Waveglow

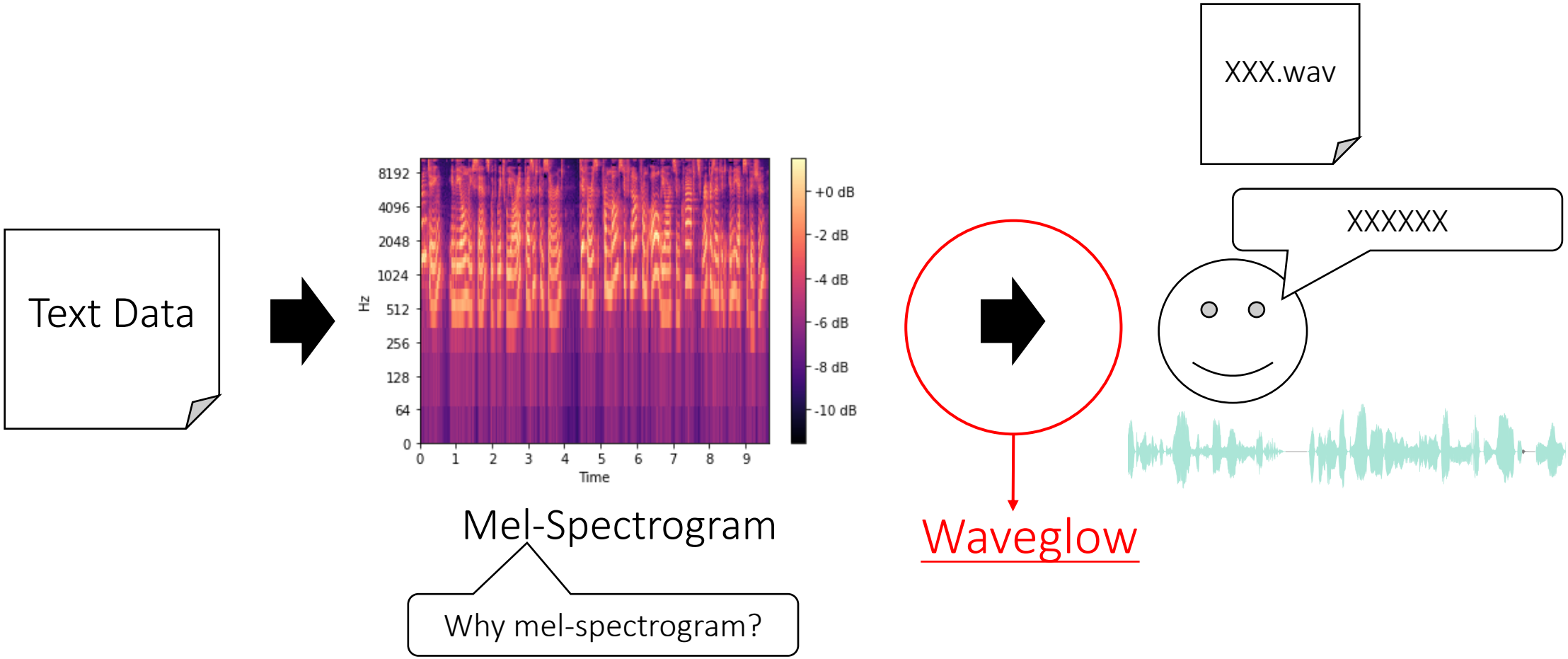
Agenda

1. Text-to-speech synthesis
2. Flow-based deep generative models
3. Waveglow
4. Demo
5. Code Analysis

1. Text-to-speech synthesis

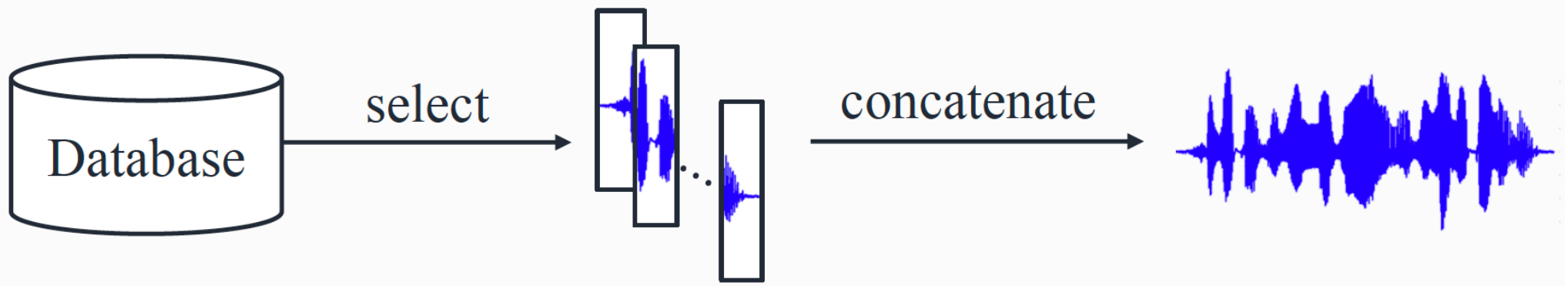
What is Waveglow?

- Waveglow is used for text-to-speech synthesis



Text-to-speech synthesis

- Concatenative TTS
  - Very large database of short speech fragments are recorded from a single speaker and then recombined to form complete utterances.



Sounds natural ☺

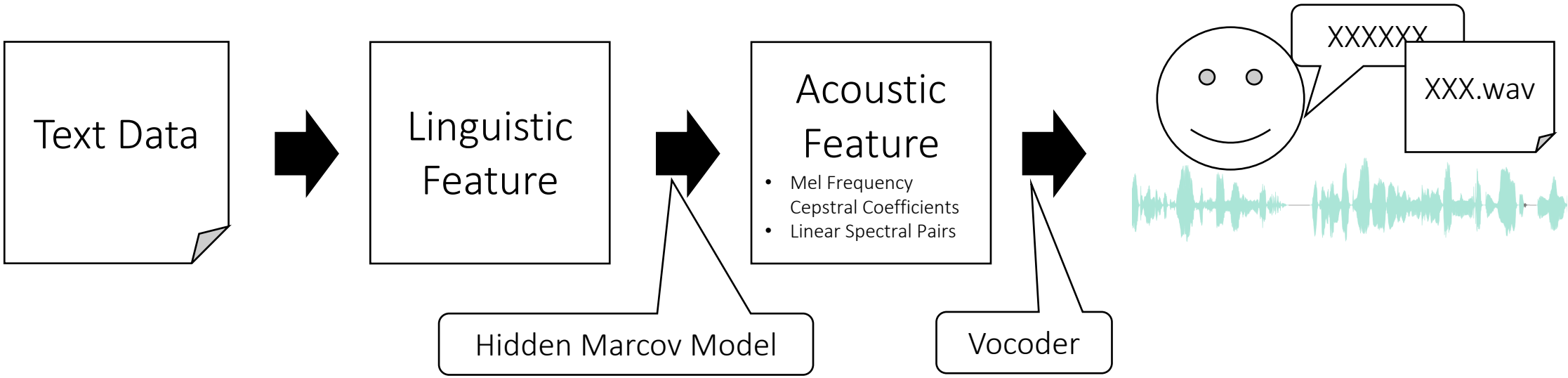
Can’t modify the voice without recording a whole database ☹

https://deepmind.com/blog/article/wavenet-generative-model-raw-audio

https://www.slideshare.net/t_koshikawa/wavenet-87105461?from_action=save

Text-to-speech synthesis

- Statistical Parametric TTS
  - Divide the problem into 3 subproblems and solve them separately.



Sounds less natural ☹

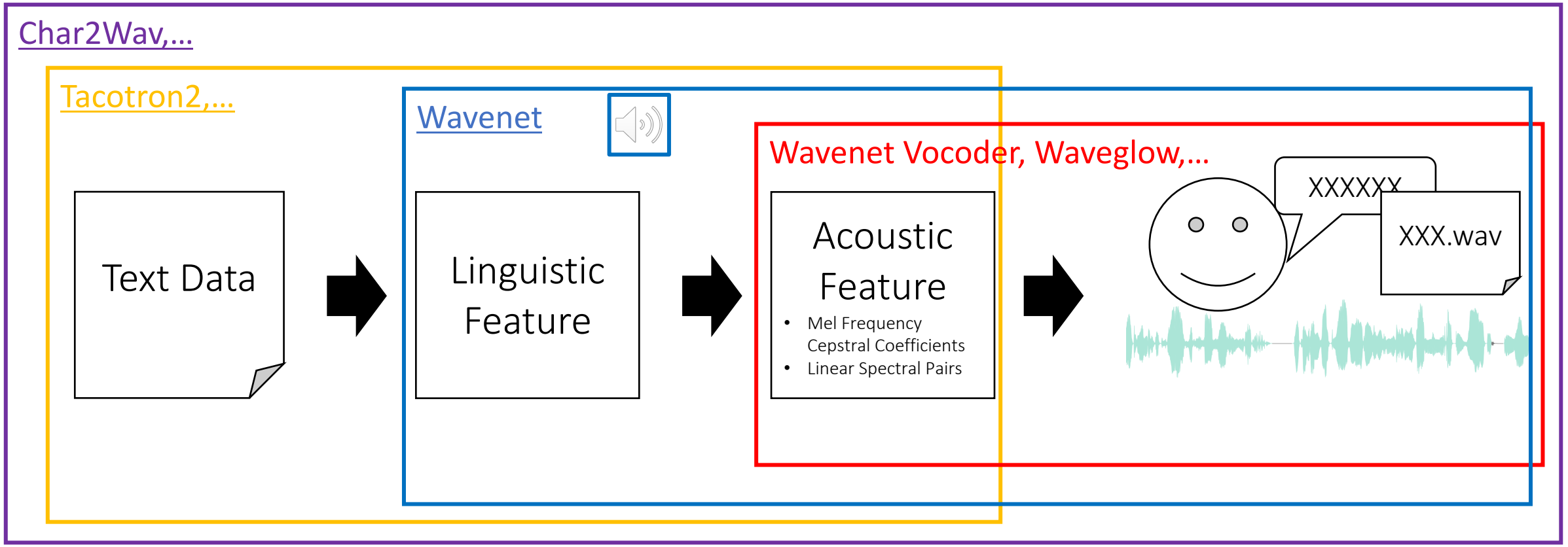
Can modify the voice without recording a whole database ☺

Zen H, Tokuda K, Black AW. Statistical parametric speech synthesis. speech communication. 2009;51(11):1039-64.

https://deepmind.com/blog/article/wavenet-generative-model-raw-audio

Text-to-speech synthesis

- DL-based TTS
  - Use DL-based models
  - Merge some of the steps



2. Flow-based generative models

What is ”Flow-based deep generative model”?

