# MoGlow: Probabilistic and controllable motion synthesis using normalising flows (2019)

Gustav Eje Henter*  Simon Alexanderson*  Jonas Beskow

Division of Speech, Music and Hearing, KTH Royal Institute of Technology, Stockholm, Sweden

# Abstract

- Prediction models suffer from regression to mean pose (“mean collapse”) and artefacts like foot sliding.
- Normalising Flow (Glow) to map pose to latent space
- Sample a natural motion given a condition (e.g., walk along a trajectory)

- LSTM to generate motion sequence of arbitrary length and ensure tempor al consistency
- Autoregression to ensure temporal consistency

# Background: Normalising Flow

- Is a generative model like VAE, GAN etc.
- Estimates data distribution $P_{\mathrm{data}}(\mathbf{x})$
- Allows exact probability computation unlike VAEs
- Easier to train than VAEs (may suffer “posterior collapse”, i.e., strong decoders yield models where z has little impact), and GANs

# Background: Normalising Flow – Maximum Likelihood Estimation

**Assumptions and definitions:**

- probability of observing data point **x** (unknown): $P_{\text{data}}\left(\mathbf{x}\right)$
- samples drawn from $p_{\text{data}}$ are independent (like images, CIFAR, MNIST)

$$P_{\text{data}}\left(\mathbf{X}\right) = P_{\text{data}}\left(\mathbf{x}_1\right) P_{\text{data}}\left(\mathbf{x}_1\right) \ldots P_{\text{data}}\left(\mathbf{x}_N\right) = \prod_i P_{\text{data}}\left(\mathbf{x}_i\right)$$

- we want to create a model from which we can sample: $P_{\text{model}}\left(\mathbf{x}\right)$
- in order to build as much "realistic" model as we can we want to have: $P_{\text{model}}\left(\mathbf{x}\right) = P_{\text{data}}\left(\mathbf{x}\right)$
- this allows us to define target metric which tells us how far from true distribution we are:

$$D\left(P_{\text{data}}\left(\mathbf{x}\right), P_{\text{model}}\left(\mathbf{x}\right)\right) = 0 \ \text{ if } \ \mathrm{P}_{\text{model}}\left(\mathbf{x}\right) = \mathrm{P}_{\text{data}}\left(\mathbf{x}\right)$$

Source: https://github.com/kmkolasinski/deep-learning-notes/tree/master/seminars

**Assumptions and definitions:**

- The **natural choice** for **D** is Kullback Leibler divergence *(this is a critical point for all derivations below)*:

$$\mathrm{KL}\left(P_{\mathrm{data}}\left(\mathbf{x}\right), P_{\mathrm{model}}\left(\mathbf{x}\right)\right) = \sum_i P_{\mathrm{data}}\left(\mathbf{x}\right)\log\left(\frac{P_{\mathrm{data}}\left(\mathbf{x}\right)}{P_{\mathrm{model}}\left(\mathbf{x}\right)}\right)$$

- Our generator $P_{\mathrm{model}}$ will be defined by some neural network, let's put explicitly its dependence on some parameters (neural network weights)

$$P_{\mathrm{model}}\left(\mathbf{x}\right) = P_{\mathrm{model}}\left(\mathbf{x};\theta\right)$$

- We want to minimize KL w.r.t model parameters, hence we need to compute the gradients

$$\nabla_\theta \mathrm{KL} = \nabla_\theta \sum_i P_{\mathrm{data}}\left(\mathbf{x}\right)\log\left(\frac{P_{\mathrm{data}}\left(\mathbf{x}\right)}{P_{\mathrm{model}}\left(\mathbf{x};\theta\right)}\right) = -\nabla_\theta \sum_i P_{\mathrm{data}}\left(\mathbf{x}\right)\log\left(P_{\mathrm{model}}\left(\mathbf{x};\theta\right)\right)$$

- The samples are explicitly sampled from data distribution (N goes to infinity):

$$\nabla_\theta \mathrm{KL} = -\frac{1}{N}\nabla_\theta \sum_{\mathbf{x}_i \sim P_{\mathrm{data}}(\mathbf{x})} \log\left(P_{\mathrm{model}}\left(\mathbf{x}_i;\theta\right)\right) = -\frac{1}{N}\nabla_\theta \log \prod_{\mathbf{x}_i \sim P_{\mathrm{data}}(\mathbf{x})} P_{\mathrm{model}}\left(\mathbf{x}_i;\theta\right)$$

# Background: Normalising Flow – Maximum Likelihood Estimation

- The samples are explicitly sampled from data distribution (N goes to infinity):

$$\nabla_\theta \mathrm{KL} = -\frac{1}{N}\nabla_\theta \sum_{\mathbf{x}_i \sim P_{\text{data}}(\mathbf{x})} \log\left(P_{\text{model}}\left(\mathbf{x}_i;\theta\right)\right) = -\frac{1}{N}\nabla_\theta \log \prod_{\mathbf{x}_i \sim P_{\text{data}}(\mathbf{x})} P_{\text{model}}\left(\mathbf{x}_i;\theta\right)$$

