Topics in AI (CPSC 532S):
Multimodal Learning with Vision, Language and Sound

Lecture 4: Introduction to Deep Learning (continued)

Course **Logistics**

— **Assignment 1** is due 11:59pm on Wednsday (tomorrow)

Course **Logistics**

- **Assignment 1** is due 11:59pm on Wednsday (tomorrow)
- Small bug in **Assignment 1**: Part 4

```
total_correct += (output > 0.5).eq(target).sum().item()
# total_correct += output.argmax(dim=1).eq(target).sum().item()
```

Course **Logistics**

— **Assignment 1** is due 11:59pm on Wednsday (tomorrow)

— Small bug in **Assignment 1**: Part 4

```
total_correct += (output > 0.5).eq(target).sum().item()
# total_correct += output.argmax(dim=1).eq(target).sum().item()
```

— **Assignment 2** will be out Thursday night

(note, it will take computation time)

Short **Review** … weight regularization

L2 Regularization: Learn a more (dense) distributed representation

$$R(\mathbf{W}) = ||\mathbf{W}||_2 = \sum_i \sum_j \mathbf{W}_{i,j}^2$$

L1 Regularization: Learn a sparse representation (few non-zero wight elements)

$$R(\mathbf{W}) = ||\mathbf{W}||_1 = \sum_i \sum_j |\mathbf{W}_{i,j}|$$

(others regularizers are also possible)

Example:

$$\mathbf{x} = [1, 1, 1, 1]$$

$$\mathbf{W}_1 = [1, 0, 0, 0]$$

$$\mathbf{W}_2 = \left[\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}\right]$$

$$\mathbf{W}_1 \cdot \mathbf{x}^T = \mathbf{W}_2 \cdot \mathbf{x}^T$$

two networks will have identical output

L2 Regularizer:

$$R_{L2}(\mathbf{W}_1) = 1$$
$$R_{L2}(\mathbf{W}_2) = 0.25$$

L1 Regularizer:

$$R_{L1}(\mathbf{W}_1) = 1$$
$$R_{L1}(\mathbf{W}_2) = 1$$

Short **Review** … batch normalization

Normalize each mini-batch (using Batch Normalization layer) by subtracting empirically computed mean and dividing by variance for every dimension -> samples are approximately unit Gaussian

$$\bar{x}^{(k)} = \frac{x^{(k)} - \mathbb{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

In practice, also learn how to scale and offset:

$$y^{(k)} = \underbrace{\gamma^{(k)}\bar{x}^{(k)} + \beta^{(k)}}_{\text{BN layer parameters}}$$

Benefit:

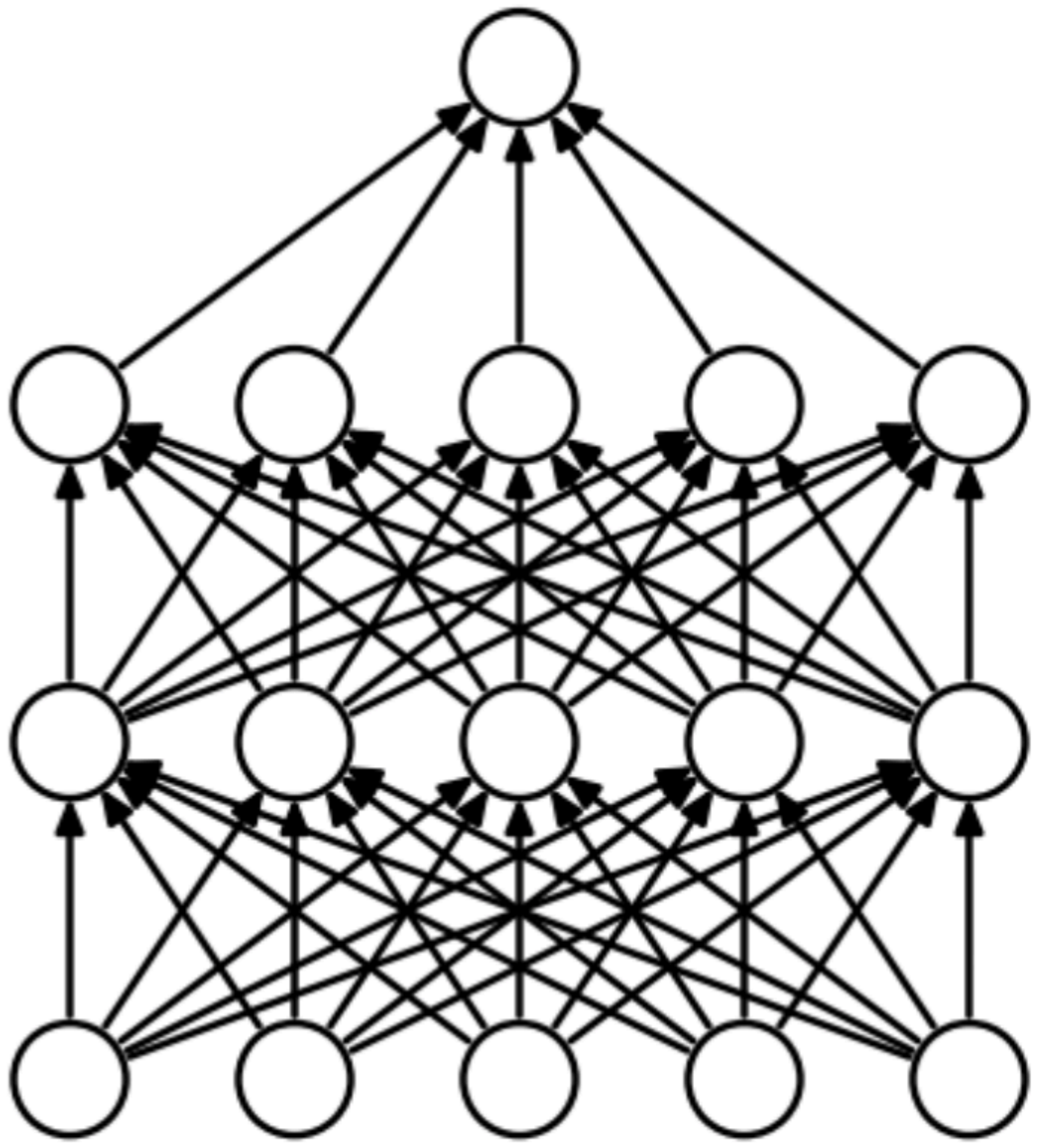
Improves learning (better gradients, higher learning rate, less reliance on initialization)

