10-301/601: Introduction to Machine Learning Lecture 25 – Pretraining, Fine-tuning & In-Context Learning

Henry Chai

6/9/25

Front Matter

- Announcements:
  - HW6 released on 6/6, due 6/10 (tomorrow) at 11:59 PM
  - HW7 to be released on 6/10 (tomorrow), due 6/13 at 11:59 PM
  - Thursday’s lecture will be a guest lecture by Alex Xie on Reinforcement Learning for LLMs
    - This content will not be covered on the quiz but…
    - **Everyone who attends** (and stays for the duration of the lecture) **will have their lowest quiz grade down-weighted by 50%**

Recall: PCA Algorithm

- Input: $\mathcal{D} = \left\{\left(\boldsymbol{x}^{(n)}\right)\right\}_{n=1}^{N}, \rho$

1. Center the data
2. Use SVD to compute the eigenvalues and eigenvectors of $X^T X$
3. Collect the top ρ eigenvectors (corresponding to the ρ largest eigenvalues), $V_\rho \in \mathbb{R}^{D \times \rho}$
4. Project the data into the space defined by V_ρ, $Z = XV_\rho$

- Output: Z, the transformed (potentially lower-dimensional) data

PCA Example: MNIST Digits

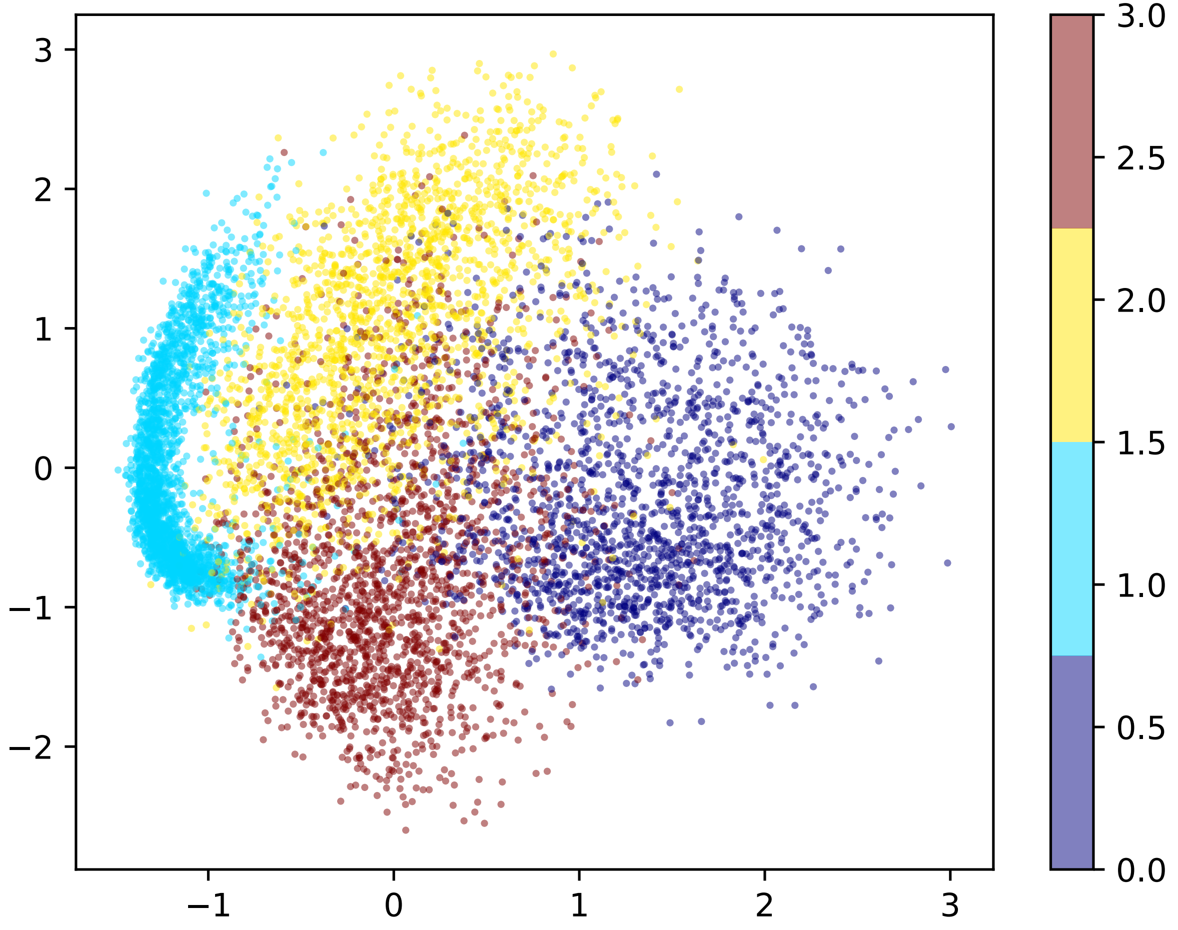
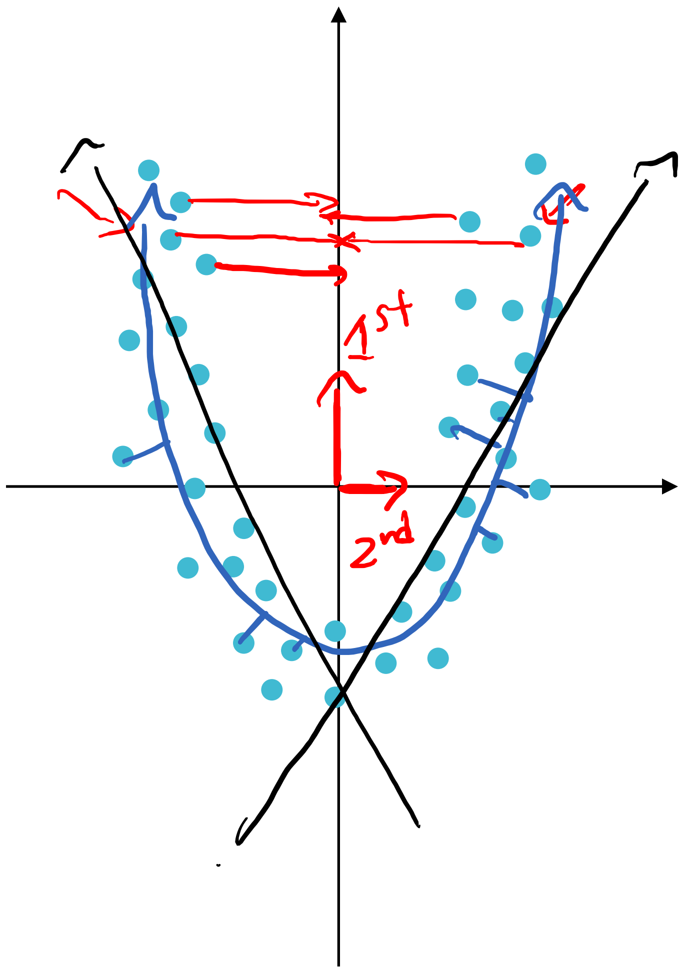


Figure courtesy of Matt Gormley

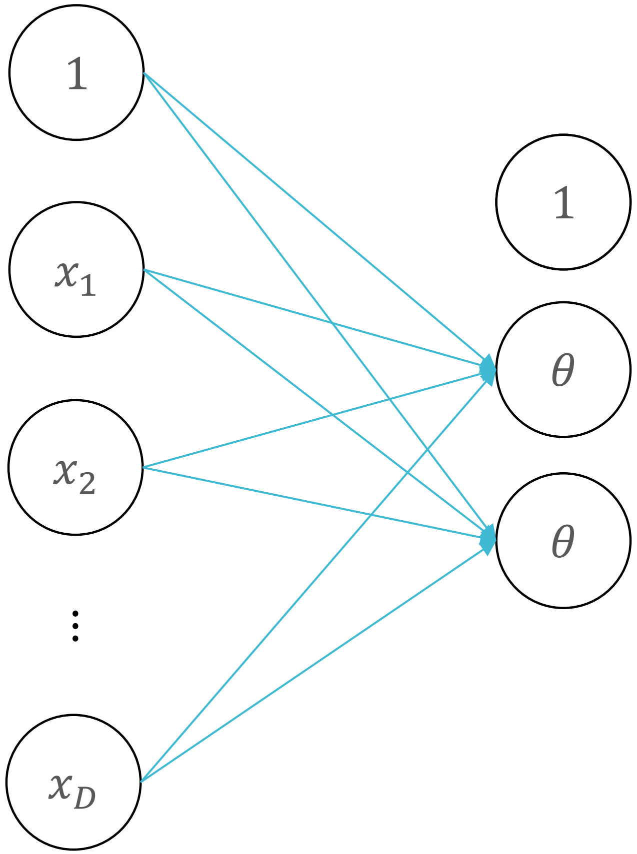
Shortcomings of PCA



- principal components are linear in the features

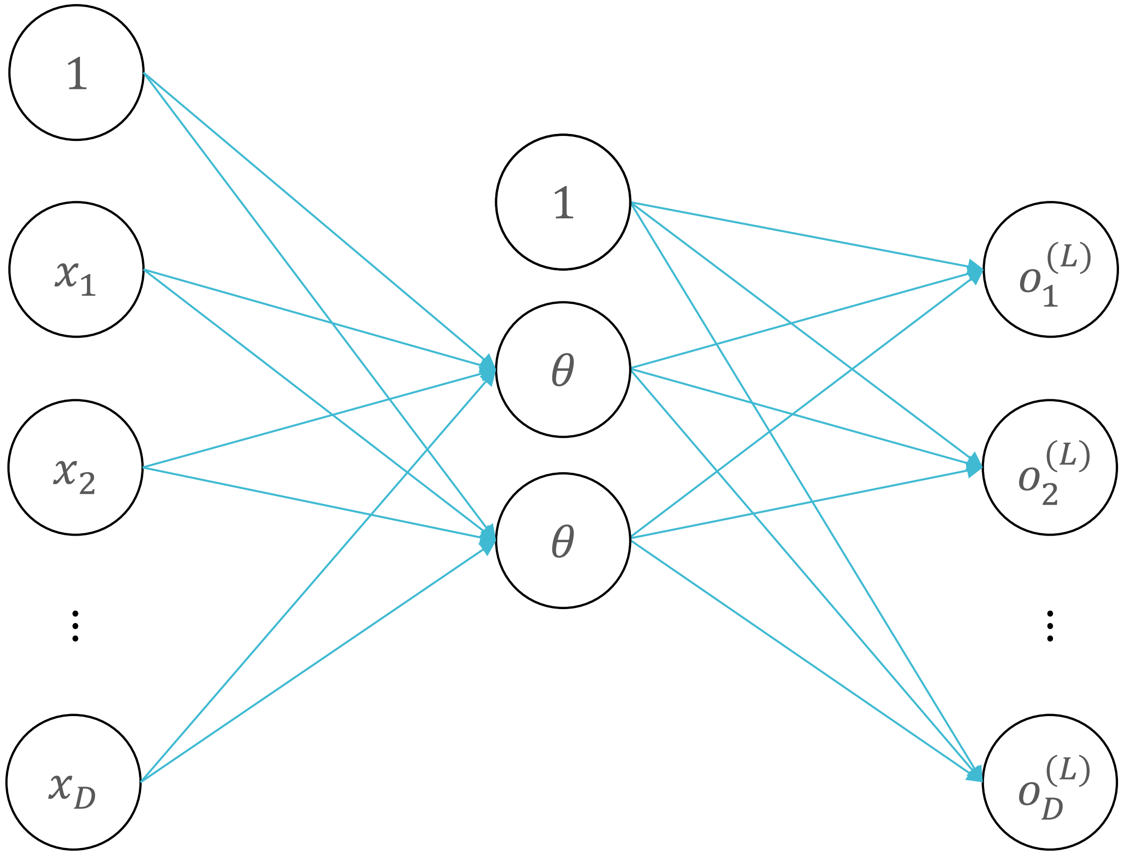
- principal components have to be orthogonal

Autoencoders



Insight: neural networks implicitly learn low-dimensional representations of inputs in hidden layers

