Continual Zero-Shot Learning

Ivan Skorokhodov

April 21, 2020

What is Continual Learning?

What is Continual Learning?

- Modern neural networks are prone to *catastrophic forgetting*: they forget previous tasks while are learning new ones.

What is Continual Learning?

- Modern neural networks are prone to *catastrophic forgetting*: they forget previous tasks while are learning new ones.
- Continual Learning tries to find ways to make model learn skills one by one in such a way that we do not forget previous skills

What is Continual Learning?

- Modern neural networks are prone to *catastrophic forgetting*: they forget previous tasks while are learning new ones.
- Continual Learning tries to find ways to make model learn skills one by one in such a way that we do not forget previous skills
  - Example 1: a robot that travels the world and learn new skills. We want it not to forget previous skills while he is acquiring new ones.

What is Continual Learning?

- Modern neural networks are prone to *catastrophic forgetting*: they forget previous tasks while are learning new ones.
- Continual Learning tries to find ways to make model learn skills one by one in such a way that we do not forget previous skills
  - Example 1: a robot that travels the world and learn new skills. We want it not to forget previous skills while he is acquiring new ones.
  - Example 2: a classification model is learning datasets one by one: we do not want its performance on previously learned datasets to decrease.

Modern Continual Learning techniques

Modern Continual Learning techniques

Modern CL techniques can be divided into three groups:

Modern Continual Learning techniques

Modern CL techniques can be divided into three groups:

- Regularization-based ([4], [1], etc): detect the weights which are important for previous tasks and do not change them much in the future.

Modern Continual Learning techniques

Modern CL techniques can be divided into three groups:

- Regularization-based ([4], [1], etc): detect the weights which are important for previous tasks and do not change them much in the future.
- Rehearsal-based ([2], [6], etc): store a part of previous data to replay it in the future.

Modern Continual Learning techniques

Modern CL techniques can be divided into three groups:

- Regularization-based ([4], [1], etc): detect the weights which are important for previous tasks and do not change them much in the future.
- Rehearsal-based ([2], [6], etc): store a part of previous data to replay it in the future.
- Component-based ([5], [3], etc): divide your network into components, and let future tasks not to break components which are important for previous tasks.

What is Zero-Shot Learning (ZSL)?

What data do we have:

What is Zero-Shot Learning (ZSL)?

What data do we have:

- We will consider classification tasks from now on...

What is Zero-Shot Learning (ZSL)?

What data do we have:

- We will consider classification tasks from now on...
- For each class $c \in \mathcal{C}$ we are given an *attribute vector* $a_c \in \mathcal{A}$ which describes the class:

What is Zero-Shot Learning (ZSL)?

What data do we have:

- We will consider classification tasks from now on...
- For each class $c \in \mathcal{C}$ we are given an *attribute vector* $a_c \in \mathcal{A}$ which describes the class:
  - Imagine that we are classifying birds

What is Zero-Shot Learning (ZSL)?

What data do we have:

- We will consider classification tasks from now on...
- For each class $c \in \mathcal{C}$ we are given an *attribute vector* $a_c \in \mathcal{A}$ which describes the class:
  - Imagine that we are classifying birds
  - Then for each bird a_c includes bird's characteristics: color of a tail, body size, length of a beak, etc

What is Zero-Shot Learning (ZSL)?

What data do we have:

- We will consider classification tasks from now on...
- For each class $c \in \mathcal{C}$ we are given an *attribute vector* $a_c \in \mathcal{A}$ which describes the class:
  - Imagine that we are classifying birds
  - Then for each bird a_c includes bird's characteristics: color of a tail, body size, length of a beak, etc
- All classes are divided into *seen* and *unseen*:

What is Zero-Shot Learning (ZSL)?

What data do we have:

- We will consider classification tasks from now on...
- For each class $c \in \mathcal{C}$ we are given an *attribute vector* $a_c \in \mathcal{A}$ which describes the class:
  - Imagine that we are classifying birds
  - Then for each bird a_c includes bird's characteristics: color of a tail, body size, length of a beak, etc
- All classes are divided into *seen* and *unseen*:
  - Seen dataset: $D^s = \{X^s, Y^s, A^s\}$

What is Zero-Shot Learning (ZSL)?

