10-301/601: Introduction to Machine Learning Lecture 19 – Pre-training, Fine-tuning & In-Context Learning

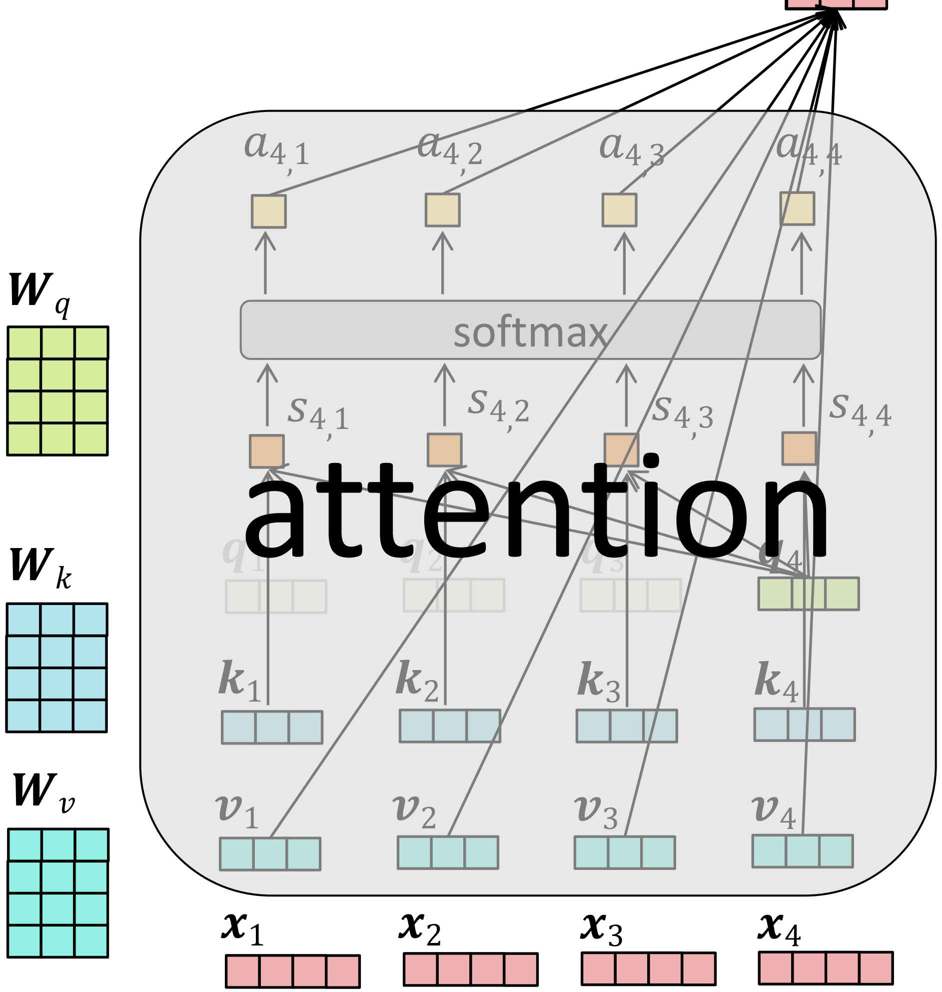
Henry Chai & Matt Gormley

11/6/23

Front Matter

- Announcements:
  - Exam 2 on 11/9 (Thursday!)
    - All topics from Lecture 8 - 16 are in-scope
    - Exam 1 content may be referenced but will not be the primary focus of any question
    - No electronic devices (you won’t need them!)
    - You may bring one letter-size sheet of notes; you can put *whatever* you want on *both sides*

Recall: Scaled Dot-Product Attention



$$x'_4 = \sum_{j=1}^{4} a_{4,j} v_j$$

$$a_{4,j} = \text{softmax}(s_{4,j})$$ Attention weights

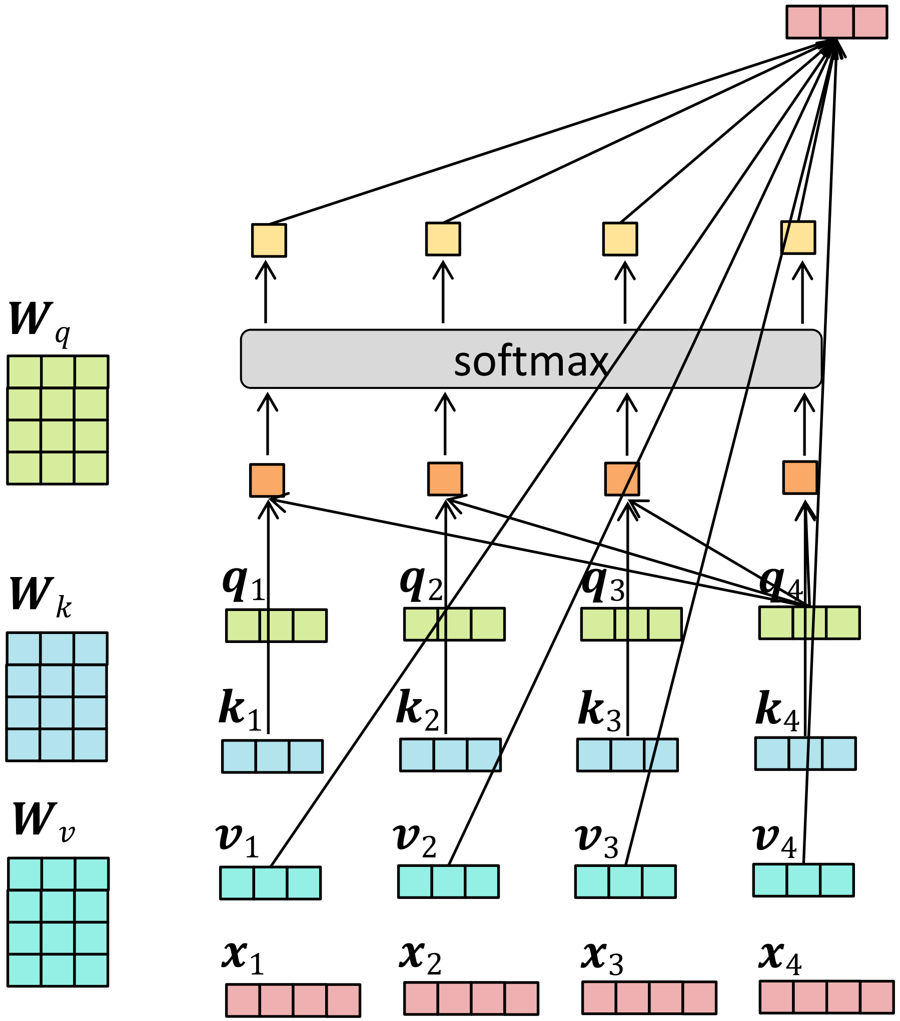
$$s_{4,j} = \frac{k_j^T q_4}{\sqrt{d_k}}$$ Scores

$$q_j = W_q^T x_j$$ Queries

$$k_j = W_k^T x_j$$ Keys

$$v_j = W_v^T x_j$$ Values

Scaled Dot-Product Attention: Matrix Form

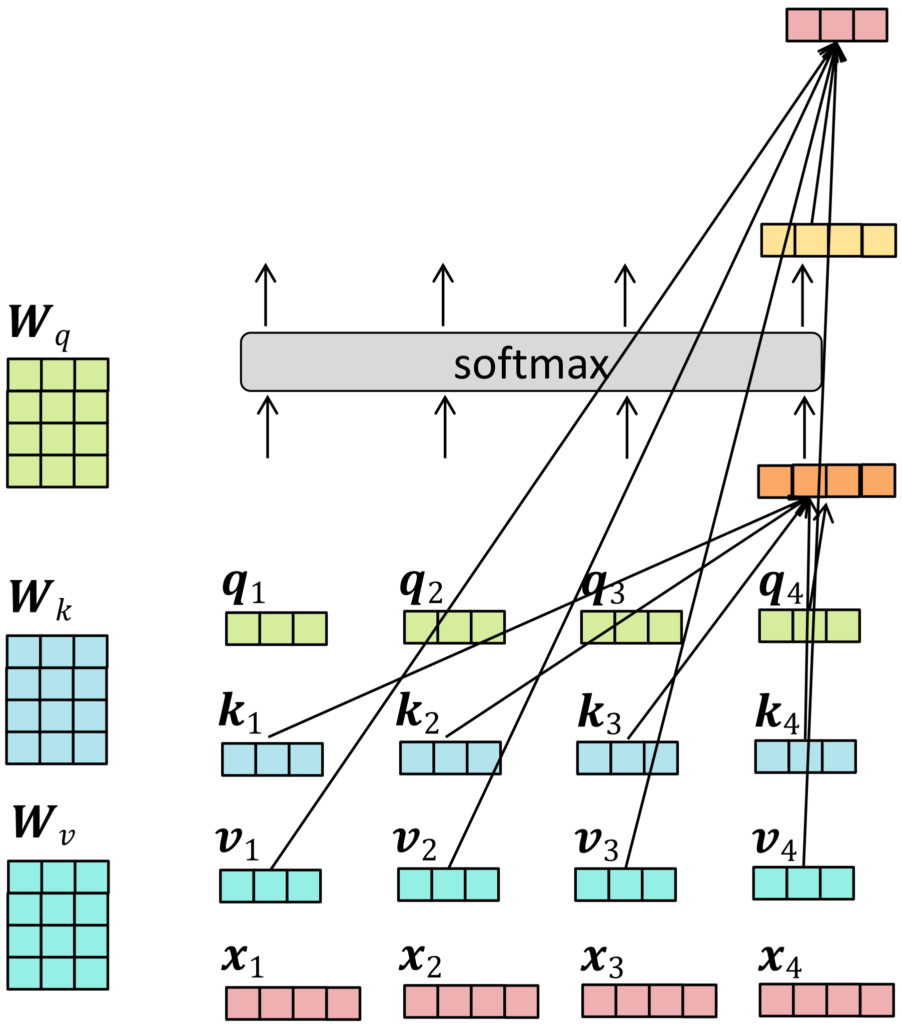


$$Q = [\boldsymbol{q}_1, \cdots, \boldsymbol{q}_N] = \boldsymbol{W}_q^T[\boldsymbol{x}_1, \cdots, \boldsymbol{x}_N]$$

$$K = [\boldsymbol{k}_1, \cdots, \boldsymbol{k}_N] = \boldsymbol{W}_k^T[\boldsymbol{x}_1, \cdots, \boldsymbol{x}_N]$$

$$V = [\boldsymbol{v}_1, \cdots, \boldsymbol{v}_N] = \boldsymbol{W}_v^T[\boldsymbol{x}_1, \cdots, \boldsymbol{x}_N]$$

Scaled Dot-Product Attention: Matrix Form

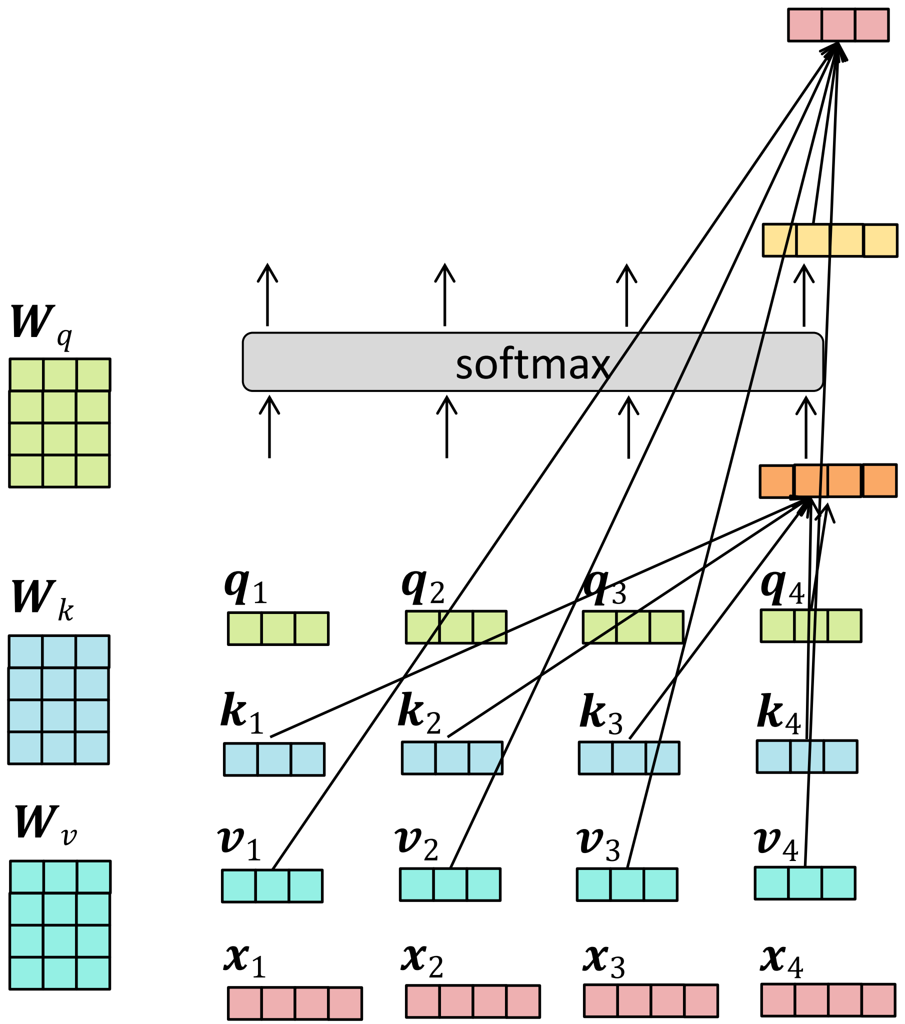


$$Q = [\boldsymbol{q}_1, \cdots, \boldsymbol{q}_N] = \boldsymbol{W}_q^T[\boldsymbol{x}_1, \cdots, \boldsymbol{x}_N]$$

$$K = [\boldsymbol{k}_1, \cdots, \boldsymbol{k}_N] = \boldsymbol{W}_k^T[\boldsymbol{x}_1, \cdots, \boldsymbol{x}_N]$$

$$V = [\boldsymbol{v}_1, \cdots, \boldsymbol{v}_N] = \boldsymbol{W}_v^T[\boldsymbol{x}_1, \cdots, \boldsymbol{x}_N]$$

