CS5489 - Machine Learning

Lecture 11b - Auto-Encoders and Deep Generative Models

Prof. Antoni B. Chan

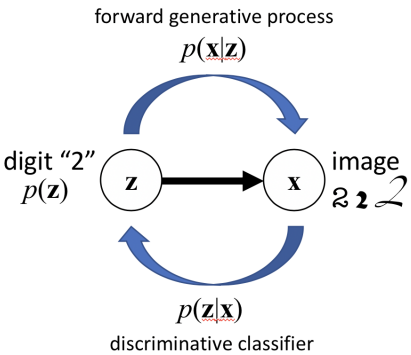
Dept. of Computer Science, City University of Hong Kong

Outline

- Unsupervised models
  - Autoencoders
  - Denoising Autoencoders
- **Deep generative models**
  - Variational Autoencoders
  - GANs
  - Diffusion Models

Deep Generative Models

- **Why are they necessary?**
  - typical mapping for discriminative classifiers:
    - many inputs to one output (e.g., images of "2" to class label "2")
  - typical mapping for synthesis:
    - one input to many outputs (e.g., label 2 to images of "2")



- typical NN can only predict one output
  - if multiple outputs are possible, it predicts the mean to minimize the loss
    - e.g. suppose $\mathbf{x}$ is associated with K outputs $\{y_j\}$.
    - $\min \sum_{j=1}^{K} ||y_j - f(\mathbf{x})||^2 \Rightarrow f(\mathbf{x}) = \frac{1}{K}\sum_j \mathbf{y}_j$
- *How to extend NN to be generative models?*

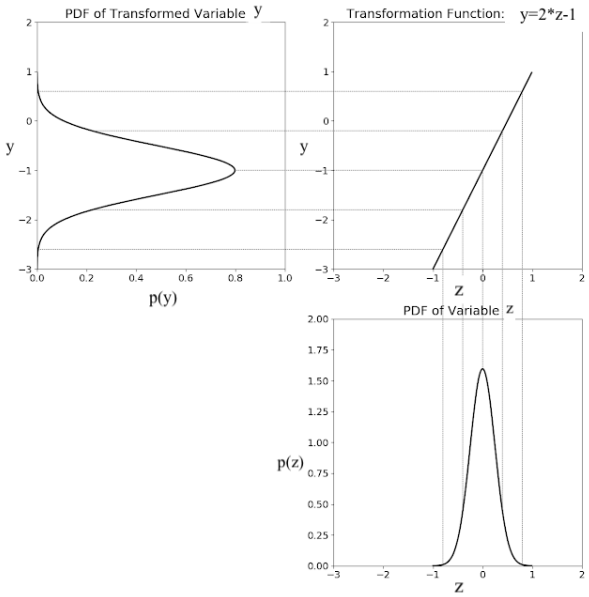
- **simple idea:** use NN to predict the parameters of a distribution
  - e.g., $y \sim \mathcal{N}(\mu(\mathbf{x}), \sigma(\mathbf{x})^2)$
  - sample from the Gaussian to obtain values of y for input $\mathbf{x}$.
- Problems:
  - *How to do back propagation through the random variable?*
  - *How to take the derivative with respect to (μ, σ^2)?*

Reparameterization Trick

- **Solution:**
  - rewrite y as a function of another random variable with known parameters
    - e.g., $y = \mu(\mathbf{x}) + \sigma(\mathbf{x})z, \quad z \sim \mathcal{N}(0, 1)$
  - decompose y into parameters and "randomness"
  - can now take derivatives w.r.t (μ, σ^2) since z is not a function of (μ, σ^2).
    - z is an extra input (random samples)

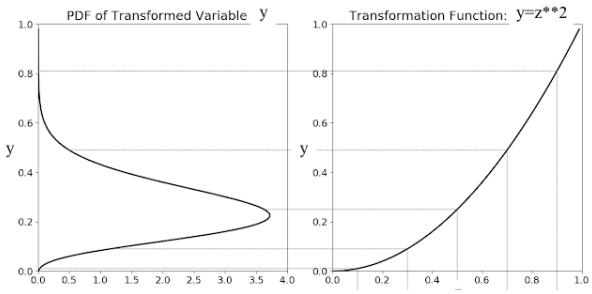
- Example:

$$y = 2 * z - 1$$



- In general, we rewrite the distribution as
  - $y \sim p(y|x) \Rightarrow y = f(z, x), z \sim p(z)$
    - x are the inputs, z is the source of randomness.
  - f transforms samples of z to samples of y
    - modeled as a NN and learned.

- Example: $y = z^2$



What is the actual pdf?

- transform of a random variable: $y = f(z), z \sim p_z(z)$
  - assume f is monotonically increasing, then $z = f^{-1}(y)$.
  - mapping density from z to y:

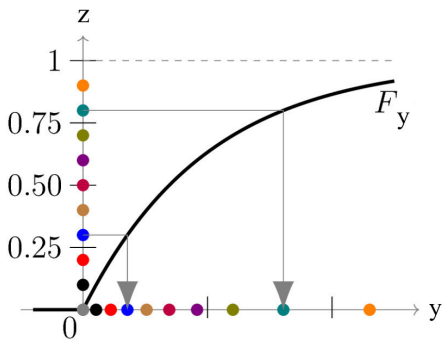
$$
\begin{aligned}
|p_y(y)dy| &= |p_z(z)dz| \\
\Rightarrow p_y(y) &= \left|\frac{dz}{dy}\right| p_z(z) = \left|\frac{df^{-1}(y)}{dy}\right| p_z(f^{-1}(y))
\end{aligned}
$$

  - hard to evaluate...
    - ... learning f is indirectly learning the density $p_y(y)$.

Closer look

- Suppose $f^{-1}(y) = F_y(y) = \int_{-\infty}^{y} p_y(y)dy$ is the CDF of y
  - then $\frac{df^{-1}(y)}{dy} = \frac{dF_y(y)}{dy} = p_y(y)$
- Assume $z \sim U(0,1)$ is a uniform distribution over [0,1]
  - then $p_z(f^{-1}(y)) = 1$.
- Finally, substituting we obtain
  - $\left|\frac{df^{-1}(y)}{dy}\right| p_z(f^{-1}(y)) = p_y(y)$
- Thus, when $y = f(z)$ and z is uniform(0,1),
  - learning f is equivalent to learning the inverse CDF of y.

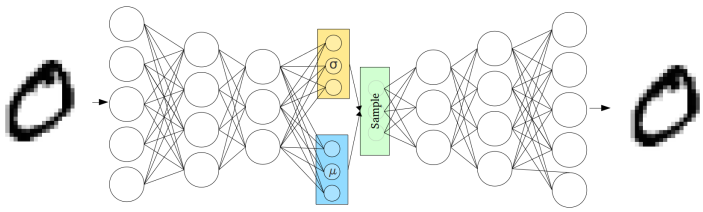
- Example:
  - z samples are mapped to x



Variational AutoEncoder (VAE)

- The standard autoencoder can have difficulty encoding/decoding new images
  - the decoder never sees (encoded) latent vectors outside of the training set
- VAE fixes this by introducing noise in the latent vectors

- the noise lets the decoder network see slightly different latent vectors for each training image.
- improves the ability to interpolate between training samples



VAE model

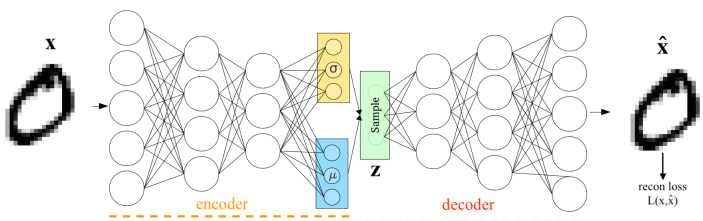
- consider the probabilistic model
  - let $\mathbf{z}$ be the latent variable, and $\mathbf{x}$ the observation.
  - assume prior $p(\mathbf{z}) \sim \mathcal{N}(0, \mathbf{I})$
  - let $p(\mathbf{x}|\mathbf{z})$ be the likeihood function
    - i.e., $-\log p(\mathbf{x}|\mathbf{z})$ is the data loss function

- given a $\mathbf{x}_i$ we want to infer $\mathbf{z}_i$
  - let $q(\mathbf{z}_i|\mathbf{x}_i) \sim \mathcal{N}(\mu(\mathbf{x}_i), \text{diag}(\sigma(\mathbf{x}_i)^2))$ be the posterior of $\mathbf{z}_i$ given observation $\mathbf{x}_i$.
    - $\mu(\cdot), \sigma(\cdot)$ are the outputs of a NN.
  - samples from posterior are generated via reparameterization trick
    - $\mathbf{z}_{il} = \mu(\mathbf{x}_i) + \sigma(\mathbf{x}_i) \circ \epsilon_{il}$
    - where $\epsilon_{il} \sim \mathcal{N}(0, \mathbf{I})$

VAE learning

- Ideally we would want to learn the parameters by maximizing the marginal log-likelihood.
  - $\log p(\mathbf{x}) = \log \int p(\mathbf{x}|\mathbf{z})p(\mathbf{z})d\mathbf{z}$.
- Intractable, so we maximize a lower-bound using a **variational approximation**
  - ...more details about this approximation in the next lecture on Graphical Models.

- After some derivation, we obtain the VAE loss function
  - $L(\mathbf{x}_i) = \underbrace{-\frac{1}{L}\sum_{l=1}^{L} \log p(\underbrace{\mathbf{x}_i|\mathbf{z}_{il}}_{\text{decoder}})}_{\text{reconstruction loss}} + \underbrace{KL(q(\mathbf{z}|\mathbf{x}_i), p(\mathbf{z}_i))}_{\text{regularizer}}$
  - where $\underbrace{\mathbf{z}_{il} = \mu(\mathbf{x}_i) + \sigma(\mathbf{x}_i) \circ \epsilon_{il}, \;\; \epsilon_{il} \sim \mathcal{N}(0, \mathbf{I})}_{\text{encoder}}$
  - decoder is the output of a NN fed into the reconstruction loss
    - (e.g., MSE, binary cross-entropy)
  - regularizer keeps the posterior close to the prior.



VAE Example

In [14]:
```
#some settings
K.clear_session()
random.seed(4487); tf.random.set_seed(4487)
original_dim = 784
input_shape = (original_dim, )
intermediate_dim = 512
batch_size = 128
latent_dim = 10
epochs = 50
```

- Build the encoder
- Map the input into the mean and log(sigma) of the Gaussian distribution
  - the mean is the encoded vector

In [15]:
```
# encoder mapping to distribution (mean and log_sigma)
x = Input(shape=(original_dim,), name='input')
h = Dense(intermediate_dim, activation='relu', name='enc_hidden')(x)

# the mean and log-sigma
z_mean = Dense(latent_dim, name='mu')(h)
z_log_sigma = Dense(latent_dim, name='log_sigma')(h)

# encoder, from inputs to latent space
encoder = Model(x, z_mean, name='encoder')
```

- Use the mean and log(sigma) to sample a latent variable z
  - the `Lambda` layer applies a function to an input, and outputs it.
  - use keras backend `K` for all math operations, so backprop can be used

In [16]:
```
# sampling function - draw Gaussian random noise
epsilon_std = 0.01  # fixed stddev
def sampling(args):
    z_mean, z_log_sigma = args
    epsilon = K.random_normal(shape=K.shape(z_mean), mean=0., stddev=epsilon_std)
    return z_mean + K.exp(z_log_sigma) * epsilon

# layer that samples according to mean and sigma
z = Lambda(sampling, name='sampler')([z_mean, z_log_sigma])
```

- Decode the latent variable z
- Construct the whole VAE

In [17]:
```
# create the layers since we need to use it later
decoder_h = Dense(intermediate_dim, activation='relu', name='dec_hidden')
decoder_mean = Dense(original_dim, activation='sigmoid', name='output')

# connect the latent variable and hidden states
h_decoded = decoder_h(z)
x_decoded_mean = decoder_mean(h_decoded)

# end-to-end variational autoencoder
vae = Model(x, x_decoded_mean)
```

- Construct the generator (decoder)
  - attach another input to the decoder layers, and connect them

In [18]:
```
# generator, from latent space to reconstructed inputs
decoder_input = Input(shape=(latent_dim,))  # make an input and attach it to the hidden stat
_h_decoded = decoder_h(decoder_input)       # and other layers
_x_decoded_mean = decoder_mean(_h_decoded)

# the generator model
generator = Model(decoder_input, _x_decoded_mean)
```

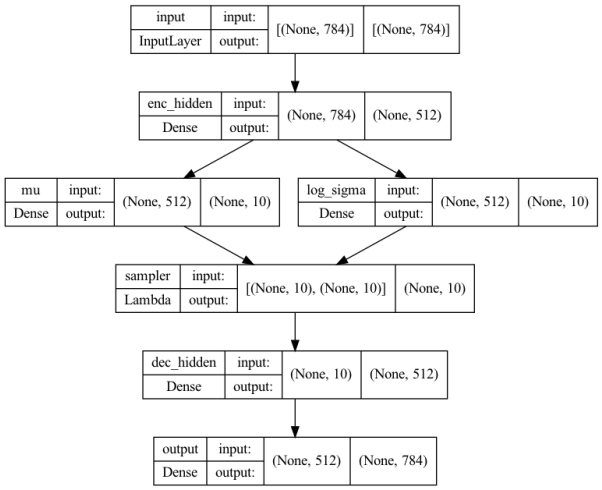
- define the VAE loss function

In [19]:
```
# define the VAE loss
def vae_loss(x, x_decoded_mean):
    # cross-entropy loss (recontruction loss)
    xent_loss = keras.losses.binary_crossentropy(x, x_decoded_mean)
    # KL divergence loss  (regularizer)
    kl_loss = -0.5 * K.mean(1 + z_log_sigma - K.square(z_mean) - K.exp(z_log_sigma), axis=-1
    return xent_loss + kl_loss

# compile the model for optimization
vae.compile(optimizer='rmsprop', loss=vae_loss)
```

- The final VAE model
  - note that there are two layers going into "sampler"

In [21]:
```
tmp=tf.keras.utils.plot_model(vae, show_shapes=True); tmp.width=500; tmp
```

Out[21]:



- The encoder and decoder

In [22]:
```
encoder.summary()
```

```
Model: "encoder"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 input (InputLayer)           [(None, 784)]             0

 enc_hidden (Dense)           (None, 512)               401920

 mu (Dense)                   (None, 10)                5130

=================================================================
Total params: 407,050
Trainable params: 407,050
Non-trainable params: 0
_________________________________________________________________
```

In [23]:
```
generator.summary()
```

```
Model: "model_1"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 input_1 (InputLayer)         [(None, 10)]              0

 dec_hidden (Dense)           (None, 512)               5632

 output (Dense)               (None, 784)               402192

=================================================================
```

```
Total params: 407,824
Trainable params: 407,824
Non-trainable params: 0
_________________________________________________________________
```

