DSCI 565: MODERN CNN; RECURRENT NEURAL NETWORKS

Ke-Thia Yao

Lecture 11: 2025 October 1

Designing Convolution Network Architectures

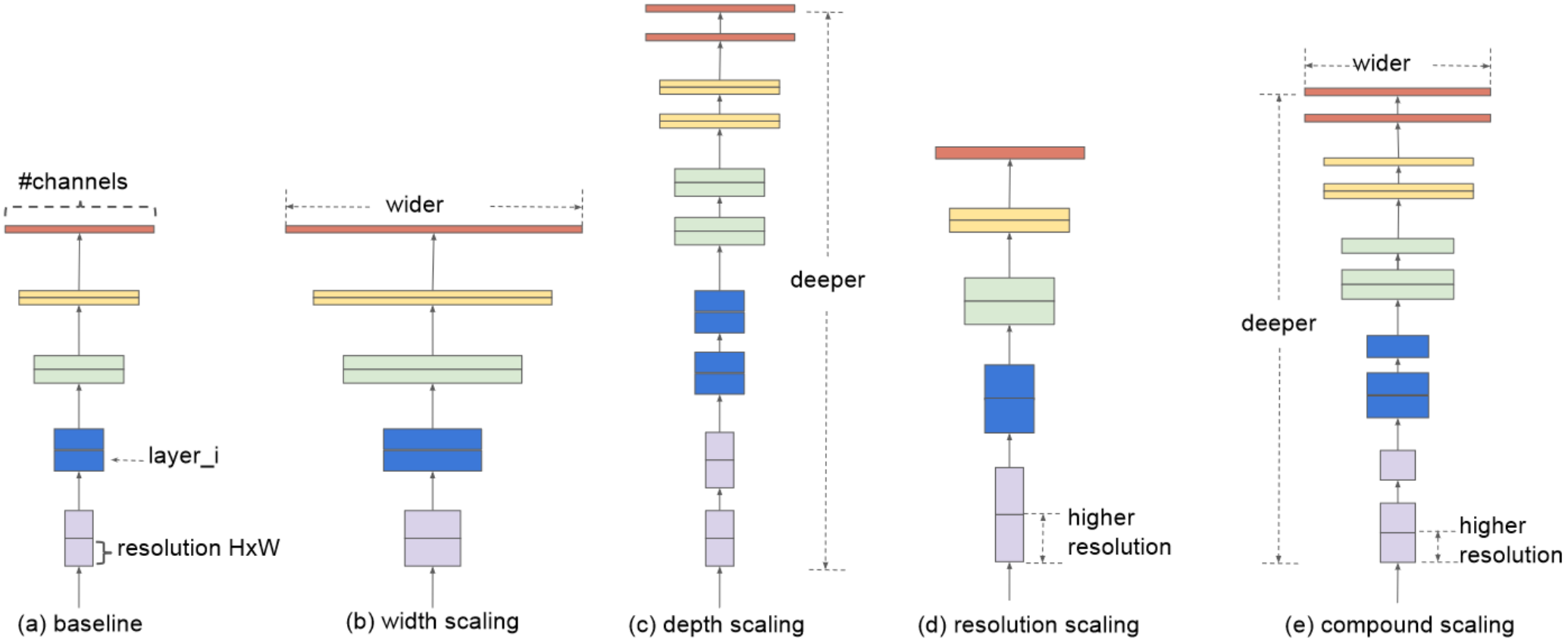
- The network design approach we have seen so far relies the intuition and ingenuity of the human engineer
  - Stacking blocks of convolutions
  - Using 3x3 convolution for deep networks
  - 1x1 convolution for local nonlinearities
  - Blocks with multiple branches
  - Global average pool to avoid large fully connect layers
  - Learning residuals

VGG , NiN
GoogLeNet (Inception)
ResNet , ResNeXt
DenseNet

- Can we find better ways to design these network architectures?

EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks (Tan & Le, 2019)

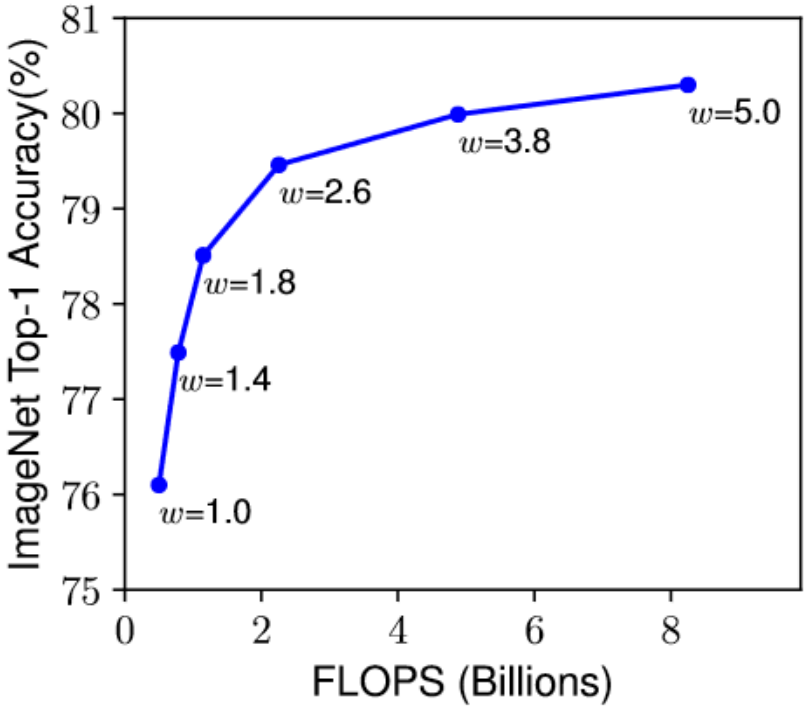
- Build network with basic blocks (from ResNet or from MobileNet)
- Experiment with scaling along multiple dimensions



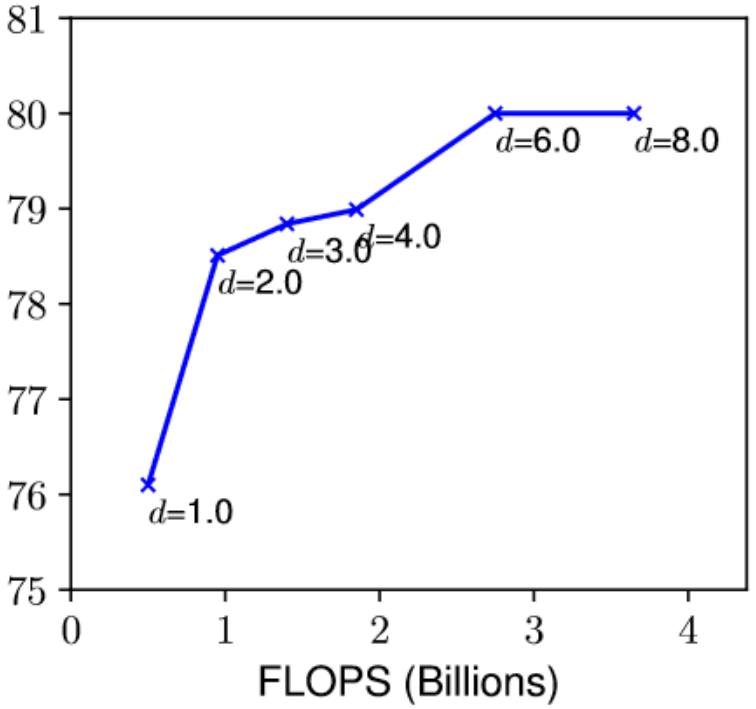
EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks (Tan & Le, 2019)