Typically inserted **before** activation layer

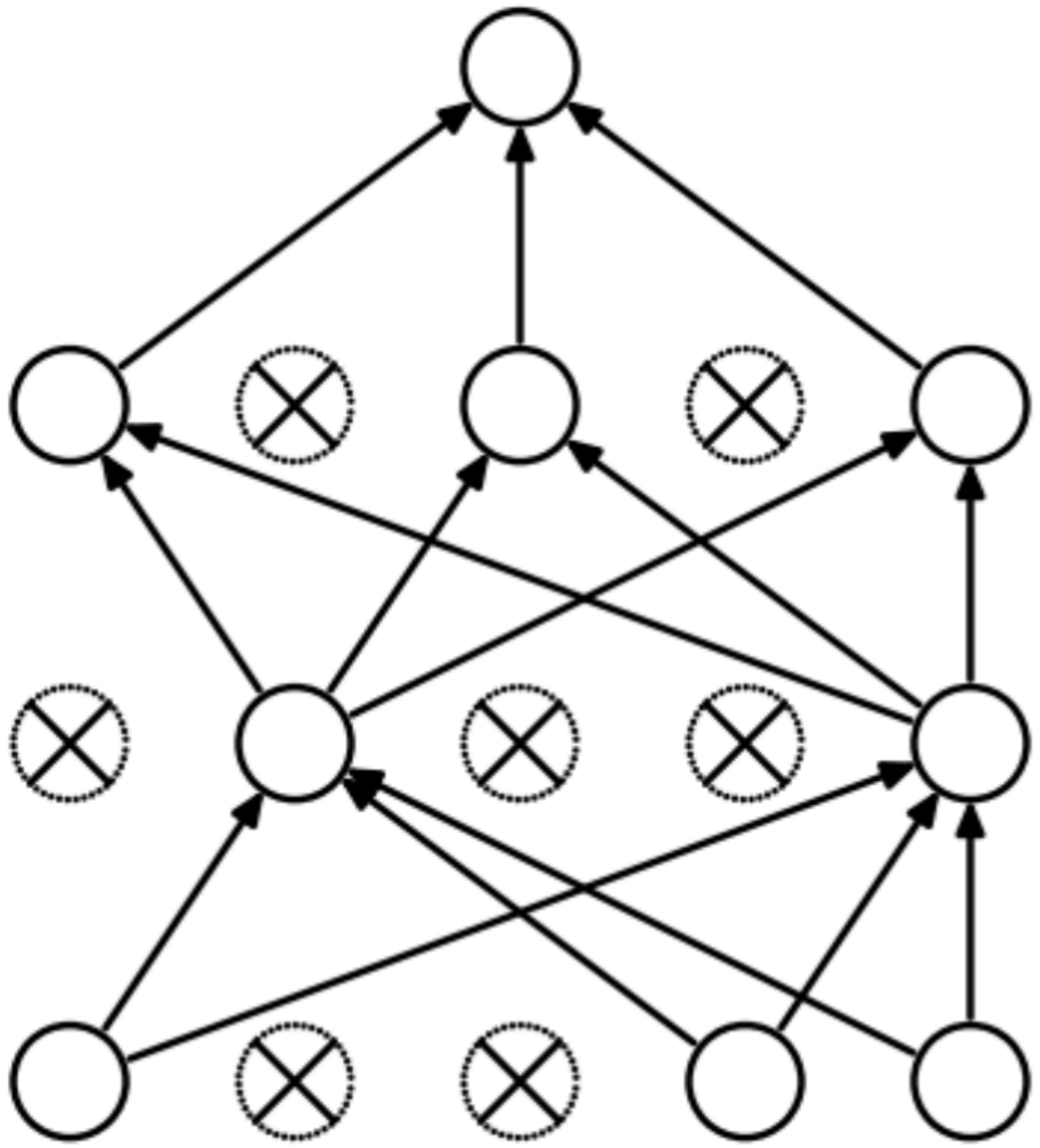
[Ioffe and Szegedy, NIPS 2015]

