CS 453X: Class 23

Jacob Whitehill

(Unsupervised) pre-training

Feature representations

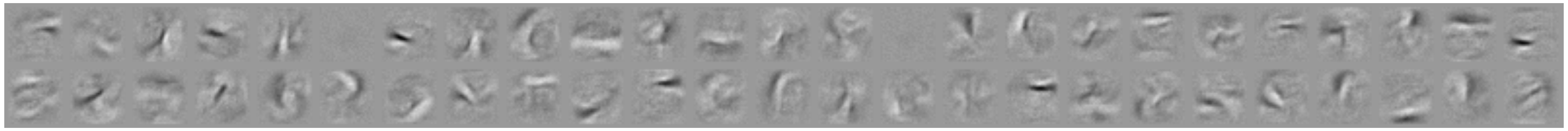
- One of the reasons why NNs are so powerful is that they can learn **feature representations** of the raw input data.
- Classifying/regressing the target variable $\hat{y}$ is often easier once the raw data have been transformed into a different feature space.
    - We saw this with XOR.

Feature representations

- One of the reasons why NNs are so powerful is that they can learn **feature representations** of the raw input data.

- Classifying/regressing the target variable $\hat{y}$ is often easier once the raw data have been transformed into a different feature space.

  - With MNIST, the NN seemed to recognize brush-strokes:



Unlabeled data

- In some application domains (e.g., object recognition/detection in images), collecting *labeled* data is hard, but collecting *unlabeled* data is easy.
- How might oodles of unlabeled data help us to train better ML models?

Learning good features

- We can harness unsupervised learning algorithms to learn good feature representations from unlabeled data.
    - **Unsupervised**: examples without labels.

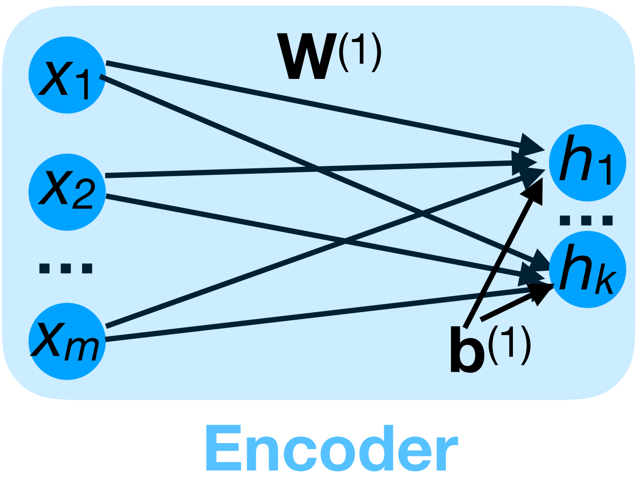
Learning good features

- We can harness unsupervised learning algorithms to learn good feature representations from unlabeled data.
    - **Unsupervised**: examples without labels.
- Key intuition: a good representation captures the essence of the raw input data.
    - We can “compress” the data into a smaller representation.
    - We can “uncompress” it to *reconstruct* the original data.

Auto-encoders

- Let **x** be the raw input data.
- Let **h** be the hidden representation that captures the "essence" of **x**. We say **h** has been **encoded** from **x**.
- We can compute **h** using a neural network (just 1 layer in this example, but could be deeper).

$$\mathbf{h} = \sigma^{(1)}\left(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}\right)$$

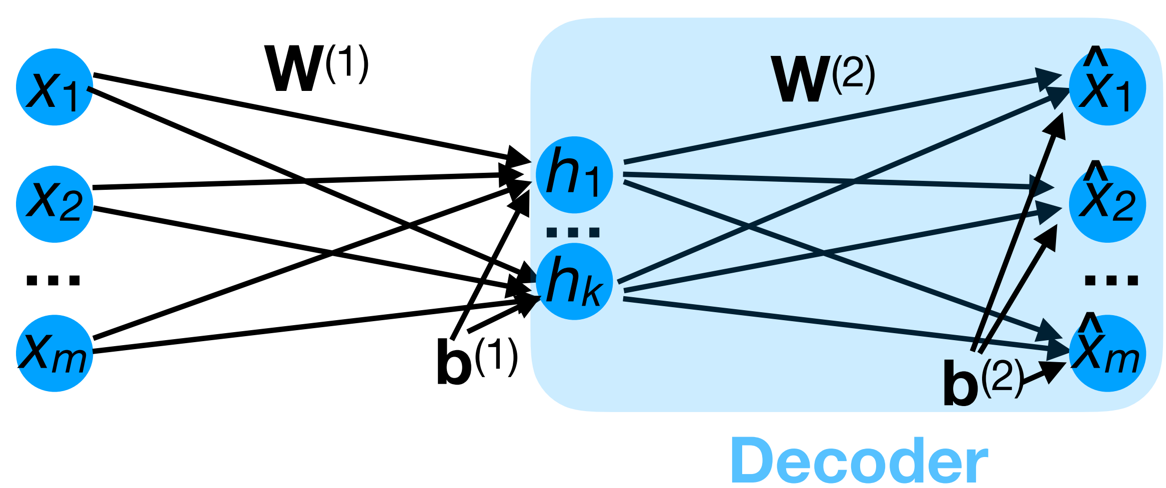


Auto-encoders

- If **h** contains the essential features of **x**, then we can use **h** to reconstruct (approximate) the original data **x**.

- Let **x̂** denote our reconstruction of **x**. We say **x̂** has been **decoded** from **h**.

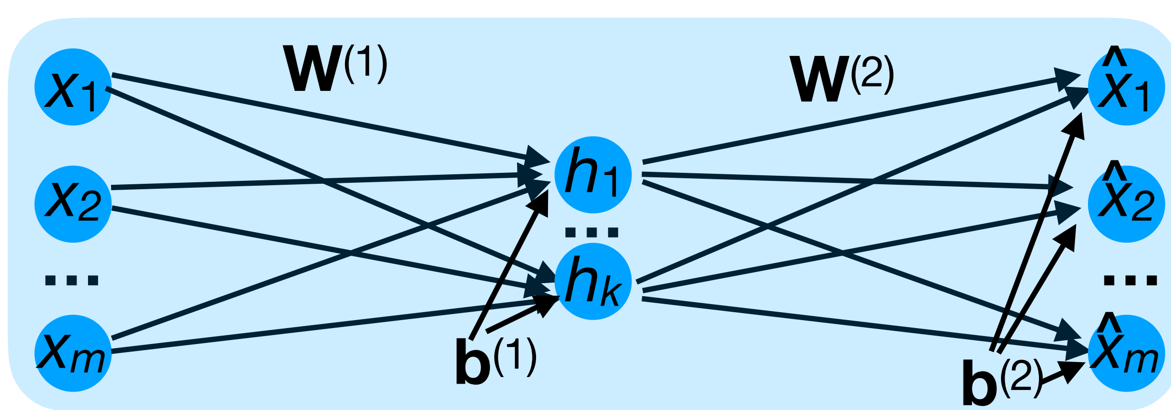
$$\hat{\mathbf{x}} = \sigma^{(2)}\left(\mathbf{W}^{(2)}\mathbf{h} + \mathbf{b}^{(2)}\right)$$