- Scaling network along individual dimensions CNN quickly saturates

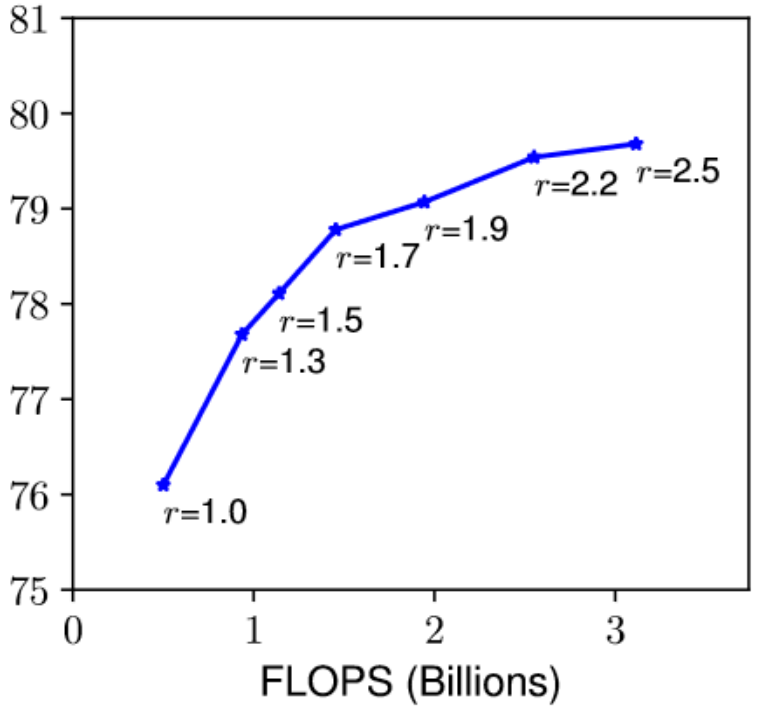
Width (# of channels)



Depth (# of layers)



Resolution (height * width)



EfficientNet

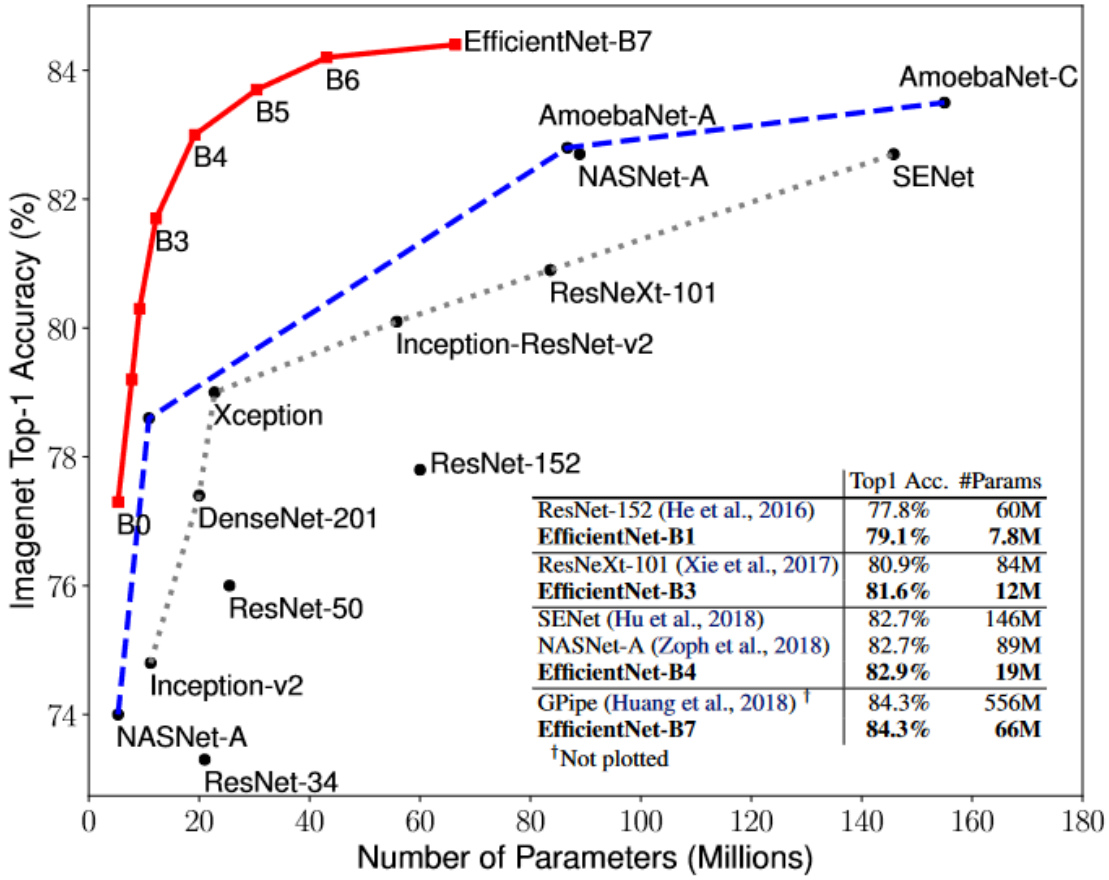
- **Compound scaling** along all 3 dims, using

$$\text{depth: } d = \alpha^{\phi}$$

$$\text{width: } w = \beta^{\phi}$$

$$\text{resolution: } r = \gamma^{\phi}$$

- Where $\alpha = 1.2, \beta = 1.1, \gamma = 1.15$ are experimentally determined using *neural architecture search (NAS)*
- And ϕ is a user specified to control resource usage.
- ϕ increases by 1, resource usage approximately doubles ($\alpha\beta^2\gamma^2 \approx 2$)



