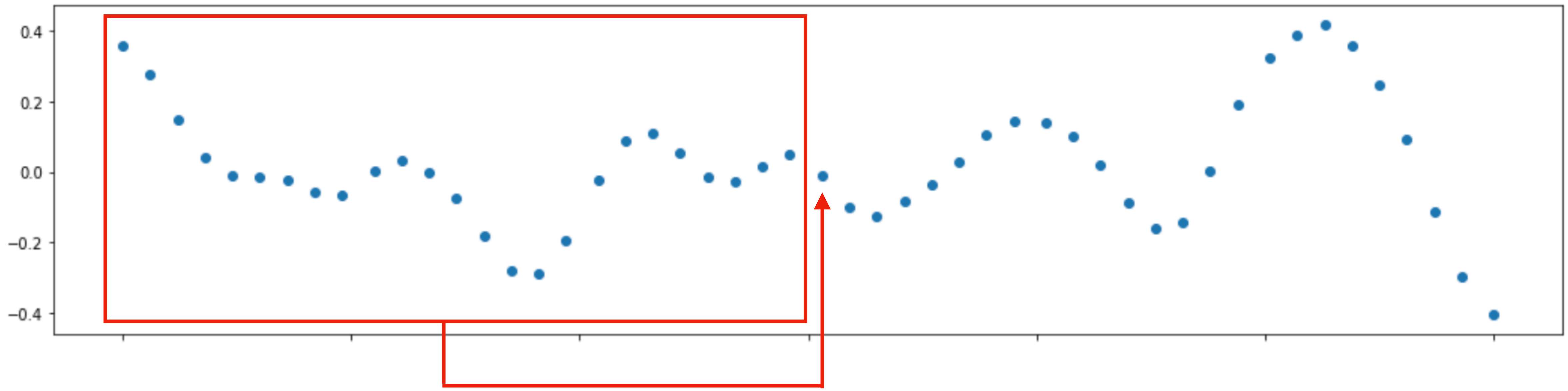
Autoregressive Models

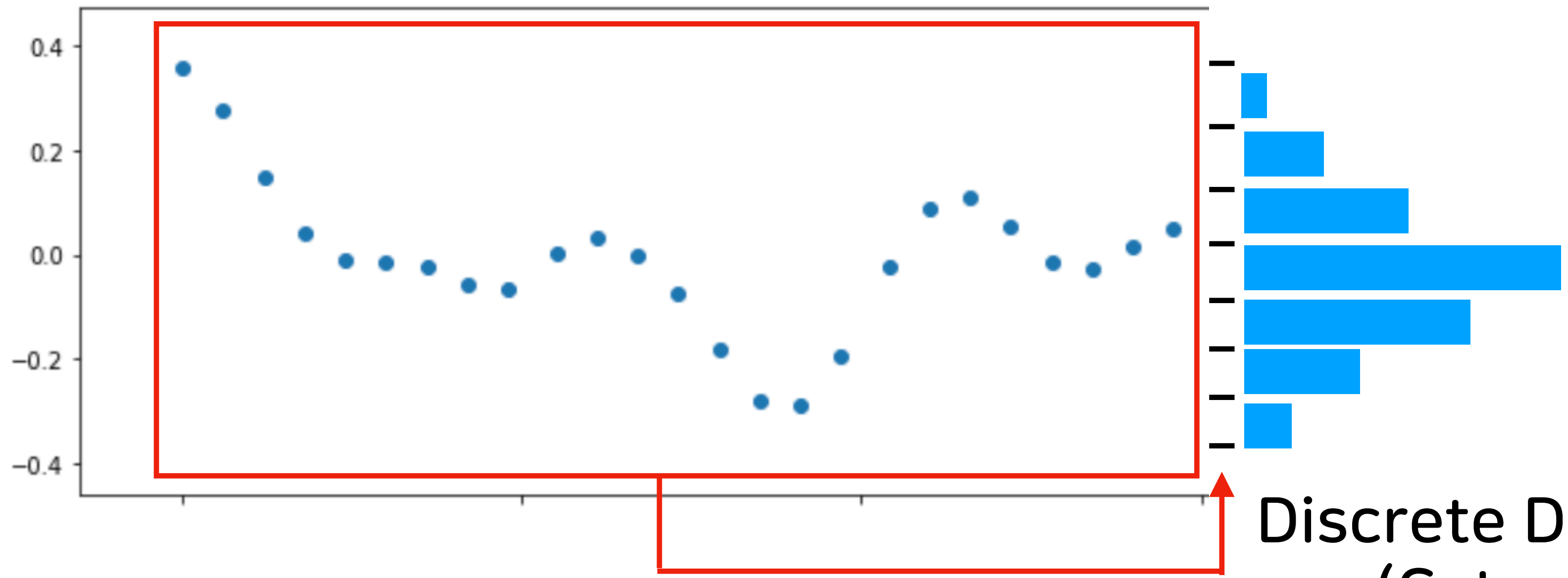
한단계 앞까지의 과거 데이터를 토대로 현재 시점의 데이터의 확률 분포를 예측



$x_t \mid \{x_1, x_2, \ldots, x_{t-1}\}$

Autoregressive Models

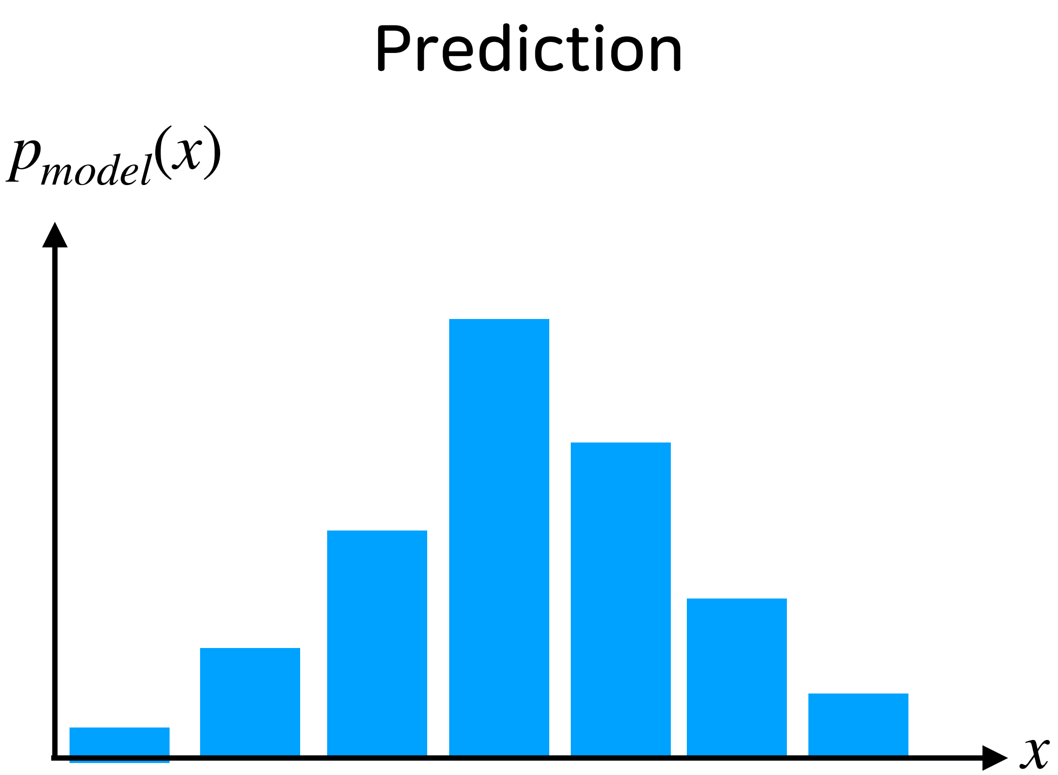
한단계 앞까지의 과거 데이터를 토대로 현재 시점의 데이터의 확률 분포를 예측

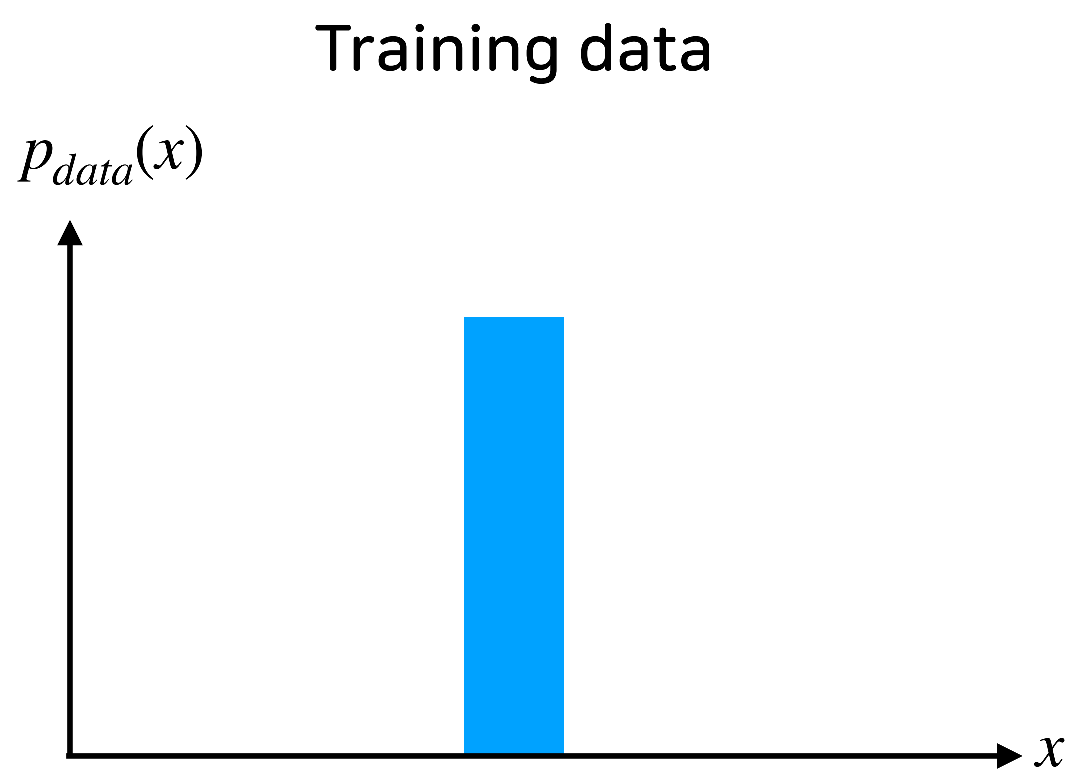


$$p(x_t \mid x_1, x_2, \ldots, x_{t-1})$$

Discrete Distribution
(Categorical)