Scaled Dot-Product Attention: Matrix Form

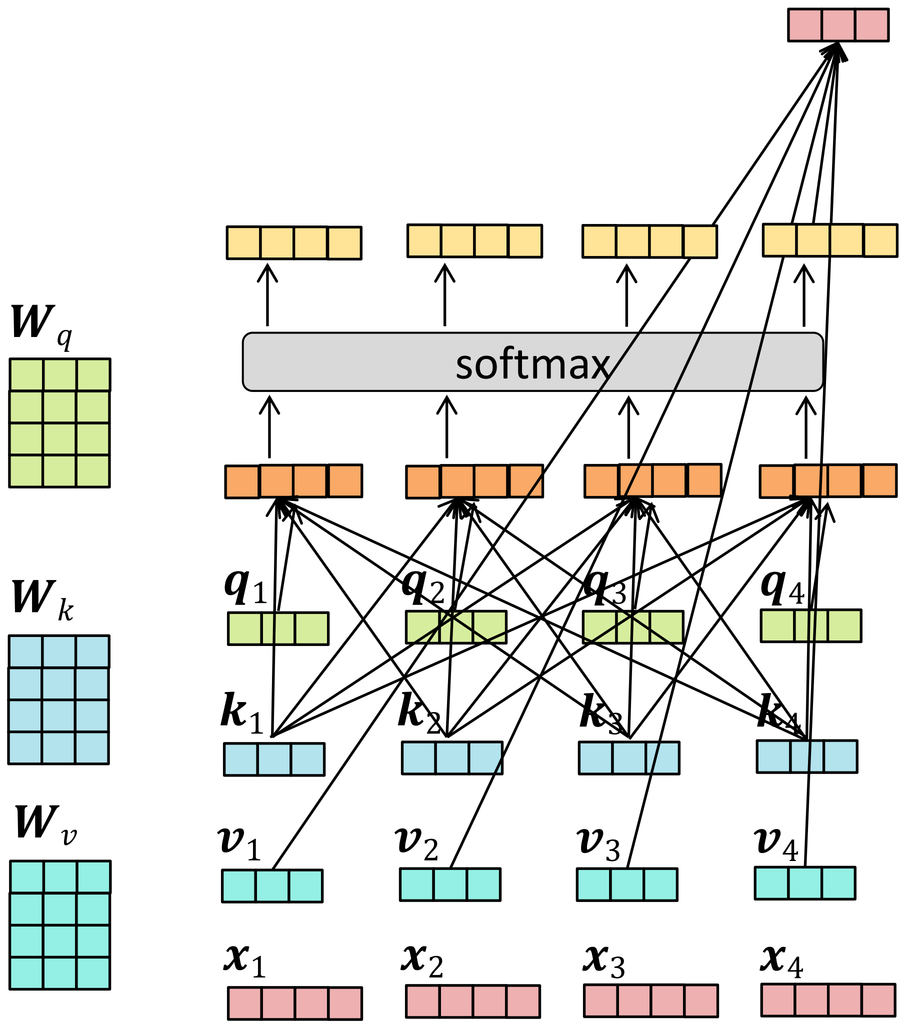


$$Q = W_q^T X^T$$

$$K = W_k^T X^T$$

$$V = W_v^T X^T$$

Scaled Dot-Product Attention: Matrix Form



$$A = \mathrm{softmax}(S)$$

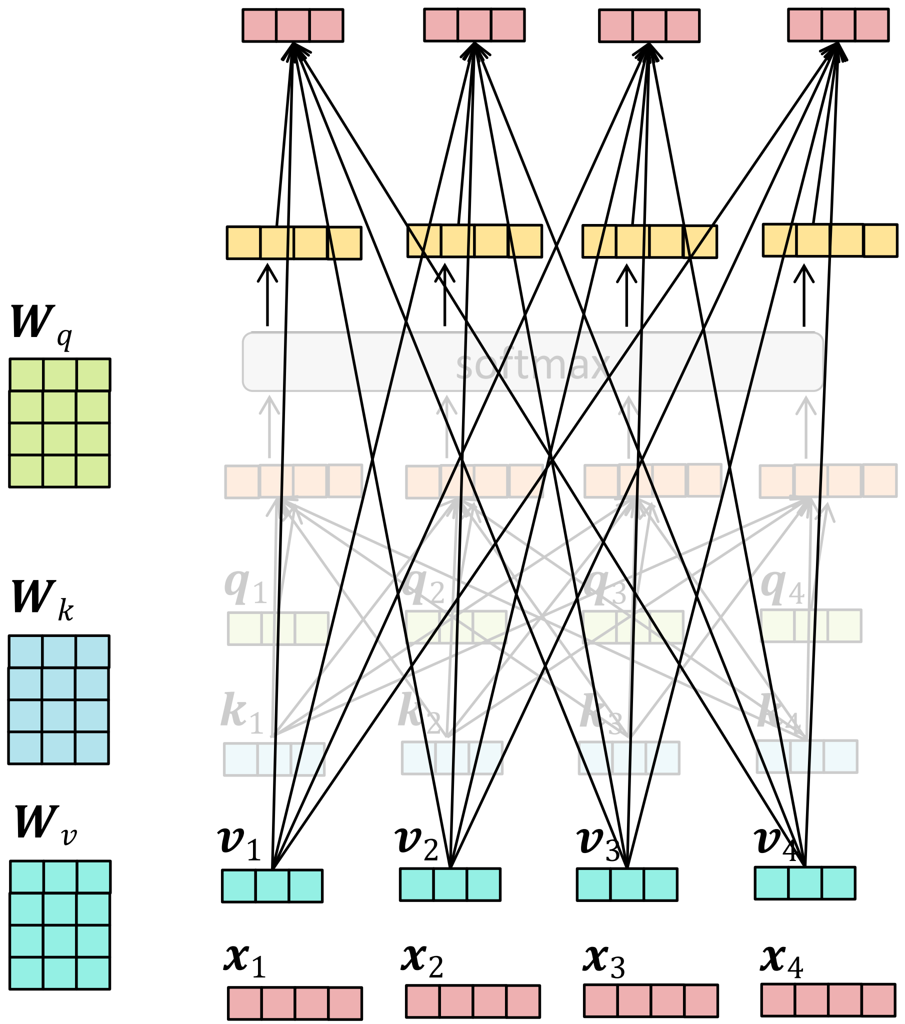
$$S = \frac{K^T Q}{\sqrt{d_k}}$$

$$Q = \boldsymbol{W}_q^T X^T$$

$$K = \boldsymbol{W}_k^T X^T$$

$$V = \boldsymbol{W}_v^T X^T$$

Scaled Dot-Product Attention: Matrix Form



$$X' = VA = \boxed{V \text{ softmax}\left(\frac{K^T Q}{\sqrt{d_k}}\right)}$$

$$A = \text{softmax}(S)$$

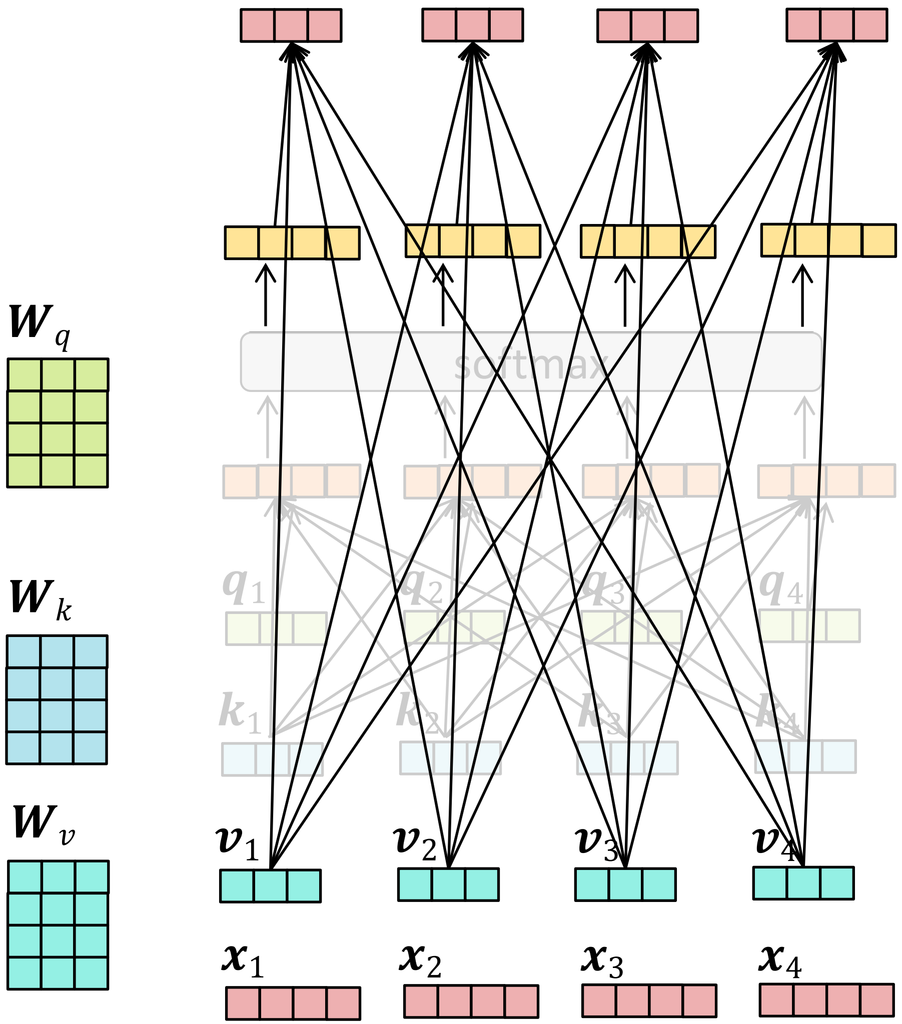
$$S = \frac{K^T Q}{\sqrt{d_k}}$$

$$Q = \boldsymbol{W}_q^T X^T$$

$$K = \boldsymbol{W}_k^T X^T$$

$$V = \boldsymbol{W}_v^T X^T$$

Holy cow, that's a lot of new arrows… do we always want/need all of those?



$$X' = VA = \boxed{V \operatorname{softmax}\left(\frac{K^T Q}{\sqrt{d_k}}\right)}$$

$$A = \operatorname{softmax}(S)$$

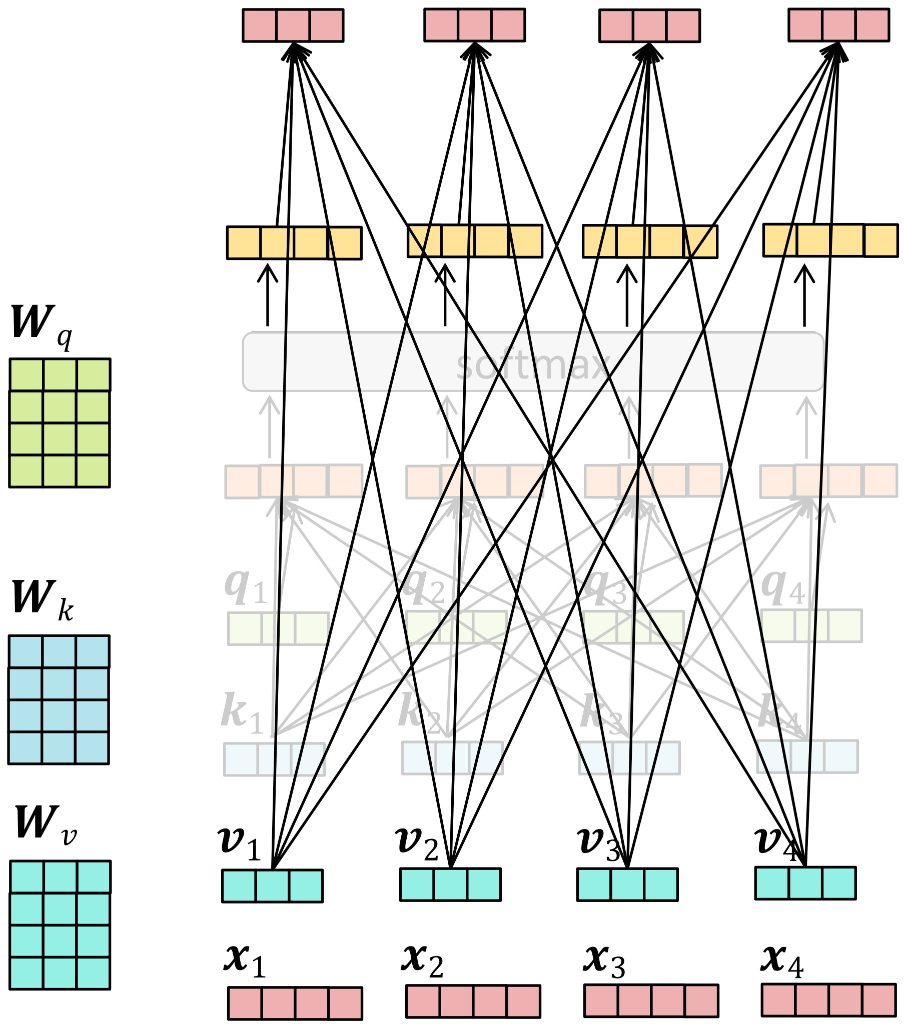
$$S = \frac{K^T Q}{\sqrt{d_k}}$$

$$Q = \boldsymbol{W}_q^T X^T$$

$$K = \boldsymbol{W}_k^T X^T$$

$$V = \boldsymbol{W}_v^T X^T$$

Decoding



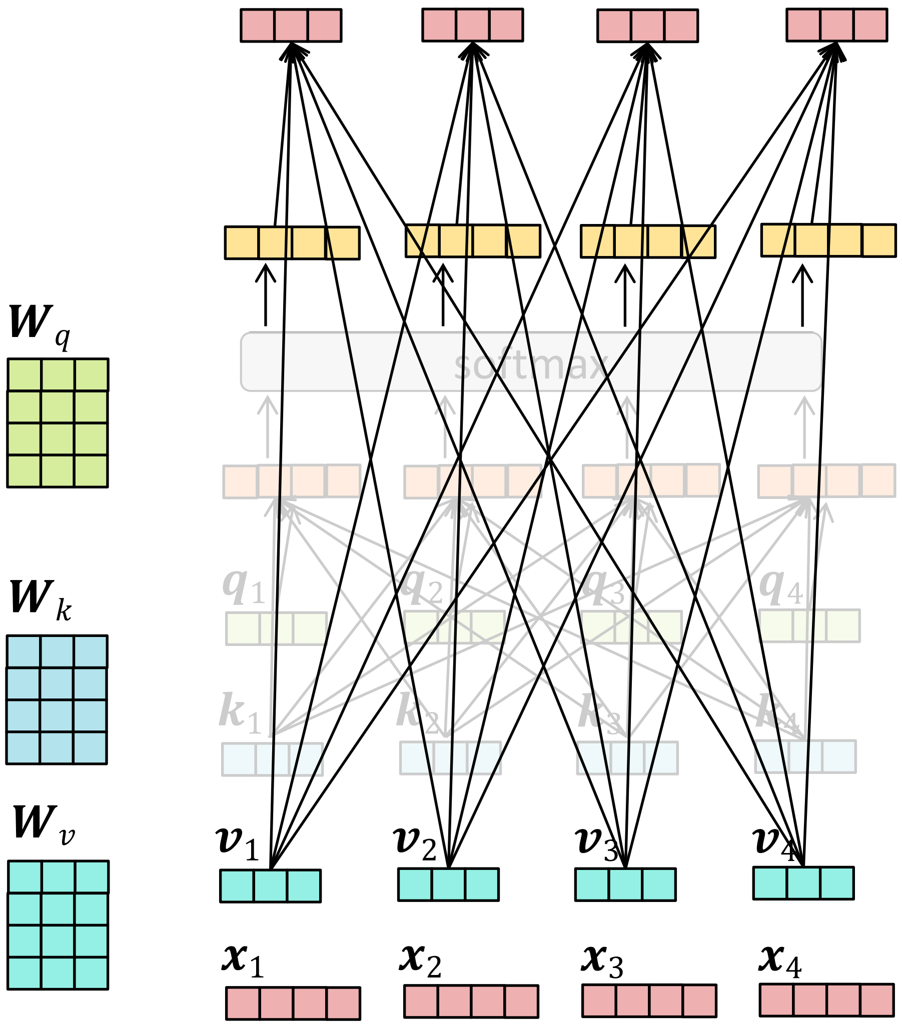
$$X' = VA = \boxed{V \text{ softmax}\left(\frac{K^T Q}{\sqrt{d_k}}\right)}$$

$$A = \text{softmax}(S)$$

- Suppose we're training our transformer to predict the next token(s) given the input…
- … then attending to tokens that come after the current token is cheating!

Masking

Idea: we can effectively delete or "mask" some of these arrows by selectively setting attention weights to 0

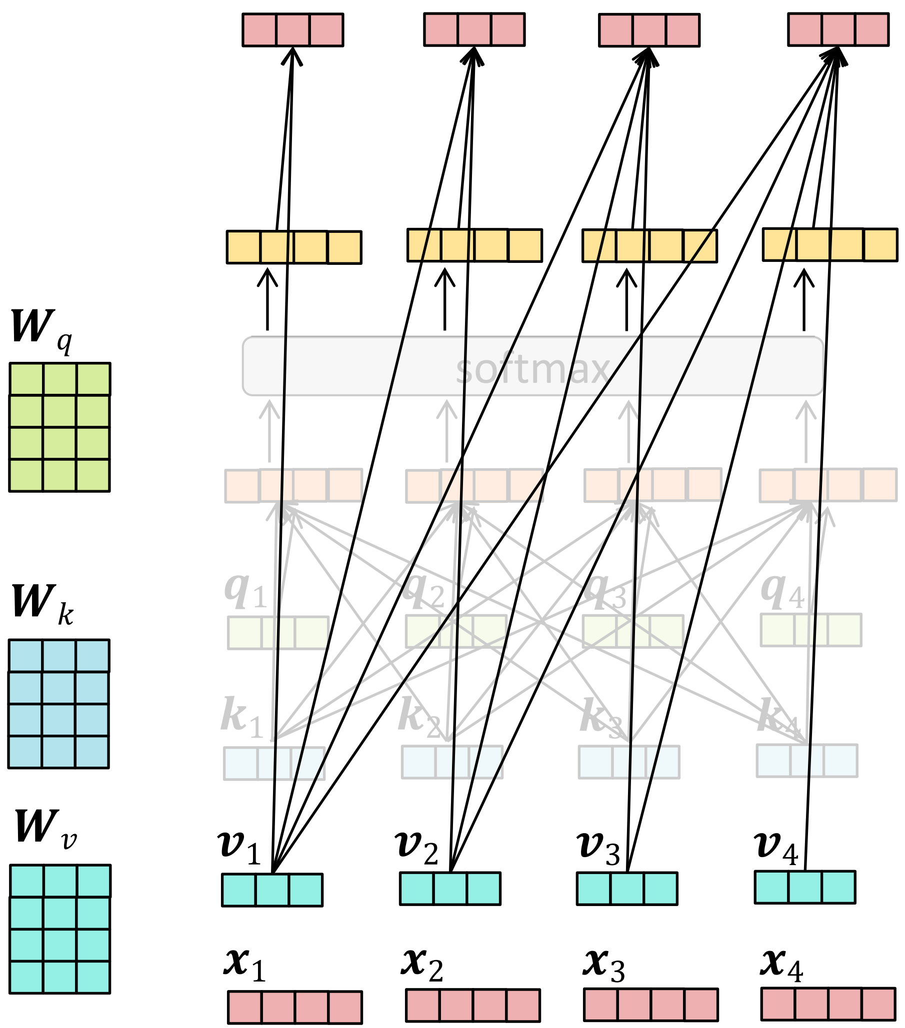


$$X' = VA = V\,\mathrm{softmax}\left(\frac{K^T Q}{\sqrt{d_k}}\right)$$

$$A = \mathrm{softmax}(S)$$

Masking

Idea: we can effectively delete or "mask" some of these arrows by selectively setting attention weights to 0



$$X' = VA = V\,\mathrm{softmax}\left(\frac{K^TQ}{\sqrt{d_k}}\right)$$

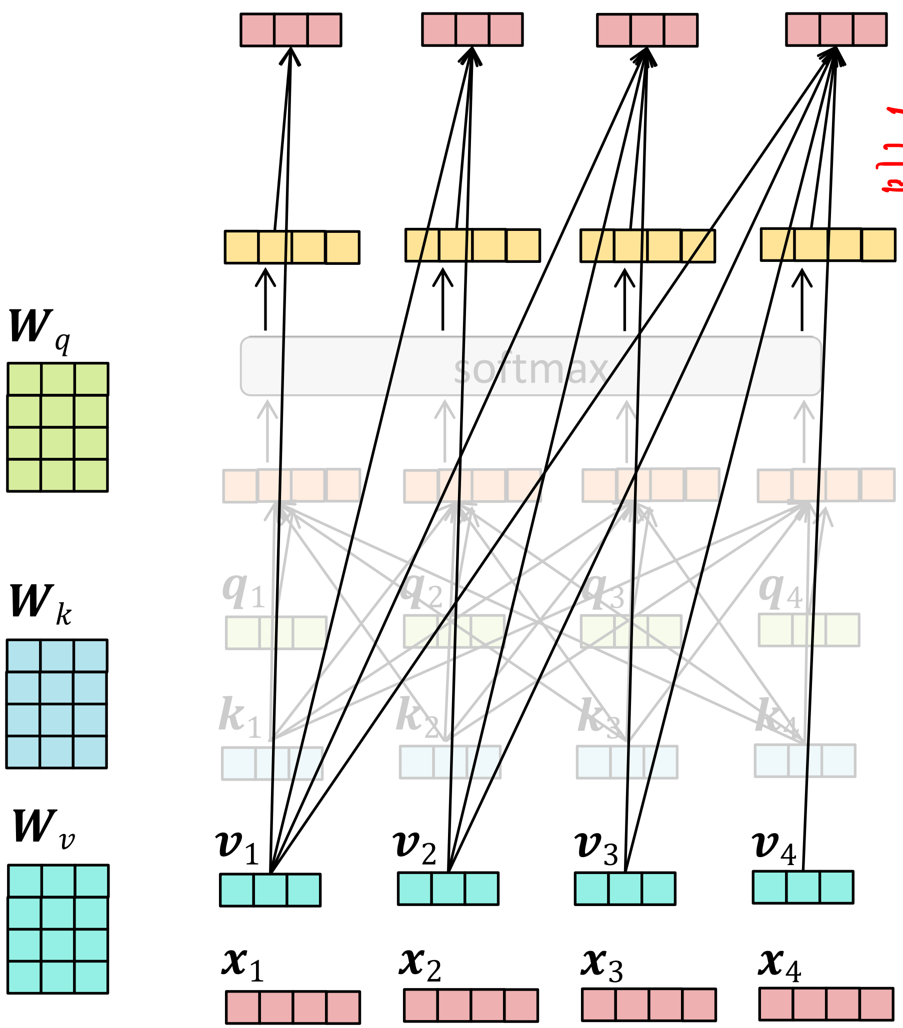
$$A = \mathrm{softmax}(S)$$

Insight: if some element in the input to the softmax is -∞, then the corresponding output is 0!

$$\frac{\exp(-\infty)}{\sum_j \exp s_j} = \frac{0}{\sum_j \exp s_j}$$

Poll Question 1: Which of the mask matrices corresponds to this set of arrows?