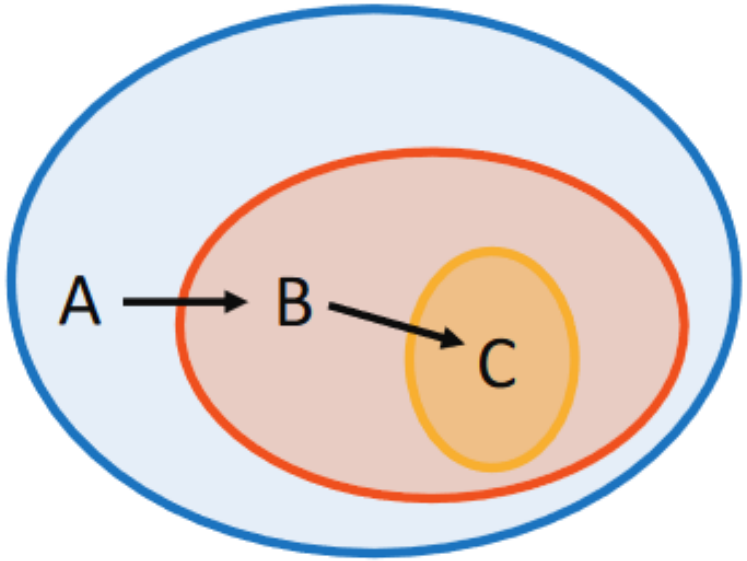
	Top1 Acc.	#Params
ResNet-152 (He et al., 2016)	77.8%	60M
EfficientNet-B1	**79.1%**	**7.8M**
ResNeXt-101 (Xie et al., 2017)	80.9%	84M
EfficientNet-B3	**81.6%**	**12M**
SENet (Hu et al., 2018)	82.7%	146M
NASNet-A (Zoph et al., 2018)	82.7%	89M
EfficientNet-B4	**82.9%**	**19M**
GPipe (Huang et al., 2018) †	84.3%	556M
EfficientNet-B7	**84.3%**	**66M**

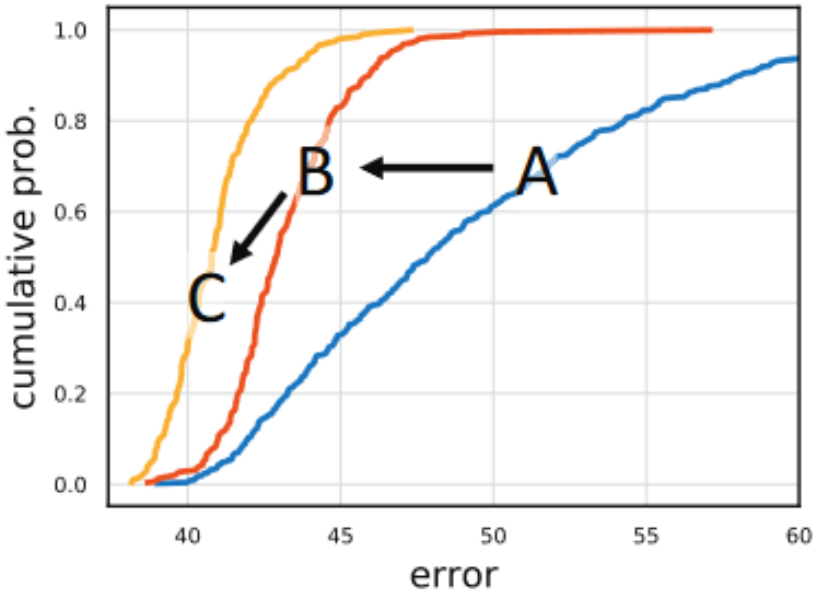
†Not plotted

Figure 1. **Model Size vs. ImageNet Accuracy.** All numbers are for single-crop, single-model. Our EfficientNets significantly outperform other ConvNets. In particular, EfficientNet-B7 achieves new state-of-the-art 84.3% top-1 accuracy but being 8.4x smaller and 6.1x faster than GPipe. EfficientNet-B1 is 7.6x smaller and 5.7x faster than ResNet-152. Details are in Table 2 and 4.

Designing Network Design Spaces (Radosavovic, et al 2020)

- *Network design space* is a parameterized set of possible architectures
- Initially, the parameters in the design spaces are relatively unconstrained
- Then, progressively simplify the design space by introducing constraints while maintaining or improving its quality





AnyNet Design Space

17-parameter design space:

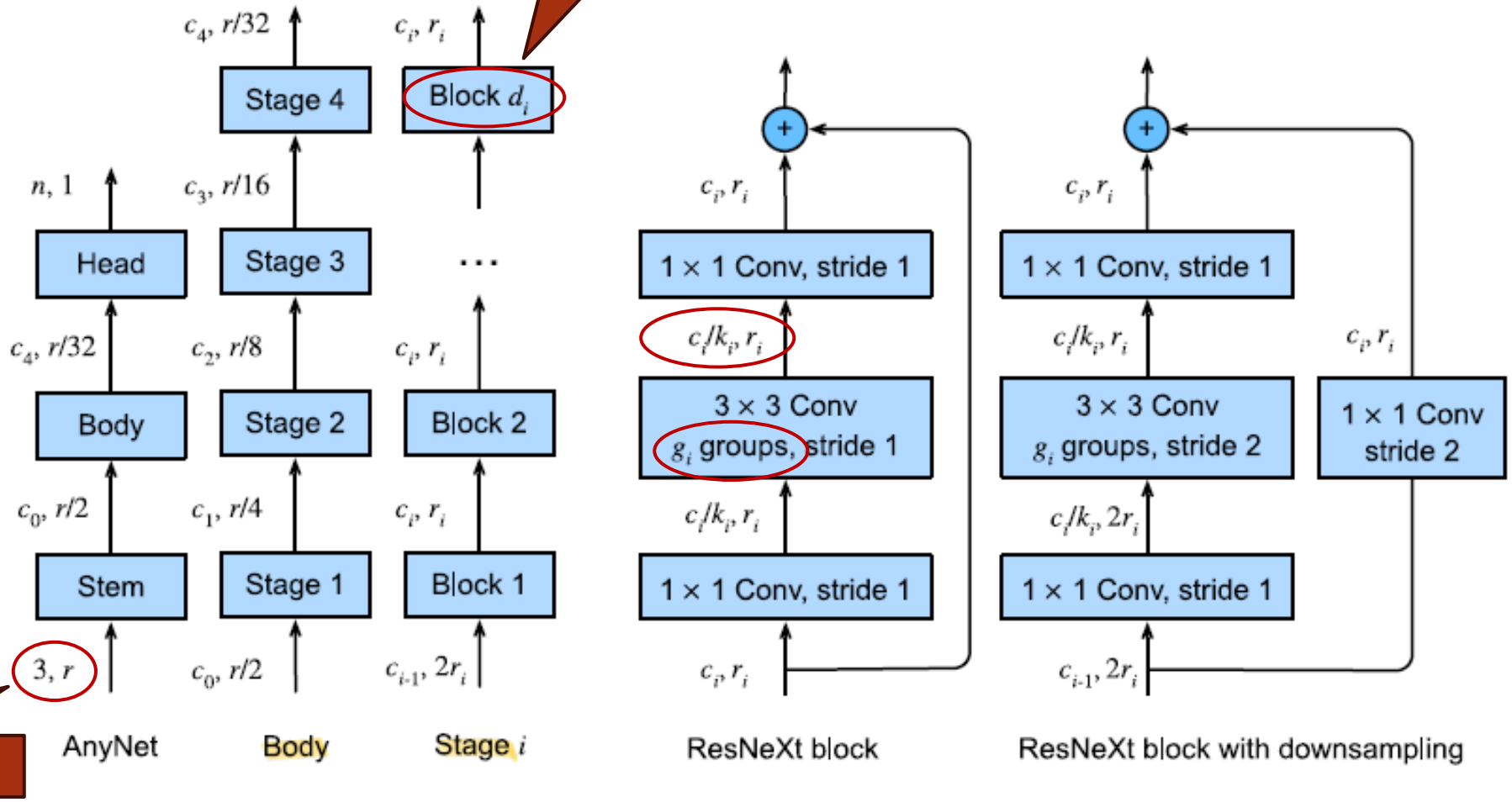
Width: $c_0, \ldots, c_4$ 5

Depth: $d_1, \ldots, d_4$ 4

Bottleneck ratios: $k_1, \ldots, k_4$ 4

Group widths: $g_1, \ldots, g_4$ 4

Depth d_i



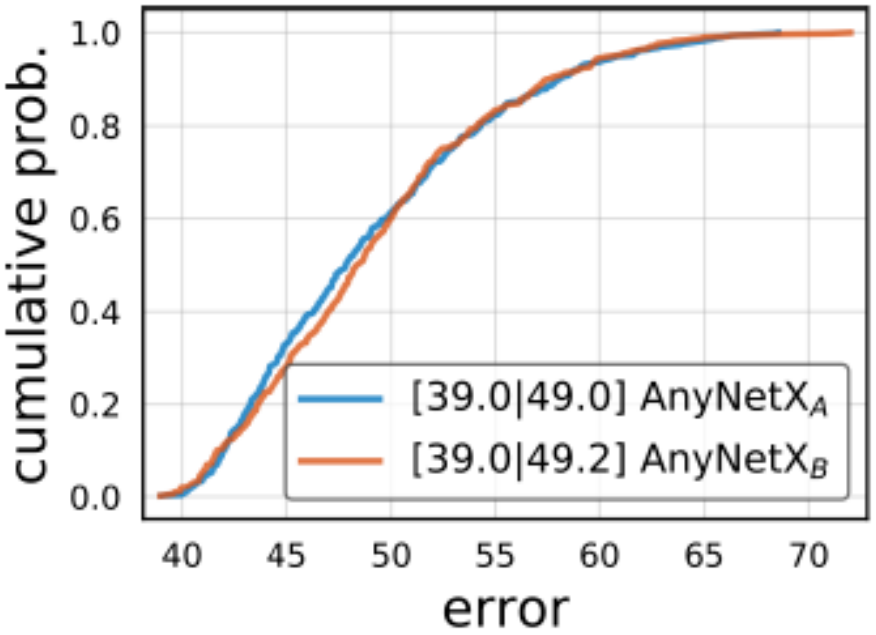
Channels (c), Resolution

AnyNet Design Space

- Although AnyNet design space has just 17 parameters, it is still too large to explore by brute-force
- Just testing 2 values per parameters implies testing $2^{17} = 131{,}072$ difference neural networks
- And, even if we test these 2^{17} networks we would not learn any generalizable design rules

Design Space Refinement

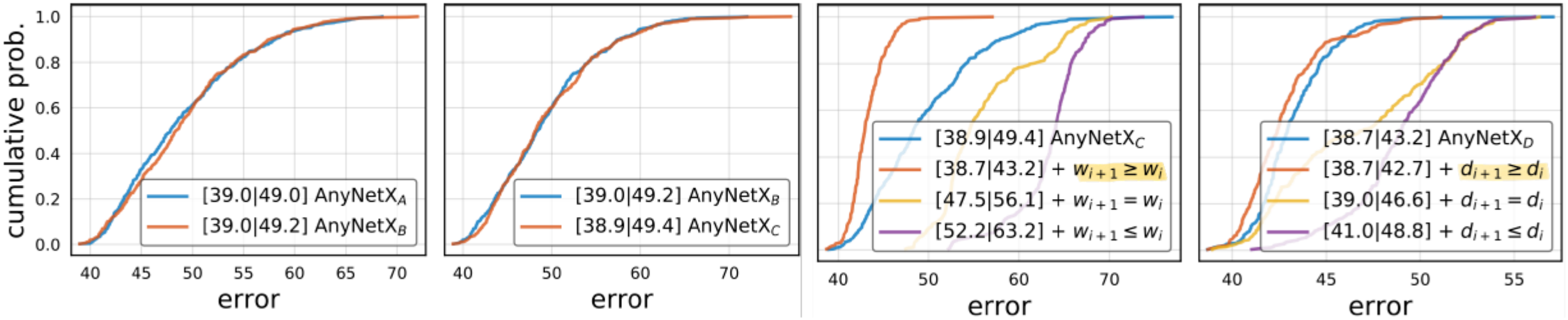
- Let AnyNet_A be the original design space 17 parameters
- Let AnyNet_B be the space where the bottleneck parameters are tied, i.e., $k_1 = k_2 = k_3 = k_4$ 14 = 17 − 4 + 1 parameters
- We randomly sample networks from AnyNet_A and AnyNet_B and test
- We discover tying bottleneck parameters has no effect on performance



Design Space Refinements

14-4+1=11

- Let AnyNet_C be the further refinement where group widths are tied
- Let AnyNet_D increase network width (channels) across stages
- Let AnyNet_E increase network depth across stages



Design Space Refinement Assumptions

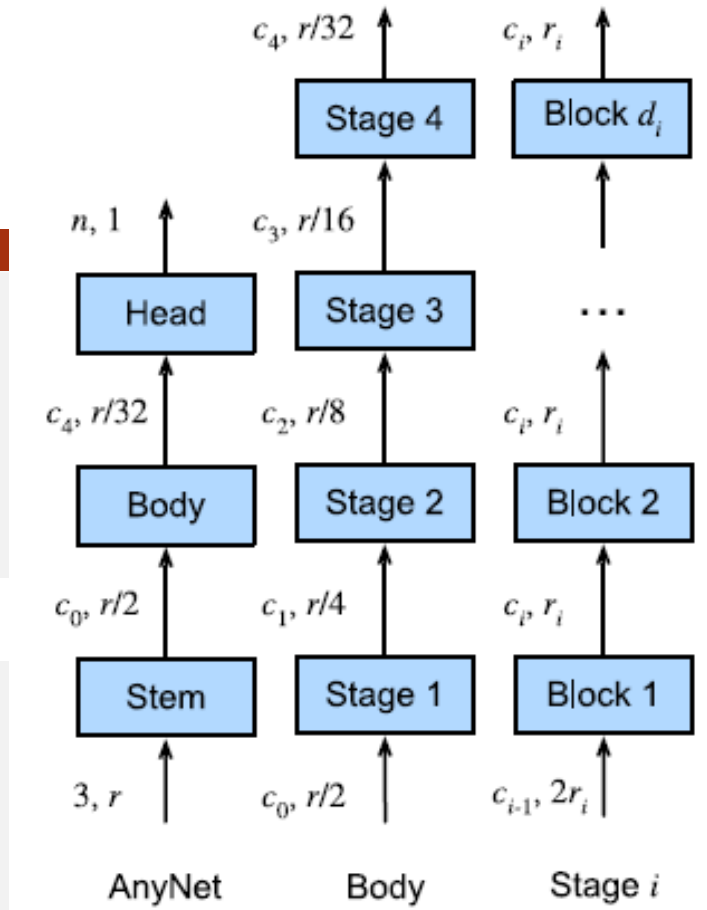
- Assume general design principles exist, so that many networks satisfying these requirements should offer good performance.
- Do not need to train network to convergence. Only need train for a few epochs to determine accuracy.
- Results obtained for smaller networks generalize to larger ones. EfficientNet uses this same assumption.
- Aspects of design can be approximately factorized, i.e., can incrementally refine design spaces