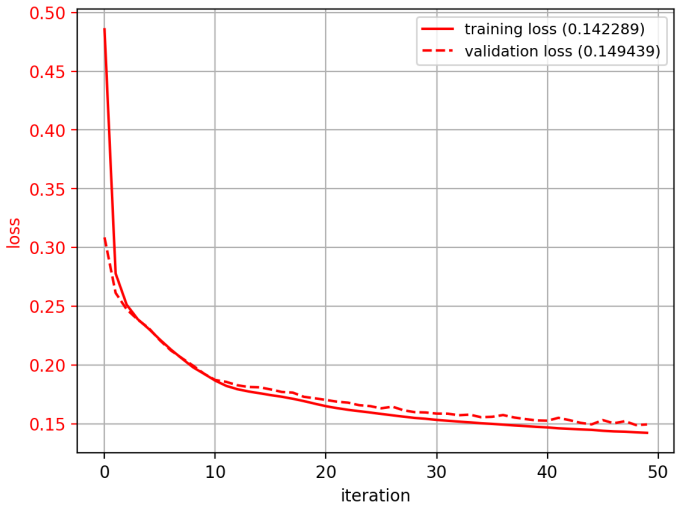
- Fit the model

In [24]:
```
history = vae.fit(vtrainXraw, vtrainXraw,
            shuffle=True,
            epochs=epochs,
            batch_size=batch_size,
            validation_data=(validXraw, validXraw),
            callbacks=[TensorBoard(log_dir='./logs/vae')],
            verbose=False
           )
```

```
Metal device set to: Apple M1 Max

2023-01-25 20:22:47.299282: I tensorflow/core/common_runtime/pluggable_device/plug
gable_device_factory.cc:305] Could not identify NUMA node of platform GPU ID 0, de
faulting to 0. Your kernel may not have been built with NUMA support.
2023-01-25 20:22:47.299421: I tensorflow/core/common_runtime/pluggable_device/plug
gable_device_factory.cc:271] Created TensorFlow device (/job:localhost/replica:0/t
ask:0/device:GPU:0 with 0 MB memory) -> physical PluggableDevice (device: 0, name:
METAL, pci bus id: <undefined>)
2023-01-25 20:22:47.307580: W tensorflow/core/platform/profile_utils/cpu_utils.cc:
128] Failed to get CPU frequency: 0 Hz
2023-01-25 20:22:47.307688: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
2023-01-25 20:22:47.321445: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
2023-01-25 20:22:47.375246: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
2023-01-25 20:22:47.630258: I tensorflow/core/profiler/lib/profiler_session.cc:11
0] Profiler session initializing.
2023-01-25 20:22:47.630272: I tensorflow/core/profiler/lib/profiler_session.cc:12
5] Profiler session started.
2023-01-25 20:22:47.641492: I tensorflow/core/profiler/lib/profiler_session.cc:67]
Profiler session collecting data.
2023-01-25 20:22:47.641904: I tensorflow/core/profiler/lib/profiler_session.cc:14
3] Profiler session tear down.
2023-01-25 20:22:47.642590: I tensorflow/core/profiler/rpc/client/save_profile.cc:
136] Creating directory: ./logs/vae/plugins/profile/2023_01_25_20_22_47

2023-01-25 20:22:47.642864: I tensorflow/core/profiler/rpc/client/save_profile.cc:
142] Dumped gzipped tool data for trace.json.gz to ./logs/vae/plugins/profile/2023
_01_25_20_22_47/Kili.trace.json.gz
2023-01-25 20:22:47.643809: I tensorflow/core/profiler/rpc/client/save_profile.cc:
136] Creating directory: ./logs/vae/plugins/profile/2023_01_25_20_22_47

2023-01-25 20:22:47.643965: I tensorflow/core/profiler/rpc/client/save_profile.cc:
142] Dumped gzipped tool data for memory_profile.json.gz to ./logs/vae/plugins/pro
file/2023_01_25_20_22_47/Kili.memory_profile.json.gz
2023-01-25 20:22:47.644336: I tensorflow/core/profiler/rpc/client/capture_profile.
cc:251] Creating directory: ./logs/vae/plugins/profile/2023_01_25_20_22_47
Dumped tool data for xplane.pb to ./logs/vae/plugins/profile/2023_01_25_20_22_47/K
ili.xplane.pb
Dumped tool data for overview_page.pb to ./logs/vae/plugins/profile/2023_01_25_20_
22_47/Kili.overview_page.pb
Dumped tool data for input_pipeline.pb to ./logs/vae/plugins/profile/2023_01_25_20
_22_47/Kili.input_pipeline.pb
Dumped tool data for tensorflow_stats.pb to ./logs/vae/plugins/profile/2023_01_25_
20_22_47/Kili.tensorflow_stats.pb
Dumped tool data for kernel_stats.pb to ./logs/vae/plugins/profile/2023_01_25_20_2
2_47/Kili.kernel_stats.pb

/Users/abc/miniforge3/envs/py38tfm28/lib/python3.8/site-packages/keras/engine/trai
ning_v1.py:2057: UserWarning: `Model.state_updates` will be removed in a future ve
rsion. This property should not be used in TensorFlow 2.0, as `updates` are applie
d automatically.
  updates = self.state_updates
```

```
2023-01-25 20:22:48.054219: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
```

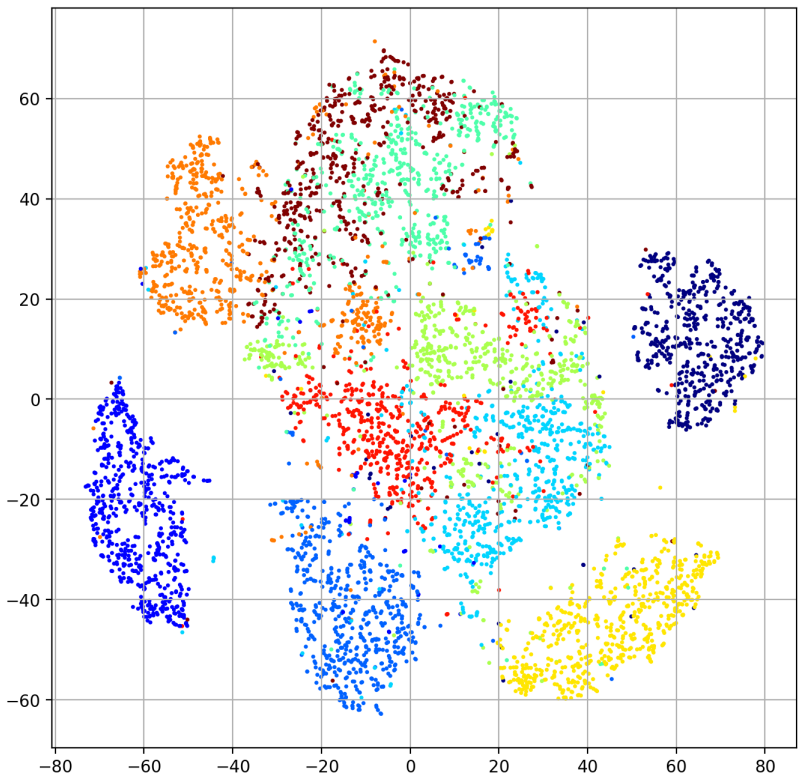
In [25]:
```
plot_history(history)
```



- View t-SNE embedding fo the latent space.

In [27]:
```
Z = encoder.predict(trainXraw)
plot_embedding(Z, trainY)
```

```
/Users/abc/miniforge3/envs/py38tfm28/lib/python3.8/site-packages/keras/engine/trai
ning_v1.py:2079: UserWarning: `Model.state_updates` will be removed in a future ve
rsion. This property should not be used in TensorFlow 2.0, as `updates` are applie
d automatically.
  updates=self.state_updates,
2023-01-25 20:23:05.313357: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
```



- View some reconstruction results

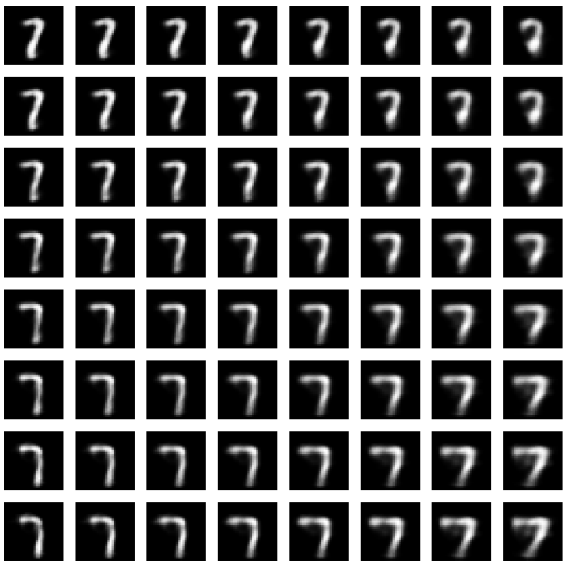
In [29]:
```
rfig
```

Out[29]:

- visualize the latent space by interpolating between

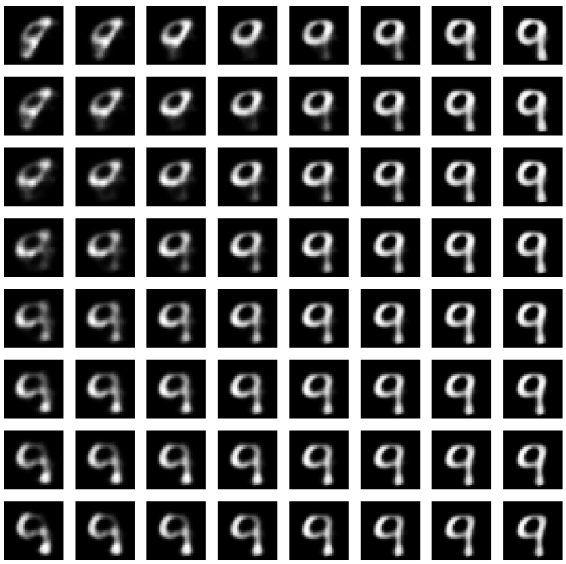
In [32]:
```
rfig
```

Out[32]:



In [34]:
```
rfig
```

Out[34]:



Convolutional VAE

- The previous VAE is using fully-connected layers
- Since the inputs are images, then replace the Dense layers with Conv2D and Pooling
  - The encoder has 3 outputs: the latent mean, log-sigma, and the sampled z
  - Latent dimension is 10

In [35]:
```
# the Conv2D encoder
K.clear_session()
random.seed(4487); tf.random.set_seed(4487)
```

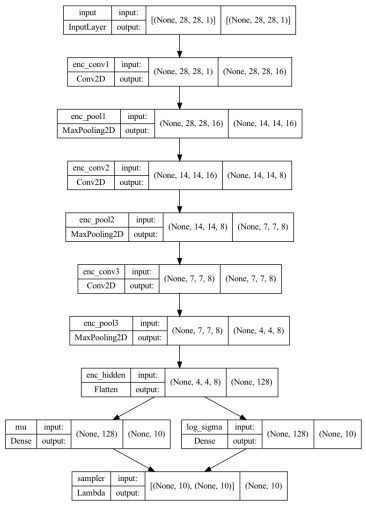
```
latent_dim = 10
input_img2 = Input(shape=(28, 28, 1), name='input')
x = Conv2D(16, (3, 3), activation='relu', padding='same', name='enc_conv1')(input_img2)
x = MaxPooling2D((2, 2), padding='same', name='enc_pool1')(x)
x = Conv2D(8, (3, 3), activation='relu', padding='same', name='enc_conv2')(x)
x = MaxPooling2D((2, 2), padding='same', name='enc_pool2')(x)
x = Conv2D(8, (3, 3), activation='relu', padding='same', name='enc_conv3')(x)
x = MaxPooling2D((2, 2), padding='same', name='enc_pool3')(x)
x = Flatten(name='enc_hidden')(x)

# the mean and log-sigma
z_mean      = Dense(latent_dim, name='mu')(x)
```

In [36]:
```
# sampling step
epsilon_std = 0.01
def sampling(args):
    z_mean, z_log_sigma = args
    epsilon = K.random_normal(shape=K.shape(z_mean), stddev=epsilon_std)
    return z_mean + K.exp(z_log_sigma) * epsilon

z = Lambda(sampling, name='sampler')([z_mean, z_log_sigma])

# build the encoder (outputs mean, log-sigma, and z)
encoder = Model(input_img2, [z_mean, z_log_sigma, z], name='encoder')
```

- Encoder summary

In [38]:
```
tmp=tf.keras.utils.plot_model(encoder, show_shapes=True); tmp.width=300; tmp
```

Out[38]:

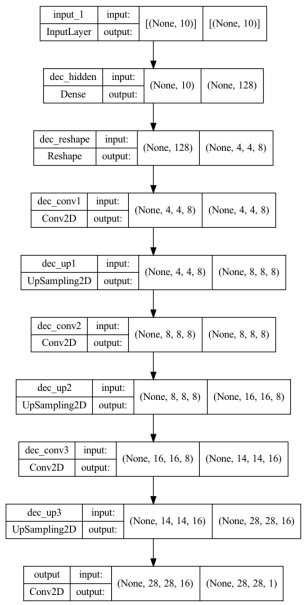


- Same for the decoder

In [39]:
```
# the Conv2D decoder
encoded_input2 = Input(shape=(latent_dim,))
x = Dense(128, activation='relu', name='dec_hidden')(encoded_input2)
x = Reshape((4,4,8), name='dec_reshape')(x)
x = Conv2D(8, (3, 3), activation='relu', padding='same', name='dec_conv1')(x)
x = UpSampling2D((2, 2), name='dec_up1')(x)
x = Conv2D(8, (3, 3), activation='relu', padding='same', name='dec_conv2')(x)
x = UpSampling2D((2, 2), name='dec_up2')(x)
x = Conv2D(16, (3, 3), activation='relu', name='dec_conv3')(x)
x = UpSampling2D((2, 2), name='dec_up3')(x)
x = Conv2D(1, (3, 3), activation='sigmoid', padding='same', name='output')(x)

decoder = Model(encoded_input2, x, name='decoder')
```

In [41]:
```
tmp=tf.keras.utils.plot_model(decoder, show_shapes=True); tmp.width=250; tmp
```

Out[41]:



- make VAE:
  - connect the encoder and decoder
  - select the sampled z of the encoder output as the input into the decoder

In [42]:
```
vae = Model(input_img2, decoder(encoder(input_img2)[2]))
```

In [43]:
```
vae.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input (InputLayer)          [(None, 28, 28, 1)]       0

 encoder (Functional)        [(None, 10),              4484
                              (None, 10),
                              (None, 10)]

 decoder (Functional)        (None, 28, 28, 1)         3889

=================================================================
Total params: 8,373
Trainable params: 8,373
Non-trainable params: 0
_________________________________________________________________
```