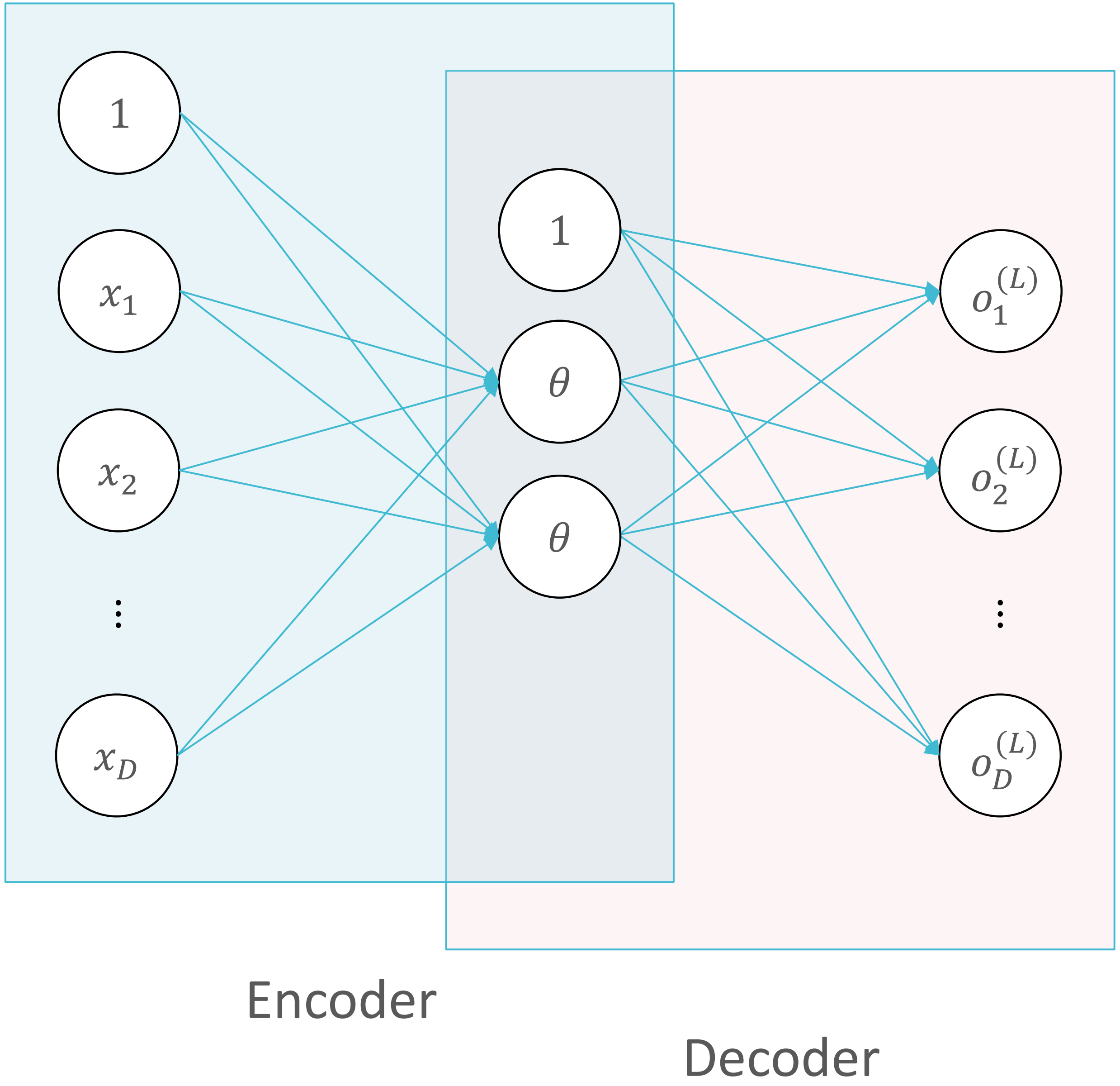
Autoencoders



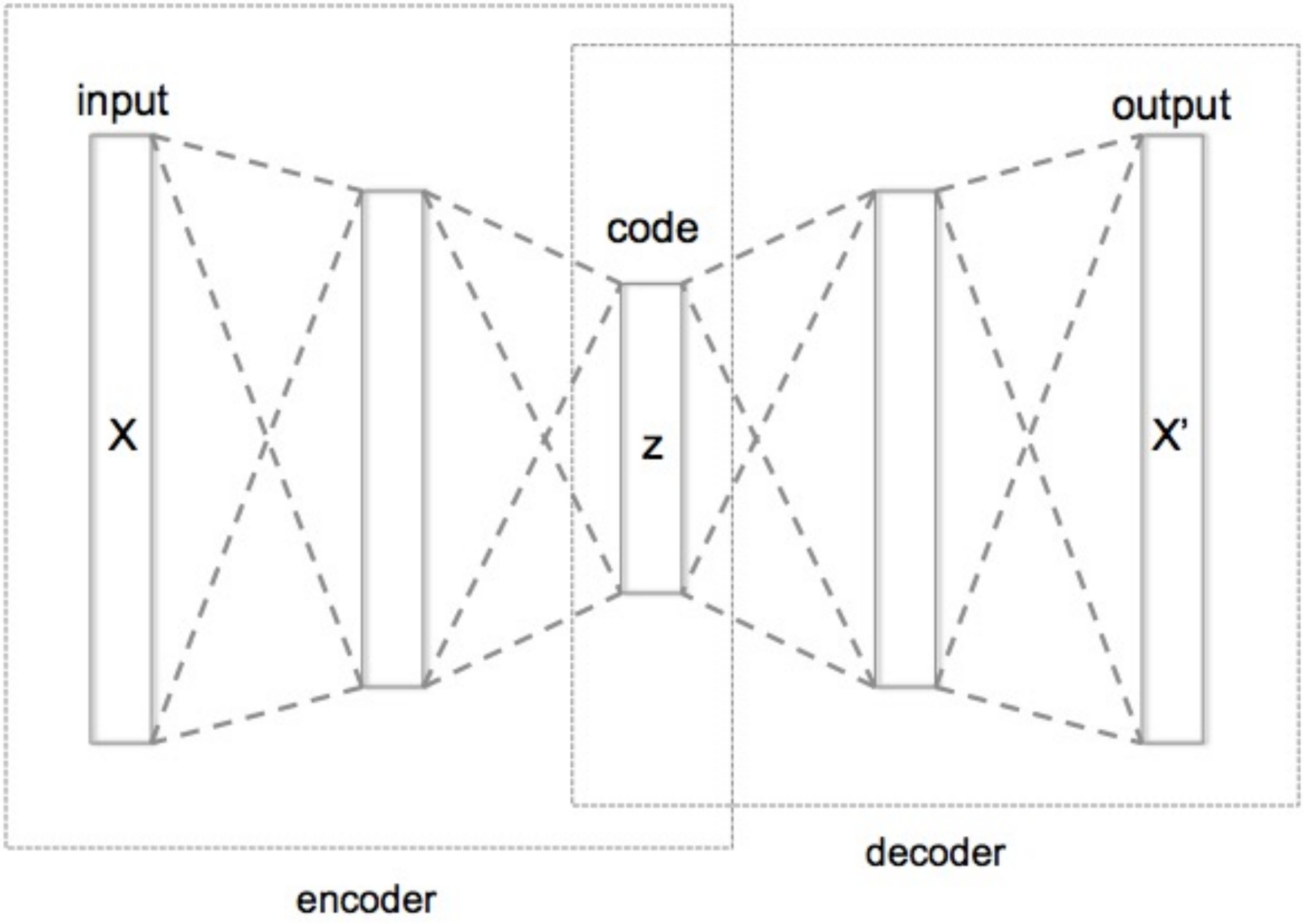
- Learn the weights by minimizing the reconstruction loss:

$$e(\boldsymbol{x}) = \left\|\boldsymbol{x} - \boldsymbol{o}^{(L)}\right\|_2^2$$

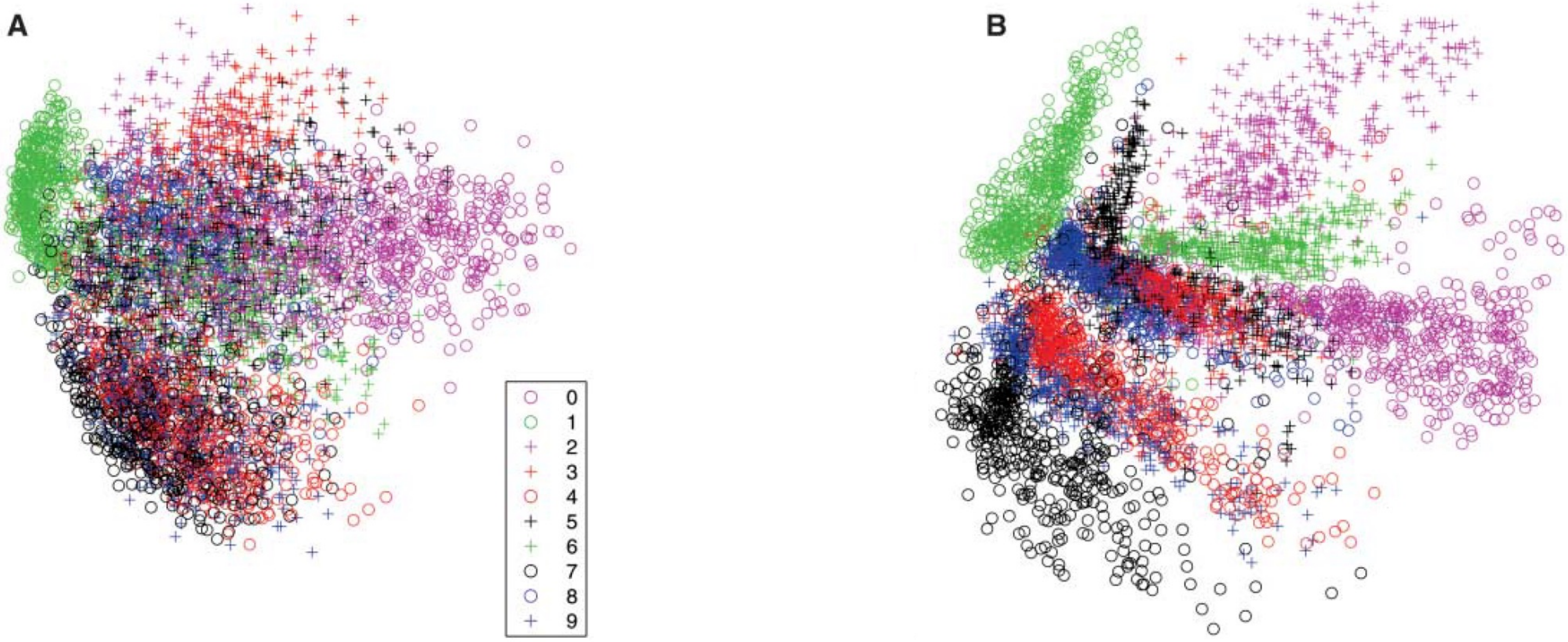
Autoencoders



Deep Autoencoders



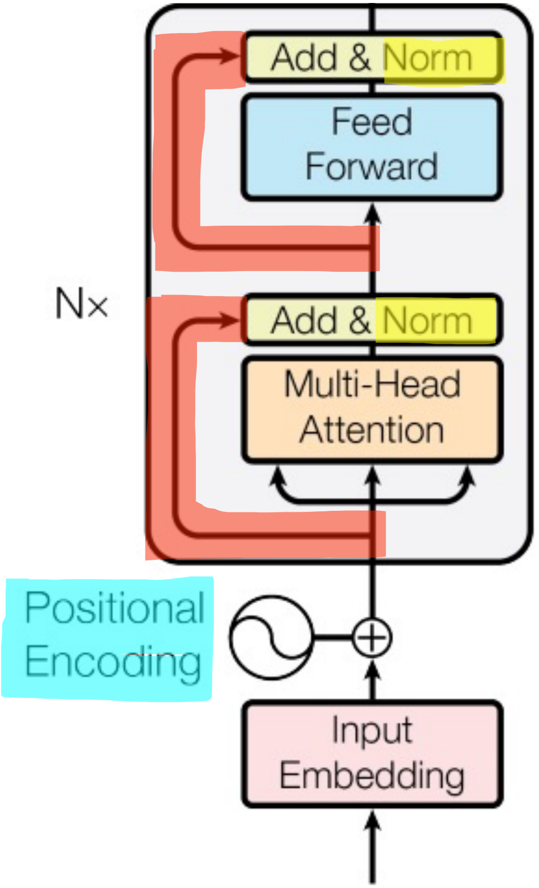
Source: https://en.wikipedia.org/wiki/Autoencoder#/media/File:Autoencoder_structure.png



PCA (A) vs. Autoencoders (B) (Hinton and Salakhutdinov, 2006)

Source: https://www.science.org/doi/10.1126/science.1127647

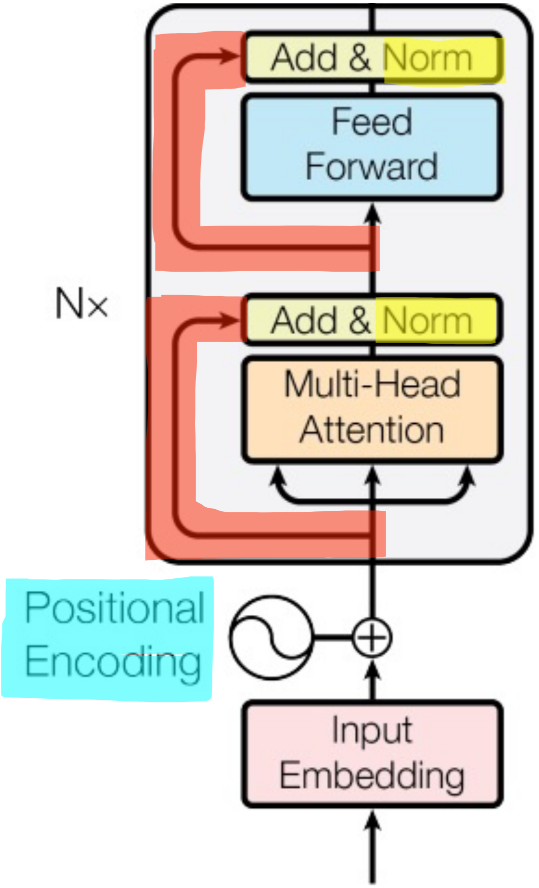
Recall: Transformers



- In addition to multi-head attention, transformer architectures use
  1. Positional encodings
  2. Layer normalization
  3. Residual connections
  4. A fully-connected feed-forward network

Source: https://arxiv.org/pdf/1706.03762.pdf

Okay, but how on earth do we go about training these things?