- Recall assumption that: $P_{\text{data}}(\mathbf{X}) = P_{\text{data}}(\mathbf{x}_1)\, P_{\text{data}}(\mathbf{x}_1) \ldots P_{\text{data}}(\mathbf{x}_N) = \prod_i P_{\text{data}}(\mathbf{x}_i)$

- This allows us to write similar expression for model (but X are drawn from "real" data)

$$\nabla_\theta \mathrm{KL} = -\frac{1}{N}\nabla_\theta \log P_{\text{model}}\left(\mathbf{X};\theta\right)$$

- Hence by minimizing **KL divergence** we **maximize log-likelihood** of observed data, both can be used to obtain same result. However with one it maybe do in easier way.

minimize $\mathrm{KL}$ maximize $\log P_{\text{model}}\left(\mathbf{X};\theta\right)$

Ignoring the gradient operator, want to minimise KL == minimise model likelihood (log is preferred due to numerical stability)

# Background: Normalising Flow

- To estimate $P_{model}$, we start from a "simple" latent distribution (charactersied by latent random variable **Z**), and find a series of invertible functions that will "recover" $P_{model}$ (characterized by latent RV **X**), i.e. (we choose a latent distribution of uniform Gaussian in this example):

$$\boldsymbol{Z} \sim \mathcal{N}(\boldsymbol{z}; \boldsymbol{0}, \boldsymbol{I})$$

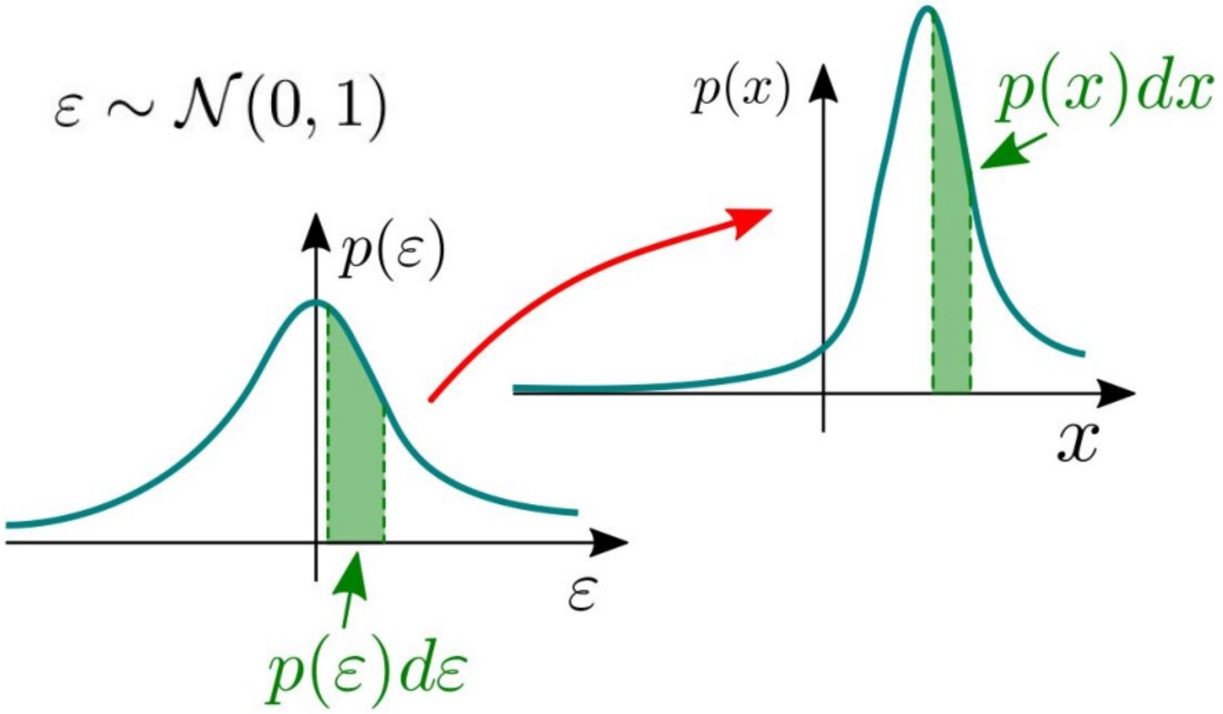
$$\boldsymbol{x} = \boldsymbol{f}(\boldsymbol{z}) = \boldsymbol{f}_1(\boldsymbol{f}_2(\ldots \boldsymbol{f}_N(\boldsymbol{z})))$$

$$\boldsymbol{z} = \boldsymbol{z}_N \xrightarrow{\boldsymbol{f}_N} \boldsymbol{z}_{N-1} \xrightarrow{\boldsymbol{f}_{N-1}} \ldots \xrightarrow{\boldsymbol{f}_2} \boldsymbol{z}_1 \xrightarrow{f_1} \boldsymbol{z}_0 = \boldsymbol{x}$$

$$\boldsymbol{z}_n(\boldsymbol{x}) = \boldsymbol{f}_n^{-1} \circ \ldots \circ \boldsymbol{f}_1^{-1}(\boldsymbol{x}).$$