Idea: we can effectively delete or "mask" some of these arrows by selectively setting attention weights to 0



$$X' = V \operatorname{softmax}\left(\frac{K^T Q}{\sqrt{d_k}} + M\right)$$

$[v_1, \ldots, v_N]$

$= x_1', x_2'$

$$A_{mask} = \operatorname{softmax}(S + M)$$

A. $M = \begin{bmatrix} 0 & 0 & 0 & 0 \\ -\infty & 0 & 0 & 0 \\ -\infty & -\infty & 0 & 0 \\ -\infty & -\infty & -\infty & 0 \end{bmatrix}$

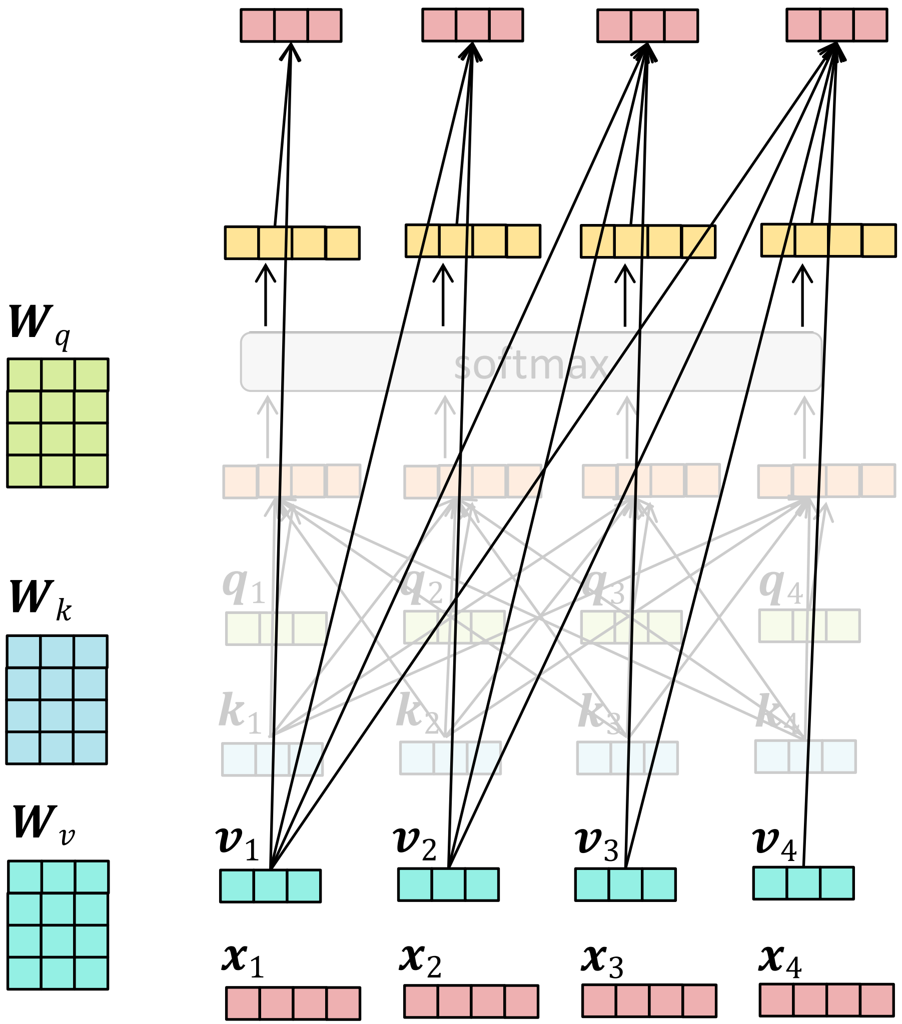
B. $M = \begin{bmatrix} 0 & -\infty & -\infty & -\infty \\ 0 & 0 & -\infty & -\infty \\ 0 & 0 & 0 & -\infty \\ 0 & 0 & 0 & 0 \end{bmatrix}$

C. TOXIC

D. $M = \begin{bmatrix} 0 & -\infty & -\infty & -\infty \\ -\infty & 0 & -\infty & -\infty \\ -\infty & -\infty & 0 & 0 \\ -\infty & -\infty & -\infty & -\infty \end{bmatrix}$

Masked Scaled Dot-Product Attention: Matrix Form

Idea: we can effectively delete or "mask" some of these arrows by selectively setting attention weights to 0



$$X' = V\ \mathrm{softmax}\left(\frac{K^T Q}{\sqrt{d_k}} + M\right)$$

$$A_{mask} = \mathrm{softmax}(S + M)$$

$$M = \begin{bmatrix} 0 & 0 & 0 & 0 \\ -\infty & 0 & 0 & 0 \\ -\infty & -\infty & 0 & 0 \\ -\infty & -\infty & -\infty & 0 \end{bmatrix}$$

Masked Multi-headed Attention: Matrix Form

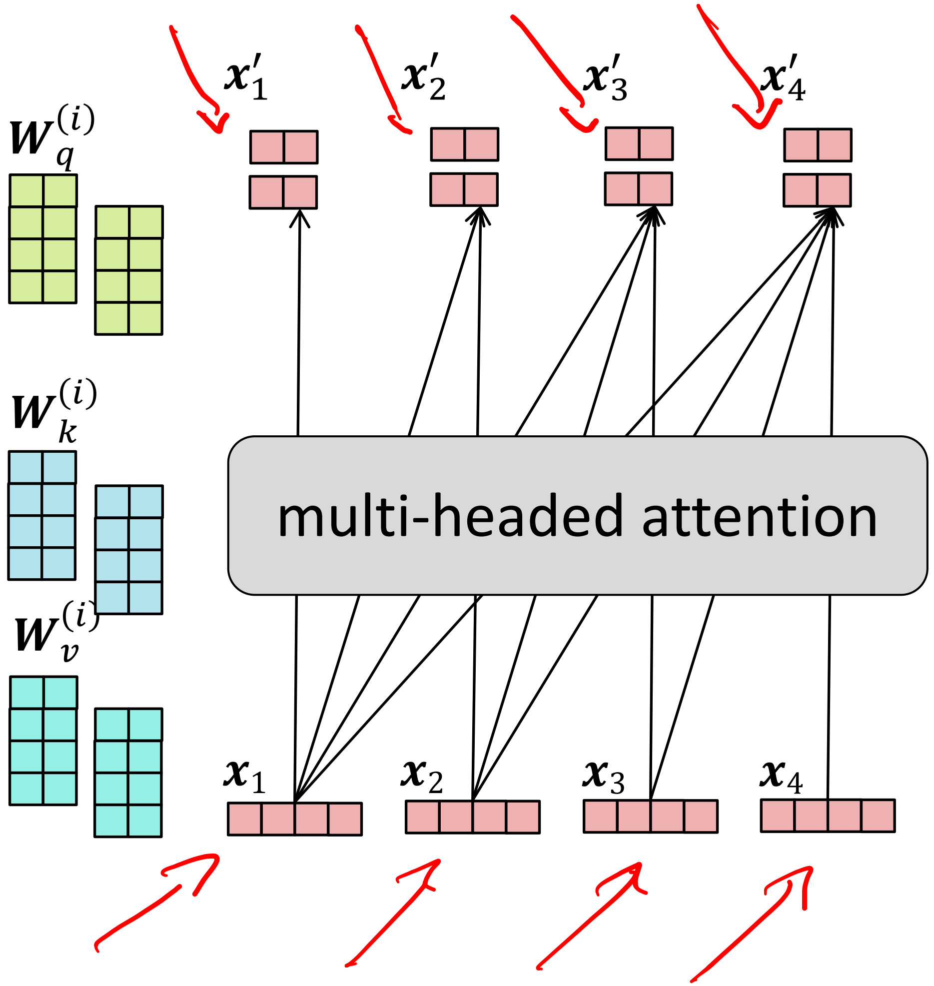
$$X' = \operatorname*{concat}_{i} \left\{ \overbrace{V^{(i)} \,\text{softmax}\Big(\frac{{K^{(i)}}^T Q^{(i)}}{\sqrt{d_k}}}^{\text{representation } x' \text{ for the i'th attention head}} + M \Big) \right\}$$

where

$$Q^{(i)} = {\boldsymbol{W}_q^{(i)}}^T X^T$$

$$K^{(i)} = {\boldsymbol{W}_k^{(i)}}^T X^T$$

$$V^{(i)} = {\boldsymbol{W}_v^{(i)}}^T X^T$$



Practical Considerations

1. Where on earth do tokens come from?
   - Example: “Henry is giving a lecture on transformers”
   - Word-based tokenization:

   [“henry”, “is”, “giving”, “a”, “lecture”, “on”, “transformers”]

2. How can we handle variable-length sequences?

Practical Considerations

1. Where on earth do tokens come from?
   - Example: “Henry is givin’ a lectrue on transformers”
   - Word-based tokenization:

     [“henry”, “is”, ???, “a”, ???, “on”, “transformers”]
   - Can have difficulty trading off between vocabulary size and computational tractability
   - Similar words e.g., “transformers” and “transformer” can get mapped to completely disparate representations
   - Typos will typically be out-of-vocabulary (OOV)
2. How can we handle variable-length sequences?

Practical Considerations

1. Where on earth do tokens come from?
   - Example: “Henry is givin’ a lectrue on transformers”
   - Character-based tokenization:

   [“h”, “e”, “n”, “r”, “y”, “i”, “s”, “g”, “i”, “v”, “i”, “n”, “ ’ ”, …]

   - Much smaller vocabularies but a lot of semantic meaning is lost…
   - Sequences will be much longer than word-based tokenization, potentially causing computational issues
   - Can do well on logographic languages e.g., Kanji 漢字

2. How can we handle variable-length sequences?

Practical Considerations

1. Where on earth do tokens come from?
    - Example: “Henry is givin’ a lectrue on transformers”
    - Subword tokenization:

      u

      [“henry”, “is”, “giv”, “##in”, “ ‘ ”, “a”, “lect” “##re”, “on”, “transform”, “##ers”]
    - Split long or rare words into smaller, semantically meaningful components or *subwords*
2. How can we handle variable-length sequences?
    - Artificially make all sequences the same length by
        - Padding: adding special *pad tokens* to short sequences
        - Truncating: using only the first few tokens for long sequences

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

1. Where on earth do tokens come from?
   - Example: “Henry is givin’ a lectrue on transformers”
   - Subword tokenization:

     [“henry”, “is”, “giv”, “##in”, “ ‘ ”, “a”, “lect” “##re”, “on”, “transform”, “##ers”]

   - Split long or rare words into smaller, semantically meaningful components or *subwords*

2. How can we handle variable-length sequences?
   - Artificially make all sequences the same length by
     - Padding: adding special *pad tokens* to short sequences
     - Truncating: using only the first few tokens for long sequences

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!~~ just the beginning!

- 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: $t \leftarrow t + 1$

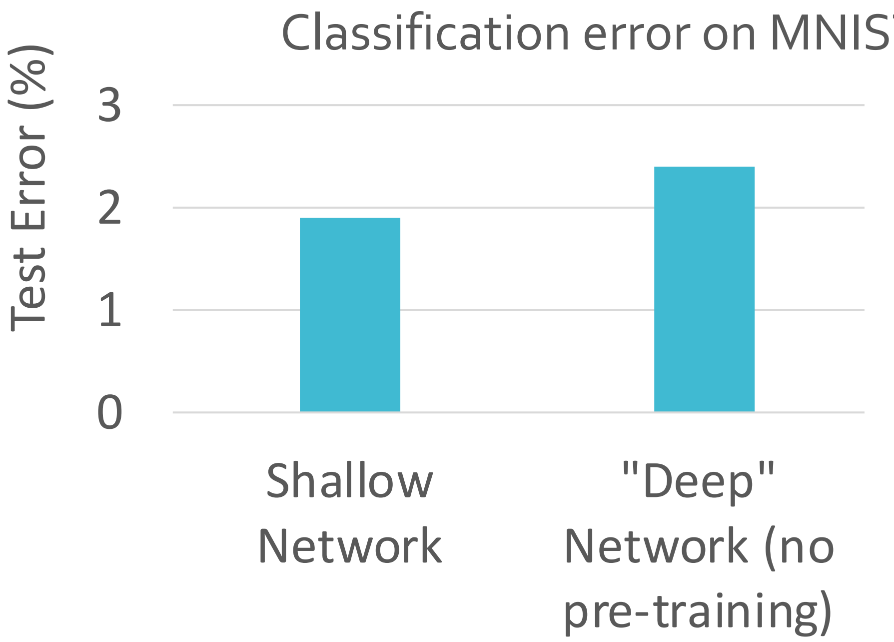
- Output: $\boldsymbol{\theta}^{(t)}$

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

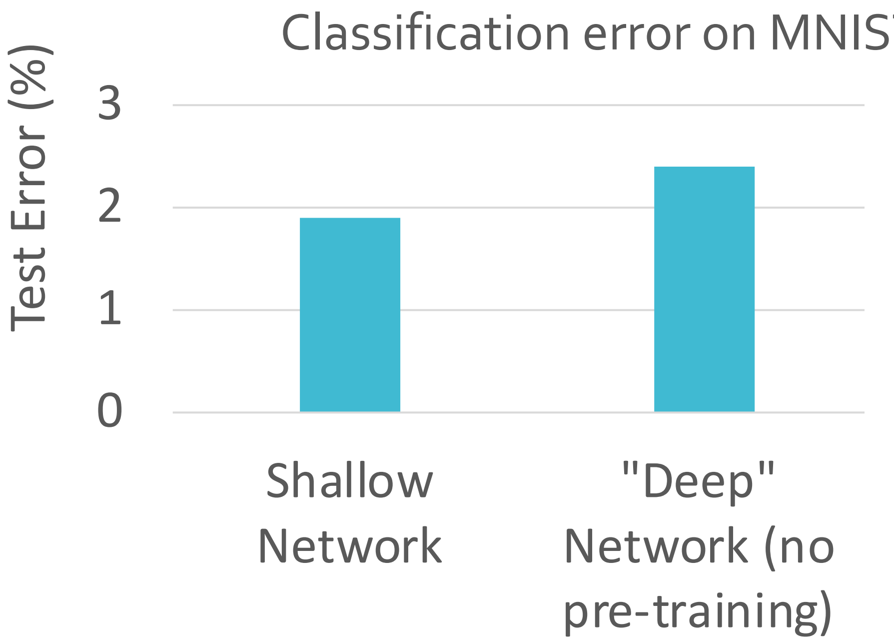


- "gradient-based optimization starting from random initialization appears to often get stuck in poor solutions for such deep networks."