- In addition to multi-head attention, transformer architectures use
    1. Positional encodings
    2. Layer normalization
    3. Residual connections
    4. A fully-connected feed-forward network

Source: https://arxiv.org/pdf/1706.03762.pdf

Recall: Mini-batch Stochastic Gradient Descent…

- Input: $\mathcal{D} = \left\{\left(\boldsymbol{x}^{(n)}, y^{(n)}\right)\right\}_{n=1}^{N}, \eta_{MB}^{(0)}, B$

1. Initialize all weights $W_{(0)}^{(1)}, \dots, W_{(0)}^{(L)}$ to small, random numbers and set $t = 0$
2. While TERMINATION CRITERION is not satisfied
   a. Randomly sample B data points from $\mathcal{D}$, $\left\{\left(\boldsymbol{x}^{(b)}, y^{(b)}\right)\right\}_{b=1}^{B}$
   b. Compute the gradient of the loss w.r.t. the sampled *batch*,
   $$G^{(l)} = \frac{1}{B}\sum_{b=1}^{B} \nabla_{W^{(l)}} \ell^{(b)}\left(W_{(t)}^{(1)}, \dots, W_{(t)}^{(L)}\right) \;\forall\, l$$
   c. Update $W^{(l)}$: $W_{t+1}^{(l)} \leftarrow W_{t}^{(l)} - \eta_{MB}^{(0)} G^{(l)} \;\forall\, l$
   d. Increment t: $t \leftarrow t + 1$

- Output: $W_t^{(1)}, \dots, W_t^{(L)}$

Mini-batch Stochastic Gradient Descent is a lie!

- Input: $\mathcal{D} = \left\{\left(\boldsymbol{x}^{(n)}, y^{(n)}\right)\right\}_{n=1}^{N}, \eta_{MB}^{(0)}, B$

1. Initialize all weights $W_{(0)}^{(1)}, \dots, W_{(0)}^{(L)}$ to small, random numbers and set $t = 0$
2. While TERMINATION CRITERION is not satisfied
   a. Randomly sample B data points from $\mathcal{D}$, $\left\{\left(\boldsymbol{x}^{(b)}, y^{(b)}\right)\right\}_{b=1}^{B}$
   b. Compute the gradient of the loss w.r.t. the sampled *batch*,
   $$G^{(l)} = \frac{1}{B}\sum_{b=1}^{B} \nabla_{W^{(l)}} \ell^{(b)}\left(W_{(t)}^{(1)}, \dots, W_{(t)}^{(L)}\right) \ \forall\, l$$
   c. Update $W^{(l)}$: $W_{t+1}^{(l)} \leftarrow W_t^{(l)} - \eta_{MB}^{(0)} G^{(l)} \ \forall\, l$
   d. Increment t: $t \leftarrow t + 1$

- Output: $W_t^{(1)}, \dots, W_t^{(L)}$

Mini-batch Stochastic Gradient Descent is ~~a lie!~~ just the beginning!

- Input: $\mathcal{D} = \left\{\left(\boldsymbol{x}^{(n)}, y^{(n)}\right)\right\}_{n=1}^{N}, \eta_{MB}^{(0)}, B$

1. Initialize all weights $W_{(0)}^{(1)}, \ldots, W_{(0)}^{(L)}$ to small, random numbers and set $t = 0$
2. While TERMINATION CRITERION is not satisfied
   a. Randomly sample B data points from $\mathcal{D}$, $\left\{\left(\boldsymbol{x}^{(b)}, y^{(b)}\right)\right\}_{b=1}^{B}$
   b. Compute the gradient of the loss w.r.t. the sampled *batch*,
   $$G^{(l)} = \frac{1}{B} \sum_{b=1}^{B} \nabla_{W^{(l)}} \ell^{(b)} \left(W_{(t)}^{(1)}, \ldots, W_{(t)}^{(L)}\right) \ \forall\, l$$
   c. Update $W^{(l)}$: $W_{t+1}^{(l)} \leftarrow W_{t}^{(l)} - \eta_{MB}^{(0)} G^{(l)} \ \forall\, l$
   d. Increment t: $t \leftarrow t + 1$

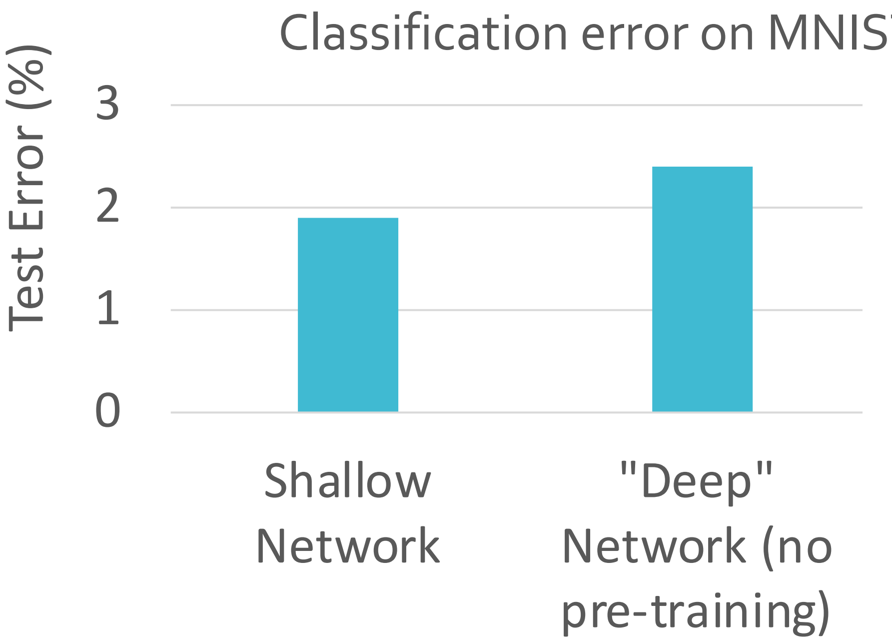
- Output: $W_{t}^{(1)}, \ldots, W_{t}^{(L)}$

Traditional Supervised Learning

- You have some task that you want to apply machine learning to
- You have a labelled dataset to train with
- You fit a deep learning model to the dataset

Reality

- You have some niche task that you want to apply machine learning to e.g., predicting how Henry will get to work
- You have a tiny labelled dataset to train with
- You fit a massive deep learning model to the dataset
- Surprise, surprise: it overfits and your test error is super high

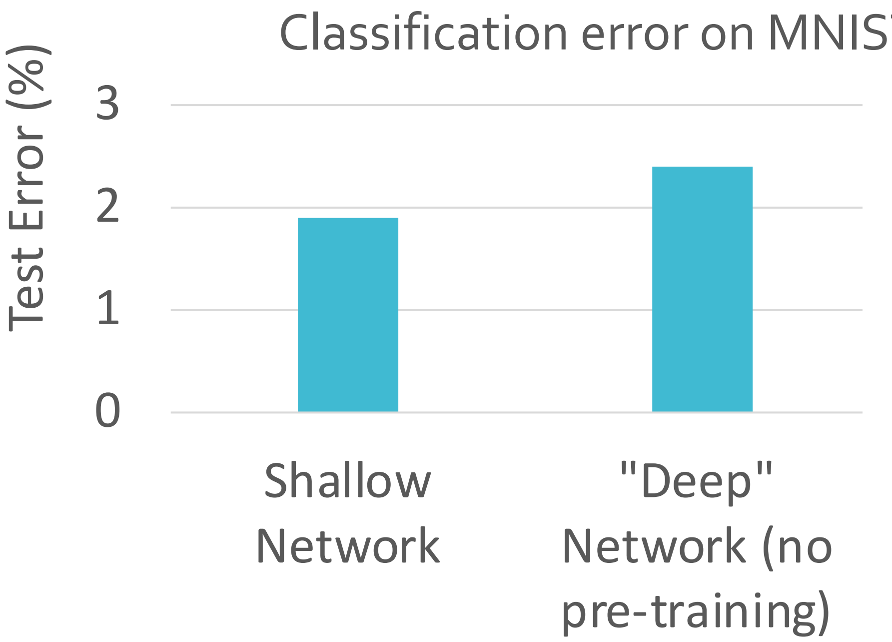


- "gradient-based optimization starting from random initialization appears to often get stuck in poor solutions for such deep networks."

Source: https://www.cs.toronto.edu/~larocheh/publications/dbn_supervised_tr1282.pdf

Reality

- You have some niche task that you want to apply machine learning to e.g., predicting how Henry will get to work
- You have a tiny labelled dataset to train with
- You fit a massive deep learning model to the dataset
- Surprise, surprise: it overfits and your test error is super high

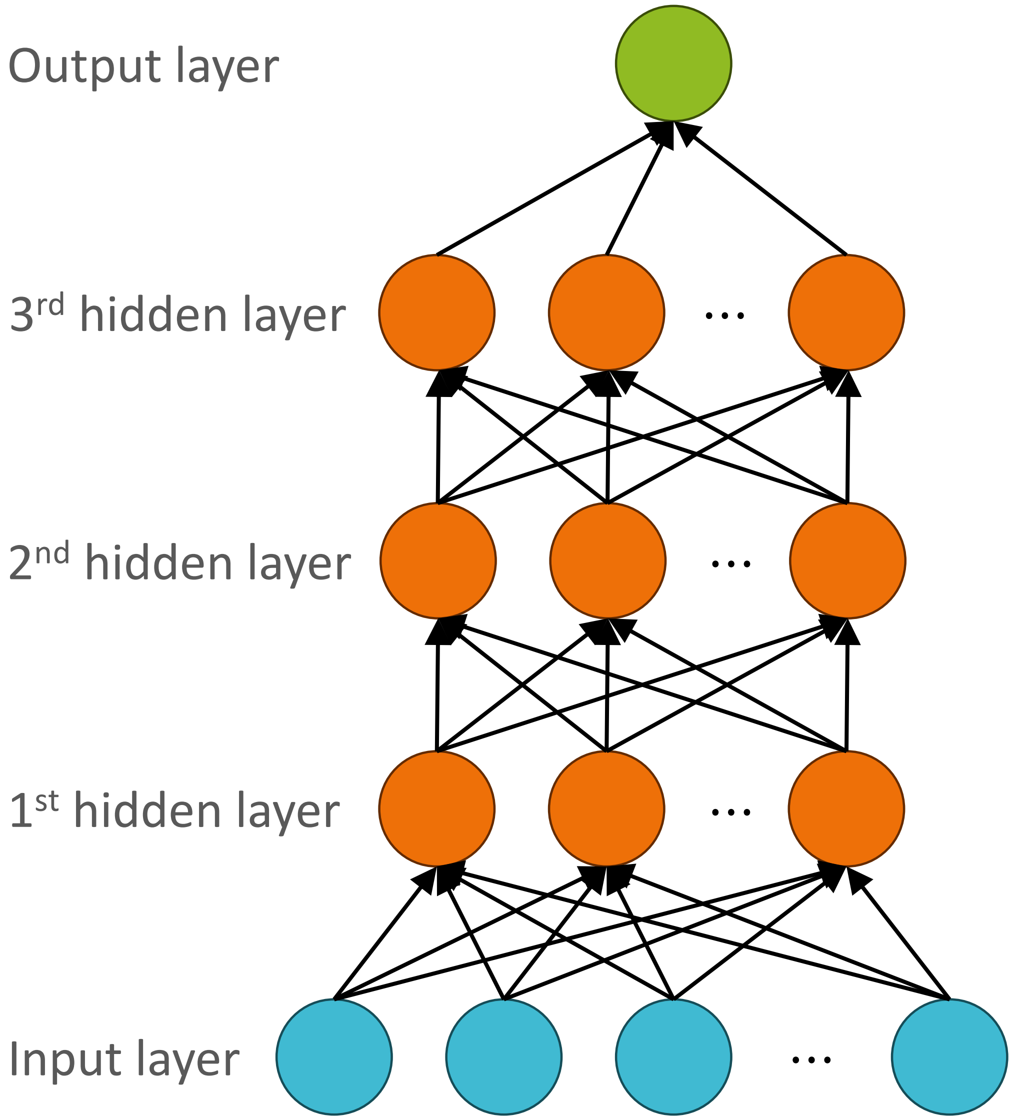


- Idea: if shallow networks are easier to train, let's just decompose our deep network into a series of shallow networks!