What data do we have:

- We will consider classification tasks from now on...
- For each class $c \in \mathcal{C}$ we are given an *attribute vector* $a_c \in \mathcal{A}$ which describes the class:
  - Imagine that we are classifying birds
  - Then for each bird a_c includes bird's characteristics: color of a tail, body size, length of a beak, etc
- All classes are divided into *seen* and *unseen*:
  - Seen dataset: $D^s = \{X^{\mathrm{s}}, Y^{\mathrm{s}}, A^{\mathrm{s}}\}$
  - Unseen dataset: $D^u = \{X^{\mathrm{u}}, Y^{\mathrm{u}}, A^{\mathrm{u}}\}$

What is Zero-Shot Learning (ZSL)?

What data do we have:

- We will consider classification tasks from now on...
- For each class $c \in \mathcal{C}$ we are given an *attribute vector* $a_c \in \mathcal{A}$ which describes the class:
  - Imagine that we are classifying birds
  - Then for each bird a_c includes bird's characteristics: color of a tail, body size, length of a beak, etc
- All classes are divided into *seen* and *unseen*:
  - Seen dataset: $D^s = \{X^{\mathrm{s}}, Y^{\mathrm{s}}, A^{\mathrm{s}}\}$
  - Unseen dataset: $D^u = \{X^{\mathrm{u}}, Y^{\mathrm{u}}, A^{\mathrm{u}}\}$
- During training we have an access only to seen dataset D^s.

What is Zero-Shot Learning (ZSL)?

What data do we have:

- We will consider classification tasks from now on...
- For each class $c \in \mathcal{C}$ we are given an *attribute vector* $a_c \in \mathcal{A}$ which describes the class:
  - Imagine that we are classifying birds
  - Then for each bird a_c includes bird's characteristics: color of a tail, body size, length of a beak, etc
- All classes are divided into *seen* and *unseen*:
  - Seen dataset: $D^s = \{X^{\text{s}}, Y^{\text{s}}, A^{\text{s}}\}$
  - Unseen dataset: $D^u = \{X^{\text{u}}, Y^{\text{u}}, A^{\text{u}}\}$
- During training we have an access only to seen dataset D^s.
  - Our goal is to learn to match images with class descriptions

What is Zero-Shot Learning (ZSL)?

What data do we have:

- We will consider classification tasks from now on...
- For each class $c \in \mathcal{C}$ we are given an *attribute vector* $a_c \in \mathcal{A}$ which describes the class:
  - Imagine that we are classifying birds
  - Then for each bird a_c includes bird's characteristics: color of a tail, body size, length of a beak, etc
- All classes are divided into *seen* and *unseen*:
  - Seen dataset: $D^s = \{X^{\text{s}}, Y^{\text{s}}, A^{\text{s}}\}$
  - Unseen dataset: $D^u = \{X^{\text{u}}, Y^{\text{u}}, A^{\text{u}}\}$
- During training we have an access only to seen dataset D^s.
  - Our goal is to learn to match images with class descriptions
  - I.e. model learns to detect "blue tails", "large heads", "short beaks", etc and not only concrete bird species

What is Zero-Shot Learning (ZSL)?

What data do we have:

- We will consider classification tasks from now on...
- For each class $c \in \mathcal{C}$ we are given an *attribute vector* $a_c \in \mathcal{A}$ which describes the class:
  - Imagine that we are classifying birds
  - Then for each bird a_c includes bird's characteristics: color of a tail, body size, length of a beak, etc
- All classes are divided into *seen* and *unseen*:
  - Seen dataset: $D^s = \{X^\text{s}, Y^\text{s}, A^\text{s}\}$
  - Unseen dataset: $D^u = \{X^\text{u}, Y^\text{u}, A^\text{u}\}$
- During training we have an access only to seen dataset D^s.
  - Our goal is to learn to match images with class descriptions
  - I.e. model learns to detect "blue tails", "large heads", "short beaks", etc and not only concrete bird species
- At test time we evaluate model performance on unseen dataset D^u

What is Zero-Shot Learning (ZSL)?