- GAN is popular as a generative model
- Flow-based model is another type of generative model
- It is often used for image generation
- Waveglow is flow-based,

similar to "Glow”

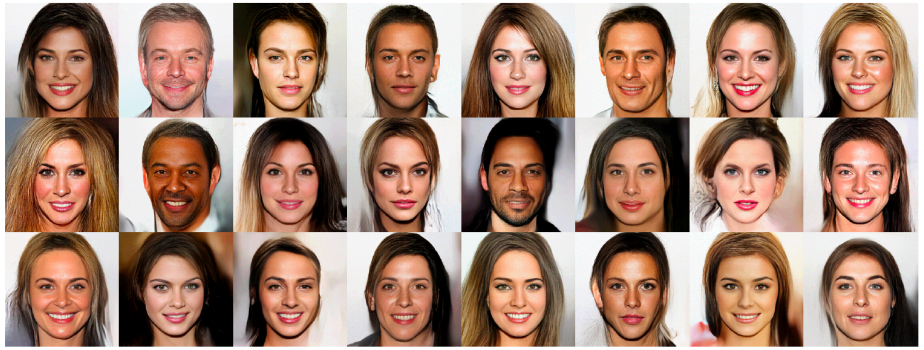


Figure 4: Random samples from the model, with temperature 0.7.

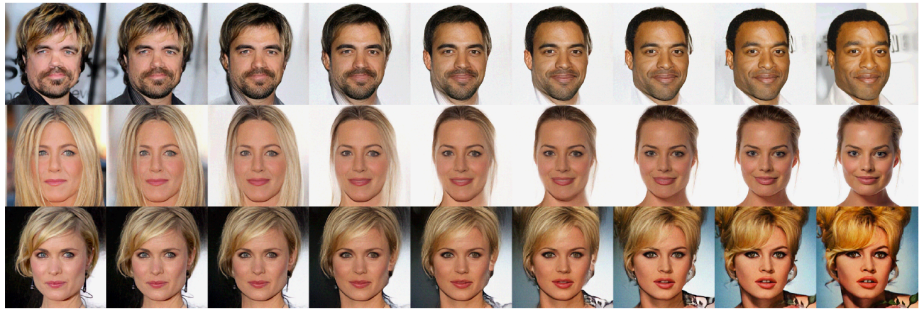
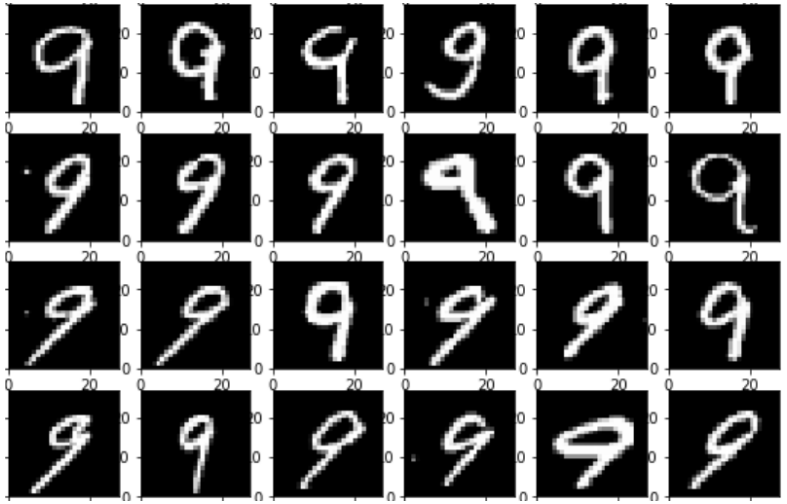


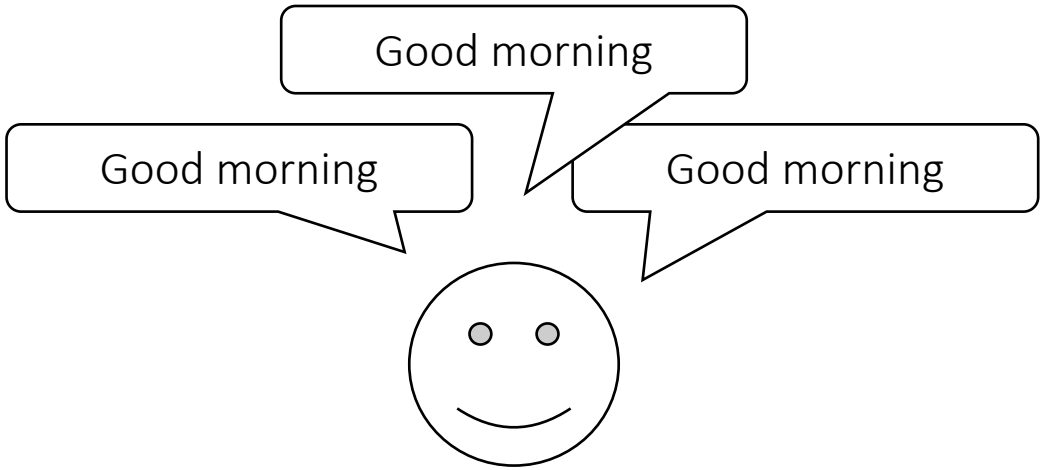
Figure 5: Linear interpolation in latent space between real images.

Kingma DP, Dhariwal P, editors. Glow: Generative flow with invertible 1x1 convolutions. Advances in Neural Information Processing Systems; 2018.

What is “Generative Model” in general?



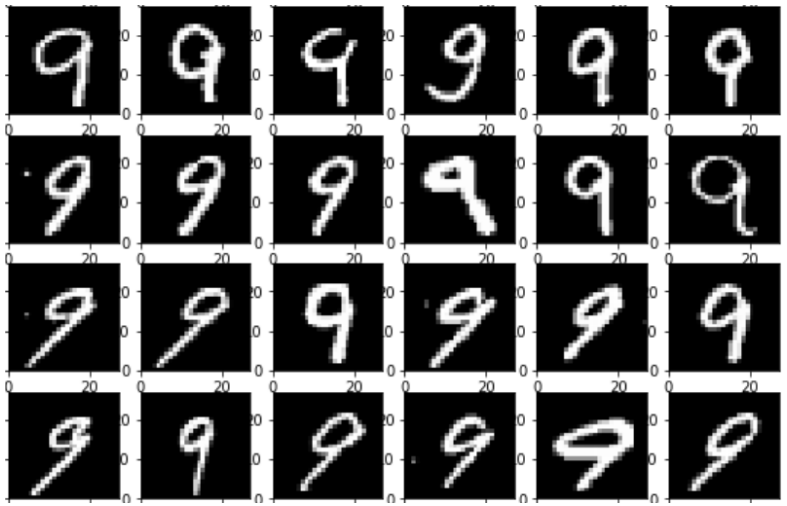
Same number, but so different…
Even when one person writes the same number multiple times, they’re not the same



Even if the same person reads the same sentence several times, they are never the same

Human speech, writing, drawing, or photo image are not deterministic…

Represent it as “Probability distribution”…

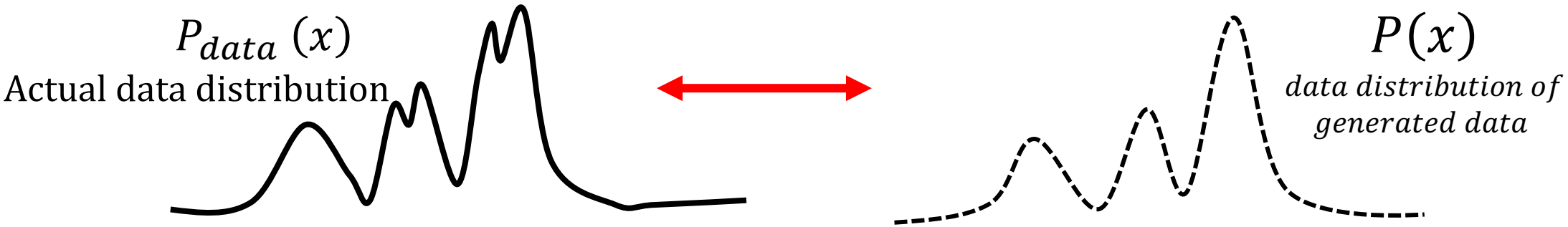


Probability to output x when θ is given

$$P(x;\ \theta)$$

$x : Output(audio, image, etc\ \ldots)$

$\theta : Parameters$



Minimizing the difference is the problem to solve

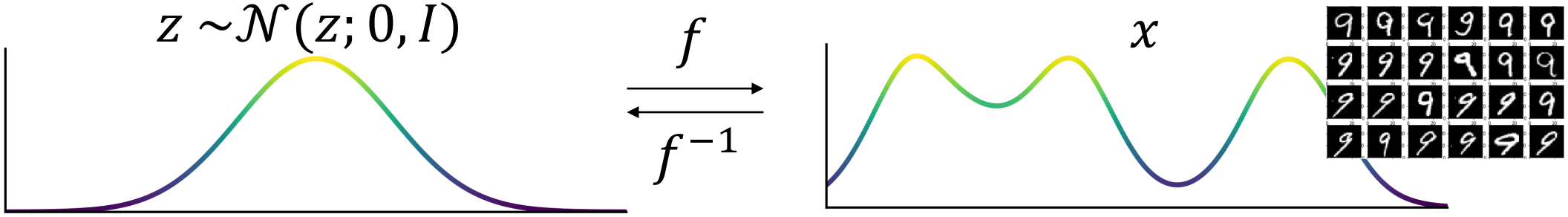
How can this be solved?