Cross-Entropy Loss





$$H(p_{data}, p_{model}) = -\sum_{x} p_{data}(x) \log p_{model}(x)$$

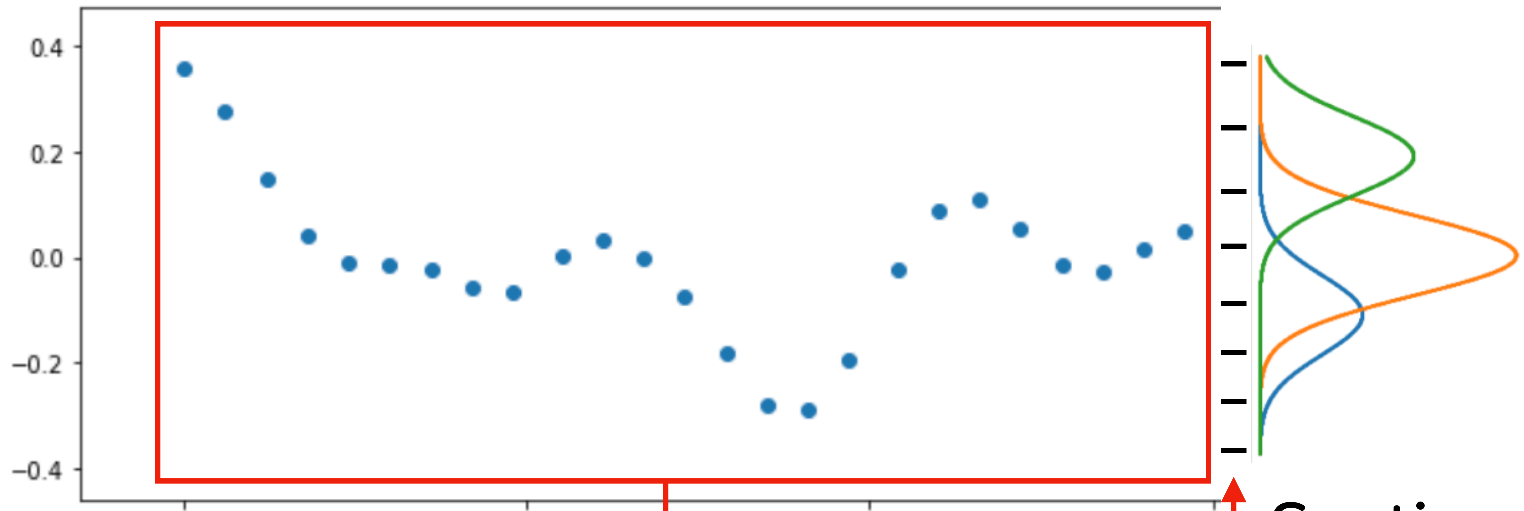
Cross-Entropy Loss

```
class MaskedCrossEntropyLoss(nn.Module):
    def __init__(self):
        super(MaskedCrossEntropyLoss, self).__init__()
        self.criterion = nn.CrossEntropyLoss(reduction='none')

    def forward(self, input, target, lengths=None, mask=None, max_len=None):
        if lengths is None and mask is None:
            raise RuntimeError("Should provide either lengths or mask")

        # (B, T, 1)
        if mask is None:
            mask = sequence_mask(lengths, max_len).unsqueeze(-1)

        # (B, T, D)
        mask_ = mask.expand_as(target)
        losses = self.criterion(input, target)
        return ((losses * mask_).sum()) / mask_.sum()
```

https://github.com/r9y9/wavenet_vocoder/blob/master/train.py

2. Autoregressive Models

한단계 앞까지의 과거 데이터를 토대로 현재 시점의 데이터의 확률 분포를 예측



$p(x_t | x_1, x_2, ..., x_{t-1})$

Chain Rule

$$p(X) = p(x_1, x_2, ..., x_T) = p(x_1)p(x_2 | x_1)p(x_3 | x_1, x_2)...p(x_T | x_1, ..., x_{T-1})$$

$$= \prod_{t=1}^{T} p(x_t | x_1, .., x_{t-1}) = \prod_{t=1}^{T} p(x_t | x_{<t})$$

Mixture Density Networks

Figure 5.19 On the left is the data set for a simple 'forward problem' in which the red curve shows the result of fitting a two-layer neural network by minimizing the sum-of-squares error function. The corresponding inverse problem, shown on the right, is obtained by exchanging the roles of x and t. Here the same network trained again by minimizing the sum-of-squares error function gives a very poor fit to the data due to the multimodality of the data set.

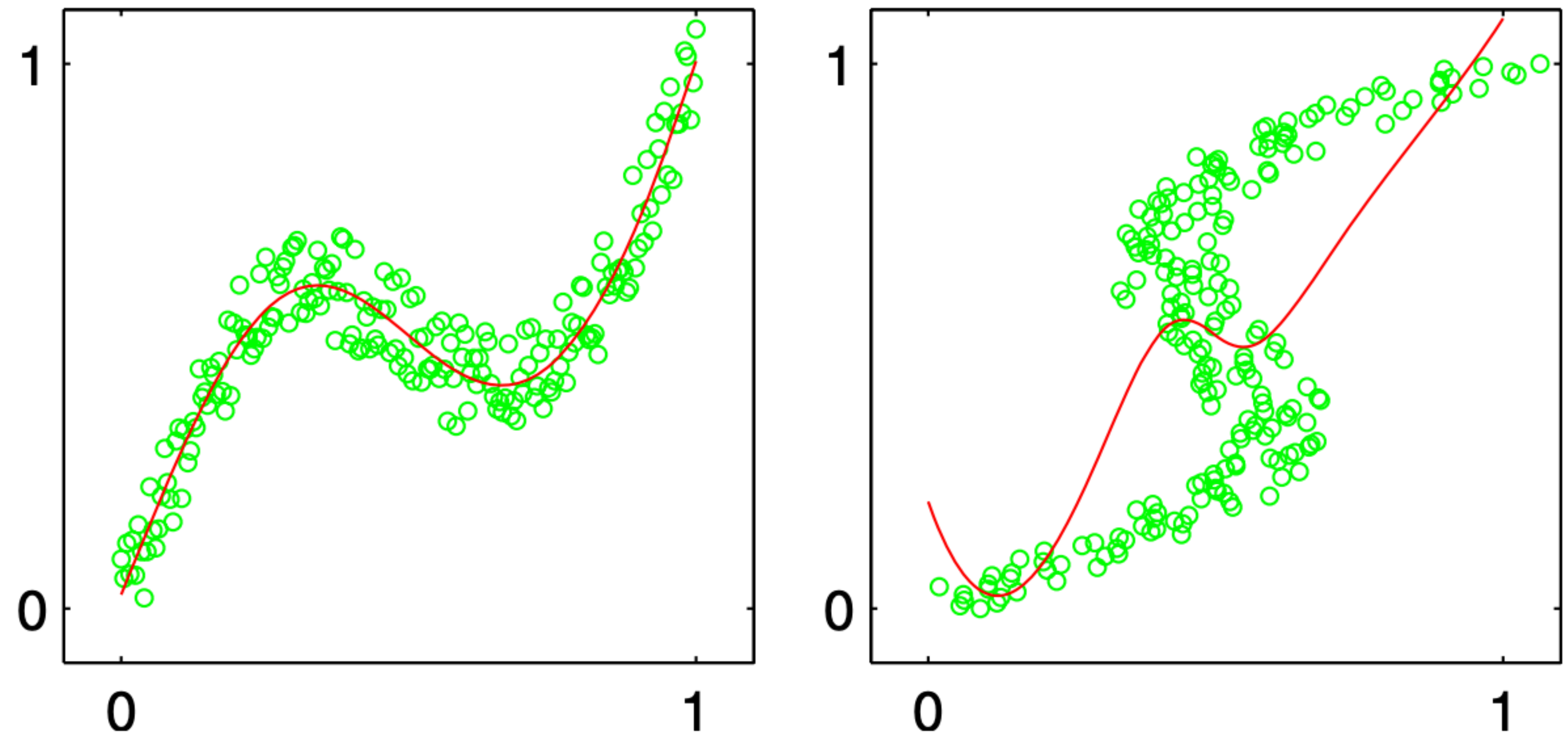


Figure credit: Christopher M. Bishop, Pattern Recognition And Machine Learning 2006, p.273

Mixture Density Networks

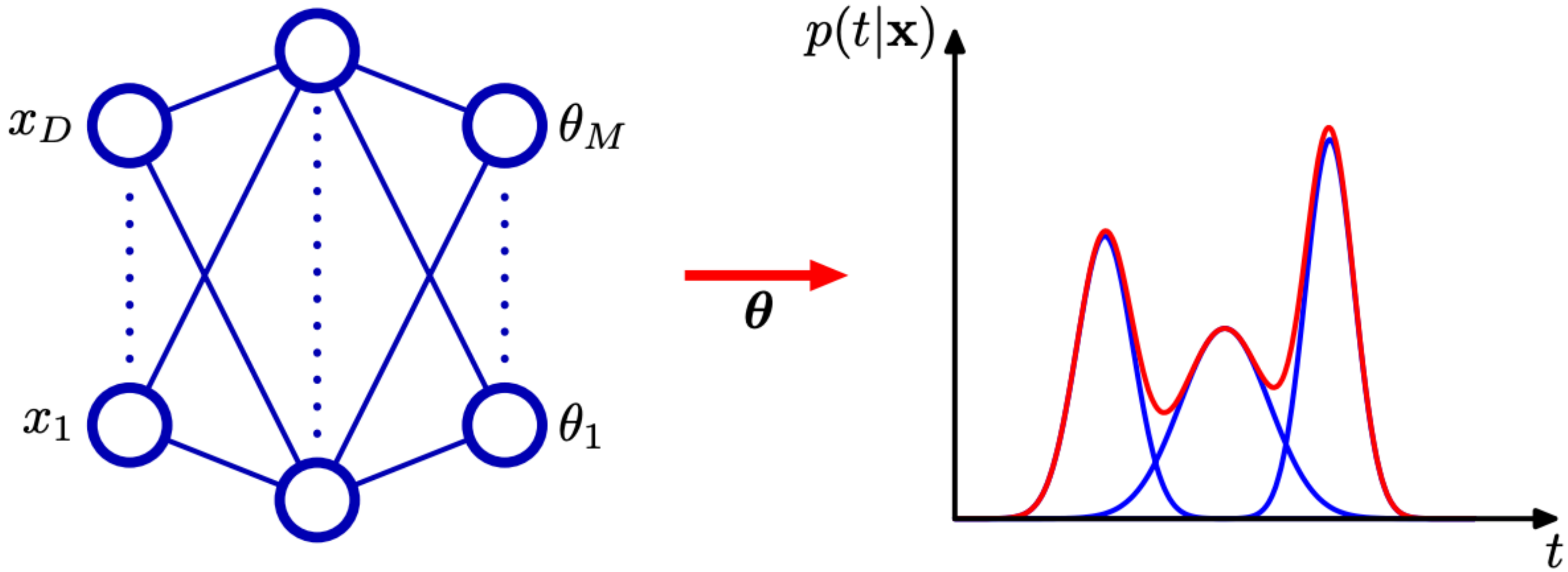


Figure 5.20 The *mixture density network* can represent general conditional probability densities $p(\mathbf{t}|\mathbf{x})$ by considering a parametric mixture model for the distribution of $\mathbf{t}$ whose parameters are determined by the outputs of a neural network that takes $\mathbf{x}$ as its input vector.

$$p(\mathbf{t}\,|\,\mathbf{x}) = \sum_{k=1}^{K} \pi_k(\mathbf{x}) \mathcal{N}\left(\mathbf{t}\,|\,\boldsymbol{\mu}_k(\mathbf{x}), \sigma_k^2(\mathbf{x})\right)$$

Figure credit: Christopher M. Bishop, Pattern Recognition And Machine Learning 2006, p.274

Mixture Density Networks

The Mixing coefficients constraints

$$\sum_{k=1}^{K} \pi_k(\mathbf{x}) = 1, \quad 0 \leqslant \pi_k(\mathbf{x}) \leqslant 1$$

which can be achieved by softmax

$$\pi_k(\mathbf{x}) = \frac{\exp\left(a_k^{\pi}\right)}{\sum_{l=1}^{K} \exp\left(a_l^{\pi}\right)}$$

The Variances should satisfy

$$\sigma_k^2(\mathbf{x}) \geqslant 0$$

can be represented by the exponentials

$$\sigma_k(\mathbf{x}) = \exp\left(a_k^{\sigma}\right)$$

The Means have real components

$$\mu_{kj}(\mathbf{x}) = a_{kj}^{\mu}$$

K: # of components k: component index

Mixture Density Networks

$$Loss(data, \theta) = \mathbb{E}_{p_{data}(\mathbf{x})}[-\log p_\theta(\mathbf{x})]$$

$$\approx \sum_{n}^{N} [-\log p_\theta(\mathbf{x}_n)], where\ \mathbf{x}_n \sim p_{data}(\mathbf{x})$$

$$= -\sum_{n=1}^{N} \log \left\{ \sum_{k=1}^{K} \pi_k\left(\mathbf{x}_n, \theta\right) \mathcal{N}\left(\mathbf{t}_n | \boldsymbol{\mu}_k\left(\mathbf{x}_n, \theta\right), \sigma_k^2\left(\mathbf{x}_n, \theta\right)\right) \right\}$$

