Batching

- Batching data gives speedups due to more efficient matrix operations

- Need to make the computation graph process a batch at the same time

```
# input is [batch_size, num_feats]
# gold_label is [batch_size, num_classes]
def make_update(input, gold_label)
    ...
    probs = ffnn.forward(input) # [batch_size, num_classes]
    loss = torch.sum(torch.neg(torch.log(probs)).dot(gold_label))
    ...
```

- Batch sizes from 1-100 often work well

Training Basics

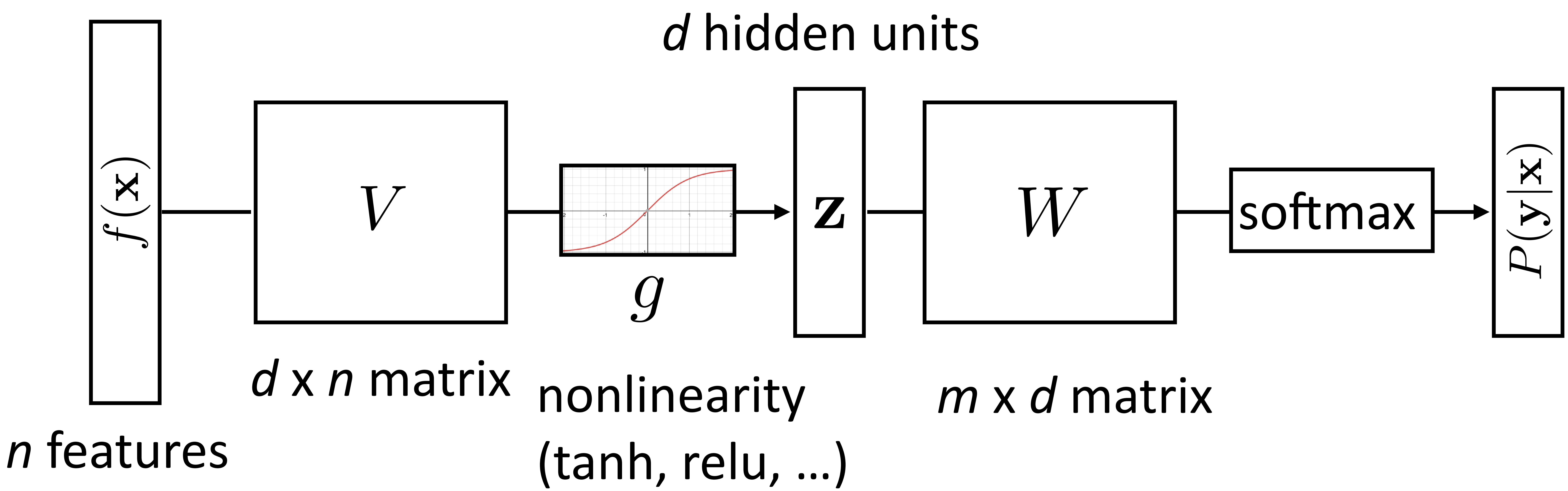
- Basic formula: compute gradients on batch, use first-order optimization method (SGD, Adagrad, etc.)

- How to initialize? How to regularize? What optimizer to use?

- This segment: some practical tricks. Take deep learning or optimization courses to understand this further

How does initialization affect learning?