	Approach	Model Architecture	How to train / Objective Function	Computational Cost
GAN	Model a generator and train it to indirectly minimize the difference between $P(x;\ \theta)$ $P_{data}\ (x)$	Generator $x = G(z)$ $z : Latenet\ Variable$ Discriminator $D\ (x)$	Adversarial learning (Min-Max game)	Low
VAE (Variational Auto Encoder)		Encoder $z = E(x)$ $z : Latenet\ Variable$ Decoder $x = D\ (z)$	Maximize lower bound	Low
Flow-based		Flow-based invertible model $x = f_0 \cdot f_1 \cdots f_k(z)$ $z : Latenet\ Variable$	Maximize log likelihood	Low
Auto-regressive	Model probability distribution itself	Probability distribution conditioned on the past data $P(x) = \prod_i P(x_t\ ; x_1, x_2, \ldots, x_{t-1})$	Maximize log likelihood	High (parallel computation is difficult due to auto-regression)

Wavenet is an auto-regressive model, but very slow…

Basic idea of “Flow-based generative model”

- What if there is an invertible transformation f from a simple distribution z to the desired distribution of x ?



Simple distribution like gaussian distribution

Desired distribution of x (=audio, image, etc…)

Objective function

- For training, find $f^{-1}(x)$, which maximizes the probability distribution $P(x)$ with given training dataset of x (=audio, image, etc.)
- For inference, randomly pick z from the simple distribution and calculate $f(z)$

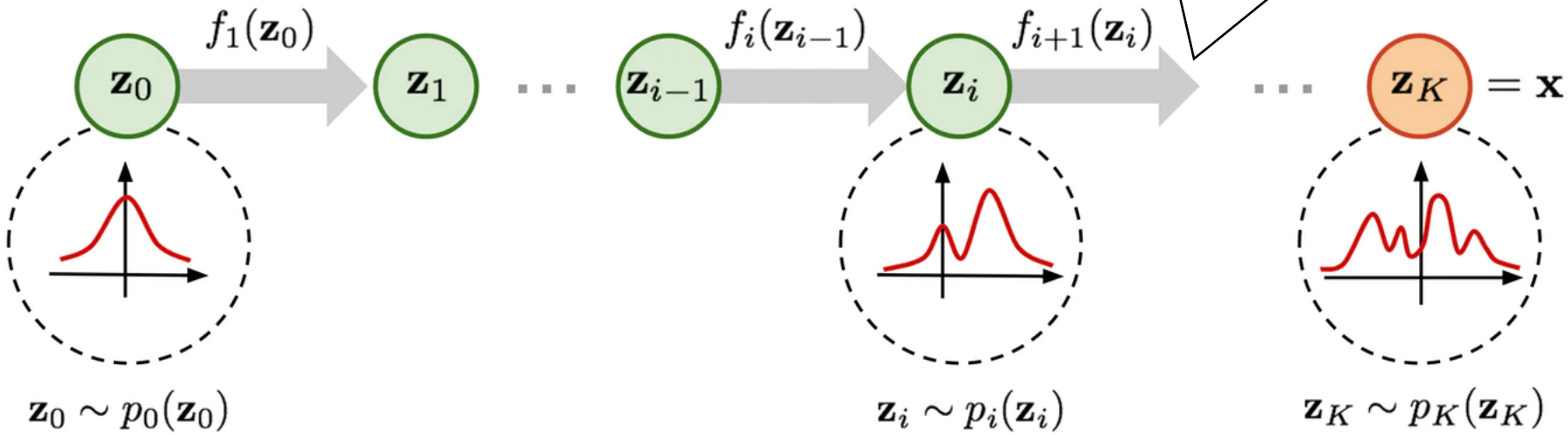
Difficult to find a single f because the desired distribution is usually complex

How can such complex f be found?

- Think about applying multiple simple functions : $f_i \in f_1, f_2, \ldots, f_K$
  - Invertible $z_i = f_i(z_{i-1})$

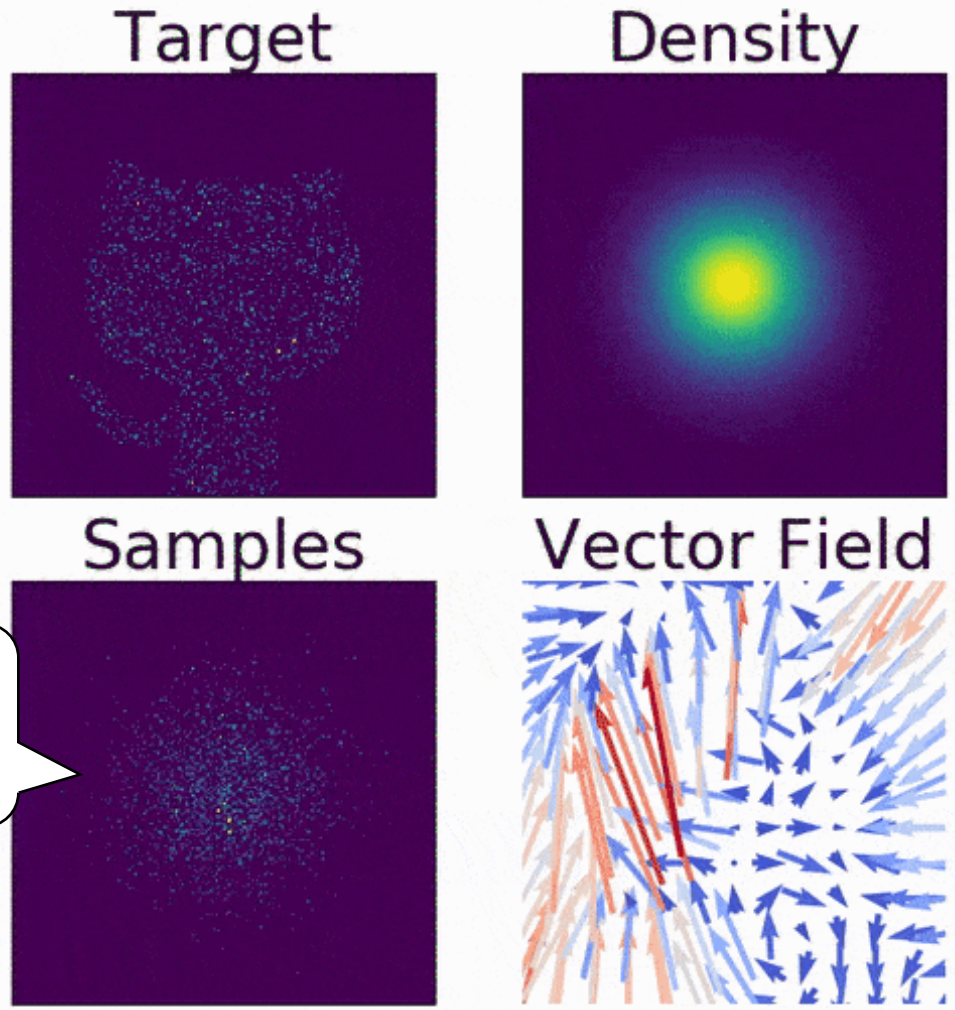
    $z_{i-1} = f_i^{-1}(z_i)$
  - $z_0, z_1, \ldots, z_k = \mathrm{x}$ have the same dimension

Each f_i is simple, but $f_1 \cdot f_2 \cdots f_K(\mathrm{z})$ can represent x with complex probability distribution



https://lilianweng.github.io/lil-log/2018/10/13/flow-based-deep-generative-models.html

2D Example of flow-based generative model



Simple 2D gaussian distribution gradually transfer to the complex distributions…

https://github.com/rtqichen/ffjord/blob/master/assets/github_flow.gif

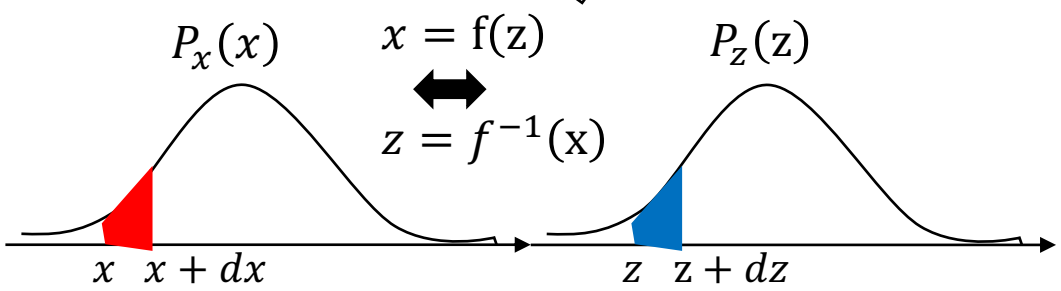
It can generate more complex, multi-dimensional data.

