photo by unsplash user @tuvaloland

CMP784

DEEP LEARNING

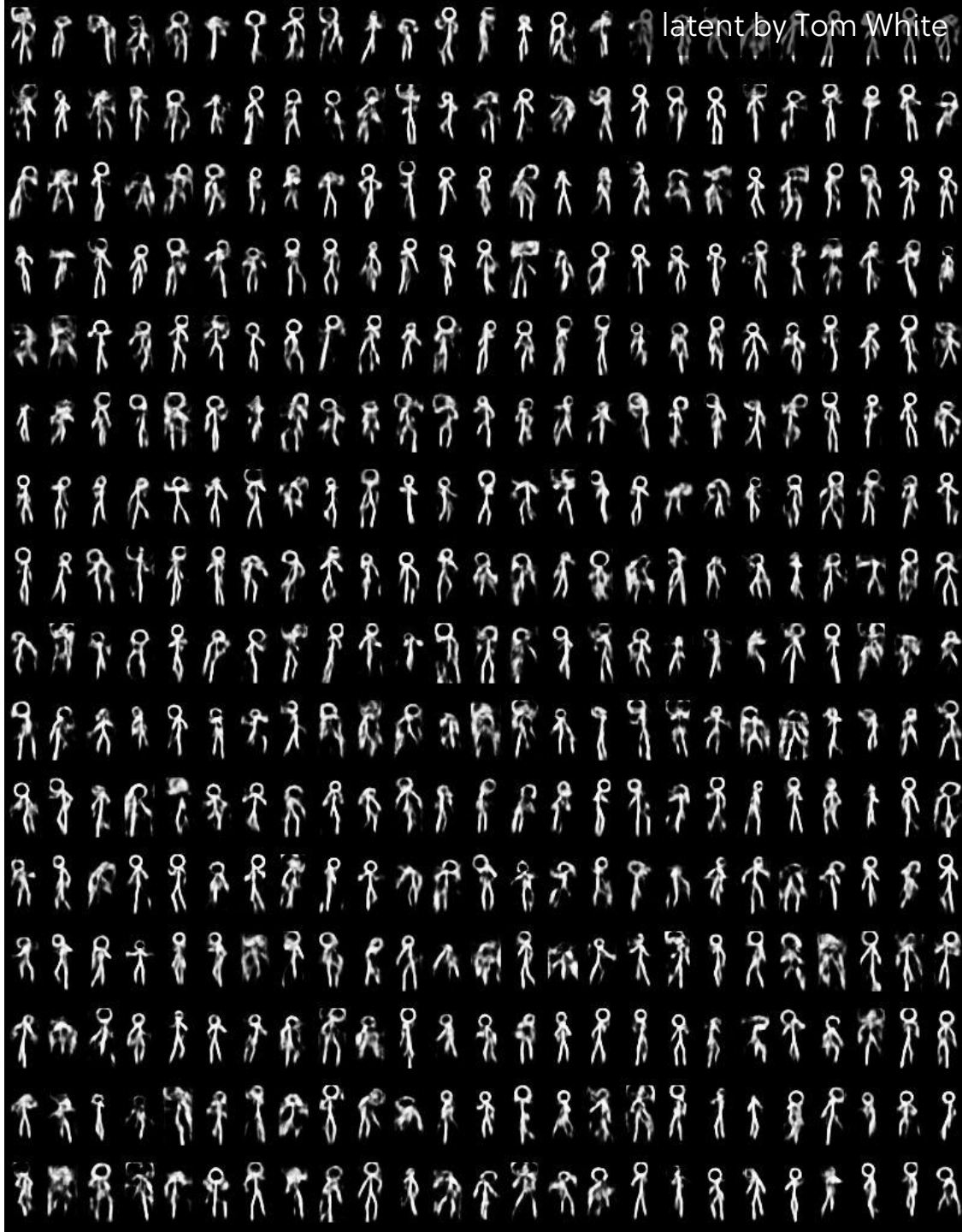
Lecture #12 – Self-Supervised Learning

HACETTEPE UNIVERSITY COMPUTER VISION LAB

Aykut Erdem // Hacettepe University // Spring 2020

Previously on CMP784

- Motivation for Variational Autoencoders (VAEs)
- Mechanics of VAEs
- Separatibility of VAEs
- Training of VAEs
- Evaluating representations
- Vector Quantized Variational Autoencoders (VQ-VAEs)



Lecture Overview

- Predictive / Self-supervised learning
- Self-supervised learning in NLP
- Self-supervised learning in vision

Disclaimer: Much of the material and slides for this lecture were borrowed from

—Andrej Risteski's CMU 10707 class

—Jimmy Ba's UToronto CSC413/2516 class

Unsupervised Learning

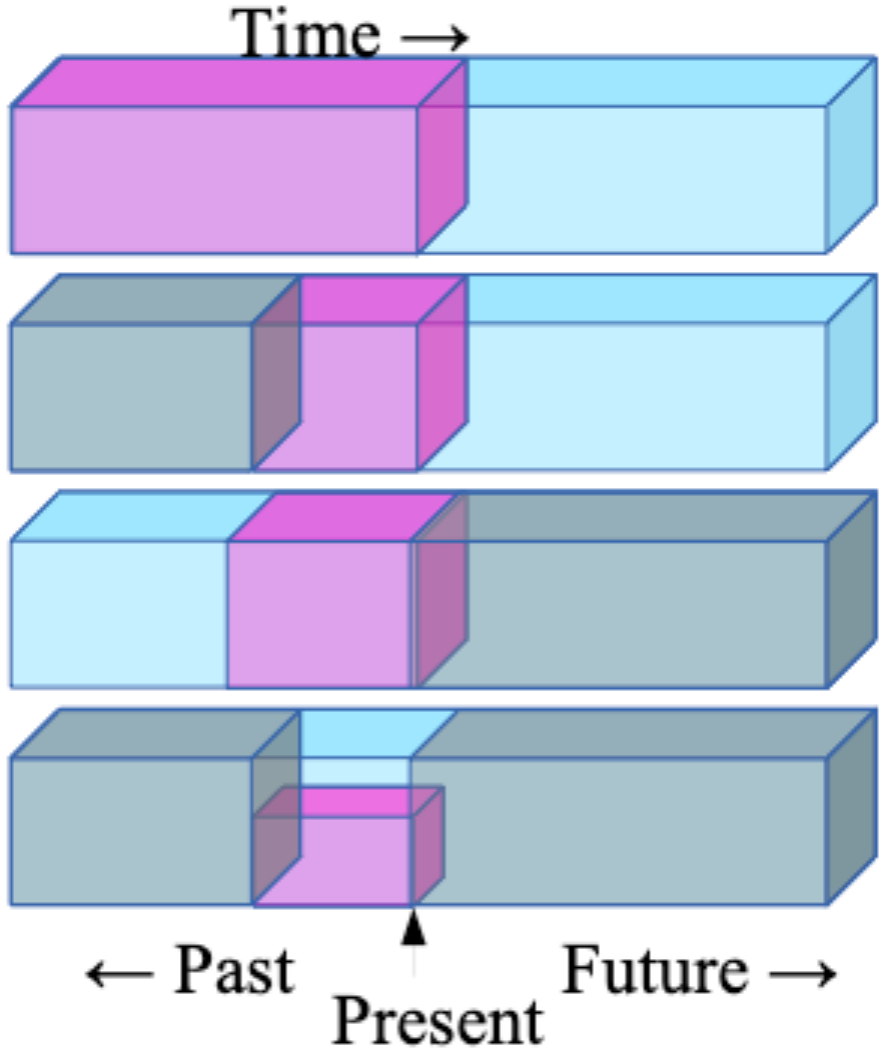
- Learning from data **without** labels.
- What can we hope to do:
  - **Task A:** Fit a parametrized structure (e.g. clustering, low-dimensional subspace, manifold) to data to reveal something meaningful about data **(Structure learning)**
  - **Task B:** Learn a (parametrized) distribution close to data generating distribution. **(Distribution learning)**
  - **Task C:** Learn a (parametrized) distribution that implicitly reveals an "embedding"/"representation" of data for downstream tasks. **(Representation/feature learning)**
- Entangled! The "structure" and "distribution" often reveals an embedding.

Self-Supervised/Predictive Learning

- Given unlabeled data, design supervised tasks that induce a good representation for downstream tasks.
- No good mathematical formalization, but the intuition is to "force" the predictor used in the task to learn something "semantically meaningful" about the data.

Self-Supervised/Predictive Learning

- Predict any part of the input from any other part.
- Predict the future from the past.
- Predict the future from the recent past.
- Predict the past from the present.
- Predict the top from the bottom.
- Predict the occluded from the visible
- Pretend there is a part of the input you don't know and predict that.



How Much Information Does the Machine Need to Predict?

Y LeCun

"Pure" Reinforcement Learning (cherry)

- The machine predicts a scalar reward given once in a while.
- **A few bits for some samples**

Supervised Learning (icing)

- The machine predicts a category or a few numbers for each input
- Predicting human-supplied data
- **10→10,000 bits per sample**

Unsupervised/Predictive Learning (cake)

- The machine predicts any part of its input for any observed part.
- Predicts future frames in videos
- **Millions of bits per sample**

(Yes, I know, this picture is slightly offensive to RL folks. But I'll make it up)

- LeCun's original cake analogy slide, presented at his keynote speech in NIPS 2016.

How Much Information is the Machine Given during Learning?

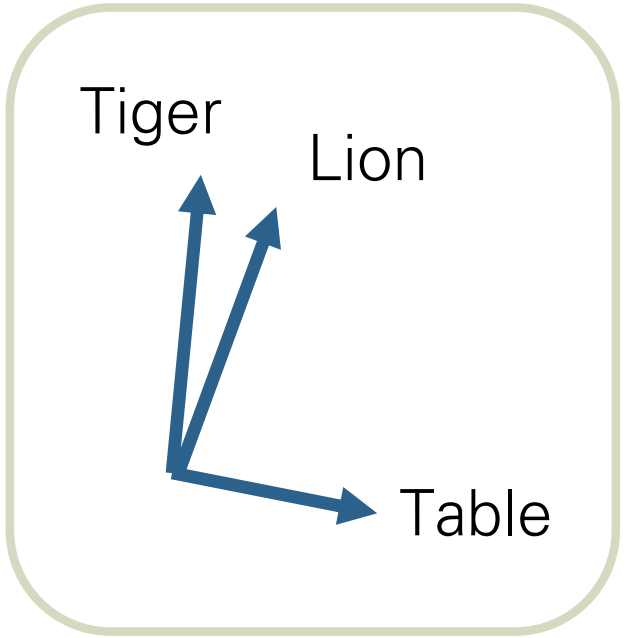
- **"Pure" Reinforcement Learning (cherry)**
  - The machine predicts a scalar reward given once in a while.
  - **A few bits for some samples**
- **Supervised Learning (icing)**
  - The machine predicts a category or a few numbers for each input
  - Predicting human-supplied data
  - **10→10,000 bits per sample**
- **Self-Supervised Learning (cake génoise)**
  - The machine predicts any part of its input for any observed part.
  - Predicts future frames in videos
  - **Millions of bits per sample**

- Updated version at (ISSCC 2019, where he replaced "**unsupervised learning**" with "**self-supervised learning**".