Auto-encoders

- Putting the two components (encoder+decoder) together, we arrive at an **auto-encoder**.

$$\hat{\mathbf{x}} = \sigma^{(2)}\left(\mathbf{W}^{(2)}\sigma^{(1)}\left(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}\right) + \mathbf{b}^{(2)}\right)$$

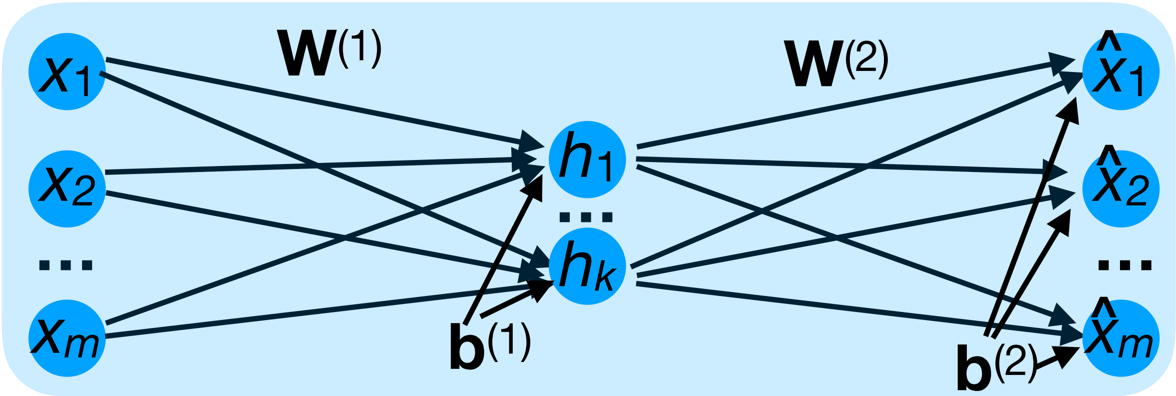


Auto-encoder

Auto-encoders: training loss function

- With auto-encoders, we optimize **W**$^{(1)}$, **W**$^{(2)}$, **b**$^{(1)}$, **b**$^{(2)}$ to make our reconstructions as accurate as possible, i.e., minimize:

$$
\begin{aligned}
f_{\mathrm{MSE}}(\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \mathbf{W}^{(2)}, \mathbf{b}^{(2)}) &= \frac{1}{n}\sum_{i=1}^{n}(\hat{\mathbf{x}}^{(i)} - \mathbf{x}^{(i)})^2 \\
&= \frac{1}{n}\sum_{i=1}^{n}(\hat{\mathbf{x}}^{(i)} - \mathbf{x}^{(i)})^\top(\hat{\mathbf{x}}^{(i)} - \mathbf{x}^{(i)})
\end{aligned}
$$

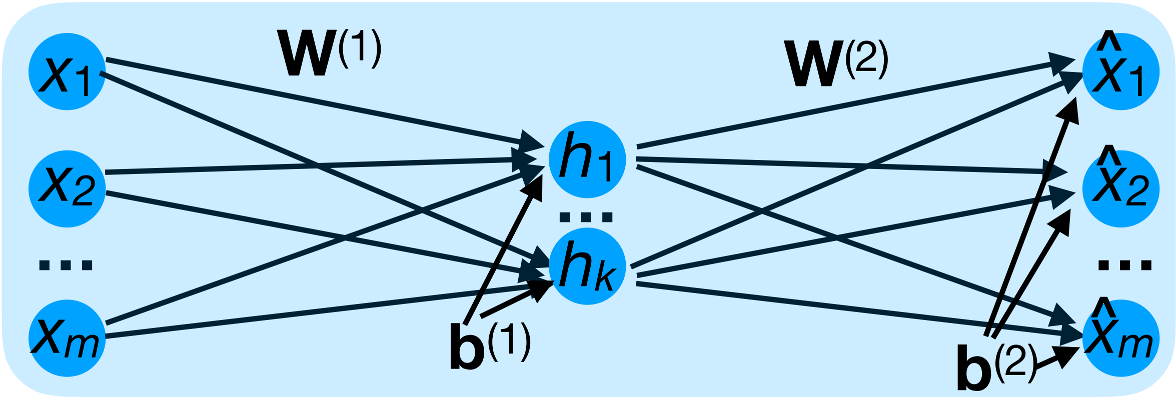


Auto-encoder

Auto-encoders: training loss function

- Notice that this loss function does not require any training labels — there is no mention of any *y*!

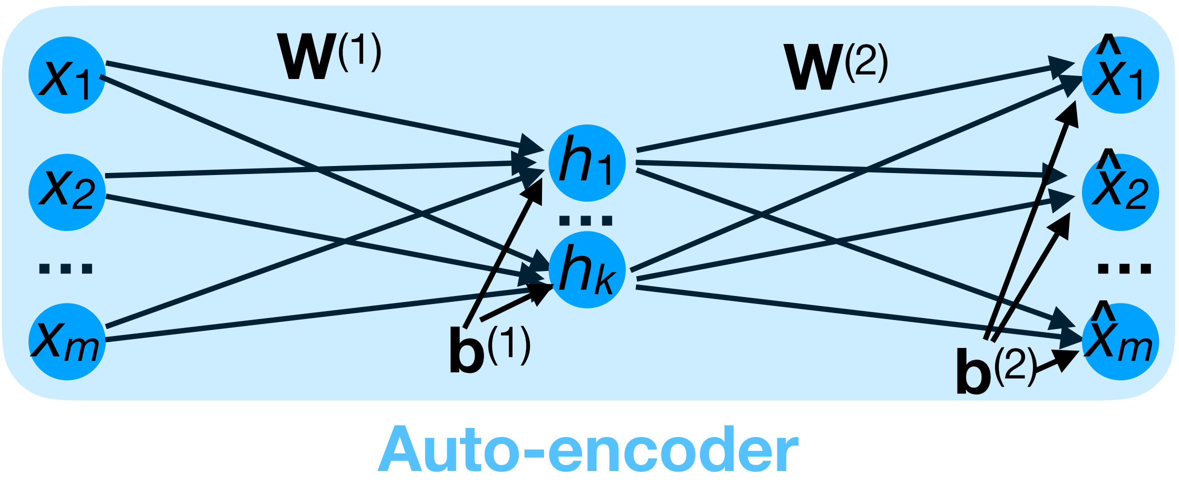
$$\begin{aligned} f_{\mathrm{MSE}}(\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \mathbf{W}^{(2)}, \mathbf{b}^{(2)}) &= \frac{1}{n}\sum_{i=1}^{n}(\hat{\mathbf{x}}^{(i)} - \mathbf{x}^{(i)})^2 \\ &= \frac{1}{n}\sum_{i=1}^{n}(\hat{\mathbf{x}}^{(i)} - \mathbf{x}^{(i)})^\top(\hat{\mathbf{x}}^{(i)} - \mathbf{x}^{(i)}) \end{aligned}$$