What data do we have:

- We will consider classification tasks from now on...
- For each class $c \in \mathcal{C}$ we are given an *attribute vector* $a_c \in \mathcal{A}$ which describes the class:
  - Imagine that we are classifying birds
  - Then for each bird a_c includes bird's characteristics: color of a tail, body size, length of a beak, etc
- All classes are divided into *seen* and *unseen*:
  - Seen dataset: $D^s = \{X^{\text{s}}, Y^{\text{s}}, A^{\text{s}}\}$
  - Unseen dataset: $D^u = \{X^{\text{u}}, Y^{\text{u}}, A^{\text{u}}\}$
- During training we have an access only to seen dataset D^s.
  - Our goal is to learn to match images with class descriptions
  - I.e. model learns to detect "blue tails", "large heads", "short beaks", etc and not only concrete bird species
- At test time we evaluate model performance on unseen dataset D^u
- Using the knowledge about how inputs and attributes correspond to each other we can detect birds that we have not seen before just based on their class description a_c.

Modern ZSL techniques (for classification)

Modern ZSL techniques (for classification)

- Embedding-based: build two embedder models:

Modern ZSL techniques (for classification)

- Embedding-based: build two embedder models:
  - Model $f(x)$ to embed images

Modern ZSL techniques (for classification)

- Embedding-based: build two embedder models:
  - Model $f(x)$ to embed images
  - Model $h(a)$ to embed attributes

Modern ZSL techniques (for classification)

- Embedding-based: build two embedder models:
  - Model $f(x)$ to embed images
  - Model $h(a)$ to embed attributes
  - Compute classification logits just by computing $d(f(x), h(a_c))$ for each class c (here d is some distance function.

Modern ZSL techniques (for classification)

- Embedding-based: build two embedder models:
  - Model $f(x)$ to embed images
  - Model $h(a)$ to embed attributes
  - Compute classification logits just by computing $d(f(x), h(a_c))$ for each class c (here d is some distance function.
  - I.e. we just measure distance between an image embedding and attribute embeddings

Modern ZSL techniques (for classification)

- Embedding-based: build two embedder models:
  - Model $f(x)$ to embed images
  - Model $h(a)$ to embed attributes
  - Compute classification logits just by computing $d(f(x), h(a_c))$ for each class c (here d is some distance function.
  - I.e. we just measure distance between an image embedding and attribute embeddings
  - Challenges: how to embed images properly, how to embed attributes properly, what distance function to use, etc

Modern ZSL techniques (for classification)

- Embedding-based: build two embedder models:
  - Model $f(x)$ to embed images
  - Model $h(a)$ to embed attributes
  - Compute classification logits just by computing $d(f(x), h(a_c))$ for each class c (here d is some distance function.
  - I.e. we just measure distance between an image embedding and attribute embeddings
  - Challenges: how to embed images properly, how to embed attributes properly, what distance function to use, etc
- Generative-based

Modern ZSL techniques (for classification)

- Embedding-based: build two embedder models:
  - Model $f(x)$ to embed images
  - Model $h(a)$ to embed attributes
  - Compute classification logits just by computing $d(f(x), h(a_c))$ for each class c (here d is some distance function.
  - I.e. we just measure distance between an image embedding and attribute embeddings
  - Challenges: how to embed images properly, how to embed attributes properly, what distance function to use, etc
- Generative-based
  - Train a conditional generative model that will learn to generate images based on class descriptions

Modern ZSL techniques (for classification)

- Embedding-based: build two embedder models:
    - Model $f(x)$ to embed images
    - Model $h(a)$ to embed attributes
    - Compute classification logits just by computing $d(f(x), h(a_c))$ for each class c (here d is some distance function.
    - I.e. we just measure distance between an image embedding and attribute embeddings
    - Challenges: how to embed images properly, how to embed attributes properly, what distance function to use, etc
- Generative-based
    - Train a conditional generative model that will learn to generate images based on class descriptions
    - I.e. GAN model that can generate a pigeon given description "grey feather, white head, short legs, etc"

