10-701: Introduction to Machine Learning Lecture 18 – Pretraining, Fine-tuning & In-Context Learning

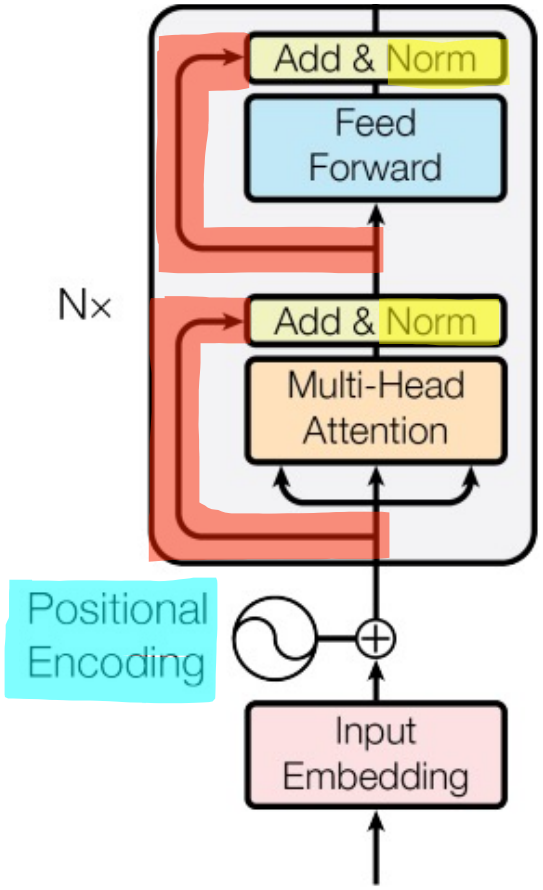
Henry Chai

3/25/24

Front Matter

- Announcements:
  - HW5 released 3/22, due 4/1 at 11:59 PM
  - Project mentors will be assigned later this week
    - Recitation on 3/29 is dedicated time to meet with your project mentors
    - **Your group must meet with your assigned project mentor and receive approval on your proposal to move forward to the next deliverable**
  - **Daniel is on leave and will be for an indeterminate amount of time, please direct all course requests/questions to Henry**

Okay, one massive detour later, 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: training dataset $\mathcal{D} = \left\{\left(\boldsymbol{x}^{(i)}, y^{(i)}\right)\right\}_{i=1}^{N}$, step size γ, and batch size B

1. Randomly initialize the parameters $\boldsymbol{\theta}^{(0)}$ 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*, $\nabla J^{(B)}\left(\boldsymbol{\theta}^{(t)}\right)$
   c. Update $\boldsymbol{\theta}$: $\boldsymbol{\theta}^{(t+1)} \leftarrow \boldsymbol{\theta}^{(t)} - \gamma \nabla J^{(B)}\left(\boldsymbol{\theta}^{(t)}\right)$
   d. Increment t: $t \leftarrow t + 1$

- Output: $\boldsymbol{\theta}^{(t)}$

Mini-batch Stochastic Gradient Descent is a lie!

- Input: training dataset $\mathcal{D} = \{(\boldsymbol{x}^{(i)}, y^{(i)})\}_{i=1}^{N}$, step size γ, and batch size B

1. Randomly initialize the parameters $\boldsymbol{\theta}^{(0)}$ and set $t = 0$
2. While TERMINATION CRITERION is not satisfied
   a. Randomly sample B data points from $\mathcal{D}$, $\{(\boldsymbol{x}^{(b)}, y^{(b)})\}_{b=1}^{B}$
   b. Compute the gradient of the loss w.r.t. the sampled *batch*, $\nabla J^{(B)}(\boldsymbol{\theta}^{(t)})$
   c. Update $\boldsymbol{\theta}$: $\boldsymbol{\theta}^{(t+1)} \leftarrow \boldsymbol{\theta}^{(t)} - \gamma \nabla J^{(B)}(\boldsymbol{\theta}^{(t)})$
   d. Increment t: $t \leftarrow t + 1$

- Output: $\boldsymbol{\theta}^{(t)}$

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

- Input: training dataset $\mathcal{D} = \{(\boldsymbol{x}^{(i)}, y^{(i)})\}_{i=1}^{N}$, step size γ, and batch size B

1. Randomly initialize the parameters $\boldsymbol{\theta}^{(0)}$ and set $t = 0$
2. While TERMINATION CRITERION is not satisfied
   a. Randomly sample B data points from $\mathcal{D}$, $\{(\boldsymbol{x}^{(b)}, y^{(b)})\}_{b=1}^{B}$
   b. Compute the gradient of the loss w.r.t. the sampled *batch*, $\nabla J^{(B)}(\boldsymbol{\theta}^{(t)})$
   c. Update $\boldsymbol{\theta}$: $\boldsymbol{\theta}^{(t+1)} \leftarrow \boldsymbol{\theta}^{(t)} - \gamma \nabla J^{(B)}(\boldsymbol{\theta}^{(t)})$
   d. Increment t: $t \leftarrow t + 1$

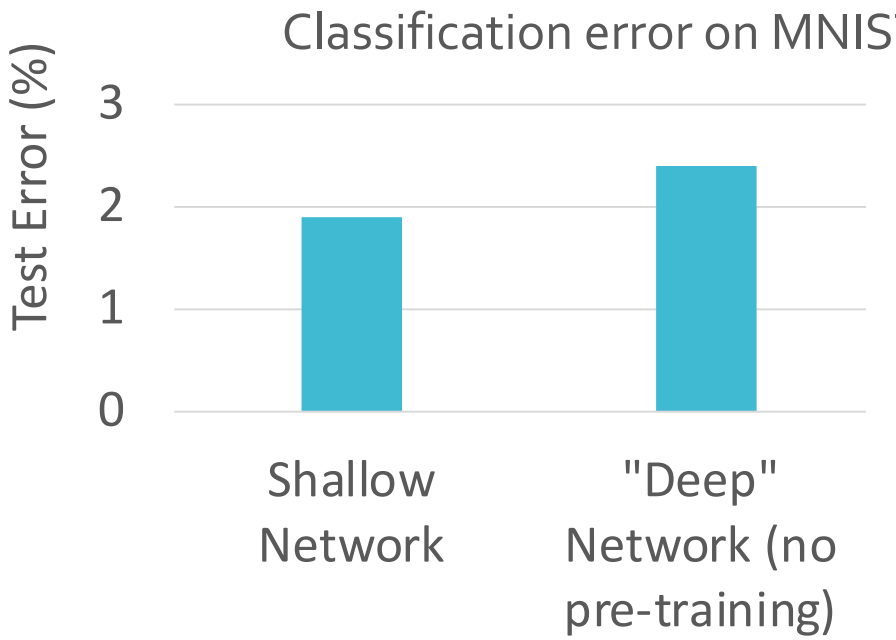
- Output: $\boldsymbol{\theta}^{(t)}$

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 the author of children's books
- 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

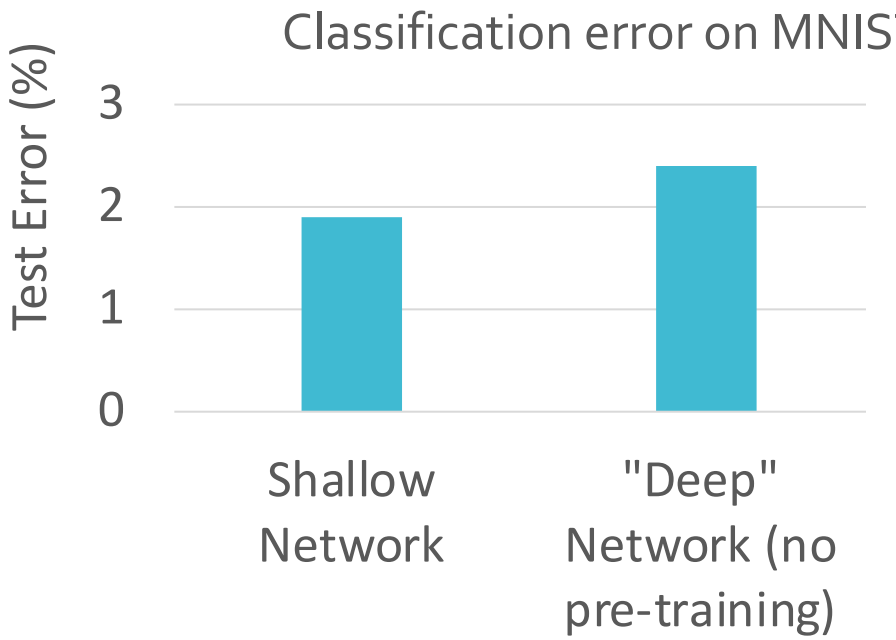


- "gradient-based optimization starting from random initialization appears to often get stuck in poor solutions for such deep networks."

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 the author of children's books
- 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

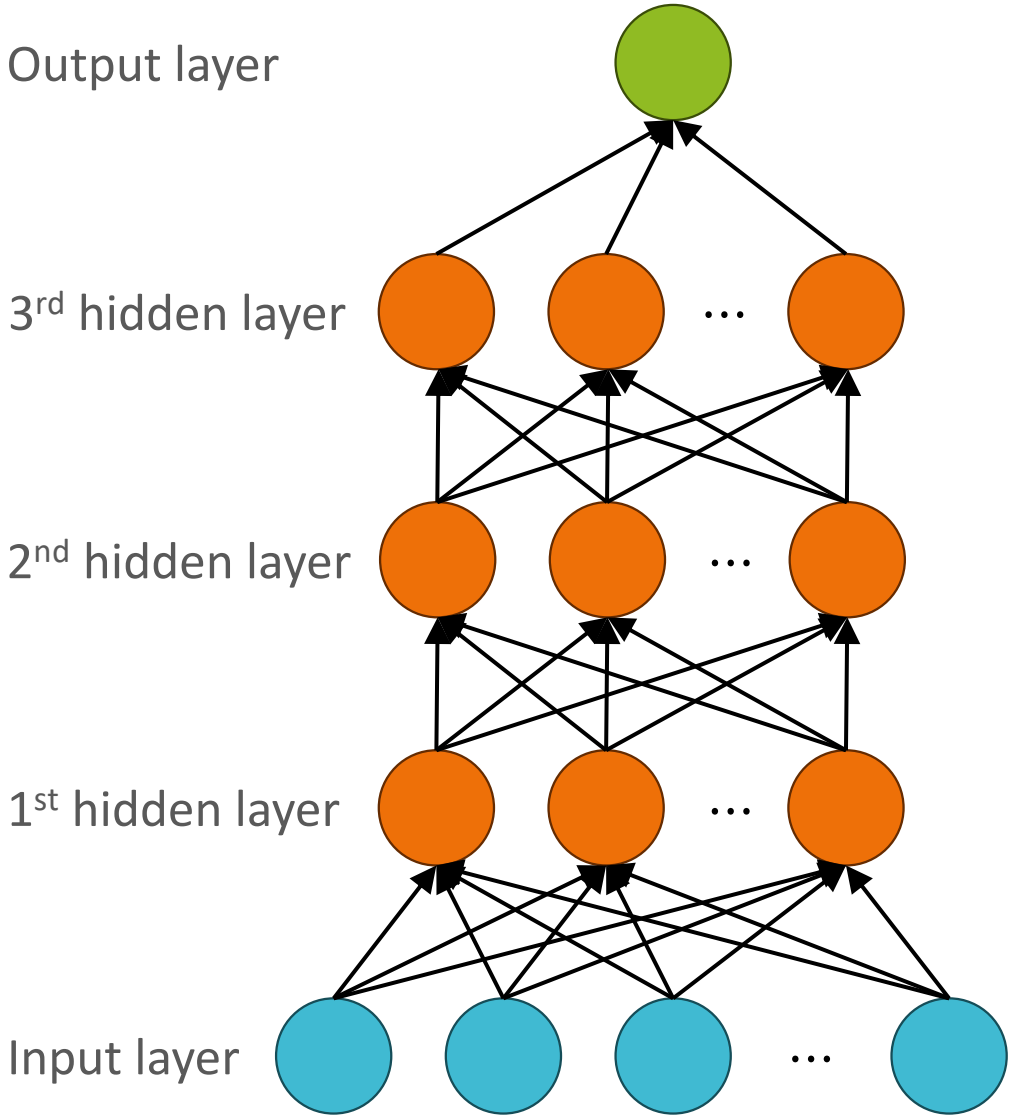


- Idea: if shallow networks are easier to train, let's just decompose our deep network into a series of shallow networks!