Auto-encoder

Exercise

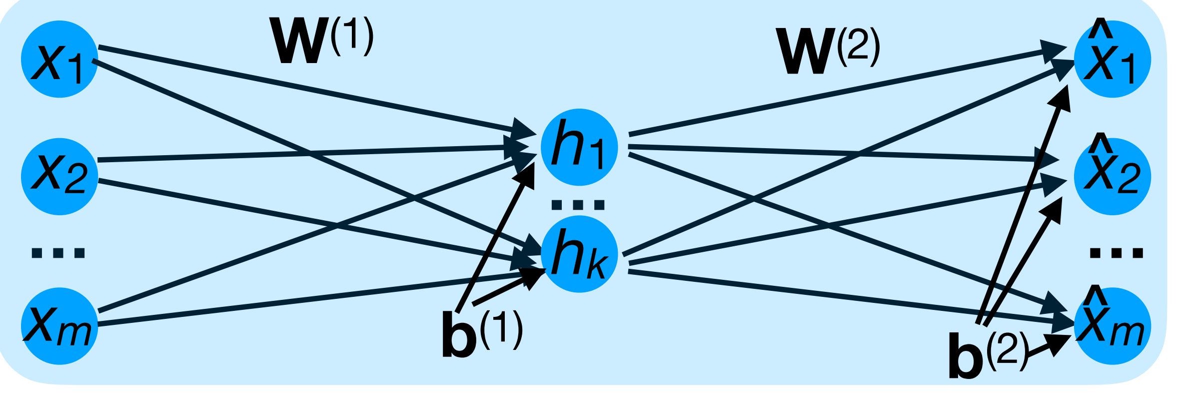
- If we let k=m, then what is an easy (but useless) way to set the weights to give a perfect reconstruction (0 MSE)?



Exercise

- If we let k=m, then what is an easy (but useless) way to set the weights to give a perfect reconstruction (0 MSE)?

- If $k = m$, then we can just set $\mathbf{W}^{(1)} = \mathbf{W}^{(2)} = \mathbf{I}$ (identity matrix). This gives 0 MSE but does not learn any interesting representation!

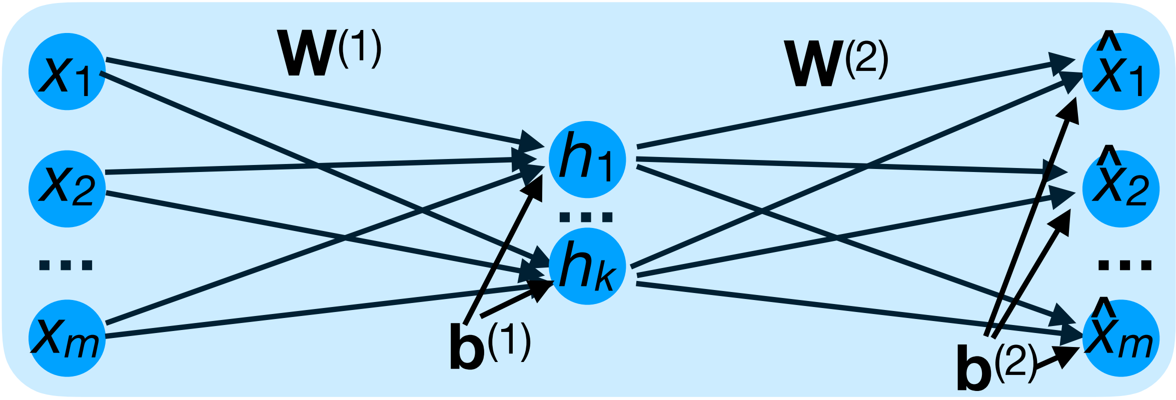


Auto-encoder

Exercise

- If we let k=m, then what is an easy (but useless) way to set the weights to give a perfect reconstruction (0 MSE)?

- For this reason, we usually set $k < m$*; the hidden layer is then called a **bottleneck**.

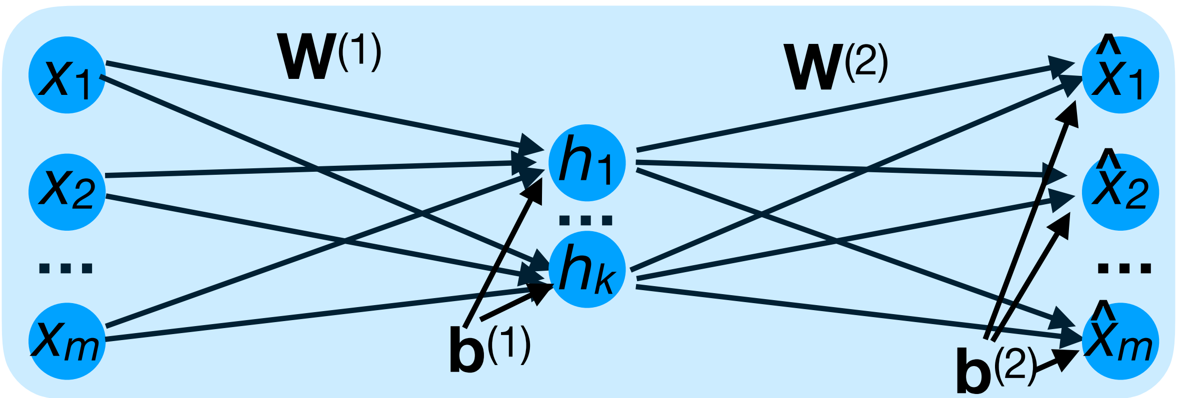


Auto-encoder

*An important exception are de-noising auto-encoders.

Auto-encoders for unsupervised pre-training

- After training the auto-encoder NN, $\mathbf{W}^{(1)}$ and $\mathbf{b}^{(1)}$ have hopefully learned to encode $\mathbf{x}$ into a representation that is useful for a variety of classification/regression problems.