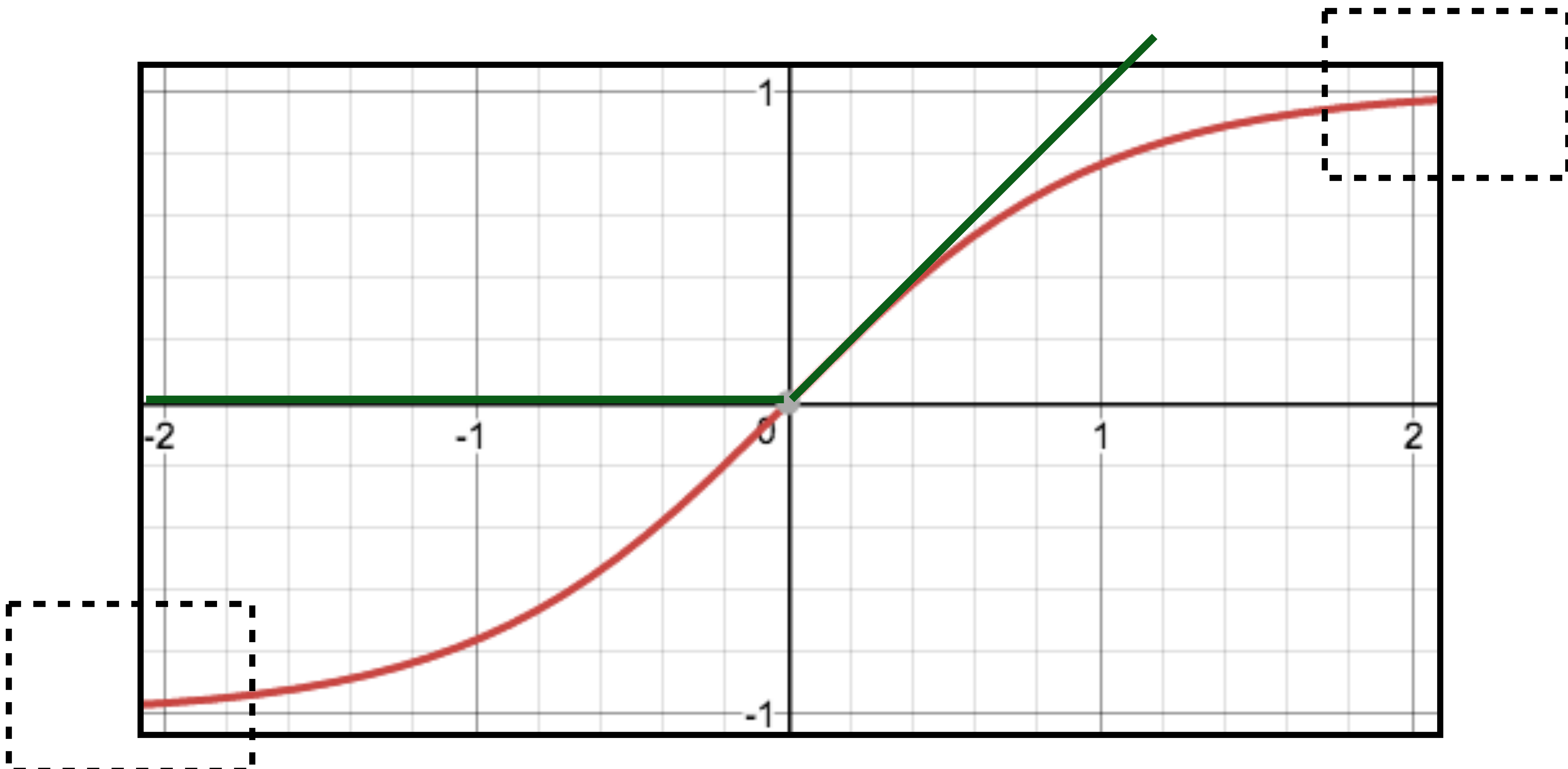
$$P(\mathbf{y}|\mathbf{x}) = \mathrm{softmax}(Wg(Vf(\mathbf{x})))$$



- How do we initialize V and W? What consequences does this have?
- *Nonconvex* problem, so initialization matters!

How does initialization affect learning?

- Nonlinear model…how does this affect things?



- If cell activations are too large in absolute value, gradients are small

- ReLU: larger dynamic range (all positive numbers), but can produce big values, can break down if everything is too negative

Initialization

1) Can't use zeroes for parameters to produce hidden layers: all values in that hidden layer are always 0 and have gradients of 0, never change

2) Initialize too large and cells are saturated

- Can do random uniform / normal initialization with appropriate scale
- Glorot initializer: $U\left[-\sqrt{\frac{6}{\text{fan-in} + \text{fan-out}}}, +\sqrt{\frac{6}{\text{fan-in} + \text{fan-out}}}\right]$
  - Want variance of inputs and gradients for each layer to be the same
- Batch normalization (Ioffe and Szegedy, 2015): periodically shift+rescale each layer to have mean 0 and variance 1 over a batch (useful if net is deep)