Short **Review** … dropout

Randomly **set some neurons to zero** in the forward pass, with probability proportional to `dropout rate` (between 0 to 1)



Standar Neural Network

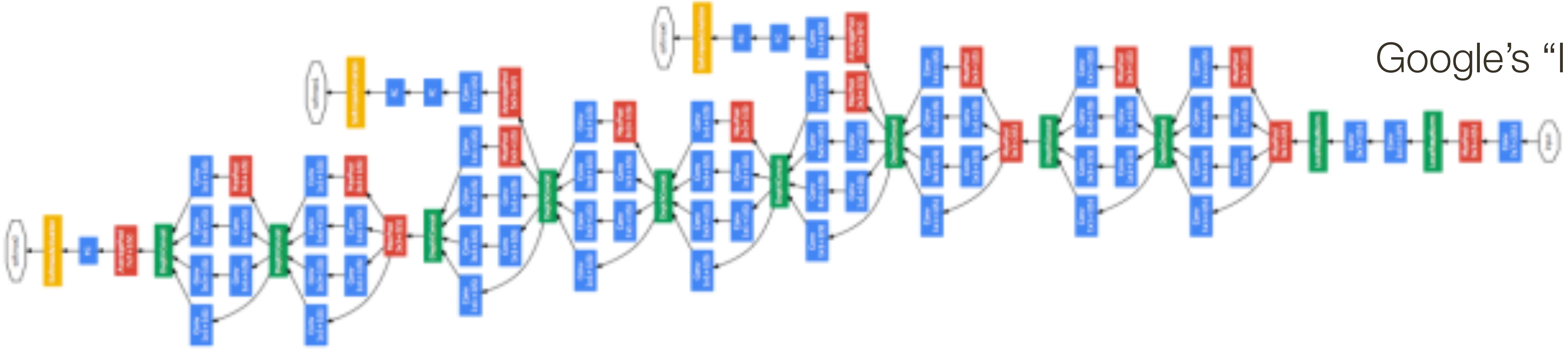


After Applying **Dropout**

[Srivastava et al, JMLR 2014]

* adopted from slides of **CS231n at Stanford**

Deep Learning **Terminology**



Google's "Inception" network

- **Network structure:** number and types of layers, forms of activation functions, dimensionality of each layer and connections (defines computational graph)

generally kept fixed, requires some knowledge of the problem and NN to sensibly set

deeper = better

- **Loss function:** objective function being optimized (`softmax`, `cross entropy`, *etc.*)

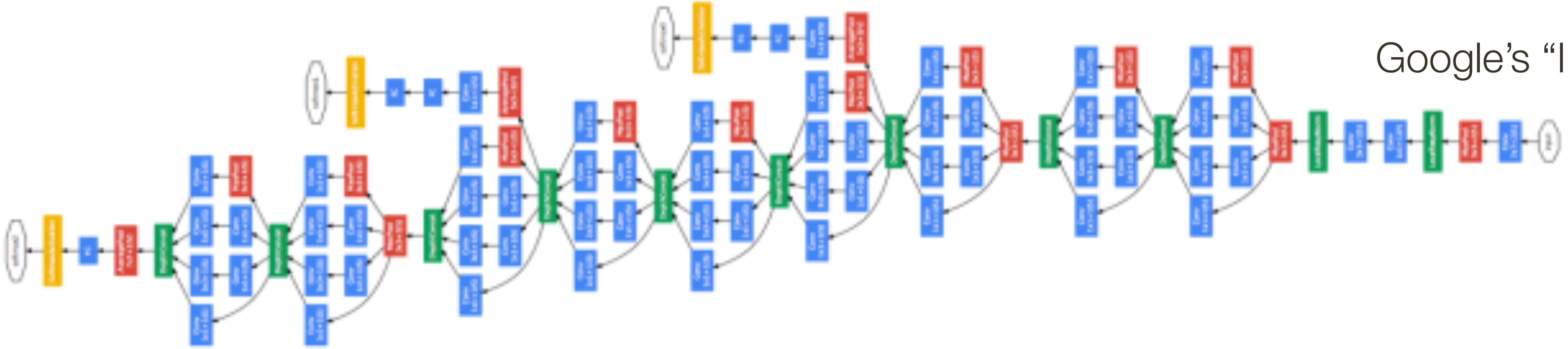
requires knowledge of the nature of the problem