Source: https://www.cs.toronto.edu/~larocheh/publications/dbn_supervised_tr1282.pdf

Pre-training (Bengio et al., 2006)

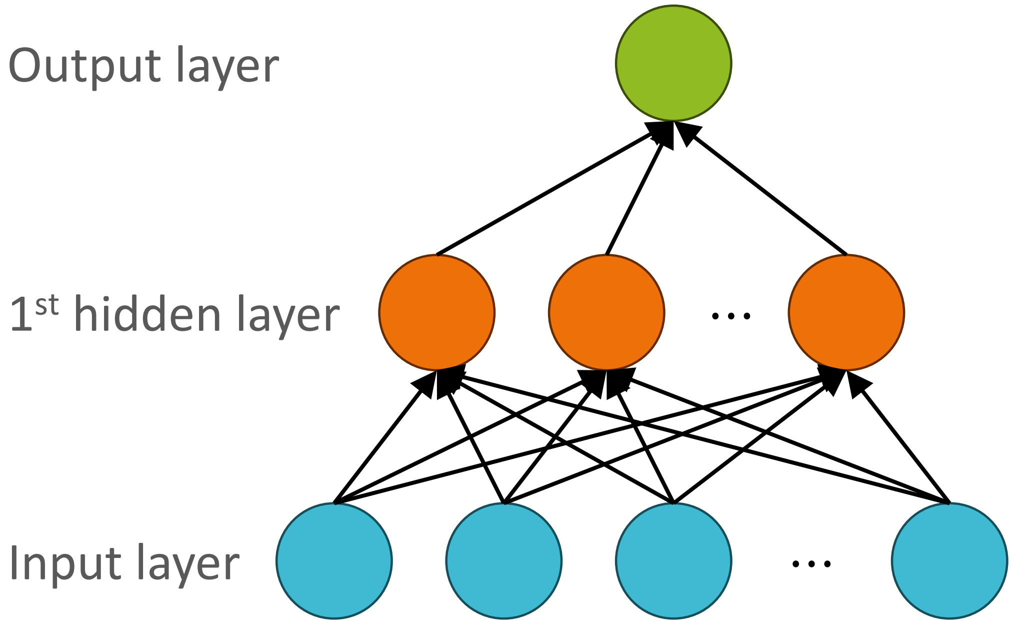
- Train each layer of the network iteratively using the training dataset
- Start at the input layer and move towards the output layer
- Once a layer has been trained, fix its weights and use those to train subsequent layers



Source: https://www.cs.toronto.edu/~larocheh/publications/dbn_supervised_tr1282.pdf

Pre-training (Bengio et al., 2006)

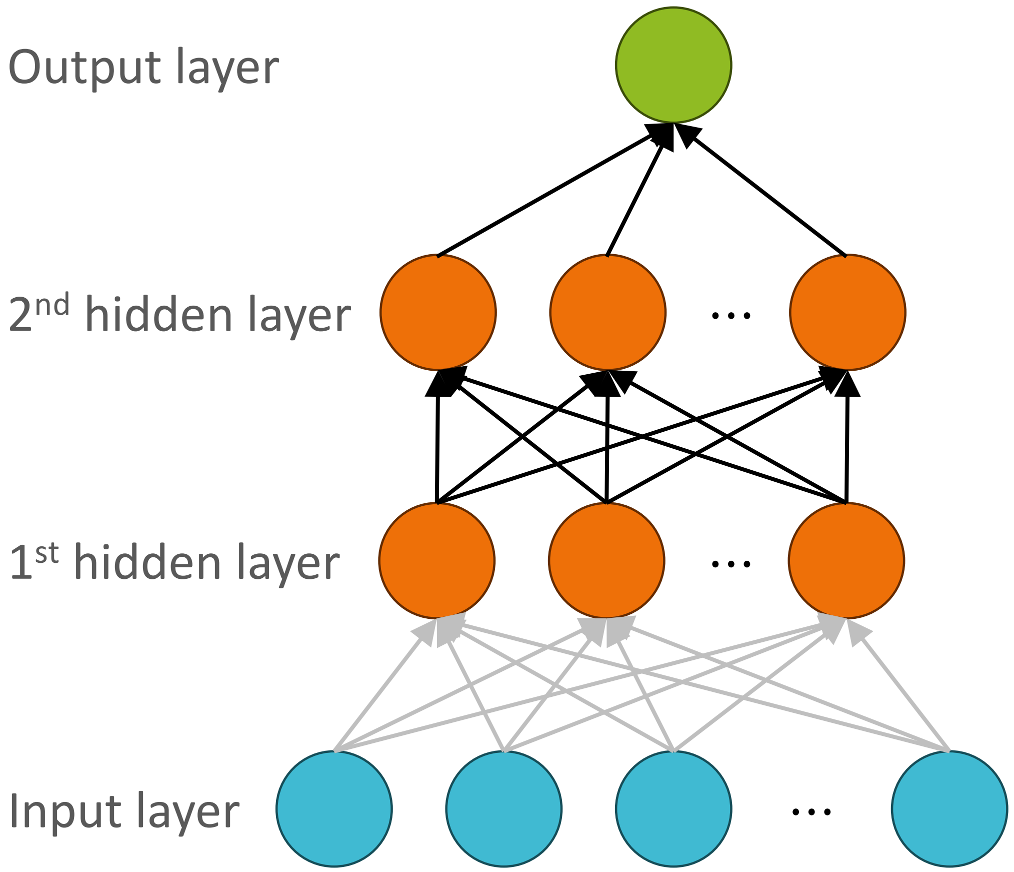
- Train each layer of the network iteratively using the training dataset
- Start at the input layer and move towards the output layer
- Once a layer has been trained, fix its weights and use those to train subsequent layers



Source: https://www.cs.toronto.edu/~larocheh/publications/dbn_supervised_tr1282.pdf

Pre-training (Bengio et al., 2006)

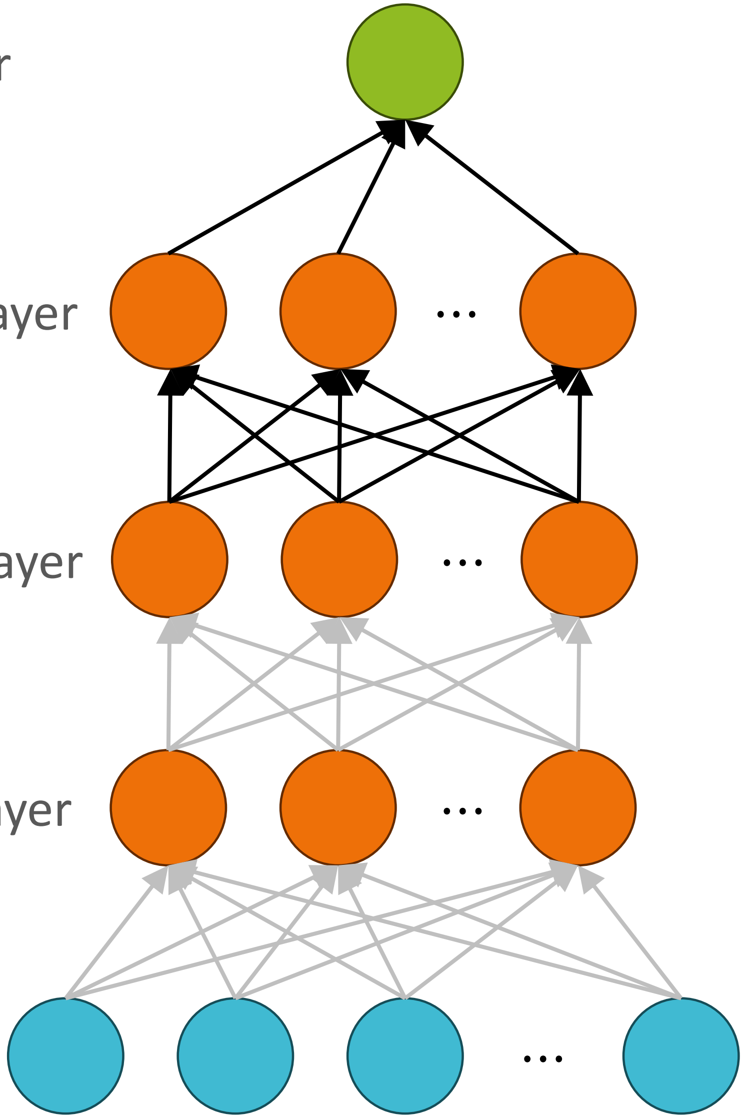
- Train each layer of the network iteratively using the training dataset
- Start at the input layer and move towards the output layer
- Once a layer has been trained, fix its weights and use those to train subsequent layers



Source: https://www.cs.toronto.edu/~larocheh/publications/dbn_supervised_tr1282.pdf

Pre-training (Bengio et al., 2006)

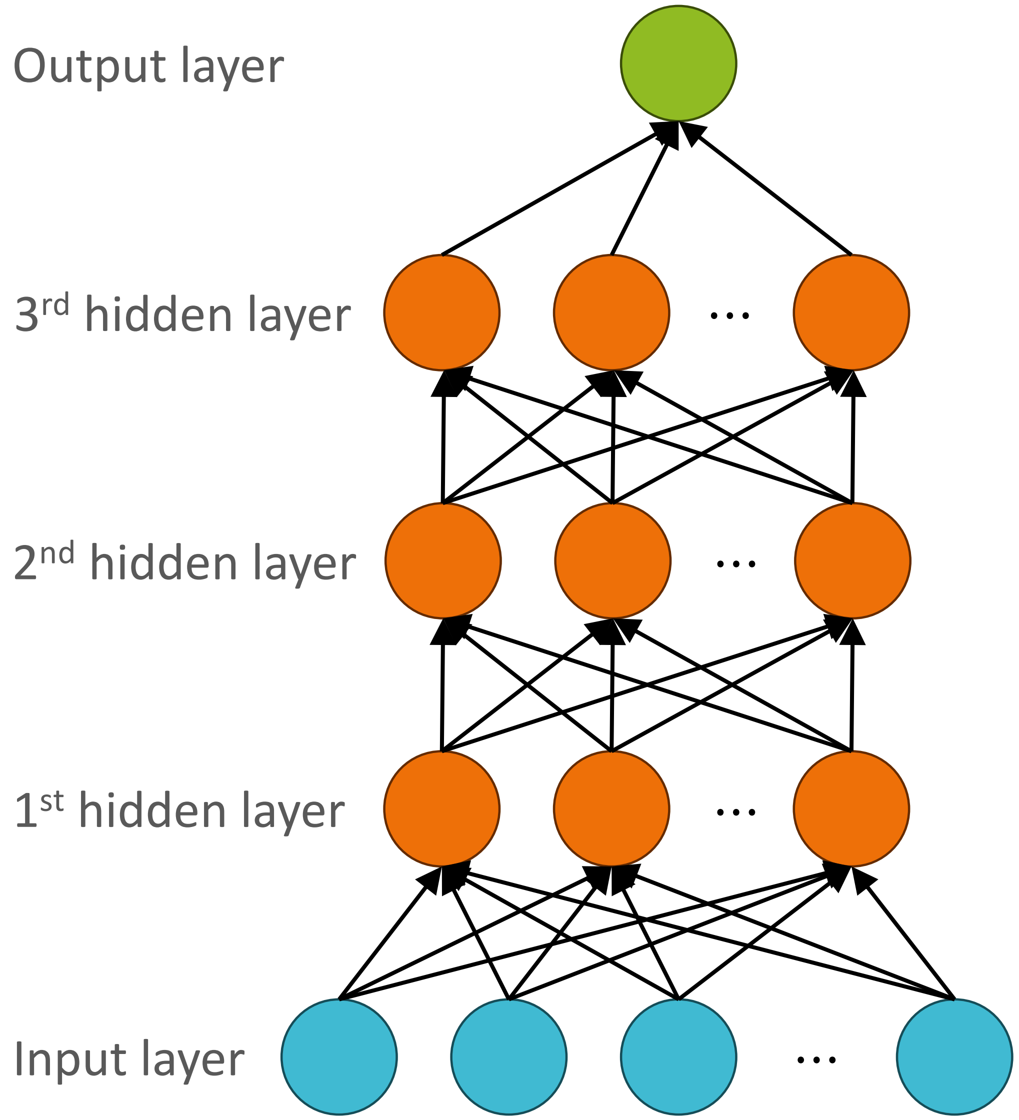
- Train each layer of the network iteratively using the training dataset
- Start at the input layer and move towards the output layer
- Once a layer has been trained, fix its weights and use those to train subsequent layers



Source: https://www.cs.toronto.edu/~larocheh/publications/dbn_supervised_tr1282.pdf

Fine-tuning (Bengio et al., 2006)

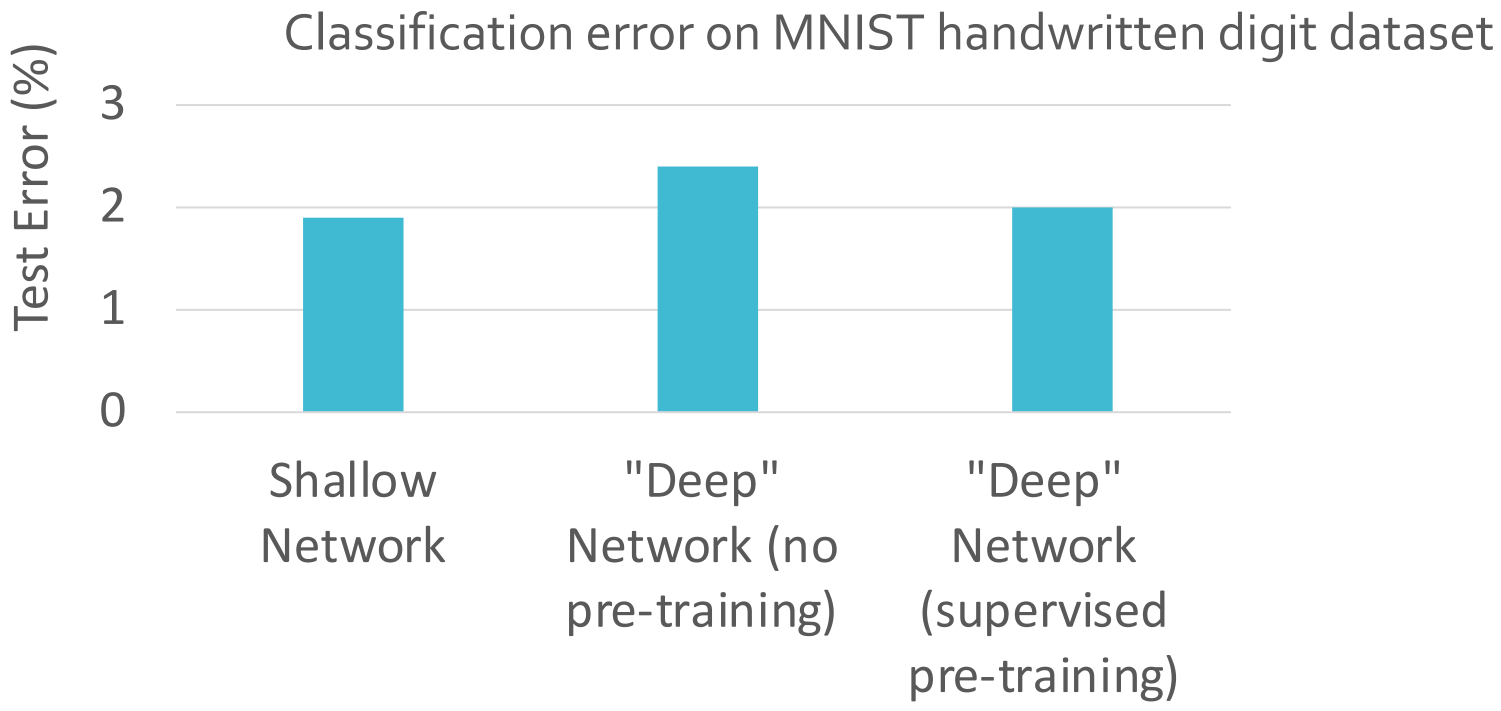
- Train each layer of the network iteratively using the training dataset
- Use the pre-trained weights as an initialization and *fine-tune* the entire network e.g., via SGD with the training dataset