Modern ZSL techniques (for classification)

- Embedding-based: build two embedder models:
  - Model $f(x)$ to embed images
  - Model $h(a)$ to embed attributes
  - Compute classification logits just by computing $d(f(x), h(a_c))$ for each class c (here d is some distance function.
  - I.e. we just measure distance between an image embedding and attribute embeddings
  - Challenges: how to embed images properly, how to embed attributes properly, what distance function to use, etc
- Generative-based
  - Train a conditional generative model that will learn to generate images based on class descriptions
  - I.e. GAN model that can generate a pigeon given description "grey feather, white head, short legs, etc"
  - At test time generate a lot of synthetic images, then train a classifier based on this synthetic dataset

Modern ZSL techniques (for classification)

- Embedding-based: build two embedder models:
  - Model $f(x)$ to embed images
  - Model $h(a)$ to embed attributes
  - Compute classification logits just by computing $d(f(x), h(a_c))$ for each class c (here d is some distance function.
  - I.e. we just measure distance between an image embedding and attribute embeddings
  - Challenges: how to embed images properly, how to embed attributes properly, what distance function to use, etc
- Generative-based
  - Train a conditional generative model that will learn to generate images based on class descriptions
  - I.e. GAN model that can generate a pigeon given description "grey feather, white head, short legs, etc"
  - At test time generate a lot of synthetic images, then train a classifier based on this synthetic dataset
  - Challenges: how to train a good conditional generative model?

Modern ZSL techniques (for classification)

- Embedding-based: build two embedder models:
  - Model $f(x)$ to embed images
  - Model $h(a)$ to embed attributes
  - Compute classification logits just by computing $d(f(x), h(a_c))$ for each class c (here d is some distance function.
  - I.e. we just measure distance between an image embedding and attribute embeddings
  - Challenges: how to embed images properly, how to embed attributes properly, what distance function to use, etc
- Generative-based
  - Train a conditional generative model that will learn to generate images based on class descriptions
  - I.e. GAN model that can generate a pigeon given description "grey feather, white head, short legs, etc"
  - At test time generate a lot of synthetic images, then train a classifier based on this synthetic dataset
  - Challenges: how to train a good conditional generative model?
  - Currently performs better than embedding-based approaches

Continual Zero-Shot Learning

Continual Zero-Shot Learning

- Project: let's use class attributes to improve the performance on future tasks (and this also should improve past performance)

Continual Zero-Shot Learning

- Project: let's use class attributes to improve the performance on future tasks (and this also should improve past performance)
- Why?

Continual Zero-Shot Learning

- Project: let's use class attributes to improve the performance on future tasks (and this also should improve past performance)
- Why?
  - A robot should be able to understand things it has never seen before but only heard about.

Continual Zero-Shot Learning

- Project: let's use class attributes to improve the performance on future tasks (and this also should improve past performance)
- Why?
  - A robot should be able to understand things it has never seen before but only heard about.
  - And it shouldn't forget previously seen objects while doing so...

Continual Zero-Shot Learning

- Project: let's use class attributes to improve the performance on future tasks (and this also should improve past performance)
- Why?
  - A robot should be able to understand things it has never seen before but only heard about.
  - And it shouldn't forget previously seen objects while doing so...
  - And the set of attribute descriptions can grow over time...

Continual Zero-Shot Learning

- Project: let's use class attributes to improve the performance on future tasks (and this also should improve past performance)
- Why?
  - A robot should be able to understand things it has never seen before but only heard about.
  - And it shouldn't forget previously seen objects while doing so...
  - And the set of attribute descriptions can grow over time...
  - In some sense, it is "the next" step of ZSL