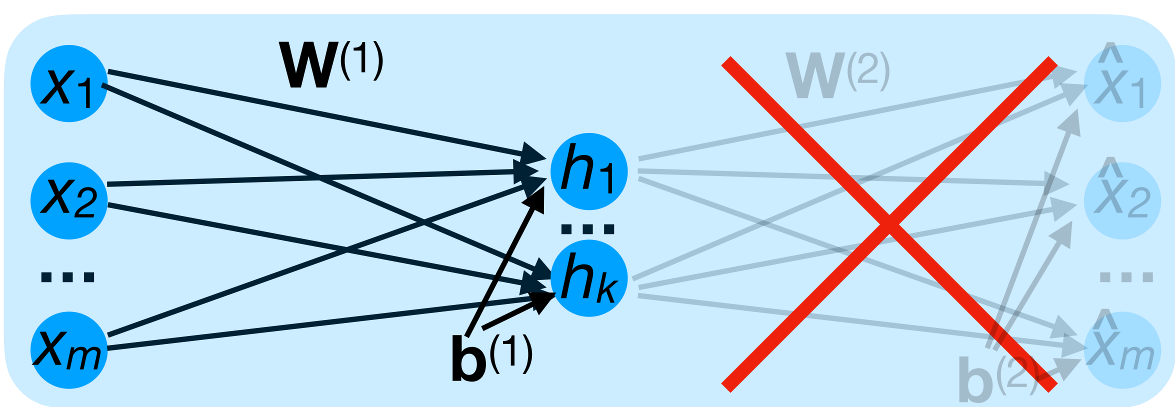


Auto-encoder

Auto-encoders for unsupervised pre-training

- After training the auto-encoder NN, $\mathbf{W}^{(1)}$ and $\mathbf{b}^{(1)}$ have hopefully learned to encode $\mathbf{x}$ into a representation that is useful for a variety of classification/regression problems.

- We can now just "chop off" the decoder layer(s)…

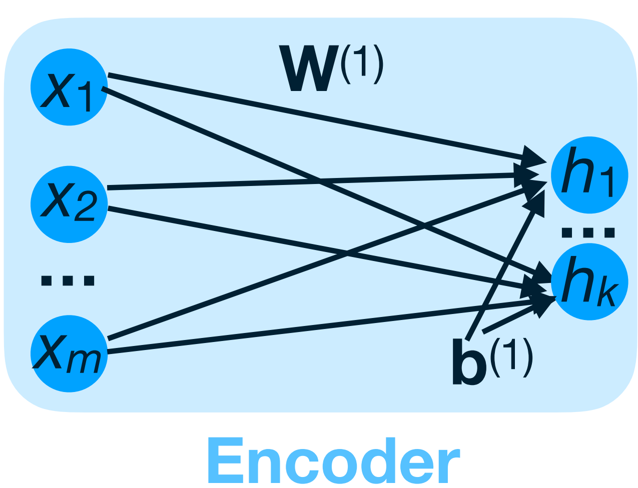


Auto-encoder

Auto-encoders for unsupervised pre-training

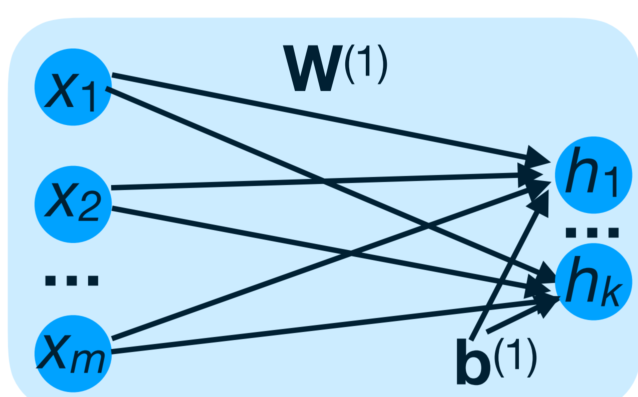
- After training the auto-encoder NN, $\mathbf{W}^{(1)}$ and $\mathbf{b}^{(1)}$ have hopefully learned to encode $\mathbf{x}$ into a representation that is useful for a variety of classification/regression problems.

- …and keep just the encoder layer(s).



Auto-encoders for unsupervised pre-training

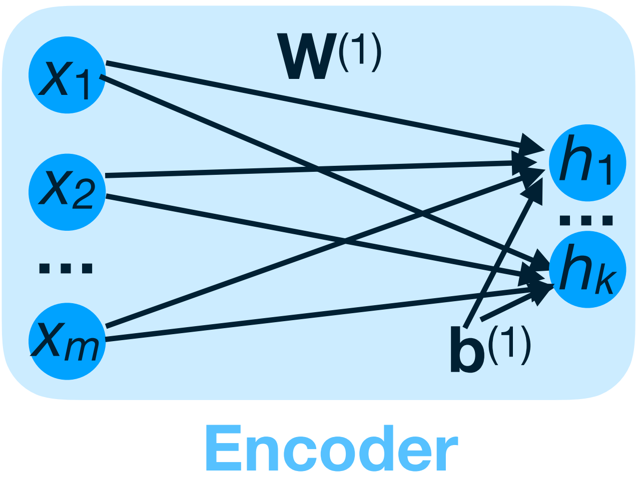
- We now have a trained encoder network that can "compress" every input example **x** into its "essence" **h**.



Encoder

Auto-encoders for unsupervised pre-training

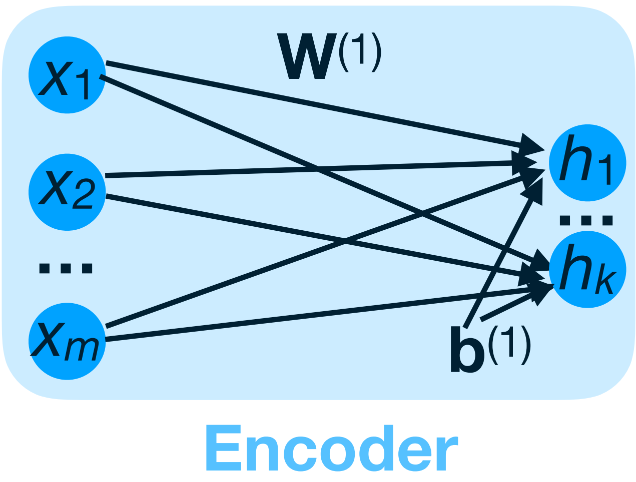
- Now, suppose we also have a (typically smaller) set of *labeled* examples $\{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{n}$.