- **Parameters:** trainable parameters of the network, including weights/biases of linear/fc layers, parameters of the activation functions, *etc.*

optimized using SGD or variants

- **Hyper-parameters:** parameters, including for optimization, that are not optimized directly as part of training (*e.g.*, `learning rate`, `batch size`, `drop-out rate`)

Deep Learning **Terminology**



Google's "Inception" network

- **Network structure:** number and types of layers, forms of activation functions, dimensionality of each layer and connections (defines computational graph)

generally kept fixed, requires some knowledge of the problem and NN to sensibly set

deeper = better

- **Loss function:** objective function being optimized (`softmax`, `cross entropy`, *etc.*)

requires knowledge of the nature of the problem

- **Parameters:** trainable parameters of the network, including weights/biases of linear/fc layers, parameters of the activation functions, *etc.*

optimized using SGD or variants

- **Hyper-parameters:** parameters, including for optimization, that are not optimized directly as part of training (*e.g.*, `learning rate`, `batch size`, `drop-out rate`)

grid search

Multivariate **Regression**

Input: feature vector $\mathbf{x} \in \mathbb{R}^n$

Output: output vector $\mathbf{y} \in \mathbb{R}^m$

Neural Network (input + intermediate hidden layers) $f(\mathbf{x};\Theta) : \mathbb{R}^n \rightarrow \mathbb{R}^k$

with **sigmoid** activations: $\mathbf{0} \leq f(\mathbf{x};\Theta) \leq \mathbf{1}$

with **Tanh** activations: $-\mathbf{1} \leq f(\mathbf{x};\Theta) \leq \mathbf{1}$

with **ReLU** activations: $\mathbf{0} \leq f(\mathbf{x};\Theta)$

Neural Network (output): linear layer

$$\hat{\mathbf{y}} = g(\mathbf{x};\mathbf{W},\mathbf{b}) = \mathbf{W}f(\mathbf{x};\Theta) + \mathbf{b} : \mathbb{R}^k \rightarrow \mathbb{R}^m$$

Loss: $\mathcal{L}(\mathbf{y},\hat{\mathbf{y}}) = ||\mathbf{y} - \hat{\mathbf{y}}||^2$

Binary Classification (Bernoulli)

Input: feature vector $\mathbf{x} \in \mathbb{R}^n$

Output: binary label $y \in \{0, 1\}$

Neural Network (input + intermediate hidden layers) $f(\mathbf{x}; \Theta) : \mathbb{R}^n \rightarrow \mathbb{R}$

with **sigmoid** activations: $\mathbf{0} \leq f(\mathbf{x}; \Theta) \leq \mathbf{1}$

Neural Network (output): interpret sigmoid output as probability

$$p(y = 1) = f(\mathbf{x}; \Theta)$$

can interpret the score as the log-odds of $y = 1$ (a.k.a. the **logits**)

Loss:

$$\mathcal{L}(y, \hat{y}) = \begin{cases} -log[1 - f(\mathbf{x}; \Theta)] & y = 0 \\ -log[f(\mathbf{x}; \Theta)] & y = 1 \end{cases}$$

Binary Classification (Bernoulli)

Input: feature vector $\mathbf{x} \in \mathbb{R}^n$

Output: binary label $y \in \{0, 1\}$

Neural Network (input + intermediate hidden layers) $f(\mathbf{x}; \Theta) : \mathbb{R}^n \to \mathbb{R}$

with **sigmoid** activations: $\mathbf{0} \leq f(\mathbf{x}; \Theta) \leq \mathbf{1}$

Neural Network (output): interpret sigmoid output as probability

$$p(y = 1) = f(\mathbf{x}; \Theta)$$

Minimizing this **loss** is the same as maximizing **log likelihood** of data

Loss:

$$\mathcal{L}(y, \hat{y}) = \begin{cases} -log[1 - f(\mathbf{x}; \Theta)] & y = 0 \\ -log[f(\mathbf{x}; \Theta)] & y = 1 \end{cases}$$

Multiclass Classification (e.g, ImageNet)

