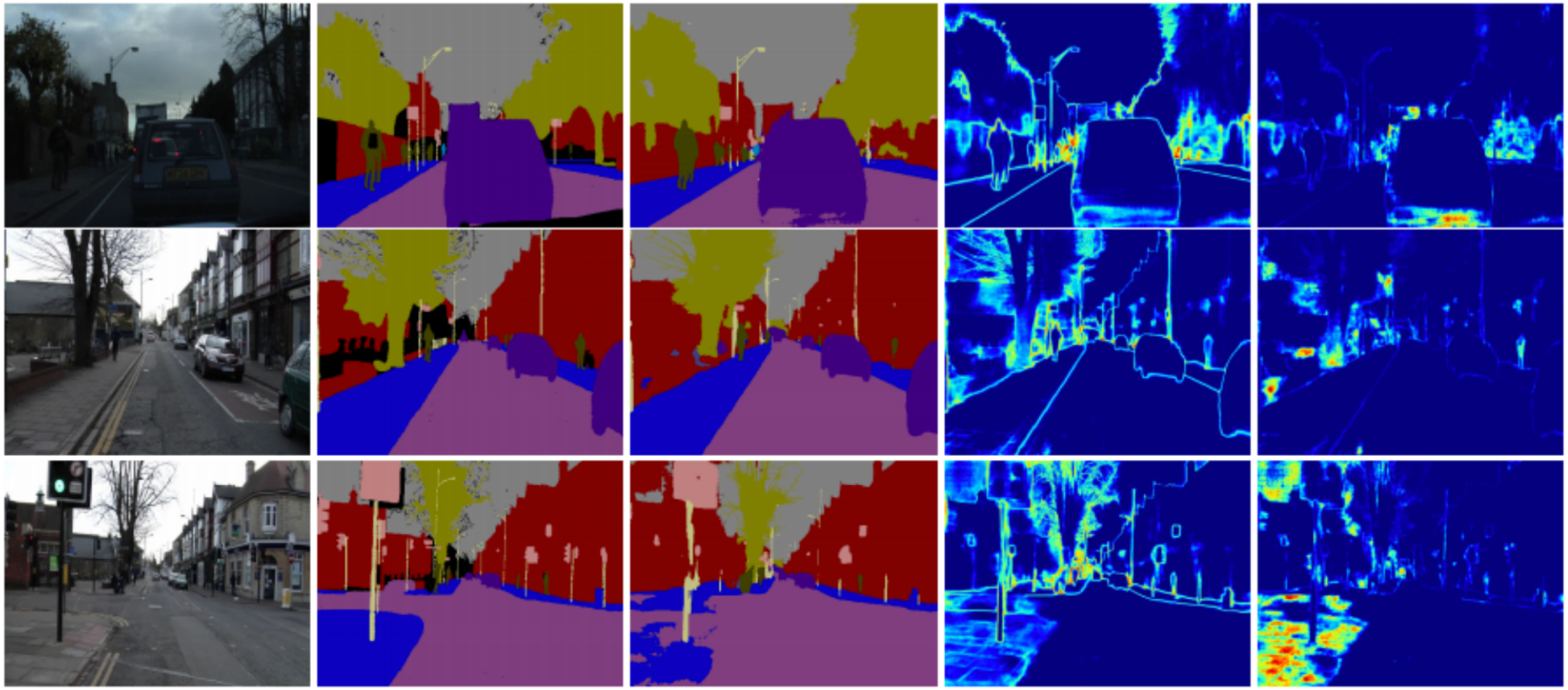
1. Calculating the slope

$$\frac{\Delta y}{\Delta x} = \frac{-2}{-4} = 0.5$$

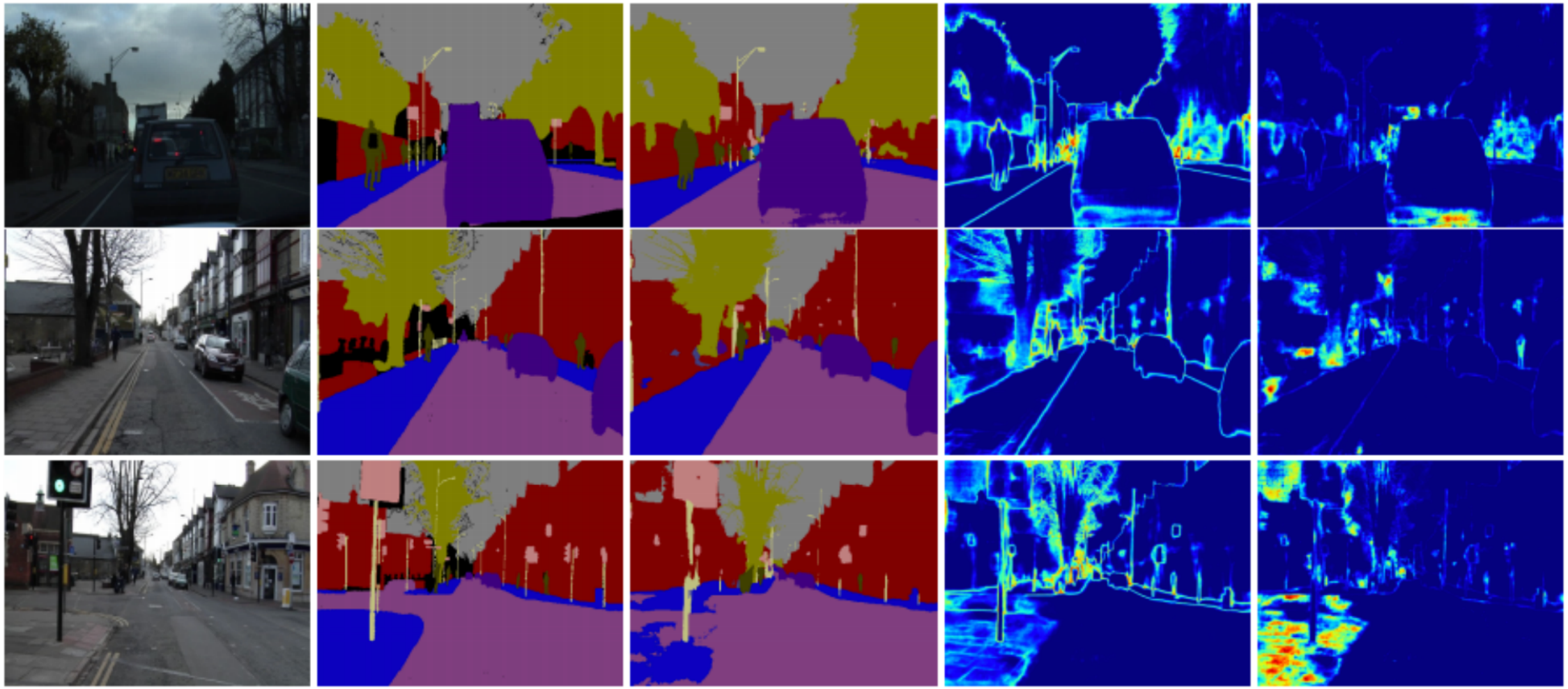
2. Finding the feature importance

$$\Delta x \times \frac{\Delta y}{\Delta x} = -4 \times 0.5 = -2$$

Model uncertainty: terminology



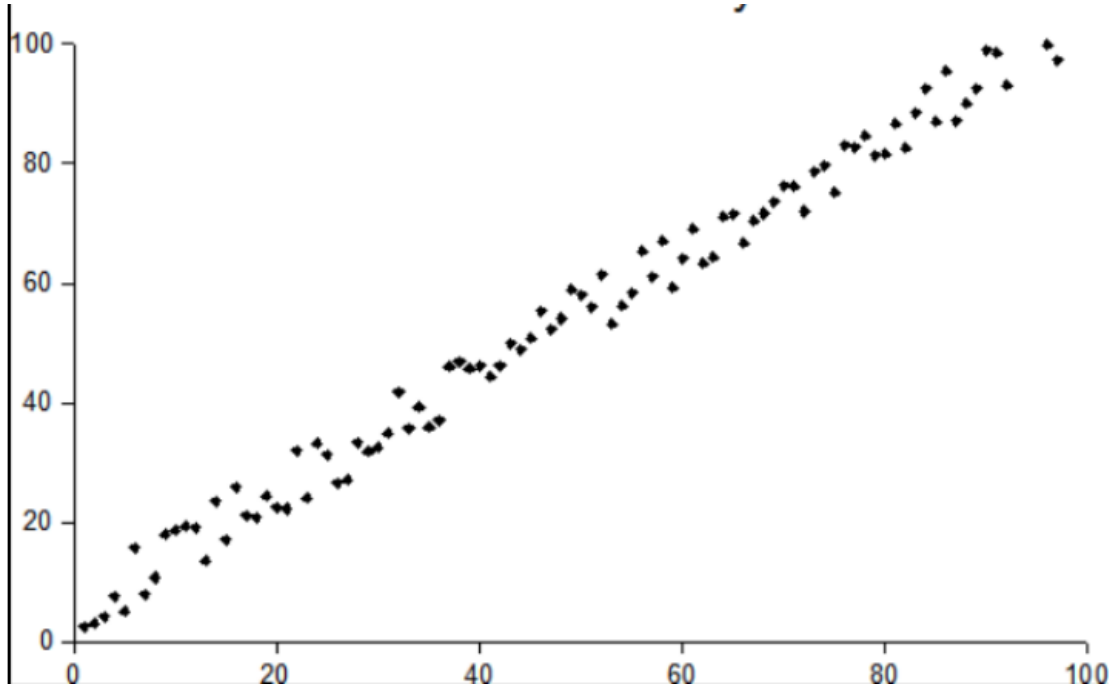
Model uncertainty: terminology

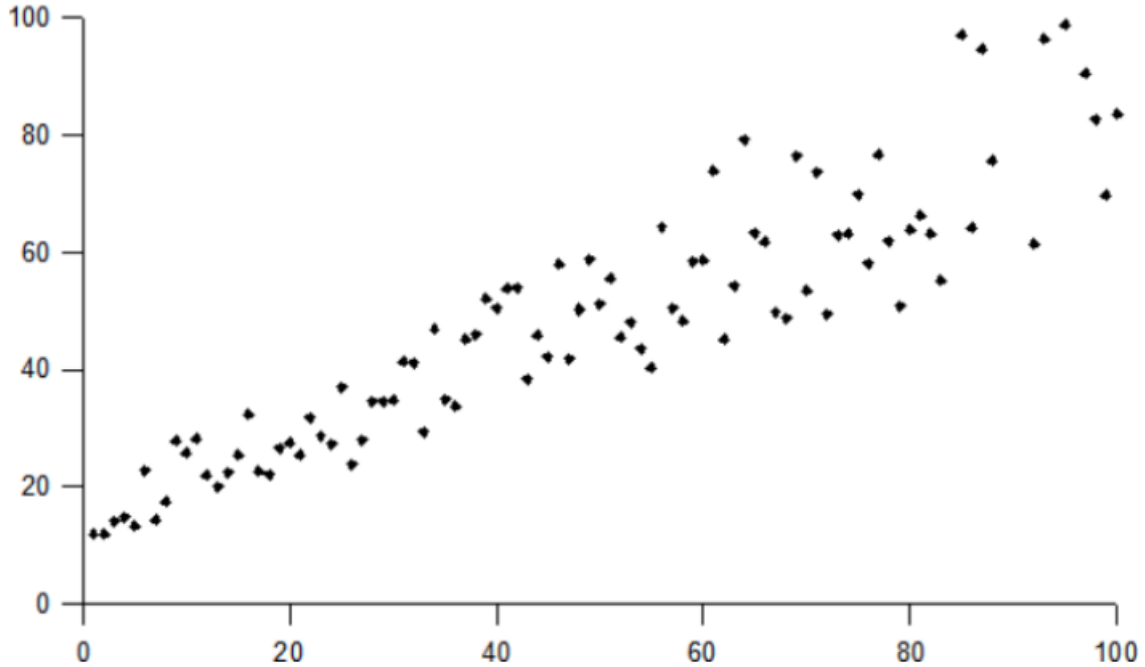


Aleatoric uncertainty

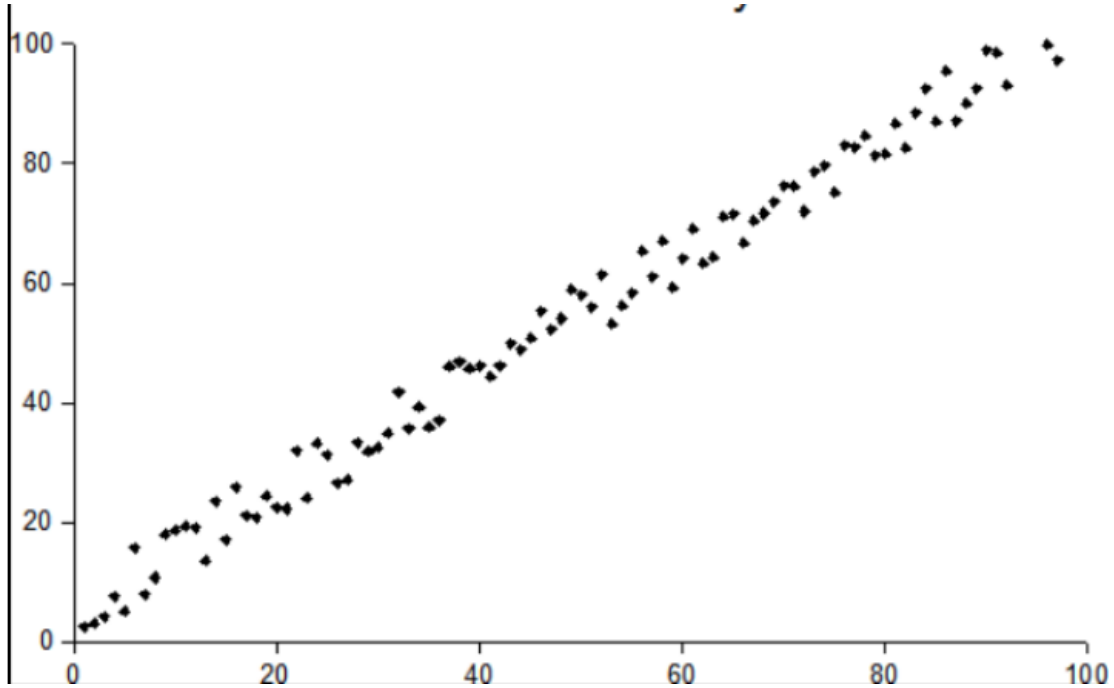
Epistemic uncertainty

Model uncertainty: terminology