# Background: Normalising Flow

- When we transform from $\mathbf{z}_i$ to $\mathbf{z}_{i+1}$ (assume $\mathbf{z}_0 = \mathbf{x}$ and $\mathbf{z}_n = \mathbf{z}$), we want the densities preserved. 1D example:




$$p(\varepsilon)d\varepsilon = p(x)dx$$

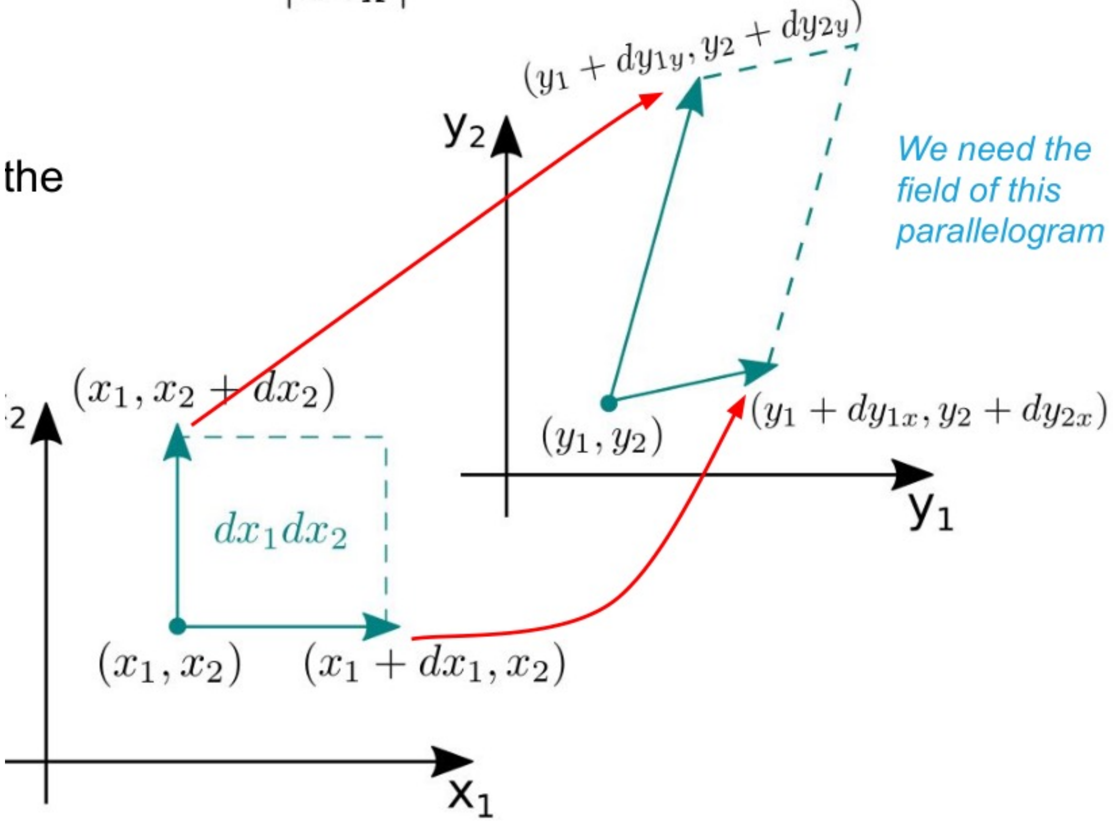
$$p(x) = p(\varepsilon)\left|\frac{dx}{d\varepsilon}\right|^{-1}$$

# Background: Normalising Flow

- For dim > 1 we want volume preservation
- 2D case:

$$p(\mathbf{y}) = p(\mathbf{x}) \left| \frac{dV_{\mathbf{y}}}{dV_{\mathbf{x}}} \right|^{-1}$$

the




We derive:

$$p(\mathbf{y}) = p(\mathbf{x}) \left| \det \mathbf{J} \right|^{-1}$$

# Background: Normalising Flow

- General case for 1 step of transformation:

$$q'(\mathbf{z}') = q(\mathbf{z}) \left| \det \frac{\partial f\left(\mathbf{z}^{(n)}\right)}{\partial \mathbf{z}^{(n)}} \right|^{-1}$$

- Through chaining:

$$\begin{aligned} \mathbf{z}^{(0)} &\sim p(\mathbf{z}) \\ \mathbf{z}^{(1)} &= f^{(1)}(\mathbf{z}^{(0)}) \\ \mathbf{z}^{(2)} &= f^{(2)}(\mathbf{z}^{(1)}) \\ &\cdots \\ \mathbf{z}^{(n)} &= f^{(n)}(\mathbf{z}^{(n-1)}) \end{aligned} \quad \longrightarrow \quad q^{(n)}(\mathbf{z}^{(n)}) = q(\mathbf{z}^{(0)}) \prod_{i=1}^{n} \left| \det \frac{\partial \mathbf{z}^{(i)}}{\partial \mathbf{z}^{(i-1)}} \right|^{-1}$$