Input: feature vector $\mathbf{x} \in \mathbb{R}^n$

Output: muticlass label $\mathbf{y} \in \{0, 1\}^m$

(**one-hot** encoding)

Multiclass Classification (e.g, ImageNet)

Input: feature vector $\mathbf{x} \in \mathbb{R}^n$

Output: muticlass label $\mathbf{y} \in \{0, 1\}^m$

(**one-hot** encoding)

Neural Network (input + intermediate hidden layers) $f(\mathbf{x}; \Theta) : \mathbb{R}^n \to \mathbb{R}^m$

with **ReLU** activations: $\mathbf{0} \leq f(\mathbf{x}; \Theta)$

Multiclass Classification (e.g, ImageNet)

Input: feature vector $\mathbf{x} \in \mathbb{R}^n$

Output: muticlass label $\mathbf{y} \in \{0, 1\}^m$

(**one-hot** encoding)

Neural Network (input + intermediate hidden layers) $f(\mathbf{x}; \Theta) : \mathbb{R}^n \to \mathbb{R}^m$

with **ReLU** activations: $\mathbf{0} \leq f(\mathbf{x}; \Theta)$

Neural Network (output): **softmax** function, where probability of class k is:

$$p(\mathbf{y}_k = 1) = \frac{\exp\left[f(\mathbf{x}; \Theta)_i\right]}{\sum_{j=1}^{C} \exp\left[f(\mathbf{x}; \Theta)_j\right]}$$

Multiclass Classification (e.g, ImageNet)

Input: feature vector $\mathbf{x} \in \mathbb{R}^n$

Output: muticlass label $\mathbf{y} \in \{0,1\}^m$

(**one-hot** encoding)

Neural Network (input + intermediate hidden layers) $f(\mathbf{x}; \Theta) : \mathbb{R}^n \to \mathbb{R}^m$

with **ReLU** activations: $\mathbf{0} \leq f(\mathbf{x}; \Theta)$

Neural Network (output): **softmax** function, where probability of class k is:

$$p(\mathbf{y}_k = 1) = \frac{\exp\left[f(\mathbf{x}; \Theta)_i\right]}{\sum_{j=1}^{C} \exp\left[f(\mathbf{x}; \Theta)_j\right]}$$

convert score into **probability**

normalize to sum up to 1 across classes

Multiclass Classification (e.g, ImageNet)

Input: feature vector $\mathbf{x} \in \mathbb{R}^n$

Output: muticlass label $\mathbf{y} \in \{0, 1\}^m$

(**one-hot** encoding)

Neural Network (input + intermediate hidden layers) $f(\mathbf{x}; \Theta) : \mathbb{R}^n \to \mathbb{R}^m$

with **ReLU** activations: $\mathbf{0} \leq f(\mathbf{x}; \Theta)$

Neural Network (output): **softmax** function, where probability of class k is:

$$p(\mathbf{y}_k = 1) = \frac{\exp\left[f(\mathbf{x}; \Theta)_i\right]}{\sum_{j=1}^{C} \exp\left[f(\mathbf{x}; \Theta)_j\right]}$$

Loss: $\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}) = H(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_i \mathbf{y}_i \log \hat{\mathbf{y}}_i$

Multiclass Classification (e.g, ImageNet)

Input: feature vector $\mathbf{x} \in \mathbb{R}^n$

Output: muticlass label $\mathbf{y} \in \{0, 1\}^m$

(**one-hot** encoding)

Neural Network (input + intermediate hidden layers) $f(\mathbf{x}; \Theta) : \mathbb{R}^n \to \mathbb{R}^m$

with **ReLU** activations: $\mathbf{0} \leq f(\mathbf{x}; \Theta)$

Neural Network (output): **softmax** function, where probability of class k is:

$$p(\mathbf{y}_k = 1) = \frac{\exp\left[f(\mathbf{x}; \Theta)_i\right]}{\sum_{j=1}^{C} \exp\left[f(\mathbf{x}; \Theta)_j\right]}$$

Loss: $$\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}) = H(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_i \mathbf{y}_i \log \hat{\mathbf{y}}_i = -\log \hat{\mathbf{y}}_i$$

Special case for multi-class single label

Neural Network Debugging

Neural Network Debugging

1. There is **<u>no way</u>** to write “unit tests” for NN training or inference

Neural Network Debugging

1. There is **<u>no way</u>** to write “unit tests” for NN training or inference
2. **Visualize** your data coming out of a data loader (and inference)

   (a lot of mistakes can be made in the data loader itself)

Neural Network Debugging

1. There is **<u>no way</u>** to write “unit tests” for NN training or inference
2. **Visualize** your data coming out of a data loader (and inference)

   (a lot of mistakes can be made in the data loader itself)
3. Learning code (and sometimes inference code) is stochastic which makes it very hard to debug. Until you are sure code is correct, **fix all the random seeds**

   (Python, NumPy, PyTorch, and Dataloader classes all have separate seeds)

Neural Network Debugging

1. There is **no way** to write "unit tests" for NN training or inference
2. **Visualize** your data coming out of a data loader (and inference)

   (a lot of mistakes can be made in the data loader itself)
3. Learning code (and sometimes inference code) is stochastic which makes it very hard to debug. Until you are sure code is correct, **fix all the random seeds**.