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

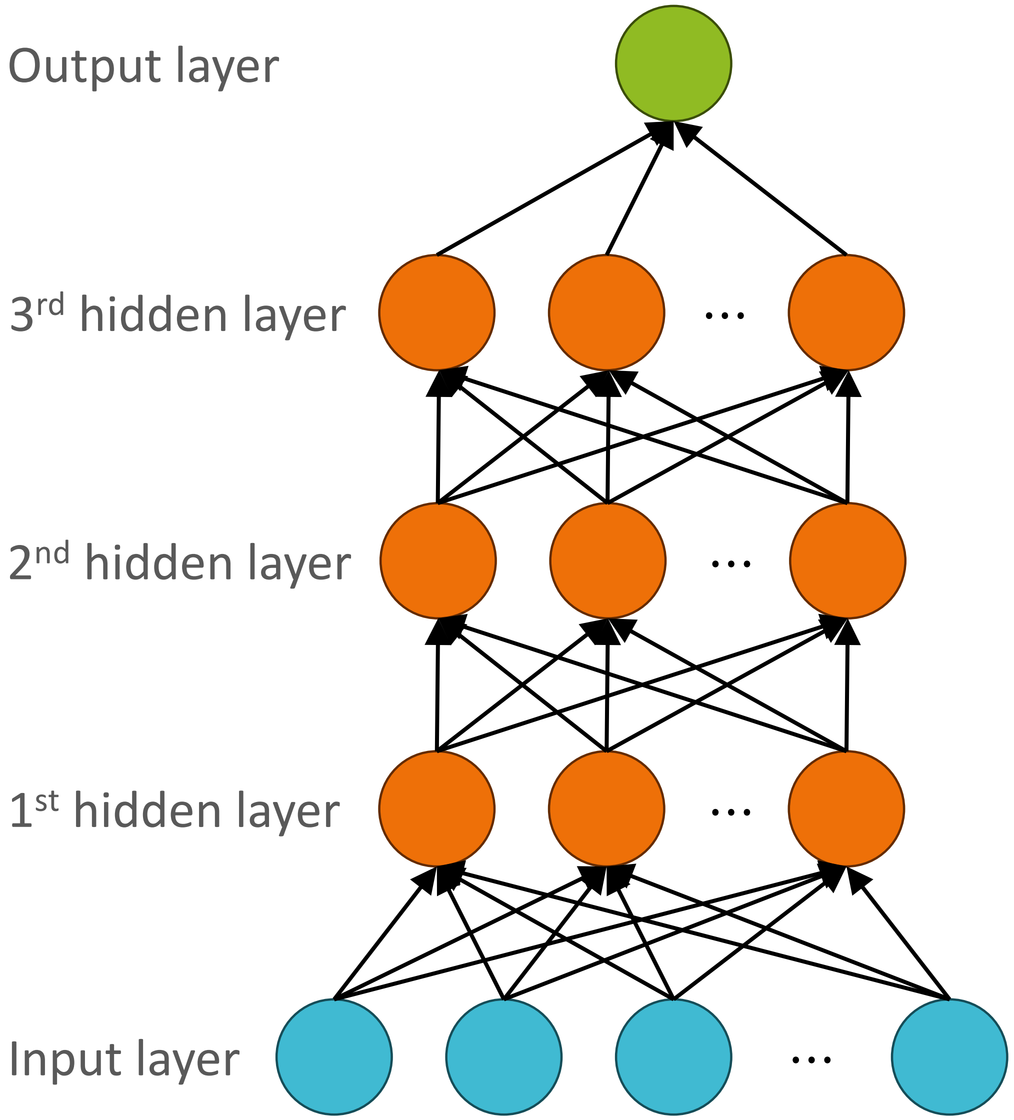


- Idea: if shallow networks are easier to train, let's just decompose our deep network into a series of shallow networks!

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

Pre-training (Bengio et al., 2006)

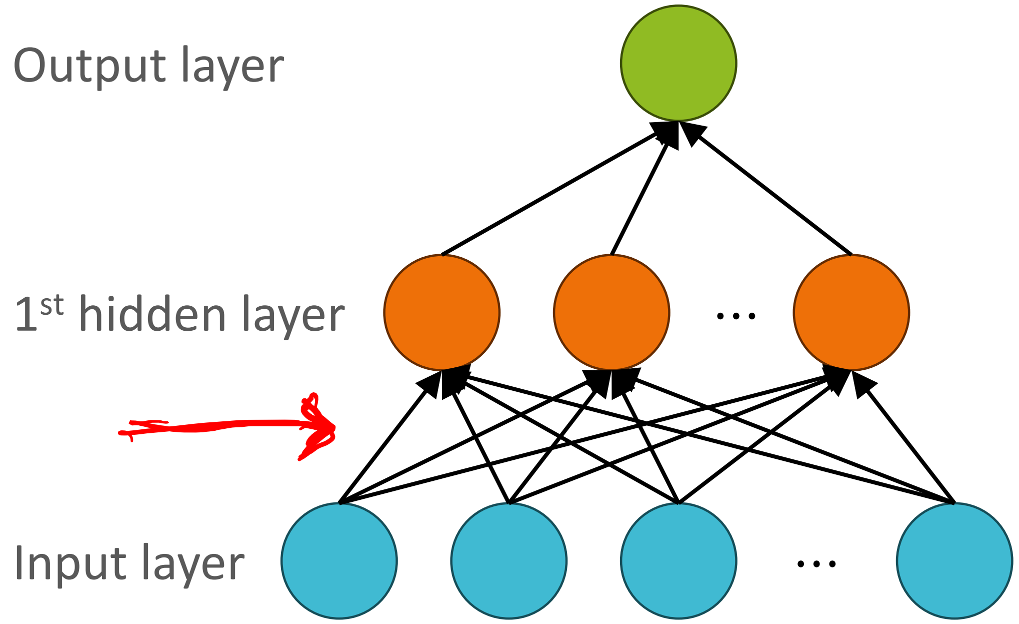
- Train each layer of the network iteratively using the training dataset
- Start at the input layer and move towards the output layer
- Once a layer has been trained, fix its weights and use those to train subsequent layers



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

Pre-training (Bengio et al., 2006)

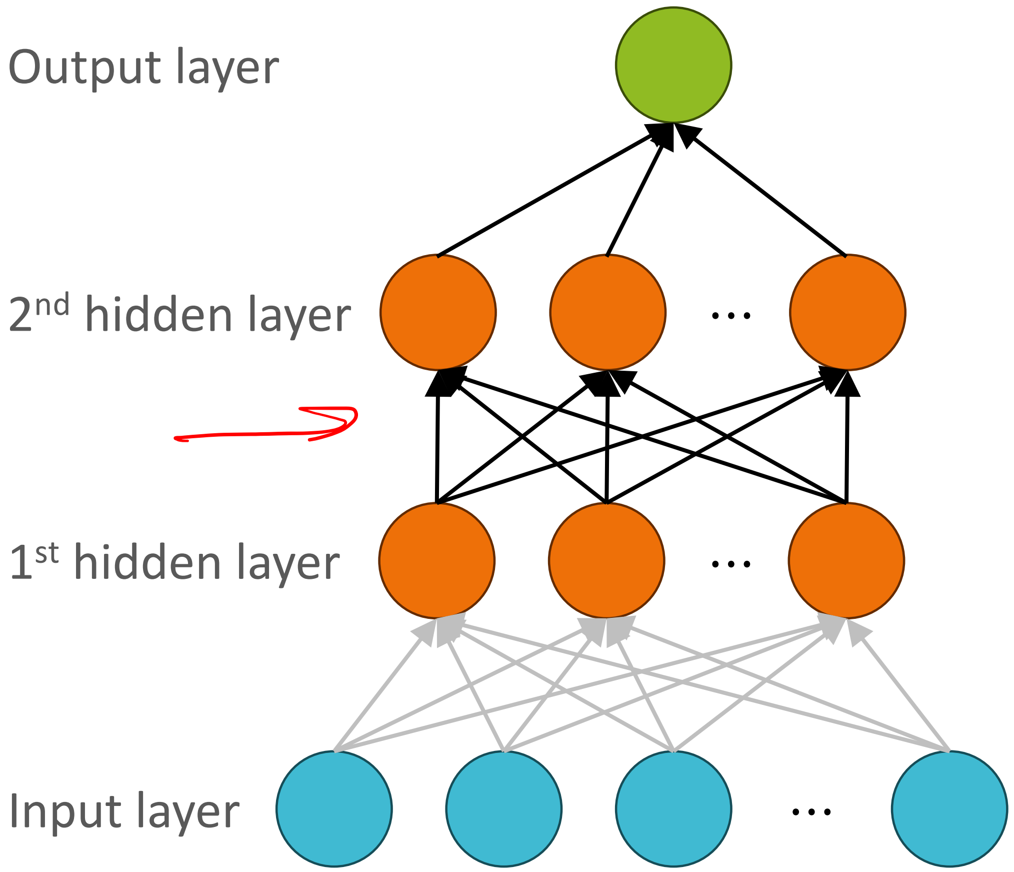
- Train each layer of the network iteratively using the training dataset
- Start at the input layer and move towards the output layer
- Once a layer has been trained, fix its weights and use those to train subsequent layers



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

Pre-training (Bengio et al., 2006)

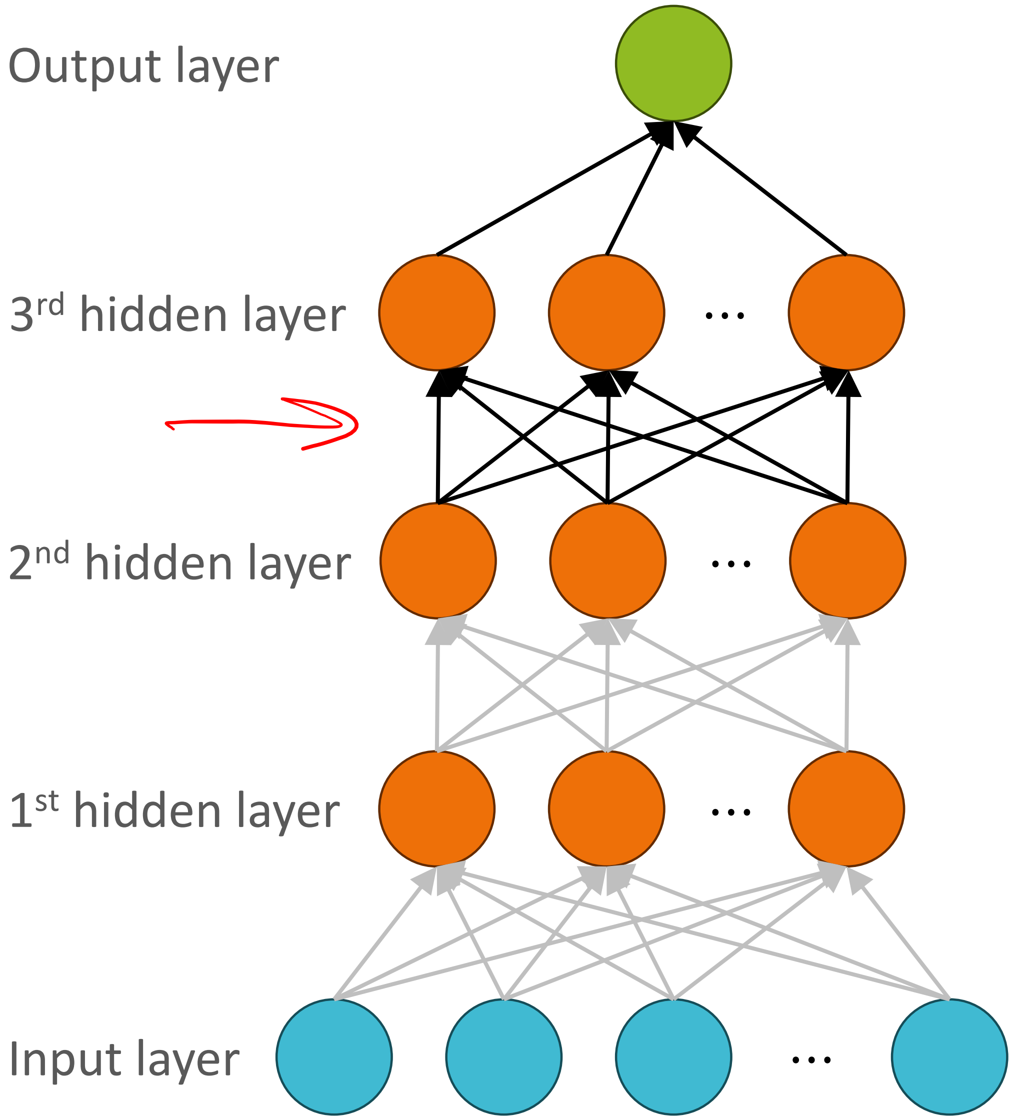
- Train each layer of the network iteratively using the training dataset
- Start at the input layer and move towards the output layer
- Once a layer has been trained, fix its weights and use those to train subsequent layers



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

Pre-training (Bengio et al., 2006)

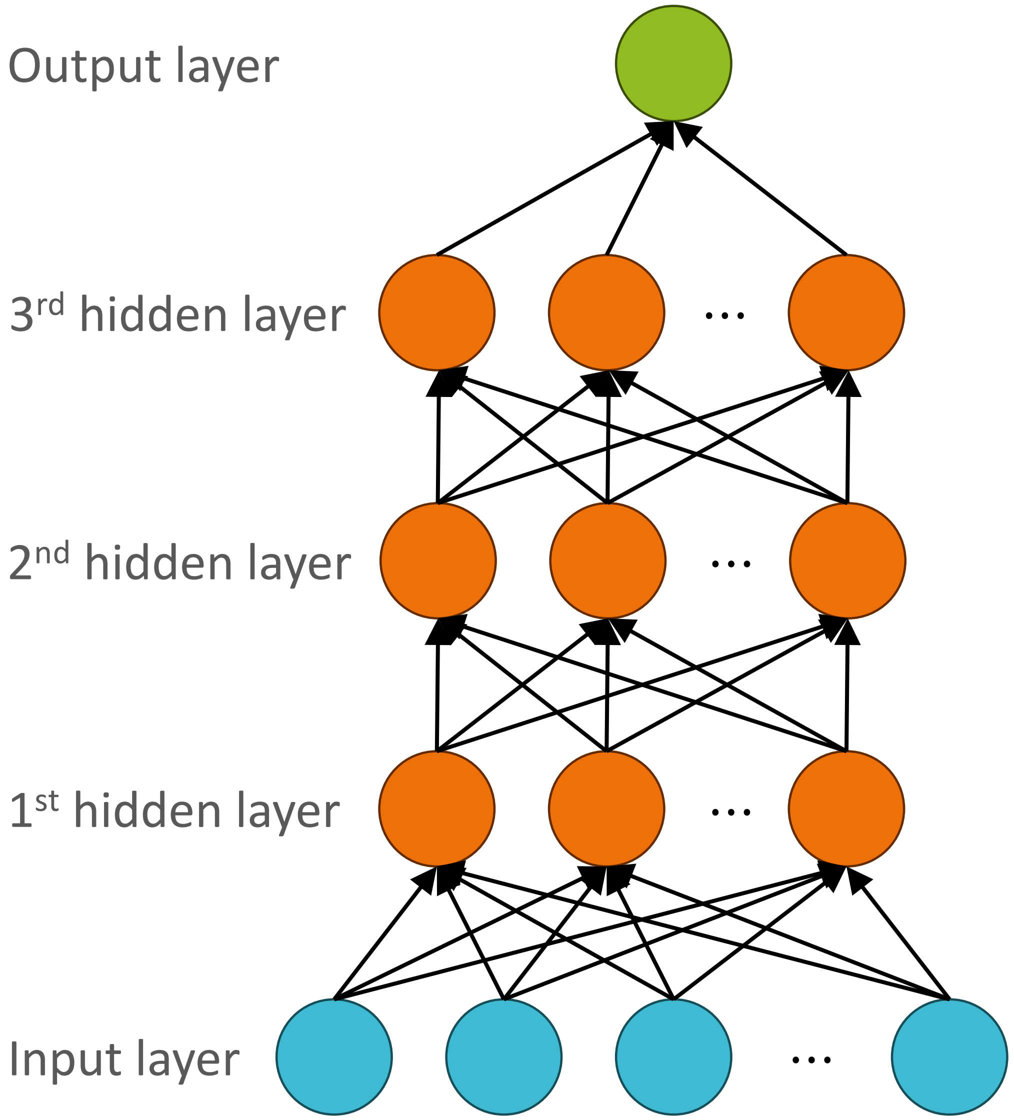
- Train each layer of the network iteratively using the training dataset
- Start at the input layer and move towards the output layer
- Once a layer has been trained, fix its weights and use those to train subsequent layers



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

Fine-tuning (Bengio et al., 2006)

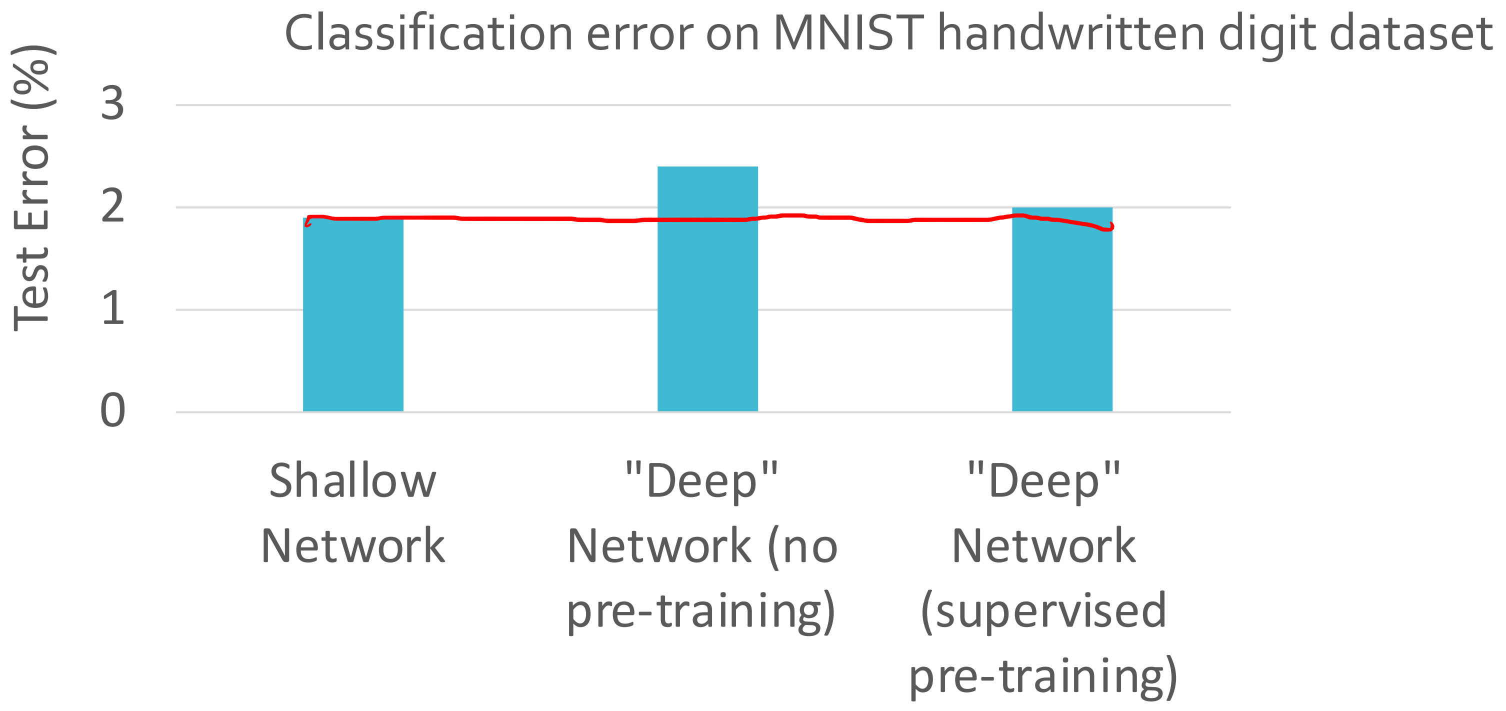
- Train each layer of the network iteratively using the training dataset
- Use the pre-trained weights as an initialization and *fine-tune* the entire network e.g., via SGD with the training dataset