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

Pre-training (Bengio et al., 2006)

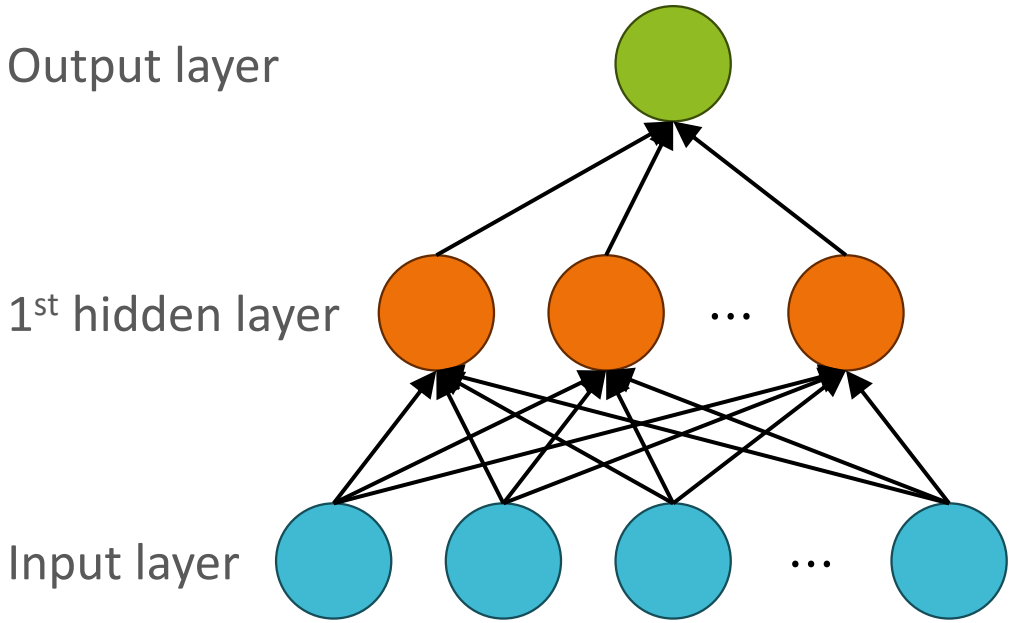
- Train each layer of the network iteratively using the training dataset
- Start at the input layer and move towards the output layer
- Once a layer has been trained, fix its weights and use those to train subsequent layers



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

Pre-training (Bengio et al., 2006)

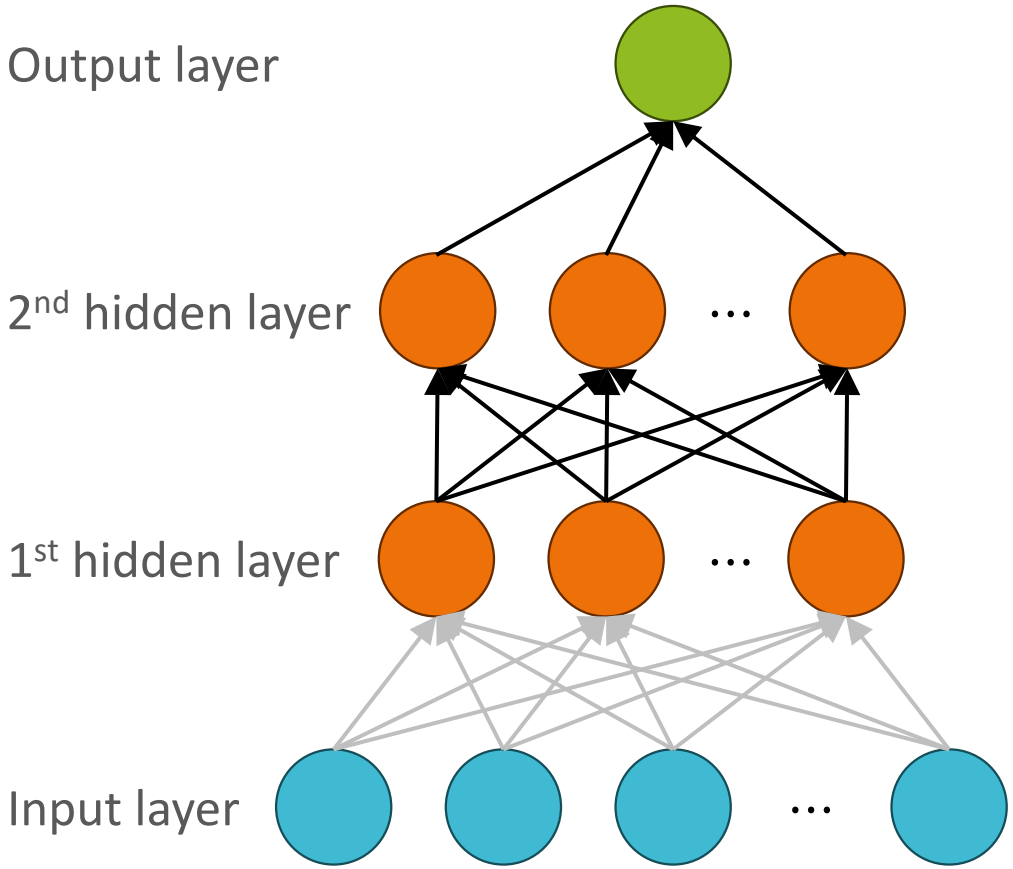
- Train each layer of the network iteratively using the training dataset
- Start at the input layer and move towards the output layer
- Once a layer has been trained, fix its weights and use those to train subsequent layers



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

Pre-training (Bengio et al., 2006)

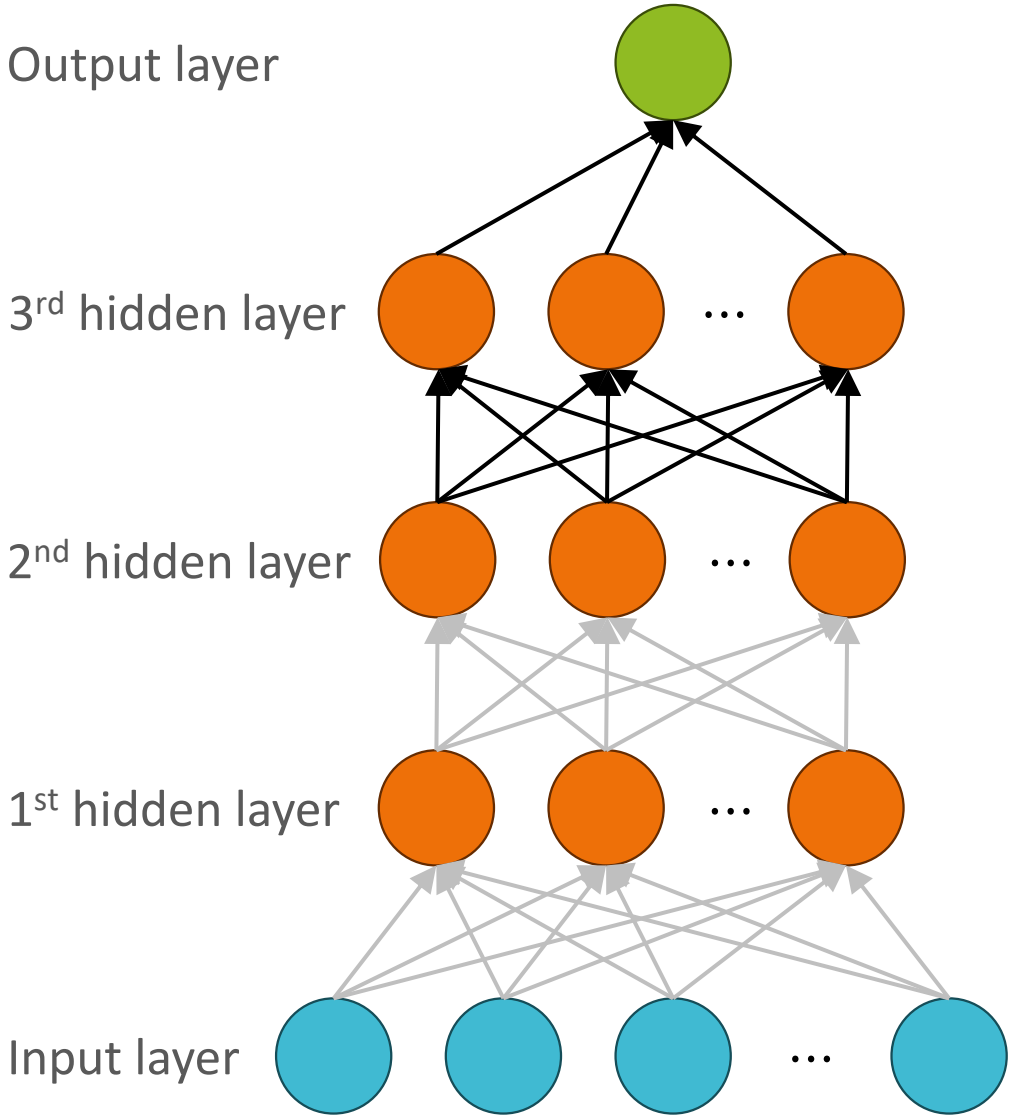
- Train each layer of the network iteratively using the training dataset
- Start at the input layer and move towards the output layer
- Once a layer has been trained, fix its weights and use those to train subsequent layers



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

Pre-training (Bengio et al., 2006)