Design Space Analysis

- Resulting AnyNet_E design space shows the following design principles
  - Share the bottleneck ratio $k_i = k$ for all stages i
  - Share the group width $g_i = g$ for all stages i
  - Increase network width across stages: $\mathrm{c_i} \leq \mathrm{c_{i+1}}$
  - Increase network depth across stages: $\mathrm{d_i} \leq d_{\mathrm{i+1}}$
- Then, by looking at sample networks from AnyNet_E
  - Networks with linear increases seem to work best: $c_j \approx c_0 + c_a j$, for $c_a > 0$
  - Bottlenecks do not help : $k = 1$

AnyNet


```
class AnyNet(d2l.Classifier):
    def stem(self, num_channels):
        return nn.Sequential(
            nn.LazyConv2d(num_channels, kernel_size=3, stride=2, padding=1),
            nn.LazyBatchNorm2d(), nn.ReLU())
```

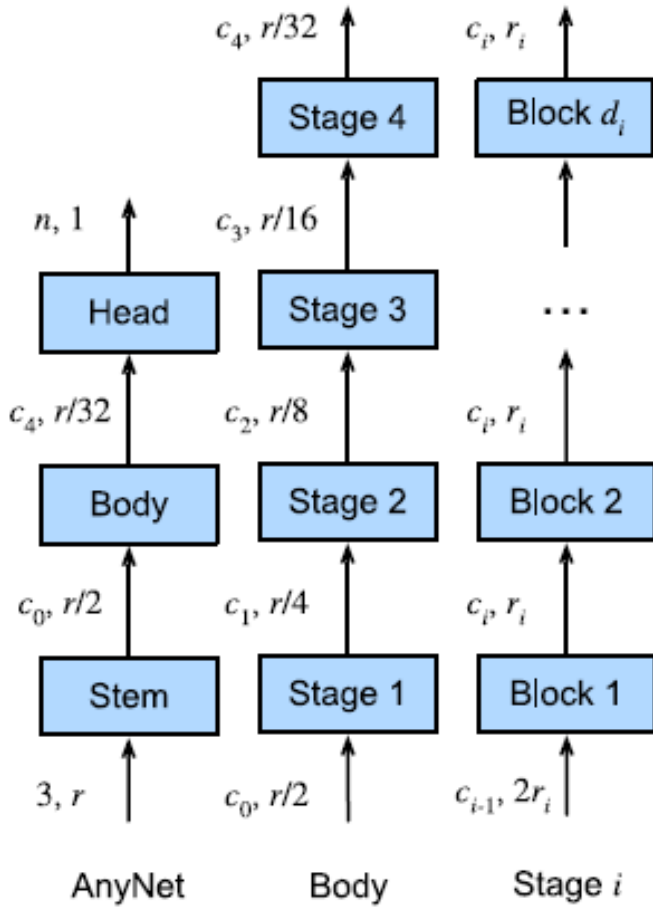



```
@d2l.add_to_class(AnyNet)
def stage(self, depth, num_channels, groups, bot_mul):
    blk = []
    for i in range(depth):
        if i == 0:
            blk.append(d2l.ResNeXtBlock(num_channels, groups, bot_mul,
                use_1x1conv=True, strides=2))
        else:
            blk.append(d2l.ResNeXtBlock(num_channels, groups, bot_mul))
    return nn.Sequential(*blk)
```

Stage

AnyNet


```
@d2l.add_to_class(AnyNet)
def __init__(self, arch, stem_channels, lr=0.1, num_classes=10):
    super(AnyNet, self).__init__()
    self.save_hyperparameters()
    self.net = nn.Sequential(self.stem(stem_channels))
    for i, s in enumerate(arch):
        self.net.add_module(f'stage{i+1}', self.stage(*s))
    self.net.add_module('head', nn.Sequential(
        nn.AdaptiveAvgPool2d((1, 1)), nn.Flatten(),
        nn.LazyLinear(num_classes)))
    self.net.apply(d2l.init_cnn)
```



RegNetX

Inherits from AnyNet

Constant groups across stages

No bottleneck

Increasing depth: 4 to 6
Increasing channels: 6 to 80

```
class RegNetX32(AnyNet):
    def __init__(self, lr=0.1, num_classes=10):
        stem_channels, groups, bot_mul = 32, 16, 1
        depths, channels = (4, 6), (32, 80)
        super().__init__(
            ((depths[0], channels[0], groups, bot_mul),
             (depths[1], channels[1], groups, bot_mul)),
            stem_channels, lr, num_classes)
```

```
RegNetX32().layer_summary((1, 1, 96, 96))
```

```
Sequential output shape:     torch.Size([1, 32, 48, 48])
Sequential output shape:     torch.Size([1, 32, 24, 24])
Sequential output shape:     torch.Size([1, 80, 12, 12])
Sequential output shape:     torch.Size([1, 10])
```