Auto-encoders for unsupervised pre-training

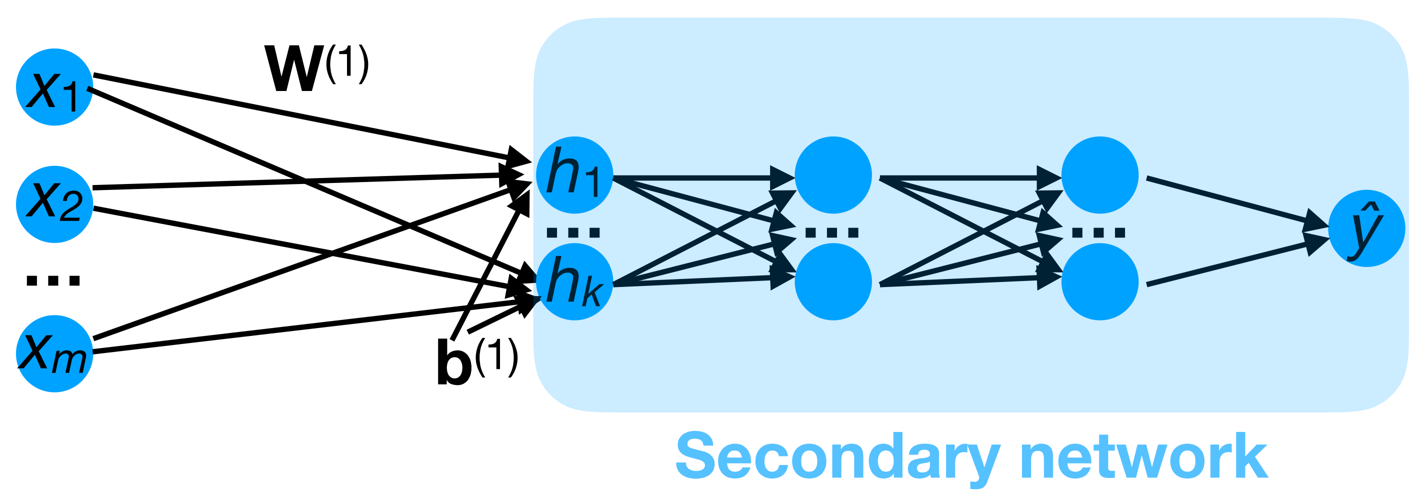
- Now, suppose we also have a (typically smaller) set of *labeled* examples $\{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{n}$.

- We can use the encoder network to convert each **x** to **h** to obtain $\{(\mathbf{h}^{(i)}, y^{(i)})\}_{i=1}^{n}$.



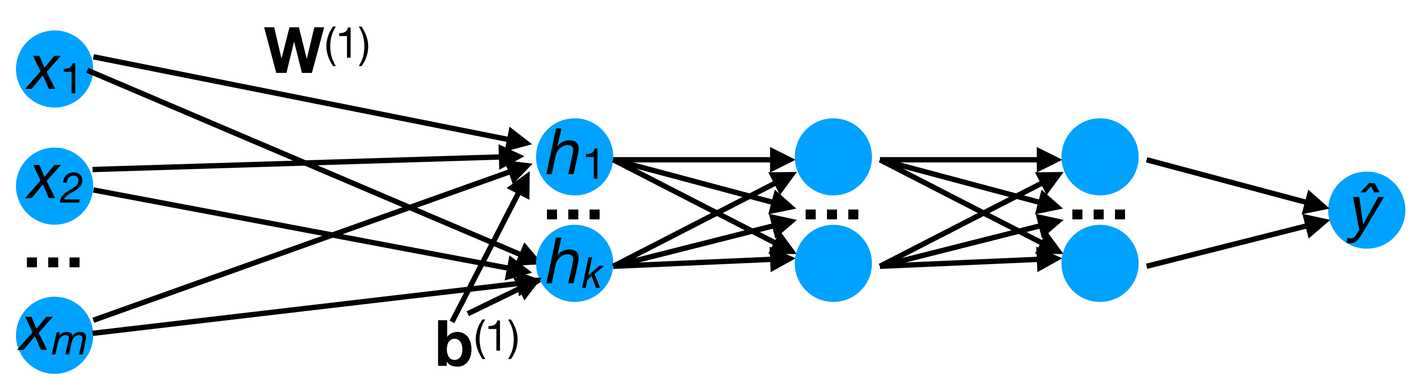
Auto-encoders for unsupervised pre-training

- Now, suppose we also have a (typically smaller) set of *labeled* examples $\{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^n$.
- We can use the encoder network to convert each **x** to **h** to obtain $\{(\mathbf{h}^{(i)}, y^{(i)})\}_{i=1}^n$.
- We then train a *secondary* NN (or any other ML model) to predict *y* from **h**.



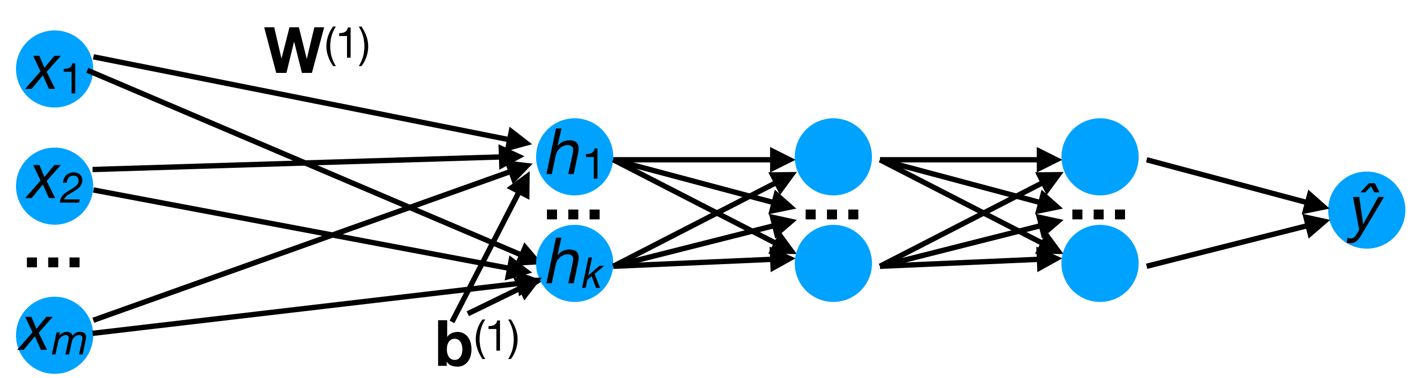
Auto-encoders for unsupervised pre-training

- After training, the two networks (encoder + secondary) can be seen as a *single NN* that analyzes each input $\mathbf{x}$ to make a prediction $\hat{y}$.