Source: https://www.cs.toronto.edu/~larocheh/publications/dbn_supervised_tr1282.pdf

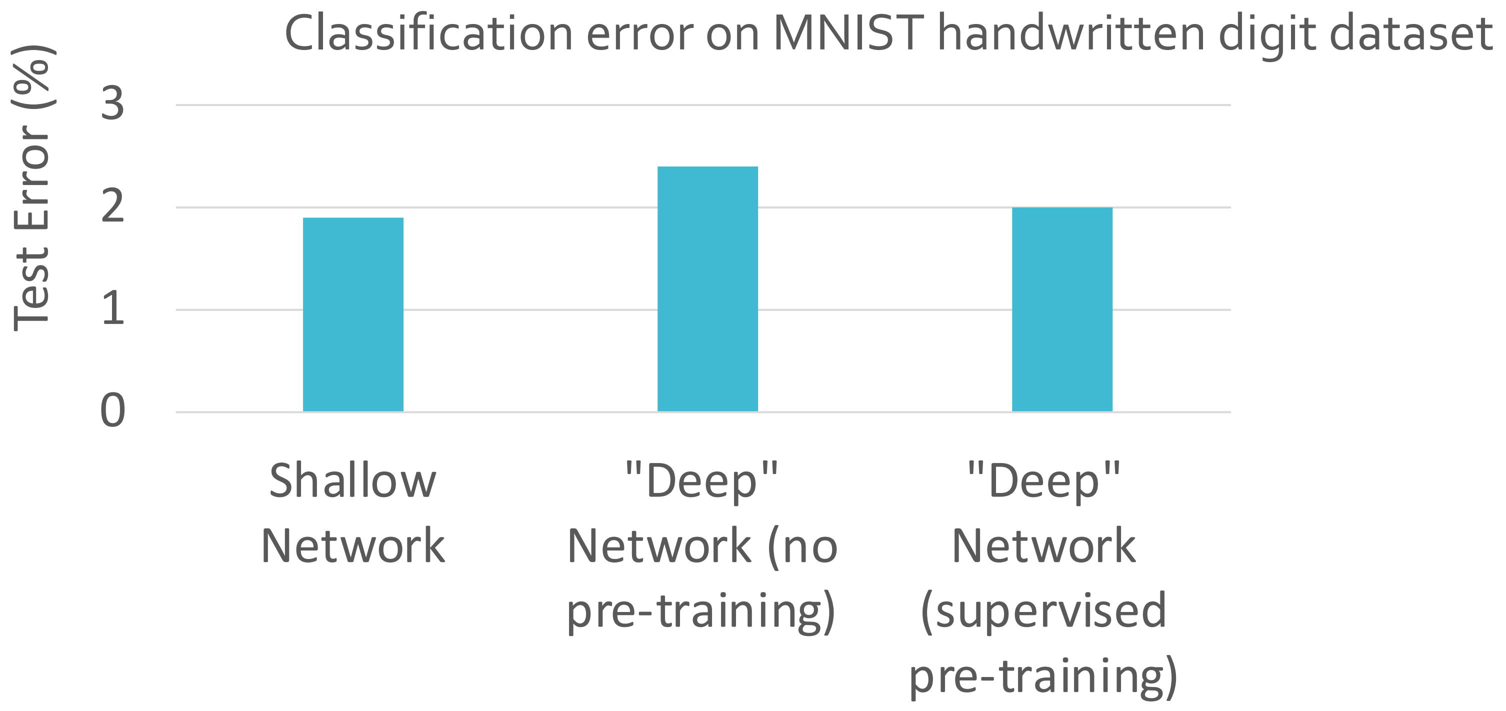
Supervised Pre-training (Bengio et al., 2006)

- Train each layer of the network iteratively using the training dataset
- Use the pre-trained weights as an initialization and *fine-tune* the entire network e.g., via SGD with the training dataset



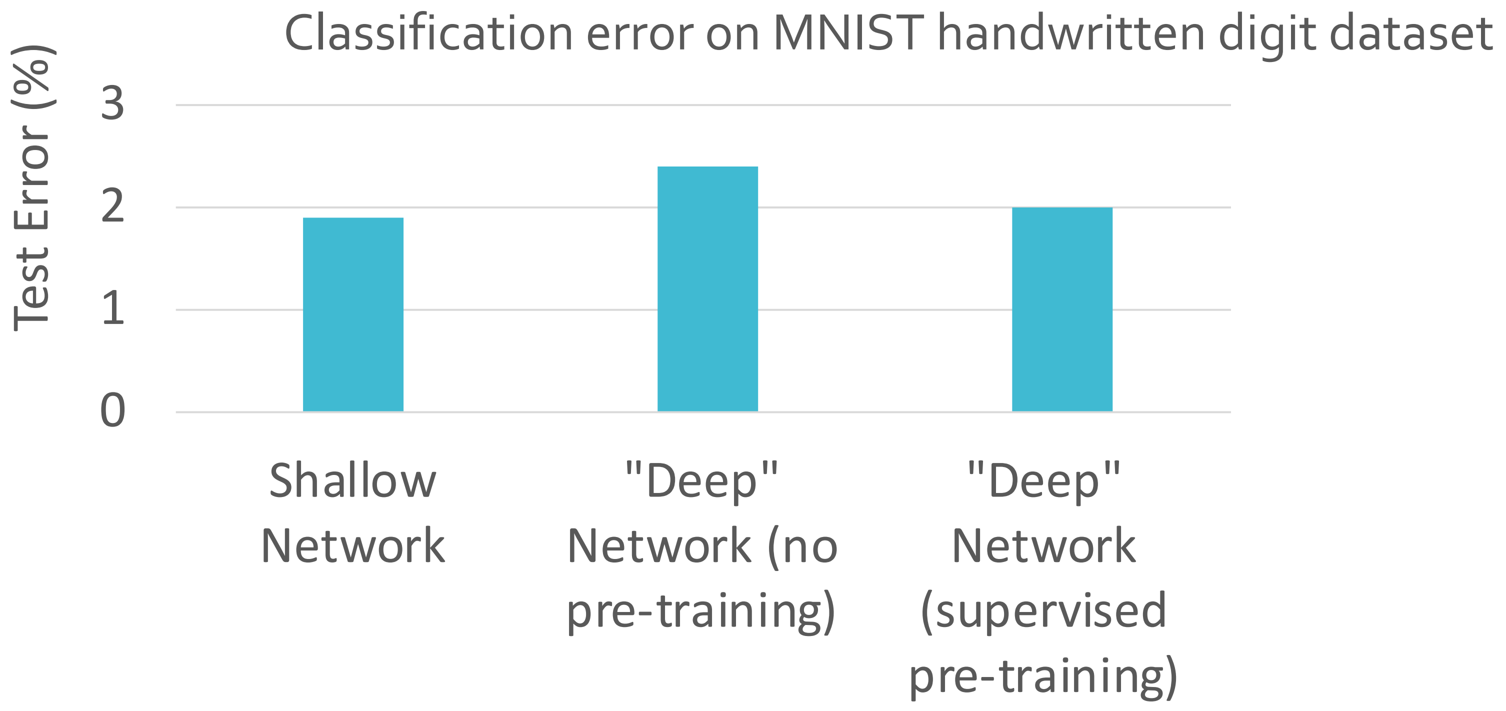
Supervised Pre-training (Bengio et al., 2006)

- Train each layer of the network iteratively using the training dataset *to predict the labels*
- Use the pre-trained weights as an initialization and *fine-tune* the entire network e.g., via SGD with the training dataset



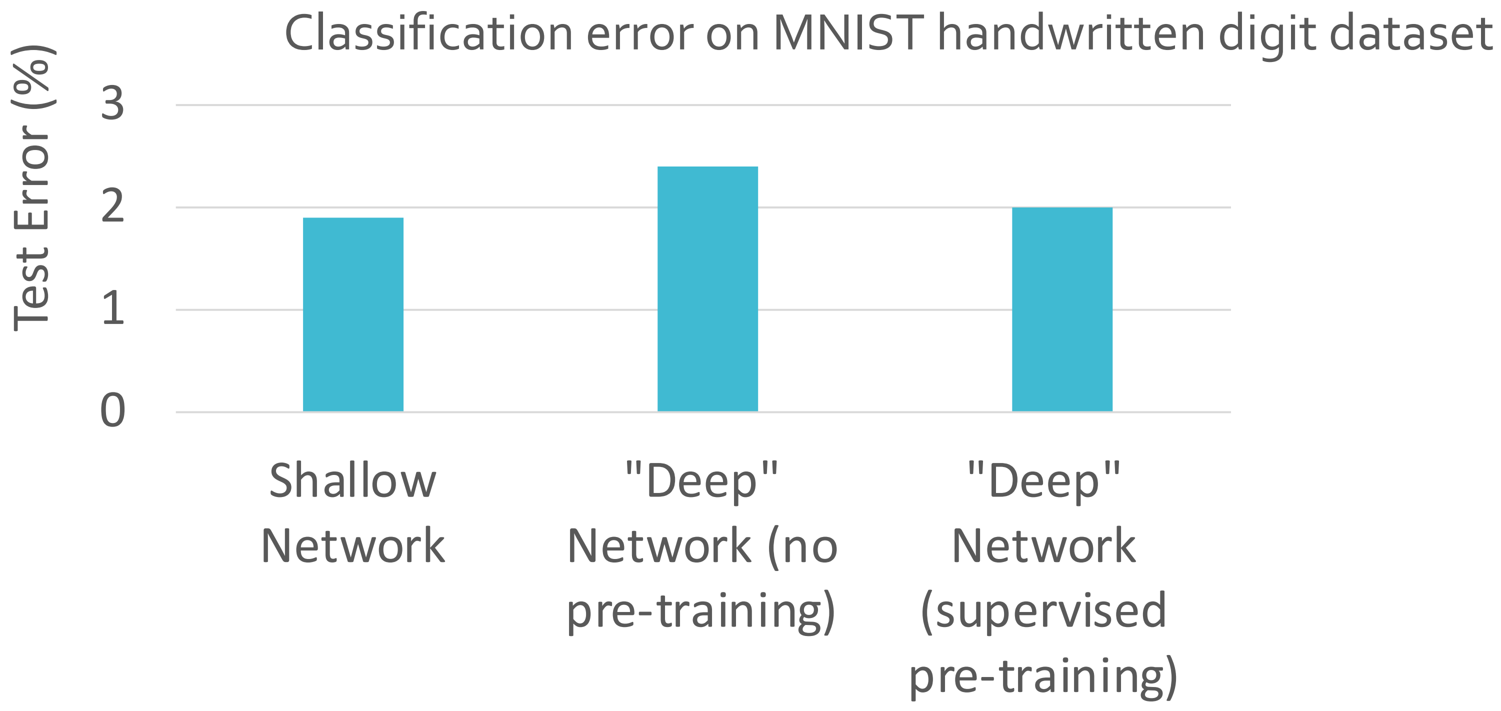
Is this the only thing we could do with the training data?