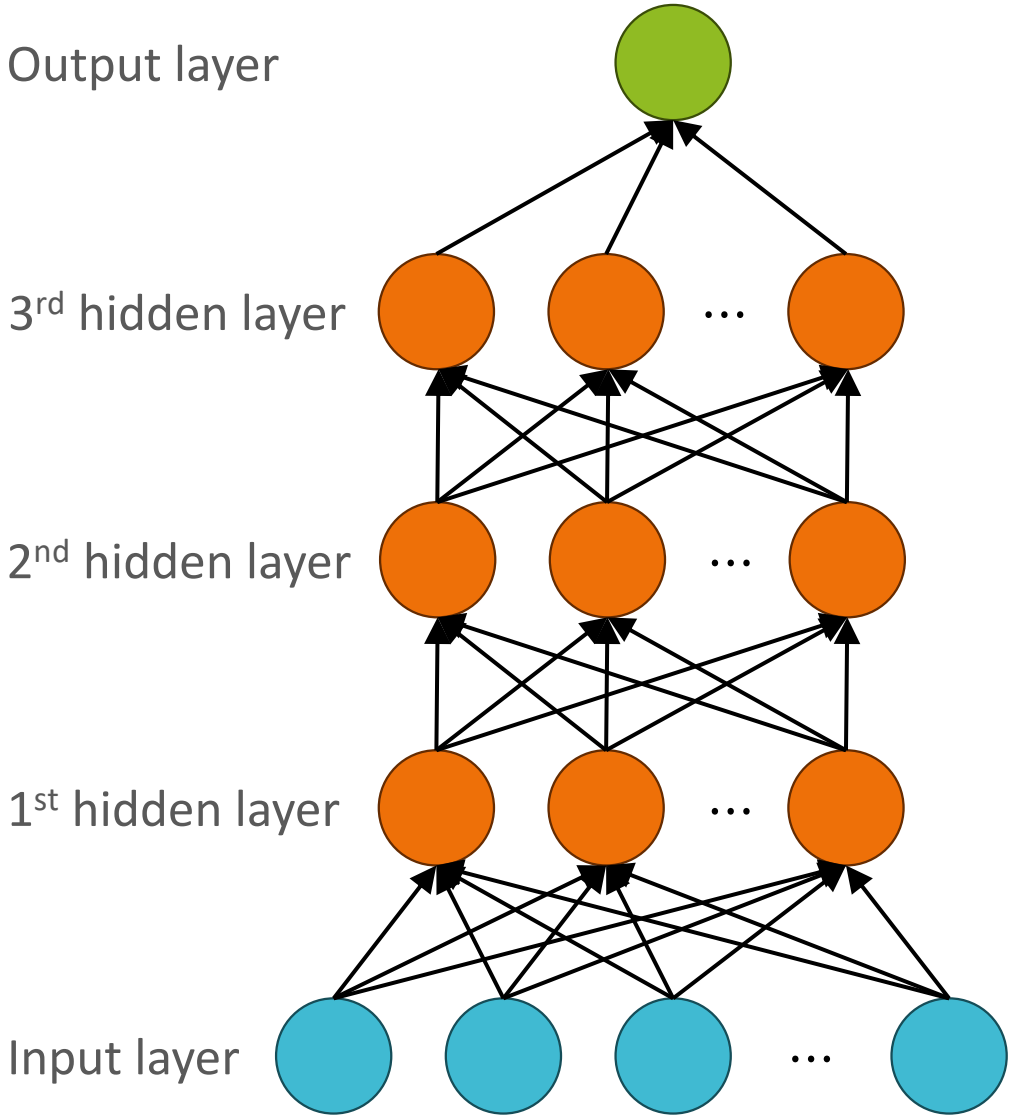
- Train each layer of the network iteratively using the training dataset
- Start at the input layer and move towards the output layer
- Once a layer has been trained, fix its weights and use those to train subsequent layers



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

Fine-tuning (Bengio et al., 2006)

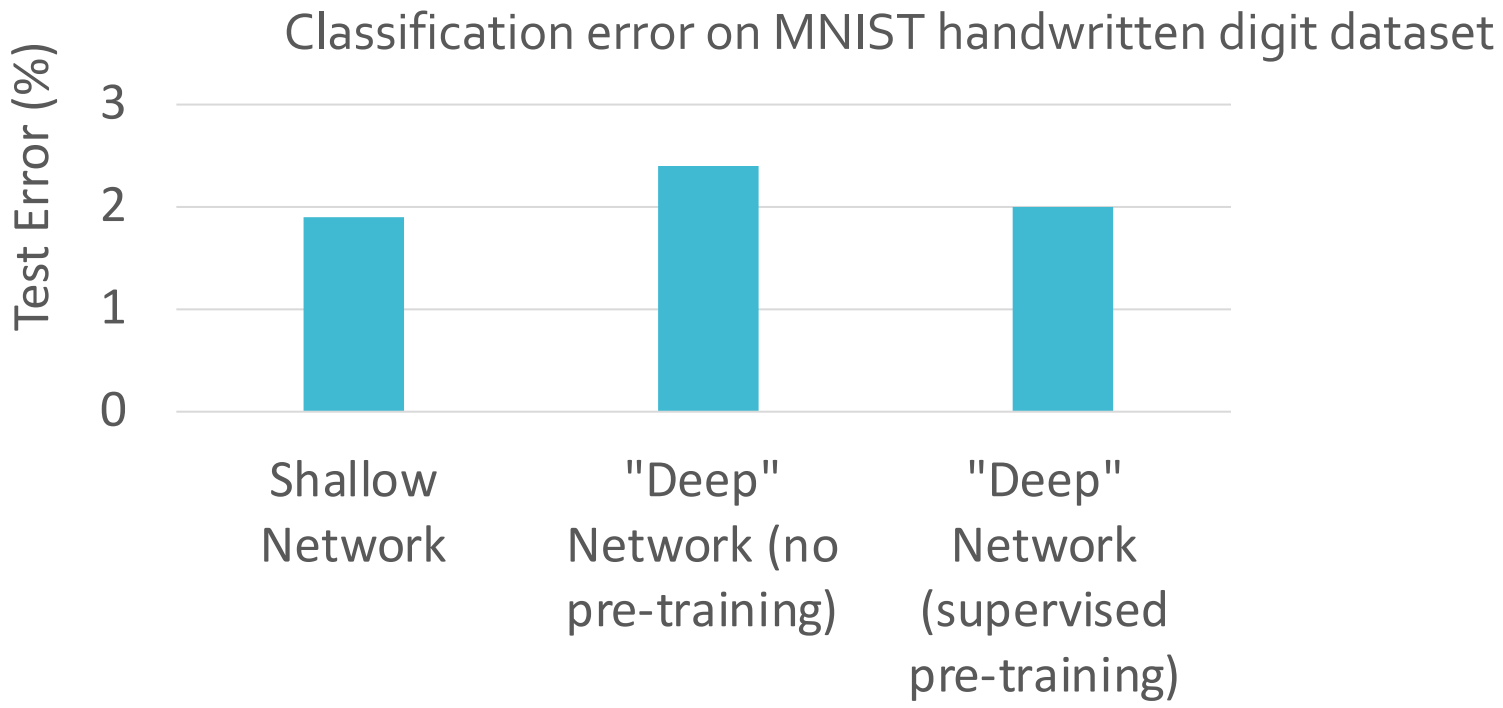
- Train each layer of the network iteratively using the training dataset
- Use the pre-trained weights as an initialization and *fine-tune* the entire network e.g., via SGD with the training dataset



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

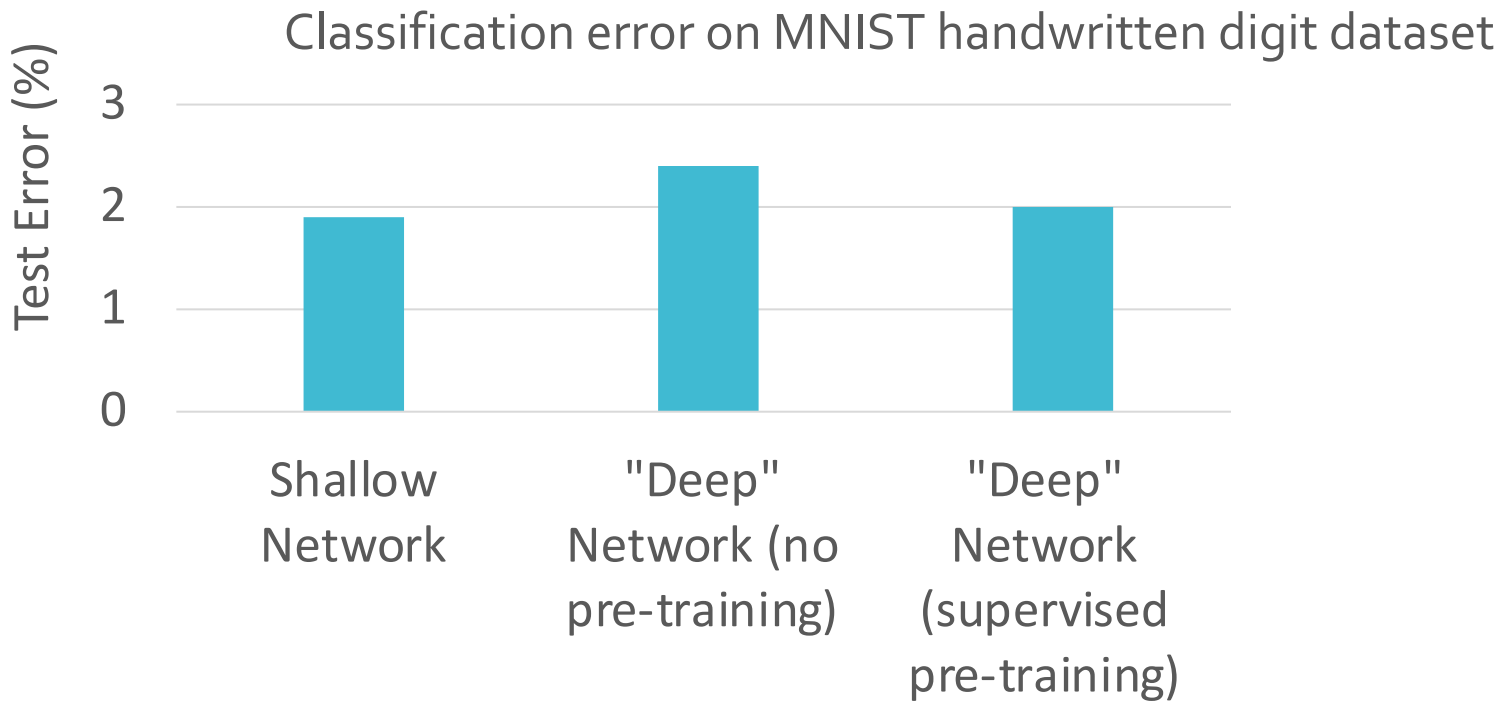
Supervised Pre-training (Bengio et al., 2006)

- 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



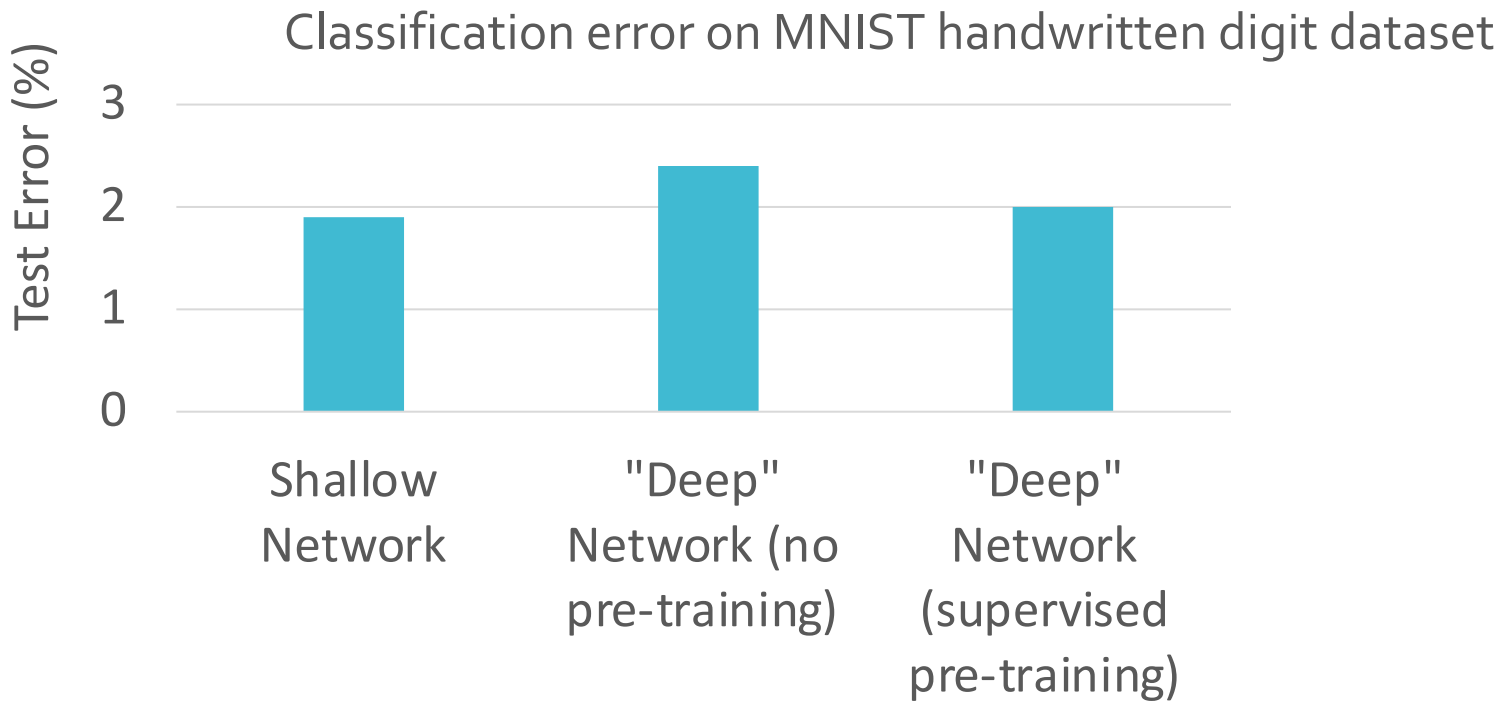
Supervised Pre-training (Bengio et al., 2006)