Auto-encoders for unsupervised pre-training

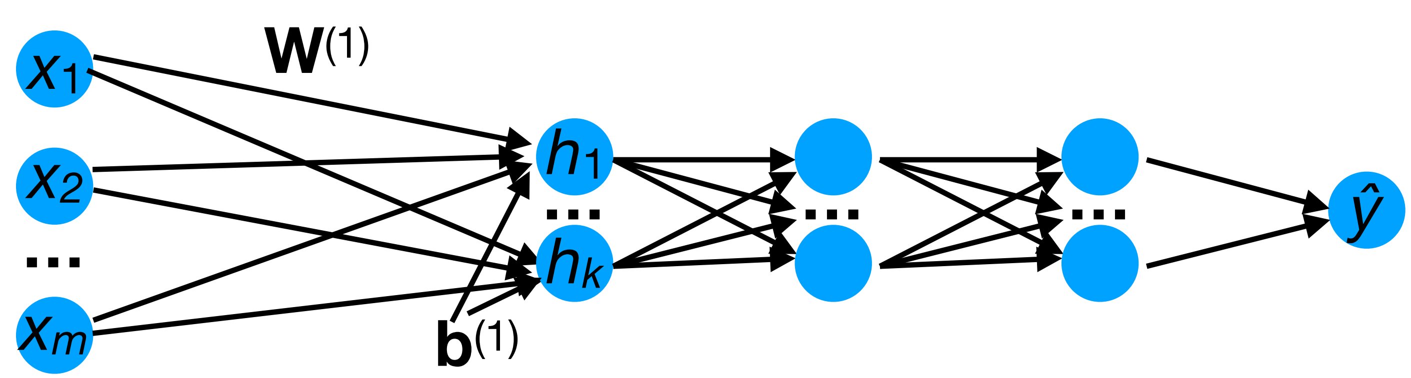
- Why does this help?
- The first layers of the overall network were trained on a large amount of data.
- Compressing **x** into **h** makes the secondary predictions (hopefully) easier.



Auto-encoders for unsupervised pre-training

- In addition to training the secondary NN, we can — optionally — adjust the parameters of the encoder network.

- Since the encoder was trained on a much larger (unlabeled) dataset, we don't want to "mess up" its weights too much based on just a small labeled dataset.

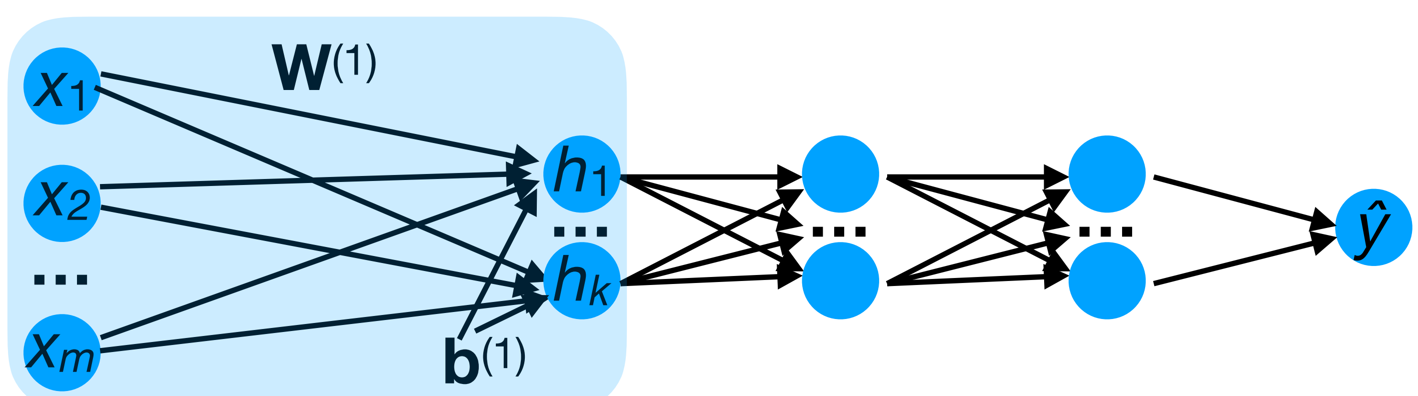


Auto-encoders for unsupervised pre-training

- In addition to training the secondary NN, we can — optionally — adjust the parameters of the encoder network.

- Since the encoder was trained on a much larger (unlabeled) dataset, we don't want to "mess up" its weights too much based on just a small labeled dataset.

  - Hence, we often use a small learning rate ==> **fine-tuning**.

