BST 261: Data Science II
Lecture 14

Text Generation,
Deep Dream,
Neural Style Transfer,
Advanced Network Architectures

Heather Mattie
Harvard T.H. Chan School of Public Health
Spring 2 2020

Recipe of the Day!

[Watermelon Salad with Feta and Cucumber]

Paper Presentations

Distilling the Knowledge in a Neural Network

Geoffrey Hinton, Oriol Vinyals, Jeff Dean

Presenter: Kexin Huang

Motivation

“Many insects have a **larval form** that is optimized for extracting energy and nutrients from the environment and ***a completely different adult form*** that is optimized for the very different requirements of traveling and reproduction.”

Analogy

“In large-scale machine learning, …, **training** must extract structure from very large, highly redundant datasets but it does not need to operate in real time and it can use a huge amount of computation. … ***Deployment*** to a large number of users, however, has much more stringent requirements on latency and computational resources.”

Question

However, small models usually do not have good performance as large model. How do we transfer knowledge learned from large model to small model to improve small model’s performance?

Large Model

Knowledge Transfer

Small Model

How do we define "***knowledge***" in a machine learning model?

- a learned **mapping** from input vectors (x) to output vectors (logits z)

therefore

For a small model to share the same knowledge with a large model: we want the logits from the small model to **match** the logits from the large model.

to summarize

Knowledge Transfer = **Match the logits**!

Typical Workflow:

$$\mathbf{x}$$

(input)

$$z_i = \text{model}(\mathbf{x})_i$$

(logits)

$$q_i = \frac{exp(z_i)}{\sum_j exp(z_j)}$$

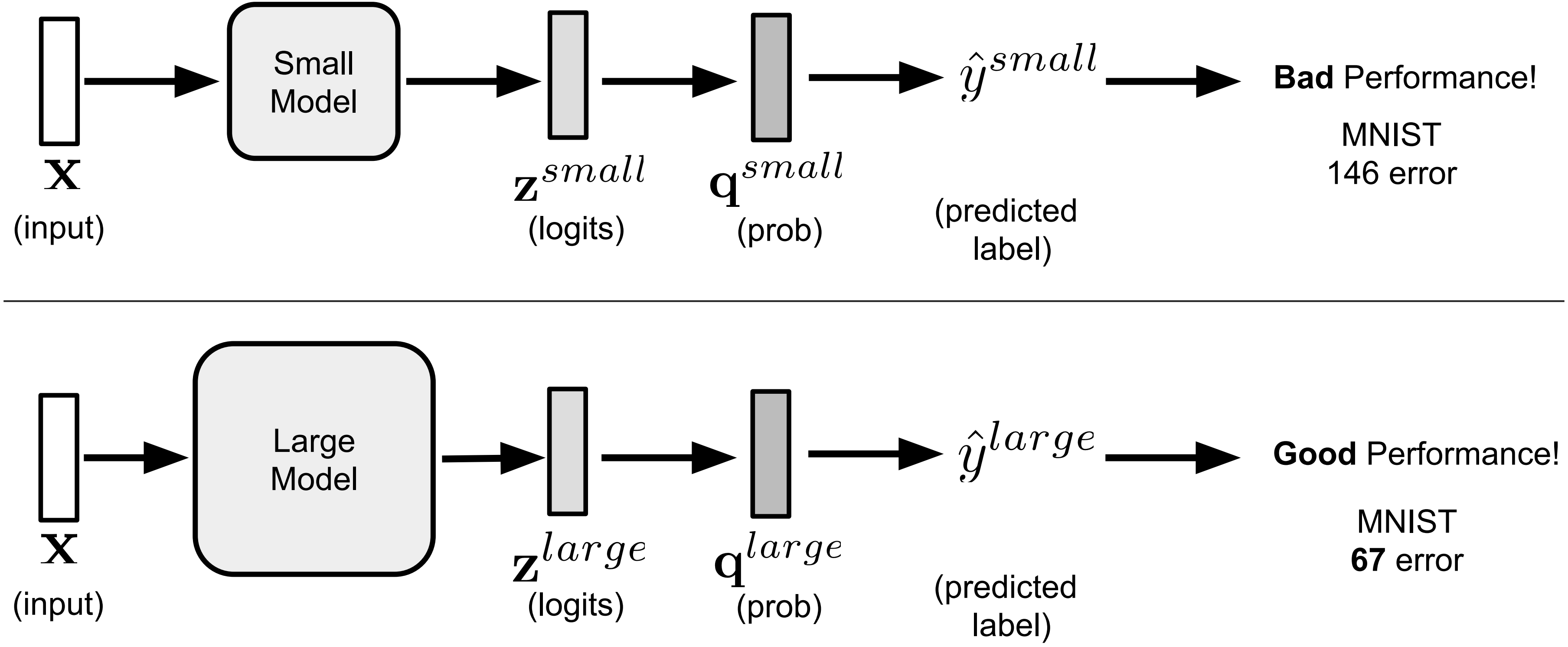
(prob)

$$\hat{y}$$

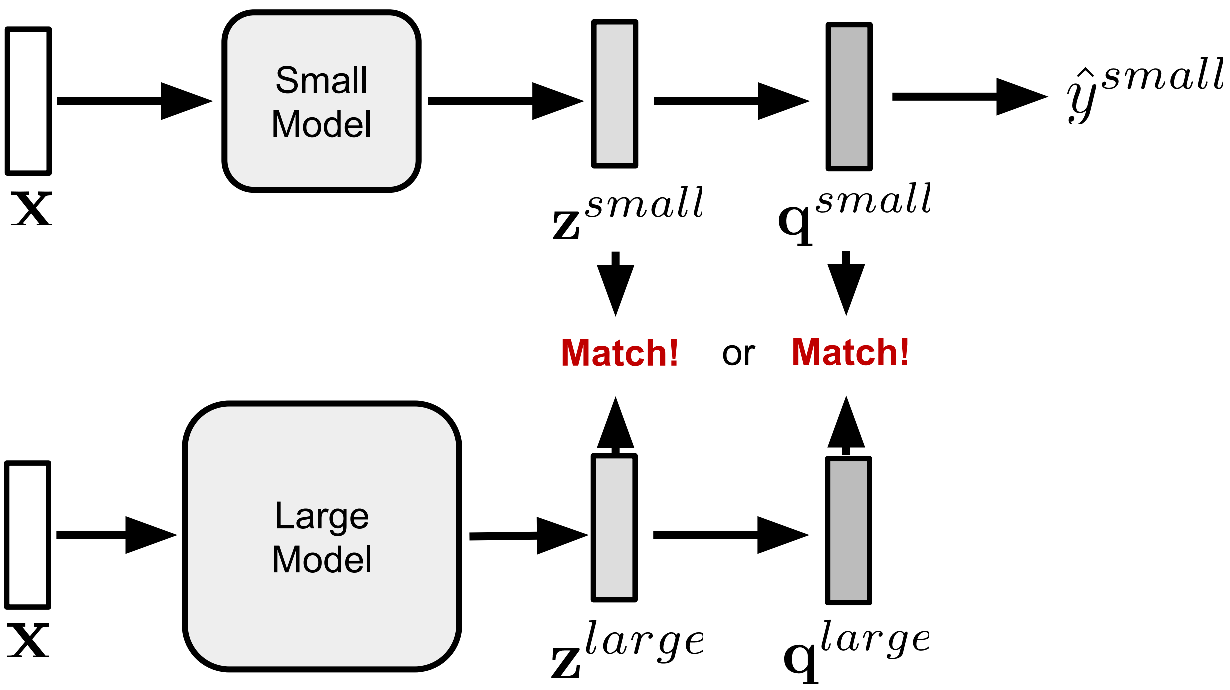
(predicted label)

Cross-Entropy Loss

Without any knowledge transferring…



Knowledge transferring!



Two ways:

1. Logits matching:
Caruana et al.

add an additional loss
Mean Squared Error
between $\mathbf{z}^{small}$
and $\mathbf{z}^{large}$

2. Soft target matching:

add an additional loss
Cross Entropy Loss
between $\mathbf{q}^{small}$
and $\mathbf{q}^{large}$.