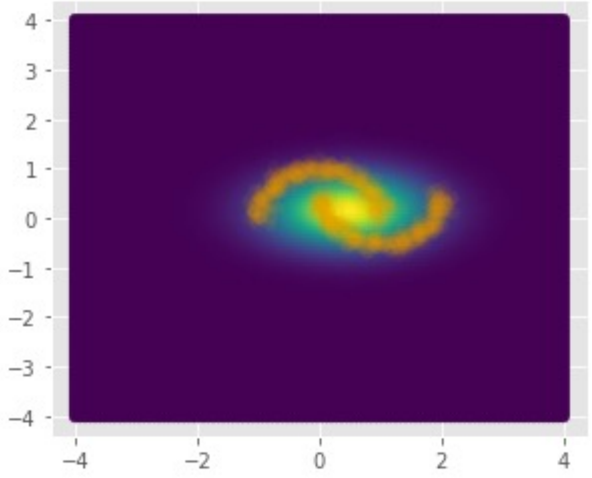
- Using log probability (or log likelihood if summed over minibatch):

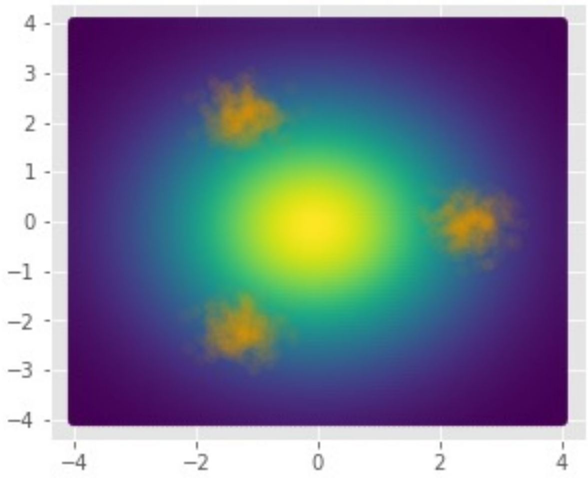
$$\ln p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \ln p_{\mathcal{N}}(\boldsymbol{z}_N(\boldsymbol{x})) + \sum_{n=1}^{N} \ln \left| \det \frac{\partial \boldsymbol{z}_n(\boldsymbol{x})}{\partial \boldsymbol{z}_{n-1}} \right|$$

- For the choice of invertible function $\mathbf{f}_i$, we may use a simple affine transfor mation e.g.,

$$x = \mu + \varepsilon\sigma$$
$$\varepsilon = (x - \mu)/\sigma$$

- But then it struggles to fit (i.e., transform from the unit Gaussian to) a com plex, nonlinear data distribution:



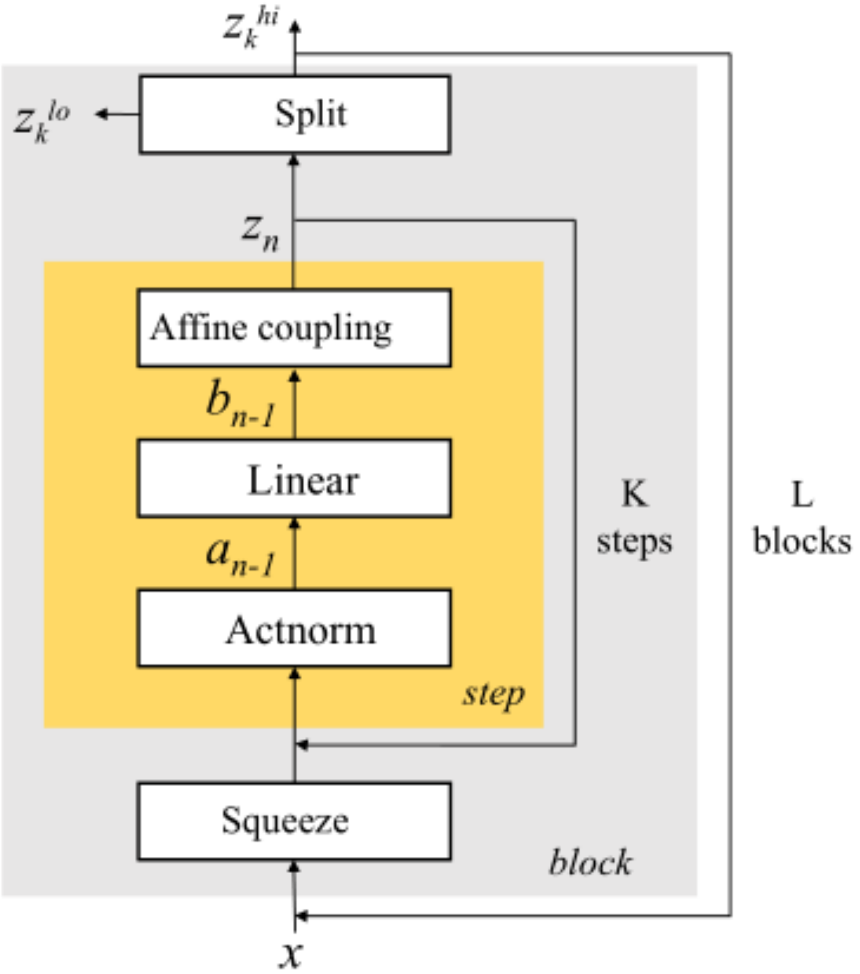




Examples from: https://github.com/AxelNathanson/pytorch-normalizing-flows/blob/main/Normalizing_Flows_with_Pytorch.ipynb

# Background: Normalising Flow: Choice of Invertible Functions

- Several good candidates proposed: MADE, RealNVP, Glow
- Glow is used in this project.