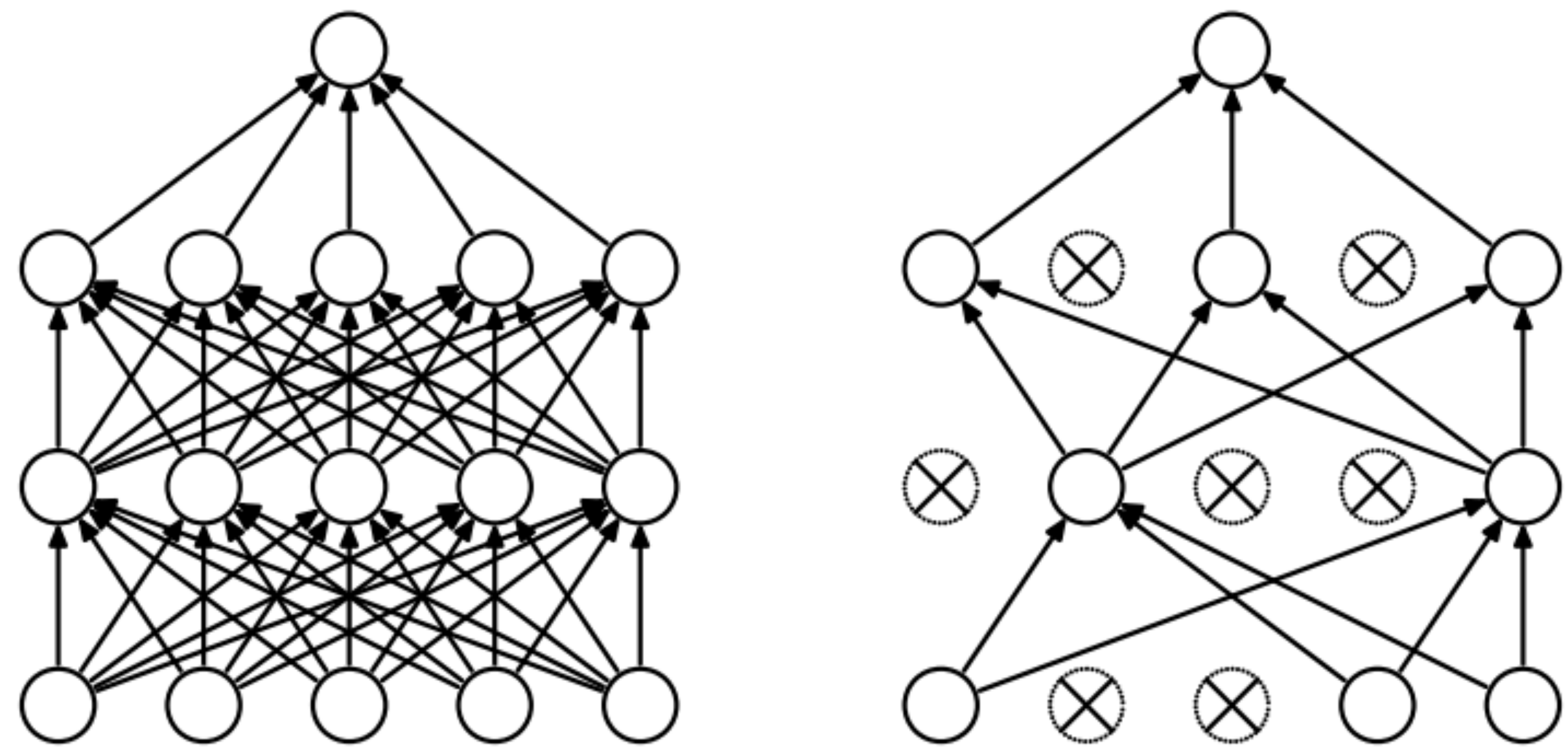
Dropout

- Probabilistically zero out parts of the network during training to prevent overfitting, use whole network at test time

- Form of stochastic regularization

- Similar to benefits of ensembling: network needs to be robust to missing signals, so it has redundancy

- One line in Pytorch/Tensorflow



(a) Standard Neural Net (b) After applying dropout.