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

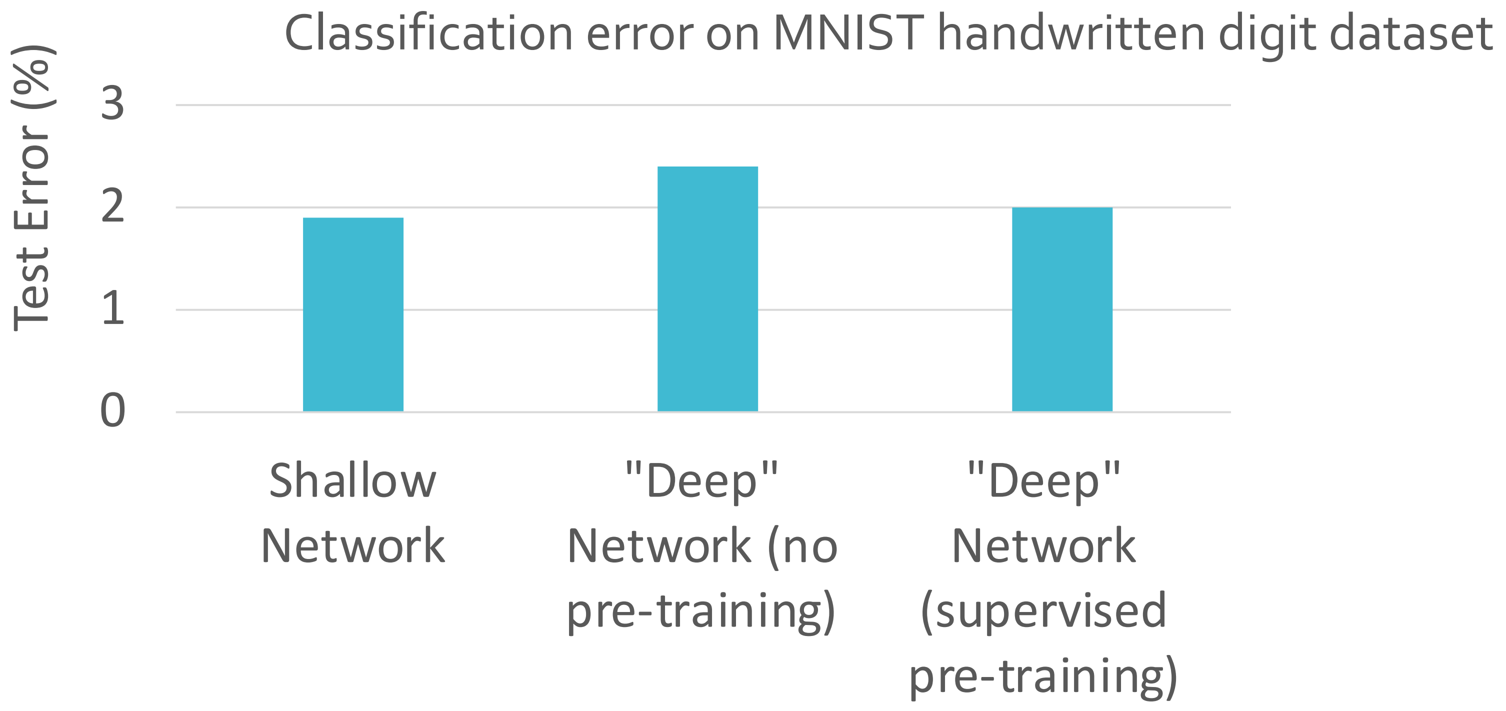
Supervised Pre-training (Bengio et al., 2006)

- 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



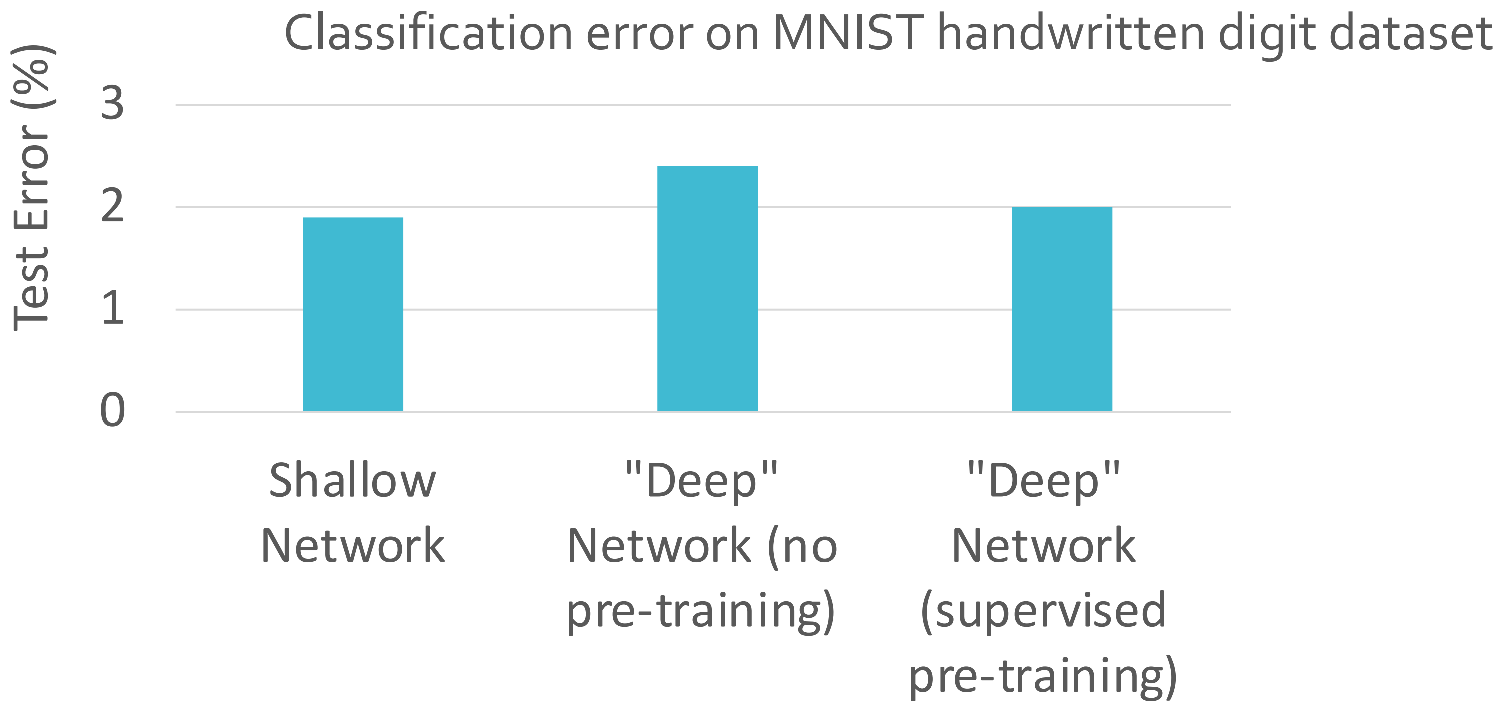
Supervised Pre-training (Bengio et al., 2006)

- 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



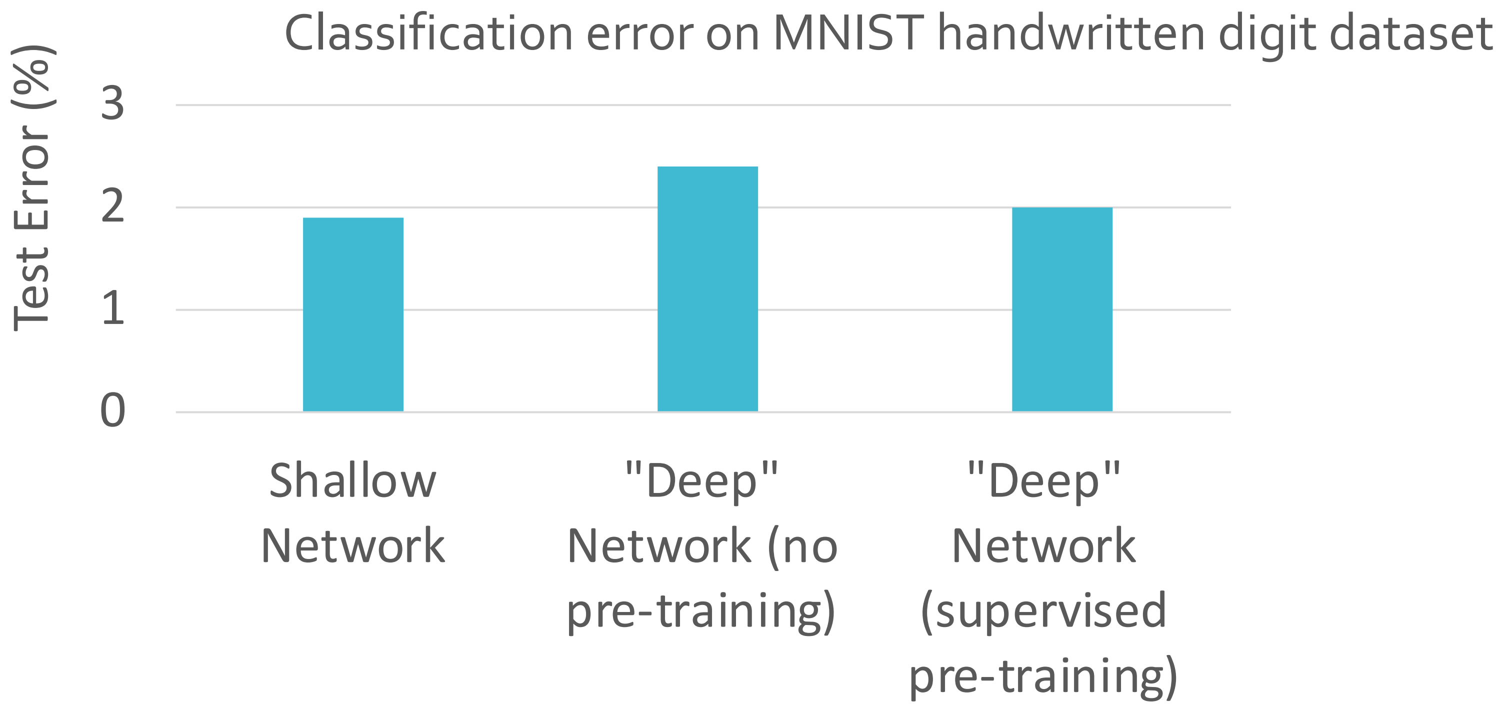
Is this the only thing we could do with the training data?

- 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)

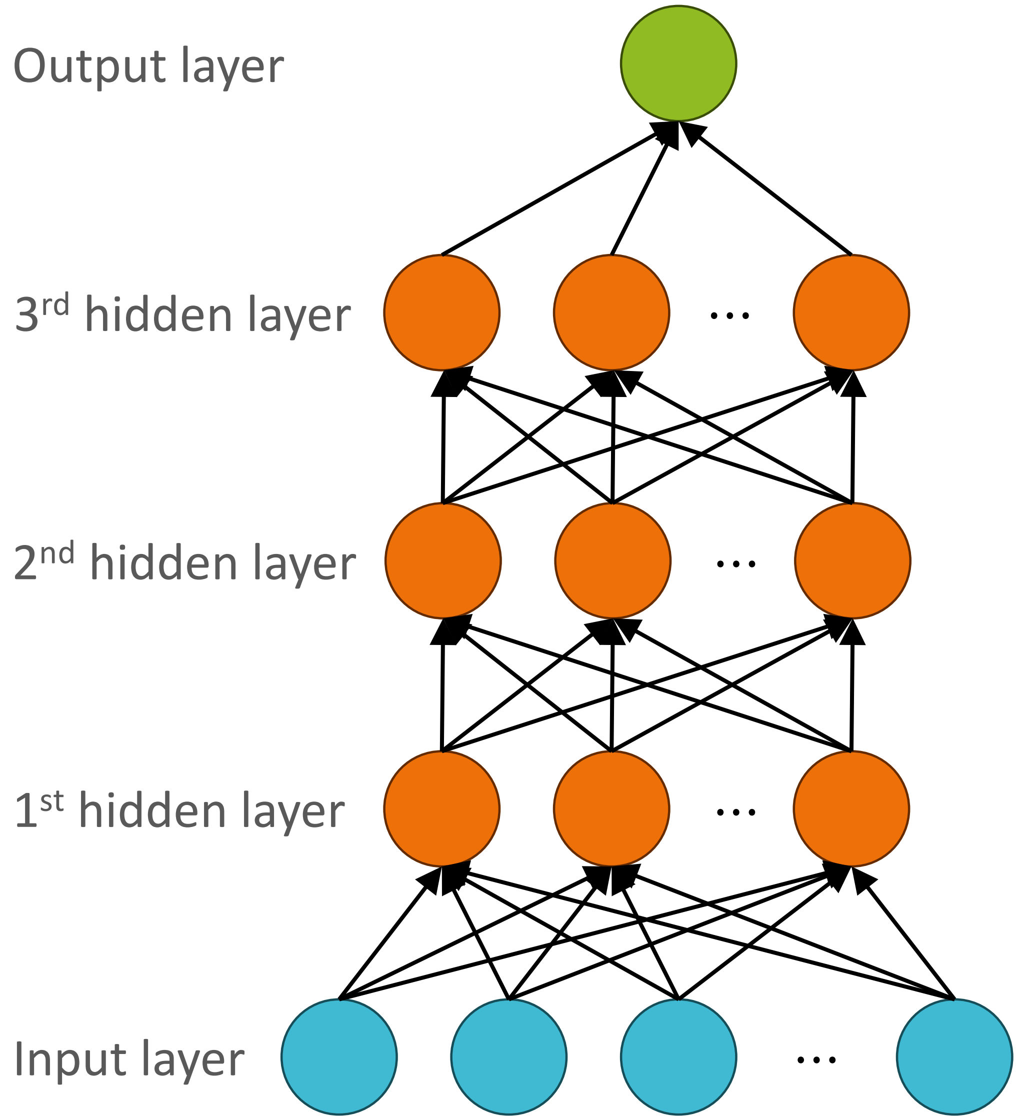
- Train each layer of the network iteratively using the training dataset *to learn useful representations*
- Idea: a good representation is one preserves a lot of information and could be used to recreate the inputs



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

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

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



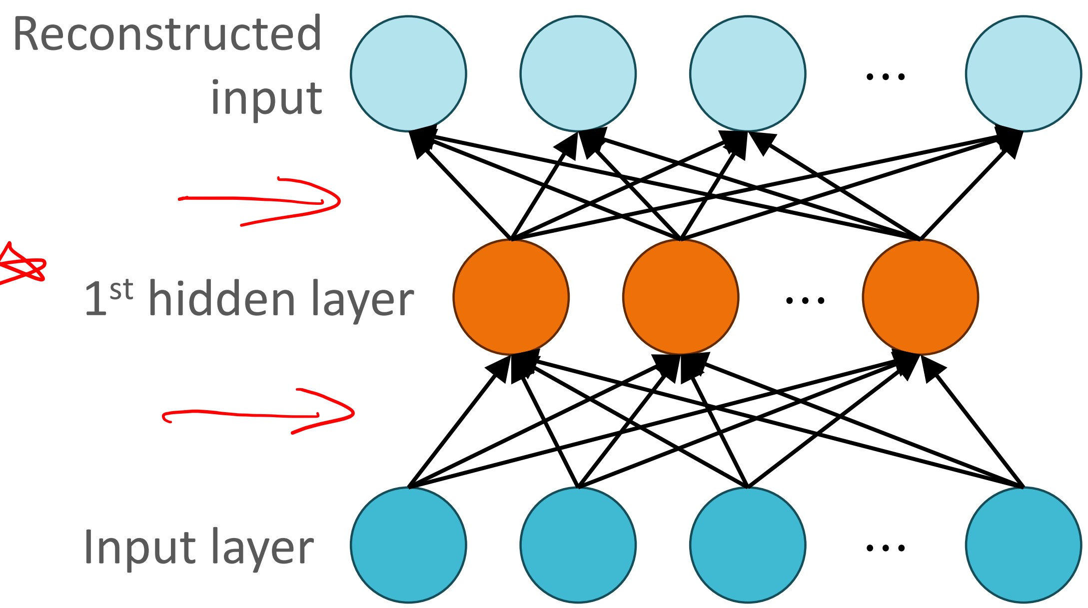
Source: https://www.cs.toronto.edu/~larocheh/publications/dbn_supervised_tr1282.pdf