Key Insight

"The relative probabilities of incorrect answers tell us a lot about how the cumbersome model tends to generalize."

"An image of a BMW, for example, may only have a very small chance of being mistaken for a garbage truck, but that mistake is still many times more probable than mistaking it for a carrot."

The old soft target:

$$q_i = \frac{exp(z_i)}{\sum_j exp(z_j)}$$

The incorrect class always have very low probabilities for a confident (good) model such as a large model.

This prevents knowledge transfer since lots of the information rely in **the ratios of very small probabilities in the soft targets.**

Knowledge Distillation

$$q_i = \frac{exp(z_i)}{\sum_j exp(z_j)}$$

$$\Big\downarrow \textbf{ Distillation}$$

$$q_i = \frac{exp(z_i/T)}{\sum_j exp(z_j/T)}$$

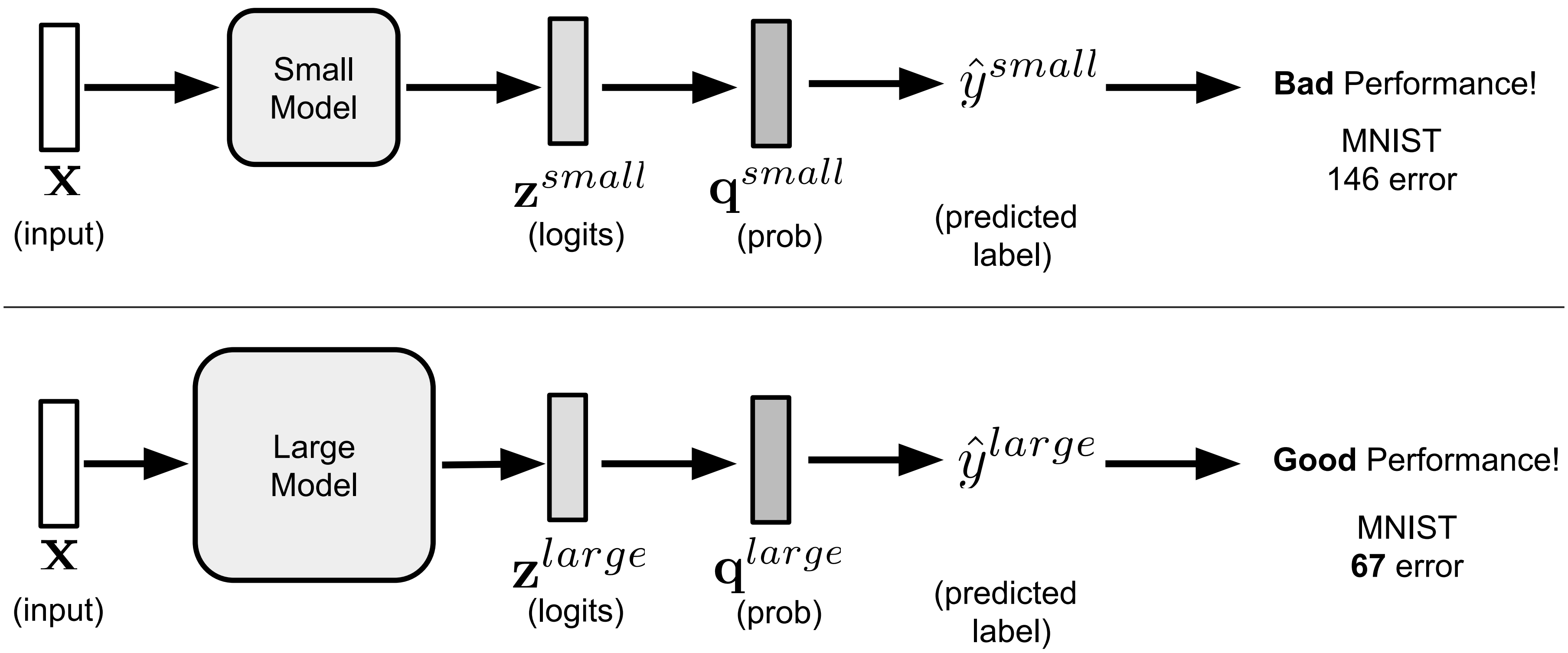
T : temperature

As temperature increases, the incorrect classes have higher probabilities (the entropy increases)

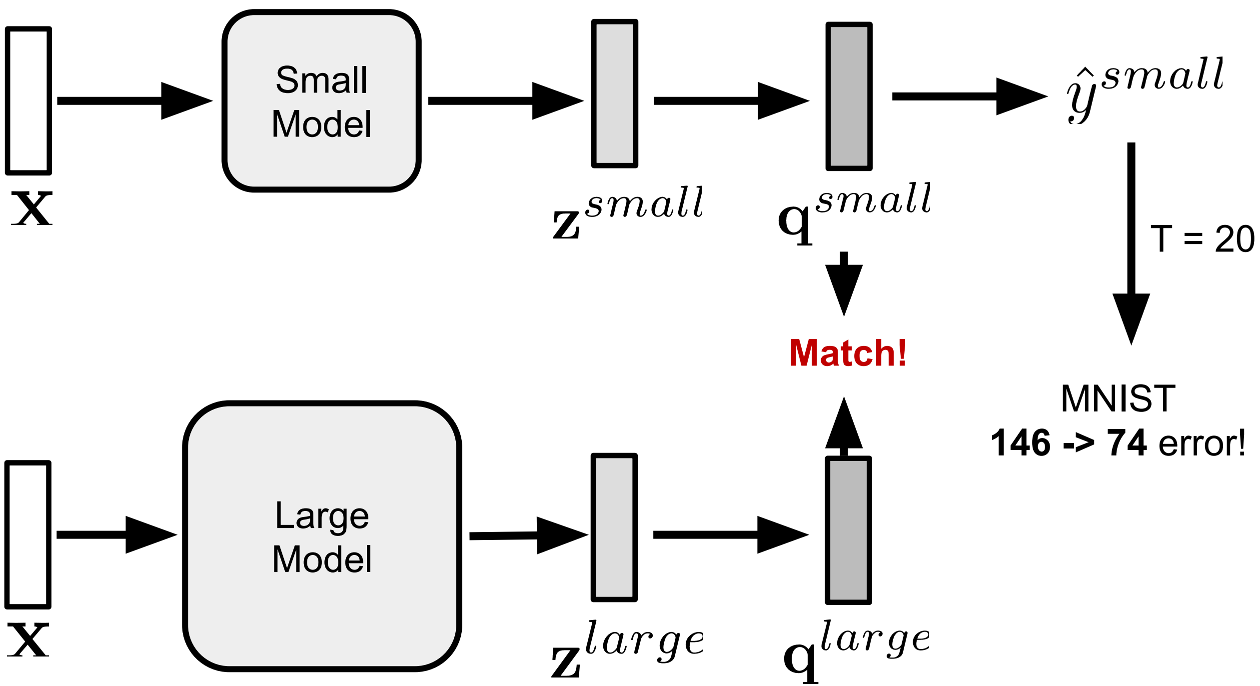
$$H(\mathbf{q}) = -\sum_i^n q_i \log q_i$$

Without any knowledge transferring…

$$q_i = \frac{exp(z_i)}{\sum_j exp(z_j)}$$



Knowledge Distillation!



The training use the distillation probability formula:

$$q_i = \frac{exp(z_i/T)}{\sum_j exp(z_j/T)}$$

Where temperature T is large

System	Test Frame Accuracy	WER
Baseline	58.9%	10.9%
10xEnsemble	61.1%	10.7%
Distilled Single model	60.8%	10.7%

Table 1: Frame classification accuracy and WER showing that the distilled single model performs about as well as the averaged predictions of 10 models that were used to create the soft targets.