- Train each layer of the network iteratively using the training dataset *to predict the labels*
- Use the pre-trained weights as an initialization and *fine-tune* the entire network e.g., via SGD with the training dataset



Unsupervised Pre-training (Bengio et al., 2006)

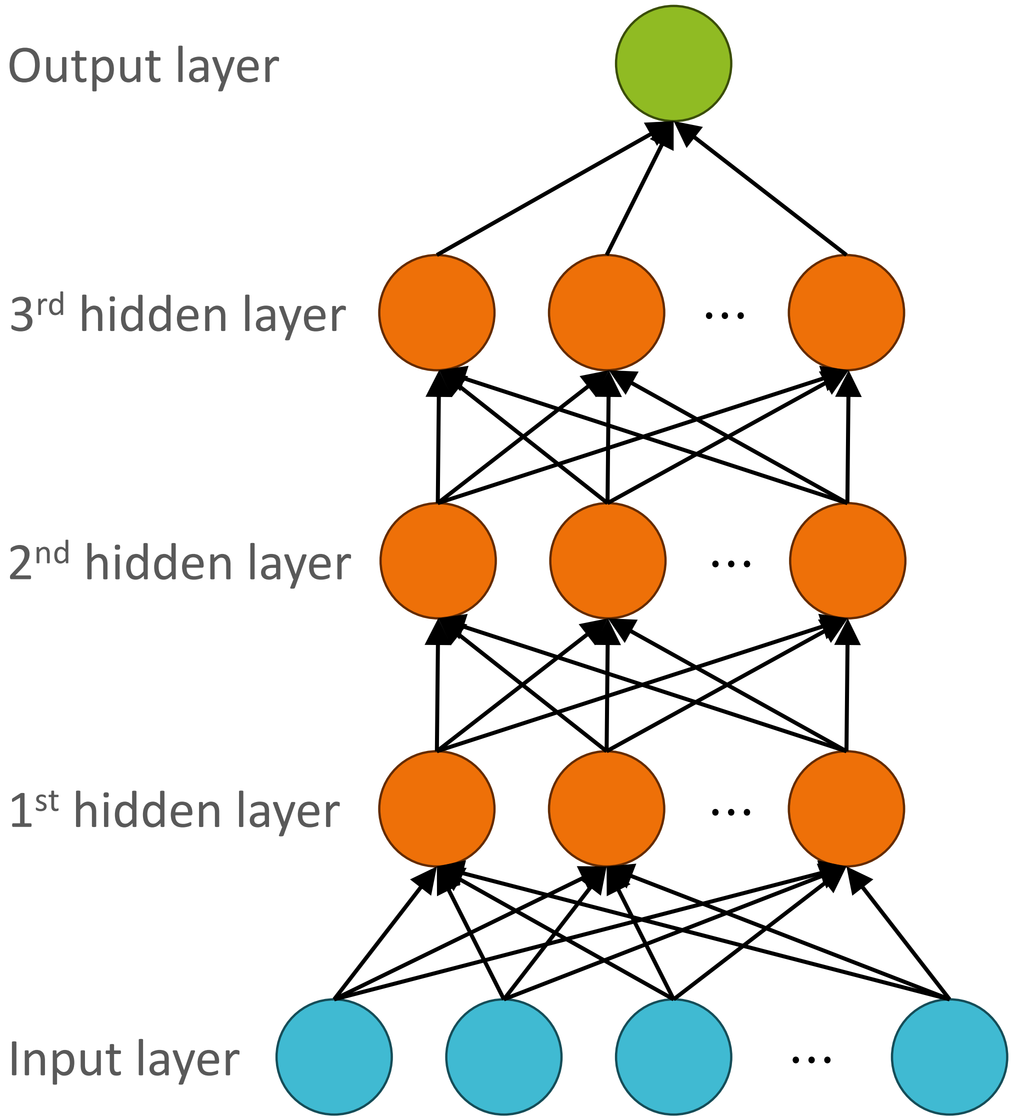
- Train each layer of the network iteratively using the training dataset *to learn useful representations*
- Idea: a good representation is one preserves a lot of information and could be used to recreate the inputs



Unsupervised Pre-training (Bengio et al., 2006)

- Train each layer of the network iteratively using the training dataset by minimizing the *reconstruction error*

$$\|\boldsymbol{x} - h(\boldsymbol{x})\|_2$$



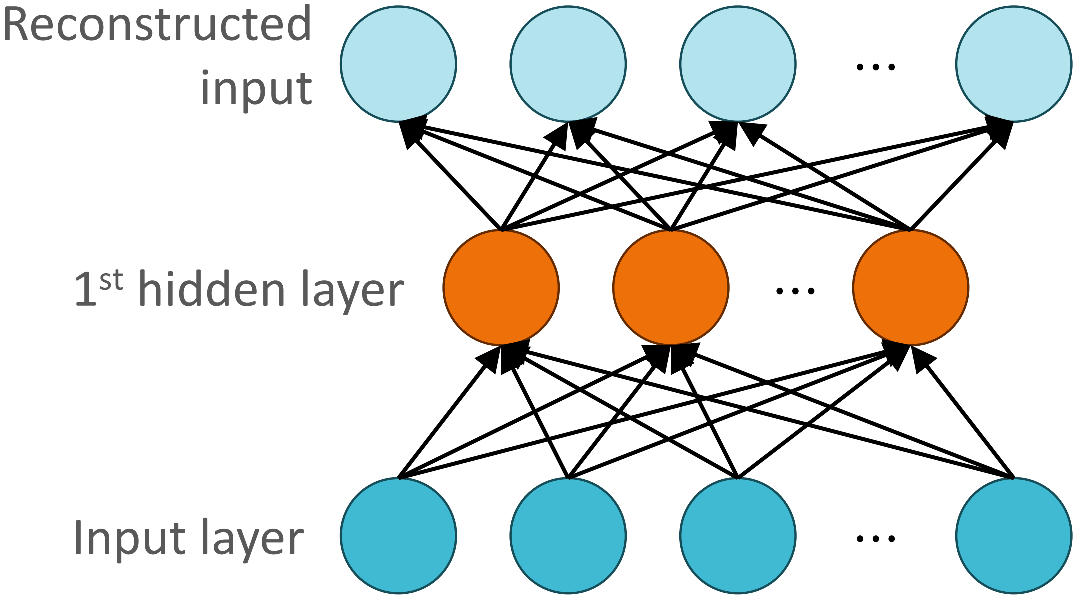
Source: https://www.cs.toronto.edu/~larocheh/publications/dbn_supervised_tr1282.pdf

Unsupervised Pre-training (Bengio et al., 2006)

- Train each layer of the network iteratively using the training dataset by minimizing the *reconstruction error*

$$\|\boldsymbol{x} - h(\boldsymbol{x})\|_2$$

- This architecture/ objective defines an *autoencoder*



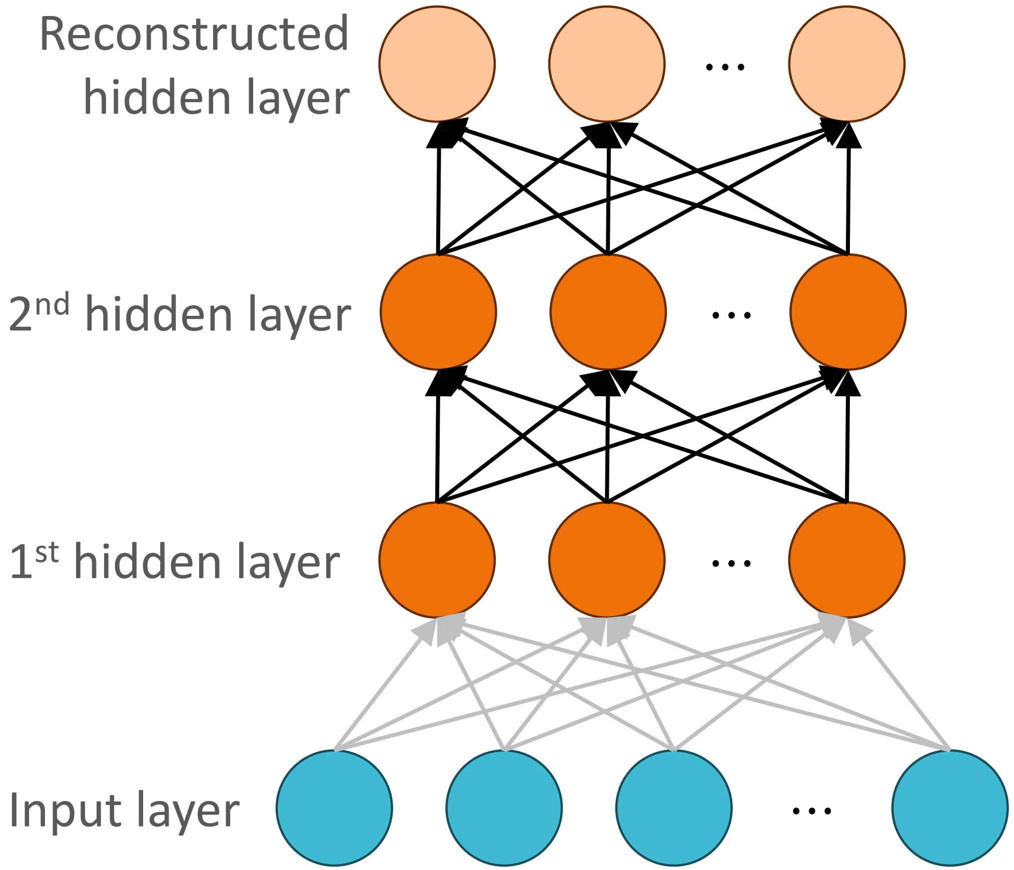
Source: https://www.cs.toronto.edu/~larocheh/publications/dbn_supervised_tr1282.pdf

Unsupervised Pre-training (Bengio et al., 2006)

- Train each layer of the network iteratively using the training dataset by minimizing the *reconstruction error*

$$\|\boldsymbol{x} - h(\boldsymbol{x})\|_2$$

- This architecture/ objective defines an *autoencoder*



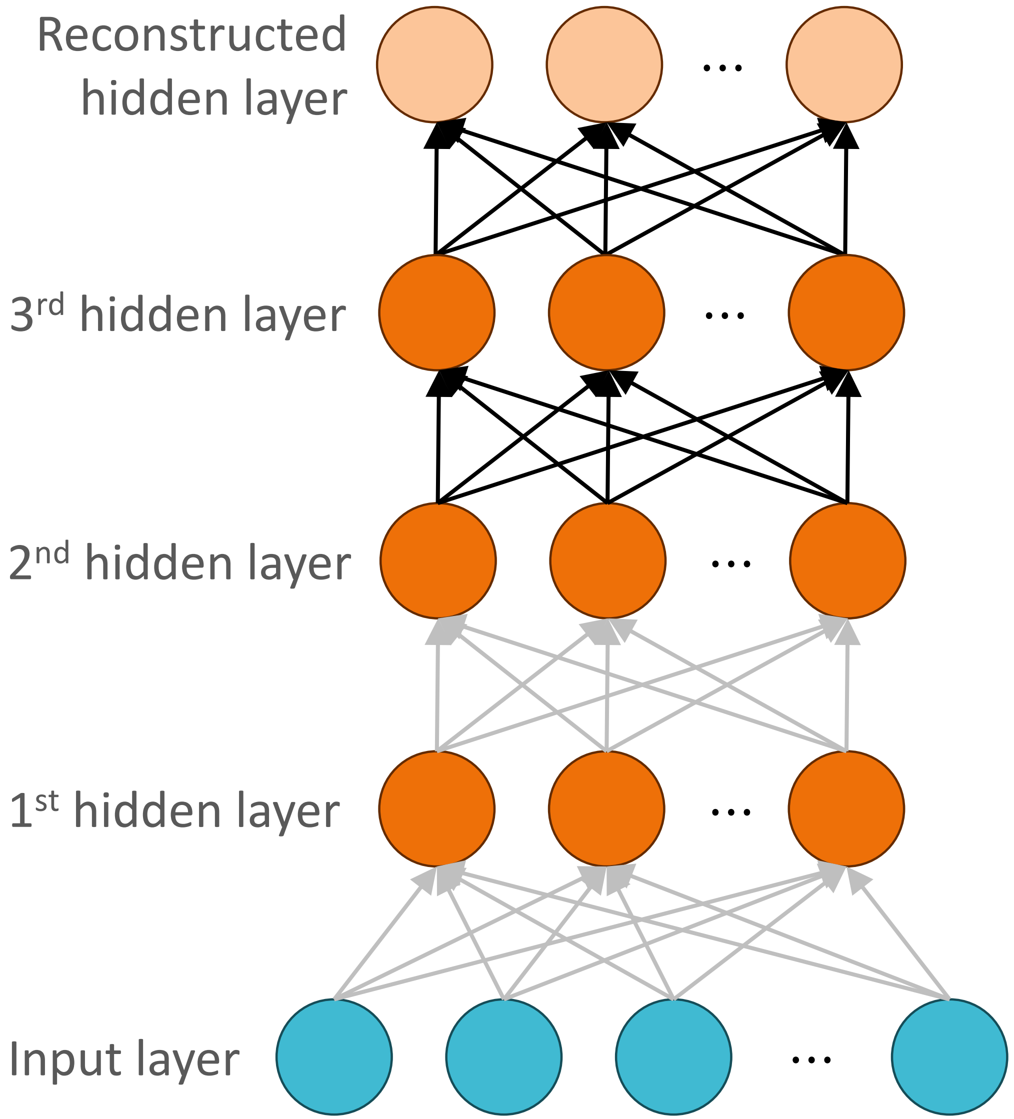
Source: https://www.cs.toronto.edu/~larocheh/publications/dbn_supervised_tr1282.pdf

Unsupervised Pre-training (Bengio et al., 2006)