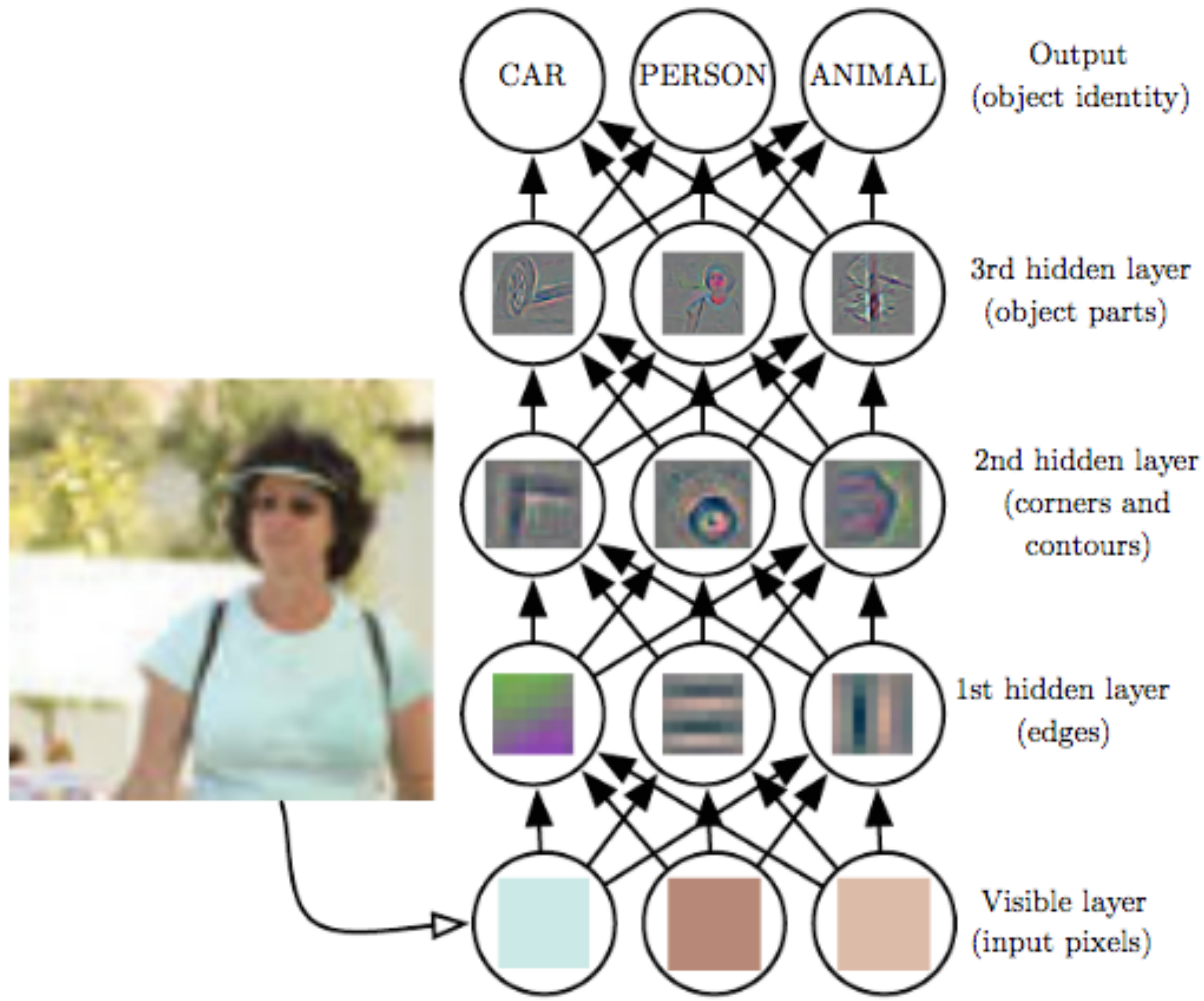


(Supervised) pre-training

Supervised pre-training

- An alternative strategy to finding good feature representations is to borrow a NN from a related task.
- For instance, there now exist high-accuracy networks for recognizing 1000+ object categories from images (next slide).
- We can “borrow” the feature representation from one ML model and apply it to another application domain…

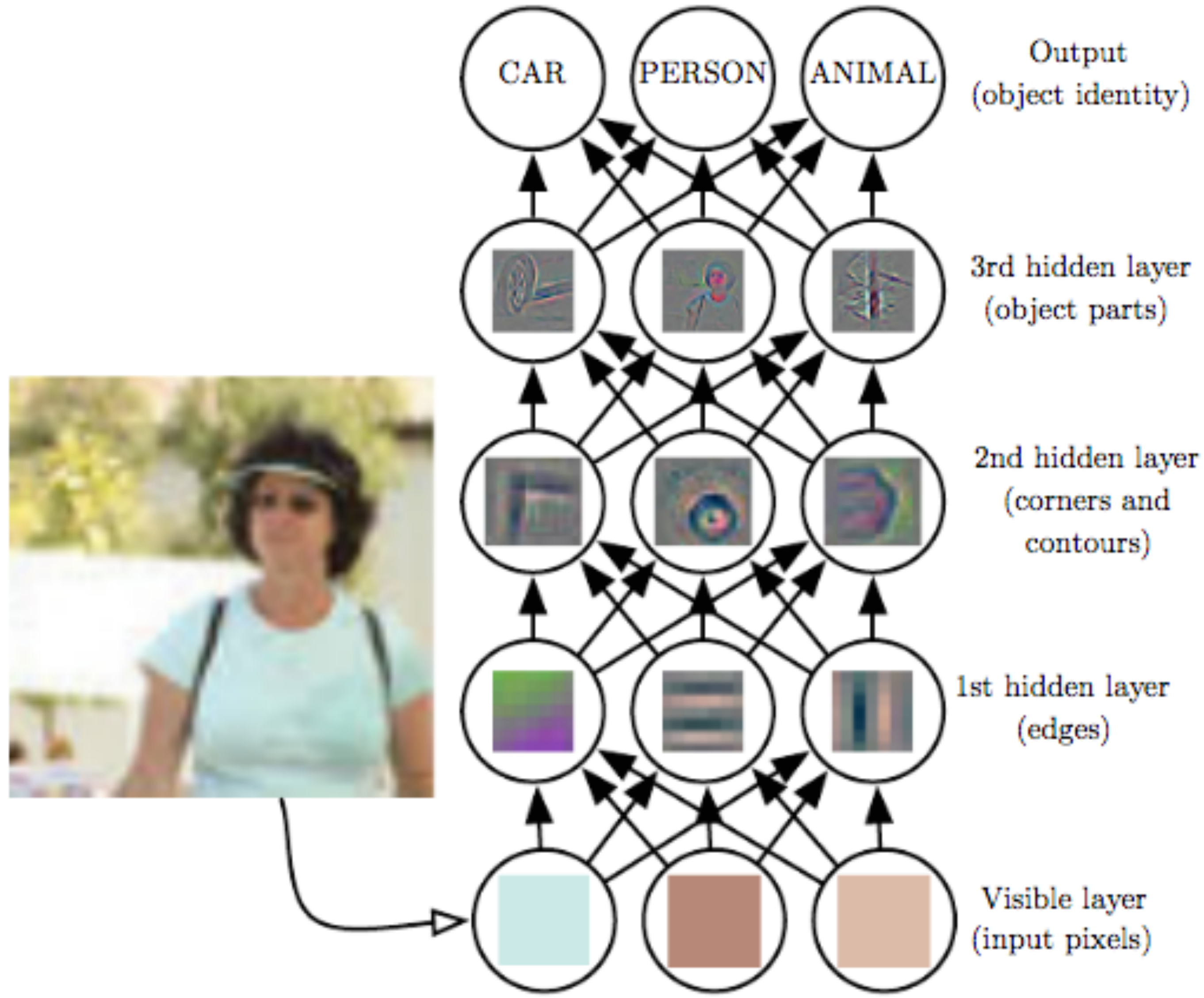
Learning representations



- The first feature representation looks vaguely like the representation learned by my MNIST network.

http://www.deeplearningbook.org/contents/intro.html

Learning representations



- Each layer of the network finds successively more abstract feature representations.
  - This was not "hard-coded" — it just turned out that these representations were useful for predicting the target labels.