Self-Supervised Learning in NLP

Word Embeddings

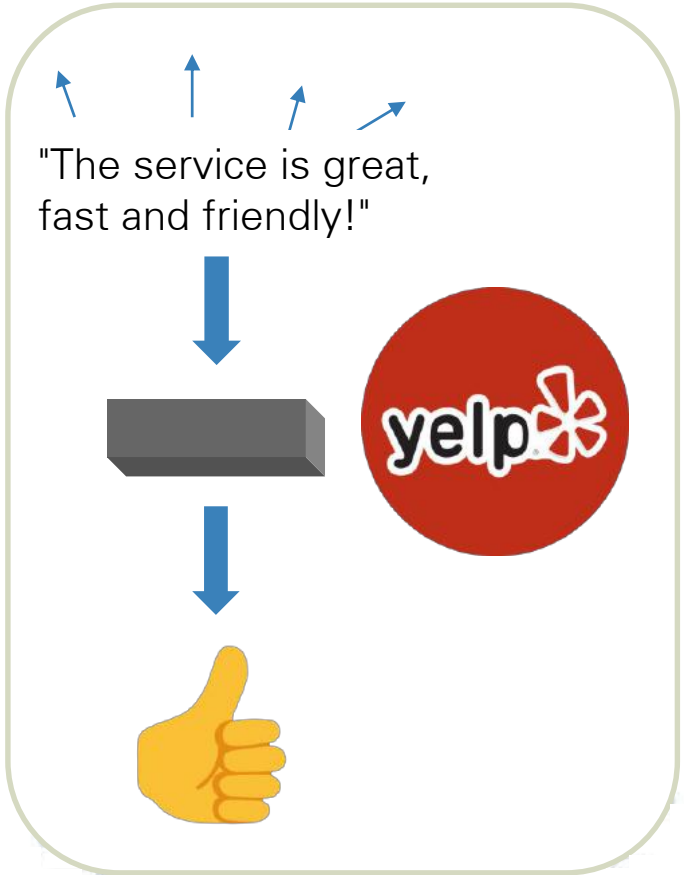
- Semantically meaningful **vector representations** of words



Example: Inner product (possibly scaled, i.e. cosine similarity) correlates with word similarity.

Word Embeddings

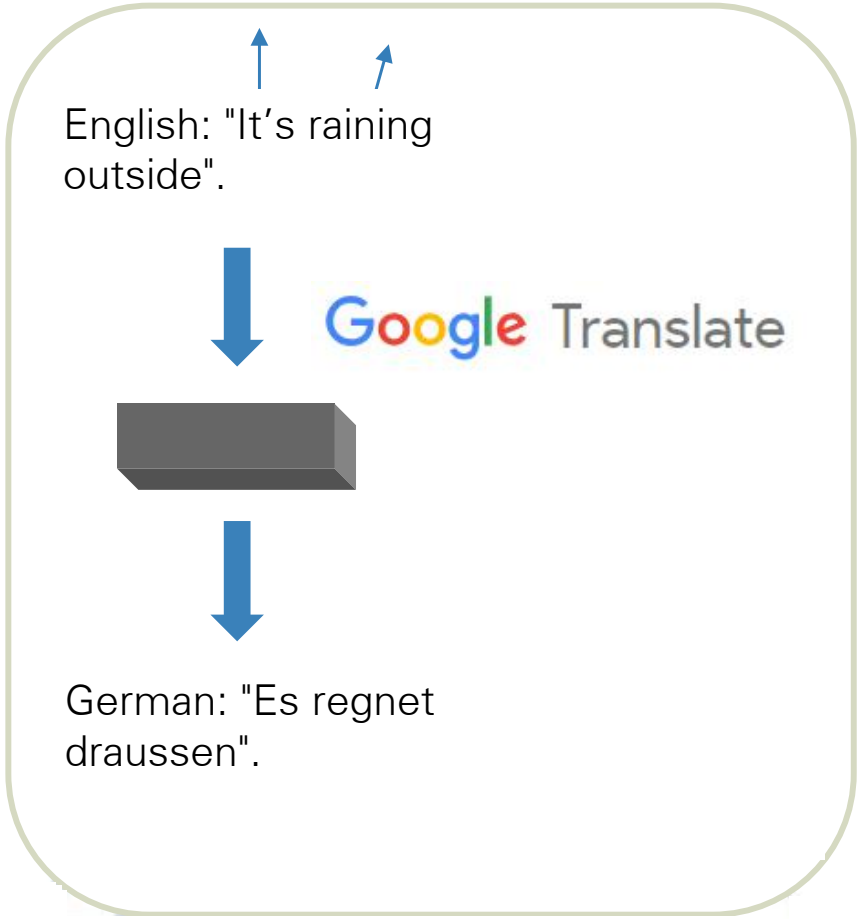
- Semantically meaningful **vector representations** of words



Example: Can use embeddings to do sentiment classification by training a simple (e.g. linear) classifier

Word Embeddings

- Semantically meaningful **vector representations** of words



Example: Can train a "simple" network that if fed word embeddings for two languages, can effectively translate.

Word Embeddings via Predictive Learning

- **Basic task:** predict the next word, given a few previous ones.

In other words, optimize for

$$\max_{\theta} \sum_{t} \log p_{\theta} \left(x_t | x_{t-1}, x_{t-2}, \ldots, x_{t-L} \right)$$

Word Embeddings via Predictive Learning

- **Basic task:** predict the next word, given a few previous ones.

$$\max_{\theta} \sum_{t} \log p_\theta \left(x_t | x_{t-1}, x_{t-2}, \ldots, x_{t-L} \right)$$

Inspired by classical assumptions in NLP that the underlying distribution is Markov – that is, x_t only depends on the previous few words.

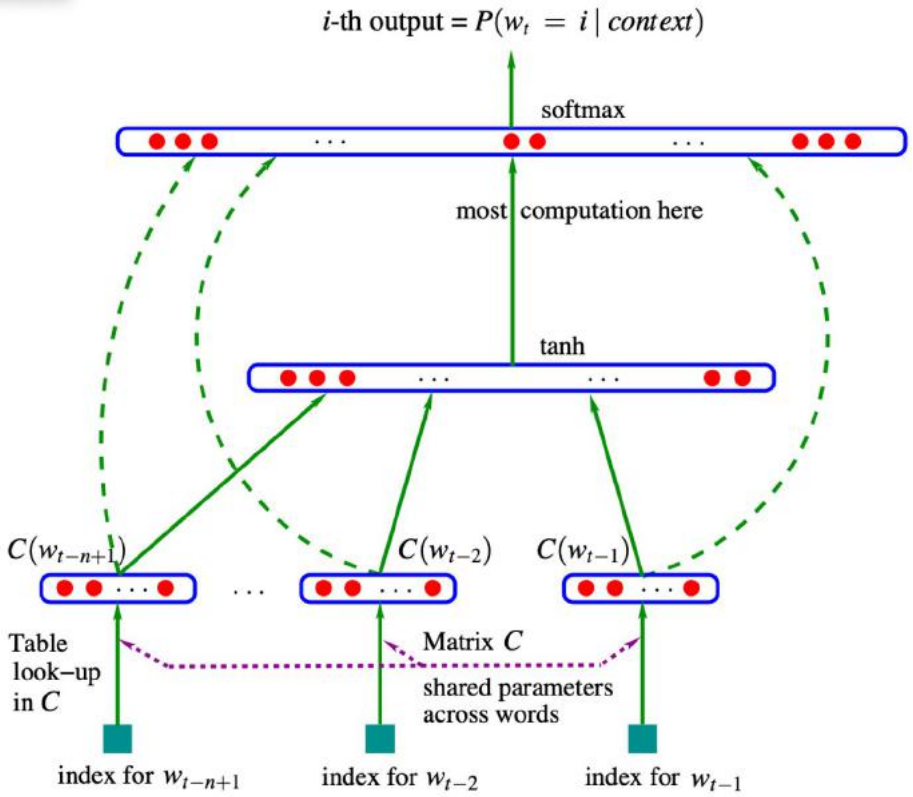
(Of course, this is violated if you wish to model long texts like paragraphs / books.)

The main issue: The trivial way of parametrizing $p_\theta \left(x_t | x_{t-1}, x_{t-2}, \ldots, x_{t-L} \right)$ is a "lookup table" with V^L entries.

Word Embeddings via Predictive Learning

- **Basic task:** predict the next word, given a few previous ones.

$$\max_{\theta} \sum_{t} \log p_{\theta}\left(x_t | x_{t-1}, x_{t-2}, \ldots, x_{t-L}\right)$$



[Bengio-Ducharme-Vincent-Janvin '2003]: A neural parametrization of the above probabilities.

Main ingredients:

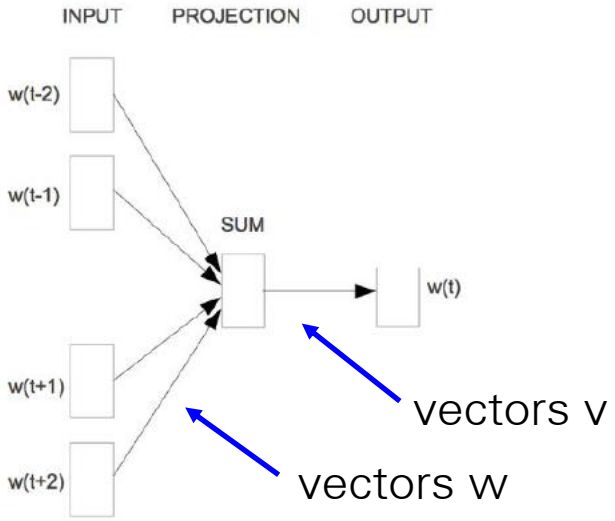
- Embeddings: A word embedding $C(w)$ for all words w in dictionary.
- Non-linear transforms: Potentially deep network taking as inputs i, $C(x_{t-1})$, $C(x_{t-2})$,...,$C(x_{t-L})$, and outputting some vector o. Can be recurrent net too.
- Softmax: Softmax distribution for x_t with parameters given by o.

Word Embeddings via Predictive Learning

- **Related:** predict middle word in a sentence, given surrounding ones.

$$\max_{\theta} \sum_{t} \log p_\theta \left(x_t | x_{t-L}, \ldots, x_{t-1}, x_{t+1}, \ldots, x_{t+L} \right)$$

CBOW (Continuous Bag of Words): proposed by Mikolov et al. '13



Parametrization is chosen s.t.

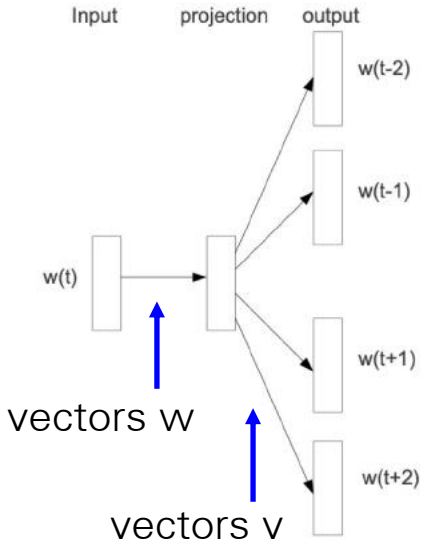
$$p_\theta \left(x_t | x_{t-L}, \ldots, x_{t-1}, x_{t+1}, \ldots, x_{t+L} \right) \propto \exp\left(v_{x_t}, \sum_{i=t-L}^{t+L} w_{t_i} \right)$$

Word Embeddings via Predictive Learning

- **Related:** predict middle word in a sentence, given surrounding ones.

$$\max_{\theta} \sum_{t} \sum_{i=t-L, i \neq t}^{t+L} \log p_{\theta}\left(x_i | x_t\right)$$

Skip-Gram: also proposed by Mikolov et al. '13



Parametrization is chosen s.t. $p_{\theta}\left(x_i | x_t\right) \propto \exp\left(v_{x_i}, w_{x_t}\right)$

In practice, lots of other tricks are tacked on to deal with the slowest part of training: the softmax distribution (partition function sums over entire vocabulary).