Mixture Density Networks

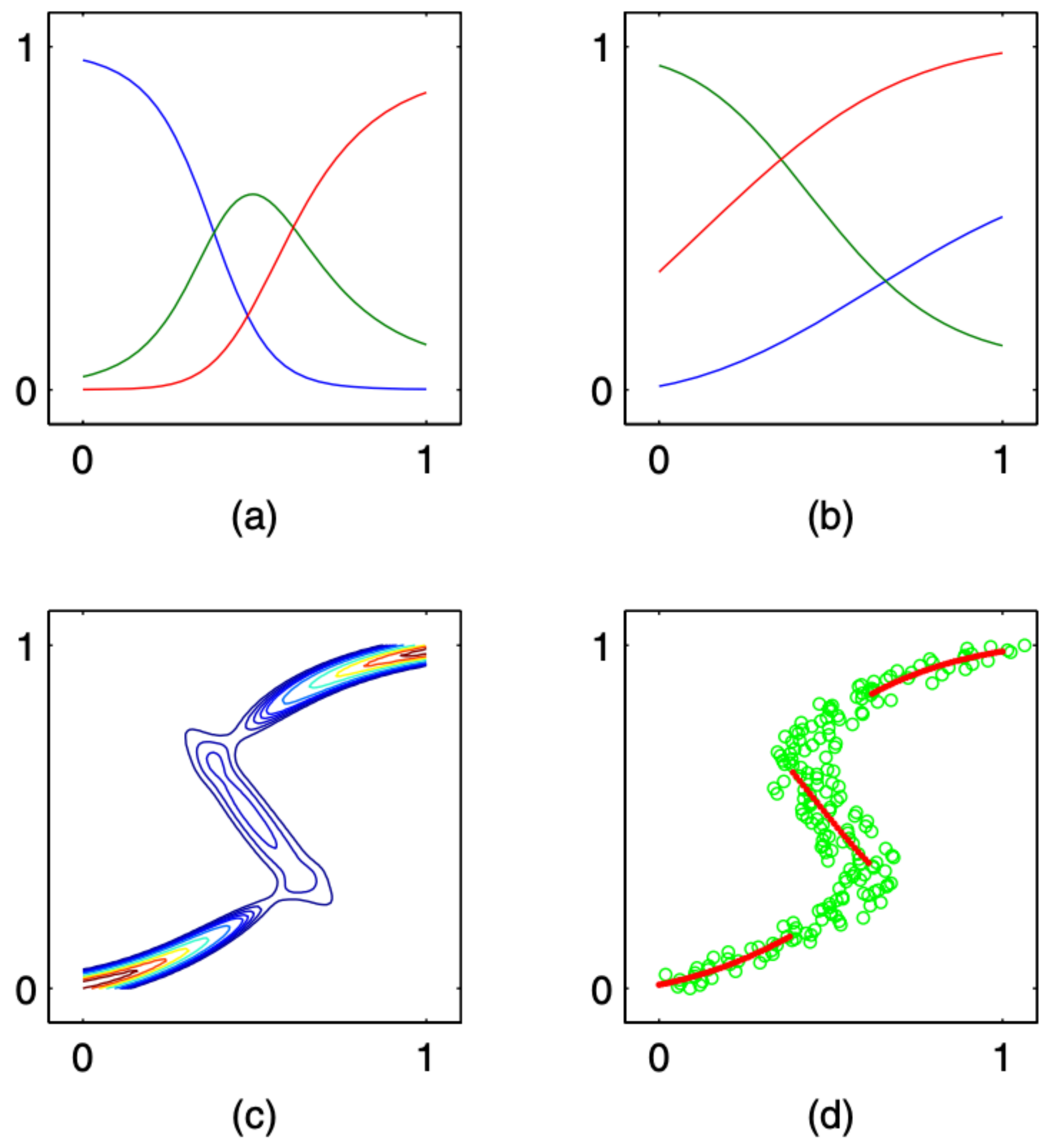
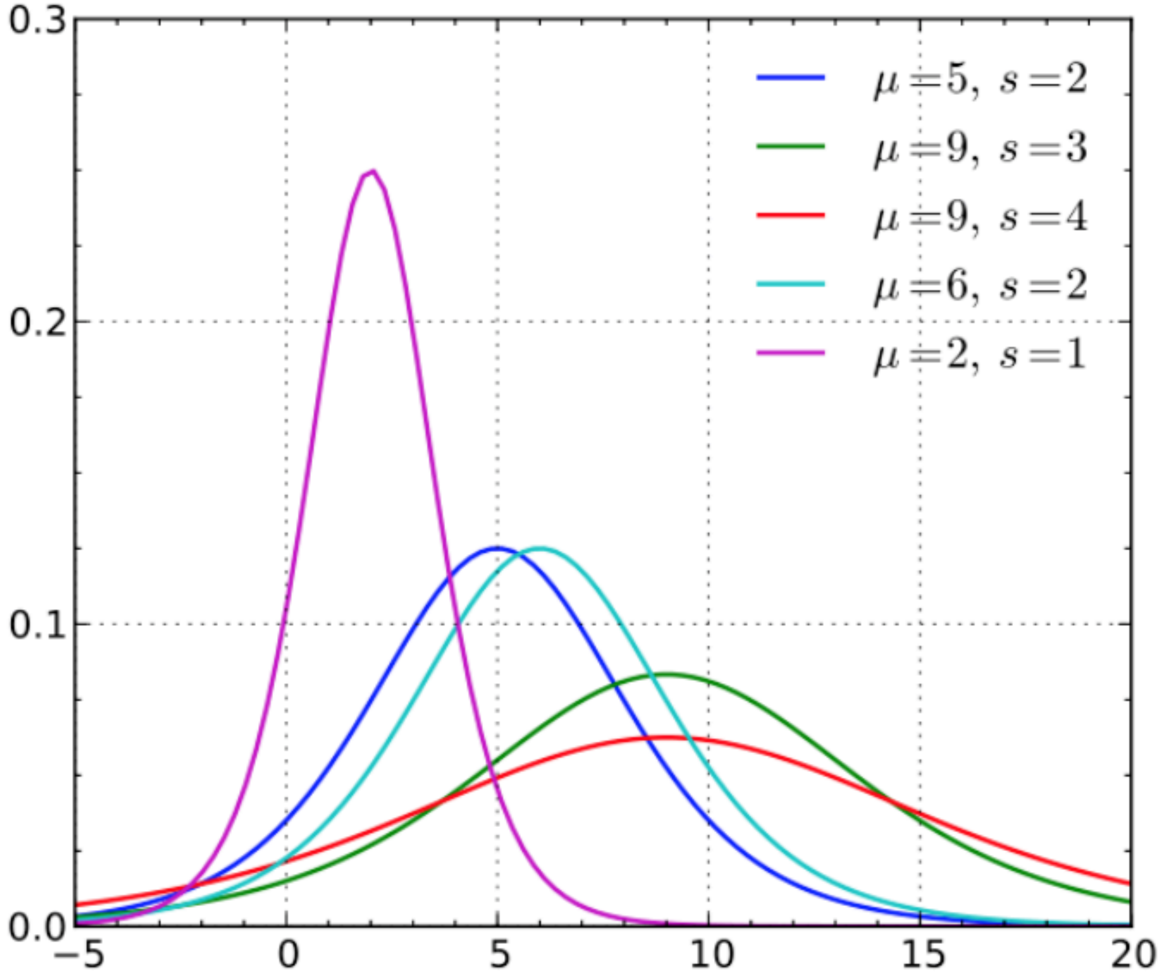


Figure 5.21 (a) Plot of the mixing coefficients $\pi_k(x)$ as a function of x for the three kernel functions in a mixture density network trained on the data shown in Figure 5.19. The model has three Gaussian components, and uses a two-layer multi-layer perceptron with five 'tanh' sigmoidal units in the hidden layer, and nine outputs (corresponding to the 3 means and 3 variances of the Gaussian components and the 3 mixing coefficients). At both small and large values of x, where the conditional probability density of the target data is unimodal, only one of the kernels has a high value for its prior probability, while at intermediate values of x, where the conditional density is trimodal, the three mixing coefficients have comparable values. (b) Plots of the means $\mu_k(x)$ using the same colour coding as for the mixing coefficients. (c) Plot of the contours of the corresponding conditional probability density of the target data for the same mixture density network. (d) Plot of the approximate conditional mode, shown by the red points, of the conditional density.