   (Python, NumPy, PyTorch, and Dataloader classes all have separate seeds)
4. Train with a **single example** first (always!). You should be able to obtain 0 loss, i.e., overfit. If this is not the case, there is typically error in model definition.

Neural Network Debugging

1. There is **<u>no way</u>** to write "unit tests" for NN training or inference
2. **Visualize** your data coming out of a data loader (and inference)

   (a lot of mistakes can be made in the data loader itself)
3. Learning code (and sometimes inference code) is stochastic which makes it very hard to debug. Until you are sure code is correct, **fix all the random seeds**

   (Python, NumPy, PyTorch, and Dataloader classes all have separate seeds)
4. Train with a **single example** first (always!). You should be able to obtain 0 loss, i.e., overfit. If this is not the case, there is typically error in model definition.
5. Train with a single **min-batch** next. Your loss may not be 0 at this point, but you should see convergence.

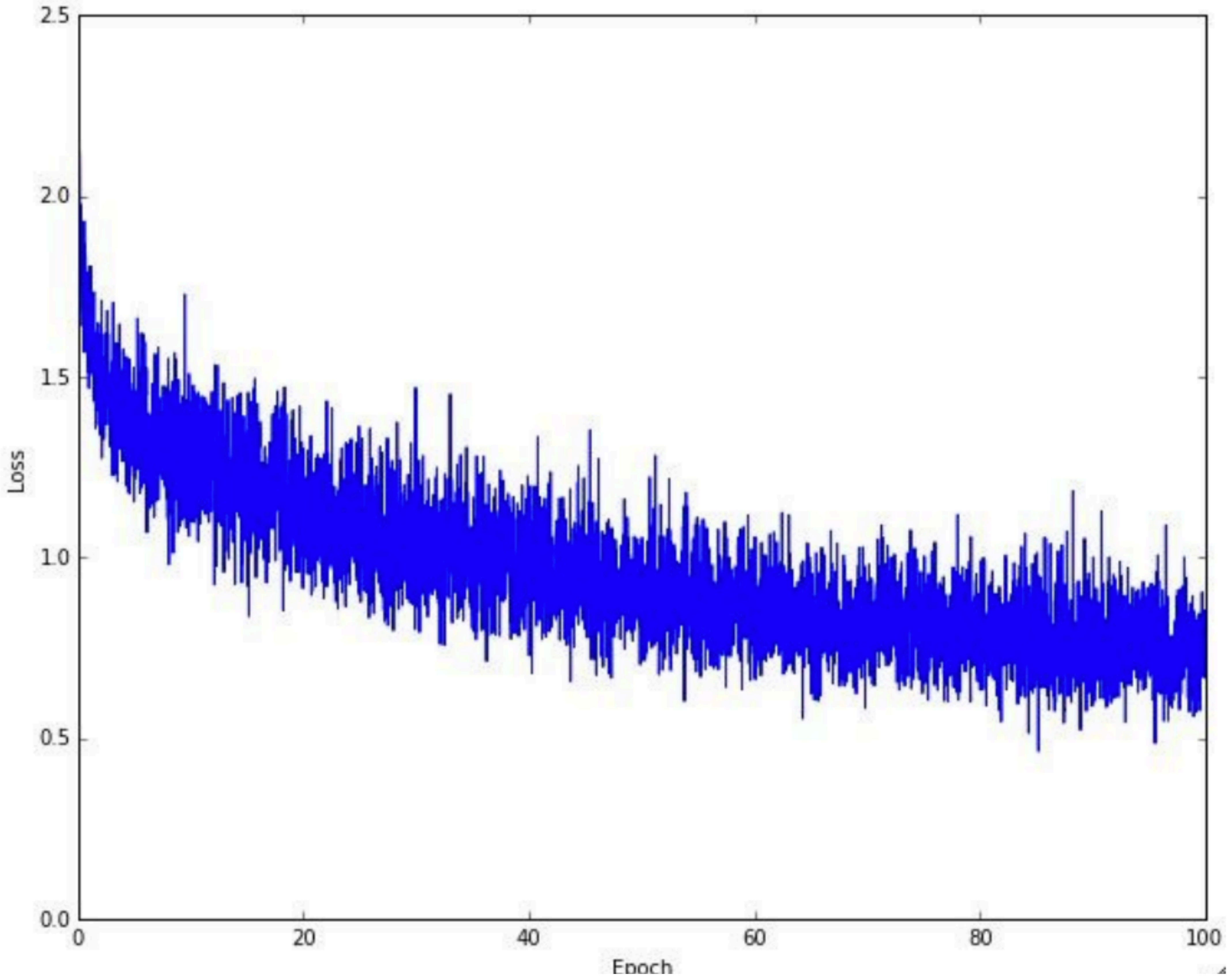
Neural Network Debugging

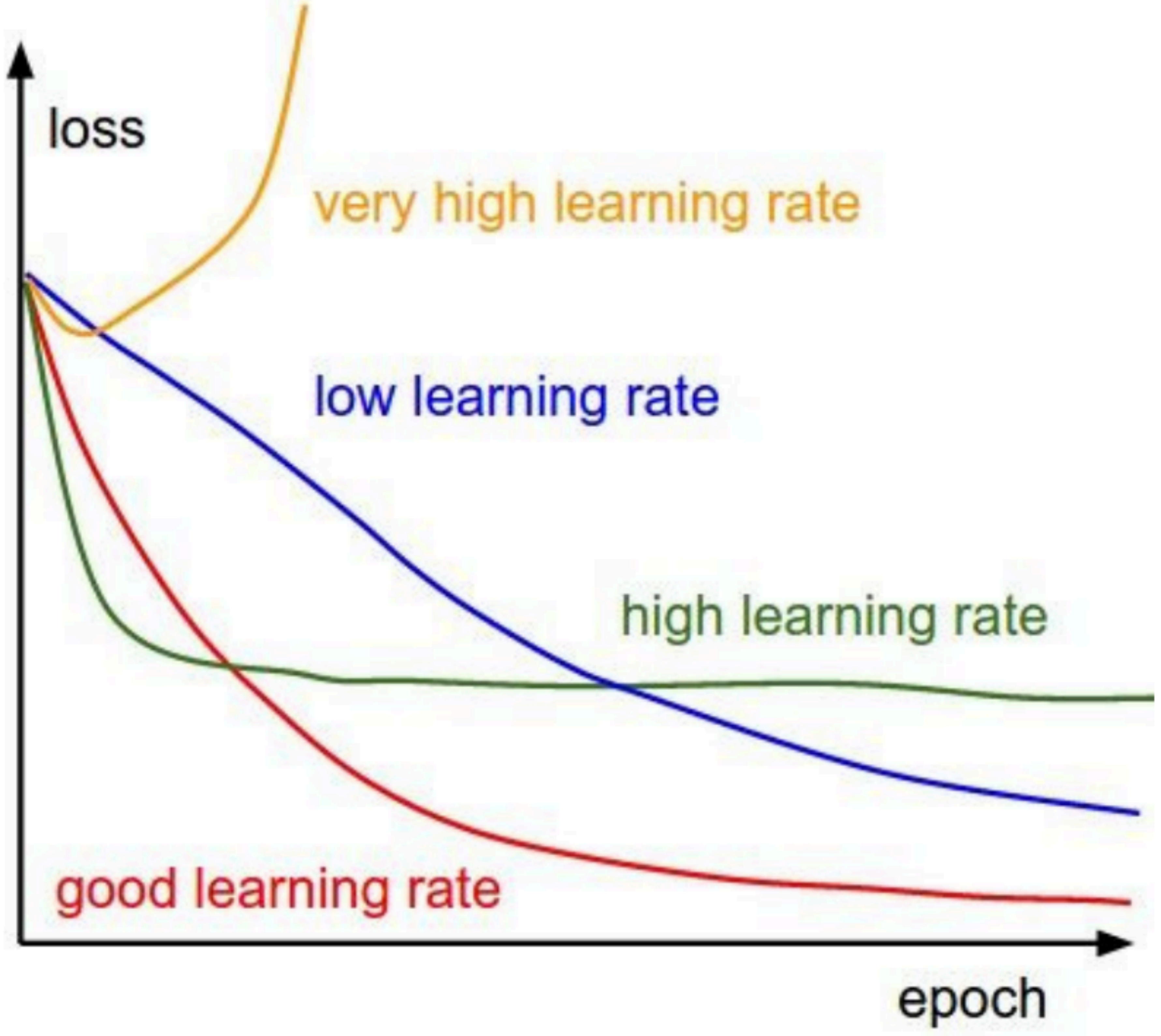
1. There is **<u>no way</u>** to write "unit tests" for NN training or inference
2. **Visualize** your data coming out of a data loader (and inference)

   (a lot of mistakes can be made in the data loader itself)
3. Learning code (and sometimes inference code) is stochastic which makes it very hard to debug. Until you are sure code is correct, **fix all the random seeds**

   (Python, NumPy, PyTorch, and Dataloader classes all have separate seeds)
4. Train with a **single example** first (always!). You should be able to obtain 0 loss, i.e., overfit. If this is not the case, there is typically error in model definition.
5. Train with a single **min-batch** next. Your loss may not be 0 at this point, but you should see convergence.
6. Use **Tensorboard** or **Weights & Biases** to keep track of experiments and visualize training & validation/testing loss and accuracy curves as you are training.