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

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

$\min_h \quad \|x - h(x)\|_2$

- This architecture/ objective defines an *autoencoder*

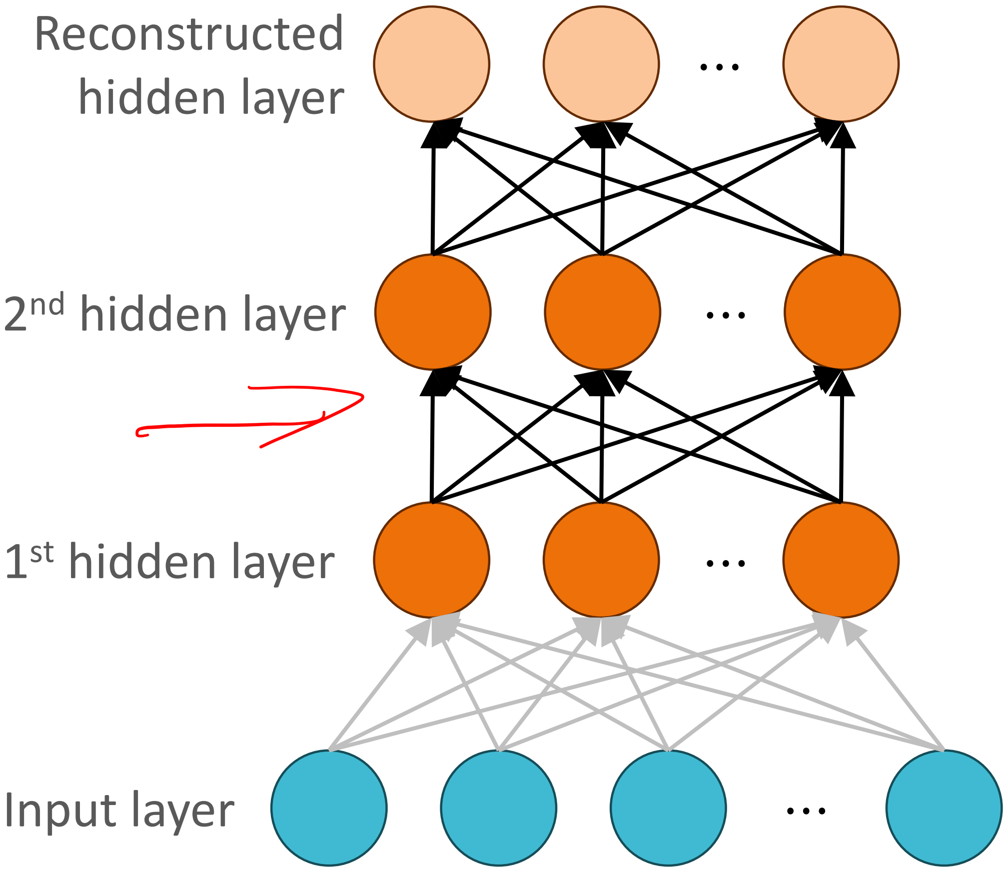


Unsupervised Pre-training (Bengio et al., 2006)

- 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*



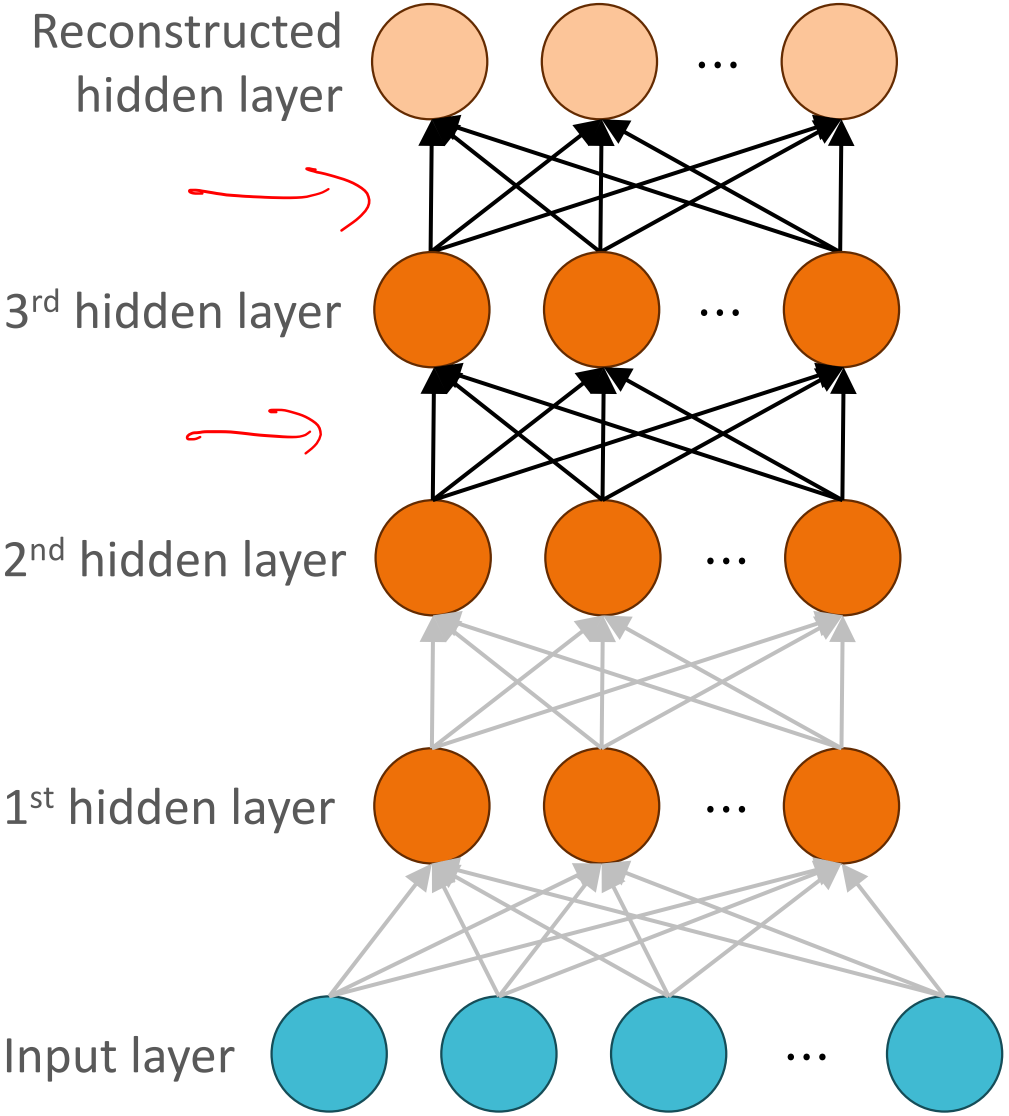
Source: https://www.cs.toronto.edu/~larocheh/publications/dbn_supervised_tr1282.pdf

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

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

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

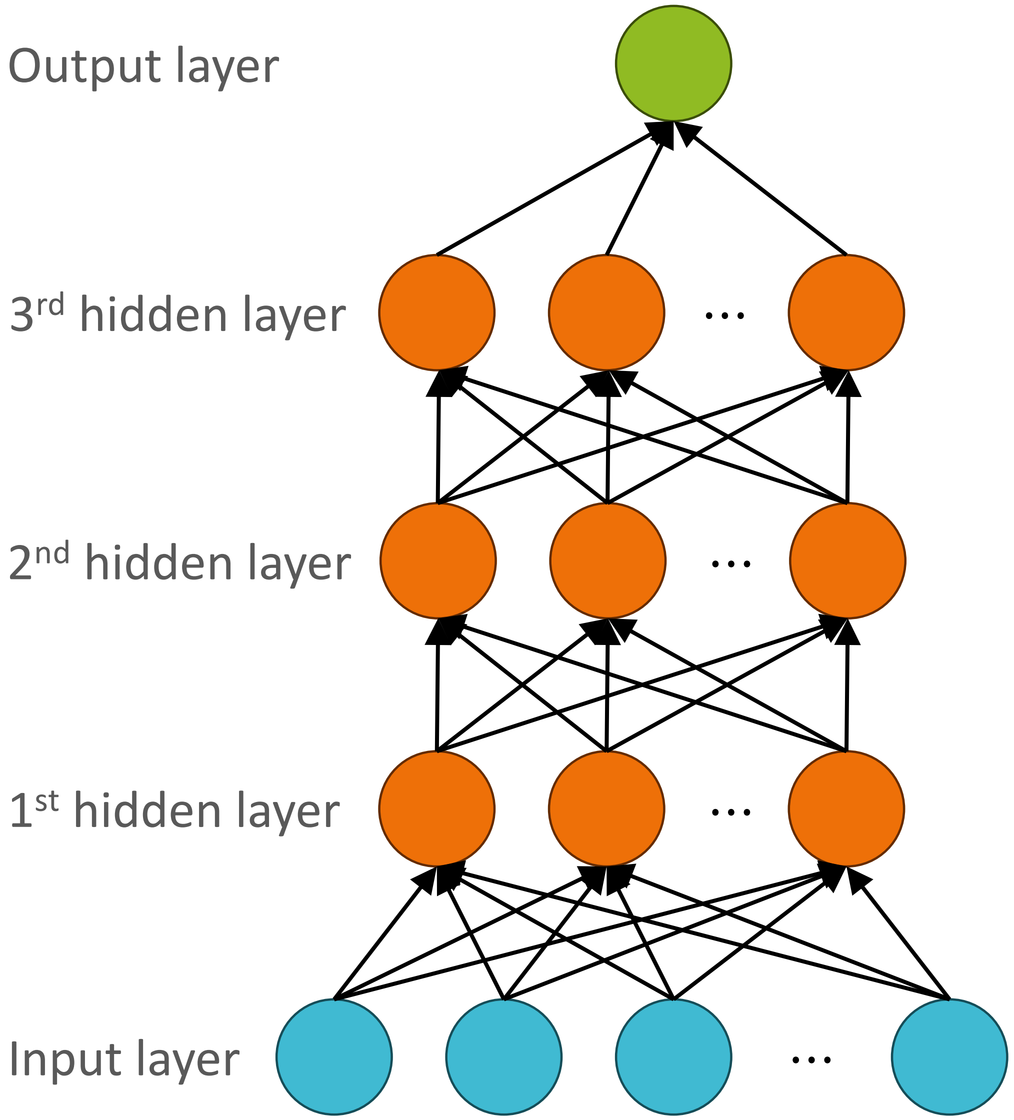
- This architecture/ objective defines an *autoencoder*



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

Fine-tuning (Bengio et al., 2006)

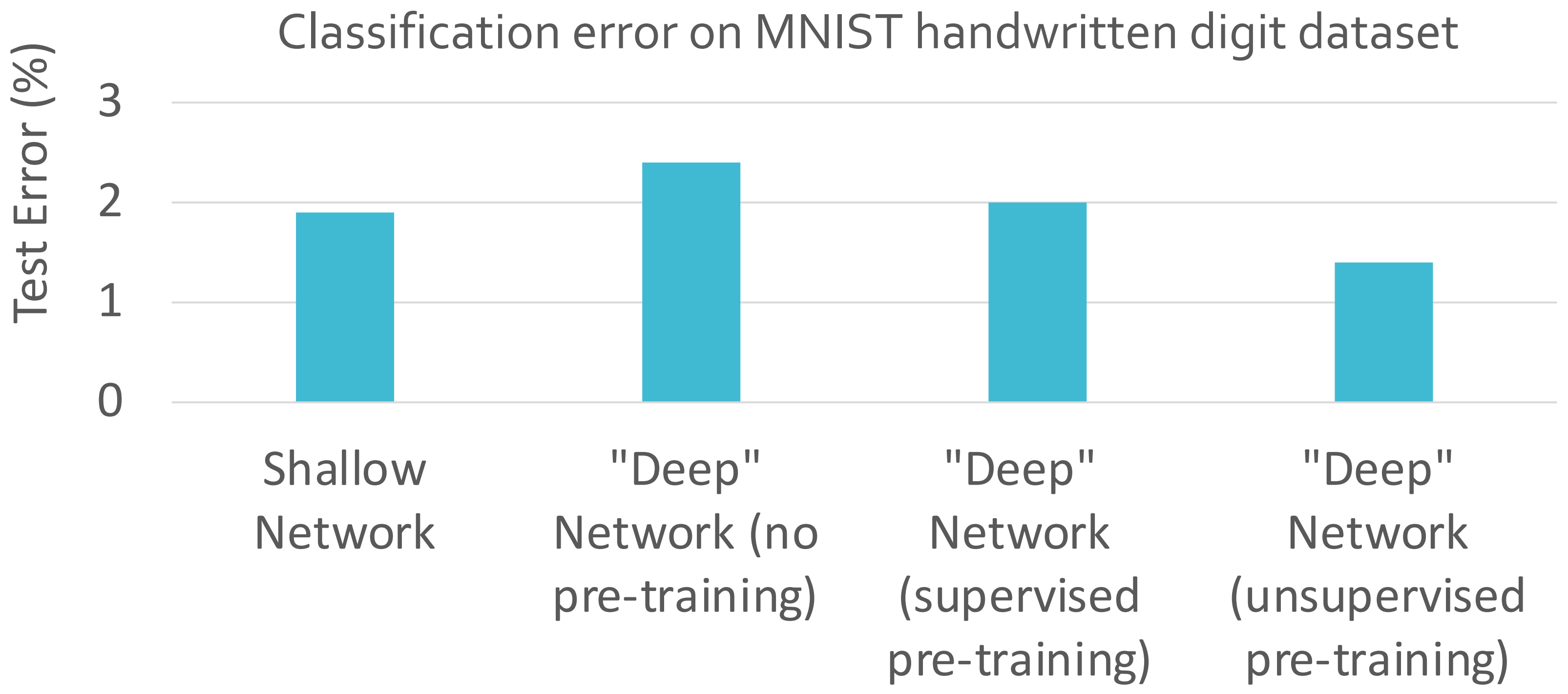
- Train each layer of the network iteratively using the training dataset by minimizing the *reconstruction error*
  $$\|\boldsymbol{x} - h(\boldsymbol{x})\|_2$$
- When fine-tuning, we're effectively swapping out the last layer and fitting all the weights to the training dataset



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

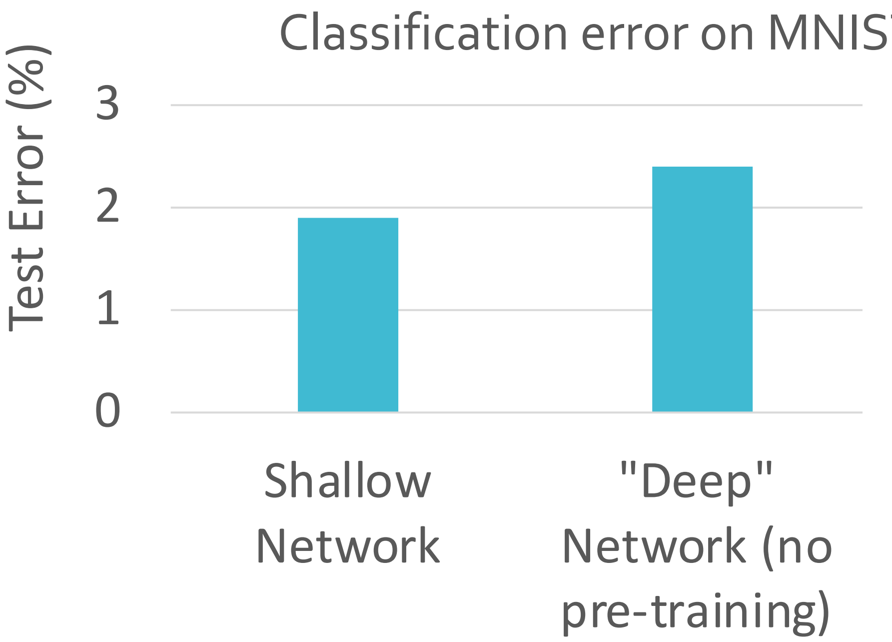
Unsupervised Pre-training (Bengio et al., 2006)

- 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

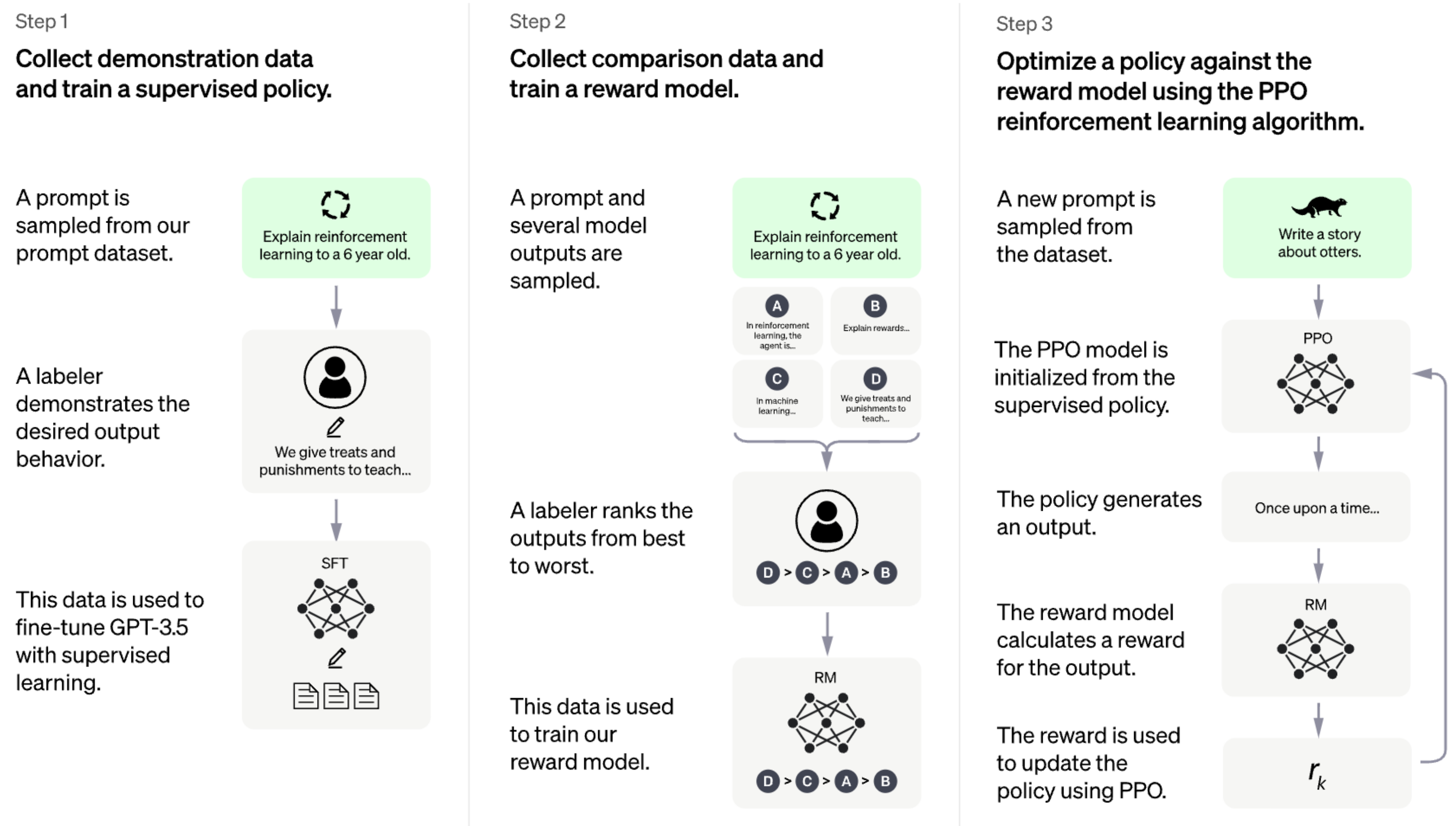
Another dose of Reality

- 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!