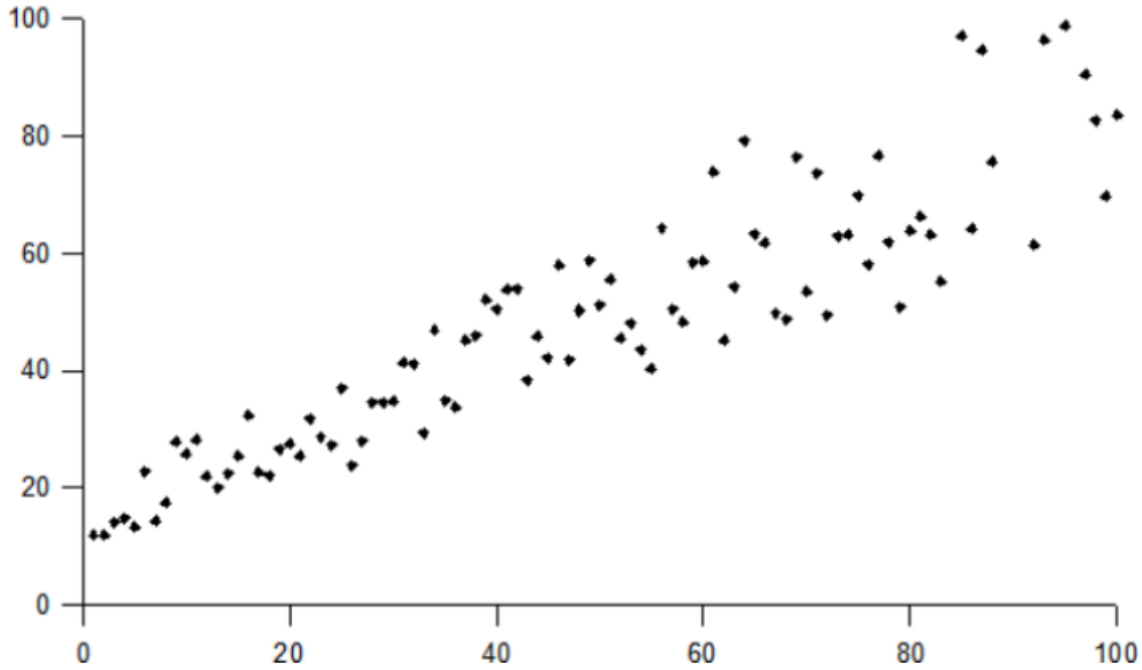


Model uncertainty: terminology



Homoscedastic noise

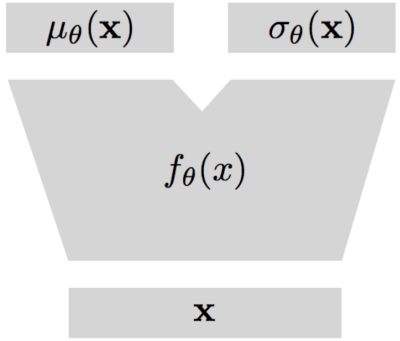
$$\epsilon \sim N(0, 1)$$



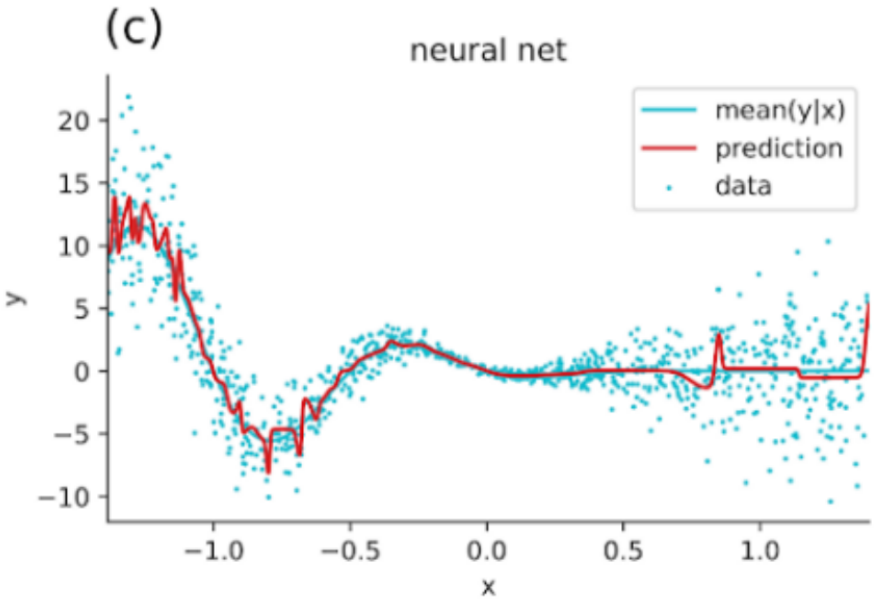
Heteroscedastic noise

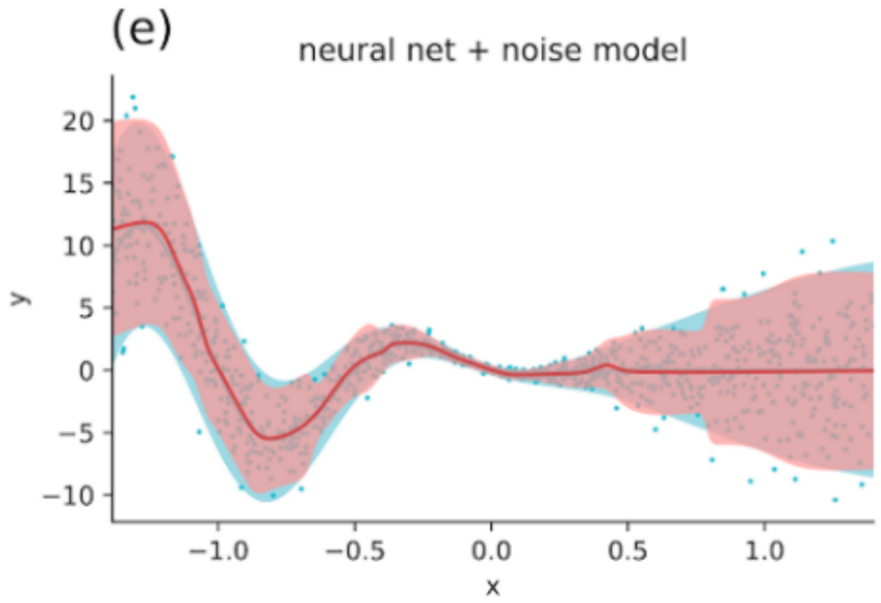
$$\epsilon \sim N(0, g(x))$$

Model uncertainty: modeling aleatoric uncertainty

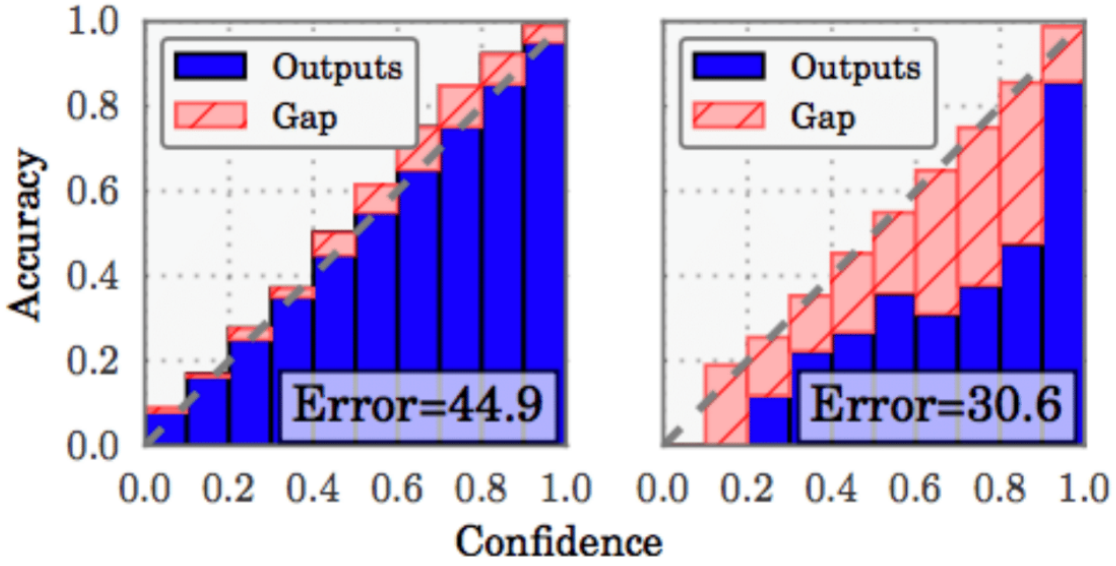


$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N} \log \mathcal{N}(y_i; \mu_\theta(x_i), \sigma_\theta^2(x_i))$$