System & training set	Train Frame Accuracy	Test Frame Accuracy
Baseline (100% of training set)	63.4%	58.9%
Baseline (3% of training set)	67.3%	44.5%
Soft Targets (3% of training set)	65.4%	57.0%

Table 5: Soft targets allow a new model to generalize well from only 3% of the training set. The soft targets are obtained by training on the full training set.

https://arxiv.org/pdf/1503.02531.pdf

Text Generation with LSTM

Generative Deep Learning

- Generative deep learning methods using RNNs and CNNs have been around for awhile, but have recently been getting a lot of attention
  - 2002: Douglas Eck applied LSTM to music generation - he is now at Google Brain and started a research group called Magenta to use deep learning to create engaging music
  - 2013: Alex Graves applies recurrent mixture density networks to generate human-like handwriting
  - Many more, and one that we'll talk about today

- Many researchers in this field have said that "generating sequential data is the closest computers get to **dreaming**"

Generative RNNs for Text

- RNNs have been successfully used for
  - Music generation
  - Dialogue generation
  - Image generation
  - Speech synthesis
  - Molecule design

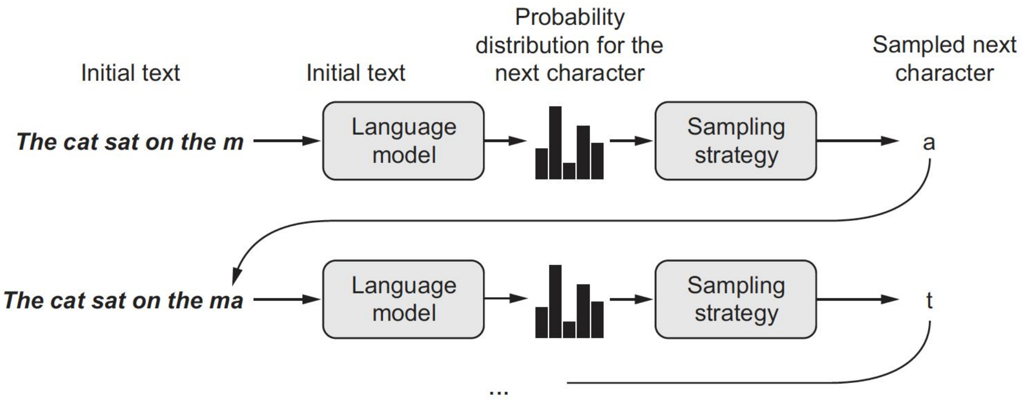
- Main idea for text generation: train a model to predict the next token or next few tokens in a sequence

Generative RNNs for Text

- **Language Model:** any network that can model the probability of the next token given the previous ones
  - Captures the latent space of language - its statistical structure
  - Once it is trained, you can sample from it to generate new sequences

Generative RNNs for Text

- Process
  - 1. Feed it an initial string of text (called conditioning data)
  - 2. Ask the model to generate the next character or word
  - 3. Add the generated output back to the input data
  - 4. Repeat many times



Generative RNNs for Text

- We can choose the next character or word in different ways - some are better than others

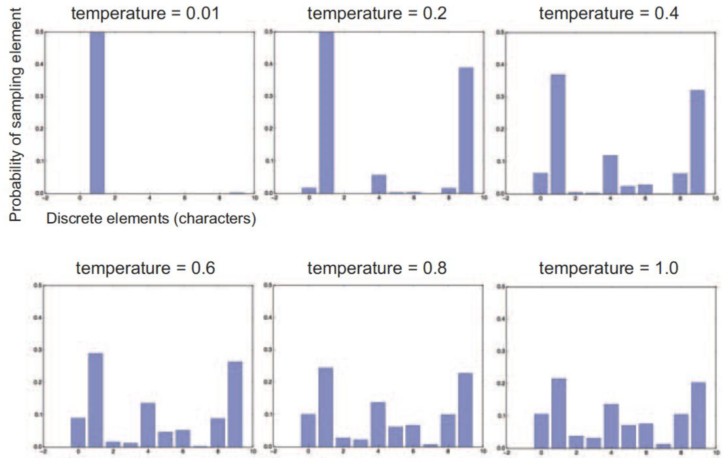
- A naive approach is greedy sampling - always choosing the most likely next character or word
  - This results in repetitive, predictable strings and not very coherent language

Generative RNNs for Text

- Better approach is stochastic sampling
  - Sample next characters or words with specific probability from a probability distribution
  - Allows even unlikely characters or words to be sampled at times, generating more interesting and creative sentences
  - Doesn’t offer a way of controlling the randomness in the sampling process

Generative RNNs for Text

- New parameter to tune: **softmax temperature**
  - Controls the amount of randomness
  - More randomness = similar probability for every character or word and results in more interesting output
  - Less randomness = higher probability for just one or a few characters or words and results in repetitive output
  - Can change the amount of randomness via the temperature value
    - **Higher** temperature = more **randomness**
    - **Lower** temperature = more **deterministic**



Character-level LSTM Generation

- Need a lot of data to train from

- Can choose from many sources, referred to as a **corpus**
  - Wikipedia
  - The Lord of the Rings
  - The writings of Nietzsche translated into English

- Let's see an example with the writings of Nietzsche as our corpus
  - Colab notebook

Character-level LSTM Generation

- Training the language model and sampling from it:
- Given a trained model and a seed text snippet, do the following repeatedly

  - 1. Draw from the model a probability distribution for the next character, given the generated text available so far
  - 2. Reweight the distribution to a certain temperature
  - 3. Sample the next character at random according to the reweighted distribution
  - 4. Add the new character at the end of the available text

Character-level LSTM Generation

- Random seed:
  - “new faculty, and the jubilation reached its climax when kant"
- Output at epoch 20 with temperature = 0.2:

“new faculty, and the jubilation reached its climax when kant and such a man in the same time the spirit of the surely and the such the such as a man is the sunligh and subject the present to the superiority of the special pain the most man and strange the subjection of the special conscience the special and nature and such men the subjection of the special men, the most surely the subjection of the special intellect of the subjection of the same things and”

Character-level LSTM Generation

- Random seed:
  - “new faculty, and the jubilation reached its climax when kant"
- Output at epoch 20 with temperature = 0.5:

“new faculty, and the jubilation reached its climax when kant in the eterned and such man as it’s also become himself the condition of the experience of off the basis the superiory and the special morty of the strength, in the langus, as which the same time life and "even who discless the mankind, with a subject and fact all you have to be the stand and lave no comes a troveration of the man and surely the conscience the superiority, and when one must be w "

Character-level LSTM Generation

- Random seed:
  - “new faculty, and the jubilation reached its climax when kant"
- Output at epoch 20 with temperature = 1.0:

“new faculty, and the jubilation reached its climax when kant, as a periliting of manner to all definites and transpects it it so hicable and ont him artiar resull too such as if ever the proping to makes as cnecience. to been juden, all every could coldiciousnike hother aw passife, the plies like which might thiod was account, indifferent germin, that everythery certain destrution, intellect into the deteriorablen origin of moralian, and a lessority o"

Character-level LSTM Generation

- Random seed:
  - “new faculty, and the jubilation reached its climax when kant"
- Output at epoch 60 (model has now fully converged) with temperature = 0.2:

“cheerfulness, friendliness and kindness of a heart are the sense of the spirit is a man with the sense of the sense of the world of the self-end and self-concerning the subjection of the strengthorixes–the subjection of the subjection of the subjection of the self-concerning the feelings in the superiority in the subjection of the subjection of the spirit isn’t to be a man of the sense of the subjection and said to the strength of the sense of the"

Character-level LSTM Generation

- Random seed:
  - “new faculty, and the jubilation reached its climax when kant"
- Output at epoch 60 with temperature = 0.5:

“cheerfulness, friendliness and kindness of a heart are the part of the soul who have been the art of the philosophers, and which the one won’t say, which is it the higher the and with religion of the frences. the life of the spirit among the most continuess of the strengther of the sense the conscience of men of precisely before enough presumption, and can mankind, and something the conceptions, the subjection of the sense and suffering and the"

Character-level LSTM Generation

- Random seed:
  - “new faculty, and the jubilation reached its climax when kant"
- Output at epoch 60 with temperature = 1.0:

“cheerfulness, friendliness and kindness of a heart are spiritual by the ciuture for the entalled is, he astraged, or errors to our you idstood–and it needs, to think by spars to whole the amvives of the newoatly, prefectly raals! it was name, for example but voludd atu-especity"–or rank onee, or even all "solett increessic of the world and implussional tragedy experience, transf, or insiderar,–must hast if desires of the strubction is be stronges"