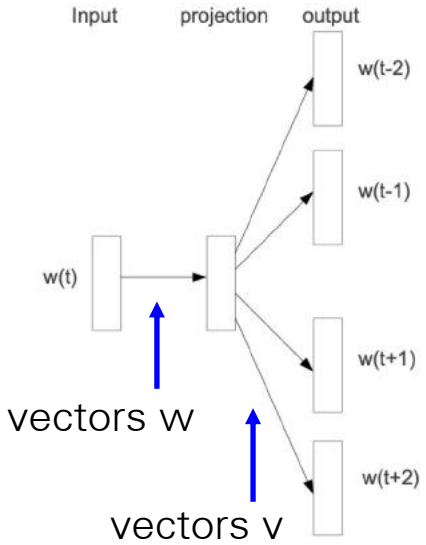
Common ones are negative sampling, hierarchical softmax, etc.

Word Embeddings via Predictive Learning

- **Related:** predict middle word in a sentence, given surrounding ones.

$$\max_{\theta} \sum_{t} \sum_{i=t-L, i \neq t}^{t+L} \log p_{\theta}\left(x_i | x_t\right)$$

Skip-Gram: also proposed by Mikolov et al. '13



Tomas Mikolov 10/7/13

There are quite a few differences between the skip-gram and the CBOW models. However, if you have a lot of training data, their performance should be comparable.

If you want to see a list of advantages of each model, then my current experience is:

Skip-gram: works well with small amount of the training data, represents well even rare words or phrases
CBOW: several times faster to train than the skip-gram, slightly better accuracy for the frequent words

This can get even a bit more complicated if you consider that there are two different ways how to train the models: the normalized hierarchical softmax, and the un-normalized negative sampling. Both work quite differently.

Overall, the best practice is to try few experiments and see what works the best for you, as different applications have different requirements.

- show quoted text -

Evaluating Word Embeddings

- First variant (predict next word, given previous ones) can be used as a **generative model** for text. (Also called language model.) The other ones cannot.
- In former case, a natural measure is the cross-entropy

$$-\mathbb{E}_{x_1,x_2,\ldots,x_T} \log p_\theta\left(x_{\leq T}\right) = \mathbb{E}_{x_1,x_2,\ldots,x_T} \sum_t \log p_\theta\left(x_t | x_{<t}\right)$$

- For convenience, we often take exponential of this (called perplexity)
- If we do not have a generative model, we have to use indirect means.

Evaluating Word Embeddings

- **Intrinsic tasks:** Test performance of word embeddings on tasks measuring their "semantic" properties. Examples include solving "which is the most similar word" queries, analogy queries (i.e. "man is to woman as king is to ??"
- **Extrinsic tasks:** How well can we "finetune" the word embeddings to solve some (supervised) downstream task. "Finetune" usually means train a (relatively small) feedforward network. Examples of such tasks include:
  - Part-of-Speech Tagging (determine whether a word is noun/verb/...),
  - Named Entity Recognition (recognizing named entities like persons, places) – e.g. label a sentence as Picasso[person] died in France[country], many others.

Semantic Similarity

- **Observation**: similar words tend to have larger (renormalized) inner products (also called cosine similarity).
- Precisely, if we look at the word embeddings for words i,j
  $\left\langle \frac{w_i}{||w_i||}, \frac{w_j}{||w_j||} \right\rangle = \cos(w_i, w_j)$ tends to be larger for similar words i,j

  Example: the nearest neighbors to "Frog" look like

0. frog
1. frogs
2. toad
3. litoria
4. leptodactylidae
5. rana
6. lizard
7. eleutherodactylus

3. litoria

4. leptodactylidae

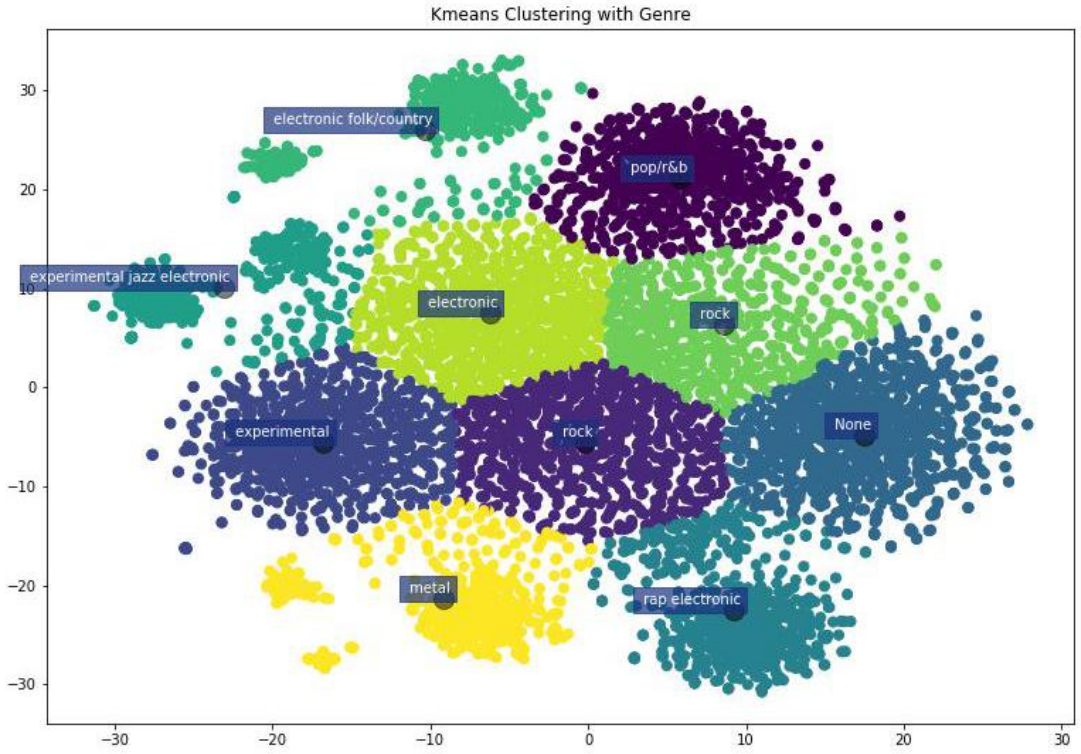
5. rana

7. eleutherodactylus

- To solve semantic similarity query like "which is the most similar word to", output the word with the highest cosine similarity.

Semantic Clustering

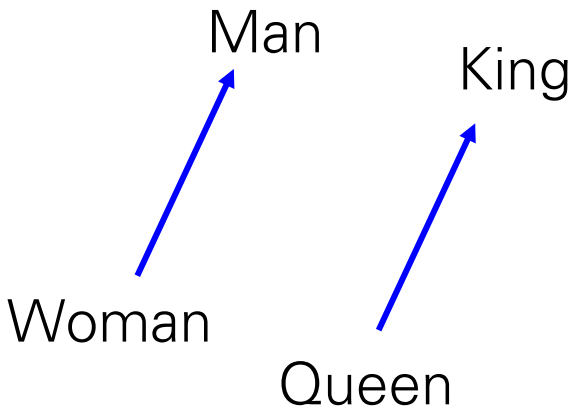
- Consequence: clustering word embeddings should give "semantically" relevant clusters.



t-SNE projection of word embeddings for artists (clustered by genre). Image from https://medium.com/free-code-camp/learn-tensorflow-the- word2vec-model-and-the-tsne-algorithm-using-rock-bands-97c99b5dcb3a

Analogies

- **Observation**: You can solve analogy queries by linear algebra.



Precisely, w = queen will be the solution to:

$$\text{argmin}_w \|v_w - v_{\text{king}} - (v_{\text{woman}} - v_{\text{man}})\|^2$$