How can $x = f_0 \cdot f_1 \cdots f_k(z)$ maximize the likelihood $P(x)$?

We don't know the likelihood $P(x)$....

- Convert$P(x)$ to $P(z)$ using "change-of-variable" technique in the probability density function

Blue area and red area should stay the same through the change of variable



Change of variable (1D)

$$P_x(x)\mathrm{dx} = P_z(\mathrm{z})\mathrm{dz}$$

$$P_x(\mathrm{x}) = P_z(z)\left|\frac{dz}{dx}\right|$$

$$= P_z(z)\left|\frac{df^{-1}(x)}{dx}\right|$$

Change of variable (Multi-dementional)

$$P_x(\mathrm{x}) = P_z(z)\left|det\,\frac{dz}{dx}\right|$$

$$= P_z(z)\left|det\,\frac{df^{-1}(x)}{dx}\right|$$

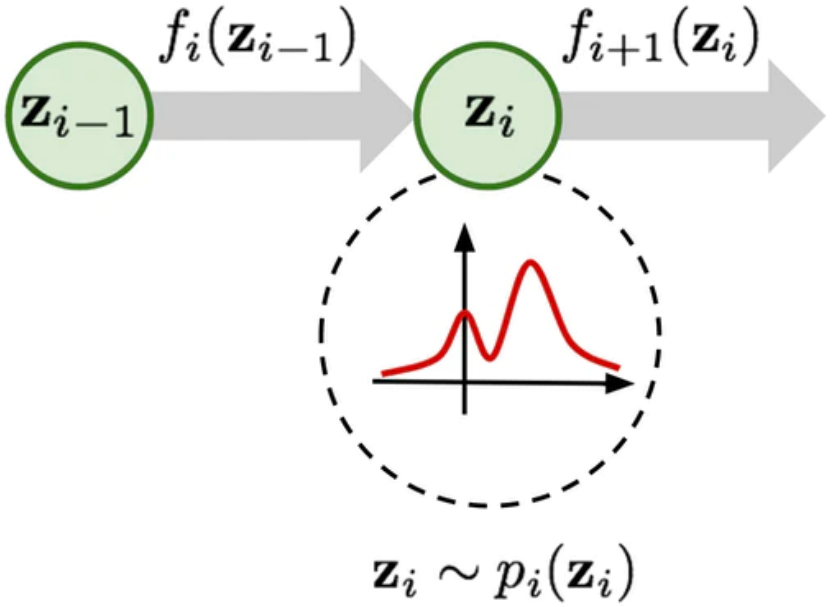
: Jacobian determinant

How can $x = f_0 \cdot f_1 \cdots f_k(z)$ maximize the likelihood $P(x)$?

- At each layer, the likelihood can be represented with the variables of the previous layer

Change of variable (Multi-dementional)

$$P_x(\mathrm{x}) = P_z(z)\left|det\,\frac{dz}{dx}\right|$$



$$P_i(z_i) = P_{i-1}(z_{i-1})\left|det\,\frac{dz_{i-1}}{dz_i}\right|$$

$$= P_{i-1}(z_{i-1})\left|\det\left(\frac{dz_i}{dz_{i-1}}\right)^{-1}\right|$$

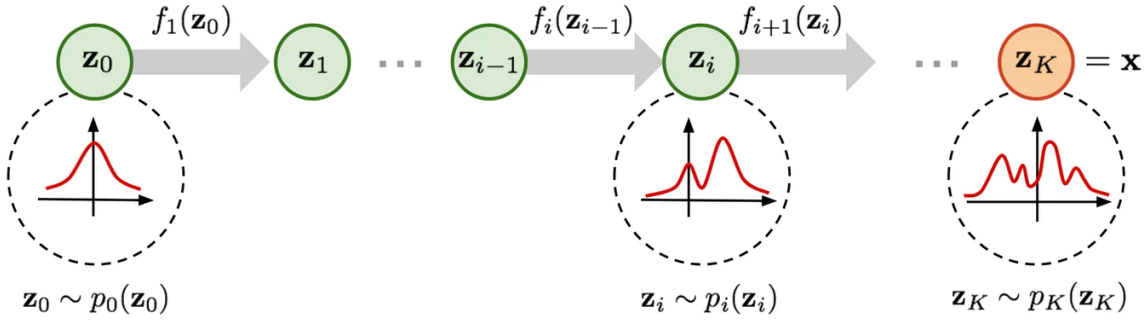
$$= P_{i-1}(z_{i-1})\left|\det\frac{df_i(z_{i-1})}{dz_{i-1}}\right|^{-1}$$

$$\det(X^{-1}) = (\det(X))^{-1}$$

How can $x = f_0 \cdot f_1 \cdots f_k(z)$ maximize the likelihood $P(x)$?

Change of variable (Multi-dementional)

$$= P_{i-1}(z_{i-1}) \left| \det \frac{df_i(z_{i-1})}{dz_{i-1}} \right|^{-1}$$



- By applying this recursively…

$$\log P_x(x) = \log P_K(z_K) = \log P_{K-1}(z_{K-1}) - \log \left| \det \frac{df_K(z_{K-1})}{dz_{K-1}} \right|$$

$$= \left(\log P_{K-2}(z_{K-2}) - \log \left| \det \frac{df_K(z_{K-2})}{dz_{K-2}} \right| \right) - \log \left| \det \frac{df_K(z_{K-1})}{dz_{K-1}} \right|$$

$$\vdots$$

$$= \log P_0(z_0) - \sum_{i=1}^{K} \log \left| \det \frac{df_i(z_{i-1})}{dz_{i-1}} \right|$$

z_0 is simple gaussian distribution, so we can calculate $P_0(z_0)$ easily!

Log Jacobian at each layer…

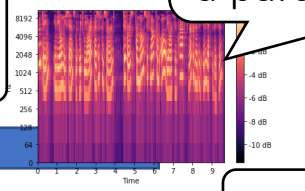
For the ease of calculation, instead of maximizing the likelihood, minimize the negative log likelihood as loss function

How does actual training & inference work?

Loss function

$$-\log P_x(x) = -\log P_0(z_0) + \sum_{i=1}^{K} \log \left| \det \frac{df_i(z_{i-1})}{dz_{i-1}} \right|$$

Condition is given as a parameter of f_i

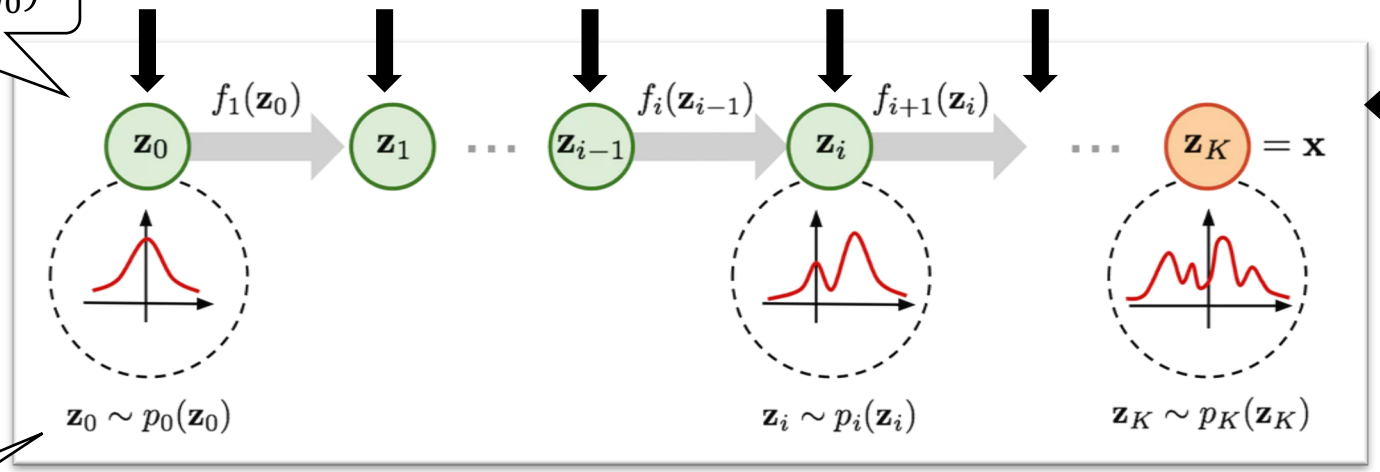


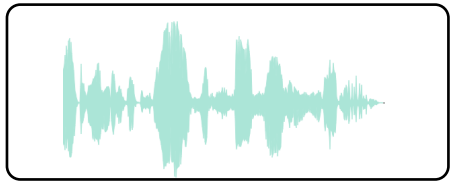
TRAINING

$z = f_0^{-1} \cdot f_1^{-1} \cdots f_K^{-1}(z)$

At every layer, apply f_i^{-1} and calculate log Jacobian

Calculate $\log P_0(z_0)$



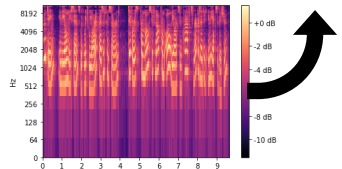


Give the original data as x

Train $f_1, f_2, \ldots f_k$ to maximize loss function

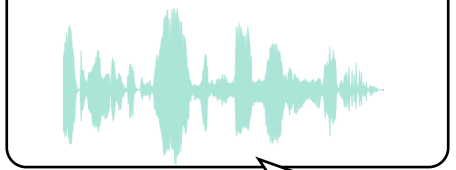
Pick z_0 randomly

INFERENCE



At every layer, apply f_i

$x = f_0 \cdot f_1 \cdots f_k(z)$



Generated!!!