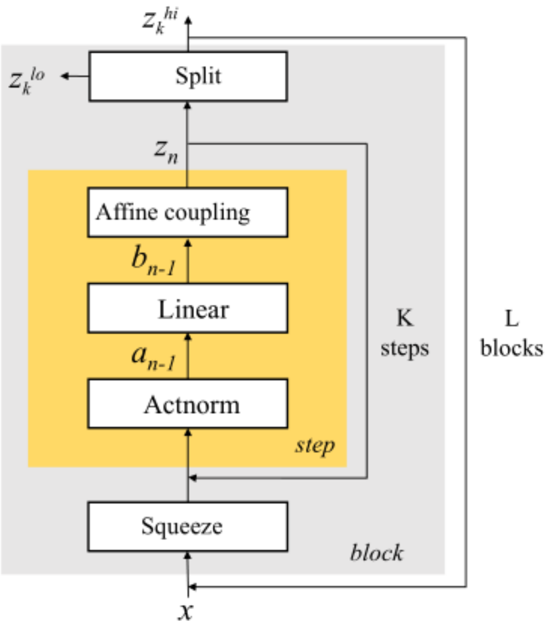
- One glow step (inverse of $f_i$)

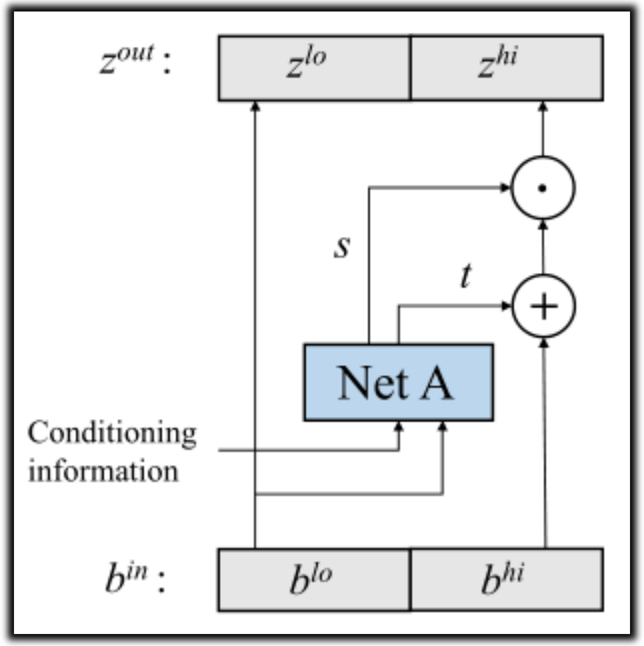



**Actnorm**: $\boldsymbol{a}_{t,n} = \boldsymbol{s}_n \odot \boldsymbol{z}_{t,n} + \boldsymbol{t}_n$
Kind of like a reversible batchnorm

**Linear transformation**: $\boldsymbol{b}_{t,n} = \boldsymbol{W}_n \boldsymbol{a}_{t,n}$
Weight matrix is LU-decomposed where one of the diagonals is constrained to contain just ones to ease computation of | det J |.
Note that by keeping the diagonals of the other natrix to have value != 0, W is kept invertible.
Akin to permutation layer in RealNVP.

- One glow step (inverse of $f_i$)