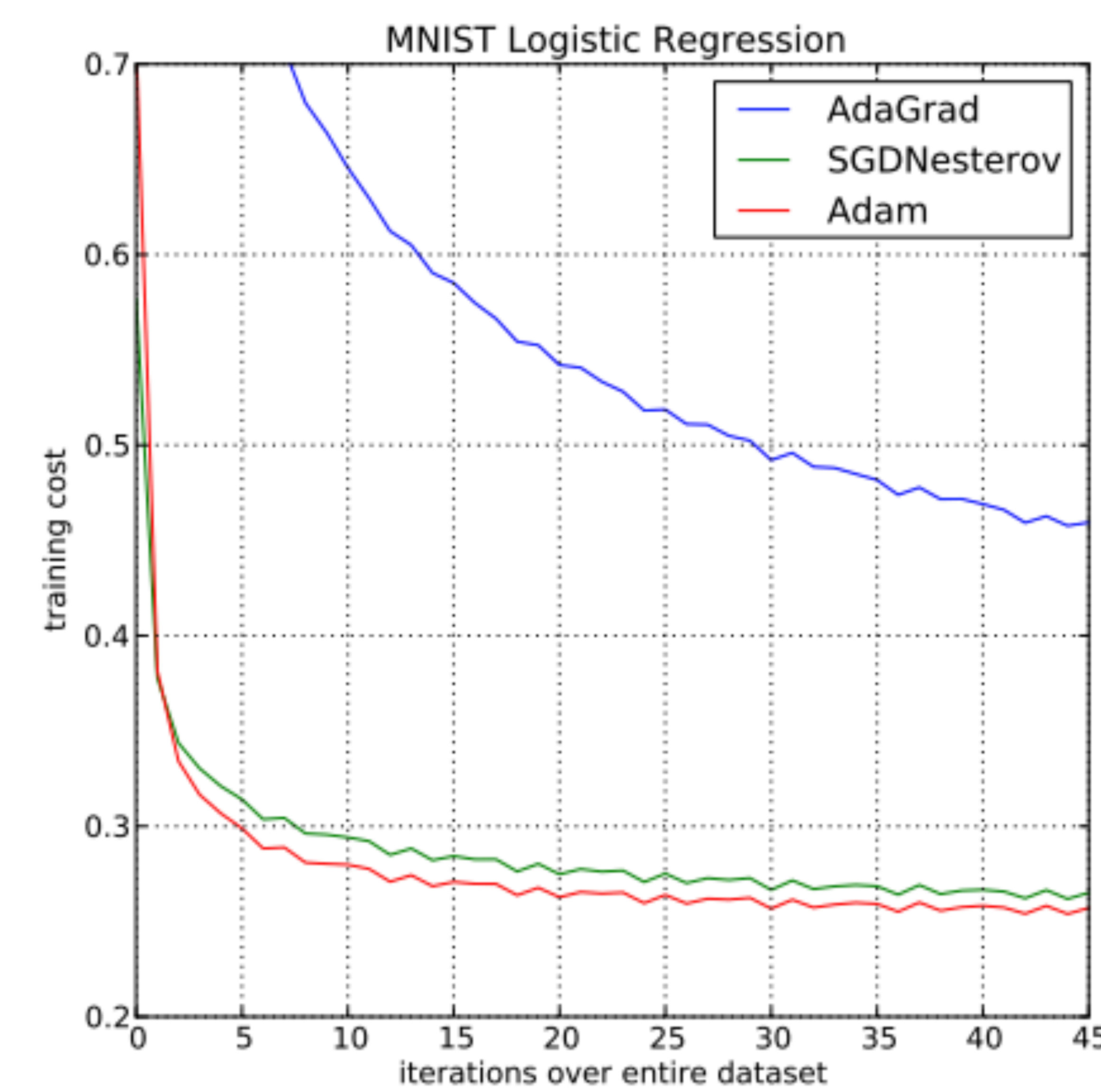
Srivastava et al. (2014)