Flow-based generative models

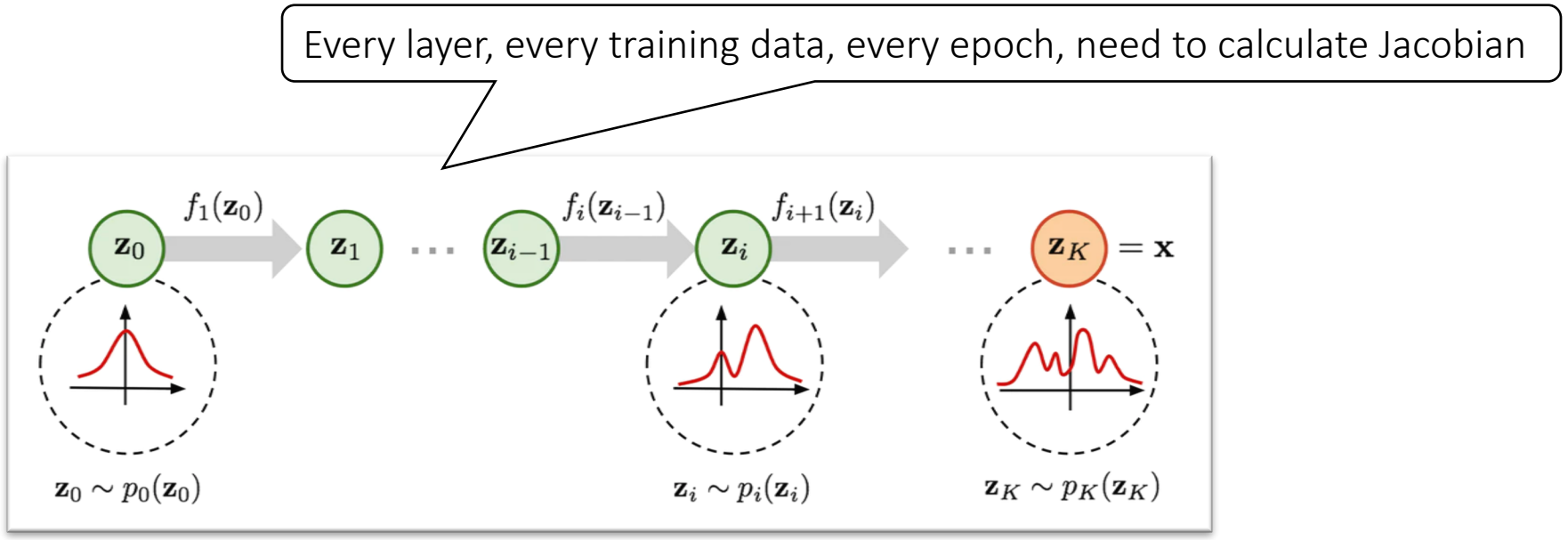
TABLE 1
List of Normalizing Flows for which we show performance results.

Architecture	Coupling function	Flow name
Coupling, 3.4.1	Affine, 3.4.4.1	RealNVP Glow
	Mixture CDF, 3.4.4.3	Flow++
	Splines, 3.4.4.4	quadratic (C) cubic RQ-NSF(C)
	Piecewise Bijective, 3.4.4.7	RAD
Autoregressive, 3.4.2	Affine	MAF
	Polynomial, 3.4.4.6	SOS
	Neural Network, 3.4.4.5	NAF UMNN
	Splines	quadratic (AR) RQ-NSF(AR)
Residual, 3.5		iResNet
		Residual flow
ODE, 3.6.1		FFJORD

Kobyzev I, Prince S, Brubaker M. Normalizing Flows: An Introduction and Review of Current Methods. IEEE transactions on pattern analysis and machine intelligence. 2020.

Computational cost of flow-based models

- Computational cost of Jacobian :$O(D^3)$ (D : dimension)



As dimension becomes higher, computation cost becomes very big ☹

How can computational cost be reduced?

- Introduce "Coupling layer"
  - Divide input x into 2 blocks $x = [x_{1:d}, x_{d+1:D}]$
  - Introduce the following transformation

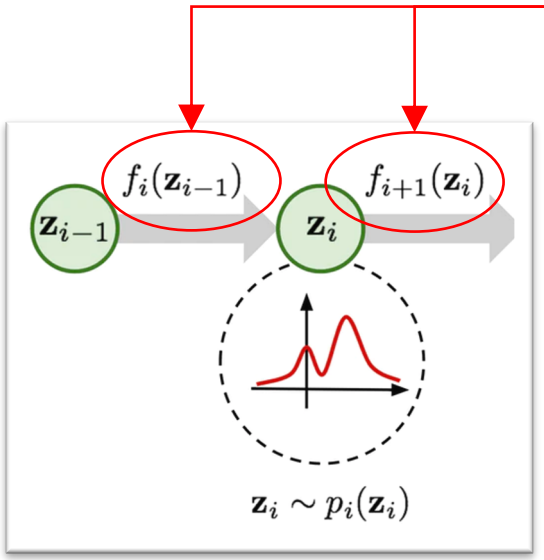
$$\begin{cases} y_1 = x_1 \\ y_2 = g(x_2, m(x_1)) \end{cases}$$

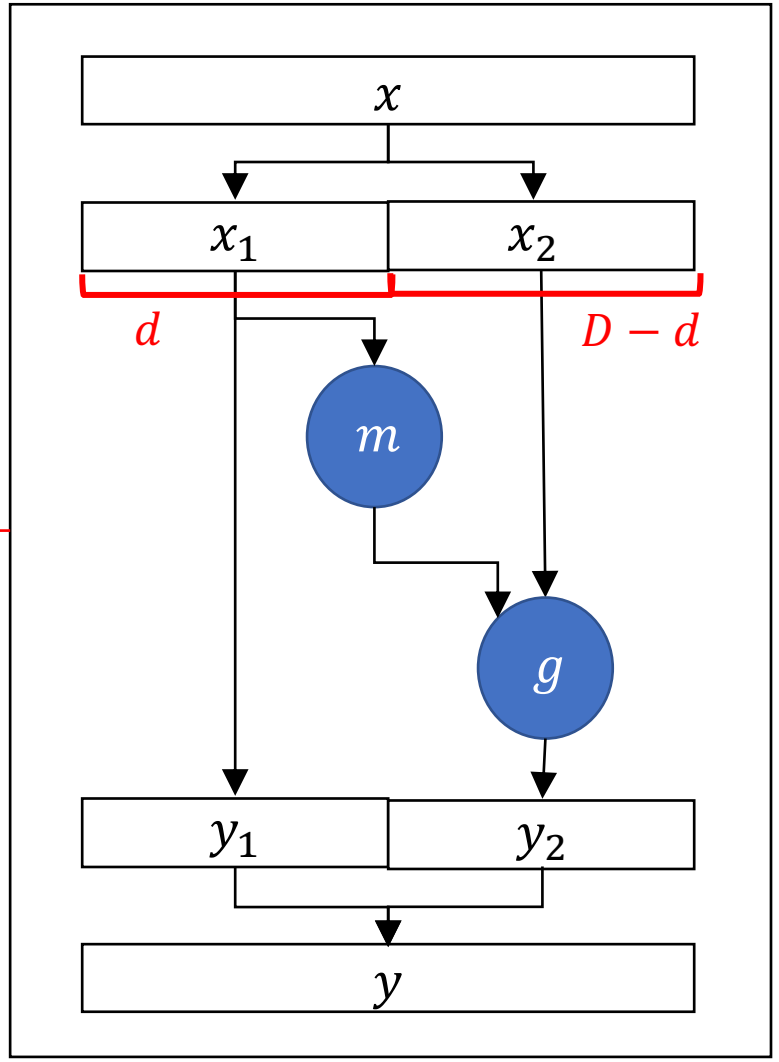
$g : Invertible\ function$

$m : Arbitrary\ function$

$$\Rightarrow \begin{cases} x_1 = y_1 \\ x_2 = g^{-1}(y_2) - m(x_1) \end{cases}$$

This layer is invertible!





Dinh L, Krueger D, Bengio Y. Nice: Non-linear independent components estimation. arXiv preprint arXiv:14108516. 2014.