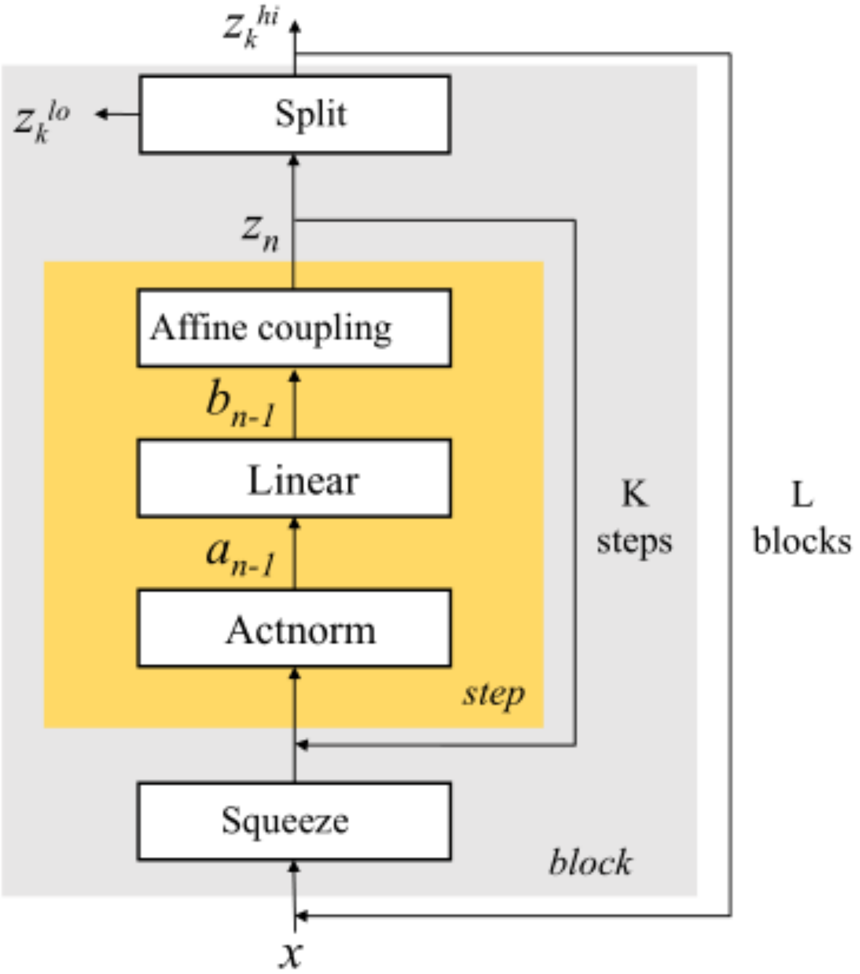



**Affine coupling layer**:
Half of the layer goes through unchanged ($b_{lo}$)
Other half ($b_{hi}$) appended to the transformed
(by Net A or $A_n$; possibly non-invertible) $b_{lo}$.

| Function | Reverse Function |
|---|---|
| $\mathbf{x}_a, \mathbf{x}_b = \mathtt{split}(\mathbf{x})$ | $\mathbf{y}_a, \mathbf{y}_b = \mathtt{split}(\mathbf{y})$ |
| $(\log \mathbf{s}, \mathbf{t}) = \mathtt{NN}(\mathbf{x}_b)$ | $(\log \mathbf{s}, \mathbf{t}) = \mathtt{NN}(\mathbf{y}_b)$ |
| $\mathbf{s} = \exp(\log \mathbf{s})$ | $\mathbf{s} = \exp(\log \mathbf{s})$ |
| $\mathbf{y}_a = \mathbf{s} \odot \mathbf{x}_a + \mathbf{t}$ | $\mathbf{x}_a = (\mathbf{y}_a - \mathbf{t}) / \mathbf{s}$ |
| $\mathbf{y}_b = \mathbf{x}_b$ | $\mathbf{x}_b = \mathbf{y}_b$ |
| $\mathbf{y} = \mathtt{concat}(\mathbf{y}_a, \mathbf{y}_b)$ | $\mathbf{x} = \mathtt{concat}(\mathbf{x}_a, \mathbf{x}_b)$ |

- One glow step (inverse of $f_i$)




**Hierarchical Decomposition:**
Process iterated for L blocks,
Which can increase the power to model long-range dependencies.

# Model: Conditional NF & Fixed-length models with control

- Consider that in a vanilla NF model, one can sample a random point in **z** and reconstruct a likely **x** in the data distribution.
- But this is not very helpful if you want to generate a sequence of poses that is not totally random.
- That is why we condition the NF during training and testing using a co ntrol signal, and it is basically appended (author uses the term "squeez ed") in the affine coupling layers.

- First we present the fixed-length model which is a convolutional NF mo del that accepts a fixed-length sequence: $\boldsymbol{x}_t \in \mathbb{R}^D$, $\boldsymbol{c}_t \in \mathbb{R}^C$
- $x_t$ is pose at time t, and $c_t$ is the control signal at time t or the "global" control parameter.

# Model: Fixed-length models with control

$$\left[\boldsymbol{z}_{t,\,n+1}^{\mathrm{lo}},\, \boldsymbol{z}_{t,\,n+1}^{\mathrm{hi}}\right] = \left[\boldsymbol{b}_{t,\,n}^{\mathrm{lo}},\, \left(\boldsymbol{b}_{t,\,n}^{\mathrm{hi}} + \boldsymbol{t}'_{t,\,n}\right) \odot \boldsymbol{s}'_{t,\,n}\right]$$

$$\left[\boldsymbol{s}'_{t,\,n},\, \boldsymbol{t}'_{t,\,n}\right] = A_n\left(\boldsymbol{b}_{t,\,n}^{\mathrm{lo}}\right)$$

$$\Longrightarrow$$

Add past frames info
(convolutional fixed-length model)

$$\left[\boldsymbol{s}'_{t,\,n},\, \boldsymbol{t}'_{t,\,n}\right] = A_n\left(\boldsymbol{b}_{t-1:t+1,\,n}^{\mathrm{lo}}\right).$$

Add conditioning

$$\Big\Downarrow$$

$$\left[\boldsymbol{s}'_{t,\,n},\, \boldsymbol{t}'_{t,\,n}\right] = A_n\left(\boldsymbol{b}_{t-1:t+1,\,n}^{\mathrm{lo}},\, \boldsymbol{c}_{t-1:t+1,\,n}\right)$$