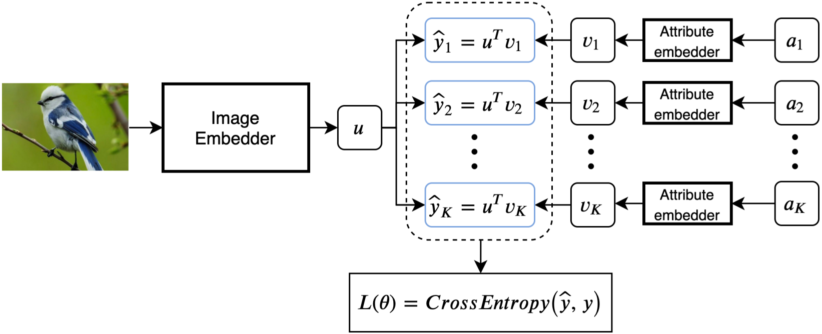
Continual Zero-Shot Learning

- Project: let's use class attributes to improve the performance on future tasks (and this also should improve past performance)
- Why?
  - A robot should be able to understand things it has never seen before but only heard about.
  - And it shouldn't forget previously seen objects while doing so...
  - And the set of attribute descriptions can grow over time...
  - In some sense, it is "the next" step of ZSL
- I.e. build a *zero-shot* model that is trained in a *continual learning* fashion

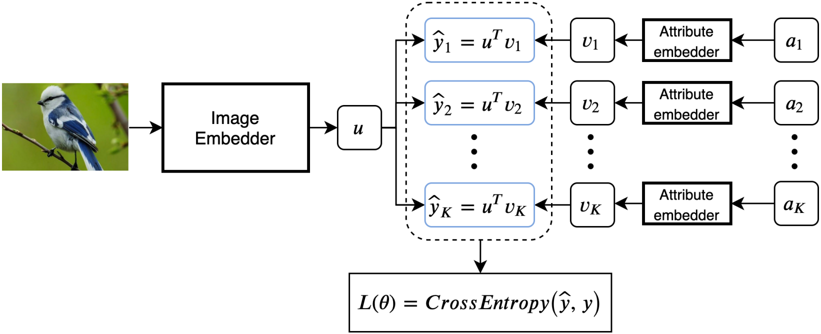
Continual Zero-Shot Learning

- Project: let's use class attributes to improve the performance on future tasks (and this also should improve past performance)
- Why?
  - A robot should be able to understand things it has never seen before but only heard about.
  - And it shouldn't forget previously seen objects while doing so...
  - And the set of attribute descriptions can grow over time...
  - In some sense, it is "the next" step of ZSL
- I.e. build a *zero-shot* model that is trained in a *continual learning* fashion
- Semantic guidance should help to alleviate forgetting without additional regularization and tricks

Model overview

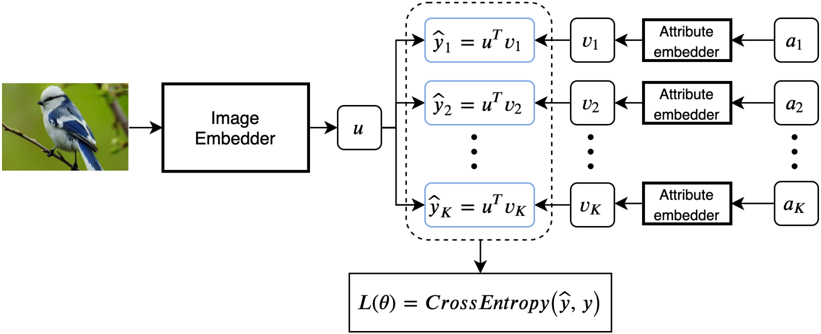


Model overview



The approach is similar in spirit to metric learning:

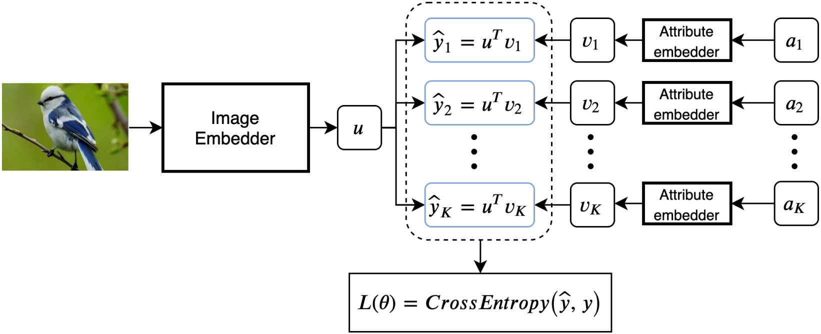
Model overview



The approach is similar in spirit to metric learning:

- Image embedder produces $u = f_\theta(x)$

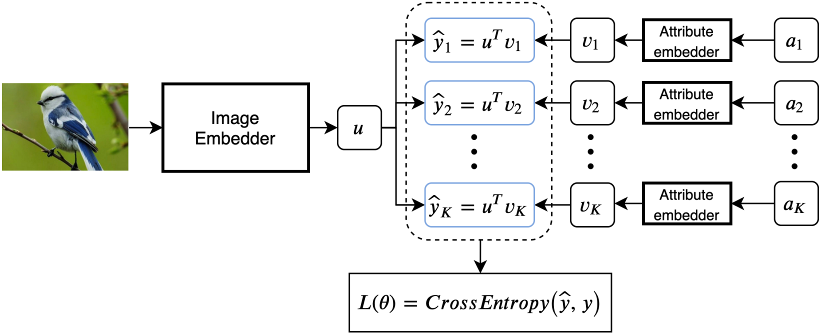
Model overview



The approach is similar in spirit to metric learning:

- Image embedder produces $u = f_\theta(x)$
- Attribute embedder produces $v = g_\phi(a)$

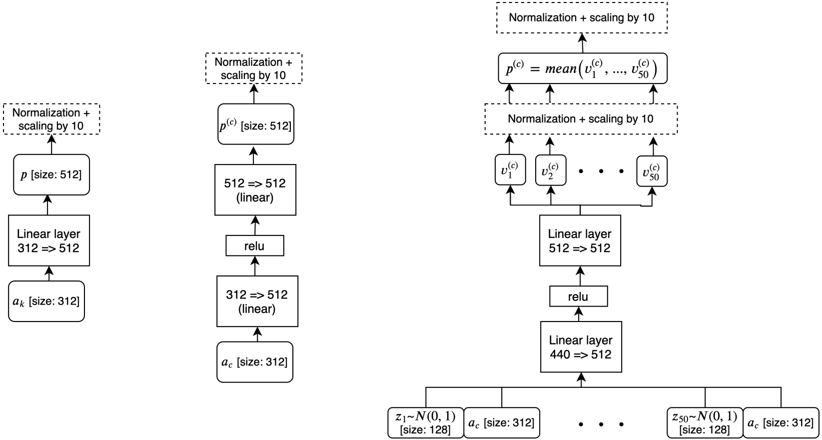
Model overview



The approach is similar in spirit to metric learning:

- Image embedder produces $u = f_\theta(x)$
- Attribute embedder produces $v = g_\phi(a)$
- We want the distance $d(u, v)$ to be low for proper pairs x, a and large for improper ones.

Trying different attribute embedders



Question 1: why the 2-layer attribute embedder works worse?

For some reason, a 2-layer attribute embedder works (much) worse than a linear one

Question 1: why the 2-layer attribute embedder works worse?

For some reason, a 2-layer attribute embedder works (much) worse than a linear one

- It is not overfitting since train/test gap for it is even better

Question 1: why the 2-layer attribute embedder works worse?

For some reason, a 2-layer attribute embedder works (much) worse than a linear one

- It is not overfitting since train/test gap for it is even better
- Can it be due to a bad initialization?

Question 1: why the 2-layer attribute embedder works worse?

For some reason, a 2-layer attribute embedder works (much) worse than a linear one

- It is not overfitting since train/test gap for it is even better
- Can it be due to a bad initialization?
- What could be the other reasons?

Already tried:

Question 1: why the 2-layer attribute embedder works worse?

For some reason, a 2-layer attribute embedder works (much) worse than a linear one

- It is not overfitting since train/test gap for it is even better
- Can it be due to a bad initialization?
- What could be the other reasons?