Vs EfficientNet

Compared to EfficientNet

With the same FLOPS

Approx same top-1 error

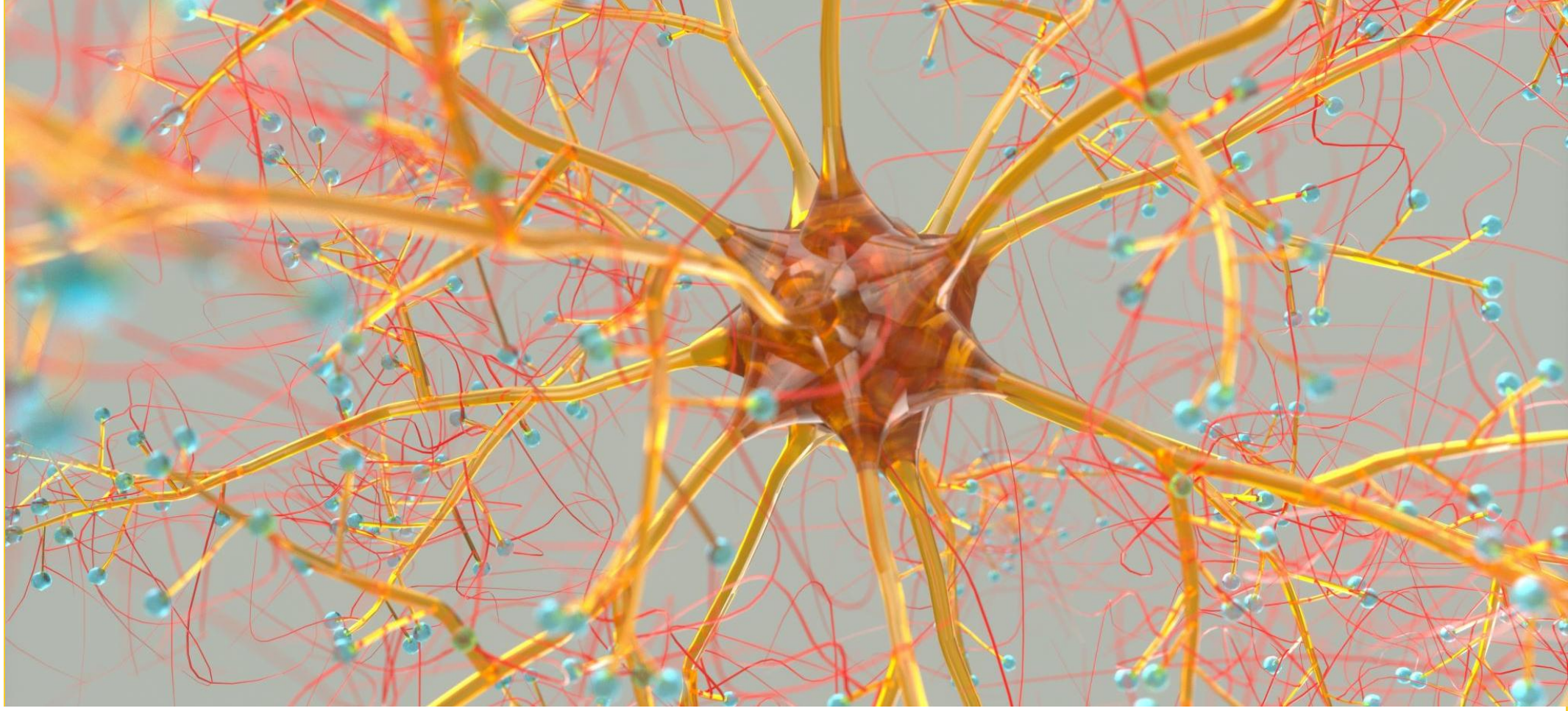
Trains and Runs faster

RegNetY adds squeeze-and-excitation

EfficientNet
RegNet

	flops (B)	params (M)	acts (M)	batch size	infer (ms)	train (hr)	top-1 error ours±std [orig]
EFFICIENTNET-B0	0.4	5.3	6.7	256	34	11.7	**24.9**±0.03 [23.7]
REGNETY-400MF	0.4	4.3	**3.9**	1024	19	5.1	25.9±0.16
EFFICIENTNET-B1	0.7	7.8	10.9	256	52	15.6	**24.1**±0.16 [21.2]
REGNETY-600MF	0.6	6.1	**4.3**	1024	19	5.2	24.5±0.07
EFFICIENTNET-B2	1.0	9.2	13.8	256	68	18.4	**23.4**±0.06 [20.2]
REGNETY-800MF	0.8	6.3	**5.2**	1024	22	6.0	23.7±0.03
EFFICIENTNET-B3	1.8	12.0	23.8	256	114	32.1	22.5±0.05 [18.9]
REGNETY-1.6GF	1.6	11.2	**8.0**	1024	39	10.1	**22.0**±0.08
EFFICIENTNET-B4	4.2	19.0	48.5	128	240	65.1	21.2±0.06 [17.4]
REGNETY-4.0GF	4.0	20.6	**12.3**	512	68	16.8	**20.6**±0.08
EFFICIENTNET-B5	9.9	30.0	98.9	64	504	135.1	21.5±0.11 [16.7]
REGNETY-8.0GF	8.0	39.2	**18.0**	512	113	28.1	**20.1**±0.09

Table 4. **EFFICIENTNET comparisons** using our standard training schedule. Under *comparable training settings*, REGNETY outperforms EFFICIENTNET for most flop regimes. Moreover, REGNET models are considerably faster, *e.g.*, REGNETX-F8000 is about $5\times$ *faster* than EFFICIENTNET-B5. Note that originally reported errors for EFFICIENTNET (shown grayed out), are much lower but use longer and enhanced training schedules, see Table 7.