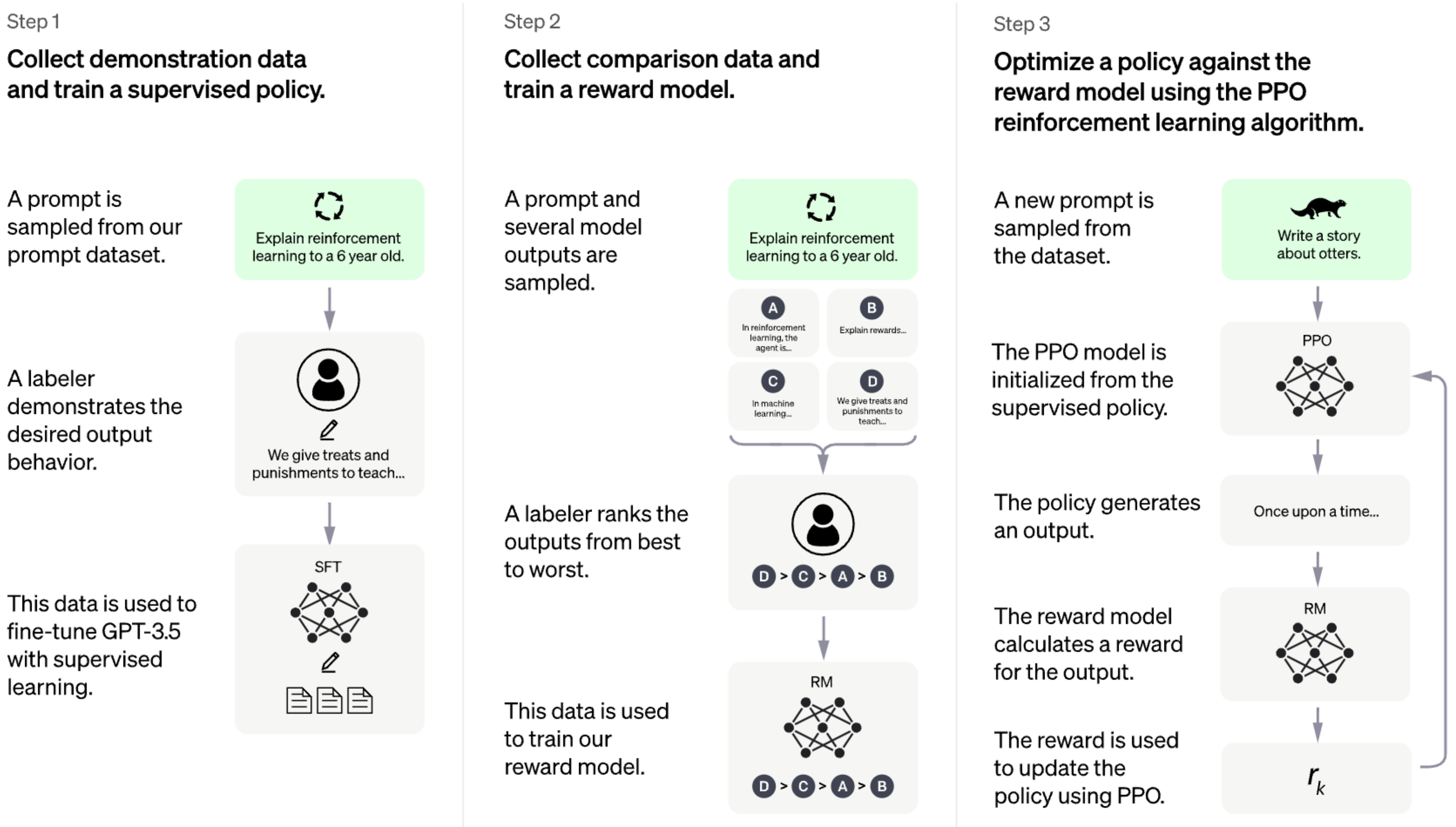
Reinforcement Learning from Human Feedback (RLHF)



- RLHF is a special form of fine-tuning, used to fine-tune GPT-3.5 into ChatGPT

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

Okay, so this is great if our problem is subjective, but again, what can we do for objective tasks where training data is scarce?



- RLHF is a special form of fine-tuning, used to fine-tune GPT-3.5 into ChatGPT

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

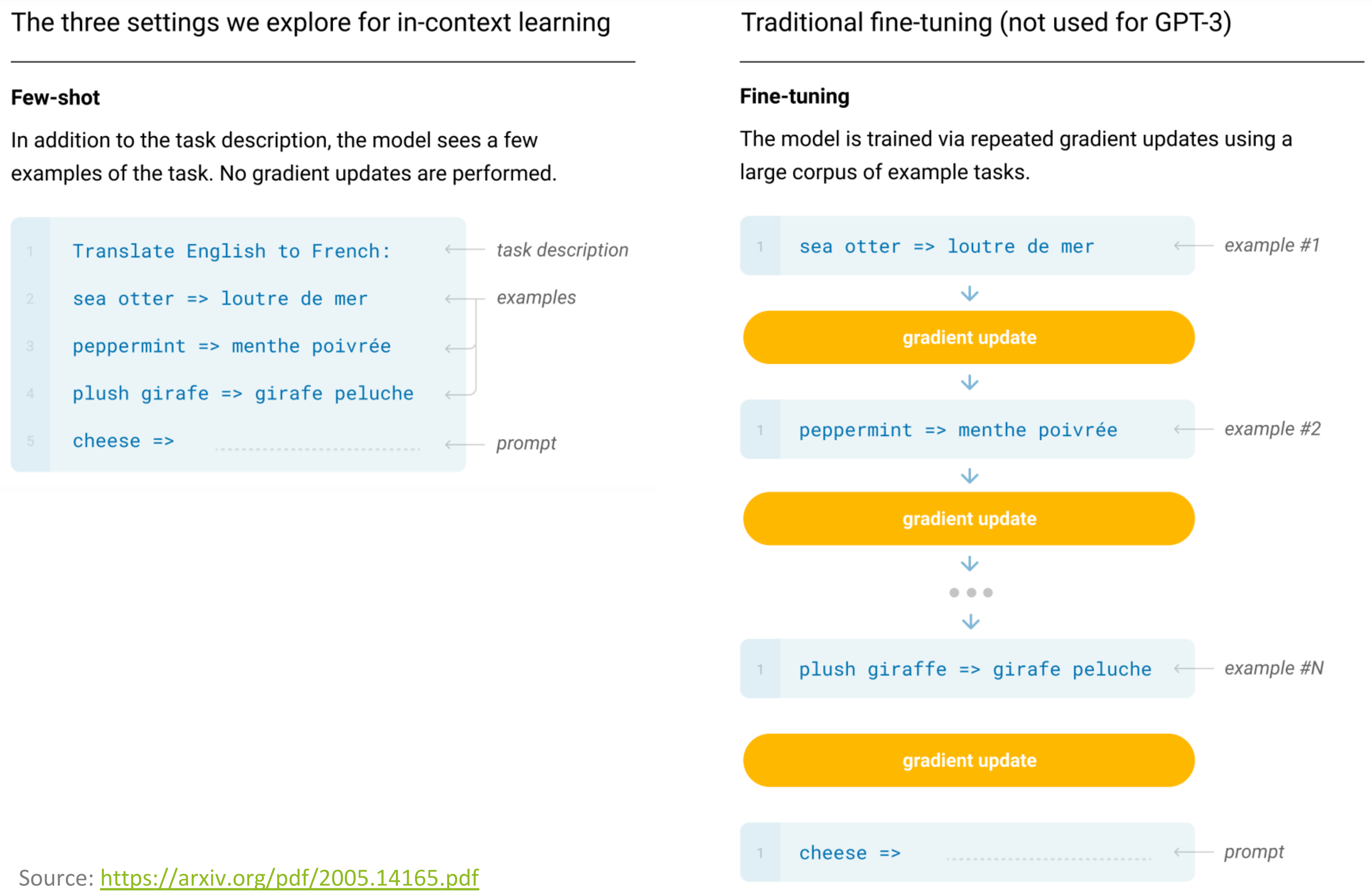
In-context Learning

- 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

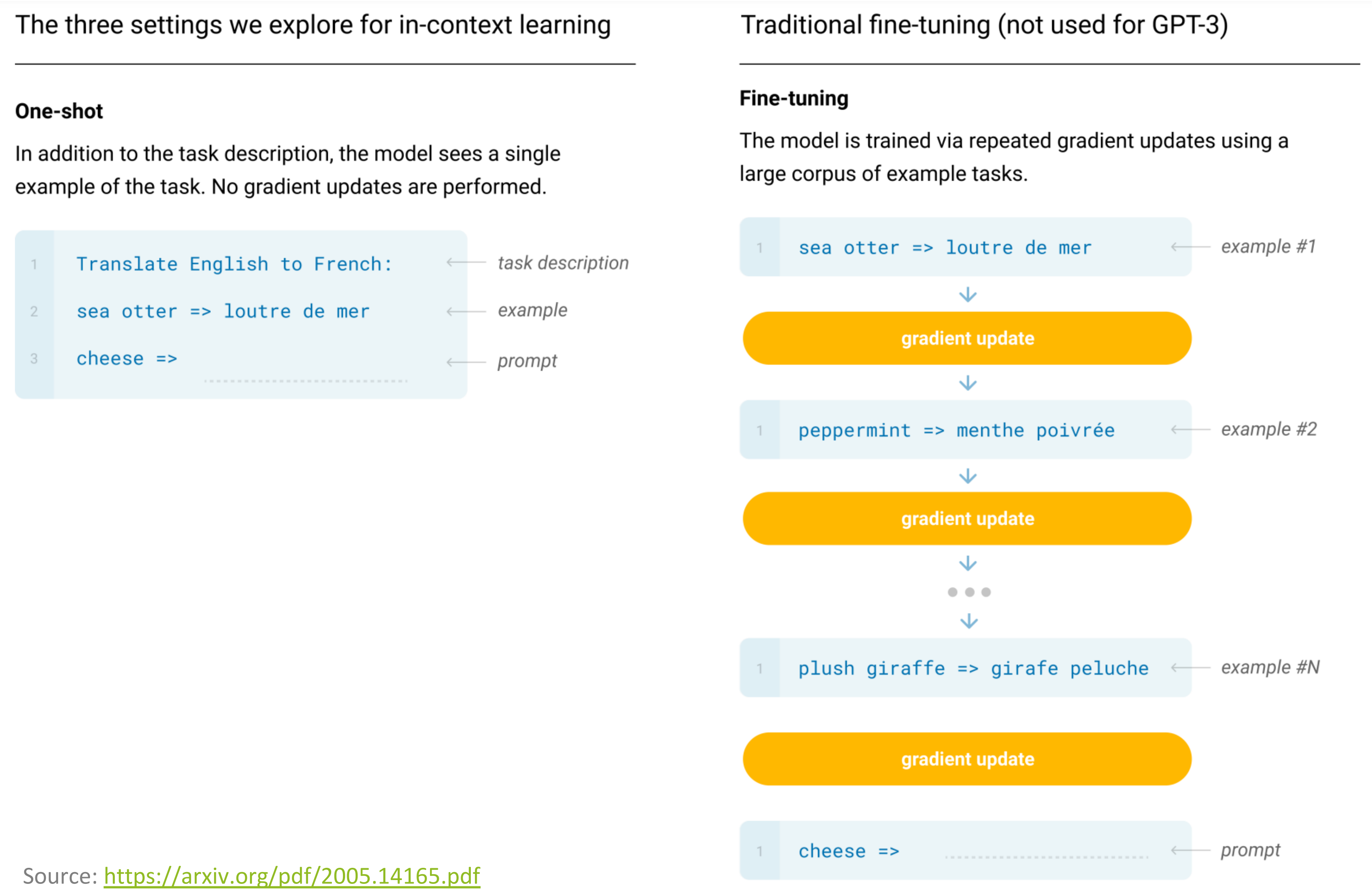
- Idea: leverage the LLM's context window by passing a few examples to the model as input, *without performing any updates to the parameters*



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

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

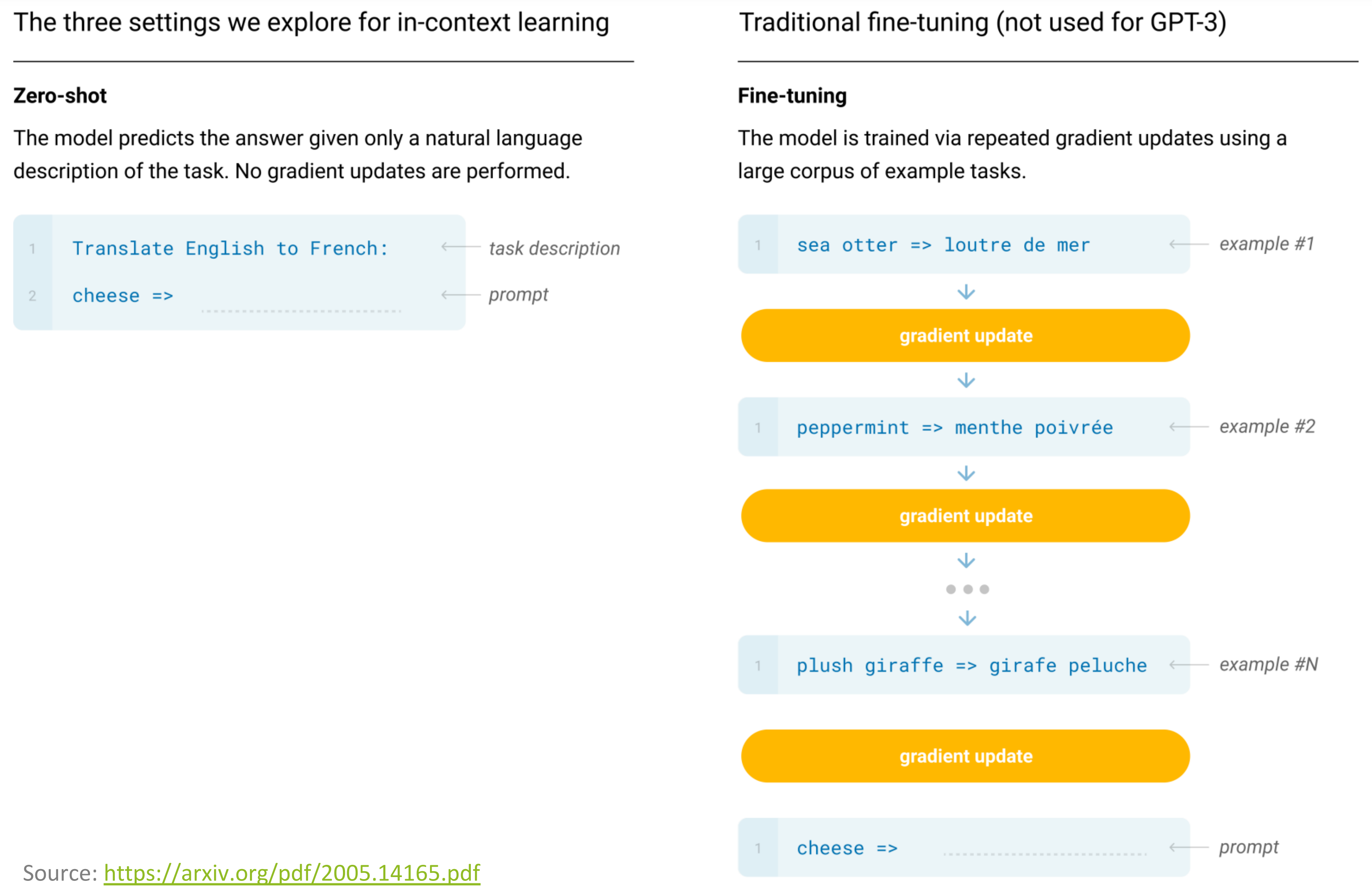
- Idea: leverage the LLM's context window by passing ~~a~~ ~~few~~ one examples to the model as input, *without performing any updates to the parameters*