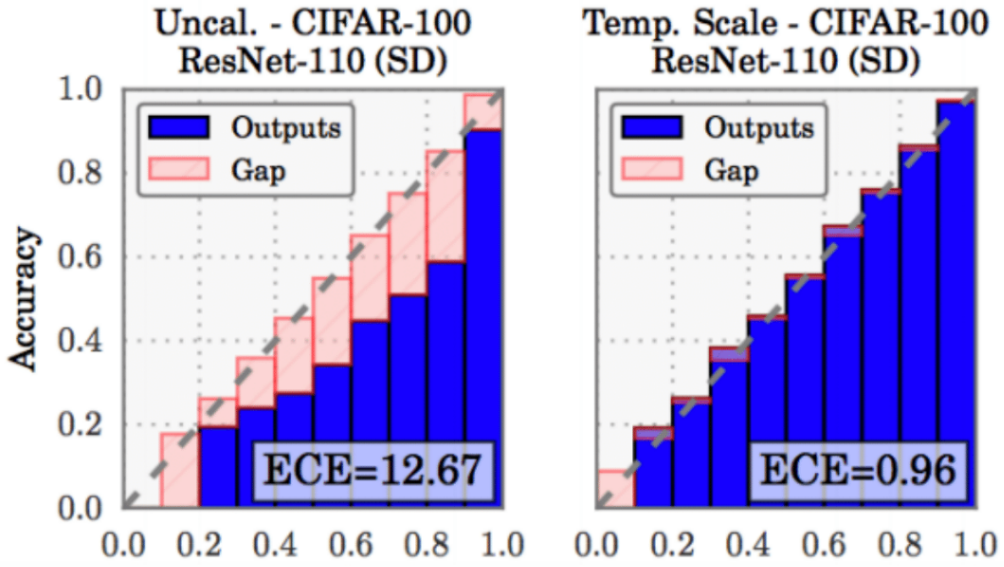




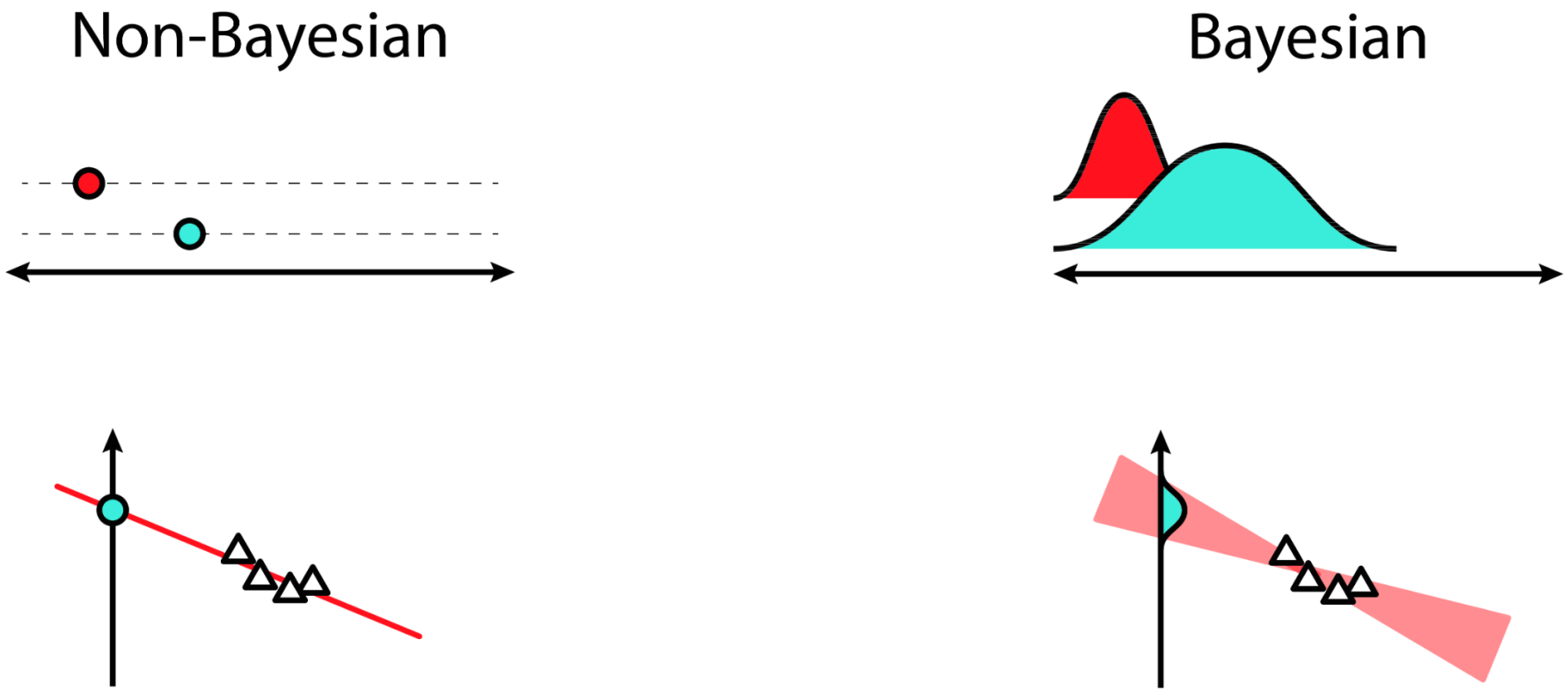
Model uncertainty: model calibration



$$P(\hat{\mathbf{y}}) = \frac{e^{\mathbf{z}/T}}{\sum_j e^{z_j/T}}$$

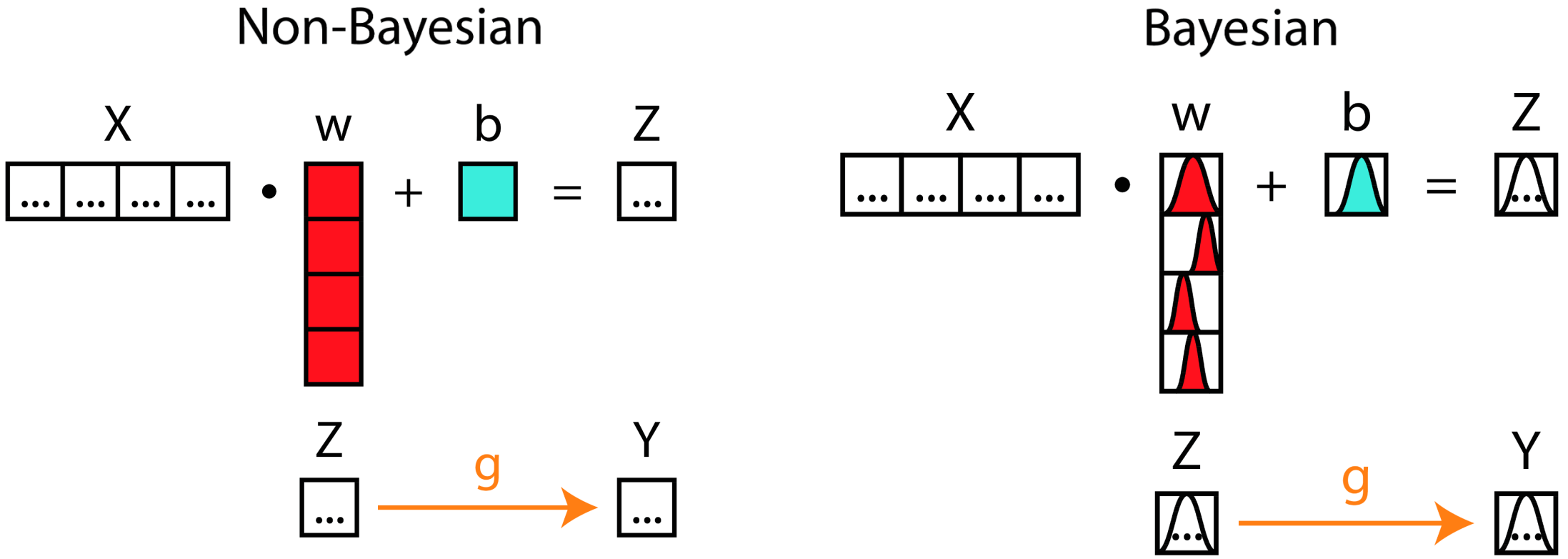


Model uncertainty: bayesian neural networks



https://ericmjl.github.io/bayesian-deep-learning-demystified/#/intuition

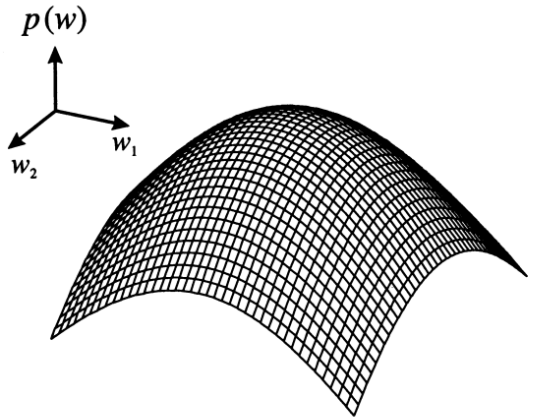
Model uncertainty: bayesian neural networks

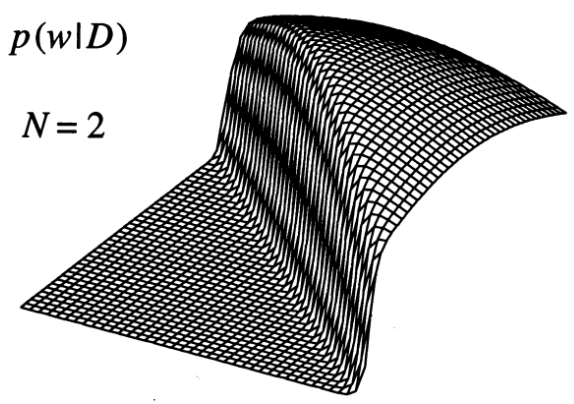


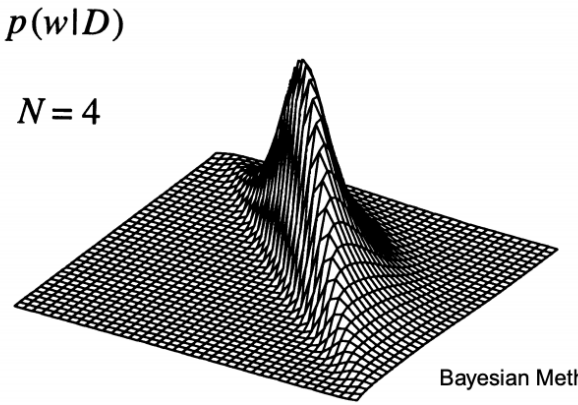
https://ericmjl.github.io/bayesian-deep-learning-demystified/#/intuition