Character-level LSTM Generation

- Low temperature results in repetitive and predictable text, but local structure is highly realistic

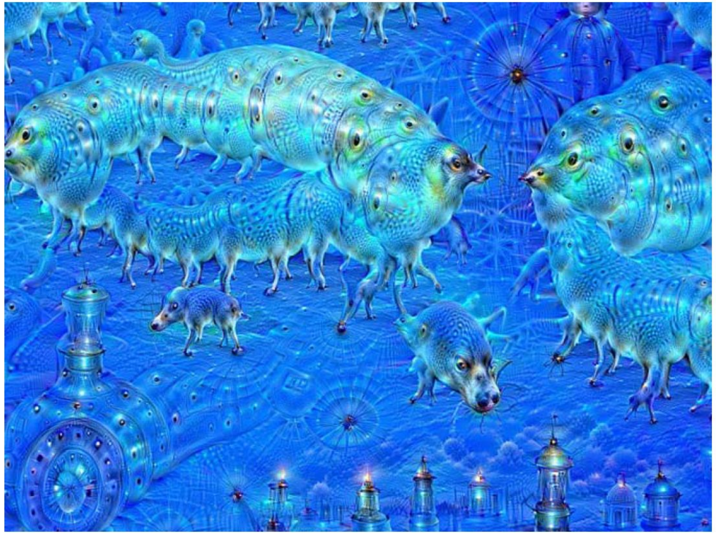
- Higher temperatures result in more interesting, surprising and creative text, sometimes creating new words - but the local structure breaks down and most words are strings of random characters

- Generally, somewhere in the middle (around 0.5) creates the most interesting text - but this depends on the corpus and the human reading the results

DeepDream

DeepDream

- DeepDream is an artistic image-modification technique that uses the representations learned by convolutional neural networks
  - Released by Google in the summer of 2015
  - Trained on ImageNet



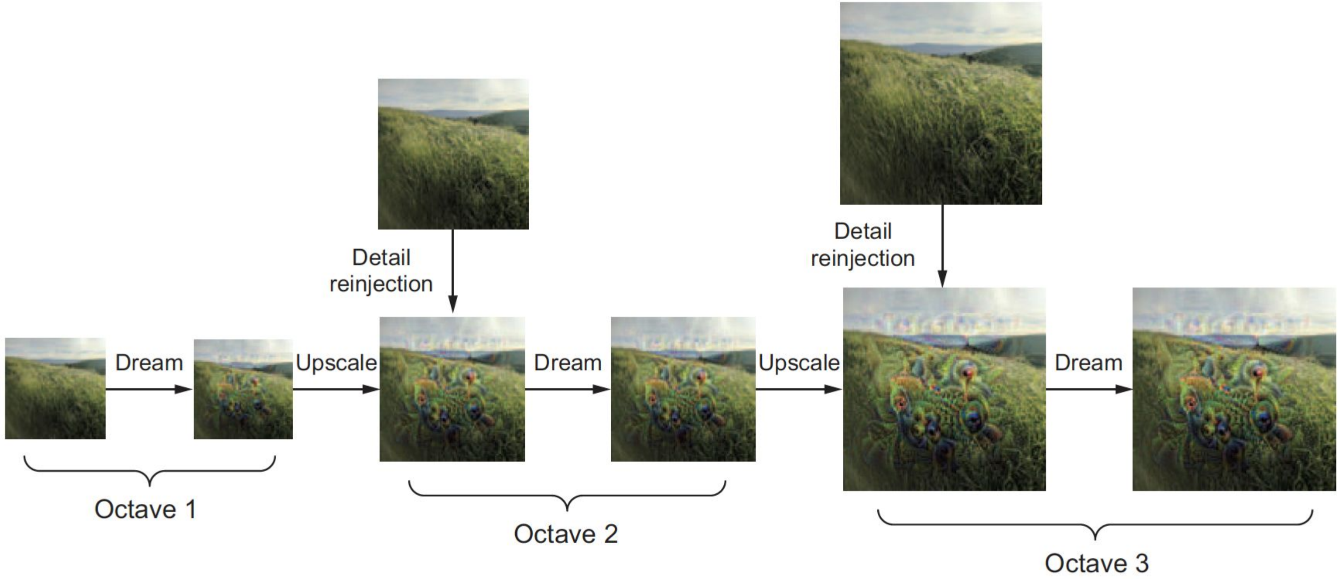
DeepDream

- The DeepDream algorithm is almost identical to the convnet filter-visualization technique introduced in lecture 8, consisting of running a convnet in reverse: doing gradient ascent on the input to the convnet in order to maximize the activation of a specific filter in an upper layer of the convnet

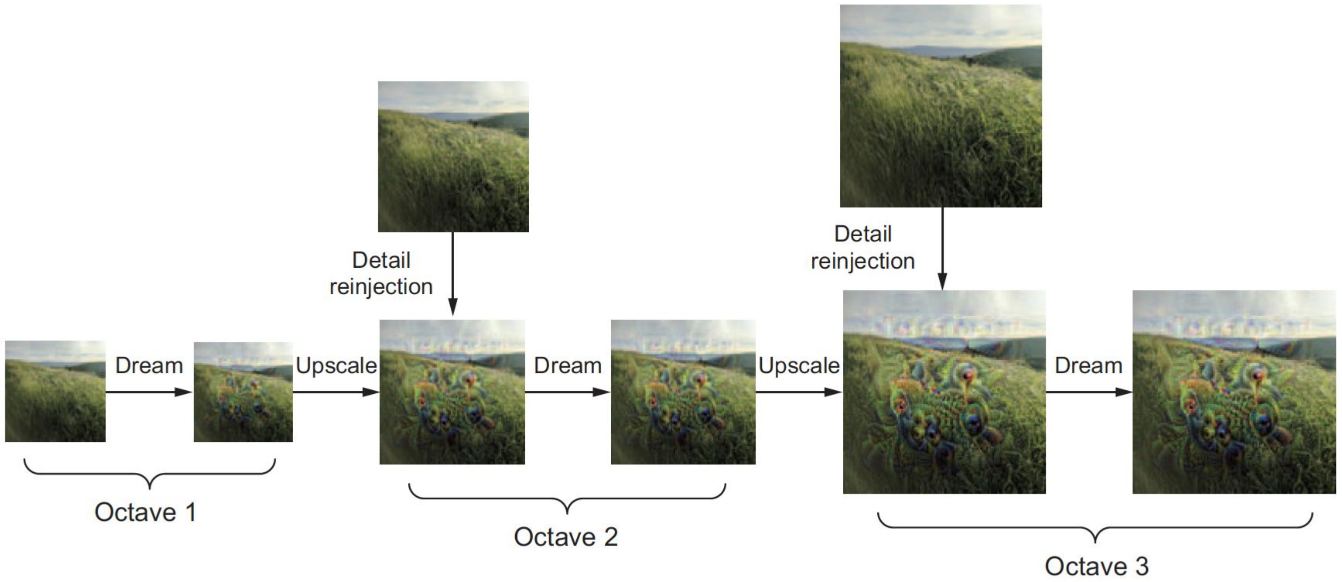
DeepDream

- DeepDream uses this same idea, with a few simple differences
  - You try to maximize the **activation of entire layers** rather than that of a specific filter, thus mixing together visualizations of large numbers of features at once
  - You start not from blank, slightly noisy input, but rather from an existing image—thus the resulting effects latch on to preexisting visual patterns, distorting elements of the image in a somewhat artistic fashion
  - The input images are processed at different scales (called octaves), which improves the quality of the visualizations

- First, define a list of scales (also called **octaves**) at which to process the images
- Each successive scale is larger than the previous one by a factor of 1.4 (it's 40% larger): you start by processing a small image and then increasingly scale it up
- For each successive scale, from the smallest to the largest, you run gradient ascent to maximize the loss you previously defined, at that scale. After each gradient ascent run, you upscale the resulting image by 40%



- To avoid losing a lot of image detail after each successive scale-up (resulting in increasingly blurry or pixelated images), you can use a simple trick: after each scaleup, you'll reinject the lost details back into the image, which is possible because you know what the original image should look like at the larger scale