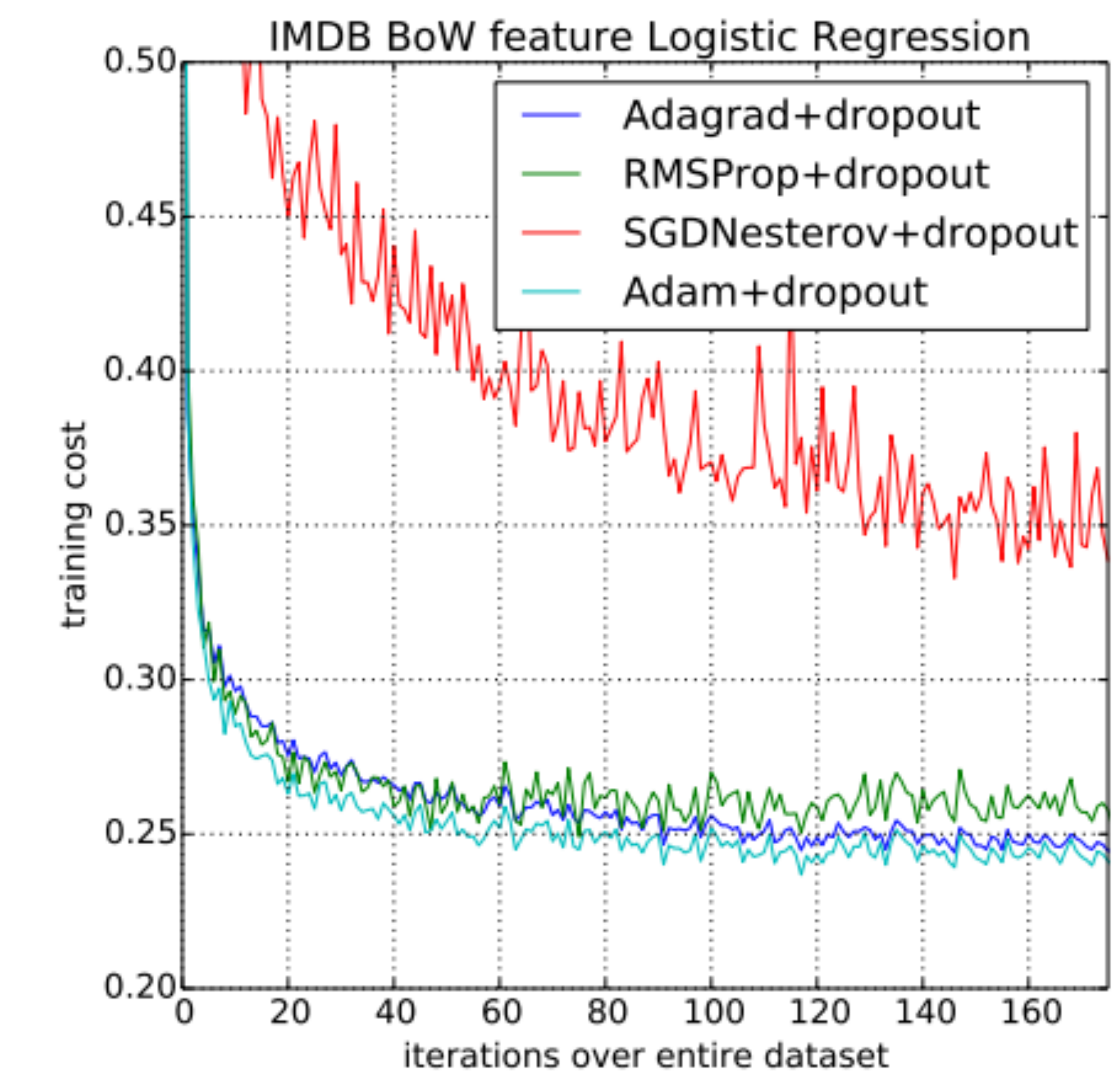
Optimizer

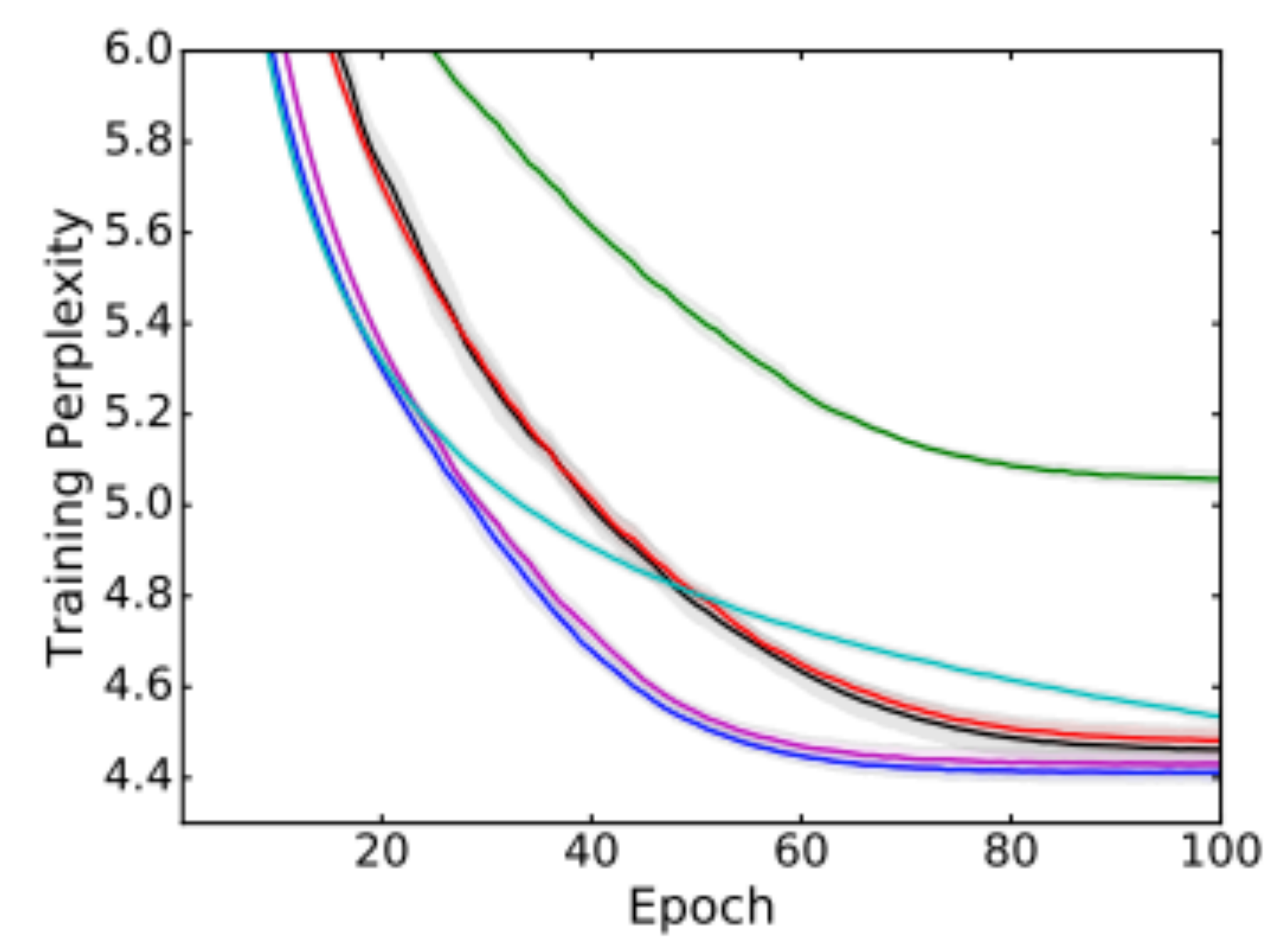
- Adam (Kingma and Ba, ICLR 2015): very widely used. Adaptive step size + momentum

- Wilson et al. NeurIPS 2017: adaptive methods can actually perform badly at test time (Adam is in pink, SGD in black)