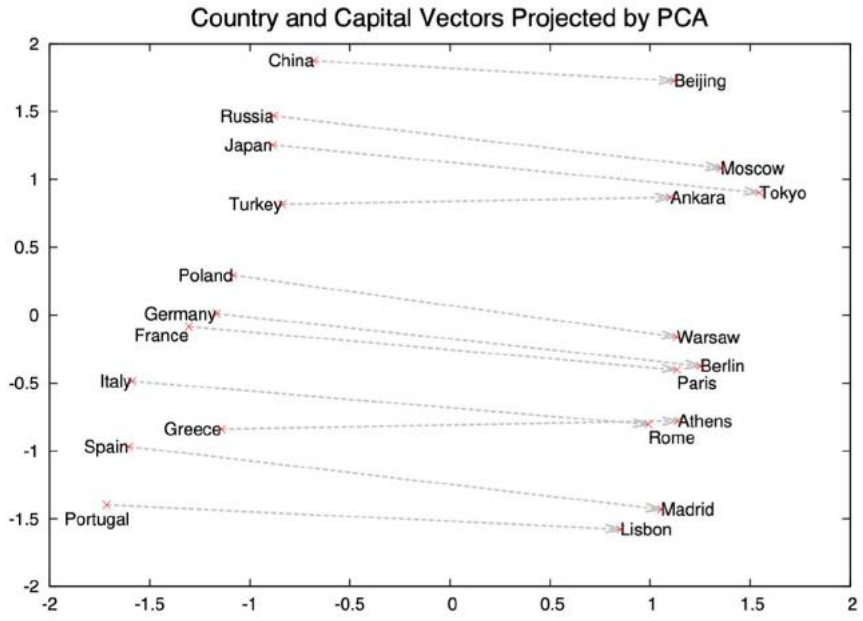


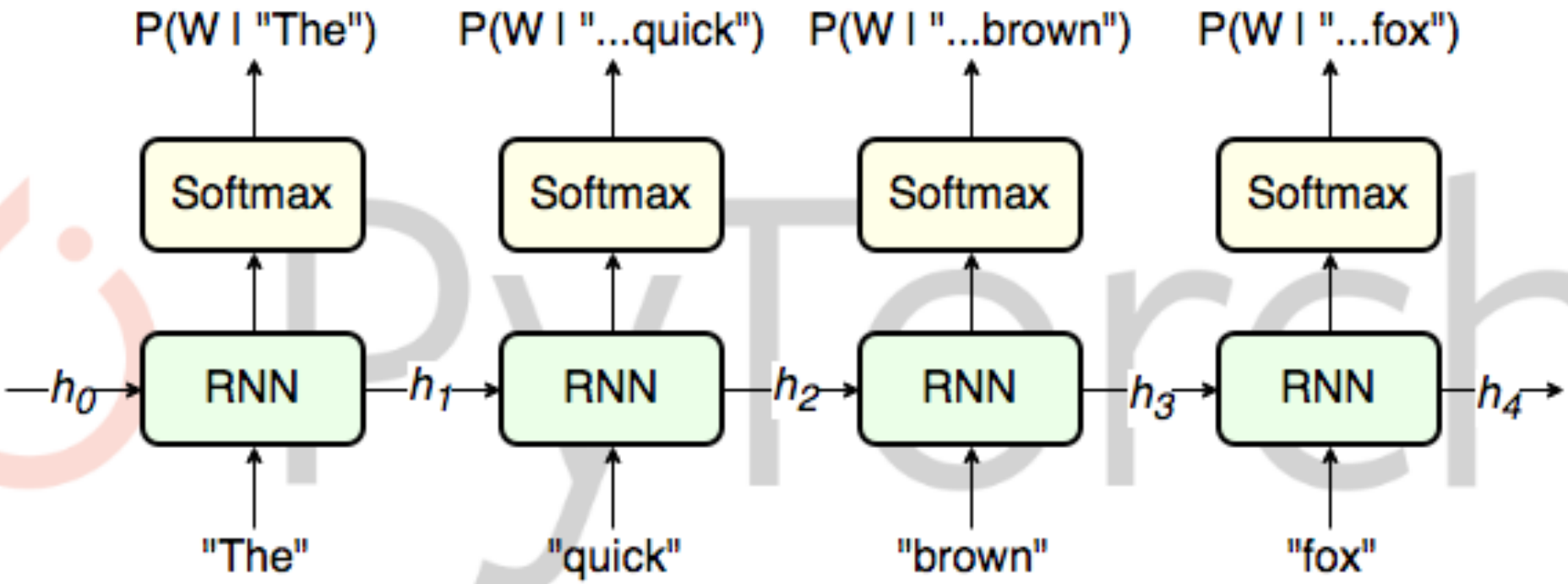
Figure 2: Two-dimensional PCA projection of the 1000-dimensional Skip-gram vectors of countries and their capital cities. The figure illustrates ability of the model to automatically organize concepts and learn implicitly the relationships between them, as during the training we did not provide any supervised information about what a capital city means.

Language Models (LMs)

- A statistical model that assigns probabilities to the words in a sentences.
- **Most commonly:** Given previous words, what should the next one be?
- **Neural language model:** Model the probability of words given others using neural networks.

Recurrent Architectures for LM

- We can use recurrent architectures.
- LSTM, GRU ...
- Great for variable length inputs, like sentences.



Recurrent Architectures for LM

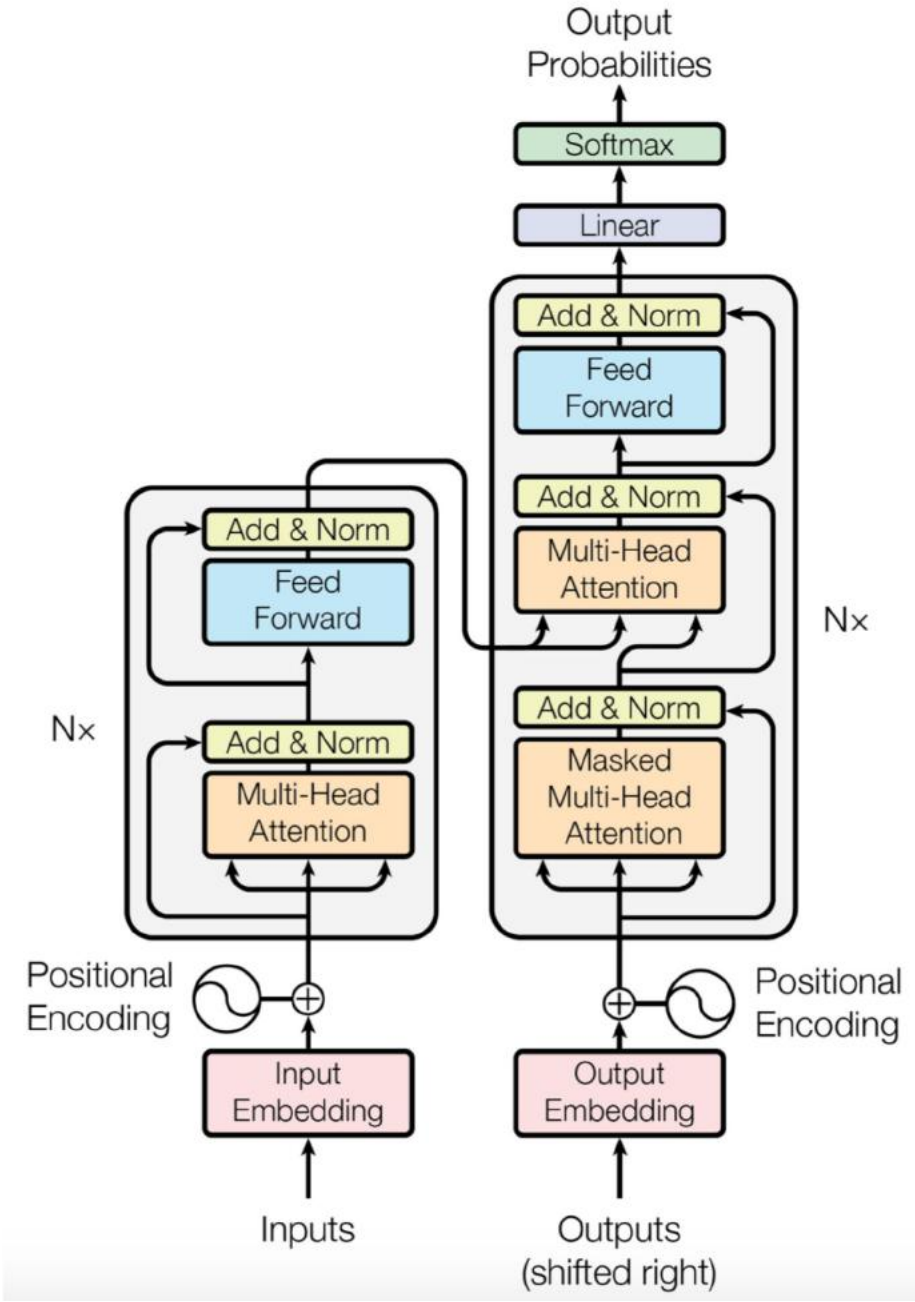
- What are some of the problems with recurrent architectures?
  - Not parallelizable across instances.
  - Cannot model long dependences.
  - Optimization difficulties (vanishing gradients).

- Attention to the rescue!

Transformers

Properties of the transformer architecture:

- Fully feed forward.
- Equivariance properties of scaled dot product attention (important):
  - How does the output change if we permute the order of queries? (equivariance)
  - How does the output change if we permute the key-value pairs in unison? (invariance)



Performance Comparison

Table 1: Maximum path lengths, per-layer complexity and minimum number of sequential operations for different layer types. n is the sequence length, d is the representation dimension, k is the kernel size of convolutions and r the size of the neighborhood in restricted self-attention.

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

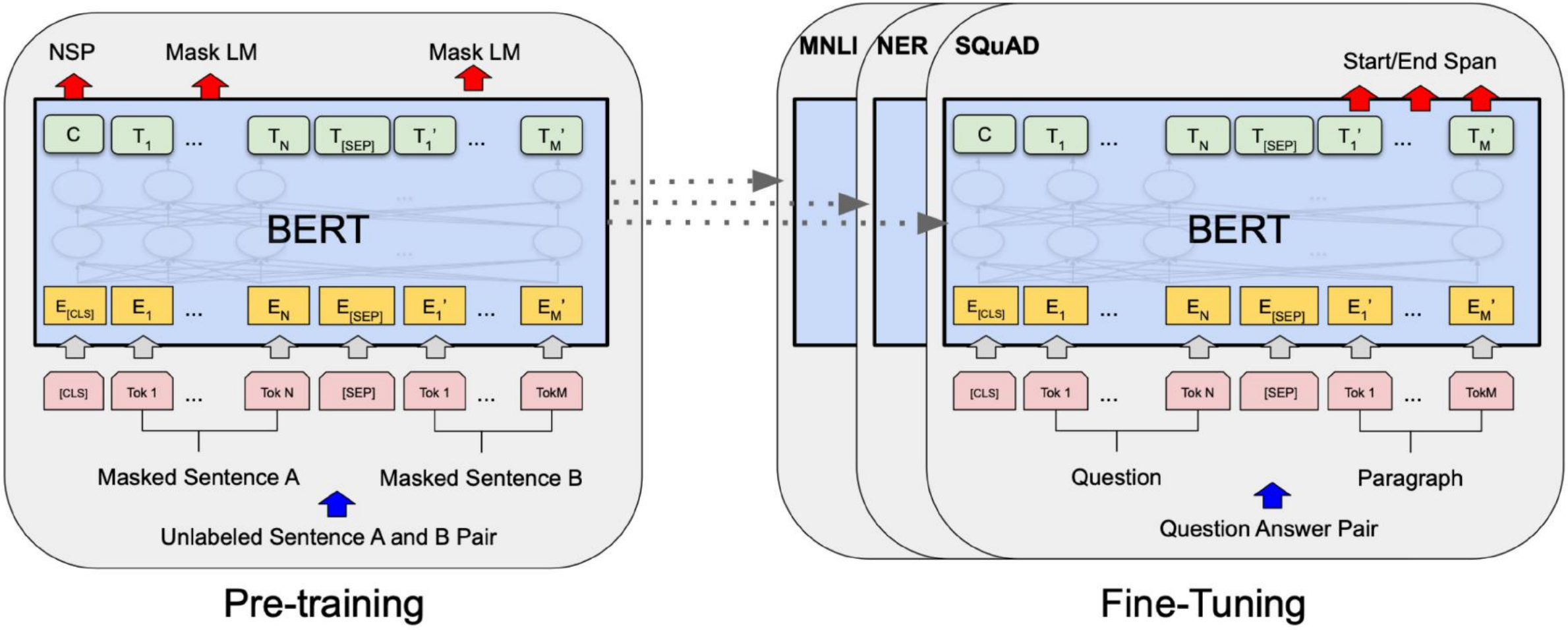
Pretraining Language Models

- Can we use large amounts of text data to pretrain language models?
- Considerations:
  - How can we fuse both left-right and right-left context?
  - How can we facilitate non-trivial interactions between input tokens?
- Previous approaches:
  - ELMO (Peters. et. al., 2017): Bidirectional, but shallow.
  - GPT (Radford et. al., 2018): Deep, but unidirectional.
  - BERT (Devlin et. al., 2018): Deep and bidirectional!

BERT Workflow

- The BERT workflow includes:
  - Pretrain on generic, self-supervised tasks, using large amounts of data (like all of Wikipedia)
  - Fine-tune on specific tasks with limited, labelled data.
- The pretraining tasks (will talk about this in more detail later):
  - Masked Language Modelling (to learn contextualized token representations)
  - Next Sentence Prediction (summary vector for the whole input)

BERT Architecture



BERT Architecture

Properties:

- Two input sequences.
  - Many NLP tasks have two inputs (question answering, paraphrase detection, entailment detection etc.)
- Computes embeddings
  - Both token, position and segment embeddings.
  - Special start and separation tokens.
- Architecture
  - Basically the same as transformer encoder.
- Outputs:
  - Contextualized token representations.
  - Special tokens for context.