DeepDream

DeepDream

- Layers that are lower in the network contain more-local, less-abstract representations and lead to dream patterns that look more geometric
- Layers that are higher up lead to more-recognizable visual patterns based on the most common objects found in ImageNet, such as dog eyes, bird feathers, and so on
- [DeepDream video that may make you feel dizzy]()

Neural Style Transfer

Neural Style Transfer

- Another major development in deep-learning-driven image modification
- Introduced by [Leon Gatys et al.] in 2015
- Variations have been introduced, some even as smartphone apps
- Neural style transfer consists of **applying the style** of a reference image to a target image **while conserving the content** of the target image

Neural Style Transfer

- Style
  - textures, colors, and visual patterns in the image, at various spatial scales
- Content
  - higher-level macrostructure of the image

Neural Style Transfer

- Like other neural nets, we need to define and minimize a loss function
  - We want to conserve the content of the original image, while adopting the style of the reference image
  - If we can mathematically define content and style, our loss function would be:

Loss = distance(style(ref_image) - style(generated_image)) +

distance(content(original_image) - content(generated_image))

Neural Style Transfer

- Deep CNNs offer a way to define this loss function mathematically
- Recall:
  - Activations from earlier layers in a network contain local information about the image
  - Activations from higher layers contain increasingly global, abstract information

Neural Style Transfer

- Content loss
  - The content of an image is more global and abstract and should be captured by the representations of later layers
  - Loss is the L2 norm between the activations of an upper layer in a pretrained convnet, computed over the target image, and the activations of the same layer computed over the generated image
  - Ensures the generated image will look similar to the original target image

Neural Style Transfer

- Style loss
  - Uses multiple layers of the CNN
  - Try to capture the appearance of the style reference image at all spatial scales extracted by the convnet, not just a single scale
  - Use the Gram matrix of a layer’s activations: the inner product of the feature maps of a given layer
  - This inner product can be understood as representing a map of the correlations between the layer’s features

Neural Style Transfer

- Style loss
  - These feature correlations capture the statistics of the patterns of a particular spatial scale, which empirically correspond to the appearance of the textures found at this scale
  - Aims to preserve similar internal correlations within the activations of different layers, across the style-reference image and the generated image
  - Guarantees that the textures found at different spatial scales look similar across the style-reference image and the generated image

Neural Style Transfer

- Summary
  - Preserve content by maintaining similar high-level layer activations between the target content image and the generated image. The convnet should “see” both the target image and the generated image as containing the same things

  - Preserve style by maintaining similar correlations within activations for both low-level layers and high-level layers. Feature correlations capture textures: the generated image and the style-reference image should share the same textures at different spatial scales

Neural Style Transfer

[Awesome blog post]()

Content C Style S Generated Image G

Picasso Dancer

Neural Style Transfer

[Image Style Transfer Using CNNs]()

“ A Neural Algorithm of Artistic Style that can separate and recombine the image content and style of natural images. The algorithm allows us to produce new images of high perceptual quality that combine the content of an arbitrary photograph with the appearance of numerous well-known artworks”

Neural Style Transfer

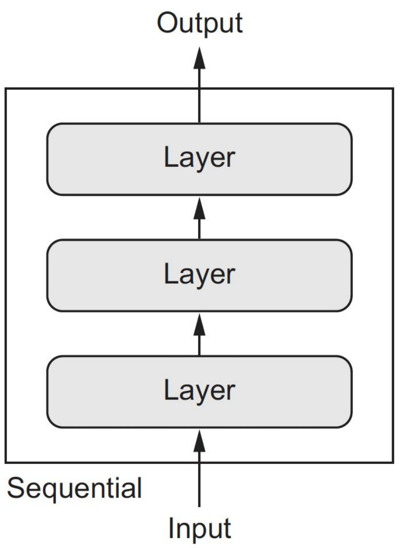
Advanced Architectures

Beyond the Sequential Model

- Throughout the course we have assumed each network has exactly one input and exactly one output, and that it consists of a linear stack of layers

- But what if we have **multiple types** of inputs? Or multiple types of outputs?

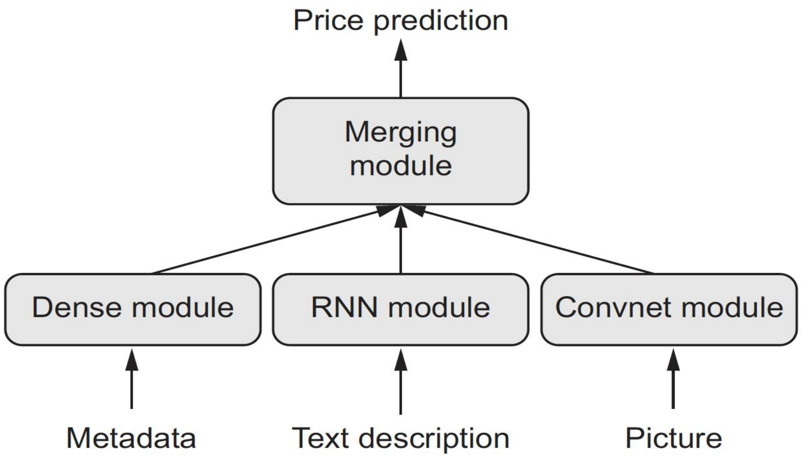
- We can change the network structure - Keras makes this easy to do



Multimodal (Multi-inputs) Model

- Multimodal inputs merge data coming from different input sources, processing each type of data using different kinds of neural layers

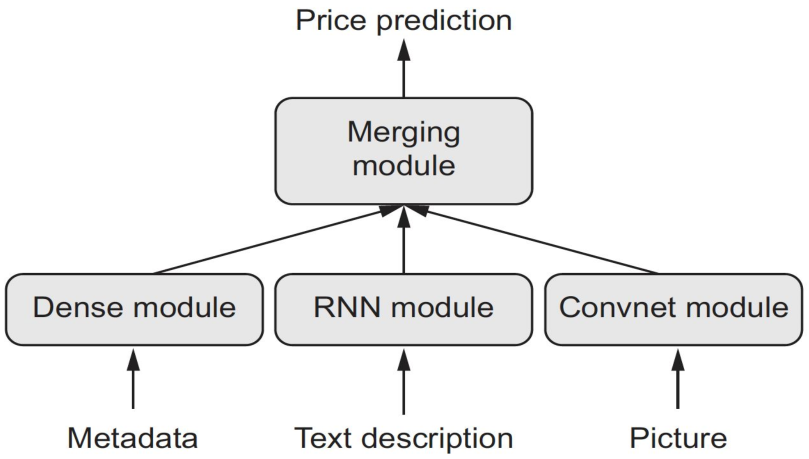
- Example: predict the most likely market price of a second-hand piece of clothing, using the following inputs:
  - User-provided **metadata** (brand, age, etc.)
  - User-provided **text** description
  - **Picture** of the item



Multimodal (Multi-inputs) Model

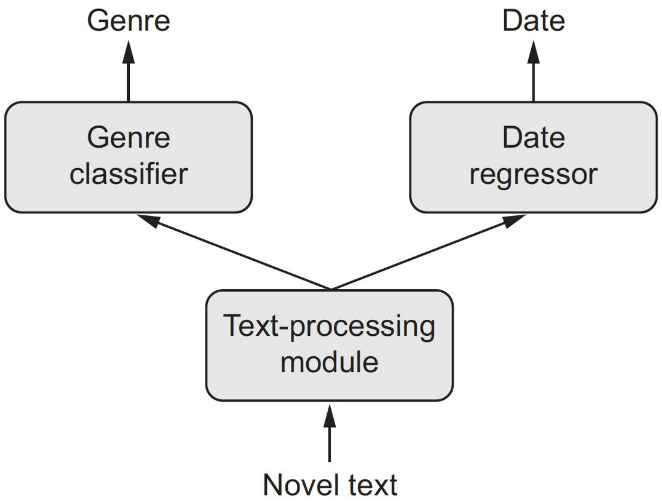
- Suboptimal approach: **train three separate models** and then do a weighted average of their predictions
  - Information may be redundant