Model uncertainty: bayesian neural networks

Let's take a 3-dimensional training distribution that is roughly bimodal that we want to estimate. Using BNNs, you estimate a probability distribution over epochs (N) of training that starts to estimate the distribution from a Gaussian prior. Perhaps with more training or model complexity, we can estimate the bimodality.



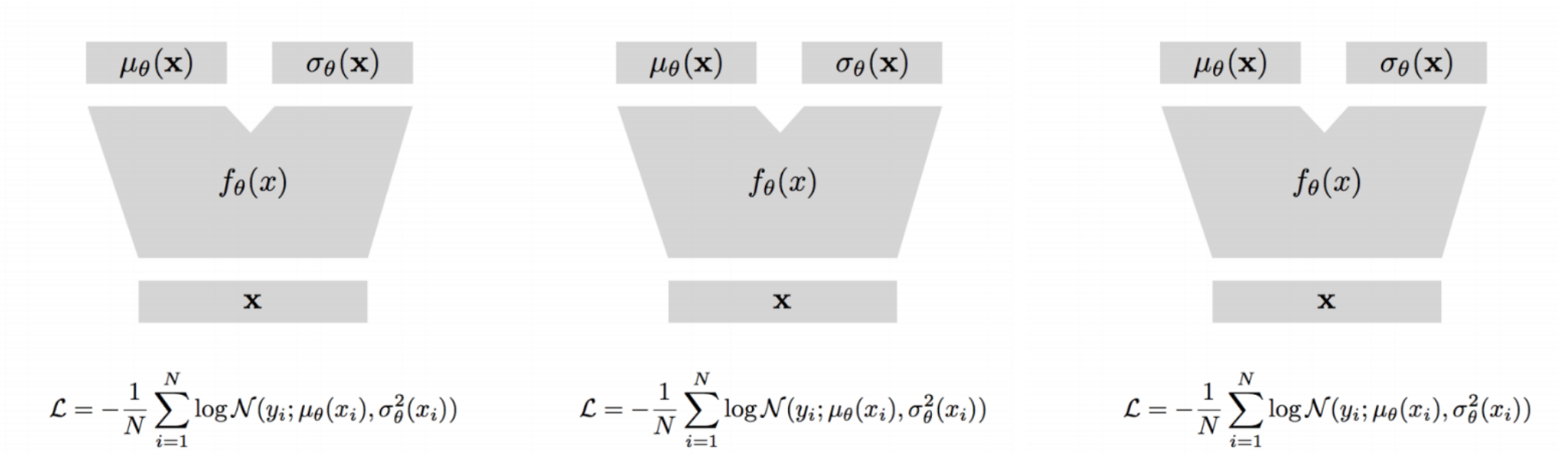




Bayesian Methods for Neural Networks – p.14/29

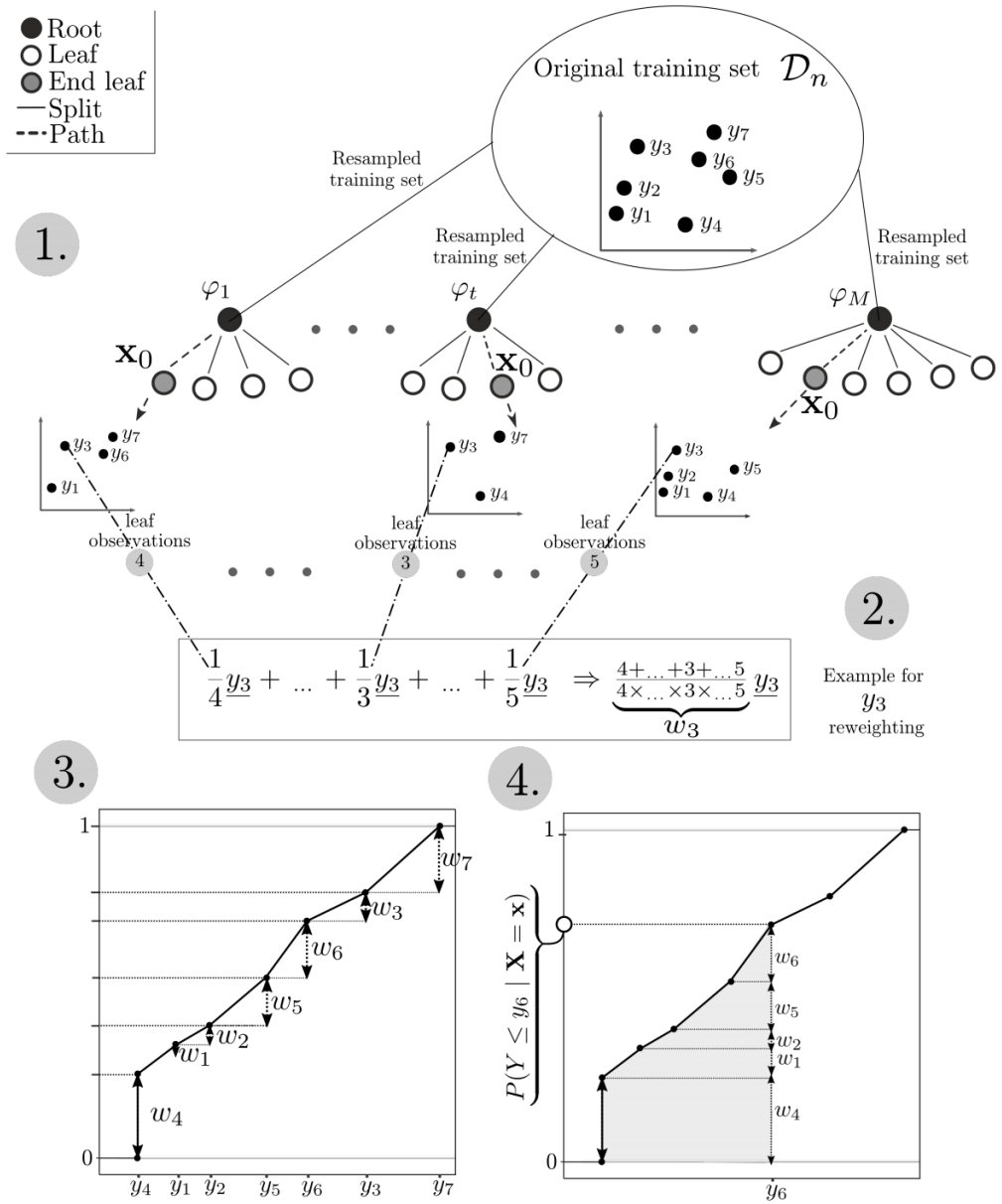
https://ericmjl.github.io/bayesian-deep-learning-demystified/#/intuition