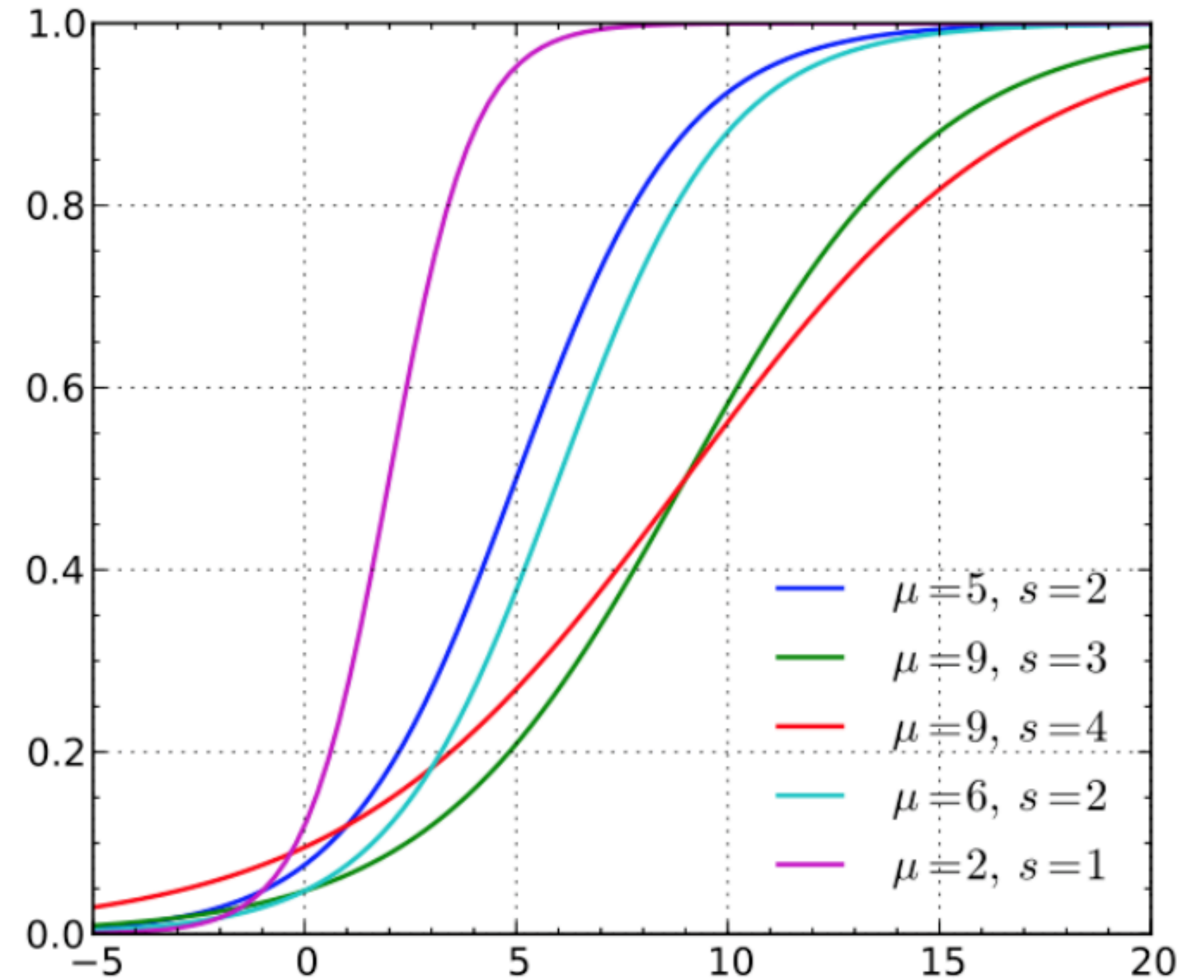
Logistic Mixture

PDF

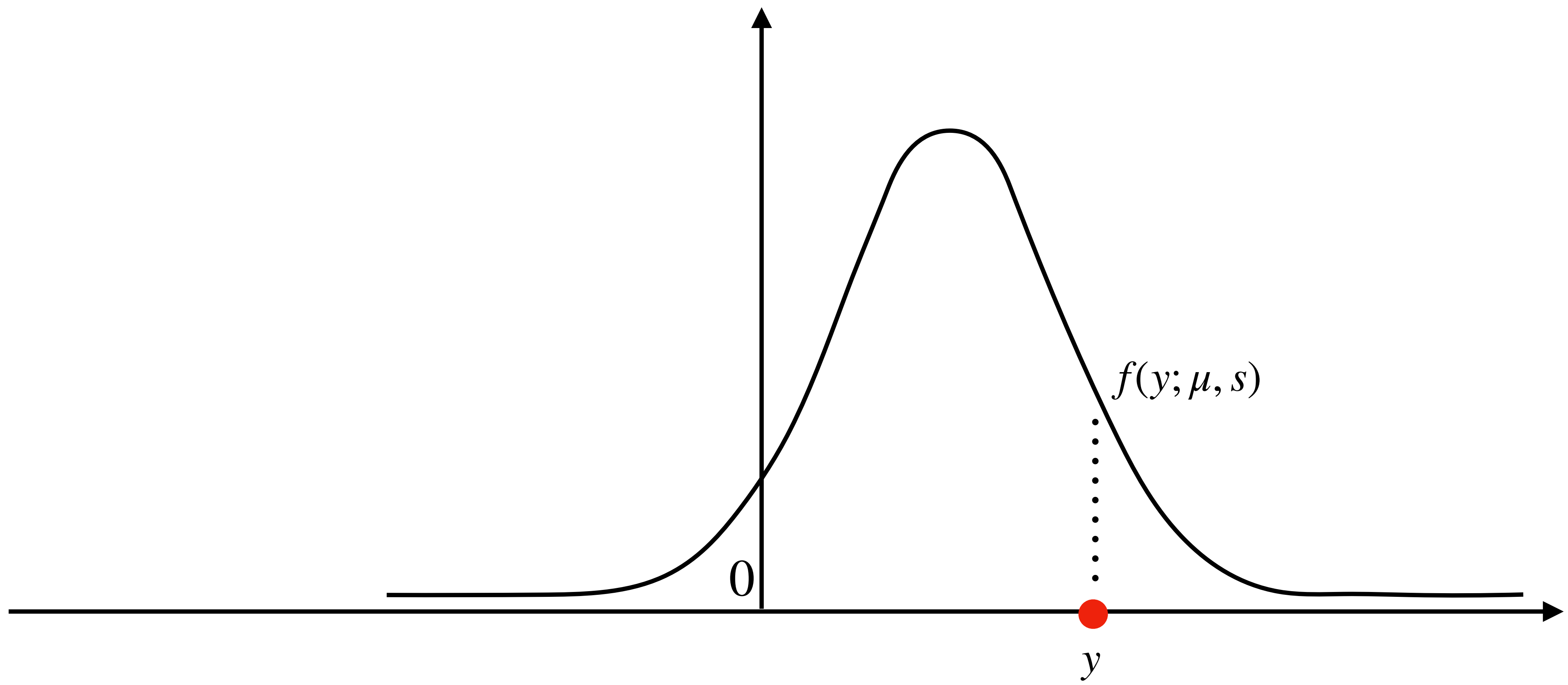


$$f(y;\mu,s)=\frac{1}{4s}\operatorname{sech}^2\left(\frac{x-\mu}{2s}\right)$$

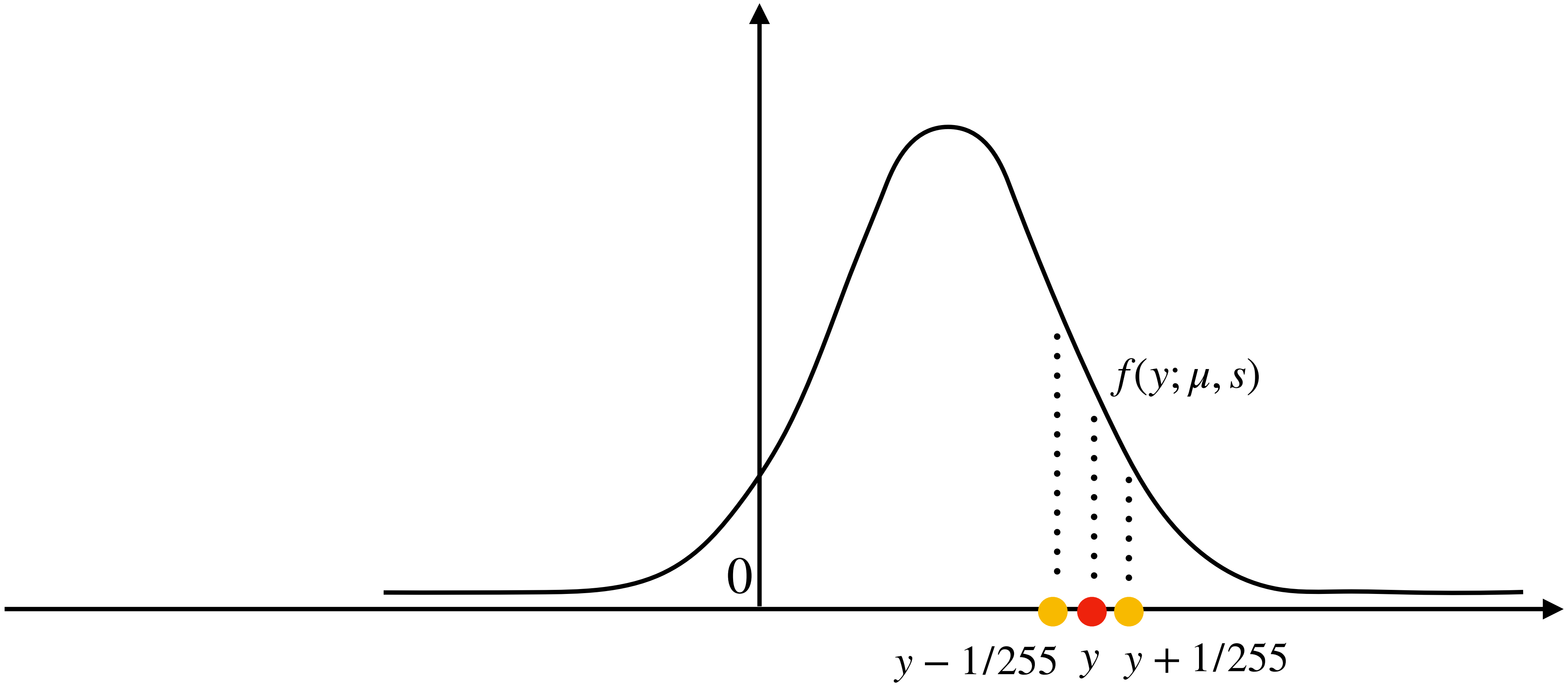
CDF



$$F(y;\mu,s)=\frac{1}{1+e^{-(y-\mu)/s}}$$



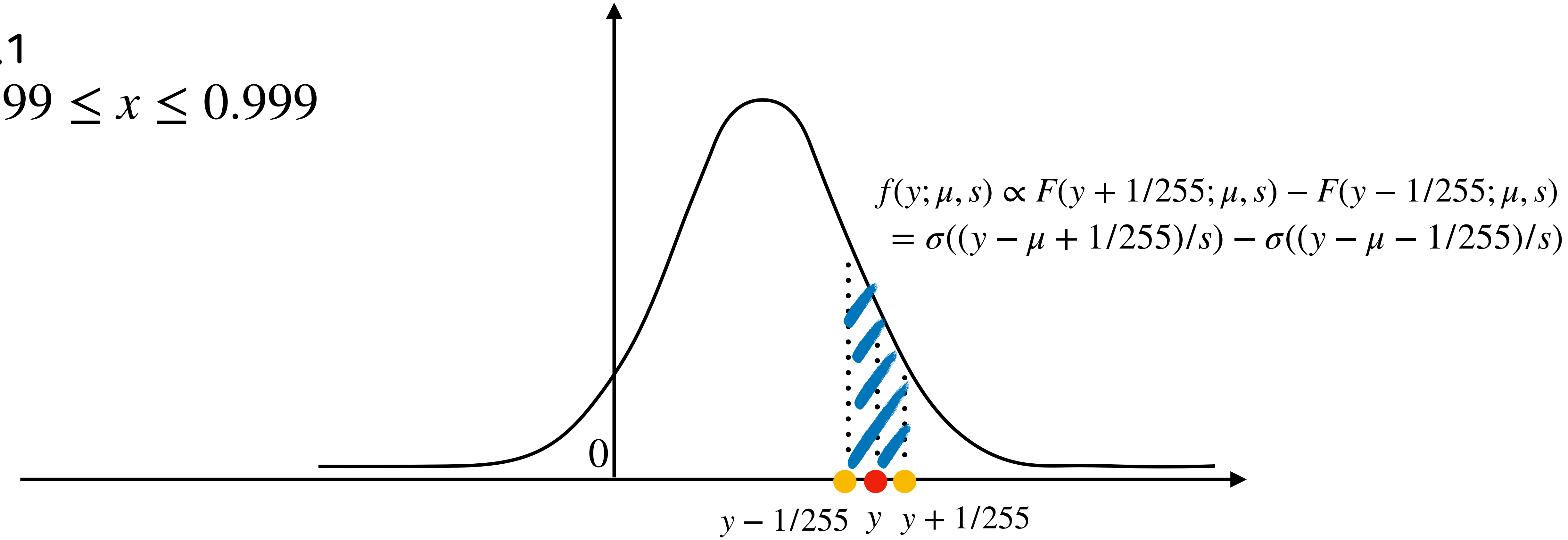
$f(y; \mu, s)$
0
y



$f(y; \mu, s)$
0
$y - 1/255$
y
$y + 1/255$

Case.1

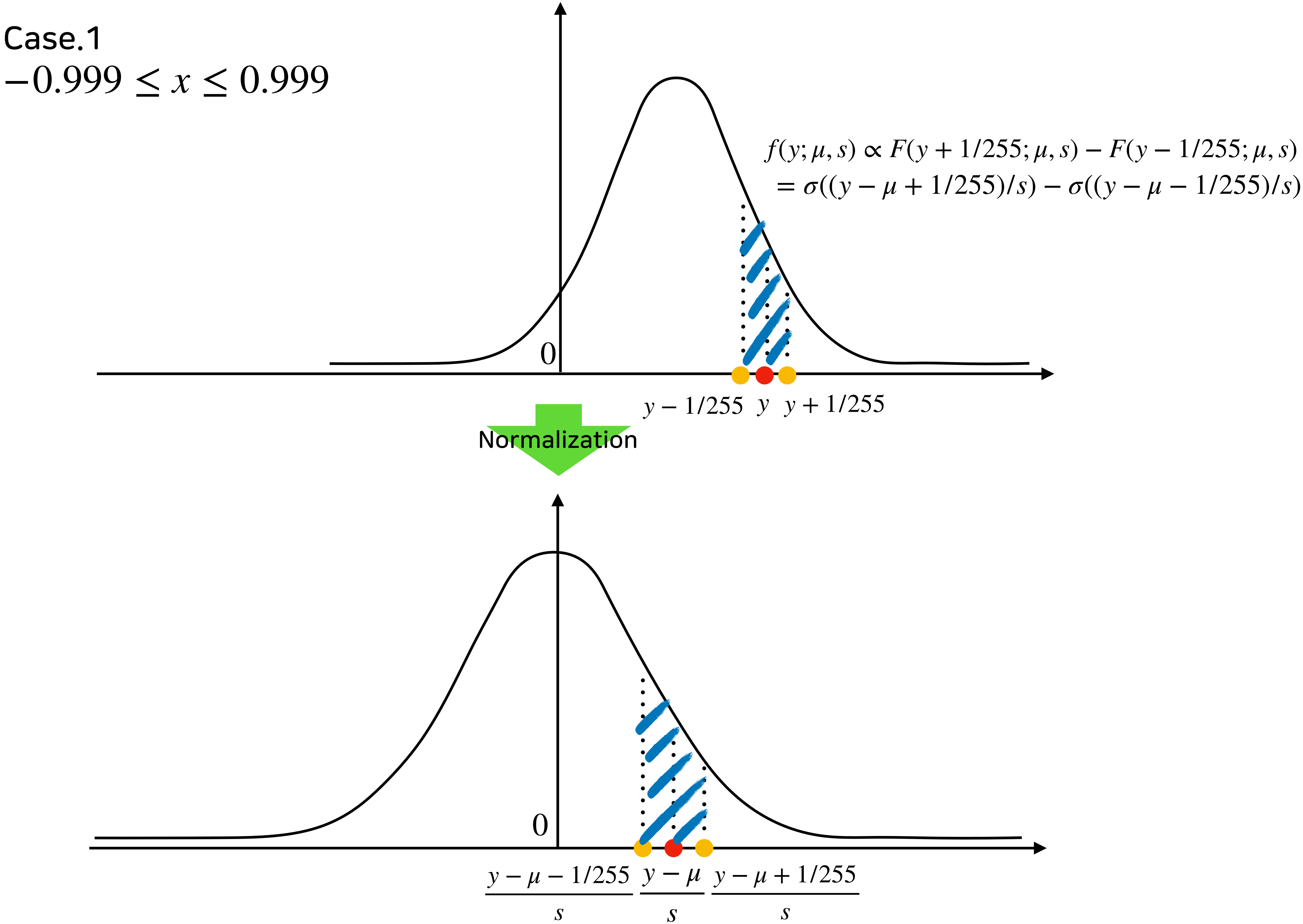
$-0.999 \le x \le 0.999$



$$\nu \sim \sum_{i=1}^{K} \pi_i \text{logistic}(\mu_i, s_i) \tag{1}$$

$$P(x|\pi,\mu,s) = \sum_{i=1}^{K} \pi_i \left[\sigma((x+0.5-\mu_i)/s_i) - \sigma((x-0.5-\mu_i)/s_i)\right], \tag{2}$$