- Better approach: **jointly learn** a more accurate model of the data by using a model that can see all available input modalities simultaneously: a model with three input branches



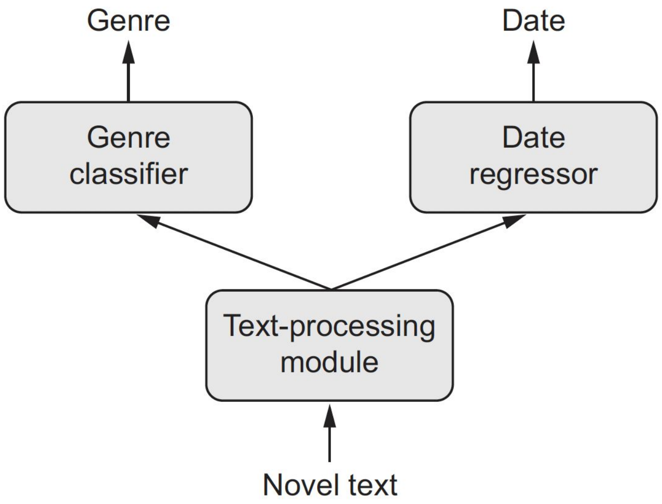
Multi-output (multihead) Model

- Predict multiple target attributes (outputs) of input data
- Example: predict the genre and date of a novel using the novel's text



Multi-output (multihead) Model

- Suboptimal approach: **train two separate models**: one for the genre and one for the date
  - But these attributes aren't statistically independent
- Better approach: **jointly predict** both genre and date at the same time
  - Correlations between date and genre of the novel helps training



Directed Acyclic Graphs (DAGs)...

… not [THOSE DAGs].

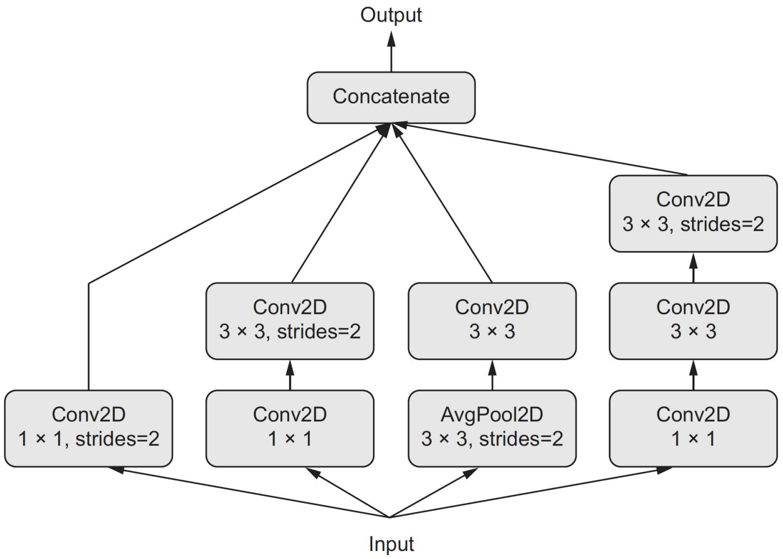
- Nonlinear network architectures
- Examples
  - Inception models
  - ResNet
  - Etc.

Directed Acyclic Graphs (DAGs)...

… not [THOSE DAGs].

- Nonlinear network architectures
- Examples
  - Inception models
  - ResNet
  - Etc.

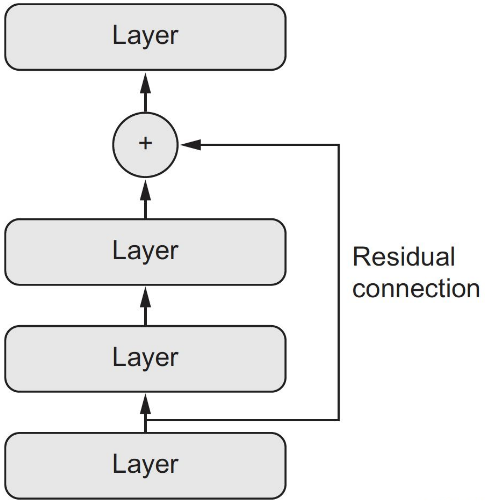
The input is processed by several convolutional branches whose outputs are merged back into a single tensor



Directed Acyclic Graphs (DAGs)...

… not THOSE DAGs.

- Nonlinear network architectures
- Examples
  - Inception models
  - ResNet
  - Etc.
- A residual connection consists of reinjecting previous representations into the downstream flow of data by adding a past output tensor to a later output tensor.



- This helps prevent information loss along the data-processing flow.

The Functional API in Keras

- Directly manipulate tensors
- Use layers as **functions** that take tensors and return tensors

```
from keras.models import Sequential, Model
from keras import layers
from keras import Input
seq_model = Sequential()
seq_model.add(layers.Dense(32, activation='relu', input_shape=(64,)))
seq_model.add(layers.Dense(32, activation='relu'))
seq_model.add(layers.Dense(10, activation='softmax'))


input_tensor = Input(shape=(64,))
x = layers.Dense(32, activation='relu')(input_tensor)
x = layers.Dense(32, activation='relu')(x)
output_tensor = layers.Dense(10, activation='softmax')(x)
model = Model(input_tensor, output_tensor)
model.summary()
```

Sequential model (what we've seen before)

Functional equivalent to the above model

The Model class turns an input tensor and output tensor into a model

The Functional API in Keras

Keras retrieves every layer involved in going from the input tensor to the output tensor and brings them together into a graph-like structure (a Model)

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 64)                0
_________________________________________________________________
dense_4 (Dense)              (None, 32)                2080
_________________________________________________________________
dense_5 (Dense)              (None, 32)                1056
_________________________________________________________________
dense_6 (Dense)              (None, 10)                330
=================================================================
Total params: 3,466
Trainable params: 3,466
Non-trainable params: 0
_________________________________________________________________
```

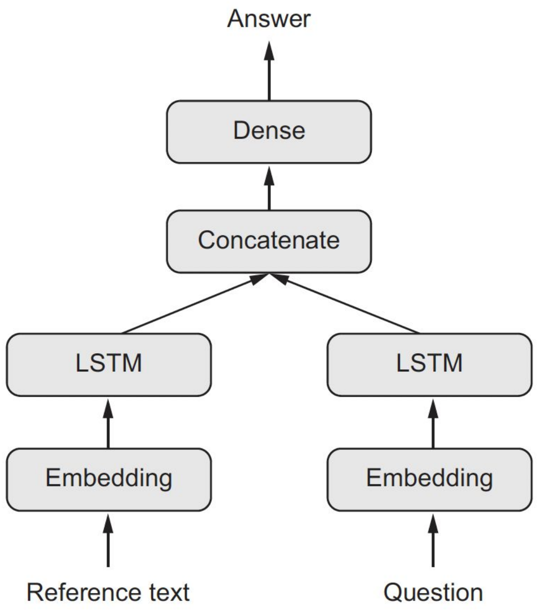
The Functional API in Keras

Everything is the same when you compile, train and evaluate an instance of Model:

```
model.compile(optimizer='rmsprop', loss='categorical_crossentropy')
import numpy as np
x_train = np.random.random((1000, 64))
y_train = np.random.random((1000, 10))
model.fit(x_train, y_train, epochs=10, batch_size=128)
score = model.evaluate(x_train, y_train)
```

Multi-input models

- The functional API can be used to build models that have multiple inputs.
- Typically, such models at some point merge their different input branches using a layer that can combine several tensors: by adding them, concatenating them, and so on.
- This is usually done via a Keras merge operation such as `keras.layers.add`, `keras.layers.concatenate`, etc.

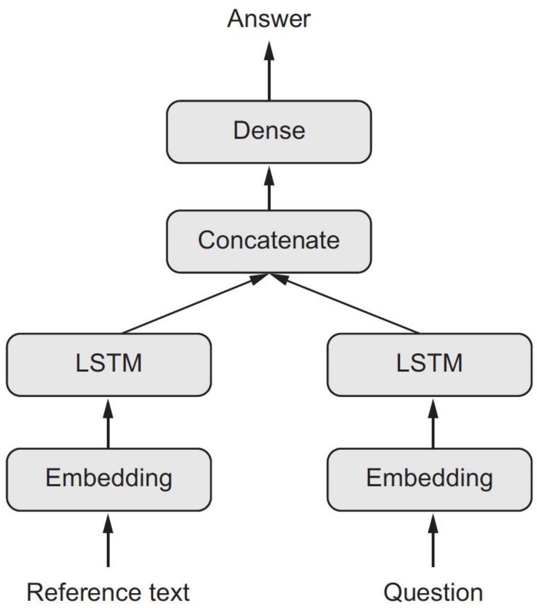


Multi-input models

- Example: **question-answering model**

- A typical question-answering model has two inputs: a natural-language question and a text snippet (such as a news article) providing information to be used for answering the question.

- The model must then produce an answer: in the simplest possible setup, this is a one-word answer obtained via a softmax over some predefined vocabulary



Multi-input models