Already tried:

- Initialize the second layer very close to identity

Question 1: why the 2-layer attribute embedder works worse?

For some reason, a 2-layer attribute embedder works (much) worse than a linear one

- It is not overfitting since train/test gap for it is even better
- Can it be due to a bad initialization?
- What could be the other reasons?

Already tried:

- Initialize the second layer very close to identity
- Different learning rates/momentums for attribute embedder and image embedder

Question 1: why the 2-layer attribute embedder works worse?

For some reason, a 2-layer attribute embedder works (much) worse than a linear one

- It is not overfitting since train/test gap for it is even better
- Can it be due to a bad initialization?
- What could be the other reasons?

Already tried:

- Initialize the second layer very close to identity
- Different learning rates/momentums for attribute embedder and image embedder
- Normalize attributes to zero-mean and unit variance instead of just unit-norm

Question 1: why the 2-layer attribute embedder works worse?

For some reason, a 2-layer attribute embedder works (much) worse than a linear one

- It is not overfitting since train/test gap for it is even better
- Can it be due to a bad initialization?
- What could be the other reasons?

Already tried:

- Initialize the second layer very close to identity
- Different learning rates/momentums for attribute embedder and image embedder
- Normalize attributes to zero-mean and unit variance instead of just unit-norm
- Bernoulli/gaussian dropout

Question 1: why the 2-layer attribute embedder works worse?

For some reason, a 2-layer attribute embedder works (much) worse than a linear one

- It is not overfitting since train/test gap for it is even better
- Can it be due to a bad initialization?
- What could be the other reasons?

Already tried:

- Initialize the second layer very close to identity
- Different learning rates/momentums for attribute embedder and image embedder
- Normalize attributes to zero-mean and unit variance instead of just unit-norm
- Bernoulli/gaussian dropout

What else to try?

- Initializations?
- Residual connections?

Question 2: is it sensible to have a linear generator?

Question 2: is it sensible to have a linear generator?

- Imagine we have a linear generator $G(z, a_k) = W \cdot \binom{z}{a_k}$ for some weight matrix W, class attributes $a_1, ..., a_K$ and $z \sim \mathcal{N}(0, I)$

Question 2: is it sensible to have a linear generator?

- Imagine we have a linear generator $G(z, a_k) = W \cdot \binom{z}{a_k}$ for some weight matrix W, class attributes $a_1, ..., a_K$ and $z \sim \mathcal{N}(0, I)$
- This is equivalent to: $G(z, a_k) = W \cdot \binom{z}{a_k} = W_z z + W_a a_k$, where we split the weight matrix into 2 parts: $W = [W_z, W_a]$.

Question 2: is it sensible to have a linear generator?

- Imagine we have a linear generator $G(z, a_k) = W \cdot \binom{z}{a_k}$ for some weight matrix W, class attributes $a_1, ..., a_K$ and $z \sim \mathcal{N}(0, I)$
- This is equivalent to: $G(z, a_k) = W \cdot \binom{z}{a_k} = W_z z + W_a a_k$, where we split the weight matrix into 2 parts: $W = [W_z, W_a]$.
- Imagine we solve a classification task on top of the generator by using generated prototypes:

Question 2: is it sensible to have a linear generator?

- Imagine we have a linear generator $G(z, a_k) = W \cdot \binom{z}{a_k}$ for some weight matrix W, class attributes $a_1, ..., a_K$ and $z \sim \mathcal{N}(0, I)$
- This is equivalent to: $G(z, a_k) = W \cdot \binom{z}{a_k} = W_z z + W_a a_k$, where we split the weight matrix into 2 parts: $W = [W_z, W_a]$.
- Imagine we solve a classification task on top of the generator by using generated prototypes:
  - I.e. prototype p_k for class k is computed as $p_k = \mathbb{E}_z [G(z, a_k)]$

Question 2: is it sensible to have a linear generator?