# Model: MoGlow

- The official model that is proposed is, as opposed to the fixed-length sequ ence model from earlier, an autoregressive model that can generate sequ ence of arbitrary length.
- Because the fixed-length model contains a time-sequence (which may be l ong), L = 4 blocks, K = 32 each block.
- The autoregressive model only has L = 1 block, K = 16 steps each.
- LSTM used within affine coupling layer

# Model: MoGlow

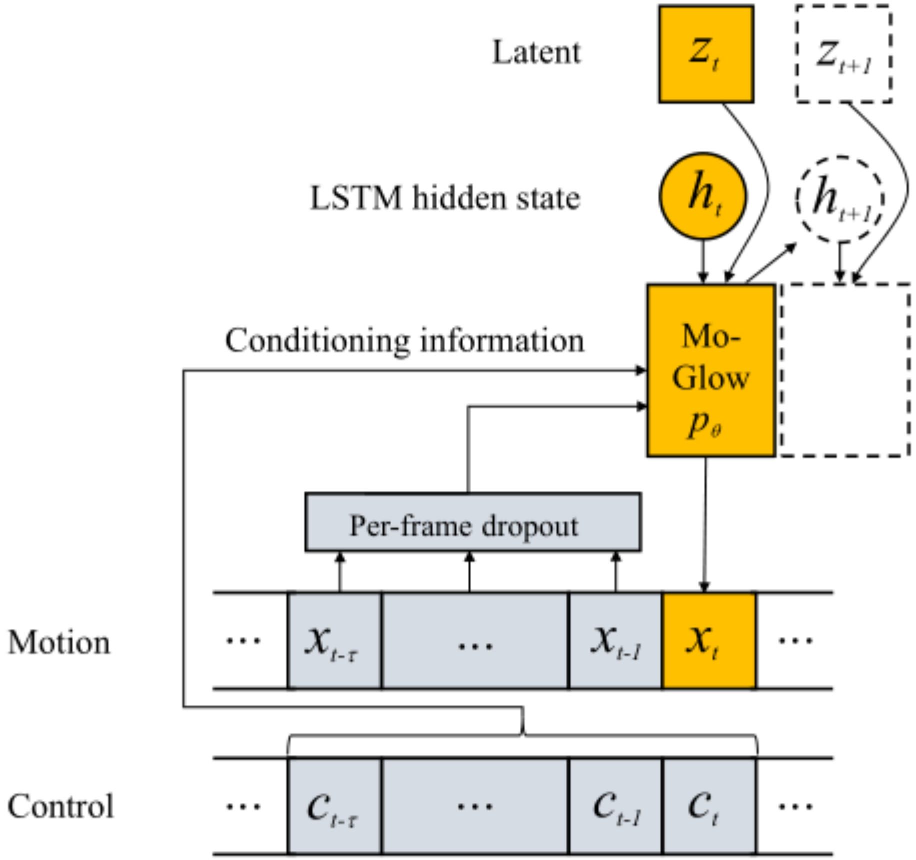
- Model is formulated as follows:

$$p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{c}) = p(\boldsymbol{x}_{1:\tau} \mid \boldsymbol{c}_{1:\tau}) \cdot \prod_{t=\tau+1}^{T} p_{\boldsymbol{\theta}}(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-\tau:t-1}, \boldsymbol{c}_{t-\tau:t}, \boldsymbol{h}_t) \quad (16)$$

$$\boldsymbol{h}_{t+1} = \boldsymbol{g}_{\boldsymbol{\theta}}(\boldsymbol{x}_{t-\tau:t-1}, \boldsymbol{c}_{t-\tau:t}, \boldsymbol{h}_t) \quad (17)$$

$$[\boldsymbol{s}'_{t,n}, \boldsymbol{t}'_{t,n}] = A_n\left(\boldsymbol{b}^{\mathrm{lo}}_{t,n}, \boldsymbol{x}_{t-\tau:t-1}, \boldsymbol{c}_{t-\tau:t,n}, \boldsymbol{h}_t\right), \quad (18)$$

- Notice that the hidden state ht of LSTM is also added as a conditioning var (along with control parameter), and allows for longer time dependence than τ.