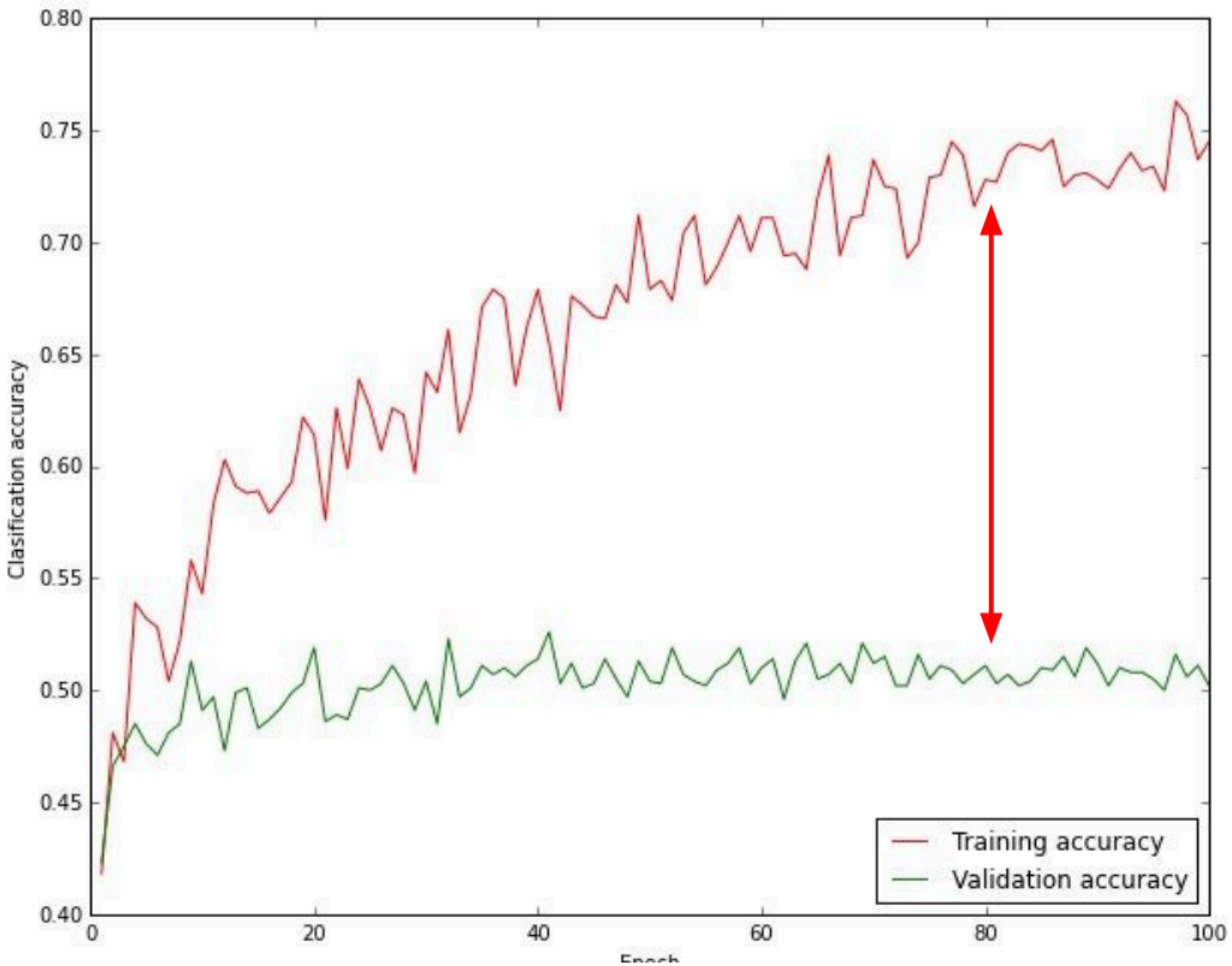
Monitoring Learning: Visualizing the (training) loss





* slide from Li, Karpathy, Johnson's **CS231n at Stanford**

Monitoring Learning: Visualizing the (training) loss



Big gap = overfitting

Solution: increase regularization

No gap = undercutting

Solution: increase model capacity

Small gap = **ideal**