- Imagine we have a linear generator $G(z, a_k) = W \cdot \binom{z}{a_k}$ for some weight matrix W, class attributes $a_1, ..., a_K$ and $z \sim \mathcal{N}(0, I)$
- This is equivalent to: $G(z, a_k) = W \cdot \binom{z}{a_k} = W_z z + W_a a_k$, where we split the weight matrix into 2 parts: $W = [W_z, W_a]$.
- Imagine we solve a classification task on top of the generator by using generated prototypes:
  - I.e. prototype p_k for class k is computed as $p_k = \mathop{\mathbb{E}}_{z}[G(z, a_k)]$
  - But then we have

$$p_k = \mathop{\mathbb{E}}_{z}[G(z, a_k)] = \mathop{\mathbb{E}}_{z}[W_z z + W_a a_k] = 0 + W_a a_k = W_a a_k$$

Question 2: is it sensible to have a linear generator?

- Imagine we have a linear generator $G(z, a_k) = W \cdot \binom{z}{a_k}$ for some weight matrix W, class attributes $a_1, ..., a_K$ and $z \sim \mathcal{N}(0, I)$
- This is equivalent to: $G(z, a_k) = W \cdot \binom{z}{a_k} = W_z z + W_a a_k$, where we split the weight matrix into 2 parts: $W = [W_z, W_a]$.
- Imagine we solve a classification task on top of the generator by using generated prototypes:
  - I.e. prototype p_k for class k is computed as $p_k = \mathbb{E}_z [G(z, a_k)]$
  - But then we have

$$p_k = \mathbb{E}_z [G(z, a_k)] = \mathbb{E}_z [W_z z + W_a a_k] = 0 + W_a a_k = W_a a_k$$

  - This means that the generator works just by adding a bit of noise to our prototypes.

Question 2: is it sensible to have a linear generator?

- Imagine we have a linear generator $G(z, a_k) = W \cdot \binom{z}{a_k}$ for some weight matrix W, class attributes $a_1, ..., a_K$ and $z \sim \mathcal{N}(0, I)$
- This is equivalent to: $G(z, a_k) = W \cdot \binom{z}{a_k} = W_z z + W_a a_k$, where we split the weight matrix into 2 parts: $W = [W_z, W_a]$.
- Imagine we solve a classification task on top of the generator by using generated prototypes:
  - I.e. prototype p_k for class k is computed as $p_k = \mathbb{E}_z [G(z, a_k)]$
  - But then we have

$$p_k = \mathbb{E}_z [G(z, a_k)] = \mathbb{E}_z [W_z z + W_a a_k] = 0 + W_a a_k = W_a a_k$$

  - This means that the generator works just by adding a bit of noise to our prototypes.
- Does it mean that using a linear generator is not sensible?
- If so, can we fix that without introducing new layers (since it works worse)?

Question 3: what is the simplest generative model?

Question 3: what is the simplest generative model?

- Our image embedder outputs feature vectors

Question 3: what is the simplest generative model?

- Our image embedder outputs feature vectors
- We want to fit this distribution, but based on class attributes, so to have
  - We can't store mean/covariances of the past task since our feature extractor is changing over time

References

- Rahaf Aljundi et al. "Memory Aware Synapses: Learning what (not) to forget". In: *CoRR* abs/1711.09601 (2017).
- Arslan Chaudhry et al. "Efficient Lifelong Learning with A-GEM". In: *International Conference on Learning Representations*. 2019.
- Chrisantha Fernando et al. "PathNet: Evolution Channels Gradient Descent in Super Neural Networks". In: *CoRR* abs/1701.08734 (2017).
- James Kirkpatrick et al. "Overcoming catastrophic forgetting in neural networks". In: *Proceedings of the National Academy of Sciences* 114.13 (2017), pp. 3521–3526.
- Joan Serra et al. "Overcoming Catastrophic Forgetting with Hard Attention to the Task". In: *ICML*. Vol. 80. Proceedings of Machine Learning Research. PMLR, Oct. 2018, pp. 4548–4557.
- Chenshen Wu et al. "Memory Replay GANs: Learning to Generate New Categories without Forgetting". In: *Advances in Neural Information Processing Systems 31*. Curran Associates, Inc., 2018, pp. 5962–5972.