```
from keras.models import Model
from keras import layers
from keras import Input
text_vocabulary_size     = 10000
question_vocabulary_size = 10000
answer_vocabulary_size   = 500

text_input         = Input(shape=(None,), dtype = 'int32', name = 'text')
embedded_text      = layers.Embedding(64, text_vocabulary_size)(text_input)
encoded_text       = layers.LSTM(32)(embedded_text)

question_input     = Input(shape=(None,),dtype = 'int32', name = 'question')
embedded_question  = layers.Embedding(32, question_vocabulary_size)(question_input)
encoded_question   = layers.LSTM(16)(embedded_question)

concatenated       = layers.concatenate([encoded_text, encoded_question], axis = -1)
answer             = layers.Dense(answer_vocabulary_size, activation = 'softmax')(concatenated)

model = Model([text_input, question_input], answer)
model.compile(optimizer='rmsprop',
loss='categorical_crossentropy',
metrics=['acc'])
```

Branch for encoding the text input

Branch for encoding the question

Multi-input models

How do you train this two-input model?
There are two possible APIs:

1. You can feed the model a list of Numpy arrays as inputs, or
2. you can feed it a dictionary that maps input names to Numpy arrays.

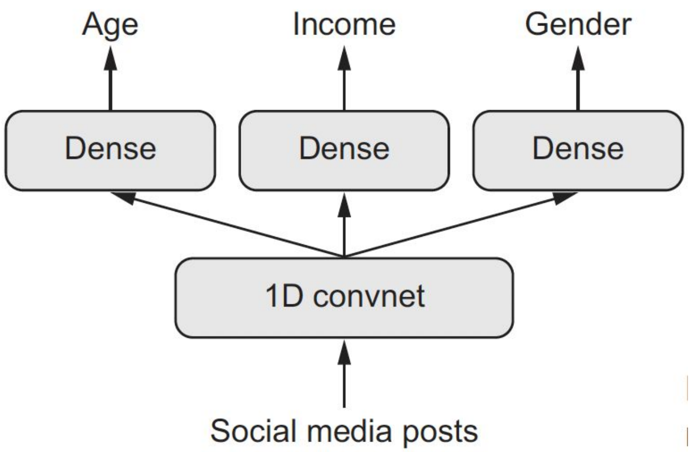
```
import numpy as np
num_samples = 1000
max_length = 100
text = np.random.randint(1, text_vocabulary_size, size = (num_samples, max_length))

question = np.random.randint(1, question_vocabulary_size,
size     = (num_samples, max_length))
answers  = np.random.randint(0, 1, size = (num_samples, answer_vocabulary_size))

model.fit([text, question], answers, epochs = 10, batch_size = 128)
model.fit({'text': text, 'question': question}, answers, epochs=10, batch_size=128)
```

Fitting using a list of inputs

Fitting using a dictionary of inputs

Multi-output Models

- Example: a network that attempts to simultaneously predict different properties of the data, such as a network that takes as input a series of social media posts from a single anonymous person and tries to predict attributes of that person, such as age, gender, and income leve



Multi-output Models

Can specify different functions for different outcomes

```
from keras import layers
from keras import Input
from keras.models import Model
vocabulary_size = 50000
num_income_groups = 10
posts_input = Input(shape=(None,), dtype = 'int32', name = 'posts')
embedded_posts = layers.Embedding(256, vocabulary_size)(posts_input)

x = layers.Conv1D(128, 5, activation='relu')(embedded_posts)
x = layers.MaxPooling1D(5)(x)
x = layers.Conv1D(256, 5, activation='relu')(x)
x = layers.Conv1D(256, 5, activation='relu')(x)
x = layers.MaxPooling1D(5)(x)
x = layers.Conv1D(256, 5, activation='relu')(x)
x = layers.Conv1D(256, 5, activation='relu')(x)
x = layers.GlobalMaxPooling1D()(x)
x = layers.Dense(128, activation='relu')(x)

age_prediction    = layers.Dense(1, name = 'age')(x)
income_prediction = layers.Dense(num_income_groups, activation='softmax', name='income')(x)
gender_prediction = layers.Dense(1, activation='sigmoid', name='gender')(x)
model = Model(posts_input, [age_prediction, income_prediction, gender_prediction])
```

Multi-output Models

- This model requires the ability to specify different loss functions for different heads of the network:
  - Age prediction is a scalar regression task
  - Gender prediction is a binary classification task
- But because gradient descent requires you to minimize a scalar, you must combine these losses into a single value in order to train the model.
- The simplest way to combine different losses is to sum them all.
  - In Keras, you can use either a list or a dictionary of losses in **compile** to specify different objects for different outputs; the resulting loss values are summed into a global loss, which is minimized during training.

```
model.compile(optimizer='rmsprop', loss = ['mse', 'categorical_crossentropy', 'binary_crossentropy'])
```

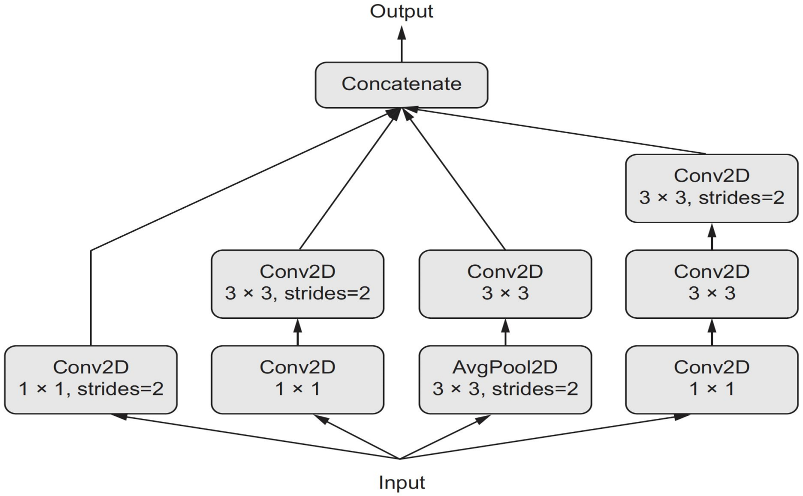
Multi-output Models

- A note on imbalanced loss contributions
  - Imbalances will cause the model representations to be optimized preferentially for the task with the largest individual loss, at the expense of the other tasks
  - To remedy this, you can assign different levels of importance to the loss values in their contribution to the final loss
  - This is particularly useful if the losses' values use different scales
  - Example:
    - MSE takes values around 3-5
    - Cross-entropy loss can be as low as 0.1
    - Here we could assign a weight of 10 for the cross-entropy loss and a weight of 0.25 to the MSE loss