- The VAE loss as before

In [44]:
```
def vae_loss(x, x_decoded_mean):
    xent_loss = keras.losses.binary_crossentropy(x, x_decoded_mean)
    kl_loss = - 0.5 * K.mean(K.mean(1 + z_log_sigma - K.square(z_mean)
                                    - K.exp(z_log_sigma), axis=-1), axis=-1)
    return xent_loss + kl_loss
vae.compile(optimizer='adam', loss=vae_loss)
```

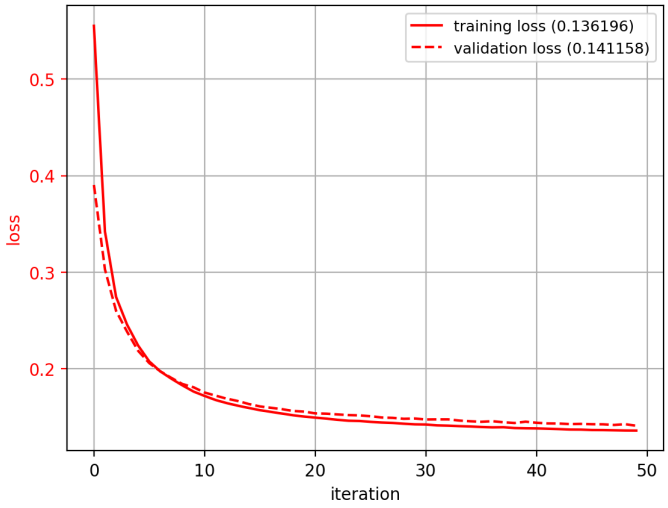
- Train the model

In [45]:
```
 # early stopping criteria
earlystop = keras.callbacks.EarlyStopping(monitor='val_loss',
                           min_delta=0.0001, patience=10,
                           verbose=1, mode='auto')

history = vae.fit(vtrainI, vtrainI,
            shuffle=True,
            epochs=epochs,
            batch_size=batch_size,
```

```
            validation_data=(validI, validI),
            callbacks=[earlystop, TensorBoard(log_dir='./logs/vae')],
            verbose=False
           )
```

```
2023-01-25 20:23:21.735836: I tensorflow/core/common_runtime/pluggable_device/plug
gable_device_factory.cc:305] Could not identify NUMA node of platform GPU ID 0, de
faulting to 0. Your kernel may not have been built with NUMA support.
2023-01-25 20:23:21.735855: I tensorflow/core/common_runtime/pluggable_device/plug
gable_device_factory.cc:271] Created TensorFlow device (/job:localhost/replica:0/t
ask:0/device:GPU:0 with 0 MB memory) -> physical PluggableDevice (device: 0, name:
METAL, pci bus id: <undefined>)
2023-01-25 20:23:21.750214: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
2023-01-25 20:23:21.775544: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
2023-01-25 20:23:21.849793: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
2023-01-25 20:23:22.262396: I tensorflow/core/profiler/lib/profiler_session.cc:11
0] Profiler session initializing.
2023-01-25 20:23:22.262412: I tensorflow/core/profiler/lib/profiler_session.cc:12
5] Profiler session started.
2023-01-25 20:23:22.287540: I tensorflow/core/profiler/lib/profiler_session.cc:67]
Profiler session collecting data.
2023-01-25 20:23:22.288099: I tensorflow/core/profiler/lib/profiler_session.cc:14
3] Profiler session tear down.
2023-01-25 20:23:22.288603: I tensorflow/core/profiler/rpc/client/save_profile.cc:
136] Creating directory: ./logs/vae/plugins/profile/2023_01_25_20_23_22

2023-01-25 20:23:22.288956: I tensorflow/core/profiler/rpc/client/save_profile.cc:
142] Dumped gzipped tool data for trace.json.gz to ./logs/vae/plugins/profile/2023
_01_25_20_23_22/Kili.trace.json.gz
2023-01-25 20:23:22.289450: I tensorflow/core/profiler/rpc/client/save_profile.cc:
136] Creating directory: ./logs/vae/plugins/profile/2023_01_25_20_23_22

2023-01-25 20:23:22.289796: I tensorflow/core/profiler/rpc/client/save_profile.cc:
142] Dumped gzipped tool data for memory_profile.json.gz to ./logs/vae/plugins/pro
file/2023_01_25_20_23_22/Kili.memory_profile.json.gz
2023-01-25 20:23:22.290133: I tensorflow/core/profiler/rpc/client/capture_profile.
cc:251] Creating directory: ./logs/vae/plugins/profile/2023_01_25_20_23_22
Dumped tool data for xplane.pb to ./logs/vae/plugins/profile/2023_01_25_20_23_22/K
ili.xplane.pb
Dumped tool data for overview_page.pb to ./logs/vae/plugins/profile/2023_01_25_20_
23_22/Kili.overview_page.pb
Dumped tool data for input_pipeline.pb to ./logs/vae/plugins/profile/2023_01_25_20
_23_22/Kili.input_pipeline.pb
Dumped tool data for tensorflow_stats.pb to ./logs/vae/plugins/profile/2023_01_25_
20_23_22/Kili.tensorflow_stats.pb
Dumped tool data for kernel_stats.pb to ./logs/vae/plugins/profile/2023_01_25_20_2
3_22/Kili.kernel_stats.pb

/Users/abc/miniforge3/envs/py38tfm28/lib/python3.8/site-packages/keras/engine/trai
ning_v1.py:2057: UserWarning: `Model.state_updates` will be removed in a future ve
rsion. This property should not be used in TensorFlow 2.0, as `updates` are applie
d automatically.
  updates = self.state_updates
2023-01-25 20:23:23.467634: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
```

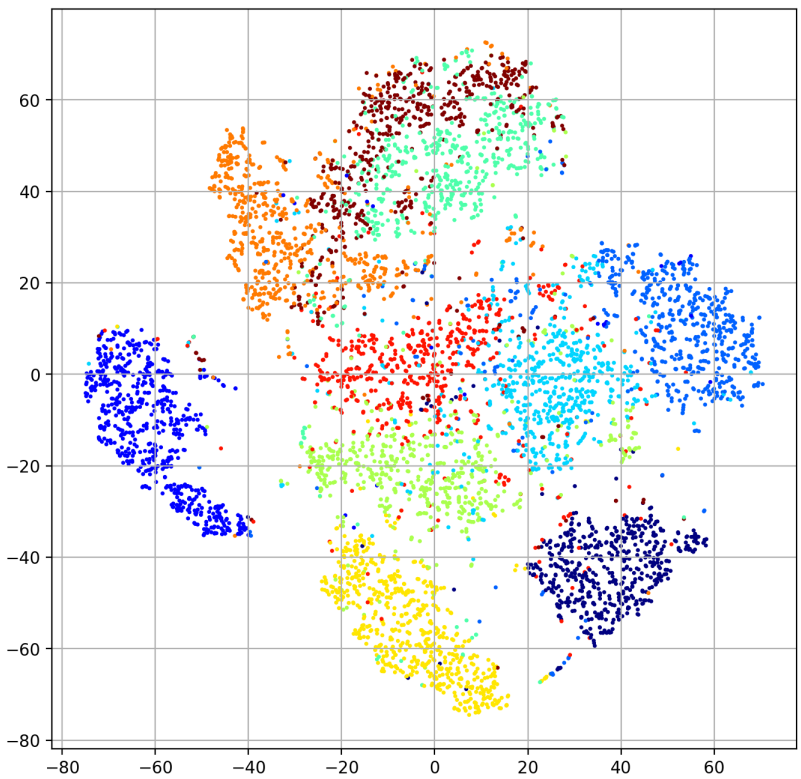
In [46]:
```
plot_history(history)
```



- Visualize the 10-dim latent space using t-SNE

In [47]:
```
# select z_mean from the encoder
Z = encoder.predict(trainI)[0]
plot_embedding(Z.reshape((Z.shape[0],-1)), trainY)
```

```
/Users/abc/miniforge3/envs/py38tfm28/lib/python3.8/site-packages/keras/engine/trai
ning_v1.py:2079: UserWarning: `Model.state_updates` will be removed in a future ve
rsion. This property should not be used in TensorFlow 2.0, as `updates` are applie
d automatically.
  updates=self.state_updates,
2023-01-25 20:24:14.464274: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
```



- Visualize the reconstruction

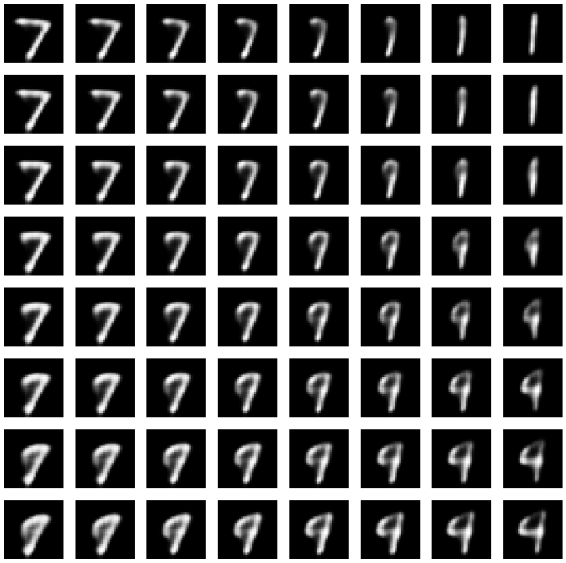
In [49]:
```
rfig
```

Out[49]:

- Visualize the latent space between a 7, 1, 9, and 4

In [51]:
```
rfig
```

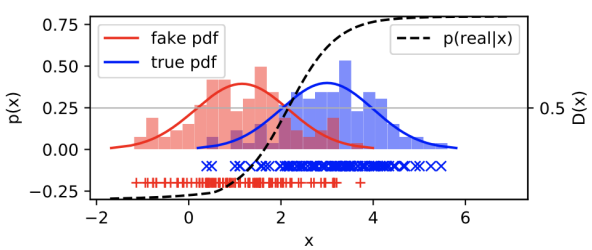
Out[51]:

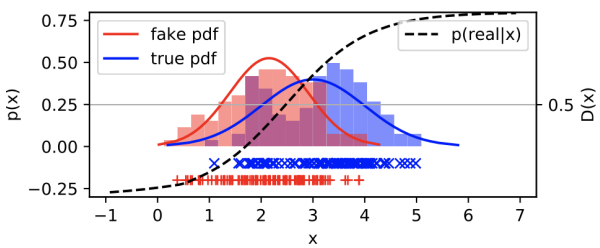


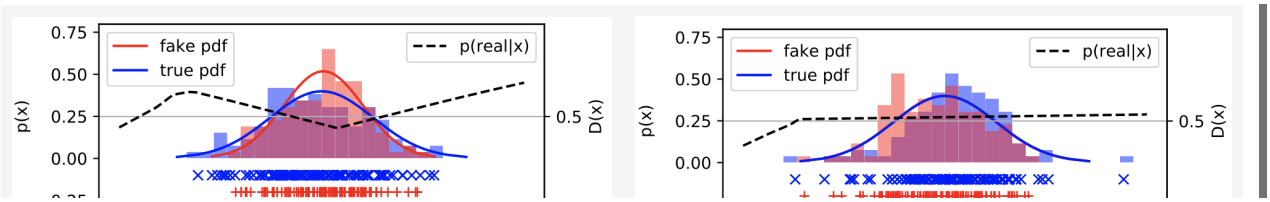
Generative Adversarial Networks (GAN)

- **Goal:** generate samples from a probability density $p_{data}(x)$ which is represented by a training set $\{x_i\}$.
- Setup
  - *true data samples:* $x \sim p_d(x)$
    - we have samples $\mathcal{X} = \{x_i\}$, but not p_d.
  - *generated samples:* $x \sim p_m(x)$
    - use the reparam trick to generate samples
    - $x = g(z)$, $z \sim N(0, I)$
    - $g(\cdot)$ is a neural network
  - Usually maximize the likelihood of $p_m(x)$ using samples $\mathcal{X}$.
- **Problem:**
  - cannot explicitly compute $p_m(x)$ because of the reparam trick.

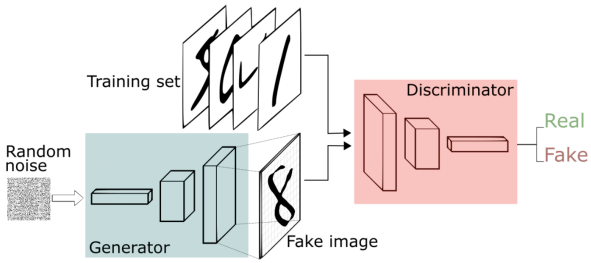
- **Solution:** look at the samples directly
- Build a classifier to predict whether a sample is *real* (from training samples) or *fake* (generated)







- Two models:
  - Generator (G) - make fake samples to fool the discriminator
  - Discriminator (D) - classify x as real or fake



- **Advantages:**
  - don't need to approximate the likelihood function or inference procedure
  - don't need the partition function (normalization constant)
  - easy to generate samples (good for synthesis)
- **Disadvantages:**
  - cannot compute the likelihood value (probability density) of a sample
  - difficult to train
  - ...

GAN Training

- Define two class labels:
  - $y = 1$ means "real" sample
  - $y = 0$ means "fake" sample
- Define discriminator (classifier): $D(x) = p(y = 1|x)$
  - probability of real sample
- Cross-entropy loss to train the discriminator
  - let $\{x_i, y_i\}$ be a dataset of true/fake samples.

$$\begin{aligned}&\min_D -\sum_i y_i \log D(x_i) + (1-y_i)\log(1-D(x_i))\\&= \max_D \underbrace{\sum_{y_i=1} \log D(x_i)}_{\text{real examples}} + \underbrace{\sum_{y_i=0} \log(1-D(x_i))}_{\text{generated examples}}\end{aligned}$$

- Replace the sum of samples with expectations

$$\begin{aligned}&\max_D \sum_{y_i=1} \log D(x_i) + \sum_{y_i=0} \log(1-D(x_i))\\&= \max_D \mathbb{E}_{x\sim p_d}[\log D(x_i)] + \mathbb{E}_{x\sim p_m}[\log(1-D(x_i))]\\&= \max_D \mathbb{E}_{x\sim p_d}[\log D(x_i)] + \mathbb{E}_{z\sim \mathcal{N}(0,I)}[\log(1-D(G(z))]\end{aligned}$$

- The G's objective is to maximize the cross entropy loss, i.e., make D predict wrongly.
- Final training problem:

$$\min_G \max_D \mathbb{E}_{x\sim p_d}[\log D(x_i)] + \mathbb{E}_{z\sim\mathcal{N}(0,I)}[\log(1-D(G(z))]$$

- **Notes:**
  - it is a *minimax* game (zero-sum game)
    - the values for D and G move in opposite directions
    - the equilibrium point is a *saddle* point
      - i.e., a local max for D, and local min for G
  - G can also be trained to make the D predict "real"
    - $\min_G -\mathbb{E}_z[\log D(G(z))]$
    - i.e., pass the generated samples with "real" class label to D

Analysis of GAN

- Assuming that G and D have enough capacity
- Considering the defined training problem:
  - the optimal discriminator is
    - $D^*(x) = \frac{p_d(x)}{p_d(x)+p_m(x)} = p(\text{real}|x)$
  - given the optimal discriminator, the optimal generator is
    - $p_m^*(x) = p_d(x)$
- thus, ideally, we can obtain the true distribution.