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

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

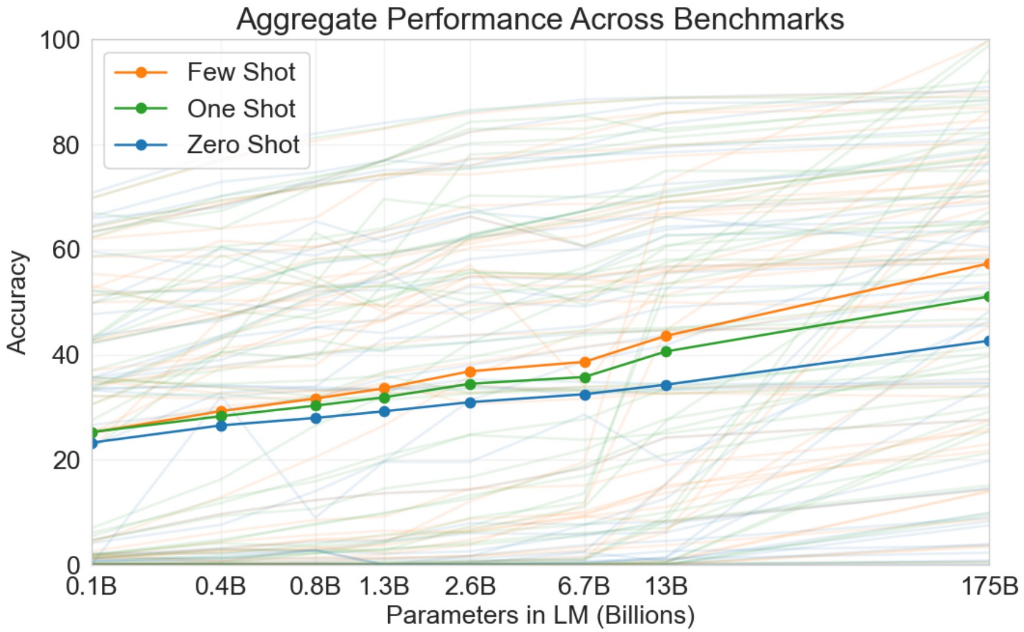
- Idea: leverage the LLM's context window by passing ~~a few one~~ zero(!) examples to the model as input, *without performing any updates to the parameters*



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

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

- 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 just the beginning!

- 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. *Pre-train* 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 *fine-tuning* loss
   $$\nabla J^{(B)}\left(\boldsymbol{\theta}^{(t)}\right)$$
   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: $\boldsymbol{\theta}^{(t)}$

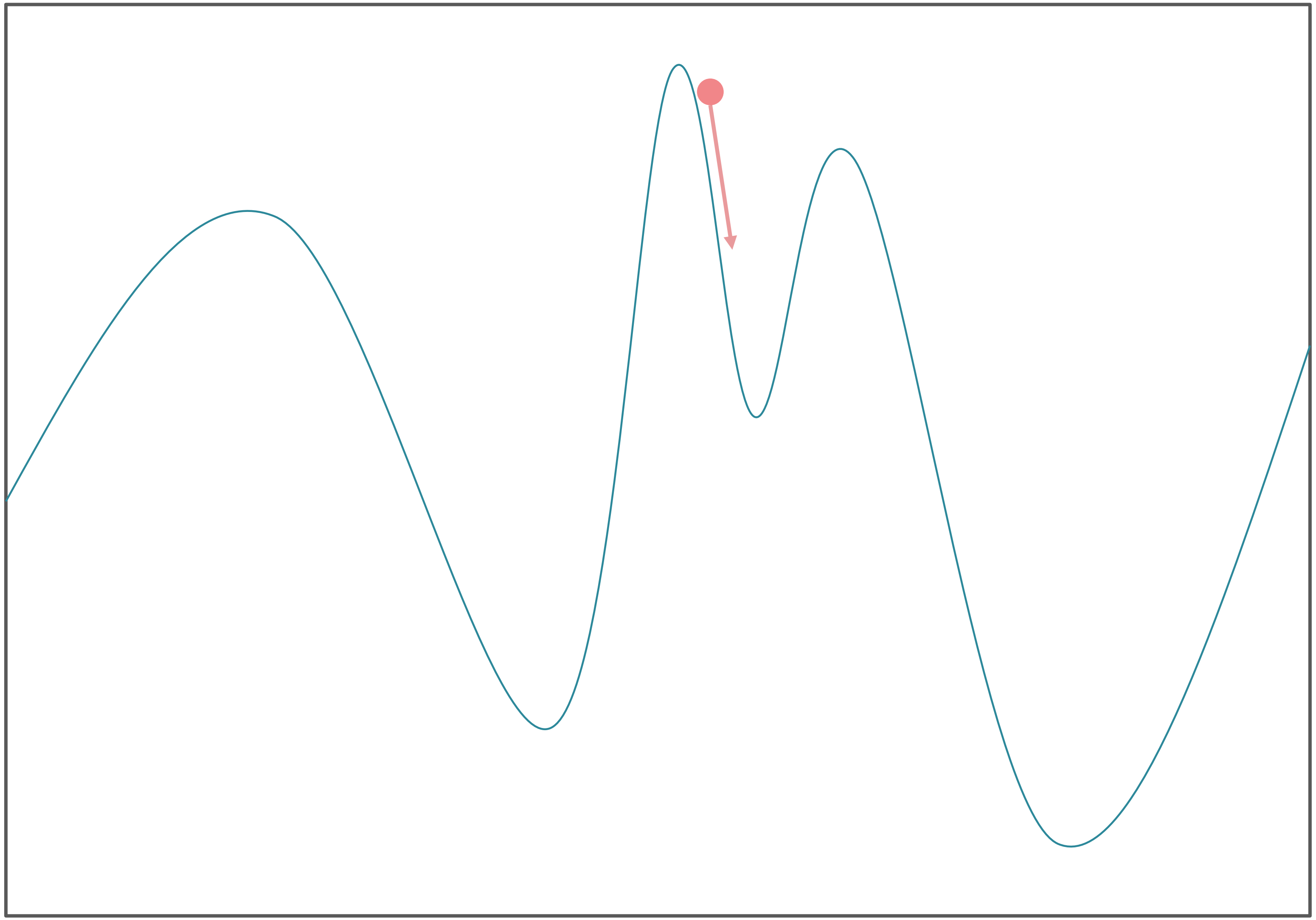
Mini-batch Stochastic Gradient Descent with Momentum

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

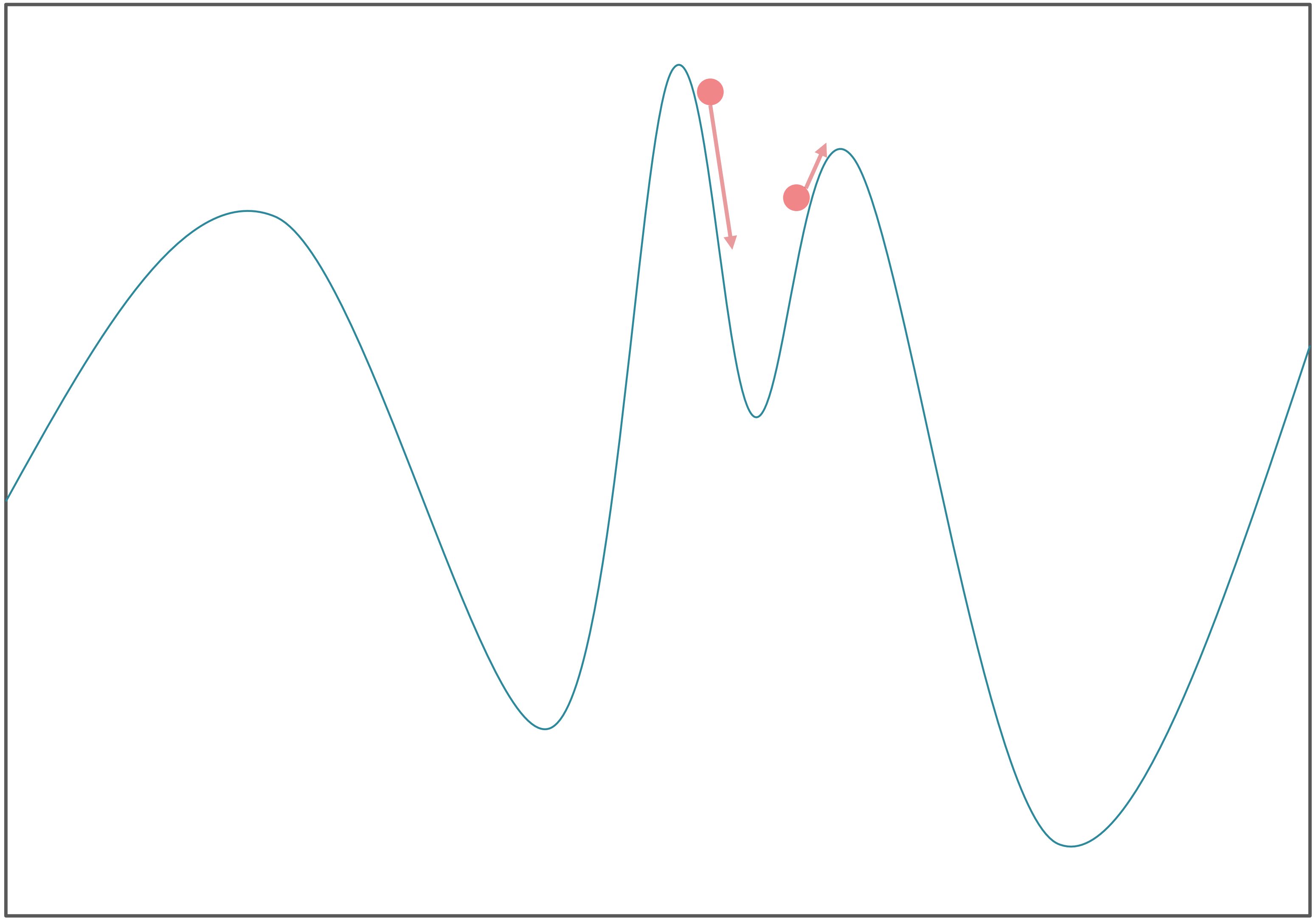
1. *Pre-train* the parameters $\boldsymbol{\theta}^{(0)}$ and set $t = 0$, $G_{-1} = 0 \odot \boldsymbol{\theta}^{(0)}$
2. While TERMINATION CRITERION is not satisfied
   a. Randomly sample B data points from $\mathcal{D}$, $\left\{\left(\boldsymbol{x}^{(b)}, y^{(b)}\right)\right\}_{b=1}^{B}$
   b. Compute the gradient of the *fine-tuning* loss
   $$G_t = \nabla J^{(B)}\left(\boldsymbol{\theta}^{(t)}\right)$$
   c. Update $\boldsymbol{\theta}$: $\boldsymbol{\theta}^{(t+1)} \leftarrow \boldsymbol{\theta}^{(t)} - \gamma\left(\beta G_{t-1} + G_t\right)$
   d. Increment t: $t \leftarrow t + 1$

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

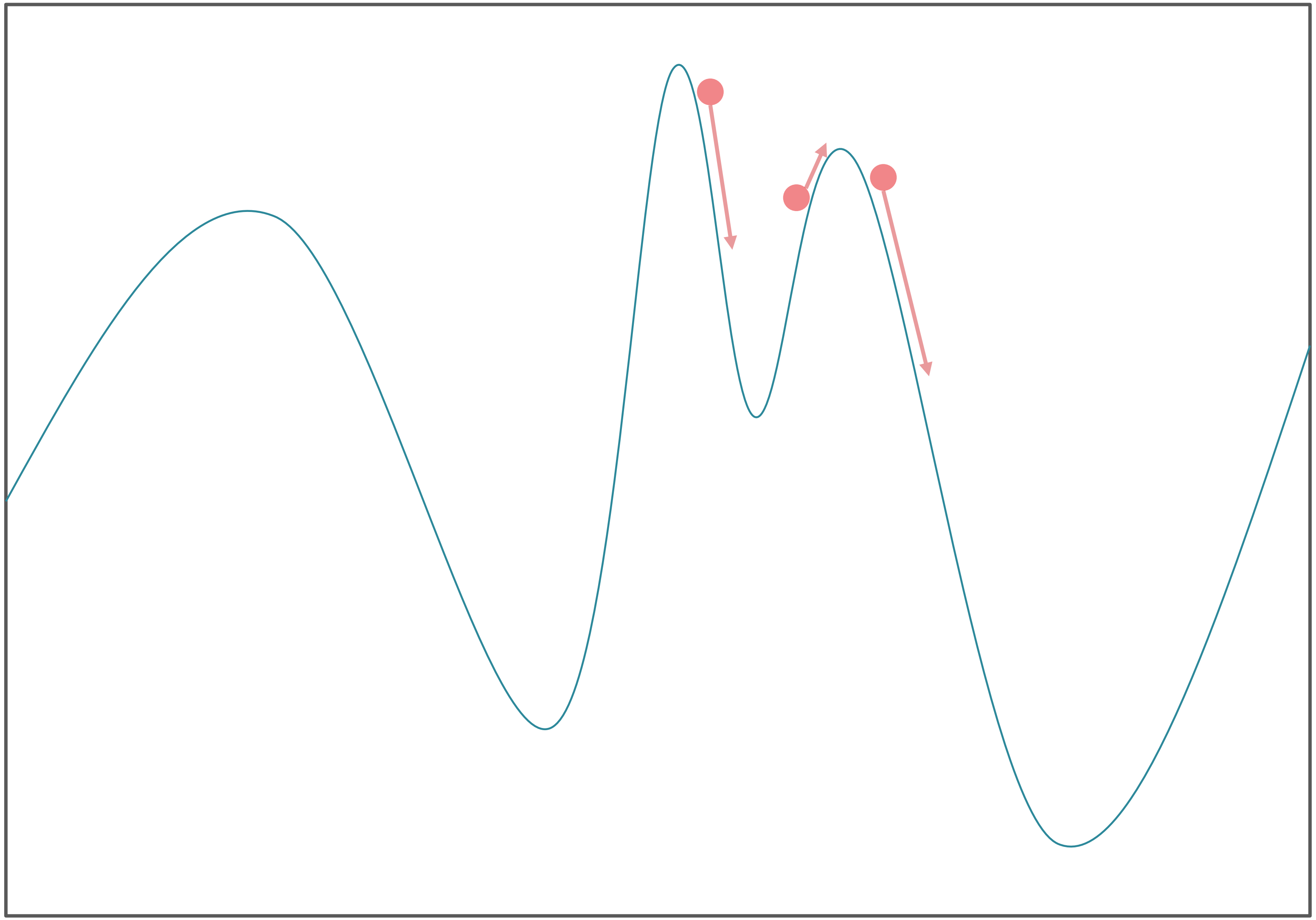
Mini-batch Stochastic Gradient Descent with Momentum



Mini-batch Stochastic Gradient Descent with Momentum



Mini-batch Stochastic Gradient Descent with Momentum



Mini-batch Stochastic Gradient Descent with Root Mean Square Propagation (RMSProp)

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

1. *Pre-train* the parameters $\boldsymbol{\theta}^{(0)}$ and set $t = 0$, $S_{-1} = 0 \odot \boldsymbol{\theta}^{(0)}$
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 *fine-tuning* loss
   $$G_t = \nabla J^{(B)}\left(\boldsymbol{\theta}^{(t)}\right)$$
   c. Update the scaling factor: $S_t = \beta S_{t-1} + (1 - \beta)(G_t \odot G_t)$
   d. Update $\boldsymbol{\theta}$: $\boldsymbol{\theta}^{(t+1)} \leftarrow \boldsymbol{\theta}^{(t)} - \frac{\gamma}{\sqrt{S_t}} \odot G_t$
   e. Increment t: $t \leftarrow t + 1$

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

Adam (Adaptive Moment Estimation) = SGD + Momentum + RMSProp

- Input: training dataset $\mathcal{D} = \left\{\left(\boldsymbol{x}^{(i)}, y^{(i)}\right)\right\}_{i=1}^{N}$,

  step size γ, and batch size B, decay parameters β_1 and β_2

1. *Pre-train* the parameters $\boldsymbol{\theta}^{(0)}$, $t = 0$, $M_{-1} = S_{-1} = 0 \odot \boldsymbol{\theta}^{(0)}$
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, momentum and scaling factor

$$G_t = \nabla J^{(B)}\left(\boldsymbol{\theta}^{(t)}\right)$$

$$M_t = \beta_1 M_{t-1} + (1 - \beta_1) G_t \text{ and } S_t = \beta_2 S_{t-1} + (1 - \beta_2)(G_t \odot G_t)$$

   c. Update $\boldsymbol{\theta}$: $\boldsymbol{\theta}^{(t+1)} \leftarrow \boldsymbol{\theta}^{(t)} - \frac{\gamma}{\sqrt{S_t/(1-\beta_2^t)}} \odot \left(M_t/(1 - \beta_1^t)\right)$
   d. Increment t: $t \leftarrow t + 1$

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