- 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



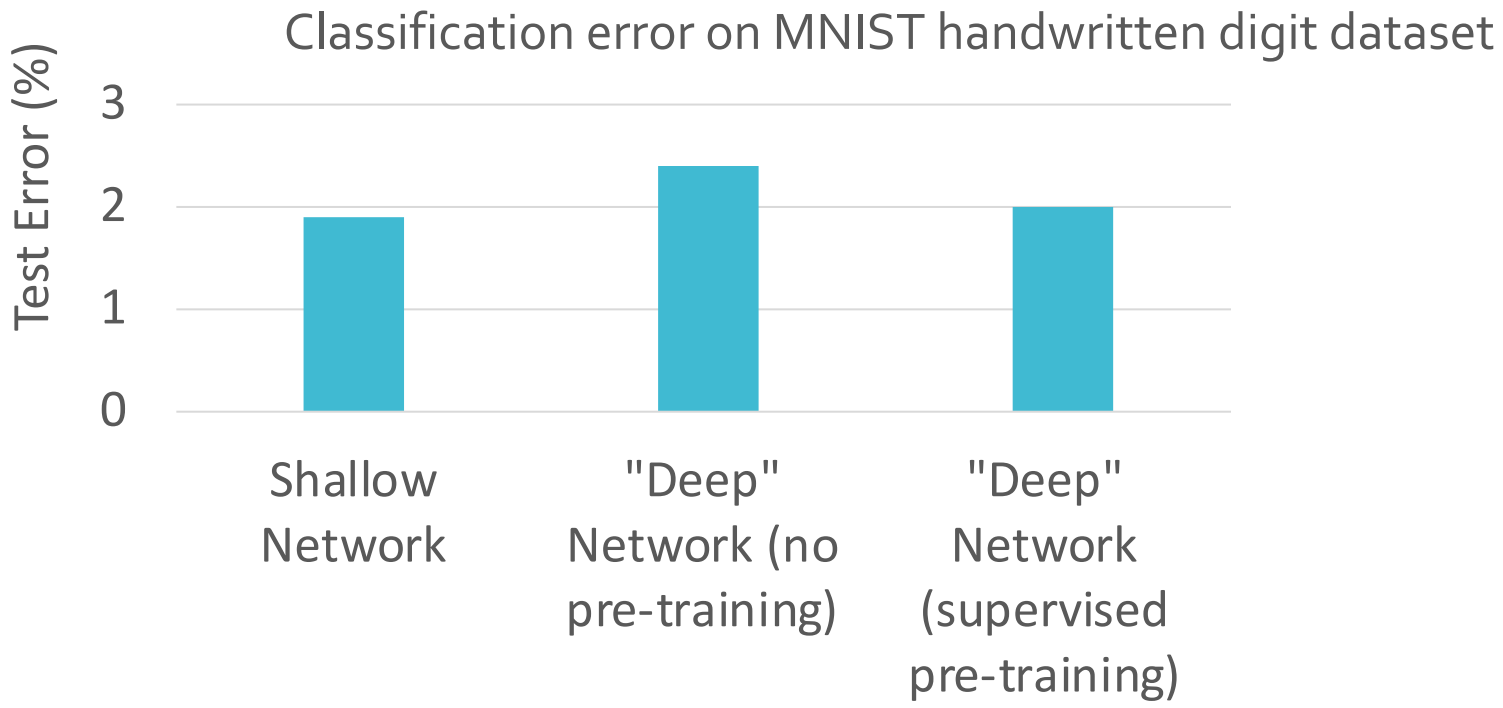
Is this the only thing we could do with the training data?

- 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)

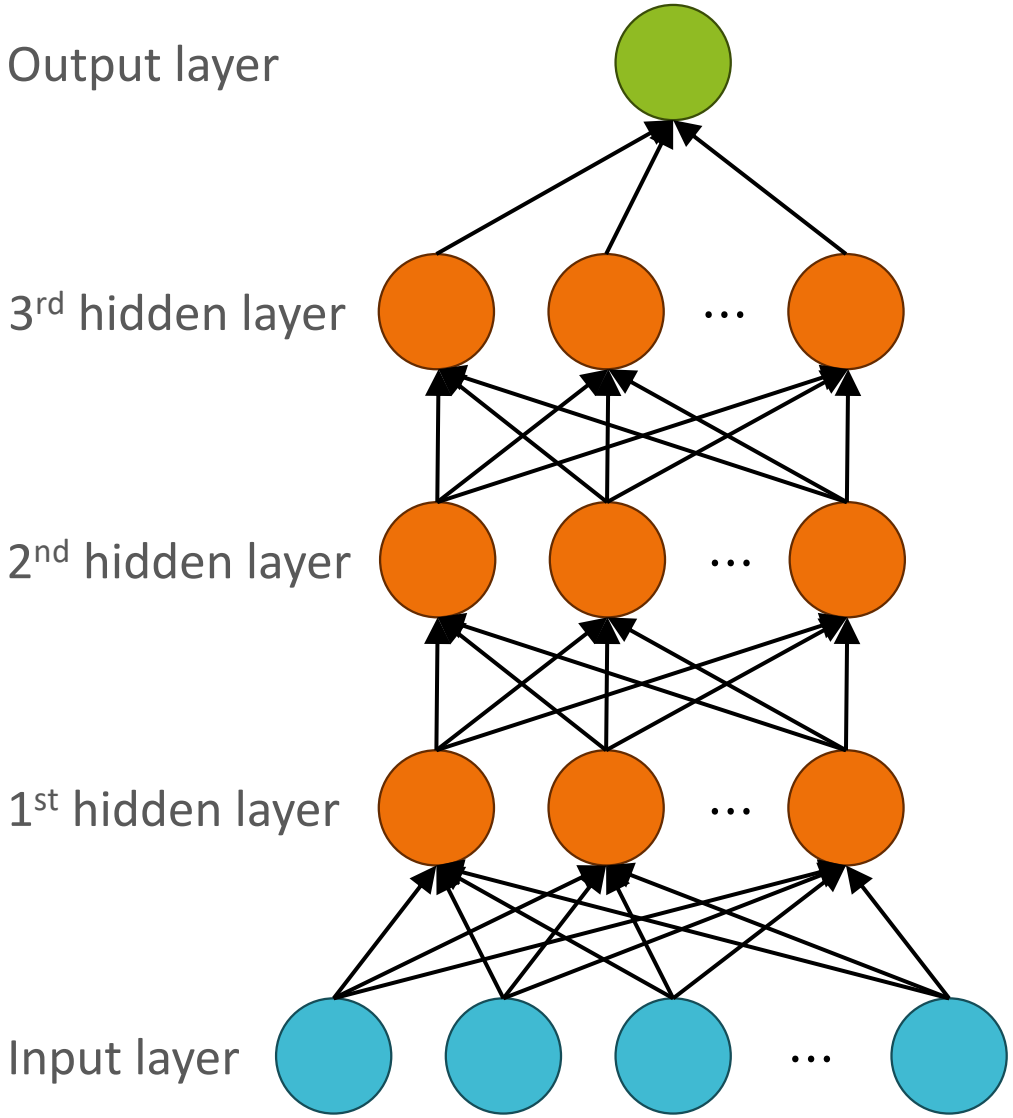
- Train each layer of the network iteratively using the training dataset *to learn useful representations*
- Idea: a good representation is one preserves a lot of information and could be used to recreate the inputs



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

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

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



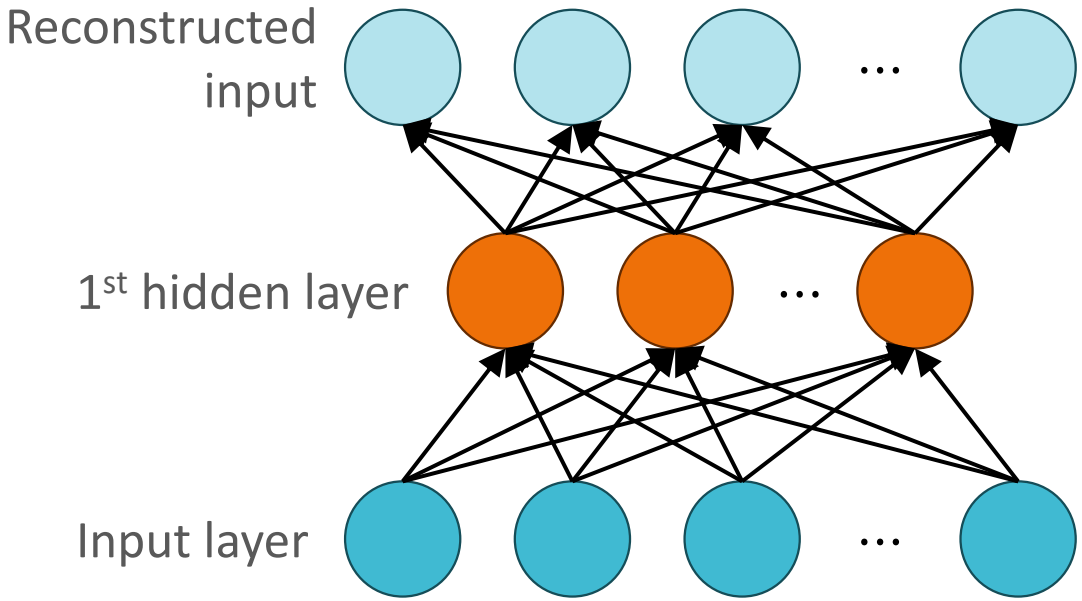
Source: https://www.cs.toronto.edu/~larocheh/publications/dbn_supervised_tr1282.pdf