BERT Embeddings

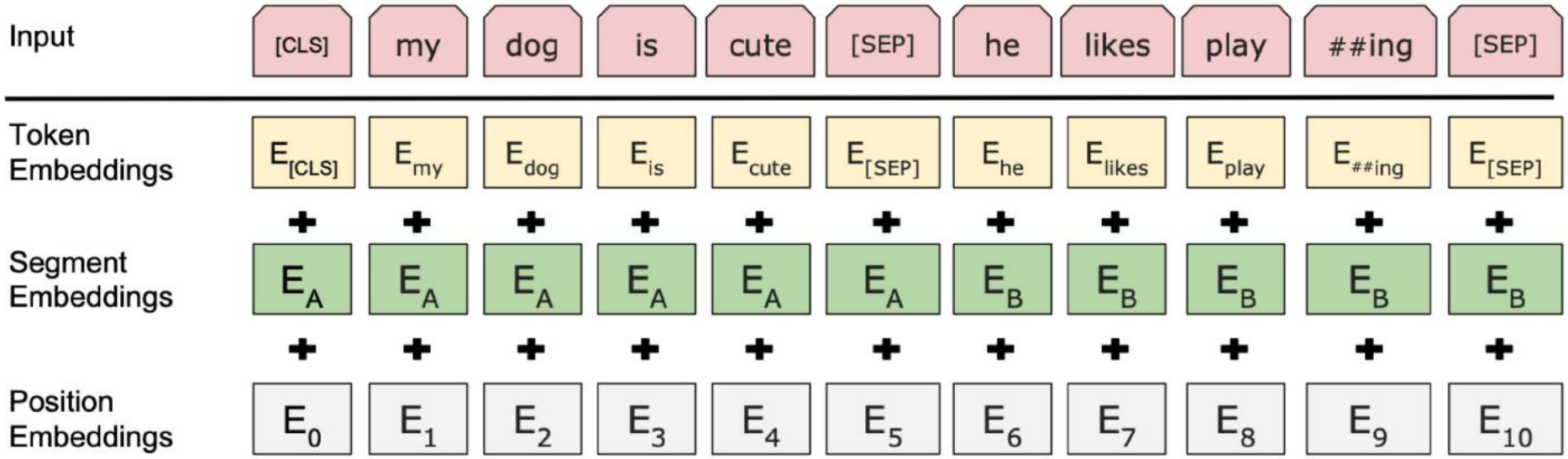


Figure 2: BERT input representation. The input embeddings are the sum of the token embeddings, the segmentation embeddings and the position embeddings.

- How we tokenize the inputs is very important!
- BERT uses the WordPiece tokenizer (Wu et. al. 2016)

(Aside) Tokenizers

- Tokenizers have to balance the following:
  - Being comprehensive (rare words? translation to different languages)
  - Total number of tokens
  - How semantically meaningful each token is.
- This is an activate area of research.

Pretraining tasks

- Masked Language Modelling, i.e. Cloze Task (Taylor, 1953)
- Next sentence prediction

Masked Language Modelling

- Mask 15% of the input tokens. (i.e. replace with a dummy masking token)
- Run the model, obtain the embeddings for the masked tokens.
- Using these embeddings, try to predict the missing token.
- ”I love to eat peanut ___ and jam. ” Can you guess what’s missing?

- **This procedure forces the model to encode context information in the features of all of the tokens.**

Next Sentence Prediction

- Goal is to summarize the complete context (i.e. the two segments) in a single feature vector.
- Procedure for generating data
  - Pick a sentence from the training corpus and feed it as ”segment A”.
  - With 50% probability, pick the following sentence and feed that as ”segment B”.
  - With 50% probability, pick the a random sentence and feed it as ”segment B”.
- Using the features for the context token, predict whether segment B is the following sentence of segment A.
- Turns out to be a very effective pretraining technique!

Fine Tuning

Procedure:

- Add a final layer on top of BERT representations.
- Train the whole network on the fine-tuning dataset.
- Pre-training time: In the order of days on TPUs.
- Fine tuning task: Takes only a few hours max.

Fine Tuning

System	MNLI-(m/mm) 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k	**Average** -
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
$\text{BERT}_{\text{BASE}}$	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
$\text{BERT}_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**92.7**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**82.1**

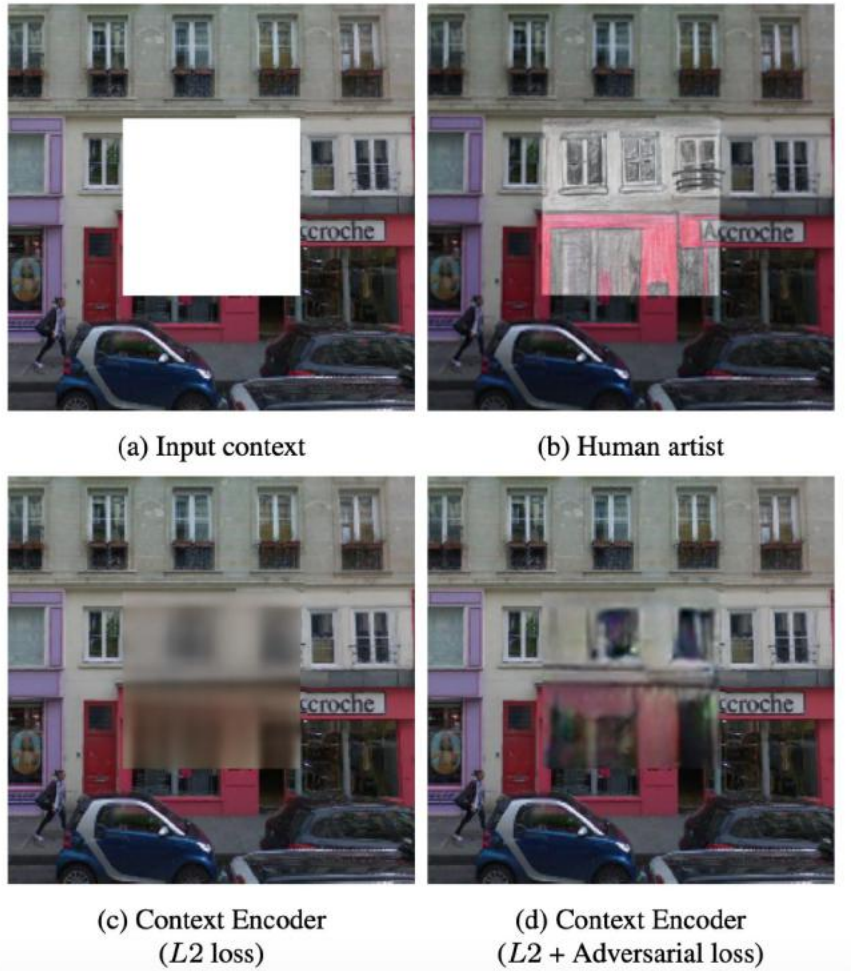
Table 1: GLUE Test results, scored by the evaluation server (https://gluebenchmark.com/leaderboard). The number below each task denotes the number of training examples. The "Average" column is slightly different than the official GLUE score, since we exclude the problematic WNLI set.[8] BERT and OpenAI GPT are single-model, single task. F1 scores are reported for QQP and MRPC, Spearman correlations are reported for STS-B, and accuracy scores are reported for the other tasks. We exclude entries that use BERT as one of their components.

Self-Supervised Learning in Vision

Inpainting

- The most obvious analogy to word embeddings: predict parts of image from remainder of image.

Pathak et al. ’16: Context Encoders: Feature Learning by Inpainting



(a) Input context (b) Human artist

(c) Context Encoder ($L2$ loss) (d) Context Encoder ($L2$ + Adversarial loss)

Inpainting

- The most obvious analogy to word embeddings: predict parts of image from remainder of image.

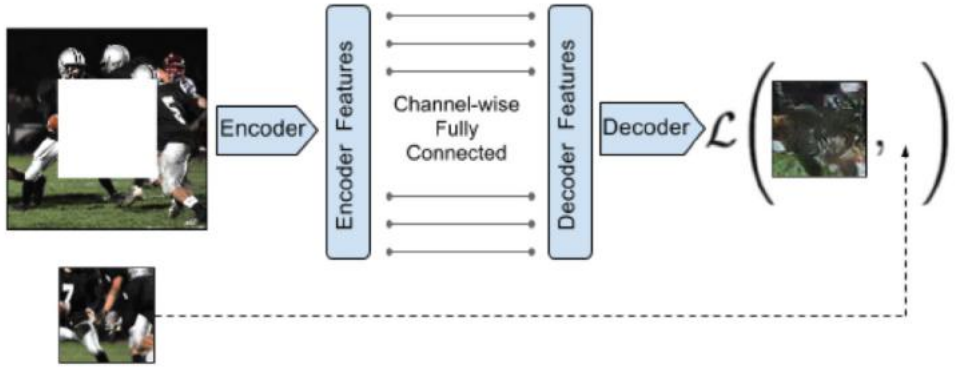


Figure 2: Context Encoder. The context image is passed through the encoder to obtain features which are connected to the decoder using channel-wise fully-connected layer as described in Section 3.1. The decoder then produces the missing regions in the image.

Pathak et al. '16: Context Encoders: Feature Learning by Inpainting

Architecture:
An encoder E takes a part of image, constructs a representation.

A decoder D takes representation, tries to reconstruct missing part.

- **Much** trickier than in NLP:
  As we have seen, meaningful losses for vision are much more difficult to design.
  Choice of region to mask out is much more impactful.

Inpainting

- The most obvious analogy to word embeddings: predict parts of image from remainder of image.

  Pathak et al. ’16: Context Encoders: Feature Learning by Inpainting

- If reconstruction loss is L_2: tendency to produce blurry images.

  Remember: one of the usefulness of GANs is to provide a better loss for images.

(c) Context Encoder ($L2$ loss)

(d) Context Encoder ($L2$ + Adversarial loss)

Inpainting

- The most obvious analogy to word embeddings: predict parts of image from remainder of image.

  Pathak et al. '16: Context Encoders: Feature Learning by Inpainting

- If reconstruction loss is L_2: tendency to produce blurry images.
- Remember: one of the usefulness of GANs is to provide a better loss for images.