How can computational cost be reduced?

$$\begin{cases} y_1 = x_1 \\ y_2 = g(x_2, m(x_1)) \end{cases} \Rightarrow \quad \frac{dy_1}{dx_1} = I_d \quad \frac{dy_1}{dx_2} = 0$$

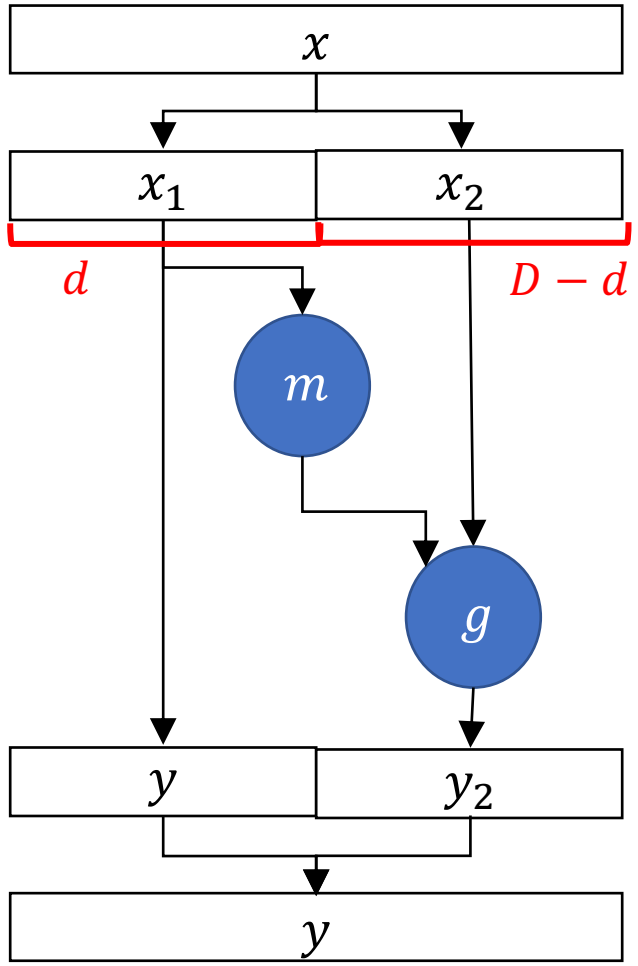
$g : Invertible\ function$

$m : Arbitrary\ function$

$$\frac{dy}{dx} = \begin{bmatrix} \frac{dy_1}{dx_1} & \frac{dy_1}{dx_2} \\ \frac{dy_2}{dx_1} & \frac{dy_2}{dx_2} \end{bmatrix} = \begin{bmatrix} I_d & 0 \\ \frac{dy_2}{dx_1} & \frac{dy_2}{dx_2} \end{bmatrix}$$

$$det\,\frac{dy}{dx} = det\,\frac{dy_2}{dx_2} = det\,\frac{dg}{dx_2}$$

Computational cost : $O(D^3) \rightarrow O((D-d)^3)$



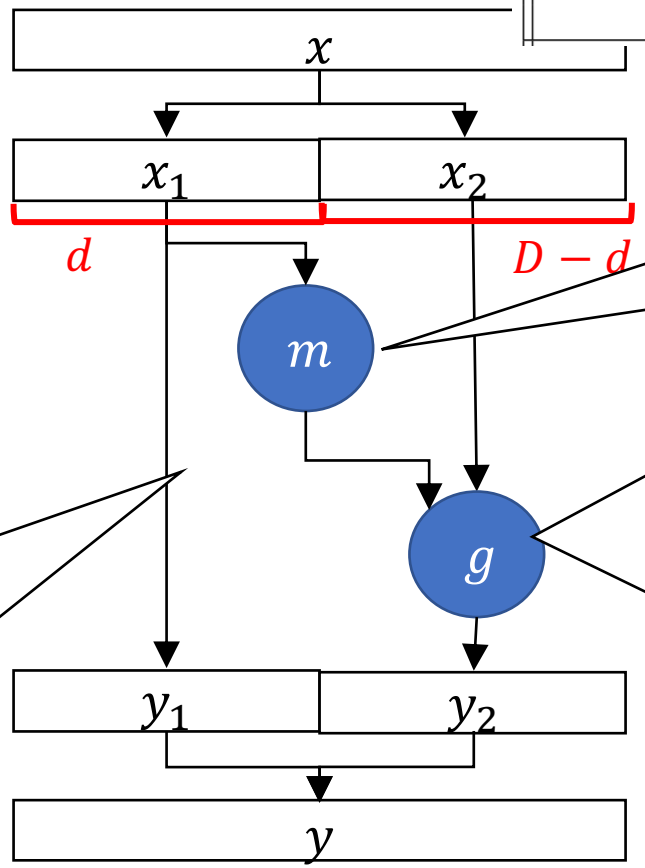
Various types of coupling flow-based models

Architecture	Coupling function	Flow name
Coupling, 3.4.1	Affine, 3.4.4.1	RealNVP Glow
	Mixture CDF, 3.4.4.3	Flow++
	Splines, 3.4.4.4	quadratic (C) cubic RQ-NSF(C)
	Piecewise Bijective, 3.4.4.7	RAD

$$\begin{cases} y_1 = x_1 \\ y_2 = g(x_2, m(x_1)) \end{cases}$$

$g : Invertible\ function$

$m : Arbitrary\ function$



This function can be designed freely.
→ Waveglow adopts wavenet-like architecture.

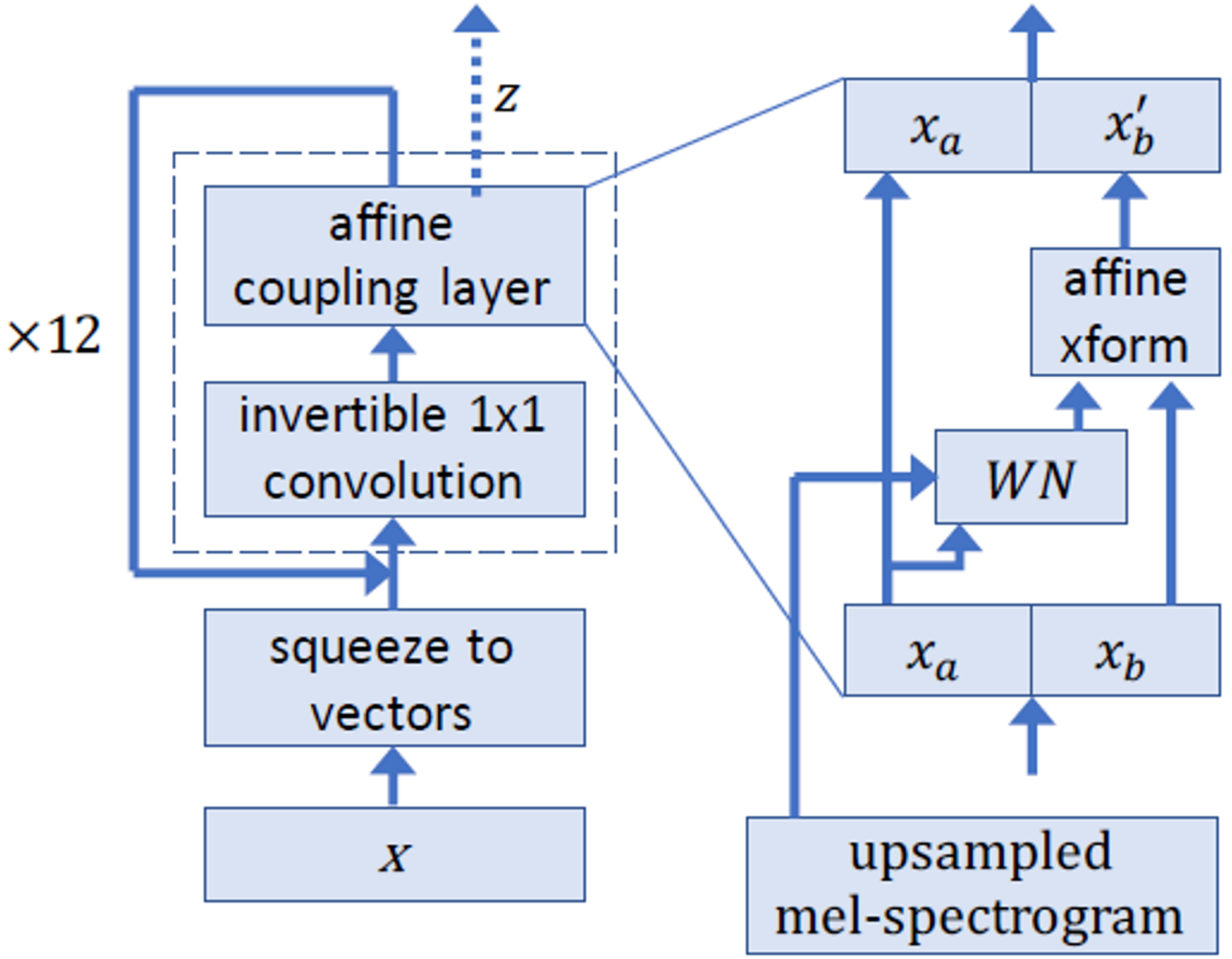
Severe restriction..
A few methods have been proposed to reorder the channels.
→ Waveglow uses 1x1 invertible convolution layer.

Challenging point when designing g is …
- Can it represent a transformation to complex distribution as whole?
- It should be invertible.
- Computational cost of $det\ \frac{dg}{dx_2}$ can be high…

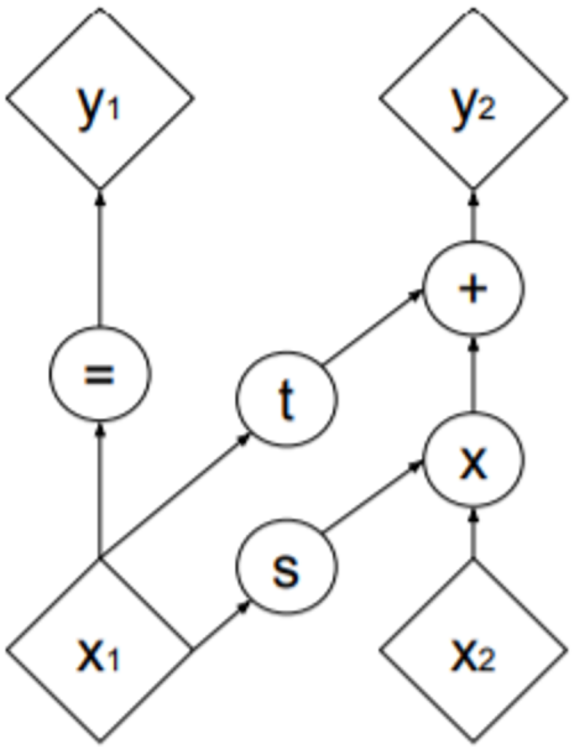
→ Waveglow uses “affine coupling layer”.