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*



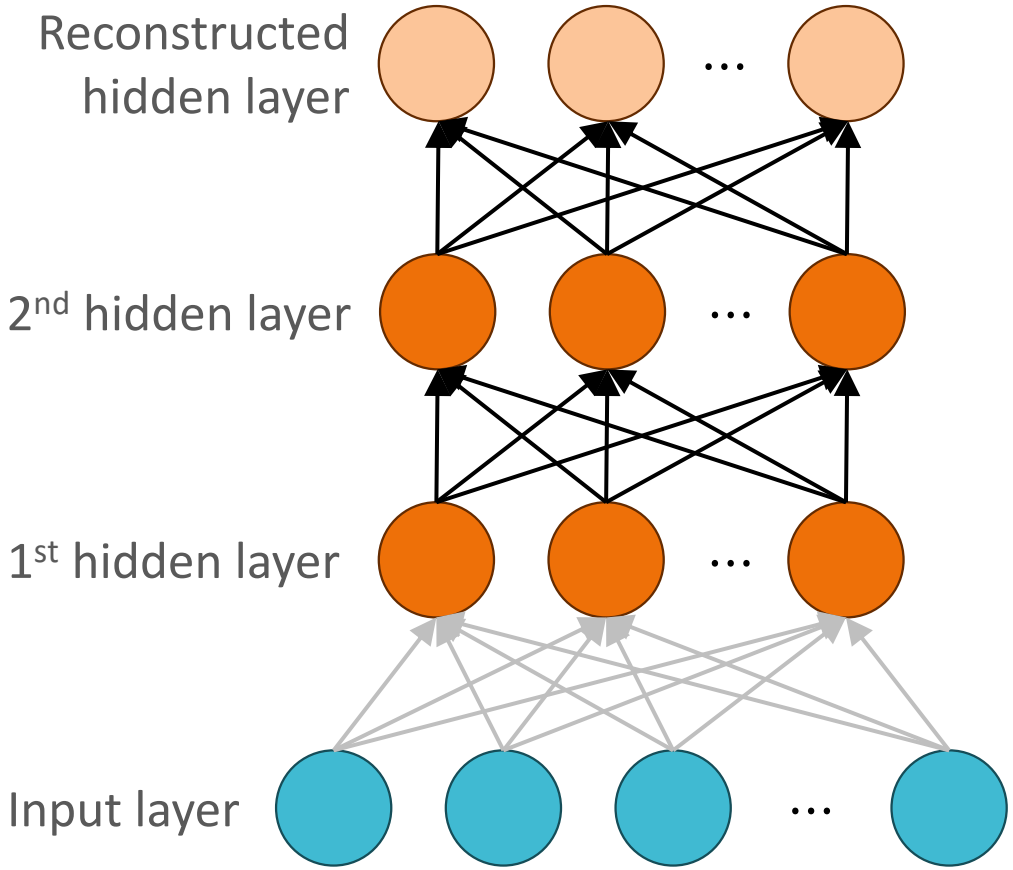
Source: https://www.cs.toronto.edu/~larocheh/publications/dbn_supervised_tr1282.pdf

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*



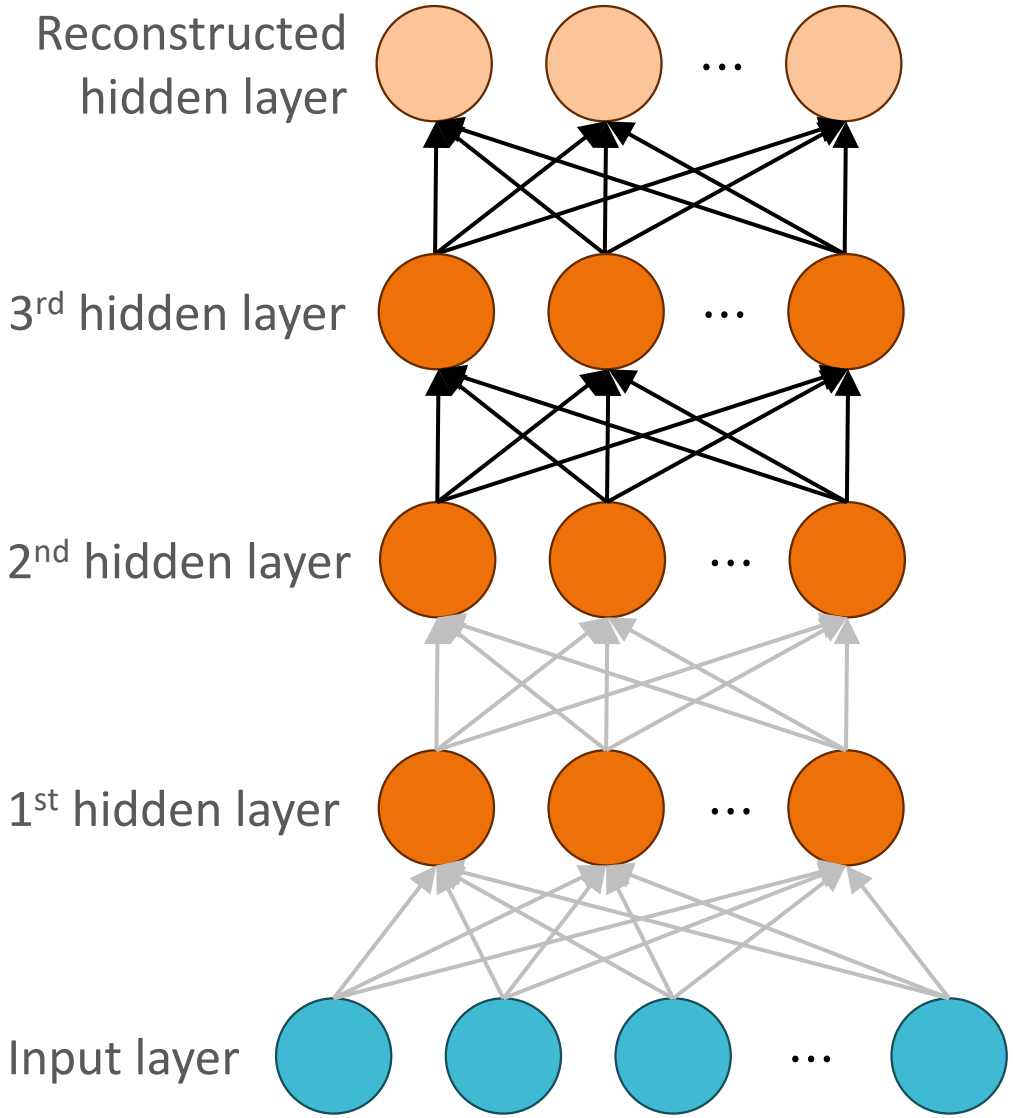
Source: https://www.cs.toronto.edu/~larocheh/publications/dbn_supervised_tr1282.pdf

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*



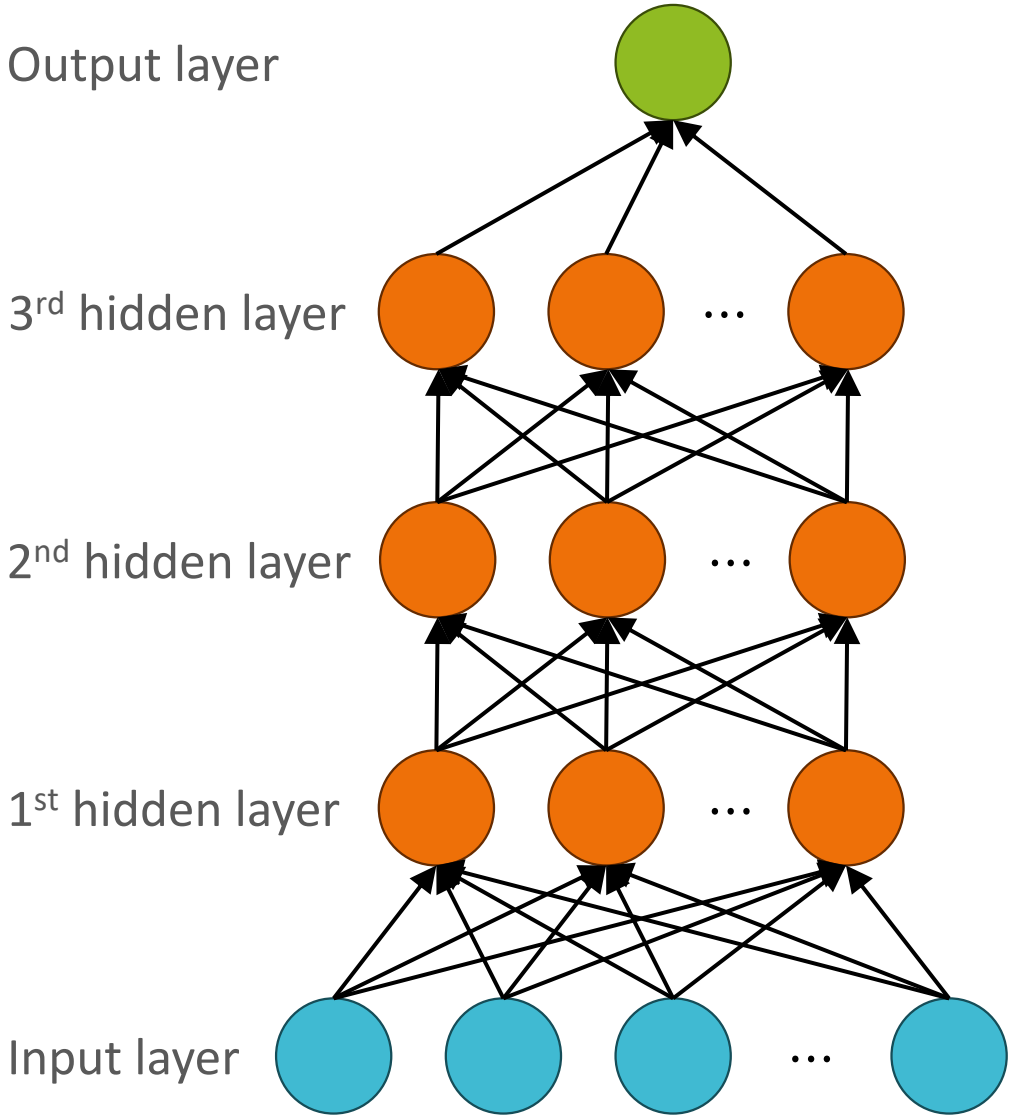
Source: https://www.cs.toronto.edu/~larocheh/publications/dbn_supervised_tr1282.pdf

Fine-tuning (Bengio et al., 2006)

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

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

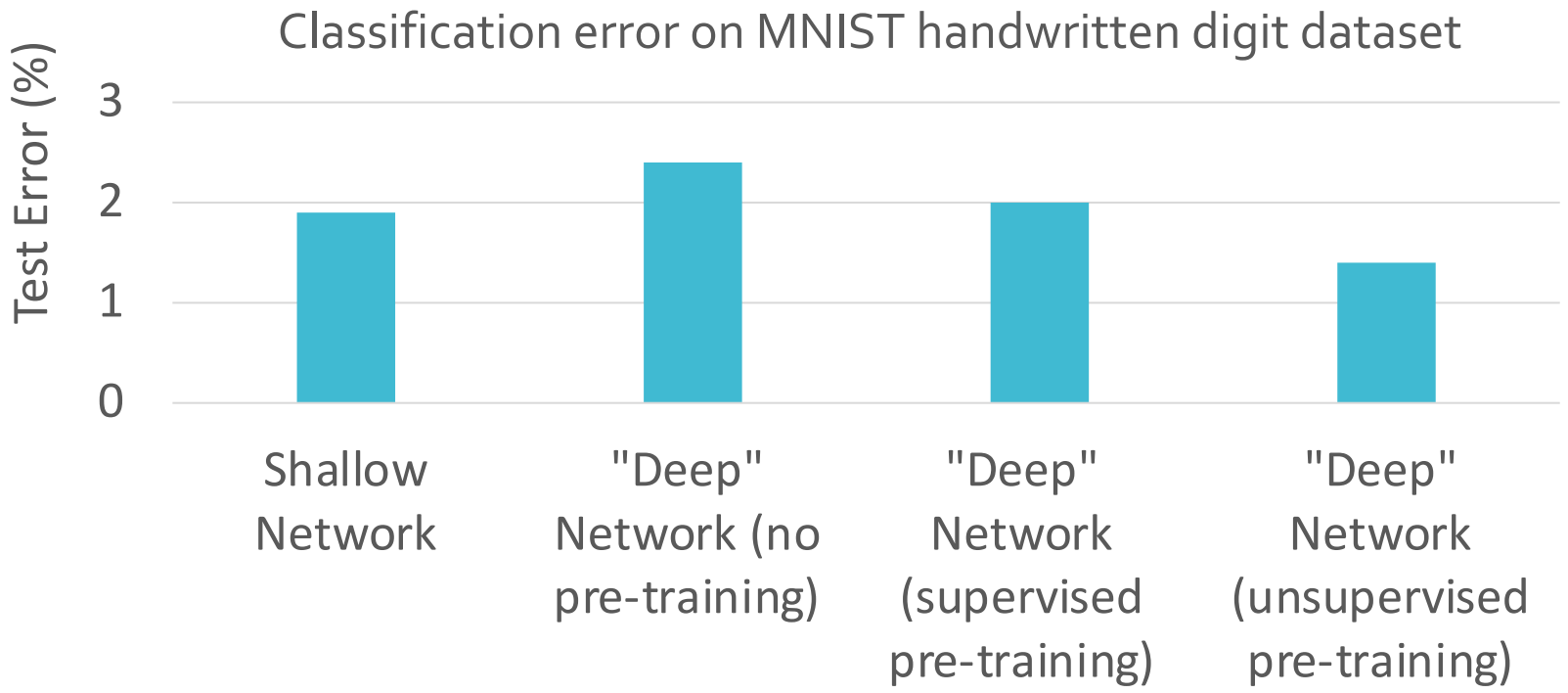
- When fine-tuning, we're effectively swapping out the last layer and fitting all the weights to the training dataset