Composition of encoder+decoder

Mask

$$\mathcal{L}_{rec}(x) = \|\hat{M} \odot (x - F((1 - \hat{M}) \odot x))\|_2^2,$$

DC-GAN objective

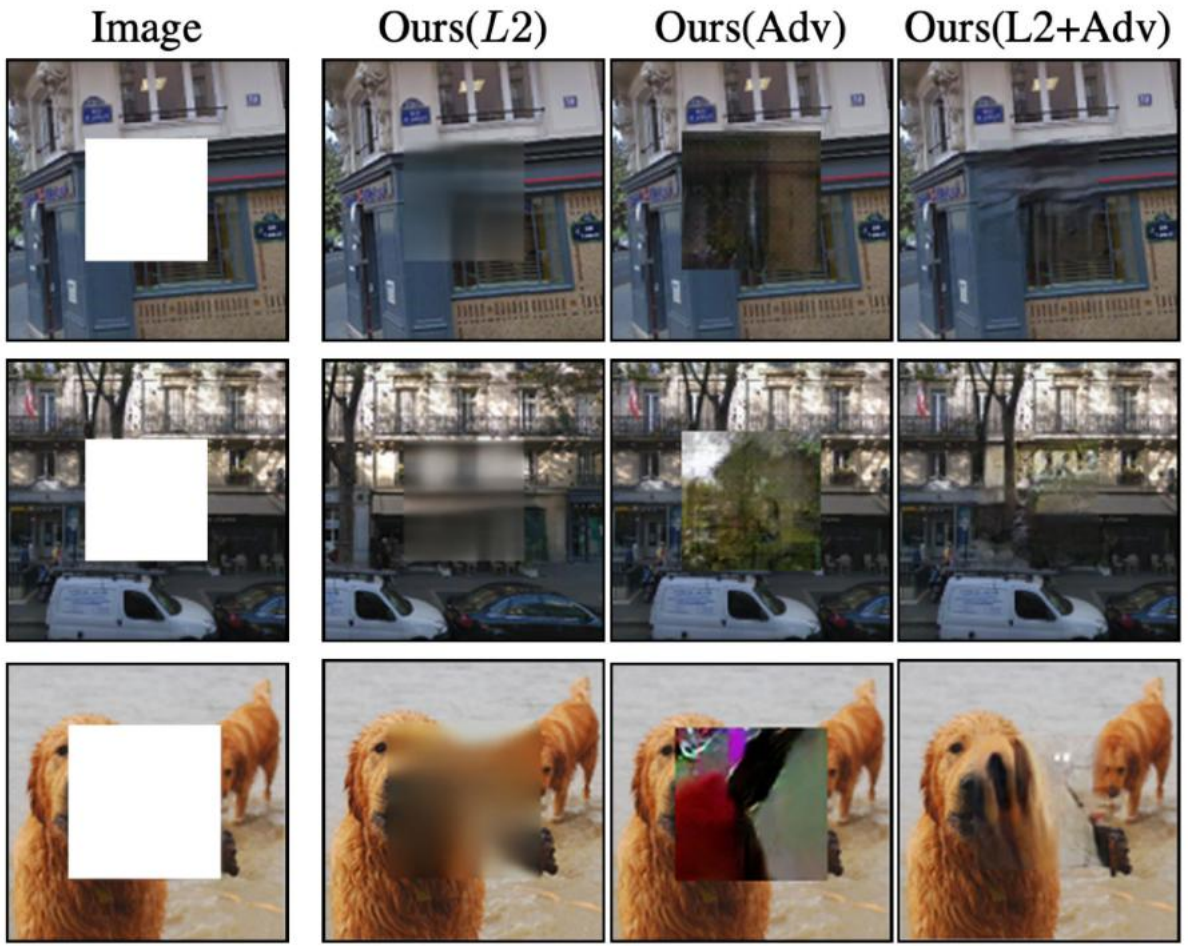
$$\mathcal{L}_{adv} = \max_D \quad \mathbb{E}_{x \in \mathcal{X}}[\log(D(x)) + \log(1 - D(F((1 - \hat{M}) \odot x)))],$$

$$\mathcal{L} = \lambda_{rec}\mathcal{L}_{rec} + \lambda_{adv}\mathcal{L}_{adv}.$$

Inpainting

- The most obvious analogy to word embeddings: predict parts of image from remainder of image.

  Pathak et al. ’16: Context Encoders: Feature Learning by Inpainting

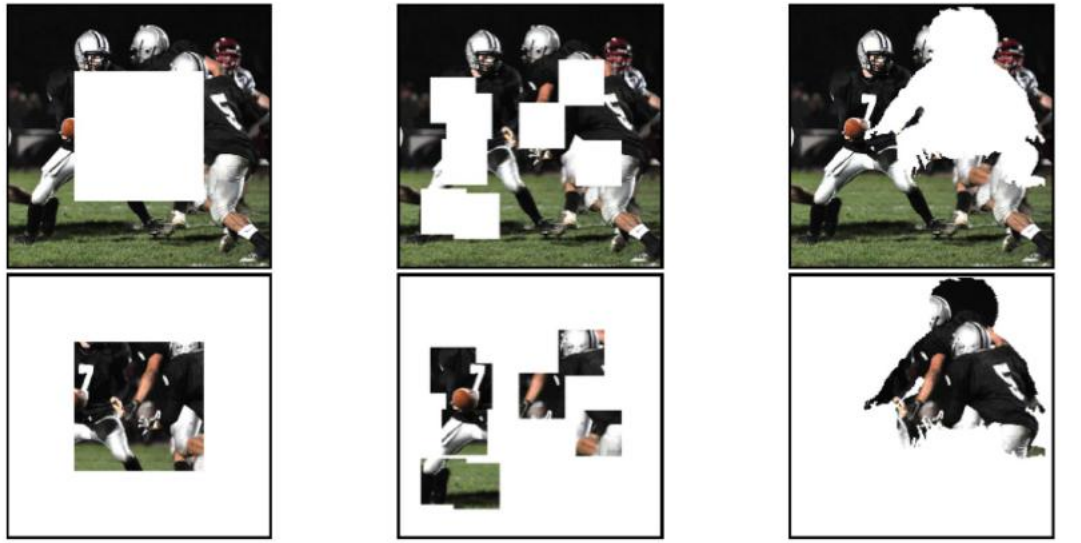


Inpainting

- The most obvious analogy to word embeddings: predict parts of image from remainder of image.

  Pathak et al. ’16: Context Encoders: Feature Learning by Inpainting

- How to choose the region?



(a) Central region (b) Random block (c) Random region

Figure 3: An example of image x with our different region masks $\hat{M}$ applied, as described in Section 3.3.

Task should be “solvable”, but not “too easy”.

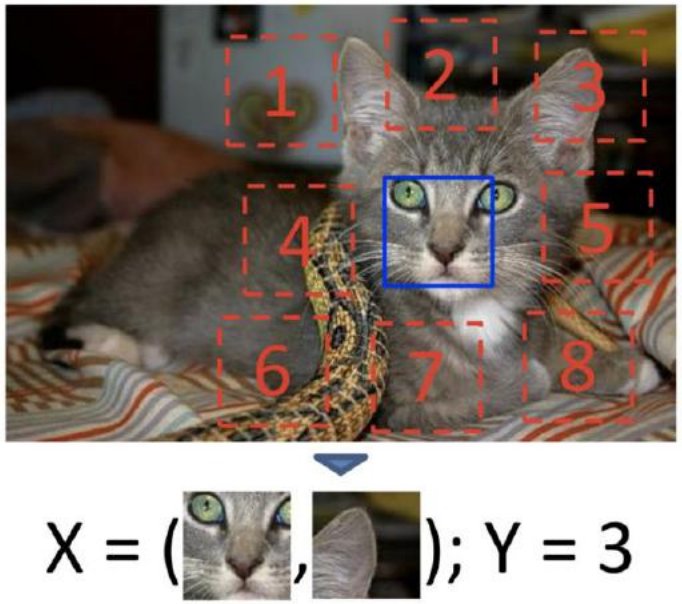
- Fixed (central region): tends to produce less generalizeable representations
- Random blocks: slightly better, but square borders still hurt.
- Random silhouette (fully random doesn’t make sense – prediction task is too ill-defined) – even better!

Jigsaw puzzles

- In principle, what we want is a task "hard enough", that any model that does well on it, should learn something "meaningful" about the task.

  Doersch et al. '15: Unsupervised Visual Representation Learning by Context Prediction

- **Task**: Predict ordering of two randomly chosen pieces from the image.



Representation: penultimate layer of a neural net used to solve task.

Intuition: understanding relative positioning of pieces of an image requires some understanding of how images are composed.

Jigsaw puzzles

- In principle, what we want is a task "hard enough", that any model that does well on it, should learn something "meaningful" about the task.

  Doersch et al. '15: Unsupervised Visual Representation Learning by Context Prediction

- **Quite finnicky**: one needs to make sure the predictor cannot take any obvious "shortcuts".

  - Boundary texture continuity is a big clue: include gaps in tiles.
  - Long lines spanning tiles are a clue: jitter location of tiles.
  - Chromatic aberration (some cameras tend to focus different wavelengths at different position – e.g. green shifts towards center of image): randomly drop 2 of the 3 channels

Predicting rotations

- In principle, what we want is a task "hard enough", that any model that does well on it, should learn something "meaningful" about the task.

  Gidaris et al. '18: Unsupervised representation learning via predicting image rotations

- Task: predict one of 4 possible rotations of an image.

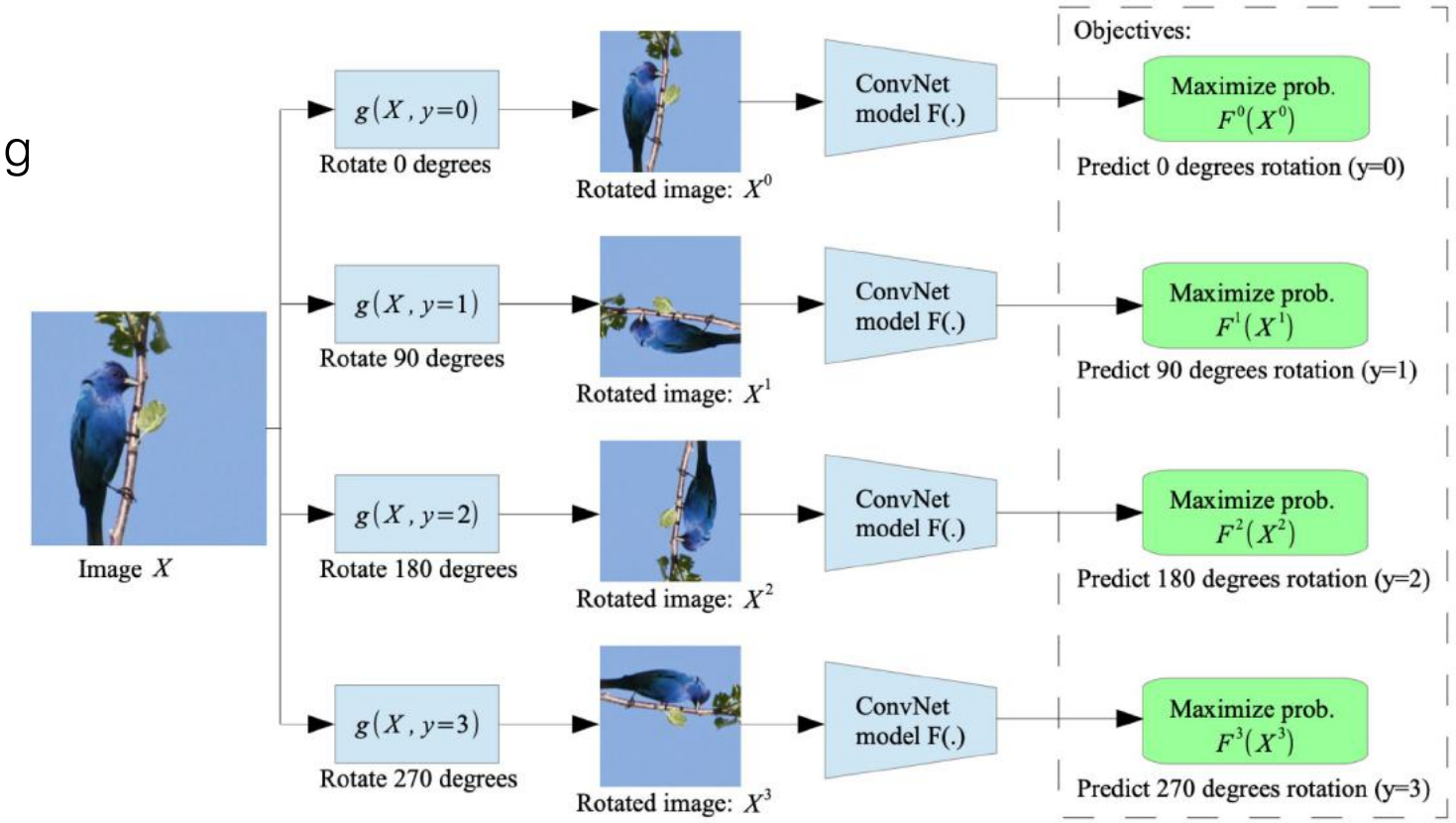


Figure 2: Illustration of the self-supervised task that we propose for semantic feature learning. Given four possible geometric transformations, the 0, 90, 180, and 270 degrees rotations, we train a ConvNet model $F(.)$ to recognize the rotation that is applied to the image that it gets as input. $F^y(X^{y^*})$ is the probability of rotation transformation y predicted by model $F(.)$ when it gets as input an image that has been transformed by the rotation transformation y^*.