http://www.deeplearningbook.org/contents/intro.html

Supervised pre-training

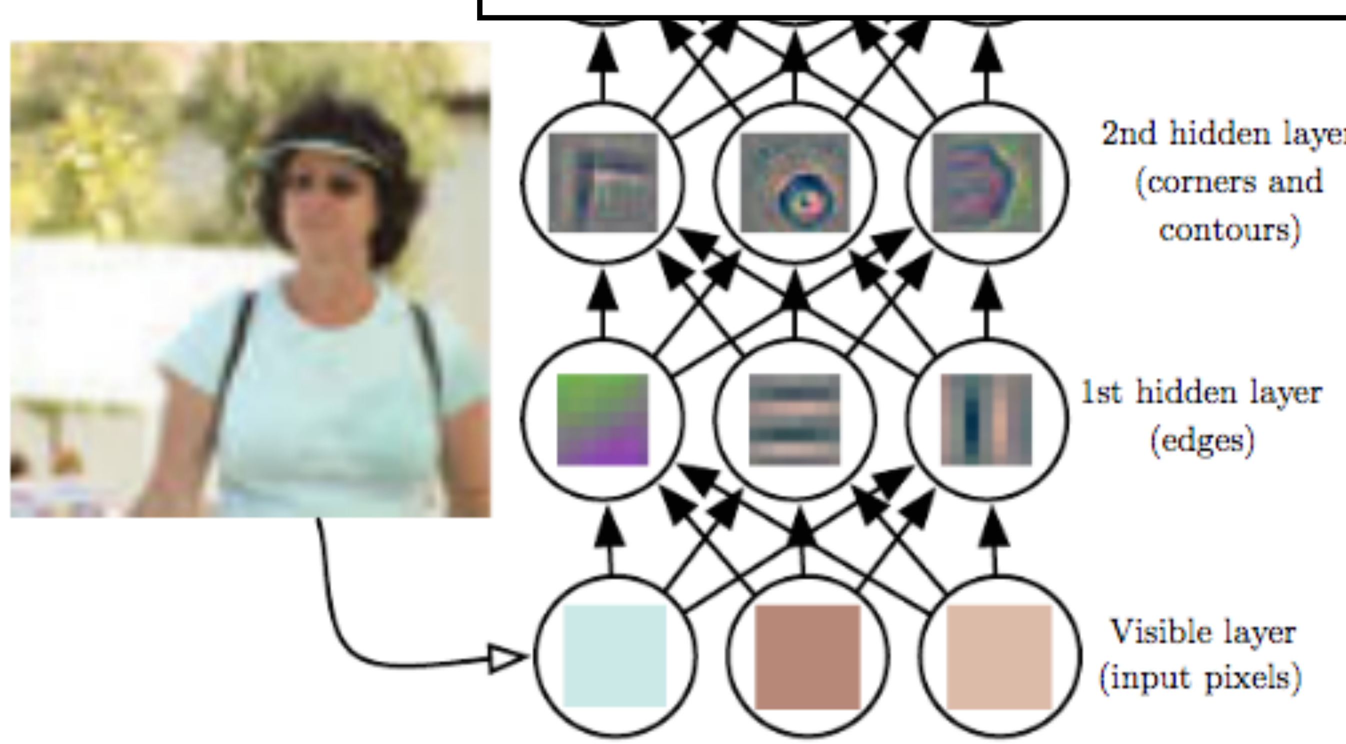
- Might one (or more) of the feature representations from this NN do well on a different but related problem, e.g., smile detection or age estimation?

- Strategy:

1. Pre-train a NN on a large dataset for a general-purpose image recognition task.
2. “Chop off” the final layer(s).
3. Add a secondary network in place of the deleted layers, and train it for the new prediction task.
4. Optional: fine-tune the rest of the NN.

Supervised pre-training

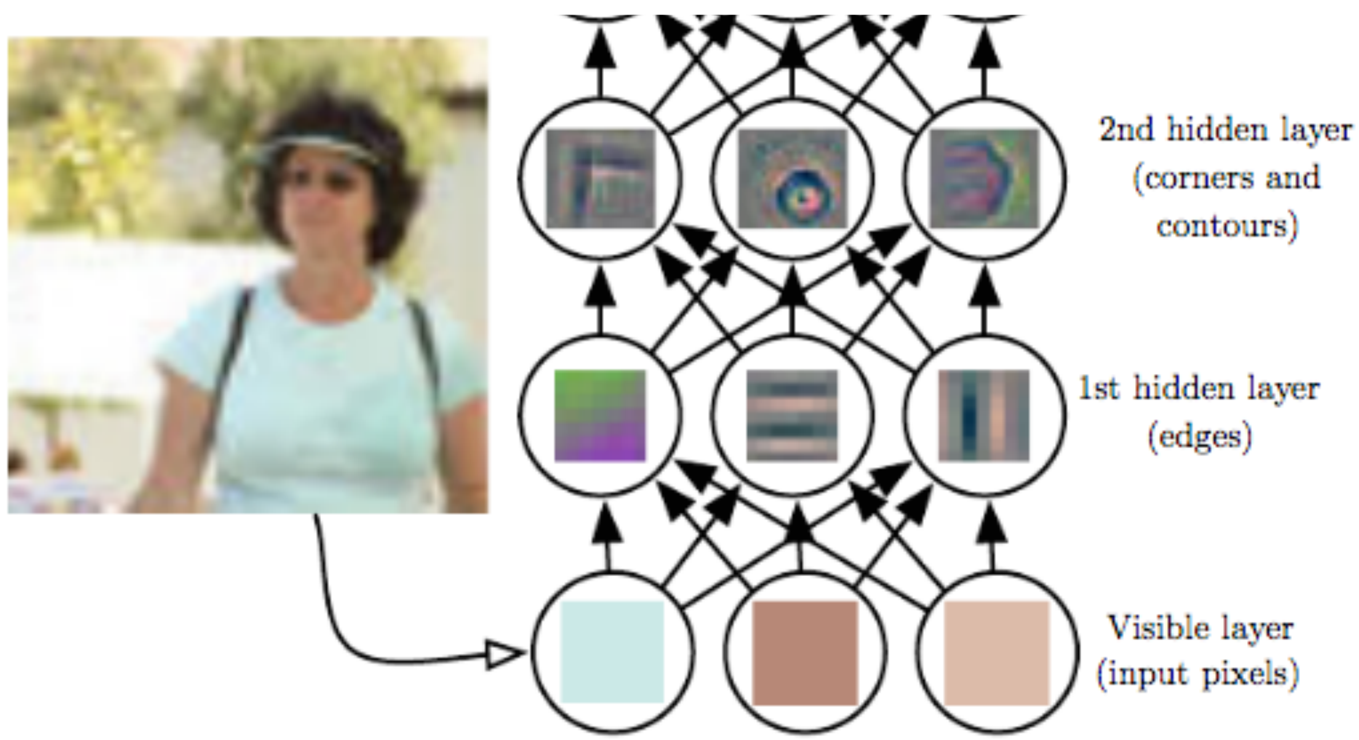
Replace with secondary network
for new application domain.



http://www.deeplearningbook.org/contents/intro.html

Supervised pre-training

Chop off.



http://www.deeplearningbook.org/contents/intro.html

Supervised pre-training

- This strategy is known as **supervised pre-training** and can be highly effective for application domains for which only a small number of labeled data are available.

Convolution neural networks