Tim Salimans et al. PixelCNN++: Improving the PixelCNN with Discretized Logistic Mixture Likelihood and Other Modifications

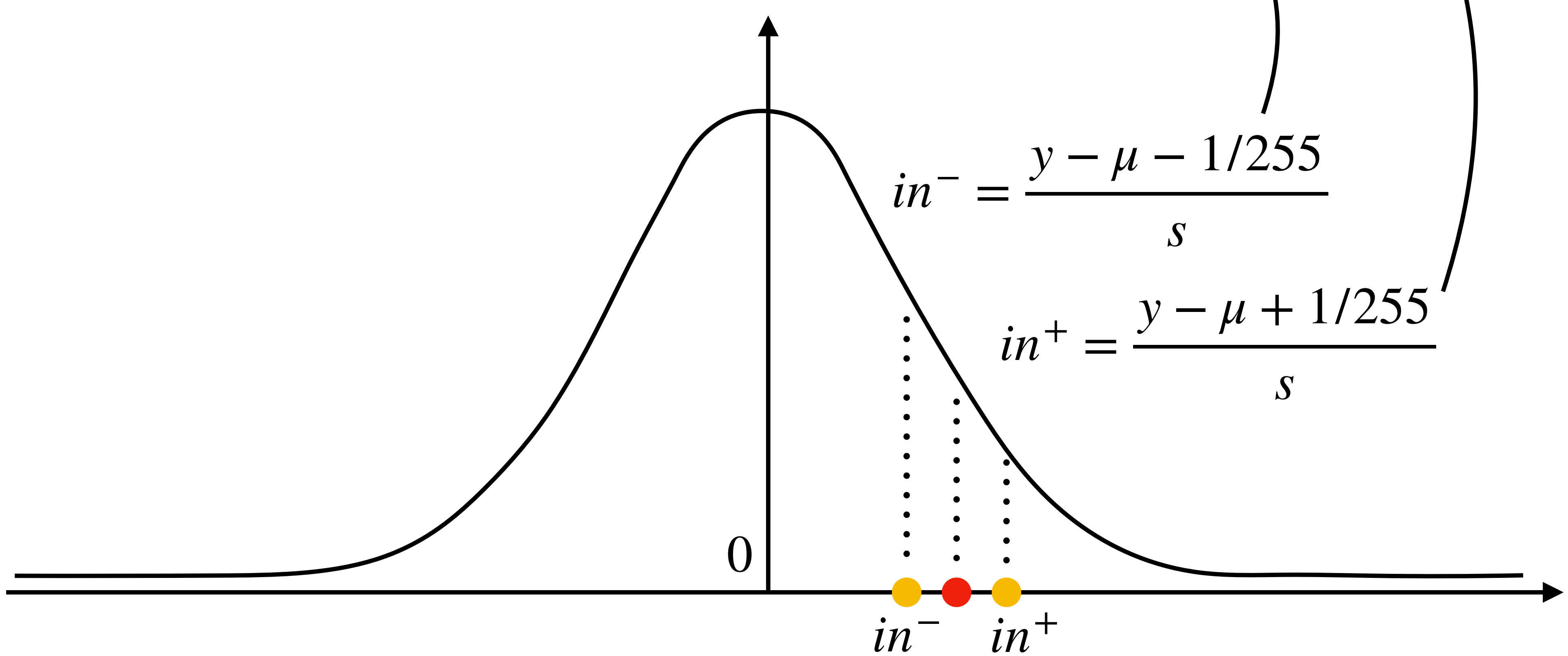


Case.1
$-0.999 \leq x \leq 0.999$
$f(y;\mu,s) \propto F(y+1/255;\mu,s) - F(y-1/255;\mu,s)$
$= \sigma((y-\mu+1/255)/s) - \sigma((y-\mu-1/255)/s)$
0
$y-1/255$
y
$y+1/255$
Normalization
0
$\frac{y-\mu-1/255}{s}$
$\frac{y-\mu}{s}$
$\frac{y-\mu+1/255}{s}$

Case.1

$$-0.999 \leq x \leq 0.999$$

```python
centered_y = y - means
inv_stdv = torch.exp(-log_scales)
plus_in = inv_stdv * (centered_y + 1. / (num_classes - 1))
cdf_plus = torch.sigmoid(plus_in)
min_in = inv_stdv * (centered_y - 1. / (num_classes - 1))
cdf_min = torch.sigmoid(min_in)
```

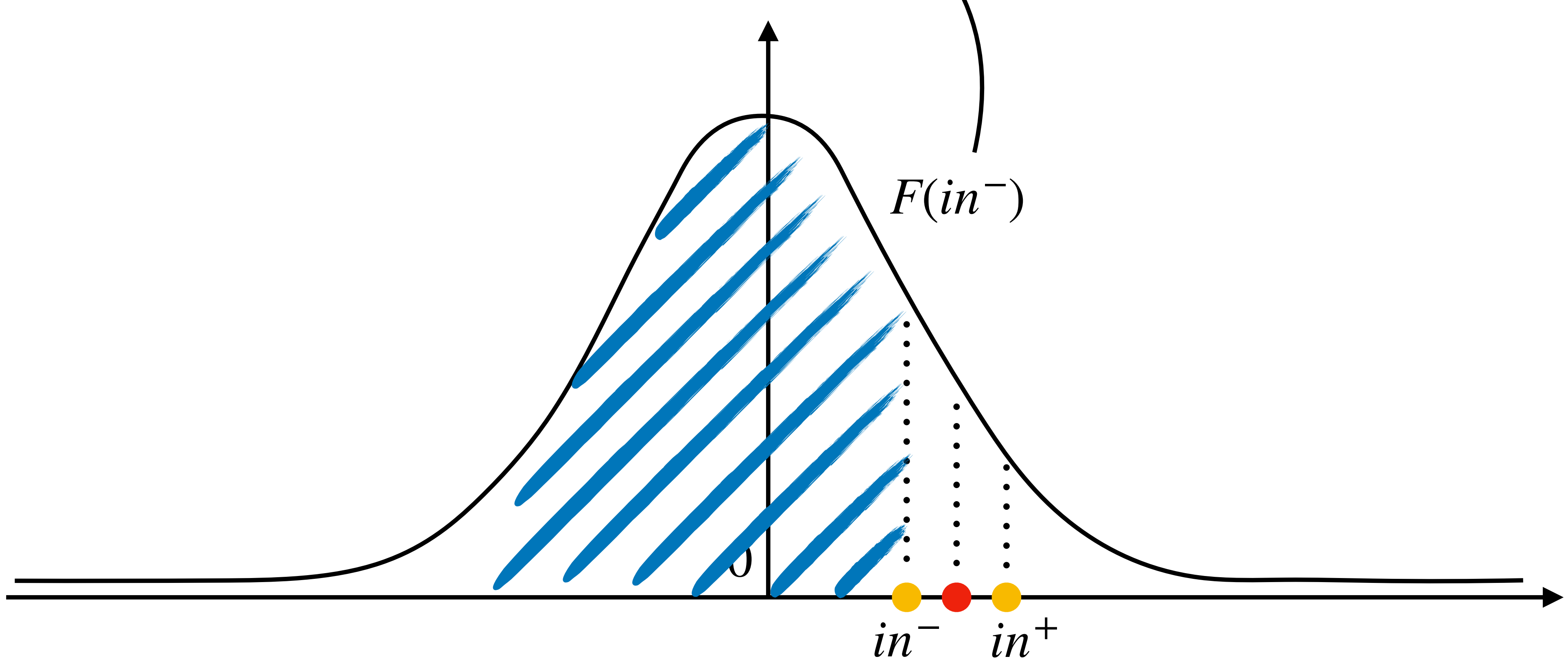


Source reference: https://github.com/r9y9/wavenet_vocoder/blob/master/wavenet_vocoder/mixture.py

Case.1

$$-0.999 \leq x \leq 0.999$$

```
centered_y = y - means
inv_stdv = torch.exp(-log_scales)
plus_in = inv_stdv * (centered_y + 1. / (num_classes - 1))
cdf_plus = torch.sigmoid(plus_in)
min_in = inv_stdv * (centered_y - 1. / (num_classes - 1))
cdf_min = torch.sigmoid(min_in)
```

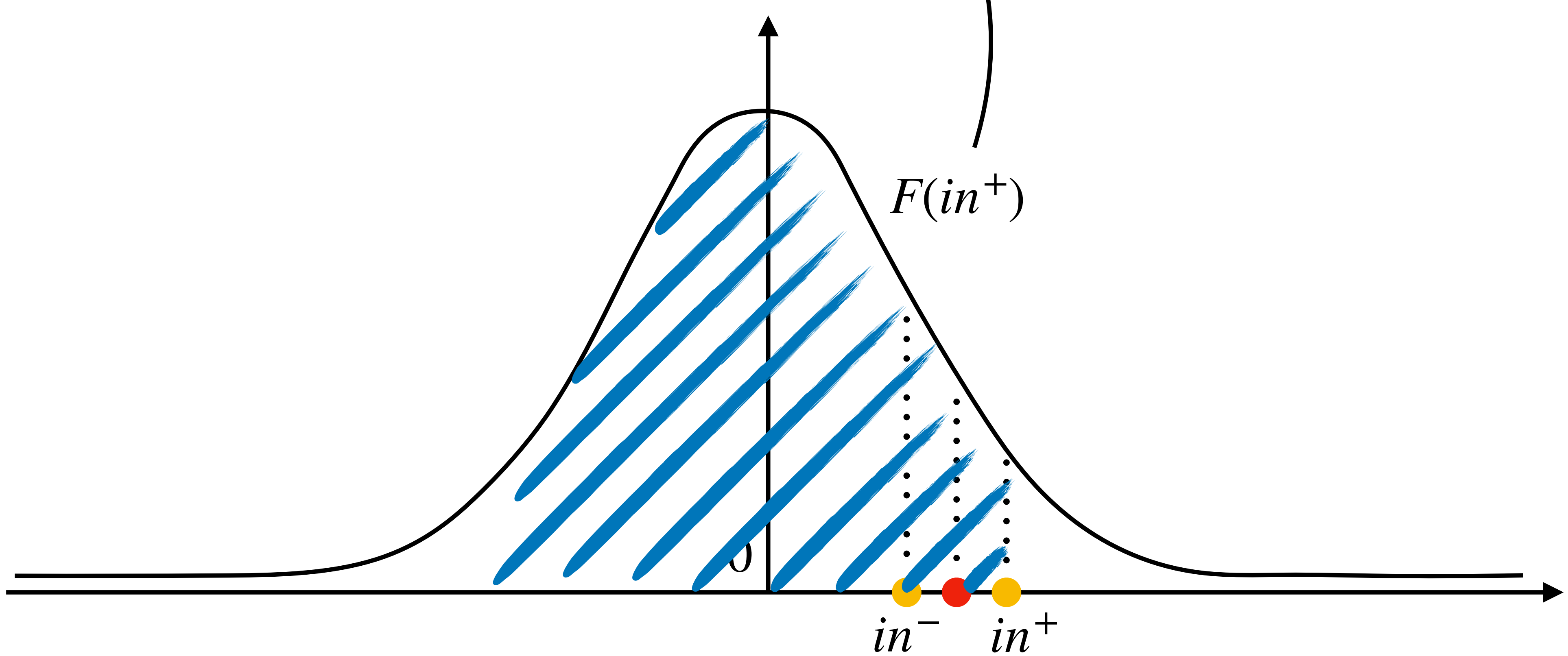


Source reference: https://github.com/r9y9/wavenet_vocoder/blob/master/wavenet_vocoder/mixture.py