3. Waveglow

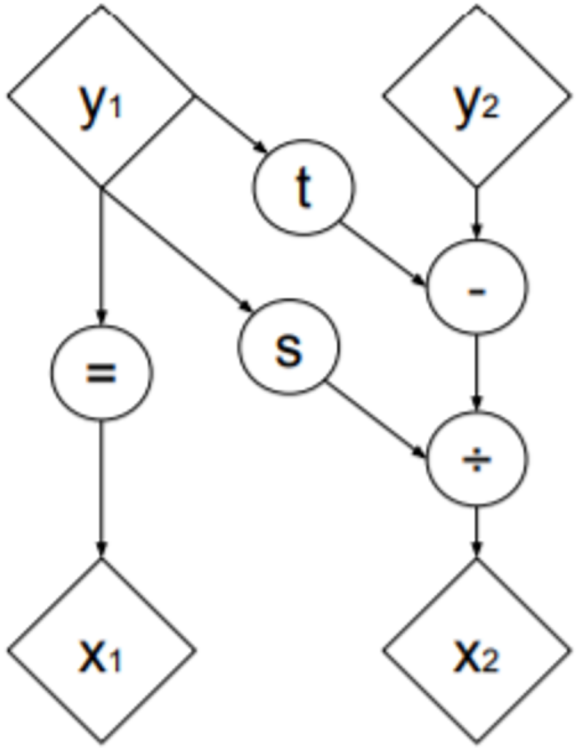
WaveGlow Network



Affine Coupling Layer



(a) Forward propagation



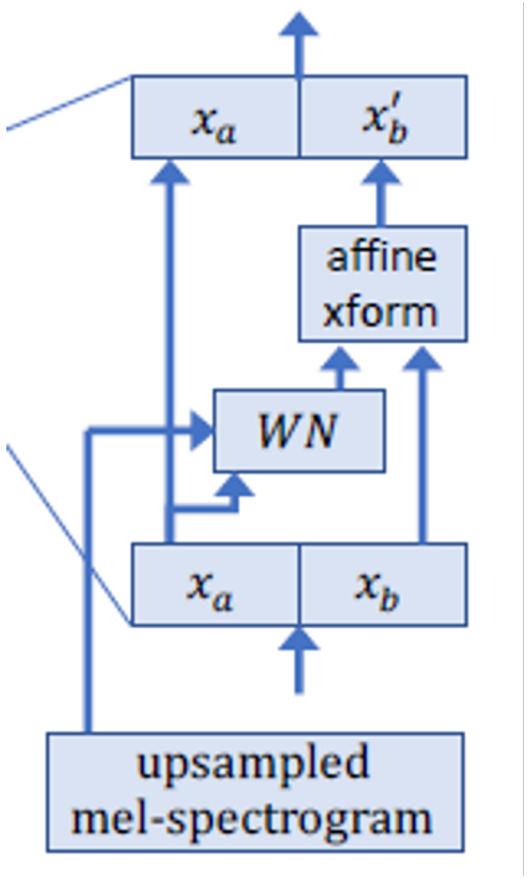
(b) Inverse propagation

A coupling layer applies a simple invertible transformation consisting of scaling followed by addition of a constant offset to one part x2 of the input vector conditioned on the remaining part of the input vector x1. Because of its simple nature, this transformation is both easily invertible and possesses a tractable determinant. However, the conditional nature of this transformation, captured by the functions ‘s’ and ‘t’, significantly increase the flexibility of this otherwise weak function. The forward and inverse propagation operations have identical computational cost.

WaveGlow Network: Affine Coupling layer

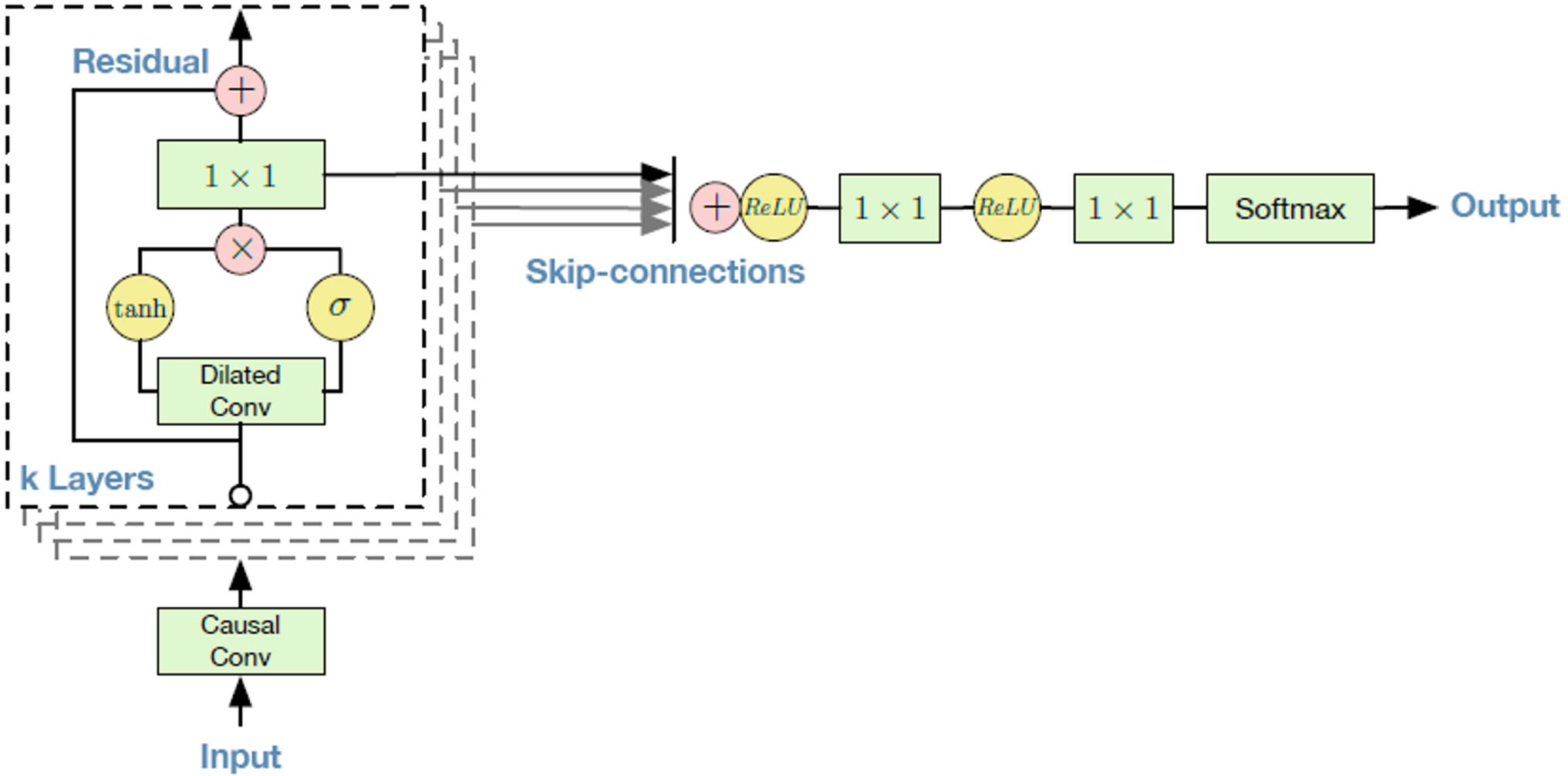
- Half of the channels serve as inputs, which then produce multiplicative and additive terms that are used to scale and translate the remaining channels:

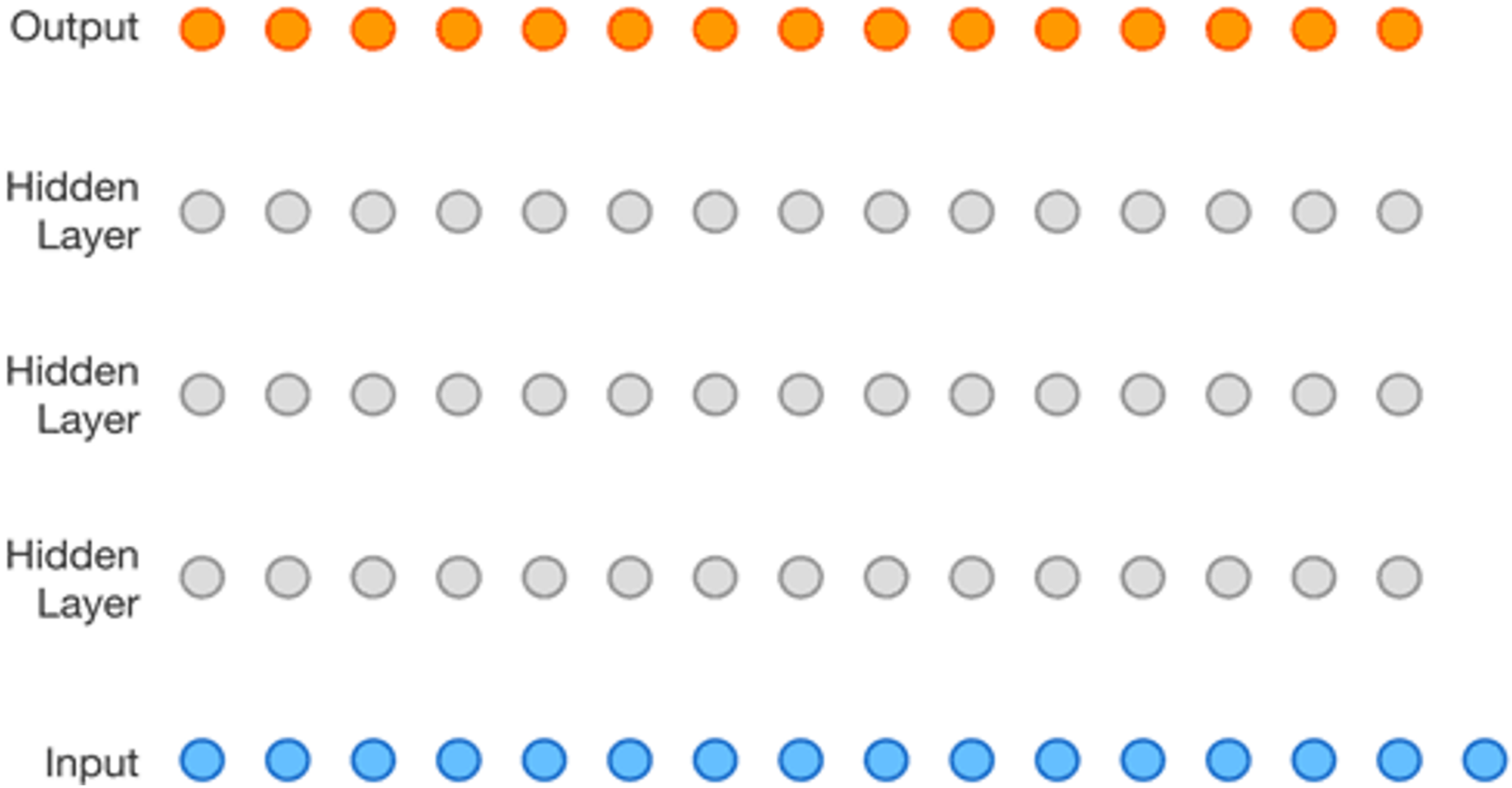
$$\boldsymbol{x}_a, \boldsymbol{x}_b = split(\boldsymbol{x})$$
$$(\log \boldsymbol{s}, \boldsymbol{t}) = WN(\boldsymbol{x}_a, mel\text{-}spectrogram)$$
$$\boldsymbol{x}_b\prime = \boldsymbol{s} \odot \boldsymbol{x}_b + \boldsymbol{t}$$
$$\boldsymbol{f}^{-1}_{coupling}(\boldsymbol{x}) = concat(\boldsymbol{x}_a, \boldsymbol{x}_b\prime)$$



- the channels are used as the inputs to WN(), in this case x_a, are passed through unchanged to the output of the layer.
- Accordingly, when inverting the network, we can compute s and t from the output x_a, and then invert xb' to compute xb, by simply recomputing WN(x_a; mel-spectrogram). In our case, WN() uses layers of dilated convolutions with gated-tanh nonlinearities, as well as residual connections and skip connections. This WN architecture is similar to WaveNet. The difference is that Wavenet uses causal convolutions while WaveGlow does not. In Causal Convolutions, the filter output doesn't depend on future inputs.
- The affine coupling layer is also where we include the mel-spectrogram in order to condition the generated result on the input.

WaveNet Architecture (WN Function)



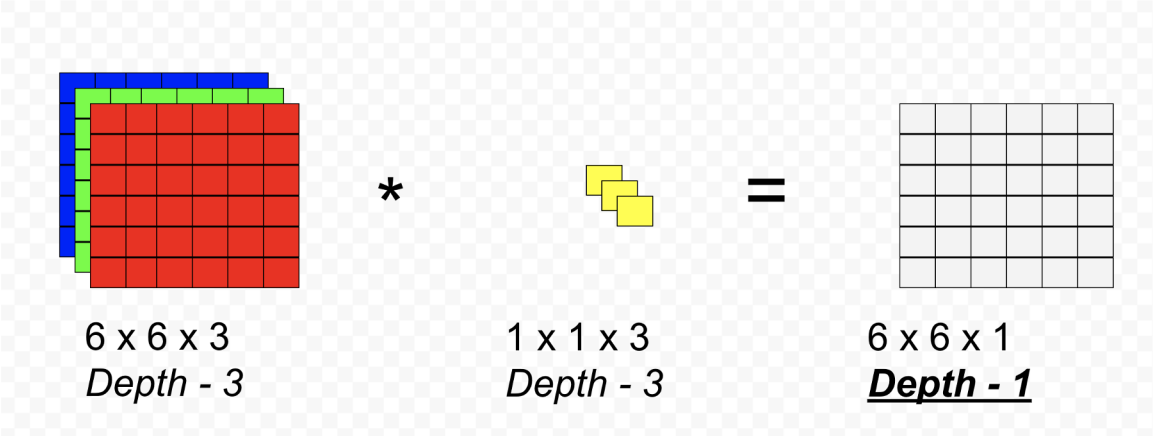


This is a visualization of the dilated causal convolutional layers

1X1 Invertible Convolution

- In the affine coupling layer, channels in the same half never directly modify one another. Without mixing information across channels, this would be a severe restriction.
- So by adding a 1X1 Convolutional layer before the affine coupling layer, information is mixed across channels.The log-determinant of the Jacobian of this transformation joins the loss function due to the change of variables

$$f_{conv}^{-1} = \boldsymbol{W}\boldsymbol{x}$$



Log Likelihood

- The log likelihood function was defined as below, with the first term representing the likelihood for the spherical gaussian distribution and the second term from change of variables.

$$\log p_\theta(\boldsymbol{x}) = \log p_\theta(\boldsymbol{z}) + \sum_{i=1}^{k} \log \left| \det(\boldsymbol{J}(\boldsymbol{f}_i^{-1}(\boldsymbol{x}))) \right|$$

- Now, the second we get from the coupling and convolutional layers.

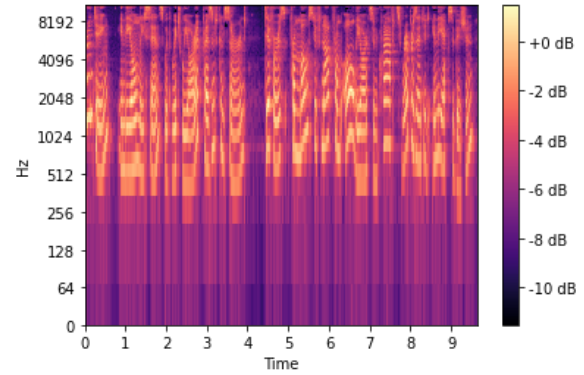
$$\log p_\theta(x) = -\frac{\boldsymbol{z}(\boldsymbol{x})^T \boldsymbol{z}(\boldsymbol{x})}{2\sigma^2} + \sum_{j=0}^{\#coupling} \log \boldsymbol{s}_j(x, \textit{mel-spectrogram}) + \sum_{k=0}^{\#conv} \log \det |\boldsymbol{W}_k|$$

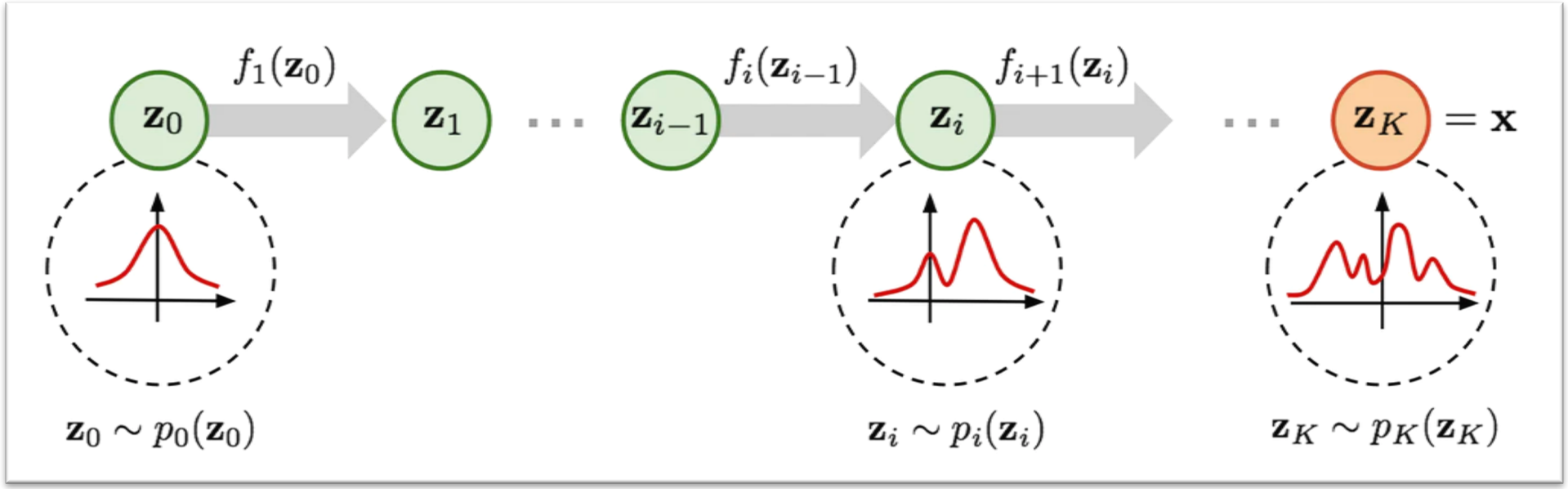
4. Demo

Recap : Waveglow

Demo

- Waveglow has 12 functions as a default.
- How does the output changes through the flow?





5. Code Analysis

https://github.com/asadayuki/waveglow/blob/master/doc/CodeAnalysis.pdf