- Train each layer of the network iteratively using the training dataset by minimizing the *reconstruction error*

$$\|\boldsymbol{x} - h(\boldsymbol{x})\|_2$$

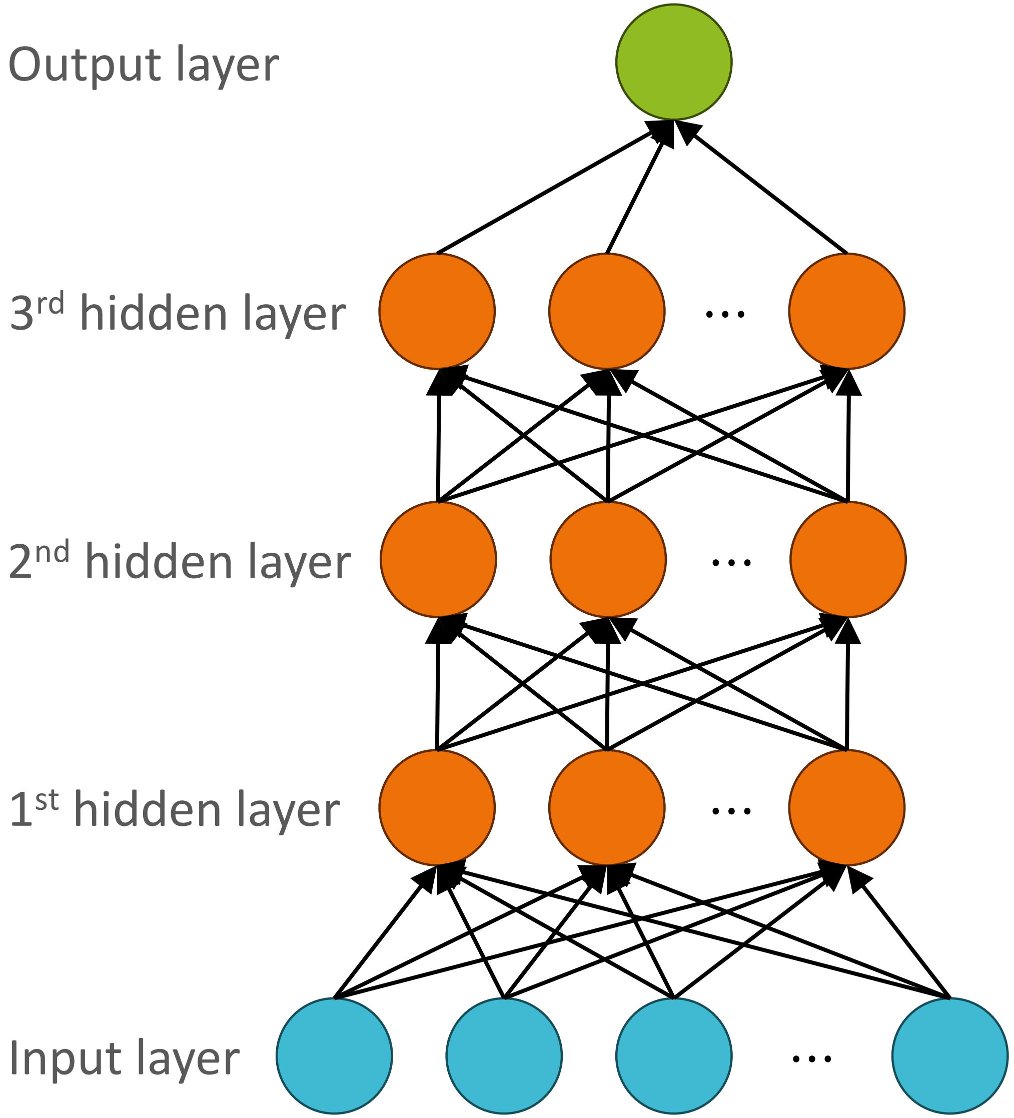
- This architecture/ objective defines an *autoencoder*



Source: https://www.cs.toronto.edu/~larocheh/publications/dbn_supervised_tr1282.pdf

Fine-tuning (Bengio et al., 2006)

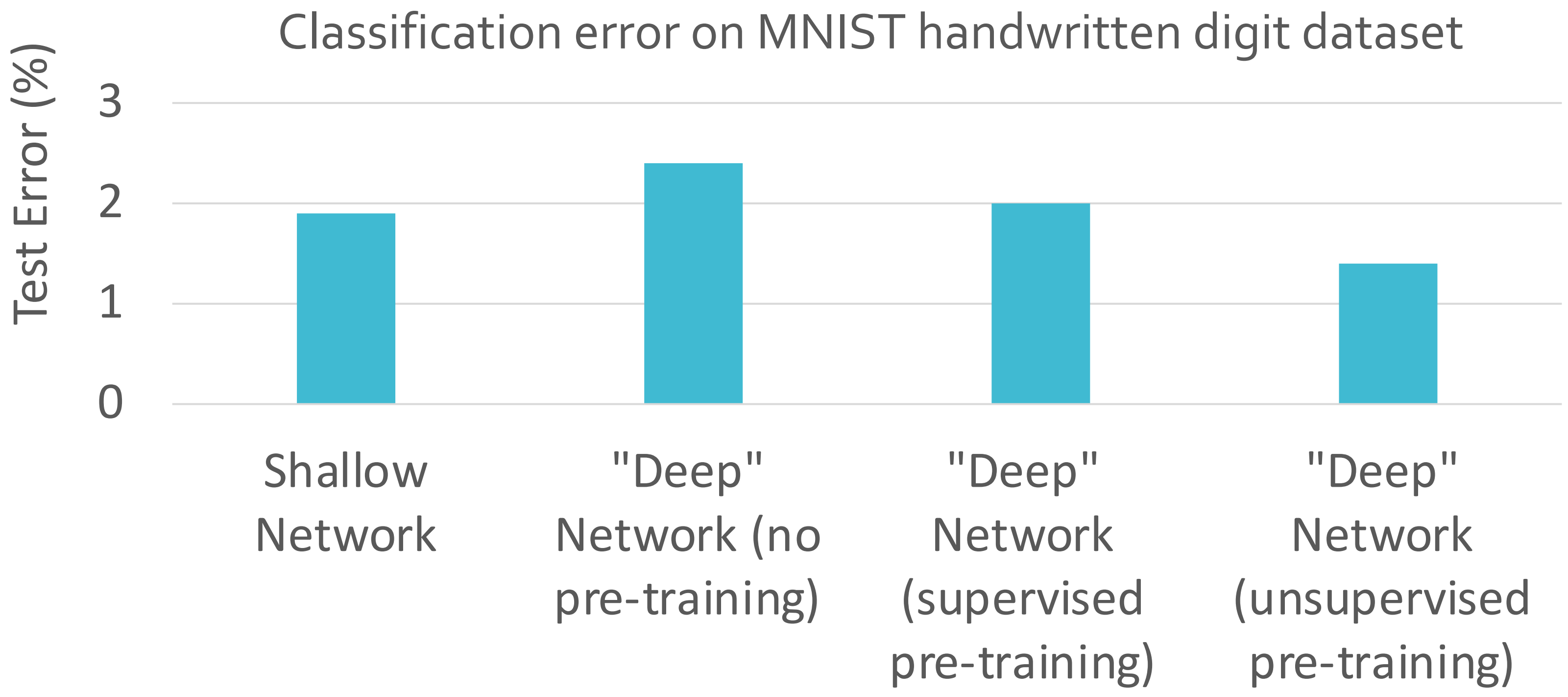
- Train each layer of the network iteratively using the training dataset by minimizing the *reconstruction error* $\|\boldsymbol{x} - h(\boldsymbol{x})\|_2$
- When fine-tuning, we're effectively swapping out the last layer and fitting all the weights to the training dataset



Source: https://www.cs.toronto.edu/~larocheh/publications/dbn_supervised_tr1282.pdf

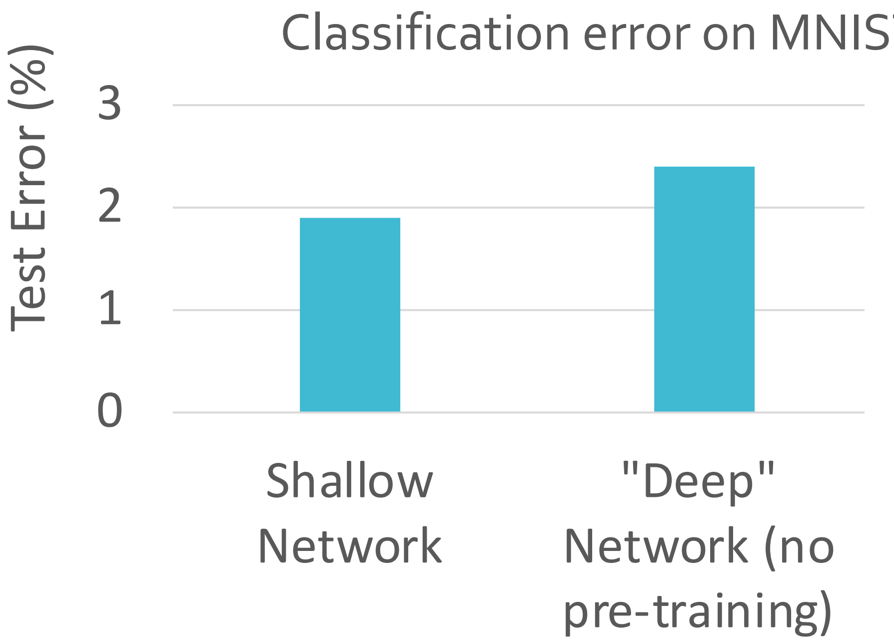
Unsupervised Pre-training (Bengio et al., 2006)

- Train each layer of the network iteratively using the training dataset by minimizing the *reconstruction error*
- Idea: a good representation is one preserves a lot of information and could be used to recreate the inputs



Another dose of Reality

- You have some niche task that you want to apply machine learning to e.g., predicting how Henry will get to work
- You have a tiny labelled dataset to train with
- You fit a massive deep learning model to the dataset
- Surprise, surprise: it overfits and your test error is super high



- Problem: what if you don't even have enough data to train a single layer/fine-tune the pre-trained network?

Source: https://www.cs.toronto.edu/~larocheh/publications/dbn_supervised_tr1282.pdf

Another dose of Reality

- You have some niche task that you want to apply machine learning to e.g., predicting how Henry will get to work
- You have a tiny labelled dataset to train with
- You fit a massive deep learning model to the dataset
- Surprise, surprise: it overfits and your test error is super high
- Key observation: you can pre-train on basically any labelled or unlabelled dataset!
  - Ideally, you want to use a *large* dataset *related* to your goal task

Another dose of Reality

- You have some niche task that you want to apply machine learning to e.g., predicting how Henry will get to work
- You have a tiny labelled dataset to train with
- You fit a massive deep learning model to the dataset
- Surprise, surprise: it overfits and your test error is super high
- Key observation: you can pre-train on basically any labelled or unlabelled dataset!
  - GPT-3 pre-training data:

Dataset	Quantity (tokens)	Weight in training mix
Common Crawl (filtered)	410 billion	60%
WebText2	19 billion	22%
Books1	12 billion	8%
Books2	55 billion	8%
Wikipedia	3 billion	3%

Source: https://arxiv.org/pdf/2005.14165.pdf

Another dose of Reality

- You have some niche task that you want to apply machine learning to e.g., predicting how Henry will get to work
- You have a tiny labelled dataset to train with
- You fit a massive deep learning model to the dataset
- Surprise, surprise: it overfits and your test error is super high
- Key observation: you can pre-train on basically any labelled or unlabelled dataset!
- Okay that's great for pre-training and all, but what if you don't even have enough data to fine-tune your model?