Predicting rotations

- In principle, what we want is a task "hard enough", that any model that does well on it, should learn something "meaningful" about the task.

  Gidaris et al. '18: Unsupervised representation learning via predicting image rotations

- **Task**: predict one of 4 possible rotations of an image.
  - **Representation**: penultimate layer of a neural net used to solve task.
  - **Intuition**: a rotation is a global transformation. ConvNets are much better at capturing local transformations (as convolutions are local), so there is no obvious way to "cheat".

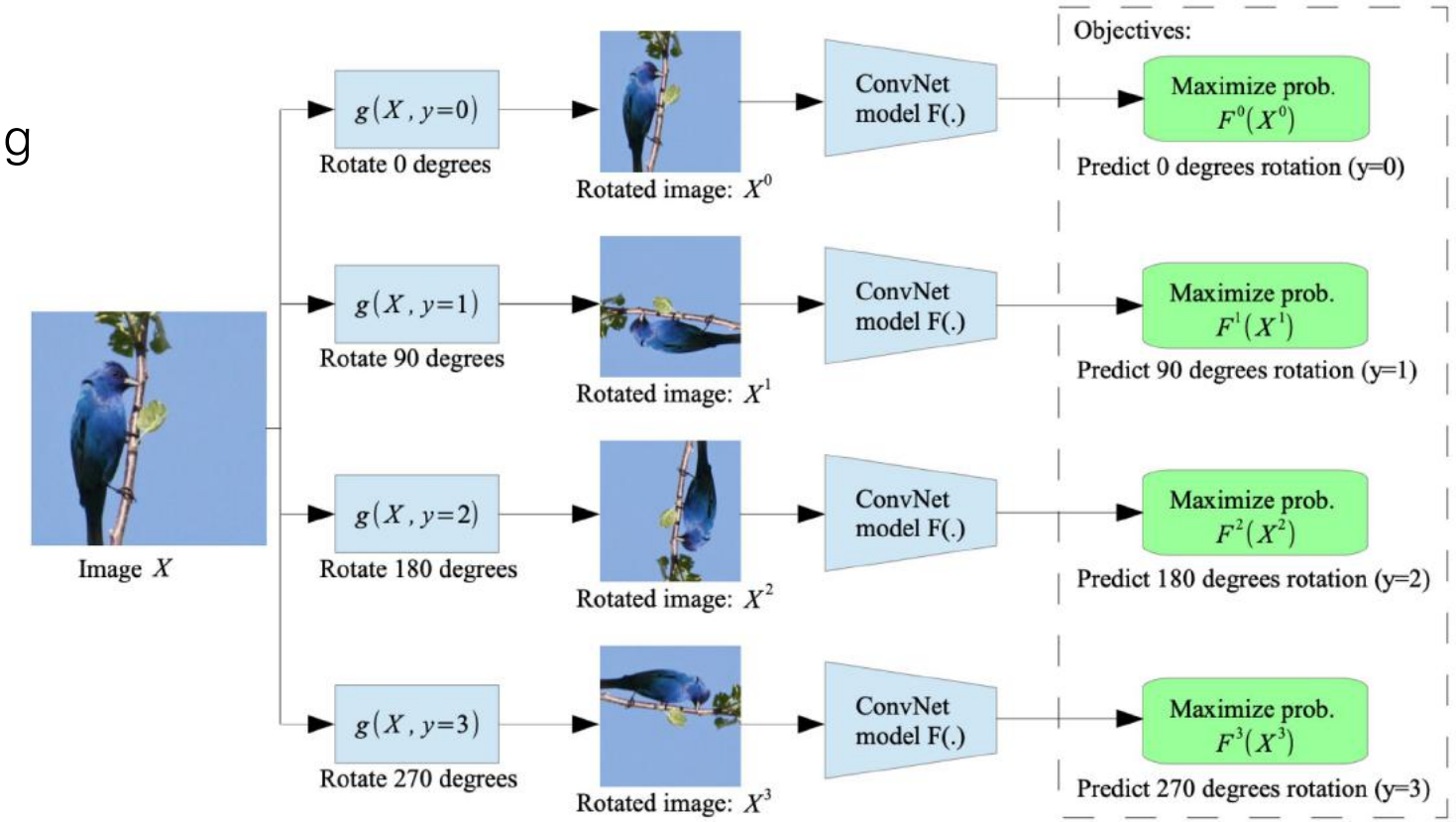


Figure 2: Illustration of the self-supervised task that we propose for semantic feature learning. Given four possible geometric transformations, the 0, 90, 180, and 270 degrees rotations, we train a ConvNet model $F(.)$ to recognize the rotation that is applied to the image that it gets as input. $F^y(X^{y^*})$ is the probability of rotation transformation y predicted by model $F(.)$ when it gets as input an image that has been transformed by the rotation transformation y^*.

Predicting rotations

- In principle, what we want is a task "hard enough", that any model that does well on it, should learn something "meaningful" about the task.

  Gidaris et al. '18: Unsupervised representation learning via predicting image rotations

- Task: predict one of 4 possible rotations of an image.
  - Less finicky to get right: no obvious artifacts the model can make use of to cheat.
  - The 90 deg. rotations also don't introduce any additional artifacts due to discretization.

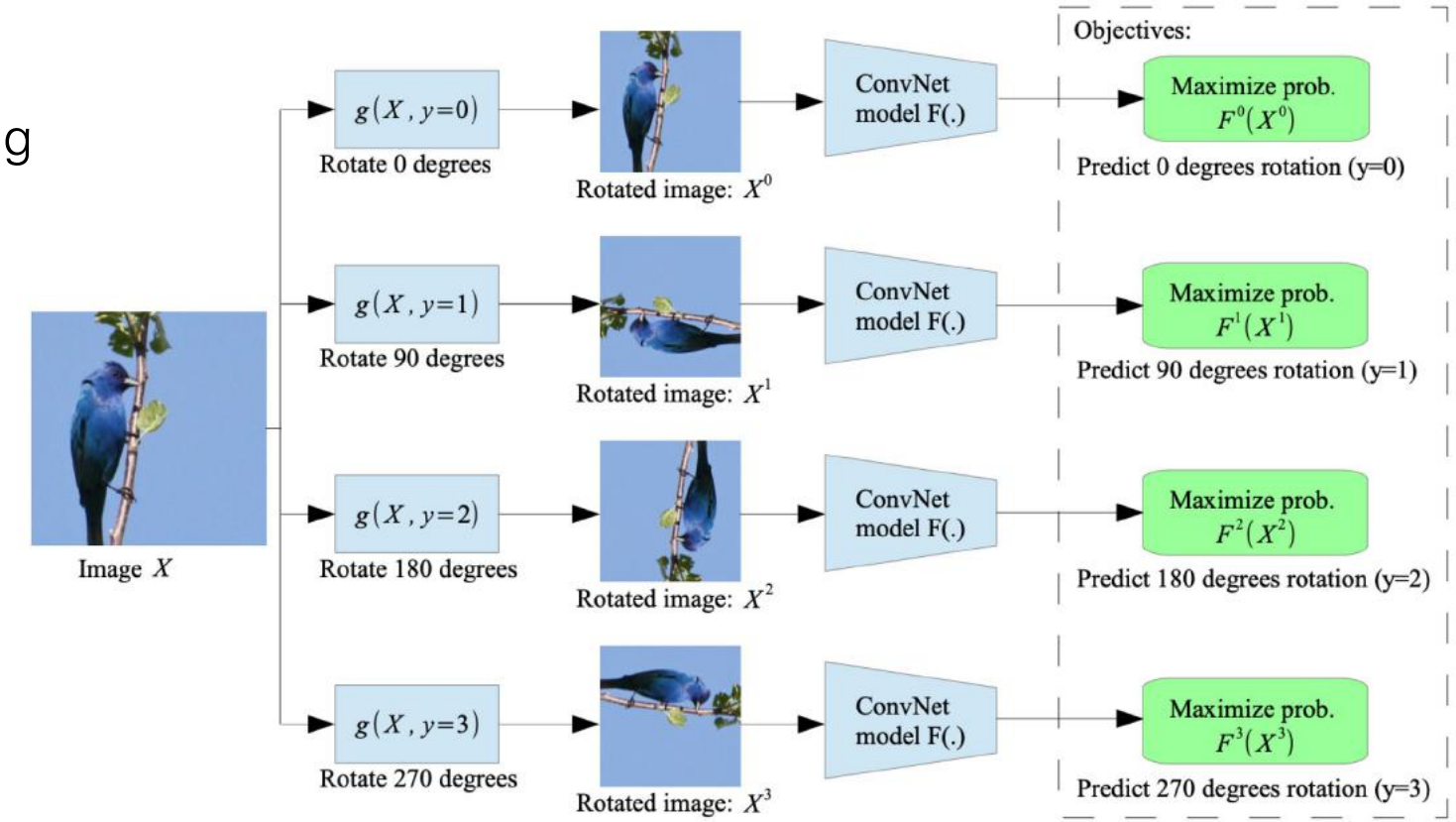


Figure 2: Illustration of the self-supervised task that we propose for semantic feature learning. Given four possible geometric transformations, the 0, 90, 180, and 270 degrees rotations, we train a ConvNet model $F(.)$ to recognize the rotation that is applied to the image that it gets as input. $F^y(X^{y^*})$ is the probability of rotation transformation y predicted by model $F(.)$ when it gets as input an image that has been transformed by the rotation transformation y^*.