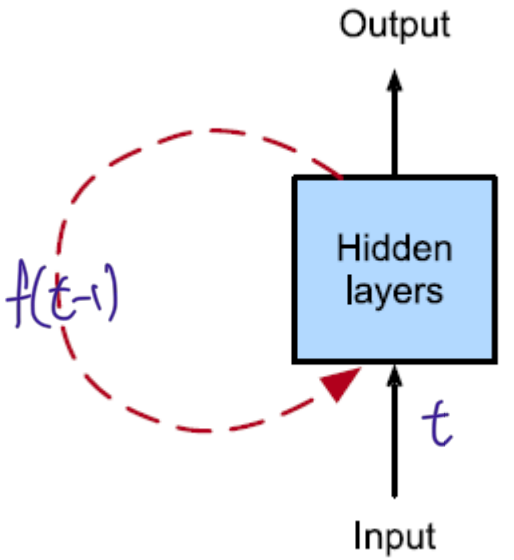


17

Recurrent Neural Networks

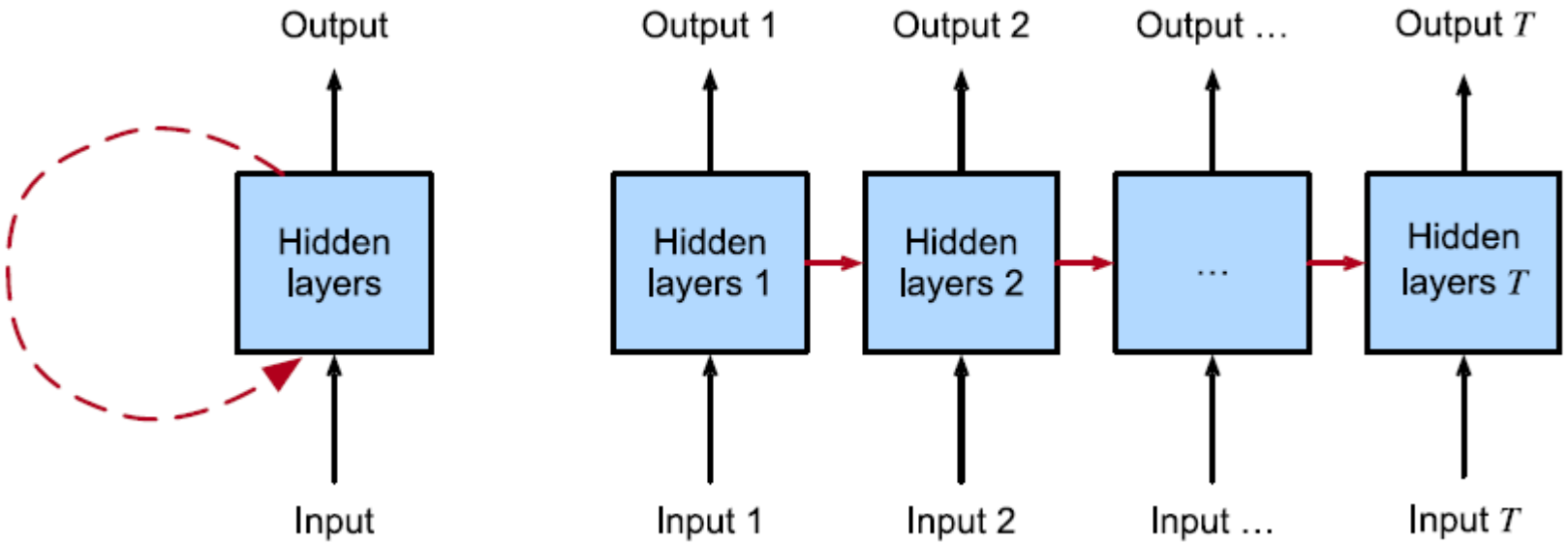
Recurrent Neural Networks

- So far this semester we have look at tabular and image data types
- Sequential data is another major data type
  - Next item in the sequence depends on the previous items
  - Varying length
- Timeseries, text, audio, video data are instances of this data type
- Recurrent neural networks are deep learning models that capture the dynamics of sequences via recurrent connections
- Recurrent network has **recurrent** connections, i.e,. cycles in the network



Unrolling Recurrent Network Across Time

- A way to think about recurrent network is to unroll it across time
- The current connection points to itself in the next time step
- All of the unrolled versions have the **same** parameters



Very Brief History of RNN

- RNN has been around for a long time
- The Ising model of particle spins from physics can be thought of as a type of recurrent neural network
- Hopfield networks and Boltzmann machines were studied in the 1980s
- Long Short-Term Memory (LSTM) networks was developed in 1990s
- Became popular in the 2010's for achieving state-of-the-art results in handwriting recognition, speech recognition, machine translation, …
- Attention mechanism was first developed for RNNs

Working with Sequences

- Represent a sequence as an ordered list of feature vectors:

$$\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T$$

- Where $\mathbf{x}_t \in \mathcal{R}^d$

d=1

- Some sequence datasets is a single long sequence. To learn we need to sample and create subsequences
- Other datasets are collections of sequences, such as documents or event sequences
- We assume each sequence is independently sampled from some fixed underlying distribution

Some Sequence Machine Learning Tasks

- Predict the next vector(s) in the sequence
- Classify a sequence.
  - Output a target y, e.g., positive/negative restaurant review
- Sequence-to-sequence translation
  - Aligned: step-to-step correspondence between sequences, e.g., parts of speech tagging
  - Unaligned: length of the sequences differ, e.g., natural language translation

Autoregressive Models

AROMA

AR

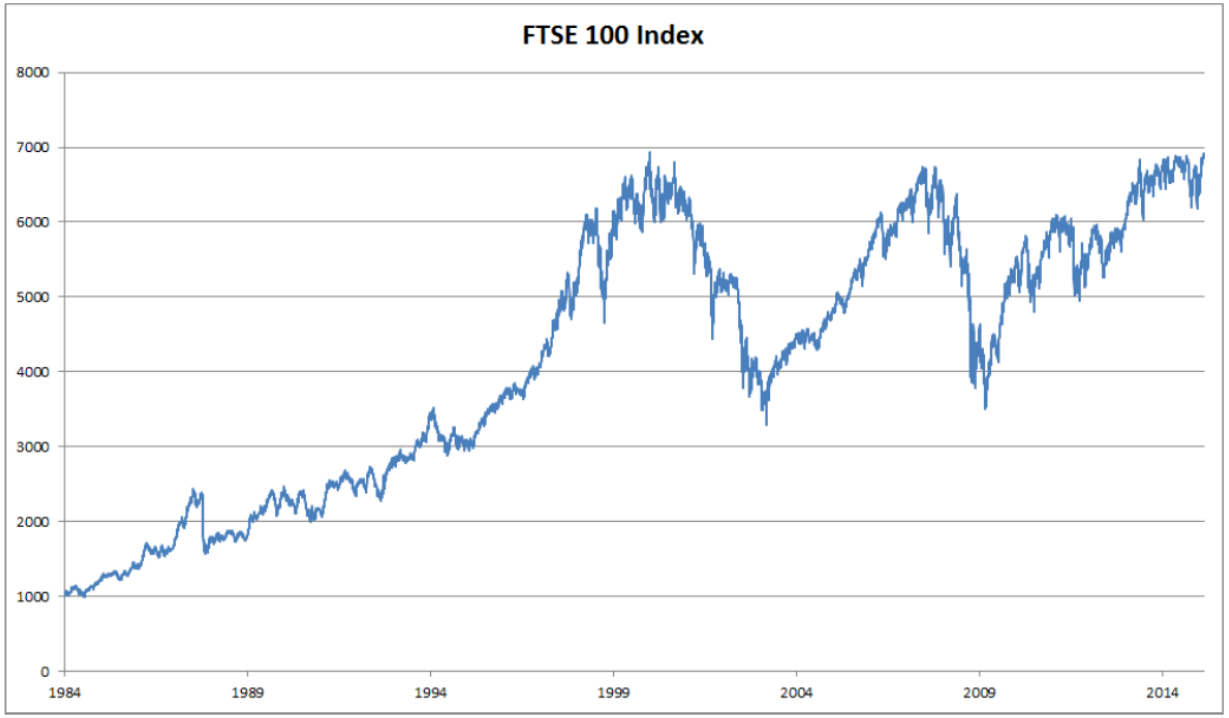
- Predict the next step, by looking at just the history of the timeseries

$$P(x_t \mid x_{t-1}, \ldots, x_1)$$

- E.g., we could try to generate a ML model to estimate

$$\mathbb{E}[(x_t \mid x_{t-1}, \ldots, x_1)]$$

- But the number of inputs change with each timestep

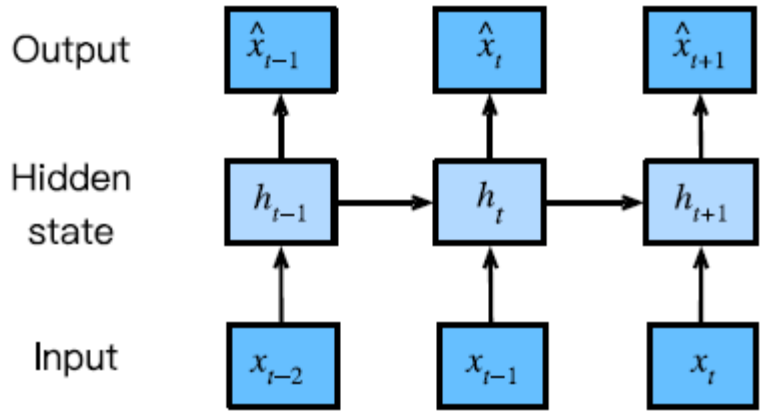


Dealing with Variable Length Inputs

- One way is to assume the steps too far back in the sequence do not influence the next time step
  - We pick a window size τ, and consider only $x_{t-1}, \ldots, x_{t-\tau}$

$X_{t-1}, \cdots X_{t-\tau}$ only influence

- Another ways is to maintain a summary h_t, i.e., $P(x_t|h_t, x_{t-1})$
  - This is called latent autoregressive model