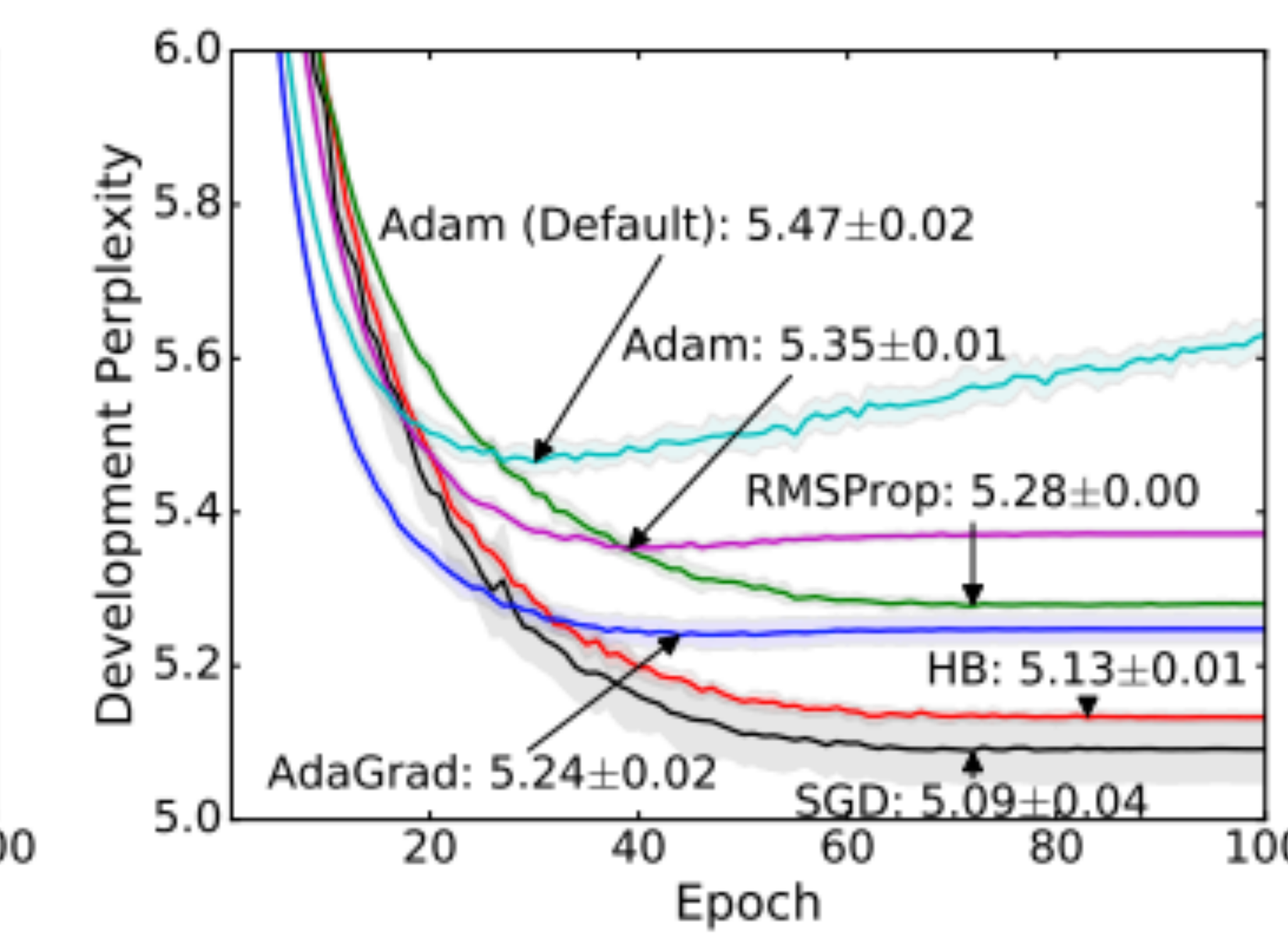
- One more trick: **gradient clipping** (set a max value for your gradients)







(e) Generative Parsing (Training Set)



(f) Generative Parsing (Development Set)