1-D GAN example

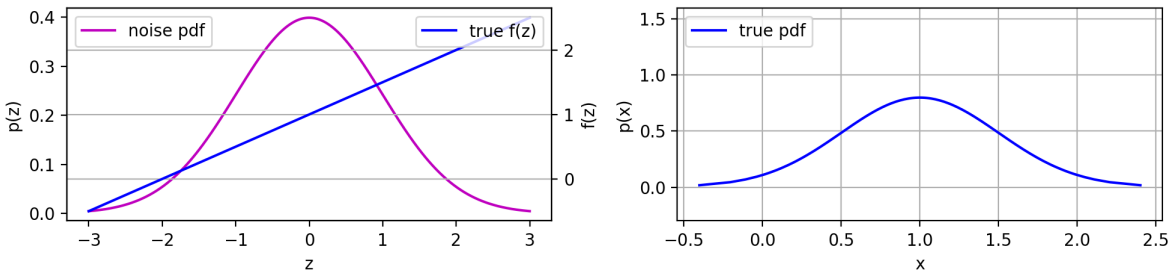
- the true distribution
  - transform `xform` of Gaussian(0,1)

In [55]:
```
# transformed gaussian
xform = lambda x: (0.5*x)+1
sample_range = get_bounds(xform)
```

In [56]:
```
# samples from the true distribution
def sample(N, mu=0.0, sigma=1):
    samples = xform(random.normal(mu, sigma,N))
    return reshape(samples,(len(samples),1))
```

In [57]:
```
plot_epoch(onlygt=True)
```

Out[57]:



- samples from the noise source

In [58]:
```
# samples from the generator noise source (Gaussian(0,1))
def g_sample(N,rng=5):
    a = random.normal(0,1,N)
    return reshape(a,(len(a),1))
```

- function to build a generator
  - the input is a 1-D noise source
  - output is a 1-D sample

In [59]:
```
# build the generator
def gen(hidden_size, n_layers=2, activation="tanh"):
    g_input = Input(shape=[1],name="in1")
    x = g_input
    # hidden layers
    for n in range(n_layers):
        x = Dense(hidden_size, activation=activation,
                  name="g{}".format(n))(x)
    # output layer
    ho = Dense(1, name="go")(x)
    g = Model(g_input,ho, name='generator')
    return g
```

- function to build a discriminator
  - the input is a 1-D sample
  - the output is class label (0=fake, 1=real)
  - need to `compile` since D is trained by itself

In [60]:
```
# build the discriminator
def disc(hidden_size, n_layers=2, activation='tanh'):
    inputd = Input(shape=[1],name="in2")
    x = inputd
    for n in range(n_layers):
        x = Dense( hidden_size * 2, activation=activation,
                  name="d{}".format(n))(x)
    ho = Dense(1, activation="sigmoid", name="do")(x)
    d = Model(inputd, ho, name='discriminator')
    d.compile(loss="binary_crossentropy", optimizer="sgd",
              metrics=['accuracy'])
    return d
```

- we will alternate training D and G
- this function sets the trainable flags in a `Model` to prevent/allow updating the weights

In [61]:
```
# set trainable flag for a model
def make_trainable(net, val):
    net.trainable = val   # this only works before compiling
    for l in net.layers:
        l.trainable = val # this works before/after compiling
```

- build the G and D
- connect G and D to make GAN
  - compile GAN for training G

In [62]:
```
K.clear_session()
random.seed(5489); tf.random.set_seed(5488)

# the generator
noise = Input(shape=[1])
generator = gen(10, activation='softplus')
g = generator(noise)

# the discriminator
discriminator = disc(10, activation='tanh')
d = discriminator(g)
make_trainable(discriminator, False)

# the GAN w/ noise input and discrim output
GAN = Model(noise, d)
GAN.compile(loss="binary_crossentropy", optimizer="sgd", metrics=['accuracy'])
GAN.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
```

```
 input_1 (InputLayer)        [(None, 1)]               0

 generator (Functional)      (None, 1)                 141

 discriminator (Functional)  (None, 1)                 481

=================================================================
Total params: 622
Trainable params: 141
Non-trainable params: 481
_________________________________________________________________
```

In [63]:
```
generator.summary()
```

```
Model: "generator"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 in1 (InputLayer)            [(None, 1)]               0

 g0 (Dense)                  (None, 10)                20

 g1 (Dense)                  (None, 10)                110

 go (Dense)                  (None, 1)                 11

=================================================================
Total params: 141
Trainable params: 141
Non-trainable params: 0
_________________________________________________________________
```

In [64]:
```
discriminator.summary()
```

```
Model: "discriminator"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 in2 (InputLayer)            [(None, 1)]               0

 d0 (Dense)                  (None, 20)                40

 d1 (Dense)                  (None, 20)                420

 do (Dense)                  (None, 1)                 21

=================================================================
Total params: 962
Trainable params: 481
Non-trainable params: 481
_________________________________________________________________
```

GAN Training

- Loop over training the discriminator and the generator
  - train the discriminator well (20 updates), vs 1 update for the generator

In [66]:
```
def train_for_n(nb_epoch=50, plt_frq=25, BATCH_SIZE=100, lr=0.01, g_inner=1, d_inner=20):
    epochs = {'g_loss':[], 'g_acc':[], 'd_loss':[], 'd_acc':[], 'figs':[]}

    for e in range(nb_epoch):
        #  Make generative sample
        sample_batch = sample(BATCH_SIZE)
        noise_gen = g_sample(BATCH_SIZE)
        generated_sample = generator.predict(noise_gen)

        # Train discriminator on generated samples
        X = concatenate((sample_batch, generated_sample))
        # class 1=real, class 0 = fake
        y = concatenate((ones((BATCH_SIZE,)), zeros((BATCH_SIZE,))))
```

```
        # train the discriminator a few steps on this batch
        make_trainable(discriminator,True)          # make it trainable
        K.set_value(discriminator.optimizer.lr, lr) # set the learning rate
        for i in range(d_inner):                    # train...
            d_loss = discriminator.train_on_batch(X,y)

        epochs['d_loss'].append(d_loss[0])
        epochs['d_acc'].append(d_loss[1])

        # Train the generator
        y2 = ones((BATCH_SIZE,))  # target for generator is real class (1)
        noise_gen = g_sample(BATCH_SIZE)

        # keep discriminator fixed

        # run a few updates on this batch
        make_trainable(discriminator,False)  # freeze discriminator
        K.set_value(GAN.optimizer.lr, lr)    # set learning rate
        for i in range(g_inner):             # train...
            g_loss = GAN.train_on_batch(noise_gen, y2)

        epochs['g_loss'].append(g_loss[0])
        epochs['g_acc'].append(g_loss[1])

        # some plots
        if (e % plt_frq==0) or (e+1==nb_epoch):
            fig=plot_epoch(noise_gen, sample_batch, "epoch {}".format(e))
            epochs['figs'].append(fig)

    return epochs
```

- train the GAN for 3000 epochs

In [67]:
```
epochs = train_for_n(nb_epoch=3000, lr=0.01, plt_frq=500)
```

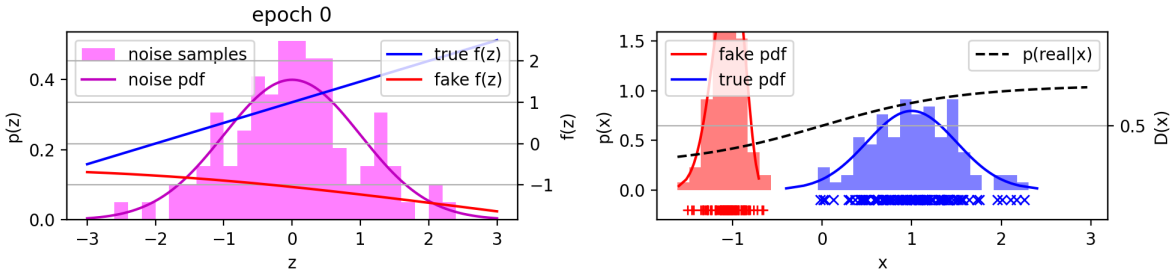
```
2023-01-25 20:24:30.842743: I tensorflow/core/common_runtime/pluggable_device/plug
gable_device_factory.cc:305] Could not identify NUMA node of platform GPU ID 0, de
faulting to 0. Your kernel may not have been built with NUMA support.
2023-01-25 20:24:30.842764: I tensorflow/core/common_runtime/pluggable_device/plug
gable_device_factory.cc:271] Created TensorFlow device (/job:localhost/replica:0/t
ask:0/device:GPU:0 with 0 MB memory) -> physical PluggableDevice (device: 0, name:
METAL, pci bus id: <undefined>)
2023-01-25 20:24:30.847946: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
2023-01-25 20:24:30.854803: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
2023-01-25 20:24:30.887429: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
2023-01-25 20:24:30.921538: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
2023-01-25 20:24:30.925908: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
2023-01-25 20:24:30.929600: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
2023-01-25 20:24:30.954262: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
2023-01-25 20:24:30.958606: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
2023-01-25 20:24:30.961102: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
2023-01-25 20:24:31.060969: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
2023-01-25 20:24:31.168323: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
2023-01-25 20:24:31.174022: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
2023-01-25 20:24:31.177585: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
2023-01-25 20:24:31.206336: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
2023-01-25 20:24:31.212040: I tensorflow/core/grappler/optimizers/custom_graph_opt
```

```
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
2023-01-25 20:24:31.214883: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
2023-01-25 20:24:31.335185: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
2023-01-25 20:24:31.470135: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
```

- Visualization:
  - (left) noise source and transformations
  - (right) true/fake pdfs; discriminator function $p(y=1|x)$ (prob of true sample)

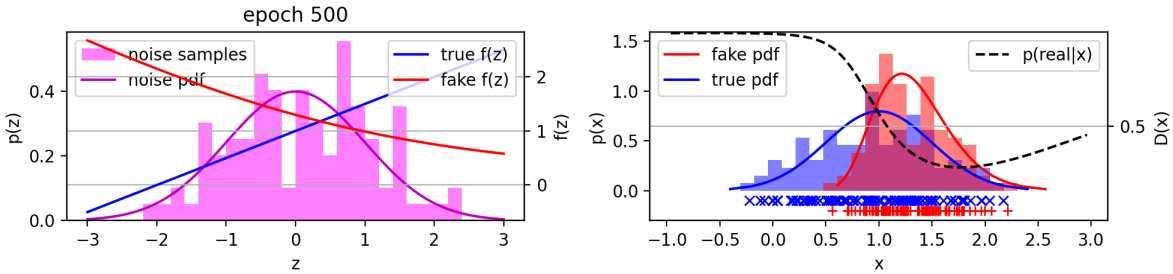
In [68]:
```
epochs['figs'][0]
```

Out[68]:



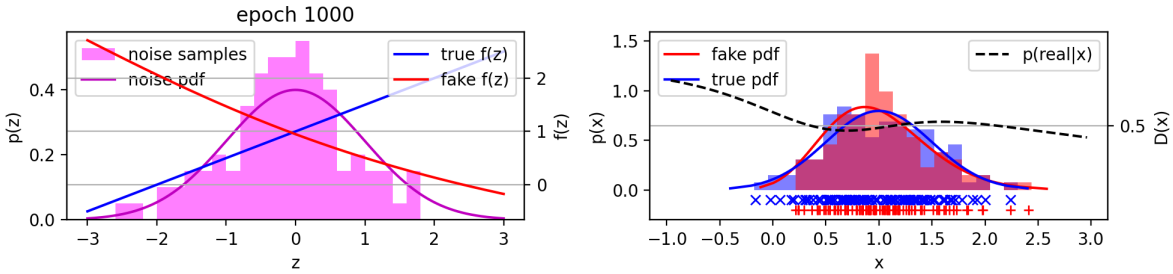
In [69]:
```
epochs['figs'][1]
```

Out[69]:



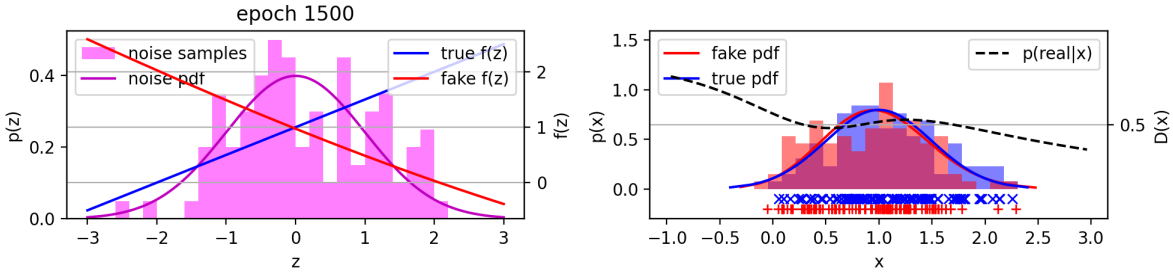
In [70]:
```
epochs['figs'][2]
```

Out[70]:



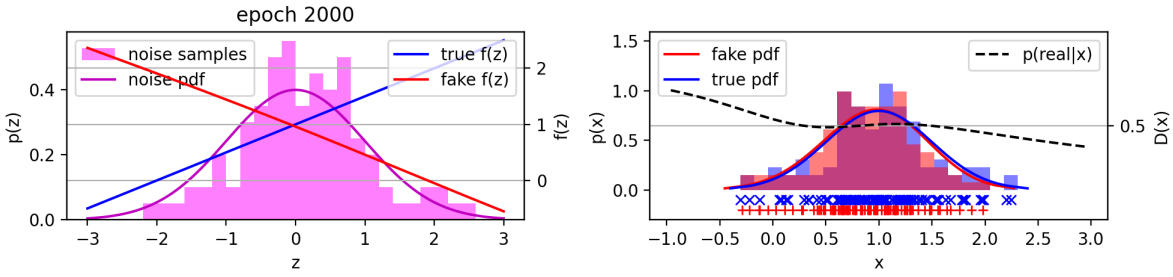
In [71]:
```
epochs['figs'][3]
```

Out[71]:



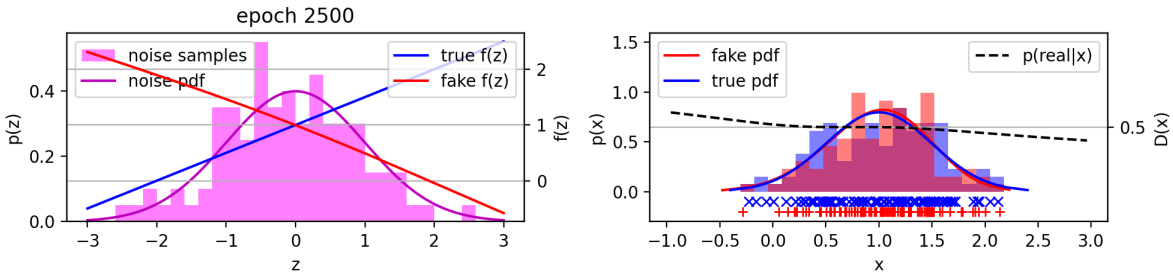
In [72]:
```
epochs['figs'][4]
```

Out[72]:



In [73]:
```
epochs['figs'][5]
```

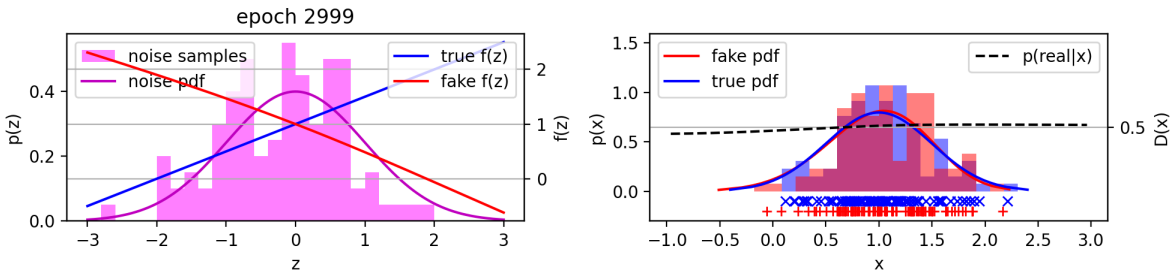
Out[73]:



- Converges to the true transformation.
  - discriminator can't tell the difference

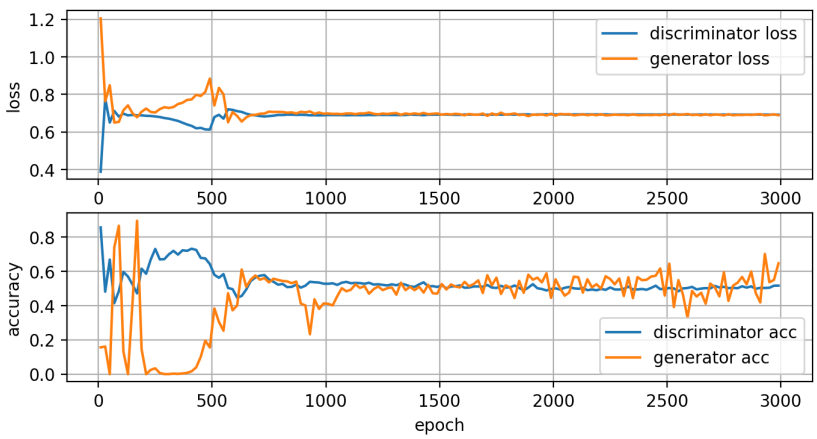
In [74]:
```
epochs['figs'][6]
```

Out[74]:



- loss/accuracy during training.
  - the values will oscillate, since the two networks are playing against each other.
  - at convergence, both G & D obtain 50% accuracy (random chance).

In [75]:
```
plot_epochs(epochs)
```



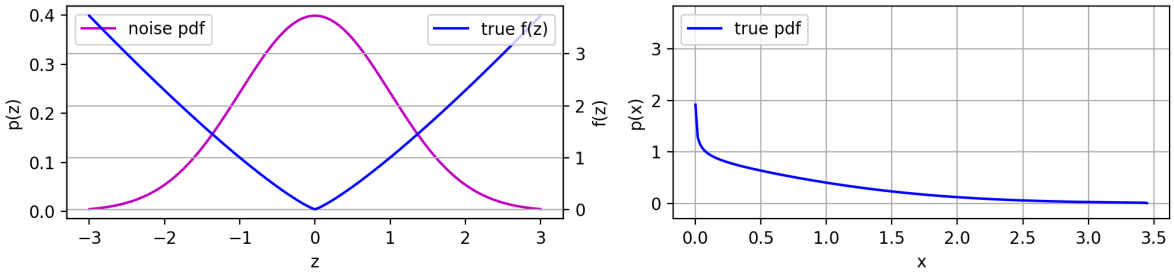
Another example

- distribution with a long-tail

In [76]:
```
xform = lambda x: abs(x)**1.2
sample_range = get_bounds(xform)
```

In [77]:
```
plot_epoch(onlygt=True)
```

Out[77]:



- build the GAN

In [78]:
```
K.clear_session()
random.seed(5489); tf.random.set_seed(5489)
noise = Input(shape=[1])
generator = gen(10, activation='relu')
g = generator(noise)
discriminator = disc(10, activation='relu')
d = discriminator(g)
make_trainable(discriminator, False)
GAN = Model(noise, d)
GAN.compile(loss="binary_crossentropy", optimizer="sgd", metrics=['accuracy'])
```

- train for 3000 epochs

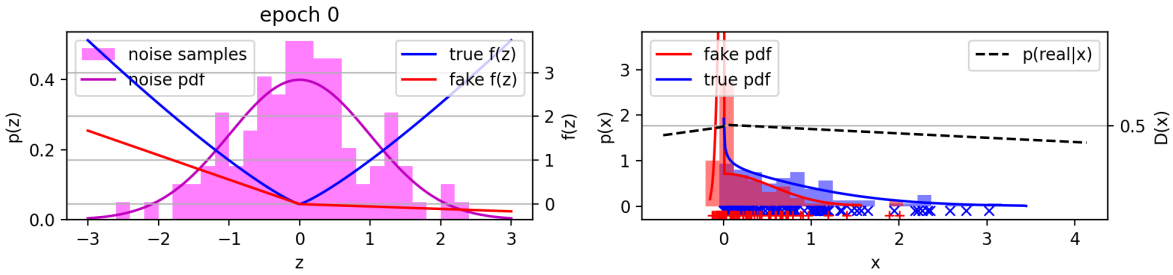
In [79]:
```
epochs = train_for_n(nb_epoch=3000, lr=0.01, plt_frq=500)
```

```
2023-01-25 20:30:24.777467: I tensorflow/core/common_runtime/pluggable_device/plug
gable_device_factory.cc:305] Could not identify NUMA node of platform GPU ID 0, de
faulting to 0. Your kernel may not have been built with NUMA support.
2023-01-25 20:30:24.777488: I tensorflow/core/common_runtime/pluggable_device/plug
gable_device_factory.cc:271] Created TensorFlow device (/job:localhost/replica:0/t
ask:0/device:GPU:0 with 0 MB memory) -> physical PluggableDevice (device: 0, name:
METAL, pci bus id: <undefined>)
2023-01-25 20:30:24.782135: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
2023-01-25 20:30:24.789004: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
2023-01-25 20:30:24.823720: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
2023-01-25 20:30:24.846783: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
2023-01-25 20:30:24.850821: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
2023-01-25 20:30:24.853938: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
2023-01-25 20:30:24.877972: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
2023-01-25 20:30:24.882310: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
2023-01-25 20:30:24.884685: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
2023-01-25 20:30:24.964359: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
2023-01-25 20:30:25.074930: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
2023-01-25 20:30:25.080190: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
2023-01-25 20:30:25.083465: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
2023-01-25 20:30:25.110549: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
2023-01-25 20:30:25.116362: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
```

```
2023-01-25 20:30:25.118978: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
2023-01-25 20:30:25.216480: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
2023-01-25 20:30:25.424121: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
```

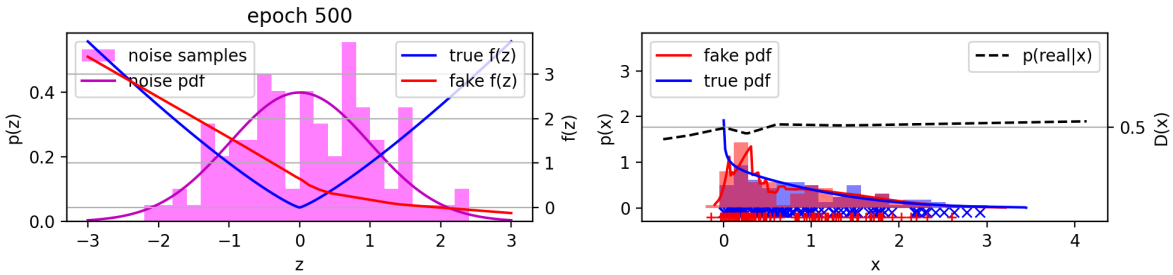
In [80]:
```
epochs['figs'][0]
```

Out[80]:



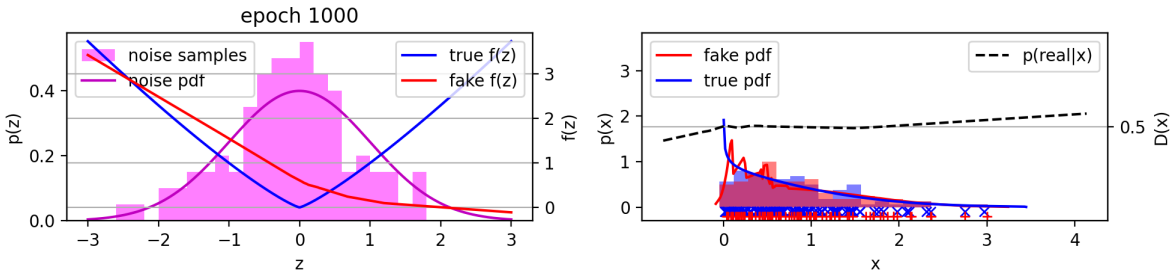
In [81]:
```
epochs['figs'][1]
```

Out[81]:



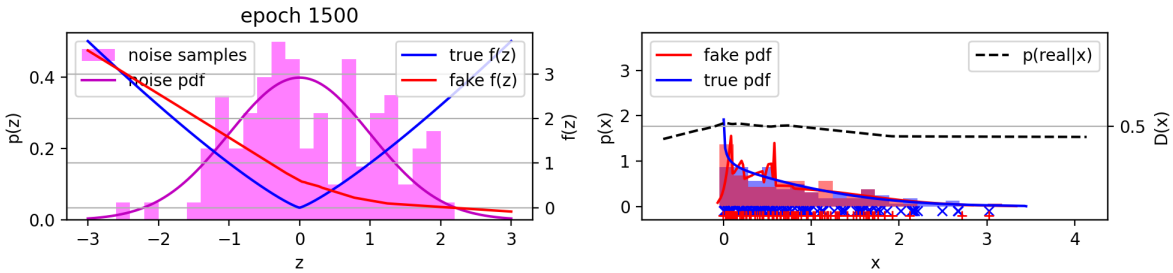
In [82]:
```
epochs['figs'][2]
```

Out[82]:



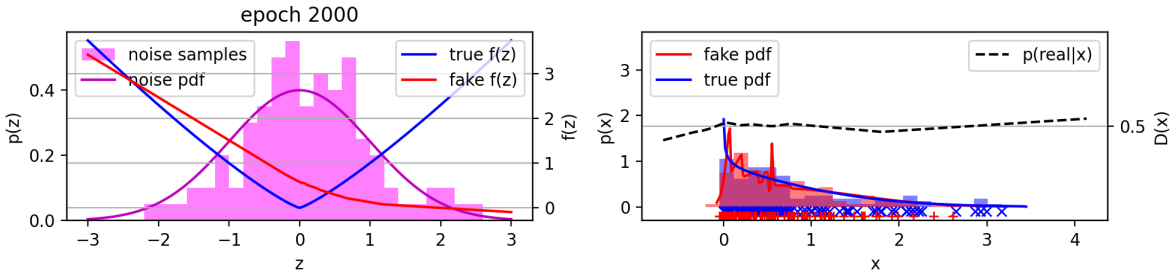
In [83]:
```
epochs['figs'][3]
```

Out[83]:



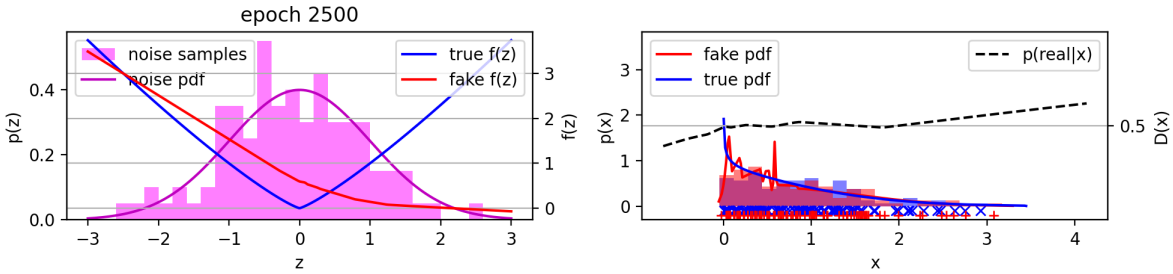
In [84]:
```
epochs['figs'][4]
```

Out[84]:



In [85]:
```
epochs['figs'][5]
```

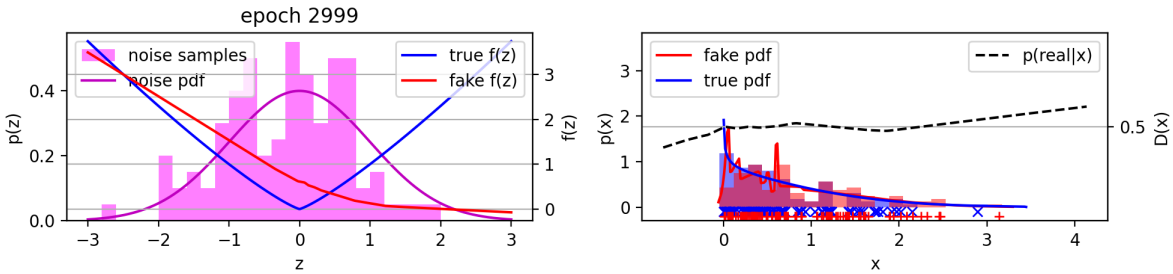
Out[85]:



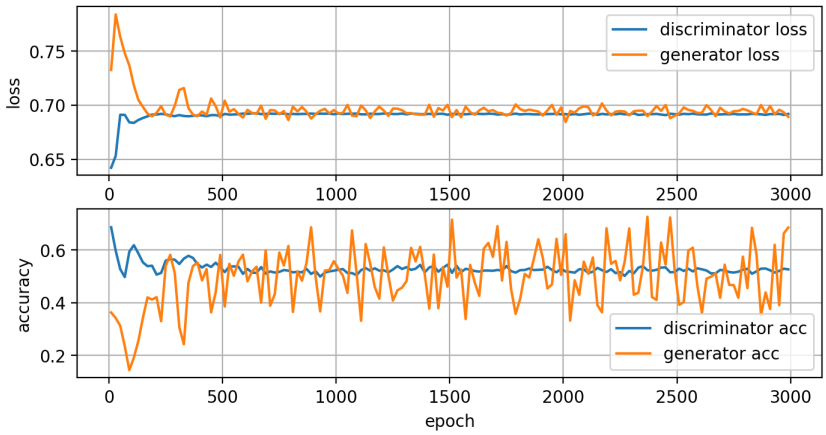
- converges to an equivalent transformation to the real one.