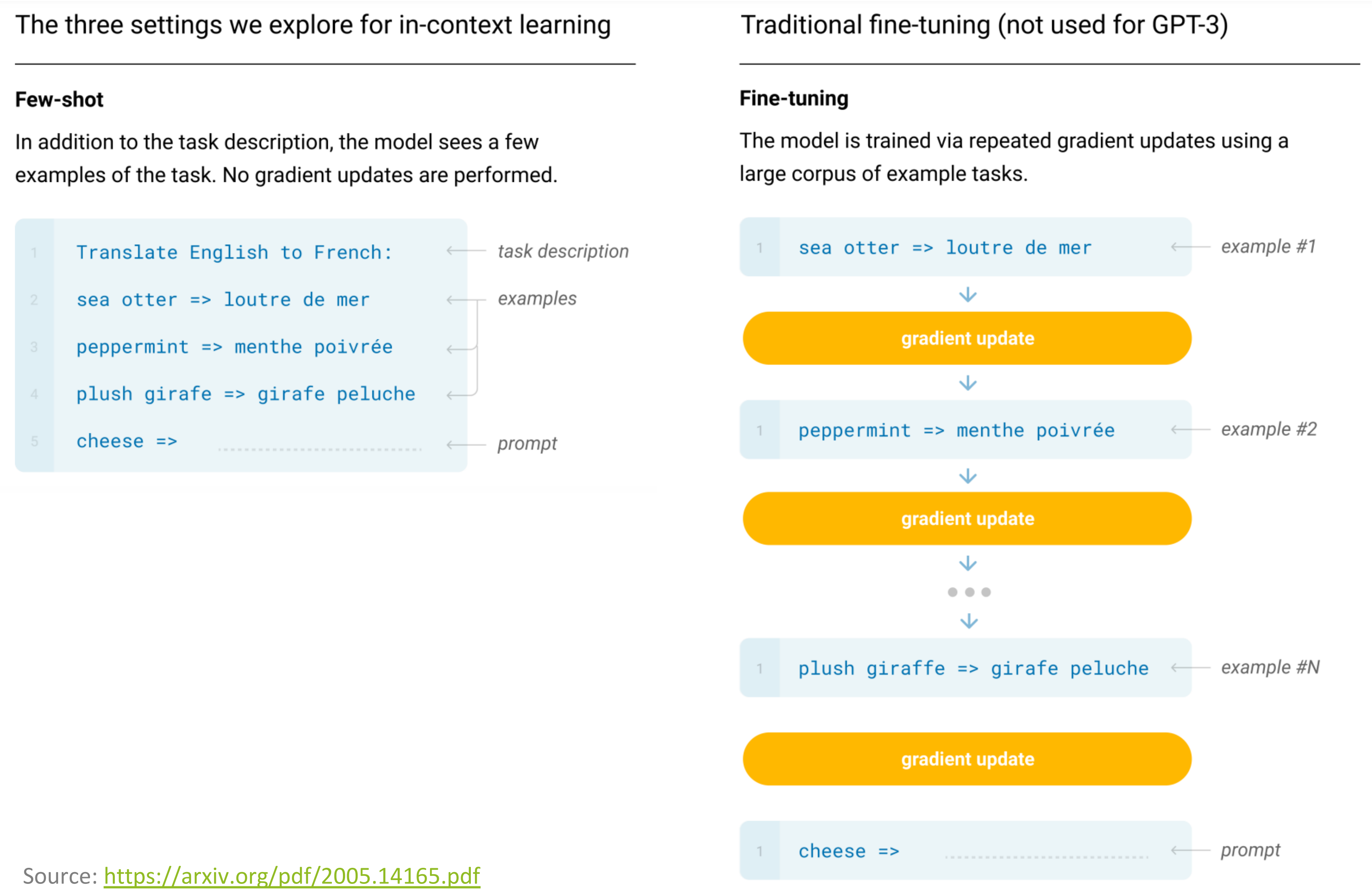
In-context Learning

- Problem: given their size, effectively fine-tuning LLMs can require lots of labelled data points.
- Idea: leverage the LLM's context window by passing a few examples to the model as input, *without performing any updates to the parameters*
- Intuition: during training, the LLM is exposed to a *massive* number of examples/tasks and the input conditions the model to "locate" the relevant concepts

Source: https://arxiv.org/pdf/2111.02080.pdf

Few-shot, One-shot & Zero-shot (in-context) Learning

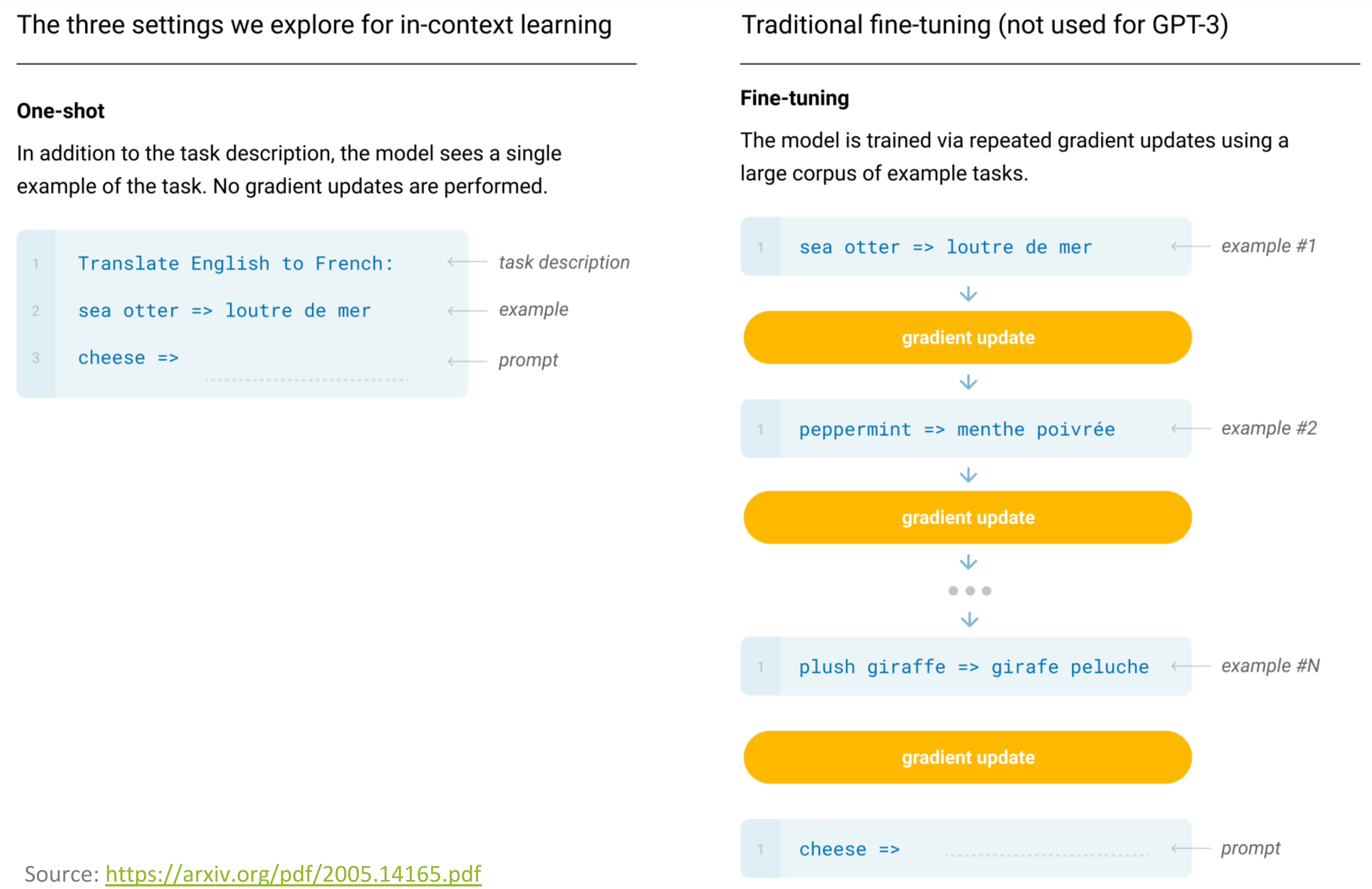
- Idea: leverage the LLM's context window by passing a few examples to the model as input, *without performing any updates to the parameters*



Source: https://arxiv.org/pdf/2005.14165.pdf

Few-shot, One-shot & Zero-shot (in-context) Learning

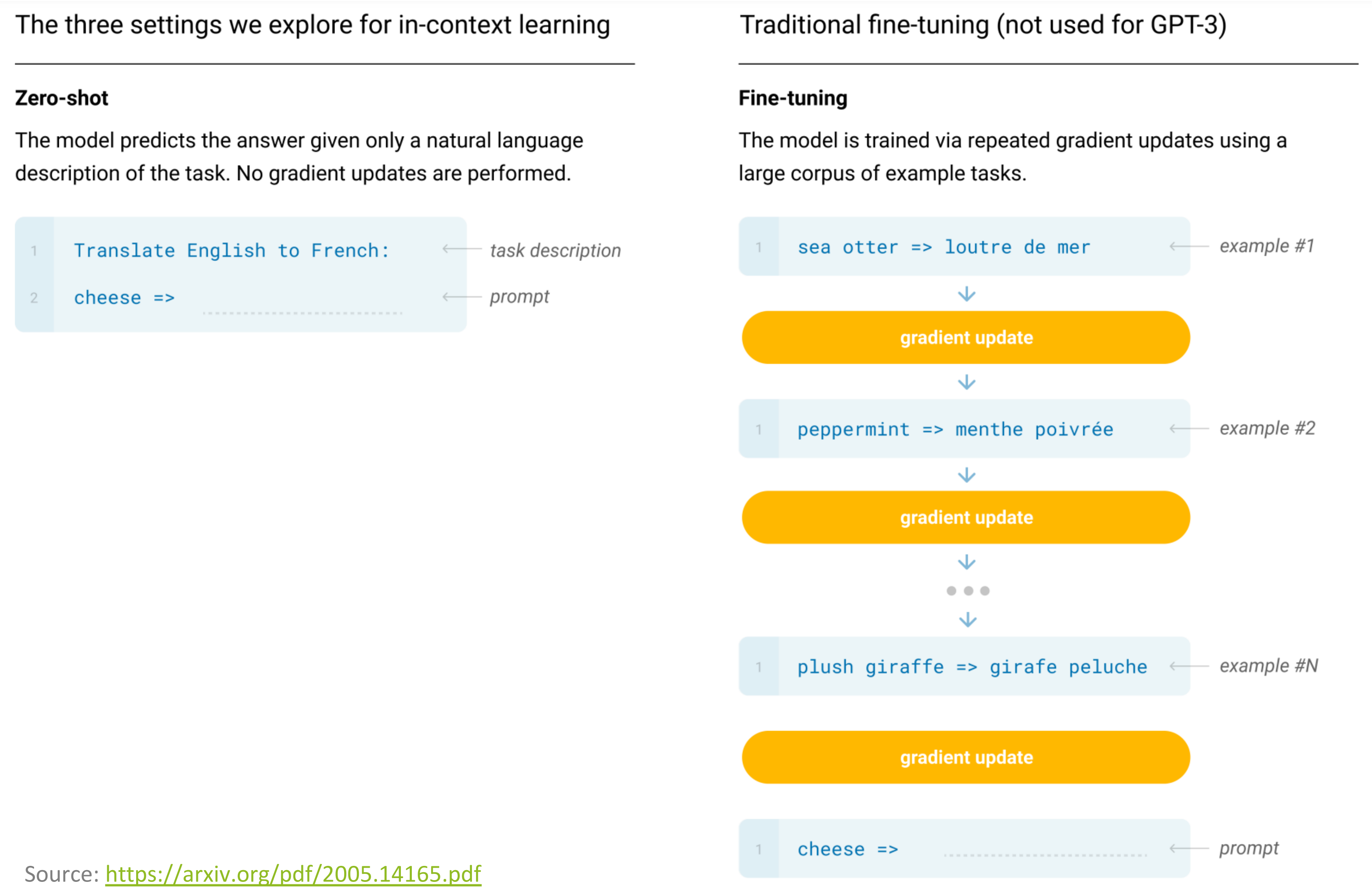
- Idea: leverage the LLM's context window by passing ~~a~~ ~~few~~ one example~~s~~ to the model as input, *without performing any updates to the parameters*



Source: https://arxiv.org/pdf/2005.14165.pdf

Few-shot, One-shot & Zero-shot (in-context) Learning

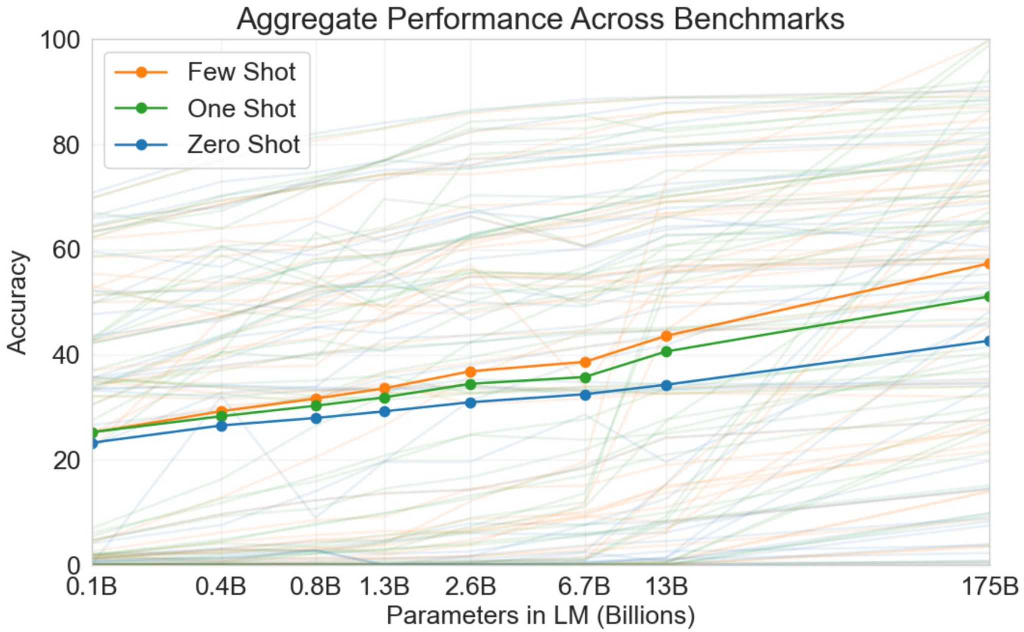
- Idea: leverage the LLM's context window by passing ~~a few one~~ zero(!) examples to the model as input, *without performing any updates to the parameters*



Source: https://arxiv.org/pdf/2005.14165.pdf

Few-shot, One-shot & Zero-shot (in-context) Learning

- Idea: leverage the LLM's context window by passing ~~a few one~~ zero(!) examples to the model as input, *without performing any updates to the parameters*



- Key Takeaway: LLMs can perform well on novel tasks without having to fine-tune the model, sometimes even with just one or zero labelled training data points!

Source: https://arxiv.org/pdf/2005.14165.pdf

Key Takeaways

- Instead of random initializations, modern deep learning typically initializes weights via pretraining, then fine-tunes them to the specific task
  - Supervised vs. unsupervised fine-tuning
  - Pretraining need not occur on the task of interest
- Some tasks can be performed by a pretrained LLM without any fine-tuning via in-context learning