Model uncertainty: ensembling

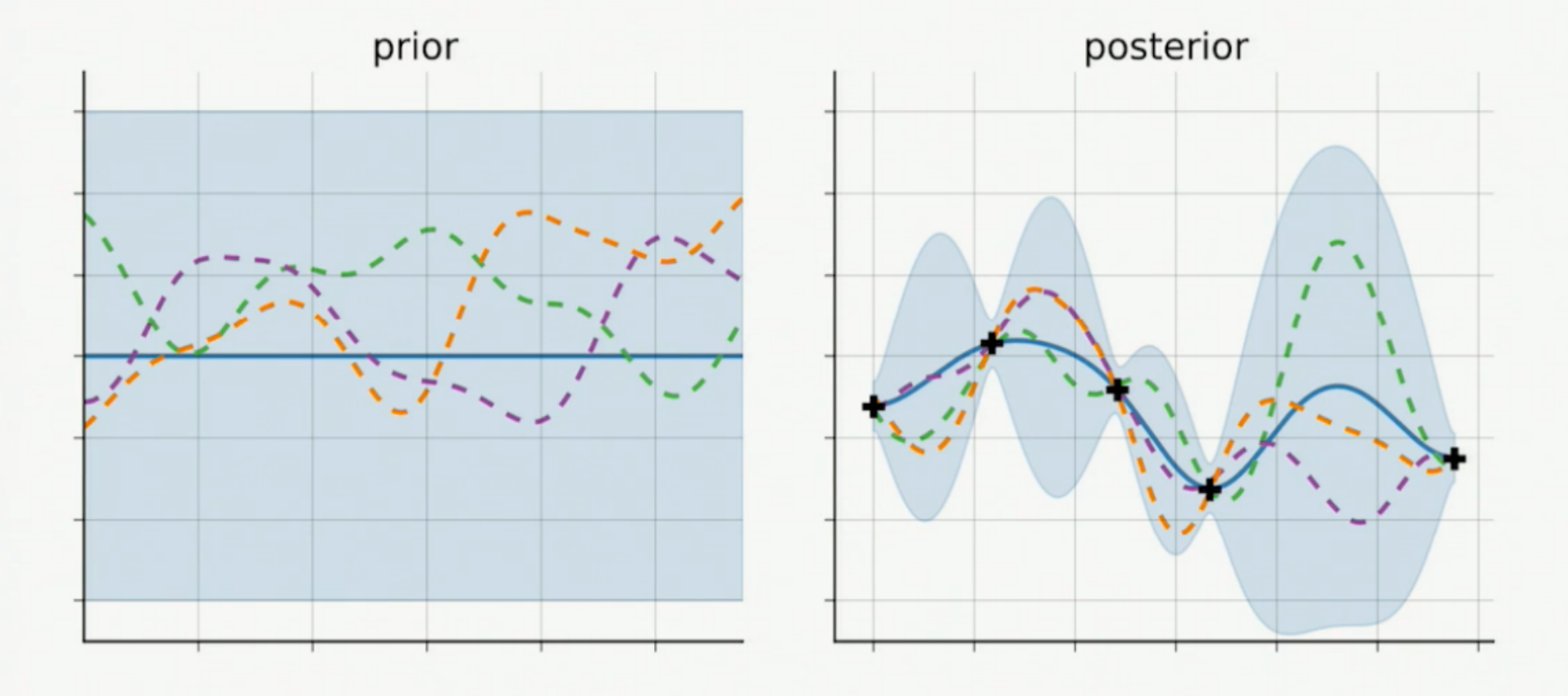


Epistemic uncertainty - $Var(\mu_{\theta_i}((x)))$

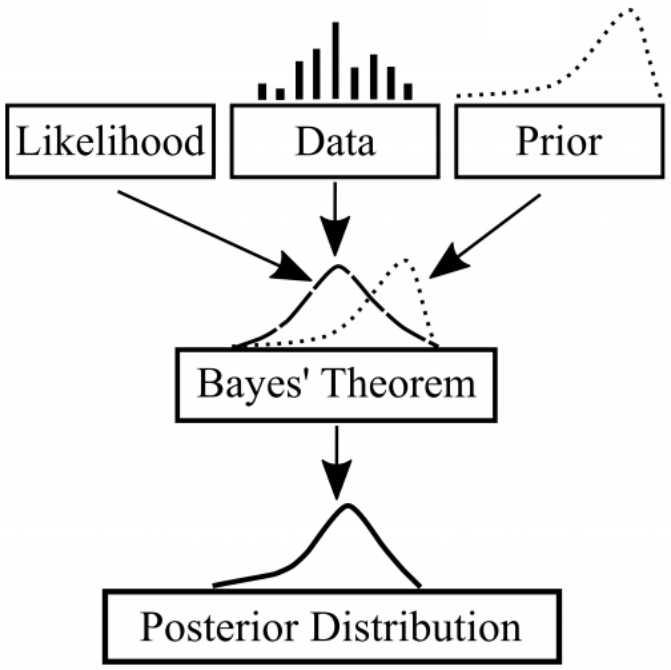
Model uncertainty: ensembling

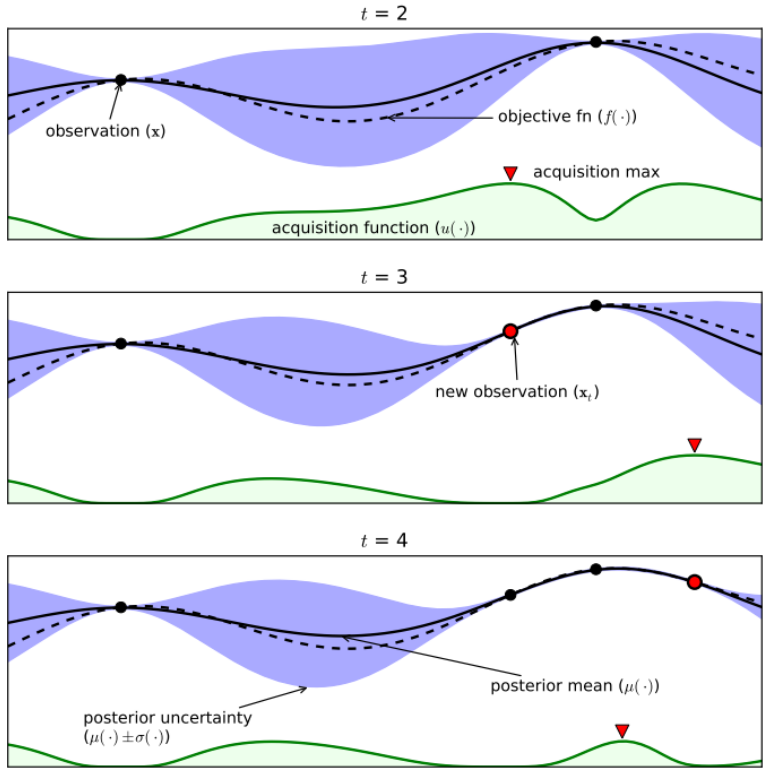


Experimental design: gaussian processes



Experimental design: bayesian optimization overview





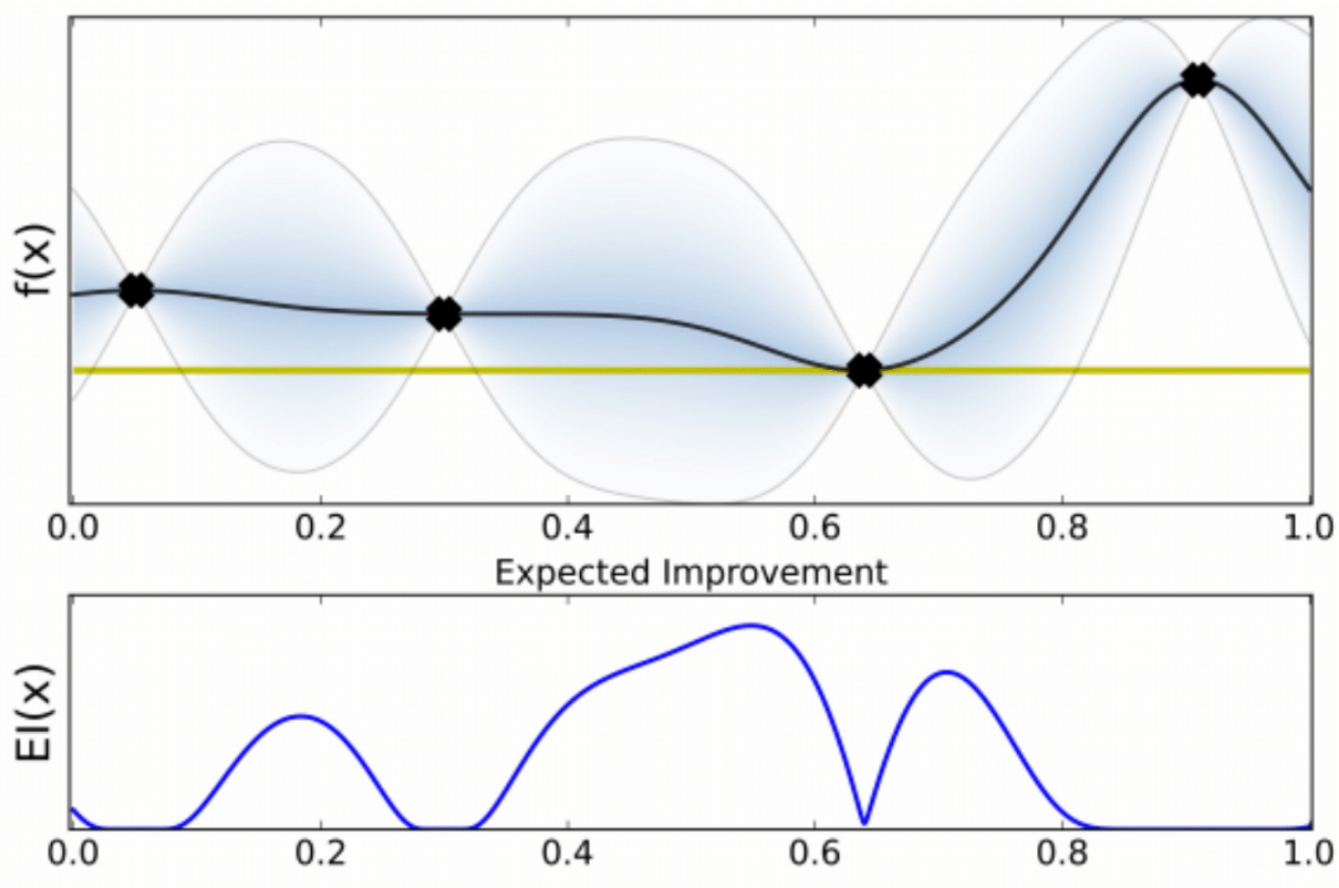
Black-box optimization in a nutshell:

1. initial sample
2. initialize our model
3. get the acquisition function $\alpha(\mathbf{x})$
4. optimize it! $\mathbf{x}_{\text{next}} = \arg\max \alpha(\mathbf{x})$
5. sample new data; update model
6. repeat!
7. **make recommendation**

https://arxiv.org/pdf/1012.2599.pdf

Experimental design: EI

$$\alpha_{EI}(\mathbf{x};\theta,\mathcal{D}) = \int_y \max(0, y_{best} - y)p(y|\mathbf{x};\theta,\mathcal{D})dy$$



Experimental design: exploration vs. exploitation (in words)

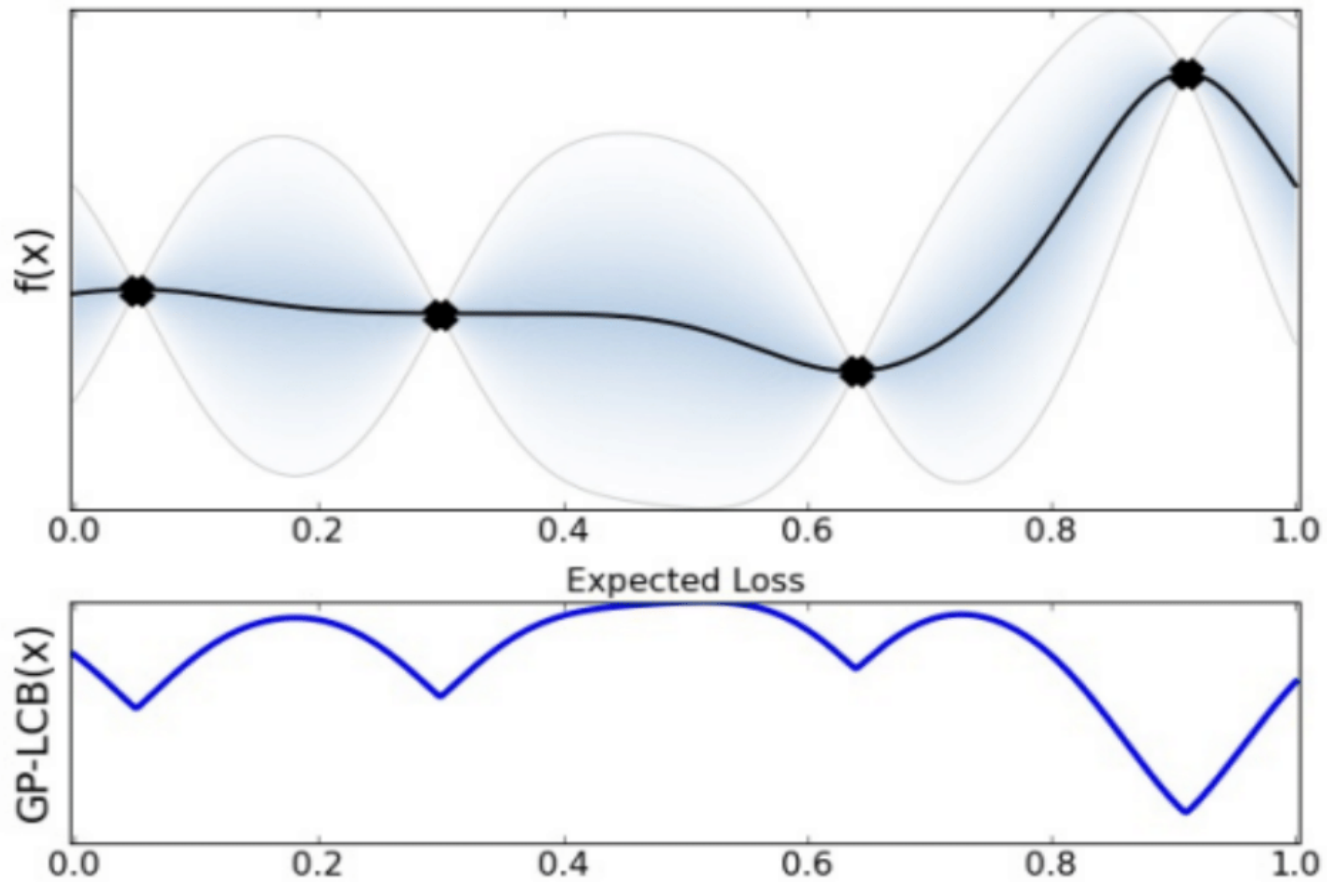
Exploration is preferentially evaluating points with high uncertainty to traverse more of the space of training points.

Exploitation is preferentially evaluating points by “trusting” the model output itself, which is placed closer to the previous point evaluated.

https://arxiv.org/pdf/1012.2599.pdf

Experimental design: LCB

$$\alpha_{LCB}(\mathbf{x};\theta,\mathcal{D}) = -\mu(\mathbf{x};\theta,\mathcal{D}) + \beta_t \sigma(\mathbf{x};\theta,\mathcal{D})$$

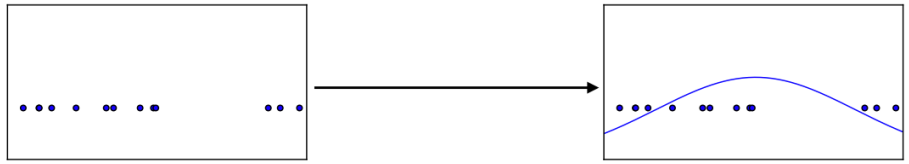


Generative models: estimating probability distributions

What does it mean to ”learn” a probability distribution? Classically, this means to learn a probability density. This is often done by defining a parametric family of densities $(P_\theta)_{\theta \in \mathbb{R}^d}$ and finding the one that maximizes the likelihood of observing our training data:

$$\max_{\theta \in \mathbb{R}^d} \frac{1}{m} \sum_{i=1}^{m} \log P_\theta\left(x^{(i)}\right)$$

- Density estimation



This asymptotically amounts to minimizing the Kullback-Leibler divergence, but there are many forms that this can take resulting in different “distance” metrics (i.e the divergence between two probability distributions). Different distance metrics have different properties, namely the rate of convergence on sequences of probability distributions. A sequence of distributions $(\mathbb{P}_t)_{t \in \mathbb{N}}$ converges if there is a distribution $\mathbb{P}_\infty$ and a distance metric ρ such that $\rho(\mathbb{P}_t, \mathbb{P}_\infty)$tends to zero. This is highly subject to how ρ is defined. Different distance metrics are discussed on the next slide.

https://mit6874.github.io/assets/misc/l13/wgan.pdf

Generative models: distance metrics

First we want general definitions of distances and divergences between two distributions $\mathbb{P}_r, \mathbb{P}_g \in \mathrm{Prob}(\mathcal{X})$

Total variation distance

$$\delta(\mathbb{P}_r, \mathbb{P}_g) = \sup_{A \in \Sigma} |\mathbb{P}_r(A) - \mathbb{P}_g(A)|$$

KL-divergence

$$KL(\mathbb{P}_r \| \mathbb{P}_g) = \int \log\left(\frac{P_r(x)}{P_g(x)}\right) P_r(x) d\mu(x)$$

Jensen-Shannon divergence

$$JS(\mathbb{P}_r, \mathbb{P}_g) = KL(\mathbb{P}_r \| \mathbb{P}_m) + KL(\mathbb{P}_g \| \mathbb{P}_m)$$

Earth-mover (Wasserstein) distance

$$W(\mathbb{P}_r, \mathbb{P}_g) = \inf_{\gamma \in \mathbf{\Pi}(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(x,y)\sim\gamma}[\|x - y\|]$$

https://mit6874.github.io/assets/misc/l13/wgan.pdf

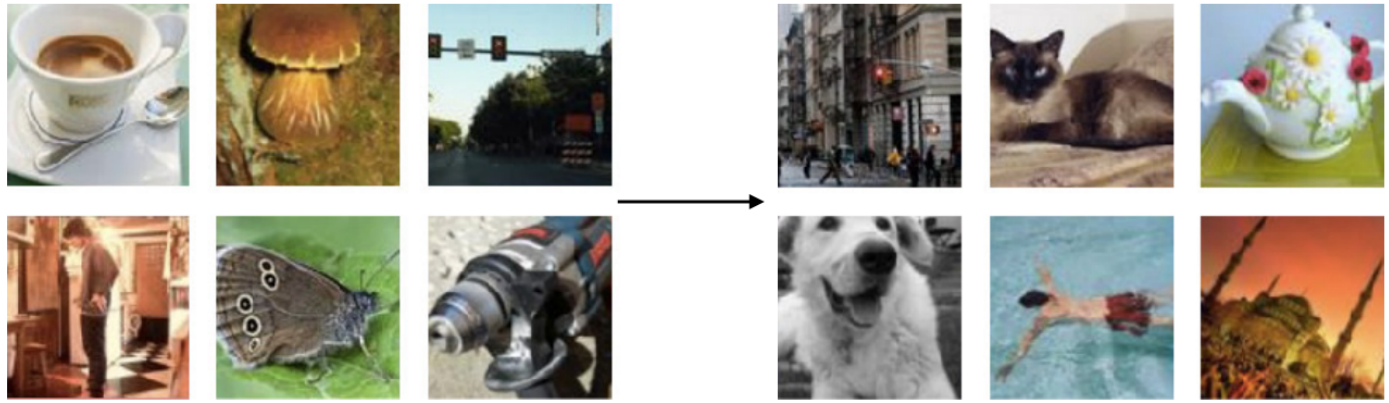
Generative models: adversarial training

Philosophy

"Training a model in a worst-case scenario, with inputs chosen by an adversary" – Ian Goodfellow, 2016