Contrastive divergence

- Another natural idea: if features are "semantically" relevant, a "distortion" of an image should produce similar features. Some instances of distortions:

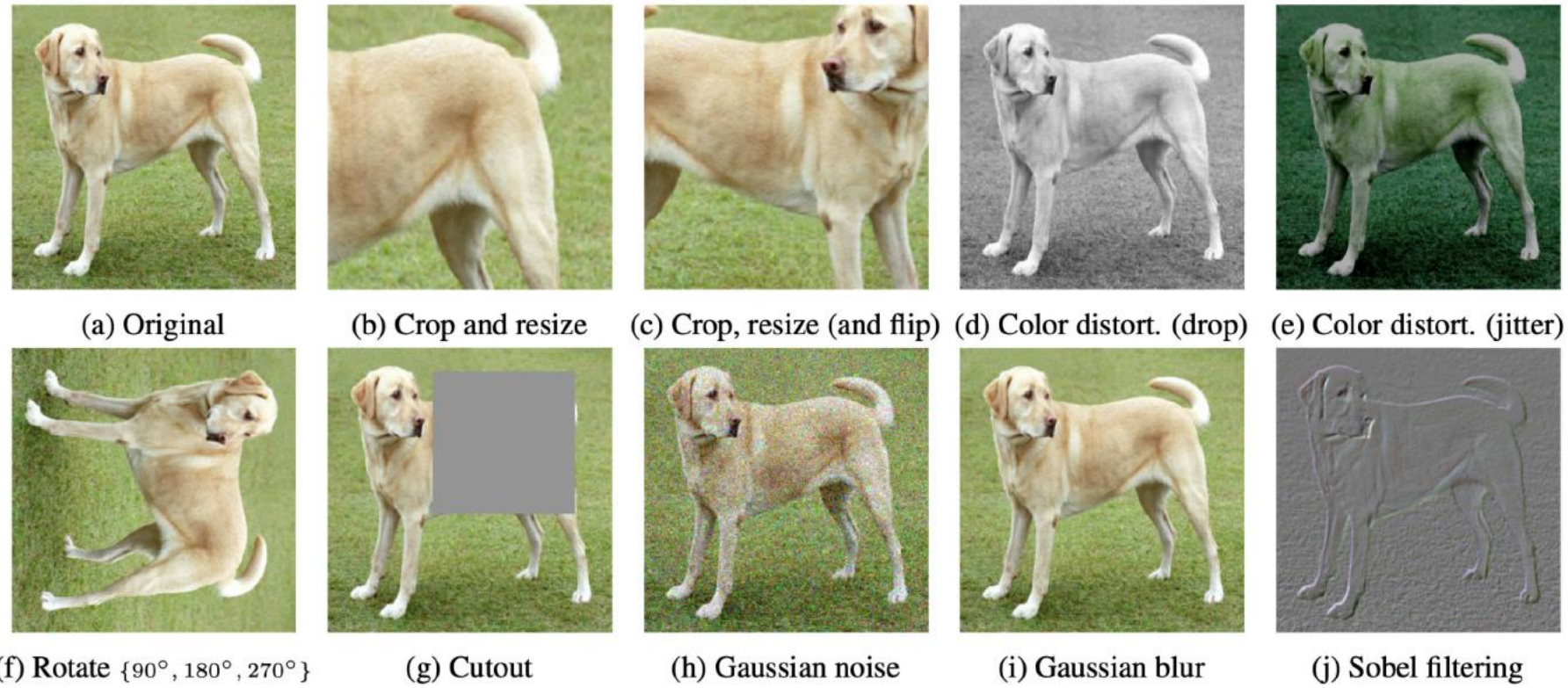


Figure 4. Illustrations of the studied data augmentation operators. Each augmentation can transform data stochastically with some internal parameters (e.g. rotation degree, noise level). Note that we *only* test these operators in ablation, the *augmentation policy used to train our models* only includes *random crop (with flip and resize)*, *color distortion*, and *Gaussian blur*. (Original image cc-by: Von.grzanka)

Contrastive divergence

- Another natural idea: if features are "semantically" relevant, a "distortion" of an image should produce similar features. Some instances of distortions:
- **Contrastive divergence framework**:

  For every training sample, produce multiple augmented samples by applying various transformations.

  Train an encoder E (i.e. map that produces features) to predict whether two samples are augmentations of the same base sample.

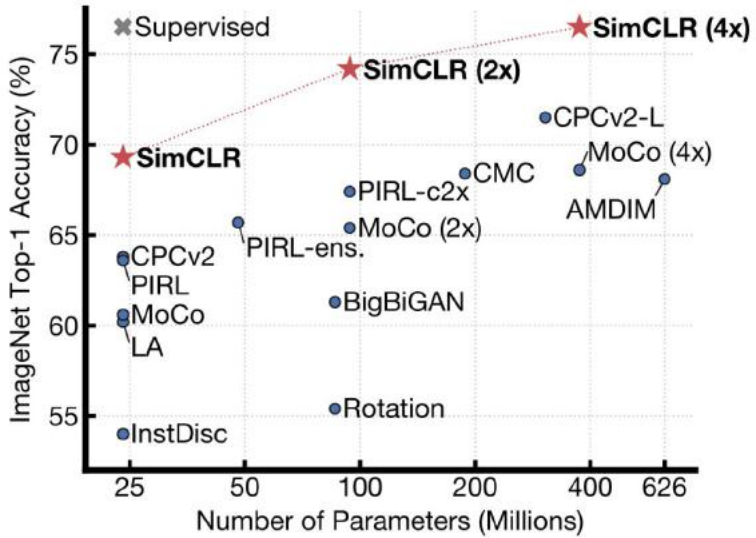
  A common way is to train E to make $\langle E(x)\ , E(x')\rangle$ big if x, x' are two augmentations from same sample, small otherwise, e.g.

$$l_{x,x'} = -\log\left(\frac{\exp\left(\tau\left\langle E(x), E\left(x'\right)\right\rangle\right.}{\sum_{x,x'}\exp(\tau\langle E(x), E(x')\rangle)}\right)$$

$$\min \sum_{x,x' \text{ augments of each other}} l_{x,x'}$$

Contrastive divergence

- Another natural idea: if features are "semantically" relevant, a "distortion" of an image should produce similar features. Some instances of distortions:
- Many works follow this framework, starting with Oord '18: Representation Learning with Contrastive Predictive Learning.
- Current state of the art for self-supervised learning is in fact using this framework: Chen, Kornblith, Norouizi, Hinton '20: A Simple Framework for Contrastive Learning of Visual Representations



Several tricks needed to gain this improvement.

Most important one seems to be that augmentations that work best are compositions of a geometric one (e.g. crop/rotation/..) and an appearance one (color distortion/blur/..)

Troubling fact: Architecture Matters

- Kolesnikov et al. '19: Revisiting Self-Supervised Visual Representation Learning

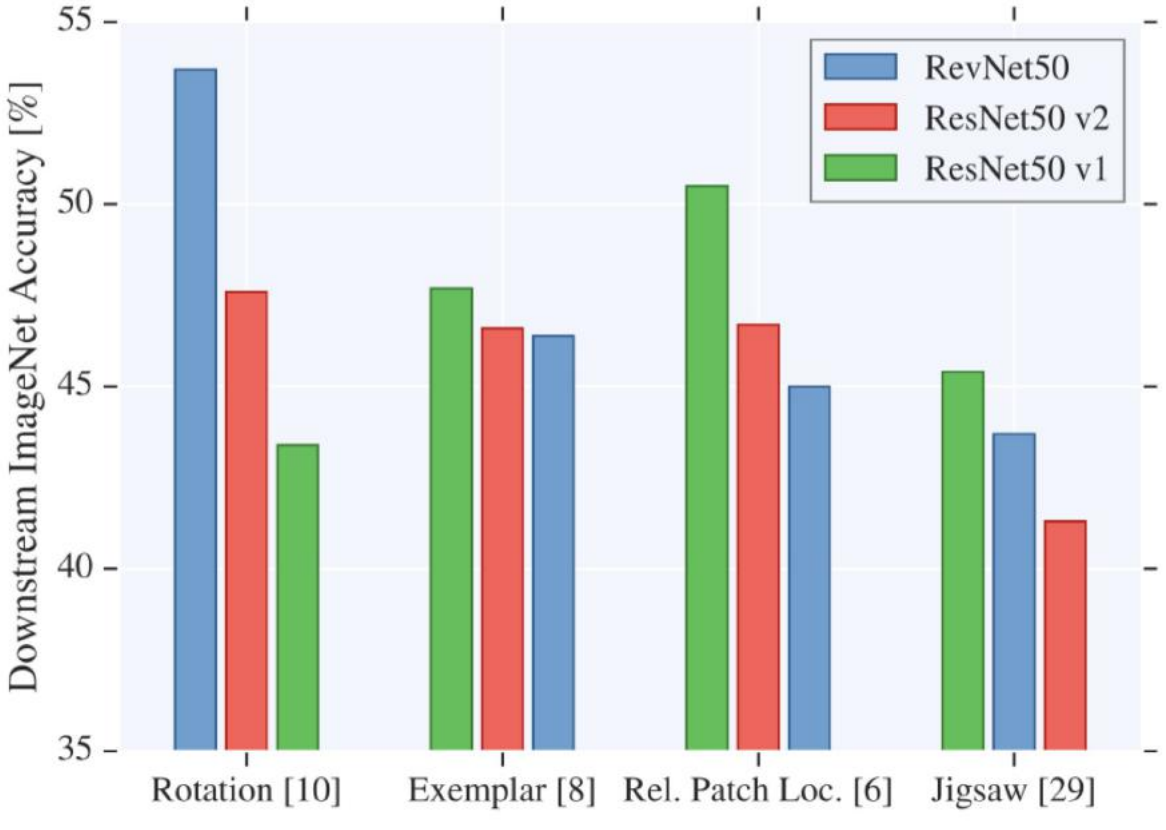


Figure 1. Quality of visual representations learned by various self-supervised learning techniques significantly depends on the convolutional neural network architecture that was used for solving the self-supervised learning task. For instance, *ResNet50 v1* excels when trained with the *relative patch location* self-supervision [7], but produces suboptimal results when trained with the *rotation* self-supervision [11]. In our paper we provide a large scale in-depth study in support of this observation and discuss its implications for evaluation of self-supervised models.

Troubling fact: Architecture Matters

- Kolesnikov et al. '19: Revisiting Self-Supervised Visual Representation Learning

Table 1. Evaluation of representations from self-supervised techniques based on various CNN architectures. The scores are accuracies (in %) of a linear logistic regression model trained on top of these representations using *ImageNet* training split. Our validation split is used for computing accuracies. The architectures marked by a "(-)" are slight variations described in Section 3.1. Sub-columns such as 4× correspond to widening factors. Top-performing architectures in a column are bold; the best pretext task for each model is underlined.

Model	Rnd	Rotation				Exemplar			RelPatchLoc		Jigsaw	
	4×	4×	8×	12×	16×	4×	8×	12×	4×	8×	4×	8×
RevNet50	**8.1**	**47.3**	**50.4**	**53.1**	**53.7**	**42.4**	45.6	46.4	40.6	45.0	40.1	43.7
ResNet50 v2	4.4	43.8	47.5	47.2	47.6	**43.0**	45.7	46.6	42.2	46.7	38.4	41.3
ResNet50 v1	2.5	41.7	43.4	43.3	43.2	**42.8**	**46.9**	**47.7**	**46.8**	**50.5**	**42.2**	**45.4**
RevNet50 (-)	**8.4**	45.2	**51.0**	**52.8**	**53.7**	38.0	42.6	44.3	33.8	43.5	36.1	41.5
ResNet50 v2 (-)	5.3	38.6	44.5	47.3	48.2	33.7	36.7	38.2	38.6	43.4	32.5	34.4
VGG19-BN	2.7	16.8	14.6	16.6	22.7	26.4	28.3	29.0	28.5	29.4	19.8	21.1