```
model.compile(optimizer = 'rmsprop',
loss=['mse', 'categorical_crossentropy', 'binary_crossentropy'], loss_weights = [0.25, 1., 10.])
```

DAGs of Layers

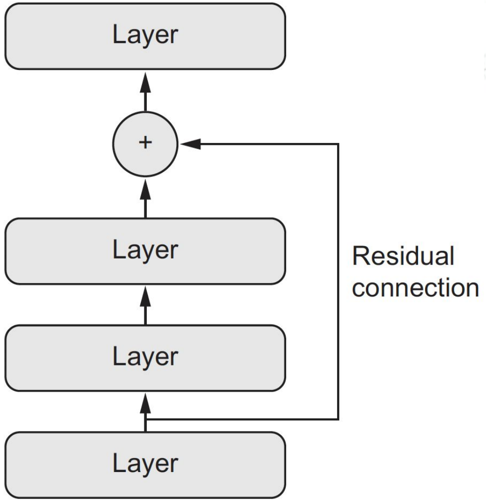
- You can also code complex network architectures in Keras
- Can specify independent branches
- Need to make sure the output of each branch is the same size so you can concatenate them at the end




```python
from keras import layers

branch_a = layers.Conv2D(128, 1, activation='relu', strides=2)(x)

branch_b = layers.Conv2D(128, 1, activation='relu')(x)
branch_b = layers.Conv2D(128, 3, activation='relu', strides=2)(branch_b)

branch_c = layers.AveragePooling2D(3, strides=2)(x)
branch_c = layers.Conv2D(128, 3, activation='relu')(branch_c)

branch_d = layers.Conv2D(128, 1, activation='relu')(x)
branch_d = layers.Conv2D(128, 3, activation='relu')(branch_d)
branch_d = layers.Conv2D(128, 3, activation='relu', strides=2)(branch_d)

output = layers.concatenate([branch_a, branch_b, branch_c, branch_d], axis = -1)
```

Residual Connections

- Residual connections are a common graph-like network component found in many post-2015 network architectures, including Xception
- In general, adding residual connections to any model that has more than 10 layers is likely to be beneficial
- A residual connection consists of making the output of an earlier layer available as input to a later layer, effectively creating a shortcut in a sequential network.
    - Rather than being concatenated to the later activation, the earlier output is summed with the later activation, which assumes that both activations are the same size



Reintroduces x

```
from keras import layers
x = ...
y = layers.Conv2D(128, 3, activation='relu', padding='same')(x)
y = layers.Conv2D(128, 3, activation='relu', padding='same')(y)
y = layers.Conv2D(128, 3, activation='relu', padding='same')(y)
y = layers.add([y, x])
```

Advanced Architecture Patterns

Batch Normalization

- Normalization is a broad category of methods that seek to make different samples seen by a machine learning model more similar to each other, which helps the model learn and generalize well to new data

- We have already done normalization by transforming data to have mean 0 and standard deviation equal to 1

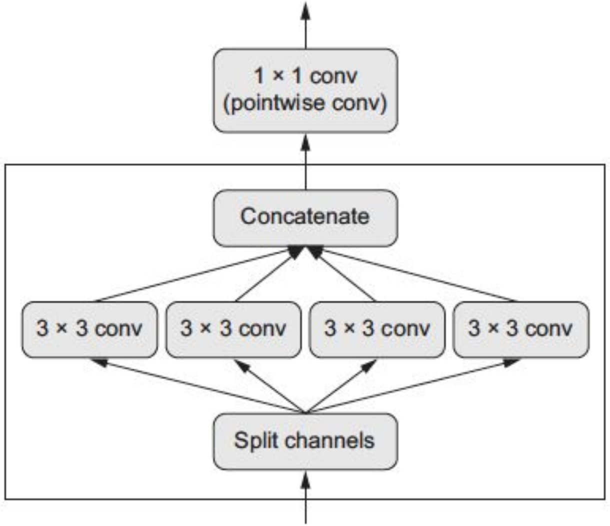
Batch Normalization

- Batch normalization is a type of layer (**BatchNormalization** in Keras)
  - It can adaptively normalize data even as the mean and variance change over time during training.
  - It works by internally maintaining an exponential moving average of the batch-wise mean and variance of the data seen during training.

- The main effect of batch normalization is that it helps with gradient propagation—much like residual connections—and thus allows for deeper networks

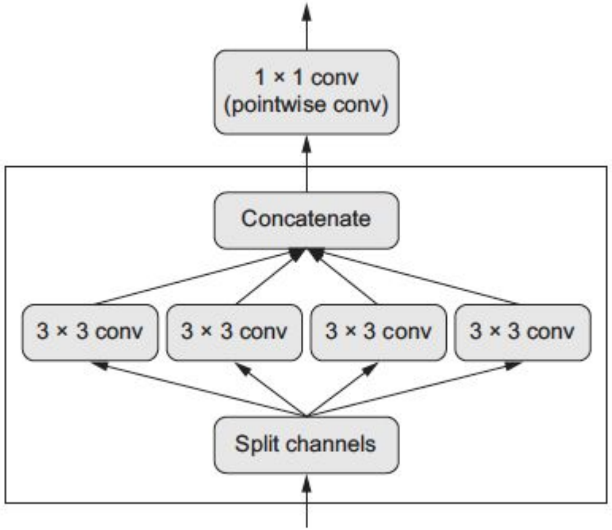
Depthwise Separable Convolution

- A depthwise separable convolution layer performs a spatial convolution on each channel of its input, independently, before mixing output channels via a pointwise convolution
  - Separates the learning of spatial features and the learning of channel-wise features
  - Makes sense if you assume that spatial locations in the input are highly correlated but different channels are fairly independent



Depthwise Separable Convolution

- It requires significantly fewer parameters and involves fewer computations, making it much faster
- Tends to learn better representations using less data, resulting in better-performing models
- Are the basis for the Xception architecture



Depthwise Separable Convolution

Example of an image classification task on a small data set

```
from keras.models import Sequential, Model
from keras import layers
height = 64
width = 64
channels = 3
num_classes = 10

model = Sequential()
model.add(layers.SeparableConv2D(32, 3,activation='relu',input_shape=(height, width, channels,)))
model.add(layers.SeparableConv2D(64, 3, activation='relu'))
model.add(layers.MaxPooling2D(2))

model.add(layers.SeparableConv2D(64, 3, activation='relu'))
model.add(layers.SeparableConv2D(128, 3, activation='relu'))
model.add(layers.MaxPooling2D(2))

model.add(layers.SeparableConv2D(64, 3, activation='relu'))
model.add(layers.SeparableConv2D(128, 3, activation='relu'))
model.add(layers.GlobalAveragePooling2D())

model.add(layers.Dense(32, activation='relu'))
model.add(layers.Dense(num_classes, activation='softmax'))
model.compile(optimizer='rmsprop', loss='categorical_crossentropy')
```

Hyperparameter Optimization

- There is no way of knowing which values are the optimal ones before building and training your model
- Even with experience and intuition, your first pass at the values will be suboptimal
- There are no formal rules to tell you which values for are the best ones for your task
- You can (and we have in this course) tweak the value of each hyperparameter by hand
    - But this is inefficient
- It’s better to let the machine do this, and there is a whole field of research around this
    - Bayesian optimization
    - Genetic algorithms
    - Simple random search
    - Grid search
    - Etc.
- The decision between manual and automated comes down to a balance between understanding your model and computational cost

Hyperparameter Optimization

- The process:
  1. Choose a set of hyperparameters (automatically)
  2. Build the corresponding model
  3. Fit it to your training data and measure the final performance on validation data
  4. Choose the next set of hyperparameters to try (automatically)
  5. Repeat
  6. Eventually, measure performance on your test data

Hyperparameter Optimization

- More tools available each year
  - One option is Hyperopt
    - A Python library for hyperparameter optimization that internally uses trees of Parzen estimators to predict sets of hyperparameters that are likely to work well
  - Another option is Hperas
    - Another library that integrates Hyperopt for use with Keras
  - Weights and Biases
  - SageMaker
  - Comet.ml
  - This great post

Hyperparameter Optimization

- CAUTION!!
  - Can easily overfit to the validation data
  - Can be very computationally expensive

Model Ensembling

Model Ensembling

- Ensembling consists of pooling together the predictions of a set of different models to produce better predictions
- Ensemble models are **as good or better** than one model alone
- Assumes that different good models trained independently are **likely to be good for different reasons** - each model looks at slightly different aspects of the data to make its predictions, getting part of the "truth", but not all of it
- In Keras can combine predictions in different ways (mean, weighted average, etc.)
- SuperLearner
  - In R
  - In Python

SuperLearner