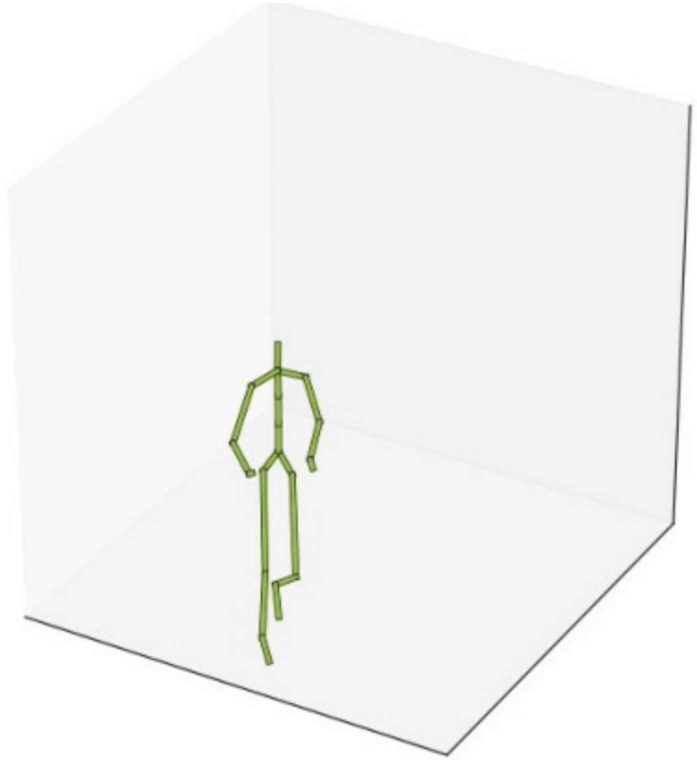


# Implementation: Locomotion Trials

- Trained on **locomotion trials** from CMU and HDM05 motion capture data sets (part of AMASS).
- The input signals are local 3D positions of 21 joints, and the control signal s are the displacement of the root at each frame.
- Data augmented by mirroring spatially and temporally.
- Fixed length model trained 20k steps, autoregressive 80k steps with Ada m optimiser.

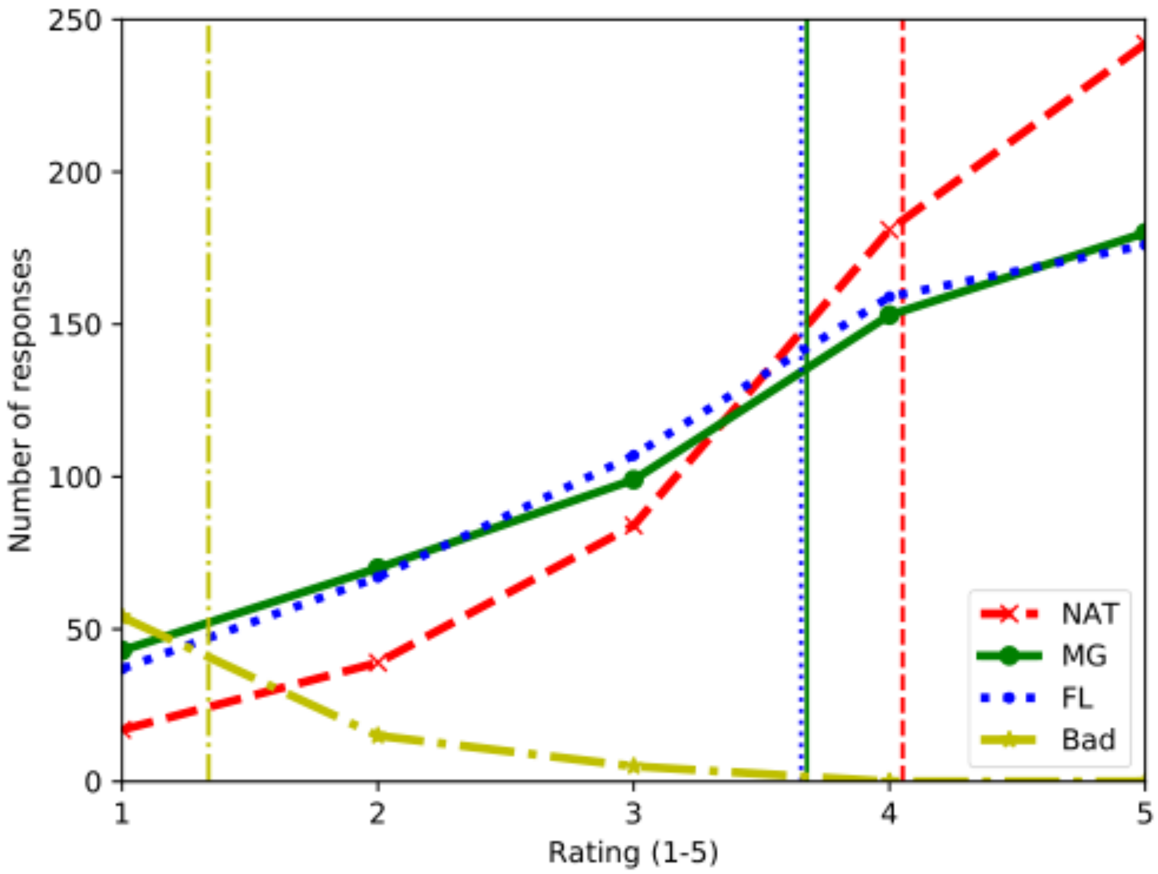
# Results

- During inference provided the path at each frame a random **z** is sampled a nd with the conditionals (control signal & hidden states) a random walking motion is generated autoregressively.
- Synthetic motion examples: https://youtu.be/lYhJnDBWyeo



# Results

- For quantitative evaluation, participants were asked to give a subjective ev aluation of how “natural” the generated/real motions were on a scale: 1-5.




Nat: real data; MG: MoGlow; FL: fixed-length model; Bad: bad animation from early training iters

# Discussion

- Official open-source implementation available:

https://github.com/simonalexanderson/StyleGestures

- Updated research: *Style-Controllable Speech-Driven Gesture Synthesis Using Normalising Flows* came out in 2020