Case.1

$$-0.999 \leq x \leq 0.999$$

```
centered_y = y - means
inv_stdv = torch.exp(-log_scales)
plus_in = inv_stdv * (centered_y + 1. / (num_classes - 1))
cdf_plus = torch.sigmoid(plus_in)
min_in = inv_stdv * (centered_y - 1. / (num_classes - 1))
cdf_min = torch.sigmoid(min_in)
```

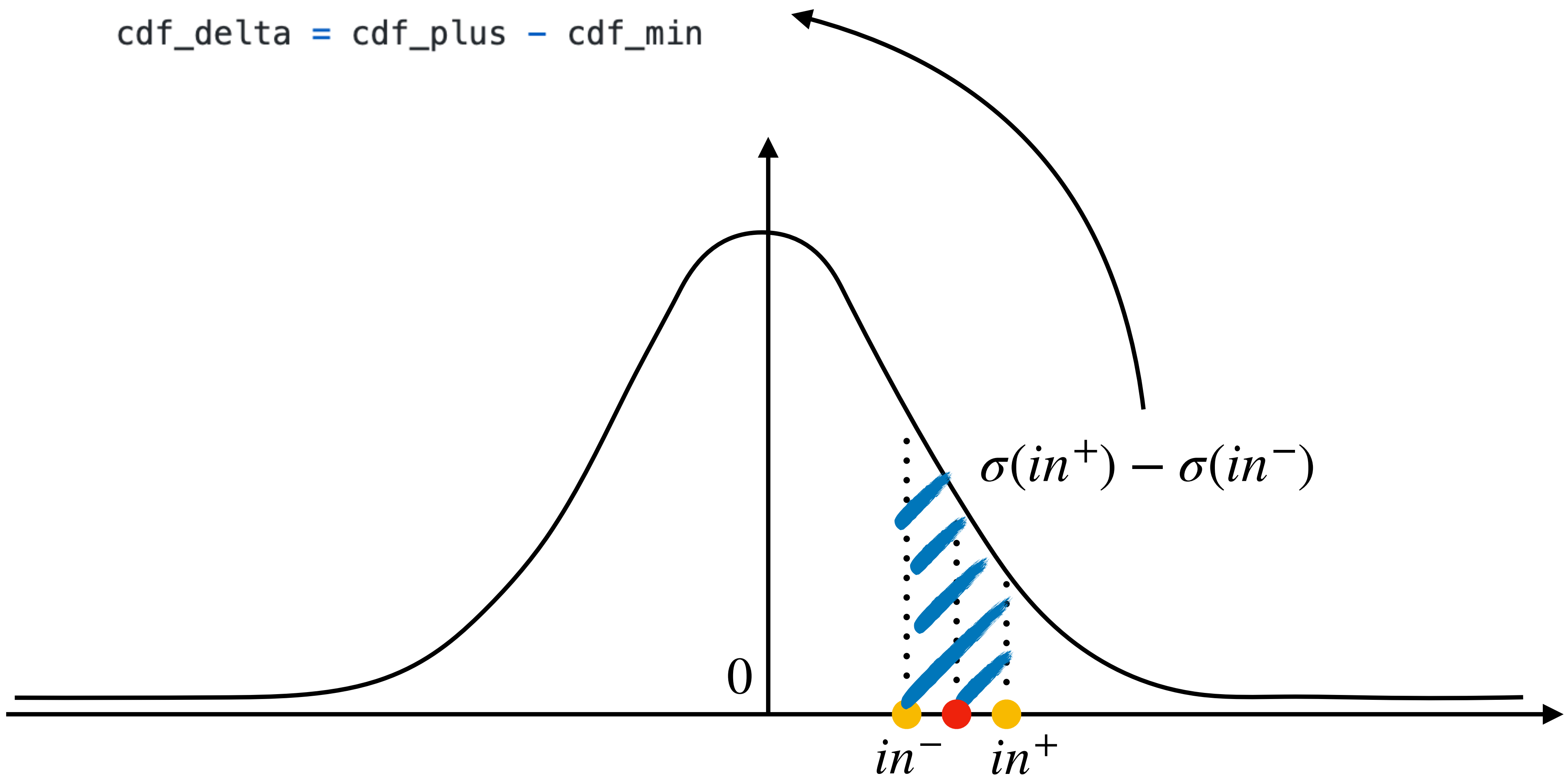


Source reference: https://github.com/r9y9/wavenet_vocoder/blob/master/wavenet_vocoder/mixture.py

Case.1

$$-0.999 \leq x \leq 0.999$$

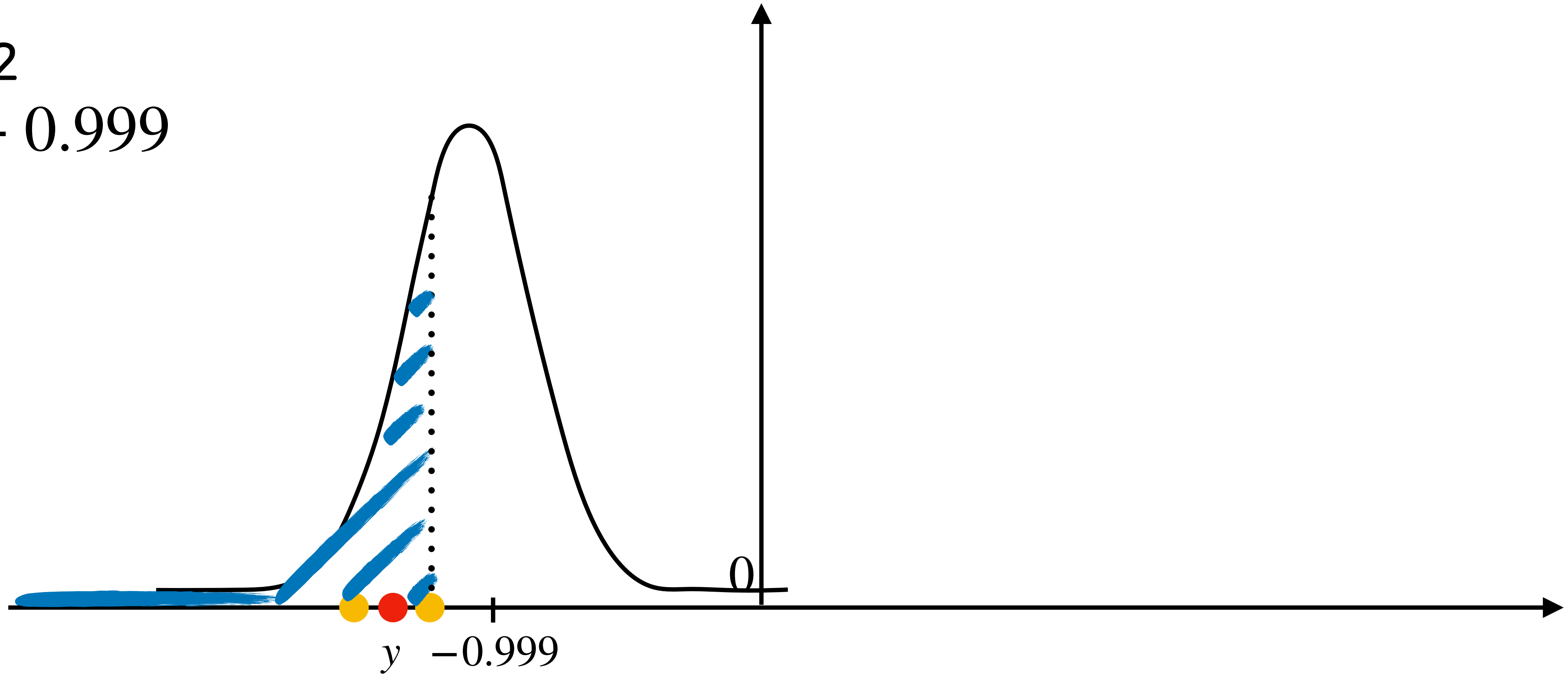
```python
# log probability for edge case of 0 (before scaling)
# equivalent: torch.log(torch.sigmoid(plus_in))
log_cdf_plus = plus_in - F.softplus(plus_in)

# log probability for edge case of 255 (before scaling)
# equivalent: (1 - torch.sigmoid(min_in)).log()
log_one_minus_cdf_min = -F.softplus(min_in)

# probability for all other cases
cdf_delta = cdf_plus - cdf_min
```



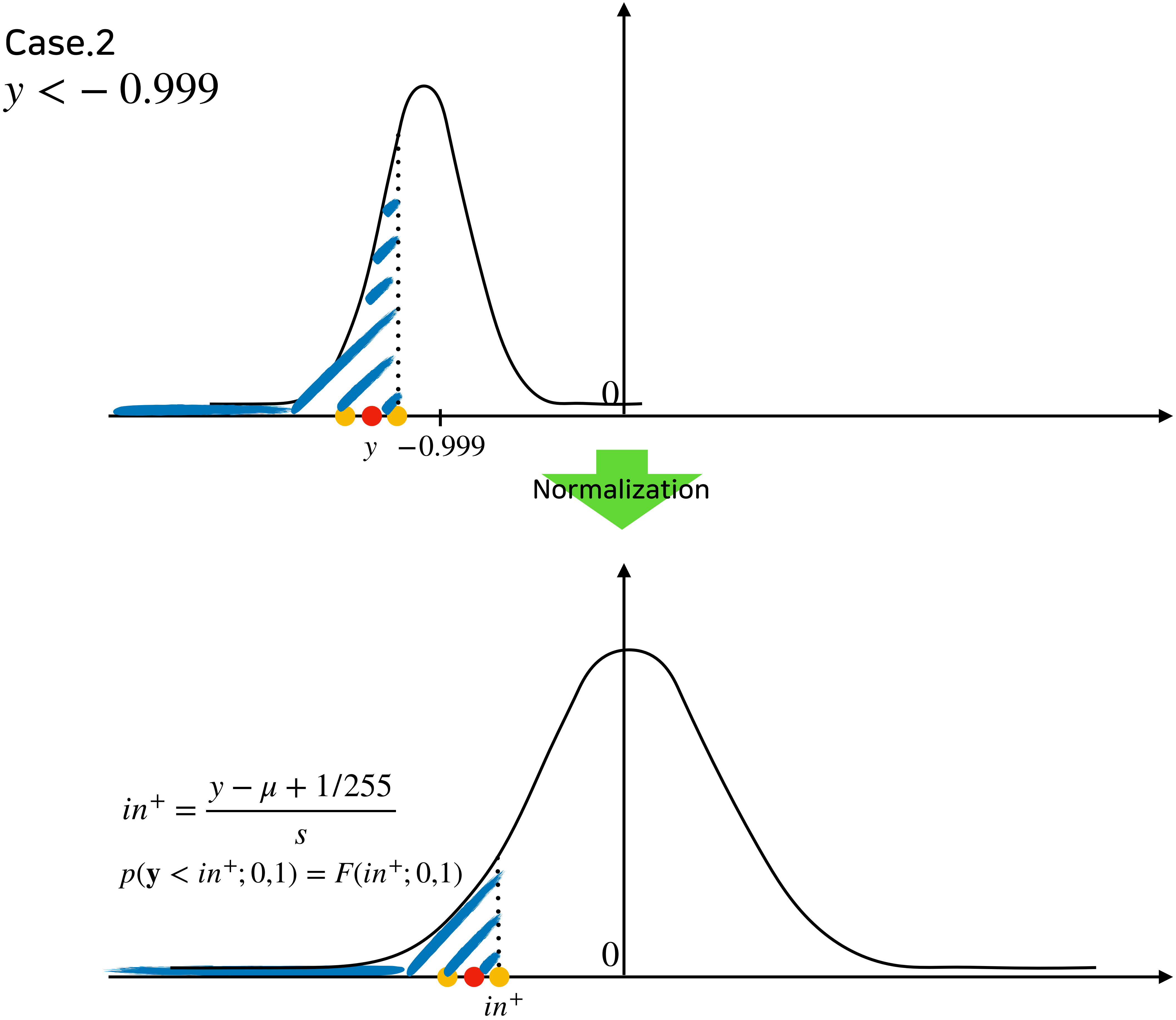
Source reference: https://github.com/r9y9/wavenet_vocoder/blob/master/wavenet_vocoder/mixture.py