In [86]:
```
epochs['figs'][6]
```

Out[86]:



In [87]:
```
plot_epochs(epochs)
```



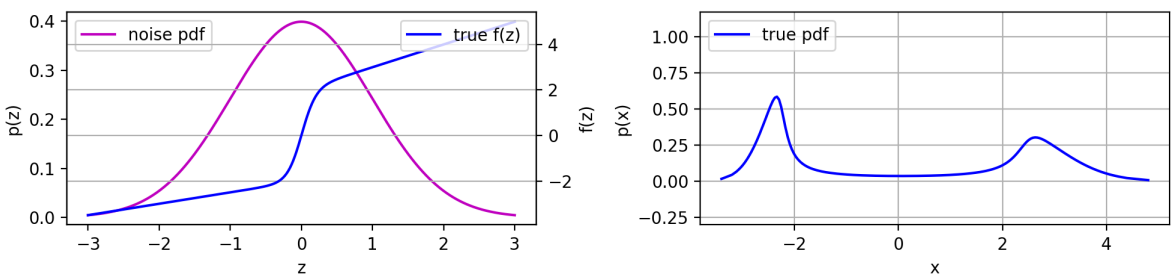
Mode collapse problem

- GANs also work to learn multi-modal distributions

In [88]:
```
# two modes
sigmoid = lambda x,a: 1/(1+exp(-a*x))
xform = lambda x: (0.5*x-2)*sigmoid(-x,10) + (x+2)*sigmoid(x,10)
sample_range = get_bounds(xform)
```

In [89]:
```
plot_epoch(onlygt=True)
```

Out[89]:



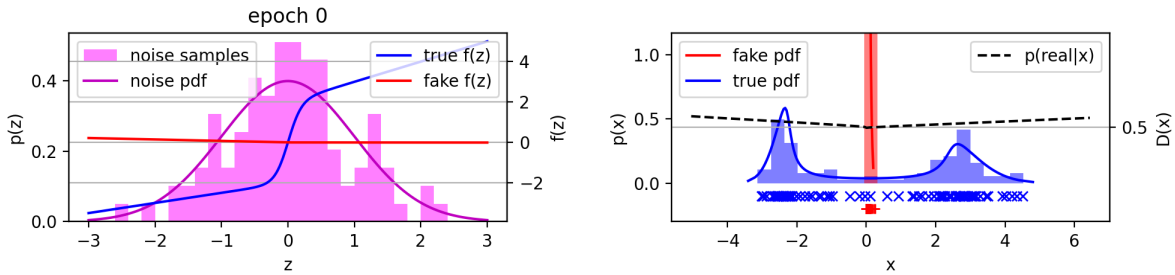
- train 3000 epochs

In [91]:
```
epochs = train_for_n(nb_epoch=3000, lr=0.01, plt_frq=500)
```

```
2023-01-25 20:36:01.060915: I tensorflow/core/common_runtime/pluggable_device/plug
gable_device_factory.cc:305] Could not identify NUMA node of platform GPU ID 0, de
faulting to 0. Your kernel may not have been built with NUMA support.
2023-01-25 20:36:01.060939: I tensorflow/core/common_runtime/pluggable_device/plug
gable_device_factory.cc:271] Created TensorFlow device (/job:localhost/replica:0/t
ask:0/device:GPU:0 with 0 MB memory) -> physical PluggableDevice (device: 0, name:
METAL, pci bus id: <undefined>)
2023-01-25 20:36:01.065167: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
2023-01-25 20:36:01.071950: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
2023-01-25 20:36:01.105224: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
2023-01-25 20:36:01.132430: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
2023-01-25 20:36:01.136558: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
2023-01-25 20:36:01.139555: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
2023-01-25 20:36:01.165359: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
2023-01-25 20:36:01.170034: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
2023-01-25 20:36:01.172477: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
2023-01-25 20:36:01.266105: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
2023-01-25 20:36:01.388049: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
2023-01-25 20:36:01.394342: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
2023-01-25 20:36:01.397976: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
2023-01-25 20:36:01.427604: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
2023-01-25 20:36:01.434035: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
2023-01-25 20:36:01.437259: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
2023-01-25 20:36:01.548121: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
2023-01-25 20:36:01.653903: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
```

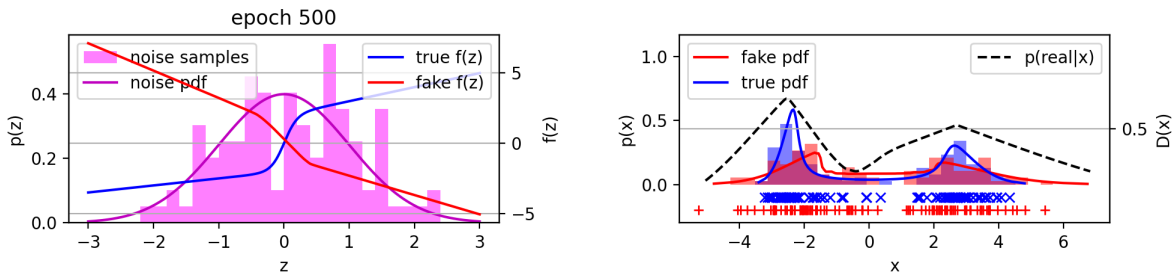
In [92]:
```
epochs['figs'][0]
```

Out[92]:



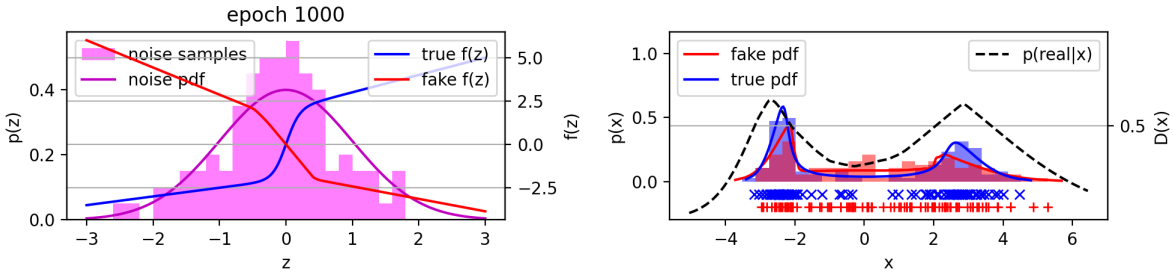
In [93]:
```
epochs['figs'][1]
```

Out[93]:



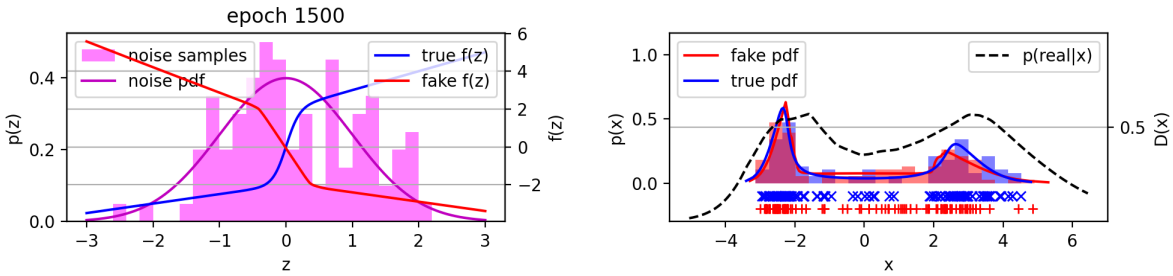
In [94]:
```
epochs['figs'][2]
```

Out[94]:



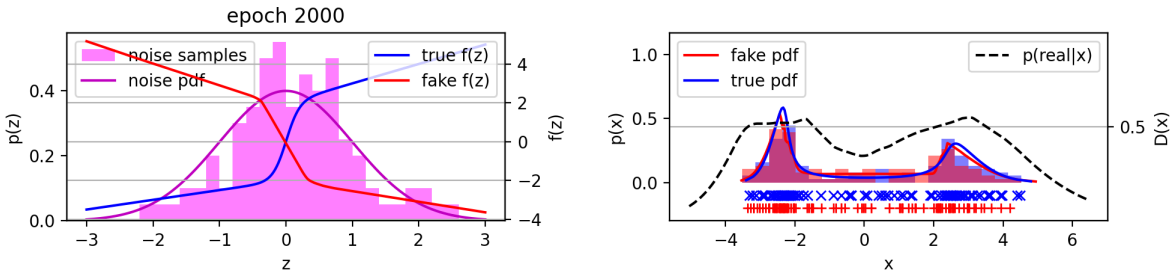
In [95]:
```
epochs['figs'][3]
```

Out[95]:



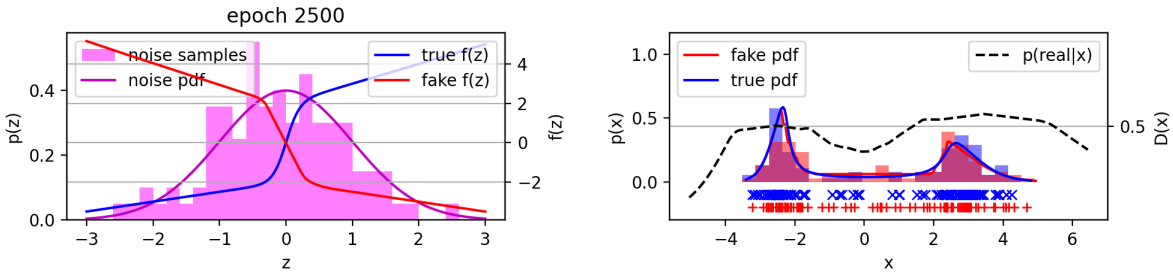
In [96]:
```
epochs['figs'][4]
```

Out[96]:



In [97]:
```
epochs['figs'][5]
```

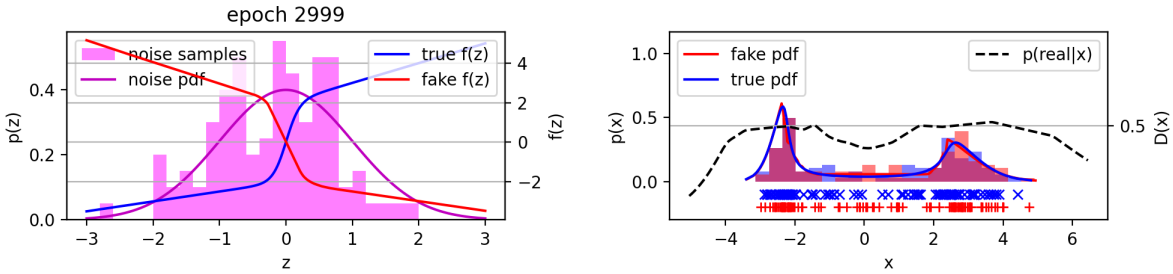
Out[97]:



- represents both modes of the target density
- converges to true transformation

In [98]:
```
epochs['figs'][6]
```

Out[98]:

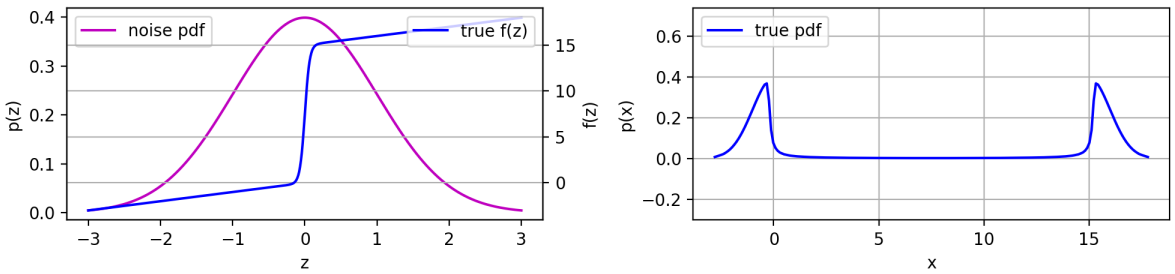


- try another example
  - modes are farther apart

In [99]:
```
# two modes
sigmoid = lambda x,a: 1/(1+exp(-a*x))
xform = lambda x: (x)*sigmoid(-x,30) + (x+15)*sigmoid(x,30)
sample_range = get_bounds(xform)
```

In [100]:
```
plot_epoch(onlygt=True)
```

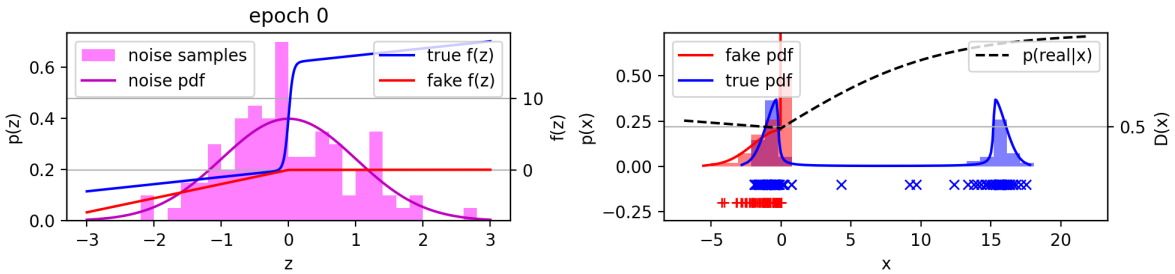
Out[100]:



- G collapses to sample from only one of the modes
  - G can always fool D with this one mode

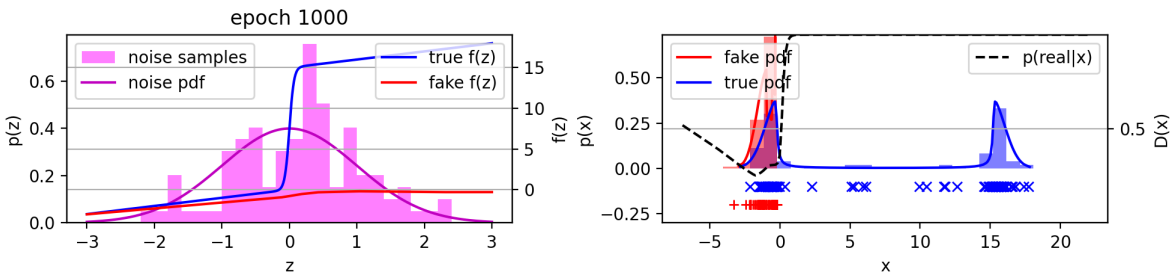
In [103]:
```
epochs['figs'][0]
```

Out[103]:



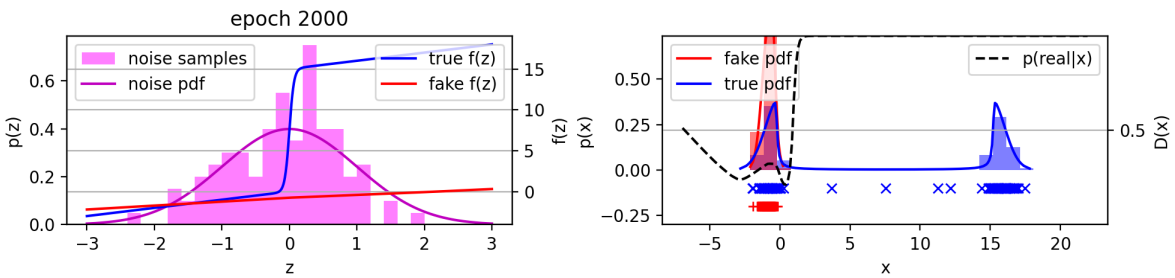
In [104]:
```
epochs['figs'][1]
```

Out[104]:



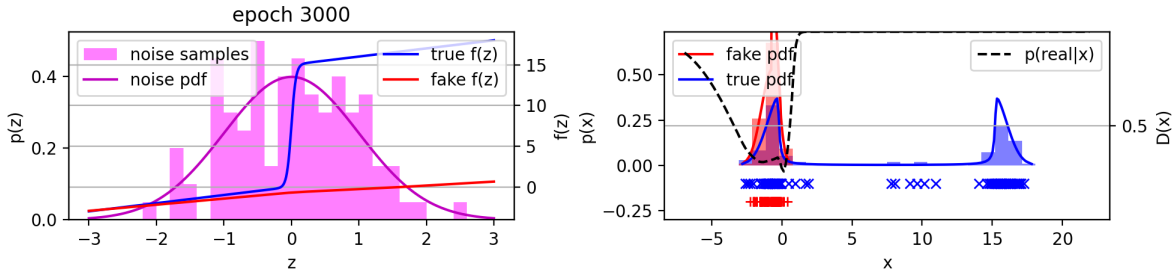
In [105]:
```
epochs['figs'][2]
```

Out[105]:



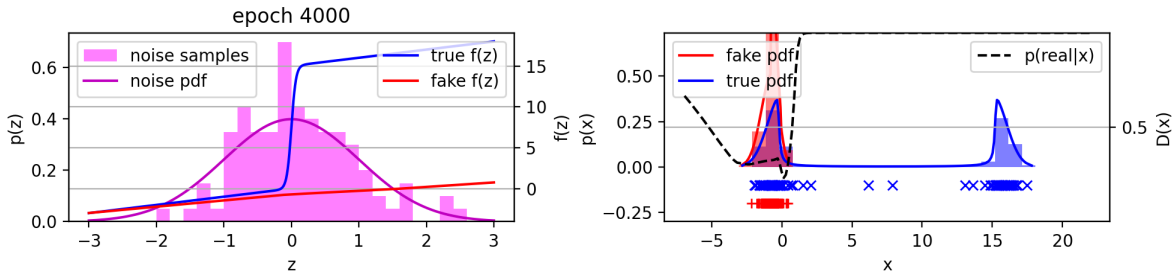
In [106]:
```
epochs['figs'][3]
```

Out[106]:



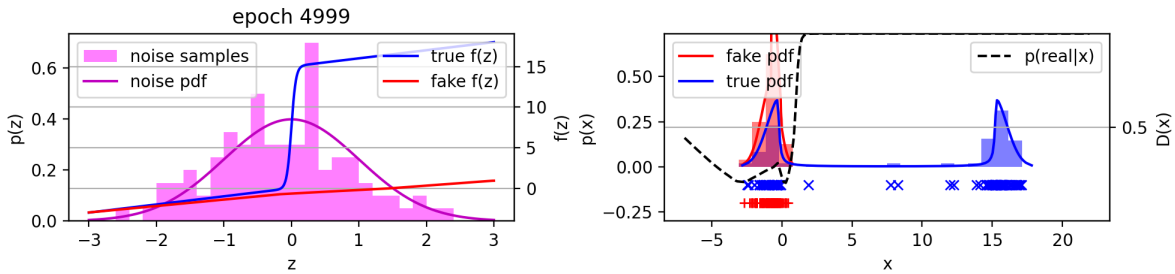
In [107]:
```
epochs['figs'][4]
```

Out[107]:



In [108]:
```
epochs['figs'][5]
```

Out[108]:

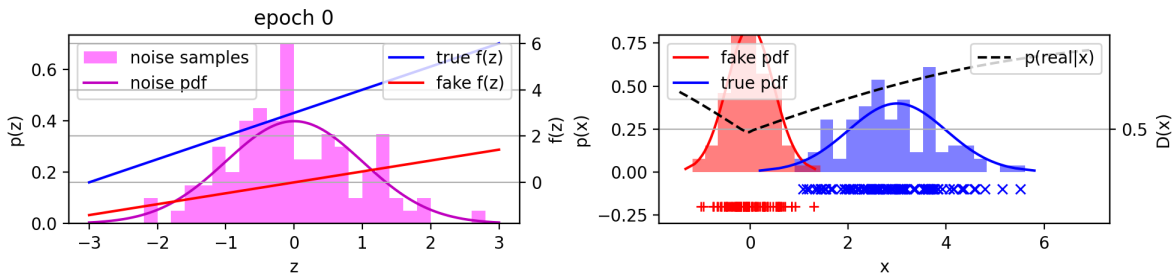


- **Mode collapse problem**
  - G samples from one mode.
  - when D catches this, G switches to another mode
  - and so on

example from beginning

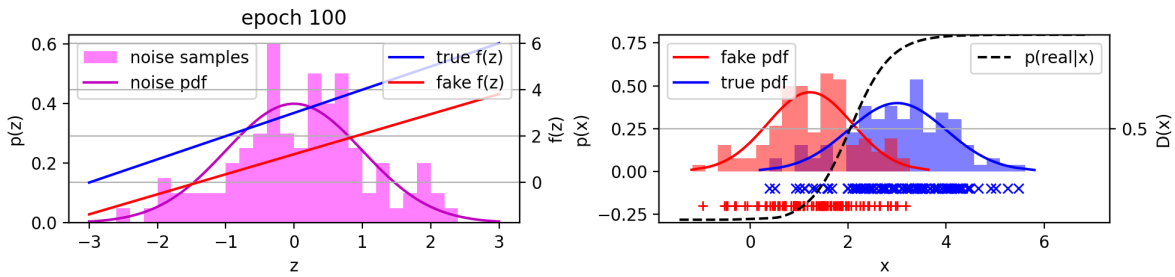
In [111]:
```
epochs['figs'][0]
```

Out[111]:



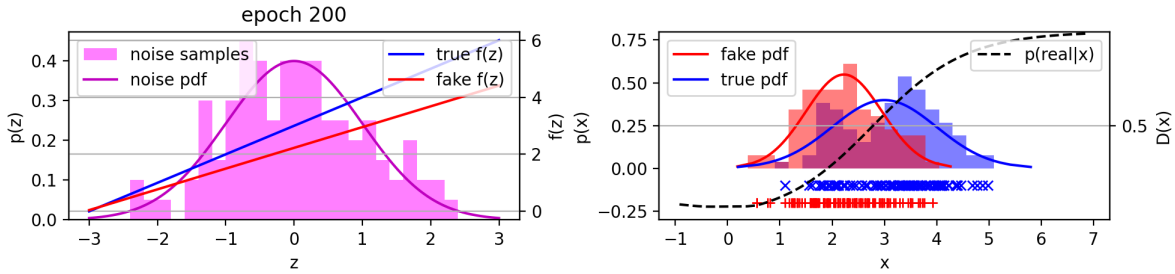
In [112]:
```
epochs['figs'][1]
```

Out[112]:



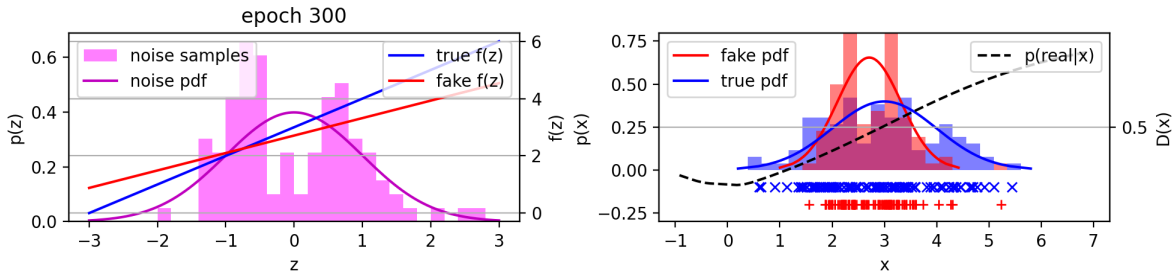
In [113]:
```
epochs['figs'][2]
```

Out[113]:



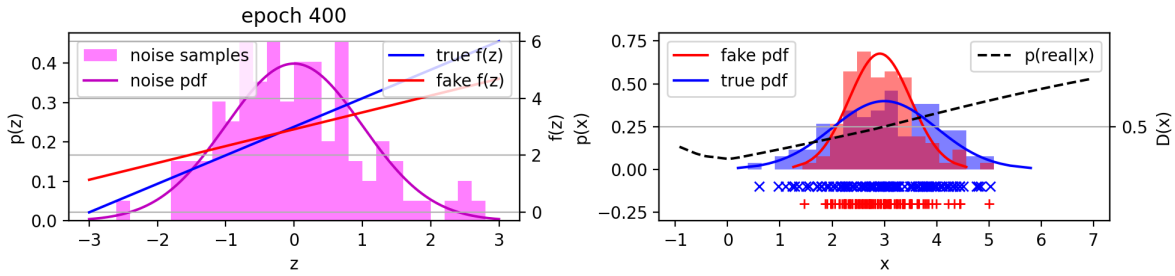
In [114]:
```
epochs['figs'][3]
```

Out[114]:



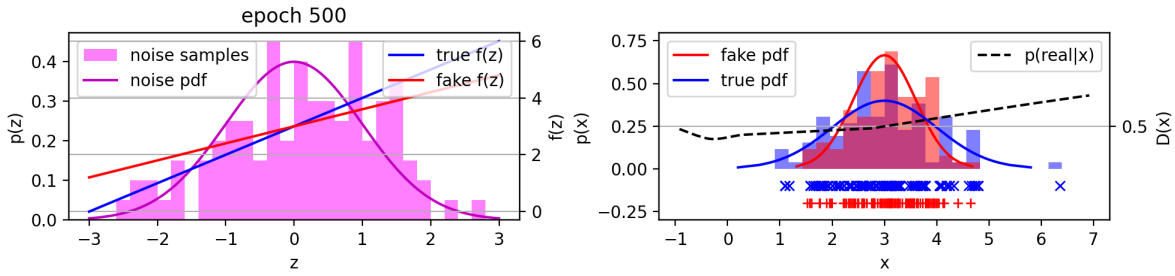
In [115]:
```
epochs['figs'][4]
```

Out[115]:



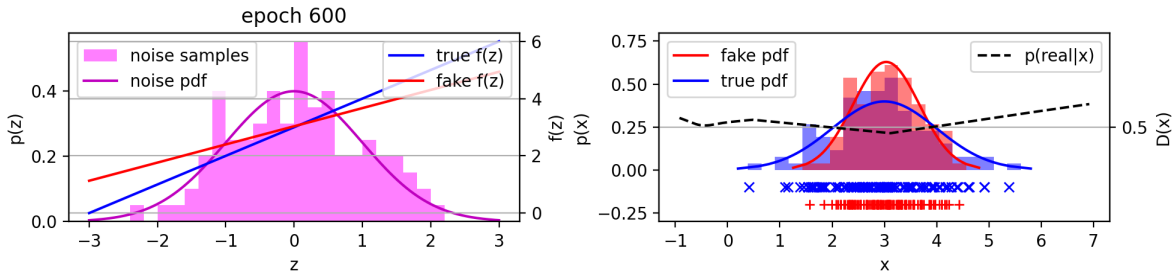
In [116]:
```
epochs['figs'][5]
```

Out[116]:



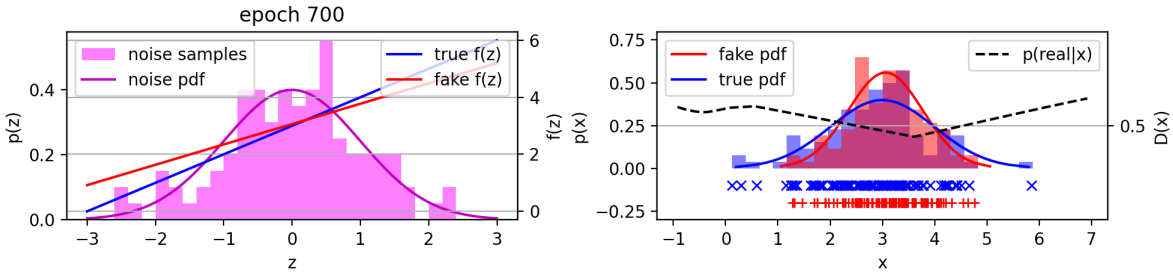
In [117]:
```
epochs['figs'][6]
```

Out[117]:



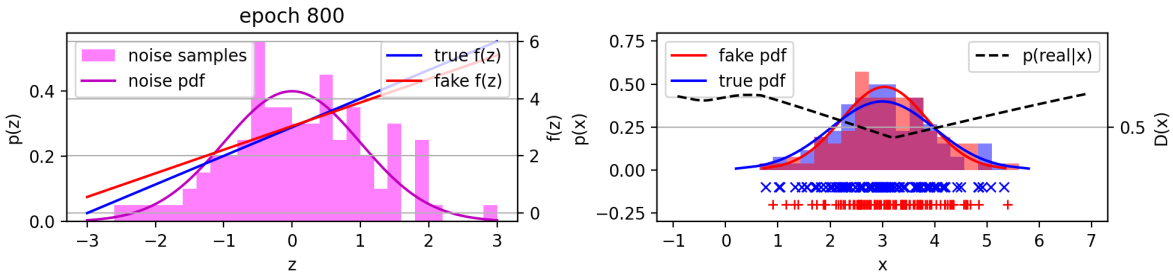
In [118]:
```
epochs['figs'][7]
```

Out[118]:



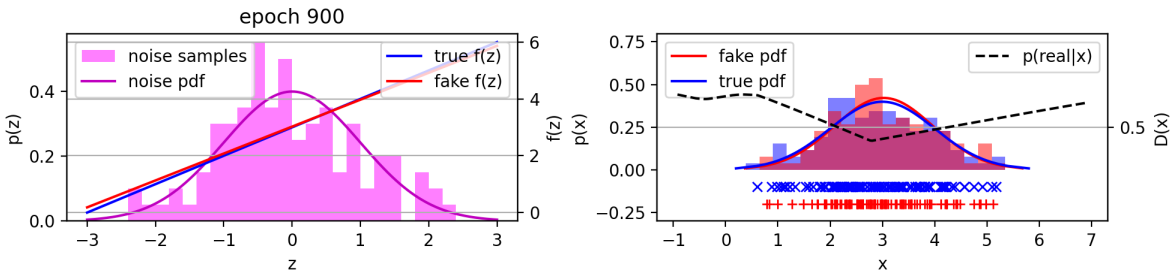
In [119]:
```
epochs['figs'][8]
```

Out[119]:



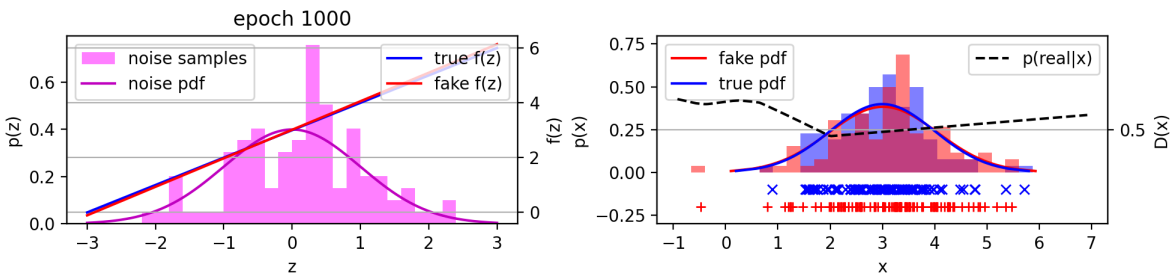
In [120]:
```
epochs['figs'][9]
```

Out[120]:



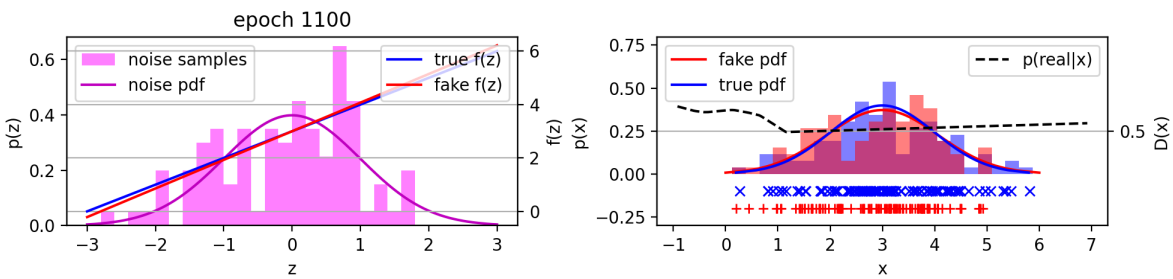
In [121]:
```
epochs['figs'][10]
```

Out[121]:



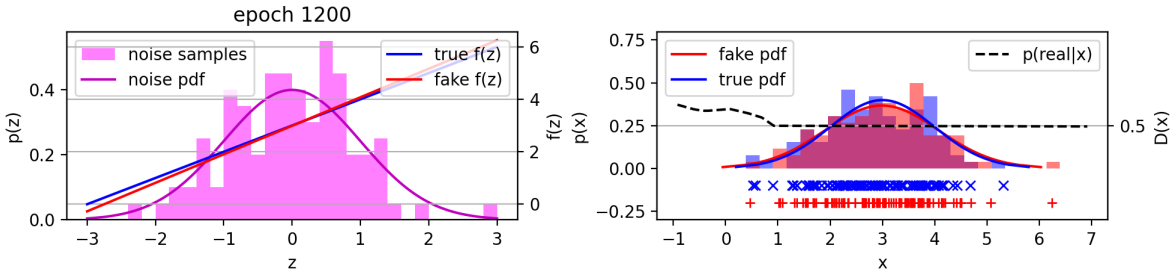
In [122]:
```
epochs['figs'][11]
```

Out[122]:



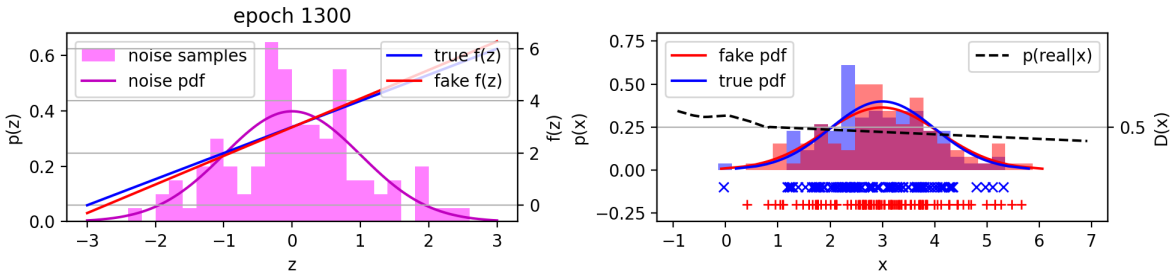
In [123]:
```
epochs['figs'][12]
```

Out[123]:



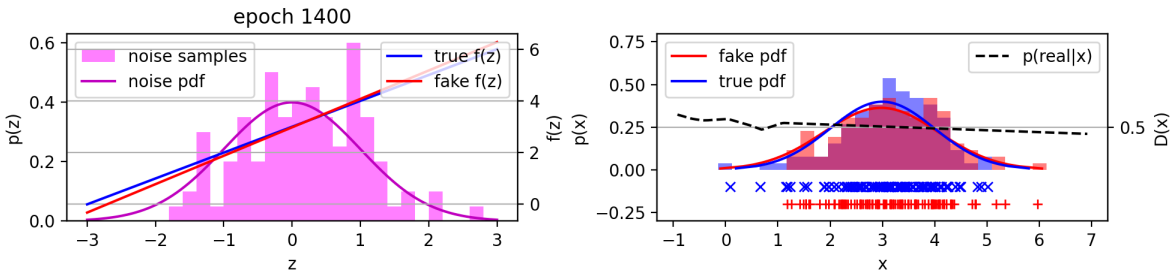
In [124]:
```
epochs['figs'][13]
```

Out[124]:



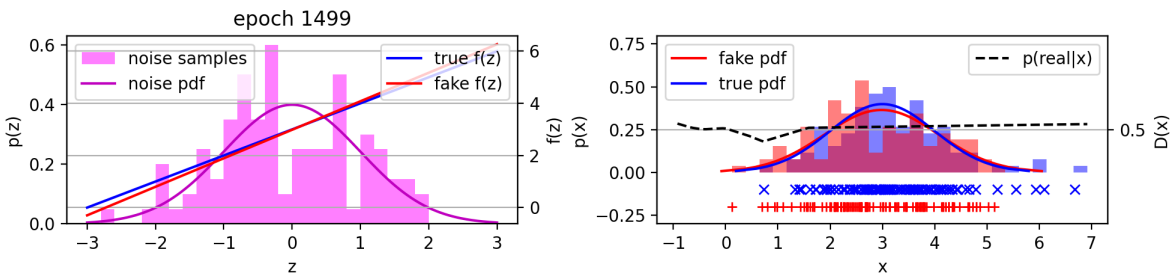
In [125]:
```
epochs['figs'][14]
```

Out[125]:



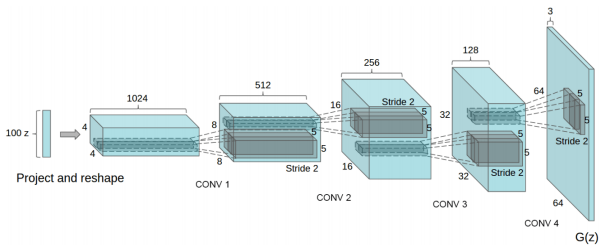
In [126]:
```
epochs['figs'][15]
```

Out[126]:



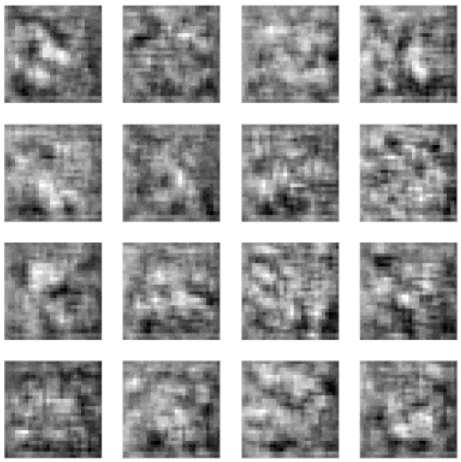
Example on MNIST

- use a deep convolutional network as the generator (Source: post, code)

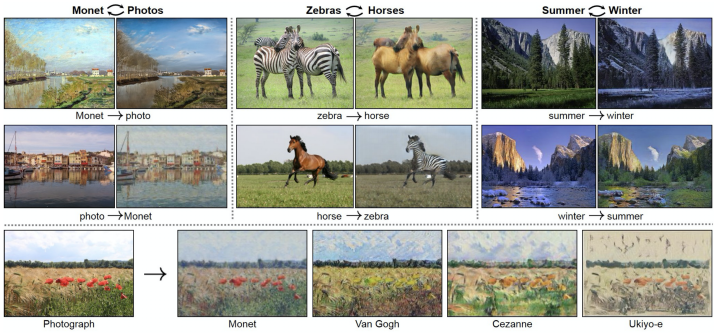


- generated images

- generated examples during training



Example: Style transfer (CycleGAN)



Example: Face Synthesis

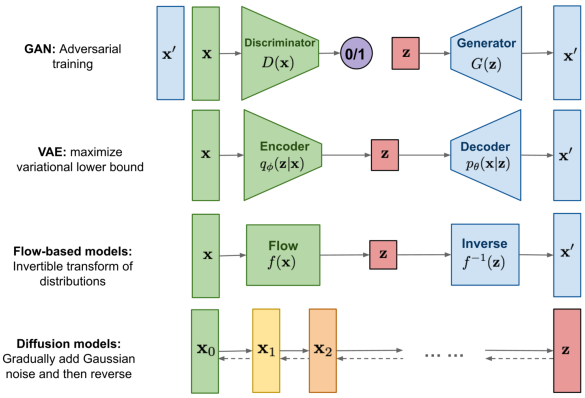
- Which faces are fake?

- Try it out: https://thispersondoesnotexist.com

Diffusion Models

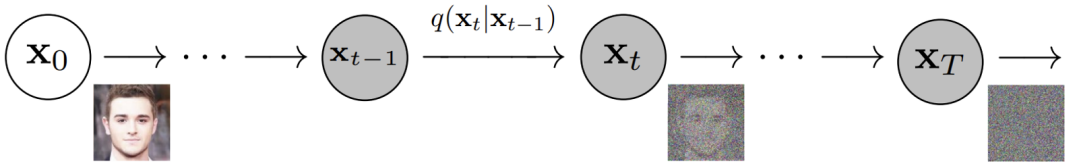
- Define a forward diffusion process that gradually turns an image into noise.
- Learn the reverse process that can gradually remove the noise and recover the image.

- key difference: supervision on each step.



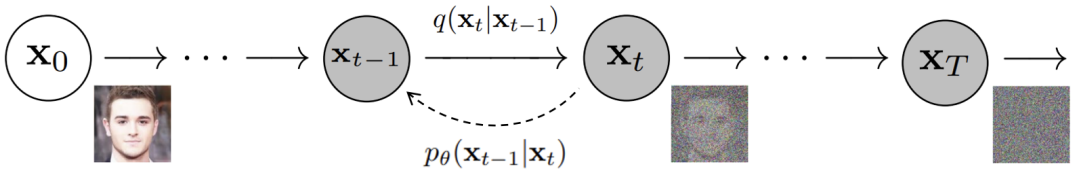
Forward diffusion process

- In each step, fade the image and add Gaussian noise.
  - $\mathbf{x_0}$ = image
  - $\mathbf{x_t} = \sqrt{1-\beta_t}\mathbf{x}_{t-1} + \sqrt{\beta_t}\epsilon_t, \epsilon_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
  - Noise variance schedule: $\{\beta_1, \cdots, \beta_T\}$
- Probability distributions
  - for one step: $q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t|\sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I})$
  - for all steps: $q(\mathbf{x}_{1:T}|\mathbf{x}_0) = \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1})$



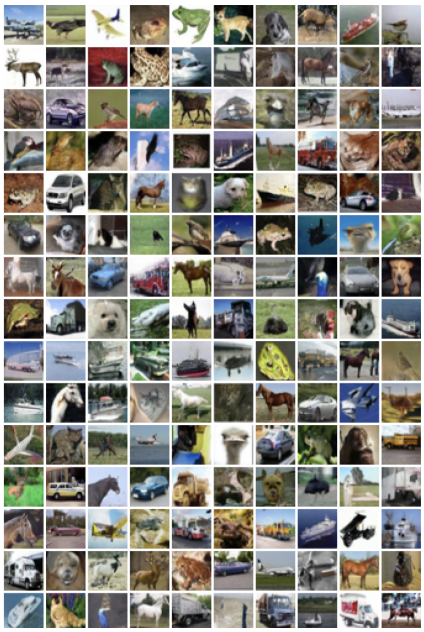
Reverse diffusion process

- We want recover the image from the noise.
  - the true reverse process $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$ is not tractable.
- Approximate it with a parametric model (NN) $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$
  - assume Gaussian: $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}|\mu_\theta(\mathbf{x}_t, t), \Sigma_\theta(\mathbf{x}_t, t))$
  - for all steps: $p(\mathbf{x}_{0:T}) = p_\theta(\mathbf{x}_T)\prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$



Training

- minimize an upper-bound on the negative data log-likelihood.
  - $\mathbb{E}[-\log p_\theta(\mathbf{x}_0)] \leq \mathbb{E}_q\left[-\log \frac{p_\theta(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\right] = \sum_{t=0}^{T} L_t$
  - $L_0 = -\log p_\theta(\mathbf{x}_0|\mathbf{x}_1)$
  - $L_{t-1} = KL(q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0), p(\mathbf{x}_{t-1}|\mathbf{x}_t))$
  - $L_T = KL(q(\mathbf{x}_T|\mathbf{x}_0, p(\mathbf{x}_T))$
- Usually use a U-net to model $\mu_\theta, \Sigma_\theta$.

Summary

- **Unsupervised Learning**
  - Autoencoder - unsupervised dimensionality reduction and clustering.
  - Convolutional autoencoder - AE for images.
  - Masked Autoencoder (MAE) - self-supervised representation learning
  - Variational autoencoder (VAE) - improve interpolation ability, generative model
  - Generative Adversarial Networks (GAN) - learn to generate samples from a distribution
  - Diffusion model - transform noise into image, step by step.