Sequence Models

- Sequence models estimate the probability of entire sequences
- Sometimes they are called language models, because they are used in NLP to estimate the probability of sentences
- Estimating probability of a sequence can be decomposed to estimating probability of the next element in the sequence, using the conditional probability products rule

$$P(x_1, \ldots, x_T) = P(x_1) \prod_{t=2}^{T} P(x_t \mid x_{t-1}, \ldots, x_1)$$

Markov Models

- Markov condition: the future is conditionally independent of the past, given the recent history $x_{t-1}, \ldots, x_{t-\tau}$
- If $\tau = k$, then we have a k^{th}-order Markov model
- For $\tau = 1$, a first-order Markov model, the joint probability simplifies to

$$P(x_1, \ldots, x_T) = P(x_1) \prod_{t=2}^{T} P(x_t \mid x_{t-1})$$

$$P(X_t \mid X_{t-1}) = P(X_t \mid X_{t-1} X_{t-2} \cdots X_1)$$

Notebooks

- chapter_recurrent-neural-networks/sequence.ipynb
  - Use linear regression with Markov assumption for prediction
  - Result is reasonable for predication the next 1step
  - But fails for multi-steps
- chapter_recurrent-neural-networks/text-sequence.ipynb
  - Converting raw text into sequence data
  - Unigram, bigram, trigram
  - Zipf's Law

Learning Language Models

- Language models estimate the probability of whole sequences, e.g., sentences:

$$P(x_1, x_2, \ldots, x_T)$$

- Using definition of conditional probability:

$$P(x_1, x_2, \ldots, x_T) = \prod_{t=1}^{T} P(x_t \mid x_1, \ldots, x_{t-1})$$

Markov Model

- Markov model
  full Markov model
  $$P(\text{deep}, \text{learning}, \text{is}, \text{fun}) = P(\text{deep})P(\text{learning}|\text{deep})P(\text{is}|\text{deep}, \text{learning})\ P(\text{fun}|\text{deep}, \text{learning}, \text{is})$$
- 1st order Markov model
  $$P(\text{deep}, \text{learning}, \text{is}, \text{fun}) = P(\text{deep})P(\text{learning}|\text{deep})P(\text{is}|\text{deep})P(\text{fun}|\text{is})$$
  learning
- Give a corpus, we can estimate $\hat{P}(\text{deep})$ by counting the occurrences of the word "deep" divided by the number of words in the corpus
- Or, better by counting the frequency of the work "deep" occurring as the first word of the sentence

Word Frequency

- To estimate: $\hat{P}(\text{learning}|\text{deep}) = \frac{n(\text{deep,learning})}{n(\text{deep})}$,

  where $n(x)$ and $n(x, x')$ is the number of occurrences of singletons and word pairs

Problem

- We may encounter word pairs that are not in the corpus, but they are plausible word pairs
- A zero probability in one factor, causes the entire sentence to have zero probability

Laplace Smoothing

- Add a small constant to all counts

$$\hat{P}(x) = \frac{n(x) + \epsilon_1/m}{n + \epsilon_1},$$

$$\hat{P}(x' \mid x) = \frac{n(x, x') + \epsilon_2 \hat{P}(x')}{n(x) + \epsilon_2},$$

$$\hat{P}(x'' \mid x, x') = \frac{n(x, x', x'') + \epsilon_3 \hat{P}(x'')}{n(x, x') + \epsilon_3}$$

$\lim_{n \to \infty} \hat{P}(x) = \frac{1}{m}$

  - Where $\epsilon_1, \epsilon_2, \epsilon_3$ are hyperparameters and m is size of vocabulary
  - If $\epsilon_1 = 0$ then counts; If $\epsilon_1 \to \infty$ then equal probability
- Problem: Not all unseen pairs and triples are equally likely. Need more complex models then word frequencies

Perplexity

- Perplexity measure the quality of a language model
- A high-quality language model should be able to predict the next word with high accuracy
- Consider predicating the next words of "It is raining":

1. "It is raining outside"
2. "It is raining banana tree"
3. "It is raining piouw;kcj pwepoiut"

Perplexity and Cross-Entropy Loss

- Cross-entropy loss average over n tokens of a sequence

$$\frac{1}{n}\sum_{t=1}^{n} -\log P(x_t \mid x_{t-1}, \ldots, x_1),$$

- Where P is probability returned by the model, and x_t is the actual observed token
  - If $P(\text{learning}|\text{deep}) = 1.0$, then $-\log 1.0 = 0$ bit
  - If $P(\text{learning}|\text{deep}) = 0.25$, then $-\log 0.25 = 1.336$ nats or 2 bits
  - If $P(\text{learning}|\text{deep}) = 0.0$, then $-\log 0.0 = \infty$ bit
- Perplexity is exponential of cross-entropy loss:

$$\exp\left(-\frac{1}{n}\sum_{t=1}^{n} \log P(x_t \mid x_{t-1}, \ldots, x_1)\right)$$

Perplexity

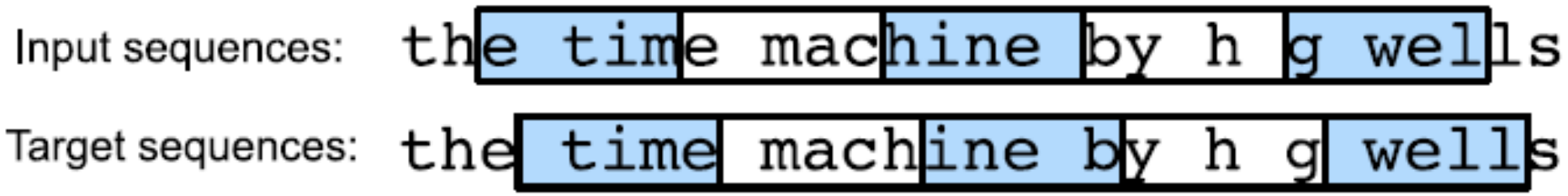
- The model always predicts the next token correctly with probability 1
  - Perplexity is 1 $\exp(0)=1$
- The model predicts any next token correctly with probability 0
  - Perplexity is ∞ $\exp(\infty)=\infty$
- If the model random guess a word from the vocabulary of size v
  - Perplexity is v $\exp\left(-\frac{1}{n}\sum_{t=1}^{n}\log\frac{1}{n}\right)=n$
- If the model on average predicts the next token correctly with prob 0.25
  - Perplexity is 4

Partitioning Sequences

- Suppose the corpus is a sequence of T tokens
- To get multiple instances, partition into subsequences of n tokens
- To introduce randomness, we can skip the first $d \in [0, n)$ tokens
- We can get $m = \left\lfloor \frac{T-d}{n} \right\rfloor$ non-overlapping subsequences
- These are the input sequences
- For prediction, the target sequences shifted by one token by skipping the first $d + 1$ tokens

Example Partitioning Sequences

- Suppose
  - Tokens are characters
  - Subsequences are $n = 5$ tokens
  - Skip $d = 2$ tokens



Notebook

- chapter_recurrent-neural-networks/language-model.ipynb