Case.2

$y < -0.999$



$p(\mathbf{y} < y + 1/255; \mu, s) = F(y + 1/255; \mu, s)$



Case.2
$y < -0.999$
0
y
-0.999
Normalization
$in^{+} = \frac{y - \mu + 1/255}{s}$
$p(\mathbf{y} < in^{+}; 0,1) = F(in^{+}; 0,1)$
0
in^{+}

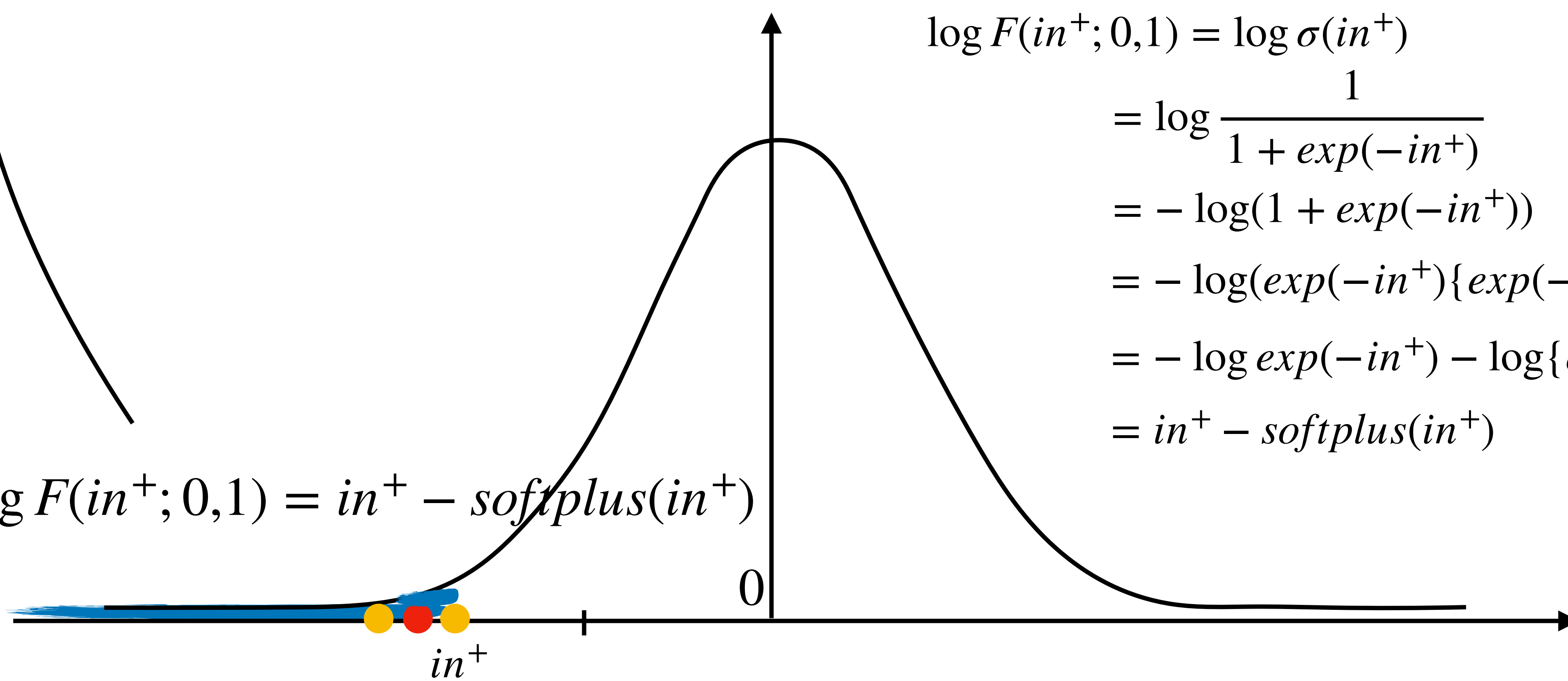
Case.2

$y < -0.999$

```python
# log probability for edge case of 0 (before scaling)
# equivalent: torch.log(torch.sigmoid(plus_in))
log_cdf_plus = plus_in - F.softplus(plus_in)

# log probability for edge case of 255 (before scaling)
# equivalent: (1 - torch.sigmoid(min_in)).log()
log_one_minus_cdf_min = -F.softplus(min_in)

# probability for all other cases
cdf_delta = cdf_plus - cdf_min
```

$$
\begin{aligned}
\log F(in^+;0,1) &= \log \sigma(in^+) \\
&= \log \frac{1}{1+exp(-in^+)} \\
&= -\log(1+exp(-in^+)) \\
&= -\log(exp(-in^+)\{exp(-in^+)+1\}) \\
&= -\log exp(-in^+) - \log\{exp(-in^+)+1\} \\
&= in^+ - softplus(in^+)
\end{aligned}
$$

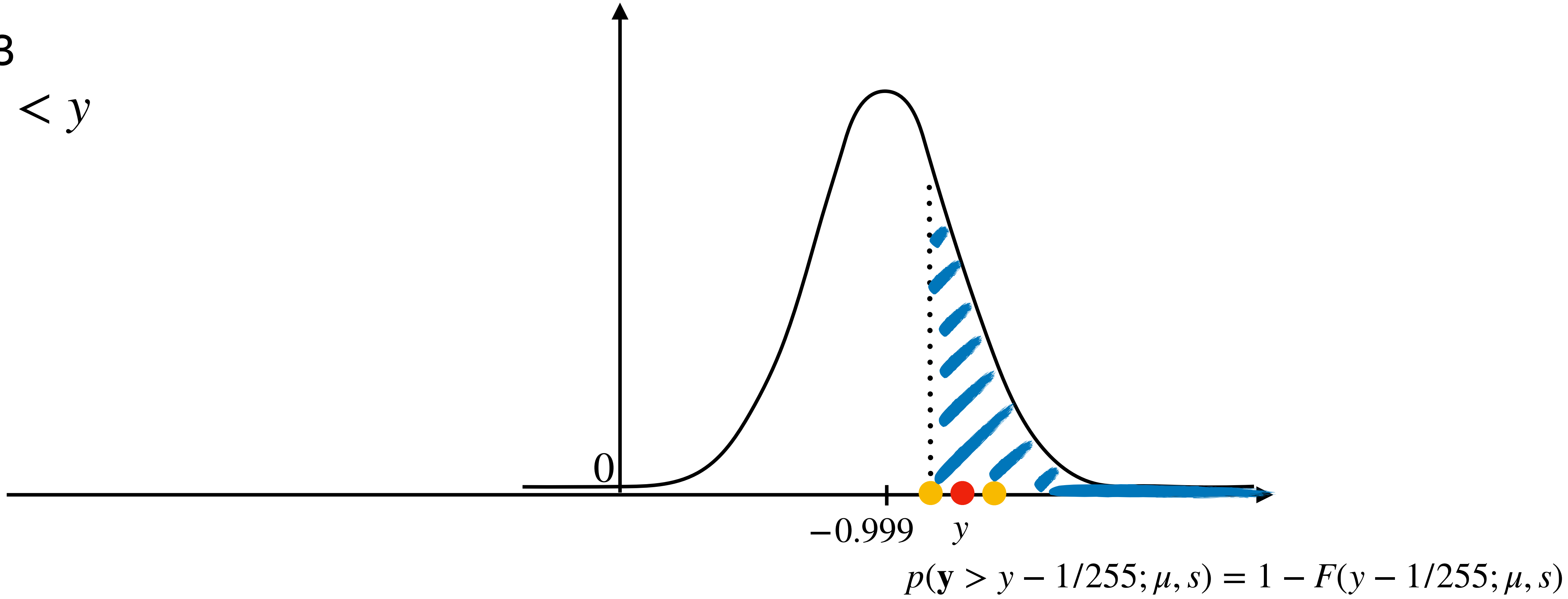
$$\log F(in^+;0,1) = in^+ - softplus(in^+)$$

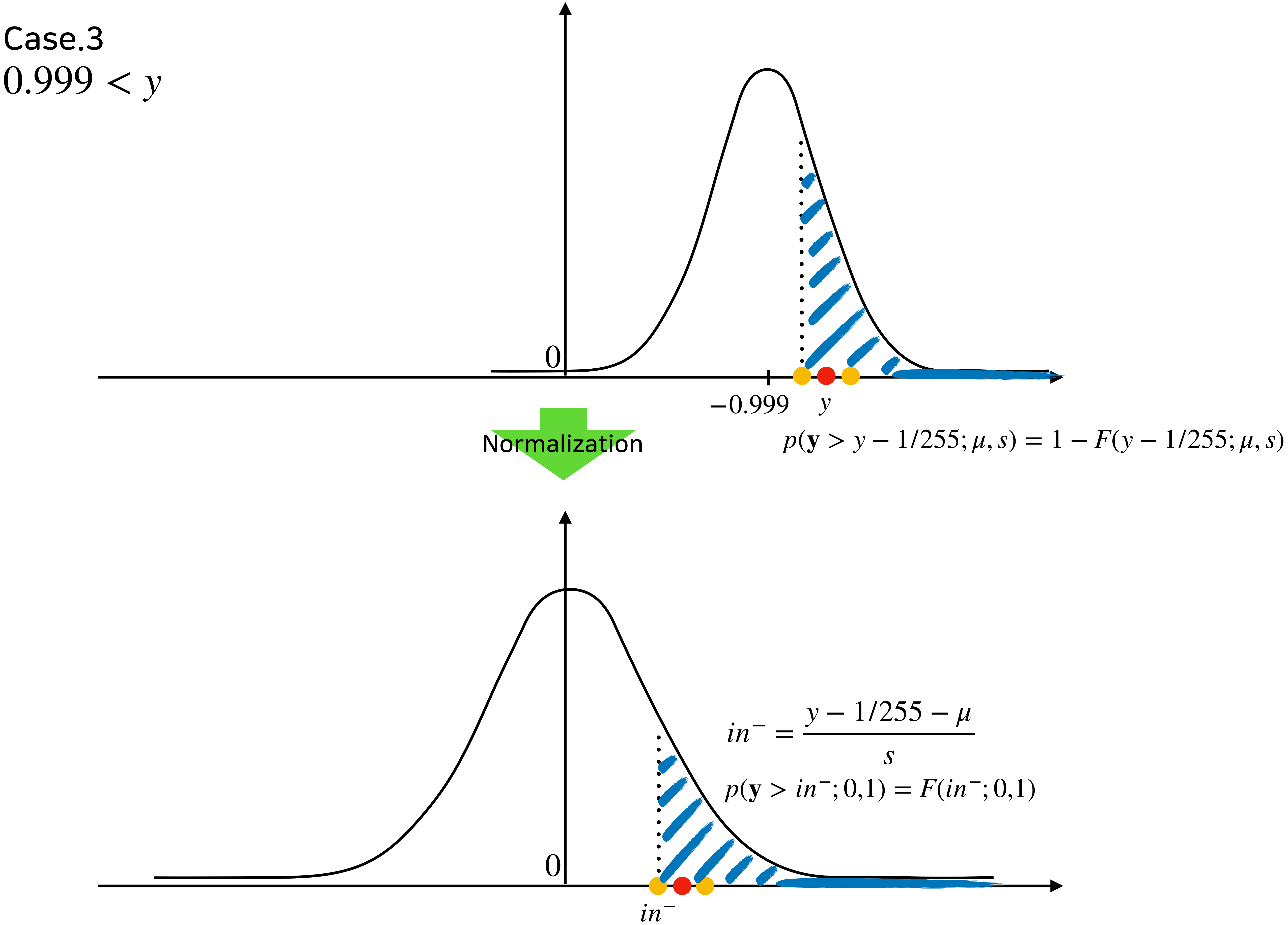


Source reference: https://github.com/r9y9/wavenet_vocoder/blob/master/wavenet_vocoder/mixture.py

Case.3

$0.999 < y$





Case.3
$0.999 < y$
0
-0.999
y
$p(\mathbf{y} > y - 1/255; \mu, s) = 1 - F(y - 1/255; \mu, s)$
Normalization
$in^{-} = \dfrac{y - 1/255 - \mu}{s}$
$p(\mathbf{y} > in^{-}; 0,1) = F(in^{-}; 0,1)$
0
in^{-}