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

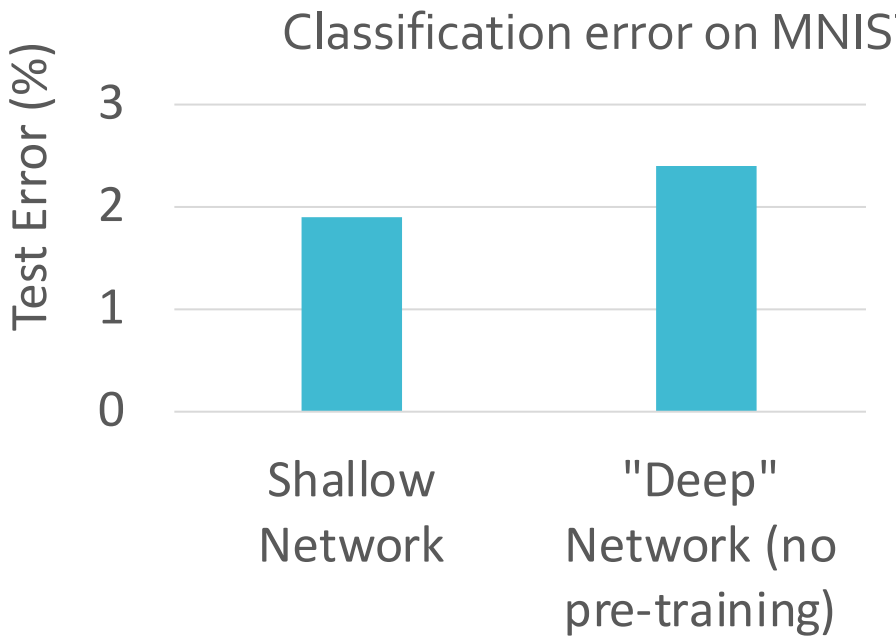
Unsupervised Pre-training (Bengio et al., 2006)

- 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 the author of children's books
- 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 the author of children's books
- 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 the author of children's books
- 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

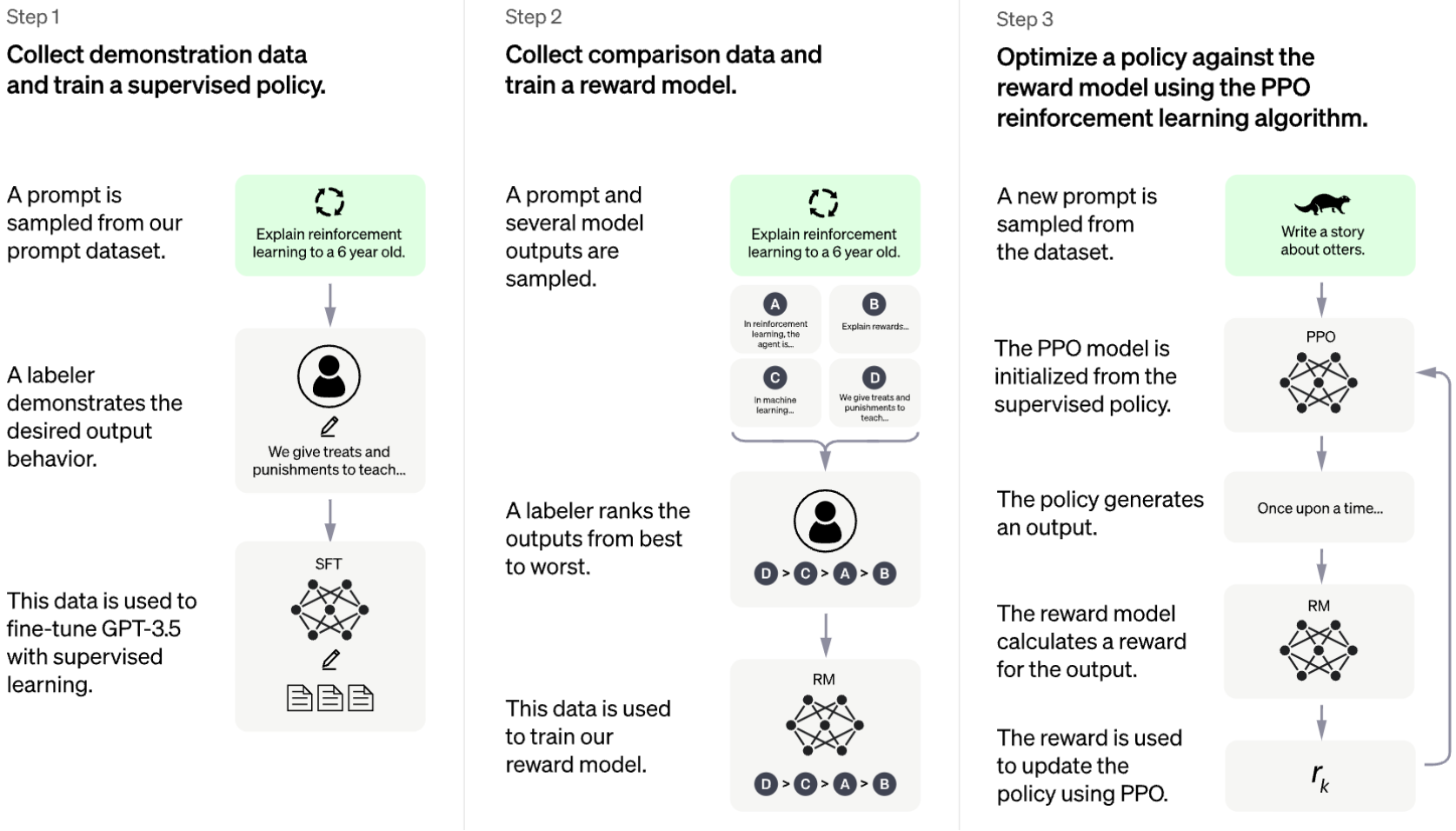
Another dose of Reality

- You have some niche task that you want to apply machine learning to e.g., predicting the author of children's books
- 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
  A. the concept of labelled data doesn't apply to your task i.e., not every input has a "correct" label e.g., chatbots?
  B. you don't have enough data to fine-tune your model?

Reinforcement Learning from Human Feedback (RLHF)

- Insight: for many machine learning tasks, there is no universal ground truth, e.g., there are lots of possible ways to respond to a question or prompt.
- Idea: use human feedback to determine how good or bad some prediction/response is!
- Issue: if the input space is huge (e.g., all possible chat prompts), to train a good model, we might need tons and tons of (potentially expensive) human annotation…
- Idea: use a small number of annotations to learn a "reward" function!

Reinforcement Learning from Human Feedback (RLHF)

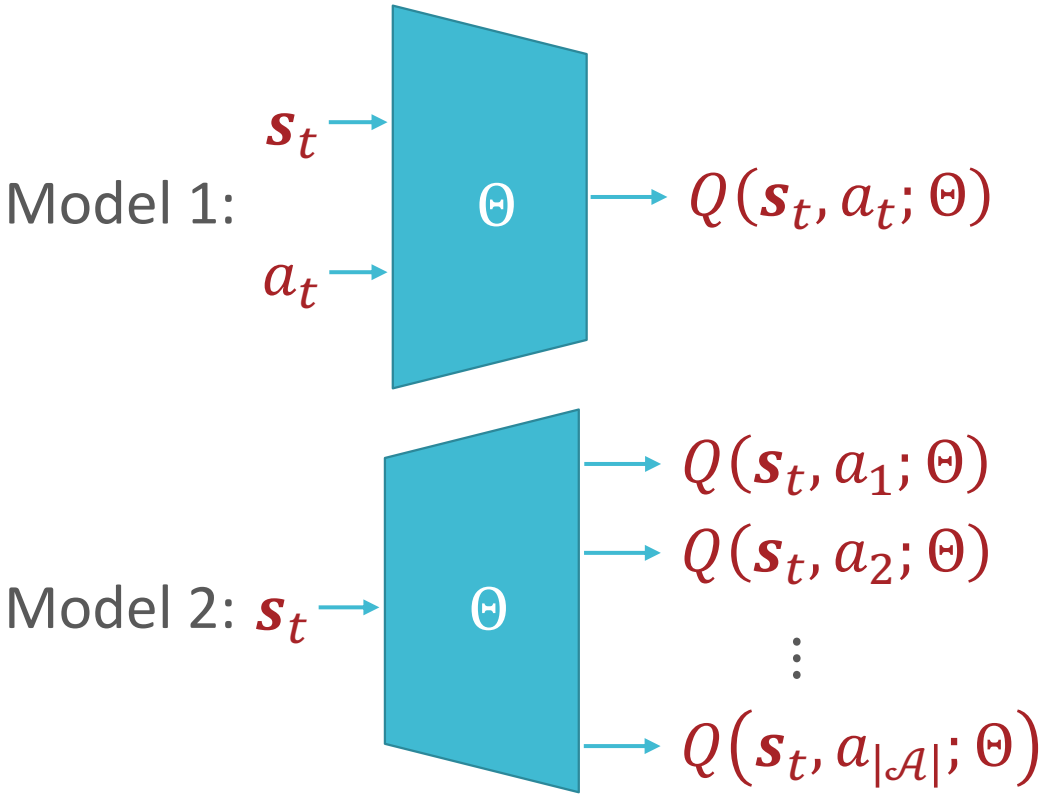


- RLHF is a special form of fine-tuning that uses *proximal policy optimization* (PPO)