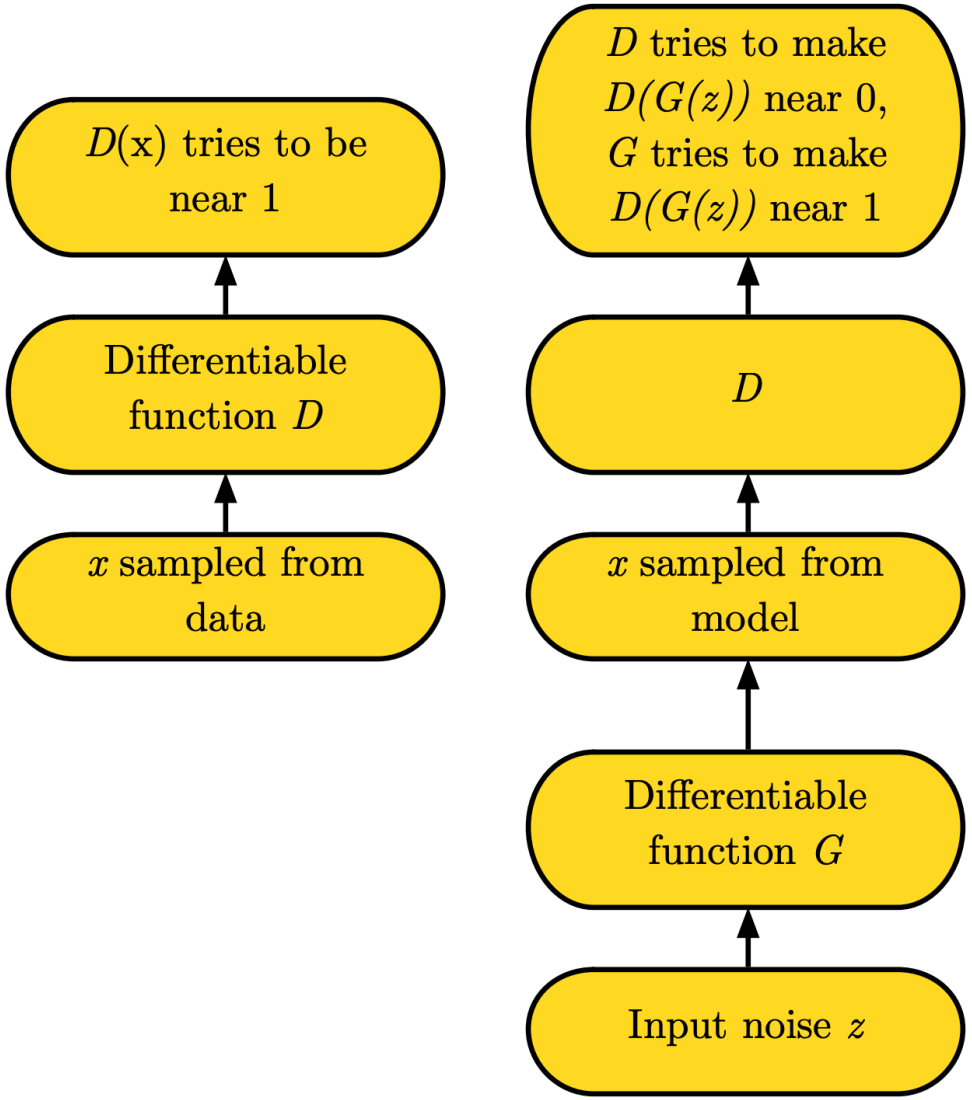
Tasks

- Sample generation



http://www.iangoodfellow.com/slides/2016-12-9-gans.pdf

Generative models: generative adversarial networks



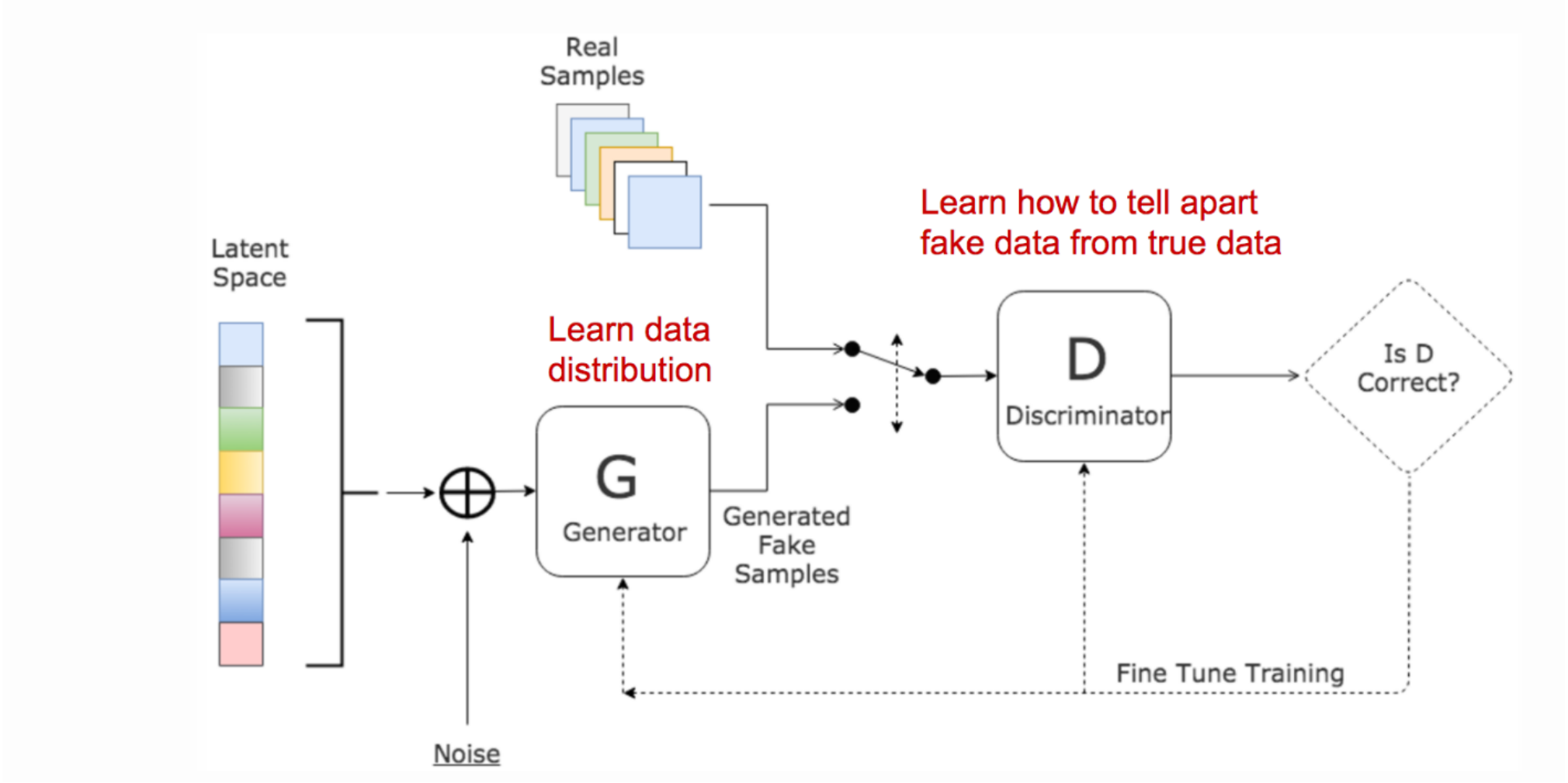
Modified minimax objective function

$$J^{(D)} = -\frac{1}{2}\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}} \log D(\boldsymbol{x}) - \frac{1}{2}\mathbb{E}_{\boldsymbol{z}} \log(1 - D(G(\boldsymbol{z})))$$

$$, J^{(G)} = -\frac{1}{2}\mathbb{E}_{\boldsymbol{z}} \log D(G(\boldsymbol{z}))$$

http://www.iangoodfellow.com/slides/2016-12-9-gans.pdf

Generative models: generative adversarial networks



https://lilianweng.github.io/lil-log/2017/08/20/from-GAN-to-WGAN.html

Generative models: generative adversarial networks

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

  for k steps **do**

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
  - Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

  end for

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Update the generator by descending its stochastic gradient:

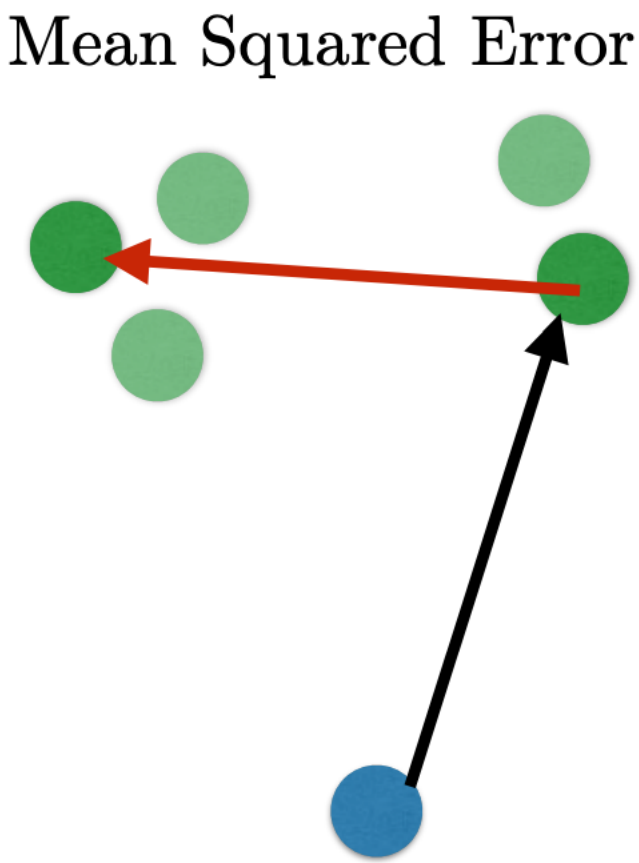
$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