Case.3

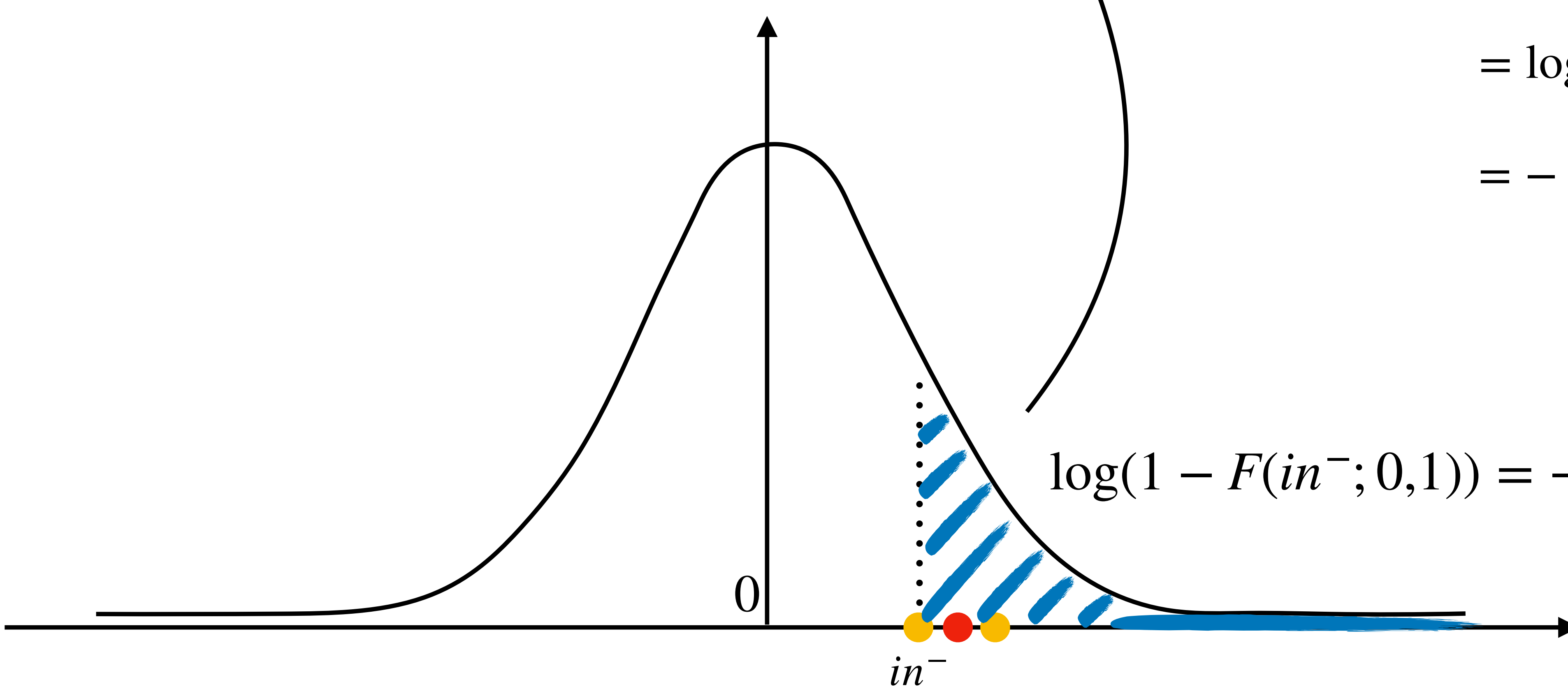
$0.999 < y$

```
        # log probability for edge case of 0 (before scaling)
        # equivalent: torch.log(torch.sigmoid(plus_in))
        log_cdf_plus = plus_in - F.softplus(plus_in)

        # log probability for edge case of 255 (before scaling)
        # equivalent: (1 - torch.sigmoid(min_in)).log()
        log_one_minus_cdf_min = -F.softplus(min_in)

        # probability for all other cases
        cdf_delta = cdf_plus - cdf_min
```

$$\log(1 - F(in^{-};0,1)) = \log\left(1 - \frac{1}{1+e^{-in^{-}}}\right)$$

$$= \log\left(\frac{1}{1+e^{in^{-}}}\right)$$

$$= -\,softplus(in^{-})$$



$$\log(1 - F(in^{-};0,1)) = -\,softplus(in^{-})$$

Source reference: https://github.com/r9y9/wavenet_vocoder/blob/master/wavenet_vocoder/mixture.py

Case.4

$$\sigma(in^+) - \sigma(in^-) < 1e-5$$

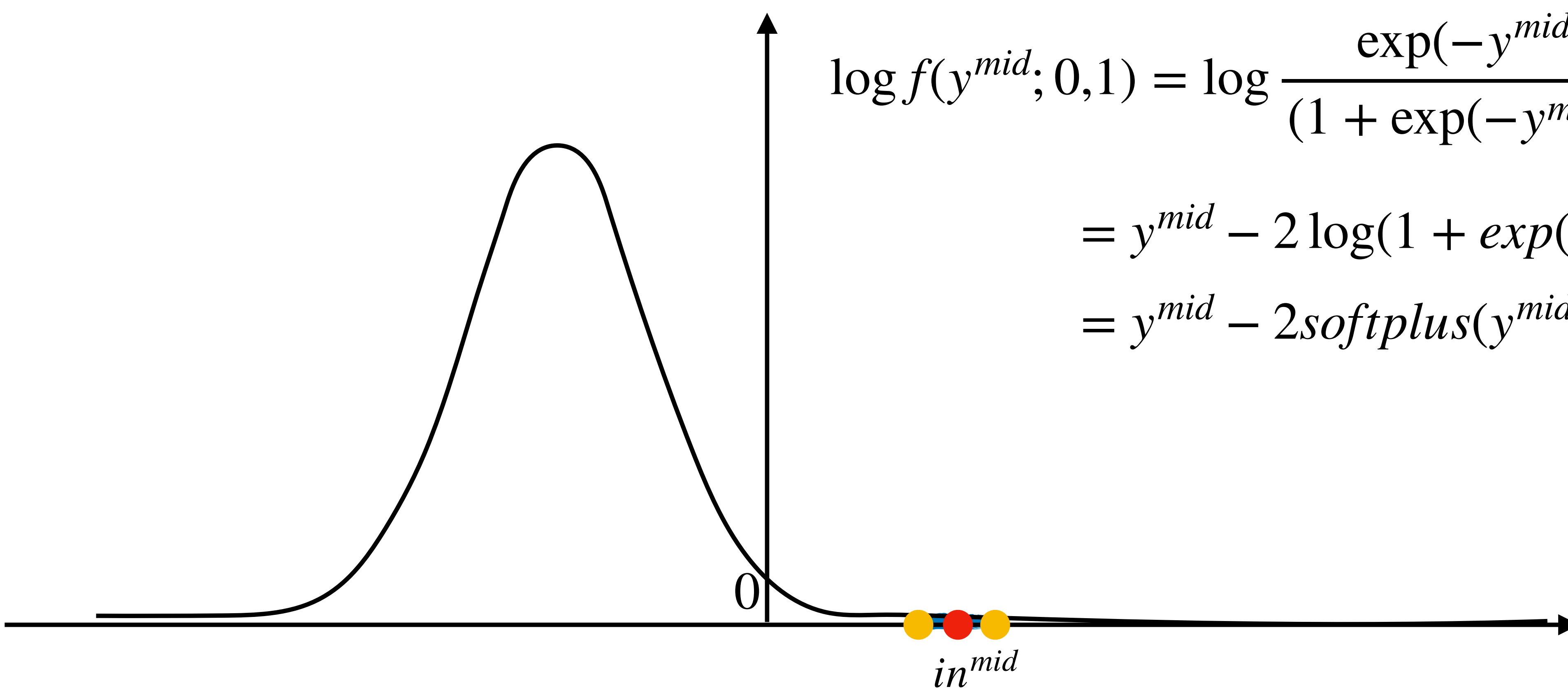
```
mid_in = inv_stdv * centered_y
# log probability in the center of the bin, to be used in extreme cases
# (not actually used in our code)
log_pdf_mid = mid_in - log_scales - 2. * F.softplus(mid_in)
```

?

$$\log f(y^{mid}; 0,1) = \log \frac{\exp(-y^{mid})}{(1+\exp(-y^{mid}))^2}$$

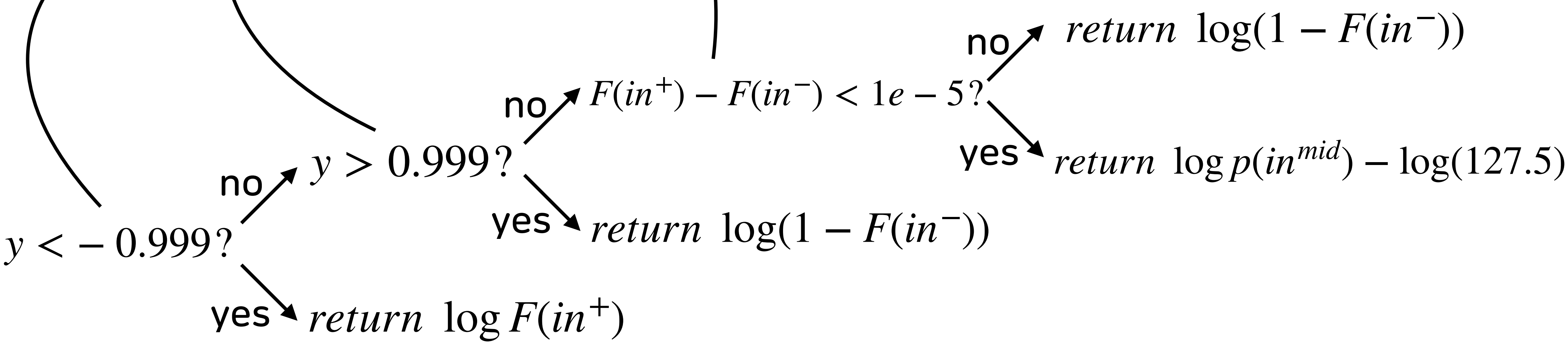
$$= y^{mid} - 2\log(1 + exp(y^{mid}))$$

$$= y^{mid} - 2softplus(y^{mid})$$



Source reference: https://github.com/r9y9/wavenet_vocoder/blob/master/wavenet_vocoder/mixture.py

```
    # tf equivalent
    """
    log_probs = tf.where(x < -0.999, log_cdf_plus,
                         tf.where(x > 0.999, log_one_minus_cdf_min,
                                  tf.where(cdf_delta > 1e-5,
                                           tf.log(tf.maximum(cdf_delta, 1e-12)),
                                           log_pdf_mid - np.log(127.5))))
    """
    # TODO: cdf_delta <= 1e-5 actually can happen. How can we choose the value
    # for num_classes=65536 case? 1e-7? not sure..
    inner_inner_cond = (cdf_delta > 1e-5).float()

    inner_inner_out = inner_inner_cond * \
        torch.log(torch.clamp(cdf_delta, min=1e-12)) + \
        (1. - inner_inner_cond) * (log_pdf_mid - np.log((num_classes - 1) / 2))
    inner_cond = (y > 0.999).float()
    inner_out = inner_cond * log_one_minus_cdf_min + (1. - inner_cond) * inner_inner_out
    cond = (y < -0.999).float()
    log_probs = cond * log_cdf_plus + (1. - cond) * inner_out
```



Source reference: https://github.com/r9y9/wavenet_vocoder/blob/master/wavenet_vocoder/mixture.py

```python
        log_probs = log_probs + F.log_softmax(logit_probs, -1)

        if reduce:
            return -torch.sum(log_sum_exp(log_probs))
        else:
            return -log_sum_exp(log_probs).unsqueeze(-1)
```

$$p(y) = \sum_{k=1}^{K} \pi_k f(y; \mu, s)$$

$$NLL = -\log p(y) = \log \sum_{k=1}^{K} \pi_k f(y; \mu, s)$$

$$= \log \sum_{k=1}^{K} \exp\{\log \pi_k + \log f(y; \mu, s)\}$$

Source reference: https://github.com/r9y9/wavenet_vocoder/blob/master/wavenet_vocoder/mixture.py