Source: https://openai.com/blog/chatgpt

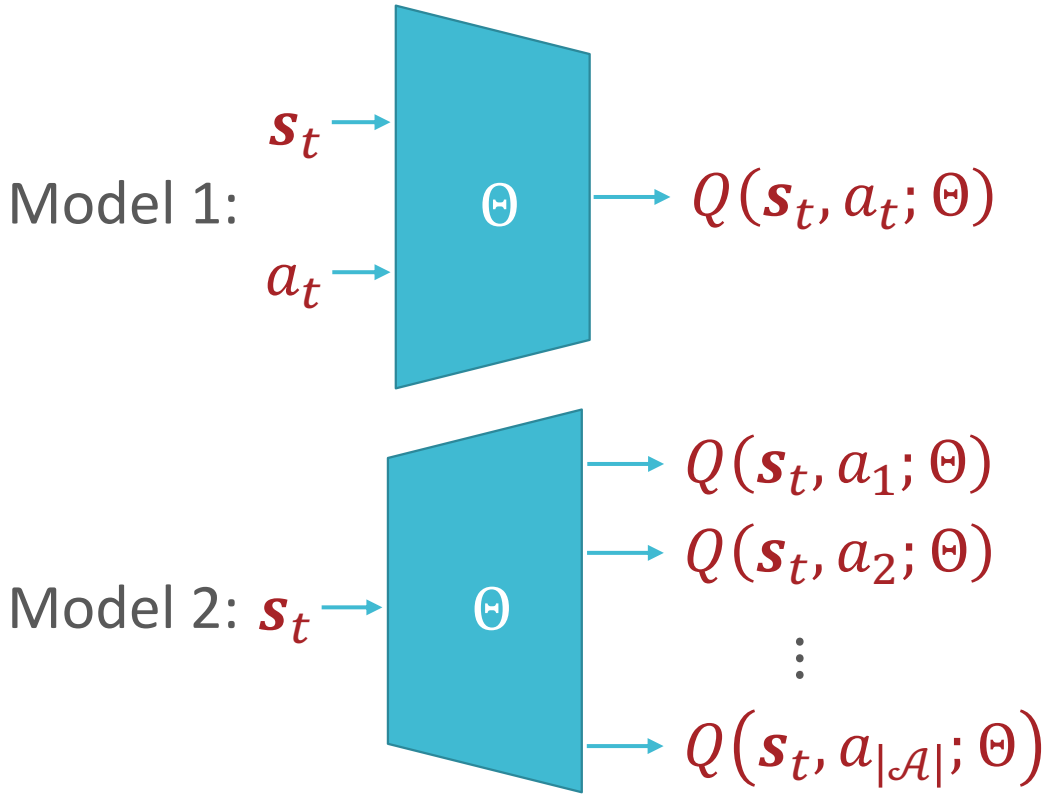
Recall: Deep Q-learning

- Represent states using some feature vector $\boldsymbol{s}_t \in \mathbb{R}^M$
  e.g. for Go, $\boldsymbol{s}_t = [1, 0, -1, \dots, 1]^T$
- Define a *differentiable* function that approximates Q



What if instead of optimizing the Q-function, we could optimize the policy directly?

- Represent states using some feature vector $\boldsymbol{s}_t \in \mathbb{R}^M$
  e.g. for Go, $\boldsymbol{s}_t = [1, 0, -1, \dots, 1]^T$
- Define a *differentiable* function that approximates Q

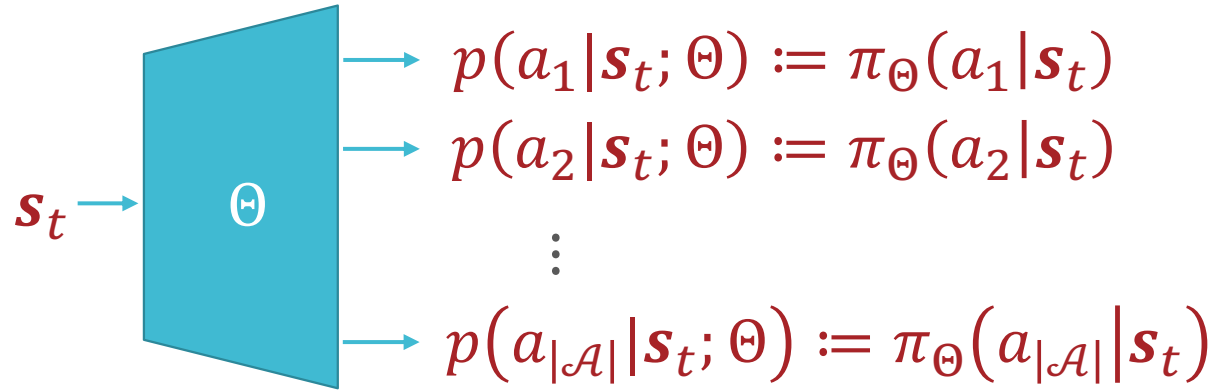


Parametrized Stochastic Policies

- Represent states using some feature vector $\boldsymbol{s}_t \in \mathbb{R}^M$
  e.g. for Go, $\boldsymbol{s}_t = [1, 0, -1, \ldots, 1]^T$
- Define a *differentiable* function that specifies a *stochastic* policy π_Θ
- Minimize the negative expected total reward w.r.t. Θ

$$\ell(\Theta) = -\mathbb{E}_{\pi_\Theta}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right]$$

Model:



Okay… but how on earth do we compute the gradient of this thing?

- Represent states using some feature vector $\boldsymbol{s}_t \in \mathbb{R}^M$
  e.g. for Go, $\boldsymbol{s}_t = [1, 0, -1, \ldots, 1]^T$
- Define a *differentiable* function that specifies a stochastic policy π_Θ
- Minimize the negative expected total reward w.r.t. Θ

$$\ell(\Theta) = -\mathbb{E}_{\pi_\Theta}\left[\mathbb{E}_{p(s'|s,a)}\left[\sum_{t=0}^{\infty}\gamma^t r_t\right]\right]$$

Model: $\boldsymbol{s}_t \rightarrow \Theta \rightarrow$

$$p(a_1|\boldsymbol{s}_t;\Theta) \coloneqq \pi_\Theta(a_1|\boldsymbol{s}_t)$$
$$p(a_2|\boldsymbol{s}_t;\Theta) \coloneqq \pi_\Theta(a_2|\boldsymbol{s}_t)$$
$$\vdots$$
$$p(a_{|\mathcal{A}|}|\boldsymbol{s}_t;\Theta) \coloneqq \pi_\Theta(a_{|\mathcal{A}|}|\boldsymbol{s}_t)$$

Trajectories

- A *trajectory* $\mathrm{T} = \{\boldsymbol{s}_0, a_0, \boldsymbol{s}_1, a_1, \ldots, \boldsymbol{s}_T\}$ is one run of an agent through an MDP ending in a terminal state, $\boldsymbol{s}_T$
- Our stochastic policy and the transition distribution induce a distribution over trajectories

$$p_\Theta(\mathrm{T}) = p(\{\boldsymbol{s}_0, a_0, \boldsymbol{s}_1, a_1, \ldots, \boldsymbol{s}_T\})$$
$$= p(\boldsymbol{s}_0) \prod_{t=0}^{T-1} p(s_{t+1}|s_t, a_t)\, \pi_\Theta(a_t|\boldsymbol{s}_t)$$

  - Requires a distribution over initial states $p(\boldsymbol{s}_0)$ e.g., uniform over all states, fixed or deterministic, etc…

- If all runs end at a terminal state, then we can rewrite the negative expected total reward as

$$\ell(\Theta) = -\mathbb{E}_{p_\Theta(\mathrm{T}=\{\boldsymbol{s}_0, a_0, \ldots, \boldsymbol{s}_T\})}\left[\sum_{t=0}^{T-1} \gamma^t R(\boldsymbol{s}_t, a_t)\right] \coloneqq -\mathbb{E}_{p_\Theta(\mathrm{T})}[R(\mathrm{T})]$$

Likelihood Ratio Method a.k.a. REINFORCE (Williams, 1992)

$$\nabla_\Theta \ell(\Theta) = \nabla_\Theta\left(-\mathbb{E}_{p_\Theta(\mathrm{T})}[R(\mathrm{T})]\right) = \nabla_\Theta\left(-\int R(\mathrm{T}) p_\Theta(\mathrm{T})\, d\mathrm{T}\right)$$

$$= -\int R(\mathrm{T}) \nabla_\Theta p_\Theta(\mathrm{T})\, d\mathrm{T}$$

$$= -\int R(\mathrm{T}) \nabla_\Theta \left(p(\boldsymbol{s}_0) \prod_{t=0}^{T-1} p(s_{t+1}|s_t, a_t)\, \pi_\Theta(a_t|\boldsymbol{s}_t) \right) d\mathrm{T}$$

- Issues:
  - The transition probabilities $p(s_{t+1}|s_t, a_t)$ are unknown a priori
  - Computing $\nabla_\Theta p_\Theta(\mathrm{T})$ involves taking the gradient of a product

Source: https://citeseerx.ist.psu.edu/doc/10.1.1.129.8871

Likelihood Ratio Method a.k.a. REINFORCE (Williams, 1992)

$$\nabla_\Theta \ell(\Theta) = \nabla_\Theta\left(-\mathbb{E}_{p_\Theta(\mathrm{T})}[R(\mathrm{T})]\right) = \nabla_\Theta\left(-\int R(\mathrm{T}) p_\Theta(\mathrm{T})\, d\mathrm{T}\right)$$

$$= -\int R(\mathrm{T}) \nabla_\Theta p_\Theta(\mathrm{T})\, d\mathrm{T}$$

$$= -\int R(\mathrm{T}) \nabla_\Theta \left(p(\boldsymbol{s}_0) \prod_{t=0}^{T-1} p(s_{t+1}|s_t, a_t)\, \pi_\Theta(a_t|\boldsymbol{s}_t) \right) d\mathrm{T}$$

- Insight:

$$\nabla_\Theta p_\Theta(\mathrm{T}) = \frac{p_\Theta(\mathrm{T})}{p_\Theta(\mathrm{T})} \nabla_\Theta p_\Theta(\mathrm{T}) = p_\Theta(\mathrm{T}) \nabla_\Theta (\log p_\Theta(\mathrm{T}))$$

$$\log p_\Theta(\mathrm{T}) = \log p(s_0) + \sum_{t=0}^{T-1} \log p(s_{t+1}|s_t, a_t) + \log \pi_\Theta(a_t|\boldsymbol{s}_t)$$

$$\nabla_\Theta (\log p_\Theta(\mathrm{T})) = \sum_{t=0}^{T-1} \nabla_\Theta \log \pi_\Theta(a_t|\boldsymbol{s}_t)$$ ← No longer depends on $p(s_{t+1}|s_t, a_t)$!

Likelihood Ratio Method a.k.a. REINFORCE (Williams, 1992)

$$\nabla_\Theta \ell(\Theta) = \nabla_\Theta\left(-\mathbb{E}_{p_\Theta(\mathrm{T})}[R(\mathrm{T})]\right) = \nabla_\Theta\left(-\int R(\mathrm{T}) p_\Theta(\mathrm{T})\, d\mathrm{T}\right)$$

$$= -\int R(\mathrm{T}) \nabla_\Theta p_\Theta(\mathrm{T})\, d\mathrm{T} = -\int R(\mathrm{T}) \nabla_\Theta (\log p_\Theta(\mathrm{T})) p_\Theta(\mathrm{T}) d\mathrm{T}$$

$$= -\mathbb{E}_{p_\Theta(\mathrm{T})}[R(\mathrm{T}) \nabla_\Theta (\log p_\Theta(\mathrm{T}))]$$

$$\approx -\frac{1}{N}\sum_{n=1}^{N} R\left(\mathrm{T}^{(n)}\right) \nabla_\Theta\left(\log p_\Theta\left(\mathrm{T}^{(n)}\right)\right)$$

(where $\mathrm{T}^{(n)} = \left\{\boldsymbol{s}_0^{(n)}, a_0^{(n)}, \boldsymbol{s}_1^{(n)}, a_1^{(n)}, \ldots, \boldsymbol{s}_{T^{(n)}}^{(n)}\right\}$ is a sampled trajectory)

$$= -\frac{1}{N}\sum_{n=1}^{N}\left(\sum_{t=0}^{T^{(n)}-1} \gamma^t R\left(\boldsymbol{s}_t^{(n)}, a_t^{(n)}\right)\right)\left(\sum_{t=0}^{T^{(n)}-1} \nabla_\Theta \log \pi_\Theta\left(a_t^{(n)} \middle| \boldsymbol{s}_t^{(n)}\right)\right)$$