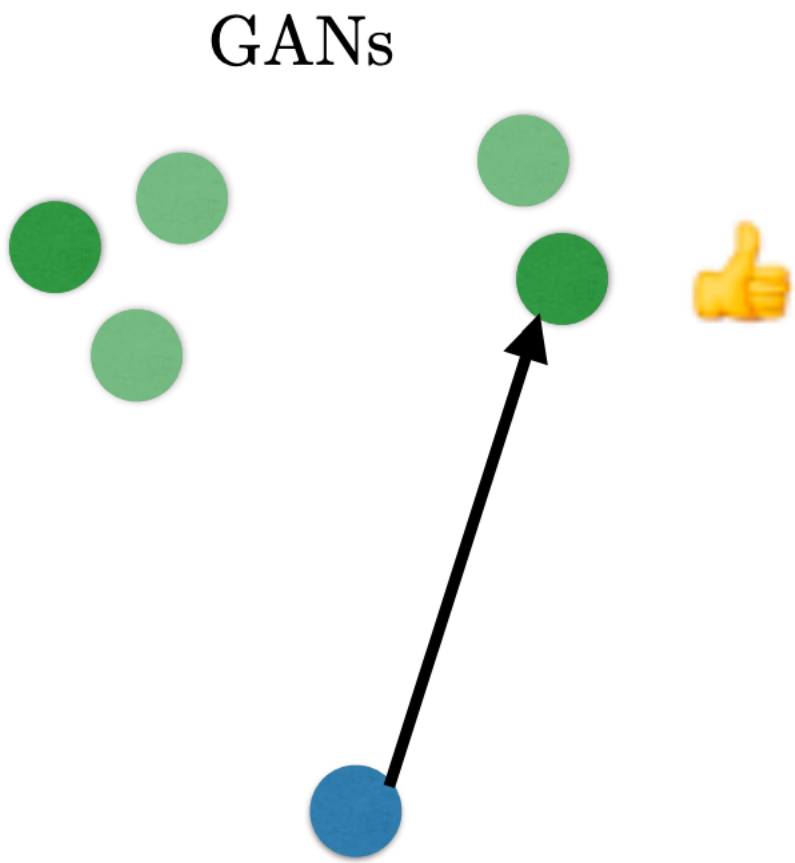
end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

https://lilianweng.github.io/lil-log/2017/08/20/from-GAN-to-WGAN.html

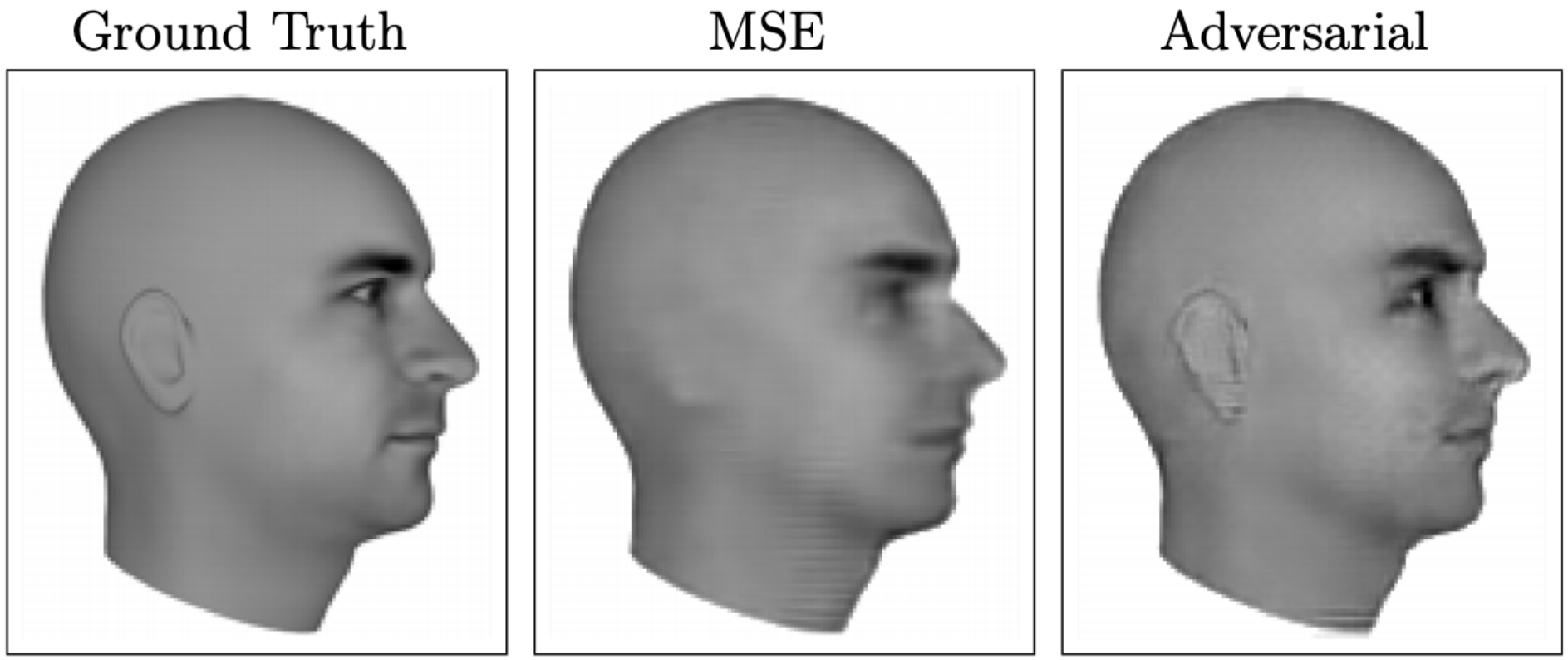
Generative models: generative adversarial networks





http://www.iangoodfellow.com/slides/2016-12-9-gans.pdf

Generative models: generative adversarial networks

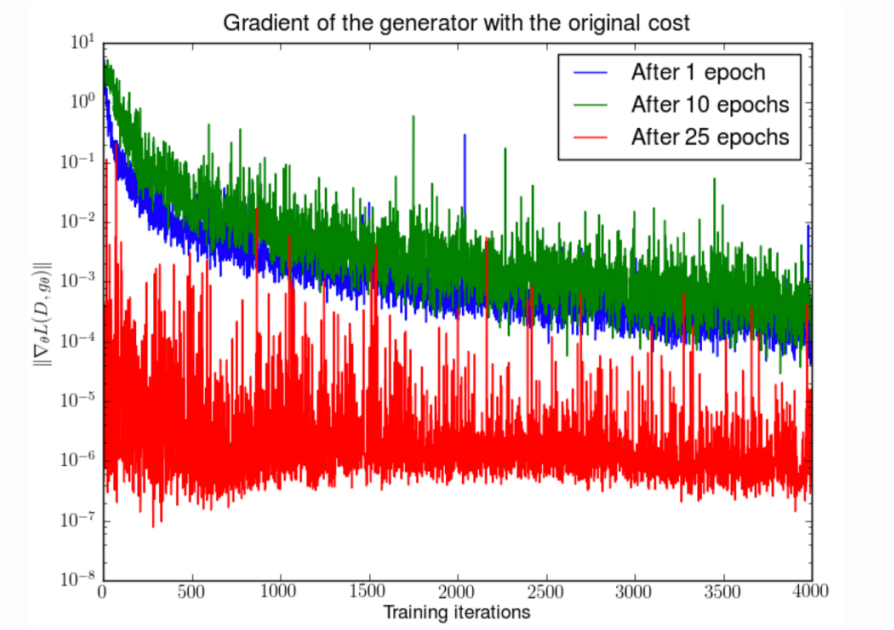


http://www.iangoodfellow.com/slides/2016-12-9-gans.pdf

Generative models: problems

Vanishing gradients

If the discriminator does a great job (read, “the task is too easy”), the gradient of the loss function drops down to close to zero and the learning becomes super slow or even jammed.



Mode collapse

Each iteration of generator over-optimizes for a particular discriminator, producing the same output distribution over and over (mode of the superset of possible distributions) and the discriminator never manages to learn its way out of the trap.

http://www.iangoodfellow.com/slides/2016-12-9-gans.pdf

Generative models: solutions

(1) **Feature Matching**

Feature matching suggests to optimize the discriminator to inspect whether the generator's output matches expected statistics of the real samples. In such a scenario, the new loss function is defined as $\|\mathbb{E}_{x\sim p_r} f(x) - \mathbb{E}_{z\sim p_z(z)} f(G(z))\|_2^2$, where $f(x)$ can be any computation of statistics of features, such as mean or median.

(2) **Minibatch Discrimination**

With minibatch discrimination, the discriminator is able to digest the relationship between training data points in one batch, instead of processing each point independently.

In one minibatch, we approximate the closeness between every pair of samples, $c(x_i, x_j)$, and get the overall summary of one data point by summing up how close it is to other samples in the same batch, $o(x_i) = \sum_j c(x_i, x_j)$. Then $o(x_i)$ is explicitly added to the input of the model.

(3) **Historical Averaging**

For both models, add $\|\Theta - \frac{1}{t}\sum_{i=1}^{t}\Theta_i\|^2$ into the loss function, where Θ is the model parameter and Θ_i is how the parameter is configured at the past training time i. This addition piece penalizes the training speed when Θ is changing too dramatically in time.

(4) **One-sided Label Smoothing**

When feeding the discriminator, instead of providing 1 and 0 labels, use soften values such as 0.9 and 0.1. It is shown to reduce the networks' vulnerability.

(5) **Virtual Batch Normalization** (VBN)

Each data sample is normalized based on a fixed batch (*"reference batch"*) of data rather than within its minibatch. The reference batch is chosen once at the beginning and stays the same through the training.

(6) **Adding Noises**.

Based on the discussion in the previous section, we now know p_r and p_g are disjoint in a high dimensional space and it causes the problem of vanishing gradient. To artificially "spread out" the distribution and to create higher chances for two probability distributions to have overlaps, one solution is to add continuous noises onto the inputs of the discriminator D.

(7) **Use Better Metric of Distribution Similarity**

The loss function of the vanilla GAN measures the JS divergence between the distributions of p_r and p_g. This metric fails to provide a meaningful value when two distributions are disjoint.

Wasserstein metric is proposed to replace JS divergence because it has a much smoother value space. See more in the next section.

Earth-mover (Wasserstein) distance

$$W(\mathbb{P}_r, \mathbb{P}_g) = \inf_{\gamma\in\Pi(\mathbb{P}_r,\mathbb{P}_g)} \mathbb{E}_{(x,y)\sim\gamma}[\|x - y\|]$$

Generative models: wasserstein GANs

Even when two distributions are located in lower dimensional manifolds without overlaps, Wasserstein distance can still provide a meaningful and smooth representation of the distance in-between.