Policy Gradient Methods

- Practical considerations:
  - Not compatible with deterministic policies (would require knowledge of the transition probabilities)
  - Sampled trajectories/rewards can be highly variable, which leads to unstable estimates of the expectation
    - Can compare sampled rewards against a constant *baseline* to get an *advantage function* (Peters and Schaal, 2008): $A(\mathrm{T}) = R(\mathrm{T}) - B$

Policy Gradient Methods

- Practical considerations:
  - Policy gradient methods are *on-policy*: they require using the current (potentially bad) policy to sample (a lot of) trajectories…
    - *Trust region methods* (Schulman et al., 2015) impose a constraint on how far the policy distribution can shift from one iteration to the next (in terms of a KL divergence)
    - *Proximal policy optimization* (Schulman et al., 2017) limits how much the magnitude of the objective function can change from one iteration to the next via clipping

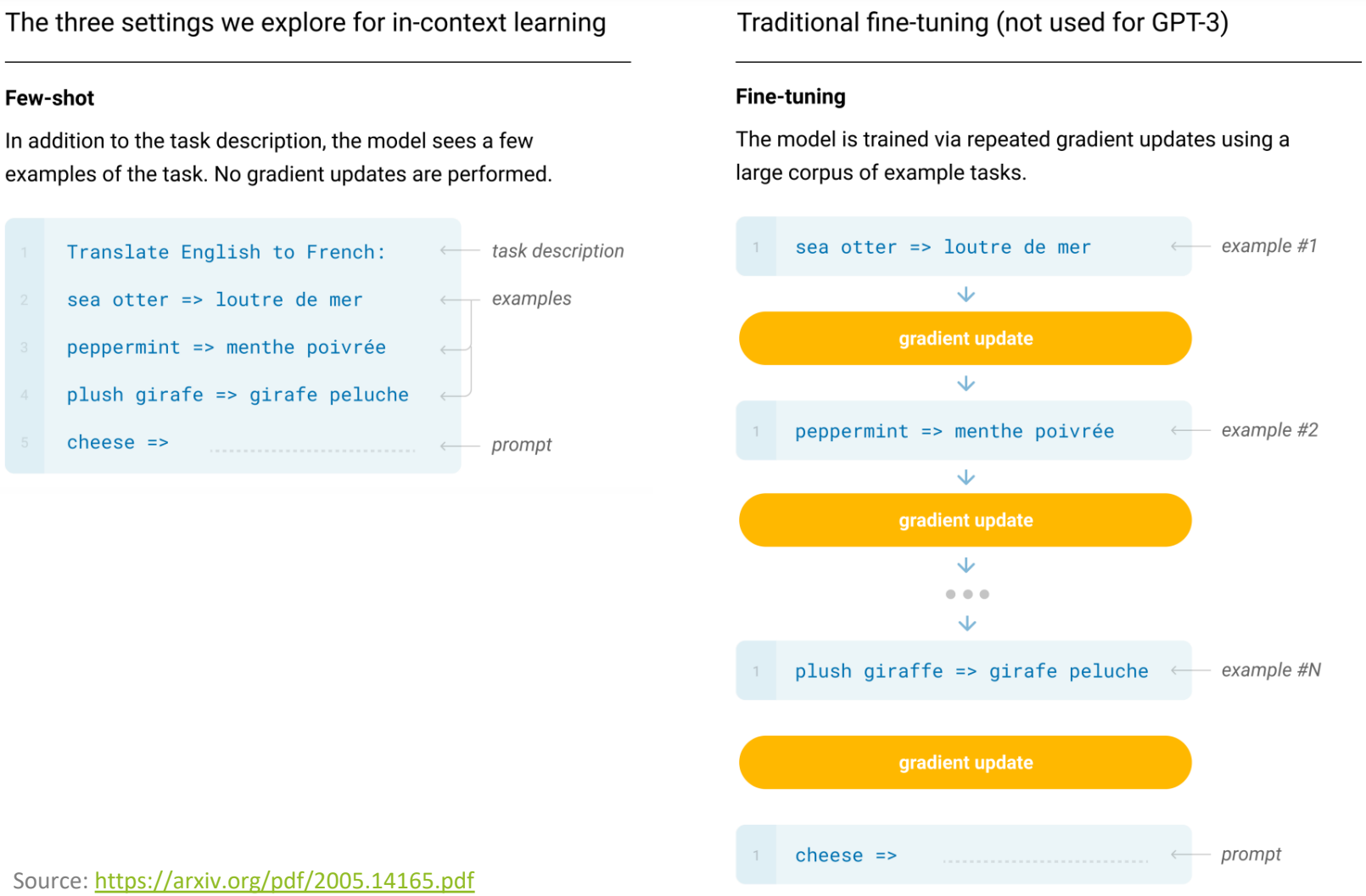
In-context Learning

- 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 examples~~s~~ to the model as input, *without performing any updates to the parameters*

The three settings we explore for in-context learning

One-shot

In addition to the task description, the model sees a single example of the task. No gradient updates are performed.

```
1  Translate English to French:        ← task description
2  sea otter => loutre de mer          ← example
3  cheese =>                           ← prompt
```

Traditional fine-tuning (not used for GPT-3)

Fine-tuning

The model is trained via repeated gradient updates using a large corpus of example tasks.

```
1  sea otter => loutre de mer          ← example #1
```

gradient update

```
1  peppermint => menthe poivrée        ← example #2
```

gradient update

...

```
1  plush giraffe => girafe peluche     ← example #N
```

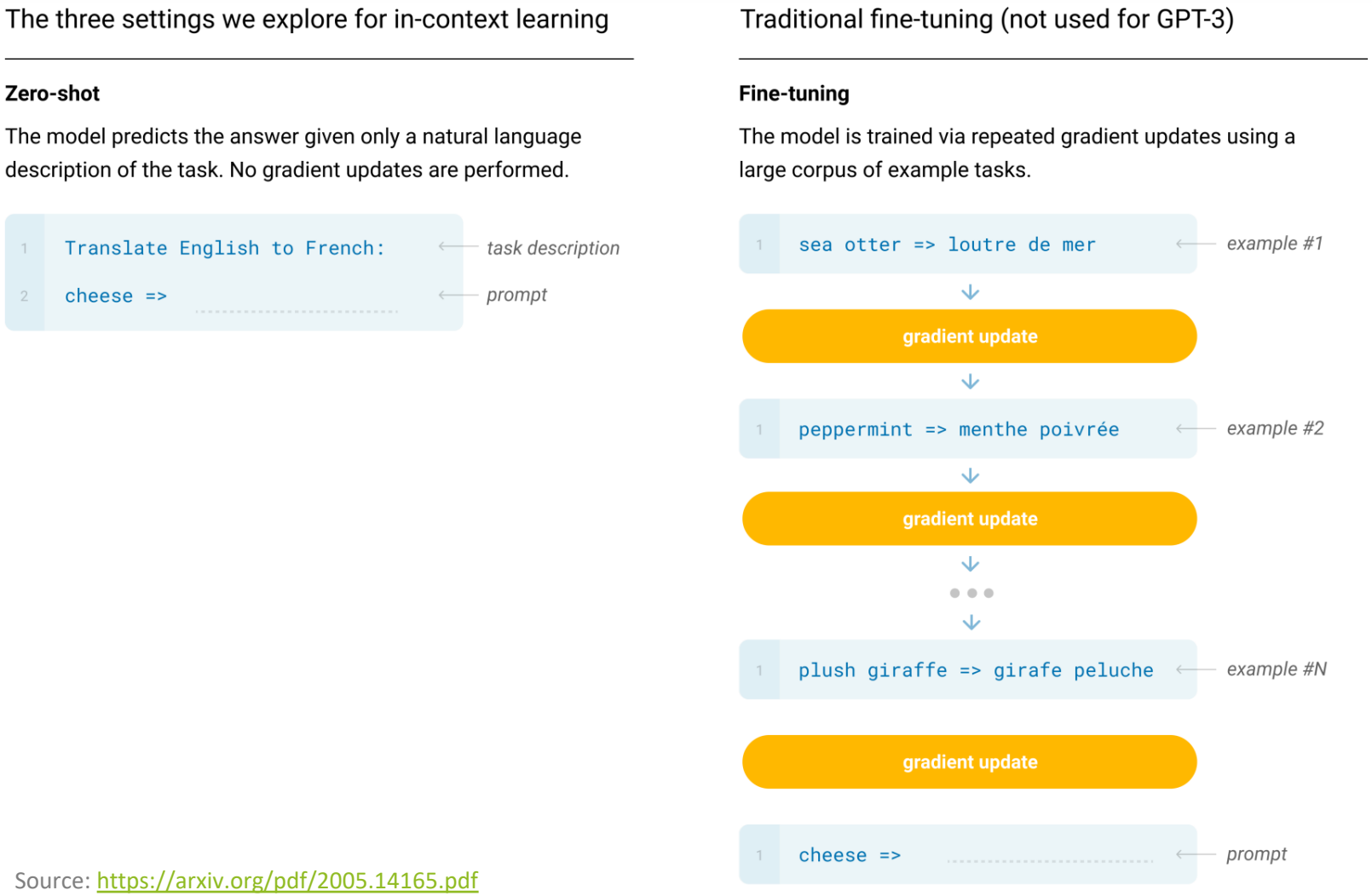
gradient update

```
1  cheese =>                           ← prompt
```

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

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

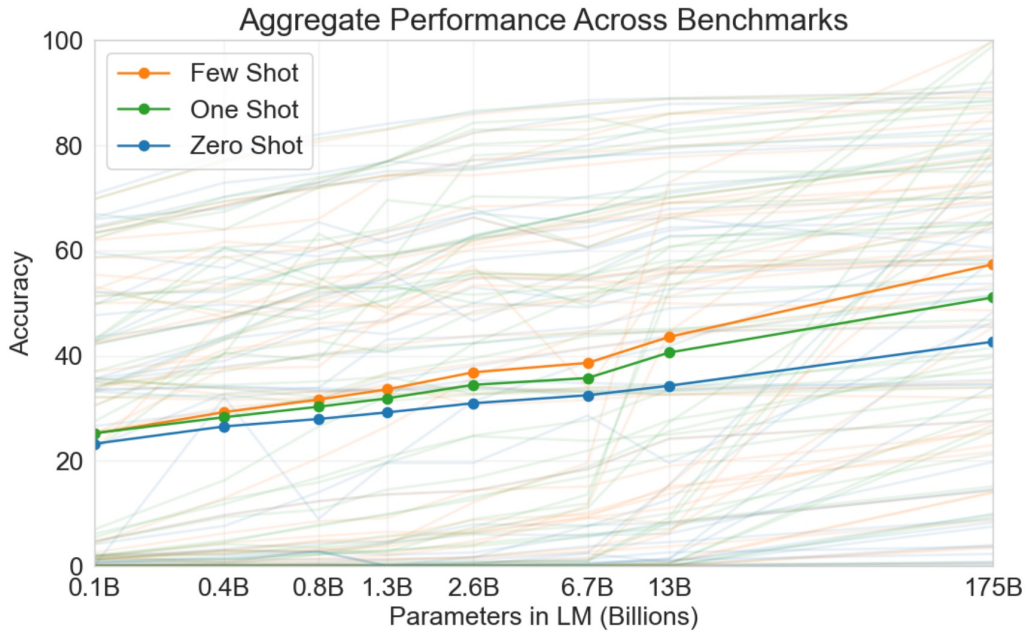
- Idea: leverage the LLM's context window by passing ~~a~~ ~~few one~~ zero(!) examples to the model as input, *without performing any updates to the parameters*



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
- For tasks with subjective outputs, models can be fine-tuned using reinforcement learning with human feedback
  - Uses (proximal) policy optimization to optimize a parametrized policy directly
- Some tasks can be performed by a pretrained LLM without any fine-tuning via in-context learning